МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Вычислительные машины и контроллеры

Методические указания по выполнению практических работ

Направление подготовки 15.03.04 Автоматизация технологических процессов и производств

Направленность (профиль) Информационно-управляющие системы

Невинномысск 2022

Содержание

Введение

Интернет вещей – активно развивающееся направление, несмотря на отсутствие единых стандартов и разнообразие платформ и самих интернет-вещей, предлагающихся различными компаниями в виде готовых решений со своим дизайном. Существует огромное количество определений этого термина, однако в большинстве случаев под интернетом вещей понимается сбор и обмен данными между различными физическими устройствами (также называемыми умными устройствами, подключёнными устройствами, интернет-вещами и т. д.) на основе определённой сети (здесь иногда добавляют фразу «там, где раньше это было невозможно»). Физическими устройствами могут быть автомобили, здания, камеры, бытовая техника, компьютерная техника и любые другие устройства, оснащённые электроникой, программным обеспечением, сенсорами, двигателями и модулями для подключения к сети интернет, даже города. Данные, получаемые с физических устройств, как правило, собираются в определённом сетевом хранилище – дата-центре, облаке, сервере, репозитории и т. д., которые затем могут быть подвержены анализу методами data mining, machine learning, cloud computing и другими с целью решения тех или иных задач, стоящих перед разработчиками. Для хранения, обработки и визуализации этих данных, а также для предоставления различных сервисов для управления интернет-вещами и анализа данных существуют различные облачные платформы для интернета вещей.

Если взглянуть на кривую компании Гартнер по появляющимся технологиям (emerging technologies), то можно увидеть интернет вещей на пике в 2014–2015 годах, исчезновение IoT в 2016–2017 годах и затем появление на кривой в 2018 году на пути к области спада и разочарования. Однако реальные надёжные технологии (второй версии) появляются только после прохождения через эту впадину избавления от иллюзий. Кроме того, если мы возьмём 2016–2017 годы, на кривой Гартнер область интернета вещей представляет другое направление – платформы для интернета вещей, – активно развивающееся в настоящее время и находящееся на пике этой кривой в 2018 году. В одном из практических заданий второй части данного учебного пособия применяется одна из облачных платформ для интернета вещей и две небольшие вспомогательные платформы со своими сервисами.

С интернетом вещей вплотную я познакомился около полутора лет назад. До этого я, конечно, слышал об этом направлении, или области, – ведь один из студентов, руководителем которого был Восков Л. С., ещё в МИЭМе показывал мне свой проект по умной розетке, которая контролировалась удалённо через вебинтерфейс и зажигала включённую в неё настольную лампу. На слуху были разнообразные умные вещи – умный дом, умный автомобиль, умный холодильник, умная подставка для яиц в холодильнике и т. д. Безусловно, тогда это всё было непросто – датчики и сенсоры, платы и прочее оборудование стоили немало.

Но потом появилась потребность подготовить курс «Экосистемы интернета вещей», и… началось. Прежде всего было не понятно, что делать с практическими занятиями по этому курсу – что давать студентам? «Железок» никаких не было и в помине, в то же время очень хотелось, чтобы студенты работали на реальном оборудовании. Но на каком? Как построить практику? Времени оставалось всё меньше, и тогда я решил попробовать купить комплект с Raspberry Pi 3 – тогда, в 2017 году, несмотря на то что операционная система Android Things была ещё без официального релиза первой версии, по ней были довольно интересные примеры и проекты на официальном сайте разработчиков под Android Things, и не только там. Однако, начав разбираться с системой, я понял, что начинать надо не с этого, а с самых основ, с физики, схемотехники, которых у студентов-программистов, для которых предназначался курс, конечно же, не было. Поэтому вторым шагом была покупка комплекта с платой Arduino Uno – такого, в котором было бы максимально возможное количество «железок». Этот комплект заставил вспомнить основы физики, научиться терпению при определении ошибки в проекте – в схеме она, или в компонентах, или в коде, или в методичке, или где? – и многому другому, в том числе пришлось пропустить через себя методичку с огромным количеством ошибок на ломаном английском языке, которая была на сайте магазина, продававшего комплект. Так появилась первая часть этой книги. Тогда же, когда в моём распоряжении оказались Raspberry Pi 3, HDMI-мини-монитор и Android Things, на книжной полке внезапно появилась и книга Android-разработчика Франческо Эззолы «Android Things Projects», которую пришлось печатать на заказ – недешёвое удовольствие. И, несмотря на устаревший на 1 год код, английский язык и некоторые ошибки в заданиях, или просто иногда пропуск некоторых важных деталей кода в книге, в этом году выросла вторая часть этой книги, посвящённая нескольким проектам под Android Things на плате Raspberry Pi 3. Они приведены к реалиям существующей в настоящее время 1-й версии данной операционной системы и переработаны с целью добавления пользовательского интерфейса, которого в оригинале в некоторых заданиях просто нет. Осваивая книгу и пропуская задания через себя, я понимал, что вот оно – как это просто сделать, оказывается: контроль и полив растения удалённо через смартфон, контроль за удалённой квартирой с помощью ИК-датчика и камеры, пока ты находишься в отпуске, или – получение всей информации о погоде за окном на твоём мониторе в комнате в виде приложения Android Things, включая прогноз погоды.

Данное учебное пособие предлагает повторить этот путь. Первая часть этой книги предназначена для того, чтобы читатели (студенты) освоили основы схемотехники. Да-да, именно в этом и заключается предназначение комплекта модулей, датчиков, проводов, экранов, двигателей, джойстиков, индикаторов, карт и ключей, резисторов и светодиодов с кнопками, вместе с макетной платой и платой Arduino Uno – для того чтобы читатели освоили прежде всего схемы подключения, схемотехническую базу и вспомнили основы физики в части тока и напряжения, а также номиналы резисторов, как их посчитать, – на реальных устройствах, которые могут выйти из строя или работать неправильно при неправильном подключении. Задача же второй части данного учебного

пособия – перейти от схемотехники к реальным проектам для интернета вещей, с их типовой архитектурой, передачей данных от сенсоров к плате, от платы в облако, из облака в мобильное приложение-компаньон, позволяющее осуществить управление платой удалённо, или визуализировать и проанализировать полученные данные, и главное – воплотить в жизнь реальную архитектуру приложений для интернета вещей.

Практические эксперименты первой части учебного пособия построены по определённой схеме. Сначала перечисляются компоненты, необходимые для выполнения задания. Затем приводится описание задания – та часть, которую все обычно пропускают (особенно студенты). После этого дается сначала принципиальная электрическая схема соединения компонентов, а затем – более интересная цветная схема с макетной платой, на которую обычно и ориентируется большинство читателей (она красивее). Затем идёт код программы (иногда их несколько), который необходимо скопировать в Arduino IDE и запустить. Для того чтобы развлечь читателя, комментарии в исправленном коде оставлены без изменений, как они есть в оригинале [9], – на ломаном английском.

Эксперименты из второй части построены немного по-другому – сначала идёт небольшое введение, потом – перечень компонентов и их свойства и изображения. После этого приводится схема подключения компонентов, причём только цветная, с макетной платой. А затем идут этапы выполнения задания, в которых присутствует как исходный код для копирования в Android Studio, так и пояснения к этому коду, а также, при необходимости, скриншоты различных ресурсов и платформ интернета вещей, используемых для выполнения задания.

Отдельно хочется выразить слова благодарности студентам 3-го курса 2017–2018 и 2018–2019 учебных годов образовательной программы «Программная инженерия» департамента программной инженерии факультета компьютерных наук Национального исследовательского университета «Высшая школа экономики», на которых проходила апробация всех заданий из этого пособия. Они не только находили и исправляли ошибки и неточности в некоторых заданиях, но и предлагали разумные вещи для улучшения методических указаний, послуживших основой для этой книги, и даже предоставили материал для научной статьи. И конечно, нельзя не сказать эти слова руководителю упомянутого департамента – Авдошину С. М., который в своё время ошарашил меня курсом «Экосистемы интернета вещей», без которого не было бы никакого учебного пособия.

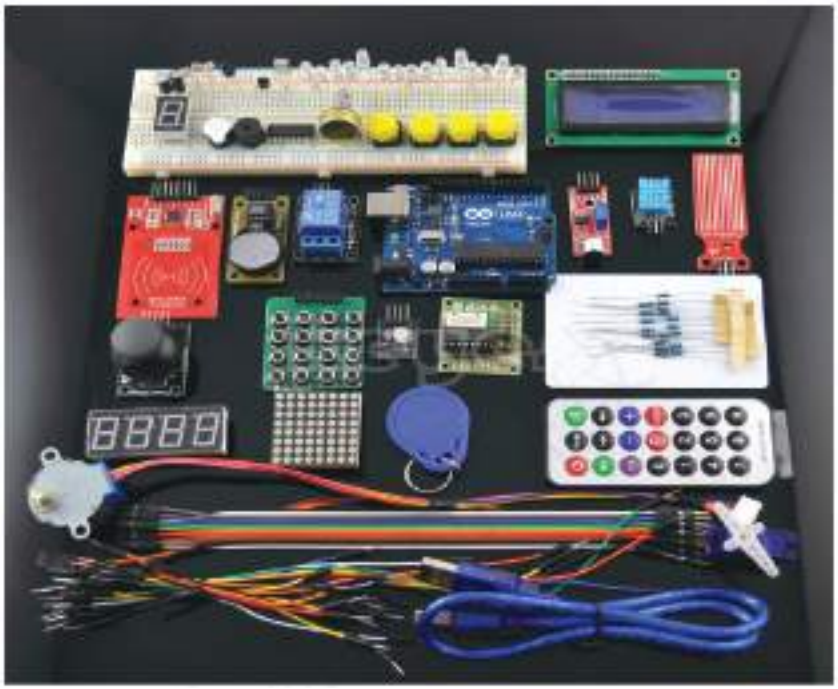
Для выполнения заданий из этого учебного пособия необходимо приобрести, как минимум, 2 комплекта: [1] и [21]. Также отдельно могут понадобиться дополнительные модули и датчики для выполнения заданий из 2-й части учебного пособия, а именно: модуль с Raspberry Pi камерой v 2.1, PIR сенсор и датчик давления, влажности и температуры BME280.

Данная книга выложена в электронном виде на сайте [22], поэтому код из любого рассмотренного практического задания можно скопировать непосредственно из электронной версии.

При подготовке этого учебного пособия не только ни одна плата, но и ни один светодиод, модуль или датчик не пострадал.

Часть 1

Практика на базе комплекта интернет-вещей Arduino Uno R3 Starter Learning Kit с RFID-модулем



Основано на 34 практических заданиях на английском языке [9]

Введение

Arduino – наиболее популярная платформа для разработки как простых, так и достаточно сложных проектов для интернета вещей. Популярность этой платформы обусловлена не только её низкой ценой, но и огромным количеством обучающих материалов, примеров проектов в сети, в том числе и на таких ресурсах, как YouTube, различных форумов разработчиков, а также хорошим официальным сайтом [2, 3], на котором Arduino Team постоянно предлагает ознакомиться с новыми проектами на базе одной из плат Arduino. Существует множество разновидностей плат Arduino, например: Uno, Leonardo, 101, Mega, Zero, Ethernet, Gemma, MKR FOX 1200 и т. д. Все платы на официальном сайте разделены на категории: начальный уровень (для обучения, к нему относится Uno), платы с расширенными функциями, платы для интернета вещей, платы для носимых устройств (в основном в тандеме с Lillypad, например для умной одежды). Несмотря на простоту среды Arduino IDE и её недостатки, в ней можно разрабатывать интересные и сложные проекты; кроме того, существует официальный онлайн-аналог среды и многочисленные библиотеки, способные нарастить её небольшой функционал. Именно поэтому для данного учебного пособия выбрана эта платформа, а конкретно Arduino Uno версии R3, и наиболее богатый и разнообразный в отношении количества различных модулей, сенсоров, датчиков и других компонентов комплект с этой платой – комплект интернет-вещей для начинающих на базе Arduino Uno версии R3 (Starter Learning Kit) с RFID-модулем [1].

Arduino Uno Rev. 3 – лучшая плата для начинающих. Согласно официальному сайту, эта плата является наиболее используемой и обладает наибольшим количеством документации из всех плат семейства Arduino. Arduino Uno – это плата на основе 8-битного микроконтроллера ATmega328P, см. рис. 1 – самая большая чёрная деталь на плате. Таким образом, по сути, плата Arduino Uno является платой расширения, или платой разработчика (developer board), сердце которой – ATmega328P.

Рис. 1 ❖ Лицевая сторона платы Arduino Uno

На плате есть и другой микроконтроллер – ATmega16U2, служащий для связи микроконтроллера ATmega328P и USB порта платы (на рис. 1 – чёрный квадрат слева от TX и RX). У Arduino Uno есть 14 цифровых выходов (пинов) ввода/вывода, обозначенных на плате цифрами (из них 6 ШИМ (PWM, Pulse-Width Modulation) выходных пинов широтно-импульсной модуляции, обозначенных символом ~), 6 аналоговых входов – A0-A6 (с разрешением в 10 бит, т. е. 1024 различных значения), кварцевый кристалл-резонатор на 16 Мгц, выход USB-Bf (на рис. 1 – слева вверху под кнопкой перезагрузки), выход для подключения питания от адаптера питания (7–12 В) или батарейки на 9 В (обычно – в виде прямоугольного параллелепипеда; выход находится слева внизу на рис. 1), ICSP-разъём (In Circuit Serial Programming, программирование по последовательному протоколу чипа, уже подключённого в некоторую схему, или просто – программирование контроллера внутри схемы; на рис. 1 – посередине правого края) и кнопка перезагрузки (слева вверху, см. рис. 1). Кроме этого, плата располагает тремя пинами земли (GND), одним пином на 5 В, одним – на 3.3 В, Vin-пином для подключения внешнего источника питания или для получения напряжения, если плата подключена к внешнему адаптеру питания через разъём питания (через USB-соединение питание ограничивается 5 В), IOREF-пином (Input Output Reference – информация о напряжении микроконтроллера) и встроенным светодиодом L (или 13).

Микроконтроллер ATmega328 на Arduino Uno поставляется уже с загрузчиком (bootloader), что позволяет загружать код без использования внешнего программатора. При желании можно обойти загрузчик и запрограммировать микроконтроллер через ICSP. ATmega328 обладает 32 Кб встроенной памяти (0.5 Кб из которых отведено загрузчику). В дополнение ко всему некоторые пины платы имеют несколько функций, например пины 2 и 3 могут использоваться для вызова внешних прерываний при некоторых событиях, например при событии смены высокого сигнала на пине на низкий (falling edge). Мы не будем углубляться в мельчайшие подробности характеристик платы – приведённой информации вполне достаточно для знакомства с платой и выполнения 34 практических заданий части 1 этого учебного пособия. Также здесь не будет приведена распиновка платы, так как, по сравнению с платой Raspberry Pi 3, с которой мы познакомимся в части 2 этого учебного пособия, для Arduino Uno делать распиновку нет смысла – все пины уже подписаны на плате, см. рис. 1.

Далее рассмотрим, что же входит, помимо самой платы Arduino Uno R3, в состав упомянутого комплекта интернет-вещей для начинающих на базе Arduino Uno версии R3 (Starter Learning Kit) с RFID-модулем. Этот комплект был выбран также и потому, что позволяет сделать очень много практических заданий на основе различных компонентов и устройств, которые в него входят: конечно же, число возможных проектов на его основе далеко не ограничивается 34 экспериментами, приведёнными в этой части учебного пособия, а ограничивается лишь фантазией разработчика. Практические задания предусмотрены для

всех компонентов этого комплекта, чтобы познакомить читателя со всеми возможными составными частями набора и их функциями. В следующем разделе приведены реальные фотографии компонентов комплекта, как они выглядят на самом деле, а не схематичные изображения или фотографии из интернета.

Состав комплекта

Комплект интернет-вещей Arduino Uno Starter Learning Kit с RFID-модулем [1] состоит из следующих компонентов, показанных на рис. 2–38.

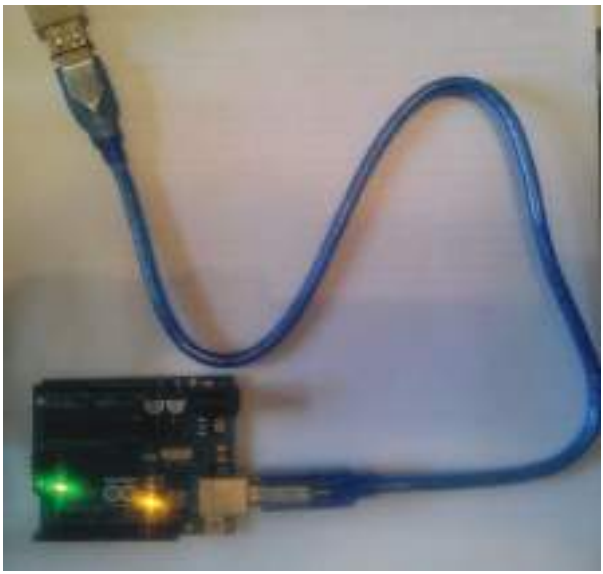


Рис. 2 ❖ Плата Arduino Uno R3 + USB-кабель (Am-Bm)

Рис. 3 ❖ Макетная плата (breadboard)

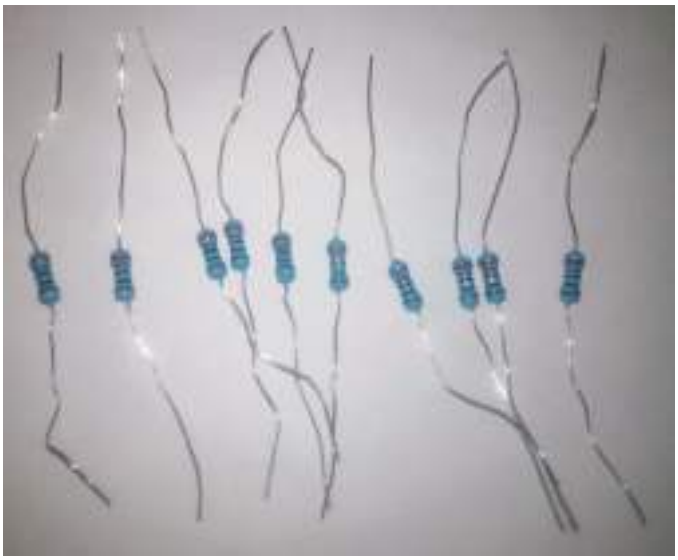


Рис. 4 ❖ Резисторы на 220 Ом, 10 шт.

Рис. 5 ❖ Резисторы на 1 кОм, 10 шт.

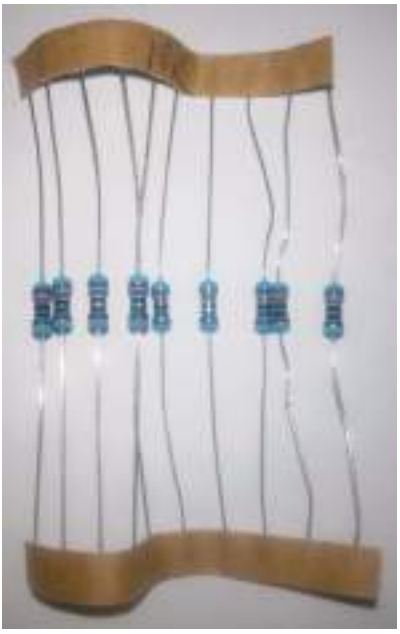


Рис. 6 ❖ Резисторы на 10 кОм, 10 шт.

Рис. 7 ❖ Светодиоды, 15 шт. (5 синих, 5 жёлтых, 5 красных)

Рис. 8 ❖ Кнопки, 4 шт.

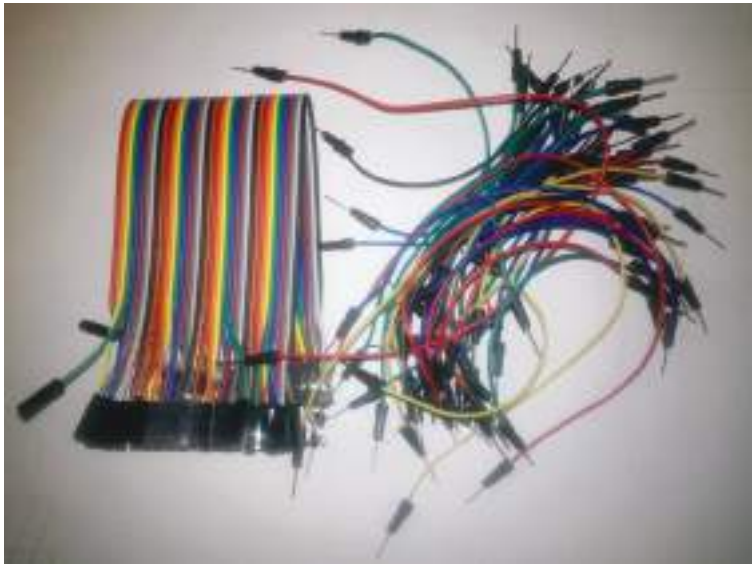


Рис. 9 ❖ Соединительные провода: m-m и f-f

Рис. 10 ❖ Шнур питания от батарейки на 9 В для Arduino Uno

Рис. 11 ❖ Динамик-пищалка, 2 шт.

Рис. 12 ❖ Датчик наклона (tilt switch), 2 шт.

Рис. 13 ❖ Потенциометр (10 КОм)

Рис. 14 ❖ Фоторезистор, 2 шт.

Рис. 15 ❖ Инфракрасный датчик огня

Рис. 16 ❖ Температурный сенсор LM35

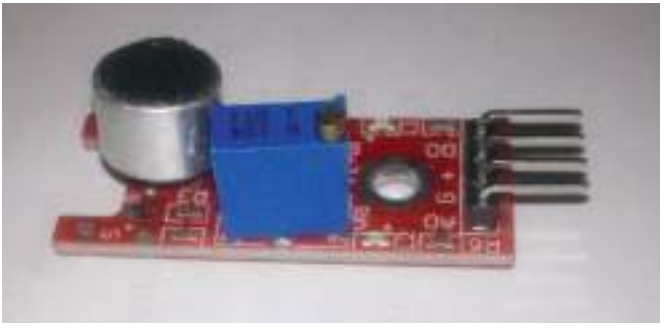


Рис. 17 ❖ Датчик уровня шума

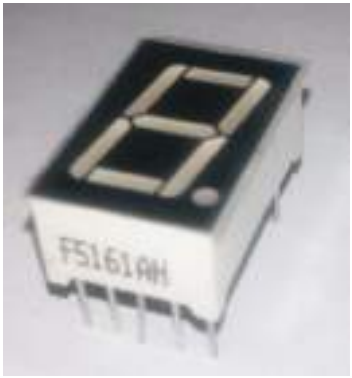


Рис. 18 ❖ Одноразрядный цифровой светодиодный индикатор

Рис. 19 ❖ Четырёхразрядный цифровой светодиодный индикатор

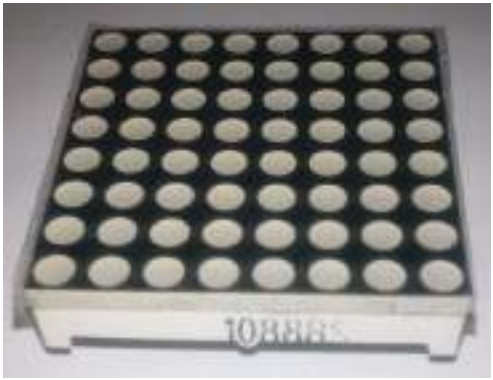


Рис. 20 ❖ Светодиодная матрица 8×8

Рис. 21 ❖ Трёхцветный светодиод с общим катодом (на модуле)

Рис. 22 ❖ Модуль 74HC595

Рис. 23 ❖ Кнопочный модуль 4×4

Рис. 24 ❖ Часы реального времени RTC DS1307

Рис. 25 ❖ Релейный модуль

Рис. 26 ❖ Датчик уровня воды (Water Sensor)

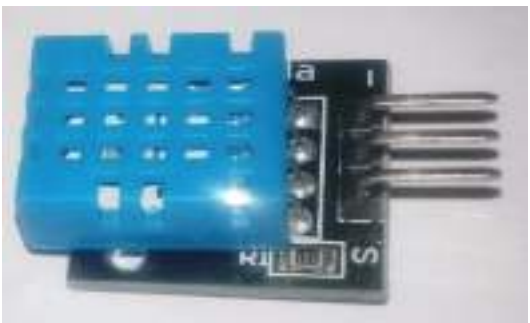


Рис. 27 ❖ Сенсор температуры и влажности DHT11

Рис. 28 ❖ Жидкокристаллический монитор (дисплей) LCD1602A

Рис. 29 ❖ Два штырьковых коннектора по 8 пинов каждый

Рис. 30 ❖ Шаговый двигатель

Рис. 31 ❖ Модуль для шагового двигателя

Рис. 32 ❖ Серводвигатель (сервопривод) с комплектом насадок

Рис. 33 ❖ Игровой джойстик

Рис. 34 ❖ Инфракрасный пульт дистанционного управления NEC

Рис. 35 ❖ Инфракрасный приёмник

Рис. 36 ❖ RFID-модуль RC522

Рис. 37 ❖ RDIF-карта

Рис. 38 ❖ RFID-ключ

Немного о макетной плате, резисторах и безопасности

Как в этой, так и во второй части книги вам понадобятся знания о том, что такое макетная плата, как включать в схему светодиод и как отличить один резистор от другого. Но сначала – о безопасности.

Главное правило обращения с электричеством, компонентами и модулями гласит: помните, что как вы можете повредить технику, так и она может нанести вам вред! Перед тем как выполнять задания, нужно помнить о простых правилах работы с электронными компонентами и тем более системами на модуле (SoM, System on Module), к которым относится Arduino Uno, платами и прочими электронными изделиями:

- собирать и разбирать/менять схему можно только при выключенном питании (отсоединённом USB-кабеле) – имейте терпение;
- светодиоды и другие чувствительные компоненты подключаются строго согласно схеме – через резисторы;
- не стоит путать питание с землёй, плюс с минусом;
- никакого статического электричества! Если на вас свитер из синтетики или шерсти, если вы любите часто поправлять свои волосы, заземляйте свои руки, перед тем как дотрагиваться до электронных изделий (дотроньтесь до корпуса компьютера, железной ножки стола, батареи и т. д.)!

Макетная плата – удобное средство для соединения электрических компонентов в простые схемы и даже в схемы среднего уровня сложности. Макетные платы бывают разных типов, но в основном выделяют два типа: с разрывом горизонтальных линий земли и питания сверху и снизу посередине и без разрыва. В комплекте вам могут попасться оба типа. Макетная плата без разрыва горизонтальных линий земли и питания сверху и снизу показана на рис. 3. Если разрыва нет, это явно показывается синими и красными линиями: на

рис. 3 линии идут непрерывно, значит, разрыва нет. В случае наличия разрыва красные и синие линии прерываются посередине платы.

Соединения макетной платы без разрыва линий земли и питания показаны на рис. 39. Соединения макетной платы с разрывом этих линий, соответственно, проходят снизу и сверху по горизонтали от краёв только до середины макетной платы. Что же касается вертикальных соединений, у макетных плат обоих рассмотренных типов соединения прерываются 3 раза по вертикали: между зелёными разъёмами верхнего и нижнего рядов на рис. 39 нет соединения, так же, как и между зелёными и красными рядами.

Все принципиальные схемы и схемы с макетной платой в этой книге нарисованы с помощью наиболее распространённой и популярной открытой библиотеки + редактора электронных компонентов и схем Fritzing [10].

Рис. 39 ❖ Соединения макетной платы без разрывов верхних и нижних горизонтальных линий питания и земли

Теперь – светодиод. У каждого светодиода есть короткий и длинный выходы (пины) – см. рис. 7. Они представляют, соответственно, катод и анод. На схеме у светодиодов эта особенность выражена следующим способом: более длинный пин (анод) изогнут у основания цветной колбы светодиода, более короткий же (катод) входит в колбу прямо, без изгиба (см., например, рисунок с макетной платой к практическому занятию 3 в этой части учебного пособия). При подключении светодиода надо помнить, что ток по нему может протекать только в одном направлении – от анода к катоду (светодиод – вид диода, который работает только в одном направлении), таким образом, анод (длинный пин) всегда подключается к источнику питания или управляющему сигналу, тогда как катод (короткий пин) обычно подключается к земле через сопротив-

ление, ограничивающее ток, протекающий через светодиод. Светодиод нельзя подключать без сопротивления, иначе он может сгореть. Чем больше сопротивление в схеме со светодиодом (от 220 Ом до 10 кОм), тем меньше яркость свечения светодиода (меньший ток проходит через него). Подробнее о светодиодах можно почитать, например, в источнике [12].

И наконец, резисторы. На схемах резисторы обозначаются следующим образом (слева – в англоязычных источниках, справа – в русскоязычных источниках):

Рис. 40 ❖ Обозначение резисторов на принципиальных схемах [11]

У каждого резистора есть номинал: 220 Ом, 1 кОм и т. д. Резисторы, входящие в комплект с Arduino Uno и в другие комплекты с иными платами, обладают цветовыми насечками, см., например, рис. 4–6. Каждый цвет обозначает цифру, от 0 до 9, и цветовых насечек на резисторе несколько: таким образом можно определить номинал резистора, пользуясь правилами, изображёнными на рис. 41 [11]. Подобные резисторы, несмотря на их размер, всё же встречаются в реальных схемах, используемых в промышленности: например, управляющая плата холодильника Whirlpool собрана с помощью таких резисторов, поскольку холодильник большой и делать миниатюрную плату с применением сверхточных технологий, которые используются при производстве материнской платы для настольного компьютера, не имеет смысла.

Обладая знаниями из таблицы, взятой с сайта [11] и изображённой на рис. 41, вы можете подсчитать номиналы и точность резисторов, входящих в комплект с Arduino Uno и показанных на рис. 4, 5 и 6. На рис. 42 даны примеры резисторов и их номиналов.

Резисторы для схем бывают двух типов (здесь мы для простоты не говорим о специфических типах резисторов – потенциометрах, термисторах, варисторах, фоторезисторах и т. д., хотя некоторые из них встретятся нам в этой части книги): включённые последовательно в электрическую цепь (series resistors) по отношению к пинам платы, и – параллельно, которые, в свою очередь, подразделяются на стягивающие и подтягивающие резисторы (pull-down и pull-up resistors). Значения последовательных резисторов обычно варьируются от 100 до 300 Ом, стягивающих и подтягивающих – от 1 кОм до 10 кОм. Последовательные резисторы подключаются в электрическую цепь, чтобы защитить оборудование и, главное, пины платы от больших значений тока: например, так подключается светодиод через резистор номиналом 220 Ом, чтобы светодиод не перегорел. Каждый пин платы обладает ограниченной способностью быть источником (высокое значение, логическая 1) или приёмником (низкое значение, логический 0) электрического тока через электрическую схему, под-

ключённую к нему. Периферия, которая в схеме потребляет большие значения тока – больше, чем может позволить себе пин платы, даже если это происходит очень короткое время, может повредить пин на плате – именно поэтому в схему последовательно включается ограничивающий ток резистор, например как показано на рис. 43 в схеме со светодиодом [19], где OUT – это пины платы, а Vcc – источник питания.

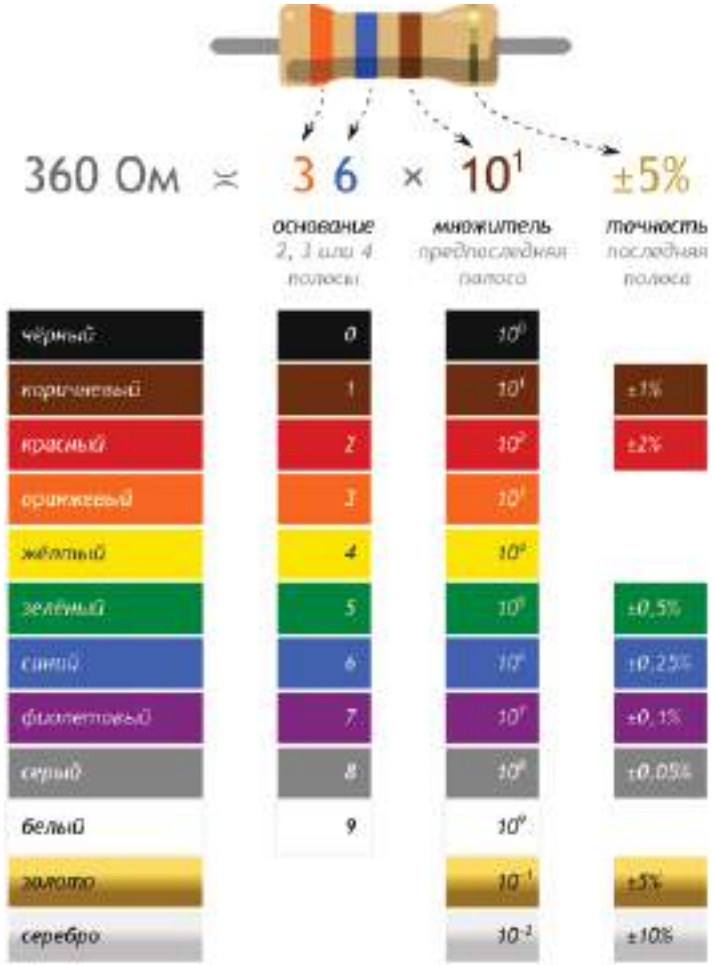


Рис. 41 ❖ Правила подсчёта номинала резистора и его точности [11]

Рис. 42 ❖ Примеры резисторов и их номиналов [11]

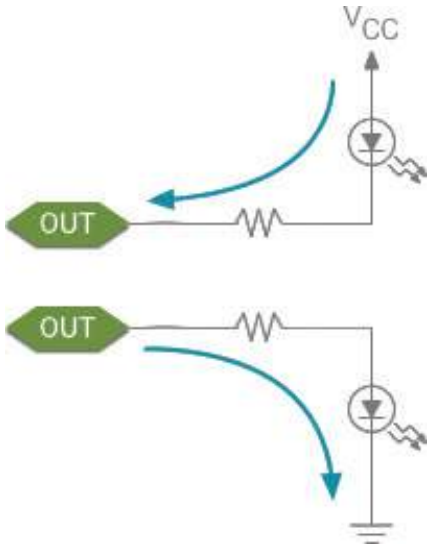


Рис. 43 ❖ Пример подключения последовательных резисторов [19]

Стягивающие и подтягивающие резисторы подключаются в схему параллельно по отношению к пинам и используются для того, чтобы, соответственно, «стягивать» значение напряжения на пине к низкому (обеспечивать стабильный сигнал логического 0) или «подтягивать» значение напряжения на пине к высокому (стабильная логическая 1). Делается это потому, что цифровые входы, не подключённые ни к какой нагрузке, являются «плавающими» (см. рис. 44, левую часть – представьте её без включённых туда резисторов; I/O – пины платы): они подвержены различного рода помехам и искажениям, появляющимся из-за электромагнитных возмущений, которые влияют на значения, читаемые с пинов платы приложениями, и могут способствовать непредсказуемым изменениям этих значений. Стягивающие и подтягивающие резисторы заставляют пины показывать правильные значения 0 или 1, даже если к ним ничего не подключено [19]. В правой части рис. 44 изображён случай с переключателем: если в схеме не будет подтягивающего резистора, значение при открытом переключателе на входном пине IN платы, читающем значения, будет плавающим; если в схеме есть такой резистор, изображённый на рисунке, значение на пине IN будет точно соответствовать логической единице. Для более подробного ознакомления с резисторами рекомендуются источники [11] или [19].

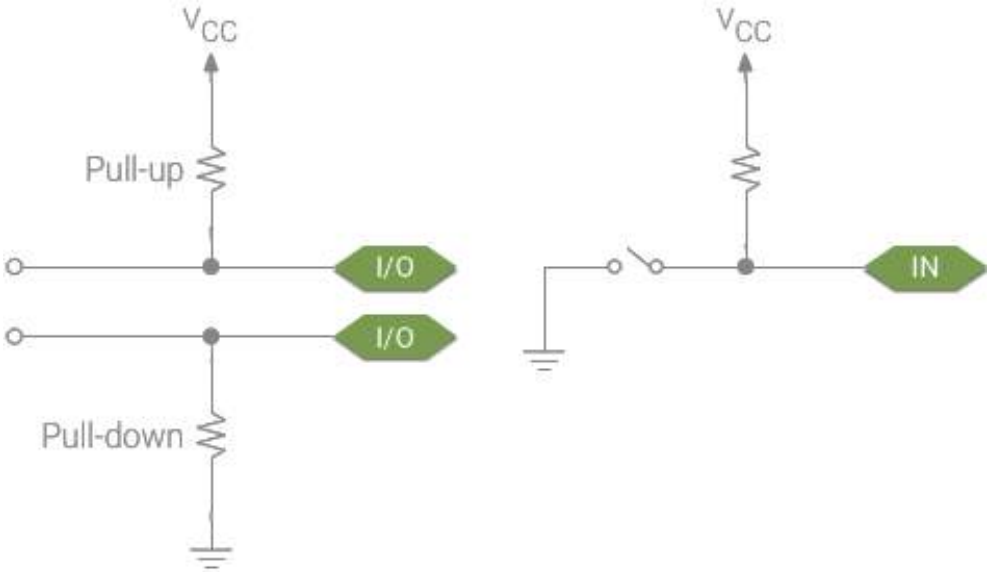


Рис. 44 ❖ Пример подключения стягивающего (pull-down) и подтягивающего (pull-up) резисторов [19]

Ещё один момент, о котором следует упомянуть, – это тот факт, что при подключении различных модулей и сенсоров надо также обращать внимание не только на схему, но и на надписи рядом с выходами (пинами) этих сенсоров. Сенсоры в наборах могут отличаться, хоть эта вероятность и мала, поэтому всегда проверяйте наличие информации/меток рядом с пинами устройств, используемых в практических заданиях, – это первично, а схема вторична. Простой пример – задание 24 из этой части книги, где к конкретным пинам платы подключаются конкретные выходы модуля часов реального времени, обозначения которых можно увидеть на самом модуле – см. рис. 24. Некоторые обозначения: – (минус), GND, G – земля; + (плюс), VCC, VIN, +5V, 3.3V – питание; CLK, SCK – clock (время, частота), DAT, SDA – date/data (дата, данные), RST – reset (сброс настроек), R,G,B – цвета на трёхцветном светодиоде; A0 (аналоговый), D0 (цифровой), SIG, S, VRx,VRy, SW – пин для передачи сигнала (данных).

Теперь, получив базовые знания об основных элементах, использующихся в практических экспериментах, и о технике безопасности, можно приступать к выполнению заданий.

Практическое занятие 1. Hello, world!

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm).

Первое, что нужно сделать, – это выбрать, где вы будете работать: в среде Arduino IDE или с помощью онлайн-системы Arduino Create (Arduino Web Editor) через веб-браузер. В первом случае необходимо скачать Arduino IDE ([2] -> Windows Installer) и установить её, включая установку драйвером для COM- и USB-портов. Во втором случае необходимо зарегистрироваться на сайте ([3] -> sign up).

Далее требуется пройти урок-инструкцию по следующему адресу: [4]. Это нужно для того, чтобы установить драйверы для платы, если они правильно не установились или нет прав администратора на компьютере, а также для того, чтобы правильно настроить среду и выбрать плату Arduino Uno и COM-порт.

После этого можно приступать к занятию. Подсоедините плату Arduino Uno к USB-порту компьютера (если вы ещё этого не сделали). Если драйверы установлены правильно, плата должна определиться, её название появится в панели уведомлений операционной системы. На плате есть встроенный мини-светодиод (miniLED), подключённый к 13-му цифровому порту. В этом занятии мы напишем код, который будет ожидать ввода через консоль буквы R, при её вводе заставлять miniLED 13 загораться на полсекунды, гаснуть на полсекунды и писать в консоль фразу Hello, World! Так как miniLED является встроенным, никаких дополнительных схем создавать не надо. Для того чтобы открыть консоль, надо выбрать в пункте меню **Инструменты** Монитор порта, или нажать комбинацию **Ctrl**+**Shift**+**M** (Arduino IDE), или выбрать пункт меню **Монитор порта** слева (Arduino Web Editor). Когда всё готово, можно скопировать код ниже в среду и загрузить программу на плату с помощью кнопки ->.

В коде используется команда Serial.begin(9600), означающая, что скорость/частота обмена данными платы с компьютером по UBS-соединению составляет 9600 bps (bits per second, бит в секунду). В консоли можно увидеть, что есть и другие частоты, но для выполнения задания в консоли должна быть выставлена такая же частота (справа внизу).

Схема представлена на рис. 45.

Рис. 45 ❖ Схема подключения для практического занятия 1

Код программы

```
int val ;// define a variable val
int ledpin = 13 ;// define the digital interface 13
void setup ()
{
    Serial.begin (9600) ;// set the baud rate to 9600, where the software settings keep
consistent.
    pinMode (ledpin, OUTPUT) ;// set the digital output interface 13 is, Arduino, we use
the I / O port should be carried out like this definition.
}
void loop ()
{
    val = Serial.read () ;// read the PC sends a command to the Arduino or characters, and
the
    instruction or character assigned val
    if (val == 'R') // determine the received command or character is «R».
    {// If you receive a «R» character
        digitalWrite (ledpin, HIGH) ;// lit Digital 13 LED.
        delay (500);
        digitalWrite (ledpin, LOW) ;// Off Digital 13 LED
        delay (500);
        Serial.println ("Hello World!") ;// Displays «Hello World!» String
    }
}
```

Практическое занятие 2. Эксперимент с мигающим светодиодом

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm).

В этом занятии мы всё ещё работаем со встроенным мини-светодиодом на плате (miniLED), подключённым к 13-му цифровому порту. Напишем код, который будет заставлять miniLED 13 загораться на секунду и гаснуть на секунду. Так как miniLED является встроенным, никаких дополнительных схем создавать не надо.

Схема представлена на рис. 46.

Код программы

```
int ledpin = 13 ;// define the digital interface 13
void setup ()
{
        pinMode (ledpin, OUTPUT) ;// set the digital output interface 13 is, Arduino, we
use the I / O port should be carried out like this definition.
}
void loop ()
{
        digitalWrite (ledpin, HIGH) ;// lit Digital 13 LED.
```

```
        delay (1000);
        digitalWrite (ledpin, LOW) ;// Off Digital 13 LED
        delay (1000);
}
```

Рис. 46 ❖ Схема подключения для практического занятия 2

Практическое занятие 3. Эксперимент с контролируемой потенциометром яркостью свечения светодиода через порт PWM

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- потенциометр;
- светодиод;
- резистор на 220 Ом;
- макетная плата;
- соединительные провода.

В этом занятии мы познакомимся с потенциометром и портом PWM. PWM (Pulse Width Modulation) – широтно-импульсная модуляция, или процесс управления мощностью, подводимой к нагрузке, путём изменения отношения периода импульса к длительности импульса при неизменной частоте. На плате Arduino Uno можно подавать значения от 0 до 255 на PWM-пин, что заставит плату выдавать PWM-сигнал в определённые моменты времени, соответствующие поданному входному значению. Другими словами, в терминах напряжения, подавая разные значения от 0 до 255 на PWM-пин, мы заставляем плату менять напряжение от 0 до 5 В. В коде используется функция analogWrite(pin, value), с помощью которой можно менять напряжение, подаваемое на PWM-пин, задавая value от 0 до 255. Эта функция используется для регулировки скорости вращения мотора, яркости светодиода и т. д.

Arduino Uno располагает несколькими PWM-пинами, обозначенными символом ~ рядом с номером пина: 3, 5, 6, 9, 10, 11 (см. рис. 1). С помощью потенци-

ометра, подключённого к одному из аналоговых портов платы (A0–A6), и светодиода, катод (короткий пин) которого подключён через резистор на 220 ОМ к земле (всегда! к минусу) и анод (длинный пин) которого подключён к одному из цифровых пинов (0–13) платы (всегда! к плюсу), мы будем регулировать значение переменной val (крутя ручку потенциометра) и, соответственно, напряжение, которое выдаёт плата из этого цифрового пина.

Схема представлена на рис. 47–48.

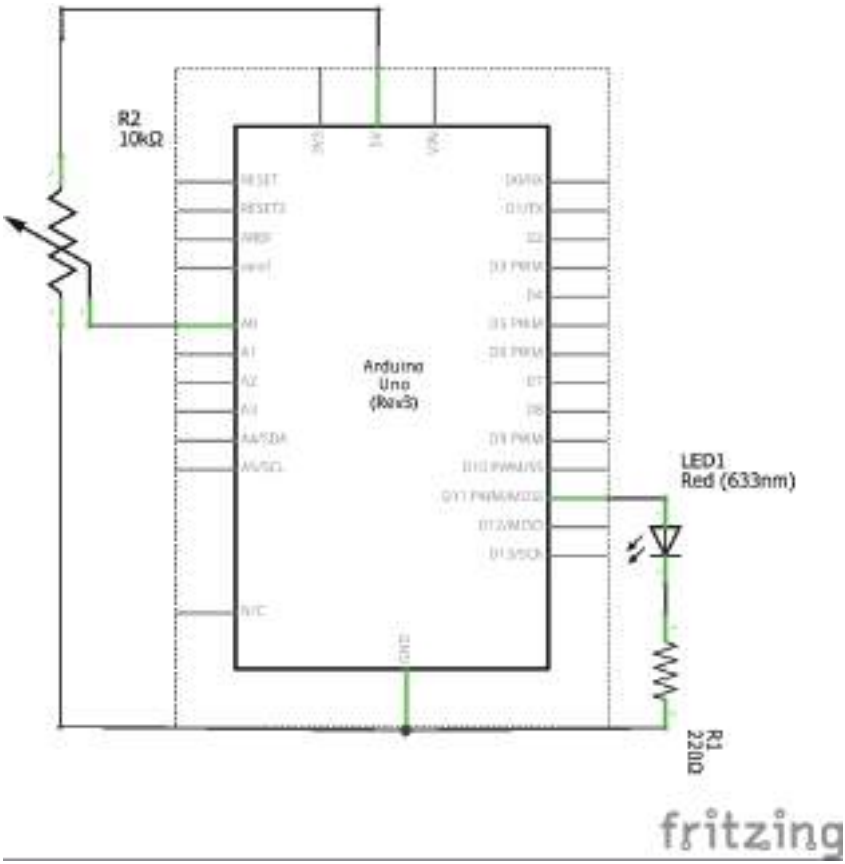


Рис. 47 ❖ Принципиальная схема подключения для практического занятия 3

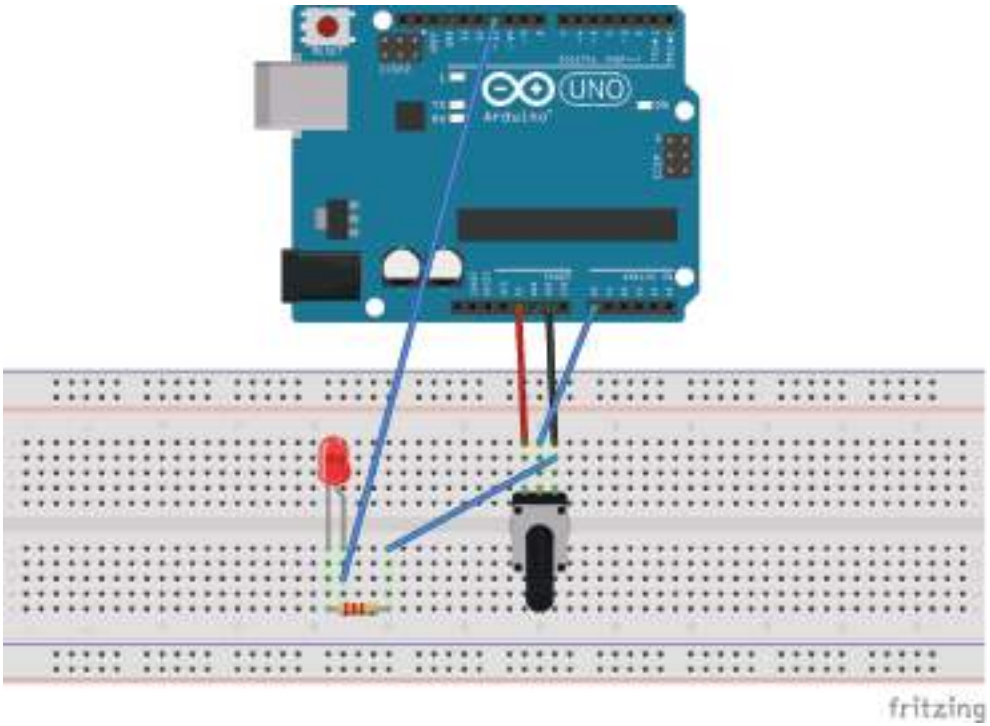


Рис. 48 ❖ Схема подключения с макетной платой для практического занятия 3

Код программы

```
    int potpin = 0 ;// define analog interface 0
    int ledpin = 11 ;// define the digital interface 11 (PWM output)
    int val = 0 ;// temporary values of the variables from the sensor
    void setup ()
    {
        pinMode (ledpin, OUTPUT) ;// define the digital interface 11 as output
        Serial.begin (9600) ;// set the baud rate to 9600
// NOTE: analog interface is automatically set to the input
    }
    void loop ()
    {
        val = analogRead (potpin) ;// read sensor analog values and assigned to val
        Serial.println (val) ;// display val variable
        analogWrite (ledpin, val / 4) ;// turn on the LED and set the brightness (PWM
output max 255)
        delay (10) ;// delay of 0.01 seconds
    }
```

Практическое занятие 4. Эксперимент с внешним мигающим светодиодом

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- светодиод;
- резистор на 220 Ом;
- макетная плата;
- соединительные провода.

В этом занятии мы подсоединим внешний светодиод к цифровому порту 10 платы. Напишем код, который будет заставлять светодиод загораться на секунду и гаснуть на секунду. В этот раз, по сравнению с занятием 2, светодиод у нас внешний, поэтому надо собрать схему, показанную ниже. В схеме с внешним светодиодом всегда должен быть резистор с малым сопротивлением (220 Ом), иначе светодиод может перегореть.

Схема представлена на рис. 49–50.

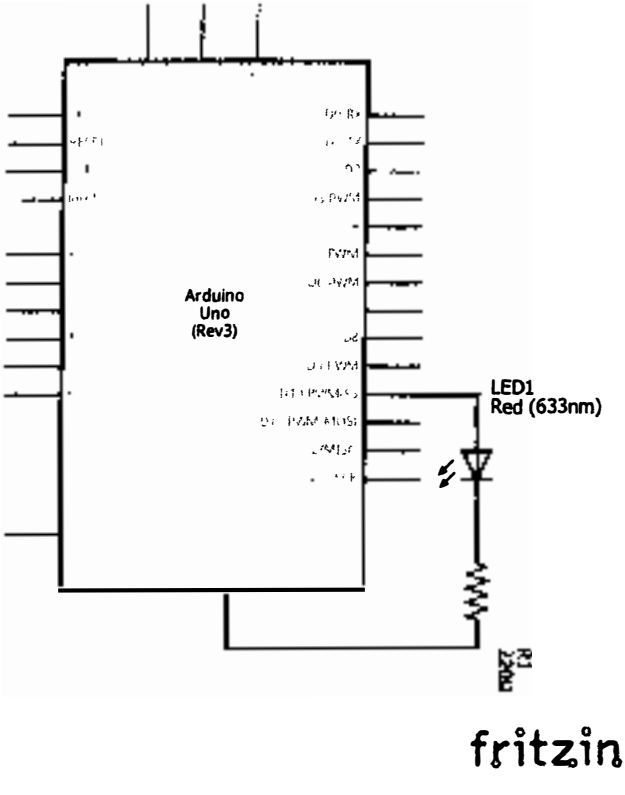


Рис. 49 ❖ Принципиальная схема подключения для практического занятия 4

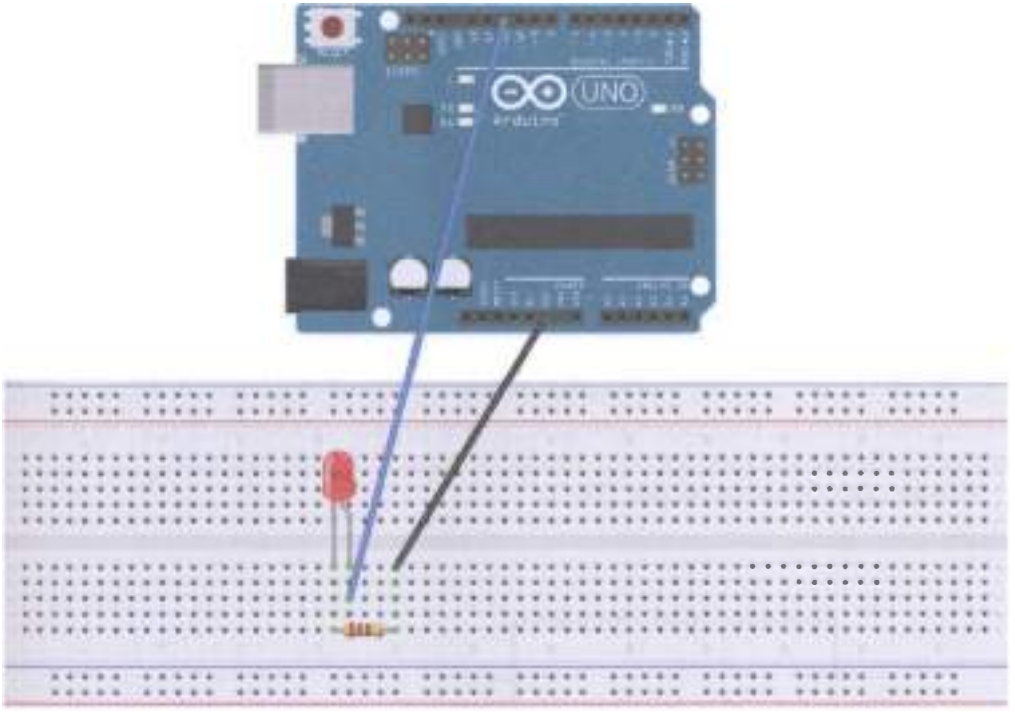


Рис. 50 ❖ Схема подключения с макетной платой для практического занятия 4

Код программы

```
int ledPin = 10; // define the interface number 10
void setup ()
{
  pinMode (ledPin, OUTPUT) ;// define a small lamp interface output interface
}
void loop ()
{
  digitalWrite (ledPin, HIGH); // lit a small lamp
  delay (1000); // Delay 1 second
  digitalWrite (ledPin, LOW); // extinguish small lights
  delay (1000); // Delay 1 second
}
```

Практическое занятие 5. Эксперимент с рекламной расцветкой

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- светодиоды – 6 шт., по 2 каждого цвета;
- резистор на 220 Ом – 6 шт.;
- макетная плата;
- соединительные провода.

В этом занятии мы напишем код, который будет заставлять 6 светодиодов, подключённых к чётным цифровым пинам платы Arduino Uno (2, 4, 6, 8, 10, 12), загораться волной, имитируя ёлочную гирлянду или рекламный щит – сначала все светодиоды загораются по очереди, потом гаснут по очереди.

Схема представлена на рис. 51–52.

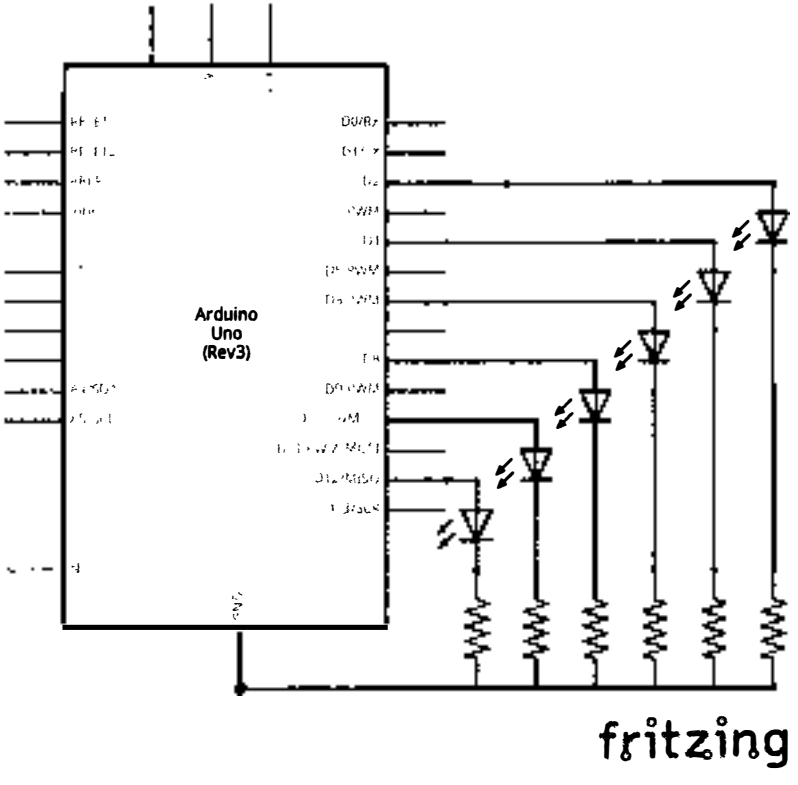


Рис. 51 ❖ Принципиальная схема подключения для практического занятия 5

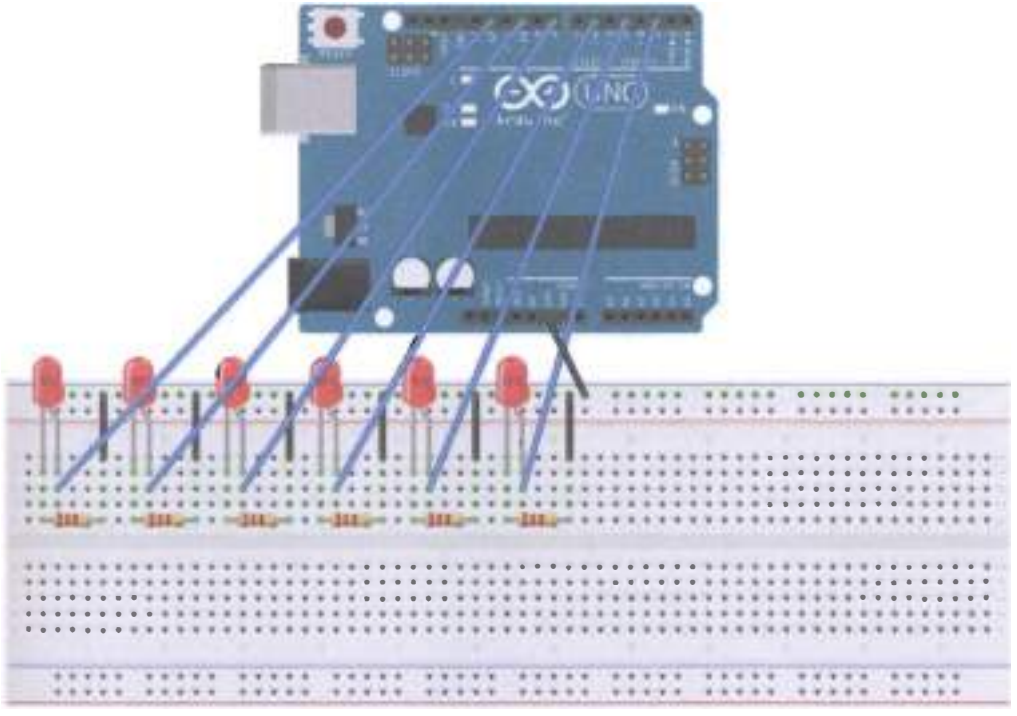


Рис. 52 ❖ Схема подключения с макетной платой для практического занятия 5

Код программы

```
int BASE = 2; // the first one LED connected to the I / O pins
int NUM = 6; // LED's total
void setup ()
{
for (int i = BASE; i <=BASE*NUM; i+=2)
{
    pinMode (i, OUTPUT); // set the digital I / O pin as an output
}
}
void loop ()
{
  for (int i = BASE; i <=BASE*NUM; i+=2)
  {
    digitalWrite (i, LOW); // set the digital I / O pin output is «low», that gradually
turn off the
    lights
    delay (200); // delay
  }
  for (int i = BASE; i <=BASE*NUM; i+=2)
  {
    digitalWrite (i, HIGH); // set the digital I / O pin output is «low», that gradually
lights
```

```
    delay (200); // delay
  }
}
```

Практическое занятие 6. Светофорный эксперимент

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- светодиоды – 3 шт., разных цветов;
- резистор на 220 Ом – 3 шт.;
- макетная плата;
- соединительные провода.

В этом занятии мы эмулируем работу светофора. В наборе не оказалось зелёного светодиода, поэтому будем использовать красный (цифровой пин 10 платы), жёлтый (пин 7) и синий (пин 4). Возможно, вам повезёт больше, и у вас будет зелёный светодиод. Напишем код, который будет заставлять 3 светодиода переключаться с примерно той задержкой, которая принята в реальных светофорах (от красного к зелёному).

Схема представлена на рис. 53–54.

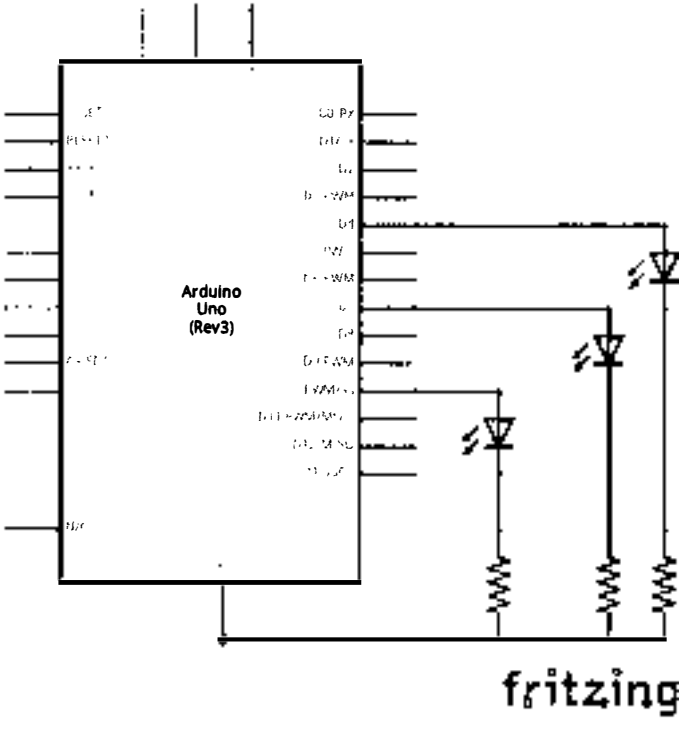


Рис. 53 ❖ Принципиальная схема подключения для практического занятия 6

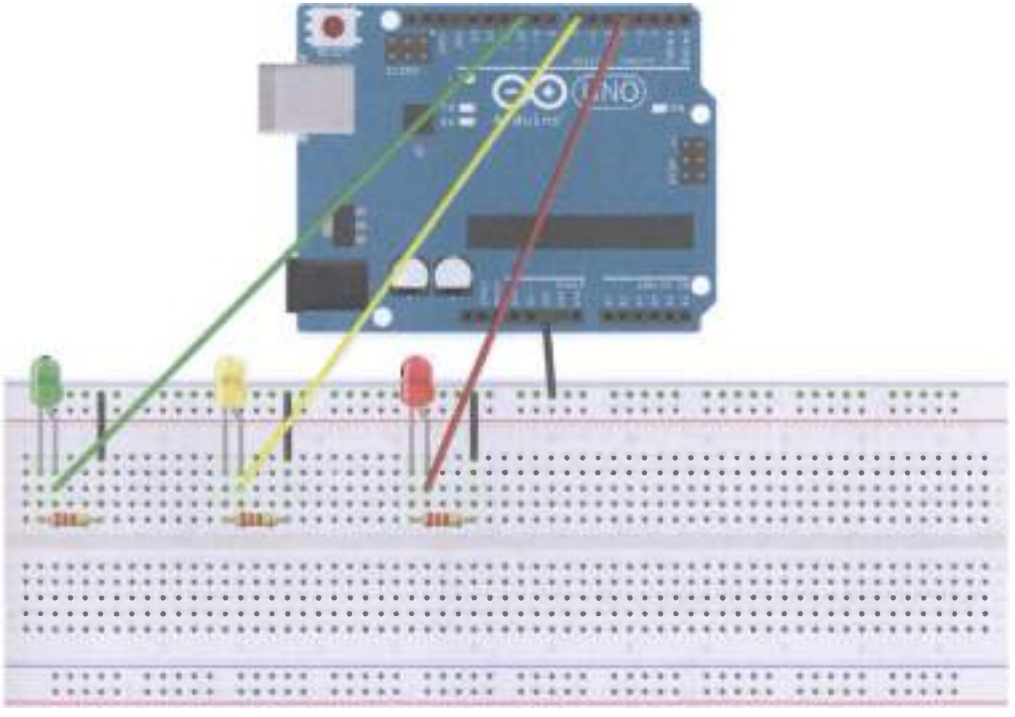


Рис. 54 ❖ Схема подключения с макетной платой для практического занятия 6

Код программы

```
int redled = 10; // define the interface number 10
        int yellowled = 7; // define the number 7 Interface
        int greenled = 4; // define the number 4 Interface
        void setup ()
        {
            pinMode (redled, OUTPUT) ;// define a small red light interface output
interface
            pinMode (yellowled, OUTPUT); // define the yellow light interface output
interface
            pinMode (greenled, OUTPUT); // define the small green light interface output
interface
}
        void loop ()
        {
            digitalWrite (redled, HIGH) ;// lit red lights
            delay (1000) ;// delay of 1 second
            digitalWrite (redled, LOW); // off red light
            digitalWrite (yellowled, HIGH) ;// light yellow light
            delay (200) ;// delay of 0.2 seconds
            digitalWrite (yellowled, LOW) ;// off yellow light
            digitalWrite (greenled, HIGH) ;// flashes the green LED
```

```
    delay (1000) ;// delay of 1 second
    digitalWrite (greenled, LOW) ;// green LED off
}
```

Практическое занятие 7. Эксперимент с пищалкой

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- динамик-пищалка;
- кнопка;
- макетная плата;
- соединительные провода.

В этом занятии мы должны слышать динамик-пищалку только тогда, когда нажимаем на кнопку. Также важно знать, что у пищалки есть + и – выходы и + подключается только к питанию или цифровому пину платы (в данном случае – пину 8), а минус – к земле. Если внимательно посмотреть на динамик-пищалку, можно увидеть помеченный + на корпусе рядом с + контактом.

Схема представлена на рис. 55–56.

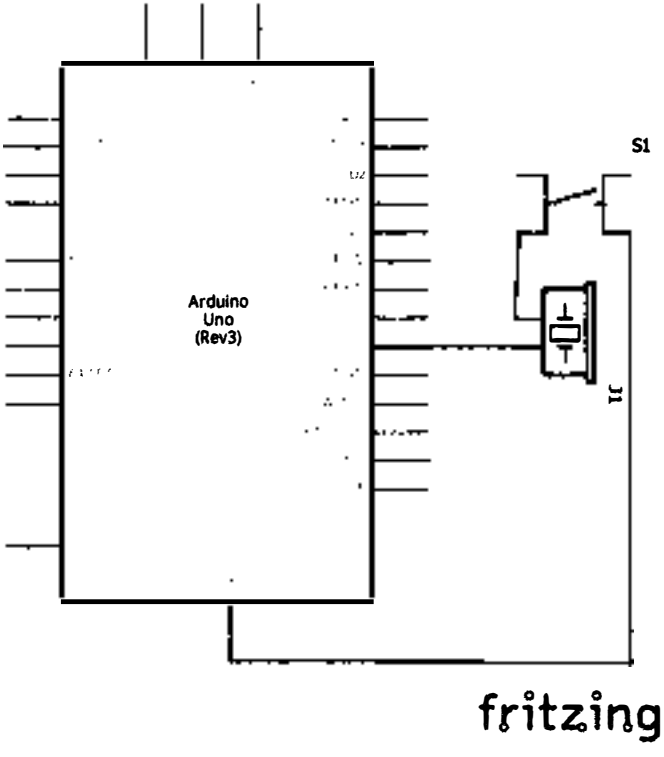


Рис. 55 ❖ Принципиальная схема подключения для практического занятия 7

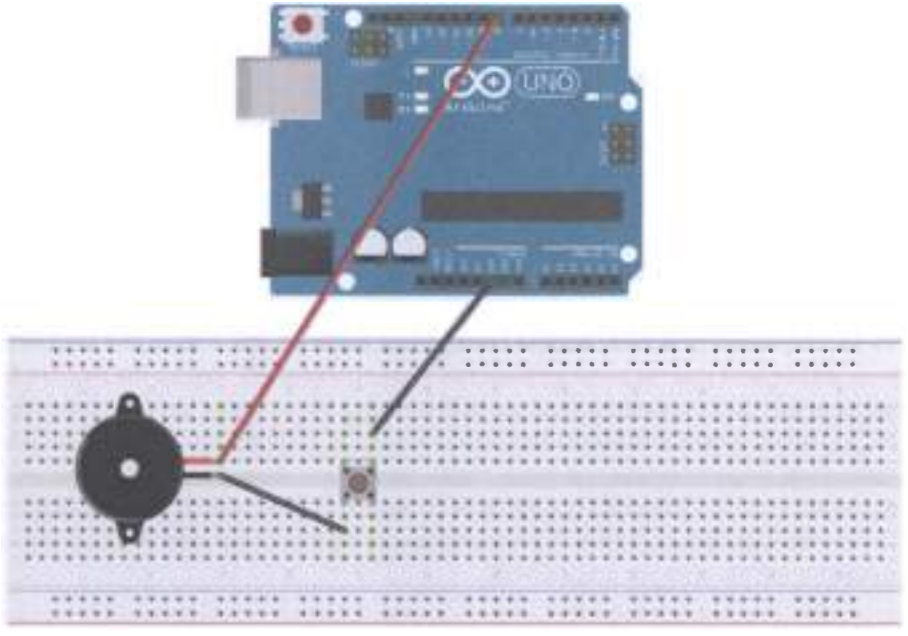


Рис. 56 ❖ Схема подключения с макетной платой для практического занятия 7

Код программы

```
int buzzer = 8 ;// setting controls the digital IO foot buzzer
void setup ()
{
    pinMode (buzzer, OUTPUT) ;// set the digital IO pin mode, OUTPUT out of Wen
}
void loop ()
{
    unsigned char i, j ;// define variables
    while (1)
    {
    for (i = 0; i <80; i++) // Wen a frequency sound
    {
        digitalWrite (buzzer, HIGH) ;// send voice
        delay (1) ;// Delay 1ms
        digitalWrite (buzzer, LOW) ;// do not send voice
        delay (1) ;// delay ms
    }
    for (i = 0; i <100; i++) // Wen Qie out another frequency sound
    {
        digitalWrite (buzzer, HIGH) ;// send voice
        delay (2) ;// delay 2ms
        digitalWrite (buzzer, LOW) ;// do not send voice
        delay (2) ;// delay 2ms
    }
    }
}
```

Практическое занятие 8. Эксперимент с датчиком наклона

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- датчик наклона (tilt switch);
- светодиод;
- резистор на 220 Ом;
- макетная плата;
- соединительные провода.

Суть эксперимента в том, чтобы светодиод включался, когда мы меняем положение макетной платы в пространстве (наклоняем её). Когда вы будете брать и вставлять в плату датчик наклона (tilt switch), то можете почувствовать, что внутри датчика действительно находится шарик (tilt mechanism – mechanical ball), который перекатывается туда-сюда, тем самым меняя напряжение. Если угол наклона платы около 90 градусов и больше, светодиод должен зажигаться, если же плата возвращается в горизонтальное положение, светодиод гаснет.

Схема представлена на рис. 57–58.

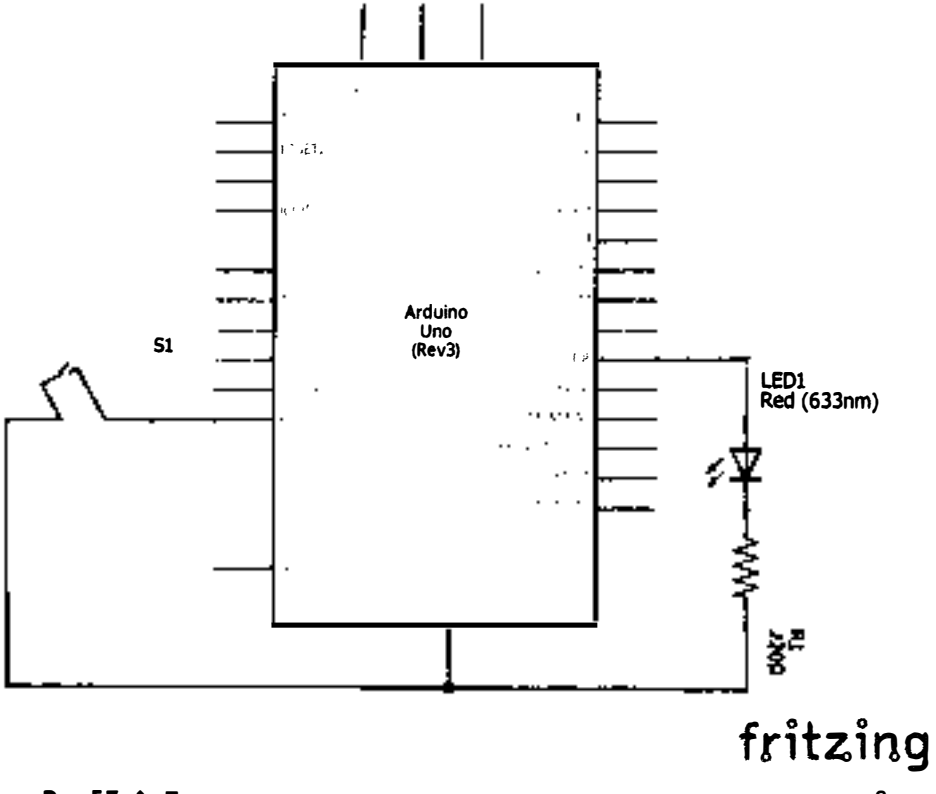


Рис. 57 ❖ Принципиальная схема подключения для практического занятия 8

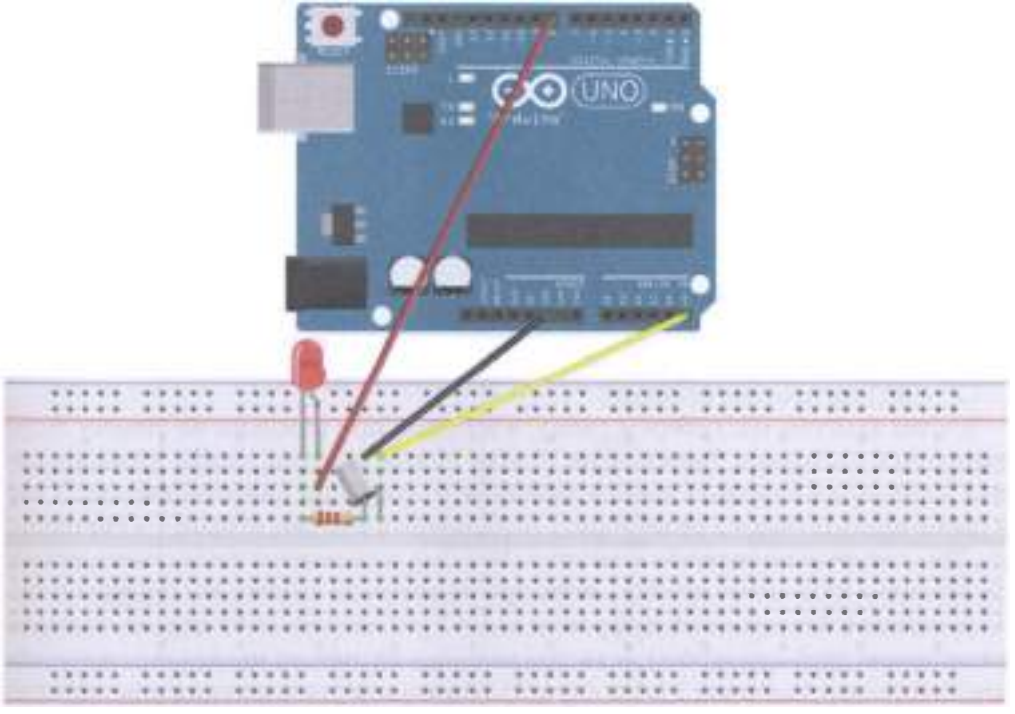


Рис. 58 ❖ Схема подключения с макетной платой для практического занятия 8

Код программы

```
void setup ()
{
    pinMode (8, OUTPUT) ;// set the digital 8 pin to out mode
}
void loop ()
{
    int i ;// define the amount of i
    while (1)
    {
    i = analogRead (5) ;// read the analog voltage value 5
    if (i> 200) // if greater than 512 (2.5V)
    {
        digitalWrite (8, HIGH) ;// led light is lit
    }
    else // Otherwise
    {
        digitalWrite (8, LOW) ;// off led light
 }
 }
}
```

Практическое занятие 9. Эксперимент с чистым входным сигналом

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- кнопка;
- светодиод;
- резистор на 220 Ом;
- резистор на 10 кОм;
- макетная плата;
- соединительные провода.

В этом занятии мы соберём схему, в которой при нажатии на кнопку будет загораться светодиод. Резистор на 10 кОм нужен для того, чтобы избежать помех в сигнале и подавать точное значение 0 или 1 при нажатой кнопке. У нас в схеме присутствуют 2 резистора: 1 подключён к кнопке, второй – к светодиоду. Тот, который подключён к кнопке (10 кОм), называется либо стягивающим резистором (pull-down resistor), если при нажатии на кнопку мы должны получать сигнал 0 без помех на цифровом пине 7 (см. схему), либо подтягивающим резистором (pull-up resistor), если при нажатии на кнопку мы должны получить чистый сигнал 1 на пине 7 (наш случай). Номиналы стягивающих или подтягивающих резисторов обычно варьируются от 1 кОм до 10 кОм. Тот же резистор, который подключён к катоду светодиода, называется серийным, или последовательным (series resistor), и нужен для того, чтобы брать на себя часть тока во избежание перегорания светодиода. Значения таких резисторов обычно варьируются от 100 Ом до 300 Ом. Почему нам важен чистый сигнал в этом занятии? Потому что мы читаем его с пина 7 с помощью функции digitalRead(7).

Схема представлена на рис. 59–60.

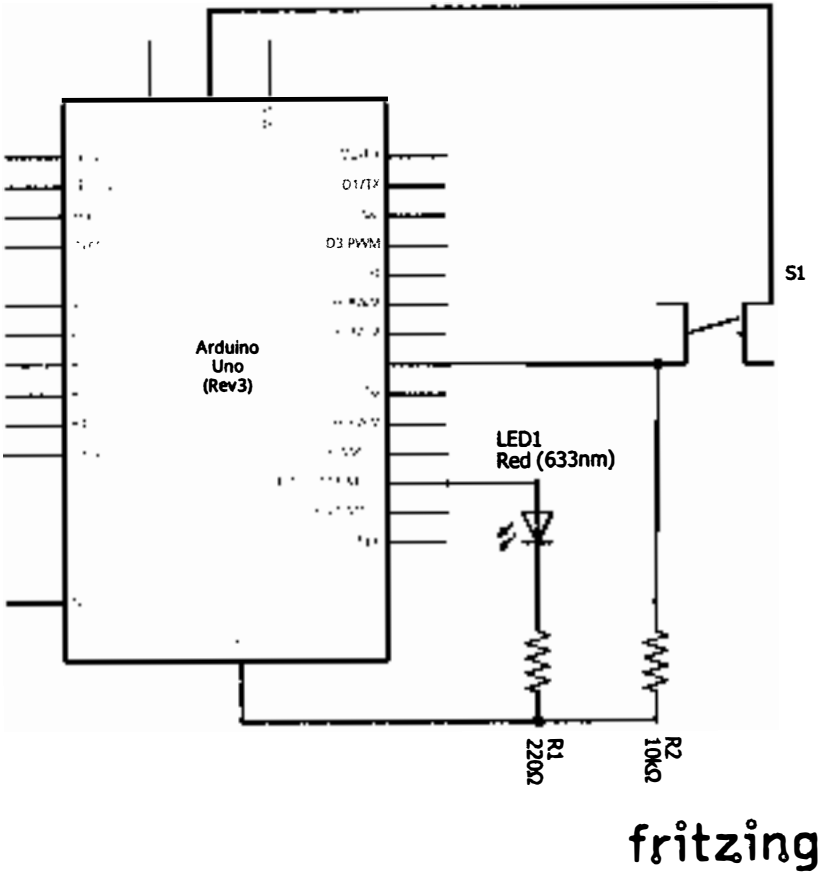


Рис. 59 ❖ Принципиальная схема подключения для практического занятия 9

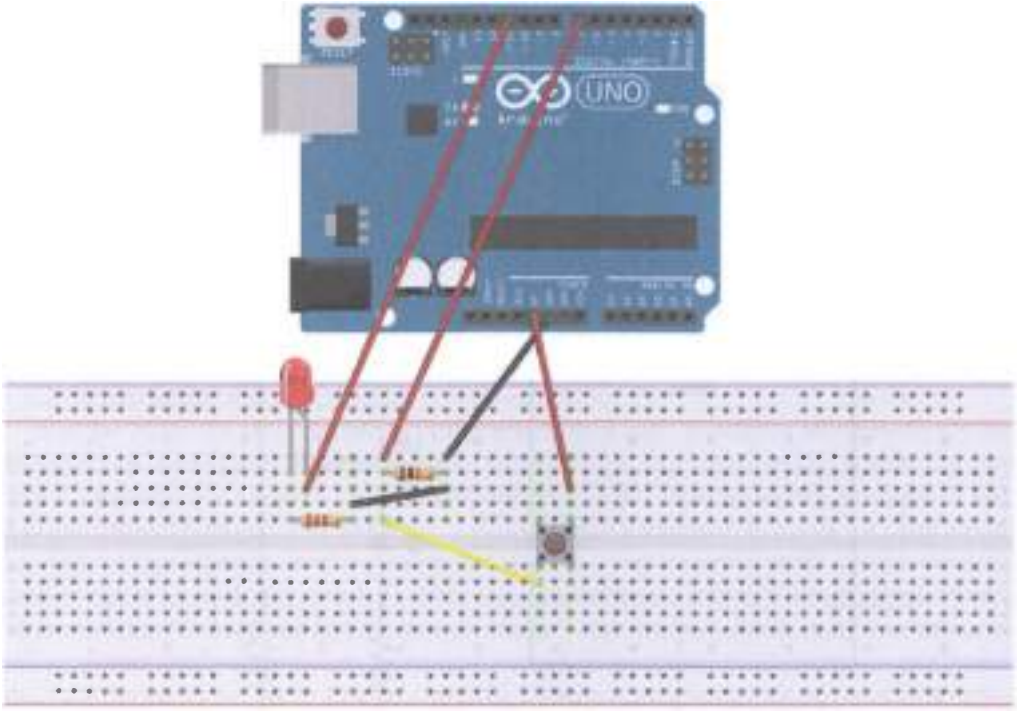


Рис. 60 ❖ Схема подключения с макетной платой для практического занятия 9

Код программы

```
int ledpin = 11 ;// define the interface number 11
int inpin = 7 ;// define the number 7 Interface
int val ;// define a variable val
void setup ()
{
    pinMode (ledpin, OUTPUT) ;// define a small lamp interface output interface
    pinMode (inpin, INPUT) ;// define the key interface for the input interface
}
void loop ()
{
    val = digitalRead (inpin) ;// read digital 7-level value assigned to val
    if (val == LOW) // test button is pressed, the button lights up when pressed small
    {digitalWrite (ledpin, LOW);}
    else
    {digitalWrite (ledpin, HIGH);}
}
```

Практическое занятие 10. Расширенный эксперимент с чистым сигналом

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- кнопка – 3 шт.;
- светодиоды разных цветов – 3 шт.;
- резистор на 220 Ом – 3 шт.;
- резистор на 10 кОм – 3 шт.;
- макетная плата;
- соединительные провода.

В этом занятии мы соберём схему, в которой при нажатии на каждую из кнопок будет загораться соответствующий светодиод. В сложных схемах, подобных этой, лучше всего выводить питание и землю на отдельные рельсы (+ и –, или красный и синий горизонтальный рельс) макетной платы.

Схема представлена на рис. 61–62.

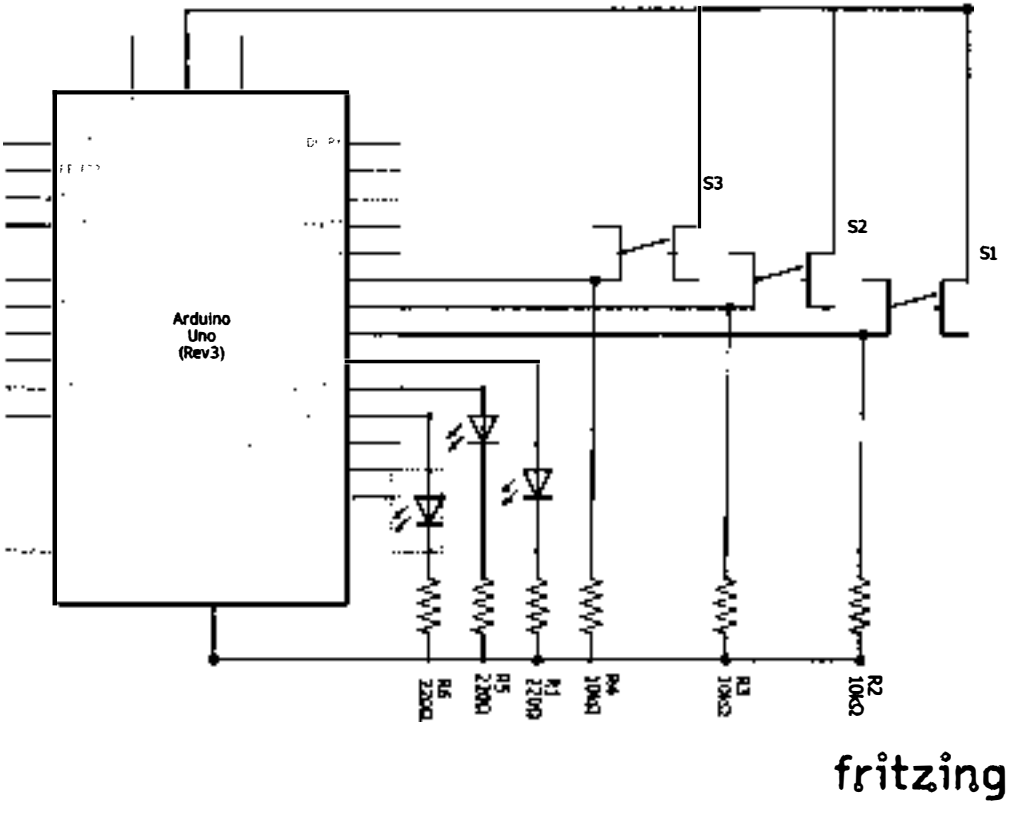


Рис. 61 ❖ Принципиальная схема подключения для практического занятия 10

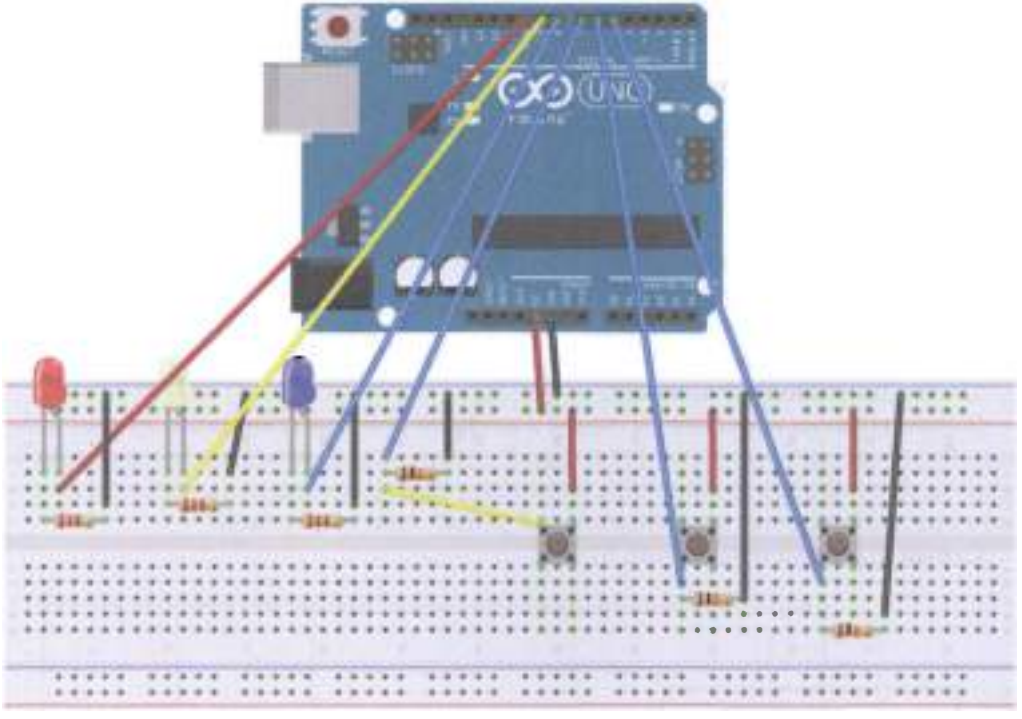


Рис. 62 ❖ Схема подключения с макетной платой для практического занятия 10

Код программы

```
int redled = 10;
int yellowled = 9;
int greenled = 8;
int redpin = 7;
int yellowpin = 6;
int greenpin = 5;
int red;
int yellow;
int green;
void setup ()
{
    pinMode (redled, OUTPUT);
    pinMode (yellowled, OUTPUT);
    pinMode (greenled, OUTPUT);
    pinMode (redpin, INPUT);
```

```
    pinMode (yellowpin, INPUT);
    pinMode (greenpin, INPUT);
}
void loop ()
{
    red = digitalRead (redpin);
    if (red == LOW)
    {digitalWrite (redled, LOW);}
    else
    {digitalWrite (redled, HIGH);}
    yellow = digitalRead (yellowpin);
    if (yellow == LOW)
    {digitalWrite (yellowled, LOW);}
    else
    {digitalWrite (yellowled, HIGH);}
    green = digitalRead (greenpin);
    if (green == LOW)
    {digitalWrite (greenled, LOW);}
    else
    {digitalWrite (greenled, HIGH);}
}
```

Практическое занятие 11. Эксперимент по чтению аналогового значения

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- потенциометр;
- макетная плата;
- соединительные провода.

У Arduino Uno есть аналоговые входы A0–A5. Воспользуемся входом A0 и функцией analogRead(), чтобы считывать данные о текущем сопротивлении потенциометра (значения от 0 до 1023, т. к. Arduino Uno располагает 10-битными аналоговыми или цифровыми портами, 2^10 = 1024 значения) и выводить их на экран. В коде используется команда Serial.begin(9600), означающая, что скорость/частота обмена данными платы с компьютером по UBS-соединению составляет 9600 bps (bits per second, битов в секунду). В консоли должна быть выставлена такая же частота (справа внизу). Не забудьте открыть консоль, чтобы видеть сообщения (**Ctrl**+**Shift**+**M**)!

Схема представлена на рис. 63–64.

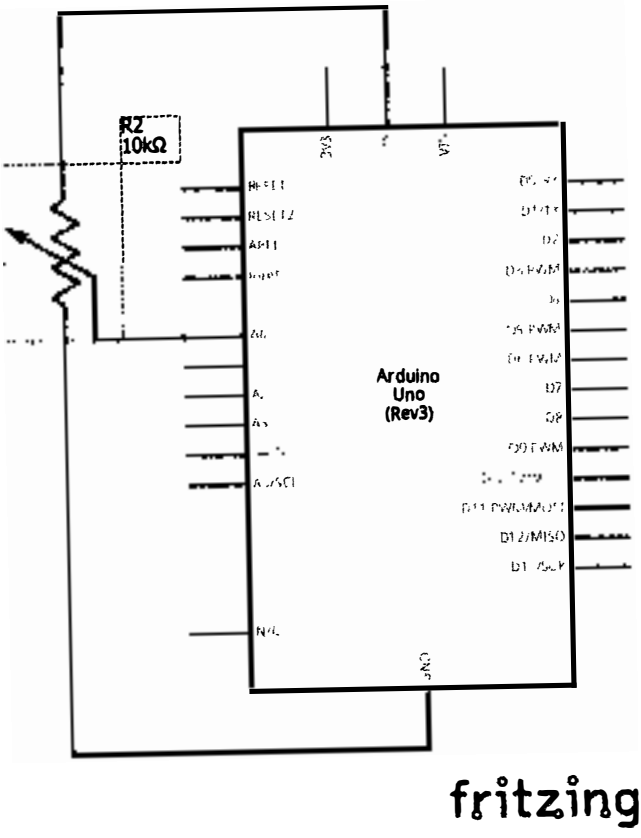


Рис. 63 ❖ Принципиальная схема подключения для практического занятия 11

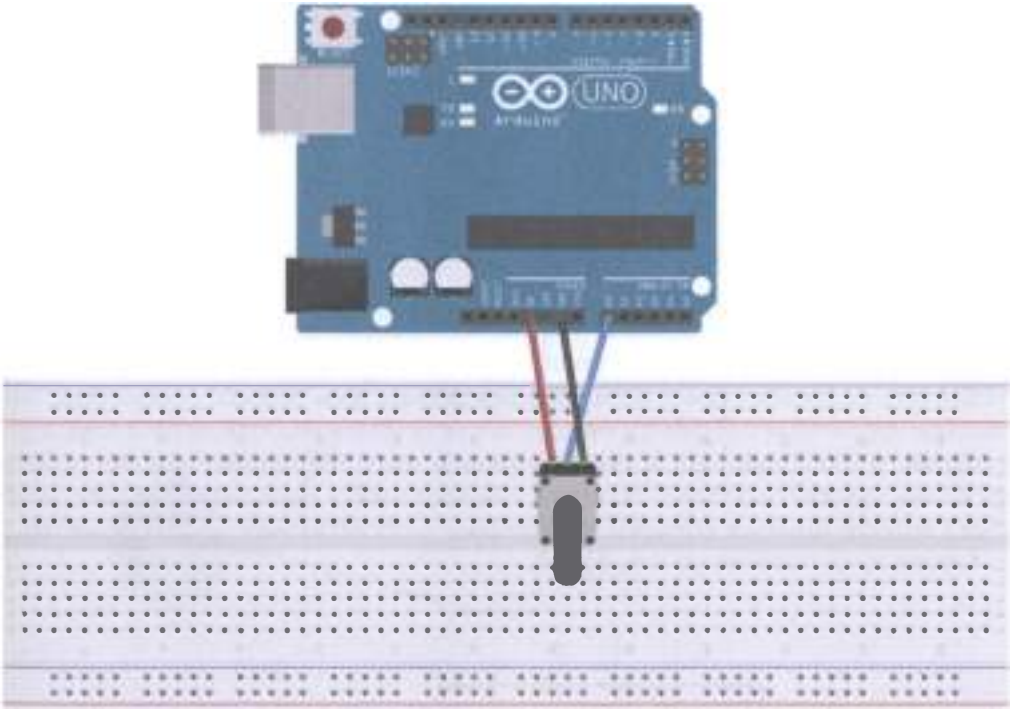


Рис. 64 ❖ Схема подключения с макетной платой для практического занятия 11

Код программы

```
int potpin = 0 ;// define analog interface 0
int ledpin = 13 ;// define the digital interface 13
int val = 0 ;// will define the variable val, and the initial value 0
void setup ()
{
    pinMode (ledpin, OUTPUT) ;// output interface defines the digital interface
    Serial.begin (9600) ;// set the baud rate to 9600
}
void loop ()
{
    digitalWrite (ledpin, HIGH) ;// digital interface 13 of the LED lights
    delay (50) ;// delay of 0.05 seconds
    digitalWrite (ledpin, LOW) ;// off LED digital interface 13
    delay (50) ;// delay of 0.05 seconds
    val = analogRead (potpin) ;// read the value of analog interface 0, and assign val
    Serial.println (val) ; // shows the value of val
}
```

Практическое занятие 12. Эксперимент по управлению звуком и светом

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- фоторезистор;
- динамик-пищалка;
- макетная плата;
- соединительные провода.

В данном занятии мы подсоединяем фоторезистор к пищалке, для того чтобы управлять её громкостью с помощью освещения. Чем больше освещения, тем меньше сопротивление фоторезистора и, соответственно, больше сила тока, протекающего через пищалку, а значит – громче писк. Фоторезисторы могут быть не очень чувствительными, поэтому, для того чтобы услышать пищалку, надо будет воспользоваться фонарём в смартфоне – посветить им на фоторезистор.

Схема представлена на рис. 65–66.

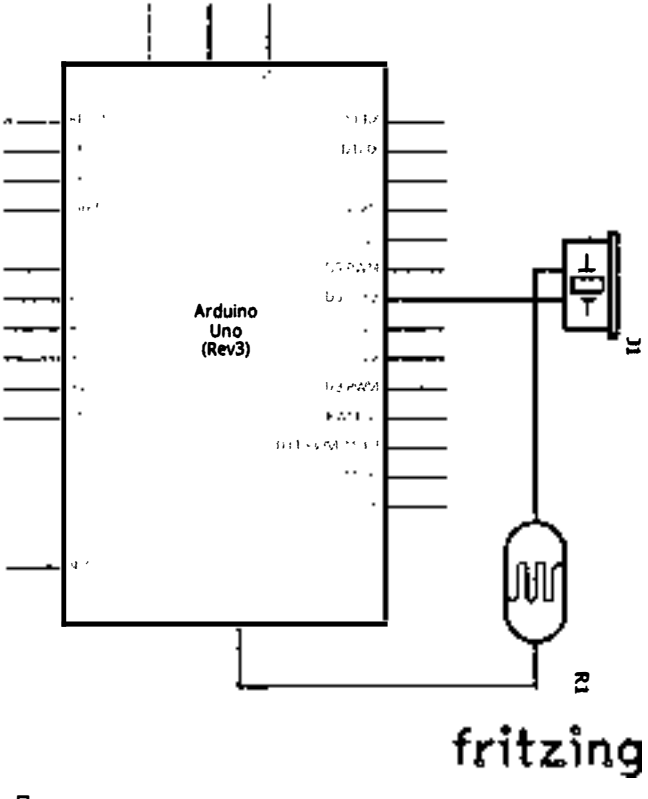


Рис. 65 ❖ Принципиальная схема подключения для практического занятия 12

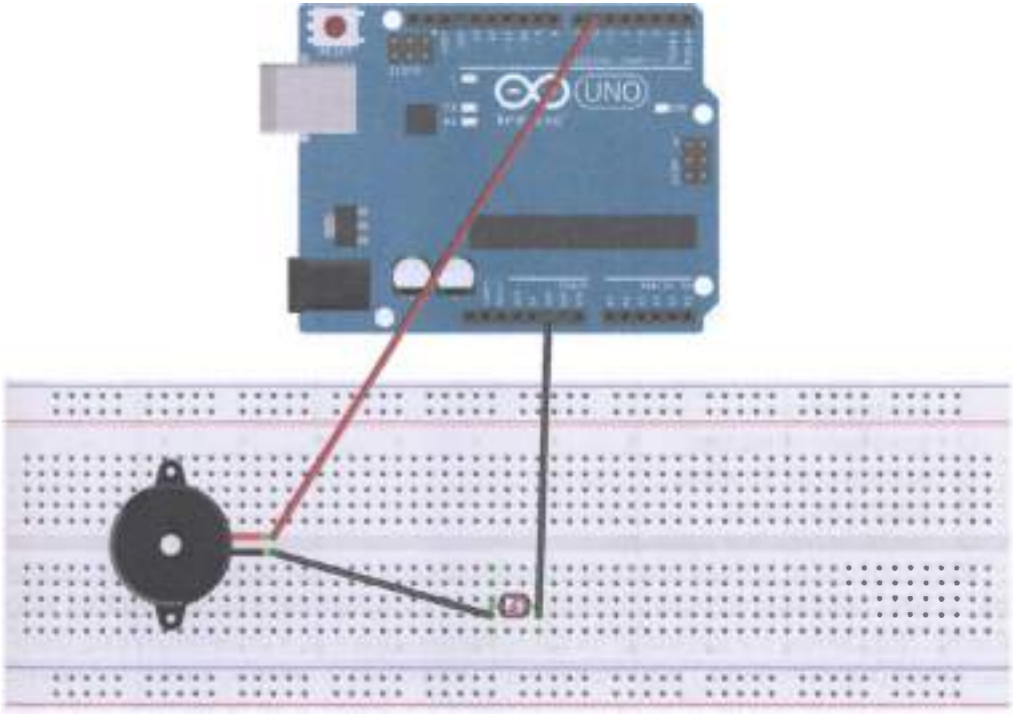


Рис. 66 ❖ Схема подключения с макетной платой для практического занятия 12

Код программы

```
void setup ()
{
    pinMode (6, OUTPUT);
}
void loop ()
{
    while (1)
    {
        char i, j;
        while (1)
        {
            for (i = 0; i <80; i++) // Wen Qie one frequency sound
            {
                digitalWrite (6, HIGH);
                delay (1);
                digitalWrite (6, LOW);
                delay (1);
            }
            for (i = 0; i <100; i++) // Wen Qie out another frequency sound
            {
                digitalWrite (6, HIGH);
```

```
                delay (2);
                digitalWrite (6, LOW);
                delay (2);
            }
        }
    }
}
```

Практическое занятие 13. Эксперимент с датчиком огня

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- датчик огня;
- динамик-пищалка;
- резистор на 10 кОм;
- макетная плата;
- соединительные провода.

Инфракрасный датчик огня очень чувствителен к цветовому спектру пламени. В нём свечение пламени преобразуется в слабый сигнал, идущий на вход платы. Подключается датчик огня следующим образом, в отличие от светодиода: катод (короткий пин) подключается к источнику питания, а анод (длинный пин) через резистор на 10 кОм подключается к земле, а до резистора идёт выход на аналоговый пин платы Arduino Uno. Датчик огня работает и без огня, выдавая на аналоговый пин платы около 0.3 В. Когда пламя находится слишком близко, датчик выдаёт уже 1.0 В и больше. Поэтому в коде было бы целесообразно использовать пороговое значение 0.6 В. Однако практика показала, что на самом деле датчик очень чувствительный, и даже в комнате с верхним освещением из трёх обыкновенных ламп накаливания выдаёт значение 984–985, что соответствует 4.8 В, хотя никакого огня в комнате нет. Поэтому в коде пороговое значение выставлено в 1000. В этом задании нам также понадобится консоль (**Ctrl+Shift+M**) и зажигалка/спички (зажигалка, конечно, получше спичек).

Схема представлена на рис. 67–68.

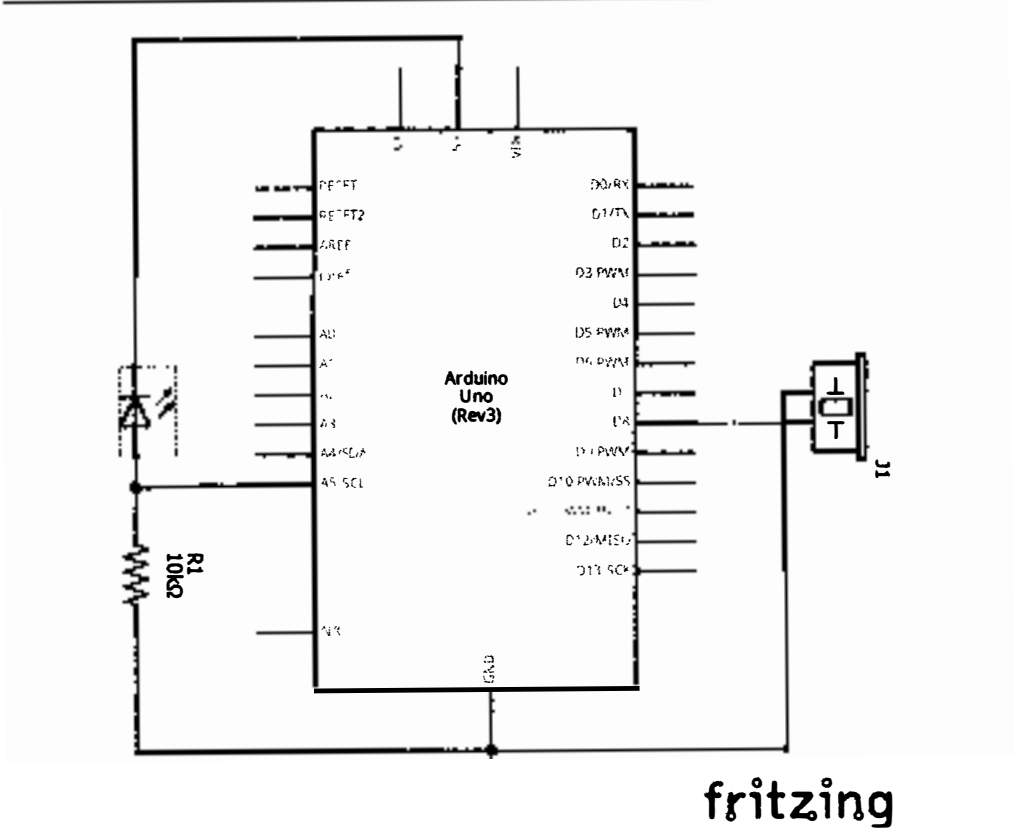


Рис. 67 ❖ Принципиальная схема подключения для практического занятия 13

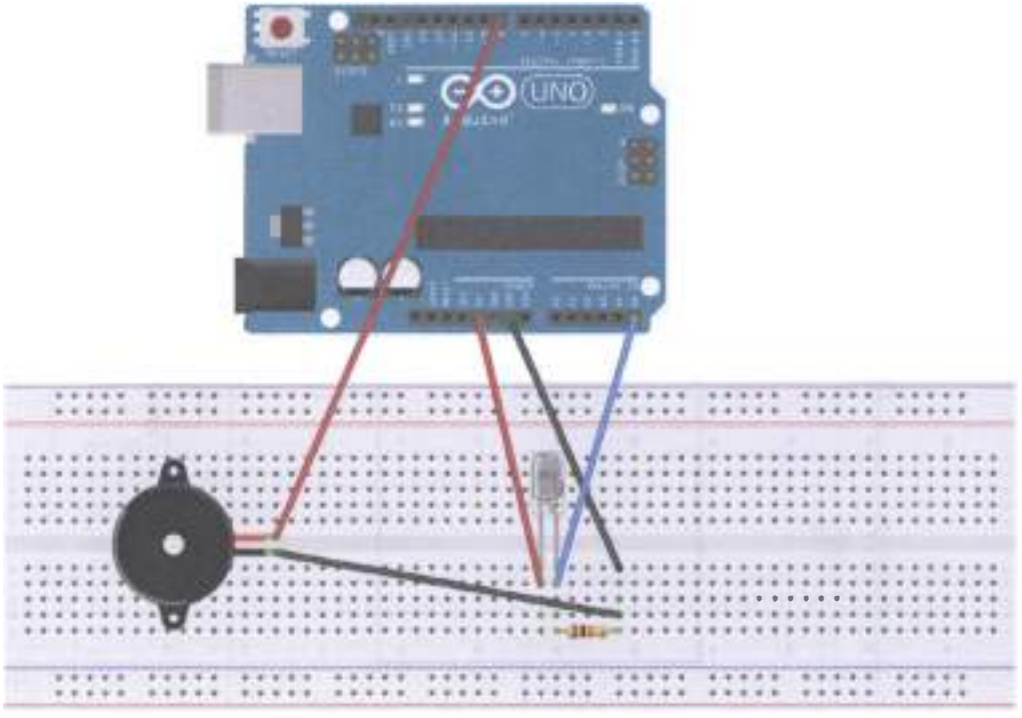


Рис. 68 ❖ Схема подключения с макетной платой для практического занятия 13

Код программы

```
int flame = A5;// define the flame interface analog 0 interface
int Beep = 8;// buzzer interface defines the interface number 7
int val = 0;// define numeric variables val
void setup ()
    {
        pinMode (Beep, OUTPUT) ;// define LED as output interface
        pinMode (flame, INPUT) ;// define the buzzer as the input interface
        Serial.begin (9600) ;// set the baud rate to 9600
    }
void loop () {
        val = analogRead (flame) ;// read the analog value flame sensor
        Serial.println (val) ;// output analog values, and print them out
        if (val >= 1000) // When the analog value is greater than 600 when the buzzer
sounds
            {
            digitalWrite (Beep, HIGH);
            } else {
            digitalWrite (Beep, LOW);
            }
    }
```

Практическое занятие 14. Эксперимент с вольтметром

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- резистор на 1 кОм;
- макетная плата;
- батарейка/аккумулятор на 1.5 В (**не больше**);
- соединительные провода.

В этом занятии будем строить схему с вольтметром с диапазоном 0–5 В. Нам также понадобится консоль (**Ctrl+Shift+M**), чтобы смотреть, какие значения мы считываем с аналогового порта 0 платы. При подключении схемы в консоли мы будем видеть, какой заряд у батарейки/аккумулятора.

Схема представлена на рис. 69–70.

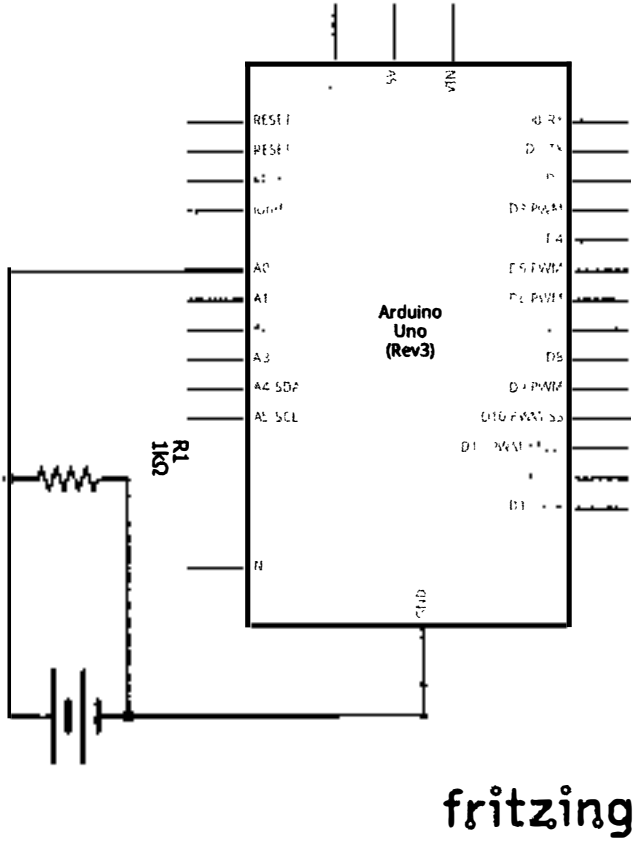


Рис. 69 ❖ Принципиальная схема подключения для практического занятия 14

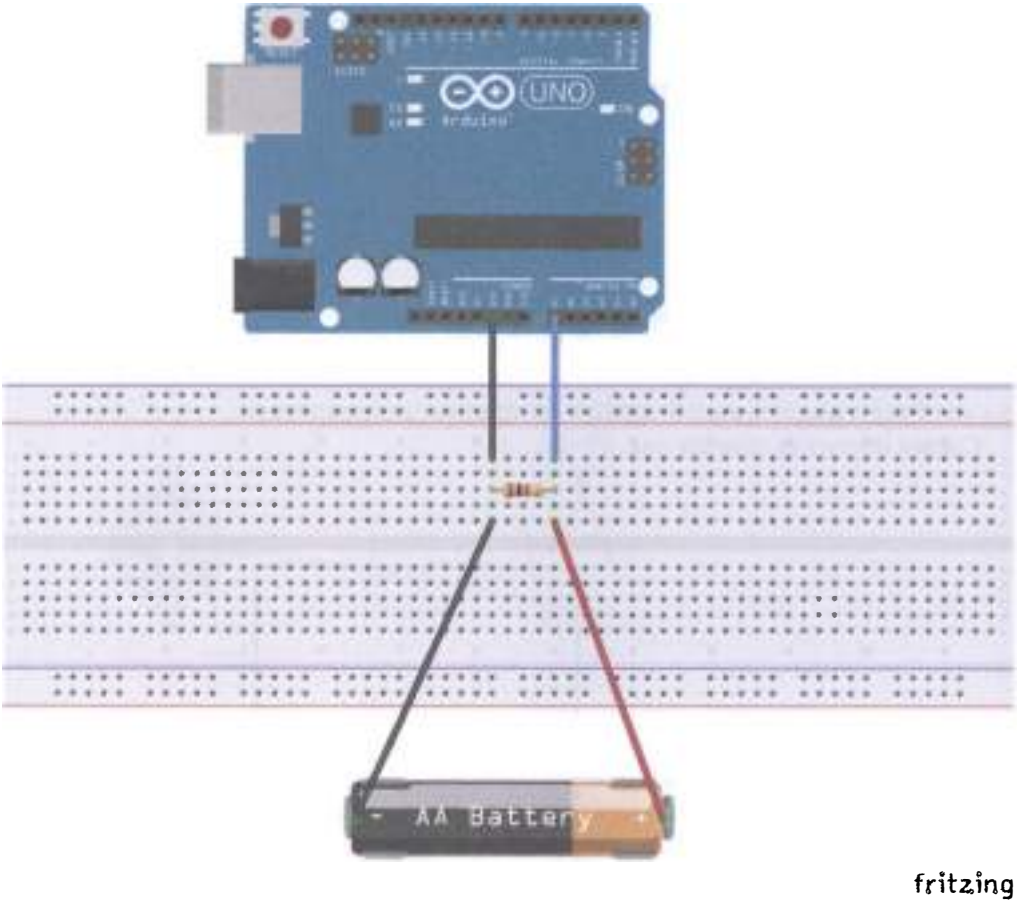


Рис. 70 ❖ Схема подключения с макетной платой для практического занятия 14

Код программы

```
float temp; // create a float variable temp as a storage space to store data preparation
void setup ()
{
Serial.begin (9600); // use 9600 baud rate serial communication
}
void loop ()
{
int V1 = analogRead (A0);
// Read voltage from A0 mouth integer type data into the newly created variable V1, analog port voltage measurement range of 0-5V returns a value of 0-1024
float vol = V1 * (5.0 / 1023.0);
// We will be converted into the actual value of V1 voltage value into a float variable vol
if (vol == temp)
// This part of the judgment is used to filter duplicate data, only the second voltage value when the output and the last mixed
```

```
{
temp = vol; // After the completion of the comparison, the ratio of this value into a
variable temp for comparison
}
else
{
Serial.print (vol); // serial output voltage value, and do not wrap
Serial.println ("V"); // serial output character V, and line breaks
temp = vol;
delay (1000); // Wait a second output is completed, the data used to control the refresh
rate.
}
}
```

Практическое занятие 15. Эксперимент с распознаванием голоса

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- модуль распознавания шумов;
- светодиод;
- резистор на 220 Ом;
- макетная плата;
- соединительные провода.

У модуля распознавания шумов есть 4 выхода. A0 – аналоговый выход, который мы подключаем к аналоговому входу A0 платы. G – земля, а + выход надо подключить к 5 В на плате. Далее собираем схему со светодиодом, как обычно, и открываем консоль (**Ctrl+Shift+M**), чтобы смотреть значение шума, создаваемого в аудитории. На схеме ниже представлен другой элемент с 4 выходами (не модуль распознавания шумов), так как в библиотеке Fritzing не оказалось подобного модуля или микрофона с 3 выходами; но главное – суть подключения.

Схема представлена на рис. 71–72.

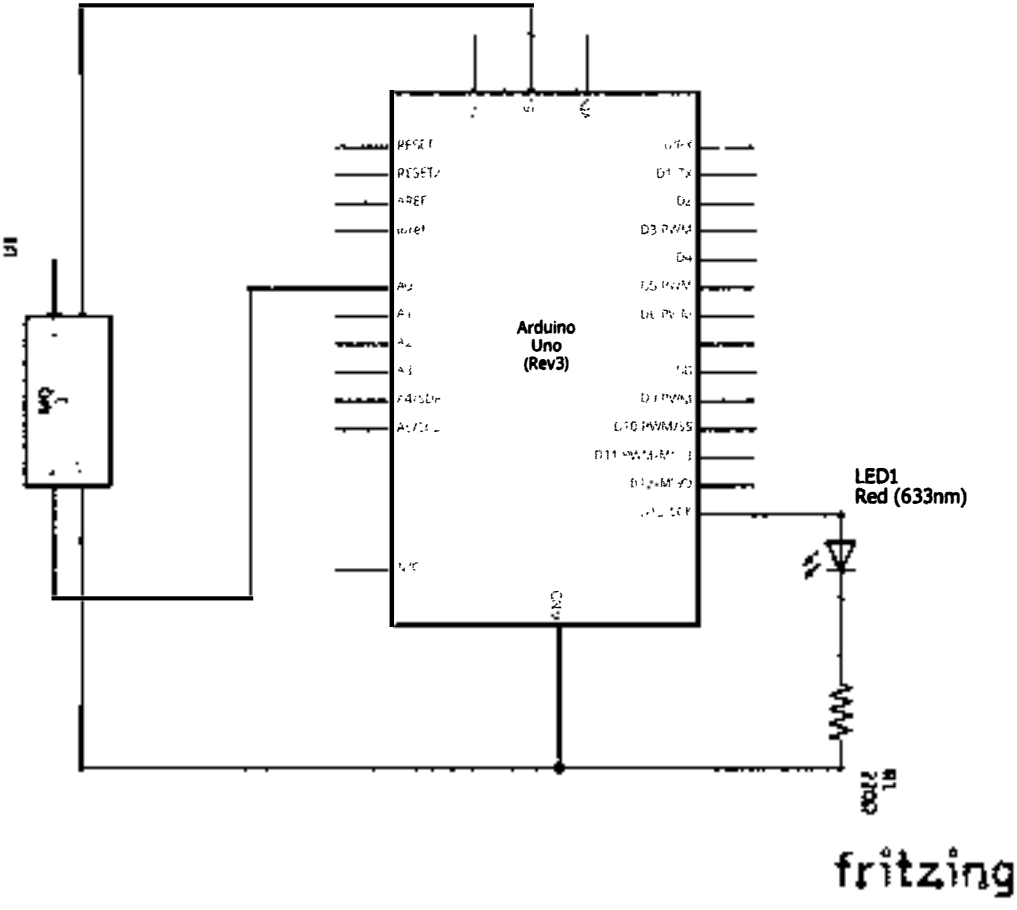


Рис. 71 ❖ Принципиальная схема подключения для практического занятия 15

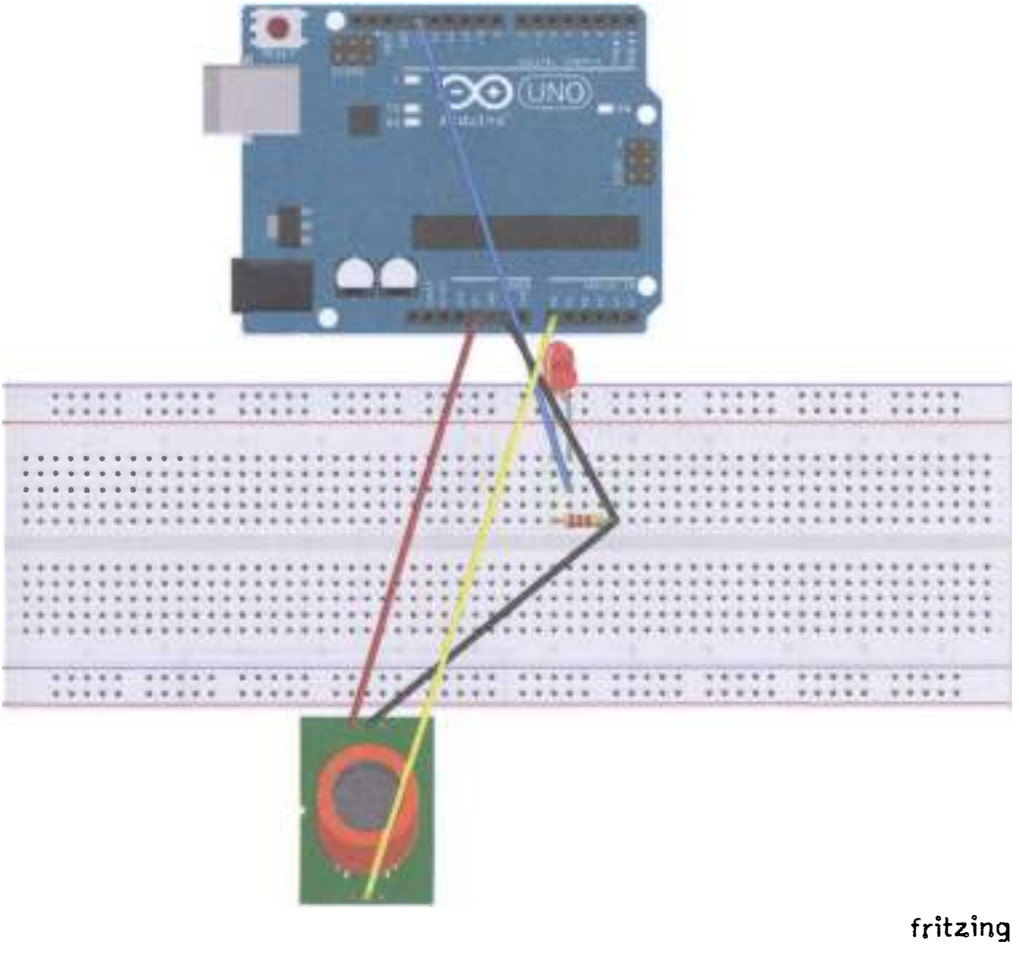


Рис. 72 ❖ Схема подключения с макетной платой для практического занятия 15

Код программы

```
int sensorPin = A0; // select the input pin for the potentiometer
int ledPin = 13; // select the pin for the LED
int sensorValue = 0; // variable to store the value coming from the sensor
void setup () {
pinMode (ledPin, OUTPUT);
Serial.begin (9600);
}
void loop () {
sensorValue = analogRead (sensorPin);
digitalWrite (ledPin, HIGH);
delay (sensorValue);
digitalWrite (ledPin, LOW);
delay (sensorValue);
Serial.println (sensorValue, DEC);
}
```

Практическое занятие 16. Эксперимент с температурным сенсором

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- температурный сенсор LM35;
- макетная плата;
- соединительные провода.

Температурный сенсор LM35 достаточно широко используется и прост в эксплуатации. Единственная нетривиальная часть этого задания – перевод аналоговых значений, которые выдаёт сенсор, в привычный вид. Также нам понадобится консоль (**Ctrl+Shift+M**).

Схема представлена на рис. 73–74.

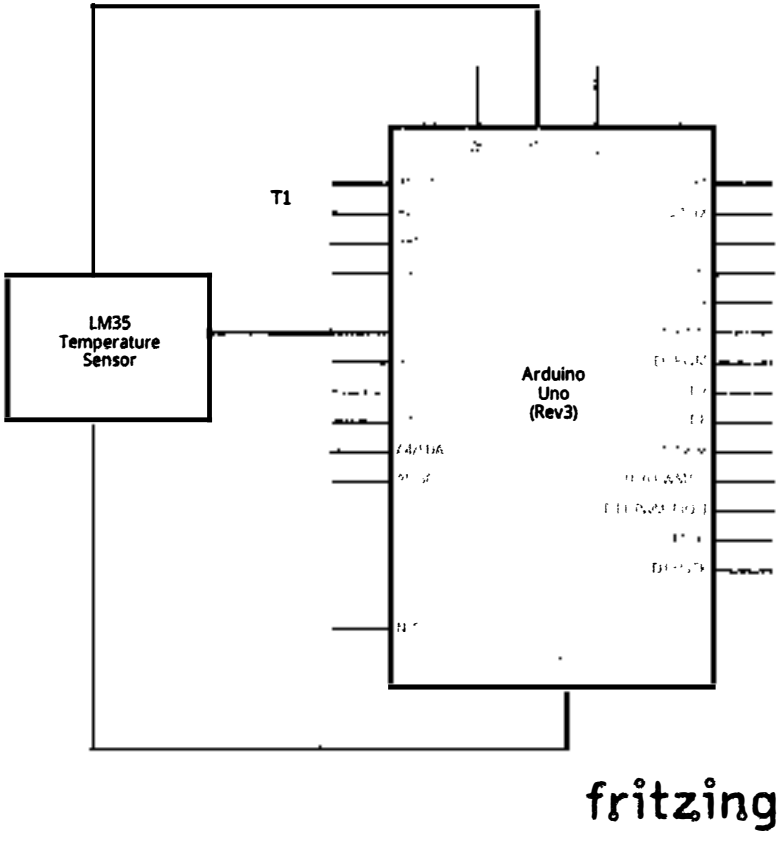


Рис. 73 ❖ Принципиальная схема подключения для практического занятия 16

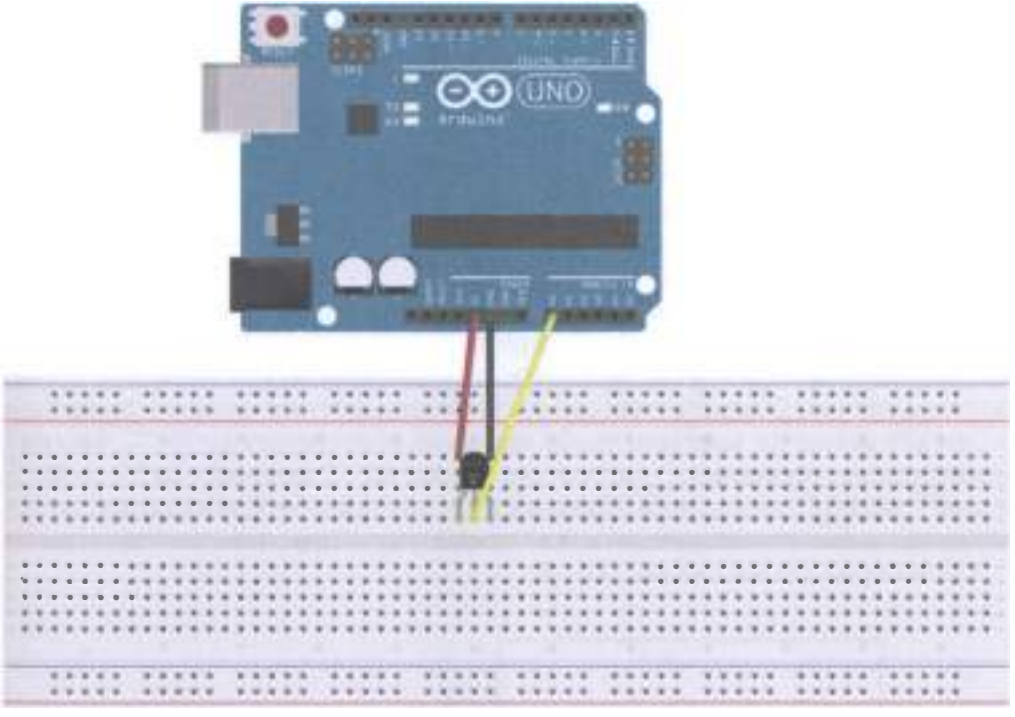


Рис. 74 ❖ Схема подключения с макетной платой для практического занятия 16

Код программы

```
int potPin = A0; // define the analog interface 0 LM35 temperature sensor connection
void setup ()
{
Serial.begin (9600) ;// set the baud rate
}
void loop ()
{
int val ;// define variables
int dat ;// define variables
val = analogRead (potPin) ;// read sensor analog values and assigned to val
dat = (125 * val) >> 8 ;// temperature formula
Serial.print ("Temp:") ;// output as a string represents the temperature display Tep
Serial.print (dat) ;// output shows the value of dat
Serial.println ("C") ;// output display as a C string
delay (500) ;// delay of 0.5 seconds
}
```

Практическое занятие 17. Разноцветный термостат

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- температурный сенсор LM35;
- 3 разноцветных светодиода;
- 3 резистора на 220 Ом;
- макетная плата;
- соединительные провода.

Суть этого задания в том, чтобы при определённой температуре зажигался соответствующий светодиод. Для нагрева атмосферы может понадобиться зажигалка/спички. С другой стороны, достаточно коснуться пальцем до датчика в течение какого-то времени, чтобы повысить температуру до 36 °C. Красный светодиод загорается, если температура больше 40 °C, зелёный – от 32 до 40 °C, синий – до 31 °C. Ещё хотелось бы видеть значения температуры в консоли, поэтому добавляем соответствующий код из предыдущего практического занятия.

Схема представлена на рис. 77–78.

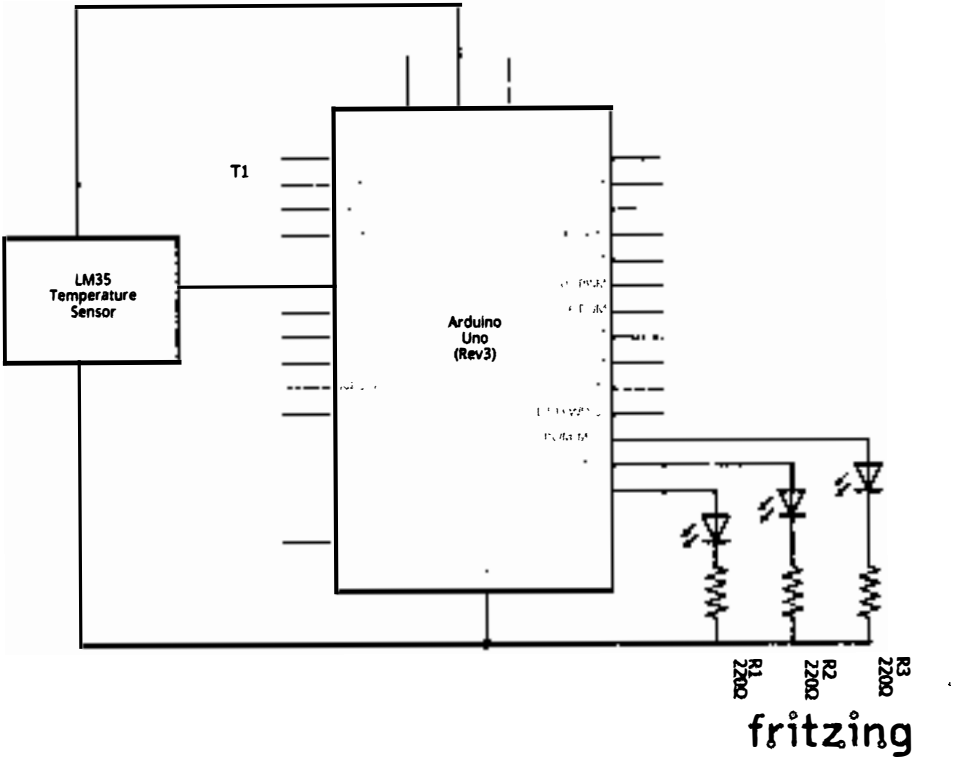


Рис. 75 ❖ Принципиальная схема подключения для практического занятия 17

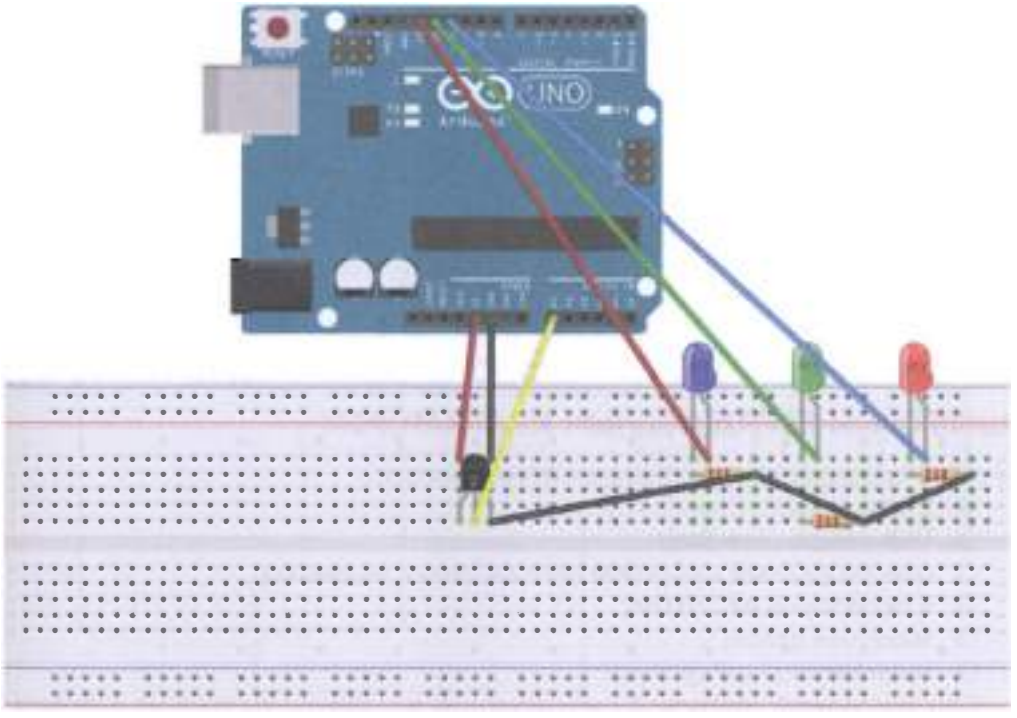


Рис. 76 ❖ Схема подключения с макетной платой для практического занятия 17

Код программы

```
   void setup () {
    pinMode (13, OUTPUT);
    pinMode (12, OUTPUT);
    pinMode (11, OUTPUT);
    Serial.begin (9600);
}
void loop () {
    int vol = analogRead (A0) * (5.0 / 1023.0 * 100); // read the LM35 temperature
    Serial.print ("Temp:") ;// output as a string represents the temperature display Tep
    Serial.print (vol) ;// output shows the value of dat
    Serial.println ("C") ;// output display as a C string
    if (vol <= 31) // set the value of the temperature of the low temperature region, and
led display
    {
        digitalWrite (13, HIGH);
        digitalWrite (12, LOW);
        digitalWrite (11, LOW);
    }
    else if (vol >= 32 && vol <= 40)
    {
        digitalWrite (13, LOW);
```

```
        digitalWrite (12, HIGH);
        digitalWrite (11, LOW);
    }
    else if (vol >= 41) // hot zone temperature setting
    {
        digitalWrite (13, LOW);
        digitalWrite (12, LOW);
        digitalWrite (11, HIGH);
    }
    delay (500);
}
```

Практическое занятие 18. Эксперимент с одноразрядным цифровым светодиодным индикатором

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- одноразрядный цифровой светодиодный индикатор;
- 8 резисторов на 220 Ом;
- макетная плата;
- соединительные провода.

Одноразрядный цифровой светодиодный индикатор состоит из 8 сегментов, каждый из которых представляет собой маленький светодиод, отвечающий за соответствующий сегмент цифры на индикаторе (или точки). Бывают два типа разноцветных светодиодов: с общим катодом (когда катодный выход один, а анодных несколько), который подключается к земле, и с общим анодом (анод один, катодов несколько), который подключается к 3.3 или 5 В выходу платы. В этом задании мы построим схему и напишем код так, чтобы индикатор показывал цифры от 1 до 8.

Схема представлена на рис. 77–78.

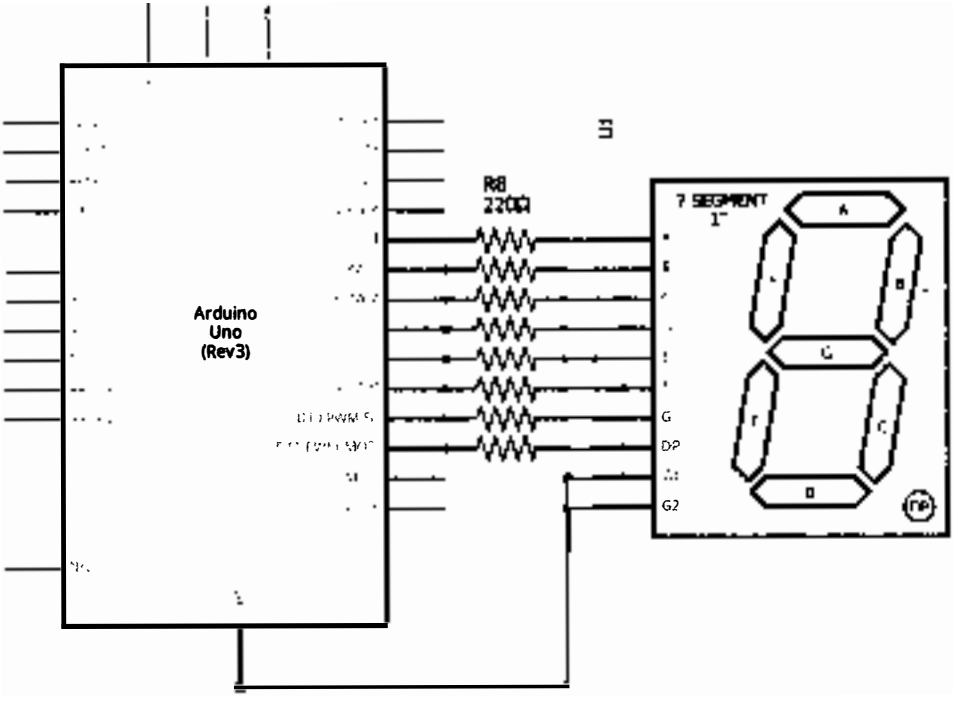


Рис. 77 ❖ Принципиальная схема подключения для практического занятия 18

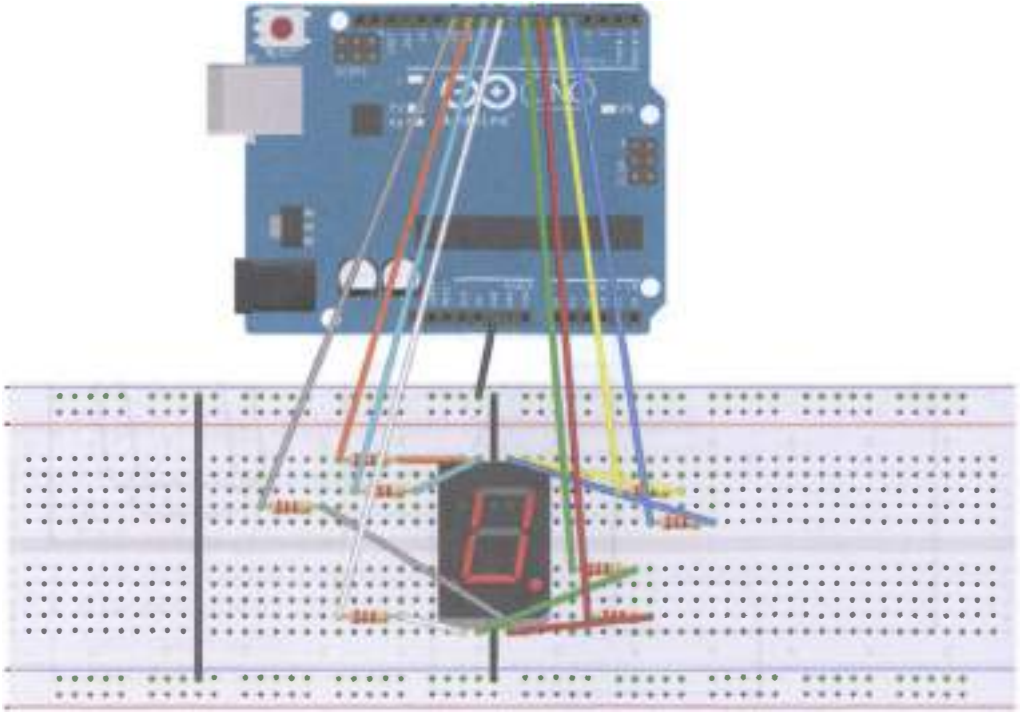


Рис. 78 ❖ Схема подключения с макетной платой для практического занятия 18

Код программы

```
// Set control each segment digital IO pin
    int a = 4 ;// define the digital interface to connect a seven segment LED
    int b = 5 ;// define the connection b Digital Interface 6-segment LED
    int c = 6 ;// define paragraph (c) Digital Interface 5 digital connection
    int d = 7 ;// define the digital interface 11 is connected to d-segment digital tube
    int e = 8 ;// define the digital interface 10 is connected to e-segment digital tube
    int f = 9 ;// define the digital interface 8 digital tube connection f
    int g = 10 ;// define the digital interface 9 g of the digital control connection
    int dp = 11 ;// define the digital interface 4 digital tube connecting dp
    void digital_1 (void) // display the number 1
    {
        digitalWrite (a, LOW);
        digitalWrite (b, HIGH);
        digitalWrite (c, HIGH);
        digitalWrite (d, LOW);
        digitalWrite (e, LOW);
        digitalWrite (f, LOW);
        digitalWrite (g, LOW);
        digitalWrite (dp, LOW);
    }
    void digital_2 (void) // display number 2
```

```
{
    digitalWrite (a, HIGH);
    digitalWrite (b, HIGH);
    digitalWrite (c, LOW);
    digitalWrite (d, HIGH);
    digitalWrite (e, HIGH);
    digitalWrite (f, LOW);
    digitalWrite (g, HIGH);
    digitalWrite (dp, LOW);
}
void digital_3 (void) // display the number 3
{
    digitalWrite (a, HIGH);
    digitalWrite (b, HIGH);
    digitalWrite (c, HIGH);
    digitalWrite (d, HIGH);
    digitalWrite (e, LOW);
    digitalWrite (f, LOW);
    digitalWrite (g, HIGH);
    digitalWrite (dp, LOW);
}
void digital_4 (void) // show 4
{
    digitalWrite (a, LOW);
    digitalWrite (b, HIGH);
    digitalWrite (c, HIGH);
    digitalWrite (d, LOW);
    digitalWrite (e, LOW);
    digitalWrite (f, HIGH);
    digitalWrite (g, HIGH);
    digitalWrite (dp, LOW);
}
void digital_5 (void) // display the number 5
{
    digitalWrite (a, HIGH);
    digitalWrite (b, LOW);
    digitalWrite (c, HIGH);
    digitalWrite (d, HIGH);
    digitalWrite (e, LOW);
    digitalWrite (f, HIGH);
    digitalWrite (g, HIGH);
    digitalWrite (dp, LOW);
}
void digital_6 (void) // display the number 6
{
    digitalWrite (a, HIGH);
    digitalWrite (b, LOW);
    digitalWrite (c, HIGH);
    digitalWrite (d, HIGH);
    digitalWrite (e, HIGH);
    digitalWrite (f, HIGH);
    digitalWrite (g, HIGH);
```

```
    digitalWrite (dp, LOW);
}
void digital_7 (void) // display the number 7
{
    digitalWrite (a, HIGH);
    digitalWrite (b, HIGH);
    digitalWrite (c, HIGH);
    digitalWrite (d, LOW);
    digitalWrite (e, LOW);
    digitalWrite (f, LOW);
    digitalWrite (g, LOW);
    digitalWrite (dp, LOW);
}
void digital_8 (void) // display the number 8
{
    digitalWrite (a, HIGH);
    digitalWrite (b, HIGH);
    digitalWrite (c, HIGH);
    digitalWrite (d, HIGH);
    digitalWrite (e, HIGH);
    digitalWrite (f, HIGH);
    digitalWrite (g, HIGH);
    digitalWrite (dp, LOW);
}
void setup ()
{
int i ;// define variables
for (i = 4; i <= 11; i++)
pinMode (i, OUTPUT) ;// set 4 to 11 pin to output mode
}
void loop ()
{
while (1)
{
digital_1 () ;// display numbers 1
delay (1000) ;// delay 2s
digital_2 () ;// display number 2
delay (1000); // delay 1s
digital_3 () ;// display the number 3
delay (1000); // delay 1s
digital_4 () ;// show 4
delay (1000); // delay 1s
digital_5 () ;// display the number 5
delay (1000); // delay 1s
digital_6 () ;// display the number 6
delay (1000); // delay 1s
digital_7 () ;// display the number 7
delay (1000); // delay 1s
digital_8 () ;// display the number 8
delay (1000); // delay 1s
}
}
```

Думаю, вам не составит особого труда **добавить ещё 2 цифры – 0 и 9!**

Практическое занятие 19. Эксперимент с четырёхразрядным цифровым светодиодным индикатором

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- четырёхразрядный цифровой светодиодный индикатор;
- 8 резисторов на 220 Ом;
- макетная плата;
- соединительные провода.

У четырёхразрядного цифрового светодиодного индикатора 12 разъёмов, по сравнению с одноразрядным, у которого 10. В принципе, не намного больше. Нумеруются они против часовой стрелки: если положить перед собой индикатор, внизу слева направо будут пины 1–6, а вверху справа налево – 7–12. Соберём схему и напишем программу, которая будет показывать значения от 0:00 до 9:00 в цикле.

Схема представлена на рис. 79–80.

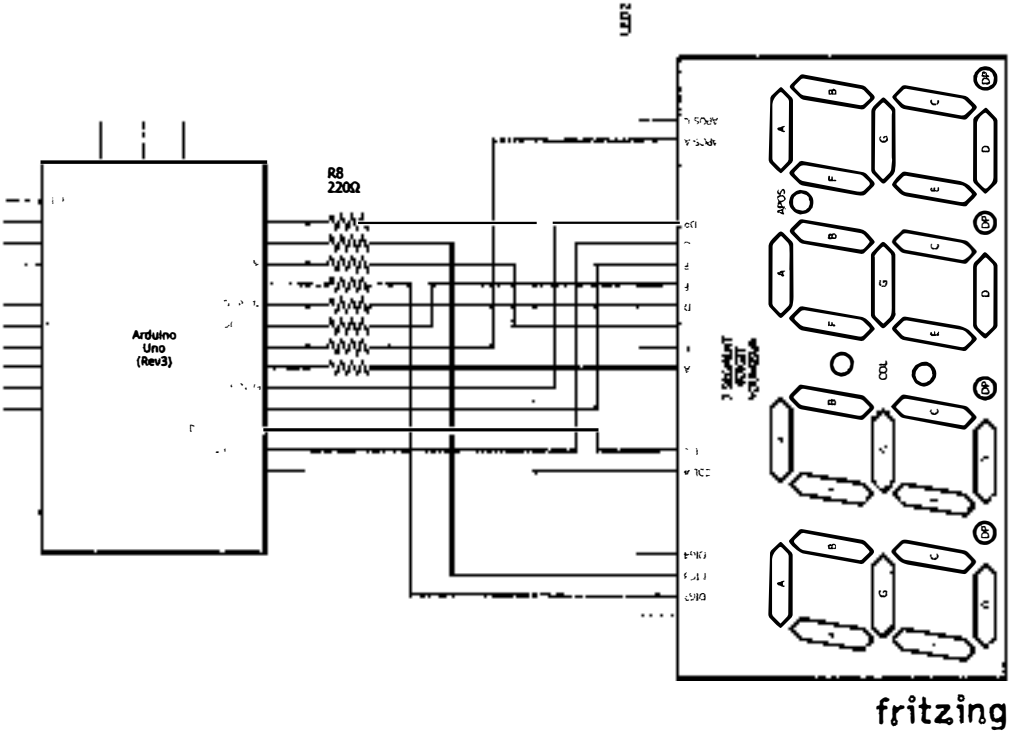


Рис. 79 ❖ Принципиальная схема подключения для практического занятия 19

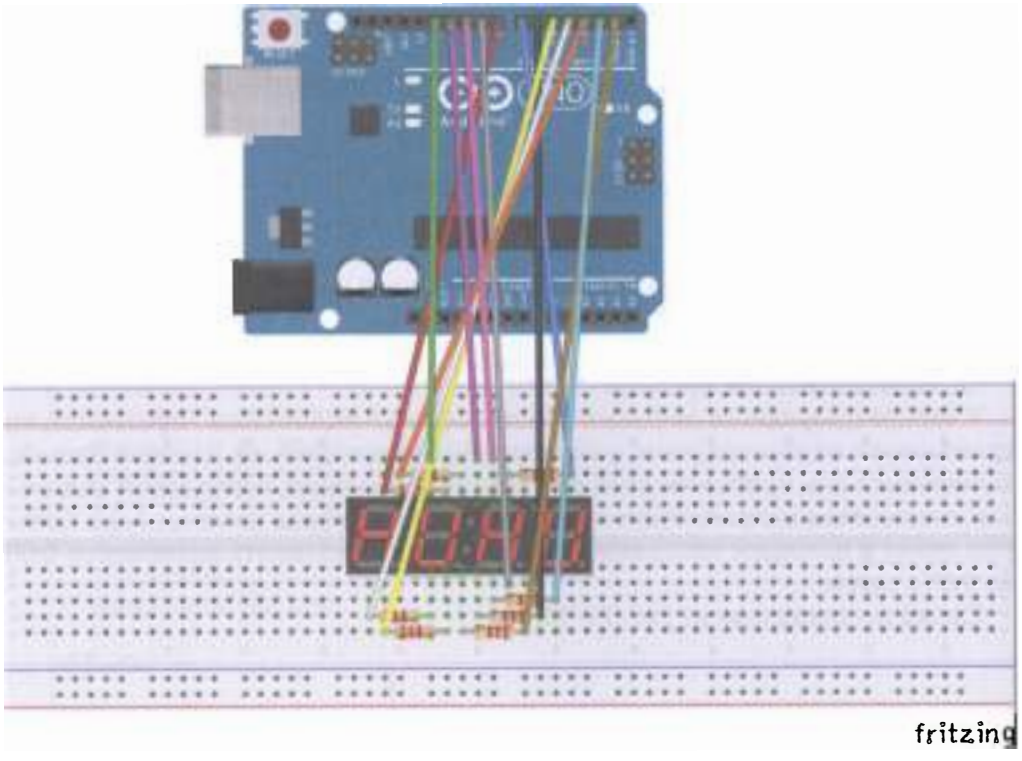


Рис. 80 ❖ Схема подключения с макетной платой для практического занятия 19

Код программы

```
int a = 8;
int b = 7;
int c = 6;
int d = 5;
int e = 4;
int f = 3;
int g = 2;
int p = 1;

int d4 = 9;
int d3 = 10;
int d2 = 11;
int d1 = 12;

long n = 0;
int x = 100;
int del = 55;

void setup()
{
    pinMode(d1, OUTPUT);
    pinMode(d2, OUTPUT);
```

```
    pinMode(d3, OUTPUT);
    pinMode(d4, OUTPUT);
    pinMode(a, OUTPUT);
    pinMode(b, OUTPUT);
    pinMode(c, OUTPUT);
    pinMode(d, OUTPUT);
    pinMode(e, OUTPUT);
    pinMode(f, OUTPUT);
    pinMode(g, OUTPUT);
    pinMode(p, OUTPUT);
}

void loop()
{
    clearLEDs();
    pickDigit(1);
    pickNumber((n/x/1000)%10);
    delayMicroseconds(del);

    clearLEDs();
    pickDigit(2);
    pickNumber((n/x/100)%10);
    delayMicroseconds(del);

    clearLEDs();
    pickDigit(3);
    dispDec(3);
    pickNumber((n/x/10)%10);
    delayMicroseconds(del);

    clearLEDs();
    pickDigit(4);
    pickNumber(n/x%10);
    delayMicroseconds(del);

    n++;

    if (digitalRead(13) == LOW)
    {
        n = 0;
    }
}

void pickDigit(int x)
{
    digitalWrite(d1, HIGH);
    digitalWrite(d2, HIGH);
    digitalWrite(d3, HIGH);
    digitalWrite(d4, HIGH);

    switch(x)
    {
        case 1:
            digitalWrite(d1, LOW);
            break;
```

```
        case 2:
            digitalWrite(d2, LOW);
            break;
        case 3:
            digitalWrite(d3, LOW);
            break;
        default:
            digitalWrite(d4, LOW);
            break;
    }
}

void pickNumber(int x)
{
    switch(x)
    {
        default:
            zero();
            break;
        case 1:
            one();
            break;
        case 2:
            two();
            break;
        case 3:
            three();
            break;
        case 4:
            four();
            break;
        case 5:
            five();
            break;
        case 6:
            six();
            break;
        case 7:
            seven();
            break;
        case 8:
            eight();
            break;
        case 9:
            nine();
            break;
    }
}

void dispDec(int x)
{
    digitalWrite(p, LOW);
}
```

```
void clearLEDs()
{
    digitalWrite(a, LOW);
    digitalWrite(b, LOW);
    digitalWrite(c, LOW);
    digitalWrite(d, LOW);
    digitalWrite(e, LOW);
    digitalWrite(f, LOW);
    digitalWrite(g, LOW);
    digitalWrite(p, LOW);
}

void zero()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);
    digitalWrite(f, HIGH);
    digitalWrite(g, LOW);
}

void one()
{
    digitalWrite(a, LOW);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, LOW);
    digitalWrite(e, LOW);
    digitalWrite(f, LOW);
    digitalWrite(g, LOW);
}

void two()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, LOW);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);
    digitalWrite(f, LOW);
    digitalWrite(g, HIGH);
}

void three()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, LOW);
    digitalWrite(f, LOW);
    digitalWrite(g, HIGH);
```

```
}
void four()
{
    digitalWrite(a, LOW);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, LOW);
    digitalWrite(e, LOW);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
}
void five()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, LOW);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, LOW);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
}
void six()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, LOW);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
}
void seven()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, LOW);
    digitalWrite(e, LOW);
    digitalWrite(f, LOW);
    digitalWrite(g, LOW);
}
void eight()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);
    digitalWrite(f, HIGH);
```

```
    digitalWrite(g, HIGH);
}
void nine()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, LOW);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
}
```

Практическое занятие 20. Эксперимент со светодиодной матрицей

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- светодиодная матрица 8×8;
- 8 резисторов на 220 Ом;
- макетная плата;
- соединительные провода.

Светодиодная матрица 8×8 содержит 64 светодиода, соединённых так, чтобы можно было зажечь соответствующий светодиод, пользуясь его *X*- и *У*-координатой и подавая, соответственно, на одну координату землю через резистор, а на другую – управляющий сигнал с одного из цифровых пинов-выходов платы Arduino Uno. К сожалению, в parts-библиотеке Fritzing не нашлось соответствующего элемента, поэтому пришлось пририсовывать светодиодной матрице 8×8 с 12 выходами ещё 4 на схеме с макетной платой и рисовать с нуля принципиальную схему. Напишем два кода: первый будет просто зажигать и гасить верхний левый светодиод на матрице, второй – выводить буквы английского алфавита от A до I.

Схема представлена на рис. 81–82.

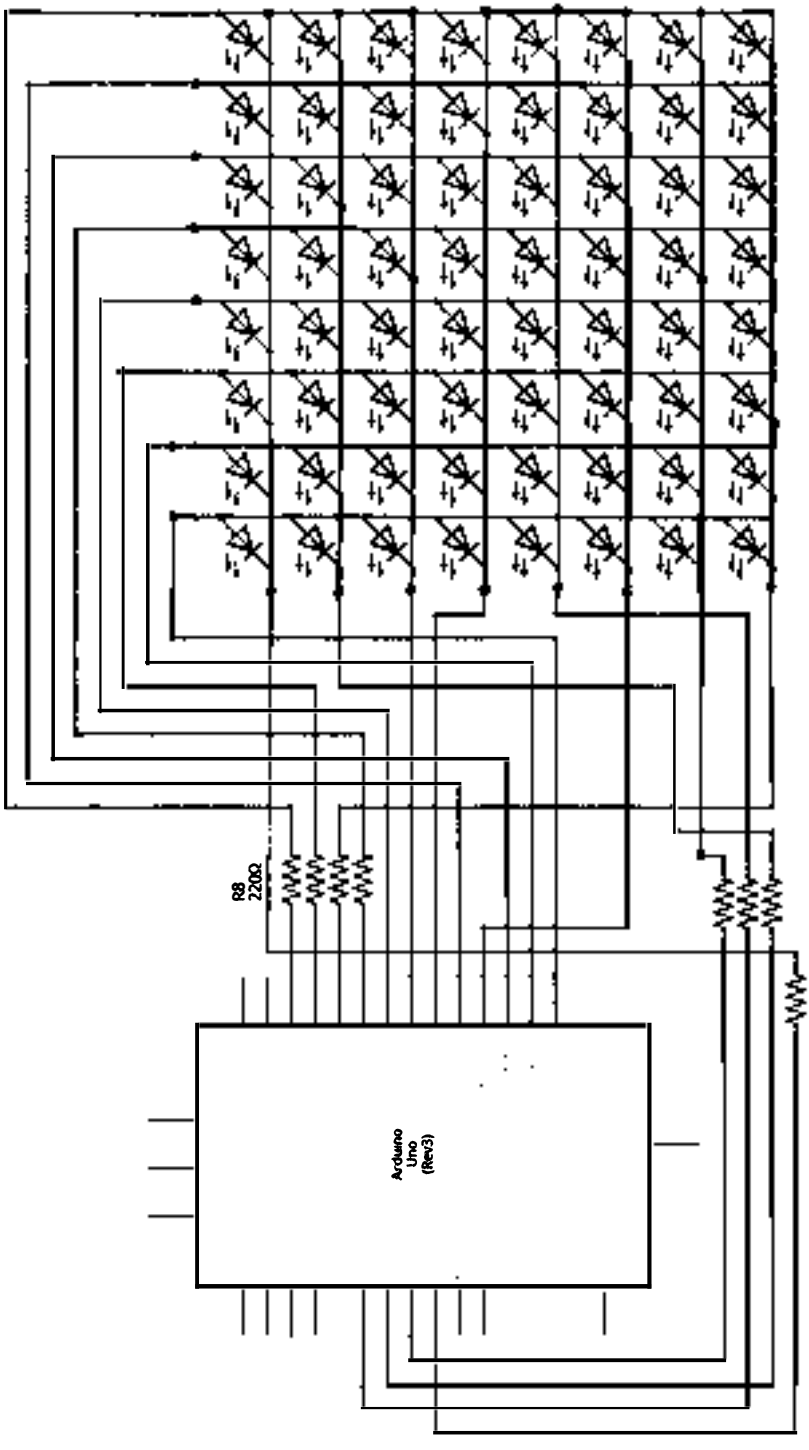


Рис. 81 ❖ Принципиальная схема подключения для практического занятия 20

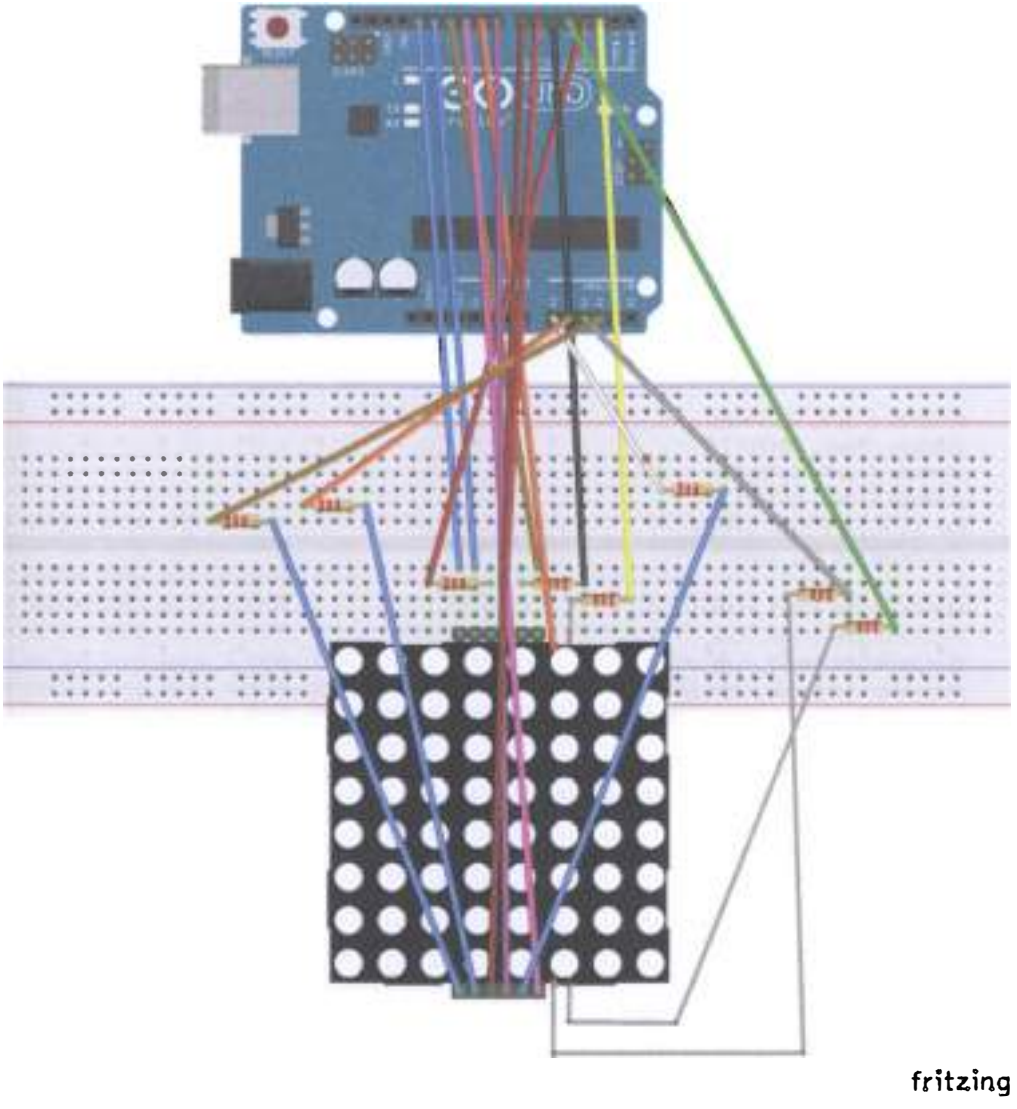


Рис. 82 ❖ Схема подключения с макетной платой для практического занятия 20

Код программы 1

```
const int row1 = 2; // the number of the row pin 9
const int row2 = 3; // the number of the row pin 14
const int row3 = 4; // the number of the row pin
const int row4 = 5; // the number of the row pin 12
const int row5 = 17; // the number of the row pin 1
const int row6 = 16; // the number of the row pin 7
const int row7 = 15; // the number of the row pin 2
const int row8 = 14; // the number of the row pin 5
```

```
// The pin to control COl
const int col1 = 6; // the number of the col pin 13
const int col2 = 7; // the number of the col pin 3
const int col3 = 8; // the number of the col pin 4
const int col4 = 9; // the number of the col pin 10
const int col5 = 10; // the number of the col pin 6
const int col6 = 11; // the number of the col pin 11
const int col7 = 12; // the number of the col pin 15
const int col8 = 13; // the number of the col pin 16
void setup () {
int i = 0;
for (i = 2; i<18; i++)
    {
    pinMode (i, OUTPUT);
    }
    pinMode (row5, OUTPUT);
    pinMode (row6, OUTPUT);
    pinMode (row7, OUTPUT);
    pinMode (row8, OUTPUT);
    for (i = 2; i <18; i++) {
    digitalWrite (i, LOW);
    }
    digitalWrite (row5, LOW);
    digitalWrite (row6, LOW);
    digitalWrite (row7, LOW);
    digitalWrite (row8, LOW);
    }
    void loop () {
    int i;
    // The row # 1 and col # 1 of the LEDs turn on
    digitalWrite (row1, HIGH);
    digitalWrite (row2, LOW);
    digitalWrite (row3, LOW);
    digitalWrite (row4, LOW);
    digitalWrite (row5, LOW);
    digitalWrite (row6, LOW);
    digitalWrite (row7, LOW);
    digitalWrite (row8, LOW);
    digitalWrite (col1, LOW);
    digitalWrite (col2, HIGH);
    digitalWrite (col3, HIGH);
    digitalWrite (col4, HIGH);
    digitalWrite (col5, HIGH);
    digitalWrite (col6, HIGH);
    digitalWrite (col7, HIGH);
    digitalWrite (col8, HIGH);
    delay (1000);
    // Turn off all
    for (i = 2; i <18; i++) {
    digitalWrite (i, LOW);
    }
    delay (1000);
    }
```

Код программы 2

```
# define display_array_size 8
// Ascii 8x8 dot font
# define data_null 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 // null char
# define data_ascii_A 0x02, 0x0C, 0x18, 0x68, 0x68, 0x18, 0x0C, 0x02 /* «A», 0 */
/**
** «A»
# Define A {//
{0, 0, 0, 0, 0, 0, 1, 0}, / / 0x02
{0, 0, 0, 0, 1, 1, 0, 0}, / / 0x0C
{0, 0, 0, 1, 1, 0, 0, 0}, / / 0x18
{0, 1, 1, 0, 1, 0, 0, 0}, / / 0x68
{0, 1, 1, 0, 1, 0, 0, 0}, / / 0x68
{0, 0, 0, 1, 1, 0, 0, 0}, / / 0x18
{0, 0, 0, 0, 1, 1, 0, 0}, / / 0x0C
{0, 0, 0, 0, 0, 0, 1, 0} / / 0x02
}
**/
# define data_ascii_B 0x00, 0x7E, 0x52, 0x52, 0x52, 0x52, 0x2C, 0x00 /* «B», 1 */
# define data_ascii_C 0x00, 0x3C, 0x66, 0x42, 0x42, 0x42, 0x2C, 0x00 /* «C», 2 */
# define data_ascii_D 0x00, 0x7E, 0x42, 0x42, 0x42, 0x66, 0x3C, 0x00 /* «D», 3 */
# define data_ascii_E 0x00, 0x7E, 0x52, 0x52, 0x52, 0x52, 0x52, 0x42 /* «E», 4 */
# define data_ascii_F 0x00, 0x7E, 0x50, 0x50, 0x50, 0x50, 0x50, 0x40 /* «F», 5 */
# define data_ascii_G 0x00, 0x3C, 0x66, 0x42, 0x42, 0x52, 0x16, 0x1E /* «G», 6 */
# define data_ascii_H 0x00, 0x7E, 0x10, 0x10, 0x10, 0x10, 0x7E, 0x00 /* «H», 7 */
# define data_ascii_I 0x00, 0x00, 0x00, 0x7E, 0x00, 0x00, 0x00, 0x00 /* «I», 8 */
// Display array
byte data_ascii [] [display_array_size] = {
data_null,
data_ascii_A, data_ascii_B,
data_ascii_C,
data_ascii_D,
data_ascii_E,
data_ascii_F,
data_ascii_G,
data_ascii_H,
data_ascii_I,
};
// The pin to control ROW
const int row1 = 2; // the number of the row pin 24
const int row2 = 3; // the number of the row pin 23
const int row3 = 4; // the number of the row pin 22
const int row4 = 5; // the number of the row pin 21
const int row5 = 17; // the number of the row pin 4
const int row6 = 16; // the number of the row pin 3
const int row7 = 15; // the number of the row pin 2
const int row8 = 14; // the number of the row pin 1
// The pin to control COl
const int col1 = 6; // the number of the col pin 20
const int col2 = 7; // the number of the col pin 19
const int col3 = 8; // the number of the col pin 18
const int col4 = 9; // the number of the col pin 17
```

```
const int col5 = 10; // the number of the col pin 16
const int col6 = 11; // the number of the col pin 15
const int col7 = 12; // the number of the col pin 14
const int col8 = 13; // the number of the col pin 13
void displayNum (byte rowNum, int colNum)
  {
     int j;
     byte temp = rowNum;
     for (j = 2; j <6; j++)
       {
          digitalWrite (j, LOW);
       }
    digitalWrite (row5, LOW);
    digitalWrite (row6, LOW);
    digitalWrite (row7, LOW);
    digitalWrite (row8, LOW);
    for (j = 6; j <14; j++)
      {
        digitalWrite (j, HIGH);}
        switch (colNum)
      {
    case 1: digitalWrite (col1, LOW); break;
    case 2: digitalWrite (col2, LOW); break;
    case 3: digitalWrite (col3, LOW); break;
    case 4: digitalWrite (col4, LOW); break;
    case 5: digitalWrite (col5, LOW); break;
    case 6: digitalWrite (col6, LOW); break;
    case 7: digitalWrite (col7, LOW); break;
    case 8: digitalWrite (col8, LOW); break;
    default: break;
    }
    for (j = 1; j <9; j++)
      {
       temp = (0x80) & (temp);
       if (temp == 0)
         {
           temp = rowNum << j;
           continue;
         }
        switch (j)
        {
         case 1: digitalWrite (row1, HIGH); break;
         case 2: digitalWrite (row2, HIGH); break;
         case 3: digitalWrite (row3, HIGH); break;
         case 4: digitalWrite (row4, HIGH); break;
         case 5: digitalWrite (row5, HIGH); break;
         case 6: digitalWrite (row6, HIGH); break;
         case 7: digitalWrite (row7, HIGH); break;
         case 8: digitalWrite (row8, HIGH); break;
         default: break;
        }
      temp = rowNum << j;
```

```
  }
}

void setup () {
int i = 0;
for (i = 2; i <18; i++)
{
pinMode (i, OUTPUT);
}
for (i = 2; i <18; i++) {
digitalWrite (i, LOW);
}
}
void loop () {
int t1;
int l;
int arrage;
for (arrage = 0; arrage <10; arrage++)
{
for (l = 0; l <512; l++)
  {
    for (t1 = 0; t1 <8; t1++)
      {
        displayNum (data_ascii [arrage] [t1], (t1 +1));
      }
   }
}
}
```

Практическое занятие 21. Эксперимент с трёхцветным светодиодом

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- трёхцветный светодиод;
- 3 резистора на 220 Ом;
- макетная плата;
- соединительные провода.

В этом занятии мы сначала выведем красный, белый и синий цвета с помощью трёхцветного (RGB) светодиода, а затем выведем смешанные цвета. У этого светодиода 4 выхода: 3 анода (красный, жёлтый, синий цвета) и 1 катод (на землю). Такой светодиод называется светодиодом с общим катодом. Есть также светодиод с общим анодом, когда трём цветам соответствуют 3 катодных выхода. В комплекте Arduino Uno Starter Learning Kit с RFID-модулем нет отдельного трёхцветного светодиода, но есть небольшой модуль со встроенным трёхцветным светодиодом, который мы и будем использовать. Вообще,

на модуле уже есть встроенные сопротивления, но на всякий случай лучше всё-таки использовать дополнительные внешние резисторы на 220 Ом. Обратите внимание на маркировку пинов на модуле с трёхцветным светодиодом – она может отличаться от приведённой ниже схемы подключения (например: R, G, B подключаются, соответственно, к 11, 9, 10 и GND пинам платы)!

Схема представлена на рис. 83–84.

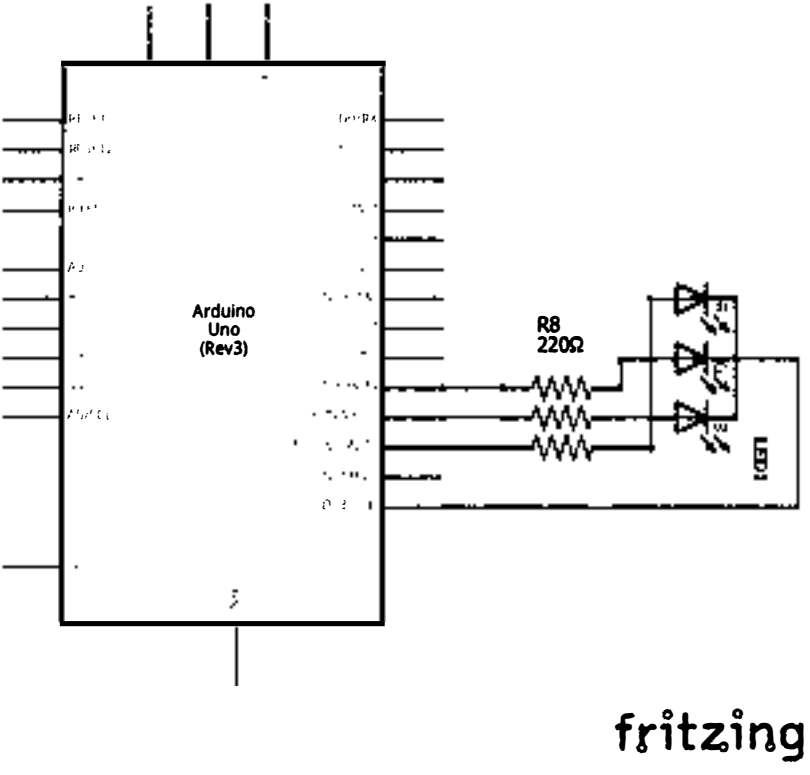


Рис. 83 ❖ Принципиальная схема подключения для практического занятия 21

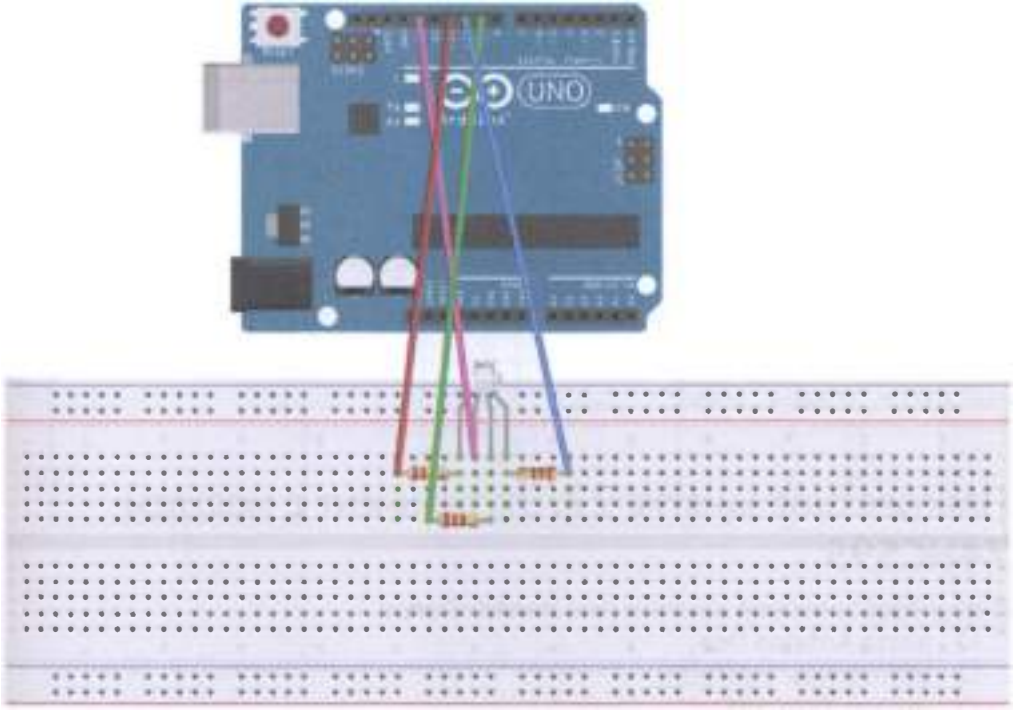


Рис. 84 ❖ Схема подключения с макетной платой для практического занятия 21

Код программы

```
int ledPin = 13; // LED is connected to digital pin 13
int redPin = 11; // R petal on RGB LED module connected to digital pin 11
int greenPin = 9; // G petal on RGB LED module connected to digital pin 9
int bluePin = 10; // B petal on RGB LED module connected to digital pin 10
void setup ()
{
pinMode (ledPin, OUTPUT); // sets the ledPin to be an output
pinMode (redPin, OUTPUT); // sets the redPin to be an output
pinMode (greenPin, OUTPUT); // sets the greenPin to be an output
pinMode (bluePin, OUTPUT); // sets the bluePin to be an output
}
void loop () // run over and over again
{
// Basic colors:
color (255, 0, 0); // turn the RGB LED red
delay (1000); // delay for 1 second
color (0,255, 0); // turn the RGB LED green
delay (1000); // delay for 1 second
color (0, 0, 255); // turn the RGB LED blue
delay (1000); // delay for 1 second
// Example blended colors:
```

```
color (255,255,0); // turn the RGB LED yellow
delay (1000); // delay for 1 second
color (255,255,255); // turn the RGB LED white
delay (1000); // delay for 1 second
color (128,0,255); // turn the RGB LED purple
delay (1000); // delay for 1 second
color (0,0,0); // turn the RGB LED off
delay (1000); // delay for 1 second
}
void color (unsigned char red, unsigned char green, unsigned char blue) // the color
generating function
{
digitalWrite (redPin, red);
digitalWrite (bluePin, blue);
digitalWrite (greenPin, green);
}
```

Практическое занятие 22. Эксперимент с модулем 74HC595

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- модуль 74HC595;
- 8 светодиодов;
- 8 резисторов на 220 Ом;
- макетная плата;
- соединительные провода.

Зачем использовать модуль 74HC595, у которого есть сдвиговые регистры и память, для работы 8 светодиодов? Дело в том, что на любой плате есть ограниченное количество входов и выходов (пинов). Для работы 8 светодиодов нам надо занять 9 пинов платы (вместе с землёй). Рассматриваемый модуль сократит количество задействованных пинов до 5. **Будьте внимательны, подключайте модуль правильно, иначе плата будет перегреваться и выключаться.**

Схема представлена на рис. 85–86.

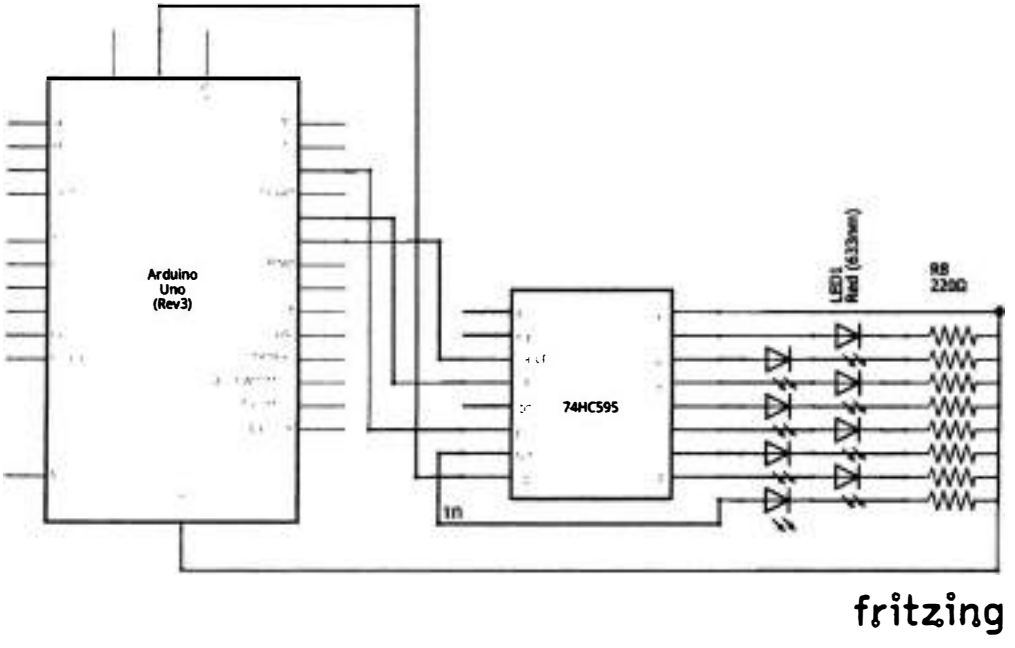


Рис. 85 ❖ Принципиальная схема подключения для практического занятия 22

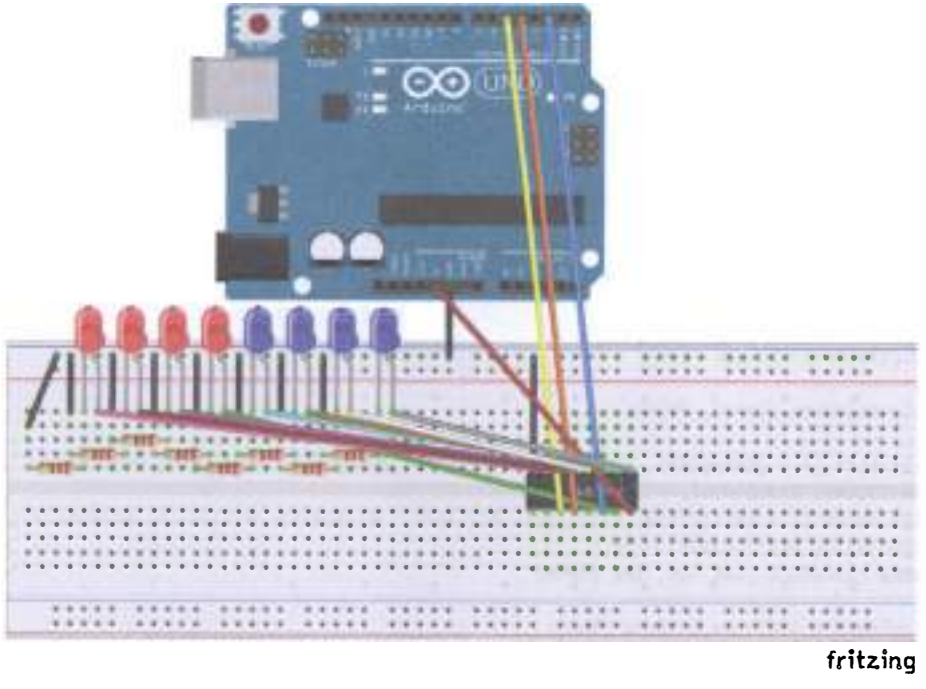


Рис. 86 ❖ Схема подключения с макетной платой для практического занятия 22

Код программы

```
int data = 2;
int clock = 4;
int latch = 5;
int ledState = 0;
const int ON = HIGH;
const int OFF = LOW;
void setup ()
{
    pinMode (data, OUTPUT);
    pinMode (clock, OUTPUT);
    pinMode (latch, OUTPUT);
}
void loop ()
    {
    int delayTime = 100;
    for (int i = 0; i <256; i++)
    {
    updateLEDs (i);
    delay (delayTime);
    }
    }
    void updateLEDs (int value)
    {
    digitalWrite (latch, LOW);
    shiftOut (data, clock, MSBFIRST, value);
    digitalWrite (latch, HIGH);
    }
    void updateLEDsLong (int value)
    {
    digitalWrite (latch, LOW);
    for (int i = 0; i <8; i++)
    {
    int bit = value & B10000000;
    value = value << 1;
    if (bit == 128) {digitalWrite (data, HIGH);}
    else {digitalWrite (data, LOW);}
    digitalWrite (clock, HIGH);
    delay (1);
    digitalWrite (clock, LOW);
    }
    digitalWrite (latch, HIGH);
    }
    int bits [] = {B00000001, B00000010, B00000100, B00001000, B00010000, B00100000,
    B01000000, B10000000};
    int masks [] = {B11111110, B11111101, B11111011, B11110111, B11101111, B11011111,
    B10111111, B01111111};
    void changeLED (int led, int state)
    {
    ledState = ledState & masks [led];
    if (state == ON) {ledState = ledState | bits [led];}
    updateLEDs (ledState);
    }
```

Практическое занятие 23. Кнопочный модуль 4×4 и библиотеки

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- кнопочный модуль 4×4;
- соединительные провода.

В этом занятии мы будем использовать кнопочный модуль 4×4 с 16 кнопками. В библиотеке компонентов Fritzing есть похожий компонент Buttonpad-4×4 в разделе Sparkfun-Electromechanical, однако внутри кнопки никак не соединены между собой и присутствуют светодиоды, которые так же висят в воздухе, как и кнопки, и вообще нам не нужны; поэтому для принципиальной схемы пришлось соединять кнопки между собой и выводить необходимые в занятии 8 выходов компонента. Также нам понадобится консоль, или монитор порта (**Ctrl+Shift+M**), чтобы смотреть результаты нажатия кнопок. Кроме того, нам понадобится библиотека Keypad.h, которую можно скачать с официального сайта Arduino по адресу [5]. Чтобы добавить библиотеку в проект Arduino IDE, нужно выбрать пункт меню **Скетч – Подключить библиотеку – Добавить .ZIP библиотеку...**, выбрать скачанный архив Keypad.zip и затем снова нажать **Скетч – Подключить библиотеку**, после чего выбрать библиотеку Keypad – она будет внизу списка. При этом в начало кода добавляется строка `#include <Keypad.h>`. Первая программа выводит символы в соответствии с нажатой кнопкой, вторая – задействует встроенный мини-светодиод на плате, который горит, пока нажата кнопка, ответственная за знак `'*'`, и загорается или гаснет по нажатии кнопки, ответственной за символ `'#'`.

Схема представлена на рис. 87–88.

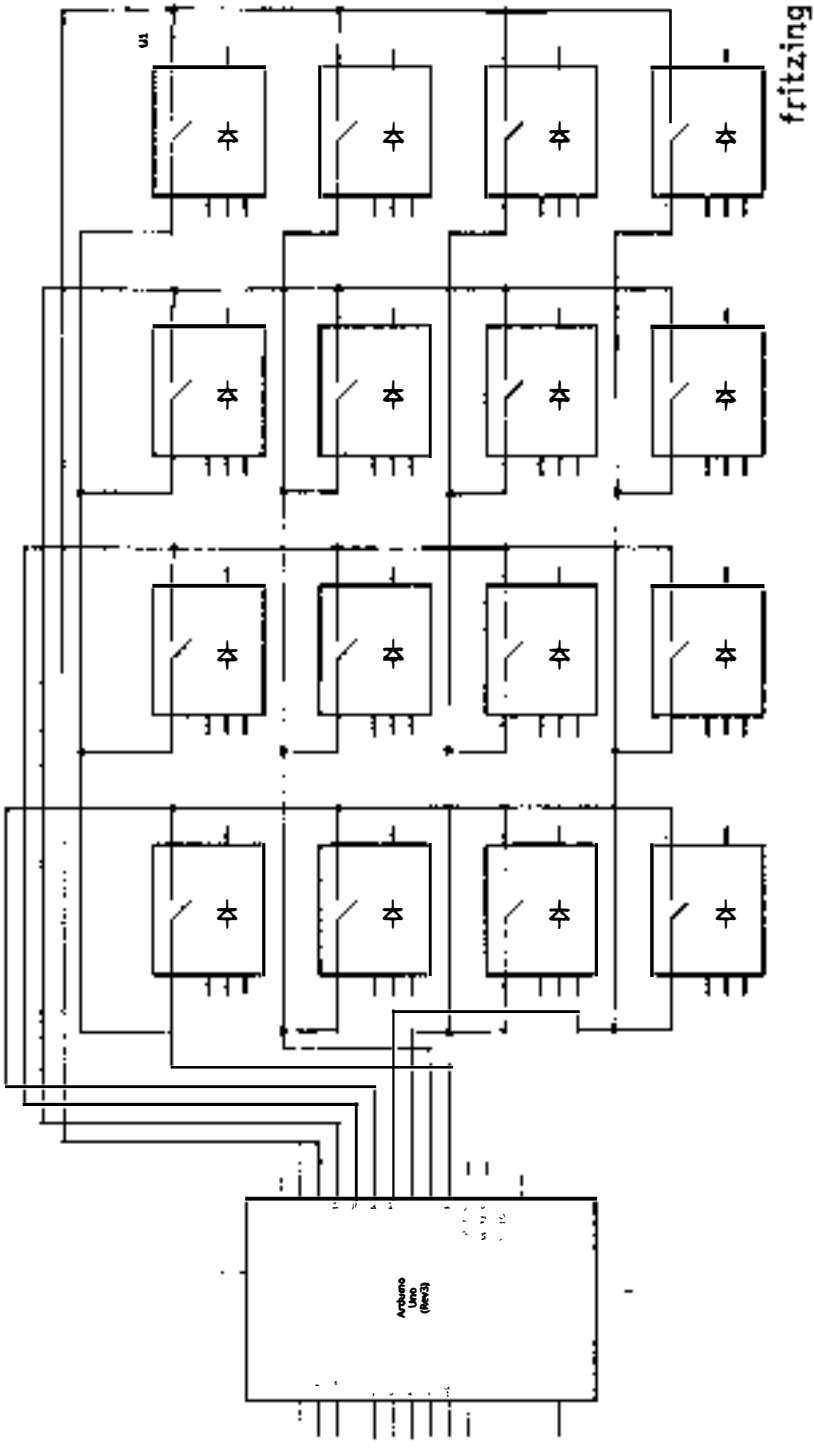


Рис. 87 ❖ Принципиальная схема подключения для практического занятия 23

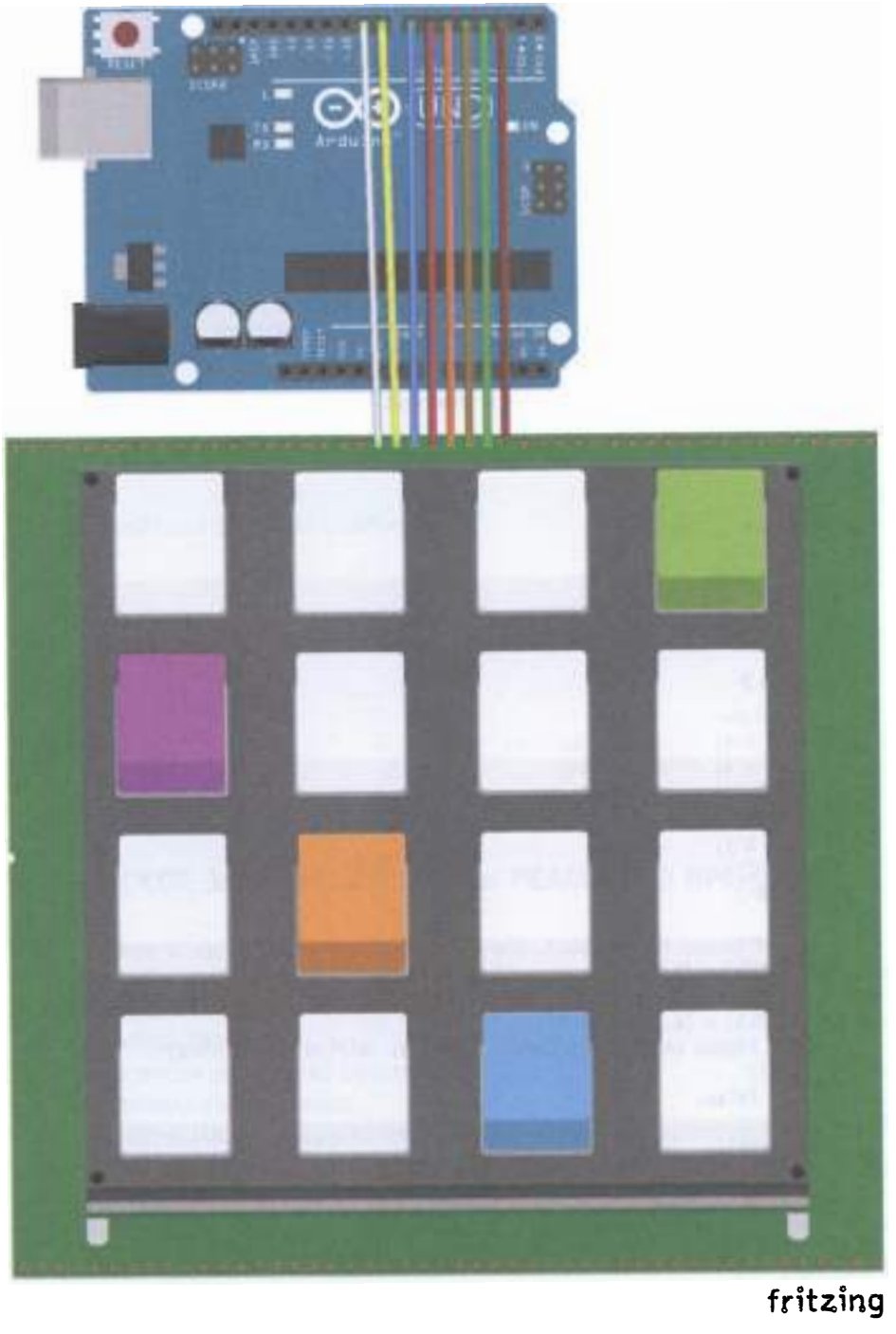


Рис. 88 ❖ Схема подключения с макетной платой для практического занятия 23

Код программы 1

```
#include <Keypad.h>
const byte ROWS = 4; // define four rows
const byte COLS = 4; // define four
char keys [ROWS] [COLS] = {
{'1','2','3','A'},
{'4','5','6','B'},
{'7','8','9','C'},
{'*','0','#','D'}
};
// Connect 4 * 4 keypad row-bit port, the corresponding digital IO ports panel
byte rowPins [ROWS] = {2,3,4,5};
// Connect 4 * 4 buttons faithfully port, the corresponding digital IO ports panel
byte colPins [COLS] = {6,7,8,9};
// Call the function library function Keypad
Keypad keypad = Keypad (makeKeymap (keys), rowPins, colPins, ROWS, COLS);
void setup () {
Serial.begin (9600);
}
void loop () {
char key = keypad.getKey ();
if (key!= NO_KEY) {
Serial.println (key);
}
}
```

Код программы 2

```
#include <Keypad.h>
const byte ROWS = 4; // define four rows
const byte COLS = 4; // define four
char keys [ROWS] [COLS] = {
{'1', '2', '3','A'},
{'4', '5', '6','B'},
{'7', '8', '9','C'},
{'*', '0','#','D'}
};
// Connect 4 * 4 keypad row-bit port, the corresponding digital IO ports panel
byte rowPins [ROWS] = {2,3,4,5};
// Connect 4 * 4 buttons faithfully port, the corresponding digital IO ports panel
byte colPins [COLS] = {6,7,8,9};
Keypad keypad = Keypad (makeKeymap (keys), rowPins, colPins, ROWS, COLS);
byte ledPin = 13;
boolean blink = false;
void setup () {
Serial.begin (9600);
pinMode (ledPin, OUTPUT); // sets the digital pin as output
digitalWrite (ledPin, HIGH); // sets the LED on
keypad.addEventListener (keypadEvent); // add an event listener for this keypad
}
void loop () {
char key = keypad.getKey ();
if (key!= NO_KEY) {
Serial.println (key);
```

```
}
if (blink) {
digitalWrite (ledPin,! digitalRead (ledPin));
delay (100);
}
}
// Take care of some special events
void keypadEvent (KeypadEvent key) {
switch (keypad.getState ()) {
case PRESSED:
switch (key) {
case '#': digitalWrite (ledPin,!digitalRead (ledPin)); break;
case '*':
digitalWrite (ledPin,!digitalRead (ledPin));
break;
}
break;
case RELEASED:
switch (key) {
case '*':
digitalWrite (ledPin,!digitalRead (ledPin));
blink = false;
break;
}
break;
case HOLD:
switch (key) {
case '*': blink = true; break;
}
break;
}
}
```

Практическое занятие 24. Часы реального времени DS1307

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- модуль часов реального времени;
- соединительные провода.

В этом занятии мы будем работать с модулем часов реального времени (RTC module DS1307). Он поддерживает отображение года, месяца, дня, дня недели, часа, минуты и секунды. Нам понадобятся: консоль, или монитор порта (**Ctrl+Shift+M**), чтобы видеть время, выдаваемое модулем, и следующие библиотеки: stdio.h, string.h (они установлены по умолчанию) и DS1302.h, которую можно найти в [6]. Она вполне подходит для работы с модулем DS1307, хотя для него можно отдельно скачать библиотеку через **Скетч – Подключить библиотеку – Управлять библиотеками...** главного меню Adruino IDE и применить

код из примеров именно для него. Когда вы запустите программу, в консоль наверняка будет выводиться что-то типа Sunday 2000-1-1 1:11:17, т. е. часы немного отстали. Однако это можно исправить, отправив в текстовое поле корректные дату, время и день недели в следующем формате: год,месяц,день,час, минута,секунда,день недели, при этом день недели задаётся в виде числа, где воскресенье (Sunday) = 1, понедельник (Monday) = 2 и т. д. Например, можно ввести так: 2017,11,29,0,17,59,4 (откуда вы узнаете, когда я делал это задание). При этом после ввода вы увидите запись в консоли: You inputed:2017,11,29,0,17,59,4, после которой пойдёт отображение правильной даты и времени в соответствии с тем, что было введено: Wednesday 2017-11-29 0:17:59. Вы можете отключить на время модуль, а потом включить его снова и убедиться, что часы работают (потому что в них есть батарейка, рассчитанная на 9 лет работы). Такие же часы установлены на материнской плате любого компьютера. Обратите внимание на маркировку пинов на модуле с часов – она может отличаться от приведённой ниже схемы подключения!

Схема представлена на рис. 89–90.

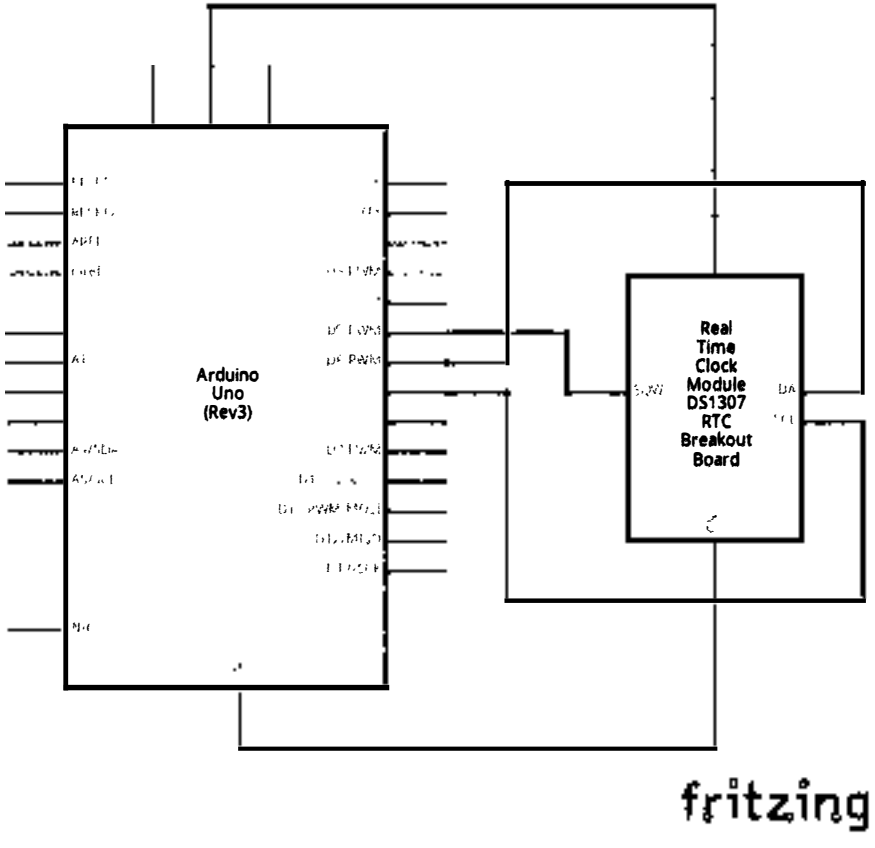


Рис. 89 ❖ Принципиальная схема подключения для практического занятия 24

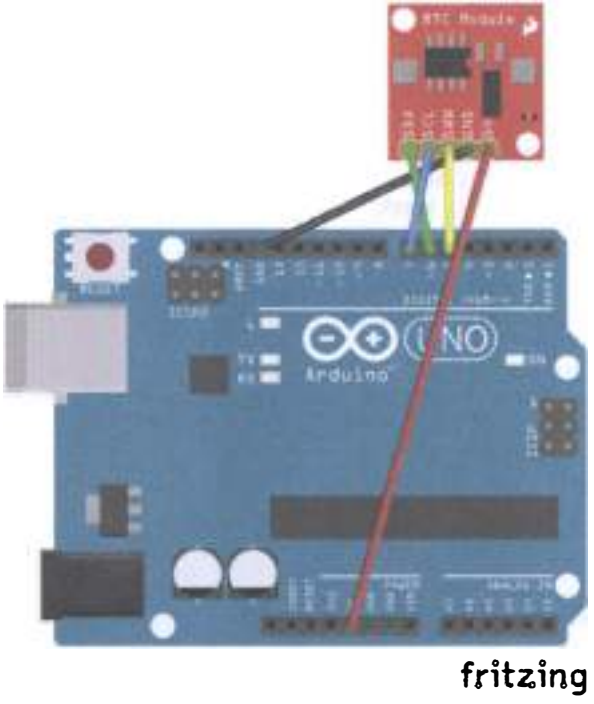


Рис. 90 ❖ Схема подключения для практического занятия 24

Код программы

```
#include <DS1302.h>
#include <stdio.h>
#include <string.h>

/* Interface Definition
CE (DS1302 pin5) -> Arduino D5
IO (DS1302 pin6) -> Arduino D6
SCLK (DS1302 pin7) -> Arduino D7
*/
uint8_t CE_PIN = 5;
uint8_t IO_PIN = 6;
uint8_t SCLK_PIN = 7;
/* Date variable buffer */
char buf [50];
char day [10];
/* Serial data cache */
String comdata = "";
int numdata [7] = {0}, j = 0, mark = 0;
/* Create DS1302 object */
DS1302 rtc (CE_PIN, IO_PIN, SCLK_PIN);
void print_time ()
{
/* Get the current time from the DS1302 */
```

```
Time t = rtc.time ();
/* The week from digital to name */
memset (day, 0, sizeof (day));
switch (t.day)
{
case 1: strcpy (day, "Sunday"); break;
case 2: strcpy (day, "Monday"); break;
case 3: strcpy (day, "Tuesday"); break;
case 4: strcpy (day, "Wednesday"); break;
case 5: strcpy (day, "Thursday"); break;
case 6: strcpy (day, "Friday"); break;
case 7: strcpy (day, "Saturday"); break;
}
/* The date code format to make up for output buf */
snprintf (buf, sizeof (buf), "% s% 04d-% 02d-% 02d% 02d:% 02d:% 02d", day, t.yr, t.mon,
t.date, t.hr, t. min, t.sec);
/* Output the date to the serial port */
Serial.println (buf);
}
void setup ()
{
Serial.begin (9600);
rtc.writeProtect (false);
rtc.halt (false);
}
void loop ()
{
/* When the serial port has data, the data is spliced into a variable comdata */
while(Serial.available()>0)
{
comdata+=char(Serial.read());
delay(2);
mark=1;
}
/* A comma-separated string comdata decomposition, the decomposition results become
converted into digital to numdata [] array */
if(mark==1)
{
Serial.print("You inputed:");
Serial.println(comdata);
for(int i=0;i<comdata.length();i++)
{
if(comdata[i]==','||comdata[i]==0x10||comdata[i]==0x13)
{
j++;
}
else
{
numdata[j]=numdata[j]*10+(comdata[i]-'0');
}
}
/* Make up the converted numdata time format, write DS1302 */
```

```
Time t(numdata[0],numdata[1],numdata[2],numdata[3],numdata[4],numdata[5],
numdata[6]);
rtc.time(t);
mark=0;j=0;
/* Empty comdata variable to wait for the next input */
comdata=String("");
/* Empty numdata */
for(int i=0;i<7;i++)numdata[i]=0;
}
/* Print the current time */
print_time();
delay(1000);
}
```

Практическое занятие 25. Эксперимент с датчиком уровня воды

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- датчик уровня воды;
- красный светодиод;
- резистор на 220 Ом;
- макетная плата;
- соединительные провода.

Датчик уровня воды (Water Sensor) определяет количество (точнее, уровень) воды, гибок в применении, высокочувствителен и потребляет мало энергии, а также поддерживается многими контроллерами и платами расширения/разработчика. Однако в библиотеке Fritzing его почему-то нет, поэтому в схемах пришлось воспользоваться датчиком влажности почвы (Soil Moisture Sensor) из набора элементов (parts) SparkFun-Sensors, который обладает такими же тремя выходами и, в принципе, решает те же задачи. В этом задании мы будем зажигать светодиод, подключённый к цифровому выходу 12 платы Arduino Uno, если уровень воды превышает определённое пороговое значение (550), выдаваемое на аналоговый порт A1 платы. Кроме того, нам понадобится ёмкость с водой, куда мы будем погружать (**только не до конца!**) датчик уровня воды. Не бойтесь погрузить датчик в воду на несколько секунд, бойтесь погрузить его не тем концом, или тем, но – по уровень надписи Water Sensor и глубже, или бойтесь оставить его в воде на долгое время. И ещё нам надо открыть консоль (монитор порта), чтобы видеть показания датчика. Не забудьте вытереть датчик после выполнения задания!

Схема представлена на рис. 91–92.

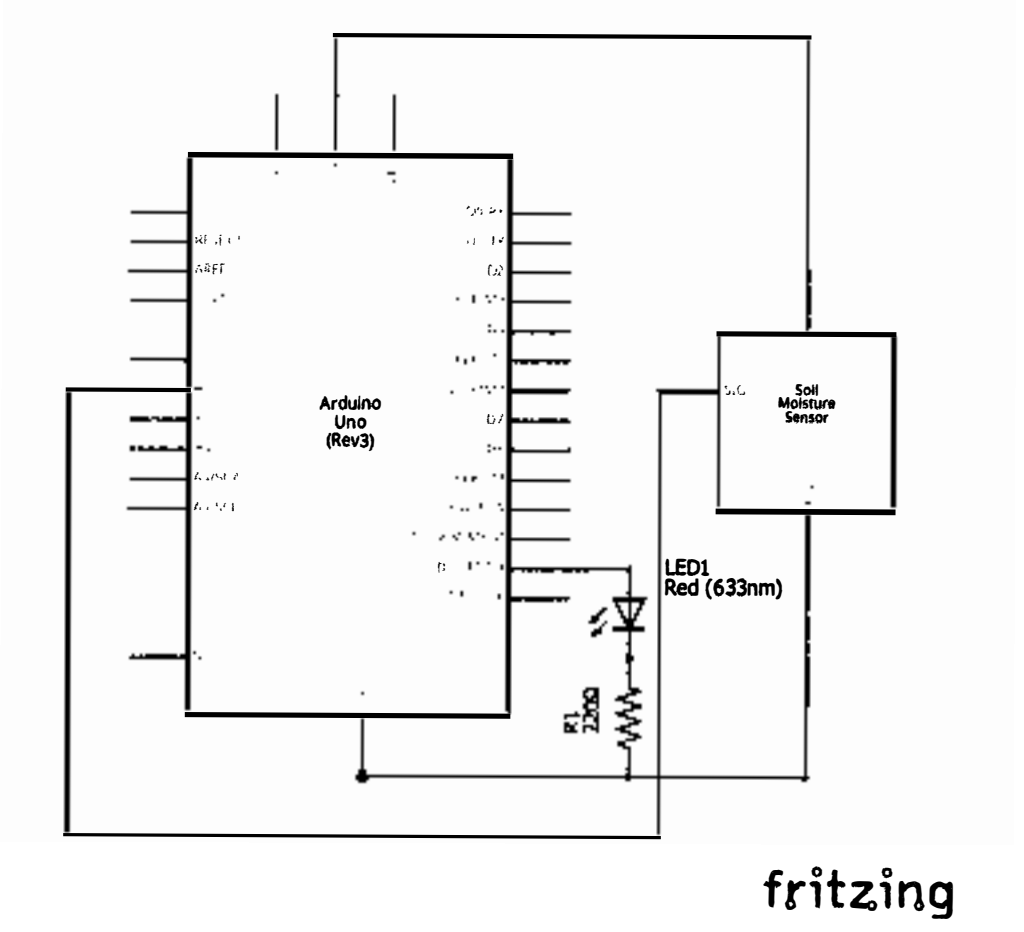


Рис. 91 ❖ Принципиальная схема подключения для практического занятия 25

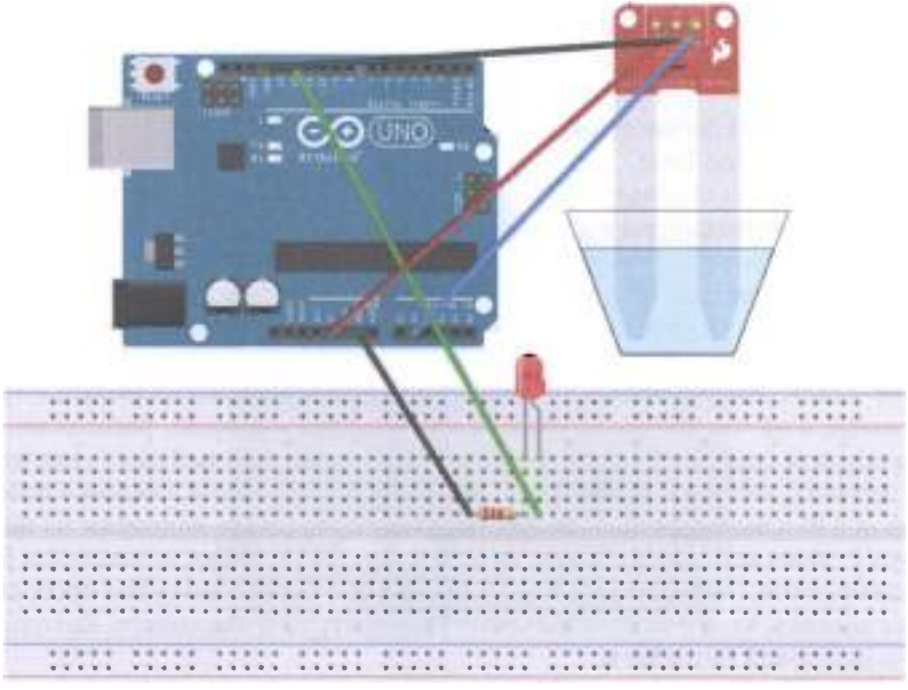


Рис. 92 ❖ Схема подключения с макетной платой для практического занятия 25

Код программы

```
int analogPin = 1; // level sensor connected to an analog port
int led = 12; // LED connected to digital port 12
int val = 0; // define the variable val initial value is 0
int data = 0; // define a variable data initial value is 0
void setup ()
{
pinMode (led, OUTPUT); // define led pin as an output
Serial.begin (9600); // set the baud rate to 9600
}
void loop ()
{
val = analogRead (analogPin); // read the analog value to give the variable val
if (val> 550) {// determine the variable val is greater than 700
digitalWrite (led, HIGH); // variable val is greater than 700, the light LED
}
else {
digitalWrite (led, LOW); // variable val is less than 700, the light goes off
}
data = val; // variable val assigned to the variable data
Serial.println (data); // serial print variable data
delay (100);
}
```

Практическое занятие 26. Эксперимент с сенсором температуры и влажности DHT11

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- сенсор температуры и влажности;
- соединительные провода.

Сенсор температуры и влажности гораздо меньше сенсора уровня воды. Он потребляет мало энергии и измеряет температуру в диапазоне от 0 до 50 °C, влажность – от 20 до 90 %. Точность измерений по влажности – ±5 %, по температуре – ±2 °C. У нас в комплекте сенсор DHT11 прикреплён на модуль с тремя выходами, а не четырьмя. На рис. 94 изображены 4 выхода, расположенных в правильном порядке: слева-направо идёт неподключённый пин (у нас на модуле его нет), затем – подключение сигнала к 8 пину платы, следом – пин питания (к 5 В) и земля. Но на всякий случай надо следовать надписям на нашем модуле с тремя выходами: (S – сигнал, на 8 пин; «–» – земля; оставшийся 3 выход посередине, очевидно, + (5 В)). После подключения и загрузки программы нужна консоль, чтобы видеть значения температуры и влажности, выводимые на 8-й цифровой пин платы.

Схема представлена на рис. 93–94.

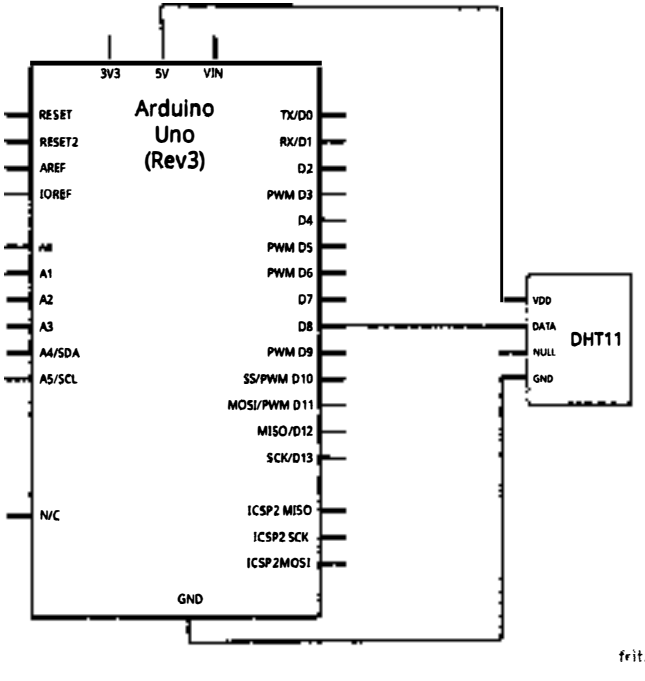


Рис. 93 ❖ Принципиальная схема подключения для практического занятия 26

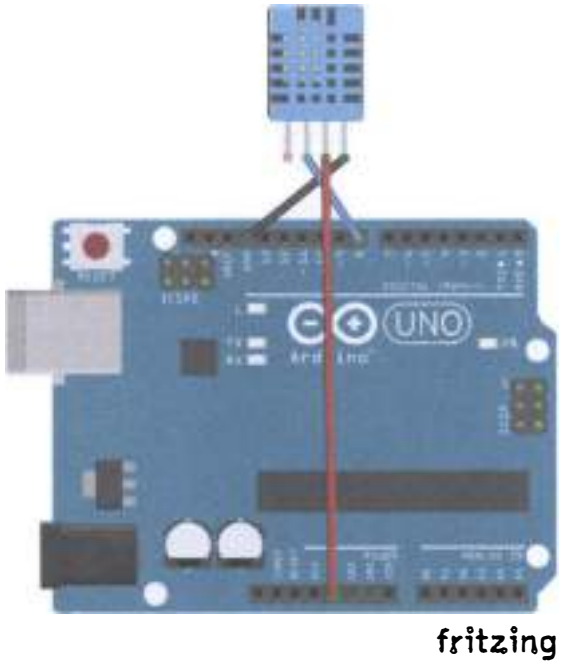


Рис. 94 ❖ Схема подключения для практического занятия 26

Код программы

```
int DHpin = 8;
byte dat [5];
byte read_data ()
{
byte data;
for (int i = 0; i <8; i++)
{
if (digitalRead (DHpin) == LOW)
{
while (digitalRead (DHpin) == LOW); // wait for 50us;
delayMicroseconds (30); // determine the duration of the high level to determine the data
is '0 'or
'1';
if (digitalRead (DHpin) == HIGH)
data |= (1 << (7-i)); // high front and low in the post;
while (digitalRead (DHpin) == HIGH); // data '1 ', wait for the next one receiver;
}
}
return data;
}
void start_test ()
{
```

```
digitalWrite (DHpin, LOW); // bus down, send start signal;
delay (30); // delay greater than 18ms, so DHT11 start signal can be detected;
digitalWrite (DHpin, HIGH);
delayMicroseconds (40); // Wait DHT11 response;
pinMode (DHpin, INPUT);
while (digitalRead (DHpin) == HIGH);
delayMicroseconds (80); // DHT11 a response, pulled the bus 80us;
if (digitalRead (DHpin) == LOW);
delayMicroseconds (80); // DHT11 80us after the bus pulled to start sending data;
for (int i = 0; i <4; i++) // receives temperature and humidity data, the parity bit is not
considered;
dat [i] = read_data ();
pinMode (DHpin, OUTPUT);
digitalWrite (DHpin, HIGH); // sending data once after releasing the bus, wait for the host
to open the next Start signal;
}
void setup ()
{
Serial.begin (9600);
pinMode (DHpin, OUTPUT);
}
void loop ()
{
start_test ();
Serial.print ("Current humdity =");
Serial.print (dat [0], DEC); // display the humidity-bit integer;
Serial.print ('.');
Serial.print (dat [1], DEC); // display the humidity decimal places;
Serial.println ('%');
Serial.print ("Current temperature =");
Serial.print (dat [2], DEC); // display the temperature of integer bits;
Serial.print ('.');
Serial.print (dat [3], DEC); // display the temperature of decimal places;
Serial.println ('C');
delay (700);
}
```

Практическое занятие 27. Эксперимент с релейным модулем

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- релейный модуль;
- синий светодиод;
- резистор на 220 Ом;
- макетная плата;
- соединительные провода.

Релейный модуль используется для переключения электрической схемы при возникновении определённого условия, подающегося на управляющий выход модуля, например определённого сигнала (0 или 1), значения и т. д. Применение такого модуля весьма широкое – от игрушек до систем сигнализации. У релейного модуля всего три выхода, помеченных соответственно +, – и S (управляющий/сигнальный выход). В этом занятии при срабатывании релейного модуля мы будем зажигать светодиод, подключённый не к плате, а к релейному модулю вместе с резистором на 220 Ом. Для этого понадобится отвёртка, чтобы открутить два винта двух контактов релейного модуля. Код программы крошечный: каждую секунду мы то зажигаем светодиод, посылая на 10-й цифровой пин платы 1, то гасим, посылая 0. При этом можно услышать соответствующий звук переключения релейного модуля – не пугайтесь.

Схема представлена на рис. 95–96.

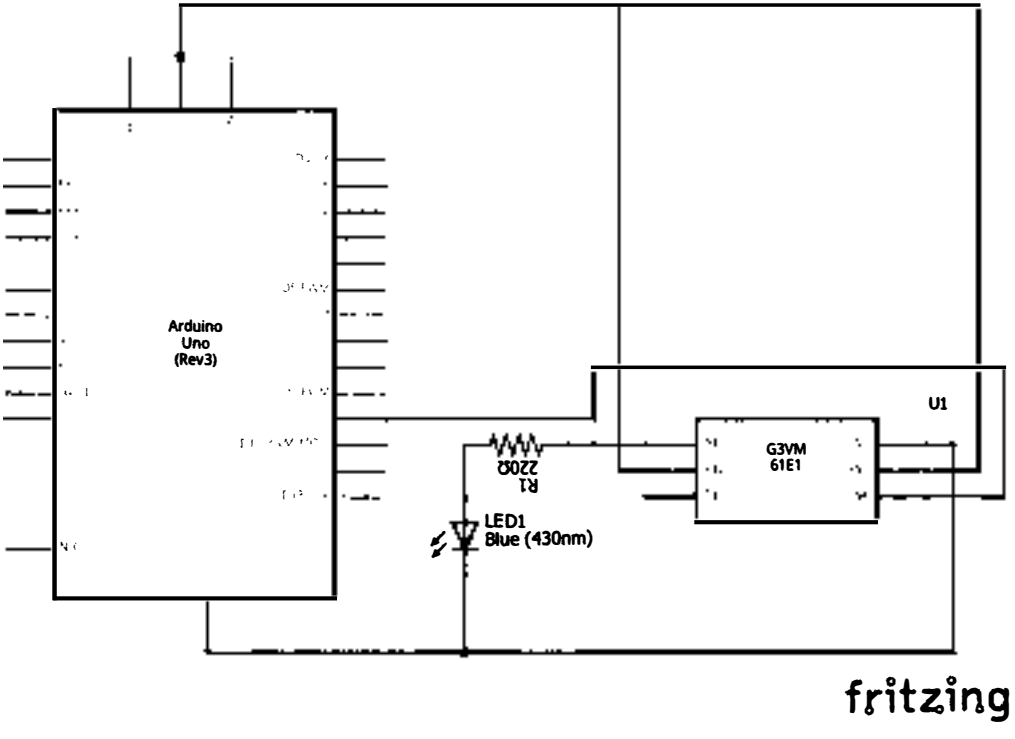


Рис. 95 ❖ Принципиальная схема подключения для практического занятия 27

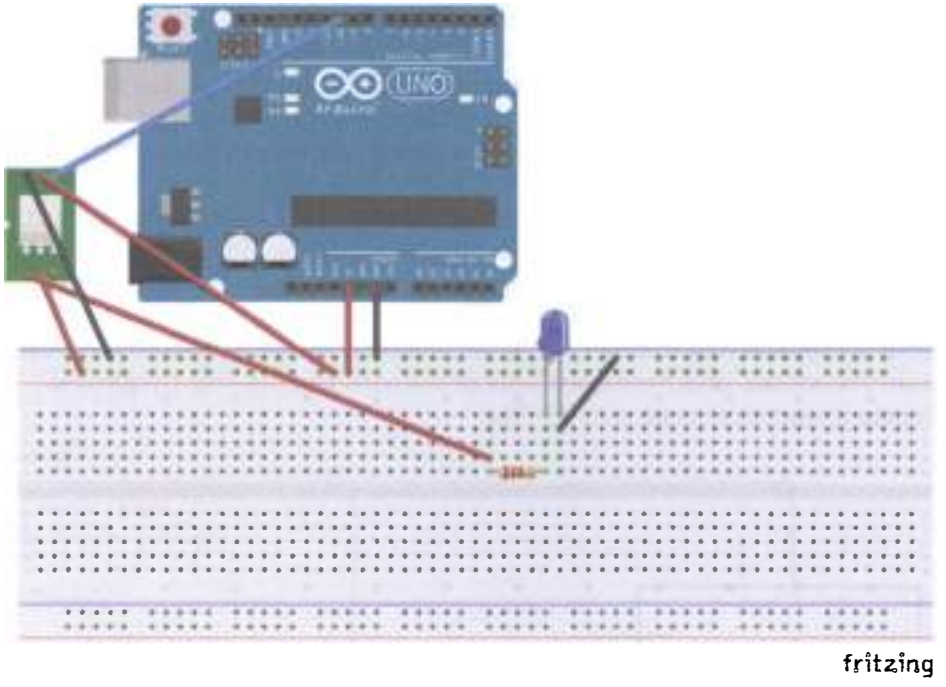


Рис. 96 ❖ Схема подключения с макетной платой для практического занятия 27

Код программы

```
int relay = 10; // relay turns trigger signal - active high;
void setup ()
{
pinMode (relay, OUTPUT); // Define port attribute is output;
}
void loop ()
{
digitalWrite (relay, HIGH); // relay conduction;
delay (1000);
digitalWrite (relay, LOW); // relay switch is turned off;
delay (1000);
}
```

Практическое занятие 28. Эксперимент с жидкокристаллическим монитором LCD1602A

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- жидкокристаллический монитор;
- потенциометр;
- 2 штырьковых коннектора по 8 пинов каждый;

- резистор на 220 Ом;
- макетная плата;
- соединительные провода.

Для работы с жидкокристаллическим монитором LCD1602A, у которого 16 выходов, нам понадобится библиотека LiquidCrystal.h, которая уже встроена в Arduino IDE. Этот мини-монитор может выводить 16 символов в каждой из 2 строк, работает от 5 В; с помощью потенциометра мы сможем регулировать яркость монитора. Фактически мы будем задействовать 12 пинов мини-монитора, поэтому предполагается, что к ним будут припаяны провода. Но можно обойтись и без этого с помощью макетной платы и штырьковых коннекторов двух типов – углового и прямого, каждый из которых содержит 8 пинов-выходов и вставляется через дырки-выходы LCD монитора в макетную плату, или сначала вставляется в макетную плату, а потом на него надеваются дырки вместе с монитором. С помощью представленного ниже кода выведем стандартную для подобных заданий фразу Hello, world! и начнём считать секунды. Обратите внимание, что контакт с монитором должен быть хороший всё время выполнения задания – сначала вы обеспечиваете хороший контакт с пинами монитора, потом подключаете плату к питанию и загружаете скетч. Можно обойтись без штырьковых коннекторов, если воткнуть провода через дырки пинов монитора в макетную плату, тем самым «пригвоздив» его к ней.

Схема представлена на рис. 97–98.

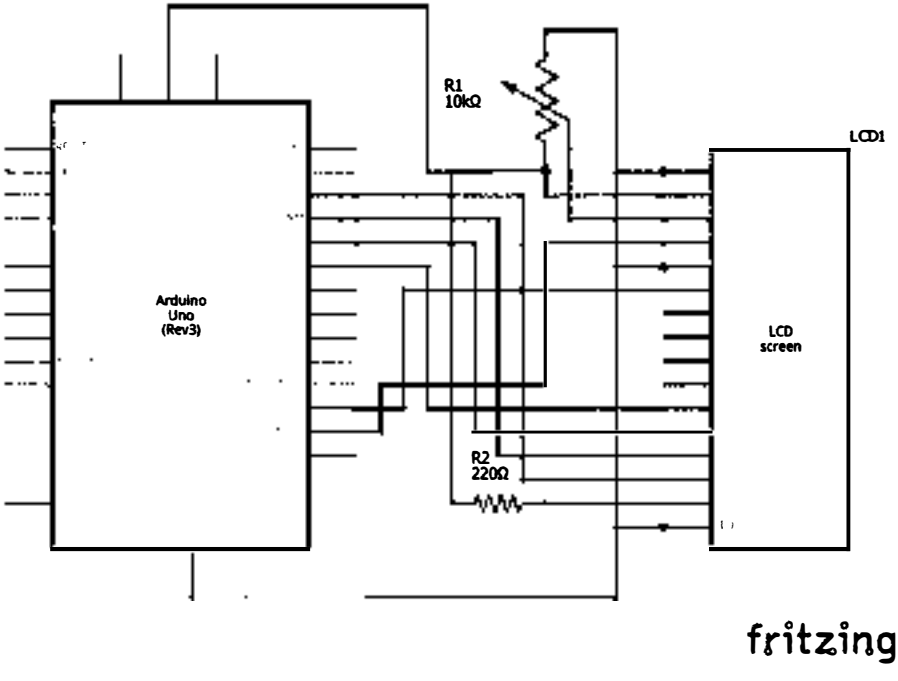


Рис. 97 ❖ Принципиальная схема подключения для практического занятия 28

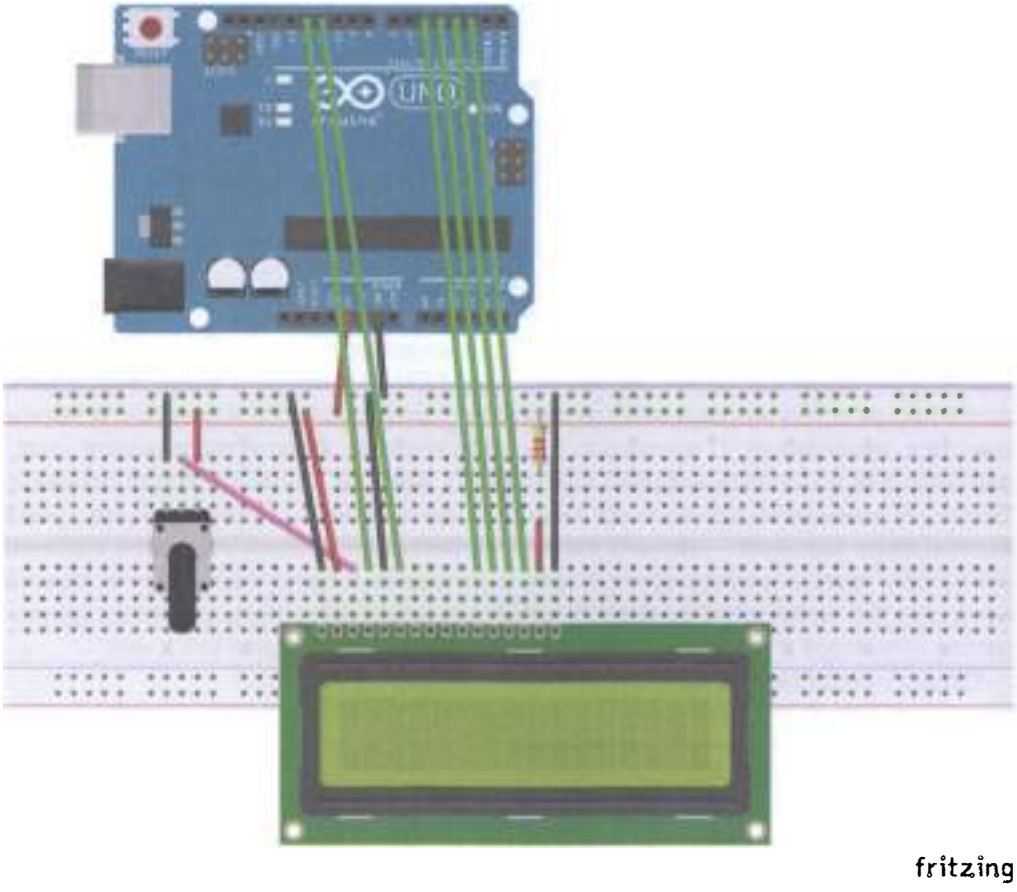


Рис. 98 ❖ Схема подключения с макетной платой для практического занятия 28

Код программы

```
// include the library code:
#include <LiquidCrystal.h>
// initialize the library by associating any needed LCD interface pin
// with the arduino pin number it is connected to
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
// set up the LCD's number of columns and rows:
lcd.begin(16, 2);
// Print a message to the LCD.
lcd.print("hello, world!");
}

void loop() {
// set the cursor to column 0, line 1
// (note: line 1 is the second row, since counting begins with 0):
```

```
lcd.setCursor(0, 1);
// print the number of seconds since reset:
lcd.print(millis() / 1000);
}
```

Практическое занятие 29. Эксперимент с шаговым двигателем

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- шаговый двигатель;
- модуль для шагового двигателя;
- потенциометр;
- макетная плата;
- соединительные провода.

Шаговый двигатель при получении импульсного сигнала поворачивает силовой привод (и рукоятку, прикреплённую к нему, или вообще любую деталь – actuator) на определённый угол (шаговый угол). Направление поворота, скорость, ускорение можно контролировать с помощью изменения частоты импульса или количества импульсов. Мы будем использовать четырёхфазный шаговый двигатель диаметром 28 мм с 5 выходами, который работает от 5 В и может поворачивать привод на угол 5,625*(1...64), т. е. на 5,625 градуса, 11,25 градуса и т. д. до 360 (т. е. вращаться до бесконечности). Также нам понадобится модуль ULN2003APG для шагового двигателя (подключается к земле и питанию 5 В соответствующими пинами – и +, а также к выходам 8–11 платы Arduino Uno с помощью пинов IN1–IN4), потенциометр для контроля скорости вращения привода (подключён к пину A0 платы) и библиотека Stepper.h, уже установленная в Arduino IDE. На схемах изображён немного другой шаговый двигатель, т. к. нашего в библиотеке Fritzing не нашлось.

Схема представлена на рис. 99–100.

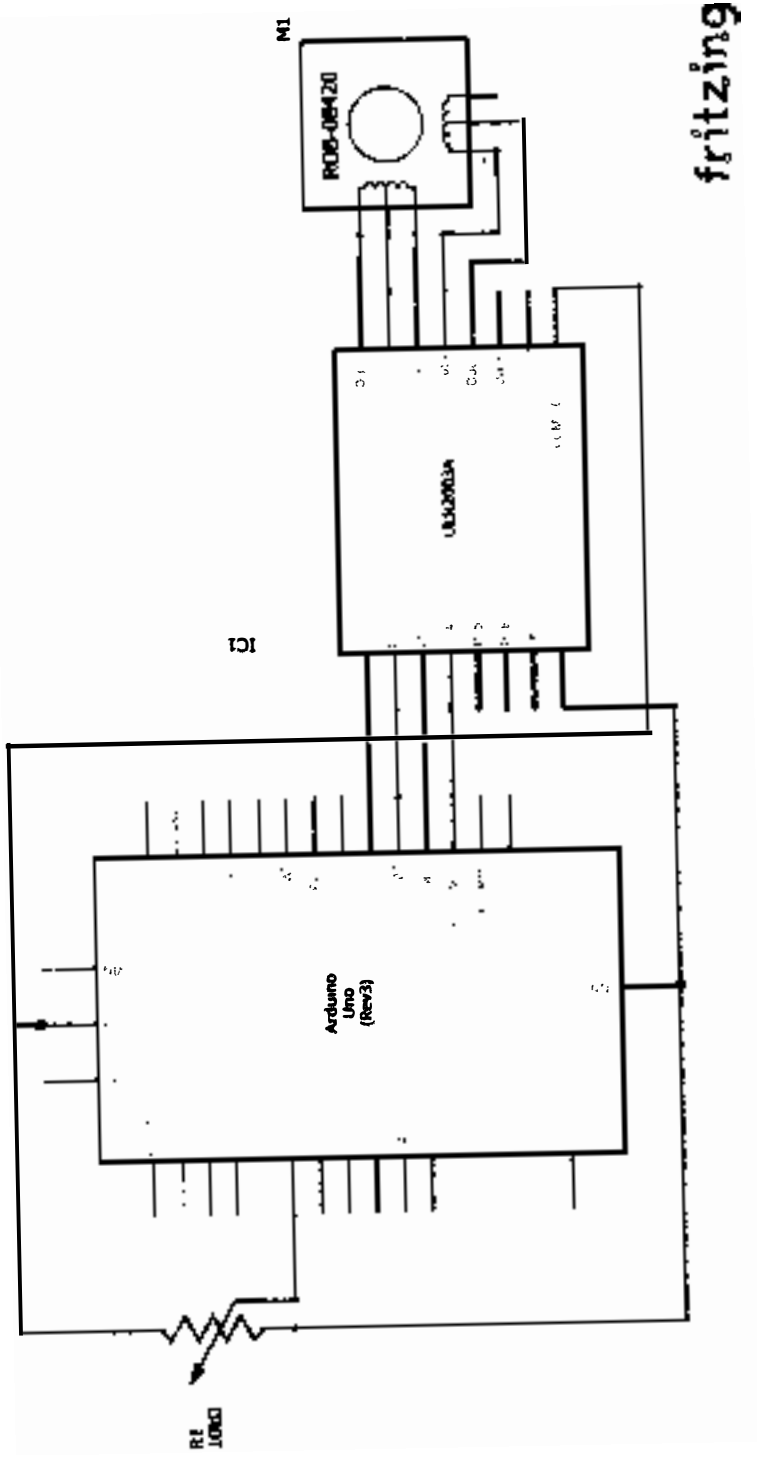


Рис. 99 ❖ Принципиальная схема подключения для практического занятия 29

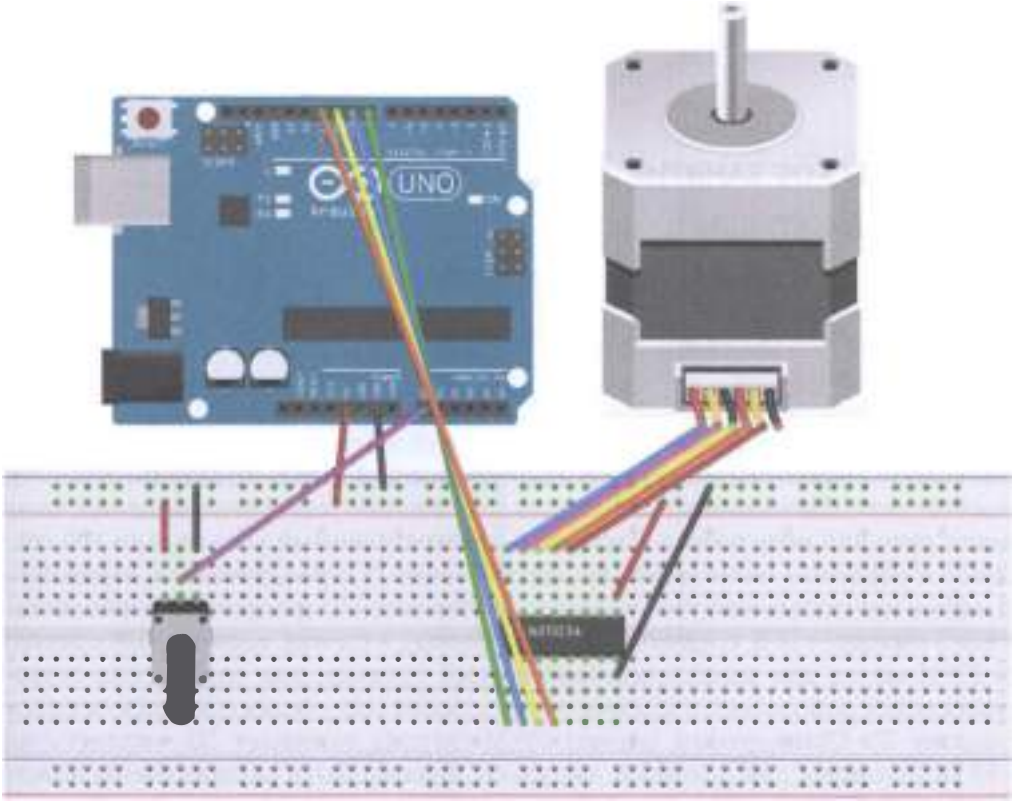


Рис. 100 ❖ Схема подключения с макетной платой для практического занятия 29

Код программы

```
#include <Stepper.h>
// Set here is the number of revolution of the stepper motor step
#define STEPS 100
// Attached to set the number of steps of the stepper motor and pin
Stepper stepper (STEPS, 8, 9, 10, 11);
// Define a variable to store the history of reading
int previous = 0;
void setup ()
{
// Set the motor speed of 90 per minute step
stepper.setSpeed (90);
}
void loop ()
{
// Get the sensor readings
int val = analogRead (0);
// Move to the current reading minus the number of steps historical readings
```

```
stepper.step (val - previous);
// Save history readings
previous = val;
}
```

Практическое занятие 30. Эксперимент с серводвигателем

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- серводвигатель;
- соединительные провода.

Серводвигатель (сервопривод), в отличие от шагового двигателя, представляет собой поворотный двигатель, перемещающий какую-то деталь (рукоятку, вообще любую деталь с ограниченным диапазоном/углом перемещения, чаще всего руку или ногу робота – actuator) на определённый угол от 0 до 180 градусов и обратно – он не может вращаться вечно (у серводвигателя, входящего в набор, максимальный угол поворота составляет 160 градусов). У серводвигателя 3 выхода: + (5 В, средний красный провод), – (земля, коричневый провод) и PWM-выход для управления (оранжевый провод); для управления серводвигателем используется широтно-импульсная модуляция (см. практическое занятие 3). Стандартный период PWM-сигнала занимает 20 миллисекунд (50 Гц), поэтому для поворота импульсы должны быть распределены между 1 и 2 миллисекундами, но на практике эти значения могут варьироваться от 0.5 до 2.5 миллисекунды для поворотов на углы от 0 до 180 градусов. Для поворота на 0 градусов (т. е. никакого поворота) на соответствующий PWM-пин платы подаётся сигнал 1000 (1000 микросекунд = 1 миллисекунда), на 45 градусов – 1250, на 180 градусов – 2000 микросекунд. В реальности эти значения могут варьироваться от 500 до 2480 микросекунд. Чтобы увидеть поворот, можно надеть самую большую насадку из маленького пакетика с 3 насадками и 3 шурупами на белую шестерню двигателя. Напишем программу, в которой через консоль будем вводить значения угла, на который мы хотим повернуть пластмассовую насадку, надетую на двигатель. Если вводить цифры от 1 до 9, то можно поворачивать насадку, соответственно, на 20 градусов, 40, 60 и т. д. (9 = 180 градусов; работать не должно, т. к. у серводвигателя, входящего в набор, максимальный угол поворота составляет 160 градусов). Потом напишем другую программу, использующую встроенную библиотеку Servo.h для управления серводвигателем.

Схема представлена на рис. 101–102.

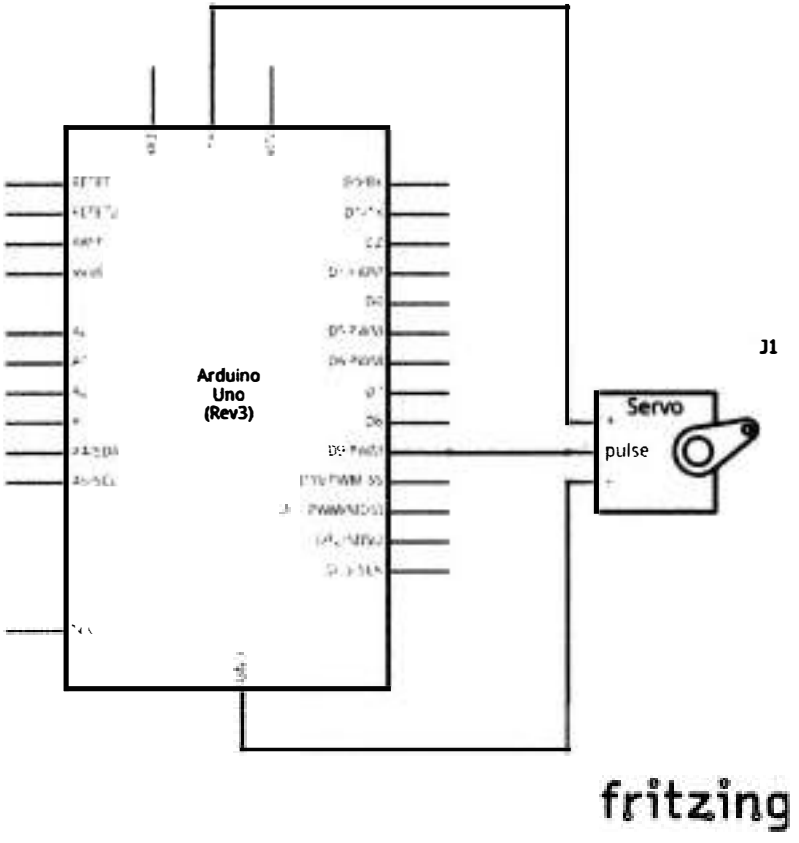


Рис. 101 ❖ Принципиальная схема подключения для практического занятия 30

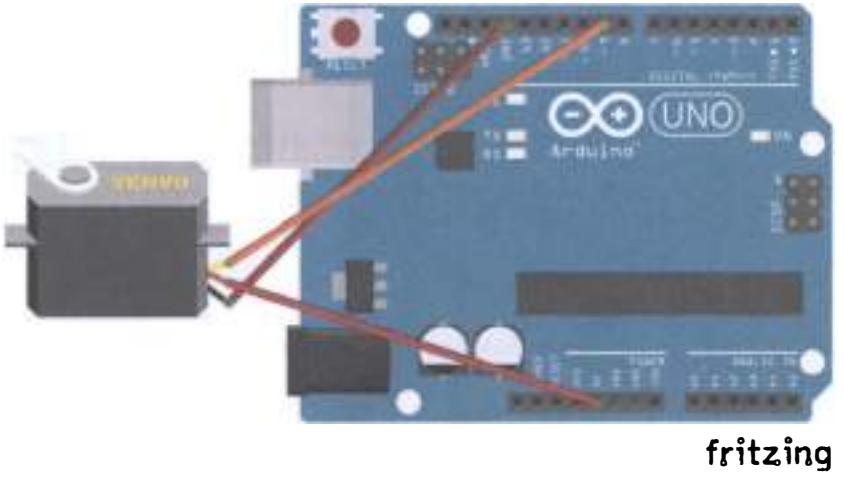


Рис. 102 ❖ Схема подключения для практического занятия 30

Код программы 1

```
int servopin = 9 ;// define the digital interface to connect the servo servo signal line 9
int myangle ;// define the angle variables
int pulsewidth ;// define variable pulse width
int val;
void servopulse (int servopin, int myangle) // define a pulse function
{
    pulsewidth = (myangle * 11) +500 ;// the angle value into a pulse width 500-2480
    digitalWrite (servopin, HIGH) ;// the servo interface level to high
    delayMicroseconds (pulsewidth) ;// number of microseconds delay pulse width value
    digitalWrite (servopin, LOW) ;// the servo interface level to Low
    delay (20-pulsewidth/1000);
}
void setup ()
{
    pinMode (servopin, OUTPUT) ;// set servo interface output interface
    Serial.begin (9600) ;// connect to the serial port, the baud rate is 9600
    Serial.println ("servo = o_seral_simple ready");
}
void loop () // will be 0-9's 0-180 number into perspective, and let the number of times
the  corresponding LED flashes
{
    val = Serial.read () ;// read the value of the serial port
    if (val> '0'&& val <= '9')
    {
        val = val-'0'; // will be converted to numeric variables characteristic quantities
        val = val * (180/9) ;// will figure into perspective
        Serial.print ("moving servo to");
        Serial.print (val, DEC);
        Serial.println ();
        for (int i = 0; i <= 50; i++) // give enough time to let it go to the steering
angle specified
        {
            servopulse (servopin, val) ;// reference pulse function
        }
    }
}
```

Код программы 2

```
#include <Servo.h> // define header files, there is one thing to note, you can directly
click on themenu bar at the Arduino software Sketch> Importlibrary> Servo, Servo function
call, you can also directly enter the # include <Servo.h>, But at the input to the
attention of the # include and <Servo.h> should be a space between, otherwise a compile-
time error.
Servo myservo ;// define a variable name servos
void setup ()
{
myservo.attach (9) ;// define Servo Interface (9,10 all OK, shortcomings can only control
2)
}
void loop ()
{
myservo.write (90) ;// set the steering angle of rotation
}
```

Практическое занятие 31. Эксперимент с игровым джойстиком

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- джойстик;
- соединительные провода.

Игровой джойстик позволяет контролировать значения по осям *x* и *y* и нажатия кнопки; при этом значения по *x* и по *y* отсылаются на аналоговые порты платы (например, A0 и A1 соответственно), а нажатия кнопки – на цифровой порт (например, 7). У джойстика 5 выходов, но они подписаны, так что проблем возникнуть не должно, если учесть, что SW – это определение нажатия кнопки (подаём на 7-й цифровой пин платы). Также нам понадобится консоль (монитор порта, **Ctrl+Shift+M**), чтобы видеть выводимые значения при выполнении программы 1. Вторая программа просто немного по-другому делает то же самое.

Схема представлена на рис. 103–104.

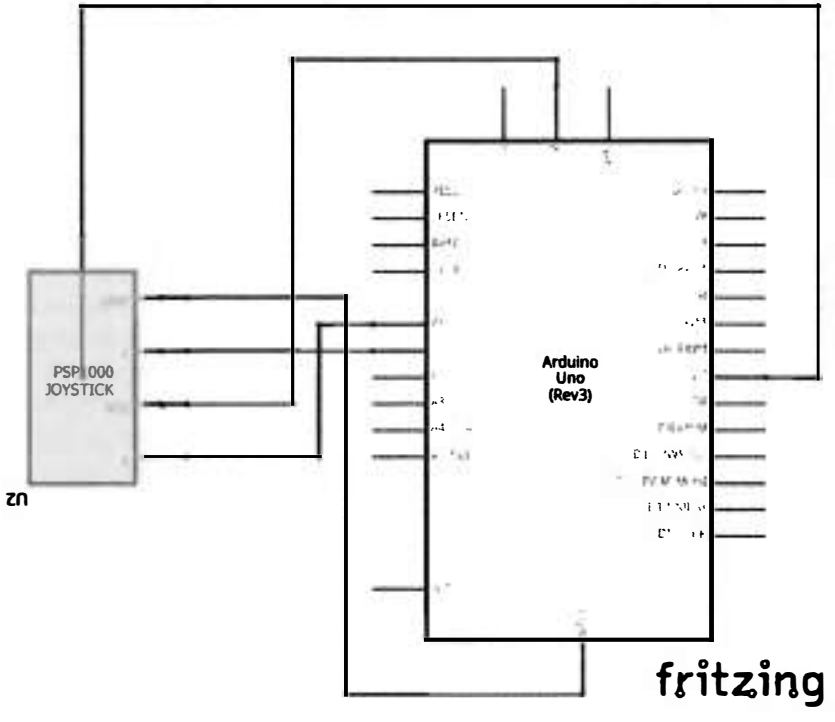


Рис. 103 ❖ Принципиальная схема подключения для практического занятия 31

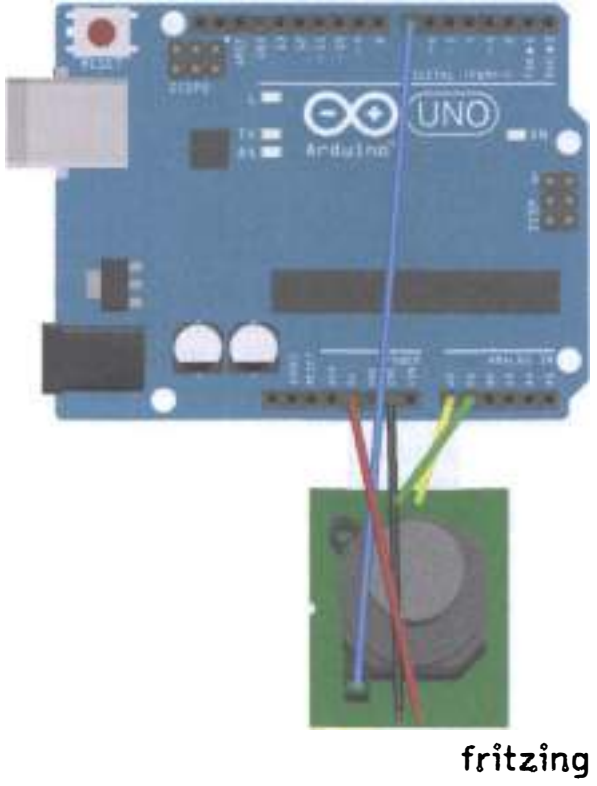


Рис. 104 ❖ Схема подключения для практического занятия 31

Код программы 1

```
int value = 0;
void setup () {
    pinMode (7, INPUT);
    Serial.begin (9600);
}
void loop () {
    value = analogRead (0);
    Serial.print ("X:");
    Serial.print (value, DEC);
    value = analogRead (1);
    Serial.print (" | Y:");
    Serial.print (value, DEC);
    value = digitalRead (7);
    Serial.print (" | Z:");
    Serial.println (value, DEC);
    delay (100);
}
```

Код программы 2

```
int JoyStick_X = 0; // x
int JoyStick_Y = 1; // y
int JoyStick_Z = 7; // key
void setup ()
{
    pinMode (JoyStick_X, INPUT);
    pinMode (JoyStick_Y, INPUT);
    pinMode (JoyStick_Z, INPUT);
    Serial.begin (9600); // 9600 bps
}
void loop ()
{
    int x, y, z;
    x = analogRead (JoyStick_X);
    y = analogRead (JoyStick_Y);
    z = digitalRead (JoyStick_Z);
    Serial.print (x, DEC);
    Serial.print (",");
    Serial.print (y, DEC);
    Serial.print (",");
    Serial.println (z, DEC);
    delay (100);
}
```

Практическое занятие 32. Эксперимент с инфракрасным пультом дистанционного управления

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- инфракрасный пульт дистанционного управления;
- инфракрасный приёмник;
- 6 светодиодов;
- 6 резисторов на 220 Ом;
- макетная плата;
- соединительные провода.

Сигналы, поступающие с инфракрасного пульта дистанционного управления (ИК-пульта ДУ), представляют собой двоичный импульсный код. Чтобы передать без проводов сигнал в ИК-диапазоне, сначала двоичный код проходит модуляцию на определённой частоте, и затем результат высылается ИК-светодиодами в направлении ИК-приёмника, который демодулирует полученный сигнал и превращает его обратно в двоичный код. Оптический сигнал, передаваемый в ИК-диапазоне ИК-светодиодами, преобразуется в слабый электрический сигнал, который затем усиливается, пропускается через фильтр, демодулятор и, таким образом, восстанавливается в исходный сигнал

в двоичном коде. ИК-приёмник сигнала располагает 3 пинами: + (5 В), – (земля) и VOUT для приёма аналогового оптического сигнала. Протокол передачи данных, по которому исходный сигнал кодируется ИК-пультом ДУ и затем декодируется ИК-приёмником сигнала, – это NEC-протокол: он использует 8 адресных битов и 8 битов для команды; адресные и командные биты пересылаются дважды, чтобы убедиться в их правильности; используется импульсно-позиционная модуляция; частота сигнала равна 38 КГц; логическая единица длится 2,25 миллисекунды, в которых первые 0,56 миллисекунды уровень сигнала 1, остальные – 0; логический ноль длится 1,12 миллисекунды, где опять первые 0,56 миллисекунды уровень сигнала равен 1, а остальные – 0. При подаче сигнала первые 9 миллисекунд сигнал равен 1, а затем следует 0 уровень, длящийся 4,5 миллисекунды. Такая комбинация означает, что дальше будет передаваться, собственно, основная информация (адрес + адрес + команда + команда для каждого сигнала). Если нажать на кнопку пульта и не отпускать её, вся подготовка и пересылка команды занимает 110 миллисекунд, которые затем повторяются каждые 110 миллисекунд с помощью повторяющего импульса (9 миллисекунд подаём 1, и 2,25 миллисекунды – 0), если кнопка остаётся нажатой. Для выполнения этого задания нам понадобится библиотека IRemote.h – придётся её установить из [7], т. к. встроенная библиотека немного другая – для роботов; также нам понадобится консоль (монитор порта). Программа, которую мы напишем, будет зажигать светодиоды и выводить в консоль информацию о том, что сигнал распознан (это будет NEC-сигнал, но можно попробовать и другой ИК-пульт ДУ: у меня получилось распознать сигнал с пульта от телевизора SONY). Светодиоды будут зажигаться в соответствии с нажатыми на пульте кнопками: первые 6 кнопок сверху (двух верхних рядов) зажигают светодиоды, вторые 6 кнопок сверху (ряд 3 и 4) гасят светодиоды. Прежде чем нажимать на кнопки пульта, не забудьте вынуть блокирующую контакты батарейки полоску из него! И вставить её обратно после завершения выполнения задания. А теперь самое интересное (для тех, кто дочитал до конца): у ИК-приёмника 3 выхода, и подключать их надо следующим образом: если ИК-приёмник повёрнут к вам «лицом» (на котором «маска» крест-накрест), то справа от вас находится пин питания, слева от вас – пин OUT, который подаётся к 11-му цифровому пину платы, а посередине у ИК-приёмника – земля.

Схема представлена на рис. 105–106.

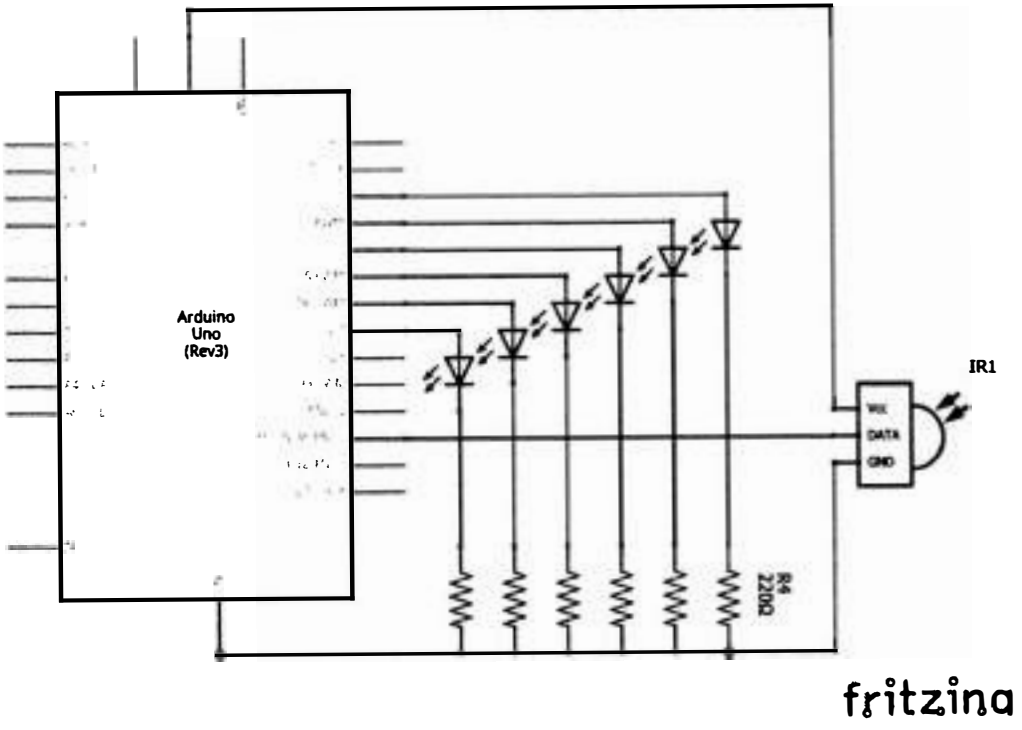


Рис. 105 ❖ Принципиальная схема подключения для практического занятия 32

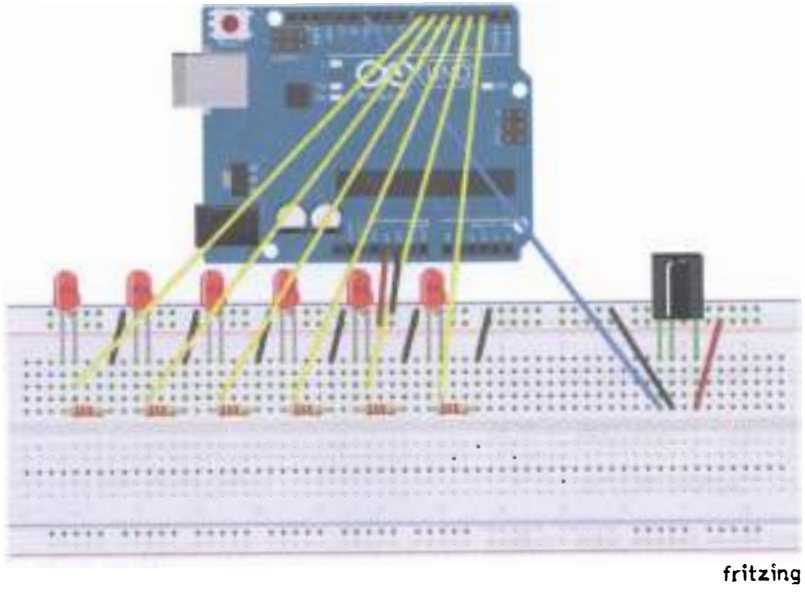


Рис. 106 ❖ Схема подключения с макетной платой для практического занятия 32

Код программы

```
#include <IRremote.h>
int RECV_PIN = 11;
int LED1 = 2;
int LED2 = 3;
int LED3 = 4;
int LED4 = 5;
int LED5 = 6;
int LED6 = 7;
long on1 = 0x00FFA25D;
long off1 = 0x00FFE01F;
long on2 = 0x00FF629D;
long off2 = 0x00FFA857;
long on3 = 0x00FFE21D;
long off3 = 0x00FF906F;
long on4 = 0x00FF22DD;
long off4 = 0x00FF6897;
long on5 = 0x00FF02FD;
long off5 = 0x00FF9867;
long on6 = 0x00FFC23D;
long off6 = 0x00FFB04F;
IRrecv irrecv (RECV_PIN);
decode_results results;
// Dumps out the decode_results structure.
// Call this after IRrecv :: decode ()
// Void * to work around compiler issue
// Void dump (void * v) {
// Decode_results * results = (decode_results *) v
void dump (decode_results * results) {
int count = results-> rawlen;
if (results-> decode_type == UNKNOWN)
{
Serial.println ("Could not decode message");
}
else
{
if (results-> decode_type == NEC)
{
Serial.print ("Decoded NEC:");
}
else if (results-> decode_type == SONY)
{
Serial.print ("Decoded SONY:");
}
else if (results-> decode_type == RC5)
{
Serial.print ("Decoded RC5:");
}
else if (results-> decode_type == RC6)
{
Serial.print ("Decoded RC6:");
}
```

```
Serial.print (results-> value, HEX);
Serial.print ("(");
Serial.print (results-> bits, DEC);
Serial.println ("bits)");
}
Serial.print ("Raw (");
Serial.print (count, DEC);
Serial.print ("):");
for (int i = 0; i <count; i++)
{
if ((i%2) == 1) {
Serial.print (results-> rawbuf [i] * USECPERTICK, DEC);
}
else
{
Serial.print (- (int) results-> rawbuf [i] * USECPERTICK, DEC);
}
Serial.print ("");
}
Serial.println ("");
}
void setup ()
{
pinMode (RECV_PIN, INPUT);
pinMode (LED1, OUTPUT);
pinMode (LED2, OUTPUT);
pinMode (LED3, OUTPUT);
pinMode (LED4, OUTPUT);
pinMode (LED5, OUTPUT);
pinMode (LED6, OUTPUT);
pinMode (13, OUTPUT);
Serial.begin (9600);
irrecv.enableIRIn (); // Start the receiver
}
int on = 0;
unsigned long last = millis ();
void loop ()
{
if (irrecv.decode (& results))
{
// If it's been at least 1/4 second since the last
// IR received, toggle the relay
if (millis () - last> 250)
{
on =! on;
// DigitalWrite (8, on? HIGH: LOW);
digitalWrite (13, on? HIGH: LOW);
dump (& results);
}
if (results.value == on1)
digitalWrite (LED1, HIGH);
if (results.value == off1)
```

```
digitalWrite (LED1, LOW);
if (results.value == on2)
digitalWrite (LED2, HIGH);
if (results.value == off2)
digitalWrite (LED2, LOW);
if (results.value == on3)
digitalWrite (LED3, HIGH);
if (results.value == off3)
digitalWrite (LED3, LOW);
if (results.value == on4)
digitalWrite (LED4, HIGH);
if (results.value == off4)
digitalWrite (LED4, LOW);
if (results.value == on5)
digitalWrite (LED5, HIGH);
if (results.value == off5)
digitalWrite (LED5, LOW);
if (results.value == on6)
digitalWrite (LED6, HIGH);
if (results.value == off6)
digitalWrite (LED6, LOW);
last = millis ();
irrecv.resume (); // Receive the next value
}
}
```

Практическое занятие 33. Эксперимент с RFID-модулем RC522

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- RFID-модуль;
- RFID-ключ;
- RFID-карта;
- штырьковый коннектор на 8 пинов;
- макетная плата;
- соединительные провода.

RFID – Radio Frequency IDentification, т. е. идентификация по радиочастоте. Принцип действия схож с ИК-пультом ДУ и ИК-приёмником, только здесь используется радиосигнал, а не оптический сигнал. Тоже есть источник сигнала (у нас это будет карточка, также он называется тег, метка, носитель сигнала, транспондер) и приёмник – RFID-модуль (считыватель, ридер, сканер). **Для RFID-модуля надо использовать питание 3.3 В, иначе он сгорит.** Также в этом занятии нам понадобятся RFID-ключ и RFID-карта – это 2 отдельных носителя сигнала, или метки, а ещё – библиотека SPI.h, которая уже встроена в Arduino IDE, библиотека MFRC522.h, которую можно скачать и установить из

[8], и консоль, в которой мы будем, поднося к RFID-модулю сначала карточку, потом ключ, считывать данные, записанные на них. Карту надо удерживать у модуля секунд 10, чтобы прочитались все данные! На схемах ниже представлен другой элемент – RFID USB Reader из набора SparkFun библиотеки Fritzing, но главное, что у него 8 выходов, и последовательность подключения сохранена (лучше смотреть цветную схему с макетной платой). У нашего RFID-модуля RC522 тоже 8 выходов, и подключаются они следующим образом: 3,3V – 3,3V, RST – D9 (или просто 9 – здесь имеются в виду только цифровые выходы платы), GND – GND, MISO – 12, MOSI – 11, SCK – 13, SDA – 10. Пропускаем мы только выход прерываний IRQ – он не подключён ни к чему. Обратите внимание, что контакт с RFID-модулем должен быть хороший всё время выполнения задания – сначала вы обеспечиваете хороший контакт с пинами модуля, потом подключаете плату к питанию и загружаете скетч. Можно обойтись без штырьковых коннекторов, если воткнуть провода через дырки пинов RFID-модуля в макетную плату.

Схема представлена на рис. 107–108.

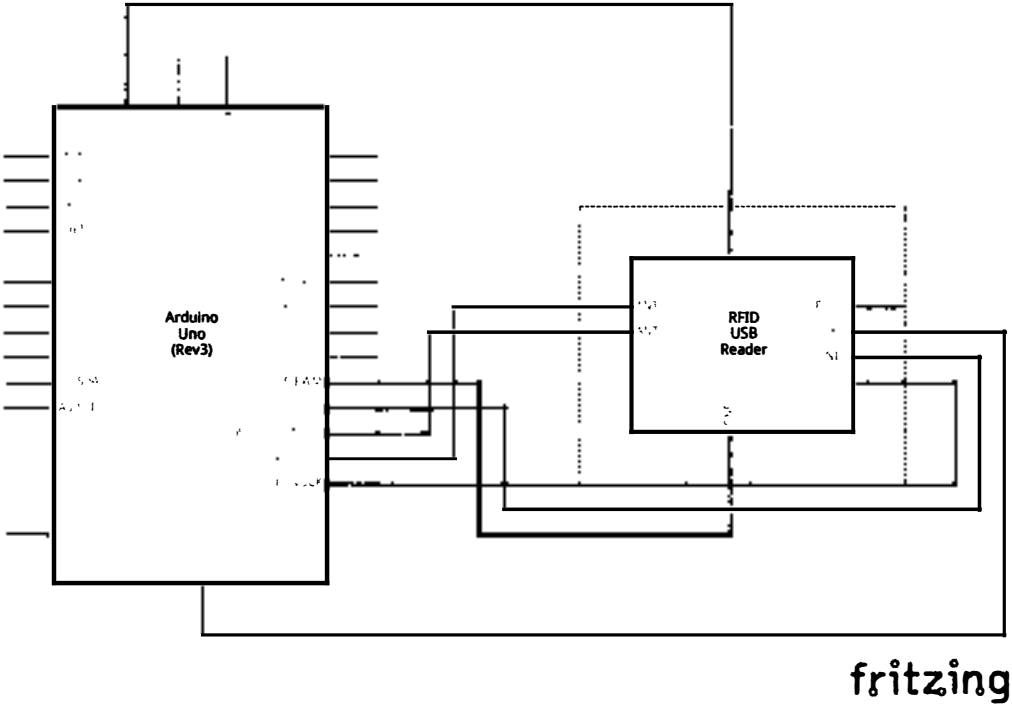


Рис. 107 ❖ Принципиальная схема подключения для практического занятия 33

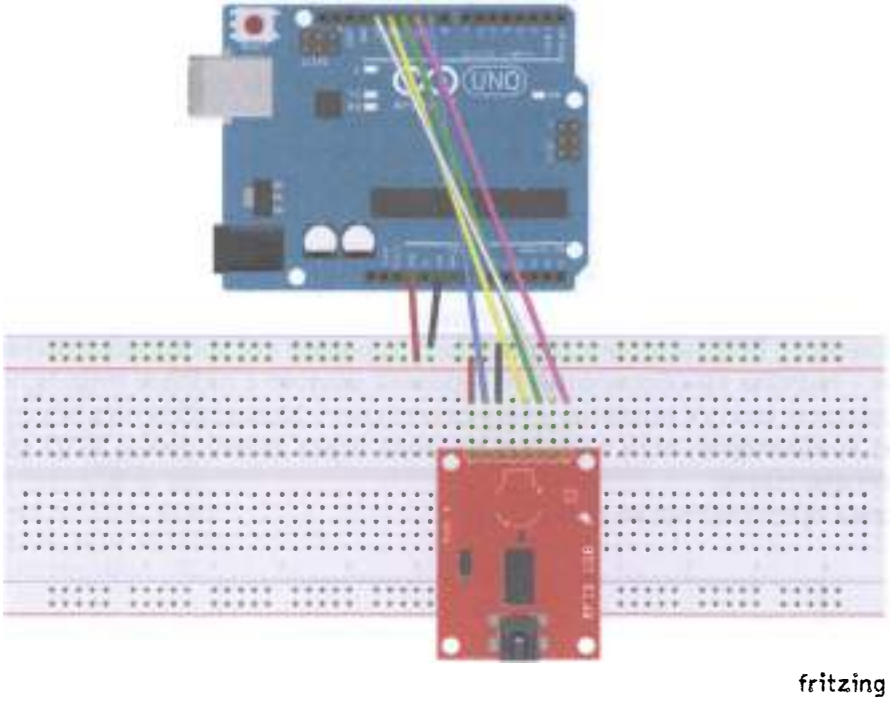


Рис. 108 ❖ Схема подключения с макетной платой для практического занятия 33

Код программы

```
#include <SPI.h>
#include <MFRC522.h>

#define RST_PIN         9          //
#define SS_PIN          10         //

MFRC522 mfrc522(SS_PIN, RST_PIN);  // Create MFRC522 instance

void setup() {
Serial.begin(9600);       // Initialize serial communications with the PC
while (!Serial);      // Do nothing if no serial port is opened (added for Arduinos based
on ATMEGA32U4)
SPI.begin();          // Init SPI bus
mfrc522.PCD_Init();       // Init MFRC522
ShowReaderDetails();   // Show details of PCD - MFRC522 Card Reader details
Serial.println(F("Scan PICC to see UID, type, and data blocks..."));
}

void loop() {
// Look for new cards
if ( ! mfrc522.PICC_IsNewCardPresent()) {
return;
}

// Select one of the cards
if ( ! mfrc522.PICC_ReadCardSerial()) {
```

```
return;
}
// Dump debug info about the card; PICC_HaltA() is automatically called
mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
}

void ShowReaderDetails() {
// Get the MFRC522 software version
byte v = mfrc522.PCD_ReadRegister(mfrc522.VersionReg);
Serial.print(F("MFRC522 Software Version: 0x"));
Serial.print(v, HEX);
if (v == 0x91)
Serial.print(F(" = v1.0"));
else if (v == 0x92)
Serial.print(F(" = v2.0"));
else
Serial.print(F(" (unknown)"));
Serial.println("");
// When 0x00 or 0xFF is returned, communication probably failed
if ((v == 0x00) || (v == 0xFF)) {
Serial.println(F("WARNING: Communication failure, is the MFRC522 properly connected?"));
}
}
```

Практическое занятие 34. Эксперимент с системой контроля доступа

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- RFID-модуль;
- RFID-ключ;
- RFID-карта;
- штырьковый коннектор на 8 пинов;
- релейный модуль;
- 2 светодиода – красный и синий (зелёный);
- 2 резистора на 220 Ом;
- макетная плата;
- соединительные провода.

В этом занятии нам понадобится опыт предыдущего занятия и 27-го занятия, где мы работали с релейным модулем. Суть задачи в том, чтобы собрать схему с RFID-модулем и релейным модулем, в которой будут 2 светодиода – красный и синий (или зелёный, но зелёных у нас в наборе нет), и затем написать программу, которая при поднесении RFID-карты с правильным записанным на неё паролем к RFID-модулю будет переключать релейный модуль и включать синий светодиод, сигнализируя о том, что доступ открыт (access granted), в противном случае (неправильный пароль, записанный на RFID-ключ) будет гореть красный светодиод (доступ закрыт, access denied). **Не забудьте, что**

RFID-модуль подключается к 3.3 В, а релейный модуль – к 5 В! Сначала мы должны запустить код в программе 1 (SetPassword), открыть консоль (монитор порта, **Ctrl+Shift+M**) и поднести RFID-карту к RFID-модулю, чтобы изменить пароль; при этом мы увидим сообщение «The password has been changed!» среди бесчисленных сообщений «Error!» в консоли. RFID-ключ мы не подносим, т. е. пароль на нём не меняется на правильный. Таким образом, на карте у нас записан правильный пароль, на ключе – неправильный. Затем мы запускаем код программы 2 (DoorCon) и подносим RFID-карту – релейный модуль переключается, и на 5 секунд должен загореться синий светодиод, сигнализирующий о том, что на карте пароль правильный; при этом красный светодиод гаснет на те же 5 секунд. Затем подносим RFID-ключ и видим, что ничего не происходит.

Схема представлена на рис. 109–110.

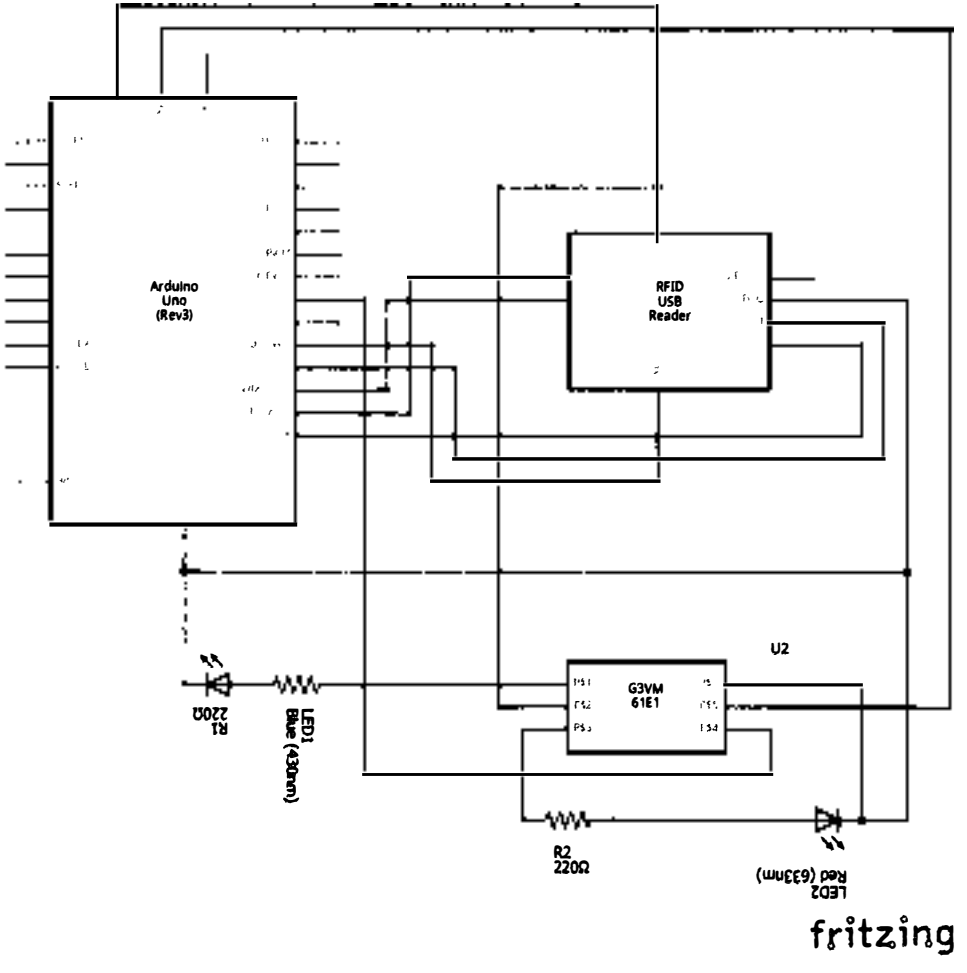


Рис. 109 ❖ Принципиальная схема подключения для практического занятия 34

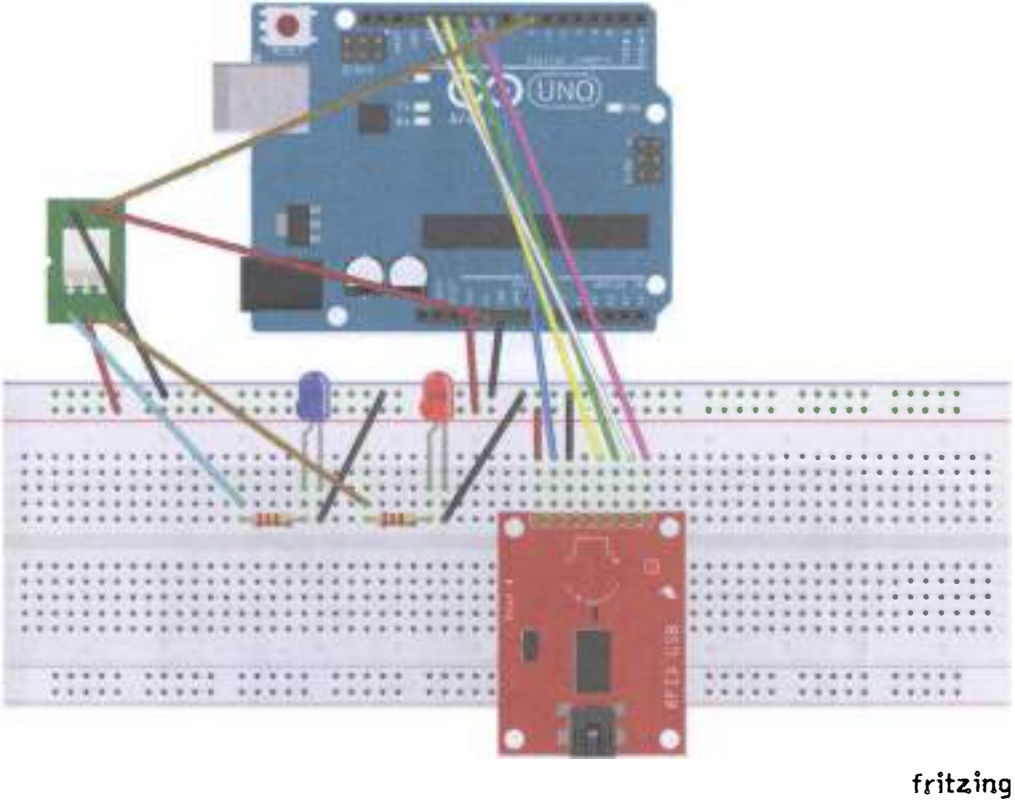
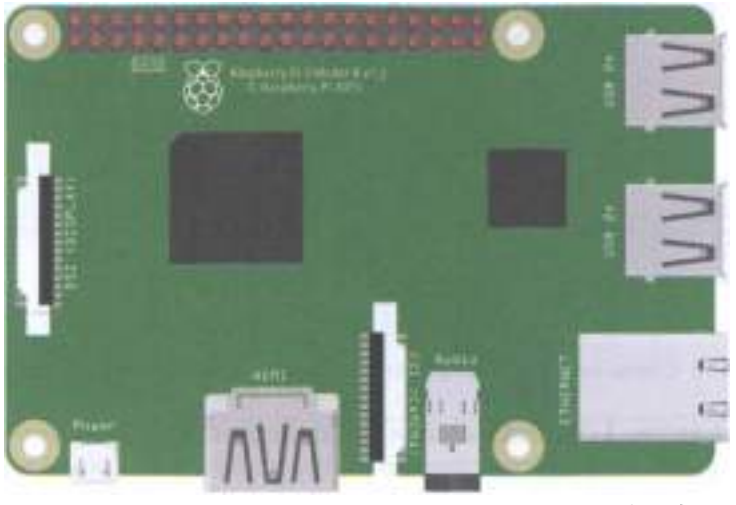


Рис. 110 ❖ Схема подключения с макетной платой для практического занятия 34

Код программы 1 (SetPassword) и **код программы 2 (DoorCon)** загружены по адресам [13] и [14] соответственно ввиду их большого объёма и большого количества ошибок при их копировании в Arduino IDE.

Часть 2

Практика на Raspberry Pi 3 (модель B)



fritzing

Основано на книге Android Things Projects [15]

Введение

Raspberry Pi 3 Model B продолжает оставаться наиболее популярным в мире компьютером на плате размером с кредитную карту (SBC – Single Board Computer, также SoM – System on Module), даже несмотря на выход в 2017 году новой версии – Model B+. По сути, эта плата может заменить ваш настольный (десктопный) компьютер, если, конечно, вы не играете в требовательные компьютерные игры и не занимаетесь редактированием 3D-графики и прочим майнингом. На плате нет никаких вентиляторов, поэтому она будет самым тихим компьютером, на который для теплоотведения можно прикрепить три радиатора, входящих в официальный комплект поставки этой платы [21]. Raspberry Pi 3 Model B располагает следующими характеристиками [20], см. рис. 111:

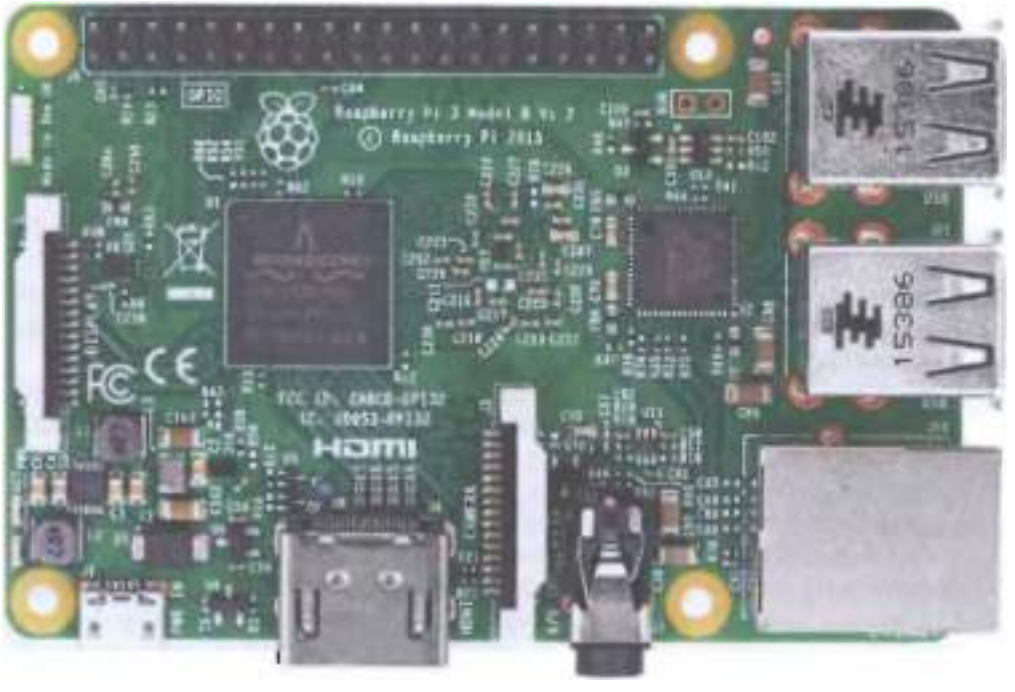


Рис. 111 ❖ Лицевая сторона платы Raspberry Pi 3 модели B [20]

- 64-битным четырёхъядерным процессором ARM Cortex-A53 с тактовой частотой 1,2 ГГц на ядро на однокристальном чипе Broadcom BCM2837 (самый большой чёрный квадрат на рисунке, на него прикрепляется самый большой радиатор из комплекта в первую очередь);
- графическим двухъядерным процессором VideoCore IV®, поддерживающим стандарты OpenGL ES 2.0, OpenVG, MPEG-2, VC-1 и способным ко-

дировать, декодировать и выводить Full HD-видео (1080p, 60 FPS, H.264 High-Profile);

- 1 ГБ оперативной памяти LPDDR2 SDRAM, которая делится с графической подсистемой (чип памяти располагается в виде чёрного квадрата в центре оборотной стороны платы; один из радиаторов (самый маленький) в комплекте [21] предназначен именно для него);
- цифровым видеовыходом HDMI (на рисунке внизу, под надписью HDMI; при подключении к нему разрешение варьируется от 640×350 (EGA) до 1920×1200 (WUXGA) для HDMI. Композитный выход работает в форматах PAL и NTSC);
- композитным выходом (видео/аудио): 3,5 мм (4 pin, на рисунке внизу справа от разъёма CSI-2 для камеры);
- USB/Ethernet Hub'ом – второй по величине чёрный квадрат на рисунке (с буквами SMSC), на который тоже надо поставить радиатор;
- 4 портами USB 2.0 (на рисунке справа вверху и справа по центру), в которые можно подключать клавиатуру, мышь и другие устройства;
- сетевыми модулями: WiFi 802.11n, 10/100 Мб RJ45 Ethernet (на рисунке справа внизу);
- Bluetooth 4.1, Bluetooth Low Energy;
- разъёмом для дисплея: Display Serial Interface (DSI, на рисунке – слева, рядом с надписью DISPLAY);
- разъёмом для видеокамеры: MIPI Camera Serial Interface (CSI-2, на рисунке – внизу по центру, рядом с надписью CAMERA);
- разъёмом для карты памяти microSD (с левой стороны на обратной стороне платы, от 4 Гб), на которую устанавливаются и с которой загружаются образы операционных систем;
- 40 пинами/портами для ввода/вывода – больше, чем на любой другой плате (GPIO, на рисунке – вверху, между двумя дырками), включающими UART (Serial), I2C/TWI, SPI с селектором между двумя устройствами и пины земли и питания на 3,3 В и 5 В;
- разъёмом питания micro-USB (от адаптера на 5 В, входящего в комплект поставки [21], на рисунке разъём снизу слева, рядом с дыркой).

Обычно на microSD-карту уже установлены операционная система Raspbian OS и загрузчик Noobs, с помощью которого можно установить другую систему. Операционные системы, которые поддерживаются платой и устанавливаются с помощью копирования образа на microSD-карту, благодаря архитектуре процессора ARMv53 могут быть совершенно разнообразными: Debian Wheezy, Ubuntu Mate, Fedora Remix, MS Windows 10, Android Things, Windows 10 IoT Core, Android, OSMC, LIBREELEC, PINET, RISC OS, ICHIGOJAM RPI и т. д. Также можно установить Raspberry Pi Desktop для платформ PC и Mac. Полный список поддерживаемых операционных систем, а также многое другое можно найти на официальном сайте https://www.raspberrypi.org.

Мы будем работать с Raspberry Pi 3 Model B в связи с тем, что, кроме всего прочего, она официально поддерживается операционной системой Android

Things (см. [16]), для которой мы будем разрабатывать приложения в этой части учебного пособия.

Установка ОС Android Things

0. Вставьте Raspberry Pi 3 в пластмассовый корпус. Будьте осторожны! Не используйте грубую силу!

1. Ознакомьтесь с начальной информацией об операционной системе Android Things: https://developer.android.com/things/get-started/.

2. Чтобы скачать и установить образ Android Things, следуйте инструкциям на странице https://developer.android.com/things/hardware/raspberrypi.html#flashing_the_image:

1) перейдите на страницу https://partner.android.com/things/console и загрузите Setup Utility (версия 1.0.21 на данный момент; требуется аккаунт Google (gmail)). По желанию вы можете создать новый продукт, см. инструкцию здесь: https://developer.android.com/things/console/create;
2) разархивируйте файл android-things-setup-utility.zip и запустите соответствующий файл. В окне консоли нажмите 1 (install Android Things and optionally set up Wi-Fi). Затем нажмите 1 снова для выбора Raspberry Pi 3 в качестве платы. И затем – ещё раз 1 (Choosing a generic image of Android Things for flashing the board);
3) найдите в наборе microSD-карту на 16 Гб, вставьте её в больший адаптер, а его – в кардридер, затем подключите кардридер к USB, как просит программа установки, и нажмите **Enter** (Plug the SD card into your computer. Press [Enter] when ready). Выберите диск (например, \\.\PHYSICALDRIVE3 (15.9 GB) – Generic-SD/MMC USB Device) и нажмите **Enter**. Вы увидите сообщение: **This will erase the selected drive. Are you sure? (Y/N)** – нажмите **Y**. Затем вы должны увидеть следующее: **Flashing [ ] 1% eta 7m11s**. Подождите указанное время. Далее идёт процесс валидации, и когда он закончится, на вопрос **Would you like to set up Wi-Fi on this device? (Y/N)** выберите **N**. Нажмите **Enter** для выхода из программы установки;
4) когда программа установки закончит запись образа на microSD-карту, извлеките кардридер и выньте из него адаптер с картой, а затем выньте карту из адаптера. Вставьте карту в microSD-слот на обратной стороне платы Raspberry Pi 3;
5) подключите плату Raspberry Pi 3 с помощью кабеля USB Am-micro-USBm к адаптеру питания (miniUSB-слот на плате), затем подключите к плате Ethernet-кабель в слот RJ-45 платы, потом подключите HDMI-кабель к плате и к внешнему дисплею (если у дисплея нет HDMI-выхода, используйте переходник HDMI-VGA и кабель VGA m-m). При этом, если у вас обычный монитор без сенсорного экрана, понадобится USB-мышь, которую надо подключить к одному из USB-портов Raspberry Pi 3. Если у вас монитор с сенсорным экраном, нежно подключить кабель USB-USB

к входу сенсорного управления на мониторе и к плате и воспользоваться сенсорным экраном монитора. Включите адаптер питания и монитор в сеть и убедитесь, что вы видите на мониторе заставку операционной системы Android Things. Если вы её не видите, отключите всё от питания, выньте microSD-карту из слота платы и повторите шаги 2.2–2.5;

6) пропустите установку Android Things Toolkit. После загрузки вы должны увидеть начальный экран Android Things (Overview). В разделе меню menu/system/system updates/check for update вы должны видеть, что установлена система последней версии (System version 8.1.0; Security patch level: Aug. 5, 2018). **Запомните IP-адрес, который отображается в разделе Network!** (например: 192.168.1.106).

Пины Raspberry Pi 3:

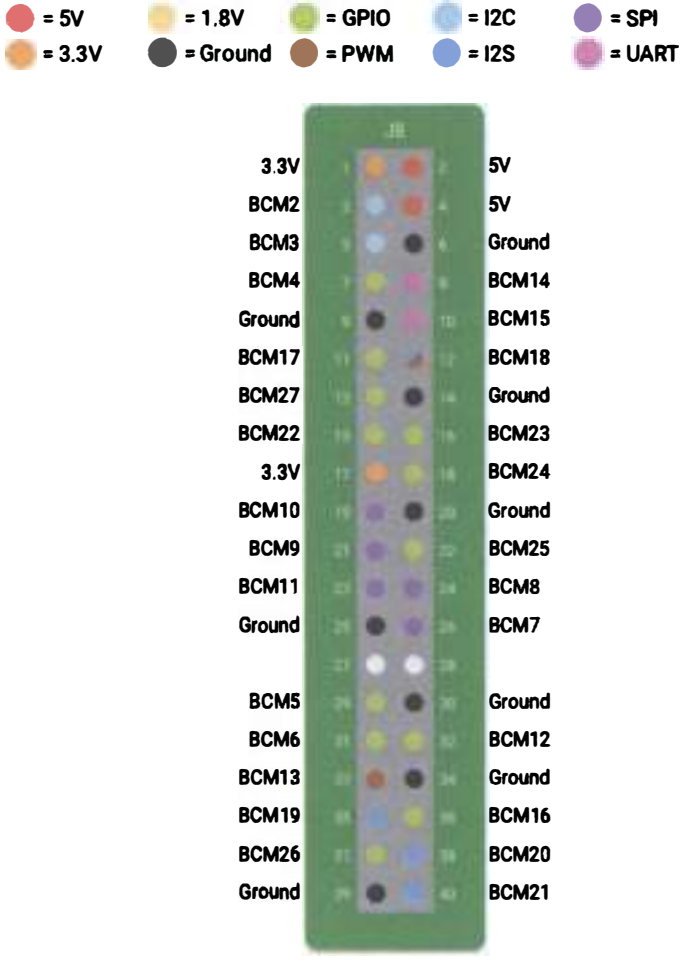


Рис. 112 ❖ Пины платы Raspberry Pi 3 [16]

Первый проект в ОС Android Things – трёхцветный светодиод

0. Создайте новый проект в Android Studio (версии 3.1.4; если у вас не установлена среда Android Studio, перейдите на сайт developer.android.com, загрузите и установите её). Дайте имя проекту, в следующем окне выберите дополнительно с Phone and Tablet платформу Android Things внизу и minimum SDK установите как API 24 (устанавливается по умолчанию). В следующем окне выберите Empty Activity, дайте имя activity, затем в следующем окне выберите Android Things Empty Activity; нажмите **Next** и **Finish**.

Вообще, конкретно в этом проекте нам не нужно выбирать Phone and Tablet-приложение, но для интернета вещей такой выбор естественен: как правило, приложения для интернета вещей используют следующую архитектуру: датчики/сенсоры – плата SoM (System on Module) – приложение, работающее на этой плате – интернет/облако/сервер/хранилище данных/облачная платформа – мобильное или иное приложение с пользовательским интерфейсом. Поэтому здесь и в дальнейшем рекомендуется создавать как Android Things-приложение, так и приложение для Phone и Tablet (смартфона и планшета).

1. Нам понадобятся:
   - соединительные провода,
   - 3 резистора на 220 Ом;
   - 3 резистора на 10 кОм;
   - макетная плата;
   - трёхцветный светодиод (с общим катодом, возьмите его из набора с Arduino Uno [1]);
   - 3 кнопки;
   - Raspberry Pi 3;
   - монитор с HDMI-выходом;
   - кабель HDMI m-m;
   - адаптер питания / кабель питания для HDMI-монитора;
   - адаптер питания для Raspberry Pi 3 (5 В).

Перед подключением всех компонентов к плате убедитесь, что она отключена от питания, иначе вы можете повредить её или компоненты.

Следующая схема с макетной платой показывает, как нужно соединить все компоненты и подключить их к Raspberry Pi 3 (рис. 113).

Стягивающий резистор номиналом 10 кОм соединяет один выход кнопки с землёй. У каждой кнопки есть такой стягивающий резистор. Резистор номиналом 220 Ом соединяет каждый из 3 пинов (анод, отвечающий за определённый цвет) трёхцветного светодиода с пином на плате, ограничивая ток, текущий через светодиод. Четвёртый пин светодиода, или общий катод, соединяется с пином земли на плате Raspberry Pi 3.

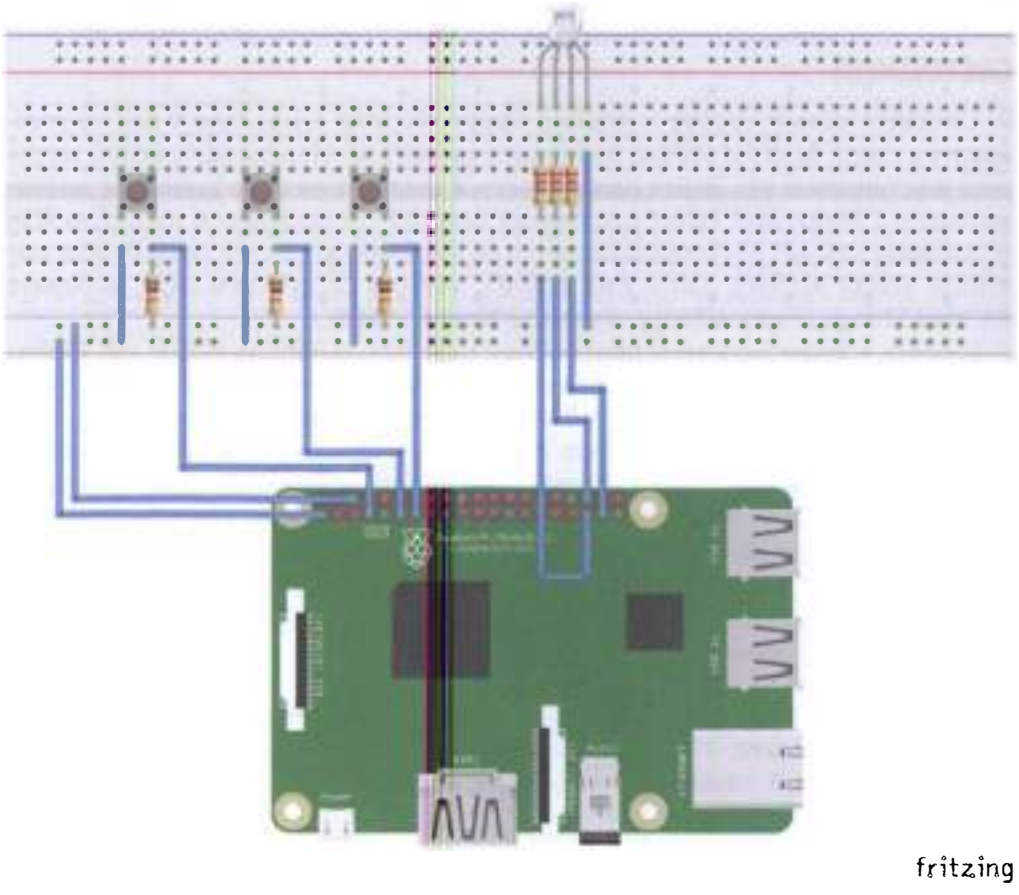


Рис. 113 ❖ Схема подключения для задания 1

2. Откройте файл в папке manifests -> AndroidManifest.xml и добавьте следующую строку перед тегом application (это нужно для того, чтобы получить разрешение обращаться к пинам платы):

```
<uses-permission android:name="com.google.android.things.permission.USE_PERIPHERAL_IO" />
```

3. Откройте файл MainActivity.java (в папке things -> название пакета, например things -> com.example.me.IoTbook1) и добавьте следующие строки перед методом onCreate в раздел объявлений переменных класса:

```
private Gpio mREDLedGpio, mBlueLedGpio, mGREENLedGpio, mREDButtonGpio, mBlueButtonGpio,
mGREENButtonGpio;;
boolean redPressed = false;
boolean greenPressed = false;
boolean bluePressed = false;
TextView text1;
```

Эти переменные понадобятся нам далее в коде. У нас в проекте есть интерфейс (things –> res –> main_activity.xml) для нашего приложения, и мы будем менять цвет и текст элемента TextView, когда будет нажата одна из трёх кнопок. Чтобы инициализировать переменную text1, добавьте в метод onCreate следующую строку (при этом убедитесь, что в свойствах элемента TextView атрибут id=text1):

```
text1 = findViewById(R.id.text1);
```

Следующий код в методе onCreate работает только для одной кнопки и одного красного цвета трёхцветного светодиода. Вам нужно самостоятельно выяснить названия пинов и дописать код для остальных цветов светодиода и оставшихся двух кнопок. Чтобы выяснить названия и номера пинов, см. рис. 112 выше.

```
PeripheralManager manager = PeripheralManager.getInstance();
    try {
        mREDLedGpio = manager.openGpio("BCM19"); //red color
        // Step 2. Configure as an output.
        mREDLedGpio.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW); //is lit
        // Step 1. Create GPIO connection.
        mREDButtonGpio = manager.openGpio("BCM4"); //red color button
        // Step 2. Configure as an input.
        mREDButtonGpio.setDirection(Gpio.DIRECTION_IN);
        // Step 3. Enable edge trigger events.
        mREDButtonGpio.setEdgeTriggerType(Gpio.EDGE_FALLING);
        // Step 4. Register an event callback.
        mREDButtonGpio.registerGpioCallback(mREDCallback);
    } catch (IOException e) {
    Log.e("onCreate", "Error on PeripheralIO API", e);
}
```

Затем нам нужно добавить несколько методов вне метода onCreate для работы с трехцветным светодиодом и для корректного завершения работы нашего приложения:

```
// Step 4. Register an event callback.
private GpioCallback mREDCallback = new GpioCallback() {
    @Override
    public boolean onGpioEdge(Gpio gpio) {
        try {
            redPressed = !redPressed;
            mREDLedGpio.setValue(redPressed);
            text1.setTextColor(Color.RED);
            text1.setText("RED");
            Log.i("GpioCallback", "Red LED button was pressed");

        } catch (IOException e) {
            Log.e("onKeyDown", "Error on PeripheralIO API", e);
        }
        // Step 5. Return true to keep callback active.
        return true;
```

```
    }
};

@Override
protected void onDestroy() {
    super.onDestroy();

    // Step 6. Close the resource
    if (mREDButtonGpio != null) {
        mREDButtonGpio.unregisterGpioCallback(mREDCallback);
        try {
            mREDButtonGpio.close();
        } catch (IOException e) {
            Log.e("onDestroy", "Error on PeripheralIO API", e);
        }
    }
    if (mREDLedGpio != null) {
        try {
            mREDLedGpio.close();
        } catch (IOException e) {
            Log.e("onDestroy", "Error on PeripheralIO API", e);
        }
    }

}
```

4. Теперь добавьте код для остальных двух кнопок и зелёного и синего цветов светодиода. Исправляйте ошибки в коде по мере его добавления с помощью блоков try + catch, импорта классов и т. д.

5. Подключите к питанию HDMI-монитор и плату Raspberry Pi 3. Для запуска своего приложения (нажатием зелёной кнопки **Run** на панели инструментов Android Studio, при этом выделив MainActivity.java из папки things, не из папки mobile!) необходимо подключиться к плате, чтобы она была в списке устройств в ADB (Android Debug Bridge) tool. Для этого сначала вы должны найти Android Studio SDK-папку на вашем компьютере (C:\Users\UserName\AppData\Local\Android\Sdk по умолчанию), найдите папку platform tools и откройте консоль для папки platform-tools (для windows – **Shift**+клик правой кнопкой мыши по папке – открыть окно команд; для windows 10 – открыть окно **PowersShell** здесь); затем в окне команд напечатайте «adb connect IP», где IP – тот адрес, который вы запомнили из раздела «Введение» этой части пособия, он же отображается на начальном экране Android Things, который виден на HDMI-мониторе (например: adb connect 192.168.1.106; для windows 10 PowerShell: .\adb.exe connect 192.168.1.106); затем убедитесь, что плата подключена, с помощью команды «adb devices» (или «.\adb.exe devices») – вы увидите название платы в окне команд. После этого подключения по IP вы получите доступ к плате в среде Android Studio (в окне ADB после запуска приложения вы увидите что-то наподобие «Google Iot_rpi3 Android 7.0, API 24» и то же самое – внизу окна **Android Studio** в окне **Android Monitor**). В окне устройств, открытом

ADB, выберите плату (Google Iot_rpi3 Android 7.0, API 24); откажитесь или примите установку Instant Run, на ваш выбор.

6. После запуска приложение должно работать. Заметьте, что вы можете нажать две или даже три кнопки последовательно или одновременно, чтобы получить различные цвета с помощью трёхцветного светодиода, при этом последний выбранный цвет в виде его названия и цвета отображается в окне приложения на мониторе. Насладитесь своим первым приложением на плате Raspberry Pi 3 с пользовательским интерфейсом, информацией на котором вы управляете с помощью аппаратной составляющей приложения – нажатия кнопок!

Второй проект в Android Things – система сигнализации

0. В этом задании мы создадим проект из реальной жизни, использующий PIR (Passive InfraRed, пассивный инфракрасный) сенсор для обнаружения движения и затем уведомляющий пользователя через его смартфон (или другое мобильное устройство) с помощью дополнительного приложения-компаньона, написанного для смартфона и не зависящего от платы Raspberry Pi 3. Основными шагами будут следущие:

- PIR-сенсор сканирует область обнаружения с целью обнаружения движения;
- как только он обнаруживает движение, он уведомляет нашу плату (или приложение, которое на ней работает) под управлением операционной системы Android Things;
- плата (приложение) обрабатывает событие уведомления и соединяется с Google Firebase, чтобы отправить сообщение на смартфон пользователя.

Создайте новый проект в Android Studio. Дайте имя проекту, в следующем окне выберите дополнительно с Phone and Tablet платформу Android Things внизу и minimum SDK установите как API 24 (устанавливается по умолчанию). В следующем окне выберите Empty Activity, дайте имя activity, затем в следующем окне выберите Android Things Empty Activity; потом нажмите **Next** и **Finish.**

1. Нам понадобятся:

- соединительные провода,
- PIR-сенсор;
- плата Raspberry Pi 3;
- HDMI-монитор;
- кабель HDMI m-m;
- адаптер питания / кабель для HDMI-монитора;
- адаптер питания для платы Raspberry Pi 3 (5 В);
- ваш Google Firebase-аккаунт.

PIR-сенсор выглядит так:

Рис. 114 ❖ PIR-сенсор

Наиболее распространённая модель – та, которая использует линзы Френеля, которые помогают расширить область обнаружения движения. У сенсора есть два потенциометра: один для регулировки чувствительности, другой – для регулировки времени, в течение которого сигнал удерживается высоким (логическая единица) при обнаружении объекта. Это время меняется в диапазоне от 0.3 до 18 секунд. Максимальное расстояние обнаружения составляет 7 метров, угол обзора сенсора (FOV, Field Of View) равен 120 градусам. Рабочая температура может меняться в пределах от –15 до +70 °C.

Перед подключением всех компонентов к плате убедитесь, что она отключена от питания, иначе вы можете повредить её или компоненты.

Следующая схема с макетной платой показывает, как нужно подключить PIR-сенсор к Raspberry Pi 3 (рис. 115).

2. Откройте файл в папке manifests -> AndroidManifest.xml и добавьте следующую строку перед тегом application (это нужно для того, чтобы получить разрешение обращаться к пинам платы):

```
<uses-permission android:name="com.google.android.things.permission.USE_PERIPHERAL_IO" />
```

3. Откройте файл MainActivity.java (в папке things -> название пакета, например things -> com.example.me.IoTbook2) и добавьте следующие строки перед методом onCreate в раздел объявлений переменных класса:

```
private Gpio gpioPin;
private boolean status;
```

Эти переменные понадобятся нам далее в коде. Следующие строки кода нужны для установления соединения с сенсором: добавьте их в метод onCreate:

```
PeripheralManager manager = PeripheralManager.getInstance();
gpioPin = null;
try {
    gpioPin = manager.openGpio("BCM4"); // the signal pin of the PIR sensor is connected to
```

```
BCM4
    gpioPin.setDirection(Gpio.DIRECTION_IN); // we want to read values; in we want to write
values - DIRECTION_OUT
    gpioPin.setActiveType(Gpio.ACTIVE_HIGH); // the value is true if the pin is at high
voltage; LOW - true at low voltage
} catch (IOException e) {
    e.printStackTrace();
}
```

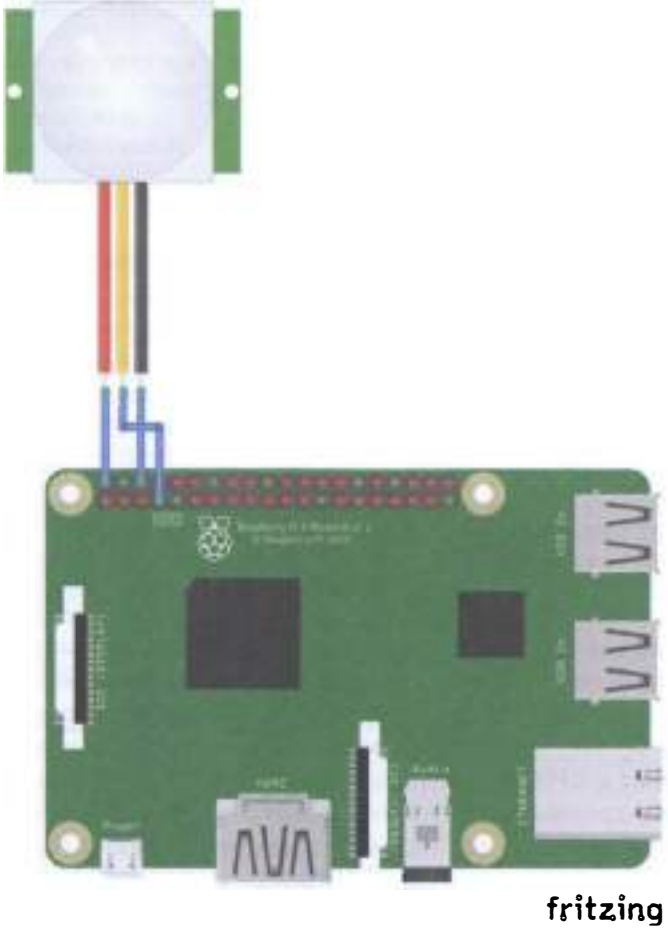


Рис. 115 ❖ Схема подключения для задания 2

Мы могли бы сделать следующее: создать отдельный поток и считывать состояние с пина всё время, как указано в следующем коде, помещённом в конец метода onCreate:

```
(new Thread(new Runnable() {
    @Override
```

```
    public void run() {
        try {
            while (true) {
                status = gpioPin.getValue(); // we read the state of the sensor: true or
false
                Log.d("Runnable","State ["+status+"]");
                if (status) {
                    Log.i("Runnable","Motion detected...");
                }
                Thread.sleep(5000); // every 5 seconds
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
})).start();
```

Если мы так сделаем, подключим плату и запустим приложение, то получим следующие строки в логе Logcat:

```
09-23 12:04:05.231 1862-1879/com.example.me.iotbook2 I/Runnable: Motion detected...
09-23 12:04:10.232 1862-1879/com.example.me.iotbook2 D/Runnable: State [false]
09-23 12:04:15.234 1862-1879/com.example.me.iotbook2 D/Runnable: State [true]
09-23 12:04:15.234 1862-1879/com.example.me.iotbook2 I/Runnable: Motion detected...
09-23 12:04:20.235 1862-1879/com.example.me.iotbook2 D/Runnable: State [false]
09-23 12:04:25.236 1862-1879/com.example.me.iotbook2 D/Runnable: State [false]
09-23 12:04:30.237 1862-1879/com.example.me.iotbook2 D/Runnable: State [true]
09-23 12:04:30.237 1862-1879/com.example.me.iotbook2 I/Runnable: Motion detected...
```

Хотя этот подход работает, он требует много временных ресурсов, потому что нашему Android Things-приложению приходится «мониторить» состояние пина всё время, даже если PIR-сенсор никого не обнаруживает. К счастью, есть другой подход, использующий механизм listeners (слушателей событий). Такой подход потребляет меньше ресурсов по времени и очень похож на подход, с помощью которого мы привыкли разрабатывать Android-приложения. Чтобы добавить listener к GPIO (General Purpose Input/Output, интерфейс ввода-вывода общего назначения), мы должны объявить событие, которое хотим прослушивать, и реализовать callback-класс для регистрации и обработки этого события (см. первый проект в Android Things – там то же самое).

Есть 4 типа изменяющихся событий: **EDGE_NONE** (не произошло никакого события), **EDGE_RISING** (увеличивающийся уровень напряжения, значение напряжения на пине меняется с логического 0 на 1), **EDGE_FALLING** (уменьшающийся уровень напряжения, значение напряжения на пине меняется с логической 1 на 0) и **EDGE_BOTH** (комбинация предыдущих двух событий: мы хотим знать, меняется ли сигнал хоть как-то – от 0 до 1 или от 1 до 0). В этом проекте нам нужно знать, меняется ли сигнал с низкого значения (0) на высокое значение напряжения (1), так как мы обнаруживаем движение; поэтому

в коде мы используем тип триггера EDGE_RISING. Добавьте следующую строку кода в метод onCreate в блок try catch:

```
gpioPin.setEdgeTriggerType(Gpio.EDGE_RISING);
```

В этот раз мы создадим новый callback-класс вместо добавления callback-метода. Добавьте следующий код в конец класса MainActivity.java:

```
public class SensorCallBack implements GpioCallback {
    @Override
    public boolean onGpioEdge(Gpio gpio) {
        try {
            boolean callBackState = gpio.getValue();
            Log.d("SensorCallBack","Callback state ["+callBackState+"]");
        } catch (IOException e) {
            e.printStackTrace();
        }
        return true;
    }

    @Override
    public void onGpioError(Gpio gpio, int error) {
        Log.e("SensorCallBack","GPIO error");
    }
}
```

Есть два важных метода, которые мы должны переопределить, чтобы настроить поведение callback-класса: public Boolean onGpioEdge и public Boolean onGpioError. Первый вызывается тогда, когда срабатывает событие, которое мы зарегистрировали с помощью setEdgeTriggerType. Второй срабатывает при условии возникновения ошибки при чтении данных с пина платы.

Добавьте следующую строку в секцию объявления переменных класса MainActivity:

```
SensorCallBack callback;
```

Наконец, мы должны зарегистрировать наш callback-класс. В методе onCreate добавьте следующий код в конце:

```
callback = new SensorCallBack();
try {
    gpioPin.registerGpioCallback(callback);
} catch (IOException e) {
    e.printStackTrace();
}
```

После того как вы сделали всё это, можно подключить плату к питанию и запустить приложение; как только вы махнёте рукой перед PIR-сенсором, в логе Logcat увидите следующее:

```
09-23 14:11:04.444 2452-2452/com.example.me.iotbook2 D/SensorCallBack: Callback state [true]
09-23 14:11:10.874 2452-2452/com.example.me.iotbook2 D/SensorCallBack: Callback state [true]
09-23 14:11:41.494 2452-2452/com.example.me.iotbook2 D/SensorCallBack: Callback state [true]
```

В завершение мы должны закрыть соединение с GPIO-пином платы. Это делается в методе onDestroy внутри класса MainActivity.java. Мы удаляем все listeners и закрываем соединение с GPIO-пинами (это называется корректное завершение приложения):

```
@Override
protected void onDestroy() {
    super.onDestroy();
    Log.d("MainActivity","onDestroy");
    if (gpioPin != null) {
        gpioPin.unregisterGpioCallback(callback);
        try {
            gpioPin.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        gpioPin = null;
    }
}
```

4. Что, если мы хотим, чтобы наше приложение работало на любой плате, которую поддерживает ОС Android Things, другими словами – что, если мы хотим разработать платонезависимое приложение? Это вполне возможно, так как существует константа Build.BOARD, хранящая автоматически определённый системой тип (название) платы, на которую установлена система. Используя эту информацию, мы можем выбрать название пина платы во время выполнения программы. Создайте новый Java-класс (things -> java -> щёлкните правой кнопкой мыши по названию пакета (package name) и выберите New -> Java Class), дайте ему название «BoardPins» и вставьте следующий код:

```
import android.os.Build;

import com.google.android.things.pio.PeripheralManager;

import java.util.List;

public class BoardPins {
    private static final String EDISON_ARDUINO = "edison_arduino";
    private static final String RASPBERRY  ="rpi3";
    public static String getPirPin() {
        switch (getBoardName()) {
            case RASPBERRY:
                return "BCM4";
            case EDISON_ARDUINO:
                return "IO4";
            default:
                throw new IllegalArgumentException ("Unsupported device");
        }
    }

    public static String getBoardName() {
        String name = Build.BOARD;
```

```
        if (name.equals("edison")) {
            PeripheralManager manager = PeripheralManager.getInstance();
            List<String> pinList = manager.getGpioList();
            if (pinList.size() > 0) {
                String pinName = pinList.get(0);
                if (pinName.startsWith("IO")) return EDISON_ARDUINO;
            }
        }
        return name;
    }
}
```

Название платы, возвращаемое с помощью Android Things SDK (Software Developer Kit), не помогает нам отличить различные варианты платы Intel Edison. Поэтому мы пролистываем список пинов и ищем в нём особое название пина, чтобы определить тип платы и её конкретный вариант.

После создания класса BoardPins можно открыть MainActivity.java и заменить строку

```
gpioPin = manager.openGpio("BCM4");
```

следующей строкой:

```
gpioPin = manager.openGpio(BoardPins.getPirPin());
```

Теперь наше приложение будет работать, по крайней мере, на двух платах, названия которых очевидны из кода. Можно даже изменить значение TextView в интерфейсе приложения: показать там название платы и имя пина, к которому подключён PIR-сенсор вместо стандартной фразы Hello World!:

```
TextView text1 = findViewById(R.id.text1);
text1.setText("This app runs on the "+BoardPins.getBoardName()+" board using "+BoardPins.
getPirPin()+" pin!");
```

5. Теперь мы займёмся системой уведомлений. Мы будем использовать сервис уведомлений Google Firebase, поэтому вы должны создать там аккаунт, если у вас его ещё нет. Есть несколько способов посылать уведомление из Android Things-приложения на смартфон пользователя. Мы будем использовать тему (**topic**). Topic похож на канал. После того как приложение подписывается на определённый topic, оно получит все сообщения, опубликованные в этом канале. В нашем проекте смартфон пользователя (точнее, приложение, которое на нём работает) ведёт себя как подписчик (**subscriber**), получающий сообщения из канала, в то время как Android Things-приложение ведёт себя как издатель/тот, кто публикует сообщения (**publisher**).

5.1. Перейдите на домашнюю страницу Firebase (https://firebase.google.com/) и щёлкните по Get Started, для того чтобы создать ваш аккаунт. Заполните все необходимые поля и создайте аккаунт.

5.2. Перейдите в консоль Firebase (https://console.firebase.google.com). Добавьте новый проект (нажмите на **Add/create a new project**). В появившемся окне введите название проекта и страну/регион по необходимости и выберите все

3 чекбокса (галочки). Нажмите **Create Project**. Затем немного подождите, пока создаётся проект, и нажмите **Continue**. Вы увидите консоль администратора для вашего проекта. Добавьте наше Android-приложение в этот проект с помощью нажатия на кнопку **Add firebase to your Android app** или на круглую иконку **Android**. В следующем окне надо добавить свойства Android Things-приложения. Введите название пакета (package name, его можно найти в файле манифеста AndroidManifext.xml в качестве значения переменной package, например: **package="com.example.me.iotbook2"**) и псевдоним для нашего приложения. Нажмите кнопку **Register application**. Затем загрузите файл google-services.json и поместите его в корень папки things в Android Studio: нажмите **Копировать на этом файле** (или выделите его и нажмите **Ctrl+C**) в папке, затем нажмите правой кнопкой мыши по папке things в Android Studio. Выберите любую папку, нажмите **OK** и затем в поле **To directory** выберите папку things, нажмите **OK**. Если переключиться в отображение Project view в Android Studio, вы увидите файл в папке things. Наконец, нужно нажать **Next** в консоли Firebase и добавить Firebase SDK в ваши файлы build.gradle. Добавьте строку **classpath 'com.google.gms:google-services:4.1.0'** в ваш файл build.gradle уровня проекта (Project) в раздел dependencies, а затем добавьте строку **implementation 'com.google.firebase:firebase-core:16.0.1'** в ваш файл build.gradle уровня модуля things (Module: things) в раздел dependencies; после этого добавьте строку **apply plugin: 'com.google.gms.google-services'** в самый конец того же файла (вне последнего символа }). Теперь нажмите **Sync Now** для синхронизации и построения проекта. После того как синхронизация завершится, нажмите **Next** в консоли Firebase и запустите своё приложение (папка things), чтобы проверить наличие связи между приложением и Firebase. Если всё в порядке, вы должны увидеть зелёное сообщение Congratulations в консоли Firebase.

5.3. Создайте следующий класс NotificationManager.java в вашем проекте (things – java – package – клик правой кнопкой мыши – New – Java Class):

```
import android.os.AsyncTask;
import android.util.Log;

import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;

public class NotificationManager {

    private static NotificationManager me = null;

    private NotificationManager() {}

    public static NotificationManager getInstance() {
        if (me == null)
            me = new NotificationManager();

        return me;
    }

    public void sendNotificaton(String message, String key) {
```

```
        (new FirebaseNotificationTask()).execute(new String[]{message, key});
    }

    private class FirebaseNotificationTask extends AsyncTask<String, Void, Void> {

        @Override
        protected Void doInBackground(String... strings) {
            String msg = strings[0];
            String key = strings[1];
            Log.d("Alm", "Send data");
            try {
                HttpURLConnection con = (HttpURLConnection) (new URL("http://fcm.
googleapis.com/fcm/send")).openConnection();
                con.setRequestMethod("POST");
                con.setRequestProperty("Authorization", "key=" + key);
                con.setRequestProperty("Content-Type", "application/json");
                con.setDoInput(true);
                con.setDoOutput(true);
                con.connect();

                String body = "{\n" +
                        "  \"to\": \"/topics/alarm\",\n" +
                        "  \"data\": {\n" +
                        "  \"message\": \"" + msg + "\"" +
                        "  }\n" +
                        "}";
                Log.d("Alm", "Body ["+body+"]");
                con.getOutputStream().write(body.getBytes());
                InputStream is = con.getInputStream();
                byte[] buffer = new byte[1024];
                while ( is.read(buffer) != -1)
                    Log.d("Alm", new String(buffer));
                con.disconnect();

            }
            catch(Throwable t) {
                t.printStackTrace();
            }

            return null;
        }
    }
}
```

Теперь откройте MainActivity.java и в методе onGpioEdge callback класса добавьте следующую строку после сообщения Log.d:

```
NotificationManager.getInstance().sendNotificaton("Alarm!", server_key);
```

где server_key – это ключ, который вы можете получить из консоли Firebase: перейдите в **Project Overview** в меню слева вверху с иконкой home; нажмите на псевдоним (alias) вашего проекта, созданный ранее (с помощью круглой иконки Android), затем нажмите на иконку настроек (шестерёнку) и потом в открывшемся окне перейдите на закладку **Cloud Messaging** – там вы найдёте значение server key (которое очень длинное).

5.4. Измените файл AndroidManifest.xml file в вашей папке things – добавьте следующие две строки перед тегом application, так как мы работаем с Firebase и нам нужен доступ в интернет:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Теперь наше Android Things-приложение готово посылать уведомления. Но нам нужно **приложение для смартфона**, чтобы получать эти уведомления. Такое приложение обычно называют приложением-компаньоном (**companion app**).

5.5. Наше Android-приложение-компаньон будет подписываться на канал, используемый нашим Android Things-приложением для отправки уведомлений; задействовать сервис для прослушивания входящих уведомлений; показывать уведомления пользователю. Самое интересное в том, что у нас уже есть мобильное приложение в нашем проекте – см. папку mobile. Всё, что нам нужно сделать, – это удалить стандартный элемент TextView со значением Hello World! из интерфейса приложения (mobile – res – layout – activity_main.xml) и добавить туда кнопку. В последней (на данный момент) версии Android Studio (3.1.4) существует ошибка с пользовательским интерфейсом на закладке **Design** в редакторе интерфейса приложения. Чтобы исправить эту ошибку, перейдите в файл build.gradle уровня приложения (Module: mobile) и замените строку

```
implementation 'com.android.support:appcompat-v7:28.0.0-rc02'
```

строкой

```
implementation 'com.android.support:appcompat-v7:27.1.1'
```

Затем синхронизируйте проект. После этого удалите TextView, добавьте кнопку (**Button**) и установите её свойство text в значение SUBSCRIBE через создание нового строкового ресурса. Затем вы должны скопировать **тот же самый** файл google-services.json, который вы скопировали ранее в папку things, в корень папки mobile проекта и изменить файл build.gradle уровня приложения (Module: mobile): добавьте строку **implementation 'com.google.firebase:firebase-core:16.0.1'** в файл build.gradle (Module: mobile) в раздел dependencies; после этого добавьте строку **apply plugin: 'com.google.gms.google-services'** в самый конец того же файла (вне последнего символа }). Дополнительно добавьте следующую строку в раздел dependencies и затем снова синхронизируйте проект:

```
implementation 'com.google.firebase:firebase-messaging:17.3.2'
```

Теперь добавьте следующий код в файл MainActivity.java в метод onCreate вашего мобильного приложения (папка mobile, а не things!):

```
Button but1 = (Button) findViewById(R.id.button);
but1.setOnClickListener(new View.OnClickListener() {
```

```
    @Override
    public void onClick(View v) {
        FirebaseMessaging.getInstance().subscribeToTopic("alarm");
        String tkn = FirebaseInstanceId.getInstance().getToken();

        Toast.makeText(MainActivity.this,"Device subscribed to the channel", Toast.LENGTH_
LONG).show();
        Log.d("App", "Token ["+tkn+"]");
    }
});
```

Нам нужно создать ещё два класса. Один будет отвечать за ввод токена:

```
import android.util.Log;

import com.google.firebase.iid.FirebaseInstanceId;
import com.google.firebase.iid.FirebaseInstanceIdService;

public class FireIDService extends FirebaseInstanceIdService {

    @Override
    public void onTokenRefresh() {
        String tkn = FirebaseInstanceId.getInstance().getToken();
        Log.d("Not","Token ["+tkn+"]");

    }
}
```

А другой класс будет реализовывать сервис, который обрабатывает входящее пуш-уведомление (push notification):

```
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.support.v4.app.NotificationCompat;
import android.util.Log;

import com.google.firebase.messaging.FirebaseMessagingService;
import com.google.firebase.messaging.RemoteMessage;

public class FireMsgService extends FirebaseMessagingService {

    @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
        super.onMessageReceived(remoteMessage);

        Log.d("Msg", "Message received ["+remoteMessage+"]");

        // Create Notification
        Intent intent = new Intent(this, MainActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 1410, intent,
                PendingIntent.FLAG_ONE_SHOT);

        String info = null;

        if (remoteMessage.getData().size() > 0) {
```

```
            info = remoteMessage.getData().get("message");
        }

        if (remoteMessage.getNotification() != null) {
            info = remoteMessage.getNotification().getBody();
        }

        NotificationCompat.Builder notificationBuilder = new NotificationCompat.
Builder(this)
                .setSmallIcon(R.drawable.ic_launcher_background)
                .setContentTitle("Message")
                .setContentText(info)
                .setAutoCancel(true)
                .setContentIntent(pendingIntent);

        NotificationManager notificationManager =
                (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

        notificationManager.notify(1410, notificationBuilder.build());
    }
}
```

Не стоит забывать и про объявление сервисов в файле AndroidManifest.xml. Добавьте в этот файл следующее внутри тега application:

```
<service
    android:name=".FireIDService">
    <intent-filter>
        <action android:name="com.google.firebase.INSTANCE_ID_EVENT"/>
    </intent-filter>
</service>

<service
    android:name=".FireMsgService">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT"/>
    </intent-filter>
</service>
```

Следующие строки, добавленные также внутрь тега application, устанавливают иконку уведомления и его цвет, которые мы будем видеть в панели уведомлений:

```
<meta-data
    android:name="com.google.firebase.messaging.default_notification_icon"
    android:resource="@drawable/ic_launcher_background" />
<meta-data
    android:name="com.google.firebase.messaging.default_notification_color"
    android:resource="@color/colorAccent" />
```

6. После этого всё, что вам нужно сделать, – это подключить ваш смартфон к компьютеру (режим отладки) и запустить приложение, выбрав ваш смартфон в качестве устройства для выполнения приложения в ADB, для того чтобы установить приложение на смартфон. Можно также сделать это по-другому – сгенерировать apk-файл приложения с помощью пункта главного меню Android Studio

Build -> Build APK(s), затем найти сгенерированный файл mobile-debug.apk, скопировать его на смартфон, установить приложение и открыть его. Нажмите на кнопку subscribe в вашем приложении, чтобы подписать ваше устройство (точнее, приложение) на канал уведомлений. Если плата Raspberry Pi 3 подключена к питанию и на ней запущено приложение из папки things, вы можете протестировать PIR-сенсор и получить уведомление на ваше мобильное устройство, которое подключено к интернету, где бы вы не находились. Как только система определит двигающийся объект, она свяжется с платформой Google Firebase посредством отправки сообщения с уведомлением. В ответ платформа Firebase пошлёт это сообщение на смартфон пользователя (вы должны увидеть зелёный квадрат в трее сообщений). Как правило, уведомление приходит сразу после запуска обоих приложений, так как вы будете находиться рядом с PIR-сенсором.

Третий проект в Android Things – система мониторинга окружающей среды

0. В этом проекте мы построим сложную систему интернета вещей с помощью операционной системы Android Things, которая измеряет некоторые физические параметры окружающей среды. Этот проект фокусируется на том, как использовать I2C (Inter-Integrated Circuit) с помощью Android Things, как использовать класс Sensor Manager и визуализировать данные, получаемые с сенсоров, с помощью светодиодов. Целью этого проекта является построение системы мониторинга окружающей среды, которая определяет температуру и давление. Вы должны помнить подобные эксперименты из практики по Arduino, но в этот раз мы будем использовать другие сенсоры, и реакция светодиодов будет иной.

Создайте новый проект в Android Studio. Дайте имя проекту, в следующем окне выберите дополнительно с Phone and Tablet платформу Android Things внизу и minimum SDK установите как API 24 (устанавливается по умолчанию). В следующем окне выберите Empty Activity, дайте имя activity, затем в следующем окне выберите Android Things Empty Activity; нажмите **Next** и **Finish**.

1. Нам понадобятся:
   - соединительные провода;
   - сенсор BME280 (или BMP280, если есть);
   - 4 резистора на 220 Ом;
   - макетная плата;
   - трёхцветный светодиод (с общим катодом – он есть в наборе с Arduino Uno);
   - красный светодиод;
   - плата Raspberry Pi 3;
   - HDMI-монитор;
   - кабель HDMI m-m;
   - адаптер питания / кабель для HDMI-монитора;
   - адаптер питания для платы Raspberry Pi 3 (5 В).

Сенсор BME280 выглядит следующим образом:

Рис. 116 ❖ Первый вариант сенсора BME280

В других исполнениях он может выглядеть так:

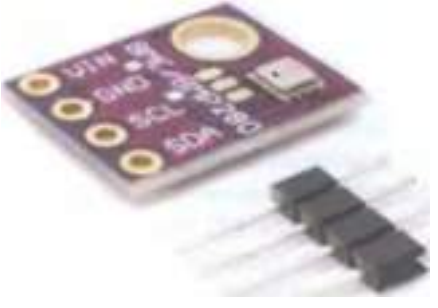


Рис. 117 ❖ Второй вариант сенсора BME280

Этот сенсор может измерять температуру, давление и влажность, в то время как сенсор BMP280 измеряет только температуру и влажность. Сенсор BME280 обладает следующими характеристиками:

- напряжение: 1.8–5 В (но может быть и максимум 3В, что должно быть написано на сенсоре; будьте осторожны и внимательны!);
- интерфейс: I2C (до 3.4 МГц), SPI (до 10 МГц);
- диапазон измеряемой температуры: от –40 °C до +85 °C;
- диапазон измеряемой влажности: от 0 до 100 %;
- диапазон измеряемого давления: от 300 до 1100 ГПа (мб);
- точность измерения температуры: 0.5 °C;

- точность измерения влажности: 3 %;
- точность измерения давления: 1 ГПа (мб).

Теперь немного о единицах измерения давления: 1 ГПа (Гектопаскаль) = $1*10^2$ Па = 100 Па, поэтому 1100 ГПа = 110 000 Па. 1 мб (миллибар) = 1 ГПа = 100 Па, поэтому 110 000 Па = 1100 мб. И наконец, 1 мм рт. ст. (миллиметр ртутного столба) = 1.33322387415 мб, поэтому, например, 735 мм. рт. ст. – это 980 мб, или 980 ГПа.

Перед подключением всех компонентов к плате убедитесь, что она отключена от питания, иначе вы можете повредить её саму или её компоненты.

Следующая схема с макетной платой показывает, как нужно соединить все компоненты и подключить их к Raspberry Pi 3 (рис. 7). В соответствии со схемой мы будем использовать следующие пины: Vin (пин на 3.3 В), GND (земля), SCK/SCL (тактовый сигнал, так как I2C-сенсор использует тактовый/синхронизирующий сигнал в своей работе) и SDI/SDA (пин данных). В некоторых исполнениях сенсора есть и другие пины, но мы не будем их использовать, так как мы будем подключать пины с помощью шины I2C.

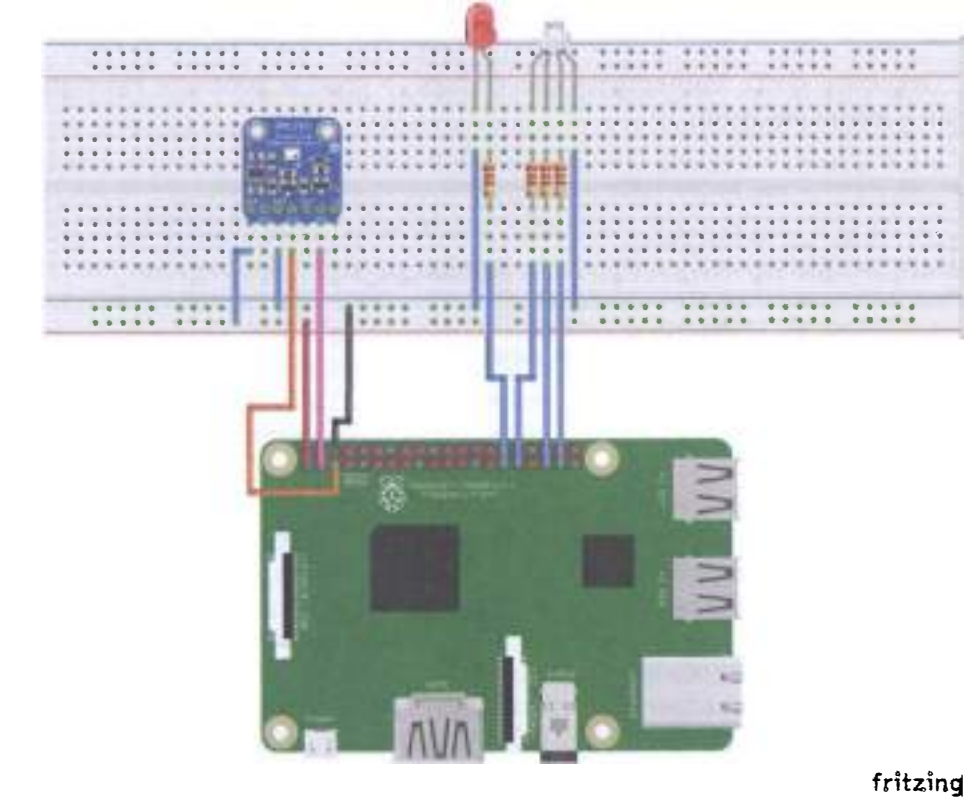


Рис. 118 ❖ Схема подключения для задания 3

2. Откройте файл manifests -> AndroidManifest.xml и добавьте следующую строку перед тегом application:

```
<uses-permission android:name="com.google.android.things.permission.USE_PERIPHERAL_IO" />
```

Обычно для использования I2C-периферии нам нужен драйвер. Драйвер – это набор классов, отвечающих за общение между платой с Android Things и периферией. Более того, эти классы обрабатывают специальные протоколы, используемые периферией. Все драйверы, которые официально поддерживаются ОС Android Things, доступны на GitHub в папке **contrib-drivers** по адресу https://github.com/androidthings/contrib-drivers. Откройте файл build.gradle (Module: things) и добавьте следующую строку в раздел dependencies:

```
implementation 'com.google.android.things.contrib:driver-bmx280:1.0'
```

Дополнительно вам придётся поменять minSdkVersion с 24 на 27 в этом файле. Синхронизируйте проект, и теперь вы можете использовать BME280/BMP280-сенсор в проекте. Так как мы используем драйвер для этого сенсора, мы должны добавить следующее разрешение (permission) в файл AndroidManifest.xml перед тегом application:

```
<uses-permission android:name="com.google.android.things.permission.MANAGE_SENSOR_DRIVERS" />
```

3. Откройте файл MainActivity.java (things -> java) и добавьте следующие строки в метод onCreate:

```
try {
    Bmx280 sensor = new Bmx280("I2C1", 0x76); //the class accepts the SDA pin id for
handling the sensor communication details which is I2C1 for Rpi3; 0x76 address means the
SDO pin is connected to the ground
    sensor.setTemperatureOversampling(Bmx280.OVERSAMPLING_1X); //sampling rate
    float temp = sensor.readTemperature();
    Log.i("onCreate", "Temperature: "+temp);
    sensor.setPressureOversampling(Bmx280.OVERSAMPLING_1X);
    float press = sensor.readPressure();
    Log.i("onCreate", "Pressure: "+press);
} catch (IOException e) {
    e.printStackTrace();
}
```

Заметьте, что имя пина – I2C1, а следующее значение установлено в 0x76. Это уникальный адрес устройства, который обычно устанавливается с помощью пина SDO, потому что у каждого периферийного устройства, поддерживающего соединение по I2C, есть свой адрес. Если на вашем сенсоре есть отдельный SDO-пин, есть два способа его подключения. Вы можете подключить этот пин к питанию (Vcc), тогда адрес I2C-сенсора будет 0x77; и вы можете подключить SDO пин к земле, тогда адрес будет 0x76. В коде выше адрес установлен в значение 0x76, что означает, что даже если у вас нет SDO-пина на сенсоре, внутри сенсора этот пин есть, и он подключён к земле. Теперь подключите плату, монитор и запустите приложение: в логе Logcat вы увидите температуру

и давление окружающей вас среды (заметьте, что сенсор показывает значение давления в миллибарах):

```
09-27 12:42:24.507 2132-2132/? I/zygote: Late-enabling -Xcheck:jni
09-27 12:42:24.800 2132-2132/com.example.me.iotbook3 I/InstantRun: starting instant run
server: is main process
09-27 12:42:25.113 2132-2132/com.example.me.iotbook3 I/onCreate: Temperature: 19.92114
09-27 12:42:25.119 2132-2132/com.example.me.iotbook3 I/onCreate: Pressure: 718.10345
```

По желанию вы можете добавить к этим значениям показания влажности; строки кода аналогичны приведённым выше.

4. В приведённом коде мы читаем данные с сенсора только один раз. В нашем проекте мы хотим читать их постоянно. Поэтому удобно будет использовать другой подход. Этот подход аналогичен тому, который мы используем в операционной системе Android, когда приложению нужно отслеживать показания сенсоров смартфона. Как вы знаете, в настоящее время у смартфона есть несколько встроенных сенсоров, и, чтобы читать с них данные, мы используем sensor framework, предоставляемый Android SDK.

Ключевыми элементами этого фреймворка являются SensorManager, Sensor, SensorEvent и SensorEventListener. Эти классы и интерфейсы также входят в состав Android Things SDK. **SensorManager** является базовым классом, когда мы хотим работать с сенсорами. Используя SensorManager, мы можем регистрировать или удалять регистрацию слушателей (listeners) либо получать список доступных сенсоров устройства. Класс **Sensor** представляет сенсор и его возможности. Класс **SensorEvent** описывает событие, зарегистрированное сенсором. Экземпляр класса SensorEvent содержит информацию о сенсоре, данные, снятые сенсором, точность этих данных и метку времени. Наконец, класс **SensorEventListener** представляет callback-класс, который вызывается, когда сенсор считывает новые данные, или точность измерения данных меняется. Мы будем использовать все эти классы и ещё один класс – **SensorManager.DynamicSensorCallback**. Он полезен тогда, когда мы хотим получать уведомления при подключении или отключении от нашей платы *динамического сенсора*.

Откройте файл MainActivity.java (things -> java); закомментируйте предыдущий код и добавьте следующие строки перед методом onCreate:

```
SensorManager sensorManager;
TemperatureCallback tempCallback;
PressureCallback pressCallback;
Bmx280SensorDriver mySensorDriver;
BMX280Callback callback;
```

Теперь нам нужно создать **callback**-класс сенсора. Так как мы хотим получать уведомления о двух параметрах (температуре и давлении), нам нужны два слушателя (listeners), по одному на каждый параметр. Для значений температуры, получаемых от сенсора, callback-класс будет выглядеть так:

```
private class TemperatureCallback implements SensorEventListener {

    @Override
    public void onSensorChanged(SensorEvent event) {
        float val = event.values[0];
        Log.i("TemperatureCallback","Temperature: "+val);
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        Log.i("TemperatureCallback","Accuracy: "+accuracy);
    }
}
```

А для значений давления – так:

```
private class PressureCallback implements SensorEventListener {

    @Override
    public void onSensorChanged(SensorEvent event) {
        float val = event.values[0];
        Log.i("PressureCallback","Pressure: "+val);
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        Log.i("PressureCallback","Accuracy: "+accuracy);
    }
}
```

Метод **onSensorChanged** вызывается, когда доступно новое значение параметра, прочитанное сенсором; метод **onAccuracyChanged** – когда меняется точность измерений этого параметра.

Теперь необходимо зарегистрировать эти пользовательские callback-классы, чтобы получать события. Это возможно только тогда, когда Android Things-приложение уведомлено о том, что сенсор подключён, иначе нельзя зарегистрировать слушателей событий. Вот здесь-то приложение и использует SensorManager.DynamicSensorCallback для обработки событий. Добавьте следующий код в MainActivity.java:

```
private class BMX280Callback extends SensorManager.DynamicSensorCallback {
    @Override
    public void onDynamicSensorConnected (Sensor sensor) {
        int sensorType = sensor.getType();
        Log.i("BMX280Callback","On Sensor connected...");
        if (sensorType==Sensor.TYPE_AMBIENT_TEMPERATURE) {
            Log.i("BMX280Callback", "Temp sensor...");
            tempCallback = new TemperatureCallback();
            sensorManager.registerListener(tempCallback,sensor,SensorManager.SENSOR_DELAY_
NORMAL);
        } else if (sensorType==Sensor.TYPE_PRESSURE) {
            Log.i("BMX280Callback", "Pressure sensor...");
            pressCallback = new PressureCallback();
```

```
            sensorManager.registerListener(pressCallback,sensor,SensorManager.SENSOR_DELAY_
NORMAL);
        }
    }

    @Override
    public void onDynamicSensorDisconnected (Sensor sensor) {
        super.onDynamicSensorDisconnected(sensor);
    }
}
```

Существует 4 возможных значения для частоты снятия показаний с сенсора: SENSOR_DELAY_NORMAL – задержка около 2 секунд между показаниями; SENSOR_DELAY_UI – задержка в 0.6 секунды; SENSOR_DELAY_GAME – задержка в 0.2 секунды; SENSOR_DELAY_FASTEST – никакой задержки.

5. Чтобы собрать всё вместе, откройте файл MainActivity.java (things -> java) и в методе onCreate добавьте следующее (в коде есть одна строка с ошибкой, найдите её и исправьте в соответствии с вашими знаниями, полученными в предыдущем разделе этого задания):

```
sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
callback = new BMX280Callback();
sensorManager.registerDynamicSensorCallback(callback);
try {
    mySensorDriver = new Bmx280SensorDriver(BoardPins.getSDAPin());
    mySensorDriver.registerTemperatureSensor();
    mySensorDriver.registerPressureSensor();
} catch (IOException e) {
    e.printStackTrace();
}
```

Заметьте, что, для того чтобы сделать наше приложение платонезависимым, мы используем новый метод в классе BoardPins.java, который мы создали в предыдущем задании; поэтому вам нужно скопировать этот класс из предыдущего проекта Android Studio и добавить в класс метод getSDAPin:

```
public static String getSDAPin() {
    switch (getBoardName()) {
        case RASPBERRY:
            return "I2C1";
        case EDISON_ARDUINO:
            return "IO4";
        default:
            throw new IllegalArgumentException("Unsupported device");
    }
}
```

И наконец, нам нужно закрыть соединение с сенсором, чтобы освободить пин SDA, используемый для общения с сенсором, чтобы другие приложения тоже могли через него обратиться к сенсору. Нам нужно удалить регистрацию слушателя значений от сенсора (sensor listener), удалить регистрацию слушателя определения статуса подключения сенсора к Android Things-плате и за-

крыть соединение с сенсором. Для этого добавьте следующий метод onDestroy в конец класса MAinActivity.java:

```
@Override
protected void onDestroy () {
    super.onDestroy();
    Log.i("MainActivity","onDestroy");
    sensorManager.unregisterListener(tempCallback);
    sensorManager.unregisterListener(pressCallback);
    sensorManager.unregisterDynamicSensorCallback(callback);
    try {
        mySensorDriver.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Теперь вы можете подключить плату и монитор к питанию и проверить, работает ли приложение. Сенсор надо придерживать, чтобы был стабильный контакт с пинами, как вы делали это с жидкокристаллическим монитором или RFID-модулем в ходе очередного задания по практике с Arduino, иначе значения, получаемые от сенсора, будут странные (как значения давления в следующем логе). В логе приложения будет видно следующее:

```
09-27 15:05:59.791 1972-1972/com.example.me.iotbook3 I/BMX280Callback: On Sensor
connected...
09-27 15:05:59.792 1972-1972/com.example.me.iotbook3 I/BMX280Callback: Temp sensor...
09-27 15:05:59.807 1972-1972/com.example.me.iotbook3 I/BMX280Callback: On Sensor
connected...
    Pressure sensor...
09-27 15:05:59.821 1972-1972/com.example.me.iotbook3 I/TemperatureCallback: Accuracy: 3
    Temperature: 29.686142
09-27 15:05:59.840 1972-1972/com.example.me.iotbook3 I/PressureCallback: Accuracy: 3
09-27 15:05:59.841 1972-1972/com.example.me.iotbook3 I/PressureCallback: Pressure:
729.39264
09-27 15:05:59.856 1972-1972/com.example.me.iotbook3 D/vndksupport: Loading /vendor/lib/
hw/android.hardware.graphics.mapper@2.0-impl.so from current namespace instead of sphal
namespace.
09-27 15:05:59.872 1972-1972/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.701126
09-27 15:05:59.872 1972-1972/com.example.me.iotbook3 I/PressureCallback: Pressure: 729.4043
09-27 15:05:59.912 1972-1972/com.example.me.iotbook3 I/PressureCallback: Pressure: 729.4217
09-27 15:05:59.939 1972-1972/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.706121
09-27 15:05:59.955 1972-1972/com.example.me.iotbook3 I/PressureCallback: Pressure: 729.4217
09-27 15:05:59.982 1972-1972/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.72111
09-27 15:05:59.998 1972-1972/com.example.me.iotbook3 I/PressureCallback: Pressure: 729.4332
```

Теперь вы знаете, как использовать I2C-сенсор в Android Things. Заключительный этап этого задания – в том, чтобы создать пользовательскую (кастомную) логику с помощью светодиодов, которая основана на показаниях сенсора.

6. Трёхцветный светодиод должен показывать текущее состояние по давлению, а красный светодиод показывает, не превысила ли определённый порог температура. Предположим, что есть два порога по давлению: LEVEL_1 = 1022.9 мб, или 767.2 мм рт. ст.; LEVEL_2 = 1009.14 мб, или 756.92 мм рт. ст. Логика нашего приложения будет следующей:

- если текущее давление больше LEVEL_1, то трёхцветный светодиод будет светить зелёным и красным цветом (т. е. жёлтым);
- если текущее давление между LEVEL_1 и LEVEL_2, то трёхцветный светодиод будет светить зелёным цветом;
- если текущее давление ниже LEVEL_2, то трёхцветный светодиод будет светить синим цветом.

Таким образом, трёхцветный светодиод будет представлять прогноз погоды:

- если давление выше LEVEL_1, погода будет стабильной;
- если давление между LEVEL_1 и LEVEL_2, погода будет облачной;
- если давление ниже LEVEL_2, погода будет дождливой.

Красный светодиод будет использоваться как сигнализация. Он будет загораться, если температура в помещении стала выше 30 °C.

Откройте файл MainActivity.java (things -> java) и добавьте следующие переменные перед методом onCreate:

```
PeripheralManager pManager;
Gpio redPin, greenPin, bluePin, redLedPin;
```

В методе onCreate добавьте следующую строку:

```
pManager = PeripheralManager.getInstance();
```

Добавьте следующий метод в тот же класс:

```
private void initRGBPins() {
        try {
            redPin = pManager.openGpio(BoardPins.getRedPin());
            redPin.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
            redPin.setActiveType(Gpio.ACTIVE_HIGH);
            greenPin = pManager.openGpio(BoardPins.getGreenPin());
            greenPin.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
            greenPin.setActiveType(Gpio.ACTIVE_HIGH);
            bluePin = pManager.openGpio(BoardPins.getBluePin());
            bluePin.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
            bluePin.setActiveType(Gpio.ACTIVE_HIGH);
            redLedPin = pManager.openGpio(BoardPins.getRedLedPin());
            redLedPin.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
            redLedPin.setActiveType(Gpio.ACTIVE_HIGH);
        } catch (IOException e) {
            e.printStackTrace();
        }
}
```

Этот метод открывает сообщение между пинами платы и подключёнными к ним светодиодами, устанавливает тип пинов в режим записи и затем уста-

навливает значения (напряжения) пинов (ACTIVE_HIGH – мы используем это значение для трёх цветов, так как у нас трёхцветный светодиод с общим катодом, поэтому ток течёт от GPIO-пинов платы на землю (пин земли) через общий катод; если у вас трёхцветный светодиод с общим анодом, вам нужно использовать значение ACTIVE_LOW, так как ток будет течь в обратном направлении – от питания 3.3 В через общий анод к GPIO-пинам платы, которые представляют собой землю (пины заземления)).

Как вы, наверное, заметили, мы используем ещё 4 метода из класса BoardPins.java, поэтому вы должны добавить их в него:

```
public static String getRedPin() {
    switch (getBoardName()) {
        case RASPBERRY:
            return "BCM6";
        case EDISON_ARDUINO:
            return "IO4";
        default:
            throw new IllegalArgumentException("Unsupported device");
    }
}

public static String getBluePin() {
    switch (getBoardName()) {
        case RASPBERRY:
            return "BCM26";
        case EDISON_ARDUINO:
            return "IO4";
        default:
            throw new IllegalArgumentException("Unsupported device");
    }
}

public static String getGreenPin() {
    switch (getBoardName()) {
        case RASPBERRY:
            return "BCM19";
        case EDISON_ARDUINO:
            return "IO4";
        default:
            throw new IllegalArgumentException("Unsupported device");
    }
}

public static String getRedLedPin() {
    switch (getBoardName()) {
        case RASPBERRY:
            return "BCM5";
        case EDISON_ARDUINO:
            return "IO4";
        default:
            throw new IllegalArgumentException("Unsupported device");
    }
}
```

Затем добавьте следующую строку в метод onCreate:

```
initRGBPins();
```

Теперь нам необходимо реализовать логику приложения. Мы сделаем это в методах слушателей сенсора (sensor listener). Сначала изменим код слушателя значений давления от сенсора: добавим в раздел объявления переменных следующее:

```
private static final float LEVEL_1 = 1022.69f; //or 767.2f mm Hg
private static final float LEVEL_2 = 1009.14f; //or 756.92f mm Hg
private int currentWeather = -100;
```

А в метод **onSensorChanged** класса **PressureCallback** – следующее:

```
int newWeather = -200;
if (val >= LEVEL_1)
    newWeather = 1;
else if (val >= LEVEL_2 && val <= LEVEL_1)
    newWeather = 0;
else
    newWeather = -1;

if (newWeather != currentWeather) {
    currentWeather = newWeather;
    // Set the RGB color
    switch (newWeather) {
        case 1:
            setRGBPins(true, true, false);
            break;
        case 0:
            setRGBPins(false, true, false);
            break;
        case -1:
            setRGBPins(false, false, true);
            break;
    }
}
```

У нас нет метода setRGBPins, используемого, чтобы избежать изменения цвета трёхцветного светодиода каждый раз, когда сенсор считывает новое значение. Вместо этого приложение просто проверяет, может изменить цвет светодиода новое значение давления или нет; и если да – меняет с помощью вызова метода setRGBPins:

```
private void setRGBPins(boolean red, boolean green, boolean blue) {
    try {
        Log.i("setRGBPins", "Change RGB led color. Red ["+red+"] - Green ["+green+"] - Blue
["+blue+"]");
        redPin.setValue(red);
        greenPin.setValue(green);
        bluePin.setValue(blue);
    }
```

```
    catch (IOException ioe) {
        ioe.printStackTrace();
    }
}
```

И наконец, мы должны реализовать логику работы красного светодиода. Для этого надо поменять код класса, отвечающего за слушатель показаний температуры. Сначала добавим следующую переменную в раздел объявления переменных:

```
boolean currentState = false;
```

И затем перейдём в метод **onSensorChanged** класса **TemperatureCallback** и допишем следующий код:

```
boolean turnOn = false;
if (val >= 30)
    turnOn = true;
else
    turnOn = false;

if (currentState != turnOn) {
    Log.d("TemperatureCallback", "Change RED led color. New state ["+turnOn+"]");
    try {
        redLedPin.setValue(turnOn);
        currentState = turnOn;
    }
    catch(IOException ioe) {
        ioe.printStackTrace();
    }
}
```

7. Кроме того, можно выводить текущие значения температуры и давления в пользовательский интерфейс приложения – вместо текстового значения Hello World! элемента TextView мы можем добавить ещё один TextView и менять их значения соответственно (things -> res -> layout -> activity_main.xml). Также мы можем добавить простой прогноз погоды, используя третий TextView в соответствии с логикой нашего приложения. После редактирования пользовательского интерфейса добавьте следующие переменные в начало класса MainActivity.java:

```
TextView text1, text2, text3;
```

В методе onCreate добавьте инициализацию этих переменных (помните о том, что надо установить сначала их ID в свойствах):

```
text1 = findViewById(R.id.text1); //temperature
text2 = findViewById(R.id.text2); //pressure
text3 = findViewById(R.id.text3); //simple weather forecast
```

И затем в методах onSensorChanged соответствующих классов-слушателей добавьте:

```
text1.setText("Temperature: "+String.valueOf(Math.round(val))+" C");// to the
TemperatureCallback class
text2.setText("Pressure: "+String.valueOf(Math.round(val))+" millibars"); // to the
PressureCallback class
```

Что касается элемента text3, измените конструкцию if – elseif – else метода onSensorChanged класса PressureCallback следующим образом (можно перевести на русский язык):

```
if (val >= LEVEL_1) {
    newWeather = 1;
    text3.setText("The weather will be stable");
}
else if (val >= LEVEL_2 && val <= LEVEL_1) {
    newWeather = 0;
    text3.setText("The weather will be cloudy");
}
else {
    newWeather = -1;
    text3.setText("The weather will be rainy");
}
```

Теперь можно протестировать наше приложение! Подключите к питанию Android Things-плату и монитор и запустите приложение. В логе Logcat (и в интерфейсе приложения) вы должны увидеть что-то, похожее на следующее:

```
09-28 11:11:58.348 1878-1878/com.example.me.iotbook3 I/BMX280Callback: On Sensor
connected...
    Temp sensor...
09-28 11:11:58.362 1878-1878/com.example.me.iotbook3 I/BMX280Callback: On Sensor
connected...
    Pressure sensor...
09-28 11:11:58.376 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Accuracy: 3
09-28 11:11:58.377 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
28.92688
09-28 11:11:58.395 1878-1878/com.example.me.iotbook3 I/PressureCallback: Accuracy: 3
09-28 11:11:58.396 1878-1878/com.example.me.iotbook3 I/setRGBPins: Change RGB led color.
Red [false] - Green [false] - Blue [true]
09-28 11:11:58.399 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.9729
09-28 11:11:58.430 1878-1878/com.example.me.iotbook3 D/vndksupport: Loading /vendor/lib/
hw/android.hardware.graphics.mapper@2.0-impl.so from current namespace instead of sphal
namespace.
09-28 11:11:58.460 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
28.94686
09-28 11:11:58.462 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
28.98682
09-28 11:11:58.465 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.9318
09-28 11:11:58.472 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure:
979.95435
09-28 11:11:58.520 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.01679
09-28 11:11:58.522 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.9579
09-28 11:11:58.553 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.061749
```

```
09-28 11:11:58.555 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.95703
09-28 11:11:58.585 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.08173
09-28 11:11:58.617 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.9202
09-28 11:11:58.625 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.116695
09-28 11:11:58.656 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure:
979.92365
09-28 11:11:58.687 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.121689
09-28 11:11:58.689 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.95514
```

Если температура достигает значения 30 °C (потому что вы придерживаете сенсор рукой), в логе появится соответствующее сообщение об изменении статуса красного светодиода:

```
09-28 11:12:14.684 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.985851
09-28 11:12:14.701 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 980.0391
09-28 11:12:14.731 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.995844
09-28 11:12:14.749 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 980.0391
09-28 11:12:14.786 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure:
979.98315
09-28 11:12:14.816 1878-1878/com.example.me.iotbook3 D/TemperatureCallback: Change RED led
color. New state [true]
09-28 11:12:14.817 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
30.00084
09-28 11:12:14.834 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.9271
09-28 11:12:14.872 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.935
09-28 11:12:14.900 1878-1878/com.example.me.iotbook3 D/TemperatureCallback: Change RED led
color. New state [false]
09-28 11:12:14.902 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.995844
09-28 11:12:14.919 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.991
09-28 11:12:14.959 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure:
979.96313
09-28 11:12:14.987 1878-1878/com.example.me.iotbook3 D/TemperatureCallback: Change RED led
color. New state [true]
09-28 11:12:14.988 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
30.005833
09-28 11:12:15.005 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.9709
09-28 11:12:15.045 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.991
09-28 11:12:15.089 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 980.0067
09-28 11:12:15.132 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure:
979.97876
09-28 11:12:15.175 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 980.0067
09-28 11:12:15.219 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.9866
09-28 11:12:15.249 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
30.01582
09-28 11:12:15.266 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure:
979.96643
09-28 11:12:15.290 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
30.010828
```

А в пользовательском интерфейсе приложения вы увидите что-то вроде этого: округлённые значения температуры и давления + простой прогноз погоды. Сегодня 28 сентября 2018 года, и дождь идёт целый день, так что прогноз работает! На рис. 119 показан скриншот работающего на Raspberry Pi 3 приложения:

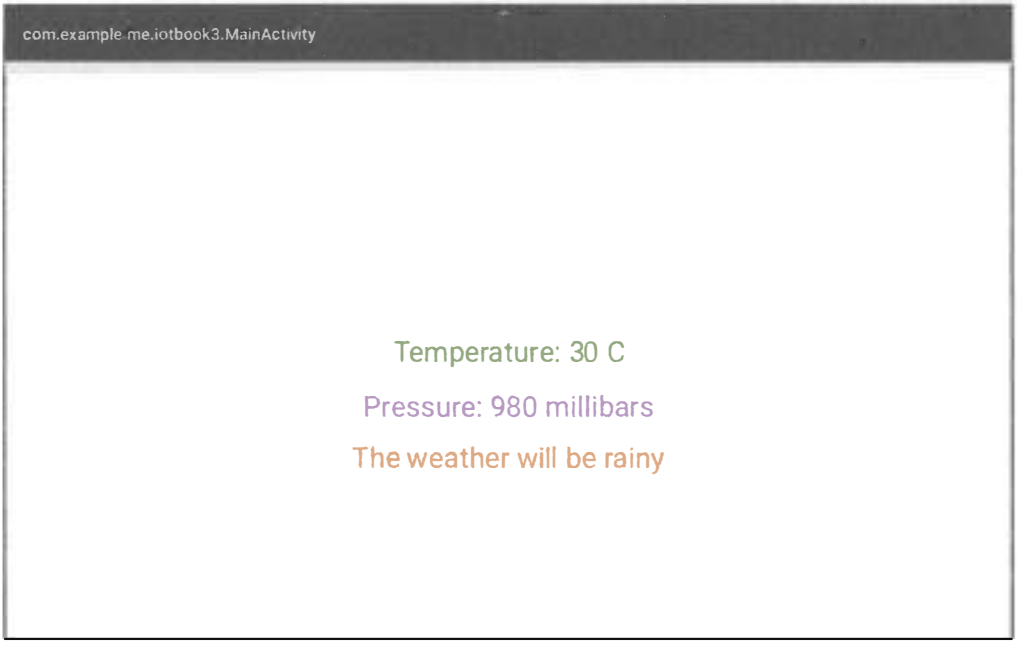


Рис. 119 ❖ Интерфейс Android Things-приложения мониторинга окружающей среды

8. Теперь вы, наверное, думаете, что задание, наконец, выполнено, но – нет! Мы могли бы создать пользовательский (кастомный) драйвер сенсора для класса BmxSensorDriver.java, но он уже содержит способность читать значения влажности с сенсора BME280, так как он содержит внутри класс HumidityUserDriver – вы, вероятно, уже видели эту возможность считывать влажность, когда писали код приложения. Поэтому было бы логично добавить измерение влажности в наше приложение. Вам придётся сделать это самостоятельно! Когда вы закончите, в логе вашего Android Things-приложения будет что-то вроде:

```
09-28 12:25:17.464 1618-1618/com.example.me.iotbook3 I/SensorManager: DYNS native
SensorManager.getDynamicSensorList return 3 sensors
09-28 12:25:17.647 1618-1618/com.example.me.iotbook3 I/BMX280Callback: On Sensor
connected...
    Temp sensor...
09-28 12:25:17.672 1618-1618/com.example.me.iotbook3 I/BMX280Callback: On Sensor
connected...
    Pressure sensor...
09-28 12:25:17.699 1618-1618/com.example.me.iotbook3 I/BMX280Callback: On Sensor connected...
```

```
    Humidity sensor...
09-28 12:25:17.727 1618-1618/com.example.me.iotbook3 I/TemperatureCallback: Accuracy: 3
09-28 12:25:17.728 1618-1618/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
27.987799
09-28 12:25:17.730 1618-1618/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
28.002787
09-28 12:25:17.730 1618-1618/com.example.me.iotbook3 I/PressureCallback: Accuracy: 3
09-28 12:25:17.732 1618-1618/com.example.me.iotbook3 I/setRGBPins: Change RGB led color.
Red [false] - Green [false] - Blue [true]
09-28 12:25:17.745 1618-1618/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.87885
09-28 12:25:17.780 1618-1618/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
28.042746
09-28 12:25:17.782 1618-1618/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.886
09-28 12:25:17.782 1618-1618/com.example.me.iotbook3 I/HumidityCallback: Accuracy: 3
09-28 12:25:17.783 1618-1618/com.example.me.iotbook3 I/humidityCallback: Humidity: 68.94584
09-28 12:27:42.581 1618-1618/com.example.me.iotbook3 I/humidityCallback: Humidity: 69.07395
09-28 12:27:42.599 1618-1618/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.907
09-28 12:27:42.615 1618-1618/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.995844
09-28 12:27:42.631 1618-1618/com.example.me.iotbook3 I/humidityCallback: Humidity: 69.07331
09-28 12:27:42.634 1618-1618/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.935
09-28 12:27:42.650 1618-1618/com.example.me.iotbook3 D/TemperatureCallback: Change RED led
color. New state [true]
09-28 12:27:42.651 1618-1618/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
30.00084
```

И интерфейс вашего приложения будет выглядеть, как показано на рис. 120.

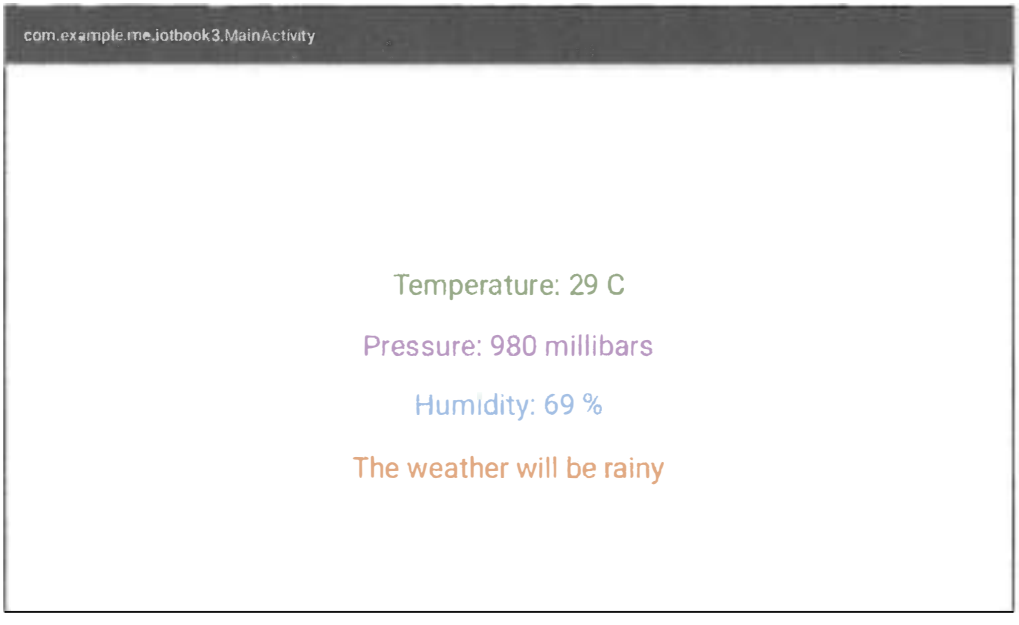


Рис. 120 ❖ Интерфейс Android Things-приложения мониторинга окружающей среды с влажностью

Четвёртый проект в Android Things – объединение Android Things с облачной платформой интернета вещей

0. В этом проекте мы научимся интегрировать ОС Android Things с облачными платформами интернета вещей (IoT cloud platforms). Это – важный аспект при разработке приложения для интернета вещей. Существует несколько сценариев, когда требуется передавать в облако данные, полученные от Android Things-плат. Мы изучим архитектуру облака интернета вещей, научимся конфигурировать облачную платформу интернета вещей, подключать Android Things-приложение к этой платформе и посылать (стримить) данные в реальном времени в облако, создавая инструментальные панели с графиками для анализа данных (dashboards).

К этому моменту мы научились разрабатывать Android Things-приложения, которые самодостаточны: они не взаимодействуют с внешними системами или платформами (кроме того, приложения с Google Firebase). Все данные, полученные через сенсоры, обрабатываются локально. Но обычно приложения для интернета вещей посылают данные в облако, поэтому облачные платформы интернета вещей играют важную роль. В настоящее время облачные платформы интернета вещей являются важной частью в экосистеме интернета вещей. Используя эти платформы, мы можем расширить сервисы, которые предлагаем пользователю, и раскрыть всю мощь плат с Android Things. Как только данные доступны на уровне облака интернета вещей, эти платформы могут обеспечить комплексный анализ, где требуется большая вычислительная мощность, применяя такие технологии, как машинное обучение, искусственный интеллект и анализ больших данных – задачи, которые невозможно выполнить на Android Things-платах. Но без последних не было бы данных для анализа.

Обычная облачная платформа интернета вещей предоставляет следующий набор сервисов: **сервис соединения/подключения, сервис хранения данных, сервис обработки событий, управление устройствами, визуализация данных, интеграция сервисов**.

Ядром сервиса подключения являются соединение и передача данных между облачной платформой интернета вещей и удалённой платой. Платформы поддерживают различные протоколы для упрощения процесса подключения: Rest API и HTTP, MQTT, CoAP и т. д. Другими словами, платформы предоставляют набор программных интерфейсов, которые могут быть использованы удалёнными платами интернета вещей для подключения и обмена данными. Кроме этого, они предоставляют набор SDK для различных плат, чтобы сделать процесс подключения быстрым и более простым.

Сервис хранения данных служит для хранения данных в облаке. Эти данные являются основой для других сервисов.

Эти первые два сервиса предоставляются почти всеми облачными платформами интернета вещей, в то время как сервис обработки событий является более сложным. Такой сервис основан на правилах и использует сохранённые

данные и события для запуска действий, которые могли бы иметь влияние на платы интернета вещей. Обычно все события и действия конфигурируются через веб-интрерфейс.

Сервис управления устройствами берёт на себя управление всеми устройствами интернета вещей, подключёнными к платформе. Иными словами, это централизованная административная консоль для удалённых устройств.

Визуализация данных – это сервис, предоставляемый некоторыми облачными платформами интернета вещей для создания инструментальных панелей (dashboards) с целью графической визуализации полученных данных с помощью гистограмм/графиков.

Сервис интеграции полезен, когда мы хотим интегрировать какие-то внешние сервисы типа сообщений по электронной почте, сообщений в Твиттере и т. д. и запускать их в соответствии с заранее сконфигурированными событиями.

На рынке существует несколько облачных платформ интернета вещей, например (совсем небольшой пример): Google IoT cloud, Microsoft Azure IoT, Amazon AWS IoT, Samsung Artik Cloud, Temboo, Ubidots.

Архитектура интернета вещей обычно следующая: уровень сенсоров -> платы интернета вещей -> облачные платформы интернета вещей -> высокоуровневые сервисы с пользовательским интерфейсом для конечного пользователя. Иногда 3-й и 4-й слои смешаны друг с другом.

Чтобы использовать облачную платформу интернета вещей, нам нужно сделать два шага: сконфигурировать проект интернета вещей в облачной платформе, предоставив всю информацию, включая тип данных, и создать клиента облачной платформы интернета вещей (Android Things-приложение), который обрабатывает подключение и посылает данные.

В качестве облачной платформы интернета вещей мы будем использовать Samsung Artik Cloud (https://artik.cloud). Это профессиональная платформа, предоставляющая почти все упомянутые выше сервисы. Она проста в использовании и предоставляет несколько SDK, которые упрощают процесс обмена данными. Мы вручную реализуем обмен данными между Android Things-платой и облаком Samsung Artik, используя его Rest API-механизм.

1. Создайте бесплатный аккаунт в Samsung Artik Cloud: перейдите по ссылке https://my.artik.cloud/ и затем зарегистрируйтесь (sign up) и залогиньтесь (sign in). После этого перейдите по адресу https://developer.artik.cloud/ и нажмите на кнопку **Create your first device type**. Заполните поля **device display name** и **unique name**, например как показано на рис. 121.

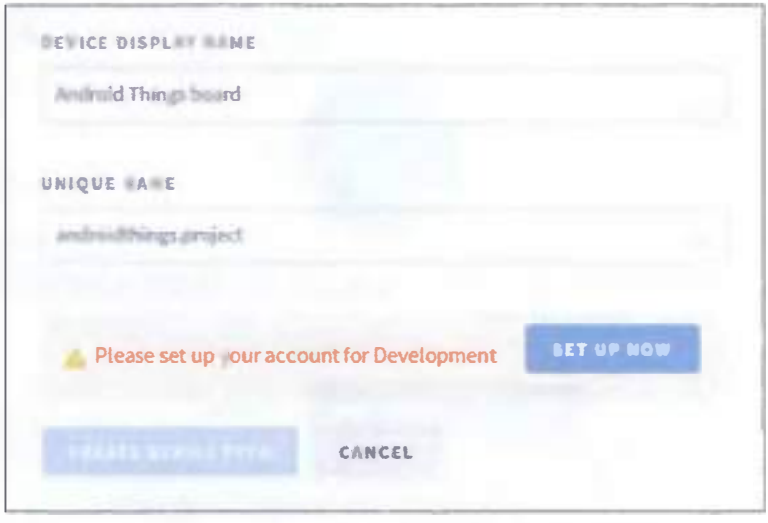


Рис. 121 ❖ Создание типа устройства (device type)

Нажмите на кнопку **Set up now**, чтобы настроить ваш аккаунт для разработки. В следующем окне выберите, какой проект у вас будет – групповой (**Team**) или индивидуальный (**Individual**) **project**, затем введите название своей должности и, наконец, введите информацию о вашей организации и метоположении, нажмите **Complete**. После этого вы увидите следующее сообщение в окне, показанном на рис. 121, и сможете нажать на кнопку **Create device type** – она станет активной.

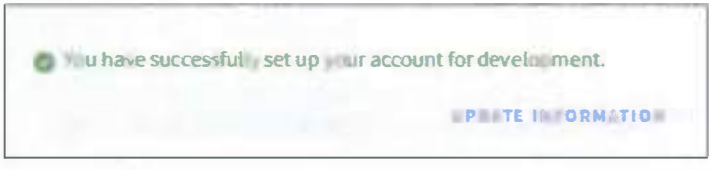


Рис. 122 ❖ Сообщение об успешном завершении конфигурации аккаунта в Artik Cloud

Возможно, вам придётся поменять поле **unique name**, если уникальное имя уже занято. После успешного нажатия на кнопку **Create device type** вы увидите следующее окно.

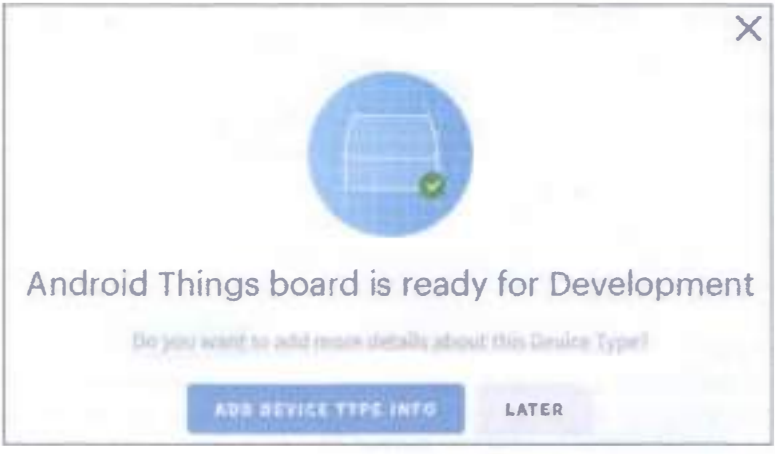


Рис. 123 ❖ Окно с сообщением о готовности системы к поддержке приложений для Android Things-платы

В этом окне нажмите кнопку **Later** и затем нажмите кнопку **New manifest**. Когда мы настраиваем манифест, мы уведомляем Artik Cloud о данных, которые наше приложение будет посылать, чтобы облачная платформа могла извлечь данные, посылаемые Android Things-приложением. В нашем предыдущем проекте мы измеряли три параметра: температуру, давление и влажность. Поэтому мы должны сконфигурировать три переменные в манифесте. Сначала мы настроим поле температуры.

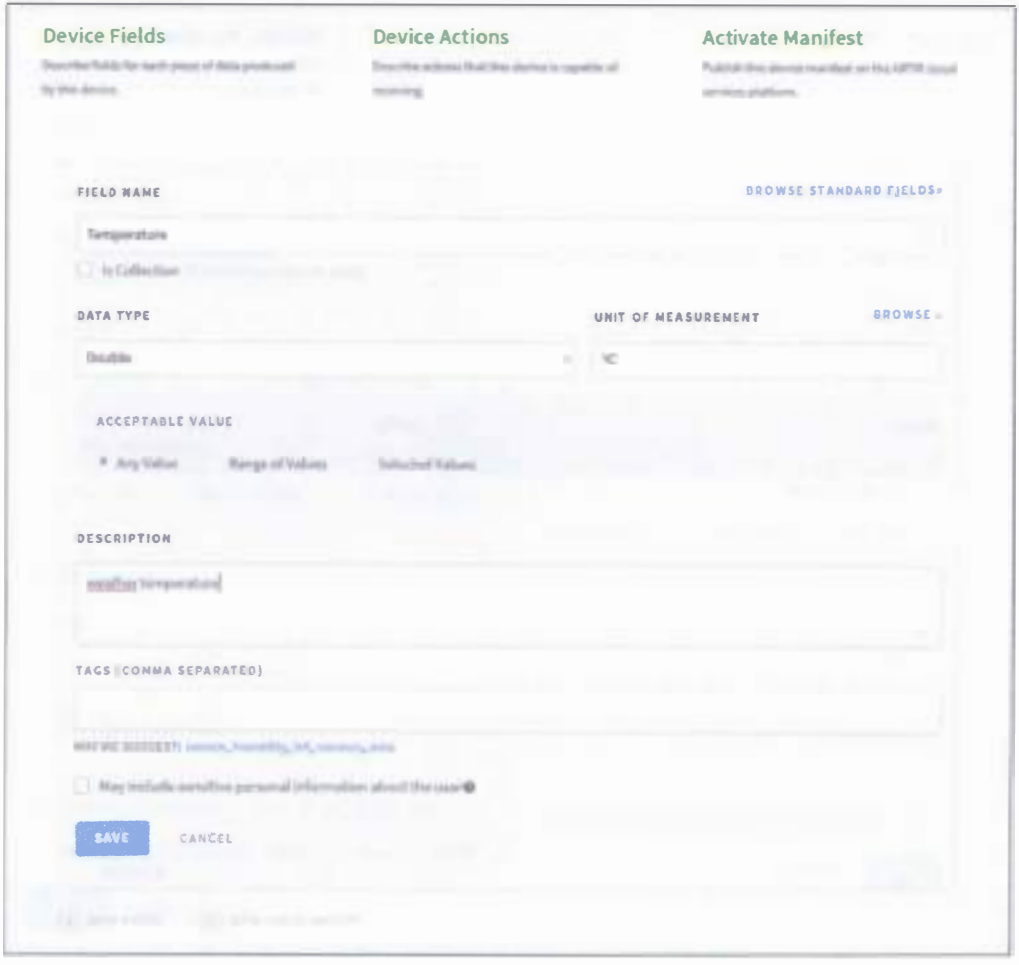


Рис. 124 ❖ Настройка поля/переменной температуры

Самое главное поле в этом окне – это поле **name**, которое мы используем для ссылки на переменную. Нажмите **Save** и затем нажмите **New Field** для переменной давления.

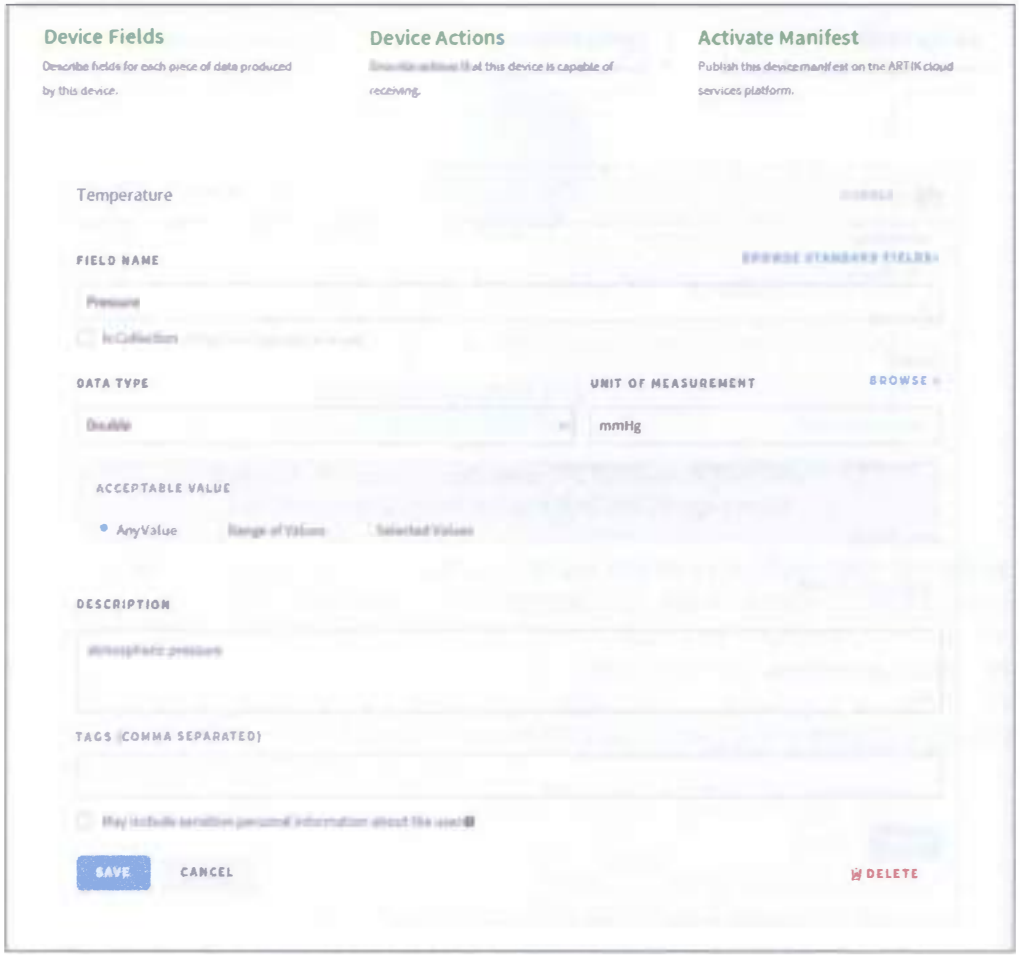


Рис. 125 ❖ Настройка поля/переменной давления

Далее нажмите кнопку **Save** и проделайте то же самое с влажностью.

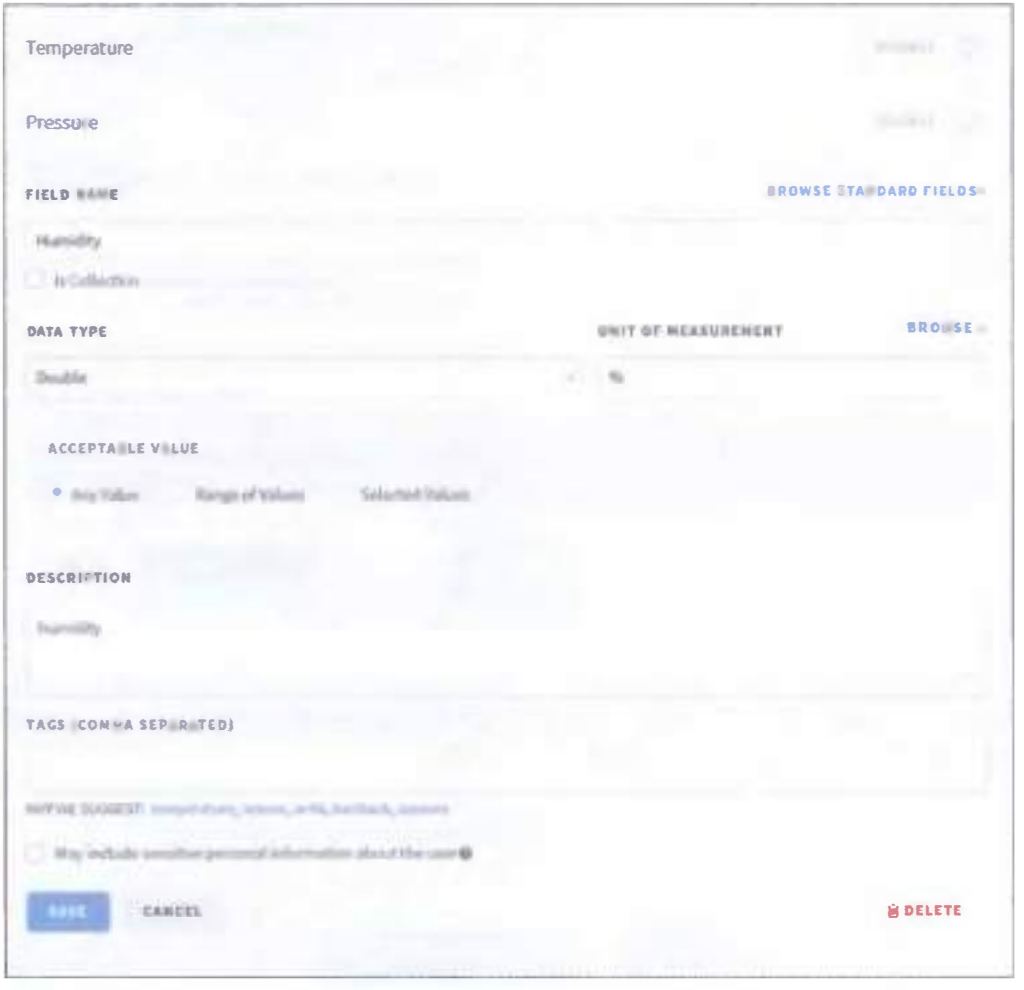


Рис. 126 ❖ Настройка поля/переменной влажности

Нажмите **Save**. После этого нажмите на закладку **Activate Manifest** и затем нажмите на кнопку **Activate Manifest**.

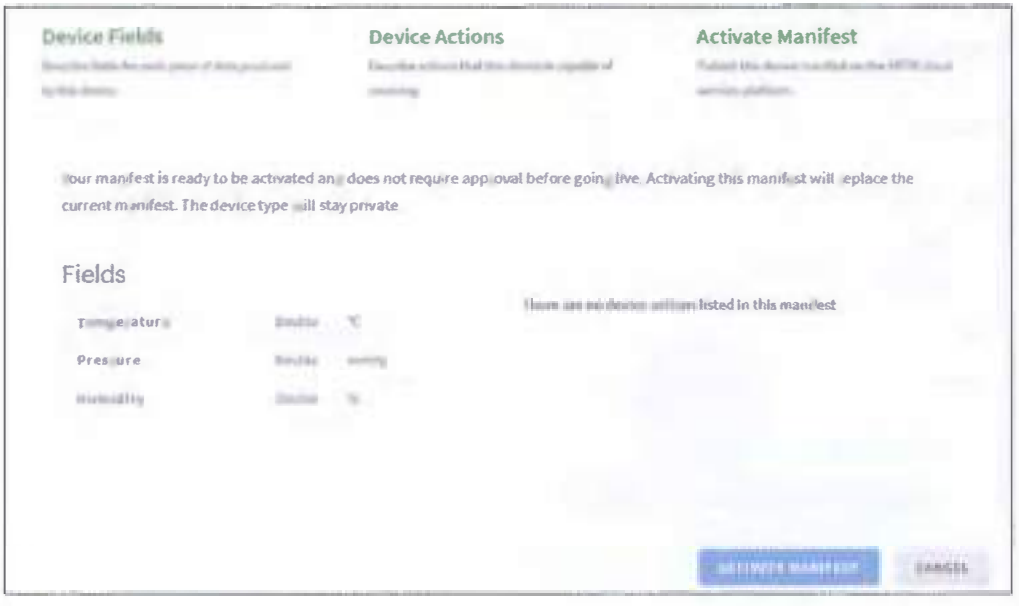


Рис. 127 ❖ Закладка Activate Manifest

Теперь нажмите **на My Cloud** вверху справа, чтобы перейти в панель инструментов (dashboard). На странице **Devices** вы увидите тип устройства вашей организации, который мы только что создали.

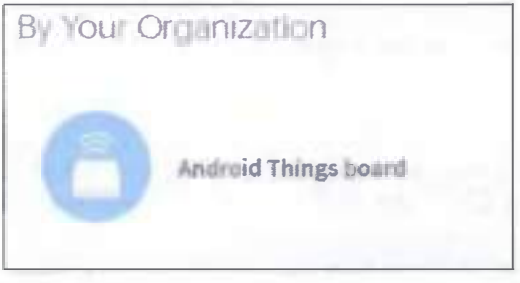


Рис. 128 ❖ Тип устройства в панели инструментов

Устройство – это экземпляр типа устройства, который мы создали. Разместите указатель мыши поверх этого типа и нажмите **Select**. Назовите ваше новое устройство, например:

Рис. 129 ❖ Окно добавления нового устройства

И затем нажмите на кнопку **Add Device**, в результате откроется окно, изображённое на рис. 130.

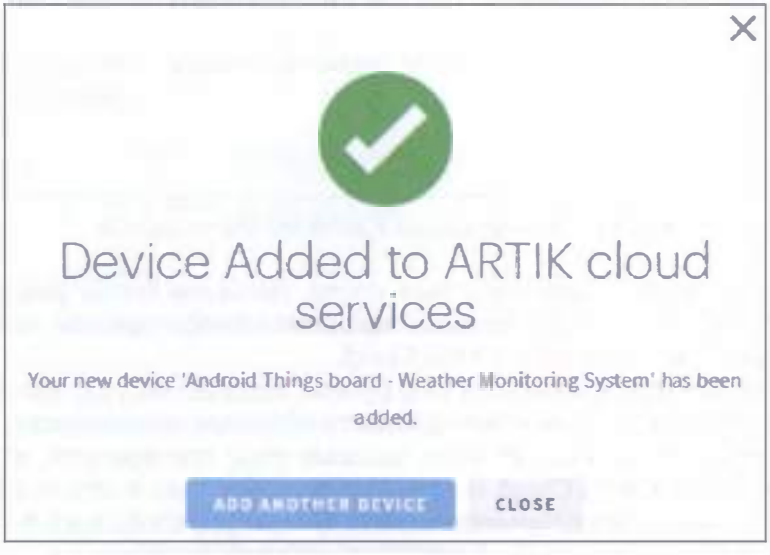


Рис. 130 ❖ Окно с сообщением об успешном добавлении нового устройства

Нажмите кнопку **Close**. Теперь если вы нажмёте на созданное устройство, то увидите следующее.

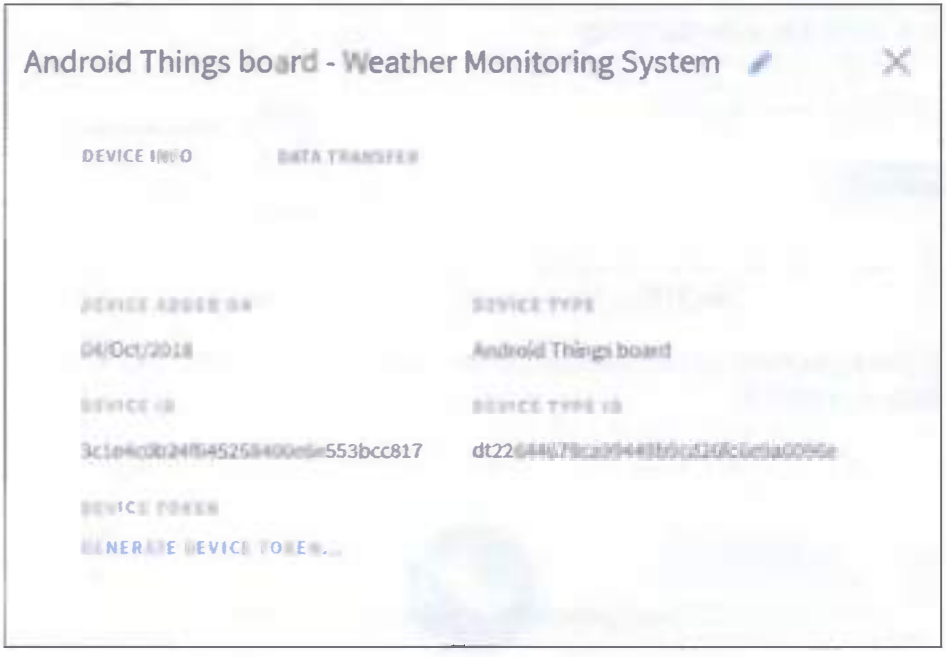


Рис. 131 ❖ Информация о добавленном устройстве

Эта информация понадобится нам позже, когда мы будем разрабатывать Android Things-клиента. На этом мы закончили конфигурацию нашего проекта интернета вещей в облаке Artik Cloud.

2. Теперь нам нужно изменить наш проект, который мы разработали в прошлом задании, так как нам нужно добавить облачные возможности. Но перед этим всё ещё есть некоторые шаги, которые надо предпринять, чтобы подключить клиента к Artik Cloud. Нам нужно подключиться к облаку с помощью следующей ссылки: **https://api.artik.cloud/v1.1/messages**, чтобы посылать данные об аутентификации нашего Android Things-клиента и сообщение, содержащее данные с сенсора и другую информацию. Затем для аутентификации клиента заголовок http-запроса должен содержать следующую строку: **Authorization: Bearer device_token**, поэтому нам нужен токен устройства (device token). Мы можем получить его, если нажмём на ссылку **Generate device token** во всплывающем окне, показанном на рис. 131. Сообщение, которое клиент посылает в облако, также должно обладать определённой структурой. Чтобы знать его структуру, перейдите в консоль разработчика (**https://developer.artik.cloud/**), нажмите на **Device Types** и затем – на **Android Things board**. Нажмите на кнопку **Edit Info**, затем – на пункт меню **Manifest** (в меню слева), и вы должны увидеть что-то вроде следующего:

Рис. 132 ❖ Информация из манифеста

Нажмите на **View Sample Message**, чтобы знать, как выглядит структура данных сообщения.

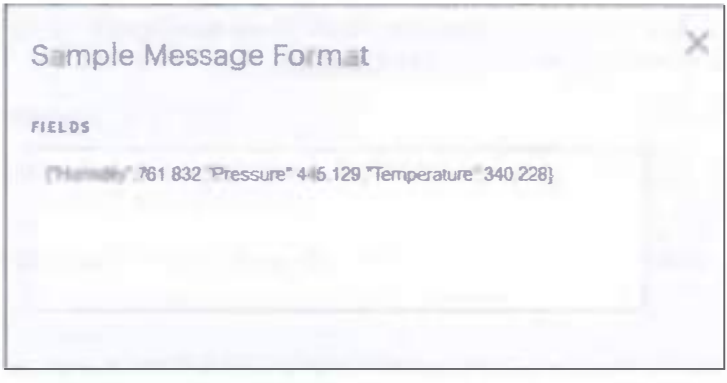


Рис. 133 ❖ Окно со структурой сообщения

Эти данные сенсора должны содержаться внутри **сообщения**, которое является обёрткой, добавляющей другую информацию к реальным данным и выглядящей так:

```
{
    "sdid":"device_id",
    "ts":timestamp,
    "data":
    {
        "Pressure":765.111,
        "Temperature":30.659,
        "Humidity":65.345
    }
}
```

Здесь device_id – это уникальный идентификатор устройства, а ts – метка времени. Теперь мы знаем, как построить сообщение и как извлечь параметры аутентификации.

2.1. Сейчас мы разработаем Android Things Artik-клиента. Мы используем Android Things-приложение из предыдущего практического задания. Мы можем использовать как библиотеку Android HTTP, так и пользовательскую (кастомную) библиотеку для обработки http-соединений. В этом проекте мы будем использовать библиотеку Volley (https://developer.android.com/training/volley/). Эта библиотека широко используется и предлагает интересные возможности. Также она упрощает управление http-соединением.

Откройте файл build.gradle(Module: things) и добавьте следующую строку в раздел dependencies:

```
implementation 'com.android.volley:volley:1.1.0'
```

Добавьте следующую строку в файл AndroidManifest.xml file, чтобы получить доступ к интернету через приложение:

```
uses-permission android:name="android.permission.INTERNET" />
```

Теперь создадим новый класс ArtikClient.java в папке things -> java -> packagename (в моём случае название пакета – com.example.me.IoTbook4) для обработки всех деталей обмена сообщениями.

```
public class ArtikClient {

private static String TAG = "Artik";
private static ArtikClient me;
private Context ctx;
private RequestQueue queue;
private static final String ARTIK_URL = "https://api.artik.cloud/v1.1/messages";
private String deviceId;
private String token;

    private ArtikClient(Context ctx, String deviceId, String token) {
        this.ctx = ctx;
        this.deviceId = deviceId;
        this.token = token;
        createQueue();
    }

}
```

ArtikClient – это синглтон (singleton), и он принимает Android Context, ID устройства и параметры токена, упомянутые ранее. Теперь нам нужно создать createQueue()-метод, который инициализирует очередь запросов Volley, чтобы приложение могло посылать запросы к Artik Rest API.

```
private void createQueue() {
    if (queue == null)
        queue = Volley.newRequestQueue(ctx.getApplicationContext());
}
```

Добавьте метод getInstance – нам он понадобится позже.

```
public static final ArtikClient getInstance(Context ctx, String deviceId, String token) {
    if (me == null)
        me = new ArtikClient(ctx,deviceId,token);
    return me;
}
```

Следующий метод, который надо добавить, – это sendData():

```
public void sendData(final double temp, final double press, final double hum) {

}
```

Он вызывает Artik Rest API, используя структуру сообщения, описанную ранее. Первый шаг добавления этого метода – создание экземпляра StringRequest, который представляет наш http-запрос. Добавьте следующее в метод sendData():

```
StringRequest request = new StringRequest(Request.Method.POST,
        ARTIK_URL,
        new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                Log.d(TAG, "Response ["+response+"]");
            }
        },
        new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                error.printStackTrace();
            }
        }) {
```

Второй шаг – переопределить метод getHeaders(), чтобы настроить (кастомизировать) заголовки http-запроса, так как приложение должно посылать параметр заголовка Authorization, как предписано Artik-спецификациями:

```
@Override
public Map<String, String> getHeaders() throws AuthFailureError {
    Log.d(TAG, "Get headers..");
    Map<String, String> headers = new HashMap<String, String>();
    headers.put("Content-Type", "application/json");
    headers.put("Authorization", "Bearer " + token);
    return headers;
}
```

И третий шаг – переопределить метод getBody(), чтобы настроить тело сообщения, которое мы посылаем:

```
@Override
public byte[] getBody() throws AuthFailureError {
    Log.d(TAG, "Creating body...");

    try {
```

```
            JSONObject jsonRequest = new JSONObject();
            jsonRequest.put("sdid", deviceId);
            jsonRequest.put("ts", System.currentTimeMillis());
            JSONObject data = new JSONObject();
            data.put("Temperature", temp);
            data.put("Pressure", press);
            data.put("Humidity", hum);
            jsonRequest.put("data", data);

            String sData = jsonRequest.toString();
            Log.d(TAG, "Body:" + sData);

            return sData.getBytes();
        } catch (JSONException jsoe) {
            jsoe.printStackTrace();

        }
        return "".getBytes();
    }

};
```

Этот метод использует библиотеку JSON для построения JSON-сообщения. Он добавляет к сообщению все параметры, рассмотренные ранее.

Последний шаг для подготовки Artik-клиента – это добавление запроса в очередь, чтобы Volley смогла обработать его:

```
queue.add(request);
```

Эта строка завершает метод sendData().

2.2. Наконец, нам надо посылать данные из Android Things-приложения. Чтобы делать это, нам надо вызывать клиента из класса MainActivity.java (things -> java -> package name). Самый простой способ посылать данные в Artik Cloud – это использовать его API каждый раз, когда сенсор считывает новое значение. Это потребовало бы почти бесконечного обращения к Artik API. Поэтому, чтобы не перегружать Artik Cloud, лучший подход – это использование планировщика для отправки данных. С помощью планировщика Android Things-приложение посылает данные с определёнными временными интервалами. Мы можем настроить частоту отправки данных, тем самым больше влияя на поведение приложения и полосу пропускания данных, которую потребляет приложение.

Добавьте следующие переменные в секцию объявления переменных класса MainActivity.java (**поменяйте значения переменных device_id и token на значения вашего устройства**):

```
private static final int TIMEOUT = 1;
private static final double FACTOR = 0.750061561303;
private int tempCounter = 0;
private int pressCounter = 0;
private int humCounter = 0;
private double totalTemp;
private double totalPress;
```

```
private double totalHum;
private static String DEVICE_ID = "3c1e4c0b24f645258400e6e553bcc817";
private static String TOKEN = "04b92cb5070f48aba72f1e030325c6f9";
```

Мы должны немного поменять наши callback-классы. В классе TemperatureCallback в методе OnSensorChanged добавьте следующие строки в конец:

```
totalTemp += val;
tempCounter++;
```

В классе PressureCallback в методе OnSensorChanged добавьте следующие строки:

```
totalPress += val;
pressCounter++;
```

И в классе HumidityCallback в методе OnSensorChanged добавьте следующее:

```
totalHum += val;
humCounter++;
```

И затем добавьте следующий метод в класс MainActivity.java:

```
private void initScheduler() {
    ScheduledExecutorService scheduler =
            Executors.newSingleThreadScheduledExecutor();

    scheduler.scheduleAtFixedRate(new Runnable() {
        @Override
        public void run() {
            //Log.d(TAG, «Scheduler running...»);
            double mTemp = totalTemp / tempCounter;
            double mPress = totalPress / pressCounter * FACTOR;
            double mHum = totalHum / humCounter;

            totalTemp = 0;
            totalPress = 0;
            totalHum = 0;
            tempCounter = 0;
            pressCounter = 0;
            humCounter = 0;
            // call artik
            ArtikClient.getInstance(MainActivity.this,
                    DEVICE_ID, TOKEN).sendData(mTemp, mPress, mHum);

        }
    }, 1, TIMEOUT, TimeUnit.MINUTES);
}
```

В этом классе мы используем SchedulerExecutorService, который запускает конкретную задачу всё время с задержкой, указанной в переменной TIMEOUT. Задача определяется в методе run(). В этом методе приложение выполняет следующие шаги: вычислить среднее значение температуры за интервал времени между временем последней отправки данных в Artik и текущим временем; таким же образом вычислить средние значения давления и влажности; преобра-

зовать значение давления из миллибаров в мм рт. ст., которые требуются в консоли Artik Cloud; вызвать ArtikClient для отправки средних значений; сбросить общее значение переменных и общий счётчик полученных значений температуры, давления и влажности. И последний шаг – вызвать этот метод, чтобы запланировать задачу: в методе onCreate() добавьте следующую строку в конце:

```
initScheduler();
```

Можно видеть, что переменная TIMEOUT установлена в 1 минуту (потому что присутствует константа TimeUnit.***MINUTES***, и TIMEOUT равна 1). Вы можете изменить эти значения, если хотите. Теперь вы можете подключить к питанию плату и монитор и запустить приложение. В логе Logcat вы увидите примерно следующее:

```
2018-10-04 16:23:18.680 1588-1588/com.example.me.iotbook4 I/BMX280Callback: On Sensor
connected...
2018-10-04 16:23:18.680 1588-1588/com.example.me.iotbook4 I/BMX280Callback: Temp sensor...
2018-10-04 16:23:18.695 1588-1588/com.example.me.iotbook4 I/BMX280Callback: On Sensor
connected...
2018-10-04 16:23:18.695 1588-1588/com.example.me.iotbook4 I/BMX280Callback: Pressure
sensor...
2018-10-04 16:23:18.710 1588-1588/com.example.me.iotbook4 I/BMX280Callback: On Sensor
connected...
2018-10-04 16:23:18.710 1588-1588/com.example.me.iotbook4 I/BMX280Callback: Humidity
sensor...
2018-10-04 16:23:18.725 1588-1588/com.example.me.iotbook4 I/TemperatureCallback: Accuracy:
3
2018-10-04 16:23:18.727 1588-1588/com.example.me.iotbook4 I/TemperatureCallback:
Temperature: 27.84794
2018-10-04 16:23:18.727 1588-1588/com.example.me.iotbook4 I/PressureCallback: Accuracy: 3
2018-10-04 16:23:18.730 1588-1588/com.example.me.iotbook4 I/setRGBPins: Change RGB led
color. Red [false] - Green [false] - Blue [true]
2018-10-04 16:23:18.733 1588-1588/com.example.me.iotbook4 I/PressureCallback: Pressure:
727.2553
2018-10-04 16:23:18.757 1588-1588/com.example.me.iotbook4 I/TemperatureCallback:
Temperature: 27.852932
2018-10-04 16:23:18.759 1588-1588/com.example.me.iotbook4 I/PressureCallback: Pressure:
979.8375
2018-10-04 16:23:18.759 1588-1588/com.example.me.iotbook4 I/HumidityCallback: Accuracy: 3
2018-10-04 16:23:18.760 1588-1588/com.example.me.iotbook4 I/humidityCallback: Humidity:
68.93547
2018-10-04 16:23:19.619 1588-1626/com.example.me.iotbook4 D/Artik: Get headers..
2018-10-04 16:23:19.688 1588-1626/com.example.me.iotbook4 D/Artik: Creating body...
2018-10-04 16:23:19.692 1588-1626/com.example.me.iotbook4 D/Artik: Body:{"sdid":"3c1e4c0b2
4f645258400e6e553bcc817","ts":1538659399688,"data":{"Temperature":27.98967432975769,"Press
ure":725.0232714798924,"Humidity":50.68603856940018}}
2018-10-04 16:23:19.708 1588-1588/com.example.me.iotbook4 I/humidityCallback: Humidity:
49.181366
2018-10-04 16:23:19.709 1588-1588/com.example.me.iotbook4 I/TemperatureCallback:
Temperature: 28.09769
2018-10-04 16:23:19.712 1588-1588/com.example.me.iotbook4 I/PressureCallback: Pressure:
979.8723
```

```
2018-10-04 16:23:21.313 1588-1588/com.example.me.iotbook4 D/Artik: Response [{"data":{"mid
":"c5b4bf2ba7464f8d97edc2114aafc961"}}]
2018-10-04 16:23:29.545 1588-1627/com.example.me.iotbook4 D/Artik: Get headers..
2018-10-04 16:23:29.549 1588-1627/com.example.me.iotbook4 D/Artik: Creating body...
2018-10-04 16:23:29.552 1588-1588/com.example.me.iotbook4 I/humidityCallback: Humidity:
50.674202
2018-10-04 16:23:29.553 1588-1627/com.example.me.iotbook4 D/Artik: Body:{"sdid":"3c1e4c0b2
4f645258400e6e553bcc817","ts":1538659409550,"data":{"Temperature":28.236814498901367,"Pres
sure":734.9934985935055,"Humidity":51.02160032171952}}
2018-10-04 16:23:29.577 1588-1588/com.example.me.iotbook4 I/PressureCallback: Pressure:
979.9393
2018-10-04 16:23:29.593 1588-1588/com.example.me.iotbook4 I/humidityCallback: Humidity:
50.696575
2018-10-04 16:23:29.605 1588-1588/com.example.me.iotbook4 I/TemperatureCallback:
Temperature: 28.322474
2018-10-04 16:23:30.336 1588-1588/com.example.me.iotbook4 D/Artik: Response [{"data":{"mid
":"78b00da261fb43ccba50ac3f0b7cbcdf"}}]
```

Вы увидите, что от платформы Artik Cloud приходит ответ (response), уведомляющий приложение, что посланные данные приняты облаком Artik. Таким образом, можно переслать достаточно много показаний, которые позже можно анализировать в облаке.

Вы можете проверить значения, посланные в Artik Cloud, и можете создать панель инструментов (dashboard). Войдите в аккаунт Artik Cloud, перейдите по адресу https://my.artik.cloud/ и выберите **Charts** в меню сверху. Вы увидите следующее всплывающее окно.

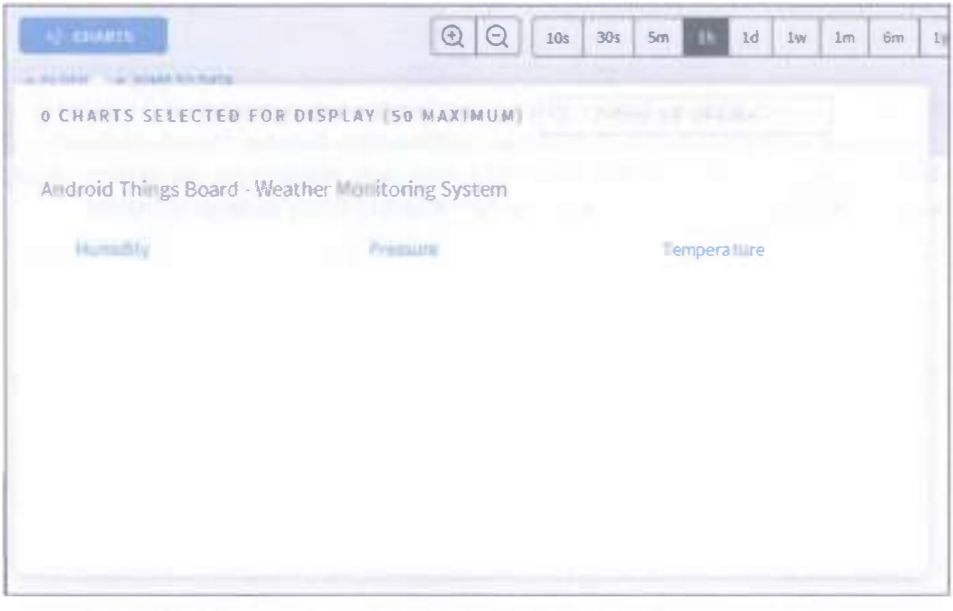


Рис. 134 ❖ Окно выбора переменных для отображения на графиках

Здесь вы можете добавить переменные, которые вас интересуют, которые мы создали в начале этого практического задания. Затем вы увидите данные, и вы можете поменять период времени, чтобы точно установить период, когда данные отсылались в облако, – это сделает график более понятным и наглядным, например таким:

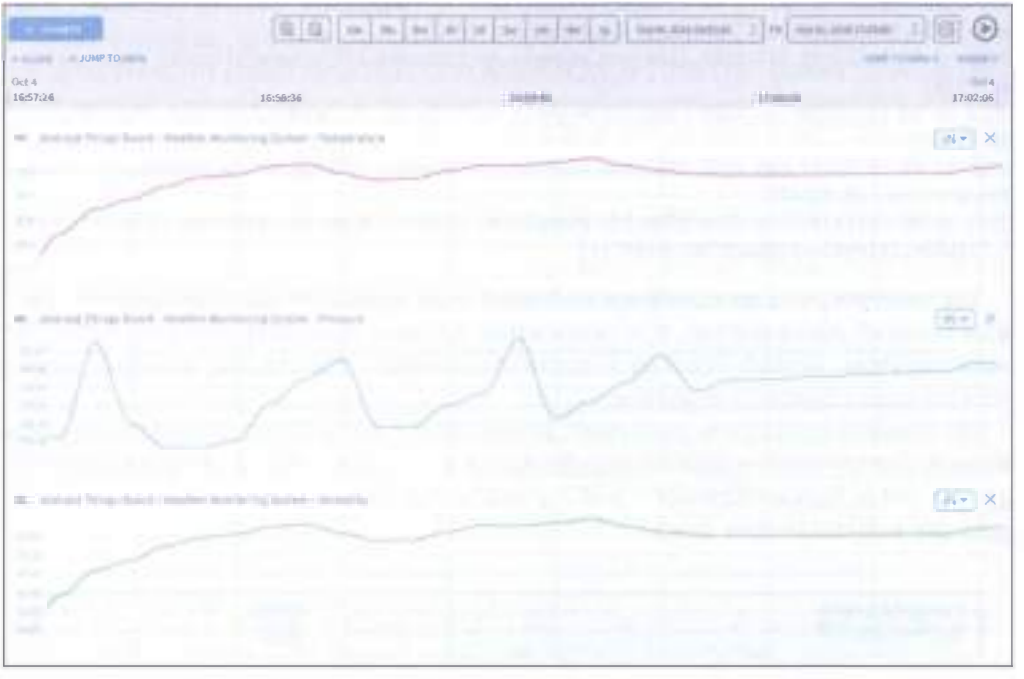


Рис. 135 ❖ Окно с графиками для всех трёх переменных

Около кнопки закрытия **X** есть другие два вида графиков, вы можете попробовать их, чтобы добиться лучшей визуализации полученных данных.

Что касается точных данных, отосланных в облако, – вы можете посмотреть их, используя пункт **Data Logs** верхнего меню. Логи данных (Data Logs) показывают вам все запросы, которые сделал Android Things-клиент в облако Artik Cloud.

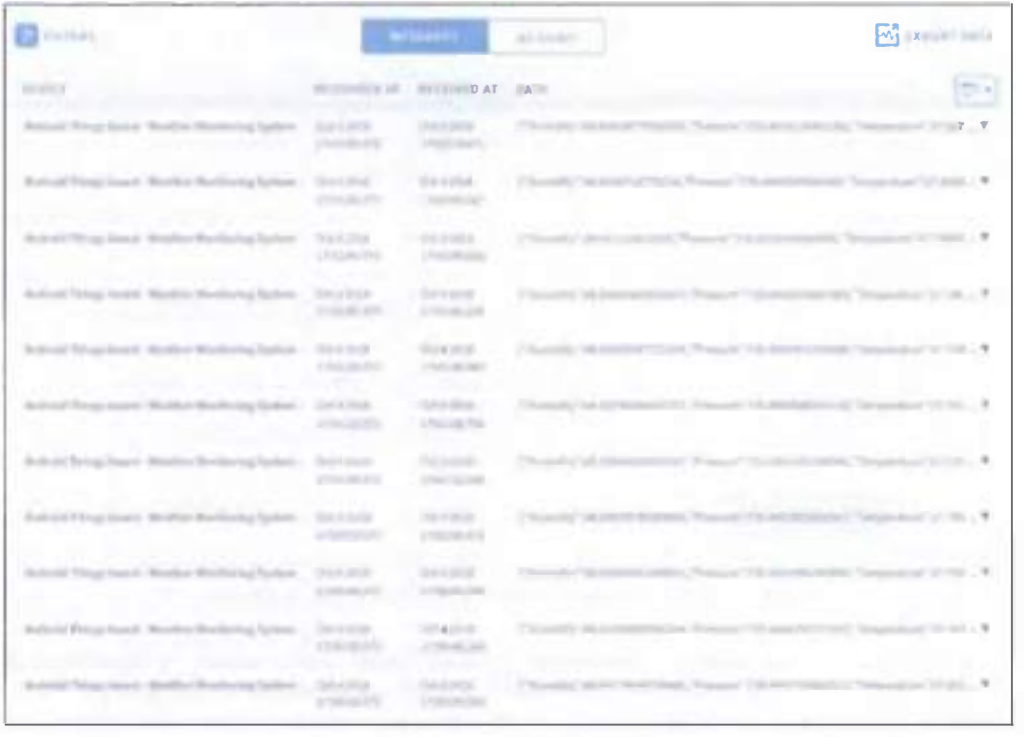


Рис. 136 ❖ Логи данных, переключённые на просмотр сообщений от клиента

Также вы можете экспортировать данные:

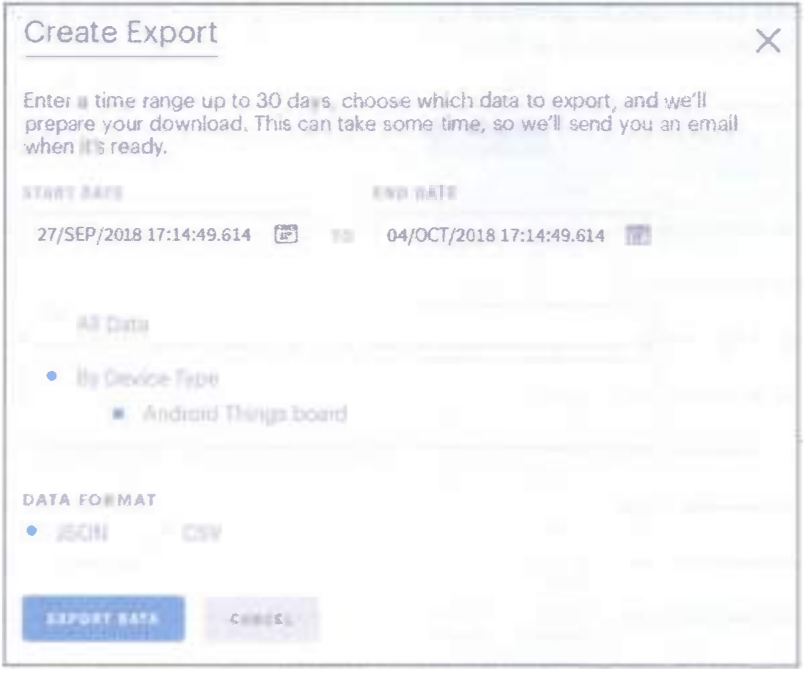


Рис. 137 ❖ Возможности Artik Cloud по экспорту данных

3. Эта часть задания посвящена добавлению голосовых способностей в Android Things. Платформы интернета вещей предлагают различные типы сервисов, не только для хранения данных. Есть некоторые платформы интернета вещей, которые предоставляют сервисы интеграции. Другими словами, они не сфокусированы на получении данных от сенсоров и их хранении, их цель – предоставление сервисов интеграции с другими облачными системами. Одна из таких платформ – **Temboo** (https://temboo.com/). Она предлагает большой набор сервисов интеграции, которые могут быть использованы, чтобы расширить возможности приложения интернета вещей. Temboo поддерживает несколько языков программирования и платформ интернета вещей, включая ОС Android.

Что мы хотим сделать – так это добавить голосовые возможности в наше Android Things-приложение, чтобы оно задействовало голосовой звонок с заранее настроенным сообщением, дабы проинформировать нас о том, что произошло какое-то конкретное событие. Android Things-приложение использует Temboo-сервис, который называется **choreo** и упрощает интеграцию с платформой Nexmo. **Nexmo** (https://www.nexmo.com/) – это голосовая облачная платформа. Наше Android Things-приложение с помощью Android Things-платы будет вызывать Temboo, когда температура будет больше либо равна 30 °C. В ответ Temboo будет вызывать Nexmo, которая будет совершать звонок на наш смарт-

фон, используя движок Text To Speech (TTS), чтобы преобразовать текст сообщения в человеческий голос, который мы слышим во время телефонного звонка.

3.1. Первый шаг – это настроить Temboo choreo, чтобы она могла общаться с платформой Nexmo. Сначала мы должны создать бесплатный аккаунт Nexmo для нашего приложения. При регистрации вы должны предоставить номер своего телефона и бизнес-адрес электронной почты (т. е. адрес почты, зарегистрированный в какой-то организации (HSE подойдёт), а не в бесплатных почтовых системах, таких как Mail.ru, Gmail, Yandex и т. д.). Затем вы должны верифицировать свой номер телефона и адрес электронной почты. Внимание: письмо с верификацией приходит в течение 5–10 минут. После верификации вы можете войти на сайт платформы и ответить на дополнительные вопросы, например такие:

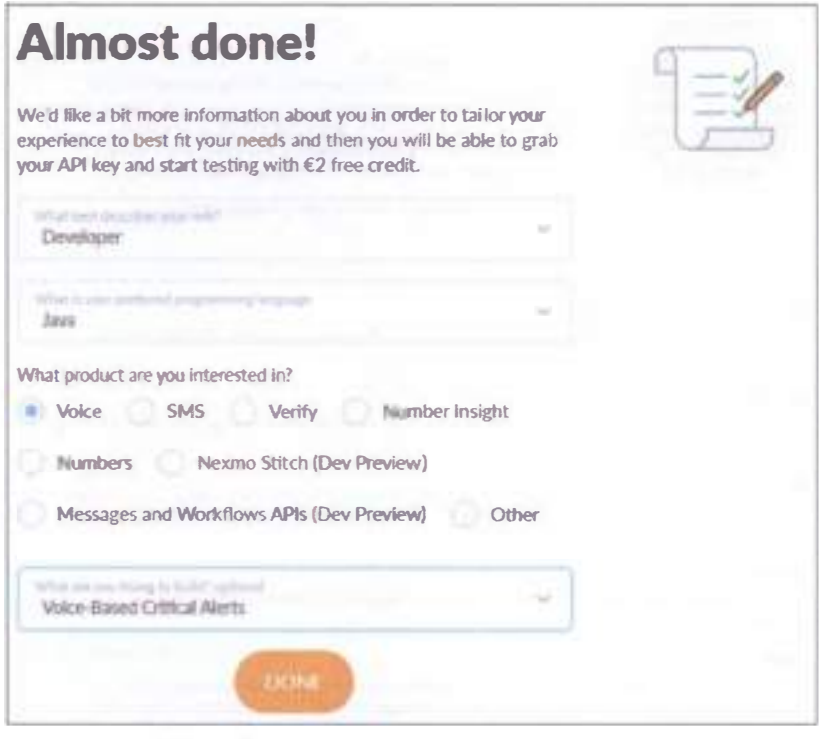


Рис. 138 ❖ Дополнительные вопросы платформы Nexmo

Нажмите кнопку **Done** и перейдите в панель инструментов консоли. Нам нужна эта консоль, чтобы получить два важных параметра: ключ (key) и секрет (secret). Нажмите на логотип Nexmo в верхнем левом углу консоли, и вы увидите эти параметры (вместе с приветственным сообщением и вашим номером телефона в разделе **Try the API**). Эта пара параметров используется платформой Temboo для аутентификации запроса к серверу.

Теперь нам нужно создать аккаунт Temboo (**https://temboo.com/**). Бесплатная пробная версия действует только 14 дней! Перейдите по ссылке **https://temboo.com/signup** и заполните все текстовые поля. Нажмите кнопку **Try it now**, затем прокликайте через 3 примера, потом нажмите **Start building**. Закройте приветственное всплывающее окно и найдите Nexmo в левой панели. Выберите её и заметьте, что в окне справа есть информация о функции TTS.

Рис. 139 ❖ Информация о поддерживаемых возможностях интеграции с Nexmo

Ниже перечня возможностей располагается список поддерживаемых функций, среди которого нет **Nexmo.Voice choreo** в последней на данный момент версии (2.22.0), но мы можем выбрать, например, SMS-функционал (под Bundles) и затем функцию SendMessage, и вы увидите следующее окно:

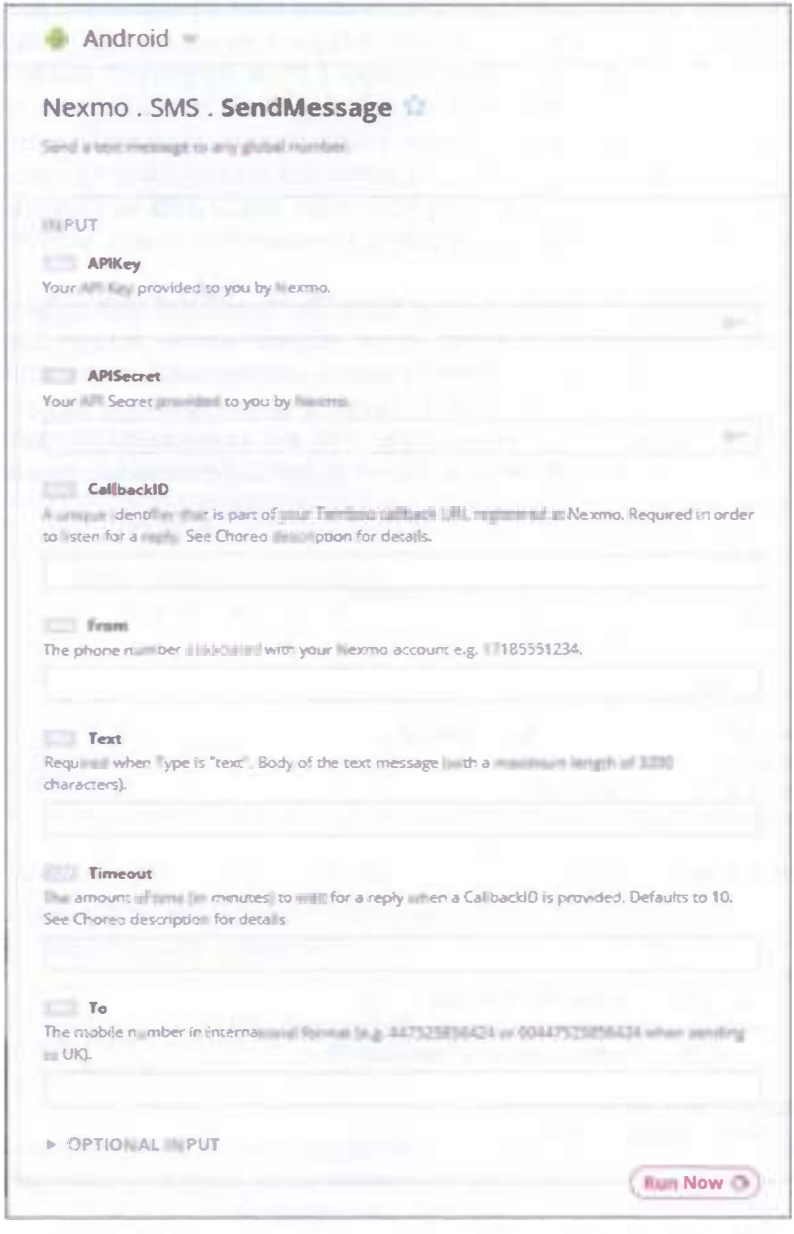


Рис. 140 ❖ Окно настройки функции отправки сообщений Nexmo

Вам нужно заполнить все поля, кроме CallbackID и Timeout. При этом важно, чтобы вверху вы выбрали Android, как показано на рис. 140. После заполнения всех полей вы автоматически получите код ниже, в разделе **code**. Теперь мы можем интегрировать код в наше Android Things-приложение – нам понадобятся некоторые части этого кода на следующих этапах.

3.2. Откройте приложение в Android Studio. Нам понадобятся две библиотеки из архива: temboo-android-sdk-core-xxx.jar и Nexmo-xxx.jar (текущая версия 2.22.0, мы будем использовать версию 2.19.0). Загрузите эти библиотеки из источников [17] и [18]. Добавьте их в библиотеки проекта (поменяйте вид с Android на Project в Project Explorer слева вверху, скопируйте библиотеки и вставьте их в папку things -> libs; по необходимости нажмите правой кнопкой мыши по каждой из них и выберите пункт меню **Add as Library** (если вы видите этот пункт меню **Add as Library** в контекстном меню, значит, необходимость есть)).

Теперь нам нужно добавить новый класс TembooClient.java (в папке things -> java ->packagename). Этот класс будет обрабатывать детали интеграции с Temboo. Ядром этого класса является метод, показанный ниже, который вызывает Temboo choreo и оборачивает choreo-код, который мы сконфигурировали в предыдущем пункте (**убедитесь, что вы поменяли значения констант ID, KEY, APIKey, APISecret и To на ваши собственные значения**, см. комментарии в коде с инструкциями, откуда что брать):

```
import android.util.Log;

import com.temboo.Library.Nexmo.Voice.TextToSpeech;
import com.temboo.core.TembooException;
import com.temboo.core.TembooSession;

public class TembooClient {

    private static final String TAG = "Temboo";
    private static TembooClient me;
    private TembooSession session;
    private static final String ID = ""; //enter your ID - the first parameter from the
Temboo CODE section: new TembooSession(
private static final String KEY = ""; //enter your KEY - the last parameter from the Temboo
CODE section: new TembooSession(

    private TembooClient() {
        try {
            Log.d(TAG, "Temboo session init...");
            session = new TembooSession(ID, "myFirstApp", KEY);
            Log.d(TAG, "Temboo session ["+session+"]");
        }
        catch (TembooException te) {
            te.printStackTrace();
        }
    }

    public static TembooClient getInstance() {
```

```
        if (me == null)
            me = new TembooClient();

        return me;
    }

    public void callTemboo() {
        Runnable r = new Runnable() {
            @Override
            public void run() {
                Log.d(TAG, "Call Temboo...");
                TextToSpeech textToSpeechChoreo = new TextToSpeech(session);

// Get an InputSet object for the choreo
                TextToSpeech.TextToSpeechInputSet textToSpeechInputs = textToSpeechChoreo.
newInputSet();

// Set inputs
textToSpeechInputs.set_APIKey(""); //enter your Key from Nexmo
textToSpeechInputs.set_Text("Hello, the temperature is too high");
textToSpeechInputs.set_To(""); // russian number pattern is 7XXXXXXXXXX
textToSpeechInputs.set_APISecret(""); //enter your Secret from Nexmo

// Execute Choreo
                try {
                    TextToSpeech.TextToSpeechResultSet textToSpeechResults =
textToSpeechChoreo.execute(textToSpeechInputs);
                    Log.d(TAG, "TTS Result ["+textToSpeechResults.get_Response()+"]");
                }
                catch (TembooException te) {
                    te.printStackTrace();
                }

            }
        };

        Thread t = new Thread(r);
        t.start();
    }

}
```

И последнее, что мы должны сделать, – это вызвать этот класс в методе onSensorChanged класса TemperatureCallback:

```
if (val >= 30) {
    turnOn = true;
    TembooClient client = TembooClient.getInstance();
    client.callTemboo();
}
```

Теперь мы, наконец, можем запустить приложение и проверить, **звонит ли нам платформа в случае, если температура больше либо равна 30 °C**. Заметьте, что при этом, согласно логике проекта из предыдущего задания, который мы использовали, красный светодиод будет гореть в этой ситуации!

Номер звонящего будет неизвестен. Вы можете изменить код, так как звонки будут поступать постоянно!

Пятый проект в Android Things – шпионский глаз

0. В этом задании мы создадим проект, который использует Android Things-плату, чтобы контролировать камеру и сервомотор, который мы используем для поворота камеры. Также мы научимся использовать PWM-пины платы (Pulse Width Modulation, широтно-импульсная модуляция). ОС Android Things поддерживает CSI-2-протокол для камеры (Camera Serial Interface, последовательный интерфейс камеры) так же, как и плата Raspberry Pi 3, у которой есть два CSI-2-порта – один для камеры и один для монитора.

1. Нам понадобятся:
- соединительные провода,
- модуль с Raspberry-камерой;
- сервомотор;
- плата Raspberry Pi 3;
- HDMI-монитор;
- кабель HDMI m-m;
- адаптер питания / кабель для HDMI-монитора;
- адаптер питания для платы Raspberry Pi 3 (5 В).

Модуль с Raspberry камерой выглядит так:

Рис. 141 ❖ Raspberry-камера

Модуль с камерой очень популярен в приложениях по охране дома и в ловушках с камерой для дикой природы. Камера обладает следующими характеристиками:

- напряжение: 5 В;
- ток, минимум: 1.8 А ;
- интерфейс: CSI-2;
- тип сенсора: Sony IMX 219 PQ CMOS ¼ дюйма;
- 8 MegaPixels максимум (3280×2464);
- поддерживаемые видеоформаты: 1080 p (30 fps), 720 p (60 fps), 640×480 p (90 fps);
- фокусное расстояние: 33 м.

И после практики на Arduino Uno 3 вам должен быть знаком сервомотор и PWM (ШИМ):

Рис. 142 ❖ Сервомотор

Сервомотор обладает следующими характеристиками:

- напряжение: 3–5 В;
- усилие на вале: 1.2 кг/см (4.8 В); 1.6 кг/см (6.0 В);
- ширина импульса: 2 мс;
- максимальный угол: 160 градусов.

Широтно-импульсная модуляция используется для того, чтобы формировать на выходе переменное напряжение, применяя цифровой сигнал. Она может быть использована для управления сервомоторами, интенсивностью освещения (светодиодами), звуком и аудио. ШИМ основана на изменении времени, при котором сигнал высокий. Есть два важных фактора в этой модуляции: **частота** и **рабочий цикл**. Например, рабочий цикл равен 50%, если сигнал высокий половину времени (периода), а другую половину периода – низкий. В Android Things нам нужно знать как частоту, так и рабочий цикл после открытия PWM-пина и перед активацией PWM-сигнала.

Перед подключением устройств к плате убедитесь, что она отключена от компьютера и питания, иначе вы можете повредить её или устройства. **Пожалуйста, будьте очень аккуратны с камерой!**

Следующая схема с макетной платой показывает, как подключить сервомотор и камеру к Raspberry Pi 3. Чтобы подключить камеру, сдвиньте вверх белый замок на CSI-2-порте для камеры, затем вставьте шлейф камеры так, чтобы контакты на шлейфе смотрели в противоположную сторону по отношению к белому замку (или так, чтобы синий ключ/полоса на конце шлейфа смотрел в сторону белого замка), затем опустите замок вниз (оба конца), чтобы обеспечить хороший контакт. Если камера подключена правильно, вы должны видеть её характеристики и картинку с трансляцией (live feed) с камеры в небольшом окне сверху характеристик, если зайдёте в пункт меню **Android Things -> Peripherals -> Camera** (включая поддерживаемые режимы работы и ID камеры: Camera 0). Если вы не видите «прямой эфир» с камеры в этом

окне или пункт меню **Camera** недоступен – значит либо вы подключили камеру неправильно, либо камера не работает (свойства камеры доступны в окне, открывающемся по щелчку на активном пункте меню **Camera**, но картинки с камеры нет).

Подключите сервомотор к плате и откройте маленькую упаковку (пакетик) с тремя насадками на серво и тремя шурупами – используйте их и металлический угловой держатель для камеры с двумя отверстиями (или любой другой угловой держатель), чтобы прикрепить сервомотор к камере. Вам понадобится крестовая отвёртка. Я рекомендую сначала ввернуть шурупы в небольшие дырки на белой насадке для сервомотора, чтобы расширить дырки, а затем вывернуть их обратно и ввернуть их снова, в этот раз продев через дырку в модуле камеры и дырку углового металлического держателя.

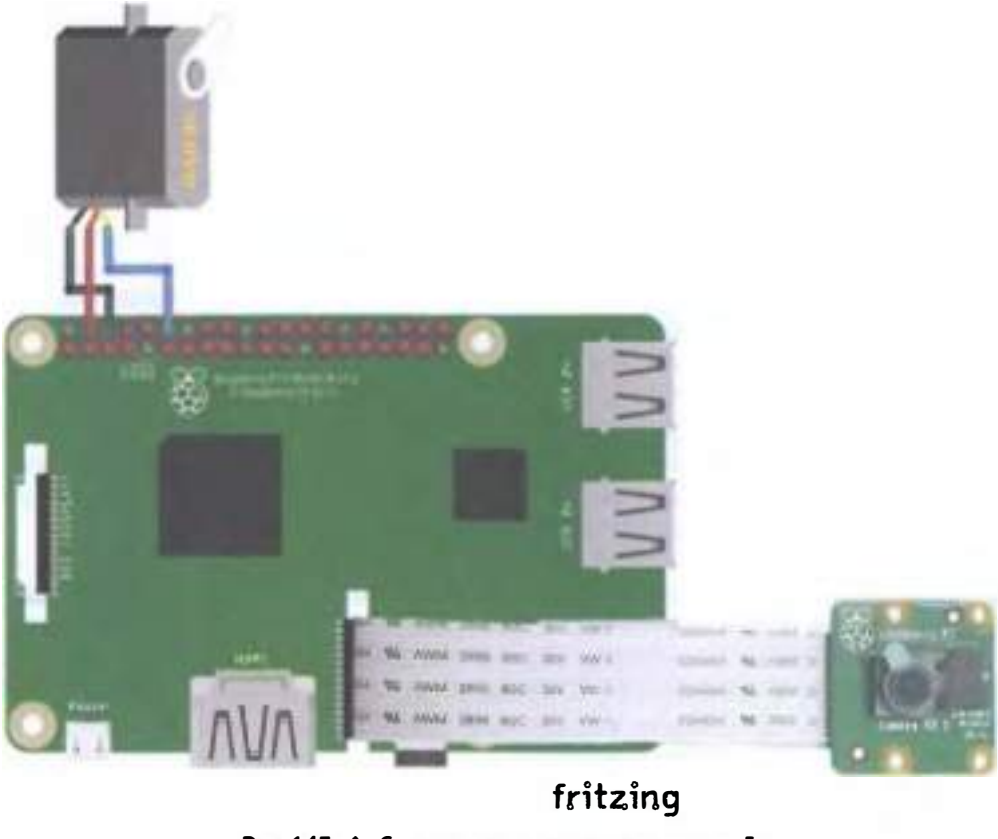


Рис. 143 ❖ Схема подключения для задания 5

2. Создайте новый проект в Android Studio. Дайте имя проекту, в следующем окне выберите дополнительно с Phone and Tablet платформу Android Things внизу и minimum SDK установите как API 24 (устанавливается по умолчанию).

В следующем окне выберите **Empty Activity**, дайте имя activity, затем в следующем окне выберите **Android Things Empty Activity**; потом нажмите **Next** и **Finish**.

Прежде всего нам нужно создать пользовательский интерфейс для нашего приложения, чтобы управлять сервомотором, который поворачивает камеру, и фотографировать помещение. Пользовательский интерфейс должен выглядеть так (две кнопки, чтобы поворачивать камеру влево/вправо (используйте для них значения **android:text=">"** и **android:text="<"**), и одна кнопка для фотографирования + ImageView, в котором будет отображаться фотография):

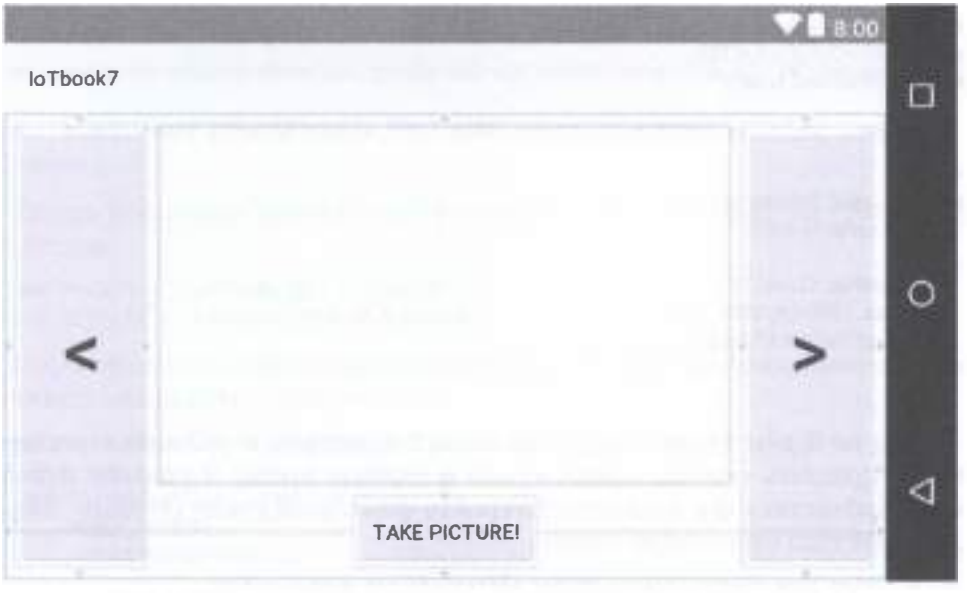


Рис. 144 ❖ Пример пользовательского интерфейса для задания 5

Чтобы открыть PWM-пин, мы могли бы использовать уже знакомый класс PeripheralManager (заметьте, что нам также нужен созданный ранее класс BoardPins в этом задании):

```
Pwm pwmPin;

PeripheralManager pm = PeripheralManager.getInstance();
try {
    pwmPin = (Pwm) pm.openPwm(getPWMPin()); // will give us «PWM0»
} catch (IOException e) {
    e.printStackTrace();
}
```

И следующий код должен быть добавлен в файл BoardPins.java:

```
public static String getPWMPin() {
    switch (getBoardName()) {
```

```
        case RASPBERRY:
            return "PWM0";
        case EDISON_ARDUINO:
            return "IO3";
        default:
            throw new IllegalArgumentException("Unsupported device");
    }
}
```

pwmPin – это экземпляр класса Pwm, у которого есть 4 метода: setPwmFrequencyHz(), setPwmDutyCycle(), setEnabled() и close(). Поэтому для частоты 50 Гц и 75 % рабочего цикла код будет такой:

```
pwmPin.setPwmFrequencyHz(50);
pwmPin.setPwmDutyCycle(75);
pwmPin.setEnabled(true);
```

Также вы должны корректно закрывать пин, когда Activity уничтожается:

```
@Override
protected void onDestroy() {
    super.onDestroy();
    try {
        pwmPin.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Но мы не будем использовать код выше с частотами и рабочими циклами. Чтобы управлять сервомотором, лучше и гораздо проще и удобнее использовать библиотеку его драйвера. Откройте файл build.gradle (Module: things) и добавьте туда следующую строку:

```
implementation 'com.google.android.things.contrib:driver-pwmservo:1.0'
```

Синхронизируйте проект, перейдите в файл MainActivity.java и добавьте следующий код в раздел объявления переменных класса:

```
private static final String TAG = MainActivity.class.getSimpleName();

private Servo mServo;

private Button btnPicture;
private ImageView imgView;

private int angle = 0;
private final int STEP = 15;
```

Затем добавьте следующий код внутрь класса:

```
private void initServo() {
    try {
        mServo = new Servo(getPWMPin());// will give us «PWM0»
        mServo.setAngleRange(0f, 180f);
```

```
            mServo.setEnabled(true);

        }
        catch(Exception e) {
            e.printStackTrace();
        }
}
```

В этом коде мы устанавливаем имя пина (**PWM0**) и минимальные и максимальные углы поворота вала сервомотора, которые определяют диапазон угла вращения. Затем мы активируем пин. Так как мы используем метод getPWMPin() класса BoardPins.java, то должны добавить разрешение управления периферией по входу и выходу в файл AndroidManifest.xml:

```
<uses-permission android:name="com.google.android.things.permission.USE_PERIPHERAL_IO" />
```

В методе onCreate() мы должны вызвать рассмотренный метод:

```
initServo();
```

Теперь нам нужно добавить две кнопки, чтобы контролировать работу сервомотора:

```
Button btnLeft = findViewById(R.id.btnLeft);
Button btnRight = findViewById(R.id.btnRight);
```

После этого мы должны обрабатывать события нажатия этих кнопок и поворачивать вал серво соответственно:

```
btnLeft.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        angle += STEP;
        setServoAngle(angle);
    }
});

btnRight.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        angle -= STEP;
        setServoAngle(angle);
    }
});
```

И наконец, мы должны добавить метод setServoAngle():

```
private void setServoAngle(int angle) {
    if (angle > mServo.getMaximumAngle())
        angle = (int) mServo.getMaximumAngle();

    if (angle < mServo.getMinimumAngle())
        angle = (int) mServo.getMinimumAngle();

    try {
```

```
            mServo.setAngle(angle);
        } catch (IOException e) {
            e.printStackTrace();
        }
}
```

Вы можете запустить Android Things-приложение (вам придётся подключить плату к HDMI-монитору и использовать USB-мышь, которую также надо подключить к плате, либо подключить кабель USB-USB к входу сенсорного управления на мониторе и к плате и воспользоваться сенсорным экраном монитора, если у вас есть сенсорный экран), проверить, работает ли управление сервомотора через интерфейс приложения.

3. Теперь поработаем с камерой. Чтобы работать с ней, мы будем использовать пакет **android.hardware.camera2** (добавленный с API уровня 21). Он содержит класс **CameraManager**, чтобы определять и открывать камеру, класс **CameraDevice** для установки её свойств и возможностей и класс **CaptureSession** для получения фотографии и представления её на какой-то поверхности.

Добавим ещё один класс в проект, назовём этот класс AndroidCamera.java (things -> java -> packagename -> щелчок правой кнопкой мыши -> new -> Java class).

3.1. Прежде всего мы должны обнаружить камеру и убедиться, что она подключена к плате, поэтому мы используем класс CameraManager. Объявим некоторые переменные в классе AndroidCamera.java:

```
private Context ctx;
private static final String TAG = "AndroidCamera";
private String camId;
private CameraManager cManager;
private CameraDevice camera;
private CameraCaptureSession session;
private CameraListener listener;
private ImageReader iReader;

private HandlerThread imgHandler = new HandlerThread("ImageThread");

public AndroidCamera(Context ctx, CameraListener listener) {
    this.ctx = ctx;
    this.listener = listener;
}
```

И затем создадим новый метод initCamera():

```
public void initCamera() {

    cManager =  (CameraManager) ctx.getSystemService(Context.CAMERA_SERVICE);
    Log.d(TAG, "Camera Manager ["+cManager+"]");

    // Retrieve the list of cams
    try {
        String[] idCams = cManager.getCameraIdList();
        Log.d(TAG, "Camera Ids ["+idCams+"]");
        camId = idCams[0];
```

```
    } catch (CameraAccessException e) {
        e.printStackTrace();
    }

    // Initialize the ImageReader
    imgHandler.start();
    iReader = ImageReader.newInstance(640, 480 , ImageFormat.JPEG, 1);
    iReader.setOnImageAvailableListener(new ImageReader.OnImageAvailableListener() {
        @Override
        public void onImageAvailable(ImageReader reader) {
            listener.onImageReady(reader);
        }
    }, new Handler(imgHandler.getLooper()));
}
```

В этом методе мы сначала получаем ссылку на менеджер камеры, затем приложение перечисляет все подключённые к плате камеры. Мы будем использовать первую же найденную камеру (с индексом [0]). Также мы должны инициализировать контейнер картинки, который используется приложением для получения прямого доступа к данным, переведённым на какую-то поверхность. После создания обработчика (Handler), требуемого объектом ImageReader, и установки формата изображения мы используем ImageReader, который содержит только одну картинку. Затем мы подключаем слушателя (listener), чтобы этот класс получал уведомление, когда фотография становится доступной. В свою очередь, класс AndroidCamera использует другой слушатель (listener) для уведомления вызвавшего его класса (MainActivity.java) о том, что изображение доступно.

3.2. Следующий шаг – это реализовать метод, который используется для начала общения с камерой:

```
public void openCamera() {
    try {
        CameraCharacteristics characteristics = cManager.getCameraCharacteristics(camId);
        cManager.openCamera(camId, stateCallback, null);

    } catch (CameraAccessException e) {
        e.printStackTrace();
    } catch (SecurityException se)  {
        se.printStackTrace();
    }
}
```

3.3. Теперь нам нужно создать callback-класс, использованный на предыдущем шаге, чтобы получать уведомления, когда происходят события, связанные с открытием камеры:

```
private final CameraDevice.StateCallback stateCallback = new CameraDevice.StateCallback() {
    @Override
    public void onOpened(@NonNull CameraDevice camera) {
        Log.d(TAG, "Camera opened");
        AndroidCamera.this.camera = camera;
        listener.onCameraAvailable();
```

```
    }

    @Override
    public void onDisconnected(@NonNull CameraDevice camera) {
        Log.d(TAG, "Camera disconnected");
    }

    @Override
    public void onError(@NonNull CameraDevice camera, int error) {
        Log.d(TAG, "Camera Error" + error);
    }
};
```

Есть несколько методов, которые надо добавить в callback-класс. Мы заинтересованы в методе, вызывающемся тогда, когда открывается камера, потому что мы храним экземпляр **CameraDevice**, для того чтобы обращаться к подключённой камере на следующих шагах. В то же время в том же методе мы информируем вызвавший класс (caller) о том, что камера подключена.

3.4. Как только камера подключена, мы можем задействовать метод, чтобы сделать фотографию:

```
public void takePicture() {
    try {
        camera.createCaptureSession(Collections.singletonList(iReader.getSurface()),
                sessionCallback, null);
    }
    catch(Exception e) {
        e.printStackTrace();
    }

}
```

Приложение создаёт *сессию получения изображения (capture session)*, которая используется для фотографирования. Мы используем callback-класс, чтобы быть проинформированными об этих событиях. Заметьте, что метод **createCaptureSession** использует поверхность **ImageReader** для удержания фотографии.

3.5. Ниже представлен callback-метод для обработки *событий, связанных с сессией получения изображения (capture session events)*:

```
private CameraCaptureSession.StateCallback sessionCallback = new CameraCaptureSession.
StateCallback() {
    @Override
    public void onConfigured(@NonNull CameraCaptureSession session) {
        Log.d(TAG, "Camera configured");
        AndroidCamera.this.session = session;
        startCaptureImage();
    }

    @Override
    public void onConfigureFailed(@NonNull CameraCaptureSession session) {
        Log.e(TAG, "Configuration failed");

    }
};
```

Здесь важным является метод **onConfigured**, который вызывается, когда камера готова получить изображение и конфигурационный процесс закончен. Приложение использует этот метод для того, чтобы начать захват изображения.

3.6. Последний шаг – это разработка метода, который на самом деле получает изображение:

```
private void startCaptureImage() {
    try {
        CaptureRequest.Builder captureBuilder =
                camera.createCaptureRequest(CameraDevice.TEMPLATE_STILL_CAPTURE);
        captureBuilder.addTarget(iReader.getSurface());
        captureBuilder.set(CaptureRequest.CONTROL_AE_MODE, CaptureRequest.CONTROL_AE_MODE_
ON);
        Log.d(TAG, "Session initialized.");
        session.capture(captureBuilder.build(), captureCallback, null);
    }
    catch(CameraAccessException cae) {
        cae.printStackTrace();
    }

}
```

Этот метод подготавливает запрос, устанавливая некоторые параметры, и запускает сессию получения изображения. Как обычно, мы используем callback-метод, чтобы получать уведомления о событиях.

3.7. И наконец, мы определяем callback-интерфейс (или слушатель/listener), используемый классом AndroidCamera для уведомления вызывающего класса о наиболее важных событиях:

```
public static interface CameraListener {
    public void onCameraAvailable();
    public void onImageReady(ImageReader reader);
}
```

И добавляем отсутствующий captureCallback:

```
private CameraCaptureSession.CaptureCallback captureCallback = new CameraCaptureSession.
CaptureCallback() {
    @Override
    public void onCaptureCompleted(@NonNull CameraCaptureSession session, @NonNull
CaptureRequest request, @NonNull TotalCaptureResult result) {
        Log.d(TAG, "Capture completed");
        session.close();
    }
};
```

4. Теперь нам нужно изменить MainActivity.hava, чтобы закончить Android Things-приложение. Мы должны позаботиться о кнопке, которая фотографирует помещение. В методе onCreate добавьте следующее:

```
final AndroidCamera aCamera = new AndroidCamera(this, listener);
aCamera.initCamera();
aCamera.openCamera();
```

Камера была инициализирована. Теперь нам нужно добавить ссылку на ImageView, который будет показывать фотографию:

```
imgView = findViewById(R.id.img);
```

И – на кнопку:

```
btnPicture = findViewById(R.id.btnPicture);
btnPicture.setEnabled(false);
```

Изначально кнопка неактивна, пока камера не готова фотографировать. Для этого приложение использует слушатель, чтобы знать, когда камера готова, и активировать кнопку (в методе **onCameraAvailable**).

Событие onClick для этой кнопки выглядит так:

```
btnPicture.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Log.d(TAG, "Start capturing the image");
        aCamera.takePicture();
    }
});
```

Обработчик вызывает метод **takePicture**, чтобы сделать фотографию.

Наконец, приложение реализует callback-интерфейс, который мы использовали ранее в конструкторе:

```
private AndroidCamera.CameraListener listener = new AndroidCamera.CameraListener() {
    @Override
    public void onCameraAvailable() {
        Log.d(TAG, "Camera Ready");
        btnPicture.setEnabled(true);
    }

    @Override
    public void onImageReady(ImageReader reader) {
        Log.d(TAG, "Image ready");
        Image img1 = reader.acquireLatestImage();
        ByteBuffer bBuffer = img1.getPlanes()[0].getBuffer();
        final byte[] buffer = new byte[bBuffer.remaining()];
        bBuffer.get(buffer);
        img1.close();
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                imgView.setImageBitmap(BitmapFactory.decodeByteArray(buffer, 0, buffer.
length));

            }
        });
    }
};
```

В первом callback-методе, названном **onCameraAvailable()**, приложение активирует кнопку, как только камера готова. Во втором методе, названном

onImageReady, мы обновляем изображение, помещая фотографию в виджет **ImageView**. Код, содержащийся в этом методе, используется для извлечения изображения и его адаптации к виду, приемлемому в ImageView.

Возможно, вы уже заметили ошибку в методе openCamera класса AndroidCamera.java. Она возникла из-за того, что мы до сих пор не добавили разрешение на использование камеры в файл манифеста AndroidManifest.xml. Вы можете исправить ошибку, нажав на красную лампочку с восклицательным знаком (выберите **Add Permission CAMERA**), или добавить следующую строку в файл манифеста вручную:

```
<uses-permission android:name="android.permission.CAMERA" />
```

Поздравляю! Работа над приложением закончена, и теперь вы можете подключить к питанию вашу плату Raspberry Pi 3, HDMI-монитор, подключиться к плате через ADB и запустить приложение снова, чтобы проверить его и поэкспериментировать. После получения фотографии вы должны увидеть в интерфейсе вашего приложения что-то, похожее на рис. 145.

Рис. 145 ❖ Скриншот работающего приложения после получения фотографии с помощью камеры

Заключение

Прочитав эту книгу и пройдя все практические задания, вы получили базовый опыт в области интернета вещей и, что немаловажно, в основном практический опыт работы с реальным оборудованием, с реальными облачными инструментами для интернета вещей, наряду с некоторыми теоретическими положениями. Практический опыт гораздо ценнее, особенно для университетской среды, где, как правило, до сих пор делают упор на теоретические лекции, почти не уделяя времени практике. Вы вспомнили основы физики, закон Ома, увидели много датчиков, модулей и прочих сенсоров в работе, и у вас наверняка появились идеи разработки нескольких Android Things- или Arduino-приложений для своих нужд, при использовании различных комбинаций датчиков, модулей, индикаторов и программного кода.

Кроме того, возможно, вы усвоили правила определения ошибки в отдельно взятом проекте, если что-то не работает. Эти правила сложнее, чем если бы вам требовалось найти ошибку только в коде программы, т. е. с точки зрения программиста. Для определения ошибки начать нужно с анализа и проверки схемы подключения компонентов. При этом надо помнить, что подключение пинов на модуле к плате может не совпадать со схемой, так как на самих модулях есть собственная информация о назначении этих пинов. Затем – переходить к анализу кода: возможно, ошибка скрывается именно в нём. Если всё правильно и ошибка всё равно не уходит, возможно, надо попробовать поменять компоненты схемы, один за другим. Дело может быть совсем не в схеме или в коде, а в конкретном модуле, подключённом в схему, который просто работает неправильно или перестал работать. У меня был случай с камерой: я захотел выполнить проект с Хабрахабра по умной дверной консоли, которая автоматически открывает дверь с помощью сервомотора с RFID-картой изнутри помещения, если получен положительный ответ распознавания лица человека, подходящего к двери, с помощью облачного сервиса Google Vision, и всё это происходит в светлое время суток. Всё было хорошо, но проект не работал, так как не было изображения с камеры – сессия получения изображения не завершалась, а в коде не было никаких ошибок, так как камера определялась правильно, открывалась, и ей даже присваивался ID. В конечном итоге дело оказалось именно в камере, которая просто была бракованной, так как не могла передавать изображение даже в Android Things, если выбрать пункт меню Camera. Однако я этого не знал и искал ошибку в коде. А код у этого проекта был увесистый и состоял из нескольких классов. Таким образом, можно потратить массу времени совершенно впустую, поэтому важно возвращаться к этим трём этапам проверки и проходить их снова при необходимости, не застревая на каком-то конкретном этапе надолго, например на анализе кода. Важно так-

же понимание того, сколько займёт времени каждый из этих этапов, и сначала выполнять самые быстрые этапы, а трудные и времязатратные оставлять на крайний случай, когда ничего не помогает. Если же в нахождении ошибки не помогает ни один из перечисленных этапов, можно обратиться к аналогичным проектам в интернете, но такое случается крайне редко.

Помимо всего прочего, освоив это учебное пособие, вы получили некоторые навыки работы со средами программирования Arduino IDE и Android Studio, которые вам наверняка пригодятся в будущем, ведь Android Studio – это официальная среда разработки нативных Android-приложений. Я надеюсь, вам понравилась эта книга!

Список использованных источников

1. Комплект интернет-вещей Arduino Uno R3 Starter Learning Kit с RFID-модулем [Электронный ресурс] // Интернет-магазин onpad.ru. – URL: http://onpad.ru/shop/cubie/arduino/ardiuno_kit/1680.html.
2. Arduino IDE [Электронный ресурс] // Официальный сайт Arduino. Язык: английский. – URL: https://www.arduino.cc/en/Main/Software.
3. Arduino Create Online IDE [Электронный ресурс] // Официальный сайт Arduino. Язык: английский. – URL: https://create.arduino.cc/editor.
4. Первый урок-инструкция по Arduino IDE [Электронный ресурс] // Официальный сайт Arduino. Язык: английский. – URL: https://www.arduino.cc/en/Guide/ArduinoUno.
5. Библиотека Keypad.h для работы с кнопочным модулем 4×4 [Электронный ресурс] // Официальный сайт Arduino. Язык: английский. – URL: http://arduino.cc/playground/uploads/Code/Keypad.zip.
6. Исправленная библиотека DS1302.h для работы с модулем часов реального времени [Электронный ресурс] // Matt Sparks, Github. Язык: английский. – URL: https://github.com/msparks/arduino-ds1302.
7. Библиотека IRemote.h для работы с инфракрасным модулем [Электронный ресурс] // *Макаров С. Л.* Официальная страница курса «Экосистемы интернета вещей». Язык: английский. – URL: http://serjmak.com/2students/IoTE/IRremote.zip.
8. Исправленная библиотека MFRC522.h для работы с RFID-модулем RC522 [Электронный ресурс] // *Макаров С. Л.* Официальная страница курса «Экосистемы интернета вещей». Язык: английский. – URL: http://serjmak.com/2students/IoTE/mfrc522.zip.
9. Архив из 34 практических заданий на английском языке и некоторых библиотек [Электронный ресурс] // Интернет-магазин onpad.ru (ссылка взята из [1]). Язык: английский. – URL: https://yadi.sk/d/YUYx69mEm3xPV.
10. Открытая библиотека и редактор электронных компонентов и схем Fritzing [Электронный ресурс] // Friends-of-Fritzing foundation. Язык: английский. – URL: http://fritzing.org/home./
11. Резистор [Электронный ресурс] // Амперка вики. – URL: http://wiki.amperka.ru/конспект-arduino:резистор.
12. Светодиод [Электронный ресурс] // Амперка вики. – URL: http://wiki.amperka.ru/конспект-arduino:светодиод.

13. Код программы 1 для задания 34 [Электронный ресурс] // Загружено на сайт serjmak.com 12 октября 2018. – URL: http://serjmak.com/2students/IoTE/SetPassword.txt.
14. Код программы 2 для задания 34 [Электронный ресурс] // Загружено на сайт serjmak.com 12 октября 2018. – URL: http://serjmak.com/2students/IoTE/DoorCon.txt
15. Android Things Projects [Text] // *Francesco Azzola*. – Packt, 2017. – 232 p. – ISBN: 9781787289246.
16. Raspberry Pi 3 [Electronic resource] // Official Google site for Android Things developers, last updated: September 25, 2018. – URL: https://developer.android.com/things/hardware/raspberrypi#io-pinout.
17. Библиотека Nexmo (версия 2.19.0) [Электронный ресурс] // Загружено на сайт serjmak.com: 4 октября 2018. – URL: http://serjmak.com/2students/IoTE/Nexmo-2.19.0.jar.
18. Библиотека Temboo (версия 2.19.0) [Электронный ресурс] // Загружено на сайт serjmak.com: 4 октября 2018. – URL: http://serjmak.com/2students/IoTE/temboo-android-sdk-core-2.19.0.jar.
19. Build a Proof of Concept (former Hardware 101 page) [Электронный ресурс] // Сайт разработчиков под Android, Google Inc., обновлено 7 августа 2018. – URL: https://developer.android.com/things/get-started/proof-of-concept.
20. Raspberry Pi 3 Model B [Электронный ресурс] // Сайт ООО «Амперка», 2018. – URL: http://amperka.ru/collection/soc-boards/product/raspberry-pi-3-model-b.
21. Raspberry Pi 3 Official Starter Kit 16GB Black – Комплект с Raspberry Pi 3 [Электронный ресурс] // Интернет-магазин onpad.ru. – URL: https://onpad.ru/catalog/cubie/raspberrypi/raspberrypi_kit/kit_raspberrypi3/2279.html.
22. Учебное пособие по практическим занятиям в рамках курса «Экосистемы интернета вещей» (IoT Ecosystems) [Электронный ресурс] // Загружено на сайт serjmak.com: 12 ноября 2018. – URL: http://serjmak.com/2students/IoTE/Makarov_S.L._Uchebnoye_posobiye_po_IoT_2018.docx.

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «Планета Альянс» наложенным платежом,
выслав открытку или письмо по почтовому адресу:
115487, г. Москва, 2-й Нагатинский пр-д, д. 6А.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.alians-kniga.ru**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@alians-kniga.ru**.

Сергей Львович Макаров

Arduino Uno **и** Raspberry Pi 3:
от схемотехники к интернету вещей

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Гарнитура «PT Serif». Печать офсетная.
Усл. печ. л. 16,41. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**

Основы информационных технологий

Новиков Ю. В.

ВВЕДЕНИЕ В ЦИФРОВУЮ СХЕМОТЕХНИКУ

Учебное пособие

Интернет-Университет
Информационных Технологий
www.intuit.ru

БИНОМ.
Лаборатория знаний
www.lbz.ru

Москва
2007

УДК [004:621.38](075.4)
ББК 32.844я78-1+32.973.2-02я78-1
Н73

Новиков Ю.В.

Н73 Введение в цифровую схемотехнику / Ю.В. Новиков — М: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007. — 343 с: ил., табл. — (Серия «Основы информационных технологий»).

ISBN 5-94774-600-X (БИНОМ.ЛЗ)
ISBN 5-9556-0082-5 (ИНТУИТ.РУ)

Книга представляет собой краткое учебное пособие по основам цифровой схемотехники. В нем рассматриваются принципы работы цифровой электроники, базовые элементы цифровых схем, стандартные схемы включения этих элементов, алгоритмы проектирования цифровых устройств — от простейших до сложных.

Данное пособие предназначено для тех, кто желает самостоятельно освоить цифровую схемотехнику, для студентов соответствующих специальностей, а также может быть полезна специалистам, занимающимся разработкой и обслуживанием цифровых электронных систем.

УДК [004:621.38](075.4)
ББК 32.844я78-1+32.973.2-02я78-1

Издание осуществлено при финансовой и технической поддержке издательства «Открытые Системы», «РМ Телеком» и Kraftway Computers.

По вопросам приобретения обращаться:
«БИНОМ. Лаборатория знаний»
Телефон (499) 157-1902, (495) 157-5272,
e-mail: Lbz@aha.ru, http://www.Lbz.ra

ISBN 5-94774-600-X (БИНОМ.ЛЗ)
ISBN 5-9556-0082-5 (ИНТУИТ.РУ)

О проекте

Интернет-Университет Информационных Технологий — это первое в России высшее учебное заведение, которое предоставляет возможность получить дополнительное образование во Всемирной сети. Web-сайт университета находится по адресу www.intuit.ru.

Мы рады, что вы решили расширить свои знания в области компьютерных технологий. Современный мир — это мир компьютеров и информации. Компьютерная индустрия — самый быстрорастущий сектор экономики, и ее рост будет продолжаться еще долгое время. Во времена жесткой конкуренции от уровня развития информационных технологий, достижений научной мысли и перспективных инженерных решений зависит успех не только отдельных людей и компаний, но и целых стран. Вы выбрали самое подходящее время для изучения компьютерных дисциплин. Профессионалы в области информационных технологий сейчас востребованы везде: в науке, экономике, образовании, медицине и других областях, в государственных и частных компаниях, в России и за рубежом. Анализ данных, прогнозы, организация связи, создание программного обеспечения, построение моделей процессов — вот далеко не полный список областей применения знаний для компьютерных специалистов.

Обучение в университете ведется по собственным учебным планам, разработанным ведущими российскими специалистами на основе международных образовательных стандартов Computer Curricula 2001 Computer Science. Изучать учебные курсы можно самостоятельно по учебникам или на сайте Интернет-Университета, задания выполняются только на сайте. Для обучения необходимо зарегистрироваться на сайте университета. Удостоверение об окончании учебного курса или специальности выдается при условии выполнения всех заданий к лекциям и успешной сдачи итогового экзамена.

Книга, которую вы держите в руках, — очередная в многотомной серии «Основы информационных технологий», выпускаемой Интернет-Университетом Информационных Технологий. В этой серии будут выпущены учебники по всем базовым областям знаний, связанным с компьютерными дисциплинами.

Добро пожаловать в
Интернет-Университет Информационных Технологий!

Анатолий Шкред
anatoli@shkred.ru

Предисловие

Цифровая электроника в настоящее время все более и более вытесняет традиционную аналоговую. Ведущие фирмы, производящие самую разную электронную аппаратуру, все чаще заявляют о полном переходе на цифровую технологию. Причем это относится как к бытовой технике (аудио-, видеоаппаратура, средства связи), так и к профессиональной технике (измерительная, управляющая аппаратура). Ставшие уже привычными персональные компьютеры также полностью реализованы на цифровой электронике. Видимо, в ближайшем будущем аналоговым устройствам будет отведена вспомогательная роль: они будут применяться в основном для связи цифровых систем с аналоговыми датчиками и аналоговыми исполнительными элементами.

Для обслуживания цифровой техники, тем более, для ее ремонта и разработки, требуются специалисты, досконально знающие принципы работы цифровых устройств и систем, базовые элементы цифровой электроники, типовые схемы их включения, правила взаимодействия цифровых узлов, способы построения наиболее типичных цифровых устройств. При этом в процессе подготовки таких специалистов необходимо учитывать следующие специфические особенности.

Во-первых, цифровая техника не слишком сильно связана с аналоговой техникой и с физическими эффектами, используемыми в электронике. Отсюда следует, что специалист по цифровой схемотехнике совсем не обязательно должен быть классным специалистом по аналоговой технике и по физическим основам электроники. Строго говоря, такому специалисту не очень важно, на каких электронных компонентах и на каких физических принципах построена проектируемая система и ее элементы. Гораздо важнее логика ее работы и протоколы взаимодействия цифровых элементов, узлов и устройств, входящих в систему.

Во-вторых, стать настоящим специалистом по разработке цифровых устройств и систем невозможно без овладения азбукой цифровой электроники. То есть разработчик обязан понимать логику работы таких базовых компонентов цифровой схемотехники, как логические элементы, буферы, триггеры, регистры, дешифраторы, мультиплексоры, счетчики, сумматоры, оперативная и постоянная память и т.д. Кроме того, он должен знать типовые схемы включения этих компонентов и правила их корректной работы. Даже если разрабатывается устройство на базе микросхем с программируемой логикой или на базе микроконтроллеров, такие знания совершенно необходимы.

Данная книга как раз посвящена самым основам цифровой схемотехники, ее азбуке, ее основным методам, подходам и приемам. Она, конечно, не претендует на то, чтобы охватить весь круг вопросов проектирова-

ния цифровых систем вплоть до интеллектуальных многокомпьютерных комплексов. Ее назначение — ввести в тему, разъяснить даже неподготовленному читателю, что, собственно, происходит в любом цифровом устройстве. В то же время, она позволяет достаточно глубоко освоить методы проектирования сравнительно простых цифровых устройств. После изучения материала книги читатель сможет и уверенно разбираться в работе готовых цифровых приборов, и проектировать новые устройства, приборы, системы.

Материал данной книги представляет собой тот необходимый минимум знаний, который должен иметь и которым должен свободно и активно пользоваться каждый профессиональный разработчик цифровой аппаратуры. Любые другие, дополнительные знания, конечно же, не повредят, но заменить собой то, что изложено здесь, они не смогут.

Возможно, подход, предлагаемый в данной книге, несколько отличается от общепринятого. Возможно также, что используемый набор терминов не полностью совпадает со стандартным (отечественные стандарты слишком часто меняются). Но главное — это научить проектировать цифровые устройства и системы, а какие для этого используются подходы и термины, наверное, не слишком принципиально.

В качестве учебного базиса выбрана хорошо зарекомендовавшая себя и вполне функционально полная серия логических микросхем SN74 компании Texas Instruments (отечественные аналоги — серии К155, К555, КР531, КР1533, КР1531 и др.). Также описываются микросхемы памяти, ЦАП и АЦП других компаний.

Книга написана на основе многолетнего личного опыта автора по разработке цифровых устройств, а также на базе материала учебных курсов, читаемых автором, доцентом кафедры электроники Московского инженерно-физического института (МИФИ).

Несколько слов о структуре книги.

Первая глава рассматривает основополагающие принципы цифровой электроники, знакомит с терминологией и основными правилами оформления схем.

Главы со второй по шестую посвящены основным базовым элементам цифровых устройств, типовым и нестандартным схемам их включения. Практически все приведенные схемы проверялись автором на практике. Микросхемы описаны, начиная с самых простейших логических элементов в порядке усложнения через комбинационные микросхемы, триггеры, регистры, счетчики до микросхем памяти. В каждой главе приведено множество примеров включений микросхем., как рассматриваемых в данной главе, так и рассмотренных в предыдущих главах.

Седьмая глава содержит краткие сведения о цифро-аналоговых и аналого-цифровых преобразователях и основных методах их включе-

ния в аналого-цифровых устройствах. Без этих сведений книга была бы не полна.

В восьмой главе даны примеры нескольких сравнительно сложных цифровых устройств с подробным описанием всех этапов проектирования и принципов работы и взаимодействия всех узлов и микросхем. Цель состоит не в том, чтобы читатель повторил материалы о данных устройствах, а в том, чтобы на практике показать приемы проектирования, которые затем позволят строить любые другие цифровые устройства.

В приложении приведены таблицы параметров микросхем, основных обозначений микросхем и сигналов, соответствия отечественных и зарубежных микросхем. В конце книги имеется подробный словарь терминов и сокращений цифровой схемотехники.

Об авторе

Новиков Юрий Витальевич, Кандидат технических наук, доцент факультета автоматики и электроники МИФИ, автор семи книг по электронике и компьютерным локальным сетям: "Разработка устройств сопряжения для персонального компьютера типа IBM PC", "Аппаратура локальных сетей: функции, выбор, разработка", "Локальные сети: архитектура, алгоритмы, проектирование", "Основы цифровой схемотехники. Базовые элементы и схемы, методы проектирования", "Основы микропроцессорной техники", "Основы локальных сетей", "Введение в цифровую схемотехнику".

Лекции

Содержание

Глава 1. Философия цифровой электроники

Лекция 1. Базовые понятия цифровой электроники

В лекции рассказывается о базовых терминах цифровой электроники, о цифровых сигналах, об уровнях представления цифровых устройств, об их электрических и временных параметрах.

Ключевые слова: сигналы аналоговые и цифровые, помехоустойчивость, логические уровни, модели цифровых устройств, таблица истинности, задержки, входные и выходные токи и напряжения, элементный базис, типы выходов, организация связей.

Пусть не пугает читателя слово «философия» в названии главы. В данном случае имеются в виду всего лишь главные принципы цифровой электроники и обоснование ее преимуществ.

1.1. Аналог или цифра?

Для начала дадим несколько базовых определений.

Сигнал — это любая физическая величина (например, температура, давление воздуха, интенсивность света, сила тока и т. д.), изменяющаяся со временем. Именно благодаря этому изменению сигнал может нести в себе какую-то информацию.

Электрический сигнал — это электрическая величина (например, напряжение, ток, мощность), изменяющаяся со временем. Вся электроника в основном работает с электрическими сигналами, хотя сейчас все больше используются световые сигналы, которые представляют собой изменяющуюся во времени интенсивность света.

Аналоговый сигнал — это сигнал, который может принимать любые значения в определенных пределах (например, напряжение может плавно изменяться в пределах от нуля до десяти вольт). Устройства, работающие только с аналоговыми сигналами, называются аналоговыми устройствами. Название «аналоговый» подразумевает, что сигнал изменяется аналогично физической величине, то есть непрерывно.

Цифровой сигнал — это сигнал, который может принимать только два (иногда — три) значения, причем разрешены некоторые отклонения от этих значений (рис. 1.1). Например, напряжение может принимать два значения: от 0 до 0,5 В (уровень нуля) или от 2,5 до 5 В (уровень едини-

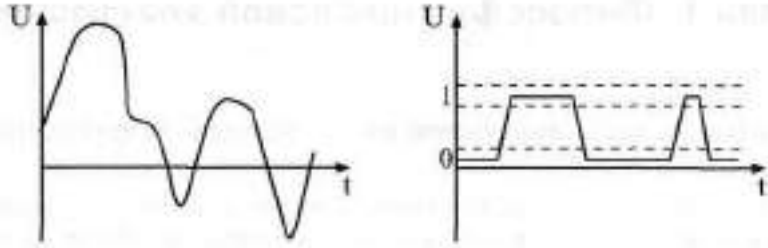


Рис. 1.1. Электрические сигналы: аналоговый (слева) и цифровой (справа)

цы). Устройства, работающие исключительно с цифровыми сигналами, называются цифровыми устройствами.

Можно сказать, что в природе практически все сигналы — аналоговые, то есть они изменяются непрерывно в каких-то пределах. Именно поэтому первые электронные устройства были аналоговыми. Они преобразовывали физические величины в пропорциональные им напряжение или ток, производили над ними какие-то операции и затем выполняли обратные преобразования в физические величины. Например, голос человека (колебания воздуха) с помощью микрофона преобразуется в электрические колебания, затем эти электрические сигналы усиливаются электронным усилителем и с помощью акустической системы снова преобразуются в колебания воздуха — в более сильный звук.

Однако аналоговые сигналы и работающая с ними аналоговая электроника имеют большие недостатки, связанные именно с природой аналоговых сигналов. Дело в том, что аналоговые сигналы чувствительны к действию всевозможных паразитных сигналов — шумов, наводок, помех. Шум — это внутренние хаотические слабые сигналы любого электронного устройства (микрофона, транзистора, резистора и т. д.). Наводки и помехи — это сигналы, приходящие на электронную систему извне и искажающие полезный сигнал (например, электромагнитные излучения от радиопередатчиков или от трансформаторов).

Все операции, производимые электронными устройствами над сигналами, можно условно разделить на три большие группы:

- обработка (или преобразование);
- передача;
- хранение.

Во всех этих трех случаях полезные сигналы искажаются паразитными — шумами, помехами, наводками. Кроме того, при обработке сигналов (например, при усилении, фильтрации) еще и искажается их форма — из-за несовершенства, неидеальности электронных устройств.

А при передаче на большие расстояния и при хранении сигналы к тому же ослабляются.

В случае аналоговых сигналов все это существенно ухудшает полезный сигнал, так как все его значения разрешены (рис. 1.2). Поэтому каждое преобразование, каждое промежуточное хранение, каждая передача по кабелю или эфиру ухудшает аналоговый сигнал, иногда вплоть до его полного уничтожения. Надо еще учесть, что все шумы, помехи и наводки принципиально не поддаются точному расчету, поэтому *точно* описать поведение любых аналоговых устройств абсолютно невозможно. К тому же со временем параметры всех аналоговых устройств изменяются из-за старения элементов, поэтому характеристики этих устройств не остаются постоянными.

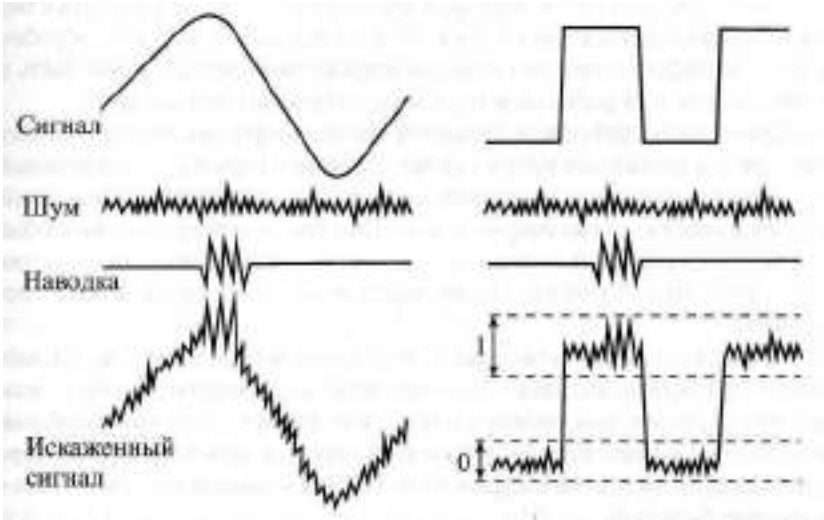


Рис. 1.2. Искажение шумами и наводками аналогового (слева) и цифрового (справа) сигналов

В отличие от аналоговых, цифровые сигналы, имеющие всего два разрешенных значения, защищены от действия шумов, наводок и помех гораздо лучше. Небольшие отклонения от разрешенных значений никак не искажают цифровой сигнал, так как всегда существуют зоны допустимых отклонений (рис. 1.2). Именно поэтому цифровые сигналы допускают гораздо более сложную и многоступенчатую обработку, гораздо более длительное хранение без потерь и гораздо более качественную передачу, чем аналоговые. К тому же поведение цифровых устройств всегда можно абсолютно точно рассчитать и предсказать. Цифровые устройства гораздо меньше подвержены старению, так как небольшое изменение их парамет-

ров никак не отражается на их функционировании. Кроме того, цифровые устройства проще проектировать и отлаживать. Понятно, что все эти преимущества обеспечивают бурное развитие цифровой электроники.

Однако у цифровых сигналов есть и крупный недостаток. Дело в том, что на каждом из своих разрешенных уровней цифровой сигнал должен оставаться хотя бы в течение какого-то минимального временного интервала, иначе его невозможно будет распознать. А аналоговый сигнал может принимать любое свое значение бесконечно малое время. Можно сказать и иначе: аналоговый сигнал определен в непрерывном времени (то есть в любой момент времени), а цифровой — в дискретном (то есть только в выделенные моменты времени). Поэтому максимально достижимое быстродействие аналоговых устройств всегда принципиально больше, чем цифровых. Аналоговые устройства могут работать с более быстро меняющимися сигналами, чем цифровые. Скорость обработки и передачи информации аналоговым устройством всегда может быть выше, чем скорость обработки и передачи цифровым устройством.

Кроме того, цифровой сигнал передает информацию только двумя уровнями и изменением одного своего уровня на другой, а аналоговый — еще и каждым текущим значением своего уровня, то есть он более емкий с точки зрения передачи информации. Поэтому для передачи того объема информации, который содержится в одном аналоговом сигнале, чаще всего приходится использовать несколько цифровых (чаще всего от 4 до 16).

К тому же, как уже отмечалось, в природе все сигналы — аналоговые, то есть для преобразования их в цифровые и обратного преобразования требуется применение специальной аппаратуры (аналого-цифровых и цифро-аналоговых преобразователей). Так что ничто не дается даром, и плата за преимущества цифровых устройств может порой оказаться неприемлемо большой.

1.2. Уровни представления цифровых устройств

Все цифровые устройства строятся из логических микросхем, каждая из которых (рис. 1.3) обязательно имеет следующие выводы (или, как их еще называют в просторечии, «ножки»):

- выводы питания: общий (или «земля») и напряжения питания (в большинстве случаев — +5 В или +3,3 В), которые на схемах обычно не показываются;
- выводы для входных сигналов (или «входы»), на которые поступают внешние цифровые сигналы;
- выводы для выходных сигналов (или «выходы»), на которые выдаются цифровые сигналы из самой микросхемы.

Каждая микросхема преобразует тем или иным способом последовательность входных сигналов в последовательность выходных сигналов. Способ преобразования чаще всего описывается или в виде таблицы (так называемой таблицы истинности), или в виде временных диаграмм, то есть графиков зависимости от времени всех сигналов.

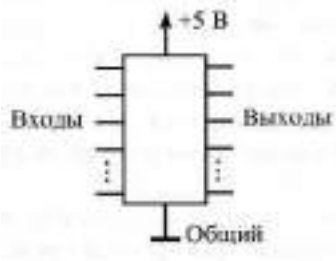


Рис. 1.3. Цифровая микросхема

Все цифровые микросхемы работают с логическими сигналами, имеющими два разрешенных уровня напряжения. Один из этих уровней называется уровнем логической единицы (или единичным уровнем), а другой — уровнем логического нуля (или нулевым уровнем). Чаще всего логическому нулю соответствует низкий уровень напряжения, а логической единице — высокий уровень. В этом случае говорят, что принята «положительная логика». Однако при передаче сигналов на большие расстояния и в системных шинах микропроцессорных систем порой используют и обратное представление, когда логическому нулю соответствует высокий уровень напряжения, а логической единице — низкий уровень. В этом случае говорят об «отрицательной логике». Иногда логический нуль кодируется положительным уровнем напряжения (тока), а логическая единица — отрицательным уровнем напряжения (тока), или наоборот. Есть и более сложные методы кодирования логических нулей и единиц. Но мы в основном будем говорить о положительной логике.

Для описания работы цифровых устройств используют самые различные модели, отличающиеся друг от друга сложностью, точностью, большим или меньшим учетом тонких физических эффектов. В основном эти модели используются при компьютерных расчетах цифровых схем. В настоящее время существуют компьютерные программы, которые не только рассчитывают готовые схемы, но способны и проектировать новые схемы по формализованным описаниям функций, которые данное устройство должно выполнять. Это довольно удобно, но ни одна программа никогда не может сравниться с человеком. По-настоящему эффективные, минимизированные по аппаратуре, наконец, красивые

схемы может разрабатывать только человек, который всегда подходит к проектированию творчески и использует оригинальные идеи.

Разработчик цифровой аппаратуры тоже использует своеобразные модели или, как еще можно сказать, различные уровни представления цифровых схем. Но, в отличие от компьютера, человек может гибко выбирать нужную модель — ему надо только взглянуть на схему, чтобы понять, где достаточно простейшей модели, а где требуется более сложная. То есть человек никогда не будет делать лишней, избыточной работы и, следовательно, не будет вносить дополнительных ошибок, свойственных любой, даже самой сложной, модели. Правда, простота цифровых устройств по сравнению с аналоговыми обычно не провоцирует на чересчур серьезные ошибки.

В подавляющем большинстве случаев для разработчика цифровых схем достаточно трех моделей, трех уровней представления о работе цифровых устройств:

1. Логическая модель.
2. Модель с временными задержками.
3. Модель с учетом электрических эффектов (или электрическая модель).

Опыт показывает, что первой, простейшей модели достаточно примерно в 20 % всех случаев. Она применима для всех цифровых схем, работающих с низкой скоростью, в которых быстродействие не принципиально. Привлечение второй модели, учитывающей задержки срабатывания логических элементов, позволяет охватить около 80 % всех возможных схем. Ее применение необходимо для всех быстродействующих устройств и для случая одновременного изменения нескольких входных сигналов. Наконец, добавление третьей модели, учитывающей входные и выходные токи, входные и выходные сопротивления и емкости элементов, дает возможность проектирования практически 100 % цифровых схем. В первую очередь, эту третью модель надо применять при объединении нескольких входов и выходов, при передаче сигналов на большие расстояния и при нетрадиционном включении логических элементов (с переводом их в аналоговый или в линейный режимы).

Для иллюстрации работы перечисленных моделей рассмотрим работу самого простейшего логического элемента — инвертора. Инвертор изменяет (инвертирует) логический уровень входного сигнала на противоположный уровень выходного сигнала или, как еще говорят, изменяет полярность логического сигнала. Его таблица истинности (табл. 1.1) элементарно проста, так как возможно только две ситуации: нуль на входе или единица на входе. На рис. 1.4 показано, как будет выглядеть выходной сигнал инвертора при использовании трех его моделей

(трех уровней его представления). Такие графики логических сигналов называются временными диаграммами, они позволяют лучше понять работу цифровых схем.

Из рисунка видно, что в первой, логической модели считается, что элемент срабатывает мгновенно, любое изменение уровня входного сигнала сразу же, без всякой задержки приводит к изменению уровня выходного сигнала. Во второй модели выходной сигнал изменяется с некоторой задержкой относительно входного. Наконец, в третьей модели выходной сигнал не только задерживается по сравнению с входным, но и его изменение происходит не мгновенно — процесс смены уровней сигнала (или, как говорят, *фронт сигнала)* имеет конечную длительность. Кроме того, третья модель учитывает изменение уровней логических сигналов.

Таблица 1.1. Таблица истинности инвертора

Вход	Выход
0	1
1	0

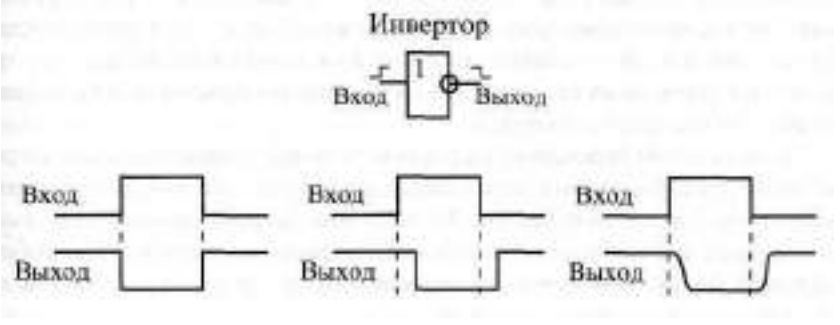


Рис. 1.4. Три уровня представления цифровых устройств

На практике разработчик, как правило, в начале проектирования пользуется исключительно первой моделью, а затем для некоторых узлов применяет вторую или (реже) еще и третью модель. При этом первая модель не требует вообще никаких цифровых расчетов, для нее достаточно только знание таблиц истинности или алгоритмов функционирования микросхем. Вторая модель предполагает расчет (по сути, суммирование) временных задержек элементов на пути прохождения сигналов (рис. 1.5). В результате этого расчета может выясниться, что требуется внесение изменений в схему.

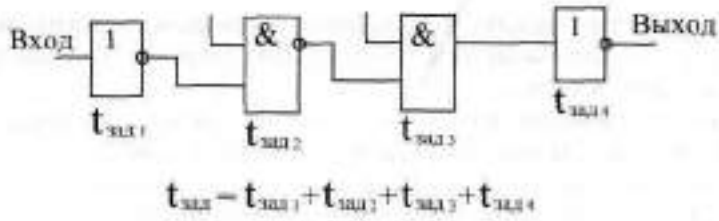


Рис. 1.5. Суммирование задержек элементов

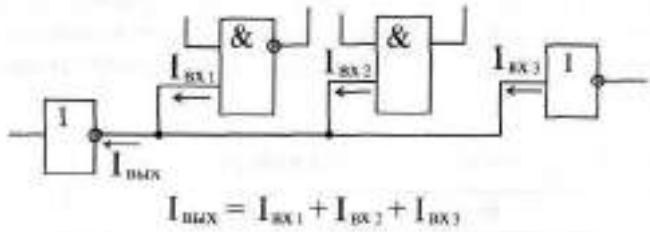


Рис. 1.6. Суммирование входных токов элементов

Расчеты по третьей модели могут быть различными, в том числе и довольно сложными, но в большинстве случаев они все-таки сводятся всего лишь к суммированию входных и выходных токов логических элементов (рис. 1.6). В результате этих расчетов может выясниться, что требуется применение микросхем с более мощными выходами или включение дополнительных элементов.

То есть проектирование цифровых устройств принципиально отличается от проектирования аналоговых устройств, при котором сложные расчеты абсолютно неизбежны. Разработчик цифровых устройств имеет дело только с логикой, с логическими сигналами и с алгоритмами работы цифровых микросхем. А что происходит внутри этих микросхем, для него практически не имеет значения.

Справочные данные на цифровые микросхемы обычно содержат большой набор параметров, каждый из которых можно отнести к одному из трех перечисленных уровней представления, к одной из трех моделей.

Например, таблица истинности микросхемы (для простых микросхем) или описание алгоритма ее работы (для более сложных микросхем) относится к первому, логическому уровню. Поэтому знать их наизусть каждому разработчику необходимо в любом случае.

Величины задержек логических сигналов между входами и выходами относятся ко второму уровню представления. Типичные величины задержек составляют от единиц наносекунд (1 нс = 10^{-9} с) до десятков наносекунд. Величины задержек для разных микросхем могут быть различ-

ными, поэтому в справочниках всегда указывается максимальное значение. Необходимо также помнить, что задержка при переходе выходного сигнала из единицы в нуль (t_{PHL}), как правило, отличается от задержки при переходе выходного сигнала из нуля в единицу (t_{PLH}). Например, для одной и той же микросхемы $t_{PLH} < 11$ нс, а $t_{PHL} < 8$ нс. Здесь английская буква P означает Propagation (распространение), L означает Low (низкий уровень сигнала, нуль), а H — High (высокий уровень сигнала, единица). Количество величин задержек, определяемых справочником для микросхемы, может изменяться от двух до нескольких десятков.

Уровни входных и выходных токов, а также уровни входных и выходных напряжений относятся к третьему уровню представления.

Входной ток микросхемы при приходе на вход логического нуля (I_{IL}), как правило, отличается от входного тока при приходе на вход логической единицы (I_{IH}). Например, $I_{IL} = -0{,}1$ мА, а $I_{IH} = 20$ мкА (считается, что положительный ток втекает во вход микросхемы, а отрицательный — вытекает из него). Точно так же выходной ток микросхемы при выдаче логического нуля (I_{OL}) может отличаться (и обычно отличается) от выходного тока при выдаче логической единицы (I_{OH}). Например, для одной и той же микросхемы $I_{OH} < -0{,}4$ мА, а $I_{OL} < 8$ мА (считается, что положительный ток втекает в выход микросхемы, а отрицательный — вытекает из него). Надо также учитывать, что разные входы и выходы одной и той же микросхемы могут иметь различные входные и выходные токи.

Для выходных напряжений логического нуля (U_{OL}) И единицы (U_{OH}) в справочниках обычно задаются предельно допустимые значения при данной величине выходного тока. В этом случае, чем больше выходной ток, тем меньше напряжение логической единицы и тем больше напряжение логического нуля. Например, $U_{OH} > 2{,}5$ В (при $I_{OH} < -0{,}4$ мА), а $U_{OL} < 0{,}5$ В (при $I_{OL} < 8$ мА).

Задаются в справочниках также и допустимые уровни входных напряжений, которые микросхема еще воспринимает как правильные логические уровни нуля и единицы. Например, $U_{IH} > 2{,}0$ В, $U_{IL} < 0{,}8$ В. Как правило, входные напряжения логических сигналов не должны выходить за пределы напряжения питания.

В обозначениях напряжений и токов буква I означает Input (вход), буква 0 означает Output (выход), L — Low (нуль), а H — High (единица).

К третьему уровню представления относятся также величины внутренней емкости входов микросхемы (обычно от единиц до десятков пикофарад) и допустимая величина емкости, к которой может подключаться выход микросхемы, то есть емкость нагрузки C_L (порядка 100 пФ). Отметим, что 1 пФ = 10^{-12} Ф. На этом же уровне представления задаются максимально допустимые величины длительности положительного фронта (t_{LH}) и отрицательного фронта (t_{HL}) входного сигнала, например, $t_{HL} <$

1,0 мкс, t_{LH} < 1,0 мкс. То есть при большей длительности перехода входного сигнала из единицы в нуль и из нуля в единицу микросхема может работать неустойчиво, неправильно, нестандартно.

К третьему уровню представления можно отнести также такие параметры, как допустимое напряжение питания микросхемы (U_{CC}) И максимальный ток, потребляемый микросхемой *(IQQ).* Например, может быть задано

$$4,5\ В < U_{CC} < 5,5\ В;\ I_{CC} < 100\ мА.$$

При этом потребляемый ток I_{CC} зависит от уровней выходных токов микросхемы I_{OH} И I_{OL}. ЭТИ параметры надо учитывать при выборе источника питания для проектируемого устройства, а также в процессе изготовления печатных плат — при выборе ширины токоведущих дорожек.

Наконец, к третьему же уровню относится ряд параметров, которые часто упоминаются в литературе, но не всегда приводятся в справочных таблицах:

- Порог срабатывания — уровень входного напряжения, выше которого сигнал воспринимается как единица, а ниже — как нуль. Для наиболее распространенных ТТЛ микросхем он примерно равен 1,3...1,4 В.
- Помехозащищенность — характеризует величину входного сигнала помехи, накладывающегося на входной сигнал, который еще не может изменить состояние выходных сигналов. Помехозащищенность определяется разницей между напряжением U_{IH} и порогом срабатывания (это помехозащищенность единичного уровня), а также разницей между порогом срабатывания и U_{IL} (это помехозащищенность нулевого уровня).
- Коэффициент разветвления — число входов, которое может быть подключено к данному выходу без нарушения работы. Определяется отношением выходного тока к входному. Стандартная величина коэффициента разветвления при использовании микросхем одного типа (одной серии) равна 10.
- Нагрузочная способность — параметр выхода, характеризующий величину выходного тока, которую может выдать в нагрузку данный выход без нарушения работы. Чаще всего нагрузочная способность прямо связана с коэффициентом разветвления.

Таким образом, большинство справочных параметров микросхемы относятся к третьему уровню представления (к модели с учетом электрических эффектов), поэтому в большинстве случаев (до 80 %) знать их точные значения наизусть не обязательно. Достаточно помнить примерные типовые значения параметров для данной серии микросхем.

1.3. Входы и выходы цифровых микросхем

Характеристики и параметры входов и выходов цифровых микросхем определяются прежде всего технологией и схемотехникой их внутреннего строения. Но для разработчика цифровых устройств любая микросхема представляет собой всего лишь «черный ящик», внутренности которого знать не обязательно. Ему важно только четко представлять себе, как поведет себя та или иная микросхема в данном конкретном включении, будет ли она правильно выполнять требуемую от нее функцию.

Наибольшее распространение получили две технологии цифровых микросхем:

- ТТЛ (TTL) и ТТЛШ (TTLS) — биполярная транзисторно-транзисторная логика и ТТЛ с диодами Шоттки;
- КМОП (CMOS) — комплементарные транзисторы со структурой «металл—окисел—полупроводник».

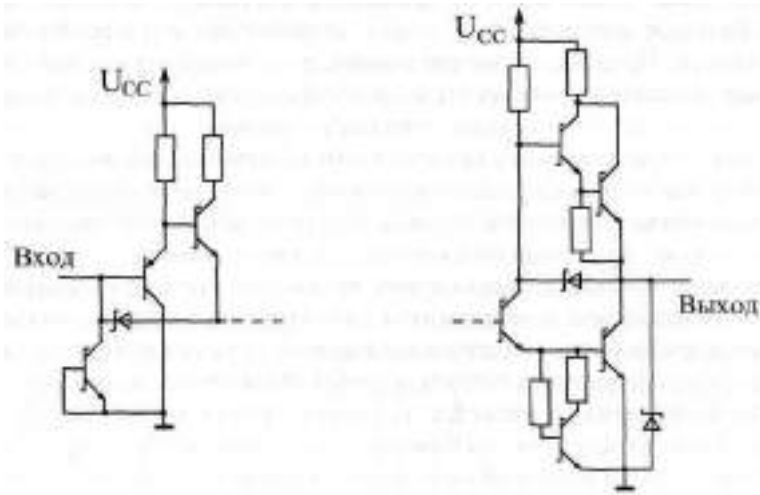


Рис. 1.7. Входной и выходной каскады микросхем ТТЛШ

Различаются они типами используемых транзисторов и схемотехническими решениями внутренних каскадов микросхем. Отметим также, что микросхемы КМОП потребляют значительно меньший ток от источника питания, чем такие же микросхемы ТТЛ (или ТТЛШ) — правда, только в статическом режиме или на небольших рабочих частотах. На рис. 1.7 и 1.8 показаны примеры схем входных и выходных каскадов микросхем, выполненных по этим технологиям. Понятно, что точный учет всех эффектов в этих схемах, включающих в себя множество транзи-

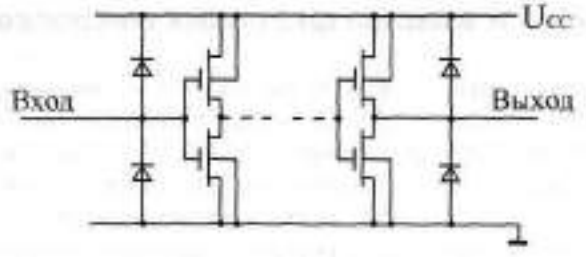


Рис. 1.8. Входной и выходной каскады микросхем КМОП

сторов, диодов и резисторов, крайне сложен, но обычно он просто не нужен разработчику цифровых схем.

Рассмотрим сначала **входы** микросхем.

На первом уровне представления (логическая модель) и на втором уровне представления (модель с временными задержками) о входах микросхем вообще ничего знать не нужно. Вход рассматривается как бесконечно большое сопротивление, никак не влияющее на подключенные к нему выходы. Правда, количество входов, подключенных к одному выходу, влияет на задержку распространения сигнала, но, как правило, незначительно, поэтому это влияние учитывается редко.

Даже на третьем уровне представления (электрическая модель) в большинстве случаев не нужно знать о внутреннем строении микросхемы, о схемотехнике входов. Достаточно считать, что при подаче на вход сигнала логического нуля из этого входа вытекает ток, не превышающий I_{IL}, а при подаче сигнала логической единицы в этот вход втекает ток, не превышающий I_{IH}. А для правильной логики работы микросхемы достаточно, чтобы уровень напряжения входного сигнала логического нуля был меньше U_{IL}, а уровень напряжения входного сигнала логической единицы был больше U_{IH}.

Особым случаем является ситуация, когда какой-нибудь вход не подключен ни к одному из выходов — ни к общему проводу, ни к шине питания (так называемый *висящий вход).* Иногда возможности микросхемы используются не полностью и на некоторые входы не подается сигналов. Однако при этом микросхема может не работать или работать нестабильно, так как ее правильное включение подразумевает наличие на всех входах логических уровней, пусть даже и неизменных. Поэтому рекомендуется подключать неиспользуемые входы к напряжению питания микросхемы U_{CC} ИЛИ К общему проводу (к земле) в зависимости от того, какой логический уровень необходим на этом входе. Но для некоторых серий микросхем, выполненных по технологии **ТТЛ** (например, К155 или КР531), неиспользуемые входы надо подключать к напряжению питания не напрямую, а только через резистор величиной около 1 кОм (достаточно одного резистора на 20 входов).

На неподключенных входах микросхем ТТЛ формируется напряжение около 1,5–1,6 В, которое иногда называют висячим потенциалом. Обычно этот уровень воспринимается микросхемой как сигнал логической единицы, но рассчитывать на это не стоит. Потенциал, образующийся на неподключенных входах микросхем КМОП, может восприниматься микросхемой и как логический нуль, и как логическая единица. В любом случае все входы надо куда-то подключать. Неподключенными допускается оставлять только те входы (ТТЛ, а не КМОП), состояние которых в данном включении микросхемы не имеет значения.

Выходы микросхем принципиально отличаются от входов тем, что учет их особенностей необходим даже на первом и втором уровнях представления.

Существуют три разновидности выходных каскадов, существенно различающиеся как по своим характеристикам, так и по областям применения:

- стандартный выход или выход с двумя состояниями (обозначается 2С, 2S или, реже, ТТЛ, TTL);
- выход с открытым коллектором (обозначается ОК, OC);
- выход с тремя состояниями или (что то же самое) с возможностью отключения (обозначается ЗС, 3S).

Стандартный выход 2С имеет всего два состояния: логический нуль и логическая единица, причем оба они активны, то есть выходные токи в обоих этих состояниях (I_{OL} И I_{OH}) могут достигать заметных величин. На первом и втором уровнях представления такой выход можно считать состоящим из двух выключателей, которые замыкаются по очереди (рис. 1.9), причем замкнутому верхнему выключателю соответствует логическая единица на выходе, а замкнутому нижнему — логический нуль.

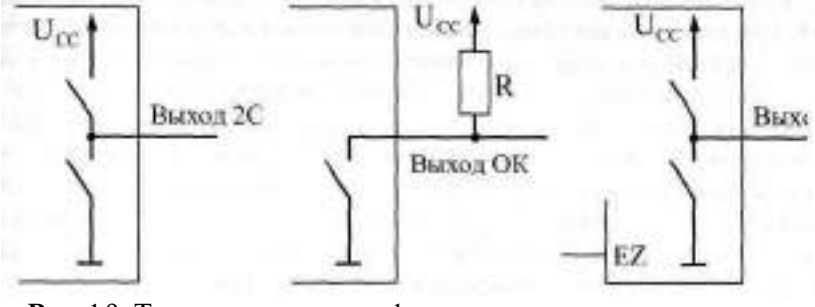


Рис. 1.9. Три типа выходов цифровых микросхем

Выход с открытым коллектором ОК тоже имеет два возможных состояния, но только одно из них (состояние логического нуля) активно, то есть обеспечивает большой втекающий ток I_{OL}. Второе состояние сво-

дится, по сути, к тому, что выход полностью отключается от присоединенных к нему входов. Это состояние может использоваться в качестве логической единицы, но для этого между выходом ОК и напряжением питания необходимо подключить нагрузочный резистор R (так называемый pull-up) величиной порядка сотен Ом. На первом и втором уровнях представления такой выход можно считать состоящим из одного выключателя (рис. 1.9), замкнутому состоянию которого соответствует сигнал логического нуля, а разомкнутому — отключенное, пассивное состояние. Правда, от величины резистора R зависит время переключения выхода из нуля в единицу, что влияет на задержку t_{LH}, но при обычно используемых номиналах резисторов это не слишком важно.

Наконец, выход с тремя состояниями ЗС очень похож на стандартный выход, но к двум состояниям добавляется еще и третье — пассивное, в котором выход можно считать отключенным от последующей схемы. На первом и втором уровнях представления такой выход можно считать состоящим из двух переключателей (рис. 1.9), которые могут замыкаться по очереди, давая логический нуль и логическую единицу, но могут и размыкаться одновременно. Это третье состояние называется также *высокоимпедансным* или *Z-состоянием*. Для перевода выхода в третье Z-состояние используется специальный управляющий вход, обозначаемый OE (Output Enable — разрешение выхода) или EZ (Enable Z-state).

Почему же помимо стандартного выхода (2С) были предложены еще два типа выходов (ОК и ЗС)? Дело в том, что выходы, имеющие помимо активных еще и пассивное состояние, очень удобны для объединения их между собой. Например, если на один и тот же вход надо по очереди подавать сигналы с двух выходов (рис. 1.10), то выходы 2С для этого не подходят, а вот выходы ОК и ЗС — подходят.

При объединении двух или более выходов 2С вполне возможна ситуация, при которой один выход стремится выдать сигнал логической единицы, а другой — сигнал логического нуля. Легко заметить, что в этом случае через верхний замкнутый ключ выхода, выдающего единицу, и через нижний замкнутый ключ выхода, выдающего нуль, пойдет недопустимо большой ток короткого замыкания $I_{кз}$. Это аварийная ситуация, при которой уровень получаемого выходного логического сигнала точно не определен — он может восприниматься последующим входом и как нуль, и как единица. Конфликтующие выходы могут даже выйти из строя, нарушив работу микросхем и схемы в целом.

Зато в случае объединения двух выходов ОК такого конфликта в принципе произойти не может. Даже если ключ одного выхода замкнут, а другого — разомкнут, аварийной ситуации не произойдет, так как недопустимо большого тока не будет, а на объединенном выходе будет сигнал логического нуля. А при объединении двух выходов ЗС аварийная ситуа-

ция хоть и возможна (если оба выхода одновременно находятся в активном состоянии), но ее легко можно предотвратить, если организовать схему так, что в активном состоянии всегда будет находиться только один из объединенных выходов 3С.

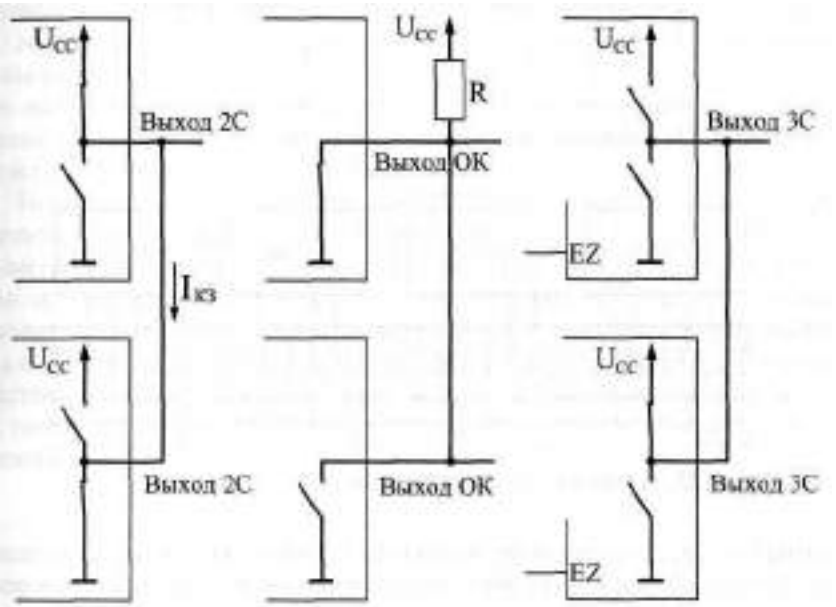


Рис. **1.10.** Объединение выходов цифровых микросхем

Объединение выходов цифровых микросхем совершенно необходимо также при шинной (или, как еще говорят, магистральной) организации связей между цифровыми устройствами. Шинная организация связей применяется, например, в компьютерах и в других микропроцессорных системах. Суть ее сводится к следующему.

При классической организации связей (рис. 1.11) все сигналы между устройствами передаются по своим отдельным линиям (проводам). Каждое устройство передает свои сигналы всем другим независимо от других устройств. В этом случае обычно получается очень много линий связи, к тому же правила обмена сигналами по этим линиям (или протоколы обмена) чрезвычайно разнообразны.

При шинной же организации связей (рис. 1.12) все сигналы между устройствами передаются по одним и тем же линиям (проводам), но в разные моменты времени (это называется временным мультиплексированием). В результате количество линий связи резко сокращается, а правила обмена сигналами существенно упрощаются. Группа линий

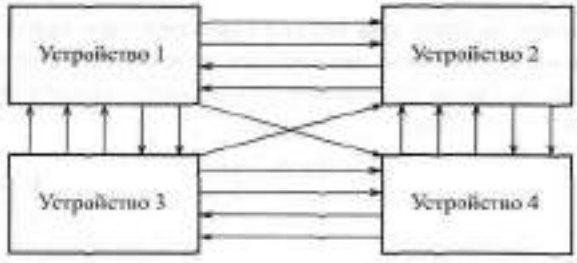


Рис. 1.11. Классическая организация связей

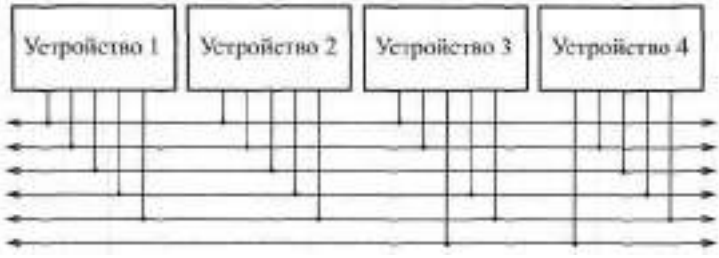


Рис. 1.12. Шинная организация связей

(сигналов), используемая несколькими устройствами, как раз и называется шиной. Понятно, что объединение выходов в этом случае совершенно необходимо — ведь каждое устройство должно иметь возможность выдавать свой сигнал на общую линию. К недостаткам шинной организации относится прежде всего невысокая (по сравнению с классической структурой связей) скорость обмена сигналами. При простых структурах связи она может быть избыточна.

Но вернемся к типам выходов цифровых микросхем.

На третьем уровне представления (электрическая модель) необходимо уже учитывать, что выходные ключи (рис. 1.9) представляют собой не простые тумблеры (как на первых двух уровнях представления), а транзисторные ключи со своими специфическими параметрами. Однако в большинстве случаев достаточно знать, какой ток может выдать данный выход при логическом нуле (I_{OL}) и при логической единице (I_{OH}). Величины этих токов не должны превышать суммы токов всех входов, подключенных к данному выходу (соответственно I_{IL} и I_{IH}). Количеством входов, которое можно подключить к одному выходу, определяется коэффициент разветвления или нагрузочную способность микросхемы. Существуют микросхемы с обычной нагрузочной способностью и с повышенной нагрузочной способностью (больше обычной в два раза и более). Выходы ЗС, как правило, имеют повышенную нагрузочную способ-

ность (то есть обеспечивают большие выходные токи). Выходы 2С и ОК могут быть как с обычной, так и с повышенной нагрузочной способностью.

Также на третьем уровне представления (электрическая модель) необходимо учитывать выдаваемые выходом микросхемы величины выходных напряжений U_{OL} и U_{OH}. выходы ОК могут быть рассчитаны как на обычное выходное напряжение логической единицы ($U_{OH} = U_{CC} = 5$ В), так и на повышенное напряжение логической единицы (до 30 В). В последнем случае внешний резистор этого выхода (см. рис. 1.9) подключается к источнику повышенного напряжения.

Только в сложных случаях, например, при переводе логического элемента в линейный режим за счет обратных связей, нужен учет других параметров входных и выходных каскадов. Но в этих редких случаях гораздо проще и надежнее не считать ничего самому, а воспользоваться стандартными схемами включения микросхем или подобрать режимы работы и номиналы внешних элементов (резисторов, конденсаторов) непосредственно на макете проектируемого устройства. В отличие от расчетов, такой подход даст полную гарантию работоспособности выбранного решения.

Лекция 2. Микросхемы и их функционирование

В лекции рассматриваются обозначения цифровых микросхем, их выводов и сигналов на принципиальных схемах, особенности основных серий простейших цифровых микросхем, базовые типы корпусов микросхем, а также принципы двоичного кодирования и принципы работы цифровых устройств.

Ключевые слова: схемы — принципиальная, структурная и функциональная, полярность сигнала, фронт сигнала, шина, серии микросхем, корпуса микросхем, двоичный код, двоичный разряд, байт, тетрада, слово, двоичные операции, коды, жесткая логика, программируемые устройства.

1.4. Основные обозначения на схемах

Для изображения электронных устройств и их узлов применяется три основных типа схем:

- принципиальная схема;
- структурная схема;
- функциональная схема.

Различаются они своим назначением и, самое главное, степенью детализации изображения устройств.

Принципиальная схема — наиболее подробная. Она обязательно показывает все использованные в устройстве элементы и все связи между ними. Если схема строится на основе микросхем, то должны быть показаны номера выводов всех входов и выходов этих микросхем. Принципиальная схема должна позволять полностью воспроизвести устройство. Обозначения принципиальной схемы наиболее жестко стандартизованы, отклонения от стандартов не рекомендуются.

Структурная схема — наименее подробная. Она предназначена для отображения общей структуры устройства, то есть его основных блоков, узлов, частей и главных связей между ними. Из структурной схемы должно быть понятно, зачем нужно данное устройство и что оно делает в основных режимах работы, как взаимодействуют его части. Обозначения структурной схемы могут быть довольно произвольными, хотя некоторые общепринятые правила все-таки лучше выполнять.

Функциональная схема представляет собой гибрид структурной и принципиальной. Некоторые наиболее простые блоки, узлы, части устройства отображаются на ней, как на структурной схеме, а остальные - как на принципиальной схеме. Функциональная схема дает возможность понять всю логику работы устройства, все его отличия от других подоб-

ных устройств, но не позволяет без дополнительной самостоятельной работы воспроизвести это устройство. Что касается обозначений, используемых на функциональных схемах, то в части, показанной как структура, они не стандартизованы, а в части, показанной как принципиальная схема, — стандартизованы.

В технической документации обязательно приводятся структурная или функциональная схема, а также обязательно принципиальная схема. В научных статьях и книгах чаще всего ограничиваются структурной или функциональной схемой, приводя принципиальные схемы только некоторых узлов.

А теперь рассмотрим основные обозначения, используемые на схемах.

Все узлы, блоки, части, элементы, микросхемы показываются в виде прямоугольников с соответствующими надписями. Все связи между ними, все передаваемые сигналы изображаются в виде линий, соединяющих эти прямоугольники. Входы и входы/выходы должны быть расположены на левой стороне прямоугольника, выходы — на правой стороне, хотя это правило часто нарушают, когда необходимо упростить рисунок схемы. Выводы и связи питания, как правило, не прорисовывают, если, конечно, не используются нестандартные включения элементов схемы. Это самые общие правила, касающиеся любых схем.

Прежде чем перейти к более частным правилам, дадим несколько определений.

Положительный сигнал (сигнал положительной полярности) - это сигнал, активный уровень которого — логическая единица. То есть нуль - это отсутствие сигнала, единица — сигнал пришел (рис. 1.13).

Рис. 1.13. Элементы цифрового сигнала

Отрицательный сигнал (сигнал отрицательной полярности) — это сигнал, активный уровень которого — логический нуль. То есть единица — это отсутствие сигнала, нуль — сигнал пришел (рис. **1.13).**

Активный уровень сигнала — это уровень, соответствующий приходу сигнала, то есть выполнению этим сигналом соответствующей ему функции.

Пассивный уровень сигнала — это уровень, в котором сигнал не выполняет никакой функции.

Инвертирование или инверсия сигнала — это изменение его полярности.

Инверсный выход — это выход, выдающий сигнал инверсной полярности по сравнению с входным сигналом.

Прямой выход — это выход, выдающий сигнал такой же полярности, какую имеет входной сигнал.

Положительный фронт сигнала - это переход сигнала из нуля в единицу.

Отрицательный фронт сигнала (спад) — это переход сигнала из единицы в нуль.

Передний фронт сигнала — это переход сигнала из пассивного уровня в активный.

Задний фронт сигнала — это переход сигнала из активного уровня в пассивный.

Тактовый сигнал (или строб) — управляющий сигнал, который определяет момент выполнения элементом или узлом его функции.

Шина — группа сигналов, объединенных по какому-то принципу, например, шиной называют сигналы, соответствующие всем разрядам какого-то двоичного кода.

Рис. 1.14. Обозначение входов и выходов

Для обозначения полярности сигнала на схемах используется простое правило: если сигнал отрицательный, то перед его названием ставится знак минус, например, WR или OE, или же (реже) над названием сигнала ставится черта. Если таких знаков нет, то сигнал считается положительным. Для названий сигналов обычно используются латинские буквы,

представляющие собой сокращения английских слов, например, WR — сигнал записи (от «write» — «писать»).

Инверсия сигнала обозначается кружочком на месте входа или выхода. Существуют инверсные входы и инверсные выходы (рис. 1.14).

Если какая-то микросхема выполняет функцию по фронту входного сигнала, то на месте входа ставится косая черта (под углом 45°, причем наклон вправо или влево определяется тем, положительный или отрицательный фронт используется в данном случае (рис. 1.14).

Тип выхода микросхемы помечается специальным значком: выход ЗС — перечеркнутым ромбом, а выход ОК — подчеркнутым ромбом (рис. 1.14). Стандартный выход (2С) никак не помечается.

Наконец, если у микросхемы необходимо показать неинформационные выводы, то есть выводы, не являющиеся ни логическими входами, ни логическими выходами, то такой вывод помечается косым крестом (две перпендикулярные линии под углом 45°. Это могут быть, например, выводы для подключения внешних элементов (резисторов, конденсаторов) или выводы питания (рис. 1.15).

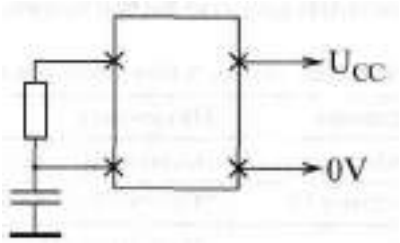


Рис. 1.15. Обозначение неинформационных выводов

В схемах также предусматриваются специальные обозначения для шин (рис. 1.16). На структурных и функциональных схемах шины обозначаются толстыми линиями или двойными стрелками, причем количество сигналов, входящих в шину, указывается рядом с косой чертой, пересекающей шину. На принципиальных схемах шина тоже обозначается толстой линией, а входящие в шину и выходящие из шины сигналы изображаются в виде перпендикулярных к шине тонких линий с указанием их номера или названия (рис. 1.16). При передаче по шине двоичного кода нумерация начинается с младшего разряда кода.

При изображении микросхем используются сокращенные названия входных и выходных сигналов, отражающие их функцию. Эти названия располагаются на рисунке рядом с соответствующим выводом. Также на изображении микросхем указывается выполняемая ими функция (обычно в центре вверху). Изображение микросхемы иногда делят на три вертикальные поля. Левое поле относится к входным сигналам, правое —

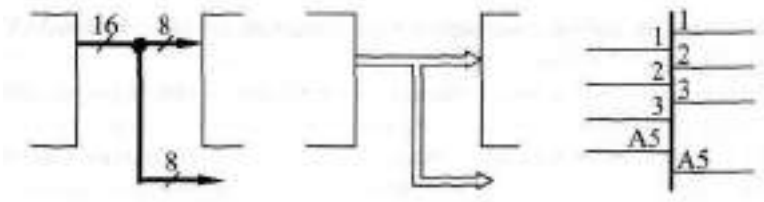


Рис. 1.16. Обозначение шин

к выходным сигналам. В центральном поле помещается название микросхемы и символы ее особенностей. Неинформационные выводы могут указываться как на левом, так и на правом поле; иногда их показывают на верхней или нижней стороне прямоугольника, изображающего микросхему.

В табл. 1.2 приведены некоторые наиболее часто встречающиеся обозначения сигналов и функций микросхем. Микросхема в целом обозначается на схемах буквами DD (от английского «digital» — «цифровой») с соответствующим номером, например, DDl, DD20.1, DD38.2 (после точки указывается номер элемента или узла внутри микросхемы).

Таблица 1.2. Некоторые обозначения сигналов и микросхем

Обозначение	**Название**	**Назначение**
&	And	Элемент И
=1	Exclusive Or	Элемент Исключающее ИЛИ
1	Or	Элемент ИЛИ
A	Address	Адресные разряды
BF	Buffer	Буфер
C	Clock	Тактовый сигнал (строб)
CE	Clock Enable	Разрешение тактового сигнала
CT	Counter	Счетчик
CS	Chip Select	Выбор микросхемы
D	Data	Разряды данных, данные
DC	Decoder	Дешифратор
EZ	Enable Z-state	Разрешение третьего состояния
G	Generator	Генератор
I	Input	Вход
I/O	Input/Output	Вход/Выход

OE	Output Enable	Разрешение выхода
MS	Multiplexer	Мультиплексор
Q	Quit	Выход
R	Reset	Сброс (установка в нуль)
RG	Register	Регистр
S	Set	Установка в единицу
SUM	Summator	Сумматор
T	Trigger	Триггер
TC	Terminal Count	Окончание счета
Z	Z-state	Третье состояние выхода

Более полная таблица обозначений сигналов и микросхем, используемых в принципиальных схемах, приведена в приложении.

1.5. Серии цифровых микросхем

В настоящее время выпускается огромное количество разнообразных цифровых микросхем: от простейших логических элементов до сложнейших процессоров, микроконтроллеров и специализированных БИС (Больших Интегральных Микросхем). Производством цифровых микросхем занимается множество фирм — как у нас в стране, так и за рубежом. Поэтому даже классификация этих микросхем представляет собой довольно трудную задачу.

Однако в качестве базиса в цифровой схемотехнике принято рассматривать классический набор микросхем малой и средней степени интеграции, в основе которого лежат ТТЛ серии семейства 74, выпускаемые уже несколько десятилетий рядом фирм, например, американской фирмой Texas Instruments (TI). Эти серии включают в себя функционально полный комплект микросхем, используя который, можно создавать самые разные цифровые устройства. Даже при компьютерном проектировании современных сложных микросхем с программируемой логикой (ПЛИС) применяются модели простейших микросхем этих серий семейства 74. При этом разработчик рисует на экране компьютера схему в привычном для него элементном базисе, а затем программа создает прошивку ПЛИС, выполняющую требуемую функцию.

Каждая микросхема серий семейства 74 имеет свое обозначение, и система обозначений отечественных серий существенно отличается от принятой за рубежом.

Рис. 1.17. Система обозначений фирмы Texas Instruments

В качестве примера рассмотрим систему обозначений фирмы Texas Instruments (рис. 1.17). Полное обозначение состоит из шести элементов:

1. Идентификатор фирмы SN (для серий AC и ACT отсутствует).
2. Температурный диапазон (тип семейства):
   - 74 — коммерческие микросхемы (температура окружающей среды для биполярных микросхем — 0...70°С, для КМОП микросхем — —40...+85°С),
   - 54 - микросхемы военного назначения (температура —55...+125°С).
3. Код серии (до трех символов):
   - Отсутствует — стандартная ТТЛ-серия.
   - LS (Low Power Schottky) - маломощная серия ТТЛШ.
   - S (Schottky) - серия ТТЛШ.
   - ALS (Advanced Schottky) — улучшенная серия ТТЛШ.
   - F (FAST) — быстрая серия.
   - HC (High Speed CMOS) — высокоскоростная КМОП-серия.
   - HCT (High Speed CMOS with TTL inputs) - серия HC, совместимая по входу с ТТЛ.
   - AC (Advanced CMOS) — улучшенная серия КМОП.
   - ACT (Advanced CMOS with TTL inputs) — серия AC, совместимая по входу с ТТЛ.
   - BCT (BiCMOS Technology) - серия с БиКМОП-технологией.
   - ABT (Advanced BiCMOS Technology) — улучшенная серия с БиКМОП-технологией.
   - LVT (Low Voltage Technology) — серия с низким напряжением питания.
4. Идентификатор специального типа (2 символа) — может отсутствовать.
5. Тип микросхемы (от двух до шести цифр). Перечень некоторых типов микросхем приведен в приложении.
6. Код типа корпуса (от одного до двух символов) — может отсутствовать. Например, N — пластмассовый корпус DIL (DIP), J — керамический DIL (DIC), T — плоский металлический.

Примеры обозначений: SN74ALS373, SN74ACT7801, SN7400.

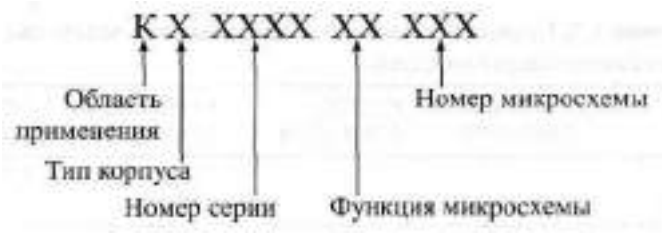


Рис. 1.18. Обозначения отечественных микросхем

Отечественная система обозначений микросхем отличается от рассмотренной довольно существенно (рис. 1.18). Основные элементы обозначения следующие:

1. Буква К обозначает микросхемы широкого применения, для микросхем военного назначения буква отсутствует.
2. Тип корпуса микросхемы (один символ) — может отсутствовать. Например, Р — пластмассовый корпус, [illegible] — керамический, Б — бескорпусная микросхема.
3. Номер серии микросхем (от трех до четырех цифр).
4. Функция микросхемы (две буквы).
5. Номер микросхемы (от одной до трех цифр). Таблица функций и номеров микросхем, а также таблица их соответствия зарубежным аналогам приведены в приложении.

Например, КР1533ЛА3, КМ531ИЕ17, КР1554ИР47.

Главное достоинство отечественной системы обозначений состоит в том, что по обозначению микросхемы можно легко понять ее функцию. Зато в системе обозначений Texas Instruments виден тип серии с его особенностями.

Чем отличается одна серия от другой?

На первом уровне представления (логическая модель) серии не различаются ничем. То есть одинаковые микросхемы разных серий работают по одним и тем же таблицам истинности, по одним и тем же алгоритмам. Правда, надо учитывать, что некоторые микросхемы имеются только в одной из серий, а некоторых нет в нескольких сериях.

На втором уровне представления (модель с учетом задержек) серии отличаются величиной задержки распространения сигнала. Это различие может быть довольно существенным. Поэтому в тех схемах, где величина задержки принципиальна, надо использовать микросхемы более быстрых серий (табл. 1.3).

На третьем уровне представления (электрическая модель) серии различаются величинами входных и выходных токов и напряжений, а также,

Таблица 1.3. Сравнение параметров одинаковых микросхем в разных стандартных сериях

	К155ЛА3 (SN7400N)	К555ЛА3 (SN74LS00N)	КР1533ЛА3 (SN74ALS00N)	КР1554ЛА3 (SN74AC00N)
t_{PLH}, нс не более	22	15	11	8,5
t_{PHL}, нс не более	15	15	8	7,0
I_{IL}, мА не более	-1,6	-0,4	-0,1	-0,001
I_{IH}, мА не более	0,04	0,02	0,02	0,001
I_{OL}, мА не менее	16	8	15	86
I_{OH}, мА не менее	-0,4	-0,4	-0,4	-75
U_{OL}, В не более	0,4	0,5	0,5	0,3
U_{OH}, В не менее	2,4	2,7	2,5	4,4
I_{CC}, мА не более	12	4,4	3	0,04

что не менее важно, токами потребления (табл. 1.3). Поэтому в тех устройствах, где эти параметры принципиальны, надо применять микросхемы, обеспечивающие, к примеру, низкие входные токи, высокие выходные токи и малое потребление.

Серия К155 (SN74) — это наиболее старая серия, которая постепенно снимется с производства. Она отличается не слишком хорошими параметрами по сравнению с другими сериями. С этой классической серией принято сравнивать все остальные.

Серия К555 (SN74LS) отличается от серии К155 малыми входными токами и меньшей потребляемой мощностью (ток потребления — почти втрое меньше, чем у К155). По быстродействию (по временам задержек) она близка к К155.

Серия КР531 (SN74S) отличается высоким быстродействием (ее задержки примерно в 3—4 раза меньше, чем у серии К155), но большими входными токами (на 25 % больше, чем у К155) и большой потребляемой мощностью (ток потребления — больше в полтора раза по сравнению с К155).

Серия КР1533 (SN74ALS) отличается повышенным примерно вдвое по сравнению с К155 быстродействием и малой потребляемой мощностью (в четыре раза меньше, чем у К155). Входные токи еще меньше, чем у К555.

Серия КР1531 (SN74F) отличается высоким быстродействием (на уровне КР531), но малой потребляемой мощностью. Входные токи и ток потребления примерно вдвое меньше, чем у К155.

Серия КР1554 (SN74AC) отличается от всех предыдущих тем, что она выполнена по КМОП-технологии. Поэтому она имеет сверхмалые входные токи и сверхмалое потребление при малых рабочих частотах. Задержки примерно вдвое меньше, чем у К155.

Наибольшим разнообразием имеющихся микросхем отличаются серии К155 и КР1533, наименьшим - КР1531 и КР1554.

Надо отметить, что приведенные здесь соотношения по быстродействию стандартных серий довольно приблизительны и верны не для всех разновидностей микросхем, имеющихся в разных сериях. Точные значения задержек необходимо смотреть в справочниках, причем желательно в фирменных справочных материалах.

Микросхемы разных серий обычно легко сопрягаются между собой, то есть сигналы с выходов микросхем одной серии можно смело подавать на входы микросхем другой серии. Одно из исключений — соединение выходов ТТЛ-микросхем со входами КМОП-микросхем серии КР1554 (74AC). При таком соединении необходимо применение резистора номиналом 560 Ом между сигналом и напряжением питания (рис. 1.19).

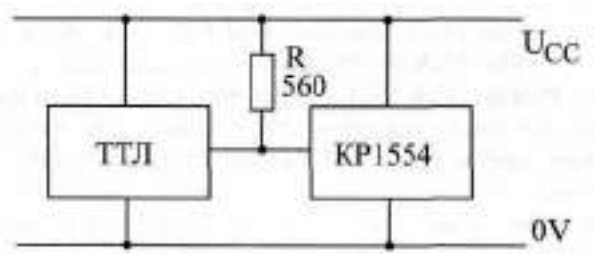


Рис. 1.19. Сопряжение микросхем ТТЛ и КР1554 (КМОП)

При выборе той или иной серии микросхем следует также учитывать, что микросхемы мощной и быстрой серии КР531 создают высокий уровень помех по шинам питания, а микросхемы маломощной серии К555 очень чувствительны к таким помехам. Поэтому серию КР531 рекомендуется использовать только в крайних случаях, при необходимости получения очень высокого быстродействия. Не рекомендуется также применять в одном устройстве мощные быстродействующие микросхемы и маломощные микросхемы.

1.6. Корпуса цифровых микросхем

Большинство микросхем имеют корпус, то есть прямоугольный контейнер (пластмассовый, керамический, металлокерамический) с металлическими выводами (ножками). Предложено множество различных типов корпусов, но наибольшее распространение получили два основных типа:

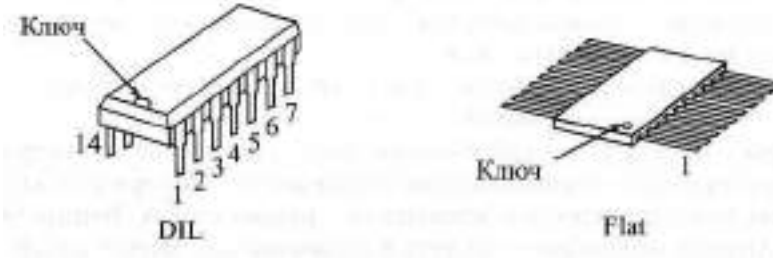


Рис. 1.20. Примеры корпусов DIL и Flat

- Корпус с двухрядным вертикальным расположением выводов, например, DIP (Dual In Line Package, Plastic) — пластмассовый корпус, DIC (Dual In Line Package, Ceramic) — керамический корпус. Общее название для таких корпусов — DIL (рис. 1.20). Расстояние между выводами составляет 0,1 дюйма (2,54 мм). Расстояние между рядами выводов зависит от количества выводов.
- Корпус с двухрядным плоскостным расположением выводов, например, FP (Flat-Package, Plastic) — пластмассовый плоский корпус, FPC (Flat-Package, Ceramic) — керамический плоский корпус. Общее название для таких корпусов - Flat (рис. 1.20). Расстояние между выводами составляет 0,05 дюйма (1,27 мм) или 0,025 дюйма (0,0628 мм).

Номера выводов всех корпусов отсчитываются начиная с вывода, помеченного ключом, по направлению против часовой стрелки (если смотреть на микросхему сверху). Ключом может служить вырез на одной из сторон микросхемы, точка около первого вывода или утолщение первого вывода (рис. 1.20). Первый вывод может находиться в левом нижнем или в правом верхнем углу (в зависимости от того, как повернут корпус). Микросхемы обычно имеют стандартное число выводов из ряда: 4, 8, 14, 16, 20, 24, 28,.. Для микросхем стандартных цифровых серий используются корпуса с количеством выводов начиная с 14.

Назначение каждого из выводов микросхемы приводится в справочниках по микросхемам, которых сейчас имеется множество. Правда, лучше ориентироваться на справочники, издаваемые непосредственно

фирмами-изготовителями. В данной книге назначение выводов не приводится.

Отечественные микросхемы выпускаются в корпусах, очень похожих на DIL и Flat, но расстояния между их выводами вычисляются по метрической шкале и поэтому чуть-чуть отличаются от принятых за рубежом. Например, 2,5 мм вместо 2,54 мм, 1,25 мм вместо 1,27 мм и т. д. Для корпусов с малым числом выводов (до 20) это не слишком существенно, но для больших корпусов расхождение в расстоянии может стать существенным. В результате на плату, рассчитанную на зарубежные микросхемы, нельзя поставить отечественные микросхемы, и наоборот.

1.7. Двоичное кодирование

Одиночный цифровой сигнал не слишком информативен, ведь он может принимать только два значения: нуль и единица. Поэтому в тех случаях, когда необходимо передавать, обрабатывать или хранить большие объемы информации, обычно применяют несколько параллельных цифровых сигналов. При этом все эти сигналы должны рассматриваться только одновременно, каждый из них по отдельности не имеет смысла. В таких случаях говорят о двоичных кодах, то есть о кодах, образованных цифровыми (логическими, двоичными) сигналами. Каждый из логических сигналов, входящих в код, называется *разрядом*. Чем больше разрядов входит в код, тем больше значений может принимать данный код.

В отличие от привычного для нас десятичного кодирования чисел, то есть кода с основанием десять, при двоичном кодировании в основании кода лежит число два (рис. 1.21). То есть каждая цифра кода (каждый разряд) двоичного кода может принимать не десять значений (как в десятичном коде: 0, 1,2, 3, 4, 5, 6, 7, 8, 9), а всего лишь два — 0 и 1. Система позиционной записи остается такой же, то есть справа пишется самый младший разряд, а слева — самый старший. Но если в десятичной системе вес каждого следующего разряда больше веса предыдущего в десять раз, то в двоичной системе (при двоичном кодировании) — в два раза. Каждый разряд двоичного кода называется *бит* (от английского «Binary Digit» — «двоичное число»).

Рис. 1.21. Десятичное и двоичное кодирование

В табл. 1.4 показано соответствие первых двадцати чисел в десятичной и двоичной системах.

Из таблицы видно, что требуемое количество разрядов двоичного кода значительно больше, чем требуемое количество разрядов десятичного кода. Максимально возможное число при количестве разрядов, равном трем, составляет при десятичной системе 999, а при двоичной — всего лишь 7 (то есть 111 в двоичном коде). В общем случае n-разрядное двоичное число может принимать 2^n различных значений, а n-разрядное десятичное число — 10^n значений. То есть запись больших двоичных чисел (с количеством разрядов больше десяти) становится не слишком удобной.

Таблица 1.4. Соответствие чисел в десятичной и двоичной системах

Десятичная система	Двоичная система	Десятичная система	Двоичная система
0	0	10	1010
1	1	11	1011
2	10	12	1100
3	11	13	1101
4	100	14	1110
5	101	15	1111
6	110	16	10000
7	111	17	10001
8	1000	18	10010
9	1001	19	10011

Для того чтобы упростить запись двоичных чисел, была предложена так называемая шестнадцатиричная система (16-ричное кодирование). В этом случае все двоичные разряды разбиваются на группы по четыре разряда (начиная с младшего), а затем уже каждая группа кодируется одним символом. Каждая такая группа называется *полубайтом* (или *нибблом, тетрадой),* а две группы (8 разрядов) — *байтом.* Из табл. 1.4 видно, что 4-разрядное двоичное число может принимать 16 разных значений (от 0 до 15). Поэтому требуемое число символов для шестнадцатиричного кода тоже равно 16, откуда и происходит название кода. В качестве первых 10 символов берутся цифры от 0 до 9, а затем используются 6 начальных заглавных букв латинского алфавита: A, B, C, D, E, F.

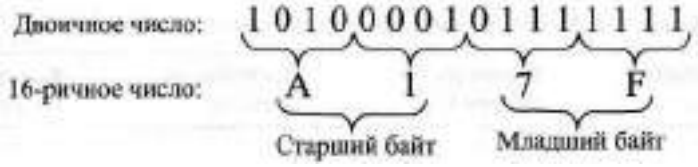


Рис. 1.22. Двоичная и 16-ричная запись числа

В табл. 1.5 приведены примеры 16-ричного кодирования первых 20 чисел (в скобках приведены двоичные числа), а на рис. 1.22 показан пример записи двоичного числа в 16-ричном виде. Для обозначения 16-ричного кодирования иногда применяют букву «h» или «H» (от английского Hexadecimal) в конце числа, например, запись A17F h обозначает 16-ричное число A17F. Здесь A1 представляет собой старший байт числа, а 7F — младший байт числа. Все число (в нашем случае — двухбайтовое) называется *словом*.

Таблица 1.5. 16-ричная система кодирования

Десятичная система	**16-ричная система**	**Десятичная система**	**16-ричная система**
0	0 (0)	10	A (1010)
1	1 (1)	11	B (1011)
2	2 (10)	12	C (1100)
3	3 (11)	13	D (1101)
4	4 (100)	14	E (1110)
5	5 (101)	15	F (1111)
6	6 (110)	16	10 (1 0000)
7.	7 (111)	17	11 (1 0001)
8	8 (1000)	18	12 (1 0010)
9	9 (1001)	19	13 (1 0011)

Для перевода 16-ричного числа в десятичное необходимо умножить значение младшего (нулевого) разряда на единицу, значение следующего (первого) разряда на 16, второго разряда на 256 (16^2) и т. д., а затем сложить все произведения. Например, возьмем число A17F:

$$A17F = F \cdot 16^0 + 7 \cdot 16^1 + 1 \cdot 16^2 + A \cdot 16^3 =$$
$$= 15 \cdot 1 + 7 \cdot 16 + 1 \cdot 256 + 10 \cdot 4096 = 41343$$

Таблица 1.6. 8-ричная система кодирования

Десятичная система	8-ричная **система**	**Десятичная система**	**8-ричная система**
0	0 (0)	10	12 (1 010)
1	1 (D	11	13 (1 011)
2	2 (10)	12	14 (1 100)
3	3 (11)	13	15 (1 101)
4	4 (100)	14	16 (1 110)
5	5 (101)	15	17 (1 111)
6	6 (110)	16	20 (10 000)
7	7 (111)	17	21 (10 001)
8	10 (1 000)	18	22 (10 010)
9	11 (1 001)	19	23 (10 011)

Но каждому специалисту по цифровой аппаратуре (разработчику, оператору, ремонтнику, программисту и т. д.) необходимо научиться так же свободно обращаться с 16-ричной и двоичной системами, как и с обычной десятичной, чтобы никаких переводов из системы в систему не требовалось.

Значительно реже, чем 16-ричное, используется восьмеричное кодирование, которое строится по такому же принципу, что и 16-ричное, но двоичные разряды разбиваются на группы по три разряда. Каждая группа (разряд кода) затем обозначается одним символом. Каждый разряд 8-ричного кода может принимать восемь значений: 0, 1,2, 3, 4, 5, 6, 7 (табл. 1.6).

Помимо рассмотренных кодов, существует также и так называемое двоично-десятичное представление чисел. Как и в 16-ричном коде, в двоично-десятичном коде каждому разряду кода соответствует четыре двоичных разряда, однако каждая группа из четырех двоичных разрядов может принимать не шестнадцать, а только десять значений, кодируемых символами 0, 1,2, 3,4, 5, 6, 7, 8, 9. То есть одному десятичному разряду соответствует четыре двоичных. В результате получается, что написание чисел в двоично-десятичном коде ничем не отличается от написания в обычном десятичном коде (табл. 1.7), но в реальности это всего лишь специальный двоичный код, каждый разряд которого может принимать только два значения: 0 и 1. Двоично-десятичный код иногда очень удобен для организации десятичных цифровых индикаторов и табло.

В двоичном коде над числами можно проделывать любые арифметические операции: сложение, вычитание, умножение, деление.

Таблица 1.7. Двоично-десятичная система кодирования

Десятичная система	Двоично-десятичная система	Десятичная система	Двоично-десятичная система
0	0 (0)	10	10 (1 0000)
1	**1 (D**	11	11 (1 0001)
2	2 (10)	12	12 (1 0010)
3	3 (11)	13	13 (1 0011)
4	4 (100)	14	14 (1 0100)
5	5 (101)	15	15 (1 0101)
6	6 (110)	16	16 (1 0110)
7	7 (111)	17	17 (1 0111)
8	8 (1000)	18	18 (1 1000)
9	9 (1001)	19	19 (1 1001)

Рассмотрим, например, сложение двух 4-разрядных двоичных чисел. Пусть надо сложить число 0111 (десятичное 7) и 1011 (десятичное 11). Сложение этих чисел не сложнее, чем в десятичном представлении:

```
  0111
+ 1011

 10010
```

При сложении 0 и 0 получаем 0, при сложении 1 и 0 получаем 1, при сложении 1 и 1 получаем 0 и перенос в следующий разряд 1. Результат — 10010 (десятичное 18). При сложении любых двух n-разрядных двоичных чисел может получиться n-разрядное или (n+1)-разрядное число.

Точно так же производится вычитание. Пусть из числа 10010 (18) надо вычесть число 0111 (7). Записываем числа с выравниванием по младшему разряду и вычитаем точно так же, как в случае десятичной системы:

```
  10010
- 0111

  1011
```

При вычитании 0 из 0 получаем 0, при вычитании 0 из 1 получаем 1, при вычитании 1 из 1 получаем 0, при вычитании 1 из 0 получаем 1 и заем 1 в следующем разряде. Результат — 1011 (десятичное 11).

При вычитании возможно получение отрицательных чисел, поэтому необходимо использовать двоичное представление отрицательных чисел.

Для одновременного представления как двоичных положительных, так и двоичных отрицательных чисел чаще всего используется так называемый дополнительный код. Отрицательные числа в этом коде выражаются таким числом, которое, будучи сложено с положительным числом такой же величины, даст в результате нуль. Для того чтобы получить отрицательное число, надо поменять все биты такого же положительного числа на противоположные (0 на 1, 1 на 0) и прибавить к результату 1. Например, запишем число —5. Число 5 в двоичном коде выглядит 0101. Заменяем биты на противоположные: 1010 и прибавляем единицу: 1011. Суммируем результат с исходным числом: 1011 + 0101 = 0000 (перенос в пятый разряд игнорируем).

Отрицательные числа в дополнительном коде отличаются от положительных значением старшего разряда: единица в старшем разряде определяет отрицательное число, а нуль — положительное.

Помимо стандартных арифметических операций, в двоичной системе счисления используются и некоторые специфические операции, например, сложение по модулю 2. Эта операция (обозначается $\oplus$) является побитовой, то есть никаких переносов из разряда в разряд и заемов в старших разрядах здесь не существует. Правила сложения по модулю 2 следующие: $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, $1 \oplus 1 = 0$. Этаже операция называется функцией *Исключающее ИЛИ*. Например, просуммируем по модулю 2 два двоичных числа 0111 и 1011:

```
  0111
⊕ 1011
  ----
  1100
```

Среди других побитовых операций над двоичными числами можно отметить функцию И и функцию ИЛИ. Функция И дает в результате единицу только тогда, когда в соответствующих битах двух исходных чисел обе единицы, в противном случае результат -0. Функция ИЛИ дает в результате единицу тогда, когда хотя бы один из соответствующих битов исходных чисел равен 1, в противном случае результат 0.

1.8. Функции цифровых устройств

Любое цифровое устройство от самого простейшего до самого сложного всегда действует по одному и тому же принципу (рис. 1.23). Оно принимает входные сигналы, выполняет их обработку, передачу, хранение и выдает выходные сигналы. При этом совсем не обязательно любое изменение входных сигналов приводит к немедленному и однозначному изменению выходных сигналов. Реакция устройства может быть очень сложной, отложенной по времени, неочевидной, но суть от этого не меняется.

В качестве входных сигналов нашего устройства могут выступать сигналы с выходов других цифровых устройств, с тумблеров и клавиш или с датчиков физических величин. Причем в последнем случае, как правило, необходимо преобразование аналоговых сигналов с датчиков в потоки цифровых кодов (рис. 1.24) с помощью аналого-цифровых преобразователей (АЦП). Например, в случае персонального компьютера входными сигналами являются сигналы с клавиатуры, с датчиков перемещения мыши, с микрофона (давление воздуха, то есть звук, преобразуется в аналоговый электрический сигнал, а затем — в цифровые коды), из кабеля локальной сети и т. д.

Рис. 1.23. Включение цифрового устройства

Выходные сигналы цифрового устройства могут предназначаться для подачи на другие цифровые устройства, для индикации (на экране монитора, на цифровом индикаторе и т. д.), а также для формирования физических величин. Причем в последнем случае необходимо преобразовывать потоки кодов с цифрового устройства в непрерывные (аналоговые) сигналы (рис. 1.24) с помощью цифро-аналоговых преобразователей (ЦАП) и в физические величины. Например, в случае персонального компьютера выходными сигналами будут: сигналы, подаваемые компьютером на принтер; сигналы, идущие на видеомонитор (аналоговые или цифровые); звук, воспроизводимый динамиками компьютера (потоки кодов с компьютера преобразуются в аналоговый электрический сигнал, который затем преобразуется в давление воздуха — звук).

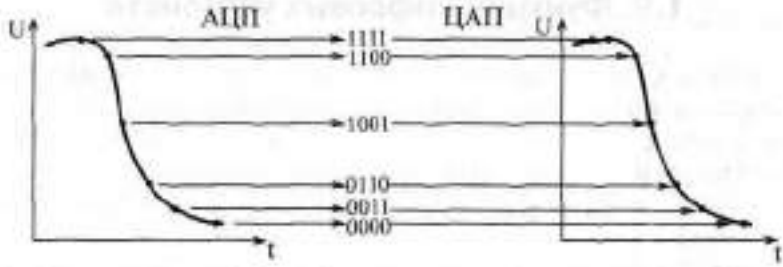


Рис. 1.24. Аналого-цифровое и цифро-аналоговое преобразование

Одно цифровое устройство может состоять из нескольких более простых цифровых устройств. Часто эти составные элементы называют блоками, модулями, узлами, частями. Если объединяются несколько сложных цифровых устройств, то говорят уже о цифровых системах, комплексах, установках. Мы в основном будем использовать термин «устройство», как занимающий промежуточное положение.

Связь между входными и выходными сигналами может быть жесткой, неизменной или гибко изменяемой (программируемой). То есть цифровое устройство может работать по жесткому, раз и навсегда установленному алгоритму или по алгоритму программируемому. Как правило, при этом выполняется один очень простой принцип: чем больше возможностей для изменения связи входных и выходных сигналов, чем больше возможностей изменения алгоритма работы, тем цифровое устройство будет медленнее. Речь в данном случае, конечно же, идет о предельно достижимом быстродействии.

Иначе говоря, простые устройства с жесткой логикой работы всегда могут быть сконструированы с более высоким быстродействием по сравнению с программируемыми, гибкими устройствами со сложным алгоритмом работы. Жесткая логика также обеспечивает малый объем аппаратуры (малые аппаратурные затраты) для реализации простых функций. Зато программируемые, интеллектуальные устройства обеспечивают более высокую гибкость и меньшую стоимость при необходимости сложной обработки. А для реализации простых функций они часто оказываются избыточно сложными. Так что выбор между двумя этими типами цифровых устройств зависит от конкретной решаемой задачи.

Значительное число задач может быть решено как чисто аппаратным путем (с помощью устройств на жесткой логике), так и программно-аппаратным путем (с помощью программируемых устройств). В таких случаях надо смотреть, какие характеристики устройства являются самыми важными: скорость работы, стоимость, гибкость, простота проектирования и т. д. — и в зависимости от этого выбирать то или иное решение, так

или иначе перераспределять функции между программным обеспечением и аппаратурой.

В данной книге основное внимание будет уделено устройствам и узлам с жесткой логикой работы. Однако уяснение принципов их работы и их проектирования может оказать большую помощь и при создании программируемых, интеллектуальных устройств.

Глава 2. Применение логических элементов

Лекция 3. Простейшие логические элементы

В лекции рассматриваются принципы работы, характеристики и типовые схемы включения простейших логических элементов — инверторов, буферов, элементов И и ИЛИ, а также приводятся схемотехнические решения, позволяющие реализовать на их основе часто встречающиеся функции.

Ключевые слова: вентиль, инвертор, буферы однонаправленные и двунаправленные, элементы И и И-НЕ, элементы ИЛИ и ИЛИ-НЕ, генераторы импульсов, мультиплексирование, двунаправленная передача, пропускатель, смеситель, схема совпадения, селектирование кодов.

Изучение базовых элементов цифровой электроники мы начнем с наиболее простых, а затем будем рассматривать все более сложные. Примеры применения каждого следующего элемента будут опираться на все элементы, рассмотренные ранее. Таким образом, будут постепенно даны главные принципы построения довольно сложных цифровых устройств.

Логические элементы (или, как их еще называют, *вентили,* «gates») — это наиболее простые цифровые микросхемы. Именно в этой простоте и состоит их отличие от других микросхем. Как правило, в одном корпусе микросхемы может располагаться от одного до шести одинаковых логических элементов. Иногда в одном корпусе могут располагаться и разные логические элементы.

Обычно каждый логический элемент имеет несколько входов (от одного до двенадцати) и один выход. При этом связь между выходным сигналом и входными сигналами (таблица истинности) предельно проста. Каждой комбинации входных сигналов элемента соответствует уровень нуля или единицы на его выходе. Никакой внутренней памяти у логических элементов нет, поэтому они относятся к группе так называемых *комбинационных микросхем.* Но в отличие от более сложных комбинационных микросхем, рассматриваемых в следующей главе, логические элементы имеют входы, которые не могут быть разделены на группы, различающиеся по выполняемым ими функциям.

Главные достоинства логических элементов, по сравнению с другими цифровыми микросхемами, — это их высокое быстродействие (малые времена задержек), а также малая потребляемая мощность (ма-

лый ток потребления). Поэтому в тех случаях, когда требуемую функцию можно реализовать исключительно на логических элементах, всегда имеет смысл проанализировать этот вариант. Недостаток же их состоит в том, что на их основе довольно трудно реализовать сколько-нибудь сложные функции. Поэтому чаще всего логические элементы используются только в качестве дополнения к более сложным, к более «умным» микросхемам. И любой разработчик обычно стремится использовать их как можно меньше и как можно реже. Существует даже мнение, что мастерство разработчика обратно пропорционально количеству используемых им логических элементов. Однако это верно далеко не всегда.

2.1. Инверторы

Самый простой логический элемент — это инвертор (логический элемент НЕ, «inverter»), уже упоминавшийся в первой главе. Инвертор выполняет простейшую логическую функцию — инвертирование, то есть изменение уровня входного сигнала на противоположный. Он имеет всего один вход и один выход. Выход инвертора может быть типа 2С или типа ОК. На рис. 2.1 показаны условные обозначения инвертора, принятые у нас и за рубежом, а в табл. 2.1 представлена таблица истинности инвертора.

Рис. 2.1. Условные обозначения инверторов: зарубежные (слева) и отечественные (справа)

В одном корпусе микросхемы обычно бывает шесть инверторов. Отечественное обозначение микросхем инверторов — «ЛН». Примеры: КР1533ЛН1 (SN74ALS04) - шесть инверторов с выходом 2С, КР1533ЛН2 (SN74ALS05) — шесть инверторов с выходом ОК. Существуют также инверторы с выходом ОК и с повышенным выходным током (ЛН4), а также с повышенным выходным напряжением (ЛН3, ЛН5). Для инверторов с выходом ОК необходимо включение выходного нагрузочного резистора pull-up. Его минимальную величину можно рассчитать очень просто: $R < U/I_{OL}$, где U — напряжение питания, к которому подключается резистор. Обычно величина резистора выбирается порядка сотен Ом — единиц кОм.

Таблица 2.1. Таблица истинности инвертора

Вход	Выход
0	1
1	0

Две основные области применения инверторов — это изменение полярности сигнала и изменение полярности фронта сигнала (рис. 2.2). То есть из положительного входного сигнала инвертор делает отрицательный выходной сигнал и наоборот, а из положительного фронта входного сигнала — отрицательный фронт выходного сигнала и наоборот. Еще одно важное применение инвертора — буферирование сигнала (с инверсией), то есть увеличение нагрузочной способности сигнала. Это бывает нужно в том случае, когда какой-то сигнал надо подать на много входов, а выходной ток источника сигнала недостаточен.

Рис. 2.2. Инверсия полярности сигнала и инверсия полярности фронта сигнала

Именно инвертор, как наиболее простой элемент, чаще других элементов используется в нестандартных включениях. Например, инверторы обычно применяются в схемах генераторов прямоугольных импульсов (рис. 2.3), выходной сигнал которых периодически меняется с нулевого уровня на единичный и обратно. Все приведенные схемы, кроме схемы *д,* выполнены на элементах К155ЛН1, но могут быть реализованы и на инверторах других серий при соответствующем изменении номиналов резисторов. Например, для серии К555 номиналы резисторов увеличиваются примерно втрое. Схема *д* выполнена на элементах КР531ЛН1, так как она требует высокого быстродействия инверторов.

Схемы *а, б* и *в* представляют собой обычные RC-генераторы, характеристики которых (выходную частоту, длительность импульса) можно рассчитать только приблизительно. Для схем *а* и *б* при указанных номиналах резистора и конденсатора частота генерации составит порядка 100 кГц, для схемы *в* — около 1 МГц. Эти схемы рекомендуется использовать только в тех случаях, когда частота не слишком важна, а важен сам факт генерации. Если же точное значение частоты принципиально, то рекомендуется применять схемы *г* и *д,* в которых частота выходного сигнала определяется только характеристиками кварцевого резонатора.

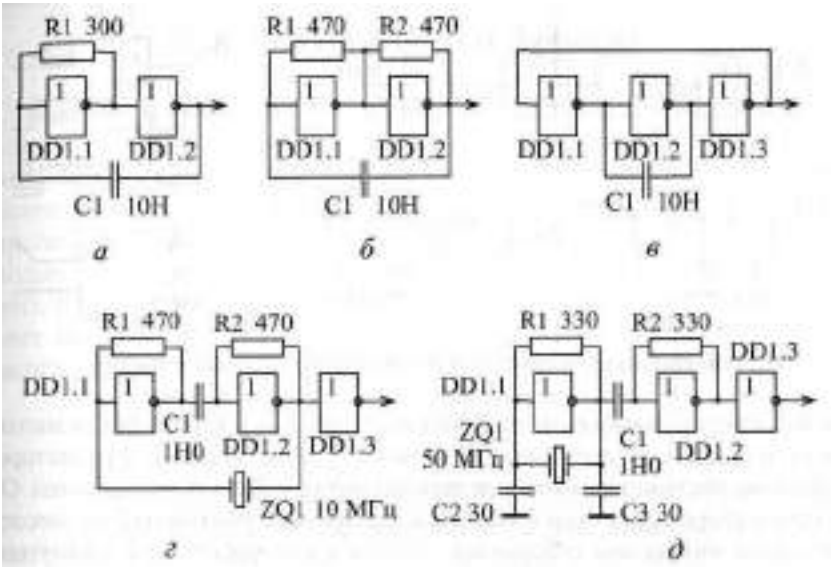


Рис. 2.3. Схемы генераторов импульсов на инверторах

Схема *г* используется для кварцевого резонатора, работающего на первой (основной) гармонике. Величину емкости можно оценить по формуле:

$$C > 1/(2\pi F),$$

где F — частота генерации. Схема *д* применяется для гармониковых кварцевых резонаторов, которые работают на частоте, большей основной в 3, 5, 7 раз (это бывает нужно для частот генерации выше 20 МГц).

Инверторы также применяются в тех случаях, когда необходимо получить задержку сигнала, правда, незначительную (от 5 до 100 нс). Для получения такой задержки последовательно включается нужное количество инверторов (рис. 2.4, вверху). Суммарное время задержки, например, для четырех инверторов, можно оценить по формуле

$$t_З = 2t_{PHL} + 2t_{PLH}.$$

Правда, надо учитывать, что обычно реальные задержки элементов оказываются существенно ниже (иногда даже вдвое), чем табличные параметры t_{PHL} и t_{PLH}. То есть о точном значении получаемой задержки говорить не приходится, ее можно оценить только примерно.

Для задержки сигнала используются также конденсаторы (рис. 2.4, внизу). При этом задержка возникает из-за медленного заряда и разряда

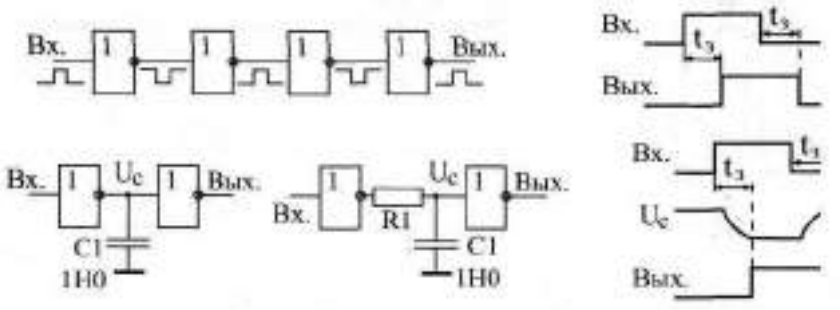


Рис. 2.4. Использование инверторов для задержки сигнала

конденсатора (напряжение на конденсаторе — U_c). Схема без резистора (слева на рисунке) дает задержку около 100 **нс**. В схеме с резистором (справа на рисунке) номинал резистора должен быть порядка сотен Ом. Но при выборе таких схем с конденсаторами надо учитывать, что некоторые серии микросхем (например, КР1533) плохо работают с затянутыми фронтами входных сигналов. Кроме того, надо учитывать, что количество времязадающих конденсаторов в схеме обратно пропорционально уровню мастерства разработчика схемы.

Наконец, еще одно применение инверторов, но только с выходом ОК, состоит в построении на их основе так называемых элементов «Проводного ИЛИ». Для этого выходы нескольких инверторов с выходами ОК объединяются, и через резистор присоединяются к источнику питания (рис. 2.5). Выходом схемы является объединенный выход всех элементов. Такая конструкция выполняет логическую функцию **ИЛИ-НЕ,** то есть на выходе будет сигнал логической единицы только при нулях на всех входах. Но о логических функциях подробнее будет рассказано в разделе 2.3.

В заключение раздела надо отметить, что инверсия сигнала применяется и внутри более сложных логических элементов, а также внутри цифровых микросхем, выполняющих сложные функции.

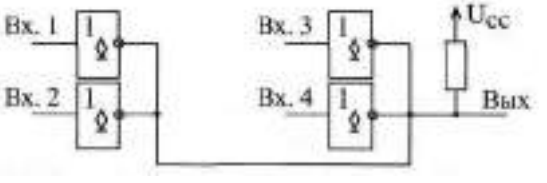


Рис. 2.5. Объединение выходов инверторов с ОК для функции ИЛИ-НЕ

2.2. Повторители и буферы

Повторители и буферы отличаются от инверторов прежде всего тем, что они не инвертируют сигнал (правда, существуют и инвертирующие буферы). Зачем же тогда они нужны? Во-первых, они выполняют функцию увеличения нагрузочной способности сигнала, то есть позволяют подавать один сигнал на много входов. Для этого имеются буферы с повышенным выходным током и выходом 2С, например, ЛП16 (шесть буферных повторителей). Во-вторых, большинство буферов имеют выход ОК или ЗС, что позволяет использовать их для получения двунаправленных линий или для мультиплексирования сигналов. Поясним подробнее эти термины.

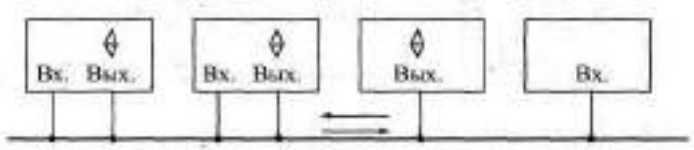


Рис. 2.6. Двунаправленная линия

Под *двунаправленными линиями* понимаются такие линии (провода), сигналы по которым могут распространяться в двух противоположных направлениях. В отличие от однонаправленных линий, которые идут от одного выхода к одному или нескольким входам, к двунаправленной линии могут одновременно подключаться несколько выходов и несколько входов (рис. 2.6). Понятно, что двунаправленные линии могут организовываться только на основе выходов ОК или ЗС. Поэтому почти все буферы имеют именно такие выходы.

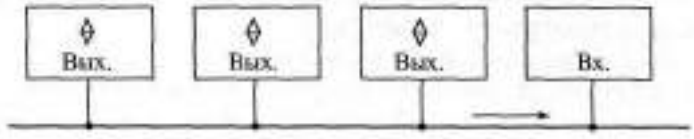


Рис. 2.7. Однонаправленная мультиплексированная линия на основе буферов

Мультиплексированием называется передача разных сигналов по одним и тем же линиям в разные моменты времени. Основная цель мультиплексирования состоит в сокращении общего количества соединительных линий. Двунаправленная линия обязательно является мультиплексированной, а мультиплексированная линия может быть как однонаправленной, так и двунаправленной. Но в любом случае к ней присоединяет-

ся несколько выходов, только один из которых в каждый момент времени находится в активном состоянии. Остальные выходы в это время отключаются (переводятся в пассивное состояние). В отличие от двунаправленной линии, к мультиплексированной линии, построенной на основе буферов, может быть подключен всего лишь один вход, но обязательно несколько выходов с ОК или ЗС (рис. 2.7). Мультиплексированные линии могут строиться не только на буферах, но и на микросхемах мультиплексоров, которые будут рассмотрены в главе 3.

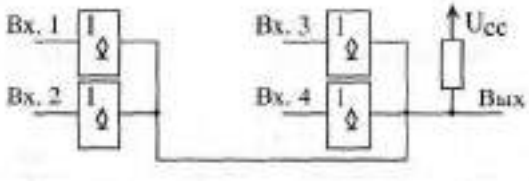


Рис. 2.8. Объединение выходов буферов с ОК

Примером буферов с выходом О К является микросхема ЛП17 (шесть буферов о ОК). Точно так же, как и в случае инверторов с ОК (см. рис. 2.5), выходы нескольких буферов с ОК могут объединяться для получения функции «Монтажное И», то есть на выходе будет сигнал логической единицы только при единицах на всех входах (рис. 2.8). То есть реализуется многовходовой элемент И (см. раздел 2.3).

Буферы с выходом ЗС представлены гораздо большим количеством микросхем, например, ЛП8, ЛП11, АП5, АП6, АП14. Эти буферы обязательно имеют управляющий вход EZ (или OE), переводящий выходы в третье, пассивное состояние. Как правило, третьему состоянию соответствует единица на этом входе, а активному состоянию выходов — нуль, то есть сигнал EZ имеет отрицательную полярность.

Буферы бывают однонаправленные или двунаправленные, с инверсией или без инверсии сигналов, с управлением всеми выходами одновременно или с управлением группами выходов. Всем этим и определяется большое разнообразие микросхем буферов.

Таблица 2.2. Таблица истинности буфера без инверсии

Вход	**-EZ**	**Выход**
0	**0**	**0**
1	**0**	**1**
0	**1**	ЗС
1	**1**	ЗС

Простейшим однонаправленным буфером без инверсии является микросхема ЛП8 (четыре буфера с выходами типа ЗС и раздельным управлением). Каждый из четырех буферов имеет свой вход разрешения EZ. Таблица истинности буфера очень проста (табл. 2.2): при нулевом сигнале на входе управления выход повторяет вход, а при единичном — выход отключен. Эту микросхему удобно применять для обработки одиночных сигналов, то есть для повторения входного сигнала с возможностью отключения выхода.

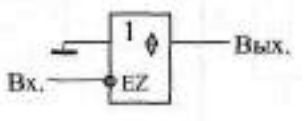


Рис. 2.9. Применение буфера с ЗС в качестве буфера с ОК

Эти же буферы иногда удобно использовать для замещения буферов с выходом ОК (рис. 2.9). В этом случае вход управления служит информационным входом. При нуле на входе мы получаем нуль на выходе, а при единице на входе — третье состояние на выходе.

Очень часто надо обрабатывать не одиночные сигналы, а группы сигналов, например, сигналы, передающие многоразрядные коды. В этом случае удобно применять буферы с групповым управлением, То есть имеющие один вход разрешения EZ для нескольких выходов. Примерами могут служить микросхемы ЛП11 (шесть буферов, разделенные на две группы: четыре и два буфера, для каждой из которых имеется свой вход управления) и АП5 (восемь буферов, разделенные на две группы по четыре буфера, каждая из которых имеет свой вход управления).

На рис. 2.10 показан пример мультиплексирования двух восьмиразрядных кодов с помощью двух микросхем АП5. Одноименные выходы обеих микросхем объединены между собой. Пропускание на выход каждого из двух входных кодов разрешается своим управляющим сигналом (Упр. 1 и Упр. 2), причем должен быть исключен одновременный приход этих двух сигналов, чтобы не было конфликтов на выходах.

Двунаправленные буферы, в отличие от однонаправленных, позволяют передавать сигналы в обоих направлениях. В зависимости от специального управляющего сигнала Т (другое обозначение — BD), входы могут становиться выходами и наоборот: выходы — входами. Обязательно имеется и вход управления третьим состоянием EZ, который может отключить как входы, так и выходы.

На рис. 2.11 для примера показан двунаправленный буфер АП6, который может передавать данные между двумя двунаправленными шинами

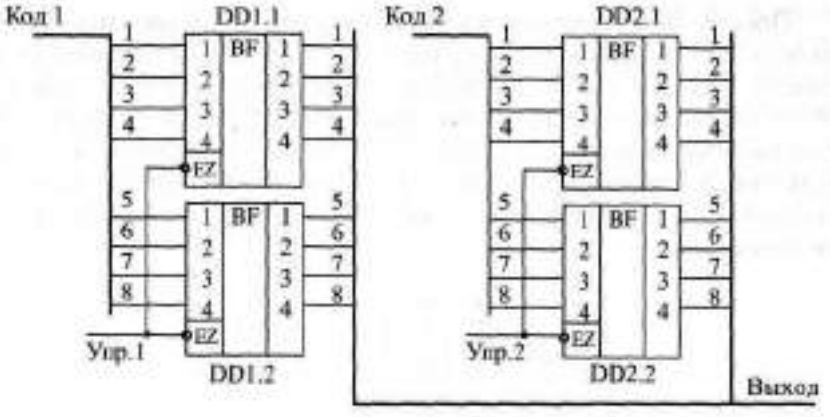


Рис. 2.10. Мультиплексирование двух входных кодов с помощью буферов с ЗС

А и В в обоих направлениях. При единичном уровне на управляющем входе Т (сигнал Напр.) данные передаются из шины А в шину В, а при нулевом уровне — из шины В в шину А (табл. 2.3). Единичный уровень на управляющем входе EZ (сигнал Откл.) отключает микросхему от обеих шин.

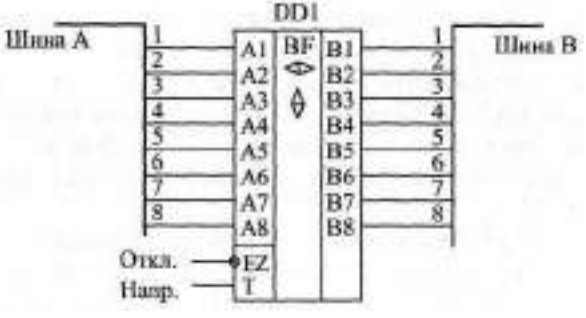


Рис. 2.11. Включение двунаправленного буфера

Таблица 2.3. Таблица истинности двунаправленного буфера

Вход Т	Вход-EZ	Операция
0	0	**В -> А**
1	**0**	**А -> В**
0	**1**	ЗС
1	**1**	ЗС

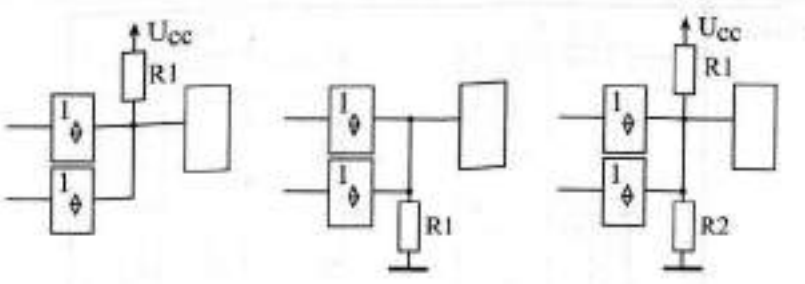


Рис. 2.13. Включение резисторов на выходе буферов ЗС

пряжением питания (рис. 2.13), тогда при отключенном выходе на вход будет поступать уровень логической единицы. Однако можно включить и резистор между выходом и землей, тогда при отключенном выходе на вход будет поступать сигнал логического нуля. Применяется также и включение двух резисторов (резистивного делителя), при этом величина верхнего резистора (присоединенного к шине питания) обычно выбирается в 2–3 раза меньше, чем нижнего резистора (присоединенного к «земле»), а величина параллельного соединенных двух резисторов выбирается равной примерно 100 Ом. Например, резисторы могут иметь номиналы 240 Ом и 120 Ом, 360 Ом и 130 Ом. Отключенный выход воспринимается в данном случае присоединенным к нему входом как единица.

Иногда к выходам ЗС резисторы не присоединяют вообще, но в этом случае надо обеспечить, чтобы последующий вход воспринимал сигнал с выхода ЗС (то есть реагировал на него) только тогда, когда выход находится в активном состоянии. Иначе возможны сбои и отказы в работе устройства.

Еще одно типичное применение буферов, связанное с их большими выходными токами, - это светодиодная индикация. Светодиоды могут подключаться к выходу буферов двумя основными способами (рис. 2.14). При первом из них (слева на рисунке) светодиод горит, когда на выходе ЗС или 2С-сигнал логической единицы, а при втором (справа на рисунке) — когда на выходе ОК сигнал логического нуля. Величина резистора выбирается исходя из характеристик светодиода, но обычно составляет порядка 1 кОм.

Рис. **2.14.** Применение буферов для индикации

2.3. Элементы И, И-НЕ, ИЛИ, ИЛИ-НЕ

Следующий шаг на пути усложнения компонентов цифровой электроники — это элементы, выполняющие простейшие логические функции. Объединяет все эти элементы то, что у них есть несколько *равноправных* входов (от 2 до 12) и один выход, сигнал на котором определяется комбинацией входных сигналов.

Самые распространенные логические функции — это И (в отечественной системе обозначений — ЛИ), И-НЕ (обозначается ЛА), ИЛИ (обозначается ЛЛ) и ИЛИ-НЕ (обозначается ЛЛ). Присутствие слова НЕ в названии элемента обозначает только одно — встроенную инверсию сигнала. В международной системе обозначений используются следующие сокращения: AND — функция И, NAND — функция И-НЕ, OR — функция ИЛИ, NOR — функция ИЛИ-НЕ.

Название самих функций И и ИЛИ говорит о том, при каком условии на входах появляется сигнал на выходе. При этом важно помнить, что речь в данном случае идет о положительной логике, о положительных, единичных сигналах на входах и на выходе.

Элемент И формирует на выходе единицу тогда и только тогда, если на всех его входах (и на первом, и на втором, и на третьем и т. д.) присутствуют единицы. Если речь идет об элементе И-НЕ, то на выходе формируется нуль, когда на всех входах — единицы (табл. 2.4). Цифра перед названием функции говорит о количестве входов элемента. Например, 8И-НЕ — это восьмивходовой элемент И с инверсией на выходе.

Таблица 2.4. Таблица истинности двухвходовых элементов И, И-НЕ, ИЛИ, ИЛИ-НЕ

Вход 1	Вход 2	Выход И	Выход И-НЕ	Выход ИЛИ	Выход ИЛИ-НЕ
0	0	0	1	0	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	0	1	0

Элемент ИЛИ формирует на выходе нуль тогда и только тогда, если хотя бы на одном из входов присутствует единица (или на первом, или на втором, или на третьем и т. д.). Элемент ИЛИ-НЕ дает на выходе нуль при наличии хотя бы на одном из входов единицы (табл. 2.4). Пример обозначения: 4ИЛИ-НЕ — четырехвходовой элемент ИЛИ с инверсией на выходе.

Отечественные и зарубежные обозначения на схемах двухвходовых элементов И, И-НЕ, ИЛИ, ИЛИ-НЕ показаны на рис. 2.15. Все эти элементы

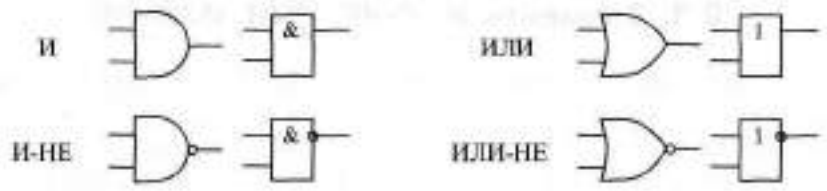


Рис. 2.15. Обозначения элементов И, И-НЕ, ИЛИ, ИЛИ-НЕ: зарубежные (слева) и отечественные (справа)

бывают с выходами типа 2С, ОК и ЗС. В последнем случае обязательно имеется вход разрешения -EZ.

Нетрудно заметить (см. табл. 2.4), что в случае отрицательной логики, при нулевых входных и выходных сигналах, элемент И выполняет функцию ИЛИ, то есть на выходе будет нуль в случае, когда хотя бы на одном из входов нуль. А элемент ИЛИ при отрицательной логике выполняет функцию И, то есть на выходе будет нуль только тогда, когда на всех входах присутствуют нули. И так как в реальных электронных устройствах сигналы могут быть любой полярности (как положительные, так и отрицательные), то надо всегда очень аккуратно выбирать требуемый в каждом конкретном случае элемент. Особенно об этом важно помнить тогда, когда последовательно соединяются несколько разноименных логических элементов с инверсией и без нее для получения сложной функции.

Поэтому элементы И, И-НЕ, ИЛИ, ИЛИ-НЕ разработчику далеко не всегда удобно применять именно как выполняющие указанные в их названии логические функции. Иногда их удобнее использовать как элементы разрешения/запрещения или смешивания/совпадения. Но сначала мы рассмотрим случаи реализации именно логических функций на этих элементах.

На рис. 2.16 приведены примеры формирования элементами выходных сигналов на основании требуемых временных диаграмм входных и выходных сигналов. В случае *а* выходной сигнал должен быть равен единице при двух единичных входных сигналах, следовательно, достаточно элемента 2И. В случае *б* выходной сигнал должен быть равен нулю, когда хотя бы один из входных сигналов равен единице, следовательно, требуется элемент 2ИЛИ-НЕ. Наконец, в случае *в* выходной сигнал должен быть равен нулю при одновременном приходе единичного сигнала Вх. 1, нулевого сигнала Вх. 2 и единичного сигнала Вх. 3. Следовательно, требуется элемент ЗИ-НЕ, причем сигнал Вх. 2 надо предварительно проинвертировать.

Любой из логических элементов рассматриваемой группы можно рассматривать как управляемый пропускатель входного сигнала (с инверсией или без нее).

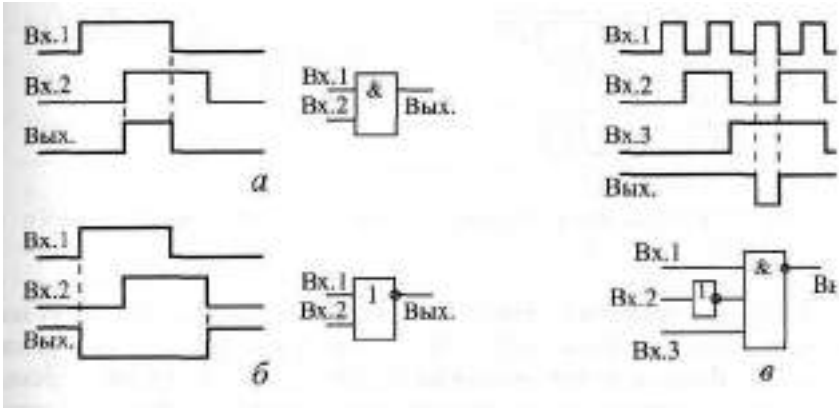


Рис. 2.16. Примеры применения элементов И и ИЛИ

Например, в случае элемента 2И-НЕ один из входов можно считать информационным, а другой — управляющим. В этом случае при единице на управляющем входе выходной сигнал будет равен проинвертированному входному сигналу, а при нуле на управляющем входе выходной сигнал будет постоянно равен единице, то есть прохождение входного сигнала будет запрещено. Элементы 2И-НЕ с выходом О К часто используют именно в качестве управляемых буферов для работы на мультиплексированную или двунаправленную линию.

Точно так же в качестве элемента разрешения/запрещения могут применяться элементы И, ИЛИ, ИЛИ-НЕ (рис. 2.17). Разница между элементами состоит только в полярности управляющего сигнала, в инверсии (или ее отсутствии) входного сигнала, а также в уровне выходного сигнала (нуль или единица) при запрещении прохождения входного сигнала.

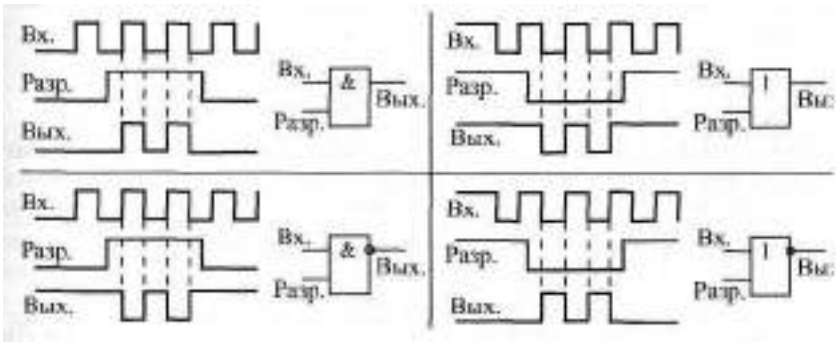


Рис. 2.17. Разрешение/запрещение прохождения сигналов на элементах И, И-НЕ, ИЛИ, ИЛИ-НЕ

Рис. 2.18. Появление лишнего фронта при запрещении входного сигнала

При использовании элементов разрешения/запрещения могут возникнуть дополнительные проблемы в случае, когда сигнал с выхода элемента идет на вход, реагирующий на фронт сигнала. В момент перехода из состояния разрешения в состояние запрещения и из состояния запрещения в состояние разрешения в выходном сигнале может появиться дополнительный фронт, никак не связанный с входным сигналом (рис. 2.18). Чтобы этого не произошло, надо придерживаться следующего простого правила: если вход реагирует на положительный фронт, то в состоянии запрещения на выходе элемента должен быть нуль, и наоборот.

Иногда необходимо реализовать функцию смешивания двух сигналов той или иной полярности. То есть выходной сигнал должен вырабатываться как при приходе одного входного сигнала, так и при приходе другого входного сигнала. Если оба входных сигнала положительные и выходной сигнал положительный, то мы имеем в чистом виде функцию ИЛИ, и требуется элемент 2ИЛИ. Однако при отрицательных входных сигналах и отрицательном выходном сигнале для такого же смешивания понадобится уже элемент 2И. А если полярность входных сигналов не совпадает с нужной полярностью выходного сигнала, то нужны уже элементы с инверсией (И-НЕ при положительных выходных сигналах и ИЛИ-НЕ при отрицательных выходных сигналах). На рис. 2.19 показаны варианты смешивания на разных элементах.

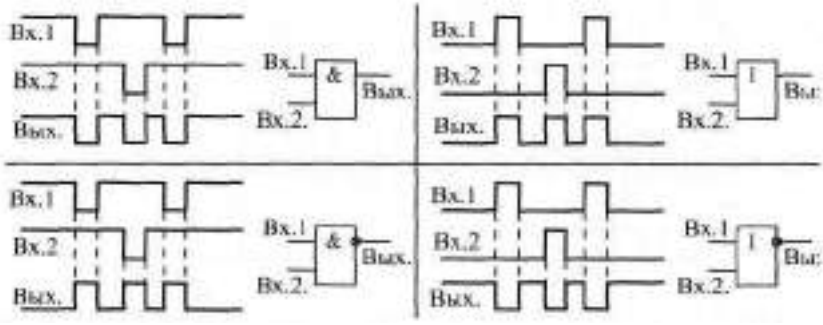


Рис. 2.19. Реализация смешивания двух сигналов

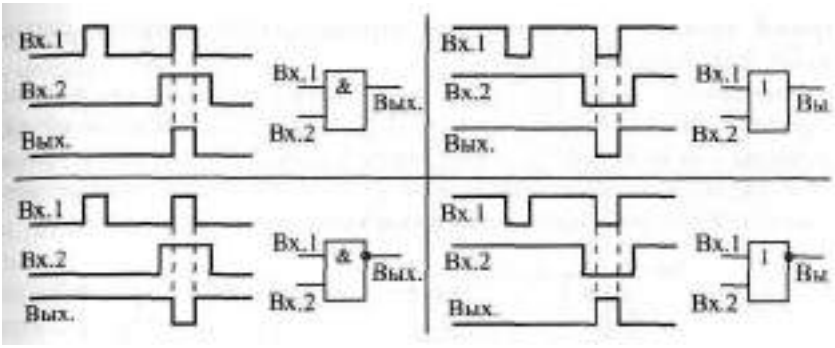


Рис. 2.20. Схемы совпадения двух сигналов

Наконец, рассматриваемые элементы И, И-НЕ, ИЛИ, ИЛИ-НЕ иногда бывает удобно применять в качестве схем совпадения различных сигналов. То есть выходной сигнал должен вырабатываться тогда, когда сигналы на входах совпадают (приходят одновременно). Если же совпадения нет, то выходной сигнал должен отсутствовать. На рис. 2.20 показаны варианты таких схем совпадения на четырех разных элементах. Различаются они полярностями входных сигналов, а также наличием или отсутствием инверсии выходного сигнала.

Рассмотрим два примера совместного использования элементов И, И-НЕ, ИЛИ, ИЛИ-НЕ (рис. 2.21).

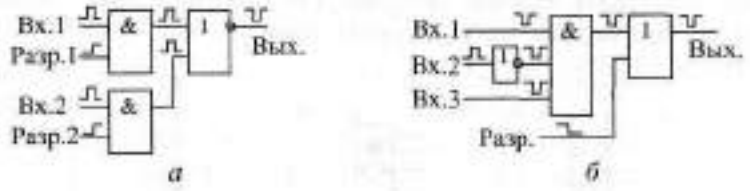


Рис.2.21. Примеры совместного использования элементов

Пусть необходимо смешать два сигнала, каждый из которых может быть разрешен или запрещен. Пусть полярность входных сигналов и сигналов разрешения — положительная, а выходной сигнал должен быть отрицательным. В этом случае надо взять два двухвходовых элемента И и смешать их выходные сигналы с помощью двухвходового элемента ИЛИ-НЕ (*а*).

Пусть необходимо смешать два отрицательных сигнала и один положительный сигнал, причем результирующий сигнал может быть разрешен или запрещен. Полярность сигнала разрешения — отрицательная, полярность выходного сигнала — отрицательная. Для этого нужно взять трех-

входовой элемент И, инвертор для отрицательного входного сигнала и двухвходовой элемент ИЛИ (б).

Элементы И, И-НЕ, ИЛИ, ИЛИ-НЕ могут использоваться также в качестве инверторов или повторителей (рис. 2.22), для чего необходимо объединить входы или на неиспользуемые входы подать сигнал нужного уровня. Второе предпочтительнее, так как объединение входов не только увеличивает входной ток, но и несколько снижает быстродействие элементов.

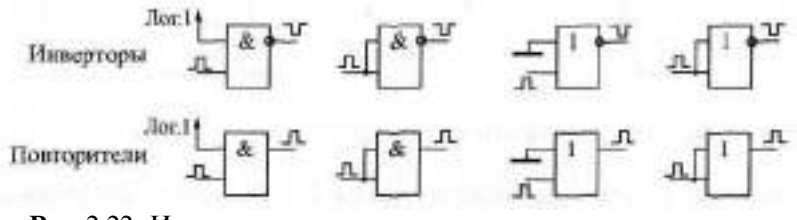


Рис. 2.22. Инверторы и повторители

По функции И часто объединяются входы более сложных микросхем. Иначе говоря, какая-то функция выполняется только тогда, когда на все объединенные по И входы поступают сигналы нужной полярности. Чаще всего по И объединяются входы выбора микросхемы CS и входы управления третьим состоянием выхода микросхемы EZ. На рис. 2.23 показано три примера такого объединения по И. При этом надо учитывать, что на инверсные входы для выполнения функции должны поступать нулевые сигналы, а на прямые входы — единичные сигналы. Примерами могут служить микросхемы КР556РТ4, КР556РТ5, КР1533АП14, КР1533АП15.

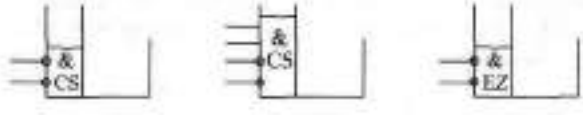


Рис. 2.23. Объединение по И входов микросхем

До сих пор, рассматривая элементы И, И-НЕ, ИЛИ, ИЛИ-НЕ, мы не выходили за рамки первого уровня представления (логической модели). Это вполне допустимо в том случае, когда входные сигналы элементов не меняются одновременно или почти одновременно, когда их фронты разнесены во времени существенно (больше, чем на время задержки элемента). При одновременном изменении входных сигналов все будет гораздо сложнее необходимо привлекать второй и порой третий уровень представления. В момент изменения входных сигналов выходной сигнал становится неопределенным, нестабильным, непредсказуемым. В результате

при неправильном проектировании может не работать вся сложная схема, целый прибор или даже большая система.

Например, возьмем логический элемент 2И-НЕ. Пусть на его входы приходят сигналы, изменяющиеся одновременно, причем в противофазе, то есть один переключается из нуля в единицу, а другой — из единицы в нуль. Пусть по тем или иным причинам (вследствие передачи по проводам, вследствие разных задержек элементов и т. д.) один из сигналов чуть-чуть сдвинулся во времени относительно другого (рис. 2.24). При этом на двух входах в течение кратковременного периода будет присутствовать два единичных сигнала. В результате выход начнет переключаться из единицы в нуль. Он может успеть переключиться, и тогда сформируется короткий импульс. Он может не успеть переключиться, и тогда импульса не будет. Он может иногда успевать переключиться, а иногда не успевать, и тогда выходной импульс то будет появляться, то не будет. Здесь все зависит от быстродействия элемента и величины задержки. Последняя ситуация наиболее неприятна, так как может вызвать нестабильную неисправность, выявить которую крайне сложно.

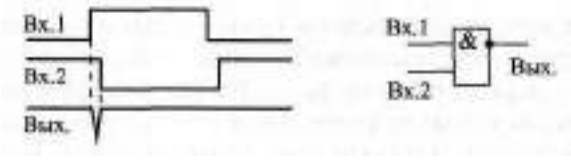


Рис. 2.24. Короткий импульс на выходе элемента 2И-НЕ

На этапе проектирования схемы бороться с такими паразитными импульсами довольно просто: надо всего лишь обеспечить, чтобы вся дальнейшая схема на них не реагировала, например, отключалась на некоторое время после изменения входных сигналов элементов. То есть необходимо временное согласование, синхронизация различных элементов схемы.

В качестве примера возьмем одно из самых распространенных применений рассматриваемых элементов И, И-НЕ, ИЛИ, ИЛИ-НЕ — селектирование кодов. Суть селектирования сводится к следующему. Пусть имеется некоторая шина, по которой передаются коды. Необходимо выявить появление на этой шине какого-то определенного кода, то есть сформировать выходной сигнал, соответствующий требуемому входному коду.

Схема, выполняющая такую функцию, довольно проста (рис. 2.25). В ее основе — многовходовые элементы И-НЕ. При этом сигналы, соответствующие разрядам кода, на которых должны быть единицы, подаются непосредственно на входы элементов И-НЕ. А сигналы, соответствующие разрядам кода, на которых должны быть нули, подаются на входы эле-

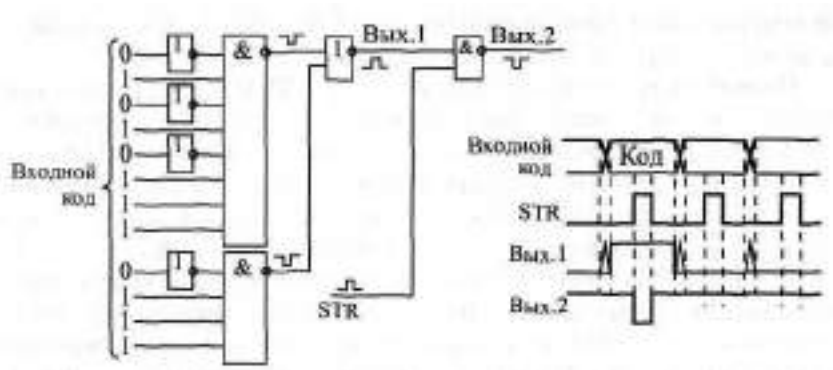


Рис. 2.25. Селектирование кодов со стробированием

ментов И-НЕ через инверторы. Выходные сигналы элементов И-НЕ объединяются с помощью элемента ИЛИ-НЕ. В результате на выходе элемента ИЛИ-НЕ формируется сигнал Вых. 1 в тот момент, когда на входе присутствует нужный код.

Однако в момент установления нужного кода и в момент его снятия возникает период неопределенности, когда в выходном сигнале могут быть короткие паразитные импульсы. Это связано как с неодновременным приходом различных разрядов, так и с внутренними задержками нашей схемы. Более того, короткие паразитные импульсы могут возникать на выходе и в том случае, когда любой входной код меняется на любой другой входной код, даже если оба эти кода не селектируются нашей схемой. То есть любое изменение кода всегда сопровождается периодом неопределенности в сигнале Вых. 1.

Как же добиться, чтобы выходной сигнал не имел паразитных импульсов, не имел периодов неопределенности? Для этого обычно используется стробирование или тактирование передаваемого кода. То есть, помимо кода, параллельно с ним передается стробирующий или тактирующий сигнал STR, задержанный во времени относительно кода. Активным этот сигнал становится тогда, когда все предыдущие переходные процессы уже завершены, все разряды кода установились в нужные уровни и схема, обрабатывающая код, тоже закончила свою работу. А пассивным он становится, пока еще не начались новые переходные процессы. Это называется *вложенным циклом* (то есть в нашем случае сигнал STR вложен в сигналы кода). В результате, если мы будем разрешать выходной сигнал нашей схемы Вых. 1 таким сигналом STR с помощью элемента 2И-НЕ, то получим сигнал Вых. 2, свободный от паразитных импульсов и периодов неопределенности.

Подробнее о синхронизации будет рассказано в следующих главах книги.

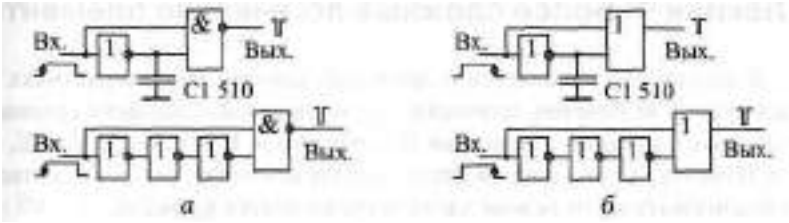


Рис. 2.26. Формирователи коротких импульсов по фронту входного сигнала

Однако бывают случаи, когда указанная особенность элементов И, И-НЕ, ИЛИ, ИЛИ-НЕ формировать короткие импульсы при изменении входных сигналов оказывается очень полезной. Например, нам необходимо сформировать короткий импульс по положительному или отрицательному фронту имеющегося сигнала. Тогда этот сигнал инвертируют, специально задерживают с помощью цепочки элементов или емкости и подают исходный сигнал и задержанный сигнал на входы элемента (рис. 2.26).

Импульс по положительному фронту входного сигнала формируется на элементе 2И или 2И-НЕ (*а*), а импульс по отрицательному фронту входного сигнала — на элементе 2ИЛИ или 2ИЛИ-НЕ (*б*). Если элемент с инверсией, то выходной импульс будет отрицательным, если без инверсии, то положительным. При указанной на схемах величине емкости длительность импульса получается около 50 нс. Для увеличения длительности импульса надо увеличивать величину емкости или же количество инверторов в цепи задержки (при этом количество инверторов обязательно должно быть нечетным).

Лекция 4. Более сложные логические элементы

В лекции рассказывается о принципах работы, характеристиках и типовых схемах включения логических элементов, выполняющих сравнительно сложные функции — элементов Исключающее ИЛИ, И-ИЛИ-НЕ, триггеров Шмитта, а также приводятся схемотехнические решения, позволяющие реализовать на их основе часто встречающиеся функции.

Ключевые слова: Исключающее ИЛИ, суммирование по модулю 2, управляемый инвертор, выделение фронтов, И-ИЛИ-НЕ, гистерезис, формирователь импульса сброса, управляемый генератор, устранение дребезга контактов.

2.4. Элементы Исключающее ИЛИ

Элементы Исключающее ИЛИ (по-английски — Exclusive-OR) также можно было бы отнести к простейшим элементам, но функция, выполняемая ими, несколько сложнее, чем в случае элемента И или элемента ИЛИ. Все входы элементов Исключающее ИЛИ равноправны, однако ни один из входов не может заблокировать другие входы, установив выходной сигнал в уровень единицы или нуля.

Таблица 2.5. Таблица истинности элемента Исключающее ИЛИ

Вход 1	**Вход** 2	Выход
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

Под функцией Исключающее ИЛИ понимается следующее: единица на выходе появляется тогда, когда только на одном входе присутствует единица. Если единиц на входах две или больше, или если на всех входах нули, то на выходе будет нуль. Таблица истинности двухвходового элемента Исключающее ИЛИ приведена в табл. 2.5. Обозначения, принятые в отечественных и зарубежных схемах, показаны на рис. 2.27. Надпись на отечественном обозначении элемента Исключающее ИЛИ «=1» как раз и обозначает, что выделяется ситуация, когда на входах одна и только одна единица.

Элементов Исключающее ИЛИ в стандартных сериях немного. Отечественные серии предлагают микросхемы ЛП5 (четыре двухвходовых элемен-

Рис. 2.27. Обозначения элементов Исключающее ИЛИ: зарубежные (слева) и отечественные (справа)

та с выходом 2С), ЛЛ3 и ЛП12, отличающиеся от ЛП5 выходом ОК. Слишком уж специфическая функция реализуется этими элементами.

С точки зрения математики, элемент Исключающее ИЛИ выполняет операцию так называемого суммирования по модулю 2. Поэтому эти элементы также называются сумматорами по модулю два. Как уже отмечалось в предыдущей главе, обозначается суммирование по модулю 2 знаком плюса, заключенного в кружок.

Основное применение элементов Исключающее ИЛИ, прямо следующее из таблицы истинности, состоит в сравнении двух входных сигналов. В случае, когда на входы приходят две единицы или два нуля (сигналы совпадают), на выходе формируется нуль (см. табл. 2.5). Обычно при таком применении на один вход элемента подается постоянный уровень, с которым сравнивается изменяющийся во времени сигнал, приходящий на другой вход. Но значительно чаще для сравнения сигналов и кодов применяются специальные микросхемы компараторов кодов, которые будут рассмотрены в следующей главе.

В качестве сумматора по модулю 2 элемент Исключающее ИЛИ используется также в параллельных и последовательных делителях по модулю 2, служащих для вычисления циклических контрольных сумм. Но подробно эти схемы будут рассмотрены в восьмой главе книги.

Важное применение элементов Исключающее ИЛИ — это управляемый инвертор (рис. 2.28). В этом случае один из входов элемента используется в качестве управляющего, а на другой вход элемента поступает информационный сигнал. Если на управляющем входе единица, то входной сигнал инвертируется, если же нуль — не инвертируется. Чаще всего управляющий сигнал задается постоянным уровнем, определяя режим работы

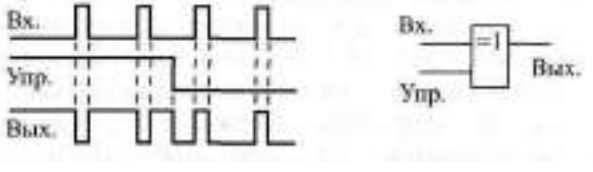


Рис. 2.28. Элемент Исключающее ИЛИ как управляемый инвертор

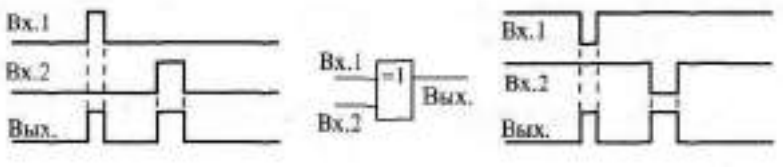


Рис. 2.29. Применение элемента Исключающее ИЛИ для смешивания двух неодновременных сигналов

элемента, а информационный сигнал является импульсным. То есть элемент Исключающее ИЛИ может изменять полярность входного сигнала или фронта, а может и не изменять в зависимости от управляющего сигнала.

В случае, когда имеется два сигнала одинаковой полярности (положительные или отрицательные), и при этом их одновременный приход исключается, элемент Исключающее ИЛИ может быть использован для смешивания этих сигналов (рис. 2.29). При любой полярности входных сигналов выходные сигналы элемента будут положительными. При положительных входных сигналах элемент Исключающее ИЛИ будет работать как элемент 2ИЛИ, а при отрицательных он будет заменять элемент 2И-НЕ. Такие замены могут быть полезны в тех случаях, когда в схеме остаются неиспользованными некоторые элементы Исключающее ИЛИ. Правда, при этом надо учитывать, что задержка распространения сигнала в элементе Исключающее ИЛИ обычно несколько больше (примерно в 1,5 раза), чем задержка в простейших элементах И, И-НЕ, ИЛИ, ИЛИ-НЕ.

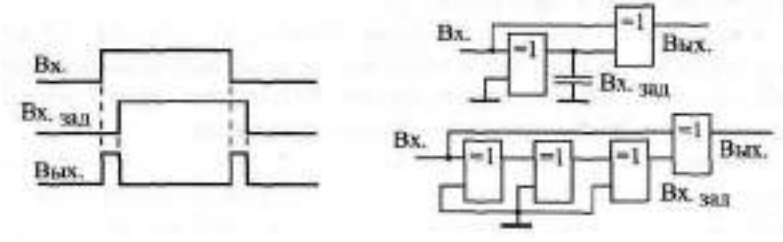


Рис. 2.30. Выделение фронтов входного сигнала с помощью элемента Исключающее ИЛИ

Еще одно важнейшее применение элемента Исключающее ИЛИ — формирование коротких импульсов по любому фронту входного сигнала (рис. 2.30). В данном случае не важно, положительный фронт входного сигнала или отрицательный, на выходе все равно формируется положительный импульс. Входной сигнал задерживается с помощью конденсатора или цепрчки элементов, а затем исходный сигнал и его задержанная копия поступают на входы элемента Исключающее ИЛИ. В обеих схемах в ка-

честве элементов задержки используются также двувходовые элементы Исключающее ИЛИ в неинвертирующем включении (на неиспользуемый вход подается нуль). В результате такого преобразования можно говорить об удвоении частоты входного сигнала, так как выходные импульсы следуют вдвое чаще, чем входные.

Данную особенность элементов Исключающее ИЛИ надо учитывать в том случае, когда на оба входа элемента поступают изменяющиеся одновременно сигналы. При этом на выходе элемента возможно появление коротких паразитных импульсов по любому из фронтов входных сигналов. Исключить их влияние на дальнейшую схему можно, например, с помощью синхронизации, подобной рассмотренной в предыдущем разделе.

2.5. Сложные логические элементы

Помимо простейших логических элементов, рассмотренных в предыдущих разделах, в состав стандартных серий входит и несколько более сложных логических элементов. Они представляют собой несложную комбинацию из простейших логических элементов. От более сложных комбинационных микросхем, которым будет посвящена следующая глава, эти элементы отличаются именно очевидной сводимостью к простейшим элементам. Поэтому в справочниках обычно даже не приводятся таблицы истинности этих элементов.

Типичный пример сложного логического элемента — ЛР1. В корпусе микросхемы содержится два элемента, каждый из которых представляет собой комбинацию из двух элементов 2И и одного элемента 2ИЛИ-НЕ (рис. 2.31). По такому же принципу строятся и другие микросхемы ЛР. Разница между ними только в количестве элементов И и в количестве входов этих элементов (рис. 2.32). Некоторые из микросхем ЛР (ЛР1, ЛР3) допускают подключение к специальным входам микросхем расширителей ЛД, хотя такое расширение применяется на практике довольно редко. Микросхема ЛР10 отличается от ЛР9 выходом ОК.

На рис. 2.33 приведено несколько примеров наиболее типичных применений микросхемы ЛР1. Самое распространенное ее использование (*а*) состоит в организации двухканального мультиплексирования, то есть в переключении сигналов с двух входов на один выход. При этом один

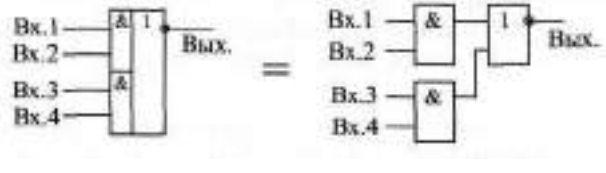


Рис. 2.31. Логический элемент ЛР1 и его эквивалентная схема

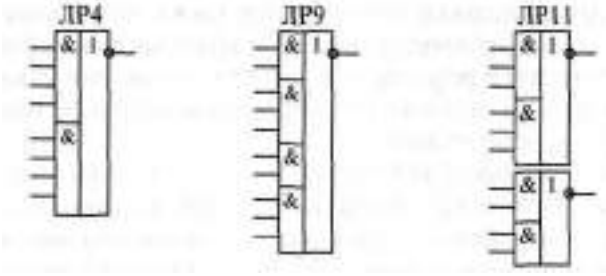


Рис. 2.32. Примеры логических элементов ЛР

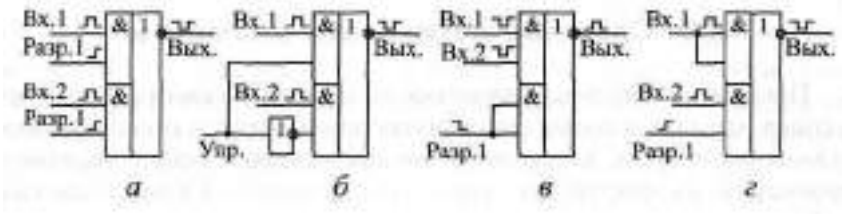


Рис. 2.33. Примеры использования элементов ЛР1

из входов каждого из элементов 2И используется в качестве информационного, а другой — в качестве разрешающего. Вариант этого включения *(б)* — использование одного управляющего входа переключения каналов и дополнительного инвертора. При единице на управляющем входе работает верхний канал, при нуле — нижний. Еще один вариант использования элемента ЛР1 *(в)* — смешивание двух отрицательных входных сигналов с возможностью разрешения/запрета выходного сигнала. Наконец, последний показанный на рисунке вариант *(г)* — смешивание двух положительных сигналов, один из которых может быть разрешен или запрещен. То есть такое объединение в одном элементе функций И и ИЛИ довольно удобно.

На других элементах ЛР можно строить более сложные схемы. Например, элемент ЛР9 позволяет построить четырехканальный мультиплексор, так как в его структуре четыре элемента И и элемент 4ИЛИ-НЕ. Однако в большинстве случаев применение элементов ЛР для мультиплексирования оказывается не слишком удобным, так как в стандартных сериях имеются специальные микросхемы мультиплексоров с более удобным управлением.

При необходимости элементы ЛР1 могут использоваться в качестве более простых элементов 2И-НЕ и 2ИЛИ-НЕ (рис. 2.34). Элемент 2ИЛИ-НЕ получается при попарном объединении входов. Элемент 2И-НЕ получается

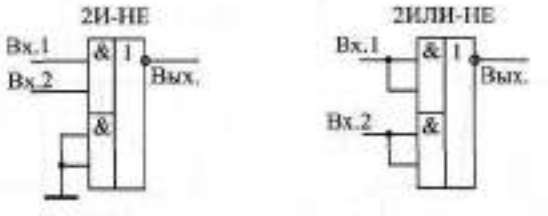


Рис. 2.34. Использование элементов ЛР в качестве элементов 2И-НЕ и 2ИЛИ-НЕ

при отключении половины схемы путем подачи нулей на два входа. При желании можно, конечно, свести элемент ЛР даже к простому инвертору, но это, наверное, уже недопустимая роскошь.

К сложным логическим элементам, помимо ЛР, можно отнести также и элементы И-НЕ с выходом ЗС (например, ЛА17 - 4И-НЕ, ЛА19 - 12И-НЕ). Их можно рассматривать как комбинацию обычного элемента И-НЕ и выходного буфера с выходом ЗС. Наличие дополнительного управляющего входа EZ и выход ЗС создают принципиально новые возможности применения этих элементов. Например, их можно использовать для работы на мультиплексированную или двунаправленную линию, при этом они еще и выполняют функцию И-НЕ над входными сигналами. Но на практике значительно чаще элемент ЛА19 используют как самый обычный элемент 12И-НЕ с выходом 2С, для чего на управляющий вход EZ постоянно подается сигнал логического нуля.

Среди элементов И, ИЛИ, ИЛИ-НЕ элементы с выходом ЗС отсутствуют.

2.6. Триггеры Шмитта

Триггеры Шмитта представляют собой специфические логические элементы, специально рассчитанные на работу с входными аналоговыми сигналами. Они предназначены для преобразования входных аналоговых сигналов в выходные цифровые сигналы. Появление таких микросхем связано в первую очередь с необходимостью восстановления формы цифровых сигналов, искаженных в результате прохождения по линиям связи. Фронты таких сигналов оказываются пологими, в результате чего форма сигналов вместо прямоугольной может стать близкой к треугольной или синусоидальной. К тому же сигналы, передаваемые на большие расстояния, сильно искажаются шумами и помехами. Восстановить их форму в исходном виде, устранить влияние помех и шумов как раз и призваны триггеры Шмитта.

На первом и втором уровнях представления (логическая модель и модель с временными задержками) триггеры Шмитта представляют со-

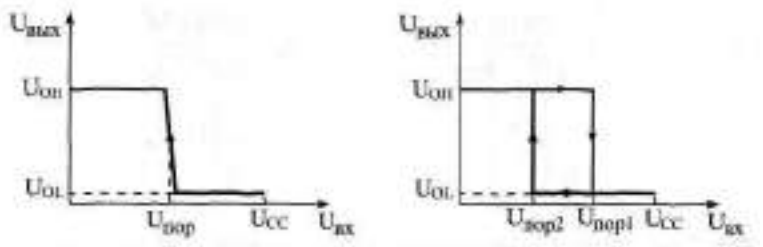


Рис. 2.35. Передаточные характеристики обычного инвертора и триггера Шмитта с инверсией

бой обычные логические элементы, которые с определенной задержкой распространения выполняют логическую функцию над входными цифровыми сигналами. Но на третьем уровне представления их отличие от обычных логических элементов очень существенно.

Если построить график зависимости выходного напряжения элемента от входного (передаточную характеристику), то для триггера Шмитта он будет гораздо сложнее, чем для обычного элемента (рис. 2.35).

В случае обычного элемента с инверсией (*а*) при входных напряжениях ниже определенного порога срабатывания $U_{пор}$ выходной сигнал имеет высокий уровень, а при входных напряжениях выше этого порога $U_{пор}$ — низкий уровень. При этом не имеет значения, возрастает входное напряжение или убывает.

А в случае триггера Шмитта с инверсией (*б*) принципиально как раз направление изменения сигнала. При возрастании входного сигнала от нуля до напряжения питания порог срабатывания будет одним ($U_{пор1}$), а при уменьшении сигнала от напряжения питания до нуля — другим ($U_{пор2}$), причем $U_{пор1} > U_{пор2}$ В результате на графике образуется своеобразная петля. Выходной сигнал как бы запаздывает переключаться при возврате входного к исходному уровню. Это называется эффектом *гистерезиса* (запаздывания).

Наличие гистерезиса приводит к тому, что любой шум, любые помехи с амплитудой, меньшей величины ($U_{пор1} - U_{пор2}$), отсекаются, а любые фронты входного сигнала, даже самые пологие, преобразуются в крутые фронты выходного сигнала. Главное — чтобы амплитуда входного сигнала была большей, чем ($U_{пор1} - U_{пор2}$). На рис. 2.36 показано, как будет реагировать на сигнал с пологими фронтами и с шумами обычный инвертор и триггер Шмитта с инверсией.

В стандартные серии цифровых микросхем входят триггеры Шмитта, представляющие собой инверторы (ТЛ2 — 6 инверторов), элементы 2И-НЕ (ТЛ3 — 4 элемента) и элементы 4И-НЕ (ТЛ1 — 2 элемента). Пороговые напряжения составляют для всех этих микросхем около 1,7 В ($U_{пор1}$)

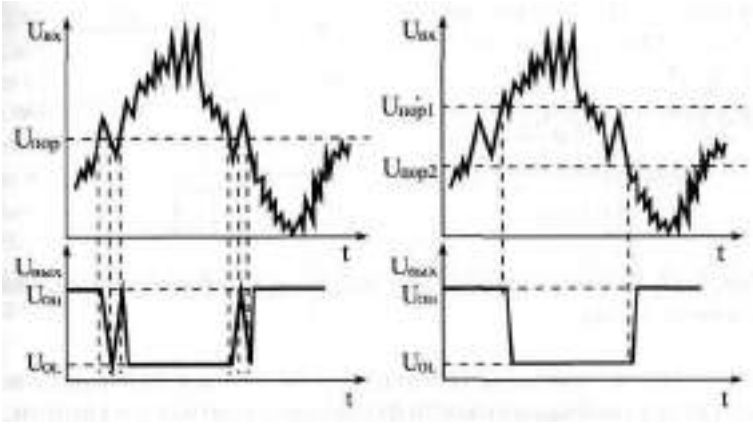


Рис. 2.36. Реакция на искаженный входной сигнал инвертора (слева) и триггера Шмитта с инверсией (справа)

и около 0,9 В ($U_{пор2}$). Графическое обозначение триггера Шмитта представляет собой упрощенное изображение его передаточной характеристики с гистерезисом (рис. 2.37).

Наиболее распространенное применение триггеров Шмитта — это формирователь сигнала начального сброса по включению питания схемы. Необходимость такого сигнала сброса вызвана тем, что при включении питания выходные сигналы сложных микросхем, имеющих внутреннюю память (например, регистров, счетчиков), могут принимать произвольные значения, что не всегда удобно. Привести их в необходимое состояние (чаще всего — установить их в нуль) как раз и призван сигнал начального сброса.

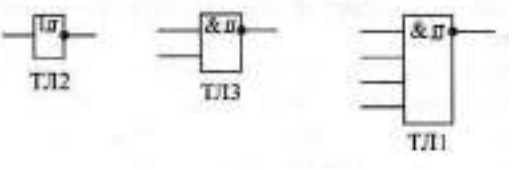


Рис. 2.37. Триггеры Шмитта

Для формирования сигнала начального сброса используется простая RC-цепочка, причем конденсатор берется с большой емкостью. Напряжение на конденсаторе при включении питания нарастает медленно, в результате чего на выходе триггера Шмитта формируется положительный импульс (рис. 2.38). Использовать для этого обычный инвертор не рекомендуется.

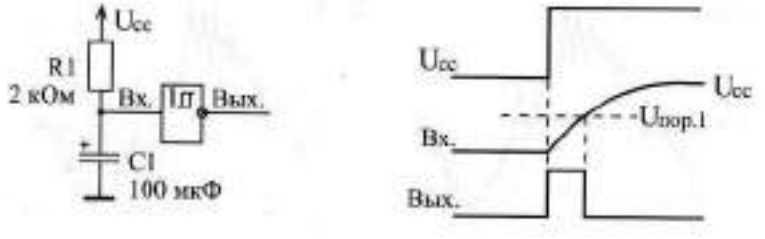


Рис. 2.38. Формирователь импульса начальной установки по включению питания

Точно так же триггеры Шмитта рекомендуется применять во всех случаях, когда с помощью емкости формируется сигнал с пологими, затянутыми фронтами. В отличие от обычных логических элементов, триггеры Шмитта всегда обеспечивают надежную и стабильную работу. Правда, надо учитывать, что триггеры Шмитта имеют несколько большую задержку, чем обычные логические элементы.

Еще одно применение использование триггера Шмитта состоит в построении генераторов импульсов. В отличие от генераторов на обычных инверторах (см. раздел 2.1), в данном случае схема получается гораздо проще: достаточно всего лишь одного инвертирующего триггера Шмитта, одного резистора (порядка сотен Ом) и одного конденсатора (рис. 2.39). При этом очень удобно, что конденсатор одним выводом присоединен к общему проводу, к «земле». Это позволяет применять электролитические конденсаторы большой емкости, а также переменные конденсаторы. Использование двухвходовых триггеров Шмитта дает возможность легко разрешать или запрещать генерацию с помощью управляющего сигнала Разр. При уровне логической единицы на входе Разр. генерация идет, при уровне логического нуля генерации — нет.

Нестандартные триггеры Шмитта можно строить также на основе самых обычных логических элементов с обратной связью через резисторы. При этом путем подбора величин этих резисторов можно выбирать значения пороговых напряжений триггера Шмитта.

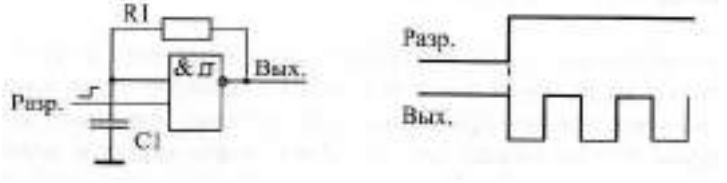


Рис. 2.39. Управляемый генератор на триггере Шмитта

Для примера на рис. 2.40 показана схема триггера Шмитта на инверторах, которая работает с входными сигналами, симметричными относительно нулевого уровня. Такие сигналы могут быть, например, в передающем кабеле с трансформаторной развязкой. В данном случае триггер Шмитта не только позволяет восстановить искаженную форму сигнала, но еще и усиливает сигнал, а также сдвигает его уровни до значений стандартных нуля и единицы.

Но чаще всего вполне хватает возможностей стандартных триггеров Шмитта, которые не требуют включения внешних элементов и имеют гарантированные характеристики.

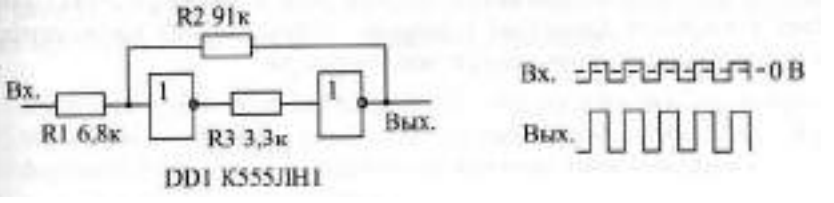


Рис. 2.40. Триггер Шмитта, построенный на обычных логических элементах

Наконец, последнее применение триггеров Шмитта, которое мы здесь рассмотрим, состоит в подавлении так называемого дребезга контактов. Дело в том, что любой механический контакт (в кнопках, тумблерах, переключателях и т. д.) не замыкается и не размыкается сразу, мгновенно. Любое замыкание и размыкание сопровождается несколько-

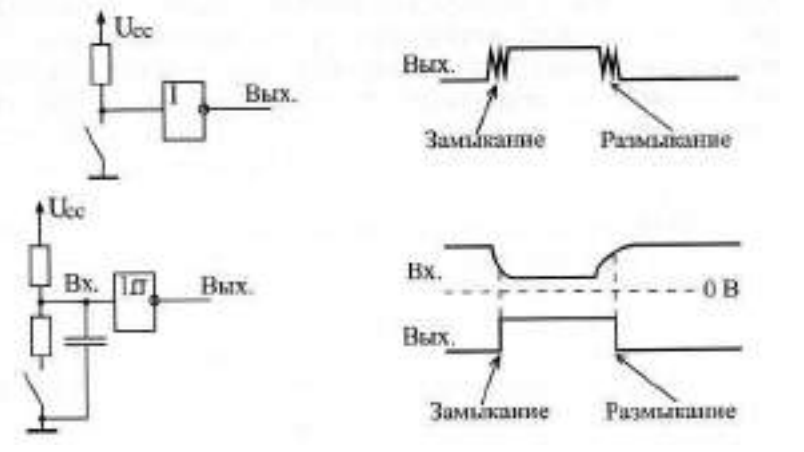


Рис. 2.41. Дребезг контактов (вверху) и его подавление с помощью триггера Шмитта (внизу)

ми быстрыми замыканиями и размыканиями, приводящими к появлению паразитных коротких импульсов, которые могут нарушить работу дальнейшей цифровой схемы. Триггер Шмитта с RC-цепочкой на входе позволяет устранить этот эффект (рис. 2.41).

Конденсатор заряжается и разряжается довольно медленно, в результате чего короткие импульсы подавляются и не проходят на выход триггера Шмитта. Номинал верхнего по схеме резистора должен в данном случае быть в 6–7 раз больше номинала нижнего, чтобы резистивный делитель при замкнутом тумблере давал на входе триггера Шмитта уровень логического нуля. Сопротивления резисторов должны быть порядка сотен Ом — единиц килоОм. Емкость конденсатора может выбираться в широком диапазоне и зависит от того, какова продолжительность дребезга контактов конкретного тумблера.

Глава 3. Применение комбинационных микросхем

Лекция 5. Комбинационные микросхемы. Часть 1

В лекции рассказывается о комбинационных микросхемах: шифраторах, дешифраторах, мультиплексорах и компараторах кодов, об их алгоритмах работы, параметрах, типовых схемах включения, а также о реализации на их основе некоторых часто встречающихся функций.

Ключевые слова: шифратор, дешифратор, мультиплексор, компаратор кодов, увеличение количества разрядов, демультиплексор, стробирование, увеличение количества каналов, неопределенные состояния выхода, каскадирование.

Комбинационные микросхемы выполняют более сложные функции, чем простые логические элементы. Их входы объединены в функциональные группы и не являются полностью взаимозаменяемыми. Например, любые два входа логического элемента И-НЕ совершенно спокойно можно поменять местами, от этого выходной сигнал никак не изменится, а для комбинационных микросхем это невозможно, так как у каждого входа — своя особая функция.

Объединяет комбинационные микросхемы с логическими элементами то, что они не имеют внутренней памяти. То есть уровни их выходных сигналов всегда однозначно определяются текущими уровнями входных сигналов и никак не связаны с предыдущими значениями входных сигналов. Любое изменение входных сигналов обязательно изменяет состояние выходных сигналов. Именно поэтому логические элементы иногда также называют комбинационными микросхемами, в отличие от последовательных (или последовательностных) микросхем, которые имеют внутреннюю память и управляются не уровнями входных сигналов, а их последовательностями.

Строго говоря, все комбинационные микросхемы внутри построены из простейших логических элементов, и эта их внутренняя структура часто приводится в справочниках. Но для разработчика цифровой аппаратуры эта информация обычно лишняя, ему достаточно знать только таблицу истинности, только принцип преобразования входных сигналов в выходные, а также величины задержек между входами и выходами и уровни входных и выходных токов и напряжений. Внутренняя

же структура важна для разработчиков микросхем, а также в тех редчайших случаях, когда надо построить новую комбинационную микросхему из микросхем простых логических элементов.

Состав набора комбинационных микросхем, входящих в стандартные серии, был определен исходя из наиболее часто встречающихся задач. Требуемые для этого функции реализованы в комбинационных микросхемах наиболее оптимально, с минимальными задержками и минимальным потреблением мощности. Поэтому пытаться повторить эту уже проделанную однажды работу не стоит. Надо просто уметь грамотно применять то, что имеется.

3.1. Дешифраторы и шифраторы

Функции дешифраторов и шифраторов понятны из их названий. Дешифратор преобразует входной двоичный код в номер выходного сигнала (дешифрирует код), а шифратор преобразует номер входного сигнала в выходной двоичный код (шифрует номер входного сигнала). Количество выходных сигналов дешифратора и входных сигналов шифратора равно количеству возможных состояний двоичного кода (входного кода у дешифратора и выходного кода у шифратора), то есть 2^n, где n — разрядность двоичного кода (рис. 3.1). Микросхемы дешифраторов обозначаются на схемах буквами DC (от английского Decoder), а микросхемы шифраторов — CD (от английского Coder).

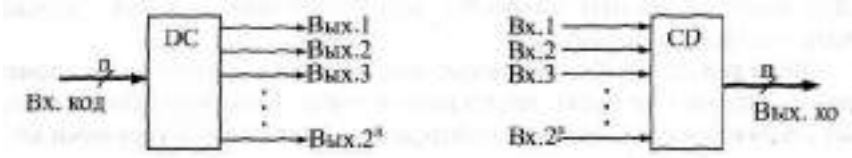


Рис. 3.1. Функции дешифратора (слева) и шифратора (справа)

На выходе дешифратора всегда присутствует только один сигнал, причем номер этого сигнала однозначно определяется входным кодом. Выходной код шифратора однозначно определяется номером входного сигнала.

Рассмотрим подробнее функцию дешифратора.

В стандартные серии входят дешифраторы на 4 выхода (2 разряда входного кода), на 8 выходов (3 разряда входного кода) и на 16 выходов (4 разряда входного кода). Они обозначаются соответственно как 2-4, 3-8, 4-16. Различаются микросхемы дешифраторов входами управления (разрешения/запрета выходных сигналов), а также типом выхода: 2С или ОК. Выходные сигналы всех дешифраторов имеют отрицательную поляр-

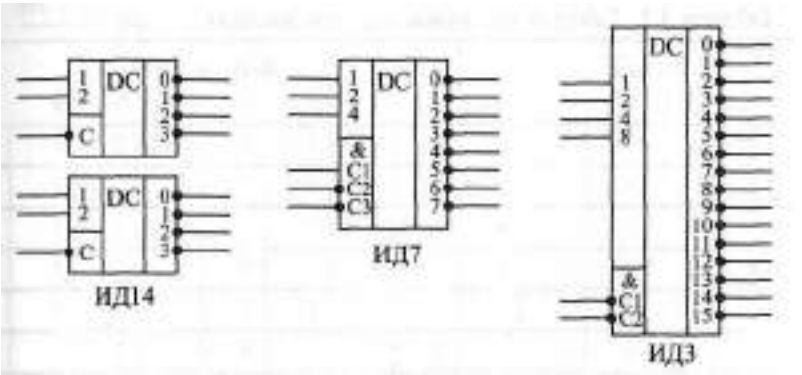


Рис. 3.2. Примеры микросхем дешифраторов

ность. Входы, на которые поступает входной код, называют часто адресными входами. Обозначают эти входы 1, 2, 4, 8, где число соответствует весу двоичного кода (1 — младший разряд, 2 — следующий разряд и т. д.), или А0, А1, А2, А3. В отечественных сериях микросхемы дешифраторов обозначаются буквами ИД. На рис. 3.2 показаны три наиболее типичных микросхемы дешифраторов.

Код на входах 1, 2, 4, 8 определяет номер активного выхода (вход 1 соответствует младшему разряду кода, вход 8 — старшему разряду кода). Входы разрешения С1, С2, С3 объединены по функции И и имеют указанную на рисунке полярность. Для примера в табл. 3.1 приведена таблица истинности дешифратора ИД7 (3-8). Существуют и дешифраторы 4-10 (например, ИД6), которые обрабатывают не все возможные 16 состояний входного кода, а только первые 10 из них.

Первые три строки таблицы соответствуют запрету выходных сигналов. Разрешением выхода будет единица на входе С1 и нули на входах С2 и С3. Символ "Х" обозначает безразличное состояние данного входа (неважно, нуль или единица). Нижние восемь строк соответствуют разрешению выходных сигналов. Номер активного выхода (на котором формируется нулевой сигнал) определяется кодом на входах 1, 2, 4, причем вход 1 соответствует младшему разряду кода, а вход 4 — старшему разряду кода.

Наиболее типичное применение дешифраторов состоит именно в дешифрировании входных кодов, при этом входы С используются как стробирующие, управляющие сигналы. Номер активного (то есть нулевого) выходного сигнала показывает, какой входной код поступил. Если нужно дешифровать код с большим числом разрядов, то можно объединить несколько микросхем дешифраторов (пример показан на рис. 3.3).

При этом старшие разряды кода подаются на основной дешифратор, выходы которого разрешают работу нескольких дополнительных дешиф-

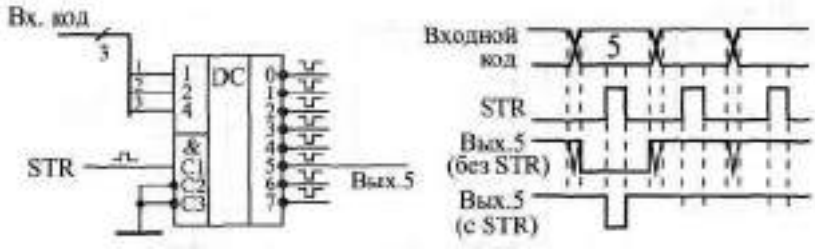


Рис. 3.6. Стробирование выходных сигналов дешифратора

бирующих входов С, то оставшиеся входы С можно использовать в качестве разрешающих работу дешифратора.

Как и для любых других цифровых микросхем, для дешифраторов наиболее критична ситуация одновременного или почти одновременного изменения входных сигналов. Например, если стробы С постоянно разрешают работу дешифратора, то в момент изменения входного кода на любом выходе дешифратора могут появиться паразитные отрицательные короткие импульсы. Это может быть связано как с неодновременным выставлением разрядов кода (из-за несовершенства микросхем источников кода или из-за разных задержек распространения по линиям связи), так и с внутренними задержками самих микросхем дешифраторов.

Если такие паразитные импульсы нужно исключить, то можно применять синхронизацию с помощью стробирующих сигналов. Используемый для этого сигнал С должен начинаться *после* текущего изменения кода, а заканчиваться *до* следующего изменения кода, то есть должен быть реализован вложенный цикл. На рис. 3.6 показано, как будет выглядеть выходной сигнал дешифратора без стробирования и со стробированием.

На втором уровне представления (модель с временными задержками) также надо учитывать, что задержки дешифратора больше задержки простых логических элементов примерно вдвое для входного кода и примерно в полтора раза — для стробирующих входов. То есть если попытаться заменить дешифратор схемой на логических элементах, то такой дешифратор получится медленнее. Точные величины задержек надо смотреть в справочниках.

Дешифраторы, имеющие выходы типа ОК (ИД5, ИД10), удобно применять в схемах позиционной индикации на светодиодах. На рис. 3.7 приведен пример такой индикации на микросхеме ИД5, которая представляет собой два дешифратора 2-4 с объединенными входами для подачи кода и стробами, позволяющими легко строить дешифратор 3-8. При этом старший разряд кода выбирает один из дешифраторов 2-4 (нуль соответствует верхнему по схеме дешифратору, а единица — нижнему). То есть в данном случае номер горящего светодиода равен входному коду дешифратора. Такая индикация называется позиционной.

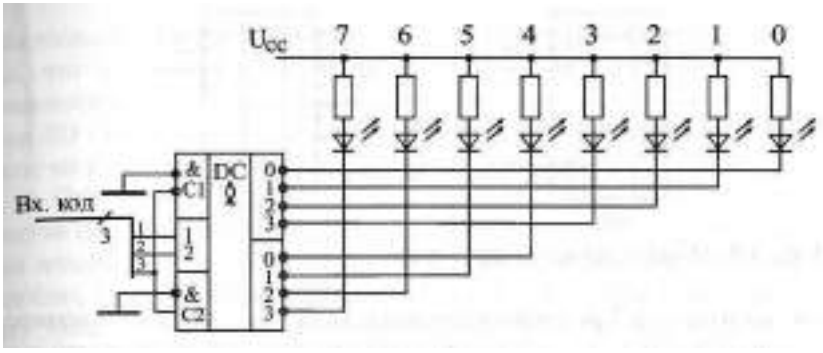


Рис. 3.7. Позиционная индикация на дешифраторе с выходами ОК

Выходы микросхем дешифраторов с ОК можно объединять между собой для реализации проводного **ИИ** (рис. 3.8). Нуль на объединенном выходе будет тогда, когда хотя бы на одном из выходов вырабатывается нуль. При равномерном пошаговом наращивании входного кода (например, с помощью счетчика) такое решение позволяет формировать довольно сложные последовательности выходных сигналов. Правда, каждый выход дешифратора может использоваться для получения только одного выходного сигнала. Это ограничивает возможности таких схем.

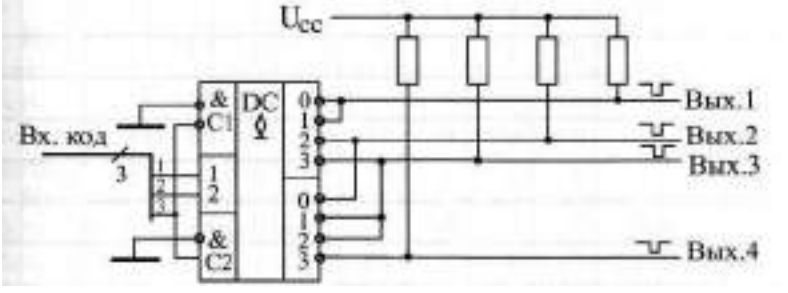


Рис. 3.8. Объединение выходов дешифраторов с ОК

Шифраторы используются гораздо реже, чем дешифраторы. Это связано с более специфической областью их применения. Значительно меньше и выбор микросхем шифраторов в стандартных сериях. В отечественных сериях шифраторы имеют в названии буквы ИВ.

На рис. 3.9 показаны для примера две микросхемы шифраторов ИВ1 и ИВ3. Первая имеет 8 входов и 3 выхода (шифратор 8-3), а вторая — 9 входов и 4 выхода (шифратор 9-4). Все входы шифраторов — инверсные (активные входные сигналы — нулевые). Все выходы тоже инверсные, то есть формируется инверсный код. Микросхема ИВ1, помимо 8 инфор-

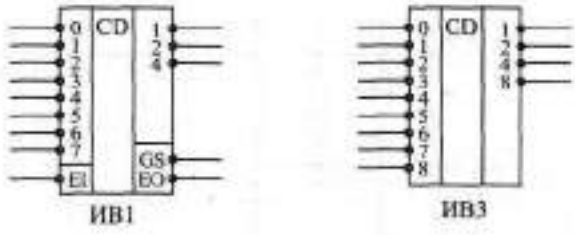


Рис. 3.9. Микросхемы шифраторов

мационных входов и 3 разрядов выходного кода (1, 2, 4), имеет инверсный вход разрешения —EI, выход признака прихода любого входного сигнала —GS, а также выход переноса —EO, позволяющий объединять несколько шифраторов для увеличения разрядности.

Таблица истинности шифратора ИВ1 приведена в табл. 3.2.

Таблица 3.2. Таблица истинности шифратора ИВ1

Входы									Выходы				
-EI	0	1	2	3	4	5	6	7	-GS	4	2	1	-EO
1	X	X	X	X	X	X	X	X	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	X	X	X	X	X	X	X	0	0	0	0	0	1
0	X	X	X	X	X	X	0	1	0	0	0	1	1
0	X	X	X	X	X	0	1	1	0	0	1	0	1
0	X	X	X	X	0	1	1	1	0	0	1	1	1
0	X	X	X	0	1	1	1	1	0	1	0	0	1
0	X	X	0	1	1	1	1	1	0	1	0	1	1
0	X	0	1	1	1	1	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	1	0	1	1	1	1

Из таблицы видно, что на выходах кода 1,2,4 формируется инверсный двоичный код номера входной линии, на который приходит отрицательный входной сигнал. При одновременном поступлении нескольких входных сигналов формируется выходной код, соответствующий входу с наибольшим номером, то есть старшие входы имеют приоритет перед младшими. Поэтому такой шифратор называется приоритетным. При отсутствии входных сигналов (вторая строчка таблицы) формируется выходной код 111. Единичный сигнал -EI (первая строчка) запрещает работу шифратора (все выходные сигналы устанавливаются в единицу).

Да выходе -GS вырабатывается нуль при приходе любого входного сигнала, что позволяет, в частности, отличить ситуацию прихода нулевого входного сигнала от ситуации отсутствия любых входных сигналов. Выход EO становится активным (нулевым) при отсутствии входных сигналов, но разрешении работы шифратора сигналом -EI.

Стандартное применение шифраторов состоит в сокращении количества сигналов. Например, в случае шифратора ИВ1 информация о восьми входных сигналах сворачивается в три выходных сигнала. Это очень удобно, например, при передаче сигналов на большие расстояния. Правда, входные сигналы не должны приходить одновременно. На рис. 3.10 показаны стандартная схема включения шифратора и временные диаграммы его работы.

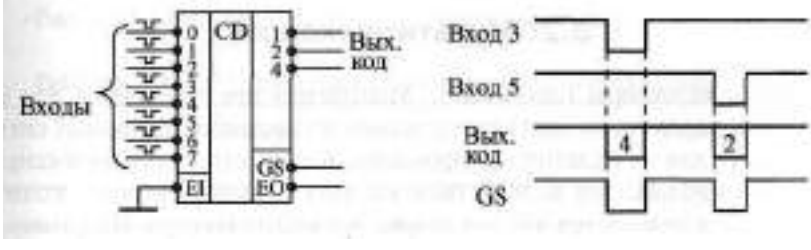


Рис. 3.10. Стандартное включение шифратора

Инверсия выходного кода приводит к тому, что при приходе нулевого входного сигнала на выходе формируется не нулевой код, а код 111, то есть 7. Точно так же при приходе, например, третьего входного сигнала на выходе образуется код 100, то есть 4, а при приходе пятого выходного сигнала — код 010, то есть 2.

Наличие у шифраторов входов EI и E0 позволяет увеличивать количество входов и разрядов шифратора, правда, с помощью дополнительных элементов на выходе. На рис. 3.11 показан пример построения шифратора 16-4 на двух микросхемах шифраторов ИВ1 и трех элементах 2И-НЕ (ЛА3).

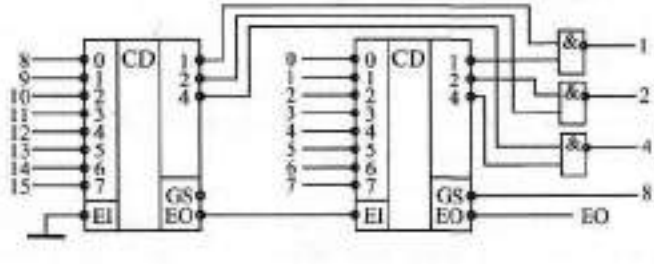


Рис. **3.11.** Шифратор 16-4 на двух шифраторах 8-3

Одновременное или почти одновременное изменение сигналов на входе шифратора приводит к появлению периодов неопределенности на выходах. Выходной код может на короткое время принимать значение, не соответствующее ни одному из входных сигналов. Поэтому в тех случаях, когда входные сигналы могут приходить одновременно, необходима синхронизация выходного кода, например, с помощью разрешающего сигнала EI, который должен приходить только тогда, когда состояние неопределенности уже закончилось.

Задержка шифратора от входа до выхода кода примерно в полтора раза превышает задержку логического элемента, а задержка до выхода GS — примерно в два раза больше. Точные величины задержек микросхем надо смотреть в справочниках.

3.2. Мультиплексоры

Мультиплексоры (английское Multiplexer) предназначены для поочередной передачи на один выход одного из нескольких входных сигналов, то есть для их мультиплексирования. Количество мультиплексируемых входов называется количеством каналов мультиплексора, а количество выходов называется числом разрядов мультиплексора. Например, 2-канальный 4-разрядный мультиплексор имеет 4 выхода, на каждый из которых может передаваться один из двух входных сигналов. А 4-канальный 2-разрядный мультиплексор имеет 2 выхода, на каждый из которых может передаваться один из четырех входных сигналов. Число каналов мультиплексоров, входящих в стандартные серии, составляет от 2 до 16, а число разрядов — от 1 до 4, причем чем больше каналов имеет мультиплексор, тем меньше у него разрядов.

Управление работой мультиплексора (выбор номера канала) осуществляется с помощью входного кода адреса. Например, для 4-канального мультиплексора необходим 2-разрядный управляющий (адресный) код, а для 16-канального — 4-разрядный код. Разряды кода обозначаются 1, 2, 4, 8 или A0, A1, A2, A3. Мультиплексоры бывают с выходом 2С и с выходом 3С. Выходы мультиплексоров бывают прямыми и инверсными. Выход 3С позволяет объединять выходы мультиплексоров с выходами других микросхем, а также получать двунаправленные и мультиплексированные линии. Некоторые микросхемы мультиплексоров имеют вход разрешения/запрета С (другое обозначение — S), который при запрете устанавливает прямой выход в нулевой уровень.

На рис. 3.12 показаны для примера несколько микросхем мультиплексоров из состава стандартных серий. В отечественных сериях мультиплексоры имеют код типа микросхемы КП. На схемах микросхемы мультиплексоров обозначаются буквами MS.

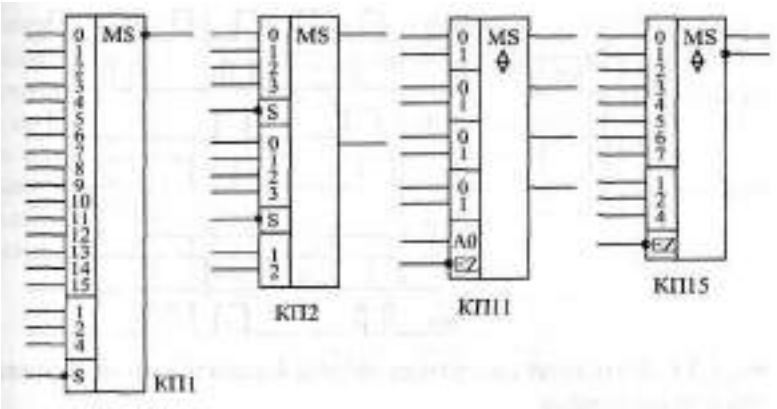


Рис. 3.12. Примеры микросхем мультиплексоров

Таблица 3.3. Таблица истинности 8-канального мультиплексора

Входы				Выходы	
4	2	1	-EZ	Q	-Q
X	X	X	1	Z	Z
0	0	0	0	D0	-D0
0	0	1	0	D1	-D1
0	1	0	0	D2	-D2
0	1	1	0	D3	-D3
1	0	0	0	D4	-D4
1	0	1	0	D5	-D5
1	1	0	0	D6	-D6
1	1	1	0	D7	-D7

В табл. 3.3 в качестве примера приведена таблица истинности одноразрядного 8-канального мультиплексора с выходами ЗС (КП15).

В таблице сигналы на входах 0. . . 7 обозначены D0 . . D7, прямой выход — Q, инверсный выход — -Q, Z — третье состояние выхода. При единице на входе -EZ оба выхода находятся в третьем состоянии. При нуле на входе -EZ выходной сигнал на прямом выходе повторяет состояние входного сигнала, номер которого задается входным кодом на входах 1, 2, 4. Сигнал на инверсном выходе противоположен по полярности сигналу на прямом выходе.

На рис. 3.13 приведена временная диаграмма работы 4-канального мультиплексора. В зависимости от входного кода на выход передается

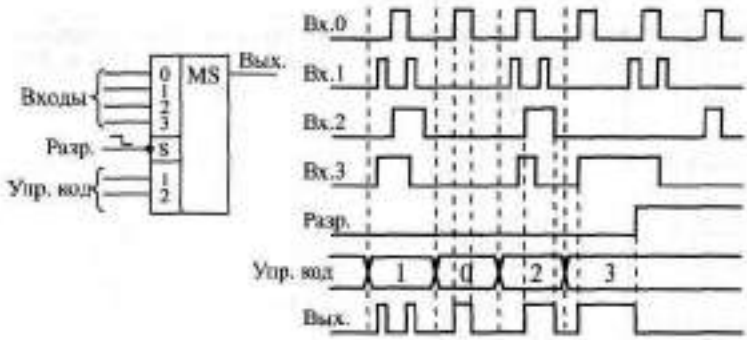


Рис. 3.13. Временная диаграмма работы 4-канального мультиплексора с разрешением

один из четырех входных сигналов. При запрещении работы на выходе устанавливается нулевой сигнал вне зависимости от входных сигналов.

Микросхемы мультиплексоров можно объединять для увеличения количества каналов. Например, два 8-канальных мультиплексора легко объединяются в 16-канальный с помощью инвертора на входах разрешения и элемента 2ИНЕ для смешивания выходных сигналов (рис. 3.14). Старший разряд кода будет при этом выбирать один из двух мультиплексоров. Точно так же из двух 16-канальных мультиплексоров можно сделать 32-канальный. Если нужно большее число каналов, то необходимо вместо инвертора включать дешифратор, на который подаются старшие разряды кода. Выходные сигналы дешифратора будут выбирать один из мультиплексоров.

Состояния неопределенности, сопровождающиеся короткими паразитными импульсами, могут возникать на выходе мультиплексоров при почти одновременном изменении входных сигналов. Здесь возможны две

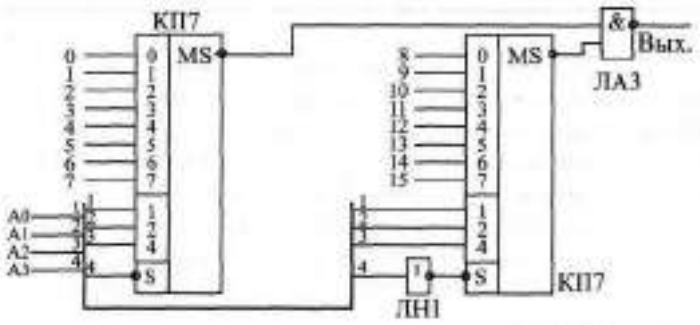


Рис. 3.14. Объединение мультиплексоров для увеличения количества каналов

ситуации. Во-первых, управляющий код может переключаться сразу после изменения передаваемого в данный момент на выход входного сигнала или сразу перед изменением входного сигнала, который будет передавать на выход следующий код. Во-вторых, сигналы (разряды) управляющего кода могут переключаться не одновременно, что приведет к кратковременной передаче на выход входного сигнала, не соответствующего ни одному из значений кода. В любом случае, в момент переключения каналов сигнал на выходе мультиплексора не определен (рис. 3.15).

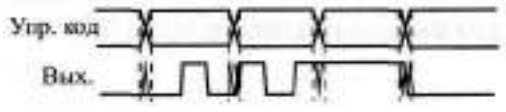


Рис. 3.15. Неопределенные состояния на выходе мультиплексора

Чтобы избежать состояния неопределенности, лучше всего задавать состояние управляющего кода еще до начала работы схемы (до прихода входных сигналов) и в дальнейшем его не менять. Если же это невозможно, то необходима синхронизация, стробирование выходного сигнала, то есть его разрешение только тогда, когда все переходные процессы, связанные с изменением кода, уже закончились. Правда, обычно применять стробирование довольно непросто, так как мультиплексор, как правило, должен без изменений передавать любой входной сигнал.

Задержки выходного сигнала мультиплексора по входам управляющего (адресного) кода примерно в два раза превышают задержки логических элементов, а по информационным входам — примерно в полтора раза. Точные величины задержек надо смотреть в справочниках.

3.3. Компараторы кодов

Микросхемы компараторов кодов (английское Comparator) применяются для сравнения двух входных кодов и выдачи на выходы сигналов о результатах этого сравнения (о равенстве или неравенстве кодов). На схемах компараторы кодов обозначаются двумя символами равенства: "= =". Код типа микросхемы компаратора кода в отечественных сериях - СП.

Примером такой микросхемы может служить СП1 — 4-х разрядный компаратор кодов, сравнивающий величины кодов и выдающий информацию о том, какой код больше, или о равенстве кодов (рис. 3.16).

Помимо восьми входов для сравниваемых кодов (два 4-х разрядных кода, обозначаемых A0...A3 и B0...B3), компаратор СП1 имеет три управляющих входа для наращивания разрядности (A>B, A<B, A = B)и три

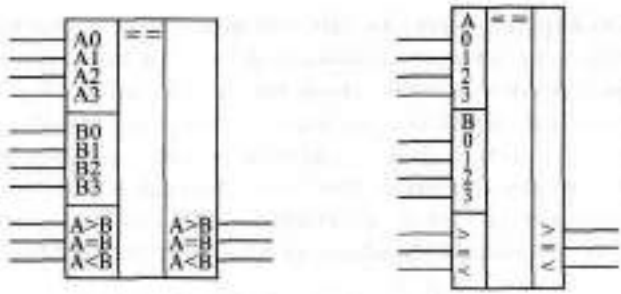


Рис. 3.16. 4-х разрядный компаратор кодов СП1 (два варианта обозначения)

выхода результирующих сигналов (A > B, A < B, A = B). Для удобства на схемах управляющие входы и выходы иногда обозначают просто ">", "<" и "=". Нулевые разряды кодов (A0 и B0) — младшие, третьи разряды (A3 и B3) - старшие.

Таблица истинности компаратора кодов (табл. 3.4) кажется на первый взгляд довольно сложной, но на самом деле все просто.

Если используется одиночная микросхема, то для ее правильной работы достаточно подать единицу на вход A = B, а состояния входов A < B и A > B не важны, на них можно подать как нуль, так и единицу. Назначение выходов понятно из их названия, а полярность выходных сигналов положительная (активный уровень — единица). Если микросхемы компараторов кодов каскадируются (объединяются) для увеличения числа разрядов сравниваемых кодов, то надо выходные сигналы микросхемы, обрабатывающей младшие разряды кода, подать на одноименные входы микросхемы, обрабатывающей старшие разряды кода (рис. 3.17).

В зарубежные стандартные серии входят также и 8-разрядные компараторы, сравнивающие два кода по величине (то есть имеющие выходы "=", ">" и "<"). Примером может служить микросхема SN74AS885.

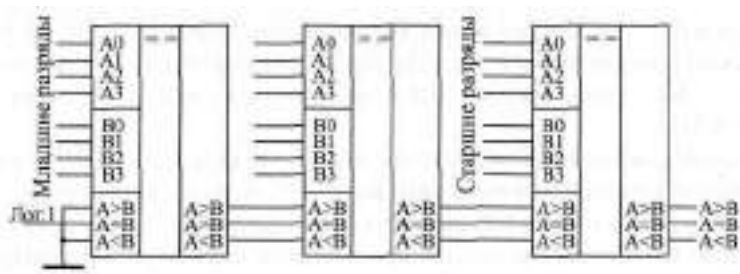


Рис. 3.17. Каскадирование компараторов кодов

Таблица 3.4. Таблица истинности компаратора СП1

Входы				Входы наращивания			Выходы		
A3, B3	A2, B2	A1, B1	A0, B0	A>B	A<B	A=B	A>B	A<B	A=B
A3>B3	X	X	X	X	X	X	1	0	0
A3<B3	X	X	X	X	X	X	0	1	0
A3=B3	A2>B2	X	X	X	X	X	1	0	0
A3=B3	A2<B2	X	X	X	X	X	0	1	0
A3=B3	A2=B2	A1>B1	X	X	X	X	1	0	0
A3=B3	A2=B2	A1<B1	X	X	X	X	0	1	0
A3=B3	A2=B2	A1=B1	A0>B0	X	X	X	1	0	0
A3=B3	A2=B2	A1=B1	A0<B0	X	X	X	0	1	0
A3=B3	A2=B2	A1=B1	A0=B0	1	0	0	1	0	0
A3=B3	A2=B2	A1=B1	A0=B0	0	1	0	0	1	0
A3=B3	A2=B2	A1=B1	A0=B0	X	X	1	0	0	1
A3=B3	A2=B2	A1=B1	A0=B0	1	1	0	0	0	0
A3=B3	A2=B2	A1=B1	A0=B0	0	0	0	1	1	0

Одно из основных применений компараторов кодов состоит в селектировании входных кодов. В этом случае достаточно иметь информацию только о совпадении кодов на входах компаратора, а не о соотношении их величин. Интересующий нас код (эталонный) подается на один вход компаратора, а изменяющийся код (входной) — на другой вход. Используется только выход равенства кодов A = B. Для подобных применений выпускаются и специальные компараторы, определяющие только совпадение кодов. Примерами могут служить 8-разрядные микросхемы SN74ALS520, SN74ALS521, DC102A фирмы Dionics (отечественный аналог - КР559СК1), а также 6-разрядная микросхема DM8136 фирмы National Semiconductors (отечественный аналог — КР559СК2).

На рис. 3.18 показано применение компараторов SN74ALS521 для селектирования 16-разрядных кодов. Инверсный сигнал с выхода первой микросхемы подается на инверсный вход разрешения второй микросхемы, выходной сигнал которой (отрицательный) говорит о совпадении входного и эталонного 16-разрядных кодов.

Неопределенные состояния на выходах компараторов кодов могут возникать при любом изменении любого из двух входных кодов. Это связано с неодновременным изменением разрядов кодов (рис. 3.19). На всех

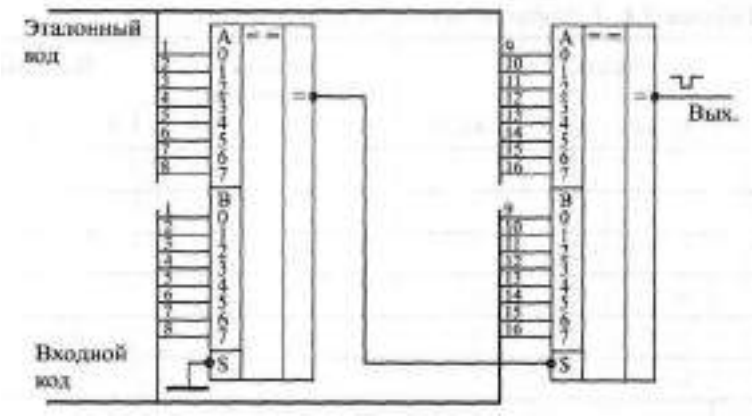


Рис. 3.18. Селектирование 16-разрядных кодов

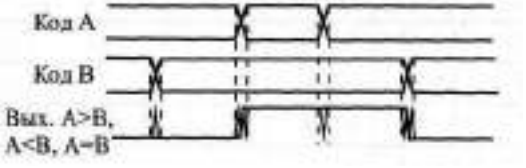


Рис. 3.19. Неопределенные состояния на выходах компаратора при изменении входных кодов

выходах компаратора СП1 могут появляться короткие паразитные импульсы. Чтобы устранить их влияние на дальнейшую часть схемы, применяется синхронизация и стробирование. Но для этого надо точно знать момент изменения входных кодов, что далеко не всегда возможно.

При применении компараторов надо учитывать также, что при каскадировании задержки микросхем суммируются и объединенный компаратор будет во столько раз медленнее одиночного, сколько микросхем в нем используется. Задержки компараторов кодов по входам разрядов кодов примерно вчетверо больше задержек логических элементов, а по входам расширения — примерно втрое. Так что эти микросхемы довольно медленные по сравнению с другими комбинационными микросхемами. Точные значения задержек надо смотреть в справочниках.

Если нам важен только факт равенства или неравенства входных кодов, то увеличить быстродействие при объединении компараторов можно, если подавать их выходные сигналы на элемент И (рис. 3.20). В этом случае суммарная задержка схемы превысит задержку одного компаратора всего лишь на задержку элемента И. При применении компараторов

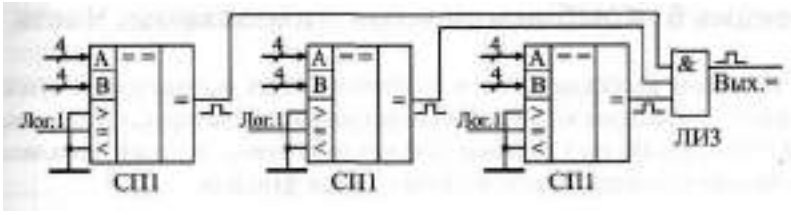


Рис. 3.20. Уменьшение задержки при каскадировании компараторов

с инверсным выходом (например, SN74ALS521) надо брать элемент **ИИ** с нужным **числом входов**.

При необходимости сравнения кодов не только на совпадение, но еще и по величине, такого простого решения не существует. Однако эта задача встречается гораздо реже.

Лекция 6. Комбинационные микросхемы. Часть 2

В лекции рассказывается о комбинационных микросхемах: сумматорах, преобразователях кодов, одновибраторах и генераторах, об их алгоритмах работы, параметрах, типовых схемах включения, а также о реализации на их основе некоторых часто встречающихся функций.

Ключевые слова: сумматор, преобразователь кода, одновибратор, ждущий мультивибратор, мультивибратор, генератор, вход расширения, выход переноса, вычитатель, каскадирование, времязадающие элементы, перезапуск, ГУН.

3.4. Сумматоры

Микросхемы сумматоров (английское Adder), как следует из их названия, предназначены для суммирования двух входных двоичных кодов, то есть выходной код будет равен арифметической сумме двух входных кодов. Например, если один входной код — 7 (0111), а второй — 5 (0101), то суммарный код на выходе будет 12 (1100). Сумма двух двоичных чисел с числом разрядов N может иметь число разрядов (N + 1). Например, при суммировании чисел 13 (1101) и 6 (0110) получается число 19 (10011). Поэтому количество выходов сумматора на единицу больше количества разрядов входных кодов. Этот дополнительный (старший) разряд называется выходом переноса.

На схемах сумматоры обозначаются буквами SM. В отечественных сериях код, обозначающий микросхему сумматора, — ИМ.

Сумматоры бывают одноразрядные (для суммирования двух одноразрядных чисел), 2-х разрядные (суммируют 2-х разрядные числа) и 4-х разрядные (суммируют 4-х разрядные числа). Чаще всего применяют именно 4-разрядные сумматоры. На рис. 3.21 показаны для примера 2-разрядный и 4-разрядный сумматоры. Микросхема ИМ6 отличается от ИМ3 только повышенным быстродействием и номерами используемых выводов микросхемы, функция же выполняется та же самая.

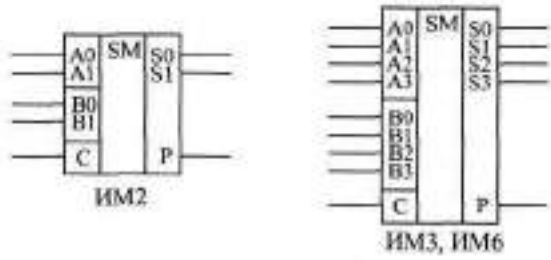


Рис. 3.21. Примеры микросхем сумматоров

Таблица 3.5. Таблица истинности микросхемы 2-разрядного сумматора ИМ2

Входы				Выходы					
A1	AO	B1	BO	p	C=0 S1	SO	p	C=1 S1	SO
0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0	1	0
0	0	1	0	0	1	0	0	1	1
0	0	1	1	0	1	1	1	0	0
0	1	0	0	0	0	1	0	1	0
0	1	0	1	0	1	0	0	1	1
0	1	1	0	0	1	1	1	0	0
0	1	1	1	1	0	0	1	0	1
1	0	0	0	0	1	0	0	1	1
1	0	0	1	0	1	1	1	0	0
1	0	1	0	1	0	0	1	0	1
1	0	1	1	1	0	1	1	1	0
1	1	0	0	0	1	1	1	0	0
1	1	0	1	1	0	0	1	0	1
1	1	1	0	1	0	1	1	1	0
1	1	1	1	1	1	0	1	1	1

Помимо выходных разрядов суммы и выхода переноса, сумматоры имеют вход расширения (другое название — вход переноса) С для объединения нескольких сумматоров с целью увеличения разрядности. Если на этот вход приходит единица, то выходная сумма увеличивается на единицу, если же приходит нуль, то выходная сумма не увеличивается. Если используется одна микросхема сумматора, то на ее вход расширения С необходимо подать нуль.

Для примера в табл. 3.5 приведена полная таблица истинности 2-разрядного сумматора ИМ2. Как видно из таблицы, выходной 3-разрядный код (P, S1, SO) равен сумме входных 2-разрядных кодов (A1, AO) и (B1, BO), а также сигнала С. Нулевые разряды — младшие, первые разряды — старшие. Полная таблица истинности 4-разрядного сумматора будет чрезмерно большой, поэтому она не приводится.

Но суть работы остается точно такой же, как и в случае 2-разрядного сумматора.

Сумматоры могут использоваться также для суммирования чисел в отрицательной логике (когда логической единице соответствует электрический нуль, и наоборот, логическому нулю соответствует электрическая единица). Но в этом случае входной сигнал переноса С также становится инверсным, поэтому при использовании одной микросхемы сумматора на вход С надо подать электрическую единицу (высокий уровень напряжения). Инверсным становится и выходной сигнал переноса Р, низкий уровень напряжения на нем (электрический нуль) соответствует наличию переноса. То есть получается, что сумматор абсолютно одинаково работает как с положительной, так и с отрицательной логикой.

Рассмотрим пример. Пусть нам надо сложить два числа 5 и 7 в отрицательной логике. Числу 5 в положительной логике соответствует двоичный код 0101, а в отрицательной — код 1010. Числу 7 в положительной логике соответствует двоичный код 0111, а в отрицательной — код 1000. При подаче на вход сумматора кодов 1010 (десятичное число 10 в положительной логике) и 1000 (десятичное число 8 в положительной логике) получаем сумму 10 + 8 = 18, то есть код 10010 в положительной логике. С учетом входного сигнала переноса С=1 (отсутствие входного переноса в отрицательной логике) выходной код сумматора получится на единицу больше: 18 + 1 = 19, то есть 10011. При отрицательной логике это будет соответствовать числу 01100, то есть 12 при отсутствии выходного переноса. В результате получили 5+7=12.

Сумматор может вычислять не только сумму, но и разность входных кодов, то есть работать вычитателем. Для этого вычитаемое число надо просто поразрядно проинвертировать, а на вход переноса С подать единичный сигнал (рис. 3.22).

Например, пусть нам надо вычислить разность между числом 11 (1011) и числом 5 (0101). Инвертируем поразрядно число 5 и получаем 1010, то есть десятичное 10. Сумматор при суммировании 11 и 10 даст 21, то есть двоичное число 10101. Если сигнал С равен 1, то результат будет

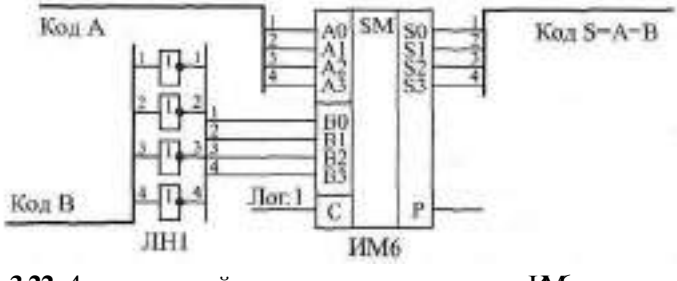


Рис. 3.22. 4-х разрядный вычитатель на сумматоре ИМ6 и инверторах ЛН1

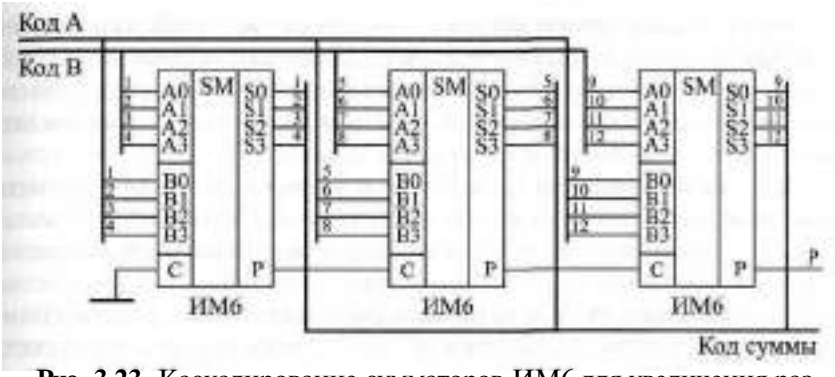


Рис. 3.23. Каскадирование сумматоров ИМ6 для увеличения разрядности

10110. Отбрасываем старший разряд (выходной сигнал P) и получаем разность 0110, то есть 6.

Еще пример. Пусть надо вычислить разность между числом 12 (1100) и числом 9 (1001). Инвертируем поразрядно 9, получаем 0110, то есть десятичное 6. Находим сумму 12 и 6, получаем 18, а с учетом C = 1 получаем 19, то есть двоичное 10011. В четырех младших разрядах имеем 0011, то есть десятичное 3.

Каскадировать сумматоры для увеличения разрядности очень просто. Надо сигнал с выхода переноса сумматора, обрабатывающего младшие разряды, подать на вход переноса сумматора, обрабатывающего старшие разряды (рис. 3.23). При объединении трех 4-разрядных сумматоров получается 12-разрядный сумматор, имеющий дополнительный 13-й разряд (выход переноса P).

Неопределенные состояния на выходах сумматора могут возникать при любом изменении любого из входных кодов (рис. 3.24). Выходной код суммы может принимать в течение короткого времени значения, никак не связанные с входными кодами, а на выходе переноса могут появляться короткие паразитные импульсы. Это связано прежде всего с неод-

Рис. 3.24. Неопределенные состояния на выходах сумматора при изменении входных кодов

новременным изменением разрядов входных кодов. Чтобы избежать влияния этих неопределенных состояний на дальнейшую схему, необходимо предусматривать синхронизацию или стробирование выходных сигналов. Но для этого надо располагать информацией о моментах изменения входных кодов, которая имеется далеко не всегда.

Задержки сумматора ИМ6 от входов до выходов суммы примерно вдвое превышает задержку логического элемента, а от входов до выхода переноса — примерно в полтора раза. Задержки сумматора ИМ3 больше задержек ИМ6 почти вдвое. Поэтому в схемах, где важно быстродействие, лучше использовать ИМ6. Особенно это существенно при каскадировании для увеличения разрядности, так как там задержки отдельных микросхем суммируются. Точные величины задержек надо смотреть в справочниках.

3.5. Преобразователи кодов

Микросхемы преобразователей кодов (англ. converter) служат для преобразования входных двоичных кодов в выходные двоично-десятичные и наоборот - входных двоично-десятичных кодов в выходные двоичные. Они используются довольно редко, так как применение двоично-десятичных кодов ограничено узкой областью, например, они применяются в схемах многоразрядной десятичной индикации. К тому же при правильной организации схемы часто можно обойтись без преобразования в двоично-десятичный код, например, выбирая счетчики, работающие в двоично-десятичном коде. Преобразование двоично-десятичного кода в двоичный встречается еще реже.

На схемах микросхемы преобразователей обозначаются буквами X/Y. В отечественных сериях преобразователи имеют обозначения ПР.

Кроме того, надо учесть, что любые преобразования параллельных кодов, даже самые экзотические, могут быть легко реализованы на микросхемах постоянной памяти нужного объема. Обычно это намного удобнее, чем брать стандартные микросхемы преобразователей кодов.

В стандартные серии входят две микросхемы преобразователей кодов: ПР6 для преобразования двоично-десятичного кода в двоичный

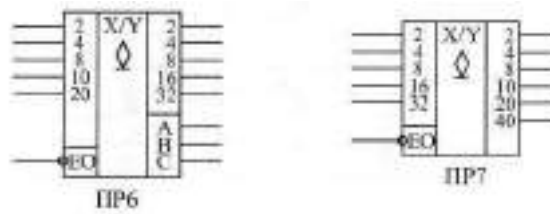


Рис. 3.25. Микросхемы преобразователей кодов

и ПР7 для преобразования двоичного кода в двоично-десятичный (рис. 3.25). Обе микросхемы имеют выходы ОК, поэтому к ним надо присоединять нагрузочные резисторы величиной около 1 кОм, но для удобства в дальнейших схемах эти резисторы не показаны. Обе микросхемы имеют также вход разрешения выхода ЕО при нулевом уровне на котором все выходы активны, а при единичном — переходят в состояние единицы. Преобразователь ПР6 имеет дополнительные выходы A, B, C, не участвующие в основном преобразовании.

Таблица 3.6. Таблица истинности преобразователя ПР6

Входы						Выходы				
-ЕО	20	10	8	4	2	32	16	8	4	2
1	X	X	X	X	X	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	1	1	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	0	1	0	0	1	0	0	1	1	0
0	0	1	0	1	0	0	0	1	1	1
0	0	1	0	1	1	0	1	0	0	0
0	0	1	1	0	0	0	1	0	0	1
0	1	0	0	0	0	0	1	0	1	0
0	1	0	0	0	1	0	1	0	1	1
0	1	0	0	1	0	0	1	1	0	0
0	1	0	0	1	1	0	1	1	0	1
0	1	0	1	0	0	0	1	1	1	0
0	1	1	0	0	0	0	1	1	1	1
0	1	1	0	0	1	1	0	0	0	0
0	1	1	0	1	0	1	0	0	0	1
0	1	1	0	1	1	1	0	0	1	0
0	1	1	1	0	0	1	0	0	1	1

Таблицы истинности преобразователей просты (табл. 3.6 и 3.7). Например, двоично-десятичный код без младшего разряда на входе ПР6 преобразуется в двоичный код без младшего разряда на выходе ПР6. Младший разряд не участвует в преобразовании, он непосредственно передается со входа на выход. Одна микросхема ПР6 обрабатывает входные коды в диапазоне от 0 (двоично-десятичный код 00 000) до 39 (код 11 1001).

Точно так же двоичный код без младшего разряда на входе ПР7 преобразуется в двоично-десятичный код без младшего разряда на выходе ПР7. Одна микросхема ПР7 может обрабатывать входные коды в диапазоне от 0 (двоичный код 000000) до 63 (код 111111). Младшие разряды входных кодов передаются на выход без обработки в обход микросхемы, так как они одинаковые как в двоичном, так и в двоично-десятичном кодах. Простейшие схемы включения одиночных микросхем ПР6 и ПР7 приведены на рис. 3.26.

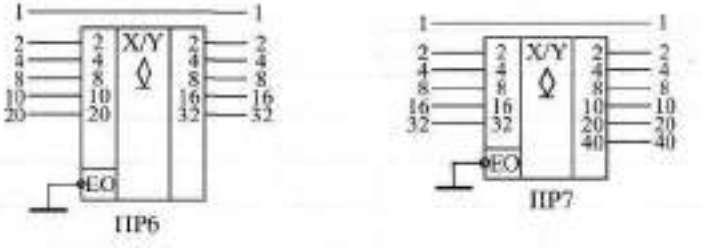


Рис. 3.26. Простейшее включение одиночных преобразователей кода ПР6 и ПР7

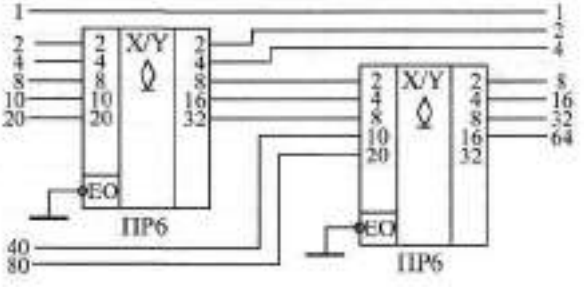


Рис. 3.27. Преобразователь двоично-десятичного кода от 0 до 99 в двоичный код

Каскадировать преобразователи ПР6 и ПР7 для увеличения разрядности также несложно. Для преобразования двоично-десятичных кодов от 0 до 99 достаточно двух микросхем ПР6 (рис. 3.27), а для преобразования двоичных кодов от 0 до 255 требуется три микросхемы ПР7 (рис. 3.28).

Таблица 3.7. Таблица истинности преобразователя ПР7

Входы						Выходы					
-E0	32	16	8	4	2	40	20	10	8	4	2
1	X	X	X	X	X	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	1
0	0	0	0	1	0	0	0	0	0	1	0
0	0	0	0	1	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	1	0	1	0	0	1	0	0	0
0	0	0	1	1	0	0	0	1	0	0	1
0	0	0	1	1	1	0	0	1	0	1	0
0	0	1	0	0	0	0	0	1	0	1	1
0	0	1	0	0	1	0	0	1	1	0	0
0	0	1	0	1	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	0	0	0	1
0	0	1	1	0	0	0	1	0	0	1	0
0	0	1	1	0	1	0	1	0	0	1	1
0	0	1	1	1	0	0	1	0	1	0	0
0	0	1	1	1	1	0	1	1	0	0	0
0	1	0	0	0	0	0	1	1	0	0	1
0	1	0	0	0	1	0	1	1	0	1	0
0	1	0	0	1	0	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1	1	0	0
0	1	0	1	0	0	1	0	0	0	0	0
0	1	0	1	0	1	1	0	0	0	0	1
0	1	0	1	1	0	1	0	0	0	1	0
0	1	0	1	1	1	1	0	0	0	1	1
0	1	1	0	0	0	1	0	0	1	0	0
0	1	1	0	0	1	1	0	1	0	0	0
0	1	1	0	1	0	1	0	1	0	0	1
0	1	1	0	1	1	1	0	1	0	1	0
0	1	1	1	0	0	1	0	1	0	1	1
0	1	1	1	0	1	1	0	1	1	0	0
0	1	1	1	1	0	1	1	0	0	0	0
0	1	1	1	1	1	1	1	0	0	0	1

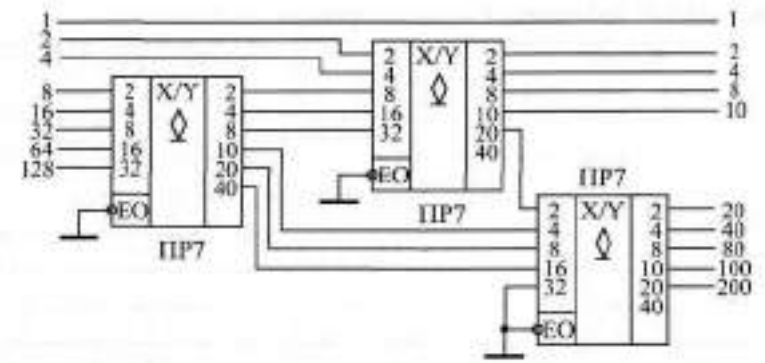


Рис. 3.28. Преобразователь двоичного кода от 0 до 255 в двоично-десятичный код

Если надо преобразовывать двоично-десятичные коды до 999, то понадобится 6 микросхем ПР6, а для преобразования двоичных кодов до 511 потребуется 4 микросхемы ПР7. На всех выходах микросхем необходимо включать нагрузочные резисторы.

Наличие дополнительных выходов A, B, C у микросхемы ПР6 позволяет преобразовывать двоично-десятичный код от 0 до 9 в код дополнения до 9 или до 10 (рис. 3.29). То есть сумма входного и выходного кодов в этом случае равна, соответственно, 9 или 10. Например, при входном коде 6 на выходе схемы *а* будет код 3, а на выходе схемы *б* — код 4. В схеме *б* при входном коде 0 на выходе также формируется код 0. Как и все остальные выходы микросхемы ПР6, выходы A, B, C имеют тип ОК, поэтому к ним необходимо присоединять нагрузочные резисторы, для удобства не показанные на схеме. Такие схемы "дополнителей" применяются редко, поэтому о них упоминают не во всех справочниках и учебниках, но иногда подобные функции бывают довольно удобны.

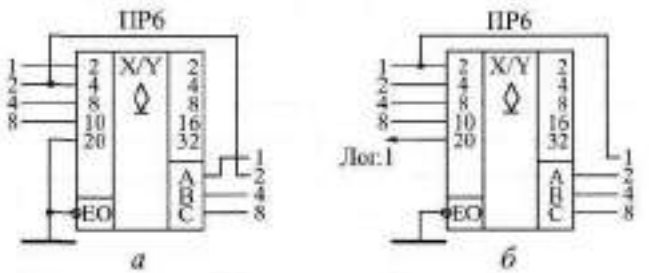


Рис. 3.29. Преобразователи входного кода в дополнение до 9 (а) и в дополнение до 10 (б)

Задержки преобразователей кодов примерно вдвое превосходят задержки логических элементов. Точные величины задержек надо смотреть в справочниках.

3.6. Одновибраторы и генераторы

Одновибраторы и генераторы вообще-то нельзя отнести к комбинационным микросхемам. Они занимают промежуточное положение между комбинационными микросхемами и микросхемами с внутренней памятью. Их выходные сигналы не определяются однозначно входными сигналами, как у комбинационных микросхем. Но в то же время они и не хранят информацию длительное время.

Одновибраторы ("ждущие мультивибраторы", английское название "Monostable Multivibrator") представляют собой микросхемы, которые в ответ на входной сигнал (логический уровень или фронт) формируют выходной импульс заданной длительности. Длительность определяется внешними времязадающими резисторами и конденсаторами. То есть можно считать, что у одновибраторов есть внутренняя память, но эта память хранит информацию о входном сигнале строго заданное время, а потом информация исчезает. На схемах одновибраторы обозначаются буквами G1.

В стандартные серии микросхем входят одновибраторы двух основных типов (отечественное обозначение функции микросхемы — АГ):

- Одновибраторы без перезапуска (АГ1 — одиночный одновибратор, АГ4 — два одновибратора в корпусе).
- Одновибраторы с перезапуском (АГ3 — два одновибратора в корпусе).

Разница между этими двумя типами иллюстрируется рис. 3.30. Одновибратор без перезапуска не реагирует на входной сигнал до окончания своего выходного импульса. Одновибратор с перезапуском начинает отсчет нового времени выдержки T с каждым новым входным сигналом независимо от того, закончилось ли предыдущее время выдержки. В случае, когда период следования входных сигналов меньше времени выдержки T, выходной импульс одновибратора с перезапуском не прерывается. Если период следования входных запускающих импульсов больше време-

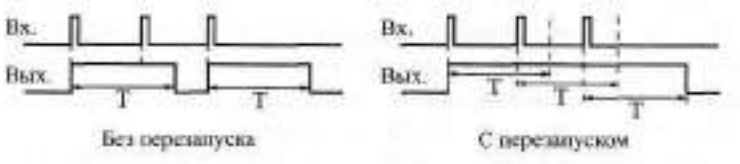


Рис. 3.30. Принцип работы одновибраторов без перезапуска и с перезапуском

Рис. 3.31. Микросхемы одновибраторов

ни выдержки одновибратора T, то оба типа одновибраторов работают одинаково.

На рис. 3.31 приведены обозначения микросхем одновибраторов стандартных серий. Микросхемы АГ3 и АГ4 отличаются друг от друга только тем, что АГ3 работает с перезапуском, а АГ4 — без перезапуска.

Микросхемы имеют входы запуска, объединенные по И и ИЛИ, прямые и инверсные выходы, а также выводы для подключения внешних времязадающих цепей (резисторов и конденсаторов). Запускается работа всех одновибраторов по фронту результирующего входного сигнала. Использованная логика объединения входов микросхем позволяет запустить все одновибраторы как по положительному, так и по отрицательному фронту входного сигнала (рис. 3.32 и 3.33).

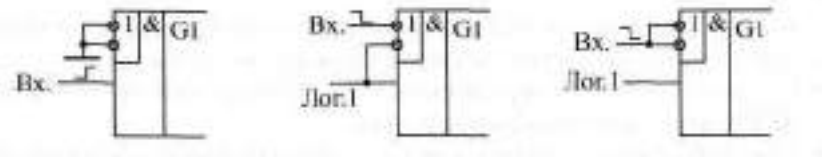


Рис. 3.32. Варианты запуска одновибратора АГ1

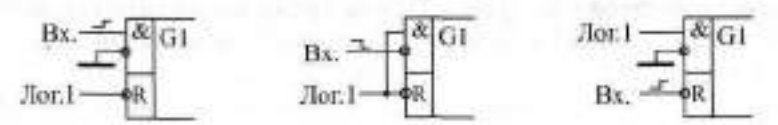


Рис. 3.33. Варианты запуска одновибраторов АГ3 и АГ4

На неиспользуемые входы при этом надо подавать сигналы логического нуля или логической единицы. Можно также использовать остающиеся входы для разрешения или запрещения входного запускающего сигнала.

Одновибраторы АГ3 и АГ4 имеют также дополнительный вход сброса -R, логический нуль на котором не только запрещает выработку выходного сигнала, но и прекращает его. Вход -R можно также использовать для запуска одновибратора.

Таблица 3.8. Таблица истинности одновибратора АГ1

Входы			Выходы	
-A1	-A2	B	Q	-Q
0	X	1	0	1
X	0	1	0	1
X	X	0	0	1
1	1	X	0	1
1	⌐	1	⌐ L	⌐¬
⌐	1	1	J L	⌐— Г
⌐	⌐	1	J L	⌐— Г
0	X	¬	J L	⌐¬
X	0	_Г	¬⌐ L	⌐¬

Таблица 3.9. Таблица истинности одновибраторов АГ3 и АГ4

Входы			Выходы	
-R	-A	**в**	Q	-Q
0	X	X	0	1
X	1	0	0	1
X	X	0	0	1
1	0	¬	J L	⌐¬
1	⌐	1	¬⌐ L	1__ Г
¬	0	1	J L	⌐— Г

Таблицы истинности одновибраторов приведены в табл. 3.8 и 3.9. Здесь инверсные входные сигналы обозначены -A, -A1, -A2, прямые входные сигналы — B, а прямой и инверсный выходные сигналы — соответственно, Q и -Q.

Стандартное включение одновибраторов предполагает подключение внешнего резистора и внешнего конденсатора (рис. 3.34).

Для одновибратора АГ1 длительность выходного импульса можно оценить по формуле T = 0,7RC. Эта формула работает при величине сопротивления резистора в пределах от 1,5 кОм до 43 кОм. Емкость конденсатора может быть любой. Внутри микросхемы имеется внутренний Резистор сопротивлением около 2 кОм, подключенный к выводу R, поэтому можно включать одновибратор без внешнего резистора, подклю-

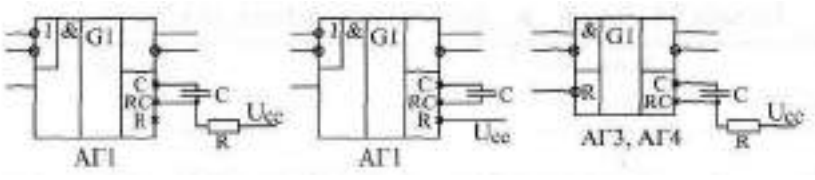


Рис. 3.34. Стандартные схемы включения одновибраторов

чая вывод R к напряжению питания. Повторный запуск одновибратора невозможен сразу после окончания выходного импульса, до повторного запуска обязательно должен пройти интервал t = C (если емкость измеряется в нанофарадах, то временной интервал получается в микросекундах).

Для одновибраторов АГ3 и АГ4 длительность импульса можно оценить по формуле: T = O, 32C (R + 0, 7), где сопротивление резистора измеряется в килоОмах. Сопротивление резистора может находиться в пределах от 5,1 кОм до 51 кОм, емкость конденсатора — любая. Перезапуск одновибратора возможен только в том случае, когда интервал между входными запускающими импульсами больше O, 224C (если емкость измеряется в нанофарадах, то временной интервал — в микросекундах).

Наиболее распространенные применения одновибраторов следующие (рис. 3.35):

а) увеличение длительности входного импульса;
б) уменьшение длительности входного импульса;
в) деление частоты входного сигнала в заданное число раз;
г) формирование сигнала огибающей последовательности входных импульсов.

Для увеличения или уменьшения длительности входного сигнала (*а* и *б*) надо всего лишь выбрать сопротивление резистора и емкость конденсатора, исходя из требуемой длительности выходного сигнала. В этом

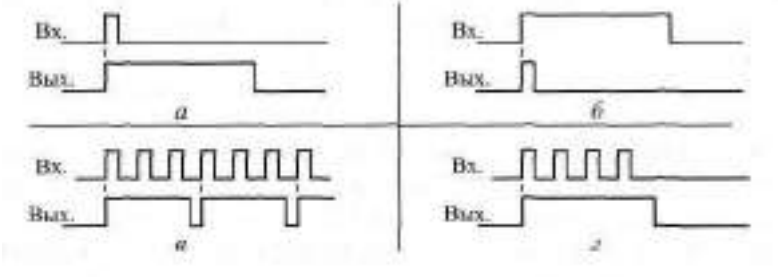


Рис. 3.35. Стандартные применения одновибраторов

случае можно использовать одновибратор любого типа: как с перезапуском, так и без перезапуска.

Для деления частоты входных импульсов в заданное число раз *(в)* применяется только одновибратор без перезапуска. При этом надо выбрать такую длительность выходного сигнала, чтобы одновибратор пропускал нужное количество входных импульсов. Например, если требуется разделить на 3 частоту входных импульсов f, то длительность выходного сигнала одновибратора надо выбрать в пределах от $2/f$ до $3/f$. При этом одновибратор будет пропускать два входных импульса из каждых трех.

Для формирования огибающей входного сигнала (г) используется только одновибратор с перезапуском. При этом длительность его выходного импульса должна быть выбрана такой, чтобы каждый следующий входной сигнал перезапускал одновибратор. Если частота входного сигнала равна f, то длительность выходного сигнала одновибратора должна быть не меньше, чем $1/f$.

Еще одно важное применение одновибратора состоит в подавлении дребезга контактов кнопки. Одновибратор с большим временем выдержки (порядка нескольких десятых долей секунды) надежно подавляет паразитные импульсы, возникающие из-за дребезга контактов, и формирует идеальные импульсы на любое нажатие кнопки (рис. 3.36).

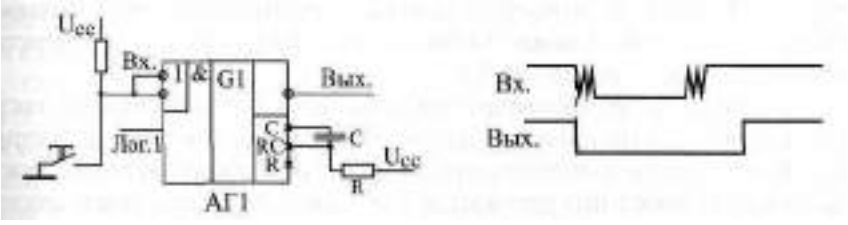


Рис. 3.36. Использование одновибратора для подавления дребезга контактов кнопки

Для этого можно использовать как одновибратор с перезапуском, так и одновибратор без перезапуска (на рисунке). Можно также подобрать время выдержки так, что одновибратор будет давать один импульс по нажатию кнопки, а другой импульс — по отпусканию кнопки. Иногда это бывает удобнее.

Одновибраторы можно также применять для построения генераторов (мультивибраторов) прямоугольных импульсов с различными значениями длительности импульсов и паузы между ними. При этом два одновибратора замыкаются в кольцо так, что каждый из них запускает другой после окончания своего выходного импульса (рис. 3.37). Один одновиб-

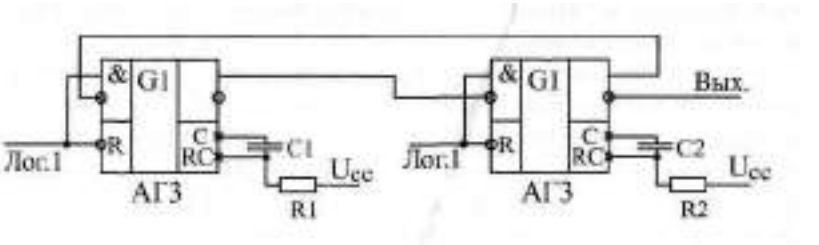


Рис. 3.37. Генератор импульсов на двух одновибраторах

братор формирует длительность импульса, а другой определяет паузу между импульсами. Изменяя номиналы резисторов и конденсаторов, можно получить нужные соотношения импульса и паузы.

Таким образом, одновибраторы довольно легко позволяют решать самые разные задачи. Однако, применяя одновибраторы, надо всегда помнить, что длительность их выходных импульсов нельзя задать очень точно — ведь одновибратор имеет аналоговые цепи. На длительность выходного импульса одновибратора влияют разбросы номиналов резисторов и конденсаторов, температура окружающей среды, старение элементов, помехи по цепям питания и другие факторы. Поэтому применение одновибраторов нужно по возможности ограничивать только теми случаями, когда время выдержки можно задавать с не слишком высокой точностью (погрешность не менее 20—30 %).

Любую функцию одновибратора может выполнить синхронное тактируемое устройство (на основе кварцевого генератора, триггеров, регистров, счетчиков), причем выполнить гораздо точнее и надежнее. И ему не нужно никаких дополнительных времязадающих элементов (резисторов и конденсаторов). Количество одновибраторов, использованных в схеме, как правило, обратно пропорционально уровню мастерства разработчика этой схемы.

Задержки запуска одновибраторов примерно в два—три раза превосходят задержку логического элемента. Точные величины задержек надо смотреть в справочниках.

Помимо одновибраторов, в стандартные серии включены также специализированные **генераторы** ("мультивибраторы", англ. "Multivibrator"). Обозначаются они на схемах буквой G. В отечественных сериях этот тип микросхемы кодируется буквами ГТ. Например, микросхема ГИ представляет собой два генератора в одном корпусе.

Микросхемы генераторов используют довольно редко, чаще применяют генераторы на инверторах или на триггерах Шмитта, описанные во второй главе. Однако в некоторых случаях генераторы ГТ не могут быть заменены ничем. Дело в том, что они допускают изменение частоты

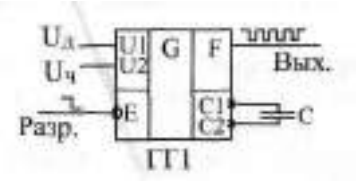


Рис. 3.38. Схема включения генератора ГГ1

выходных импульсов с помощью уровней двух входных управляющих напряжений. Поэтому они называются также "генераторы, управляемые напряжением" или ГУН. Эффект изменения частоты можно использовать, например, в системах автоподстройки частоты (АПЧ) или в устройствах с частотной модуляцией (ЧМ).

Стандартная схема включения генератора ГГ1 приведена на рис. 3.38. Генератор имеет выводы для подключения внешнего конденсатора С1 и С2, к которым можно также подключать кварцевый резонатор, но при этом уже нельзя управлять частотой. Имеется два входа управления частотой U1 и U2, а также вход разрешения —E, при подаче на который логической единицы генерация прекращается и на выходе F устанавливается единица.

Один из входов управления (U1) обычно называется диапазонным или $U_{д}$, а другой (U2) — входом управления частоты или $U_{ч}$. При увеличении напряжения $U_{ч}$ частота увеличивается, при увеличении напряжения на входе $U_{д}$ — уменьшается. Рекомендуемый диапазон изменения напряжения $U_{д}$ составляет от 2 до 4,5 В, а диапазон изменения $U_{ч}$ — от 0 до 5 В. В зависимости от напряжения $U_{д}$, меняется диапазон изменения частоты из-за изменения напряжения $U_{ч}$. Например, при $U_{д}$ = 2 В и изменении $U_{ч}$ от 1 до 5 В частота изменяется примерно на 15 %, а при $U_{д}$ = 4 В - приблизительно в 4 раза.

Частота выходного сигнала ГГ1 определяется также внешним конденсатором, например, при $U_{д}$ = $U_{ч}$ = 2 В и при С = 1 мкФ частота будет около 100 Гц, а при С = 100 пФ — порядка 10 МГц. Максимально возможное значение частоты генератора составляет около 80 МГц. В справочниках приводятся графики зависимости частоты выходного сигнала ГГ1 от уровней управляющих напряжений и от величины внешнего конденсатора. Однако точно определить значение частоты по этим графикам Невозможно, в любом случае требуется подстройка. К тому же наличие в схеме аналоговых узлов делает генератор ГГ1 чувствительным к разбросу номиналов конденсаторов, к изменению температуры окружающей

среды, к старению элементов, к помехам по цепям питания и к другим факторам. Именно поэтому использование этих генераторов крайне ограничено.

И последнее. В микросхеме ПЛ существует взаимное влияние двух генераторов друг на друга, хотя в ней и приняты меры по снижению этого влияния. Поэтому не рекомендуется использовать одновременно два генератора одной микросхемы в режиме генерации частоты, управляемой напряжением.

Глава 4. Применение триггеров и регистров

Лекция 7. Триггеры

В лекции рассказывается о триггерах различных типов, об алгоритмах их работы, параметрах, типовых схемах включения, а также о реализации на их основе некоторых часто встречающихся функций.

Ключевые слова: триггерная ячейка, триггеры RS, JK и D, сверхоперативная память, устойчивые состояния, счетный режим, флаг процесса, синхронизация сигналов, линия задержки, разделение импульсов, формирователь огибающей, частотный фильтр, манчестерское кодирование.

Триггеры и регистры являются простейшими представителями цифровых микросхем, имеющих внутреннюю память. Если выходные сигналы логических элементов и комбинационных микросхем однозначно определяются их текущими входными сигналами, то выходные сигналы микросхем с внутренней памятью зависят также еще и от того, какие входные сигналы и в какой последовательности поступали на них в прошлом, то есть они помнят предысторию поведения схемы. Именно поэтому их применение позволяет строить гораздо более сложные и интеллектуальные цифровые устройства, чем в случае простейших микросхем без памяти. Микросхемы с внутренней памятью называются еще последовательными или последовательностными, в отличие от комбинационных микросхем.

Триггеры и регистры сохраняют свою память только до тех пор, пока на них подается напряжение питания. Иначе говоря, их память относится к типу *оперативной* (в отличие от *постоянной* памяти и *перепрограммируемой постоянной* памяти, которым отключение питания не мешает сохранять информацию). После выключения питания и его последующего включения триггеры и регистры переходят в случайное состояние, то есть их выходные сигналы могут устанавливаться как в уровень логической единицы, так и в уровень логического нуля. Это необходимо учитывать при проектировании схем.

Большим преимуществом триггеров и регистров перед другими типами микросхем с памятью является их максимально высокое быстродействие (то есть минимальные времена задержек срабатывания и максимально высокая допустимая рабочая частота). Именно поэтому триггеры и регистры иногда называют также *сверхоперативной* памятью. Однако

недостаток триггеров и регистров в том, что объем их внутренней памяти очень мал, они могут хранить только отдельные сигналы, биты (триггеры) или отдельные коды, байты, слова (регистры).

Триггер можно рассматривать как одноразрядную, а регистр — как многоразрядную ячейку памяти, которая состоит из несколько триггеров, соединенных параллельно (обычный, параллельный регистр) или последовательно (сдвиговый регистр или, что то же самое, регистр сдвига).

4.1. Триггеры

4.1.1. Принцип работы и разновидности триггеров

В основе любого триггера (англ. — "Trigger" или "flip-flop") лежит схема из двух логических элементов, которые охвачены положительными обратными связями (то есть сигналы с выходов подаются на входы). В результате подобного включения схема может находиться в одном из двух устойчивых состояний, причем находиться сколь угодно долго, пока на нее подано напряжение питания.

Пример такой схемы (так называемой триггерной ячейки) на двух двухвходовых элементах И-НЕ представлен на рис. 4.1. У схемы есть два инверсных входа: -R — сброс (от английского Reset), и -S — установка (от английского Set), а также два выхода: прямой выход Q и инверсный выход -Q.

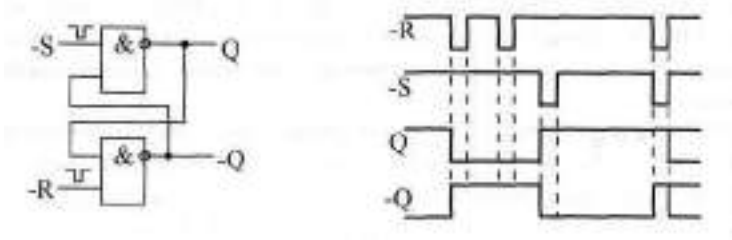


Рис. 4.1. Схема триггерной ячейки

Для правильной работы схемы отрицательные импульсы должны поступать на ее входы не одновременно. Приход импульса на вход R переводит выход Q в состояние единицы, а так как сигнал S при этом единичный, выход Q становится нулевым. Этот же сигнал Q поступает по цепи обратной связи на вход нижнего элемента. Поэтому даже после окончания импульса на входе R состояние схемы не изменяется (на Q остается нуль, на -Q остается единица). Точно так же при приходе импульса на вход -S выход Q в единицу, а выход -Q — в нуль. Оба эти устойчивых со-

стояния триггерной ячейки могут сохраняться сколь угодно долго, пока не придет очередной входной импульс, — иными словами, схема обладает памятью.

Если оба входных импульса придут строго одновременно, то в момент действия этих импульсов на обоих выходах будут единичные сигналы, а после окончания входных импульсов выходы случайным образом попадут в одно из двух устойчивых состояний. Точно так же случайным образом будет выбрано одно из двух устойчивых состояний триггерной ячейки при включении питания. Временная диаграмма работы триггерной ячейки показана на рисунке.

Таблица истинности схемы приведена в табл. **4.1.**

Таблица 4.1. Таблица истинности триггерной ячейки

Входы		Выходы	
-R	-S	Q	-Q
0	1	0	1
1	0	1	0
1	1	Без изменения	
0	0	Не определено	

В стандартные серии цифровых микросхем входит несколько типов микросхем триггеров, различающихся методами управления, а также входными и выходными сигналами. На схемах триггеры обозначаются буквой Т. В отечественных сериях микросхем триггеры имеют наименование ТВ, ТМ и ТР в зависимости от типа триггера. Наиболее распространены три типа (рис. 4.2):

- RS-триггер (обозначается ТР) — самый простой триггер, но редко используемый (а).
- JK-триггер (обозначается ТВ) имеет самое сложное управление, также используется довольно редко (б).
- D-триггер (обозначается ТМ) — наиболее распространенный тип триггера (в).

Примером RS-триггера является микросхема ТР2, в одном корпусе которой находятся четыре RS-триггера. Два триггера имеют по одному входу -R и -S, а два других триггера — по одному входу -R и по два входа -S1 и -S2, объединенных по функции И. Все триггеры имеют только по одному прямому выходу. RS-триггер практически ничем не отличается по своим функциям от триггерной ячейки, рассмотренной ранее (см. рис. 4.1). Отрицательный импульс на входе -R перебрасывает выход в нуль, а отрицательный импульс на входе -S (или на любом из входов -S1 и -S2)

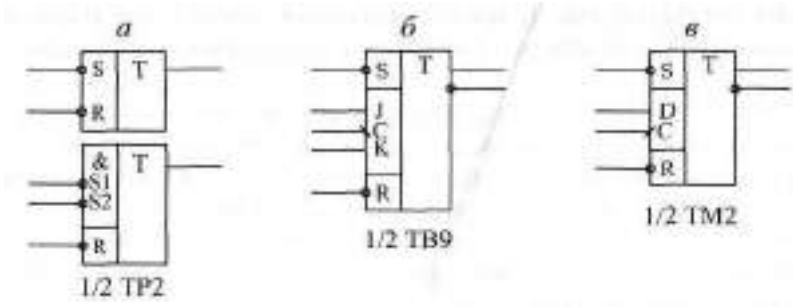


Рис. 4.2. Триггеры трех основных типов

перебрасывает выход в единицу. Одновременные сигналы на входах -R и -S переводят выход в единицу, а после окончания импульсов триггер попадает случайным образом в одно из своих устойчивых состояний. Таблица истинности триггера ТР2 с двумя входами установки -S1 и -S2 представлена в табл. 4.2.

Таблица 4.2. Таблица истинности RS-триггера ТР2

Входы			Выходы
-S1	-S2	-R	Q
1	1	1	Без изменения
X	0	1	1
0	X	1	1
1	1	0	0
X	0	0	Не определен
0	X	0	Не определен

Ж-триггер значительно сложнее по своей структуре, чем RS-триггер. Он относится к так называемым тактируемым триггерам, то есть он срабатывает по фронту тактового сигнала. Примером может служить показанная на рис. 4.2 микросхема ТВ9, имеющая в одном корпусе два JK-триггера со входами сброса и установки R и S. Входы R и S работают точно так же, как и в RS-триггере, то есть отрицательный импульс на входе R устанавливает прямой выход в нуль, а инверсный — в единицу, а отрицательный импульс на входе S устанавливает прямой выход в единицу, а инверсный — в нуль.

Однако состояние триггера может быть изменено не только этими сигналами, но и сигналами на двух информационных входах J и K и син-

хросигналом C. Переключение триггера в этом случае происходит по отрицательному фронту сигнала C (по переходу из единицы в нуль) в зависимости от состояний сигналов J и K. При единице на входе J и нуле на входе K по фронту сигнала C прямой выход устанавливается в единицу (обратный — в нуль). При нуле на входе J и единице на входе K по фронту сигнала C прямой выход устанавливается в нуль (обратный — в единицу). При единичных уровнях на обоих входах J и K по фронту сигнала C триггер меняет состояние своих выходов на противоположные (это называется счетным режимом).

Таблица 4.3. Таблица истинности JK-триггера ТВ9

Входы					Выходы	
-S	-R	C	J	K	Q	-Q
0	1	X	X	X	1	0
1	0	X	X	X	0	1
0	0	X	X	X	Не определено	
1	1	1-X)	1	0	1	0
1	1	1-X)	0	1	0	1
1	1	1-X)	0	0	Не изменяется	
1	1	1-X)	1	1	Меняется на противоположное	
1	1	1	X	X	Не изменяется	
1	1	0	X	X	Не изменяется	
1	1	0->1	X	X	Не изменяется	

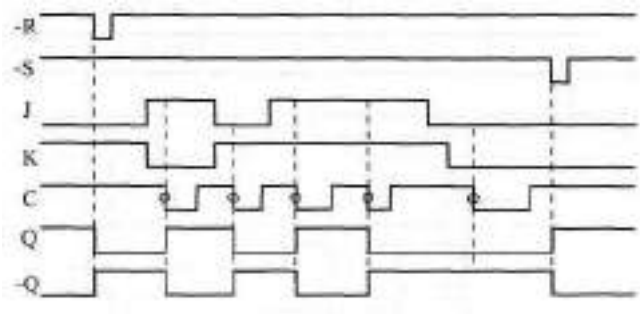


Рис. 4.3. Временная диаграмма работы JK-триггера ТВ9

Таблица истинности триггера ТВ9 представлена в табл. 4.3, а временная диаграмма работы — на рис. 4.3.

Наконец, самый распространенный D-триггер занимает, можно сказать, промежуточное положение между RS-триггером и JK-триггером. Помимо общих для всех триггеров входов установки и сброса -S и -R, он имеет один информационный вход D (вход данных) и один тактовый вход С. Примером может служить показанная на рис. 4.2 микросхема ТМ2, содержащая в одном корпусе два D-триггера с прямыми и инверсными выходами.

Тактируется триггер (то есть меняет свое состояние) по положительному фронту сигнала С (по его переходу из нуля в единицу) в зависимости от состояния входа данных D. Если на входе D единичный сигнал, то по фронту сигнала С прямой выход триггера устанавливается в единицу (инверсный — в нуль). Если же на входе D — нулевой сигнал, то по фронту сигнала С прямой выход триггера устанавливается в нуль (инверсный — в единицу).

Таблица истинности триггера ТМ2 представлена в табл. 4.4, а временная диаграмма работы — на рис. **4.4.**

Таблица 4.4. Таблица истинности D-триггера ТМ2

Входы				Выходы	
–S	–R	C	D	Q	–Q
0	1	X	X	1	0
1	0	X	X	0	1
0	0	X	X	Не определено	
1	1	0->1	1	1	0
1	1	0->1	0	0	1
1	1	0	X	Не меняется	
1	1	1	X	Не меняется	
1	1	1-X)	X	Не меняется	

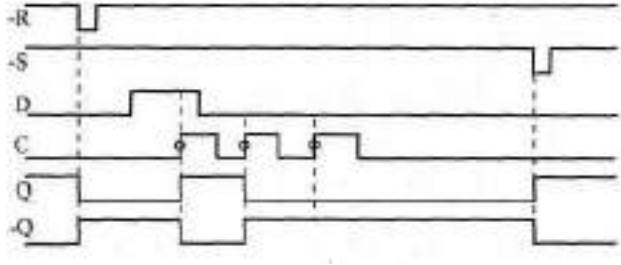


Рис. 4.4. Временная диаграмма работы D-триггера ТМ2

Остановимся на работе D-триггера чуть подробнее, так как он наиболее часто используется. При этом многие замечания, высказанные здесь относительно D-триггера, будут верны и для других типов триггеров.

Прежде всего отметим, что все приведенные временные диаграммы относятся к первому уровню представления, к уровню логической модели. Конечно же, в реальности все триггеры имеют временные задержки установки выходных сигналов, а также предъявляют определенные временные требования к входным сигналам, при нарушении которых любой триггер будет работать неустойчиво или же не будет работать вообще. Это учитывается на втором уровне представления (в модели с временными задержками).

Например, как уже отмечалось, входные сигналы R и S не должны приходить одновременно, иначе состояние триггера будет неопределенным. Длительность сигналов R и S также не должна быть слишком малой, иначе триггер может на них не среагировать. Сигнал -R должен начинаться с определенной задержкой после окончания сигнала -S, и наоборот. В первом приближении можно считать, что минимально допустимые временные интервалы между входными сигналами должны равняться 1–2 задержкам логического элемента соответствующей серии.

Точно так же не должна быть слишком малой длительность тактового сигнала C (как положительного, так и отрицательного импульса), иначе триггер может переключаться неустойчиво. Это требование универсально для всех микросхем, срабатывающих по фронту входного сигнала. Принципиально важна и величина временного сдвига (задержки) между установлением сигнала D и рабочим (положительным) фронтом сигнала C. Этот сдвиг тоже не должен быть слишком малым. Не должен быть чрезмерно малым и сдвиг между окончанием сигналов -R и -S и рабочим фронтом сигнала C. Повышенные требования предъявляются также к длительности фронта тактового сигнала C, которая не должна быть слишком большой. Это требование также универсально для всех микросхем, срабатывающих по фронту входного сигнала.

Одним словом, чем сложнее микросхема, тем важнее для нее становятся ограничения второго уровня представления, тем выше требования к разработчику по учету временных задержек и длительностей сигналов. Правда, требования эти не слишком разнообразны и не слишком жестки, поэтому, раз и навсегда усвоив их, можно проектировать любые схемы без грубых ошибок. Самое главное, что надо запомнить, состоит в следующем: цифровые схемы не любят слишком коротких входных сигналов и слишком малых задержек между входными сигналами, функционально связанными между собой. Ориентир здесь очень простой - величина задержки логического элемента данной серии. Поэтому для более быстрых серий ограничения будут менее жесткими, а для более медленных серий — более жесткими.

Несколько слов о величинах задержек микросхем триггеров.

Несмотря на свою достаточно сложную внутреннюю структуру, микросхемы триггеров являются одними из самых быстрых. Задержка срабатывания триггера обычно не превышает 1,5–2 задержки логического элемента. (причем задержки по входам -R и -S чуть меньше, чем по тактовому входу С.) В некоторых сериях Ж-триггеры несколько быстрее, чем D-триггеры, в других — наоборот. Важный параметр триггера — максимальная частота тактового сигнала С. Для ее приблизительной оценки можно придерживаться следующего простого правила: период тактового сигнала С не должен быть меньше величины задержки переключения триггера по входу С.

4.1.2. Основные схемы включения триггеров

Говоря об областях применения триггеров, мы будем рассматривать исключительно D-триггеры, так как в большинстве случаев RS- и Ж-триггеры могут быть заменены D-триггерами без ухудшения каких бы то ни было параметров схемы. Примеры такой замены показаны на рис. 4.5.

RS-триггер получается из D-триггера, если в D-триггере не использовать входы С и D, например, соединить их с общим проводом (*а*).

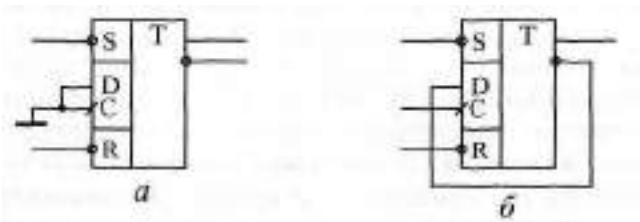


Рис. 4.5. Включение D-триггера для замены RS-триггера (а) и Ж-триггера в счетном режиме (б)

Сложнее обстоит дело с заменой JK-триггера, в котором предусмотрено больше возможностей, чем в D-триггере. Однако обычно два информационных входа Ж-триггера не так уж и нужны. А что касается счетного режима, в котором, пожалуй, наиболее часто работают Ж-триггеры, то он легко реализуется на D-триггере в результате объединения информационного входа D с инверсным выходом (*б*). При этом по каждому положительному фронту сигнала С триггер будет менять свое состояние на противоположное: нуль на прямом выходе будет сменяться единицей и наоборот. То есть частота входного сигнала триггера будет меньше частоты входного тактового сигнала С в два раза.

Отметим также, что для реализации счетного режима чаще всего используются не триггеры, а счетчики, которые будут рассмотрены в следующей главе.

Особенности триггеров обусловливают наиболее широкий диапазон схем их включения для решения самых разных задач.

Например, с помощью триггера (любого типа) очень просто и эффективно решается задача устранения влияния дребезга контактов механических переключателей (рис. 4.6). Правда в данном случае необходим тумблер (или кнопка) с тремя выводами, один из которых попеременно подключается к двум другим. При этом первый отрицательный импульс на входе -R перебрасывает триггер в состояние нуля, а первый отрицательный импульс на входе -S — в состояние единицы. Последующие же импульсы на обоих этих входах, вызванные дребезгом контактов, уже никак не влияют на триггер. Нижнее (по рисунку) положение выключателя соответствует нулю на выходе триггера, а верхнее — единице.

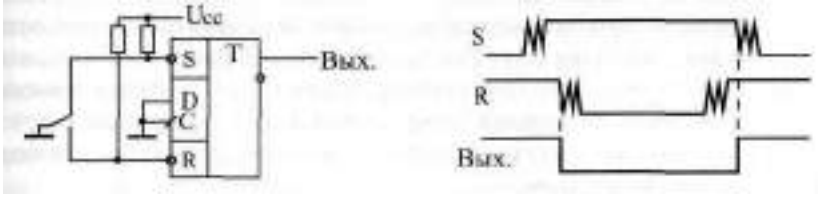


Рис. 4.6. Подавление дребезга контактов выключателя с помощью триггера

Основное применение триггеры находят в тех случаях, когда надо сформировать сигнал, длительность которого соответствует длительности какой-то выполняемой операции, какого-то продолжительного процесса в схеме. Выходной сигнал триггера при этом может разрешать этот самый процесс, а может информировать остальные узлы устройства о том, что процесс идет (или, как говорят, служить *флагом* процесса). Например, в схеме на рис. 4.7 в начале процесса (операции) по сигналу "Старт" триггер перебрасывается в единицу, а в конце процесса (операции) по сигналу "Стоп" — обратно в нуль.

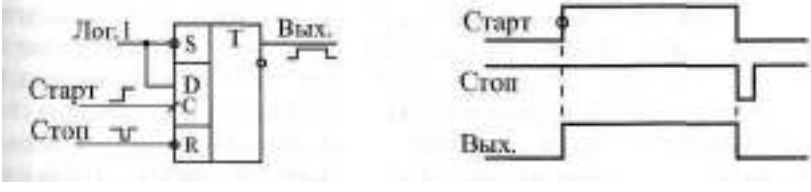


Рис. 4.7. Использование триггера в качестве флага процесса

Для сигналов "Старт" и "Стоп" можно, конечно, использовать входы триггера R и S. Однако более правильным и универсальным решением будет выбор пары входов C и R или C и S, что предотвратит неоднозначность поведения триггера при одновременном приходе сигналов "Старт" и "Стоп". Если используются входы C и R, то на вход D надо подать единицу, а если применяются входы C и S, то на вход D надо подать нуль. Такое решение удобно еще и тем, что в качестве одного из сигналов "Старт" и "Стоп" может выступать не уровень, а фронт. Именно этот фронт (в нужной полярности) и надо подать в этом случае на тактовый вход триггера C.

Вторая важнейшая область применения триггеров — синхронизация сигналов.

Например, триггер позволяет наиболее просто избавиться от паразитных коротких импульсов на выходах комбинационных схем, возникающих при почти одновременном изменении нескольких входных сигналов (рис. 4.8). Для синхронизации в данном случае необходимо иметь синхросигнал (синхропереход), сопровождающий входные информационные сигналы (входной код) и задержанный относительно момента изменения этих сигналов на время $t_з$, большее задержки комбинационной схемы. При подаче этого синхроимпульса на вход C триггера, а выходного сигнала комбинационной микросхемы (Вых. 1) на вход D триггера на выходе триггера получаем сигнал (Вых. 2), полностью свободный от паразитных импульсов.

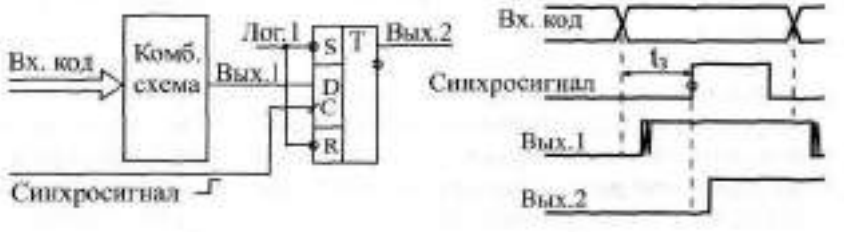


Рис. 4.8. Синхронизация с помощью триггера

Более того, в случае, когда входной код комбинационной схемы изменяется регулярно, периодически, фронт синхросигнала может даже совпадать с моментом изменения входного кода (рис. 4.9).

При этом, за счет конечной величины задержки комбинационной схемы сигнал на вход C триггера будет поступать раньше, чем начнет изменяться сигнал на его входе D. Поэтому паразитные импульсы в триггер не запишутся. Правда, в данном случае сигнал на выходе триггера (Вых. 2) будет задержан на период следования входных кодов T (или, что то же самое, на период синхросигнала) относительно выходного сигнала комбинационной схемы (Вых. 1).

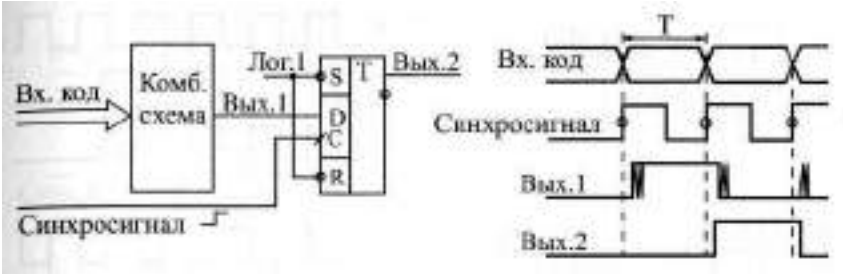


Рис. 4.9. Синхронизация с помощью триггера при периодическом изменении входного кода

При проектировании цифровых схем, работающих по тактам единого тактового генератора, часто возникает необходимость синхронизовать с работой схемы какой-то внешний сигнал. То есть требуется обеспечить, чтобы этот сигнал (асинхронный по отношению ко всей остальной схеме) изменялся по тактам тактового генератора, как и все остальные сигналы схемы (стал бы синхронным всей остальной схеме). В этом тоже может помочь триггер.

Рассмотрим самый простой пример. Пусть необходимо с помощью внешнего сигнала разрешать и запрещать прохождение сигнала непрерывно работающего тактового генератора. В случае обычного RC-генератора эта задача иногда может быть решена довольно просто — путем запуска и остановки генератора (см. рис. 2.39). Однако далеко не всегда допускается останавливать работу тактового генератора, от которого работает вся схема. В случае же кварцевого генератора его остановка и запуск вообще никогда не применяются, так как такой генератор начинает работать после разрешения с задержкой, равной нескольким периодам тактовой частоты, причем количество этих периодов не постоянно. Поэтому будем считать, что тактовый генератор работает постоянно, а по внешнему управляющему сигналу мы будем разрешать или запрещать прохождение его выходных импульсов (рис. 4.10).

В простейшем случае *(а)* для пропускания и запрещения импульсов тактового генератора Г используется логический элемент 2И. При этом вполне возможна ситуация прохождения на выход схемы импульсов неполной длительности или даже предельно коротких, нестабильно появляющихся импульсов, которые могут вносить неопределенность в работу остальной схемы.

Применение синхронизирующего триггера (б) обеспечивает прохождение на выход пропускающего элемента 2 И только импульсов полной Длительности. Разрешающий сигнал, проходя через триггер, который тактируется разрешаемым сигналом, становится синхронным с тактовым

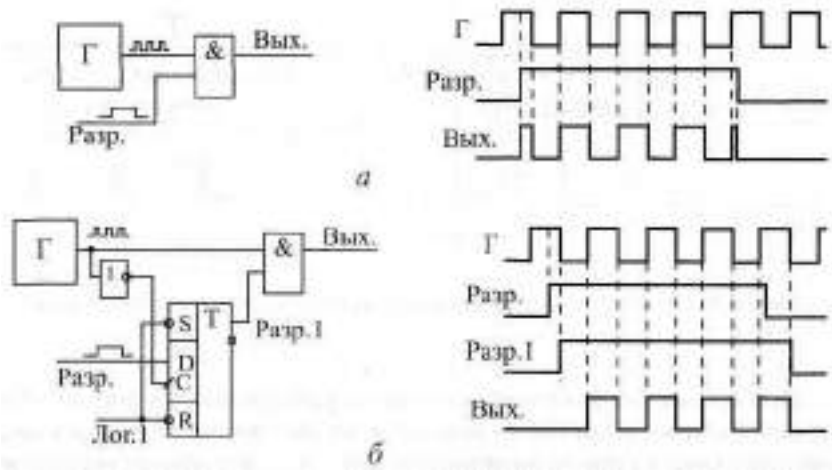


Рис. 4.10. Синхронизация сигнала разрешения

сигналом и гарантирует прохождение на выход обязательно целого количества тактовых импульсов, целого количества периодов тактового генератора.

Триггеры позволяют также строить линии задержки цифровых сигналов, для чего несколько триггеров соединяется в последовательную цепочку, причем все они тактируются единым тактовым сигналом С. Такое включение позволяет, например, одновременно обрабатывать комбинационными схемами несколько последовательных во времени состояний какого-то одного сигнала.

В качестве примера на рис. 4.11 приведена схема, которая выделяет во входном сигнале 3-х тактовую последовательность 010. Цепочка из трех триггеров Т1, Т2 и Т3, тактируемых единым синхросигналом, запоминает три последовательных состояния входного сигнала. Например, если на выходе триггера Т2 будет зафиксировано состояние входного сигнала в N-ом такте, то на выходе триггера Т1 будет состояние входного сигнала в такте (N+1), а на выходе триггера Т3 — в такте (N–1). Из-за конечной величины задержки переключения триггеров в каждый следующий триггер входной сигнал будет переписываться еще до того как он поменяет свое значение вследствие переключения предыдущего триггера.

Подавая выходные сигналы триггеров (прямые или инверсные в зависимости от нужных уровней) на элемент И с нужным числом входов, можно зафиксировать любую 3-х тактовую последовательность во входном сигнале. Для предотвращения появления паразитных импульсов в выходном сигнале (они возможны, так как входные сигналы элемента И изменяются почти одновременно) применяется выходной триггер Т,

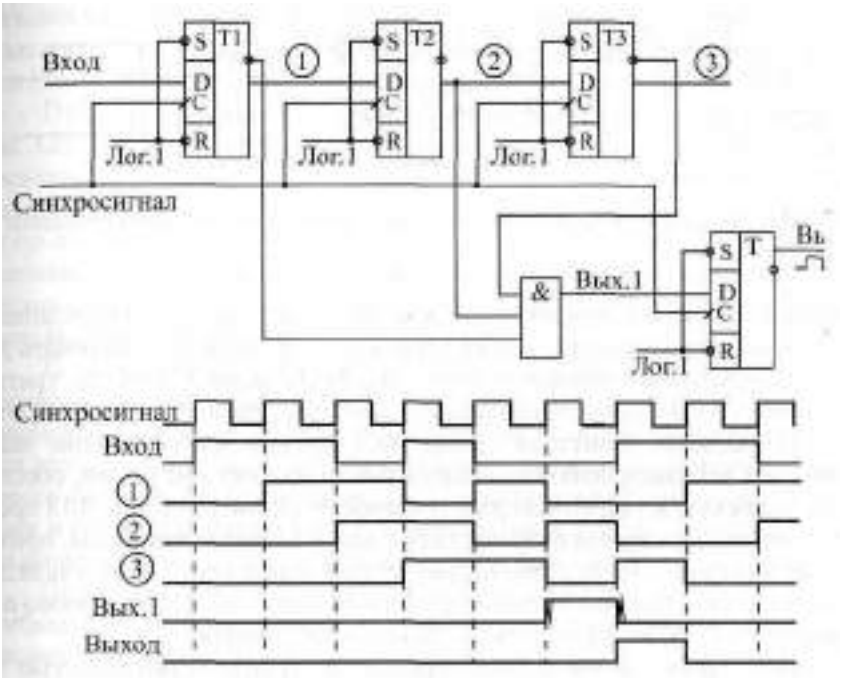


Рис. 4.11. Выделение 3-тактовой последовательности тактов во входном сигнале

тактируемый тем же самым общим синхросигналом. На выходе триггера Т получаем единичный сигнал, соответствующий последовательности 010 во входном сигнале. Правда, этот выходной сигнал будет задержан относительно конца выделяемой последовательности 010 на два такта.

Конечно, применение триггеров не ограничивается рассмотренными примерами, все области их применения трудно даже перечислить. Мы же рассмотрим здесь еще несколько примеров использования триггеров.

D-триггер позволяет довольно просто формировать выходной короткий импульс по фронту входного сигнала. Для этого даже не нужно никаких времязадающих RC-цепочек. Длительность выходного импульса определяется задержкой срабатывания триггера. Формирователь короткого импульса по положительному фронту входного сигнала (рис. 4.12) образуется путем подачи выходного сигнала триггера на вход сброса.

По положительному фронту на входе С триггер перебрасывается в единицу, но выходной сигнал триггера по цепи обратной связи тут же сбрасывает его обратно в нуль. Преимуществом данной схемы является то, что триггер имеет как прямой, так и инверсный выходы, поэтому

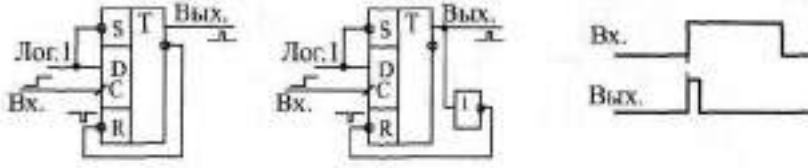


Рис. 4.12. Формирователь короткого импульса по фронту входного сигнала

мы получаем как положительный короткий импульс, так и отрицательный. В некоторых случаях в цепь этой обратной связи надо включать дополнительный инвертор для устойчивой работы схемы. Например, триггеры серии К155 не требуют инвертора, а триггеры серии КР1533 — требуют.

Применение триггеров совместно с другими микросхемами часто позволяет избежать появления паразитных коротких импульсов, обеспечить надежную и уверенную работу схемы. Например, на рис. 4.13 представлена схема, которая различает короткие и длинные импульсы, приходящие на ее вход. Такая схема позволяет применять одну линию связи для передачи двух сигналов разного назначения, что бывает очень удобно при связи устройств, находящихся на большом расстоянии.

На вход схемы поступают короткие импульсы (длительностью $t_к$) и длинные импульсы (длительностью $t_д$). Конечно, на передающем конце надо обеспечить, чтобы эти импульсы формировались по очереди и с не слишком малой задержкой друг относительно друга. На выходе схемы формируются два сигнала, один из которых соответствует приходу короткого входного импульса, а другой — приходу длинного входного импульса.

Для различения входных импульсов используется одновибратор АП с временем выдержки $t_в$, большим $t_к$, но меньшим $t_д$. Применение одновибратора в данном случае оправдано, так как требуемая точность времени выдержки невысока (считаем, что длительности импульсов раз-

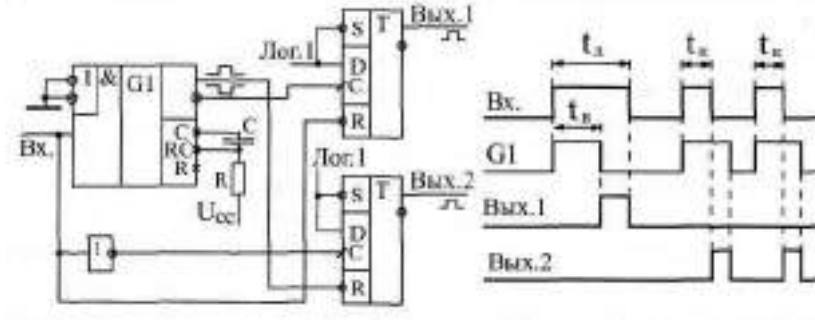


Рис. 4.13. Схема разделения коротких и длинных входных импульсов

личаются существенно). Выходные сигналы схемы формируются с помощью двух триггеров, а не простых двухвходовых логических элементов, что полностью исключает появление паразитных импульсов на фронтах.

Принцип работы схемы ясен из приведенной временной диаграммы. Одновибратор запускается по переднему фронту входного сигнала. Выходной сигнал Вых. 1, соответствующий приходу длинного импульса, начинается по заднему фронту импульса с одновибратора, а заканчивается по окончанию длинного входного импульса. Выходной сигнал Вых. 2, соответствующий приходу входного короткого импульса, начинается по заднему фронту входного импульса, а заканчивается с окончанием импульса с одновибратора.

Триггеры можно также использовать для обработки периодических последовательностей входных сигналов. При этом триггер, тактируемый кварцевым генератором, может очень точно различать частоты следования входных импульсов, то есть выполнять функцию простейшего цифрового фильтра. Такие схемы выгодно отличаются от схем с одновибраторами и времязадающими RC-цепочками возможностью полностью интегрального исполнения и отсутствием какой бы то ни было настройки.

Простейший пример подобной обработки состоит в формировании огибающей входного сигнала. То есть при приходе входного сигнала заданной частоты выходной сигнал должен быть равен единице, а при отсутствии входного сигнала — нулю. Эта задача, как уже отмечалось (см. рис. 3.35г), может быть решена с помощью одновибратора с перезапуском (типа АГ3). Однако применение триггеров значительно увеличивает точность срабатывания и позволяет работать с частотами, близкими к предельным для данного типа триггеров. Схема формирования огибающей состоит всего лишь из двух триггеров, тактируемых внешним синхросигналом, тактом (рис. 4.14). В данном случае предполагается, что частоты входного сигнала и тактового сигнала равны между собой.

Триггеры включены как двухтактная линия задержки с общим тактовым сигналом С и со сбросом входными сигналами. Первый же входной сигнал последовательности начинает выходной сигнал, а заканчива-

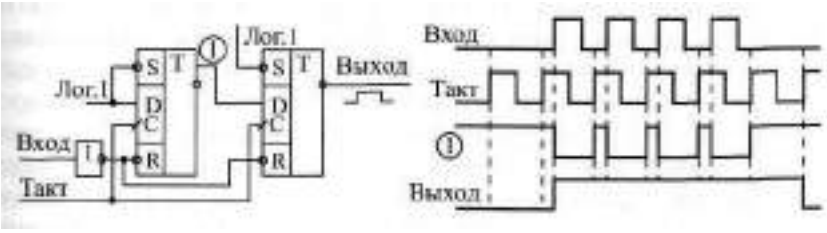


Рис. 4.14. Формирователь сигнала огибающей входного сигнала на триггерах

ется выходной сигнал через 1–2 такта после окончания входной последовательности периода тактового сигнала (в зависимости от временного сдвига входного сигнала относительно тактового сигнала). Схема работает с входным сигналом любой частоты, большей половины частоты тактового сигнала (например, при тактовой частоте 10 МГц входной сигнал должен иметь частоту, большую 5 МГц). То есть за половину периода входной частоты не должно приходить больше одного положительного фронта тактового сигнала.

Этот же формирователь огибающей можно использовать в более сложных схемах. Примером может служить фильтр, который позволяет разделить две частоты входного сигнала, пропустить более высокочастотный сигнал и отсечь более низкочастотный (рис. 4.15).

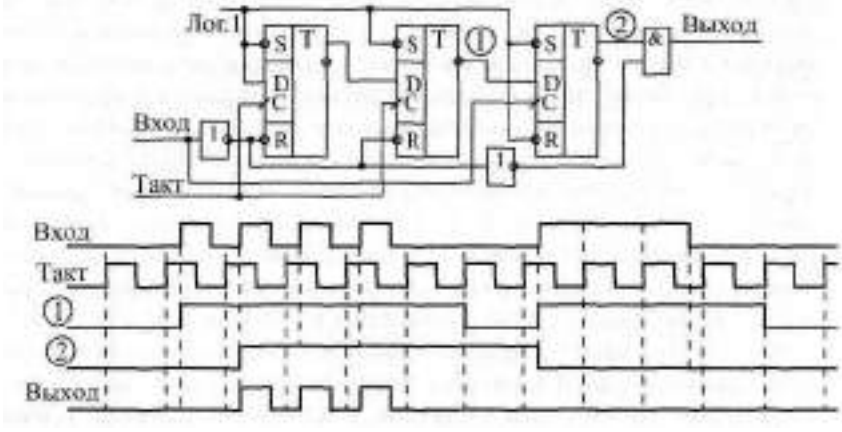


Рис. 4.15. Фильтр для пропускания высокочастотных сигналов на триггерах

Фильтр состоит из трех триггеров и элемента 2И, работающего в режиме пропускания положительных входных импульсов. Два триггера (левые по рисунку) образуют формирователь огибающей. Третий (правый по рисунку) триггер выдает сигнал пропускания в случае, когда сигнал огибающей непрерывен, то есть когда частота входного сигнала составляет больше половины тактовой частоты. Если в момент прихода положительного фронта входного сигнала сигнал огибающей на выходе второго триггера — нулевой, то пропускающий сигнал на выходе третьего триггера также нулевой и на выход импульсы не проходят. При этом первый входной импульс пропускаемого сигнала на выход не проходит. Цепочка из двух инверторов компенсирует задержку срабатывания третьего триггера, она задерживает входной сигнал перед подачей его на вход выходного пропускающего элемента 2И.

Таким образом, фильтр надежно пропускает входные сигналы с частотой, большей половины тактовой частоты, и надежно задерживает сигналы с частотой, меньшей четверти тактовой частоты. Например, при тактовой частоте 10 МГц фильтр будет пропускать сигналы с частотой выше 5 МГц, и задерживать сигналы с частотой ниже 2,5 МГц. С частотами входного сигнала от 2,5 до 5 МГц работа фильтра не будет стабильной, она будет зависеть от временного сдвига между входным сигналом и тактовым сигналом.

Наконец, последняя схема на триггерах, которую мы рассмотрим, предназначена для кодирования входного сигнала в манчестерский код (или код Манчестер-II). Этот код широко используется при передаче сигналов на большие расстояния, в частности, в локальных сетях.

Суть манчестерского кодирования иллюстрируется рис. 4.16. Входной сигнал представляет собой последовательность бит равной длительности. В каждом такте передается один бит информации. Манчестерский код заменяет единичный информационный бит на отрицательный переход в центре битового интервала, а нулевой информационный бит — на положительный переход в центре битового интервала. Таким образом, в центре каждого битового интервала сигнала в манчестерском коде обязательно имеется фронт (положительный или отрицательный), который может быть использован приемником этого сигнала для синхронизации приема каждого информационного бита. Поэтому манчестерский код называется самосинхронизирующимся кодом.

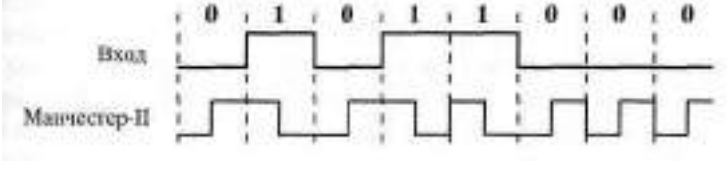


Рис. 4.16. Манчестерское кодирование

Кодировщик (он же шифратор) манчестерского кода (рис. 4.17) включает в себя элемент Исключающее ИЛИ, который, собственно, и производит кодирование, а также три триггера для синхронизации. Один триггер (на рисунке слева) работает в счетном режиме, деля частоту тактового сигнала в два раза. Один триггер (на рисунке в центре) синхронизирует входной информационный сигнал с тактовым сигналом половинной частоты. Наконец, последний, третий триггер (на рисунке справа) синхронизирует выходной сигнал для устранения в нем паразитных коротких импульсов в моменты изменения входного сигнала. Он фиксирует выходной сигнал элемента Исключающее ИЛИ (уже готовый манчестерский код)

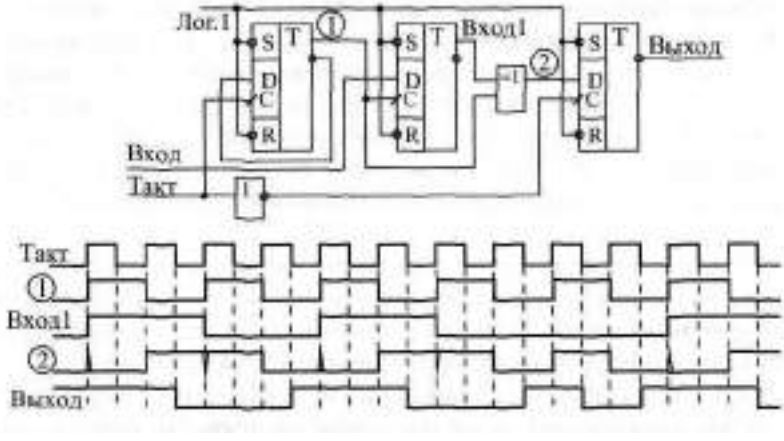


Рис. 4.17. Кодировщик манчестерского кода на триггерах

через четверть периода после изменения входного сигнала Вход 1 (по отрицательному фронту исходного тактового сигнала).

Лекция 8. Регистры

В лекции рассказывается о параллельных регистрах и регистрах сдвига, об алгоритмах их работы, параметрах, типовых схемах включения, а также о реализации на их основе некоторых часто встречающихся функций.

Ключевые слова: параллельный регистр, сдвиговый регистр, тактируемый регистр, стробируемый регистр, хранение кода, линия задержки кода, конвейер, накапливающий сумматор, вычислитель максимального кода, защелка, направление сдвига, параллельно-последовательное и последовательно-параллельное преобразование, генератор псевдослучайной последовательности.

4.2. Регистры

Регистры (англ. register) представляют собой, по сути, несколько D-триггеров (обычно от 4 до 16), соединенных между собой тем или иным способом. Поэтому принципиальной разницы между ними и отдельными D-триггерами не существует. Правда, триггеры, входящие в состав регистров, не имеют такого количества разнообразных управляющих входов, как одиночные триггеры.

На схемах регистры обозначаются буквами RG. В отечественных сериях микросхем регистрам соответствуют буквы ИР. Все регистры делятся на две большие группы (рис. 4.18):

- Параллельные регистры;
- Регистры сдвига (или сдвиговые регистры).

Существуют регистры и других типов, но они применяются гораздо реже, чем параллельные и сдвиговые, так как имеют узкоспециальное назначение.

В параллельных регистрах (*а*) каждый из триггеров имеет свой независимый информационный вход (D) и свой независимый информацион-

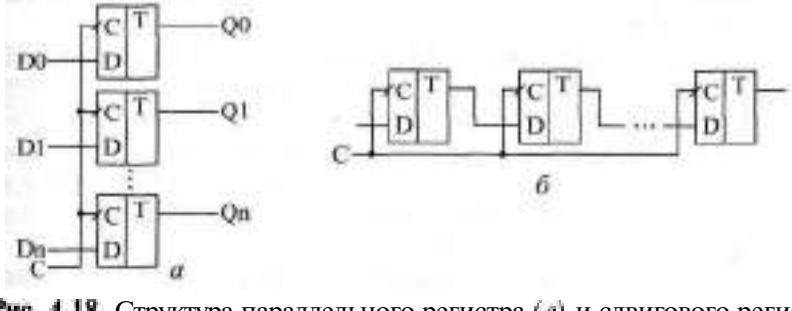


Рис. 4.18. Структура параллельного регистра (*а*) и сдвигового регистра (б)

ный выход. Тактовые входы (C) всех триггеров соединены между собой. В результате параллельный регистр представляет собой многоразрядный, многовходовый триггер.

В сдвиговых регистрах (б) все триггеры соединены в последовательную цепочку (выход каждого предыдущего триггера соединен со входом D следующего триггера). Тактовые входы всех триггеров (C) объединены между собой. В результате такой триггер может рассматриваться как линия задержки, входной сигнал которой последовательно перезаписывается из триггера в триггер по фронту тактового сигнала C. Информационные входы и выходы триггеров могут быть выведены наружу, а могут и не выводиться — в зависимости от функции, выполняемой регистром.

Параллельные регистры, в свою очередь, делятся на две группы:

- Регистры, срабатывающие по фронту управляющего сигнала C (или тактируемые регистры).
- Регистры, срабатывающие по уровню управляющего сигнала C (или стробируемые регистры).

Чаще всего в цифровых схемах используются регистры, управляемые фронтом (то есть тактируемые), однако и стробируемые регистры имеют свой круг задач, в которых их ничто не может заменить.

4.2.1. Регистры, срабатывающие по фронту

Принцип действия регистров, срабатывающих по фронту тактового сигнала, ничем не отличается от принципа действия D-триггера. По положительному фронту тактового сигнала C каждый из выходов регистра устанавливается в тот уровень, который был в этот момент на соответствующем данному выходу входе D, и сохраняется таковым до прихода следующего положительного фронта сигнала C. То есть если триггер запоминает один сигнал (один двоичный разряд, один бит), то регистр запоминает сразу несколько (4, 6, 8, 16) сигналов (несколько разрядов, битов). Память регистра сохраняется до момента выключения питания схемы.

В стандартные серии входит несколько типов параллельных регистров, срабатывающих по фронту (рис. 4.19). Различаются они количеством разрядов, наличием или отсутствием инверсных выходов, наличием или отсутствием входа сброса (-R) или разрешения записи (-WE), а также типом выходных каскадов (2C или 3C) и, соответственно, наличием или отсутствием входа разрешения -EZ. Иногда на схемах тактовый вход C обозначается WR — сигнал записи в регистр.

Большинство регистров имеют восемь разрядов, то есть запоминают один байт информации. Регистр ТМ8 в справочниках обычно называется счетверенным D-триггером (он и в наименовании несет буквы ТМ), хотя

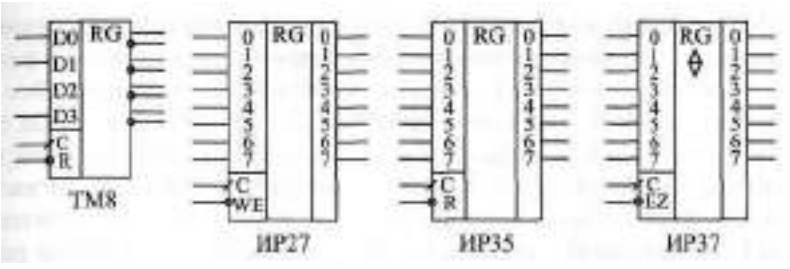


Рис. 4.19. Параллельные регистры стандартных серий, срабатывающие по фронту

он вполне может рассматриваться и как регистр, так как тактовый вход C и вход сброса -R у всех четырех триггеров объединены между собой.

Таблицы истинности регистров очень просты и не отличаются принципиально от таблицы истинности D-триггеров. Отличие от триггеров появляется только в случае наличия у регистра дополнительных управляющих входов разрешения записи -WE и разрешения выхода -EZ. В качестве примеров в табл. 4.5 приведена таблица истинности регистра ИР27, а в табл. 4.6 — регистра ИР37. По переходу тактового сигнала C из 0 в 1 (положительный фронт) оба регистра записывают в себя входную информацию.

Таблица 4.5. Таблица истинности регистра ИР27

Входы			Выходы
-WE	C	D	Q
0	0->1	0	0
0	0->1	1	1
0	0	X	Не меняется
0	1	X	Не меняется
1	X	X	Не меняется

Таблица 4.6. Таблица истинности регистра ИР37

Входы			Выходы
-EZ	C	D	Q
0	0->1	0	0
0	0->1	1	1
0	0	X	Не меняется
0	1	X	Не меняется
1	X	X	Z

Все регистры, имеющие выход с тремя состояниями, обеспечивают повышенную нагрузочную способность. Задержка переключения регистров примерно соответствует задержке переключения триггеров. Все временные ограничения, накладываемые на входные сигналы в случае триггеров (см. раздел 4.1.1), справедливы и для входных сигналов регистров. Например, не должна быть слишком малой длительность сигнала C, а также не должна быть слишком малой задержка между установлением сигнала D и приходом положительного фронта сигнала C. Иначе работа регистра может быть нестабильной или даже неправильной.

Одно из основных применений регистров состоит в хранении требуемого кода в течение нужного времени. Если для работы остальной части схемы необходимо иметь входной код, который можно легко изменять, то для этого как раз подходит регистр.

На рис. 4.20 показана типичная схема включения регистра для хранения кода и временная диаграмма его работы. Код на входе регистра может изменяться произвольным образом, но в тот момент, когда этот код принимает необходимое значение, на вход C триггера подается синхросигнал (строб), который записывает код в регистр. Этот код будет храниться в регистре до прихода следующего строба. Причем важно и то, что все разряды выходного кода регистра будут переключаться одновременно даже в том случае, когда разряды входного кода переключаются не одновременно. Главное, чтобы к приходу положительного фронта строба (сигнала C) все разряды входного кода приняли нужное, устойчивое значение.

Рис. 4.20. Хранение кода в параллельном регистре

Еще одно важнейшее применение регистров связано с запоминанием нескольких последовательных значений изменяющегося входного кода. Это позволяет, например, сравнивать предыдущее значение кода с последующим значением этого же кода или производить арифметические операции над несколькими последовательными значениями одного и того же кода. То есть регистр в данном случае выступает как элемент линии задержки, хранящей в себе историю поведения входного кода.

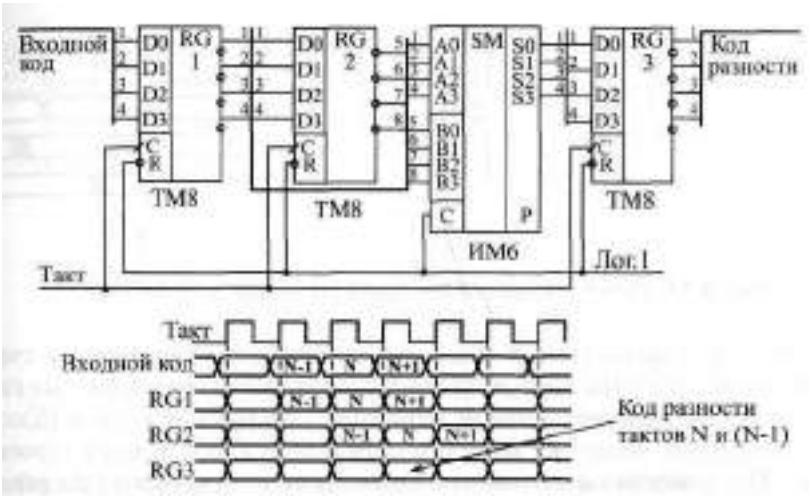


Рис. 4.21. Схема вычисления разности значений кодов в двух последовательных тактах

Для примера на рис. 4.21 показана схема вычисления разности двух последовательных значений входного кода. Такая задача возникает, в частности, при цифровой обработке аналоговых сигналов. Последовательные значения входного 4-разрядного кода сопровождаются тактовым сигналом, по положительному фронту которого производится запись в два последовательно включенных регистра. Когда на выходе регистра RG1 присутствует N-ое значение входного кода, на выходе регистра RG2 будет (N–1)-ое значение этого же кода.

Подавая эти два кода с выходов регистров на 4-разрядный сумматор, включенный в режиме вычитания (см. рис. 3.22), мы получаем на выходе сумматора код разности между N-ым значением и (N–1)-ым значением. В данном случае очень удобен регистр TM8, имеющий инверсные выходы. Для обеспечения строго одновременного изменения выходных сигналов сумматора можно включить дополнительный выходной регистр RG3, тактируемый тем же самым общим тактовым сигналом. Правда, код разности при этом будет задержан на один такт.

Регистры также широко используются для организации конвейерной обработки, позволяющей существенно повысить тактовую частоту работы схемы. Ускорение при этом достигается за счет распараллеливания работы нескольких последовательно включенных узлов схемы.

Пусть, например, последовательность входных кодов, следующих с периодом T, поступает на вход цепочки из двух узлов, производящих обработку или преобразование этих кодов (рис. 4.22). Узлы эти могут представлять собой комбинационные микросхемы (например, сумматоры)

Рис. 4.22. Работа последовательной цепочки двух узлов

или более сложные устройства, включающие в себя микросхемы счетчиков или микросхемы памяти. Главное состоит в том, что выходные сигналы этих узлов выставляются не мгновенно, а в течение какого-то конечного времени, величина которого определяется внутренним строением узла. Пусть задержка установления выходного кода первого узла равняется t_1, а задержка установления выходного кода второго узла составляет t_2. Очевидно, что период следования входных кодов T не должен быть меньше, чем сумма этих двух задержек:

$$T > t_1 + t_2$$

Иначе код на выходе цепочки может никогда не принять устойчивого значения, так как переходный процесс предыдущего такта будет сменяться переходным процессом следующего такта. То есть быстродействие узлов накладывает жесткое ограничение на тактовую частоту.

Однако можно обойти это ограничение, если воспользоваться принципом конвейера, заставить узлы работать не последовательно, а параллельно. Это достигается включением между узлами регистра, тактируемого входным тактовым сигналом. Еще один регистр целесообразно включить на входе второго узла, что обеспечит длительность устойчивого кода на выходе всего устройства, равную длительности периода тактового сигнала T (рис. 4.23). В результате ограничение на период тактового сигнала становится более мягким: T не должно быть меньше максимальной из двух величин t_1 и t_2 с добавлением времени задержки регистра:

$$T > \max(t_1, t_2) + t_{RG}$$

/

То есть к следующему фронту тактового сигнала должен закончить свою работу самый медленный из узлов, и тогда его выходной код будет записан в регистр правильно.

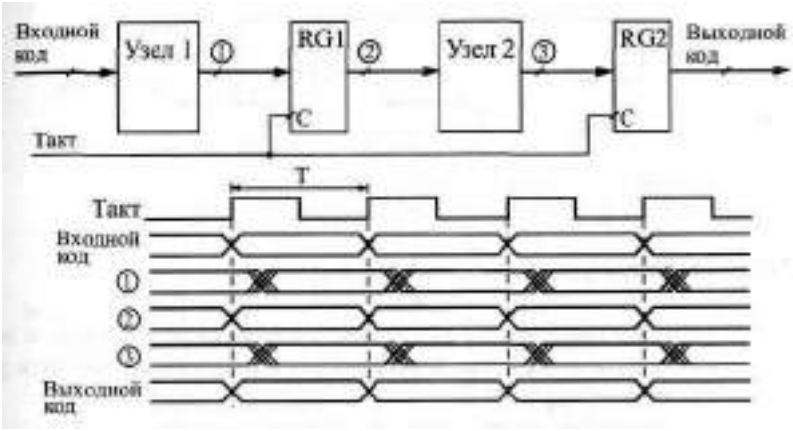


Рис. 4.23. Конвейерная обработка с помощью регистров

Точно так же можно построить конвейер на любое количество последовательно включенных узлов. Конечно, в результате введения конвейера происходит задержка выполнения полной функции устройства на число тактов, равное числу введенных регистров. Однако в том случае, когда необходимо обрабатывать большие последовательности входных кодов, эта задержка наблюдается только один раз — в самом начале последовательности, а затем уже она не имеет значения.

Регистры могут также применяться в составе вычислителей, выполняя функцию накопителя результата вычисления. В данном случае мы уже имеем дело с более сложной обработкой информации, чем при чисто комбинационных схемах. С каждым тактом в регистре обновляется содержимое, являющееся результатом математической обработки входного кода и результата предыдущего вычисления.

Рассмотрим два примера схем таких вычислителей.

Первая схема известна как **накапливающий сумматор,** применяющийся, например, в цифровых генераторах аналоговых сигналов. В самом названии схемы отражена ее функция: она суммирует и накапливает результат. Накапливающий сумматор (рис. 4.24) состоит из сумматора и выходного регистра, охваченных обратной связью.

То есть на один вход сумматора подается код с выходов регистра, а на другой вход — входной код. В результате с каждым следующим фронтом тактового сигнала в регистр записывается код суммы входного кода с предыдущим содержимым регистра, с предыдущей суммой. Например, если входной код равен 3, а в регистре записан код 6, то в следующем такте в регистр будет записан код 9 (то есть 6 + 3), в следующем такте — код 12 (то есть 9 + 3) и т. д. Получается, что на выходе накапливающего сум-

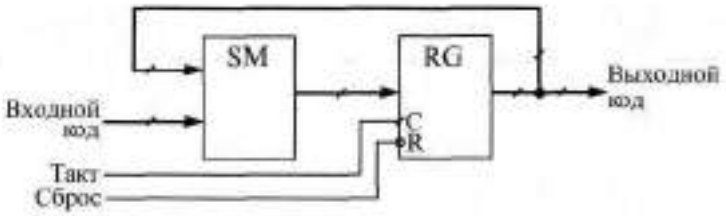


Рис. 4.24. Структура накапливающего сумматора

матора формируется равномерно увеличивающийся двоичный код, и шаг этого увеличения можно менять. В данном случае удобно применять регистр со сбросом, например, ИР35.

Отметим три особенности накапливающего сумматора.

Во-первых, когда выходной код достигает максимальной величины (становится больше 2^n, где n — количество разрядов регистра), происходит переполнение схемы и возобновление ее работы с минимальных значений кода. Однако совсем не обязательно в следующем цикле работы будут повторены те же значения кода, что и в предыдущем. Например, пусть n равняется 4, то есть максимальное число на выходе регистра равно 1111 в двоичном коде или 15 в десятичном коде. Пусть входной код равен 3. Тогда после начального сброса регистра выходной код будет нарастать так: 0, 3, 6, 9, 12, 15, 2, 5, 8, 11, 14, 1, 4, ... Это происходит потому, что суммирование чисел 15 и 3 даст 18 или, в двоичном коде, 10010, а так как мы работаем только с младшими четырьмя разрядами, у нас получится 0010 или 2.

Во-вторых, особенность накапливающего сумматора состоит в том, что при больших значениях входного кода (больших половины максимально возможной величины) он может рассматриваться как накапливающий вычитатель. Пусть, например, входной код 4-разрядного сумматора равен 15 (1111 в двоичном коде), а в регистре записано число 13 (1101 в двоичном коде). В следующем такте в регистр запишется сумма 1101 + 1111 = 11100, а без старшего разряда — 1100, то есть 12. То есть выходной код уменьшился на единицу.

Наконец, в-третьих, совсем не обязательно шаг нарастания выходного кода накапливающего сумматора должен быть целым числом (то есть 0, 1, 2, 3, . . .). Если в качестве выходного кода берутся не все, а только старшие разряды регистра, то шаг нарастания вполне может быть дробным, например, 0,5, 1,25 или 3, 75. Не вошедшие в выходной код разряды будут иметь вес 2^{-1} (то есть 0, 5), 2^{-2} (то есть 0, 25) и т. д. Правда результат суммирования в выходном коде будет представлен с точностью до целых чисел. При этом возможна ситуация, когда в течение не-

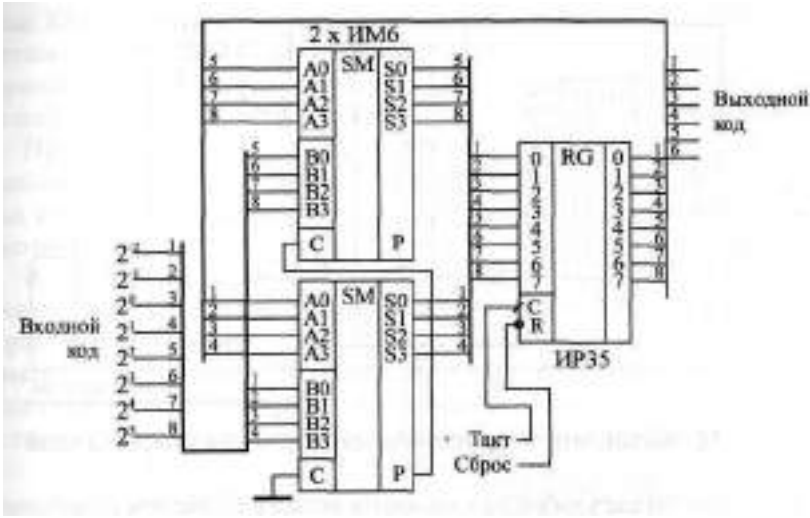


Рис. 4.25. Накапливающий сумматор

скольких тактов код на выходе не меняется, например, при входном коде 0,5 выходной код будет меняться один раз на два такта, а при входном коде 0,25 — один раз на четыре такта.

На рис. 4.25 показана схема 8-разрядного накапливающего сумматора на двух микросхемах сумматоров ИМ6 и одном регистре ИР35. В качестве выходного кода используется только 6 старших разрядов с выхода регистра, поэтому задание шага приращения возможно с точностью до 0,25. Максимально возможная частота тактового сигнала может быть определена по формуле

$$T > t_{SM} + t_{RG},$$

где T — период тактового сигнала, t_{SM} — задержка 8-разрядного сумматора, а t_{RG} — задержка регистра.

Последний пример применения регистров, который мы рассмотрим, — это вычислитель максимального значения входного кода. Такой вычислитель, например, может применяться в схемах цифровых осциллографов для измерения амплитуды входного аналогового сигнала.

Пусть мы имеем последовательность входных кодов, и нам необходимо выявить экстремальный (то есть максимальный или минимальный) код из всей этой последовательности. Эта задача решается довольно просто путем применения компаратора кодов и регистра, охваченных обратной связью (рис. 4.26).

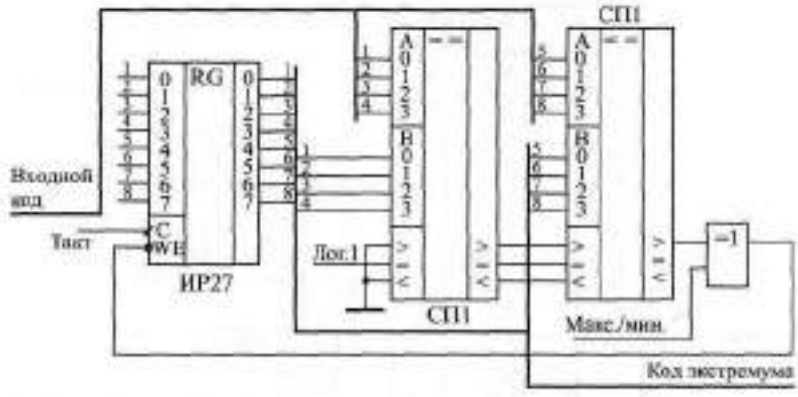


Рис. 4.26. Вычислитель экстремального значения входного кода

В данном случае удобно использовать регистр со входом разрешения записи (ИР27). В регистре сохраняется код экстремума (максимума или минимума), а 8-разрядный компаратор, составленный из двух микросхем СП1, сравнивает содержимое регистра и текущее значение входного кода. Элемент Исключающее ИЛИ выполняет функцию управляемого инвертора, выбирая режим вычисления максимума (единица на управляющем входе) или минимума (нуль на управляющем входе).

Допустим, мы вычисляем максимум. При этом запись в регистр текущего значения входного кода будет производиться только в том случае, когда это текущее значение больше числа, содержащегося в регистре. На выходе ">" компаратора кодов будет тогда сигнал логической единицы, а на входе разрешения записи регистра -WE — сигнал логического нуля. Если же текущее значение входного кода меньше кода, содержащегося в регистре, запись не производится. После окончания входной последовательности кодов (или после окончания одного ее периода при периодической последовательности) в регистре останется максимальное значение входного кода.

Аналогично вычисляется и минимум, только в данном случае в регистр будет записываться не только код, меньший числа в регистре, но и код, равный этому числу. Понятно, что на конечный результат вычисления это никак не повлияет.

4.2.2. Регистры, срабатывающие по уровню

Параллельные регистры, срабатывающие по уровню стробирующего сигнала (или, как их еще называют, регистры-защелки, английское "Latch"), можно рассматривать как некий гибрид между буфером и реги-

стром. Когда сигнал на стробирующем входе — единичный, такой регистр пропускает через себя входные информационные сигналы, а когда стробирующий сигнал становится равен нулю, регистр переходит в режим хранения последнего из пропущенных значений входных сигналов.

Применение таких регистров сильно ограничено, хотя иногда они довольно удобны. В некоторых схемах они могут успешно заменять регистры, срабатывающие по фронту, а в других схемах их применение вместо регистров, срабатывающих по фронту, недопустимо.

В стандартных сериях регистры, срабатывающие по уровню, представлены гораздо меньше, чем регистры, срабатывающие по фронту. На рис. 4.27 показаны в качестве примеров две микросхемы 4-разрядного регистра ТМ7 и 8-разрядного регистра ИР22. Стробирующие входы С нередко на схемах обозначают Е (от английского "Enable" — "разрешение"), для того чтобы не путать их с тактовыми входами D-триггеров.

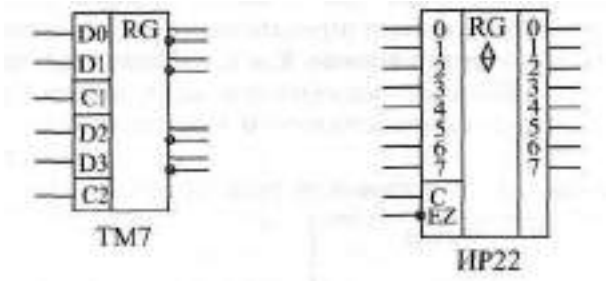


Рис. 4.27. Регистры, срабатывающие по уровню

Микросхему ТМ7 (и близкую к ней ТМ5) часто называют набором триггеров, но ее можно рассматривать и как регистр. Микросхема состоит из четырех триггеров, стробирующие входы которых С соединены попарно, то есть можно говорить о двух двухразрядных регистрах-защелках. Входы С1 и С2 микросхемы управляют каждый двумя разрядами данных. Все триггеры имеют как прямые, так и инверсные выходы, что иногда очень удобно. Таблица истинности микросхемы ТМ7 приведена в табл. 4.7.

При единице на входе С выходные сигналы повторяют входные, то есть регистр работает как обычный буфер с прямыми и инверсными выходами. При нуле на входе С на выходе регистра постоянно хранится та входная информация, которая была в момент прихода отрицательного фронта сигнала С. Однако говорить, что регистр ТМ7 срабатывает по отрицательному фронту сигнала С, неверно, так как информация на выходе меняется не только по этому фронту, но и в момент изменения входных сигналов при С = 1.

Таблица 4.7. Таблица истинности регистра ТМ7

Входы		Выходы	
D	C	Q	-Q
0	1	0	1
1	1	1	0
0	0	Не меняется	
1	0	Не меняется	

Регистр ИР22 отличается от ТМ7 тем, что имеет выходы с тремя состояниями (и соответственно, вход разрешения всех выходов -EZ) и тем, что всеми восемью разрядами управляет один стробирующий сигнал С. Суть работы от этого не изменяется. При единице на входе С регистр работает как буфер-повторитель, а при нуле на входе С — хранит ту информацию, которая была на входе в момент отрицательного фронта сигнала С. Выходы у регистра ИР22 — только прямые. Как и все регистры с тремя состояниями выхода, ИР22 имеет повышенную нагрузочную способность. В табл. 4.8 приведена таблица истинности регистра ИР22.

Таблица 4.8. Таблица истинности регистра ИР22

Входы			Выход
-EZ	C	D	Q
0	1	1	1
0	1	0	0
0	0	X	Не меняется
1	X	X	Z

Величины задержек триггеров, срабатывающих по уровню, в 1,5—2 раза превышают задержки D-триггеров. Для правильной работы микросхем положительный импульс на входе С не должен быть слишком коротким, а задержка между изменением информации на входе D и отрицательным фронтом сигнала С не должна быть слишком малой. Информация на входе D не должна слишком быстро сниматься после отрицательного фронта сигнала С.

Основное применение регистра, срабатывающего по уровню стробирующего сигнала, состоит в запоминании на какое-то заданное время входного кода, причем в остальное время выходной код регистра должен повторять входной (рис. 4.28). Стробирующий сигнал С в этом случае должен быть отрицательным на все время запоминания, и запоминаться бу-

дет входной код регистра в момент отрицательного (переднего) фронта сигнала С. Подобная функция бывает, например, необходима при построении устройств сопряжения для компьютеров. Регистр, по сути, продлевает во времени необходимое значение входного кода, в остальное время работая как повторитель.

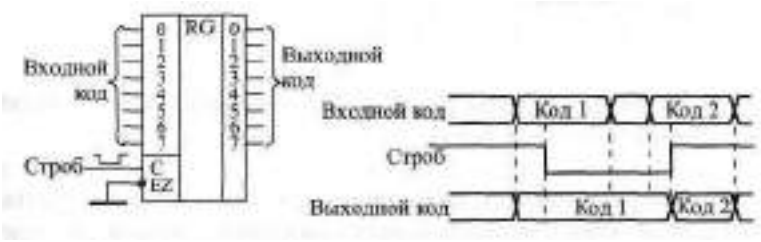


Рис. 4.28. Продление длительности входного кода с помощью регистра-защелки

В ряде случаев регистры данного типа могут успешно заменять регистры, срабатывающие по фронту. Например, такая замена возможна в случае необходимости запоминания входного кода по сигналу С до момента прихода следующего сигнала С (рис. 4.29).

Сигнал С в данном случае должен быть коротким положительным импульсом, причем он обязательно должен быть "вложен" в запоминаемый входной код, то есть начинаться *после* начала (момента установления) кода, а заканчиваться *до* конца (момента снятия) кода (это так называемый вложенный цикл). По переднему фронту сигнала С регистр перейдет в режим пропускания входного кода, а по заднему — в режим его хранения. Поэтому записываемый код на выходе регистра появится по положительному фронту сигнала С, то есть точно так же, как и в случае регистра, срабатывающего по фронту.

Однако подобная замена регистра, срабатывающего по фронту, на регистр, срабатывающий по уровню, возможна далеко не всегда. Некоторые схемы в принципе не могут работать с регистром-защелкой даже

Рис. 4.29. Использование регистра-защелки для замены регистра, срабатывающего по фронту

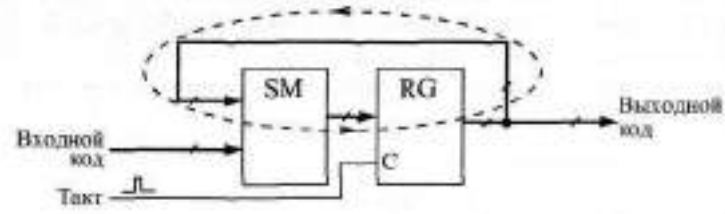


Рис. 4.30. Лавинообразная обратная связь в накапливающем сумматоре с регистром-защелкой

при очень коротком сигнале на входе С. Примером может служить уже упоминавшаяся схема накапливающего сумматора, которая работает исключительно с регистром, срабатывающим по фронту. Ведь при единичном сигнале на входе С регистр-защелка тут же перейдет в состояние пропускания входного кода, и в результате замкнется лавинообразная обратная связь: код с выхода регистра будет складываться со входным кодом бесконечное число раз (рис. 4.30). Конечно, при коротком импульсе на входе С этот неуправляемый процесс быстро прекратится, но что за информация в результате останется в регистре после окончания сигнала С, предсказать невозможно.

И таких схем, принципиально не допускающих применения регистра-защелки, довольно много. Именно поэтому использование их сильно ограничено, а выбор микросхем в стандартных сериях невелик.

4.2.3. Сдвиговые регистры

Регистры сдвига или сдвиговые регистры (англ. shift register) представляют собой, как уже отмечалось, последовательно соединенную цепочку триггеров. Основной режим их работы — это сдвиг разрядов кода, записанного в эти триггеры, То есть по тактовому сигналу содержимое каждого предыдущего триггера переписывается в следующий по порядку в цепочке триггер. Код, хранящийся в регистре, с каждым тактом сдвигается на один разряд в сторону старших разрядов или в сторону младших разрядов, что и дало название регистрам данного типа.

В связи с названием направления сдвига в сдвиговых регистрах часто возникает путаница. Сдвиг бывает двух видов: вправо (основной режим, который есть у всех сдвиговых регистров) и влево (этот режим есть только у некоторых, реверсивных сдвиговых регистров). Названия эти отражают внутреннюю структуру регистров сдвига (рис. 4.31) и перезапись сигналов последовательно по цепочке триггеров. При этом триггеры, вполне естественно, нумеруются слева направо, например, от 0 до 7 (или

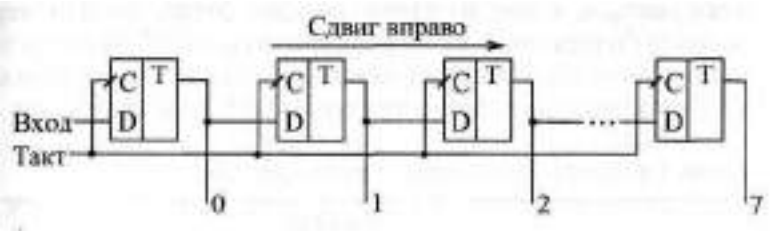


Рис. 4.31. Направление сдвига в сдвиговых регистрах

от 1 до 8) для 8-разрядных регистров. В результате сдвиг информации регистром вправо представляет собой сдвиг в сторону разрядов, имеющих большие номера, а сдвиг информации регистром влево — это сдвиг в сторону разрядов, имеющих меньшие номера.

Однако, как известно, в любом двоичном числе слева расположены старшие разряды, а справа — старшие разряды. Поэтому сдвиг двоичного числа вправо будет сдвигом в сторону младших разрядов, а сдвиг влево — сдвигом в сторону старших разрядов. Это противоречие, не чей-то злой умысел, просто так исторически сложилось, и об этом надо помнить разработчику цифровой аппаратуры.

В стандартные серии цифровых микросхем входит несколько типов сдвиговых регистров, отличающихся возможными режимами работы, режимами записи, чтения и сдвига, а также типом выходных каскадов (2С или 3С). Большинство регистров сдвига имеет восемь разрядов. На рис. 4.32 представлены для примера четыре типа микросхем регистров сдвига.

Регистр ИР8 — наиболее простой из регистров сдвига. Он представляет собой 8-разрядную линию задержки, то есть имеет только один информационный вход, на который подается последовательная сдвигаемая информация (точнее, два входа, объединенных по функции 2И), и восемь па-

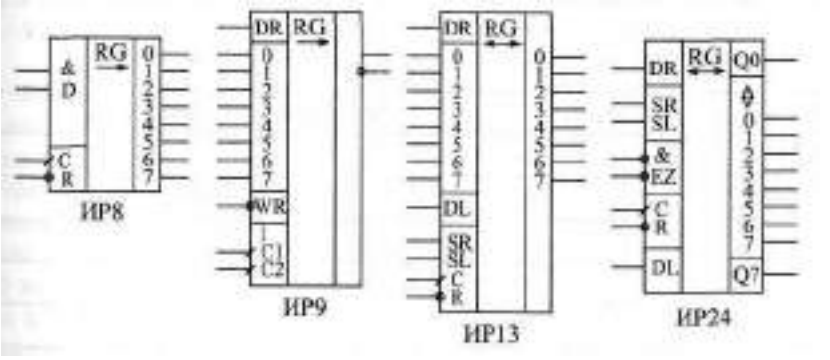


Рис. 4.32. Сдвиговые регистры

раллельных выходов. Сдвиг в сторону выходов со старшими номерами осуществляется по переднему фронту тактового сигнала С. Имеется также вход сброса -R, по нулевому сигналу на котором все выходы регистра сбрасываются в нуль. Таблица истинности регистра ИР8 приведена в табл. 4.9.

Таблица 4.9. Таблица истинности регистра сдвига ИР8

Входы				**Выходы**			
-R	**C**	D1	D2	Q0	Q1		Q7
0	**X**	X	**X**	0	0		0
1	0	X	**X**	Не меняются			
1	1	X	X	Не меняются			
1	0->1	1	1	1	Q0		Q6
1	0->1	0	X	0	Q0		Q6
1	0->1	X	0	0	Q0		Q6

Регистр ИР9 выполняет функцию, обратную регистру ИР8. Если ИР8 преобразует входную последовательную информацию в выходную параллельную, то регистр ИР9 преобразует входную параллельную информацию в выходную последовательную. Однако суть сдвига не меняется, просто в ИР9 все внутренние триггеры имеют выведенные параллельные входы, и только один, последний триггер имеет выход (причем как прямой, так и инверсный). Запись входного кода в регистр производится по нулевому сигналу на входе WR. Сдвиг осуществляется по положительному фронту на одном из двух тактовых входов С1 и С2, объединенных по функции 2ИЛИ. Имеется также вход расширения DR, сигнал с которого в режиме сдвига перезаписывается в младший разряд сдвигового регистра. Таблица истинности регистра ИР9 приведена в табл. 4.10.

Таблица 4.10. Таблица истинности регистра сдвига ИР9

Входы			**Функция**
-WR	C1	C2	
0	X	X	Параллельная запись
1	1	X	Хранение
1	**X**	1	Хранение
1	0	0->1	Сдвиг
1	0->1	0	Сдвиг

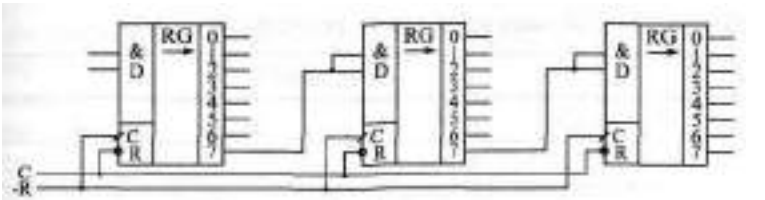


Рис. 4.33. Соединение регистров ИР8 для увеличения разрядности

Как и все остальные сдвиговые регистры, ИР8 и ИР9 допускают каскадирование, то есть совместное включение для увеличения разрядности. На рис. 4.33 показано объединение трех ИР8, а на рис 4.34 — совместное включение трех ИР9. В обоих случаях в результате объединения получается 24-разрядный сдвиговый регистр. При этом увеличение разрядности не приводит к увеличению задержки сдвига, так как тактовые входы всех используемых регистров объединяются параллельно. В случае регистров ИР8 входной последовательный код преобразуется в 24-разрядный выходной параллельный код. В случае регистров ИР9 входной 24-разрядный параллельный код преобразуется в выходной последовательный код.

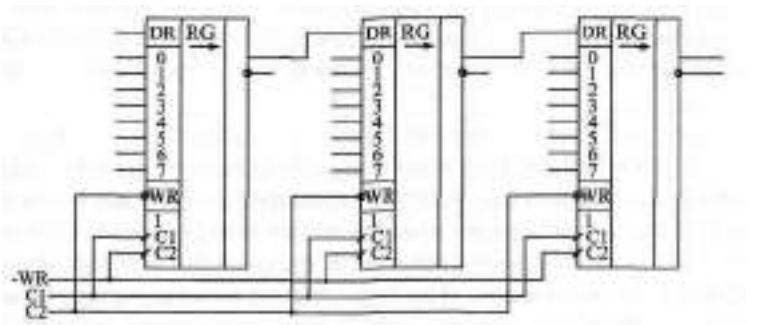


Рис. 4.34. Соединение регистров ИР9 для увеличения разрядности

Регистр ИР13 соединяет в себе возможности регистров ИР8 и ИР9. Он имеет как восемь входов для параллельной записи, так и соответствующие им восемь выходов параллельной информации. Сдвиг осуществляется по положительному фронту тактового сигнала С, причем сдвиг возможен как в сторону старших разрядов (вправо), так и в сторону младших разрядов (влево). Для наращивания разрядности у регистра ИР13 имеются последовательные информационные входы DR и DL, сигналы с которых вдвигаются, соответственно, в младший и в старший разряды. Предусмотрен сброс всех выходов регистра в нуль по нулевому сигналу на входе -R.

Таблица 4.11. Таблица истинности регистра ИР13

Входы				Функция
C	-R	SR	SL	
X	0	X	X	Сброс
0->1	1	1	0	Сдвиг вправо
0->1	1	0	1	Сдвиг влево
0->1	1	0	0	Хранение
0->1	1	1	1	Параллельная запись

Режим работы регистра ИР13 определяется двумя управляющими входами SR и SL. При единице на входе SR и нуле на входе SL по фронту сигнала С происходит сдвиг в сторону старших разрядов. При нуле на входе SR и единице на входе SL по фронту сигнала С происходит сдвиг в сторону младших разрядов. При обоих единичных сигналах на входах SR и S L по фронту сигнала С происходит параллельная загрузка информации в регистр. Таблица истинности регистра ИР13 приведена в табл. 4.11.

Отметим, что регистр ИР13 применяется заметно реже, чем более простые регистры ИР8 и ИР9, так как задач, в которых были бы нужны все возможности ИР13, не так уж много, а управление работой ИР13 — довольно сложное.

Наконец, последний сдвиговый регистр, который мы рассмотрим подробнее, это регистр ИР24. По своим возможностям он близок к ИР13, однако его главной особенностью является двунаправленная параллельная шина данных. То есть одни и те же выводы микросхемы используются как для параллельной записи информации в регистр, так и для параллельного чтения информации из регистра. При этом двунаправленные выводы данных имеют повышенную нагрузочную способность. Это позволяет легко сопрягать ИР24 с многоразрядными микросхемами памяти и с двунаправленными буферами. Поэтому применяется данный регистр чаще, чем ИР13.

Регистр ИР24 обеспечивает сдвиг информации в обоих направлениях. Имеются входы расширения DR и DL, а также выходы расширения Q0 и Q7, что позволяет легко наращивать разрядность. Отличие выходов Q0 и Q7 от нулевого и седьмого разрядов данных состоит в том, что Q0 и Q7 — однонаправленные, то есть в любом режиме работы выдают информацию с выходов внутренних триггеров младшего и старшего разрядов. Тактируется регистр положительным фронтом сигнала С. Предусмотрен сброс регистра нулевым сигналом на входе -R.

Режим работы микросхемы определяется сигналами на управляющих входах SR и SL.

При единичном сигнале на SR и нулевом сигнале на SL по положительному фронту сигнала С происходит сдвиг информации вправо (в сторону разрядов с большими номерами). Запись в разряд 0 производится при этом со входа расширения DR

При единичном сигнале на SL и нулевом сигнале на SR по положительному фронту сигнала С происходит сдвиг информации влево (в сторону разрядов с меньшими номерами). Запись в разряд 7 производится при этом со входа расширения DL

При обоих нулях на входах SR и SL регистр переходит в режим хранения. Во всех этих случаях разряды данных работают как вход или как выход в зависимости от сигналов -EZ.

Таблица 4.12. Таблица истинности регистра ИР24

Входы				Функция
-R	C	SR	SL	
0	X	X	X	Сброс
1	0->1	1	0	Сдвиг вправо
1	0->1	0	1	Сдвиг влево
1	0->1	1	1	Параллельная запись
1	X	0	0	Хранение

При обеих единицах на входах SR и SL по положительному фронту С в регистр записывается параллельный код, причем разряды данных переходят в состояние приема независимо от сигналов -EZ. Таблица истинности регистра ИР24 приведена в табл. 4.12.

Объединяя два регистра ИР24, легко получить 16-разрядный сдвиговый регистр с сохранением всех возможностей одной микросхемы (рис. 4.35). Точно так же можно объединять и большее количество микросхем.

Главное применение всех регистров сдвига состоит в преобразовании параллельного кода в последовательный, и наоборот. Такое преобразование используется, например, при передаче информации на большие расстояния (в информационных сетях), при записи информации на магнитные носители, при работе с телевизионными мониторами и с видеокамерами, а также во многих других случаях.

Для примера на рис. 4.36 показана простейшая схема передачи цифровой информации в последовательном коде по двум линиям: информационной и синхронизующей. Такая передача позволяет сократить количество соединительных проводов, а также упростить защиту передавае-

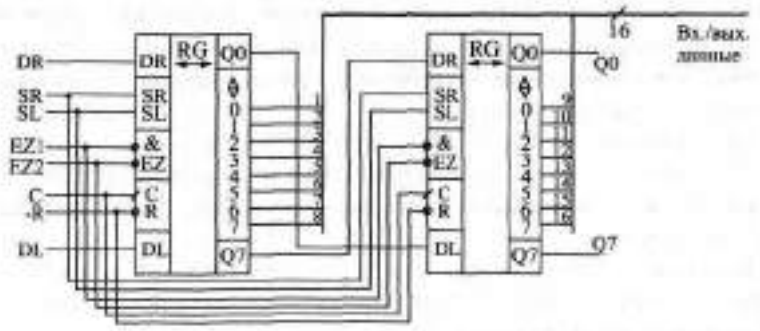


Рис. 4.35. Объединение регистров ИР24 для увеличения разрядности

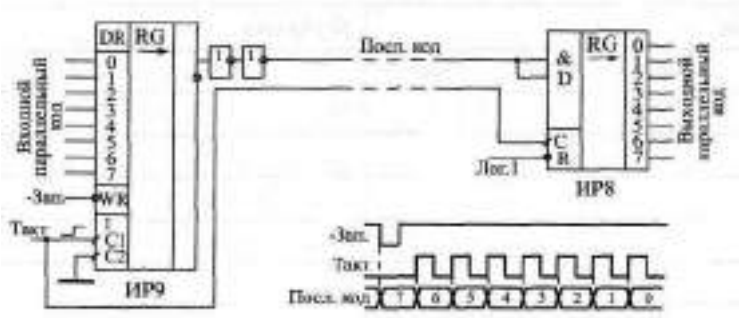


Рис. 4.36. Последовательная передача информации с помощью регистров сдвига

мых данных от действия внешних электромагнитных помех, правда, ценой снижения скорости передачи.

На передающем конце (слева на рисунке) с помощью сдвигового регистра ИР9 входной параллельный 8-разрядный код преобразуется в последовательность разрядов данных, следующих с частотой тактового сигнала. На приемном конце (справа на рисунке) с помощью сдвигового регистра ИР8 эта последовательность разрядов данных снова преобразуется в параллельный код. Оба регистра тактируются одним и тем же тактовым сигналом, который передается по линии связи параллельно с последовательностью данных. Для увеличения надежности передачи информационный сигнал дополнительно задерживается относительно фронта тактового сигнала с помощью цепочки из двух инверторов.

Первый бит последовательного входа (со входа 7 регистра ИР9) начинает передаваться с началом сигнала записи Зап. Следующие разряды передаются с каждым следующим положительным фронтом тактового сиг-

нала С. Последним передается сигнал со входа 0. В регистр ИР8 разряды последовательного кода записываются в том же самом порядке, в каком они были в регистре ИР9. По окончании передачи первый переданный сигнал данных окажется в разряде 7 шины данных регистра ИР8, а последний переданный сигнал данных — в разряде 0.

Следующее применение сдвиговых регистров состоит в организации всевозможных линий задержек, особенно имеющих значительное количество каскадов. С помощью сдвиговых регистров можно обеспечить задержку любого входного сигнала на целое число тактов. Правда, надо учитывать, что длительность входного сигнала (и любого его элемента) будет также передаваться по линии задержки с точностью до одного такта. Такие линии задержки могут применяться для сравнения нескольких последующих тактов входного сигнала, для выполнения арифметических операций с несколькими тактами входного сигнала и для других подобных целей. Работа линии задержки на регистре сдвига иллюстрируется рис. 4.37.

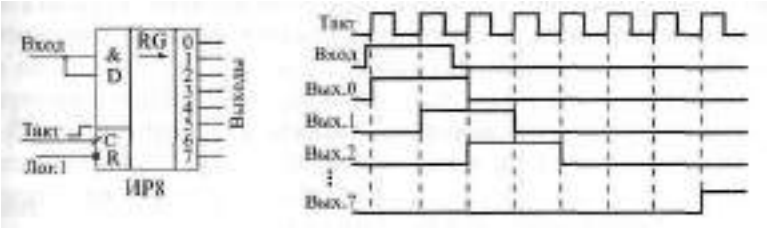


Рис. 4.37. Линия задержки входного сигнала на регистре сдвига

Сдвиговые регистры могут также использоваться для формирования импульсов заданной длительности, причем длительность импульса может задаваться управляющим кодом, то есть быть программно управляемой. На рис. 4.38 приведена возможная схема такого формирователя.

В исходном состоянии (до прихода положительного фронта входного сигнала) триггер сброшен в нуль, на всех выходах регистра сдвига — нули, на инверсном выходе мультиплексора — единица. На мультиплексор подан управляющий код, определяющий длительность выходного сигнала. При поступлении положительного фронта входного сигнала триггер перебрасывается в единицу (начинается выходной сигнал), и этот единичный сигнал начинает последовательно сдвигаться регистром сдвига по каждому фронту тактового сигнала.

Пусть управляющий код равен 5. Тогда в тот момент, когда на выходе 5 сдвигового регистра появится единица, она будет передана на выход мультиплексора КП7 с инверсией. При этом нулевой сигнал на входе -R триггера сбросит триггер в нуль, то есть выходной сигнал закончится.

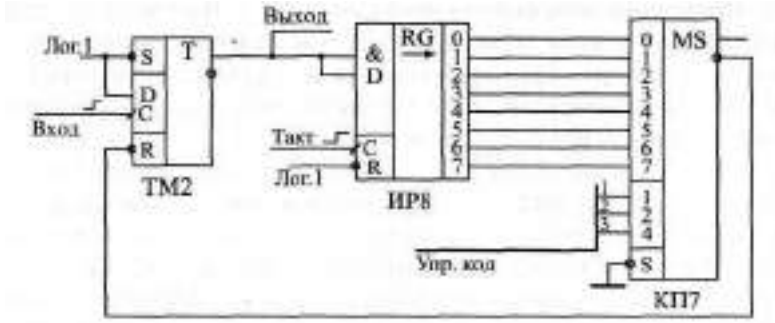


Рис. 4.38. Формирователь импульсов с длительностью, задаваемой управляющим кодом

Таким образом, длительность выходного сигнала будет определяться управляющим кодом. Погрешность установки этой длительности равна одному периоду тактового сигнала и зависит от временного сдвига между фронтом входного сигнала и фронтом ближайшего к нему тактового импульса. Чем больше длительность выходного сигнала, тем меньше относительная погрешность установки его точности. Например, при управляющем коде 0 длительность выходного сигнала может быть от 0 до T, где T — период тактового сигнала. А при управляющем коде 7 длительность выходного сигнала будет от 7T до 8T. При этом мы не учитываем задержек триггера, сдвигового регистра и мультиплексора.

Сдвиговые регистры могут также использоваться для умножения и деления двоичных чисел на 2^n, где n — целое число, большее нуля. Сдвиг двоичного числа вправо (в сторону младших разрядов) на один разряд равносилен делению на 2. Сдвиг двоичного числа влево (в сторону старших разрядов) на один разряд равносилен умножению на 2. Для того чтобы сдвиговый регистр умножал и делил двоичный код, надо всего лишь записать этот код в регистр и сдвинуть его нужное количество раз вправо или влево. Наиболее удобен для этого регистр ИР13. При этом необходимо, чтобы в освободившиеся разряды вдвигались нули, то есть на входы расширения DR и DL регистра надо подать нулевые сигналы.

Наконец, последнее применение сдвигового регистра, которое мы рассмотрим, — это генератор случайной последовательности сигналов или случайной последовательности кодов. Строго говоря, последовательности будут не полностью случайные, а квазислучайные, то есть будут периодически повторяться, но период этот довольно большой. Случайные последовательности сигналов и кодов широко применяются в тестирующей аппаратуре, в генераторах шума, в логических игровых устройствах.

Задача состоит в том, чтобы выходной сигнал или код менял свое состояние случайно (или почти случайно). Сигнал должен случайно переключаться из 0 в 1 и из 1 в 0, а код должен случайно принимать значения из диапазона от 0 до (2^N-1), где N — число разрядов кода (например, от 0 до 255 при 8-разрядном коде). Псевдослучайные последовательности имеют то преимущество перед истинно случайными, что они — предсказуемые и периодические, но в этом же и их недостаток.

Структура генератора квазислучайной последовательности на сдвиговом регистре очень проста (рис. 4.39). Она представляет собой регистр сдвига с параллельными выходами (например, ИР8), несколько (минимум два) выходных сигналов которого объединены с помощью элемента Исключающее ИЛИ, с выхода которого сигнал подается на вход регистра, замыкая схему в кольцо. Схема тактируется сигналом с частотой fу.

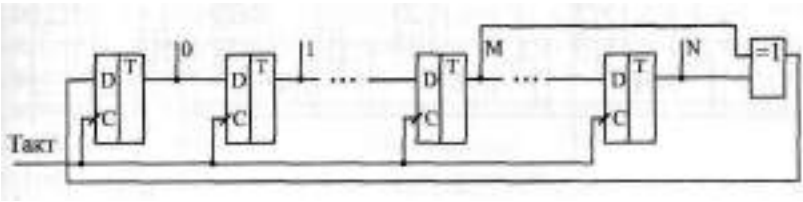


Рис. 4.39. Структура генератора псевдослучайной Последовательности

Таблица 4.13. Точки подключения обратной связи

N	7	8	15	16	24	31
Выходы	6. 5	7, 6, 4, 2	14, 13	15, 13, 10	23, 22, 21, 16	30, **17**

Выбор номеров разрядов для подключения обратной связи представляет собой непростую задачу, но существуют справочные таблицы, в которых они приведены. В любом случае одна из точек подключения — выход старшего разряда. В табл. 4.13 приведены точки подключения обратной связи для регистров сдвига с разным количеством разрядов N (номера разрядов считаются от нуля).

Из таблицы видно, что выгоднее брать число разрядов не кратное 8, например, 7, 15 или 31. В этом случае для обратной связи используются всего лишь два выхода, то есть достаточно одного двухвходового элемента Исключающее ИЛИ.

Период выходной последовательности генератора составляет (2^N-1) тактов, где N - количество разрядов регистра сдвига. За это время каждое из возможных значений выходного кода (кроме одного) встречается один раз. Количество единиц в выходном сигнале больше количества нулей на единицу.

Выходной код 000. . . 0 представляет собой запрещенное состояние, так как он блокирует работу генератора, воспроизводя сам себя снова и снова. Но в то же время получиться такой нулевой код может только сам из себя, поэтому достаточно обеспечить, чтобы его не было при включении питания схемы.

Частоты в спектре выходного сигнала будут следовать с интервалом $f_T/(2^n-1)$, а огибающая спектра будет практически постоянной до частоты $0,25f_T$, то есть шум до этой частоты можно считать белым (спад в 3 дБ происходит на частоте $0,45\ f_T$).

Рис. 4.40. 31-разрядный генератор псевдослучайной последовательности на регистрах сдвига

На рис. 4.40 показана практическая схема генератора псевдослучайной последовательности на 31-разрядном сдвиговом регистре. Обратная связь осуществляется с выходов 30 и 17 регистра через двухвходовой элемент Исключающее ИИ с инвертором. Из-за применения инвертора запрещенным состоянием генератора является код 1111. . .1 (а не код 000. . . 0), который в данном случае исключается очень просто — начальным сбросом регистров в нуль при включении питания по сигналу Сброс. Генератор выдает квазислучайную последовательность 31-разрядных кодов со всех выходов регистра, а также квазислучайную последовательность нулей и единиц на любом из выходов регистра. Такой генератор использовала известная фирма Hewlett—Packard в своем генераторе шума.

Глава 5. Применение счетчиков

Лекция 9. Асинхронные и синхронно-асинхронные счетчики

В лекции рассматриваются асинхронные счетчики и синхронные счетчики с асинхронным переносом, их алгоритмы работы, параметры, типовые схемы включения, а также способы реализации на их основе некоторых часто встречающихся функций.

Ключевые слова: двоичный счетчик, двоично-десятичный счетчик, прямой и инверсный счет, сброс, перенос, счетный вход, делитель частоты, меандр, параллельная запись, пачка импульсов, формирователь временных интервалов, измеритель длительности сигнала, измеритель частоты, переключение каналов, динамическая индикация, таймер, генератор импульсов.

Счетчики представляют собой более высокий, чем регистры, уровень сложности цифровых микросхем, имеющих внутреннюю память. Хотя в основе любого счетчика лежат те же самые триггеры, которые образуют и регистры, но в счетчиках триггеры соединены более сложными связями, в результате чего их функции — сложнее, и на их основе можно строить более сложные устройства, чем на регистрах. Точно так же, как и в случае регистров, внутренняя память счетчиков — оперативная, то есть ее содержимое сохраняется только до тех пор, пока включено питание схемы. С выключением питания память стирается, а при новом включении питания схемы содержимое памяти будет произвольным, случайным, зависящим только от конкретной микросхемы, то есть выходные сигналы счетчиков будут произвольными.

Как следует из самого названия, счетчики предназначены для счета входных импульсов. То есть с приходом каждого нового входного импульса двоичный код на выходе счетчика увеличивается (или уменьшается) на единицу (рис. 5.1). Срабатывать счетчик может по отрицательному фронту входного (тактового) сигнала (как на рисунке) или по положительному фронту. Режим счета обеспечивается использованием внутренних триггеров, работающих в счетном режиме. Выходы счетчика представляют собой как раз выходы этих триггеров. Каждый выход счетчика представляет собой разряд двоичного кода, причем разряд, переключающийся чаще других (по каждому входному импульсу), будет младшим, разряд, переключающийся реже других, — старшим.

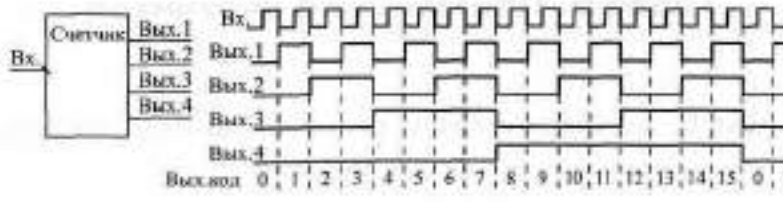


Рис. 5.1. Работа 4-разрядного двоичного счетчика

Счетчик может работать на увеличение выходного кода по каждому входному импульсу; это основной режим, имеющийся во всех счетчиках, он называется режимом прямого счета. Счетчик может также работать на уменьшение выходного кода по каждому входному импульсу; это режим обратного или инверсного счета, предусмотренный в счетчиках, называемых реверсивными. Инверсный счет бывает довольно удобен в схемах, где необходимо отсчитывать заданное количество входных импульсов.

Большинство счетчиков работают в обычном двоичном коде, то есть считают от 0 до (2^N-1), где N — число разрядов выходного кода счетчика. Например, 4-разрядный счетчик в режиме прямого счета будет считать от 0 (код 0000) до 15 (код 1111), а 8-разрядный - от 0 (код 0000 0000) до 255 (код 1111 1111). После максимального значения кода счетчик по следующему входному импульсу переключается опять в 0, то есть работает по кругу. Если же счет — инверсный, то счетчик считает до нуля, а дальше переходит к максимальному коду 111...1.

Имеются также двоично-десятичные счетчики, предельный код на выходе которых не превышает максимального двоично-десятичного числа, возможного при данном количестве разрядов. Например, 4-разрядный двоично-десятичный счетчик в режиме прямого счета будет считать от 0 (код 0000) до 9 (код 1001), а затем снова от 0 до 9. А 8-разрядный двоично-десятичный счетчик будет считать от 0 (код 0000 0000) до 99 (код 1001 1001). При инверсном счете двоично-десятичные счетчики считают до нуля, а со следующим входным импульсом переходят к максимально возможному двоично-десятичному числу (то есть 9 - для 4-разрядного счетчика, 99 — для 8-разрядного счетчика). Двоично-десятичные счетчики удобны, например, при организации десятичной индикации их выходного кода. Применяются они гораздо реже обычных двоичных счетчиков.

По быстродействию все счетчики делятся на три большие группы:

- Асинхронные счетчики (или последовательные).
- Синхронные счетчики с асинхронным переносом (или параллельные счетчики с последовательным переносом, синхронно-асинхронные счетчики).
- Синхронные счетчики (или параллельные).

Принципиальные различия между этими группами проявляются только на втором уровне представления, на уровне модели с временными задержками. Причем больше всего различия эти проявляются при каскадировании счетчиков. Наибольшим быстродействием обладают синхронные счетчики, наименьшим — асинхронные счетчики, наиболее просто управляемые среди других. Каждая группа счетчиков имеет свои области применения, на которых мы и остановимся.

5.1. Асинхронные счетчики

Асинхронные счетчики строятся из простой цепочки JK-триггеров, каждый из которых работает в счетном режиме. Выходной сигнал каждого триггера служит входным сигналом для следующего триггера. Поэтому все разряды (выходы) асинхронного счетчика переключаются последовательно (отсюда название - последовательные счетчики), один за другим, начиная с младшего и кончая старшим. Каждый следующий разряд переключается с задержкой относительно предыдущего (рис. 5.2), то есть, вообще говоря, асинхронно, не одновременно с входным сигналом и с другими разрядами.

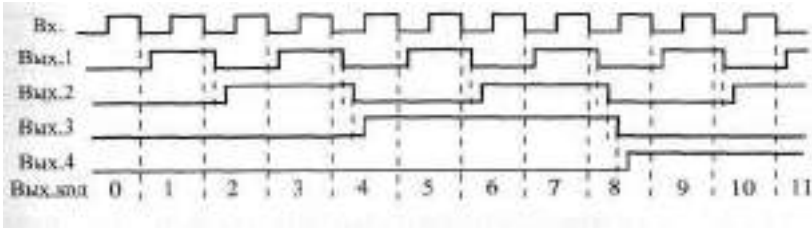


Рис. 5.2. Временная диаграмма работы 4-разрядного асинхронного счетчика

Чем больше разрядов имеет счетчик, тем большее время ему требуется на полное переключение всех разрядов. Задержка переключения каждого разряда примерно равна задержке триггера, а полная задержка установления кода на выходе счетчика равна задержке одного разряда, умноженной на число разрядов счетчика. Легко заметить, что при периоде входного сигнала, меньшем полной задержки установления кода счетчика, правильный код на выходе счетчика просто не успеет установиться, поэтому такая ситуация не имеет смысла. Это накладывает жесткие ограничения на период (частоту) входного сигнала, причем увеличение, к примеру, вдвое количества разрядов счетчика автоматически уменьшает вдвое предельно допустимую частоту входного сигнала.

Таким образом, если нам *нужен* выходной код асинхронного счетчика, то есть все его выходные сигналы (разряды) одновременно, то должно выполняться следующее неравенство:

$$T > N t_з,$$

где T — период входного сигнала, N — число разрядов счетчика, $t_з$ — время задержки одного разряда.

Надо еще учесть, что за период входного сигнала должно успеть сработать устройство (узел), на которое поступает выходной код счетчика, иначе счетчик просто не нужен; поэтому ограничение на частоту входного сигнала обычно бывает еще жестче.

В составе стандартных серий цифровых микросхем асинхронных счетчиков немного. Для примера на рис. 5.3 приведены три из них: 4-х разрядный двоично-десятичный счетчик ИЕ2, 4-х разрядный двоичный счетчик ИЕ5 и 8-и разрядный двоичный счетчик ИЕ19 (он же сдвоенный четырехразрядный счетчик).

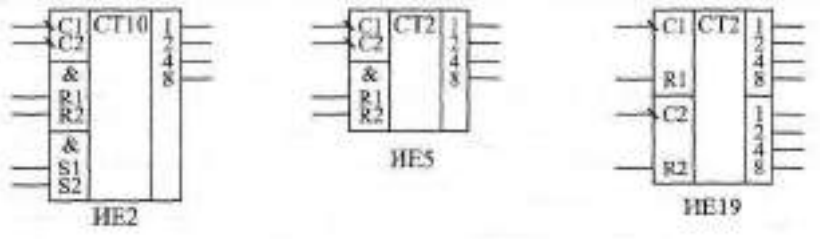


Рис. 5.3. Асинхронные счетчики стандартных серий

У всех этих счетчиков управление работой очень простое, есть всего лишь входы сброса в нуль или входы установки в 9 (только у ИЕ2). Все асинхронные счетчики работают по отрицательному фронту входного сигнала С (или, что то же самое, по заднему фронту положительного входного сигнала). У всех трех счетчиков выделены две независимые части, что увеличивает возможности их применения. При объединении этих двух частей получается счетчик максимальной разрядности. Выходы счетчиков обозначают на схемах 0, 1, 2, 3, . .. (как номера разрядов выходного двоичного кода) или 1, 2, 4, 8,... (как веса каждого разряда двоичного кода).

Счетчик ИЕ2 имеет две части: один триггер (вход С1, выход 1) и три триггера (вход С2 и выходы 2, 4, 8). Таким образом, он состоит из одноразрядного счетчика и трехразрядного счетчика. Одиночный триггер работает в обычном счетном режиме, изменяя свое состояние по каждо-

му отрицательному фронту сигнала C1, то есть делит частоту входного сигнала на 2. Три оставшихся триггера включены таким образом, чтобы считать до 5, то есть делить входную частоту сигнала C2 на 5. После достижения кода 4 (то есть 100) на выходах 2, 4 и 8 этот трехразрядный счетчик по следующему отрицательному фронту сигнала C2 сбрасывается в нуль. В результате при объединении выхода 1 микросхемы со входом C2 мы получаем 4-разрядный двоично-десятичный счетчик, делящий частоту входного сигнала C1 на 10 и сбрасывающийся в нуль после достижения на выходах 1, 2, 4, 8 кода 9 (то есть 1001) по отрицательному фронту сигнала C1.

Таблица 5.1. Таблица истинности счетчика ИЕ2

Входы					Выходы			
C1	R1	R2	S1	S2	8	4	2	1
X	1	1	0	X	0	0	0	0
X	1	1	X	0	0	0	0	0
X	X	X	1	1	1	0	0	1
1->0	X	0	X	0		Счет		
1->0	0	X	0	X		Счет		
1->0	0	X	X	0		Счет		
1->0	X	0	0	X		Счет		

Счетчик ИЕ2 имеет два входа асинхронного сброса в нуль R1 и R2, объединенных по функции И, и два входа установки в 9 — S1 и S2, также объединенных по функции И, причем установка в 9 блокирует установку в нуль. Наличие этих входов сброса и установки позволяет строить на базе счетчика ИЕ2 делители частоты с разными коэффициентами деления. Правда, этот счетчик используется довольно редко, значительно реже, чем другие асинхронные счетчики ИЕ5 и ИЕ19.

Таблица истинности асинхронного счетчика ИЕ2 при соединенном выходе 1 и входе C2 (при 4-разрядном выходном коде) приведена в табл. 5.1, а состояния выходов при счете входных импульсов по тактам представлены в табл. 5.2.

Счетчик ИЕ5, точно так же как и ИЕ2, имеет две части: один триггер (одноразрядный счетчик) со входом C1 и выходом 1 и три триггера (трехразрядный счетчик) со входом C2 и выходами 2, 4, 8. Оба счетчика — двоичные, то есть первый считает до двух, а второй — до 8. При объединении входа C2 с выходом 1 получается 4-разрядный двоичный счетчик, считающий до 16. Счет производится по отрицательному фронту входных сигна-

лов С1 и С2. Предусмотрена возможность сброса счетчика в нуль по сигналам R1 и R2, объединенным по функции И.

Таблица 5.2. Состояния выходов счетчика ИЕ2 при счете входных импульсов

Такт	Вых.8	Вых.4	Вых.2	Вых.1
0	**0**	**0**	**0**	**0**
1	**0**	**0**	**0**	**1**
2	**0**	**0**	**1**	**0**
3	**0**	**0**	**1**	**1**
4	**0**	**1**	**0**	**0**
5	**0**	**1**	**0**	**1**
6	**0**	**1**	**1**	**0**
7	**0**	**1**	**1**	**1**
8	**1**	**0**	**0**	**0**
9	**1**	**0**	**0**	**1**
10	**0**	**0**	**0**	**0**

Таблица 5.3. Таблица истинности счетчика ИЕ5

Входы			**Выходы**			
Cl	R1	R2	8	4	2	1
X	1	1	0	0	0	0
1-X)	0	X	Счет			
1-X)	X	0	Счет			

Таблица истинности счетчика ИЕ5 при соединении входа С2 и выхода 1 (при 4-разрядном выходном коде) приведена в табл. 5.3.

Объединять счетчики ИЕ5 для увеличения разрядности (каскадировать) очень просто: нужно выход 8 предыдущего счетчика (выдающего более младшие разряды) соединить со входом С1 следующего счетчика (выдающего более старшие разряды). На рис. 5.4 показано соединение трех счетчиков ИЕ5 для получения 12-разрядного асинхронного счетчика со сбросом в нуль. Точно так же можно объединять и счетчики ИЕ2, добавляя при этом входы общей установки счетчика в код 99.. . 9. Однако при объединении надо помнить, что добавление каждого нового разряда увеличивает общую задержку переключения полученного счетчика. Много-

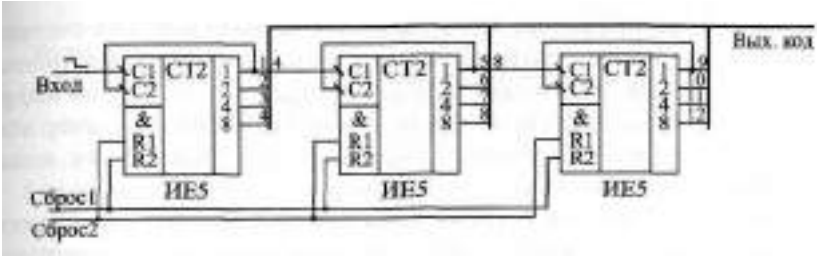


Рис. 5.4. Объединение трех счетчиков ИЕ5 для увеличения разрядности

разрядный асинхронный счетчик может получиться неприемлемо медленным.

Счетчик ИЕ19 можно считать сдвоенным вариантом счетчика ИЕ5. Он включает в себя два идентичных независимых друг от друга 4-разрядных асинхронных счетчика, каждый из которых имеет свой счетный вход С и свой вход сброса R. Считают оба счетчика, входящие в микросхему, по отрицательному фронту на своих входах C1 и С2. Сбрасываются они единичными сигналами на своих входах сброса R1 и R2.

Счетчики, входящие в ИЕ19, можно использовать самостоятельно, но можно и объединить их для получения 8-разрядного асинхронного счетчика с выходами 1, 2, 4, 8, 16, 32, 64, 128. Для такого объединения достаточно соединить выход 8 первого счетчика со счетным входом С2 второго счетчика. Если соединить два ИЕ19 (рис. 5.5), то получится уже 16-разрядный асинхронный двоичный счетчик. При этом выход 8 второго счетчика соединяется со счетным входом С1 первого счетчика. Однако и в данном случае каждый следующий разряд переключается с задержкой после переключения предыдущего.

Основное применение асинхронных счетчиков состоит в построении всевозможных делителей частоты, то есть устройств, выдающих выходной сигнал с частотой, в несколько раз меньшей, чем частота входно-

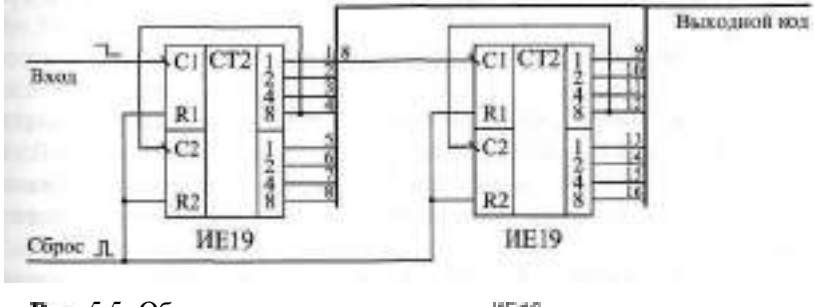


Рис. 5.5. Объединение двух счетчиков ИЕ19 для увеличения разрядности

го сигнала. В данном случае нас интересует не выходной код счетчика, то есть не все его разряды одновременно, а только один разряд, поэтому взаимные задержки отдельных разрядов не играют роли, полная задержка переключения счетчика не имеет значения. Простейший пример делителя частоты на два — это триггер в счетном режиме или счетчик, выходным сигналом которого является выход первого, младшего разряда 1.

При построении делителей частоты иногда важна не только частота выходного сигнала, но и его форма, его скважность, то есть отношение периода следования импульсов к длительности этих импульсов. В таких случаях чаще всего требуется *меандр* — цифровой сигнал со скважностью, равной двум (длительность импульсов равна длительности паузы между ними). Получить меандр из любого сигнала довольно просто: надо использовать дополнительный делитель частоты на 2, правда, при этом частота выходного сигнала уменьшится еще вдвое.

Простейший пример такого делителя частоты на десять приведен на рис. 5.6. В делителе использован счетчик ИЕ2, у которого одноразрядный внутренний счетчик включен после трехразрядного внутреннего счетчика. Трехразрядный счетчик делит частоту входного сигнала на 5, но выходные импульсы имеют скважность, не равную двум (она равна 5). Одноразрядный счетчик делит частоту еще вдвое и одновременно формирует меандр. Задержки переключения разрядов счетчика относительно друг друга на рисунке не показаны (применяем первый уровень представления, логическую модель).

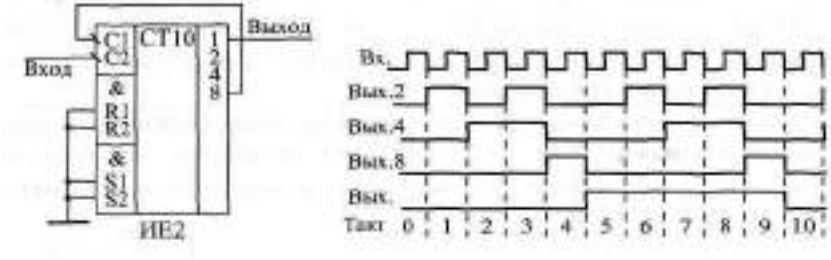


Рис. 5.6. Делитель частоты на 10, выдающий меандр

Иногда возникает задача деления частоты входного сигнала в произвольное число раз (не в 10 и не в *2",* что легко обеспечивается самой структурой стандартных счетчиков). В этом случае можно организовать сброс счетчика при достижении им требуемого кода путем введения обратных связей.

Например, на рис. 5.7 показан простейший делитель частоты на 9 на основе счетчика ИЕ5. При достижении его выходным кодом значения 9 (то есть 1001) счетчик автоматически сбрасывается в нуль по входам

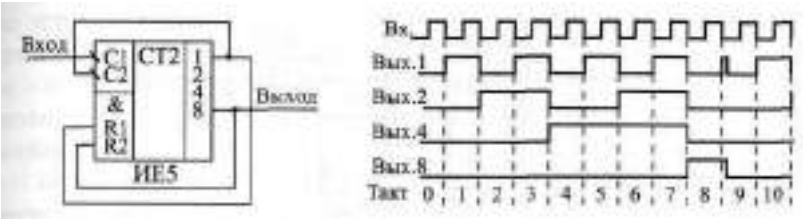


Рис. 5.7. Делитель частоты на 9 с обратными связями

R1 и R2, и счет начинается снова. В результате частота выходного сигнала в 9 раз меньше частоты входного сигнала. При этом скважность выходного сигнала не равна двум. Временная диаграмма показана на рисунке для первого уровня представления (без учета временных задержек).

Если в числе, на которое надо делить частоту, больше двух единиц (например, 15, то есть 1111, или 13, то есть 1101), то для формирования сигнала сброса надо использовать элементы 2И, 3И или 4И, чтобы объединить все выходы, равные единице. В результате можно построить делитель входной частоты в любое число раз от 2 до 2^N, где N — число разрядов используемого счетчика. Правда, при организации обратных связей надо учитывать ограничение на быстродействие счетчика. Все разряды, используемые для обратной связи, должны успеть переключиться за один период входного сигнала. Скважность выходного сигнала может принимать в данном случае самые разные значения, например, выходной сигнал может представлять собой очень короткие импульсы.

На асинхронных счетчиках можно строить также управляемые делители частоты, то есть такие делители, выходная частота которых определяется управляющим кодом. На рис. 5.8 показан делитель на 2^n, где n — целое. Восьмиразрядный счетчик ИЕ19 работает по входному сигналу с тактовой частотой f_T, а выходной 8-входовый мультиплексор КП7 передает на выход схемы один из 7 разрядов счетчика или же входной сигнал. Выбор номера канала производится входным управляющим 3-разрядным кодом. Например, при тактовой частоте f_T = 10 МГц, то есть при периоде входного сигнала 100 нс период выходного сигнала может составлять 100 нс, 200 нс, 400 нс, 800 нс, 1,6 мкс, 3,2 мкс, 6,4 мкс, 12,8 мкс.

В момент переключения управляющего кода на выходе схемы могут появиться нежелательные короткие импульсы, так как никакой синхронизации управляющего кода не предусмотрено. Поэтому схема должна работать так: сначала задается входной управляющий код, а уже потом разрешается работа той схемы, на которую поступает выходной сигнал, сформированный нашей схемой. В этом случае никаких проблем не будет. Не играют роли в данном случае и задержки переключения разрядов

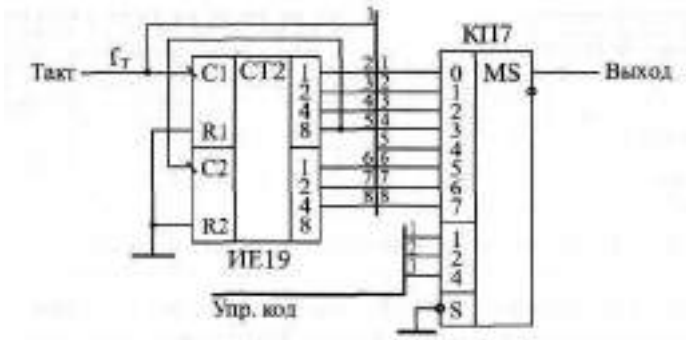


Рис. 5.8. Управляемый делитель частоты на асинхронном счетчике

счетчика, так как всегда используется только один его разряд. Главное, чтобы с частотой f_T переключался первый разряд счетчика.

Конечно, применение асинхронных счетчиков не ограничивается только делителями частоты. В случаях, когда высокого быстродействия не требуется, когда переходные процессы на выходах счетчика не имеют значения (при правильной синхронизации), асинхронные счетчики вполне могут заменить более быстрые синхронные счетчики. Доля таких задач составляет около 20 % от общего числа.

Если же включить на выходе асинхронного счетчика выходной параллельный регистр (рис. 5.9), то можно обеспечить одновременное переключение всех выходных разрядов счетчика.

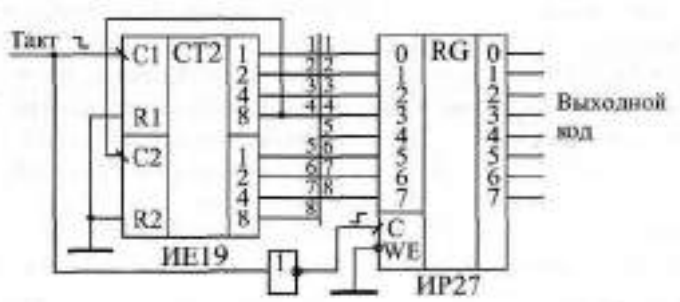


Рис. 5.9. Включение выходного регистра для одновременного переключения разрядов выходного кода

Данная схема будет работать правильно, если период следования входных тактовых импульсов будет больше, чем время установления всех разрядов счетчика (в нашем случае - 8-разрядного счетчика ИЕ19). Инвертор необходим, так как счетчик срабатывает по отрицательному фронту входного сигнала, а регистр — по положительному фронту. Данное реше-

НИС устраняет главный недостаток асинхронного счетчика — неодновременность установления его выходных разрядов. Однако второй недостаток — большая задержка установления выходного кода — сохраняется. Его устранить невозможно, можно только перейти на другие, более быстрые счетчики.

В заключение данного раздела надо отметить, что асинхронные счетчики, как и другие цифровые схемы, предъявляют требования к длительности входных сигналов. Например, не должны быть слишком короткими сигналы на тактовых входах и на входах сброса и установки. Не должны быть слишком затянутыми фронты входных сигналов. Тактовые сигналы и сигналы сброса не должны приходить со слишком малыми задержками друг относительно друга.

5.2. Синхронные счетчики с асинхронным переносом

Синхронные (или параллельные) счетчики характеризуются тем, что все их разряды в пределах одной микросхемы переключаются одновременно, параллельно. Это достигается существенным усложнением внутренней структуры микросхемы по сравнению с простыми асинхронными счетчиками. В результате полная задержка переключения синхронного счетчика примерно равна задержке одного триггера, то есть синхронные счетчики гораздо быстрее асинхронных, причем их быстродействие не падает с ростом количества разрядов выходного кода (конечно, до определенных пределов).

Управление работой синхронного счетчика гораздо сложнее, чем в случае асинхронного счетчика, а количество разрядов синхронных счетчиков обычно не превышает четырех. Поэтому синхронные счетчики не всегда могут успешно конкурировать с асинхронными, особенно при невысоких требованиях к быстродействию. Зато и возможностей у синхронных счетчиков, как правило, гораздо больше, чем у асинхронных, например, они обеспечивают параллельную запись информации в счетчик и инверсный режим счета.

Для объединения нескольких синхронных счетчиков с целью увеличения числа их разрядов (для каскадирования) используется специальный выходной сигнал переноса. В зависимости от принципов формирования этого сигнала и от принципов его использования синхронные (параллельные) счетчики делятся на счетчики с асинхронным (последовательным) переносом и счетчики с синхронным (параллельным) переносом (или полностью синхронные счетчики).

Синхронные счетчики с асинхронным переносом занимают промежуточное положение по быстродействию между асинхронными счетчиками и полностью синхронными счетчиками. Управление их работой

проще, чем у синхронных счетчиков, но сложнее, чем у асинхронных. Работают данные счетчики по положительному фронту входного сигнала (или, что то же самое, по заднему фронту отрицательного сигнала). Основная суть их работы сводится к следующему: все разряды одного счетчика переключаются одновременно, но при каскадировании каждый следующий счетчик (дающий более старшие разряды) переключается с задержкой относительно предыдущего счетчика (дающего более младшие разряды). То есть задержка переключения многоразрядного счетчика увеличивается в данном случае не с каждым новым разрядом (как у асинхронных счетчиков), а с каждой новой микросхемой (например, 4-разрядной).

Сигнал переноса у этих счетчиков при прямом счете вырабатывается тогда, когда все разряды равны единице (достигнут максимальный код) и когда приходит входной сигнал. Поэтому сигнал переноса, повторяющий входной сигнал, будет задержан относительно входного сигнала. И именно этот сигнал переноса используется в качестве входного для следующего счетчика при каскадировании. То есть входной сигнал второго счетчика задержан относительно входного сигнала первого счетчика, входной сигнал третьего счетчика задержан относительно входного сигнала второго счетчика и т. д.

Временная диаграмма 4-разрядного синхронного счетчика с асинхронным переносом показана на рис. 5.10. Из рисунка видно, что разряды переключаются одновременно по положительному фронту входного сигнала (с некоторой задержкой), а отрицательный сигнал переноса также задержан относительно входного отрицательного импульса. Понятно, что переключение разрядов счетчика, работающего с этим сигналом переноса в качестве входного, будет происходить с дополнительной задержкой относительно переключения разрядов данного счетчика.

Примерами синхронных счетчиков с асинхронным переносом могут служить двоично-десятичный счетчик ИЕ6 и двоичный счетчик ИЕ7 (рис. 5.11). Они полностью идентичны по своим возможностям и назначениям

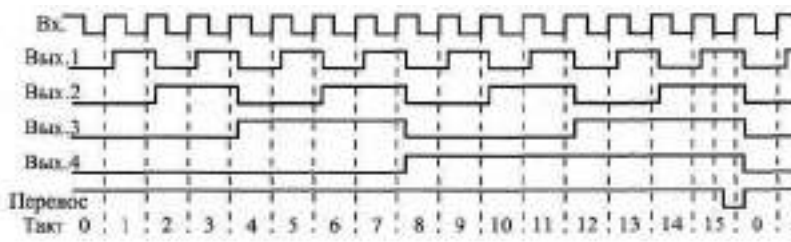


Рис. 5.10. Временная диаграмма работы синхронного счетчика с асинхронным переносом

входов и выходов, но только ИЕ6 считает от 0 до 9, а ИЕ7 — от 0 до 15. Оба счетчика реверсивные, обеспечивают как прямой счет (по положительному фронту на входе +1), так и обратный счет (по положительному фронту на входе -1). При прямом счете отрицательный сигнал переноса вырабатывается на выходе >15 (у ИЕ7) или >9 (у ИЕ6). При обратном (инверсном) счете отрицательный сигнал переноса вырабатывается на выходе < 0 после достижения выходным кодом значения 0000. Имеется возможность сброса счетчика в нуль положительным сигналом на входе R, а также возможность параллельной записи в счетчик кода со входов D1, D2, D4, 08 по отрицательному сигналу на входе -WR. При параллельной записи информации счетчики ведут себя как регистры-защелки, то есть выходной код счетчика повторяет входной код, пока на входе -WR присутствует сигнал нулевого уровня.

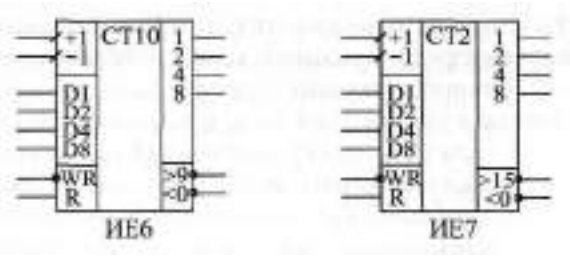


Рис. 5.11. Синхронные счетчики с асинхронным переносом

Таблица 5.4. Таблица режимов работы счетчиков ИЕ6 и ИЕ7

Входы				**Режим работы**
R	**-WR**	+1	-1	
1	**X**	**X**	**X**	Сброс в нуль
0	0	**X**	**X**	Параллельная запись
0	1	1	1	Хранение
0	1	0	0	Хранение
0	1	**0->1**	1	Прямой счет
0	1	1	0->1	Обратный счет

Вход параллельной записи обозначается иногда на схемах также L, C, а выходы переноса обозначаются также CR и BR.

Таблица режимов работы счетчиков ИЕ6 и ИЕ7 приведена в табл. 5.4.

После сброса счетчик начинает счет по положительным фронтам на счетных входах от нулевого кода. После параллельной записи счет на-

чинается от числа, записанного в счетчик. После переполнения счетчика ИЕ7 (достижения кода 1111) при прямом счете вырабатывается отрицательный сигнал переноса > 15, повторяющий входной отрицательный импульс на входе +1 с задержкой. После достижения кода 0000 при обратном счете вырабатывается отрицательный сигнал переноса < 0, повторяющий входной отрицательный импульс на входе -1 с задержкой. Точно так же работает и счетчик ИЕ6, но у него переполнение будет возникать в режиме прямого счета при достижении кода 1001.

Входные сигналы счета, записи и сброса не должны быть слишком короткими. Не должен быть слишком малым временной сдвиг между сигналами на входах D1-D8 и сигналом записи как в начале импульса записи, так и в его конце (сигнал записи -WR должен начинаться после установления входного кода, а заканчиваться — до снятия входного кода).

Объединение счетчиков ИЕ7 и ИЕ6 для увеличения разрядности (каскадирование) очень просто: нужно выходы переноса младших счетчиков (дающих младшие разряды выходного кода) соединить со счетными входами старших счетчиков (дающих старшие разряды выходного кода). На рис. 5.12 показана организация 12-разрядного счетчика на трех микросхемах ИЕ7. Этот счетчик может считать как на увеличение (прямой счет), так и на уменьшение (обратный счет). Возможны также сброс и параллельная запись в счетчики входного кода. Разряды каждого следующего счетчика будут переключаться одновременно, но с задержкой относительно переключения разрядов предыдущего счетчика. Точно так же объединяются и счетчики ИЕ6.

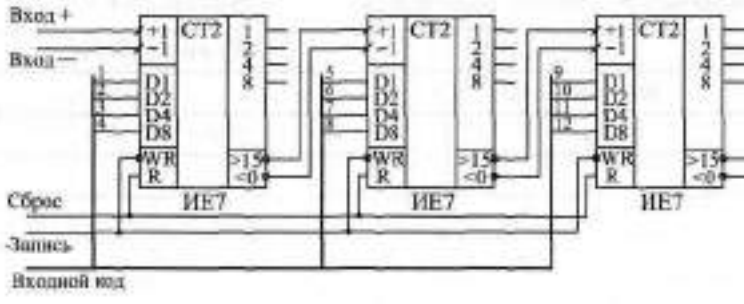


Рис. 5.12. Объединение счетчиков ИЕ7 для увеличения разрядности

Если нужно использовать все выходные разряды многоразрядного счетчика одновременно (как единый код), то необходимо выполнение следующего условия:

$$T > (N - 1)\, t_{зп} + t_{зс},$$

где T — период входного сигнала, N — число объединенных микросхем счетчиков, $t_{зп}$ — время задержки переноса одного счетчика, $t_{зс}$ — время задержки счета (переключения выходного кода) одного счетчика.

Применение синхронных счетчиков с асинхронным переносом очень многообразно, например, они могут делить частоту входного сигнала, считать входные импульсы, формировать пачки импульсов, измерять длительность временного интервала, формировать сигналы заданной длительности, измерять частоту входных импульсов, последовательно переключать входные и выходные каналы, формировать сложные последовательности сигналов, перебирать адреса памяти и многое другое. Мы рассмотрим лишь несколько наиболее типичных примеров.

В качестве делителя частоты входного сигнала синхронные счетчики с асинхронным переносом очень удобны, так как в них сочетается сравнительно высокая скорость работы с довольно простым управлением. Удобно также и то, что у них имеется режим обратного счета. На этих счетчиках можно строить делители частоты с произвольно изменяемым с помощью входного кода коэффициентом деления. Такие делители находят, например, широкое применение в аналого-цифровых системах, работающих с аналоговыми сигналами разной частоты.

Простейший пример 8-разрядного делителя частоты на счетчиках ИЕ7 показан на рис. 5.13.

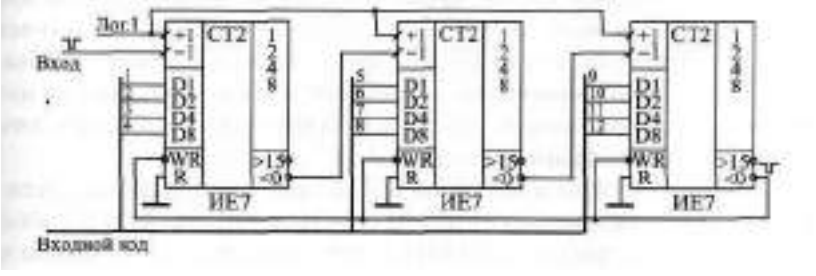


Рис. 5.13. Делитель частоты с коэффициентом деления, задаваемым входным кодом

На вход счетчиков подается 12-разрядный управляющий код, определяющий коэффициент деления входной частоты. Этот код записывается в счетчики по сигналу переноса < 0 старшего счетчика. С этого кода начинается затем счет на уменьшение. Когда счетчики отсчитают количество входных импульсов, равное входному коду, снова выработается сигнал переноса старшего счетчика и снова запишет входной код в счетчики. Коэффициент деления будет равен (N+1) при входном коде N. Отрицательный выходной сигнал будет по форме повторять входной, но с полной задержкой переноса, а его частота будет меньше частоты входного сигнала

в (N+1) раз. При 12-разрядном входном коде максимальный коэффициент деления составит 4096, а минимальный - 1.

Чтобы сформулировать условия правильной работы данного делителя частоты, надо прежде всего отметить, что запись входного кода в счетчики производится отрицательным уровнем сигнала -WR, то есть передним фронтом входного отрицательного импульса, а счет производится положительным фронтом сигнала 1, то есть задним фронтом входного отрицательного импульса. Отсюда следует, что входной импульс должен быть достаточно коротким. Если он записывает код в счетчики своим передним фронтом, он уже не должен своим задним фронтом переключать счетчики по входу -1. Поэтому длительность входного отрицательного импульса не должна превышать полного времени переключения счетчиков и записи в них входного кода. В нашем случае это три задержки переноса и задержка записи в счетчик.

Если частота входного сигнала — большая (к примеру, больше 10 МГц), то нормальная длительность входного сигнала получается сама собой. Но частота входного сигнала не должна быть и слишком большой, иначе в процессе записи счетчик пропустит один из входных импульсов или даже несколько. То есть от переднего фронта входного отрицательного сигнала до заднего фронта следующего входного отрицательного сигнала должны успеть сработать все счетчики и должна произойти запись в счетчики (суммарное время задержки опять же включит в себя сумму задержек переноса всех счетчиков и задержку записи). Ограничения на входную частоту будет тем жестче, чем больше счетчиков мы объединяем для увеличения количества разрядов. В данном случае важно именно количество примененных микросхем, а не количество используемых разрядов, как у асинхронных счетчиков.

Для решения часто встречающейся на практике задачи подсчета количества пришедших входных импульсов необходимо всего лишь объединить несколько микросхем счетчиков с целью получения требуемого числа разрядов. Например, если количество входных импульсов не превышает 255, то достаточно двух 4-разрядных счетчиков, если оно не больше 65535, то надо объединить уже четыре 4-разрядных счетчика. Так как в этом случае нас интересуют все выходные разряды одновременно, необходимо обеспечить, чтобы за период входных импульсов переключались все микросхемы счетчиков.

Обеспечить одновременность переключения всех выходных разрядов счетчика при счете входных импульсов можно, как и в случае асинхронных счетчиков, за счет включения выходного параллельного регистра, срабатывающего по фронту (рис. 5.14). Данное решение довольно универсально, оно может использоваться в самых разных ситуациях, когда необходим весь выходной код счетчика целиком. Код на выходе реги-

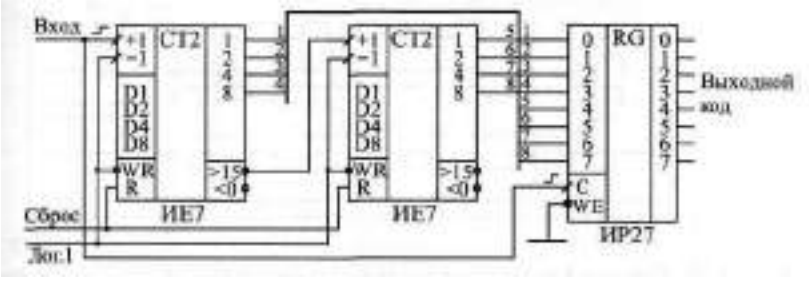


Рис. **5.14.** Включение выходного регистра для одновременного переключения разрядов выходного кода

стра будет удерживаться в течение всего периода входных импульсов. Правда, быстродействие счетчика от этого не повышается.

Формирование пачки (группы) входных импульсов с заданным количеством импульсов — довольно распространенная задача. Например, такое формирование необходимо при организации обмена информацией в последовательном коде. Если в качестве преобразователя параллельного кода в последовательный используется 8-разрядный регистр сдвига, то ему в качестве синхросигнала необходима пачка из восьми импульсов. Схема формирователя такой пачки импульсов показана на рис. 5.15, а временная диаграмма ее работы — на рис. 5.16.

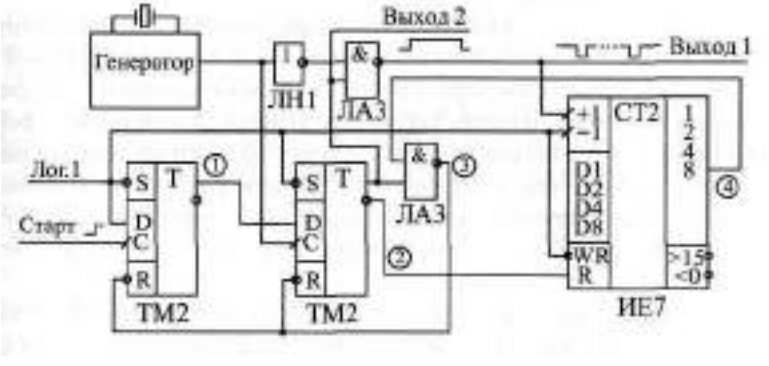


Рис. 5.15. Формирователь пачки из восьми импульсов

По сигналу "Старт" (положительный фронт) перебрасывается первый триггер, использующийся для синхронизации. По первому положительному фронту тактового сигнала с генератора перебрасывается второй триггер, разрешающий прохождение импульсов с генератора на выход через элемент 2И-НЕ, а также разрешающий работу счетчика ИЕ7.

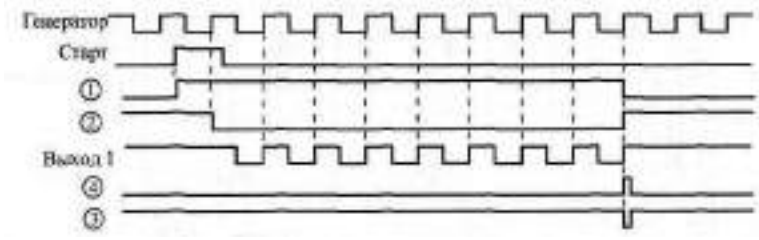


Рис. 5.16. Временная диаграмма работы формирователя пачки импульсов

После того как на Выход 1 схемы пройдут восемь отрицательных импульсов, на выходе 8 счетчика выработается единица, что приведет к сбросу в исходное нулевое состояние обоих триггеров (коротким отрицательным импульсом на выходе нижнего по рисунку элемента 2И-НЕ) и к запрету прохождения импульсов на выход. Работа формирователя возобновится после следующего сигнала "Старт".

На основе счетчиков довольно просто строить формирователи временных интервалов с длительностью, задаваемой внешним кодом. Такие формирователи находят широкое применение, например, в различных измерительных устройствах. Так как формирователь временных интервалов обычно работает с кварцевым тактовым генератором, возможны два подхода к его построению.

При первом подходе входной стартовый импульс синхронизируется с тактовым сигналом, в результате чего выходной импульс заданной длительности может начаться не сразу после стартового импульса, а через какое-то время, меньшее периода тактового сигнала. Длительность формируемого временного интервала в этом случае абсолютно точно известна и будет равна целому числу периодов тактового генератора. Именно так было сделано в предыдущей рассмотренной нами схеме (сигнал "Выход 2" на рис. 5.15 как раз и будет формируемым сигналом с заданной длительностью).

При втором подходе выходной импульс заданной длительности начинается сразу после входного сигнала, но длительность его может отличаться от заданной на какое-то время, меньшее периода тактового сигнала. Иногда это более приемлемое решение, особенно при больших длительностях выходного сигнала, значительно больших, чем период тактового сигнала. Схема формирователя временного интервала, построенного в соответствии с этим вторым подходом, показана на рис. 5.17.

Работа схемы начинается с подачи короткого отрицательного импульса —Старт. Он перебрасывает триггер, который разрешает работу счетчиков снятием сигнала параллельной записи –WR. По отрицатель-

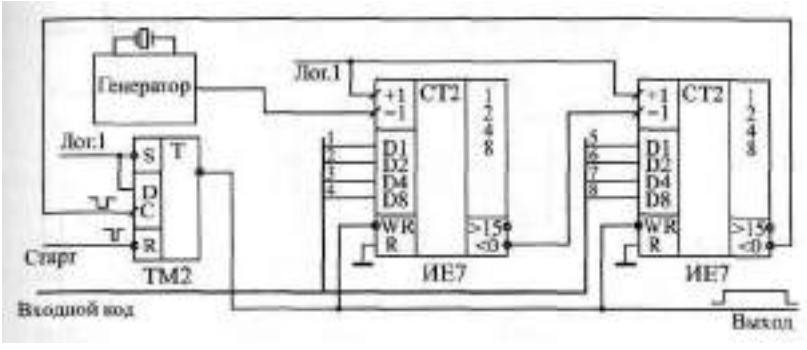


Рис. 5.17. Формирователь временного интервала

ному фронту входного сигнала начинается положительный выходной сигнал заданной длительности. Счетчики начинают считать на уменьшение кода по положительным фронтам тактового сигнала с генератора. Когда они досчитают до нуля, вырабатывается сигнал переноса, перебрасывающий триггер в исходное состояние. Работа схемы возобновится после следующего сигнала -Старт.

Если входной код равен 1, то длительность выходного сигнала составит от T до *21,* где T — период тактового сигнала. Если входной код равен N (до 255), то длительность выходного сигнала составит от NT до (N+1) T в зависимости от момента прихода входного сигнала по отношению к тактовому сигналу. Абсолютная погрешность выдержки длительности выходного сигнала в любом случае не превышает периода тактового сигнала T.

Эту же самую схему вполне можно использовать в тех случаях, когда необходимо получить убывающий код от заданного числа до нуля. При этом сигнал с выхода триггера будет только внутренним сигналом схемы, а выходными сигналами схемы будут выходные разряды счетчиков.

Иногда бывает необходимо сформировать импульс требуемой длительности, но одновременно иметь не убывающий, а возрастающий код (от нуля до заданного значения). В таком случае схема получится несколько сложнее. Пример возможного решения формирователя импульса заданной длительности показан на рис. 5.18.

По сигналу "Старт" (положительный фронт) перебрасывается левый по рисунку триггер, который начинает формировать выходной сигнал и разрешает работу счетчика (снимая сигнал сброса R). Счетчик считает на увеличение по положительным фронтам тактового сигнала от нуля. Когда выходной код счетчика достигает величины входного кода, перебрасывается правый по рисунку триггер, завершающий процесс формирования выходного сигнала. Счетчик сбрасывается в нуль, правый триг-

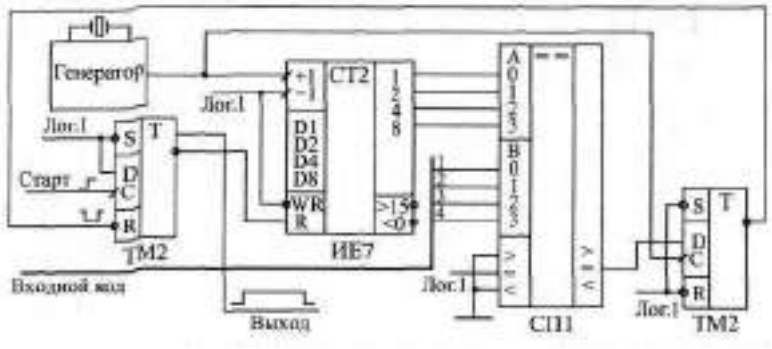


Рис. 5.18. Формирователь импульса заданной длительности (вариант с нарастающим кодом)

гер по следующему фронту попадает в исходное состояние. Новый цикл начнется с приходом следующего сигнала "Старт".

Если входной код равен 1, то длительность выходного сигнала составит от T до 2T, где T — период тактового сигнала генератора. Если входной код равен N, то длительность выходного сигнала будет равна от NT до (N + 1) T в зависимости от временного сдвига между сигналом "Старт" и тактовым сигналом. В любом случае абсолютная погрешность времени выдержки выходного сигнала не превысит периода тактового сигнала T.

Счетчики также широко применяются в различных измерителях длительности входных сигналов. Для этого они отсчитывают импульсы тактового кварцевого генератора в течение длительности входного сигнала. После окончания входного сигнала в счетчике остается код, пропорциональный длительности этого сигнала. Пример практической схемы такого измерителя показан на рис. 5.19.

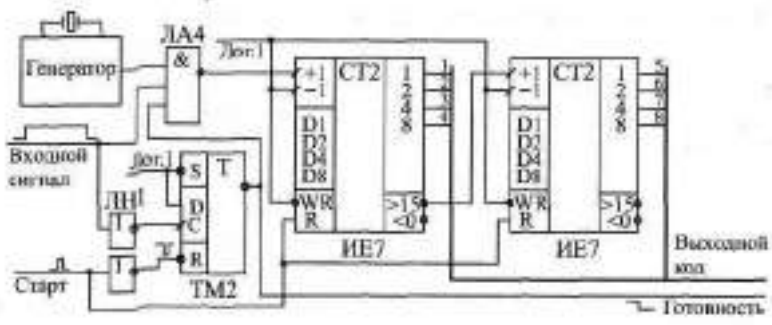


Рис. 5.19. Измеритель длительности входного сигнала

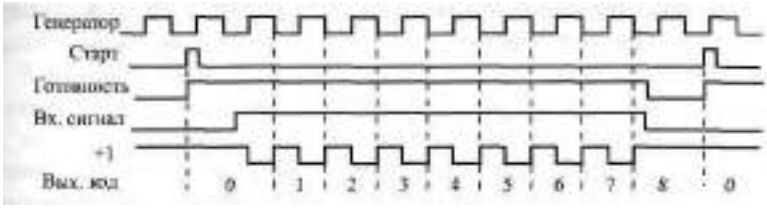


Рис. 5.20. Временная диаграмма работы измерителя длительности входного сигнала

Работа схемы начинается по короткому управляющему импульсу "Старт", который сбрасывает счетчик в нуль и переводит всю схему в режим счета, разрешая прохождение сигнала с тактового генератора на вход +1 счетчика при положительном входном сигнале. С началом входного сигнала импульсы с генератора поступают на вход счетчика, и счетчик их считает. После окончания входного сигнала поступление импульсов на вход счетчика прекращается, триггер перебрасывается в исходное состояние и сообщает отрицательным фронтом на своем инверсном выходе о готовности выходного кода (сигнал "Готовность"). Работа схемы возобновится по следующему импульсу "Старт". Временная диаграмма работы измерителя длительности входного сигнала приведена на рис. 5.20.

Выходной код N измерителя связан с длительностью входного сигнала t простым соотношением

$$t = NT,$$

где T — период тактового сигнала. Абсолютная погрешность измерения не превышает величины $\pm T$. Поэтому для уменьшения относительной погрешности измерения необходимо увеличивать частоту тактового генератора и увеличивать разрядность счетчика.

Счетчики также применяются и для измерения частоты входного цифрового сигнала.

Частоту входного сигнала можно измерить двумя путями: косвенным, то есть измерением периода входного сигнала (по принципу, рассмотренному только что) и вычислением затем частоты (по формуле $f_{Вх} = 1/T_{Вх}$), или же прямым измерением частоты. Первый метод требует вычислений с помощью компьютера или микроконтроллера, второй не требует никаких дополнительных вычислений. Поэтому мы рассмотрим здесь реализацию метода прямого измерения частоты.

В соответствии с этим методом необходимо сформировать временное окно с заданной длительностью t_{0}, в течение которого надо сосчи-

тать количество N периодов входного сигнала T (рис. 5.21). В этом случае будет выполняться соотношение

$$t_0 = NT \text{ или } f = N/t_0,$$

где f — это частота входного сигнала, равная $1/T$. То есть частота входного сигнала пропорциональна коду N, а коэффициент пропорциональности равен $1/t_0$. Если, например, выбрать $t_0 = 1$ с, то код N будет равен частоте входного сигнала в герцах, а при $t_0 = 1$ мс код N будет равен частоте входного сигнала в килогерцах.

Если длительность временного окна — строго постоянная величина, то погрешность измерения частоты будет определяться только погрешностью подсчета кода N. Абсолютная погрешность подсчета кода N не превысит единицы, а относительная погрешность не будет более $1/N$. Понятно, что для увеличения точности измерения частоты нужно увеличивать N, то есть необходимо увеличивать длительность временного окна t_0. Однако при этом автоматически увеличивается время измерения.

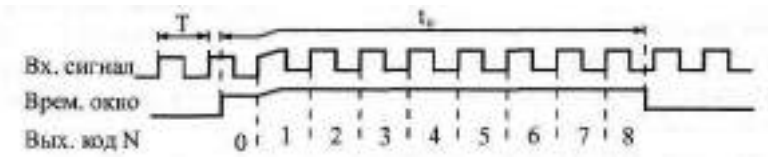


Рис. 5.21. Измерение частоты входного сигнала прямым методом

Схема измерителя частоты (рис. 5.22) практически не отличается от схемы измерителя длительности входного сигнала (рис. 5.19). Только в данном случае в качестве измеряемого сигнала будет использоваться сигнал временного окна, а в качестве тактового сигнала — входной сигнал. Для формирования сигнала временного окна можно применить схе-

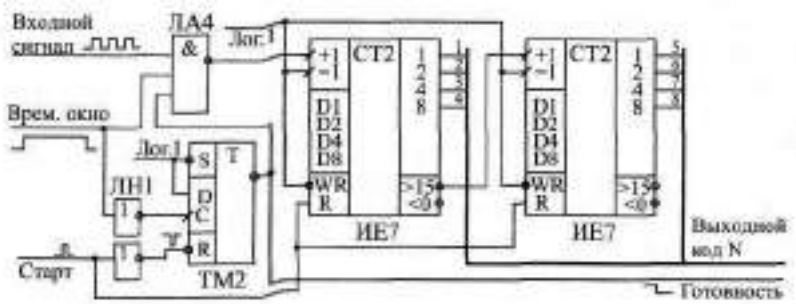


Рис. 5.22. Измеритель частоты входного сигнала прямым методом

му рис. 5.15 (сигнал "Выход 2"), которая обеспечивает постоянную длительность выходного сигнала.

Еще одно широко распространенное применение счетчиков — последовательное переключение (сканирование) нескольких устройств, узлов, индикаторов, каналов передачи и т. д. То есть имеется, например, группа устройств, которые должны по тем или иным причинам работать не одновременно, а по очереди, так, что в каждый момент активным является только одно устройство, причем очередь эта замкнута в кольцо и после последнего устройства начинает работать первое. Или же имеется несколько каналов связи (входных или выходных линий), которые надо также по очереди подключать к одному выходу (при выходных каналах) или к одному входу (при входных каналах).

Во всех подобных случаях опрос, переключение, сканирование может производить счетчик с нужным числом разрядов. Счетчик с числом разрядов п может обслуживать 2" устройств (или каналов).

В качестве первого примера рассмотрим схему переключения выходных каналов (рис. 5.23). Она последовательно, по очереди, циклически коммутирует один входной сигнал на восемь выходов, для чего используется счетчик, тактируемый сигналом задающего генератора, и дешифратор, работающий в качестве демультиплексора. Каждый из выходных каналов активен (то есть подключен) в течение одного периода тактового сигнала, а затем пассивен (то есть отключен) в течение семи периодов тактового сигнала. Предусмотрена возможность начального сброса схемы с помощью сигнала "Сброс".

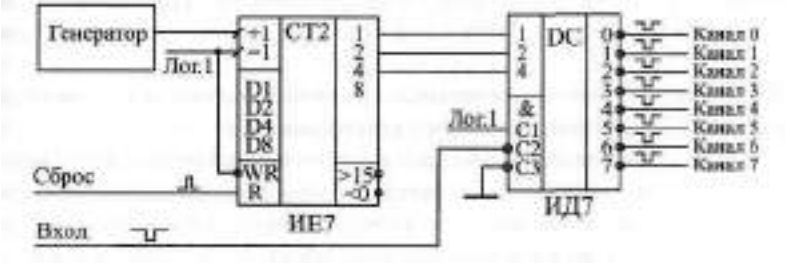


Рис. 5.23. Схема последовательного переключения выходных каналов

Используя данную схему, надо учитывать, что в момент переключения каналов может искажаться (обрезаться) выходной сигнал. Поэтому лучше всего обеспечить, чтобы входной сигнал приходил только тогда, когда переключение каналов не производится. Или на время передачи вообще останавливать процесс перебора каналов путем запрета прохождения импульсов с генератора на вход счетчика, а после окончания передачи снова разрешать последовательный перебор каналов.

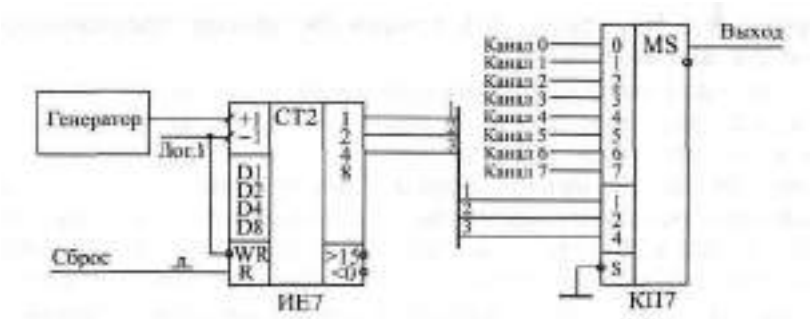


Рис. **5.24.** Схема последовательного переключения входных каналов

Второй пример, который мы рассмотрим, это схема, решающая обратную задачу — переключение входных каналов (рис. 5.24). Данная схема последовательно, циклически передает один из восьми входных сигналов на выход. Как и в предыдущем случае, перебор каналов осуществляется счетчиком, тактируемым сигналом с генератора. Непосредственно коммутация сигналов производится мультиплексором, на адресные входы которого подаются три разряда счетчика. Предусмотрена возможность начального сброса схемы с помощью сигнала "Сброс".

В момент переключения каналов здесь также возможно искажение (обрезание) коммутируемых сигналов. Поэтому желательно обеспечить передачу сигналов в момент, когда переключения каналов нет. Или же надо останавливать процесс перебора каналов на время приема сигнала из выбранного канала путем запрета прохождения тактовых импульсов на вход счетчика, а затем снова запускать перебор каналов.

Еще одно применение счетчиков из этой же области состоит в организации так называемой динамической индикации.

Суть динамической индикации состоит в следующем. Если используется табло из нескольких индикаторов (одиночных светодиодов, светодиодных семисегментных индикаторов, светодиодных матричных индикаторов и т. д.), то совсем не обязательно все эти индикаторы должны гореть постоянно, одновременно. Можно зажигать их по очереди, что существенно сократит потребляемый всей схемой ток питания. Например, если в каждый момент времени горит только один индикатор из имеющихся восьми, то ток потребления индикаторов сократится в восемь раз. Учитывая, что каждый светящийся светодиод требует тока порядка 1–5 мА, такой подход может дать большой выигрыш, особенно при матричных индикаторах, содержащих несколько десятков светодиодов. А инерционность человеческого глаза приводит к тому, что вспышки света с частотой больше 20 Гц воспринимаются как непрерывное свечение.

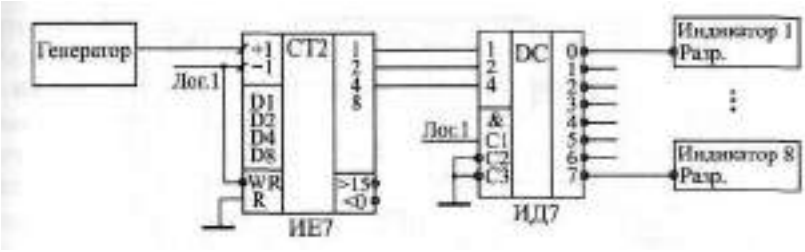


Рис. 5.25. Схема динамической индикации на восьми индикаторах

Так что при достаточной частоте перебора индикаторов глазу не будет заметно последовательное их включение.

На рис. 5.25 приведен пример схемы динамической индикации на восьми индикаторах. Для последовательного перебора индикаторов применяется счетчик, соединенный с дешифратором. Выходные сигналы дешифратора используются в качестве сигналов разрешения свечения для индикаторов. Частота сигнала тактового генератора, с которым работает счетчик, должна составлять не менее 160 Гц, чтобы каждый индикатор загорался не реже, чем с частотой 20 Гц. При этом нельзя также выбирать слишком большую частоту тактового генератора, так как в моменты переключения ток потребления микросхем сильно возрастает из-за паразитных емкостей, и при большой частоте весь эффект снижения потребления может сойти на нет.

Счетчики часто используют также для организации всевозможных таймеров, часов, то есть схем счета времени, выходной код которых необходимо время от времени читать. Для этого на вход счетчика подается сигнал образцовой частоты с кварцевого генератора. Здесь возникает следующая проблема. Если чтение происходит в тот момент, когда счетчики переключаются, то с выходов счетчиков может быть считан случайный код, который не соответствует ни предыдущему установившемуся значению, ни последующему установившемуся значению. Можно, конечно, на время чтения кода остановить счет, но тогда ход часов собьется.

Пример решения данной проблемы приведен на рис. 5.26. Здесь выходной код счетчика на каждом такте записывается в выходной регистр с разрешением записи ИР27. А в момент чтения кода (при положительном сигнале "Чтение") запись в регистр запрещается. В результате в течение всей длительности сигнала "Чтение" выходной код схемы будет неизменным, хотя счетчик будет продолжать считать без всяких помех, и ход часов не собьется.

Интересная особенность счетчиков ИЕ6 и ИЕ7 состоит в том, что они могут работать не только в режиме счета, но и в режиме повторителя входных сигналов данных. В режиме параллельной записи в счетчик при

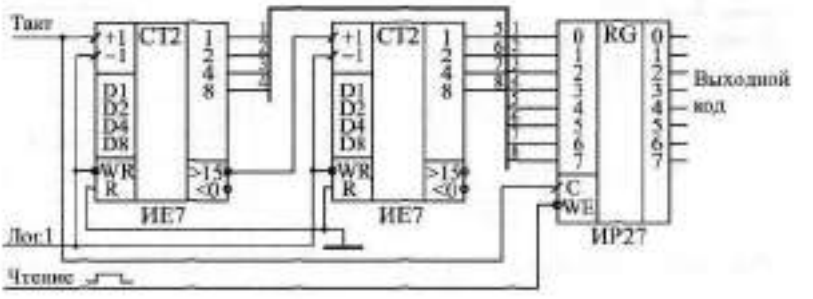


Рис. 5.26. Схема таймера с чтением выходного кода

нулевом сигнале на входе -WR выходные сигналы счетчика будут повторять любые изменения входных сигналов данных, то есть счетчик работает по сути как регистр, срабатывающий по уровню стробирующего сигнала. В ряде случаев такая особенность очень удобна, так как она позволяет существенно упростить аппаратуру.

Пусть, например, необходимо выдавать на вход схемы один из двух входных кодов: код со счетчика или код с регистра (то есть требуется мультиплексирование двух кодов). Эту задачу можно решить, применяя двухканальный мультиплексор (рис. 5.27*а*), а можно сделать проще — подавать код с регистра на входы данных счетчика и переводить в нужный момент счетчик в режим параллельной записи (рис. 5.27*б*). В обоих случаях переключение кодов, подаваемых на выход схемы, производится сигналом Упр. Правда, во втором случае счетчик возобновляет свой счет (после снятия сигнала записи -WR) с кода, записанного в регистр.

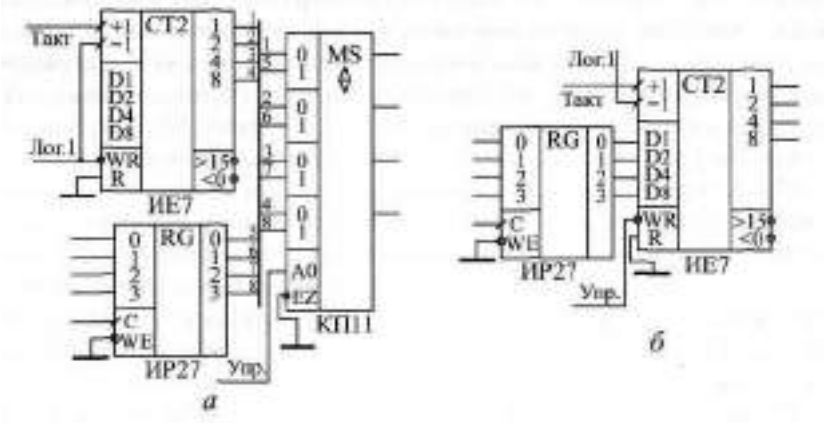


Рис. 5.27. Варианты мультиплексирования выходного кода счетчика с применением мультиплексора *(а)* и без него *(б)*

Если это неприемлемо, то можно воспользоваться входом сброса счетчика в нуль R.

И в заключение данного раздела мы рассмотрим две более сложные схемы, строящиеся на основе счетчиков. Это генератор прямоугольных импульсов с изменяемой частотой и длительностью импульса и быстродействующий высокоточный измеритель частоты входного сигнала с большим диапазоном измеряемых частот.

Генерация прямоугольных импульсов — довольно часто встречающаяся задача, в частности при разработке, отладке, тестировании электронной аппаратуры. От генератора прямоугольных импульсов требуется выдача импульсов заданной длительности при заданной паузе между импульсами (или, что то же самое, формирование импульсов заданной длительности и частоты следования). Желательно, чтобы диапазон изменения длительности импульсов и пауз между ними был как можно шире. Желательно также, чтобы был предусмотрен режим разового запуска (то есть остановка генерации после окончания одного выходного импульса) и автоматического запуска (то есть генерация периодической последовательности импульсов до прихода внешней команды остановки).

Предлагаемая здесь схема генератора не претендует, конечно, на рекордные характеристики, но она вполне может стать реальным удобным инструментом для разработчика цифровой аппаратуры, особенно если управление генератором поручить компьютеру с установленной на нем развитой сервисной управляющей программой. Благодаря своей простоте и наглядности, схема эта может служить образцом для разработки более сложных генераторов импульсов, например, имеющих более высокое быстродействие, больший диапазон изменения длительности импульсов и их частоты, обеспечивающих генерацию импульсов с разной амплитудой и полярностью.

В основе генератора (рис. 5.28) — два 16-разрядных счетчика, выполненных на основе микросхем К1Е7. Один из этих счетчиков (на рисунке внизу) отсчитывает длительность выходного импульса, другой (на рисунке вверху) — длительность паузы. Коды длительности импульса и паузы подаются, соответственно, на входы данных верхнего и нижнего счетчиков (эти коды могут храниться, например, в регистрах, не показанных на схеме). Счетчики импульса и паузы работают по очереди, что определяется управляющими сигналами на их входах параллельной записи -WR, которые также запрещают прохождение на входы 1 тактовых импульсов с помощью элементов 2И-НЕ. Эти управляющие сигналы поступают с прямого и инверсного выходов триггера TM2, на входы -R и -S которого подаются сигналы переноса с выходов < 0 обоих счетчиков.

В результате, когда один счетчик считает, другой находится в режиме параллельной записи и не считает. После того как считающий счетчик

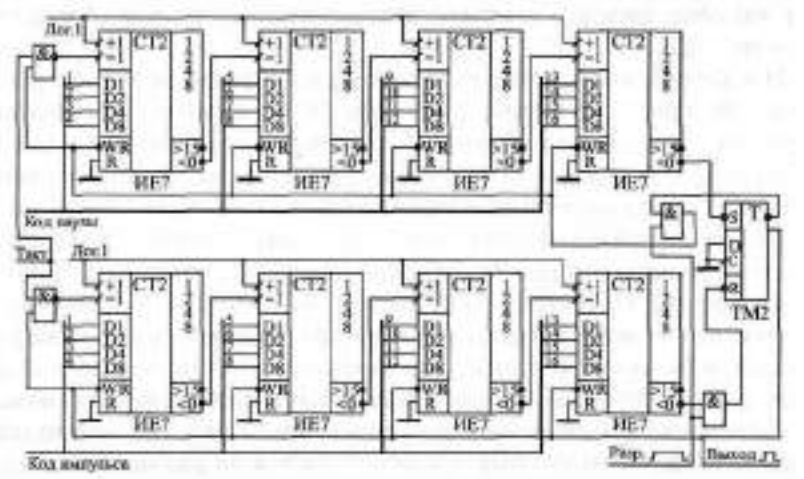


Рис. 5.28. Счетчики длительности импульса и паузы для генератора прямоугольных импульсов

досчитает до нуля, он перебросит выходной триггер, который переведет этот счетчик в состояние параллельной записи, запретит поступление на его вход тактовых импульсов и разрешит считать другому счетчику. Описанная последовательность действий повторится уже для другого счетчика. И этот процесс будет повторяться до тех пор, пока разрешена генерация.

В данном случае смело можно одновременно использовать как вход -R, так и вход -S триггера, поскольку сигналы, приходящие на них, гарантированно разнесены во времени. Сигнал с прямого выхода триггера служит выходным сигналом всего генератора в целом. Разрешается генерация положительным сигналом "Разр". Когда генерация запрещена (нулевой сигнал "Разр."), триггер сброшен в нуль по входу -R и оба счетчика находятся в состоянии параллельной записи. Поэтому генератор всегда начинает работу с отработки паузы заданной длительности, а потом отрабатывает выходной импульс заданной длительности.

Сформулируем условия правильной работы данной схемы.

Во-первых, как и в случае управляемого делителя частоты (см. рис. 5.13), перевод счетчиков из режима счета в режим параллельной записи осуществляется передним (отрицательным) фронтом тактового отрицательного импульса, а счет производится задним (положительным) фронтом отрицательного тактового импульса. Поэтому отрицательный тактовый импульс должен быть достаточно коротким. Один и тот же тактовый импульс не должен своим передним фронтом менять режим счетчиков,

а задним фронтом переключать счетчики по входу -1. Длительность тактового отрицательного импульса не должна превышать полного времени переключения режимов счетчиков, включающего в себя четыре задержки переноса счетчиков, задержку переключения выходного триггера и задержку элементов 2И и 2И-НЕ.

Во-вторых, частота тактового сигнала не должна быть слишком большой, чтобы за время переключения режимов на вход -1 не пришел еще один положительный фронт тактового сигнала. Иначе этот фронт будет потерян. То есть от момента отрицательного фронта тактового импульса до момента положительного фронта следующего тактового импульса схема должна успеть полностью закончить переключение режимов счетчиков.

Пусть, например, мы хотим выбрать максимальную тактовую частоту 10 МГц (период Ту = 100 нс). Посмотрим, можно ли использовать микросхемы счетчиков серии КР1533. Для счетчиков КР1533ИЕ7 задержка сигнала переноса составляет не более 18 нс. Для четырех микросхем задержка переноса составит 72 нс. Тогда на сумму задержек триггера, элемента 2И и элемента 2И-НЕ остается не более 28 нс. Следовательно, если мы возьмем эти элементы из более быстрых серий (например, КР531 или КР1531), мы легко удовлетворим этому требованию.

При величине кода импульса N длительность импульса $T_И$ составит (N+1) • Ту. При величине кода паузы М длительность паузы $T_П$ составит (М+1) • Ту. Период выходных импульсов $T_{ВЫХ}$ будет равен (M+N+2) • Ту. Коды М и N могут принимать значения от 0 до 65535. То есть минимальная длительность импульса и паузы равна Ту, максимальная длительность импульса и паузы равна 65536 Ту, минимальная длительность периода выходного сигнала равна 2Ту, а максимальная — 131072 Ту. Например, при тактовой частоте 10 МГц максимальный период выходного сигнала будет равен 13,1072 мс, а минимальный — 200 нс.

Для расширения диапазона изменения периода выходного сигнала можно применить управляемый делитель тактовой частоты. Другой возможный путь — наращивание разрядности счетчиков — приводит к снижению максимально допустимой тактовой частоты, так как обязательно вызывает увеличение задержек переключения счетчиков. К тому же, как правило, нет необходимости задавать длительность периода выходного сигнала, скажем, в 1 секунду с абсолютной погрешностью 100 нс (относительная погрешность — 10^{-7}). Гораздо важнее обеспечить стабильность частоты и периода выходного сигнала. Поэтому применение управляемого делителя частоты тактового сигнала не ухудшает характеристик генератора. Схема управления генератором прямоугольным импульсом с делителем частоты показана на рис. 5.29.

Делитель частоты работает с кварцевым генератором с частотой 10 МГц и включает в себя три делителя на 16 на счетчиках ИЕ7. На выход

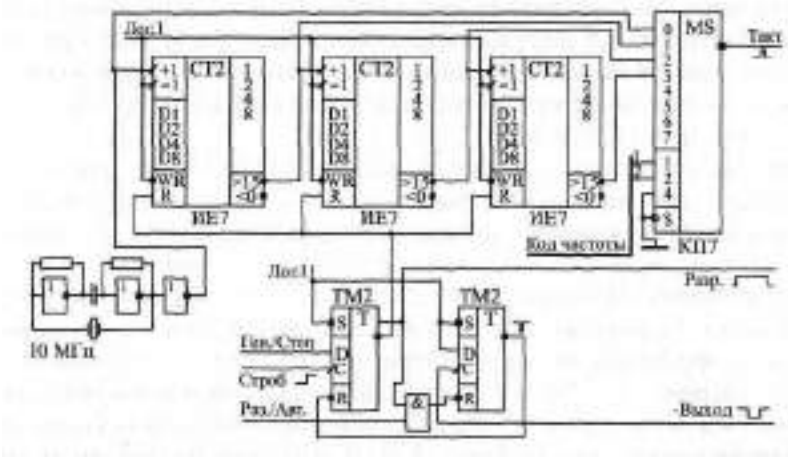


Рис. 5.29. Схема управления и делитель частоты для генератора прямоугольных импульсов

мультиплексора (сигнал "Такт") проходит один из сигналов с периодом 100 нс, 1,6 мкс, 25,6 мкс, 409,6 мкс. Длительность сигнала "Такт" не превышает половины периода сигнала с частотой 10 МГц, то есть 50 нс, что обеспечивает правильную работу счетчиков импульса и паузы (см. рис. 5.28). Выбор тактовой частоты осуществляется 2-разрядным кодом частоты. При запрете генерации все счетчики сбрасываются в нуль, это увеличивает точность привязки момента начала генерации к моменту подачи команды на начало генерации.

Схема управления генератором прямоугольных импульсов, также показанная на рис. 5.29, включает в себя два триггера ТМ2 и логический элемент 2И (ЛИ1).

Левый по рисунку триггер вырабатывает сигнал разрешения генерации "Разр". В этот триггер необходимо записать единицу для разрешения генерации или нуль для остановки генерации. Запись в триггер входного сигнала Ген./Стоп производится передним фронтом сигнала "Строб".

Правый по рисунку триггер служит для организации разового запуска генератора. Переключение режима разового или автоматического запуска производится управляющим сигналом "Раз./Авт". При автоматическом запуске (нуль на входе Раз./Авт.) данный триггер не работает, он всегда находится в нулевом состоянии и дает уровень логической единицы на своем инверсном выходе. При разовом запуске (единица на входе Раз./Авт.) правый триггер переходит в рабочий режим сразу после начала генерации (положительный сигнал "Разр."). После окончания генерации первого выходного импульса на инверсном выходе генератора (ин-

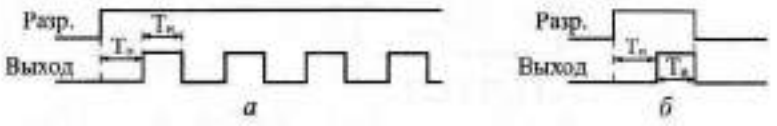


Рис. 5.30. Режимы работы генератора импульсов: автоматический *(а)* и разовый *(б)*

версный выход триггера на рис. 5.28) появляется положительный перепад, который перебрасывает правый триггер на рис. 5.29. В результате он своим выходным сигналом сбрасывает левый триггер, что приводит к остановке генерации (так как сигнал "Разр." становится нулевым). После этого схема снова готова к разовому запуску генерации. Временные диаграммы работы схемы в режимах автоматического и разового запуска показаны на рис. 5.30.

Асинхронность (независимость) момента прихода команды на начало передачи и сигнала задающего кварцевого генератора приводит к тому, что длительность первой паузы может оказаться на 100 нс меньше, чем она задана кодом паузы. Но это не слишком существенно, так как гораздо важнее длительность выходного импульса. Все последующие импульсы и паузы выдерживаются точно.

Абсолютная погрешность установки длительностей импульса и паузы $T_И$ и $T_П$ составляет половину периода тактового сигнала Ту. Относительная погрешность установки этих величин составляет, соответственно, 0,5/N и 0,5/M. Понятно, что при малых величинах N и M погрешность будет большой (в пределе — даже 50 %). Но при больших величинах длительностей импульса и паузы относительная погрешность не превышает 0,5/4096, то есть 0,012%.

Таким образом, рассмотренный генератор может формировать импульсы длительностью от 100 нс с паузой между импульсами от 100 нс. Максимально возможная длительность импульса составляет 2^{16} (2^{12} (100 нс = 26, 84 с. Такой же может быть и пауза. Правда, отношение длительности импульса к длительности паузы (или длительности паузы к длительности импульса) не может превышать 65536. Величина периода выходного сигнала генератора может достигать 53,69 с.

Теперь рассмотрим вторую схему.

Задача измерения частоты следования входных прямоугольных импульсов также часто встречается как в чисто цифровых, так и в аналого-цифровых системах. Как уже упоминалось, существует два традиционных метода измерения частоты (рис. 5.31): один предполагает измерение периода $T_{ВХ}$ путем подсчета тактовых импульсов с периодом $T_Т$ в течение $T_{ВХ}$ и дальнейшее вычисление частоты по формуле $f_{ВХ} = 1/T_{ВХ}$ *(а),* а дру-

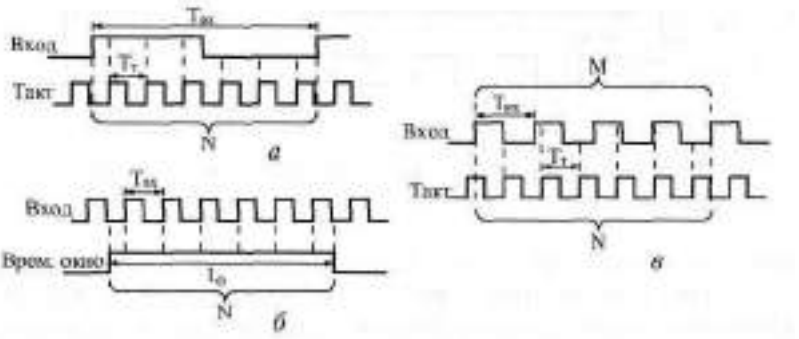


Рис. **5.31.** Методы измерения частоты: через период *(а),* прямой *(б)* и комбинированный *(в)*

гой прямо измеряет частоту f_{BX} путем подсчета входных импульсов в течение временного окна t_0 *(б)*.

Относительная погрешность и того, и другого метода не превышает величины 1/N, где N — полученный в результате подсчета код. Понятно, что первый метод дает хорошую точность только для низких частот f_{BX} (то есть для больших T_{BX} и соответственно больших N). Второй метод дает хорошую точность только для больших частот f_{BX} или в случае большого временного окна tg (то есть для больших N). В первом случае для увеличения точности необходимо увеличивать тактовую частоту, во втором — увеличивать длительность временного окна.

Время измерения частоты по первому методу составляет T_{BX}. Для второго метода оно постоянно и равно длительности временного окна t_0.

Поэтому желательно было бы соединить достоинства обоих методов, чтобы частота f_{BX} измерялась бы достаточно быстро и с заданной точностью (с погрешностью, не меньшей заданной). Это возможно при использовании комбинированного метода (рис. 5.3 *\в)*. При данном методе импульсы тактовой частоты с периодом T_T подсчитываются в течение М полных периодов входного сигнала. При этом количество сосчитанных импульсов N определяет точность измерения (относительная погрешность не превышает 1/N). Значит, необходимо обеспечить, чтобы N было достаточно большим, например, при N>100 относительная погрешность не превысит 1 %, а при N > 1000 она будет меньше 0,1 %. Обеспечить достаточную величину N можно простым выбором числа М.

Недостаток данного комбинированного метода состоит в том, что измеренное значение частоты необходимо вычислять. Так как при этом методе выполняется равенство $MT_{BX} = NT_T$, следовательно, $f_{BX} = M/(NT_T)$. Однако при использовании компьютера или микроконтроллера такое вычисление не представляет особого труда. Зато данный комбинированный

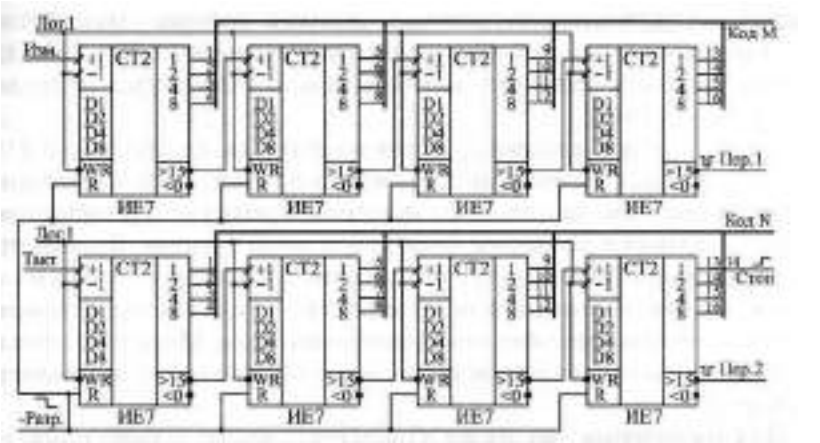


Рис. 5.32. Счетчики измерителя частоты входного сигнала

метод позволяет измерять частоту входного сигнала в широком диапазоне быстро и с заданной точностью. Поэтому мы подробно рассмотрим практическую реализацию именно этого метода.

В основе схемы измерителя частоты по комбинированному методу (рис. 5.32) — два 16-разрядных счетчика на основе микросхем ИЕ7, одновременно работающих в режиме прямого счета. На тактовый вход одного счетчика (верхнего по рисунку) подается измеряемый сигнал "Изм.", на тактовый вход второго (нижнего по рисунку) счетчика — тактовый сигнал образцовой частоты "Такт". Выходные коды обоих счетчиков (соответственно, M и N) используются после окончания измерения для вычисления значения частоты входного сигнала.

Работа счетчиков разрешается отрицательным сигналом "-Разр." по фронту (например, положительному) входного сигнала. После окончания измерения по такому же фронту входного сигнала поступление сигналов "Изм." и "Такт" запрещается. То есть счет производится в течение целого числа периодов входного сигнала.

Выход "Стоп" (положительный фронт) говорит о том, что код N достиг достаточной величины (в нашем случае ≥ 8192), и, следовательно, можно останавливать измерение (но только по ближайшему фронту входного сигнала). Иначе говоря, код N в конце измерения будет не менее 8192, и поэтому погрешность измерения частоты входного сигнала не превысит 1/8192 или 0, 012 %.

Для правильной работы схемы частота входного сигнала должна быть не более тактовой частоты $f_T = 1/T_y$ и не менее $f_T/65536$. Если она будет слишком малой, то наступит переполнение нижнего счетчика (выработается сигнал переноса "-Пер. 2"). Если же она будет слишком

большой, то наступит переполнение верхнего счетчика (выработается сигнал переноса "Пер. 1"). Например, при тактовой частоте 10 **МГц** измеряемая частота входного сигнала может находиться в пределах от 152,6 **Гц** до 10 МГц.

Полное время измерения будет изменяться в пределах от $8192T_T$ до $(8192T_T + 2T_{BX})$. Один период T_{BX} может прибавляться к времени измерения из-за того, что после разрешения измерения счет начинается не сразу, а только с приходом фронта входного сигнала. Второй период T_{BX} может прибавляться за счет того, что счет заканчивается не сразу после достижения кодом N величины 8192, а только с приходом нужного (положительного) фронта входного сигнала. Максимальное время измерения в любом случае не превышает $65536T_T$ для всех измеряемых частот.

Для увеличения диапазона измеряемых частот можно применить предварительный управляемый делитель частоты (рис. 5.33). Он обеспечивает выбор период тактового сигнала из ряда 100 **нс**, 400 **нс**, 1,6 **мкс**, 6,4 **мкс** и 25,6 **мкс** с помощью кода такта. В результате применения этого делителя при минимальной тактовой частоте возможно измерение частоты входного сигнала до 0,6 Гц. Естественно, переход на каждый следующий диапазон измеряемых частот может увеличить время измерения в 4 раза, но точность измерения в любом случае останется прежней.

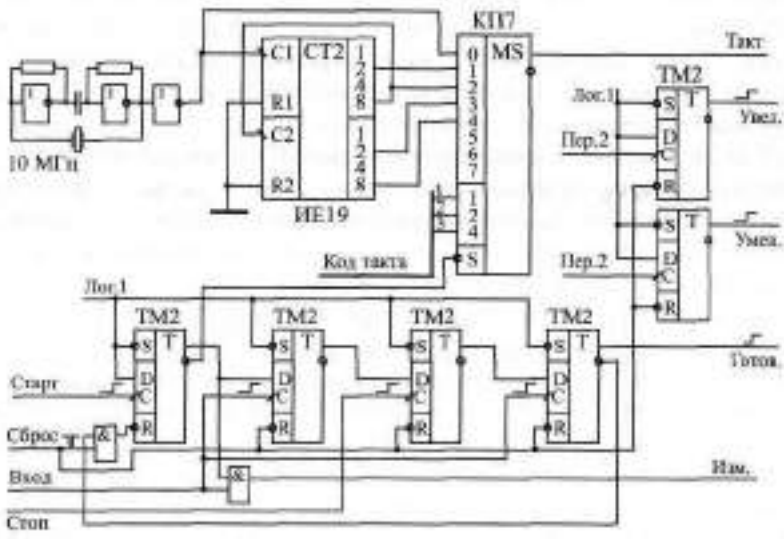


Рис. 5.33. Делитель частоты и схема управления для измерителя частоты входного сигнала

Схема управления измерителем частоты, также показанная на рис. 5.33, включает в себя цепочку из четырех последовательно срабатывающих триггеров (ТМ2). Перед началом измерения все эти триггеры сбрасываются в нуль сигналом "-Сброс".

Первый триггер перебрасывается в единицу по сигналу начала измерения "Старт" (положительный фронт). При этом разрешается прохождение подсчитываемых импульсов "Изм." и "Такт" на вход счетчиков (рис. 5.32). Одновременно разрешается работа второго триггера.

Второй триггер перебрасывается в единицу по положительному фронту входного сигнала. Тем самым он с помощью сигнала со своего инверсного выхода разрешает работу счетчиков (сигнал "-Разр."). Одновременно разрешается работа третьего триггера.

Третий триггер перебрасывается в единицу по сигналу "Стоп" (то есть при достижении кодом N числа 8192). Он разрешает работу четвертого триггера.

Наконец, четвертый триггер перебрасывается по положительному фронту входного сигнала и сигналом со своего инверсного выхода сбрасывает первый триггер. Поступление сигналов "Изм." и "Такт" прекращается. Выходной сигнал четвертого триггера служит флагом готовности выходных кодов N и M, которые необходимо прочитать для дальнейшего вычисления частоты. Перед новым измерением надо подать сигнал "Сброс".

Кроме четырех управляющих триггеров, в схему управления введены еще два триггера (справа на рисунке), выходные сигналы которых служат флагами переполнения и показывают после окончания измерения, правильно ли сработал измеритель частоты. Перед началом измерения оба эти триггера сбрасываются по сигналу "Сброс". Если частота входного сигнала в нужных пределах, то оба триггера останутся в нуле. Если частота входного сигнала очень большая, то сработает верхний по рисунку триггер по входному сигналу переноса "Пер. 1" (см. рис. 5.32) и выдаст сигнал "Увел.", говорящий о том, что надо поднять частоту тактового сигнала (если это возможно). Если же частота входного сигнала слишком мала, то сработает нижний по рисунку триггер по входному сигналу переноса "Пер. 2" (см. рис. 5.32) и выдаст сигнал "Умен.", говорящий о том, что надо уменьшить частоту тактового сигнала (если возможно).

Лекция 10. Синхронные счетчики

В этой рассматриваются синхронные счетчики, их алгоритмы работы, параметры, типовые схемы включения, а также способы реализации на их основе некоторых часто встречающихся функций.

Ключевые слова: синхронный (параллельный) счетчик, разрешение счета, разрешение переноса, вход направления счета, реверсивный счетчик, разрешение записи, накапливание задержек, формирователь интервала, быстродействующий генератор импульсов.

5.3. Синхронные счетчики

Синхронные (или параллельные) счетчики представляют собой наиболее быстродействующую разновидность счетчиков. Наращивание их разрядности при соблюдении определенных условий не приводит к увеличению полной задержки срабатывания. То есть можно считать, что именно синхронные счетчики работают как идеальные счетчики, все разряды которых срабатывают одновременно, параллельно. Задержка срабатывания счетчика в этом случае примерно равна задержке срабатывания одного триггера. Достигается такое быстродействие существенным усложнением внутренней структуры микросхемы.

Вместе с тем недостатком синхронных счетчиков является более сложное управление их работой по сравнению с асинхронными счетчиками и с синхронными счетчиками с асинхронным переносом. Поэтому синхронные счетчики целесообразно применять только в тех случаях, когда действительно требуется очень высокое быстродействие, очень высокая скорость переключения разрядов. Иначе усложнение схемы управления может быть не оправдано.

Временная диаграмма работы синхронного счетчика (рис. 5.34) отличается от временной диаграммы синхронного счетчика с асинхронным

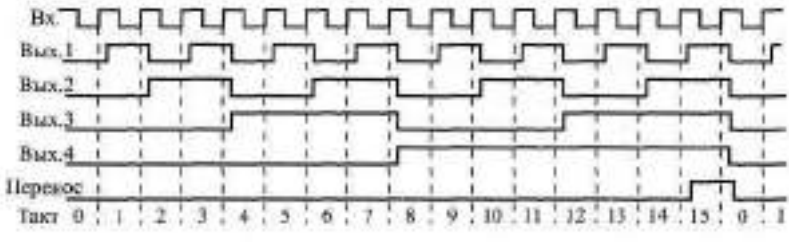


Рис. 5.34. Временная диаграмма работы синхронных двоичных счетчиков

переносом способом формирования сигнала переноса, используемого при каскадировании счетчиков для увеличения разрядности. Сигнал переноса CR (от английского "Carry") вырабатывается в данном случае тогда, когда все выходы счетчика устанавливаются в единицу (при прямом счете) или в нуль (при обратном, инверсном счете). Входной тактовый сигнал в образовании сигнала переноса при этом не участвует.

При каскадировании (совместном включении для увеличения разрядности), например, двух счетчиков тактовые входы С обоих счетчиков объединяются, а сигнал переноса первого счетчика подается на вход разрешения счета (ECT) второго счетчика. В результате второй счетчик будет считать каждый шестнадцатый входной тактовый импульс (так как он будет срабатывать только при переносе от первого счетчика). Выходные сигналы второго счетчика будут переключаться по фронту общего тактового сигнала одновременно с выходными сигналами первого счетчика. Условием правильной работы будет в данном случае следующее: за период тактового сигнала должен успеть выработаться сигнал переноса первого счетчика.

В стандартные серии микросхем входят несколько разновидностей синхронных (параллельных) счетчиков (рис. 5.35). Различаются они способом счета (двоичные или двоично-десятичные, реверсивные или не реверсивные) и управляющими сигналами (наличием или отсутствием сигнала сброса). Все счетчики считают по положительному фронту тактового сигнала, все имеют выход переноса CR и входы расширения для каскадирования. Все счетчики имеют возможность параллельной записи информации.

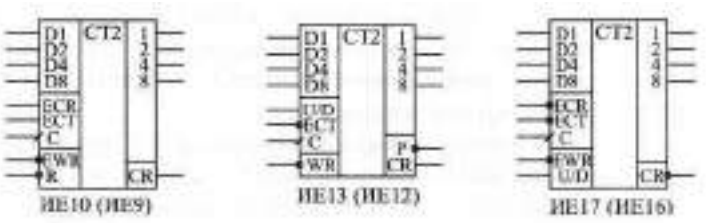


Рис. 5.35. Синхронные счетчики стандартных серий

Таблица 5.5. Режимы работы счетчиков ИЕ9 и ИЕ10

Входы					Режим
-R	-EWR	ECR	ECT	C	
0	X	X	X	X	Сброс
1	0	X	X	0->1	Параллельная запись
1	1	0	X	X	Хранение
1	1	X	0	X	Хранение
1	1	1	1	0->1	Прямой счет

Счетчики ИЕ9 и ИЕ10 отличаются друг от друга только тем, что ИЕ9 — двоично-десятичный, а ИЕ10 — двоичный. Микросхемы имеют вход асинхронного сброса -R, по нулевому уровню на котором все выходы счетчика сбрасываются в нуль. Счет (только прямой) производится по положительному фронту на тактовом входе С. Параллельная запись осуществляется синхронно, по положительному фронту на тактовом входе С при установленном в нуль сигнале разрешения записи -ЕWR. Сигналы ECR ("Enable Carry" — разрешение переноса) и ECT ("Enable Count" — разрешение счета) используются при каскадировании микросхем. Разница между этими сигналами в том, что сигнал ECR не только запрещает счет, как сигнал ECT, но еще и запрещает выработку сигнала переноса CR. Счет идет при единичных сигналах на обоих входах ECT и ECT и при единичном сигнале на входе -EWR. Положительный сигнал переноса CR вырабатывается при максимально возможном коде на выходах счетчика (15 для ИЕ10 и 9 для ИЕ9) и при положительном сигнале на входе ECR. Таблица режимов работы счетчиков ИЕ9 и ИЕ10 представлена в табл. 5.5.

Счетчики ИЕ12 (двоично-десятичный) и ИЕ13 (двоичный) отличаются от ИЕ9 и ИЕ10 тем, что они реверсивные, то есть допускают как прямой, так и обратный счет. Кроме того, у них несколько другое управление. Считают они также по положительному фронту тактового сигнала С при нулевом уровне на входе разрешения счета ECT. Прямой счет осуществляется при нулевом уровне на входе управления U/D, обратный — при единичном уровне на входе U/D. Переключение уровней на входах U/D и ECT допускается только при положительном сигнале на тактовом входе С. Сброс счетчиков ИЕ12 и ИЕ13 в нуль не предусмотрен, зато имеется возможность асинхронной параллельной записи информации по нулевому уровню сигнала параллельной записи -WR.

Положительный сигнал на выходе параллельного переноса CR появляется при достижении максимального кода (15 для ИЕ13 и 9 для ИЕ12) при прямом счете или при достижении нулевого кода при обратном (инверсном) счете. Имеется также выход последовательного переноса Р, отрицательный импульс на котором вырабатывается при положительном сигнале CR и повторяет отрицательный импульс на тактовом входе С (аналогично рассмотренным ранее счетчикам ИЕ6 и ИЕ7).

Режимы работы счетчиков ИЕ12 и ИЕ13 представлена в табл. 5.6.

Микросхемы ИЕ16 (двоично-десятичный счетчик) и ИЕ17 (двоичный счетчик) отличаются от рассмотренных синхронной параллельной записью по фронту тактового сигнала С, возможностью прямого и обратного счета и отсутствием сигнала сброса в нуль.

Срабатывают счетчики ИЕ16 и ИЕ17 по положительному фронту тактового сигнала С. При нулевом уровне на входе разрешения записи EWR по фронту сигнала С в счетчик записывается информация со входов дан-

ных D1, D2, D4, D8. При единичном уровне на входе -EWR по положительному фронту сигнала С происходит счет. Направление счета определяется входом U/D: при единице на этом входе счет прямой, при нуле — обратный. Имеются два входа расширения: вход разрешения счета -ECT и вход разрешения переноса -ECR. Различаются эти два входа тем, что сигнал -ECR не только запрещает счет, как сигнал -ECT, но еще и запрещает выработку сигнала переноса. Переключение уровней на входах U/D, -ECT и -ECR надо производить только при единичном уровне на тактовом входе С.

Таблица 5.6. Режимы работы счетчиков ИЕ12 и ИЕ13

Входы				Режим
-WR	U/D	-ECR	C	
0	X	X	X	Параллельная запись
1	X	1	X	Хранение
1	0	0	0->1	Прямой счет
1	1	0	0->1	Обратный счет

Отрицательный сигнал переноса -CR (синхронный) вырабатывается при достижении на выходах счетчика максимального кода (15 для ИЕ7 или 9 для ИЕ16) при прямом счете или нулевого кода при обратном счете.

Режимы работы счетчиков ИЕ16 и ИЕ17 приведены в табл. 5.7.

Таблица 5.7. Режимы работы счетчиков ИЕ16 и ИЕ17

Входы					Режим
-EWR	U/D	-ECT	-ECR	C	
0	X	X	X	0->1	Параллельная запись
1	1	0	0	0->1	Прямой счет
1	0	0	0	0->1	Обратный счет
1	X	1	X	X	Хранение
1	X	X	1	X	Хранение

Возможности применения синхронных (параллельных) счетчиков очень широки. Достаточно сказать, что они без всяких проблем могут заменить во всех схемах как асинхронные (последовательные) счетчики, так и синхронные счетчики с асинхронным (последовательным) переносом. При необходимости достижения максимального быстродействия они имеют большие преимущества по сравнению со всеми другими счетчиками. Их выходной код устанавливается одновременно при любом ко-

личестве разрядов без применения дополнительных выходных регистров (которые требовались в случае асинхронных счетчиков и синхронных счетчиков с асинхронным переносом).

Мы рассмотрим здесь всего несколько схем, иллюстрирующих характерные особенности именно синхронных счетчиков.

Сначала остановимся на методах каскадирования счетчиков. В отличие от других типов счетчиков, синхронные счетчики можно соединять различными способами, причем способ соединения различен для разного количества микросхем. В качестве примера возьмем микросхемы ИЕ17.

При объединении двух счетчиков (рис. 5.36) никаких проблем не возникает: выход переноса -CR младшего счетчика соединяется со входом разрешения счета старшего счетчика -ECT. На входы -ECR обоих счетчиков подается нулевой уровень. Условие правильной работы будет простым и легко выполнимым: период тактового сигнала C не должен быть меньше, чем задержка выработки сигнала переноса CR.

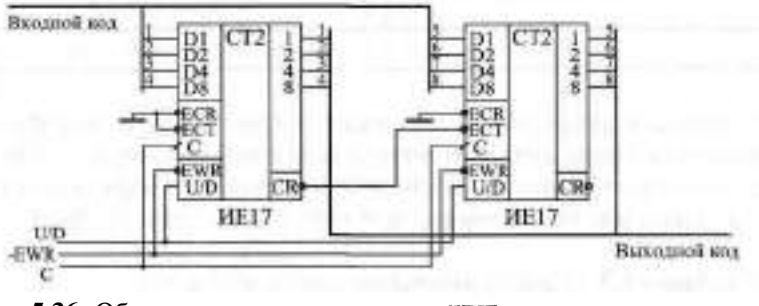


Рис. 5.36. Объединение двух счетчиков ИЕ17

При объединении трех счетчиков ситуация несколько усложняется (рис. 5.37). Сигнал с выхода переноса первого счетчика подается на входы ECT второго и третьего счетчиков. Сигнал с выхода переноса второго счетчика подается на вход ECR третьего счетчика. В результате третий счетчик будет считать только тогда, когда имеется перенос как у первого, так и у второго счетчика. На рисунке для простоты не показано подключение входных и выходных сигналов, не участвующих в каскадировании.

Условие правильной работы схемы остается тем же, что и в случае двух счетчиков: период тактового сигнала C не должен быть меньше задержки выработки сигнала переноса CR.

При объединении четырех (и более) счетчиков уже возникает проблема, так как у старших счетчиков не остается свободных управляющих входов для собирания всех сигналов переноса более младших счетчиков. Поэтому в данном случае используется способность входного сигнал -ECR запрещать выходной сигнал переноса CR (рис. 5.38). На четвертый и по-

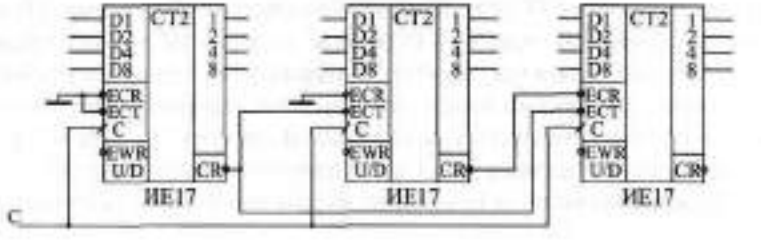


Рис. 5.37. Объединение трех счетчиков ИЕ17

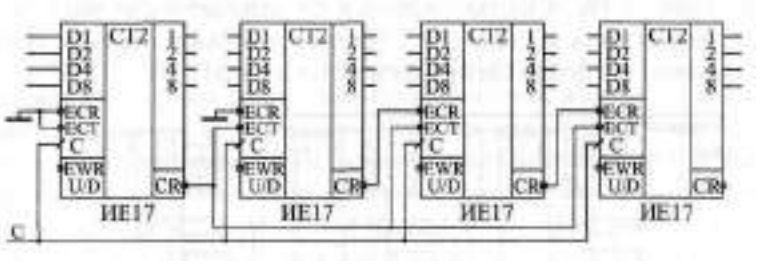


Рис. 5.38. Объединение четырех счетчиков ИЕ17

следующие счетчики подаются уже сигналы переноса не со всех предыдущих счетчиков, а только с первого и с предыдущего. На рисунке для простоты не показано подключение входов и выходов, не участвующих непосредственно в каскадировании.

При таком включении происходит уже накапливание задержки сигналов переноса. Максимальной задержка будет для сигнала переноса второго счетчика. Условие правильной работы всех счетчиков будет следующее: период тактового сигнала С не должен быть меньше, чем максимальная суммарная задержка сигналов переноса до входа последнего счетчика. При объединении четырех счетчиков в эту максимальную задержку входят задержка сигнала переноса -CR микросхемы относительно фронта сигнала С и задержка сигнала переноса -CR относительно сигнала -ECR. При объединении пяти счетчиков добавится еще одна задержка сигнала переноса -CR относительно сигнала -ECR и т. д. Поэтому с увеличением количества объединяемых счетчиков будет снижаться допустимая тактовая частота.

При необходимости объединения большого количества счетчиков (большего четырех) можно избежать накопления суммарной задержки переноса, включив на входах старших счетчиков ECT логические элементы И с нужным числом входов. Эти элементы должны собирать все сигналы переноса с более младших счетчиков, то есть на их выходах должен быть нуль

тогда, когда сигналы CR всех предыдущих счетчиков нулевые. При этом, правда, в суммарную задержку переноса, которая не должна превышать периода тактового сигнала С, войдут задержки этих самых элементов ИЛИ

В любом случае при выполнении условия правильной работы счетчиков схема будет работать как идеальный счетчик, то есть все разряды многокаскадного счетчика будут переключаться одновременно.

А теперь рассмотрим некоторые схемы на основе синхронных счетчиков.

Управляемый делитель частоты с коэффициентом пересчета, задаваемым входным кодом, реализуется на синхронных счетчиках довольно просто (рис. 5.39). Сигнал переноса CR старшего счетчика подается на вход разрешения записи -EWR. Счетчики работают в режиме обратного счета (на вход U/0 подан сигнал логического нуля).

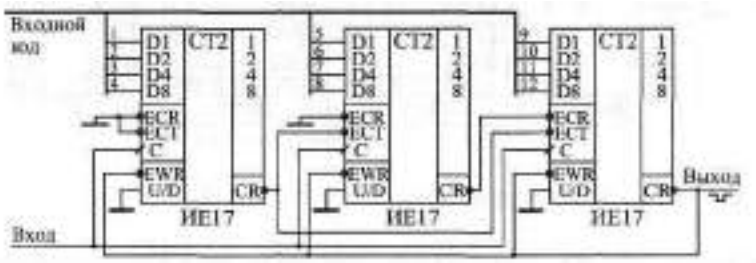


Рис. 5.39. Управляемый делитель частоты

При достижении всеми счетчиками нулевого кода вырабатывается сигнал переноса CR, переводящий счетчики в режим параллельной записи входного управляющего кода. Следующим положительным фронтом тактового сигнала С входной код записывается в счетчики. Это приводит к новому циклу счета от входного кода до нуля.

Коэффициент пересчета делителя частоты равен (N+1), где N — входной код, который может принимать значения от 1 до (2^n-1), где n — количество разрядов кода. Условие правильной работы делителя частоты следующее: период тактового сигнала не должен быть меньше полной задержки переноса. Длительность выходного сигнала делителя частоты равна периоду тактовой частоты.

Следующая схема — формирователь временного интервала заданной длительности (рис. 5.40) демонстрирует, как надо использовать выходной сигнал переноса синхронных счетчиков при необходимости организации разового (не периодического) цикла работы.

Работа формирователя начинается по короткому отрицательному импульсу "-Старт", перебрасывающему управляющий триггер в единицу

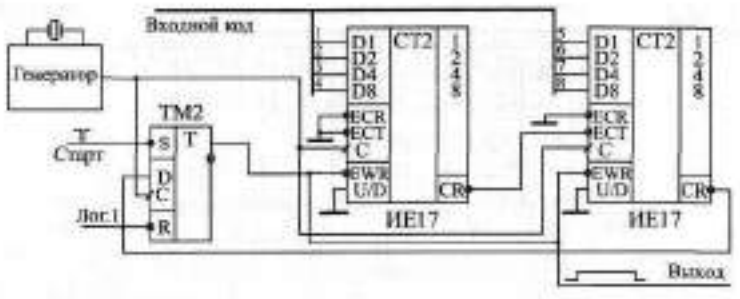


Рис. **5.40.** Формирователь интервала заданной длительности

и начинающему выходной сигнал. Положительный сигнал с выхода триггера переводит 8-разрядный синхронный счетчик из режима параллельной записи входного кода в режим счета (по входу -EWR). Счет на уменьшение идет по положительным фронтам тактового сигнала с генератора. Когда счетчик досчитает до нуля, следующим положительным фронтом тактового сигнала нулевой сигнал переноса -CR будет записан в триггер. Тем самым будет завершен выходной сигнал, а счетчик будет переведен в режим параллельной записи. Следующий цикл работы формирователя начнется по сигналу "-Старт".

В данном случае триггер, обрабатывающий сигнал переноса, работает синхронно со счетчиками, так как тактируется тем же (положительным) фронтом единого тактового сигнала. Длительность выходного сигнала будет находиться в интервале от NT до (N+1) T, где T - период тактового сигнала с генератора, а N — входной код от 0 до 255.

Посмотрим, как на синхронных счетчиках можно построить генератор прямоугольных импульсов с регулируемыми длительностью импульса и длительностью паузы, который был рассмотрен в предыдущем разделе (см. рис. 5.28 и 5.29). Будем ориентироваться на достижение максимального быстродействия, то есть на максимально возможную тактовую частоту.

Схема управления будет мало отличаться от схемы рис. 5.29, поэтому мы остановимся только на схеме счетчиков импульса и паузы. Выберем разрядность обоих этих счетчиков равной 16. Тогда схема счетчиков импульса и паузы (рис. 5.41) будет включать в себя восемь микросхем счетчиков ИЕ17 и выходной триггер, а также логические элементы 4ИЛИ-НЕ для уменьшения задержек переноса. В данном случае очень удобно брать JK-триггер, так как он имеет два информационных входа и тактовый вход.

Триггер тактируется отрицательным фронтом сигнала C, а счетчики — положительным фронтом, поэтому для обеспечения синхронной работы всей схемы по одному фронту тактового сигнала сигнал на вход C триггера подается через инвертор.

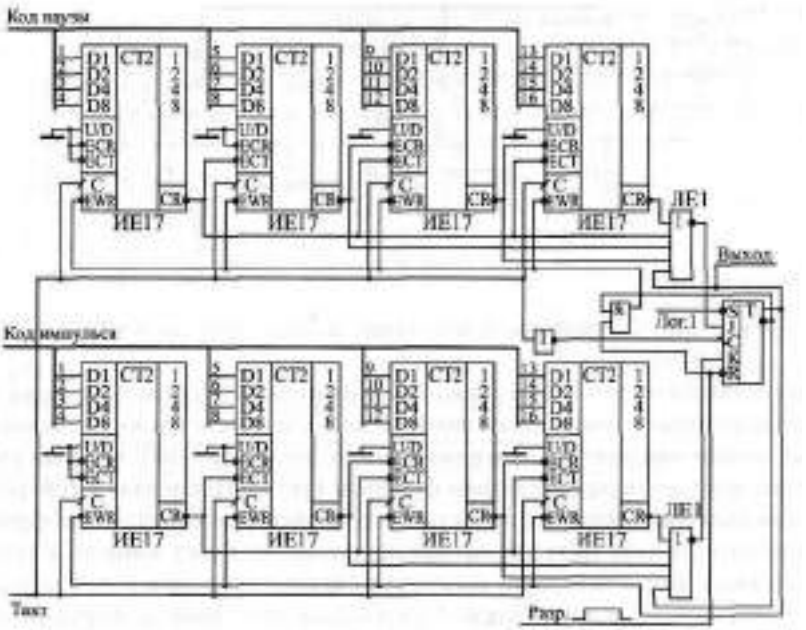


Рис. 5.41. Синхронные счетчики импульса и паузы для генератора прямоугольных импульсов

Суть работы схемы остается прежней: 16-разрядные счетчики импульса и паузы работают по очереди, что определяется управляющими сигналами с выходов триггера (прямого и инверсного). Счетчики считают на уменьшение (в режиме инверсного счета) от кода, параллельно записанного в них, до нуля.

До начала работы (сигнал "Разр." нулевой) оба счетчика находятся в состоянии параллельной записи и записывают в себя код импульса и паузы. После прихода положительного сигнала разрешения генерации "Разр." начинает счет верхний по рисунку счетчик (счетчик паузы).

Когда счетчик паузы досчитывает до нуля, его сигнал переноса записывается в триггер по входу J и перебрасывает выход триггера в единицу, что переводит счетчик паузы из состояния счета в состояние параллельной записи и запрещает поступления сигнала на вход J. Одновременно переходит в состояние счета нижний по рисунку счетчик (счетчик импульса), который, в свою очередь досчитав до нуля, перебрасывает триггер в нуль по входу K. Этот процесс периодически повторяется, пока разрешена генерация (то есть сигнал "Разр." — положительный).

Сформулируем условия правильной работы схемы.

Во-первых, за период тактового сигнала должен успеть полностью сработать 16-разрядный счетчик, выполненный на четырех микросхемах синхронных счетчиков. То есть сигнал на входы ECR и ECT последнего счетчика должны успеть придти до следующего фронта тактового сигнала.

Во-вторых, за период тактового сигнала должна успеть сработать цепочка из инвертора (ЛН1), триггера (ТВ11) и элемента 2И (ЛИ1). Это более мягкое требование, чем предыдущее, если, конечно, взять перечисленные элементы из быстродействующих серий КР531 или КР1531.

Рассмотренный переход на синхронные счетчики позволяет повысить максимальную частоту тактового сигнала генератора прямоугольных импульсов по меньшей мере вдвое (до 20 МГц) по сравнению со схемой на синхронных счетчиках с асинхронным переносом.

Наконец, последнее применение синхронных счетчиков, которое мы рассмотрим, связано с их возможностью параллельной записи по фронту тактового сигнала. То есть в режиме параллельной записи счетчик представляет собой регистр, срабатывающий по фронту тактового сигнала. Благодаря этой особенности при объединении нескольких счетчиков их выходные коды можно последовательно считывать с выходов последнего в цепочке, старшего счетчика (рис. 5.42). Счетчики в данном случае образуют своеобразный многоразрядный сдвиговый регистр.

Режим работы схемы определяется управляющим сигналом "Счет/Сдвиг". При высоком уровне этого сигнала счетчики находятся в режиме прямого счета по фронту сигнала "Такт". При низком уровне сигнала счетчики переходят в режим последовательного счета 12-разрядного счетчика через четыре разряда правого по рисунку счетчика. Первым читается состояние старшего счетчика, последним — младшего. Сдвиг выходного кода происходит по положительному фронту тактового сигнала. После трех импульсов тактового сигнала во все три счетчика оказывается записанным нулевой код, то есть схема готова к режиму прямого счета.

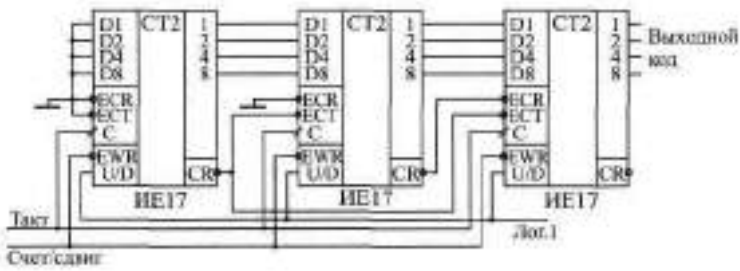


Рис. 5.42. Последовательное чтение выходного кода многокаскадного счетчика

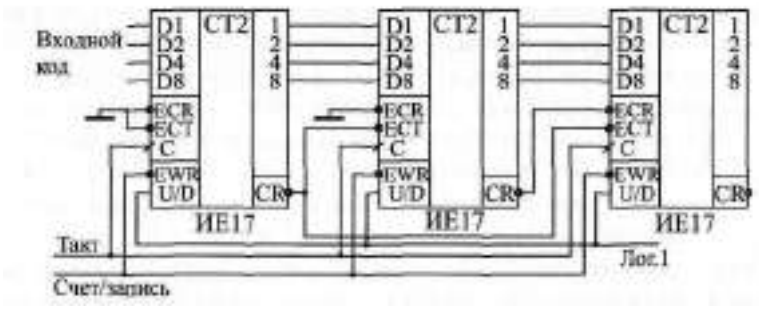


Рис. **5.43.** Последовательная запись в счетчики исходного состояния

И точно такая же последовательная перезапись информации из счетчика в счетчик позволяет с помощью 4-разрядных входных кодов записать исходное состояние нескольких последовательно соединенных счетчиков (рис. 5.43).

Перед началом работы схема переводится в состояние параллельной записи нулевым уровнем сигнала "Счет/Запись". При этом 4-разрядные коды, которые надо записать во все счетчики, по очереди подаются на вход первого (младшего) счетчика и сдвигаются по направлению к старшему счетчику по положительному фронту тактового сигнала С. Для записи всех трех счетчиков необходимо подать три тактовых импульса подряд. Причем первым надо записывать код, предназначенный для старшего (на рисунке справа) счетчика, а последним — код, предназначенный для младшего (на рисунке слева) счетчика.

Глава 6. Применение микросхем памяти

Лекция 11. Постоянная память

В лекции рассказывается о типах микросхем памяти и о микросхемах постоянной памяти, их алгоритмах работы, параметрах, типовых схемах включения, а также о способах реализации на их основе некоторых часто встречающихся **функций.**

Ключевые слова: память, ЗУ, организация памяти, типы памяти, постоянная память, ПЗУ, ПГТЗУ, ROM, адрес памяти, разряды данных, ячейка памяти, программатор, карта прошивки памяти, увеличение объема памяти, табличный вычислитель, знакогенератор, генератор импульсных последовательностей, микропрограммный автомат, дешифратор манчестерского кода.

Микросхемы памяти (или просто память, или запоминающие устройства — ЗУ, английское "Memory") представляют собой следующий шаг на пути усложнения цифровых микросхем по сравнению с микросхемами, рассмотренными ранее. Память — это всегда очень сложная структура, включающая в себя множество элементов. Правда, внутренняя структура памяти — регулярная, большинство элементов одинаковые, связи между элементами сравнительно простые, поэтому функции, выполняемые микросхемами памяти, не слишком сложные.

Память, как и следует из ее названия, предназначена для запоминания, хранения каких-то массивов информации, проще говоря, наборов, таблиц, групп цифровых кодов. Каждый код хранится в отдельном элементе памяти, называемом ячейкой памяти. Основная функция любой памяти как раз и состоит в выдаче этих кодов на выходы микросхемы по внешнему запросу. А основной параметр памяти — это ее объем, то есть количество кодов, которые могут в ней храниться, и разрядность этих кодов.

Для обозначения количества ячеек памяти используются следующие специальные единицы измерения:

- 1К — это 1024, то есть 2^{10} (читается "кило-" или "ка-"), примерно равно одной тысяче;
- 1М — это 1048576, то есть 2^{20} (читается "мега-"), примерно равно одному миллиону;
- 1Г — это 1073741824, то есть 2^{30} (читается "гига-"), примерно равно одному миллиарду.

Принцип организации памяти записывается следующим образом: сначала пишется количество ячеек, а затем через знак умножения (косой крест) — разрядность кода, хранящегося в одной ячейке. Например, организация памяти 64Kx8 означает, что память имеет 64K (то есть 65536) ячеек и каждая ячейка — восьмиразрядная. А организация памяти 4M x 1 означает, что память имеет 4M (то есть 4194304) ячеек, причем каждая ячейка имеет всего один разряд. Общий объем памяти измеряется в байтах (килобайтах — Кбайт, мегабайтах - Мбайт, гигабайтах — Гбайт) или в битах (килобитах — Кбит, мегабитах — Мбит, гигабитах — Гбит).

В зависимости от способа занесения (записи) информации и от способа ее хранения, микросхемы памяти разделяются на следующие основные типы:

- Постоянная память (ПЗУ — постоянное запоминающее устройство, ROM — Read Only Memory — память только для чтения), в которую информация заносится один раз на этапе изготовления микросхемы. Такая память называется еще масочным ПЗУ. Информация в памяти не пропадает при выключении ее питания, поэтому ее еще называют энергонезависимой памятью.
- Программируемая постоянная память (ППЗУ — программируемое ПЗУ, PROM — Programmable ROM), в которую информация может заноситься пользователем с помощью специальных методов (ограниченное число раз). Информация в ППЗУ тоже не пропадает при выключении ее питания, то есть она также энергонезависимая.
- Оперативная память (ОЗУ — оперативное запоминающее устройство, RAM — Random Access Memory — память с произвольным доступом), запись информации в которую наиболее проста и может производиться пользователем сколько угодно раз на протяжении всего срока службы микросхемы. Информация в памяти пропадает при выключении ее питания.

Существует множество промежуточных типов памяти, а также множество подтипов, но указанные — самые главные, принципиально отличающиеся друг от друга. Хотя, разница между ПЗУ и ППЗУ с точки зрения разработчика цифровых устройств, как правило, не так уж велика. Только в отдельных случаях, например, при использовании так называемой флэш-памяти (flash-memory), представляющей собой ППЗУ с многократным электрическим стиранием и перезаписью информации, эта разница действительно чрезвычайно важна. Можно считать, что флэш-память занимает промежуточное положение между ОЗУ и ПЗУ.

В общем случае любая микросхема памяти имеет следующие информационные выводы (рис. 6.1):

- Адресные выводы (входные), образующие шину адреса памяти. Код на адресных линиях представляет собой двоичный номер

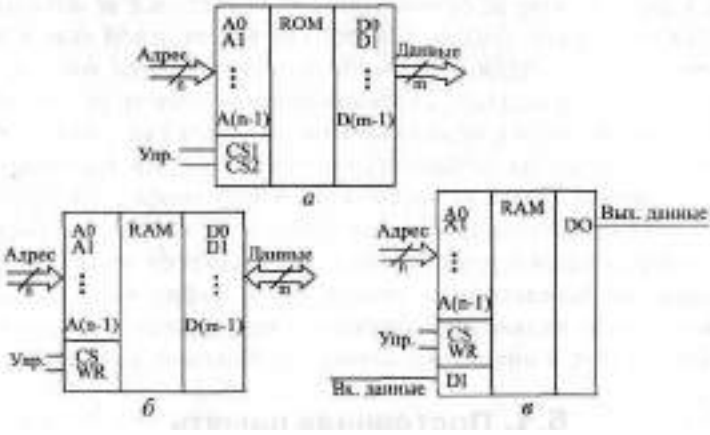


Рис. 6.1. Микросхемы памяти: ПЗУ *(а),* ОЗУ с двунаправленной шиной данных (б), ОЗУ с раздельными шинами входных и выходных данных *(в)*

ячейки памяти, к которой происходит обращение в данный момент. Количество адресных разрядов определяет количество ячеек памяти: при количестве адресных разрядов n количество ячеек памяти равно 2^n.

- Выводы данных (выходные), образующие шину данных памяти. Код на линиях данных представляет собой содержимое той ячейки памяти, к которой производится обращение в данный момент. Количество разрядов данных определяет количество разрядов всех ячеек памяти (обычно оно бывает равным 1, 4, 8, 16). Как правило, выходы данных имеют тип выходного каскада ОК или ЗС.
- В случае оперативной памяти, помимо выходной шины данных, может быть еще и отдельная входная шина данных, на которую подается код, записываемый в выбранную ячейку памяти. Другой возможный вариант — совмещение входной и выходной шин данных, то есть двунаправленная шина, направление передачи информации по которой определяется управляющими сигналами. Двунаправленная шина применяется обычно при количестве разрядов шины данных 4 или более.
- Управляющие выводы (входные), которые определяют режим работы микросхемы. В большинстве случаев у памяти имеется вход выбора микросхемы CS (их может быть несколько, объединенных по функции И). У оперативной памяти также обязательно есть вход записи WR, активный уровень сигнала на котором переводит микросхему в режим записи.

Мы в данной главе не будем, конечно, изучать все возможные разновидности микросхем памяти, для этого не хватит целой книги. К тому же эта информация содержится в многочисленных справочниках. Микросхемы памяти выпускаются десятками фирм во всем мире, поэтому даже перечислить все их не слишком просто, не говоря уже о том, чтобы подробно рассматривать их особенности и параметры. Мы всего лишь рассмотрим различные схемы включения типичных микросхем памяти для решения наиболее распространенных задач, а также методы проектирования некоторых узлов и устройств на основе микросхем памяти. Именно это имеет непосредственное отношение к цифровой схемотехнике. И именно способы включения микросхем мало зависят от характерных особенностей той или иной микросхемы той или иной фирмы.

6.1. Постоянная память

В данном разделе мы не будем говорить о флэш-памяти, так как это отдельная большая тема. Мы ограничимся только простейшими микросхемами ПЗУ и ППЗУ, информация в которые заносится раз и навсегда (на этапе изготовления или же самим пользователем). Мы также не будем рассматривать здесь особенности оборудования для программирования ППЗУ (так называемых программаторов), принципы их построения и использования, — это отдельная большая тема. Мы будем считать, что нужная нам информация может быть записана в ПЗУ или ППЗУ, а когда, как, каким способом она будет записана, нам не слишком важно. Все эти допущения позволят нам сосредоточиться именно на схемотехнике узлов и устройств на основе ПЗУ и ППЗУ (для простоты будем называть их в дальнейшем просто ПЗУ).

Упомянем здесь только, что ППЗУ делятся на репрограммируемые или перепрограммируемые ПЗУ (РПЗУ, EPROM — Erasable Programmable ROM), то есть допускающие стирание и перезапись информации, и однократно программируемые ПЗУ. В свою очередь, РПЗУ делятся на ПЗУ, информация в которых стирается электрическими сигналами (EEPROM - Electrically Erasable Programmable ROM), и на ПЗУ, информация в которых стирается ультрафиолетовым излучением через специальное прозрачное окошко в корпусе микросхемы (собственно EPROM — Erasable Programmable ROM). Запись информации в любые ППЗУ производится с помощью подачи определенных последовательностей электрических сигналов (как правило, повышенного напряжения) на выводы микросхемы.

Фирмами-производителями цифровых микросхем выпускается немало самых разнообразных ПЗУ и ППЗУ. Различаются микросхемы постоянной памяти своим объемом (от 32 байт до 8 Мбайт и более), органи-

зацией (обычно количество разрядов данных бывает 4, 8 или 16), способами управления (назначением управляющих сигналов), типами выходных каскадов (обычно О К или 3С), быстродействием (обычно задержка составляет от единиц до сотен наносекунд). Но суть всех микросхем ПЗУ остается одной и той же: имеется шина адреса, на которую надо подавать код адреса нужной ячейки памяти, имеется шина данных, на которую выдается код, записанный в адресуемой ячейке, и имеются входы управления, которые разрешают или запрещают выдачу информации из адресуемой ячейки на шину данных.

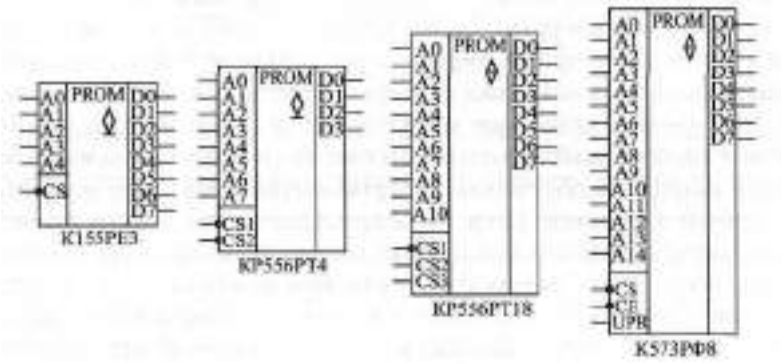


Рис. 6.2. Примеры микросхем ППЗУ отечественного производства

На рис. 6.2 представлены для примера несколько простейших и типичных микросхем постоянной памяти.

Микросхема К155РЕ3 (аналог — N8223N) представляет собой однократно программируемое ППЗУ с организацией 32 х 8. Исходное состояние (до программирования) — все биты всех ячеек нулевые. Для программирования (записи информации) используется специальный программатор, подающий на разряды данных импульсы высокого напряжения. Тип выходных каскадов — открытый коллектор, то есть обязательно надо включать на выходах резисторы, подсоединенные к шине питания. Имеется один управляющий вход -CS, при положительном уровне сигнала на котором на всех выходах устанавливаются единицы.

Микросхема КР556РТ4 (аналог — I3601) — это также однократно программируемая постоянная память с организацией 256х4. Исходное состояние (до программирования): все биты всех ячеек нулевые. Тип выходных каскадов — ОК. Два управляющих входа CS1 и -CS2 объединены по принципу И, то есть для разрешения работы микросхемы (для перевода выходов в активное состояние) оба эти сигнала должны быть нулевыми. Для записи информации в микросхему используется программатор.

Микросхема КР556РТ18 (аналог — HM7616l) также является однократно программируемым ППЗУ и имеет организацию 2Кх8. Тип выходов микросхемы — ЗС. Имеются три управляющих входа: один инверсный -CS1, два других — прямые CS2 и CS3, объединенных по функции И. Выходы данных переходят в активное состояние при нулевом уровне на -CS1 и при единичных уровнях на CS2 и CS3. Если входы управления используются для подачи управляющих сигналов (то есть выходы могут переходить в третье состояние) то на выходы надо включать нагрузочные резисторы, подключенные к шине питания. Исходное состояние микросхемы (до программирования) — все биты всех ячеек в единице.

Наконец, микросхема К573РФ8 (аналог — I27256) - это пример памяти РПЗУ с ультрафиолетовым стиранием информации. Чтобы перепрограммировать память, необходимо ее стереть, для чего в течение некоторого времени (обычно несколько минут) надо облучать микросхему через окошко в корпусе ультрафиолетовым светом (можно использовать медицинский кварцевый облучатель). Стертая микросхема имеет все биты, установленные в единицу. Затем проводится процедура записи с помощью программатора, несколько отличающегося от программаторов однократно программируемых микросхем. Управляющие входы -CS и -CE должны быть установлены в нуль для перевода выходов микросхемы в активное состояние. Имеется специальный вход UPR для подачи программирующего высокого напряжения, который при чтении информации из микросхемы надо подключать к напряжению питания. Тип выходных каскадов — ЗС. Микросхемы этого типа самые медленные, их задержки самые большие.

Основные временные характеристики микросхем ПЗУ — это две величины задержки. Задержка выборки адреса памяти - время от установки входного кода адреса до установки выходного кода данных. Задержка выборки микросхемы — время от установки активного разрешающего управляющего сигнала CS до установки выходного кода данных памяти. Задержка выборки микросхемы обычно в несколько раз меньше задержки выборки адреса.

Содержимое ПЗУ обычно изображается в виде специальной таблицы, называемой картой прошивки памяти. В таблице показывается содержимое всех ячеек памяти, причем в каждой строке записывается содержимое 16 (или 32) последовательно идущих (при нарастании кода адреса) ячеек. При этом, как правило, используется 16-ричное кодирование.

Пример карты прошивки ПЗУ с организацией 256х8 показан в табл. 6.1 (все биты всех ячеек считаются установленными в единицу). Пользоваться таблицей очень просто. Например для того, чтобы посмотреть содержимое ячейки памяти с 16-ричным адресом 8А, надо взять строку таблицы с номером 80 и столбец таблицы с номером А (данная ячейка в таблице выделена жирным шрифтом).

Таблица 6.1. Пример карты прошивки ПЗУ

Адрес	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
10	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
20	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
30	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
40	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
50	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
60	FF	FF	FF		FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
70	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
80	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
90	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
E0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
F0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Любые микросхемы ПЗУ легко можно включать так, чтобы уменьшать или увеличивать количество адресных разрядов, то есть уменьшать или увеличивать количество используемых ячеек памяти. И то, и другое часто требуется при построении схем цифровых устройств.

Для уменьшения количества адресных разрядов необходимо на нужное число старших адресных входов подать нулевые сигналы. Каждый отключенный таким образом адресный разряд уменьшает количество ячеек ПЗУ вдвое. Например, на рис. 6.3 показано, как из микросхемы с организацией 2Кх8 сделать микросхему 512х8. Два старших разряда адреса памяти отключены (на них поданы нулевые сигналы). Использоваться будут только младшие (верхние в таблице прошивки) 512 ячеек, и только их надо будет программировать. Конечно, гораздо лучше подобрать микросхему именно с тем количеством ячеек, которое действительно необходимо в данной схеме, но это, к сожалению, возможно не всегда.

Задача увеличения количества адресных разрядов ПЗУ встречается значительно чаще задачи уменьшения количества адресных разрядов. В результате такого увеличения возрастает объем ПЗУ, объемы отдельных микросхем суммируются. Для увеличения адресных разрядов обычно

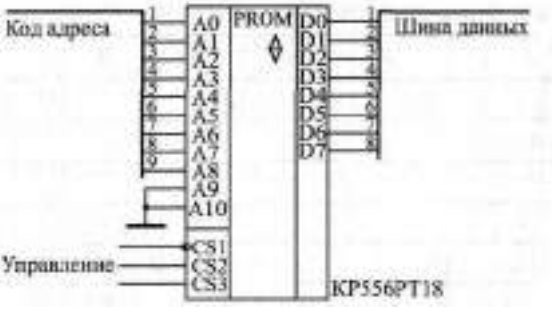


Рис. 6.3. Уменьшение количества адресных разрядов ПЗУ

применяются микросхемы дешифраторов (рис. 6.4). Младшие разряды шины адреса при этом подаются на объединенные адресные входы всех микросхем, а старшие — на управляющие (адресные) входы дешифратора. Выходные сигналы дешифратора разрешают работу всегда только одной микросхемы памяти. В результате на общую шину данных всех ПЗУ выдает свою информацию только одна микросхема. На рисунке для простоты не показаны выходные резисторы с разрядов данных на шину питания, подключение которых чаще всего необходимо, так как тип выходов данных микросхем ПЗУ — это ОК или 3С.

В результате подобного объединения микросхем ПЗУ может увеличиться время выборки адреса полученного единого ПЗУ. В данном случае

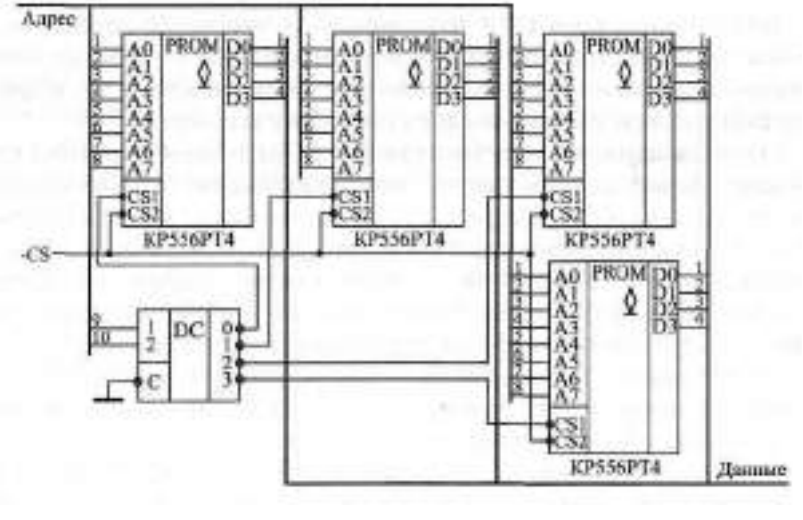


Рис. 6.4. Увеличение количества адресных разрядов ПЗУ с помощью дешифратора

(см. рис. 6.4) оно будет равно максимальной из двух величин: времени выборки адреса одной микросхемы и суммы двух задержек — задержки дешифратора и задержки выборки микросхемы ПЗУ.

Если надо объединить две микросхемы (то есть добавить всего один разряд адресной шины), можно обойтись без дешифратора, подавая на вход -CS одной микросхемы прямой дополнительный сигнал адреса, а на вход -CS другой микросхемы — этот же сигнал с инверсией. Применение дешифратора 3-8 позволяет объединить 8 микросхем ПЗУ (добавить три адресных разряда), а применение дешифратора 4-16 добавляет четыре адресных разряда, объединяя 16 микросхем ПЗУ.

Часто возникает также задача увеличения количества разрядов данных. Для этого необходимо всего лишь объединить одноименные адресные входы нужного количества микросхем ПЗУ; выходы же данных ПЗУ не объединяются, а образуют код с большим числом разрядов. Например, при объединении таким образом двух микросхем с организацией 8Kx8 можно получить ПЗУ с организацией 8Kx16.

6.1.1. ПЗУ как универсальная комбинационная микросхема

Одно из самых распространенных применений микросхем ПЗУ — замена ими сложных комбинационных схем. Такое решение позволяет существенно упростить проектируемое устройство и снизить количество используемых комбинационных микросхем, а иногда даже уменьшить потребляемый ток и увеличить быстродействие схемы.

Суть предлагаемого подхода сводится к следующему. Если рассматривать адресные входы микросхемы ПЗУ как входы комбинационной схемы, а разряды данных — как выходы этой комбинационной схемы, то можно сформировать *любую* требуемую таблицу истинности данной комбинационной схемы. Для этого всего лишь надо составить таблицу прошивки ПЗУ, соответствующую нужной таблице истинности. В этом случае не надо ни подбирать логические элементы, ни оптимизировать их соединения, ни думать о том, можно ли вообще построить заданную комбинационную схему из стандартных микросхем. Важно только, чтобы количество требуемых входов не превышало количества адресных разрядов ПЗУ, а количество требуемых выходов не превышало разрядности шины данных ПЗУ.

В качестве примера рассмотрим довольно сложную комбинационную схему (рис. 6.5), имеющую восемь входов и четыре выхода. Функция схемы сводится к следующему. Прежде всего она распознает два различных 5-разрядных входных кода (11001 и 10011) в случае, когда на входе разрешения "-Разр." присутствует нулевой сигнал, а при приходе сигналов "-Строб 1" и "Строб 2" схема выдает на выход отрицательные импуль-

сы. Причем первый выходной сигнал вырабатывается в случае, когда входной код равен 11001 и пришел сигнал "-Строб 1", второй выходной сигнал — при том же коде, но по входному сигналу '-Строб 2". Третий и четвертый выходной сигналы вырабатываются при входном коде 10011 и при приходе соответственно управляющих сигналов "-Строб 1" и "-Строб 2". То есть логика работы довольно сложная и разнообразных логических элементов требуется немало.

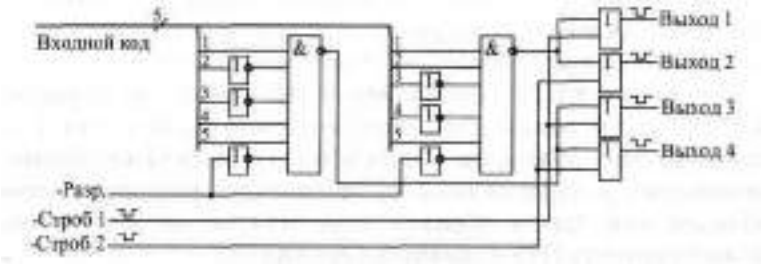


Рис. 6.5. Пример комбинационной схемы, заменяемой ПЗУ

Но всю эту схему можно заменить всего лишь одной микросхемой ПЗУ, например, типа РТ4, имеющей 8 адресных входов и 4 выхода данных (рис. 6.6). При этом пять разрядов входного кода подаются на младшие разряды адреса ПЗУ (АО... А4), входной сигнал "Разр." — на адресный вход А5, сигнал "-Строб 1" - на вход А6, сигнал "-Строб 2" - на вход А7. Младший разряд данных памяти DO используется для первого выходного сигнала, D1 — для второго выходного сигнала, D2 — для третьего выходного сигнала, D3 — для четвертого выходного сигнала. Микросхема ПЗУ всегда выбрана (управляющие сигналы -CS1 и -CS2 - нулевые). На выходах данных памяти включены резисторы, так как тип выходов микросхемы РТ4 - ОК.

Составим карту прошивки ПЗУ. Активные выходные сигналы - нулевые, а пассивные — единичные. Значит, в большинстве ячеек ПЗУ будут за-

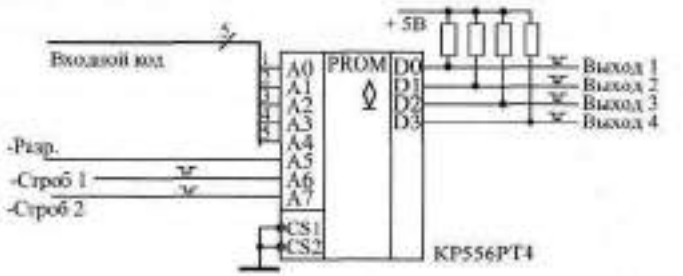


Рис. 6.6. Включение ПЗУ для замены комбинационной схемы, показанной на рис. 6.5

писаны коды F (все выходные сигналы пассивны). Активному (нулевому) первому выходному сигналу при пассивных остальных будет соответствовать двоичный код данных 1110 (16-ричный код — E), активному второму выходному сигналу будет соответствовать двоичный код 1101 (16-ричный — D), активному третьему выходному сигналу — двоичный код 1011 (или B), активному четвертому выходному сигналу — двоичный код 0111 (или 7). То есть только содержимое четырех ячеек памяти будет отличаться от F.

Например, код E будет записан в ячейку с таким адресом, пять младших разрядов которого (A0 ... A4) равны селектируемому входному коду 11001, разряд A5 равен нулю (сигнал "Разр." активен), разряд A6 равен нулю (сигнал "Строб 1" активен), а разряд A7 равен единице (сигнал "Строб 2" пассивен). Таким образом, получаем двоичный код адреса 10011001 (или в 16-ричном коде 99). Точно так же код D будет записан в ячейку с адресом 01011001 (то есть 16-ричное 59), код B — в ячейку с адресом 10010011 (то есть 93), а код 7 — в ячейку с адресом 01010011 (то есть 53). Получившаяся карта прошивки ПЗУ приведена в табл. 6.2. Она полностью совпадает с таблицей истинности заменяемой комбинационной схемы.

Таблица 6.2. Карта прошивки ПЗУ для замены комбинационной схемы

Адрес	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
10	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
20	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
30	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
40	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
50	F		F	7	F	F	F	F	F	D	F	F	F	F	F	F
60	F	F	F	F	F	F		F	F	F		F		F	F	F
70	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
80	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
90	F	F	F	B	F	F	F	F	F	E	F	F	F	F	F	F
A0	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
B0	F	F		F	F	F	F	F	F	F	F	F	F	F		F
C0	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
D0	F	F	F	F	F	F	F	F	F	F	F		F	F	F	F
E0	F	F	F	F	F			F	F	F	F	F	F	F	F	F
F0	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

Может показаться, что такое использование микросхемы ПЗУ чересчур расточительно, избыточно, но это не так. Гораздо важнее, что схема сильно упрощается. Если же взять комбинационную схему с более сложной таблицей истинности, то возможности ПЗУ будут использованы полнее. К тому же большое достоинство такого решения состоит в том, что при необходимости изменения логики работы комбинационной схемы потребуется всего лишь перепрошивка ПЗУ, а не проектирование новой схемы из логических элементов. Задержка ПЗУ при замене комбинационной схемы любой сложности остается одной и той же, она равна задержке выборки адреса микросхемы ПЗУ. При сложной заменяемой комбинационной схеме ПЗУ может оказаться даже быстрее.

Однако использование ПЗУ для замены комбинационных схем имеет и довольно серьезные недостатки. Дело в том, что микросхемы ПЗУ еще больше, чем комбинационные микросхемы, чувствительны к моменту изменения входных сигналов (адресных разрядов). На выходах данных микросхем ПЗУ при любом изменении входного кода адреса могут появляться короткие паразитные импульсы. Поэтому лучше всего использовать ПЗУ для замены таких комбинационных схем, которые работают в статическом режиме и в которых короткие импульсы не имеют значения.

Можно также применять методы синхронизации выходных сигналов ПЗУ (рис. 6.7) с помощью управляющих сигналов выбора микросхемы CS *(а)* или же с помощью выходных триггеров и регистров *(б)*. Суть синхронизации состоит в том, что выходные сигналы ПЗУ надо разрешать или фиксировать с помощью синхросигнала только тогда, когда все переходные процессы внутри микросхемы, вызванные сменой кода адреса, уже закончились и паразитные импульсы на выходах гарантированно отсутствуют.

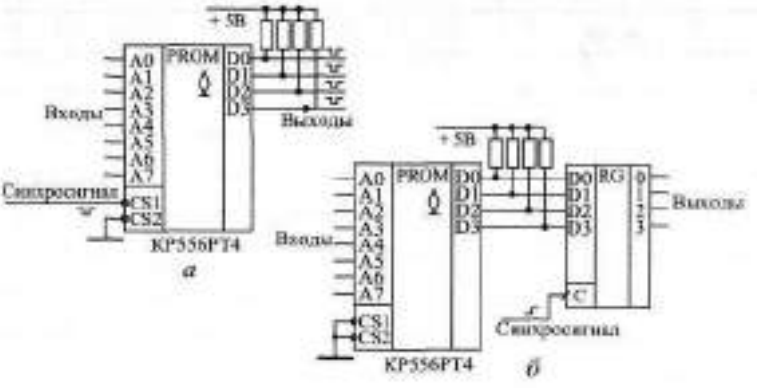


Рис. 6.7. Методы синхронизации выходных сигналов ПЗУ с помощью сигнала CS *(а)* и выходного регистра *(б)*

Микросхемы ПЗУ могут заменять собой любые комбинационные микросхемы: дешифраторы, шифраторы, компараторы кодов, сумматоры, мультиплексоры, преобразователи кодов и т. д. Однако при подобной замене всегда стоит подумать, не лучше ли использовать уже готовые микросхемы, чем изготавливать новые (программировать ПЗУ). Микросхемы ПЗУ могут оказаться медленнее стандартных комбинационных микросхем и потреблять больший ток питания. К тому же они могут требовать выходных резисторов, если микросхемы имеют выходы 0 К или при использовании входов CS у микросхем с выходами 3С. Другое дело, когда ПЗУ выполняет функцию, отличающуюся от функции стандартной комбинационной микросхемы. Простейший пример — дешифратор с положительными, а не с отрицательными (как в стандартных сериях) активными выходными сигналами.

В общем случае ПЗУ можно рассматривать как преобразователь входного кода (кода адреса) в выходной код (код данных) по произвольному закону, задаваемому разработчиком. Это позволяет не только преобразовывать друг в друга различные стандартные коды, но и выполнять множество других функций, например, использовать ПЗУ как простейший табличный вычислитель. Для этого нужно на адресные разряды ПЗУ подать код входного числа (аргумента), а на выходах разрядов данных получить код выходного числа (функции). Такой табличный вычислитель имеет очень высокое быстродействие по сравнению с другими типами вычислителей (время вычисления функции равно задержке выборки адреса ПЗУ).

В качестве простейшего примера рассмотрим вычислитель для возведения в квадрат 4-разрядного двоичного числа (рис. 6.8). Вычислитель выполнен на микросхеме ПЗУ типа РЕ3, у которого использованы четыре разряда адреса и восемь разрядов данных. Он позволяет получать двоичные коды квадратов любых чисел в диапазоне от 0 (или в двоичном коде 0000, в 16-ричном коде 0) до 15 (или в двоичном коде 1111, в 16-ричном коде F), которые принимают значения от 0 (или в двоичном коде 00000000, в 16-ричном - 00) до 225 (или в двоичном коде 11100001, в 16-ричном — E1).

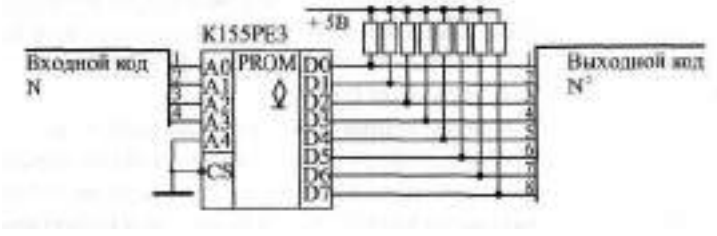


Рис. 6.8. Вычислитель квадратов входных чисел

Карта прошивки ПЗУ вычислителя квадратов (табл. 6.3) будет очень проста: код данных в каждой ячейке равен квадрату кода адреса этой ячейки. Используется всего 16 ячеек памяти, содержимое остальных 16 ячеек не имеет значения (что обозначено в таблице XX).

Таблица 6.3. Карта прошивки ПЗУ-вычислителя квадратов

Адрес	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	01	04	09	10	19	24	31	40	51	64	79	90	A9	C4	E1
10	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX

На такой же точно схеме (рис. 6.8) можно сделать и вычислитель квадратов в двоично-десятичном коде. В этом случае будет использовано всего лишь десять ячеек памяти с адресами от 0 до 9, а в ячейках будут записаны их квадраты от 0 до 81.

Недостаток любого табличного вычислителя на ПЗУ — это необходимость увеличения вдвое требуемого объема памяти при увеличении разрядности входного числа на единицу. Например, при 8-разрядных входных числах требуется ПЗУ с количеством ячеек 256, при 16-разрядных входных числах — ПЗУ с количеством ячеек 64K, а при 32-разрядных входных числах — ПЗУ с количеством ячеек 4Г. Такие большие объемы очень трудно реализовать на серийно выпускаемых микросхемах. Поэтому табличные вычислители на ПЗУ обычно строятся только для разрядности входных чисел не более 16.

Одно из наиболее распространенных применений ПЗУ как преобразователя кодов — это построение на их основе всевозможных индикаторов, отображающих на экране буквы и цифры. ПЗУ в данном случае переводит код (номер) буквы или цифры в ее изображение. Конечно, в данном случае заменить ПЗУ комбинационной схемой совершенно невозможно, так как букв и цифр очень много, а их изображения очень разнообразны.

Простейший пример данного применения ПЗУ — это управление знаковым семисегментным индикатором, знакомым всем по калькуляторам, кассовым аппаратам, электронным часам, весам и т. д. В семисегментных индикаторах изображение всех цифр от 0 до 9 строится всего из семи сегментов (отрезков линий) (рис. 6.9).

Чтобы отобразить в виде цифры 4-разрядный двоичный код, надо этот код преобразовать в 7-разрядный код, каждому разряду которого будет соответствовать один сегмент индикатора. То есть коду 0000 должно соответствовать изображение нуля (6 сегментов, расположенных по периметру), а коду 0001 — изображение единицы (два правых вертикальных сег-

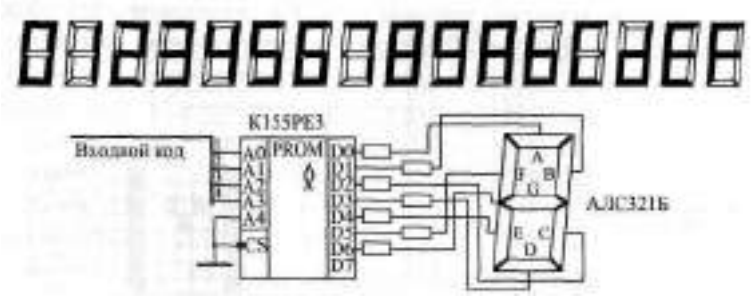


Рис. 6.9. Дешифратор знакового семисегментного индикатора на ПЗУ

мента). Для повышения универсальности индикатора удобно дополнить десять цифр еще и шестью буквами, использующимися в 16-ричном коде (A, B, C, D, E, F). Семь сегментов индикатора позволяют сделать и это, правда, изображения букв получаются не слишком качественными.

ПЗУ типа РЕ3, используемое в качестве дешифратора индикатора, имеет 4 входа и 7 выходов (старший разряд адреса и старший разряд данных не используются). Карта прошивки ПЗУ приведена в табл. 6.4. Нулевой сигнал на каждом из выходов данных ПЗУ зажигает соответствующий ему сегмент.

Таблица 6.4. Карта прошивки ПЗУ для дешифратора знакового индикатора

Адрес	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	40	79	24	30	19	12	02	78	00	10	08	03	46	21	06	0E
10	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX

ПЗУ позволяют также формировать и более сложные изображения букв и цифр — матричные. Такие изображения используются, например, в табло типа "бегущая строка", на экранах мониторов, в больших рекламных табло. Каждая буква, цифра, другой знак располагается в данном случае на прямоугольной матрице, называемой знакоместом и состоящей из нескольких строк и нескольких столбцов точечных элементов изображения, которые могут зажигаться независимо друг от друга. Чем больше строк и столбцов в знакоместе, тем более качественное изображение букв и цифр можно получить. Минимально возможный размер знакоместа — 5 столбцов на 7 строк, то есть всего 35 элементов изображения.

ПЗУ в данном случае содержит в себе информацию об изображениях всех возможных букв и цифр (обычно этот набор включает в себя 256 символов). Но выходной код ПЗУ имеет мало разрядов, поэтому

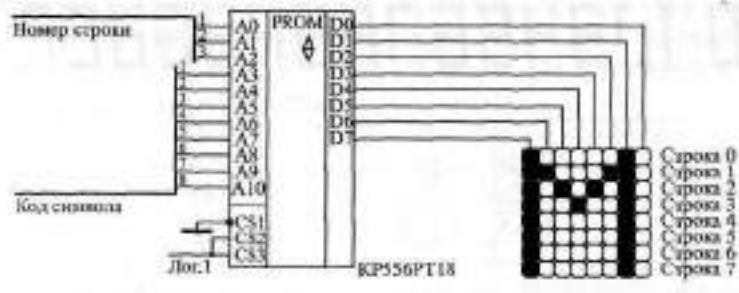


Рис. 6.10. Матричный знакогенератор на ПЗУ

каждый такой код, соответствующий одному адресу, представляет собой информацию об изображении не целого символа, а только одной его строки (или столбца). Информация о целом символе занимает в ПЗУ столько ячеек, сколько в изображении символа имеется столбцов (или строк). Пример матричного знакогенератора на ПЗУ приведен на рис. 6.10.

В данном случае используется знакоместная матрица из 8 строк и 8 столбцов. В каждую ячейку ПЗУ записывается код изображения одной из 8 строк одного из 256 символов. Изображение одного символа занимает 8 последовательно расположенных ячеек в ПЗУ. Для букв и цифр правый столбец знакоместа не используется, он служит для отделения знаков друг от друга, но для специальных символов (например, для графических) он может и использоваться. В случае матричного светодиодного индикатора перебор строк может осуществляться 3-разрядным счетчиком с дешифратором 3-8 на его выходе. В случае телевизионного монитора перебор строк осуществляется с помощью генератора вертикальной развертки изображения.

Составление карты прошивки такого ПЗУ непросто, оно обычно производится с помощью специальных программ на компьютере. Но принцип составления прост. Например, если активному (зажженному) элементу изображения соответствует единичный сигнал, то для нулевой строки символа "М", показанного на рисунке, в ПЗУ надо записать 10000010, для первой строки — код 11000110, для второй — код 10101010, для третьей - 10010010 и т. д.

6.1.2. ПЗУ в генераторах импульсных последовательностей

Следующее важнейшее применение ПЗУ — это построение генераторов сложных последовательностей цифровых импульсов. Такие генераторы широко используются в самых разных измерительных системах,

в устройствах автоматики, в телевизионных системах, в схемах управления линейными или матричными индикаторами и т. д.

Задача в данном случае ставится следующим образом. Необходимо сформировать последовательность из нескольких сигналов различной длительности, сдвинутых относительно друг друга на различные временные интервалы. Причем последовательность эта может быть как разовой (однократно начинающейся по внешнему сигналу), так и периодической, непрерывно повторяющейся.

Наиболее распространенная структура генератора последовательностей выходных сигналов на ПЗУ включает в себя тактовый генератор нужной частоты, счетчик с требуемым числом разрядов, ПЗУ и выходной регистр (рис. 6.11). Счетчик перебирает адреса ПЗУ, ПЗУ последовательно выдает на выходы данных все записанные в него коды. Выходной регистр, тактируемый тем же тактовым сигналом, что и счетчик, служит для предотвращения появления в выходных сигналах паразитных импульсов и для обеспечения одновременного переключения всех выходных сигналов (что особенно важно в случае, когда используются несколько параллельно включенных микросхем ПЗУ для увеличения разрядности шины данных).

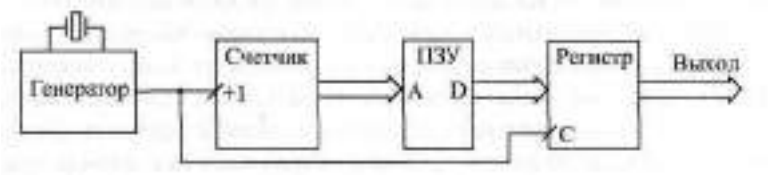


Рис. 6.11. Пример структуры генератора последовательностей сигналов на ПЗУ

Рассмотрим пример. Пусть необходимо непрерывно формировать периодическую последовательность из шести выходных сигналов в соответствии с временной диаграммой рис. 6.12. Получить такую последовательность можно, конечно, с помощью комбинационных схем, включенных на выходе счетчика, или с помощью множества одновибраторов, запускающих друг друга, но и то, и другое решение чересчур громоздко и сложно как в проектировании, так и в настройке. Применение же ПЗУ значительно упрощает задачу. Достаточно провести несложные расчеты и составить карту прошивки ПЗУ.

Расчеты сводятся к следующему.

Прежде всего определяем минимально возможную тактовую частоту (с целью минимизации требуемого объема ПЗУ). Для этого надо выделить максимальный временной интервал (дискрет времени), который укладывается целое число раз во все временные сдвиги, задержки, длительности

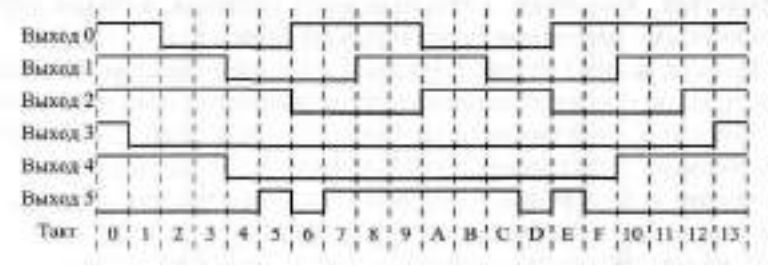


Рис. 6.12. Временная диаграмма формируемых выходных сигналов

требуемой диаграммы. В нашем случае этот дискрет равен одному делению по оси времени. Например, если длительность этого деления равна 250 нс, то и период тактового сигнала надо выбирать 250 нс, то есть тактовая частота будет равна 4 МГц. Можно, конечно, выбрать ее и кратной 4 МГц, например, 8 МГц, 12 МГц, но тогда потребуется вдвое или втрое больший объем ПЗУ. Если бы нам надо было формировать только три верхних сигнала (Выход 0, Выход 1, Выход 2), то период тактовой частоты можно было бы брать вдвое больше (в нашем примере — 500 нс), так как для этих сигналов все длительности кратны двум делениям.

Второй расчет сводится к определению количества ячеек и разрядности ПЗУ. Шесть выходных сигналов схемы требуют шести разрядов данных ПЗУ. Длительность последовательности равна 20 тактам (или 14 в 16-ричном коде), то есть не равна 2^n, поэтому счетчик придется сбрасывать в нуль через каждые 20 тактов, для чего потребуется еще один разряд данных ПЗУ. Итого потребуется 7 разрядов. А для перебора 20 тактов последовательности потребуется 5-разрядный счетчик, так как $2^4 = 16$ (недостаточно), а $2^5 = 32$ (достаточно). Значит, разрядность шины адреса ПЗУ также должна быть не менее пяти, то есть минимальные требования к организации ПЗУ — это 32x8, значит, подойдет микросхема ПЗУ типа РЕ3.

Наконец, третий расчет касается условий правильной работы схемы. Генератор последовательности будет работать правильно, если за период тактового сигнала успеют сработать счетчик и ПЗУ. То есть сумма задержки полного переключения счетчика и задержки выборки адреса ПЗУ не должна превышать периода тактового сигнала.

Таким образом, один из возможных вариантов схемы генератора последовательности импульсов (рис. 6.13) будет включать в себя тактовый генератор, пятиразрядный счетчик на основе ИЕ19, ПЗУ типа К155РЕ3 и 8-разрядный выходной регистр ИР27. Так как счетчик срабатывает по отрицательному фронту тактового сигнала, а регистр - по положительному фронту тактового сигнала, необходимо включить инвертор. На схеме для простоты не показаны резисторы на выходах данных типа 0К микросхемы ПЗУ.

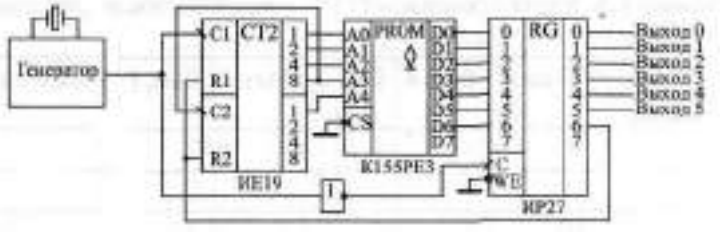


Рис. 6.13. Первый вариант схемы генератора последовательности сигналов на ПЗУ

Из схемы видно, что существует еще одно дополнительное условие правильной работы, связанное с использованием сигнала сброса. За период тактового сигнала должен успеть сработать регистр (записать в себя сигнал сброса с выхода ПЗУ), должен сброситься счетчик и должно выдать код ПЗУ. То есть сумма задержек регистра, сброса счетчика и выборки адреса ПЗУ не должна превышать периода тактового сигнала.

Теперь составим карту прошивки ПЗУ. Для этого сформируем таблицу значений всех семи выходных сигналов во всех двадцати рабочих тактах генератора последовательностей (табл. 6.5). Значения шести выходных сигналов (Выход 0. . . Выход 5) мы непосредственно берем из требуемой временной диаграммы (см. рис. 6.12). Хотя эти сигналы будут приходить на выход схемы с задержкой на один такт из-за наличия выходного регистра, но при периодической работе генератора последовательности это не имеет никакого значения.

Седьмой сигнал (Выход 6) используется в качестве сброса счетчика, он равен нулю от нулевого такта до 18 (в 16-ричном коде — 12) такта и равен единице в последнем девятнадцатом такте (в 16-ричном коде — 13). Этим единичным сигналом счетчик будет сбрасываться в нуль, то есть работа схемы после 19 такта будет возобновляться с нулевого такта.

Пусть нам требуется более высокое быстродействие схемы. Например, период тактового генератора должен быть равен 100 нс и даже меньше. В этом случае асинхронный счетчик типа ИЕ19 уже не подойдет из-за своего низкого быстродействия. Надо использовать 5-разрядный синхронный счетчик. Можно, конечно, включить две микросхемы 4-разрядных счетчиков типа ИЕ17, но можно обойтись и одним 4-разрядным счетчиком, если задействовать оставшийся свободным выход данных ПЗУ и оставшийся свободным разряд регистра (рис. 6.14). При этом без всякого ущерба для быстродействия можно использовать синхронный счетчик с асинхронным переносом ИЕ7, так как нужна всего одна микросхема счетчика

Таблица 6.5. Карта прошивки ПЗУ генератора последовательности сигналов

Такт (адрес)	Вых.6	Вых.5	Вых.4	Вых.3	Вых.2	Вых.1	Вых.О	Код (данные)
0	0	0	1	1	1	1	1	1F
1	0	0	1	0	1	1	1	17
2	0	0	1	0	1	1	0	16
3	0	0	1	0	1	.1	0	16
4	0	0	0	0	1	0	0	04
5	0	1	0	0	1	0	0	24
6	0	0	0	0	0	0	1	01
7	0	1	0	0	0	0	1	21
8	0	1	0	0	0	1	1	23
9	0	1	0	0	0	1	1	23
A	0	1	0	0	1	1	0	26
B	0	1	0	0	1	1	0	26
C	0	1	0	0	1	0	0	24
D	0	0	0	0	1	0	0	04
E	0	1	0	0	0	0	1	21
	0	0	0	0	0	0	1	01
10	0	0	1	0	0	1	1	13
11	0	0	1	0	0	1	1	13
12	0	0	1	0	1	1	1	17
13	1	0	1	1	1	1	1	5F

Карта прошивки ПЗУ генератора последовательностей остается той же самой, что и в предыдущем случае, но добавляется один (седьмой) разряд шины данных ПЗУ. Этот разряд должен быть равен нулю в пятнадцати тактах с нулевого до 14 (в 16-ричном коде — Е), в четырех тактах с 15 (F в 16-ричном коде) по 18 (12 в 16-ричном коде) он должен быть равен единице (код данных увеличится на 80), а в последнем 19 такте (13 в 16-ричном коде) он снова должен быть равен нулю. В результате адреса ПЗУ будут перебираться от 0 до 15 (старший разряд адреса равен нулю), затем от 16 до 19 (старший разряд адреса станет равен единице, а счетчик после переполнения считает с нуля), а затем снова от нулевого адреса (счетчик сбросится сигналом "Выход 6", а старший разряд адреса памяти станет ну-

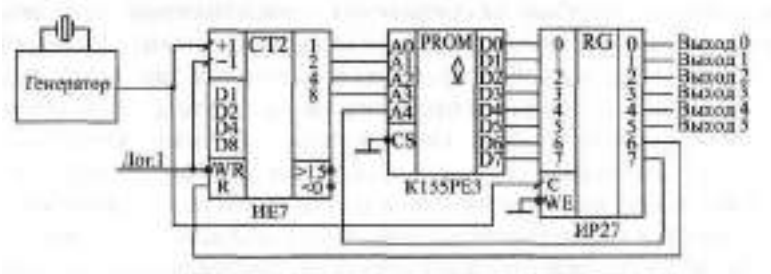


Рис. 6.14. Второй вариант схемы генератора последовательности с 4-разрядным синхронным счетчиком ИЕ7

левым). Получающаяся в результате карта прошивки ПЗУ приведена в табл. 6.6.

Таблица 6.6. Карта прошивки ПЗУ для варианта схемы генератора последовательности импульсов

Адрес	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	1F	17	16	16	04	24	01	21	23	23	26	26	24	04	21	61
10	93	93	97	5F	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX

Конечно, рассмотренный пример довольно прост, при использовании ПЗУ большего объема генерируемые последовательности сигналов могут быть гораздо сложнее рассмотренной, и выходных сигналов может быть больше, но суть остается прежней: ПЗУ выдает любые последовательности выходных сигналов минимальными средствами. Причем генерируемые последовательности можно довольно просто изменять, заменяя ПЗУ на микросхемы с другой картой прошивки.

6.1.3. Микропрограммные автоматы на ПЗУ

Микропрограммные автоматы представляют собой следующий шаг по пути усложнения интеллекта цифровых схем. На основе микропрограммных автоматов можно строить устройства, которые работают по довольно сложным алгоритмам, выполняют различные функции, определяемые входными сигналами, выдают сложные последовательности выходных сигналов. При этом алгоритм работы микропрограммного автомата может быть легко изменен заменой прошивки ПЗУ.

В отличие от рассматривавшихся ранее устройств на "жесткой" логике, принцип работы которых однозначно определяется используемыми

элементами и способом их соединения, микропрограммные автоматы с помощью одной и той же схемы могут выполнять самые разные функции. То есть они гораздо более гибкие, чем схемы на "жесткой" логике. К тому же проектировать микропрограммные автоматы с точки зрения схемотехники довольно просто. Недостатком любого микропрограммного автомата по сравнению со схемами на "жесткой" логике является меньшее предельное быстродействие и необходимость составления карты прошивки ПЗУ с микропрограммами, часто довольно сложными.

Наиболее распространенная структура микропрограммного автомата (рис. 6.15) включает в себя всего лишь три элемента: ПЗУ, регистр, срабатывающий по фронту, и тактовый генератор.

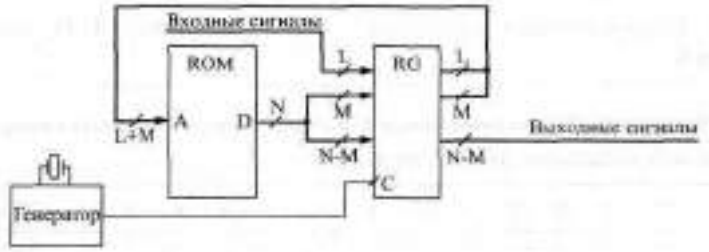


Рис. 6.15. Структура микропрограммного автомата

ПЗУ имеет (L+M) адресных разрядов и N разрядов данных. Регистр применяется с количеством разрядов (N + L). Разряды данных ПЗУ записываются в регистр по положительному фронту тактового сигнала с генератора. Часть этих разрядов (M) используется для образования адреса ПЗУ, другая часть (N-M) служит для формирования выходных сигналов. Входные сигналы (L) поступают на входы регистра и используются совместно с частью выходных разрядов ПЗУ для получения адреса ПЗУ.

Схема работает следующим образом. В каждом такте ПЗУ выдает код данных, тем самым определяя не только состояние выходных сигналов схемы, но и адрес ПЗУ, который установится в следующем такте (после следующего положительного фронта тактового сигнала). На этот следующий адрес влияют также и входные сигналы. То есть в отличие от формирователя последовательности сигналов, рассмотренного в предыдущем разделе, в данном случае адреса могут перебираться не только последовательно (с помощью счетчика), но и в произвольном порядке, который определяется прошивкой ПЗУ, называемой микропрограммой.

Условием правильной работы схемы будет следующее. За один период тактового сигнала должны успеть сработать регистр и ПЗУ. Иначе говоря, сумма задержки регистра и задержки выборки адреса ПЗУ не должна превышать периода тактового сигнала. Отметим также, что входные

сигналы микропрограммного автомата нельзя подавать непосредственно на адресные входы ПЗУ (без регистра), так как их асинхронное (по отношению к тактовому сигналу) изменение может вызвать переходный процесс на выходах данных ПЗУ именно в тот момент, когда выходные сигналы ПЗУ записываются в регистр. В результате в регистр может записаться неверная информация, что нарушит работу всей схемы. Регистр же синхронизирует изменения входных сигналов с тактовым сигналом, в результате чего все разряды кода адреса ПЗУ меняется одновременно — по положительному фронту тактового сигнала. Регистр должен обязательно срабатывать по фронту, использование регистра-защелки не допускается, так как он может вызвать лавинообразный переходный процесс.

Возможности такой простой схемы оказываются очень большими. Например, микропрограммный автомат может выдавать последовательности выходных сигналов в ответ на определенное изменение входных сигналов. Он может также временно остановить выдачу выходных сигналов до прихода входных сигналов. Он может анализировать длительность входного сигнала и в зависимости от нее выдавать те или иные выходные сигналы. Он может также и многое другое.

Сформулируем несколько элементарных функций, из которых могут складываться алгоритмы работы микропрограммного автомата (рис. 6.16):

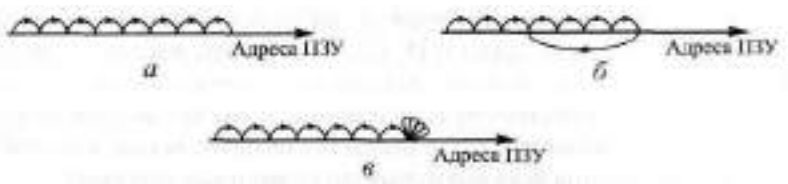


Рис. 6.16. Элементарные функции микропрограммного автомата: последовательный перебор *(а)*, циклическое повторение *(б)* и остановка *(в)*

а) Последовательный перебор адресов ПЗУ (например, для выдачи последовательности выходных сигналов).

б) Периодическое повторение последовательности адресов ПЗУ (например, для повторения последовательности выходных сигналов).

в) Остановка в каком-то адресе ПЗУ (например, для ожидания изменения входного сигнала).

г) Временное отключение реакции на входные сигналы (например, для того, чтобы закончить отработку реакции на предыдущее изменение входных сигналов). Эта функция на рисунке не показана.

В качестве примера рассмотрим выполнение некоторых элементарных функций микропрограммным автоматом, показанным на рис. 6.17.

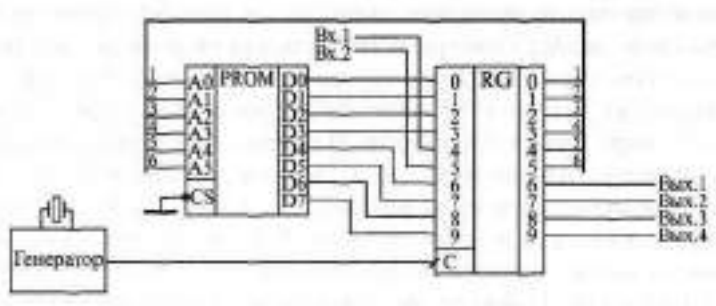


Рис. 6.17. Пример практической схемы микропрограммного автомата на ПЗУ

ПЗУ имеет организацию 64x8 (это могут быть две микросхемы РЕЗ или одна микросхема РТ18), количество разрядов регистра равно десяти (это могут быть две микросхемы ИР27). Схема имеет два входных сигнала и четыре выходных сигнала. Такая организация микропрограммного автомата хороша тем, что его микропрограмма (то есть карта прошивки ПЗУ) очень наглядна, легко составляется и читается.

Выходные разряды данных ПЗУ делятся на две группы по 4 сигнала: младшие идут на образование следующего адреса ПЗУ, старшие образуют четыре выходных сигнала. Входные сигналы поступают (через регистр) на два старших разряда адреса ПЗУ. Если мы изобразим карту прошивки ПЗУ в привычном для нас виде таблицы из четырех строк и 16 столбцов (табл. 6.7), то в этой таблице будут наглядно видны как состояние каждого из внутренних сигналов, так и реакция схемы на любой входной сигнал, а также состояния всех выходных сигналов в каждом такте.

Таблица 6.7. Карта прошивки ПЗУ 64x8 для микропрограммного автомата

Адрес	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
10	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
20	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
30	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Прежде всего, легко заметить, что выбор той или иной строки таблицы производится старшими разрядами кода адреса ПЗУ, то есть входными сигналами микропрограммного автомата. Таким образом, любое изменение входных сигналов приводит к переходу в карте прошивки

на другую строку. Например, строка 00 будет соответствовать нулевым входным сигналам микропрограммного автомата, а строка 30 — единичным входным сигналам.

Теперь посмотрим на структуру 8-разрядного кода данных ПЗУ. Младшие четыре разряда этого кода (правый, младший знак 16-ричного кода в таблице) соответствуют четырем младшим разрядам кода адреса ПЗУ. Старшие четыре разряда кода данных ПЗУ (левый, старший знак 16-ричного кода в таблице) соответствуют четырем выходным сигналам микропрограммного автомата. То есть непосредственно по 16-ричному коду данных из таблицы можно сказать, во-первых, каким будет следующий адрес ПЗУ, и во-вторых, какими будут выходные сигналы автомата в следующем такте.

Рассмотрим пример микропрограммы (табл. 6.8), реализующей некоторые элементарные функции.

Таблица 6.8. Пример микропрограммы для схемы на рис. 6.17

Адрес	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	11	22	33	44	55	66	77	88	99	AA	BB	CC	DD	EE	FF	00
10	11	22	33	44	55	66	77	88	99	55	BB	CC	DD	EE	FF	00
20	10	21	32	43	54	65	76	87	98	A9	BA	CB	DC	ED	FE	0F
30	10	20	30	40	50	60	70	80	90	AO	BO	CO	DO	E0	F0	00

Верхняя (нулевая) строка таблицы демонстрирует последовательный перебор адресов памяти при нулевых входных сигналах. Пусть, например, автомат находится в адресе 00. В ячейке с адресом 00 указан следующий адрес 1 (младший знак 16-ричного кода 11), то есть в следующем такте автомат перейдет в адрес 01 (считаем, что входные сигналы остаются нулевыми). Из адреса 01 автомат перейдет в адрес 02, так как в ячейке с адресом 01 указан следующий адрес 2. Точно так же из адреса 02 автомат перейдет в адрес 03 и так далее до адреса 0F, в котором указан следующий адрес 0, то есть в следующем такте автомат снова вернется в адрес 00. Затем цикл последовательного прохождения адресов первой строки повторится (если, конечно, входные сигналы останутся нулевыми). Четыре выходных сигнала автомата в данном случае повторяют код следующего адреса, то есть, подобно 4-разрядному двоичному счетчику, выдают постепенно нарастающий код.

Вторая сверху (первая) строка таблицы демонстрирует циклическое повторение группы тактов. Если, например, работа начинается с адреса 10, то микропрограммный автомат последовательно будет перебирать адреса 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, а затем вернется в адрес

15 и будет постоянно повторять группу адресов 15, 16, 17, 18, 19. В данном случае мы считаем, что входной сигнал "Вх. 1" постоянно равен единице, а входной сигнал "Вх. 2" постоянно равен нулю, что и соответствует второй сверху строке таблицы.

Посмотрим теперь, что будет делать автомат, если входной сигнал "Вх. 2" постоянно равен нулю, а входной сигнал "Вх. 1" меняет свое состояние с нуля на единицу и обратно. Такое переключение входных сигналов приводит к переходу между нулевой и первой строками таблицы с микропрограммой. Допустим, автомат начинает работу с адреса 00 (сигнал "Вх. 1" равен нулю). Идет последовательный перебор адресов нулевой строки. Даже если в течение первых девяти тактов сигнал "Вх. 1" будет меняться, на выходных сигналах автомата это никак не отразится, так как коды первых девяти адресов в нулевой и первой строках полностью совпадают. Поэтому можно сказать, что в этих первых девяти тактах мы отключили реакцию нашего автомата на входной сигнал "Вх. 1" за счет дублирования кодов.

Допустим теперь, что изначально нулевой входной сигнал "Вх. 1" стал равен единице в адресе 0A и далее не меняется. Это приведет к тому, что вместо адреса 0B автомат перейдет в адрес 1B. Затем он пройдет адреса 1C. .. 1F, перейдет в адрес 10, дойдет до адреса 19 и начнет повторять цикл 15. . .19. Если же, например, в адресе 18 сигнал "Вх. 1" снова станет равен нулю, то вместо адреса 19 автомат попадет в адрес 09 и далее будет выполнять микропрограмму нулевой строки.

Третья сверху (вторая) строка таблицы, которая соответствует входным сигналам Вх. 1 = 0, Вх.2 = 1, показывает пример остановки автомата в каждом адресе и ожидание прихода входного сигнала. Например, автомат находится в адресе 23. Следующим адресом в коде данной ячейки указан 3. Значит, если входные сигналы остаются неизменными, то автомат перейдет опять же в адрес 23 и будет оставаться в нем постоянно. Но если входной сигнал "Вх. 2" станет равным нулю, автомат перейдет в адрес 03 и начнет выполнять микропрограмму нулевой строки. То есть переключение входного сигнала "Вх.2" из нуля в единицу при нулевом уровне "Вх. 1" останавливает выполнение микропрограммы нулевой строки до обратного перехода входного сигнала "Вх. 2" в нуль.

Наконец нижняя (третья) строка таблицы демонстрирует переход из любого адреса строки в нулевой адрес этой же строки. Пусть, например, выполняется микропрограмма нулевой строки ("Вх. 1" и "Вх. 2" — нулевые) и в адресе 06 оба входных сигнала переходят в единицу. Автомат попадает в адрес 37, а из него — в адрес 30, где и остается постоянно до изменения входных сигналов. Если затем оба входных сигнала снова станут нулевыми, то автомат перейдет в адрес 00 и начнет снова выполнять микропрограмму нулевой строки таблицы.

Иногда бывает необходимо перевести автомат в какой-то определенный адрес. Ведь при включении питания в регистре может оказаться произвольный код. Для такой инициализации автомата можно, например, использовать регистр, имеющий вход сброса R, на который подается внешний сигнал, и тогда по этому сигналу автомат попадет в нулевой адрес. Но можно пойти и другим путем: составить микропрограмму таким образом, что автомат при включении питания сам перейдет за несколько тактов в нужный адрес независимо от того, в какой адрес он попал при включении питания.

Рассмотрим теперь пример проектирования простейшего микропрограммного автомата.

Пусть нам необходимо формировать с помощью микропрограммного автомата последовательность из трех выходных сигналов в ответ на положительный фронт одного входного сигнала (рис. 6.18).

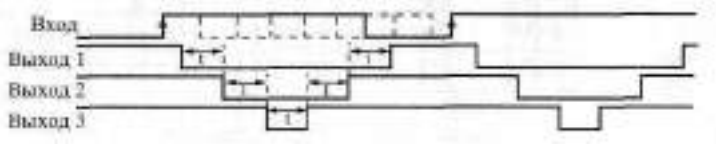


Рис. 6.18. Диаграмма работы проектируемого микропрограммного автомата

Второй выходной сигнал "вложен" в первый, а третий "вложен" во второй. Причем до тех пор, пока данная последовательность не закончится, входной сигнал может произвольно менять свое состояние, на работе схемы это никак не должно сказываться. А после окончания данной последовательности микропрограммный автомат снова должен ждать положительного фронта входного сигнала и начинать по нему выработку новой последовательности.

Все временные сдвиги в выходной последовательности (t) примем для простоты одинаковыми и равными 1 мкс. Это значительно облегчает выбор тактовой частоты микропрограммного автомата, она должна быть равной 1 МГц (период 1 мкс). В этом случае полная длина генерируемой последовательности составит всего 6 тактов (один такт соответствует переходу всех выходных сигналов в единицу).

Для формирования 6-тактовой последовательности необходимо не менее 3 адресных разрядов ПЗУ (так как $2^3 = 4$, а $2^3 = 8$). Еще один разряд адреса ПЗУ потребуется для входного сигнала схемы. Итого необходимо 4 адресных разряда ПЗУ.

Для формирования трех выходных сигналов требуется три разряда данных ПЗУ. Еще три разряда данных ПЗУ нужно для задания следующе-

го адреса при отработке выходной последовательности. Итого необходимо шесть разрядов данных ПЗУ.

Посчитаем требуемую разрядность выходного регистра. ПЗУ выдает шесть разрядов данных. Имеется один входной сигнал. Итого регистр должен фиксировать 7 сигналов, значит, он должен быть 7-разрядным.

Итак, все расчеты закончены. Полученная схема микропрограммного автомата приведена на рис. 6.19. Она включает в себя тактовый генератор с частотой 1 МГц, микросхему ПЗУ типа РЕ3, у которой не используется один разряд адреса и два разряда данных, а также 8-разрядный регистр ИР27, у которого не используется один разряд. Для простоты на рисунке не показаны резисторы на выходах данных типа ОК микросхемы ПЗУ.

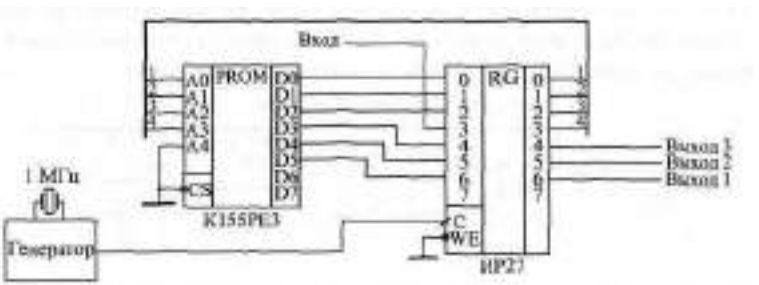


Рис. 6.19. Спроектированный микропрограммный автомат

Теперь необходимо составить микропрограмму для спроектированной схемы. Она должна включать в себя ожидание положительного фронта входного сигнала, отработку последовательности выходных сигналов с временным отключением входного сигнала, ожидание нулевого уровня входного сигнала и переход на новое ожидание положительного фронта входного сигнала. Полученная микропрограмма с необходимыми комментариями представлена в табл. 6.9.

Отключение реакции на входной сигнал достигается в микропрограмме за счет дублирования участка отработки выходной последовательности при входном сигнале, равном нулю, и при входном сигнале, равном единице. Если в конце выходной последовательности входной сигнал равен единице, то схема ожидает сначала перехода входного сигнала в нуль (отрицательный фронт), а затем уже ждет положительного фронта входного сигнала. Если в конце последовательности входной сигнал равен нулю, то схема сразу же переходит на ожидание положительного фронта входного сигнала.

Все неиспользуемые микропрограммным автоматом ячейки заполнены командами перехода в начальное состояние схемы с пассивными (единичными) выходными сигналами. Таким решением обеспечивается

Таблица 6.9. Микропрограмма для спроектированного автомата

Адрес ПЗУ					Данные ПЗУ							Комментарий
Код	Вход	Данные			Вых.1	Вых.2	Вых.3	Сл. адрес			Код	
0	0	0	0	0	1	1	1	0	0	0	38	Ожидание полож. фронта
8	1	0	0	0	0	1	1	0	0	1	19	Пришел положительный фронт входного сигнала. Отработка последовательности. Входной сигнал равен единице
9	1	0	0	1	0	0	1	0	1	0	0A	
A	1	0	1	0	0	0	0	0	1	1	03	
B	1	0	1	1	0	0	1	1	0	0	0C	
C	1	1	0	0	0	1	1	1	0	1	1D	
D	1	1	0	1	1	1	1	1	0	1	3D	Ожидание Вх.= 0
1	0	0	0	1	0	0	1	0	1	0	0A	Вх. стал равен нулю
2	0	0	1	0	0	0	0	0	1	1	03	до окончания последовательности
3	0	0	1	1	0	0	1	1	0	0	0C	
4	0	1	0	0	0	1	1	0	0	0	18	Переход на ожидание
5	0	1	0	1	1	1	1	0	0	0	38	Переход на ожидание
6	0	1	1	0	1	1	1	0	0	0	38	Не используемые ячейки
7	0	1	1	1	1	1	1	0	0	0	38	
E	1	1	1	0	1	1	1	0	0	0	38	
F	1	1	1	1	1	1	1	0	0	0	38	

правильное начало работы микропрограммного автомата при любом начальном адресе (при любом начальном коде в регистре). Начальный сброс автомата не используется.

Последний микропрограммный автомат, который мы рассмотрим в данном разделе, предназначен для дешифрации (декодирования) последовательного кода Манчестер-П, применяющегося для последовательной передачи данных на большие расстояния, в частности, в локальных сетях. Этот код уже упоминался в главе 4 (см. рис. 4.16 и 4.17). Там был рассмотрен кодировщик кода Манчестер-П.

Дешифрация (декодирование) этого кода гораздо сложнее кодирования (рис. 6.20). Она требует выделения "правильных", информационных фронтов в середине битовых интервалов (помечены на рисунке кружками) и отсечение "неправильных" фронтов между битовыми ин-

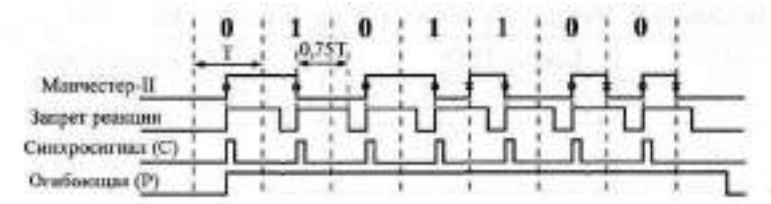


Рис. 6.20. Декодирование кода Манчестер-II

тервалами (помечены на рисунке крестиками). Для этого нужно после первого (информационного) фронта в течение временного интервала 0,75Т не реагировать на приходящие фронты входного сигнала, а затем снова обрабатывать любой приходящий фронт, снова выдерживать интервал 0,75Т и т. д. При приходе "правильных", информационных фронтов (в середине битовых интервалов) необходимо формировать выходные синхросигналы, по которым фиксируется (в регистре) информация из сигнала в коде Манчестер-II.

Таким образом, алгоритм декодирования довольно сложен. Его можно, конечно, выполнить с помощью одновибраторов и триггеров. Но можно воспользоваться и микропрограммным автоматом, который, в отличие от одновибратора, не требует настройки, не чувствителен к помехам и тактируется сигналом кварцевого генератора.

Схема такого микропрограммного автомата (рис. 6.21) очень проста, она включает в себя ПЗУ типа РЕ3, регистр ИР27 и тактовый генератор с частотой, в 8 раз превышающей частоту прихода битов в коде Манчестер-П. Например, при длительности битового интервала 1 мкс (скорость передачи информации 1 Мбит/с) частота генератора должна быть равной 8 МГц. Схема практически не отличается от схемы, рассмотренной в предыдущем примере, хотя выполняемая ею функция гораздо сложнее.

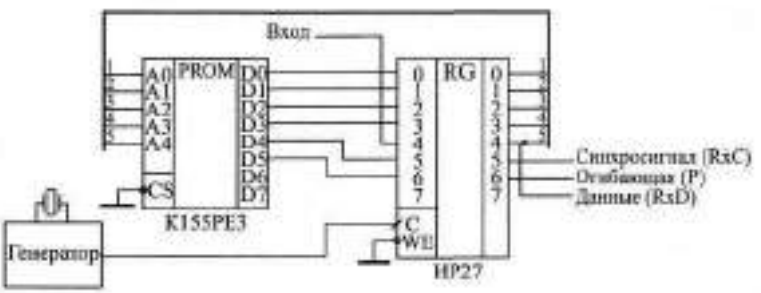


Рис. 6.21. Микропрограммный автомат для декодирования кода Манчестер-П

Таблица 6.10. Микропрограмма декодирования кода Манчестер-II (сигнал RxC обозначен в таблице С)

Адрес ППЗУ					Данные ПЗУ						Комментарий
Вх	Адрес				С	р	След. адрес				
0	0	0	0	0	0	1	0	0	0	1	Задержка и ожидание положительного
0	0	0	0	1	0	1	0	0	1	0	фронта входного сигнала
0	0	0	1	0	0	1	0	0	1	1	
1	0	0	0	0	1	1	0	1	0	1	
1	0	0	0	1	1	1	0	1	0	1	Снятие Р и ожидание входного сигнала
1	0	0	0	0	1	1	0	1	0	1	Выставление RxC и переход
1	0	0	0	1	1	1	0	1	0	1	на обработку положительного
1	0	0	1	0	1	1	0	1	0	1	входного сигнала
1	0	0	1	1	1	1	0	1	0	1	
1	0	1	0	0	1	1	0	1	0	1	
0	0	1	0	1	1	1	0	1	1	0	Снятие RxC и задержка
0	0	1	1	0	0	1	0	1	1	1	с отключением входа
0	0	1	1	1	0	1	1	0	0	0	
0	1	0	0	0	0	1	1	0	0	1	
0	1	0	0	1	0	1	0	0	0	0	На ожидание положительного фронта
1	0	1	0	1	1	1	0	1	1	0	Снятие RxC и задержка
1	0	1	1	0	0	1	0	1	1	1	с отключением входа
1	0	1	1	1	0	1	1	0	0	0	
1	1	0	0	0	0	1	1	0	0	1	
1	1	0	0	1	0	1	1	0	1	0	На ожидание отрицательного фронта
1	1	0	1	0	0	1	1	0	1	1	Задержка и ожидание отрицательного
1	1	0	1	1	0	1	1	1	0	0	фронта входного сигнала
1	1	1	0	0	0	1	1	1	0	1	
1	1	1	0	1	0	1	1	1	1	0	
1	1	1	1	0	0	0	1	1	1	0	Снятие Р и ожидание входного сигнала
0	1	0	1	0	1	1	0	1	0	1	Выставление RxC и переход
0	1	0	1	1	1	1	0	1	0	1	на обработку отрицательного
0	1	1	0	0	1	1	0	1	0	1	фронта входного сигнала
0	1	1	0	1	1	1	0	1	0	1	
0	1	1	1	0	0	0	0	1	0	0	На ожидание положительного фронта

Помимо синхросигнала (его обычно обозначают RxC или C), микропрограммный автомат выдает также сигнал огибающей информационного пакета (P), то есть сигнал, активный при наличии передачи информации в коде Манчестер-И и пассивный при отсутствии передачи информации (см. рис. 6.20). Сигнал C (RxC) стробирует запись сигнала данных RxD, представляющего собой просто входной сигнал в коде Манчестер-П, пропущенный через регистр.

Мы не будем подробно рассматривать составление микропрограммы для декодирования кода Манчестер-П, так как это заняло бы слишком много места. Достаточно только внимательно изучить табл. 6.10, содержащую микропрограмму с комментариями, чтобы понять, что в ней присутствуют и отключение реакции на входной сигнал, и выдержка временных интервалов (задержка), и переходы в заданные адреса, и остановки для ожидания положительного и отрицательного фронтов входного сигнала.

Лекция 12. Оперативная память

В лекции рассказывается о микросхемах оперативной памяти, алгоритмах их работы, параметрах, типовых схемах включения, а также о способах реализации на их основе некоторых часто встречающихся функций.

Ключевые слова: оперативная память, ОЗУ, RAM, статическая память, вход записи, время выборки адреса, время выборки микросхемы, тактируемая и нетактируемая память, временное хранение информации, память с параллельным и последовательным доступом, FIFO, LIFO, стек, буферная память, увеличение быстродействия памяти, энергонезависимая оперативная память.

6.2. Оперативная память

Основное отличие оперативной памяти (RAM) от постоянной (ROM) состоит в возможности оперативного изменения содержимого всех ячеек памяти с помощью дополнительного управляющего сигнала записи WR Каждая ячейка оперативной *(статической)* памяти представляет собой, по сути, регистр из триггерных ячеек, в который может быть записана информация и из которого можно информацию читать. Выбор того или иного регистра (той или иной ячейки памяти) производится с помощью кода адреса памяти. Поэтому при выключении питания вся информация из оперативной памяти пропадает (стирается), а при включении питания информация в оперативной памяти может быть произвольной.

Отметим, что существует также еще одна разновидность оперативной памяти, так называемая *динамическая* (в отличие от статической), в которой информация хранится не в регистрах (не в триггерных ячейках), а в виде заряда на конденсаторах. Эта память отличается более низкой стоимостью, меньшим быстродействием и необходимостью регулярной регенерации ("Refresh" — "освежение") информации в ней (так как конденсаторы со временем разряжаются). Область применения динамической памяти гораздо уже, чем статической, в основном она используется в качестве системной оперативной памяти компьютеров, где соображения стоимости выходят на первый план. Поэтому здесь мы о ней говорить не будем, хотя многие особенности использования статической памяти относятся и к динамической памяти.

Во всех рассмотренных в предыдущем разделе схемах постоянная память в принципе может быть заменена оперативной, только карту прошивки в данном случае придется записывать в память каждый раз заново после включения питания. Аналогично, многое из сказанного в данном

разделе про оперативную память справедливо и для постоянной памяти, но только информацию в постоянной памяти менять невозможно. Однако существуют также и специфические области применения оперативной памяти, которым и будет уделено здесь особое внимание.

Как уже отмечалось, оперативная память бывает двух основных видов: с раздельными шинами входных и выходных данных (в основном это одноразрядная память) и с двунаправленной (совмещенной) шиной входных и выходных данных (многоразрядная память). Некоторые простейшие примеры микросхем памяти обоих этих видов приведены на рис. 6.22. Выходы данных микросхем памяти имеют тип ОК (довольно редко) или 3С. Управляющие сигналы — это сигнал выбора микросхемы CS (иногда их несколько), сигнал записи WR (обычно отрицательный) и иногда сигнал разрешения выхода OE.

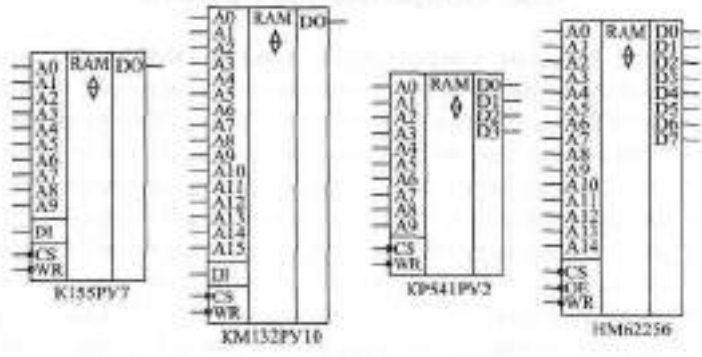


Рис. 6.22. Примеры микросхем статических ОЗУ

Микросхема оперативной памяти К155РУ7 (аналог — F9342APC) имеет организацию 1Kx1 и раздельные входной и выходной сигналы данных. Выход микросхемы — типа 3С. Управление работой микросхемы производится двумя управляющими сигналами CS и WR. Режимы работы микросхемы приведены в табл. 6.11.

Таблица 6.11. Режимы работы оперативной памяти К155РУ7

Входы и выходы					Режим работы
-CS	-WR	A0...A9	DI	DO	
1	X	X	X	3С	Хранение
0	0	Адрес	0	3С	Запись 0
0	0	Адрес	1	3С	Запись 1
0	1	Адрес	X	Данные	Чтение

Микросхема КМ 132РУ10 отличается от К155РУ7 в основном большим объемом (организация 64К х 1) и несколько меньшим быстродействием. Назначение управляющих сигналов и таблица режимов работы у этих микросхем совпадают.

Микросхема КР541РУ2 (аналог — IM7147L-3) относится к другой разновидности микросхем памяти. У нее четыре двунаправленных вывода данных типа ЗС. Управляющие сигналы те же самые: -CS и -WR. Таблица режимов работы (табл. 6.12) также похожа на таблицу для одноразрядных микросхем. Главное отличие состоит в том, что в режиме записи на входах/выходах данных присутствует записываемая информация.

Таблица 6.12. Режимы работы оперативной памяти КР541РУ2

Входы и выходы				Режим работы
-CS	-WR	A0...A9	DI00...DI03	
1	X	X	ЗС	Хранение
0	0	Адрес	0	Запись 0
0	0	Адрес	1	Запись 1
0	1	Адрес	Читаемые данные	Чтение

Микросхема HM62256 фирмы Hitachi отличается от КР541РУ2 прежде всего организацией (32К х 8) и управляющими сигналами (добавлен сигнал разрешения выхода -OE). Когда этот сигнал пассивен (равен единице), входы/выходы данных микросхемы находятся в состоянии ЗС независимо от режима работы. Введение дополнительного сигнала позволяет более гибко управлять работой микросхемы. К тому же обычно в подобных микросхемах при пассивном сигнале -CS (равном единице) значительно уменьшается потребляемая мощность.

В настоящее время имеется огромный выбор микросхем памяти с разным объемом (от нескольких байт до нескольких мегабайт), с разным количеством разрядов (обычно 1, 4, 8, 16 разрядов), с разными методами управления, с разным потреблением и быстродействием. В каждом конкретном случае надо подбирать оптимальную память, в наибольшей степени удовлетворяющую требованиям решаемой задачи.

Таблицы режимов работы (таблицы истинности) микросхем памяти не дают достаточно информации для их практического использования. Для микросхем памяти очень важны временные параметры (задержки сигналов относительно друг друга) и порядок выставления и снятия сигналов адреса, данных и управления. Всю эту информацию дают временные диаграммы циклов записи в память и чтения (считывания) из памя-

ти, приводимые в справочниках. Самые главные временные параметры оперативной памяти следующие:

- время выборки адреса (задержка между изменением адреса и выдачей данных);
- время выборки микросхемы (задержка выдачи данных по выставлению сигнала CS);
- минимальная длительность сигнала записи WR;
- минимальная длительность сигнала CS.

Всего же количество временных параметров может достигать двух-трех десятков, но мы не будем подробно останавливаться на этом, так как вся подобная информация имеется в многочисленных справочниках. Характерные величины всех временных параметров памяти составляет от единиц и даже долей наносекунд до десятков наносекунд.

Типичные временные диаграммы циклов записи и чтения приведены на рис. 6.23. Конкретные временные диаграммы для каждого типа памяти необходимо смотреть в справочниках.

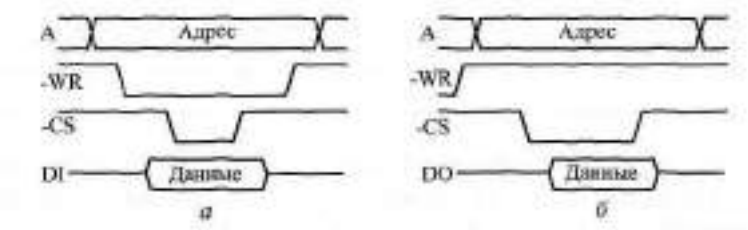


Рис. 6.23. Типичные временные диаграммы записи в память *(а)* и чтения из памяти *(б)*

Для записи информации в память надо выставить код адреса на адресных входах, выставить код записываемых в этот адрес данных на входах данных, подать сигнал записи -WR и подать сигнал выбора микросхемы -CS. Порядок выставления сигналов бывает различным, он может быть существенным или несущественным (например, можно выставлять или снимать -CS раньше или позже выставления или снятия -WR). Собственно запись обычно производится сигналом -WR или -CS, причем данные должны удерживаться в течение всего сигнала -WR (или -CS) и заданное время после его окончания.

Сигнал -CS у некоторых микросхем памяти допускается держать активным (нулевым) для всех записываемых адресов и при этом подавать импульсы -WR для каждого адреса. Точно так же у некоторых микросхем допускается держать активным (нулевым) сигнал записи WR, но при этом подавать импульсы -CS.

В случае микросхем памяти с двунаправленной шиной данных необходимо использовать источник записываемых данных с выходом ЗС или ОК, чтобы избежать конфликта данных, записываемых в память, с данными, выдаваемыми из памяти в режиме чтения.

Для чтения информации из памяти надо выставить код адреса читаемой ячейки и подать сигналы -CS и -0E (если он имеется). Сигнал WR в процессе чтения должен оставаться пассивным (равным единице). В некоторых микросхемах памяти (называемых *нетактируемыми*, например, К155РУ7, КР541РУ2, НМ62256) можно держать активным (нулевым) сигнал -CS для всех читаемых адресов. В других микросхемах (называемых *тактируемыми,* например, КМ132РУ10, К537РУ8) необходимо подавать свой импульс -CS для каждого читаемого адреса. Понятно, что нетактируемые микросхемы гораздо удобнее в применении, чем тактируемые.

Микросхемы оперативной памяти довольно часто объединяются для увеличения разрядности данных или разрядности адреса.

На рис. 6.24 показано объединение четырех микросхем К155РУ7 для получения памяти с организацией 1 Кх4. Точно так же могут быть объединены и микросхемы с двунаправленной шиной данных. К примеру, из четырех микросхем памяти с организацией 1 Кх4 можно получить память с организацией 1Кх16.

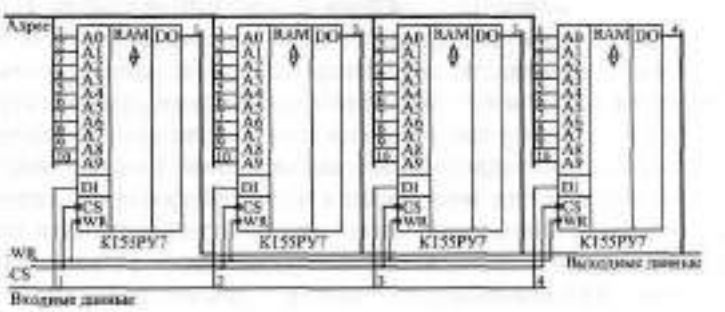


Рис. 6.24. Объединение микросхем памяти для увеличения разрядности шины данных

Для увеличения количества адресных разрядов используются те же методы, что и в случае ПЗУ (см. рис. 6.4). Если объединяются всего две микросхемы памяти, то можно обойтись без применения дешифраторов, выбирающих одну из объединяемых микросхем.

На рис. 6.25 показан вариант схемы объединения двух микросхем НМ62256 для получения памяти с организацией 64Кх8. Дополнительный старший адресный разряд управляет прохождением сигнала -CS на одну из микросхем (при нулевом уровне на дополнительном адресном разряде

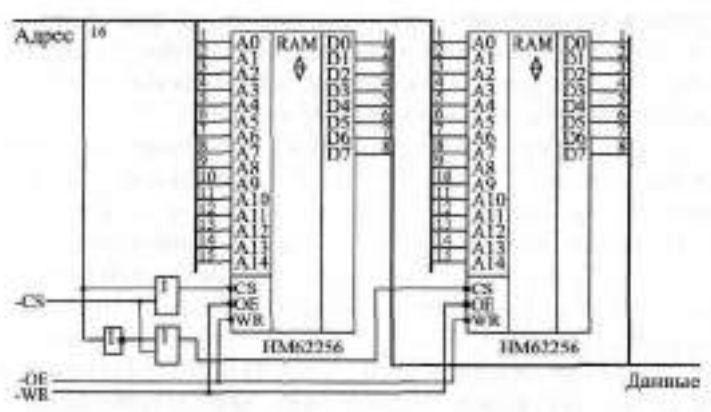


Рис. 6.25. Объединение микросхем памяти для увеличения разрядности шины адреса

сигнал -CS проходит на левую по рисунку микросхему, при единичном уровне — на правую по рисунку микросхему).

Интересной особенностью микросхем оперативной памяти является возможность произвольного изменения порядка сигналов адресных разрядов без всяких последствий для функционирования памяти. Например, сигнал, поступающий на разряд A0, можно с тем же успехом подавать на A7, сигнал, приходящий на A7, подавать на A3, сигнал, приходящий на A3, подавать на A10 и т. д. Дело в том, что информация в оперативную память записывается по тем же самым адресам, по которым потом и читается, и перестановка адресных разрядов изменяет только номер ячейки, в которую записывается информация и из которой затем читается эта же информация. Такая взаимозаменяемость адресных входов оперативной памяти бывает полезной при проектировании разводки печатных плат. В случае ПЗУ это правило не работает, так как там информация записана раз и навсегда, и читать ее надо по тем же адресам, по которым ее ранее записали.

6.2.1. ОЗУ для временного хранения информации

Главное применение микросхем оперативной памяти, непосредственно следующее из ее названия, — это временное хранение цифровой информации, всевозможных массивов кодов, таблиц данных, одиночных чисел и т. д. Цель такого хранения информации состоит в том, чтобы в любой момент иметь возможность быстро ее прочитать для дальнейшей обработки, записи в энергонезависимую память (в ПЗУ, на магнитные носители) или для другого использования. Записанная в оперативную па-

мять и непрочитанная затем информация не имеет смысла, так как при выключении питания она безвозвратно пропадет.

Другим словами, временное хранение предполагает, что к памяти имеется возможность доступа от какого-то устройства или от какой-то другой части схемы как с операцией записи, так и с операцией чтения (считывания). В зависимости от того, в каком порядке может записываться или читаться информация, существуют две разновидности ОЗУ:

- ОЗУ с параллельным или произвольным доступом (это наиболее универсальная схема);
- ОЗУ с последовательным доступом (это более специфическая схема).

Параллельный или произвольный доступ наиболее прост и обычно не требует никаких дополнительных элементов, так как именно на этот режим непосредственно рассчитаны микросхемы памяти. В этом режиме можно записывать информацию в любой адрес ОЗУ и читать информацию из любого адреса ОЗУ в произвольном порядке. Однако параллельный доступ требует формирования довольно сложных последовательностей всех входных сигналов памяти. То есть для записи информации необходимо сформировать код адреса записываемой ячейки и только потом подать данные, сопровождаемые управляющими сигналами -CS и -WR (см. рис. 6.23). Точно так же необходимо подавать полный код адреса читаемой ячейки при операции чтения. Этот режим доступа чаще всего применяется в компьютерах и контроллерах, где самыми главными факторами являются универсальность и гибкость использования памяти для самых разных целей.

Последовательный доступ к памяти предполагает более простой порядок общения с памятью. В этом случае не надо задавать код адреса записываемой или читаемой ячейки, так как адрес памяти формируется схемой автоматически. Для записи информации надо всего лишь подать код записываемых данных и сопроводить его стробом записи. Для чтения информации надо подать строб чтения и получить читаемые данные. Автоматическое задание адреса при этом осуществляется внутренними счетчиками, меняющими свое состояние по каждому обращению к памяти. Например, десять последовательных циклов записи запишут информацию в десять последовательно расположенных ячеек памяти. Недостаток такого подхода очевиден: мы не имеем возможности записывать или читать ячейки с произвольными адресами в любом порядке. Зато существенно упрощается и ускоряется процедура обмена с памятью (запись и чтение). Мы будем в данном разделе рассматривать именно этот тип памяти, этот тип доступа.

Можно выделить три основных типа оперативной памяти с последовательным доступом:

- память типа "первым вошел — первым вышел" (FIFO, First In — First Out);

- память магазинного, стекового типа, работающая по принципу "последним вошел — первым вышел" (LIFO, Last In — First Out).
- память для хранения массивов данных.

Два первых типа памяти подразумевают возможность чередования операций записи и чтения в памяти. При этом память FIFO выдает читаемые данные в том же порядке, в котором они были записаны, а память LIFO — в порядке, обратном тому, в котором они были записаны в память. Память FIFO можно сравнить со сдвиговым регистром, на выходе которого данные появляются в том же порядке, в котором они были в него записаны. А память LIFO обычно сравнивают с магазином для подачи патронов в автомате или пистолете, в котором первым выдается патрон, вставленный в магазин последним. Память с принципом LIFO используется, в частности, в компьютерах (стек), где она хранит информацию о параметрах программ и подпрограмм.

Для памяти FIFO требуется хранение двух кодов адреса (адрес для записи и адрес для чтения), для памяти LIFO достаточно одного кода адреса.

Хранение массивов в памяти предполагает, что сначала в память записывается целиком большой массив данных, а потом этот же массив целиком читается из памяти. Эта память также может быть устроена по двум принципам (FIFO и LIFO). В первом случае (FIFO) записанный массив читается в том же порядке, в котором и был записан, во втором случае (LIFO) — в противоположном порядке (начиная с конца). В обоих этих случаях для общения с памятью требуется хранить только один код адреса памяти.

Рассмотрим несколько примеров схем, реализующих перечисленные типы памяти с последовательным доступом.

На рис. 6.26 представлена функциональная схема памяти типа FIFO на микросхемах с раздельными шинами входных и выходных данных. Адреса памяти задаются двумя счетчиками — счетчиком записи и счетчиком чтения, выходные коды которых мультиплексируются с помощью 2-канального мультиплексора. Запись данных осуществляется по стробу записи "-Зап.", чтение данных — по стробу чтения "-Чт.". Своим задним фронтом сигнал "-Зап." переключает счетчик записи, а задний фронт сигнала "-Чт." переключает счетчик чтения. В результате каждая следующая запись осуществляется в следующий по порядку адрес памяти. Точно так же каждое следующее чтение производится из следующего по порядку адреса памяти.

Перед началом работы необходимо сбросить счетчики в нуль сигналом "Сброс". При отсутствии операций записи и чтения память находится в состоянии чтения (сигнал -WR равен единице), а на адресные входы памяти подается код адреса записи со счетчика записи. При подаче строба

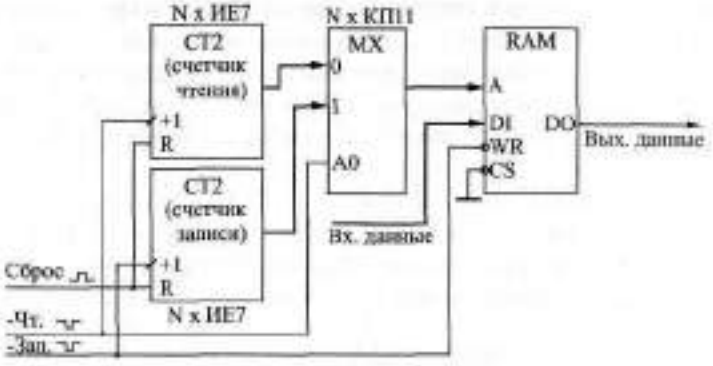


Рис. 6.26. Функциональная схема памяти типа FIFO

записи **"-Зап."** производится запись входных данных по адресу из счетчика записи. Входные (записываемые) данные должны выставляться раньше начала сигнала "-Зап.", а заканчиваться после этого сигнала. При подаче строба чтения "-Чт." мультиплексор переключается на передачу адреса чтения, и на выходе памяти появляется информация, которая считывается из адреса чтения, задаваемого счетчиком чтения. Действительными выходные (читаемые) данные будут по заднему (положительному) фронту сигнала "-Чт.".

Запись начинается с нулевого адреса памяти и производится по последовательно нарастающим адресам. Точно так же чтение начинается с нулевого адреса памяти и производится по последовательно нарастающим адресам. Операции записи и чтения могут чередоваться. Временные диаграммы циклов записи и чтения показаны на рис. 6.27.

В качестве счетчиков можно использовать нужное количество микросхем ИЕ7, в качестве мультиплексора — микросхемы КП11, в качестве памяти — К155РУ7 или любые другие нетактируемые микросхемы памяти с раздельными шинами входных и выходных данных.

Условия правильной работы схемы следующие. Длительность сигнала "-Зап." не должна быть меньше минимально допустимой длительности сигнала -WR памяти. Длительность сигнала "-Чт." не должна быть

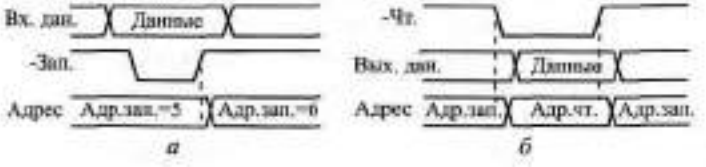


Рис. 6.27. Временные диаграммы циклов записи (*а*) и чтения (*б*) для памяти типа FIFO

меньше суммы задержки переключения мультиплексора и задержки выборки адреса памяти. Период следования сигнала **"Зап."** не должен быть меньше суммы задержки переключения счетчика записи и длительности сигнала "-Зап.". Период следования сигнала "Чт." не должен быть меньше суммы задержки переключения счетчика чтения и длительности сигнала "-Чт.".

Функциональная схема памяти типа LIFO (рис. 6.28) проще по структуре, чем схема памяти **FIFO,** так как она содержит только один счетчик и не требует мультиплексирования. В данном случае считаем, что используется память с двунаправленной шиной входных/выходных данных.

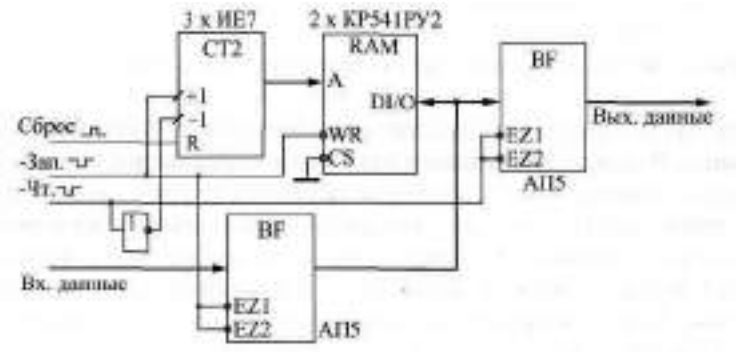


Рис. 6.28. Функциональная схема памяти типа LIFO

Счетчик адреса необходим реверсивный, с раздельными тактовыми входами прямого и обратного счета (например, **ИЕ7).** После проведения цикла записи по заднему фронту сигнала "-Зап." счетчик увеличивает свой выходной код (адрес памяти) на единицу. Перед проведением цикла чтения по переднему фронту сигнала "-Чт." счетчик уменьшает свой выходной код на единицу. Такая организация перебора адресов позволяет организовать чтение из памяти в порядке, обратном порядку записи в память.

Например, пусть исходное состояние счетчика 2. Пусть мы производим три цикла записи: первый — в адрес 2, второй — в адрес 3, третий — в адрес 4. После третьего цикла записи счетчик будет выдавать код 5. Затем проведем три цикла чтения: первый - из адреса 4 (перед чтением адрес уменьшился на единицу), второй — из адреса 3, третий — из адреса 2. После третьего цикла чтения счетчик будет выдавать код 2. Мы вернулись в исходное состояние, прочитав записанную информацию в обратном порядке.

Исходное состояние счетчика в данной схеме вообще-то не слишком важно, так как не важен текущий адрес памяти, в который производится запись и из которого потом производится чтение. Однако в случае,

когда используется начальный сброс счетчика в нулевое состояние (по сигналу "Сброс"), можно довольно просто организовать контроль за переполнением памяти **LIFO** из-за слишком большого количества записанной в нее информации. Для контроля переполнения можно использовать выходной сигнал переноса старшего счетчика (>15).

Временные диаграммы циклов записи и чтения приведены на рис. 6.29.

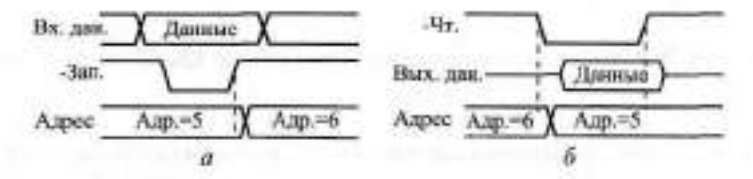


Рис. 6.29. Временные диаграммы циклов записи *(а)* и чтения (*б*) для памяти типа LIFO

В цикле записи по сигналу **"Зап."** открывается входной буфер АП5 и входные (записываемые) данные поступают на входы/выходы памяти. Одновременно по этому же сигналу память переходит в режим записи. В результате в текущий адрес памяти записываются входные данные, после чего адрес увеличивается на единицу. Входные данные должны начинаться до начала сигнала "Зап." и заканчиваться после его конца.

В цикле чтения по сигналу "4т." адрес уменьшается на единицу, после чего открывается выходной буфер АП5, который выдает на выход схемы читаемую из памяти информацию. Применение выходного буфера не обязательно, однако он предотвращает прохождение на шину выходных данных информации, записываемой в память в цикле записи. Выходные данные действительны по заднему фронту сигнала "Чт.".

Условия правильной работы схемы следующие. Длительность сигнала записи должна быть не меньше минимально допустимой длительности сигнала =WR памяти. Период следования сигналов "-Зап." не должен быть меньше суммы времени срабатывания счетчиков и длительности сигнала "-Зап.". Длительность сигнала чтения должна быть не меньше суммы времени срабатывания счетчика, времени выборки адреса памяти и задержки выходного буфера данных. Период следования сигналов "-Чт." также не должен быть меньше этой же суммы. Память должна быть нетактируемой, например, КР541РУ2.

Наконец, третий тип памяти для временного хранения данных — память для хранения массивов данных. Рассмотрим вариант схемы такой памяти типа **FIFO (рис. 6.30).**

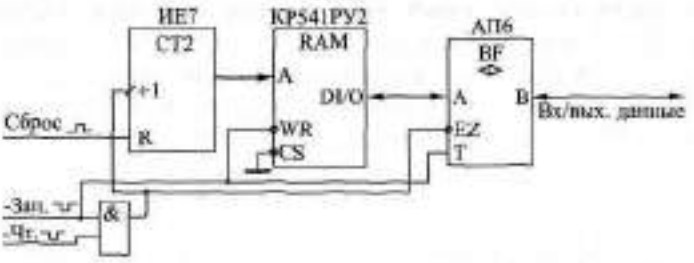


Рис. 6.30. Функциональная схема памяти для хранения массивов данных

Адреса памяти в данном случае задаются одним единственным счетчиком, который работает в режиме только прямого счета. Перед началом работы необходимо сбросить счетчик (сигнал "Сброс"). Затем производится запись массива данных. При этом после каждого цикла записи по заднему фронту сигнала "-Зап." выходной код счетчика увеличивается на единицу. После окончания записи всего массива снова надо сбросить счетчик в нуль (сигнал "Сброс"), а затем производить чтение массива, начиная с нулевого адреса. При этом после каждого цикла чтения по заднему фронту сигнала "-Чт." выходной код счетчика опять же увеличивается на единицу. В результате массив данных читается в том же порядке, что и был записан. Контроль за длиной записываемого и читаемого массивов возлагается на внешнее по отношению к приведенной схеме устройство.

Для данной схемы должна использоваться нетактируемая память (например, КР541РУ2) с двунаправленной шиной входных/выходных данных. Считаем, что данные подаются на схему и читаются из схемы также по двунаправленной шине данных. Между памятью и этой шиной включается двунаправленный буфер (типа АП6), который может понадобиться, например, для обеспечения большого выходного тока и малого входного тока со стороны внешней двунаправленной шины данных (это типичная ситуация при построении микропроцессорных и компьютерных систем). Буфер этот открывается на передачу данных в память по сигналу "-Зап." (сигнал -EZ становится равным нулю, сигнал T также нулевой) и открывается для чтения данных из памяти по сигналу "-Чт." (сигнал -EZ становится равным нулю, сигнал T равен единице).

Условия правильной работы схемы следующие. Длительность сигнала записи "-Зап." должна быть не менее минимальной длительности сигнала -WR памяти. Входные (записываемые) данные должны начинаться до начала сигнала "-Зап.", а заканчиваться после его окончания. Длительность сигнала чтения "-Чт." не должна быть меньше суммы задержки буфера и времени выборки адреса памяти. Действительными читаемые

данные будут по заднему фронту сигнала "-Чт.". За период следования сигналов "-Зап." и "-Чт." схема должна успевать выполнить операцию записи и чтения соответственно, кроме того, должен успеть полностью переключиться счетчик адреса памяти.

6.2.2. ОЗУ как информационный буфер

Второе важнейшее применение микросхем оперативной памяти состоит в организации разнообразных информационных буферов, то есть буферной памяти для промежуточного хранения данных, передаваемых между двумя устройствами или системами. Суть информационного буфера состоит в следующем: передающее устройство записывает передаваемые данные в буфер, а принимающее устройство читает принимаемые данные из буфера (рис. 6.31).

Рис. **6.31.** Включение буферной памяти

Такое промежуточное хранение позволяет лучше скоординировать работу устройств, участвующих в обмене данными, повысить их независимость друг от друга, согласовать скорости передачи и приема данных.

Пусть, например, в качестве первого устройства выступает компьютер, а в качестве второго — кабель локальной сети. Компьютеру значительно удобнее выдавать данные со скоростью, определяемой его собственным быстродействием, но в локальную сеть надо передавать данные со строго определенной скоростью, задаваемой стандартом на сеть (например, 100 Мбит/с). Кроме того, компьютер, по возможности, не должен отвлекаться на контроль за текущим состоянием сети, за ее занятостью и освобождением. Поэтому буферная память в данном случае необходима. И точно так же она нужна при приеме данных из локальной сети в компьютер.

Главное отличие буферной памяти от памяти для временного хранения информации, рассмотренной в предыдущем разделе, состоит в том, что к информационному буферу всегда имеют доступ не одно внешнее устройство, а два (или даже более). Из-за этого иногда существенно усложняется как схема задания адреса микросхемы памяти, так и схема разделения потоков данных (записываемых в память и читаемых из памяти).

Информационные буферы бывают однонаправленными (входными или выходными) и двунаправленными (то есть входными и выходными одновременно — рис. 6.32). Например, буферная память сетевого адаптера двунаправленная, так как она буферирует и информацию, передаваемую в сеть из компьютера, и информацию, принимаемую из сети в компьютер. Двунаправленные буферы всегда сложнее проектировать из-за большего количества потоков данных.

Рис. 6.32. Двунаправленный информационный буфер

Информационные буферы могут обеспечивать периодический обмен между устройствами или непрерывный обмен между ними. Примером буфера с непрерывным режимом обмена может служить контроллер видеомонитора, информация из которого постоянно выдается на видеомонитор, но может изменяться по инициативе компьютера.

Информационные буферы с периодическим режимом обмена могут быть организованы по типу FIFO или по типу LIFO.

В случае FIFO массив данных читается из памяти одним устройством в том же порядке, в каком он был записан в память другим устройством. Выпускаются даже специальные микросхемы быстродействующей буферной памяти типа FIFO, которые не имеют адресной шины и представляют собой, по сути, многоразрядный сдвиговый регистр. В отличие от обычной микросхемы сдвигового регистра, где читать вдвигаемую информацию можно только тогда, когда она продвинется по всем ячейкам регистра, информацию с выходов буфера FIFO можно начинать читать с выходов сразу же после того, как она начала записываться в его входы. Но мы будем рассматривать здесь только буферы на обычных, традиционных микросхемах памяти, как более универсальные.

В случае информационного буфера LIFO массив данных читается из памяти в порядке, противоположном тому, в котором он был записан в память. Такое решение иногда позволяет проще организовать схему перебора адресов памяти.

Тем самым, разнообразие информационных буферов огромно. Мы же рассмотрим здесь всего три примера схем буферной памяти.

Первая схема — это простейший однонаправленный буфер с периодическим режимом обмена по принципу FIFO (рис. 6.33). Одно устройство записывает информацию в буфер, на другое устройство выдается ин-

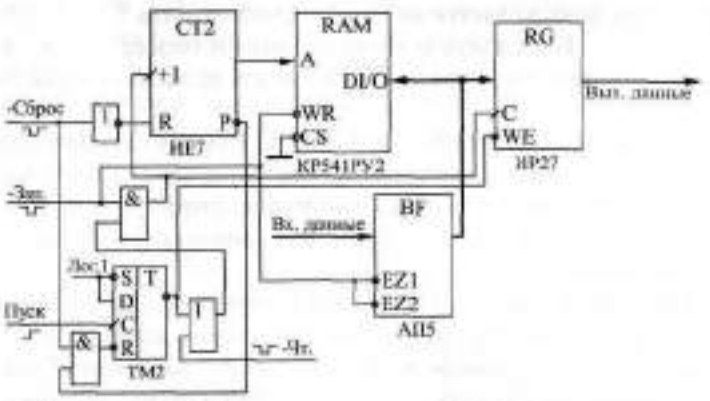


Рис. 6.33. Однонаправленный буфер типа FIFO

формация из буфера. Память всегда записывается полностью, по всем адресам, и читается также полностью. Строб записи "-Зап." поступает в режиме записи с частотой, необходимой для записи, строб чтения "-Чт." поступает при чтении с частотой, необходимой для чтения. Шины данных для записи и чтения в память в случае, показанном на рисунке, отдельные.

При таких условиях необходим всего лишь один счетчик для перебора адресов памяти, причем счетчик, работающий только в режиме прямого счета и имеющий вход начального сброса в нуль.

Перед началом работы устройство, производящее запись в память, сбрасывает счетчик в нуль сигналом "-Сброс" и устанавливает режим записи в память, перебрасывая в нуль управляющий триггер (единица на инверсном выходе). Затем начинается процесс записи: записываемые данные поступают с однонаправленного входного буфера (АП5) и записываются в память сигналом "-Зап.", который своим задним фронтом переключает адреса памяти. Полная процедура записи включает в себя столько циклов записи, сколько имеется ячеек у используемой памяти.

После окончания процедуры записи устройство, производившее запись, разрешает чтение из памяти, устанавливая в единицу триггер положительным фронтом сигнала "Пуск" (нуль на инверсном выходе). При этом разрешается прохождение сигнала "-Чт.". Адреса памяти переключаются по заднему фронту сигнала "-Чт.", и по этому же фронту данные, читаемые из памяти, фиксируются в выходном регистре, срабатывающем по фронту (например, ИР27). Выходной регистр выполняет две функции: он не пропускает на выход данные, записываемые в память (по сигналу WE запрещается запись в триггер), а также обеспечивает одновременность изменения всех разрядов читаемых данных. Выходная информация из-за

этого регистра задерживается на один период сигнала "-Чт.", что необходимо учитывать. Если взять регистр со входом сброса в нуль, то можно сделать, чтобы при процедуре записи в память на выходе схемы был нулевой код.

После окончания чтения всего объема памяти вырабатывается сигнал переноса счетчика -Р, который снова переводит всю схему в режим записи, сбрасывая триггер в нуль (единица на инверсном выходе). После этого записывающее внешнее устройство снова может начинать процедуру записи в память.

Условия правильной работы схемы следующие. Длительность сигнала **"-Зап."** не должна быть меньше минимальной длительности сигнала WR памяти. Период следования сигналов "-Зап." не должен быть меньше суммы длительности сигнала "-Зап." и задержки переключения счетчика. Период следования сигналов "-Чт." не должен быть меньше суммы времени выборки адреса памяти и задержки переключения счетчика. Память должна быть нетактируемой (например, КР541РУ2).

Более сложную структуру имеет двунаправленный буфер с периодическим режимом обмена типа LIFO. Он позволяет выдавать и принимать массивы данных произвольной длины (а не фиксированной длины, как в предыдущем случае) с заданной скоростью. Такая задача возникает, в частности, при проектировании адаптеров локальных сетей. Несмотря на то, что данные читаются из буфера в порядке, обратном тому, в котором они были записаны в буфер, на обмене информацией между двумя буферами это никак не отражается.

Пусть, например, устройство 1 передает информацию в устройство 2, а в качестве промежуточного устройства (устройство 3) выступает кабель сети (рис. 6.34).

Устройство 1 записывает в буфер 1 массив в прямом порядке, буфер 1 выдает этот массив в устройство 3 (сеть) в обратном порядке, буфер 2 принимает массив из сети в обратном порядке, а устройство 2 читает принятую информацию опять же в прямом порядке: то есть читается информация в том же порядке, в каком она и писалась. То же самое происходит и при передаче информации из устройства 2 в устройство 1.

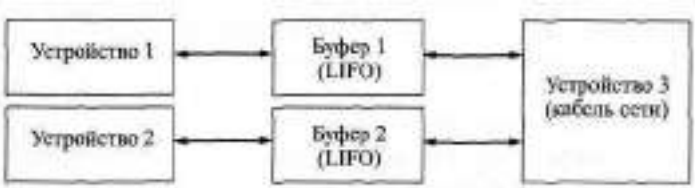


Рис. 6.34. Обмен между двумя устройствами через два буфера типа LIFO

Схема буфера LIFO (рис. 6.35) включает в себя, помимо памяти и двунаправленного буфера, реверсивный счетчик (типа ИЕ7) и реверсивный регистр сдвига (типа ИР24), служащий для преобразования параллельного кода в последовательный при передаче в сеть и последовательного кода в параллельный при приеме из сети. Режимы работы буфера задаются двумя триггерами, один из которых разрешает режим передачи в сеть, а другой — режим приема из сети.

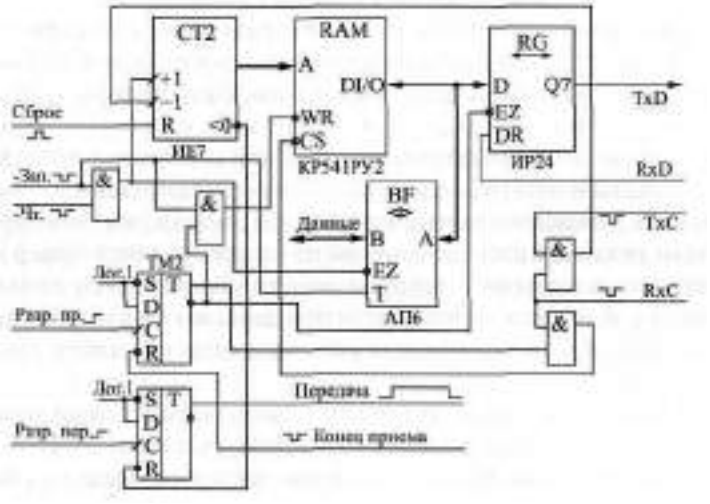


Рис. 6.35. Двунаправленный буфер типа LIFO

Перед началом работы оба триггера сброшены в нуль, счетчик также сброшен в нуль сигналом "Сброс". Сначала в память записывается передаваемый в сеть массив данных. Запись производится сигналом "-Зап.", задний фронт которого увеличивает выходной код счетчика (адрес памяти) на единицу. После окончания записи массива сигналом "Разр. пер." разрешается передача массива в сеть. В режиме передачи по сигналу строба передачи (TxC) перебираются адреса памяти в обратном порядке (счетчик работает в режиме обратного счета). Данные, читаемые из памяти, записываются в сдвиговый регистр и выдаются в сеть в последовательном коде (TxD). После того как счетчик досчитает до нуля, вырабатывается сигнал переноса <0, который сбрасывает в нуль триггер передачи. То есть в сеть выдается весь массив, записанный в память, независимо от его длины, причем массив выдается в обратном порядке.

В режиме приема информации из сети записывается единица в триггер разрешения приема по сигналу "Разр. пр.". Принимаемые из сети данные в последовательном коде RxD записываются в сдвиговый

регистр, а из него уже в параллельном коде — в память. Запись производится по сигналу строба приема RxC, задним фронтом которого переключается счетчик, работающий в режиме инверсного счета. После окончания приема по сигналу "Конец приема" сбрасывается триггер разрешения приема. После этого производится чтение информации из памяти по сигналу "-Чт.". Задним фронтом этого сигнала переключается счетчик, работающий в режиме прямого счета. То есть массив читается в порядке, обратном тому, в котором он пришел из сети.

Условия правильной работы данной схемы аналогичны тем, что были сформулированы для предыдущих рассмотренных схем буферов. Сигналы стробов записи и чтения должны иметь такую длительность, чтобы осуществлять соответственно запись в память и чтение из памяти. Период следования этих сигналов должен быть таким, чтобы успевали производиться операции записи и чтения, а также успевал переключаться счетчик.

Наконец, последняя схема, которую мы рассмотрим, это буфер с непрерывным режимом работы: с одним из устройств такой буфер общается непрерывно, а с другим — только в момент обращения со стороны этого устройства. В данном случае уже необходимо иметь два счетчика адреса памяти, выходные коды которых надо мультиплексировать с помощью мультиплексора.

Примем для простоты, что буфер — однонаправленный и передающий, то есть одно устройство только записывает в память информацию (в нужные моменты), а на другое устройство постоянно выдается читаемая из всех подряд адресов памяти информация (рис. 6.36).

Счетчик чтения непрерывно перебирает адреса памяти с частотой тактового генератора. Читаемая из памяти информация записывается в выходной регистр и выдается на выход. В момент записи по сигналу "-Зап." мультиплексор подает на адресные входы памяти выходной код

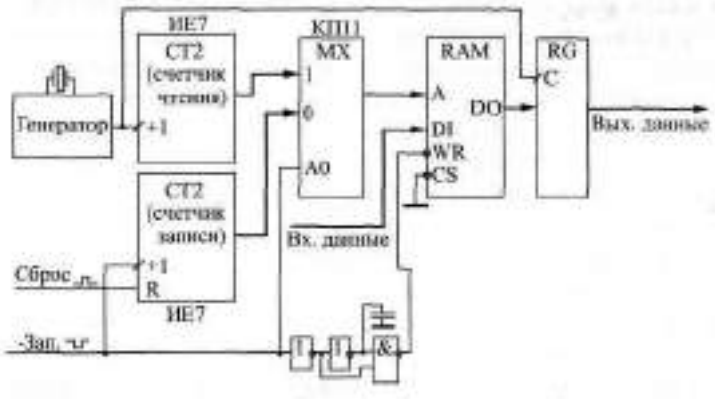


Рис. 6.36. Передающий буфер с непрерывным режимом работы

счетчика записи. На память подается сигнал –WR, вложенный в сигнал "-Зап." (он начинается после начала сигнала "-Зап." и заканчивается раньше этого сигнала). Это достигается применением цепочки из двух инверторов и элемента 2И-НЕ. Такая последовательность сигналов позволяет записать в память входные данные по адресу записи со счетчика записи и не изменять содержимое ячеек памяти с другими адресами.

Перед началом записи памяти счетчик записи сбрасывается в нуль по сигналу "Сброс". После каждой операции записи по заднему фронту сигнала "–Зап." код на выходе счетчика записи увеличивается на единицу. То есть для того чтобы записать всю память, необходимо сбросить счетчик и произвести столько циклов записи, сколько ячеек имеется в памяти.

Условия правильной работы схемы следующие. Счетчики должны быть синхронными для быстрого переключения. Память должна быть нетактируемая и с раздельными входами и выходами данных. Емкость конденсатора должна быть такой, чтобы формируемый импульс –WR имел достаточную длительность для записи информации в память. За длительность сигнала "-Зап." должен успеть сработать мультиплексор, и должна записаться информация в память. Выходной регистр должен срабатывать по фронту. Длительность периода тактового сигнала должна быть не меньше суммы задержки выборки адреса памяти и задержки переключения счетчика чтения. За период следования сигнала "-Зап." должна успеть записаться информация в память, и должен переключиться счетчик записи.

Недостаток приведенной организации буфера состоит в том, что при проведении цикла записи в память на выходе схемы будет не та информация, которая должна читаться из памяти в данный момент. Преодолеть этот недостаток можно двумя путями.

Первый путь состоит в том, что надо производить запись в память только в те моменты, когда выходная информация буфера не важна. Например, если речь идет о буфере контроллера видеомонитора, то запись в память можно производить только во время кадрового гасящего импульса, когда на экране ничего не отображается.

Второй путь более сложен. Он состоит в том, чтобы разделить во времени запись в память и чтение из памяти. Например, в первой половине такта (то есть периода тактового генератора) производится запись в память (если есть внешний сигнал записи), а во второй половине такта всегда производится чтение информации из памяти и запись ее в выходной регистр. Соответственно мультиплексор в первой половине периода подает на адресные входы памяти адрес записи, а во второй половине — адрес чтения. Временную привязку момента записи к ближайшей первой половине такта можно осуществить с помощью микропрограммного ав-

томата. При таком решении запись в память можно производить в любой момент без искажения читаемой информации, однако существенно (минимум вдвое) возрастают требования к быстродействию всех микросхем.

6.2.3. Улучшение параметров ОЗУ

При применении оперативной памяти часто встает задача улучшить ее характеристики. О совместном включении нескольких микросхем с целью увеличения разрядности шины адреса и шины данных уже говорилось. Здесь же мы остановимся на задаче повышения быстродействия памяти, то есть увеличения предельной тактовой частоты, с которой можно записывать информацию в память и читать информацию из памяти.

Наверное, самое распространенное и самое простое решение, позволяющее повысить быстродействие памяти, состоит в применении сдвиговых регистров. Сдвиговые регистры всегда имеют существенно большее быстродействие, чем память, так как они имеют гораздо более простую структуру. Частота следования тактовых импульсов этих регистров может достигать десятков и сотен мегагерц, тогда как память с такими параметрами найти трудно.

Увеличение быстродействия памяти достигается с помощью сдвиговых регистров очень просто: уменьшение в несколько раз разрядности шины данных памяти позволяет во столько же раз увеличить частоту записи информации в память или чтения информации из памяти.

Например, если необходимо в 8 раз увеличить частоту чтения информации из памяти, то надо соединить нужное количество микросхем памяти для увеличения разрядности шины данных в 8 раз, а затем применить на выходах данных схему (рис. 6.37) на основе 8-разрядного регистра сдвига. 8-разрядный код, читаемый из памяти, записывается в сдвиговый регистр, а затем сдвигается семь раз с частотой, в 8 раз большей, чем частота опроса памяти. И запись, и сдвиг производятся одним тактовым сигналом с генератора. Восемь тактовых импульсов отсчитываются синхронным счетчиком. Для управления работой регистра сдвига применен элемент ЗИЛИ-НЕ, выдающий положительный импульс в течение первой 1/8 периода опроса памяти. Этот же сигнал используется как строб чтения из памяти (своим задним фронтом он переключает адреса памяти).

В случае необходимости увеличения частоты записи в память одного сдвигового регистра недостаточно. Дело в том, что информация в память записывается не по фронту сигнала, а по уровню, то есть записываемая информация должна сохраняться на входе памяти определенное время. Поэтому код с выхода сдвигового регистра необходимо перед записью в память переписать в параллельный регистр, где он будет затем храниться в течение всего периода записи в память.

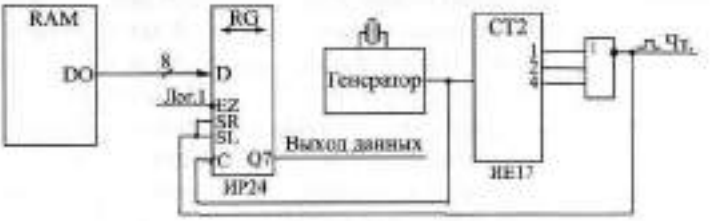


Рис. 6.37. Увеличение частоты чтения информации

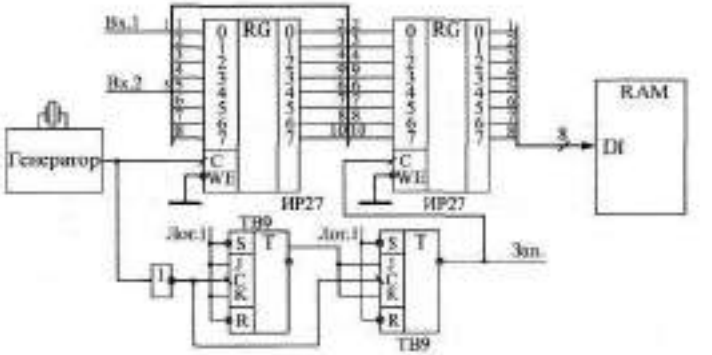


Рис. 6.38. Увеличение частоты записи информации

Схема, показанная на рис. 6.38, ускоряет частоту записи в память в 4 раза.

В данном случае в качестве регистра сдвига удобно использовать обычный параллельный регистр, срабатывающий по фронту, у которого выходы трех разрядов соединены со входами следующих разрядов. При этом из одного 8-разрядного регистра мы получаем два 4-разрядных регистра сдвига. 2-разрядная входная информация записывается в эти два 4-разрядных регистра сдвига, затем переписывается в параллельный регистр и только потом записывается в память. Для отсчета четырех импульсов тактового генератора применен 2-разрядный счетчик на двух JK-триггерах, включенных в счетном режиме, что позволяет несколько повысить быстродействие по сравнению со стандартными микросхемами счетчиков. Сигнал с выхода второго триггера записывает информацию в параллельный регистр, а также используется в качестве строба записи в память "Зап.".

Большой недостаток оперативной памяти состоит в том, что информация, записанная в нее, исчезает при выключении источника питания. Поэтому часто используется дополнительный источник питания (гальва-

ническая батарея или аккумулятор), который питает при выключении источника питания только микросхемы памяти. В данном случае очень удобны микросхемы ОЗУ, выполненные по **КМОП** технологии, ток потребления которых в статическом режиме (при неизменных входных и выходных сигналах) очень мал (порядка единиц микроампер). В результате получается так называемая энергонезависимая оперативная память, содержимое которой может легко перезаписываться, но не пропадает при выключении питания, как в ПЗУ.

Схема энергонезависимой памяти (рис. 6.39) довольно проста, хотя и имеет ряд неочевидных особенностей.

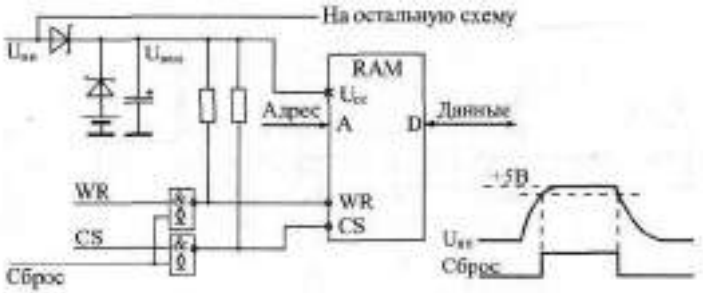


Рис. 6.39. Энергонезависимая оперативная память

Дело в том, что управляющие сигналы памяти -WR и -CS имеют активный низкий уровень, а при выключении питания все входные сигналы памяти, естественно, станут нулевыми. Это приведет к искажению записанной в память информации. Поэтому необходимо обеспечить, чтобы при выключении питания сигналы на входах -WR и -CS были пассивными, то есть имели уровень логической единицы. Для этого обычно используются логические элементы с выходами ОК, нагрузочные резисторы которых присоединяются не к пропадающему напряжению питания памяти $U_{ип}$, а к сохраняющемуся напряжению питания памяти $U_{пам}$. Для получения напряжения $U_{пам}$ используется простая схема на двух диодах (лучше брать диоды Шоттки с меньшим падением напряжения), которая передает на выход $U_{пам}$ напряжение источника питания $U_{ип}$ (если питание включено) или напряжение от гальванической батареи 3—4,5 В (если питание выключено).

Для большей гарантии от пропадания информации во время переходных процессов (при постепенном нарастании $U_{ип}$ и при постепенном его уменьшении) необходимо управлять прохождением сигналов WR и CS на память с помощью управляющего сигнала "Сброс". Этот сигнал равен нулю при напряжения $U_{ип}$ менее 4,7—4,8 В и равен единице при нормальном напряжении $U_{ип}$ = 5 В (временная диаграмма приведена на рисун-

КС). В результате такого решения память отключается от остальной схемы при недостаточном напряжении питания (сигналы WP и -CS равны единице) и подключается к остальной схеме при нормальном напряжении питания.

В заключение данной главы надо отметить, что в ней сознательно не рассмотрена одна из важнейших областей применения микросхем памяти (как постоянной, так и оперативной) — микропроцессорные системы и компьютерные системы. Дело в том, что говорить о применении памяти в этой области невозможно без изложения основ микропроцессорной и компьютерной схемотехники, а это отдельная большая тема, требующая специальной книги. К тому же изучение методов включения памяти, которые рассмотрены в данной главе, позволяет в дальнейшем довольно легко понять принципы применения памяти в любых возможных областях.

Глава 7. Применение микросхем ЦАП и АЦП

Лекция 13. Применение ЦАП и АЦП

В лекции рассматриваются принципы работы аналого-цифровых и цифро-аналоговых преобразователей, о типах микросхем ЦАП и АЦП, их алгоритмах работы, параметрах, типовых схемах включения, а также о способах реализации на их основе некоторых часто встречающихся функций.

Ключевые слова: ЦАП, АЦП, DAC, ADC, опорное напряжение, умножающий ЦАП, матрица R–2R, биполярный режим, задержка преобразования, коды выборок, управляемый аттенюатор, управляемый усилитель, управляемая схема сдвига уровня, параллельный и последовательный АЦП, динамический диапазон, вычислитель амплитуды сигнала, увеличение частоты преобразования.

Как уже отмечалось в первой главе, цифро-аналоговые преобразователи (ЦАП, DAC — "Digital-to-Analog Converter") и аналого-цифровые преобразователи (АЦП, ADC — "Analog-to-Digital Converter") главным образом применяются для сопряжения цифровых устройств и систем с внешними аналоговыми сигналами, с реальным миром. При этом АЦП преобразует аналоговые сигналы во входные цифровые сигналы, поступающие на цифровые устройства для дальнейшей обработки или хранения, а ЦАП преобразует выходные цифровые сигналы цифровых устройств в аналоговые сигналы (см. рис. 1.23).

ЦАП и АЦП применяются в измерительной технике (цифровые осциллографы, вольтметры, генераторы сигналов и т. д.), в бытовой аппаратуре (телевизоры, музыкальные центры, автомобильная электроника и т. д.), в компьютерной технике (ввод и вывод звука в компьютерах, видеомониторы, принтеры и т. д.), в медицинской технике, в радиолокационных устройствах, в телефонии и во многих других областях. Применение ЦАП и АЦП постоянно расширяется по мере перехода от аналоговых к цифровым устройствам.

В качестве ЦАП и АЦП обычно применяются специализированные микросхемы, выпускаемые многими отечественными и зарубежными фирмами.

Сразу же надо отметить, что для грамотного и профессионального использования микросхем ЦАП и АЦП совершенно не достаточно знания цифровой схемотехники. Эти микросхемы относятся к аналого-цифровым, поэтому они требуют также знания аналоговой схемотехники, су-

щественно отличающейся от цифровой. Практическое применение ЦАП и АЦП требует расчета аналоговых цепей, учета многочисленных погрешностей преобразования (как статических, так и динамических), знания характеристик и особенностей аналоговых микросхем (в первую очередь, операционных усилителей) и многого другого, что далеко выходит за рамки этой книги. Существует обширная литература, специально посвященная именно вопросам применения ЦАП и АЦП. Поэтому в данной главе мы не будем говорить о специфике выбора и принципах включения конкретных микросхем ЦАП и АЦП мы будем рассматривать только основные особенности методов соединения ЦАП и АЦП с цифровыми узлами. Нас будет в первую очередь интересовать организация цифровых узлов, предназначенных для соединения с ЦАП и АЦП.

7.1. Применение ЦАП

В общем случае микросхему ЦАП можно представить в виде блока (рис. 7.1), имеющего несколько цифровых входов и один аналоговый вход, а также аналоговый выход.

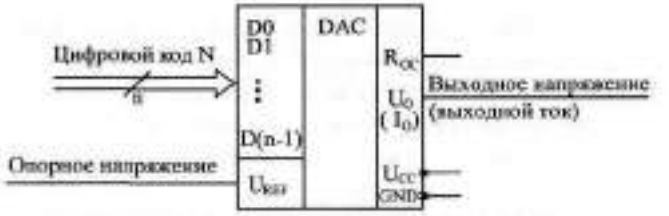


Рис. 7.1. Микросхема ЦАП

На цифровые входы ЦАП подается n-разрядный код N, на аналоговый вход — опорное напряжение $U_{ОП}$ (другое распространенное обозначение — U_{REF}). Выходным сигналом является напряжение $U_{ВЫХ}$ (другое обозначение — U_O) ИЛИ ТОК $I_{ВЫХ}$ (другое обозначение — I_O). При этом выходной ток или выходное напряжение пропорциональны входному коду и опорному напряжению. Для некоторых микросхем опорное напряжение должно иметь строго заданный уровень, для других допускается менять его значение в широких пределах, в том числе и изменять его полярность (положительную на отрицательную и наоборот). ЦАП с большим диапазоном изменения опорного напряжения называется умножающим ЦАП, так как его можно легко использовать для умножения входного кода на любое опорное напряжение.

Кроме информационных сигналов, микросхемы ЦАП требуют также подключения одного или двух источников питания и общего провода.

Обычно цифровые входы **ЦАП** обеспечивают совместимость со стандартными выходами микросхем ТТЛ.

Чаще всего в случае, если ЦАП имеет токовый выход, его выходной ток преобразуется в выходное напряжение с помощью внешнего операционного усилителя и встроенного в **ЦАП** резистора $R_{ОС}$, ОДИН ИЗ ВЫВОДОВ которого выведен на внешний вывод микросхемы (рис. 7.2). Поэтому, если не оговорено иное, мы будем в дальнейшем считать, что выходной сигнал ЦАП — напряжение U_0.

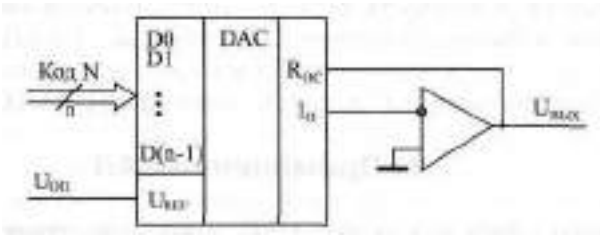


Рис. 7.2. Преобразование выходного тока ЦАП в выходное напряжение

Суть преобразования входного цифрового кода в выходной аналоговый сигнал довольно проста. Она состоит в суммировании нескольких токов (по числу разрядов входного кода), каждый последующий из которых вдвое больше предыдущего. Для получения этих токов используются или транзисторные источники тока, или резистивные матрицы, коммутируемые транзисторными ключами.

В качестве примера на рис. 7.3 показано 4-разрядное (n = 4) цифро-аналоговое преобразование на основе резистивной матрицы R-2R и ключей (в реальности используются ключи на основе транзисторов). Правому положению ключа соответствует единица в данном разряде входного кода N (разряды D0 ... D3). Операционный усилитель может быть как встро-

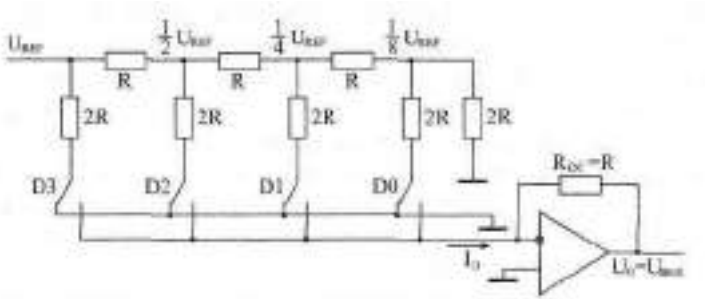


Рис. 7.3. 4-разрядное цифро-аналоговое преобразование

енным (в случае ЦАП с выходом по напряжению), так и внешним (в случае ЦАП с выходом по току).

Первым (левым по рисунку) ключом коммутируется ток величиной U_{REF} /2R, вторым ключом — ток U_{REF} /4R, третьим — ток U_{REF} /8R, четвертым — ток U_{REF} /16R. То есть токи, коммутируемые соседними ключами, различаются вдвое, как и веса разрядов двоичного кода. Токи, коммутируемые всеми ключами, суммируются и преобразуются в выходное напряжение с помощью операционного усилителя с сопротивлением R_{OC} = R в цепи отрицательной обратной связи.

При правом положении каждого ключа (единица в соответствующем разряде входного кода ЦАП) ток, коммутируемый этим ключом, поступает на суммирование. При левом положении ключа (нуль в соответствующем разряде входного кода ЦАП) ток, коммутируемый этим ключом, на суммирование не поступает.

Суммарный ток I_0 от всех ключей создает на выходе операционного усилителя напряжение $U_0 = I_0 R_{OC} = I_0 R$. То есть вклад первого ключа (старшего разряда кода) в выходное напряжение составляет $U_{REF}/2$, второго — $U_{REF}/4$, третьего — $U_{REF}/8$, четвертого — $U_{REF}/16$. Таким образом, при входном коде N = 0000 выходное напряжение схемы будет нулевым, а при входном коде N = 1111 оно будет равно $-15U_{REF}/16$.

В общем случае выходное напряжение ЦАП при R_{OC} = R будет связано со входным кодом N и опорным напряжением U_{REF} простой формулой

$$U_{ВЫХ} = -N \cdot U_{REF} 2^{-n},$$

где n — количество разрядов входного кода. Знак минус получается из-за инверсии сигнала операционным усилителем. Эту связь можно проиллюстрировать также табл. 7.1.

Таблица 7.1. Преобразование ЦАП в однополярном режиме

Входной код N	Выходное напряжение $U_{ВЫХ}$
000...000	0
000...001	$-2^{-n} U_{REF}$
100...000	$-2^{-1} U_{REF}$
111...111	$-(1-2^{-n}) U_{REF}$

Некоторые микросхемы ЦАП предусматривают возможность работы в биполярном режиме, при котором выходное напряжение изменяет-

ся не от нуля до U_{REF}, а от $-U_{REF}$ до $+U_{REF}$. При этом выходной сигнал ЦАП $U_{ВЫХ}$ умножается на 2 и сдвигается на величину U_{REF}. Связь между входным кодом N и выходным напряжением $U_{ВЫХ}$ будет следующей:

$$U_{ВЫХ} = U_{REF} (1-N \cdot 2^{1-n}).$$

Это можно проиллюстрировать табл. 7.2. Такое биполярное преобразование при возможности смены знака опорного напряжения называется также четырехквадрантным умножением (То есть и опорное, и выходное напряжения могут быть в данном случае как положительными, так и отрицательными).

Таблица 7.2. Преобразование ЦАП в биполярном режиме

Входной код N	Выходное напряжение $U_{ВЫХ}$
000...000	UREF
011...111	$2^{n} U_{REF}$
100...000	0
111...111	$-(1-2^{1-n}) U_{REF}$

Микросхемы ЦАП, имеющиеся на рынке, различаются количеством разрядов (от 8 до 24), величиной задержки преобразования (от единиц наносекунд до единиц микросекунд), допустимой величиной опорного напряжения (обычно — единицы вольт), величинами погрешностей преобразования и другими параметрами. Различаются они также технологией изготовления и особенностями внутренней структуры, что нередко накладывает ограничения на их использование. Поэтому выбирать микросхему ЦАП для конкретного применения необходимо с использованием подробной справочной информации, предоставляемой фирмами-изготовителями. Мы же будем говорить только об общих принципах включения ЦАП в цифровые схемы без учета их частных особенностей.

Иногда бывает необходимо уменьшить количество разрядов ЦАП. Для этого нужно подать сигналы логического нуля на нужное число младших разрядов ЦАП (но никак не старших разрядов). На рис. 7.4 показано, как из 10-разрядного ЦАП можно сделать 8-разрядный, подав нули на два младших разряда. Увеличение количества разрядов ЦАП представляет собой гораздо более сложную задачу, требующую построения сложных аналоговых схем, поэтому оно встречается довольно редко. Значи-

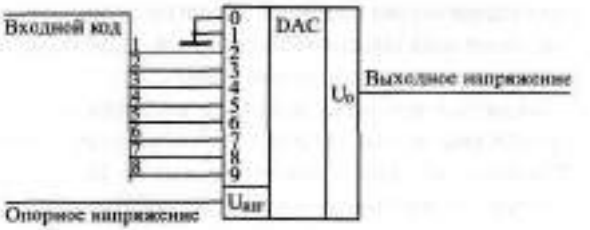


Рис. 7.4. Уменьшение разрядности ЦАП

тельно проще подобрать микросхему с нужным или с большим, чем нужно, количеством разрядов.

Основное применение микросхем ЦАП состоит в получении аналогового сигнала из последовательности цифровых кодов (рис. 7.5). Как правило, коды подаются на входы ЦАП через параллельный регистр, что позволяет обеспечить одновременность изменения всех разрядов входного кода ЦАП. При неодновременном изменении разрядов входного кода на выходе ЦАП появляются большие короткие импульсы напряжения, уровни которых не соответствуют ни одному из кодов.

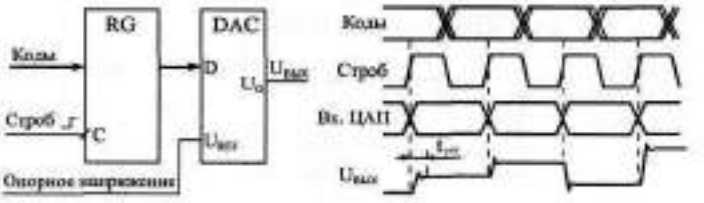


Рис. 7.5. Преобразование последовательности кодов в выходное напряжение

Однако, даже при одновременном изменении всех разрядов входного кода ЦАП, уровень напряжения, соответствующий поданному коду, устанавливается не сразу, а за время установления ЦАП $t_{УСТ}$, что связано с неидеальностью внутренних элементов ЦАП. Выходной ток ЦАП, как правило, устанавливается значительно быстрее выходного напряжения, так как он не зависит от инерционности операционного усилителя. Понятно, что условие правильной работы ЦАП состоит в том, чтобы длительность сохранения входного кода была больше, чем время установления ЦАП $t_{УСТ}$, иначе выходной сигнал не успеет принять значение, соответствующее входному коду.

Если подавать коды на вход ЦАП редко, то приведенная на рис. 7.5 схема может использоваться, например, в управляемом источнике пи-

тания, выходное напряжение которого задается входным кодом. Правда, при этом необходимо еще обеспечить большой выходной ток источника питания, применив внешний усилитель тока.

Если же подавать коды на вход ЦАП с высокой частотой, то можно получить генератор (он же синтезатор) аналоговых сигналов произвольной формы. В этом случае коды, поступающие на ЦАП, называют кодами выборок (то есть мгновенных значений) генерируемого аналогового сигнала.

В простейшем случае в качестве источника входных кодов ЦАП можно использовать обычный двоичный счетчик (рис. 7.6). Выходное напряжение ЦАП будет нарастать при этом на величину $2^n U_{REF}$ с каждым тактовым импульсом, формируя пилообразные выходные сигналы амплитудой U_{REF}. Длительность каждой ступеньки равна периоду тактового генератора T, а период всего выходного сигнала равен 2T. Количество ступенек в периоде выходного сигнала равно 2^n. Если в данной схеме использовать синхронные счетчики с синхронным переносом, то входной регистр ЦАП не нужен, так как все разряды счетчика переключаются одновременно. Если же используются асинхронные счетчики или синхронные счетчики с асинхронным переносом, то входной регистр ЦАП необходим.

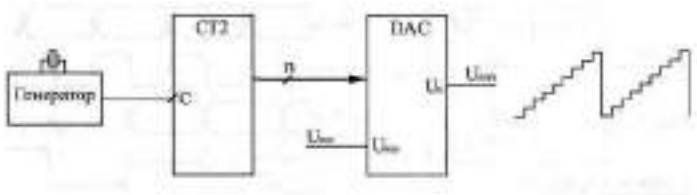


Рис. 7.6. Генератор пилообразного аналогового сигнала

В случае, когда нужно формировать аналоговые сигналы произвольной формы (синусоидальные, колоколообразные, шумовые, треугольные, импульсные и т. д.), в качестве источника кодов, поступающих на ЦАП, необходимо использовать память, работающую в режиме чтения (рис. 7.7).

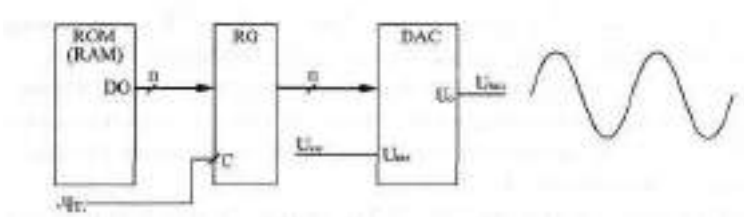


Рис. 7.7. Генерация сигналов произвольной формы

Если память постоянная, то набор форм генерируемых сигналов задается раз и навсегда. Если же память оперативная, то строится однонаправленный информационный буфер с периодическим режимом работы, что позволит записывать в память коды для генерации самых разных сигналов. В обоих случаях входной регистр ЦАП необходим, информация в него записывается стробом чтения из памяти.

Как и в предыдущем случае, выходной сигнал ЦАП будет состоять из ступенек, высота которых кратна $2^{-n}U_{REF}$. Амплитуда выходного сигнала не превышает U_{REF}. Если адреса памяти перебираются счетчиком, то период выходного аналогового сигнала равен 2^nT, где T — период тактового сигнала чтения из памяти "~Чт.", а n — количество адресных разрядов памяти.

Если надо вычислить коды выборок для генерации какого-то периодического сигнала, то необходимо его период разделить на 2^n частей и вычислить соответствующие 2^n значений этого сигнала U_i. Затем надо пересчитать значения сигнала в коды по формуле $N_i = 2^nU_i/A$, где A — амплитуда сигнала, и взять ближайшее целое значение кода. Нулевое значение сигнала даст при этом нулевой код 000. . .000, максимальное значение сигнала (равное амплитуде A) даст максимальный код 111. . . 111. В результате подачи этих кодов на ЦАП с периодом T будет генерироваться аналоговый сигнал требуемой формы с амплитудой, равной U_{REF} и с периодом $T_{ВЫХ} = 2^nT$. Пример такого вычисления проиллюстрирован рис. 7.8.

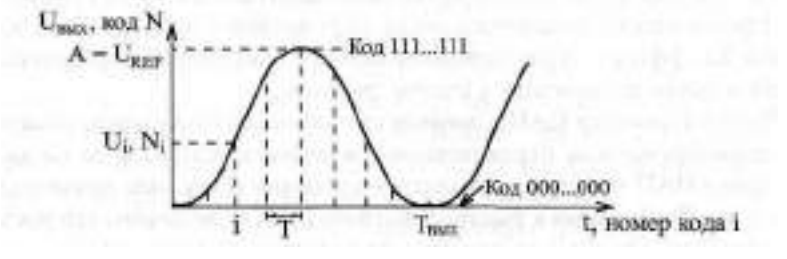


Рис. 7.8. Вычисление кодов выборок периодического сигнала

Подробнее задача проектирования генератора аналоговых сигналов произвольной формы будет рассмотрена в следующей главе.

Преобразование цифровых кодов в аналоговый сигнал — это не единственное применение микросхем ЦАП. Они могут также использоваться для управляемой обработки аналоговых сигналов, например, для усиления и ослабления аналоговых сигналов в заданное число раз. Для этого лучше всего подходят умножающие ЦАП, которые допускают изменение уровня опорного напряжения в широких пределах, в том числе и с изменением его знака. Таких микросхем ЦАП выпускается сейчас достаточно много, с различным быстродействием и с различным количеством разрядов входного кода.

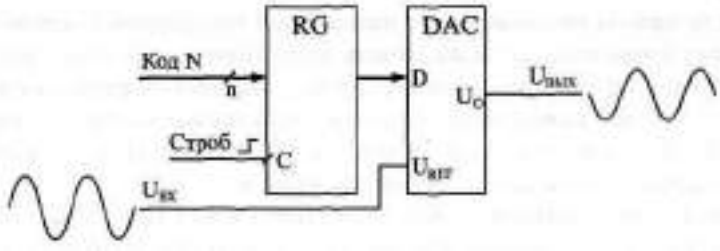


Рис. 7.9. Аттенюатор аналогового сигнала на ЦАП

Самая простейшая схема — это цифровой аттенюатор (ослабитель) аналогового сигнала (рис. 7.9), применяемый часто для регулировки амплитуды выходного сигнала генератора на основе ЦАП.

Схема практически ничем не отличается от схемы на рис. 7.5. Но два важных отличия все же имеются: вместо постоянного опорного напряжения подается переменный аналоговый сигнал, а ЦАП должен быть обязательно умножающим. Выходной сигнал связан со входным по простой формуле

$$U_{ВЫХ} = -U_{ВХ} * N\ 2^{-n},$$

то есть выходной сигнал пропорционален входному (с инверсией), а коэффициент пропорциональности определяется входным цифровым кодом N. Коэффициент пропорциональности изменяется в данном случае от нуля и почти до единицы с шагом, равным 2".

Входной регистр ЦАП в данном случае также необходим, поскольку при неодновременном переключении разрядов входного кода на выходной сигнал ЦАП могут накладываться короткие импульсы значительной амплитуды. Требования к быстродействию ЦАП (к величине его времени установления) в данном включении не слишком высоки, так как амплитуду выходного сигнала обычно требуется менять нечасто. А частота входного аналогового сигнала может быть довольно большой, она никак не связана с временем установления ЦАП.

Существует также схема включения ЦАП, которую можно использовать как управляемый усилитель аналогового сигнала с коэффициентом усиления, задаваемым входным кодом N (рис. 7.10).

В этом случае выходной ток ЦАП равен величине $U_{ВХ}/R_{ОС}$, а так как в качестве опорного напряжения используется выходное напряжение, то получается, что выходное напряжение связано со входным по формуле

$$U_{ВЫХ} = -U_{ВХ} * 2^n/N.$$

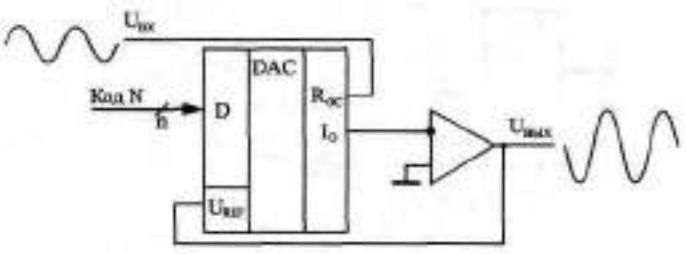


Рис. 7.10. Управляемый усилитель входного сигнала

То есть коэффициент пропорциональности между выходным и входным напряжениями обратно пропорционален коду N. Код N может меняться в этом случае от 1 до (2^n-1), что соответствует коэффициенту усиления от примерно единицы до 2^n. Например, при 10-разрядном ЦАП коэффициент усиления схемы может достигать 1024.

Как и в предыдущем случае, скорость переключения ЦАП не очень важна, так как коэффициент усиления обычно не требуется переключать слишком часто. На схеме для простоты не показан входной регистр ЦАП, который опять же необходим, чтобы обеспечить одновременность переключения всех разрядов входного кода.

Используя последовательное включение схем рис. 7.9 и рис. 7.10, можно обеспечить приведение к стандартному уровню входного напряжения, изменяемого в очень широких пределах (рис. 7.11). Такая задача часто встречается в аналого-цифровых системах. Коэффициент передачи всей схемы будет равен отношению входных кодов обоих ЦАП N/M и может быть установлен с высокой точностью как в диапазоне от 0 до 1 (аттенюатор), так и в диапазоне от 1 до 2^n (усилитель). На схеме не показаны входные регистры обоих ЦАП, но они также нужны.

Наконец, последняя схема с применением ЦАП, которую мы рассмотрим, — это схема сдвига аналогового сигнала на величину, задаваемую входным цифровым кодом. Сдвиг представляет собой, по сути, сложение аналогового сигнала с постоянным напряжением. Такая задача довольно часто встречается в аналого-цифровых системах.

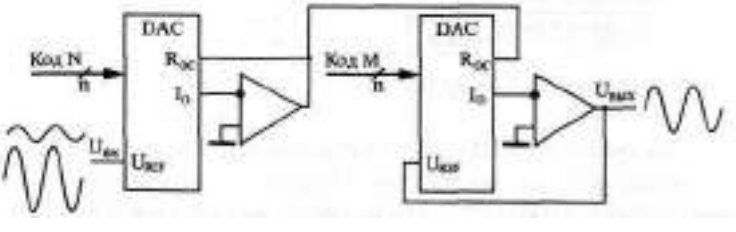


Рис. 7.11. Последовательное включение аттенюатора и усилителя

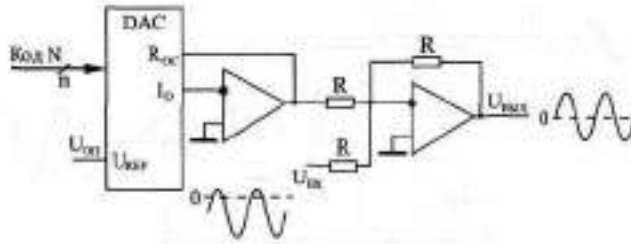


Рис. 7.12. Схема управляемого сдвига аналогового сигнала

Схема сдвига (рис. 7.12) включает в себя преобразователь цифрового кода в выходное напряжение и аналоговый сумматор на операционном усилителе. Величина напряжения сдвига входного сигнала будет равна $U_{REF} \cdot 2^{-N}$. Поскольку применяются два инвертирующих операционных усилителя, инверсии входного сигнала на выходе в данном случае не будет. Если нужен как положительный, так и отрицательный сдвиг, то необходимо применять ЦАП с биполярным выходным сигналом.

7.2. Применение АЦП

Микросхемы АЦП выполняют функцию, прямо противоположную функции ЦАП, — преобразуют входной аналоговый сигнал в последовательность цифровых кодов. В общем случае микросхему АЦП можно представить в виде блока, имеющего один аналоговый вход, один или два входа для подачи опорного (образцового) напряжения, а также цифровые выходы для выдачи кода, соответствующего текущему значению аналогового сигнала (рис. 7.13).

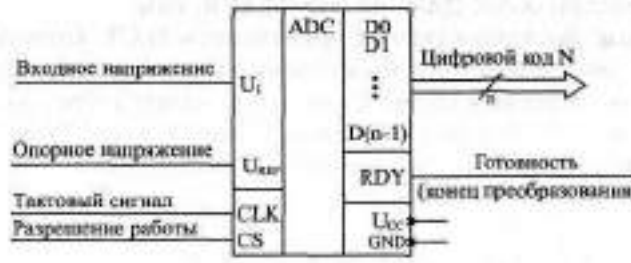


Рис. 7.13. Микросхема АЦП

Часто микросхема АЦП имеет также вход для подачи тактового сигнала CLK, сигнал разрешения работы CS и сигнал, говорящий о готовности выходного цифрового кода RDY. На микросхему подается одно или два питающих напряжения и общий провод. В целом микросхемы АЦП слож-

нее, чем микросхемы ЦАП, их разнообразие заметно больше, и поэтому сформулировать для них общие принципы применения сложнее.

Опорное напряжение АЦП задает диапазон входного напряжения, в котором производится преобразование. Оно может быть постоянным или же допускать изменение в некоторых пределах. Иногда предусматривается подача на АЦП двух опорных напряжений с разными знаками, тогда АЦП способен работать как с положительными, так и с отрицательными входными напряжениями.

Выходной цифровой код N (п-разрядный) однозначно соответствует уровню входного напряжения. Код может принимать 2^n значений, то есть АЦП может различать 2^n уровней входного напряжения. Количество разрядов выходного кода п представляет собой важнейшую характеристику АЦП. В момент готовности выходного кода выдается сигнал окончания преобразования RDY, по которому внешнее устройство может читать код N.

Управляется работа АЦП тактовым сигналом CLK, который задает частоту преобразования, то есть частоту выдачи выходных кодов. Предельная тактовая частота — второй важнейший параметр АЦП. В некоторых микросхемах имеется встроенный генератор тактовых сигналов, поэтому к их выводам подключается кварцевый генератор или конденсатор, задающий частоту преобразования. Сигнал CS разрешает работу микросхемы.

Выпускается множество самых разнообразных микросхем АЦП, различающихся скоростью работы (частота преобразования от сотен килогерц до сотен мегагерц), разрядностью (от 6 до 24), допустимыми диапазонами входного сигнала, величинами погрешностей, уровнями питающих напряжений, методами выдачи выходного кода (параллельный или последовательный), другими параметрами. Обычно микросхемы с большим количеством разрядов имеют невысокое быстродействие, а наиболее быстродействующие микросхемы имеют небольшое число разрядов. Область применения любой микросхемы АЦП во многом определяется использованным в ней принципом преобразования, поэтому необходимо знать особенности этих принципов. Для выбора и использования АЦП необходимо пользоваться подробными справочными данными от фирмы-производителя.

В качестве базового элемента любого АЦП используется компаратор напряжения (рис. 7.14), который сравнивает два входных аналоговых напряжения и, в зависимости от результата сравнения, выдает выходной цифровой сигнал — нуль или единицу. Компаратор работает с большим диапазоном входных напряжений и имеет высокое быстродействие (задержка порядка единиц наносекунд).

Существует два основных принципа построения АЦП: последовательный и параллельный.

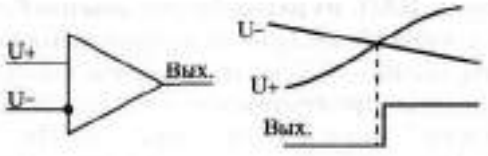


Рис. 7.14. Компаратор напряжения

В последовательном АЦП входное напряжение последовательно сравнивается одним единственным компаратором с несколькими эталонными уровнями напряжения, и в зависимости от результатов этого сравнения формируется выходной код. Наибольшее распространение получили АЦП на основе так называемого регистра последовательных приближений (рис. 7.15).

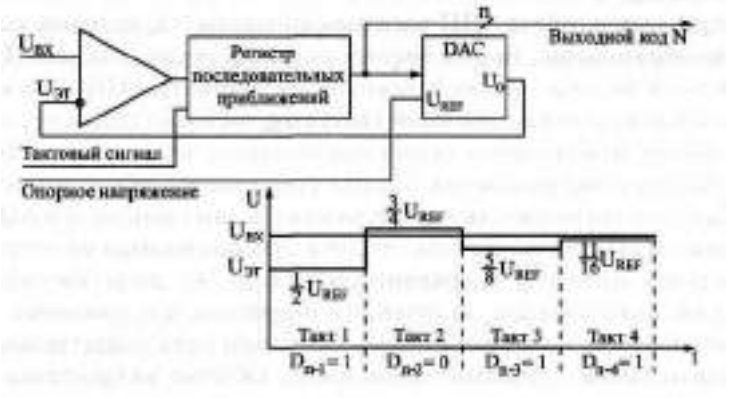


Рис. 7.15. АЦП последовательного типа

Входное напряжение подается на вход компаратора, на другой вход которого подается эталонное напряжение, ступенчато изменяющееся во времени. Выходной сигнал компаратора подается на вход регистра последовательных приближений, тактируемого внешним тактовым сигналом. Выходной код регистра последовательных приближений поступает на ЦАП, которое из опорного напряжения формирует меняющееся эталонное напряжение.

Регистр последовательных приближений работает так, что в зависимости от результата предыдущего сравнения выбирается следующий уровень эталонного напряжения по следующему алгоритму:

- В первом такте входной сигнал сравнивается с половиной опорного напряжения.
- Если входной сигнал меньше половины опорного напряжения, то на следующем такте он сравнивается с четвертью опорного напряже-

ния (то есть половина опорного напряжения уменьшается на четверть). Одновременно в регистр последовательных приближений записывается старший разряд выходного кода, равный нулю.

- Если же входной сигнал больше половины опорного напряжения, то на втором такте он сравнивается с 3/4 опорного напряжения (то есть половина увеличивается на четверть). Одновременно в регистр последовательных приближений записывается старший разряд выходного кода, равный единице.
- Затем эта последовательность сравнений повторяется нужное число раз с уменьшением на каждом такте вдвое ступени изменения эталонного напряжения (на третьем такте — 1/8 опорного напряжения, на четвертом - 1/16 и т. д.). В результате опорное напряжение в каждом такте приближается к входному напряжению. Всего преобразование занимает п тактов. В последнем такте вычисляется младший разряд.

Понятно, что процесс этот довольно медленный, требует нескольких тактов, причем в течение каждого такта должны успеть сработать компаратор, регистр последовательных приближений и ЦАП с выходом по напряжению. Поэтому последовательные АЦП довольно медленные, имеют сравнительно большое время преобразования и малую частоту преобразования.

Второй тип АЦП - АЦП параллельного типа - работает по более простому принципу. Все разряды выходного кода вычисляются в них одновременно (параллельно), поэтому они гораздо быстрее, чем последовательные АЦП. Правда, они требуют применения большого количества компараторов (2^n-1), что вызывает чисто технологические трудности при большом количестве разрядов (например, при 12-разрядном АЦП требуется 4095 компараторов).

Схема такого АЦП (рис. 7.16) включает в себя резистивный делитель из 2^n одинаковых резисторов, который делит опорное напряжение на (2^n-1) уровней.

Входное напряжение сравнивается с помощью компараторов с уровнями, формируемыми делителем напряжения. Выходные сигналы компараторов с помощью шифратора преобразуются в n-разрядный двоичный код. Шифратор выдает на выход номер последнего из сработавших (то есть выдавших сигнал логической единицы) компараторов. Например, в случае 3-разрядного АЦП (на рисунке) при величине входного напряжения от 0 до 1/8 опорного напряжения выходной код будет 000, при входном напряжении от 1/8 до 2/8 опорного напряжения сработает первый компаратор, что даст выходной код 001, при входном напряжении от 2/8 до 3/8 опорного напряжения сработают компараторы 1 и 2, что даст выходной код 010, и т. д. Процесс преобразования происходит в па-

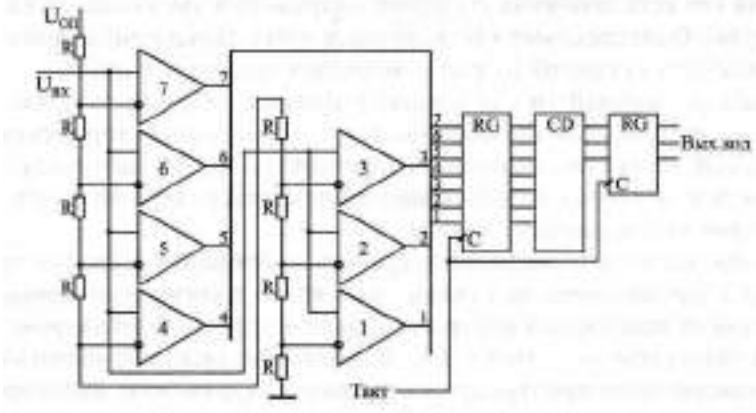


Рис. 7.16. 3-разрядный АЦП параллельного типа

раллельном АЦП очень быстро, поэтому частота преобразования может достигать сотен мегагерц.

Для повышения быстродействия в параллельном АЦП иногда применяется конвейерный принцип: выходной код компараторов записывается в (2^n-1)-разрядный параллельный регистр, показанный на рис. 7.16. Выходной код шифратора также записывается в n-разрядный параллельный регистр. Оба регистра в этом случае тактируются одним и тем же тактовым сигналом. Это снижает требования к быстродействию компараторов и шифратора. Правда, выходной код АЦП задерживается из-за таких регистров на два периода таковой частоты.

Громоздкость структуры параллельного АЦП приводит к тому, что в некоторых АЦП применяется смешанный параллельно-последовательный принцип. Это несколько снижает быстродействие подобного АЦП по сравнению с обычным параллельным АЦП, но зато позволяет получить большое число разрядов, не увеличивая количество компараторов до (2^n-1).

Для того чтобы АЦП любого типа работал с использованием всех своих возможностей, необходимо обеспечить согласование диапазона изменения входного аналогового сигнала с допустимым диапазоном (динамическим диапазоном) входного напряжения АЦП.

На рис. 7.17 показано четыре возможных случая соотношения динамического диапазона АЦП (от 0 до U_{REF} или от U_{REF1} до U_{REF2}) и входного сигнала. В случаях *а* и *б* входной сигнал меньше динамического диапазона, поэтому АЦП будет работать правильно, но не будет использовать всех своих возможностей. В случае *в* входной сигнал слишком большой, поэтому часть его значений не будет преобразова-

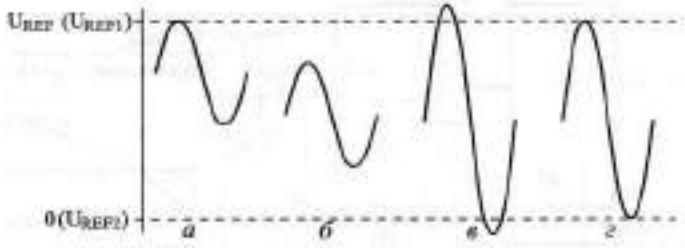


Рис. 7.17. Соотношение входного сигнала и динамического диапазона АЦП

на. Только в случае *г* АЦП действительно будет работать как n-разрядный и будет преобразовывать все значения входного сигнала. Для согласования входного сигнала с динамическим диапазоном АЦП можно применять усилители, аттенюаторы, схемы сдвига. В некоторых случаях согласование может быть достигнуто простым выбором величин опорных напряжений.

Иногда бывает необходимо уменьшить количество разрядов АЦП. В этом случае нужное количество младших разрядов выходного кода микросхемы просто не используется. На рис. 7.18 показано использование 10-разрядного АЦП в качестве 8-разрядного.

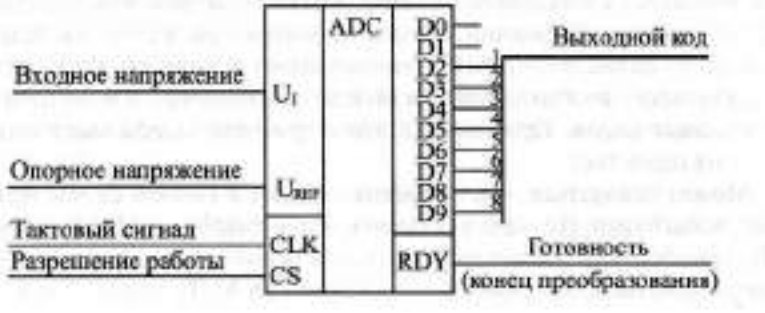


Рис. 7.18. Уменьшение количества разрядов выходного кода АЦП

Обратная задача — увеличение разрядности АЦП — встречается чаще. Существует ряд типичных схемотехнических решений по объединению нескольких микросхем АЦП для увеличения количества разрядов выходного кода, но большинство этих решений требует сложных расчетов результирующих погрешностей преобразования и применения аналоговых узлов. Мы не будем их здесь рассматривать. Отметим только, что при возникновении задачи увеличения разрядности надо прежде все-

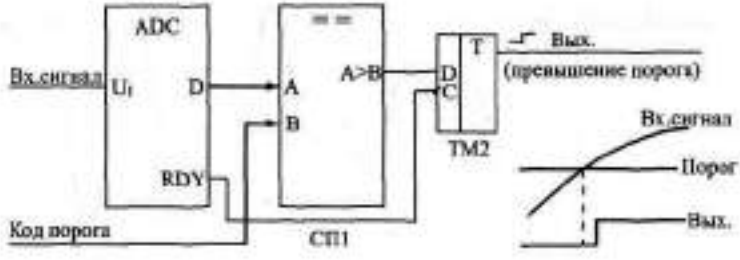


Рис. 7.19. Фиксатор превышения входным сигналом установленного порога

го попытаться найти микросхему с нужным количеством разрядов, и только потом рассматривать возможности объединения нескольких микросхем АЦП.

Рассмотрим несколько типичных схем включения АЦП, используемых в аналого-цифровых системах.

Первая схема (рис. 7.19) предназначена для фиксации момента превышения входным аналоговым сигналом заданного порогового напряжения. Схема вырабатывает выходной сигнал (положительный фронт) тогда, когда входной аналоговый сигнал становится больше установленного уровня, причем уровень этот задается цифровым кодом порога. Код порога сравнивается с выходными кодами АЦП с помощью микросхемы компаратора кодов. Выходной сигнал компаратора кодов записывается в триггер по сигналу **RDY** с АЦП, что позволяет исключить влияние коротких импульсов, возникающих на выходе компаратора в момент изменения входных кодов. Применение этого триггера задерживает выходной сигнал на один такт.

Может показаться, что применение АЦП в данном случае не оправданно, избыточно. Но надо учитывать, что в аналого-цифровых системах АЦП, преобразующий входной сигнал в последовательность кодов, как правило, уже есть, поэтому дополнительного АЦП не требуется, достаточно только включить компаратор кодов и триггер.

АЦП также применяется в схемах вычисления амплитуды входного аналогового сигнала. Для такого вычисления можно использовать уже рассмотренную схему вычислителя экстремального значения входного кода (см. рис. 4.26). В качестве источника последовательности входных кодов в данном случае выступает АЦП (рис. 7.20).

В регистр со входом разрешения записи записывается код с выхода **АЦП** по сигналу **RDY** в том случае, если текущее значение кода больше значения кода, записанного ранее в регистр. В результате уже после одного периода входного сигнала в регистре будет код амплитуды входного

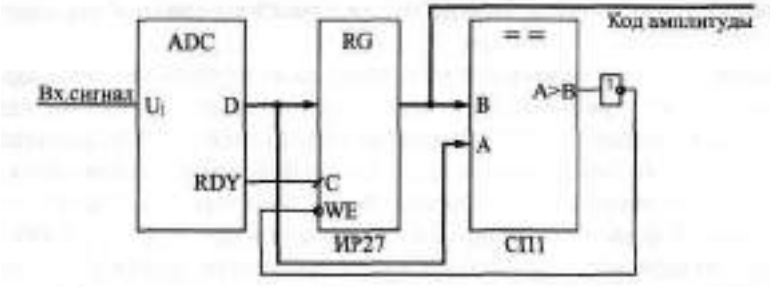


Рис. 7.20. Вычислитель амплитуды аналогового сигнала

сигнала. За период преобразования АЦП должны успеть сработать компаратор кодов и регистр.

Если такой вычислитель амплитуды входного сигнала используется в составе сложной аналого-цифровой системы, в которой уже присутствует АЦП, непрерывно преобразующий входной сигнал в коды, то дополнительно требуются только цифровые микросхемы: компаратор кодов и регистр.

Наиболее часто встречающееся использование АЦП — это преобразование входного сигнала в поток кодов, причем коды эти обычно записываются в буферную память. В данном случае наиболее подходящим является однонаправленный буфер с периодическим режимом работы. То есть сначала в буферную память заносится массив кодов выборок входного сигнала, а затем этот массив читается для дальнейшей обработки. Именно так, например, строится цифровой осциллограф, предназначенный для наблюдения аналоговых сигналов на экране.

Схема включения АЦП в этом случае показана на рис. 7.21. В качестве строба записи в буферную память используется сигнал RDY с АЦП.

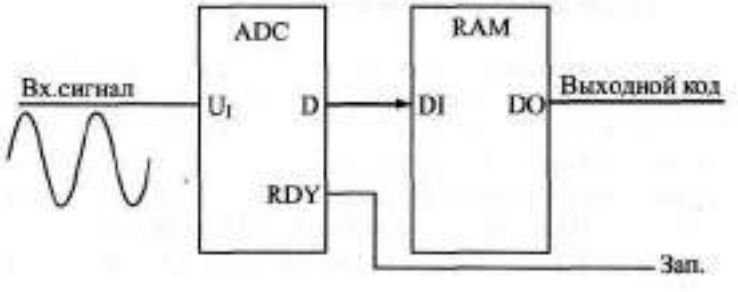


Рис. 7.21. Включение буферной памяти для запоминания кодов с выходов АЦП

Подробнее организацию буфера мы уже рассматривали в предыдущей главе.

Конечно, в реальных аналого-цифровых устройствах все гораздо сложнее, в них требуются схемы синхронизации процесса записи со входным сигналом, схемы предварительной обработки аналогового сигнала, но суть остается той же — буферная память, записывающая последовательность кодов с выхода АЦП. Чем больше объем памяти, тем больший фрагмент входного аналогового сигнала она может запомнить. Например, если память имеет организацию 64Kx8 и работает с 8-разрядным АЦП, то при частоте преобразования АЦП 10 МГц буфер сможет хранить в себе фрагмент аналогового сигнала длительностью 6,5536 мс.

Наконец, последняя схема, которую мы рассмотрим (рис. 7.22), позволяет вдвое повысить быстродействие АЦП, точнее, поднять вдвое частоту записи кодов выборок входного сигнала в буферную память.

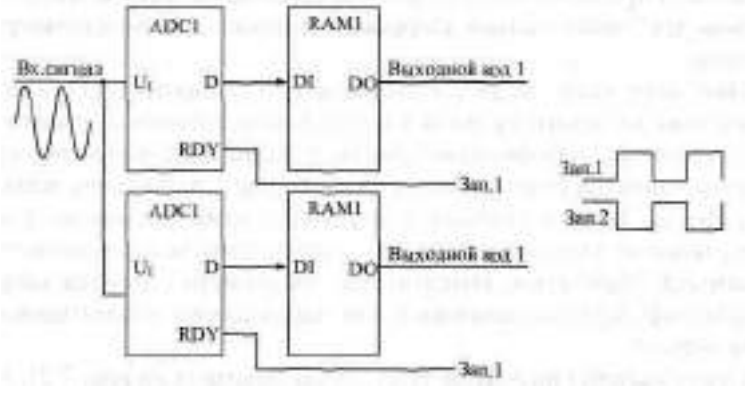


Рис. 7.22. Увеличение вдвое частоты преобразования входного сигнала с помощью двух АЦП с буферами

Идея схемы очень проста: используется два АЦП и два буфера, которые работают по очереди, например, четные выборки входного сигнала обрабатывает один АЦП со своим буфером, а нечетные — другой АЦП со своим буфером. В результате запоминание кодов входного сигнала осуществляется с частотой вдвое больше частоты преобразования каждого из АЦП. Например, если каждый АЦП и каждый буфер работают с частотой 10 МГц, то результирующая частота преобразования составит 20 МГц.

Тактовые сигналы АЦП и сигналы RDY на выходах АЦП должны быть сдвинуты один относительно другого на половину периода тактово-

го сигнала. Чтение зарегистрированных кодов из обоих буферов также должно быть организовано по очереди: первый код читается из первого буфера, второй — из второго, третий — опять из первого, четвертый — из второго и т. д. Объем обоих буферов в данном случае складывается. Например, при организации каждого буфера 64Kx8 результирующий буфер будет иметь организацию 128Kx8.

Пользуясь этим же принципом, можно повысить частоту обработки входного сигнала с помощью АЦП не только вдвое, но и втрое, в четыре раза и т. д. Необходимо только согласовать во времени работу соответственно трех, четырех и т. д. АЦП, у каждого из которых должна быть своя буферная память.

Помимо упомянутых здесь АЦП последовательно и параллельного типов существуют еще и АЦП с промежуточным преобразованием. В них входной аналоговый сигнал с помощью аналогового интегратора преобразуется во временной интервал между цифровыми импульсами или в частоту следования цифровых импульсов. Выходной цифровой код, соответствующий входному аналоговому сигналу формируется в результате измерения длительности временного интервала или частоты следования импульсов (рис. 7.23). Если используется выходная частота, то такой АЦП называется "преобразователем напряжение—частота" (ПНЧ).

Рис. 7.23. АЦП с промежуточным преобразованием

Такой подход позволяет с помощью сравнительно простых аппаратных средств получить высокую точность преобразования, не зависящую от многих параметров используемых компонентов и от характеристик окружающей среды. Измерение временных интервалов и частоты следования импульсов осуществляется простейшими цифровыми схемами, примеры которых приведены в главе 5. Измерения эти могут осуществляться с высокой точностью вследствие того, что существует очень хороший временной эталон — кварцевый генератор. Отметим, что достоинством ПНЧ является также возможность простой передачи его выходного цифрового сигнала на большие расстояния.

В конце главы надо еще раз отметить, что приведенные здесь схемы сильно упрощены. Для их практической реализации необходимо знание не только цифровой схемотехники, но и аналоговой и аналого-цифровой схемотехники, а также знание особенностей конкретных микросхем ЦАП и АЦП, что не является предметом данной книги. Однако рассмотренные ключевые принципы использования ЦАП и АЦП и их совместного включения с цифровыми схемами будут полезны любому разработчику.

Глава 8. Примеры разработки цифровых устройств

Лекция 14. Разработка простых цифровых устройств

В лекции подробно рассматриваются примеры разработки простых цифровых устройств — клавиатуры и вычислителя **контрольной суммы, начиная от анализа функций устройств и выделения основных узлов до проектирования принципиальных схем узлов и устройства в целом.**

Ключевые слова: системотехника, клавиатура, клавиша, защита от дребезга контактов, разработка "с конца", флаг нажатия, коммутационная матрица, опрос клавиатуры, темп нажатий, контрольная сумма, CRC, вероятность выявления ошибок, деление по модулю 2, образующий полином, табличное вычисление контрольной суммы, вычисление контрольной суммы в последовательном потоке.

В предыдущих главах были рассмотрены базовые элементы цифровой схемотехники и простейшие приемы проектирования узлов на их основе. Но для разработки сложных устройств и систем всех этих знаний порой оказывается недостаточно.

Чтобы создать сложное устройство, необходимо еще владеть приемами системотехники, то есть уметь на основании анализа функций, которые должно выполнять устройство в целом, спроектировать его структуру, сформулировать принципы взаимодействия узлов, четко определить все задачи, которые должен решать каждый из узлов, выработать требования к отдельным узлам. Возможно, в результате всех этих шагов будет изменена сама первоначальная задача, будут переформулированы требования к создаваемому устройству, к его месту в системе, к принципам его взаимодействия с другими устройствами. И только потом, после всей этой предварительной работы уже можно переходить к разработке узлов, собственно к схемотехнике.

Приемы системотехники сформулировать, формализовать, описать, даже перечислить гораздо сложнее, чем приемы схемотехники. Да и пользы от такой формализации зачастую немного. Проектирование сложного, надежно работающего цифрового устройства с минимальными аппаратными затратами сродни искусству и требует от разработчика определенных способностей, даже таланта. Научить этому практически не-

возможно. Более того, попытка научить системотехнике может даже принести вред, так как ограничит творческие способности разработчика несколькими жесткими стандартными алгоритмами.

Однако можно показать несколько простейших примеров разработки, продемонстрировать последовательность шагов, которые необходимы при проектировании, которые могут встретиться в процессе создания устройств некоторых распространенных типов. Исходя из этих примеров, разработчик может в дальнейшем попробовать по аналогии создать что-то свое, более или менее совершенное, но обязательно работоспособное.

Поэтому в данной главе как раз и будут подробно рассмотрены несколько простых примеров проектирования сравнительно сложных устройств на всех этапах: от анализа решаемой задачи до создания полной принципиальной схемы. Примеры эти, конечно, не отражают и малой доли всех реально встречающихся задач, но относятся к различным классам цифровых устройств.

8.1. Разработка клавиатуры

Различные клавиатуры с большим количеством клавиш (кнопок) широко используются в цифровых системах: в компьютерах, контроллерах, измерительных приборах, в бытовой технике. Основная задача любой клавиатуры довольно проста: она должна при любом нажатии на клавишу выдавать код номера этой клавиши и сигнал флага нажатия клавиши (строб этого кода). Получив этот сигнал флага, внешнее устройство читает код нажатой клавиши и предпринимает требуемые действия.

Главная задача при проектировании клавиатуры состоит в минимизации аппаратных затрат и в обеспечении надежного срабатывания в любой ситуации. Существует масса схемотехнических решений этой задачи — от примитивных до сложнейших. Клавиатуры могут быть механическими, квазисенсорными или сенсорными, клавиатуры могут иметь жесткую логику работы или быть интеллектуальными, даже допускать перепрограммирование. Мы будем в качестве примера рассматривать самую простую механическую клавиатуру с жесткой логикой работы.

Количество клавиш полноразмерной клавиатуры компьютера превышает сотню, поэтому мы будем проектировать клавиатуру на максимальное количество клавиш, равное 128. Естественно, клавиатура должна иметь защиту от дребезга механических контактов и должна корректно обрабатывать ситуацию одновременного нажатия нескольких клавиш. Примем, например, что при одновременном нажатии нескольких клавиш клавиатура должна выдавать код только одной из них. Примем также, что максимально возможный темп нажатия клавиш на клавиатуре не должен превышать 20 нажатий в секунду (это довольно много). Таким образом,

основные требования к проектируемому устройству сформулированы. Начнем разработку.

Очень часто удобным и эффективным приемом является начало разработки устройства "с конца". То есть проектирование начинается исходя из требуемого результата, из тех сигналов, которые устройство должно выдавать вовне и принимать извне. И только в конце проектирования разрабатывается та часть устройства, которая выполняет требуемую функцию. Такой подход гарантирует, что разработанное устройство не будет чрезмерно избыточным, не будет делать ничего лишнего, а также то, что оно корректно будет взаимодействовать с другими устройствами и системами. Этот принцип проектирования не универсален, порой выдержать его в течение всего процесса разработки трудно, но попробовать его применить к любому устройству никогда не помешает.

В нашем случае необходимо сначала определиться, что должна выдавать вовне клавиатура. Обычно это задается техническим заданием, но мы примем, что наша клавиатура должна выдавать 7-разрядный двоичный номер нажатой клавиши (так как $2^7 = 128$) и сопровождать его положительным сигналом флага нажатия. Сигнал флага и код клавиши должны сохраняться до тех пор, пока нажата клавиша. За это время (несколько миллисекунд) внешнее устройство должно успеть проанализировать сигнал флага и прочитать выходной код клавиатуры. Обычно данное требование не слишком жесткое.

Альтернативное решение — сохранение кода нажатой клавиши и сигнала флага до момента чтения выходного кода внешним устройством — конечно же, снижает требование к быстродействию читающего внешнего устройства, однако оно может привести к тому, что некоторые нажатия клавиш останутся без реакции, не будут обработаны.

Также необходимо определиться, как клавиатура будет вести себя при одновременном нажатии нескольких клавиш. Наиболее сложные, интеллектуальные клавиатуры выдают последовательно коды всех нажатых клавиш, запоминая их в буферной памяти. Но мы примем, что клавиатура должна выдавать только код одной из одновременно нажатых клавиш (первой по установленному порядку). Нажатия всех остальных клавиш одновременно с данной просто игнорируются.

При проектировании механической клавиатуры важно решить, как будет обрабатываться неизбежно присутствующий дребезг механических контактов клавиш. Его можно обрабатывать как внутри клавиатуры, так и вне ее (то есть перенести эту функцию на внешнее устройство). Оба эти подхода имеют свои преимущества. Но наша клавиатура будет обрабатывать дребезг контактов самостоятельно. Принцип обработки выбираем очень простой: первое зафиксированное замыкание контактов клавиши считается началом нажатия, а конец нажатия опре-

деляется тогда, когда контакты будут разомкнуты в течение заданного интервала времени.

В результате временная диаграмма работы разрабатываемой клавиатуры может быть упрощенно представлена в виде рис. 8.1. Здесь сигнал флага начинается при фиксации единичного сигнала с клавиши (это может быть как во время дребезга, так и после его окончания). После выставления флага фиксируется выходной код клавиши. После отпускания клавиши (нулевой сигнал), через время задержки $t_{зад}$ снимается сигнал флага. Время задержки должно быть заведомо больше времени дребезга контактов. Выходной код может сохраняться после отпускания клавиши до следующего нажатия, а может и сниматься.

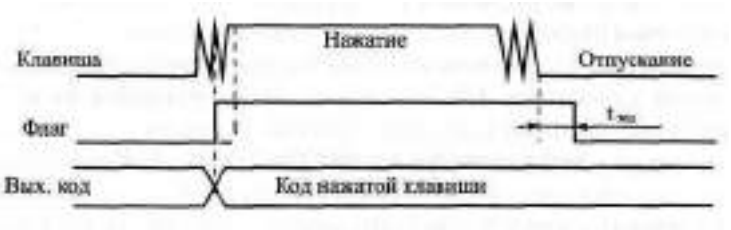


Рис. 8.1. Временная диаграмма работы клавиатуры

Дальнейшая разработка невозможна без выбора принципа преобразования сигналов от нажатия клавиш в код номера нажатой клавиши.

Простейшим путем построения подобного преобразователя является использование приоритетных шифраторов (рис. 8.2).

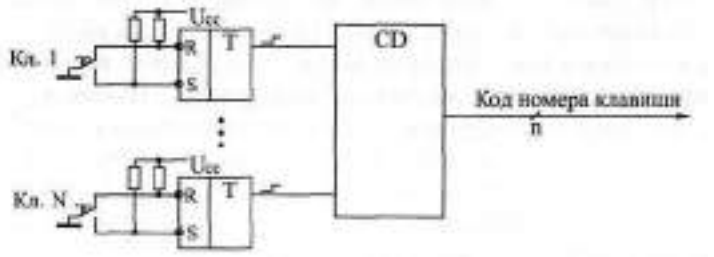


Рис. 8.2. Простейший преобразователь для клавиатуры

Каждая клавиша дает свой логический сигнал, сигналы от всех клавиш преобразуются шифратором в код номера клавиши. Однако такой простейший подход хорош только при небольшом количестве клавиш (до 8 или до 16), так как при большом количестве входов приоритетный шифратор получается довольно сложным. При малом количестве клавиш дребезг контактов обычно устраняется отдельно для каждой клавиши

с помощью RS-триггера (как это показано на рисунке). Это решение простое, но требующее больших аппаратурных затрат.

Другим путем построения преобразователя является использование так называемой коммутационной матрицы, состояние которой периодически опрашивается с частотой тактового генератора. Коммутационная матрица представляет собой две группы пересекающихся проводников (строки и столбцы), во всех точках пересечения которых находятся клавиши. В данном случае каждая клавиша не формирует своего отдельного логического сигнала, а только коммутирует (соединяет) одну из строк матрицы с одним из ее столбцов.

Наиболее универсальная схема преобразователя, легко наращиваемая и достаточно простая, приведена на рис. 8.3.

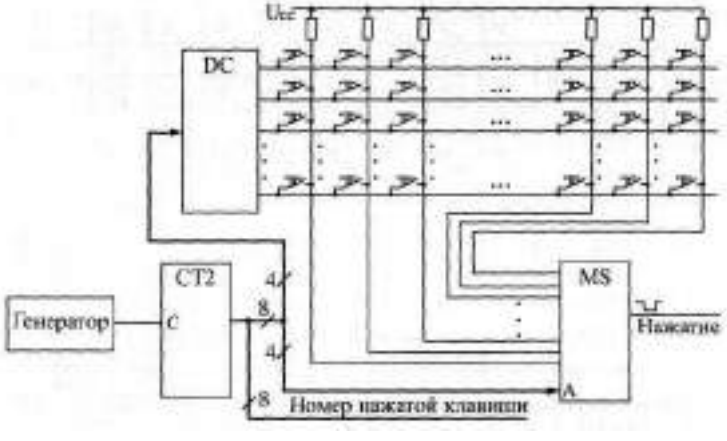


Рис. 8.3. Преобразователь с опросом всех клавиш

Для опроса коммутационной матрицы применяется счетчик, тактируемый генератором. Старшие разряды счетчика используются для выбора одной из строк матрицы с помощью дешифратора (на выбранную строку поступает сигнал логического нуля, на невыбранную — сигнал логической единицы). Младшие разряды счетчика используются для опроса столбцов матрицы с помощью мультиплексора. Сигнал с опрашиваемого столбца подается на выход мультиплексора. Признаком нажатия клавиши является нулевой сигнал на выходе мультиплексора. В этот момент на выходах счетчика присутствует код номера нажатой клавиши. Такая схема легко позволяет строить клавиатуры на большое количество клавиш (до 256, как на рисунке, и даже больше), однако она требует довольно большого времени для полного опроса клавиатуры (так как количество тактов опроса равно полному количеству клавиш).

Совмещение двух рассмотренных подходов позволяет создавать достаточно большие клавиатуры с малыми аппаратурными затратами и малым временем опроса.

При таком комбинированном методе (рис. 8.4) также используется коммутационная матрица с клавишами на всех пересечениях строк и столбцов, но опрашиваются не все клавиши по очереди, а только строки (или столбцы) матрицы. Для опроса, как и в предыдущем случае, применяются генератор, счетчик и дешифратор. Положение же нажатой клавиши в строке (или в столбце) определяется с помощью шифратора. Код нажатой клавиши образуется из выходного кода счетчика (старшие разряды) и кода с выхода шифратора (младшие разряды).

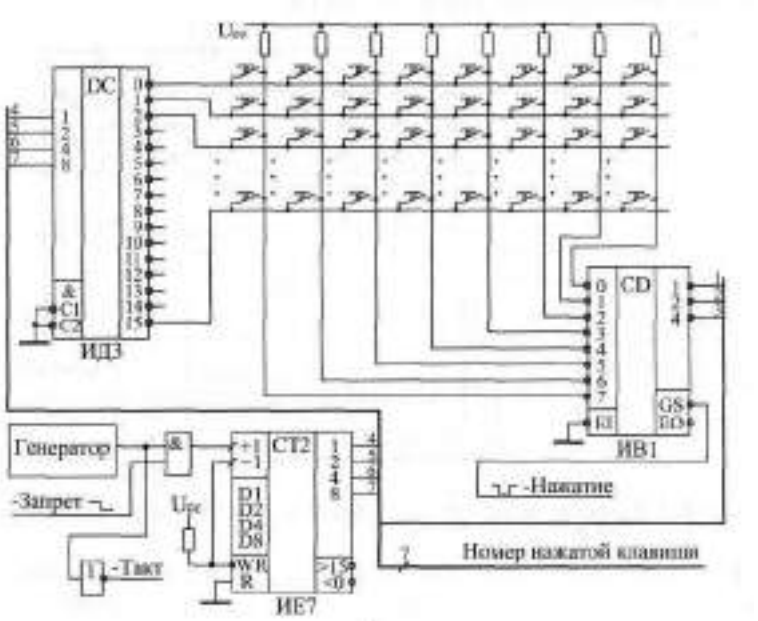


Рис. 8.4. Преобразователь с опросом строк клавиш

В нашем случае клавиатура имеет 128 клавиш, то есть коммутационная матрица должна состоять из 16 строк, опрашиваемых дешифратором 4-16 (ИД3), и 8 столбцов, сигналы с которых обрабатываются шифратором 8-3 (ИВ1). Счетчик должен иметь 4 разряда (ИЕ7). Эти 4 разряда и 3 разряда с выхода шифратора дадут 7-разрядный номер нажатой клавиши. Полный цикл опроса клавиатуры будет занимать 16 тактов генератора (по числу строк). Признаком нажатия одной из клавиш будет отрицательный сигнал на выходе -GS шифратора. Если нажато несколько кла-

виш в разных строках, то обрабатываться будет та клавиша, строка которой будет опрошена первой. Если нажато несколько клавиш в одной строке, то шифратор выдаст код клавиши, соответствующей большему номеру входа. Надо также учитывать, что шифратор ИВ1 выдает инверсный номер входа, на который пришел нулевой сигнал, - эта особенность может потребовать применения трех выходных инверторов (на рисунке не показаны).

Оценим, какой должна быть частота тактового генератора. Мы приняли, что максимальная скорость нажатия равна 20 раз в секунду. Значит, за 1/20 секунды надо успеть опросить всю клавиатуру, то есть все 16 строк. Таким образом, минимально допустимая тактовая частота составляет 16 • 20 = 320 Гц. Но надо заложить и запас на обработку дребезга контактов. Поэтому примем тактовую частоту опроса равной 400 Гц. Она может быть и больше, но чрезмерно увеличивать ее (например, выше 1 кГц) не стоит, так как при быстром переключении микросхем увеличивается потребляемый схемой ток. Понятно, что генератор должен быть не кварцевым, так как кварцевые резонаторы на низкие частоты не выпускаются, а делитель частоты резко усложнит схему. К тому же точная выдержка частоты генератора в данном случае совершенно не нужна.

Выходной сигнал "-Нажатие", конечно же, будет иметь короткие паразитные импульсы. Во-первых, они будут возникать из-за дребезга контактов нажатой в данный момент клавиши. Во-вторых, они могут появляться из-за переходных процессов при переключении счетчика и дешифратора. Эти паразитные импульсы надо исключать.

Чтобы исключить действие паразитных импульсов из-за переходных процессов при переключении счетчика и дешифратора достаточно применить стробирование или тактирование сигнала "-Нажатие" в середине каждого тактового интервала. Для этого из схемы преобразователя надо вывести сигнал "-Такт".

Исключение коротких выходных импульсов из-за дребезга контактов клавиш сложнее. Прежде всего, на время нажатия клавиши целесообразно остановить опрос строк с помощью сигнала "-Запрет". Затем надо обработать сигнал "Нажатие" по принципу, показанному на рис. 8.1. Будем считать, что при дребезге контактов длительность кратковременного размыкания не превышает периода тактового генератора (2,5 мс при тактовой частоте 400 Гц). Тогда задержка окончания сигнала флага нажатия (см. рис. 8.1) должна быть не менее одного периода тактового сигнала. Для выработки задержки можно использовать цепочку триггеров, тактируемых сигналом "-Такт".

Схема выработки выходных сигналов клавиатуры приведена на рис. 8.5.

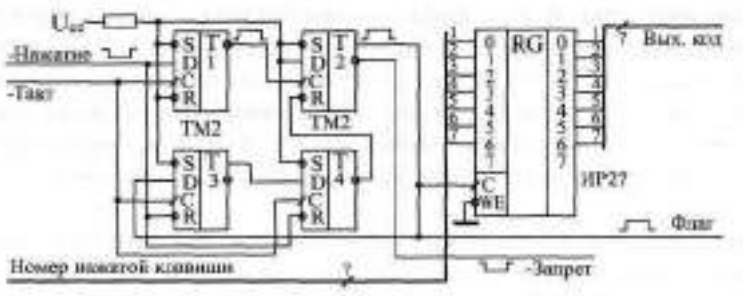


Рис. 8.5. Схема выработки выходных сигналов клавиатуры

Триггер Т1 перебрасывается в единицу (на инверсном выходе) тогда, когда в середине тактового интервала (по положительному фронту сигнала "-Такт") сигнал "-Нажатие" нулевой. Своим выходным сигналом триггер Т1 перебрасывает в единицу триггер Т2, который уже никак не связан с сигналом "-Нажатие", не реагирует ни на какой дребезг этого сигнала. Выходной сигнал триггера Т2 используется в качестве сигнала флага нажатия клавиатуры. Инверсный сигнал с выхода триггера Т2 используется в качестве сигнала "-Запрет", останавливающего опрос строк клавиатуры.

Цепочка триггеров Т3 и Т4, тактируемая сигналом "Такт", служит для задержки снятия сигнала флага после отпускания клавиши (когда сигнал "-Нажатие" становится равным единице). После установки флага в единицу, сигнал флага начинает записываться по фронту сигнала "-Такт" в триггеры Т3 и Т4, но только в том случае, когда сигнал "-Нажатие" установлен в единицу. В результате на инверсном выходе триггера Т4 появится сигнал логического нуля при нахождении сигнала "-Нажатие" в единице в моменты двух последовательных положительных фронтов сигнала "-Такт". Сигнал с выхода Т4 сбрасывает сигнал флага в нуль, после чего вся схема переходит в исходное состояние и ждет следующего нулевого сигнала "-Нажатие".

Если кратковременное размыкание при дребезге контактов клавиш длится более 2,5 мс, то можно увеличить количество триггеров в последовательной цепочке (Т3 и Т4), что приведет к увеличению задержки снятия сигнала флага на целое число тактов генератора.

Таким образом, схема полностью разработана. Отметим, что низкая тактовая частота работы схемы позволяет нам не рассчитывать задержек микросхем, то есть использовать только первый уровень представления, логическую модель. А эффекты, связанные с переходными процессами при переключении микросхем, мы устранили, обеспечив временной сдвиг на половину периода генератора между тактовыми сигналами схемы преобразователя (рис. 8.4) и схемы выработки выходных сигналов (рис. 8.5). Все резисторы, примененные в схеме, должны быть номиналом около 1 кОм.

8.2. Разработка вычислителя контрольной суммы

Различные контрольные суммы широко применяются в цифровых устройствах и системах для контроля правильности хранения или передачи массивов информации. Суть этого метода контроля проста: к хранимому или передаваемому информационному массиву присоединяется небольшой контрольный код (обычно от 1 разряда до 32 разрядов), в котором в свернутом виде содержится информация обо всем массиве. При чтении или получении этого массива еще раз вычисляется тот же самый контрольный код по тому же самому алгоритму. Если этот вновь вычисленный код равен тому коду, который был присоединен к массиву, то считается, что массив сохранен или передан без ошибок. Логика здесь следующая: контрольный код (он же контрольная сумма) гораздо меньше контролируемого массива, поэтому вероятность искажения контрольной суммы гораздо меньше, чем вероятность искажения массива. Если же исказятся как массив, так и контрольная сумма, то вероятность того, что эти искажения не будут замечены при повторном подсчете контрольной суммы, крайне мала. Существует, правда, вероятность, что массив будет искажен в нескольких местах таким образом, что контрольная сумма от этих искажений никак не изменится, но такая вероятность также обычно мала.

Контрольные суммы применяются при хранении данных в памяти (оперативной и постоянной), при хранении данных на магнитных носителях (дисках, лентах), в локальных и глобальных сетях передачи информации. В случае защиты контрольной суммой хранимой информации можно определить, что данный массив (файл, сектор на диске) испорчен и его нельзя использовать. В случае защиты контрольной суммой передаваемой по сети информации приемник может потребовать от передатчика повторной передачи искаженного массива.

Существует множество способов вычисления контрольной суммы, различающихся степенью сложности вычисления и надежностью выявления ошибок. Но наибольшее распространение получил в настоящее время так называемый "циклический метод контроля по избыточности" или CRC (Cyclic Redundancy Check), при котором применяется циклическая контрольная сумма.

Вычисляется циклическая контрольная сумма следующим образом. Весь массив информации рассматривается как одно N-разрядное двоичное число, где N — количество бит во всех байтах массива. Для вычисления контрольной суммы это N-разрядное число делится на некоторое постоянное число (полином), выбранное специальным образом (но делится не просто, а по модулю 2). Частное от этого деления отбрасывается, а остаток как раз и используется в качестве контрольной суммы.

Мы не будем углубляться в математическое обоснование этого метода. Интересующиеся читатели могут обратиться к специальной литературе. Здесь же мы отметим только, что данный метод выявляет одиночные ошибки в массиве с вероятностью 100%, а любое другое количество ошибок с вероятностью, примерно равной $(1-2^{-n})$, где n — количество разрядов контрольной суммы (это верно только при условии, что N гораздо больше n, что, впрочем, почти всегда выполняется). Например, при n = 8 данная вероятность составит 0, 996, для n = 16 она будет равна 0, 999985, а для n = 32 она будет 0, 9999999997672. Иначе говоря, почти все ошибки будут выявляться.

А теперь кратко поясним, что такое деление по модулю 2. Пусть массив (последовательность бит) имеет следующий вид: 101111001110 (для простоты берем небольшую разрядность). Число, на которое делим (называемое обычно образующим полиномом) возьмем 10011. Как оно выбирается? Оно должно делиться по модулю 2 без остатка только на единицу и само на себя (то есть это должно быть простое число в смысле деления по модулю 2). Разрядность полинома берется на единицу большая, чем требуемая разрядность контрольной суммы (остатка от деления). Так, чтобы получить 8-разрядный остаток (8-разрядную контрольную сумму), надо брать 9-разрядный полином. В нашем случае полином 5-разрядный, следовательно, остаток будет 4-разрядный. Для получения 8-разрядного остатка можно использовать, например, полином 1 0001 1101 или 11D в 16-ричном коде.

Деление по модулю 2 производится точно так же, как и привычное для нас деление "в столбик" (рис. 8.6), но вместо вычитания в данном случае используется поразрядное сложение по модулю 2, то есть каждый результирующий бит представляет собой функцию Исключающее ИЛИ от соответствующих битов слагаемых. Частное от деления нас не интересует, а остаток, равный в нашем примере 1000, и будет циклической контрольной суммой.

Как практически реализовать вычисление этого остатка (контрольной суммы)? Можно сделать это по приведенному здесь принципу деле-

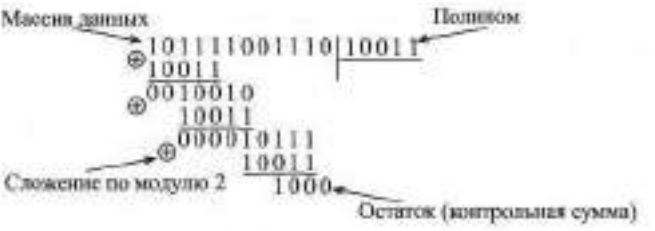


Рис. 8.6. Вычисление циклической контрольной суммы

ния в столбик (аппаратно или программно). Но в любом случае это довольно громоздко и медленно. Ускорить процесс вычисления можно, воспользовавшись табличным методом. Для этого составляется таблица чисел размером 2'хп, где п — разрядность контрольной суммы. Принцип вычисления чисел в таблице очень прост (табл. 8.1).

Таблица 8.1. Табличный метод вычисления циклической контрольной суммы

Адрес в таблице	Данные в таблице (числа)
0	0
1	Остаток от деления числа 1 0000 0000 на полином
2	Остаток от деления числа 10 0000 0000 на полином
3	Остаток от деления числа 11 0000 0000 на полином
4	Остаток от деления числа 100 0000 0000 на полином
5	Остаток от деления числа 101 0000 0000 на полином
255	Остаток от деления числа 1111 1111 0000 0000 на полином

Числа представляют собой остаток от деления по модулю 2 числа с п конечными нулями (в нашем примере п = 8) и с п начальными разрядами, равными номеру числа (его адресу) в таблице. Деление производится на выбранный полином (в нашем случае — 9-разрядный). Таблица вычисляется один раз и хранится на диске или в ПЗУ.

Алгоритм вычисления контрольной суммы с помощью этой таблицы следующий (рассматриваем случай п = 8). Берем первый байт нашего информационного массива. Рассматриваем его как адрес в таблице (номер числа). Берем из таблицы число с полученным номером — получаем остаток 0_1. Берем второй байт массива и складываем его по модулю 2 с остатком 0_1. Полученное число используем как адрес в таблице. По этому адресу выбираем из таблицы остаток 0_2. Берем третий байт массива, складываем его по модулю 2 с остатком 0_2. Используя это число как адрес в таблице, выбираем из нее остаток 0_3 и так продолжаем до последнего байта массива. Естественно, это будет гораздо быстрее, чем вычисление "в столбик".

Реализация этого алгоритма с помощью цифровых схем требует только ПЗУ с организацией $2^n \times n$ (256x8 при 8-разрядной контрольной сумме), n-разрядного регистра и п элементов Исключающее ИЛИ (рис. 8.7). В ПЗУ заносится таблица промежуточных остатков (табл. 8.1),

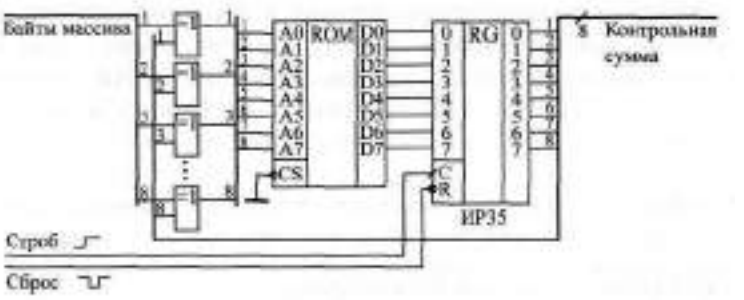


Рис. 8.7. Параллельный вычислитель 8-разрядной циклической контрольной суммы на ПЗУ

на вход схемы подаются один за другим байты массива, сопровождаемые стробом. Адресом ПЗУ служит сумма по модулю 2 входных данных и содержимого выходного регистра, в который по сигналу строба записывается выходной код ПЗУ. Перед началом вычисления состояние регистра обнуляется. После окончания всего массива в регистре образуется циклическая контрольная сумма.

Недостаток данной схемы параллельного вычислителя очевиден: в случае большого числа разрядов контрольной суммы требуется очень большой объем ПЗУ (64Кх16 для 16-разрядной суммы и 4Гх32 для 32-разрядной суммы). Поэтому она применяется сравнительно редко. Зато параллельный вычислитель обладает высоким быстродействием (байты могут поступать с периодом, равным сумме задержки выходного регистра, времени выборки адреса ПЗУ и задержки элемента Исключающее ИЛИ).

Для многоразрядной контрольной суммы чаще применяется другой подход — вычисление в последовательном коде, при котором массив данных поступает на вычислитель последовательно, бит за битом. Последовательный вычислитель контрольной суммы представляет собой сдвиговый регистр с обратными связями от некоторых разрядов через сумматоры по модулю 2 (то есть элементы Исключающее ИЛИ). Полное количество разрядов регистра сдвига должно быть равно разрядности вычисляемой контрольной суммы (или, что то же самое, быть на единицу меньше разрядности используемого полинома). Место включения обратных связей однозначно определяется выбранным полиномом. Это очень похоже на генератор квазислучайной последовательности (см. раздел 4.2.3).

Количество точек включения обратной связи определяется количеством единиц в полиноме (единица в младшем разряде не учитывается), а номера разрядов сдвигового регистра, с которых берутся сигналы обратной связи, определяются номерами единичных разрядов в коде полинома. В отличие от генератора квазислучайного сигнала, в данном случае

необходимо смешать по функции Исключающее ИИ не только сигналы обратной связи, но и входной сигнал данных в последовательном коде.

На рис. 8.8 приведен пример последовательного вычислителя 16-разрядной циклической контрольной суммы при выбранном полиноме 1 0001 0000 0010 0001 или 11021 в 16-ричном коде (рекомендация МККТТ V.41). Так как в коде полинома три единицы (без младшего разряда), необходимо взять три точки включения обратной связи. При этом номера разрядов сдвигового регистра, к которым подключаются обратные связи, определяются положением единичных битов в полиноме. Перед началом работы сдвиговый регистр необходимо сбросить в нуль (сигнал "-Сброс"). Биты массива должны сопровождаться сигналом строба. После окончания массива в регистре будет циклическая контрольная сумма.

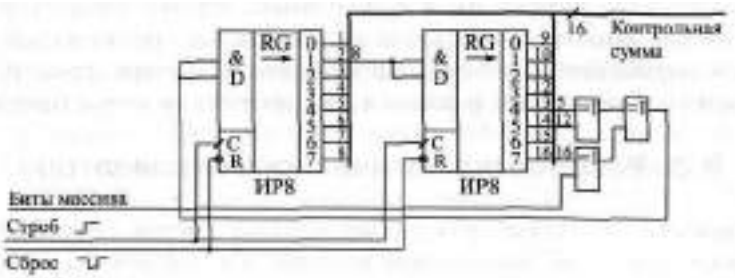


Рис. 8.8. Последовательный вычислитель 16-разрядной циклической контрольной суммы на регистре сдвига

Может показаться, что такой последовательный вычислитель не слишком удобен из-за того, что данные массива должны подаваться на него в последовательном коде. Однако именно в последовательном коде передаются данные в информационных сетях, и в последовательном коде записываются данные на магнитные носители. Поэтому во всех подобных случаях последовательные вычислители идеально подходят.

Период поступления битов массива на последовательный вычислитель не должен быть меньше суммы задержки регистра сдвига и элементов Исключающее ИЛИ. В итоге предельная скорость вычисления циклической контрольной суммы оказывается значительно меньшей, чем в случае параллельного вычислителя. Это также недостаток данного метода вычисления.

Лекция 15. Разработка более сложных цифровых устройств

В лекции подробно рассматриваются примеры разработки сравнительно сложных цифровых устройств — логического анализатора и генератора аналоговых сигналов, начиная от анализа функций устройств и выделения основных узлов до проектирования принципиальных схем узлов и устройства в целом.

Ключевые слова: логический анализатор, динамическая отладка, предпусковая регистрация, входные каналы, запуск анализатора, анализаторы синхронные и асинхронные, мертвое время, управляемый тактовый генератор, схема запуска, синтез сигналов произвольной формы, сигнал помехи, опрос памяти генератора сигналов, коды частоты и амплитуды, разовый и автоматический запуск генерации.

8.3. Разработка логического анализатора

Логический анализатор — это контрольно-измерительный прибор, предназначенный для запоминания (фиксации) и последующего анализа (например, просмотра на экране) временных диаграмм большого количества цифровых сигналов. Логические анализаторы используются при динамической отладке различных цифровых устройств и систем, а также при контроле их работы. Совершенно незаменимы они при разработке и отладке различных микропроцессорных систем, контроллеров, компьютеров, где используется большое количество многоразрядных шин цифровых сигналов. Именно логические анализаторы позволяют разработчику увидеть те временные диаграммы, которые он рисует на бумаге при проектировании своего устройства, причем увидеть их в реальном масштабе времени, посмотреть, как работает устройство на своей нормальной рабочей скорости.

Логический анализатор по своему назначению близок к осциллографу, так как он также позволяет наблюдать на экране временные диаграммы сигналов. Но существуют и существенные отличия логического анализатора от обычного (не цифрового) осциллографа:

- Логический анализатор работает только с цифровыми, то есть двухуровневыми (реже трехуровневыми) сигналами, а осциллограф — с аналоговыми сигналами, имеющими бесконечно большое число разрешенных уровней.
- Логический анализатор имеет большое количество входных линий (обычно от 16 до 64), то есть позволяет одновременно фиксировать

множество входных сигналов, а осциллографы обычно позволяют одновременно увидеть не более четырех входных сигналов.

- Логический анализатор работает в режиме однократного запоминания временных диаграмм (как запоминающий осциллограф). То есть анализатор запоминает состояния входных сигналов в течение заданного времени (называемого окном регистрации), а затем дает возможность анализировать зафиксированные последовательности. Осциллограф же работает обычно в режиме непрерывной развертки, то есть он не запоминает формы входного сигнала и позволяет наблюдать только повторяющиеся, периодические сигналы.
- Логический анализатор предусматривает возможность так называемой предпусковой регистрации. Эта возможность предусматривается и в цифровых осциллографах, но ее нет в аналоговых осциллографах. Рассмотрим подробнее, что такое предпусковая регистрация.

Процесс регистрации входных сигналов (или отображения их на экране в обычном осциллографе) всегда должен быть привязан к какому-то моменту времени, к какому-то внешнему событию, называемому запуском. Иначе разобраться в отображаемых сигналах будет совершенно невозможно. Например, в осциллографах моментом запуска обычно является момент превышения входным исследуемым сигналом установленного порога. Сигналом запуска может служить и специальный внешний синхронизирующий сигнал. В логических анализаторах в качестве запуска обычно используется момент появления на входах заданного уровня или заданной последовательности одного или нескольких входных сигналов.

В обычных осциллографах отображение формы входного сигнала (или входных сигналов) начинается в момент запуска, то есть на экране видно только то, что происходило со входными сигналами *после* момента запуска. Такая регистрация может быть названа послепусковой. Можно также сказать, что точка запуска всегда находится в начале окна регистрации (рис. 8.9).

Рис. 8.9. Послепусковая регистрация в аналоговых осциллографах

В логических анализаторах (и в цифровых осциллографах) существует возможность увидеть и зафиксировать не только то, что было *после* запуска, но еще и то, что происходило в течение определенного времени *до* момента запуска. Именно эта регистрация до момента запуска и называется предпусковой регистрацией. В этом случае точка запуска может находиться и в начале, и в середине, и в конце окна регистрации (рис. 8.10). Понятно, что такая возможность очень удобна, так как, выбирая величину длительности предпусковой регистрации, можно увидеть те события, временная привязка к началу которых затруднена или попросту невозможна. Длительность (глубина) предпусковой регистрации может быть постоянной (например, равной половине длительности окна регистрации) или переменной (то есть задаваться пользователем в пределах от нуля до полной длительности окна регистрации). При переменной глубине предпусковой регистрации точка запуска может располагаться в любой точке окна регистрации — от его начала до конца.

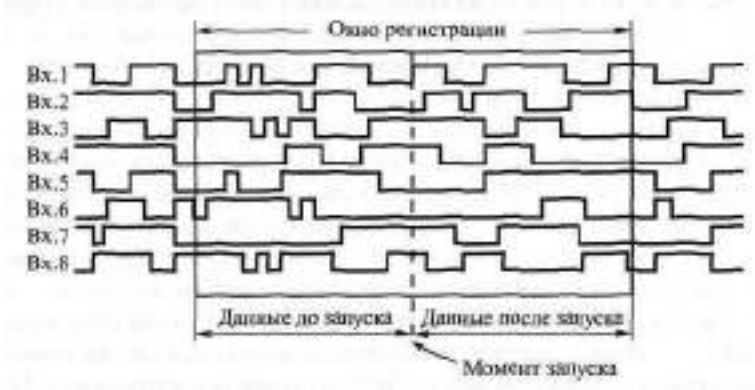


Рис. **8.10.** Предпусковая регистрация в логических анализаторах и цифровых осциллографах

С точки зрения схемотехники, логический анализатор представляет собой быстродействующую буферную оперативную память, работающую в периодическом режиме. Буфер этот однонаправленный: сначала в буферную память с большой тактовой частотой последовательно записываются состояния нескольких входных сигналов, а затем эта информация последовательно читается из буфера. Таким образом, адреса буферной памяти могут перебираться одним и тем же счетчиком как в режиме записи, так и в режиме чтения. Структура таких буферов уже рассматривалась в разделе 6.2.2.

Главные особенности логического анализатора, по сравнению со стандартной структурой информационного буфера на основе оперативной памяти, следующие:

- большое число разрядов шины данных (то есть входных сигналов, каналов регистрации анализатора);
- необходимость обеспечения режима предпусковой регистрации;
- необходимость временной привязки процесса регистрации (записи в память) к состояниям входных сигналов (обеспечение запуска).

Первая из этих особенностей приводит к тому, что данные при чтении приходится считывать не все сразу, а по очереди (особенно при числе разрядов больше 32). Обычно данные требуется читать по 8 или по 16 разрядов. В результате усложняется та часть схемы буферной памяти, которая отвечает за чтение данных.

Вторая особенность требует существенного усложнения схемы счетчика, перебирающего адреса буферной памяти.

Наконец, третья особенность требует усложнения схемы управления работой информационного буфера.

Логические анализаторы делятся на синхронные (или анализаторы логических состояний) и асинхронные (или анализаторы временных диаграмм). Синхронные анализаторы работают от тактового генератора исследуемой схемы и фиксируют только временные сдвиги, кратные его периоду, а следовательно, выявляют только нарушения в логике работы схемы. Асинхронные анализаторы работают от собственного внутреннего тактового генератора, поэтому они позволяют измерять абсолютные значения временных сдвигов между сигналами и могут выявлять ошибки из-за неправильно рассчитанных задержек, из-за емкостных эффектов и т. д. Они обычно делаются гораздо более быстрыми, чем синхронные анализаторы (рассчитываются на предельно возможную частоту регистрации). В идеале, логический анализатор должен обеспечивать оба эти режима работы, то есть работать как от своего внутреннего тактового генератора с разными тактовыми частотами, так и от внешнего тактового сигнала. Иначе говоря, тактовый генератор анализатора должен быть также достаточно сложным.

Сформулируем исходные данные для проектирования логического анализатора. В данном случае нам важно не получить рекордные характеристики, а всего лишь продемонстрировать принципы разработки подобных схем на основе буферной памяти. Пусть количество входных линий анализатора (каналов регистрации) равно 32, количество регистрируемых состояний — 4096, максимальная тактовая частота — 10 МГц, тактовый генератор — внутренний с изменяемой частотой или внешний, запуск — по положительному или отрицательному фронту (синхропереходу) на одной из 8 входных линий, глубина предпусковой регистрации — задается программно. Будем также считать, что данные из памяти читаются порциями по 8 разрядов.

Таким образом, буферная оперативная память анализатора должна иметь объем 128 Кбит при организации 4Кх32. Помимо оперативной па-

мяти, анализатор должен включать в себя счетчик для перебора адресов с количеством разрядов не менее 12. В структуре анализатора должен быть также внутренний тактовый генератор с программно изменяемой частотой и возможностью подключения внешнего тактового сигнала. Наконец, необходимо наличие схемы запуска анализатора, которая будет выбирать одну из 8 входных линий и полярность синхроперехода (положительный или отрицательный фронт).

Память целесообразно выполнить на многоразрядных микросхемах ОЗУ (для снижения количества микросхем). Требования к быстродействию памяти в данном случае не слишком высоки (при максимальной тактовой частоте 10 МГц в течение 100 нс необходимо успеть переключить счетчик адресов и записать входную информацию в ОЗУ). Микросхем памяти, способных обеспечить такую скорость работы, достаточно много.

От счетчика адресов памяти требуется максимальное быстродействие (можно взять, например, микросхемы синхронных счетчиков КР531ИЕ17, которые достаточно легко каскадируются без потери быстродействия). Кроме простого перебора адресов, счетчик должен также обеспечивать предпусковую регистрацию. Остановимся на этом несколько подробнее.

Для того чтобы реализовать предпусковую регистрацию, необходимо обеспечить непрерывную перезапись по кругу содержимого буферной памяти до момента прихода запуска (рис. 8.11).

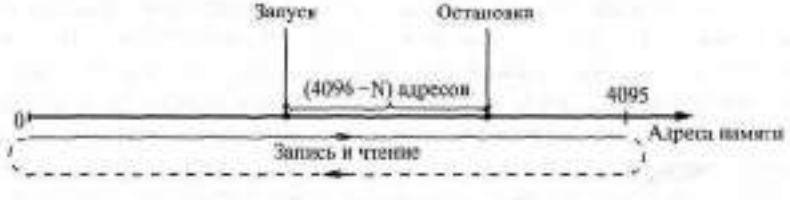


Рис. 8.11. Организация предпусковой регистрации

То есть после записи последнего 4095 адреса надо записывать информацию по нулевому адресу. Если мы выбираем глубину предпусковой регистрации N тактов, то надо остановить регистрацию через (4096-N) тактов после момента прихода запуска. После остановки регистрации надо считывать содержимое памяти, начиная с точки остановки, с перебором адресов в том же самом направлении, что и при регистрации. Проведя 4096 операций чтения содержимого памяти, мы получим информацию о состоянии входных сигналов в течение N тактов до запуска и (4096-N) тактов после запуска, то есть моменту прихода запуска будет соответствовать содержимое адреса памяти, считанного N-ым.

Однако все произойдет именно таким образом только в том случае, если от момента начала регистрации до момента прихода запуска логический анализатор успеет зафиксировать N тактов. Иначе, остановив регистрацию через (4096-N), мы не перепишем всю память, и в части его адресов будет находиться предыдущая информация. Чтобы избежать этого, надо запретить реакцию на запуск в течение N тактов после начала регистрации (выдержать своеобразное "мертвое" время). А что будет, если запуск придет в течение этого самого "мертвого" времени? Если исследуемый процесс — периодический (то есть все входные сигналы повторяются через какое-то время), то анализатор среагирует на следующий запуск после окончания "мертвого" времени. Если же исследуемый процесс - однократный, не повторяющийся, то надо начать процесс регистрации заведомо раньше (на "мертвое" время или больше), чем начнется изучаемый процесс (например, если мы исследуем старт компьютера при включении питания).

В результате счетчики анализатора должны обеспечивать временную диаграмму, показанную на рис. 8.12.

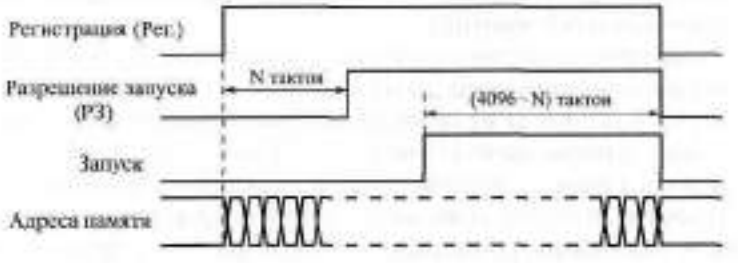


Рис. 8.12. Временная диаграмма работы счетчиков логического анализатора

Адреса памяти начинают перебираться с началом регистрации. В течение N тактов после начала регистрации реакция на запуск запрещается, а затем разрешается. Через (4096-N) тактов после прихода запуска регистрация прекращается.

Отметим, что точно так же может быть реализована предпусковая регистрация в цифровом осциллографе. По сравнению с логическим анализатором, в схему надо будет добавить только один или несколько АЦП и некоторые другие цифро-аналоговые узлы.

Спроектируем схему счетчиков, реализующую приведенную временную диаграмму.

Счетчик, перебирающий адреса памяти, должен быть 12-разрядным, так как $2^{12} = 4096$. Во время регистрации он должен работать в непрерывном режиме, реализуя постоянную перезапись по кругу всей буферной

памяти. На этот же счетчик можно возложить функцию отсчета "мертвого" времени (N тактов). Но этот счетчик не может отсчитывать ещё и (4096-N) тактов после прихода запуска, так как запуск может прийти в любой момент после окончания "мертвого" времени. Для этого понадобится уже другой счетчик, причем также 12-разрядный.

Этот второй счетчик должен начинать работу только после прихода запуска (по сигналу "Разрешение запуска") и должен отсчитывать всего (4096-N) тактов, после чего завершать регистрацию. То есть получается, что логический анализатор начинает регистрацию по внешнему управляющему сигналу, а заканчивает автоматически через (4096-N) тактов после запуска, о чем должен сообщать вовне сигнал флага окончания регистрации.

Таким образом, один 12-разрядный счетчик должен отсчитывать N тактов, а другой 12-разрядный счетчик должен отсчитывать (4096-N) тактов. Проще всего организовать такой режим, если в оба счетчика записать до начала работы код N и задать первому счетчику инверсный режим счета, а второму — прямой режим счета. Перебор адресов памяти первым счетчиком начнется с адреса N и будет происходить на уменьшение (а не на увеличение, как на рис. 8.11), однако для работы буферной памяти это не имеет никакого значения.

Сигнал переноса первого счетчика появится через N тактов после начала регистрации. Этот сигнал должен разрешать ожидание запуска (сигнал РЗ на рис. 8.12). Когда же приходит запуск, то разрешается работа второго счетчика, начинающего считать с кода N на увеличение. В результате сигнал переноса второго счетчика появится через (4096-N) тактов после начала его работы. Этот сигнал должен остановить процесс регистрации.

После окончания регистрации должен начаться процесс чтения из памяти. При этом первый счетчик должен перебирать адреса памяти по стробу чтения в том же направлении, что и при записи (то есть в режиме инверсного счета). 4096 последовательно произведенных циклов чтения позволит перебрать все 4096 адресов памяти, причем на N-ом цикле чтения будет прочитан такт, в котором произошел запуск.

Схема счетчиков логического анализатора, реализующая описанный алгоритм, приведена на рис. 8.13.

Два 12-разрядных счетчика реализованы на шести микросхемах ИЕ17. Первый счетчик (на рисунке вверху) работает в режиме инверсного счета, второй (на рисунке внизу) — в режиме прямого счета. Перед началом работы в оба счетчика по внешнему сигналу "-Зап." записывается код N. Причем четыре младших разряда 12-разрядного кода N равны нулю, а записываются только 8 старших разрядов. Это приводит к тому, что глубина предпусковой регистрации может задаваться с точностью до 16 тактов и принимать значения из ряда: 0, 16, 32, 48, 64. . . . 4080. При записи на входы счетчиков -ЕWR и С поступают отрицательные сигналы,

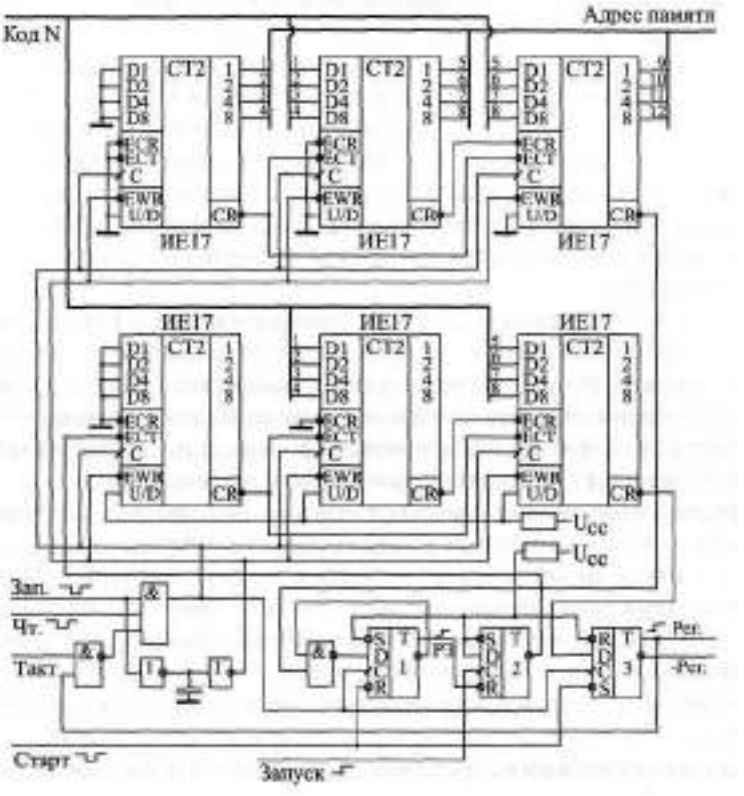


Рис. **8.13.** Схема счетчиков логического анализатора

причем сигнал -EWR задержан относительно сигнала C на двух инверторах и (при необходимости) на конденсаторе. В результате положительный фронт сигнала C приходит тогда, когда сигнал -EWR равен нулю, что и требуется для записи.

На вход C обоих счетчиков могут приходить еще два сигнала: строб чтения из памяти "-Чт." и тактовый сигнал "Такт". Сигнал "Такт" приходит при регистрации (в режиме записи в память), а сигнал "-Чт." поступает при чтении из памяти зарегистрированной информации.

После того как в оба счетчика записан код N, необходимо начать регистрацию по внешнему сигналу "-Старт". Этот сигнал сбрасывает в нуль триггеры 1 и 2 и устанавливает в единицу триггер 3. Выходной сигнал триггера 3 разрешает регистрацию (сигнал "Рег." на рис. 8.12), то есть разрешает прохождение тактовых импульсов на входы счетчиков C. Это приводит к тому, что начинает работу первый счетчик, а вто-

рому счетчику работа запрещается по входу ЕСТ выходным сигналом триггера 2.

Первый счетчик, выходной код которого используется как адрес памяти, отсчитывает N тактов в инверсном режиме и вырабатывает сигнал переноса CR. Этим сигналом перебрасывается в единицу триггер 1. Затем первый счетчик продолжает перебирать адреса памяти по кругу, а запись нулей в триггер 1 запрещается элементом 2И-НЕ на входе триггера. Выходной сигнал триггера 1 (сигнал РЗ на рис. 8.12) разрешает работу триггера 2 и тем самым разрешает реакцию схемы на сигнал запуска "Запуск" (положительный фронт).

После прихода сигнал "Запуск" перебрасывается в единицу триггер 2, разрешая работу второго счетчика. Первый счетчик в это время продолжает считать. Второй счетчик начинает свой счет с кода N, досчитывает в режиме прямого счета до 4096 и своим сигналом переноса –CR перебрасывает в нуль триггер 3. Это приводит к запрету поступления тактовых импульсов на вход С счетчиков и к остановке регистрации.

После этого может начаться чтение записанной информации по стробу "-Чт.", По заднему (положительному) фронту этого сигнала первый счетчик будет перебирать адреса памяти в инверсном режиме. Второй счетчик также будет считать, но это не имеет никакого значения. После 4096 циклов чтения вся информация из памяти будет прочитана, и схема будет готова к новой регистрации.

Перейдем теперь к проектированию других узлов логического анализатора.

Как уже отмечалось, тактовый сигнал анализатора может быть как внутренним (от внутреннего тактового генератора), так и внешним (от исследуемой схемы). Для повышения универсальности анализатора целесообразно обеспечить его работу на нескольких тактовых частотах. Большие частоты будут использоваться для анализа быстрых процессов, а малые частоты – для анализа длительных процессов. Тактовая частота не должна при этом принимать слишком много значений. Вполне достаточно ряда нескольких частот, различающихся вдвое.

Пример схемы тактового генератора для логического анализатора приведен на рис. 8.14.

Тактовый генератор анализатора выполнен на кварцевом генераторе, 6-разрядном счетчике (ИЕ19) и 8-канальном мультиплексоре (КП7). Он может выдавать на выход ряд тактовых частот, различающихся в 2 раза (период 100, 200, 400, 800, 1600, 3200, 6400 нс) или внешний тактовый сигнал ВТС. То есть он позволяет реализовать как синхронный, так и асинхронный режим работы логического анализатора. Счетчик может быть применен асинхронный (ИЕ19), так как каждый его выход используется самостоятельно, независимо от других. Выбор канала мультиплексо-

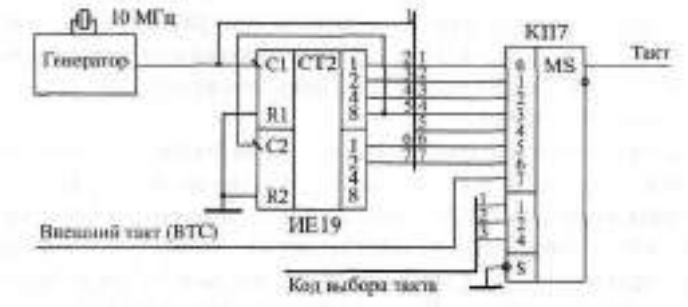


Рис. 8.14. Тактовый генератор логического анализатора

ра (вида тактового сигнала) осуществляется 3-разрядным кодом выбора такта, причем код 111 будет соответствовать внешнему тактовому сигналу.

Схема запуска анализатора должна обеспечивать выбор положительного или отрицательного фронта (синхроперехода) на одной из 8 входных линий анализатора. Выходным сигналом запуска является положительный фронт (см. рис. 8.13). Для выбора одного из восьми входных сигналов удобно использовать 8-канальный мультиплексор (КП7), а для выбора полярности перехода можно применить элемент Исключающее ИЛИ, включенный в режиме управляемого инвертора. Схема запуска получается очень простой (рис. 8.15). Выбор входного сигнала, по которому будет производиться запуск, осуществляется 3-разрядным управляющим кодом выбора синхролинии. Выбор полярности перехода производится внешним сигналом выбора полярности, причем единица на этом входе соответствует отрицательному фронту, а нуль — положительному фронту.

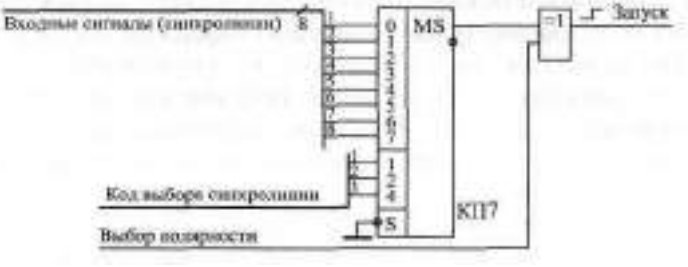


Рис. 8.15. Схема запуска логического анализатора

Наконец, последний узел логического анализатора — это память с буферами данных. Память должна иметь организацию 4Кх32, для чего придется использовать 4 микросхемы, так как обычно микросхемы оперативной памяти бывают 8-разрядные. Шина данных таких микросхем двунаправленная, поэтому требуется применение буферов для данных.

Память должна работать в режиме записи при регистрации и в режиме чтения при чтении информации, зарегистрированной логическим анализатором. Для упрощения схемы целесообразно использовать нетактируемые микросхемы памяти.

В режиме записи сигнал -CS памяти должен представлять собой отрицательные импульсы на каждый адрес памяти, а сигнал -WR должен быть постоянно активным (нулевым). Отметим, что некоторые микросхемы памяти (например, КР541РУ2) могут записывать информацию и при постоянных нулевых уровнях обоих сигналов CS и WR при изменении только адресов памяти. Использование таких микросхем еще более упрощает схему.

В режиме чтения сигнал -CS должен быть постоянно активным (нулевым), а сигнал -WR должен быть постоянно равен единице. Смена читаемой информации будет производиться только сменой адресов памяти.

Для чтения информации из памяти порциями по 8 разрядов надо применить четыре 8-разрядных однонаправленных буфера (типа АП5). Каждый из них будет открываться своим стробом чтения и выдавать на общую 8-разрядную шину данных по 8 разрядов читаемой из памяти информации. Таким образом, чтение 32 разрядов из одного адреса памяти потребует четырех циклов чтения из логического анализатора. Смена адреса памяти должна происходить после последнего из этих четырех циклов чтения. То есть для чтения всего объема памяти потребуется 16384 циклов чтения по 8 разрядов из логического анализатора.

Помимо буфера чтения необходимо применить также входной 32-разрядный буфер, который будет пропускать входные (регистрируемые) сигналы на память в режиме регистрации и будет закрываться после окончания регистрации. Назначение этого буфера состоит в том, чтобы не выдавать на входные линии логического анализатора читаемую из памяти по двунаправленной шине данных информацию. Этот буфер также можно построить на микросхемах однонаправленных буферов типа АП5.

В результате схема памяти логического анализатора будет иметь вид, показанный на рис. 8.16.

Объединение четырех микросхем памяти производится стандартным образом: объединяются одноименные разряды адреса, сигналы -CS и -WR всех микросхем. На входы -CS подается сигнал, равный нулю при отсутствии регистрации (нулевой сигнал "Рег.") и равный инверсному тактовому сигналу при регистрации (отрицательному импульсу на каждый адрес памяти). Минимальная длительность импульса WR равна в режиме записи половине периода тактового сигнала с частотой 10 МГц, то есть 50 нс, поэтому память должна успевать за это время записать информацию. На входы -WR подается сигнал "-Рег.", равный нулю при реги-

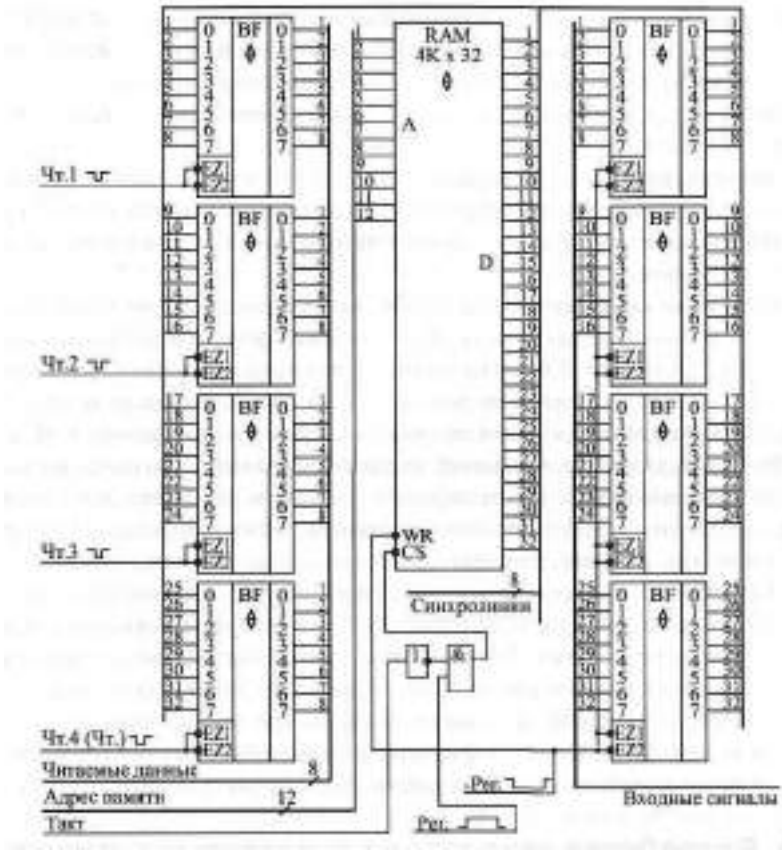


Рис. 8.16. Схема памяти логического анализатора

страции и единице при отсутствии регистрации. Если память может записывать информацию при постоянных нулевых сигналах CS и WR, то на вход -CS можно постоянно подать нулевой уровень.

Четыре микросхемы АПБ входного буфера (справа по рисунку) управляются сигналом "Рег.": они открываются на все время регистрации и закрываются, когда регистрации нет.

Четыре микросхемы АПБ буфера чтения открываются на чтение (выдают 8-разрядные читаемые данные) каждая свои стробом чтения: "-Чт. 1"… "-Чт. 4". При этом сигнал "-Чт.4" (его задний фронт) используется для переключения счетчиков адреса памяти при чтении (см. сигнал "-Чт." на рис. 8.13). Для чтения информации из одного адреса памяти нужно последовательно подать сигналы "-Чт. 1"… "Чт. 4", после чего адрес переключится на следующий.

В качестве синхролиний для запуска анализатора (см. рис. 8.15) используются восемь разрядов данных памяти. В режиме регистрации на них приходят 8 входных сигналов логического анализатора.

Таким образом, схема логического анализатора спроектирована полностью.

Сформулируем теперь порядок управления этой схемой. Управлять ею может, например, компьютер или контроллер, который будет отображать, обрабатывать и хранить зарегистрированные последовательности входных сигналов.

Перед началом регистрации необходимо записать в счетчики анализатора код N (количество тактов предпусковой регистрации) по сигналу записи "Зап." (см. рис. 8.13). Необходимо также установить 7-разрядный управляющий код, который определит режим работы анализатора. Три разряда этого кода задают тип тактового сигнала анализатора (см. рис. 8.14). Три разряда выбирают номер входного сигнала (из восьми возможных), на котором фронт будет служить запуском, а последний седьмой разряд определит полярность этого фронта, синхроперехода (см. рис. 8.15). После этого можно начинать регистрацию по сигналу "-Старт" (см. рис. 8.13). Анализатор отсчитает "мертвое" время, зафиксирует приход синхроперехода (запуск) и остановит регистрацию через нужное количество тактов после запуска. Узнать о том, что регистрация завершилась, можно, исходя из анализа сигнала "Рег." (см. рис. 8.13). Затем можно начинать чтение из буферной памяти анализатора по стробам "-Чт. 1". "-Чт.4" (см. рис. 8.16). Когда информация из всех 4096 адресов памяти будет прочитана, анализатор снова будет готов к регистрации.

8.4. Разработка генератора аналоговых сигналов

Цифровые генераторы (или, как их еще называют, синтезаторы) аналоговых сигналов произвольной формы часто используются при отладке различных аналоговых и аналого-цифровых устройств и систем. Они позволяют не только получить сигналы разных стандартных и нестандартных форм, но и обеспечить высокую точность задания амплитуды и частоты сигнала, не достижимые в случае обычных аналоговых генераторов. Цифровые генераторы работают обычно под управлением компьютеров или контроллеров, что обуславливает большие удобства пользователя и широкие возможности по заданию разнообразных форм сигналов и по их хранению.

Мы будем разрабатывать довольно простой генератор, рассчитанный на звуковой диапазон частот выходного сигнала 20 Гц... 20 кГц (период от 50 мкс до 50 мс). Генератор должен формировать сигналы произвольной формы с амплитудой, задаваемой управляющим кодом. Генера-

тор должен работать в режиме автоматической (периодической) генерации, а также в режиме разовой генерации с остановкой генерации после окончания одного периода выходного сигнала. Управление работой генератора должно быть полностью цифровым.

Отметим, что в реальности сигналы сложной формы, как правило, бывают низкочастотными. Они встречаются, например, при виброиспытаниях, в медицинской технике, в сейсмической технике и т. д. Высокочастотные сигналы обычно имеют довольно простую форму, например, синусоидальную. Поэтому наш простой генератор, рассчитанный на невысокие частоты, будет, тем не менее, удовлетворять требованиям довольно широкого спектра применений.

Разработку генератора мы начнем "с конца", то есть с того выходного сигнала, который он должен формировать.

Как уже отмечалось в главе 7 (см. раздел 7.1), выходной сигнал ЦАП $U_{ЦАП}$ представляет собой ступенчатую функцию, которую можно представить в виде суммы идеального ("гладкого") аналогового сигнала $U_{ВЫХ}$ и пилообразного сигнала помехи $U_{ПОМ}$ (рис. 8.17).

Сигнал помехи $U_{ПОМ}$ имеет основную частоту, равную частоте поступления входных кодов на ЦАП. Для сглаживания ступенек выходного сигнала ЦАП и приближения его к идеальному сигналу $U_{ВЫХ}$ можно применить простой аналоговый фильтр низкой частоты (ФНЧ), который должен существенно ослаблять сигнал помехи, но не ослаблять полезный выходной сигнал генератора. В примере на рис. 8.17 частота полезного сигнала в 16 раз меньше частоты сигнала помехи, поэтому задача фильтрации не слишком сложна. Однако от генератора сигналов произвольной формы может понадобиться синтез выходных сигналов с крутыми фронтами (например, прямоугольных или пилообразных сигналов). В этом случае применение такого выходного фильтра низкой частоты может исказить выходные сигналы, затянув их фронты. Поэтому целесообразно предусмотреть два выхода генератора: один с низкочастотной фильтрацией, а другой без нее.

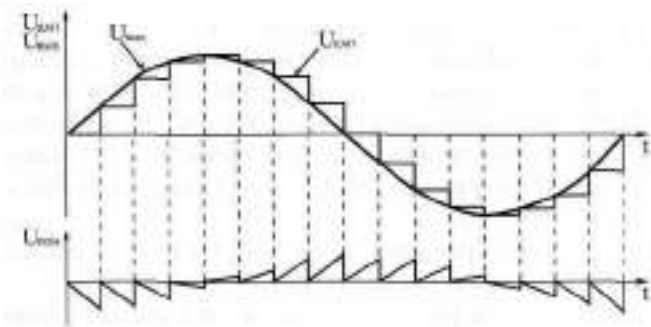


Рис. 8.17. Цифровая генерация аналогового сигнала

Помимо фильтра низкой частоты, выходной узел генератора сигналов должен содержать схему задания амплитуды выходного сигнала. В случае использования оперативной памяти для хранения кодов выборок выходного сигнала, схема задания амплитуды может и отсутствовать. При этом в память необходимо заносить коды выборок сигнала с нужной амплитудой. Однако такой подход не слишком удобен, так как он требует пересчета всех кодов выборок для каждой новой амплитуды сигнала выбранной формы. Гораздо удобнее сделать так, чтобы в памяти всегда хранились коды выборок сигнала с максимально возможной амплитудой, а выходной сигнал с ЦАП ослаблялся управляемым аттенюатором в нужное количество раз.

В результате схема выходного узла генератора аналоговых сигналов будет включать в себя еще и управляемый аттенюатор, рассмотренный в разделе 7.1 (рис. 8.18).

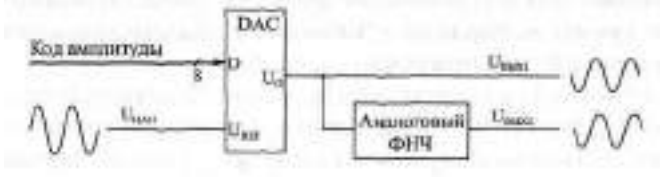


Рис. 8.18. Схема выходного узла генератора

Аналоговый фильтр нижней частоты должен иметь коэффициент передачи в полосе пропускания, равный единице и частоту среза, обеспечивающую эффективное подавление сигнала помехи. Тип схемы фильтра и его порядок не слишком важны. Для удобства пользователя целесообразно сделать фильтр неинвертирующим, чтобы выходные сигналы на обоих выходах генератора ($U_{ВЫХ1}$ и $U_{ВЫХ2}$) были одной полярности. Аттенюатор управляется 8-разрядным кодом амплитуды, что обеспечивает коэффициент деления сигнала от 1/256 до 1. Если амплитуда исходного сигнала $U_{ЦАП}$ равна 10 В, то амплитуда выходного сигнала ($U_{ВЫХ1}$ и $U_{ВЫХ2}$) может быть задана с точностью около 40 мВ. Увеличение разрядности кода амплитуды потребовало бы принятия специальных мер, так как слишком малые аналоговые сигналы сильно искажаются шумами и помехами по цепям питания. ЦАП необходимо применять умножающий с биполярным выходом, чтобы обрабатывать как положительные, так и отрицательные выходные сигналы.

Теперь переходим к проектированию собственно цифровой части генератора.

Как уже отмечалось в разделе 7.1, основной узел генератора должен представлять собой буферную оперативную память с периодическим ре-

жимом работы. Причем буфер этот должен быть однонаправленным. Перед началом работы в буфер заносится массив кодов выборок синтезируемого сигнала, а во время работы генератора адреса памяти опрашиваются в нужном темпе, и выходные коды памяти подаются на ЦАП, формирующий аналоговый сигнал $U_{ЦАП}$. Проблема состоит в выборе нужного объема памяти и в способе перебора адресов для обеспечения нужной частоты выходного сигнала. Память может также быть постоянной (ПЗУ), если необходимо формировать одну или несколько постоянных форм сигналов. В этом случае операция записи в память исключается, но проблема выбора способа перебора адресов памяти остается.

Существует два основных способа перебора адресов памяти генератора аналоговых сигналов, каждый из которых имеет свои достоинства и недостатки.

Первый, простейший способ предусматривает перебор адресов памяти генератора с помощью обычного двоичного счетчика. В данном случае, опрашиваются все адреса памяти подряд. Изменение частоты аналогового выходного сигнала генератора производится с помощью изменения тактовой частоты этого счетчика, для чего используется тот или иной управляемый делитель частоты опорного кварцевого генератора (рис. 8.19). Частота выходного сигнала будет определяться при таком решении по формуле $f_{вых} = f_Г/(N2^n)$, где $f_Г$ – частота задающего кварцевого генератора, N — управляющий код делителя частоты, n — разрядность счетчика (разрядность шины адреса памяти).

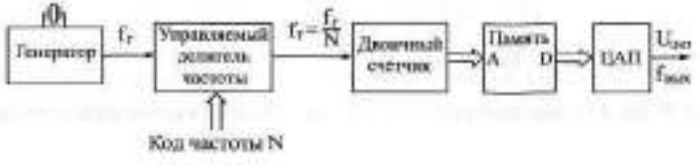


Рис. 8.19. Опрос памяти с помощью двоичного счетчика

Главное достоинство данного подхода состоит в том, что при изменении частоты выходного сигнала не меняется точность воспроизведения формы выходного сигнала. Ведь точность воспроизведения формы аналогового сигнала зависит в первую очередь от количества выборок, приходящихся на период выходного сигнала, а здесь оно постоянно и равно количеству адресов памяти. Например, если память имеет 1К адресов, то выходной сигнал при любой частоте будет задаваться с помощью 1024 точек, и он всегда будет иметь 1024 ступеньки.

Однако данное решение имеет и серьезные недостатки. Основной его недостаток состоит в том, что частота сигнала помехи в данном случае

прямо пропорциональна частоте выходного аналогового сигнала генератора (она больше частоты выходного сигнала во столько раз, сколько адресов имеет память). Например, при 1К адресов памяти частота сигнала помехи в 1024 раз больше частоты выходного сигнала, и при изменении частоты выходного сигнала в 1000 раз также в 1000 раз будет изменяться частота сигнала помехи. Отфильтровать такую помеху переменной частоты чрезвычайно трудно, если не невозможно, так как требуется применение фильтра с частотой среза, изменяемой в очень широких пределах.

Другой существенный недостаток данного метода связан с высокими требованиями к быстродействию ЦАП. Например, если максимальная частота выходного аналогового сигнала генератора должна быть 20 кГц, а память имеет 1К адресов, то ЦАП должен успевать работать с частотой более 20 МГц, то есть иметь время установления менее 50 нс. При большей частоте выходного сигнала и при большем объеме памяти требования к быстродействию **ЦАП** будут еще выше. И с такой же скоростью должна работать буферная память, то есть требования к быстродействию памяти также велики.

Второй возможный способ перебора адресов памяти генератора аналоговых сигналов состоит в применении накапливающего сумматора с переменным шагом суммирования (рис. 8.20).

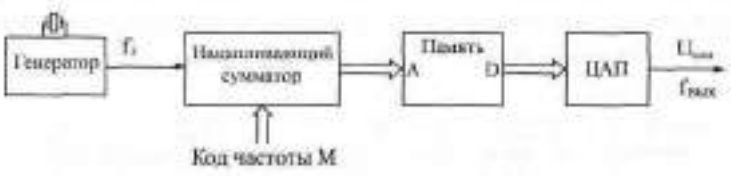


Рис. 8.20. Опрос памяти с помощью накапливающего сумматора

В память, как и в предыдущем случае, заносится массив кодов выборок периода требуемого сигнала. Но при генерации опрашиваются не все адреса памяти подряд, а только адреса с шагом, задаваемым входным кодом накапливающего сумматора M (см. раздел 4.2.1). Чем больше этот шаг, тем быстрее будет пройден весь объем памяти и тем больше будет частота выходного сигнала генератора. И, соответственно, чем меньше шаг, тем больше времени потребуется на опрос всех адресов памяти, тем меньше будет частота выходного сигнала генератора.

При изменении шага опроса памяти изменяется и количество выборок на период выходного сигнала, что приводит к изменению точности воспроизведения формы сигнала. Количество выборок K на период выходного сигнала вычисляется по формуле $K = 2^n/M$, где n — количество разрядов адреса памяти, M — управляющий код накапливающего сумматора.

А частота выходного аналогового сигнала определяется формулой $f_{ВЫХ} = f_Г$ M /2% где $f_Г$ — частота задающего кварцевого генератора. То есть выходная частота прямо пропорциональна управляющему коду M, а не обратно пропорциональна, как в предыдущем случае.

Главное достоинство данного подхода состоит в том, что сигнал помехи на выходе всегда имеет одну и ту же частоту, равную частоте задающего кварцевого генератора $f_Г$, независимо от частоты выходного аналогового сигнала. Поэтому такую помеху легко отфильтровать, никакой перестройки частоты среза фильтра не требуется.

Другое важное достоинство данного решения состоит в том, что по мере роста частоты выходного сигнала генератор сам пропорционально уменьшает количество выборок на период выходного сигнала, поэтому требования к быстродействию ЦАП, формирующего выходной сигнал, не слишком жесткие. ЦАП может быть в несколько раз более медленным, чем в предыдущем случае, при такой же максимальной выходной частоте. Или, можно сказать и так, при том же самом ЦАП генератор может выдавать выходные сигналы с гораздо более высокой частотой. Точно так же снижаются и требования к быстродействию памяти. Это приводит к тому, что объем памяти в данном случае может быть гораздо больше, чем в предыдущем.

Но ничто не дается даром, поэтому данный метод имеет и существенный недостаток. С ростом частоты выходного сигнала его форма будет передаваться все более грубо, ступеньки будут все больше. На рис. 8.21 приведен пример воспроизведения формы синусоидального сигнала, записанного в память объемом 32Kx8 для двух разных шагов наращивания адреса M (количество выборок на период K = 16 и K = 48). Понятно, что точность воспроизведения формы сигнала сильно зависит от кода M. Это может привести к тому, что некоторые фрагменты сигналов сложной фор-

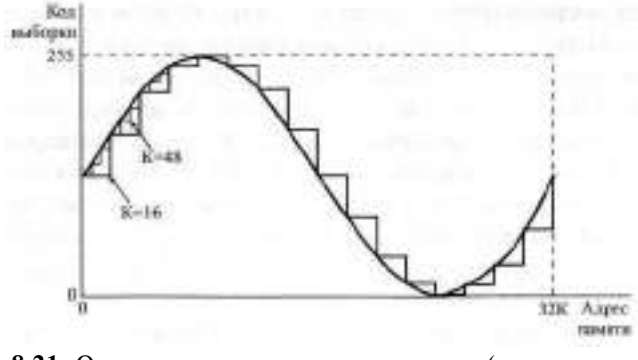


Рис. 8.21. Опрос памяти с разными шагами (количество выборок на период K = 16 и K = 48)

мы могут быть пропущены. К тому же в случае, когда количество выборок на период выходного сигнала К не равно целому числу, периоды выходного сигнала будут несколько отличаться один от другого. Смягчает этот недостаток уже упоминавшееся обстоятельство, что в природе сигналы сложной формы обычно низкочастотные, а именно низкочастотные сигналы воспроизводятся при данном методе наиболее точно.

Исходя из всех этих соображений, останавливаем свой выбор именно на этом, втором методе.

Примем для дальнейшего проектирования, что минимальное количество выборок на период выходного сигнала будет равно 32, а максимальное будет равно количеству адресов памяти. Так как от генератора требуется большой диапазон выходных частот (частоты могут различаться в 1000 раз), объем памяти должен быть большим. Если минимальное количество выборок на период равно 32, то максимальное количество выборок на период потребуется в тысячу раз больше, то есть 32000. Поэтому количество адресов памяти не должно быть меньше 32000. Возьмем память с количеством адресов, равным 32К.

Количество разрядов данных памяти, определяющее точность задания величины выборок выходного сигнала, не стоит брать слишком большим. Ведь на формируемый аналоговый сигнал будут накладываться помехи от цифровой части схемы, поэтому чрезмерно точное задание величин выборок выходного сигнала окажется попросту излишним. Поэтому выберем количество разрядов данных памяти равным 8, то есть память будет иметь организацию 32Кх8.

Спроектируем накапливающий сумматор для генератора аналоговых сигналов.

Как уже отмечалось, частота выходного аналогового сигнала прямо пропорциональна управляющему коду накапливающего сумматора [illegible]. Абсолютная погрешность установки частоты составит 0, 5/М. Поэтому для малых частот погрешность установки частоты будет максимальной. Например, если коду М = 1 будет соответствовать частота 20 Гц, то следующее разрешенное значение частоты будет равно 40 Гц (при М = 2). Это не слишком удобно, хорошо бы иметь точность установки частоты не ниже хотя бы 10 % во всем частотном диапазоне. Возьмем, например, абсолютную погрешность установки частоты 0,5 Гц. Значит, при М = 1 генератор должен выдавать частоту 1 Гц. Такие низкие частоты мы можем просто не использовать, зато частота 20 Гц (при М = 20) будет иметь точность установки 2,5 [illegible]. Разрешенные значения частот вблизи 20 Гц составят при этом 19 Гц, 20 Гц, 21 Гц.

Выберем теперь величину тактовой частоты накапливающего сумматора (то есть частоты задающего кварцевого генератора). Максимальная частота выходного сигнала нашего генератора должна быть равна

20 кГц, при этом на период выходного сигнала должно приходиться 32 выборки. То есть тактовая частота накапливающего сумматора должна быть не менее 20 кГц • 32 = 640 кГц. Выберем с запасом тактовую частоту равной 1 МГц. Максимальная частота выходного аналогового сигнала при 32 выборках на период будет при этом составлять 1 МГц/32 31,25 кГц.

Количество разрядов накапливающего сумматора должно быть таким, чтобы он обеспечивал весь выбранный частотный диапазон. Нетрудно подсчитать, что нам потребуется 20-разрядный накапливающий сумматор (так как $2^{20} = 1048576$), то есть при тактовой частоте 1 МГц минимальный период выходного сигнала составит 1048576 тактов или чуть более одной секунды, что примерно соответствует частоте выходного сигнала в 1 Гц.

Если использовать 4-разрядные микросхемы полных сумматоров (ИМ3 или ИМ6), то для построения 20-разрядного сумматора потребуется 5 микросхем сумматоров. Для запоминания выходного кода сумматоров надо будет использовать три микросхемы 8-разрядных регистров, причем регистры эти должны быть со входом сброса (например, ИР35) для начального сброса накапливающего сумматора.

Получившаяся в итоге схема накапливающего сумматора приведена на рис. 8.22. В качестве тактового сигнала она использует в режиме генерации сигнал с кварцевого генератора частотой 1 МГц (разрешающий сигнал Ген."), а в режиме записи в память кодов выборок — строб записи в память "Зап.". На входы адреса памяти подаются сигналы 15 старших выходных разрядов накапливающего сумматора, а 5 младших разрядов накапливающего сумматора не используются. Код частоты М подается на 15 младших входных разрядов накапливающего сумматора, а на старшие 5 разрядов поданы нулевые сигналы. В результате при максимальном коде M=32767 накапливающий сумматор будет переполняться за 32 такта (выходная частота 31,25 кГц), а при минимальном коде M=1 — за 1048576 тактов (выходная частота около 1 Гц).

Перед началом записи в память накапливающий сумматор должен быть сброшен в нуль сигналом "-Сброс НС". Во время записи в память каждый строб записи "-Зап." должен увеличивать на единицу адрес памяти, поэтому код частоты М должен быть установлен в данном режиме равным 32 (двоичный код 100000).

Условия правильной работы накапливающего сумматора следующие. За период тактового генератора должны успеть сработать регистр и сумматор. В нашем случае это условие довольно легко выполняется, так как период тактового генератора 1 мкс. Но при построении более высокочастотных генераторов аналоговых сигналов требуется более высокая тактовая частота, и при этом может уже сказаться накопление задержек

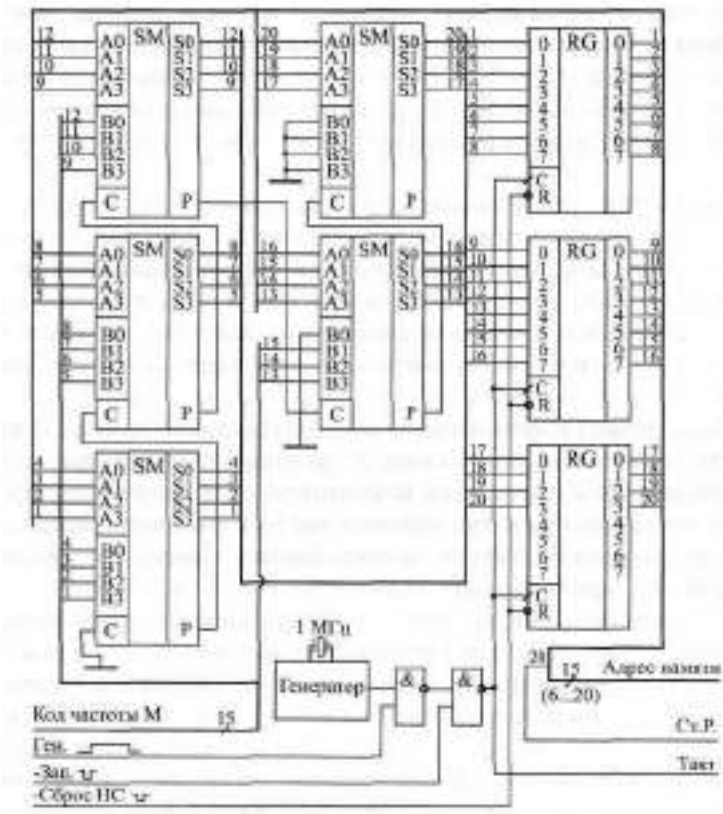


Рис. 8.22. Накапливающий сумматор генератора аналоговых сигналов

переноса пяти микросхем сумматоров. При тактовой частоте больше 10 МГц это уже может вызвать большие проблемы. Точно так же за период следования стробов записи в памяти "Зап." должны успевать срабатывать регистр и сумматоры. Это условие обычно значительно проще выполнить, чем первое.

Посмотрим, какой будет частота сигнала помехи и какой должна быть частота среза выходного аналогового низкочастотного фильтра (см. рис. 8.18). При управляющем коде частоты М больше или равном 32 каждый тактовый импульс будет вызывать изменение адреса памяти. Поэтому частота помехи будет равна частоте тактового генератора (1 МГц). Это соответствует частоте выходного сигнала, большей 32 Гц. Однако нам надо обеспечить нижнюю частоту выходного аналогового сигнала 20 Гц.

Если код частоты М будет лежать в пределах от 16 до 31, то адрес памяти будет изменяться не реже одного раза на два такта тактового генера-

тора. Частота помехи будет не менее 500 кГц. То есть при частоте выходного сигнала, большей 16 Гц, частота сигнала помехи будет в пределах от 500 кГц до 1 МГц. Максимальная частота выходного аналогового сигнала равна 31,25 кГц. Значит, частота среза фильтра должна быть такой, чтобы сильно ослаблять частоты, большие 500 кГц, но не искажать частоты, меньшие 31,25 кГц. Эти частоты различаются в 16 раз, поэтому фильтр построить не слишком сложно.

В результате мы получаем, что выбранная схема накапливающего сумматора обеспечивает диапазон частот выходного аналогового сигнала от 16 Гц до 31,25 кГц, причем погрешность установки частоты составляет 0,5 Гц во всем частотном диапазоне. Количество выборок сигнала на период будет изменяться от 32 на верхнем краю частотного диапазона до 32К на нижнем краю частотного диапазона. Это вполне удовлетворяет требованиям к генератору, сформулированным в начале данного раздела.

Переходим теперь к проектированию схемы управления для генератора аналоговых сигналов.

Схема управления генератора должна обеспечивать два режима работы: режим записи в память и режим генерации. Причем генерация может быть как автоматическая (периодическая), так и разовая. Эти режимы реализуются простой схемой на двух триггерах (рис. 8.23).

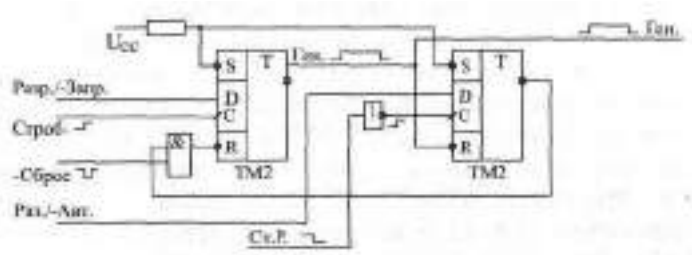


Рис. 8.23. Схема управления для генератора аналоговых сигналов

Первый (левый по рисунку) триггер служит для разрешения или запрещения генерации. По внешнему сигналу "Строб" (положительный фронт) в него записывается единица для разрешения генерации или нуль для запрещения генерации. Выходной сигнал "Ген." используется для разрешения тактовых импульсов накапливающего сумматора (см. рис. 8.22) и для управления остальной частью схемы. Перед началом работы генератора этот триггер сбрасывается в нуль внешним сигналом начального сброса "-Сброс".

Второй (правый по рисунку) триггер служит для организации режима разового запуска генератора. При запрете генерации этот триггер сброшен в нуль сигналом "Ген." (единица на инверсном выходе). При разре-

шении генерации этот триггер срабатывает по отрицательному фронту на старшем разряде накапливающего сумматора (сигнал "Ст. Р." со схемы на рис. 8.22), то есть по переполнению накапливающего сумматора, возникающему после окончания одного периода аналогового сигнала. Если внешний управляющий сигнал "Раз. /-Авт." установлен в нуль (автоматический запуск), то ничего не происходит, триггер остается сброшенным. Если же внешний сигнал "Раз. /-Авт." установлен в единицу (разовый запуск), то после окончания одного периода выходного аналогового сигнала генератора второй триггер перебросится в единицу (нуль на инверсном выходе) и сбросит тем самым первый триггер, запретив генерацию. Узнать об этом можно, анализируя флаг генерации —сигнал "Ген.". Для нового разрешения генерации надо снова записать единицу в первый триггер.

Наконец, последний узел генератора аналоговых сигналов — это память с ЦАП.

Прежде всего надо обеспечить, чтобы ЦАП, формирующий выборки аналогового сигнала по кодам из памяти, выдавал как положительные, так и отрицательные сигналы, то есть был биполярным. Это существенно повысит универсальность генератора. ЦАП должен формировать выходное напряжение (а не выходной ток), что позволит более просто обрабатывать выходной сигнал выходным узлом (см. рис. 8.18). Требования к быстродействию ЦАП в нашем случае невелики: коды всегда поступают на него с периодом в 1 мкс, значит, за это время ЦАП должен успеть установить свое выходное напряжение. Таких ЦАП существует довольно много.

Опорное напряжение ЦАП удобно выбирать равным 10 В, что обеспечит размах выходного сигнала от —10 В до +10 В. При этом шаг изменения выходного сигнала (минимально возможная высота ступеньки) составит 20 В / 256, то есть около 80 мВ. Но это только для сигнала максимальной амплитуды 10 В. Если же требуется генерация сигнала с амплитудой 1 В (ослабление выходным аттенюатором в 10 раз), то шаг изменения выходного сигнала будет около 8 мВ.

Входной код ЦАП (то есть выходной код буферной памяти) должен фиксироваться в параллельном регистре, чтобы все разряды этого кода подавались на входы ЦАП одновременно. В момент отсутствия генерации на выходе ЦАП должно быть нулевое напряжение, поэтому данный регистр должен иметь вход сброса, на который подается сигнал "Ген.". Однако надо учитывать, что при биполярном выходе ЦАП нулевому уровню выходного сигнала соответствует не нулевой код 00000000, а код 10000000 (с единицей в старшем разряде). Поэтому регистр должен сбрасываться не в нуль, а именно в состояние 10000000. При этом просто поставить дополнительный инвертор на старший разряд кода нельзя, так как он внесет задержку и старший разряд кода будет устанавливаться позже остальных разрядов, что может вызвать недопустимо большие выбро-

сы выходного напряжения. Поэтому этот входной регистр ЦАП должен иметь как прямые, так и инверсные выходы (например, ТМ8), причем все разряды, кроме старшего, надо брать с прямых выходов регистра, а старший разряд — с инверсного выхода. Это обеспечит одновременное изменение всех разрядов кода. Для компенсации инверсии старшего разряда надо дополнительно проинвертировать сигнал старшего разряда на входе регистра.

Память выборок сигнала целесообразно использовать многоразрядную с совмещенной входной и выходной шинами данных, что позволит упростить схему. Микросхемы с организацией 32Kx8 выпускаются многими фирмами. Память лучше брать нетактируемую, чтобы в режиме чтения (при генерации) можно было постоянно подавать на вход -CS сигнал логического нуля. Быстродействие памяти не слишком критично, так как перебор адресов происходит довольно медленно. За период тактового сигнала (1 мкс) в режиме чтения должен успеть сработать регистр накапливающего сумматора, и память должна успеть выдать читаемый код (с задержкой выборки адреса).

Совмещенная шина входных/выходных данных памяти требует применения однонаправленного входного буфера (например, АП5), через который в режиме записи на память будут подаваться записываемые в память коды выборок генерируемого сигнала. Буфер должен открываться тем же сигналом, который подается на вход –WR памяти. Во время генерации буфер должен быть закрыт.

В результате схема буферной памяти с ЦАП для генератора аналоговых сигналов будет выглядеть так, как показано на рис. 8.24.

Перед началом работы в память должны быть записаны коды выборок (8-разрядная шина "Зап. Дан".) по стробу "–Зап.". Данные должны выставляться до начала строба и сниматься после его окончания. Во время строба записи "Зап." память переходит в режим записи (сигнал -WR), а буфер открывается (сигналы EZ1 и EZ2). За счет задержки буфера записываемые данные снимаются со входов данных памяти позже, чем заканчивается сигнал "Зап.". Поэтому данные записываются в память. По окончании сигнала "Зап." происходит смена адреса памяти (см. рис. 8.22). Всего должно быть проведено 32K циклов записи для полного заполнения памяти.

Когда начинается генерация (сигнал "Ген."), адреса памяти перебираются накапливающим сумматором, а читаемая из них информация записывается по сигналу "Такт" (см. рис. 8.22) в 8-разрядный регистр (две микросхемы ТМ8), а затем поступает на входы ЦАП. В результате выдача выборок выходного сигнала ($U_{ЦАП}$) задерживается на один такт относительно момента чтения из памяти, но эта задержка, как правило, не имеет никакого значения. После окончания генерации регистр сбрасывается

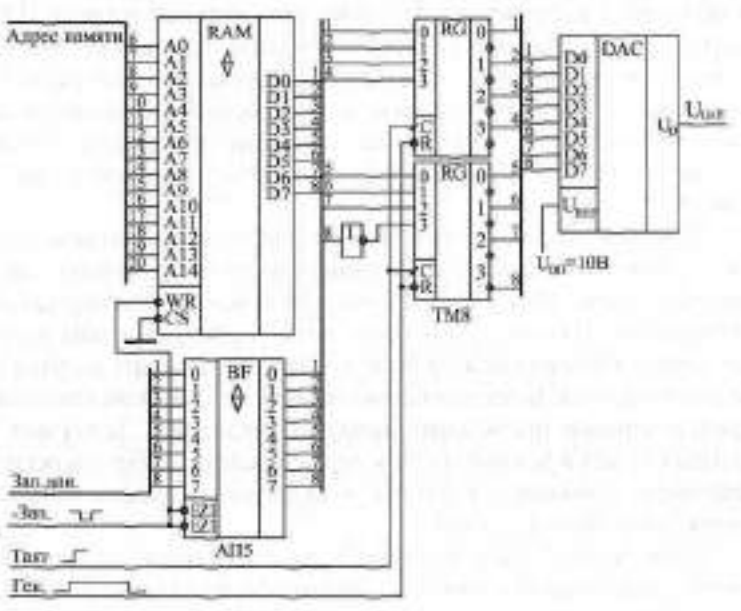


Рис. 8.24. Память и ЦАП генератора аналоговых сигналов

в состояние 10000000, соответствующее нулю выходного сигнала $U_{ЦАП}$. Так как по сигналу начального сброса "-Сброс" (см. рис. 8.23) генерация запрещается, на выходе генератора в этот момент также будет нулевое напряжение.

Таким образом, схема генератора аналоговых сигналов полностью спроектирована.

Сформулируем теперь последовательность действий, которые надо предпринимать для управления работой генератора.

После включения питания надо подать сигнал начального сброса "-Сброс" (см. рис. 8.23), который запретит генерацию и обеспечит нулевой уровень выходного напряжения генератора.

Затем необходимо записать в память массив кодов выборок сигнала требуемой формы. Для этого код частоты надо задать равным 32 и сбросить накапливающий сумматор в нуль сигналом "-СбросНС". После этого надо производить последовательную запись всех 32К кодов по шине записываемых данных Зап. дан., сопровождая их стробами записи "Зап.".

После окончания записи в память можно запускать генерацию, но перед началом генерации надо сбросить накапливающий сумматор сигналом "-СбросНС", задать режим запуска генерации (разовый или авто-

матический), а также установить код нужной выходной частоты (см. рис. 8.22 и 8.23). Кроме того, надо задать код амплитуды выходного сигнал (см. рис. 8.18). После этого надо подать положительный сигнал "Разр./Запр." и сопроводить его стробом (см. рис. 8.23). Если требуется остановить автоматическую генерацию, то нужно установить нулевой сигнал "Разр./Запр." и сопроводить его стробом. Если же генерация разовая, то узнать о том, продолжается ли она или уже закончилась, можно на основании анализа сигнала "Ген." (см. рис. 8.23).

В заключение отметим, что управление разработанным генератором аналоговых сигналов лучше возложить на компьютер или управляющий интеллектуальный контроллер, что существенно упростит работу с ним.

Приложение. Микросхемы, параметры, сигналы

Таблица П1. Основные параметры цифровых микросхем

Обозначение	Другое обозначение	Параметр
C_L	$C_н$	Допустимая емкость нагрузки
F_M	$F_{МАКС}$	Максимальная частота переключения триггера
I_{CC}	$I_П$	Ток потребления
I_I	$I_{ВХ}$	Входной ток
	$I_{ВЫХ}$	Выходной ток
I_{IL}	$I^0_{ВХ}$	Входной ток низкого уровня
I_{IH}	$I_{ВХ}$	Входной ток высокого уровня
I_{OL}	$I_{ВЫХ}$	Выходной ток низкого уровня
I_{OH}	$I_{ВЫХ}$	Выходной ток высокого уровня
t_{LH}	t^{01}	Время фронта сигнала при переходе из низкого уровня в высокий
t_{HL}	t^{10}	Время фронта сигнала при переходе из высокого уровня в низкий
t_{PLH}	$t^{01}_З$	Время задержки при переходе из низкого уровня в высокий
t_{PHL}	$t^{10}_З$	Время задержки при переходе из высокого уровня в низкий
t_{PZL}	$t^{Z0}_З$	Время задержки при переходе из третьего состояния в низкий уровень
t_{PZL}	$t^{Z1}_З$	Время задержки при переходе из третьего состояния в высокий уровень
U_{CC}	$U_П$	Напряжение питания
U_{IL}	$U^0_{ВХ}$	Входное напряжение низкого уровня
U_{IH}	$U^1_{ВХ}$	Входное напряжение высокого уровня
U_{OL}	$U^0_{ВЫХ}$	Выходное напряжение низкого уровня
U_{OH}	$U_{ВЫХ}$	Выходное напряжение высокого уровня

Таблица П2. Функциональное назначение цифровых микросхем стандартных серий

Обозначение	Аналог SN74	Функция
АГ1	121	Одновибратор без перезапуска
АГ3	123	Два одновибратора с перезапуском
АГ4	221	Два одновибратора без перезапуска
АП3	240	Два 4-разрядных буфера с ЗС и инверсией
АП4	241	Два 4-разрядных буфера с ЗС
АП5	244	Два 4-разрядных буфера с ЗС
АП6	245	8-разрядный двунаправленный буфер с ЗС
АП9	640	8-разрядный двунаправленный буфер с ЗС
АП10	640	8-разрядный двунаправленный буфер с ЗС и инверсией
АП14	465	8-разрядный буфер с ЗС
АП15	466	8-разрядный буфер с ЗС и инверсией
АП16	643	8-разрядный буфер с ЗС
АП20	646	8-разрядный двунаправленный буфер с регистром и с ЗС
ВА1	226	Схема сопряжения с магистралью
ВЖ1	630	16-разрядная схема контроля по коду Хемминга
ГГ1	124	Два генератора, управляемых напряжением
ИВ1	148	Приоритетный шифратор 8-3
ИВ3	147	Приоритетный шифратор 9-4
ИД1	14	Двоично-десятичный дешифратор с высоковольтным выходом
ИД3	154	Дешифратор 4-16
ИД4	155	Сдвоенный дешифратор 2-4
ИД5	156	Два дешифратора 2-4 с ОК
ИД6	42	Двоично-десятичный дешифратор 3-8
ИД7	138	Дешифратор 3-8
ИД10	145	Двоично-десятичный дешифратор 3-8 с большим выходным током
ИД14	139	Два дешифратора 2-4

Обозначение	Аналог SN74	Функция
ИД18	247	Дешифратор двоично-десятичного кода в код семисегментного индикатора
ИЕ2	90	4-разрядный двоично-десятичный счетчик
ИЕ4	92	Счетчик-делитель на 12
ИЕ5	93	4-разрядный двоичный счетчик
ИЕ6	192	4-разрядный реверсивный двоично-десятичный счетчик
ИЕ7	193	4-разрядный реверсивный двоичный счетчик
ИЕ8	97	Делитель частоты с переменным коэффициентомделения
ИЕ9	180	4-разрядный синхронный двоично-десятичный счетчик с асинхронным сбросом
ИЕ1О	161	4-разрядный синхронный двоичный счетчик с асинхронным сбросом
ИЕ11	162	4-разрядный двоично-десятичный счетчик с синхронным сбросом
ИЕ12	190	4-разрядный синхронный реверсивный десятичный счетчик
ИЕ13	191	4-разрядный синхронный реверсивный двоичный счетчик
ИЕ14	196	Счетчик-делитель на 2 и на 5
ИЕ15	197	4-разрядный асинхронный счетчик с предварительной установкой
ИЕ16	168	4-разрядный синхронный двоично-деся-тичный счетчик с параллельной загрузкой
ИЕ17	169	4-разрядный синхронный двоичный счетчик с параллельной загрузкой
ИЕ18	163	4-разрядный двоичный счетчик с синхронным сбросом
ИЕ19	393	Сдвоенный 4-разрядный двоичный счетчик
ИЕ20	390	Два двоично-десятичных счетчика со сбросом
ИМ1	80	1-разрядный полный сумматор
ИМ2	82	2-разрядный полный сумматор
ИМ3	83	4-разрядный полный сумматор

Обозначение	Аналог SN74	Функция
ИМ5	183	4-разрядный полный сумматор с ускоренным переносом
ИМ6	283	4-разрядный полный сумматор с ускоренным переносом
ИМ7	385	4-разрядный сумматор-вычитатель
ИП2	180	8-разрядная схема контроля четности
ИП3	181	АЛУ для двух 4-разрядных слов
ИП4	182	4-разрядная схема ускоренного переноса
ИП5	280	9-разрядная схема контроля четности
ИП6	242	Двунаправленный 4-разрядный буфер с инверсией
ИП7	243	Двунаправленный 4-разрядный буфер
ИП8	261	Параллельный умножитель 2 x 4 разряда
ИП9	384	8-разрядный последовательно-параллельный умножитель
ИР1	**95**	4-разрядный двунаправленный сдвиговый регистр
ИР8	164	8-разрядный сдвиговый регистр с последовательным входом и параллельными выходами
ИР9	165	8-разрядный сдвиговый регистр с параллельными входами и последовательным выходом
ИР10	166	8-разрядный сдвиговый регистр
ИР11	194	4-разрядный 2-направленный сдвиговый регистр
ИР12	195	4-разрядный 2-направленный сдвиговый регистр
ИР13	198	8-разрядный сдвиговый регистр
ИР15	173	4-разрядный регистр с ЗС
ИР16	295	4-разрядный реверсивный сдвиговый регистр с выходами ЗС
ИР22	373	8-разрядный регистр-защелка с ЗС
ИР23	374	8-разрядный регистр с ЗС
ИР24	299	8-разрядный двунаправленный реверсивный сдвиговый регистр с ЗС
ИР25	395	4-разрядный сдвиговый регистр с ЗС

Обозначение	Аналог SN74	Функция
ИР26	670	Регистровый файл 4х4 с ЗС
ИР27	377	8-разрядный регистр с разрешением записи
ИР29	323	8-разрядный сдвиговый регистр с ЗС
ИР30	259	8-разрядный регистр хранения с адресацией
ИР32	170	Регистровый файл 4х4 с 0К
ИР33	573	8-разрядный буферный регистр
ИР34	873	Два 4-разрядных регистра с ЗС
ИР35	273	8-разрядный регистр со сбросом
ИР37	574	8-разрядный регистр с ЗС
ИР38	874	Два 4-разрядных регистра с ЗС
ИР40	533	8-разрядный регистр-защелка с ЗС и инверсией
ИР41	534	8-разрядный регистр с ЗС и инверсией
КП1	150	16-канальный мультиплексор
КП2	153	Сдвоенный 4-канальный мультиплексор
КП5	152	8-канальный мультиплексор
КП7	151	8-канальный мультиплексор со стробированием
КП11	257	4-разрядный 2-канальный мультиплексор с ЗС
КП12	253	2-разрядный 4-канальный мультиплексор
КП13	298	4-разрядный 2-канальный мультиплексор со стробированием
КП14	258	4-разрядный 2-канальный мультиплексор с ЗС с инверсией
КП15	251	8-канальный мультиплексор с ЗС
КП16	157	4-разрядный 2-канальный мультиплексор
КП17	353	2-разрядный 4-канальный мультиплексор с ЗС и инверсией
КП18	158	4-разрядный 2-канальный мультиплексор с инверсией
КП19	352	2-разрядный 4-канальный мультиплексор с инверсией
ЛА1	20	Два логических элемента 4И-НЕ
ЛА2	30	Логический элемент 8И-НЕ

Обозначение	Аналог SN74	Функция
ЛА3	00	Четыре логических элемента 2И-НЕ
ЛА4	10	Три логических элемента 3И-НЕ
ЛА6	40	Два логических элемента 4И-НЕ с повышенным выходным током
ЛА7	22	Два логических элемента 4И-НЕ с 0К и повышенным выходным током
ЛА8	01	Четыре логических элемента 2И-НЕ с 0К
ЛА9	03	Четыре логических элемента 2И-НЕ с 0К
ЛАЮ	12	Три логических элемента ЗИ-НЕсОК
ЛА11	26	Четыре логических элемента 2И-НЕсОКи повышенным выходным напряжением
ЛА12	37	Четыре логических элемента 2И-НЕ с повышенным выходным током
ЛА13	38	Четыре логических элемента 2И-НЕсОКи повышенным выходным током
ЛА16	140	Два логических элемента 4И-НЕ для работы на линию 50 Ом
ЛА19	134	Логический элемент 12И-НЕ с разрешением
ЛА21	1000	Четыре логических элемента 2И-НЕ с повышенным выходным током
ЛА22	1020	Два логических элемента 4И-НЕ с повышенным выходным током
ЛА23	1003	Четыре логических элемента 2И-НЕ с 0К и повышенным выходным током
ЛА24	1010	Три логических элемента ЗИ-НЕ с повышенным выходным током
ЛД1	60	Два 4-входовых расширителя по ИИ
ЛЕ1	02	Четыре логических элемента 2ИЛИ-НЕ
ЛЕ2	23	Два логических элемента 4ИЛИ-НЕ со стробированием
ЛЕ3	25	Два логических элемента 4ИЛИ-НЕ со стробированием
ЛЕ4	27	Три логических элемента ЗИЛ И-НЕ
ЛЕ5	28	Четыре логических элемента 2ИЛИ-НЕ с повышенным выходным током
ЛЕ6	128	Четыре логических элемента 2ИЛИ-НЕ с повышенным выходным током

Обозначение	Аналог SN74	Функция
ЛЕ7	260	Два логических элемента 5ИЛИ-НЕ
ЛИ1	08	Четыре логических элемента 2 И
ЛИ2	09	Четыре логических элемента 2И с 0К
ЛИ3	11	Три логических элемента 3И
ЛИ4	15	Три логических элемента ЗИ с 0К
ЛИ6	21	Два логических элемента 4И
ЛЛ1	32	Четыре логических элемента 2ИЛИ
ЛЛ3	136	Четыре двухвходовых логических элемента Исключающее ИЛИ с 0К
ЛН1	04	Шесть инверторов
ЛН2	05	Шесть инверторов с 0К
ЛН3	06	Шесть инверторов с 0К и повышенным выходным напряжением
ЛН4	07	Шесть буферных элементов с 0К
ЛН5	16	Шесть инверторов с 0К и повышенным выходным напряжением
ЛН6	366	Шесть инверторов с ЗС и с управлением
ЛП4	17	Шесть буферных элементов с 0К и повышенным выходным напряжением
ЛП5	86	Четыре двухвходовых логических элемента Исключающее ИЛИ
ЛП8	125	Четыре буферных элемента с ЗС и раздельным управлением
ЛП9	07	Шесть буферных элементов с 0К и повышенным выходным напряжением
ЛП10	365	Шесть буферных элементов с ЗС
ЛП11	367	Шесть буферных элементов с ЗС
ЛП12	136	Четыре двухвходовых логических элемента Исключающее ИЛИ с ОК
ЛП16	1034	Шесть буферов с повышенным выходным током
ЛП17	1035	Шесть буферов с 0К и повышенным выходным током
ЛР1	50	Два элемента 2-2И-2ИЛИ-НЕ
ЛР3	53	Элемент 2-2-2-3И-4ИЛИ-НЕ
ЛР4	55	Элемент 4-4И-2ИЛИ-НЕ
ЛР9	64	Элемент 2-4-2-3И-ИЛИ-НЕ

Обозначение	Аналог SN74	Функция
ЛР10	65	**Элемент 2-4-2-ЗИ-ИЛИ-НЕ с 0К**
ЛР11	51	**Элементы** 2-2И-2ИЛИ-НЕ и 2-ЗИ-2ИЛИ-НЕ
ЛР13	54	**Элемент** 3-2-2-3И-4ИЛИ-НЕ
ПР6	184	Преобразователь двоично-десятичного кода в двоичный
ПР7	185	Преобразователь двоичного кода в двоично-десятичный
РП1	170	Регистровое ЗУ 4х4
РП3	172	Регистровое ЗУ 8х2 с 0К
РУ1	81	ОЗУ с организацией 4х4
РУ2	89	ОЗУ с организацией 16х4
РУ3	84	ОЗУ 4х4 с дополнительными входами записи
РУ5	130	ОЗУ с организацией 256х1
РУ9	289	ОЗУ с организацией 16х4
РУ10	225	ОЗУ с организацией 16х4
СП1	85	4-разрядный компаратор кодов
ТВ1	72	Ж-триггер с элементом 3И на входе
ТВ6	107	Два Ж-триггера
ТВ9	112	Два Ж-триггера
ТВ10	113	Два Ж-триггера
ТВ11	114	Два Ж-триггера
ТВ15	109	Два Ж-триггера
ТЛ1	13	Два триггера Шмитта с инверсией и элементом 4И на входе
ТЛ2	14	Шесть триггеров Шмитта с инверсией
ТЛ3	132	Четыре триггера Шмитта с инверсией и элементом 2И на входе
ТМ2	74	Два D-триггера с прямыми и инверсными выходами
ТМ5	77	Четыре D-триггера типа «защелка»
ТМ7	75	Четыре D-триггера типа «защелка» с прямыми и инверсными выходами
ТМ8	175	Четыре D-триггера с прямыми и инверсными выходами
ТМ9	174	Шесть D-триггеров с общим синхровходом
ТР2	279	Два RS-триггера

Таблица П3. Типы микросхем семейства SN74

Номер SN74	Обозначение
00	ЛА3
01	ЛА8
02	ЛЕ1
03	ЛА9
04	ЛН1
05	ЛН2
06	ЛН3
07	ЛН4
07	ЛП9
08	ЛИ1
09	ЛИ2
10	ЛА4
11	ЛИ3
12	ЛА10
13	ТЛ1
14	ТЛ2
15	ЛИ4
16	ЛН5
17	ЛП4
20	ЛА1
21	ЛИ6
22	ЛА7
23	ЛЕ2
25	ЛЕ3
26	ЛА11
27	ЛЕ4
28	ЛЕ5
30	ЛА2
32	ЛЛ1

Номер SN74	Обозначение
37	ЛА12
38	ЛА13
40	ЛА6
42	ИД6
50	ЛР1
51	ЛР11
53	ЛР3
54	ЛР13
55	ЛР4
60	ЛД1
64	ЛР9
65	ЛР10
72	ТВ1
74	ТМ2
75	ТМ7
77	ТМ5
80	ИМ1
81	РУ1
82	ИМ2
83	ИМ3
84	РУ3
85	СП1
86	ЛП5
89	РУ2
90	ИЕ2
92	ИЕ4
93	ИЕ5
95	ИР1
97	ИЕ8

Номер SN74	Обозначение
107	ТВ6
109	ТВ15
112	ТВ9
113	**ТВ10**
114	ТВ11
121	АГ1
123	АГ3
124	ГГ1
125	ЛП8
128	ЛЕ6
130	РУ5
132	ТЛ3
134	ЛА19
136	ЛЛ3
138	ИД7
139	ИД14
140	ЛА16
141	ИД1
145	ИД10
147	ИВ3
148	ИВ1
150	КП1
151	КП7
152	КП5
153	КП2
154	ИД3
155	ИД4
156	ИД5
157	КП16

Номер SN74	Обозначение
158	КП18
160	ИЕ9
161	ИЕ10
162	ИЕ11
163	ИЕ18
164	ИР8
165	ИР9
166	ИР10
168	ИЕ16
169 •	ИЕ17
170	ИР32
170	РП1
172	РП3
173	ИР15
174	ТМ9
175	ТМ8
180	ИП2
181	ИП3
182	ИП4
183	ИМ5
184	ПР6
185	ПР7
190	ИЕ12
191	ИЕ13
192	ИЕ6
193	ИЕ7
194	ИР11
195	ИР12
196	ИЕ14
197	ИЕ15

Номер SN74	Обозначение
198	ИР13
221	АГ4
225	РУ10
226	ВА1
240	АП3
241	АП4
242	ИП6
243	ИП7
244	АП5
245	АП6
247	ИД18
251	КП15
253	КП12
257	КП11
258	КП14
259	ИР30
260	ЛЕ7
261	ИП8
273	ИР35
279	ТР2
280	ИП5
283	ИМ6
289	РУ9
295	ИР16
298	КП13
299	ИР24
323	ИР29
352	КП19
353	КП17
365	ЛП10

Номер SN74	Обозначение
366	ЛН6
367	ЛП11
373	ИР22
374	ИР23
377	ИР27
384	ИП9
385	ИМ7
390	ИЕ20
393	ИЕ19
395	ИР25
465	АП14
466	АП15
533	ИР40
534	ИР41
573	ИР33
574	ИР37
630	ВЖ1
639	АП9
640	АП10
643	АП16
646	АП20
670	ИР26
873	ИР34
874	ИР38
1000	ЛА21
1003	ЛА23
1010	ЛА24
1020	ЛА22
1034	ЛП16
1035	ЛП17

Таблица П4. Соответствие зарубежных и отечественных серий цифровых микросхем

Серии SN74	Отечественные серии
74 ... J	КМ155
74 ... N	К155
74AC ... N	КР1554
74ALS ... N	КР1533
74F ... N	КР1531
74H ... N	К131
74L ... N	К134
74 LS ... J	КМ555
74LS ... N	К555
74S ... J	КМ531
74S ... N	КР531

Таблица П5. Обозначения сигналов и микросхем

Обозначение	**Название**	**Назначение**
&	And	Элемент И
=1	Exclusive Or	Элемент Исключающее ИЛИ
+1	-	Вход счета на увеличение
-1	-	Вход счета на уменьшение
<->	-	Двунаправленная передача
<->	-	Двунаправленный сдвиг
<	—	Вход расширения сумматора «меньше»
<0	–	Перенос (заем) счетчика при инверсном счете (на уменьшение)
>	—	Вход расширения сумматора «больше»
>9		Перенос 4-разрядного двоично-десятичного счетчика при прямом счете
>15	—	Перенос 4-разрядного двоичного счетчика при прямом счете
=	-	Вход расширения сумматора «равно»

Обозначение	Название	Назначение
0, 1, 2, 3, ...	—	Номера входных или выходных разрядов кода
0V	-	Общий вывод
1	Or	Элемент ИЛИ
1, 2, 4, 8 ...	-	Входы/выходы разрядов кода
A	Address	Адресные разряды
A0, A1,...	-	Разряды входного/выходного кода А
A=B	Parity	Вход или выход равенства кодов А и В
A>B, A<B	—	Входы или выходы сравнения кодов А и В
A, B, C,...	—	Входы и выходы различного назначения
ADC	Analog-to-Digital Converter	Аналого-цифровой преобразователь, АЦП
ALU	Arithmetic Logic Unit	Арифметическо-логическое устройство
B0, B1,...	-	Разряды входного/выходного кода В
BF	Buffer	Буфер
BR	Borrow	Заем
C	Clock	Тактовый сигнал (строб), сигнал разрешения
C	Carry	Вход переноса
C	Capacitor	Подключение внешнего конденсатора
CD	Coder	Шифратор
CE	Clock Enable	Разрешение тактового сигнала
CE	Chip Enable	Разрешение работы микросхемы
CEP	Count Enable Parallel	Вход параллельного наращивания разрядности счетчиков
CET	Count Enable Trickle	Вход наращивания разрядности счетчиков («трюковый»)
CLK	Clock	Тактовый вход
CLR	Clear	Очистка, сброс

Обозначение	Название	Назначение
CPU	Central Processor Unit	Центральный процессор
CR	Carry	Перенос
CRU	Carry lock ahead Unit	Схема ускоренного переноса
CT	Counter	Счетчик
CT10	2/10 Counter	Двоично-десятичный счетчик
CT2	Binary Counter	Двоичный счетчик
CT2/10	2/10 Counter	Двоично-десятичный счетчик
CS	Chip Select	Выбор микросхемы
D	Data	Разряды данных, данные
DAC	Digital-to-Analog Converter	Цифро-аналоговый преобразователь, ЦАП
DC	Decoder	Дешифратор
DI	Data Input	Входные данные
DIO	Data Input/Output	Входные/выходные данные
DL	Data Left	Вход данных для сдвига влево
DO	Data Output	Выходные данные
DP	Data Parallel	Параллельные данные
DR	Data Right	Вход данных для сдвига вправо
DS	Data Serial	Последовательные данные
D/U	Down/Up	Переключение направления счета счетчиков
E	Enable	Разрешение
EC	Enable Count	Разрешение счета
ECR	Enable Carry	Разрешение переноса
ECT	Enable Count	Разрешение счета
EI	Enable Input	Разрешение входа
EIO	Enable Input/Output	Разрешение входа и выхода
EO	Enable Output	Разрешение выхода
EP	Enable P	Разрешение переноса
EWR	Enable Write	Разрешение записи

Обозначение	**Название**	**Назначение**
EZ	Enable Z-state	Разрешение третьего состояния
G	Generator	Генератор
Gl	Generator	Одновибратор
I	Input	Вход
I/O	Input/Output	Вход/Выход
J	-	Вход записи нуля в JK-триггере
K	-	Вход записи единицы в JK-триггере
L	Load	Загрузка, запись
LOAD	Load	Загрузка, запись
LSB	Least Significant Bit	Младший значащий разряд
M2	-	Схема контроля четности
MS	Multiplexer	Мультиплексор
MSB	Most Significant Bit	Старший значащий разряд
MUX	Multiplexer	Мультиплексор
O	Output	Выход
OE	Output Enable	Разрешение выхода
P	-	Выход переноса
PE	Parallel Enable	Разрешение параллельной загрузки
PROM	Programmable ROM	Программируемая постоянная память
P/S	Parallel/Serial	Переключение параллельный/последовательный режим
Q	Quit	Выход
R	Reset	Сброс (установка в нуль)
R	Resistor	Подключение внешнего резистора
RAM	Random Access Memory	Оперативная память, ОЗУ
RC	Resistor/Capacitor	Подключение внешнего резистора и конденсатора
RD	Read	Чтение
RE	Read Enable	Разрешение чтения

Обозначение	Название	Назначение
RG	Register	Регистр
ROM	Read Only Memory	Постоянная память, ПЗУ
R/W	Read/Write	Чтение/Запись
S	Set	Установка в единицу
S	Strobe	Стробирующий сигнал
SO, S 1 , . ..		Входы установки режима
SO, S 1 , . . .	Sum	Разряды выходного кода суммы
SE	Set Enable	Разрешение установки
SEMO	Set Mode	Установка режима
SL	Shift Left	Сдвиг влево
SM	Summater	Сумматор
SR	Synchronous Reset	Синхронный сброс
SR	Shift Right	Сдвиг вправо
SUM	Summater	Сумматор
SYN	Synchro	Синхросигнал
T	Trigger	Триггер
TC	Terminal Count	Окончание счета
U/D	Up/Down	Переключение направления счета счетчиков
UCC	-	Напряжение питания
V	-	Входы управления работой
WE	Write Enable	Разрешение записи
WR	Write	Запись
X/Y	-	Преобразователь кодов
Z	Z-state	Третье состояние выхода

Список литературы

1. *Титце У., Шенк К.* Полупроводниковая схемотехника: Справочное руководство. Пер. с нем. М.: Мир, 1982. 512 с.
2. *Ю. Р. Гнатек* Справочник по цифро-аналоговым и аналого-цифровым преобразователям: Пер. с англ. / Под ред. Ю. А. Рюжина. М.: Радио и связь, 1982. 420 с: ил.
3. *Б. И. Горошков* Радиоэлектронные устройства: Справочник. М.: Радио и связь, 1984. 400 с: ил.
4. *В. Л. Шило* Популярные цифровые микросхемы: Справочник. М.: Радио и связь, 1987. 352 с.
5. Полупроводниковые БИС запоминающих устройств: Справочник/ Под ред. А. Ю. Гордонова и Ю. Н. Дьякова. М.: Радио и связь, 1987. 360 с: ил.
6. *С. М. Бородин, Ю. В. Новиков* Модуль логического анализатора для контрольно-измерительных систем на базе микроЭВМ// Микропроцессорные средства и системы. 1987. № 1. с. 67—68.
7. *С. М.Бородин, Ю. В. Новиков, А. П. Поддубный, А. А. Томчук* Средства отображения информации для микропроцессорных систем измерения, контроля и управления// Микропроцессорные средства и системы. 1988. № 3. с. 76-79.
8. *В. В. Овчинников, И. И. Рыбкин* Техническая база интерфейсов локальных вычислительных сетей. М.: Радио и связь, 1989. 272 с: ил.
9. *Ю. В. Новиков* Универсальный параллельный интерфейс для модульных микропроцессорных систем измерения, контроля и управления// Микропроцессорные средства и системы. 1989. № 6. с. 71-72.
10. *Б. В. Шевкопляс* Микропроцессорные структуры. Инженерные решения: Справочник. 2-е изд. перераб. и доп. М.: Радио и связь, 1990 512 с.: ил.
11. Большие интегральные схемы запоминающих устройств: Справочник/ Под ред. А. Ю. Гордонова и Ю. Н. Дьякова. М.: Радио и связь, 1990. 288 с: ил.
12. *В. С. Чернега, В. А. Василенко, В. Н. Бондарев* Расчет и проектирование технических средств обмена и передачи информации. М.: Высшая школа, 1990. 224 с: ил.
13. *Б. Г. Федорков, В. А. Телец* Микросхемы ЦАП и АЦП: функционирование, параметры, применение. М.: Энергоатомиздат, 1990. 320 с: ил.
14. *Ю. В, Новиков* Функциональные модули контрольно-измерительных систем на базе микроЭВМ// Микропроцессорные средства и системы. 1990.№ 3. с. 75-77.

15. Сопряжение датчиков и устройств ввода данных с компьютерами IBM PC: Пер. с англ./ Под ред. У. Томпкинса, Дж. Уэбстера. М.: Мир, 1992. 592 с: ил.
16. *А. М. Юшин* Цифровые микросхемы для электронных устройств: Справочник. М.: Высшая школа, 1993. 176 с: ил.
17. Логические ИС КР1533, КР1554: Справочник / И. И. Петровский, А. В. Прибыльский, А. А. Троян, В. С. Чувелев: В 2-х ч. М.: "БИНОМ", 1996.
18. The TTL Data Book. Texas Instruments, 1997.
19. *Ю. В. Новиков, О. А. Калашников, С. Э. Гуляев* Разработка устройств сопряжения для персональных компьютеров типа IBM PC/ Под общ ред. Ю. В. Новикова. Практ. пособие. М.: ЭКОМ, 1997. 224 с: ил.
20. *Б. Л. Перельман, В. И. Шевелев* Отечественные микросхемы и зарубежные аналоги: Справочник. М.: НТЦ Микротех, 1998. 376 с: ил.
21. *Ю. В. Новиков, Д. Г. Карпенко* Аппаратура локальных сетей: функции, выбор, разработка/ Под общ. ред. Ю. В. Новикова. М.: ЭКОМ, 1998. 288 с: ил.
22. *Тук М.* Аппаратные средства IBM PC. Энциклопедия. СПб: Питер Ком, 1999. 816 с.: ил.
23. *С. А. Бирюков* Применение цифровых микросхем серий ТТЛ и КМОП. М.: ДМК, 1999. 240 с: ил.
24. *Ю. В. Новиков, С. В. Кондратенко* Локальные сети: архитектура, алгоритмы, проектирование. М.: ЭКОМ, 2000 312 с: ил.
25. *Ю. В. Новиков* Основы цифровой схемотехники. М.: Мир, 2001. 379 с: ил.
26. *Хоровиц П., Хилл У.* Искусство схемотехники. Пер. с англ. 6-е изд. перераб. М.: Мир, 2001. 704 с: ил.
27. *В. Б. Бродин, А. В. Калинин* Системы на микроконтроллерах и БИС программируемой логики. М.: ЭКОМ, 2002. 400 с: ил.
28. *Е. П. Угрюмое* Цифровая схемотехника. Учебное пособие. Спб.: BHV-Санкт-Петербург, 2004. 782 с: ил.
29. *И. М. Мышляева* Цифровая схемотехника. Учебник. М.: Академия, 2005. 400 с: ил.
30. *Ю. Ф. Опадчий, О. П. Глудкин, А. И. Гуров* Аналоговая и цифровая электроника. Полный курс. Учебник для вузов. М.: Горячая линия Телеком, Радио и связь, 2005. 768 с: ил.

Словарь терминов и сокращений

2S (2-States Output) - выход с двумя активными состояниями, 2С, стандартный выход ТТЛ.

3S (3-States Output) — выход с тремя состояниями, 3С.

Adder — сумматор.

AND — логическая функция И.

AC (Alternating Current) — переменный ток.

ALU (Arithmetic and Logic Unit) — АЛУ, арифметико-логическое устройство.

ADC (Analog-to-Digital Converter) — АЦП, аналого-цифровой преобразователь.

ASCII (American Standard Code for Information Interchange) - стандартный американский код обмена символьной информацией.

BCD (Binary-Coded Decimal) — двоично-десятичный код.

BiCMOS (Bipolar Complementary Metal-Oxide-Semiconductor) — биполярно-полевая технология изготовления микросхем, сочетающая на одном кристалле биполярные и КМОП-структуры.

Buffer — буфер.

Bus — шина, магистраль.

CAS (Column-Address Select) — сигнал выбора адреса столбца (в микросхемах динамической памяти).

Chip — микросхема, чип.

Clear — очистка, сброс в нуль.

Clock — тактовый, тактирующий сигнал.

CMOS (Complementary Metal-Oxide-Semiconductor) - комплементарная МОП-технология (КМОП).

Coder (encoder) — шифратор, кодер.

Comparator — компаратор.

Converter — преобразователь.

CRC (Cyclic Redundancy Check) — циклический избыточный контроль, метод вычисления циклической контрольной суммы и сама эта сумма.

Counter — счетчик.

DAC (Digital-to-Analog Converter) - ЦАП, цифро-аналоговый преобразователь.

DC (Direct Current) — постоянный ток.

Decoder — дешифратор, декодер.

Delay - задержка.

DIC (Dual In-line Ceramic package) — керамический корпус микросхемы типа DIL.

DIL (Dual **In-Line** package) — корпус микросхемы с двухрядным вертикальным расположением выводов.

DIMM (Dual In-Line Memory Module) — модуль памяти с двусторонним расположением выводов.

DIP (Dual In-line Plastic package) — пластмассовый корпус микросхемы типа DIL.

DIP Switches — малогабаритные выключатели, смонтированные в корпусе типа DIP.

DRAM (Dynamic RAM) — динамическое ОЗУ.

Driver — выходной буфер, драйвер.

EEPROM (Electrically Erasable Programmable ROM) - ПЗУ с электрическим стиранием и возможностью программирования.

EPROM (Erasable Programmable ROM) - ПЗУ со стиранием (ультрафиолетовым излучением) и перезаписью информации (РПЗУ).

Female — разъем-розетка, гнездо.

FIFO (First In, First Out) — "первым вошел — первым вышел", один из способов организации ОЗУ с последовательным доступом.

Flash memory — разновидность памяти EEPROM, характеризующаяся высокой емкостью, малым потреблением и большим допустимым количеством циклов перезаписи, флэш-память.

Flip-flop — триггер.

Gate — логический элемент, вентиль.

GND (Ground) — общий провод схемы, "земля".

H (High) — высокий уровень сигнала, единичный уровень при положительной логике.

IC (Integrated Circuit) — интегральная микросхема, **ИС**.

Inverter — инвертор.

I/O (Input/Output) — **ввод/вывод** (В/В), **вход/выход**.

Jumper — съемная перемычка, соединяющая штыревые контакты на плате, джампер.

L (Low) — низкий уровень сигнала, нулевой уровень при положительной логике.

Latch — триггер (регистр) типа "защелка".

LCD (Liquid Crystal Display) — жидкокристаллический дисплей, индикатор.

Line driver - драйвер линии, буфер.

LSB (Least Significant Bit) — младший значащий бит (в байте или слове).

LSI (Large Scale Integration) — большая интегральная схема, БИС.

LVT (Low-Voltage Technology) - низковольтная технология микросхем (напряжение питания 3,3 В).

Male - разъем-вилка, штекер.

Master — ведущее, главное устройство, участвующее в обмене информацией, задатчик.

Monostable multivibrator — ждущий мультивибратор, одновибратор.

MSB (Most Significant Bit) — старший значащий бит (в байте или слове).

Multiplexer — мультиплексор.

Multivibrator — мультивибратор.

NAND — логическая функция И-НЕ.

Noninverter — повторитель.

NOR — логическая функция ИЛИ-НЕ.

NVRAM (Non-Volatile RAM) - энергонезависимое ОЗУ, сохраняющее информацию при отключении питания.

OC (Open-Collector Output) - выход микросхемы с открытым коллектором.

OR — логическая функция ИЛИ.

PAL (Programmable Array Logic) - программируемая логическая матрица, ПЛМ.

Parity — четность, паритет.

Plug — разъем типа вилка.

Preset — предварительная установка.

PROM (Programmable ROM) - программируемое ПЗУ, ППЗУ.

Pull-up Resistor — нагрузочный резистор, включаемый между выходом микросхемы и напряжением питания.

RAM (Random Access Memory) — оперативное запоминающее устройство, ОЗУ.

RAS (Row-Address Select) — сигнал выбора адреса строки (в микросхемах динамической памяти).

Receiver — приемник, входной буфер.

Refresh — регенерация (в динамической памяти).

Register — регистр.

Reset — сигнал установки микросхемы или устройства в нулевое или исходное состояние.

ROM (Read-Only Memory) — постоянное запоминающее устройство, ПЗУ.

RxC (Received Clock) — принимаемый синхросигнал.

RxD (Received Data) - принимаемые данные.

Schmitt trigger -триггер Шмитта.

SDRAM (Synchronous Dynamic RAM) — синхронное динамическое ОЗУ.

Set — сигнал установки выхода в единичное состояние.

SIMM (Single In-Line Memory Module) — модуль памяти с однорядным расположением выводов.

SIP (Single In-line Package) — корпус микросхемы с однорядным расположением выводов.

Slave — ведомое, пассивное устройство, участвующее в обмене информацией, исполнитель.

Slot — щелевой разъем для подключения печатных плат с разъемом в виде печатных проводников, слот.

Socket — контактирующее устройство для установки микросхем на плату, сокет.

SRAM (Static RAM) - статическое ОЗУ.

Strobe — стробирующий сигнал, строб.

Terminator — оконечное согласующее устройство на линии связи (обычно —резистор).

TI, ТП (Texas Instruments Inc.) — американская фирма, один из ведущих производителей цифровых микросхем малой и средней степени интеграции.

TR (Terminate Resistor) - нагрузочный резистор для линии связи.

Transceiver - приемопередатчик, трансивер, двунаправленный буфер.

Transmitter — передатчик, выходной буфер.

Trigger — триггер.

TTL (Transistor-Transistor Logic) — транзисторно-транзисторная (биполярная) логика, ТТЛ.

TTLS (Transistor-Transistor Logic Schottky) — транзисторно-транзисторная логика Шоттки, ТТЛШ.

TxC (Transmitted Clock) — передаваемый синхросигнал.

TxD (Transmitted Data) — передаваемые данные.

V — напряжение (Voltage), вольт (Volt).

VLSI (Very Large Scale Integration) — сверхбольшая интегральная схема (СБИС).

Waveform — форма сигнала.

Z (Z-state) — третье (высокоимпедансное) состояние выхода микросхемы.

ZIF (Zero Insertion Force) — разъем или сокет с нулевым усилием вставки.

2С — выход с двумя активными состояниями (нуль и единица), стандартный ТТЛ-совместимый выход.

3С — выход с тремя состояниями (два активных: нуль и единица, третье — пассивное, отключенное), а также само третье состояние выхода, в отличие от двух активных состояний.

Адрес — закодированный номер, определяющий, куда передается информация или откуда она принимается.

Активный уровень сигнала — уровень, соответствующий приходу, наличию сигнала, то есть выполнению этим сигналом соответствующей ему функции.

АЛУ — арифметико-логическое устройство (ALU).

АПЧ — автоматическая подстройка частоты.

Асинхронный сигнал — сигнал, не привязанный по времени к внутренним процессам схемы, не синхронизованный со схемой.

Асинхронный (последовательный) счетчик — счетчик, выходные разряды которого переключаются по очереди, начиная с младшего.

АЦП — аналого-цифровой преобразователь (ADC), преобразующий величину входного аналогового сигнала в выходной цифровой код.

Байт — группа двоичных разрядов, битов (как правило, 8 бит), содержащая какой-то код.

Биполярный сигнал — сигнал, который может быть как положительным, так и отрицательным.

Бит (от англ. Binary Digit — двоичное число) — единица двоичной информации, разряд двоичного кода, принимающий значения 0 и 1.

БИС — большая интегральная схема (LSI).

Ввод данных — то же, что чтение, считывание, прием.

Вилка (штекер) — часть разъема, контакты которого входят в контакты розетки (гнезда).

Вывод данных — то же, что запись, передача.

Выводы микросхемы — металлические контакты на корпусе микросхемы для входных и выходных сигналов, подключения внешних элементов и подачи питания.

Временная диаграмма — графики зависимости от времени входных и выходных сигналов цифрового устройства в различных режимах работы.

Выборка — мгновенное значение аналогового сигнала, которому ставится в соответствие цифровой код.

Гига- (Г) — приставка для обозначения миллиарда, 10^9.

Данные — передаваемая в закодированном виде цифровая информация.

Двоичная система счисления — система счисления по модулю два, в которой разряды числа кодируются 0 или 1 и представляют собой степени числа 2.

Двунаправленная линия (шина) — линия (шина), по которой сигналы могут передаваться в обоих направлениях (по очереди).

Дешифрация — преобразование входного двоичного кода в номер выходного сигнала.

Дорожки - проводники на поверхности печатной платы, для передачи сигналов и подачи питания.

Единичный сигнал — то же, что положительный сигнал.

Задержка - временной сдвиг между входным и выходным сигналами устройства, узла, микросхемы.

Задний фронт сигнала (спад) — переход сигнала из активного уровня в пассивный.

Защелка — триггер или регистр, стробируемый уровнем сигнала и пропускающий входной сигнал на выход при активном управляющем сигнале (стробе).

ЗУ — запоминающее устройство, память.

Импульс — сравнительно короткий сигнал.

Инверсный выход — выход, выдающий сигнал инверсной полярности по сравнению со входным сигналом.

Инвертирование или инверсия сигнала — изменение полярности сигнала.

Интерфейс — соглашение об обмене между электронными устройствами. Включает в себя требования по электрическому, логическому и конструктивному сопряжению устройств.

ИС — интегральная микросхема, ИМС (IC), чип.

Кабель — один или несколько проводов в общей оболочке, используемые для передачи сигналов.

Карта — электронное устройство, выполненное на печатной плате.

КЗ — короткое замыкание.

Кило- (к) - приставка для обозначения тысячи, 10^3.

КМОП — комплементарная технология МОП (CMOS).

Код — информация, передаваемая несколькими двоичными разрядами, битами.

Контактные площадки — проводники на поверхности печатной платы, к которым припаиваются выводы микросхем.

Коэффициент разветвления — число входов, которое может быть подключено к данному выходу без нарушения его работы. Определяется отношением выходного тока к входному. Стандартная величина коэффициента разветвления при использовании микросхем одной серии равна 10.

Линия (линия связи) — проводник (электрический или оптоволоконный), передающий сигнал.

Меандр — сигнал со скважностью, равной двум, то есть длительность импульсов равна длительности паузы между ними.

Мега- (М) — приставка для обозначения миллиона, 10^6.

Микро- (мк) — приставка для обозначения одной миллионной доли, 10^{-6}.

Милли- (м) — приставка для обозначения одной тысячной доли, 10^{-3}.

МОП — полупроводниковая технология на основе полевых транзисторов типа "металл — окисел — полупроводник" (MOS).

Мультиплексирование — передача различных сигналов по одной линии (шине) в разные моменты времени.

Нагрузочная способность — параметр выхода микросхемы, характеризующий величину выходного тока, которую может выдать в нагрузку данный выход без нарушения его работы. Чаще всего нагрузочная способность прямо связана с коэффициентом разветвления.

Нано- (н) — приставка для обозначения одной миллиардной доли, 10^{-9}.

Нулевой сигнал — то же, что отрицательный сигнал.

Обратная связь — передача сигнала или его части с выхода схемы на ее вход или один из ее входов.

Обратный (инверсный) счет - счет на уменьшение выходного кода.

ОЗУ — оперативное запоминающее устройство, оперативная память (RAM).

ОК — выход с открытым коллектором.

Опорное напряжение — напряжение эталонного уровня, с которым сравнивается входной сигнал (в АЦП), или из которого формируется выходной сигнал (в ЦАП).

Отрицательная логика — система сигналов, в которой логической единице соответствует низкий уровень напряжения, а логическому нулю — высокий.

Отрицательный сигнал (сигнал отрицательной полярности, нулевой сигнал) — сигнал, активный уровень которого — логический нуль. То есть единица — это отсутствие сигнала, нуль — сигнал пришел.

Отрицательный фронт сигнала (спад) — переход сигнала из единицы (из высокого уровня) в нуль (в низкий уровень).

Пассивный уровень сигнала — уровень, в котором сигнал не выполняет никакой функции.

Перепад (переход) сигнала — переключение сигнала из нуля в единицу или из единицы в нуль, то же что фронт сигнала.

Передний фронт сигнала — переход сигнала из пассивного уровня в активный.

ПЗУ — постоянное запоминающее устройство, постоянная память (ROM).

Пико- (п) — приставка для обозначения одной триллионной доли, 10^{-12}.

ПЛИС — программируемая логическая интегральная микросхема.

ПЛМ — программируемая логическая матрица (PAL).

Погрешность абсолютная — разность между измеренной величиной и ее истинным значением (погрешность измерения); разность между сформированной величиной и требуемым ее значением (погрешность формирования, погрешность воспроизведения).

Погрешность относительная — отношение абсолютной погрешности к требуемому (или истинному) значению данной величины. Часто измеряется в процентах.

Положительная логика — система сигналов, в которой логической единице соответствует высокий уровень напряжения, а логическому нулю — низкий.

Положительный сигнал (сигнал положительной полярности, единичный сигнал) — сигнал, активный уровень которого — логическая единица. То есть нуль — это отсутствие сигнала, единица — сигнал пришел.

Положительный фронт сигнала (или просто фронт) — переход сигнала из нуля (из низкого уровня) в единицу (в высокий уровень).

Полярность сигнала — уровень сигнала, соответствующий его активности. Положительной полярности соответствует активный единичный сигнал, отрицательной полярности — активный нулевой сигнал.

Помехи — паразитные сигналы, накладывающиеся на информационные сигналы и искажающие их. Помехи могут наводиться извне (электромагнитным полем), а также возникать в цепях питания.

Помехозащищенность — параметр, характеризующий величину входного сигнала помехи, который еще не может изменить состояние выходных сигналов. Определяется разницей между напряжением U_{IH} и порогом срабатывания (помехозащищенность единичного уровня), а также разницей между порогом срабатывания и U_{IL} (помехозащищенность нулевого уровня).

Порог срабатывания — уровень входного напряжения, выше которого сигнал воспринимается как единица, а ниже — как нуль. Для ТТЛ микросхем он примерно равен 1,3... 1,4 В.

ППЗУ - программируемое ПЗУ (PROM).

Принципиальная схема — наиболее подробная схема электронного устройства с указанием всех элементов, связей, входов и выходов, выполненная в соответствии со стандартом.

Протокол — порядок обмена сигналами между цифровыми устройствами.

Прямой выход — выход, выдающий сигнал положительной полярности.

Прямой счет — счет на увеличение выходного кода.

Разрядность (кода, шины) — количество двоичных разрядов кода или количество цифровых сигналов для передачи кода по шине.

Разъем — разъемное контактирующее устройство из двух частей (розетка и вилка), служащее для передачи сигналов и питания электронных схем.

Реверсивный счетчик — счетчик, работающий как в режиме прямого счета, так и в режиме обратного (инверсного) счета.

Регенерация — периодическое восстановление, освежение информации, записанной в динамическую память.

Розетка (гнездо) — часть разъема, в контакты которого входят контакты вилки (штекера).

РПЗУ — репрограммируемое ПЗУ (EPROM), информация в котором стирается ультрафиолетовым излучением и может быть записана вновь.

СБИС — сверхбольшая интегральная схема (VLSI).

Слово (двоичное) — группа бит (обычно 16, 32 или 64 бита), состоящая из нескольких байт.

Синхронизация - обеспечение согласованной во времени работы нескольких устройств, например, по общему тактовому сигналу.

Синхронный (параллельный) счетчик — счетчик, все разряды которого переключаются одновременно, синхронно с тактовым сигналом.

Синхросигнал — то же, что тактовый сигнал.

Скважность — отношение периода следования импульсов к длительности этих импульсов.

Сокет (Socket) — то же, что колодка, контактирующее устройство-гнездо, в которое устанавливается микросхема с возможностью простой ее замены.

Спад сигнала — то же, что задний фронт сигнала (обычно — отрицательный фронт).

Строб (стробирующий сигнал) — управляющий сигнал, который своим уровнем определяет момент выполнения элементом или узлом его функции. В более общем смысле строб — это любой синхронизирующий сигнал, тактовый сигнал.

Стробирование — согласование во времени работы узлов и устройств с помощью строба.

Структурная схема — упрощенная схема электронного устройства, показывающая только основные узлы и важнейшие связи между ними.

Схема — электронный узел, устройство, а также их изображение на чертеже.

Такт — то же, что тактовый сигнал, а также период тактового сигнала.

Тактовый сигнал — управляющий сигнал, который своим фронтом определяет момент выполнения элементом или узлом его функции. Иногда то же, что и стробирующий сигнал.

Тера- (Т) — приставка для обозначения триллиона, 10^{12}.

Тетрада (полубайт, ниббл) — группа из четырех бит, кодируемая одним символом в 16-ричной системе счисления.

Точность — показатель близости функционирования данного устройства к идеалу, обычно измеряется с помощью величины погрешности.

ТТЛ — транзисторно-транзисторная логика и соответствующая ей полупроводниковая технология (TTL).

ТТЛШ — технология ТТЛ с диодами Шоттки (TTLS). Характеризуется более высоким быстродействием при той же потребляемой мощности.

Узел — часть электронного устройства, выполняющая четко выделенную функцию или несколько взаимосвязанных функций.

Устройство (электронное) — функционально (а иногда и конструктивно) законченная электронная схема.

Флэш-память (Flash Memory) — разновидность РПЗУ с электрическим стиранием информации и возможностью многократной перезаписи.

Фронт сигнала — переход сигнала из нуля в единицу или из единицы в нуль, иногда, в более узком значении, "передний положительный фронт".

Функциональная схема - не слишком подробная схема электронного устройства, показывающая подробно только схемы отдельных, принципиально важных узлов устройства.

ЦАП — цифро-аналоговый преобразователь (DAC), преобразующий входной цифровой код в величину аналогового сигнала (тока или напряжения).

Цепочка — последовательное соединение нескольких узлов (или устройств), при котором выходы предыдущего узла соединяются со входами последующего узла. Также цепочкой называют любое соединение электронных компонентов (например, RC-цепочка, LC-цепочка).

Цикл — последовательность обмена сигналами, в течение которого выполняется только одна операция.

Чип — то же, что интегральная микросхема, ИМС.

Шина — группа сигнальных линий, объединенных по какому-то принципу, например, шиной называют сигналы, соответствующие всем разрядам какого-то двоичного кода (шина данных, шина адреса). Иногда шиной называют также провод питания ("шина питания") и общий провод ("шина земли").

Шифрация - преобразование номера приходящего входного сигнала в выходной двоичный код.

Ячейка (памяти) — элемент памяти (одноразрядный или многоразрядный), который служит для хранения информационного кода и может быть выбран с помощью кода адреса памяти.

Учебное издание

Новиков Юрий Витальевич
ВВЕДЕНИЕ В ЦИФРОВУЮ СХЕМОТЕХНИКУ
Учебное пособие

Литературный редактор *С. Перепелкина*
Корректор *Ю. Голомазова*
Компьютерная верстка *А. Савин*
Обложка *М. Автономова*

Подписано в печать 05.10.2006. Формат 60x90 1/16.
Гарнитура Таймс. Бумага офсетная. Печать офсетная.
Усл. печ. л. 21,5. Тираж 2000 экз. Заказ № 5437

ООО «ИНТУИТ.ру»
Интернет-Университет Информационных Технологий, www.intuit.ru
Москва, Электрический пер., 8, стр.3.
E-mail: admin@intuit.ru, http://www.intuit.ru

ООО «БИНОМ. Лаборатория знаний»
Москва, проезд Аэропорта, д. 3
Телефон: (499) 157-1902, (495) 157-5272
E-mail: Lbz@aha.ru, http://www.Lbz.ru

При участии ООО «ПФ «Сашко»

Отпечатано в ОАО «ИПК «Ульяновский Дом печати»
432980, г. Ульяновск, ул. Гончарова, 14

Таблица 3.1. Таблица истинности дешифратора 3-8 (ИД7)

Входы						Выходы							
C1	-C2	-C3	4	2	1	0	1	2	3	4	5	6	7
0	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	1	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	0	0	1	1	1	1	0	1	1	1
1	0	0	1	0	1	1	1	1	1	1	0	1	1
1	0	0	1	1	0	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0

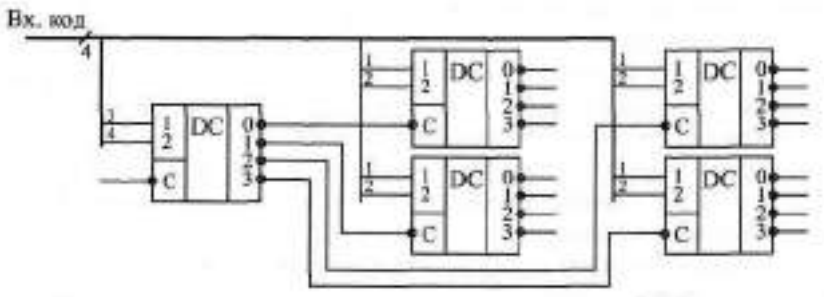


Рис. 3.3. Увеличение количества разрядов дешифратора

раторов. На объединенные входы этих дополнительных дешифраторов подаются младшие разряды входного кода. Из пяти микросхем дешифраторов 2-4 можно получить дешифратор 4-16, как показано на рисунке (хотя лучше, конечно, взять готовую микросхему). Точно так же из девяти микросхем 3-8 можно получить дешифратор 6-64, а из семнадцати микросхем 4-16 — дешифратор 8-256. Еще одно распространенное применение дешифраторов — селекция (выбор) заданных входных кодов. Появление отрицательного сигнала на выбранном выходе дешифратора будет означать поступление на вход интересующего нас кода. В данном случае увеличивать число разрядов входного селектируемого кода гораздо проще, чем в предыдущем (см. рис. 3.3). Например, две микросхемы

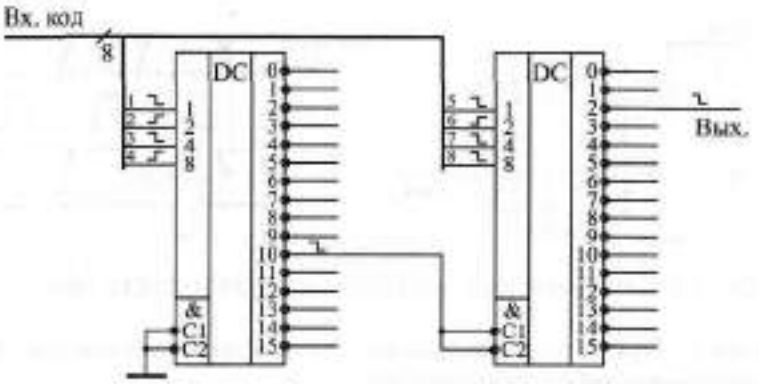


Рис. 3.4. Селектирование кода на дешифраторах

4-16 позволяют селектировать 8-разрядный код (рис. 3.4). В примере на рисунке селектируется 16-ричный код 2А (двоичный код 0010 1010). При этом один дешифратор работает с младшими четырьмя разрядами кода, а другой — со старшими четырьмя разрядами. Объединяются дешифраторы так, что один из них разрешает работу другого по входам -С1 и -С2. Применяя механические переключатели выходов дешифраторов (тумблеры, перемычки), можно легко изменять код, селектируемый данной схемой.

Еще одно важное применение дешифраторов состоит в перекоммутации одного входного сигнала на несколько выходов. Или, другими словами, дешифратор в данном случае выступает в качестве демультиплексора входных сигналов, который позволяет разделить входные сигналы, приходящие в разные моменты времени, на одну входную линию (мультиплексированные сигналы). При этом входы 1, 2, 4, 8 дешифратора используются в качестве управляющих, адресных, определяющих, на какой выход переслать пришедший в данный момент входной сигнал (рис. 3.5), а один из входов С выступает в роли входного сигнала, который пересылается на заданный выход. Если у микросхемы имеется несколько стро-

Рис. 3.5. Включение дешифратора как демультиплексора

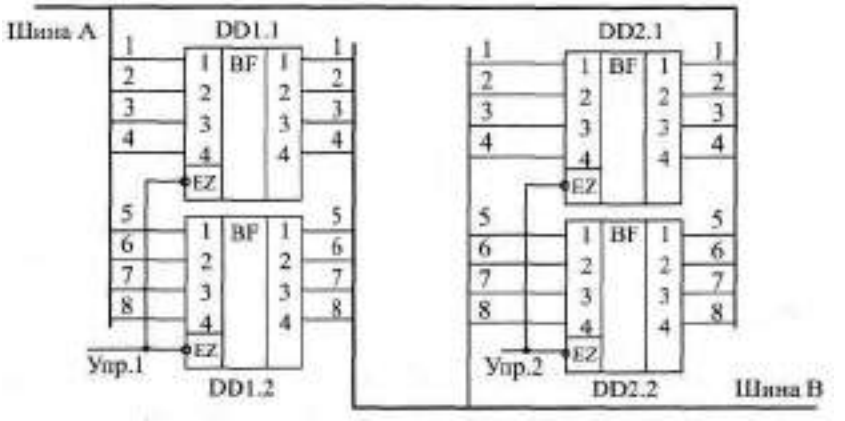


Рис. 2.12. Организация двунаправленной передачи с помощью однонаправленных буферов

Двунаправленную передачу можно организовать и на основе однонаправленных буферов. На рис. 2.12 показано, как это можно сделать на двух микросхемах АП5. Здесь при нулевом сигнале Упр. 1 информация будет передаваться с шины А на шину В, а при нулевом сигнале на входе Упр. 2 — с шины В на шину А. Если оба входа Упр. 1 и Упр. 2 находятся в единичном состоянии, то шины А и В отключены друг от друга, а подача нулей на оба входа Упр. 1 и Упр. 2 должна быть исключена, иначе состояние обеих шин А и В будет не определено.

Микросхемы буферов в отечественных сериях имеют разнообразные обозначения: ЛН, ЛП, АП, ИП, что порой затрудняет их выбор. Например, ЛН6, ЛП8, ЛП11, АП5, АП6, ИП5, ИП6. Буферы с буквами ЛН имеют инверсию, буферы АП и ИП могут быть с инверсией и без инверсии. Все параметры у буферов довольно близки, отличие — в инверсии, в количестве разрядов и в управляющих сигналах.

Временные параметры буферов включают помимо задержки сигнала от информационного входа до информационного выхода, также задержки перехода выхода в третье состояние и из третьего состояния в активное состояние (t_{PHZ}, t_{PLZ} и t_{PZH}, t_{PZL}). Величины этих задержек обычно примерно вдвое больше, чем величины задержек между информационным входом и выходом.

Отключаемый выход буферов (как ОК, так и ЗС) требует применения нагрузочных резисторов. В противном случае вход, подключенный к отключенному выходу, оказывается подвешенным, в результате чего схема может работать неустойчиво, давать сбои. Подключение резистора в случае выхода ОК (pull-up) производится стандартным способом (см. рис. 2.8). Точно так же может быть включен резистор между выходом ЗС и на-

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ И КОНТРОЛЛЕРЫ

Методические указания к самостоятельным работам

Направление подготовки 15.03.04 Автоматизация технологических процессов и производств

Направленность (профиль) «Информационно-управляющие системы »

Квалификация выпускника – бакалавр

Невинномысск 2022

Методические указания предназначены для студентов направления подготовки 15.03.04 Автоматизация технологических процессов и производств и других технических специальностей. Они содержат рекомендации по организации самостоятельных работ студента для дисциплины «Вычислительные машины и контроллеры».

Методические указания разработаны в соответствии с требованиями ФГОС ВО в части содержания и уровня подготовки выпускников направления 15.03.04 Автоматизация технологических процессов и производств

Содержание

1 Подготовка к лекциям

Главное в период подготовки к лекционным занятиям – научиться методам самостоятельного умственного труда, сознательно развивать свои творческие способности и овладевать навыками творческой работы. Для этого необходимо строго соблюдать дисциплину учебы и поведения. Четкое планирование своего рабочего времени и отдыха является необходимым условием для успешной самостоятельной работы. В основу его нужно положить рабочие программы изучаемых в семестре дисциплин.

Каждому студенту следует составлять еженедельный и семестровый планы работы, а также план на каждый рабочий день. С вечера всегда надо распределять работу на завтрашний день. В конце каждого дня целесообразно подводить итог работы: тщательно проверить, все ли выполнено по намеченному плану, не было ли каких-либо отступлений, а если были, по какой причине это произошло. Нужно осуществлять самоконтроль, который является необходимым условием успешной учебы. Если что-то осталось невыполненным, необходимо изыскать время для завершения этой части работы, не уменьшая объема недельного плана.

Слушание и запись лекций – сложный вид вузовской аудиторной работы. Внимательное слушание и конспектирование лекций предполагает интенсивную умственную деятельность студента. Краткие записи лекций, их конспектирование помогает усвоить учебный материал. Конспект является полезным тогда, когда записано самое существенное, основное и сделано это самим студентом. Не надо стремиться записать дословно всю лекцию. Такое «конспектирование» приносит больше вреда, чем пользы. Запись лекций рекомендуется вести по возможности собственными формулировками. Желательно запись осуществлять на одной странице, а следующую оставлять для проработки учебного материала самостоятельно в домашних условиях.

Конспект лекций лучше подразделять на пункты, параграфы, соблюдая красную строку. Этому в большой степени будут способствовать пункты плана лекции, предложенные преподавателям. Принципиальные места,

определения, формулы и другое следует сопровождать замечаниями «важно», «особо важно», «хорошо запомнить» и т.п. Можно делать это и с помощью разноцветных маркеров или ручек. Лучше если они будут собственными, чтобы не приходилось присить их у однокурсников и тем самым не отвлекать их во время лекции. Целесообразно разработать собственную «маркографию» (значки, символы), сокращения слов. Не лишним будет и изучение основ стенографии. Работая над конспектом лекций, всегда необходимо использовать не только учебник, но и ту литературу, которую дополнительно рекомендовал лектор. Именно такая серьезная, кропотливая работа с лекционным материалом позволит глубоко овладеть знаниями.

2 Подготовка к лабораторным работам

Подготовку к каждому практическому занятию студент должен начать с ознакомления с методическими указаниями, которые включают содержание работы. Тщательное продумывание и изучение вопросов основывается на проработке текущего материала лекции, а затем изучения обязательной и дополнительной литературы, рекомендованную к данной теме. На основе индивидуальных предпочтений студенту необходимо самостоятельно выбрать тему доклада по проблеме и по возможности подготовить по нему презентацию.

Если программой дисциплины предусмотрено выполнение практического задания, то его необходимо выполнить с учетом предложенной инструкции (устно или письменно). Все новые понятия по изучаемой теме необходимо выучить наизусть и внести в глоссарий, который целесообразно вести с самого начала изучения курса. Результат такой работы должен проявиться в способности студента свободно ответить на теоретические вопросы семинара, его выступлении и участии в коллективном обсуждении вопросов изучаемой темы, правильном выполнении практических заданий и контрольных работ.

В зависимости от содержания и количества отведенного времени на изучение каждой темы практическое занятие может состоять из четырех-пяти частей:

1. Обсуждение теоретических вопросов, определенных программой дисциплины.

2. Доклад и/ или выступление с презентациями по выбранной проблеме.

3. Обсуждение выступлений по теме – дискуссия.

4. Выполнение практического задания с последующим разбором полученных результатов или обсуждение практического задания.

5. Подведение итогов занятия.

Первая часть – обсуждение теоретических вопросов – проводится в виде фронтальной беседы со всей группой и включает выборочную проверку преподавателем теоретических знаний студентов. Примерная продолжительность — до 15 минут. Вторая часть — выступление студентов с докладами,

которые должны сопровождаться презентациями с целью усиления наглядности восприятия, по одному из вопросов практического занятия. Обязательный элемент доклада – представление и анализ статистических данных, обоснование социальных последствий любого экономического факта, явления или процесса. Примерная продолжительность — 20-25 минут. После докладов следует их обсуждение – дискуссия. В ходе этого этапа практического занятия могут быть заданы уточняющие вопросы к докладчикам. Примерная продолжительность – до 15-20 минут. Если программой предусмотрено выполнение практического задания в рамках конкретной темы, то преподавателями определяется его содержание и дается время на его выполнение, а замет идет обсуждение результатов. Подведением итогов заканчивается практическое занятие.

В процессе подготовки к практическим занятиям, студентам необходимо обратить особое внимание на самостоятельное изучение рекомендованной учебно-методической (а также научной и популярной) литературы. Самостоятельная работа с учебниками, учебными пособиями, научной, справочной и популярной литературой, материалами периодических изданий и Интернета, статистическими данными является наиболее эффективным методом получения знаний, позволяет значительно активизировать процесс овладения информацией, способствует более глубокому усвоению изучаемого материала, формирует у студентов свое отношение к конкретной проблеме. Более глубокому раскрытию вопросов способствует знакомство с дополнительной литературой, рекомендованной преподавателем по каждой теме семинарского или практического занятия, что позволяет студентам проявить свою индивидуальность в рамках выступления на данных занятиях, выявить широкий спектр мнений по изучаемой проблеме.

3 Подготовка к практическим занятиям

Подготовку к каждому практическому занятию студент должен начать с ознакомления с методическими указаниями, которые включают содержание работы. Тщательное продумывание и изучение вопросов основывается на проработке текущего материала лекции, а затем изучения обязательной и дополнительной литературы, рекомендованную к данной теме. На основе индивидуальных предпочтений студенту необходимо самостоятельно выбрать тему доклада по проблеме и по возможности подготовить по нему презентацию.

Если программой дисциплины предусмотрено выполнение практического задания, то его необходимо выполнить с учетом предложенной инструкции (устно или письменно). Все новые понятия по изучаемой теме необходимо выучить наизусть и внести в глоссарий, который целесообразно вести с самого начала изучения курса. Результат такой работы должен проявиться в способности студента свободно ответить на теоретические вопросы семинара, его выступлении и участии в коллективном обсуждении вопросов изучаемой темы, правильном выполнении практических заданий и контрольных работ.

В зависимости от содержания и количества отведенного времени на изучение каждой темы практическое занятие может состоять из четырех-пяти частей:

1. Обсуждение теоретических вопросов, определенных программой дисциплины.

2. Доклад и/ или выступление с презентациями по выбранной проблеме.

3. Обсуждение выступлений по теме – дискуссия.

4. Выполнение практического задания с последующим разбором полученных результатов или обсуждение практического задания.

5. Подведение итогов занятия.

Первая часть – обсуждение теоретических вопросов – проводится в виде фронтальной беседы со всей группой и включает выборочную проверку преподавателем теоретических знаний студентов. Примерная продолжительность

— до 15 минут. Вторая часть — выступление студентов с докладами, которые должны сопровождаться презентациями с целью усиления наглядности восприятия, по одному из вопросов практического занятия. Обязательный элемент доклада – представление и анализ статистических данных, обоснование социальных последствий любого экономического факта, явления или процесса. Примерная продолжительность — 20-25 минут. После докладов следует их обсуждение – дискуссия. В ходе этого этапа практического занятия могут быть заданы уточняющие вопросы к докладчикам. Примерная продолжительность – до 15-20 минут. Если программой предусмотрено выполнение практического задания в рамках конкретной темы, то преподавателями определяется его содержание и дается время на его выполнение, а замет идет обсуждение результатов. Подведением итогов заканчивается практическое занятие.

В процессе подготовки к практическим занятиям, студентам необходимо обратить особое внимание на самостоятельное изучение рекомендованной учебно-методической (а также научной и популярной) литературы. Самостоятельная работа с учебниками, учебными пособиями, научной, справочной и популярной литературой, материалами периодических изданий и Интернета, статистическими данными является наиболее эффективным методом получения знаний, позволяет значительно активизировать процесс овладения информацией, способствует более глубокому усвоению изучаемого материала, формирует у студентов свое отношение к конкретной проблеме. Более глубокому раскрытию вопросов способствует знакомство с дополнительной литературой, рекомендованной преподавателем по каждой теме семинарского или практического занятия, что позволяет студентам проявить свою индивидуальность в рамках выступления на данных занятиях, выявить широкий спектр мнений по изучаемой проблеме.

4 Самостоятельное изучение темы. Конспект

Конспект – наиболее совершенная и наиболее сложная форма записи. Слово «конспект» происходит от латинского «conspectus», что означает «обзор, изложение». В правильно составленном конспекте обычно выделено самое основное в изучаемом тексте, сосредоточено внимание на наиболее существенном, в кратких и четких формулировках обобщены важные теоретические положения.

Конспект представляет собой относительно подробное, последовательное изложение содержания прочитанного. На первых порах целесообразно в записях ближе держаться тексту, прибегая зачастую к прямому цитированию автора. В дальнейшем, по мере выработки навыков конспектирования, записи будут носить более свободный и сжатый характер.

Конспект книги обычно ведется в тетради. В самом начале конспекта указывается фамилия автора, полное название произведения, издательство, год и место издания. При цитировании обязательная ссылка на страницу книги. Если цитата взята из собрания сочинений, то необходимо указать соответствующий том. Следует помнить, что четкая ссылка на источник – непременное правило конспектирования. Если конспектируется статья, то указывается, где и когда она была напечатана.

Конспект подразделяется на части в соответсвтии с заранее продуманным планом. Пункты плана записываются в тексте или на полях конспекта. Писать его рекомендуется четко и разборчиво, так как небрежная запись с течением времени становиться малопонятной для ее автора. Существует правило: конспект, составленный для себя, должен быть по возможности написан так, чтобы его легко прочитал и кто-либо другой.

Формы конспекта могут быть разными и зависят от его целевого назначения (изучение материала в целом или под определенным углом зрения, подготовка к докладу, выступлению на занятии и т.д.), а также от характера произведения (монография, статья, документ и т.п.). Если речь идет просто об

изложении содержания работы, текст конспекта может быть сплошным, с выделением особо важных положений подчеркиванием или различными значками.

В случае, когда не ограничиваются переложением содержания, а фиксируют в конспекте и свои собственные суждения по данному вопросу или дополняют конспект соответствующими материалами их других источников, следует отводить место для такого рода записей. Рекомендуется разделить страницы тетради пополам по вертикали и в левой части вести конспект произведения, а в правой свои дополнительные записи, совмещая их по содержанию.

Конспектирование в большей мере, чем другие виды записей, помогает вырабатывать навыки правильного изложения в письменной форме важные теоретических и практических вопросов, умение четко их формулировать и ясно излагать своими словами.

Таким образом, составление конспекта требует вдумчивой работы, затраты времени и труда. Зато во время конспектирования приобретаются знания, создается фонд записей.

Конспект может быть текстуальным или тематическим. В текстуальном конспекте сохраняется логика и структура изучаемого произведения, а запись ведется в соответствии с расположением материала в книге. За основу тематического конспекта берется не план произведения, а содержание какой-либо темы или проблемы.

Текстуальный конспект желательно начинать после того, как вся книга прочитана и продумана, но это, к сожалению, не всегда возможно. В первую очередь необходимо составить план произведения письменно или мысленно, поскольку в соответствии с этим планом строится дальнейшая работа. Конспект включает в себя тезисы, которые составляют его основу. Но, в отличие от тезисов, конспект содержит краткую запись не только выводов, но и доказательств, вплоть до фактического материала. Иначе говоря, конспект – это

расширенные тезисы, дополненные рассуждениями и доказательствами, мыслями и соображениями составителя записи.

Как правило, конспект включает в себя и выписки, но в него могут войти отдельные места, цитируемые дословно, а также факты, примеры, цифры, таблицы и схемы, взятые из книги. Следует помнить, что работа над конспектом только тогда будет творческой, когда она не ограничена текстом изучаемого произведения. Нужно дополнять конспект данными из другими источников.

В конспекте необходимо выделять отдельные места текста в зависимости от их значимости. Можно пользоваться различными способами: подчеркиваниями, вопросительными и восклицательными знаками, репликами, краткими оценками, писать на полях своих конспектов слова: «важно», «очень важно», «верно», «характерно».

В конспект могут помещаться диаграммы, схемы, таблицы, которые придадут ему наглядность.

Составлению тематического конспекта предшествует тщательное изучение всей литературы, подобранной для раскрытия данной темы. Бывает, что какая-либо тема рассматривается в нескольких главах или в разных местах книги. А в конспекте весь материал, относящийся к теме, будет сосредоточен в одном месте. В плане конспекта рекомендуется делать пометки, к каким источникам (вплоть до страницы) придется обратиться для раскрытия вопросов. Тематический конспект составляется обычно для того, чтобы глубже изучить определенный вопрос, подготовиться к докладу, лекции или выступлению на семинарском занятии. Такой конспект по содержанию приближается к реферату, докладу по избранной теме, особенно если включает и собственный вклад в изучение проблемы.

4 Подготовка к экзамену

Экзаменационная сессия – очень тяжелый период работы для студентов и ответственный труд для преподавателей. Главная задача экзаменов – проверка качества усвоения содержания дисциплины.

На основе такой проверки оценивается учебная работа не только студентов, но и преподавателей: по результатам экзаменов можно судить и о качестве всего учебного процесса. При подготовке к экзамену студенты повторяют материал курсов, которые они слушали и изучали в течение семестра, обобщают полученные знания, выделяют главное в предмете, воспроизводят общую картину для того, чтобы яснее понять связь между отдельными элементами дисциплины.

При подготовке к экзаменам основное направление дают программы курса и конспект, которые указывают, что в курсе наиболее важно. Основной материал должен прорабатываться по учебнику, поскольку конспекта недостаточно для изучения дисциплины. Учебник должен быть проработан в течение семестра, а перед экзаменом важно сосредоточить внимание на основных, наиболее сложных разделах. Подготовку по каждому разделу следует заканчивать восстановлением в памяти его краткого содержания в логической последовательности.

До экзамена обычно проводится консультация, но она не может возместить отсутствия систематической работы в течение семестра и помочь за несколько часов освоить материал, требующийся к экзамену. На консультации студент получает лишь ответы на трудные или оставшиеся неясными вопросы. Польза от консультации будет только в том случае, если студент до нее проработает весь материал. Надо учиться задавать вопросы, вырабатывать привычку пользоваться справочниками, энциклопедиями, а не быть на иждивении у преподавателей, который не всегда может тут же, «с ходу» назвать какой-либо факт, имя, событие. На экзамене нужно показать не только знание предмета, но и умение логически связно построить устный ответ.

Получив билет, надо вдуматься в поставленные вопросы для того, чтобы правильно понять их. Нередко студент отвечает не на тот вопрос, который поставлен, или в простом вопросе ищет скрытого смысла. Не поняв вопроса и не обдумав план ответа, не следует начинать писать. Конспект своего ответа надо рассматривать как план краткого сообщения на данную тему и составлять ответ нужно кратко. При этом необходимо показать умение выражать мысль четко и доходчиво.

Отвечать нужно спокойно, четко, продуманно, без торопливости, придерживаясь записи своего ответа. На экзаменах студент показывает не только свои знания, но и учится владеть собой. После ответа на билет могут следовать вопросы, которые имеют целью выяснить понимание других разделов курса, не вошедших в билет. Как правило, на них можно ответить кратко, достаточно показать знание сути вопроса. Часто студенты при ответе на дополнительные вопросы проявляют поспешность: не поняв смысла того, что у них спрашивают, начинают отвечать и нередко говорят не по сути.

Следует помнить, что необходимым условием правильного режима работы в период экзаменационной сессии является нормальный сон, поэтому подготовка к экзаменам не должна быть в ущерб сну. Установлено, что сильное эмоциональное напряжение во время экзаменов неблагоприятно отражается на нервной системе и многие студенты из-за волнений не спят ночи перед экзаменами. Обычно в сессию студенту не до болезни, так как весь организм озабочен одним - сдать экзамены. Но это еще не значит, что последствия неправильно организованного труда и чрезмерной занятости не скажутся потом. Поэтому каждый студент помнить о важности рационального распорядка рабочего дня и о своевременности снятия или уменьшения умственного напряжения.