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

Методические указания

по выполнению лабораторных работ
по дисциплине «Микропроцессорные системы управления»
Для студентов направления подготовки 15.03.04 Автоматизация технологических процессов и производств, направленность (профиль) Информационно-управляющие системы
(ЧАСТЬ 1)

(ЭЛЕКТРОННЫЙ ДОКУМЕНТ)

СОДЕРЖАНИЕ

Введение

Платформа Arduino позиционируется как программно-аппаратная система начального обучения программированию микро-ЭВМ. Базовой моделью микро-ЭВМ изначально являлась микросхема ATMega8, однако впоследствии были использованы улучшенные модификации ATMega168 и ATMega328. В настоящее время обозначилась тенденция внедрения в среду разработки IDE Arduino микро-ЭВМ иной архитектуры, в том числе на базе ARM и Intel x86.

Простые программные проекты для встраиваемых систем на базе микроЭВМ пишутся, как правило, на языке ассемблера. Для более сложных проектов более адекватным представляется использование трансляторов с языков высокого уровня, причем в последнее время предпочтение чаще отдается языку С.

Язык С достаточно сложен для освоения, поэтому для платформы Arduino была разработана его упрощенная версия [1], [4]. Программы при этом разрабатываются с среде программирования IDE Arduino, которая пригодна для развертывания в операционных системах Windows и Linux.

При изучении среды программирования Arduino используется IDE Arduino верси1.0 или более поздней.

Назначением приведенных лабораторных работ являются:

- изучение средств разработки и отладки программного обеспечения для микропроцессорных систем;
- формирование навыков работы в среде программирования Arduino.

Лабораторная работа 1 Развертывание системы программирования IDE Arduino

Цель работы

Подготовка к дальнейшему выполнению цикла лабораторных работ в среде IDE Arduino и приобретение практических навыков в подготовке типичной рабочей среды проектирования программ для микро-ЭВМ для операционной системы Windows.

Подготовка к работе

- изучить основные аппаратные особенности платформы Arduino;
- ознакомиться с инструкцией по подключению платы Arduino к персональному компьютеру;
- уточнить место расположения дистрибутива IDE Arduino;
- уточнить точный тип и место расположения драйвера виртуального COM-порта (если плата не использует интерфейс RS-232 явным образом).
- изучить особенности установки интегрированной среды IDE Arduino [1];
- изучить способы настройки режимов работы последовательных каналов в ОС Windows;
- убедиться в отсутствии уже установленного программного продукта IDE Arduino;
- составить пошаговый план установки среды IDE Arduino с учетом возможностей и особенностей настроек предоставленного персонального компьютера.

Вопросы для самоконтроля

- какую роль играет последовательный канал (COM-port) персонального компьютера в процессе работы в IDE Arduino;
- возможна ли установка и эксплуатация IDE Arduino на персональных компьютерах, не имеющих каналов COM-port.

- какой интерфейс для подключения к персональному компьютеру (ПК) используется в выданной плате микроконтроллера?
- как осуществляется питание платы исследуемого микроконтроллера?
- что такое «виртуальный COM-порт»?

Программа работ

(Для версии платы Arduino с USB интерфейсом)

1. Запустить на исполнение дистрибутив IDE Arduino, выполнить требуемые действия для его установки и убедиться в том, что программа установлена без ошибок.
2. Установить драйвер виртуального COM-порта и убедится в том, что он установлен без ошибок.
3. Подключить плату микроконтроллера к персональному компьютеру
4. Запустить среду IDE Arduino
5. Осуществить настройку IDE Arduino, указав тип подключенной платы и номер зарегистрированного операционной системой COM-порта.
6. Проверить функционирование системы путем загрузки демонстрационной программы **blink**.

Методические указания

Определить номер виртуального COM-порта можно с использование Диспетчера устройств ОС Windows (см. рисунок ниже).

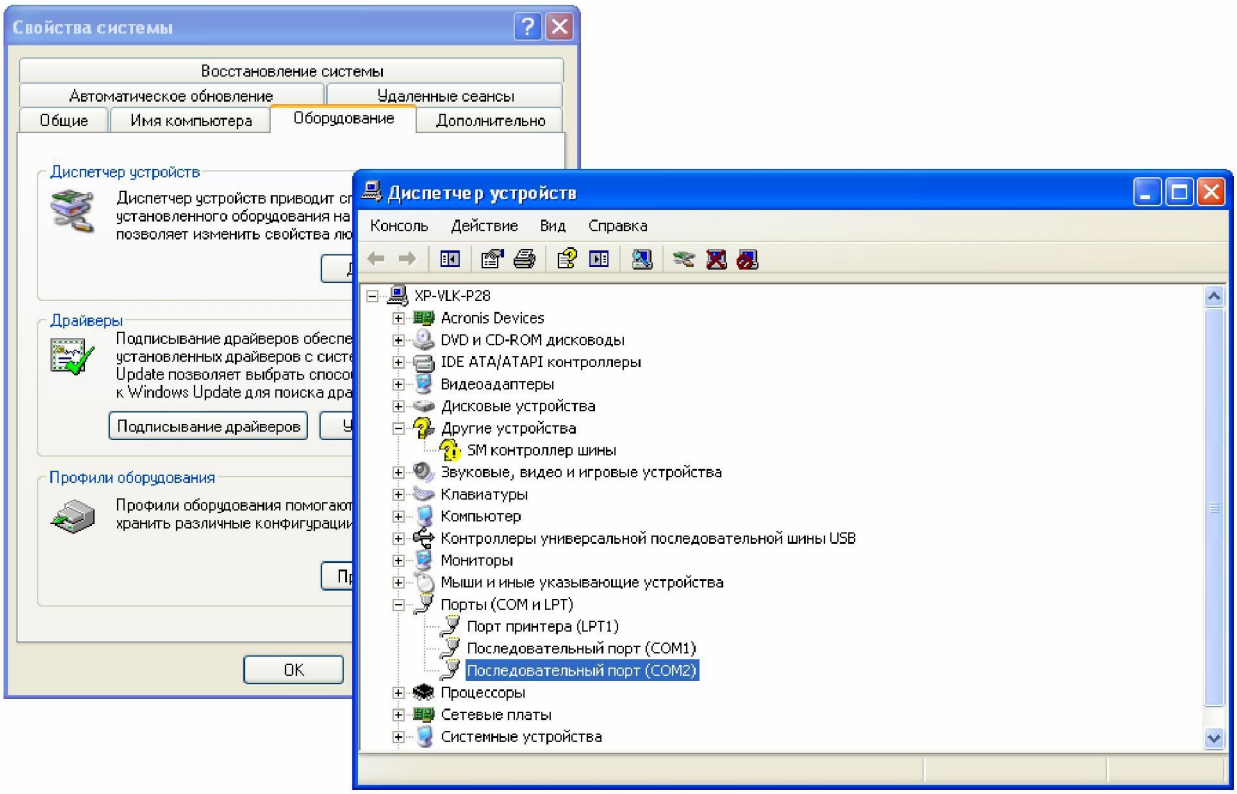


Рисунок 1 – определение номера виртуального COM-порта

Для настройки IDE Arduino (тип подключенной платы и номер зарегистрированного операционной системой COM-порта) необходимо воспользоваться соответствующими вкладками программы, что иллюстрируется следующим рисунком.

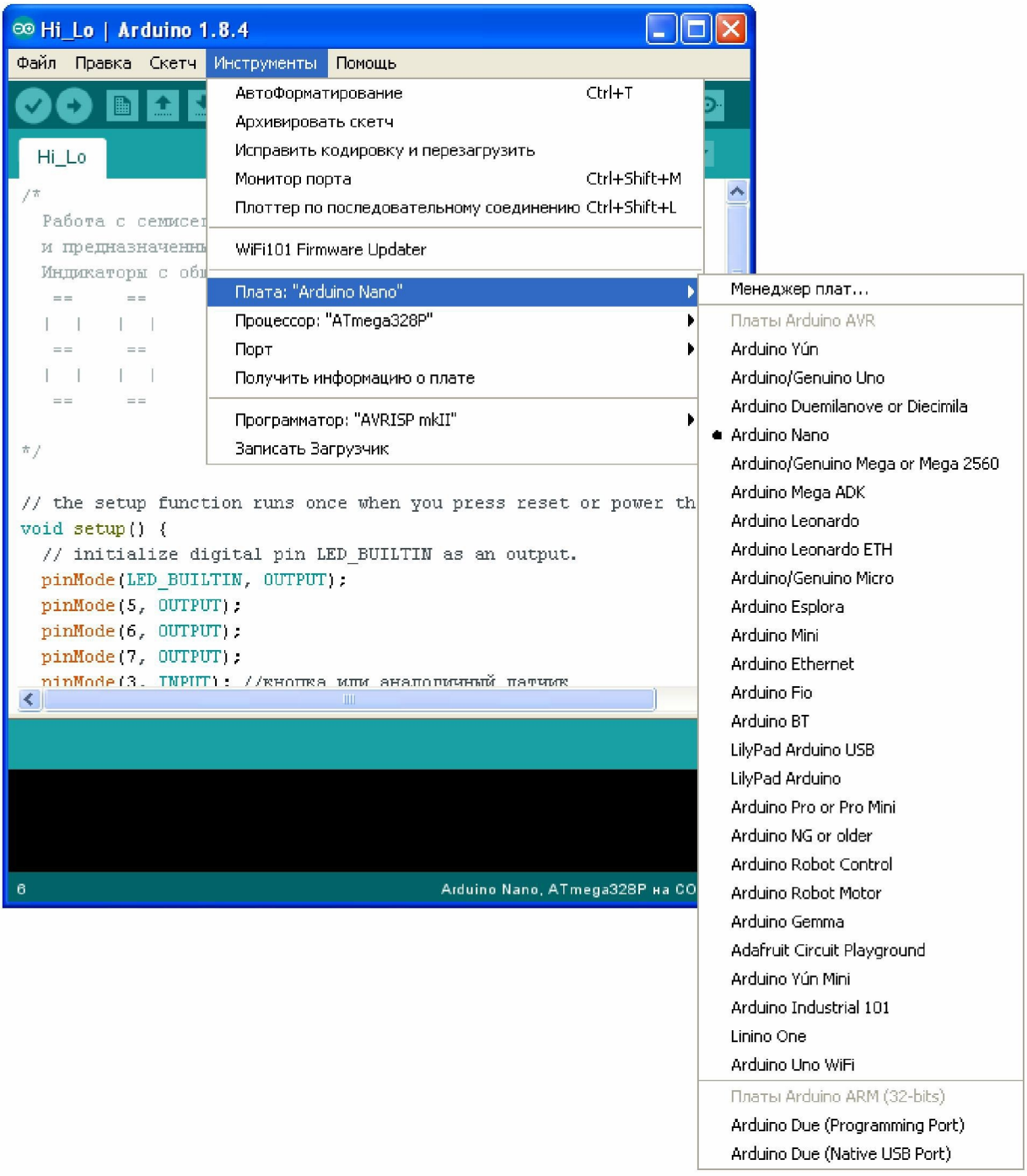


Рисунок 2 – настройка типа платы Arduino

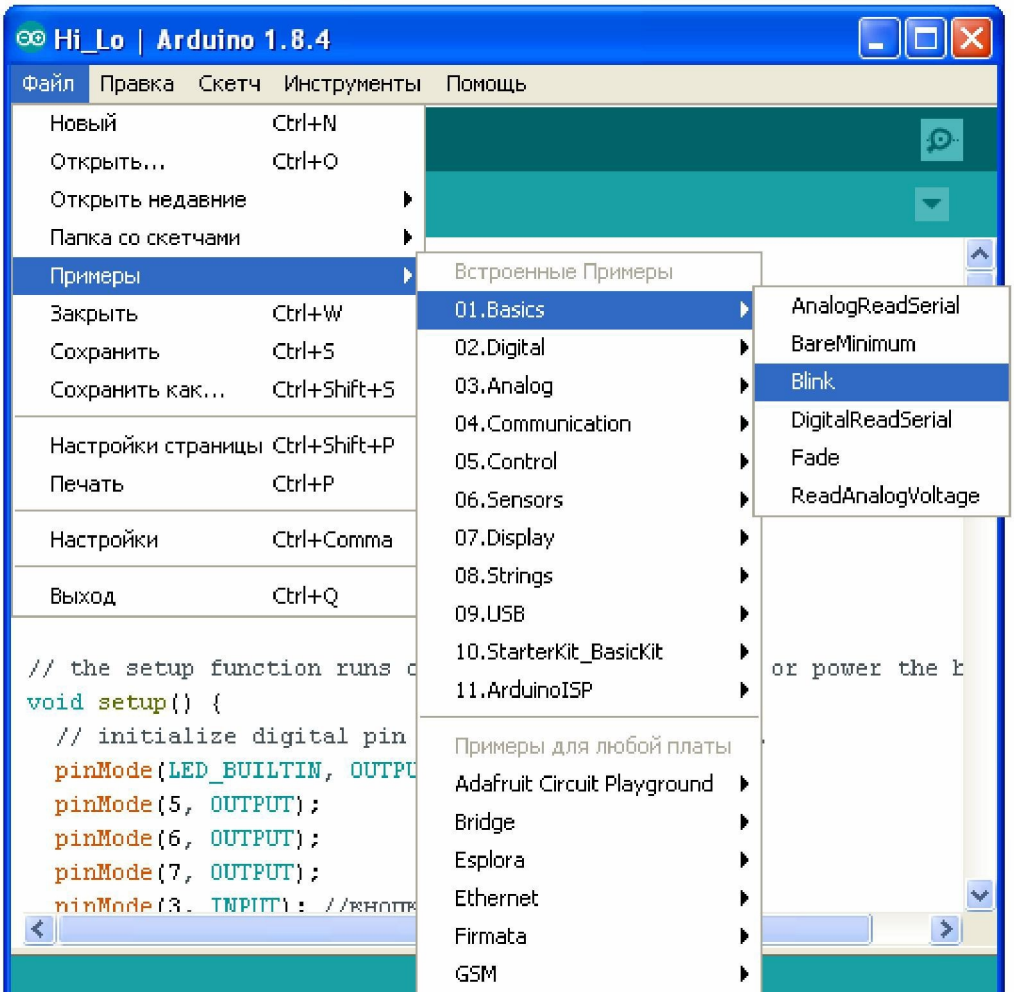


Рисунок 3 – выбор демонстрационной программы

Контрольные вопросы

1. Опишите стандартный модельный ряд плат Arduino с применением микро-ЭВМ серии ATMega.
2. Назовите требования к питанию для плат Arduino UNO и Arduino nano.
3. Каковы уровни логических сигналов на разъемах плат Arduino UNO и Arduino nano?
4. Опишите поведение демонстрационной программы **blink**.
5. Каковы особенности подключения плат, имеющих аппаратный интерфейс RS-232?
6. Какие альтернативные системы программирования разработаны для платформы Arduino?
7. Какие возможности предоставляет среда программирования IDE Arduino?

Содержание отчета

Отчет должен содержать:
- титульный лист с указанием темы работы;
- цель работы;
- описание лабораторного макета платы Arduino;
- протокол выполненных работ, содержащий изображения экрана ПК для основных этапов работ;
- выводы для важнейших этапов работ;
- ответы на контрольные вопросы (если имеются).

Лабораторная работа 2 Изучение среды разработки и отладки

Цель работы

Изучение базовых функций языка Arduino, анализ характеристик подключаемых в цикле лабораторных работ внешних устройств и выбор библиотек для управления этими устройствами.

Подготовка к работе

Изучение функций, необходимых для проведения лабораторных работ (Delay(), средств обмена по последовательному каналу Serial.begin(), Serial.print() и др.).

Вопросы для самоконтроля

- опишите назначение последовательного канала (UART, RS-232);
- опишите программные средства, используемые в программе Arduino для обмена по последовательному каналу;
- опишите средства, используемые в IDE Arduino на персональном компьютере для обмена по последовательному каналу;
- опишите уровни сигналов на цифровых и аналоговых линиях Arduino.

Программа работ

- изучить средства взаимодействия между платой Arduino и программной средой IDE Arduino на персональном компьютере;
- выполнить демонстрационный пример обмена;
- внести изменения в программу обмена в соответствии с рекомендациями преподавателя.

Методические указания

Макетная плата, содержащая микро-ЭВМ, описана в Приложении А. Плата позволяет устанавливать в гнезда изучаемые устройства и осуществлять их подключение к контактам разъема платы Arduino с помощью проводников со штырьками (гнездами).

Для отладки программ удобно обеспечить вывод сообщений на экран системы разработки. Однако плата Arduino не содержит каких-либо средств, пригодных для этой цели. Для решения этой проблемы предложен следующий способ.

Сообщения, формируемые программой, исполняемой на плате Arduino, передаются в особом формате последовательного вида по линии Tx интерфейса UART. Эти сообщения принимаются на персональном компьютере (ПК) и выводятся в текстовом виде в окне *терминала*. Хотя организация такого обмена не является простой, наличие готовых библиотечных функций для программы на плате Arduino и специальные программные средства в среде IDE Arduino на ПК существенно упрощают задачу.

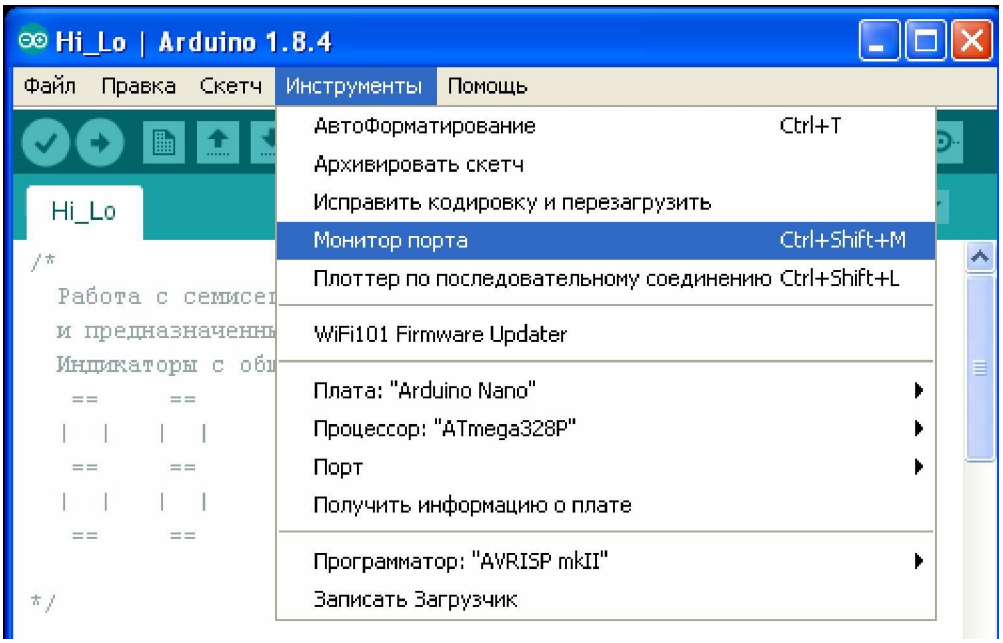


Рисунок 4 – вызов монитора порта из меню

Для работы в режиме обмена по последовательному каналу используется ***монитор порта*** (***serial monitor***), вызов которого осуществляется в соответствии с рисунком выше.

Существует альтернативный способ вызова монитора порта - активацией кнопки справа на панели IDE Arduino:

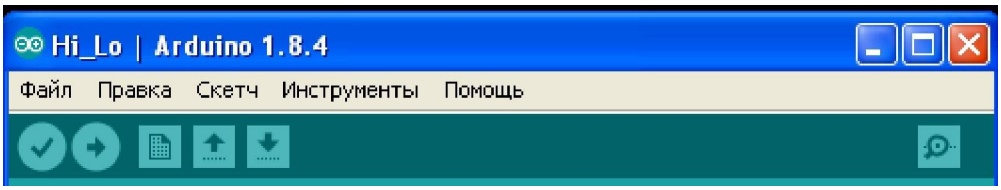


Рисунок 5 - вызов монитора порта кнопкой справа на панели

Выводимые через последовательный канал данные будут отображены в окне терминала:

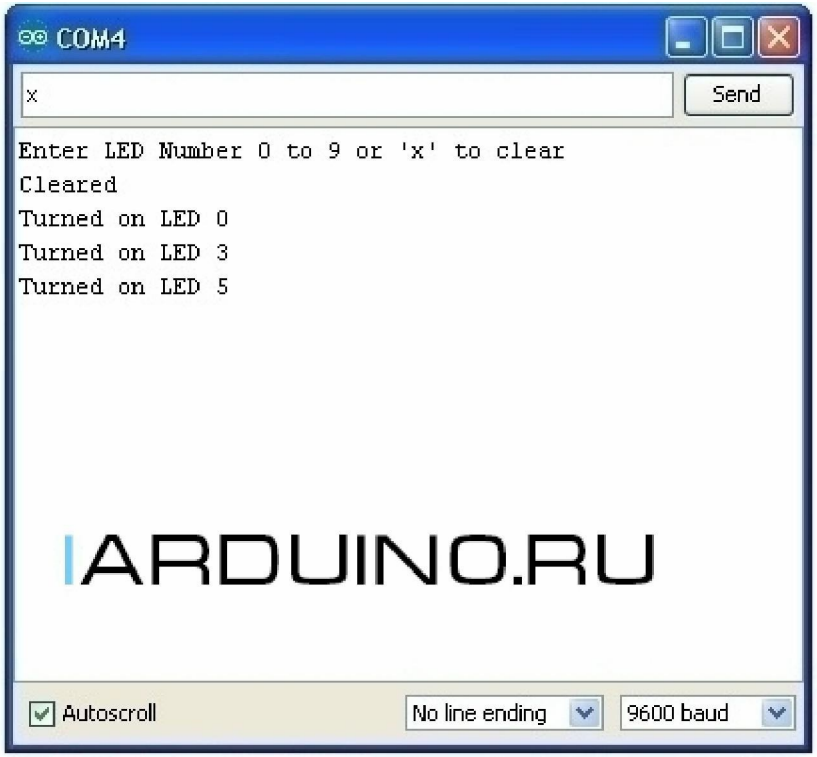


Рисунок 6 – пример вывода сообщений в окно монитора порта

Примеру на рисунке выше соответствует следующий программный код.

```
void setup()

{

  Serial.begin(9600);
// Ожидание готовности последовательного канала
  while (! Serial);
// Вывод первого сообщения
  Serial.println(\"Enter LED Number 0 to 7 or \'x\' to
clear\");
}

void loop()
{
if (Serial.available()) // если приняты данные
  {
    char ch = Serial.read(); // читаем принятые данные
    if (ch >= \'0\' && ch <= \'7\')
    {
      int led = ch - \'0\';
      //выводим принятые данные на экран
      Serial.print(\"Turned on LED \");
      Serial.println(led);
    }
    if (ch == \'x\')
    {
      Serial.println(\"Cleared\");
    }
  }
}
```

Порядок выполнения лабораторной работы

1. Изучить документацию по подключению платы Arduino и команды языка.
2. Составить схему подключения в соответствии с полученным заданием.
3. Выполнить монтаж в соответствии со схемой устройства на макетной плате
4. Предоставить смонтированное устройство для проверки преподавателю.

5. Выполнить набор, проверку и исполнение демонстрационной программы.
6. Проверить правильности функционирования программы
7. Внести изменения в программу по заданию преподавателя и добиться ее правильного функционирования.

Содержание отчета

Отчет должен содержать:

- титульный лист с указанием темы работы;
- цель работы;
- описание базовых функций языка для ввода-вывода (pinMode(), digitalRead(), digitalWrite(), analogRead(), analogWrite(), Delay() и т.д.);
- описание монтажной схемы подключения устройства;
- протокол выполненных работ, содержащий изображения экрана для основных этапов работ;
- выводы для важнейших этапов работ;
- ответы на контрольные вопросы (вопросы для самоконтроля).

Лабораторная работа 3 Проектирование программ управления объектами с цифровым интерфейсом

Программирование микро-ЭВМ обычно предполагает подключение к ним дополнительных устройств (датчиков, исполнительных механизмов, индикаторов,..). Часть программного кода разрабатываемых программ будет осуществлять взаимодействие с этими устройствами. Для правильного управления внешними устройствами необходимо знать особенности *интерфейса* этих устройств. Примерами интерфейсов являются [2]:

- цифровой интерфейс;
- аналоговый интерфейс;
- интерфейс SPI;
- интерфейс I2C;
- интерфейс 1-wire.

Наиболее простым для реализации и работы является цифровой интерфейс.

Цель работы

Изучение свойств цифровых линий платформы Arduino и приобретение навыков разработки программ для управления цифровыми линиями.

Подготовка к работе

Согласование с преподавателем устройств для исследования, их изучение .

Изучение функций, необходимых для проведения лабораторной работы (pinMode(), digitalRead(), digitalWrite(), Delay() и др.).

Вопросы для самоконтроля

- опишите назначение подключаемого устройства;

- опишите уровни сигналов на интерфейсе и сравните их с уровнями сигналов платы Arduino;
- опишите сферы практического применения устройства;
- какие программные функции языка IDE Arduino будет использовать программа?

Программа работ

Для набора предложенных преподавателем модулей:
- изучить документацию, описывающую модуль;
- изучить демонстрационный пример;
- составить схему подключения модуля к плате;
- осуществить подключение устройства к плате;
- выполнить демонстрационный пример;
- внести изменения в программу в соответствии с рекомендациями преподавателя.

Методические указания

Макетная плата, содержащая микро-ЭВМ, описана в Приложении А. Плата позволяет устанавливать в гнезда изучаемые устройства и осуществлять их подключение к контактам разъема платы Arduino с помощью проводников со штырьками (гнездами).

Цифровые линии на плате Arduino имеют порядковые номера (см. Приложение Б). Каждая из этих линий может быть настроена на ввод или вывод. Важно знать, что уровни выходных и входных сигналов на цифровых линиях должны быть согласованы со спецификацией микро-ЭВМ и не превышать напряжения питания микро-ЭВМ. Обычно напряжение питания равно 5 V, а уровни логических сигналов таковы:
- логический 0: 0…0,5V;
- логическая 1: 2,4V…5V.

Для настройки цифровой линии в режим ***ввода*** или ***вывода*** используют функцию pinMode().

Линия в режиме ***ввода*** принимает сигналы от датчиков, которые в данный момент соответствуют одному из логических уровней. Определить конкретное значение уровня можно с использованием функции digitalRead().

На рисунке ниже приводится пример подключения кнопки к одному из цифровых входов.

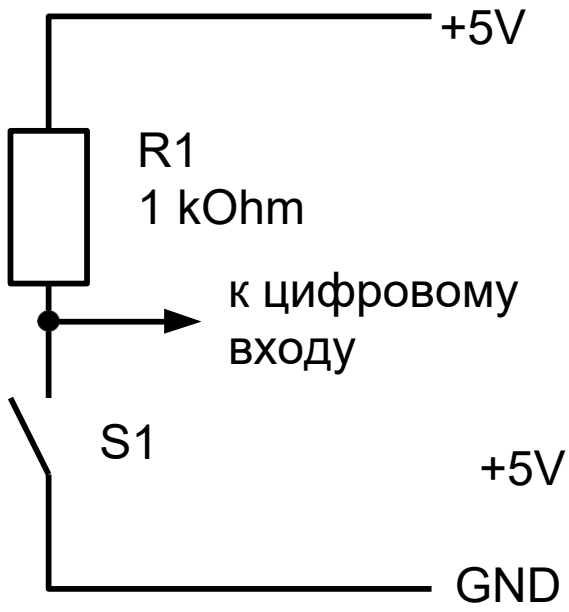


Рисунок 7 – подключение кнопки к цифровому входу

При отпущенной (не замкнутой) кнопке на цифровом входе будет присутствовать уровень логической 1. При нажатии (замыкании) кнопки этот уровень станет равен логическому 0.

Линия в режиме ***вывода*** формирует сигналы уровня логического 0 или логической 1, что достигается применением функции digitalWrite().

На рисунке ниже приводится пример подключения светодиодного индикатора к одному из цифровых выходов.

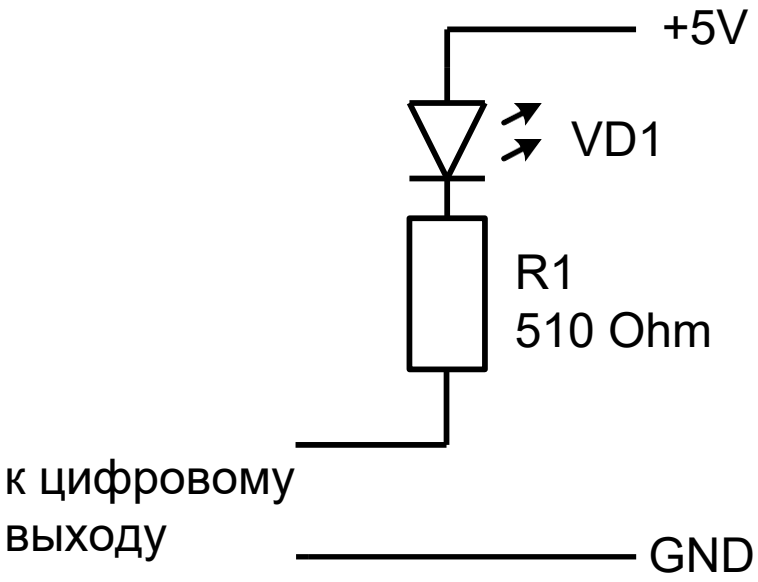


Рисунок 8 – подключение светодиода к цифровому выходу

Если на цифровом выходе сформирован уровень логического 0, замыкается цепь *источник питания–светодиод-резистор* и светодиодный индикатор светится.

Если на цифровом выходе сформирован уровень логической 1, уровень выходного сигнала близок к напряжению питания, а при этом цепь *источник питания–светодиод-резистор* будет разорвана (закрыт светодиод) и индикатор будет погашен.

Замечание:
использование ограничительного резистора R1 совместно со светодиодом обязательно!

Порядок выполнения лабораторной работы

Порядок работ зависит от того, какое устройство будет подключаться. Ниже перечислены типичные шаги выполнения работы.

1. Изучение документации.
2. Составление схемы подключения.
3. Монтаж в соответствии со схемой устройства на макетной плате.
4. Набор, проверка и исполнение демонстрационной программы.

5. Проверка правильности функционирования программы.
6. Внесение изменений в программу по заданию преподавателя и доведение ее до правильного функционирования.

Замечание: перед подключением необходимо предоставить смонтированное устройство для проверки преподавателю.

Содержание отчета

Отчет должен содержать:
- титульный лист с указанием темы работы;
- цель работы;
- описание использованных функций языка для ввода-вывода;
- описание монтажной схемы подключения устройства;
- протокол выполненных работ, содержащий изображения экрана для основных этапов работ;
- выводы для важнейших этапов работ;
- ответы на контрольные вопросы (вопросы для самоконтроля).

Лабораторная работа 4 Проектирование программ управления объектами с аналоговым интерфейсом

Цель работы

Изучение свойств аналоговых линий платформы Arduino и приобретение навыков разработки программ для управления аналоговыми линиями.

Подготовка к работе

Согласование с преподавателем устройств для исследования, их изучение .

Изучение функций, необходимых для проведения лабораторной работы (analogRead(), analogWrite(), Delay() и др.).

Вопросы для самоконтроля

- опишите назначение подключаемого устройства;
- опишите уровни сигналов на интерфейсе и сравните их с уровнями сигналов платы Arduino;
- опишите сферы практического применения устройства;
- какие программные функции языка IDE Arduino будет использовать программа?

Программа работ

Для набора предложенных преподавателем модулей:
- изучить документацию, описывающую модуль;
- изучить демонстрационный пример;
- составить схему подключения модуля к плате;
- осуществить подключение устройства к плате;
- выполнить демонстрационный пример;
- внести изменения в программу в соответствии с рекомендациями преподавателя.

Методические указания

Аналоговые линии на плате Arduino имеют порядковые номера, начинающиеся с буквы А (см. Приложение Б). Аналоговые линии Arduino обычно имеют предопределенную настройку на ввод и подключены к входам аналого-цифрового преобразователя (АЦП). Важно знать, что уровни выходных и входных сигналов на цифровых линиях должны быть согласованы со спецификацией микро-ЭВМ и не превышать напряжения питания микро-ЭВМ.

Обычно напряжение питания равно 5 V, а уровни аналоговых сигналов могут принимать любое значение в диапазоне 0…5V.

АЦП распознает эти сигналы, но воспринять бесконечное число возможных значений он не способен. Упрощенно говоря, АЦП способен представить измеренное значение с точностью, определяемой его *разрядностью*. Например, 10-разрядный АЦП выдает 10-битовый код, разбивая весь интервал измерений на 1024 части. Если номинальное напряжение на входе АЦП соответствует величине 5V, то минимальное воспринимаемое значение сигнала примерно равно 5V/1000=5mV.

Функция analogRead (pin) считывает значение из заданного аналогового входа (pin) с 10-битовым разрешением. Эта функция работает только на аналоговых портах (0-5). Результирующее целое значение находится в диапазоне от 0 до 1023.

Для типичной платы Arduino цифровые выходы работают в режиме широтно-импульсной модуляции (ШИМ, PWM). Плата Arduino с ATmega8 поддерживает только выводы 9, 10 и 11. Для плат Arduino с ATmega168 (328) функция ШИМ работает на выводах 3, 5, 6, 9, 10 и 11.

Функция analogWtite(pin, value) записывает псевдо-аналоговое значение, используя схему с широтно-импульсной модуляцией (PWM), на выходной вывод, помеченный как PWM. Значение может быть задано как переменная или константа в диапазоне 0-255.

На рисунке ниже приводится пример формирования аналогового сигнала на одном из аналоговых входов.

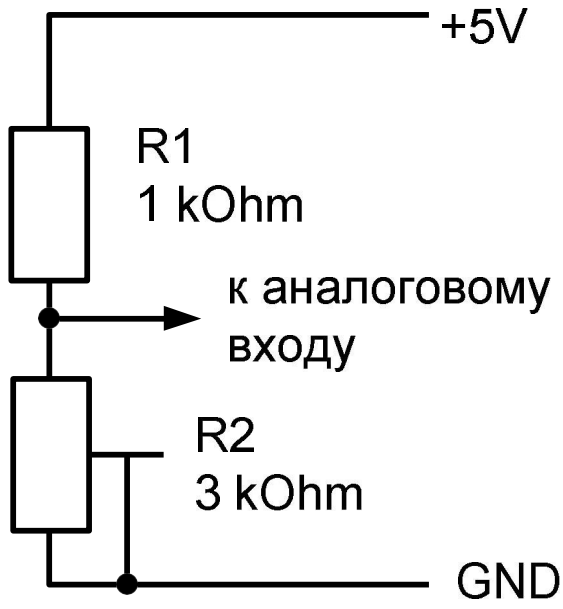


Рисунок 9 – подключение кнопки к цифровому входу

Переменный (подстроечный) резистор R2 в схеме может изменять свое сопротивление при регулировке. Совместно с резистором R1 он образует делитель напряжения, напряжение на выходе которого может принимать произвольное значение в некотором диапазоне. Минимальное напряжение при верхнем положении движка резистора R2 равно 0, максимальное при нижнем положении движка равно ¾ от напряжения питания (для указанных на схеме номиналах резисторов).

Линия в режиме ШИМ формирует сигналы уровня, который имеет 256 градаций в некотором диапазоне. Для этого используется функция analogWrite().

На рисунке ниже приводится пример подключения светодиодного индикатора к одному из аналоговых выходов.

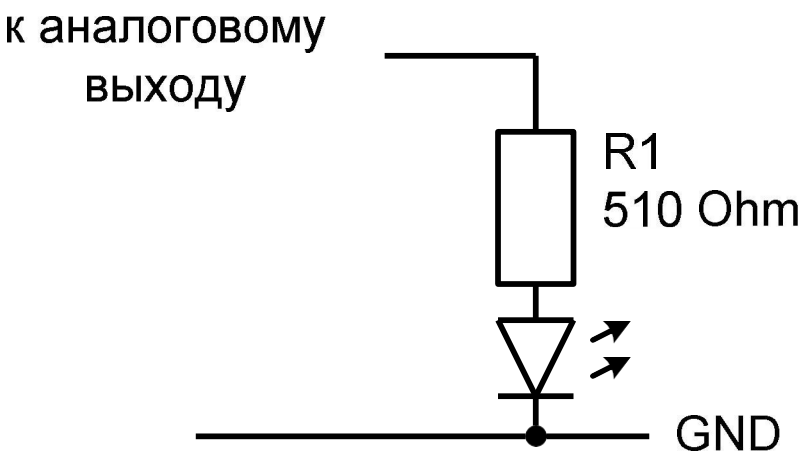


Рисунок 10 – подключение светодиода к аналоговому выходу

Средняя величина тока, протекающего через светодиод (а следовательно, и яркость свечения индикатора) определяется уровнем выходного напряжения аналогового выхода.

Замечание:
использование ограничительного резистора R1 совместно со светодиодом обязательно!

Порядок выполнения лабораторной работы

Порядок работ зависит от того, какое устройство будет подключаться. Ниже перечислены типичные шаги выполнения работы.

1. Изучение документации.
2. Составление схемы подключения.
3. Монтаж в соответствии со схемой устройства на макетной плате.
4. Набор, проверка и исполнение демонстрационной программы.
5. Проверка правильности функционирования программы.
6. Внесение изменений в программу по заданию преподавателя и доведение ее до правильного функционирования.

Замечание: перед подключением необходимо предоставить смонтированное устройство для проверки преподавателю.

Содержание отчета

Отчет должен содержать:
- титульный лист с указанием темы работы;
- цель работы;
- описание использованных функций языка для ввода-вывода;
- описание монтажной схемы подключения устройства;
- протокол выполненных работ, содержащий изображения экрана для основных этапов работ;
- выводы для важнейших этапов работ;
- ответы на контрольные вопросы (вопросы для самоконтроля).

Лабораторная работа 5 Проектирование программ управления объектами с интерфейсом 1-wire

Программирование микро-ЭВМ обычно предполагает подключение к ним дополнительных устройств (датчиков, исполнительных механизмов, индикаторов,..). Часть программного кода разрабатываемых программ будет осуществлять взаимодействие с этими устройствами. Для правильного управления внешними устройствами необходимо знать особенности *интерфейса* этих устройств. Примерами интерфейсов являются:

- цифровой интерфейс;
- аналоговый интерфейс;
- интерфейс SPI [2];
- интерфейс I2C [2];
- интерфейс 1-wire [3].

Многие интерфейсы сложны в использовании. Для облегчения процесса подключения и управления разрабатываются специальные программные библиотеки, использование которых позволяет абстрагироваться от аппаратных и программных особенностей интерфейса и взаимодействовать с устройствами через простые и понятные программные функции. Среда IDE Arduino поддерживает множество таких библиотек.

Цель работы

Изучение свойств интерфейса 1-wire и приобретение навыков разработки программ для управления периферийным оборудованием, использующим данный интерфейс в проектах с применением платформы Arduino.

Подготовка к работе

Согласование с преподавателем устройств для исследования.
Изучение документации на заданное устройство.

Изучение функций, необходимых для проведения лабораторной работы

Вопросы для самоконтроля

- опишите назначение подключаемого устройства;
- опишите интерфейс, используемый устройством;
- опишите сферы практического применения устройства;
- какие программные функции языка IDE Arduino будет использовать программа?

Программа работ

Для набора предложенных преподавателем модулей:
- изучить документацию, описывающую модуль;
- изучить демонстрационный пример;
- составить схему подключения модуля к плате;
- осуществить подключение устройства к плате;
предоставить собранную систему для проверки преподавателю;
- если ошибок в соединениях нет, исполнить демонстрационный пример.

Методические указания

Макетная плата, содержащая микро-ЭВМ, описана в Приложении А. Плата позволяет устанавливать в гнезда изучаемые устройства и осуществлять их подключение к контактам разъема платы Arduino с помощью проводников со штырьками (гнездами).

Интерфейс 1-wire позволяет передавать и принимать информацию по одному сигнальному проводу, что обусловливает сферу его применения. Встречается интерфейс чаще всего в изделиях фирмы Dallas Semiconductor, которая и представила его в конце 90-х годов прошлого века. Системы 1-Wire привлекательны благодаря легкости монтажа, низкой стоимости устройств,

возможности распознавать устройство при подключении к функционирующей сети, большому числу устройств в сети и т.д.

Типичная система 1-Wire состоит из управляющего контроллера (мастера или ведущего) и одного или нескольких устройств (ведомых), присоединенных к общей шине (см. рисунок ниже).

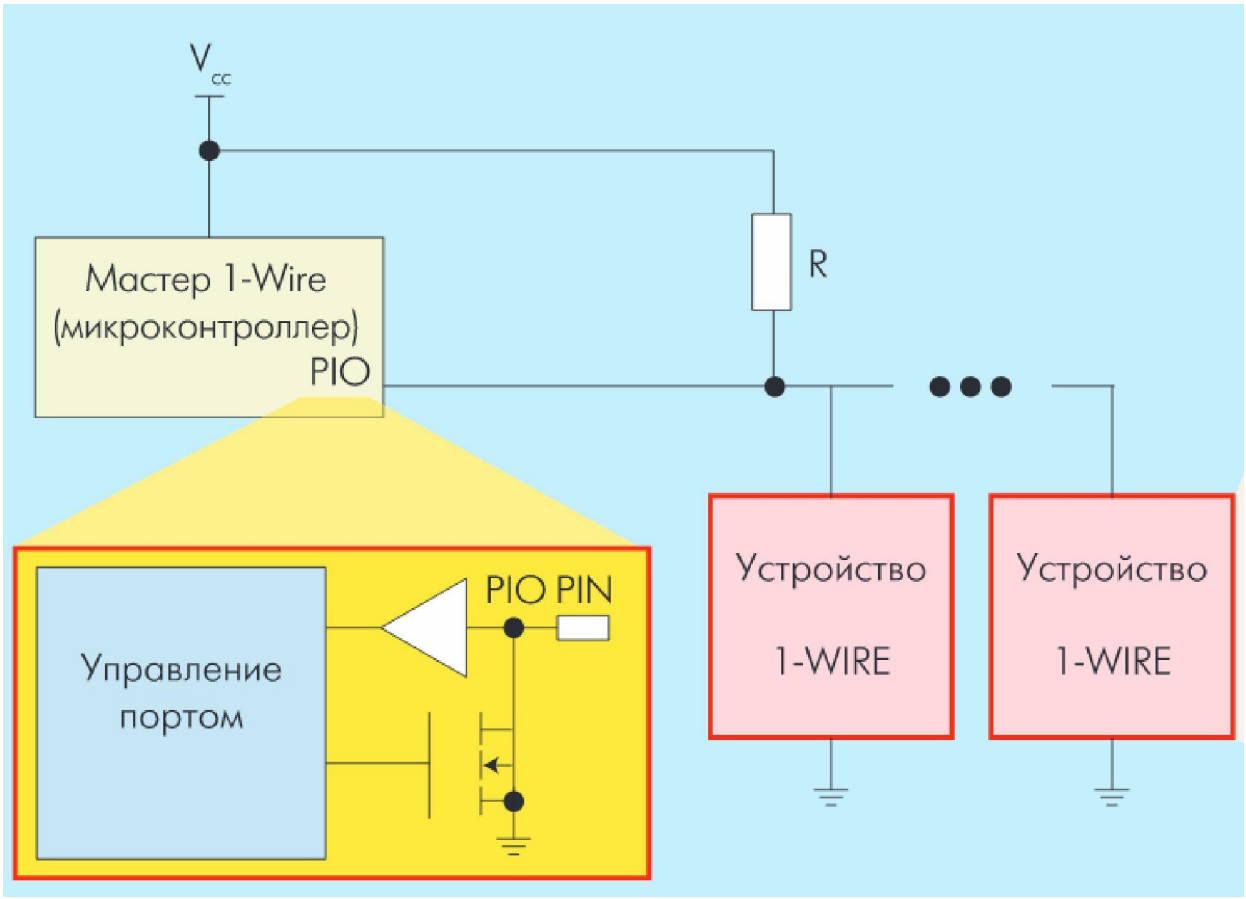


Рисунок 11 - подключение устройств по интерфейсу 1-wire

Весь обмен на шине 1-Wire происходит посредством специальных команд. Их число для каждого типа устройств различно. Но есть и минимальный набор стандартных команд, которые поддерживают все 1-Wire-устройства – так называемые ROM-команды.

У каждого устройства 1-Wire есть 64-разрядный идентификатор (ID). Он состоит из 8-разрядного кода семейства, который идентифицирует тип устройства и поддерживаемые им функции, 48-разрядного серийного номера и 8-битного поля кода циклического избыточного контроля (CRC-8). ID вводится при

изготовлении устройства и хранится в ПЗУ. Фирма Maxim гарантирует, что один раз использованный адрес никогда не повторится в другом устройстве. Для обмена с устройством необходимо этот адрес знать. С этой целью разработан специальный алгоритм, позволяющий осуществить определение номера каждого устройства в сети 1-wire.

Для Arduino существуют библиотеки, существенно упрощающие разработку программ при использовании интерфейса 1-wire.

Пример устройства с интерфейсом 1-wire (DS18B20 – цифровой термометр с программируемым разрешением, от 9 до 12–bit) приведен в Приложении В.

Порядок выполнения лабораторной работы

Порядок работ зависит от того, какое устройство будет подключаться. Ниже перечислены типичные шаги выполнения работы.

1. Изучение документации.
2. Составление схемы подключения.
3. Монтаж в соответствии со схемой устройства на макетной плате.
4. Набор, проверка и исполнение демонстрационной программы.
5. Проверка правильности функционирования программы.
6. Внесение изменений в программу по заданию преподавателя и доведение ее до правильного функционирования.

Замечание: перед подключением необходимо предоставить смонтированное устройство для проверки преподавателю.

Содержание отчета

Отчет должен содержать:

- титульный лист с указанием темы работы;
- цель работы;
- описание использованных функций языка для ввода-вывода;

- описание монтажной схемы подключения устройства;
- протокол выполненных работ, содержащий изображения экрана для основных этапов работ;
- выводы для важнейших этапов работ;
- ответы на контрольные вопросы (вопросы для самоконтроля).

Лабораторная работа 6 Проектирование программ управления объектами с интерфейсом I2C

Изучение свойств интерфейса 1-wire [3] и приобретение навыков разработки программ для управления периферийным оборудованием, использующим данный интерфейс в проектах с применением платформы Arduino.

Подготовка к работе

Согласование с преподавателем устройств для исследования.

Изучение документации на заданное устройство.

Изучение функций, необходимых для проведения лабораторной работы

Вопросы для самоконтроля

- опишите назначение подключаемого устройства;
- опишите интерфейс, используемый устройством;
- опишите сферы практического применения устройства;
- какие программные функции языка IDE Arduino будет использовать программа?

Программа работ

Для набора предложенных преподавателем модулей:

- изучить документацию, описывающую модуль;
- изучить демонстрационный пример;
- составить схему подключения модуля к плате;
- осуществить подключение устройства к плате;

предоставить собранную систему для проверки преподавателю;

- если ошибок в соединениях нет, исполнить демонстрационный пример.

Методические указания

Макетная плата, содержащая микро-ЭВМ, описана в Приложении А. Плата позволяет устанавливать в гнезда изучаемые устройства и осуществлять их подключение к контактам разъема платы Arduino с помощью проводников со штырьками (гнездами).

Интерфейс I2C (I I C) был разработан компанией Philips и зарегистрирован под запатентованным названием “I2C”. Существуют аналоги этого интерфейса, которые используют иные названия (TWI, 2 line interface). Все они работают по единому принципу.

Интерфейс I2C использует для обмена информацией 2 линии и позволяет организовать сеть, в которую включается большое (128) количество различных устройств - от датчиков температуры до микроконтроллеров.

Линии интерфейса:

- SDA - отвечает за передачу информации(начало передачи, адрес, данные);
- SCL - тактирование шины.

Интерфейс I2C предусматривает устройства двух типов: Master (главное, ведущее) и Slave (ведомое).

Структурная схема подключения устройств с интерфейсом I2C приведена на рисунке ниже. Стандарт предусматривает наличие подтягивающих резисторов R1, R2. Эти резисторы при отсутствии активности на линиях обеспечивают уровни логической 1.

Передача/прием сигналов осуществляется подтягиванием уровней на линиях к общему проводу (уровни логического 0). Взаимодействие устройств описывается стандартом.

Для упрощения программирования устройств с интерфейсом I2C для платформы Arduino разработаны специальные библиотеки.

Пример устройства с интерфейсом 1-wire (DS18B20 – цифровой термометр с программируемым разрешением, от 9 до 12–bit) приведен в Приложении Г.

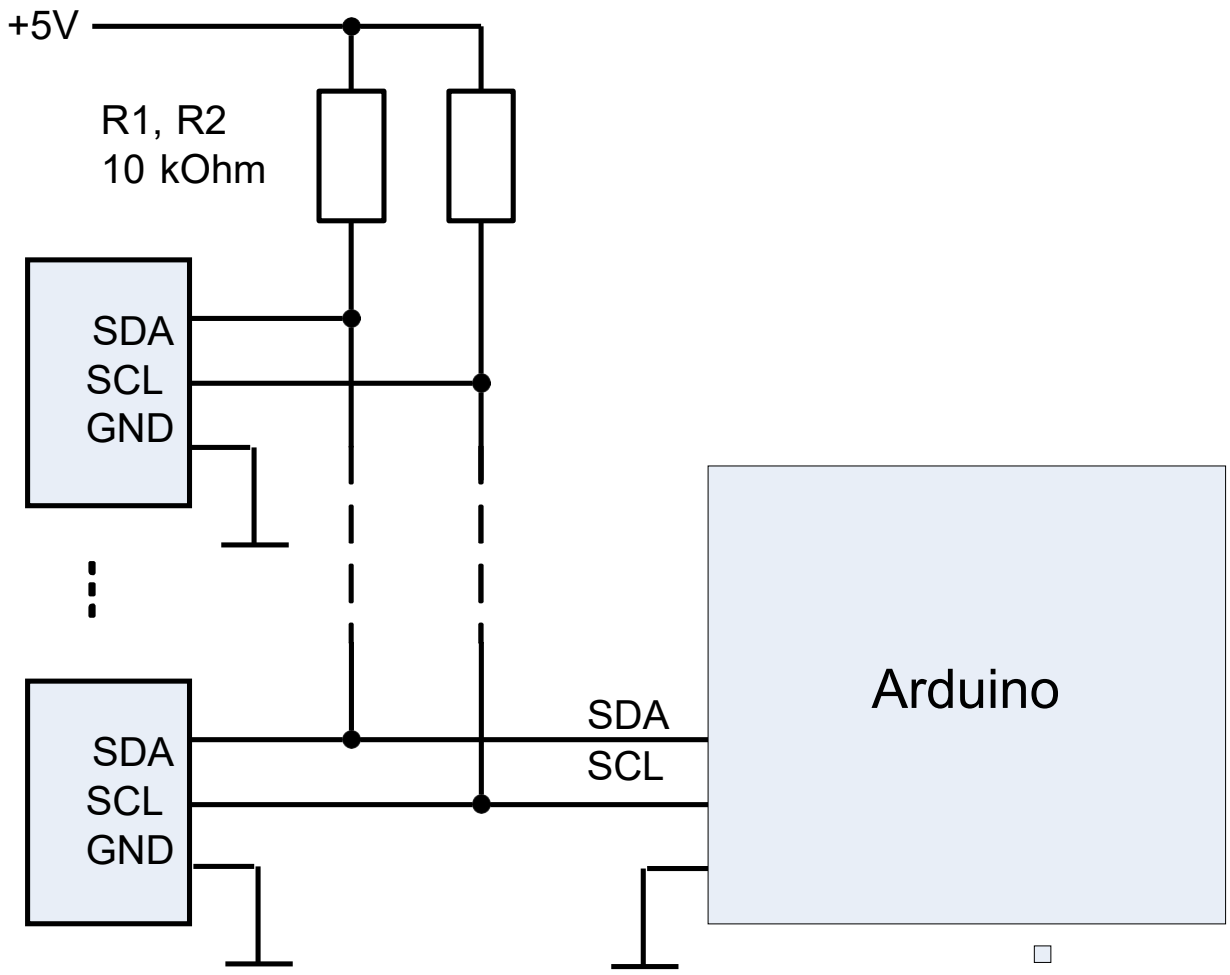


Рисунок 12 – подключение устройств с интерфейсом I2C

Порядок выполнения лабораторной работы

Порядок работ зависит от того, какое устройство будет подключаться. Ниже перечислены типичные шаги выполнения работы.

1. Изучение документации.
2. Составление схемы подключения.
3. Монтаж в соответствии со схемой устройства на макетной плате.
4. Набор, проверка и исполнение демонстрационной программы.
5. Проверка правильности функционирования программы.
6. Внесение изменений в программу по заданию преподавателя и доведение ее до правильного функционирования.

Замечание: перед подключением необходимо предоставить смонтированное устройство для проверки преподавателю.

Содержание отчета

Отчет должен содержать:
- титульный лист с указанием темы работы;
- цель работы;
- описание использованных функций языка для ввода-вывода;
- описание монтажной схемы подключения устройства;
- протокол выполненных работ, содержащий изображения экрана для основных этапов работ;
- выводы для важнейших этапов работ;
- ответы на контрольные вопросы (вопросы для самоконтроля).

Литература

1. Монк, С. Программируем Arduino. Профессиональная работа со скетчами[Текст]: пер. с английского. - СПб.: Питер, 2017.

2. Лапин, А. А. Интерфейсы. Выбор и реализация [Текст] / А. А. Лапин. - Москва : Техносфера, 2005. - 168 с.

3. Интерфейс 1-Wire: устройство и применение http://www.electronics.ru/files/article_pdf/0/article_668_639.pdf

4. Arduino - блокнот программиста - Brian W. Evans (русский перевод) http://robocraft.ru/files/books/arduino_notebook_rus_v1-1.pdf

Приложение А - Макетирование устройств при подключении к Arduino

Макетная плата (breadboard) обеспечивает беспаечный способ монтажа плат Arduino и исследуемых устройств. Конструкция одного из вариантов платы приведена на рисунке ниже.

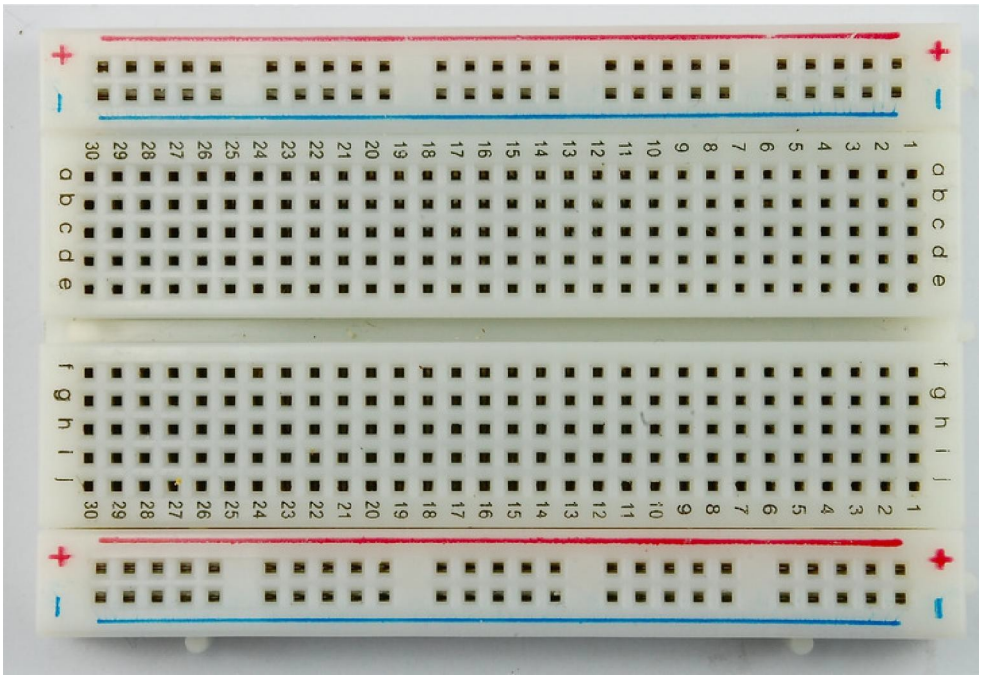


Рисунок А.1 - Макетная плата (breadboard)

Контакты, имеющие один номер в пределах верхней или нижней части платы, соединены друг с другом и изолированы от прочих элементов платы. Например, соединены контакты a30, b30, c30, d30,e30 в верхней части платы и контакты f30, g30, h30, i30, j30 в нижней части.

Цепи для подводки питания обозначены знаками «+» и «-», а также выделены цветом: синий (минус) и красный (плюс). Эти контакты соединены по горизонтали и изолированы друг от друга.

Пример установки модулей на монтажную плату показан на следующем рисунке и включает плату формата Arduino nano и дополнительный модуль питания. Заметим, что питание Arduino nano может осуществляться по линиям USB, а дополнительная плата необходима только при наличии модулей с большим потреблением или питанием 3,3V.

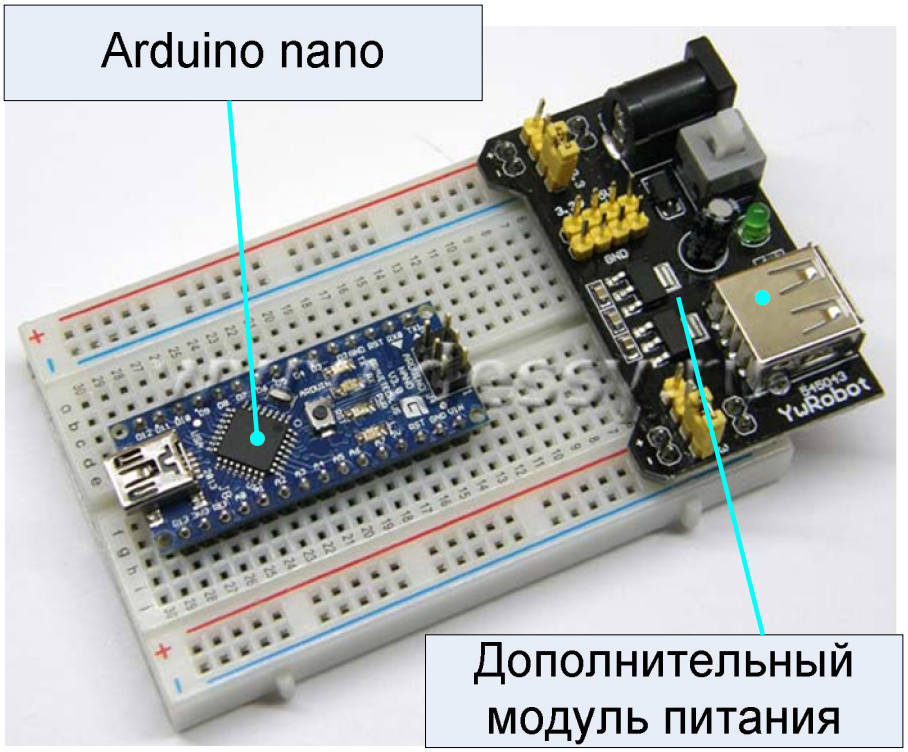


Рисунок А.2 – установка модулей на печатной плате

Дополнительные соединения осуществляются проводниками со штырьками (гнездами), что показано на рисунке ниже (https://ahrameev.ru/wp-content/uploads/2015/09/IMG_5015.jpg)

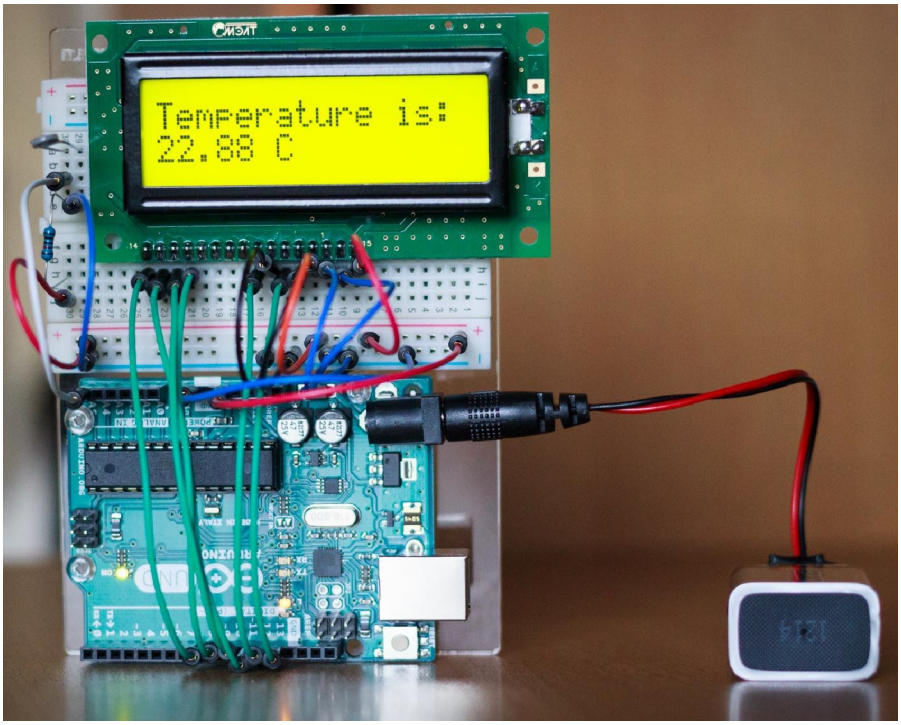
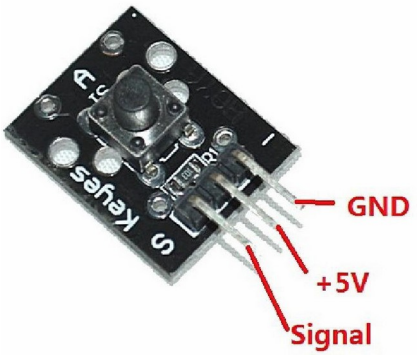
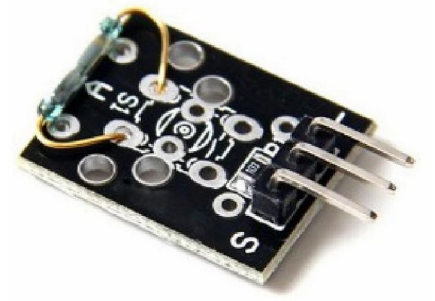


Рисунок А.3 – пример монтажа

Приложение Б – стандартные модули для учебных макетов

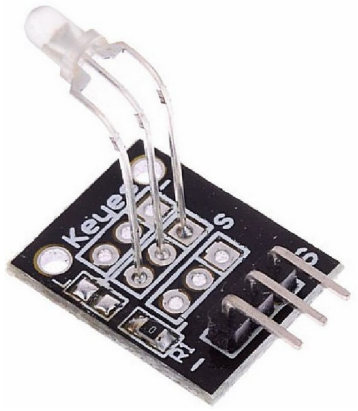
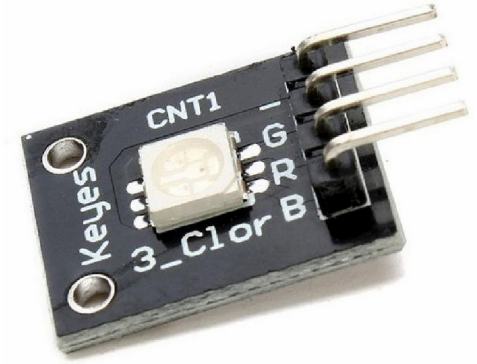
Для учебных целей выпускается большое количество модулей, содержащих типичные элементы, применяемые при проектировании простых систем: кнопки, индикаторы, датчики и.т.п. Их описание можно найти в сети Интернет, но нет гарантии, что они одинаковы для изделий разных производителей. По этой причине перед подключением необходимо изучение этих модулей и составление схемы их коммутации.

Таблица 1. Замыкатели-размыкатели

Внешний вид модуля	Описание
	Модуль **KY-004** позволяет смонтировать кнопку на передней панели небольшого прибора благодаря отверстиям в плате..
	Модуль датчика наклона **KY-017** (Mercury open optical module) осуществляет замыкание/размыкание контактов при наклоне платы. Механизм замыкания обеспечивается положением ртутного шарика в баллоне датчика.
	Модуль на основе геркона **KY-021 (**Mini magnetic reed modules) осуществляет замыкание контактов при воздействии магнитного поля.

Датчики, приведенные в таблице 1, подключаются по схеме на рисунке 7.

Таблица 2. Светодиодные индикаторы

Внешний вид модуля	Описание
	Двухцветный светодиодный индикатор (2color LED module 3MM) **KY-029** Светодиод светится красным или зеленым светом. При включении обоих источников света светодиод излучает свет оранжевого оттенка.
	Модуля **KY-009** содержит: 3-х цветный светодиод , с излучателями света красного, зеленого и синего цветов. Они могут светиться одновременно или поочередно.

Каждый светодиодный элемент устройств из таблицы 2 управляется индивидуально, как показано на рисунках 8 и 10. Подобные модули могут содержать светодиоды с общим анодом или общим катодом, что требует различных вариантов включения.

Приложение В – цифровой датчик температуры с интерфейсом 1-wire

DS18B20 – это цифровой термометр с программируемым разрешением, от 9 до 12–bit. DS18B20 обменивается данными по 1-Wire шине и при этом может быть как единственным устройством на линии, так и работать в группе. Все процессы на шине управляются центральным микропроцессором.

Диапазон измерений от –55°C до +125°C и точностью 0.5°C в диапазоне от –10°C до +85°C.

Каждая микросхема DS18B20 имеет уникальный 64-битный код, который позволяет идентифицировать каждый датчик на шине.

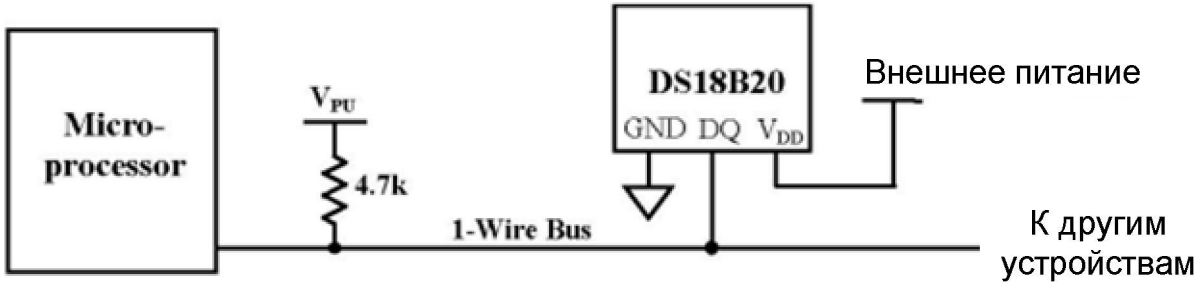


Рисунок В.1 - Схема подключения микросхемы DS18B20

На основе микросхемы DS18B20 изготавливается модуль KY-001, удобный для исследования на макетных платах с использованием Arduino.

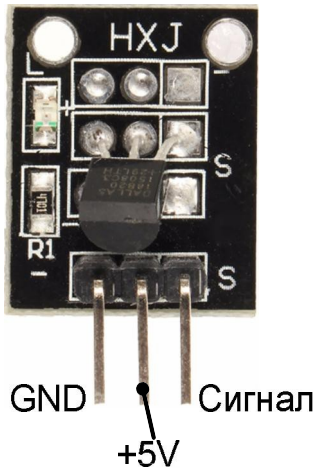


Рисунок В.2 – Модуль KY-001

Приложение Г – цифровые часы с интерфейсом I2C

В некоторых технических задачах требуется знать точное текущее время. Для этого удобно использовать специальные модули часов реального времени. Одной из самых распространённых систем для использования в модулях для Arduino используется микросхема DS1307, для которой существует большое количество библиотек под различные платформы.

Рисунок Г.1 - модуль часов реального времени на микросхеме DS1307

Стандартная схема включения микросхемы DS1307 приведена на рисунке ниже.

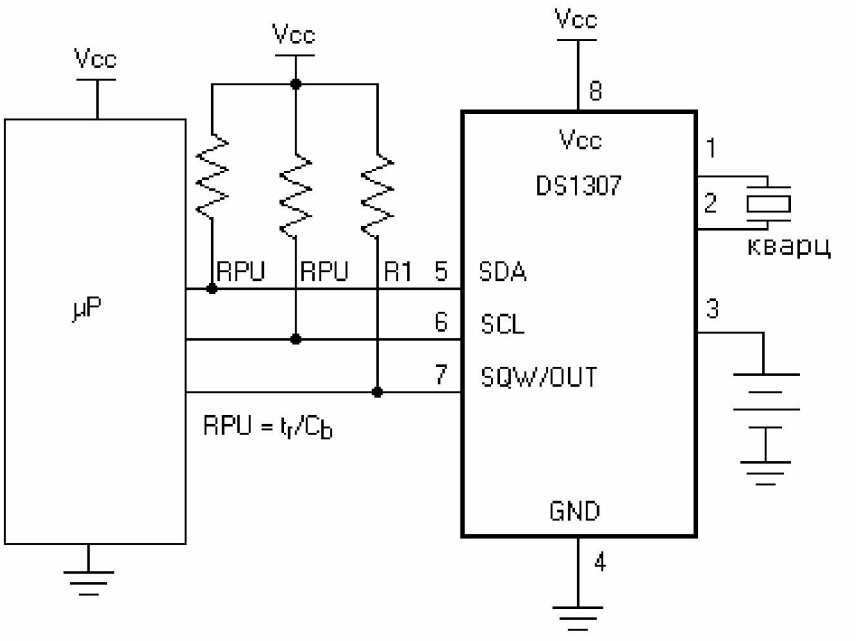


Рисунок Г.2 – схема подключения DS1307

Приложение Д – обзор конструкций языка IDE Arduino

Конструкция языка	Пример	Описание
Обязательные конструкции		

setup()	void setup()
	{
	pinMode(3, INPUT);
	}
		Конструкция используется для инициализации переменных, настройки режимов работы линий и т.д. Функция запускается только один раз при рестарте микроконтроллера.

loop()	void loop()
	{
	digitalWrite(3, HIGH);
	delay(1000);
	digitalWrite(3, LOW);
	delay(1000);
	}
		Конструкция обеспечивает бесконечный цикл включенных в нее действий.
		Функции setup() и loop() должны присутствовать в каждом скетче, даже если эти функции не используются.

Управляющие операторы		

if	…
	if (x > 100) digitalWrite(13, HIGH);
	if (x < 100) digitalWrite(13, LOW);
	…
		Оператор if используется в сочетании с операторами сравнения (==, !=, <, >) и проверяет, достигнута ли истинность условия. Пример: если значение переменной x больше 100, то включается светодиод на выходе 13, если меньше — выключается.

if..else	…
	if (x > 100) digitalWrite(13, HIGH);
	else digitalWrite(13, LOW);
	…
		Оператор else позволяет сделать проверку отличную от указанной в if, чтобы осуществлять несколько взаимоисключающих проверок. Если ни одна из проверок не получила результат ИСТИНА, то выполняется блок операторов в else.

switch…case	…
	switch (x)
	{
	case 1: digitalWrite(3, HIGH);
	case 2: digitalWrite(3, LOW);
	case 3: break;
	default: digitalWrite(4, HIGH);
	}
	…
		Оператор switch управляет программой, позволяя задавать действия, которые будут выполняться при разных условиях. Break является командой выхода из оператора, default выполняется, если не выбрана ни одна альтернатива.

for	void setup()	Конструкция for используется для

	{
	pinMode(3, OUTPUT);
	}
	void loop()
	{
	for (int i=0; i <= 255; i++){
	analogWrite(3, i);
	delay(10);
	}
	}
		повторения при известном числе повторений.
		Пример: плавное затемнение светодиода. Заголовок цикла for состоит из трех частей: for (initialization; condition; increment) — initialization выполняется один раз, далее проверяется условие condition, если условие верно, то выполняется приращение increment и цикл повторяется.

while	void loop()
	{
	while (x < 10)
	{
	x = x + 1;
	Serial.println(x);
	delay(200);
	}
	}
		Оператор while используется, как цикл, который будет выполняться, пока условие в круглых скобках является истиной.
		Пример: оператор цикла while будет повторять код в скобках бесконечно до тех пор, пока x будет меньше 10.

do…while	void loop()
	{
	do
	{
	x = x + 1;
	delay(100);
	Serial.println(x);
	}
	while (x < 10);
	delay(900);
	}
		Оператор цикла do…while работает так же, как и цикл while. Однако, при истинности выражения в круглых скобках происходит продолжение работы цикла, а не выход из цикла.
		Пример: при x больше 10 операция сложения будет продолжаться, но с большей паузой (1000 мс).

break
continue
	switch (x)
	{
	case 1: digitalWrite(3, HIGH);
	case 2: digitalWrite(3, LOW);
	case 3: break;
	case 4: continue;
	default: digitalWrite(4, HIGH);
	}
		Break используется для принудительного выхода из циклов switch, do, for и while, не дожидаясь завершения цикла.
		Оператор continue пропускает оставшиеся операторы в текущем шаге цикла.

Синтаксис		
; **(точка с запятой)**	… digitalWrite(3, HIGH); …	Обязательная точка с запятой используется для обозначения конца оператора.

{}
(фигурные скобки)
	void setup()
	{
	pinMode(3, OUTPUT);
	digitalWrite(3, HIGH);
		Фигурные скобки образуют составной оператор, исполняющий включенные действия как единое целое.

	}	
// **(комментарий)**	x = 5; // комментарий	Комментарии используются для напоминания, как работает программа. Они игнорируются компилятором и не экспортируются в процессор, не занимая место в памяти.
#define	#define ledPin 3	Директива #define позволяет задать имя константе. Директива служит исключительно для удобства и улучшения читаемости программы.
#include	// библиотека для серво #include <Servo.h>	Директива #include используется для включения сторонних библиотек в скетч. Обратите внимание, что директивы #include и #define не требует точки с запятой в конце строки.

За основу взята информация из источника:
http://роботехника18.рф/%D1%8F%D0%B7%D1%8B%D0%BA-%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F-%D0%B0%D1%80%D0%B4%D1%83%D0%B8%D0%BD%D0%BE/

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

Методические указания

по выполнению лабораторных работ
по дисциплине «Микропроцессорные системы управления»
Для студентов направления подготовки 15.03.04 Автоматизация технологических процессов и производств,
направленность (профиль) Информационно-управляющие системы
(ЧАСТЬ 2)

(ЭЛЕКТРОННЫЙ ДОКУМЕНТ)

Содержание

ЛАБОРАТОРНАЯ РАБОТА №1 СНЯТИЕ И ОБРАБОТКА ПОКАЗАНИЙ ДАТЧИКОВ И ИСПОЛНИТЕЛЬНЫХ УСТРОЙСТВ, СОВМЕСТИМЫХ С ПЛАТФОРМОЙ ARDUINO

Цель работы: изучить особенности широтно-импульсной модуляции и процесс получения данных на платформе Arduino посредством реализации програмного модуля.

2.1. Теоретические сведения

2.1.1. Широтно-импульсная модуляция

Пример использования аналогового выхода для управления светодиодом доступен из меню *Файл* среды разработки Arduino.

Широтно-импульсная модуляция (ШИМ) (рис. 2.1) – это операция получения изменяющегося аналогового значения посредством цифровых устройств. Устройства используются для получения прямоугольных импульсов – сигналов, которые постоянно переключаются между максимальным и минимальным значениями. Данный сигнал моделирует напряжение между максимальным (5 В) и минимальным (0 В) значениями, изменяя при этом длительность времени включения 5 В относительно включения 0 В. Длительность включения максимального значения называется шириной импульса. Для получения различных аналоговых величин изменяется ширина импульса. При достаточно быстрой смене периодов включения-выключения можно подавать постоянный сигнал между 0 и 5 В на светодиод, тем самым управляя яркостью его свечения.

На диаграмме линии деления отмечают постоянные временные периоды. Длительность периода обратно пропорциональна частоте ШИМ. То есть если частота ШИМ составляет 500 Гц, то указанные линии будут отмечать интервалы длительностью в 2 мс каждый. Вызов функции analogWrite() с масштабом 0–255 означает, что значение analogWrite(255) будет соответствовать 100%-му рабочему циклу (постоянное включение 5 В), а значение analogWrite(127) – 50%-му рабочему циклу.

Для примера можно взять платформу и аккуратно начать двигать ее влево-вправо, что «превращает» мигание светодиода в светящиеся линии. Наращивание или уменьшение ширины импульса будет увеличивать или уменьшать светящиеся линии светодиода.

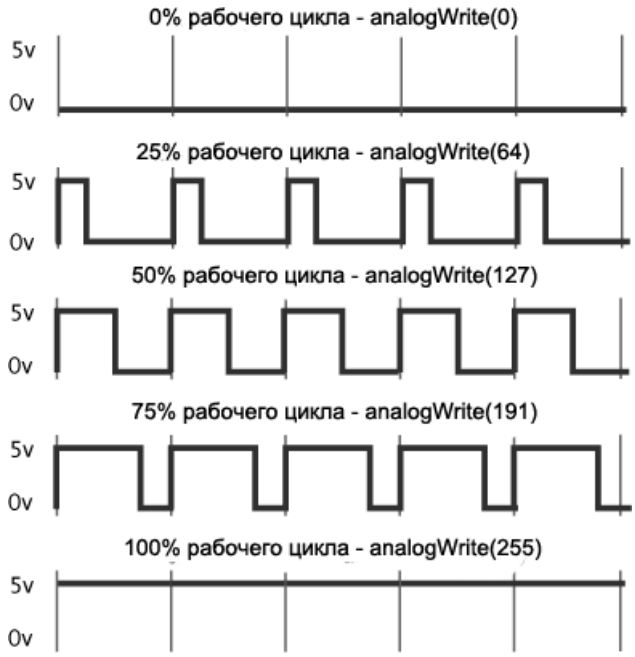


Рис. 2.1. Широтно-импульсная модуляция

2.1.2. Функции библиотеки Serial

Основные функции следующие:

1. *Serial.read()* – функция читает 1 байт данных из поступившей на Arduino информации. Каждый вызов этого метода будет возвращать следующий байт данных из тех, что поступили на Arduino. Если возвращать нечего, то данная функция вернет –1 (минус один).

Допустим, был отправлен 1 байт данных на Arduino и использован нижеприведенный участок кода:

```
int incomingByte;
void loop()
{
    if (Serial.available() > 0)
    {
      incomingByte = Serial.read();
    }
}
```

После того как считается этот байт данных, он будет перемещен в используюмую переменную, а функция Serial.available() снова будет возвращать пустое значение (Null), пока не поступят новые данные. То есть, когда считывается байт, показания счетчика принятых байтов уменьшаются и Serial.available() будет показывать на 1 байт меньше.

Следует отметить, что данная функция возвращает только 1 байт данных.

Если нужно передать 4 символа каждый по 1 байту, потребуется 4 раза вызвать

данную функцию, чтобы прочитать эти символы и самостоятельно разместить их в массив символов либо воспользоваться функцией Serial.readBytes().

2. *Serial.readBytes()* – функция практически идентична Serial.read().

В данную функцию нужно передать переменную массива типа char[] или byte[] и числовое значение, соответствующие тому, какое количество байтов из поступивших нужно прочитать. Прочитанные байты будут размещены в переданном массиве и убраны из счетчика функции Serial.available().

Следует отметить, что массив должен быть достаточно большим, чтобы вместить то количество байтов, которое требуется. Для примера можно получить на Arduino строку приветствия «Hi max» и вернуть ее на персональный компьютер:

```
char incomingBytes[7];
void loop()
{
    if (Serial.available() > 0)
    {
       Serial.readBytes(incomingBytes, 6);
       Serial.pritln(incomingBytes);
    }
}
```

В указанном блоке кода проверяется, что данные поступили, затем вызывается функция, передается в нее массив, в который требуется записать полученные байты данных, и указывается количество байтов, которое требуется прочитать. Функция осуществит чтение 6 байт и последовательно разместит их в ячейки массива, после чего полученный массив символов передается обратно на персональный компьютер.

Следует отметить, что на работу данной функции влияют настройки Serial.setTimeout(). Функция прекращает свою работу досрочно, как только считает указанное количество байтов, либо будет ждать установленный функцией Serial.SetTimeout() промежуток времени.

Функция не очищает тот массив, в который размещает переданные байты, отсюда возникают ситуации наподобие следующей. Если передана строка «Hi max», она состоит ровно из 6 байт и каждый байт последовательно будет размещен в массиве с самого начала (на мониторе получится «Hi max»). Далее, если передать строку «Hello», она займет лишь 5 байт и будет записана в сформированный массив с самого начала. В результате на экране получится «Hellox». Лишняя буква «x» – это остаток от предыдущей строки, то есть если не записать все 6 байт или в программе не отчистить массив, то при передаче блока меньшего размера получатся остатки от прежнего более длинного блока, который хранится в массиве в результате предыдущего вызова функции.

3. *Serial.parseInt()* – функция просматривает данные, поступившие на Arduino, и ищет среди них набор кодов (чисел) от 48 до 57, которые соответствуют символам чисел от 0 до 9, а затем преобразует все это в правильное целочисленное значение.

Таким образом, если из порта передать число 72, данная функция увидит 2 последовательных байта – 55 и 51, корректно преобразует их в число 72 и вернет как правильное целочисленное значение.

Эта функция может превратить строку числовых символов, разделенных нечисловыми символами, например запятой или символом новой строки, к набору правильных целочисленных значений, которые можно использовать в арифметических операциях.

4. *Serial.parseFloat()* – это аналог функции Serial.parseInt(). Разница в том, что данная функция пытается получить корректное число с плавающей точкой.

2.2. Порядок выполнения

Необходимые компоненты: плата Arduino Uno, термодатчик (1 вариант), фоторезистор (2 вариант), датчик Холла (3 вариант), ультразвуковой датчик (4 вариант) и USB-кабель.

Необходимые действия и рекомендации – корректное подключение соответствующих компонентов.

2.2.1. Индивидуальные задания

Написать программу для снятия и обработки данных, получаемых с датчика, выданного преподавателем. Порядок выбора варианта индивидуального задания соответствует номеру варианта по списку.

Вариант 1. Снять и обработать показания с термодатчика. Вывести значения в последовательный порт. Значения должны быть с точностью до 2 знаков после запятой.

Вариант 2. Снять и обработать показания с фоторезистора. Вывести значения в последовательный порт. Значения должны быть в процентном соотношении к минимальному/максимальному возможному значению.

Вариант 3. Снять и обработать показания с датчика Холла. Вывести значения в последовательный порт. Значения должны показывать расстояние в сантиметрах, на которых находится источник магнитных волн, относительно датчика.

Вариант 4. Снять и обработать показания с ультразвукового датчика. Вывести значения в последовательный порт. Значения должны показывать расстояние в метрах, на которых находится препятствие, с точностью до 3 знаков после запятой.

2.3. Содержание отчета

1. Цель работы.

2. Краткие теоретические сведения.
3. Индивидуальные задания.
4. Код программы и копии экрана с результатами вывода данных в последовательный порт.
5. Выводы.

2.4. Контрольные вопросы

1. Что такое ШИМ-сигнал?
2. Для чего изменяется ширина импульса?
3. С помощью какой команды можно регулировать уровень ШИМ-сигнала?
4. Для чего может быть использован ШИМ-сигнал?
5. В чем суть функции Serial.read()?
6. В чем суть функции Serial.readBytes()?
7. В чем суть функции Serial.parseInt()?
8. В чем суть функции Serial.parseFloat()?

Рекомендуемые источники

1. Встраиваемые системы управления [Электронный ресурс]. – Режим доступа : http://www.controlengrussia.com/programmnye-sredstva/vstraivaemy-e-sistemy-upravleniya/.
2. Хоровиц, П. Искусство схемотехники / П. Хоровиц, У. Хилл ; пер. с англ. – 2-е изд. – М. : БИНОМ. – 2014. – 704 с.

ЛАБОРАТОРНАЯ РАБОТА №2 УПРАВЛЕНИЕ НИЗКОВОЛЬТНЫМ ШАГОВЫМ ДВИГАТЕЛЕМ С ИСПОЛЬЗОВАНИЕМ ПЛАТФОРМЫ ARDUINO

Цель работы: изучить устройство и принцип работы шагового двигателя, разработать и программно реализовать алгоритм управления им.

3.1. Теоретические сведения

Шаговый электродвигатель (ШД) – это синхронный бесщеточный электродвигатель с несколькими расположенными в статоре обмотками, в котором ток, подаваемый в одну из обмоток, вызывает фиксацию ротора. Последовательная активация обмоток ШД вызывает дискретные угловые перемещения (шаги) ротора (рис. 3.1).

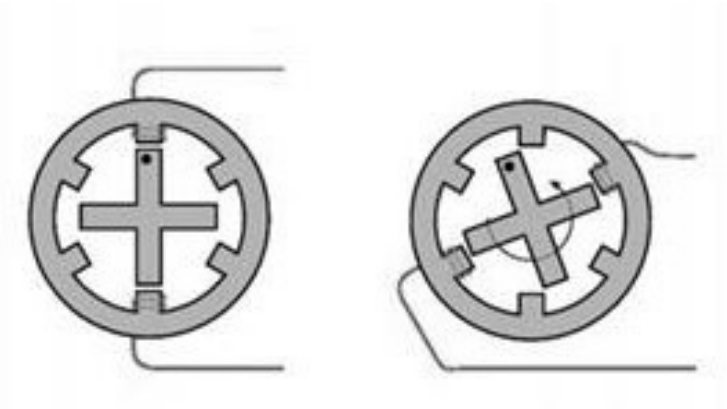


Рис. 3.1. Принцип работы ШД

Мощность шаговых двигателей лежит в диапазоне от единиц ватт до нескольких киловатт. Конструктивно ШД имеют различные исполнения и в целом их можно разделить на 3 типа:

1. Реактивные ШД, в которых ротор выполняется из магнитомягкого (ферромагнитного) материала.
2. ШД с постоянными магнитами, в которых ротор выполняется из магнитотвердого (магнитного) материала.
3. Гибридные ШД, сочетающие в себе лучшие черты реактивных ШД и ШД с постоянными магнитами.

Преимущества. Главное преимущество шаговых приводов – точность. При подаче потенциалов на обмотки шаговый двигатель повернется строго на определенный угол. К положительным моментам можно отнести стоимость шаговых приводов: в среднем в 1,5–2 раза дешевле сервоприводов. Шаговый привод как недорогая альтернатива сервоприводу наилучшим образом подходит для автоматизации отдельных узлов и систем, где не требуется

высокая динамика.

Недостатки. Возможность «проскальзывания» ротора – наиболее известная проблема этих двигателей. Это может произойти при превышении нагрузки на валу, неверной настройке управляющей программы (например, ускорение старта или торможения не адекватно перемещаемой массе) и приближении скорости вращения к резонансной. Наличие датчика позволяет обнаружить проблему, но автоматически скомпенсировать ее без остановки производственной программы возможно только в очень редких случаях. Чтобы избежать проскальзывания ротора, можно увеличить мощность двигателя (один из способов).

3.2. Порядок выполнения

Необходимые компоненты: плата Arduino Uno, ШД и USB-кабель (рис. 3.2).

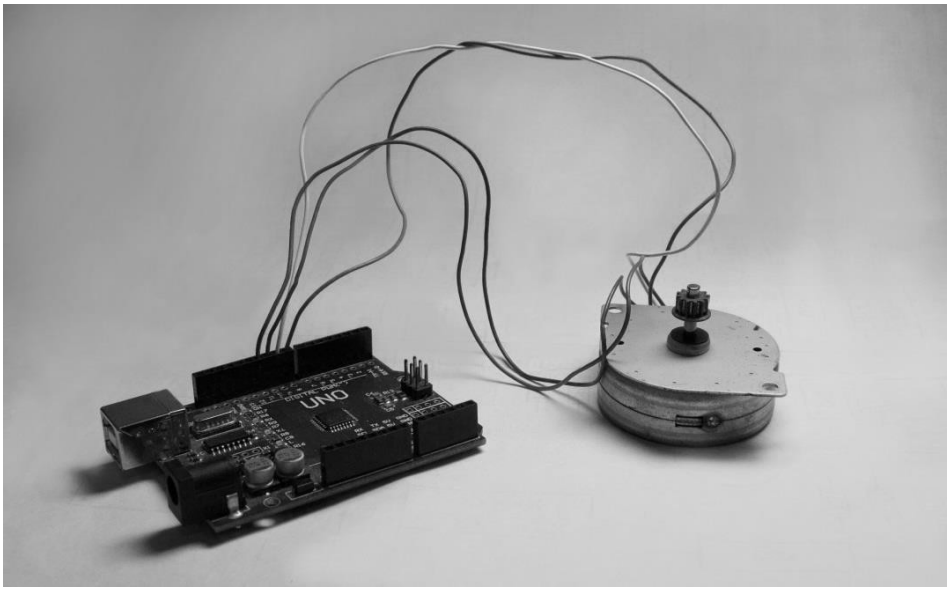


Рис. 3.2. Соединенные плата Arduino и ШД

Необходимые действия и рекомендации:

1. Соединение ШД и платы Arduino и подключение полученной системы с помощью USB-кабеля к компьютеру.
2. Для управления шаговым двигателем требуется подавать импульсы на выводы обмоток согласно типу и количеству обмоток ШД.
3. Для подачи одного импульса требуется подать напряжение на порт, после чего отключить его.
4. Длительность импульса может быть подобрана на практике либо взята из паспорта ШД.
5. Для фиксации положения ШД требуется подача сигнала на определенные выводы обмоток. Выводы обмоток берутся из паспорта ШД.

3.2.1. Индивидуальные задания

Вариант 1. Разогнать шаговый двигатель из состояния покоя до максимальной скорости, после чего сделать по два полных оборота в разные стороны и зафиксировать положение ШД.

Вариант 2. Постепенно увеличивать скорость ШД до половины максимального значения, после чего сменить направления и продолжать увеличивать скорость до максимального значения.

Вариант 3. Разработать и реализовать алгоритм, при котором двигатель делает один оборот по часовой стрелке, два – против часовой стрелки, три – по часовой и так далее до пяти оборотов.

Вариант 4. Разработать и реализовать алгоритм, при котором ШД будет постепенно увеличивать скорость до максимального значения, после чего снижать до минимального.

3.3. Содержание отчета

1. Цель работы.
2. Краткие теоретические сведения.
3. Индивидуальные задания.
4. Код программы.
5. Выводы.

3.4. Контрольные вопросы

1. Что такое шаговый двигатель?
2. Где могут быть применены шаговые двигатели?
3. С помощью чего регулируется скорость вращения ШД?

Рекомендуемые источники

Соммер, У. Программирование микроконтроллерных плат Arduino/Freeduino / У. Соммер. – СПб. : БХВ-Петербург, 2016. – 256 с.

ЛАБОРАТОРНАЯ РАБОТА №3 ВИЗУАЛИЗАЦИЯ ИНФОРМАЦИИ ПОСРЕДСТВОМ ЖИДКОКРИСТАЛЛИЧЕСКОГО ЭКРАНА, РЕАЛИЗОВАННОЙ НА ПЛАТФОРМЕ ARDUINO ПРИ ПОМОЩИ I2C ШИНЫ

Цель работы: изучить процесс подключения LCD-дисплея и произвести вывод данных с датчика.

4.1. Теоретические сведения

Жидкокристаллический дисплей (LCD-дисплей) – плоский дисплей на основе жидких кристаллов, а также устройство на основе такого дисплея. Простые приборы с дисплеем могут иметь монохромный или 2–5-цветный дисплей. Многоцветное изображение в более сложных устройствах формируется с помощью RGB-триад. Дисплей на жидких кристаллах используется для отображения графической или текстовой информации в компьютерных мониторах (также и в ноутбуках), телевизорах, телефонах, цифровых фотоаппаратах, электронных книгах, навигаторах, планшетах, электронных переводчиках, плеерах, термометрах, калькуляторах, часах, а также во многих других электронных устройствах.

Ранее в большинстве настольных мониторов, а также во всех дисплеях ноутбуков использовались матрицы с 18-битным цветом (6 бит на каждый RGB-канал), 24-битность эмулируется мерцанием с дизерингом. Жидкокристаллический дисплей с активной матрицей – разновидность жидкокристаллического дисплея, в котором используется активная матрица, управляемая тонкопленочными транзисторами.

В данной лабораторной работе используется дисплей LCD 1602 (рис. 4.1).

Рис. 4.1. Дисплей LCD 1602

Дисплеи LCD 1602 на базе контроллера HD44780 являются одними из самых простых, доступных и востребованных дисплеев для разработки различных электронных устройств. Его можно встретить как в устройствах,

собранных вручную, так и в промышленных устройствах, таких как, например,

автоматы для приготовления кофе. На базе данного дисплея собраны самые популярные модули в тематике Arduino, такие как LCD I2C-модуль и LCD Keypad Shield.

Подключать данный экран возможно с помощью 2 вариантов:

1. Прямое подключение к Arduino (рис. 4.2).
2. Использование дополнительных модулей (рис. 4.3).

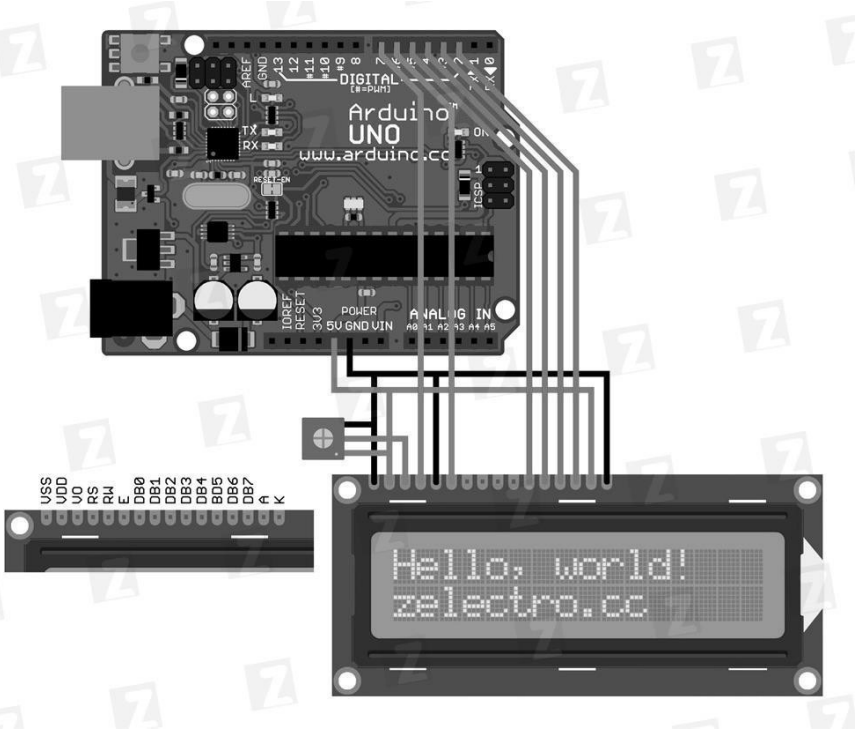


Рис. 4.2. Прямое подключение к Arduino

В данной работе используется второй вариант, так как это значительно упрощает работу и настройку данного экрана. В качестве дополнительного модуля потребуется конвертор в IIC/I2C.

Этот способ подключения работает только со специальной библиотекой. Загрузить библиотеку можно по названию LiquidCrystal_I2C1602V1.

Пример кода, реализующего инициализацию, подсветку и вывод информации на дисплей:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27,16,2); // Устанавливаем дисплей
void setup()
{
          lcd.init();
          lcd.backlight();// Включаем подсветку дисплея
          lcd.print("MyFirstText");
          lcd.setCursor(8, 1);
```

```
            lcd.print("LCD 1602");
    }
    void loop()
    {
            // Устанавливаем курсор на вторую строку и на
нулевой символ
            lcd.setCursor(0, 1);
             // Выводим на экран количество секунд с момента
запуска Arduino
            lcd.print(millis()/1000);
    }
```

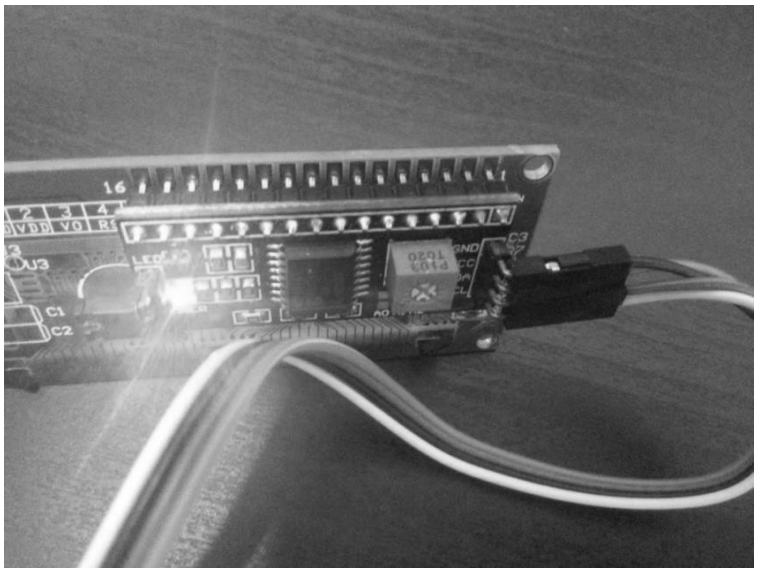


Рис. 4.3. Использование дополнительных модулей

4.2. Порядок выполнения

Необходимые компоненты: плата Arduino Uno, LCD-дисплей и USB-кабель.

Необходимые действия и рекомендации:

1. Соединение LCD-дисплея и платы Arduino.
2. Подключение полученной системы с помощью USB-кабеля к компьютеру.
3. Написание согласно примеру скетча для LCD-дисплея.
4. Вывод результата на экран.

4.2.1. Индивидуальные задания

Требуется вывести на экран в первой строке дисплея данные из лабораторной работы №2, а во второй – фамилию и инициалы студента.

4.3. Содержание отчета

1. Цель работы.
2. Краткие теоретические сведения.
3. Индивидуальные задания.
4. Код программы.
5. Выводы.

4.4. Контрольные вопросы

1. Что такое жидкокристаллический дисплей?
2. Что формируется с помощью RGB-триад?
3. Где используются жидкокристаллические дисплеи?
4. Назовите способы подключения жидкокристаллических дисплеев к отладочным платам.
5. Каковы преимущества I2C-шины?

Рекомендуемые источники

1. Соммер У. Программирование микроконтроллерных плат Arduino/Freeduino / У. Соммер. – СПб. : БХВ-Петербург, 2016. – 256 с.
2. Arduino – Home [Электронный ресурс]. – Режим доступа : https://www.arduino.cc/.
3. Banzi, M. Getting Started with Arduino / M. Banzi. – USA, 2011. – 72 с.
4. Блум, Дж. Изучаем Ардуино / Дж. Блум. – СПб. : БХВ-Петербург, 2015. – 336 с.
5. iArduino.ru – Купить микроконтроллеры Arduino в Москве [Электронный ресурс]. – Режим доступа : http://iarduino.ru/.

ЛАБОРАТОРНАЯ РАБОТА №4
РЕАЛИЗАЦИЯ РЕЛЯЦИОННОЙ СТРУКТУРЫ БАЗЫ ДАННЫХ И РАЗМЕЩЕНИЕ В ЕЕ ОБЪЕКТАХ ИНФОРМАЦИИ, СЧИТАННОЙ С ДАТЧИКОВ ВСТРАИВАЕМОЙ СИСТЕМЫ ИЗМЕРЕНИЙ И МОНИТОРИНГА

Цель работы: изучить средство реализации реляционной структуры базы данных и размещение в ее объектах информации, считанной с датчиков встраиваемой системы измерений и мониторинга.

5.1. Теоретические сведения

5.1.1. Структура системы сбора и обработки результатов измерений и мониторинга

Основными технологическими платформенными составляющими системы сбора и обработки результатов измерений и мониторинга являются:

– *аппаратная* (оборудование (солнечный коллектор) и модуль сбора *данных* (устройство сбора данных (контроллеры и интерфейсы) и блок датчиков, реализующий сбор, преобразование, передачу и форматирование данных));

– *программная* (серверное программное обеспечение для мониторинга параметров (клиентская часть) и база данных, в частности, для получения, обработки, манипулирования, хранения и визуализации данных);

– *информационная* (интерактивные электронные информационные ресурсы с возможностью аутентификации через Web-интерфейс и справочная документация) (рис. 5.1).

Блок датчиков содержит в себе как аналоговые, так и цифровые измерительные устройства, регистрирующие текущие параметры работы коллектора. Устройство сбора данных предназначено для контроля работы датчиков, их опроса и обеспечения передачи данных между датчиками и сервером. Физически оно представляет собой прибор, управляемый микроконтроллером. Сервер обеспечивает сбор и хранение полученной информации, а также формирует данные для статистических отчетов. Компьютеры пользователей выступают в роли клиентов, формирующих при помощи Web-браузера различные запросы и получающих ответы от сервера посредством локальной компьютерной сети или глобальной сети Internet.

Передача данных осуществляется по сетевому протоколу HTTP. Сервер сохраняет информацию в специальной базе данных. Такой подход имеет ряд преимуществ, поскольку исключает сильную зависимость одних компонентов системы от других. Передача и накопление информации происходят под

управлением серверного программного обеспечения.

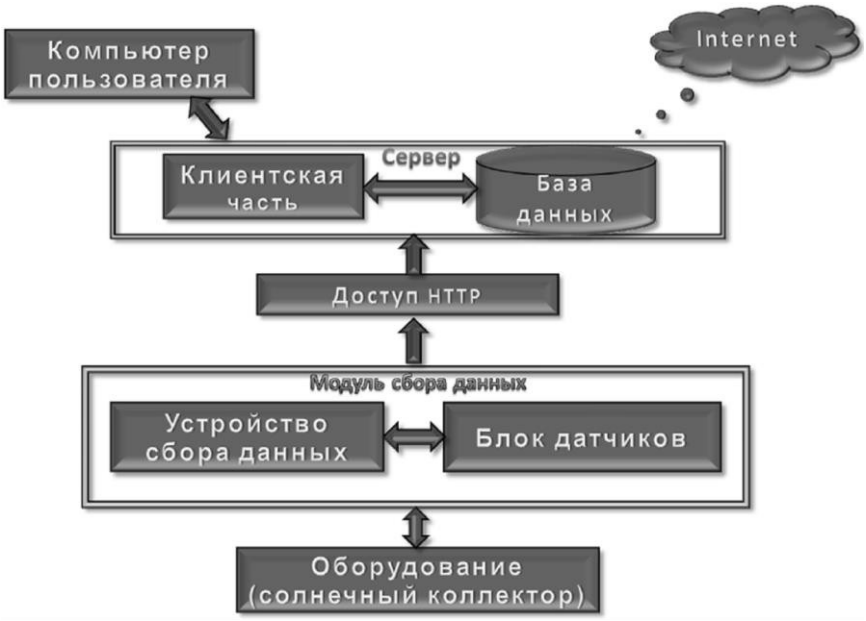


Рис. 5.1. Структурная схема системы сбора и обработки результатов измерений и мониторинга

Хранение данных, поступающих от автономных датчиков температуры теплоносителя на входе и выходе коллектора, на входе, выходе и внутри бака (бойлера) и датчиков расхода (перемещаемого объема) теплоносителя, может быть реализовано в стандартном текстовом формате представления табличных данных CSV или MDF или структурированном XML. Измерив указанные показатели в совокупности с интенсивностью света, можно рассчитать (оценить) мощность технической установки, а также потери при транспортировке тепла с отображением полученных результатов в соответствующем виде. Для хранения информации используется сервер с системой управления базами данных, доступ к которому организован не напрямую, а через Web-сервер. Модуль сбора данных периодически опрашивает датчики, формирует пакет данных и отправляет POST-запрос Web-серверу. На стороне сервера реализовано программное обеспечение для работы с данными, разработанное на языке программирования C#. Такой подход повышает надежность всей системы в целом и делает ее более гибкой. Например, при обновлении SQL-сервера, переходе на другую систему управления базами данных или изменении структуры самой базы данных не требуется вносить изменения в программное обеспечение. К тому же данный подход подразумевает достаточную простоту представления измеренных данных в сети Internet.

5.1.2. Система управления базами данных

Среда *Microsoft SQL Server Management Studio 2012* (SSMS) – это графический набор средств для разработки сценариев на структурированном языке запросов T-SQL и управления всеми компонентами сервера баз данных Microsoft SQL Server. SSMS является основным инструментом любого разработчика или администратора указанного сервера баз данных.

В SSMS объединены возможности таких программ, как Enterprise Manager, Query Analyzer и Analysis Manager.

С помощью SSMS можно подключаться к любым компонентам SQL Server, таким как Database Engine, службы Integration Services, службы Analysis Services и службы Reporting Services (рис. 5.2).

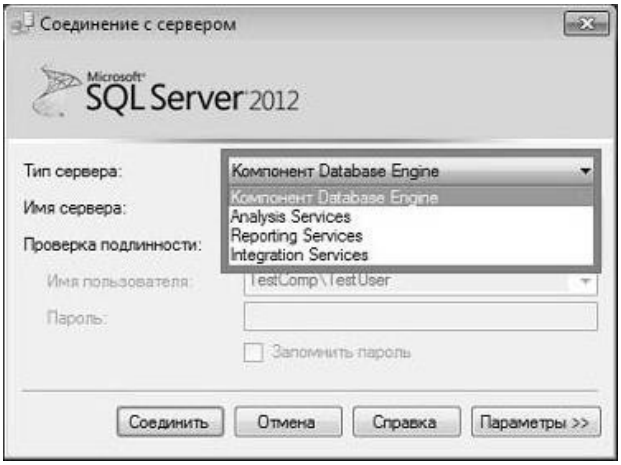


Рис. 5.2. Подключение к компонентам Microsoft SQL Server

Другими словами, с помощью SSMS можно управлять не только основным компонентом Database Engine, но и другими немаловажными компонентами. Таким образом, SSMS – это единая полнофункциональная программа по управлению SQL-сервером.

5.1.3. Обозреватель объектов

В среде SSMS встроен обозреватель объектов, который позволяет просматривать все объекты сервера и предоставляет графический интерфейс для управления этими объектами. Для запуска можно использовать меню *Вид* → *Обозреватель объектов*, хотя последний, как правило, отображен по умолчанию (рис. 5.3).

То есть с помощью обозревателя объектов можно посмотреть, какие базы данных, таблицы, функции и так далее существуют в проекте, какие пользователи созданы и какие связанные серверы настроены, в общем, получить доступ ко всем объектам сервера.

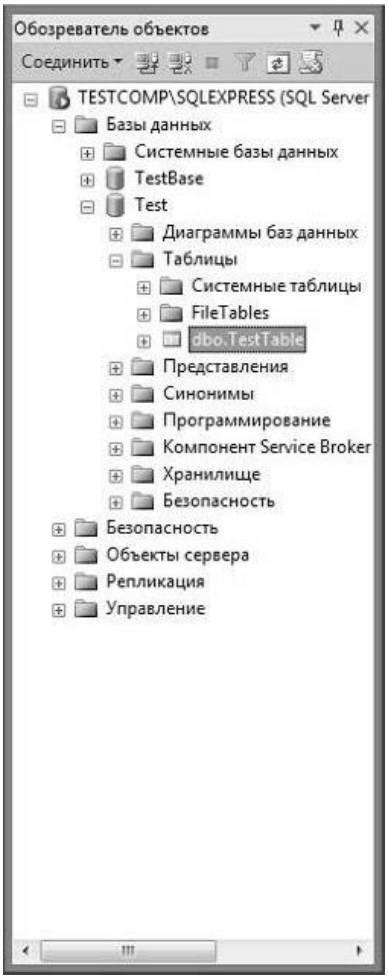


Рис. 5.3. Обозреватель объектов

5.1.4. Создание и редактирование сценариев

Данная возможность позволяет писать на T-SQL запросы или скрипты, то есть именно здесь пишутся все SQL-инструкции. Написать запрос к базе или SQL-инструкцию можно с помощью редактора кода SSMS. Чтобы открыть окно редактора кода, нужно нажать *Создать запрос* (рис. 5.4).

5.1.5. Просмотр предлагаемого плана выполнения запроса

Для упрощения оптимизации запросов в SSMS в редактор кода встроен функционал, который позволяет посмотреть план выполнения запросов, и в случае если он не оптимален, он предложит, например, создать тот или иной индекс (рис. 5.5).

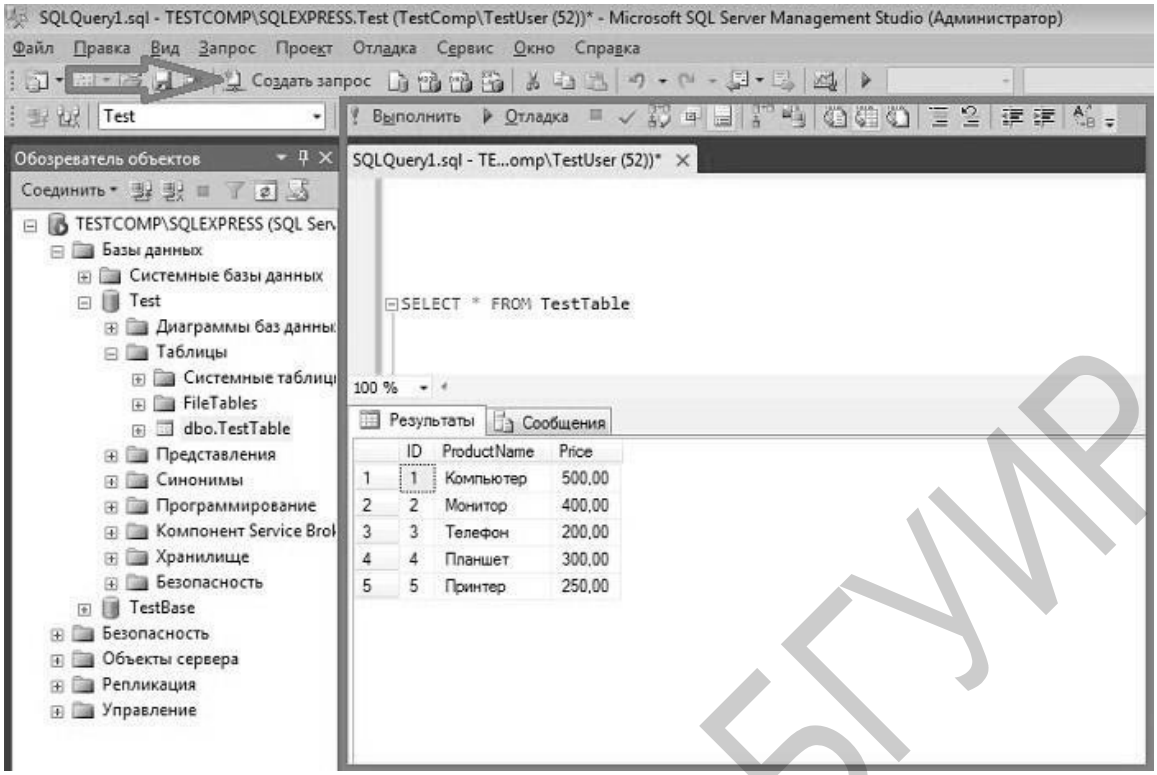


Рис. 5.4. Создание и редактирование сценариев

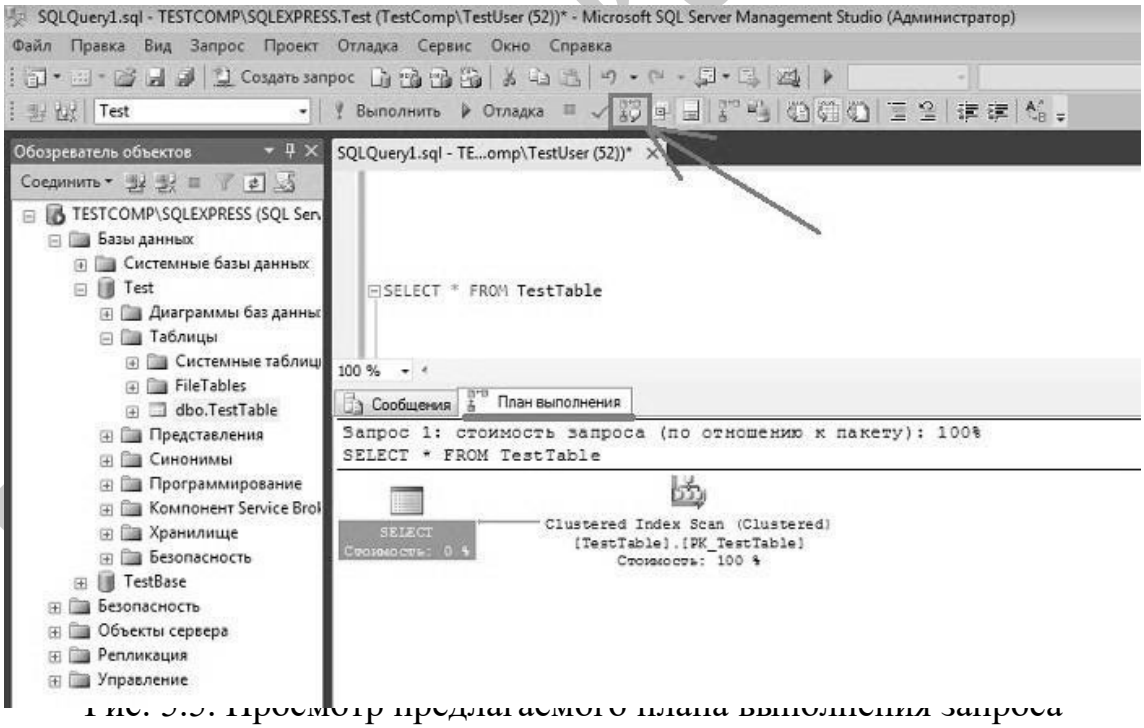


Рис. 5.5. Просмотр предлагаемого плана выполнения запроса

5.1.6. Обозреватель решений

В SSMS есть такой функционал, с помощью которого можно сгруппировать все сценарии, или скрипты, в один проект с целью их систематизации, удобного хранения и доступа к ним. Для отображения данного обозревателя следует нажать *Вид* → *Обозреватель решений* (рис. 5.6).

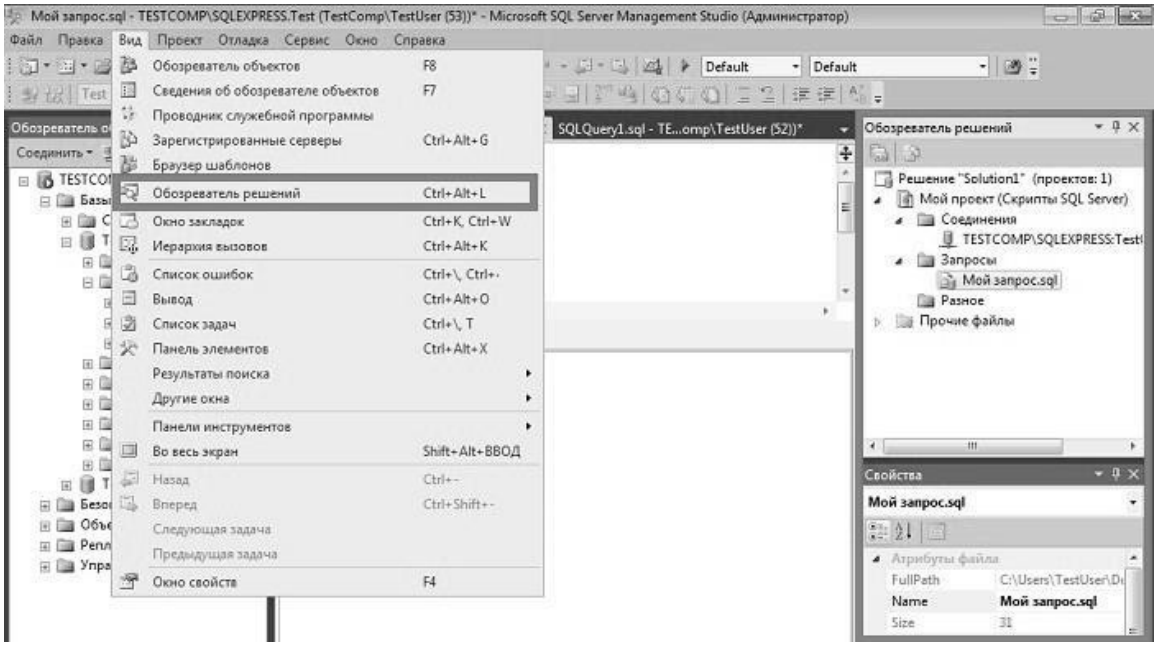


Рис. 5.6. Обозреватель решений

5.1.7. Браузер шаблонов

Для облегчения написания типичных сценариев создания или модификации объектов базы данных в среде SSMS есть возможность использовать встроенные шаблоны SQL-инструкций, то есть своего рода заголовки этих сценариев. Другими словами, если понадобилось, например, создать или изменить таблицу, но забыт синтаксис, можно открыть обозреватель шаблонов *Вид* → *Браузер шаблонов*, выбрать подходящий и всего лишь подставить нужные названия объектов (рис. 5.7).

Также существует возможность создавать собственные шаблоны для часто выполняемых задач или для случаев, когда нет подходящего встроенного шаблона.

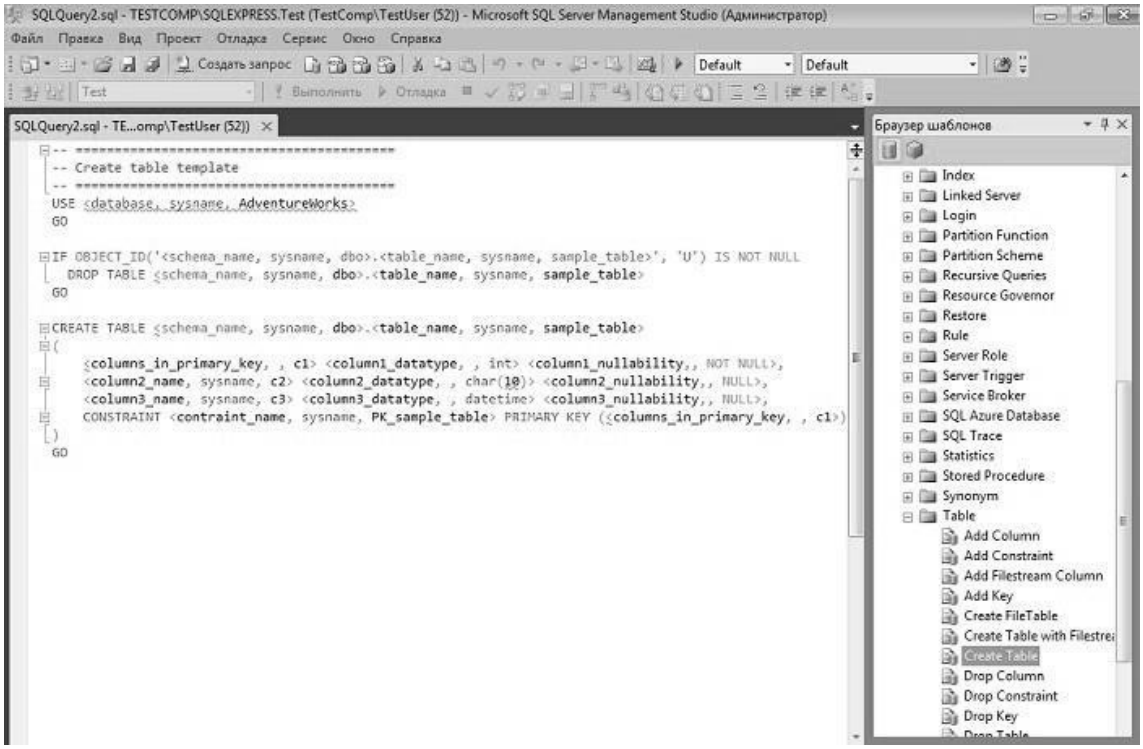


Рис. 5.7. Браузер шаблонов

5.1.8. Монитор активности

В среде SSMS есть средство, позволяющее контролировать текущую активность на сервере, например: какие запросы в данный момент выполняются, какие пользователи подключены и так далее. Для запуска можно щелкнуть правой кнопкой мыши на сервере в обозревателе объектов и выбрать *Монитор активности* или щелкнуть на значке на панели инструментов (рис. 5.8).

5.1.9. Создание резервных копий баз данных и восстановление из backup

С помощью SSMS можно легко в графическом интерактивном режиме создавать резервные копии баз данных, а также восстанавливать базы из backup. Чтобы это сделать, нужно щелкнуть правой кнопкой мыши на необходимой базе данных: *Задачи* → *Создать резервную копию/Восстановить* (рис. 5.9).

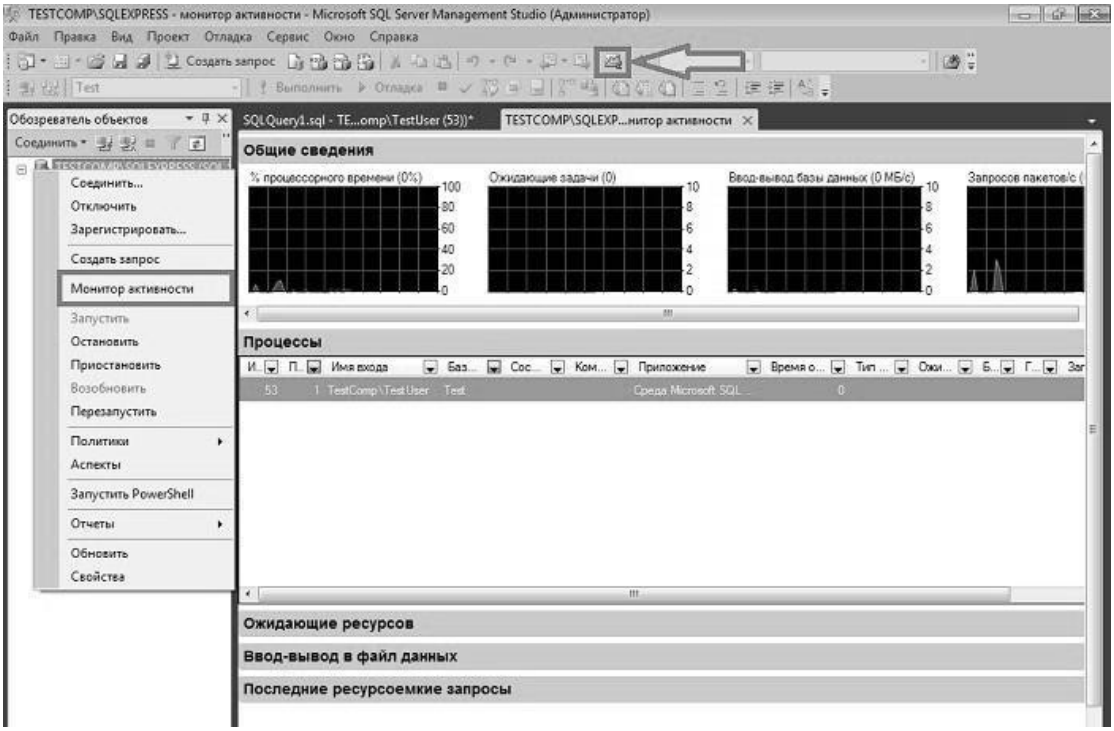
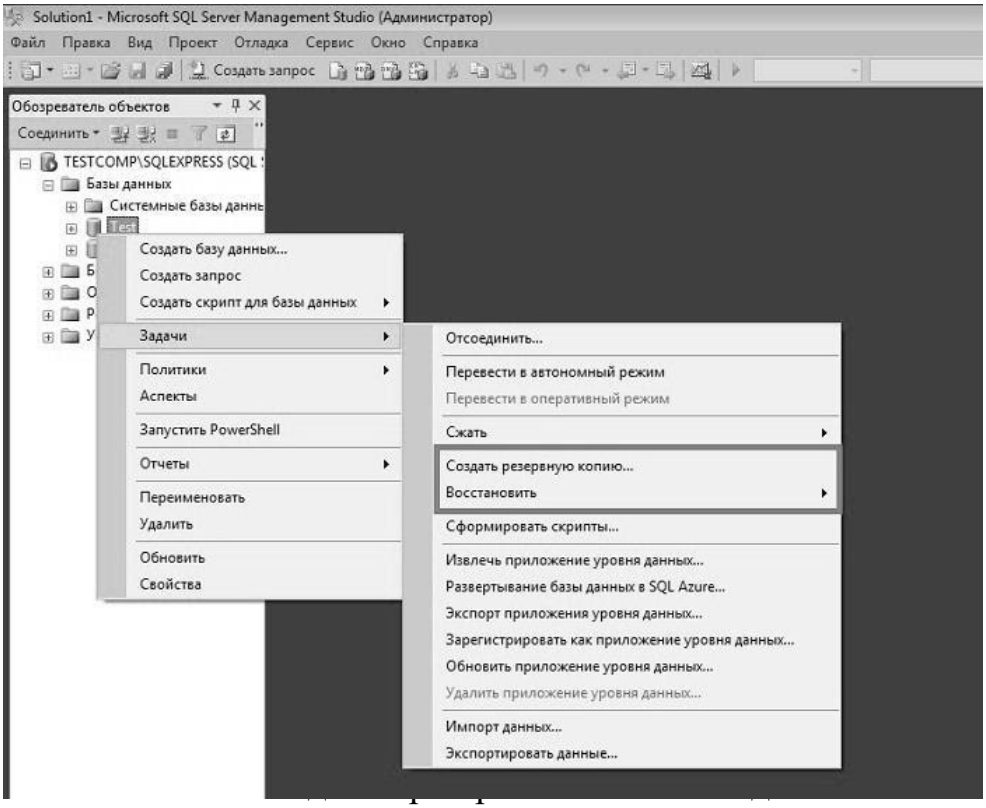


Рис. 5.8. Монитор активности



Восстановление баз данных из резервных копий

5.1.10. Настройка свойств сервера, баз данных и других объектов

Среда SSMS позволяет изменять свойства сервера и объектов этого сервера. Практически у каждого объекта на SQL-сервере есть свойства, которые как раз и можно изменить с помощью графических инструментов SSMS или просто посмотреть. Например для редактирования свойств базы данных необходимо выбрать базу, щелкнуть на ней правой кнопкой мыши и выбрать *Свойства*. Некоторые свойства доступны только для чтения, а некоторые можно изменить, например в данном случае можно перейти в раздел *Параметры* и изменить необходимые параметры (рис. 5.10).

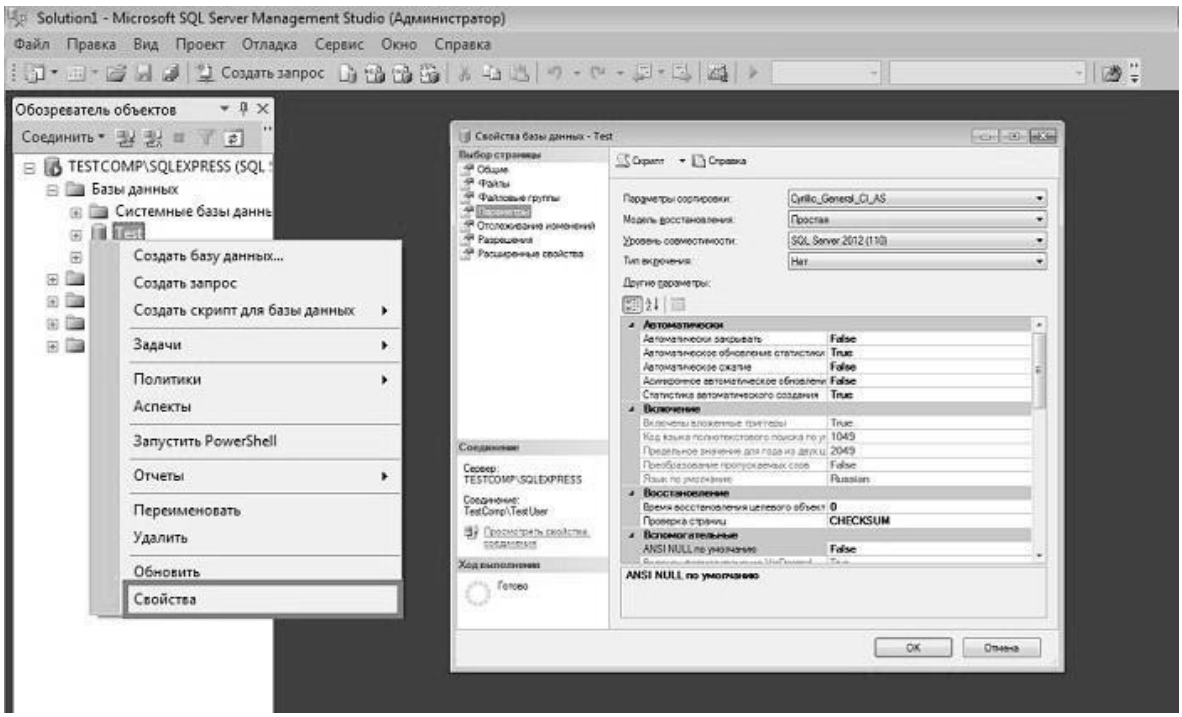


Рис. 5.10. Настройка свойств сервера, баз данных и других объектов

5.1.11. Присоединение и отсоединение баз данных

В SSMS есть возможность присоединять и отсоединять базы данных. Например, если возникла необходимость перенести базу данных с одного сервера на другой, это можно сделать с помощью графических инструментов SSMS. Для этого на одном сервере нужно правой кнопкой мыши отсоединить базу данных: *Задачи* → *Отсоединить…* (рис. 5.11).

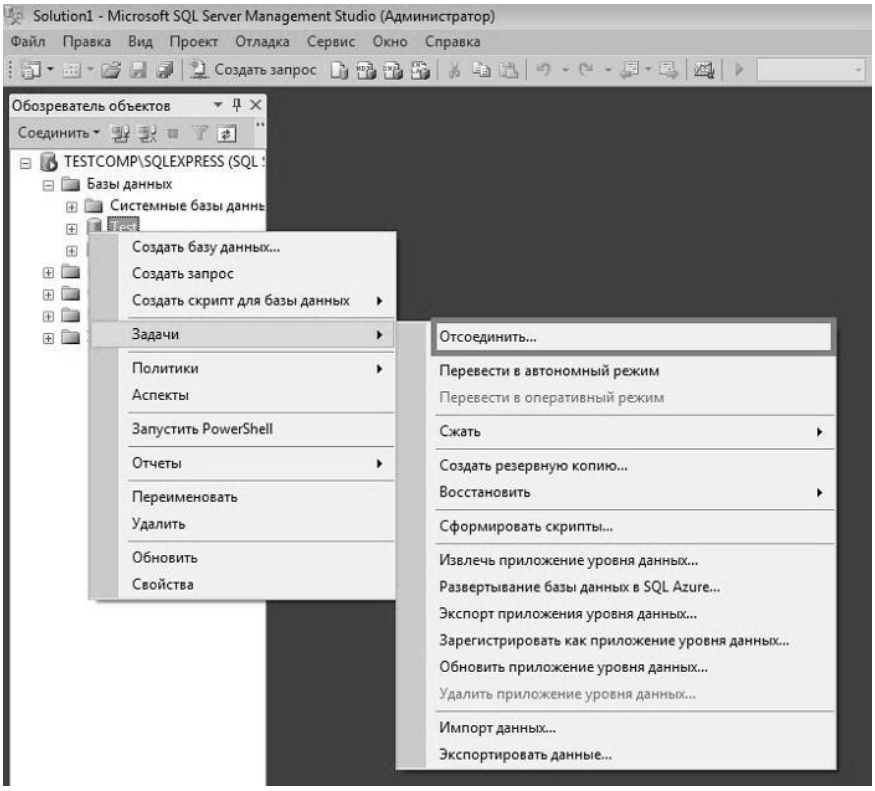


Рис. 5.11. Отсоединение баз данных

Затем можно скопировать файлы базы данных на новый сервер и в SSMS щелкнуть правой кнопкой мыши на объекте *Базы данных,* далее – *Присоединить*… (рис. 5.12).

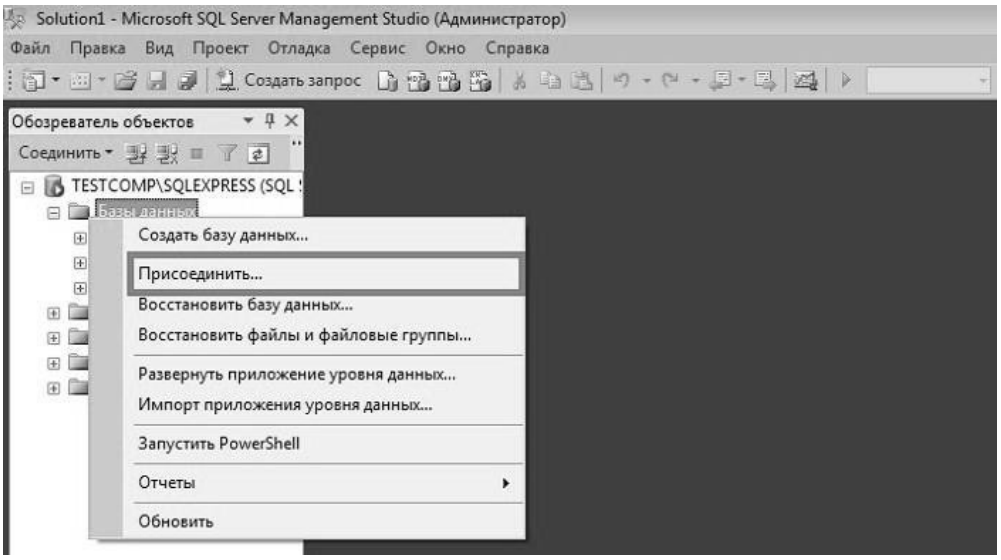


Рис. 5.12. Присоединение баз данных

5.1.12. Управление безопасностью сервера

Среда SSMS предназначена и для управления безопасностью SQL-сервера, то есть с помощью нее можно создать имя входа на сервер, пользователя базы данных или, например, настраивать доступ к объектам сервера. Для того чтобы, например, создать пользователя базы данных, необходимо выбрать *Базы данных* → *Нужная база данных* → *Безопасность* → *Пользователи (щелчок правой кнопки мыши)* → *Создать пользователя* (рис. 5.13).

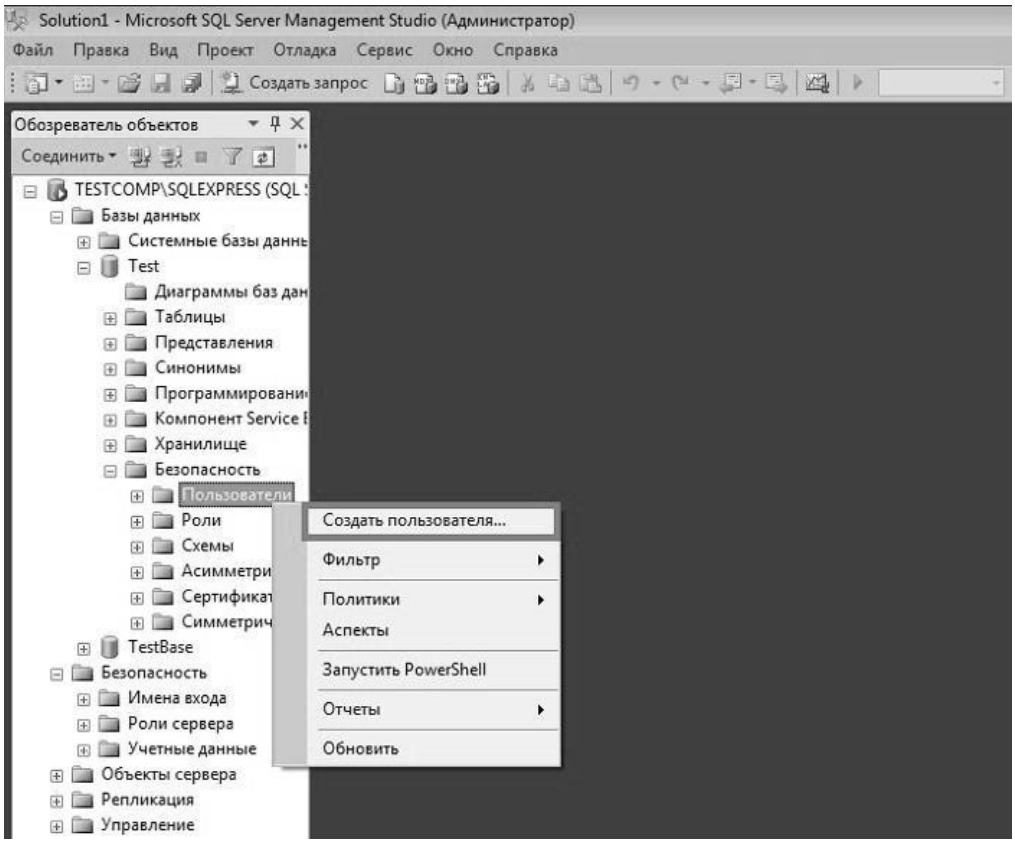


Рис. 5.13. Создание пользователя

В SSMS есть графические инструменты для создания связанных серверов. Функционал доступен в контейнере *Объекты сервера* → *Связанные серверы* (рис. 5.14).

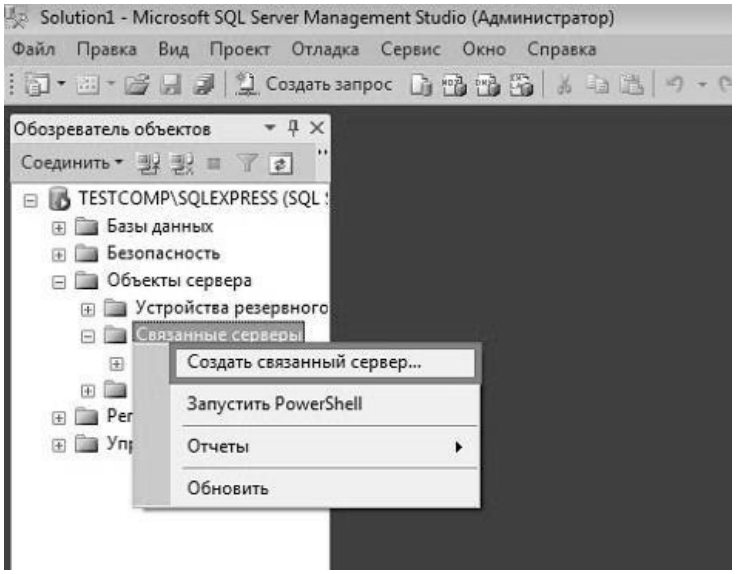


Рис. 5.14. Создание пользователя

5.1.13. Настройка репликации баз данных

Для настройки репликации баз данных есть отдельный контейнер *Репликация* (рис. 5.15).

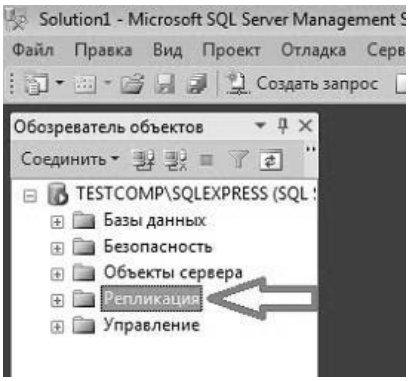


Рис. 5.15. Настройка репликации баз данных

5.1.14. Выполнение административных задач

SSMS выступает и средством администрирования SQL-сервера. С помощью данной среды можно, например, создавать планы обслуживания баз данных или просматривать системные журналы. Функционал доступен в контейнере *Управление* (рис. 5.16).

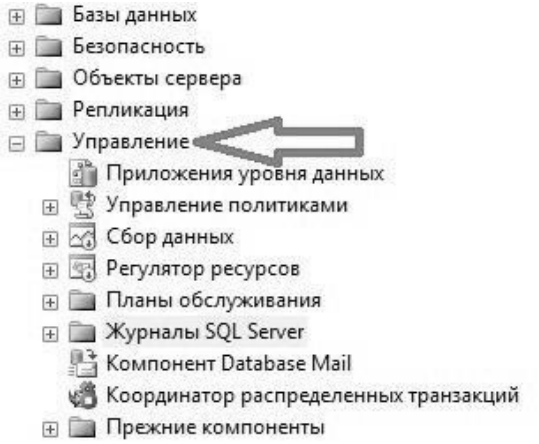


Рис. 5.16. Выполнение административных задач

Более подробную справку можно найти на официальном Web-сайте корпорации Microsoft.

5.1.15. Процедуры, хранимые в среде Microsoft SQL Server

При работе в Microsoft SQL Server пользователи могут создавать собственные *хранимые процедуры*, реализующие те или иные действия. Эти процедуры являются полноценными объектами базы данных, а потому каждая из них хранится в конкретной базе данных. Непосредственный вызов хранимой процедуры возможен, только если он осуществляется в контексте той базы данных, где находится процедура.

В Microsoft SQL Server имеется несколько типов хранимых процедур.

1. *Системные хранимые процедуры* предназначены для выполнения различных административных действий. Практически все действия по администрированию сервера выполняются с их помощью. Можно сказать, что системные хранимые процедуры являются интерфейсом, обеспечивающим работу с системными таблицами, которая, в конечном счете, сводится к изменению, добавлению, удалению и выборке данных из системных таблиц как пользовательских, так и системных баз данных. Системные хранимые процедуры имеют префикс *sp_*, хранятся в системной базе данных и могут быть вызваны в контексте любой другой базы данных.

2. *Пользовательские хранимые процедуры* реализуют те или иные пользовательские действия. Хранимые процедуры – полноценный объект базы данных. Вследствие этого каждая хранимая процедура располагается в конкретной базе данных, где и выполняется.

3. *Временные хранимые процедуры* существуют лишь некоторое время, после чего автоматически удаляются сервером. Они делятся на локальные и глобальные. Локальные временные хранимые процедуры могут быть вызваны только из того соединения, в котором созданы. При создании такой процедуры

ей необходимо дать имя, начинающееся с одного символа #. Как и все временные объекты, хранимые процедуры этого типа автоматически удаляются при отключении пользователя и перезапуске или остановке сервера. Глобальные временные хранимые процедуры доступны для любых соединений сервера, на котором имеется такая же процедура. Для ее определения достаточно дать ей имя, начинающееся с символов ##. Удаляются эти процедуры при перезапуске или остановке сервера, а также при закрытии соединения, в контексте которого они были созданы.

Создание хранимой процедуры предполагает решение следующих задач:

1. Определение типа создаваемой хранимой процедуры: временная или пользовательская. Кроме этого, можно создать свою собственную системную хранимую процедуру, назначив ей имя с префиксом *sp_* и поместив ее в системную базу данных. Такая процедура будет доступна в контексте любой базы данных локального сервера.

2. Планирование прав доступа. При создании хранимой процедуры следует учитывать, что она будет иметь те же права доступа к объектам базы данных, что и создавший ее пользователь.

3. Определение параметров хранимой процедуры. Подобно процедурам, входящим в состав большинства языков программирования, хранимые процедуры могут обладать входными и выходными параметрами.

4. Разработка кода хранимой процедуры. Код процедуры может содержать последовательность любых команд SQL, включая вызов других хранимых процедур.

Создание новой и изменение имеющейся хранимой процедуры осуществляется с помощью следующей команды:

```
<определение_процедуры>::=
{CREATE | ALTER } [PROCEDURE] имя_процедуры
 [;номер]
[{@имя_параметра тип_данных } [VARYING ]
[=default][OUTPUT] ][,...n]
[WITH { RECOMPILE | ENCRYPTION | RECOMPILE,
ENCRYPTION }]
[FOR REPLICATION]
AS
sql_оператор [...n]
```

Используя префиксы *sp_*, *#*, *##*, создаваемую процедуру можно определить в качестве системной или временной. Как видно из синтаксиса команды, не допускается указывать имя владельца, которому будет принадлежать создаваемая процедура, а также имя базы данных, где она должна быть размещена. Таким образом, чтобы разместить создаваемую хранимую процедуру в конкретной базе данных, необходимо выполнить команду *CREATE PROCEDURE* в контексте этой базы данных. При обращении из тела хранимой процедуры к объектам той же базы данных можно

использовать укороченные имена, то есть без указания имени базы данных. Когда же требуется обратиться к объектам, расположенным в других базах данных, указание имени базы данных обязательно.

Номер в имени – это идентификационный номер хранимой процедуры, однозначно определяющий ее в группе процедур. Для удобства управления процедурами логически однотипные хранимые процедуры можно группировать, присваивая им одинаковые имена, но разные идентификационные номера.

Для передачи входных и выходных данных в создаваемой хранимой процедуре могут использоваться параметры, имена которых, как и имена локальных переменных, должны начинаться с символа @. В одной хранимой процедуре можно задать множество параметров, разделенных запятыми. В теле процедуры не должны применяться локальные переменные, чьи имена совпадают с именами параметров этой процедуры.

Для определения типа данных, который будет иметь соответствующий параметр хранимой процедуры, годятся любые типы данных SQL, включая определенные пользователем. Однако тип данных *CURSOR* может быть использован только как выходной параметр хранимой процедуры, то есть с указанием ключевого слова *OUTPUT*.

Наличие ключевого слова *OUTPUT* означает, что соответствующий параметр предназначен для возвращения данных из хранимой процедуры. Однако это вовсе не означает, что параметр не подходит для передачи значений в хранимую процедуру. Указание ключевого слова *OUTPUT* предписывает серверу при выходе из хранимой процедуры присвоить текущее значение параметра локальной переменной, которая была указана при вызове процедуры в качестве значения параметра. Отметим, что при указании ключевого слова *OUTPUT* значение соответствующего параметра при вызове процедуры может быть задано только с помощью локальной переменной. Не разрешается использование любых выражений или констант, допустимое для обычных параметров.

Ключевое слово *VARYING* применяется совместно с параметром *OUTPUT*, имеющим тип *CURSOR*. Оно определяет, что выходным параметром будет результирующее множество.

Ключевое слово *DEFAULT* представляет собой значение, которое будет принимать соответствующий параметр по умолчанию. Таким образом, при вызове процедуры можно не указывать явно значение соответствующего параметра.

Так как сервер кэширует план исполнения запроса и компилированный код, при последующем вызове процедуры будут использоваться уже готовые значения. Однако в некоторых случаях все же требуется выполнять перекомпиляцию кода процедуры. Указание ключевого слова *RECOMPILE* предписывает системе создавать план выполнения хранимой процедуры при каждом ее вызове.

Параметр *FOR REPLICATION* востребован при репликации данных и включении создаваемой хранимой процедуры в качестве статьи в публикацию.

Ключевое слово *ENCRYPTION* предписывает серверу выполнить шифрование кода хранимой процедуры, что может обеспечить защиту от использования авторских алгоритмов, реализующих работу хранимой процедуры.

Ключевое слово *AS* размещается в начале собственно тела хранимой процедуры, то есть набора команд SQL, с помощью которых и будет реализовываться то или иное действие. В теле процедуры могут применяться практически все команды SQL, объявляться транзакции, устанавливаться блокировки и вызываться другие хранимые процедуры. Выход из хранимой процедуры можно осуществить посредством команды *RETURN*.

Удаление хранимой процедуры осуществляется командой *DROP PROCEDURE {имя_процедуры} [,...n]*.

5.2. Порядок выполнения

Необходимое программное обеспечение и исходные данные: Microsoft SQL Server Management Studio 2012 и файл с данными измерений и мониторинга.

Необходимые действия и рекомендации:

1. Формирование объектов базы данных и ее реляционной структуры, состоящей из 2 таблиц с полями:
   - «номер датчика»;
   - «пространственная координата датчика (долгота)»;
   - «пространственная координата датчика (широта)»;
   - «время и дата считывания показаний»;
   - «значение температуры».
2. Заполнение данными таблиц базы данных.
3. Создание хранимых процедур для обращения к данным таблиц:
   - выборка диапазона в соответствии с временем и датой считывания показаний;
   - определение минимума и максимума значений в выбранном временном интервале;
   - определение отклонения значений от среднего показателя;
   - определение территориального размещения датчиков в соответствии с выбранным регионом.
4. Формирование скриптов для автоматизации создания базы данных на удаленном сервере.

Для создания новой базы данных в SSMS нужно выбрать контейнер *Базы данных*, щелкнуть правой кнопкой мыши и выбрать в контекстном меню *Создать базу данных…* (рис. 5.17).

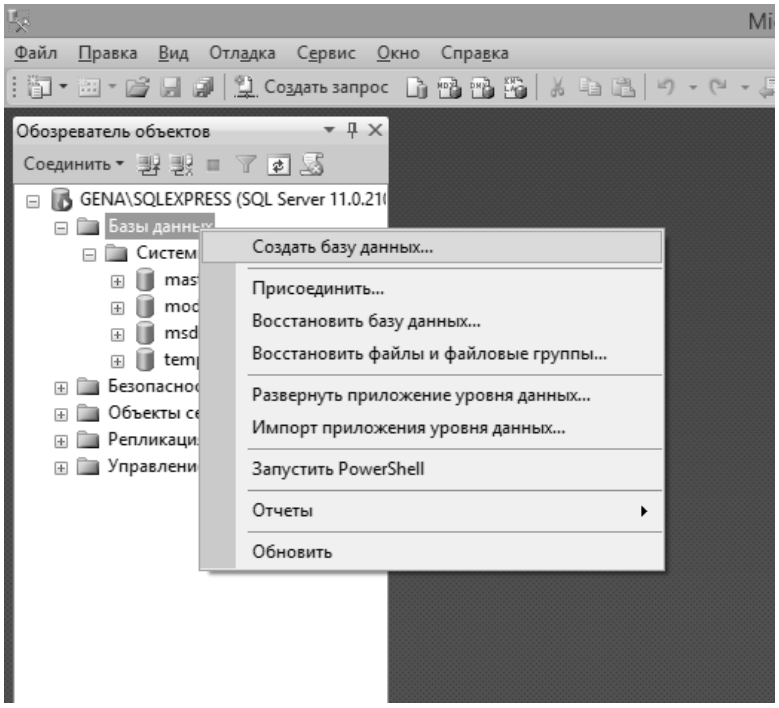


Рис. 5.17. Создание новой базы данных

Появляется форма, в которой нужно указать имя базы данных, например *Datchiki_temperatur* (рис. 5.18).

Создать новую базу данных можно и программным путем, используя конструкцию *CREATE DATABASE* на языке T-SQL (рис. 5.19).

Выполнив созданный запрос, можно прийти к результату, аналогичному тому, что получается при использовании графического пользовательского интерфейса.

Информация в базе данных хранится в таблицах, которые являются отображением некоторых логических сущностей. В базе данных *Datchiki_temperatur*, которая будет предназначена для доступа к хранящейся в ней информации (данных определенного формата) о значениях температур, снятых со специальных датчиков, будут необходимы 2 таблицы:

1. *Datchiki* – таблица будет содержать информацию о датчиках и включать в себя следующие поля:
   - «номер датчика»;
   - «пространственная координата датчика (долгота)»;
   - «пространственная координата датчика (широта)».
2. *Temperatura* – таблица будет хранить данные, поступающие с датчиков:
   - «номер датчика»;
   - «время и дата считывания показаний»;
   - «значение температуры».

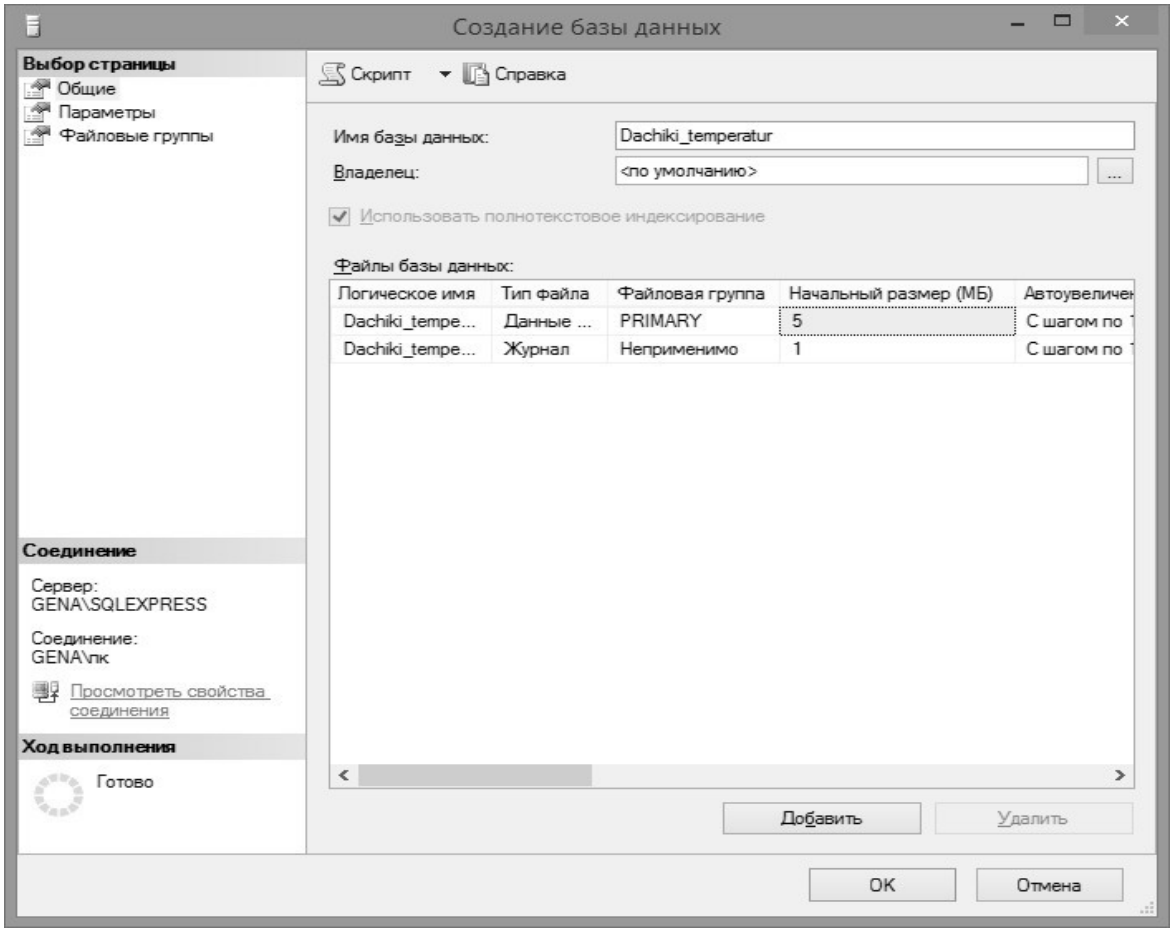


Рис. 5.18. Указание имени базы данных

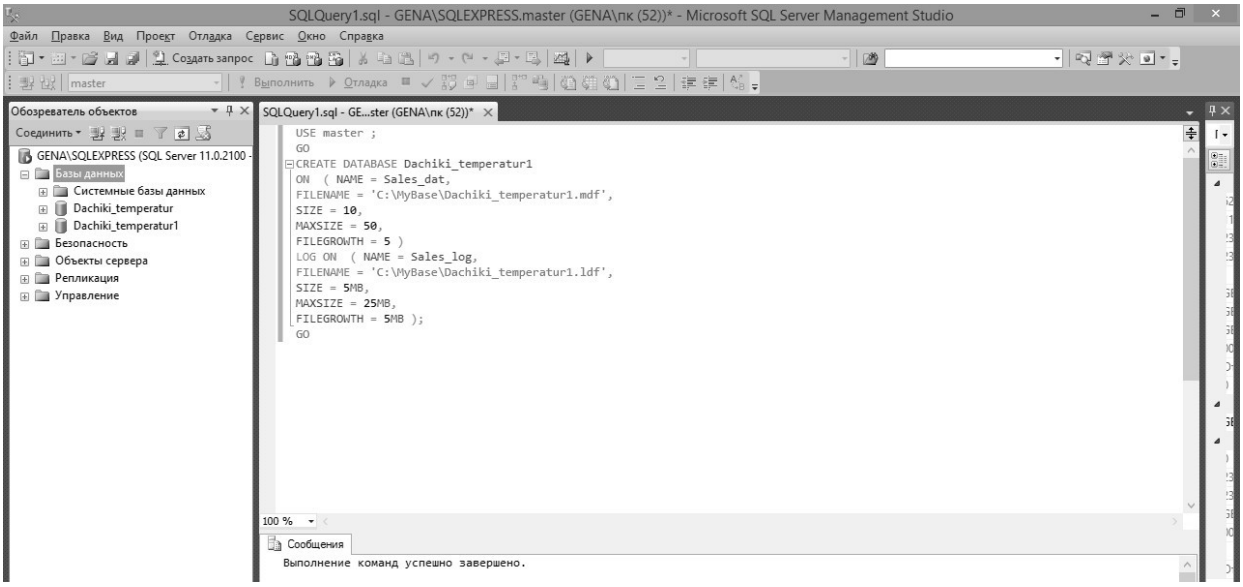


Рис. 5.19. Создание новой базы программным путем

Создание таблиц и их полей происходит через соответствующие контекстные меню SSMS: *База данных* → *Таблицы* → *Создать таблицу*…. В колонки *Имя столбца* и *Тип данных* следует внести соответствующие реквизиты полей таблицы (рис. 5.20).

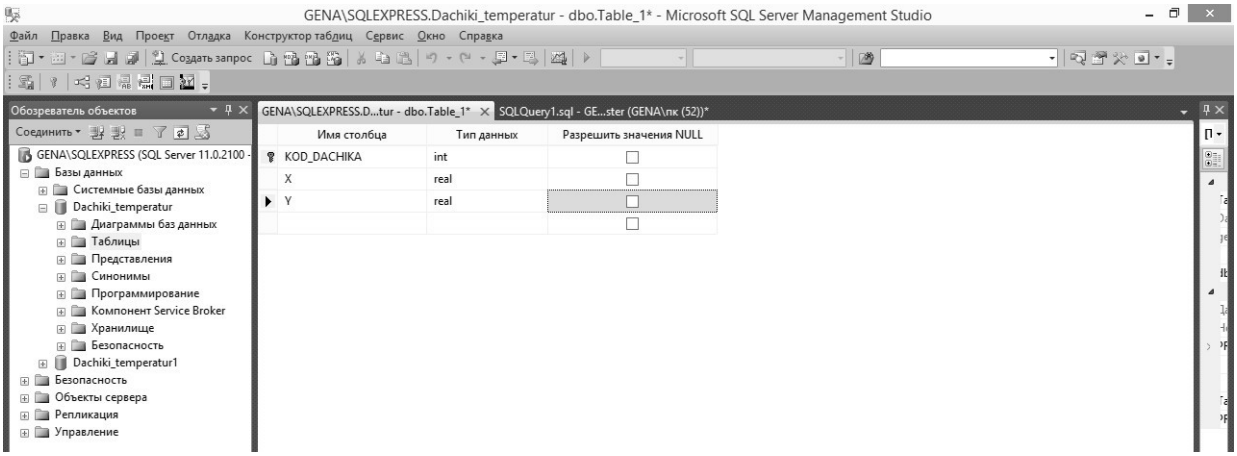


Рис. 5.20. Внесение соответствующих реквизитов полей таблицы

После обновления в *Обозревателе объектов* можно увидеть в списке таблиц созданную таблицу *Datchiki* (рис. 5.21). Префикс *dbo* в имени таблицы означает имя владельца таблицы – *database owner*.

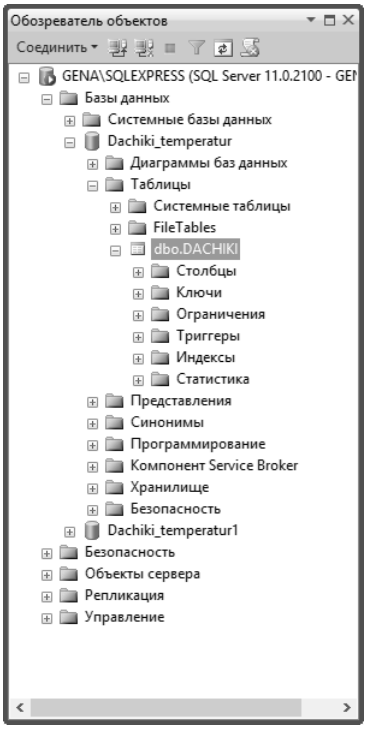


Рис. 5.21. Созданная таблица

Аналогично создается вторая таблица и связывается с первой по *первичному ключу* – полю, однозначно определяющему запись в таблице.

Далее необходимо создать соответствующие хранимые процедуры. Для этого нужно щелкнуть правой кнопкой мыши на элементе *Хранимые процедуры* и выбрать пункт *Создать хранимую процедуру…*.

Запрос SQL 1. Пример запроса SQL, в котором используется оператор *SELECT* для осуществления выборки из таблиц базы данных с целью получения необходимых полей в нужной последовательности:

```
SELECT X, Y, ID_DATCHIKA
FROM [Datchiki_temperatur].[dbo].[ DATCHIKI];
```

Запрос SQL 2. В этом примере запроса SQL символ звездочка (*) использован для вывода всех столбцов (получения всех полей отношения) таблицы *Datchiki*:

```
SELECT *
FROM Datchiki_temperatur.dbo.DATCHIKI
```

Запрос SQL 3. Инструкция *DISTINCT* используется для исключения повторяющихся записей и получения множества уникальных записей:

```
SELECT DISTINCT ID_DATCHIKA
FROM Datchiki_temperatur.dbo.DATCHIKI;
```

Запрос SQL 4. Инструкция *ORDER BY* используется для сортировки (упорядочивания) записей по значениям определенного поля. Имя поля указывается за этой инструкцией:

```
SELECT *
FROM Datchiki_temperatur.dbo.DATCHIKI
ORDER BY X;
```

Запрос SQL 5. Инструкция *ASC* используется как дополнение к инструкции *ORDER BY* и служит для определения сортировки по возрастанию. Инструкция *DESC* используется как дополнение к инструкции *ORDER BY* и служит для определения сортировки по убыванию. В случае когда ни *ASC*, ни *DESC* не указаны, подразумевается наличие *ASC*:

```
SELECT *
FROM Datchiki_temperatur.dbo.DATCHIKI
ORDER BY x DESC, y;
```

Запрос SQL 6. Для отбора необходимых записей из таблицы пользуются различными логическими выражениями, которые определяют условия отбора. Логическое выражение приводится после инструкции *WHERE*. Пример получения из таблицы всех записей, для которых значение *x* больше 40:

```
SELECT *
FROM Datchiki_temperatur.dbo.DATCHIKI
WHERE x>40;
```

Запрос SQL 7. Инструкция *BETWEEN* используется для проверки принадлежности некоторому диапазону значений. Пример запроса SQL, выводящий информацию о значениях, снятых с датчиков между 23 и 24 декабря 2016 года:

```
SELECT *
FROM Datchiki_temperatur.dbo.temperatur INNER JOIN
Datchiki_temperatur.dbo.temperatur
Datchiki_temperatur.dbo.DATCHIKI
ON Datchiki_temperatur.ID_DATCHIKA= DATCHIKI.ID_DATCHIKA
WHERE data BETWEEN '2016-12-23' And '2016-12-24'
```

5.2.1. Индивидуальные задания

Вариант 1. Создать базу данных и запрос, сортирующий по возрастанию значения температур и показывающий координаты датчиков.

Вариант 2. Создать базу данных и запрос, показывающий координаты и номера датчиков, с которых были сняты значения температур с 24.12.2016 по 25.01.2017.

Вариант 3. Создать базу данных и запрос, показывающий дату и координаты снятия температуры в диапазоне от –7 до –10 С°.

Вариант 4. Создать базу данных и запрос, показывающий минимальное и максимальное значение температуры, а также дату снятия значений температур и координаты датчиков.

5.3. Содержание отчета

1. Цель работы.
2. Краткие теоретические сведения.
3. Индивидуальные задания.
4. Код хранимой процедуры.
5. Выводы.

5.4. Контрольные вопросы

1. Каким образом можно получить доступ к базам данных SSMS?
2. С помощью каких средств можно создать таблицу в SSMS?

3. Что такое и каково назначение первичного ключа?
4. Что из себя представляют хранимые процедуры?
5. Что означает Not Null?

Рекомендуемые источники

Троелсен, Э. Язык программирования C# 6.0 и платформа .NET 4.6 / Э. Троелсен, Ф. Джепикс ; пер. с англ. – М. : Вильямс, 2016. – 1 400 с.

ЛАБОРАТОРНАЯ РАБОТА № 5 ДОСТУП И ВИЗУАЛИЗАЦИЯ ИНФОРМАЦИИ, ХРАНЯЩЕЙСЯ В БАЗЕ ДАННЫХ РЕЗУЛЬТАТОВ ИЗМЕРЕНИЙ И МОНИТОРИНГА, С ИСПОЛЬЗОВАНИЕМ WEB-ИНТЕРФЕЙСА

Цель работы: изучить методы доступа и визуализации информации, хранящейся в базе данных результатов измерений и мониторинга, с использованием Web-форм и ознакомиться с возможностями языка программирования C# при реализации соответствующих алгоритмов.

6.1. Теоретические сведения

6.1.1. Графический пользовательский интерфейс

Исходя из представленной ранее функциональности информационной системы измерений и мониторинга должен быть разработан ее Web-ориентированный графический пользовательский интерфейс, основанный на входящих в состав универсальных элементах, компонентах и объектах, предназначенных для удаленного доступа и взаимодействия как с аппаратурой, так и с серверным программным обеспечением для работы с данными в соответствии с учетными записями пользователей. Клиентская часть включает в себя программные модули для формирования графиков и диаграмм, работы с SQL-запросами и обмена данными с базой данных. Для отображения собранных данных в удобной форме должно быть создано клиентское кроссплатформенное программное обеспечение, написанное на языке программирования C#, с учетом возможности использования под управлением многих операционных систем. Его основные возможности:

– построение различных графических зависимостей температур в соответствии с запросами пользователей, а также энергетических показателей за выбранный период;

– формирование диаграмм (гистограмм), характеризующих энергоэффективность солнечного коллектора;

– отображение текущих значений параметров установки;

– экспорт полученных данных в определенном формате и так далее.

Internet-ресурс должен предоставлять пользователям удобный Web-интерфейс для формирования и просмотра графиков и данных о работе солнечного коллектора.

Клиент посылает запрос серверу, в котором содержится информация о том, что пользователь хочет получить. Сервер обрабатывает запрос, создает запрос на выборку базе данных, в которой находится вся необходимая информация о результатах работы оборудования, и отсылает эти данные обратно пользователю в табличном или графическом виде. Пользователь может

выбрать, за какой период времени необходимо посмотреть информацию о

работе солнечного коллектора. На графике можно увидеть информацию о температуре теплоносителя на выходе коллектора, а также мощность, вырабатываемую самим коллектором. Также на Web-сайте должна располагаться интерактивная схема солнечного коллектора, при наведении указателя на виртуальные датчики которой с помощью SQL-запросов выводится информация о тех характеристиках, которые снимаются в текущий момент времени. Должна также быть возможность вывести в виде гистограммы информацию о массе сбереженного условного топлива по дням (рис. 6.1).

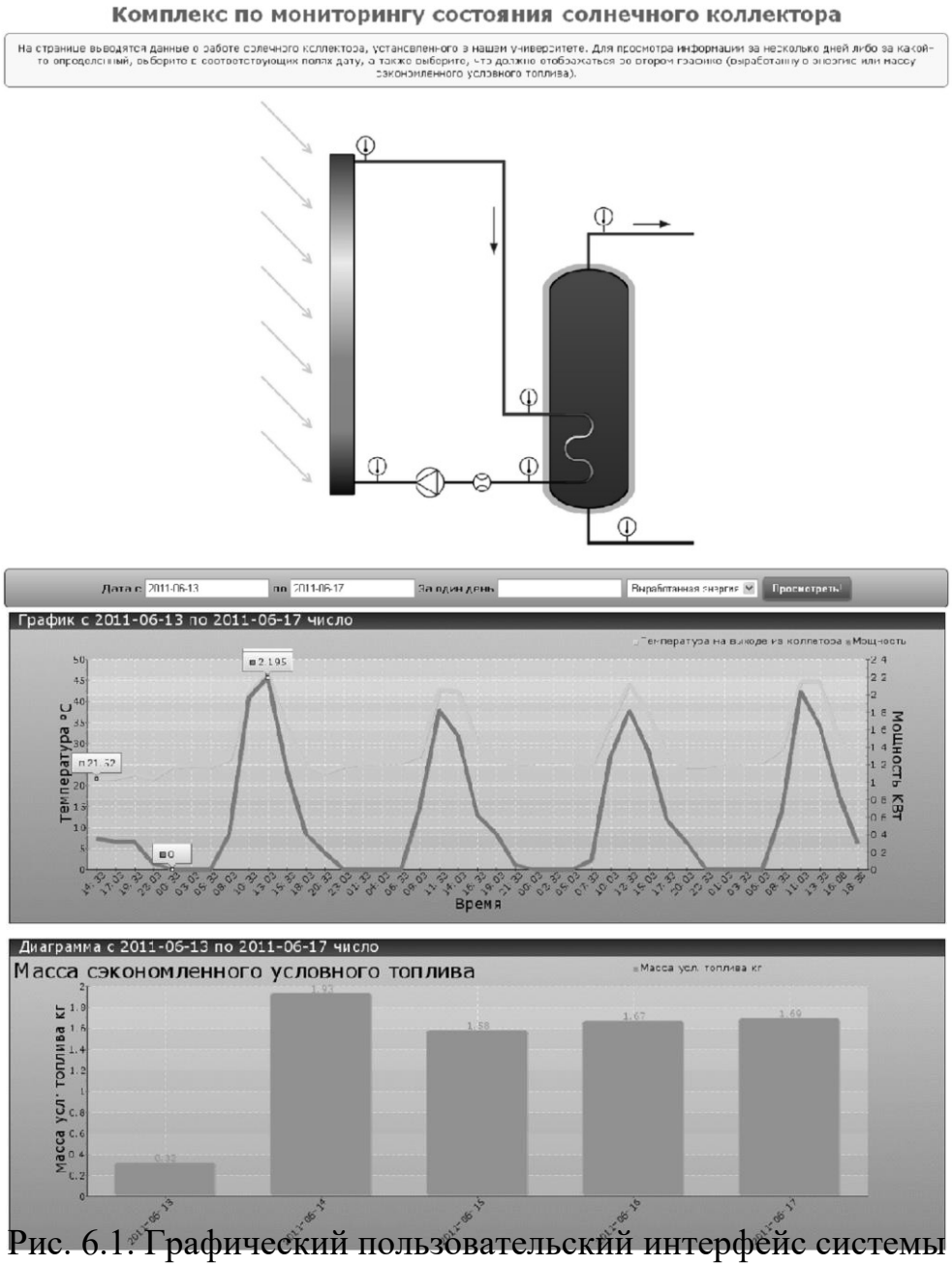


Рис. 6.1. Графический пользовательский интерфейс системы

6.1.2. Технология ASP.NET для создания Web-приложения и Web-сервера

Microsoft ASP.NET – это технология создания Web-приложений и Web-серверов. Она является составной частью платформы Microsoft .NET и развитием прежней технологии ASP.

ASP.NET основывается на единой среде выполнения *Common Language Runtime* (CLR), которая является основой всех приложений .NET. Программисты при создании приложений на ASP.NET могут писать код, используя различные языки программирования, поддерживаемые в .NET Framework, как коммерческие (Visual Basic, Visual C#, Visual C++, Visual J# и другие), так и «открытые» (Python, Perl и другие), а не только интерпретируемые языки скриптов. ASP.NET предлагает достаточно надежную модель создания Web-приложений. Например, можно разделить логику представления на HTML и бизнес-логику при помощи техники, называемой *Codebehind* (фоновый код). Для доступа к Web-приложениям клиентам нужен лишь Web-браузер.

Web-приложение – это набор взаимосвязанных файлов (HTM (HTML), ASP, ASPX, файлов изображений и тому подобное), а также связанных с ними компонентов (двоичных файлов .NET или классической COM), которые размещены на Web-сервере.

Web-сервер – это программный продукт, на котором размещаются Web-приложения и который обычно обеспечивает набор связанных с Web-приложениями служб, таких как интегрированные средства обеспечения безопасности, поддержка протокола FTP, поддержка средств передачи электронной почты и тому подобное. Web-сервер (Web-сервисы) от Microsoft называется *Internet Information Services* (IIS).

Большинство современных сред разработки Web-приложений (в том числе *Microsoft Visual Studio*) позволяют создавать Web-страницы, почти не обращаясь непосредственно к самому коду HTML с помощью специальных интегрированных средств разработки интерфейса, однако тем не менее разработчик Web-приложений должен, безусловно, знать этот язык.

Документ HTML обычно начинается с набора тегов, в которых содержится общая информация о документе: заголовок, метаданные файла и тому подобное, за которыми следует само тело документа (то есть набор текста, изображений, таблиц, гиперссылок и так далее). Теги HTML не чувствительны к регистру.

Разработку документа HTML можно начать с загрузки интегрированной среды разработки Microsoft Visual Studio и создания шаблона *Пустой веб-сайт ASP.NET* (рис. 6.2).

После этого можно добавлять в созданный проект шаблоны динамически формируемых на сервере Web-страниц ASP.NET, состоящие из файлов разметки и программной логики, содержащих коды серверной стороны.

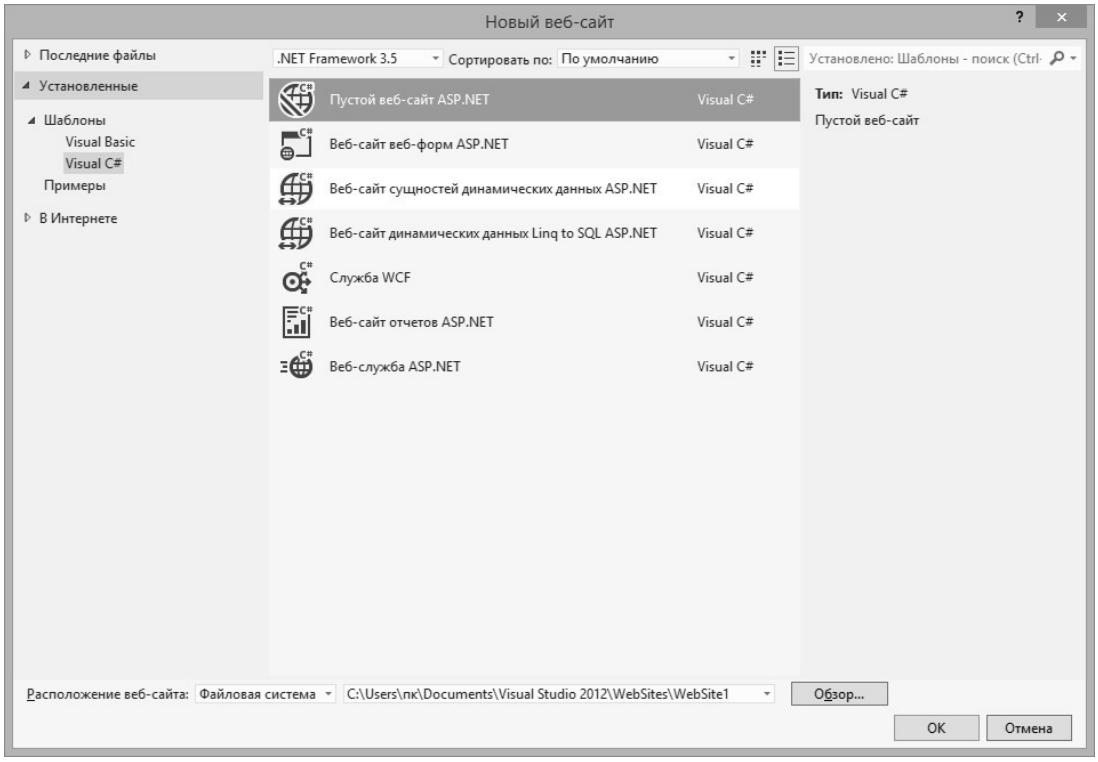


Рис. 6.2. Формирование шаблона

Графические средства, которые применяются для управления отображением всего документа, находятся в свойствах объекта Document (рис. 6.3).

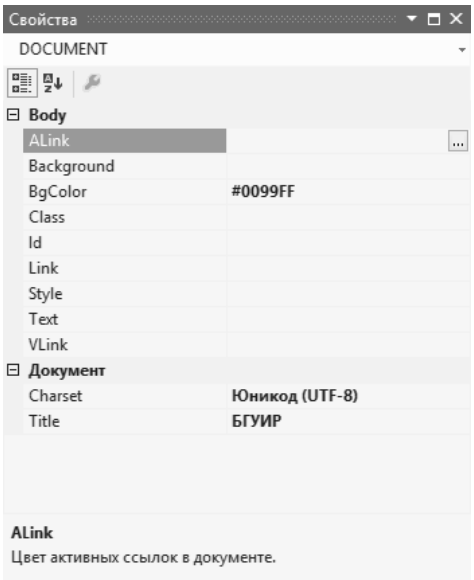


Рис. 6.3. Панель свойств объекта

В Microsoft Visual Studio также предусмотрена панель форматирования HTML. С ее помощью можно управлять представлением блоков текста, выбирая для них уровень заголовка, разметку списков, шрифт и его атрибуты и тому подобное.

Таким образом, при помощи графических средств Microsoft Visual Studio можно оформлять гипертекстовые страницы – весь необходимый для этого код HTML будет сгенерирован автоматически. Эти средства позволяют сэкономить много времени, однако иногда Web-разработчику приходится создавать код для страницы HTML вручную.

6.1.3. Разработка форм и создание пользовательского интерфейса

После внешнего оформления страницы целесообразно наделить ее новыми свойствами – возможностью принимать ввод пользователя. Для этого придется прибегнуть к помощи элементов управления HTML. Как станет очевидным далее, в среде ASP.NET предусмотрен набор элементов управления WebForm, при применении которых все необходимые теги для элементов управления HTML будут генерироваться автоматически. Однако знакомство с тегами для создания элементов управления HTML также будет нелишним. Еще раз следует подчеркнуть, что элементы управления WebForm в ASP.NET и элементы управления HTML – это разные вещи, и в процессе выполнения Web-приложения первые преобразуются во вторые.

Форма HTML – это именованная группа элементов пользовательского интерфейса HTML, используемых для ввода пользователем данных. Затем эти данные передаются на Web-сервер по протоколу HTTP. Теги для элементов пользовательского интерфейса на форме HTML помещаются между тегами <form> и </form>:

```
<form name="MainForm">
</form>
```

В этом коде создана форма и присвоено ей имя. Также можно указать идентификатор формы через свойство *Id* с передачей в него строки, например *id="MainForm"*, так как не все браузеры поддерживают свойство *Name*. С технической точки зрения использовать имя и (или) идентификатор необязательно, однако во многих ситуациях это очень удобно.

Как правило, в открывающий тег <form> помещается атрибут для действия, выполняемого этой формой. В нем содержится информация об адресе URL, на который будут передаваться данные, введенные пользователем, а также сведения о методе передачи данных. В Microsoft Visual Studio предусмотрена специальная *Панель элементов*, в которой можно выбрать различные элементы управления формой (рис. 6.4).

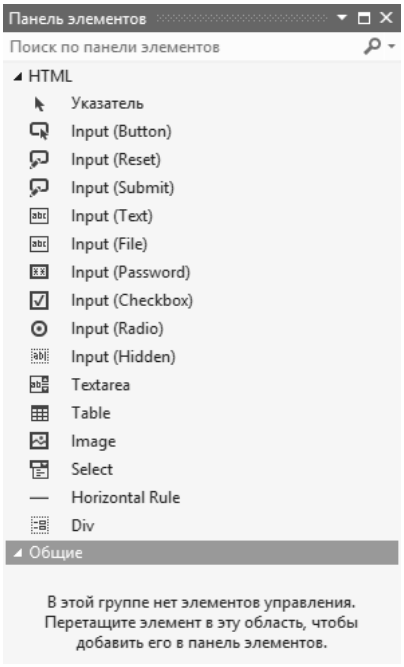


Рис. 6.4. Панель элементов

Краткий перечень наиболее часто используемых элементов представлен в табл. 6.1.

Таблица 6.1

Элементы управления HTML

Элемент управления	**Описание**
Image	Позволяет указать изображение, которое будет выведено на форме
Input (Button)	Используется для того, чтобы выполнить блок кода клиентского скрипта
Input (Checkbox) Input (Radio)	То же самое, что и аналогичные элементы управления Windows Forms, предназначенные для выбора или переключения соответственно
Input (Password)	Специальное текстовое поле, предназначенное для ввода пользователем пароля. Все введенные данные в это поле могут отображаться одинаковыми символами
Input (Reset) Input (Submit)	Специальные кнопки на форме, при нажатии которых все значения в форме принимают свой исходный вид (состояние) или производится отправка данных формы на Web-сервер соответственно
Input (Text) Textarea	Предназначены для ввода пользователем одной или нескольких строк текста

Table	Используется для формирования таблицы

В библиотеке базовых классов .NET предусмотрен набор типов .NET, которые соответствуют элементам управления HTML. Они определены в пространстве имен System.Web.UI.HtmlControls.

Если открыть автоматически сгенерированный файл ASPX, то можно найти в нем минимальный набор тегов с единственной формой:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
     <head runat="server">
          <title>Untitled Page</title>
     </head>
     <body>
          <form id="form1" runat="server">
               <div>
               </div>
          </form>
     </body>
</html>
```

В этом коде достойны внимания 2 детали:

1. Следует обратить внимание на атрибут *Runat* в открывающем теге *<form>*. Этот атрибут – один из важнейших в ASP.NET. Он означает, что данный элемент должен быть обработан средой выполнения ASP.NET, которая вернет результат Web-браузеру клиента.

2. В коде предусмотрено сразу несколько моментов, относящихся ко всей Web-странице в целом. В самом начале используется атрибут *Language*. Его значение определяет, что для создания кода HTML, возвращаемого браузеру клиента, будет использован Visual C#. Атрибут *CodeFile* определяет имя файла Visual C#, в котором содержится код этого языка и который будет использован для всех фоновых вычислений (обработок) на стороне сервера. Атрибут *Inherits* определяет имя класса, представляющего класс, определенный в *CodeFile*.

Файл *Web.config* – это текстовый файл в формате XML, который используется для определения множества параметров Web-приложения. Обычно этот файл расположен в корне виртуального каталога, соответствующего физическому, и используется для каждого подкаталога. По умолчанию в него помещается информация о компиляции, ошибках, безопасности, отладке и глобализации. Кроме того, в него могут быть помещены и другие необходимые данные. Таким образом, файл *Web.config* позволяет настраивать основные параметры Web-приложения.

Как и в классических ASP, в ASP.NET используется глобальный файл (*Global.asax*), который позволяет взаимодействовать с событиями как уровня всего приложения, так и уровня сеанса подключения. Кроме того, этот файл делает возможным совместное использование различных общих данных. Это реализуется при помощи информации, представленной классом *Global*, являющимся производным от базового класса *HttpApplicatlon*.

В некоторых отношениях класс *Global* действует в качестве промежуточного звена между внешним клиентом и элементами управления WebForm, как в классических ASP. В общем, эти события позволяют реагировать на запуск и прекращение работы как Web-приложения в целом, так и отдельных сеансов подключения.

Для более глубокого понимания механизма работы Web-приложений ASP.NET следует рассмотреть пример кода на Visual C#, внедренного в страницу ASPX. Если после создания шаблона обратиться по адресу Web-приложения, то среда выполнения ASP.NET вернет пустую страницу. Можно исправить эту ситуацию, например, изменив содержание файла Default.aspx таким образом, чтобы возвращалась информация о произведенном запросе HTTP:

```
<body>
     <span style="font-size: 14pt">
          <strong>
               Источник:
          </strong>
     </span>
     <%=Request.ServerVariables["HTTP_USER_AGENT"] %>
     <br />
     <br />
     <span style="font-size: 14pt">
          <strong>
               Приемник:
          </strong>
     </span>
     <%=this.ToString() %>
     <form id="form1" runat="server" method="post">
     </form>
</body>
```

6.2. Порядок выполнения

При создании пользовательского интерфейса первой необходимостью является обеспечение возможности страницы воспринимать ввод информации пользователя. Когда форма создана, можно приступать к добавлению в нее элементов управления. Это можно сделать при помощи графических средств Microsoft Visual Studio, а можно создать все необходимые теги вручную. Каждый элемент управления описывается атрибутом имени (имя используется при выполнении программы, чтобы определить, например, в какой элемент

управления были введены данные) и атрибутом типа (этот атрибут и определяет разновидность элемента управления). Для разных элементов управления существуют разные наборы дополнительных атрибутов, которые могут быть использованы для определения их параметров. Конечно же, эти дополнительные параметры можно также настроить при помощи окна свойств для соответствующего элемента управления в Microsoft Visual Studio.

Предполагается, что форма будет содержать, в частности, 2 текстовых поля (одно – для ввода имени пользователя, другое – специальное – для ввода пароля) и две кнопки для передачи информации на сервер и восстановления формы в исходном состоянии, если пользователь решает отменить свой ввод (рис. 6.5).

Рис. 6.5. Интерфейс главной формы

Далее необходимо реализовать подключение и настроить доступ и получение информации из базы данных с выводом результатов в соответствующие элементы управления Web-формы.

6.2.1. Настройка доступа к базам данных в SSMS

Прежде чем приступать к использованию элементов управления для работы с источниками данных, необходимо определенным образом произвести настройку доступа к базам данных в SSMS (рис. 6.6–6.8).

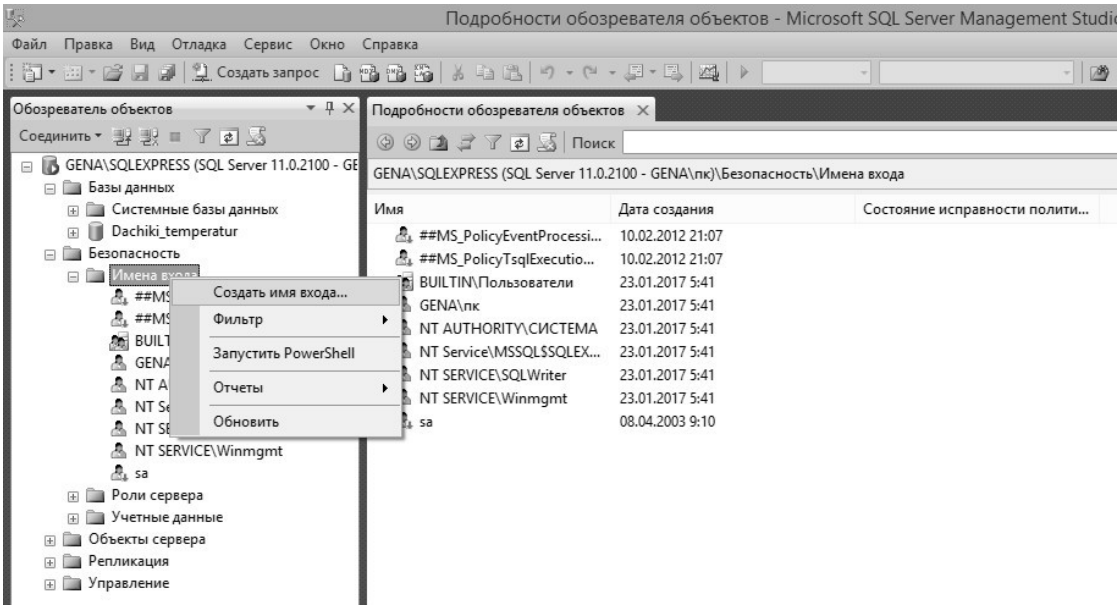


Рис. 6.6. Добавление имени новой учетной записи (сеанса)

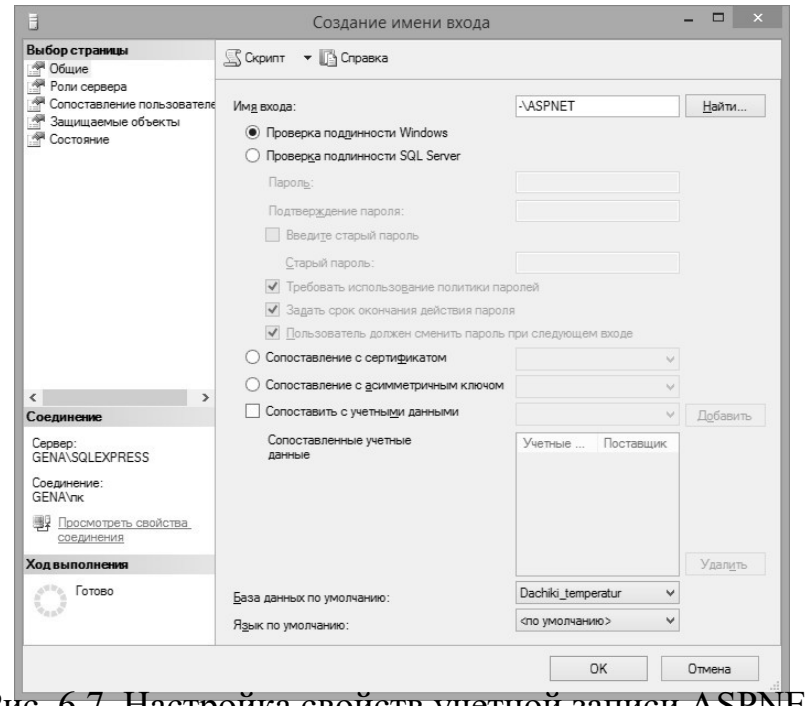


Рис. 6.7. Настройка свойств учетной записи ASPNET (указание исходной базы данных)

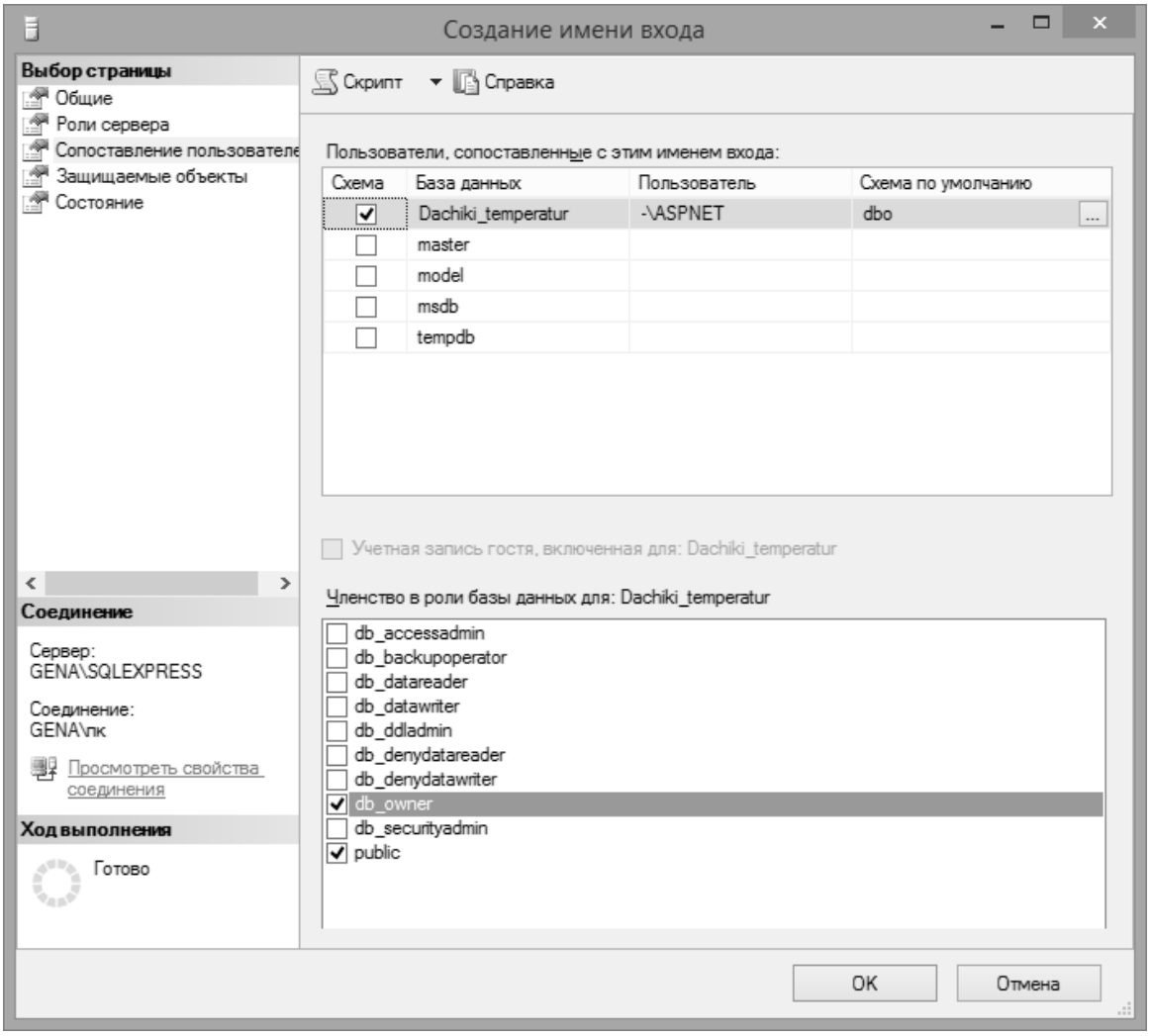


Рис. 6.8. Настройка свойств учетной записи ASPNET
(выбор ролевого членства для базы данных)

Фиксированная роль *db_owner* предоставляет все возможности манипулирования базой данных, в частности конфигурирование, сопровождение и выполнение хранимых процедур. Разрешения для этой роли включают разрешения всех остальных фиксированных ролей базы данных. В последних версиях Microsoft SQL Server члены указанной роли могут удалять базу данных.

6.2.2. Элементы управления для работы с источниками данных

В ASP.NET предусмотрено 2 элемента управления WebForm, предназначенных для отображения данных, полученных из источника (обычно в качестве источника в приложениях ASP.NET выступает объект ADO.NET *DataSet*, который, в свою очередь, может быть, например, заполнен данными с сервера баз данных). Эти элементы управления представлены в табл. 6.2.

Таблица 6.2

Элементы управления WebForm, предназначенные для работы с источниками данных

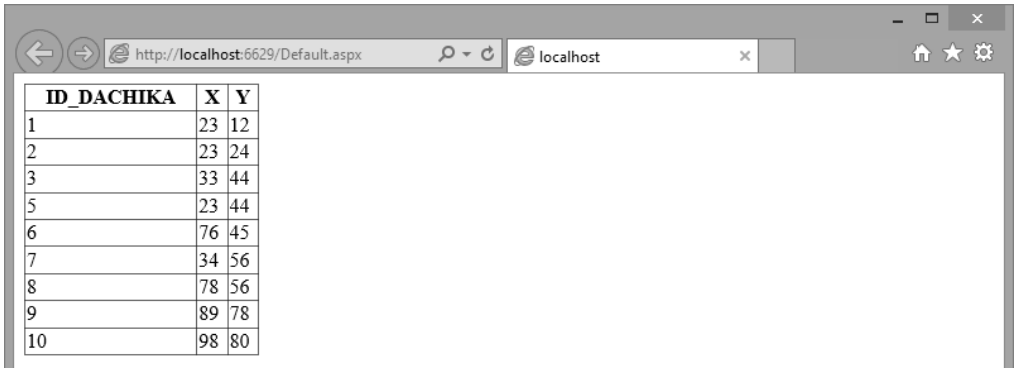
Элемент управления	**Описание**
DataGrid	Отображает содержимое объекта ADO.NET DataSet в виде таблицы
DataList	Служит для выбора значений, поступающих из источника данных

Кроме того, для работы с источниками данных можно настроить некоторые из базовых типов данных. Одна из наиболее часто встречающихся задач Web-приложений – нахождение каких-либо данных в источнике данных по запросу пользователя и возврат их в табличном формате. В классических ASP это делалось путем создания объекта ADO *Recordset* и таблицы HTML «на лету» с использованием данных из этого объекта. Тех же самых результатов гораздо проще можно достичь при помощи элемента управления WebForm – *DataGrid*.

Предполагается, что необходимо предоставить пользователю в ответ на его запрос данные из базы данных *Datchiki_temperatur*, созданной ранее. Первое, что нужно сделать, – создать обработчик для события *Load* страницы. В нем следует установить соединение с базой данных, создать и заполнить объект *DataSet* и указать его в качестве источника данных для элемента управления *DataGrid*. Соответствующий код на C# может выглядеть так:

```
protected void Page_Load(object sender, EventArgs e)
{
     if (!IsPostBack)
     {
          SqlConnection connection = new SqlConnection();
          connection.ConnectionString = @"Data Source=-
          \SQLEXPRESS; Initial Catalog=Datchiki_temperatur;
          Integrated Security=sspi";
          SqlDataAdapter dsc = new SqlDataAdapter("Select *
          from DATCHIKI", connection);
          DataSet ds = new DataSet();
          dsc.Fill(ds, "DATCHIKI");
          DataGrid1.DataSource =
          ds.Tables["DATCHIKI"].DefaultView;
          DataGrid1.DataBind();
     }
}
```

Возможный результат работы программы представлен на рис. 6.9.



ID_DACHIKA	X	Y
1	23	12
2	23	24
3	33	44
5	23	44
6	76	45
7	34	56
8	78	56
9	89	78
10	98	80

Рис. 6.9. Элемент управления DataGrid с данными, полученными из базы данных под управлением Microsoft SQL Server

Для текстовых полей Web-интерфейса может возникнуть необходимость использования элементов управления WebForm для проверки вводимых пользователем данных. Наиболее важные элементы управления этого типа представлены в табл. 6.3.

Таблица 6.3

Элементы управления для проверки данных

Элемент управления	Описание
CompareValidator	Сравнивает значение, введенное в один элемент управления, со значением, введенным во второй
CustomValidator	Позволяет определить пользовательский метод, при помощи которого будет производиться проверка
RangeValidator	Определяет, попадает ли введенное пользователем значение в определенный диапазон
RegularExpressionValidator	Проверяет введенное значение на соответствие подстановочному выражению
RequiredFieldValidator	Позволяет убедиться, что в соответствующий элемент управления действительно введено значение (он не оставлен пустым)
ValidationSummary	Отображает все ошибки, обнаруженные при проверке ввода, в виде списка, маркированного списка или обычного абзаца. Ошибки могут отображаться на Web-странице или в специальном окне оповещения Web-браузера

6.2.3. Реализация визуализации графических зависимостей

Одной из наиболее распространенных графических задач является построение диаграмм. Элемент управления *Chart* в ASP.NET предлагает широкий набор типов диаграмм и параметров конфигурации. Элемент управления *Chart* был доступен как загружаемый дополнительно в версии .NET 3.5 SP1, но теперь входит в состав .NET 4.0.

Диаграмму можно добавить, переместив элемент управления *Chart* из панели инструментов (он находится в группе *Data* (*Данные*)) на поверхность визуального конструктора или поместив дескриптор *<asp:Chart>* в код разметки. Объявление для этого примера диаграммы выглядит так:

```
<asp:Chart ID="Chart1" runat="server">
     <ChartAreas>
          <asp:ChartArea Name="ChartArea1" />
     </ChartAreas>
</asp:Chart>
```

При подключении к базе данных получится график, изображенный на рис. 6.10.

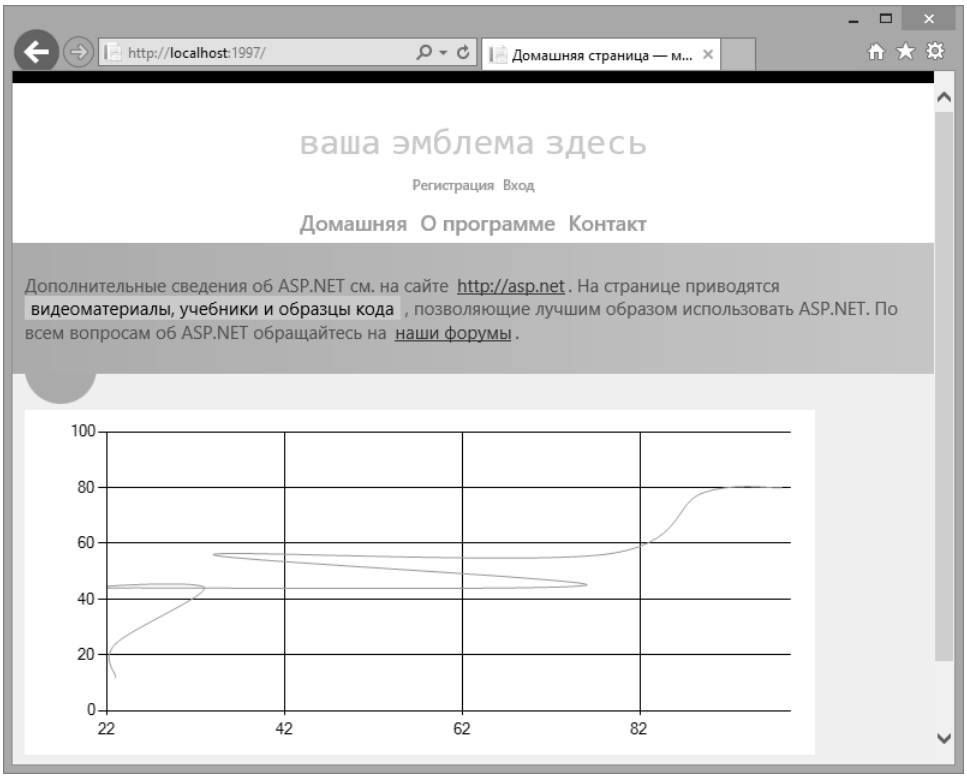


Рис. 6.10. Визуализация графической зависимости

Также для построения диаграмм можно использовать средство для формирования отчетов ASP.NET (рис. 6.11).

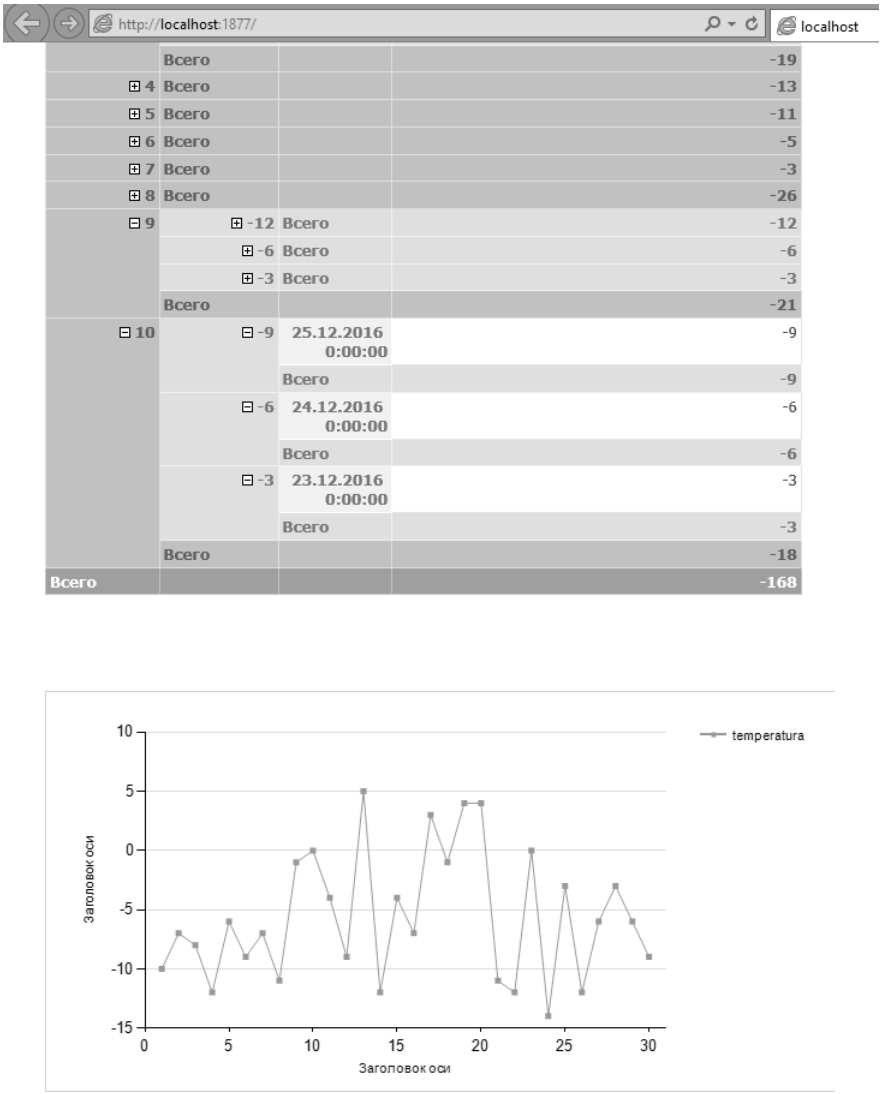


Рис. 6.11. Средство для формирования отчетов

6.2.4. Обработка событий элементов управления

События, генерируемые элементами управления WebForm, можно перехватывать и обрабатывать двумя способами:

1. Делать все непосредственно в Web-браузере клиента при помощи клиентских браузерных скриптов JavaScript. Это традиционный подход, который наиболее удобен в тех ситуациях, когда нужно выполнять форматирование на Web-странице, выводить оповещения в окне Web-браузера или осуществлять прочие взаимодействия с объектной моделью, реализованной в Web-браузерах.

2. Обрабатывать и перехватывать события элементов управления WebForm на Web-сервере. Для этого достаточно добавить обработчик события при помощи окна Properties свойств элемента управления (пиктограмма Events). Обычно такой способ наиболее удобен для выполнения операций, не связанных

с графическим интерфейсом, например для производства каких-то вычислений, редактирования таблицы с данными и тому подобное.

Таким образом, при разработке Web-приложений ASP.NET необходимо учитывать то, что каждому шаблону HTML-приложения (файлу ASPX) соответствует ASP.NET-класс, производный от System.Web.UI.Page. Работать с этим классом можно средствами привычных языков программирования .NET, например C#. Поэтому в ASP.NET можно использовать технологии объектно-ориентированного программирования, создавая код, пригодный для повторного использования. Основные свойства объекта Page (Session, Application, Request и Response) обеспечивают доступ к внутренним объектам класса, производного от Page. Элементы управления WebForm во многом аналогичны стандартным элементам управления Windows Forms, с помощью которых можно избежать трудоемкой и утомительной обязанности создавать теги HTML и клиентские скрипты вручную.

6.2.5. Индивидуальные задания

Требуется реализовать Web-сайт, состоящий из следующих Web-страниц:

1. Авторизация.
2. Доступ к данным и визуализация.
3. Выведение на Web-страницу данных из предыдущей лабораторной работы, а также фамилии и инициалов студентов.

6.3. Содержание отчета

1. Цель работы.
2. Краткие теоретические сведения.
3. Индивидуальные задания.
4. Код программы.
5. Выводы.

6.4. Контрольные вопросы

1. Каким образом происходит компиляция ASP.NET-приложения?
2. Как компилируются страницы ASPX?
3. Каким образом среда исполнения ASP.NET может отслеживать изменение страниц ASP.NET?
4. Что происходит при изменении страниц ASP.NET на Web-сервере?

Рекомендуемые источники

Троелсен, Э. Язык программирования C# 6.0 и платформа .NET 4.6 / Э. Троелсен, Ф. Джепикс ; пер. с англ. – М. : Вильямс, 2016. – 1 400 с.

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

МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ УПРАВЛЕНИЯ

Методические указания к самостоятельным работам

Направление подготовки 15.03.04 Автоматизация технологических процессов и производств

Направленность (профиль) «Информационно-управляющие системы»

Квалификация выпускника – бакалавр

Невинномысск 2021

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 «Микропроцессорные системы управления».

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, опре-

деления, 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 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.