МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Невинномысский технологический институт (филиал)

Методические указания по выполнению практических работ по дисциплине «Практикум по программированию промышленных контроллеров»

Направление подготовки 15.03.04 Автоматизация технологических процессов и производств
Квалификация выпускника – бакалавр

Невинномысск 2019

Методические указания предназначены для выполнению практических работ по дисциплине «Практикум по программированию промышленных контроллеров» для студентов направления подготовки 15.03.04 Автоматизация технологических процессов и производств и соответствуют требованиям ФГОС ВО направления подготовки бакалавров.

Составитель: старший преподаватель кафедры ИСЭА Д.В.Самойленко

Содержание

Практическое занятие №1

Цифровые выводы

Выводы платформы Arduino могут работать как входы или как выходы. Данный документ объясняет функционирование выводов в этих режимах. Также необходимо обратить внимание на то, что большинство аналоговых входов Arduino (Atmega) могут конфигурироваться и работать так же как и цифровые порты ввода/вывода.

Свойства порта вводы/вывода (pin), сконфигурированного как порт ввода

Выводы Arduino (Atmega) стандартно настроены как порты ввода, таким образом, не требуется явной декларации в функции pinMode(). Сконфигурированные порты ввода находятся в высокоимпедансном состоянии. Это означает то, что порт ввода дает слишком малую нагрузки на схему, в которую он включен. Эквивалентом внутреннему сопротивлению будет резистор 100 МОм подключенный к выводу микросхемы. Таким образом, для перевода порта ввода из одного состояния в другое требуется маленькое значение тока. Это позволяет применять выводы микросхемы для подключения емкостного датчика касания, фотодиода, аналогового датчика со схемой, похожей на RC-цепь.

С другой стороны, если к данному выводу ничего не подключено, то значения на нем будут принимать случайные величины, наводимые электрическими помехами или емкостной взаимосвязью с соседним выводом.

Подтягивающие (нагрузочные) резисторы

Если на порт ввода не поступает сигнал, то в данном случае рекомендуется задать порту известное состояние. Это делается добавлением подтягивающих резисторов 10 кОм, подключающих вход либо к +5 В (подтягивающие к питанию резисторы), либо к земле (подтягивающие к земле резисторы).

Микроконтроллер Atmega имеет программируемые встроенные подтягивающие к питанию резисторы 20 кОм. Программирование данных резисторов осуществляется следующим образом.

```
pinMode(pin, INPUT);           // назначить выводу порт ввода
digitalWrite(pin, HIGH);       // включить подтягивающий резистор
```

Подтягивающий резистор пропускает ток достаточный для того, чтобы слегка светился светодиод подключенный к выводу, работающему как порт ввода. Также легкое свечение светодиодов означает то, что при программировании вывод не был настроен как порт вывода в функции pinMode().

Подтягивающие резисторы управляются теми же регистрами (внутренние адреса памяти микроконтроллера), что управляют состояниями вывода: HIGH или LOW. Следовательно, если вывод работает как порт ввода со значением HIGH, это означает включение подтягивающего к питанию резистора, то конфигурация функцией pinMode() порта вывода на данном выводе микросхемы передаст значение HIGH. Данная процедура работает и в обратном направлении, т.е. если вывод имеет значение HIGH, то конфигурация вывода микросхемы как порта ввода функцией pinMode() включит подтягивающий к питанию резистор.

Примечание: Затруднительно использовать вывод микросхемы 13 в качестве порта ввода из-за подключенных к нему светодиода и резистора. При подключении подтягивающего к питанию резистора 20 кОм на вводе будет 1.7 В вместо 5 В, т.к. происходит падение напряжения на светодиоде и включенном последовательно резисторе. При необходимости использовать вывод микросхемы 13 как цифровой порт ввода требуется подключить между выводом и землей внешний подтягивающий резистор.

Свойства порта ввода/вывода, сконфигурированного как порт вывода

Выводы, сконфигурированные как порты вывода, находятся в низкоимпедансном состоянии. Данные выводы могут пропускать через себя достаточно большой ток. Выводы микросхемы Atmega могут быть источником (положительный) или приемником (отрицательный) тока до 40 мА для других устройств. Такого значения тока достаточно чтобы подключить светодиод (обязателен последовательно включенный резистор), датчики, но недостаточно для большинства реле, соленоидов и двигателей.

Короткие замыкания выводов Arduino или попытки подключить энергоемкие устройства могут повредить выходные транзисторы вывода или весь микроконтроллер Atmega. В большинстве случаев данные действия приведут к отключению вывода на микроконтроллере, но остальная часть схемы будет работать согласно про-

грамме. Рекомендуется к выходам платформы подключать устройства через резисторы 470 Ом или 1 кОм, если устройству не требуется больший ток для работы.

Практическое занятие №2

Аналоговые входы

Описание портов, работающих как аналоговые входы, платформы Arduino (Atmega8, Atmega168, Atmega328, или Atmega1280)

Аналого-цифровой преобразователь

Микроконтроллеры Atmega, используемые в Arduino, содержат шестиканальный аналого-цифровой преобразователь (АЦП). Разрешение преобразователя составляет 10 бит, что позволяет на выходе получать значения от 0 до 1023. Основным применением аналоговых входов большинства платформ Arduino является чтение аналоговых датчиком, но в тоже время они имеют функциональность вводов/выводов широкого применения (GPIO) (то же, что и цифровые порты ввода/вывода 0 - 13).

Таким образом, при необходимости применения дополнительных портов ввода/вывода имеется возможность сконфигурировать неиспользуемые аналоговые входы.

Цоколевка

Выводы Arduino, соответствующие аналоговым входам, имеют номера от 14 до 19. Это относится только к выводам Arduino, а не к физическим номерам выводов микроконтроллера Atmega. Аналоговые входы могут использоваться как цифровые выводы портов ввода/вывода. Например, код программы для установки вывода 0 аналогового входа на порт вывода со значением HIGH:

```
pinMode(14, OUTPUT);
digitalWrite(14, HIGH);
```

Подтягивающие резисторы

Выводы аналоговые входов имеют подтягивающие резисторы работающие как на цифровых выводах. Включение резисторов производится командой

```
digitalWrite(14, HIGH); // включить резистор на выводе аналогового входа 0
```

пока вывод работает как порт ввода.

Подключение резистора повлияет на величину сообщаемую функцией analogRead() при использовании некоторых датчиков. Большинство пользователей

использует подтягивающий резистор при применении вывода аналогового входа в его цифровом режиме.

Подробности и предостережения

Для вывода, работавшего ранее как цифровой порт вывода, команда analogRead будет работать некорректно. В этом случае рекомендуется сконфигурировать его как аналоговый вход. Аналогично, если вывод работал как цифровой порт вывода со значением HIGH, то обратная установка на ввод подключит подтягивающий резистор.

Руководство на микроконтроллер Atmega не рекомендует производить быстрое переключение между аналоговыми входами для их чтения. Это может вызвать наложение сигналов и внести искажения в аналоговую систему. Однако после работы аналогового входа в цифровом режиме может потребоваться настроить паузу между чтением функцией analogRead() других входов.

Практическое занятие №3

Широтно-импульсная модуляция

Широтно-Импульсная модуляция, сокращенно ШИМ (англ. PWM)

Пример использования аналогового выхода (ШИМ) для управления светодиодом доступен из меню File->Sketchbook->Examples->Analog программы Arduino.

Широтно-Импульсная модуляция, или ШИМ, это операция получения изменяющегося аналогового значения посредством цифровых устройств. Устройства используются для получения прямоугольных импульсов - сигнала, который постоянно переключается между максимальным и минимальным значениями. Данный сигнал моделирует напряжение между максимальным значением (5 В) и минимальным (0 В), 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 графике зеленые линии отмечают постоянные временные периоды. Длительность периода обратно пропорциональна частоте ШИМ. Т.е. если частота ШИМ составляет 500 Гц, то зеленые линии будут отмечать интервалы длительностью в 2 миллисекунды каждый. Вызов функции analogWrite() с масштабом 0 – 255 означает, что значение analogWrite(255) будет соответствовать 100% рабочему циклу (постоянное включение 5 В), а значение analogWrite(127) – 50% рабочему циклу.

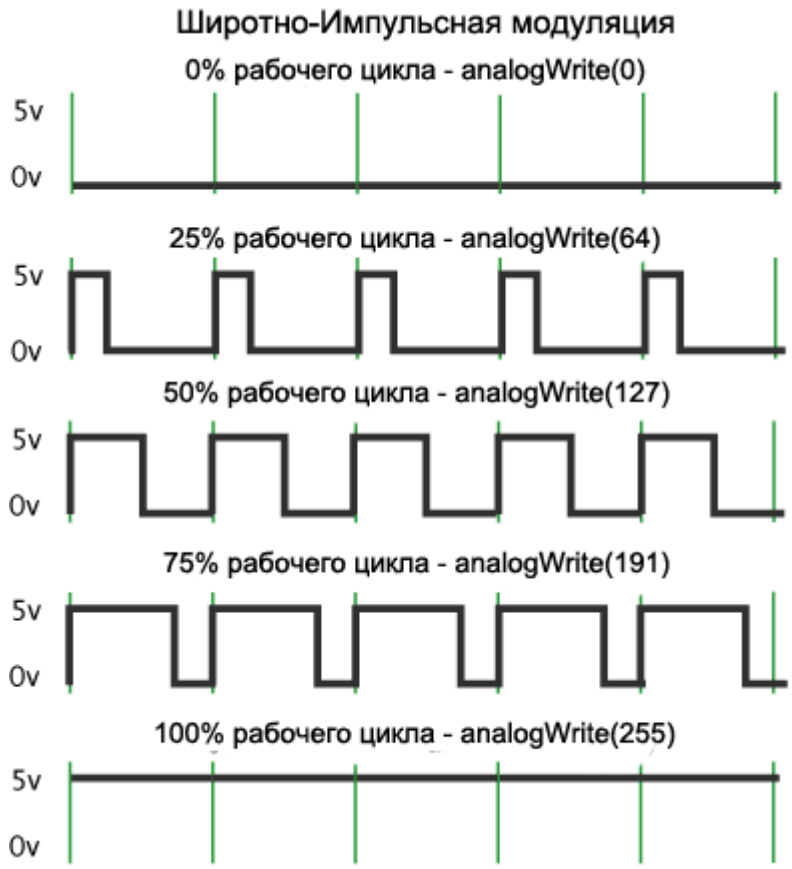

Рисунок 3.1 – Широтно-импульсная модуляция ШИМ PWM.

Для примера можно взять платформу и начать трясти ее взад и вперед. Для наших глаз данное движение превращает в светящиеся линии мигание светодиода. Наращивание или уменьшение ширины импульса на светодиоде будет увеличивать или уменьшать светящиеся линии светодиода.

Практическое занятие №4

Память в Arduino

В микроконтроллере ATmega168, используемом на платформах Arduino, существует три вида памяти:

Флеш-память: используется для хранения скетчей.

ОЗУ (Статическая оперативная память с произвольным доступом): используется для хранения и работы переменных.

EEPROM (энергонезависимая память): используется для хранения постоянной информации.

Флеш-память и EEPROM являются энергонезависимыми видами памяти (данные сохраняются при отключении питания). ОЗУ является энергозависимой памятью.

Микроконтроллер ATmega168 имеет:

16 Кб флеш-памяти (2 Кб используется для хранения загрузчика)

1024 байта ОЗУ

512 байт EEPROM

Необходимо обратить внимание на малый объем ОЗУ, т.к. большое число строк в скетче может полностью ее израсходовать. Например, следующая объявление:

```
char message[] = "I support the Cape Wind project.";
```

занимает 32 байта из общего объема ОЗУ (каждый знак занимает один байт). При наличии большого объема текста или таблиц для вывода на дисплей возможно полностью использовать допустимые 1024 байта ОЗУ.

При отсутствии свободного места в ОЗУ могут произойти сбои программы, например, она может записаться, но не работать. Для определения данного состояния требуется превратить в комментарии или укоротить строки скетча (без изменения кода). Если после этого программа работает корректно, то на ее выполнение был затрачен весь объем ОЗУ. Существует несколько путей решения данной проблемы:

При работе скетча с программой на компьютере можно перебросить часть данных или расчетов на компьютер для снижения нагрузки на Arduino.

При наличии таблиц поиска или других больших массивов можно использовать минимальный тип данных для хранения значений. Например, тип данных занимает два байта, а byte - только один (но может хранить небольшой диапазон значений).

Неизменяемые строки и данные во время работы скетча можно хранить во флеш-памяти. Для этого необходимо использовать ключ PROGMEM.

Для использования EEPROM обратитесь к библиотеке EEPROM.

Практическое занятие №5

Использование прерываний в Arduino

Часто при работе с проектами на микроконтроллерах требуется запускать фоновую функцию через равные промежутки времени. Это часто реализуется установкой аппаратного таймера для выработки прерывания. Это прерывание запускает программу обработки прерываний (Interrupt Service Routine, ISR) для управления периодическим прерыванием. В настоящей статье я описываю установку 8-битного таймера 2 для выработки прерываний на микроконтроллере ATMega168 Arduino. Я пройдусь по этапам, требуемым для установки программы обработки прерываний и внутри нее самой.

Arduino подразумевает процессор ATMega168. Этот микроконтроллер имеет несколько систем ввода-вывода, которые доступны каждому пользователю Arduino, поскольку библиотека Arduino облегчает их использование. К примеру, цифровой ввод-вывод, ШИМ, аналого-цифровые входы и последовательный порт. ATMega168 также имеет три внутренних аппаратных таймера. Хотя библиотека Arduino позволяет использовать некоторые свойства таймеров, нельзя напрямую использовать таймер для выработки периодических прерываний.

Как подсказывает название, прерывания – это сигналы, прерывающие нормальное течение программы. Прерывания обычно используются для аппаратных устройств, требующих немедленной реакции на появление событий. Например, система последовательного порта или UART (универсальный асинхронный приемопередатчик) микроконтроллера должен быть обслужен при получении нового символа. Если этого не сделать быстро, новый символ может быть потерян.

При поступлении нового символа UART генерирует прерывание. Микроконтроллер останавливает выполнение основной программы (вашего приложения) и перескакивает на программу обработки прерываний (ISR), предназначенную для данного прерывания. В данном случае это прерывание по полученному символу. Эта ISR захватывает новый символ из UART, помещает в буфер, затем очищает прерывание и выполняет возврат. Когда ISR выполняет возврат, микроконтроллер возвращается в основную программу и продолжает её с точки вызова. Все это про-

исходит в фоновом режиме и не влияет напрямую на основной код вашего приложения.

Если у вас запускается много прерываний или прерывания генерирует быстродействующий таймер, ваша основная программа будет выполняться медленнее, так как микроконтроллер распределяет свое машинное время между основной программой и всеми функциями обработки прерываний.

Вы можете задуматься, почему бы не просто проверять новый символ время от времени, вместо использования такого сложного процесса прерывания. Давайте вычислим пример, чтобы увидеть, насколько важны процессы прерывания. Скажем, у вас есть последовательный порт со скоростью передачи данных 9600 бод. Это означает, что каждый бит символа посылается с частотой 9600 Гц или около 10 кГц. На каждый бит уходит 100 мкс. Около 10 бит требуется, чтобы послать один символ, так что мы получаем один полный символ каждую миллисекунду или около того. Если наш UART буферизован, мы должны извлечь последний символ до завершения приема следующего, это дает нам на всю работу 1 мс. Если наш UART не буферизован, мы должны избавиться от символа за 1 бит или 1 мкс. Рассмотрим для начала буферизованный пример.

Мы должны проверять получение байта быстрее, чем каждую миллисекунду, чтобы предотвратить потерю данных. Применительно к Arduino это означает, что наша функция цикла должна обращаться для чтения статуса UART и возможно, байта данных, 1000 раз в секунду. Это легко выполнимо, но сильно усложнит код, который вам нужно написать. До тех пор, пока ваша функция цикла не требует больше 1 мс до завершения, вам это может сойти с рук. Но представьте, что вам нужно обслуживать несколько устройств ввода-вывода, или что необходимо работать на гораздо большей скорости передачи. Видите, какие неприятности это вскоре может принести.

С прерываниями вам не нужно отслеживать поступление символа. Аппаратура подает сигнал с помощью прерывания, и процессор быстро вызовет ISR, чтобы вовремя захватить символ. Вместо выделения огромной доли процессорного времени на проверку статуса UART, вы никогда не должны проверять статус, вы просто устанавливаете аппаратное прерывание и выполняете необходимые действия в ISR.

Ваша главная программа напрямую не затрагивается, и от аппаратного устройства не требуется особых возможностей.

Прерывание по таймеру

В настоящей статье я сосредоточусь на использовании программного таймера 2 для периодических прерываний. Исходная идея состояла в использовании этого таймера для генерации частоты биений в звуковых проектах Arduino. Чтобы выводить тон или частоту нам нужно переключать порт ввода-вывода на согласованной частоте. Это можно делать с использованием циклов задержки. Это просто, но означает, что наш процессор будет занят, ничего не выполняя, но ожидая точного времени переключения вывода. С использованием прерывания по таймеру мы можем заняться другими делами, а вывод пусть переключает ISR, когда таймер подаст сигнал, что время пришло.

Нам нужно только установить таймер, чтобы подавал сигнал с прерыванием в нужное время. Вместо прокрутки бесполезного цикла для задержки по времени, наша главная программа может делать что-то другое, например, контролировать датчик движения или управлять электроприводом. Что бы ни требовалось нашему проекту, больше нам не нужно процессорное время для получения задержек.

Практическое занятие №6

Переменные

Переменная – это место хранения данных. Она имеет имя, значение и тип. Например, данное объявление (называется декларацией):

int pin = 13;

создает переменную с именем pin, значением 13 и типом int. Затем в программе имеется возможность обратиться к данной переменной через имя с целью работы с ее значением. Например, в утверждении:

pinMode(pin, OUTPUT);

имеется значение вывода (13), которое будет передаваться в функцию pinMode(). В данном случае нет необходимости использовать переменную. Утверждение может работать и в таком виде:

pinMode(13, OUTPUT);

Преимущество переменной заключается в том, что необходимо определить значение вывода однажды и потом использовать его многократно. В последствии при изменении вывода 13 на 12 достаточно будет поменять только одну строку в программном коде. Также можно использовать специальные имена для подчеркивания значения переменной (напр., программа, управляющая светодиодом RGB, может содержать переменные redPin, greenPin и bluePin).

Переменные имеют другие преимущества перед такими значениями как число. Имеется возможность изменить значение переменной, используя присвоение. Например:

pin = 12;

изменит значение переменной на число 12. В данном примере не определяется тип переменной, т.к. он не меняется операцией присвоения. Имя переменной постоянно связано с типом, меняется только значение. [1] Перед присвоением значения необходимо декларировать переменную. Присвоение значения переменной без ее декларации вызовет следующее сообщение: error: pin was not declared in this scope".

При присвоении одной переменной другой происходит копирование значения первой переменной во вторую. Изменение значение одной переменной не влияет на другую. Например, после записи:

```
int pin = 13;
int pin2 = pin;
pin = 12;
```

только pin имеет значение 12, а pin2 еще равен 13.

Что означает слово "scope" в сообщение об ошибке, приведенной выше? Оно относится к части программы, в которой переменная может использоваться - области видимости. Область видимости определяется местом ее декларации. Например, имеется возможность использовать переменную во всей программе, если задекларировать ее в начале программного кода. Такие переменные называются глобальными. Например:

```
int pin = 13;
void setup()
{
  pinMode(pin, OUTPUT);
}
void loop()
{
  digitalWrite(pin, HIGH);
}
```

Из примера видно, что pin используется в обеих функциях setup() и loop(). Обе функции ссылаются на одну переменную, таким образом, изменение ее значения в одной функции повлияет на значение в другой:

```
int pin = 13;
void setup()
{
  pin = 12;
  pinMode(pin, OUTPUT);
}
```

```
void loop()
{
  digitalWrite(pin, HIGH);
}
```

Функции digitalWrite(), вызываемой из loop(), будет передано значение 12, т.к. оно было присвоено переменной в функции setup().

Если переменная используется только один раз в функции, то ее декларируют в данной части программного кода, ограниченной скобками функции. Например:

```
void setup()
{
  int pin = 13;
  pinMode(pin, OUTPUT);
  digitalWrite(pin, HIGH);
}
```

В данном примере переменная может использоваться только внутри функции setup(). При написании данного кода:

```
void loop()
{
  digitalWrite(pin, LOW); // wrong: pin is not in scope here.
}
```

будет выведено сообщение: «error: 'pin' was not declared in this scope». Данное сообщение будет выводиться, даже если вы задекларировали переменную где-то в программе, но пытаетесь ее использовать вне области видимости.

Почему не сделать все переменные глобальными? Если неизвестно где будет еще использоваться переменная, то почему ее надо ограничивать одной функцией? Когда переменная ограничена легче найти источник ее изменения. Если переменная глобальная, то ее значение может быть изменено в любом месте программного кода, что означает необходимость проследить ее изменение по всей программе. Например, когда переменная имеет некорректное значение, то гораздо легче найти причину если область видимости ограничена.

Практическое занятие №7

Функции

Разбиение на сегменты кода функциями позволяет создавать части кода, которые выполняют определенные задания. После выполнения происходит возврат в место, откуда была вызвана функция. Причиной создания функции является необходимость выполнять одинаковое действие несколько раз.

Для программистов, работающих с BASIC, функции в Arduino позволяют использовать подпрограммы (GOSUB в BASIC).

Разделения кода на функции имеет ряд преимуществ:

Функции позволяют организовать программу. Очень часто помогают заранее составить концепцию программы.

Функции кодируют одно действие в одном месте программы. Далее необходимо только отладить код функции.

Функции сокращают шансы на появление ошибки при необходимости изменения кода.

Функции сокращают текст скетчей и делают его компактным, т.к. некоторые секции используются много раз.

Функции облегчают использование кода в других программах делая его модульным. В этом случае функции обладают еще одним небольшим преимуществом, делая код программы легким для чтения.

Существуют две обязательные функции в скетчах Arduino setup() и loop(). Другие функции должны создаваться за скобками этих функций. В следующем примере будет создана простая функция умножения двух чисел.

Пример

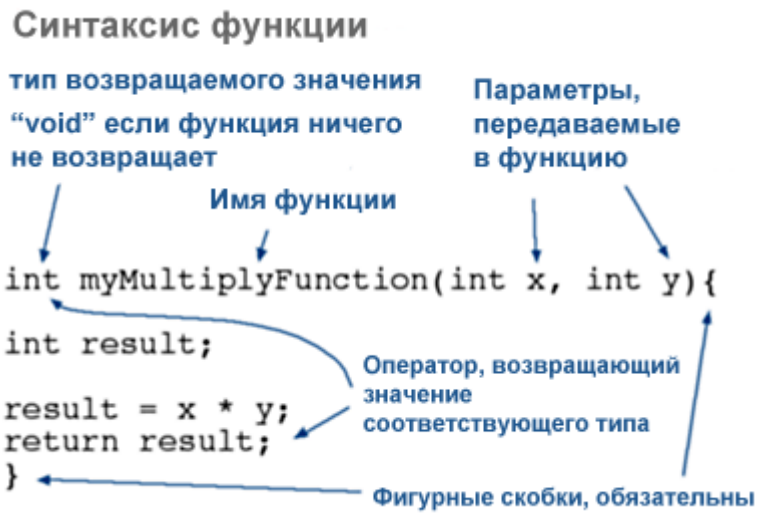

Рисунок 7.1 – Синтаксис функции Arduino

Для вызова функции умножения ей передаются параметры данных:

```
void loop(){
  int i = 2;
  int j = 3;
  int k;
  k = myMultiplyFunction(i, j); // k содержит 6
}
```

Созданную функцию необходимо задекларировать вне скобок любой другой функции, таким образом "myMultiplyFunction()" может стоять выше или ниже функции "loop()".

Весь скетч будет выглядеть следующим образом:

```
void setup(){
  Serial.begin(9600);
}
void loop(){
  int i = 2;
  int j = 3;
  int k;
  k = myMultiplyFunction(i, j); // k содержит 6
  Serial.println(k);
```

```
  delay(500);
}
int myMultiplyFunction(int x, int y){
  int result;
  result = x * y;
  return result;
}
```

Следующая функция будет считывать данные с датчика функцией analogRead() и затем рассчитывать среднее арифметическое. Затем созданная функция будет масштабировать данные по 8 битам (0-255) и инвертировать их. // датчик подключен к выводу 0

```
int ReadSens_and_Condition(){
  int i;
  int sval;
  for (i = 0; i < 5; i++){
    sval = sval + analogRead(0);    // сенсор на аналоговом входе 0
  }
  sval = sval / 5;    // среднее
  sval = sval / 4;    // масштабирование по 8 битам (0 - 255)
  sval = 255 - sval;  // инвертирование выходного значения
  return sval;
}
```

Вызов функции осуществляется присвоением ее переменной.

```
int sens;
sens = ReadSens_and_Condition();
```