Freeduino/Arduino программируется на специальном языке программирования – он основан на C/C ++, и позволяет использовать любые его функции. Строго говоря, отдельного языка Arduino не существует, как и не существует компилятора Arduino – написанные программы преобразуются (с минимальными изменениям) в программу на языке C/C++, и затем компилируются компилятором AVR-GCC. Так что фактически, используется специализированный для микроконтроллеров AVR вариант C/C++.

Разница заключается в том, что Вы получаете простую среду разработки, и набор базовых библиотек, упрощающих доступ к находящейся «на борту» микроконтроллера периферии.

Согласитесь, очень удобно начать работу с последовательным портом на скорости 9600 бит в секунду, сделав вызов одной строчкой:

```
  Serial.begin(9600);
```

А при использовании «голого» C/C++ Вам бы пришлось разбираться с документацией на микроконтроллер, и вызывать нечто подобное:

```
  UBRR0H = ((F_CPU / 16 + 9600 / 2) / 9600 - 1) >> 8;
  UBRR0L = ((F_CPU / 16 + 9600 / 2) / 9600 - 1);
  sbi(UCSR0B, RXEN0);
  sbi(UCSR0B, TXEN0);
  sbi(UCSR0B, RXCIE0);
```

Здесь кратко рассмотрены основные функции и особенности программирования Arduino. Если Вы не знакомы с синтаксисом языков C/C++, советуем обратиться к любой литературе по данному вопросу, либо Internet-источникам.

С другой стороны, все представленные примеры очень просты, и скорее всего у Вас не возникнет трудностей с пониманием исходных текстов и написанием собственных программ даже без чтения дополнительной литературы.

Более полная документация (на английском языке) представлена на официальном сайте проекта – http://www.arduino.cc . Там же есть форум, ссылки на дополнительные библиотеки и их описание.

По аналогии с описанием на официальном сайте проекта Arduino, под «портом» понимается контакт микроконтроллера, выведенный на разъем под соответствующим номером. Кроме того, существует порт последовательной передачи данных (COM-порт).

^^^

Структура программы

В своей программе Вы должны объявить две основных функции: setup() и loop().

Функция setup() вызывается один раз, после каждого включения питания или сброса платы Freeduino. Используйте её, чтобы инициализировать переменные, установить режимы работы цифровых портов и т.д.

Функция loop() последовательно раз за разом исполняет команды, которые описаны в ее теле. Т.е. после завершения функции снова произойдет ее вызов.

Разберем простой пример:

```
void setup()          // начальные установки
{
  beginSerial(9600);  // установка скорости работы серийного порта на 9600 бит/сек
  pinMode(3, INPUT);  // установка 3-его порта на ввод данных
}

// Программа проверяет 3-ий порт на наличие на нём сигнала и посылает ответ в
// виде текстового сообщения на последовательный порт компьютера
void loop() // тело программы
{
  if (digitalRead(3) == HIGH) // условие на опрос 3го порта
    serialWrite('H');         // отправка сообщения в виде буквы «H» на COM-порт
  else
    serialWrite('L');         // отправка сообщения в виде буквы «L» на COM-порт
  delay(1000);                // задержка 1 сек.
}
```

^^^

Константы

Константы – предопределенные значения. Они используются, чтобы делать программы более легкими для чтения. Мы классифицируем константы в группах.

Уровни сигналов порта HIGH и LOW

При чтении или записи к цифровому порту применимо только два возможных значения – порт может быть установлен как HIGH (высокий уровень) или LOW (низкий уровень).

Уровень HIGH соответствует 5 вольтам на выходе. При чтении значения на цифровом порте, начиная с 3 вольт и выше, микропроцессор воспримет это напряжение как HIGH. Эта константа представлена целым числом 1.

Уровень LOW соответствует 0 вольтам на выходе порта. При чтении значения на цифровом порте, начиная с 2 вольт и меньше, микропроцессор воспримет это напряжение как LOW. Эта константа представлена целым числом 0.

Таким образом, оба следующих вызова будут эквивалентны:

```
digitalWrite(13, HIGH);   // можно так,
digitalWrite(13, 1);      // а можно и так
```

Считается, что первый вариант более нагляден.

Настройка цифровых портов на ввод (INPUT) и вывод (OUTPUT) сигналов

Цифровые порты могут использоваться на ввод или вывод сигналов.

Изменение порта с ввода на вывод производится при помощи функции pinMode().

Порты, сконфигурированные на ввод сигналов, имеют большое входное сопротивление, что позволяет подключать к ним источник сигнала, и порт не будет потреблять большой ток.

Порты, сконфигурированные на вывод сигналов, имеют малое выходное сопротивление. Это означает, что такие порты могут обеспечивать подключенные к ним элементы электроэнергией. В этом состоянии порты поддерживают положительное или отрицательное направление тока до 40 мА (миллиампер) на другие устройства или схемы. Это позволяет подключить к ним какую-либо нагрузку, например светодиод (через резистор, ограничивающий ток). Порты, сконфигурированные как выводы, могут быть повреждены, если их замкнуть накоротко на «землю» (общая шина питания), на источник питания +5 В, или подсоединить к мощной нагрузке с малым сопротивлением.

Пример:

```
pinMode(13, OUTPUT);  //13й вывод будет выходом
pinMode(12, INPUT);   //а 12й - входом
```

^^^

Специфичные для Freeduino/Arduino функции и объекты

Цифровой ввод/вывод

pinMode

Вызов:

pinMode (порт, режим);

Описание:

Конфигурирует указанный порт на ввод или вывод сигнала.

Параметры:

порт – номер порта, режим которого Вы желает установить (значение целого типа от 0 до 13).

режим – либо INPUT (ввод) либо OUTPUT (вывод).

Пример:

```
pinMode(13, OUTPUT);  //13й вывод будет выходом
pinMode(12, INPUT);   //а 12й - входом
```

Примечание:

Аналоговые входы могут использоваться как цифровые входы/выходы, при обращении к ним по номерам с 14 (аналоговый вход 0) по 19 (аналоговый вход 5)

digitalWrite

Вызов:

digitalWrite(порт, значение);

Описание:

Устанавливает высокий (HIGH) или низкий (LOW) уровень напряжения на указанном порте.

Параметры:

порт: номер порта

значение: HIGH или LOW

Пример:

```
digitalWrite(13, HIGH);  // выставляем 13й вывод в «высокое» состояние
```

digitalRead

Вызов:

value = digitalRead (порт);

Описание:

Считывает значение на указанном порту

Параметры:

порт: номер опрашиваемого порта

Возвращаемое значение: возвращает текущее значение на порту (HIGH или LOW) типа int

Пример:

```
  int val;
  val = digitalRead(12);  // опрашиваем 12й вывод
```

Примечание:

Если к считываемому порту ничего не подключено, то функция digitalRead () может беспорядочно возвращать значения HIGH или LOW.

Аналоговый ввод/вывод сигнала

analogRead

Вызов:

value = analogRead(порт);

Описание:

Считывает значение с указанного аналогового порта. Freeduino содержит 6 каналов, аналого-цифрового преобразователя на 10 битов каждый. Это означает, что входное напряжения от 0 до 5В преобразовывается в целочисленное значение от 0 до 1023. Разрешающая способность считывания составляет: 5 В/1024 значений = 0,004883 В/значение (4,883 мВ). Требуется приблизительно 100 нС (0.0001 С), чтобы считать значение аналогового ввода, так что максимальная скорость считывания - приблизительно 10000 раз в секунду.

Параметры:

порт: номер опрашиваемого аналогового входа

Возвращаемое значение: возвращает число типа int в диапазоне от 0 до 1023, считанное с указанного порта.

Пример:

```
  int val;
  val = analogRead(0);  // считываем значение на 0м аналоговом входе
```

Примечание:

Аналоговые порты по умолчанию определенны на ввод сигнала и в отличие от цифровых портов их не требуется конфигурировать с помощью вызова функции pinMode.

analogWrite

Вызов:

analogWrite(порт, значение);

Описание:

Выводит на порт аналоговое значение. Эта функция работает на: 3, 5, 6, 9, 10, и 11 цифровых портах Freeduino.

Может применяться для изменения яркости светодиода, для управления двигателем и т.д. После вызова функции analogWrite, соответствующий порт начинает работать в режиме широтно-импульсного модулирования напряжения до тех пор, пока не будет следующего вызова функции analogWrite (или функций digitalRead / digitalWrite на том же самом порте).

Параметры:

порт: номер опрашиваемого аналогового входа

значение: целочисленное между 0 и 255. Значение 0 генерирует 0 В на указанном порте; значение 255 генерирует +5 В на указанном порте. Для значений между 0 и 255, порт начинает быстро чередовать уровень напряжения 0 и +5 В - чем выше значение, тем, более часто порт генерирует уровень HIGH (5 В).

Пример:

```
  analogWrite(9, 128);// устанавливаем на 9 контакте значение эквивалентное 2,5В
```

Примечание:

Нет необходимости вызвать функцию pinMode, чтобы установить порт на вывод сигналов перед вызовом функции analogWrite.

Частота генерирования сигнала – приблизительно 490 Гц.

Работа со временем

millis

Вызов:

time = millis();

Описание:

Возвращает число миллисекунд, с момента исполнения Freeduino текущей программы. Счетчик переполнится и обнулится приблизительно через 9 часов.

Возвращаемое значение: возвращает значение типа unsigned long

Пример:

```
unsigned long time; // объявление переменной time типа unsigned long
time = millis(); // передача количества миллисекунд
```

delay

Вызов:

delay(время_мс);

Описание:

Приостанавливает программу на заданное число миллисекунд.

Параметры:

время_мс – время задержки программы в миллисекундах

Пример:

```
delay(1000);  //пауза 1 секунда
```

delayMicroseconds

Вызов:

delayMicroseconds(время_мкс);

Описание:

Приостанавливает программу на заданное число микросекунд.

Параметры:

время_мкс – время задержки программы в микросекундах

Пример:

```
delayMicroseconds(500);  //пауза 500 микросекунд
```

pulseIn

Вызов:

pulseIn(порт, значение);

Описание:

Считывает импульс (высокий или низкий) с цифрового порта и возвращает продолжительность импульса в микросекундах.

Например, если параметр «значение» при вызове функции установлен в HIGH, то pulseIn() ожидает, когда на порт поступит высокий уровень сигнала. С момента его поступления начинается отсчет времени до тех пор, пока на порт не поступит низкий уровень сигнала. Функция возвращает длину импульса (высокого уровня) в микросекундах. Работает с импульсами от 10 микросекунд до 3 минут. Обратите внимание, что эта функция не будет возвращать результат, пока импульс не будет обнаружен.

Параметры:

порт: номер порта, с которого считываем импульс

значение: тип импульса HIGH или LOW

Возвращаемое значение: возвращает длительность импульса в микросекундах (тип int)

Пример:

```
int duration; // объявление переменной duration типа int
duration = pulseIn(pin, HIGH); // измеряем длительность импульса
```

Последовательная передача данных

Freeduino имеет встроенный контроллер для последовательной передачи данных, который может использоваться как для связи между Freeduino/Arduino устройствами, так и для связи с компьютером. На компьютере соответствующее соединение представлено USB COM-портом.

Связь происходит по цифровым портам 0 и 1, и поэтому Вы не сможете использовать их для цифрового ввода/вывода если используете функции последовательной передачи данных.

Serial.begin

Вызов:

Serial.begin(скорость_передачи);

Описание:

Устанавливает скорость передачи информации COM порта битах в секунду для последовательной передачи данных. Для того чтобы поддерживать связь с компьютером, используйте одну из этих нормированных скоростей: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, или 115200. Также Вы можете определить другие скорости при связи с другим микроконтроллером по портам 0 и 1.

Параметры:

скорость_передачи: скорость потока данных в битах в секунду.

Пример:

```
Serial.begin(9600); //устанавливаем скорость 9600 бит/сек
```

Serial.available

Вызов:

count = Serial.available();

Описание:

Принимаемые по последовательному порту байты попадают в буфер микроконтроллера, откуда Ваша программа может их считать. Функция возвращает количество накопленных в буфере байт. Последовательный буфер может хранить до 128 байт.

Возвращаемое значение:

Возвращает значение типа int – количество байт, доступных для чтения, в последовательном буфере, или 0, если ничего не доступно.

Пример:

```
if (Serial.available() > 0) { // Если в буфере есть данные
  // здесь должен быть прием и обработка данных
}
```

Serial.read

Вызов:

char = Serial.read();

Описание:

Считывает следующий байт из буфера последовательного порта.

Возвращаемое значение:

Первый доступный байт входящих данных с последовательного порта, или -1 если нет входящих данных.

Пример:

```
incomingByte = Serial.read(); // читаем байт
```

Serial.flush

Вызов:

Serial.flush();

Описание:

Очищает входной буфер последовательного порта. Находящиеся в буфере данные теряются, и дальнейшие вызовы Serial.read() или Serial.available() будут иметь смысл для данных, полученных после вызова Serial.flush().

Пример:

```
Serial.flush(); // Очищаем буфер - начинаем прием данных «с чистого листа»
```

Serial.print

Описание:

Вывод данных на последовательный порт.

Параметры:

Функция имеет несколько форм вызова в зависимости от типа и формата выводимых данных.

Serial.print(b, DEC) выводит ASCII-строку - десятичное представление числа b.

```
int b = 79;
Serial.print(b, DEC); //выдаст в порт строку «79»
```

Serial.print(b, HEX) выводит ASCII-строку - шестнадцатиричное представление числа b.

```
int b = 79;
Serial.print(b, HEX); //выдаст в порт строку «4F»
```

Serial.print(b, OCT) выводит ASCII-строку - восьмеричное представление числа b.

```
int b = 79;
Serial.print(b, OCT); //выдаст в порт строку «117»
```

Serial.print(b, BIN) выводит ASCII-строку - двоичное представление числа b.

```
int b = 79;
Serial.print(b, BIN); //выдаст в порт строку «1001111»
```

Serial.print(b, BYTE) выводит младший байт числа b.

```
int b = 79;
Serial.print(b, BYTE); //выведет число 79 (один байт). В мониторе
                       //последовательного порта получим символ «O» - его
                       //код равен 79
```

Serial.print(str) если str – строка или массив символов, побайтно передает str на COM-порт.

```
char bytes[3] = {79, 80, 81};    //массив из 3 байт со значениями 79,80,81
Serial.print("Here our bytes:"); //выводит строку «Here our bytes:»
Serial.print(bytes);             //выводит 3 символа с кодами 79,80,81 –
                                 //это символы «OPQ»
```

Serial.print(b) если b имеет тип byte или char, выводит в порт само число b.

```
char b = 79;
Serial.print(b); //выдаст в порт символ «O»
```

Serial.print(b) если b имеет целый тип, выводит в порт десятичное представление числа b.

```
int b = 79;
Serial.print(b); //выдаст в порт строку «79»
```

Serial.println

Описание:

Функция Serial.println аналогична функции Serial.print, и имеет такие же варианты вызова. Единственное отличие заключается в том, что после данных дополнительно выводятся два символа – символ возврата каретки (ASCII 13, или '\r') и символ новой линии (ASCII 10, или '\n').

Пример 1 и пример 2 выведут в порт одно и то же:

Пример 1:

```
int b = 79;
Serial.print(b, DEC); //выдаст в порт строку «79»
Serial.print("\r\n"); //выведет символы "\r\n" – перевод строки
Serial.print(b, HEX); //выдаст в порт строку «4F»
Serial.print("\r\n");//выведет символы "\r\n" – перевод строки
```

Пример 2:

```
int b = 79;
Serial.println(b, DEC); //выдаст в порт строку «79\r\n»
Serial.println(b, HEX); //выдаст в порт строку «4F\r\n»
```

В мониторе последовательного порта получим:

```
79
4F
```