Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Амурский государственный университет»

Основы технологий создания электронных бортовых систем (часть 1)

Учебно-методическое пособие

Благовещенск
Издательство АмГУ
2020

Основы технологий создания электронных бортовых систем (часть 1). Учебно-методическое пособие / сост.: Аревков М.А. – Благовещенск: Изд-во АмГУ, 2020. – 38 с.

Учебно-методическое пособие по дисциплине «Основы технологий создания электронных бортовых систем (часть 1)» предназначено для подготовки бакалавров по направлению 24.03.01 «Ракетные комплексы и космонавтика».

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Методическое пособие представляет собой сборник задач для самостоятельной работы студентов по дисциплине «Основы технологий создания электронных бортовых систем».

Методическое пособие посвящено проектированию электронных устройств на основе микроконтроллерной платформы Arduino. В методическом пособии приведены сведения об аппаратном и программном обеспечении Arduino, изложены принципы программирования в интегрированной среде Arduino IDE.

Платформа Arduino будет отличным вариантом для проектирования микропроцессорных систем.

Для реализации проектов на основе Arduino понадобится компьютер с операционной системой Mac OS, Windows или Linux с установленной интегрированной средой разработки IDE для Arduino. Интегрированная среда разработки находится в свободном доступе на официальном сайте.

1 ОБЩИЕ СВЕДЕНИЯ

Arduino

Arduino является платформой для разработки устройств на базе микроконтроллера. Язык программирования является интегрированная среда Arduino IDE. Подключив различные датчики, приводы, платы расширения и дополнительные микросхемы, можно использовать Arduino в качестве интеллекта для любой системы управления.

Аппаратная часть

Все платы Arduino содержат:

- Микроконтроллер Atmel;
- USB-интерфейс для программирования и передачи данных;
- Стабилизатор напряжения и выводы питания;
- Контакты входов ввода-вывода; индикаторные светодиоды (Debug, Power, Rx, Tx);
- Кнопку сброса;
- Встроенный последовательный интерфейс программирования (ICSP).

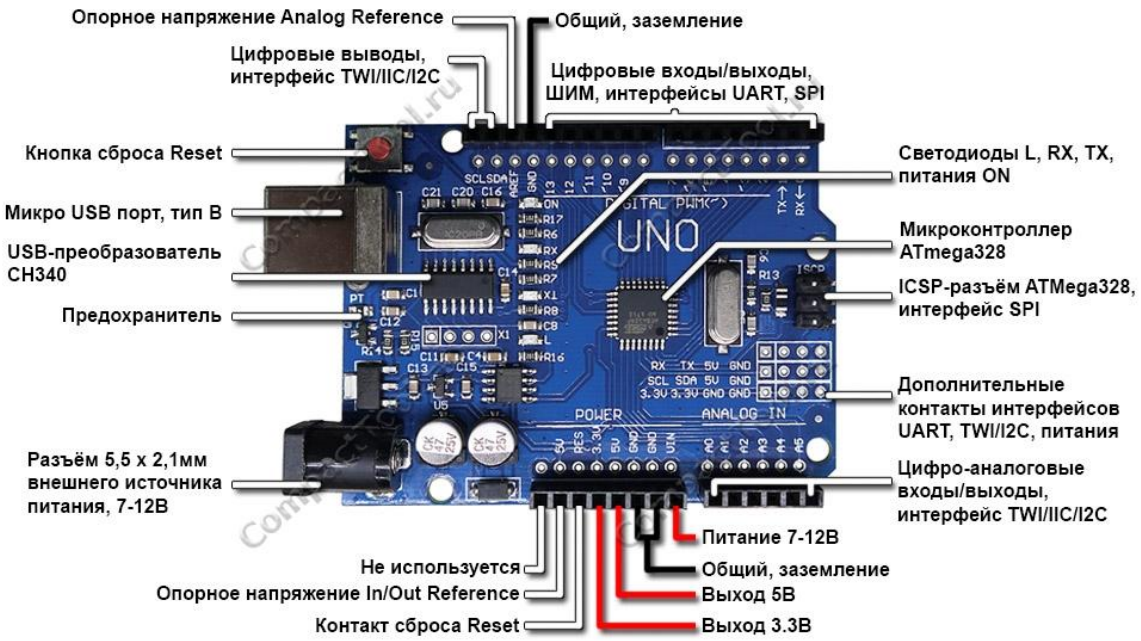

Рисунок 1 – Компоненты платы Arduino Uno

Микроконтроллеры Atmel

Основной элемент платы Arduino является микроконтроллер Atmel. На большинстве плат Arduino используется микроконтроллер ATmega. Исключением является плата Due, укомплектованная микроконтроллером ARM Cortex.

Микроконтроллер выполняет весь код программы. Язык Arduino предоставляет доступ к периферийным устройствам микроконтроллера:

- аналого-цифровым преобразователям (ADCs);
- цифровым портам ввода-вывода, коммуникационным шинам (включая I^2C и SPI) и последовательным интерфейсам.

На плате все эти порты выведены на штырьковые контакты.

Подключен кварцевый резонатор на 16 МГц к тактовым контактам микроконтроллера ATmega

Выполнение программы можно перезапустить с помощью кнопки сброса.

Интерфейсы программирования

Программы микроконтроллера ATmega обычно написаны на C или ассемблере. Коды программы загружаются в микроконтроллер через интерфейс ICSP с помощью программатора.

Главной особенностью Arduino является программирование через USB-порт, без дополнительного программатора. На микроконтроллер ATmega записан специальный загрузчик, который позволяет загружать программу на плату по последовательному порту USART.

Загрузчик — это фрагмент программного кода, который записан в зарезервированное пространство памяти программы. Микроконтроллеры AVR обычно программируются с помощью ICSP, который взаимодействует с микроконтроллером через последовательный периферийный интерфейс (SPI). Для этого способа предполагается наличие программатора.

Рисунок 2 – AVR программатор ISP MKII

После включения платы запускается загрузчик, который будет работать в течение нескольких секунд. В течение этих секунд происходит следующее:

- Если загрузчик получает команду программирования от IDE по последовательному интерфейсу UART, то загружается программа в свободную область памяти микроконтроллера.
- Если команда не поступает, запускается последняя программа, находящаяся в памяти Arduino.

Если подается команда загрузки от IDE Arduino (компьютера)

вспомогательный контроллер сбрасывает основной микроконтроллер, подготавливая его тем самым к загрузке. Затем внешний компьютер начинает отправлять код программы.

Загрузчики реализуют простое программирование через USB без внешних аппаратных средств, поэтому занимают в памяти много места. У загрузчиков есть два основных недостатка:

- Они занимают место в памяти;
- Выполнение программы всегда будет задерживаться на несколько секунд при начальной загрузке.

Есть возможность удалить загрузчик из своего контроллера ATmega и программирование осуществлять с помощью внешнего программатора.

Цифровые и аналоговые контакты ввода-вывода

Все контакты у контроллеров Arduino могут служить цифровыми входами и выходами. Часть контактов Arduino работают в качестве аналоговых входов. Многие из контактов работают в режиме мультиплексирования и выполняют дополнительные функции: различные коммуникационные интерфейсы, последовательные интерфейсы, широтно-импульсные модуляторы и внешние прерывания.

Источники питания

Для работы платы Arduino достаточно 5-вольтового питания, которое получаем от кабеля USB. Если необходима разработка автономного устройства, платы Arduino могут работать от внешнего источника от 6 до 20В. Внешнее питание может подаваться через разъем DC или на контакт v_{in}.

У плат Arduino есть встроенные стабилизаторы:

- Напряжение 5В: используется для всех логических элементов на плате;
- Напряжение 3,3В используется на отдельный контакт для подключения внешних устройств.

Платы Arduino

Рассматривать все существующие платы Arduino не будем, так как их много и постоянно выпускаются новые. Кратко рассмотрим основные платы.

• *Arduino Uno* — основная плата Arduino. Плата укомплектована микроконтроллером ATmega 328 и микросхемой 16U2 преобразователя USB.

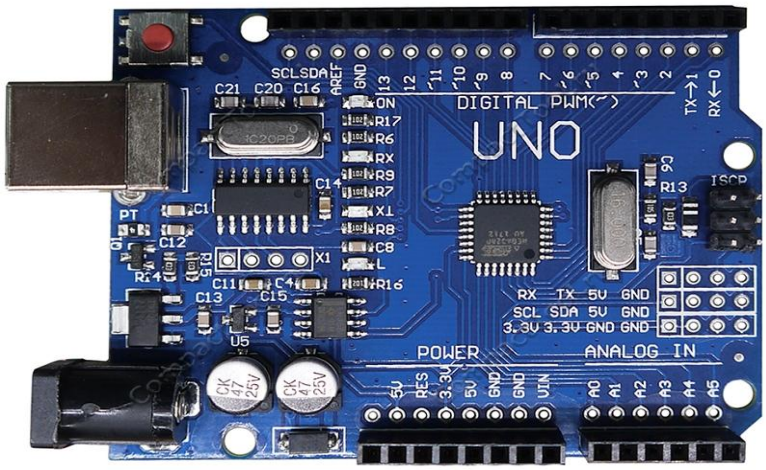

Рисунок 3 – Плата Arduino Uno

• *Arduino Leonardo* — На плате установлен контроллер 32U4 со встроенным интерфейсом USB. Это уменьшает стоимость изделия и дает возможность использовать плату в качестве USB-устройства.

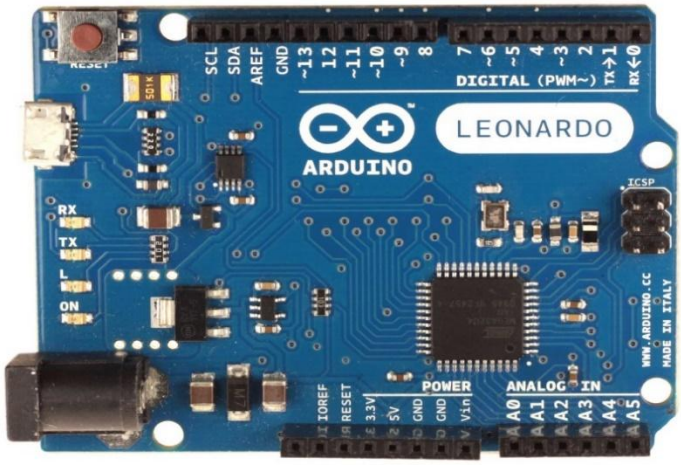

Рисунок 4 – Плата Arduino Leonardo

• *Arduino Mega 2560* — На плате установлен контроллер ATmega 2560, имеющий 54 цифровых входа-выхода. У Arduino Mega 2560 увеличено число аналоговых входов и последовательных портов.

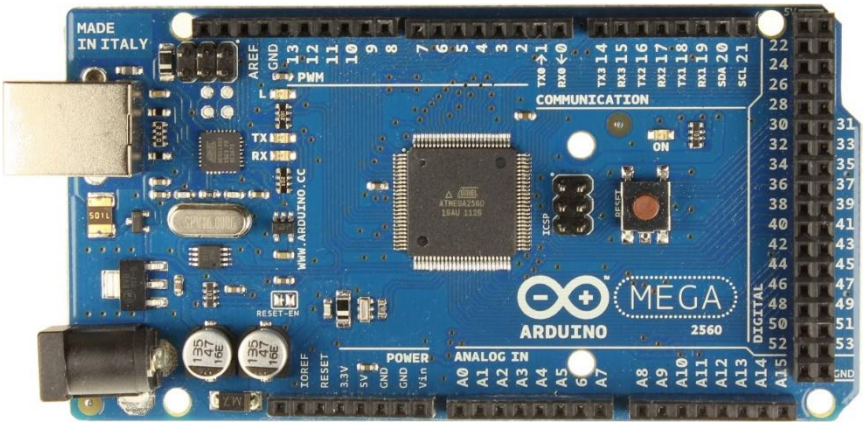

Рисунок 5 – Плата Arduino Mega 2560

• *Arduino Due* — плата создана на базе 32-разрядного процессора Atmel SAM3X8E ARM Cortex-M3 с тактовой частотой 84 МГц. Отличительные особенности платы: повышенная точность аналого-цифрового преобразователя, настраиваемая частота сигнала ШИМ, отдельные выводы цифроаналогового преобразователя, наличие встроенного последовательного порта.

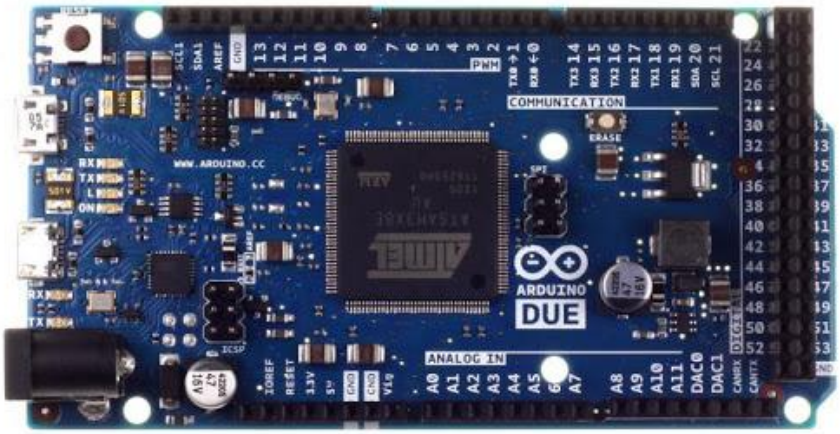

Рисунок 6 – Плата Arduino Due

• *Arduino Nano* — конструкция миниатюрной платы такова, что ее можно установить в панельку для микросхем.

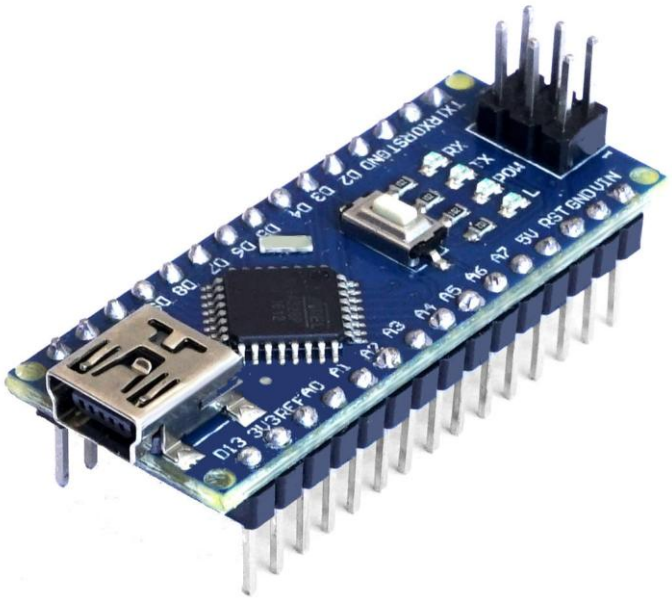

Рисунок 7 – Плата Arduino Nano

• *Arduino Mega ADK* — очень похожа на Arduino mega 2560, но у mega ADK есть дополнительная функциональность интерфейса USB, позволяющая ему соединяться с телефоном на базе Android.

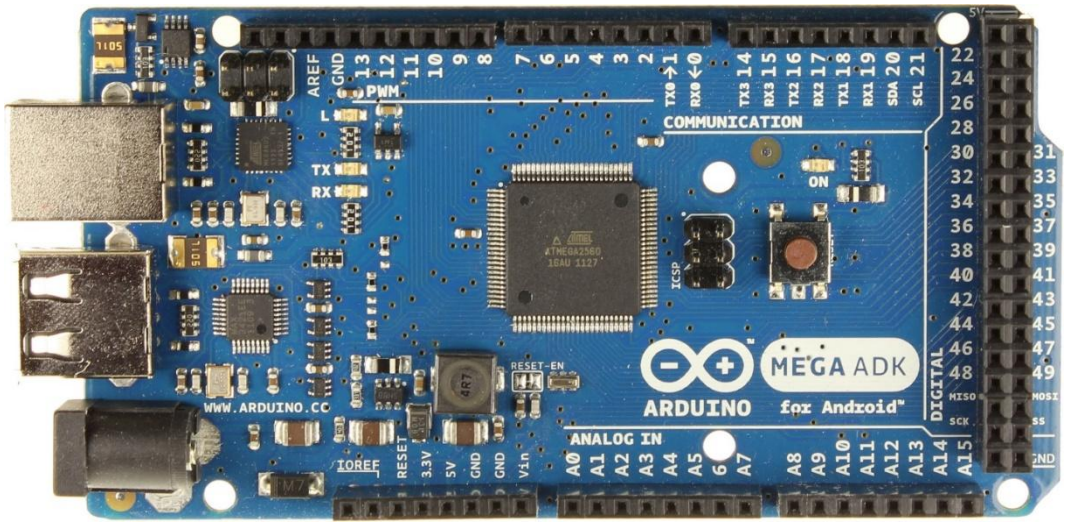

Рисунок 8 – Плата Arduino Mega ADK

2 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Загрузка и установка Arduino IDE

Зайдите на официальный сайт Arduino http://www.arduino.cc и загрузите последнюю версию Arduino IDE.

После завершения загрузки разархивируйте загруженный файл. В папке вы найдете Arduino IDE.

Запуск IDE и подключение к Arduino

Подключите Arduino к компьютеру с помощью кабеля USB, как изображено на рисунке. Компьютеры с операционной системой Mac и Linux установят драйверы автоматически. На операционной системе Windows возможно придется установить драйвера вручную.

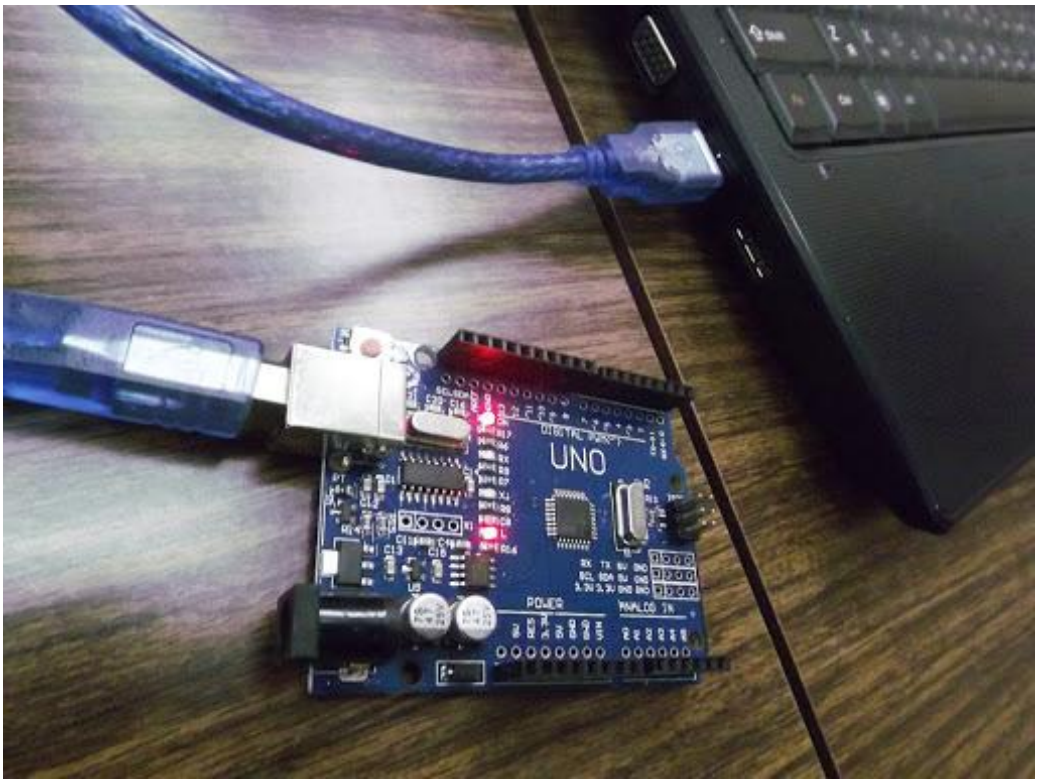

Рисунок 9 – Соединение Arduino Uno с компьютером с помощью USB – кабеля

Анализируем программу Blink

Подробно рассмотрим текст программы Blink, чтобы понять базовую структуру программ, написанных для Arduino.

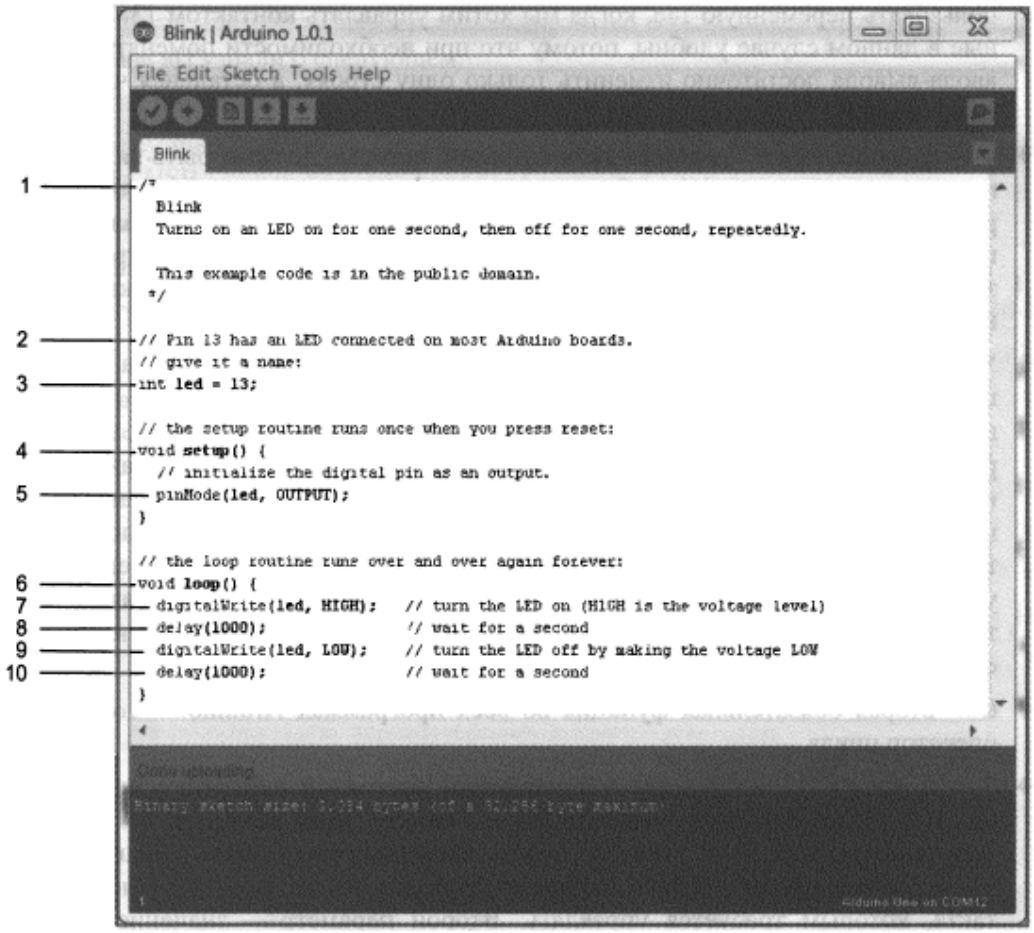

Рисунок 10 – Структура программы Blink

Цифрами на рисунке 10 обозначено следующее:

♦ 1 — *Многострочный комментарий.* Комментарии нужны для пояснения кода программы. Многострочные комментарии начинаются с /* и заканчиваются */. Все, что написано между этими символами, не будет обрабатываться компилятором.

♦ 2 — *Однострочный комментарий.* Если поместить // на любую строку, компилятор проигнорирует весь текст строки после этого символа.

♦ 3 — *Код объявления переменной*. Переменная — это ячейка памяти, содержащая информацию. Существуют переменные различных типов. В

примере указана переменная int, что означает целое число. Целочисленной переменной led присвоено значение 13 — это номер цифрового контакта, к которому подключен светодиод на плате Arduino. Всюду в остальной части программы можно использовать переменную led, когда мы хотим управлять контактом 13. Переменные в данном случае удобны, потому что при необходимости поменять контакт ввода-вывода достаточно изменить только одну строку, а остальная часть кода не изменится.

♦ 4 — *Функция setup ()*. Одна из двух функций, которые должны быть включены в каждую программу Arduino. Функция — это фрагмент кода, выполняющий определенную задачу. Код в теле функции setup () выполняется один раз в начале программы. Это полезно для установки начальных параметров настройки, назначения режимов портов ввода-вывода, инициализации коммуникационных интерфейсов и т. д.

♦ 5 — *Команда pinMode ()*. Цифровые контакты могут быть запрограммированы на ввод или вывод. Сконфигурировать их направление позволяет команда pinMode (), имеющая два параметра, указанных в круглых скобках. Первый параметр pinMode определяет номер контакта. Поскольку переменная led уже назначена ранее в программе, конфигурация задается для контакта 13. Второй параметр устанавливает направление контакта: input (вход) или output (выход). По умолчанию все контакты настроены на ввод. Чтобы сконфигурировать их на вывод, следует явно указать значение этого параметра output. Поскольку нам нужно управлять светодиодом, контакт 13 должен быть выходом.

♦ 6 — *Функция loop ()*. Вторая обязательная функция во всех программах. Это оператор цикла.

♦ 7— *Функция digitaiwrite ()*. Устанавливает состояние выходного контакта: 5 или 0 В. Если светодиод подсоединен к контакту через резистор, то установка значения логической "1" позволит зажечь светодиод. Первый параметр функции digitaiwrite () — номер контакта, которым требуется управлять. Второй параметр— значение, которое нужно задать: high (5 В)

или low (0 В). Контакт остается в этом состоянии, пока не будет изменен следующей командой digitaiwrite ().

♦ 8 — *Функция delay ()*. Эта функция имеет один аргумент — время задержки выполнения программы в миллисекундах. При вызове delay () Arduino останавливает выполнение программы на определенный интервал времени. В нашем примере задержка равна 1000 мс (1 с). Это приводит к свечению светодиода в течение одной секунды до выполнения следующей команды.

♦ 9 — *Функция digitaiwrite ()*. Здесь функция нужна, чтобы выключить светодиод, устанавливая состояние контакта в low.

♦ 10— *Функция delay ()*. Делаем задержку на одну секунду, чтобы светодиод был погашен перед повторением цикла.

Задание:

Установить программное обеспечение на компьютер. Написать код программы приведенного на рисунке 10 с изменением времени задержки свечения светодиода. Загрузить программу на плату Arduino.

Подключение внешнего светодиода

Мигающий светодиод из предыдущего примера был встроен в плату Arduino. Рассмотрим пример подключение внешнего светодиода. Соединим светодиод с контактом 9. Контакт 9 Arduino позволяет формировать сигнал широтно-импульсной модуляции.

Работа с макетной платой

Макетная плата — позволяет легко собирать простые схемы без изготовления печатных плат и пайки. С двух сторон по всей длине макетной платы расположены красные и синие отверстия. Все красные отверстия соединены между собой и служат, как правило, для подачи питания. Все синие отверстия тоже электрически соединены друг с другом и играют роль

шины заземления. Каждые пять отверстий, расположенных вертикальными рядами, также соединены друг с другом. Электрические соединения отверстий показаны на рисунке утолщенными линиями.

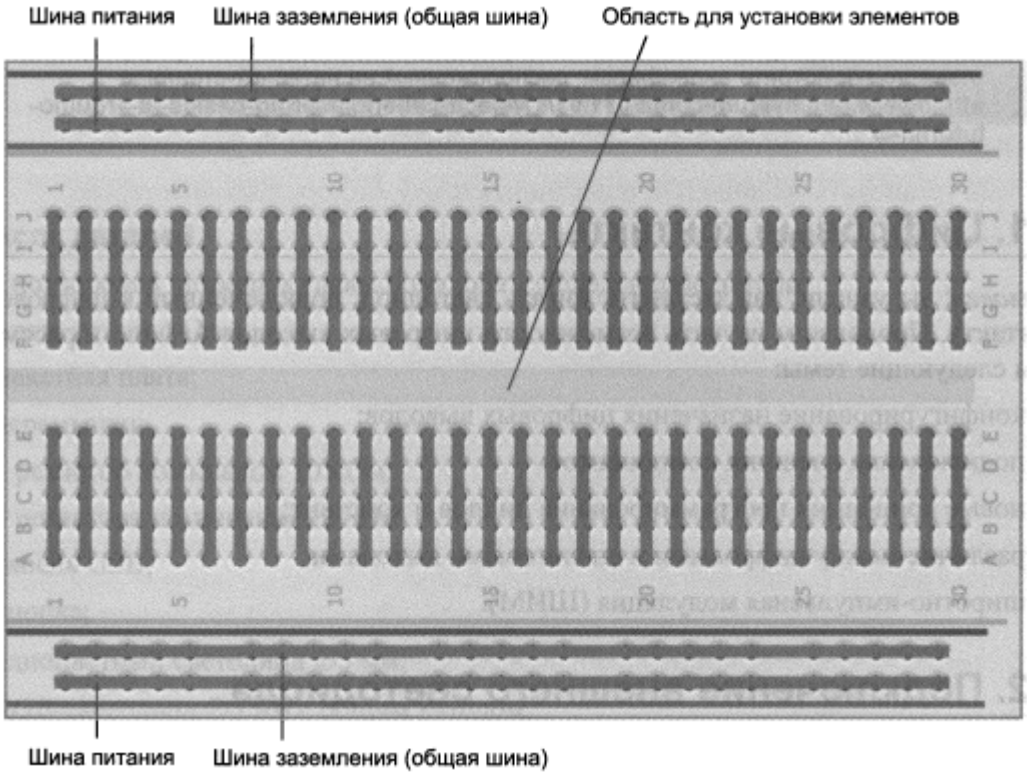

Рисунок 11 – Электрические соединения макетной платы

Подсоединение светодиодов

При подключении светодиода, необходимо соблюдать полярность. Положительный вывод называется анодом, отрицательный — катодом. Определить назначение контактов светодиода можно визуально: вывод катода короче, чем анода.

Ток через светодиод течет от анода к катоду. Поскольку ток протекает от положительного полюса к отрицательному, анод светодиода следует подключить к источнику тока (цифровой выход +5 В), а катод — к земле. Полярность подключения для резисторов не важна.

Подключать светодиод к контакту 9 нужно последовательно с резистором. Резистор выступает в качестве ограничителя тока. Чем больше

сопротивление резистора, тем сильнее он ограничивает ток. В этом примере применяется резистор номиналом 220 Ом. Монтажная схема изображена на рисунке 12.

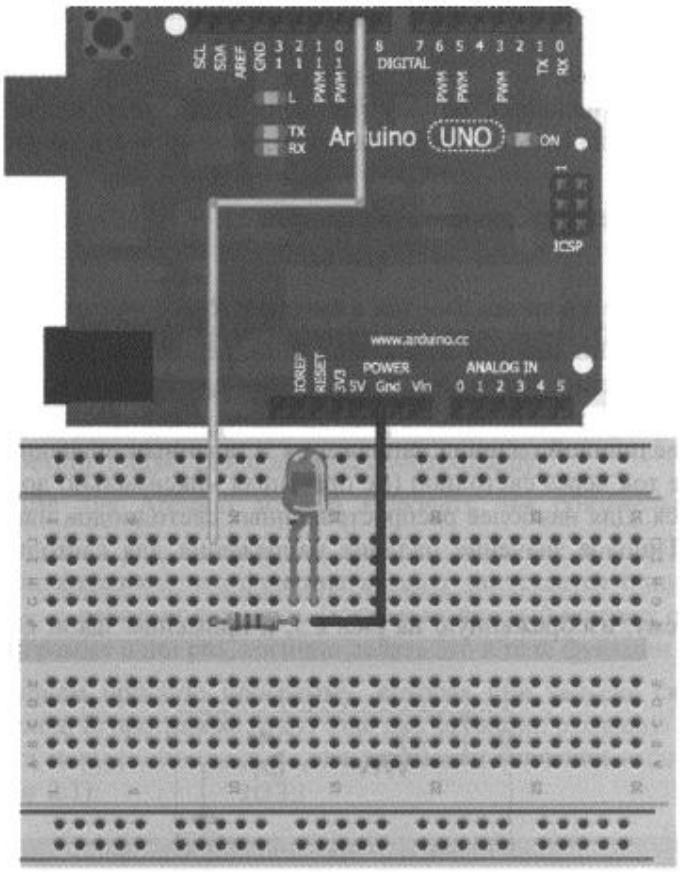

Рисунок 12 – Подключение светодиода к плате Arduino Uno

Закон Ома и формула для расчета мощности

Закон Ома позволяет определяет соотношение между напряжением (измеряется в вольтах), током (измеряется в амперах) и сопротивлением (измеряется в Ом) в цепи. Схема представляет собой замкнутый контур с источником электрической энергии и нагрузкой.

Закон Ома определяется следующим образом:

U=IR,

где U— напряжение в вольтах;

I— ток в амперах;

R — сопротивление в Ом.

В электрической цепи каждый компонент обладает некоторым сопротивлением, что снижает напряжение. Закон Ома нужен для подбора значения резистора, подключаемого последовательно со светодиодом. Светодиоды характеризуются определенной величиной падения напряжения и заданным значением рабочего тока. Чем больше ток через светодиод (не превышая максимально допустимого), тем ярче он светится. Для наиболее распространенных светодиодов максимальный ток равен 20 мА. Типовое значение падения напряжения для светодиода составляет около 2 В. Рассмотрим схему, изображенную на рисунке 13, и применим закон Ома для подбора резистора R1.

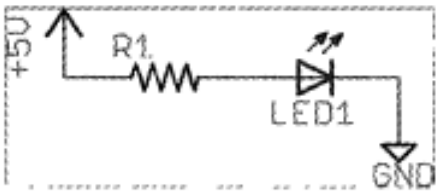

Рисунок 13 – Схема включения светодиода

LED1 является стандартный светодиод с прямым током 20 мА и падением напряжения 2В. Напряжение питания 5В должно перераспределиться между светодиодом и резистором. Поскольку светодиод потребляет 2В, оставшиеся 3В должны быть приложены к резистору. Зная максимальное значение прямого тока через светодиод (20 мА), можно найти номинал резистора:

R=U/I= 3/0,02 = 150 Ом.

При сопротивлении резистора 150 Ом через него и светодиод протекает ток 20 мА. По мере увеличения сопротивления ток будет уменьшаться. Резистор 220 Ом обеспечивает достаточную яркость свечения светодиода, к тому же этот номинал очень распространен.

Еще одно важное соотношение — формула для расчета мощности, которая показывает, сколько ватт рассеивается на каждом компоненте.

Увеличение мощности рассеивания связано с ростом тепловыделения прибора. Для каждого компонента, как правило, задается максимально допустимая мощность. Максимальная мощность резистора в нашем примере равна 0,125 Вт. Формула для расчета мощности выглядит следующим образом:

P = UI

где P — мощность, Вт;

U— напряжение, В;

I— сила тока, А.

Для резистора из схемы на рисунке 12 при падении напряжения 3 В и силе тока 20 мА мощность равна

P = 3 ⋅ 0,02 = 0,06 Вт.

Поскольку 60 мВт < 0,125 Вт = 125 мВт, следовательно, данный резистор не перегреется.

Программирование цифровых выводов

Все внешние контакты сконфигурированы как входы. Если необходимо использовать контакт как выход, нужно его переконфигурировать, подав соответствующую команду микроконтроллеру.

Каждая программа должна включать две обязательные функции:

- setup ();

- loop ().

Каждый контакт обычно конфигурируется в программе один раз, поэтому это нужно делать в теле функции setup ().

Рассмотрим программу, которая при запуске сконфигурирует контакт 9 как выход. В программе будут еще две функции:

- pinMode () — для конфигурации контакта;

- digitaiwrite () — для установки значения high (5В).

```
const int LED=9;           // Константа - номер контакта светодиода

void setup()
{
  pinMode (LED, OUTPUT);   // Конфигурируем контакт светодиода как выход
  digitalWrite(LED, HIGH); // Устанавливаем значение HIGH на выходе
}

void loop() {
                           // В цикле ничего не выполняем
}
```

Рисунок 14 – Листинг программы светодиода

После сборки схемы представленного на рисунке 12, загружается листинг программы представленного на рисунке 14. В этой программе используется оператор инициализации константы перед определением значения контакта. Обычно для хранения значений, которые могут изменяться во время выполнения программы, предназначены переменные. Поставив оператор const до объявления переменной, она не будет изменяться во время выполнения программы. Всем экземплярам переменной led в программе будет присвоено значение 9. В виде констант рекомендуется определять значения, которые не будут меняться при выполнении программы.

При объявлении любой переменной необходимо указать ее тип.

Задание:

Изучить подключение светодиода к плате Arduino Uno. Изучить закон Ома и формулу для расчета мощности. Изучить работу с макетной платой. Написать листинг программы светодиода с изменением номера контакта светодиода и с изменением подключение светодиода к плате Arduino Uno. Загрузить листинг программы на плату Arduino Uno с помощью USB кабеля.

Использование цикла

На практике часто необходимо циклически изменять значения

переменных для выполнения заданного алгоритма. В предыдущем примере можно реализовать цикл, чтобы увидеть, как влияют на частоту мигания разные значения задержки. Можно реализовать разные скорости мигания, задавая с помощью переменной цикла различные значения задержки. Пример иллюстрирует код из листинга рисунка 15.

```
const int LED=9;           // Константа номера контакта светодиода

void setup()
{
  pinMode (LED, OUTPUT);  // Конфигурируем контакт светодиода как выход
}

void loop()
{
  for (int i=100; i<=1000; i=i+100)
  {
    digitalWrite(LED, HIGH);
    delay(i);
    digitalWrite(LED, LOW);
    delay(i);
  }
}
```

Рисунок 15 – Листинг программы изменение часты мигания светодиода

Скомпилировав код листинга (рисунок 15), загружаем его на плату Arduino. Теперь разберемся, как это работает.

Оператор for всегда содержит три выражения, разделенные точкой с запятой:

♦ Первое выражение присваивает начальное значение переменной-счетчику цикла. В данном случае i получает начальное значение 100;

♦ Второе выражение указывает, когда цикл должен остановиться. Операторы в теле цикла будут выполняться снова и снова, пока условие истинно. Запись <= означает меньше или равно. Таким образом, этот цикл будет выполняться до тех пор, пока переменная i меньше или равна 1000;

♦ Последнее выражение указывает, что должно произойти с переменной i каждый раз после выполнения операторов тела цикла. В данном случае значение счетчика цикла увеличивается на 100.

Чтобы понять работу оператора for, подробно рассмотрим, что происходит за два прохода цикла:

1. Значение переменной i равно 100, 100 меньше или равно 1000, значит выполнять код в теле цикла;

2. На контакте 9 установлено значение high, светодиод горит 100 мс (текущее значение i);

3. На контакт 9 подано значение low, светодиод потушен 100 мс (текущее значение i);

4. В конце цикла значение переменной i увеличивается на 100, теперь i равно 200;

5. 200 меньше или равно 1000, цикл повторяется снова;

6. На контакте 9 установлено значение high, светодиод горит 200 мс (текущее значение i);

7. На контакт 9 подано значение low, светодиод потушен 200 мс (текущее значение i);

8. В конце цикла значение переменной i увеличивается на 100, теперь i равно 300;

9. Этот процесс повторяется, пока i не превосходит 1000 и затем i снова принимает значение 100 и все повторяется заново.

Задание:

Изучить работу оператора for. Написать листинг программы рисунка 15 с изменением в операторе for переменных. Собрать схему, указанную на рисунке 12. Загрузить листинг программы на плату Arduino Uno.

Широтно-импульсная модуляция с помощью analogWrite()

Необходимо вывести напряжение, отличное от 0 и 5 В. С помощью одной платы Arduino Uno это невозможно. Придется задействовать цифроаналоговый преобразователь или взять плату Arduino Due или добавить внешнюю микросхему ЦАП. А можно сымитировать генерацию

аналоговых значений на цифровых контактах с помощью широтно-импульсной модуляции (ШИМ). Для некоторых контактов Arduino сформировать ШИМ-сигнал можно командой analogWrite().

ШИМ-сигнал могут выдавать контакты на определенные периферийные устройства, помечены символом ~ на плате Arduino.

Если нужно уменьшить напряжение на контакте 9, яркость свечения светодиода будет меньше, потому что снизится ток, текущий через него. Этого эффекта можно добиться с помощью команды analogWrite ().

Функция analogWrite () имеет два аргумента:

- номер контакта;

- 8-разрядное значение в диапазоне от 0 до 255.

В листинге рисунок 16 приведен код программы генерации ШИМ-сигнала на контакте 9 для плавного управления яркостью светодиода.

```
const int LED=9;  // Константа номера контакта светодиода

void setup()
{
  pinMode (LED, OUTPUT);  // Конфигурируем контакт светодиода как выход
}

void loop()
{
  for (int i=0; i<256; i++)
  {
    analogWrite(LED, i);
    delay(10);
  }
  for (int i=255; i>=0; i--)
  {
    analogWrite(LED, i);
    delay(10);
  }
}
```

Рисунок 16 – Листинг программы плавное изменение яркости светодиода

При выполнении листинга программы будет наблюдаться, как свечение светодиода изменяется от тусклого к яркому в одном цикле for, а затем от яркого к тусклому в другом цикле for. Все это будет происходить в основном цикле loop () до бесконечности.

Существует различие двух циклов for. В первом цикле выражение i++ является сокращением кода i=i+i. Аналогично, запись i— эквивалентна коду i=i-i. Первый цикл плавно зажигает светодиод до его максимальной яркости, второй — постепенно гасит его.

Для большинства случаев ШИМ пригодна для эмуляции аналогового выхода, но этот вариант неприемлем, когда требуется неискаженный аналоговый сигнал. Например, ШИМ отлично подходит для регулировки скорости двигателя постоянного тока, но не годится для управления аудиоколонками.

Рассмотрим графики, представленные на рисунке 17. ШИМ представляет собой изменение скважности (отношения периода к длительности импульса) прямоугольной последовательности импульсов. Скважность можно трактовать как процент времени, когда прямоугольный импульс имеет уровень high, ко всему периоду повторения. Скважность 50% означает, что половину периода сигнал имеет высокий уровень, а половину — низкий.

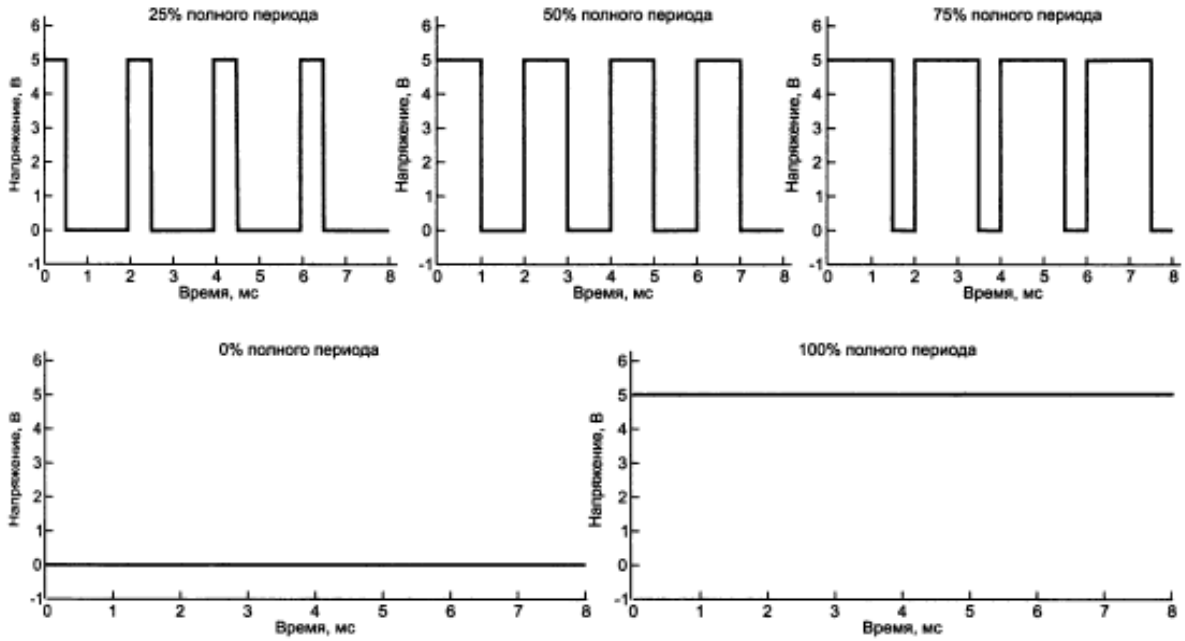

Рисунок 17 – ШИМ-сигналы с различной скважностью

Функция analogWrite () устанавливает скважность последовательности прямоугольных импульсов в зависимости от значения, передаваемого ей:

♦ значение аргумента analogWrite (), равное нулю, задает скважность 0% (всегда low);

♦ значение 255 — скважность 100% (всегда high);

♦ значение 127 соответствует скважности 50% (половина времени high, половина времени low).

На графиках рисунка 17 видно, что для сигнала со скважностью 25% значение high действует в течение четверти периода, а остальные 75% времени установлено значение low. Частота прямоугольной последовательности импульсов в случае с Arduino составляет приблизительно 490 Гц. Другими словами, уровень сигнала меняется от высокого (5 В) к низкому (0 В) приблизительно 490 раз каждую секунду.

Как видим, напряжение, подаваемое на светодиод, на самом деле не понижается, почему же при уменьшении скважности наблюдается спад яркости свечения светодиода? Это связано с особенностью нашего зрения. Если светодиод включается и выключается один раз за 1 мс (при скважности 50%), то вам кажется, что яркость свечения светодиода составляет приблизительно 50% от максимальной, потому что переключение происходит быстрее, чем глаза могут это зафиксировать. Ваш мозг фактически усредняет сигнал и создается впечатление, что светодиод работает на половине яркости.

Задание:

Изучить принцип работы широтно-импульсной модуляции с помощью команды analogWrite(). Собрать схему указанной на рисунке 12. Прописать листинг программы, указанной на рисунке 16. Загрузить листинг программы на плату Arduino Uno и посмотреть работу светодиода. Изменить переменные в листинге программы и изменённую программу записать на плату Arduino Uno.

Считывание данных с цифровых контактов

Рассмотрим еще одну функцию цифровых контактов. Мы использовали контакты в качестве выходов, генерируя цифровой сигнал и ШИМ-сигнал. Теперь будем использовать контакты платы Arduino в качестве входов.

Считывание цифровых входов со стягивающим резистором

Подключим кнопку и стягивающий резистор, в результате схема примет вид, представленный на рисунке 18.

Для установки «значения по умолчанию» на входном контакте, почти для всех цифровых входов необходим дополнительный стягивающий (pull-down) или подтягивающий (pull-up) резисторы. Если бы не было резистора в схеме на рисунке 18, то при нажатии на кнопку на выводе будет значение high. Но если кнопка не нажата, то в такой ситуации входной контакт не привязан ни к чему.

Поскольку вывод физически не подключен ни к 0 В, ни к 5 В, то электрические помехи на близлежащих выводах могут привести к тому, что значение напряжения будет колебаться между high и low. Чтобы такого не было, стягивающий резистор подключают так, как показано на рисунке 18.

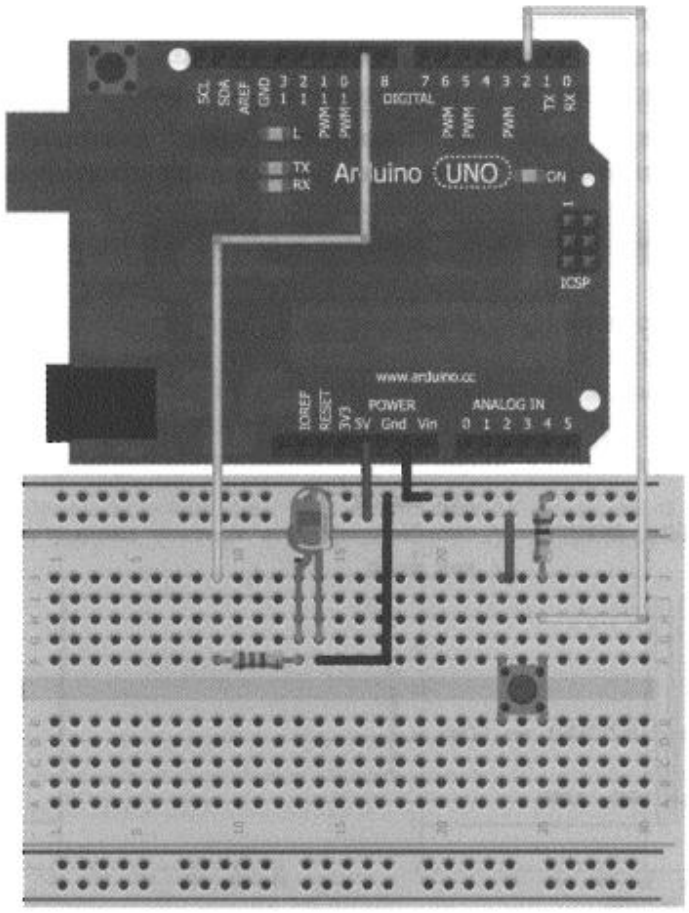

Рисунок 18 – ШИМ-сигналы с различной скважностью

Рассмотрим, что происходит, когда кнопка не нажата, а входной контакт подключен через стягивающий резистор 10 кОм к земле.

Через резистор протекает ток утечки, поэтому на входном контакте будет установлено значение напряжения low. При нажатии на кнопку входной контакт оказывается напрямую связан с шиной 5 В. Теперь ток может течь двумя путями:

♦ через практически нулевое сопротивление нажатой кнопки к шине 5 В;

♦ через высокое сопротивление резистора на землю.

Ток всегда будет идти по пути наименьшего сопротивления в соответствии с законом Ома. Большая часть тока будет протекать через замкнутую кнопку и на входе установится уровень high.

Стягивающие и подтягивающие резисторы важны, потому что они

гарантируют, что кнопка не создаст короткое замыкание между 5 В и землей при нажатии.

Рассмотрим программу для рассмотренной схемы. Светодиод должен гореть, пока кнопка нажата, и быть выключенным, когда кнопка отжата (листинг рисунка 19).

```
const int LED=9;     // Контакт 9 для подключения светодиода
const int BUTTON=2;  // Контакт 2 для подключения кнопки

void setup()
{
  pinMode (LED, OUTPUT);   // Сконфигурировать контакт светодиода
                           // как выход
  pinMode (BUTTON, INPUT); // Сконфигурировать контакт кнопки
                           // как вход
}

void loop()
{
  if (digitalRead(BUTTON) == LOW)
  {
    digitalWrite(LED, LOW);
  }
  else
  {
    digitalWrite(LED, HIGH);
  }
}
```

Рисунок 19 – Листинг программы свечение светодиода с помощью кнопки

В коде листинга рисунка 19 реализованы некоторые новые элементы:

- *Функция digitalRead ()*;

- *Оператор if /else*.

Константа button типа int добавлена для контакта кнопки.

Функция digitalRead () считывает значение сигнала на входе. Если кнопка нажата, digitalRead () возвращает значение high (лог. 1). Если кнопка не нажата, то получаем low (лог. 0).

Проверяем содержимое внутри оператора if (). Если условие внутри оператора if истинно, вызываем функцию digitaiWrite(LED, low). В противном случае выполняем код после оператора else digitalWrite (LED, HIGH)).

Задание:

Изучить принцип работы светодиода с кнопкой. Изучить принцип работы функции digitalRead () и оператора if/else. Написать листинг программы, указанной на рисунке 19 и собрать схему подключение указанной на рисунке 18. Записать программу на плату Arduino Uno.

Устранение «дребезга» кнопки

Кнопкой является механическое устройство с пружинным контактом. При нажатии на кнопку сигнал не просто меняется от низкого до высокого, он на протяжении нескольких миллисекунд неоднократно меняет свое значение, прежде чем установится уровень low. Отличие ожидаемого процесса от реального иллюстрируют осциллограммы сигнала с кнопки, приведенные на рисунок 20.

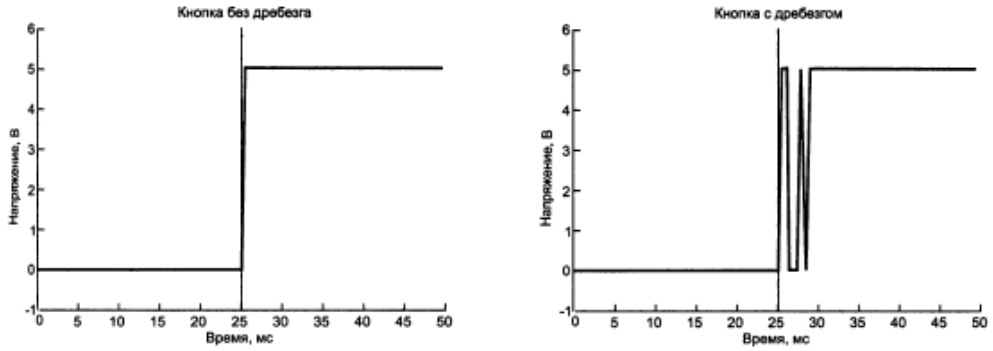

Рисунок 20 – Эффект дребезга кнопок

Кнопка была нажата в течение 25 мс. Предположение, что состояние кнопки можно определить, считав значение с входа контакта (график слева) неверно. Кнопка фактически возвращается вверх-вниз, пока значение не установится (график справа). Зная, как ведет себя кнопка, можно написать программу для кнопки с дребезгом, которая фиксирует изменение состояния кнопки, некоторое время ждет и затем снова читает состояние переключателя. Алгоритм работы такой программы можно записать следующим образом:

1. Сохраняем предыдущее и текущее состояния кнопки;

2. Считываем текущее состояние кнопки;

3. Если текущее состояние кнопки отличается от предыдущего, ждем 5 мс, потому что кнопка, возможно, изменит свое состояние;

4. Подождав 5 мс, считываем состояние кнопки и делаем его текущим состоянием кнопки;

5. Если предыдущее состояние кнопки было low, а текущее— high, переключаем состояние светодиода;

6. Устанавливаем предыдущее состояние кнопки в качестве текущего;

7. Возвращаемся к шагу 2/

Данный алгоритм —пример для изучения функций.

Функция — это оператор, который может принимать входные аргументы, выполнять фрагмент кода с их использованием и, возможно, возвращать результат. Например, digitaiWrite () — это функция, которая принимает в качестве аргументов номер контакта и значение (high или low), и устанавливает это значение на контакте.

Процесс выполнения программы представляет собой многократное повторение шагов. Составим функцию для устранения дребезга контактов, которую можно вызывать неоднократно. Функция будет принимать предыдущее состояние кнопки в качестве входных данных, выполнять противодребезговую защиту и выводить установившееся состояние кнопки. Основной цикл программы переключает состояние светодиода при каждом нажатии кнопки.

```
const int LED=9;                // Контакт 9 для подключения светодиода
const int BUTTON=2;             // Контакт 2 для подключения кнопки
boolean lastButton = LOW;       // Переменная для сохранения предыдущего
                                // состояния кнопки
boolean currentButton = LOW;    // Переменная для сохранения текущего
                                // состояния кнопки
boolean ledOn = false;          // Текущее состояние светодиода
                                //(включен/выключен)

void setup()
{
  pinMode (LED, OUTPUT);         // Сконфигурировать контакт светодиода
                                 // как выход
  pinMode (BUTTON, INPUT);       // Сконфигурировать контакт кнопки
                                 // как вход
}
/*
* Функция сглаживания дребезга
* принимает в качестве аргумента предыдущее состояние кнопки
* и выдает фактическое.
*/
boolean debounce(boolean last)
{
  boolean current = digitalRead(BUTTON);  // Считать состояние кнопки
  if (last != current)                    // Если изменилось...

  {
    delay(5);                             // Ждем 5 мс
    current = digitalRead(BUTTON);        // Считываем состояние кнопки
    return current;                       // Возвращаем состояние кнопки
  }
}

void loop()
{
  currentButton = debounce(lastButton);
  if (lastButton == LOW && currentButton == HIGH) // Если нажатие
  {
    ledOn = !ledOn;                               // Инвертировать значение
                                                  // состояния светодиода
  }
  lastButton = currentButton;
  digitalWrite(LED, ledOn);                        // Изменить статус
                                                   // состояния светодиода
}
```

Рисунок 21 – Листинг программы подавление дребезга кнопки

Рассмотрим текст листинга рисунка 21 подробнее. Сначала заданы номера контактов для подключения кнопки и светодиода. Затем объявлены три глобальные логические переменные, которые будут изменяться в программе. Каждой из трех переменных присвоены начальные значения (low, low и false). Далее в программе значения этих переменных могут изменяться с помощью оператора присваивания =.

Рассмотрим функцию подавления дребезга кнопки boolean debounce (). Эта функция принимает логическую переменную (имеющую только два состояния: true/false, high/low, вкл./выкл., 1/0) предыдущего состояния кнопки и возвращает текущее значение состояния кнопки. Внутри функции текущее состояние кнопки сравнивается с предыдущим с помощью оператора != (не равно). Если состояния отличаются, то кнопка, возможно, нажата. Затем ожидаем 5 мс (этого достаточно, чтобы состояние кнопки стабилизировалось после дребезга), прежде чем проверить состояние кнопки снова. Затем вновь проверяем состояние кнопки. Данная функция возвращает текущее значение булевой локальной переменной, которая объявлена и используется только в функции debounce (). Когда функция debounce () вызывается из основного цикла, возвращенное значение записывается в глобальную переменную currentButton, которая была определена в начале программы.

После вызова функции debounce () и установки значения переменной currentButton происходит сравнение текущего и предыдущего значений состояния кнопки с помощью оператора && (логический оператор "И", означающий, что выражение в скобках выполнится, только если истинно каждое из равенств, разделенных оператором &&).

Если ранее состояние кнопки было low, а теперь high, значит, кнопка была нажата и нужно инвертировать значение переменной ledOn. Это действие выполняет оператор ! перед переменной ledOn. Цикл закончен, обновляем предыдущую переменную состояния кнопки и изменяем состояние светодиода. Программа изменяет состояние светодиода после

каждого нажатия кнопки.

Задание:

Изучить принцип устранения «дребезга» кнопки. Изучить и написать код программы указанного на рисунке 21 с изменением переменных. Собрать схему подключения указанной на рисунке 18. Загрузить полученный листинг программы на плату Arduino Uno.

Создание управляемого ночника на RGB-светодиоде

Подключим к плате Arduino трехцветный RGB-светодиод и создадим ночник, цвет которого будет меняться при нажатии на кнопку. В RGB-светодиоде можно смешивать цвета, изменяя широтно-импульсной модуляцией яркость каждого из них.

В устройстве используем RGB-светодиод с четырьмя выводами, один из которых является катодом, общим для всех трех диодов, а остальные — аноды для диодов каждого цвета. Подключите RGB-светодиод проводами к трем ШИМ-контактам платы Arduino через токоограничивающие резисторы, как показано на рисунке 25.

Можно настроить циклическое переключение цветов светодиода при каждом нажатии на кнопку. В данном случае удобно добавить функцию для установки цвета светодиода в следующее состояние. В программе, представленной в листинге ниже, определено семь цветов и состояние, когда светодиод не горит. С помощью функции analogWrite () можно задать свои цветовые комбинации.

```
const int BLED=9;          // Контакт 9 для вывода BLUE RGB-светодиода
const int GLED=10;         // Контакт 10 для вывода GREEN RGB-светодиода
const int RLED=11;         // Контакт 11 для вывода RED RGB-светодиода
const int BUTTON=2;        // Контакт 2 для входа кнопки

boolean lastButton = LOW;    // Предыдущий статус кнопки
boolean currentButton = LOW; // Текущий статус кнопки
int ledMode = 0;             // Значение статуса RGB-светодиода

void setup()
{
  pinMode (BLED, OUTPUT);     // Сконфигурировать
                              // BLUE контакт светодиода как выход

  pinMode (GLED, OUTPUT);     // Сконфигурировать GREEN
                              // контакт светодиода как выход
  pinMode (RLED, OUTPUT);     // Сконфигурировать RED контакт
                              // светодиода как выход
  pinMode (BUTTON, INPUT);    // Сконфигурировать контакт кнопки
                              // как вход
}
/*
* Функция сглаживания дребезга
* принимает в качестве аргумента предыдущее состояние кнопки
* и выдает фактическое.
*/
boolean debounce(boolean last)
{
  boolean current = digitalRead(BUTTON);  // Считать состояние кнопки
  if (last != current)                    // Если изменилось...
  {
    delay(5);                              // Ждем 5 мс
    current = digitalRead(BUTTON);         // Считываем состояние кнопки
    return current;                        // Возвращаем
                                           // состояние кнопки
  }
}

/*
* Выбор режима светодиода.
* Передача номера режима и установка заданного режима светодиода.
*/
void setMode(int mode)
{
  // Красный
  if (mode == 1)
  {
    digitalWrite(RLED, HIGH);
    digitalWrite(GLED, LOW);
    digitalWrite(BLED, LOW);
  }
  // Зеленый
  else if (mode == 2)
  {
    digitalWrite(RLED, LOW);
    digitalWrite(GLED, HIGH);
    digitalWrite(BLED, LOW);
  }
  // Синий
  else if (mode == 3)
  {
    digitalWrite(RLED, LOW);
```

Рисунок 22 – Листинг программы управляемый ночник на светодиоде

```
    digitalWrite(GLED, LOW);
    digitalWrite(BLED, HIGH);
  }
  // Пурпурный (Красный + Синий)
  else if (mode == 4)
  {
    analogWrite(RLED, 127);
    analogWrite(GLED, 0);
    analogWrite(BLED, 127);
  }
  // Бирюзовый (Синий + Зеленый)
  else if (mode == 5)
  {
    analogWrite(RLED, 0);
    analogWrite(GLED, 127);
    analogWrite(BLED, 127);
  }
  // Оранжевый (Зеленый + Красный)
  else if (mode == 6)
  {
    analogWrite(RLED, 127);
    analogWrite(GLED, 127);
    analogWrite(BLED, 0);
  }
  // Белый (Зеленый + Красный + Синий)
  else if (mode == 7)
  {
    analogWrite(RLED, 85);
    analogWrite(GLED, 85);
    analogWrite(BLED, 85);
  }
  // Выключен (mode = 0)
  else
  {
    digitalWrite(RLED, LOW);
    digitalWrite(GLED, LOW);
    digitalWrite(BLED, LOW);
  }
}

void loop()
{
  currentButton = debounce(lastButton);           // Чтение статуса
                                                  // кнопки
  if (lastButton == LOW && currentButton == HIGH) // Если нажата кнопка
  {
```

Рисунок 23 – Листинг программы управляемый ночник на светодиоде часть 2

```
    ledMode++;                                  // Инкремент
                                                // переменной
                                                // статуса светодиода
  }
  lastButton = currentButton;
                                        // Прошли по циклу все режимы
                                        // свечения светодиода
                                        // Сброс на начальный вариант =0
  if (ledMode == 8)
    ledMode = 0;
  setMode(ledMode);                      // Изменить режим светодиода
}
```

Рисунок 24 – Листинг программы управляемый ночник на светодиоде часть 3

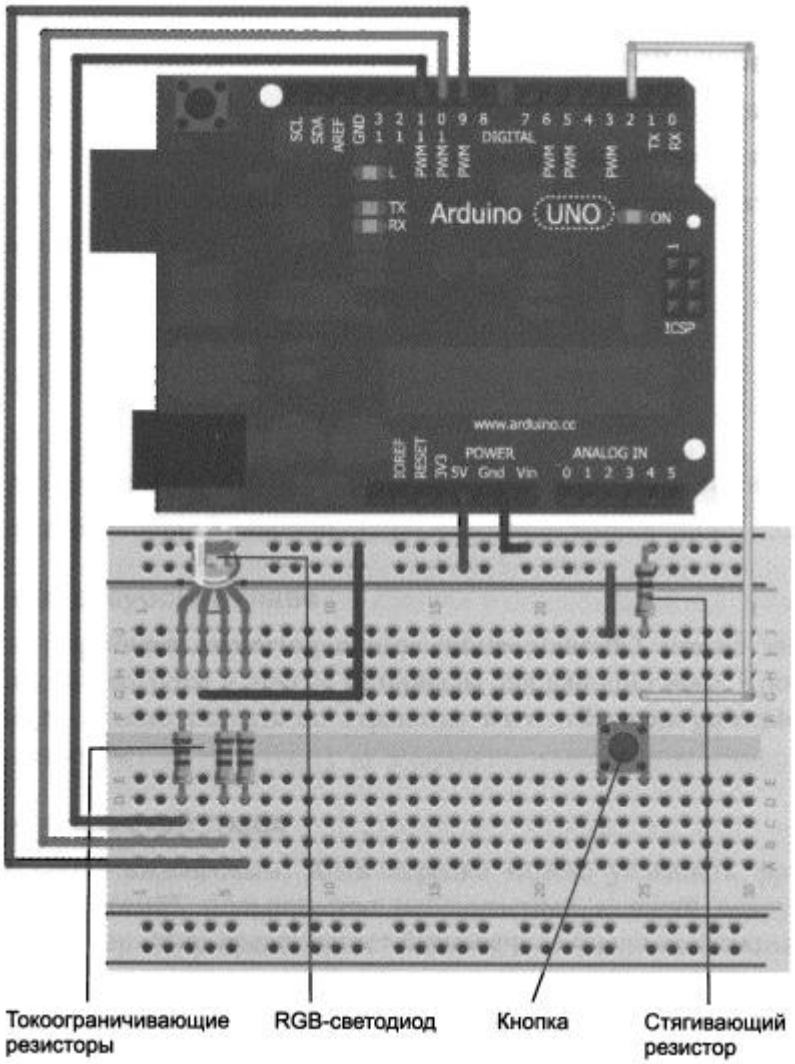

Рисунок 25 – Монтажная схема ночника

Контрольное задание:

Загрузите программу в плату и поэкспериментируйте с разноцветным ночником.

Самостоятельно измените этот проект. Например, добавить кнопки для управления каждым выводом RGB-светодиода. Или реализовать дополнительный режим мигания каждым цветом. Поменяйте цвет RGB-светодиода, изменив значения в функции analogWrite () на свои собственные.

БИОГРАФИЧЕСКИЙ СПИСОК

1 Джереми Блум. Изучаем Arduino: инструменты и методы технического волшебства: Пер. с англ. — СПб.:БХВ-Петербург, 2015. — 336.

2 С. Монк Программируемые Arduino. Профессиональная работа со скетчами – СПб.: Питер, 2017.

3 Петин В.А. Проекты с использованием контроллера Arduino. — СПб.: БХВ-Петербург, 2014. — 400 с.