12+

ПУТЬ К ПОНИМАНИЮ МИКРОКОНТРОЛЛЕРОВ: НА ОСНОВЕ ARDUINO

БЕЗ МАГИИ

```
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  delay(500);
  digitalWrite(13, LOW);
  delay(500);
}

// РЕГИСТРЫ AVR
DDRB |= (1<<5);  // PB5 -
PORTB |= (1<<5);  // PB5 -
TCCR0A = (1<<WGM01);
TCCR0B = (1<<CS01);
OCR0A = 124;
```

ОТ WIRING И ПЕРВЫХ СКЕТЧЕЙ – К ПОНИМАНИЮ МИКРОКОНТРОЛЛЕРОВ И ОСНОВ C/C++

ПРОГРАММИРОВАНИЕ
- Wiring и C/C++
- функции, циклы
- массивы, структуры
- классы и объекты

МИКРОКОНТРОЛЛЕРЫ
- регистры AVR
- таймеры
- память и биты

ВЗАИМОДЕЙСТВИЕ
- UART, I2C, SPI
- работа с датчиками
- обмен данными

ПРАКТИКА
- понятные примеры
- пошаговые объяснения
- исследовательские задания

НЕ КОПИРОВАТЬ СКЕТЧИ – А ПОНИМАТЬ, КАК РАБОТАЕТ УСТРОЙСТВО

ОТ НОВИЧКА К ИНЖЕНЕРУ

ПОНИМАТЬ, А НЕ ЗАУЧИВАТЬ

ПРАКТИКА, А НЕ ТЕОРИЯ РАДИ ТЕОРИИ

НИКОЛАЕВ ЕВГЕНИЙ

Евгений Николаев

Путь к пониманию микроконтроллеров: на основе Arduino

«Издательские решения»

Николаев Е.

Путь к пониманию микроконтроллеров: на основе Arduino /
Е. Николаев — «Издательские решения»,

Многие книги по Arduino показывают готовые решения. Эта книга объясняет, как они работают и почему работают именно так. Без перегруженной теории, но и без «магии», когда устройство работает, а почему — остаётся непонятным. Материал книги основан на многолетней инженерной и педагогической практике автора и многократно преподавался людям с разным уровнем подготовки: школьникам, студентам, механикам, электрикам, инженерам, разработчикам электроники и начинающим программистам.

Содержание

Путь к пониманию микроконтроллеров: на основе Arduino

Евгений Николаев

ISBN 978-5-0070-8049-1
Создано в интеллектуальной издательской системе Ridero

У автора более 10 лет практического опыта (работа на предприятиях, проектная деятельность, личные проекты, грант) и более 4 лет педагогической практики (институт, индивидуальное обучение, консультации).

Профессиональная деятельность включает разработку электронных устройств от проработки технического задания до разработки, изготовления, макетирования, регулировки и запуска серийного изготовления.

Окончил с отличием бакалавриат и магистратуру по направлению «Электроника и наноэлектроника». Окончил аспирантуру по направлению «Электроника, радиотехника и системы связи» с присвоением квалификации: «Исследователь. Преподаватель — исследователь».

Автор методических пособий, научных статей, патентов на полезные модели в области микроволновой электроники и множества инженерных проектов различного назначения.

Автор образовательных книг:

— «Основы разработки электронных устройств» (16+);

— Образовательной серии книг про электрончиков (обучение физике полупроводников для детей 2—6 лет через эмоции и добрую сказку).

О КНИГЕ

При создании этой книги не устала ни одна нейросеть. Как следствие не пострадали и цены на оперативную память. Зато пострадало немало чашек кофе. Книга написана автором по результатам его инженерной (более 10 лет) и педагогической (более 4 лет) деятельности.

Одной из распространённых проблем технического обучения является так называемая «слепота эксперта»: профессионалу многое кажется очевидным, тогда как для начинающего именно эти моменты становятся причиной непонимания, потери интереса и ощущения, что программирование — это слишком сложно.

В начале педагогической практики я столкнулся с этим и сам. Поэтому с первых занятий начал внимательно относиться к каждому вопросу ученика. Каждая мелочь, непонятная одному человеку, с большой вероятностью окажется сложной и для многих других. Со временем это привело к выстраиванию собственной траектории объяснения — такой, которая заранее закрывает типичные трудности понимания.

Материал книги многократно проверялся на практике: школьники, студенты, электрики, механики, инженеры, разработчики электроники и люди, которые никогда ранее не занимавшихся программированием.

Но эта книга — не просто про Arduino.

Arduino и язык Wiring здесь используются как понятная отправная точка для постепенного перехода к более глубокому пониманию программирования микроконтроллеров. Основная цель книги — построить педагогический мост между первыми скетчами и осознанным пониманием того, как работают микроконтроллеры, язык C/C++, регистры, обработка сигналов, интерфейсы связи и логика работы embedded-устройств.

Вместо принципа: «Сначала выучите сложную теорию — потом, возможно, поймёте практику» здесь используется другой подход: «Теория ради практики, а не теория ради теории».

Каждая новая тема появляется тогда, когда становится действительно нужна для решения практической задачи. От мигающего светодиода и первых условий — к обработке данных, проектированию логики устройства, взаимодействию микроконтроллеров и пониманию того, что скрывается за удобными функциями Arduino.

Что вас ждёт в книге?

Объяснение функций и синтаксиса простым языком.

Понимание, почему код устроен именно так.

Практические задачи для создания электронных устройств.

Постепенный переход от Wiring к первым осознанным прошивкам на C/C++.

Работа с логикой устройств, обработкой сигналов и интерфейсами связи.

Понимание того, как работает микроконтроллер «под капотом».

Эта книга подойдёт:

— тем, кто купил Arduino, помигал светодиодом и не знает, что и как делать дальше;

— начинающим разработчикам;

— студентам технических направлений;

— любителям электроники и embedded-разработки;

— тем, кто хочет перейти от копирования скетчей к пониманию того, как действительно работают Arduino и микроконтроллеры AVR.

Добро пожаловать в путь осознанного программирования!

1 От простого к сложному

Если последние подразделы первой главы кажутся сложными при первом изучении без опыта, то рекомендую перейти ко второй главе, а после его изучения вновь вернуться к первой.

1.1 Включение светодиода

Некоторые задачи можно выполнять как с помощью электроники (аппаратно), так и с помощью программирования (программным способом).

Когда мы программируем электронику, нам необходимо понимать физический принцип происходящего и уметь подключать радиоэлектронные компоненты или модули к Arduino.

В качестве первого практического задания всегда собирают электрическую цепь со светодиодом.

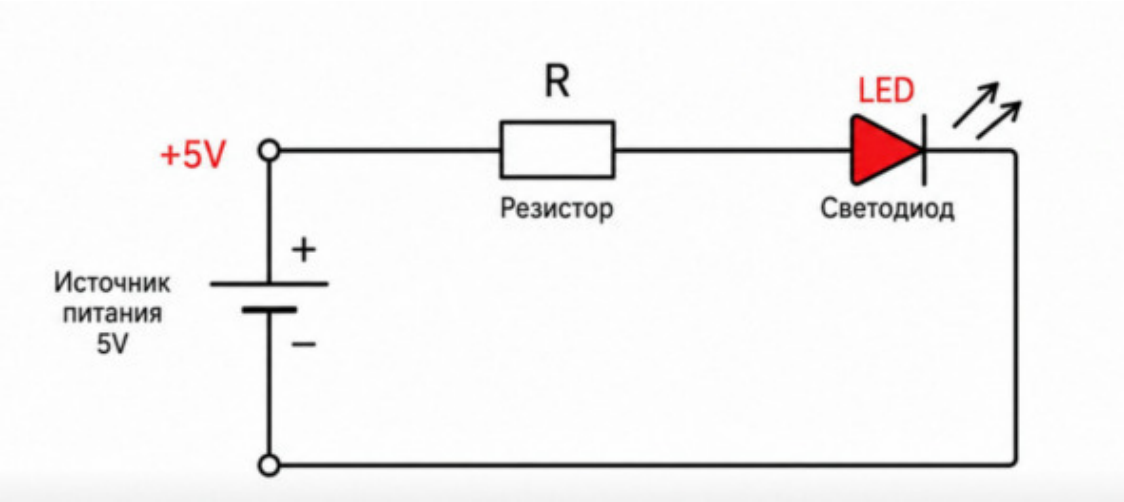
Светодиод должен светиться от постоянного источника питания.

Цель задания — подключить светодиод к постоянному источнику питания, используя токоограничивающий резистор (подробнее о резисторах в подразделе 7.1 этой книги), который нужно подключить между источником питания и анодом светодиода, катод следует подключить к минусу источника питания.

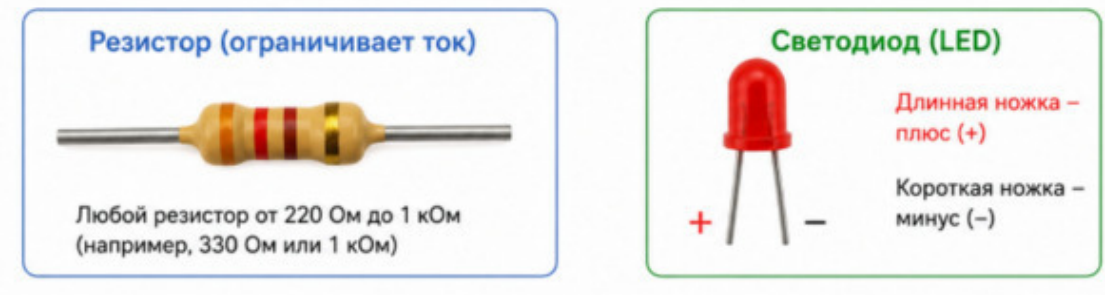
Таким образом у нас получается:

— плюс от источника питания -> вывод резистора -> (резистор) -> вывод резистора -> анод светодиода.

— минус источника питания -> катод светодиода.



Чтобы определить, где у светодиода катод, а где анод, можно обратить внимание на длину выводов (катод короче), на корпус светодиода (со стороны катода нижняя часть корпуса прямая, а не закругленная), посмотреть на металлические пластины внутри корпуса (электрод катода (-) имеет большую площадь) или воспользоваться мультиметром (измерительным прибором).



В качестве источника питания можно использовать специальный отсек с тремя батарейками AA (выдаст 4,5В), использовать крону (выдает 9В, поэтому резистор нужно подбирать сопротивлением больше) или Arduino (использовать вывод Arduino 5В, а Arduino подключить к компьютеру).

От теории к практике: включение светодиода
Подключаем светодиод к Arduino

Задача:
Необходимо написать скетч для постоянного включения светодиода.

Подготовка:
Если это ваше первое знакомство с Arduino, то рекомендую сначала ознакомиться с подразделом 8.1 и подразделом 8.3 этой книги.

Чтобы начать достаточно установить компьютерную программу Arduino IDE любой версии на компьютер (подробнее в подразделе 8.3).

Если хочется не только программировать, но и видеть результат — то нужно приобрести отладочную плату Arduino (это можно сделать и позже).

Далее по книге предполагается, что мы будем программировать отладочную плату Arduino Uno или Arduino Nano (можно купить любую из них, для учебы рекомендуется Arduino Uno). Купить можно на любом маркетплейсе (если это первая покупка — нужно выбирать плату с кабелем (домашний кабель от зарядки может не подойти).

Для программирования нужно установить на компьютер драйвер CH341.

Решение:
Когда мы программируем микроконтроллер для того, чтобы работало какое-либо электронное устройство, вначале необходимо разобраться, как оно должно работать. В данной задаче важно разобраться, что необходимо для свечения светодиода.

Светодиод излучает свет или светится, если на него подать напряжение. Значит, чтобы светодиод постоянно излучал свет, необходимо чтобы все время, пока работает Arduino, на него поступало напряжение, для этого мы можем подключить светодиод к одному из цифровых выводов (начиная со второго выхода, 2—13) или к одному из аналоговых выводов (отмечены на плате буквой A). Аналоговые выводы (A0-A5) могут использоваться, как цифровые (2—13).

Все выводы (или пины/pin) на Arduino по умолчанию находятся в режиме входа (*INPUT*), поэтому если мы используем вывод в режиме входа и не указали в разделе *void setup* команду *pinMode (pin, INPUT)* в некоторых случаях скетч все равно будет работать, но для стабильной работы рекомендуется не забывать указывать состояние вывода. Если вывод используется с подтягивающим резистором *(INPUT_PULLUP)* или в режиме выхода (*OUTPUT*), то функцию указывать необходимо обязательно.

Таким образом, если не указать режим вывода, он будет находиться в режиме входа (*INPUT*), и если в скетче на него подавать напряжение, то есть использовать его в режиме выхода (*OUTPUT*), то это может в некоторых случаях привести к повреждению Arduino (при превышении допустимых токов).

Поэтому если мы используем пин Arduino, то обязательно используем функцию *pinMode (_,_)*, где в скобках до запятой указывается номер пина Arduino, а после запятой его режим работы.

Вместо «*INPUT*» можно писать «0», потому что внутри микроконтроллера число «0» означает, что пин будет работать как вход, но обычно используют «*INPUT*», чтобы код был понятнее и проще для чтения.

То есть «*pinMode (2,0);*» и «*pinMode (2,INPUT);*» — это одно и то же.

«*OUTPUT*» соответствует «1», то есть «*pinMode (2,1);*» и «*pinMode (2,OUTPUT);*» — это одно и то же.

Допустим, что мы решили подсоединить светодиод к третьему пину Arduino, тогда функция установки режима пина будет иметь следующий вид: *pinMode (3, OUTPUT);*.

Каждая строчка заканчивается точкой с запятой. Точка с запятой позволяют компилятору определять, где оканчивается строка, поэтому это важно и без этого символа компилятор выдаст ошибку. Так же важно соблюдать регистр букв, если написать функцию только заглавными буквами, или только маленькими, или любым иным образом, кроме того, который указан выше — компилятор эту функцию не распознает и не сможет перевести код в понятный микроконтроллеру язык из нулей и единиц, в результате чего компилятор сообщит об ошибке.

Для того, чтобы подать напряжение на пин, существует функция *digitalWrite (_, _)*, в круглых скобках которой до запятой указывается номер пина, а после его логическое состояние.

Здесь так же вместо *HIGH* (высокого логического сигнала, наличия напряжения) можно написать 1, а вместо *LOW* (низкого логического сигнала, отсутствия напряжения) можно написать 0.

```
void setup () {
pinMode (3, OUTPUT);
digitalWrite (3, HIGH);
}
void loop () {
}
```

В Wiring (языке программирования на котором программируют в программе Arduino IDE) есть два основных блока программы, это *void setup () {}* и *void loop () {}*.

void setup () {} — всегда выполняется только один раз после того, как на отладочную плату поступит питание. Внутри фигурных скобок « {}» пишется та часть кода, которая должна выполняться один раз. « {» — называют открывающей фигурной скобкой, а} — называют закрывающей фигурной скобкой.

С помощью этих скобок компилятор понимает, где границы функции *void setup ()*. Чтобы микроконтроллер снова выполнил написанный в *void setup ()* код, необходимо отключить от отладочной платы Arduino питание и подключить заново, либо нажать кнопку на отладочной плате Arduino, которая используется для перезагрузки. Так как код в *void setup* исполняется один раз, здесь размещают, например, настройки портов. Команда *pinMode (_, _)* всегда прописывается в фигурных скобках *void setup (),* но может быть записана и в *void loop ()*.

В *void setup ()* ее традиционно пишут потому, что даже после того, как микроконтроллер начнет выполнять код в void loop (), вывод будет выходом или входом (в зависимости от того, как мы его назначили). А назначать вывод выходом/входом несколько раз повторно (в случае с размещением в *void loop ()* — бессмысленно.

Так же нам никто не запрещает размещать тут и другие команды, например, написать здесь «*digitalWrite (3, HIGH);*». Получается, что микроконтроллер включается, устанавливает третий вывод в режиме выхода и подает на неё высокий уровень логического напряжения, после этого он переходит в блок *void loop (),* но так, как там в фигурных скобках пусто — он больше ничего не делает, при этом продолжает подавать на третий вывод (ножку) высокое логическое напряжение.

void loop () в отличие от *void setup ()* выполняется многократно. Всю программу микроконтроллер выполняет сверху вниз, дойдя до *void loop ()* он выполняет первую строку внутри фигурных скобок этого блока, потом, если она есть, вторую, если есть третья, то третью, и так далее, когда строки с командами закончатся и микроконтроллер встретит закрывающую фигурную скобку, он перейдет к открывающей фигурной скобке и снова начнет выполнять первую после неё строку, потом вторую, третью, и так до конца. Получается, что то, что написано в *void loop ()* внутри фигурных скобок выполняется по кругу сверху вниз бесконечно, пока на отладочную плату Arduino поступает напряжение питания.

Даже если внутри фигурных скобок *void setup ()* или *void loop ()* ничего не написано, эту функцию обязательно нужно оставлять, иначе вы не сможете откомпилировать и загрузить в Arduino ваш код.

В примере выше это хорошо представлено. Блок программы *void loop ()* не используется, внутри фигурных скобок нет команд, но эта часть программы всё равно присутствует.

***** Если вы впервые знакомитесь с Arduino и впервые читаете книгу — рекомендую пропустить объяснение ниже и перейти сразу к окончанию этого подраздела — к «задачи для закрепления материала», а к тексту ниже вернуться после изучения 1 — 3 разделов.**

В микроконтроллере есть специальные ячейки памяти, которые содержат его настройки и называются регистрами. Каждый регистр выполняет свою задачу, так вместо *pinMode ()* в коде используется регистр *DDR* (подробнее в подразделе 3.4 этой книги), вместо *digitalWrite ()* используется регистр *PORT* (подробнее в подразделе 3.5 этой книги).

3 вывод Arduino относится к порту D, поэтому команды будут *DDRD* и *PORTD*. Чтобы сделать вывод выходом и подать на него напряжение (*HIGH* или логическая единица) нужно установить единицу в соответствующие биты этих регистров. Для этого пишем логическую единицу, побитовый сдвиг "<<" и номер бита порта D. Оператор "|=" изменяет только нужный бит, не затрагивая остальные.

```
void setup () {
DDRD |= (1 <<3);
PORTD |= (1 <<3);
}
void loop () {
}
```

Запись (1 <<3) говорит о том, что единица сдвигается в позицию третьего бита (или, если упростить, мы записываем в третий бит порта D логическую единицу).

Чтобы включить два светодиода, каждый из которых не будет зависеть от другого, необходимо подключить светодиоды к разным выводам и подать напряжение на каждый из них.

```
void setup () {
pinMode (3, OUTPUT);
pinMode (4, OUTPUT);
```

```
digitalWrite (3, HIGH);
digitalWrite (4, HIGH);
}
void loop () {
}
```

Скетч с использованием регистров:

```
void setup () {
DDRD |= (1 <<3);
DDRD |= (1 <<4);
PORTD |= (1 <<3);
PORTD |= (1 <<4);
}
void loop () {
}
```

Задачи для закрепления материала:

Задачи следует решать с использованием функций Wiring.

(Дополнительно) Решение задач с использованием регистров.

1. Подключить три светодиода на 2, 3 и 4 выводах Arduino и подать на них напряжение.

2. Подключить восемь светодиодов. Подать на них напряжение. Выводы Arduino выбрать самостоятельно.

Что мы изучили в подразделе 1.1:

1. Принцип подключения светодиода;

2. Назначение токоограничивающего резистора (подробнее о резисторах в подразделе 7.1);

3. Понятия анод и катод;

4. Функцию *pinMode ()* — настройка режима вывода;

5. Функцию *digitalWrite ()* — подача логического уровня;

6. Значения HIGH и LOW;

7. Назначение *void setup ();*

8. Основы управления выводами через регистры DDRD, PORTD (подробнее в подразделах 3.4 и 3.5).

1.2 Мигание светодиода
Включение и выключение светодиода

Классическая пример blink (мигание светодиодом) часто недооценивают.

Мигание светодиодом — это последовательное изменение состояния вывода Arduino (в отдельном случае — микроконтроллера): включение и выключение напряжения.

Включение и выключение напряжения актуально для использования светодиодов (индикация ошибок, загрузке и другой информации, визуальные эффекты), реле и транзисторных переключателей (управление нагрузкой), звукоизлучающих устройств и других.

Многие устройства управляются подачей напряжения на управляющий вывод.

Включением напряжения запускаются стабилизаторы напряжения (подача напряжения на разрешающий вывод Enable).

Вопреки частому мнению мигание светодиодом может быть целым искусством и иметь множество интересных решений.

Классическое решение — это постоянное мигание светодиодом, которое представлено в примерах Arduino IDE, как blink:

```
void setup () {
pinMode (13, OUTPUT);
}
void loop () {
digitalWrite (13, HIGH);
delay (300);
digitalWrite (13, LOW);
delay (300);
}
```

На Arduino Nano и Uno к 13 выводу уже подключен светодиод на отладочной плате с маркировкой «L» (это не мешает подключить к 13 выводу еще один светодиод или любое другое устройство).

Тот же код можно записать иначе (без смены подхода):

```
void setup () {
pinMode (13, 1);
}
void loop () {
digitalWrite (13, 1);
delay (300);
digitalWrite (13, 0);
delay (300);
}
```

Любопытным для многих может быть тот факт, что классическая форма — функция *pinMode ()* в *void setup (),* а *digitalWrite ()* в *void loop ()* не является единственным вариантом, так мы можем прописать назначения вывода выходом внутри *void loop ():*

```
void setup () {
}
void loop () {
pinMode (13, 1);
digitalWrite (13, 1);
delay (300);
digitalWrite (13, 0);
```

```
delay (300);
}
```

Почему тогда «привычная» для многих форма выглядит, как предыдущий вариант? Дело в том, что для большинства функций не имеет значение, где мы их запишем, в *void loop ()* или *void setup ()*. Но действовать нужно согласно логике, если нам достаточно **один** раз назначить вывод выходом и всё, то нет смысла писать *pinMode (13, 1);* в *void loop (),* избыточно расходуя ресурсы микроконтроллера, заставляя его делать одно и то же действие многократно, хотя достаточно сделать его один раз.

В то же время, если мы хотим, чтобы светодиод моргнул всего лишь один раз (частный случай), то код может быть таким:

```
void setup () {
pinMode (13, 1);
digitalWrite (13, 1);
delay (300);
digitalWrite (13, 0);
}
void loop () {
}
```

Вернёмся к многократному миганию светодиодом и добавим немного автоматизации, используя *#define* для вывода Arduino и переменную *delTime* типа данных *int* для значения времени задержки.

Для вывода 13 в Wiring для Arduino уже существует зарезервированное имя (а именно *LED_BUILTIN*, но нам никто не запретит создать новое, тем более используя *#define* мы не тратим память микроконтроллера.

```
#define pinLed 13
int delTime = 300;
void setup () {
pinMode (pinLed, 1);
}
void loop () {
digitalWrite (pinLed, 1);
delay (delTime);
digitalWrite (pinLed, 0);
delay (delTime);
}
```

Вспомним, что вместо конкретного состояния (логическая единица или логический ноль) мы можем прописать переменную, которая в свою очередь уже будет содержать нужный логический уровень.

```
#define pinLed 13
int delTime = 300;
char state = 0;
void setup () {
pinMode (pinLed, 1);
}
void loop () {
digitalWrite (pinLed, state);
delay (delTime);
state = 1;
digitalWrite (pinLed, state);
```

```
delay (delTime);
state = 0;
}
```

Так как новая переменная может принять только два состояния (логическая единица или логический ноль, согласно логике кода), то вместо типа данных char (или иного) логичней использовать bool *(подробнее в подразделе 8.6),* тогда мы получим следующий вариант кода:

```
#define pinLed 13
int delTime = 300;
bool state = LOW;
void setup () {
pinMode (pinLed, 1);
}
void loop () {
digitalWrite (pinLed, state);
delay (delTime);
state = HIGH;
digitalWrite (pinLed, state);
delay (delTime);
state = LOW;
}
```

Теперь стоит обратить внимание на то, что у нас много строчек кода. Введенные дополнительные строчки увеличивают время работы микроконтроллера, поэтому следующим логичным шагом будет следующая оптимизация:

```
#define pinLed 13
int delTime = 300;
bool state = LOW;
void setup () {
pinMode (pinLed, 1);
}
void loop () {
digitalWrite (pinLed,!state);
delay (delTime);
}
```

Инверсия имеет место быть благодаря тому, что *bool* может быть только 0 или 1, что нам и требуется. Инверсия 0 даст нам 1, инверсия 1 даст нам 0.

Так уже значительно лучше, код стал компактнее и читается проще. В некоторых случаях это также может уменьшить объём машинного кода после компиляции (перевода написанного кода в понятный микроконтроллеру).

Но в этом варианте всё еще есть лишняя часть в виде переменной, поэтому следующим шагом нам нужно уйти от переменной state.

Уход от переменной, при сохранении общего вида кода возможен благодаря функции *digitalRead ().*

Функция «цифрового» чтения напряжения на выводе Arduino может использоваться и для ввода, и для вывода.

Если вывод Arduino назначен выводом, то мы с помощью функции *digitalRead ()* сможем узнать логический уровень напряжения в данный момент.

Если вывод Arduino назначен выходом, то мы можем использовать эту функции, но с помощью нее узнаем не тот уровень напряжения, который там действительно есть, а тот, который там должен быть в данный момент (стоит учитывать, что если всё работает как нужно,

то реальный логический уровень напряжения совпадает с тем, который должен быть в тот же момент времени).

```
#define pinLed 13
int delTime = 300;
void setup () {
pinMode (pinLed, 1);
}
void loop () {
digitalWrite (pinLed,!digitalRead (pinLed));
delay (delTime);
}
```

Нет никакой официальной классификации, но лично для себя во время своей педагогической практики (занятий в институте и индивидуальных уроков) я установил, что здесь, в рамках этой задачи, заканчивается базовый уровень программирования Arduino.

Это все возможные вариации на Wiring для решения этой задачи.

***** Если вы впервые знакомитесь с Arduino и впервые читаете книгу — рекомендую пропустить объяснение ниже и перейти сразу к окончанию этого подраздела — к «задачи для закрепления материала», а к тексту ниже вернуться после изучения 1 — 3 разделов.**

Между тем, Arduino IDE позволяет писать не только исключительно на Wiring, но и используя язык программирования C. Возможность эта остается благодаря тому, что в языке Wiring все функции представляют из себя совокупность команд на языке C и если мы пишем на C, то упрощаем задачу компилятору, а все эти громоздкие функции не загружаются в микроконтроллер на Arduino.

Тогда используя регистры мигание светодиодом будет выглядеть по-другому:

```
int delTime = 300;
void setup () {
DDRB |= (1 <<PB5); // назначили выходом
}
void loop () {
PORTB |= (1 <<PB5); //включили светодиод
delay (delTime);
PORTB &= ~ (1 <<PB5); //выключили светодиод
delay (delTime);
}
```

Вот только выглядит скетч так, будто мы снова всё усложняем, даже не смотря на то, что все эти команды выполняются значительно быстрее, чем даже одна единственная функция *digitalWrite ()*.

Решение является простым:

```
int delTime = 300;
void setup () {
DDRB |= (1 <<PB5);
}
void loop () {
PORTB ^= (1 <<PB5);
/* аналог digitalWrite (pinLed,!digitalRead (pinLed));
```

*Оператор ^= выполняет побитовое исключающее ИЛИ (XOR). Если бит был 0, он станет 1; если был 1 — станет 0. Поэтому светодиод переключается между включением и выключением. */*

delay (delTime);

}

Учитывая то, что функция задержки у нас осталась в коде одна, то переменную можно исключить. Код в таком случае примет следующий вид:

```
void setup () {
DDRB |= (1 <<PB5);
}
void loop () {
PORTB ^= (1 <<PB5);
delay (300);
}
```

От Wiring уже осталось не так много и всматриваясь в код мы видим конструкции *void setup ()* и *void loop (),* а так же функцию *delay ().*

От конструкций *void setup ()* и *void loop ()* мы уйти не можем, эти конструкции являются ядром языка Wiring и обязательно должны быть в скетче в единичном экземпляре, чтобы код скомпилировался.

А вот функцию *delay* можно заменить функцией из языка С, но тогда нам нужно обозначить значение частоты кварцевого резонатора (внешнего или встроенного в микроконтроллер) не меняя название F_CPU и подключить библиотеку util/delay. h.

В результате всех изменений код будет выглядеть следующим образом:

```
#define F_CPU 16000000UL
#include <util/delay. h>
void setup () {
DDRB |= (1 <<PB5);
}
void loop () {
PORTB ^= (1 <<PB5);
_delay_ms (300);
}
```

Код выше мог бы выглядеть и проще:

```
#include <util/delay. h>
void setup () {
DDRB |= (1 <<PB5);
}
void loop () {
PORTB ^= (1 <<PB5);
_delay_ms (300);
}
```

Значение F_CPU в таком случае будет взято компилятором на основе стандартного кварцевого резонатора для выбранной платы (Мы всегда в Arduino IDE выбираем конкретную модель платы (например Arduino UNO) и номер COM — порта. Так для Arduino UNO стандартный кварцевый резонатор — 16 МГц).

Постараемся уйти и от *void setup ()* и *void loop ()*, тогда код мигания светодиодом будет выглядеть следующим образом:

```
#define F_CPU 16000000UL
#include <util/delay. h>
```

```
#include <avr/io. h>
int main (void) {
DDRB |= (1 <<PB5);
while (1) {
PORTB ^= (1 <<PB5);
_delay_ms (300);
}
}
```

Последний вариант — это прошивка на языке программирования С, для бесконечного (пока на микроконтроллер поступает питание) мигания светодиодом.

На этом моменте я вас поздравляю с первой прошивкой микроконтроллера на языке С!

Это не просто прошивка, которую вы скопировали, это прошивка, в которой каждую строчку, каждую команду вы понимаете. Вы знаете откуда каждая команда здесь появилась, зачем она нужна и почему всего этого достаточно для решения поставленной задачи.

Если вы начали обучение по этой книге, то это ваша первая осознанная программа на языке С для управления микроконтроллером.

Задачи для закрепления материала:

Задачи следует решать с использованием функций Wiring.

(Дополнительно) Решение задач с использованием регистров.

1. Постоянное одновременное мигание двумя светодиодами (мигание считается одновременным, если после задержки включаются все светодиоды, а потом после задержки все светодиоды выключаются).

2. Постоянное одновременное мигание тремя светодиодами.

3. Мигание сначала одним светодиодом, потом другим. Мигание каждым светодиодом осуществляется только один раз за всю работу Arduino.

* Мигание сначала одним светодиодом, потом другим. Мигание каждым светодиодом осуществляется два раза за всю работу Arduino.

* Одновременное мигание двумя светодиодами. Мигание светодиодами осуществляется два раза за всю работу Arduino.

Что мы изучили в подразделе 1.2:

1. Принцип мигания светодиода;
2. Функцию *delay ()* — задержка выполнения программы;
3. Назначение *void loop ();*
4. Логическое переключение состояния вывода;
5. Использование переменных для хранения состояния;
6. Функцию digitalRead ();
7. Основы оптимизации кода;
8. Переключение состояния вывода через (^=) в регистрах (подробнее в подразделе 3.3).

1.3 Мигание светодиода во время нажатия тактовой кнопки Включение и выключение светодиода только при определенном условии

Одним из способов выполнять действия, пока соблюдается некоторое условие, является цикл *while () {}* (подробное описание цикла *while ()* в подразделе 2.2.1).

while с английского переводится, как «пока» (некий процесс, во время). Как правильно прочитать строчку «*while () {}*»? **Пока** некоторое условие (событие) (в круглых скобках) происходит нужно делать следующее (то, что в фигурных скобках). В круглых скобках условие, а в фигурных скобках функционал.

Когда микроконтроллер переходит на строчку с циклом while он проверяет условие в круглых скобках. Если условие не выполняется, то микроконтроллер внутрь фигурных скобках не входит, все действия внутри фигурных скобок пропускает и перемещается на строку с закрывающей фигурной скобкой и идет дальше на следующую строку кода.

Другая ситуация — микроконтроллер добрался до проверки условия, до круглых скобок цикла *while* и условие оказалось истиной, то есть условие выполняется. Тогда микроконтроллер входит в фигурные скобки и выполняет необходимый нам функционал — выполняет команды внутри фигурных скобок цикла *while* сверху вниз. После того, как достигает закрывающую фигурную скобку} возвращается к проверке условия — в круглые скобки. Алгоритм повторяется вновь, если условие не выполняется — микроконтроллер выходит из цикла и переходит на строчку, которая следует за закрывающей фигурной скобкой}. Если выполняется, то проходит строки в фигурных скобках и снова повторяется проверка условия в круглых скобках. Количество таких повторений ничем не ограничивается. Действия выполняются пока выполняется условие.

Необходимо подключить светодиод и тактовую кнопку. Мы уже выяснили, что вывод Arduino с подключенным светодиодом нужно назначить выходом (*OUTPUT*). А как быть с тактовой кнопкой?

Есть два способа подключения тактовой кнопки к Arduino. Один из них интуитивно понятен — подключение кнопки между питанием и цифровым выводом. Тактовая кнопка отличается от любой другой тем, что она не фиксируется в нажатом положении, стоит убрать с неё палец, как она возвращается в прежнее положение. Во время нажатия она замыкает свои выводы, тем самым соединяя электрическую цепь. Так один её вывод подключаем к 5V Arduino, а второй вывод к одному из цифровых выводов Arduino (от 2 до 13 включительно). Допустим, мы будем использовать 2 вывод, тогда в тот момент, когда мы нажмем на тактовую кнопку, 5В через кнопку поступит на цифровой вывод 2 и мы сможем на нем прочитать наличие высокого логического уровня напряжения. Если мы собираемся читать наличие напряжения, и так как напряжение поступает с вывода 5В на вывод 2 Arduino, то для микроконтроллера это входящая информация, а значит вывод надо назначить входом (*INPUT*).

Получаем следующий void setup:

```
void setup () {
pinMode (2, INPUT); // вывод с тактовой кнопкой — вход
pinMode (13, OUTPUT); // вывод с светодиодом — выход
}
```

Нам нужно проверить нажата ли кнопка, для этого нужно прочитать напряжение, для этого используется функция *digitalRead (pin);,* которая дословно переводится, как цифровое чтение, то есть мы собираемся прочитать логический ноль или логическую единицу с вывода.

В круглых скобках необходимо указывать номер вывода (или переменная его содержащая), с которого мы хотим прочитать логический уровень напряжения. Функция позволяет прочитать, но никуда не сохраняет, поэтому результат нужно записать в переменную. Это как с часами на телефоне. Возможно у вас была следующая ситуация: мы достаем телефон из кармана, смотрим на время, и убираем его обратно, а время не помним, хоть только что и смотрели. Почему? Потому что необходимо увиденное время запомнить. Так и микроконтроллеру необходимо запомнить прочитанное время, а для этого нам нужно записать значение в переменную. Нужно определить тип данных этой переменной. Мы будем записывать либо HIGH, либо LOW, а значит мы можем выбрать тип данных логический — *bool.*

(Далее используется цикл while () — подробнее о цикле в подразделе 2.2.1 этой книги).

Тогда получаем следующую часть кода (внимание! код промежуточный):

```
void loop () {
bool button = digitalRead (2);
while (button == HIGH) {
digitalWrite (13, HIGH);
}
}
```

Уже хорошо, во время нажатия тактовой кнопки включится светодиод, но не хватает деталей. При таком скетче светодиод включится только один раз, последующие нажатия кнопки будут просто бессмысленными (светодиод будет светиться постоянно), поэтому за фигурными скобками цикла нужно выключить светодиод:

```
void loop () {
bool button = digitalRead (2);
while (button == HIGH) {
digitalWrite (13, HIGH);
}
digitalWrite (13, LOW);
}
```

Обратите внимание, что мы прочитали логический уровень напряжения над циклом, а после смотрим, равна ли переменная button — HIGH, вот только мы в нее после этого ничего не записываем, а значит скетч до сих пор будет работать полноценно только один раз. Чтобы исправить это нужно прочитать напряжение внутри цикла, чтобы при повторной проверке у переменной было актуальное значение.

```
void loop () {
bool button = digitalRead (2);
while (button == HIGH) {
digitalWrite (13, HIGH);
button = digitalRead (2);
}
digitalWrite (13, LOW);
}
```

Если мы отпустим кнопку быстро, то результат работы скетча мы не увидим, поэтому следует добавить задержку в цикле после подачи напряжения и *void loop ()* будет выглядеть следующим образом:

```
void loop () {
bool button = digitalRead (2);
while (button == HIGH) {
digitalWrite (13, HIGH);
button = digitalRead (2);
```

```
delay (100);
}
digitalWrite (13, LOW);
}
```

Теперь мы уже получили работающий скетч, но его можно упростить, без потери работоспособности. Внутри круглых скобок цикла можно прописать функцию цифрового чтения и тогда, во время каждой проверки, в скобках будет актуальное значение, кроме того в таком случае мы исключаем одну переменную. Если в круглых скобках мы не напишем условие, а только *digitalRead (),* то условие будет считать истинным тогда, когда прочитанное будет HIGH. Скетч в таком случае можно написать проще:

```
void setup () {
pinMode (2, INPUT); // вывод с тактовой кнопкой — вход
pinMode (13, OUTPUT); // вывод с светодиодом — выход
}

void loop () {
while (digitalRead (2)) {
digitalWrite (13, HIGH);
delay (50);
}
digitalWrite (13, LOW);
}
```

Улучшить код можно введя переменные вместо номеров выводов (2 и 13), а так же поменяв *INPUT* на *INPUT_PULLUP*.

Если кнопка подключена без подтягивающего резистора, вход может «плавать» и случайно менять состояние. Отдельная проблема — дребезг контактов, возникающий в момент механического замыкания кнопки. Дребезг контактов — кратковременные ложные переключения во время нажатия и отпускания кнопки. Реже всего это проявляется, когда тактовая кнопка подключена к Arduino с использованием макетной платы (специальной платы, позволяющей подключать электронику без использования паяльника), чаще, когда кнопку подключают к Arduino через провода с использованием паяльника или используют уже в готовом устройстве.

Чтобы такого избежать используют следующие методы: программная фильтрация (проверять как долго сохраняется на выводе логическая единица, основываясь на разумной скорости человеческого нажатия, во время дребезга контактов, как правило, логическая единица исчезает мгновенно) или добавление подтягивающего резистора к питанию или земле.

Подключение такого резистора называют «подтяжкой к питанию», она может быть реализована внешним резистором (аппаратный способ), а может внутренним, встроенным в микроконтроллер (программный способ).

В микроконтроллере к каждому выводу относится свой встроенный резистор, который можно программно (через код) подключить к питанию, если вывод мы будем использовать, как вход. Для подключения встроенного резистора достаточно и необходимо вместо *INPUT* использовать *INPUT_PULLUP*. Следует учитывать, что при использовании *INPUT_PULLUP* логика инвертируется: нажатие будет соответствовать *LOW*, а отпускание *HIGH*.

Скетч с описанными улучшениями примет следующий вид:

```
#define button 2
#define led1 13
```

```
void setup () {
pinMode (button, INPUT_PULLUP); // вывод с тактовой кнопкой — вход
pinMode (led1, OUTPUT); // вывод с светодиодом — выход
}
void loop () {
while (digitalRead (button) ==LOW) {
digitalWrite (led1, HIGH);
delay (50);
}
digitalWrite (led1, LOW);
}
```

***** Если вы впервые знакомитесь с Arduino и впервые читаете книгу — рекомендую пропустить объяснение ниже и перейти сразу к окончанию этого подраздела — к «задачи для закрепления материала», а к тексту ниже вернуться после изучения 1 — 3 разделов.**

Следующим шагом является применение регистров вместо функций языка Wiring, для этого используем уже знакомые нам регистры для установки состояния вывода DDR, включения и выключения напряжения PORT. Для чтения напряжения используется регистр PIN (подробнее в подразделе 3.6 этой книги). При использовании регистров скетч примет следующий вид:

```
void setup () {
DDRD &=~ (1 <<2); //вывод с тактовой кнопкой — вход
PORTD |= (1 <<2); // подключение подтягивающего резистора (pull-up)
DDRB |= (1 <<5); // вывод с светодиодом — выход
}
void loop () {
while (! (PIND & (1 <<2))) {//пока кнопка нажата
PORTB |= (1 <<5); // включение светодиода
delay (50);
}
PORTB &=~ (1 <<5); // выключение светодиода
}
```

Равноценной записью будет такой же скетч, но записью PD2 вместо 2 и PB5 вместо 5 для улучшения читаемости.

Задачи для закрепления материала:

Задачи следует решать с использованием функций Wiring.

(Дополнительно) Решение задач с использованием регистров.

1. Выключение светодиода пока нажата тактовая кнопка, включение светодиода пока кнопка не нажата.

2. Мигание светодиодом пока тактовая кнопка нажата.

3. Мигание светодиодом пока нажата одна из двух тактовых кнопок.

4. Мигание светодиодом пока нажаты обе тактовые кнопки.

5. Три тактовые кнопки и три светодиода. Пока нажатая первая тактовая кнопка — светится первый светодиод, пока нажата вторая тактовая кнопка — светится второй светодиод, пока нажата третья тактовая кнопка — светится третий светодиод.

6. Три тактовые кнопки и три светодиода. Пока нажата первая кнопка — три светодиода светятся. Пока нажата вторая кнопка — три светодиода мигают одновременно. Пока нажата третья кнопка — три светодиода мигают по очереди.

Что мы изучили в подразделе 1.3:

1. Работу тактовой кнопки;
2. Цикл *while ()* (подробнее в подразделе 2.2.1);
3. Чтение состояния вывода через digitalRead ();
4. Логические условия (==, HIGH, LOW);
5. Режим *INPUT_PULLUP*;
6. Основы работы с кнопками и условиями.

1.4 Мигание светодиода с плавным изменением яркости Плавное включение и выключение светодиода

Мигание светодиода состоит из включения и выключения, рассмотрим детальнее каждый обозначенный процесс.

Когда мы подключаем светодиод к Arduino и используем функцию *digitalWrite* мы можем либо включить его, либо выключить. Яркость светодиода при подаче на него напряжения будет максимальной и ограничена только параметрами светодиода и электрической схемы.

Соответственно, мы можем видеть его в один момент выключенным, а в другой уже включенным на максимальной яркости. Реализовать плавное включение светодиода можно обеспечив подачу нужного напряжения в несколько этапов *(на самом деле это происходит по-другому и обеспечивается широтно-импульсной модуляцией (ШИМ), но в качестве упрощения можно представлять процесс таким образом)*. То есть не сразу подавать 5 В, а сначала 1 В, потом 2 В, 3 В, 4 В и уже только после этого 5 В или с любым другим шагом изменения напряжения. В результате мы будем видеть, как светодиод плавно «разгорается» до полной яркости.

Такую подачу напряжения можно обеспечить с помощью радиоэлектронных компонентов, а можно запрограммировать микроконтроллер, в нашем случае — Arduino.

В Arduino для таких задач есть специальные выводы, только на них можно реализовать такую задачу. Это цифровые выводы с ШИМ (широтно-импульсной модуляцией), которые позволяют подавать не просто логический ноль или логическую единицу, но и другие уровни напряжения *(для упрощения понимания, на самом деле позволяют изменять мощность сигнала).* Такие выводы на плате обозначены символом тильда (~). На Arduino Nano и Arduino UNO таких выводов шесть (3, 5, 6, 9, 10, 11), они являются цифровыми. То есть их можно использовать, как обычные цифровые выводы (читать с них логическую единицу или ноль (для подключения тактовых кнопок, цифровых датчиков…), подавать логическую единицу или ноль (для подключения светодиодов, реле…)), а можно, как выводы с ШИМ.

Для работы с такими выводами в Wiring существует команда *analogWrite (pin, state);*

Где pin — номер вывода Arduino, state — значение ШИМ от 0 до 255. Где 0 — соответствует полностью выключенному сигналу (логический ноль поданный функцией *digitalWrite*, светодиод выключен), а 255 — постоянно включенному (логическая единица поданная функцией *digitalWrite*, максимальная яркость для светодиода).

Обеспечим плавное включение светодиода подключенного к выводу с ШИМ.

```
void setup () {
pinMode (3, OUTPUT);
}
void loop () {
analogWrite (3,0);
delay (100);
analogWrite (3,50);
delay (100);
analogWrite (3,100);
delay (100);
analogWrite (3,150);
delay (100);
analogWrite (3,200);
delay (100);
analogWrite (3,255);
delay (100);
```

```
}
```

Всё хорошо, но если мы захотим шаг не 50, а 1, то количество строк увеличивается в разы и скетч станет похож на безумство!

Увеличение яркости светодиода состоит из повторяющихся команд *analogWrite* и *delay*. Для выполнения повторяющихся действий нужное количество раз используется цикл *for* (подробное описание цикла в подразделе 2.2.3).

```
void setup () {
pinMode (3, OUTPUT);
}
void loop () {
for (int i=0; i <=255; i = i+1) {
analogWrite (3,i);
delay (10);
}
}
```

Теперь светодиод будет включаться плавно с шагом равным 1.

Для плавного выключения светодиода нужно действовать в обратном порядке, задавая яркость от 255 до 0 с нужным шагом.

```
for (int i=255; i> = 0; i = i-1) {
analogWrite (3,i);
delay (10);
}
```

Для плавного включения и последующего плавного выключения светодиода следует расположить второй цикл сразу после первого. Если необходимо, чтобы светодиод был дольше включен или дольше выключен — после цикла, вне его фигурных скобок, нужно использовать delay. Пример кода ниже:

```
void setup () {
pinMode (3, OUTPUT);
}
void loop () {
for (int i=0; i <=255; i = i+1) {
analogWrite (3,i);
delay (10);
}
delay (100);
for (int i=255; i> = 0; i = i-1) {
analogWrite (3,i);
delay (10);
}
delay (100);
}
```

***** Если вы впервые знакомитесь с Arduino и впервые читаете книгу — рекомендую пропустить объяснение ниже и перейти сразу к окончанию этого подраздела — к «задачи для закрепления материала», а к тексту ниже вернуться после изучения 1 — 3 разделов.**

Следующий шаг — изменение функций Wiring на использование регистров (подробнее в подразделе 3.7 этой книги).

```
void setup () {
DDRD |= (1 <<3); // Назначение 3 цифрового вывода Arduino выходом.
TCCR2A=0b00100011; //Включение ШИМ на 3 выводе Arduino
TCCR2B=0b00000011; // Установка частоты для 3 вывода Arduino — 976 Гц
}
void loop () {
for (int i=0; i <=255; i=i+1) {
OCR2B=i; // подача напряжения на 3 вывод Arduino
delay (10);
}
for (int i=255; i> = 0; i = i-1) {
OCR2B=i; // подача напряжения на 3 вывод Arduino
delay (10);
}
delay (100);
}
```

Задачи для закрепления материала:

Задачи следует решать с использованием функций Wiring.

(Дополнительно) Решение задач с использованием регистров.

1. Начальные состояния — первый светодиод не светится, второй светится. Первый светодиод плавно зажигается, после этого второй светодиод плавно затухает. После первый светодиод плавно затухает, а второй плавно зажигается. Описанные алгоритм выполняется только один раз.

2. Два светодиода. Одновременно первый светодиод плавно зажигается, а второй плавно затухает, после одновременно первый светодиод плавно затухает, а второй плавно зажигается. Описанный алгоритм выполняется два раза.

3. Два светодиода. Одновременно первый светодиод плавно зажигается, а второй плавно затухает, после одновременно первый светодиод плавно затухает, а второй плавно зажигается. Описанный алгоритм выполняется многократно, пока на Arduino поступает напряжение питания.

Что мы изучили в подразделе 1.4:

1. Широтно-импульсную модуляцию;
2. Функцию analogWrite ();
3. Цикл for (подробнее в подразделе 2.2.3);
4. Основы работы с таймерами через регистры (подробнее в подразделе 3.7).

1.5 Управление группой светодиодов по входному параметру Оптимизация кода

Есть модули, которые включают в себя несколько светодиодов.

Примером является семисегментный индикатор — модуль индикации, который может отобразить любую цифру (от 0 до 9) и некоторые буквы.

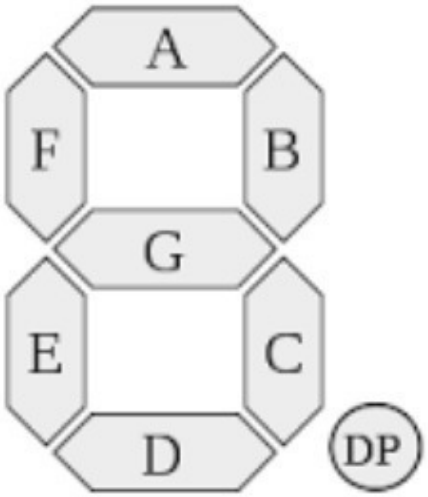
Каждый сегмент индикатора можно представить как отдельный светодиод.

Управляется каждый сегмент так же независимо, у каждого сегмента есть свой вывод.

Поэтому задача управления семисегментным индикатором сводится к формированию цифр через управление несколькими светодиодами.

Необходимо сразу определиться с нумерацией сегментов, чтобы в ходе решения задачи не было ошибок.

Вид семисегментного индикатора представлен ниже.



Необходимо каждому сегменту присвоить позицию. Допустим, сегмент A — 2, B — 3, C — 4, D — 5, E — 6, F — 7, G — 8, dp — 1.

Сегмент «dp» является дополнительным и присутствует не на всех модулях семисегментного индикатора.

Почему он под номером 1? Для реализации данной задачи нам не понадобится символ точки («dp»). Цифровые выводы 0 и 1 также использовать не будем. При такой нумерации символ A, под номером 2, можно подключить к цифровому выводу Arduino 2. Символ B (3) к выводу 3 и так далее. Номер символа будет соответствовать номеру вывода Arduino. Такое распределение будет удобно и исключит вероятность ошибки, хотя, разумеется, подключать выводы сегментов можно и к другим выводам Arduino.

Распишем, на какие сегменты нужно подать напряжение, чтобы отобразить каждую из цифр.

0: A (2),B (3),C (4),D (5),E (6),F (7).

1: B (3), C (4).

2: A (2), B (3), G (8), E (6), D (5).

3: A (2), B (3), G (8), C (4), D (5).

4: F (7), G (8), B (3), C (4).

5: A (2), F (7), G (8), C (4), D (5).
6: A (2), F (7), G (8), C (4), D (5), E (6).
7: A (2), B (3), C (4).
8: A (2), B (3), C (4), D (5), E (6), F (7), G (8).
9: A (2), B (3), C (4), D (5), F (7), G (8).

Цифры можно отображать по очереди, для этого представим список, представленный выше, с помощью функций *digitalWrite* и *delay (скетч промежуточный!)*.

```
void loop () {
//0
digitalWrite (2,1);
digitalWrite (3,1);
digitalWrite (4,1);
digitalWrite (5,1);
digitalWrite (6,1);
digitalWrite (7,1);
delay (50);
//1
digitalWrite (3,1);
digitalWrite (4,1);
delay (50);
//2
digitalWrite (2,1);
digitalWrite (3,1);
digitalWrite (5,1);
digitalWrite (6,1);
digitalWrite (8,1);
delay (50);
//3
digitalWrite (2,1);
digitalWrite (3,1);
digitalWrite (4,1);
digitalWrite (5,1);
digitalWrite (8,1);
delay (50);
//4
digitalWrite (3,1);
digitalWrite (4,1);
digitalWrite (7,1);
digitalWrite (8,1);
delay (50);
//5
digitalWrite (2,1);
digitalWrite (4,1);
digitalWrite (5,1);
digitalWrite (7,1);
digitalWrite (8,1);
delay (50);
//6
```

```
digitalWrite (2,1);
digitalWrite (4,1);
digitalWrite (5,1);
digitalWrite (6,1);
digitalWrite (7,1);
digitalWrite (8,1);
delay (50);
//7
digitalWrite (2,1);
digitalWrite (3,1);
digitalWrite (4,1);
delay (50);
//8
digitalWrite (2,1);
digitalWrite (3,1);
digitalWrite (4,1);
digitalWrite (5,1);
digitalWrite (6,1);
digitalWrite (7,1);
digitalWrite (8,1);
delay (50);
//9
digitalWrite (2,1);
digitalWrite (3,1);
digitalWrite (4,1);
digitalWrite (5,1);
digitalWrite (7,1);
digitalWrite (8,1);
delay (50);
}
```

В результате выполнения этого скетча меняющиеся цифры мы не увидим, потому, что на семисегментном индикаторе отобразится 0, а команд для того, чтобы «стереть» 0 не было. В итоге 1 и другие цифры будут «выводиться поверх» 0, а значит мы будем продолжать видеть 0.

Для решения проблемы можно после каждого *delay* вставить функции для снятия напряжения со всех выводов:

```
digitalWrite (2,0);
digitalWrite (3,0);
digitalWrite (4,0);
digitalWrite (5,0);
digitalWrite (6,0);
digitalWrite (7,0);
digitalWrite (8,0);
```

Но визуально это может выглядеть, как мерцание. Так как если смотреть на физический процесс, то мы включаем, выключаем, включаем, выключаем… и так от 0 до 9, а потом заново (так как *void loop* начинает выполняться повторно).

Решение достаточно простое, во время отображения каждой цифры, вместо подхода описанного выше, нужно подавать не только логические единицы, но и логические нули на необходимые выводы.

Тогда скетч примет следующий вид:

```
void loop () {
//0
digitalWrite (2,1);
digitalWrite (3,1);
digitalWrite (4,1);
digitalWrite (5,1);
digitalWrite (6,1);
digitalWrite (7,1);
digitalWrite (8,0);
delay (50);
//1
digitalWrite (2,0);
digitalWrite (3,1);
digitalWrite (4,1);
digitalWrite (5,0);
digitalWrite (6,0);
digitalWrite (7,0);
digitalWrite (8,0);
delay (50);
//2
digitalWrite (2,1);
digitalWrite (3,1);
digitalWrite (4,0);
digitalWrite (5,1);
digitalWrite (6,1);
digitalWrite (7,0);
digitalWrite (8,1);
delay (50);
//3
digitalWrite (2,1);
digitalWrite (3,1);
digitalWrite (4,1);
digitalWrite (5,1);
digitalWrite (6,0);
digitalWrite (7,0);
digitalWrite (8,1);
delay (50);
//4
digitalWrite (2,0);
digitalWrite (3,1);
digitalWrite (4,1);
digitalWrite (5,0);
digitalWrite (6,0);
digitalWrite (7,1);
digitalWrite (8,1);
delay (50);
//5
digitalWrite (2,1);
digitalWrite (3,0);
```

```
digitalWrite (4,1);
digitalWrite (5,1);
digitalWrite (6,0);
digitalWrite (7,1);
digitalWrite (8,1);
delay (50);
//6
digitalWrite (2,1);
digitalWrite (3,0);
digitalWrite (4,1);
digitalWrite (5,1);
digitalWrite (6,1);
digitalWrite (7,1);
digitalWrite (8,1);
delay (50);
//7
digitalWrite (2,1);
digitalWrite (3,1);
digitalWrite (4,1);
digitalWrite (5,0);
digitalWrite (6,0);
digitalWrite (7,0);
digitalWrite (8,0);
delay (50);
//8
digitalWrite (2,1);
digitalWrite (3,1);
digitalWrite (4,1);
digitalWrite (5,1);
digitalWrite (6,1);
digitalWrite (7,1);
digitalWrite (8,1);
delay (50);
//9
digitalWrite (2,1);
digitalWrite (3,1);
digitalWrite (4,1);
digitalWrite (5,1);
digitalWrite (6,0);
digitalWrite (7,1);
digitalWrite (8,1);
delay (50);
}
```

Теперь мы не включаем-выключаем индикатор, мы выключаем только отдельные сегменты, в таком случае на тех сегментах, которые должны будут включиться — уже есть напряжение. Мерцания не происходит.

Отображать цифры можно не только по возрастанию, но и по любому другому алгоритму. А можно отображать цифру в зависимости от входного параметра. Тогда, допустим, если некоторая переменная будет равна 5, то и индикатор покажет цифру 5.

В таком случае у нас будет десять (цифры от 0 до 9) заранее известных реакций (подать напряжение на нужные светодиоды для формирования заданной цифры и выключить напряжение с остальных светодиодов).

Мы можем сформировать скетч через конструкцию проверки условия *if*.

Если параметр равен 0, то вывести напряжение на следующие выводы…

Если параметр равен 1, то вывести напряжение на следующие выводы…

Можно сформировать скетч через вложенные конструкции:

Если параметр равен 0, то вывести напряжение на следующие выводы:…

Иначе если параметр равен 1, то вывести напряжение на следующие выводы:…

Но в данном случае, когда есть множество реакций (вывод цифр от 0 до 9) и один параметр, от значения которого зависит реакция, следует использовать конструкцию *switch* (подробное описание конструкции в подразделе 2.1.2).

«Switch» с Английского переводится, как «переключатель», так и мы будем переключать одну реакцию на другую при смене значения входного параметра.

Микроконтроллер должен понимать, от какого параметра зависит переключение, поэтому в круглых скобках после switch мы его записываем:

switch (a)

У конструкции switch есть свое тело выделяемое фигурными скобками:

switch (a) {

}

Переключаться микроконтроллер в конструкции switch будет между случаями, в которых переменная a равна тому или иному значению, поэтому в конструкции добавляем случаи *(скетч промежуточный!)*:

switch (a) {
case 0 :
case 1:
case 2:
}

case — с Английского «случай» или «ящик», так и мы добавляем случаи или некие ящики с действиями.

То есть код сейчас можно прочитать так:

Переключаться в зависимости от значения переменной a: если переменная a равна 0, то делать… если переменная a равна 1, то делать…

Своих фигурных скобок у *case* нет, но микроконтроллеру все еще нужно объяснить, где рабочее тело case заканчивается, для этого используется команда *break*; позволяющая после выполнения всех действий не идти дальше, а переместиться к строчке с закрывающей фигурной скобкой рабочего тела *switch*.

Тогда код принимает следующий вид:

switch (a) {
case 0 :
//функции для случая, когда a==0
break;
case 1:
//функции для случая, когда a==1
break;
case 2:
//функции для случая, когда a==2
break;
…

```
}
```

Преимущество такой записи — наглядность, это уменьшает вероятность случайных ошибок связанных с размещением не в том месте фигурных скобок или частей кода.

В итоге код для возможности отображения цифр 0,1,2 будет выглядеть следующим образом:

```
switch (a) {
case 0 :
digitalWrite (2,1);
digitalWrite (3,1);
digitalWrite (4,1);
digitalWrite (5,1);
digitalWrite (6,1);
digitalWrite (7,1);
digitalWrite (8,0);
delay (50);
break;
case 1:
digitalWrite (2,0);
digitalWrite (3,1);
digitalWrite (4,1);
digitalWrite (5,0);
digitalWrite (6,0);
digitalWrite (7,0);
digitalWrite (8,0);
delay (50);
break;
case 2:
digitalWrite (2,1);
digitalWrite (3,1);
digitalWrite (4,0);
digitalWrite (5,1);
digitalWrite (6,1);
digitalWrite (7,0);
digitalWrite (8,1);
delay (50);
break;
}
```

В случае, если предполагается, что переменная может принимать множество значений, в результате которых должна быть одна и та же реакция, или возможно нестандартное поведение, на которое нужно реагировать определенным образом, следует использовать «*default:*», который является аналогом *else*.

Код в таком случае будет выглядеть следующим образом.

```
switch (a) {
case 0 :
//функции для случая, когда a==0
break;
case 1:
//функции для случая, когда a==1
```

```
break;
case 2:
//функции для случая, когда a==2
break;
…
default:
break;
}
```

Обратите внимание, что после *default*: мы не пишем никакое значение, потому что код внутри *default* выполняется, если не подошёл ни один из *case*.

То есть, если переменная равна какому-то значению, но его нет среди значений *case*, то будут выполнены функции между *default:* и *break;*

В текущем коде после *default* будет логично выключить напряжение на всех выводах отвечающих за сегменты индикатора.

Тогда код примет следующий вид:

```
switch (a) {
case 0 :
//функции для случая, когда a==0
break;
case 1:
//функции для случая, когда a==1
break;
case 2:
//функции для случая, когда a==2
```

Конец ознакомительного фрагмента.