МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ РАДИОЭЛЕКТРОНИКИ
И ИНТЕЛЛЕКТУАЛЬНЫХ ТЕХНИЧЕСКИХ СИСТЕМ

Кафедра «Радиоэлектроника и телекоммуникации»

ПРОТОТИПИРОВАНИЕ ЭЛЕКТРОННЫХ СРЕДСТВ

Учебно-методическое пособие
по выполнению лабораторного практикума
для студентов укрупненной группы
направлений и специальностей 11.00.00

Севастополь
СевГУ
2025

УДК 621.38(076.5)
ББК 32.85я73
П837

Р е ц е н з е н т :
М. А. Дурманов - канд. техн. наук, доцент кафедры
«Радиоэлектроника и телекоммуникации»

Ответственный за выпуск:
И. Л. Афонин – д-р техн. наук, профессор, заведующий кафедрой
«Радиоэлектроника и телекоммуникации»

Составитель: Д. В. Начаров

П837 **Прототипирование электронных средств** : учебно-методическое пособие по выполнению лабораторного практикума для студентов для студентов укрупненной группы направлений и специальностей 11.00.00 / Севастопольский государственный университет, Институт радиоэлектроники и интеллектуальных технических систем, кафедра «Радиоэлектроника и телекоммуникации» ; сост. Д. В. Начаров. – Севастополь : СевГУ, 2025. – 64 с. – Текст : электронный.

Цель учебно-методического пособия – оказание помощи студентам в подготовке и выполнении лабораторных работ по дисциплине «Прототипирование электронных средств», а также в получении ими теоретических знаний по созданию макетов электронных средств с использованием платформы Arduino.

УДК 621.38(076.5)
ББК 32.85я73

Рассмотрено и рекомендовано к изданию на заседании кафедры «Радиоэлектроника и телекоммуникации», протокол № 9 от 20 февраля 2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью пособия является оказание помощи студентам для студентов укрупненной группы направлений и специальностей 11.00.00 в выполнении лабораторных работ по дисциплине «Прототипирование электронных средств», а также в получении ими теоретических знаний по созданию макетов электронных средств с использованием платформы Arduino.

В учебно-методическое пособие включены теоретические сведения по составу и назначению, видам сигналов и основам программирования отладочной платы Arduino, а также даны практические указания по сборке макетов электронных средств, составленных из таких компонентов и модулей, как резисторы, светодиоды, пьезокерамические излучатели, кнопки, переменные резисторы, фоторезисторы, семисегментные индикаторы и матричные клавиатуры.

Лабораторная работа № 1 посвящена изучению состава отладочной платы Arduino, среды разработки программ и структуры программы для платы Arduino, а также сборке схем из резисторов и светодиодов на беспаечной макетной плате и управлению ими с помощью цифровых сигналов и сигналов с широтно-импульсной модуляцией, формируемых отладочной платой Arduino.

Лабораторная работа № 2 посвящена изучению способов генерации звуковых сигналов заданной частоты и длительности с помощью пьезокерамического излучателя, содержит пример генерации мелодии на основе нотной записи.

Лабораторная работа № 3 посвящена изучению схем подключения кнопок с использование подтягивающих и стягивающих резисторов, а также способам считывания состояния кнопок в основной программе и с использованием прерываний.

Лабораторная работа № 4 посвящена приобретению знаний и навыков работы с аналого-цифровым преобразователем платы Arduino на примерах считывания состояний переменного резистора и фоторезистора с выводом считываемой информации на экран компьютера через монитор последовательного порта.

Лабораторная работа № 5 посвящена приобретению навыков вывода информации на одноразрядный и четырехразрядный семисегментные индикаторы способами статической и динамической индикации, а также освоению техники подключения к отладочной плате компонентов, содержащих большое количество выводов, с использованием последовательно-параллельного преобразования сигналов с помощью сдвигового регистра.

При выполнении лабораторных работ следует сохранять короткие видеозаписи с демонстрацией работы схем, которые затем следует преобразовывать в формат .gif и прикладывать к отчету.

В отчетах по лабораторным работам для общих и индивидуальных заданий следует приводить:

- цель работы и формулировки заданий;
- фотографии собранных схем;
- тексты разработанных программ;
- подробные выводы о проделанной работе.

Файлы отчетов и файлы в формате .gif следует загружать в соответствующие элементы электронного образовательного ресурса do.sevsu.ru.

Лабораторная работа № 1
ФОРМИРОВАНИЕ СИГНАЛОВ И ИНДИКАЦИЯ

1.1. Цель работы

Целью работы является изучение способов формирования сигналов при прототипировании электронных средств с использованием отладочной платы Arduino Uno и беспаечной макетной платы.

1.2. Отладочная плата Arduino Uno

Arduino — торговая марка аппаратно-программных средств построения и прототипирования простых систем, моделей и экспериментов в области электроники, автоматики, автоматизации процессов и робототехники. Отладочные платы Arduino получили широкое распространение в среде радиолюбителей и профессионалов в связи с минимальным входным порогом знаний в области разработки электроники и программирования, необходимым для начала сборки первых прототипов устройств. Кроме того, платы Arduino совместимы с большим количеством внешних радиоэлектронных модулей, что позволяет в короткие сроки получать работающие прототипы для отладки как программных, так и аппаратных технических решений.

Одной из наиболее распространённых отладочных плат Arduino является плата Arduino Uno (далее плата Arduino), построенная на микроконтроллере (МК) ATmega328. Эта плата имеет 14 цифровых портов ввода/вывода, 6 из которых поддерживают режим широтно-импульсной модуляции (ШИМ), 6 аналоговых входов, кварцевый генератор 16 МГц, разъем USB, разъем для подключения внешнего источника питания, разъем внутрисхемного программирования (ICSP — in-circuit serial programming) и кнопку сброса.

Вид платы Arduino показан на рисунке 1.1. По бокам платы Arduino расположены чёрные колодки с разъемами для подключения проводов. Колодки разъемов делятся на три группы:

— цифровые порты ввода/вывода — подписаны на плате как DIGITAL (PWM~);

— аналоговые порты ввода — подписаны на плате как ANALOG IN;

— источники питания — подписаны на плате как POWER.

Цифровые порты связаны с портами ввода/вывода МК. Аналоговые порты являются входами аналого-цифрового преобразователя МК. В колодке POWER есть выход напряжений 5 В и 3,3 В, а также есть два контакта «земли», обозначенные как GND.

Разъём для подключения кабеля USB используется для связи платы с компьютером и для загрузки в МК встраиваемого программного обеспечения, которое также называют микропрограммой или «прошивкой».

Рисунок 1.1 — Вид платы Arduino Uno

Питание платы может производиться либо через внешний блок питания с напряжением 7…12 В, подключенный к разъему питания, либо через кабель USB.

Также на плате есть группа компонентов преобразователя последовательного порта в интерфейс USB на микросхеме CH340, кварцевые резонаторы, используемые для тактирования МК и преобразователя последовательного порта, микросхема стабилизатора напряжения, расположенная рядом с разъемом питания и конденсаторами, а также кнопка сброса, с помощью которой плату можно перезапустить. Перезапуск также может быть выполнен кратковременным отключением питания платы.

Для индикации процессов, происходящих в плате, а также для целей отладки используются светодиоды:

— ON — индикатор наличия напряжения питания;

— RX — индикатор приема данных по последовательному интерфейсу;

— TX — индикатор передачи данных по последовательному интерфейсу;

— L — светодиод, подключенный к цифровому порту №13, который может использоваться для целей отладки.

1.3. Среда разработки Arduino IDE

Для разработки программного обеспечения платы используется свободно распространяемая среда разработки Arduino IDE.

Вид окна Arduino IDE показан на рисунке 1.2.

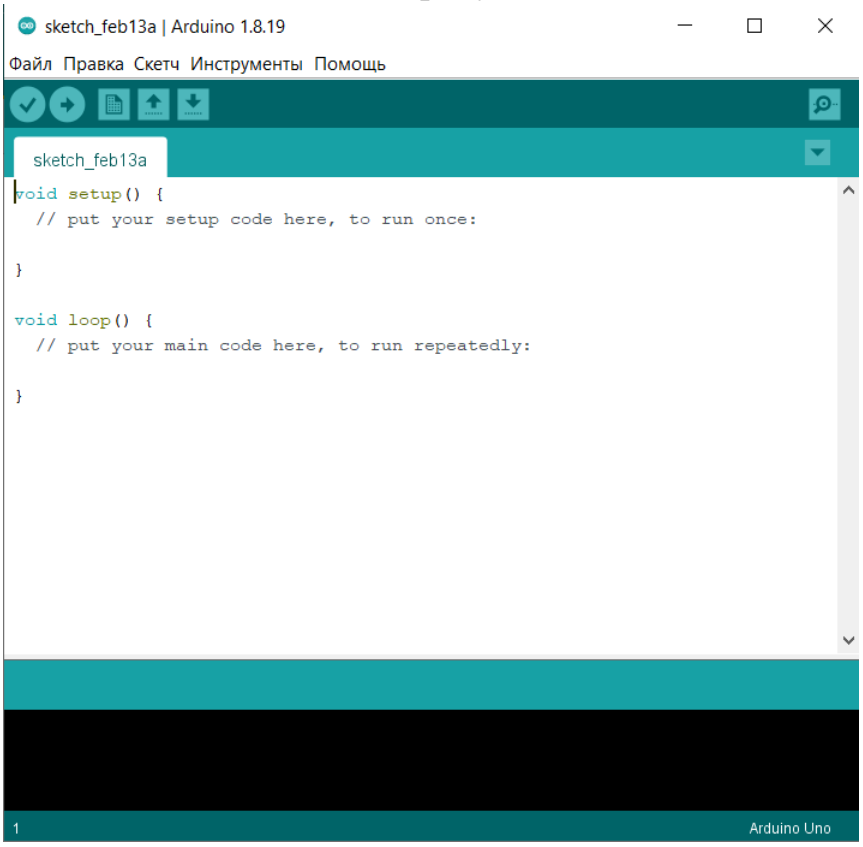


Рисунок 1.2 — Вид окна Arduino IDE при создании нового скетча

При создании нового файла открывается заготовка программы, содержащая определение функций `setup()` и `loop()`. Функция setup() выполняется один раз при старте программы и предназначена для начальной настройки платы. Функция `loop()` работает как бесконечный цикл и выполняется многократно. Именно в функцию `loop()` помещается основная программа платы. Программы для Arduino часто называю скетчами (от англ. sketch — эскиз, набросок).

В верхней части окна Arduino IDE есть набор кнопок:

 — создать новый скетч;

 — открыть другой скетч;

 — сохранить текущий скетч;

 — проверить скетч путем компиляции;

 — выполнить компиляцию и загрузку скетча на плату.

В результате компиляции программы можно убедиться в том, что в ней нет синтаксических ошибок. Arduino IDE выполняет компиляцию только сохраненных на диске программ, поэтому при работе с новым скетчем при нажатии на кнопку проверки Arduino IDE предложит сохранить скетч. Ход компиляции иллюстрируется полосой прогресса в нижнем правом углу окна. После завершения компиляции в окне вывода, расположенном в нижней части окна Arduino IDE, приводится информация об успешности компиляции, а также о количестве памяти, которое займет программа в памяти МК.

На рисунке 1.3 показаны примеры успешной компиляции и компиляции, завершившейся с ошибками.

а) б)

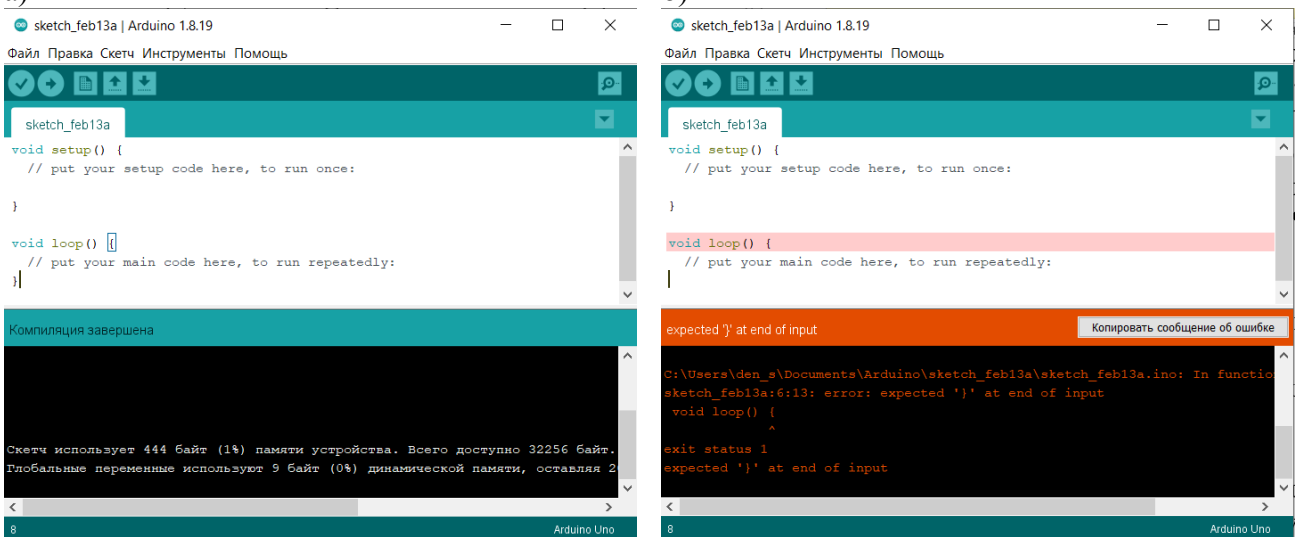


Рисунок 1.3 — Вид окна Arduino IDE после успешной (а) и неудачной (б) компиляции скетча

При наличии ошибок в программе в поле вывода указывается текстовое описание ошибки, а в поле редактора кода выделяется строка кода, имеющая отношение к ошибке. Не всегда ошибка содержится именно в выделенной строке — зачастую ошибка может находится в соседних строках либо, как в примере на рисунке 1.3, б в случае незакрытой фигурной скобки, в одной из многих строк, следующих за выделенной строкой. Поэтому при исправлении ошибок следует внимательно читать описание ошибки в поле вывода.

Для ускорения отладки скетча на плате при нажатии на кнопку  также производится компиляция, которая в случае отсутствия ошибок в скетче сопровождается загрузкой его в плату. Перед загрузкой скетча в плату нужно уточнить номер виртуального COM-порта, к которому подключена плата. Для этого необходимо запустить Диспетчер устройств, в разделе «COM и LPT» найти устройство с именем «USB Serial CH340» и определить номер COM-порта, указанный в скобках (рисунок 1.4). Далее в Arduino IDE следует указать номер COM-порта платы в меню Инструменты – Порт.

1.4. Первая программа для платы Arduino

Как правило, при освоении новой отладочной платы с МК пишут программу, выполняющую мигание светодиодом. Такая программа является аналогом программы «Hello, world!», которую обычно пишут в начале освоения нового языка программирования.

Программа мигания светодиодом L, установленным на плате, должна выполнить настройку платы, затем запустить бесконечный цикл повторения включения и включения светодиода. Настройка платы в этом случае предполагает настройку цифрового порта платы в режим вывода сигналов, то есть порт должен работать как выход. Для этого служит функция `pinMode`, вызов которой имеет следующий синтаксис

```
pinMode(<номер порта>, <режим работы порта>);
```

Режим работы порта задается с использованием следующих констант:

`OUTPUT` — режим выхода;

`INPUT` — режим входа;

`INPUT_PULLUP` — режим входа с подтягивающим резистором.

Поскольку настройка режима вывода сигналов на порте выполняется один раз, добавим вызов функции `pinMode` для настройки цифрового порта № 13 на выход внутри функции `setup()` (рисунок 1.4).

Далее внутри функции `loop()` следует выполнять включение и выключение светодиода путем выдачи на цифровом порте №13 высокого и низкого логических уровней. Для записи в порт цифровых значений служит функция `digitalWrite`, вызов которой имеет следующий синтаксис

```
digitalWrite(<номер порта>, <логическое значение>);
```

Логическое значение задается с использованием следующих констант:

`LOW` — логический 0 (низкое напряжение);

`HIGH` — логическая 1 (высокое напряжение).

Выполним поочередную отправку в порт № 13 высокого и низкого напряжения, добавив два вызова функции `digitalWrite` внутри функции `loop()` (рисунок 1.4).

Загрузите полученный скетч в плату Arduino и обратите внимание на светодиод L. Можно заметить, что вместо мигания светодиод оказывается включен с некоторым средним уровнем яркости. Такое свечение светодиода объясняется тем, что одно повторение функции `loop()` в МК происходит за достаточно короткий интервал времени — МК на плате Arduino тактируется с частотой 16 МГц, поэтому частота мигания светодиода в этом скетче имеет примерно такое же высокое значение. Для того, чтобы мигания светодиода были заметны глазу человека необходимо замедлить выполнение программы путем добавления задержек.

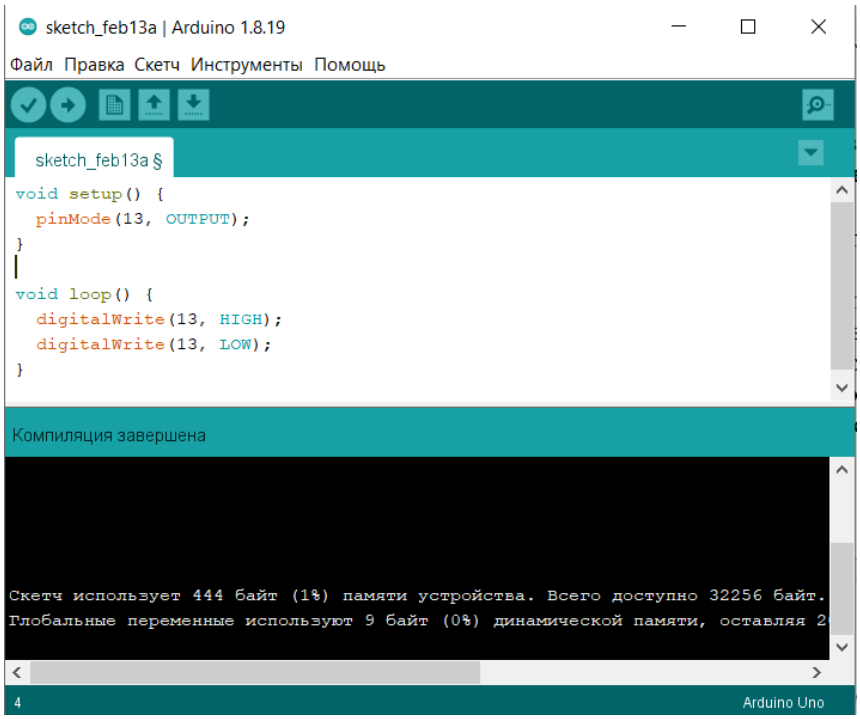


Рисунок 1.4 — Скетч мигания светодиодом без задержек

Для создания задержки в программе служит функция `delay`, вызов которой имеет следующий синтаксис

```
delay(<длительность задержки в миллисекундах>);
```

Добавим вызовы функции delay для создания задержек длительностью 500 мс после включения и выключения светодиода (рисунок 1.5).

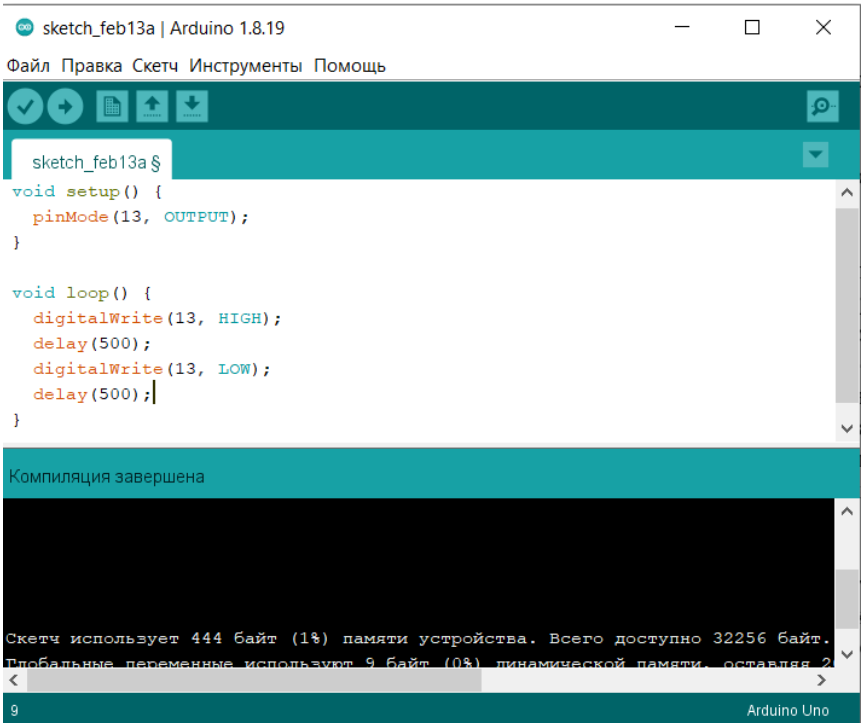


Рисунок 1.5 — Скетч мигания светодиодом с задержками

1.5. Сборка схем на беспаечной макетной плате

Для быстрой сборки электронных схем при прототипировании часто используются беспаечные макетные платы (англ. solderless breadboard), содержащие массивы подпружиненных контактов, в которые вставляются выводы электрорадиоэлементов или проводники. Вид беспаечной платы показан на рисунке 1.6.

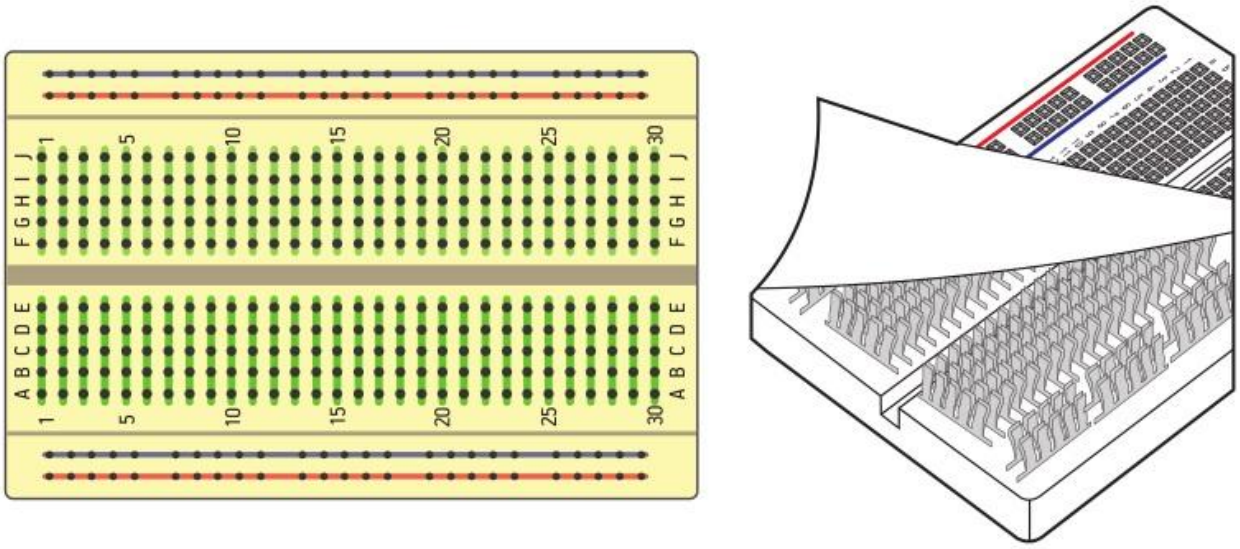
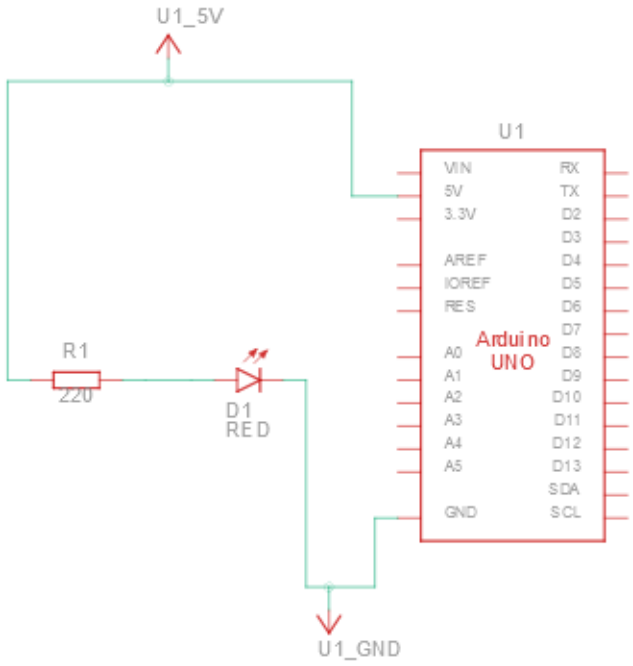


Рисунок 1.6 — Вид беспаечной макетной платы

Беспаечная макетная плата состоит из нескольких секций. В центральной секции строки пронумерованы, а столбцы обозначены буквами A-J. Строки центральной секции разделены на две части центральной канавкой. Внутри платы под центральной секцией проводники проложены горизонтально, то есть контакты в пределах строки слева и справа от центральной канавки соединены между собой. Также на плате расположены две боковые секции, контакты в которых обозначены символами + и -. Проводники в боковых секциях проложены вдоль платы, то есть все контакты линии + соединены между собой, также, как и все контакты линии -. При этом связей между боковыми секциями внутри макетной платы нет.

Выполним сборку схемы со светодиодом на макетной плате. **Сборку схемы следует выполнять при обесточенной (отключенной от компьютера) плате Arduino.** Электрическая принципиальная схема включения светодиода показана на рисунке 1.7, а, а вариант размещения компонентов и проводников на макетной плате, соответствующий принципиальной схеме, показан 1.7, б. Для подключения вывода компонента к выводу другого компонента или к проводнику необходимо разместить их в одной строке макетной платы.

а)



б)

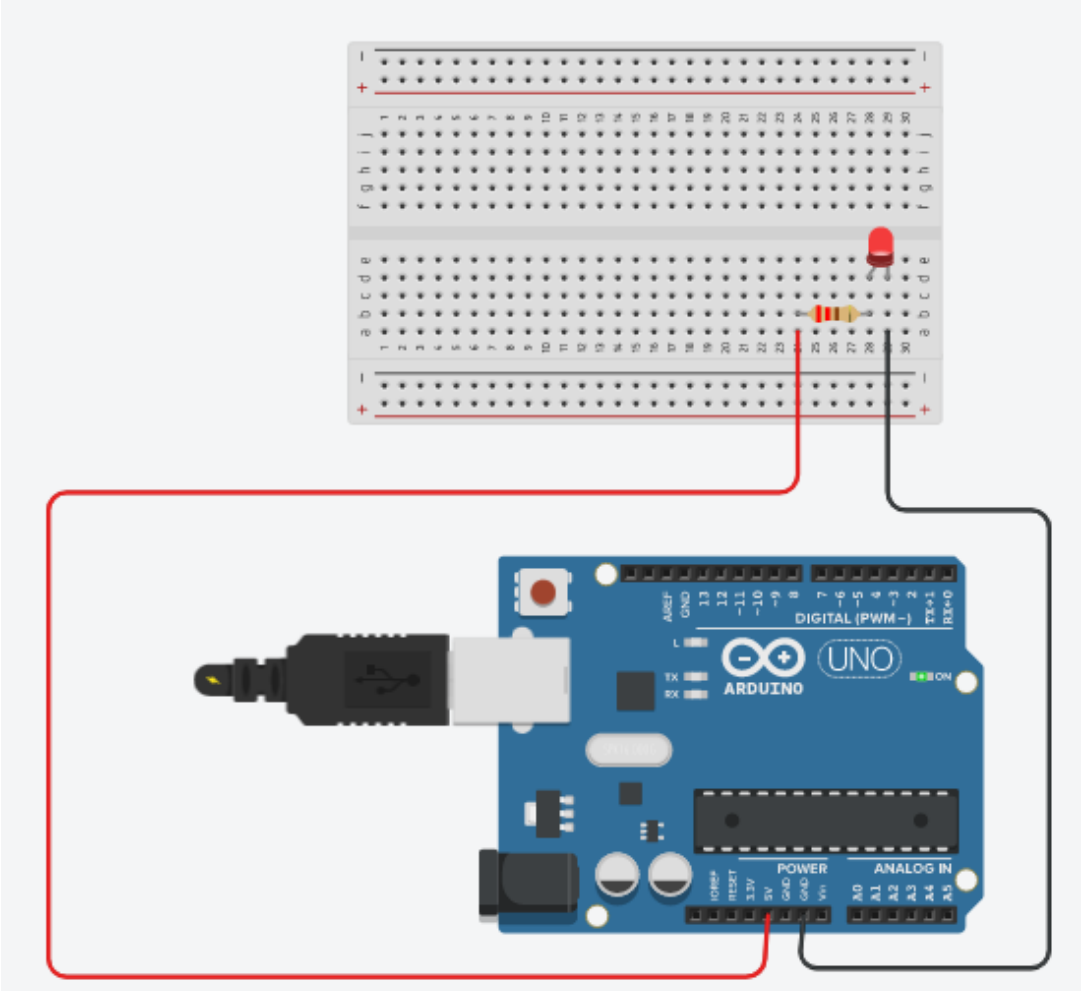
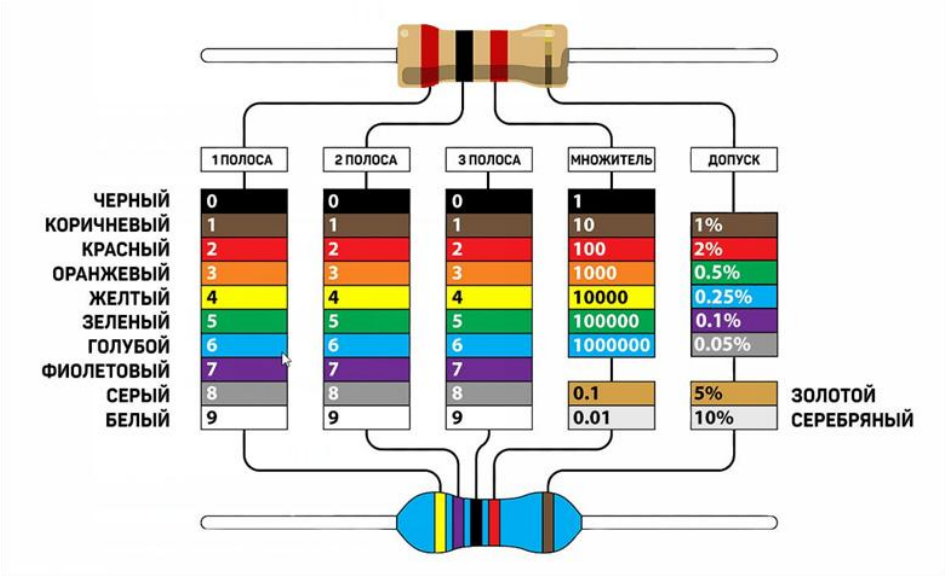


Рисунок 1.7 — Схема со светодиодом:
а — схема электрическая принципиальная; б — вариант размещения компонентов и проводников на беспаечной макетной плате

Последовательно со светодиодом включается резистор с сопротивлением 220 Ом, предназначенный для ограничения тока через светодиод. Для обозначения номинала резистора используется маркировка, состоящая из набора цветовых полос (рисунок 1.8).

20 · 100 Ом ± 5% = 2 кОм ± 5%



470 · 100 Ом ± 1% = 47 кОм ± 1%

Рисунок 1.8 — Значения цветовых полос маркировки резисторов и примеры их расшифровки

Компонент светодиода нужно подключить более длинным «плюсовым» выводом к цепи 5V, а вторым выводом — к цепи GND, которую также называют «землей» или общей.

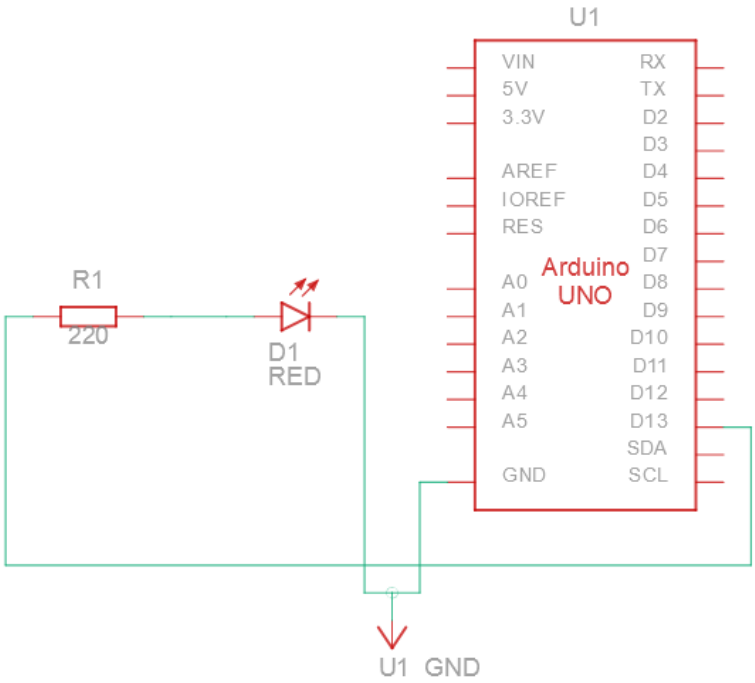
После проверки правильности сборки схемы подключите плату Arduino к компьютеру. Светодиод должен загореться.

1.6. Вывод цифровых сигналов

Измените схему как показано на рисунке 1.9, подключив ее к цифровому порту №13 платы Arduino. Загрузите в плату скетч, приведенный на рисунке 1.5, и пронаблюдайте мигание светодиода. Проведите опыт по оценке инерционности зрительной системы человека: определите при каком значении задержек наличие миганий светодиода перестает быть заметным.

Соберите схему «полицейской мигалки» в соответствии с рисунком 1.10. Обратите внимание на использование линии «-» для подключения обеих цепей к цепи GND (рисунок 1.10, б). Такой подход позволяет выполнить подключение большого числа цепей, используя только один порт платы Arduino.Разработайте скетч, включающий красный и синий светодиоды попеременно.

а)



б)

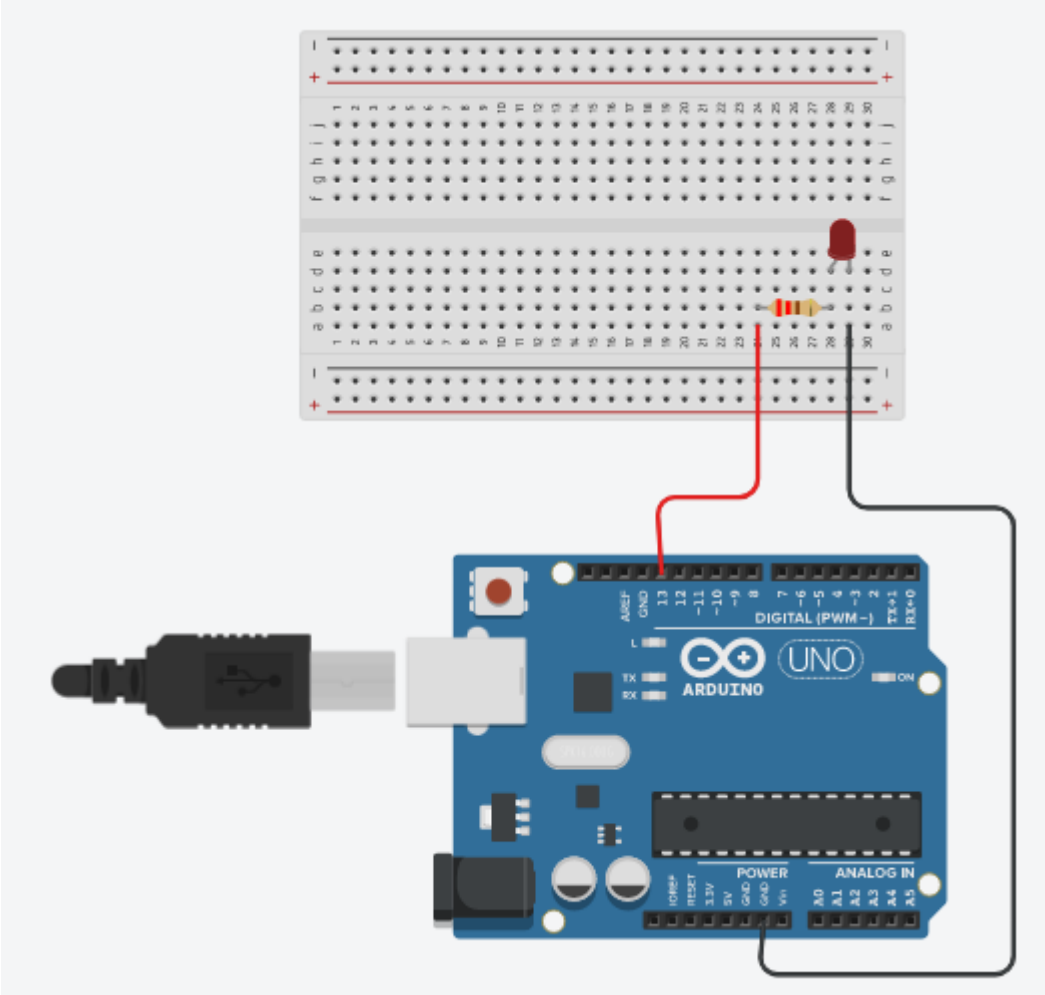
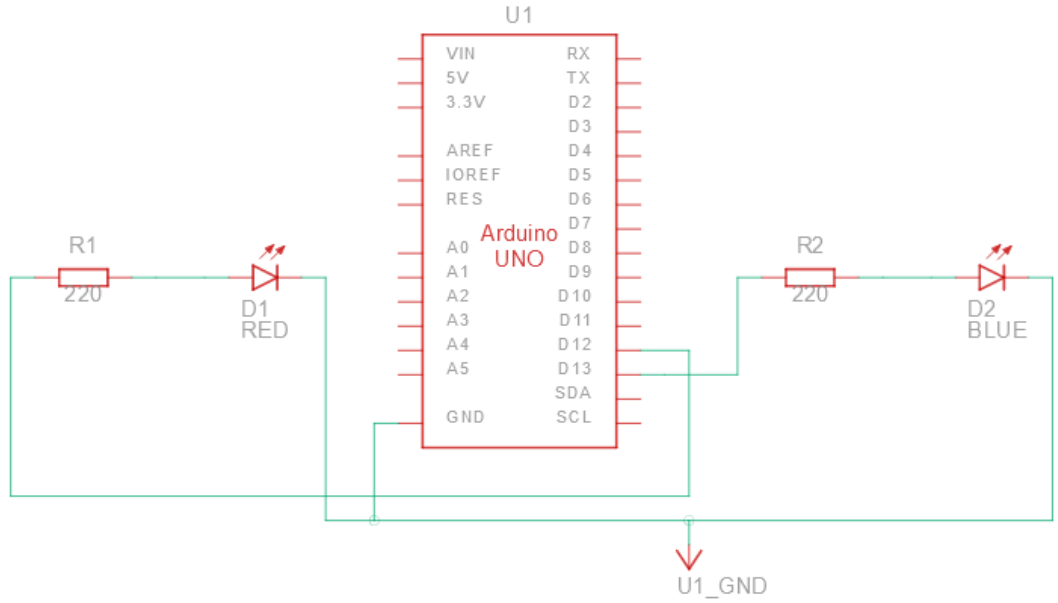


Рисунок 1.9 — Схема со светодиодом, подключенная к цифровому порту:
а — схема электрическая принципиальная; б — вариант размещения компонентов и проводников на беспаечной макетной плате

а)



б)

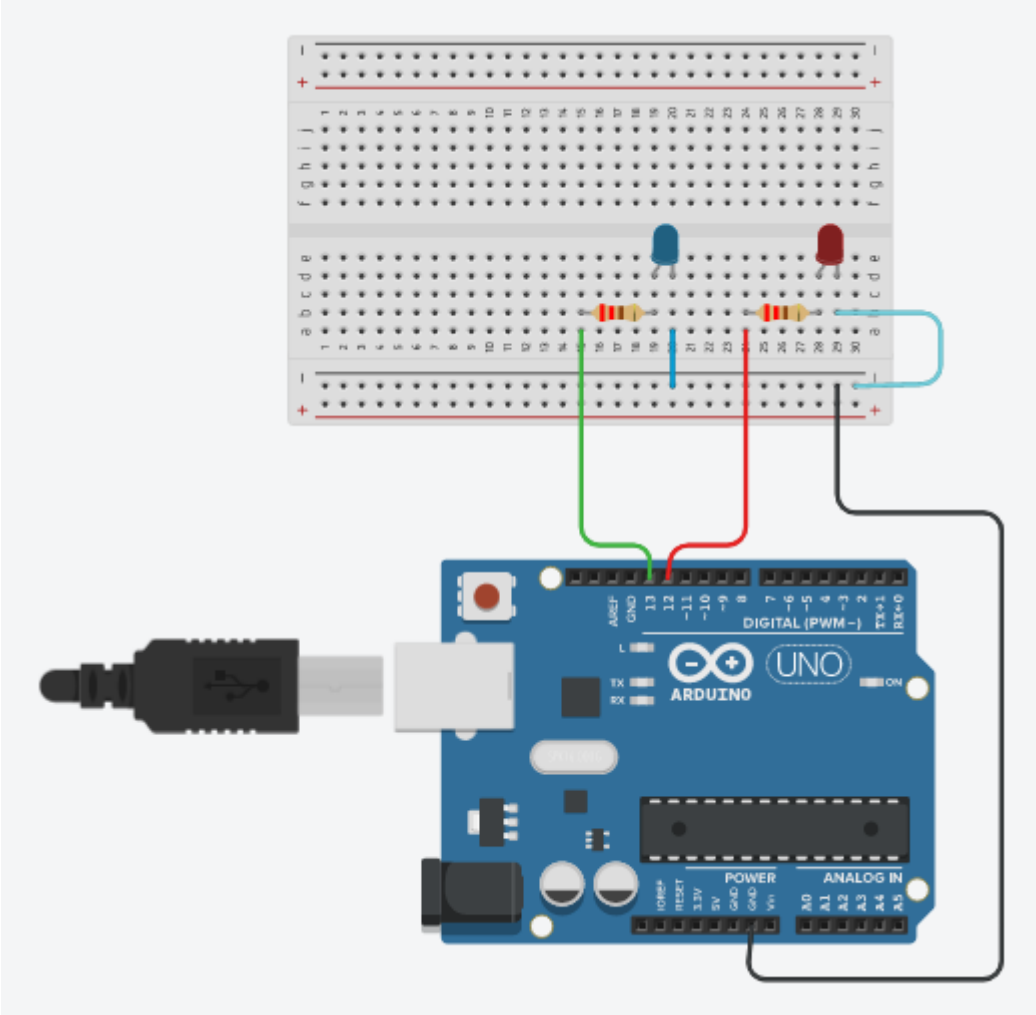


Рисунок 1.10 — Схема «полицейской мигалки»:
а — схема электрическая принципиальная; б — вариант размещения компонентов и проводников на беспаечной макетной плате

1.7. Вывод сигналов с широтно-импульсной модуляцией

Некоторые цифровые порты платы Arduino, в обозначениях которых есть символ «~», поддерживают режим выдачи сигналов с широтно-импульсной модуляцией (ШИМ). По-английски этот тип модуляции называется Pulse Width Modulation (PWM). ШИМ заключается в изменении коэффициента заполнения последовательности прямоугольных импульсов в соответствии модулирующим сигналом (рисунок 1.11). С помощью ШИМ на цифровом порту можно формировать напряжение, эквивалентное аналоговому, которое можно использовать для плавного управления инерционной нагрузкой, например, электродвигателями.

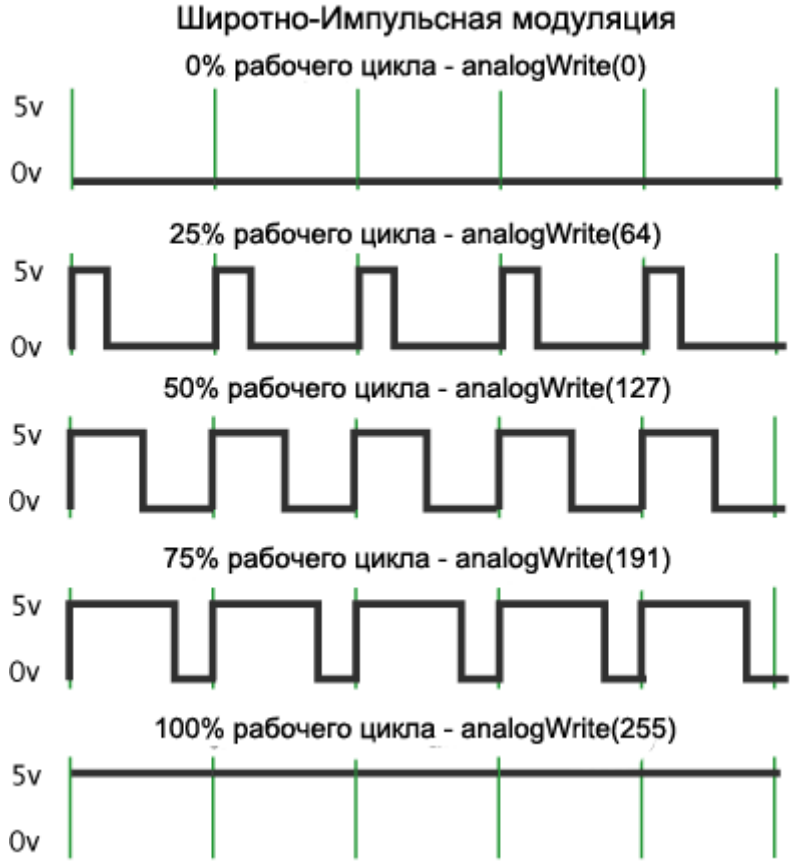


Рисунок 1.11 — Временные диаграммы, поясняющие принцип ШИМ

Для формирования ШИМ-сигнала на порту платы Arduino используется функция `analogWrite`, вызов которой имеет следующий синтаксис

```
analogWrite(<номер >, <значение модулирующего сигнала>);
```

Значение модулирующего сигнала задается из диапазона 0…255. При значении модулирующего сигнала 127, коэффициент заполнения последовательности импульсов будет равен 50%. При этом частота ШИМ-сигнала на портах 3, 9, 10 и 11 платы Arduino равна 488,28 Гц, а на портах 5 и 6 — 976,56 Гц.

Измените схему «полицейской мигалки», показанную на рисунке 1.10, подключив цепи светодиодов к цифровым портам №3 и №5. На рисунке 1.12 показан скетч, аналогичный скетчу с рисунка 1.5, в котором включение светодиода реализовано с помощью функции `analogWrite` с заданием коэффициента заполнения ШИМ сигнала, равного 50%.

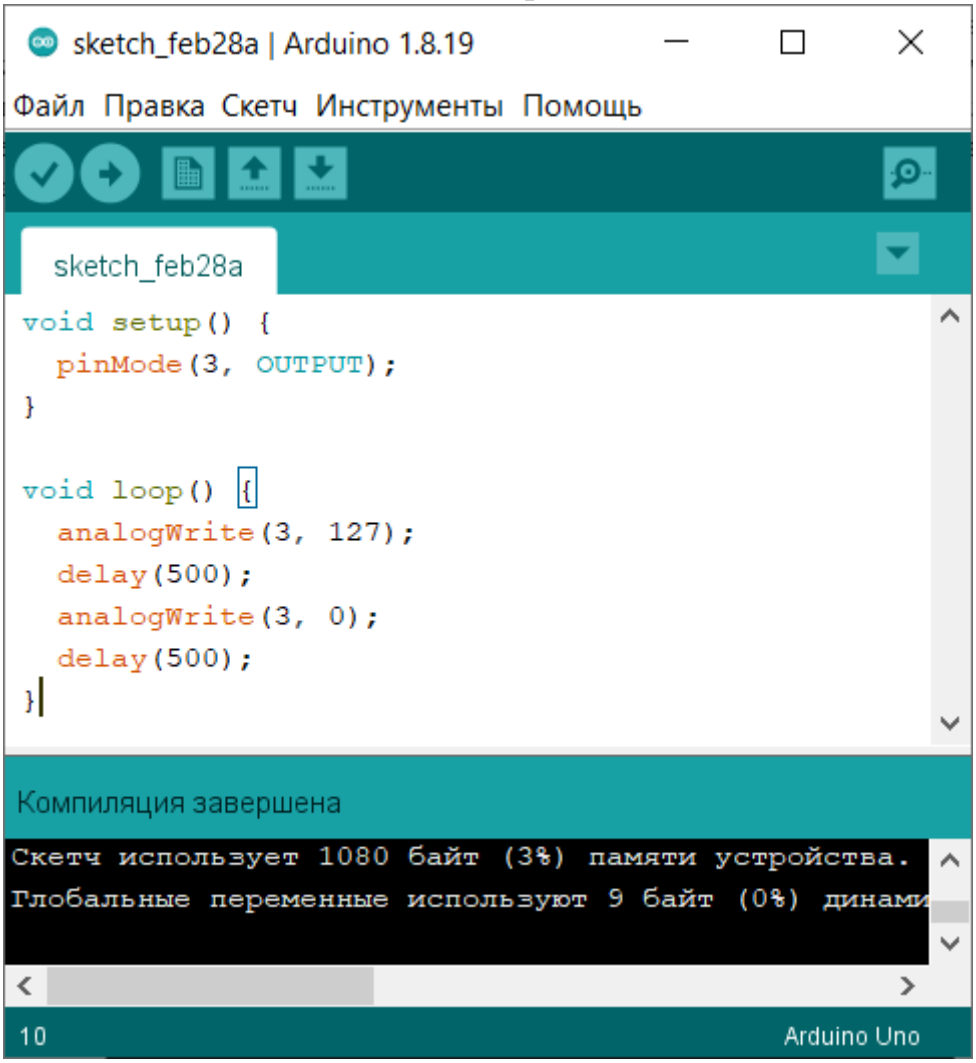


Рисунок 1.12 — Скетч мигания светодиодом с заданием коэффициента заполнения ШИМ сигнала, равного 50%

Выполните загрузку скетча с рисунка 1.11 в плату Arduino и обратите внимание на то, как изменилась яркость свечения светодиода.

На рисунке 1.13 показан скетч, выполняющий неограниченное увеличение коэффициента заполнения ШИМ сигнала в функции `loop()`. Переменная `intensity` хранит значения коэффициента заполнения ШИМ, которое увеличивается на 10 при каждом выполнении функции `loop()`.

Загрузите скетч с рисунка 1.13 в плату Arduino и пронаблюдайте характер изменения яркости светодиода. Можно заметить, что вместо неограниченного увеличения яркости происходит циклическое плавное увеличение от малой до средней яркости. Это связано с механизмом обработки чисел внутри

микроконтроллера и объясняется следующим образом. Количество двоичных разрядов, которыми кодируется значение коэффициента заполнения ШИМ сигнала равно 8, что соответствует значениям от 0 до 255 в десятичной системе счисления. При подаче значения коэффициента, большего чем 255, старшие разряды просто отбрасываются (рисунок 1.14).

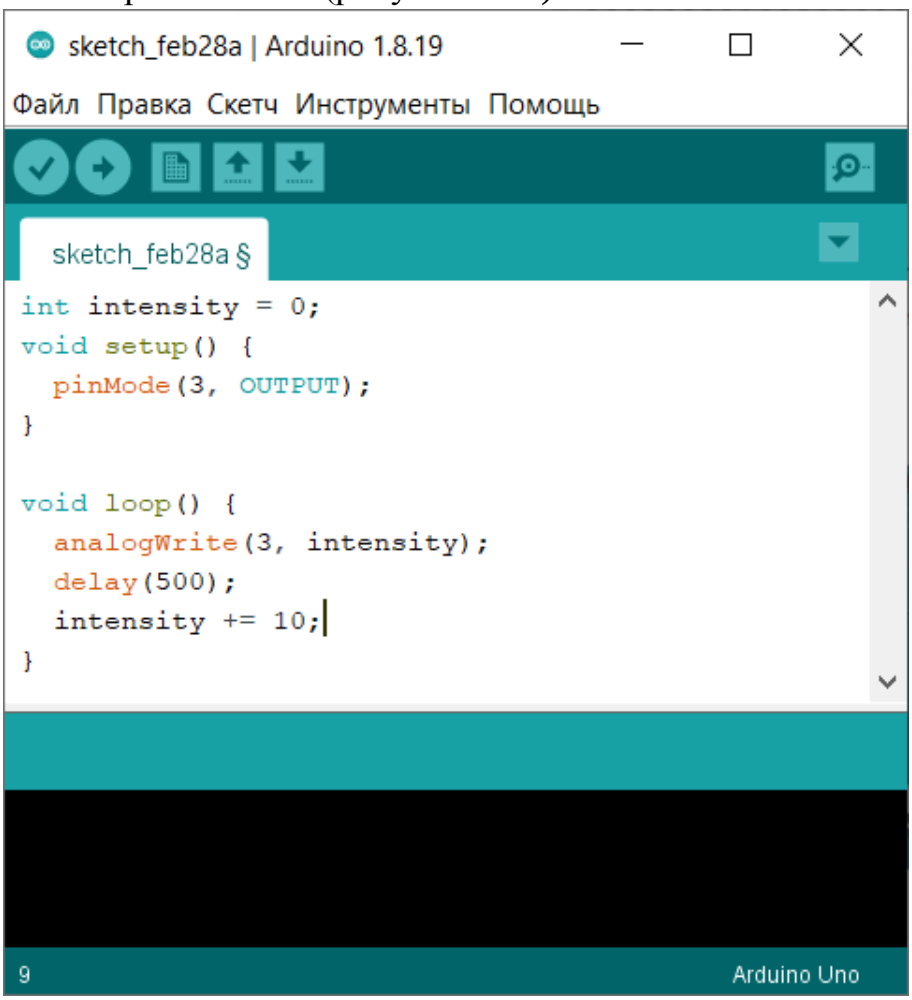


Рисунок 1.13 — Скетч, выполняющий неограниченное увеличение коэффициента заполнения ШИМ сигнала в функции `loop()`

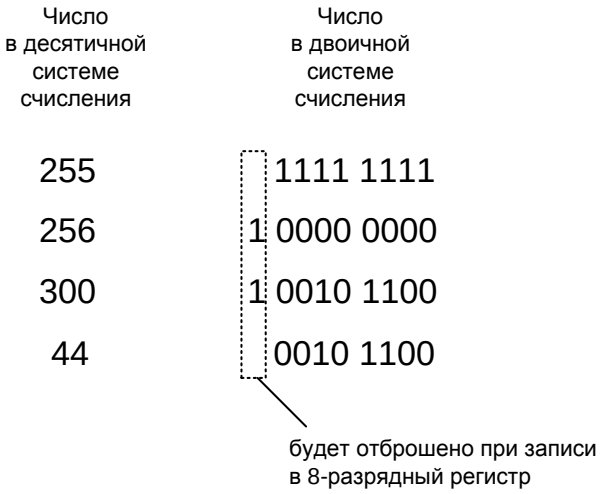


Рисунок 1.14 — Отбрасывание старшего разряда числа, отправляемого в 8-разрядный регистр, управляющий коэффициентов заполнения ШИМ сигнала

1.8. Задание на лабораторную работу

1.8.1. Напишите и загрузите в плату Arduino программу, выполняющую мигание светодиодом L на плате (п. 1.4).

1.8.2. Соберите схему, показанную на рисунке 1.7.

1.8.3. Напишите и загрузите в плату Arduino программу, выполняющую мигание светодиодом, собрав схему, показанную на рисунке 1.9 (п. 1.6). Определите значения задержек, при которых зрительная система человека перестает замечать мерцания светодиода. **Приведите в отчете скетч, в котором задержка равна 100*N*, где *N* — номер студента группы.**

1.8.4. Соберите схему (рисунок 1.10) и напишите и загрузите в плату Arduino программу «полицейской мигалки».

1.8.5. Напишите и загрузите в плату Arduino программу мигания светодиодом с заданием коэффициента заполнения ШИМ сигнала, равного 50% (п. 1.7).

1.8.6. Напишите и загрузите в плату Arduino программу, выполняющую неограниченное увеличение коэффициента заполнения ШИМ сигнала в функции `loop()` (рисунок 1.13).

1.8.7. Напишите и загрузите в плату Arduino программу, выполняющую пилообразное изменение яркости светодиода с плавными нарастанием и убыванием.

1.8.8. Напишите и загрузите в плату Arduino программу, аналогичную «полицейской мигалке», но с плавным изменением яркости светодиодов.

1.9. Содержание отчета

При выполнении работы сохраняйте короткие видеозаписи с демонстрацией работы схем, которые затем следует преобразовать в формат .gif и приложить к отчету.

В отчете для общих и индивидуальных заданий следует привести:

— цель работы и формулировки заданий;
— фотографии собранных схем;
— тексты разработанных программ;
— подробные выводы о проделанной работе.

Файл отчета и файлы в формате .gif следует загрузить в соответствующий элемент электронного образовательного ресурса (ЭОР) do.sevsu.ru.

Лабораторная работа № 2
ГЕНЕРАЦИЯ ЗВУКОВЫХ СИГНАЛОВ

2.1. Цель работы

Целью работы является изучение способов генерации звуковых сигналов при прототипировании электронных средств с использованием отладочной платы Arduino Uno и беспаечной макетной платы.

2.2. Пьезокерамический излучатель

Для генерации простых звуковых сигналов в электронных средствах часто применяются пьезокерамические излучатели (ПИ).

Пьезокерамическим излучателем или пьезопищалкой (англ. buzzer) называется устройство, в котором для получения звуковых колебаний используется пьезоэлектрический эффект — изменение длин материалов при пропускании через них электрического тока. Изменение формы материала сопровождается выгибанием пьезоэлемента, которое сохраняется до тех пор, пока к обкладкам элемента приложено напряжение (рисунок 2.1). При снятии напряжения, размеры пьезоэлемента вернутся в первоначальное состояние.

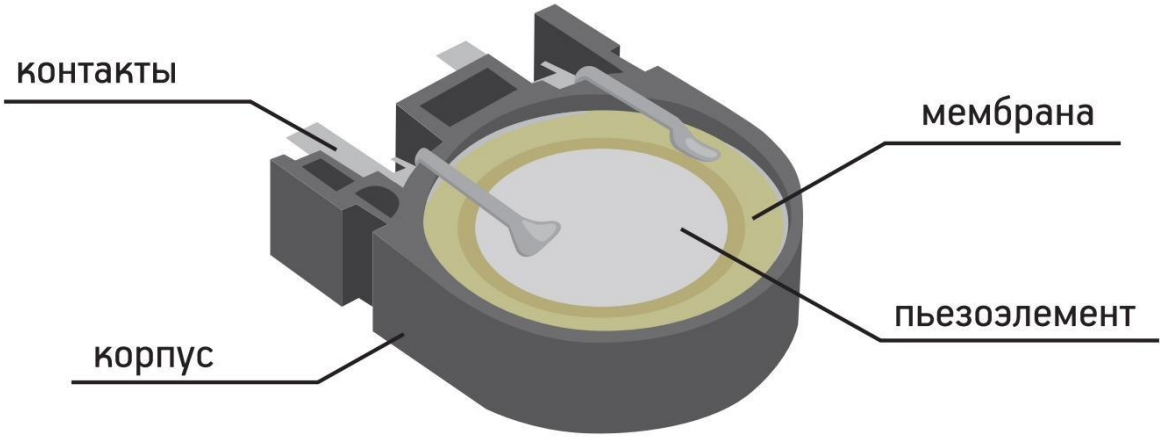
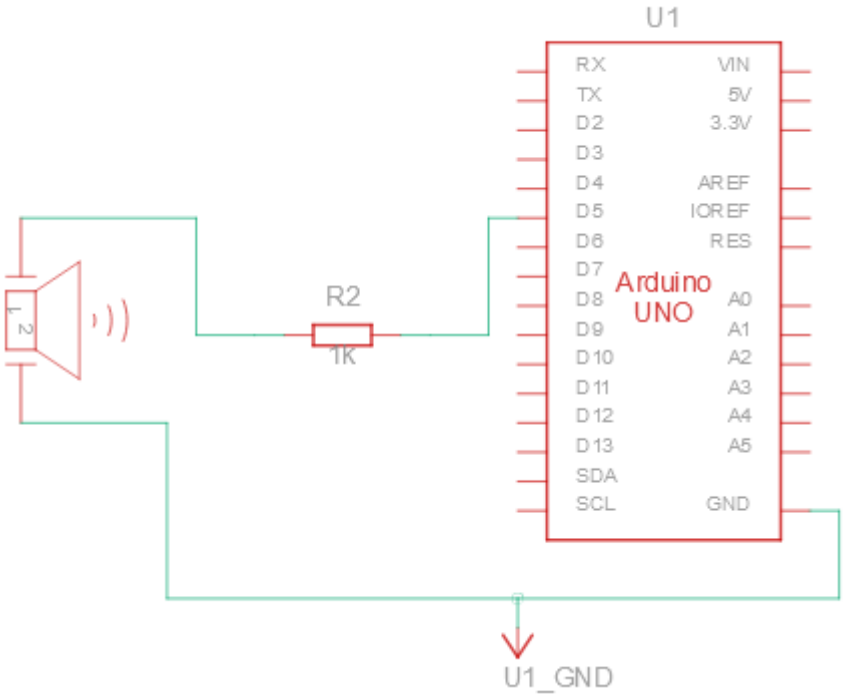


Рисунок 2.1 — Устройство пьезокерамического излучателя

Для генерации звуковых сигналов нужно обеспечить деформацию пьезоэлемента с частотой из звукового диапазона. В микроконтроллерных электронных средствах для этого часто используют подачу на ПИ последовательности прямоугольных импульсов заданной частоты с коэффициентом заполнения 50%.

На рисунке 2.2 показаны электрическая принципиальная схема включения ПИ и вариант размещения компонентов на макетной плате. Для уменьшения громкости звучания ПИ последовательно с ним включен резистор.

а)



б)

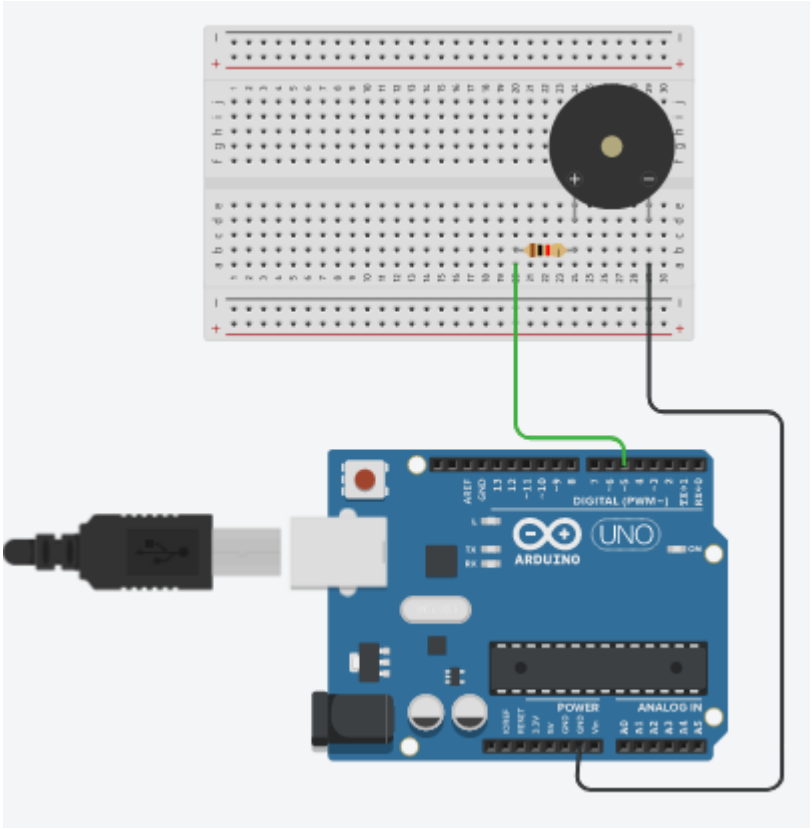


Рисунок 2.2 — Схема включения ПИ:
а — схема электрическая принципиальная; б — вариант размещения компонентов и проводников на беспаечной макетной плате

2.3. Примеры генерации звуковых сигналов

Для генерации непрерывного звукового сигнала на порту платы Arduino используется функция `tone`, вызов которой имеет следующий синтаксис

```
tone(<номер порта>, <частота звукового сигнала>);
```

Для выключения генерации звукового сигнала используется функция `noTone`, вызов которой имеет следующий синтаксис

```
noTone(<номер порта>);
```

На рисунке 2.3 показан скетч, в котором выполнена генерация непрерывного звука с частотой 440 Гц на цифровом порту №5. Частота 440 Гц соответствует ноте ля первой октавы и часто используется при настройке музыкальных инструментов и звуковой аппаратуры.

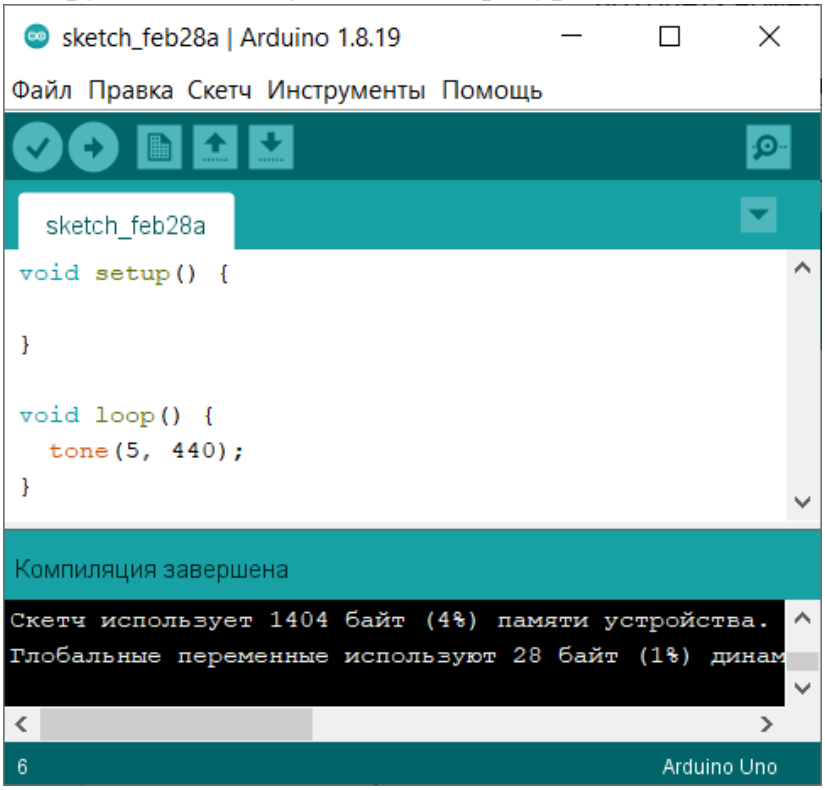


Рисунок 2.3 — Скетч генерации непрерывного звука с частотой 440 Гц на цифровом порту №5

Для генерации прерывистого звукового сигнала, как и в скетче мигания светодиодом, необходимо чередовать вызовы функций включения и отключения звука в цикле с задержкой (рисунок 2.4).

Интересной особенностью звуковых сигналов является октавная система. Октава в акустике и оптике — внесистемная безразмерная единица частотного интервала между двумя частотами f_1 и f_2, отношение которых $f_2 / f_1 = 2$. Схожим образом в музыке октавой называется музыкальный интервал с соотношением частот как 2 к 1, с разницей нот в 7 диатонических ступеней. Например, частоты нот ля первой и второй октав равны 440 Гц и 880 Гц соответственно.

На рисунке 2.4 показан скетч, последовательно генерирующий ноты ля первой и второй октав.

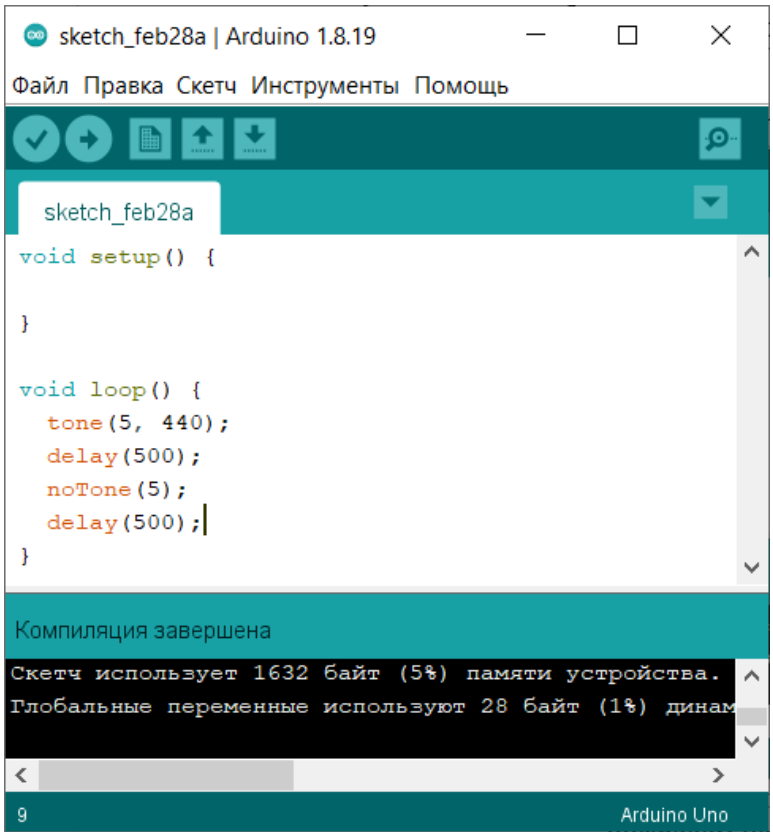


Рисунок 2.3 — Скетч генерации прерывистого звука

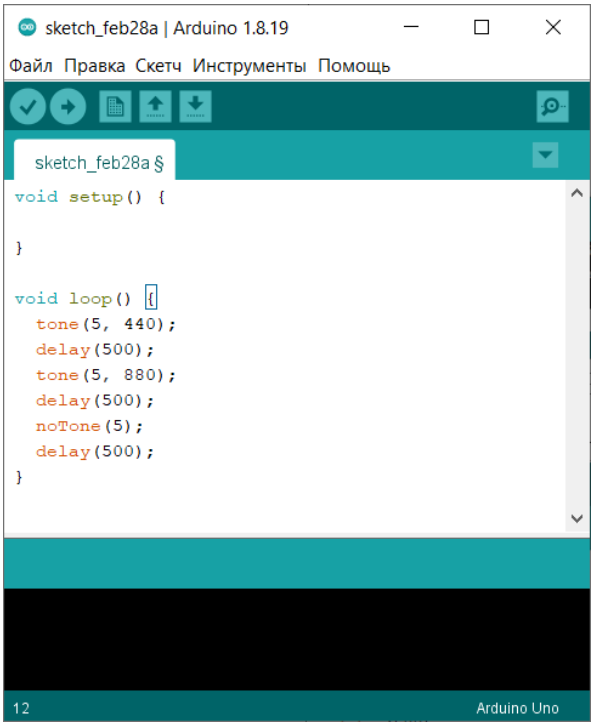


Рисунок 2.4 — Скетч генерации звуков, различающихся по высоте на октаву

Для привлечения максимального внимания человека звуковой сигнал должен характеризоваться значительной частотной модуляцией. Например,

звук сирены, оповещающей об опасности, формируется путем чередующихся увеличения и уменьшения частоты звучания в широких пределах. В программе генерация звука сирены может быть выполнена по аналогии со скетчем на рисунке 1.13 — путем задания частоты с использованием переменной, изменяемой на каждой итерации цикла.

2.4. Воспроизведение мелодий

Для воспроизведения мелодий достаточно выполнять последовательную генерацию звуков заданной высоты звучания (частоты) и длительности. Для генерации пауз необходимо выключать генерацию звука на заданную длительность.

Мелодии записываются с помощью нотной записи. Ноты записываются на параллельных горизонтальных линиях, называемых нотным станом. Положение ноты среди линий нотного стана определяет ее высоту звучания. На рисунке 2.5 показаны названия и расположение нот на нотном стане, а в таблице 2.1 приведены частоты колебаний, соответствующие нотам.

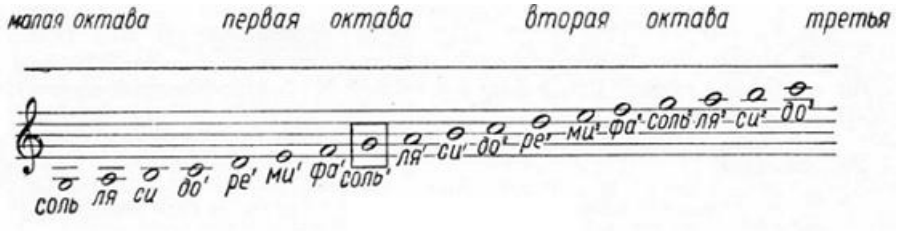


Рисунок 2.5 — Названия и расположение нот на нотном стане

Таблица 2.1 — Частоты колебаний, соответствующих нотам

Нота	Частота звучания музыкальных нот			
	Частоты звучания по октавам			
	Малая октава	Первая октава	Вторая октава	Третья октава
до	130.810	261.630	523.250	1046.500
до #	138.590	277.180	554.370	1108.730
ре	146.830	293.660	587.330	1174.700
ре #	155.560	311.130	622.260	1244.510
ми	164.810	329.630	659.260	1318.500
фа	174.610	349.230	698.460	1396.900
фа #	185.000	369.990	739.990	1479.980
соль	196.000	392.000	783.990	1568.000
соль #	207.650	415.300	830.610	1661.220
ля	220.000	440.000	880.000	1760.000
ля #	233.080	466.160	932.330	1864.660
си	246.940	493.880	987.770	1975.000

Для кодирования длительности нот и пауз применяется набор обозначений (рисунок 2.6).

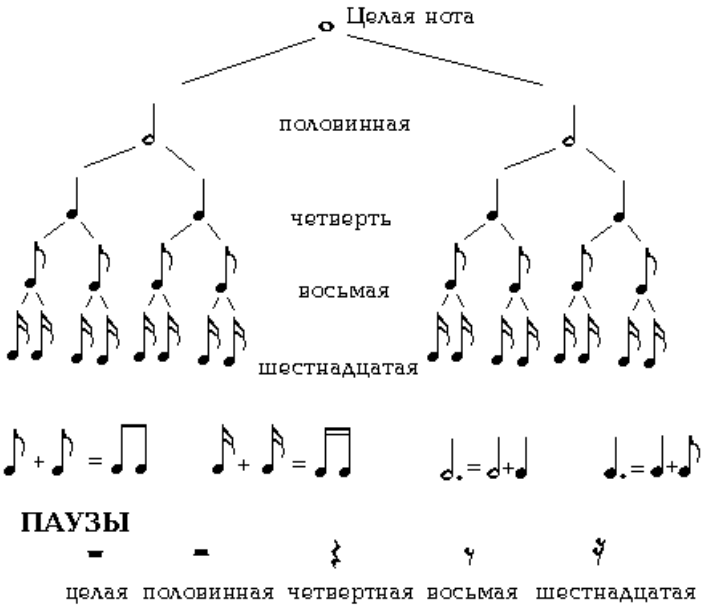


Рисунок 2.6 — Обозначения, применяемые
для кодирования длительности нот и пауз

Длительности измеряются в относительных единицах. Для того, чтобы определить абсолютное значение длительности звучания нот и пауз применяется параметр BPM (Beats Per Minute), обозначающий какое количество четвертных нот воспроизводится в минуту.

Рассмотрим пример скетча, выполняющего воспроизведение фрагмента мелодии «Севастопольский вальс» согласно нотной записи, приведенной на рисунке 2.7.

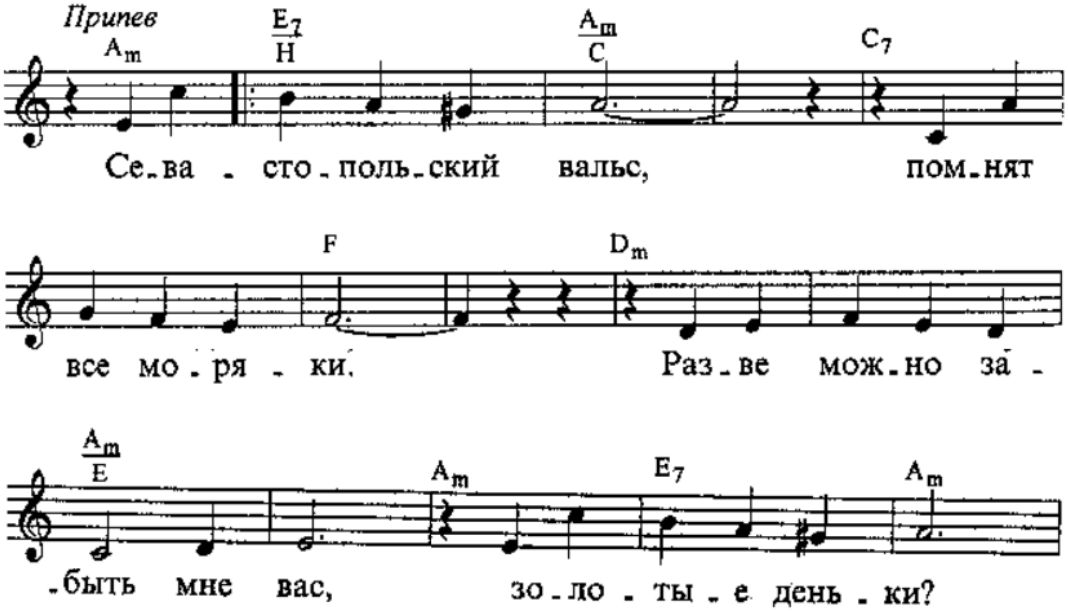


Рисунок 2.7 — Нотная запись фрагмента мелодии «Севастопольский вальс»

В начале скетча перед функцией `setup()` определим значения BPM и абсолютную длительность четвертной и целой нот с помощью констант, затем зададим константы, соответствующие частотам нот (рисунок 2.8).

```
const int BPM = 120;
// длительность четвертной ноты
const float FOURS_DURATION = 1000.0 * 60.0 / BPM;
// длительность целой ноты
const float FULL_NOTE_DURATION = 4 * FOURS_DURATION;

const float DO        = 261.63; // до первой октавы
const float DO_DIEZ   = 277.18;
const float RE        = 293.66;
const float RE_DIEZ   = 311.13;
const float MI        = 329.63;
const float FA        = 349.23;
const float FA_DIEZ   = 369.99;
const float SOL       = 392.0;
const float SOL_DIEZ  = 415.3;
const float LA        = 440;
const float LA_DIEZ   = 466.16;
const float SI        = 493.88;
const float DO_2      = 523.25; // до второй октавы
```

Рисунок 2.8 — Определение констант, задающий BPM, абсолютную длительность и частоты нот

Для воспроизведения одной ноты мелодии необходимо выполнять вызов функции `tone` с частотой ноты, задержку, соответствующую длительности ноты, и вызов функции `noTone` для отключения звука. Для воспроизведения паузы нужно выполнять вызов функции `noTone` и задержку, соответствующую длительности паузы.

Для удобства написания программы создадим следующие функции:

— для воспроизведения ноты:

```
play_note(<частота ноты>, <длительность ноты>);
```

— для воспроизведения паузы:

```
play_note(<длительность паузы>).
```

На рисунке 2.9 приведен исходный код создания функций.

```
void play_note(float freq, float duration)
{
  tone(5, freq);
  delay(duration * FULL_NOTE_DURATION);
  noTone(5);
}
void play_pause(float duration)
{
  noTone(5);
  delay(duration * FULL_NOTE_DURATION);
}
```

Рисунок 2.9 — Исходный код создания функций воспроизведения ноты и паузы

На рисунке 2.10 показан фрагмент скетча, в котором выполняется воспроизведение первых восьми нот и пауз мелодии «Севастопольский вальс» согласно записи рисунка 2.7.

```
void loop() {
  play_pause(0.25);          // четвертная пауза
  play_note(MI,    0.25);    // четвертная ми первой октавы
  play_note(DO_2, 0.25);     // четвертная до второй октавы

  play_note(SI,        0.25);// четвертная си первой октавы
  play_note(LA,        0.25);// четвертная ля первой октавы
  play_note(SOL_DIEZ, 0.25);// четвертная соль# первой октавы

  play_note(LA, 1.25);       // ля первой октавы длительности 0.5+0.25+0.5
  play_pause(0.25);          // четвертная пауза
```

Рисунок 2.10 — Фрагмент скетча, в котором выполняется воспроизведение первых восьми нот и пауз мелодии «Севастопольский вальс»

2.5. Задание на лабораторную работу

2.5.1. Соберите схему с ПИ.

2.5.2. Напишите и загрузите в плату Arduino программу, выполняющую воспроизведение непрерывного звука с частотой 440 Гц.

2.5.3. Напишите и загрузите в плату Arduino программу, выполняющую циклическое воспроизведение прерывистого звука с частотой 440 Гц.

2.5.4. **Напишите и загрузите в плату Arduino программу, выполняющую воспроизведение *N* прерывистых звуковых сигналов с частотой 100*N* Гц, где *N* — номер студента группы**.

2.5.5. Напишите и загрузите в плату Arduino программу, выполняющую циклическое воспроизведение прерывистого звука с уменьшением задержки.

2.5.6. Напишите и загрузите в плату Arduino программу, выполняющую циклическое воспроизведение прерывистого звука с уменьшением и увеличением задержки.

2.5.7. Напишите и загрузите в плату Arduino программу, выполняющую воспроизведение звука сирены (непрерывного звука с уменьшением и увеличением частоты).

2.5.8. Напишите и загрузите в плату Arduino программу, выполняющую последовательное воспроизведение двух звуковых сигналов, различающихся по высоте звучания на октаву.

2.5.9. Напишите и загрузите в плату Arduino программу, выполняющую воспроизведение мелодии «Севастопольский вальс».

2.6. Содержание отчета

В отчете для общих и индивидуальных заданий следует привести:

— цель работы и формулировки заданий;
— фотографии собранных схем;
— тексты разработанных программ;
— подробные выводы о проделанной работе.

Файл отчета следует загрузить в соответствующий элемент в ЭОР do.sevsu.ru.

Лабораторная работа № 3
ЧТЕНИЕ СОСТОЯНИЯ КНОПОК

3.1. Цель работы

Целью работы является изучение способов подключения и считывания состояния кнопок при прототипировании электронных средств с использованием отладочной платы Arduino Uno и беспаечной макетной платы.

3.2. Кнопки

Для управления электронным средством часто применяются кнопки — компоненты, представляющие собой пару механически замыкаемых или размыкаемых подпружиненных контактов. В простейшем случае кнопка имеет два вывода, однако, для повышения механической прочности при монтаже и удобства подключения часто применяют кнопки с четырьмя выводами, попарно связанными между собой внутри корпуса (рисунок 3.1). Некоторое распространение получил термин «тактовая кнопка», произошедший от английского tactile button (тактильная кнопка, то есть работающая от прикосновения) в связи с тем, что слово tactile иногда сокращают до «tact».

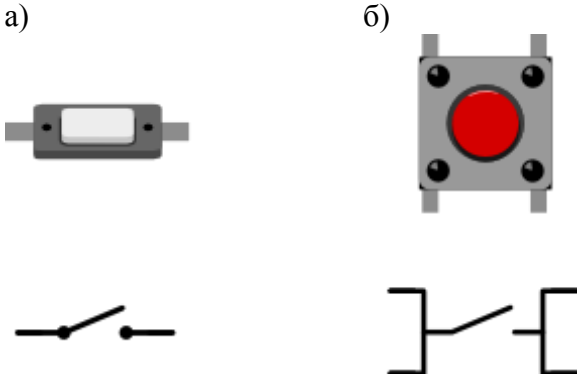


Рисунок 3.1 — Внешний вид и внутреннее устройство двухвыводной (а) и четырехвыводной (б) кнопок

3.3. Подключение кнопок

Кнопка является источником цифрового сигнала, поскольку может находится в двух устойчивых состояниях. Для считывания состояния кнопок следует подключать их к цифровым входам платы Arduino. Для того, чтобы состояние цифрового входа всегда было однозначно определено, кнопки подключаются с использованием резистора (рисунок 3.2): один вывод кнопки подключается к известному напряжению напрямую, а второй вывод — к другому известному напряжению через резистор с относительно большим

сопротивлением. При этом цифровой вход подключается к узлу цепи, в котором кнопка и резистор связаны.

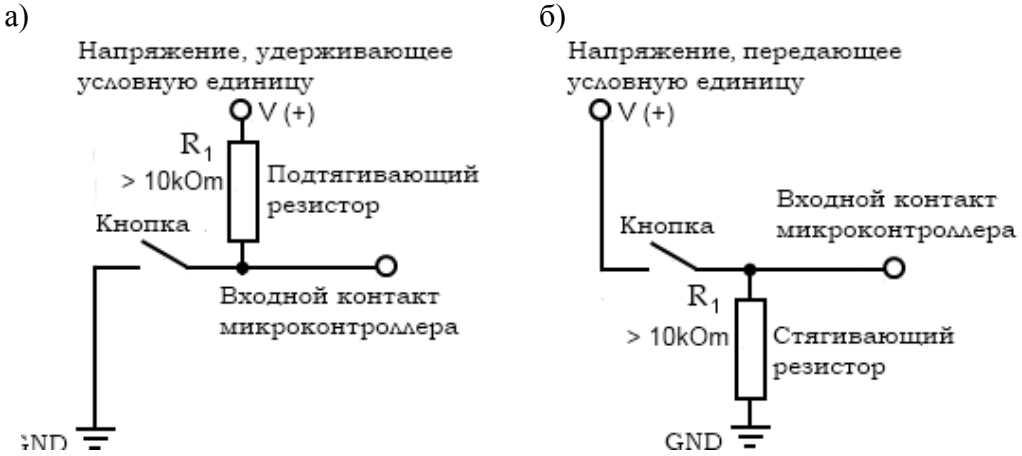


Рисунок 3.2 — Использование подтягивающего (а) и стягивающего (б) резисторов при подключении кнопки

Рассмотрим пример подключения кнопки с так называемым подтягивающим резистором (рисунок 3.2, а). При замкнутой кнопке на цифровом входе действует низкое напряжение, подключенное через внутреннее пренебрежимо малое сопротивление кнопки. При разомкнутой кнопке на цифровом входе действует высокое напряжение, подключенное через резистор. Если не подключить подтягивающий резистор, то при разомкнутой кнопке на цифровом входе будет действовать неопределенное случайное напряжение, которое будет определяться наводками — любой токопроводящий участок цепи, подключенный с одной стороны, работает как антенна. Резистор в этой схеме необходим для того, чтобы ограничить ток между точками высокого и низкого напряжения при замыкании кнопки: чем больше сопротивление подтягивающего резистора, тем меньше протекающий через замкнутую кнопку ток. Таким образом, при использовании подтягивающего резистора при нажатии на кнопку на цифровом входе будет считываться низкий уровень, а при отпускании кнопки — высокий.

Аналогичным образом работает стягивающий резистор (рисунок 3.2, б): при замкнутой кнопке на цифровом входе действует высокое напряжение, при разомкнутой — низкое, подключенное через резистор. Таким образом, при использовании стягивающего резистора при нажатии на кнопку на цифровом входе будет считываться высокий уровень, а при отпускании кнопки — низкий.

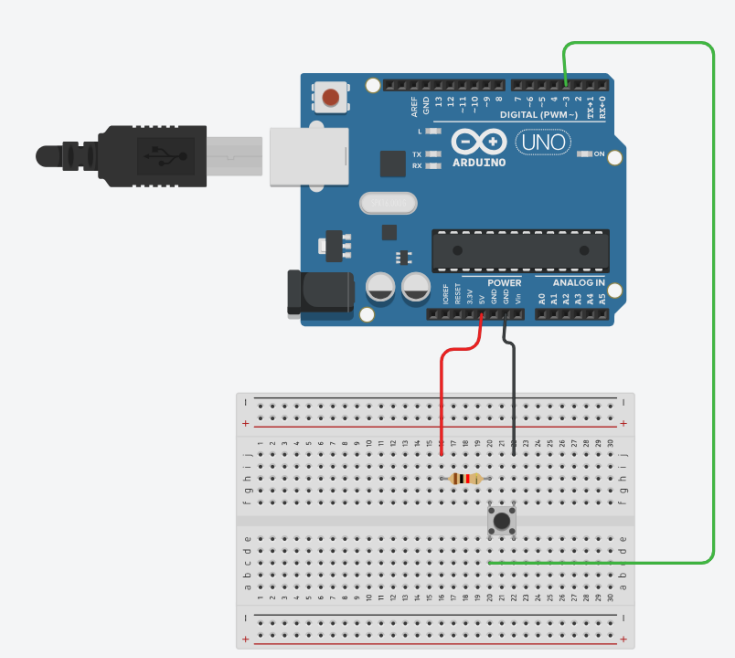
В англоязычных источниках подтягивающий резистор называется pull-up resistor, а стягивающий — pull-down resistor.

3.4. Чтение состояния кнопок

На рисунках 3.3 показаны вариант размещения компонентов и проводников на макетной плате и скетч считывания состояния кнопки при подключении кнопки к плате Arduino с подтягивающим резистором. В скетче в функции `setup` выполняется задание режима работы цифровых портов: порт 3 настроен как вход, порт 13 — как выход. Для считывания состояния кнопки в функции loop используется функция `digitalRead` вызов которой имеет следующий синтаксис

```
<логический уровень на порту> = digitalRead(<номер порта>);
```

а)



б)

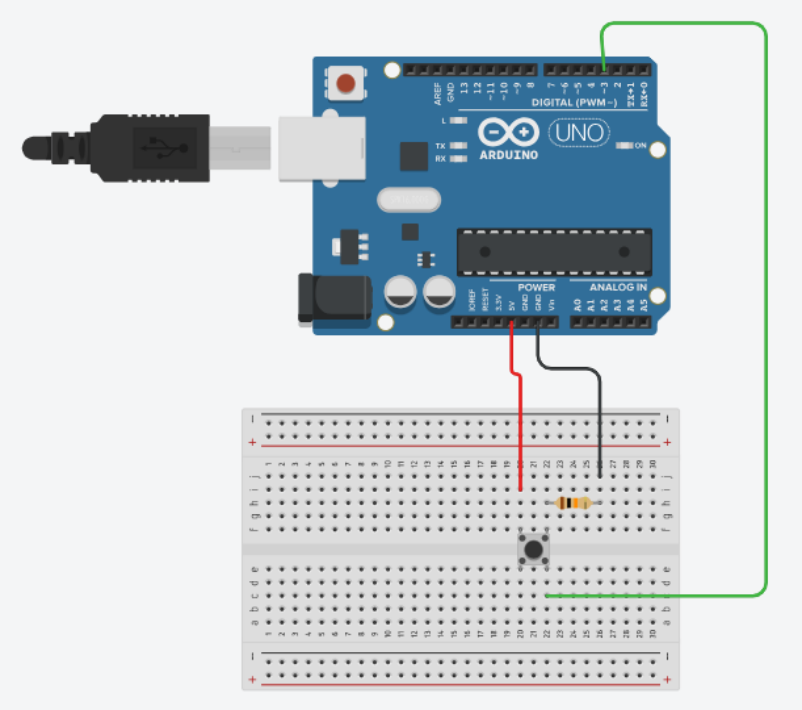
```
void setup() {
  pinMode(3, INPUT);
  pinMode(13, OUTPUT);
}

void loop() {
  if (digitalRead(3) == LOW) {
    digitalWrite(13, HIGH);
  } else {
    digitalWrite(13, LOW);
  }
}
```

Рисунок 3.3 — Вариант размещения компонентов и проводников на макетной плате (а) и скетч считывания состояния кнопки (б) при подключении кнопки к плате Arduino с подтягивающим резистором

Если кнопка нажата и функция `digitalWrite` возвращает значение `LOW`, то включается светодиод на плате Arduino, иначе светодиод выключается.

На рисунках 3.4 показаны вариант размещения компонентов и проводников на макетной плате и скетч считывания состояния кнопки при подключении кнопки к плате Arduino с стягивающим резистором.

а)



б)

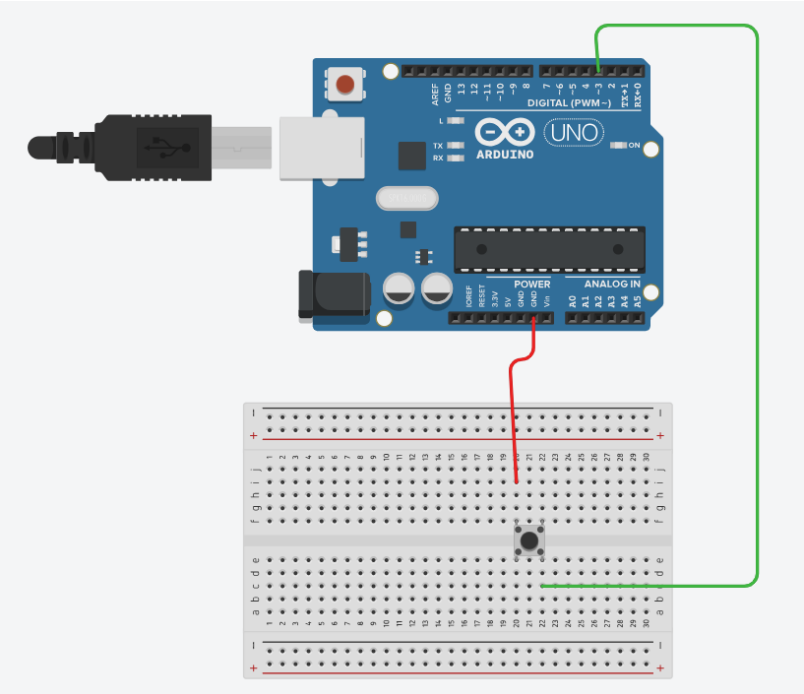
```
void setup() {
  pinMode(3, INPUT);
  pinMode(13, OUTPUT);
}

void loop() {
  if (digitalRead(3) == LOW) {
    digitalWrite(13, LOW);
  } else {
    digitalWrite(13, HIGH);
  }
}
```

Рисунок 3.4 — Вариант размещения компонентов и проводников на макетной плате (а) и скетч считывания состояния кнопки (б) при подключении кнопки к плате Arduino со стягивающим резистором

Для того чтобы упростить работу с кнопками, подтягивающие резисторы включают в состав цифровых микросхем на этапе производства. Для использования встроенных подтягивающих резисторов нужно задать соответствующий режим работы цифрового порта. На рисунке 3.5 показаны

вариант размещения компонентов и проводников на макетной плате и скетч считывания состояния кнопки при подключении к кнопке встроенного подтягивающего резистора на порту платы Arduino. В скетче в функции `setup` для цифрового порта 3 задан режим `INPUT_PULLUP`. Обратите внимание на то, что при этом внешний резистор на макетной плате подключать не нужно.

а)



б)

```
void setup() {
  pinMode(3, INPUT_PULLUP);
  pinMode(13, OUTPUT);
}

void loop() {
  if (digitalRead(3) == LOW) {
    digitalWrite(13, HIGH);
  } else {
    digitalWrite(13, LOW);
  }
}
```

Рисунок 3.5 — Вариант размещения компонентов и проводников на макетной плате (а) и скетч считывания состояния кнопки (б) при подключении к кнопке встроенного подтягивающего резистора на порту платы Arduino

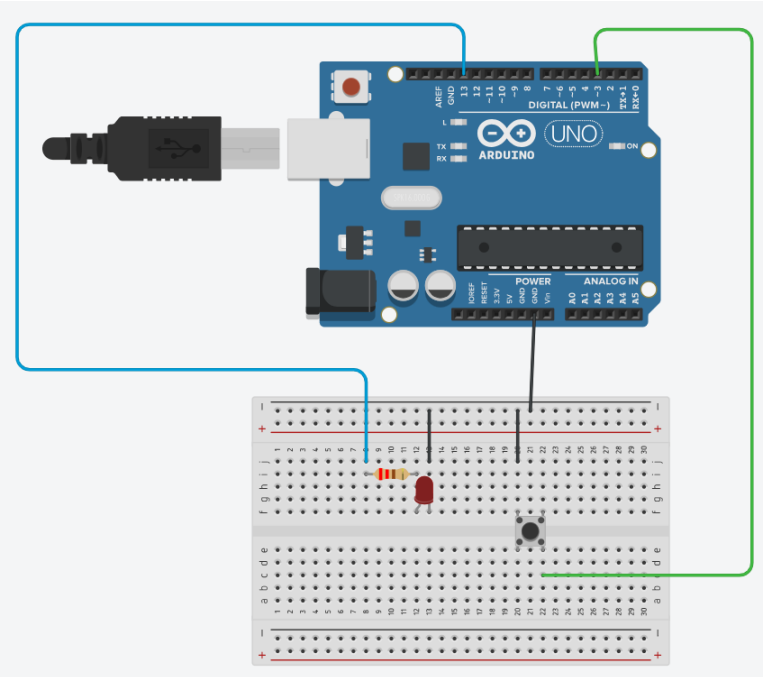
3.5. Реализация кнопки с памятью

Часто в электронных средствах на лицевых панелях или в пультах дистанционного управления применяются кнопки с памятью, реализованные с

помощью кнопок без фиксации. Например, кнопка на пульте управления может отвечать и за включение, и за выключение устройства. Такой способ управление позволяет сократить количество необходимых кнопок.

Для реализации кнопки с памятью необходимо сохранять предыдущее состояние кнопки в переменную, а управление осуществлять только при изменении текущего состояния относительно предыдущего. На рисунке 3.6 показаны схема подключения внешнего светодиода и кнопки, а также скетч, в котором реализовано управление по кнопке с памятью.

а)



б)

```
bool led = LOW;
bool button_previous_state = HIGH;
void setup() {
  pinMode(3, INPUT_PULLUP);
  pinMode(13, OUTPUT);
}
void loop() {
  bool button_current_state = digitalRead(3);
  if (button_previous_state == HIGH && button_current_state == LOW) {
    led = !led;
  }
  button_previous_state = button_current_state;
  digitalWrite(13, led);
}
```

Рисунок 3.6 — Вариант размещения компонентов и проводников на макетной плате (а) и скетч (б), в котором реализовано управление светодиодом по кнопке с памятью

Для хранения состояний светодиода и кнопки в скетче используются переменные `led` и `button_previous_state`, соответственно. В функции `loop` производится считывание текущего состояния кнопки в переменную `button_current_state`, затем производится проверка: если предыдущее

состояние кнопки было `HIGH` и текущее состояние `LOW`, значит произошло нажатие и необходимо изменить состояние светодиода путем инверсии переменной `led`. Далее текущее состояние кнопки записывается в переменную `button_previous_state` для проверки на следующей итерации функции `loop`. Таким образом, изменение состояния светодиода происходит по отрицательному фронту сигнала кнопки 3. Фактически, скетч на рисунке 3.5, а реализует работу Т-триггера с тактирование по отрицательному фронту сигналом кнопки с отправкой выходного сигнала на светодиод.

3.6. Использование внешних прерываний

Основная программа микроконтроллеров чаще всего представляет собой набор инструкций, выполняемых в бесконечном цикле: инструкции выполняются одна за одной в соответствии с программой, загруженной в соответствующую области памяти микроконтроллера. Можно сказать, что в этом случае происходит *синхронное выполнение программы* — можно заранее точно определить в какой момент времени будет выполняться тот или иной этап вычислений. Однако, зачастую микроконтроллеры используются для обработки событий реального мира, моменты наступления которых предсказать нельзя — такие события называются *асинхронными*. Для быстрой реакции на асинхронные события в микроконтроллерах реализован механизм прерываний.

Прерывание — механизм реакции микроконтроллера на асинхронное событие, заключающийся:

1) в приостановке основной программы с сохранением текущих значений переменных;
2) выполнении подпрограммы обработки прерывания;
3) возобновление выполнения основной программы.

Возможность обработки асинхронных событий позволяет реализовать квазипараллельное выполнение программы в однопроцессорной вычислительной системе.

В зависимости от источника асинхронного события прерывания могут быть внешними и внутренними. Для обработки нажатия на кнопку используются внешние прерывания.

На плате Arduino Uno для подключения схем, вызывающих внешние прерывания, могут использоваться цифровые порты 2 и 3. Для отслеживания и обработки прерываний используется функция `attachInterrupt`, вызов которой имеет следующий синтаксис

```
attachInterrupt(
    <номер прерывания>,
    <имя функции-обработчика прерывания>,
    <событие, вызывающее прерывание>
```

);

Номер внешнего прерывания, определяется номером порта, к которому подключен источник прерывания. Номера портов и номера прерываний для платы Arduino Uno не совпадают, поэтому для определения номера прерывания используется справочная информация или функция `digitalPinToInterrupt`, вызов которой имеет следующий синтаксис

```
digitalPinToInterrupt(<номер порта>);
```

Событие на порту, вызывающее прерывание, может описываться одним из значений, приведенных в таблице 3.1.

Таблица 3.1

Значение последнего аргумента функции `attachInterrupt`	Описание события
`LOW`	прерывание вызывается при значении `LOW`
`CHANGE`	прерывание вызывается при смене значения: с `LOW` на `HIGH` или с `HIGH` на `LOW`
`RISING`	прерывание вызывается только при смене значения с `LOW` на `HIGH`
`FALLING`	прерывание вызывается только при смене значения с `HIGH` на `LOW`

На рисунке 3.7 показан скетч обработки нажатия кнопки по внешнему прерыванию.

```
volatile bool led = LOW;
void setup() {
  pinMode(3, INPUT_PULLUP);
  pinMode(13, OUTPUT);
  attachInterrupt(digitalPinToInterrupt(3), btnIsr, FALLING);
}
void loop() {
  digitalWrite(13, led);
}
void btnIsr() {
  led = !led;
}
```

Рисунок 3.7 — Скетч обработки нажатия кнопки по внешнему прерыванию

При смене значения с `HIGH` на `LOW` на цифровом порту 3 запускается функция-обработчик `btnIsr()`, в которой производится инвертирование переменной `led`. Часто при именовании функции-обработчиков прерываний используется аббревиатура ISR (Interrupt Service Routine).

Функция-обработчик прерывания не может содержать задержек и должна выполняться максимально быстро для того, чтобы состояние памяти, сохраненное в момент возникновения прерывания, не потеряло актуальность. Другими словами, обработка прерывания не должна заметно нарушать и задерживать работу основной программы. Глобальные переменные, изменяемые в функции-обработчике прерывания, должны иметь квалификатор `volatile`, который указывает компилятору, что такие переменные получают значения не из основного потока исполнения программы.

3.7. Задание на лабораторную работу

3.7.1. Соберите схему подключения кнопки с подтягивающим резистором и повторите скетч управления светодиодом на плате Arduino (рисунок 3.3).

3.7.2. Соберите схему подключения кнопки со стягивающим резистором и повторите скетч управления светодиодом на плате Arduino (рисунок 3.4).

3.7.3. Соберите схему подключения кнопки с внутренним подтягивающим резистором и повторите скетч управления светодиодом на плате Arduino (рисунок 3.5).

3.7.4. Соберите схему подключения кнопки с внутренним подтягивающим резистором и светодиодом на макетной плате и повторите скетч управления светодиодом по кнопке с памятью (рисунок 3.6).

3.7.5. Выполните скетч обработки нажатия кнопки по внешнему прерыванию (рисунок 3.7).

3.7.6. **Напишите и загрузите в плату Arduino скетч, согласно которому яркость светодиода увеличивается при последовательных одиночных нажатиях на кнопку от 0 до 255 с шагом *N*, где *N* — номер студента группы.**

3.8. Содержание отчета

При выполнении работы сохраняйте короткие видеозаписи с демонстрацией работы схем, которые затем следует преобразовать в формат .gif и приложить к отчету.

В отчете для общих и индивидуальных заданий следует привести:

— цель работы и формулировки заданий;

— фотографии собранных схем;

— тексты разработанных программ;

— подробные выводы о проделанной работе.

Файл отчета и файлы в формате .gif следует загрузить в соответствующий элемент в ЭОР do.sevsu.ru.

Лабораторная работа № 4
ВВОД АНАЛОГОВЫХ СИГНАЛОВ

4.1. Цель работы

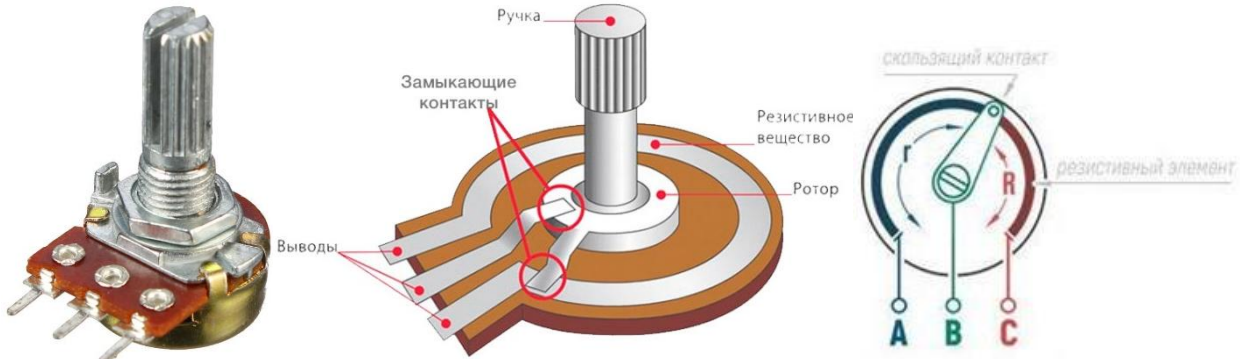
Целью работы является приобретение навыков ввода аналоговых сигналов при прототипировании электронных средств с использованием отладочной платы Arduino Uno и беспаечной макетной платы.

4.2. Аналого-цифровой преобразователь платы Arduino

Для ввода аналоговых сигналов, источниками которых могут быть датчики, используются линии аналого-цифрового преобразователя (АЦП), обозначенные на плате Arduino как ANALOG IN. На выходе АЦП формируются двоичные сигналы разрядностью 10, что соответствует десятичным значениям от 0 до 1023.

В качестве источника аналогового сигнала будем использовать переменный резистор (рисунок 4.1).

а)



б)

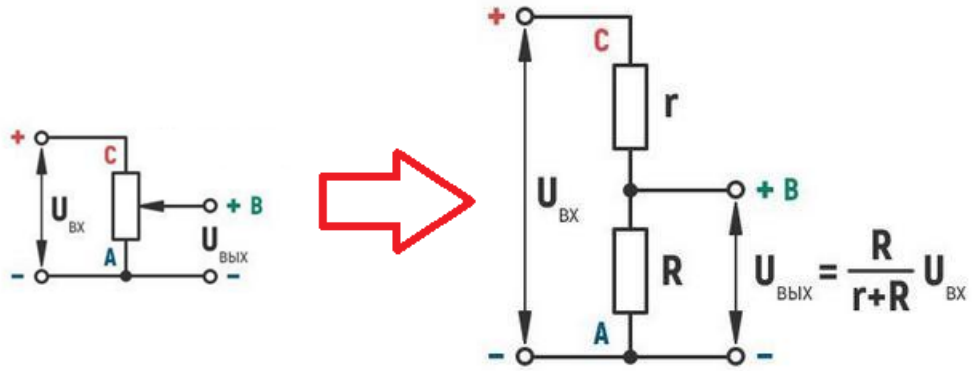


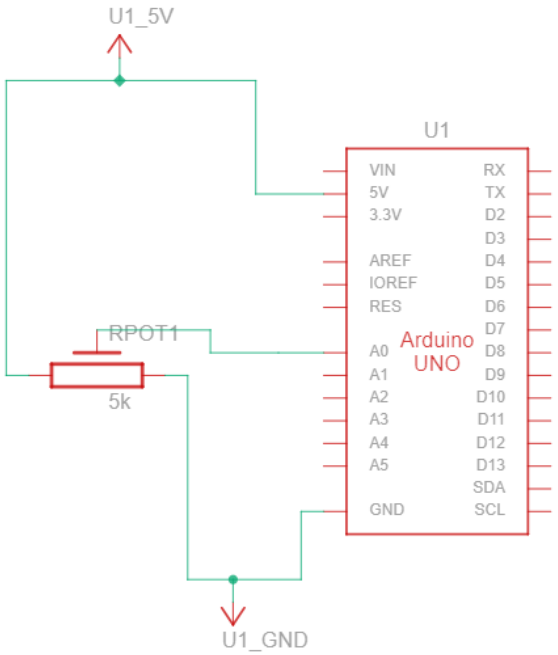
Рисунок 4.1 — Переменный резистор:
а — внешний вид и внутреннее устройство;
б — условное графическое обозначение и эквивалентная схема

Переменный резистор представляет собой полосу резистивного материала, имеющую форму незамкнутой окружности, по которой двигается

скользящий контакт. Положение скользящего контакта задается поворотом ручки (рисунок 4.1, а). Переменный резистор можно представить как делитель напряжения, составленный из сопротивлений между выводами A и B и между выводами B и C (рисунок 4.1). При этом величины этих сопротивлений изменяются при повороте ручки.

Чаще всего выводы A и C переменного резистора подключаются к источнику питания и общей цепи, а вывод B — к порту АЦП (рисунок 4.2).

а)



б)

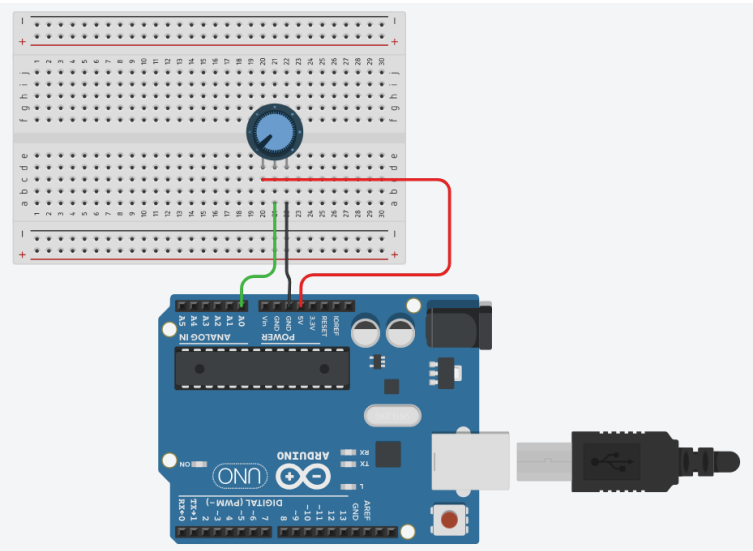


Рисунок 4.2 — Схема включения переменного резистора ко входу АЦП платы Arduino: а — схема электрическая принципиальная; б — вариант размещения компонентов и проводников на беспаечной макетной плате

Для чтения выходного значения АЦП применяется функция `analogRead()`, вызов которой имеет следующий синтаксис

```
analogRead(<номер аналогового порта>);
```

В качестве номера порта при вызове функции `analogRead()` используются значения от 0 до 5.

Также для того, что иметь возможность наблюдать за значением на выходе АЦП при повороте ручки переменного резистора, удобно отправлять соответствующие значения на компьютер через последовательный порт. Последовательный порт — это программно-аппаратный канал передачи данных между платой Arduino и компьютером. Аппаратно последовательный порт представляет собой линию связи, реализованную с помощью четырехпроводного кабеля USB соединение, две сигнальные линии которого со стороны платы Arduino подключены к линиям Rx и Tx модуля универсального асинхронного приемопередатчика (UART — Universal Asynchronous Receiver and Transmitter) микроконтроллера через микросхему преобразователя USB-UART, а со стороны компьютера — к порту USB. Программно на стороне компьютера прием и передача данных выполняется через виртуальный COM-порт.

Для передачи данных через последовательный порт в скетче следует воспользоваться объектом `Serial` и его методами. Перед началом отправки данных через последовательный порт в функции `setup()` следует добавить вызов метода `begin()`, имеющий следующий синтаксис

```
Serial.begin(<скорость передачи данных>);
```

Скорость передачи данных выбирается из ряда стандартных скоростей. Чаще всего при небольших объемах передаваемых данных задается скорость 9600 бод/с.

Для отправки данных на компьютер используется метод `println()`, вызов которой имеет следующий синтаксис

```
Serial.println(<данные для вывода в последовательный порт>);
```

Для отображения принятых данных на экране компьютера нужно открыть окно Монитор порта, нажав на кнопку в правом верхнем углу окна Arduino IDE (рисунок 4.3).

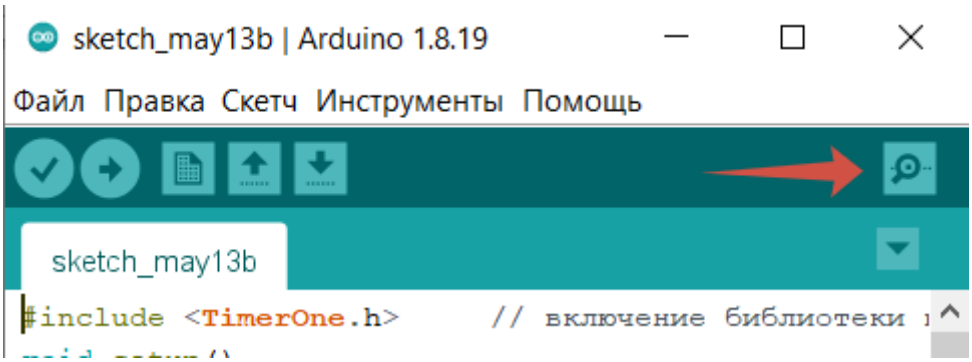


Рисунок 4.3 — Кнопка, открывающее окно Монитор порта

На рисунке 4.4 приведен скетч чтения положения переменного резистора с выводом значений в последовательный порт.

```
int var_res = 0;

void setup() {
  Serial.begin(9600);
}

void loop() {
  var_res = analogRead(A0);
  Serial.println(var_res);
}
```

Рисунок 4.4 — Скетч чтения положения переменного резистора с выводом значений в последовательный порт

В крайних положениях ручки переменного резистора в окне Монитор порта будут выводится значения 0 или 1023. В среднем положении ручки сопротивления обоих плеч делителя напряжения равны, поэтому в окне Монитор порта будут выводится значения 511 или 512.

Можно использовать переменный резистор, например, для управления яркостью светодиода (рисунок 4.5). Для этого следует подключить светодиод к цифровому порту платы Arduino, а в скетче следует добавить строки отправки на светодиод определенных значений при различных значениях переменной, соответствующей значению на выходе АЦП.

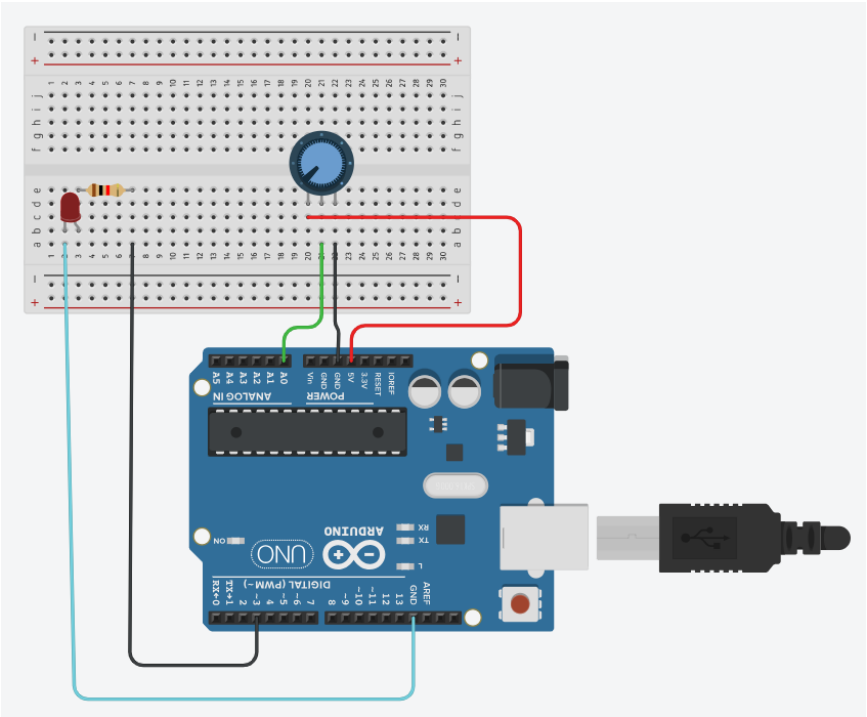


Рисунок 4.5 — Вариант размещения компонентов и проводников на беспаечной макетной плате для схемы с переменным резистором и светодиодом

Описанный выше способ подключения и считывания состояния переменного резистора может использоваться также и при подключении других резистивных датчиков. Например, фоторезистора — компонента, сопротивление которого изменяется в зависимости от освещенности его светочувствительной поверхности (рисунок 4.6). Чувствительность фоторезистора обусловлена свойствами полупроводникового материала, из которого он изготовлен. При увеличении освещенности сопротивление фоторезистора уменьшается, и наоборот.

При подключении к АЦП необходимо сформировать делитель напряжения путем последовательного подключения к фоторезистору резистора с соизмеримым сопротивлением. Средняя точка такого делителя напряжения подключается ко входу АЦП. Направление изменения сигнала на выходе делителя напряжения зависит от того, подключен ли фоторезистор в нижнее или верхнее плечо делителя напряжения. При подключении по схеме, показанной на рисунке 4.6, а, сигнал на выходе АЦП будет увеличиваться при увеличении сопротивления фоторезистора, что будет соответствовать уменьшению освещенности. При подключении по схеме, показанной на рисунке 4.6, б, сигнал на выходе АЦП будет уменьшаться при увеличении сопротивления фоторезистора.

а) б)

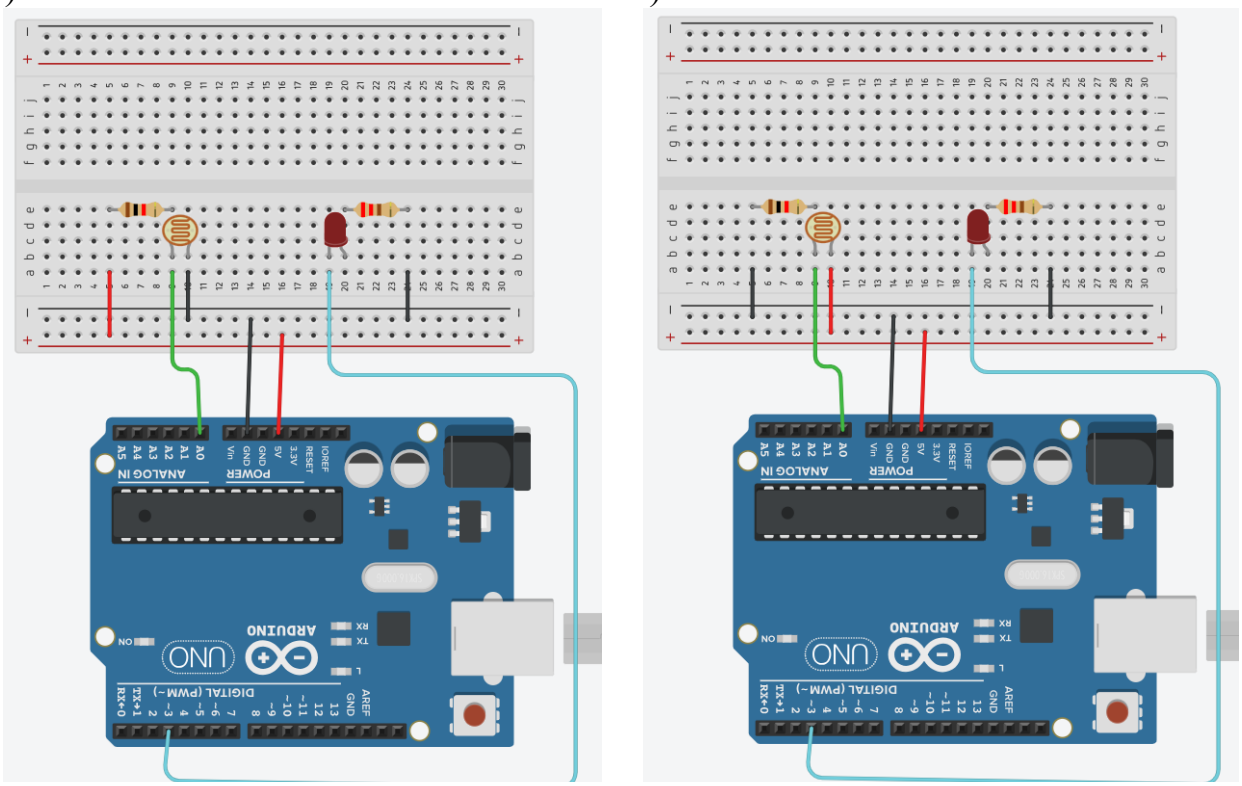


Рисунок 4.6 — Варианты размещения компонентов и проводников на беспаечной макетной плате для схемы с фоторезистором и светодиодом:
а — фоторезистор подключен к общей цепи;
б — фоторезистор подключен к напряжению питания

4.3. Задание на лабораторную работу

4.3.1. Соберите схему подключения переменного резистора и повторите скетч чтения положения переменного резистора и вывод значений в последовательный порт (рисунки 4.2, 4.4).

4.3.2. Добавьте в схему светодиод (рисунок 4.5), напишите и загрузите в плату Arduino программу, выполняющую управление яркостью светодиода с помощью переменного резистора.

4.3.3. Напишите и загрузите в плату Arduino программу, выполняющую управление периодом мигания светодиода с помощью переменного резистора.

4.3.4. Соберите схему подключения фоторезистора в соответствии с рисунком 4.6, а, напишите и загрузите в плату Arduino программу, выполняющую включение светодиода при затенении фоторезистора.

В качестве индивидуального задания приведите в отчет код скетча, в котором светодиод включается, если сигнал фоторезистора превышает *N*, где *N* — это номер студента в списке группы.

4.3.5. Соберите схему подключения фоторезистора в соответствии с рисунком 4.6, б, напишите и загрузите в плату Arduino программу, выполняющую плавную регулировку яркости светодиода по значениям фоторезистора.

4.4. Содержание отчета

При выполнении работы сохраняйте короткие видеозаписи с демонстрацией работы схем, которые затем следует преобразовать в формат .gif и приложить к отчету.

В отчете для общих и индивидуальных заданий следует привести:

— цель работы и формулировки заданий;
— фотографии собранных схем;
— тексты разработанных программ;
— подробные выводы о проделанной работе.

Файл отчета и файлы в формате .gif следует загрузить в соответствующий элемент в ЭОР do.sevsu.ru.

Лабораторная работа № 5
ВЫВОД ИНФОРМАЦИИ НА СЕМИСЕГМЕНТНЫЕ ИНДИКАТОРЫ

5.1. Цель работы

Целью работы является изучение способов вывода символов на семисегментные индикаторы.

5.2. Одноразрядный семисегментный индикатор

5.2.1. Устройство и схема подключения

Одноразрядный семисегментный индикатор (ОСИ) представляет собой микросборку, содержащую набор светодиодов, расположенных в удобной для отображения цифр и букв форме. При этом для уменьшения количества выводов внутри корпуса семисегментного индикатора светодиоды соединяются по схеме с общим катодом или с общим анодом (рисунок 5.1). Также часто семисегментные индикаторы содержат восьмой светодиод, предназначенный для отображения точки.

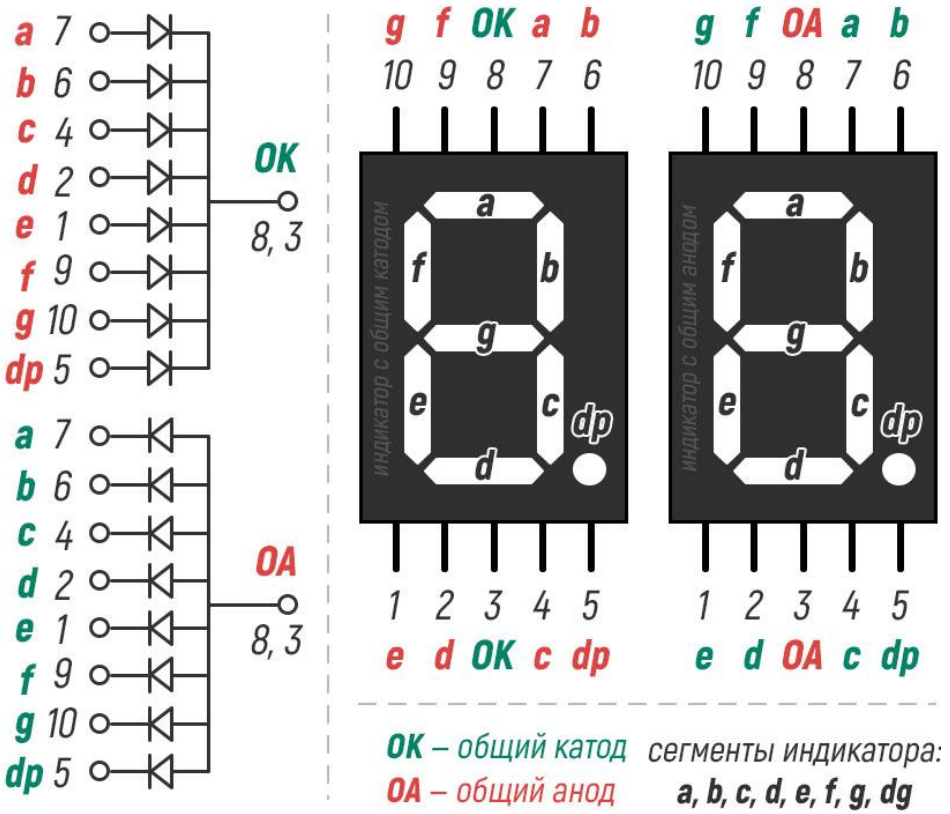
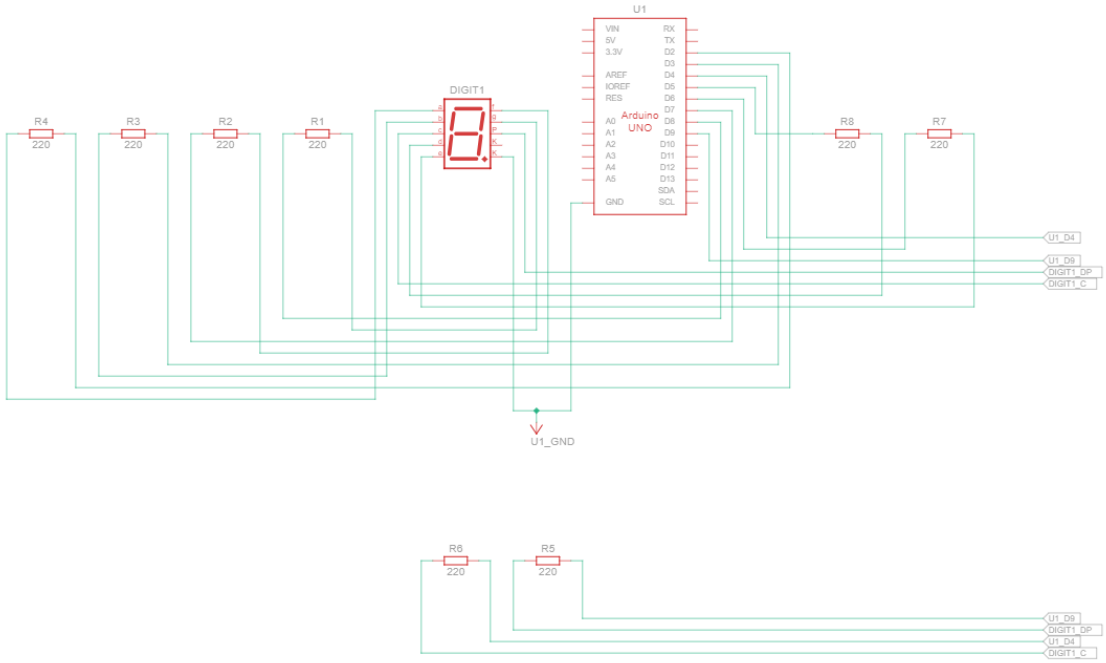


Рисунок 5.1 — Назначение выводов и эквивалентные схемы одноразрядных семисегментных индикаторов

Для управления ОСИ платой Arduino выводы сегментов подключаются к цифровым портам через резистор 220 Ом (рисунок 5.2).

а)



б)

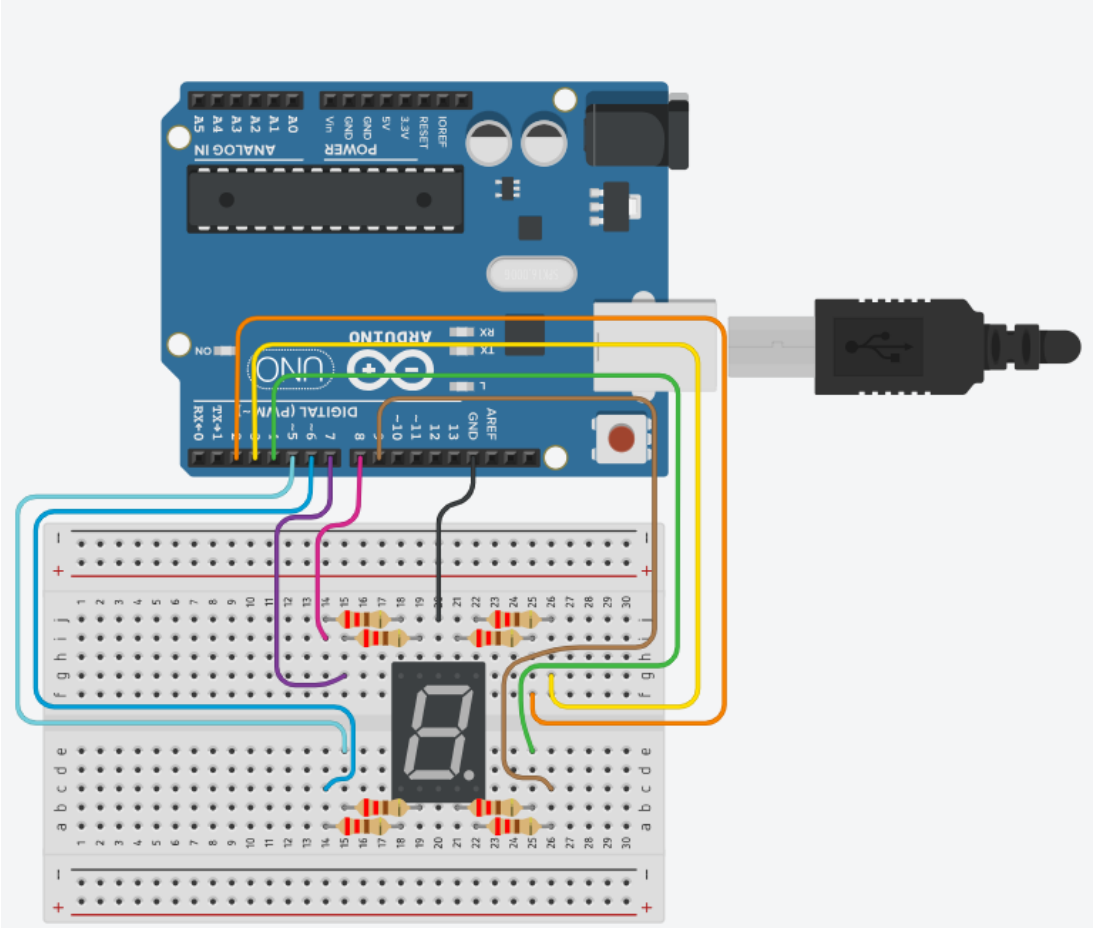


Рисунок 5.2 — Схема подключения одноразрядного семисегментного индикатора с общим катодом:
а — схема электрическая принципиальная; б — вариант размещения компонентов и проводников на беспаечной макетной плате

5.2.2. Управление сегментами

Для включения сегмента ОСИ с общим катодом на цифровом порту необходимо сформировать высокий логический уровень. При этом сам цифровой порт должен работать в режиме `OUTPUT`.

Для быстрой проверки правильности собранной схемы можно выполнить включение всех сегментов ОСИ. Для этого, используя циклы, переведите все задействованные в схеме цифровые порты в режим `OUTPUT` и выполните запись значений `HIGH` в цифровые порты с помощью функции `digitalWrite`.

5.2.3. Вывод символов

Для вывода символа на ОСИ следует сформировать на цифровых портах код символа — последовательность логических уровней, включающих и выключающих сегменты. На рисунке 5.3 показан фрагмент скетча, выполняющего вывод символа «1» на ОСИ.

```
void loop() {
  digitalWrite(2, LOW);
  digitalWrite(3, HIGH);
  digitalWrite(4, HIGH);
  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, LOW);
  digitalWrite(8, LOW);
  digitalWrite(9, LOW);
}
```

Рисунок 5.3 — Фрагмент скетча, выполняющего вывод символа «1» на ОСИ

Также для вывода символа на ОСИ удобно записать код символа в байт, значения битов в котором соответствуют состоянию сегментов ОСИ. Для чтения битов кода символа может использоваться функция `bitRead`, вызов которой имеет следующий синтаксис

```
<значение бита> = bitRead(<байт>, <номер бита>);
```

Номера битов начинаются с нуля; нулевой бит соответствует крайнему справа биту и является младшим битом байта. Пример скетча, в котором в цикле выполняется считывания кода символа из байта, показан на рисунке 5.4.

Коды символов, соответствующих десятичным цифрам, можно сохранить в виде массива байтов. Для улучшения читабельности исходного кода программы цикл вывода символа также может быть перенесен в функцию `displaySymbol` (рисунок 5.5).

Функция `displaySymbol` принимает код символа в виде байта и выполняет вывод символа на ОСИ путем циклического считывания битов кода и выдачи сигналов на цифровые порты путем вызова функции `digitalWrite`. Обратите внимание на то, что счетчик цикла `i` принимает значения от 2 до 9 в соответствии с номерами портов, к которым подключены сегменты ОСИ, а

номер бита в вызове функции `bitRead` изменяется от 0 до 7, что обеспечивается вычитанием 2 из `i`.

```
                //DpGFEDCBA
byte mask = B00000110;
// биты      76543210
void setup() {
  for (int i = 2; i < 10; i++)
  {
    pinMode(i, OUTPUT);
  }
}

void loop() {
  for (int i = 2; i < 10; i++)
  {
    digitalWrite(i, bitRead(mask, i - 2));
  }
}
```

Рисунок 5.4 — Запись кода символа ОСИ в байт и его считывание в цикле с помощью функции `bitRead`

```
byte symbols[] =
{
//DpGFEDCBA
  B00111111, // 0
  B00000110, // 1
  B01011011, // 2
  B01001111, // 3
  B01100110, // 4
  B01101101, // 5
  B01111101, // 6
  B00000111, // 7
  B01111111, // 8
  B01101111  // 9
};
void setup() {
  for (int i = 2; i < 10; i++)  {
    pinMode(i, OUTPUT);
  }
}
void loop() {
  displaySymbol(symbols[3]);
}
void displaySymbol(byte s)
{
  for (int i = 2; i < 10; i++)  {
    digitalWrite(i, bitRead(s, i - 2));
  }
}
```

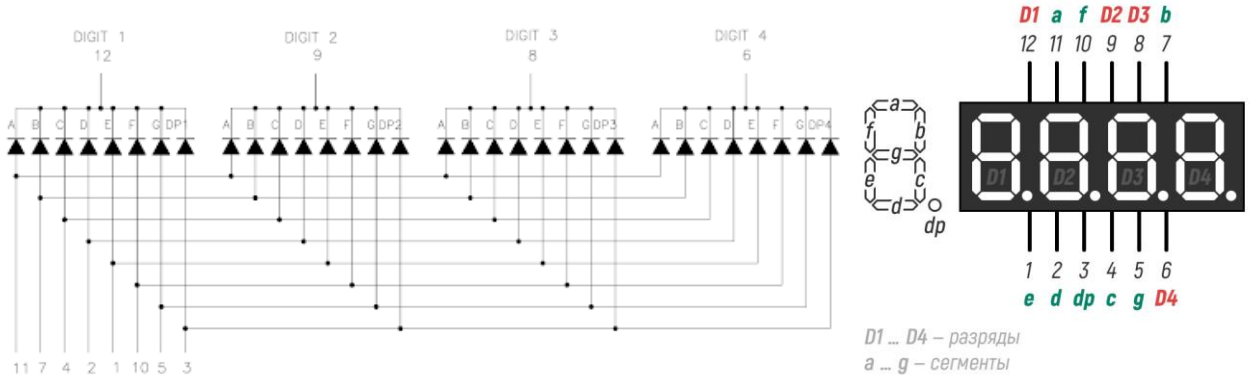
Рисунок 5.5 — Скетч вывода на ОСИ символа «3», код которого сохранен в массиве байтов

5.3. Четырехразрядный семисегментный индикатор

5.3.1. Устройство и схема подключения

Четырехразрядный семисегментный индикатор (ЧСИ) представляет собой микросборку, в корпусе которой объединены четыре ОСИ. Сегменты разрядов ЧСИ соединены параллельно. Каждый разряд ЧСИ имеет собственный общий вывод, подключенный к катодам светодиодов (рисунок 5.5).

а)



б)

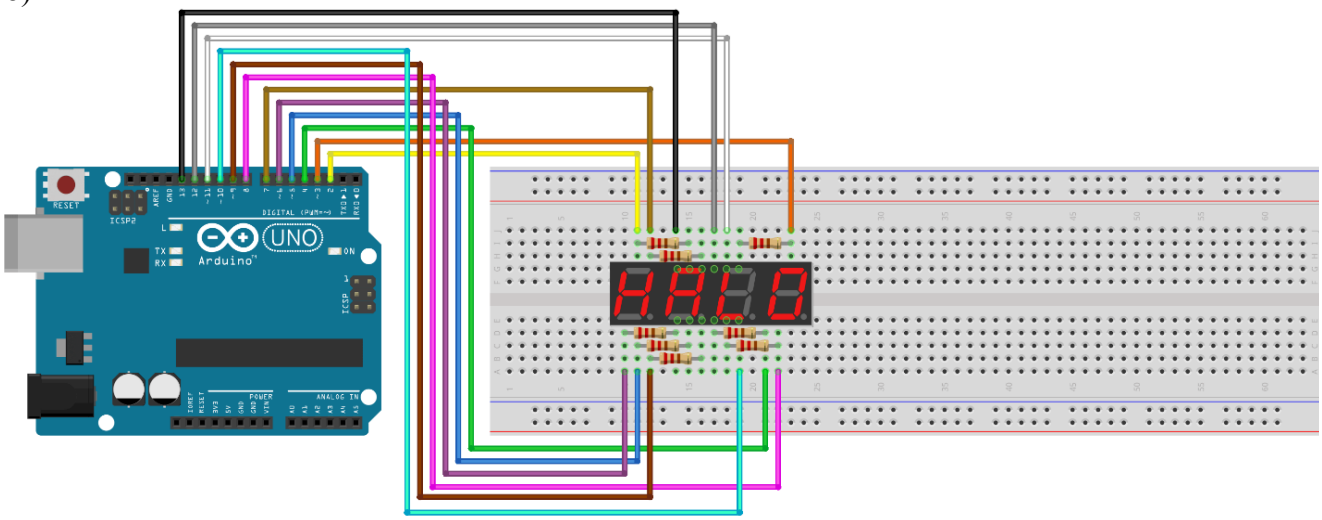


Рисунок 5.5 — Четырехразрядный семисегментный индикатор:
а — назначение выводов; б — вариант размещения компонентов и проводников на беспаечной макетной плате

5.3.2. Управление разрядами и сегментами

Для включения сегментов ЧСИ необходимо формировать положительное напряжение между выводами сегментов и общим выводом для разряда ЧСИ. Например, для того, чтобы включить сегмента А младшего разряда ЧСИ необходимо подать высокий уровень на цифровом порту №7 с одновременной подачей низкого уровня на цифровом порту № 10 (рисунок 5.5).

Для проверки правильность сборки схемы удобно загрузить в плату Arduino программу, выполняющую включение всех сегментов всех разрядов ЧСИ (рисунок 5.6). Если после загрузки программы не горит один из разрядов,

значит необходимо проверить правильность подключение и качество контактов для цифровых портов 10…13, если не горят сегменты — портов 2…9.

Рисунок 5.6 — Скетч включения всех сегментов всех разрядов ЧСИ

5.3.3. Динамическая индикация

Рассмотрим скетч последовательного включения разрядов и сегментов (рисунок 5.7), в котором для лучшего понимания работы программы и схемы в циклах добавлены задержки. Обратите внимание на однократное выключение всех сегментов и разрядов путем добавления двух циклов в функции `setup()`.

```
void setup() {
  for (int i = 2; i <= 13; i++)  {
    pinMode(i, OUTPUT);
  }
  for (int i = 10; i <= 13; i++) {
    digitalWrite(i, HIGH);
  }
  for (int i = 2; i <= 9; i++)   {
    digitalWrite(i, LOW);
  }
}

void loop() {
  for (int i = 10; i <= 13; i++)     // для всех разрядов
  {
    for (int i = 2; i <= 9; i++)   { // --для всех сегментов разряда
      digitalWrite(i, LOW);          // --выключить
    }
    digitalWrite(i, LOW);            // -включить разряд

    for (int i = 2; i <= 9; i++)   { // --для всех сегментов разряда
      digitalWrite(i, HIGH);         // --отобразить символ на сегментах
      delay(200);
    }
    delay(200);
    digitalWrite(i, HIGH);            // -выключить разряд
  }
}
```

Рисунок 5.7 — Скетч включения всех сегментов ЧСИ
с задержкой по сегментам и разрядам

Как видно из работы схемы, внутреннее устройство ЧСИ не позволяет в один момент времени выводить разные символы на несколько разрядов, поскольку выводы сегментов подключены ко всем разрядам параллельно. Для вывода разных символов на разряды ЧСИ следует применить динамическую индикацию.

Динамическая индикация — это метод отображения информации с помощью светодиодных индикаторов, при котором производится поочередное отображение символов каждого разряда в отдельности с частотой обновления, выбранной с учетом инерционности зрительной системы человека. Реализовать динамическую индикацию можно, если в скетче на рисунке 5.7 удалить задержку в цикле включения сегментов и уменьшить задержку, определяющую длительность отображения разряда. На рисунке 5.8 показан скетч, реализующий динамическую индикацию цифры 5 на все разрядах ЧСИ.

```
byte symbols[] = {
  //DpGFEDCBA
  B00111111, // 0
  B00000110, // 1
  B01011011, // 2
  B01001111, // 3
  B01100110, // 4
  B01101101, // 5
  B01111101, // 6
  B00000111, // 7
  B01111111, // 8
  B01101111  // 9
};
void displaySymbol(byte s) {
  for (int i = 2; i < 10; i++)  { digitalWrite(i, bitRead(s, i - 2)); }
}
void clearSegments() {
  for (int i = 2; i <= 9; i++)  { digitalWrite(i, LOW); }
}
void setup() {
  for (int i = 2; i <= 13; i++)  { pinMode(i, OUTPUT);    }
  for (int i = 10; i <= 13; i++) { digitalWrite(i, HIGH); }
  clearSegments();
}
void loop() {
  for (int i = 10; i <= 13; i++)    // для всех разрядов
  {
    clearSegments();
    digitalWrite(i, LOW);                    // -включить разряд
    displaySymbol(symbols[5])                // отобразить символ
    delay(2);
    digitalWrite(i, HIGH);                   // -выключить разряд
  }
}
```

Рисунок 5.8 — Скетч, реализующий динамическую индикацию цифры 5 на всех разрядах ЧСИ

По сравнению с примером кода на рисунке 5.7, в скетче на рисунке 5.8 также внесены следующие изменения:

— цикл выключения всех сегментов перенесен в функцию `clearSegments()`;

— для отображения символов реализована функция `displaySymbol()` и создан массив с кодами десятичных цифр в виде байтов.

5.3.4. Использование таймера

Для реализации таймеров, секундомеров или часов с помощью микроконтроллера, необходимо отсчитывать временные интервалы. Одним из возможных решений подобных задач является использование в программе переменных-счетчиков, значение которых изменяются после выполнения программных задержек с помощью функции `delay()`. Однако при применении динамической индикации такой подход неприменим — введение дополнительных задержек в цикл обновления индикаторов делает заметным последовательное обновление индикаторов. Поэтому при использовании динамической индикации применяются встроенные в микроконтроллер таймеры, формирующие периодические прерывания через интервалы времени, которые можно настраивать программно.

Для работы с встроенными таймерами микроконтроллера можно использовать библиотеку TimerOne.h. Для подключения библиотек к скетчу в Adruino IDE следует выбрать пункт меню *Скетч – Подключить библиотеку…*, и далее выбрать одну из перечисленных в списке библиотеку (рисунок 5.9).

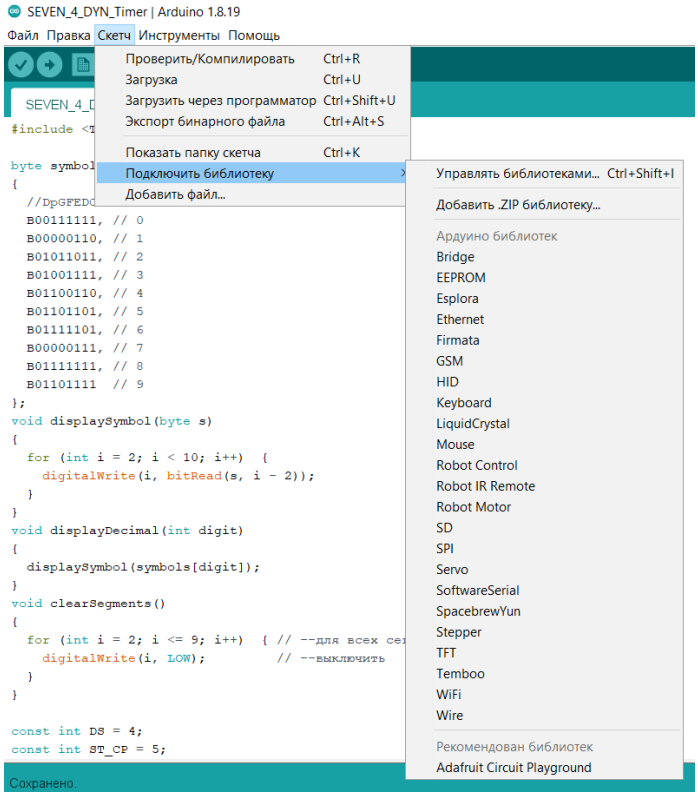


Рисунок 5.9 — Меню выбора библиотек в Arduino IDE

В случае отсутствия нужной библиотеки в списке, можно воспользоваться пунктом меню *Скетч – Подключить библиотеку… - Управление библиотеками*, после выбора которого откроется диалоговое окно Менеджер библиотек, в котором можно выполнить поиск библиотек в интернете (рисунок 5.10). Также отдельно скачанную библиотеку в виде zip-архива в Arduino IDE можно добавить, выбрав пункт меню *Скетч – Подключить библиотеку… - Добавить ZIP библиотеку*.

Добавим в Arduino IDE библиотеку TimerOne.h, выполнив ее поиск и установку в окне Менеджер библиотек (рисунок 5.10).

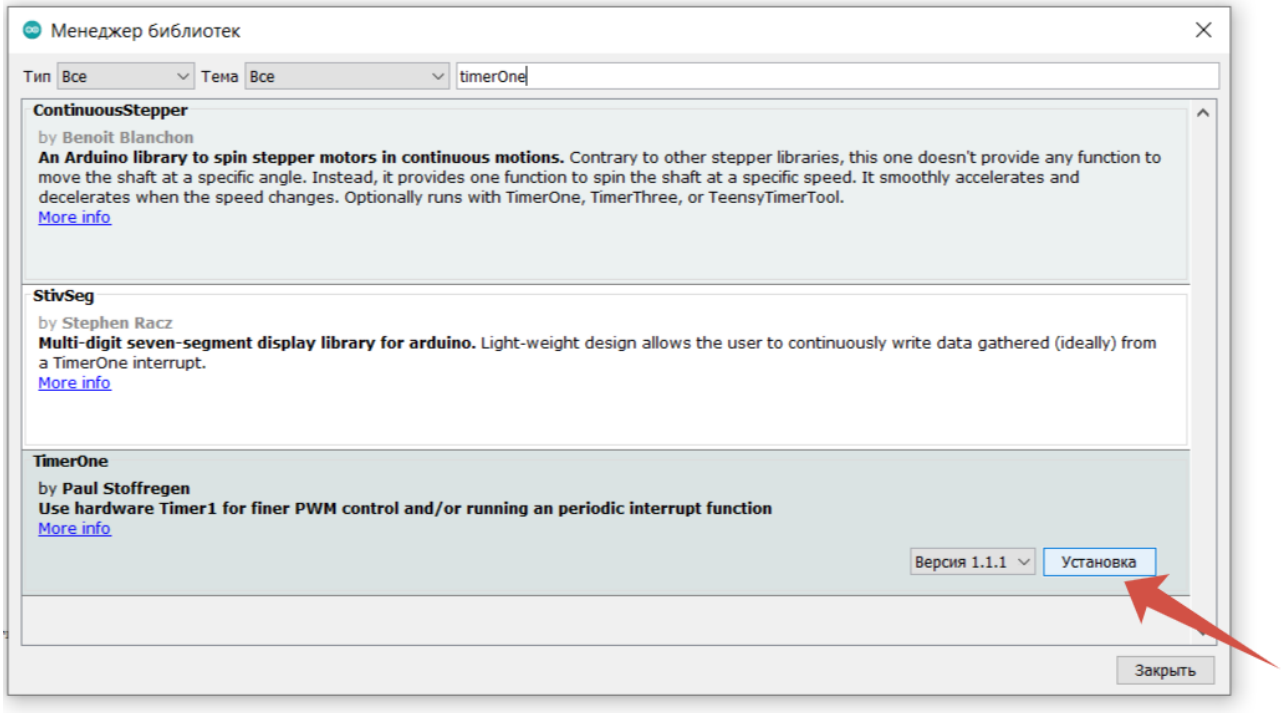


Рисунок 5.10 — Вид окна Менеджер библиотек Arduino IDE

Пример использования прерывания по таймеру для управление светодиодом показан на рисунке 5.11. Для использования таймера в скетче следует подключить библиотеку, с помощью директивы `#include`, после чего в пространстве имен скетча будет доступен объект таймера, который называется `Timer1`. В функции setup() следует выполнить инициализацию таймера с помощью метода initialize, вызов которого имеет следующий синтаксис

```
Timer1.initialize(<период срабатывания в микросекундах>);
```

Каждое срабатывание таймера будет приводить к возникновению прерывания, обработчик для которого указывается вызовом метода `attachInterrupt`

```
Timer1.attachInterrupt(<имя функции-обработчика прерывания>);
```

Реализация часов с помощью таймера с индикацией прошедшего времени на ЧСИ может быть выполнена путем создания переменной-счетчика, инкрементируемой в обработчике прерывания, значение которой затем

разделяется на единицы, десятки, сотни, тысячи секунд путем применения операций деления нацело и деления по модулю (рисунок 5.12).

```
#include <TimerOne.h>      // включение библиотеки в файл
void setup()
{
  Timer1.initialize(1000000);             // инициализация таймера с периодом, выраженным в мкс
  Timer1.attachInterrupt(Timer1_action);  // ссылка на функцию-обработчик прерываний

  pinMode(10, OUTPUT);
}
bool led = LOW;
void Timer1_action()
{
  led = !led;
}
void loop()
{
  digitalWrite(10, led);
}
```

Рисунок 5.11 — Пример использования прерывания по таймеру для управление светодиодом

```
#include <TimerOne.h>      // включение библиотеки в файл
void setup()
{
  Timer1.initialize(1000000);             // инициализация таймера с периодом, выраженным в мкс
  Timer1.attachInterrupt(Timer1_action);  // ссылка на функцию-обработчик прерываний
}
int cnt = 0;                              // переменная-счетчик секунд
void Timer1_action()
{
  cnt++;                                  // подсчет прошедших секунд
}
void loop()
{
  int ones     = cnt % 10;                // выделение разряда единиц
  int tens     = (cnt / 10) % 10;         // выделение разряда десятков
  int hundreds = (cnt / 100)% 10;         // выделение разряда сотен
}
```

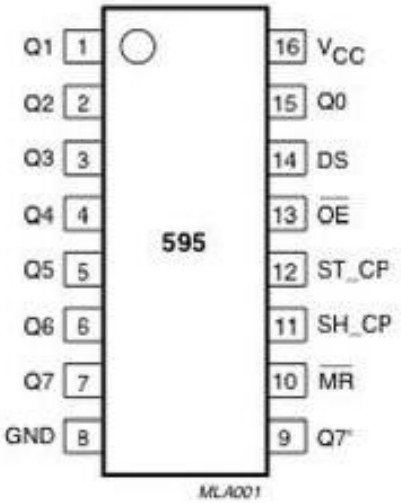
Рисунок 5.12 — Пример реализации часов с использованием переменной-счетчика секунд с выделением ее разрядов единиц, десятков и сотен

5.3.5. Подключение семисегментных индикаторов через сдвиговый регистр

Анализируя схемы на рисунках 5.2 и 5.5 можно заметить, что для управления семисегментными индикаторами требуется большое количество цифровых выводов микроконтроллера. В частности, в схеме на рисунке 5.5 задействованы 12 из 14 доступных на плате Arduino цифровых выводов.

Освободить часть цифровых выводов микроконтроллера для подключения других компонентов можно путем использования микросхемы сдвигового регистра. Сдвиговый регистр — это цифровая схема, позволяющая выполнять преобразование последовательного способа передачи данных в параллельный: данные в сдвиговый регистр загружаются последовательно бит

за битом, а выгружаются — параллельно и одновременно. На рисунке 5.13 показаны схема расположения и назначения выводов микросхемы сдвигового регистра 74HC595.



Номер вывода	Обозначение вывода	Назначение вывода
1…7, 15	Q0…Q7	Параллельный выход
8	GND	Общий
9	Q7''	Выход для последовательного соединения регистров
10	$\overline{MR}$	Master Reset: Сброс значений регистра с активным низким уровнем
11	SH_CP	Shift Register Clock Pin: Тактирования записи входных данных в регистр
12	ST_CP	Storage Register Clock Pin: Тактирования выдачи выходных данных
13	$\overline{OE}$	Output Enable: Сигнал, разрешающий выдачу выходных данных с активным низким уровнем
14	DS	D Serial: Вход последовательных данных
16	Vcc	Питание

Рисунок 5.13 — Назначение выводов микросхемы сдвигового регистра 74HC595

Временная диаграмма сигналов показана на рисунке 5.14. Данные загружаются в сдвиговый регистр на вход DS бит за битом в моменты положительных фронтов сигнала SH_CP. На выходных линиях Q0…Q7 данные появятся только в момент появления положительного фронта сигнала ST_CP.

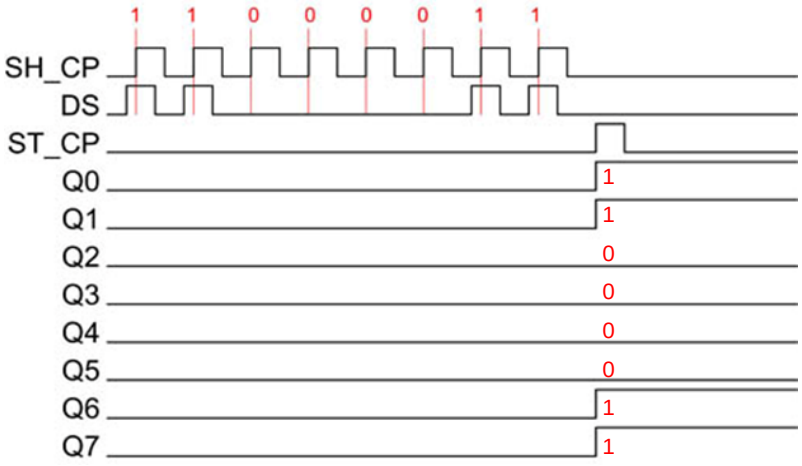


Рисунок 5.14 — Временная диаграмма сигналов микросхемы сдвигового регистра 74HC595

На рисунке 5.15 показан вариант размещения компонентов и проводников на макетной плате при подключении ЧСИ к плате Arduino через сдвиговый регистр. Обратите внимание на то, что сигнал сброса значений регистра подключен к напряжению питания, что соответствует постоянному не сброшенному состоянию, а сигнал, разрешающий выдачу выходных данных, подключен к общему проводу — выдача выходных сигналов всегда разрешена.

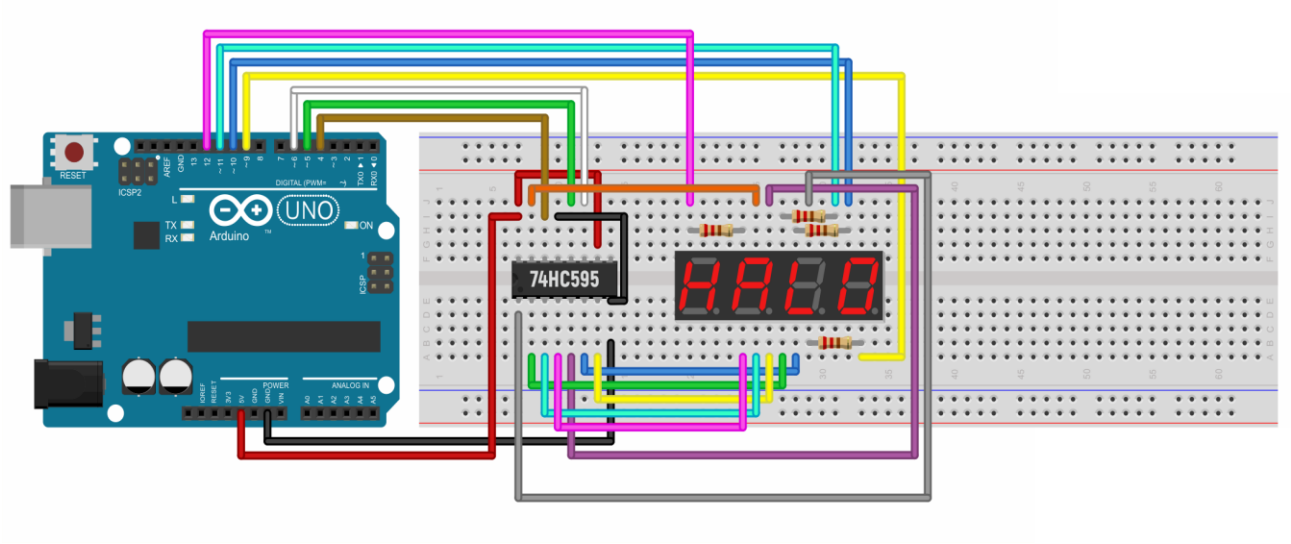


Рисунок 5.15 — Четырехразрядный семисегментный индикатор, подключенный к плате Arduino через сдвиговый регистр

По сравнению со схемой на рисунке 5.5 в схеме на рисунке 5.15 для управления ЧСИ используется только 7 цифровых выходов платы Arduino. Токоограничивающие резисторы включены в цепях катодов разрядов ЧСИ.

Для управления ЧСИ через сдвиговый регистр необходимо формировать данные и фронты сигналов в соответствии с временной диаграммой на рисунке 5.14. При этом ранее разработанный код для управления ЧСИ может быть модифицирован путем переопределения функций `displaySymbol()` и `clearSegments()`. На рисунке 5.16 показан фрагмент скетча с переопределенной функцией и функцией `setup()`. Фронты сигналов формируются в коде последовательными вызовами функции `digitalWrite()` с соответствующими аргументами.

Функция `clearSegments()` может содержать вызов функции `displaySymbol()`, отправляющей на все сегменты значение 0.

```
const int DS = 4;
const int ST_CP = 5;
const int SH_CP = 6;
void displaySymbol(byte s)
{
  for (int i = 7; i >= 0; i--)
  {
    // формируем бит кода индикатора
    digitalWrite(DS, bitRead(s, i));
    // формируем фронт входного тактового сигнала
    digitalWrite(SH_CP, LOW);
    digitalWrite(SH_CP, HIGH);
  }
  // формируем фронт выходного тактового сигнала
  digitalWrite(ST_CP, LOW);
  digitalWrite(ST_CP, HIGH);
}
void setup() {
  pinMode(DS, OUTPUT);
  pinMode(ST_CP, OUTPUT);
  pinMode(SH_CP, OUTPUT);
  for (int i = 9; i <= 12; i++)     // для всех разрядов
  {
    pinMode(i, OUTPUT);
    digitalWrite(i, HIGH);
  }
}
```

Рисунок 5.16 — Переопределение функции `displaySymbol()` при подключении ЧСИ через сдвиговый регистр

5.4. Задание на лабораторную работу

5.4.1. Соберите схему с одноразрядным семисегментным индикатором (п. 5.2.1).

5.4.2. Напишите и загрузите в плату Arduino программу, выполняющую включение всех сегментов ОСИ (п. 5.2.2).

5.4.3. **Напишите и загрузите в плату Arduino программу, выполняющую вывод символа *n* на ОСИ, где *n* — последняя цифра номера студента группы**.

5.4.4. Напишите и загрузите в плату Arduino программу, выполняющую вывод на ОСИ десятичного символа, код которого сохранен в массиве (п. 5.2.3).

5.4.5. Напишите и загрузите в плату Arduino программу «часы», выполняющую отсчет количества прошедших секунд от 0 до 9.

5.4.6. **Напишите и загрузите в плату Arduino программу «часы», выполняющую отсчет количества прошедших секунд от 0 до *n*, где *n* — последняя цифра номера студента группы**.

5.4.7. Дополните схему кнопкой, напишите и загрузите в плату Arduino программу «секундомер», выполняющую отсчет количества прошедших секунд от 0 до 9. Обработку нажатия кнопки следует производить с использованием внешнего прерывания (п. 3.6).

5.4.8. Соберите схему с ЧСИ (п. 5.3.1).

5.4.9. Напишите и загрузите в плату Arduino программу, выполняющую включение всех сегментов ЧСИ (п. 5.3.2).

5.4.10. Напишите и загрузите в плату Arduino программу, выполняющую включение всех сегментов ЧСИ с задержкой 200 мс по сегментам и разрядам (п. 5.3.2).

5.4.11. Напишите и загрузите в плату Arduino программу, выполняющую включение всех сегментов ЧСИ с задержкой 200 мс по разрядам и без задержки по сегментам. Подберите значение задержки, при которой все разряды отображают символы и мерцание ЧСИ незаметны глазу (п. 5.3.2).

5.4.12. Напишите и загрузите в плату Arduino программу, выполняющую динамическую индикацию четырех произвольных десятичных цифр на ЧСИ (п. 5.3.3).

5.4.13. Напишите и загрузите в плату Arduino программу «часы», выполняющую динамическую индикацию количества прошедших секунд от 0 до 9999. Для ускорения отладки программы допускается выполнять подсчет меньших интервалов времени. Для отсчета временных интервалов следует использовать прерывания по таймеру (п. 5.3.4).

5.4.14. Соберите схему ЧСИ со сдвиговым регистром. Напишите и загрузите в плату Arduino программу «часы» по п. 5.3.13. Дополните схему кнопкой, напишите и загрузите в плату Arduino программу «секундомер», выполняющую динамическую индикацию количества прошедших секунд от 0 до 9999 с остановкой счета при нажатии на кнопку (п. 5.3.5). **В качестве индивидуального задания выполните остановку счета спустя *N* секунд, где *N* — последняя цифра номера студента группы; приведите в отчет фотографию ЧСИ.**

5.5. Содержание отчета

При выполнении работы сохраняйте короткие видеозаписи с демонстрацией работы схем, которые затем следует преобразовать в формат .gif и приложить к отчету.

В отчете для общих и индивидуальных заданий следует привести:

— цель работы и формулировки заданий;

— фотографии собранных схем;
— тексты разработанных программ;
— подробные выводы о проделанной работе.

Файл отчета и файлы в формате .gif следует загрузить в соответствующий элемент в ЭОР do.sevsu.ru.

Лабораторная работа № 6
ВВОД ДАННЫХ С МАТРИЧНОЙ КЛАВИАТУРЫ

6.1. Цель работы

Целью работы является приобретение навыков чтения состояний кнопок матричной клавиатуры с использованием платы Arduino Uno и беспаечной макетной платы.

6.2. Матричная клавиатура

Во многих электронных средствах человеко-машинный интерфейс реализован в виде клавиатуры. При этом для уменьшения количества выводов, которые необходимо подключить к портам ввода-вывода микроконтроллера, проводники для подключения кнопок внутри клавиатур часто организуются в виде матрицы: кнопки располагаются на пересечении проводников строк и столбцов (рисунок 6.1, а). Количество портов ввода-вывода микроконтроллера в случае организации шестнадцати кнопок в виде матрицы равно не шестнадцати, а восьми.

Для того, чтобы определить какая из кнопок нажата, выполняется следующий алгоритм установки, считывания и сравнения сигналов: на один из проводников столбцов последовательно подается известный уровень напряжения, например, уровень логического нуля, после чего производится последовательное измерение уровня напряжения на всех проводниках строк (рисунок 6.1, б).

а) б)

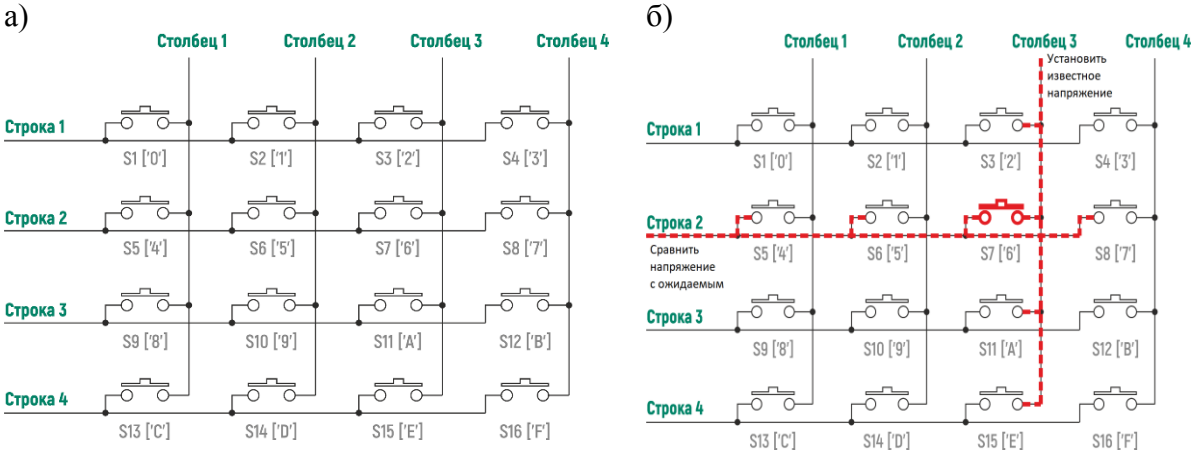


Рисунок 6.1 — Схема организации матричной клавиатуры 4×4 (а) и иллюстрация к определению расположения нажатой кнопки (б)

В случае обнаружения ожидаемого уровня напряжения, которое возникает вследствие нажатия кнопки, делается вывод о нажатии кнопки, находящейся на пересечении текущих столбца и строки. Важно чтобы при

выполнении такого алгоритма при подключении выводов строк использовались подтягивающие резисторы (см. рисунок 3.2).

На рисунке 6.2 показан вариант размещения компонентов и проводников при подключении матричной клавиатуры к плате Arduino.

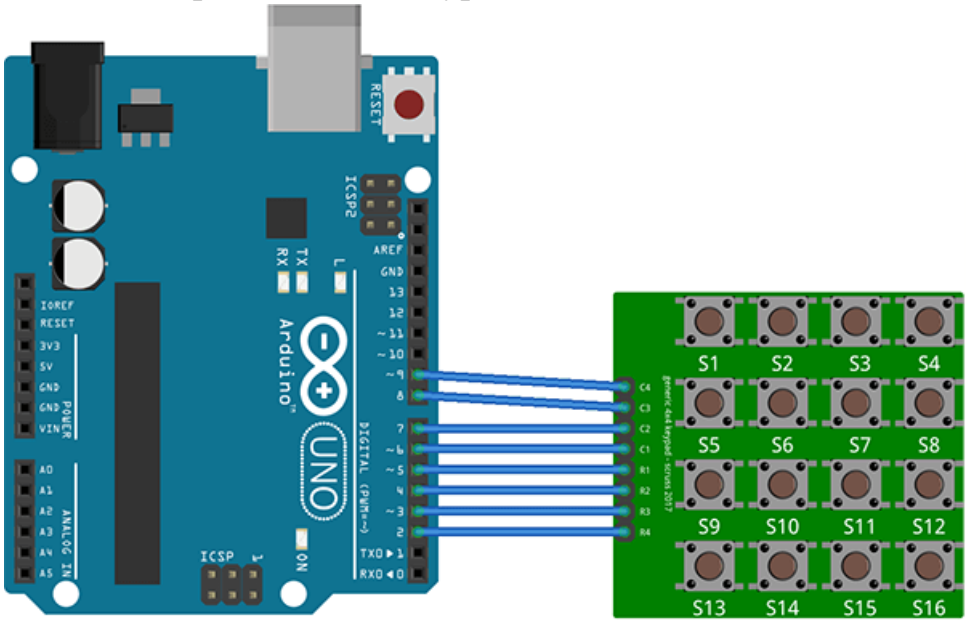


Рисунок 6.2 — Вариант размещения компонентов и проводников при подключении матричной клавиатуры к плате Arduino

Для работы с матричной клавиатурой в программе удобно объединить в массивы номера портов, управляющих строками и столбцами клавиатуры, а затем производить операции с портами в циклах. На рисунке 6.3 приведен фрагмент скетча, в котором выполняется назначение режимов работы портов с помощью функции `pinMode()`.

```
1  int cols[4] = { 6, 7, 8, 9 };
2  int rows[4] = { 5, 4, 3, 2 };
3  void setup() {
4    for (int i = 0; i < 4; i++)
5    {
6      pinMode(cols[i], OUTPUT);
7      pinMode(rows[i], INPUT_PULLUP);
8    }
```

Рисунок 6.3 — Фрагмент скетча, в котором номера портов, управляющих строками и столбцами, объединены в массивы

На рисунке 6.4 приведена реализация алгоритма считывания состояния кнопок клавиатуры с выводом номеров строки и столбца нажатой кнопки в монитор последовательного порта.

```
11  void loop() {
12    for (int col_i = 0; col_i < 4; col_i++){
13      digitalWrite(cols[col_i], LOW);
14      for (int row_i = 0; row_i < 4; row_i++)
15      {
16        if (digitalRead(rows[row_i]) == LOW)
17        {
18            Serial.print("Нажата кнопка ");
19            Serial.print(row_i);
20            Serial.print(", ");
21            Serial.println(col_i);
22            delay(300);
23        }
24      }
25      digitalWrite(cols[col_i], HIGH);
26    }
27  }
```

Рисунок 6.4 — Реализация алгоритма считывания состояния кнопок клавиатуры

Зачастую кнопки клавиатур имеют символьные обозначения, которые можно сохранить в двумерный массив (рисунок 6.5) и для считывания символа нажатой кнопки использовать номера строк и столбцов как индексы.

```
4  String symbols[4][4] = {
5    { "S1",  "S2",  "S3",  "S4"  },
6    { "S5",  "S6",  "S7",  "S8"  },
7    { "S9",  "S10", "S11", "S12" },
8    { "S13", "S14", "S15", "S16" },
9  };
```

Рисунок 6.5 — Символьные обозначения кнопок клавиатуры, сохраненные в массиве

В случае необходимости считывания последовательности символов, можно использовать методы типа `String`, в частности, метод `concat()`, выполняющий конкатенацию символов, вызов которого имеет следующий синтаксис

`<переменная типа String>.concat(<строковое значение>);`

На рисунке 6.6 показан фрагмент скетча, в котором выполняется добавление символа, соответствующего нажатой клавише, и пробела в переменную `code`.

```
 9  String code = "";
24          code.concat(symbols[row_i][col_i]);
25          code.concat(" ");
```

Рисунок 6.6 — Пример конкатенации строк

6.3. Задание на лабораторную работу

6.3.1. Соберите схему с матричной клавиатурой (рисунок 6.2).

6.3.2. Напишите и загрузите в плату Arduino программу, выполняющую считывание и вывод в последовательный порт номеров строки и столбца нажатой кнопки (рисунки 6.3, 6.4).

6.3.3. Измените программу из п. 6.3.2 таким образом, чтобы в последовательный порт выводился символ, соответствующий нажатой кнопке (рисунок 6.5).

6.3.4. Напишите и загрузите в плату Arduino программу, выполняющую считывание и вывод в последовательный порт последовательности из четырех символов (рисунок 6.6).

6.3.5. Добавьте в схему пьезокерамический излучатель (см. рисунок 2.2), напишите и загрузите в плату Arduino программу «электронного пианино», в которой при нажатии кнопок клавиатуры должны воспроизводиться восемь нот: от До первой октавы до До второй октавы (см. рисунок 2.5 и таблицу 2.1).

6.3.6. Реализуйте программу «электронный синтезатор», изменив звучание нот в программе п. 6.3.5 таким образом, чтобы при нажатии кнопок воспроизводились эффекты

— «вибрато» — высота звука должна меняться в некотором небольшом диапазоне вокруг частоты выбранной ноты;

— «арпеджио» — при нажатой кнопке воспроизводится последовательность звуков: с частотой выбранной ноты и с удвоенной частотой выбранной ноты.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. База знаний [Электронный ресурс] / AmperMarket.kz - Arduino и радиодетали. — 2025. — Режим доступа: https://ampermarket.kz/base/.

У ч е б н о е и з д а н и е

ПРОТОТИПИРОВАНИЕ ЭЛЕКТРОННЫХ СРЕДСТВ

Учебно-методическое пособие

Составитель: **Начаров** Денис Владимирович

В авторской редакции

Изд. № 36/2025. Объем 4 п.л. Усл. печ. л. 3,72. Уч.-изд. л. 3,92.
Издательский центр ФГАОУ ВО «Севастопольский государственный университет»