ЛАБОРАТОРНАЯ РАБОТА № 3

РАБОТА С АНАЛОГОВЫМИ СИГНАЛАМИ

Цель работы: знакомство с методами ввода аналоговых сигналов в Arduino, формирование ШИМ-сигналов с помощью цифровых выводов, изучение методов программирования аналоговых и цифровых выводов на ввод и вывод информации, программирование ШИМ-сигналов.

3.1. Теоретические сведения

Одноплатная ЭВМ Arduino имеет шесть аналоговых входов, позволяющих измерять напряжения в диапазоне от 0 до +5 В. На плате установлен шестиканальный 10-разрядный АЦП (аналого-цифровой преобразователь). Это означает, что АЦП может разделить аналоговый сигнал на 2^{10} различных состояний. Следовательно, Arduino может присвоить 2^{10} = 1 024 аналоговых значений, от 0 до 1 023. Уровни входного напряжения преобразуются в выходные дискретные цифровые коды, с которыми может оперировать микроконтроллер.

Для чтения значения напряжения аналогового входа предесмотрена функция `analogRead(), Serial.pritln()`. Программа определения величины напряжения сигнала, подаваемого на 0-й вход и вывода значений на экран компьютера, представлена в листинге 3.1.

Листинг 3.1. Программа определения входного напряжения и вывода значений на экран компьютера

```
// Программа чтения аналоговых данных
const int POT=0; // Аналоговый вход 0 для подключения
              потенциометра
int val = 0;    // Переменная для хранения значения потенциометра

void setup ()
{
    Serial.begin(9600);
}//символ запуска встроенного монитора последовательного порта
void loop()
{
    val = analogRead(POT);
    Serial.println(val);
    delay(500);
}
```

Сначала необходимо инициировать последовательное соединение, вызвав функцию `Serial.begin ()`, единственный аргумент которой задает скорость передачи данных в бодах. В наших примерах выбрана скорость 9 600 бод.

В каждой итерации цикла переменная `val` получает аналоговое значение, считанное командой `analogRead ()` с входа, соединенного со средним контактом потенциометра (в нашем случае это вход A0). Далее это значение функция `Serial.println()` выводит в последовательный порт, соединенный с компьютером. Затем следует задержка в полсекунды.

После загрузки на плату Arduino можно заметить, что светодиод TX, расположенный на плате, мигает каждые 500 мс. Этот индикатор показывает, что плата Arduino передает данные через последовательный USB-интерфейс на компьютер. Для просмотра данных подойдут любые терминальные программы, но в Arduino IDE есть встроенный монитор последовательного порта, для запуска которого нажмите кнопку на знак `}`, помеченный комментарием на листинге 3.1.

После запуска монитора последовательного порта на экране компьютера появляется окно с отображением потока передаваемых чисел. Изменяя величину входного напряжения, можно увидеть, что выводимые значения меняются между значениями 0 и 1 023.

Одноплатная ЭВМ Arduino не имеет аналоговых выходов, но имеет возможность сымитировать генерацию аналоговых значений на цифровых контактах с помощью широтно-импульсной модуляции (ШИМ). Для некоторых контактов Arduino сформировать ШИМ-сигнал можно командой `analogWrite ().` Контакты, которые могут выдавать ШИМ-сигнал на определенные периферийные устройства, помечены символом ~ на плате. Выдачу ШИМ-сигнала поддерживают 3, 5, 6, 9, 10 и 11 контакты. Проверить команду `analogWrite ()` можно с помощью схемы, показанной на рис. 2.2, используя 9-й вывод. Если уменьшить напряжение на контакте 9, яркость свечения светодиода должна стать меньше, потому что снизится ток, текущий через него. Этого эффекта можно добиться с помощью ШИМ и команды `analogWrite ().`

Функция `analogWrite ()` имеет два аргумента: номер контакта и 8-разрядное значение в диапазоне от 0 до 255, устанавливаемое на этом контакте.

В листинге 3.2 приведен код программы генерации ШИМ-сигнала на контакте 9 для плавного управления яркостью светодиода.

Листинг 3.2. Управление яркостью свечения светодиода

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


void setup ()
{
    pinmode (LED, OUTPUT);     // Конфигурируем контакт светодиода
                                как выход
}
void loop()
{
    for (int i=0; i<256; i++)
    {
        analogWrite(LED, i);
        delay (10);
    }
    for (int i=255; i>=0; i--)
    {
        analogWrite(LED, i);
        delay(l0);
    }
}
```

При выполнении программы свечение светодиода изменяется от тусклого к яркому в одном цикле *for*, а затем от яркого к тусклому в другом цикле *for*. Все это будет происходить в основном цикле *loop ()* до бесконечности. Обязательно обратите внимание на различие двух циклов *for*. В первом цикле выражение *i++* является сокращением кода *i=i+l*. Аналогично, запись *i--* эквивалентна коду *i=i-1*. Первый цикл плавно зажигает светодиод до его максимальной яркости, второй – постепенно гасит его.

3.2. Порядок выполнения работы

1. Собрать схему, представленную на рис. 2.2.
2. Составить программу управления яркостью свечения светодиода на основе примера, показанного на листинге 3.2. Отладить программу, записать ее в Arduino, запустить программу. Сохранить программу. В программе обязательны комментарии.
3. Собрать схему, представленную на рис. 3.1. Подключите один крайний вывод потенциометра к земле, другой к шине 5 В. Потенциометры симметричны, так что не имеет значения, с какой стороны вы подключите шину питания, а с какой – землю. Средний вывод соеди-

ните с аналоговым контактом 0 на плате Arduino. При повороте ручки потенциометра аналоговый входной сигнал будет плавно изменяться от 0 до 5 В. В этом можно убедиться с помощью мультиметра. Переведите мультиметр в режим измерения напряжения, подсоедините его и следите за показаниями, поворачивая ручку потенциометра. Красный (положительный) щуп мультиметра должен быть подключен к среднему контакту потенциометра, а черный (отрицательный) щуп – к земле.

4. Составить программу определения входного напряжения и вывода значений на экран компьютера на основе примера, показанного на листинге 3.1. Отладить программу, записать ее в Arduino, запустить. Сохранить программу. В программе обязательны комментарии. Снять скриншот программы и скриншот экрана компьютера с данными, полученными от Arduino.

5. Собрать схему управления яркостью свечения светодиода с управлением от потенциометра. Собрать схему, представляющую комбинацию схем, показанных на рис. 2.2 и 3.1.

6. Составить программу управления яркостью свечения светодиода с управлением от потенциометра. Отладить программу, скомпилировать, загрузить ее в Arduino и запустить. В программе обязательны комментарии.

7. Собрать схему, отображающую с помощью трех светодиодов уровень напряжения на входе, для чего соберите схему, представляющую собой комбинацию схем, показанных на рис. 2.7. и 3.1.

8. Составить программу, индицирующую уровни входного напряжения, подаваемого от потенциометра: 0-1,6 В – светится синий светодиод, 1,61-3,2 В – светится зеленый светодиод, 3,21-5 В – светится красный светодиод. Отладить программу, скомпилировать, загрузить ее в Arduino и запустить. В программе обязательны комментарии. Сохранить программу.

9. Составить индивидуальную программу управления светодиодами для каждого варианта. При увеличении входного напряжения от потенциометра необходимо включать светодиоды в последовательности, указанной в таблице 2.1.

10. Составить программу управления светодиодами от кнопки аналогичную программе, составленной в п. 7 второй лабораторной работы. Ввести дополнительную функцию – уровень яркости свечения текущего светодиода будет определяться величиной напряжения, подаваемого от потенциометра. Отладить программу, скомпилиро-

вать, загрузить ее в Arduino и запустить. В программе обязательны комментарии. Сохранить программу.

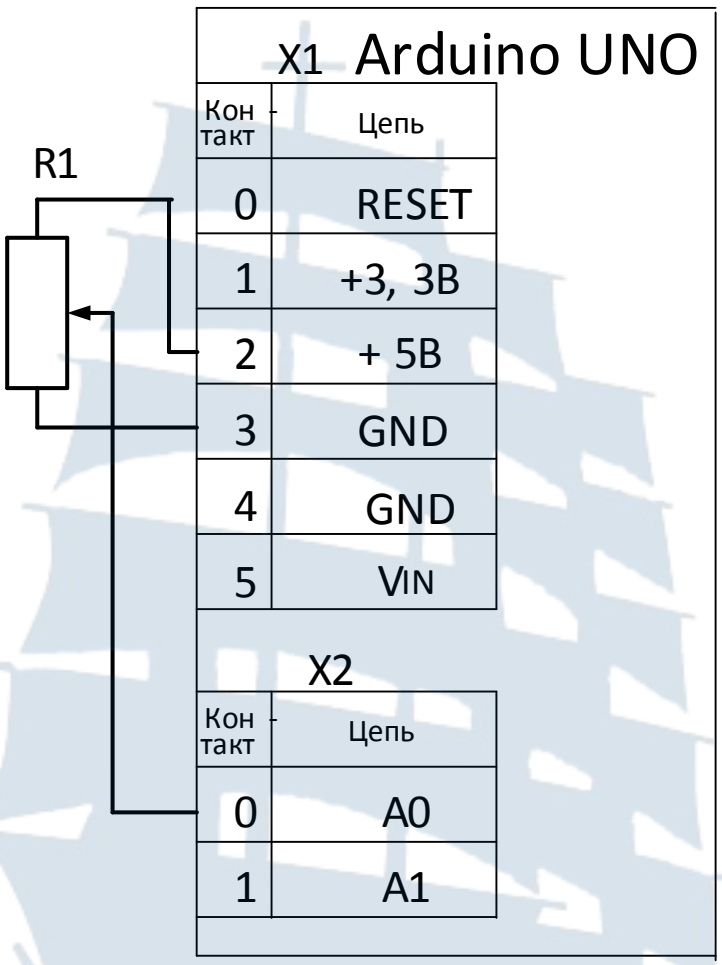

Рис. 3.1. Схема ввода аналогового сигнала с помощью потенциометра

3.3. Содержание отчета

1. Принципиальные схемы, изучаемые в данной лабораторной работе.
2. Фотографии собранных схем.
3. Листинги программ с комментариями.
4. Скриншот экрана компьютера с данными, которые передаются от Arduino.
5. Графические схемы алгоритмов разработанных программ.

3.4. Контрольные вопросы

1. Чем отличаются аналоговые сигналы от цифровых?
2. Как преобразовать аналоговые сигналы в цифровые?
3. Как считать аналоговый сигнал с потенциометра?

4. Как вывести на экран данные, используя монитор последовательного порта?

5. Как ограничить значения для управления аналоговыми выходами?

6. Что такое ШИМ-сигнал?

7. Как управлять с помощью ШИМ-сигнала уровнем напряжения на цифровом выходе?

8. Как величина аналогового выходного напряжения соотносится в параметрами функции `analogWrite()` и с напряжением питания платы?

9. Что такое потенциометр, каков принцип работы потенциометра?

10. Какое напряжение на среднем контакте потенциометра?

11. От чего зависит время пребывания светодиода во включенном или выключенном состоянии?

12. Зачем нужна встроенная функция `analogRead()`? Какие параметры она принимает?

13. Поясните назначение и синтаксис функции `analogWrite()`.

14. Как согласовать разрешающую способность АЦП (10 бит) и ШИМ-выхода (8 бит)?

ЛАБОРАТОРНАЯ РАБОТА № 4

РАБОТА ARDUINO UNO СО ЗВУКОМ

Цель работы – знакомство с методами генерирования звуковых сигналов с помощью одноплатной ЭВМ Arduino, изучение возможностей функции `tone`, способов формирования высоты и длительности звуков, подключение заголовочных файлов к программе.

4.1. Теоретические сведения

Генерировать звук с помощью Arduino можно несколькими способами. Самый простой способ – использование функции `tone()`, которую рассмотрим в данной лабораторной работе.

Звук будем воспроизводить с помощью пьезоизлучателя. Пьезокерамические излучатели (пьезоизлучатели) – электроакустические устройства воспроизведения звука, использующие пьезоэлектрический эффект (эффект возникновения поляризации диэлектрика под действием механических напряжений (прямой пьезоэлектрический эффект). Существует и обратный пьезоэлектрический эффект – возникновение механических деформаций под действием электрического поля.

Прямой пьезоэффект используется в пьезозажигалках, для получения высокого напряжения на разряднике; обратный пьезоэлектрический эффект используется в пьезоизлучателях (эффективны на высоких частотах и имеют небольшие габариты).

Пьезоизлучатели широко используются в различных электронных устройствах – часах-будильниках, телефонных аппаратах, электронных игрушках, бытовой технике. Пьезокерамический излучатель состоит из металлической пластины, на которую нанесён слой пьезоэлектрической керамики, имеющий на внешней стороне токопроводящее напыление. Пластина и напыление являются двумя контактами. Пьезоизлучатель также может использоваться в качестве пьезоэлектрического микрофона или датчика.

Для генерации звуков на Arduino существует функция `tone()`, которая генерирует сигнал прямоугольной формы с заданной частотой. Длительность может быть задана параметром. Без указания длительности сигнал генерируется, пока не будет вызвана функция `noTone()`. К порту Arduino может быть подключен пьезо- или другой высокоомный динамик для воспроизведения сигнала. Одновременно может воспроизводиться только один сигнал.

Синтаксис функции tone():

- tone(pin, частота)
- tone(pin, частота, длительность)

Листинг 4.1. Пример использования функции tone()

```
const int SoundPin = 9; // Пин подключения пьезоизлучателя - 9 дискретный
int DelaySound = 1000; // Пауза 1 секунда
void setup()
{
}
void loop()
{
                        // Пример использования tone()
                        //tone(pin, частота)
    tone(SoundPin, 1915);  //Звучит сигнал с частотой 1915 Гц
    delay(DelaySound);  // Пауза 1 секунда (1000 миллисекунд - значение переменной DelaySound ) - длительность воспроизведения сигнала
    tone(SoundPin, 1700);
    delay(DelaySound);
    tone(SoundPin, 1519);
    delay(DelaySound);
    tone(SoundPin, 1432);
    delay(DelaySound);
    tone(SoundPin, 1275);
    delay(DelaySound);
    tone(SoundPin, 1136);
    delay(DelaySound);
    tone(SoundPin, 1014);
    delay(DelaySound);
    noTone(9); // Выключаем звук
}
```

Иногда для удобства хранения последовательности нот используется массив tones. Вывод последовательности звуков реализуется простым циклом перебора массива нот с отправкой текущего значения на динамик.

Массив представляет собой упорядоченную последовательность элементов одного типа, которые связаны между собой. Массив можно представить как нумерованный список. Каждый элемент массива имеет индекс, который указывает его местоположение в списке. В первом примере в массиве хранится звуковая последовательность – список нот, которые будем воспроизводить по порядку.

В Arduino при объявлении массива необходимо указывать его размер. Это делается либо явно указав размерность массива, либо за-

полнив массив всеми значениями. При необходимости можно инициализировать массив значениями при объявлении.

При объявлении массива таким способом указывать число элементов массива необязательно; предполагается, что длина массива равна числу объявленных элементов. Обратите внимание, что массивы индексируются с нуля. Доступ к элементу массива можно получить, поставив после имени массива в квадратных скобках соответствующее значение индекса. Например, если требуется установить яркость светодиода, подключенного к выводу 9 Arduino, равной значению третьего элемента в массиве, то можно сделать это следующим образом:

```
analogWrite(9, numbers [2]);
```

Обратите внимание, что индекс 2 представляет собой третье значение в массиве, поскольку нумерация начинается с нуля. Изменить одно из значений массива можно так:

```
numbers [2] = 10;
```

Далее массивы потребуются нам, чтобы создать структуру, которая может содержать последовательность нот, воспроизводимую на динамике. В следующем примере будут проигрываться десятки нот, и без массива программа стала бы слишком громоздкой и непонятной.

Листинг 4.2. Программа с использованием массива `tones`

```
const int dynPin = 2;
int numTones = 10;
// Ноты C,C#,D,D#,E,F,F#,G,G#,A
int tones[10] = {261, 277, 294, 311, 330, 349, 370, 392, 415,
440};

void setup()
{
    pinMode( dynPin, OUTPUT );
}

void loop(){
    for( int i = 0; i < numTones; i++ )
    {
        tone( dynPin, tones[i] );
        delay( 500 );
    }
    noTone( dynPin );
}
```

При составлении программ, воспроизводящих звук, удобно создать заголовочный файл, определяющий частоты для музыкальных нот. Это делает программу более понятной при составлении простых музыкальных мелодий.

При обозначении музыкальных знаков ноты обозначаются буквами. В Arduino IDE есть специальный файл, содержащий значения частот для всех нот. В Arduino IDE создайте пустой новый файл. Как вы, наверное, заметили, Arduino IDE создает новый файл внутри папки с одноименным названием. Добавляя в эту папку новые файлы, вы можете включать их в свою программу, в результате код будет лучше структурирован. Скопируйте файл pitches.h в папку, созданную Arduino IDE, для нового проекта. Теперь заново откройте в Arduino IDE этот файл. Обратите внимание на содержимое файла pitches.h (рис. 4.1).

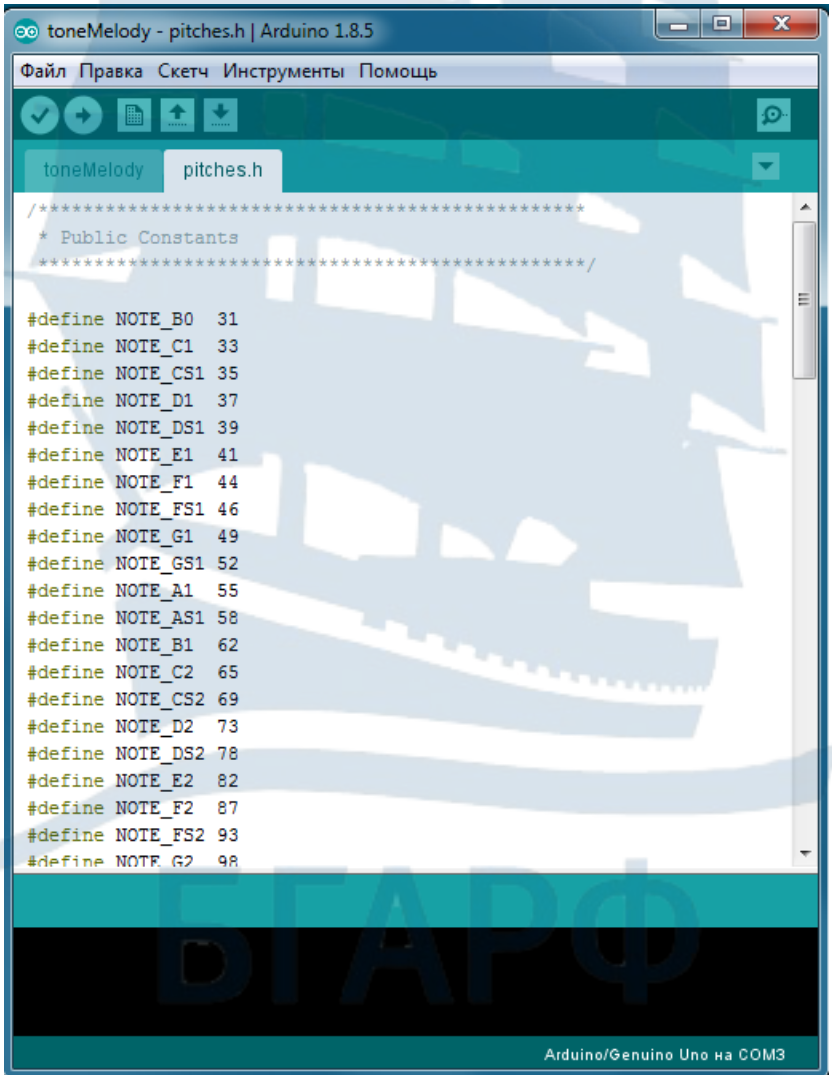

Рис. 4.1. Содержимое файла pitches.h

Перейдите на вкладку pitches.h, чтобы увидеть содержимое файла. Обратите внимание, что это всего лишь список операторов определений, которые задают соответствие названий нот и значений частот. Чтобы использовать эти определения при компиляции программы для Arduino, необходимо сообщить компилятору, где искать данный файл. Сделать это легко. Просто добавьте соответствующую строку кода в начало файла * .ino:
#include "pitches.h"

Для компилятора это, по существу, то же самое, что копирование и вставка содержимого файла заголовка в начало основного файла. Тем не менее, код становится аккуратнее и проще для чтения, при написании программ рекомендуем использовать данный заголовочный файл для определения высоты тона (частоты ноты).

В следующем примере приведена программа воспроизведения звука с помощью заголовочного файла.

Листинг 4.3. Программа воспроизведения звука с использованием заголовочного файла

```
// Проигрывание мелодии на динамике
#include "pitches.h" // Заголовочный файл со значениями частоты нот
const int SPEAKER=9; // Вывод подключения динамика
// Массив нот
int notes [] =
{
NOTE_A4, NOTE_E3, NOTE_A4, 0,
NOTE_A4, NOTE_E3, NOTE_A4, 0,
NOTE_E4, NOTE_D4, NOTE_C4, NOTE_B4, NOTE_A4, NOTE_B4, NOTE_C4,
NOTE_D4, NOTE_E4, NOTE_E3, NOTE_A4, 0
} ;

// Массив длительностей звучания нот в мс
int times [] =
{
250, 250, 250, 250,
250, 250, 250, 250,
125, 125, 125, 125, 125, 125, 125, 125,
250, 250, 250, 250
} ;
void setup ()
{
// Выбор каждой ноты перебором массива нот
    for (int i = 0; i < 20; i++)
    {
        tone(SPEAKER, notes[i], times[i]);
```

```
        delay(times[i]);
    }
}
void loop()
{
// Чтобы повторить воспроизведение, необходимо нажать кнопку
Reset
}
```

Если необходимо создать свою собственную мелодию, нужно проследить, чтобы массивы нот и длительностей имели равный размер, и правильно задать верхнюю границу для цикла перебора `for ()`. Поскольку функция `tone ()` может работать в фоновом режиме, важно определить задержку `delay ()`, чтобы следующая нота не звучала, пока не закончится воспроизведение предыдущей.

4.2. Порядок выполнения работы

1. Подключите пьезоизлучатель к одному из цифровых выходов.
2. Составить программу на основе листинга 4.1. Отладить программу, записать ее в Arduino, запустить программу.
3. Составить программу «Электронный метроном». Метроном – это устройство, которое задает ритм для музыканта. Необходимо составить программу, позволяющую издавать краткий звук с заданным периодом, равным одной секунде. Частоту генерируемого звука необходимо сделать равной 100 Гц.

Как только вызывается функция `tone()`, Arduino начнет генерировать импульсный сигнал и будет делать это, пока принудительно не выключится генерация с помощью другой функции – `noTone(контакт)`, где аргумент `контакт` – это номер вывода Arduino, к которому подключён динамик.

Примечание. Важно учитывать, что Arduino может одновременно генерировать только один тон на одном контакте. Если вызовем функцию `tone()` для одного контакта, и пока идет генерация, попытаемся вызвать `tone()` для другого контакта, то последний вызов будет попросту проигнорирован.

Программа метронома идентична программе для мигания светодиодом, за исключением того, что мы вместо функции `digitalWrite` применяем `tone()` и `noTone()`.

Отладить программу, записать ее в Arduino, запустить программу. Сохранить программу. В программе обязательны комментарии.

4. Составить модернизированную программу «Электронный метроном». Задать в программе циклическое увеличение воспроизводимого звука от 100 до 1 000 Гц с одновременным уменьшением времени

звучания каждого звука. Отладить программу, записать ее в Arduino, запустить программу. Сохранить программу. В программе обязательны комментарии.

5. Составить программу на основе листинга 4.2. Отладить программу, записать ее в Arduino, запустить программу.

6. Составить программу воспроизведения известной мелодии с использованием двух массивов tones. В первом массиве содержится информация о нотах, во втором – о длительности нот.

```
// массив 1 – частоты нот
int tones[COUNT_NOTES] = {
 392, 392, 392, 311, 466, 392, 311, 466, 392,
 587, 587, 587, 622, 466, 369, 311, 466, 392,
 784, 392, 392, 784, 739, 698, 659, 622, 659,
 415, 554, 523, 493, 466, 440, 466,
 311, 369, 311, 466, 392
};
// массив 2 - длительности нот
int durations[COUNT_NOTES] = {
 350, 350, 350, 250, 100, 350, 250, 100, 700,
 350, 350, 350, 250, 100, 350, 250, 100, 700,
 350, 250, 100, 350, 250, 100, 100, 100, 450,
 150, 350, 250, 100, 100, 100, 450,
 150, 350, 250, 100, 750
};
```

Отладить программу, записать ее в Arduino, запустить программу. Сохранить программу. В программе обязательны комментарии.

7. Составить программу воспроизведения звука с использованием заголовочного файла на основе листинга 4.3. Отладить программу, записать ее в Arduino, запустить программу. Сохранить программу. В программе обязательны комментарии.

8. Найти ноты и длительности их звучания одной из мелодий в соответствии с заданным вариантом. Составить программу воспроизведения звука заданной мелодии с использованием заголовочного файла. Отладить программу, записать ее в Arduino, запустить программу. Сохранить программу. В программе обязательны комментарии.

Таблица 4.1.

Варианты выполнения задания 8

Номер варианта	Мелодия
1	Песенка Красной шапочки
2	В лесу родилась елочка
3	Маленькая елочка
4	Антошка
5	Пусть бегут неуклюже пешеходы по лужам
6	Жил был у бабушки серенький козлик
7	Песенка друзей (Бременские музыканты)
8	Пусть всегда будет солнце
9	Оранжевая песня
10	Буратино
11	Танец маленьких утят
12	Улыбка

4.3. Содержание отчета

1. Фотография собранной схемы.
2. Листинги программ с комментариями.
3. Скриншот файла pitches.h.
4. Графические схемы алгоритмов разработанных программ.

4.4. Контрольные вопросы

1. Как динамики создают вибрацию воздуха, которая распространяется в пространстве и воспринимается нашей барабанной перепонкой в виде звука.
2. Что такое пьезоэлемент, как он работает?
3. Как создавать звуки произвольной частоты и длительности с помощью функции tone().
4. Как язык программирования Arduino поддерживает массивы, используемые для перебора последовательностей данных.
5. Как можно регулировать громкость воспроизведения в Arduino?
6. Почему к выходу Arduino нельзя подключать напрямую высокоомный динамик?
7. Что такое заголовочный файл, каково содержимое файла pitches.h, в чем состоит правило его использования?
8. Зачем нужна встроенная функция tone()? Какие параметры она принимает?
9. Как регулируется частота звука?

ПРИЛОЖЕНИЕ

Программное обеспечение одноплатной ЭВМ ARDUINO

Arduino программируется на специальном языке программирования – он основан на языке C и позволяет использовать любые его функции. Строго говоря, отдельного языка Arduino не существует, как и не существует компилятора Arduino – написанные программы преобразуются с минимальными изменениями в программу на языке C, и затем компилируются компилятором AVG-GCC. Так что фактически используется специализированный для микроконтроллеров вариант C. Разница заключается в том, что создана простая среда разработки и набор базовых библиотек, упрощающих доступ к находящейся в схеме периферии.

Структура языка. Базовая структура программы для Arduino довольно проста и состоит, по меньшей мере, из двух частей. В этих двух обязательных частях, или функциях, заключён выполняемый код.

```
void setup()
{
    statements;
}
void loop()
{
    statements;
}
```

В программе: `setup()` – это подготовка, `loop()` – выполнение. Обе функции требуются для работы программы.

Перед функцией `setup` –, в самом начале программы обычно идёт объявление всех переменных. `setup` – это первая функция, выполняемая программой, и выполняемая только один раз, поэтому она используется для установки режима работы портов (`pinMode()`) или инициализации последовательного соединения Следующая функция `loop` содержит код, который выполняется постоянно – читаются входы, переключаются выходы и т. д. Эта функция – ядро всех программ Arduino и выполняет основную работу.

Функция *setup()*. Функция `setup()` вызывается один раз, когда программа стартует, используется для установки режима выводов

или инициализации последовательного соединения. Она должна быть включена в программу, даже если в ней нет никакого содержания.

```
void setup()
{
    pinMode (pin, OUTPUT); //Устанавливает «pin» как выход
}
```

Функция *loop().* После вызова функции `setup()` – управление переходит к функции `loop()`, которая задает выполнение программы по командам.

```
void loop()
{
    digitalWrite(pin, HIGH); // включает «pin»
    delay(1000);             // пауза 1секунда
    digitalWrite(pin, LOW);  // выключает «pin»
    delay(1000);             // пауза 1 секунда
}
```

Функции. Функция – это блок кода, имеющего имя, которое указывает на исполняемый код, который выполняется при вызове функции. Функции `void setup()` и `void loop()` уже обсуждались, а другие встроенные функции будут рассмотрены позже.

Могут быть написаны различные пользовательские функции для выполнения повторяющихся задач и уменьшения беспорядка в программе. При создании функции первым делом указывается тип функции. Это тип значения, возвращаемого функцией, такой как «`int`» для целого (`integer`) типа функции. Если функция не возвращает значения, её тип должен быть `void`. За типом функции следует её имя, а в скобках параметры, передаваемые в функцию.

```
void setup()
{
    pinMode (pin, OUTPUT); //Устанавливает «pin» как выход
}
```

Следующая функция целого типа `delayVal()` используется для задания значения паузы в программе чтением значения с потенциометра. Вначале объявляется локальная переменная `v`, затем `v` устанавливается в значение потенциометра, определяемое числом между 0–1023, затем это значение делится на 4, чтобы результирую-

щее значение было между 0 и 255, а затем это значение возвращается в основную программу.

```
Int delayVal()
{
    int v;                //создаем временную переменную «v»
    v=analogRead(pot);    //считываем значение с потенциометра
    v/=4;                 //конвертируем 0-1023 в 0-255
    return v;             //возвращаем конечное значение
}
```

Фигурные скобки {}. Фигурные скобки (также упоминаются как просто скобки) определяют начало и конец блока функции или блока выражений, таких как функция `void loop()` или выражений (`statements`) типа `for` и `if`.

```
type function()
{
    statements;
}
```

За открывающейся фигурной скобкой { всегда должна следовать закрывающаяся фигурная скобка }. Несбалансированные скобки могут приводить к критическим, неясным ошибкам компиляции, вдобавок иногда и трудно выявляемым в больших программах.

Arduino-IDE включает возможность удобной проверки баланса фигурных скобок. Достаточно выделить скобку, или даже щёлкнуть по точке вставки сразу за скобкой, чтобы её пара была подсвечена.

Точка с запятой ;. Точка с запятой должна использоваться в конце выражения и разделять элементы программы. Также точка с запятой используется для разделения элементов цикла `for` (см. предыдущий пример).

При незавершенной строке без точки с запятой возникает ошибка компиляции. Текст ошибки может быть очевиден и указывать на пропущенную точку с запятой, но может быть и не таким очевидным.

Если появляется непонятная ошибка компилятора, первое, что следует проверить – не пропущена ли точка с запятой вблизи строки, где компилятор указал на ошибку.

Объявление переменных. Все переменные должны быть задекларированы до того, как они могут использоваться. Объявление переменной означает определение типа её значения: `int`, `long`, `float` и т. д., задание уникального имени переменной, и дополнительно ей

можно присвоить начальное значение. Всё это следует делать только один раз в программе, но значение может меняться в любое время при использовании арифметических или других разных операций.

Следующий пример показывает, что объявленная переменная `inputVariable` имеет тип `int`, и её начальное значение равно нулю. Это называется простым присваиванием:

```
int inputVariable = 0;
```

Переменная может быть объявлена в разных местах программы, и то, где это сделано, определяет, какие части программы могут использовать переменную

Границы переменных. Переменные могут быть объявлены в начале программы перед `void setup()`, локально внутри функций, и иногда в блоке выражений таком, как цикл `for`. То, где объявлена переменная, определяет её границы (область видимости), или возможность некоторых частей программы её использовать.

Глобальные переменные таковы, что их могут видеть и использовать любые функции и выражения программы. Такие переменные декларируются в начале программы перед функцией `setup()`.

Локальные переменные определяются внутри функций или таких частей, как цикл `for`. Они видимы и могут использоваться только внутри функции, в которой объявлены. Таким образом, могут существовать несколько переменных с одинаковыми именами в разных частях одной программы, которые содержат разные значения. Уверенность, что только одна функция имеет доступ к её переменной, упрощает программу и уменьшает потенциальную опасность возникновения ошибок.

Следующий пример показывает, как декларировать несколько разных типов переменных, и демонстрирует видимость каждой переменной:

```
int value;                // «value» видима для любой функции
void setup()
{
     // no setup needed
}
void loop()
{
    for (int i=0; i<20;) // «i» видима только внутри цикла for
    {
    i++;
    }
    float f;              // «f» видима только внутри loop
}
```

Byte. Байт хранит 8-битовое числовое значение без десятичной точки. Он имеет диапазон от 0 до 255.

```
byte someVariable = 180; //объявление «someVariable» как имеющий
                         //тип byte
```

Int. Целое – это первый тип данных для хранения чисел без десятичной точки, и хранит 16-битовое значение в диапазоне от 32 767 до -32 768.

```
int someVariable = 1500; //объявляет переменную «someVariable»
                         // как переменную целого типа
```

Целые переменные будут переполняться, если форсировать их переход через максимум или минимум при присваивании или сравнении. Например, если x = 32 767 и следующее выражение добавляет 1 к x (x = x +1 или x++), в этом случае x переполняется и будет равен -32 678.

Long. Тип данных увеличенного размера для больших целых, без десятичной точки, сохраняемый в 32-битовом значении с диапазоном от 2 147 383 647 до -2 147 383 648.

```
long someVariable = 1500; //декларирует «someVariable» типа long
```

Float. Тип данных для чисел с плавающей точкой или чисел, имеющих десятичную точку. Числа с плавающей точкой имеют большее разрешение, чем целые и сохраняются как 32-битовые значения в диапазоне от 3.4028235E+38 до -3.4028235E+38.

```
float someVariable = 3.14; //объявляет переменную «someVariable»
                           // как floating-point тип
```

Вычисления с плавающей точкой медленнее, чем вычисления целых при выполнении расчётов, так что, без нужды, их следует избегать.

Массивы. Массив – это набор значений, к которым есть доступ через значение индекса.

```
int myArray[] = {value0, value1, value2,…}
```

Любое значение в массиве может быть вызвано через вызов имени массива и индекса значения. Индексы в массиве начинаются с нуля с первым значением, имеющим индекс 0. Массив нуждается в объявлении, а дополнительно может заполняться значениями до то-

го, как будет использоваться. Схожим образом можно объявлять массив, указав его тип и размер, а позже присваивать значения по позиции индекса:

```
int myArray[5];      // объявляет массив целых длиной в 6 позиций
myArray[3] = 10;     // присваивает по 4-у индексу значение 10
```

Чтобы извлечь значение из массива, присвоим переменной значение по индексу массива:

```
X = myArray[3];      // x теперь равно 10
```

Массивы часто используются в цикле `for`, где увеличивающийся счётчик применяется для индексации позиции каждого значения. Следующий пример использует массив для мерцания светодиода. Используемый цикл `for` со счётчиком, начинающимся с 0, записывает значение из позиции с индексом 0 массива `flicker[]`, в данном случае 180, на PWM-вывод (широтно-импульсная модуляция) 10; затем пауза в 200 мс, а затем переход к следующей позиции индекса.

```
int ledPin = 10;                //LED на выводе 10
byte flicker [] = {180, 30, 255, 200, 10, 90, 150, 60};
                                //массив из 8 разных значений
void setup ()
{
    pinMode (ledPin, OUTPUT); //задаем OUTPUT вывод
}
void loop()
{
    for (int i=0; i<7; i++)           //цикл равен числу
    {                                 //значений в массиве
    analogWrite (ledPin, flicker[i]); //пишем значение по индексу
    delay (200);                      //пауза 200 мс
    }
}
```

Арифметика. Арифметические операции включают сложение, вычитание, умножение и деление. Они возвращают сумму, разность, произведение или частное (соответственно) двух операндов.

```
y = y + 3;
x = x - 7;
I = j * 6;
r = r / 5;
```

Операция управляется используемым типом данных операндов, так что, например, 9/4 даёт 2 вместо 2.25, поскольку 9 и 4 имеют тип `int` и не могут использовать десятичную точку. Это также означает, что операция может вызвать переполнение, если результат больше, чем может храниться в данном типе. Если используются операнды разного типа, то для расчётов используется больший тип. Например, если одно из чисел (операндов) типа `float`, а второе целое, то для вычислений используется тип с плавающей точкой.

Следует выбирать типы переменных, достаточные для хранения результатов ваших вычислений. Прикиньте, в какой точке ваша переменная переполнится, а также, – что случится в другом направлении, то есть, (0-1) или (0- -32768). Для вычислений, требующих дробей, используйте переменные типа `float`, но остерегайтесь их недостатков: большой размер и маленькая скорость вычислений.

Используйте оператор приведения типа (название типа) для округления, то есть, `(int)myFloat` – для преобразования переменной одного типа в другой «на лету». Например, `i = (int)` 3.6 – поместит в `i` значение 3.

Смешанное присваивание. Смешанное присваивание сочетает арифметические операции с операциями присваивания. Чаще всего встречается в цикле `for`, который описан ниже. Наиболее общее смешанное присваивание включает:

```
x ++      // то же, что x = x + 1, или увеличение x на +1
x --      // то же, что x = x - 1, или уменьшение x на -1
x += y    // то же, что x = x + y, или увеличение x на +y
x -= y    // то же, что x = x - y, или уменьшение x на -y
x *= y    // то же, что x = x * y, или умножение x на y
x /= y    // то же, что x = x / y, или деление x на y
```

Например, `x *= 3` утроит старое значение `x` и присвоит полученный результат x.

Операторы сравнения. Сравнения одной переменной или константы с другой используются в выражении для `if`, чтобы проверить истинность заданного условия. В примерах на следующих страницах указанные знаки сравнения используются для обозначения любого из следующих условий:

```
x == y    // x равно y
x != y    // x не равно y
x < y     // x меньше, чем y
```

```
x > y     // x больше, чем y
x <= y    // x меньше, чем или равно y
x >= y    // x больше, чем или равно y
```

Логические операторы. Логические операторы, чаще всего, это способ сравнить два выражения и вернуть «ИСТИНА» или «ЛОЖЬ», в зависимости от оператора. Есть три логических оператора: `AND`, `OR` и `NOT`, часто используемые в конструкциях `if`:

Logical AND:
```
if (x > 0 && x < 5) //true, только если оба выражения true
```

Logical OR:
```
if (x > 0 || y > 0) //true, только если любое из выражений true
```

Logical NOT:
```
if (!x > 0)         //true, только если выражение false
```

Константы. Язык Arduino имеет несколько предопределённых величин, называемых константами. Они используются, чтобы сделать программу удобной для чтения. Константы собраны в группы.

True/false – это булевы константы, определяющие логические уровни. FALSE легко определяется как 0 (ноль), а TRUE, как 1, но может быть и чем-то другим, отличным от нуля. Так что в булевом смысле -1, 2 и 200 – это всё определяется как TRUE.

```
if (b == True);
{
    doSomething;
}
```

High/low. Эти константы определяют уровень выводов как HIGH или LOW и используются при чтении или записи на логические выводы. HIGH определяется как логический уровень 1, ON или 5 вольт(3-5), тогда как LOW - 0, OFF или 0 вольт (0-2В).

```
digitalWrite (13, HIGT);
```

Input/output. Константы используются с функцией `pinMode()` для задания режима работы цифровых выводов: либо как `INPUT` (вход), либо как `OUTPUT` (выход).

```
pinMode (13, OUTPUT);
```

Управление программой

If. Конструкция `if` проверяет, будет ли выполнено некое условие, такое, как например: будет ли аналоговое значение больше заданного числа, – и выполняет какое-то выражение в скобках, если это условие `true` (истинно). Если нет, то выражение в скобках будет пропущено. Формат для `if` следующий:

```
if (someVariable ?? value)
{
    doSomething;
}
```

В примере сравнивается `someVariable` со значением (`value`), которое может быть и переменной, и константой. Если выражение или условие в скобках истинно, выполняется выражение в фигурных скобках. Если нет, выражение в фигурных скобках пропускается и программа выполняется с оператора, следующего за скобками.

Следует избегать случайного использования «=», как в `if (x = 10)`, что технически правильно, определяя x равным 10, но результат этого всегда `true`. Вместо этого используйте «==», как в `if (x == 10)`, что осуществляет проверку значения `x` – равно ли оно 10 или нет. Запомните «=» – равно, а «==» – равно ли?

If...else. Конструкция `if...else` позволяет сделать выбор «либо, либо». Например, если вы хотите проверить цифровой вход и выполнить что-то, если он HIGH, или выполнить что-то другое, если он был LOW, вы должны записать следующее:

```
if (inputPin == HIGH)
    {
    doThingA;
    }
    else
    {
    doThingB;
    }
```

`Else` может также предшествовать другой проверке `if` так, что эти множественные, взаимоисключающие проверки могут запускаться одновременно. И возможно даже неограниченное количество подоб-

ных `else` переходов. Хотя следует помнить, что только один набор выражений будет выполнен в зависимости от результата проверки:

```
if (inputPin < 500)
    {
    doThingA;
    }
    Else if (input >= 1000)
    {
        doThingB;
    }
    else
    {
        doThingC;
    }
```

Конструкция `if` просто проверяет, будет ли выражение в круглых скобках истинно или ложно. Это выражение может быть любым правильным, относительно языка Си, выражением, как в первом примере `if (inputPin == HIGH)`. В этом примере `if` проверяет только то, что означенный вход в состоянии высокого логического уровня или действительно напряжение на нём 5 вольт?

For. Конструкция `for` используется для повторения блока выражений, заключённых в фигурные скобки заданное число раз. Наращиваемый счётчик часто используется для увеличения и прекращения цикла. Есть три части, разделённые точкой с запятой, в заголовке цикла `for:`

```
for (инициализация; условие; выражение)
{
    doSomething;
}
```

Инициализация локальной переменной, или счётчика, имеет место в самом начале и происходит только один раз. При каждом проходе цикла проверяется условие. Если условие остаётся истинным, то следующее выражение и блок выполняются, а условие проверяется вновь. Когда условие становится ложным, цикл завершается.

Следующий пример начинается с целого i равного 0, проверяет, остаётся ли i ещё меньше 20 и, если так, увеличивает i на 1 и выполняет блок в фигурных скобках:

```
for (int i=0; i<20; i++)      // декларирует i, проверяет меньше
                              // ли чем 20, увеличивает i на 1
{
    digitalWrite(13, HIGH);   // устанавливает вывод 13 в ON
    delay(250);               // пауза 1/4 секунды
    digitalWrite(13, LOW);    // сбрасывает вывод 13 в OFF
    delay(250);               // пауза 1/4 секунды
}
```

While. Цикл `while` продолжается, и может продолжаться бесконечно, пока выражение в скобках не станет `false` (ложно). Что-то должно менять проверяемую переменную, иначе из цикла никогда не выйти. И это должно быть в вашем коде, как скажем, увеличение переменной, или внешнее условие, как например, проверяемый сенсор.

```
while (someVariable ?? value)
{
    doSomething;
}
```

Следующий пример проверяет, будет ли someVariable меньше 200, и если да, то выполняются выражения в фигурных скобках, и цикл продолжается, пока someVariable остаётся меньше 200.

```
while (someVariable < 200)    //проверяет, меньше ли 200
{
    doSomething;              //выполняет выражение в скобках
    someVariable++;           //увеличивает переменную на 1
}
```

Do...while. Цикл `do` – управляемый «снизу» цикл, работающий на манер цикла `while`, с тем отличием, что условие проверки расположено в конце цикла; таким образом, цикл выполнится хотя бы один раз.

```
do
{
    doSomething;
}   while (someVariable ?? value);
```

Serial.println (data). Передаёт данные в последовательный порт, сопровождая автоматической командой возврат каретки и переходом на новую строку. Команда такая же, что и `Serial.print()`, но легче для последующего чтения на данных в терминале.

```
Serial.println (analogValue); //отправляет значение «analoValue»
```

Следующий пример читает аналоговый вывод 0 и отсылает эти данные на компьютер каждую секунду.

```
void setup ()
{
    Serial.begin (9600);  // задаем скорость 9600 бод
}
void loop ()
{
    Serial.println (analogRead (0)); //
    delay (1000);
}
```

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Блум Дж. Изучаем Arduino: инструметы и методы технического волшебства: пер. с англ. – СПб.: БХВ-Петербург, 2015. – 336 с.
2. Соммер У. Программирование микроконтроллерных плат Arduino/Freeduino. – СПб.: БХВ-Петербург, 2012. – 256 с.
3. Brian W. Evans. Блокнот программиста. Creative Commons, 2007. – 40 с.
4. Евстифеев А.В. Микроконтроллеры AVR семейств Tiny и Mega фирмы «Atmel» / А.В. Евстифеев. – М.: Издательский дом «Додэка-XXI», 2004. – 560 с.

97821000201 1

Сергей Николаевич Чижма

ИЗУЧЕНИЕ ОДНОПЛАТНОЙ ЭВМ ARDUINO

Методические указания
по выполнению лабораторных работ
для курсантов специальности 25.05.03
«Техническая эксплуатация
транспортного радиооборудования»
всех форм обучения

Ведущий редактор О.В. Напалкова
Младший редактор Г.В. Деркач

Компьютерное редактирование
И.В. Леонова

Подписано в печать 18.06.2019 г.
Усл. печ. л. 3,5. Уч.-изд. л. 3,5.

Лицензия № 021350 от 28.06.99.

Печать офсетная.

Формат 60 x 90 1/16.

Заказ № 1410. Тираж 40 экз.

Доступ к архиву публикации и условия доступа к нему:
http://bgarf.ru/academy/biblioteka/elektronnyj-katalog/

БГАРФ ФГБОУ ВО «КГТУ»

Издательство БГАРФ,
член Издательско-полиграфической ассоциации высших учебных заведений
236029, Калининград, ул. Молодежная, 6.