издательство

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

УЧЕБНОЕ ПОСОБИЕ

В.А. МОЖАЕВ
В.И. БУСУРИН
Л.А. ШЛЕЕНКИН

ПРОГРАММНО-АППАРАТНЫЕ СРЕДСТВА АВИАЦИОННОЙ АВТОМАТИКИ НА ОСНОВЕ МИКРОКОНТРОЛЛЕРОВ

Москва • 2019

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)

В.А. МОЖАЕВ, В.И. БУСУРИН, Л.А. ШЛЕЁНКИН

ПРОГРАММНО-АППАРАТНЫЕ СРЕДСТВА АВИАЦИОННОЙ АВТОМАТИКИ НА ОСНОВЕ МИКРОКОНТРОЛЛЕРОВ

Учебное пособие

Утверждено
на заседании редсовета
14 ноября 2018 г.

Москва
Издательство МАИ
2019

Можаев В.А., Бусурин В.И., Шлеёнкин Л.А.

Программно-аппаратные средства авиационной автоматики на основе микроконтроллеров: Учебное пособие. — М.: Изд-во МАИ, 2019. — 76 с.: ил.

Пособие предназначено для изучения основ разработки микроконтроллерных устройств, приобретения навыков построения и отладки реальных функциональных блоков в системе Arduino. Плата построена на базе микросхемы Atmel ATmega328P. Она поддерживается большим разнообразием модулей расширения, как со стороны датчиков, так и со стороны актюаторов. Программы составляются на простом языке, основанном на C++. Интегрированная среда программирования Arduino IDE, устанавливается на компьютере с операционной системой Windows.

Для студентов, обучающихся по специальности 24.05.06 “Системы управления летательными аппаратами” и изучающих дисциплину “Микропроцессорная техника в приборах, системах и комплексах”.

Рецензенты:

кафедра “Системы управления ДС” ВА РВСН им. Петра Великого;

докт. техн. наук, профессор *В.П. Харьков*

ISBN 978-5-4316-0652-6

ВВЕДЕНИЕ

Макетные (отладочные) платы микроконтроллеров содержат микросхемы микроконтроллеров, стабилизаторы напряжения питания, набор вспомогательных схем для связи с компьютером, индикаторные светодиоды и др.

Программы для микроконтроллеров создаются на персональном компьютере в интегрированной среде разработки (IDE). Загрузка отлаженных программ в память микроконтроллеров выполняется из среды разработки через последовательный интерфейс.

Макетная плата является полноценной работоспособной микроконтроллерной системой. Все информационные и управляющие сигналы микроконтроллера выводятся на штырьковые разъёмы платы. Через эти разъёмы к микроконтроллеру можно подключать разнообразные сенсорные и актюаторные устройства.

В области макетирования несложных микроконтроллерных устройств, популярностью пользуется система Arduino. Она разрабатывалась как образовательный проект и предназначалась для начального знакомства с микропроцессорной техникой и основами программирования микроконтроллеров. Система включает плату с работоспособной микроконтроллерной системой и интегрированную среду разработки программ “Arduino IDE”. В дополнение к плате выпускается большое разнообразие внешних периферийных модулей (шилдов) с необходимой программной поддержкой в виде библиотек функций.

1. ПЛАТА Arduino

Макетная плата Arduino Uno (рис. 1.1) построена на базе микроконтроллера ATmega328 фирмы Atmel. Среда Arduino IDE устанавливается на компьютер с операционной системой Windows и используется для разработки и загрузки рабочих программ в микроконтроллер. Плата может подключаться к компьютеру для про-

граммирования и обмена данными через порт USB. При этом возможен дуплексный обмен данными между компьютером и платой. Программы в системе Arduino принято называть скетчами. После загрузки скетча плата может работать автономно.

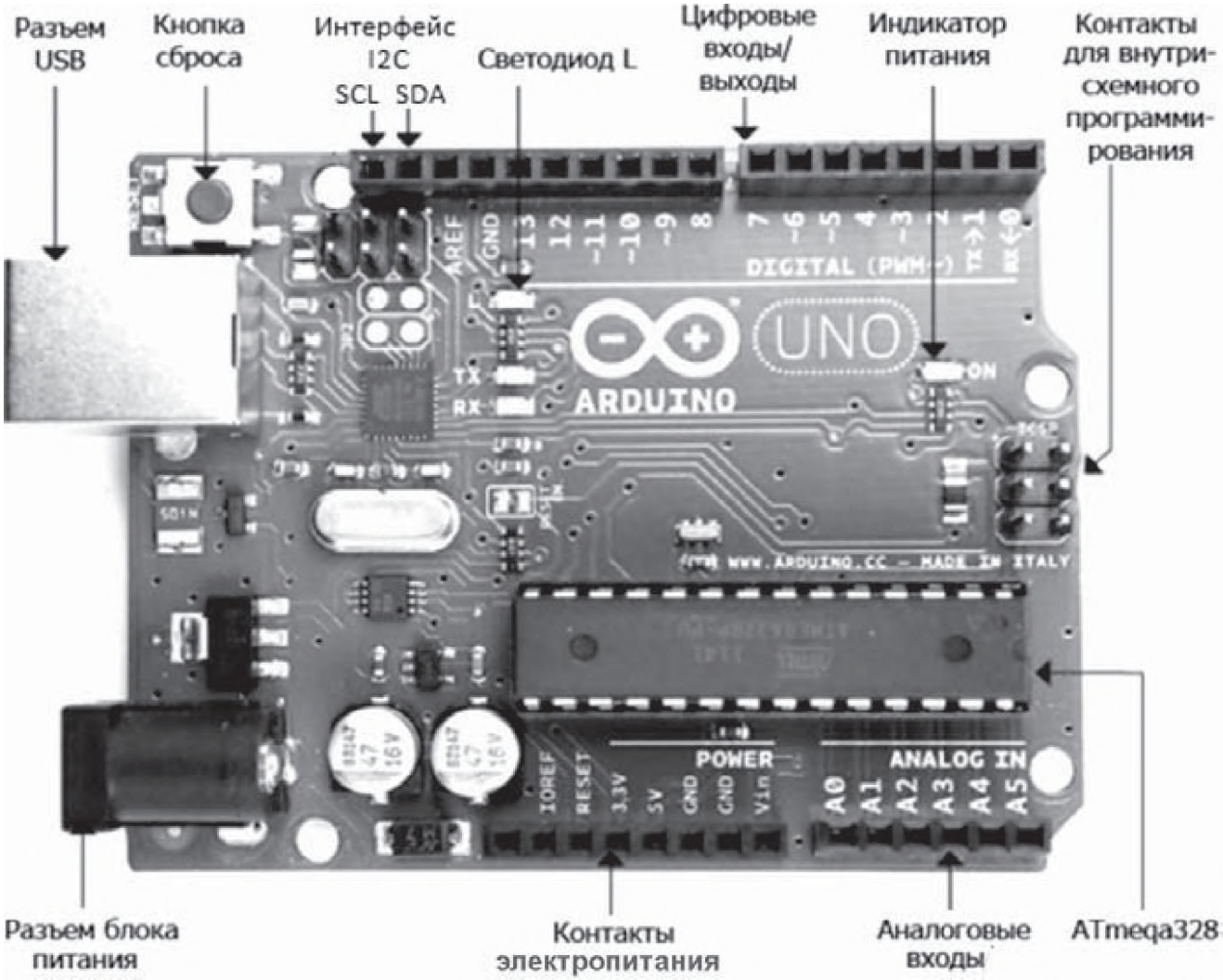

Рис. 1.1

По верхнему и нижнему краю платы (см. рис. 1.1) размещены разъёмы, на которые выведены основные контакты микроконтроллера. Через разъёмы к контроллеру подключаются внешние датчики (сенсоры) и исполнительные устройства (актюаторы).

В левом верхнем углу находится кнопка сброса платы в начальное состояние. Если в ПЗУ микроконтроллера записана программа, то после сброса она запускается на исполнение.

Питание платы Arduino возможно от компьютера, через порт USB, или от внешнего блока питания с напряжением Vin=7,5...12 В, через разъем блока питания. Внешнее напряжение можно контро-

лировать на контакте Vin. При подаче питания на плате загорается индикатор ON.

Контакты нижнего разъёма электропитания предназначены для внешних устройств. Первый контакт, без метки, зарезервирован для использования в будущем. Следующий контакт, IOREF, позволяет измерить напряжение, на котором работает плата. Плата Uno использует напряжение 5 В.

Следующий контакт RESET позволяет сбросить микроконтроллер в исходное состояние кратковременной подачей на него напряжения 0 В.

Остальные контакты в этой группе служат для вывода электропитания с разными уровнями напряжения: 3.3V, 5V, GND. GND обозначает напряжение 0 В. В верхнем разъёме на плате есть еще один контакт GND.

Контакты в следующей группе подписаны ANALOG IN (аналоговые входы) с номерами от A0 до A5. Эти шесть контактов можно подключать к входу АЦП микроконтроллера.

В верхнем разъёме находятся цифровые контакты DIGITAL 0...13. В режиме вывода цифровые контакты могут выдавать ток до 40 мА с напряжением 5 В.

Дополнительно на плате размещаются: стабилизаторы напряжения на 5 и 3,3 В; кварцевый резонатор на 16 МГц для стабилизации тактовых импульсов; преобразователь интерфейсов USB-UART для связи микроконтроллера с компьютером. В ATmega328 нет модуля USB и связь реализуется через USB-UART преобразователь.

Для контроля над состоянием платы на ней размещены четыре светодиода: ON — “питание включено”; RX и TX — “идёт обмен с компьютером”; L — индикаторный диод, подключенный к цифровому контакту 13.

Через контакты разъёмов к плате может присоединяться “плата расширения”, на которой все контакты платы Arduino распределены по функционально обособленным группам, для подключения часто используемых периферийных модулей (рис. 1.2). В средней части платы расширения дополнительно, в ряд, располагаются четыре группы контактов. Верхние и средние контакты всех групп подключены к линиям питания — GND и Vcc соответственно. Нижние ряды подключены к пронумерованным контактам ввода/вывода микроконтроллера.

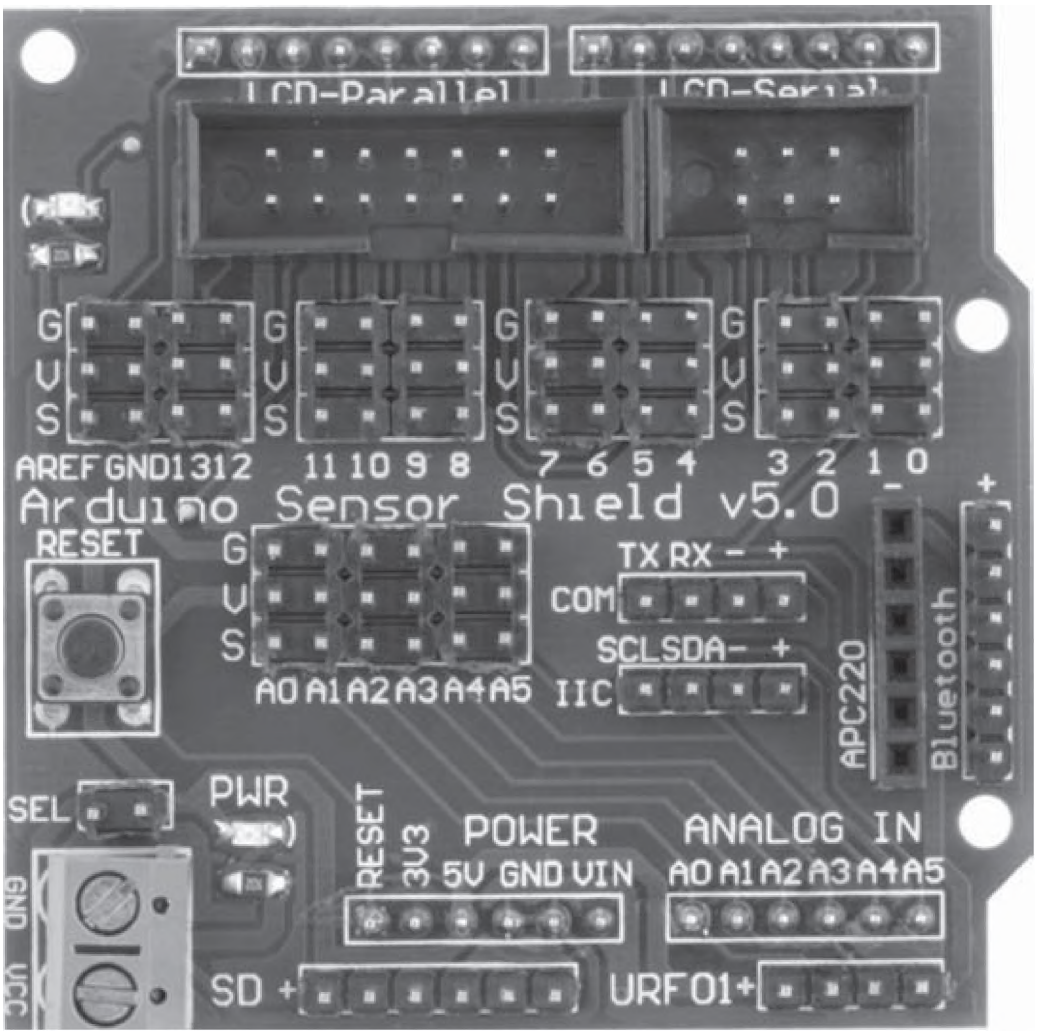

Рис. 1.2

Контроллер ATmega328

Восьмиразрядный микроконтроллер ATmega328 предназначен для встраиваемых приложений. Он имеет RISC архитектуру и производится по КМОП-технологии.

Структура микроконтроллера (рис. 1.3) объединяет центральный процессор, модули памяти (FLASH память программ — 32 Кбайта; статическое ОЗУ — 2 Кбайта; 1 Кбайт энергонезависимой памяти данных EEPROM). Для связи с внешней средой контроллер включает периферийные модули:

- 8-разрядные таймеры/счетчики (T0, T2) и 16-разрядный таймер/счётчик T1. Таймеры имеют режимы ШИМ-генераторов;
- сторожевой таймер WDT;
- аналоговый компаратор;

– многоканальный 10-разрядный АЦП;
– последовательный универсальный приемопередатчик USART;
– последовательный синхронный интерфейс SPI;
– последовательный двухпроводный интерфейс TWI (аналог интерфейса I^2C).

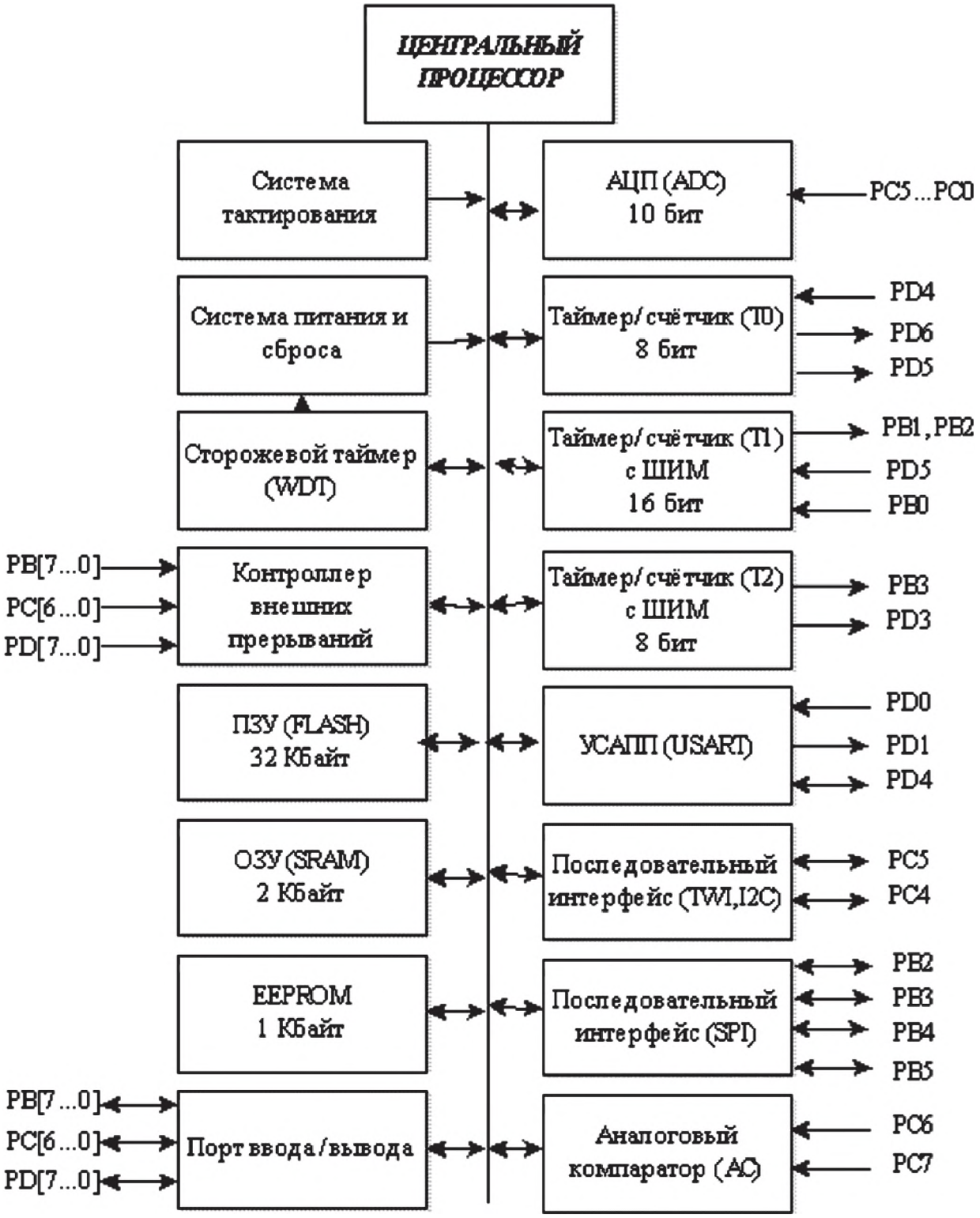

Рис. 1.3

Мультиплексирование внешних контактов микроконтроллера исключает одновременное их использование периферийными устройствами и портами ввода-вывода общего назначения. На плате Arduino контакты микроконтроллера обозначены как цифровые и аналоговые (рис. 1.4). При желании аналоговые контакты могут использоваться как цифровые, а некоторые цифровые — для вывода ШИМ (PWM) сигналов.

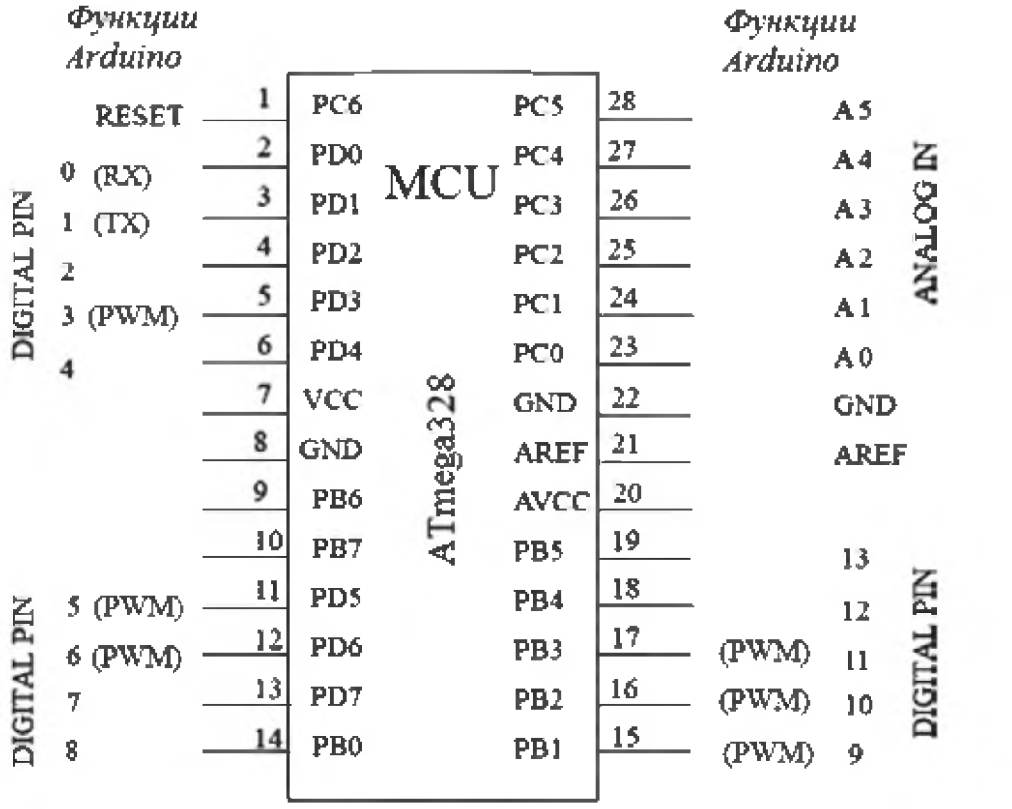

Рис. 1.4

Информационные контакты группируются в порты ввода-вывода общего назначения — два восьмиразрядных (PB[7..0], PD[7..0]) и один семиразрядный (PC[6..0]). Разряды портов независимо могут быть настроены на разные режимы работы. Каждый порт управляется тремя *регистрами*.

Регистр DDR*x* задаёт направление передачи данных через контакты порта ($x = \{B, C, D\}$). Если разряд регистра установлен в «1», соответствующий контакт работает как выход, а если сброшен в «0» — как вход.

Регистр PORT*x* используется для передачи сигнала на выход. Если в бит регистра записывается «1» или «0», то на выходном контакте будет высокий (HIGH) или низкий (LOW) уровень сигнала.

Значения битов третьего регистра PINx соответствуют уровням входного сигнала на контактах порта («1» — HIGH, «0» — LOW).

Все контакты портов (рис. 1.5) снабжены защитными диодами, предотвращающими появление на них как отрицательного напряжения, так и напряжения, превышающего напряжение питания.

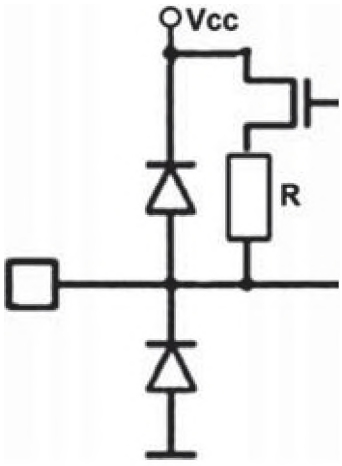

Рис. 1.5

При замыкании ключа на полевом транзисторе в режиме ввода напряжение питания подаётся на контакт через “подтягивающий резистор” R.

В процессе выполнения скетча процессор читает команды из флеш-памяти и выполняет их. Переменные, используемые в программе, хранятся в статическом энергозависимом ОЗУ. Данные, полученные во время работы скетча и которые нужно сохранить после выключения питания, можно разместить в модуле ЭСППЗУ (EEPROM).

В контроллере Arduino предусмотрена небольшая резидентная программа-загрузчик (bootloader), которая выполняется каждый раз при включении питания. Если на плату через USB передаётся скетч, загрузчик сохраняет его во флеш-памяти программ. После сброса выполняется скетч из флеш-памяти.

2. СРЕДА РАЗРАБОТКИ Arduino IDE

Для программирования платы Arduino используется интегрированная среда разработки Arduino IDE. Для работы требуется плата Arduino, компьютер и кабель USB для их соединения между собой.

Arduino IDE, работающая на копьютере, позволяет писать программы (скетчи) для платы Arduino на простом языке, основанном на C++. Оттранслированная программа записывается во Flash-память микроконтроллера.

Для программирования Arduino нужно:

- подключить плату через USB-порт к компьютеру с Arduino IDE;
- написать нужный скетч в среде Arduino IDE;
- выгрузить этот скетч на плату через USB.

Через несколько секунд плата перезапускается и начинается выполнение написанного скетча.

2.1. Запуск и настройка системы Arduino IDE

Для знакомства с системой используем готовый скетч Blink, который будет включать и выключать светодиод, отмеченный на плате меткой L. После вызова в системе Windows программы Arduino.exe открывается окно среды разработки Arduino IDE. В этом окне размещается заготовка нового скетча (рис. 2.1). Заготовка содержит объявление двух обязательных функций setup и loop. Функция setup выполняется первой и только один раз. В ней выполняются начальные установки и объявления. Далее вызывается функция loop. Она содержит основную программу, которая выполняется в бесконечном цикле.

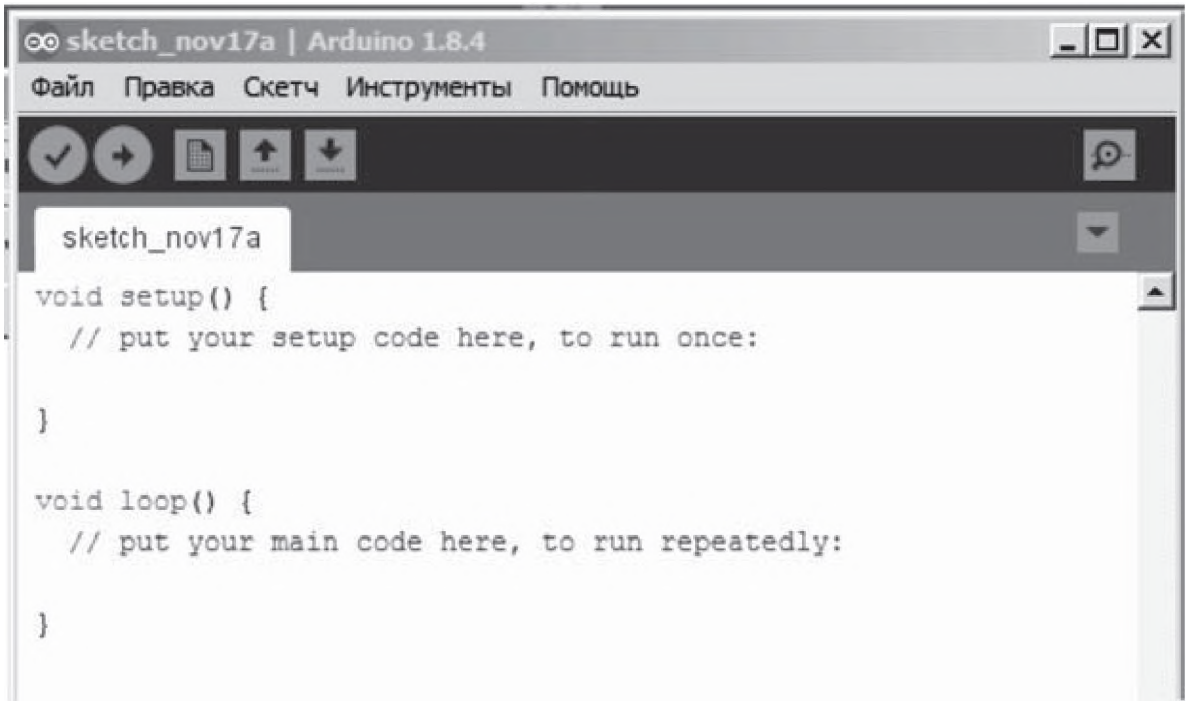

Рис. 2.1

Прежде чем выгрузить скетч Blink на плату Arduino, нужно указать тип платы. Для этого последовательно выбираются пункты меню:

Инструменты → Плата → Arduino/Genuino Uno (рис. 2.2).

Далее выбирается порт, к которому подключена плата (рис. 2.3).

Теперь можно открыть нужный скетч (рис. 2.4):

Файл → Примеры → 01. Basics → Blink.

На рабочее поле Arduino IDE выводится текст скетча Blink (рис. 2.5).

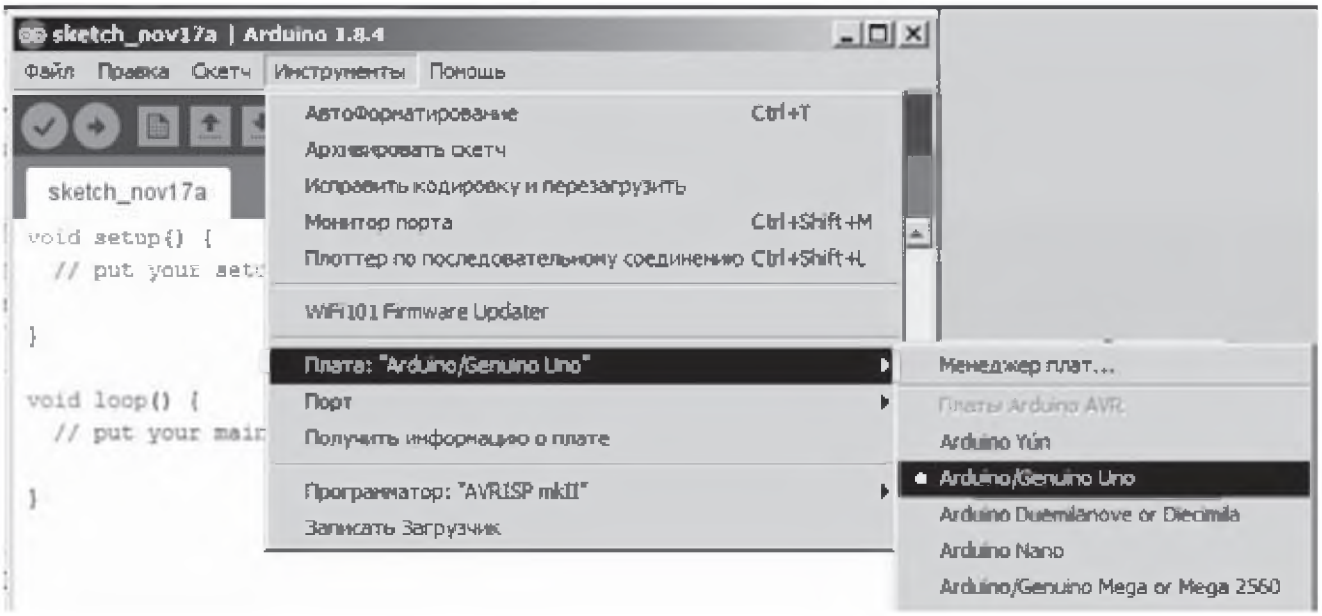

Рис. 2.2

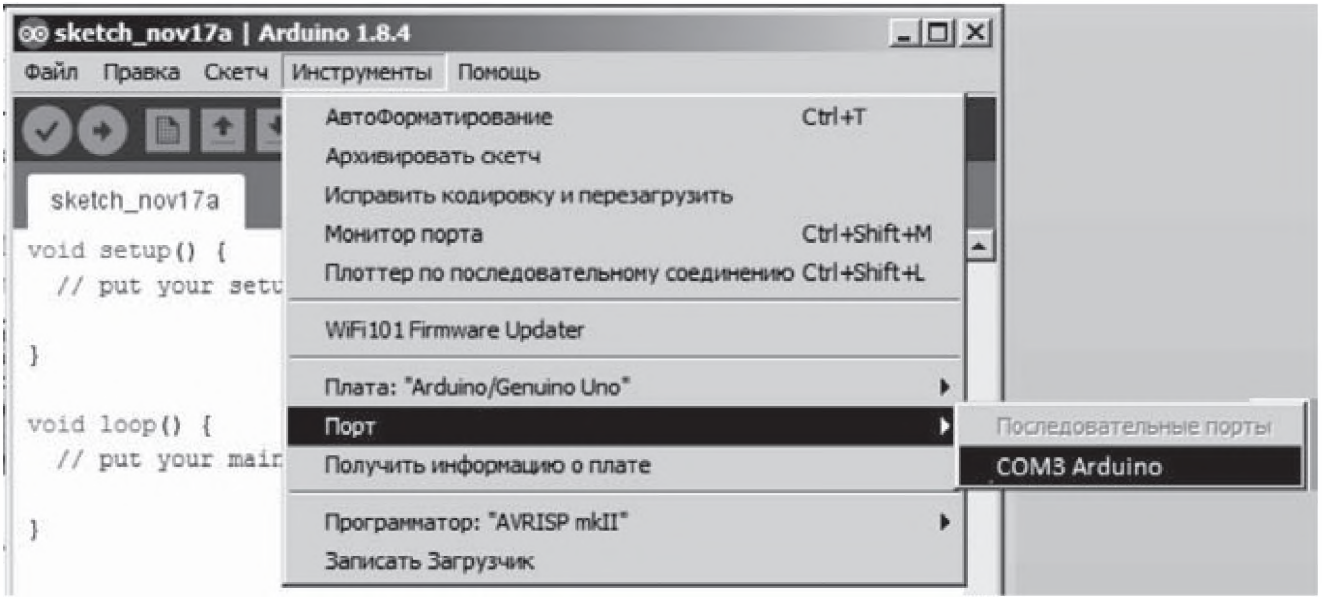

Рис. 2.3

Парные символы /* и */ ограничивают многострочный комментарий — пояснительный текст к скетчу: имя и основное назначение. С двойного слеша // начинается однострочный комментарий.

Далее идёт объявление функции setup:

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

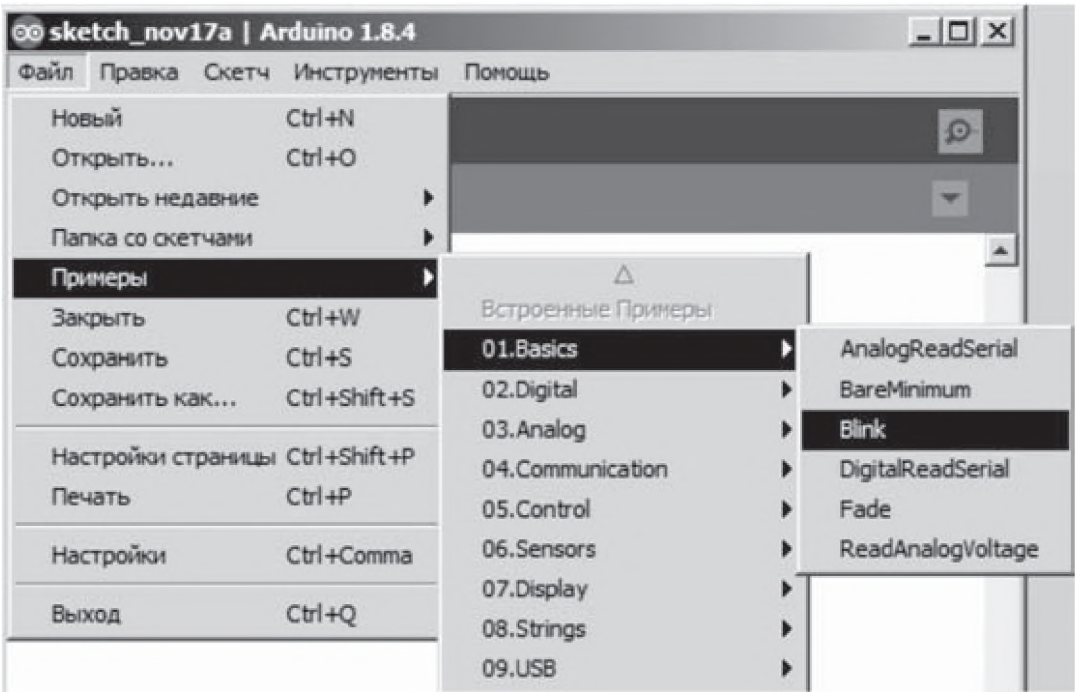
sketch_nov17a | Arduino 1.8.4
Файл Правка Скетч Инструменты Помощь
Новый Ctrl+N
Открыть... Ctrl+O
Открыть недавние
Папка со скетчами
Примеры
Закрыть Ctrl+W
Сохранить Ctrl+S
Сохранить как... Ctrl+Shift+S
Настройки страницы Ctrl+Shift+P
Печать Ctrl+P
Настройки Ctrl+Comma
Выход Ctrl+Q
Встроенные Примеры
01.Basics
02.Digital
03.Analog
04.Communication
05.Control
06.Sensors
07.Display
08.Strings
09.USB
AnalogReadSerial
BareMinimum
Blink
DigitalReadSerial
Fade
ReadAnalogVoltage

Рис. 2.4

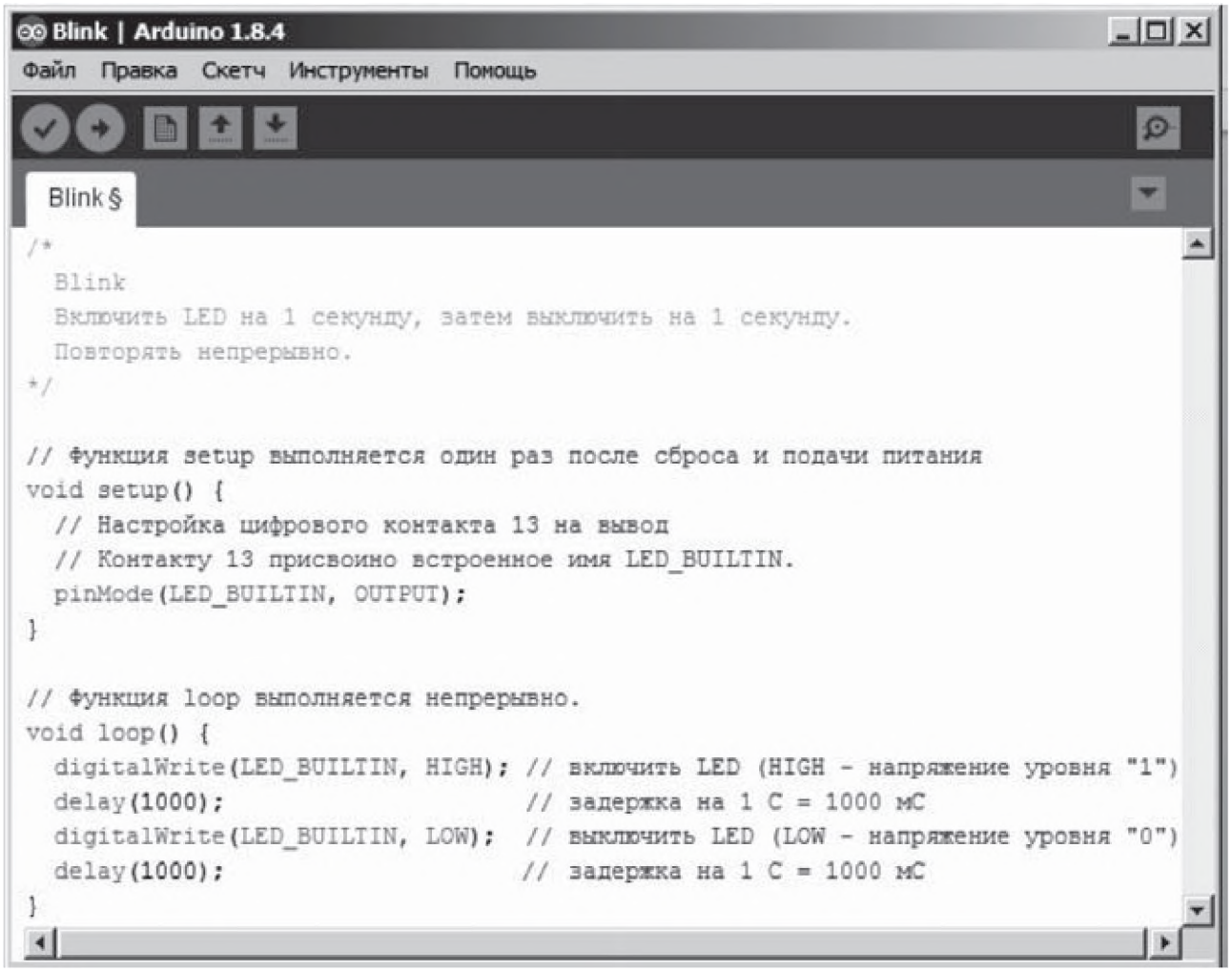
Blink | Arduino 1.8.4
Файл Правка Скетч Инструменты Помощь
Blink §
/*
Blink
Включить LED на 1 секунду, затем выключить на 1 секунду.
Повторять непрерывно.
*/
// Функция setup выполняется один раз после сброса и подачи питания
void setup() {
// Настройка цифрового контакта 13 на вывод
// Контакту 13 присвоино встроенное имя LED_BUILTIN.
pinMode(LED_BUILTIN, OUTPUT);
}
// Функция loop выполняется непрерывно.
void loop() {
digitalWrite(LED_BUILTIN, HIGH); // включить LED (HIGH - напряжение уровня "1")
delay(1000); // задержка на 1 С = 1000 мС
digitalWrite(LED_BUILTIN, LOW); // выключить LED (LOW - напряжение уровня "0")
delay(1000); // задержка на 1 С = 1000 мС
}

Рис. 2.5

Ключевое слово void указывает, что функция не возвращает значения, характеризующего результат её работы. Отсутствие формальных параметров в круглых скобках означает, что все необходимые данные для работы функции содержатся в её теле. Тело функции (блок) ограничивается фигурными скобками. Операции и объявления данных завершаются символом “точка с запятой”.

Операция pinMode настраивает контакт 13 платы на вывод — OUTPUT. Контакт 13 имеет встроенное имя LED_BUILTIN. К этому контакту на плате подключен светодиод L, который светится при выводе сигнала HIGH. Уровень напряжения HIGH близок к 5 В и соответствует «1» — логической единице (TRUE).

Первая операция в функции loop «digitalWrite(LED_BUILTIN, HIGH);» выводит на контакт 13 сигнал HIGH, включающий светодиод.

Вторая операция функции loop «delay(1000);» приостанавливает работу скетча на 1000 мс, и свечение светодиода продолжается в течение 1 с.

Аналогично работают третья и четвёртая операции. При выключении светодиода уровень сигнала LOW близок к 0 В и соответствует логическому значению «0» — FALSE. После выполнения последней операции функции loop, происходит переход к первой операции этой функции и т.д.

Чтобы загрузить скетч в микроконтроллер на плате Arduino, нужно щелкнуть на кнопке Загрузка (стрелка вправо). На панели инструментов — вторая слева (см. рис. 2.5).

После щелчка, справа внизу в окне Arduino IDE появится индикатор выполнения, отражающий ход компиляции скетча. Загрузка программы сопровождается миганием светодиодов Rx и Tx на плате Arduino. После загрузки, программа запускается и начинает мигать светодиод L.

2.2. Управление светодиодом L

Изменим скетч Blink, чтобы заставить светодиод мигать с частотой 5 Гц (задержка 200 мс). Попутно изменим имя контакта 13 на led. Имя контакта должно быть глобальным и доступным из любой части скетча. Объявление глобального имени выполняется вне фукций setup и loop:

```
int led = 13;      // номер контакта (13) хранится в памяти
                   // как двухбайтное число (int) и ему присвоено имя led
```

```
void setup() {
  pinMode(led, OUTPUT); // контакт led — цифровой выход
}
void loop() {
  digitalWrite(led, HIGH);
  delay(200);
  digitalWrite(led, LOW);
  delay(200);
}
```

После загрузки изменённого скетча светодиод будет мигать чаще.

Монитор последовательного порта

Плата Arduino подключается к компьютеру по интерфейсу USB, в котором предусмотрен двухсторонний обмен данными. В процессе выполнения скетча для обмена данными используется компонент Arduino IDE *монитор последовательного порта (последовательный монитор,* Serial Monitor).

Запишем скетч, который передаёт на компьютер сообщение о состоянии светодиода:

```
int led = 13;
void setup() {
pinMode(led, OUTPUT);
Serial.begin(9600);   //функция активирования обмена по каналу
}                     // USART-USB со скоростью 9600 бод
void loop() {
digitalWrite(led, HIGH);
Serial.println("Led ON");        //вывести в монитор строку символов
delay(1000);
digitalWrite(led, LOW);
Serial.println("Led OFF");    //вывести в монитор строку символов
delay(1000);
}
```

После выгрузки скетча на плату можно открыть монитор последовательного порта, щелкнув на кнопке с изображением, напоминающим лупу. На экране должны появляться сообщения о текущем состоянии светодиода (рис. 2.6).

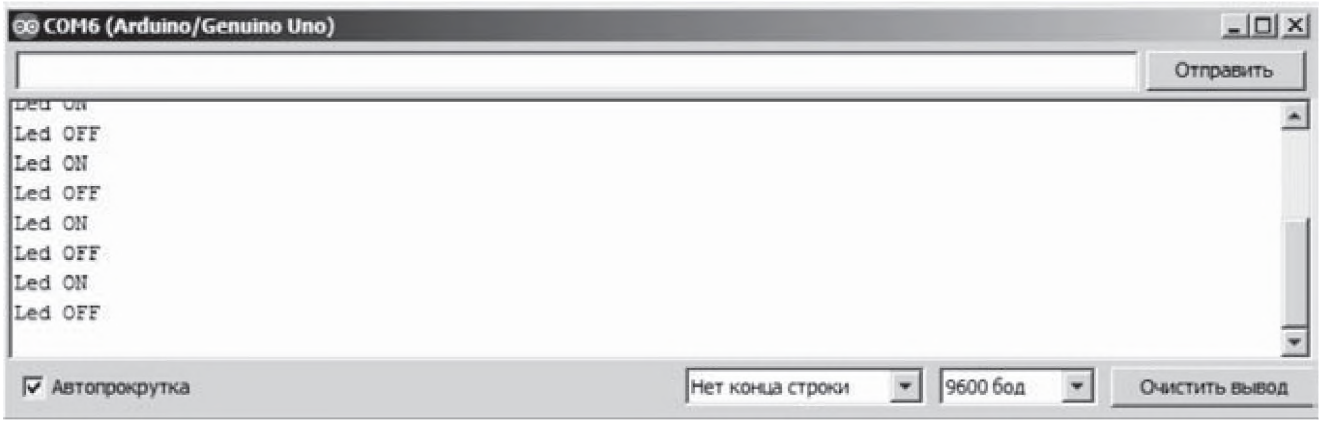

Рис. 2.6

Через монитор последовательного порта можно передать в работающую программу нужную информацию. Следующий скетч получает через монитор величину паузы dt в операторе delay(dt), задающем частоту мигания светодиода.

```
int led = 13;
int dt=1000;
void setup() {
pinMode(led, OUTPUT);
Serial.begin(9600); // активирование канала USART-USB}
void loop() {
if (Serial.available()) dt=Serial.parseInt(); //приём данных из монитора
digitalWrite(led, HIGH);
delay(dt);
digitalWrite(led, LOW);
delay(dt);
}
```

Величина задержки dt, в виде целого числа из диапазона 100...1000, вводится в текстовое поле в верхней части окна последовательного монитора, слева от кнопки "Отправить" (рис. 2.7). Передача происходит после щелчка на этой кнопке.

Рис. 2.7

В функции loop() скетча проверяется состояние приёмника. Пояснения к функциям библиотеки Serial приведены в разделе 4. Если из монитора получены данные (функция Serial.available() ≠ 0) то:

- функция Serial.parseInt() приводит принятые данные к типу integer;
- переменная dt получает новое значение.

3. ЭЛЕМЕНТЫ ЯЗЫКА ПРОГРАММИРОВАНИЯ Arduino IDE

На языке программирования формулируется задание для процессора по обработке данных. Данные размещаются в памяти или в регистрах процессора. Последовательность действий, которую должен выполнить процессор над данными для достижения конечного результата, называют *алгоритмом*, а запись алгоритма на языке программирования — *программой*. Элементарное действие процессора, задаваемое в программе, называют *оператором* или *командой*, а данные, над которыми выполняются действия — операндами. Для обращения к данным процессор использует адреса ячеек. При написании программ, адресам ячеек могут присваиваться символические имена. Имя — это последовательность алфавитно-цифровых символов, заменяющая в операторе адрес операнда.

При оформлении программ используются специальные символы:

- «;» — конец оператора или объявления переменной и константы;
- «{ *опер* }» — фигурные скобки объединяют группу операторов *опер* в блок;
- «//» — начало однострочного комментария;
- «/* *текст* */» — ограничители многострочного комментария.

3.1. Типы данных

В зависимости от точности представления числовых данных и особенностей их обработки командами, они могут занимать в памяти одну или несколько смежных ячеек. Специфические особенности операндов задаются типом данных (операндов). Основные типы данных приведены в табл. 3.1.

Таблица 3.1

Тип	Размер ячейки (байт)	Диапазон значенийе	Назначение
boolean	1	TRUE / FALSE (1/0)	Логические значения
char	1	−128...+127	Символы и целые числа
byte	1	0...255	Натуральные числа
int	2	−32 768...+32 767	Целые числа
unsigned int	2	0...65535	Натуральные числа
long	4	−2147483648... +2147483647	Длинные целые числа
float	4	−3,4028235E+38... +3,4028235E+38	Вещественные числа

Данные подразделяются на константы и переменные в зависимости от того, могут изменяться их значения при выполнении программы или нет.

3.2. Объявление переменных

До использования переменных (и констант) в программе они должны быть объявлены. При объявлении переменной необходимо указать ее тип и имя. Например, в строках

```
int i, j, k;
char ch, chr;
```

объявляются три переменные (i, j, k) типа int и две переменные (ch и chr) типа char. При объявлении можно задать начальное значение переменной:

```
int pinLed = 13;
```
— переменной pinLed присваивается начальное значение 13.

Однотипные переменные могут быть объединены в массивы. При объявлении массиву присваивается тип, имя и указывается число элементов массива в квадратных скобках:

int arr [6]; — массив из шести элементов типа int с именем arr.

Возможно задание начальных значений элементам массива при объявлении:

```
char arrCh[]={2, –4, 6,–5, 10};
```

Для выбора нужного элемента массива используются их порядковые номера (индексы), начиная с нуля. Так, второй элемент массива arrCh[2] имеет значение «6».

Операторы преобразования типов

При необходимости, в процессе работы скетча тип переменной var можно изменить оператором преобразования типов:

```
newType(var);
```

где newType = {char, byte, int, float, long} преобразует переменную var к типу char, byte, int, float или long соответственно.

3.3. Константы

Константами называют фиксированные значения. В команде константа может задаваться в виде числа. Целочисленные константы могут задаваться в разных системах счисления, например: 123 — десятичная система; B1111011 — двоичная система (используется префикс «B»); 0x7B — шестнадцатеричная система (с префиксом «0x»).

Целочисленные константы имеют тип int. Беззнаковые константы задаются с постфиксом «u»: 129u. В двоичной системе константы имеют однобайтовый формат. Константы с плавающей точкой записываются в виде десятичной дроби или в показательной форме, например: 120.5; 1.205E2. По желанию константы объявляются так же, как переменные, с присвоением типа и имени:

```
const float pi=3.14;
```

Квалификатор const делает объявленную величину доступной только для чтения.

Предопределённые константы

В языке программирования по умолчанию заданы имена некоторым константам.

– Логические константы:

FALSE — определяется как нуль в логическом выражении;
TRUE — определяется как число, отличное от нуля.

– Уровни цифровых сигналов:

HIGH — уровень сигнала логической единицы (≈5 В);
LOW — уровень сигнала логического нуля (≈0 В).

– Направление передачи сигнала через цифровой порт:

INPUT — настройка порта на ввод данных;
OUTPUT — настройка порта на вывод данных;
INPUT_PULLUP — ввод данных с подключением подтягивающего резистора.

3.4. Функции

Обособленная группа операторов, предназначенная для выполнения сравнительно более сложных действий над операндами, называется *функцией*. Функция является основной структурной единицей программы. Функции помогают поделить скетч на управляемые и простые в использовании фрагменты. В процессе работы функция может вызывать другие функции. После выполнения всех операторов вызванная функция может возвращать нужное значение.

При описании функции указывается тип (type) возвращаемого значения, имя функции (name) и список имён входных переменных с указанием типов — формальных параметров:

```
type name (type1 dat1, ..., typeN datN) {
        // блок операторов функции
}
```

Пара фигурных скобок объединяет все операторы функции в единый блок. Набор входных данных может быть пустым.

Для вызова функции указывается её имя и в скобках перечисляются данные, с которыми должна оперировать вызываемая функция — список фактических параметров:

```
name (d1, ..., dn);
```

При переходе на вызванную функцию формальные параметры получают значения фактических в порядке их перечисления.

По степени доступности различают глобальные и локальные переменные. Глобальная переменная объявляется вне функции, и она видна (доступна) из любой функции скетча. Локальная переменная видна только в той функции, в которой она объявлена.

При выходе из функции значение локальной переменной теряется. Если требуется сохранить текущее значение локальной переменной до следующего вызова функции, то объявление локальной переменной (var) должно выполняться с квалификатором static:

```
static int var;
```

3.5. Функции setup и loop

В любом скетче обязательно присутствуют две функции — setup и loop. Они ничего не возвращают (имеют тип void) и не используют входных данных. Функция setup вызывается только один раз, после подачи питания или сброса платы сигналом Reset. Она используется для начального определения режимов работы контактов платы, инициирования работы периферийных модулей и т.п. Функция loop автоматически вызывается после функции setup. Она выполняется в бесконечном цикле, организуя активное управление платой Arduino:

```
void setup() {
  // начальная настройка платы
}
void loop() {
  // основной программный код
}
```

Для структурной организации скетча могут разрабатываться дополнительные функции, вызываемые из основных функций setup и loop.

3.6. Операторы

3.6.1. Арифметические операторы

Арифметические операторы представляются символами математических операций (табл. 3.2).

Таблица 3.2

Символ	Действие	Пример	Результат
=	Присвоение	x=3;	Значение x равно 3
+	Сложение	y=x+5;	Переменная y равна 8
–	Вычитание	z=y–x;	Переменная z равна 5
*	Умножение	y=z*x;	Переменная y равна 15
/	Деление	y=y/2;	Переменная y равна 7
%	Остаток от деления	x=y%2;	Переменная x равна 1

Слева от символа операции присвоения указывается переменная, которой будет присвоен результат выполнения арифметического действия над операндами. В одном операторе может быть указано несколько арифметическом операций. Арифметические действия выполняются с учётом старшинства. Круглыми скобками можно изменить порядок вычислений.

3.6.2. Логические и битовые операторы

Структура логических операторов подобна структуре арифметических операторов. В качестве операндов используются логические переменные, над которыми могут выполняться логические операции (табл. 3.3).

Результат — логическое значение TRUE или FALSE, фиксируется в переменной слева от символа операции присвоения или используется в условных выражениях.

Битовые операторы выполняют логические операции над битами двоичных чисел (табл. 3.4).

Таблица 3.3

Символ	Действие	Пример	Результат
	объявление	boolean x=3; boolean y=0;	x = =TRUE y= =FALSE
&&	коньюнкция	z= x && y;	z= =FALSE
||	дизьюнкция	z= y || x;	z = =TRUE
!	инверсия	z= ! y;	z = =TRUE

Таблица 3.4

Символ	Действие	Пример	Результат
&	И	10&3;	B00000010
|	ИЛИ	10|3;	B00001011
^	искл. ИЛИ	10^3;	B00001001
~	NOT	~10;	B11110101
<<	сдвиг влево	10<<3;	B01010000
>>	сдвиг вправо (char)	10>>3; 138>>3;	B00000001 B11110001

3.6.3. Операторы управления

Операторы управления нарушают последовательное выполнение команд, организуя ветвящиеся алгоритмы и циклические вычисления. Далее приводится выборка из доступных операторов языка C++.

1) Оператор выбора if

Оператор if позволяет программе делать выбор между альтернативными блоками операторов, заключёнными в фигурные скобки. После слова if в круглых скобках записывается логическое выражение (условие). Если это выражение истинно (TRUE), выполняется первый блок. Если выражение ложно (FALSE), выполняется

второй блок, размещённый после ключевого слова else (в частных случаях группа else может отсутствовать).

Пример: если X>5 — выполняется первый блок, если X≤5 — выполняется второй блок:

```
if (>5) { // первый блок (условие==TRUE)
}
else { // второй блок (уловие==FALSE)
}
```

Условное выражение составляется из операций сравнения и логических операторов. Операции сравнения двух величин формируют логическое значение TRUE, если (табл. 3.5):

Таблица 3.5

Условие	TRUE если
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 или равно y
x >= y	x больше или равно y

Если условное выражение содержит несколько операций сравнения, то результаты сравнений объединяются логическими операторами, которые формируют значение условие == TRUE / FALSE.

2) Оператор выбора switch

Оператор позволяет задавать альтернативный код, выполняемый при разных условиях. Конструкция switch (val) поочерёдно сравнивает значение переменной val со значениями, заданными в блоках case Y=[y1, y2, ..., yM]. Если val== yN, то выполняются программные коды от N-блока до ключевых слов break или default, а если ключевые слова не встречаются, то до конца оператора switch.

От ключевого слова default и до конца оператора размещаются команды, которые выполняются, если нет ни одного совпадения.

```
switch (val) {
case y1:
  // блок 1 при val =y1
  break;
case y2:
  // блок 2 при val =y2
   break;
default:
  // блок при val ∉Y }
```

3) Оператор цикла for

Конструкция for (*init; cond; incr*) {*операторы*} используется для циклического повторения блока в фигурных скобках:

- выражение *init* присваивает счётчику цикла val начальное значение;
- выражение *cond* задаёт условие окончания циклических вычислений;
- оператор *incr* изменяет значение *val* после каждого выполнения блока.

Например, блок цикла будет выполняться шесть раз:

```
for (int i=0; i <= 5; i=i+1) {
   //операторы цикла
}
```

4) Оператор break

Break используется для принудительного выхода из цикла, не дожидаясь завершения по условию. Например, цикл выполняется, пока значения ($y \le 100$) и ($x < 255$):

```
for (int x=0; x < 255; x=x+1){
    y=y*x;
    if (y>100) {break;}
}
```

5) Оператор continue

Оператор continue пропускает оставшиеся операторы блока на текущем шаге цикла и возвращается к проверке условного выражения, которая происходит при каждой следующей итерации. Например, операция (y=x*x) будет выполняться, если параметр цикла x не лежит в интервале [41, 119]:

```
for (x=0; x<250; x=x+1) {
   if (x<40 && x>119) {continue;}
   y=x*x;
}
```

6) Оператор return

Прекращает работу вызываемой функции и передаёт управление вызывающей. Оператор может возвращать значение переменной любого типа. Например, функция func возвращает 1, если (x > 400), и 0 — если (x <= 400).

```
int func() {
   if (x > 400) return 1;
   else return 0;
}
```

7) Оператор goto

Передача управления. Например, оператор goto передаёт управление оператору, помеченному меткой label:

```
label: x=x+10;
       //операторы
       goto label;
```

3.7. Встроенные функции

В языке программирования имеются встроенные функции, упрощающие работу с контроллером ATmega328.

3.7.1. Функции цифрового ввода/вывода

- `pinMode (pin, mode);` — устанавливает режим `(mode)` ввода или вывода `(INPUT` или `OUTPUT )` цифровому выводу с номером `pin`.
- `digitalWrite (pin, value);` — передаёт значение `(value) HIGH` или `LOW` на цифровой контакт `pin`. Если `pin` настроен на `INPUT`, то при передаче `HIGH` контакт `pin` будет к подключен к питающему напряжению 5 В через подтягивающий резистор 20 кОм.
- `digitalRead (pin);` — считывает значение `(HIGH` или `LOW)` с указанного контакта.

3.7.2. Функции аналогового ввода/вывода

analogRead (pin); — считывает значение с аналогового входа. Входное напряжение 0...5 В будет представлено числом типа int в диапазоне 0...1023.

analogWrite (pin, value); — выдаёт ШИМ-сигнал через вывод pin. Частота импульсов ≈490 Гц. Скважность импульсов задаётся величиной value = 0...255.

3.7.3. Дополнительные функции ввода/вывода

- tone (pin, freq); и tone (pin, freq, durat); — генерирует прямоугольные импульсы со скважностью 0,5. Частота сигнала в герцах задаётся переменной freq. Одновременно может выдаваться только один сигнал. Функция блокирует вывод ШИМ-сигнала на контактах 3 и 11. Дополнительно переменной durat может быть задана продолжительность генерации в миллисекундах.
- noTone (pin); — останавливает генерацию сигнала функцией tone.
- pulseIn (pin, value); — ожидает появления на контакте pin импульса value={HIGH, LOW} и возвращает его длительность в микросекундах. Если импульс отсутствует в течение 1 с, возвращается значение 0.

3.7.4. Функция работы со временем

- delay (ms); — приостанавливает работу программы на ms миллисекунд.
- delayMicroseconds (us); — приостанавливает работу программы на us микросекунд.

3.7.5. Математические функции

- min (value1, value2); — возвращает меньшее из двух значений.
- max (value1, value2); — возвращает большее из двух значений.
- abs (value); — возвращает абсолютное значение числа.

- map (valu, aLow, aHigh, bLow, bHigh); — пропорционально переносит значение valu из текущего диапазона [aLow...aHigh] в новый диапазон [bLow...bHigh] и возвращает его.
- sq (valu); — возведение valu в квадрат.
- sqrt (valu); — извлечение квадратного корня из valu.
- sin (rd); — вычисление синуса угла rd, заданного в радианах.
- cos (rd); — вычисление косинуса угла rd.
- tan (rd); — вычисление тангенса угла rd.

3.7.6. Функции генерирования случайных чисел

- randomSeed (val); — инициализирует генератор псевдослучайных чисел. Первое число последовательности зависит от параметра val.
- random (min, max); — возвращает псевдослучайное число из диапазона [min, max-1].

3.7.7. Функции поддержки внешних прерываний

Процедуры обработки прерываний не должны возвращать значений (квалификатор void), и они не имеют списка формальных параметров. Если для их работы требуются данные из основного скетча, то эти данные объявляются глобальными и защищёнными от изменений оптитмизирующим компилятором (квалификатор volatile).

- attachInterrupt (*inter, func, mode*); — задаёт функцию обработки прерывания, где *inter* — номер прерывания: 0 (D2) или 1 (D3); *func* — имя функции обработки прерывания типа void; *mode* — режим вызова прерывания:
  LOW — низкий уровень на входе порта,
  CHANGE — любое изменение уровня,
  RISING — возрастающий уровень на входе,
  FALLING — падающий уровень.
- detachInterrupt (inter); — выключает обработку прерывания.

Для обработки запросов прерывания от встроенных модулей микроконтроллера можно использовать специализированные библиотеки. Напрмер, для работы с модулем таймера T1 существует библиотека TimerOne.

4. СПЕЦИАЛИЗИРОВАННЫЕ БИБЛИОТЕКИ

В языке прграммирования предусмотрено большое количество функций для решения типовых задач. Эти функции объединены в специализированные библиотеки. Чтобы использовать нужную функцию в прикладной программе, необходимо подключить соответствующую библиотеку к проекту.

Для подключения библиотеки к прикладной программе в текст программы нужно добавить директиву

```
#include.<elibr.h>
```

где elibr.h — заголовочный файл библиотеки elibr, в котором размещаются прототипы библиотечных функций. Другим вариантом является выбор библиотеки из меню Sketch > Import Library.

Любое внешнее устройство можно описать набором данных, характеризующих это устройство, и набором функций для обработки этих данных. Такие данные и функции объединяются понятием «класс» (class) — особый тип данных языка программирования. Внешнее устройство является объектом, если оно соответствует данным и функциям (методам) определённого класса. Объектов одного класса может быть несколько, но данные объектов размещаются в разных областях памяти и не пересекаются.

В Arduino IDE задано большое количество классов для различных внешних устройств. В объявлении объекта указываются класс и имя создаваемого объекта:

```
className objName;
```

Для вызова нужной функции-члена класса нужно указать имя объекта и имя функции:

```
objName.funcName();
```

4.1. Библиотека *Serial*

Библиотека подключается к скетчу автоматически и содержит набор функций для организации связи платы с другими устройствами по последовательному каналу USART через контакты 0 (RX) и 1 (TX). Обычно по каналу USART плата соединяется с компьютером.

- Serial.begin (*speed*); — организует соединение и задаёт скорость обмена в бодах из стандартного ряда. Часто выбирают speed=9600.
- Serial.end (); — закрывает последовательное соединение.
- Serial.available (); — возвращает количество байтов, доступных для чтения.
- Serial.read(); — возвращает очередной доступный для чтения байт. Если доступных байтов нет, то возвращается минус 1.
- Serial.print (*value*); — передаёт данные value как ASCII текст.
- Serial.println (*value*); — передаёт данные value как ASCII текст и переносит строку.
- Serial.parceInt (); — ждет 1 с во входном потоке код ASCII целого числа. Если код не поступил, возвращается нуль.
- Serial.write (*value*); — передаёт данные как бинарный код.

4.2. Библиотека *Servo*

Набор функций позволяет просто управлять сервоприводом. Для использования библиотеки необходимо объявить объект класса Servo:

- Servo objName, где objName — имя создаваемого объекта.
- objName.attach (*pin*); — подключает привод к указанному выходу *pin*, с которого осуществляется управление приводом.
- objName.write (*angle*); — передает значение *angle*=[0...180] для управления приводом: угол поворота или угловая скорость.
- objName.detach(); — отсоединяет привод от указанного выхода.

4.3. Библиотека *Wire*

Библиотека предоставляет функции для связи платы с дополнительными модулями по интерфейсу I^2C. Стандарт шины I^2C допускает подключение нескольких различных устройств. Плата Arduino выступает в роли ведущего устройства, под управлением которого выполняется обмен данными. Все ведомые устройства имеют уникальные адреса. На плате Arduino интерфейс связан с контактами A4 — линия данных SDA , и A5 — линия тактирования SCL.

Для запроса *n* байтов данных от ведомого устройства *addr* нужно выполнить команду Wire.requestFrom (*addr, n*). Далее, с помощью

команды Wire.available() нужно определить, сколько байтов действительно получено.

Если получено нужное число байтов, их можно считать по одному командой Wire.read().

- Wire.begin() — инициализация библиотеки. Плата — ведущее устройство.
- Wire.requestFrom(*addr, n*) — запрос данных для считывания командами read и available: *addr* — адрес ведомого; *n* — число запрашиваемых байтов. Возвращается число доступных байтов.
- Wire.available() — возвращает количество байтов, готовых для чтения. Вызывается после Wire.requestFrom.
- Wire.read() — чтение байта данных.
- Wire.beginTransmission(*addr*) — начало передачи данных. Следующие данные ставятся в очередь функцией Wire.write(). Передача пакета — функция Wire.endTransmission().
- Wire.write(val) — ведущее устройство ставит данные в очередь. val=[v|s|d,l], где v — байт; s — строка байтов; d, *l* — массив из *l* байтов.
- Wire.endTransmission() — завершение процедуры передачи данных. Возвращает один байт: 0 — успешная передача.

4.4. Библиотека *Ultrasonic*

Библиотека предназначена для работы с ультразвуковым дальномером.

- Ultrasonic name(t, e); — объявление объекта Ultrasonic с именем name. Устройство подключается к цифровым контактам: вывод Trig к контакту *t* и вывод Echo к контакту e.
- name.distanceRead(); — команда запускает процесс измерения и возвращает величину расстояния до препятствия в сантиметрах.

4.5. Библиотека *TimerOne*

Библиотека предназначена для работы с Таймер/счётчиком 1.

- Timer1.initialize(*n*); — включение и установка периода срабатывания таймера *n* мкс.

- Timer1.attachInterrupt(*proc*); — указание процедуры обработки прерывания *proc*.
- Timer1.pwm(*pin, duty, period*); — включение ШИМ на контакте *pin* с заполнением *duty* ={0...1023} и периодом *period*.
- Timer1.stop(); — останавливает таймер.

5. ПОДКЛЮЧЕНИЕ ВНЕШНИХ УСТРОЙСТВ

5.1. Подключение бинарного датчика

Режим работы скетча может изменяться под действием внешних условий. Количественная оценка этих условий производится входными преобразователями или датчиками. В качестве простого бинарного датчика может использоваться электрическая кнопка. Подключим кнопку между контактами GND и 7 на плате (рис. 5.1).

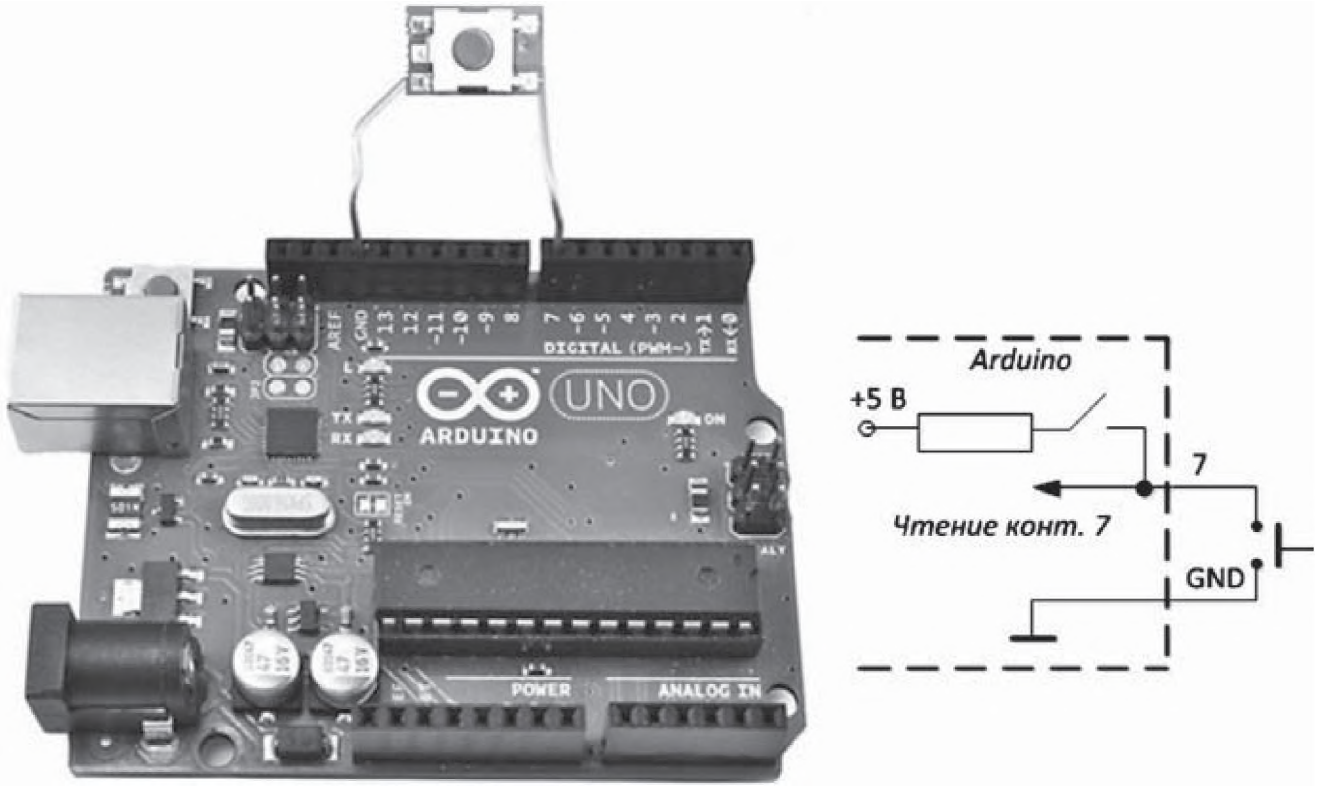

Рис. 5.1

Заметим, что при подключении внешних устройств к плате на лабораторном стенде рекомендуется использовать монтажную плату. На плате имеется 34 ряда закороченных гнёзд по пять гнёзд в каждом ряду.

На контакте 7 возможны только два логических уровня — TRUE или FALSE.

Контакт 7 программно должен быть подключен к напряжению 5 В через подтягивающий резистор R ≈ 40 кОм. Этим на контакте создаётся уровень HIGH при отпущенной кнопке. При нажатой кнопке на контакт подаётся уровень LOW.

Предлагаемый скетч включает четыре фрагмента. В *первом фрагменте* объявляются имена контакта 13 — для подключения светодиода и 7 — для подключения кнопки:

а) объявить переменную ledPin типа int и связать её с контактом 13;

б) объявить переменную switchPin типа int и связать её с контактом 7.

Во *втором фрагменте* описывается функция setup начальной настройки платы:

а) объявить функцию setup;

б) открыть блок функции setup и вызвать функцию настройки контакта ledPin на цифровой вывод;

в) настроить контакт switchPin на цифровой ввод с подключением подтягивающего резистора и закрыть блок функции setup.

В *третьем фрагменте* объявляется основная функция loop, которая не возвращает значений и не использует фактических параметров. Функция выполняет в бесконечном цикле опрос кнопки и может вызывать другие функции для реализации заданного алгоритма:

а) объявить функцию loop и открыть блок операторов;

б) выполнить цифровое чтение контакта switchPin и определить его состояние — если на контакте уровень LOW, то вызвать функцию flash с фактическим параметром 100, а если на контакте уровень HIGHT, то вызвать функцию flash с фактическим параметром 500;

в) закрыть блок операторов loop.

Четвёртый фрагмент содержит функцию flash одного цикла мигания светодиода — включение/выключение. Функция не возвращает значения и использует один формальный параметр. Длительность цикла определяется величиной паузы, которая передаётся в подпрограмму как фактический параметр:

а) объявить функцию flash с формальным параметром, определяющим величину паузы в работе функции;

б) открыть блок операторов функции и вывести на ledPin уровень HIGHT;

в) вызвать функцию паузы с полученным значением формального параметра;

г) вывести на ledPin уровень LOW;

д) вызвать функцию паузы и закрыть блок операторов. После выполнения последнего оператора блока управление автоматически возвращается в вызвавшую функцию loop.

Для уяснения указанных шагов приведём текст скетча на языке программирования:

```
int ledPin = 13;                  // присвоить имя ledPin контакту 13
int switchPin = 7;                // присвоить имя switchPin контакту 7
void setup() {
  pinMode(ledPin, OUTPUT);     //настроить контакт 13 на вывод
  pinMode(switchPin, INPUT_PULLUP);
       /* настроить контакт 7 на ввод и подключить его к
          подтягивающему резистору — параметр INPUT_PULLUP
*/
}
void loop() {
  if (digitalRead(switchPin) == LOW) {
   flash(100); //если на switchPin уровень LOW — выполнить функцию
               //flash с фактическим параметром 100
 }
else {
 flash(500); //если на switchPin уровень LOW — выполнить функцию
             //flash с фактическим параметром 500
 }
}
/*Функция однократного включения и выключения светодиода. Формальный параметр delayPeroid задаёт длительность свечения. */

void flash(int delayPeriod) {
  digitalWrite(ledPin, HIGH); //установка на ledPin уровня HIGH
  delay(delayPeriod); //пауза длительностью delayPeriod (мс)
  digitalWrite(ledPin, LOW); //установка на ledPin уровня LOW
  delay(delayPeriod); //пауза длительностью delayPeriod (мс)
}
```

В функции loop используется оператор digitalRead для проверки уровня напряжения на входном контакте. Если он равен LOW (кнопка нажата), оператор if вызывает функцию flash с параметром 100. Это заставляет светодиод мигать чаще.

При отпущенной кнопке на входном контакте устанавливается уровень HIGH и выполняется блок в разделе else оператора if. Здесь вызывается та же функция flash, но с более продолжительной задержкой, заставляющей светодиод мигать реже.

Если сигналы HIGH и LOW будут формироваться внешним устройством, то необходимость в подтягивающем резисторе отпадает. В таком случае функции pinMode передаётся параметр INPUT вместо INPUT_PULLUP.

5.2. Аналоговые выходы

Для преобразования цифрового сигнала в аналоговую форму плата Arduino Uno использует широтно-импульсную модуляцию (Pulse-Width Modulation, PWM). Контакты (3, 5, 6, 9, 10 и 11), отмеченные на плате Arduino Uno значком «~», можно использовать как ШИМ выходы.

Управляя скважностью импульсов (рис. 5.2), на них можно сформировать аналоговые сигналы. Чем длиннее импульс при постоянном периоде, тем больше среднее значение напряжения на выходе.

Так как импульсы следуют с частотой ≈ 500 Гц, а большинство исполнительных устройств обладает свойствами фильтра НЧ,

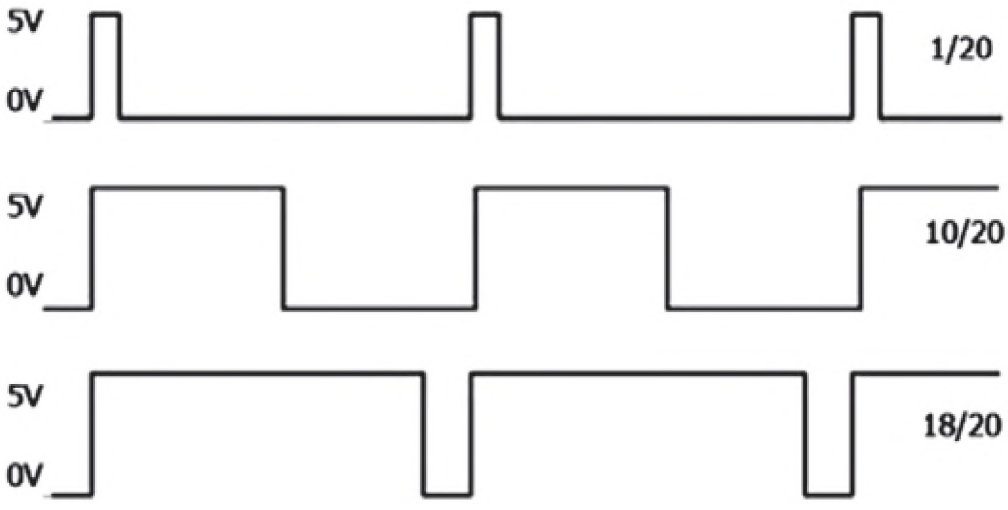

Рис. 5.2

возникает эффект вывода постоянного напряжения, равного среднему значению на периоде колебаний.

Для проверки работы ШИМ-генератора можно написать скетч и подключить осциллограф к выводу 6. Если открыть монитор последовательного порта и вводить числа в диапазоне 0...255 в текстовое поле, то на экране осциллографа можно наблюдать, как будет изменяться скважность импульсов при изменении вводимого числа.

В скетче нужно:

1) Настроить контакт 6 на вывод;

2) Активироать последовательный монитор;

3) Проверить готовность приёмника канала связи: Serial.available();

4) Если данные приняты, преобразовать их в целое число duty функцией Serial.parseInt();

5) Записать duty в ШИМ для вывода импульсов на контакт 6: analogWrite(6, duty);

6) Вернуться к пункту 3.

ШИМ-сигнал можно использовать для изменения яркости свечения светодиода, подключенного к контакту 9 (рис. 5.3):

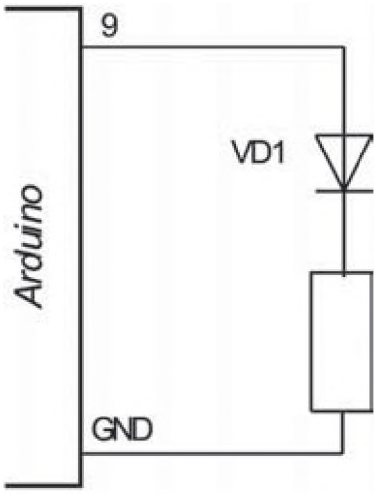

Рис. 5.3

1) Настроить контакт 9 на вывод;

2) Организовать цикл for с увеличивающимся параметром duty $\in$ [0, 255]. В блоке цикла установить яркость свечения (analogWrite(9, duty);), и паузу 10 мс;

3) Организовать цикл for с уменьшающимся параметром duty. В блоке цикла установить яркость свечения и паузу 10 мс;

4) Перейти к пункту 2.

5.3. Аналоговые входы

Контакты с метками от A0 до A5 на плате Arduino можно использовать для измерения величины приложенного к ним напряжения. Уровень напряжения должен находиться в диапазоне от 0 до 5 В. Аналого-цифровое преобразование выполняется с помощью встроенной функции analogRead, которая возвращает число в диапазоне 0...1023: значение 0 соответствует напряжению 0 В, а значение 1023 — напряжению 5 В.

Чтобы преобразовать полученное число в величину входного напряжения, нужно определить коэффициент преобразования: $\Delta v = 1023/5 = 204{,}6$ 1/В.

Для отображения дробных чисел следует использовать тип данных float.

Подключим напряжение 3,3 В к входу АЦП через кнопку и контакт A0 (рис. 5.4), составим скетч. В скетче нужно:

а) присвоить аналоговому входу A0 имя analogPin;

б) написать функцию setup с настройкой обмена по последовательному каналу с частотой 9600 бод — Serial.begin(9600);

в) написать функцию loop, в которой:

- получить результат АЦП-преобразования входного напряжения функцией analogRead(analogPin);
- используя коэффициент преобразования 204,6, получить величину напряжения volt в формате float;
- вывести в монитор последовательного порта величину напряжения операцией Serial.println(volt);
- реализовать паузу длительностью 1 с.

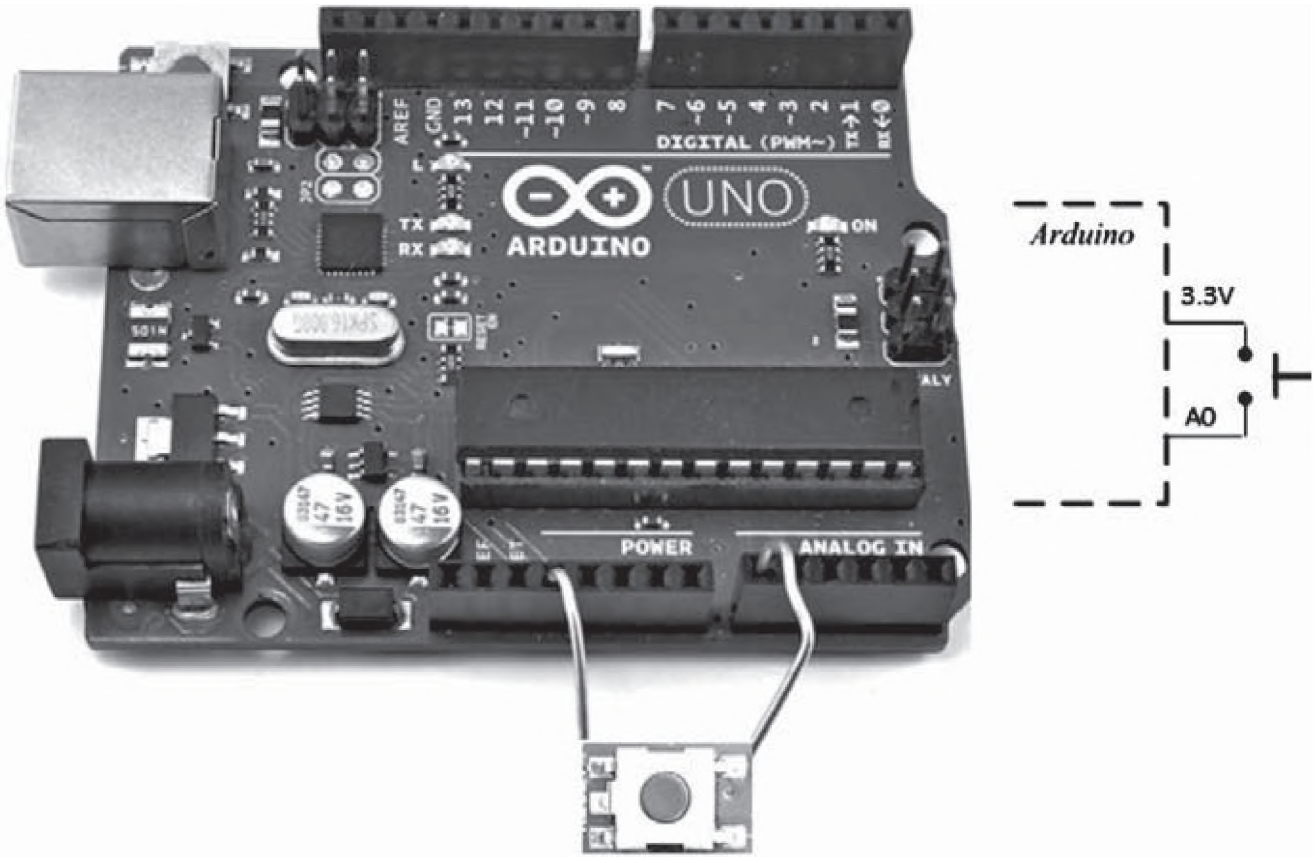

Рис. 5.4

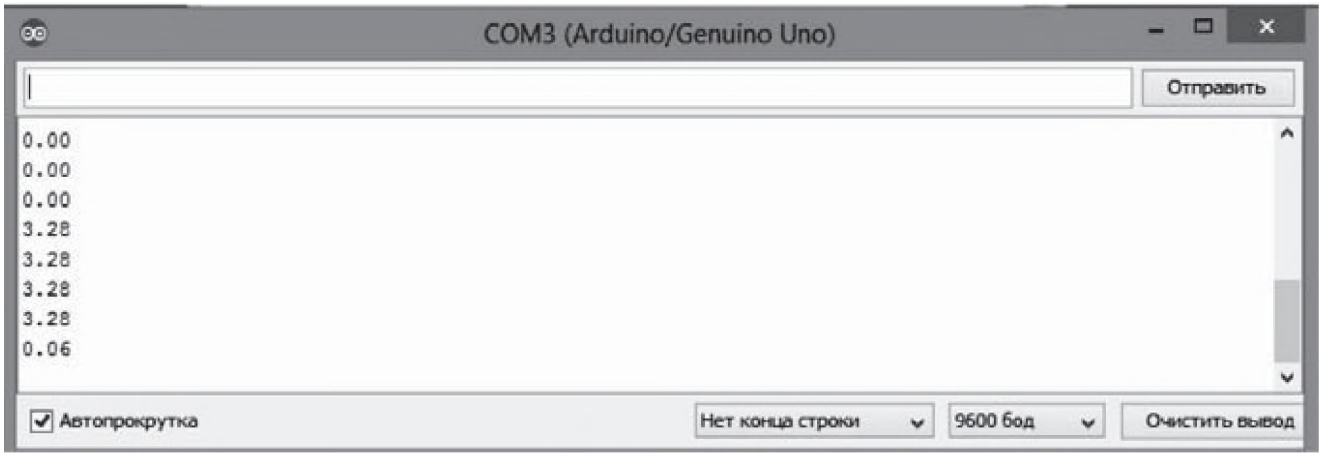

Рис. 5.5

После включения монитора на экране отображается входное напряжение при нажатой кнопке (рис. 5.5), а если кнопка отпущена — напряжение на *A*0 равно нулю.

5.4. Подключение резистивных датчиков

Многие схемы датчиков формируют выходной сигнал в виде электрического сопротивления: переменное сопротивление, терморезистор, фоторезистор, магниторезистор и др. Для дальнейшей обработки сопротивление преобразуется в электрическое напряжение. При этом используется схема резистивного делителя напряжения (рис. 5.6,*а*).

Резистивный делитель напряжения состоит из двух резисторов *R*1 и *R*0. От соотношения сопротивлений резисторов зависит выходное напряжение, снимаемое со средней точки делителя (*A*0).

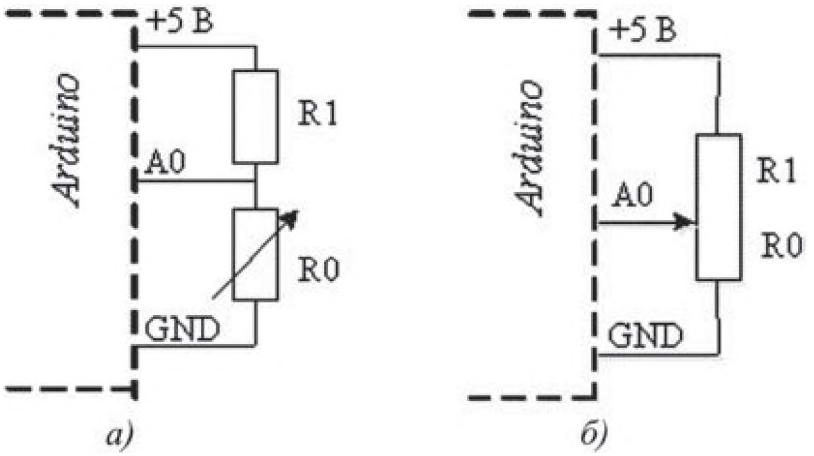

Рис. 5.6

Если один из резисторов переменный, то выходное напряжение может изменяться:

$$U_{A0} = f(U; R0) = \frac{(+5)\,R0}{R0 + R1}.$$

Как вариант, в качестве делителя напряжения может использоваться потенциометр (рис. 5.6,*б*). Входное воздействие — перемещение подвижного контакта вдоль рабочей поверхности потенциометра. При перемещении изменяется напряжение, снимаемое с контакта:

$$U_{A0} = f(U; x) = \frac{(+5)\,R}{L}\,x,$$

где R — номинальное сопротивление потенциометра; L — длина рабочей поверхности потенциометра; x — величина перемещения контакта.

5.5. Инфракрасные датчики препятствия и отражения

Датчики состоят из инфракрасного светодиода, фототранзистора и компаратора (рис. 5.7). Излучаемый светодиодом VD луч, отражаясь от препятствия, попадает на фототранзистор.

При освещении проводимость транзистора VT увеличивается, и напряжение на коллекторе уменьшается. Это напряжение сравнивается компаратором DA с напряжением на подвижном контакте потенциометра $R3$.

Если напряжение на коллекторе меньше, чем напряжение с движка потенциометра, то выход $D0$ компаратора устанавливается в состояние LOW. Если препятствие на значительном удалении, то фототранзистор освещается слабо и напряжение на коллекторе становится больше, чем на потенциометре. При этом на выходе $D0$ формируется уровень HIGH. Порог срабатывания компаратора регулируется потенциометром $R3$.

В датчике отражения, кроме цифрового вывода $D0$, предусмотрен вывод аналогового сигнала $A0$, который снимается с коллектора фототранзистора. Чем больше величина отражённого света, тем меньше уровень сигнала на аналоговом выводе $A0$.

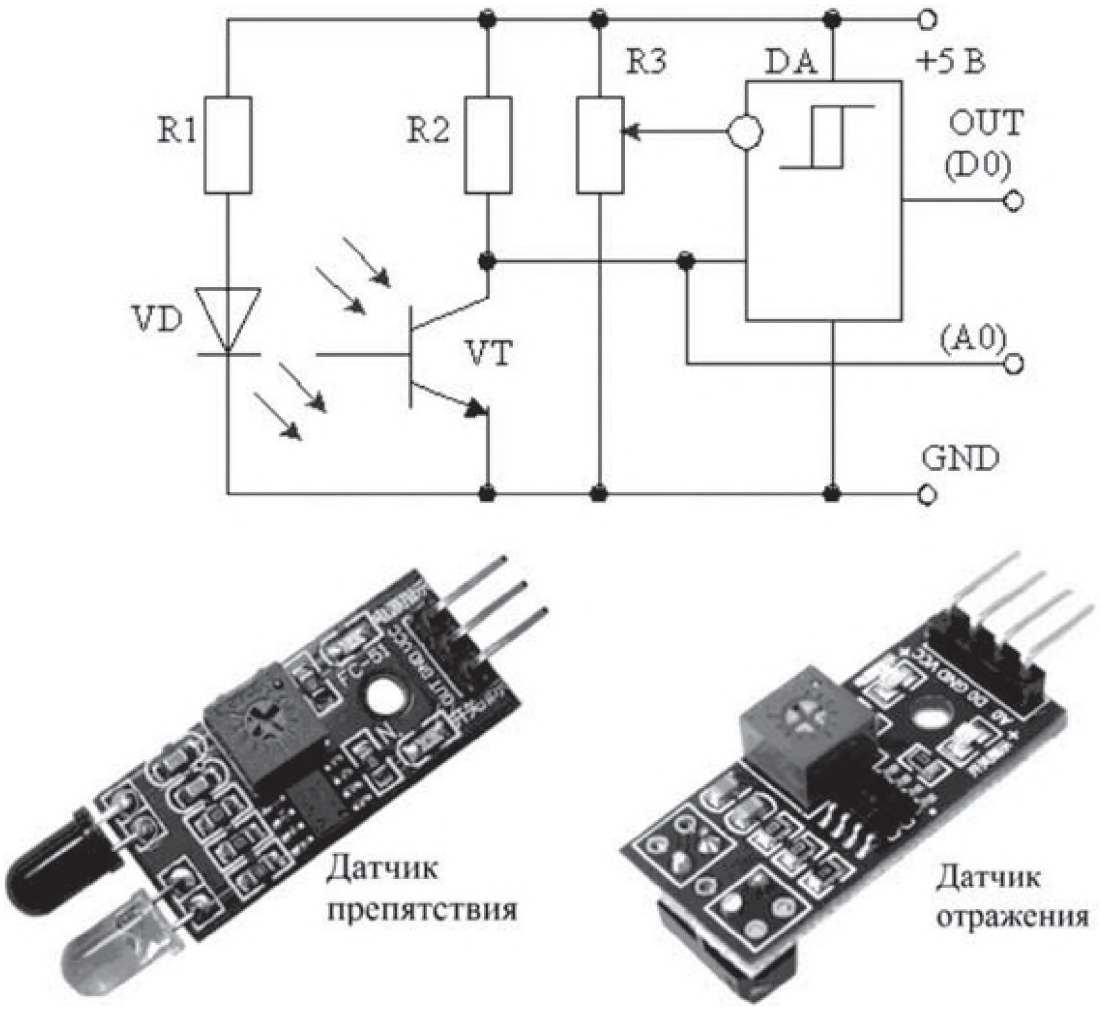

Рис. 5.7

5.6. Управление ультразвуковым дальномером

Дальномер предназначен для измерения расстояния до препятствия на линии распространения ультразвукового импульса. Излучатель отправляет импульс, который отражается от препятствия и поступает на приёмник. Время T, необходимое для прохождения имульса до препятствия и обратно, зависит от скорости звука V и расстояния L:

$$L = (VT)/2 = T/58 \text{ см},$$

где T — время в микросекундах.

Дальномер подключается к плате четырьмя проводниками: V_{CC} и GND — для подключения питания; Trig и Echo — цифровые сигналы управления излучателем и приёмником (рис. 5.8).

Для начала измерения на вход Trig подаётся импульс длительностью 10 мкс. Излучатель формирует ультразвуковой сигнал

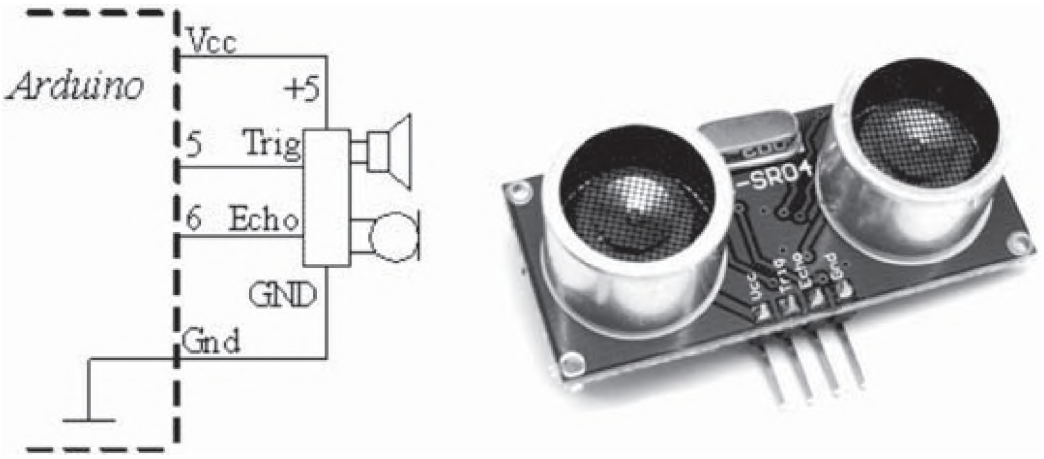

Рис. 5.8

с частотой 40 кГц длительностью 0,2 мс. На выводе Echo устанавливается уровень HIGH.

Приход отражённого импульса сбрасывает сигнал Echo в LOW. Время движения сигнала до препятствия и обратно равно длительности импульса с уровенем HIGH на выводе Echo.

Диаграмма направленности излучателя ограничивается углом ≈40°. Мощность излучателя и временные параметры позволяют измерять расстояния от 1 см до 4 м с погрешностью ≈1 см.

Для получения данных из дальномера напишем скетч:

а) присвоить имена echoPin и trigPin контактам 6 и 5;

б) написать функцию setup, в которой:

- настроить последовательный монитор на скорость 9600 бод;
- настроить контакты 5 и 6 на вывод и ввод соответственно;

в) написать функцию loop, в которой:

- объявить переменные tau и distSm типа int для описания длительности импульса и дальности;
- привести дальномер в исходное состояние подачей на контакт trigPin уровня LOW длительностью не менее 3 мкс. Для паузы использовать оператор delayMicroseconds(3):
- выдать на trigPin уровень HIGH длительностью 10 мкс;
- считать длительность импульса Echo: tau=pulseIn(echoPin, HIGH);
- вычислить дальность: distSm=tau/58;
- вывести дальность в последовательный монитор;
- выполнить паузу в 200 мс.

Для упрощения работы с УЗ-дальномером можно использовать специализированную библиотеку Ultrasonic.

5.7. Работа с акселерометром

Аналоговый акселерометр использует три измерительных канала на тензорезисторах. На тензорезисторы воздействуют силы инерции при ускоренном движении и сила притяжения к Земле. Три аналоговых сигнала с выходов X, Y и Z акселерометра подаются на аналоговые входы на плате Arduino, где преобразуются в числовые эквиваленты (рис. 5.9).

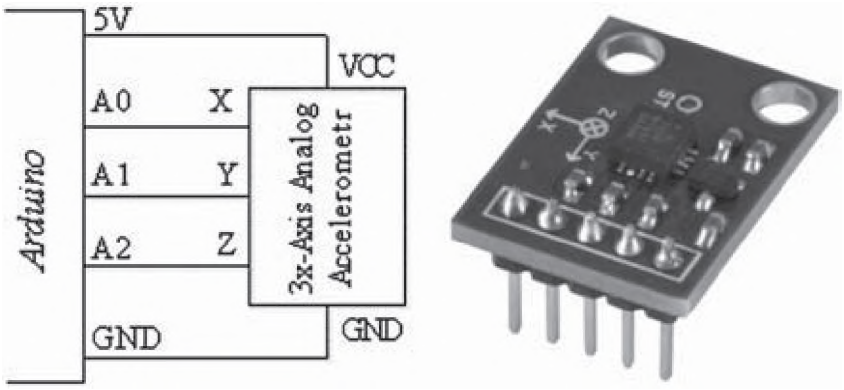

Рис. 5.9

Для определения коэффициентов преобразования измерительных каналов можно использовать силу тяготения при неподвижном датчике. Направляя координатные оси поочерёдно вертикально вверх и вниз, можно определить диапазоны изменения сигналов в каналах.

Выровнять диапазоны в каналах X, Y и Z можно используя встроенную математическую функцию «map».

Интегрируя сигналы с акселерометра, можно вычислить его перемещение в проекциях на оси.

Если ось Z направить перпендикулярно наклонной плоскости, а ось X — вдоль, то, используя проекции вектора силы тяготения по осям, можно вычислить угол наклона плоскости:

$$\alpha = \arctan\left(\frac{f_X}{f_Z}\right).$$

5.8. Работа с электронным магнитометром

Магнитометр использует три измерительных канала (x, y, z) на магниторезисторах. Вектор напряжённости магнитного поля

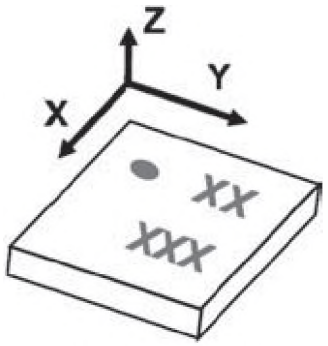

Рис. 5.10

раскладывается по трём ортогональным осям X, Y и Z (рис. 5.10). Направления осей X и Y указываются на шилде магнитометра.

Результаты измерений по осям, в условных числах, фиксируются в 16-разрядных регистрах данных. Для точных измерений датчик необходимо калибровать. Связь магнитометра с платой Arduino реализуется по интерфейсу TWI (I^2C). Плата является ведущим устройством, а магнитометр — ведомым. Встроенный адрес магнитометра на шине — 0x0D. Схема соединений приведена на рис. 5.11. Для работы с интерфейсом TWI предназначена библиотека Wire.

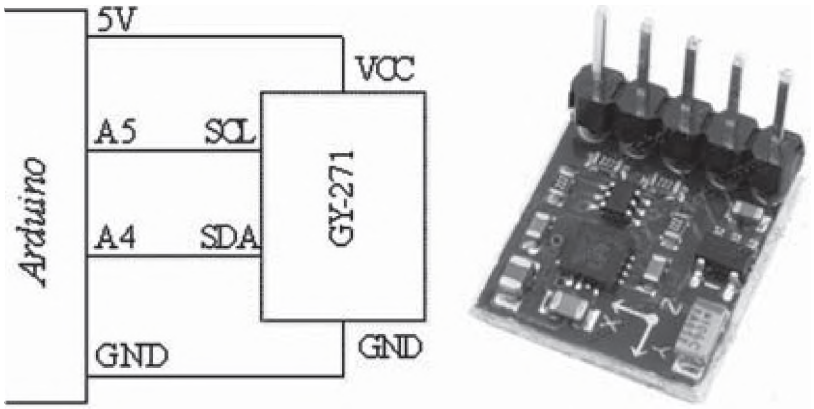

Рис. 5.11

Если вращать магнитометр вокруг оси Z, направленной вертикально вверх, то результаты измерений в каналах x и y будут периодически изменяться. При максимальном значении в канале x ось X будет указывать на север. В таком варианте магнитометр можно рассматривать как электронный компас.

Для управления микросхемой магнитометра используются 12 его регистров — 0...11. Шесть первых регистров используются для хранения результатов измерения:

0 — младший байт данных по оси X;

1 — старший байт данных по оси X;

2 — младший байт данных по оси Y;

3 — старший байт данных по оси Y;

4 — младший байт данных по оси Z;

5 — старший байт данных по оси Z.

Байт 6 содержит информацию о готовности результата измерения. Байты 7 и 8 хранят результаты измерения температуры. Через регистры 9, 10, 11 задаются параметры работы магнитометра — чувствительность, частота выдачи результата, непрерывное/одиночное преобразование и т.п. Для практической работы достаточно записать в регистр 9 число 0x11, а в регистр 11 — число 0x01.

При передаче данных в регистры магнитометра нужно соблюдать последовательность:

- передача адреса магнитометра: Wire.beginTransmission(0x0D);
- выбор нужного регистра: Wire.write(11);
- запись байта в регистр: Wire.write(1);
- закончить передачу: Wire.endTransmission();
- передача адреса магнитометра: Wire. beginTransmission(0x0D);
- выбор нужного регистра: Wire.write(9);
- запись байта в регистр: Wire.write(0x11);
- закончить передачу: Wire.endTransmission().

Для считывания данных из регистров нужно:

- передать адрес магнитометра: Wire. beginTransmission(0x0D);
- передать номер регистра, с которого нужно начать считывание:Wire.write(0);
- закончить передачу: Wire.endTransmission();
- запросить у магнитометра передачу нужного числа байтов: Wire.requestFrom(0x0D, 6);
- проверить готовность данных к чтению: Wire.available();
- если данные готовы — выполнить чтение: Wire.read()... Wire.read().

После считывания байтов из регистров данных нужно сформировать слова, содержащие результаты измерения по каналам (x, y, z), и вывести их через последовательный монитор.

5.9. Воспроизведение звука

Носителем звука является упругая среда (в частности — воздух), в которой распространяется продольная волна давления. Звуковая волна характеризуется частотой и амплитудой. Звукозаписывающая аппаратура преобразует звуковой сигнал в электрический, который может быть зафиксирован в разных формах для длительного хранения.

Звуковоспроизводящая аппаратура считывает параметры звуковой волны из устройства хранения и формирует электрический сигнал с соответсвующей частотой и амплитудой. Преобразование электрического сигнала в звук выполняется специальным устройством — динамиком.

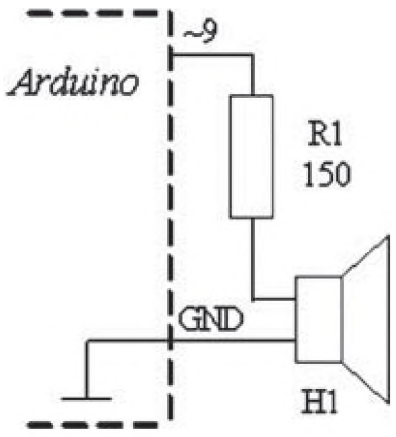

Рис. 5.12

Плата Arduino может хранить данные о звуковых колебаниях и выводить соответствующий электрический сигнал на один из контактов с функцией ШИМ — 3, 5, 6, 9, 10, 11. Для воспроизведения звука динамик подключается к нужному контакту (рис. 5.12).

Если активное сопротивление обмотки динамика $R_H = 8$ Ом, а ток, выдаваемый контактом $I = 40$ мА, то последовательно с динамиком надо подключить ограничительный резистор с сопротивлением

$$R1 = (U_{\text{ВЫХ}}/I_{\text{ВЫХ}} - R_H) = 5/4 \cdot 10^{-2} - 8 = 117 \approx 150 \text{ Ом}$$

— из стандартного ряда.

Дополнительно следует проверить ограничение по мощности динамика:

$$P = U_{\text{ВЫХ}} I_{\text{ВЫХ}} = 0{,}2 \text{ Вт}$$

— в пределах допустимого.

Мелодии формируются последовательностью воспроизведения звуковых фрагментов — нот. Ноты различаются частотой и длительностью звучания. Воспринимемый ухом звуковой диапазон разбивается на октавы, включающие по семь нот каждая.

Приведём названия нот с частотами первой октавы в Гц:

До — 261,7

Ре — 293,7

Ми — 329,6

Фа — 349,2

Соль — 392,0

Ля — 440,0

Си — 493,9.

Частоты нот второй октавы в два раза больше нот первой октавы, а частоты малой октавы — в два раза меньше.

Абсолютная длительность звучания ноты не задаётся. Длительность задаётся по отношению к «целой ноте» (1) — половинная нота (1/2), четвертная нота (1/4) и т.д.

Для примера запишем последовательность нот первой октавы с указанием длительности звучания:

Ми(1/4), Ми(1/4), Ми(1/2), Ми(1/4), Ми(1/4), Ми(1/2),

Ми(1/4), Соль(1/4), До(1/4), Ре(1/4), Ми(1).

В Arduino IDE есть встроенная функция для генерации звуков произвольной частоты. Функция tone(pin, freq, lng), где pin — номер выходного контакта, freq — частота звука, lng — длительность звучания. Функция tone поддерживает громкость звучания ноты на постоянном уровне в течение времени lng. Для разделения смежных нот длительность воспроизведения ноты должна включать время звучания lng и паузу pause. Соотношение между временем lng и pause подбирается экспериментально. Для примера на рис. 5.13 приведен график формирования ноты длительностью 400 мс.

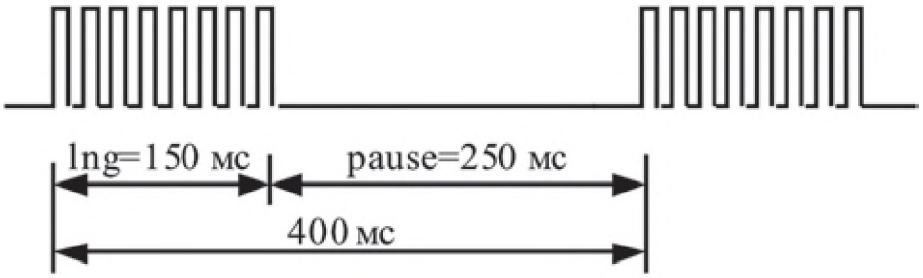

Рис. 5.13

Напишем простой скетч для воспроизведения мелодии на контакте 9 в нотах первой октавы. Мелодия хранится в трёх массивах — массив нот, массив длительности звучания и массив пауз. В скетче:

- присвоить переменным типа float названия нот и значения частот первой октавы;
- присвоить целой переменной t длительность звучания ноты 150 мс;
- присвоить целой переменной p длительность паузы между нотами 250 мс;
- объявить массив нот: {mi, mi, mi, mi, mi, mi, mi, sol, do_, re, mi};

- объявить массив длительности звучания: {t, t, t*2, t, t, t*2, t, t , t, t, t*4};
- объявить массив пауз между нотами:{p, p, p*2, p, p, p*2, p, p, p,p, p*4};
- настроить контакт 9 на вывод;
- организовать цикл с параметром i, задающим индексы элементов массивов;
- в цикле, используя функцию tone, выполнить последовательную выборку всех элементов массивов нот и дительностей звучания. После воспроизведения очередной ноты организовать паузу соответствующей длительности;
- организовать бесконечный пустой цикл.

6. УПРАВЛЕНИЕ БОЛЬШИМИ НАГРУЗКАМИ

6.1. Управление двигателем постоянного тока

Каждый из выводов Arduino может быть использован для питания устройств током до 40 миллиампер. Этого тока достаточно для питания светодиода, при большей нагрузке контроллер может выйти из строя. Для управления большими нагрузками (электродвигатели, лампы накаливания и др.) необходим усилитель мощности, например на МОП-транзисторе.

Подача напряжения на затвор транзистора может перевести его в режим насыщения (малое сопротивление) или отсечки (большое сопротивление).

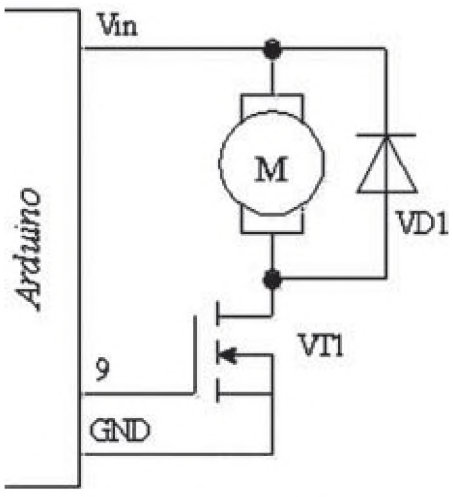

Рис. 6.1

На рис. 6.1 показано, как подключить полевой транзистор для включения и выключения маломощного двигателя вентилятора. Двигатель получает питание от разъёма 9 В (Vin) на плате Arduino. Так как транзистор подклю-чён к выводу 9, то можно использовать команду analogWrite() для управления скоростью двигателя при помощи ШИМ. Диод защищает транзистор от скачка напряжения при выключении двигателя.

Если подавать на затвор транзистора ШИМ-сигнал, то скорость вращения дви-

гателя будет пропорциональной коэффициенту заполнения импульсов.

Для изменения направления вращения ротора требуется изменить направление тока в обмотках. При однополярном питании изменить направление тока позволяет мостовой инвертор (рис. 6.2).

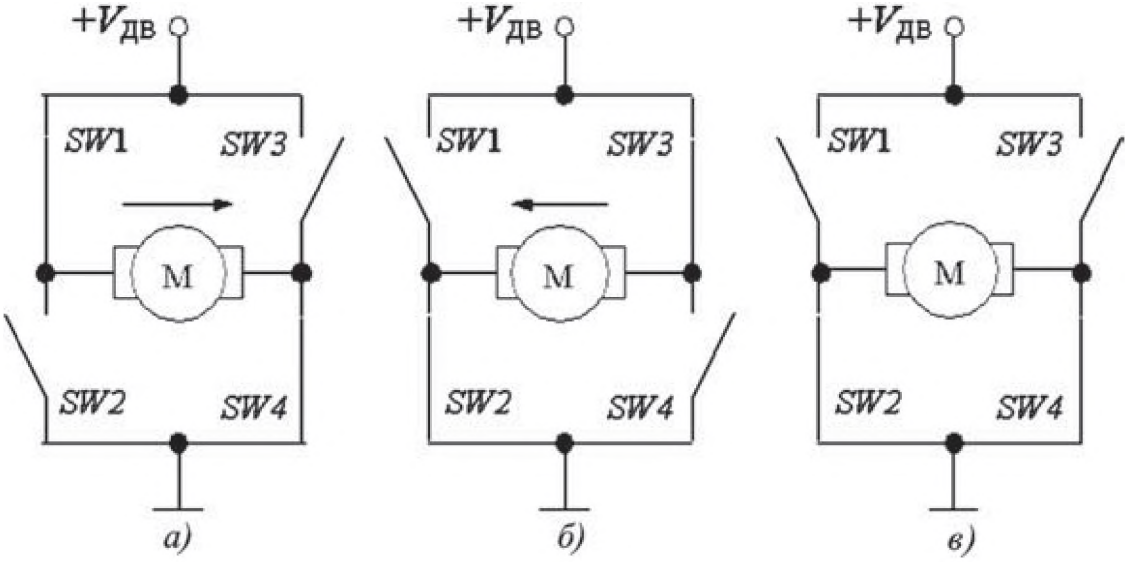

Рис. 6.2

Если одна пара ключей замкнута (SW1, SW4), а другая разомкнута (SW2, SW3) — двигатель вращается в одну сторону (рис. 6.2,*а*). Изменив состояние ключей на протвоположное (рис. 6.2,*б*), изменяем направление вращения. Для торможения двигателя замыкаются ключи (SW2, SW4) или (SW1, SW3), а оставшаяся пара ключей размыкается (рис. 6.2,*в*).

В схеме мостового инвертора (рис. 6.3) в качестве ключей используются полевые транзисторы VT1...VT4. Ключ замыкается при уровне HIGH на затворе. Сигналы на затворы подаются через вентили &.

Две схемы мостовых инверторов со стабилизатором напряжения объединены в модуле L298N. Он позволяет организовать два канала управления двигателями постоянного тока. Подключение модуля к плате Arduino показано на рис. 6.4.

При разрешающем сигнале EN=HIGH через вентили могут проходить управляющие сигналы In1 — левая пара ключей и In2 — правая пара ключей. Если In1,2=HIGH — замыкаются ключи SW1,3, а ключи SW2,4 — размыкаются. Изменение сигналов In1,2 на LOW изменяет состояние ключей на противоположное. Режимы работы двигателя сведены в табл. 6.1.

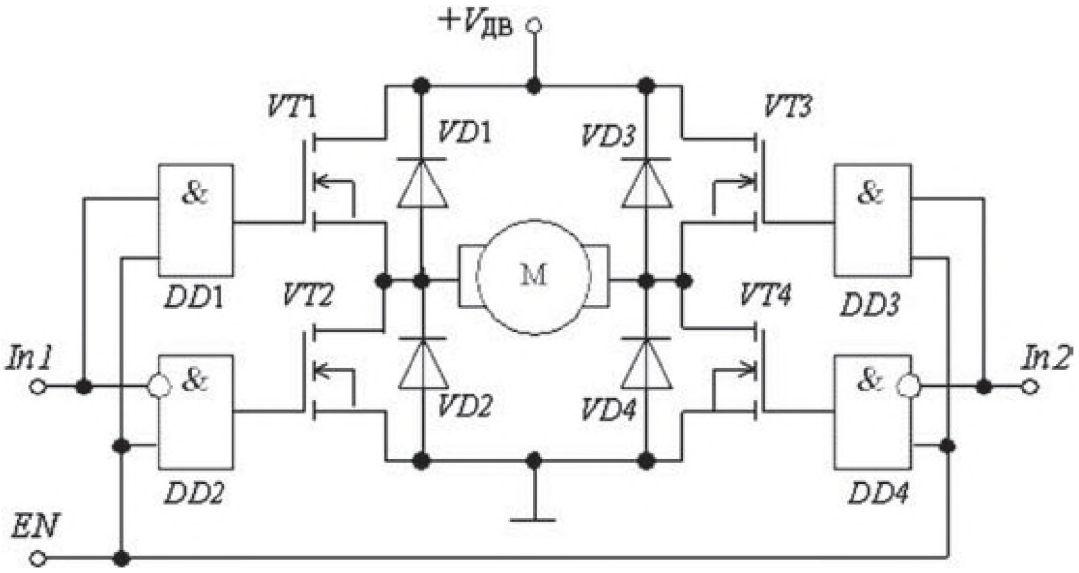

Рис. 6.3

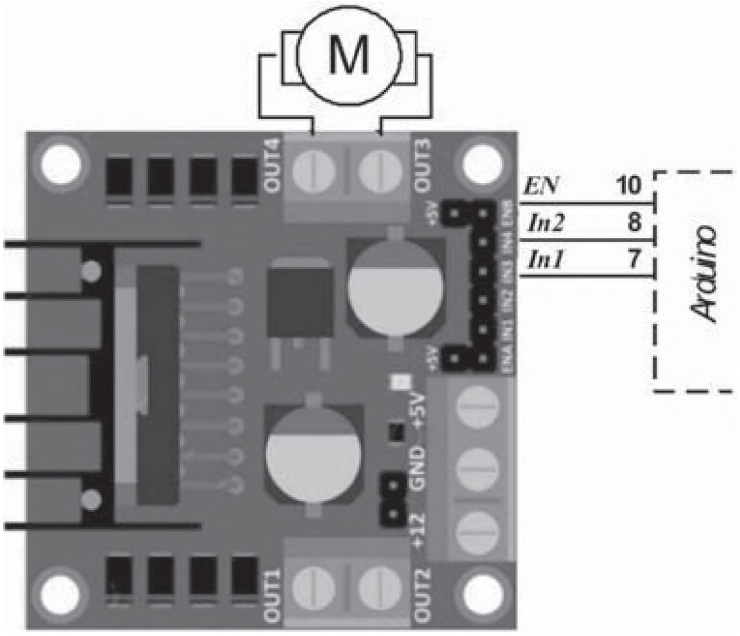

Рис. 6.4

Таблица 6.1

EN	In1	In2	Скорость ω
0	0/1	0/1	ω = 0
1	0	0	ω↓
1	1	1	ω↓
1	0	1	ω > 0
1	1	0	ω < 0

Скорость вращения может регулироваться ШИМ-сигналом, подаваемым на вход EN. Диоды VD1...VD4 защищают транзисторы от перенапряжения при выключении двигателя.

6.2. Управление серводвигателем

Серводвигатели предназначенны для точного позиционирования ротора. С помощью специального сигнала ротор можно установить в нужное положение, в котором он будет находиться, пока управляющий сигнал не изменится. Используя серводвигатель, можно развернуть на заданный угол исполнительный механизм или датчик.

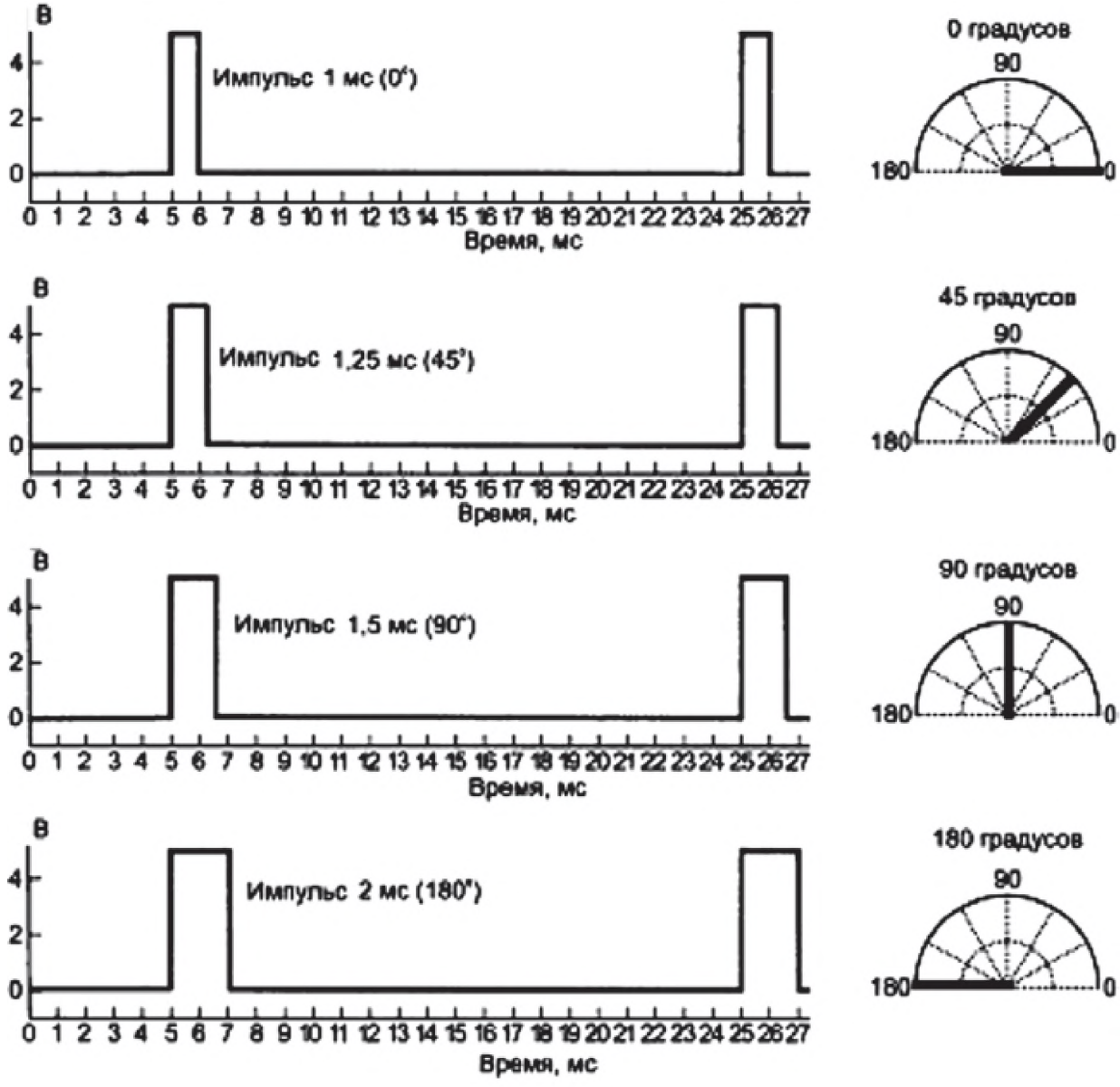

Рис. 6.5

Часто угол поворота ротора лежит в диапазоне 0...180°. Для управления двигателем используются импульсы определённой длительности и фиксированной частоты 50 Гц (период T = 20 мс).

Длительность импульса t задаёт угол поворота ротора (рис. 6.5):

при t = 1 мс угол поворота $\alpha = 0^\circ$;

при t = 1,5 мс угол поворота $\alpha = 90^\circ$;

при t = 2 мс угол поворота $\alpha = 180^\circ$.

Серводвигатель подключается к схеме тремя проводами: два провода подачи питания +5 В и GND; один провод для управляющих импульсов. Привод и контроллер должны иметь общую землю.

Для работы с сервоприводом предназначена библиотека Servo. Использование библиотеки исключает возможность использовать выходы 9 и 10 в режиме ШИМ, даже если привод не подключен к этим выводам.

7. СИСТЕМА ПРЕРЫВАНИЙ

7.1. Внешние прерывания

Периферийные устройства могут самостоятельно обращаться к контроллеру с запросом на обслуживание. Программные и аппаратные средства обработки запросов составляют систему прерываний. Появление запроса вызывает прерывание работы процессора по основной программе и переход на подпрограмму обслуживания внешнего устройства — процедуру обработки прерывания. Выполнив процедуру, процессор возвращается к выполнению прерванной программы.

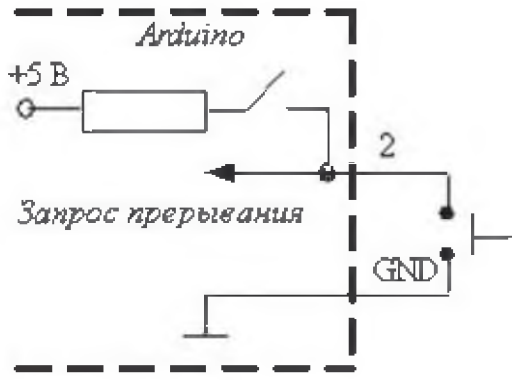

Рис. 7.1

В Arduino Uno с аппаратными прерываниями связаны два цифровых контакта — 2 и 3. Запросам прерывания присвоены имена 0 (контакт 2) и 1 (контакт 3). Эти имена используются в соответствующих операциях.

На рис. 7.1 изображена схема формирования запроса прерывания от кнопки. Через подтягивающий резистор на контакт 2 подается напряжение HIGH. При нажатии кнопки контакт заземляется и уровень напряжения на нем падает до LOW.

Загрузим в плату скетч, который включает и выключает светодиод L каждый раз при нажатии кнопки:

```
int ledPin = 13;
volatile boolean flash = FALSE; //объявление глобальной переменной
void setup(){
   pinMode(ledPin, OUTPUT);
   pinMode(2, INPUT_PULLUP);
   attachInterrupt(0, stuffHapenned, FALLING);
      /*настройка системы прерывания: 0 — номер прерывания;
        stuffHapenned — имя процедуры обработки прерывания;
        FALLING — запрос по переходу от HIGH к LOW. */
    }
void loop() {} //пустой бесконечный цикл — ожидание запроса прерывания
void stuffHapenned() {   //процедура обработки прерывания
 flash = ! flash;            //инверсия логического значения переменной
 if (flash) digitalWrite(ledPin, HIGH);  //выполнить, если flash==true
 else digitalWrite(ledPin, LOW);         //выполнить, если flash==false
}
```

Процедуры обработки прерываний должны быть короткими и быстрыми, обычно их выполняют функции ввода/вывода. На время обработки режим прерываний автоматически отключается. Если в это время возникнет другое прерывание, оно будет проигнорировано. Кроме того, пока выполняется процедура, код в функции loop простаивает.

Процедура обработки прерываний не может иметь параметров и не должна ничего возвращать. Для передачи информации между процедурой и остальной программой обычно используются глобальные переменные.

В этом скетче функция stuffHapenned использует глобальную переменную flash, чтобы определить текущее состояние светодиода.

Для передачи данных между подпрограммой обработки прерываний и остальной программой должны использоваться переменные, объявленные со спецификатором volatile, как в объявлении переменной flash. Такое объявление защитит переменную от изменений из-за вмешательства компилятора.

7.2. Прерывания от таймера

Запрос прерываний можно организовать не только от внешних устройств, но и от устройств, размещённых на кристалле микроконтроллера.

Например, набор команд из библиотеки TimerOne упрощает настройку прерываний от таймера.

Следующий скетч показывает, как с помощью TimerOne сформировать последовательность прямоугольных импульсов с частотой 1 кГц. Сигнал выводится через контакт 12, и его можно наблюдать на осциллографе.

```
#include < TimerOne.h>          //подключение библиотеки TimerOne
int outputPin = 12;              //именование выходного контакта
volatile int output = LOW;       //объявление глобальной переменной
void setup(){
  pinMode(outputPin, OUTPUT);
   Timer1.initialize(500); //установка периода срабатывания таймера 500
мкс
  Timer1.attachInterrupt(toggleOutput);
                                 //указание процедуры обработки прерыва-
ния.
}
void loop() {} //цикл ожидания запроса прерывания от таймера
void toggleOutput() { //процедура обработки прерывания
  digitalWrite(outputPin, output); //вывод текущего значения output
  output = ! output; //инвертирование значения output
}
```

8. ДИСТАНЦИОННОЕ УПРАВЛЕНИЕ УСТРОЙСТВАМИ

Дистанционное управление применяется при подключении конструктивно обособленного устройства управления к исполнительному устройству. Сигналы управления передаются по проводным или беспроводным линиям связи. Среди беспроводных линий можно указать:

- акустические линии, в которых сигналы являются ультразвуковыми импульсами;
- радиоволновые линии, передающие сигналы в виде радиоволн (Bluetooth, Wi-Fi);

– инфракрасные линии (IR) с сигналами в форме световых импульсов.

8.1. Инфракрасная линия связи

Если исполнительное устройство и устройство управления находятся в пределах прямой видимости, то проще всего использовать IR-линию связи. В качестве передающего устройства оператор может использовать инфракрасный пульт от телевизора или другой бытовой техники.

Передающее устройство формирует пакет данных для удалённого приёмника. Пакет содержит: стартовую последовательность; адрес приёмника (один байт); код команды (один байт), которую должно выполнить исполнительное устройство.

В пультах фирмы NEC пакеты формируются из пачек световых IR-импульсов с частотой 38 кГц (со скважностью 3) и длительностью 560 мкс (рис. 8.1). Начало бита обозначается пачкой

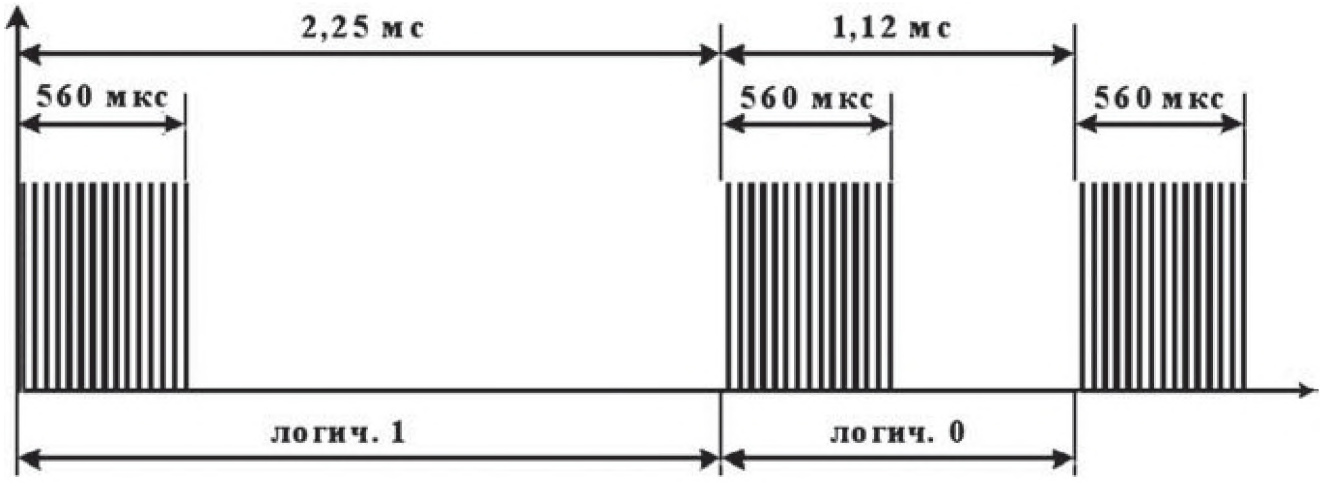

Рис. 8.1

IR-импульсов, а логическое значение бита кодируется длительностью паузы между пачками IR-импульсов. Последний бит в пакете завершается пачкой импульсов.

Стартовая последовательность содержит пачку импульсов длительностью 9 мс и паузу в 4,5 мс. Далее следуют: адрес приёмника, инвертированный адрес приёмника, код команды и инвертированный код команды. Байты передаются младшим битом

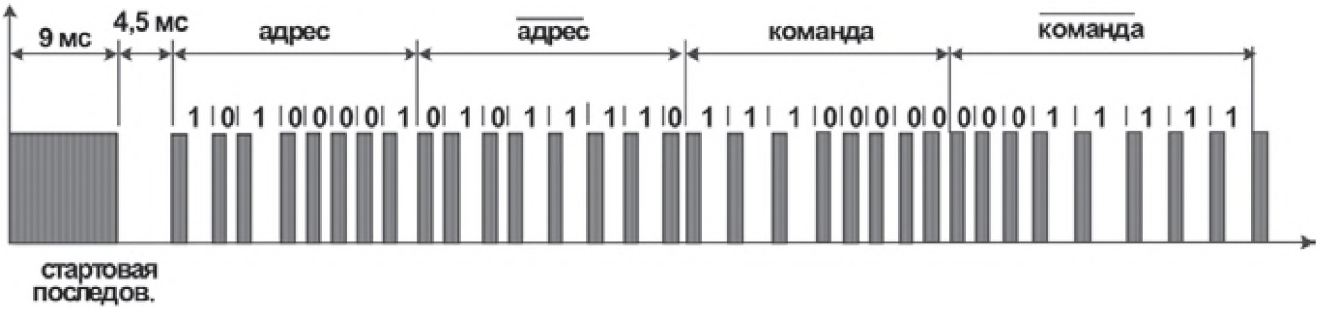

Рис. 8.2

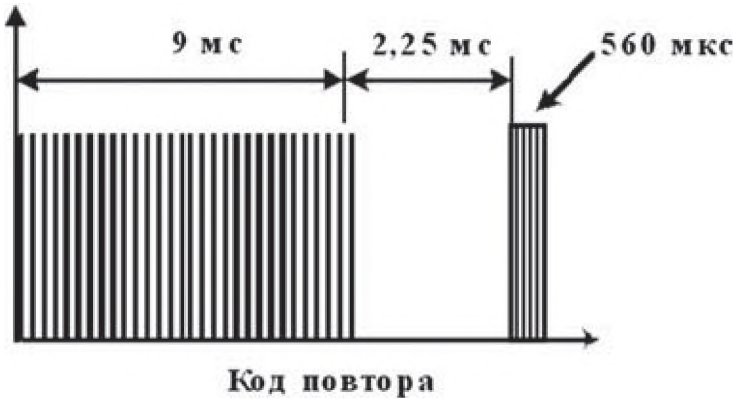

Рис. 8.3

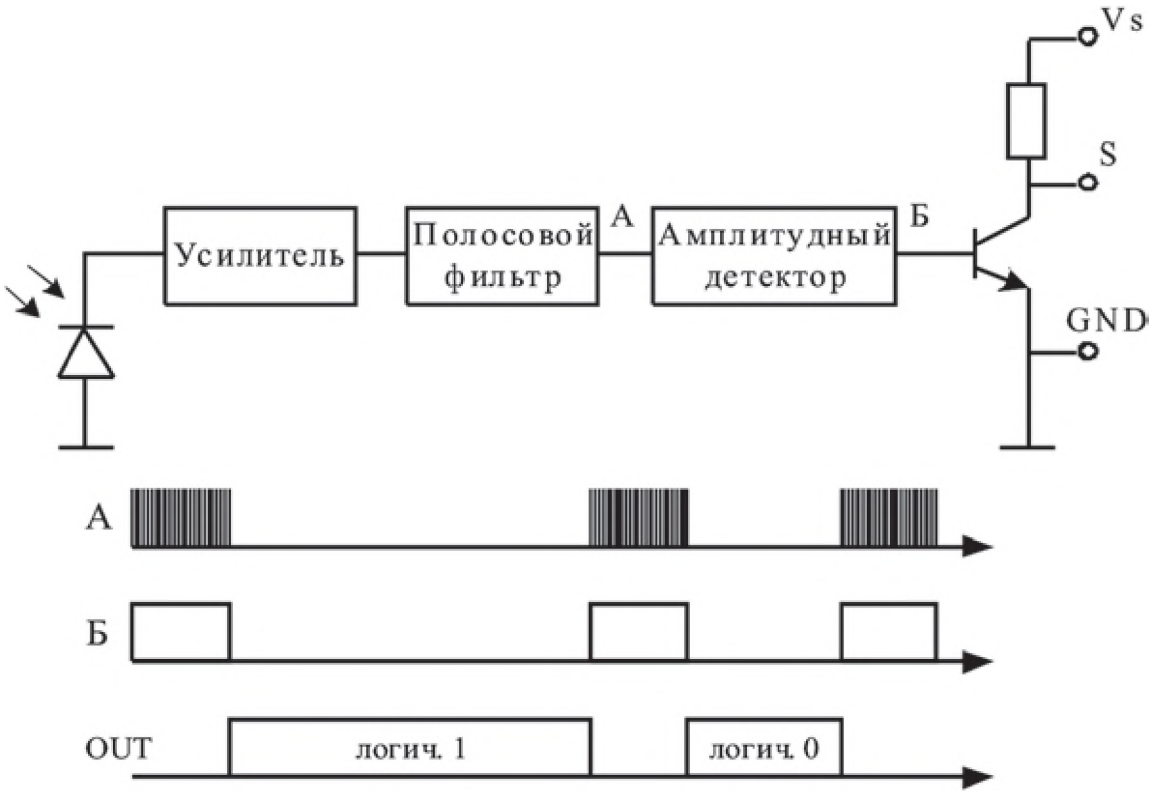

Рис. 8.4

вперёд. Всего в пакете стартовая последовательность и 4 байта данных (рис. 8.2). При удержании кнопки пульта каждые 110 мс передаётся код повтора (рис. 8.3).

Для приёма световых импульсов используется IR-приёмник (рис. 8.4). Входной фотодиод приёмника преобразует IR-импульсы в электрические импульсы. Для защиты от внешних помех импульсы проходят через полосовой фильтр с частотой пропускания 38 кГц. В амплитудном детекторе пачка импульсов преобразуется в один импульс длительностью 9 мс (стартовая последовательность) или 560 мкс (данные). Часто выход приёмника S имеет инвертирующий усилительный каскад, и пачки импульсов представляются нулевым логическим уровнем, а паузы — единичным.

Подключение IR-приёмника

IR-приёмник должен располагаться таким образом, чтобы световой сигнал передатчика (пульта управления рис. 8.5) не перекрывался другими предметами. Шилд приёмника (рис. 8.6) объединяет датчик излучения и дополнительные фильтры для уменьшения помех (см. рис. 8.4). Приём сигнала можно визуально контролировать по миганию индикатора. Для включения в систему плата имеет три вывода: S — выходной сигнал приёмника; VCC — напряжение питания 5 В; GND — “земляной” вывод. Сигнальный вывод S подключается к цифровому контакту платы Arduino, настроенному на ввод.

Рис. 8.5

Рис. 8.6

Для работы с IR-приёмником используется библиотека IRremote.h. Приведённая далее программа позволяет определить, какие коды

соответствуют кнопкам пульта управления. Код нажатой кнопки отражается на мониторе последовательного порта. Полученные коды кнопок можно использовать для управления актюаторами.

```
#include <IRremote.h>  //подключение библиотеки
IRrecv irrecv(12);        //объект IRrecv, подключён к контакту 12
decode_results results;  //создаём структуру результата приёма данных
void setup() {
    Serial.begin(9600); //настраиваем канал связи с ПК
    irrecv.enableIRIn(); //включаем IR-приёмник
}
void loop() {
   if(irrecv.decode(&results)) {   //если данные получены, передать их в ПК
     Serial.println(results.value, HEX);
     irrecv.resume();           //принять следующий пакет
  }
  delay(100);
}
```

8.2. Радиоволновая линия связи

Радиоволновые линии связи объединяют в систему различные мобильные функциональные устройства. Если расстояния между устройствами не превышают десятков метров, то радиоволновая сеть может быть организована по стандарту Bluetooth (*блютуз*). Стандарт изначально разрабатывался как беспроводная альтернатива кабельной системе RS-232. Bluetooth объединяет до восьми функционирующих устройств, одно из которых является ведущим.

Для организации радиоканала устройства с интерфейсом Bluetooth имеют радиопередатчик и радиоприёмник, работающие на частотах в диапазоне 2,402...2,480 ГГц. Частотный диапазон разбит на 79 каналов (F = 2402 + k (МГц), где k = 0...78). Полоса каждого канала 1 МГц. В каждый момент времени устройство работает на одном из каналов.

При обмене данными используется частотная модуляция (рис. 8.7). Логической единице соответствует положительная девиация частоты (+160 кГц), а логическому нулю — отрицательная девиация (−160 кГц).

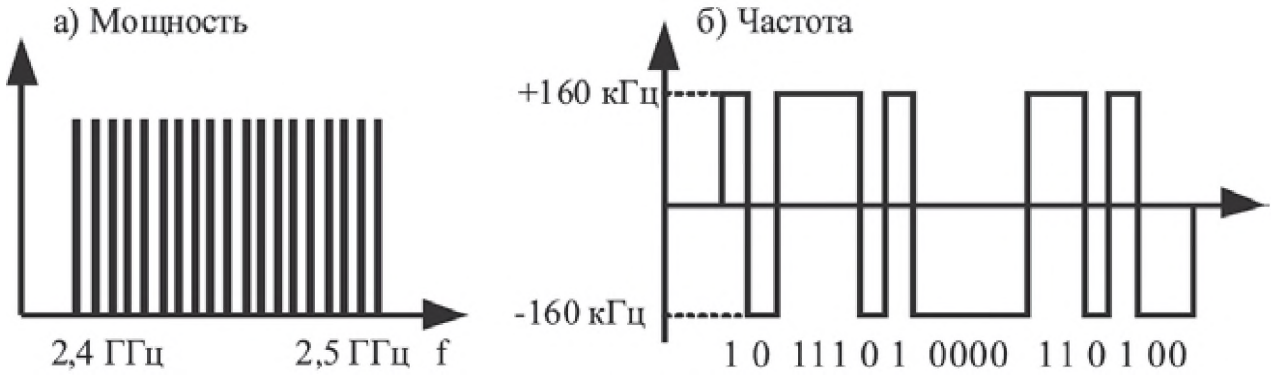

Рис. 8.7

Для защиты от помех передача ведётся с перескоком несущей частоты с одного канала на другой 1600 раз в секунду. Переключение каналов выполняется в соответствии с псевдослучайной последовательностью, формируемой по адресу ведущего устройства. Последовательность задаётся при установлении связи ведущего с ведомым. Все устройства в сети синхронизируются и используют общий порядок изменения несущей частоты.

Каждой частоте соответствует тайм-слот длительностью 625 мкс. Передачи ведутся пакетами, занимающими от 1 до 5 тайм-слотов. Пакет может иметь длину 0...2745 бит.

В плате Arduino в качестве Bluetooth устройства используется модуль HC-05 (рис. 8.8). В исходном состоянии модуль работает в режиме ведомого с выходом по интерфейсу UART (RS-232) со скоростью обмена 9600 бод. Модуль получает питание (VCC = 5 В и GND) от платы Arduino.

Принятые по радиоканалу данные передаются на контакт TXD модуля (рис. 8.8). Для приёма данных от модуля в плате Arduino

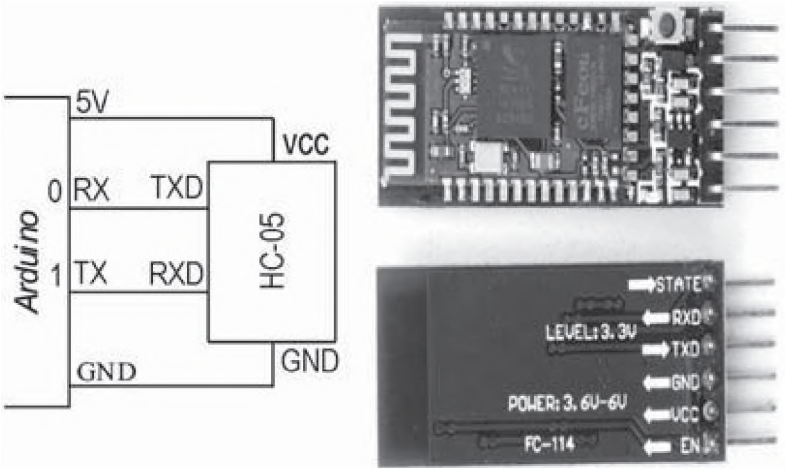

Рис. 8.8

используется контакт приёмника RX. Передаваемые из Arduino данные передаются на контакт TX платы. В модуль HC-05 они поступают через контакт RXD. Контакты STATE и EN можно оставить свободными.

При подаче напряжения питания на модуль он переходит в режим ожидания запроса на подключение к сети. При этом светодиод на модуле мигает с частотой 2 Гц.

В качестве ведущего устройства можно использовать смартфон. Для этого в него нужно загрузить программу обмена данными по Bluetooth из Google Play. Например, программа “Bluetooth Terminal HC-05”.

После запуска программы выбирается пункт “SCAN”. На экране отображаются имена доступных для подключения устройств. Выбираем устройство “HC-05”. При запросе пароля нужно ввести 1234.

После установки связи с Arduino в центральной части экрана смартфона отображается принятая информация. В нижней части экрана выделяется строка, в которую записываются символы для передачи в Arduino.

Для проверки передачи данных из Arduino в смартфон загрузим Скетч_1:

```
int cnt = 0;  // счетчик
void setup() { Serial.begin(9600); } // инициализация порта
void loop() {
   cnt++;
   Serial.print(“Hello from Arduino! Counter:”);    // выводим строку
   Serial.println(cnt);                            // выводим значение счетчика
   delay(1000);                                    // ждем 1 секунду
}
```

ВАЖНО: при загрузке скетча модуль Bluetooth нужно отключить, т.к. контакты TX и RX платы Arduino используются для связи с компьютером.

Существует библиотека для эмуляции последовательного порта “SoftwareSerial”, позволяющая формировать сигналы TX и RX на любых цифровых контактах платы. Используя библиотеку, можно организовать связь между компьютером и смартфоном через плату Arduino. При этом плата с компьютером соединена через встроенный интерфейс USART (цифровые контакты 0 и 1), а связь платы с модулем HC-05 осуществляется через программный последовательный порт.

Для проверки связи нужно подключить контакты TXD и RXD модуля к контактам RX и TX программного порта и загрузить скетч_2 управления светодиодом через смартфон:

```
#include <SoftwareSerial.h>
SoftwareSerial BT_Serial(12, 2); //(Rx, Tx) на плате Arduino подключить
                                 //к (TXD, RXD) модуля HC-05
char Char_Inp;
void setup() {
   Serial.begin(9600);          //порт для связи с компьютером
   BT_Serial.begin(9600);    //программный порт для связи с HC-05
   pinMode(13, OUTPUT);
   digitalWrite(13, 0);         //выключение светодиода
}
void loop() {
  if (BT_Serial.available()) {              //данные приняты в буфер
    Char_Inp=(char)BT_Serial.read();     //чтение данных из буфера
    Serial.print(Char_Inp);                //вывод данных в компьютер
    delay(100);
    BT_Serial.println(Char_Inp);   //обратная передача данных в смартфон
    switch (Char_Inp) {                    //управление светодиодом
      case '0':
        digitalWrite(13, LOW);  //выключение светодиода
        BT_Serial.println(" OFF");   //передача в смартфон
        break;
    case '1':
        digitalWrite(13, HIGH); //включение светодиода
        BT_Serial.println(" ON");   //передача в смартфон
        break;
}
  }
}
```

9. ЗАДАНИЯ ДЛЯ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

Практические занятия выполняются на лабораторном стенде с платой Arduino UNO, которая объединяет микроконтроллер Atmel ATmega328 с дополнительными модулями, обеспечивающими полноценную автономную работу платы в качестве встраиваемой вычислительной системы.

Скетчи разрабатываются в интегрированной среде Arduino IDE на персональном компьютере. Тексты скетчей пишутся на языке, подобном C++ в текстовом редакторе. При компиляции и загрузке скетчей в микроконтроллер в рабочее окно Arduino IDE выводятся диагностические сообщения о наличии или отсутствии ошибок.

Если ошибок не обнаружено, то загрузка скетча в микроконтроллер выполняется через интерфейс USB. После загрузки скетч сразу начинает выполняться.

ЗАДАНИЕ 1. **Скетч управления светодиодом L**

1. Подключить плату к компьютеру кабелем USB.
2. Запустить среду Arduino IDE на компьютере. Поверить правильность типа подключенной платы Arduino и порта связи платы с компьютером.
3. Изучить раздел 2.
4. Составить блок-схему скетча, в котором:
- задаётся имя ledPin светодиоду L;
- устанавливается частота мигания светодиода 8 Гц.
5. Выполнить скетч.
6. Изменить скетч таким образом, чтобы частота мигания задавалась компьютером через монитор последовательного порта.

ЗАДАНИЕ 2. **Управление частотой мигания светодиода L**

Составить блок-схему и написать скетч управления светодиодом L:

1. Светодиод должен мигнуть 10 раз с периодом 1 с.
2. Светодиод должен мигнуть 10 раз с периодом 2 с и далее выключиться.
3. В процессе работы скетча сообщение о состоянии светодиода нужно выводить через последовательный монитор с периодом 0,5 с.

ЗАДАНИЕ 3. **Контроль состояния электрической кнопки**

Изучить подраздел 5.1.

1. Подключить кнопку между цифровым контактом 7 и GND (см. рис 5.1). При сборке схемы использовать монтажную плату.
2. Написать скетч:
- при нажатой кнопке на контакт 7 подаётся уровень LOW;
- при отпущенной кнопке на контакт 7 через подтягивающий резистор подаётся уровень HIGH;

– состояние кнопки, в виде строки символов (“button ON”/ “button OFF”), выводится в компьютер через последовательный монитор;
– после вывода строки нужно выдержать паузу в 1с.

3. Для подключения каналя связи платы с компьютером используется библиотечная команда Serial.begin(9600).

4. Вывод строки символов выполняется командой Serial.println (“*символы*”).

ЗАДАНИЕ 4. **Изменение частоты мигания светодиода нажатой кнопкой**

Собрать схему рис. 5.1. Написать скетч, в котором:

1. Контакт 7 настроить на ввод с подтягивающим резистором.

2. При отпущенной кнопке частоту мигания светодиода L установить 1 Гц, а при нажатой кнопке — частоту 5 Гц.

Нарисовать схему подключения компонентов к плате. Процедуры включения и выключения светодиода нужно оформить как функцию с формальным параметром, задающим частоту мигания.

ЗАДАНИЕ 5. **Триггерный режим кнопки при управлении светодиодом**

Выполнить включение/выключение светодиода L по нажатию кнопки:

1. Изменение свечения светодиода происходит после нажатия кнопки.

2. Для подавления дребезга кнопки считывание её состояния выполняется повторно через интервал 0,2 с. Кнопка считается нажатой, если уровень LOW подтверждается.

ЗАДАНИЕ 6. **Управление яркостью свечения светодиода**

Изучить раздел 5.2. Написать скетч управления яркостью светодиода ШИМ-сигналами. ШИМ-сигналы формируются на контактах платы, помеченных символом “~”. Скважность имульсов задаётся командой analogWrite(val), где val={0...255}.

Нарисовать схему подключения компонентов к плате. Собрать схему на монтажной плате в соответствии с рис. 5.3.

В скетче функция loop() организует два последовательных цикла — увеличение val от нуля до максимума, и уменьшения от масимума до нуля. После каждой команды инкремента/декремента val установить паузу 10 мс.

ЗАДАНИЕ 7. **Управление яркостью светодиода через компьютер**

Рис. 9.1

Собрать схему рис. 5.3. В скетче величина скважности ШИМ-сигналов должна передаваться на плату через монитор последовательного порта (подраздел 2.2).

Доработать скетч так, чтобы наблюдать изменение скважности на контакте ~6 с помощью осциллографа (рис. 9.1). Нарисовать схему подключения компонентов к плате.

ЗАДАНИЕ 8. **Измеритель напряжения постоянного тока**

Предварительно нужно определить коэффициент преобразования АЦП (подраздел 5.3):

a. Подать входной аналоговый сигнал $U_{вх} = 3{,}3$ В на вход АЦП (см. рис 5.4);

b. Вывести в монитор результат преобразования N и вычислить (вручную) коэффициент преобразования: $k = U_{вх}/N$.

Изучить подраздел 5.4. Для подачи измеряемого напряжения на АЦП (контакт A0) собрать схему рис. 5.6,*б*. В скетче вольтметра нужно:

1. Активировать монитор последовательного порта;
2. Прочитать результат преобразования: int N=analogRead(A0);
3. Вычислить величину входного напряжения: float U=k*N;
4. Вывести результат измерения в монитор с частотой 1 Гц.

ЗАДАНИЕ 9. **Работа с датчиком препятствий**

Изучить подраздел 5.5. Схема и модуль датчика приведены на рис. 5.7. Вывод OUT подключается к цифровому контакту 7 платы. Инфракрасный двоичный датчик обнаруживает препятствие по отражению излучения инфракрасного светодиода. Отсутствие отражённого сигнала означает отсутствие препятствия на определённом расстоянии от датчика. При этом на выходе (контакт 7) формируется сигнал LOW. При появлении препятствия, от которого происходит отражение излучения, состояние выхода изменяется на HIGH.

Состояние датчика нужно отобразить на двух светодиодах: свечение красного диода означает, что препятствие обнаружено, а зелёного — путь свободен. Красный и зелёный светодиоды поклю-

чить к контактам 4 и 5 соответственно (рис. 9.2). Нарисовать схему подключения компонентов к плате. Опытным путём определить область срабатывания датчика.

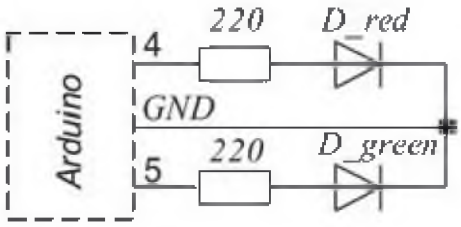

Рис. 9.2

Алгоритм скетча:

1. Присвоить имена контактам 7, 5 и 4.

2. Задать направление передачи контактов.

3. Выполнить считывание сигнала с датчика.

4. Если введено значение HIGH — включить зелёный светодиод.

5. Если введено значение LOW — включить красный светодиод.

6. Перейти к пункту 3.

ЗАДАНИЕ 10. Обнаружение трассирующей линии

Для обнаружения трассирующей линии нужно использовать два инфракрасных датчика отражения. Чёрная трассирующая линия не отражает света. Схема и модуль датчика приведены на рис. 5.7.

Датчик отражения работает в режиме компаратора. Цифровой выход D0 одного датчика подключен к контакту 0 платы, а другого — контакту 2. Приём отражённого сигнала переводит выходной контакт D0 датчика в состояние LOW. Когда трассирующая линия располагается под датчиком, отражения нет, и на D0 устанавливается уровень HIGH.

Состояние входных контактов 0 и 2 передаётся на выходные контакты 1 и 3.

К выходным контактам подключить светодиоды. Смещение трассирующей линии влево или вправо от среднего положения должно вызывать свечение соответствующего светодиода. Нарисовать схему подключения компонентов к плате. Алгоритм скетча:

1. Присвоить имена контактам 0, 1, 2, 3.

2. Настроить контакты 0 и 2 на ввод (чтение), а 1 и 3 — на вывод.

3. Выполнить считывание сигналов с датчиков и передать их на вывод.

4. Перейти к пункту 3.

ЗАДАНИЕ 11. **Инфракрасный датчик отражения как измеритель расстояния**

Схема и модуль датчика приведены на рис. 5.7. Датчик отражения работает в автономном режиме. Аналоговый выход датчика A0 подключен к контакту A0 платы. Уровень отражённого сигнала преобразуется в цифровой код, характеризующий расстояние до отражающей поверхности. Нарисовать схему подключения компонентов к плате. Алгоритм скетча:

1. Присвоить имя контакту A0.
2. Настроить контакт A0 на ввод (чтение).
3. Выполнить считывание сигнала с датчика.
4. Вывести уровень сигнала в монитор последовательного порта.
5. Перейти к пункту 3.

Изобразить зависимость цифрового кода от расстояния до поверхности.

ЗАДАНИЕ 12. **Управление ультразвуковым датчиком расстояния**

Изучить подраздел 5.6. Схема подключения датчика приведена на рис. 5.8. Нарисовать схему подключения компонентов к плате. Написать скетч измерения расстояния. Оценить диапазон измеряемого расстояния и зависимость результата измерения от угла между линией визирования и центральной осью датчика.

ЗАДАНИЕ 13. **Работа с электронным акселерометром**

Изучить подраздел 5.7. Схема подключения приведена на рис. 5.9.

Выполнить масштабирование принятых данных.

Откалибровать датчик и расположить его в пространстве для измерения угла наклона плоскости.

ЗАДАНИЕ 14. **Работа с электронным магнитометром**

Изучить подраздел 5.8. Схема подключения магнитометра приведена на рис. 5.11. Магнитометр связывается с платой Arduino по интерфейсу I^2C в режиме ведомого. Требуется:

- определить диапазон изменения данных по осям x, y, z;
- преобразовать полученные данные в угловые величины;
- расположить датчик в пространстве таким образом, чтобы использовать его в качестве компаса.

Алгоритм скетча:

1. Задать адрес магнитометра.

2. Подключить библиотеку Wire.

3. Инициализировать интерфейсы Serial и I^2C.

4. Установить связь с магнитометром.

5. Записать в регистр управления 0xB байт 0x01.

6. Записать в регистр управления 0x9 байт 0x11.

7. Установить связь с магнитометром.

8. Задать начальный регистр данных 0x00.

9. Запросить ввод данных из шести регистров.

10. Дождаться готовности данных в регистрах.

11. Ввести данные из регистров.

12. Закончить обмен с магнитометром.

13. Вывести данные в монитор.

14. Прейти к п.7.

***ЗАДАНИЕ 15*. Формирование случайной последовательности чисел**

Написать скетч формирования случайной последовательности целых чисел из дианазона {1...10}. В скетче нужно:

1. Описать массив целых чисел: `int mass[]={1,2,3,4,5,6,7,8,9,10}`.

2. Активировать монитор последовательного порта.

3. Сформировать случайный индекс элемента массива:
`int index=random(0,8)`.

4. Вывести в монитор случайную последовательность элементов массива `mass[index]` с частотой 2 Гц.

***ЗАДАНИЕ 16*. Генератор синусоидального сигнала**

Соберите схему рис. 9.3. К контакту 9 подключен ФНЧ для подавления несущей частоты ШИМ и осциллограф.

Алгоритм работы:

1. Сформировать массив значений синусов углов с шагом 2^0.

2. Вывести элементы массива в монитор и на контакт 9.

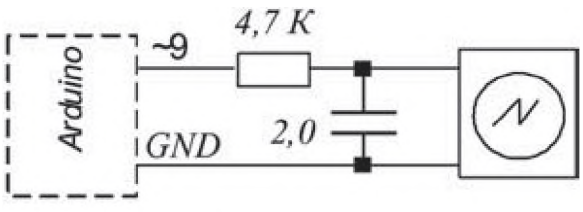

Рис. 9.3

Для улучшения сглаживания сигнала нужно подобрать период смены выходных значений, а для увеличения частоты синусоидального сигнала вывод в монитор нужно исключить.

***ЗАДАНИЕ 17*. Воспроизведение мелодии**

Изучить раздел 5.9 и собрать схему рис. 5.12. Скетч должен однократно воспроизводить заданную мелодию по нотам первой октавы:

Ми(1/4), Ми(1/4), Ми(1/2), Ми(1/4), Ми(1/4), Ми(1/2), Ми(1/4), Соль(1/4), До(1/4), Ре(1/4), Ми(1).

Длительность звучания целой ноты принять 1 с. Для формирования звука использовать команду tone().

Алгоритм скетча:

1. Сформировать массивы нот, длительностей и пауз.
2. В цикле вывести текущую ноту и выдержать паузу.
3. Организовать бесконечный цикл loop().

***ЗАДАНИЕ 18*. Управление двигателем постоянного тока**

Изучить подраздел 6.1. Написать скетч управления двигателем постоянного тока с реверсом направления вращения. Двигатель в лабораторном стенде управляется сигналами *In1* и *In2* на контактах 7 и 8 (см. рис. 6.4). Скорость вращения задаётся ШИМ-сигналом *EN* на контакте 10. Нарисовать схему подключения компонентов к плате.

Алгоритм работы:

1. Присвоить имена контактам 7, 8, 10.
2. Установить режим работы контактов “на вывод”.
3. Написать функцию задания скорости вращения ШИМ-сигналом на контакте 10.
4. Написать функцию включения “прямого” вращения двигателя установкой контактов 7 и 8 в состояние 1 (HIGH) и 0 (LOW) соответственно.
5. Написать функцию включения “обратного” вращения двигателя установкой контактов 7 и 8 в состояние 0 (LOW) и 1 (HIGH) соответственно.
6. Написать функцию остановки двигателя установкой контактов 7 и 8 в состояние 0 (LOW).
7. Задать начальную скорость равной нулю.
8. Включить прямое вращение на 3 с и остановить вращение на 0,5 с.
9. Включить обратное вращение на 3 с и остановить вращение на 0,5 с.
10. Перейти к п. 8.

ЗАДАНИЕ 19. **Управление двигателем через монитор порта**

Написать скетч управления скоростью и направлением вращения двигателя от компьютера. Через монитор последовательного порта на плату передаётся число. Знак числа указывает направление вращения, а величина (50...255) — скорость. Нарисовать схему подключения компонентов к плате.

Алгоритм работы:

1. Присвоить имена контактам 7, 8, 10.
2. Установить режим работы контактов “на вывод”.
3. Написать функцию задания скорости вращения ШИМ-сигналом на контакте 10.
4. Написать функцию включения “прямого” вращения двигателя установкой контактов 7 и 8 в состояние 1 (HIGH) и 0 (LOW) соответственно.
5. Написать функцию включения “обратного” вращения двигателя установкой контактов 7 и 8 в состояние 0 (LOW) и 1 (HIGH) соответственно.
6. Написать функцию остановки двигателя установкой контактов 7 и 8 в состояние 0 (LOW).
7. Задать начальную скорость равной нулю.
8. Если есть приём из монитора, ввести целое число, остановить вращение и выполнить паузу 0,1 с.
9. Настроить скорость вращения по модулю принятого числа.
10. Если принятое число положительное, включить прямое вращение.
11. Если принятое число отрицательное, включить обратное вращение.
12. Перейти к п. 8.

ЗАДАНИЕ 20. **Управление серводвигателем**

Изучить раздел 6.2. Серводвигатель должен непрерывно вращаться в пределах $\pm 45^{\circ}$ с шагом 2^0 и с выводом на монитор текущего положения ротора. Управляющий сигнал серводвигателя поступает с контакта 9 платы (рис. 9.4). Для управления серводвигателем нужно использовать библиотеку для работы с объектом класса Servo. Нарисовать схему подключения компонентов к плате. Алгоритм работы:

1. Подключить к скетчу библиотеку Servo: #include<Servo.h>.
2. Объявить объект с именем scanner: Servo scanner.

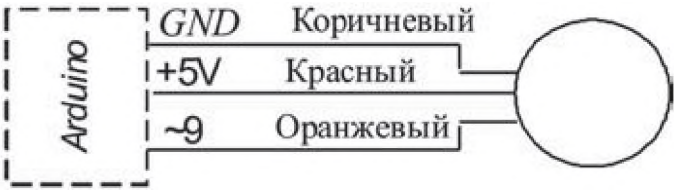

Рис. 9.4

3. Инициализировать объект с управлением через контакт 9: scanner.attach(9).

4. Активировать последовательный монитор.

5. Организовать цикл вращения ротора от угла -45° до $+45^{\circ}$, используя команду scanner.write(val), где int val={0...180} — угол поворота ротора.

6. Организовать цикл вращения ротора от угла $+45^{\circ}$ до -45°.

7. Вернуться к п. 5.

Примечание:

- после каждого обновления угла поворота вывести в монитор текущий угол из диапазона $\pm 45^{\circ}$;
- после вывода данных в scanner выполнить паузу 1 с для отработки поворота ротора и вывода данных на монитор.

Изменить скетч таким образом, чтобы угол поворота задавался потенциометром. Потенциометр подключить к контакту A0. Для изменения диапазона сигнала с АЦП используйте встроенную функцию map (см. п. 3.7.5).

ЗАДАНИЕ 21. Управление ультразвуковым локатором

Ультразвуковой датчик крепится на роторе серводвигателя. Схема подключения датчика приведена на рис. 5.8. Серводвигатель должен однократно просканировать сектор с углом $\pm 45^{\circ}$ с шагом 2^0 и с выводом на монитор текущего положения ротора. Серводвигатель подключается к плате через контакт 9 (см. рис. 9.4). Для управления серводвигателем используется библиотека для работы с объектом класса Servo. После сканирования плоскость датчика должна расположиться ортогонально кратчайшему расстоянию до препятствия. Нарисовать схему подключения компонентов к плате.

Для управления датчиком расстояния используется библиотека Ultrasonic.

Алгоритм работы:

1. Подключить к скетчу библиотеки Servo и Ultrasonic: #include < Servo.h> и #include <Ultrasonic.h>.

2. Объявить объекты Servo и Ultrasonic: Servo scanner; и Ultrasonic sonar(5,6).

3. Инициализировать Servo: scanner.attach(9).

4. Активировать монитор последовательного порта.

5. Организовать цикл вращения ротора от угла −45° до +45°, синхронно измеряя расстояние до препятствия. Для каждого положения ротора фиксировать пару чисел, определяющих угол поворота и расстояние.

6. После сканирования всего сектора выбрать минимальное расстояние и соответствующий угол поворота.

7. Развернуть датчик на выбранный угол.

8. Организовать бесконечный цикл ожидания сброса.

Угол поворота представляется числом типа int, а расстояние — double.

ЗАДАНИЕ 22. Организация прерываний

Изучить подраздел 7.1 и собрать схему рис. 9.5. RC-цепочка предназначена для подавления дребезга контактов кнопки. Написать скетч включения/выключения светодиода L по прерыванию при нажатии кнопки в соответствии с алгоритмом:

1. Присвоить имя ledL контакту 13.

2. Объявить глобальную логическую переменную flash с квалификатором volatile и присвоить ей начальное значение false. Значение переменной соответствует текущему состоянию светодиода: вкл/выкл → true/false.

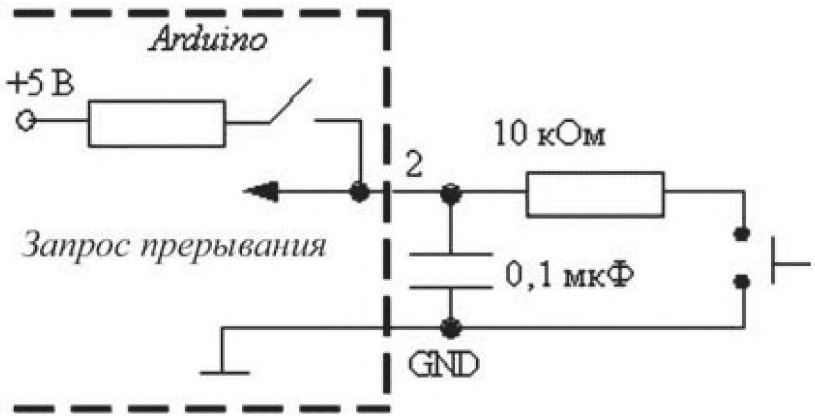

Рис. 9.5

3. Настроить контакт 13 на вывод.

4. Настроить контакт 2 на ввод с подтягивающим резистором (INPUT_PULLUP).

5. Задать режим обработки прерывания командой

```
attachInterrupt(0, interHepp, FALLING),
```

где 0 — номер прерывания (контакт 2); interHepp — имя процедуры обработки прерывания; FALLING — запрос по спадающему фронту сигнала на контакте 2.

6. Организовать бесконечный пустой цикл loop.

7. В процедуре interHepp:

- инвертировать переменную flash;
- если flash==true — включить светодиод, иначе — выключить светодиод.

ЗАДАНИЕ 23. Организация прерываний от таймера T1

Изучить подраздел 7.2. Скетч должен формировать последовательность прямоугольных импульсов с частотой 1 кГц с помощью прерываний от таймера T1. Импульсы выводятся через контакт 12, и их нужно подать на осциллограф, подлючённый между контактом 12 и GND.

Нарисовать схему подключения компонентов к плате.

Алгоритм работы:

1. Подключить библиотеку таймера: #include <TimerOne.h>.

2. Присвоить имя pwm контакту 12.

3. Объявить глобальную переменную output=LOW с квалификатором volatile. Значение output — текущее значение на контакте 12.

4. Настроить pwm на вывод.

5. Установить период срабатывания таймера 0,5 мс: Timer1.initialize(500).

6. Указать процедуру обработки прерывания:

```
Timer1.attachInterrupt(toggleOutput);
```

7. Организовать бесконечный цикл loop.

8. В процедуре обработки прерывания:

- вывести в pwm текущее выходное значение;
- инвертировать текущее выходное значение.

ЗАДАНИЕ 24. **Управление платой Arduino по инфракрасному каналу**

Изучить подраздел 8.1. Написать скетч для определения кодов нажатия трёх кнопок (1, 2, 3) IR-пульта. Написать скетч управления светодиодами:

1. При нажатии кнопки 1 включается красный светодиод;
2. При нажатии кнопки 3 включается зелёный светодиод;
3. При нажатии кнопки 2 светодиоды выключаются.

Оба светодиода одновременно светиться не должны. Нарисовать схему подключения компонентов к плате.

ЗАДАНИЕ 25. **Дистанционное управление двигателем постоянного тока**

Изучить подраздел 8.1. Написать скетч для определения кодов нажатия трёх кнопок (“<”, “>”, “ОК”) IR-пульта. Написать скетч управления двигателем постоянного тока:

1. При нажатии кнопки “<” двигатель вращается по часовой стрелке;
2. При нажатии кнопки “>” двигатель вращается против часовой стрелки;
3. При нажатии кнопки “ОК” двигатель выключаются.

Нарисовать схему подключения компонентов к плате.

ЗАДАНИЕ 26. **Управление платой Arduino по радиоканалу Bluetooth**

Изучить подраздел 8.2. Установить на смартфоне программу “Bluetooth Terminal HC-05" из Google Play. Проверить наличие связи между смартфоном и Arduino, используя Скетч_1 из подраздела 8.2.

Сформировать программный последовательный порт на контактах 2 (Tx) и 12 (Rx). Проверить наличие связи смартфона с компьютером, используя Скетч_2 из подраздела 8.2. Составить блок-схему Скетча_2.

Составить скетч включения/выключения в разных комбинациях двух светодиодов. Нарисовать схему подключения компонентов к плате.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Момот М.В.* Мобильные роботы на базе Arduino. — СПб.: БХВ-Петербург, 2017. — 288 с.

2. *Монк Саймон.* Программируем Arduino. Профессиональная работа со скетчами. — СПб.: Питер, 2017. — 252 с.

3. *Монк Саймон.* Программируем Arduino. Основы работы со скетчами. — СПб.: Питер, 2017. — 208 с.

4. *Евстифеев А.В.* Микроконтроллеры AVR семейства Mega. — М.: Додека, 2007. — 592 с.

5. *Шарапов В.М.* и др., Датчики: Справочное пособие. — М.: Техносфера, 2012. — 624 с.

6. *Бусурин В.И.* и др. Технические средства микропроцессорных устройств авиационной автоматики. — М.: Изд-во МАИ, 2018. — 136 с.

7. *Бусурин В.И.* и др. Функциональные устройства автоматики. — М.: Изд-во МАИ, 2017. — 84 с.

8. Atmel microcontroller ATmega328-328P, Datasheet. URL: http://mkprog.ru/wp-content/uploads/2017/09/ATmega328-328P_Datasheet.pdf

ОГЛАВЛЕНИЕ

Тем. план 2019, поз. 2

Можаев Виктор Александрович
Бусурин Владимир Игоревич
Шлеёнкин Лев Алексеевич

ПРОГРАММНО-АППАРАТНЫЕ СРЕДСТВА АВИАЦИОННОЙ АВТОМАТИКИ НА ОСНОВЕ МИКРОКОНТРОЛЛЕРОВ

Редактор *М.С. Винниченко*
Компьютерная верстка *Т.С. Евгеньевой*

Сдано в набор 30.09.2019. Подписано в печать 13.11.2019.
Бумага писчая. Формат 60×84 1/16. Печать офсетная.
Усл. печ. л. 4,42. Уч.-изд. л. 4,75. Тираж 150 экз.
Заказ 1067/697.

Издательство МАИ
(МАИ), Волоколамское ш., д. 4,
Москва, А-80, ГСП-3 125993

Типография Издательства МАИ
(МАИ), Волоколамское ш., д. 4,
Москва, А-80, ГСП-3 125993