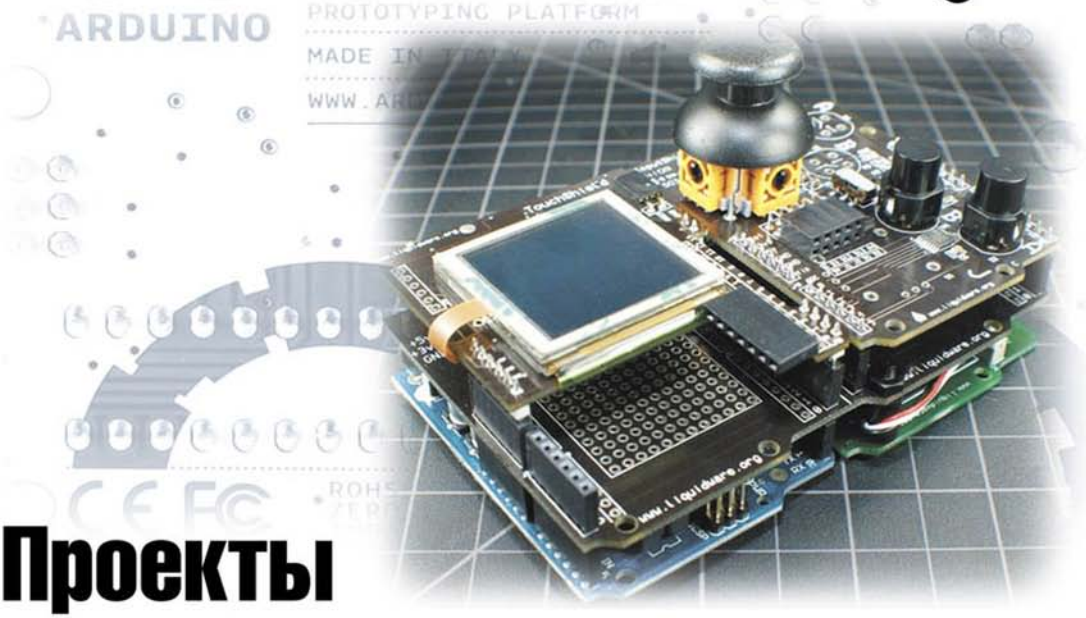
Электроника

Виктор Петин

bhv



Проекты с использованием контроллера Arduino

+ Материалы на www.bhv.ru

- исходные коды программ проектов из книги
- исходные коды библиотек
- описания и спецификация электронных компонентов
- схемы из книги в формате spl7

Arduino — проекты любой сложности легко и быстро!

Виктор Петин

Проекты с использованием контроллера Arduino

Санкт-Петербург
«БХВ-Петербург»
2014

УДК 004.4
ББК 32.973.26-018.2
П29

Петин В. А.

П29 Проекты с использованием контроллера Arduino. — СПб.: БХВ-Петербург, 2014. — 400 с.: ил. — (Электроника)

ISBN 978-5-9775-3337-9

Рассмотрены основные платы Arduino и платы расширения (шилды), добавляющие функциональность основной плате. Подробно описан язык и среда программирования Arduino IDE. Тщательно разобраны проекты с использованием контроллеров семейства Arduino. Это проекты в области робототехники, создания погодных метеостанций, "умного дома", вендинга, телевидения, Интернета, беспроводной связи (bluetooth, радиоуправление). Для всех проектов представлены схемы и исходный код. Также приведен исходный код для устройств Android, используемых в проектах для связи с контроллерами Arduino. На сайте издательства размещен архив с исходными кодами программ проектов из книги, исходными кодами библиотек, описаниями и спецификациями электронных компонентов, схемами из книги в формате spl7.

Для читателей, интересующихся современной электроникой

УДК 004.4
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зав. редакцией	*Екатерина Капалыгина*
Редактор	*Григорий Добин*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Марины Дамбиевой*

Подписано в печать 28.02.14.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 32,25.
Тираж 1700 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

ISBN 978-5-9775-3337-9

Оглавление

Предисловие

Для кого и о чем эта книга?

Предлагаемая книга ориентирована на читателей, желающих быстро войти в темы программирования микроконтроллеров и использования микроконтроллеров для связи с внешними системами в проектах автоматизации и робототехники.

Книга содержит описание языка программирования плат Arduino в среде Arduino IDE и предлагает изучение предмета на реальных проектах, имеющих практическое значение. В ней вы найдете множество примеров и проектов использования Arduino, представляющих собой законченные решения, пригодные для использования в ваших проектах.

Книга сопровождается электронным архивом, содержащим исходный код всех рассмотренных примеров и проектов, а также используемые в проектах необходимые библиотеки *(см. приложение 2)*. Этот электронный архив можно скачать с FTP-сервера издательства "БХВ-Петербург" по ссылке **ftp://ftp.bhv.ru/9785977533379.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Структура книги

Книга состоит из трех частей и включает предисловие, тридцать глав и два приложения.

Часть I содержит описание Arduino, обзор контроллеров семейства Arduino и плат расширения для Arduino.

В *части II* книги рассмотрены среда разработки и язык программирования для контроллеров Arduino.

Часть III посвящена созданию конкретных устройств на основе контроллера Arduino. Проекты содержат электрические схемы и листинги программ. Рассмотрено использование плат расширения (шилдов). В книге широко используются библиотеки Arduino.

В *приложениях* приведены перечень использованной литературы и интернет-ресурсов и описание электронного архива, сопровождающего книгу.

Благодарности

Хочу поблагодарить родных и близких, которые с пониманием относились к потраченному на книгу (за счет общения с ними) времени.

Большая благодарность издательству "БХВ-Петербург", где поверили в необходимость этой книги, и всем сотрудникам издательства, которые помогали мне в ее создании.

Благодарю также всех читателей, купивших эту книгу, — надеюсь, она поможет им в разработке собственных проектов на основе Arduino.

ЧАСТЬ I

Arduino — общий обзор

ГЛАВА 1

Введение в Arduino

1.1. Arduino — что это?

Появление первых микроконтроллеров ознаменовало начало новой эры в развитии микропроцессорной техники. Наличие в одном корпусе большинства системных устройств сделало микроконтроллер подобным обычному компьютеру. В отечественной литературе они даже назывались однокристальными микроЭВМ. Соответственно и желание использовать микроконтроллеры как обычные компьютеры появилось практически с их появлением. Но желание это сдерживалось многими факторами. Например, чтобы собрать устройство на микроконтроллере, необходимо знать основы схемотехники, устройство и работу конкретного процессора, уметь программировать на ассемблере и изготавливать электронную технику. Потребуются также программаторы, отладчики и другие вспомогательные устройства. В итоге без огромного объема знаний и дорогостоящего оборудования не обойтись. Такая ситуация долго не позволяла многим любителям использовать микроконтроллеры в своих проектах. Сейчас, с появлением устройств, дающих возможность работать с микроконтроллерами без наличия серьезной материальной базы и знания многих предметов, все изменилось. Примером такого устройства может служить проект Arduino итальянских разработчиков.

Arduino и его клоны представляют собой наборы, состоящие из готового электронного блока и программного обеспечения. Электронный блок здесь — это печатная плата с установленным микроконтроллером и минимумом элементов, необходимых для его работы. Фактически электронный блок Arduino является аналогом материнской платы современного компьютера. На нем имеются разъемы для подключения внешних устройств, а также разъем для связи с компьютером, по которому и осуществляется программирование микроконтроллера. Особенности используемых микроконтроллеров ATmega фирмы Atmel позволяют производить программирование без применения специальных программаторов. Все, что нужно для создания нового электронного устройства, — это плата Arduino, кабель связи и компьютер. Второй частью проекта Arduino является программное обеспечение для создания управляющих программ. Оно объединило в себе простейшую среду разработки и язык программирования, представляющий собой вариант языка C/C++ для микро-

контроллеров. В него добавлены элементы, позволяющие создавать программы без изучения аппаратной части. Так что для работы с Arduino практически достаточно знания только основ программирования на C/C++. Создано для Arduino и множество библиотек, содержащих код, работающий с различными устройствами.

1.2. В чем преимущество Arduino?

Пользователь современного компьютера не задумывается о функционировании отдельных частей ПК. Он просто запускает нужные программы и работает с ними. Точно так же и Arduino позволяет пользователю сосредоточиться на разработке проектов, а не на изучении устройства и принципов функционирования отдельных элементов. Нет надобности и в создании законченных плат и модулей. Разработчик может использовать готовые платы расширения или просто напрямую подключить к Arduino необходимые элементы. Все остальные усилия будут направлены на разработку и отладку управляющей программы на языке высокого уровня. В итоге доступ к разработке микропроцессорных устройств получили не только профессионалы, но и просто любители что-то сделать своими руками. Наличие готовых модулей и библиотек программ позволяет непрофессионалам в электронике создавать готовые работающие устройства для решения своих задач. А варианты использования Arduino ограничены только возможностями микроконтроллера и имеющегося варианта платы, ну и, конечно, фантазией разработчика.

1.3. История создания Arduino

В 2002 году программист Массимо Банци (Massimo Banzi) был принят на работу в должности доцента в Институт проектирования взаимодействий города Ивреа (Interaction Design Institute Ivrea, IDII) для продвижения новых способов разработки интерактивных проектов. Однако крошечный бюджет и ограниченное время доступа к лабораторной базе сводили его усилия практически на нет. В проектах Банци использовал устройство BASIC Stamp, разработанное калифорнийской компанией Parallax. Stamp представлял собой небольшую печатную плату с размещенными на ней источником питания, микроконтроллером, памятью и портами ввода/вывода для соединения с различной аппаратурой. Программирование микроконтроллера осуществлялось на языке BASIC. BASIC Stamp имел две проблемы: недостаток вычислительной мощности и достаточно высокую цену — плата с основными компонентами стоила около 100 долларов. И команда Банци решила самостоятельно создать плату, которая удовлетворяла бы всем их потребностям.

Банци и его сотрудники поставили себе целью создать устройство, представляющее собой простую, открытую и легкодоступную платформу для разработки, с ценой — не более 30 долларов — приемлемой для студенческого кармана. Хотели они и выделить чем-то свое устройство на фоне прочих. Поэтому в противовес другим производителям, экономящим на количестве выводов печатной платы, они решили добавить их как можно больше, а также сделали свою плату синей, в отличие от обычных зеленых плат.

Продукт, который создала команда, состоял из дешевых и доступных компонентов — например, базировался он на микроконтроллере ATmega328. Но главная задача состояла в том, чтобы гарантировать работу устройства по принципу plug-and-play, — чтобы пользователь, достав плату из коробки и подключив к компьютеру, мог немедленно приступить к работе.

Первый прототип платы был сделан в 2005 году, она имела простейший дизайн и еще не называлась Arduino. Чуть позже Массимо Банци придумал назвать ее так — по имени принадлежащего ему бара, расположенного в городе Ивреа. Бренд "Arduino" без какой-либо рекламы и привлечения средств маркетинга быстро приобрел высокую популярность в Интернете. С начала распространения продано более 250 тыс. комплектов Arduino, и это не учитывая множества клонов. В мире насчитывается более двухсот дистрибьюторов продукции Arduino — от крупных фирм, таких как SparkFun Electronics, до мелких компаний, работающих на местный рынок. На сегодня платформа Arduino представлена не одной платой, а целым их семейством. В дополнение к оригинальному проекту, называемому Arduino Uno, новые модели, имеющие на плате более мощные средства, носят название Arduino Mega, компактные модели — Arduino Nano, платы в водонепроницаемом корпусе — LilyPad Arduino, а новая плата с 32-разрядным процессором Cortex-M3 ARM — Arduino Due.

Своим успехом проект Arduino обязан существовавшим до него языку Processing и платформе Wiring. От этих проектов Arduino унаследовал одну сильную черту — удобную для пользователя среду разработки. До появления Arduino программирование микроконтроллеров требовало сложного и рутинного предварительного обучения. А с Arduino даже те, кто не имеет опыта работы с электронными устройствами, теперь могут приобщиться к миру электроники. Начинающим уже не приходится тратить много времени на изучение сопутствующего материала — они могут быстро разработать прототип, который будет полноценно рабочим.

По словам Массимо Банци, пятьдесят лет назад, чтобы написать программное обеспечение требовалась команда людей в белых халатах, которые знали все об электронных лампах. Теперь же, с появлением Arduino, множество людей получили возможность создавать электронные устройства самостоятельно. Как утверждает один из членов команды Банци, инженер по телекоммуникациям Дэвид Куартилльз, философия Arduino как раз и состоит в том, что желающий разобраться в электронике может сразу же приступить к ее изучению, вместо того чтобы сначала учиться алгебре.

ГЛАВА 2

Обзор контроллеров семейства Arduino

Основные версии плат Arduino представлены следующими моделями:

- **Due** — плата на базе 32-битного ARM микропроцессора Cortex-M3 ARM SAM3U4E;
- **Leonardo** — плата на микроконтроллере ATmega32U4;
- **Uno** — самая популярная версия базовой платформы Arduino;
- **Duemilanove** — плата на микроконтроллере ATmega168 или ATmega328;
- **Diecimila** — версия базовой платформы Arduino USB;
- **Nano** — компактная платформа, используемая как макет. Nano подключается к компьютеру при помощи кабеля USB Mini-B;
- **Mega ADK** — версия платы Mega 2560 с поддержкой интерфейса USB-host для связи с телефонами на Android и другими устройствами с интерфейсом USB;
- **Mega2560** — плата на базе микроконтроллера ATmega2560 с использованием чипа ATMega8U2 для последовательного соединения по USB-порту;
- **Mega** — версия серии Mega на базе микроконтроллера ATmega1280;
- **Arduino BT** — платформа с модулем Bluetooth для беспроводной связи и программирования;
- **LilyPad** — платформа, разработанная для переноски, может зашиваться в ткань;
- **Fio** — платформа разработана для беспроводных применений. Fio содержит разъем для радио XBee, разъем для батареи LiPo и встроенную схему подзарядки;
- **Mini** — самая маленькая платформа Arduino;
- **Pro** — платформа, разработанная для опытных пользователей, может являться частью большего проекта;
- **Pro Mini** — как и платформа Pro, разработана для опытных пользователей, которым требуется низкая цена, меньшие размеры и дополнительная функциональность.

Рассмотрим более подробно некоторые из этих плат.

2.1. Arduino Pro Mini

Arduino Pro Mini (рис. 2.1) построена на микроконтроллере ATmega168.

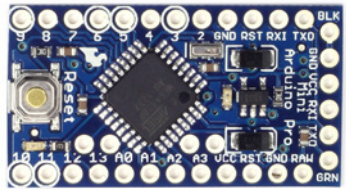


Рис. 2.1. Плата Arduino Pro Mini

Характеристики платы Arduino Pro Mini представлены в табл. 2.1.

Таблица 2.1. *Характеристики платы Arduino Pro Mini*

Микроконтроллер	ATmega168
Рабочее напряжение	3,3 В или 5 В (в зависимости от модели)
Входное напряжение	3,35–12 В (модель 3,3 В) или 5–12 В (модель 5 В)
Цифровые входы/выходы	14 (6 из которых могут использоваться как выходы ШИМ)
Аналоговые входы	6
Постоянный ток через вход/выход	40 мА
Флеш-память	16 Кбайт (2 — используются для загрузчика)
ОЗУ	1 Кбайт
EEPROM	512 байтов
Тактовая частота	8 МГц (модель 3,3 В) или 16 МГц (модель 5 В)

Arduino Pro Mini может получать питание: через кабель FTDI, или от платы-конвертера, или от регулируемого источника питания 3,3 В или 5 В (зависит от модели платформы) через вывод VCC, или от нерегулируемого источника через вывод RAW.

Выводы питания:

- RAW — для подключения нерегулируемого напряжения;
- VCC — для подключения регулируемых 3,3 В или 5 В;
- GND — выводы заземления.

2.2. Arduino Duemilanove

Arduino Duemilanove (рис. 2.2) построена на одном из микроконтроллеров: ATmega168 или ATmega328.

Характеристики платы Arduino Duemilanove представлены в табл. 2.2.

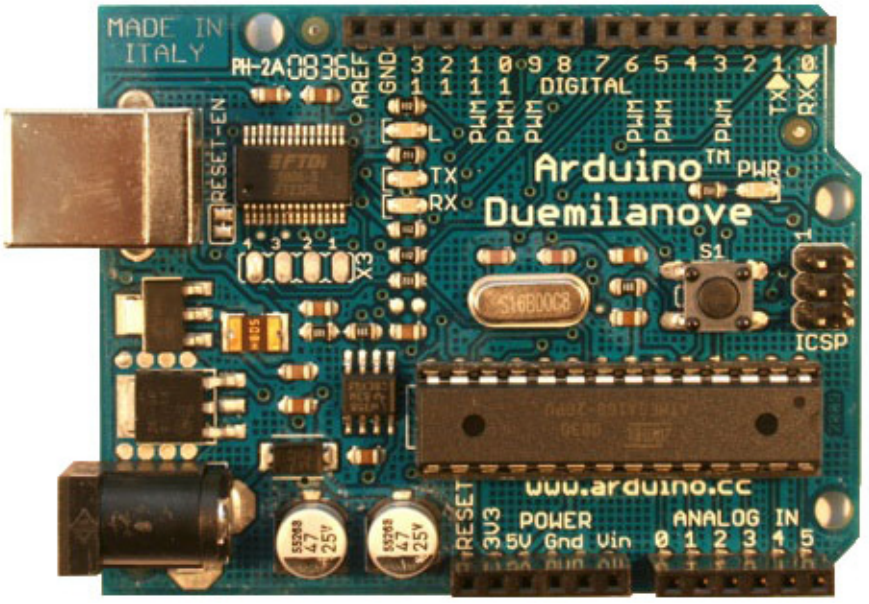


Рис. 2.2. Плата Arduino Duemilanove

Таблица 2.2. *Характеристики платы Arduino Duemilanove*

Микроконтроллер	ATmega168 или ATmega328
Рабочее напряжение	5 В
Входное напряжение (рекомендуемое)	7–12 В
Входное напряжение (предельное)	6–20 В
Цифровые входы/выходы	14 (6 из которых могут использоваться как выходы ШИМ)
Аналоговые входы	6
Постоянный ток через вход/выход	40 мА
Постоянный ток для вывода 3,3 В	50 мА
Флеш-память	16 Кбайт (ATmega168) или 32 Кбайт (ATmega328), при этом 2 Кбайт используются для загрузчика
ОЗУ	1 Кбайт (ATmega168) или 2 Кбайт (ATmega328)
EEPROM	512 байтов (ATmega168) или 1 Кбайт (ATmega328)
Тактовая частота	16 МГц

2.3. Arduino Nano

Платформа Nano (рис. 2.3), построенная на микроконтроллере ATmega328 (Arduino Nano 3.0) или ATmega168 (Arduino Nano 2.*x*), имеет небольшие размеры и может использоваться в лабораторных работах.

Рис. 2.3. Плата Arduino Nano

Arduino Nano может получать питание через подключение USB Mini-B, или от нерегулируемого 6–20 В (вывод 30) или регулируемого 5 В (вывод 27), внешнего источника питания. Автоматически выбирается источник с самым высоким напряжением.

Характеристики платы Arduino Nano представлены в табл. 2.3.

Таблица 2.3. *Характеристики платы Arduino Nano*

Микроконтроллер	ATmega168 или ATmega328
Рабочее напряжение	5 В
Входное напряжение (рекомендуемое)	7–12 В
Входное напряжение (предельное)	6–20 В
Цифровые входы/выходы	14 (6 из которых могут использоваться как выходы ШИМ)
Аналоговые входы	6
Постоянный ток через вход/выход	40 мА
Постоянный ток для вывода 3,3 В	50 мА
Флеш-память	16 Кбайт (ATmega168) или 32 Кбайт (ATmega328), при этом 2 Кбайт используются для загрузчика
ОЗУ	1 Кбайт (ATmega168) или 2 Кбайт (ATmega328)

Таблица 2.3 *(окончание)*

EEPROM	512 байтов (ATmega168) или 1 Кбайт (ATmega328)
Тактовая частота	16 МГц

2.4. Arduino LilyPad

Платформа Arduino LilyPad (рис. 2.4) разработана с целью использования в качестве части одежды. Она может быть зашита в ткань со встроенными источниками питания, датчиками и приводами с проводкой. Платформа построена на микроконтроллере ATmega168V.

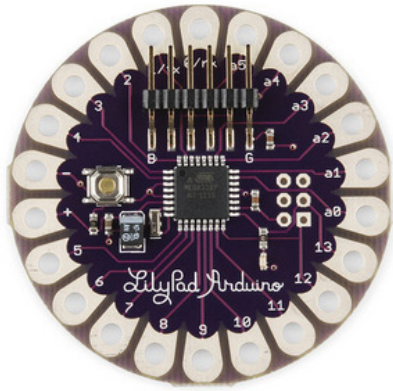


Рис. 2.4. Плата Arduino LilyPad

Характеристики платы Arduino LilyPad представлены в табл. 2.4.

Таблица 2.4. *Характеристики платы Arduino LilyPad*

Микроконтроллер	ATmega168 или ATmega328
Рабочее напряжение	2,7–5,5 В
Входное напряжение	2,7–5,5 В
Цифровые входы/выходы	14 (6 из которых могут использоваться как выходы ШИМ)
Аналоговые входы	6
Постоянный ток через вход/выход	40 мА
Флеш-память	16 Кбайт (ATmega168) или 32 Кбайт (ATmega328), при этом 2 Кбайт используются для загрузчика
ОЗУ	1 Кбайт (ATmega168) или 2 Кбайт (ATmega328)
EEPROM	512 байтов (ATmega168) или 1 Кбайт (ATmega328)
Тактовая частота	16 МГц

2.5. Arduino Uno

Контроллер Arduino Uno (рис. 2.5) построен на микроконтроллере ATmega328. В отличие от всех предыдущих плат, использовавших для связи по USB микроконтроллер FTDI USB, новый Arduino Uno использует микроконтроллер ATmega8U2.

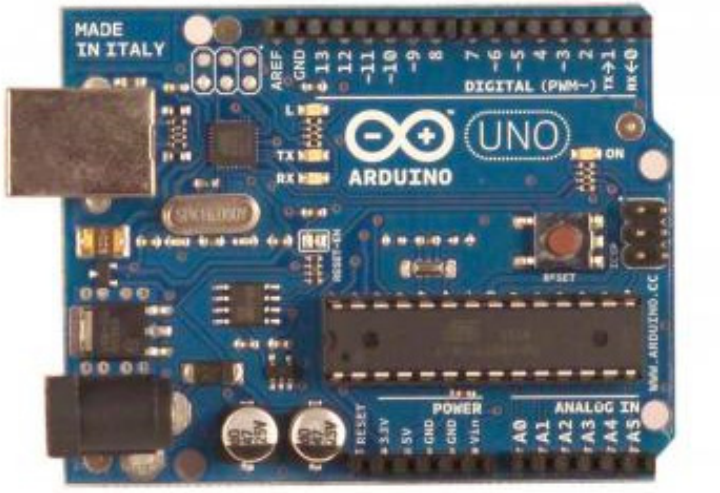


Рис. 2.5. Плата Arduino Uno

Характеристики платы Arduino Uno представлены в табл. 2.5.

Таблица 2.5. *Характеристики платы Arduino Uno*

Микроконтроллер	ATmega328
Рабочее напряжение	5 В
Входное напряжение (рекомендуемое)	7–12 В
Входное напряжение (предельное)	6–20 В
Цифровые входы/выходы	14 (6 из которых могут использоваться как выходы ШИМ)
Аналоговые входы	6
Постоянный ток через вход/выход	40 мА
Постоянный ток для вывода 3,3 В	50 мА
Флеш-память	32 Кбайт, при этом 0,5 Кбайт используются для загрузчика
ОЗУ	2 Кбайт
EEPROM	1 Кбайт
Тактовая частота	16 МГц

2.6. Arduino Mega2560

Arduino Mega (рис. 2.6) построена на микроконтроллере ATmega2560.

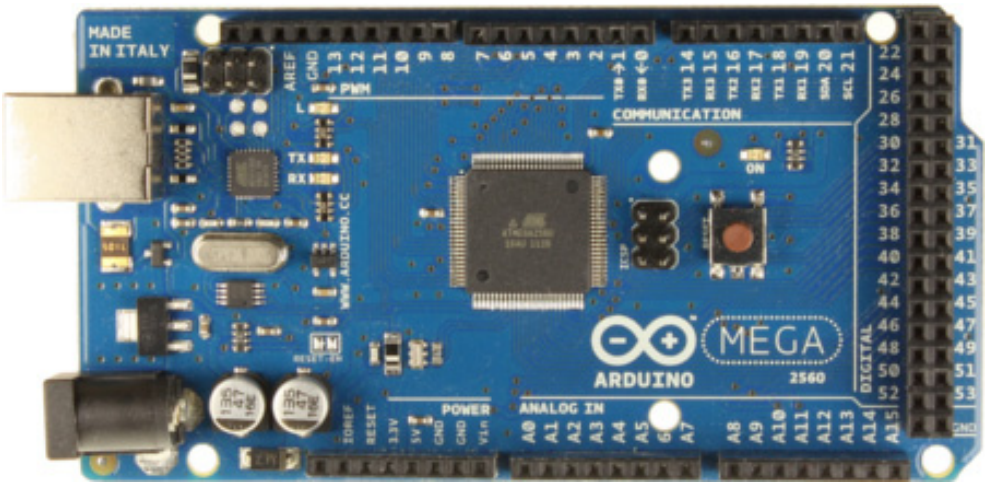


Рис. 2.6. Плата Arduino Mega2560

Характеристики платы Arduino Mega2560 представлены в табл. 2.6.

Таблица 2.6. *Характеристики платы Arduino Mega2560*

Микроконтроллер	ATmega2560
Рабочее напряжение	5 В
Входное напряжение (рекомендуемое)	7–12 В
Входное напряжение (предельное)	6–20 В
Цифровые входы/выходы	54 (14 из которых могут работат так же, как выходы ШИМ)
Аналоговые входы	16
Постоянный ток через вход/выход	40 мА
Постоянный ток для вывода 3,3 В	50 мА
Флеш-память	256 Кбайт, из которых 8 Кбайт используются для загрузчика
ОЗУ	8 Кбайт
EEPROM	4 Кбайт
Тактовая частота	16 МГц

2.7. Arduino Leonardo

Arduino Leonardo (рис. 2.7) — контроллер на базе микроконтроллера ATmega32U4. В отличие от всех предыдущих плат ATmega32U4 имеет встроенную поддержку для USB-соединения.

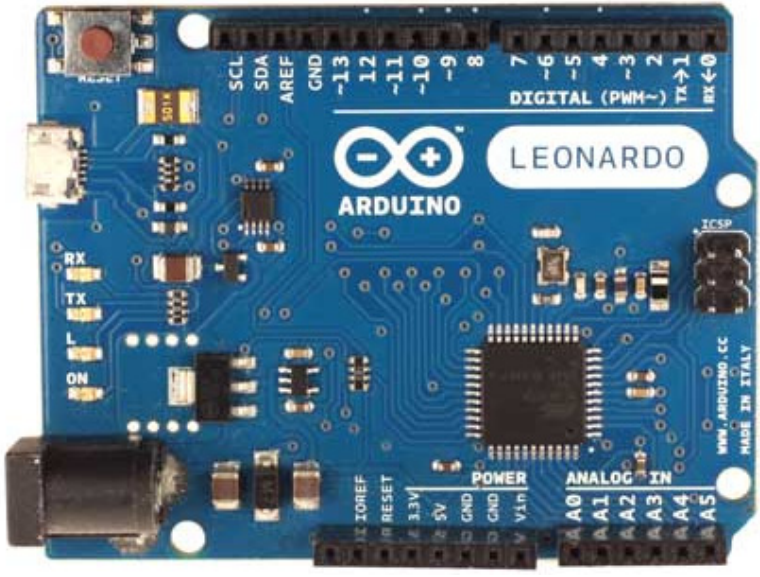


Рис. 2.7. Плата Arduino Leonardo

Характеристики платы Arduino Leonardo представлены в табл. 2.7.

Таблица 2.7. *Характеристики платы Arduino Leonardo*

Микроконтроллер	ATmega32U4
Рабочее напряжение	5 В
Входное напряжение (рекомендуемое)	7–12 В
Входное напряжение (предельное)	6–20 В
Цифровые входы/выходы	20 (7 из которых могут использоваться как выходы ШИМ)
Аналоговые входы	12
Постоянный ток через вход/выход	40 мА
Постоянный ток для вывода 3,3 В	50 мА
Флеш-память	32 Кбайт, из которых 4 Кбайт используются для загрузчика
ОЗУ	2 Кбайт
EEPROM	1 Кбайт
Тактовая частота	16 МГц

2.8. Arduino Due

Arduino Due (рис. 2.8) — плата микроконтроллера на базе процессора Atmel SAM3X8E ARM Cortex-M3. Это первая плата Arduino на основе 32-битного микроконтроллера с ARM-ядром.

В отличие от других плат Arduino, Arduino Due работает от 3,3 В. Максимальное напряжение, которое выдерживают входы/выходы, составляет 3,3 В.

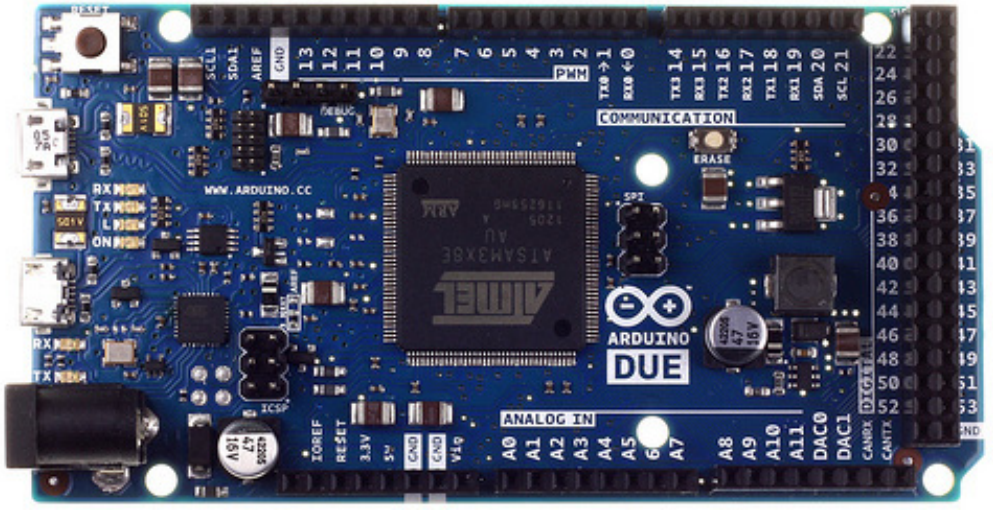


Рис. 2.8. Плата Arduino Due

Характеристики платы Arduino Due представлены в табл. 2.8.

Таблица 2.8. *Характеристики платы Arduino Due*

Микроконтроллер	AT91SAM3X8E
Рабочее напряжение	3,3 В
Входное напряжение (рекомендуемое)	7–12 В
Входное напряжение (предельное)	6–20 В
Цифровые входы/выходы	54 (на 12 из которых реализуется выход ШИМ)
Аналоговые входы	12
Аналоговые выходы	2 (ЦАП)
Постоянный ток через вход/выход	50 мА
Постоянный ток для вывода 3,3 В	800 мА
Постоянный ток для вывода 5 В	800 мА
Флеш-память	512 Кбайт
ОЗУ	96 Кбайт (два банка: 64 Кбайт и 32 Кбайт)
Тактовая частота	84 МГц

ГЛАВА 3

Платы расширения Arduino

Большую популярность плата Arduino приобрела не только из-за низкой стоимости, легкости разработки и программирования, но, главным образом, благодаря наличию плат расширения (так называемых *шилдов*), добавляющих Arduino дополнительную функциональность. Шилды (кроме маленьких модулей и платы LilyPad) подключаются к Arduino с помощью имеющихся на них штыревых разъемов (рис. 3.1).

Существует множество различных по функциональности шилдов — от простейших, предназначенных для макетирования, до сложных, представляющих собой отдельные многофункциональные устройства.

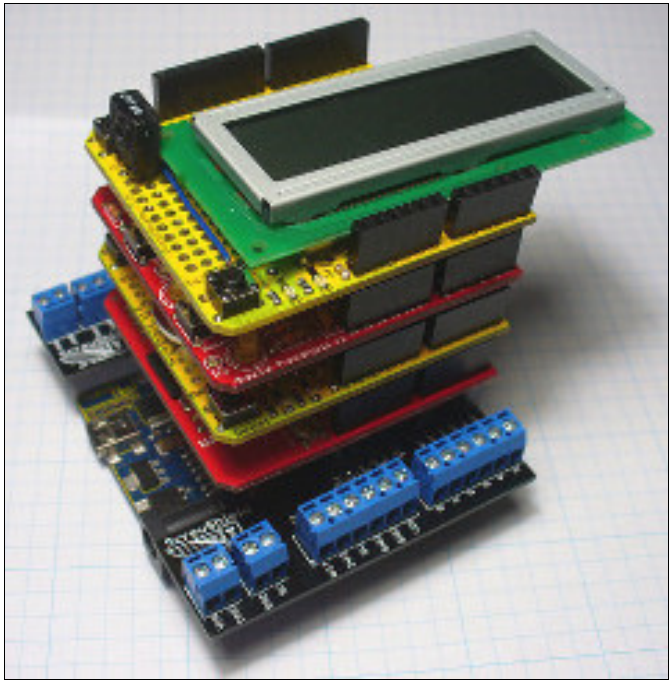


Рис. 3.1. Модульная структура установки плат расширения для Arduino

Рассмотрим примеры некоторых шилдов:

- **Ethernet Shield** (рис. 3.2) — обеспечивает подключение к Интернету;
- **XBee Shield** (рис. 3.3) — обеспечивает при помощи модуля Maxstream Xbee Zigbee беспроводную связь нескольким устройствам Arduino;

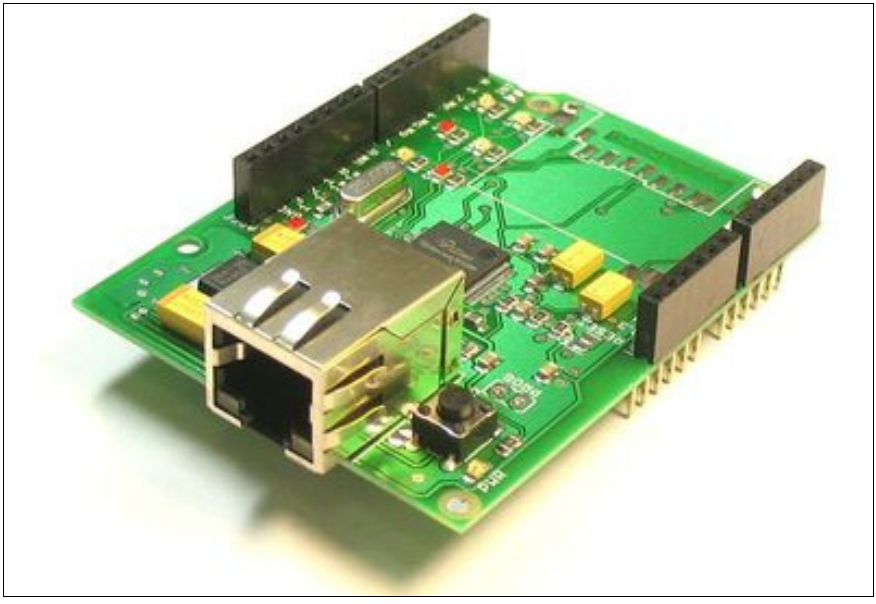


Рис. 3.2. Ethernet Shield

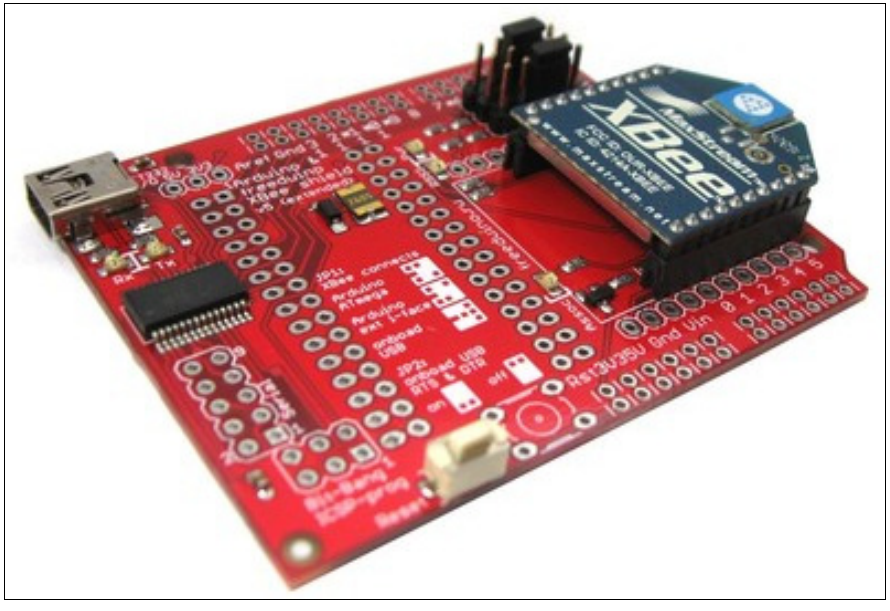


Рис. 3.3. XBee Shield

- **MicroSD Shield** (рис. 3.4) — обеспечивает запись данных на карты microSD;
- **MP3 Shield** (рис. 3.5) — плата для воспроизведения звука в форматах Ogg Vorbis/MP3/AAC/WMA/MIDI и записи в Ogg Vorbis;

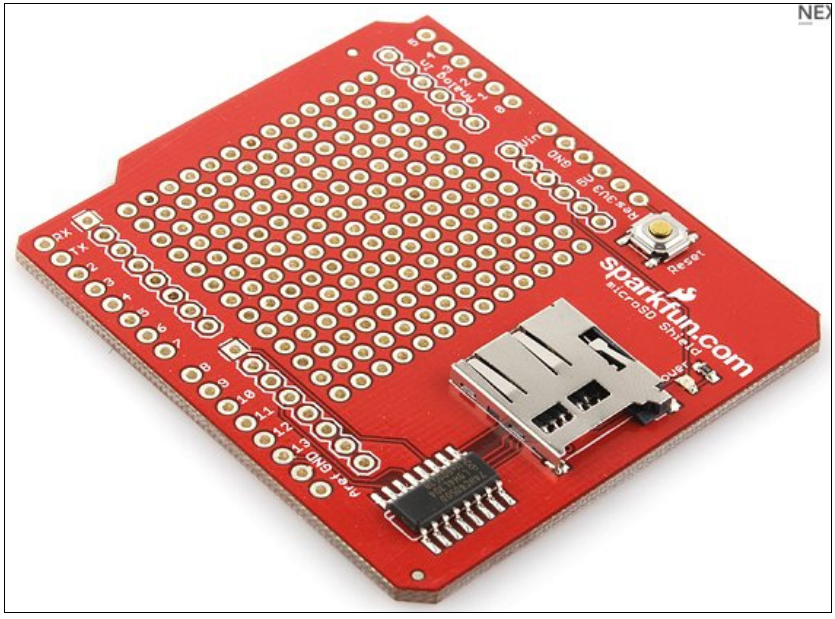


Рис. 3.4. MicroSD Shield

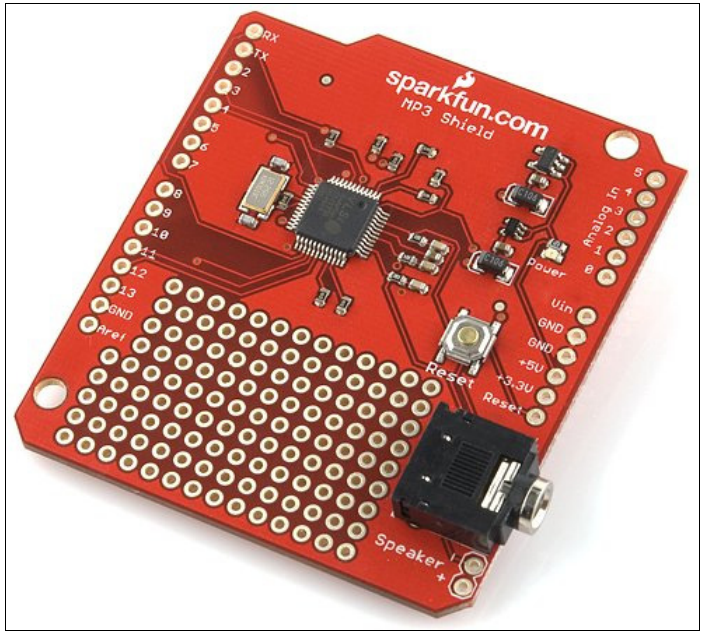


Рис. 3.5. MP3 Shield

- **Motor Shield** (рис. 3.6) — обеспечивает управление двигателями постоянного тока;
- **GSM/GPRS Shield** (рис. 3.7) — позволяет отправлять SMS-сообщения, делать звонки, обмениваться данными по GPRS;

Рис. 3.6. Motor Shield

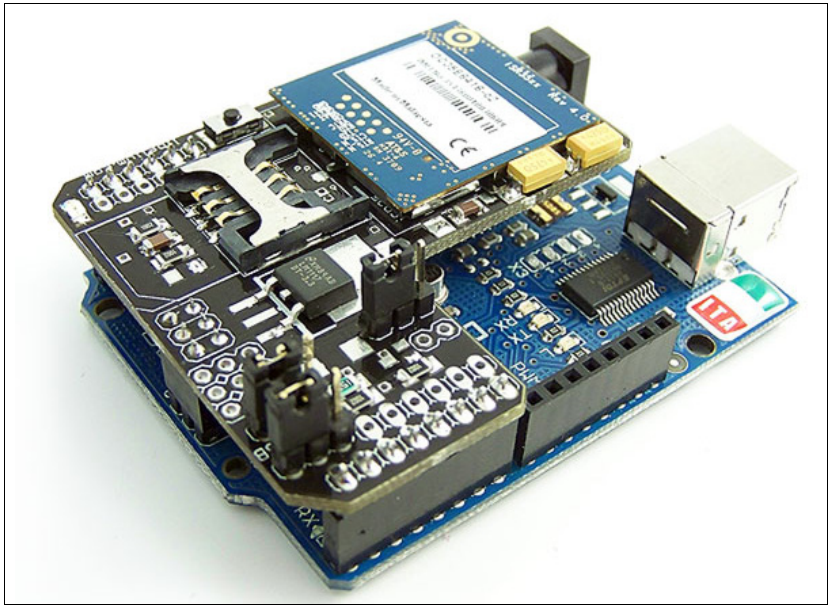


Рис. 3.7. GSM/GPRS Shield

- **Cosmo WiFi Connect** (рис. 3.8) — плата предназначена для организации беспроводной сети стандарта IEEE 802.11b/g.

Существуют также шилды: **Video Overlay Shield** — для наложения текста на аналоговое видео, **EasyVR Arduino Shield** — многоцелевой модуль распознавания речи, **Music Shield** — профессиональный аудиокодек и др.

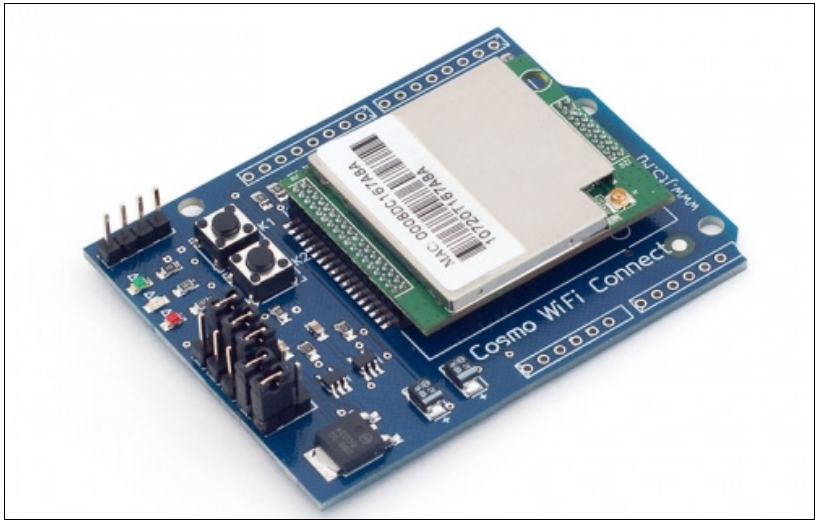


Рис. 3.8. Cosmo WiFi Connect

Количество плат расширения (шилдов) постоянно растет. Ознакомиться с их списком можно на официальном сайте проекта Arduino по адресу **http://www.arduino.cc/playground/Main/SimilarBoards#goShie**. Имеется также и еще один интересный ресурс: **http://shieldlist.org/**, где представлено описание 244 шилдов для Arduino.

ЧАСТЬ II

Среда разработки и язык программирования контроллеров Arduino

ГЛАВА 4

Среда программирования Arduino IDE

Разработка собственных приложений на базе плат, совместимых с архитектурой Arduino, осуществляется в официальной бесплатной среде программирования Arduino IDE. Среда предназначена для написания, компиляции и загрузки собственных программ в память микроконтроллера, установленного на плате Arduino-совместимого устройства. Основой среды разработки является язык Processing/Wiring — это фактически обычный C++, дополненный простыми и понятными функциями для управления вводом/выводом на контактах. Существуют версии среды для операционных систем Windows, Mac OS и Linux.

Последнюю версию среды Arduino 1.0.5 и бета-версию Arduino 1.5.2 (с поддержкой Arduino Due) можно скачать со страницы загрузки официального сайта **http://arduino.cc/en/Main/Software**.

4.1. Установка Arduino IDE в Windows

Отправляемся на страницу **http://arduino.cc/en/Main/Software** (рис. 4.1), выбираем версию для операционной системы Windows и скачиваем архивный файл. Он занимает чуть более 80 Мбайт и содержит все необходимое, в том числе и драйверы. По окончании загрузки распаковываем скачанный файл в удобное для себя место.

Теперь необходимо установить драйверы. Подключаем Arduino к компьютеру. На контроллере должен загореться индикатор питания — зеленый светодиод. Windows начинает попытку установки драйвера, которая заканчивается сообщением **Программное обеспечение драйвера не было установлено**.

Открываем Диспетчер устройств. В составе устройств находим значок Arduino Uno — устройство отмечено восклицательным знаком. Щелкаем правой кнопкой мыши на значке Arduino Uno и в открывшемся окне выбираем пункт **Обновить драйверы** и далее пункт **Выполнить поиск драйверов на этом компьютере**. Указываем путь к драйверу — ту папку на компьютере, куда распаковывали скачанный архив. Пусть это будет папка drivers каталога установки Arduino — например, C:\arduino-1.0\drivers. Игнорируем все предупреждения Windows и получаем в результате сообщение **Обновление программного обеспечения для данного устройст-**

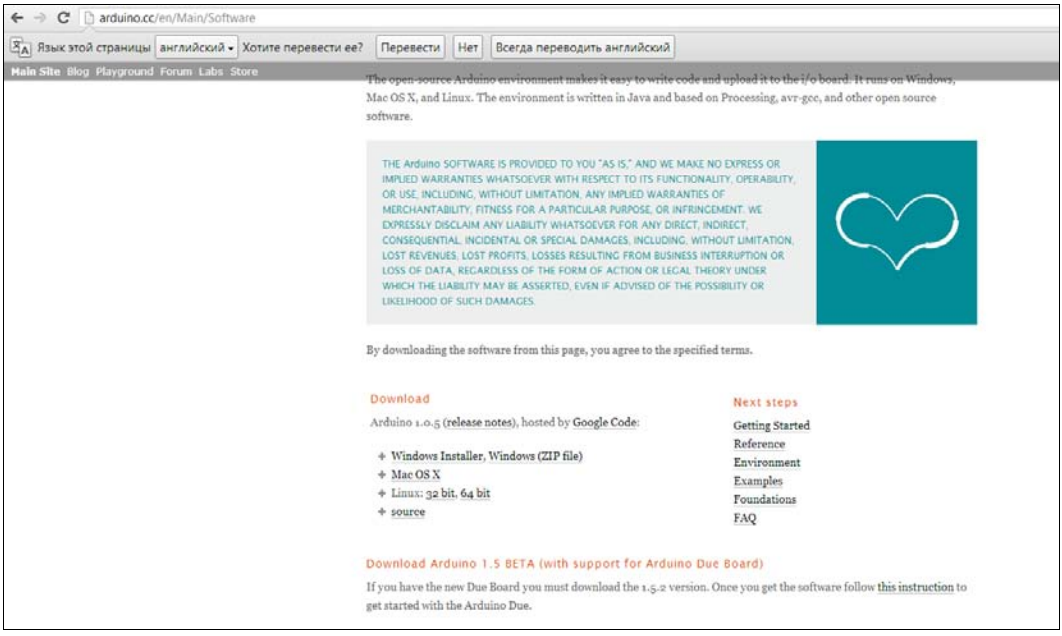


Рис. 4.1. Страница загрузки официального сайта Arduino

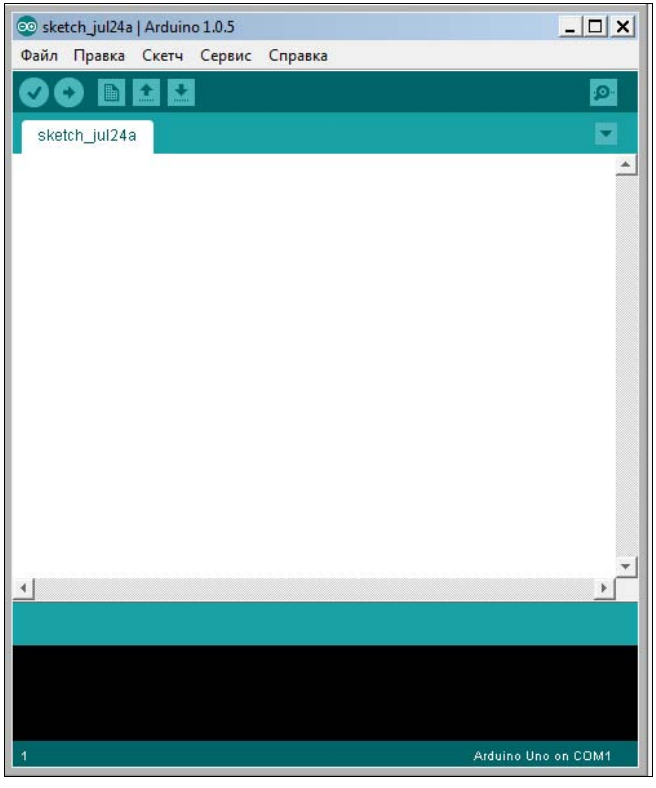


Рис. 4.2. Arduino IDE — среда разработки

ва завершено успешно. В заголовке окна будет указан и COM-порт, на который установлено устройство.

Осталось запустить среду разработки Arduino IDE (рис. 4.2).

4.2. Установка Arduino IDE в Linux

В Linux Ubuntu среда Arduino IDE устанавливается просто — она находится в депозитарии стандартных приложений Linux. Выбираем Arduino IDE из списка доступных программ в меню Ubuntu **Приложения | Центр приложений Ubuntu | Загрузить приложение**. В списке разделов выбираем **Инструменты разработчика**, в списке следующего уровня — **Все приложения** и в следующем открывшемся списке — **Arduino IDE** (рис. 4.3). Щелкаем левой кнопкой мыши на значке этой программы, справа от нее появляется кнопка **Установить**, нажимаем на эту кнопку, и среда устанавливается автоматически.

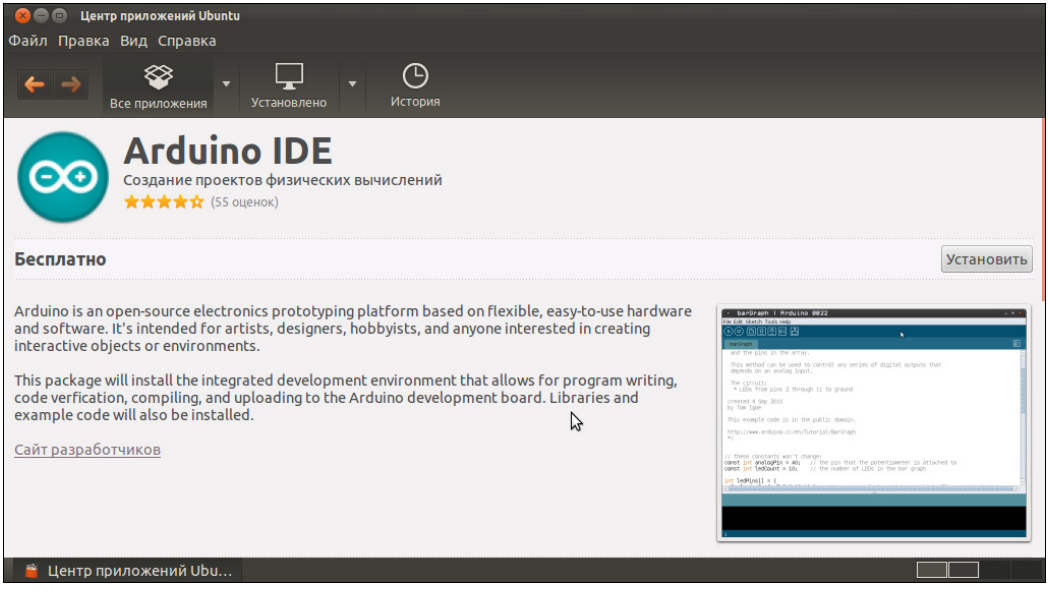


Рис. 4.3. Выбор программы из центра приложений Ubuntu

4.3. Настройка средв Arduino IDE

Среда разработки Arduino состоит из:

- редактора программного кода;
- области сообщений;
- окна вывода текста;
- панели инструментов с кнопками часто используемых команд;
- нескольких меню.

Программа, написанная в среде Arduino, носит название *скетч*. Скетч пишется в текстовом редакторе, который имеет цветовую подсветку создаваемого программного кода. Во время сохранения и экспорта проекта в области сообщений появляются пояснения и информация об ошибках. Окно вывода текста показывает сообщения Arduino, включающие полные отчеты об ошибках и другую информацию. Кнопки панели инструментов позволяют проверить и записать программу, создать, открыть и сохранить скетч, открыть мониторинг последовательной шины.

Разрабатываемым скетчам дополнительная функциональность может быть добавлена с помощью *библиотек*, представляющих собой специальным образом оформленный программный код, реализующий некоторый функционал, который можно подключить к создаваемому проекту. Специализированных библиотек существует множество. Обычно библиотеки пишутся так, чтобы упростить решение той или иной задачи и скрыть от разработчика детали программно-аппаратной реализации. Среда Arduino IDE поставляется с набором стандартных библиотек: Serial, EEPROM, SPI, Wire и др. Они находятся в подкаталоге libraries каталога установки Arduino. Необходимые библиотеки могут быть также загружены с различных ресурсов. Папка библиотеки копируется в каталог стандартных библиотек (подкаталог libraries каталога установки Arduino). Внутри каталога с именем библиотеки находятся файлы *.cpp, *.h. Многие библиотеки снабжаются примерами, расположенными в папке examples. Если библиотека установлена правильно, то она появляется в меню **Sketch | Import Library**. Выбор библиотеки в меню приведет к добавлению в исходный код строчки:

```
#include <имя библиотеки.h>
```

Эта директива подключает заголовочный файл с описанием объектов, функций и констант библиотеки, которые теперь могут быть использованы в проекте. Среда Arduino будет компилировать создаваемый проект вместе с указанной библиотекой.

Перед загрузкой скетча требуется задать необходимые параметры в меню **Сервис | Плата** (Tools | Board) — как показано на рис. 4.4, и **Сервис | Последовательный порт** (Tools | Serial Port) — как на рис. 4.5.

Современные платформы Arduino перезагружаются автоматически перед загрузкой. На старых платформах необходимо нажать кнопку перезагрузки. На большинстве плат во время процесса загрузки будут мигать светодиоды RX и TX.

При загрузке скетча используется загрузчик (bootloader) Arduino — небольшая программа, загружаемая в микроконтроллер на плате. Она позволяет загружать программный код без использования дополнительных аппаратных средств. Работа загрузчика распознается по миганию светодиода на цифровом выводе D13.

Монитор последовательного порта (Serial Monitor) отображает данные, посылаемые в платформу Arduino (плату USB или плату последовательной шины). Для отправки данных необходимо ввести в соответствующее поле текст и нажать кнопку **Послать** (Send) или клавишу <Enter> (рис. 4.6). Затем следует из выпадающего списка выбрать скорость передачи, соответствующую значению `Serial.begin` в скетче. На ОС Mac или Linux при подключении мониторинга последовательной шины платформа Arduino будет перезагружена (скетч начнется сначала).

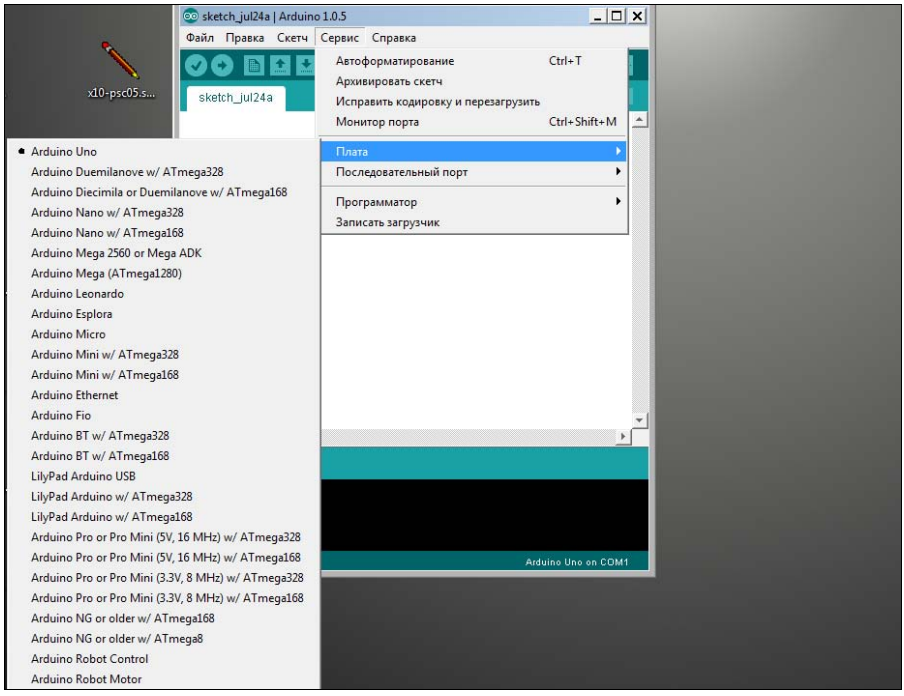


Рис. 4.4. Arduino IDE — выбор платы

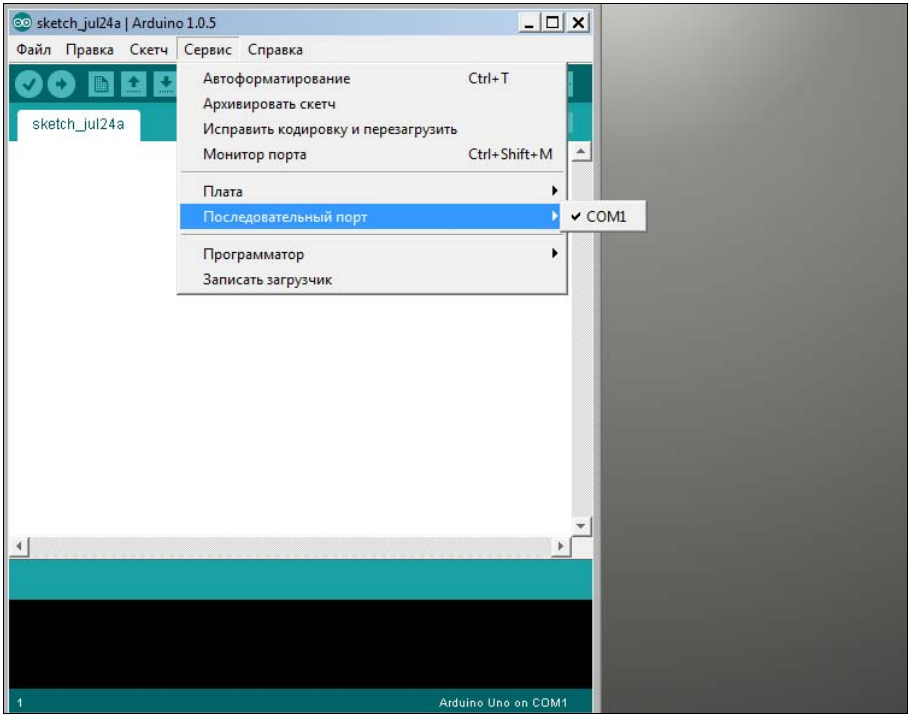


Рис. 4.5. Arduino IDE — выбор последовательного порта

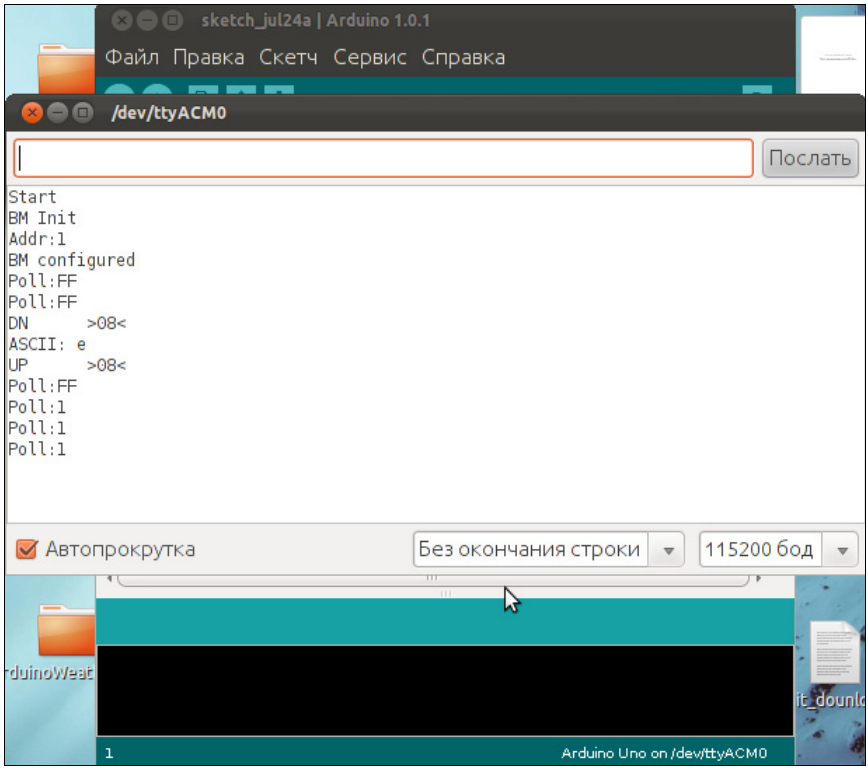


Рис. 4.6. Arduino IDE — монитор последовательного порта

ГЛАВА 5

Программирование в Arduino

5.1. Базовые знания

5.1.1. Цифровые выводы

Выводы платформы Arduino могут работать как входы или как выходы. Также аналоговые входы Arduino (ATmega) могут конфигурироваться и работать так же, как и цифровые порты ввода/вывода.

Выводы Arduino настроены как порты ввода, поэтому не требуется декларации в функции `pinMode()`. Сконфигурированные порты ввода находятся в высокоимпедансном состоянии. Это означает, что порт ввода дает слишком малую нагрузку на схему, в которую он включен. Для перевода порта ввода из одного состояния в другое требуется маленькое значение тока. Если к выводу ничего не подключено, то значения на нем будут принимать случайные величины, наводимые электрическими помехами.

Если на порт ввода не поступает сигнал, то рекомендуется задать порту известное состояние. Это делается добавлением подтягивающих резисторов 10 кОм, подключающих вход либо к питанию +5 В, либо к земле.

Микроконтроллер ATmega имеет программируемые встроенные подтягивающие резисторы 20 кОм. Программирование данных резисторов осуществляется так:

```
pinMode(pin, INPUT);     // назначить выводу порт ввода
digitalWrite(pin, HIGH);   // включить подтягивающий резистор
```

Выводы, сконфигурированные как порты вывода находятся в низкоимпедансном состоянии. Данные выводы могут пропускать через себя достаточно большой ток. Выводы микросхемы ATmega могут быть источником тока до 40 мА. Такого значения тока недостаточно для большинства реле, соленоидов и двигателей.

Короткие замыкания выводов Arduino или попытки подключить энергоемкие устройства могут повредить выходные транзисторы вывода или весь микроконтроллер ATmega.

5.1.2. Аналоговые входы

Микроконтроллеры ATmega, используемые в Arduino, содержат шестиканальный аналого-цифровой преобразователь (АЦП). Разрешение преобразователя составляет 10 битов, что позволяет на выходе получать значения от 0 до 1023. Аналоговые входы могут использоваться как цифровые выводы портов ввода/вывода, при этом они имеют номера от 14 до 19:

```
pinMode(14,OUTPUT);
digitalWrite(14, HIGH);
```

Для вывода, работавшего ранее как цифровой порт вывода, команда `analogRead` будет работать некорректно. В этом случае рекомендуется сконфигурировать его как аналоговый вход.

5.1.3. Широтно-импульсная модуляция

Широтно-импульсная модуляция (ШИМ) — это операция получения изменяющегося аналогового значения посредством цифровых устройств. Подавая на выход сигнал, состоящий из высоких и низких уровней, мы моделируем напряжение между максимальным значением (5 В) и минимальным (0 В). Длительность включения максимального значения называется шириной импульса. Для получения различных аналоговых величин изменяется ширина импульса. В результате на выходе будет величина напряжения, равная площади под импульсами (рис. 5.1).

Вызов функции `analogWrite()` с масштабом 0–255 означает, что значение `analogWrite(255)` будет соответствовать 5 В (100 % рабочий цикл — постоянное включение 5 В), а значение `analogWrite(127)` — 2,5 В (50 % рабочий цикл).

5.1.4. Память в Arduino

В микроконтроллерах ATmega168, ATmega328, ATmega1280, ATmega2560, используемых на платформах Arduino, существует три вида памяти:

- ❒ флеш-память — используется для хранения скетчей;
- ❒ ОЗУ (статическая оперативная память) — служит для хранения и работы переменных;
- ❒ EEPROM (энергонезависимая память) — применяется для хранения постоянной информации.

Флеш-память и EEPROM являются энергонезависимыми видами памяти (данные сохраняются при отключении питания). ОЗУ является энергозависимой памятью.

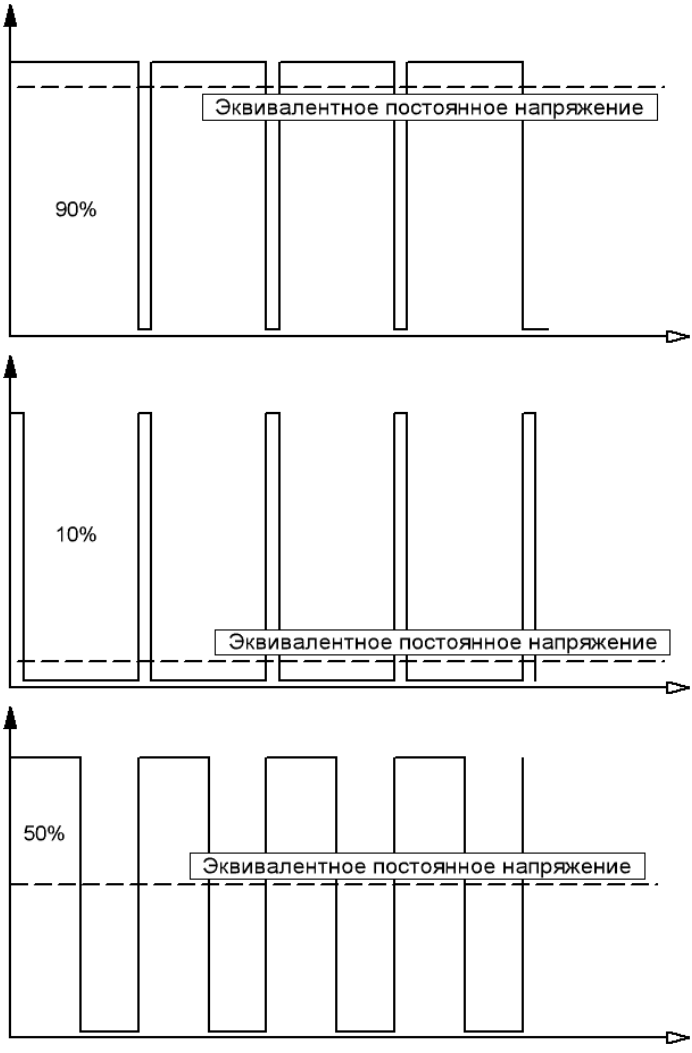


Рис. 5.1. Широтно-импульсная модуляция

Микроконтроллер ATmega168 имеет:

- 16 Кбайт флеш-памяти (2 Кбайт используется для хранения загрузчика);
- 1024 байта ОЗУ;
- 512 байт EEPROM.

Для ATmega328 эти показатели следующие:

- 32 Кбайт флеш-памяти (2 Кбайт используется для хранения загрузчика);
- 2 Кбайт ОЗУ;
- 1024 байт EEPROM.

Для ATmega1280 эти показатели следующие:

- 128 Кбайт флеш-памяти (2 Кбайт используется для хранения загрузчика);
- 8 Кбайт ОЗУ;
- 4096 байт EEPROM.

Для ATmega2560 эти показатели следующие:

- 256 Кбайт флеш-памяти (2 Кбайт используется для хранения загрузчика);
- 16 Кбайт ОЗУ;
- 9182 байт EEPROM.

При отсутствии свободного места в ОЗУ могут произойти сбои программы.

5.2. Структура программы

Arduino программируется на языке Wiring, которого на самом деле не существует, как не существует и компилятора Wiring — написанные на Wiring программы преобразуются в программу на языке C/C++ и затем компилируются компилятором AVR-GCC. Фактически используется специализированный для микроконтроллеров AVR вариант C/C++.

5.2.1. Функции *setup()* и *loop()*

Базовая структура программы для Arduino состоит, по меньшей мере, из двух обязательных частей: функций `setup()` и `loop()`. Перед функцией `setup()` идет объявление переменных, подключение библиотек. Функция `setup()` запускается один раз после каждого включения питания или сброса платы Arduino. Она используется для инициализации переменных, установки режима работы портов и прочих подготовительных для основного цикла программы действий. Она обязательно должна быть включена в программу, даже если не выполняет никаких действий.

Функция `loop()` в бесконечном цикле последовательно исполняет команды, которые описаны в ее теле. Эта функция выполняется циклически, она выполняет основную работу.

Пример простейшей программы представлен в листинге 5.1.

Листинг 5.1

```
void setup()
 {
 Serial.begin(9600);
 }
void loop()
 {
 Serial.println(millis());
 delay(1000);
 }
```

5.3. Синтаксис и операторы

5.3.1. Управляющие операторы

5.3.1.1. Оператор *if* (условие) и операторы сравнения ==, *!*=, < , >

Оператор `if` используется в сочетании с операторами сравнения, он проверяет, достигнута ли истинность условия — например, превышает ли входное значение заданное число. Формат оператора `if` следующий:

```
if (someVariable > 50){
 // выполнять действия
 }
```

Программа проверяет, значение `someVariable` больше чем 50 или нет. Если да, то выполняются определенные действия. Говоря иначе, если выражение в круглых скобках истинно, выполняются операторы внутри фигурных скобок. Если нет, программа пропускает этот код.

Выражения, которые вычисляются внутри круглых скобок, могут состоять из одного или нескольких операторов.

Операторы сравнения:

- `x == y` (x равно y);
- `x != y` (x не равно y);
- `x < y` (x меньше чем y);
- `x > y` (x больше чем y);
- `x <= y` (x меньше чем или равно y);
- `x >= y` (x больше чем или равно y).

5.3.1.2. Оператор *if..else*

Конструкция `if..else` предоставляет больший контроль над процессом выполнения кода, чем базовый оператор `if`, позволяет сделать выбор "либо, либо". Например:

```
if (pinInput==HIGH)
 {doFun1();}
else
 {doFun2();}
```

`Else` позволяет делать отличную от указанной в `if` проверку, чтобы можно было осуществлять сразу несколько взаимоисключающих проверок. Каждая проверка позволяет переходить к следующему за ней оператору не раньше, чем получит логический результат ИСТИНА. Когда проверка с результатом ИСТИНА найдена, запускается вложенный в нее блок операторов, и затем программа игнорирует все следующие строки в конструкции `if..else`. Если ни одна из проверок не получила результат ИСТИНА, по умолчанию выполняется блок операторов в `else`, если по-

следний присутствует, и устанавливается действие по умолчанию. Конструкция `else if` может быть использована с или без заключительного `else` и наоборот. Допускается неограниченное число таких переходов `else if` (листинг 5.2).

Листинг 5.2

```
if (pinAnalogInput < 100)
 {doFun1();}
else if (pinAnalogInput >= 150)
 {doFun2();}
else
 {doFun3();}
```

Другой способ создания переходов со взаимоисключающими проверками использует оператор `switch case`.

5.3.1.3. Оператор *for*

Конструкция `for` используется для повторения блока операторов, заключенных в фигурные скобки. Счетчик приращений обычно используется для приращения и завершения цикла. Оператор `for` подходит для любых повторяющихся действий и часто используется в сочетании с массивами коллекций данных/выводов.

Заголовок цикла `for` состоит из трех частей:

```
for (initialization; condition; increment) {операторы, выполняющиеся в цикле}
```

Инициализация (`initialization`) выполняется самой первой и один раз. Каждый раз в цикле проверяется условие (`condition`), если оно верно, выполняется блок операторов и приращение (`increment`), затем условие проверяется вновь. Когда логическое значение условия становится ложным, цикл завершается. В листинге 5.3 приведен пример затемнения светодиода с использованием ШИМ-вывода.

Листинг 5.3

```
// Затемнение светодиода с использованием ШИМ-вывода
int PWMpin = 10; // Светодиод последовательно с R=470 Ом на 10 выводе
void setup()
 {;}
void loop()
 {
 for (int i=0; i <= 255; i++)
 {
 analogWrite(PWMpin, i);
  delay(10);
 }
 }
```

5.3.1.4. Оператор *switch*

Конструкция `switch...case` управляет процессом выполнения программы, позволяя программисту задавать альтернативный код, который будет выполняться при разных условиях. Оператор `switch` сравнивает значение переменной со значением, определенном в операторах `case`. Когда найден оператор `case`, значение которого равно значению переменной, выполняется программный код в этом операторе. Ключевое слово `break` является командой выхода из оператора `case` и обычно используется в конце каждого `case`. Без оператора `break` оператор `switch` будет продолжать вычислять следующие выражения, пока не достигнет `break` или конец оператора `switch`. Синтаксис команды `switch...case` представлен в листинге 5.4.

Листинг 5.4

```
switch (var)
 {
 case label1:
 // код для выполнения
 break;
 case label2:
 // код для выполнения
 break;
 case label3:
 // код для выполнения
 break;
 default:
 // код для выполнения
 break;
 }
```

Параметры:

- `var` — переменная, которая вычисляется для сравнения с вариантами в `case`;
- `label` — значение, с которым сравнивается значение переменной.

5.3.1.5. Оператор *while*

Оператор `while` будет вычислять в цикле непрерывно и бесконечно до тех пор, пока выражение в круглых скобках не станет равно логическому ЛОЖНО. Что-то должно изменять значение проверяемой переменной, иначе выход из цикла `while` никогда не будет достигнут. Это изменение может происходить как в программном коде, например, при увеличении переменной, так и во внешних условиях, например, при тестировании датчика. Синтаксис команды следующий:

```
while(выражение)
 {
 // операторы
 }
```

Пример использования оператора `while` представлен в листинге 5.5.

Листинг 5.5

```
var i=0;
while($i<100)
 {
 // операторы
 i++;
 }
```

5.3.1.6. Оператор *do ... while*

Цикл `do` работает так же, как и цикл `while`, за исключением того, что условие проверяется в конце цикла. Таким образом, цикл `do` будет всегда выполняться хотя бы раз. Пример использования оператора `do ... while` представлен в листинге 5.6.

Листинг 5.6

```
do {
delay(50);    // подождать, пока датчики стабилизируются
x = readSensors(); // проверить датчики
} while (x < 100);
```

5.3.1.7. Оператор *break*

Оператор `break` используется для принудительного выхода из циклов `do`, `for` или `while`, не дожидаясь завершения цикла по условию. Он также используется для выхода из оператора `switch`. Пример приведен в листинге 5.7.

Листинг 5.7

```
for (x = 0; x < 255; x ++)
 {
 digitalWrite(PWMpin, x);
 sens = analogRead(sensorPin);
 if (sens > threshold)
 { // выходим из цикла, если есть сигнал с датчика
 x = 0;
 break;
 }
 delay(50);
 }
```

5.3.1.8. Оператор *continue*

Оператор `continue` пропускает оставшиеся операторы в текущем шаге цикла. Вместо них выполняется проверка условного выражения цикла, которая происходит при каждой следующей итерации. Пример приведен в листинге 5.8.

Листинг 5.8

```
for (x = 0; x < 255; x ++)
 {
 if (x > 40 && x < 120)
 { // если истина то прыгаем сразу на следующую итерацию цикла
 continue;
 }
 digitalWrite(PWMpin, x);
 delay(50);
 }
```

5.3.1.9. Оператор *return*

Оператор `return` прекращает вычисления в функции и возвращает значение из прерванной функции в вызывающую, если это нужно. Пример возврата значения из функции в зависимости от значения на входе аналогового входа представлен в листинге 5.9.

Листинг 5.9

```
int checkSensor()
 {
 if (analogRead(0) > 200)
 return 1;
 else{
 return 0;
 }
```

5.3.2. Синтаксис

5.3.2.1. *;* (точка с запятой) *;* (semicolon)

`;` (точка с запятой) используется для обозначения конца оператора.

```
int a = 13;
```

5.3.2.2. *{}* (фигурные скобки) *{}* (curly braces)

Фигурные скобки `{}` — важный элемент языка программирования C. Открывающая скобка `{` должна всегда сопровождаться закрывающей скобкой `}`. Это условие, известное как парность (симметричность) фигурных скобок.

Основные способы использования фигурных скобок:

- функции:
  - `void НазваниеФункции (тип данных аргумента) { оператор(ы) };`

- циклы:
  - `while (логическое выражение){ оператор(ы)};`
  - `do { оператор(ы)} while (логическое выражение);`
  - `for (инициализация; условие окончания цикла; приращения цикла) { оператор(ы)};`
- условные операторы:
  - `if (логическое выражение) {оператор(ы)}.`

5.3.2.3. Комментарии // (single line comment), /* */ (multi-line comment)

Комментарии — это строки в программе, которые используются для информирования вас самих или других о том, как работает программа. Они игнорируются компилятором и не занимают место в памяти микроконтроллера.

Есть два способа пометить строку как комментарий:

- однострочный комментарий — `// ;`
- многострочный комментарий — `/* ... */.`

Пример приведен в листинге 5.10.

Листинг 5.10

```
x = 5; // Это комментарий в одной строке. Все после двойного
       // слэша – комментарий до конца строки
/* это многострочный комментарий – используйте его для закомментирования целых
кусков кода */
```

5.3.3. Арифметические операторы

5.3.3.1. = (assignment) = оператор присваивания

Присваивает переменной слева от оператора значение переменной или выражения, находящееся справа (листинг 5.11).

Листинг 5.11

```
int sensVal;      // объявление переменной типа integer
senVal=analogRead(0); // присваивание переменной sensVal значение,
            // считанное с аналогового входа 0
```

Переменная слева от оператора присваивания (=) должна быть способна сохранить присваиваемое значение. Если оно выходит за диапазон допустимых значений, то сохраненное значение будет не верно. Необходимо различать оператор присваивания (=) и оператор сравнения (== двойной знак равенства), который осуществляет проверку на равенство.

5.3.3.2. + (сложение), - (вычитание), * (умножение) , / (деление)

Операторы `+`, `-`, `*` и `/`, соответственно, возвращают результат выполнения арифметических действий над двумя операндами. Возвращаемый результат будет зависеть от типа данных операндов, например, 9 / 4 возвратит 2, т. к. операнды 9 и 4 имеют тип `int`. Также следует следить за тем, чтобы результат не вышел за диапазон допустимых значений для используемого типа данных. Так, например, сложение 1 с переменной типа `int` и значением 32 767 возвратит –32 768. Если операнды имеют разные типы, то тип с более "широким" диапазоном будет использован для вычислений. Если один из операндов имеет тип `float` или `double`, то арифметика "с плавающей запятой" будет использована для вычислений.

5.3.3.3. % (modulo)

Возвращает остаток от деления одного целого (`int`) операнда на другой. Примеры:

```
x = 9 % 5; // x имеет значение 4
x = 5 % 5; // x имеет значение 0
```

Нельзя применить к типу `float`.

5.3.4. Операторы сравнения

Операторы сравнения:

- `x == y` (x равно y);
- `x != y` (x не равно y);
- `x < y` (x меньше чем y);
- `x > y` (x больше чем y);
- `x <= y` (x меньше чем или равно y);
- `x >= y` (x больше чем или равно y).

5.3.5. Логические операторы

Логические операторы чаще всего используются в проверке условия оператора `if`.

5.3.5.1. && (логическое И)

Истина, если оба операнда истина (`true`). Пример:

```
if (digitalRead(2) == HIGH && digitalRead(3) == HIGH)
 Serial.println("ok");
```

5.3.5.2. || (логическое ИЛИ)

Истина, если хотя бы один операнд истина. Пример:

```
if (digitalRead(2) == || digitalRead(3) == HIGH)
 Serial.println("ok");
```

5.3.5.3. ! (логическое отрицание)

Истина, если операнд `false`, и наоборот. Пример:

```
if (!(digitalRead(2)== HIGH))
 Serial.println("ok");
```

5.3.6. Унарные операторы

5.3.6.1. ++ (увеличение значения) / -- (уменьшение значения)

Унарные (имеющие один операнд) операторы ++, -- увеличивают, уменьшают значение переменной соответственно (листинг 5.12).

Листинг 5.12

```
x++; // увеличивает значение x на единицу и возвращает старое значение x
++x; // увеличивает значение x на единицу и возвращает новое значение x
x-- ; // уменьшает значение x на единицу и возвращает старое значение x
--x ; // уменьшает значение x на единицу и возвращает новое значение x
```

5.3.6.2. += , -= , *= , /=

Короткий способ записи арифметических действий над переменной и одним операндом (листинг 5.13).

Листинг 5.13

```
x += y; // эквивалент записи x = x + y;
x -= y; // эквивалент записи x = x - y;
x *= y; // эквивалент записи x = x * y;
x /= y; // эквивалент записи x = x / y;
```

5.4. Данные

5.4.1. Типы данных

Компилятор Arduino определяет следующие типы данных:

- `boolean`;
- `char`;
- `byte`;
- `int`;
- `unsigned int`;
- `long`;
- `unsigned long`;
- `float`;
- `double`;
- `string`;
- `массив (array)`;
- `void`.

Рассмотрим типы данных более подробно.

5.4.1.1. *boolean*

Логический (булевый) тип данных — `boolean`. Может принимать одно из двух значений: `true` или `false`. Данные типа `boolean` занимают в памяти один байт.

5.4.1.2. *char*

Переменная типа `char` занимает 1 байт памяти и может хранить один алфавитно-цифровой символ (литеру). При объявлении литеры используются одиночные кавычки: 'A' (двойные кавычки используются при объявлении строки символов — тип `string`: "ABC").

Символ хранится в памяти как число, соответствующее коду символа в таблице кодировки символов ASCII. Так как символ хранится как число, в памяти над ним возможно производить арифметические действия (например, 'A' + 1 будет 66, т. к. ASCII код для 'A' — 65).

Тип `char` знаковый тип, т. е. число (код), хранящийся в памяти, может принимать значения от –128 до 127. Если необходима знаковая однобайтовая переменная, используйте тип `byte`.

Пример:

```
char myChar = 'A';
char myChar = 65;   // Варианты эквивалентны
```

5.4.1.3. *byte*

Хранит 8-битовое числовое значение без десятичной точки. Имеет диапазон от 0 до 255. Пример:

```
byte someVariable=150; // объявление переменной someVariable,
            // имеющей тип byte
```

5.4.1.4. *int*

Тип данных `int` (от англ. *integer* — целое число) — один из наиболее часто используемых типов данных для хранения чисел. `int` занимает 2 байта памяти и может хранить числа от –32 768 до 32 767.

Для размещения отрицательных значений `int` использует так называемый *дополнительный* код представления числа. Старший бит указывает на отрицательный знак числа, остальные биты инвертируются с добавлением 1.

Arduino-компилятор сам заботится о размещении в памяти и представлении отрицательных чисел, поэтому арифметические действия над целыми числами производятся как обычно.

Когда переменная типа `int` вследствие арифметической операции достигает своего максимального значения, она "перескакивает" на самое минимальное значение и наоборот (листинг 5.14).

Листинг 5.14

```
int x;
x = -32,768;
x = x - 1;   // x теперь равно 32,767
x = 32,767;
x = x + 1;   // x теперь равно -32,768
```

5.4.1.5. *unsigned int*

Тип данных `unsigned int` — беззнаковое целое число, так же как и тип `int` (знаковое), занимает в памяти 2 байта. Но в отличие от `int`, тип `unsigned int` может хранить только положительные целые числа в диапазоне от 0 до 65 535.

Отличие кроется в том, как `unsigned int` использует старший бит, иногда называемый *знаковым битом*. Если старший бит равен 1, то для типа `int` компилятор Arduino считает, что это число отрицательное, а остальные 15 битов несут информацию о модуле целого числа в дополнительном коде представления числа, в то время как `unsigned int` использует все 16 битов для хранения модуля числа.

Когда переменная типа `unsigned int` вследствие арифметической операции достигает своего максимального значения, она "перескакивает" на самое минимальное значение и наоборот (листинг 5.15).

Листинг 5.15

```
unsigned int x;
x = 0;
x = x - 1;   // x теперь равна 65535
x = x + 1;   // x теперь 0
```

5.4.1.6. *long*

Тип данных `long` используется для хранения целых чисел в расширенном диапазоне от –2 147 483 648 до 2 147 483 647. `long` занимает 4 байта в памяти. Пример:

```
long var1 = -178000;
```

5.4.1.7. *unsigned long*

`Unsigned long` используется для хранения положительных целых чисел в диапазоне от 0 до 4 294 967 295 и занимает 32 бита (4 байта) в памяти. Пример вывода в миллисекундах (мс) с начала выполнения программы приведен в листинге 5.16.

Листинг 5.16

```
void loop()
 {
 Serial.print("Time: ");
 time = millis();
```

```
//выводит время, прошедшее с момента начала выполнения программы
Serial.println(time);
function1();
}
```

5.4.1.8. *float*

Тип данных `float` служит для хранения чисел с плавающей запятой. Этот тип часто используется для операций с данными, считываемыми с аналоговых входов. Диапазон значений — от –3,4028235E+38 до 3,4028235E+38. Переменная типа `float` занимает 32 бита (4 байта) в памяти.

Тип `float` имеет точность 6–7 знаков, имеются в виду все знаки, а не только мантисса. Обычно для увеличения точности используют другой тип — `double`, но на платформе Arduino `double` и `float` имеют одинаковую точность.

5.4.1.9. *double*

Тип данных `double`, в отличие от большинства языков программирования, имеет ту же точность, что и тип `float` и занимает также 4 байта памяти.

Тип `double` поддерживается в Arduino для совместимости кода с другими платформами.

5.4.1.10. *string* — текстовые строки

Текстовые строки в Arduino объявляются как массив (`array`) типа `char` (символов, литер), оканчивающийся символом "конца строки". Возможны следующие варианты объявления текстовых строк;

- ❒ объявить массив символов без присваивания значений;
- ❒ объявить массив символов и присвоить значения всем элементам, кроме последнего, компилятор Arduino автоматически добавит символ конца строки;
- ❒ явно объявить завершающий символ;
- ❒ инициализировать массив строковой константой в двойных кавычках. Компилятор автоматически задаст требуемый размер на массив, равный количеству символов плюс завершающий символ;
- ❒ инициализировать массив с явным заданием размера и присвоением строковой константы;
- ❒ инициализировать массив с явным заданием дополнительного размера (с запасом), фактически превышающего размер строковой константы при начальном присвоении.

В листинге 5.17 приведены варианты объявления и присвоения строк.

Листинг 5.17

```
char Str1[15];
char Str2[8] = {'a','r','d','u','i','n','o'};
char Str3[8] = {'a','r','d','u','i','n','o','\0'};
```

```
char Str4[ ] = "arduino";
char Str5[8] = "arduino";
char Str6[15] = "arduino";
```

Обычно строки оканчиваются нулевым символом (код 0 в ASCII). Это позволяет функциям (таким как `Serial.print()`) выявлять окончание строки. В противном случае могут считаться байты памяти, не принадлежащие переменной.

Массив символов, выделяемый под строку, должен иметь один дополнительный элемент для символа конца строки. Если объявить строку без символа окончания строки, то это приведет к некорректной работе функций, оперирующих строками.

Строки всегда объявляются внутри двойных кавычек (`"Abc"`).

При работе с большими объемами текстовой информации бывает удобно использовать массивы строк. Так как строки сами по себе массивы, массив строк будет двумерным массивом.

В примере, приведенном в листинге 5.18, символ звездочки после объявления типа `"char*"` указывает на то, что это массив указателей. Это необходимо для задания двумерного массива.

Листинг 5.18

```
char* myStrings[]={"string 1", "string 2", "string 3","string 4",
         "string 5","string 6"};
void setup()
 {Serial.begin(9600);}
void loop()
 {
 for (int i = 0; i < 6; i++){
 Serial.println(myStrings[i]);
 delay(500);
 }
}
```

5.4.1.11. Массивы

Массивы (`arrays`) — именованный набор однотипных переменных с доступом к отдельным элементам по их индексу. Существует несколько вариантов объявления массива:

- массив может быть объявлен без непосредственной инициализации элементов массива:

  ```
  int myInts[6];
  ```

- массив может быть объявлен без явного задания размера. Компилятор сам посчитает фактическое количество элементов и создаст в памяти массив необходимого размера:

  ```
  int myPins[] = {2, 4, 8, 3, 6};
  ```

- при объявлении массива размер может быть задан явно, одновременно с инициализацией элементов массива, при создании массива типа `char` необходим дополнительный элемент массива для нулевого символа:

```
int mySensVals[6] = {2, 4, -8, 3, 2};
char message[6] = "hello";
```

Индексация массива начинается с 0. Присваивание значения элементу массива происходит следующим образом:

```
mySensVals[0] = 10;
Получение значения массива:
x = mySensVals[4];
```

Чаще всего для перебора элементов цикла используется цикл `for`, счетчик цикла используется как индекс для доступа к каждому элементу массива. Например, для вывода массива через последовательный порт (`Serial`) можно использовать следующий код:

```
int i;
for (i = 0; i < 5; i = i + 1)
 { Serial.println(myPins[i]); }
```

5.4.1.12. *void*

Ключевое слово `void` используется при объявлении функций, если функция не возвращает никакого значения при ее вызове.

5.4.2. Константы

Константы — предопределенные значения. Они используются, чтобы делать программы более легкими для чтения. Объявление констант (а также базовых макросов и функций) можно посмотреть в файле **\hardware\arduino\cores\arduino\wiring.h**. Рассмотрим некоторые константы.

`true/false` — это булевы константы, определяющие логические уровни. `false` легко определяется как 0 (ноль), а `true`, как 1, но может быть и чем-то другим, отличным от нуля. Поэтому –1, 2 и 200 — это все тоже определяется как `true`.

```
#define true 0x1
#define false 0x0
```

`HIGH/LOW` — уровни сигналов порта `HIGH` и `LOW`:

```
#define HIGH 0x1
#define LOW 0x0
```

`INPUT/OUTPUT` — настройка цифровых портов на ввод (`INPUT`) и вывод (`OUTPUT`) сигналов:

```
#define INPUT 0x0
#define OUTPUT 0x1
```

Цифровые порты могут использоваться на ввод или вывод сигналов. Изменение порта с ввода на вывод производится при помощи функции `pinMode()`:

```
pinMode(13, OUTPUT);  // 13 вывод будет выходом
pinMode(12, INPUT);   // 12 - входом
```

В программе можно создавать собственные константы:

```
#define LEFT 0x95
#define MESS_LEFT "поворот влево"
```

5.4.3. Переменные

Переменные — это способ именовать и хранить числовые значения для последующего использования программой. Переменные — это значения, которые могут последовательно меняться, в отличие от констант, чье значение никогда не меняется. Переменные нужно декларировать (объявлять). Следующий код объявляет переменную `inputVariable`, а затем присваивает ей значение, полученное от 2-го аналогового порта:

```
int inputVariable=0;
inputVariable=analogRead(2);
```

Переменные могут быть названы любыми именами, которые не являются ключевыми словами языка программирования Arduino.

5.4.3.1. Объявление переменных

Все переменные должны быть задекларированы до того, как они могут использоваться. Объявление переменной означает определение типа ее значения: `int`, `long`, `float` и т. д., задание уникального имени переменной, и дополнительно ей можно присвоить начальное значение. Все это следует делать только один раз в программе, но значение может меняться в любое время при использовании арифметических или других разных операций.

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

```
int inputVariable = 0;
```

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

5.4.3.2. Границы переменных

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

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

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

5.4.4. Преобразование типов данных

5.4.4.1. *char()*

`char()` приводит значение к типу `char`.

Синтаксис:

```
char(x);
```

где `x` — переменная любого типа.

5.4.4.2. *byte()*

`byte()` приводит значение к типу `byte`.

Синтаксис:

```
byte(x);
```

где `x` — переменная любого типа.

5.4.4.3. *int()*

`int()` приводит значение к типу `int`.

Синтаксис:

```
int(x);
```

где `x` — переменная любого типа.

5.4.4.4. *long()*

`long()` приводит значение к типу `long`.

Синтаксис:

```
long(x);
```

где `x` — переменная любого типа.

5.4.4.5. *float()*

`float()` приводит значение к типу `float`.

Синтаксис:

```
long(x);
```

где `x` — переменная любого типа.

5.5. Функции

5.5.1. Цифровой ввод/вывод

Рассмотрим функции цифрового ввода/вывода:

- `pinMode()`;
- `digitalWrite()`;
- `digitalRead()`.

5.5.1.1. Функция *pinMode*

Устанавливает режим работы заданного входа/выхода (pin) как входа или как выхода.

Синтаксис:

```
pinMode(pin, mode);
```

Параметры:

- `pin` — номер входа/выхода (pin), который вы хотите установить;
- `mode` — режим. Одно из двух значений: `INPUT` или `OUTPUT` устанавливает на вход или выход соответственно.

Пример:

```
int ledPin = 13; // Светодиод, подключенный к входу/выходу 13
void setup()
 {
 pinMode(ledPin, OUTPUT);   // устанавливает режим работы - выход
 }
```

5.5.1.2. Функция *digitalWrite()*

Подает `HIGH` или `LOW` значение на цифровой вход/выход (pin).

Если вход/выход (pin) был установлен в режим выход (`OUTPUT`) функцией `pinMode()`, то для значения `HIGH` напряжение на соответствующем входе/выходе (pin) будет 5 В (3,3 В для плат 3,3 В) и 0 В (земля) для `LOW`.

Если вход/выход (pin) был установлен в режим вход (`INPUT`), то функция `digitalWrite` со значением `HIGH` будет активировать внутренний нагрузочный рези-

стор 20 K. Подача LOW в свою очередь отключает этот резистор. Нагрузочного резистора достаточно, чтобы светодиод, подключенный к входу, светил тускло. Если вдруг светодиод работает, но очень тускло, возможно необходимо установить режим выход (OUTPUT) функцией pinMode().

Синтаксис:

```
digitalWrite(pin, value);
```

Параметры:

- pin — номер входа/выхода (pin);
- value — значение HIGH или LOW.

Пример представлен в листинге 5.19.

Листинг 5.19

```
int ledPin = 13;       // Светодиод, подключенный к входу/выходу 13
void setup()
 {
 pinMode(ledPin, OUTPUT);   // устанавливает режим работы - выход
 }
void loop()
 {
 digitalWrite(ledPin, HIGH); // включает светодиод
 delay(1000);          // ждет секунду
 digitalWrite(ledPin, LOW);  // выключает светодиод
 delay(1000);          // ждет секунду
 }
```

5.5.1.3. Функция *digitalRead()*

Функция считывает значение с заданного входа: HIGH или LOW.

Синтаксис:

```
digitalRead(pin);
```

Параметр: pin — номер входа/выхода (pin), который вы хотите считать.

Пример представлен в листинге 5.20.

Листинг 5.20

```
int ledPin = 13;      // Светодиод, подключенный к входу/выходу 13
int inPin = 7;       // кнопка на входе 7
int val = 0;        // переменная для хранения значения
void setup()
 {
 pinMode(ledPin, OUTPUT);  // устанавливает режим работы - выход для 13
```

```
 pinMode(inPin, INPUT);   // устанавливает режим работы - вход для 7
 }
void loop()
 {
 val = digitalRead(inPin);  // считываем значение с входа
 digitalWrite(ledPin, val); // устанавливаем значение на светодиоде
               // равным значению входа кнопки
 }
```

ЗАМЕЧАНИЕ

Если вход не подключен, то `digitalRead` может возвращать значения `HIGH` или `LOW` случайным образом. Аналоговые входы (analog pins) могут быть использованы как цифровые входы/выходы (digital pins). Обращение к ним идет по номерам от 14 (для аналогового входа 0) до 19 (для аналогового входа 5).

5.5.2. Аналоговый ввод/вывод

Рассмотрим функции аналогового ввода/вывода:

- `analogRead()`;
- `analogReference()`;
- `analogWrite()`.

5.5.2.1. Функция *analogRead()*

Функция считывает значение с указанного аналогового входа. Большинство плат Arduino имеют 6 каналов (8 каналов у платы Mini и Nano, 16 — у Mega) с 10-битным аналого-цифровым преобразователем (АЦП). Напряжение, поданное на аналоговый вход (обычно от 0 до 5 вольт), будет преобразовано в значение от 0 до 1023 — это 1024 шага с разрешением 0,0049 вольт. Разброс напряжения и шаг может быть изменен функцией `analogReference()`. Считывание значения с аналогового входа занимает примерно 100 микросекунд (0,0001 сек), т. е. максимальная частота считывания приблизительно 10 000 раз в секунду.

Синтаксис:

```
analogRead(pin);
```

Параметр: `pin` — номер порта аналогового входа, с которого будет производиться считывание: 0..5 для большинства плат, 0..7 для Mini и Nano и 0..15 для Mega.

Возвращаемое значение `int` (0 to 1023).

ЗАМЕЧАНИЕ

Если аналоговый вход не подключен, то значения, возвращаемые функцией `analogRead()`, могут принимать случайные значения.

Пример представлен в листинге 5.21.

Листинг 5.21

```
int analogPin = 3;  // номер порта, к которому подключен потенциометр
int val = 0;     // переменная для хранения считываемого значения
void setup()
 {
 Serial.begin(9600);       // установка связи по serial
 }
void loop()
 {
 val = analogRead(analogPin);  // считываем значение
 Serial.println(val);      // выводим полученное значение
 }
```

5.5.2.2. Функция *analogReference()*

Функция определяет опорное напряжение, относительно которого происходят аналоговые измерения. Функция `analogRead()` возвращает значение с разрешением 8 битов (1024) пропорционально входному напряжению на аналоговом входе и в зависимости от опорного напряжения.

Возможные настройки:

- `DEFAULT` — стандартное опорное напряжение 5 В (на платформах с напряжением питания 5 В) или 3,3 В (на платформах с напряжением питания 3,3 В);
- `INTERNAL` — встроенное опорное напряжение 1,1 В на микроконтроллерах ATmega168 и ATmega328 и 2,56 В на ATmega8;
- `EXTERNAL` — внешний источник опорного напряжения, подключенный к выводу AREF.

Синтаксис:

```
analogReference(type);
```

Параметр: `type` — определяет используемое опорное напряжение (`DEFAULT`, `INTERNAL` или `EXTERNAL`).

Внешнее напряжение рекомендуется подключать к выводу AREF через резистор 5 кОм.

Рекомендуемой настройкой для вывода AREF является `EXTERNAL`. При этом происходит отключение обоих внутренних источников, и внешнее напряжение будет являться опорным для АЦП.

5.5.2.3. Функция *analogWrite()*

Выдает аналоговую величину (ШИМ-волну) на порт входа/выхода. Функция может быть полезна для управления яркостью подключенного светодиода или скоростью вращения электродвигателя. После вызова `analogWrite()` на выходе будет генерироваться постоянная прямоугольная волна с заданной шириной импульса до сле-

дующего вызова `analogWrite` (или вызова `digitalWrite` или `digitalRead` на том же порту входа/выхода). Частота ШИМ-сигнала приблизительно 490 Гц.

На большинстве плат Arduino (на базе микроконтроллера ATmega168 или ATmega328) ШИМ поддерживают порты 3, 5, 6, 9, 10 и 11, на плате Arduino Mega — порты с 2 по 13. На более ранних версиях плат Arduino `analogWrite()` работал только на портах 9, 10 и 11.

Для вызова `analogWrite()` нет необходимости устанавливать тип входа/выхода функцией `pinMode()`. Функция `analogWrite()` никак не связана с аналоговыми входами и с функцией `analogRead()`.

Синтаксис:

```
analogWrite(pin, value);
```

Параметры:

- `pin` — порт входа/выхода, на который подается ШИМ-сигнал;
- `value` — период рабочего цикла: значение между 0 (полностью выключено) и 255 (сигнал подан постоянно).

> ***Замечание***
>
> Период ШИМ-сигнала на портах входа/выхода 5 и 6 будет несколько длиннее. Это связано с тем, что таймер для данных выходов также задействован функциями `millis()` и `delay()`. Данный эффект более заметен при установке коротких периодов ШИМ-сигнала (0–10).

Пример задания яркости светодиода пропорционально значению, снимаемому с потенциометра, представлен в листинге 5.22.

Листинг 5.22

```
int ledPin = 9;  // Светодиод подключен к выходу 9
int analogPin = 3; // потенциометр подключен к выходу 3
int val = 0;   // переменная для хранения значения
void setup()
 {
 pinMode(ledPin, OUTPUT);   // установка порта на выход
 }
void loop()
 {
 val = analogRead(analogPin); // считываем значение с порта,
              // подключенного к потенциометру
 analogWrite(ledPin, val / 4); // analogRead возвращает значения от 0
              // до 1023, analogWrite должно быть
              // в диапазоне от 0 до 255
 }
```

5.5.3. Дополнительные фунции ввода/вывода

5.5.3.1. Функция *tone()*

Генерирует на порту входа/выхода сигнал — прямоугольную "волну" заданной частоты и с 50 % рабочим циклом. Длительность может быть задана параметром, в противном случае сигнал генерируется до тех пор, пока не будет вызвана функция `noTone()`. К порту входа/выхода может быть подключен пьезо- или иной динамик для воспроизведения сигнала.

Воспроизводиться одновременно может только один сигнал. Если сигнал уже воспроизводится на одном порту, то вызов `tone()` с номером другого порта в качестве параметра ни к чему не приведет, если же `tone()` будет вызвана с тем же номером порта, то будет установлена новая частота сигнала.

Использование функции `tone()` помешает использовать ШИМ на портах входа/выхода 3 и 11 (кроме платы Arduino Mega).

Синтаксис:

```
tone(pin, frequency);
tone(pin, frequency, duration);
```

Параметры:

- `pin` — номер порта входа/выхода, на котором будет генерироваться сигнал;
- `frequency` — частота сигнала в герцах;
- `duration` — длительность сигнала в миллисекундах.

5.5.3.2. Функция *noTone()*

Останавливает сигнал, генерируемый на порту входа/выхода, вызовом функции `tone()`. Если сигнал не генерировался, то вызов `noTone()` ни к чему не приводит.

> ***Замечание***
>
> Если необходимы сигналы на разных портах, то следует сначала остановить один сигнал функцией `noTone()`, а лишь затем создавать новый сигнал на другом порту функцией `Tone()`.

Синтаксис:

```
noTone(pin);
```

Параметр: `pin` — номер порта входа/выхода, на котором прекращается сигнал.

5.5.3.3. Функция *shiftOut()*

Выводит байт информации на порт входа/выхода последовательно (побитно). Вывод может осуществляться как с первого (левого), так и с последнего (правого) бита. Каждый бит последовательно подается на заданный порт, после чего подается сигнал на синхронизирующий порт входа/выхода, информируя о доступности к считыванию бита.

Такой способ передачи данных называется *последовательным протоколом с синхронизацией*. Он часто используется для взаимодействия микроконтроллеров с датчиками и сенсорами, а также другими микроконтроллерами. Последовательная передача с синхронизацией позволяет устройствам связываться на максимальной скорости. Смотрите также документацию (на англ. языке) по протоколу последовательного периферийного интерфейса (SPI, Serial Peripheral Interface Protocol).

Синтаксис:

```
shiftOut(dataPin, clockPin, bitOrder, value);
```

Параметры:

- `dataPin` — номер порта входа/выхода, на который выводятся биты (`int`);
- `clockPin` — номер порта, по которому производится синхронизация (`int`);
- `bitOrder` — используемая последовательность вывода битов. `MSBFIRST` (Most Significant Bit First) — слева или `LSBFIRST` (Least Significant Bit First) — справа;
- `value` — значение (байт) для вывода (`byte`).

> ***Замечание***
>
> Порт вывода (`dataPin`) и синхронизирующий порт (`clockPin`) должны быть предварительно сконфигурированы как порты вывода с помощью функции `pinMode()`.

Текущая реализация функции `shiftOut()` может выводить только один байт (8 битов) информации, поэтому необходимо произвести несколько действий, чтобы вывести значения больше 255. Пример вывода приведен в листинге 5.23.

Листинг 5.23

```
// Вывод будет MSBFIRST с первого (левого) бита
int data = 500;
// выводим старший байт
shiftOut(dataPin, clock, MSBFIRST, (data >> 8));
// выводим младший бит
shiftOut(dataPin, clock, LSBFIRST, data);
// выводим старший бит
shiftOut(dataPin, clock, LSBFIRST, (data >> 8));
```

Пример вывода счетчика от 0 до 255 на сдвиговый регистр с последовательным вводом 74HC595 представлен в листинге 5.24.

Листинг 5.24

```
// Порт, подключенный к ST_CP 74HC595
int latchPin = 8;
// Порт, подключенный к SH_CP 74HC595
int clockPin = 12;
```

```
// Порт, подключенный к DS 74HC595
int dataPin = 11;
void setup()
 {
 // устанавливаем режим порта выхода
 pinMode(latchPin, OUTPUT);
 pinMode(clockPin, OUTPUT);
 pinMode(dataPin, OUTPUT);
 }
void loop()
 {
 for (int j = 0; j < 256; j++)
 {
 // устанавливаем LOW на latchPin, пока не окончена передача байта
 digitalWrite(latchPin, LOW);
 shiftOut(dataPin, clockPin, LSBFIRST, j);
 // устанавливаем HIGH на latchPin, чтобы проинформировать регистр, что
 // передача окончена.
 digitalWrite(latchPin, HIGH);
 delay(1000);
 }
 }
```

5.5.3.4. Функция *pulseIn()*

Считывает длину сигнала на заданном порту (`HIGH` или `LOW`). Например, если задано считывание `HIGH` функцией `pulseIn()`, функция ожидает, пока на заданном порту не появится `HIGH`. Когда `HIGH` получен, включается таймер, который будет остановлен, когда на порту входа/выхода будет `LOW`. Функция `pulseIn()` возвращает длину сигнала в микросекундах. Функция возвращает 0, если в течение заданного времени (тайм-аута) не был зафиксирован сигнал на порту.

Возможны некоторые погрешности в измерении длинных сигналов. Функция может измерять сигналы длиной от 10 микросекунд до 3 минут.

Синтаксис:

```
pulseIn(pin, value);
pulseIn(pin, value, timeout);
```

Параметры:

- `pin` — номер порта входа/выхода, на котором будет ожидаться сигнал;
- `value` — тип ожидаемого сигнала: `HIGH` или `LOW`;
- `timeout` — время ожидания сигнала (тайм-аут) в секундах (`unsigned long`).

Возвращаемые значения: длина сигнала в микросекундах или 0, если сигнал не получен до истечения тайм-аута (тип `unsigned long`).

Пример использования функции представлен в листинге 5.25.

Листнг 5.25

```
int pin = 7;
unsigned long duration;
void setup()
 {
 pinMode(pin, INPUT);
 }
void loop()
 {
 duration = pulseIn(pin, HIGH);
 }
```

5.5.4. Работа со временем

5.5.4.1. Функция *millis()*

Возвращает количество миллисекунд с момента начала выполнения текущей программы на плате Arduino. Это количество сбрасывается на ноль вследствие переполнения значения приблизительно через 50 дней.

Параметров нет.

Возвращаемое значение — количество миллисекунд с момента начала выполнения программы (тип `unsigned long`).

Пример использования функции представлен в листинге 5.26.

Листинг 5.26

```
unsigned long time;
void setup()
 {
 Serial.begin(9600);
 }
void loop()
 {
 Serial.print("Time: ");
 time = millis();
 // выводит количество миллисекунд с момента начала выполнения программы
 Serial.println(time);
 // ждет секунду перед следующей итерацией цикла.
 delay(1000);
 }
```

5.5.4.2. Функция *micros()*

Возвращает количество микросекунд с момента начала выполнения текущей программы на плате Arduino. Значение переполняется и сбрасывается на ноль приблизительно через 70 минут. На платах Arduino с 16 МГц (Duemilanove и Nano) функ-

ция `micros()` имеет разрешение 4 секунды (возвращаемое значение всегда кратно 4). На платах с 8 МГц (Arduino Lilypad) — разрешение функции 8 секунд.

Параметров нет.

Возвращаемое значение — количество микросекунд с момента начала выполнения программы (`unsigned long`).

Пример использования функции представлен в листинге 5.27.

Листинг 5.27

```
unsigned long time;
void setup()
 {
 Serial.begin(9600);
 }
void loop()
 {
 Serial.print("Time: ");
 time = micros();
 // выводит количество микросекунд с момента начала выполнения
 // программы
 Serial.println(time);
 // ждет секунду перед следующей итерацией цикла.
 delay(1000);
 }
```

5.5.4.3. Функция *delay()*

Останавливает выполнение программы на заданное в параметре количество миллисекунд (1000 миллисекунд в 1 секунде).

Синтаксис:

```
delay(ms);
```

Параметр: `ms` — количество миллисекунд, на которое приостанавливается выполнение программы (тип `unsigned long`).

Пример использования функции представлен в листинге 5.28.

Листинг 5.28

```
int ledPin = 13;       // светодиод подключен на порт 13
void setup()
 {
 pinMode(ledPin, OUTPUT);   // устанавливается режим порта - выход
 }
void loop()
 {
 digitalWrite(ledPin, HIGH); // включаем светодиод
```

```
 delay(1000);          // ожидаем секунду
 digitalWrite(ledPin, LOW);  // выключаем светодиод
 delay(1000);          // ожидаем секунду
 }
```

Не рекомендуется использовать эту функцию для событий длиннее 10 миллисекунд, т. к. во время останова не могут быть произведены манипуляции с портами, не могут быть считаны сенсоры или произведены математические операции. В качестве альтернативного подхода возможно контролирование времени выполнения тех или иных функций с помощью `millis()`. При использовании функции `delay()` работа прерываний не останавливается, продолжается запись последовательно (serial) передаваемых данных на RX-порту, ШИМ-сигнал (`analogWrite`) продолжает генерироваться на портах.

5.5.4.4. Функция *delayMicroseconds()*

Останавливает выполнение программы на заданное в параметре количество микросекунд (1 000 000 микросекунд в 1 секунде).

В данной версии Arduino максимальная пауза, воспроизводимая корректно, — 16 383. Возможно, это будет изменено в следующих версиях Arduino. Для остановки выполнения программы, более чем на несколько тысяч микросекунд, рекомендуется использовать функцию `delay()`.

Синтаксис:

```
delayMicroseconds(us);
```

Параметр: `us` — количество микросекунд, на которое приостанавливается выполнение программы (`unsigned int`).

Пример использования функции представлен в листинге 5.29.

Листинг 5.29

```
int outPin = 8;         // цифровой порт входа/выхода 8
void setup()
 {
 pinMode(outPin, OUTPUT);   // устанавливается режим порта – выход
 }
void loop()
 {
 digitalWrite(outPin, HIGH); // подаем HIGH на выход
 delay(50);           // ожидаем 50 микросекунд
 digitalWrite(outPin, LOW);  // устанавливаем LOW на выходе
 delay(50);           // ожидаем 50 микросекунд
 }
```

5.5.5. Математические функции

В языке представлены следующие математические функции:

- `min()`;
- `max()`;
- `abs()`;
- `constrain()`;
- `map()`;
- `pow()`;
- `sq()`;
- `sqrt()`.

5.5.5.1. Функция *min(x,yx)*

Возвращает наименьшее из двух значений.

Параметры:

- `x` — первое число, любой тип;
- `y` — второе число, любой тип.

Возвращаемое значение — возвращает меньшее из двух сравниваемых значений.

Пример использования функции:

```
sensVal = min(sensVal, 100);
// проверяем, если sensVal больше 100, то senseVal будет присвоено 100
```

5.5.5.2. Функция *max(x, y)*

Возвращает большее из двух значений.

Параметры:

- `x` — первое число, любой тип;
- `y` — второе число, любой тип.

Возвращаемое значение — возвращает большее из двух сравниваемых значений.

Пример использования функции:

```
sensVal = max(sensVal, 20);
// проверяем, если sensVal меньше 20, то senseVal будет присвоено 20
```

Функция `max()` зачастую используется для ограничения нижней границы значений переменной. Функцией `min()` ограничивают верхнюю границу переменной. В силу специфики реализации функции `max()` следует избегать использования других функций в качестве параметров. Например:

```
max(a--, 0); // может привести к некорректным результатам
a--;
max(a, 0);  // так корректно
```

5.5.5.3. Функция *abs()*

Возвращает модуль числа.

Параметр: `x` — число.

Возвращаемые значения:

- `x` — если `x` больше или равен 0;
- `-x` — если `x` меньше 0.

В силу специфики реализации функции `abs()` следует избегать использования других функций в качестве параметров:

```
abs (a++); // может привести к некорректным результатам
a++;
abs(a, 0);  // так корректно
```

5.5.5.4. Функция *constrain(x, a, b)*

Функция проверяет и, если надо, задает новое значение так, чтобы оно было в области допустимых значений, заданной параметрами.

Параметры:

- `x` — проверяемое значение, любой тип;
- `a` — нижняя граница области допустимых значений, любой тип;
- `b` — верхняя граница области допустимых значений, любой тип.

Возвращаемое значение:

- `x` — если `x` входит в область допустимых значений `[a..b]`;
- `a` — если `x` меньше `a`;
- `b` — если `x` больше `b`.

Пример:

```
sensVal = constrain(sensVal, 10, 150);
// ограничиваем значения sensVal диапазоном от 10 до 150
```

5.5.5.5. Функция *map(value, fromLow, fromHigh, toLow, toHigh)*

Функция пропорционально переносит значение (`value`) из текущего диапазона значений (`fromLow .. fromHigh`) в новый диапазон (`toLow .. toHigh`), заданный параметрами.

Функция `map()` не ограничивает значение рамками диапазона, как это делает функция `constrain()`. Функция `contsrain()` может быть использована до или после вызова `map()`, если необходимо ограничить допустимые значения заданным диапазоном.

Обратите внимание, что "нижняя граница" может быть как меньше, так и больше "верхней границы". Это может быть использовано, чтобы "перевернуть" диапазон:

```
y = map(x, 1, 50, 50, 1);
```

Возможно использование отрицательных значений:

```
y = map(x, 1, 50, 50, -100);
```

Функция `map()` оперирует целыми числами. При пропорциональном переносе дробная часть не округляется по правилам, а просто отбрасывается.

Параметры:

- `value` — значение для переноса;
- `fromLow` — нижняя граница текущего диапазона;
- `fromHigh` — верхняя граница текущего диапазона;
- `toLow` — нижняя граница нового диапазона, в который переносится значение;
- `toHigh` — верхняя граница нового диапазона.

Возвращаемое значение — значение в новом диапазоне.

Пример использования функции представлен в листинге 5.30.

Листинг 5.30

```
// Переносим значение с аналогового входа
// (возможные значения от 0 до 1023) в 8 бит (0..255)
void setup()
 {;}
void loop()
 {
 int val = analogRead(0);
 val = map(val, 0, 1023, 0, 255);
 analogWrite(9, val);
 }
```

5.5.5.6. Функция *pow(base, exponent)*

Вычисляет значение, возведенное в заданную степень. Функция `pow()` может возводить и в дробную степень.

Параметры:

- `base` — число (тип `float`);
- `exponent` — степень, в которую будет возводиться число (тип `float`).

Возвращаемое значение — результат возведения в степень, число (тип `double`).

5.5.5.7. Функция *sq(x)*

Функция возвращает квадрат числа, заданного параметром.

Параметр: `x` — число, любой тип.

Возвращаемое значение — квадрат числа.

5.5.5.8. Функция *sqrt(x)*

Функция вычисляет квадратный корень числа, заданного параметром.

Параметры: `x` — число, любой тип.

Возвращаемое значение — квадратный корень числа (тип `double`).

5.5.6. Тригонометрические функции

В языке представлены следующие тригонометрические функции:

- `sin()`;
- `cos()`;
- `tan()`.

5.5.6.1. Функция *sin(rad)*

Возвращает синус угла, заданного в радианах в передаваемом параметре. Результат функции всегда в диапазоне –1 .. 1.

Параметр: `rad` — угол в радианах (`float`).

Возвращаемое значение: синус угла (тип `double`).

5.5.6.2. Функция *cos(rad)*

Возвращает косинус угла, заданного в радианах в передаваемом параметре. Результат функции всегда находится в диапазоне –1 .. 1.

Параметр: `rad` — угол в радианах (тип `float`).

Возвращаемое значение: косинус угла (тип `double`).

5.5.6.3. Функция *tan(rad)*

Возвращает тангенс угла, заданного в радианах в передаваемом параметре. Результат функции в диапазоне от минус бесконечности до плюс бесконечности.

Параметр: `rad` — угол в радианах (тип `float`).

Возвращаемое значение: тангенс угла (тип `double`).

5.5.7. Генераторы случайных значений

Функции формирования случайных чисел:

- `randomSeed()`;
- `random()`.

5.5.7.1. Функция *randomSeed(seed)*

Функция `randomSeed()` инициализирует генератор псевдослучайных чисел. Генерируемая последовательность случайных чисел очень длинная, и всегда одна и та же. Точка в этой последовательности, с которой начинается генерация чисел, зависит от параметра `seed`.

Параметр: `seed` — параметр, задающий начало выдачи псевдослучайных значений на последовательности (тип `int`, `long`).

5.5.7.2. Функция *random()*

Функция `random()` возвращает псевдослучайное число.

Синтаксис:

```
random(max);
random(min, max);
```

Параметры:

- `min` — нижняя граница случайных значений, включительно (опционально);
- `max` — верхняя граница случайных значений, включительно.

Возвращаемое значение: случайное число между `min` и `max` – 1 (тип `long`).

Если при каждом запуске программы необходимо получать разные последовательности значений, генерируемых функцией `random()`, то необходимо инициализировать генератор псевдослучайных чисел со случайным параметром. Например, можно использовать значение, отдаваемое функцией `analogRead()` с неподключенного порта входа/выхода. В некоторых случаях необходимо получать одинаковую последовательность при каждом запуске программы на Arduino. Тогда инициализировать генератор псевдослучайных чисел следует вызовом функции `randomSeed()` с фиксированным параметром.

Пример использования функции представлен в листинге 5.31.

Листинг 5.31

```
long randNumber;
void setup()
 {
 Serial.begin(9600);
 }
void loop()
 {
 // выводим случайное число из диапазона 0..299
 randNumber = random(300);
 Serial.println(randNumber);
 // выводим случайное число из диапазона 0..19
 randNumber = random(10, 20);
 Serial.println(randNumber);
 delay(50);
 }
```

5.5.8. Операции с битами и байтами

Функции — операции с битами и байтами:

- `lowByte()`;
- `highByte()`;
- `bitRead()`;
- `bitWrite()`;
- `bitSet()`;
- `bitClear()`;
- `bit()`.

5.5.8.1. Функция *lowByte()*

Извлекает младший (самый правый) байт переменной (например, типа `word`).

Синтаксис:

```
lowByte(x);
```

Параметр: `x` — величина любого типа.

Возвращает байт.

5.5.8.2. Функция *highByte()*

Извлекает старший (крайний левый) байт слова (или второй младший байт большего типа данных).

Синтаксис:

```
highByte(x);
```

Параметр: `x` — величина любого типа.

Возвращает байт.

5.5.8.3. Функция *bitRead()*

Читает определенный бит переменной.

Синтаксис:

```
bitRead(x, n);
```

Параметры:

- `x` — число, из которого необходимо прочитать;
- `n` — указывает бит, который необходимо прочитать, начиная с 0 для младшего (правого) бита.

Возвращает: значение бита (0 или 1).

5.5.8.4. Функция *bitWrite()*

Записывает бит числовой переменной.

Синтаксис:

```
bitWrite(x, n, b);
```

Параметры:

- `x` — числовая переменная, в которую необходимо записать;
- `n` — номер бита, который необходимо записать, начиная с 0 для младшего (левого) бита;
- `b` — значение, которое необходимо записать в бит (0 или 1).

5.5.8.5. Функция *bitSet()*

Устанавливает (записывает 1) бит числовой переменной.

Синтаксис:

```
bitSet(x, n)
```

Параметры:

- `x` — числовая переменная, которую необходимо записать;
- `n` — номер бита, который необходимо установить, начиная с 0 для младшего (левого) бита.

5.5.8.6. Функция *bitClear()*

Сбрасывает (записывает 0) бит числовой переменной.

Синтаксис:

```
bitClear(x, n);
```

Параметры:

- `x` — числовая переменная, которую необходимо записать;
- `n` — номер бита, который необходимо установить, начиная с 0 для младшего (левого) бита.

5.5.8.7. Функция *bit()*

Вычисляет значение указанного бита (бит 0 — это 1, бит 1 — это 2, бит 2 — это 4 и т. д.).

Синтаксис:

```
bit(n);
```

Параметр: `n` — номер бита, который необходимо вычислить.

Возвращает: значение бита.

5.5.9. Внешние прерывания

Прерывание (англ. *interrupt*) — сигнал, сообщающий процессору о наступлении какого-либо события. При этом выполнение текущей последовательности команд приостанавливается, и управление передается обработчику прерывания, который выполняет работу по обработке события и возвращает управление в прерванный код. Arduino так же предоставляет свои функции для работы с прерываниями.

Их всего две:

- `attachInterrupt()`;
- `detachInterrupt()`.

5.5.9.1. Функция *attachInterrupt*

Задает функцию обработки внешнего прерывания, т. е. функцию, которая будет вызвана по внешнему прерыванию. Если до этого была задана другая функция, то назначается новая. Вычисляет значение указанного бита (бит 0 — это 1, бит 1 — это 2, бит 2 — это 4 и т. д.).

Синтаксис:

```
attachInterrupt(interrupt, function, mode);
```

Параметры:

- `interrupt` — номер прерывания:
  - `0` — на цифровом порту 2;
  - `1` — на цифровом порту 3;
  - `2` — на цифровом порту 21 (для Arduino Mega);
  - `3` — на цифровом порту 21 (для Arduino Mega);
  - `4` — на цифровом порту 21 (для Arduino Mega);
  - `5` — на цифровом порту 21 (для Arduino Mega);
- `function` — функция, вызываемая прерыванием (должна быть без параметров и не возвращать значений);
- `mode` — задает режим обработки прерывания, допустимо использование следующих констанст:
  - `LOW` — вызывает прерывание, когда на порту `LOW`;
  - `CHANGE` — прерывание вызывается при смене значения на порту с `LOW` на `HIGH` и наоборот;
  - `RISING` — прерывание вызывается только при смене значения на порту с `LOW` на `HIGH`;
  - `FALLING` — прерывание вызывается только при смене значения на порту с `HIGH` на `LOW`.

Возвращаемого значения нет.

Внутри функции обработки прерывания не работает функция `delay()`, значения, возвращаемые функцией `millis()`, не изменяются. Возможна потеря данных, передаваемых по последовательному соединению (Serial data) в момент выполнения функции обработки прерывания. Переменные, изменяемые в функции, должны быть объявлены как `volatile`.

5.5.9.2. Функция *detachInterrupt*

Выключает обработку внешнего прерывания.

Синтаксис:

```
detachInterrupt(interrupt);
```

Параметр: `interrupt` — номер прерывания (0 или 1), для Arduino Mega еще 2, 3, 4 или 5.

Возвращаемого значения нет.

В листинге 5.32 приведен пример использования прерывания 0 при наступлении события `CHANGE` на порту 2. При этом светодиод на выводе 13 Arduino при каждом прерывании меняет статус (горит либо гаснет).

Листинг 5.32

```
int pin = 13;
volatile int state = LOW;

void setup()
{
 pinMode(pin, OUTPUT);
 attachInterrupt(0, blink, CHANGE);
}

void loop()
{
 digitalWrite(pin, state);
}
// функция обработки прерывания
void blink()
{
 state = !state;
}
```

Теперь можно переходить к изучению Arduino на практических примерах. Этому будут посвящены *главы 6–30*.

ЧАСТЬ III

Практическое применение Arduino

ГЛАВА 6

Arduino и набор функций *Serial*

Набор функций `Serial` служит для связи устройства Arduino с компьютером или другими устройствами, поддерживающими последовательный интерфейс обмена данными. Все платы Arduino имеют хотя бы один последовательный порт UART. Для обмена данными `Serial` служат цифровые порты ввода/вывода 0 (RX) и 1 (TX), а также порт USB. Важно учитывать, что если вы используете функции `Serial`, то нельзя одновременно с этим задействовать порты 0 и 1 для других целей.

Среда разработки Arduino имеет встроенный монитор последовательного интерфейса (Serial monitor), показанный на рис. 6.1. Для начала обмена данными необходимо запустить монитор нажатием кнопки **Serial monitor** и выставить ту же скорость связи (`baud rate`), с которой вызвана функция `begin()`.

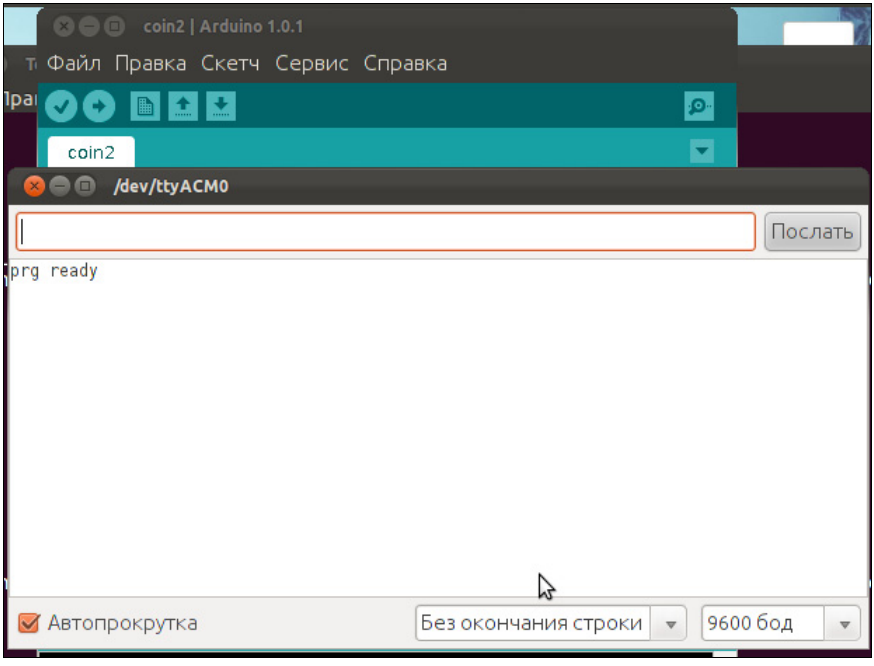


Рис. 6.1. Монитор последовательного порта

Плата Arduino Mega имеет три дополнительных последовательных порта: `Serial1` на портах 19 (RX) и 18 (TX), `Serial2` на портах 17 (RX) и 16 (TX), `Serial3` на портах 15 (RX) и 14 (TX). Чтобы использовать эти порты для связи с компьютером, понадобятся дополнительные адаптеры USB-to-serial, т. к. они не подключены к встроенному адаптеру платы Mega.

6.1. Функции библиотеки *Serial*

6.1.1. Функция *Serial.begin()*

Функция `Serial.begin()` инициирует последовательное соединение и задает скорость передачи данных в бит/с (бод). Для обмена данными с компьютером используйте следующие значения: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 или 115200. При соединении через порты входа/выхода 0 и 1 могут быть использованы другие значения скорости, требуемые устройством, с которым будет осуществляться обмен данными.

Синтаксис функции `Serial.begin()`:

```
Serial.begin(speed)
```

Для Arduino Mega и Arduino Due:

```
Serial1.begin(speed)
Serial2.begin(speed)
Serial3.begin(speed)
```

Параметр: `speed` — скорость в бит/с (бод) (`long`).

Возвращаемого значения нет.

6.1.2. Функция *Serial.end()*

Функция `Serial.end()` закрывает последовательное соединение, порты RX и TX освобождаются и могут быть использованы для ввода/вывода.

Синтаксис функции `Serial.end()`:

```
Serial.end()
```

Для Arduino Mega и Arduino Due:

```
Serial1.end()
Serial2.end()
Serial3.end()
```

Параметра нет.

Возвращаемого значения нет.

6.1.3. Функция *Serial.available()*

Функция `Serial.available()` получает количество байтов (символов), доступных для чтения из последовательного интерфейса связи. Это те байты, которые уже по-

ступили и записаны в буфер последовательного порта. Буфер может хранить до 128 байтов.

Синтаксис функции `Serial.available()`:

```
Serial. available()
```

Для Arduino Mega и Arduino Due:

```
Serial1.available()
Serial2.available()
Serial3.available()
```

Параметра нет.

Возвращаемое значение — количество байтов, доступных для чтения (`int`).

6.1.4. Функция *Serial.read()*

Функция `Serial.read()` считывает очередной доступный байт из буфера последовательного соединения.

Синтаксис функции `Serial.read()`:

```
Serial.read()
```

Для Arduino Mega и Arduino Due:

```
Serial1.read()
Serial2.read()
Serial3.read()
```

Параметра нет.

Возвращаемое значение — следующий доступный байт или `-1`, если его нет (`int`).

Примеры использования функций `Serial.available()` и `Serial.read()` представлены в листинге 6.1.

Листинг 6.1

```
int in=0;   // переменная для хранения полученного байта

void setup() {
    Serial.begin(9600); // устанавливаем последовательное соединение
    }

void loop()
   {
    if (Serial.available() > 0) // если есть доступные данные
      {
       // считываем байт
       in = Serial.read();
```

```
      // отсылаем то, что получили
      Serial.println(in,DEC);
     }
  }
```

6.1.5. Функция *Serial.flush()*

Функция `Serial.flush()` ожидает окончания передачи исходящих данных (до версии Arduino 1.0 функция очищала буфер последовательного соединения).

Синтаксис функции `Serial.flush()`:

```
Serial.flush()
```

Для Arduino Mega и Arduino Due:

```
Serial1.flush()
Serial2.flush()
Serial3.flush()
```

Параметра нет.

Возвращаемого значения нет.

6.1.6. Функция *Serial.print()*

Функция `Serial.print()` передает данные через последовательный порт как текст ASCII. Эта функция может принимать различные типы данных. С помощью второго опционального параметра можно задать базис (систему счисления) для чисел.

Синтаксис функции `Serial.print()`:

```
Serial.print(val)
Serial.print(val, format)
```

Для Arduino Mega и Arduino Due: `Serial1`, `Serial2`, `Serial3`.

Параметры:

- `val` — данные для передачи через последовательное соединение;
- `format` — базис для целых чисел или количество знаков после запятой для вещественных:
  - `BYTE`;
  - `BIN` (двоичный);
  - `OCT` (восьмеричный);
  - `DEC` (десятеричный);
  - `HEX` (шестнадцатеричный);
  - для вещественных (дробных) чисел второй параметр задает количество знаков после запятой.

Возвращаемого значения нет.

6.1.7. Функция *Serial.println()*

Функция `Serial.println()` передает данные через последовательный порт как текст ASCII вместе со следующим за ним символом переноса строки (символ ASCII 13 или '`\r`') и символом новой строки (ASCII 10 или '`\n`'). Параметры и типы данных для этой функции такие же, как и для функции `Serial.print()`.

6.1.8. Функция *Serial.write()*

Функция `Serial.write()` передает данные как бинарный код через последовательное соединение. Данные посылаются единичным байтом или серией байтов. Для того, чтобы передать данные как символы, следует использовать другую функцию — `print()`.

Синтаксис функции `Serial.write()`:

```
Serial.write(val)
Serial.write(str)
Serial.print(buf, len)
```

Для Arduino Mega: `Serial1`, `Serial2`, `Serial3`.

Параметры:

- `val` — один байт;
- `str` — строка как серия байтов;
- `buf` — массив байтов;
- `len` — длина массива.

Возвращаемого значения нет.

6.1.9. Функция *Serial.peek()*

Функция `Serial.peek()` возвращает следующий доступный байт (символ) из буфера входящего последовательно соединения, не удаляя его из этого буфера. То есть успешный вызов этой фунции вернет то же значение, что и следующий за ним вызов функции `read()`.

Синтаксис функции `Serial.peek()`:

```
Serial.peek()
```

Для Arduino Mega и Arduino Due:

```
Serial1.peek()
Serial2.peek()
Serial3.peek()
```

Параметра нет.

Возвращаемое значение — следующий доступный байт или `-1`, если его нет (`int`).

6.2. Библиотека *SoftwareSerial*

Библиотека `SoftwareSerial` позволяет осуществить последовательную передачу данных через другие цифровые контакты Arduino. Можно иметь несколько программных последовательных портов со скоростью до 115 200 битов. Программное обеспечение `SoftwareSerial` копирует функциональность библиотеки `Serial` (отсюда и название "SoftwareSerial"), но с некоторыми ограничениями:

- при использовании нескольких программных последовательных портов только один может принимать данные одновременно;
- на платах Mega и Mega 2560 для RX могут быть использованы только следующие контакты: 10, 11, 12, 13, 14, 15, 50, 51, 52, 53, A8 (62), A9 (63), A10 (64), A11 (65), A12 (66), A13 (67), A14 (68), A15 (69);
- на плате Leonardo для RX могут быть использованы только следующие контакты: 8, 9, 10, 11, 14 (MISO), 15 (SCK), 16 (MOSI).

Библиотека `SoftwareSerial` включает следующие функции:

- `SoftwareSerial()`;
- `available()`;
- `begin()`;
- `isListening()`;
- `overflow()`;
- `peek()`;
- `read()`;
- `print()`;
- `println()`;
- `listen()`;
- `write()`.

В целом эти функции аналогичны функциям библиотеки `Serial`, поэтому далее мы рассмотрим только функции, уникальные для библиотеки `SoftwareSerial`.

6.2.1. Функция *SoftwareSerial()*

Функция `SoftwareSerial()` создает экземпляр объекта последовательного порта `SoftwareSerial`. Для запуска последовательного порта на выводах rxPin, txPin необходимо выполнить функцию `SoftwareSerial.begin()`.

Синтаксис функции `SoftwareSerial()`:

```
SoftwareSerial(rxPin, txPin)
```

Параметры:

- `rxPin` — вывод для получения последовательных данных;
- `txPin` — вывод для отправки последовательных данных.

Возвращаемое значение — экземпляр объекта `SoftwareSerial`.

6.2.2. Функция *SoftwareSerial.listen()*

Функция `SoftwareSerial.listen()` назначает выбранный порт в качестве "слушателя" данных, при этом только один последовательный порт программное обеспече-

ние может слушать одновременно — данные, которые прибывают для других портов, будут отвергнуты. Любые данные, которые уже получены, во время вызова `listen()` отбрасываются (если данный экземпляр уже не слушает).

Синтаксис функции `SoftwareSerial.listen()`:

```
SoftwareSerial.listen(port)
```

Параметра нет.

Возвращаемого значения нет.

6.2.3. Функция *SoftwareSerial.isListening()*

Функция `SoftwareSerial.isListening()` проверяет, выступает ли выбранный порт в данный момент в качестве "слушателя" данных.

Синтаксис функции `SoftwareSerial.isListening()`:

```
SoftwareSerial.isListening()
```

Параметра нет.

Возвращаемое значение — `boolean`: `false` или `true`.

Примеры назначения порта слушателем и проверки его состояния представлены в листинге 6.2.

Листинг 6.2

```
#include <SoftwareSerial.h>
// software serial : TX = digital pin 10, RX = digital pin 11
SoftwareSerial portOne(10, 11);
// software serial : TX = digital pin 8, RX = digital pin 9
SoftwareSerial portTwo(8, 9);

void setup()
{
  Serial.begin(9600);
  //  Start both software serial ports
  portOne.begin(9600);
  portTwo.begin(9600);
}

void loop()
{
  portOne.listen();
  if (portOne.isListening())
     {Serial.println("Port One is listening!");}
  else
    {Serial.println("Port One is not listening!");}
```

```
  if (portTwo.isListening())
   {Serial.println("Port Two is listening!");}
  else
   {Serial.println("Port Two is not listening!");}
}
```

6.2.4. Функция *SoftwareSerial.overflow()*

Функция `SoftwareSerial.overflow()` проверяет, произошло ли переполнение буфера для данного экземпляра последовательного порта. Размер буфера 64 байта. Вызов этой функции очищает флаг переполнения.

Синтаксис функции `SoftwareSerial.overflow()`:

```
SoftwareSerial.overflow()
```

Параметра нет.

Возвращаемое значение — `boolean`: `false` или `true`.

ГЛАВА 7

Arduino и знакосинтезирующие жидкокристаллические индикаторы

Понятно, что использовать последовательный порт в качестве монитора вывода данных не совсем удобно, а для автономных проектов и невозможно. Разумно для этой цели, а также для вывода информации при отладке создаваемых проектов применить жидкокристаллический индикатор. Жидкокристаллические индикаторы (ЖКИ) различных фирм (POWERTIP, MICROTIPS, WINSTAR, Fordata, Anshan Yes, Data Vision, Newtec Sunlike) уверенно завоевывают мировой рынок электроники сегодняшнего дня и постепенно вытесняют старые добрые стрелочные приборы. Современному разработчику аппаратуры для физических экспериментов и промышленности необходимо знать основные принципы работы этих удобных и полезных устройств. Все жидкокристаллические индикаторы, выпускаемые в различных странах и в России в том числе, с точки зрения программиста однотипны. Но руководства, даваемые разработчиками, обычно написаны на английском языке, скупы и трудны для понимания. Тем не менее следует учесть, что в устройствах, выпускаемых в различных странах, наблюдается явная тенденция к стандартизации, и практически все выпускаемые ЖК-индикаторы ориентированы на применение контроллера HD44780.

Подберем недорогой, но в то же время надежный, символьный индикатор достаточно большого размера для вывода при отладке большого объема информации. В проектах Arduino частенько используют недорогие индикаторы фирмы WINSTAR Display Co., Ltd. — одного из крупнейших мировых разработчиков и производителей высококачественных символьных и графических (в т. ч. цветных) ЖК-индикаторов. Штаб-квартира компании находится на Тайване, а производственные мощности, расположенные на площадях более 3000 кв. метров, обеспечивают выпуск свыше миллиона различных индикаторов в год. Производство сертифицировано по системе ISO 9001.

Полагаю, индикатор WINSTAR 1604A-NGG-CT (рис. 7.1) — станет удачным выбором для намеченных нами задач. Количество выводимых символов 16×4 при цене около 300 руб. — вполне приемлемый вариант.

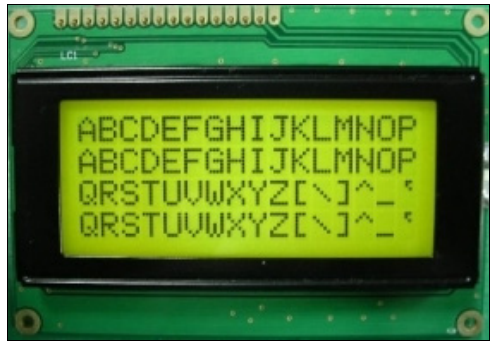


Рис. 7.1. ЖКИ WINSTAR 1604A-NGG-CT

7.1. Принцип работы модулей ЖКИ WINSTAR WH1604

Назначение выводов ЖКИ WINSTAR WH1604 представлено в табл 7.1.

Таблица 7.1. *Назначение выводов WH1604*

№ вывода	Название	Функция
1	Vss	Общий (GND)
2	Vdd	Напряжение питания (3 или 5 В)
3	Vo	Контрастность
4	RS	Команды/Данные
5	R/W	Чтение/Запись
6	E	Разрешение чтения/записи
7	DB0	Линия данных 0
8	DB1	Линия данных 1
9	DB2	Линия данных 2
10	DB3	Линия данных 3
11	DB4	Линия данных 4
12	DB5	Линия данных 5
13	DB6	Линия данных 6
14	DB7	Линия данных 7
15	A	Напряжение подсветки (+)
16	K	Напряжение подсветки (–)

Здесь 8 линий (DB0...DB7) — стандартная шина данных. Уровни на выводах DB0...DB7 — коды операций или данные. Имеются три управляющих линии (RS, R/W, E). Уровни на входах RS, R/W, E определяют режимы (табл. 7.2).

Контроллер ЖКИ может совместно с внешним контроллером выполнять различные действия: передавать данные по параллельному интерфейсу в обоих направлениях, менять направления заполнения знакомест ЖК-дисплея, манипулировать строками и т. д. После включения питания работа ЖК-модуля всегда начинается с процедуры его инициализации, установки которой сохраняются до выключения питания устройства. К концу процедуры инициализации модуль ЖКИ приводится в состояние готовности к работе с данными. Если инициализация ЖК-модуля прошла успешно, модуль готов к работе — для него задано число строк и размер знакоместа (где знакоместо — это матрица 5×7 или 5×10 точек).

Таблица 7.2. *Условия выполнения операций чтения/записи*

RS	R/W	E	Операция
0	0	1->0	Запись в регистр команд
1	0	1->0	Запись в регистр данных
1	1	0->1->0	Чтение из регистра данных
0	1	0->1->0	Чтение флага занятости (DB7) и счетчика адреса (DB0–DB6)

Выбор матрицы производится один раз в ходе процедуры инициализации ЖК-модуля, но можно сменить тип матрицы и в ходе работы. "Высвечивание" точки происходит при подаче на нее единицы, если на точку подан ноль, то она "погашена".

В обоих типах матриц имеется дополнительная (8-я или 11-я) строка точек — так называемая *курсорная*. При включении курсора (программно) все 5 точек этой строки "высвечиваются" под тем символом, который будет выведен в последующем цикле вывода на индикацию (курсор — элемент удобства и нужен не во всех устройствах). Соседние элементы матрицы отделены друг от друга зазором 0,7 мм. Чаще применима матрица 5×7 точек, выбор символов в этом случае больше, чем при использовании матрицы 5×10 точек.

Символы для отображения размещаются в области энергонезависимой памяти знакогенератора. Поскольку это ПЗУ, набор символов пользователь изменить не может. Однако кроме области энергонезависимой памяти знакогенератор имеет в своем составе область оперативной памяти CG RAM, куда можно записать "сконструированные" программистом символы. Разработчики оставляют программисту возможность создавать свои символы: до 8-ми символов для матрицы 5×7 точек в одной ячейке CG RAM или до 4-х символов для матрицы 5×10 точек (один символ — в двух ячейках CG RAM). Впрочем, количество "штатных" символов настолько велико и разнообразно, что их хватает с избытком.

Знакогенератор — это программируемый счетчик, управляющий работой обычной микросхемы ПЗУ. Числа в двоичном представлении знакогенератора выводятся из ячеек ПЗУ запрошенных адресов. Таблица знакогенератора разбита на ячейки, в которых записаны символы по указанным адресам.

Чтобы "запустить" режим записи данных, необходимо сначала выставить "1" на входе RS модуля ЖКИ (R/W — на корпусе), а после этого сформировать на входе E

модуля ЖКИ стробирующий перепад от "1" к "0", инициализирующий команду. После смены на выводе E "1" на "0" содержимое из заранее выбранного адреса ячейки знакогенератора выведется в текущее знакоместо видеопамяти ЖК-дисплея. Таблица адресов видеопамяти ЖК-дисплея представлена в табл. 7.3.

Таблица 7.3. *Таблица адресов видеопамяти ЖК-дисплея*

Знакоместо		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Адреса по строкам	**1**	00	01	02	03	04	05	06	07	08	09	1A	1B	1C	1D	1E	1F
		20	21	22	23	24	25	26	27								
	2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
										28	29	2A	2B	2C	2D	2E	2F
	3	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
		30	31	32	33	34	35	36	37								
	4	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
										38	39	3A	3B	3C	3D	3E	3F

К работе видеопамяти привязан курсор. Такая привязка при работе с подобного рода устройствами очень полезна. Курсор отмечает то знакоместо, в которое будет выведен на индикацию следующий символ. Текущее знакоместо видеопамяти ЖК-дисплея определяет счетчик AC — реверсивный счетчик с программной установкой автоинкремента и автодекремента и с предустановкой числа счета. Если же по ходу заполнения строки символами возникает необходимость в увеличении или уменьшении значения адреса, формируемого счетчиком адреса AC, то программно необходимо сформировать соответствующую команду. Инструкции по формированию команды изменения текущего положения представлены в табл. 7.4. В табл. 7.3 белым фоном выделены адреса ячеек видеопамяти, содержимое которых видимо на экране дисплея сразу (адреса в таблице показаны с привязкой к строкам ЖКИ), серым — ячейки, содержимое которых становится видимым на ЖКИ только после циклического сдвига. При этом адреса в области видеопамяти не сдвигаются. После исполнения команды сдвига сдвигается содержимое всех ячеек, перемещаясь из ячейки в ячейку.

Таблица 7.4. *Инструкции вывода символов и задания знакоместа на экране ЖКИ*

Инструкция	**Режим**		**Двоичный код инструкции**							
	RS	**RW**	**DB7**	**DB6**	**DB5**	**DB4**	**DB3**	**DB2**	**DB1**	**DB0**
Установка знакоместа ЖКИ дисплея	0	0	1	Адрес ячейки видеопамяти						
Выбор адреса выводимого из ПЗУ знакогенератора символа в видеопамять	1	0	Адрес ячейки знакогенератора							

Список команд управления ЖКИ представлен в табл. 7.5. В таблице не приведены команды, связанные с чтением из памяти контроллера ЖКИ, но их использование рационально при конструировании сложных фирменных систем индикации и вряд ли потребуется в относительно простых проектах с использованием Arduino.

Таблица 7.5. *Команды управления*

Инструкция	Режим		Двоичный код инструкции								16-р. код
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
Очистка дисплея, сброс данных, курсор в начало	0	0	0	0	0	0	0	0	0	1	01
Возврат курсора в начало без сброса данных	0	0	0	0	0	0	0	0	1	x	02
Установка направления сдвига курсора I/D дисплея SH	0	0	0	0	0	0	0	1	I/D	SH	04 05 06 07
Дисплей вкл/откл D = (1/0) Изображение курсора B в виде подчеркивания C или мерцающего знакоместа	0	0	0	0	0	0	1	D	C	B	08 09 0A 0B 0C 0D 0E 0F
Сдвиг курсора или дисплея вправо или влево Сдвиг дисплея и курсора S/C = 1 Сдвиг только курсора S/C = 0 R/L = 1 вправо R/L = 0 влево	0	0	0	0	0	1	S/C	R/L	x	x	10 18 14 1C
Установка разрядности интерфейса 8-DL = 1, 4-DL = 0 тип матрицы F = 0 — 5×7 F = 0 — 5×10 Число строк N = 1 4 строки	0	0	0	0	1	DL	N	F	x	x	40 60

Рассмотрим более детально работу этих команд и флаговых битов.

- **Очистка дисплея со сбросом данных** — это принудительная запись во все ячейки видеопамяти символа "пробел" (ячейка с адресом 20h таблицы знакогенератора).
- **Возврат курсора в начало без сброса данных** — означает, что происходит сброс счетчика AC в 0, курсор устанавливается в крайнее левое знакоместо первой строки и ждет вывода символа.
- **Установка направлений сдвигов курсора (I/D) и дисплея (S)** — это касается сдвига при выводе текущего символа на индикацию. I/D, S — управляющие флаги:
  - I/D — флаг реверса счетчика адреса (при инкременте курсор сдвигается вправо на одну позицию, а при декременте — влево на одну позицию):
    - I/D = "0" — декремент счетчика адреса AC;
    - I/D = "1" — инкремент;
  - S — флаг сдвига всего дисплея:
    - S = "0" — сдвиг дисплея не производится;
    - S = "1" — производится сдвиг всего дисплея на одну позицию влево (при I/D = 1) или вправо (при I/D = 0).
- **Включение/выключение изображения курсора в виде подчерка или мерцающего знакоместа** — в состав этой команды входят 3 флага:
  - D — вкл./выкл. изображения (дисплея):
    - D = "1" — изображение включено (основной режим);
    - D = "0" — изображение выключено, однако выключается только изображение, и в этом случае можно производить какие угодно действия, а затем, в нужный момент, включить изображение;
  - C — вкл./выкл. визуального отображения курсора в виде подчерка:
    - C = "1" — курсор в виде подчерка включен;
    - C = "0" — курсор в виде подчерка выключен;
  - B — вкл./выкл. визуального отображения курсора в виде мерцающего знакоместа:
    - B = "1" — курсор в виде мерцающего знакоместа включен;
    - B = "0" — курсор в виде мерцающего знакоместа выключен.

  Для выключения визуального отображения курсора (при D = 1) необходимо в битах C и B "выставить" нули.
- **Сдвиг курсора или дисплея на одну позицию вправо или влево**:
  - S/C — флаг указателя объекта сдвига:
    - S/C = "1" — сдвигается дисплей вместе с курсором в направлении, указанном флагом R/L;

- S/C = "0" — сдвигается курсор (без сдвига дисплея) в направлении, указанном флагом R/L;

- R/L — флаг указателя направления сдвига:
  - R/L = "1" — сдвиг объекта сдвига (определяется флагом S/C) на одно знакоместо вправо;
  - R/L = "0" — сдвиг объекта сдвига (определяется флагом S/C) на одно знакоместо влево.

❒ **Установка разрядности интерфейса, типа матрицы и количества активных строк**:

- DL — флаг, определяющий разрядность параллельного интерфейса, соединяющего контроллер с ЖК-модулем в части касающейся линий DB0...DB7:
  - DL = 1 — 8 разрядов;
  - DL = "0" — 4 разряда.
- N — флаг, определяющий количество активных строк:
  - **N** = "1" — работают 4 строки;
  - **N** = "0" — работают 2 строки.
- F — флаг, определяющий тип матрицы:
  - F = "1" — 5×10 точек;
  - F ="0" — 5×7 точек.

Необходимая длительность временных задержек (не меньше заданного техническими условиями) установки байта данных или команды на линиях DB0...DB7 по отношению к моменту активного перепада на выводе E обеспечивается программными средствами.

7.2. Библиотека *LiquidCrystal*

Управлять ЖК-индикаторами, работающими на контроллере Hitachi HD44780 (или совместимом чипсете), который находится в большинстве символьных жидкокристаллических индикаторов, позволяет библиотека `LiquidCrystal`, входящая в стандартный комплект поставки Arduino. Библиотека `LiquidCrystal` работает в 4-х или в 8-разрядном режиме (т. е. использует 4 или 8 линий данных в дополнение к линиям управления RS, ENABLE и опционально RW). В библиотеке реализованы следующие функции:

❒ `LiquidCrystal();`
❒ `begin();`
❒ `clear();`
❒ `home();`
❒ `setCursor();`
❒ `write();`
❒ `print();`
❒ `cursor();`
❒ `noCursor();`
❒ `blink();`

- `noBlink();`
- `display();`
- `noDisplay();`
- `scrollDisplayLeft();`
- `scrollDisplayRight();`
- `autoscroll();`
- `noAutoscroll();`
- `leftToRight();`
- `rightToLeft();`
- `createChar()`.

Рассмотрим все эти функции подробно.

7.2.1. Функция *LiquidCrystal()*

Функция `LiquidCrystal()` создает переменную типа `LiquidCrystal`. Индикатором можно управлять, используя 4 или 8 линий данных. Для подключения по 4-м линиям пропустите контакты от D0 до D3 и оставьте эти линии неподключенными. Контакт RW может быть соединен с "землей" вместо соединения с контактом платы Arduino. Если в вашей схеме так — пропустите этот параметр функции.

Синтаксис функции `LiquidCrystal()`:

```
LiquidCrystal(rs, enable, d4, d5, d6, d7)
LiquidCrystal(rs, rw, enable, d4, d5, d6, d7)
LiquidCrystal(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)
LiquidCrystal(rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)
```

Параметры:

- `rs` — номер контакта платы Arduino, который соединяется с контактом RS жидкокристаллического индикатора;
- `rw` — номер контакта платы Arduino, который соединяется с контактом RW жидкокристаллического индикатора (опционально);
- `enable` — номер контакта платы Arduino, который соединяется с контактом ENABLE жидкокристаллического индикатора;
- `d0`, `d1`, `d2`, `d3`, `d4`, `d5`, `d6`, `d7` — номера контактов платы Arduino, которые соединяются соответственно с контактами данных жидкокристаллического индикатора. D0, D1, D2 И D3 опциональны — если они не задействованы, жидкокристаллический индикатор будет управляться только через 4 линии данных (D4, D5, D6, D7).

Пример использования функции `LiquidCrystal()`:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 10, 5, 4, 3, 2);
void setup()
{lcd.print("hello, world!");}
void loop() {;}
```

7.2.2. Функция *begin()*

Функция `begin()` определяет размерность (количество символов в ширину и высоту) индикатора.

Синтаксис функции `begin()`:

```
lcd.begin(cols, rows)
```

Параметры:

- `lcd` — переменная типа `LiquidCrystal`;
- `cols` — количество символов в строке;
- `rows` — количество строк.

7.2.3. Функция *clear()*

Функция `clear()` очищает экран жидкокристаллического индикатора и располагает курсор в верхнем левом углу.

Синтаксис функции `clear()`:

```
lcd.clear();
```

Параметр: `lcd` — переменная типа `LiquidCrystal`.

7.2.4. Функция *home()*

Функция `home()` располагает курсор в верхнем левом углу жидкокристаллического индикатора. Используется для определения начального положения при выводе последовательного текста на экран индикатора. Чтобы еще и очистить экран индикатора, используйте вместо этой функции функцию `clear()`.

Синтаксис функции `home()`:

```
lcd.home()
```

Параметр: `lcd` — переменная типа `LiquidCrystal`.

7.2.5. Функция *setCursor()*

Функция `setCursor()` позиционирует курсор жидкокристаллического индикатора, т. е. устанавливает положение, в котором на его экран будет выведен последующий текст.

Синтаксис функции `setCursor()`:

```
lcd.setCursor(col, row)
```

Параметры:

- `lcd` — переменная типа `LiquidCrystal`;
- `col` — номер знакоместа в ряду (начиная с 0 для первого знакоместа);
- `row` — номер ряда (начиная с 0 для первого ряда).

7.2.6. Функция *write()*

Функция `write()` записывает символ в жидкокристаллический индикатор.

Синтаксис функции `write()`:

```
lcd.write(data)
```

Параметры:

- `lcd` — переменная типа `LiquidCrystal`;
- `data` — символ, записываемый в индикатор.

Пример использования функции `write()`:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 10, 5, 4, 3, 2);
void setup()
{Serial.begin(9600);}
void loop()
{
if (Serial.available())
 lcd.write(Serial.read());
}
```

7.2.7. Функция *print()*

Функция `print()` печатает текст на жидкокристаллическом индикаторе.

Синтаксис функции `print()`:

```
lcd.print(data)
lcd.print(data, BASE)
```

Параметры:

- `lcd` — переменная типа `LiquidCrystal`;
- `data` — данные для печати (тип `char`, `byte`, `int`, `long` или `string`);
- `BASE` (опционально) — основание, по которому печатаются числа: `BIN` для двоичных (основание 2), `DEC` для десятичных (основание 10), `OCT` для восьмеричных (основание 8), `HEX` для шестнадцатеричных (основание 16).

Пример использования функции `print()`:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 10, 5, 4, 3, 2);
void setup()

{lcd.print("hello, world!");}
void loop() {;}
```

7.2.8. Функция *cursor()*

Функция `cursor()` выводит на экран жидкокристаллического индикатора курсор — подчеркивание знакоместа в позиции, в которую будет записан следующий символ.

Синтаксис функции `cursor()`:

```
lcd.cursor()
```

Параметр: `lcd` — переменная типа `LiquidCrystal`.

7.2.9. Функция *noCursor()*

Функция `noCursor()` скрывает курсор с экрана жидкокристаллического индикатора.

Синтаксис функции `noCursor()`:

```
lcd.noCursor()
```

Параметр: `lcd` — переменная типа `LiquidCrystal`.

7.2.10. Функция *blink()*

Функция `blink()` выводит на экран жидкокристаллического индикатора мигающий курсор. Если она используется в комбинации с функцией `cursor()`, результат будет зависеть от конкретного индикатора.

Синтаксис функции `blink()`:

```
lcd.blink()
```

Параметр: `lcd` — переменная типа `LiquidCrystal`.

7.2.11. Функция *noBlink()*

Функция `noBlink()` выключает мигающий курсор на экране жидкокристаллического индикатора.

Синтаксис функции `noBlink()`:

```
lcd.noBlink()
```

Параметр: `lcd` — переменная типа `LiquidCrystal`.

7.2.12. Функция *display()*

Функция `display()` включает жидкокристаллический индикатор после того, как он был выключен функцией `noDisplay()`. Эта функция восстанавливает текст (и курсор), который был на дисплее.

Синтаксис функции `display()`:

```
lcd.display()
```

Параметр: `lcd` — переменная типа `LiquidCrystal`.

7.2.13. Функция *noDisplay()*

Функция `noDisplay()` выключает жидкокристаллический индикатор без потери отображаемой на нем информации.

Синтаксис функции `noDisplay()`:

```
lcd.noDisplay()
```

Параметр: `lcd` — переменная типа `LiquidCrystal`.

7.2.14. Функция *scrollDisplayLeft()*

Функция `scrollDisplayLeft()` прокручивает информацию на экране индикатора (текст и курсор) на одно знакоместо влево.

Синтаксис функции `scrollDisplayLeft()`:

```
lcd.scrollDisplayLeft()
```

Параметр: `lcd` — переменная типа `LiquidCrystal`.

7.2.15. Функция *scrollDisplayRight()*

Функция `scrollDisplayRight()` прокручивает информацию на экране индикатора (текст и курсор) на одно знакоместо вправо.

Синтаксис функции `scrollDisplayRight()`:

```
lcd.scrollDisplayRight()
```

Параметр: `lcd` — переменная типа `LiquidCrystal`.

7.2.16. Функция *autoscroll()*

Функция `autoscroll()` включает автоматическую прокрутку экрана жидкокристаллического индикатора и заставляет каждый вывод символа на экран индикатора перемещать предыдущие символы на одно знакоместо. Если текущее направление вывода символов слева направо (значение по умолчанию) — экран индикатора прокручивается влево; если текущее направление вывода символов справа налево — экран индикатора прокручивается вправо. Это производит эффект вывода каждого нового символа в одно и то же знакоместо на экране жидкокристаллического индикатора.

Синтаксис функции `autoscroll()`:

```
lcd.autoscroll()
```

Параметр: `lcd` — переменная типа `LiquidCrystal`.

7.2.17. Функция *noAutoscroll()*

Функция `noAutoscroll()` выключает автоматическую прокрутку экрана жидкокристаллического индикатора.

Синтаксис функции `noAutoscroll()`:

```
lcd.noAutoscroll()
```

Параметр: `lcd` — переменная типа `LiquidCrystal`.

7.2.18. Функция *leftToRight()*

Функция `leftToRight()` устанавливает направление вывода символов на экран жидкокристаллического индикатора слева направо (значение по умолчанию). Это означает, что последующие символы, выводимые на экран индикатора, пойдут слева направо, но не повлияют на выведенный ранее текст.

Синтаксис функции `leftToRight()`:

```
lcd.leftTorRight();
```

Параметр: `lcd` — переменная типа `LiquidCrystal`.

7.2.19. Функция *rightToLeft()*

Функция `rightToLeft()` устанавливает направление вывода символов на экран жидкокристаллического индикатора справа налево (значение по умолчанию — слева направо). Это означает, что последующие символы, выводимые на экран индикатора, пойдут справа налево, но не повлияют на выведенный ранее текст.

Синтаксис функции `rightToLeft()`:

```
lcd.rightToLeft()
```

Параметр: `lcd` — переменная типа `LiquidCrystal`.

7.2.20. Функция *createChar()*

Функция `createChar()` создает пользовательский символ (глиф) для использования на жидкокристаллическом дисплее. Поддерживаются до восьми символов 5×8 пикселов (нумерация с 0 до 7). Создание каждого пользовательского символа определяется массивом из восьми байтов — один байт для каждой строки. Пять младших значащих битов каждого байта определяют пикселы в этой строке. Чтобы вывести пользовательский символ на экран, используйте функцию `write()` с номером символа в качестве параметра.

Синтаксис функции `createChar()`:

```
lcd.createChar(num, data)
```

Параметры:

- `lcd` — переменная типа `LiquidCrystal`;
- `num` — номер создаваемого символа (0 to 7);
- `data` — данные символьных пикселов.

Пример создания символа приведен в листинге 7.1.

Листинг 7.1

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
byte smiley[8]={
 B00000,
 B10001,
 B00000,
 B00000,
 B10001,
 B01110,
 B00000,
 };
void setup()
 {
 lcd.createChar(0, smiley);
 lcd.begin(16, 2);
 lcd.write(0);
 }
void loop() {;}
```

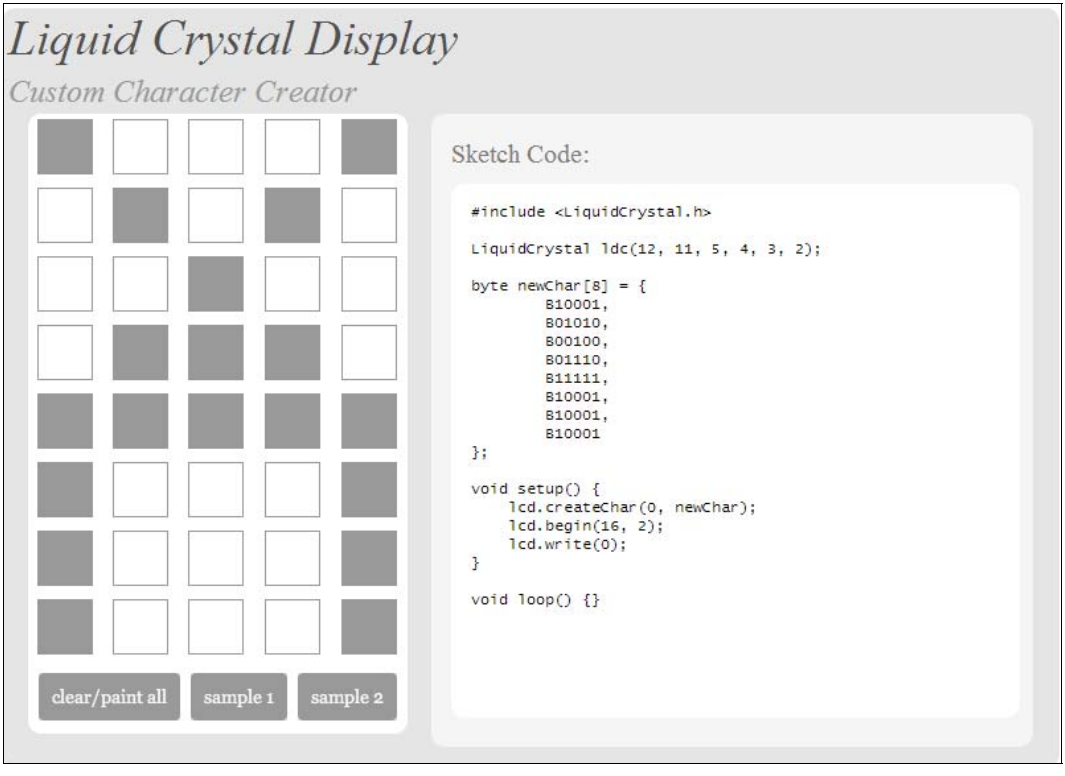


Рис. 7.2. Создание кода пользовательских символов для `LiquidCrystal` с помощью интернет-ресурса LiquidCrystal Display

С помощью интернет-ресурса **http://mikeyancey.com/hamcalc/lcd_characters.php** в режиме online можно нарисовать пользовательские символы и получить соответствующий им программный код (рис. 7.2).

7.3. Библиотека *LiquidCrystalRus*

Дисплеям WINSTAR присуща определенная проблема с поддержкой кириллицы. Дело в том, что китайцы решили, будто включить поддержку кириллицы — это значит произвольным образом раскидать по таблице знаков кириллические символы без соответствия какой-либо кодировке. В 2010 году Илья Данилов написал библиотеку `LiquidCrystalRus`, которая умело справляется с китайской кириллицей, делая ее поддержку прозрачной. Создадим пример, показывающий возможности библиотеки `LiquidCrystalRus`.

Схема подключения дисплея WINSTAR к плате Arduino представлена на рис. 7.3.

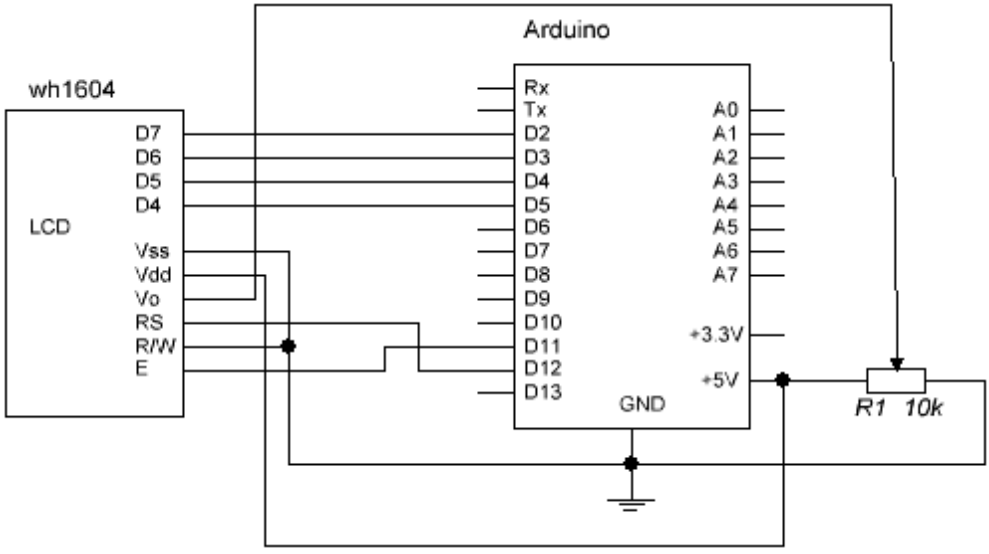


Рис. 7.3. Схема подключения дисплея WH1604 к плате Arduino

Дисплей WINSTAR, как и прочие дисплеи на контроллере HD44780, поддерживает два варианта параллельного интерфейса:

- 8-битный — выводы DB0–DB7, за один такт передается 1 байт (8 бит);
- 4-битный — выводы DB4–DB7, за один такт передается половина байта (4 бита).

Смысла использовать 8-битный вариант нет, потому что он требует задействовать больше контактов, а выигрыша в скорости все равно не дает — частота обновления дисплея не более 10 раз в секунду, так что мы в любом случае не сможем увидеть часто обновляемые данные. Поэтому выводы DB0–DB3 оставляем неподключенными.

Управление контрастностью изображения напряжением осуществляет потенциометр. В данном случае нормальное изображение наблюдалось при значениях напряжения на входе Vo в пределах 0,45–1,2 В.

Приступаем к программированию и постараемся на практическом примере показать возможности дисплея. Сначала подключаем библиотеку `LiquidCrystalRus` для работы с ЖКИ (LCD). Далее создаем объект LCD-дисплея, используя конструктор класса `LiquidCrystalRus` с 6 аргументами. Библиотека по количеству аргументов сама определит, что нужно использовать 4-битный интерфейс. Указываем, к каким пинам (контактам) Arduino подключены выводы дисплея: RS, E, DB4, DB5, DB6, DB7. Далее в функции `setup()` инициализируем дисплей (4 строки по 16 символов) и записываем выводимую на дисплей фразу. Данный код представлен в листинге 7.2.

Листинг 7.2

```
// Подключение библиотеки
#include <LiquidCrystalRus.h>
// Создаем объект, используя конструктор класса LiquidCrystal
// 12 - RS,11 - Enable,5 - D4, 4 - D5,3 - D6,2 - D7
LiquidCrystalRus lcd(12, 11, 5, 4, 3, 2);
// начальная установка
void setup()
 {
 // размерность индикатора
 lcd.begin(16, 4);
 // Начальное сообщение
 lcd.setCursor(0, 0);
 lcd.print("Демо к книге");
 lcd.setCursor(0, 1);
 lcd.print("Arduino");
 lcd.setCursor(0, 2);
 lcd.print("Возможности");
 lcd.setCursor(0, 3);
 lcd.print("WH1604");
 lcd.cursor();
 delay(10000);
}
```

Сначала посмотрим, как действуют методы `display()` и `noDisplay()`. Затем продемонстрируем вывод символов на экран дисплея. Методы `leftToRight()` и `rightToLeft()` устанавливают направление вывода текста — слева направо и справа налево соответственно. Для наглядности будем выводить строки посимвольно. При этом покажем и разные формы курсора. Первую строку выводим слева направо с обычным курсором (в виде подчеркивания), вторую — с невидимым курсором слева направо, третью — справа налево с большим мигающим курсором. Затем методом `clear()` очищаем экран и переводим курсор в позицию 0,0. Код данного отрезка представлен в листинге 7.3.

Листинг 7.3

```
void loop()
{
 // выключить дисплей без потери отображаемой на нем информации
 lcd.noDisplay();
 delay(3000);
 // включить дисплей без потери отображаемой на нем информации
 lcd.display();
 delay(3000);
//***** Вывод слева направо *******
lcd.clear();      // очистить экран
lcd.leftToRight();    // слева направо
lcd.cursor();     // курсор - подчеркивание
lcd.setCursor(0, 0);    // позиция 0,0
String text1="Слева направо";
for(int i = 0; i < text1.length(); i++)
{
lcd.write(text1[i]);   // посимвольный вывод
delay(300);      // задержка
}
lcd.noCursor();     // курсор - спрятан
lcd.setCursor(0, 1);    // позиция 0,1
for(int i = 0; i < text1.length(); i++)
{
lcd.write(text1[i]);   // посимвольный вывод
delay(300);      // задержка
}
//***** Вывод справа налево *******
lcd.rightToLeft();    // справа налево
lcd.blink();      // курсор - мигающий
lcd.setCursor(15, 2);   // позиция 15,2
String text2="овелан оварпС";
for(int i = 0; i < text2.length(); i++)
{
lcd.write(text2[i]);   // посимвольный вывод
delay(300);      // задержка
}
 lcd.clear();
}
```

Теперь продемонстрируем функцию `autoscroll()` при выводе символов на экран жидкокристаллического индикатора. Эта функция заставляет каждый вывод символа на экран перемещать предыдущие символы на одно знакоместо, что производит эффект вывода каждого нового символа в одно и то же знакоместо на экране. При этом предыдущие символы будут смещаться либо влево (направление вывода слева направо), либо вправо (направление вывода справа налево). Данный код представлен в листинге 7.4.

Листинг 7.4

```
//***** autoscroll слева направо *******
lcd.leftToRight();
lcd.autoscroll();
String text3="autoscroll";
lcd.blink();
lcd.setCursor(7, 1);
for(int i = 0; i < text3.length(); i++)
{
lcd.write(text3[i]);
delay(300);
}
// справо налево
lcd.rightToLeft();
lcd.setCursor(7, 2);
for(int i = 0; i < text3.length(); i++)
{
lcd.write(text3[i]);
delay(300);
}
lcd.noAutoscroll();
lcd.clear();
```

Теперь рассмотрим метод установки положения курсора — `setCursor()`. В примере будем каждую секунду переустанавливать курсор в случайную позицию на экране. Код данного отрезка представлен в листинге 7.5.

Листинг 7.5

```
//***** перемещения курсора *******
lcd.leftToRight();
lcd.cursor();
lcd.blink();
for(int i=0;i<10;i++)
{
int rand1=(micros()+millis())%1000;
lcd.setCursor(rand1%15,rand1%4);
delay(1000);
}
```

```
lcd.noBlink();
for(int i=0;i<10;i++)
{
int rand1=(micros()+millis());
lcd.setCursor(rand1%15,rand1%4);
delay(1000);
}
delay(1000);
lcd.home();
```

В заключение рассмотрим пример создания пользовательских символов. Для режима 5×7 можно создать 8 пользовательских символов с номерами от 0 до 7. Для вывода на экран дисплея используется метод `write(N)`, где N — номер созданного символа. Создадим 8 символов. Описываем наши собственные символы в виде массивов битовых масок. Каждый символ — это восемь масок по пять битов. Выводим символы на дисплей с интервалом 1 сек. Код представлен в листинге 7.6.

Листинг 7.6

```
byte s1[8]={B11111,B11111,B11111,B11111,B11111,B11111,B11111};
byte s2[8]={B00000,B00000,B11111,B11111,B11111,B11111,B11111};
byte s3[8]={B00000,B00000,B00000,B00000,B11111,B11111,B11111};
byte s4[8]={B00011,B00011,B00011,B00011,B00011,B00011,B00011};
. . . . . .
void loop()
 {
 . . . . . .
 for(int i=0;i<4;i++)
 {
 ;
 }
 lcd.createChar(0, s1);
 lcd.createChar(1, s1);
 lcd.createChar(2, s1);
 lcd.createChar(3, s1);
 lcd.setCursor(0, 0);
 lcd.write(0);lcd.write("");
 lcd.write(1);lcd.write("");
 lcd.write(2);lcd.write("");
 lcd.write(3);lcd.write("");
 delay(10000);
 }
```

Полный вариант рассмотренного скетча находится в папке examples/_07_1 сопровождающего книгу электронного архива. Напоминаю, что этот электронный архив можно скачать с FTP-сервера издательства "БХВ-Петербург" по ссылке **ftp://ftp.bhv.ru/9785977533379.zip**, а также со страницы книги на сайте **www.bhv.ru**.

ГЛАВА 8

Библиотека *EEPROM*

Микроконтроллеры ATmega имеют на борту энергонезависимую память EEPROM, не потеряющую записанные в нее данные даже после отключения питания. 512 байтов такой памяти несут ATmega8 и ATmega168, 1024 байта — ATmega328, 4096 байтов — Arduino Mega. Память типа EEPROM допускает несколько десятков тысяч циклов записи и стирания данных. Она может пригодиться для сохранения изменяющихся настроек при отключении питания от устройств Arduino. Для работы с этой памятью в составе Arduino IDE имеется удобная библиотека `EEPROM`.

8.1. Функции библиотек *EEPROM*

Библиотека `EEPROM` содержит две функции: чтения и записи в память данных.

8.1.1. Функция чтения *EEPROM.read*

Функция `EEPROM.read` считывает байт из энергонезависимой памяти EEPROM. Если байт до этого никогда не перезаписывался — вернет значение 255.

Синтаксис функции `EEPROM.read`:

```
EEPROM.read(address)
```

Параметр: `address` — порядковый номер ячейки памяти для чтения от 0 до 512 (или 1024) (`int`);

Возвращаемое значение — байт, хранимый в ячейке памяти.

Пример считывания значения всех байтов энергонезависимой памяти EEPROM и вывода их в COM-порт представлен в листинге 8.1.

Листинг 8.1

```
#include <EEPROM.h>
// начальный адрес памяти EEPROM
int address = 0;
```

```
byte value;
void setup()
 {
 Serial.begin(9600);
 }
void loop()
 {
 // считываем значение по текущему адресу EEPROM
 value = EEPROM.read(address);

 Serial.print(address);
 Serial.print("\t");
 Serial.print(value, DEC);
 Serial.println();

 // устанавливаем следующую ячейку памяти
 address = address + 1;
 // EEPROM содержит всего 512(1024) байт
 // если адрес достиг 512(1024), то снова переходим на 0
 if (address == 512)
  address = 0;
 delay(500);
 }
```

8.1.2. Функция записи *EEPROM.write*

Функция `EEPROM.write` записывает байт в энергонезависимую память.

Синтаксис функции `EEPROM.write`:

```
EEPROM.write(address, value)
```

Параметры:

- `address` — порядковый номер ячейки памяти для записи — от 0 до 511 (`int`);
- `value` — байт для записи — от 0 до 255 (`byte`).

Возвращаемого значения нет.

Возможное количество циклов перезаписи данных (Write/Erase Cycles) в памяти ограничено 100 тыс. раз — это следует учитывать при использовании данной памяти. Время, требуемое для завершения цикла записи, составляет 3,3 ms. Данная задержка уже учитывается библиотекой `EEPROM`.

Пример сохранения в энергонезависимой памяти EEPROM значений, считанных с аналогового входа `analog input 0`, представлен в листинге 8.2. Данные значения останутся в памяти и после отключения питания от платы и в будущем могут быть доступны для другого скетча.

Листинг 8.2

```
#include <EEPROM.h>
// текущее значение адреса EEPROM
int addr = 0;
void setup()
{;}
void loop()
 {
 if (addr < 512)
 {
 // EEPROM может хранить только значения от 0 до 255.
 int val = map(analogRead(0),0,1024,0,256);
 // записываем значение в энергонезависимую память
 EEPROM.write(addr, val);
 // устанавливаем следующую ячейку памяти.
 addr = addr + 1;
 }
 }
```

8.2. Примеры использования памяти EEPROM

8.2.1. Воспроизведение звука

Создадим проект, в котором устройство Arduino используется для генерации звука. Мелодии хранятся в памяти EEPROM и выводятся на динамик. Для этого на динамик следует подать сигнал соответствующей частоты. В качестве динамика используем динамическую головку 8 Ом, подключаемую к выводу D8. Схема подключения представлена на рис. 8.1.

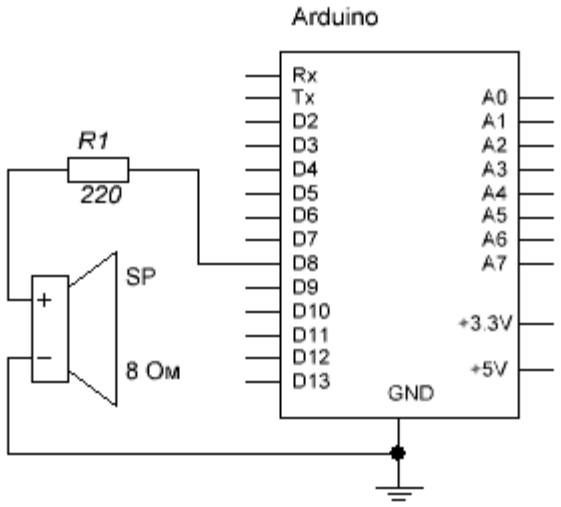


Рис. 8.1. Схема включения

Вместо сопротивления с номиналом 220 Ом можно использовать и большее, например, 510 Ом или 1 кОм. Недостатком такого подключения является то, что звук получается очень-очень тихий. Поэтому, чтобы получить громкость более приличного уровня, динамик можно подключить к выводу не напрямую, а через транзистор, как показано на рис. 8.2.

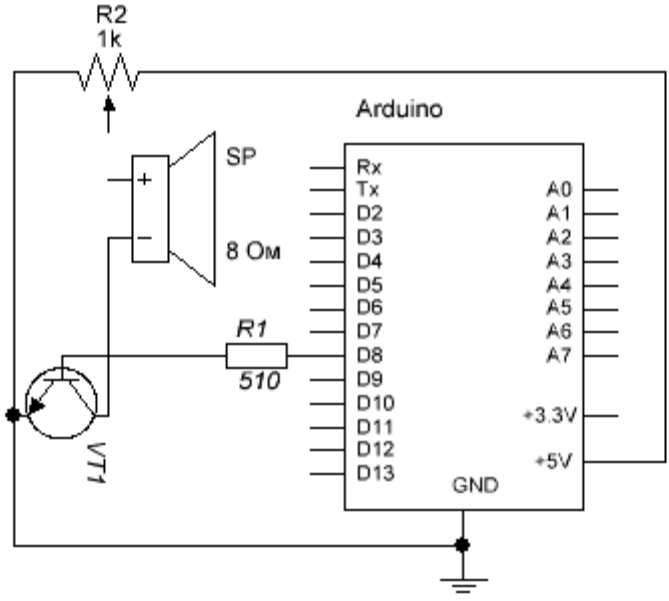


Рис. 8.2. Схема включения с транзистором

Громкость при этом получается весьма большой, поэтому в схему в качестве регулятора громкости добавлен потенциометр R2. Транзистор включен по схеме с общим эмиттером, и в данном случае выступает не в роли усилителя, а в качестве электронного ключа для согласования нагрузок. Дело в том, что у динамической головки сопротивление очень маленькое, и при подаче на нее напряжения +5 В через нее станет протекать ток около 625 мА. Однако максимальный ток, который могут обеспечить все выводы микроконтроллера, составляет всего 150 мА, т. е. в 4 раза меньше. И таким образом, подключая динамик к микроконтроллеру не напрямую, а через транзистор, способный пропускать через себя большее напряжение и ток большей силы, мы обеспечиваем электрическое согласование — в данном случае согласование по току.

Сначала составим скетч для воспроизведения одной мелодии. Мелодия состоит из двух массивов: массива с последовательным списком нот и массива со значениями длительности воспроизведения каждой ноты. Для воспроизведения одной ноты подаем на динамик сигнал определенной частоты и длительности. Затем воспроизводим следующую ноту. И так далее до окончания мелодии. Переменная `tempo` отвечает за скорость воспроизведения. Расчет тонов для нот одной октавы представлен в табл. 8.1.

Для подачи частотного сигнала на динамик воспользуемся функцией `tone()`. Скетч для воспроизведения мелодии представлен в листинге 8.3, а также находится в папке examples/_08_1 сопровождающего книгу электронного архива.

Таблица 8.1. *Таблица расчета тонов для нот одной октавы*

Нота	Обозначение	Частота, Гц	Период
до	c	261	3830
ре	d	294	3400
ми	e	329	3038
фа	f	349	2864
соль	g	392	2550
ля	a	440	2272
си	b	493	2028

Листинг 8.3

```
// ноты мелодии
char notes[]="GECgabCaCg DGECabCDED EFEDGEDC CECaCag gCEDgCEDEFGECDgC "; //
пробел - это пауза
// длительность для каждой ноты и паузы
int beats[] = { 4, 4, 4, 4, 1, 1, 1, 2, 1, 4, 2, 4, 4, 4, 4, 1, 1, 1, 2,
    1,4, 2, 1, 1, 1, 1, 2, 1, 1, 4, 2, 1, 2, 1, 2, 1, 1, 4, 2, 1,
    2, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 4, 4, 4} ;
// подключить динамик к pin 8
int speakerPin = 8;
int length = sizeof(beats); // число нот
// темп воспроизведения
int tempo = 400;
// проиграть ноту
void playNote(char note, int duration)
 {
 // массив для наименований нот (до ре ми ... и т. д. в пределах двух
 // октав)
 char names[]={'c','d','e','f','g','a','b','C','D','E','F','G','A','B'};
 int tones[]={3830,3400,3038,2864,2550,2272,2028,1912,1700,
       1518,1432,1276,1136,1014 };
 // проиграть тон, соответствующий ноте
for (int i = 0; i < sizeof(tones); i++)
 {
 if (names[i] == note)
 tone(speakerPin,tones[i],duration * 1000L);
 }
}
```

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

void loop()
 {
 for (int i = 0; i < length; i++)
 {
 if (notes[i] == ' ')
  tone(speakerPin,0, beats[i]*tempo* 1000L); // пауза
 else
 playNote(notes[i], beats[i] * tempo);
 }
 // пауза между нотами
 delay(tempo / 2);
 }
delay(3000);
}
```

8.2.2. Звонок с мелодиями

Проигрывать мелодию через динамик мы уже умеем. Теперь напишем скетч загрузки мелодий в память EEPROM. Зарезервируем там место для 10 мелодий, которые сможем проигрывать в нужный момент. Карта адресов памяти будет иметь следующий вид:

- `0-9` — указатель на адрес мелодии;
- `10-511` — место для мелодий.

Каждая мелодия будет представлять следующую последовательность данных:

`tempo(2 байта), нота1, длит.1, нота2, длит.2 ,..., нотаN, длит.N, end`

где:

- `tempo` — темп проигрываемой мелодии;
- `нота1, нота2 ,..., нотаN` — ноты;
- `длит1, длит2 ,..., длитN` — длительности проигрывания нот;
- `end` — байт конца мелодии (значение 0).

Запись последовательности нот и длительности их воспроизведения для каждой мелодии хранятся в массивах. В нашем скетче мы после записи мелодий в память EEPROM выведем содержимое памяти EEPROM. Фрагмент содержимого скетча представлен в листинге 8.4. Текст рассмотренного скетча находится в папке examples/_08_2 сопровождающего книгу электронного архива.

Листинг 8.4

```
// Использование библиотеки EEPROM
// запись мелодий в EEPROM
// подключение библиотеки EEPROM
#include <EEPROM.h>
// ноты мелодии
String notes[] ={ "ccggaagffeeddc ",
 "GECgabCaCg DGECabCDED EFEDGEDC CECaCag gCEDgCEDEFGECDgC "};
// пробел - это пауза
// длительность для каждой ноты и паузы
String beats[]={"1,1,1,1,1,1,2,1,1,1,1,1,1,2,4",
 "4,4,4,4,1,1,1,2,1,4,2,4,4,4,4,1,1,1,2,1,4,2,1,1,1,1,2,1,1,4,2,1,
 2,1,2,1,1,4,2,1,2,1,2,1,2,1,2,1,1,1,1,1,2,1,4,4,4"} ;
int tempo[]= {300,400};
.........
int addr1=0;
int addr2=10;
void write_melody(int num)
 {
EEPROM.write(addr1,addr2);     // указатель на мелодию
EEPROM.write(addr2,tempo[num]>>8); // для tempo - 2 байта
 addr2++;
 EEPROM.write(addr2,tempo[num]);
 addr2++;
for(int j=0;j<notes[num].length();j++)
 {
 EEPROM.write(addr2,notes[num][j]); // нота
 addr2++;
 EEPROM.write(addr2,beats[num][j*2]); // длительность
 addr2++;
 Serial.print(notes[num][j]);Serial.print(" ");
 Serial.print(beats[num][j*2]-48);Serial.println(" ");
 }
 // записать 0 - end
 EEPROM.write(addr2,0);
 addr2++;
 addr1++;
 }
void setup()
 {
 .........
 for(int i=0;i<sizeof(tempo)/2;i++)
  {write_melody(i);}
 Serial.println("end..."); Serial.println("control...");
 for(int i=0;i<511;i++)
 {
 Serial.print(i);Serial.print(" - ");
```

```
 Serial.println(EEPROM.read(i),DEC);
 }
 }
void loop()
 {;}
```

В примере мы загружаем в память 2 мелодии. Я думаю, у вас не возникнет трудностей для изменения скетча для загрузки большего количества мелодий.

А теперь напишем скетч, проигрывающий случайную мелодию из памяти EEPROM при нажатии кнопки. Схема при этом изменяется незначительно. Один контакт кнопки подсоединяем к D9. К нему же подключаем подтягивающий резистор 10 кОм, который в свою очередь соединяем с землей. Другой выход кнопки соединяем с питанием 5 В. При включении в программе `setup()` подсчитаем количество записанных мелодий. Затем в цикле на динамик выдается мелодия. Данные мелодии (темп, список нот и их продолжительность) берутся из памяти EEPROM. Для контроля данные выводятся в монитор последовательного порта. Содержимое скетча представлено в листинге 8.5. Текст рассмотренного скетча находится в папке examples/_08_3 сопровождающего книгу электронного архива.

Листинг 8.5

```
// ***** Воспроизведение мелодии, записанной в EEPROM
// подключение библиотеки EEPROM
#include <EEPROM.h>
// подключить динамик к pin 8
int speakerPin = 8;
// подключить кнопку к pin 9
int buttonPin = 9;
// темп воспроизведения, ноты, длительности
int tempo,notes,beats;
// адрес указателя мелодии, случайная мелодия, количество мелодий
int addr1;
int rand1;
int count=0;
// проиграть ноту
void playNote(char note, int duration)
 {
 // массив для наименований нот (до ре ми ... и т. д. в пределах двух октав)
 char names[] = { 'c', 'd', 'e', 'f', 'g', 'a', 'b',
'C','D','E','F','G','A','B' };
 int tones[] = { 3830, 3400, 3038, 2864, 2550, 2272, 2028, 1912, 1700, 1518,
1432, 1276, 1136, 1014 };
 // проиграть тон, соответствующий ноте
 for (int i = 0; i < sizeof(tones); i++)
 {
 if (names[i] == note)
```

```
  {
  tone(speakerPin,tones[i],duration * 1000L);
  }
 }
 }

void setup()
 {
 pinMode(speakerPin, OUTPUT);
 pinMode(buttonPin, INPUT);
 Serial.begin(9600);
 for(int i=0;i<200;i++)
 {
 Serial.print(i);Serial.print(" - ");
 Serial.println(EEPROM.read(i),DEC);
 }
 int i=0;
 while(EEPROM.read(i)<255)
 {count++;i++;}
 Serial.print("count=");Serial.println(count);
 }

void loop()
 {
 if(digitalRead(buttonPin)==HIGH)
 {
 Serial.println("click");
 // получить случайное
 rand1=random(0,count+1);
 addr1=EEPROM.read(rand1);
 Serial.println(addr1);
 //
 tempo=EEPROM.read(addr1);
 addr1++;
 Serial.print(tempo,DEC);Serial.println("");
 tempo=tempo*256+EEPROM.read(addr1);
 addr1++;
 Serial.print(tempo,DEC);Serial.println("");
 while(EEPROM.read(addr1)>0)
  {
  notes=EEPROM.read(addr1);
  addr1++;
  beats=EEPROM.read(addr1);
  addr1++;
  Serial.print(notes);Serial.print(" ");
  Serial.print(beats,DEC);Serial.println(" ");
```

```
 if (notes == ' ')
  tone(speakerPin,0, beats*tempo*1000L); // пауза
 else
  playNote(notes, beats*tempo);
 // пауза между нотами
 delay(tempo / 2);
 }
Serial.println("end");
}
}
```

ГЛАВА 9

Подключение клавиатуры и мыши

Широкую популярность интерфейс PS/2 (Personal System/2) приобрел благодаря использованию для взаимодействия с манипулятором типа мышь в компьютерах Apple Macintosh и позднее, в конце 1980-х годов, в ОС Windows для IBM PC. Однако сегодня PS/2 практически полностью заменен интерфейсом USB. Тем не менее, протокол PS/2 представляет собой интересную платформу для различных экспериментов. Например, можно с помощью клавиатуры, подключенной через этот интерфейс, осуществлять ввод фраз для бегущей строки из светодиодных матриц, подключенных к Arduino.

Схема выводов разъема PS/2 (в просторечии *распиновка*) представлена на рис. 9.1.

Рис. 9.1. Разъем PS/2

9.1. Обмен данными по протоколу PS/2

Обмен данными через PS/2 осуществляется асинхронно по последовательному протоколу. Для обмена информацией используются 2 линии:

- 1 — `DATA` (по этой линии передаются сами данные);
- 5 — `CLOCK` (по этой линии передаются тактовые сигналы).

При передаче от устройства (клавиатуры или мыши), подключенного через интерфейс PS/2 к компьютеру (или, в нашем случае, к контроллеру), используется протокол, суть которого иллюстрирует схема, показанная на рис. 9.2. Устройство не начинает передачу, если линия `Clock` не находилась в "1" по крайней мере 50 микросекунд.

Устройство передает последовательно:

- ❒ стартовый бит — всегда ноль;
- ❒ 8 битов данных;
- ❒ бит четности;
- ❒ стоп-бит — всегда единица.

Таким образом, данные передаются по одному байту за раз, при этом байт данных передается побитно. Первым, как можно видеть, идет стартовый бит (Start bit) — он всегда передается низким уровнем (логический "0"). Далее передается сам байт данных, начиная от младшего бита к старшему. То есть, если нужно передать байт 0xF0, который в двоичной системе записывается так: 11110000, то передаваться его биты будут в следующем порядке: "0","0","0","0","1","1","1","1". После данных идет бит четности (Parity bit): если число четное — будет передана "1", а если нечетное — "0". Бит четности — это контрольный бит, принимающий значения "0" или "1" и служащий для проверки общей четности двоичного числа (четности количества единичных битов в числе). Стоп-бит (Stop bit) — всегда логическая "1".

Устройство устанавливает/меняет сигнал `Data`, когда линия `Clock` находится в логической единице. Контроллер читает данные, когда линия `Clock` находится в логическом нуле.

Частота сигнала `Clock` примерно 10–16,7 кГц. Время от фронта сигнала `Clock` до момента изменения сигнала `Data` — не менее 5 микросекунд.

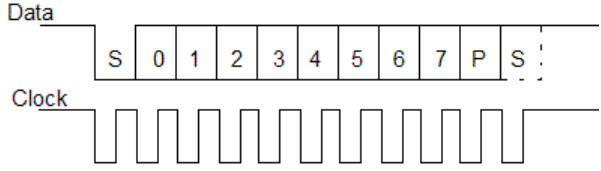


Рис. 9.2. Передача данных от PS/2-клавиатуры (мыши) к контроллеру

При передаче команд в обратную сторону — от контроллера к клавиатуре или мыши — протокол отличается от ранее описанного. Последовательность передаваемых битов здесь будет такая (рис. 9.3):

1. Хост-контроллер опускает сигнал `Clock` в ноль на время примерно 100 микросекунд.
2. Хост-контроллер опускает сигнал `Data` в ноль, формируя стартовый бит.
3. Хост-контроллер отпускает сигнал `Clock` в логическую единицу, клавиатура фиксирует стартовый бит.
4. Далее клавиатура генерирует сигнал `Clock`, а хост-контроллер подает передаваемые биты.
5. После того, как хост-контроллер передаст все свои биты, включая бит четности и стоп-бит, клавиатура посылает последний бит "0", который является подтверждением приема.

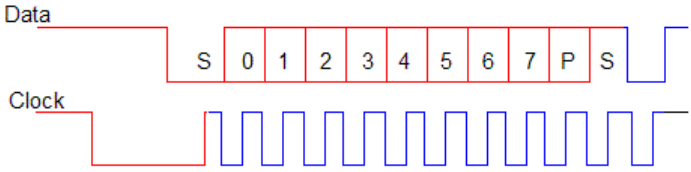


Рис. 9.3. Отправка данных по протоколу PS/2 от контроллера к клавиатуре (мыши)

9.2. Библиотека *ps2dev*

На официальном сайте Arduino имеется библиотека для работы с устройствами PS/2. Скачать библиотеку можно по адресу **http://www.arduino.cc/playground/uploads/ComponentLib/ps2dev.zip.** В архиве находится папка ps2dev, которую нужно поместить в библиотечный каталог Arduino IDE (\hardware\libraries\).

Библиотека PS/2 представляет собой класс C++:

```
class PS2dev
{
       public:
              PS2dev(int clk, int data);
              int write(unsigned char data);
              int read(unsigned char * data);
       private:
              int _ps2clk;
              int _ps2data;
              void golo(int pin);
              void gohi(int pin);
};
```

Конструктор класса принимает два параметра типа `int` — номера портов устройства Arduino, к которым подключены линии `Clock` и `Data`, например:

```
PS2dev mouse(6, 5);
```

Здесь к `Pin5` подключена линия `Data`, а к `pin6` — линия `Clock`.

Функция `golo` переводит порт в режим вывода и устанавливает на нем логический "0", функция же `gohi` переводит порт в режим ввода и подключает на нем подтягивающий резистор. Обе функции имеют тип `private` и используются только внутри класса. Нам же доступны всего две функции:

- `write` — записывает байт в линию PS/2;
- `read` — считывает байт из линии PS/2.

Файлы библиотеки вы можете найти папке libraries/ps2dev сопровождающего книгу электронного архива.

9.3. Подключение клавиатуры

В далекие 1990-е годы мне пришлось разрабатывать клавиатуру XT/AT. От этого времени осталось только воспоминание о множестве малопонятных последовательностей байтов, отсылаемых из клавиатуры в компьютер. Было бы интересно, насколько изменился протокол для клавиатуры PS/2. Изучение скудных материалов по теме в Интернете, а также анализ полученных кодов из подключенной к Arduino PS/2-клавиатуры позволил выявить следующее.

На каждую принятую от контроллера команду (или — проще сказать — на каждый принятый байт) клавиатура должна обязательно ответить одним из следующих байтов:

- `0xFA` — подтверждение об успешном приеме;
- `0xFE` — команда принята с ошибкой — вероятно, это ошибка циклической контрольной суммы (CRC).

При получении от контроллера команды `0xFF` (начальная установка) клавиатура отвечает `0xFA`, а затем сбрасывается и посылает в ответ байт `0xAA`.

В клавиатуру контроллер может послать следующие команды:

- `0xED` — зажечь или потушить светодиоды CAPS/NUM/SCROLL. Если клавиатура принимает эту команду, то она не пошлет более никаких ответов до тех пор, пока компьютер не пришлет следующий байт — параметр, который определяет битовую маску: один бит — это один из светодиодов. Битовая маска для светодиодов клавиатуры определена так:
  - `#define KEYBOARD_CAPS_LOCK_ON  4`
  - `#define KEYBOARD_NUM_LOCK_ON   2`
  - `#define KEYBOARD_SCROLL_LOCK_ON 1`
- `0xF3` —— это тоже двухбайтовая команда. После этой команды следует параметр, определяющий частоту повтора кодов при нажатой клавише и интервал времени между нажатием и началом повторов.

После подачи напряжения питания клавиатура посылает код `0xAA` и немедленно готова к работе. Она сразу, без дополнительного программирования, готова посылать коды нажатых клавиш. По умолчанию клавиатура посылает на нажатие клавиши один байт-код, а на отпускание клавиши два байта. Первый байт в кодах отпускания клавиши — это префикс отпускания `0xF0`. Например, если нажать и отпустить клавишу <1>, то клавиатура пошлет последовательность кодов `0x16`, `0xF0`, `0x16`. Существует еще так называемый *дополнительный код* — это префикс `0xE0`. Он посылается вместе с кодами дополнительных клавиш. Например, при нажатии клавиши <Insert> (нецифрового блока) отправляется последовательность `0xE0`, `0x70`, `0xE0`, а при ее отпускании `0xF0`, `0x70`.

До сих пор все более или менее понятно, но, как оказалось позже (при написании редактора текста — см. *разд. 9.4*), это еще не все... Получив от контроллера команду зажечь светодиод — например, Num_Lock, клавиатура зажигает соответствую-

щий светодиод, но теперь при нажатии клавиш основной панели клавиатуры, которые продублированы на цифровой панели (<Del>, <Home>, <Insert> и пр.), отправляется иная последовательность байтов, например:

- клавиша <Home> основной панели при потушенном Num_Lock:

```
E0 6C E0
```

- клавиша <Home> основной панели при включенном Num_Lock:

```
E0 12 E0 6C E0
```

Это все необходимо учитывать при приеме кодов с клавиатуры. В следующем разделе мы создадим редактор текста. Текст, получаемый с клавиатуры, будет выводиться на дисплей WINSTAR WH1604. При этом вводимый текст мы сможем редактировать.

9.4. Редактор текста на дисплее WH1604

Для чего нам необходим редактор текста? Казалось бы, при столь небольшом дисплее пользы от него не будет никакой. На самом деле существует множество задач, когда он может пригодиться. Например, вы используете в своем Arduino-проекте GSM-GPRS-shield и отправляете произвольные SMS-сообщения, которые сначала необходимо набрать и, возможно, отредактировать. Или отправляете сообщения на сайт, используя Ethernet-shied. Придумать применений можно много.

Сначала определим список служебных клавиш, необходимых при редактировании текста или выполнении действий с набранным текстом (сохранить в файл, отправить SMS и пр.), и зададим их значения в константах. Список используемых констант приведен в листинге 9.1. Этот список в последующем мы будем расширять.

Листинг 9.1

```
#define CAPS 0x58  // включить/выключить режим Caps_Lock
#define NUM 0x77   // включить/выключить режим Num_Lock
#define SCROLL 0x7E // включить/выключить режим Scroll

#define ESC 0x76   // выход из редактора
#define F10 0x9   // сохранить в файл на SD-карте памяти
// будем еще добавлять служебные клавиши

#define ALT 0x11   // выбор языка eng/rus
#define CTRL 0x14   // нажатие Ctrl
#define L_SHIFT 0x12  // нажатие клавиши shift
#define R_SHIFT 0x59  // нажатие клавиши shift

#define LEFT 0x6B   // курсор влево
#define RIGHT 0x74   // курсор вправо
#define UP 0x75    // курсор вверх на 1 строку
```

```
#define DOWN 0x72   // курсор вниз на 1 строку
#define PGUP 0x7D   //
#define PGDOWN 0x7A  //
#define END 0x69    // курсор в конец строки
#define HOME 0x6C   // курсор в начало строки
#define NUM5 0x73   // пустая клавиша цифровой панели

#define BACKSPACE 0x66 // удалить предыдущий символ
#define DELETE 0x71  // удалить символ из текущей позиции
#define INSERT 0x70  // включить/выключить режим Insert
```

При нажатии обычных клавиш их визуальное представление будет выводиться в текущую позицию дисплея с заменой текущего символа (режим Insert) или со сдвигом курсора вправо. Линию `Clock` клавиатуры подсоединим к `pin3` и запустим обработчик прерывания 1 для отслеживания момента передачи данных с клавиатуры. Схема подключения клавиатуры и дисплея представлена на рис. 9.4.

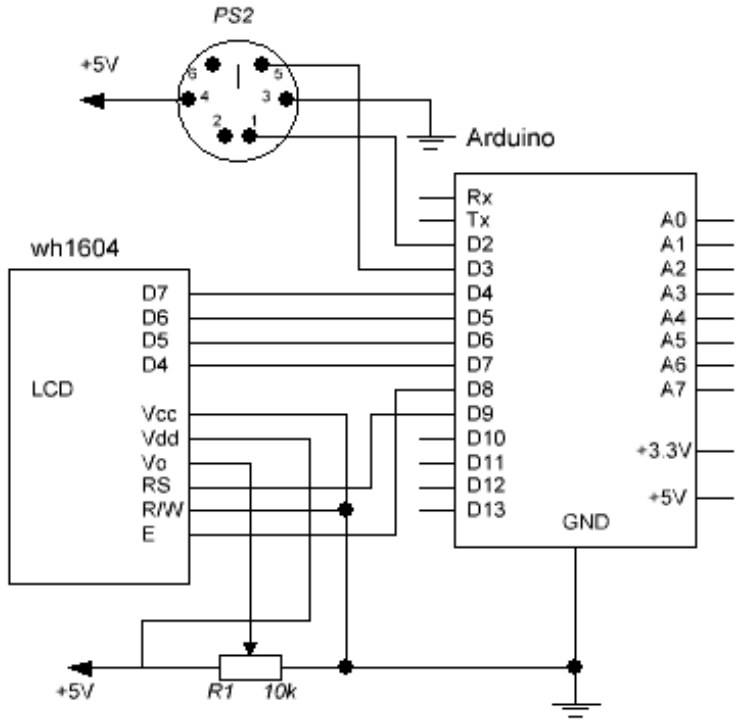


Рис. 9.4. Схема подключения

В функции начальной установки необходимо инициализировать клавиатуру и дисплей и запустить обработчик прерывания по событию начала отправки кода с клавиатуры. Содержимое функции `setup()` представлено в листинге 9.2.

Листинг 9.2

```
void setup()
{
 Serial.begin(9600);
 Serial.println("start-ok");
 // инициализация клавиатуры
 kbd_init();
 Serial.println("keyboard-ok");
 // инициализация дисплея
 lcd.begin(16, 4);
 // Начальное сообщение
 lcd.clear();
 lcd.print("Редактор текста");
 lcd.cursor();
 delay(5000);
 lcd.clear();
 // ожидание передачи от клавиатуры по прерыванию 1
 attachInterrupt(1, get_char, FALLING);
}
```

Коды, поступающие с клавиатуры PS/2, не соответствуют кодам ASCII. Если отсечь все служебные коды, из клавиатуры выдаются коды в интервале `0 – 0x7f`. Для перевода поступающих кодов в коды ASCII создадим массив размером 1024 байта. Массив памяти условно разделен на 8 блоков. Карта распределения памяти в массиве для разных режимов приведена в табл. 9.1. Код нажатия неслужебной клавиши клавиатуры будет являться адресом нахождения соответствующего кода ASCII для нажатой клавиши. Отпускание клавиши отсечем программно. Переключение между языками станем осуществлять по нажатии клавиши <Alt>. В скетче предусмотрим возможность перехода и по нажатии клавиши <Scroll> — этот вариант более удобен, т. к. при этом будет осуществляться индикация на светодиоде Scroll клавиатуры выбранного языка. Выбор нужного символа для режима Num_Lock будем подготавливать программно.

Таблица 9.1. *Карта распределения памяти для перевода кодов клавиатуры в коды ASCII*

Адреса	Eng/Rus	Caps_Lock	Shift
0 – 0x7f	Eng	1	0
0x80 – 0xff	Eng	1	1
0x100 – 0x17f	Eng	1	0
0x180 – 0x1ff	Eng	1	1
0x200 – 0x21f	Rus	0	0
0x280 – 0x2ff	Rus	0	1

Таблица 9.1 *(окончание)*

Адреса	Eng/Rus	Caps_Lock	Shift
0x300 — 0x31f	Rus	0	0
0x380 — 0x3ff	Rus	0	1

Фрагмент содержимого массива приведен в листинге 9.3.

Листинг 9.3

```
unsigned char table1[1024]={
 // 0 - 0x7f
 ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
 ' ',' ',' ',' ',' ','q','1',' ',' ',' ','z','s','a','w','2',' ',
 ' ','c','x','d','e','4','3',' ',' ',' ','v','f','t','r','5',' ',
 ' ','n','b','h','g','y','6',' ',' ',' ','m','j','u','7','8',' ',
 ' ',',','k','i','o','0','9',' ',' ','.','/','l',';','p','-',' ',
 ' ',' ','\'',' ','[','=',' ',' ',' ',' ',' ',' ',']','\\',' ',' ',
 ' ',' ',' ',' ',' ',' ',' ',' ',' ','1',' ',' ','4','7',' ',' ',' ',
 '0',' ',' ','2','5','6','8',' ',' ',' ','+','3','-','*','9',' ',' ',
 // 0x80 - 0xff
 ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
 ' ',' ',' ',' ',' ','Q','1',' ',' ',' ','Z','S','A','W','2',' ',
 ' ','C','X','D','E','4','3',' ',' ',' ','V','F','T','R','5',' ',
 ' ','N','B','H','G','Y','6',' ',' ',' ','M','J','U','7','8',' ',
 ' ','<','K','I','O','0','9',' ',' ','.','/','L',';','P','-',' ',
 ' ',' ',' ','\'',' ','[','=',' ',' ',' ',' ',' ',' ',']','\\',' ',' ',
 ' ',' ',' ',' ',' ',' ',' ',' ',' ','1',' ',' ','4','7',' ',' ',' ',
 '0',' ',' ','2','5','6','8',' ',' ',' ','+','3','-','*','9',' ',' ',
 // 0x100 - 0x17f
.........
};
```

При начале передачи кода из клавиатуры при переходе тактового сигнала `Clock` из логической "1" в "0" на digital pin 3 происходит вызов процедуры обработки прерывания 1 — функции `get_char()`. Функция отключает прерывание 1, получает байт от клавиатуры и производит анализ полученного байта. При этом возможны следующие действия:

- установка служебных переменных:
  - `f0` — признак получения кода отпускания клавиши;
  - `e0` — начало служебной последовательности;
  - `lang` — язык (для выбора языка по клавише <Scroll>);
  - `alt` — язык (для выбора языка по клавише <Alt>);

- `ins` — режим Insert;
- `l_shift,r_shift` — состояние клавиш <Shift>;
- `leds` — состояние светодиодов на клавиатуре;

❒ отправка команд установки светодиодов на клавиатуре;

❒ вывод символа на дисплей и в буфер данных;

❒ действия с курсором;

❒ действия со всем набранным текстом — выход, сохранение, отправка и т. д.

После выполнения всех действий с полученным байтом, которые мы рассмотрим далее, запускается прерывание 1. Фрагмент функции `get_char()` приведен в листинге 9.4.

Листинг 9.4

```
void get_char()
 {
 unsigned char code;int ledsb=0;
 detachInterrupt(1);  // отключить прерывание 0
 code = kbd.read();  // получить байт от клавиатуры
switch(code)
 {
 case 0: break;
 case 0xF0: f0=1; break;
 case 0xFA: break;
 case 0xE0: e0=1-e0;
   break;
 case ALT: if(f0==0)alt=1-alt; else f0=0;
   break; // eng/rus
 .. .. ..
      }
 attachInterrupt(1, get_char, FALLING);
 }
```

При получении от клавиатуры кодов нажатия клавиш <Num Lock>, <Caps Lock> и <Scroll Lock> необходимо установить новое значение для служебной переменной `leds` (листинг 9.5) и вызвать функцию `set_leds()` для отправки команд клавиатуре (листинг 9.6).

Листинг 9.5

```
 case NUM: // NumLock
      if(bitRead(leds,1)==1 && f0<1)
       {bitClear(leds,1);set_leds();}
      else if(bitRead(leds,1)==0 && f0<1)
       {bitSet(leds,1);set_leds();}
```

```
      else f0=0;
   break;
 case CAPS: // CapsLock
      if(bitRead(leds,2)==1 && f0<1)
       {bitClear(leds,2);set_leds();}
      else if(bitRead(leds,2)==0 && f0<1)
       {bitSet(leds,2);set_leds();}
      else f0=0;
   break;
 case SCROLL: // ScrollLock
      if(bitRead(leds,0)==1 && f0<1)
       {bitClear(leds,0);set_leds();lang=0;}
      else if(bitRead(leds,0)==0 && f0<1)
       {bitSet(leds,0);set_leds();lang=1;}
      else f0=0;
   break;
```

Листинг 9.6

```
void set_leds()
 {
 kbd.write(0xED); // send reset code
 kbd.write(leds); // send reset code
 }
```

При получении кодов нажатия клавиш цифровой панели (для режима не Num_Lock) и аналогичных им клавиш основной клавиатуры (кроме <Backspace> и <Delete>) происходит вызов функций для расчета нового положения курсора `new_cursor()` и установки нового положения курсора `set_cursor()`. Содержимое этих функций приведено в листинге 9.7.

Листинг 9.7

```
// рассчитать для нового положения курсора
// смещения для Left=-1, Right = +1, Up = -width дисплея и т. д.
void new_cursor(int to)
 {
 //txt_tek_pos=min(txt_tek_pos+to,lcd_cols*lcd_rows-1);
 txt_tek_pos=min(txt_tek_pos+to,txt_end_pos);
 txt_tek_pos=max(txt_tek_pos,0);
 set_cursor(txt_tek_pos);
 }
// установка курсора
void set_cursor(int pos)
 {
 int col;int row;
 col=max(0,pos%lcd_cols);
```

```
 if(pos==lcd_cols*lcd_rows)
 col=lcd_cols-1;
 row=min(3,pos/lcd_cols);
    Serial.print("pos=");Serial.print(pos);
    Serial.print("txt_end_pos=");Serial.print(txt_end_pos);
    Serial.print(" col=");Serial.print(col);
    Serial.print(" row=");Serial.println(row);
 lcd.setCursor(col,row);
 }
```

При получении кодов нажатия обычных клавиш клавиатуры либо клавиш цифровой панели при установленном режиме Num_Lock получаем код ASCII нажатой клавиши из массива `table1`. При получении адреса в таблице учитываются состояния переменных `caps`, `l_shift`, `r_shift` и `alt` (l или ang). Полученный из таблицы символ выводим на дисплей в текущую позицию курсора. Кроме того, весь набираемый текст хранится в буфере (массив `txt_buffer[]`). Код для указанных действий приведен в листинге 9.8.

Листинг 9.8

```
 default: if(f0==1)
      {f0=0;}
     else
      {
      int addr=bitRead(leds,2)*128+min(1,l_shift+r_shift)*256;
      int addr+=alt*512+code;
      // если для переключения eng/rus используется Scroll_Lock
      // int addr+=lang*512+code;
      add_buffer(table1[addr]);
      }
     break;
.. .. ..
// ****************** добавление в буфер символа
void add_buffer(char char1)
 {
 if(txt_end_pos==txt_tek_pos)
 {txt_end_pos++;}
 else
 {txt_end_pos=txt_end_pos+(1-ins);shift_buffer_right();}
 txt_buffer[txt_tek_pos]=char1;
 if(txt_end_pos!=txt_tek_pos)
 shift_lcd_right();
 txt_tek_pos=min(txt_tek_pos+1,lcd_cols*lcd_rows-1);
 txt_tek_pos=max(txt_tek_pos,0);
 set_cursor(txt_tek_pos-offset);
 }
```

```
// сдвиг буфера текста вправо
void shift_buffer_right()
 {
 if(ins==1)
 return;
 for(int i=txt_end_pos;i>=txt_tek_pos;i--)
 txt_buffer[i]=txt_buffer[i-1];
 }
// сдвиг на дисплее вправо
void shift_lcd_right()
 {;
 for(int j=txt_tek_pos;j<min(txt_end_pos,lcd_cols*lcd_rows);j++)
 {set_cursor(j);lcd.write(txt_buffer[j]);}
 }
```

При получении с клавиатуры кода нажатия клавиши <Backspace> необходимо удалить предыдущий символ из буфера текста и вывести заново на дисплей содержимое буфера от удаляемого символа до конца. Код для данных действий приведен в листинге 9.9.

Листинг 9.9

```
 case BACKSPACE: if(f0==0){delete_from_buffer();}
         else(f0=0);
       break;
.. .. ..
// ****************** удаление из буфера backspace
void delete_from_buffer()
 {
 if(txt_tek_pos>0)
 {
 txt_buffer[txt_end_pos]=' ';
 txt_end_pos--;
 txt_tek_pos--;
 shift_buffer_left();
 shift_lcd_left();
 }
 set_cursor(txt_tek_pos);
 }
// сдвиг буфера текста влево
void shift_buffer_left()
 {
 for(int i=txt_tek_pos;i<=txt_end_pos;i++)
 txt_buffer[i]=txt_buffer[i+1];
 }
// сдвиг на дисплее влево
void shift_lcd_left()
```

```
 {;
 for(int j=txt_tek_pos;j<=min(txt_end_pos,lcd_cols*lcd_rows);j++)
 {set_cursor(j);lcd.write(txt_buffer[j]);}
 }
```

При получении с клавиатуры кода нажатия клавиши <Delete> необходимо удалить символ из буфера текста из текущей позиции и вывести заново на дисплей содержимое буфера от текущего символа до конца. Код для данных действий приведен в листинге 9.10.

Листинг 9.10

```
 case DELETE: if(f0==0){del_from_buffer();}
         else(f0=0);
         Serial.println("delete");
       break;
.. .. ..
// ****************** удаление из буфера delete
void del_from_buffer()
 {
 if(txt_tek_pos<txt_end_pos)
 {
 txt_buffer[txt_end_pos]=' ';
 txt_end_pos--;
 shift_buffer_left();
 shift_lcd_left();
 }
set_cursor(txt_tek_pos);
 }
```

Текст рассмотренного скетча находится в папке examples/_09_1 сопровождающего книгу электронного архива.

9.5. Подключение мыши

Рассмотрим протокол PS/2 для мыши (обычной 3-кнопочной с колесом). Программировать мышь не легче, чем клавиатуру, а, может быть, даже и труднее. После включения питания мышь не готова к работе, она лишь посылает байты `0xAA` и, следом, `0x00`. Требуется запрограммировать мышь на разрешение передачи. Для этого, как минимум, нужно послать ей специальную команду `0xF4`. Только после этой команды мышь начнет посылать пакеты с координатами и состоянием кнопок. Вторая проблема состоит в том, что по умолчанию мышь работает без колеса прокрутки. Для того чтобы включить колесо, нужно выполнить инициализацию мыши, послав с контроллера в мышь следующую последовательность байтов:

```
0xF3, 200, 0xF3, 100, 0xF3, 80, 0xF2
```

На каждый из этих байтов мышь отвечает 0xFA. Последняя команда из посланной последовательности здесь — это 0xF2. После этой последовательности мышь должна прислать ID. Если пришел ноль, значит колеса нет и не будет. Если пришло "3", то это значит, что колесо прокрутки включено.

Приведем перечень наиболее важных команд для мыши:

- SET_MOUSE_RESOLUTION = 0xE8
- SET_MOUSE_SAMPLING_RATE = 0xF3
- ENABLE_MOUSE_TRANSMISSION = 0xF4
- SET_MOUSE_SCALING_1TO1 = 0xE6
- READ_MOUSE_STATUS = 0xE9
- GET_DEVICE_ID = 0xF2
- MOUSE_RESET = 0xFF

Некоторые из них идут в паре с параметром, некоторые — команды одиночные.

В ответ на команду 0xF4 мышь без колеса прокрутки посылает пакеты по три байта, а мышь с колесом — пакеты по четыре байта.

Первый байт в пакете передает знак перемещения SY и SX (вверх-вниз и влево-вправо), а также состояние трех кнопок: BM (middle — центральная), BR (right — правая), BL (left — левая):

```
0 0 SY SX 1 BM BR BL
```

Второй байт передает смещение по координате X:

```
X7 X6 X5 X4 X3 X2 X1 X0
```

Третий байт передает смещение по координате Y:

```
Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
```

Четвертый байт посылается только для мышей с колесом и только, если оно включено. Он и определяет вращение колеса.

```
 0 0 0 0 Z3 Z2 Z1 Z0
```

Теперь закрепим полученные знания на практическом примере.

9.6. Опрос состояния мыши

Создадим пример опроса состояния мыши с выводом значений состояния в последовательный порт. Линию Clock мыши (вывод 5) подсоединим к pin 11, линию Data (вывод 1) к pin 10.

В функции начальной установки необходимо инициализировать мышь для работы в трехкнопочном режиме (листинг 9.11). Отправленные с контроллера команды и ответы мыши для наглядности выводим в последовательный порт (рис. 9.5).

Листинг 9.11

```
void mouse_init()
{
 mouse.write(0xff); // команда сброса
 mouse.read(); // ответ мыши
 mouse.read(); //
 mouse.read(); //
 mouse.write(0xf0); // remote mode
 mouse.read(); // ответ мыши
 delayMicroseconds(100);
 // установим режим третьей кнопки
 mouse.write(0xF3);Serial.print(0xF3,HEX);Serial.print(" - ");
 Serial.println(mouse.read(),HEX);
 mouse.write(200);Serial.print(200,DEC);Serial.print(" - ");
 Serial.println(mouse.read(),HEX);
 mouse.write(0xF3);Serial.print(0xF3,HEX);Serial.print(" - ");
 Serial.println(mouse.read(),HEX);
 mouse.write(100);Serial.print(100,DEC);Serial.print(" - ");
 Serial.println(mouse.read(),HEX);
 mouse.write(0xF3);Serial.print(0xF3,HEX);Serial.print(" - ");
 Serial.println(mouse.read(),HEX);
 mouse.write(80);Serial.print(80,DEC);Serial.print(" - ");
 Serial.println(mouse.read(),HEX);
 mouse.write(0xF2);Serial.print(0xF2,HEX);Serial.print(" - ");
 Serial.println(mouse.read(),HEX);
 Serial.println(mouse.read(),HEX);
 delay(1000);
}
void setup()
{
 Serial.begin(9600);
 mouse_init();
}
```

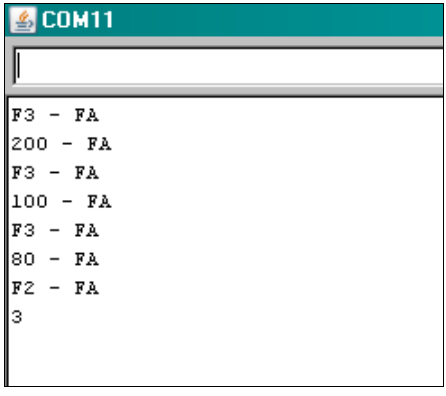


Рис. 9.5. Установка режима трехкнопочной мыши

Как видно из рис. 9.5, последний ответ мыши — байт `0x03` — колесо прокрутки включено.

В функции `loop()` происходит постоянный опрос состояния мыши (отправка байта запроса и чтение 4-х байтов). В случае изменения состояния мыши происходит анализ полученных байтов и отправка в последовательный порт событий для мыши: изменение координат, нажатие клавиш (рис. 9.6). Содержимое функции `loop()` представлено в листинг 9.12.

Листинг 9.12

```
void loop()
{
 char mstat;
 char mx;
 char my;
 char mz;
 /* чтение из мыши */
 mouse.write(0xeb); // запрос
 mouse.read();   //
 mstat = mouse.read();
 mx = mouse.read();
 my = mouse.read();
 mz = mouse.read();
 if(mstat!=0x08) // ничего не произошло
 {
 Serial.print(mstat, BIN);
 if(bitRead(mstat,0)==1)
  Serial.print("Click left button ");
 if(bitRead(mstat,1)==1)
  Serial.print("Click right button ");
 if(bitRead(mstat,2)==1)
  Serial.print("Click middle button ");
 Serial.print("\tX=");
 Serial.print(mx, DEC);
 Serial.print("\tY=");
 Serial.print(my, DEC);
 Serial.print("\tZ=");
 Serial.print(mz, DEC);
 Serial.println();
 delay(300);
 }
}
```

Текст рассмотренного скетча находится в папке examples/_09_2 сопровождающего книгу электронного архива.

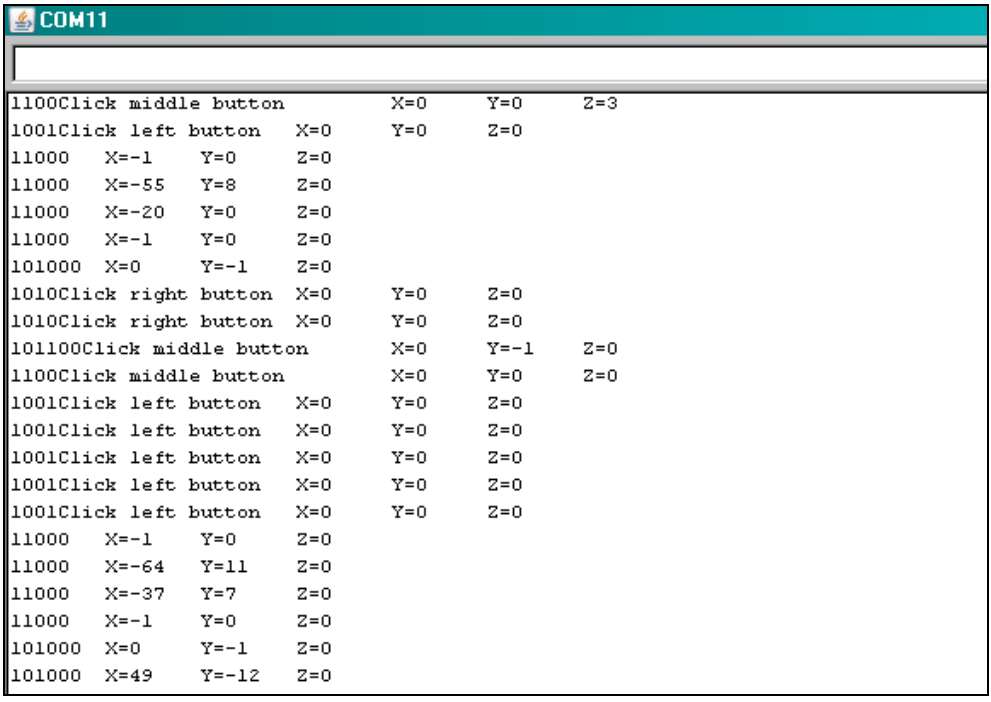


Рис. 9.6. Вывод состояния мыши

ГЛАВА 10

Arduino и сенсорная панель

Сенсорный экран на самом деле не является полноценным экраном с ЖК-дисплеем. Это всего лишь прозрачная панель, которая устанавливается перед соответствующего размера дисплеем так, чтобы процессор мог определить координаты точки на экране, к которой прикасаются в настоящий момент. Если вы хотите чтобы ваш дисплей, используемый в Arduino-проекте, не только отображал информацию на экране, но и давал выбирать ее или управлять ею касанием, придется перед дисплеем поместить сенсорную панель и программно отслеживать точки касания.

Сенсорный экран может быть смонтирован не только на ЖКИ, но и на любой плоской поверхности, так что он отлично подходит для создания маленьких пользовательских панелей управления с "кнопками", напечатанными на поверхности под сенсорным экраном (рис. 10.1). Все что потребуется — это сделать карту X/Y-коор-

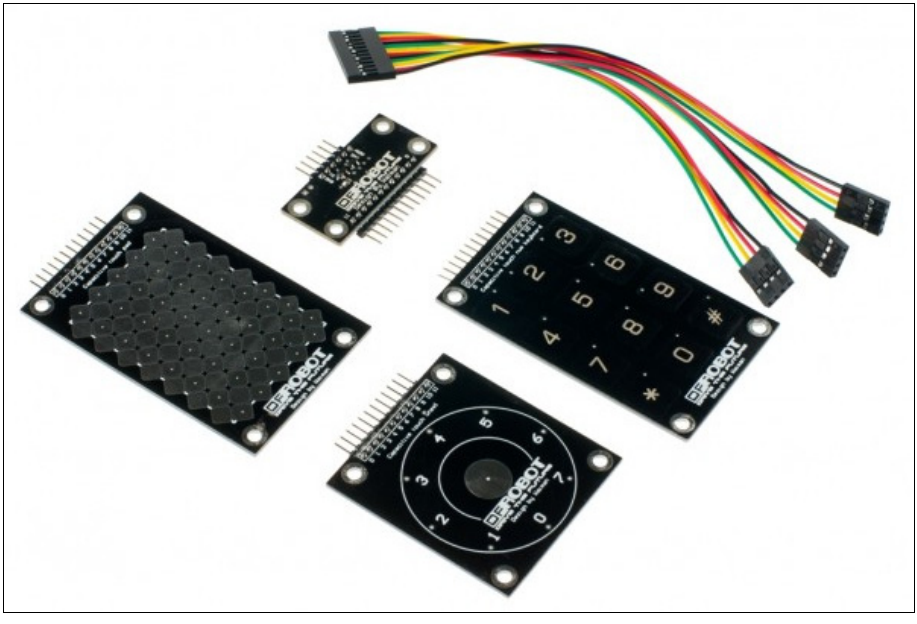


Рис. 10.1. Набор touch-панелей фирмы DFRobot

динат кнопок, и контроллер сможет выяснить, какая кнопка была нажата, путем сопоставления координат. Разумеется, панель управления может представлять не только кнопки, но еще какие-либо органы управления. Можно поместить на нее ползунок, чтобы выбирать громкость звучания медиацентра или назначать температуру в помещении касанием нужного значения на изображенной шкале. Это может быть и план дома, где прикоснувшись к выбранной части поэтажного плана, вы сможете управлять освещением в тех или комнатах.

В настоящее время 4-проводные резистивные сенсорные экраны недороги. В магазине запчастей для сотовых телефонов я приобрел 5-дюймовый сенсорный экран для китайского всего за 300 рублей, панели меньшего размера можно найти и дешевле.

10.1. Как работает резистивный экран?

Резистивная сенсорная панель собрана на стеклянной подложке, которая обеспечивает жесткость всей конструкции. Над этой подложкой расположена тонкая пластиковая пластина с резистивным покрытием. В некоторых сенсорных панелях резистивный слой наносится непосредственно на стеклянную подложку. Над пластиной с резистивным покрытием размещена еще одна пластиковая пластина с микроточками, которые представляют собой крошечные распорки, предназначенные для создания некоторого промежутка между слоями, покрытыми резистивным материалом (рис. 10.2).

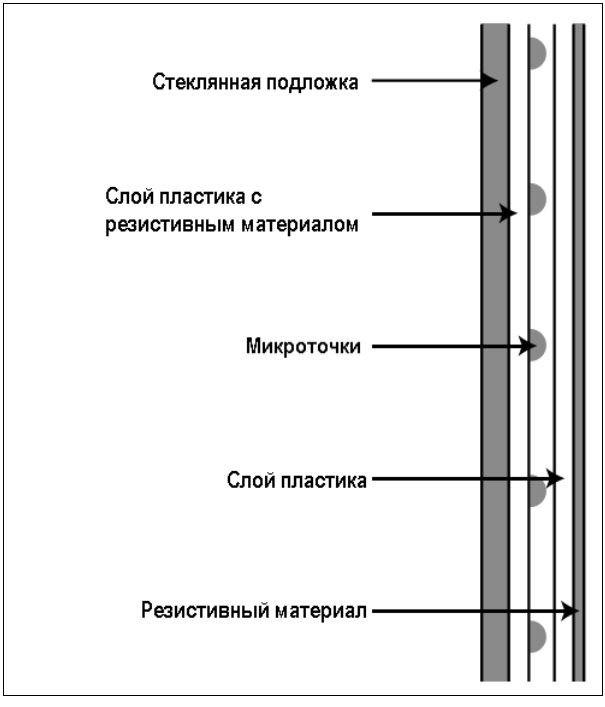


Рис. 10.2. Устройство резистивного сенсорного экрана

Когда к экрану не прикасаются, микроточки отделяют два резистивных слоя друг от друга. Если к верхнему слою прикоснуться, он деформируется и вступает в контакт с нижним слоем, что позволяет проводить электричество из одного слоя в другой. Каждая из четырех сторон сенсорного экрана содержит проводящий электрод, который расположен по всей длине ребра и подключается к резистивной поверхности на верхнем или нижнем слое, совпадающем по оси. Например, левый и правый края могут соединяться с лицевым слоем, в то время как верхняя и нижняя кромки — подключаться к заднему слою (рис. 10.3).

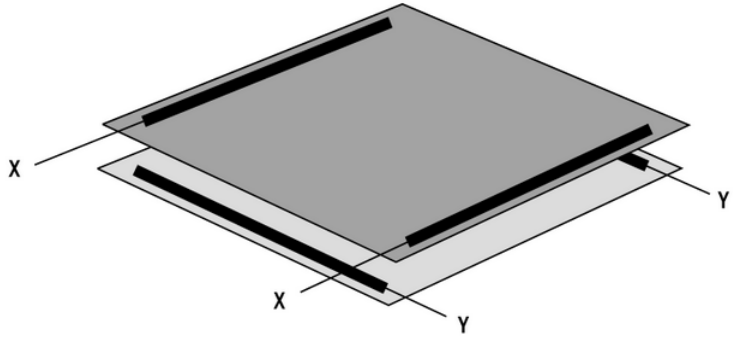


Рис. 10.3. Электроды осей `X` и `Y` в резистивной сенсорной панели

Эти четыре электрода выведены на разъем сенсорной панели как `X1`, `X2`, `Y1` и `Y2`, хотя порядок нумерации на разъеме может варьироваться в зависимости от конкретной марки и модели сенсорного экрана. Выяснить пары соединений можно путем измерения сопротивления между ними мультиметром в то время, когда к экрану не прикасаются. Каждые соответствующие пары будут иметь сопротивление между ними где-то около 1 K или ниже, в то время как несоответствующие электроды будут разомкнуты.

Чтобы прочитать одну ось координаты касания к экрану, один из слоев подключается к GND с одной стороны и к +5 В с другой, что обеспечивает непрерывно изменяющееся напряжение на поверхности этого слоя. Затем один из выводов на другом слое считывается с помощью аналогового ввода для определения относительного положения контакта между двумя краями: если ближе к краю GND, будет читаться меньшее напряжение, а если коснуться ближе к краю с +5 В, то читаемое напряжение будет выше.

Чтение других осей требует переключения между выводами таким образом, чтобы слой, ранее подключенный к аналоговому входу, был переключен на GND и +5 В, а один из электродов другого слоя был включен на аналоговый вход для чтения из него данных.

С Arduino это довольно легко, т. к. выводы аналогового входа на самом деле являются и выводами цифрового ввода/вывода общего назначения, и их режим может переключаться в программе при необходимости. Нам не придется тратить время и линии ввода/вывода для внешней коммутации подключения сенсорной панели к аналоговому входу или GND и +5 В. При подключении выводов `X1`, `X2`, `Y1` и `Y2`

сенсорного экрана прямо на 4 аналоговых входа (рис. 10.4) мы можем использовать программное обеспечение для переключения режимов коммутации между показаниями и использовать их как цифровые выходы, которые обеспечивают GND и +5 В на любом слое так, как требуется в данный момент.

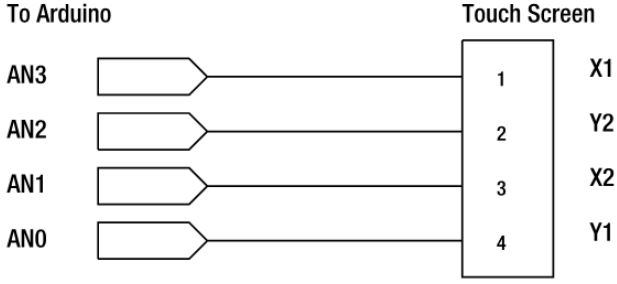


Рис. 10.4. Подключение сенсорного экрана к Arduino

10.2. Программа чтения координат сенсорного экрана

Программа для Arduino, которая просто читает и возвращает значения, очень проста и поможет понять, как работает сенсорный экран. Рассмотрим скетч, читающий данные с сенсорной панели пять раз в секунду и сообщающий эти значения на компьютер по USB.

Основной цикл программы читает значения `X` и `Y` при каждом проходе. При чтении значения `X` вывода аналогового входа, соответствующие оси `Y`, переключаются на цифровой вывод, а затем происходит чтение аналогового значения. Необходимо ввести ожидание 2 мс после настройки выводов для стабилизации установленного напряжения. Добавление задержки в несколько миллисекунд позволяет вводу стабилизироваться и дает более устойчивое чтение.

Содержимое скетча представлено в листинге 10.1.

Листинг 10.1

```
int x = 0;
int y = 0;
void setup()
{
 Serial.begin(9600);
}
void loop()
 {
 pinMode( 15, INPUT );  // Аналоговый вывод 1
 pinMode( 17, INPUT );  // Аналоговый вывод 3
 pinMode( 14, OUTPUT );  // Аналоговый вывод 0
 digitalWrite( 14, LOW ); // Используем вывод 0 как подключение к GND
```

```
 pinMode( 16, OUTPUT );  // Аналоговый вывод 2
 digitalWrite( 16, HIGH ); // Используем вывод 2 как подключение к +5 V
 delay(2);         // для стабилизации напряжения
 x = analogRead( 1 ); // Читаем значение X
 // переключить режимы выводов и сделать то же самое для оси Y.
 pinMode( 14, INPUT );  // Аналоговый вывод 0
 pinMode( 16, INPUT );  // Аналоговый вывод 2
 pinMode( 15, OUTPUT );  // Аналоговый вывод 1
 digitalWrite( 15, LOW ); // Используем вывод 1 как подключение к GND
 pinMode( 17, OUTPUT );  // Аналоговый вывод 3
 digitalWrite( 17, HIGH ); // Используем вывод 3 как подключение к +5 V
 delay(2);         // для стабилизации напряжения
 y = analogRead( 0 ); // Читаем значение Y
 // выводим значения в последовательный порт
 Serial.print(x);
 Serial.print(",");
 Serial.println(y);
 delay (200);
 }
```

10.3. Библиотека Arduino *TouchScreen*

Предыдущий скетч — отличный способ понять, как работать с сенсорными экранами и как взаимодействовать с ними. А чтобы облегчить написание программ, можно использовать библиотеку `TouchScreen`. Эту библиотеку можно найти в папке libraries сопровождающего книгу электронного архива. Скетч, читающий данные с сенсорной панели пять раз в секунду и сообщающий эти значения на компьютер по USB, представлен в листинге 10.2.

Листинг 10.2

```
#include <TouchScreen.h>
TouchScreen ts(3, 1, 0, 2);
void setup()
{
 Serial.begin(9600);
}
void loop()
{
 int coords[2];
 ts.Read(coords);
 Serial.print(coords[0]);
 Serial.print(",");
 Serial.println(coords[1]);
 delay (200);
}
```

ГЛАВА 11

Arduino и 1-Wire

В этой главе мы рассмотрим примеры использования Arduino в связке с устройствами, использующими протокол 1-Wire.

11.1. Что такое 1-Wire?

1-Wire (англ. — один провод) — зарегистрированная торговая марка корпорации Dallas Semiconductor для системотехники шины устройств связи Dallas Semiconductor. Обеспечивает низкоскоростной интерфейс для данных (обычно 16,4 Кбит/с, максимум 125 Кбит/с в режиме overdrive). Для связи с устройством необходимо лишь два провода: на данные и заземление. Для этого интегральная схема включает конденсатор емкостью 800 пФ для питания от линии данных (так называемое *питание от паразитного источника*). Электропитание осуществляется за счет заряда конденсатора, который заряжается во время наличия высокого уровня напряжения на линии данных. Здесь следует учитывать, что связь с устройствами, использующими паразитное питание, возможна только на коротких линиях. На длинных линиях приходится применять дополнительный провод питания.

Протокол регламентирован разработчиками для применения в четырех основных сферах-приложениях:

- приборы в специальных корпусах MicroCAN для решения проблем идентификации, переноса или преобразования информации (технология iButton);
- программирование встроенной памяти интегральных компонентов;
- идентификация элементов оборудования и защита доступа к ресурсам электронной аппаратуры;
- системы автоматизации (технология сетей 1-Wire).

Первое из этих направлений широко известно на мировом рынке и уже давно пользуется заслуженной популярностью. Второе с успехом обеспечивает возможность легкой перестройки функций полупроводниковых компонентов, производимых фирмой Dallas Semiconductor и имеющих малое количество внешних выводов. Третье позволяет обеспечить недорогую, но достаточно эффективную идентификацию и надежную защиту самого разнообразного оборудования. Что касается чет-

вертого применения, то реализация локальных распределенных систем на базе шины 1-Wire является на сегодня де-факто наиболее оптимальным решением для большинства практических задач автоматизации. В настоящее время Dallas Semiconductor поставляет широкую номенклатуру однопроводных компонентов различных функциональных назначений для реализации самых разнообразных сетевых приложений. Поэтому имеется огромное число конкретных примеров использования интерфейса 1-Wire для целей автоматизации в самых различных областях, и все больше разработчиков проявляют интерес к этой технологии.

Так в чем же особенность этого сетевого стандарта? Ведь в качестве среды для передачи информации по однопроводной линии чаще всего возможно использование обычного телефонного кабеля и, следовательно, скорость обмена в этом случае не велика. Однако если внимательно проанализировать большинство объектов, требующих автоматизации, то больше чем для 60 % из них предельная скорость обслуживания в 16,3 Кбит/с будет более чем удовлетворительной. Другие преимущества технологии 1-Wire:

- ❒ простое и оригинальное решение адресуемости абонентов;
- ❒ несложный протокол;
- ❒ простая структура линии связи;
- ❒ малое потребление компонентов;
- ❒ легкое изменение конфигурации сети;
- ❒ значительная протяженность линий связи;
- ❒ исключительная дешевизна всей технологии в целом.

Все эти приемущества отражают очевидную рациональность и высокую эффективность этого инструмента при решении задач комплексной автоматизации в самых различных областях деятельности.

Сеть 1-Wire представляет собой информационную сеть, использующую для осуществления цифровой связи одну линию данных (DATA) и один возвратный (или земляной) провод (RET). Таким образом, для реализации среды обмена этой сети могут быть применены доступные кабели, содержащие неэкранированную витую пару той или иной категории, и даже обычный телефонный провод. Такие кабели при их прокладке не требуют наличия какого-либо специального оборудования, а ограничение максимальной длины однопроводной линии регламентировано разработчиками на уровне 300 м.

Основой архитектуры сетей 1-Wire является топология общей шины, когда каждое из устройств подключено непосредственно к единой магистрали, без каких-либо каскадных соединений или ветвлений. При этом в качестве базовой используется структура сети с одним ведущим (или мастером) и многочисленными ведомыми. Хотя существует ряд специфических приемов организации работы однопроводных систем в режиме мультимастера.

Конфигурация любой сети 1-Wire может произвольно меняться в процессе ее работы, не создавая помех дальнейшей эксплуатации и работоспособности всей систе-

мы в целом, если при этих изменениях соблюдаются основные принципы организации однопроводной шины. Эта возможность достигается благодаря присутствию в протоколе интерфейса 1-Wire специальной команды поиска ведомых устройств (Поиск ПЗУ), которая позволяет быстро определить новых участников информационного обмена. Благодаря наличию в составе любого устройства, снабженного интерфейсом 1-Wire, индивидуального уникального адреса (отсутствие совпадения адресов для компонентов, когда-либо выпускаемых Dallas Semiconductor, гарантируется самой фирмой-производителем), такая сеть имеет практически неограниченное адресное пространство. При этом каждый из однопроводных компонентов сразу готов к использованию в составе сети 1-Wire без каких-либо дополнительных аппаратно-программных модификаций.

Однопроводные компоненты являются самотактируемыми полупроводниковыми устройствами, в основе обмена информацией между которыми лежит управление длительностью импульсных сигналов в однопроводной среде и их измерение. Передача сигналов для интерфейса 1-Wire — асинхронная и полудуплексная, а вся информация, циркулирующая в сети, воспринимается абонентами либо как команды, либо как данные. Команды сети генерируются мастером и обеспечивают различные варианты поиска и адресации ведомых устройств, определяют активность на линии даже без непосредственной адресации отдельных компонентов, управляют обменом данными в сети.

Стандартная скорость работы сети 1-Wire, изначально нормированная на уровне 16,3 Кбит/с, была выбрана, во-первых, исходя из обеспечения максимальной надежности передачи данных на большие расстояния, и, во-вторых, с учетом быстродействия наиболее широко распространенных типов универсальных микроконтроллеров, которые в основном должны использоваться при реализации ведущих устройств шины 1-Wire. Эта скорость обмена может быть снижена до любой возможной благодаря введению принудительной задержки между передачей в линию отдельных битов данных (растягиванию временных слотов протокола). Увеличение скорости обмена в сети 1-Wire выше значения 16,3 Кбит/с приводит к сбоям и ошибкам при работе на магистрали 1-Wire длиной более 1 м. Однако если длина линии 1-Wire не превышает 0,5 м, то скорость обмена может быть значительно увеличена за счет перехода на специальный режим ускоренной передачи (overdrive — до 125 Кбит/с), который допускается для отдельных типов однопроводных компонентов. Как правило, такой режим обмена аппаратно реализован для однопроводных компонентов, имеющих большой объем встроенной памяти, предназначенных для эксплуатации на небольшой, но качественной и не перегруженной другими устройствами линии 1-Wire. Типичным примером таких компонентов являются микросхемы семейства iButton.

При реализации интерфейса 1-Wire используются стандартные КМОП/ТТЛ логические уровни сигналов, а питание большинства однопроводных компонентов может осуществляться от внешнего источника с рабочим напряжением в диапазоне от 2,8 до 6,0 В. Причем такой источник может быть расположен либо непосредственно возле компонента (например, батарея в составе микросхем iButton), либо энергия от него может поступать по отдельной линии магистрали 1-Wire. Альтернативой

к применению внешнего питания служит так называемый *механизм паразитного питания*, действие которого заключается в использовании каждым из ведомых компонентов линии 1-Wire электрической энергии импульсов, передаваемых по шине данных, аккумулируемой затем специальной, встроенной в микросхему, емкостью. Кроме того, отдельные однопроводные компоненты сетей 1-Wire могут использовать особый режим питания по шине данных, когда энергия к приемнику поступает непосредственно от мастера по шине `DATA` магистрали, при этом обмен информацией в сети принудительно прекращается.

11.2. Применение 1-Wire

О признании шины 1-Wire в качестве международного стандарта и серьезности отношения к этому интерфейсу со стороны маститых разработчиков и производителей электроники говорят многочисленные факты. Например, нет практически ни одного универсального микроконтроллера, в литературе по применению которого не обсуждались бы способы организации на его базе мастера линии 1-Wire.

Рационально использование технологии 1-Wire при построении систем автоматизации контроля и управления для разнообразного рассредоточенного оборудования, когда не требуется высокая скорость при обслуживании, но необходима существенная гибкость и наращиваемость при невысоких затратах на реализацию. Приведем некоторые примеры активного применения устройств 1-Wire:

- автоматизация;
- метеостанции;
- многоточечные системы контроля температуры — применяются при мониторинге микроклимата самых различных объектов, при ревизии любых операций по переработке и хранению пищи и медикаментов, для контроля температуры во многих областях промышленного производства, в холодильной технике и т. п.;
- умный дом (Home Automation) — комплекс автоматики, который управляет инженерными системами жилища (освещением, отоплением, вентиляцией, кондиционированием, энергоснабжением, водоснабжением, электроприводами, оборудованием пожарных и охранных систем и пр.);
- идентификация и защита электронных изделий — например, маркирование и защита картриджей для принтеров и копировальных аппаратов, а также любых требующих этого электронных изделий;
- менеджмент автономных источников энергии — строгая и полная идентификация источников энергии, сохранение в памяти встроенного в батарею электронного устройства особенностей ее изготовления и индивидуальных технических характеристик, наиболее полный мониторинг их основных эксплуатационных параметров на протяжении всего срока службы, а также формирование корректного управляющего воздействия, связанного с восстановлением заряда обслуживаемого аккумулятора;
- ограничения и сопряжение с промышленными и глобальными сетями.

Русскоязычную информацию об однопроводных компонентах, наиболее подходящих для организации сетей 1-Wire, можно посмотреть в сети Интернет по адресу **http://www.elin.ru/1-Wire/?topic=component**. Полные списки всех однопроводных устройств и приспособлений для их поддержки можно получить на интернет-сайте компании-производителя Dallas Semiconductor **http://www.maxim-ic.com**

Устройства 1-Wire достаточно широко представлены в российских интернет-магазинах по приемлемым, достаточно низким, ценам.

11.3. Протокол 1-Wire

11.3.1. Описание интерфейса 1-Wire

Интерфейс 1-Wire разработан фирмой Dallas Semiconductor (ныне MAXIM) в конце 1990-х годов. Этот интерфейс интересен тем, что для двустороннего обмена требуется всего одна линия. Правда, еще требуется общий провод (земля) и провод питания (не всегда). Причем, на эту одну линию можно повесить несколько устройств. Протокол очень прост и легко реализуется на микроконтроллере программно. На рис. 11.1 представлена блок-схема аппаратной реализации 1-Wire.

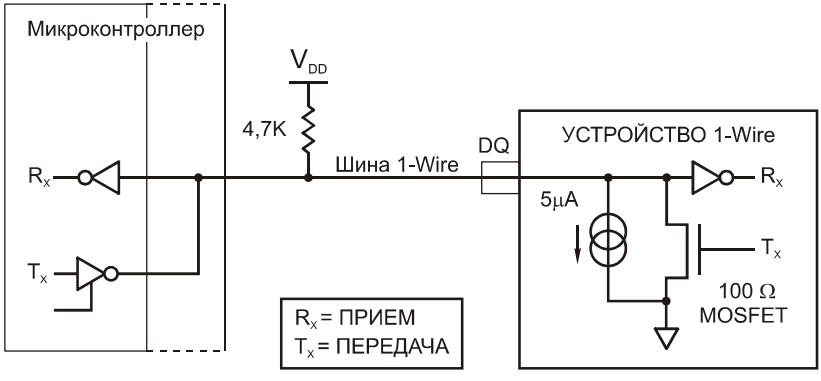


Рис. 11.1. Блок-схема аппаратной реализации 1-Wire

Вывод DQ устройства представляет собой вход КМОП-логического элемента, который может быть зашунтирован (замкнут на общий провод) полевым транзистором. Сопротивление канала этого транзистора в открытом состоянии — около 100 Ом. Когда транзистор заперт — имеется небольшой ток утечки (примерно 5 мкА) на общий провод.

Шина 1-Wire должна быть подтянута отдельным резистором к напряжению питания (может быть от 3 до 5 В — нужно уточнять по характеристикам подключаемого устройства). Сопротивление этого резистора 4,7 К, однако это значение подходит только для достаточно коротких линий. Если шина используется для подключения устройств на большее расстояние, то сопротивление подтягивающего резистора необходимо уменьшить (сопротивление зависит от величины максимального втекающего тока линии DQ конкретного устройства 1-Wire).

Примечательный момент — как уже отмечалось, некоторые устройства 1-Wire могут использовать так называемое *паразитное/фантомное питание* (Parasite power) — при котором питание устройства осуществляется от линии данных за счет заряда встроенного конденсатора, который заряжается во время наличия высокого уровня напряжения на линии данных. Здесь следует учитывать, что связь с устройствами, использующими паразитное питание, допустима только на коротких линиях. На длинных линиях могут присутствовать непонятные побочные эффекты. Поэтому, если возможно, такого типа питания устройств следует избегать.

11.3.2. Обмен информацией по 1-Wire

Рассмотрим, как происходит обмен информацией по 1-Wire.

- Передача информации возможна только выдачей низкого уровня в линию, т. е. замыканием ее на общий провод. В высокий логический уровень линия вернется сама из-за наличия подтягивающего резистора (теперь становится понятно, что наличие внешнего подтягивающего резистора — обязательное условие работы 1-Wire).
- Обмен происходит по инициативе ведущего устройства (обычно — микроконтроллера).
- Обмен информацией начинается с подачи импульса сброса (`RESET pulse`) на линию.
- Устройства 1-Wire предусматривают "горячее" подключение.
- При подключении устройство 1-Wire выдает в линию `DQ` импульс присутствия (`PRESENCE pulse`). Этот же импульс выдается при обнаружении сигнала `RESET`.
- Обмен информацией ведется так называемыми *тайм-слотами* — один слот содержит один бит информации.
- Данные передаются побайтно — бит за битом, начиная с младшего байта. Достоверность данных (проверка отсутствия искажений) гарантируется путем подсчета циклической контрольной суммы (CRC).

Микроконтроллер (МК) формирует импульс `RESET`, переводя в низкий логический уровень шину 1-Wire и удерживая ее не менее 480 микросекунд. Затем МК "отпускает" шину и напряжение возвращается к высокому логическому уровню (время зависит от емкости линии и сопротивления подтягивающего резистора). Протокол 1-Wire ограничивает это время диапазоном от 15 до 60 микросекунд, что и влияет на выбор подтягивающего резистора (на время возврата линии к высокому уровню большее влияние оказывает емкость линии, но, чаще всего, мы изменить ее не можем). Обнаружив импульс `RESET`, ведомое устройство формирует ответный импульс `PRESENCE`. Для этого устройство прижимает линию `DQ` к "земле" и удерживает от 60 до 240 микросекунд. Затем устройство так же "отпускает" шину. После этого устройству еще дается время для завершения внутренних процедур инициализации. Таким образом, МК должен приступить к любому обмену с устройством не ранее, чем через 480 микросекунд после завершения импульса `RESET`. Соответственно процедура инициализации, с которой начинается обмен данными между устройствами, длится минимум 960 микросекунд (рис. 11.2).

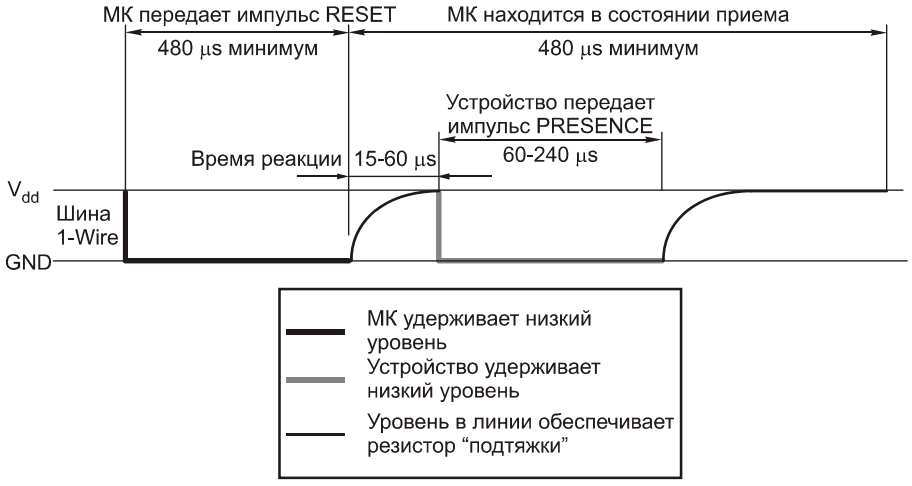


Рис. 11.2. Процедура инициализации

Теперь рассмотрим процедуры обмена битами информации, которые осуществляются определенными тайм-слотами — жестко лимитированными по времени последовательностями смены уровней сигнала в линии 1-Wire (рис. 11.3).

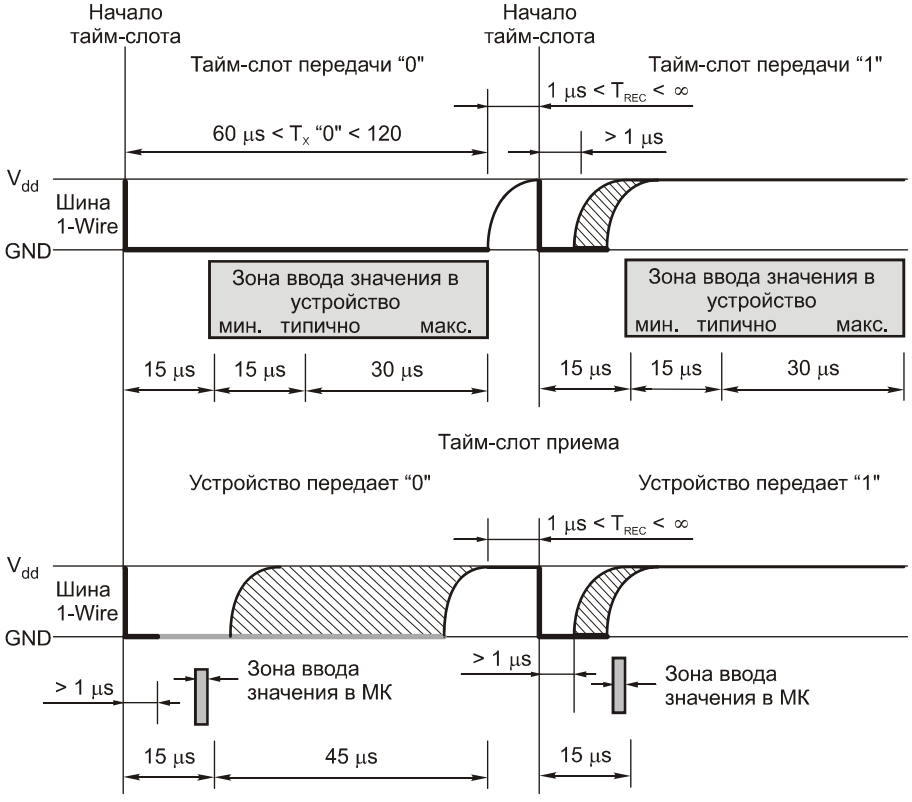


Рис. 11.3. Процедура обмена битами информации

Различают 4 типа тайм-слотов:

- передача "1" от МК;
- передача "0" от МК;
- прием "1" от устройства;
- прием "0" от устройства.

Тайм-слот всегда начинает микроконтроллер, прижимая шину к "земле". Длительность тайм-слота находится в пределах от 60 до 120 микросекунд. Между тайм-слотами всегда должен быть интервал не менее 1 микросекунды (определяется параметрами ведомого устройства). Тайм-слоты передачи отличаются от тайм-слотов приема поведением микроконтроллера:

- при передаче МК только формирует сигналы;
- при приеме МК еще и опрашивает уровень сигнала в линии 1-Wire.

Тайм-слот передачи "0" заключается просто в прижимании шины 1-Wire к "земле" в течение всей длительности тайм-слота. Передача "1" осуществляется путем "отпускания" шины 1-Wire со стороны МК не ранее чем через 1 микросекунду после начала тайм-слота, но не позже чем через 15 микросекунд. Ведомое устройство опрашивает уровень в шине 1-Wire в течение временного интервала (показанного в виде серого прямоугольника), т. е. начиная с 15-й микросекунды от начала тайм-слота и заканчивая 60-й микросекундой от начала (для большинства устройств — около 30-й микросекунды от начала тайм-слота).

Заштрихованная на рис. 11.3 область — это область "нарастания" уровня в шине 1-Wire, которая зависит от емкости линии и сопротивления подтягивающего резистора. Тайм-слоты приема информации отличаются тем, что МК формирует только начало тайм-слота (так же, как при передаче "1"), а затем управление уровнем шины 1-Wire берет на себя устройство, а МК осуществляет ввод этого уровня так же в определенной зоне временных интервалов. Зона эта, как видно из рис. 11.3, довольно мала. Поскольку заштрихованная область — область неопределенности, то для ввода микроконтроллеру остается даже не промежуток, а скорее конкретный момент, когда он должен ввести уровень сигнала из линии. Этот момент времени — 14-я или 15-я микросекунда от начала тайм-слота.

Резюмируем:

- микроконтроллер начинает тайм-слот, прижимая шину 1-Wire к логическому "0" в течение 1 микросекунды;
- последующий уровень зависит от типа тайм-слота: для приема и передачи "1" уровень должен стать высоким, а для передачи "0" — оставаться низким вплоть до конца тайм-слота, т. е. от 60 до 120 микросекунд;
- принимая данные, МК должен считать уровень в шины 1-Wire в промежутке от 13-й до 15-й микросекунде тайм-слота;
- микроконтроллер должен обеспечить интервал между тайм-слотами не менее 1 микросекунды (лучше — больше, максимальное значение не ограничено).

Для достижения требуемых временных интервалов надо следовать простым правилам:

- все сигналы, которые должен формировать МК, следует формировать по принципу необходимого минимума длительности (т. е. немного больше, чем указанная минимальная длительность);
- от устройства следует ожидать сигналов по принципу наихудшего (т. е. ориентироваться на самые худшие варианты временных параметров сигнала).

11.3.3. Протокол обмена информацией 1-Wire

Теперь рассмотрим протокол обмена информацией 1-Wire.

Каждое устройство 1-Wire обладает уникальным идентификационным 64-битным номером, программируемым на этапе производства микросхемы:

- первые 8 битов — номер серии устройства (список некоторых кодов семейств 1-Wire-устройств приведен в табл. 11.1);
- следующие 48 битов — уникальный серийный номер;
- последние 8 битов — CRC-код предыдущих 56 битов информации.

Таблица 11.1. *Список некоторых кодов семейств устройств 1-Wire*

HEX-код устройства	Устройства	Описание
01	DS1990	Уникальный серийный номер-ключ
04	DS1994	4 Кбайт энергонезависимого ОЗУ + часы, таймер и будильник
05	DS2405	Одиночный адресуемый ключ
06	DS1993	4 Кбайт энергонезависимого ОЗУ
08	DS1992	1 Кбайт энергонезависимого ОЗУ
09	DS2502	1 Кбайт электрически программируемого ПЗУ
0A	DS1995	16 Кбайт энергонезависимого ОЗУ
0B	DS2505	16 Кбайт EEPROM
0F	DS2506	64 Кбайт EEPROM
10	DS18S20	Цифровой датчик температуры
12	DS2406	1 Кбайт EEPROM + двухканальный адресуемый ключ
1A	DS1963L	4 Кбайт энергонезависимого ОЗУ со счетчиком числа циклов записи
1D	DS2423	4 Кбайт энергонезависимого ОЗУ с внешним счетчиком
20	DS2450	Четырехканальный АЦП
24	DS2415	Часы реального времени
28	DS18B20	Цифровой датчик температуры
2C	DS2890	Одноканальный цифровой потенциометр
30	DS2760	Датчик температуры, тока, АЦП

Сказанное относится не только к нашему ключу-таблетке, а ко всем устройствам 1-Wire. Фирма-производитель гарантирует, что не найдется двух микросхем с одинаковым идентификационным номером. Нетрудно посчитать, что устройств одного типа может быть выпущено 281 474 976 710 655 (десятичное представление 0xFFFFFFFFFFFF — 48 бит, или 6 байтов идентификационного номера).

Предположим, что на шине 1-Wire имеется более одного устройства. В этом случае перед микроконтроллером встают две проблемы: определение количества имеющихся устройств и выбор (адресация) одного конкретного из них для обмена данными. Номера некоторых устройств наносятся прямо на корпус микросхем (например, для ключей-таблеток — iButton), а номера других можно определить при помощи специальных программ или устройств. Итак, предположим, что мы знаем номера всех устройств 1-Wire на шине. Алгоритм работы с ними следующий:

1. Микроконтроллер посылает импульс `RESET`, и все имеющиеся устройства выдают `PRESENCE`.
2. Микроконтроллер посылает в шину команду, которую принимают все устройства. Определено несколько общих команд для всех типов устройств 1-Wire, есть также и команды, уникальные для отдельных типов устройств.
3. После того как микроконтроллер выдаст команду `READ ROM`, от устройства поступит 8 байтов его собственного уникального адреса — микроконтроллер должен их принять. Любая процедура обмена данными с устройством должна быть завершена полностью либо прервана посылкой сигнала `RESET`.
4. Если отправлена команда `MATCH ROM`, то после нее микроконтроллер должен передать 8 байтов адреса конкретного устройства, с которым будет осуществляться последующий обмен данными.
5. Приняв эту команду, каждое устройство сравнивает передаваемый адрес со своим собственным. Все устройства, адрес которых не совпал, прекращают анализ и выдачу сигналов в линии 1-Wire, а опознавшее адрес устройство продолжает работу. Теперь все данные, передаваемые МК, будут попадать только к этому "адресованному" устройству.
6. Если устройство одно на шине — можно ускорить процесс взаимодействия с ним при помощи команды `SKIP ROM`. Получив эту команду, устройство сразу считает адрес совпавшим, хотя никакого адреса за этой командой не следует. Некоторые процедуры не требуют приема от устройства никаких данных, в этом случае команду `SKIP ROM` можно использовать для передачи какой-то информации сразу всем устройствам — например, для одновременного запуска цикла измерения температуры несколькими датчиками-термостатами типа DS18S20.

Общие команды для всех типов устройств 1-Wire представлены в табл. 11.2.

Прием и передача байтов всегда начинается с младшего бита. Порядок следования байтов при передаче и приеме адреса устройства так же ведется от младшего к старшему. Порядок передачи другой информации зависит от конкретного устройства.

Таблица 11.2. *Общие команды для всех типов устройств 1-Wire*

Команда	Значение байта	Описание
SEARCH ROM	0xF0	Поиск адресов — используется при универсальном алгоритме определения количества и адресов подключенных устройств
READ ROM	0x33	Чтение адреса устройства — используется для определения адреса единственного устройства на шине
MATCH ROM	0x55	Выбор адреса — используется для обращения к конкретному адресу устройства из многих подключенных
SKIP ROM	0xCC	Игнорировать адрес — используется для обращения к единственному устройству на шине, при этом адрес устройства игнорируется (можно обращаться к неизвестному устройству)

Алгоритм поиска устройств 1-Wire следующий:

1. Поиск начинается с импульса RESET от ведущего устройства и принятия PRESENCE от ведомых.
2. Затем посылается 1 байт команды:
   - 0xF0 — осуществляется поиск всех устройств на линии;
   - 0xEC — поиск среди устройств, находящихся в состоянии тревоги (alarm state).
3. Устройства отправляют первый бит своего уникального номера. Если несколько устройств передают свой бит одновременно, результирующий бит на линии получится как результат операции логического И (AND).
4. Следующий бит, который отправляют устройства, — это дополнение первого бита (если первый бит был 1, то будет 0 и наоборот: если был 0 — теперь будет 1). На основании этих двух битов ведущее устройство может сделать вывод о первом бите устройств на линии.
5. Далее микроконтроллер отправляет принятый бит назад. И теперь продолжат работу только те ведомые устройства, у которых этот бит установлен. Если же устройство такого бита не имеет, оно должно перейти в режим ожидания до следующего сигнала RESET.
6. Данная "двубитная передача" повторяется для всех следующих 63 битов ROM.
7. Все устройства на линии, кроме одного, перейдут в состояние ожидания, а код ROM этого единственного устройства будет известен.

11.4. Библиотека *OneWire*

В версии Arduino 1.0 произошли изменения — функции send() и receive() были заменены на функции read() и write().

Набор функций библиотеки OneWire:

- begin();
- requestFrom();

- `beginTransmission()`;
- `endTransmission()`;
- `write()`;
- `available()`;
- `read()`;
- `onReceive()`;
- `onRequest()`.

Библиотека `OneWire` находится в папке libraries/OneWire сопровождающего книгу электронного архива.

11.4.1. Функция *begin()*

Функция `begin()` инициализирует библиотеку `OneWire` и присоединяет устройство к шине I2C как master или как slave.

Синтаксис функции `begin()`:

```
begin()
begin(address)
```

Параметр: `address` — 7-битный адрес slave-устройства. Если параметр не указан, устройство подсоединяется к шине как master.

Возвращаемого значения нет.

11.4.2. Функция *requestFrom()*

Функция `requestFrom()` запрашивает байты с другого устройства. Байты могут быть получены при помощи функций `available()` и `receive()`.

Синтаксис функции `requestFrom()`:

```
requestFrom(address, quantity)
```

Параметры:

- `address` — 7-битный адрес устройства, у которого запрашиваем данные;
- `quantity` — число запрашиваемых байтов.

Возвращаемого значения нет.

11.4.3. Функция *beginTransmission()*

Функция `beginTransmission()` начинает передачу на slave-устройство с установленным адресом. Впоследствии необходимо поставить в очередь байты с помощью функции `write()` и передать их вызовом `endTransmission()`.

Синтаксис функции `beginTransmission()`:

```
beginTransmission(address)
```

Параметр: `address` — 7-битный адрес устройства, к которому передаются данные.

Возвращаемого значения нет.

11.4.4. Функция *endTransmission()*

Функция `endTransmission()` завершает передачу на slave-устройство, которая была начата командой `beginTransmission()`, и передает очередь байтов, установленных функцией `write()`.

Синтаксис функции `endTransmission()`:

```
beginTransmission()
```

Параметров нет.

Возвращаемого значения нет.

11.4.5. Функция *write()*

Функция `write()` отправляет данные со slave-устройства в ответ на запрос master или создает очередь для передачи от master к slave-устройству (между вызовами `beginTransmission()` и `endTransmission()`).

Синтаксис функции `write()`:

```
write(value)
write(string)
write(data,quantity)
```

Параметры:

- `value` — байт для передачи (byte);
- `string` — строка для передачи (char*);
- `data` — массив байтов для передачи (byte*);
- `quantity` — количество байтов для передачи.

Возвращаемого значения нет.

11.4.6. Функция *available()*

Функция `available()` возвращает число байтов, доступных для получения функцией `receive()`, которая должна быть вызвана master-устройством после вызова `requestFrom()` или slave-устройством внутри `onReceive()`.

Синтаксис функции `available()`:

```
available()
```

Параметров нет.

Возвращаемое значение: число байтов, доступных для чтения.

11.4.7. Функция *read()*

Функция `read()` получает байты, которые были переданы от slave к master после вызова `requestFrom()` или переданы от master к slave.

Синтаксис функции `read()`:

```
read()
```

Параметров нет.

Возвращаемое значение: следующий переданный байт.

11.4.8. Функция *onReceive()*

Функция `onReceive()` регистрирует функцию, которая должна быть вызвана, когда slave получит данные от master.

Синтаксис функции `onReceive()`:

```
onReceive(handler)
```

Параметр: `handler` — вызываемая после получения данных функция, она должна иметь численный параметр `int` и ничего не возвращать.

Возвращаемого значения нет.

11.4.9. Функция *onRequest()*

Функция `onRequest()` регистрирует функцию, которая должна быть вызвана, когда master запросит данные от slave.

Синтаксис функции `onRequest()`:

```
onRequest (handler)
```

Параметр: `handler` — функция, которая будет вызвана. Она не должна иметь параметров и не должна ничего не возвращать.

Возвращаемого значения нет.

11.5. Устройство iButton и программирование электронного замка

Приступим к рассмотрению практических примеров использования устройств 1-Wire в связке с Arduino. Создадим открыватель электромагнитного замка, в качестве ключей которого используем ключи-таблетки от домофона (рис. 11.4). Это устройства из семейства 1-Wire — iButton DS1990. В силу невысокой стоимости, надежности и простоты считывающих устройств этот тип устройств iButton (DS1990) получил массовое внедрение в качестве электронного ключа в системах охранной сигнализации, разграничения доступа к информации и физическим объектам, электронным проходным, электронным замкам и в системах безопасности для банков и офисов. Применяется оно также для маркировки объектов и маршру-

тов в системах контроля патрульно-постовой службы, контроля передвижения транспорта, системах инвентарного и складского учета и в качестве кредитных карт в локальных платежных системах. Итак, наш ключ хранит 64 бита уникальной информации, которые нанесены на самом ключе (см. рис. 11.4):

- код устройства 01;
- уникальный 48-битный код устройства;
- CRC-код предыдущих 56 битов информации.

Питается наш открыватель от напряжения от 2,8 до 6,0 В. К центральной контактной площадке подключается линия данных, а к боковой каемке — "земля". Специально для нашего проекта приобретена контактная площадка (рис. 11.5). Ключ может работать в диапазоне температур от –40 °С до +85 °С. Величина подтягивающего резистора рекомендуется в 2,2 кОм.

Рис. 11.4. Ключи iButton

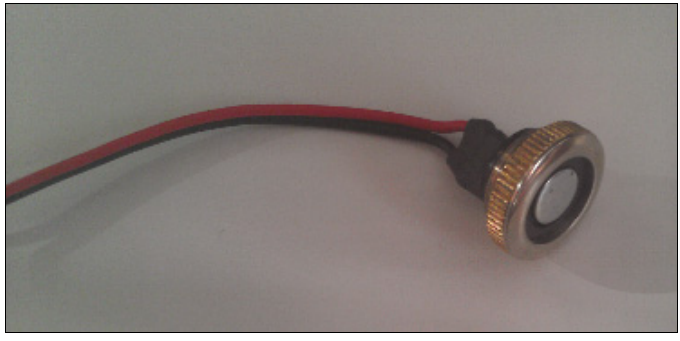


Рис. 11.5. Контактная площадка для устройств iButton

Один из ключей предназначен для использования в качестве ключа администратора, с его помощью мы станем добавлять или удалять ключи в список разрешенных ключей. Коды ключей будут добавляться в память EEPROM. При поднесении к контактной площадке ключа из списка электромагнитный замок на двери откроется. Предусмотрим проигрывание двух различных мелодий при считывании кодов ключа (одну — при открывании замка, вторую — при отказе в доступе). Для про-

ведения операций в режиме администратора (добавление или удаление ключей из памяти) добавим две кнопки. Внешний вид устройства в сборе представлен на рис. 11.6.

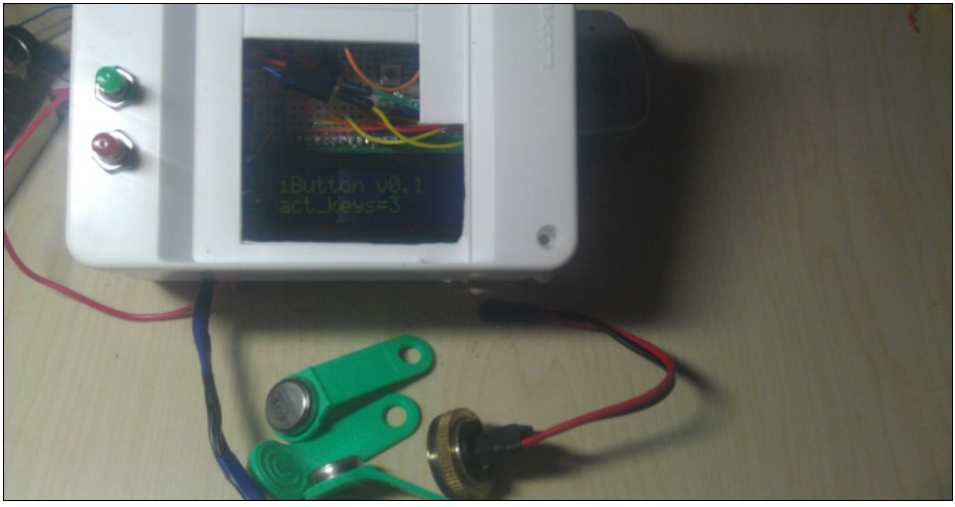


Рис. 11.6. Вид устройства в сборе

11.5.1. Поиск устройств 1-Wire и получение уникального кода

Для начала напишем скетч, который постоянно опрашивает линию для подключения ключей iButton (контактную площадку) и в случае прикосновения ключа-таблетки к устройству считывает уникальный код из устройства, проверяет его CRC (контрольную сумму) и выводит 64-битный код побайтно в монитор последовательного порта. В скетче используется библиотека `1-Wire` (библиотека `OneWire` находится в папке libraries/OneWire сопровождающего книгу электронного архива). Код данного фрагмента представлен в листинге 11.1.

Листинг 11.1

```
#include <OneWire.h>
OneWire ds(10); // используем pin 10

void setup(void) {
 Serial.begin(9600);
}
void loop(void) {
 byte i;
 byte present = 0;
 byte data[12];
 byte addr[8];
```

```
 // поиск устройств
 if ( !ds.search(addr)) {
   Serial.print("No more addresses.\n");
   ds.reset_search();
   return;
 }
 // вывод кода
 Serial.print("R=");
 for( i = 0; i < 8; i++) {
  Serial.print(addr[i], HEX);
  Serial.print(" ");
 }
 // проверка CRC
 if ( OneWire::crc8( addr, 7) != addr[7]) {
   Serial.print("CRC is not valid!\n");
   return;
 }
 // проверка кода устройства
 if ( addr[0] != 0x01) {
   Serial.print("Device is not a DS1990A family device.\n");
   return;
 }
 Serial.println();
 ds.reset();
 delay(1000);
}
```

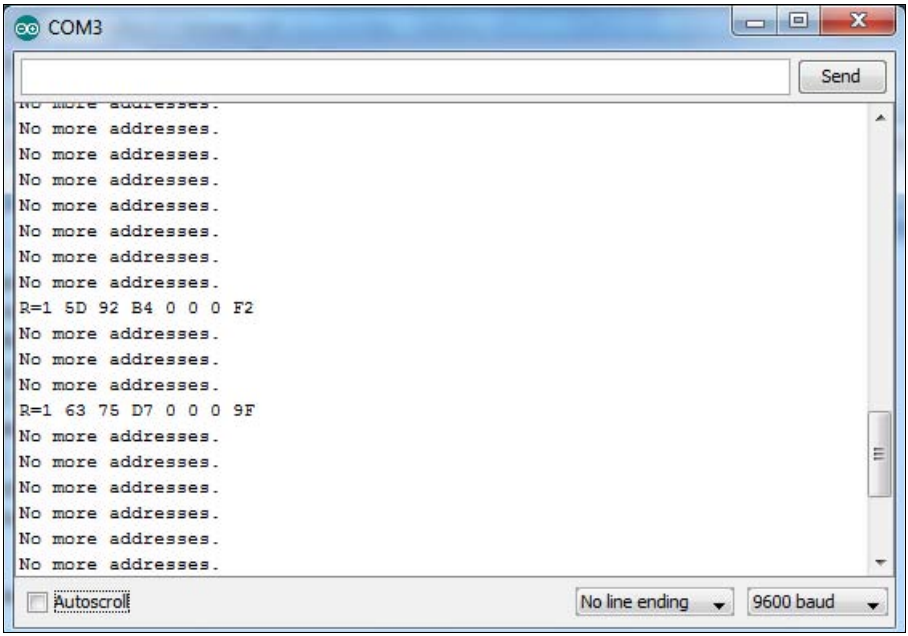


Рис. 11.7. Выдача в последовательный порт

Здесь происходит постоянный поиск устройств на линии 1-Wire. Пока ключ не поднесен к контактной площадке, в монитор последовательного порта выдается сообщение: **No more addresses**.

При подключении ключа-таблетки к контактной площадке — ее номер считывается и выдается в последовательный порт (рис. 11.7). Только выведен этот номер наоборот — как и положено по протоколу 1-Wire — начиная с младшего байта. Сравните выдаваемый код с кодом, нанесенным на корпусе ключей.

11.5.2. Режимы работы

В нашем устройстве предусмотрено два режима работы:

- режим пользователя — дает пользователю возможность открыть замок, приложив к считывающей площадке устройства ключ ibutton из списка допуска;

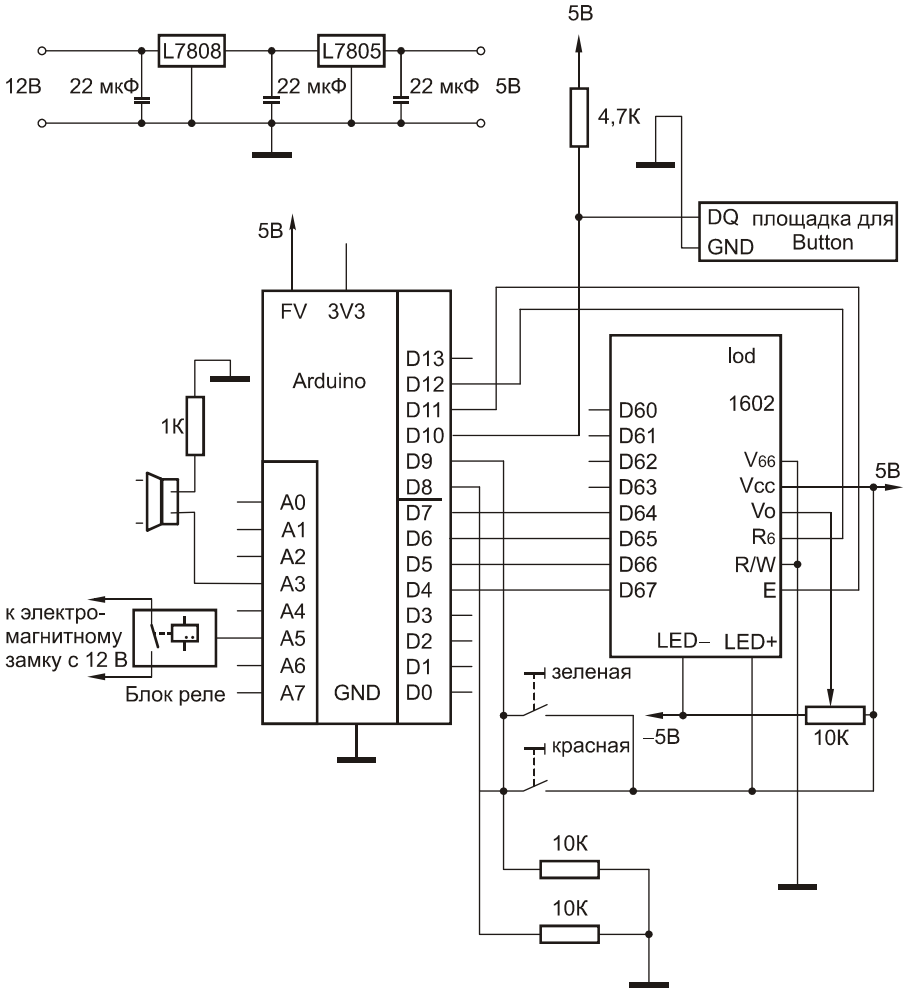


Рис. 11.8. Схема подключения

- режим администратора — позволяет удалять ключи из списка допуска и вносить ключи в список допуска.

Для аппаратной реализации режима администратора применим две кнопки и жидкокристаллический дисплей WH1602. Для подключения дисплея задействуем выводы Arduino 4–7, 11, 12. Для подключения кнопок используем выводы 8 и 9. Электрическая схема представлена на рис. 11.8.

Список ключей допуска будем хранить в памяти EEPROM. Карта распределения памяти EEPROM в нашем скетче такова: вся память распределена на блоки по 8 байтов. В первом байте хранится количество ключей в системе, следующие 7 байтов не используются. Далее каждые 8 байтов отводятся под 64-битный уникальный код каждого ключа. Байты 8–15 отводятся под ключ администратора. Следующие байты — под коды ключей для доступа. Пустые байты заполнены значением `0xFF`.

11.5.3. Режим пользователя

В режиме пользователя осуществляется проверка ключа на наличие в списке разрешенных. Проверка подключения ключа происходит в цикле `loop()`. При поднесении ключа к площадке сверяем его код с кодами разрешенных ключей из памяти EEPROM. Функция `prov_key()` (листинг 11.2) возвращает либо номер ключа в списке, либо 0 при его отсутствии там. Сообщение о результате проверки выводится на дисплей.

Листинг 11.2

```
// ************* проверка наличия ключа в EEPROM
int prov_key()
 {
 byte i1; byte i2;
 int count11;
 for( i1 = 1; i1 <= count; i1++)
 {count11=0;
 for( i2 = 0; i2 < 8; i2++) {
  if(addr[i2]==EEPROM.read(i1*8+i2))
   count11++;
 }
 if(count11==8)
  return i1;
 }
 return 0;
 }
```

11.5.4. Первоначальный ввод ключа в пустую базу

Процедура `setup()` проверяет память EEPROM для определения количества ключей в системе (листинг 11.3). Количество ключей выводится на дисплей.

Листинг 11.3

```
void setup(void) {
 byte val1;
 ... ... ...
 for(int i1=8;i1<512;)
  { val1=EEPROM.read(i1);
  Serial.println(val1,HEX);
  if(val1==0xFF)
   break;
  else
   count++;;
  i1=i1+8;}
 EEPROM.write(address1,count);
 if(count==0)
  {admin=2;type1=1;}
 ... ... ...
 lcd.begin(16, 2);
 lcd.clear();
 // Сообщение при загрузке
 lcd.print("iButton v0.1");
 lcd.setCursor(0,7);
 lcd.print("act_keys=");
 lcd.print(EEPROM.read(address1));
 ... ... ...
}
```

При первоначальном запуске устройства список ключей в системе пуст, и первый введенный ключ будет ключом администратора. При этом 64-битный код заносится в память EEPROM (8 байтов), счетчик количества ключей увеличивается на 1, и на дисплей выводится сообщение о новом ключе и код нового ключа. Данные действия выполняет процедура `add_new_key()`, содержимое которой представлено в листинге 11.4.

Листинг 11.4

```
// ****** добавление ключа в EEPROM
 void add_new_key(int nnew)
 {
 byte i1;
 if ( OneWire::crc8( addr, 7) != addr[7]) {
   Serial.print("CRC is not valid!\n");
   return;
 }
 if ( addr[0] != 0x01) {
   Serial.print("Device is not a DS1990A family device.\n");
```

```
   return;
 }
 count++;
 lcd.setCursor(0,0);
 lcd.print("key_new = ");
 lcd.print(count,HEX);
 for( i1 = 0; i1 < 8; i1++) {
  lcd.setCursor((7-i1)*2,1);
  lcd.print(addr[i1],HEX);
  EEPROM.write(nnew*8+i1, addr[i1]);
 }
 ds.reset();
 delay(1000);
 }
```

11.5.5. Просмотр, запись и удаление кодов ключей в режиме администратора

При входе с ключом администратора мы получаем права администратора. Выбор режима и проведение операций в режиме администратора осуществляется с помощью двух кнопок (см. рис. 11.6): красной (нижняя) и зеленой (верхняя). Кнопки подсоединены к входам Arduino 8 и 9. Проверка кнопок на нажатие проходит в процедуре `loop()`. Проверка на дребезг производится программно. Код данного фрагмента представлен в листинге 11.5.

Листинг 11.5

```
void loop(void) {
 byte i;
 int res1;
 // чтение pin8
 if(digitalRead(8)!=click11) // изменение положения кнопки
  {click11=1-click11;
  endmillis11=millis();}
 // чтение pin9
 if(digitalRead(9)!=click12) // изменение положения кнопки
  {click12=1-click12;
  endmillis12=millis();}
 if(click11==1 && millis()-endmillis11>5000) // нажата более 5 сек
  {
  clear_EEPROM();
  lcd.clear();
  lcd.print("iButton v0.1");
  lcd.setCursor(0,7);
  lcd.print("act_keys=0");
  admin=1;type1=1;
  }
```

```
 if(admin==1)
 {// режим программирования - проверяем кнопки
 // чтение pin8
 if(click11==1 && millis()-endmillis11>1000 && count>0) // более 1 сек
  {type1=0;
  endmillis11=millis();
  pos11++;
  if(pos11>count)
   pos11=min(2,count);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(" tekkey= ");
  view_key1(pos11);
  }
 else
  ;
 // чтение pin9
  // нажата более 1 сек
 if(click12==1 && millis()-endmillis12>1000 && type1==2)
 // подтверждение удаления
  {if(pos11>1)
   delete_key(pos11);
  else
   admin_display(" empty ",0);
  type1=0;
  endmillis12=millis();
  admin_display("admin = ",0);
  view_key1(1);
  }
 else if(click12==1 && millis()-endmillis12>1000 && type1==1)
 // режим - удалить?
  {
  type1=2;
  endmillis12=millis();
  admin_display("delete ? ",0);
  view_key1(pos11);
  }
 else if(click12==1 && millis()-endmillis12>1000 && type1==0)
 // вход в режим добавления ключей
  {
  type1=1;
  endmillis12=millis();
  admin_display("add key ",0);}
 else
  ;
 }
 ... ... ...
}
```

При нажатии на кнопку, подсоединенную к входу 8, вызывается процедура `view_key1()` последовательного просмотра кодов всех введенных ключей. Содержимое процедуры представлено в листинге 11.6. Коды выводятся на дисплей.

Листинг 11.6

```
// ****** вывод ключа из EEPROM
 void view_key1(int rr)
 {
 byte i;
 lcd.setCursor(8,0);
 lcd.print(rr);
 for( i = 0; i < 8; i++) {
  lcd.setCursor((7-i)*2,1);
  lcd.print(EEPROM.read(rr*8+i),HEX);
 }
 delay(1000);
 return;
 }
```

Нажатие на кнопку, подсоединенную к входу 9, переводит программу последовательно в режимы ввода новых ключей, выбора ключа для удаления и подтверждения удаления ключа. На дисплей выводится сообщение о текущем режиме. Если устройство находится в режиме добавления ключей в список разрешенных, при считывании ключа его код заносится в память EEPROM, счетчик количества ключей увеличивается на 1, и на дисплей выводится сообщение о добавлении и код добавленного ключа. Код функции `add_new_key()`, отвечающей за добавление ключа в список разрешенных, представлен в листинге 11.7.

Листинг 11.7

```
// добавление ключа в EEPROM
 void add_new_key(int nnew)
 {
 byte i1;
 if ( OneWire::crc8( addr, 7) != addr[7]) {
   Serial.print("CRC is not valid!\n");
   return;
 }
 if ( addr[0] != 0x01) {
   Serial.print("Device is not a DS1990A family device.\n");
   return;
 }
 count++;
 Serial.print("new=");
 lcd.setCursor(0,0);
```

```
 lcd.print("new=");
 lcd.setCursor(4,0);
 lcd.print(count,HEX);
 for( i1 = 0; i1 < 8; i1++) {
  Serial.print(addr[i1], HEX);
  Serial.print(" ");
  lcd.setCursor((7-i1)*2,1);
  lcd.print(addr[i1],HEX);
  EEPROM.write(nnew*8+i1, addr[i1]);
 }
 ds.reset();
 delay(1000);
 }
```

Если устройство находится в режиме удаления, при подтверждении удаления удаляется код выбранного ключа из EEPROM, производится сдвиг кодов на место удаленного, счетчик количества ключей уменьшается на 1, и на дисплей выводится сообщение об удалении. Код функции `delete_key()`, отвечающей за удаление ключа из списка разрешенных, представлен в листинге 11.8.

Листинг 11.8

```
// удаление ключа из EEPROM
void delete_key(int nn)
{
byte val1;
for(int i1=nn;i1<=count;i1++)
 {
 for(int i2=0;i2<8;i2++)
   {
   val1=EEPROM.read((i1+1)*8+i2);
   Serial.print(i1*8+i2,DEC);
   Serial.print(" ");
   Serial.print(val1,HEX);
   Serial.println();
   EEPROM.write(i1*8+i2,val1);
   }
 }
 count--;
}
```

11.5.6. Блок-реле. Открывание замка

В качестве устройства доступа используется электромеханическая защелка AT-ES01, которая открывается при подаче на контакты напряжения 12 В. В качестве коммутирующего устройства взят блок-реле (рис. 11.9): питание 5 В, коммута-

ция до 250 В, 10А. Блок имеет 4 контакта: GND, Vcc, NG, CSI. Для замыкания контактов реле необходимо подать 1 на вывод SCI.

Управлять реле будем с вывода Arduino A5, который в данном случае используется в качестве цифрового вывода:

```
int relayPin = 18;
pinMode(relayPin, OUTPUT);
```

Для открывания замка подаем на вывод 18 — 1, для закрывания — 0.

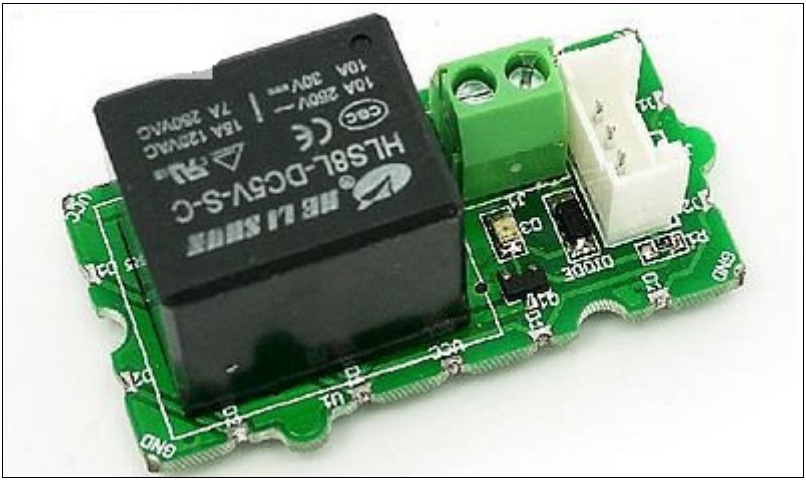


Рис. 11.9. Блок-реле

11.5.7. Проигрывание мелодий

Создадим звуковое сопровождение для вариантов удачной и неудачной аутентификации с помощью ключа. Вывод мелодий на динамик мы рассмотрели в примере к *главе 8*. На динамик подается сигнал соответствующей частоты. Используется выход Arduino 17.

```
// подключить динамик к pin 17
int speakerPin = 17;
pinMode(speakerPin, OUTPUT);
```

Код функций воспроизведения звука представлен в листинге 11.9.

Листинг 11.9

```
// мелодия YES
void playYES()
 {
 for (int i = 0; i < sizeof(notes1); i++)
 {
 if (notes1[i] == ' ')
```

```
  //tone(speakerPin,0, beats1[i]*tempo1* 1000L); // пауза
  delay(beats1[i] * tempo1); // пауза
 else
  playNote(notes1[i], beats1[i] * tempo1);
 }
 }
// мелодия NO
void playNO()
 {
 for (int i = 0; i < sizeof(notes2); i++)
 {
 if (notes2[i] == ' ')
  //tone(speakerPin,0, beats1[i]*tempo2* 1000L); // пауза
  delay(beats2[i] * tempo2); // пауза
 else
  playNote(notes2[i], beats2[i] * tempo2);
 } }
// проиграть ноту
void playNote(char note, int duration)
 {
 // массив для наименований нот (до ре ми ... и т. д. в пределах двух
 // октав)
 char names[]={'c','d','e','f','g','a','b','C','D','E','F','G','A','B'};
 int tones[]={3830,3400,3038,2864,2550,2272,2028,1912,1700,
       1518,1432,1276,1136,1014 };
 // проиграть тон, соответствующий ноте
for (int i = 0; i < sizeof(tones); i++)
 {
 if (names[i] == note)
 playTone(tones[i], duration);
 }
}
void playTone(int tone, int duration) {
 for (long i = 0; i < duration * 1000L; i += tone * 2) {
  digitalWrite(speakerPin, HIGH);
  delayMicroseconds(tone);
  digitalWrite(speakerPin, LOW);
  delayMicroseconds(tone);
 }
}
```

Полностью скетч для рассмотренного проекта находится в папке examples/_11_1 сопровождающего книгу электронного архива.

ГЛАВА 12

Arduino и цифровой датчик температуры DS18B20

Рассмотрим взаимодействие Arduino с цифровым датчиком температуры DS18B20, работающим по протоколу 1-Wire. Его мы будем использовать при разработке домашней метеостанции на Arduino (см. *главу 14*).

12.1. Описание датчика DS18B20

DS18B20 (рис. 12.1) — цифровой термометр с программируемым разрешением от 9 до 12 битов, которое может сохраняться в памяти EEPROM прибора. DS18B20 обменивается данными по шине 1-Wire и при этом может быть как единственным устройством на линии, так и работать в группе. Все процессы на шине управляются центральным микропроцессором.

Диапазон измерений датчика: от –55 до +125 °C и точностью 0,5 °C в диапазоне от –10 до +85 °C. В дополнение DS18B20 может питаться напряжением линии данных (так называемое *питание от паразитного источника* — см. *главу 11*) при отсутствии внешнего источника напряжения.

Каждый датчик типа DS18B20 имеет уникальный 64-битный последовательный код, который позволяет общаться с множеством датчиков DS18B20, установленных на одной шине. Первые 8 битов — код серии (для DS18B20 — `28h`), затем 48 битов уникального номера, и в конце 8 битов CRC-кода. Такой принцип позволяет использовать один микропроцессор, чтобы контролировать множество датчиков DS18B20, распределенных по большому участку.

Характеристики датчика DS18S20:

- ❒ интерфейс 1-Wire;
- ❒ измеряемая температура от –55 до +125 °C;
- ❒ точность 0,5 °C в диапазоне от –10 до +85 °C;
- ❒ температура считывается 9-ю битами данных;
- ❒ время на конвертацию температуры — 750 ms (максимальное).

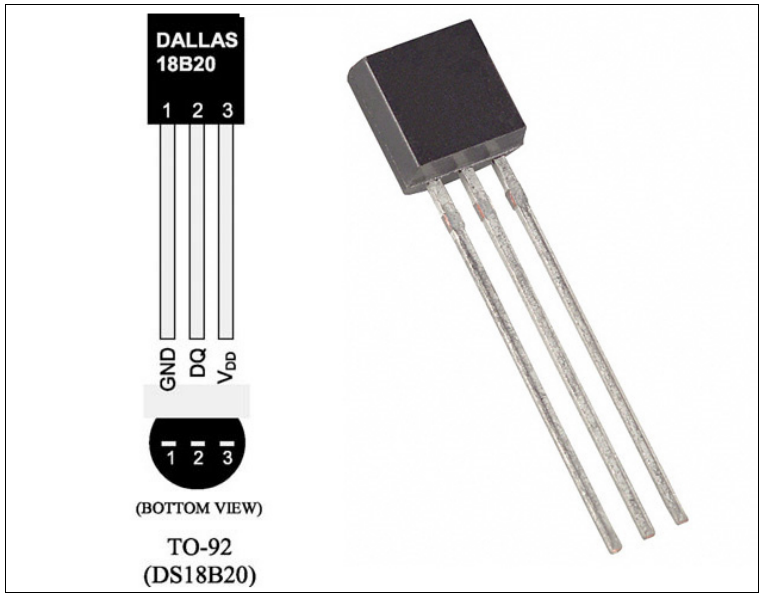


Рис. 12.1. Датчик DS18B20

Данные о температуре хранятся в оперативной памяти датчика (рис. 12.2). Память состоит из оперативной ROM и энергонезависимой EEPROM:

- первые два байта — содержат данные об измеренной температуре;
- третий и четвертый байты хранят верхний (`TH`) и нижний (`TL`) пределы температуры;
- пятый и шестой — не используются;
- седьмой и восьмой — байты-счетчики. Они могут использоваться для более точного измерения температуры;
- девятый байт хранит CRC-код предыдущих восьми.

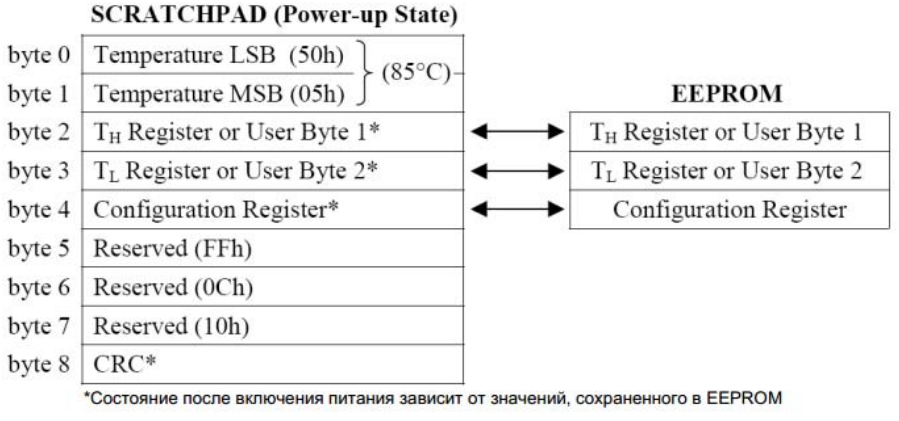


Рис. 12.2. Карта памяти DS18B20

Кроме общих команд для всех типов устройств 1-Wire, представленных в табл. 11.2, датчик может выполнять следующие команды:

- `Alarm Search [ECh]` — операция этой команды идентична операции поиска адресов `[F0h]`, за исключением того, что в данном случае ответят только те датчики, у которых при последнем измерении температура вышла за установленные пределы (выше `TH` или ниже `TL`);
- `Convert T [44h]` — конвертировать температуру. Датчик произведет измерение и запись данных о текущей температуре. Если ведущее устройство будет за этой командой слать тайм-слоты чтения, то пока конвертация не закончена, DS18S20 будет выдавать в линию "0", а после завершения конвертации "1".
- `Write Scratchpad [4Eh]` — запись в память. Эта команда позволяет записать 3 байта в память датчика. Первый байт запишется в `TH`, второй — в `TL`, а третий байт запишется в пятый байт памяти — это байт конфигурации;
- `Read Scratchpad [BEh]` — чтение памяти. Команда позволяет нам считать память датчика. В ответ на эту команду датчик вернет 9 байтов своей памяти, начиная с 0-го байта `TEMPERATURE LSB` и заканчивая восьмым — CRC;
- `Copy Scratchpad [48h]` — копировать память. Датчик скопирует содержимое ОЗУ — `TH` и `TL` в EEPROM.

Последовательность команд для получения данных от датчика о температуре:

1. Произвести `RESET` и поиск устройств на линии 1-Wire.
2. Выдать команду `0x44`, чтобы запустить конвертацию температуры датчиком.
3. Подождать не менее 750 ms.

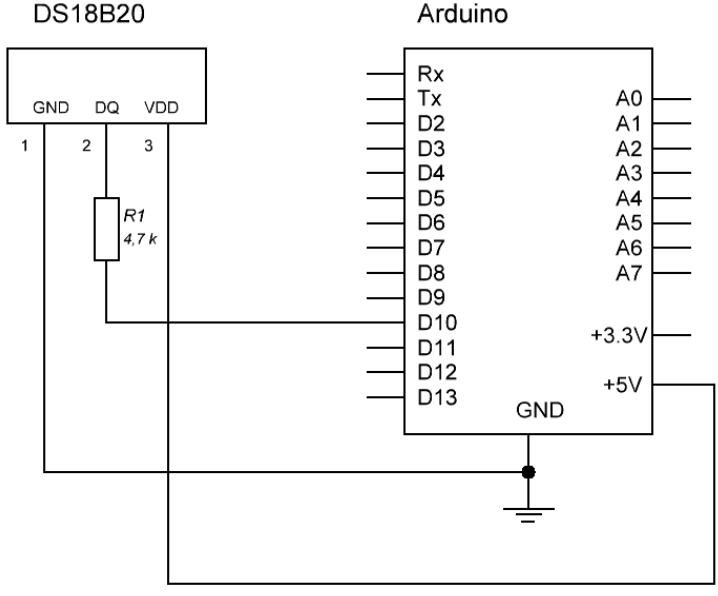


Рис. 12.3. Подключение датчикаDS18B20 к Arduino

4. Выдать команду `0xBE`, чтобы считать ОЗУ датчика (данные о температуре будут в первых двух байтах).

Датчик может запитываться двумя способами: внешним питанием (3 провода) или паразитным (питание от шины, 2 провода). Схема подключения датчика от внешнего питания к Arduino представлена на рис. 12.3.

12.2. Использование библиотеки *OneWire*

Для работы с датчиками с интерфейсом 1-Wire можно использовать библиотеку `OneWire` (см. *главу 11*). Напомню, что библиотека `OneWire` находится в папке libraries/OneWire сопровождающего книгу электронного архива.

Скетч получения данных с датчика температуры DS18B20 и вывода данных в последовательный порт с помощью библиотеки `OneWire` представлен в листинге 12.1.

Листинг 12.1

```
#include <OneWire.h>
OneWire ds(10); // линия 1-Wire будет на pin 10

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

void loop(void)
{
 byte i;
 byte present = 0;
 byte data[12];
 byte addr[8];

 if ( !ds.search(addr)) {
   Serial.print("No more addresses.\n");
   ds.reset_search();
   return;
 }
 if ( OneWire::crc8( addr, 7) != addr[7]) {
   Serial.print("CRC is not valid!\n");
   return;
 }

 if ( addr[0] != 0x28) {
   Serial.print("Device is not a DS18B20 family device.\n");
   return;
 }
```

```
 ds.reset();
 ds.select(addr);
 ds.write(0x44,1);     // запускаем конвертацию

 delay(750);  // ждем 750ms

 present = ds.reset();
 ds.select(addr);
 ds.write(0xBE);     // считываем ОЗУ датчика

 for ( i = 0; i < 9; i++) {     // обрабатываем 9 байт
  data[i] = ds.read();
  Serial.print(data[i], HEX);
  Serial.print(" ");
 }

 // высчитываем температуру :)
 int HighByte, LowByte, TReading, Tc_100;
 LowByte = data[0];
 Serial.print("LB= ");Serial.print(LowByte,HEX);
 HighByte = data[1];
 Serial.print(" HB= ");Serial.print(HighByte,HEX);
 TReading = (HighByte << 8) + LowByte;
 Tc_100 = TReading/2;
 Serial.print(" T = ");Serial.print(Tc_100);
 Serial.println();
}
```

12.3. Библиотека *DallasTemperature*

Специально для температурных датчиков Dallas (DS18S20, DS18B20, DS1822) создана Arduino-библиотека `DallasTemperature`, скачать которую можно со страницы **https://github.com/milesburton/Arduino-Temperature-Control-Library**.

Библиотека `DallasTemperature` находится в папке libraries/DallasTemperature сопровождающего книгу электронного архива.

Скетч получения данных с датчика температуры DS18B20 и вывода данных в последовательный порт с помощью библиотеки `DallasTemperature` представлен в листинге 12.2.

Листинг 12.2

```
#include <DallasTemperature.h>

DallasTemperature tempSensor;

void setup(void)
```

```
{
 Serial.begin(9600);
 tempSensor.begin(10); // Датчик на 10 порт
}

void loop(void)
{
  switch(tempSensor.isValid())
  {
    case 1:
      Serial.println("Invalid CRC");
      tempSensor.reset(); // сбросить девайс
      return;
    case 2:
      Serial.println("Invalid device");
      tempSensor.reset(); // сбросить девайс
      return;
  }
 Serial.print(" T = ");Serial.print(tempSensor.getTemperature()); // отправить
                                                                  температуру
 Serial.println(); // перенос строки
}
```

ГЛАВА 13

Arduino и датчики температуры и влажности DHT

Рассмотрим здесь датчики влажности и температуры DHT (BP11 и DH22), которые будем использовать в проекте домашней метеостанции (см. *главу 14*). Одновременно измерять температуру и относительную влажность выглядит логично, поскольку второе напрямую зависит от первого. Так, повышение температуры батарей центрального отопления приводит к уменьшению относительной влажности воздуха. Насколько важна влажность в помещении? Считается, что оптимальное ее значение около 50 % — именно при такой влажности растения, люди и животные чувствуют себя комфортно.

13.1. Характеристики датчиков DH11, DH22

Датчики DH11 (рис. 13.1) и DH22 (рис. 13.2) не отличаются высоким быстродействием и точностью, однако могут найти свое применение в радиолюбительских проектах из-за своей невысокой стоимости. Датчики DHT состоят из емкостного датчика влажности и термистора. Кроме того, датчик содержит в себе простенький АЦП для преобразования аналоговых значений влажности и температуры.

Датчик имеет 4 вывода стандарта 2,54 мм:

- 1 — VCC (питание 3–5 В);
- 2 — DATA (вывод данных);
- 3 — не используется;
- 4 — GND (земля).

Протокол обмена — однопроводный, по структуре весьма похож на DS18B20, но с важными оговорками:

- DHT не умеет работать в "паразитном" режиме (питание по линии данных);
- каждый DS18B20 имеет персональный идентификатор, что дает возможность подключения нескольких таких датчиков к одному пину Arduino. Однако у DHT такой возможности нет — один датчик будет использовать строго один цифровой пин.

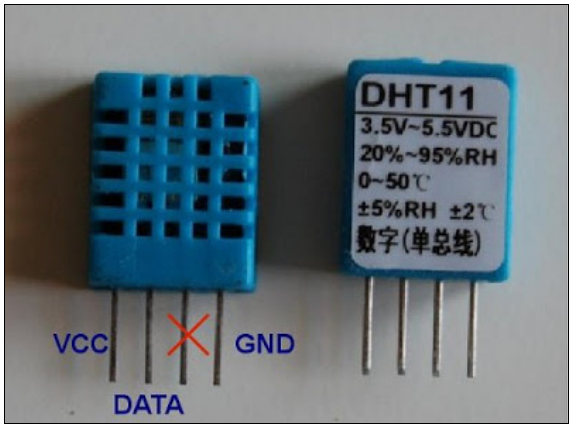


Рис. 13.1. Датчик DH11

Рис. 13.1. Датчик DH22

Общие характеристики датчиков:

- DHT11:
  - очень низкая стоимость;
  - питание и I/O: 3–5 В;
  - определение влажности 20–80 % с точностью 5 %;
  - определение температуры 0–50 °С с точностью 2 %;
  - частота опроса не более 1 Гц (не более одного раза в 1 сек.);
  - размеры 15,5×12×5,5 мм;
- DHT22:
  - низкая стоимость;
  - питание и I/O: 3–5 В;
  - определение влажности 0–100 % с точностью 2–5 %;
  - определение температуры –40…+125 °С с точностью ±0,5 °С;
  - частота опроса не более 0,5 Гц (не более одного раза в 2 сек);
  - размеры 15,1×25×7,7 мм.

Сенсор DHT22 имеет лучшие, чем у DHT11, характеристики, но более высокую стоимость.

13.2. Подключение к Arduino

Рекомендуемая схема подключения к Arduino (рис. 13.3) содержит обязательный для однопроводных линий резистор-подтяжку к VCC и, в качестве опции, рекомендуется конденсатор (фильтр по питанию между VCC и GND).

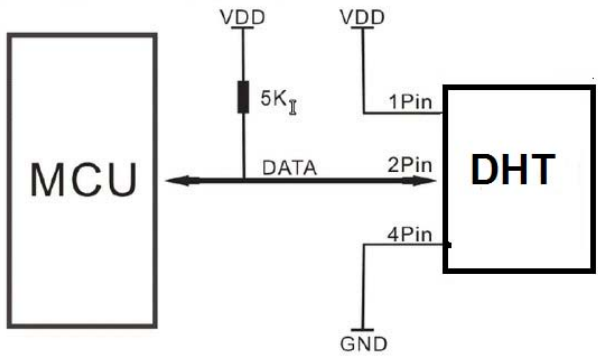


Рис. 13.3. Схема подключения к Arduino

Если к вам в руки попали DHT11 или DH22 на небольшой плате, можно подключать их напрямую к Arduino — резистор и конденсатор там уже и так есть.

13.3. Библиотека *DHT*

Для работы Arduino с датчиками DH11 и DH22 уже есть готовая библиотека — DHT, скачать которую можно со страницы в Интернете **https://github.com/adafruit/DHT-sensor-library**. Эта библиотека находится в папке libraries/DHT сопровождающего книгу электронного архива

Для использования датчика DHT11 необходимо закомментировать строку:

```
//#define DHTTYPE DHT22 // DHT 22 (AM2302)
```

и раскомментировать:

```
#define DHTTYPE DHT11 // DHT 11
```

Скетч получения данных с датчика температуры и влажности DH22 и вывода данных в последовательный порт (рис. 13.4) представлен в листинге 13.1.

Листинг 13.1

```
#include "DHT.h"
#define DHTPIN 2  // пин подключения
//#define DHTTYPE DHT11 // DHT 11
#define DHTTYPE DHT22 // DHT 22 (AM2302)
//#define DHTTYPE DHT21 // DHT 21 (AM2301)

DHT dht(DHTPIN, DHTTYPE);

void setup() {
 Serial.begin(9600);
 Serial.println("DHTxx test!");
 dht.begin();
}
```

```
void loop() {
 // Reading temperature or humidity takes about 250 milliseconds!
 // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
 float h = dht.readHumidity();
 float t = dht.readTemperature();

 // check if returns are valid, if they are NaN (not a number) then something
went wrong!
 if (isnan(t) || isnan(h)) {
  Serial.println("Failed to read from DHT");
 } else {
  Serial.print("Humidity: ");
  Serial.print(h);
  Serial.print(" %\t");
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.println(" *C");
 }
}
```

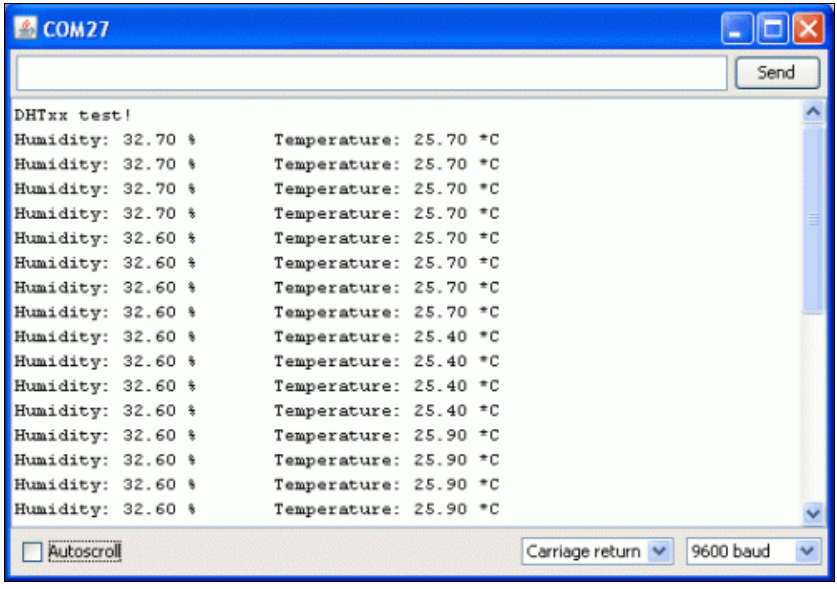


Рис. 13.4. Вывод данных с датчика в монитор последовательного порта

ГЛАВА 14

Сетевой обмен с помощью Arduino

14.1. Устройство Arduino Ethernet shield

Arduino Ethernet shield (рис. 14.1) позволяет подключить плату Arduino к сети. Она основана на ethernet-микросхеме Wiznet W5100. Wiznet W5100 поддерживает стеки TCP и UDP в IP-сети, а также до четырех одновременных подключений к сокетам. Для создания скетчей, которые подключаются к сети при помощи данной платы, используйте библиотеку `Ethernet Library`. Плата Ethernet shield соединяется с платой Arduino при помощи длинных штырьков, проходящих через нее. Это позволяет не изменять расположение выводов и устанавливать другие платы поверх нее (рис. 14.2).

Плата Ethernet shield имеет стандартный разъем RJ-45 со встроенным линейным трансформатором и опцией PoE (Power over Ethernet) для получения питания от обычной витой пары 5 категории. Последние версии платы имеют также разъем для карт типа microSD, которые могут использоваться для хранения файлов и работы

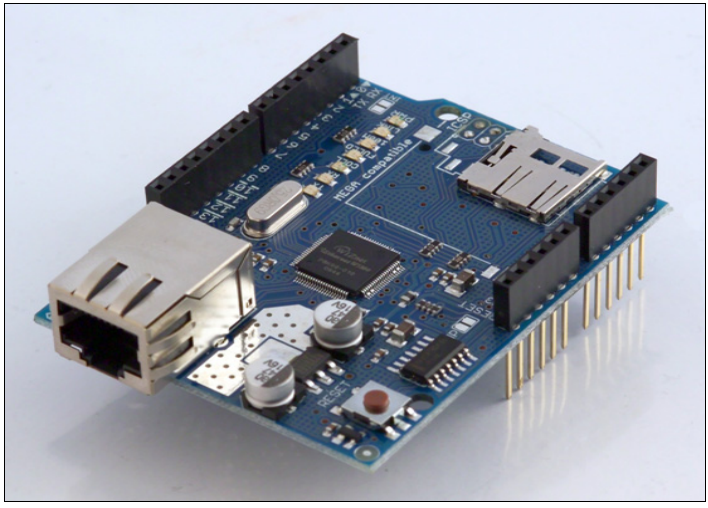


Рис. 14.1. Arduino Ethernet shield

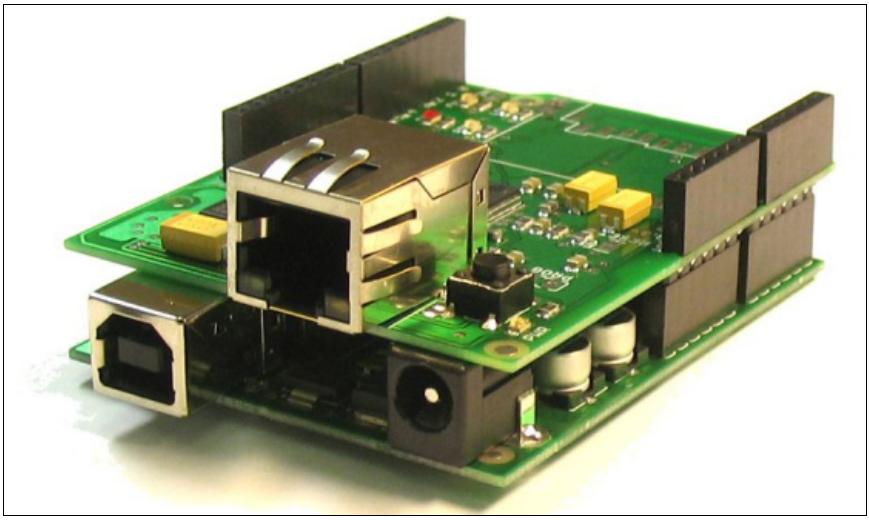


Рис. 14.2. Подключение Ethernet shield к Arduino

с ними по сети. Плата совместима с Arduino Uno и Mega (при использовании библиотеки `Ethernet Library`). Картридер microSD доступен при помощи библиотеки `SD Library`. При применении этой библиотеки вывод 4 используется для сигнала SS (Slave Select).

Последние версии платы также имеют контроллер сброса, который позволяет быть уверенным в правильном перезапуске W5100 при запуске. Предыдущие версии платы были не совместимы с Arduino Mega и требовали ручного сброса после включения. Предыдущая версия платы имела разъем для полноразмерной карты SD, что в настоящее время не поддерживается.

6-контактный разъем для последовательного программирования совместим с кабелями и платами-переходниками FTDI-USB. Он поддерживает автоматический сброс, что позволяет загружать скетчи без нажатия кнопки сброса на плате. При подключении через адаптер FTDI-USB, Arduino и Ethernet shield получают питание от адаптера.

Как уже отмечалось, текущая версия платы поддерживает подключение адаптера Power over Ethernet (PoE) для получения питания от обычной витой пары 5 категории, а также обеспечивает:

- ❒ совместимость с EEE802.3af;
- ❒ низкие пульсации и шум на выходе (100m Vpp);
- ❒ диапазон входного напряжения от 36 до 57 В;
- ❒ защиту от перегрузки и короткого замыкания;
- ❒ выходное напряжение 9 ВУ;
- ❒ высокоэффективный DC/DC-преобразователь: 75 % при нагрузке в 50 %;
- ❒ напряжение пробоя изоляции 1500 В (на входе и выходе).

Arduino осуществляет связь с W5100 и картой SD посредством шины SPI (через разъем ICSP header). Она расположена на выводах 11, 12 и 13 платы Duemilanove и выводах 50, 51 и 52 платы Mega. На обеих платах вывод 10 используется для выбора W5100 и ввод 4 — для карты SD. Эти контакты не могут быть использованы для другого ввода/вывода. На плате Mega аппаратный вывод SS 53 не используется для выбора ни W5100, ни карты SD, но он должен быть сконфигурирован на вывод, иначе интерфейс SPI не будет работать.

Поскольку W5100 и карта SD разделяют шину SPI, одновременно работать они не могут. Если вы используете оба этих периферийных устройства в своей программе, вам следует позаботиться о соответствующих библиотеках. Если вы не используете одно из этих периферийных устройств, вам следует явно отключить его. Чтобы сделать это, сконфигурируйте вывод платы 4 как выход и запишите в него "1". Для W5100 установите на выводе 10 высокий уровень.

Кнопка сброса платы перезапускает и дочернюю плату, и плату Arduino.

Плата имеет несколько индикаторных светодиодов:

- `PWR` — индикация наличия питания платы;
- `LINK` — индикация наличия сетевого линка, мигание при отправке или получении данных;
- `FULLD` — индикация полнодуплексного соединения;
- `100M` — индикация соединения на скорости 100 Мбит/с (в отличие от соединения на 10 Мбайт/с);
- `RX` — мигает при получении платой данных;
- `TX` — мигает при отправке платой данных;
- `COLL` — мигает при сетевой коллизии.

Запаиваемая перемычка "INT" может быть замкнута, что позволит плате Arduino получать уведомления (через прерывания) о событиях от W5100, но в настоящее время это не поддерживается библиотекой `Ethernet library`. Перемычка соединяет вывод INT микросхемы W5100 и цифровой вывод 2 платы Arduino.

14.2. Библиотека *Ethernet library*

Для работы с Ethernet shield используется стандартная Arduino-библиотека `Ethernet library`. Она поставляется в составе дистрибутива Arduino. Рассмотрим ее подробнее.

14.2.1. Класс Ethernet (*Ethernet class*)

Класс Ethernet инициализирует библиотеку и сетевые настройки. Начиная с версии 1.0, библиотека поддерживает DHCP. Использование `Ethernet.begin (mac)` позволяет автоматически получать IP-адрес.

Функции класса Ethernet:

- `begin()`;
- `localIP()`.

Функция *Ethernet.begin()*

Функция `Ethernet.begin()` инициализирует библиотеку и сетевые настройки. Синтаксис функции:

```
Ethernet.begin(mac)
Ethernet.begin(mac,ip)
Ethernet.begin(mac,ip,gateway)
Ethernet.begin(mac, ip, gateway, subnet)
```

Параметры:

- `mac` — mac-адрес (Media Access Control) устройства (массив из 6 байтов). Это аппаратный адрес адаптера вашего устройства. Новые Arduino Ethernet shield включают наклейку с MAC-адресом устройства. Для более старых его нужно найти самостоятельно;
- `ip` — IP-адрес устройства (массив из 4 байтов);
- `gateway` — IP-адрес сетевого шлюза (массив из 4 байтов). Опционально — по умолчанию IP-адрес устройства с последнего октета установлен в 1;
- `subnet` — маска подсети сети (массив из 4 байтов). Опционально — по умолчанию 255.255.255.0.

Возвращаемого значения нет.

Функция *Ethernet.localIP()*

Функция `Ethernet.localIP()` возвращает IP-адрес Ethernet shield.

Синтаксис функции:

```
Ethernet.localIP()
```

Возвращаемое значение: IP-адрес устройства.

14.2.2. Класс IPAddress (*IPAddress class*)

Класс IPAddress работает с локальными и удаленными IP-адресами.

Функция *IPAddress()*

Функция `IPAddress()` определяет IP-адрес. Он может быть использован для объявления локальных и удаленных адресов.

Синтаксис:

```
IPAddress(address)
```

Параметр: `address` — назначаемый IP-адрес.

Возвращаемого значения нет.

14.2.3. Класс Server (*Server class*)

Класс Server создает сервер, который может передавать данные и получать данные от подключенных клиентов.

Функции класса Server:

- `ethernetServer()`;
- `begin()`;
- `available()`;
- `write()`;
- `print()`;
- `println()`.

Функция *ethernetServer()*

Функция `ethernetServer()` создает сервер, который прослушивает входящие соединения на указанный порт.

Синтаксис:

```
ethernetServer (port)
```

Параметр: `port` — назначаемый для прослушивания порт.

Возвращаемого значения нет.

Функция *begin()*

Функция `begin()` запускает прослушивание входящих соединений.

Синтаксис:

```
server.begin()
```

Параметров нет.

Возвращаемого значения нет.

Функция *available()*

Функция `available()` получает ссылку на подключенного к серверу клиента для обмена данными.

Синтаксис:

```
server. available()
```

Параметров нет.

Возвращаемое значение: ссылка на подключенный к серверу клиент (`EthernetClient client = server.available();`).

Пример использования функции `available()` представлен в листинге 14.1.

Листинг 14.1

```
#include <Ethernet.h>
#include <SPI.h>

// mac-адрес Ethernet shield:
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
// IP-адрес, назначаемый Ethernet shield:
byte ip[] = { 192, 168, 0, 177 };
// адрес шлюза:
byte gateway[] = { 192, 168, 0, 1 };
// маска:
byte subnet[] = { 255, 255, 0, 0 };

// порт для сервера
EthernetServer server = EthernetServer(23);

void setup()
{
  // инициализация Ethernet shield
  Ethernet.begin(mac, ip, gateway, subnet);
  // запуск сервера
  server.begin();
}

void loop()
{
  // получение ссылки на подключенного клиента
  EthernetClient client = server.available();
  // пока подключение активно
  if (client == true) {
    // читаем данные, отправляемые клиентом,
    // и отправляем их обратно клиенту:
    server.write(client.read());
  }
}
```

Функция *write()*

Функция `write()` отправляет данные всем клиентам, подключенным к серверу.

Синтаксис:

```
server.write(data)
```

Параметр: `data` — отправляемые данные (байт или символ).

Возвращаемого значения нет.

Функция *print()*

Функция `print()` отправляет данные всем клиентам, подключенным к серверу. При этом число отправляется как последовательность цифр данного числа.

Синтаксис:

```
server.print(data)
server.print(data,BASE)
```

Параметры:

- `data` — отправляемые данные (байт, символ, число, строка).
- `BASE` — вид передаваемых символов:
  - `BIN` — двоичный;
  - `DEC` — десятичный;
  - `OCT` — восьмеричный;
  - `HEX` — шестнадцатеричный.

Возвращаемое значение: количество переданных байтов.

Функция *println()*

Функция `println()` отправляет данные всем клиентам, подключенным к серверу. Аналогична функции `print()` с добавлением в отправку символа перехода на новую строку.

Синтаксис:

```
server.println()
server.println(data)
server.println(data,BASE)
```

Параметры:

- `data` — отправляемые данные (байт, символ, число, строка).
- `BASE` — вид передаваемых символов:
  - `BIN` — двоичный;
  - `DEC` — десятичный;
  - `OCT` — восьмеричный;
  - `HEX` — шестнадцатеричный.

Возвращаемое значение: количество переданных байтов.

14.2.4. Класс Client (*Client class*)

Класс Client создает клиентов, которые могут подключаться к серверам и обмениваться данными.

Функции класса Client:

- `client`;
- `EthernetClient()`;
- `connected()`;
- `connect()`;
- `write()`;
- `print()`;
- `println()`;
- `available()`;
- `read()`;
- `flush()`;
- `stop()`.

Функция *client()*

Функция `client()` создает клиента, который подключается к серверу по указанному в параметрах IP-адресу и порту.

Синтаксис:

```
client(ip, port)
```

Параметры:

- `ip` — IP-адрес подключения клиента (массив из 4 байтов);
- `port` — порт подключения клиента.

Возвращаемого значения нет.

Функция *EthernetClient()*

Функция `EthernetClient()` создает клиента, который подключается к серверу. Параметры подключения (IP-адрес и порт) определяются в функции `client.connect()`).

Синтаксис:

```
EthernetClient()
```

Параметров нет.

Возвращаемого значения нет.

Функция *connected()*

Функция `connected()` определяет, подключен клиент или нет. Клиент считается подключенным, если соединение закрыто, но есть непрочитанные данные.

Синтаксис:

```
client.connected()
```

Параметров нет.

Возвращаемые значения: `true` — если клиент подключен, и `false` — если нет.

Функция *connect()*

Функция `connect()` подключается к серверу по указанному IP-адресу и порту.

Синтаксис:

```
client.connect()
```

Параметры:

- `ip` — IP-адрес подключения клиента (массив из 4 байтов);
- `port` — порт подключения клиента.

Возвращаемые значения: `true` — если клиент подключен, и `false` — если нет.

Функция *write()*

Функция `write()` отправляет на сервер данные с подключенного клиента.

Синтаксис:

```
client.write(data)
```

Параметр: `data` — отправляемые данные (байт или символ).

Возвращаемого значения нет.

Функция *print()*

Функция `print()` отправляет данные на сервер с подключенного клиента. При этом число отправляется как последовательность цифр данного числа.

Синтаксис:

```
client.print(data)
client.print(data,BASE)
```

Параметры:

- `data` — отправляемые данные (байт, символ, число, строка).
- `BASE` — вид передаваемых символов:
  - `BIN` — двоичный;
  - `DEC` — десятичный;
  - `OCT` — восьмеричный;
  - `HEX` — шестнадцатеричный.

Возвращаемое значение: количество переданных байтов.

Функция *println()*

Функция `println()` отправляет данные на сервер с подключенного клиента. Аналогична функции `print()` с добавлением в отправку символа перехода на новую строку.

Синтаксис:

```
client.println()
client.println(data)
client.println(data,BASE)
```

Параметры:

- `data` — отправляемые данные (байт, символ, число, строка).
- `BASE` — вид передаваемых символов:
  - `BIN` — двоичный;
  - `DEC` — десятичный;
  - `OCT` — восьмеричный;
  - `HEX` — шестнадцатеричный.

Возвращаемое значение: количество переданных байтов.

Функция *available()*

Функция `available()` возвращает число байтов, доступных для чтения (т. е. объем данных, который был отправлен сервером для клиента).

Синтаксис:

```
client.available()
```

Параметров нет.

Возвращаемое значение: количество доступных для чтения байтов.

Функция *read()*

Функция `read()` получает следующий (после последнего, полученного командой `read()`) байт от сервера или –1, если данных больше нет.

Синтаксис:

```
client.read()
```

Параметров нет.

Возвращаемое значение: байт или –1.

Функция *flush()*

Функция `flush()` сбрасывает полученные от сервера, но еще не прочитанные данные.

Синтаксис:

```
client.flush()
```

Параметров нет.

Возвращаемого значения нет.

Функция *stop()*

Функция `stop()` отключает клиента от сервера.

Синтаксис:

```
client.stop()
```

Параметров нет.

Возвращаемого значения нет.

14.2.5. Класс EthernetUDP (*EthernetUDP class*)

Класс EthernetUDP позволяет отправлять и получать UDP-сообщения.

Функции класса EthernetUDP:

- `begin()`;
- `read()`;
- `write()`;
- `beginPacket()`;
- `endPacket()`;
- `parsePacket()`;
- `available()`;
- `remoteIP()`;
- `remotePort()`.

Функция *begin()*

Функция `begin()` инициализирует библиотеки Ethernet UDP и сетевые параметры.

Синтаксис:

```
client.begin(port)
```

Параметр: `port` — локальный порт для прослушивания.

Возвращаемого значения нет.

Функция *read()*

Функция `read()` читает UDP-данные из указанного буфера. Если аргументы не заданы, то она вернет следующий символ в буфере.

Синтаксис:

```
UDP.read()
UDP.read(packetBuffer,maxSize)
```

Параметры:

- `packetBuffer` — буфер для хранения входящих пакетов;
- `maxSize` — максимальный размер буфера.

Возвращаемое значение: символы из буфера.

Функция *write()*

Функция `write()` записывает UDP-данные в удаленное соединение. Данные должны быть заключены между `beginPacket()` и `endPacket()`. `beginPacket()` инициализирует пакет данных, данные не отправляются до вызова `endPacket()`.

Синтаксис:

```
UDP.write(message)
```

Параметр: `message` — отправляемое сообщение.

Возвращаемое значение: количество возвращаемых символов.

Пример использования функции `write()` представлен в листинге 14.2.

Листинг 14.2

```
// mac-адрес Ethernet shield:
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
// IP-адрес в локальной сети:
IPAddress ip(192, 168, 1, 177);
// порт
unsigned int localPort = 8888;
EthernetUDP Udp;
void setup() {
  // запустить UDP-соединение
  Ethernet.begin(mac,ip);
  Udp.begin(localPort);
}
void loop() {
  Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
    Udp.write("hello");
    Udp.endPacket();
}
```

Функция *beginPacket()*

Функция `beginPacket()` инициирует UDP-подключение на отправку данных в удаленное соединение.

Синтаксис:

```
UDP. beginPacket (remoteIP, port)
```

Параметры:

- `remoteIP` — IP-адрес удаленного соединения;
- `port` — порт удаленного соединения.

Возвращаемого значения нет.

Функция *endPacket()*

Функция `endPacket()` вызывается после записи UDP-данных удаленного соединения.

Синтаксис:

```
UDP. endPacket()
```

Параметров нет.

Возвращаемого значения нет.

Функция *parsePacket()*

Функция `parsePacket()` проверяет наличие пакета UDP и его размер. Функция `parsePacket()` должна вызываться перед чтением буфера с `UDP.read()`.

Синтаксис:

```
UDP. parsePacket()
```

Параметров нет.

Возвращаемое значение: размер полученных данных.

Пример использования функции `parsePacket()` представлен в листинге 14.3.

Листинг 14.3

```
// mac-адрес Ethernet shield
byte mac[] = {  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
// ip-адрес
IPAddress ip(192, 168, 1, 177);
// порт
unsigned int localPort = 8888;
EthernetUDP Udp;

void setup() {
  // запустить UDP-соединение
  Ethernet.begin(mac,ip);
  Udp.begin(localPort);
  Serial.begin(9600);
}

void loop() {
  // получить размер данных
  int packetSize = Udp.parsePacket();
  if(packetSize)
  {
    Serial.print("size=");
    Serial.println(packetSize);
  }
  delay(10);
}
```

Функция *available()*

Функция `available()` получает количество данных, доступных для чтения из буфера.

Синтаксис:

```
UDP. available()
```

Параметров нет.

Возвращаемое значение: количество байтов, доступных для чтения.

Функция *remoteIP()*

Функция `remoteIP()` получает IP-адрес удаленного соединения. Эта функция должна быть вызвана после `UDP.parsePacket()`.

Синтаксис:

```
UDP. remoteIP()
```

Параметров нет.

Возвращаемое значение: IP-адрес удаленного соединения (4 байта).

Функция *remotePort()*

Функция `remotePort()` получает IP-адрес удаленного соединения UDP.

Синтаксис:

```
UDP. remotePort()
```

Параметров нет.

Возвращаемое значение: порт соединения с удаленным хостом.

14.3. Домашняя метеостанция с доступом через Интернет

14.3.1. Устройство, настройка и отладка метеостанции

Рассмотрим здесь, как можно использовать Ethernet shield в связке с Arduino в качестве веб-сервера, при обращении к которому будет выдаваться страница с данными датчика температуры DS18B20, датчика влажности DH11 и цифрового датчика атмосферного давления BMP085 (рис. 14.3), подключенных к Arduino. Электрическая схема устройства представлена на рис. 14.4.

Наш веб-сервер подключен к сети Интернет через маршрутизатор TL-WR743ND и ADSL-модем D-Link 2640NRU. Присвоим ему статический IP-адрес — 192.168.1.111. Кроме этого нам необходим доступ к нему из внешней сети Интернет. Поскольку порт 80 у меня уже занят для доступа из внешней сети к веб-серверу, на котором находятся мои сайты, откроем доступ к нашему веб-серверу Arduino по порту

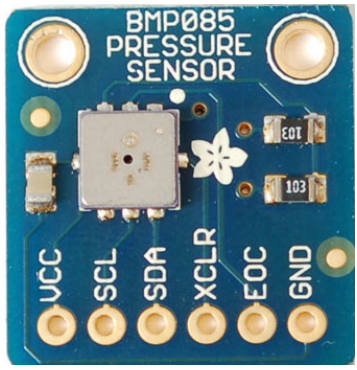


Рис. 14.3. Блок датчика BMP085

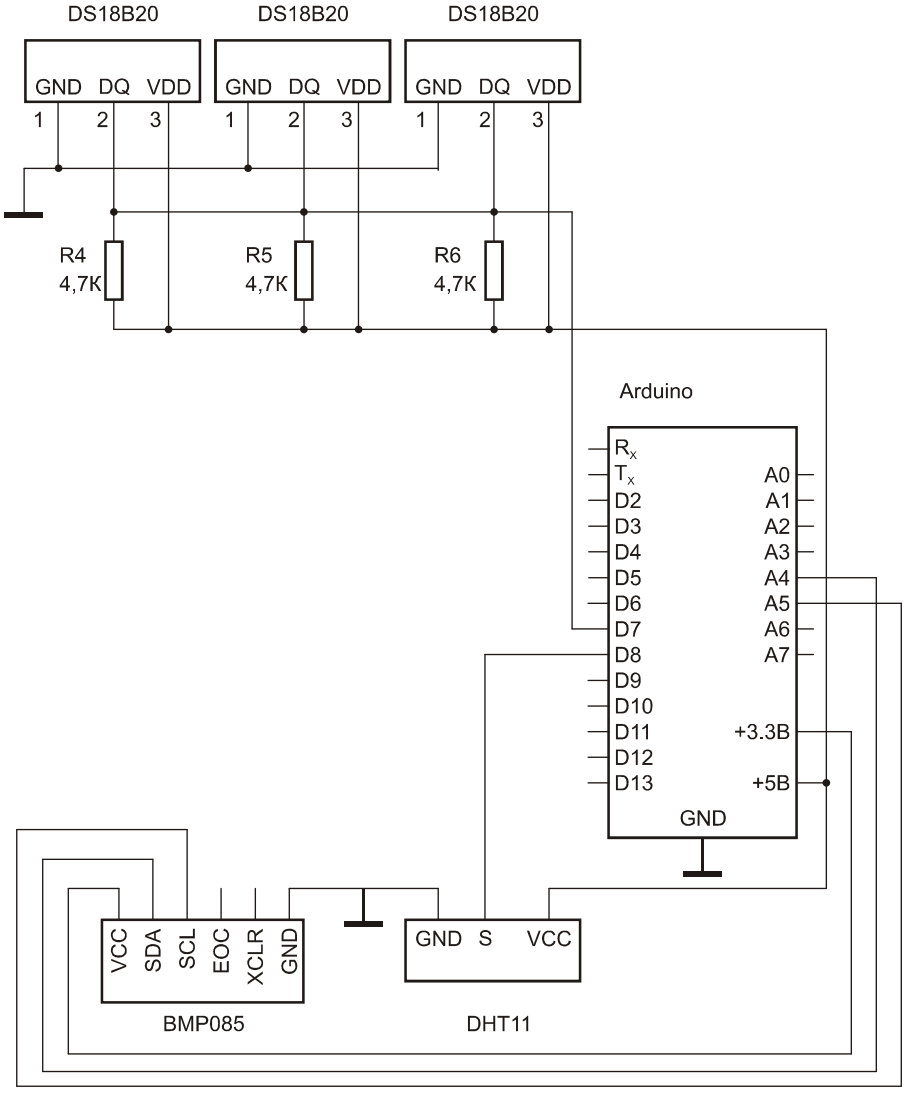


Рис. 14.4. Электрическая схема устройства

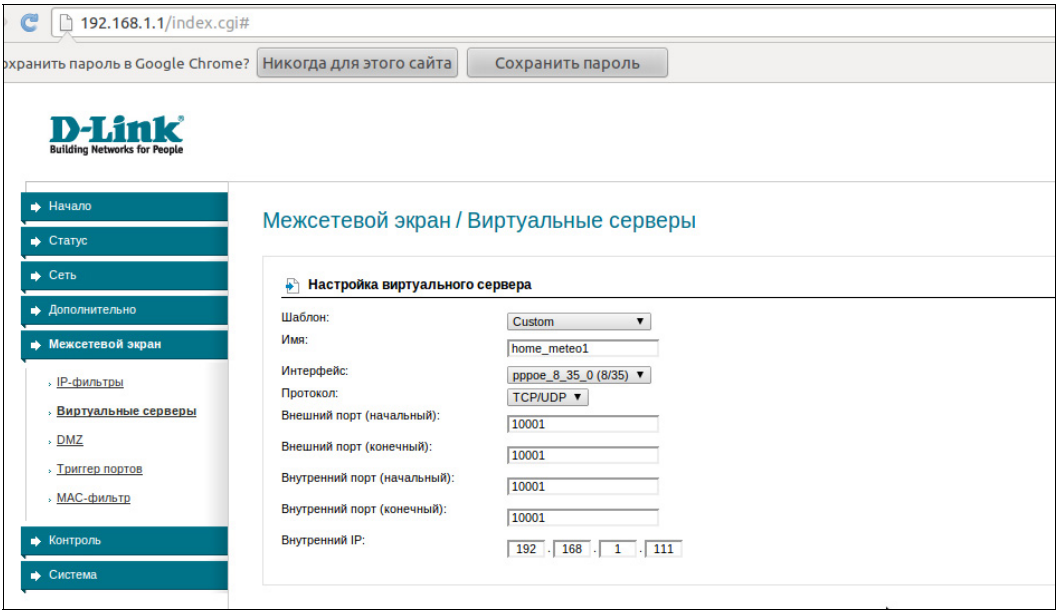


Рис. 14.5. Настройка виртуального сервера

Рис. 14.6. Виртуальный сервер создан

10001. Настройка доступа для ADSL-модема Dlink 2640NRU представлена на рис. 14.5 и 14.6. Не забываем для сохранения настроек перегружать модем.

Приступим к созданию скетча. Сначала запускаем веб-сервер Arduino с IP = = 192.168.1.111 на порту 10001. Содержимое данного фрагмента скетча представлено в листинге 14.4.

Листинг 14.4

```
// mac для новых плат указан на бирке
// для старых пишем произвольный
byte mac[] = { 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF };
```

```
// IP-адрес, назначаемый Ethernet shield:
byte ip[] = { 192, 168, 1, 111 };
// Инициализация библиотеки Ethernet server library
EthernetServer server(10001);
void setup()
{
  ... ... ...
  // запуск сервера
  server.begin();
}
```

Ожидание обращения к серверу происходит в процедуре `loop()`. При обращении к веб-серверу Arduino необходимо получить показания с трех датчиков: температуры DS18B20, температуры и влажности с датчика DH11, температуры и давления с датчика BM085 и выдать ответ в формате JSON. Код этого фрагмента представлен в листинге 14.5. HTML-страница, выдаваемая клиенту, представлена на рис. 14.7.

Листинг 14.5

```
void loop()
{
  // ожидание запроса
  EthernetClient client = server.available();
  if (client) {
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        // конец строки запроса
        if (c == '\n' && currentLineIsBlank) {
          // отправка стандартного заголовка
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println();
          // формирование ответа JSON
          client.print('{');client.print('"');client.print("meteo");
          client.print('"');client.println(":");client.print('{');
          // 3 датчика температуры
          for(int j=1;j<4;j++)
            {
            int Temp=get_temp(j);
            client.print('"');client.print("temp");client.print(j);
            client.print('"');client.print(":");
            client.print('"');client.print(Temp/16);client.print(".");
            client.print(((Temp%16)*100)/16);client.print('"');client.print(',');
           }
```

```
            // датчик DH11
            float h = dht.readHumidity();
            float t = dht.readTemperature();
            client.print('"');client.print("temp4");client.print('"');
            client.print(":");

client.print('"');client.print(t);client.print('"');client.print(',');
            client.print('"');client.print("humidity4");client.print('"');

client.print(":");client.print('"');client.print(h);client.print('"');
            client.print(',');
            // датчик bmp085
            dps.getTemperature(&Temperature085);
            dps.getPressure(&Pressure085);
            dps.getAltitude(&Altitude085);
            client.print('"');client.print("temp5");client.print('"');
            client.print(":");
            client.print('"');client.print(Temperature085*0.1);
            client.print('"');client.print(',');
            client.print('"');client.print("pressure5");client.print('"');
            client.print(":");
            client.print('"');client.print(Pressure085/133.3);client.print('"');
            client.println("}");client.print("}");
          break;
          }
          if (c == '\n') {
            currentLineIsBlank = true;
          }
          else if (c != '\r') {
            currentLineIsBlank = false;
          }
        }
      }
      delay(1);
      // закрыть соединение
      client.stop();
    }
}
```

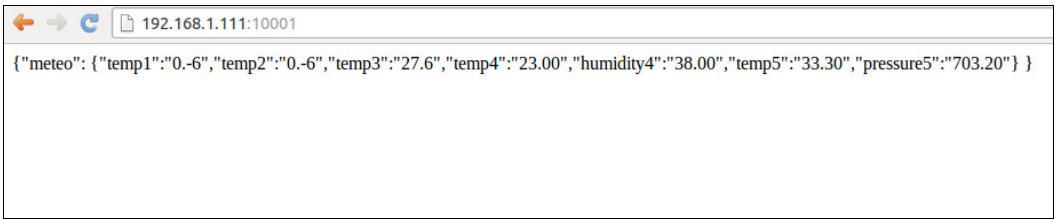


Рис. 14.7. Страница в браузере при обращении к серверу из сети

Полностью этот скетч находится в папке examples/_14_1 сопровождающего книгу электронного архива.

14.3.2. Создание виджета для планшетов с ОС Android

Теперь напишем виджет для Android, чтобы при наличии соединения с Интернетом видеть показания датчиков на телефоне, где бы мы ни находились.

> ***Пояснение***
>
> Виджеты — это маленькие приложения, которые могут быть размещены на рабочем столе вашего Android-устройства. Виджет периодически получает новые данные и обновляет свой вид.

Для создания виджета необходимо:

1. Создать файл XML-layout со слоем, в котором описывается внешний вид виджета.
2. Создать XML-файл метаданных, в котором задаются различные характеристики виджета:
   - layout-файл (из п. 1) — чтобы виджет знал, как он будет выглядеть;
   - размер виджета — чтобы он знал, сколько места должен занять на экране;
   - интервал обновления — чтобы система знала, как часто ей надо будет обновлять виджет.
3. Создать приемник `BroadcastReceiver`, который будет использован для обновления виджетов. Этот приемник расширяет `AppWidgetProvider`, который обеспечивает жизненный цикл виджета.
4. Внести изменения в файл AndroidManifest.xml.

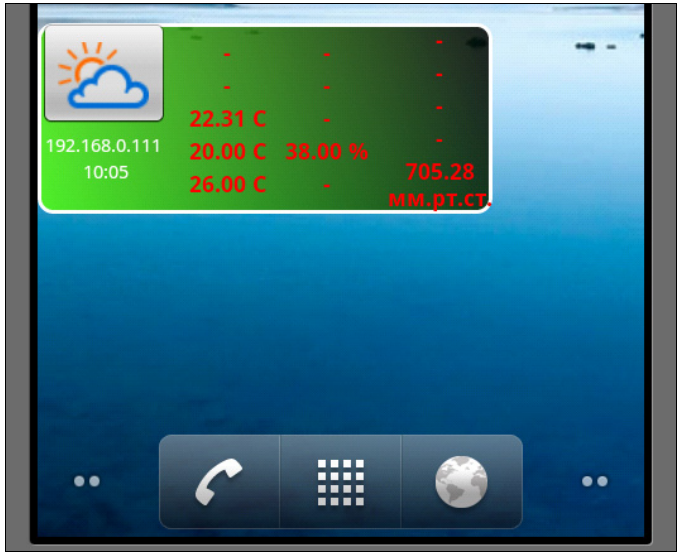


Рис. 14.8. Внешний вид виджета на экране Android-устройства

При запуске виджета он пытается соединиться с внутренней сетью 192.168.1.111:10001 (на тот случай, если я нахожусь дома, или при неудаче с адресом в сети Интернет XX.XX.XX.XX:10001). Если соединение удачно — парсим JSON-ответ, выводим данные виджета и обновляем виджет. Каждые 2 минуты виджет обращается к серверу домашней метеостанции и обновляет показания. Внешний вид виджета представлен на рис. 14.8.

Программирование на Android здесь рассматривать не будем, это тема для другой книги. Архив с файлами проекта для среды Eclipse находится в файле examples/Android/_14_1.rar сопровождающего книгу электронного архива.

14.3.3. Размещение данных метеостанции на сайте

Разместим данные с нашей домашней метеостанции на сайте. Пример размещения данных на сайте **http://www.lermontov-kmv.ru** показан на рис. 14.9.

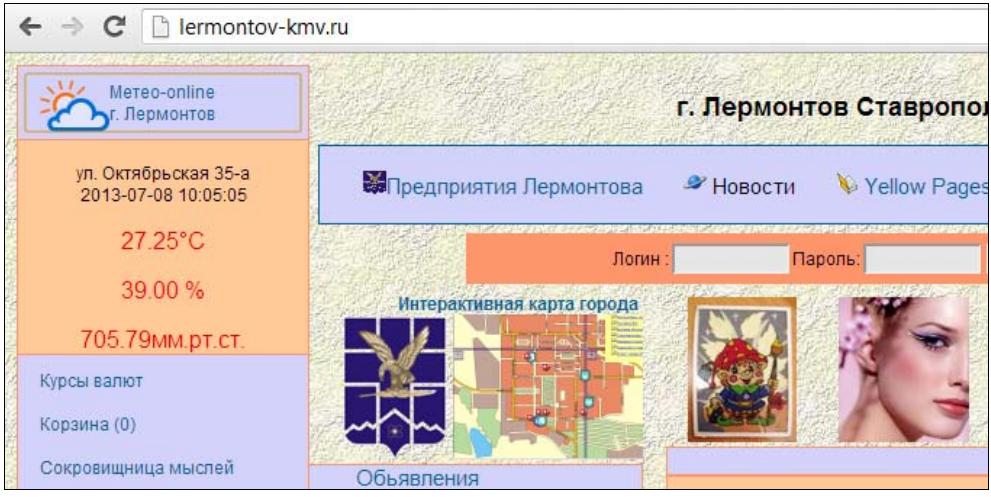


Рис. 14.9. Вывод данных датчиков домашней метеостанции на сайте

Для получения данных с сервера домашней метеостанции на сайте по `cron` каждые 5 минут запускаем скрипт getmeteo1.php, который обращается по адресу 77.39.66.172:10001, получает данные в формате JSON, обрабатывает и заносит часть данных в базу данных MySQL. Содержимое скрипта getmeteo1.php представлено в листинге 14.6.

Листинг 14.6

```
#!/usr/bin/php
<?php
require_once("../mybaza.php");
  // получение страницы
  $contentall=file_get_contents("http://77.39.66.172:10001");
```

```
  // файл
  $data = json_decode($contentall);
  $text="";
  $temp=$data->meteo->temp3;
  $humidity=$data->meteo->humidity4;
  $pressure=$data->meteo->pressure5;

  $data=getDate();
  $hours=$data[hours];
  $mday=$data[mday];
  $mon=$data[mon];
  $year=$data[year];
  $query1="INSERT INTO temperatura SET
         temp='".$temp."',
         humidity='".$humidity."',pressure='".$pressure."',
         year='".$year."',mon='".$mon."',
         mday='".$mday."',hours='".$hours."',
         data='".date("Y-m-d H:i:s")."'  ";
?>
```

Далее на странице сайта **www.lermontov-kmv.ru** при выборе пункта меню вызывается AJAX-функция, которая получает из базы MySQL последние показания датчиков и выводит их на страницу.

Данные из базы можно использовать для построения на сайте графиков изменения в течение дня показаний датчиков температуры, влажности, давления.

ГЛАВА 15

Arduino и карта памяти SD

Как упоминалось в *главе 14*, последние версии платы Ethernet shield имеют разъем для карт типа microSD, который может использоваться для хранения файлов и работы с ними по сети. Картридер microSD доступен при помощи библиотеки `SD Library`.

Библиотека `SD` предназначена для чтения и записи информации карт памяти SD. Библиотека поддерживает файловые системы FAT16 и FAT32 на стандартных SD-картах и картах памяти SDHC. При этом необходимо помнить, что корректно библиотека поддерживает работу только с короткими именами файлов по схеме 8.3. Начиная с версии 1.0, библиотека поддерживает открытие нескольких файлов.

Библиотека использует выводы 11, 12 и 13 (на большинстве плат Arduino) или 50, 51 и 52 (Arduino Mega). При использовании этой библиотеки вывод 4 задействуется для сигнала SS (Slave Select). Рассмотрим подробнее библиотеку `SD library` (она также находится в папке libraries сопровождающего книгу электронного архива).

15.1. Arduino-библиотека *SD*

15.1.1. Класс *SD*

Класс `SD` предоставляет функции для доступа к карте памяти SD и манипулирования файлами и каталогами.

Функции класса `SD`:

- `begin();`
- `exists();`
- `mkdir();`
- `open();`
- `remove();`
- `rmdir()`.

Функция *begin()*

Функция `begin()` инициализирует библиотеку `SD`, назначая вывод для сигнала SS (Slave Select).

Синтаксис

```
SD.begin();
SD.begin(pin)
```

Параметр: `pin` — вывод для сигнала SS.

Возвращаемые значения: `true` — в случае удачного соединения, `false` — в случае неудачи.

Функция *exists()*

Функция `exists()` проверяет, существует на карте SD файл или каталог.

Синтаксис:

```
SD.exists(path)
```

Параметр: `path` — имя файла или каталога.

Возвращаемые значения: `true` — в случае удачного соединения, `false` — в случае неудачи.

Функция *mkdir()*

Функция `mkdir()` создает папку на карте памяти SD.

Синтаксис:

```
SD.exists(dirpath)
```

Параметр: `dirpath` — имя создаваемого каталога (папки). Если параметр задается в виде пути с несуществующими промежуточными каталогами, то создаются и все промежуточные каталоги.

Возвращаемые значения: `true` — если папка создана, `false` — в случае неудачи.

Функция *rmdir()*

Функция `rmdir()` удаляет папку с карты памяти SD, папка должна быть пустой.

Синтаксис:

```
SD.exists(dirpath)
```

Параметр: `dirpath` — имя удаляемой папки.

Возвращаемые значения: `true` — если папка удалена, `false` — в случае неудачи.

Функция *open()*

Функция `open()` открывает файл на карте памяти SD. Если файл не существует и открывается для записи, то он будет создан.

Синтаксис:

```
SD.open(filepath)
SD.open(filepath, mode)
```

Параметры:

- `filepath` — имя открываемого файла;
- `mode` — режим открытия файла:
  - `FILE_READ` — открыть файл для чтения, начиная с начала файла;
  - `FILE_WRITE` — открыть файл для чтения и записи, начиная с конца файла.

Возвращаемые значения: `true` — в случае удачи, `false` — в случае неудачи.

Функция *remove()*

Функция `remove()` удаляет файл с карты памяти SD.

Синтаксис:

```
SD.remove(filepath)
```

Параметр: `filepath` — имя удаляемого файла.

Возвращаемые значения: `true` — в случае удачи, `false` — в случае неудачи.

15.1.2. Класс *File*

Класс `File` предоставляет функции для чтения и записи отдельных файлов на карте памяти SD.

Функции класса `File`:

- `available()`;
- `close()`;
- `flush()`;
- `peek()`;
- `position()`;
- `print()`;
- `println()`;
- `seek()`;
- `size()`;
- `read()`;
- `write()`;
- `isDirectory()`;
- `openNextFile()`;
- `rewindDirectory()`.

Функция *available()*

Функция `available()` проверяет, есть ли доступный для чтения байт в открытом файле.

Синтаксис:

```
myFile.available()
```

Параметр: `File` — экземпляр файла `File`, возвращаемый `SD.open()`.

Возвращаемое значение: число доступных для чтения байтов.

Пример использования функции `available()` представлен в листинге 15.1.

Листинг 15.1

```
// открыть файл для чтения
myFile = SD.open("test.txt",);
if (myFile)
 {
 // чтение из файла и отправка
 // в последовательный порт
 while (myFile.available())
  {
      Serial.write(myFile.read());
  }
 }
 // закрыть файл
 myFile.close();
```

Функция *close()*

Функция `close()` закрывает файл и сохраняет записанные на него данные на карту SD.

Синтаксис:

```
myFile.close()
```

Параметр: `File` — экземпляр файла `File`, возвращаемый `SD.open()`.

Возвращаемого значения нет.

Функция *flush()*

Функция `flush()` гарантирует, что любые байты, записанные в файл, физически сохранятся на SD-карте (при закрытии файла это делается автоматически).

Синтаксис:

```
myFile.flush()
```

Параметр: `File` — экземпляр файла `File`, возвращаемый `SD.open()`.

Возвращаемого значения нет.

Функция *peek()*

Функция `peek()` считывает байт из файла из текущей позиции, не продвигаясь к следующей позиции.

Синтаксис:

```
myFile.peek()
```

Параметр: `File` — экземпляр файла `File`, возвращаемый `SD.open()`.

Возвращаемые значения: байт из файла или –1, если байт не считан.

Функция *position()*

Функция `position()` возвращает текущую позицию в файле, куда следующий байт будет записываться или откуда считываться.

Синтаксис:

```
myFile.position()
```

Параметр: `File` — экземпляр файла `File`, возвращаемый `SD.open()`.

Возвращаемое значение: текущая позиция в файле.

Функция *print()*

Функция `print()` записывает данные в файл, который был открыт для записи.

Синтаксис:

```
myFile.print(data)
myFile.print(data, BASE)
```

Параметры:

- `File` — экземпляр файла `File`, возвращаемый `SD.open()`;
- `data` — данные для записи в файл;
- `BASE` — вид передаваемых символов:
  - `BIN` — двоичный;
  - `DEC` — десятичный;
  - `OCT` — восьмеричный;
  - `HEX` — шестнадцатеричный.

Возвращаемого значения нет.

Функция *println()*

Функция `println()` записывает данные в файл, который был открыт для записи, данные дополняются символами перевода строки.

Синтаксис:

```
myFile.println()
myFile.println(data)
myFile.println(data, BASE)
```

Параметры:

- `File` — экземпляр файла `File`, возвращаемый `SD.open()`;
- `data` — данные для записи в файл;
- `BASE` — вид передаваемых символов:
  - `BIN` — двоичный;
  - `DEC` — десятичный;

- `OCT` — восьмеричный;
- `HEX` — шестнадцатеричный.

Возвращаемого значения нет.

Функция *seek()*

Функция `seek()` устанавливает новую позицию в файле для чтения или записи байта.

Синтаксис:

```
myFile.seek(pos)
```

Параметры:

- `File` — экземпляр файла `File`, возвращаемый `SD.open()`;
- `pos` — новая устанавливаемая позиция в файле для чтения или записи в файле.

Возвращаемые значения: `true` — в случае удачи, `false` — в случае неудачи.

Функция *size()*

Функция `size()` возвращает размер файла в байтах.

Синтаксис:

```
myFile.size()
```

Параметр: `File` — экземпляр файла `File`, возвращаемый `SD.open()`.

Возвращаемое значение: размер файла в байтах.

Функция *read()*

Функция `read()` возвращает байт из открытого файла.

Синтаксис:

```
myFile.read()
```

Параметр: `File` — экземпляр файла `File`, возвращаемый `SD.open()`.

Возвращаемое значение: полученный из файла байт.

Функция *write()*

Функция `write()` записывает данные в открытый для записи файл.

Синтаксис:

```
myFile.write(data)
myFile.write(buf, len)
```

Параметры:

- `File` — экземпляр файла `File`, возвращаемый `SD.open()`;
- `data` — передаваемые данные;

- `buf` — массив символов или байтов;
- `len` — размер символов в буфере для записи.

Возвращаемое значение: количество записанных байтов.

Функция *isDirectory()*

Функция `isDirectory()` проверяет, является ли текущий файл каталогом или нет.

Синтаксис:

```
myFile.isDirectory()
```

Параметр: `File` — экземпляр файла `File`, возвращаемый `SD.open()`.

Возвращаемое значение: `true` — если да, `false` — если нет.

Функция *openNextFile()*

Функция `openNextFile()` возвращает имя следующего по позиции файла из каталога.

Синтаксис:

```
myFile.openNextFile()
```

Параметр: `File` — экземпляр файла `File`, возвращаемый `SD.open()`.

Возвращаемое значение: имя следующего файла из каталога.

Функция *rewindDirectory()*

Функция `rewindDirectory()` приведет вас обратно в первый файл в каталоге. Функция используется в сочетании с `openNextFile()`.

Синтаксис:

```
myFile. rewindDirectory()
```

Параметр: `File` — экземпляр файла `File`, возвращаемый `SD.open()`.

Возвращаемое значение: имя первого файла каталога.

15.2. Запись показаний датчиков на SD-карту

Создадим скетч на Arduino, позволяющий каждую секунду записывать показания с нескольких датчиков на SD-карту. Схема подключения представлена на рис. 15.1. Воспользуемся платой Ethernet shield, имеющей разъем для карт типа microSD. Для сигнала SS используется вывод 4.

Показания с датчиков температуры будем снимать 1 раз в 60 сек. Для получения времени применим библиотеку `RTClib`, которую можно скачать по ссылке **https://github.com/adafruit/RTClib**. Библиотека эта также находится в папке libraries сопровождающего книгу электронного архива. Класс `RTC_Millis` использует Arduino-функцию `millis()`, добавляющую смещение к первоначально установленному времени — оно устанавливается в момент компиляции скетча из системного времени компьютера.

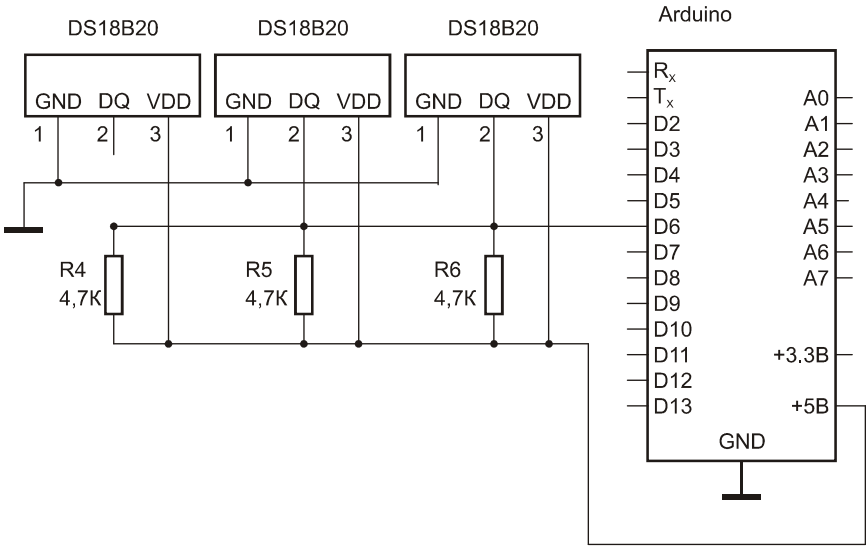


Рис. 15.1. Схема подключения

Сначала необходимо подключить необходимые библиотеки, инициализировать их и установить начальное время (листинг 15.2).

Листинг 15.2

```
#include <Wire.h>
#include "RTClib.h"
#include <SD.h>
#include <SPI.h>
#include <OneWire.h>
// вход 1wire датчиков
OneWire ds(7);
// 1wire коды датчиков
byte my_addr[3][8]={{0x28,0x81,0xC4,0xBA,2,0,0,0x3B},
          {0x28,0x67,0xE5,0xC7,2,0,0,0xA0},
          {0x28,0xF6,0x98,0xBA,2,0,0,0x92}};
// текущий датчик
int tek_sensor=0;
File myFile;
RTC_Millis RTC;
DateTime dt;
unsigned int gettemp=0;
// имя файла текущего дня ymd
String sfilename;
char filename[20];
// температура, полученная с датчика
String sTemp;
```

```
// строка для записи в файл
String record;

void setup () {
  Serial.begin(9600);
  Wire.begin();
  RTC.begin(DateTime(__DATE__, __TIME__));
  if (!SD.begin(4)) {
   Serial.println("initialization SD failed!");
  return;
  }
  Serial.println("initialization SD done.");
}
```

Запись данных о состоянии датчиков происходит в циклической процедуре `loop()`. Каждые 60 секунд получаем текущее время (в формате H:s) и температуру для текущего датчика и формируем строку для записи в файл текущего дня (формат y-m-d.txt). Для проверки выводим строку в монитор последовательного порта (рис. 15.2). Содержимое функции `loop()` представлено в листинге 15.3. Проверяем на компьютере содержимое файлов на SD-карте (рис. 15.3).

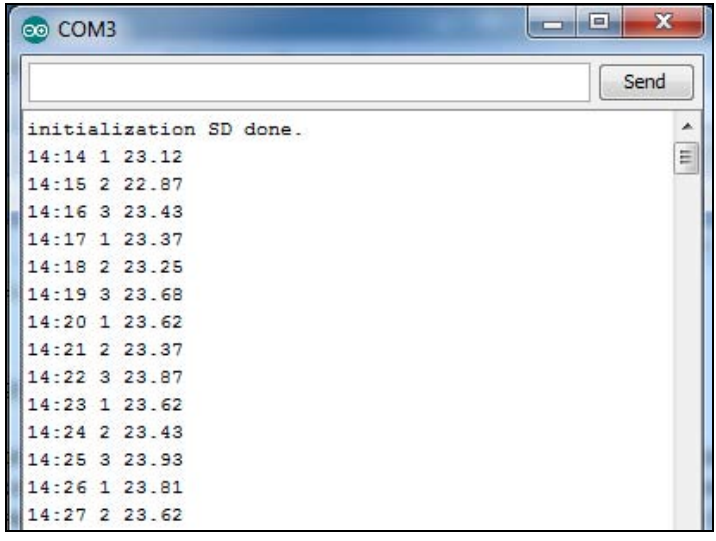


Рис. 15.2. Вывод данных в монитор последовательного порта

Листинг 15.3

```
void loop ()
 {
 // проверка, прошло 60 сек?
 if(millis()/60000!=gettemp)
```

```
  {
  gettemp=millis()/60000;
  // получить имя файла для текущего дня
  sfilename=get_file_name();
  sfilename.toCharArray(filename,20);
   if (SD.exists(filename))
   {Serial.println("file exists."); }
   else
   {Serial.println("file not exists."); }
   // открыть файл или создать
   myFile = SD.open(filename, FILE_WRITE);
   // получить температуру
   int Temp=get_temp(tek_sensor+1);
   sTemp=String(Temp/16);
   sTemp+=".";
   sTemp+=String(((Temp%16)*100)/16);
   // получить время H:m
   // создать запись для файла
   record=get_time();
   record+=" ";
   record+=String(tek_sensor+1);
   record+=" ";
   record+=sTemp;
   record=get_time();
   record+=" ";
   record+=String(tek_sensor+1);
   record+=" ";
   record+=sTemp;
   Serial.println(record);
   myFile.println(record);
   myFile.close();
   // указатель на следующий сенсор
   tek_sensor=(tek_sensor+1)%3;
   }
}
```

Функции получения имени файла `get_file_name()` и текущего времени `get_time()` представлены в листинге 15.4.

Листинг 15.4

```
// получение времени дня
String get_time()
 {
  String time1;
  dt = RTC.now();
```

```
  if(dt.hour()<10)
   time1="0"+String(dt.hour(),DEC);
  else
   time1=String(dt.hour(),DEC);
  if(dt.minute()<10)
   time1+=":0"+String(dt.minute(),DEC);
  else
   time1+=":"+String(dt.minute(),DEC);
  return time1;
 }
// получение имени файла для текущего дня
String get_file_name()
 {
  String filename1;
  dt = RTC.now();
  filename1+=String(dt.year()-2000,DEC);
  if(dt.month()<10)
   filename1+="-0"+String(dt.month(),DEC);
  else
   filename1+="-"+String(dt.month(),DEC);
  if(dt.day()<10)
   filename1+="-0"+String(dt.day(),DEC);
  else
   filename1+="-"+String(dt.day(),DEC);
  filename1+=".txt";
  return filename1;
 }
```

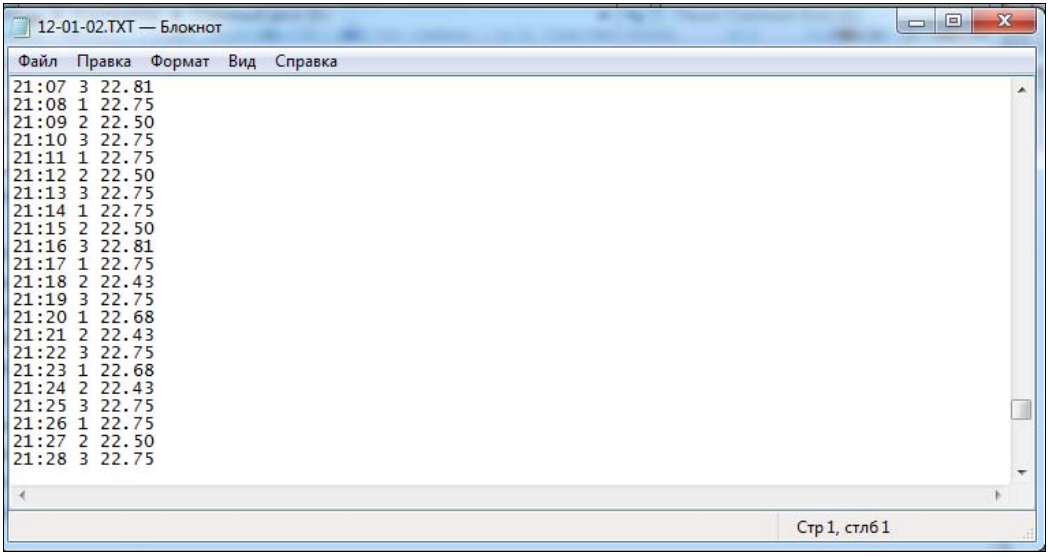


Рис. 15.3. Содержимое файлов с данными датчиков температуры

Полностью представленный здесь скетч можно найти в папке examples/ _15_1 сопровождающего книгу электронного архива. Напомню также, что библиотеки `SD` и `RTClib` находятся папке libraries этого же архива.

Рассмотренный пример имеет один недостаток — при пропадании питания теряется точное время. Поэтому, как вариант, можно использовать микросхему DS1307, где будет сохраняться точное время и в отсутствие питания на плате Arduino (при подключении к батарейке).

ГЛАВА 16

Arduino и светодиодные матрицы

16.1. Светодиоды и светодиодные матрицы

Светодиод, или LED (LightEmittingDiode), — это электронный источник света, представляющий собой полупроводниковый прибор, преобразующий пропускаемый через него электрический ток в световое излучение. Световое излучение возникает при рекомбинации носителей электрического заряда (электронов и дырок) в области p-n-перехода. При переходе электронов с одного энергетического уровня на другой осуществляется выделение энергии в виде излучения кванта света — фотона. При этом, как и в обычных диодах, электрический ток должен проходить через светодиод в прямом направлении, т. е. от p-стороны (анода) к n-стороне (катоду).

Светодиод состоит из одного или нескольких полупроводниковых кристаллов, пропитанных или лакированных примесями для создания p-n-перехода, размещенных в корпусе с контактными выводами и оптической системы, формирующей световой поток. Излучаемый полупроводниковым кристаллом свет попадает в миниатюрную оптическую систему, состоящую из рефлектора и прозрачного корпуса (линзы) светодиода. Таким образом, изменяя конфигурацию рефлектора и линзы можно добиться необходимого угла и направленности излучения светодиода.

Спектральные характеристики излучаемого светодиодом света напрямую зависят как от типа и химического состава используемых в нем полупроводниковых материалов, так и легирующих примесей, образующих p-n-переход. Светодиодное развитие началось с инфракрасных устройств, изготовленных на основе арсенида галлия. В настоящее время светодиоды изготавливаются из различных неорганических полупроводниковых материалов, излучая свет различных цветов. Но не все полупроводниковые материалы могут быть использованы для создания светодиодов, поскольку не все из них при рекомбинации могут эффективно излучать свет. К лучшим излучателям света относятся прямозонные полупроводники типа AIIIBV — например, арсенид галлия (GaAs) или фосфид индия (InP), и типа AIIBVI — например, селенид цинка (ZnSe), а такие непрямозонные полупроводники, как, например, кремний и германий, практически не излучают свет. Важнейшими полупроводниковыми материалами для создания светодиодов являются: алюминий (Al),

галлий (Ga), индий (In) и фосфор (P), создающие свечение в диапазоне от красного до желтого цвета; индий (In), галлий (Ga) и азот (N), используемые для получения голубого и зеленого цвета. Для получения белого цвета кристалл, излучающий голубой цвет, покрывают слоем люминофора — фосфором. Таким образом, варьируя состав полупроводников можно изготавливать светодиоды различных длин волн от ультрафиолета (GaN) до среднего инфракрасного диапазона (PbS).

RGB-светодиод — это три близко расположенных светодиода (красный, зеленый и синий под одной общей линзой, что соответствует его названию. Обычно у этих трех светодиодов объединены плюсовые (т. е. с общим анодом) или минусовые (с общим катодом) выводы — соответственно, всего у RGB-светодиода четыре вывода. Фактически, управление RGB — это управление тремя светодиодами, что позволяет получать произвольный цвет.

В последнее время широкое применение получили светодиодные матрицы, где светодиоды расположены в определенном порядке, а выводы сделаны в порядке, удобном для монтажа. Светодиодные матрицы бывают одноцветными, двухцветными и RGB (позволяют получать любой цвет). Рассмотрим примеры использования Arduino для управления светодиодными матрицами.

16.2. Светодиодная матрица FYM-23881BUG-11

В рассматриваемых далее проектах мы будем использовать светодиодную матрицу FYM-23881BUG-11, которая представляет собой набор из 64 светодиодов зеленого цвета, собранных в матрицу 8×8. Внешний вид матрицы изображен на рис. 16.1.

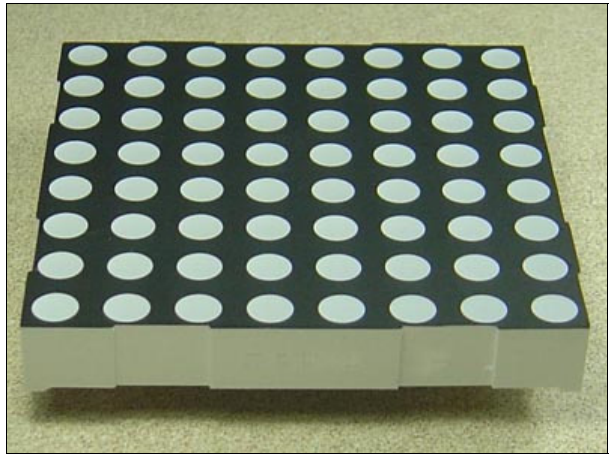


Рис. 16.1. Светодиодная матрица FYM-23881BUG-11

Характеристики этой матрицы:

- цвет свечения зеленый ультра (UltraGreen);
- длина волны 570 нм;

- интенсивность 60–110 мКд;
- схема с общим анодом;
- размеры (В×Ш×Г) 60,20 × 60,20 × 9,20 мм;
- размер точки 5,00 мм;
- расстояние между точками (X/Y) 7,62/7,62 мм;
- количество выводов (рис. 16.2) — 16 (8 анодов, 8 катодов).

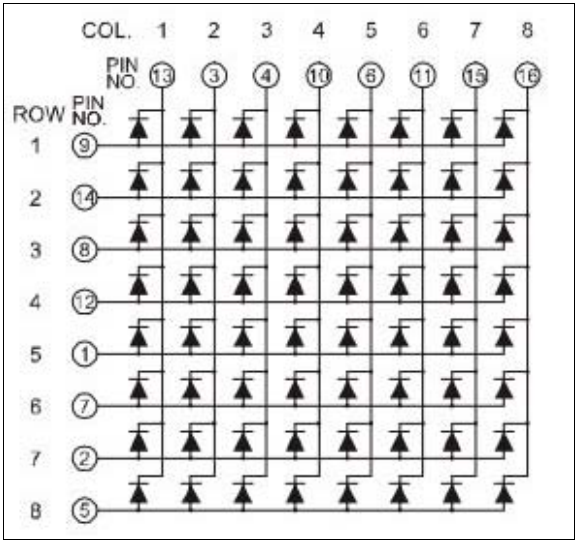


Рис. 16.2. Расположение выводов матрицы FYM-23881BUG-11

16.3. SPI-расширитель выходов 74HC595

Чтобы управлять большим количеством пинов светодиодной матрицы, не обязательно использовать большое количество выходов Arduino. Можно обойтись всего тремя. Применим для этого сдвиговый регистр. Рассмотрим микросхему 74HC595 — восьмиразрядный сдвиговый регистр с последовательным вводом, последовательным или параллельным выводом информации, с триггером-защелкой и тремя состояниями на выходе (рис. 16.3).

Назначение выводов микросхемы 74HC595 приведено в табл. 16.1.

Таблица 16.1. *Назначение выводов микросхемы 74HC595*

Контакт	Обозначение	Назначение
1–7, 15	Q0–Q7	Параллельные выходы
8	GND	Земля
9	Q7''	Выход для последовательного соединения регистров

Таблица 16.1 *(окончание)*

Контакт	Обозначение	Назначение
10	MR	Сброс значений регистра. Сброс происходит при получении LOW
11	SH_CP	Вход для тактовых импульсов
12	ST_CP	Синхронизация ("защелкивание") выходов
13	OE	Вход для переключения состояния выходов из высокоомного в рабочее
14	DS	Вход для последовательных данных
16	Vcc	Питание 2–6 В

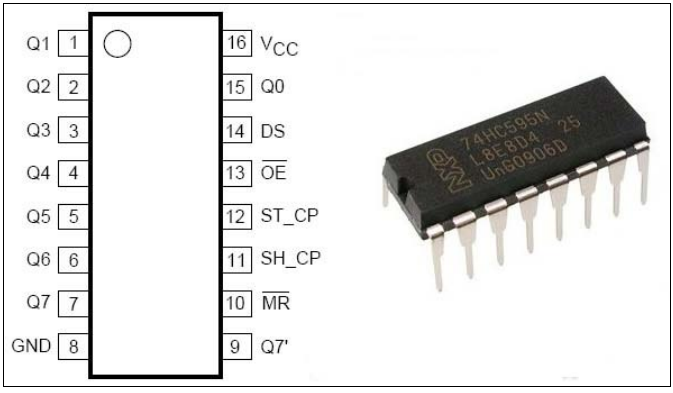


Рис. 16.3. Микросхема 74HC595

Для управления нам вполне достаточно всего лишь трех выводов: `SH_CP`, `ST_CP` и `DS`. Когда на тактовом входе `SH_CP` появляется логическая единица, регистр считывает бит со входа данных `DS` и записывает его в самый младший разряд. При поступлении на тактовый вход следующего импульса все повторяется, только бит, записанный ранее, сдвигается на один разряд, а его место занимает вновь пришедший бит. Когда все восемь битов заполнились, и приходит девятый тактовый импульс, регистр снова начинает заполняться с младшего разряда, и все повторяется вновь. Чтобы данные появились на выходах `Q0...Q7`, нужно их "защелкнуть". Для этого необходимо подать логическую единицу на вход `ST_CP`. Что бы мы ни делали с регистром, данные на выходах не изменятся, пока мы вновь не "защелкнем" их.

Вывод `Q7"` предназначен для последовательного (каскадного) соединения сдвиговых регистров. При таком подключении биты из первого регистра будут проталкиваться в следующий в каскаде регистр, из него — в следующий, и т. д. Таким образом, каскад из двух 8-битных регистров будет работать как один 16-битный. Можно соединить хоть десять штук.

16.4. Игра "Тетрис" на светодиодных матрицах FYM-23881BUG-11

Создадим небольшую игровую приставку. Дисплей ее состоит из двух светодиодных матриц FYM-23881BUG-11 размерностью 8×8. Для управления двумя матрицами (128 светодиодов) задействованы три выхода Arduino (8, 11, 13) и четыре микросхемы 74HC595. В качестве клавиатуры возьмем аналоговую клавиатуру (рис. 16.4), для подключения которой к Arduino требуется один аналоговый вход (рис. 16.5). Электрическая схема приставки представлена на рис. 16.6.

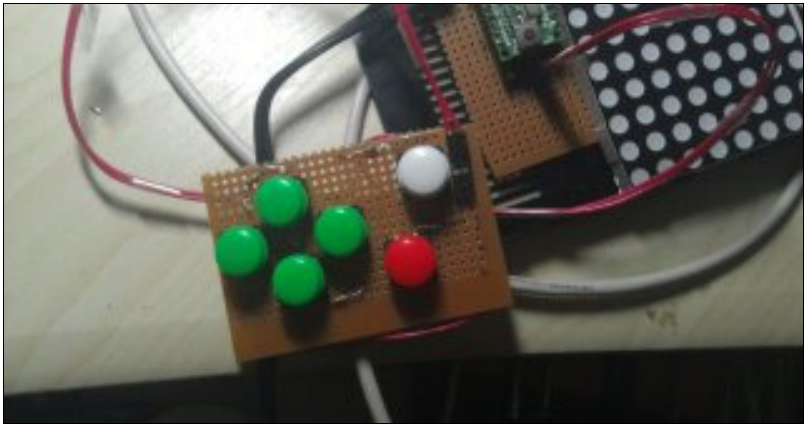


Рис. 16.4. Аналоговая клавиатура для игровой приставки

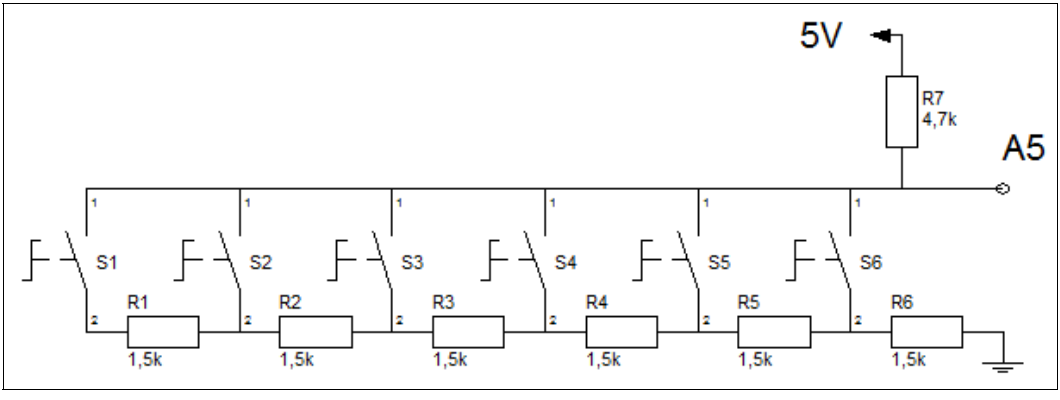


Рис. 16.5. Схема аналоговой клавиатуры для Arduino

Предварительно необходимо клавиатуру откалибровать. Если не нажата ни одна из кнопок, на вход A5 через резистор 4,7 кОм подается напряжение 5 В. Для этого напишем небольшой скетч (листинг 16.1). Будем выводить в последовательный порт значения, считываемые на входе A5, при нажатии на разные кнопки.

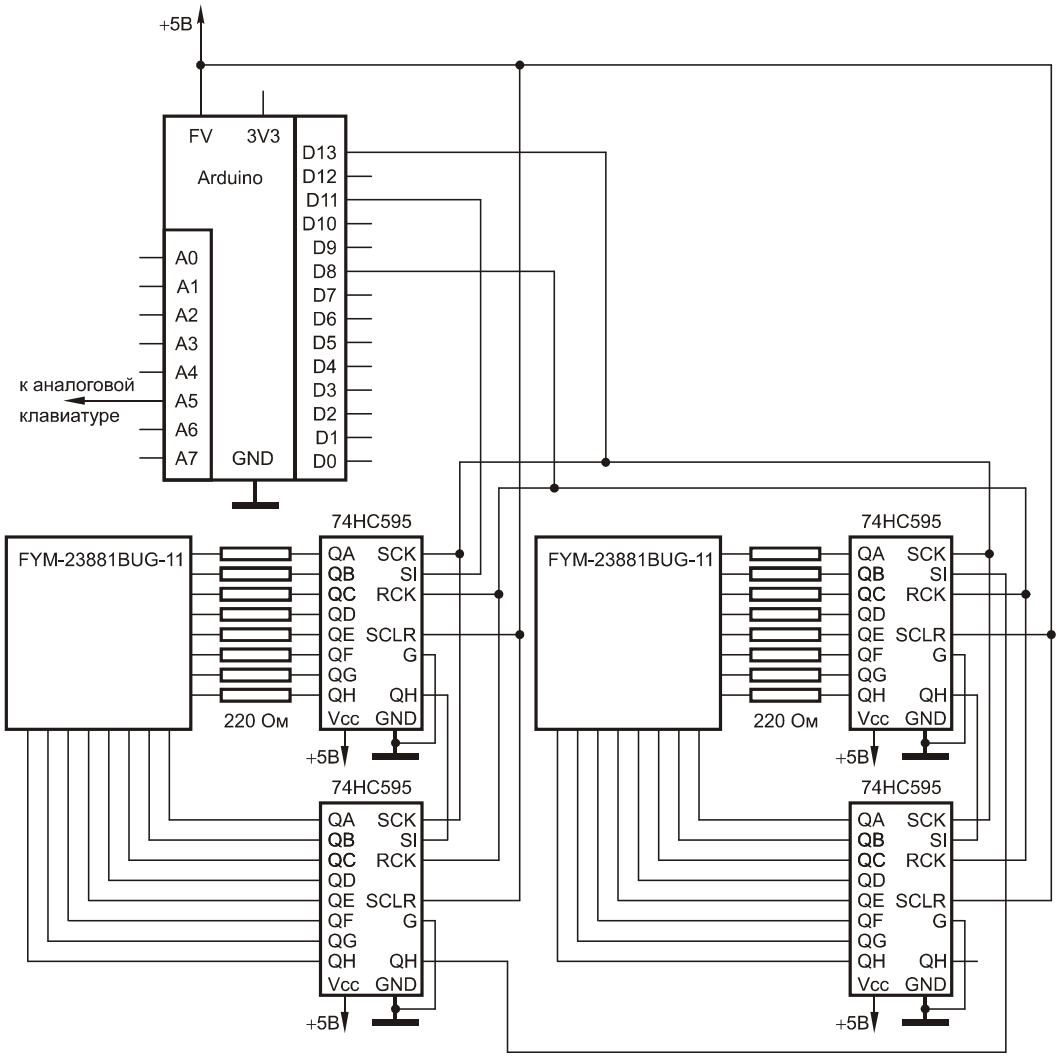


Рис. 16.6. Электрическая схема приставки

Листинг 16.1

```
void loop() {
 int valbutton;
 game1();
 valbutton=analogRead(keyboardPin);
 Serial.println(valbutton);
}
```

Внешний вид приставки в разобранном состоянии представлен на рис. 16.7.

Теперь можно приступать к написанию программы.

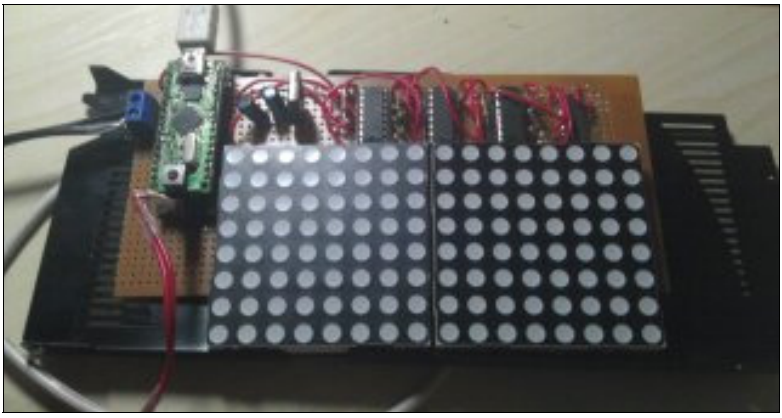


Рис. 16.7. Вид игровой приставки

16.4.1. Управление изображением на светодиодных матрицах

Данные для вывода на экран хранятся в массиве `byte displayArray[24]`, где верхние и нижние 4 байта служебные (будут использоваться в игре "Тетрис"), а средние 16 байтов и есть данные для двух матриц.

Для реализации динамической индикации на выходах 1 и 3 микросхемы будет выбираться один из столбцов, подаваемых на выходы 2 и 4 микросхемы. Обновление информации происходит 1 раз в 1000 мкс. При этом мелькания нет.

Управление светодиодами основано на внутреннем обработчике прерываний. Применение обработчика прерываний позволяет нам обновлять матрицу вне основного цикла программы, как бы в параллельном процессе. Для работы с прерываниями воспользуемся Arduino-библиотекой `Timer2`. Файлы этой библиотеки вы можете найти в папке libraries/MsTimer2 сопровождающего книгу электронного архива. Код данного фрагмента представлен в листинге 16.2.

Листинг 16.2

```
 // библиотека для прерываний по таймеру
#include <MsTimer2.h>
byte displayArray[24] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
void setup() {
 // инициализация SPI:
 pinMode (displayPin, OUTPUT);
 SPI.begin();
 // запуск прерывания по таймеру
 MsTimer2::set(2, showDisplay);
 MsTimer2::start();
```

```
 // очистка дисплея
 clearDisplay()
}
void loop() {;}
// обработка прырывания по таймеру
// динамическая индикация
void showDisplay() {
 for(byte i=0;i<8;i++)
 {
 // SS=0 – защелкнуть выводы
 digitalWrite(displayPin,LOW);
 // 4,3
 byte bhigh=displayArray[(i+OFFSET_DISPLAY1.offsetY+8+16)%16+4];
 bhigh=circleShift(bhigh,OFFSET_DISPLAY1.offsetX);
 SPI.transfer(255-bhigh);
 SPI.transfer(B00000001<<i);
 // 2,1
 byte blow=255-displayArray[(i+OFFSET_DISPLAY1.offsetY+16)%16+4];
 blow=circleShift(blow,OFFSET_DISPLAY1.offsetX);
 SPI.transfer(blow);
 SPI.transfer(B00000001<<i);
 // SS=1 – вывести данные на выводы 74HC595
 digitalWrite(displayPin,HIGH);
 // 1000 мкс
 delayMicroseconds(1000);
 }
}
// циклический сдвиг байта
byte circleShift(byte var,int offset)
 {
 byte varnew;
 for(int i=0;i<8;i++)
 bitWrite(varnew,i,bitRead(var,(i+offset+8)%8));
 return varnew;
 }
// очистка дисплея
void clearDisplay() {
 for(int i=0;i<sizeof(displayArray);i++)
  displayArray[i]=0;
}
```

16.4.2. Фигуры игры "Тетрис"

Для реализации игры создадим массив со всеми изображениями фигур для "Тетриса" `arrFigure[][]`, массив указателей на новую фигуру при повороте `transFigure[]`, а также структуру `FIGURA` для сохранения информации о текущей фигуре (листинг 16.3).

Листинг 16.3

```
byte arrFigure[16][5]={
  {0,0,0,0,0},{15,0,0,0,4},{1,1,1,1,1},{3,3,0,0,2},
  {7,1,0,0,3},{7,4,0,0,3},{1,7,0,0,3},{4,7,0,0,3},
  {3,1,1,0,2},{3,2,2,0,2},{1,1,3,0,2},{2,2,3,0,2},
  {7,2,0,0,3},{2,3,2,0,2},{2,7,0,0,3},{1,3,1,0,2}};
byte transFigure[16]={0,2,1,3,9,8,10,11,5,7,4,6,13,14,15,12};
struct FIGURA  // структура для хранения фигуры на экране
{
 int driving; // 1 - в движении
 int offsetX; // смещение по x
 int offsetY; // смещение по y
 int dX;   // изменение влево-вправо
 int dY;   // dy
 int dT;   // 1 - поворот 90 градусов
 int c; // номер фигуры
 int length; // ширина фигуры
 long millis1;
};
FIGURA FIGURA1={0,3,20,0,0,0,0,0,0};
```

В массиве `arrFigure[][5]` — первые четыре байта используются для прорисовки изображения фигуры, пятый байт — ширина фигуры.

Добавление фигуры на поле выполняет процедура `add_figura()` (листинг 16.4). Фигура выбирается с помощью функции `random()`. В структуру `FIGURA` заносятся следующие начальные значения:

- `offsetY` — номер фигуры, выдаваемый функцией `random()`;
- `offsetX=3`;
- `int offsetY=20` — верх буфера экрана;
- `c` — номер фигуры;
- `length` — ширина;
- `FIGURA1.driving=1` — фигура движется.

и изображение фигуры выводится в буфер экрана. Данные при этом заносятся в верхнюю невидимую часть буфера экрана (байты 20–23 массива `displayArray`).

Листинг 16.4

```
void add_figura()
 {
 FIGURA1.figura=random(1,sizeof(transFigure));
 FIGURA1.figura=millis()%sizeof(transFigure);
```

```
 FIGURA1.dX=0;FIGURA1.dY=0;FIGURA1.dT=0;
 FIGURA1.offsetX=3;FIGURA1.offsetY=20;
 FIGURA1.millis1=millis();
 FIGURA1.length=arrFigure[FIGURA1.figura][4];
 for(int i=0;i<4;i++)
  {displayArray[FIGURA1.offsetY+i]=
    arrFigure[FIGURA1.figura][i]<<FIGURA1.offsetX;}
 FIGURA1.driving=1;
 }
```

Рассмотрим, как осуществляется движение фигуры вниз.

Через время, установленное в переменной `speed2` (меняется нажатием на белую кнопку), происходит стирание из буфера экрана фигуры (процедура `figura_clear()`), вычисление нового положения фигуры и прорисовка (занесение в буфер экрана) фигуры в измененных координатах (процедура `figura_driving()`). Данная часть скетча представлена в листинге 16.5.

Листинг 16.5

```
// очищение изображения фигуры
void figura_clear(int y,int x)
 {
 for(int i=0;i<4;i++)
  displayArray[y+i]=displayArray[y+i]-(arrFigure[FIGURA1.figura][i]<<x);
 }
// перемещение фигуры
void figura_driving(int y,int x,int figura)
 {
 for(int i=0;i<4;i++)
  displayArray[y+i]=displayArray[y+i]+(arrFigure[figura][i]
   <<FIGURA1.offsetX);
 }
```

16.4.3. Управление фигурами игры "Тетрис"

Назначение кнопок клавиатуры:

- зеленая левая/правая — фигура влево/вправо;
- зеленая вверх — поворот фигуры но 90 градусов;
- белая — увеличение скорости на 25 пунктов;
- красная — выход из игры.

Содержимое скетча для обработки нажатий кнопок представлено в листинге 16.6.

Листинг 16.6

```
// обработка нажатия кнопки
int buttonClick1(int val)
 {
 if(val>650) {KEYS1.button=1;return 1;}
 if(val>600) {KEYS1.button=2;return 2;}
 if(val>530) {KEYS1.button=3;return 3;}
 if(val>450) {KEYS1.button=4;return 4;}
 if(val>300) {KEYS1.button=5;return 5;}
 if(val>200) {KEYS1.button=6;return 6;}
 return 0;
 }
void buttonClick2(int val)
 {
 if(millis()-KEYS1.millisbutton[val]<100)
  return;
 KEYS1.millisbutton[val]=millis();
 KEYS1.button=val;
 switch(val) {
  case 1:
    FIGURA1.dX=FIGURA1.dX-1;
    break;
  case 2:
    FIGURA1.dT=1;    // (поворот 90)
    Serial.println("up");
   break;
  case 3:
   break;
  case 4:
    FIGURA1.dX=FIGURA1.dX+1;
    break;
  case 5:
    // переключение на другую программу
    speed2=max(speed2-25,5);
   break;
  case 6:
    clearDisplay();
    tekgame=(tekgame+1)%3;
    break;
  default:
    break;
 }
 }
```

16.4.4. Проверка столкновения фигур

Перед изменением положения фигуры (вниз, влево, вправо, поворот) необходимо производить проверку на столкновение с другими (уже неподвижными фигурами) и выход из границ. Для движения влево, вправо и поворота изменение положения производится до движения вниз. Для движения вниз при положительной проверке на столкновение происходит останов фигуры (`FIGURA1.driwing=0`). Содержимое процедур проверки на столкновение `collision1()` и `collision2()` представлено в листинге 16.7.

Листинг 16.7

```
// контроль столкновения с кирпичиками при повороте
 boolean collision1(int yd,int figura,int figurapr,int dx,int ddx)
 {
 byte opand,disp;
 boolean collision=false;
 for(int i=0;i<4;i++) // по высоте фигуры - 4
  {
  disp=displayArray[yd+i];
  disp=disp-(arrFigure[figurapr][i]<<dx);
  opand=disp & (arrFigure[figura][i]<<ddx);
  if(opand>0)
   {collision=true;}
  }
 return collision;
 }
 // контроль столкновения с кирпичиками вниз
 boolean collision2(int yd,int figura,int figurapr,int dx,int ddx)
 {
 byte opand,disp;
 boolean collision=false;
 for(int i=0;i<4;i++) // по высоте фигуры - 4
  {
  disp=displayArray[yd+i];
  if(i>0)
   disp=disp-(Figure[figurapr][i-1]<<dx);
  opand=disp & (arrFigure[figura][i]<<dx);
  if(opand>0)
   {collision=true;}
  }
 return collision;
 }
```

И код главной поцедуры — `game2()` — представлен в листинге 16.8.

Листинг 16.8

```
// игра тетрис
boolean game2()
{
if(FIGURA1.driving==0)
 add_figura();
if((millis()-FIGURA1.millis1)>speed2 && FIGURA1.driving>0)
 {
 //figura_clear(FIGURA1.offsetY);
 // по X
 if(FIGURA1.dX!=0)
   {
   int dxpr=FIGURA1.dX;
   int offxpr=FIGURA1.offsetX+dxpr;
   offxpr=max(offxpr,0);
   offxpr=min(offxpr,8-FIGURA1.length);
   if(!collision1(FIGURA1.offsetY,FIGURA1.figura,
     FIGURA1.figura,FIGURA1.offsetX,offxpr))
   {
   figura_clear(FIGURA1.offsetY,FIGURA1.offsetX);
   FIGURA1.offsetX=FIGURA1.offsetX+FIGURA1.dX;
   FIGURA1.offsetX=max(FIGURA1.offsetX,0);
   FIGURA1.offsetX=min(FIGURA1.offsetX,8-FIGURA1.length);
   FIGURA1.dX=0;
   Serial.println(FIGURA1.offsetX);
   figura_driving(FIGURA1.offsetY,0,FIGURA1.figura);
   }

   //figura_clear(FIGURA1.offsetY);
  }
 // поворот 90 градусов
 if(FIGURA1.dT>0)
   {
   if(!collision1(FIGURA1.offsetY,transFigure[FIGURA1.figura],
     FIGURA1.figura,FIGURA1.offsetX,FIGURA1.offsetX))
   {
   figura_clear(FIGURA1.offsetY,FIGURA1.offsetX);
   figura_driving(FIGURA1.offsetY,0,transFigure[FIGURA1.figura]);
   FIGURA1.figura=transFigure[FIGURA1.figura];
   FIGURA1.length=arrFigure[FIGURA1.figura][4];
   }
   FIGURA1.dT=0;
   //figura_clear(FIGURA1.offsetY);
  }
 // вниз по времени
 if(collision2(FIGURA1.offsetY-1,FIGURA1.figura,FIGURA1.figura,
   FIGURA1.offsetX,FIGURA1.offsetX))
  {FIGURA1.millis1=millis();FIGURA1.driving=0;
```

```
  if(FIGURA1.offsetY==20)
   return false;
  else
   return true;
  }
  figura_clear(FIGURA1.offsetY,FIGURA1.offsetX);
  figura_driving(FIGURA1.offsetY-1,0,FIGURA1.figura);
  FIGURA1.offsetY=FIGURA1.offsetY-1;
  if(FIGURA1.offsetY<5)
   FIGURA1.driving=0;
  FIGURA1.dX=0;FIGURA1.dY=0;FIGURA1.dT=0;
  FIGURA1.millis1=millis();
   }
 return true;
 }
```

Все — игра готова. Полностью данный скетч можно найти в папке examples/_16_1 сопровождающего книгу электронного архива.

16.5. Светодиодная матрица RGB

Гораздо более интересны светодиодные матрицы RGB — они позволяют воспроизводить любой цвет. Рассмотрим светодиодную матрицу RGB типа GTM2088 раз-

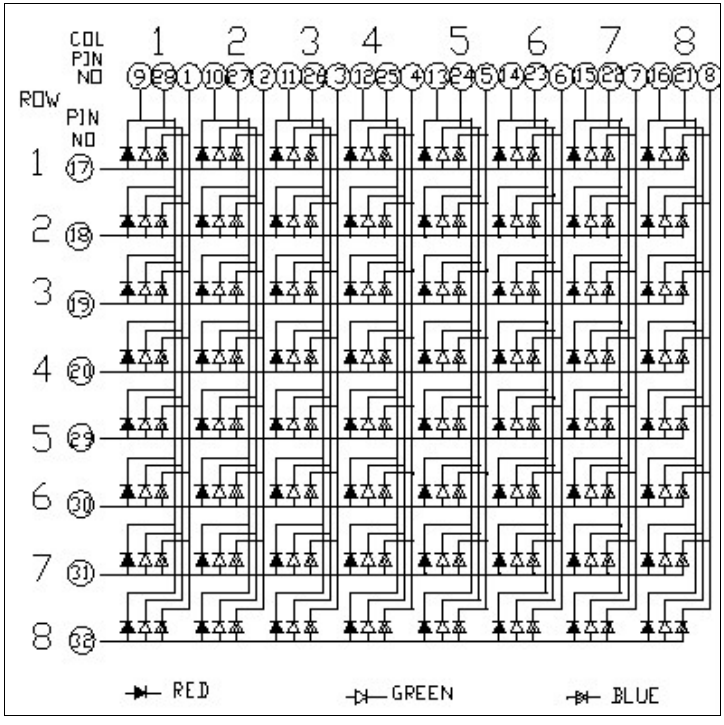


Рис. 16.8. Расположение выводов RGB-матрицы GTM2088

мерности 8×8 с общим анодом. Схема расположения ее выводов представлена на рис. 16.7. Сама матрица имеет 32 входа: 8 анодов, 8 катодов красного цвета, 8 — зеленого и 8 — синего.

16.6. RGB-ночник, управляемый с помощью движения рук

Создадим проект с использованием светодиодной матрицы RGB типа GTM2088. Пусть это будет RGB-ночник с возможностью управления его цветом движениями рук. Яркость каждой из трех составляющих RGB-цвета будет изменяться при приближении/удалении руки. Помогут нам в этом три датчика расстояния HC-SR04. В качестве микроконтроллера возьмем плату Arduino. Для управления матрицей будут использоваться три выхода Arduino (8, 11,13) и четыре микросхемы 74HC595. Электрическая схема ночника представлена на рис. 16.9.

Для проекта потребовались следующие детали:

- контроллер Arduino — 1 шт;
- 8×8 RGB-матрица — 1 шт;
- ультразвуковой датчик расстояния HC-SR04 — 3 шт;
- сдвиговый регистр — микросхема 74HC595 — 4 шт;
- резистор 220 Ом — 24 шт (12 руб.);
- блок питания 5 В 3А (Китай) — 1 шт.;
- плафон с подставкой;
- провода, припой и пр.

Внешний вид ночника представлен на рис. 16.10.

При разработке скетча были заданы следующие параметры:

- расстояние 1–20 см, 1 — максимальная яркость, 20 — нулевая;
- датчики расстояния в цикле считывают данные и применяют полученное расстояние для установления яркости;
- если за цикл расстояние меняется больше, чем на 20 см (рука уходит в сторону) — эта яркость фиксируется для данного цвета.

Яркость задается подачей ШИМ-сигнала на выводы матрицы для групп R, G и B. Частота ШИМ-сигнала примерно 60 Гц. Сигнал ШИМ формируется следующим образом.

Например, расстояние равно 5 см. Тогда сигнал ШИМ:

- 15 – (5 – 1) = 11 циклов прерывания, светодиоды данного цвета горят;
- 5 циклов — светодиоды данного цвета не горят;
- на анод всегда подается 1.

Управление светодиодами основано на внутреннем обработчике прерываний. Для работы с прерываниями воспользуемся Arduino-библиотекой `Timer2`.

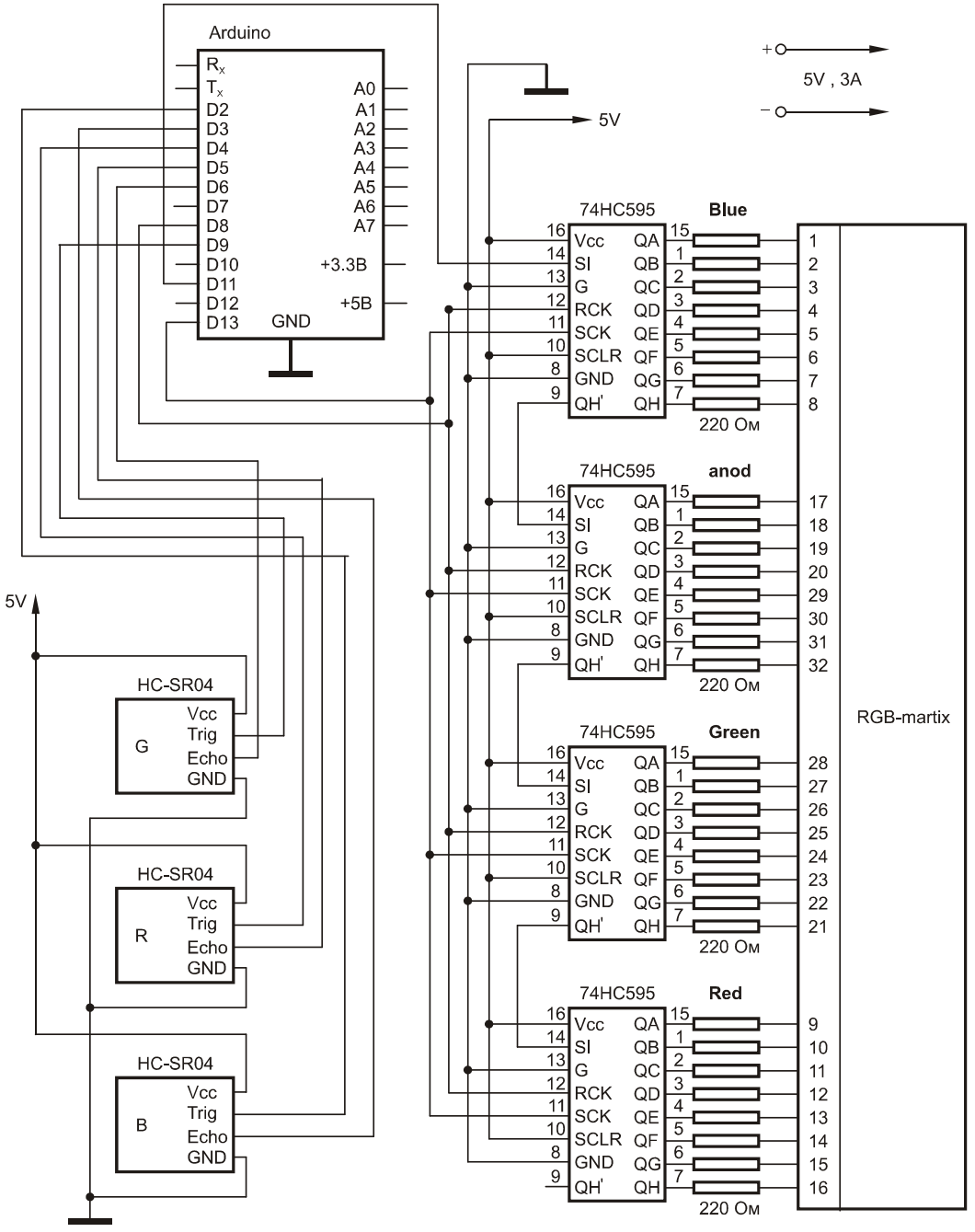


Рис. 16.9. Электрическая схема ночника

Рис. 16.10. Внешний вид RGB-ночника

Содержимое скетча представлено в листинге 16.9.

Листинг 16.9

```
// библиотека SPI
#include "SPI.h"
const int displayPin = 8;

// библиотека для прерываний по таймеру
#include "MsTimer2.h"
int offset=0;
// библиотека для ультразвукового датчика
#include "Ultrasonic.h"

Ultrasonic ultrasonic1(4,5); // красный
Ultrasonic ultrasonic2(9,6); // зеленый
Ultrasonic ultrasonic3(2,3); // синий
// структура для хранения значений R, G, B
struct RGB
{
 int r;   // позиция R
```

```
 int g;   // позиция G
 int b;   // позиция B
 int offset; // кол-во ШИМ-циклов
};
RGB RGB1={0,0,0,0};

void setup() {

 // set the slaveSelectPin as an output:
 pinMode (displayPin, OUTPUT);
 // initialize SPI:
 SPI.begin();
 // запуск прерывания по таймеру
 MsTimer2::set(1, showDisplay); //
 MsTimer2::start();
 //clearDisplay();
 Serial.println("led matrix");

}
int dist=0;
void loop() {
 delay(100);
 // для красного
 dist = (int)ultrasonic1.Ranging(CM); // считать для R
 if(dist>0 && dist<20)
  {RGB1.r=dist;}
 // для зеленого
 dist = (int)ultrasonic2.Ranging(CM); // считать для R
 if(dist>0 && dist<20)
  {RGB1.g=dist;}
 // для синего
 dist = (int)ultrasonic3.Ranging(CM); // считать для R
 if(dist>0 && dist<20)
  {RGB1.b=dist;}
}
// обработка прырывания по таймеру
// динамическая индикация
void showDisplay() {
 // инкремент счетчика
 RGB1.offset=max(2,(RGB1.offset+1)%15);
 digitalWrite(displayPin,LOW);
 // вывод данных
 if(RGB1.b меньше RGB1.offset)   // синий
  SPI.transfer(B00000000);
 else
  SPI.transfer(B11111111);
 SPI.transfer(B11111111); // анод
```

```
 if(RGB1.g меньше RGB1.offset)    // зеленый
  SPI.transfer(B00000000);
 else
  SPI.transfer(B11111111);
 if(RGB1.r меньше RGB1.offset)  // красный
  SPI.transfer(B00000000);
 else
  SPI.transfer(B11111111);
 // take the SS pin high
 digitalWrite(displayPin,HIGH);

}
```

Полностью данный скетч можно найти в папке examples/_16_2 сопровождающего книгу электронного архива.

ГЛАВА 17

Работа Arduino с купюроприемником

Продажа товаров и услуг с помощью автоматизированных систем (торговых автоматов) называется *вендингом*. Вендинг получил широкое распространение в мире как удобный и не очень требовательный способ вести торговлю или оказывать услуги. На рынке представлен большой выбор торговых автоматов. Если вас не устраивает цена, или вы желаете добавить функционала действующему автомату, или решили собрать что-то новенькое — Arduino в качестве микроконтроллера для управления отдельными блоками устройства вам в этом поможет.

Рассмотрим работу Arduino с купюроприемником. Купюроприемник (тот же банкнотник, купюрник, валидатор, биллакцептор) — устройство, предназначенное для приема наличных платежей банкнотами. Купюроприемники устанавливают в платежные терминалы, развлекательные, лотерейные и игровые автоматы, в вендинговые (торговые) автоматы. Купюроприемники осуществляют прием купюр, распознавание номинала и (при наличии механизма укладки и кассеты) хранение принятых купюр.

17.1. Купюроприемник ICT серий A7 и V7

Купюроприемники ICT серий A7 и V7 (MDB-версия) по-настоящему проверены временем — на российском рынке эта модель появилась еще в 2003 году и с успехом эксплуатируется до сих пор как в индустрии игорного бизнеса, так и в вендинге и системах моментальных платежей (рис. 17.1). Предоставляет возможность быстрой замены микропрограммы с помощью ПК. Модель поставляется с механизмом укладки на 200, 400 и 800 купюр.

Основные возможности:

- ❒ осуществляет прием купюры, поданной в любом направлении;
- ❒ уровень распознавания купюр — более 96 %;
- ❒ время операции — 3 секунды (включая время укладки);
- ❒ обеспечивает работу по протоколу PULSE, RS 232, MDB;
- ❒ контейнер укладчика на 300 купюр;

- напряжение питания — 12 В;
- позволяет обновлять микропрограмму в Flash Rom;
- имеется модель с горизонтальным механизмом укладки.

Рис. 17.1. Купюроприемник ICT V7

Немаловажный факт — в Интернете можно приобрести работающие б/у экземпляры по цене до 1000 рублей (лично я приобрел два у разных продавцов).

Для настройки купюроприемник имеет три блока переключателей. Первый и второй блоки переключателей находятся с боковой стороны купюроприемника (рис. 17.2). Третий блок спрятан. Разворачиваем купюроприемник к себе лицевой стороной и кассетой вверх. Над лицевой панелью находим пластиковую крышку. Чтобы она открылась, достаточно потянуть ее немного вверх. Вы увидите плату и, поискав, найдете колодку с четырьмя переключателями (рис. 17.3).

Каждый переключатель (в просторечии дип) имеет два положения: on и off. Рассмотрим назначение переключателей.

- Первый блок:
  - sw 1–5 — настройка приема по номиналам купюр;
  - sw 6 — качество приема. Если установить `HighAcceptance`, то купюроприемник принимает даже помятые купюры;

- sw7 — отключает прием `Inhibit` (сигнал, устанавливающий блокировку приема купюр);
- sw 8 — настройка уровня сигнала `Inhibit`.

❒ Второй блок:

- sw 1–2 — настройка количества импульсов;
- sw 3–4 — настройка длины импульса и длины паузы.

Сводная таблица настройки переключателей первого и второго блоков представлена в табл. 17.1.

Рис. 17.2. Первый и второй блоки переключателей купюроприемника ICT V7

❒ Третий блок (который под крышкой):

- sw 1 — полярность импульса `High` (on) или `Low` (off);
- sw 2 — протокол `pulse` (on) или `rs232` (off);
- sw 3–4 — не используются.

Чтобы любые изменения вступили в силу, купюроприемник надо перезагрузить.

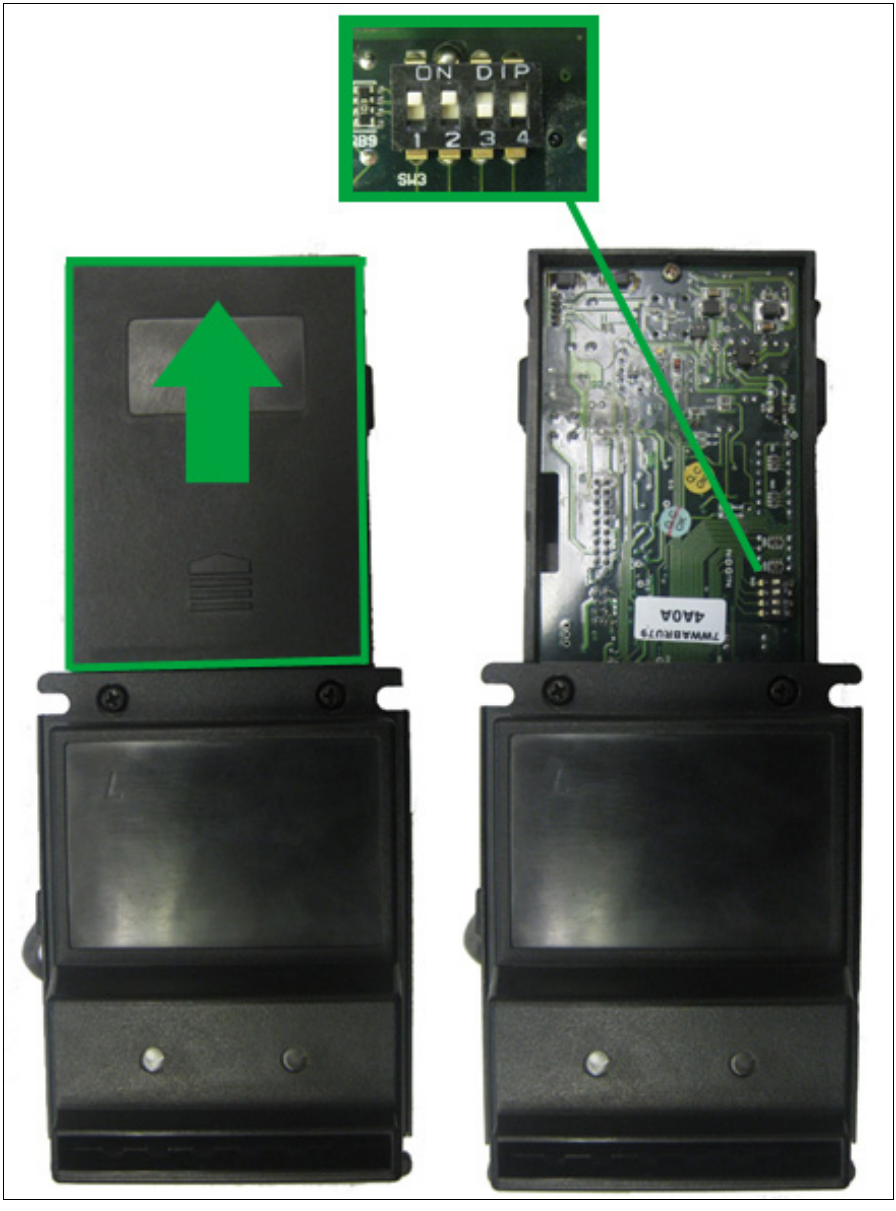


Рис. 17.3. Третий блок переключателей купюроприемника ICT V7

Таблица 17.1. *Таблица настройки переключателей первого и второго блока*

Функция	Первый блок								Второй блок			
	1	2	3	4	5	6	7	8	1	2	3	4
Reject Ruble 10	on											
Accept Ruble 10	off											
Reject Ruble 50		on										
Accept Ruble 50		off										
Reject Ruble 100			on									
Accept Ruble 100			off									
Reject Ruble 500				on								
Accept Ruble 500				off								
Reject Ruble 1000					on							
Accept Ruble 1000					off							
High Acceptance						on						
High Security						off						
Harness disable							on					
Harness enable							off					
Inhibit Active High								on				
Inhibit Active Low								off				
1 pulse / Ruble 10									off	off		
2 pulse / Ruble 10									on	off		
5 pulse / Ruble 10									off	on		
20 pulse / Ruble 10									on	on		
50ms on/50 ms off											off	off
60ms on/300 ms off											on	off
30ms on/50 ms off											off	on
150ms on/150 ms off											on	on

17.2. Подключение купюроприемника ICT V7 к Arduino

Купюроприемник оснащен кабелем для подключения к автомату, на конце кабеля имеется разъем 3×3 (рис. 17.4). Назначение интересующих нас выводов следующее:

- красный — 12 В;
- оранжевый — GND;

- желтый — `INHIBIT+`;
- зеленый — `INHIBIT-`;
- синий — `SIGNAL+`;
- фиолетовый — `SIGNAL-`.

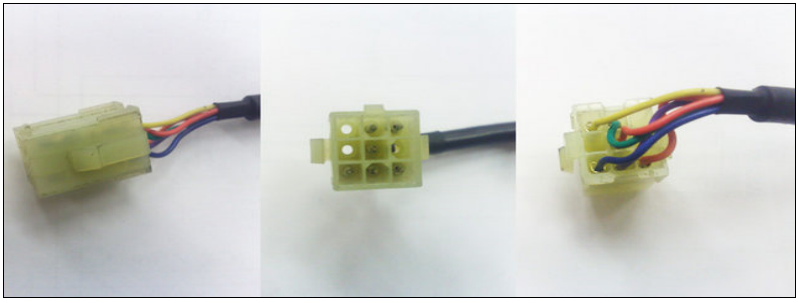


Рис. 17.4. Кабель подключения купюроприемника ICT V7

Подключаем купюроприемник к Arduino с помощью протокола `pulse`. Установка переключателей приведена в табл. 17.2.

Таблица 17.2. *Таблица установки переключателей*

Первый блок								Второй блок				Третий блок			
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
off	off	off	on	on	off	off	off	off	off	off	off	on	on	off	off

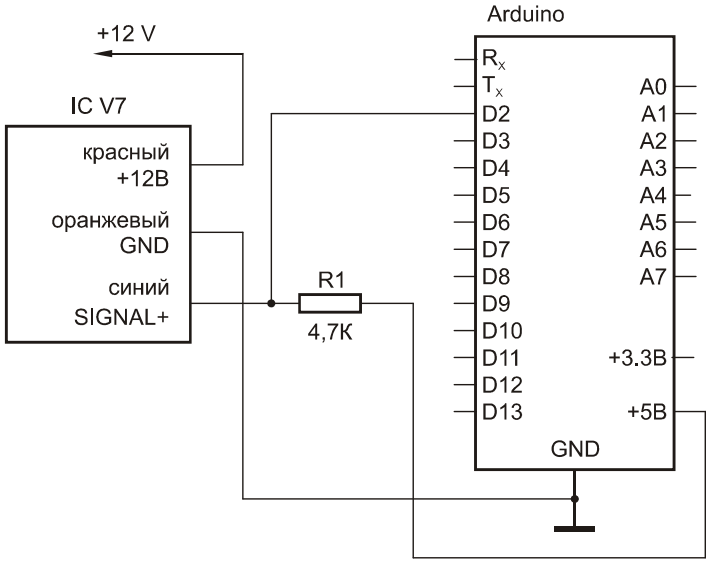


Рис. 17.5. Схема подключения купюроприемника ICT V7 к Arduino

Я установил режим `pulse` с полярностью `High`, 1 импульс на 10 руб. (5 — на 50 руб., 10 — на 100 руб.), длительность импульса 50 мсек, качество приема низкое, отсутствие сигнала блокировки купюроприемника и прием 10, 50, 100 рублевых купюр с отторжением 500- и 1000-рублевых купюр.

Схема подключения приведена на рис. 17.5.

17.3. Скетч для получения номинала принимаемой купюры

Выход `D2` Arduino подсоединен к выходу `SIGNAL+` купюроприемника и установлен в режим получения данных (`INPUT`). Через резистор 4,7 кОм (см. рис. 17.5) он подтянут к питанию +5 В и на нем находится уровень `HIGH`. После получения купюры купюроприемник посылает импульсы продолжительностью 50 мсек на выход `SIGNAL+`, устанавливая на входе `D2` уровень `LOW` и вызывая процедуру обработки прерывания, где инкрементируется счетчик количества импульсов. В основном цикле программы проверяется время, прошедшее после получения первого импульса, и если оно превысило 1000 мсек, выводится номинал полученной купюры (на основе количества импульсов), а счетчик обнуляется до поступления первого импульса при приеме новой купюры.

Код скетча представлен в листинге 17.1.

Листинг 17.1

```
const int moneyPin=2;      // подключение SIGNAL+
int money=0;          // номинал принятой купюры
unsigned long timeAllPulse=2000; // макс. время приема купюры
unsigned long timeCount=0;

void setup()
{
 Serial.begin(9600);
 pinMode(moneyPin,INPUT);
 attachInterrupt(0,count_pulse,FALLING);
 Serial.println("ready");
}

void loop()
{
 // прошло максимальное время приема купюры? - вывести номинал
 if(money>0 && (millis()-timeCount)>timeAllPulse)
  {
  Serial.print("money=");
  Serial.println(10*money);
```

```
  money=0;
  }
}
// обработка прерывания - получить кол-во импульсов
//
void count_pulse()
 {
 detachInterrupt(0);
 money++;
 if(money==1)
  timeCount=millis();
 attachInterrupt(0,count_pulse,FALLING);
 }
```

Код данного скетча находится в папке examples/_17_1 сопровождающего книгу электронного архива.

ГЛАВА 18

Arduino и радиочастотная идентификация (RFID)

18.1. Радиочастотная идентификация

Радиочастотная идентификация (RFID) — это технология автоматической бесконтактной идентификации объектов при помощи радиочастотного канала связи. Сейчас эта технология не имеет себе равных, и ее применение стало настолько популярным, что, по словам экспертов, RFID в скором времени вытеснит технологию штрихового кодирования. Базовая система RFID состоит из:

- радиочастотной метки;
- считывателя информации (ридера);
- компьютера для обработки информации.

Идентификация объектов производится по уникальному цифровому коду, который считывается из памяти электронной метки, прикрепляемой к объекту идентификации. Считыватель содержит в своем составе передатчик и антенну, посредством которых излучается электромагнитное поле определенной частоты. Попавшие в зону действия считывающего поля радиочастотные метки "отвечают" собственным сигналом, содержащим информацию (идентификационный номер товара, пользовательские данные и т. д.). Сигнал улавливается антенной считывателя, информация расшифровывается и передается в компьютер для обработки.

Особенности технологии RFID:

- отсутствие необходимости контакта и прямой видимости;
- возможность скрытой установки электронной метки;
- возможность чтения/записи информации;
- высокая скорость считывания данных;
- работа в сложных климатических условиях и вредных средах;
- неограниченный срок эксплуатации (для пассивных меток);
- невозможность подделки.

Существует несколько способов систематизации RFID-меток и систем:

- ❒ по рабочей частоте;
- ❒ по источнику питания;
- ❒ по типу памяти.

По типу источника питания RFID-метки делятся на:

- ❒ пассивные;
- ❒ активные;
- ❒ полупассивные.

Пассивные RFID-метки не имеют встроенного источника энергии. Электрический ток, индуцированный в антенне электромагнитным сигналом от считывателя, обеспечивает достаточную мощность для функционирования кремниевого КМОП-чипа, размещенного в метке, и передачи ответного сигнала.

Активные RFID-метки обладают собственным источником питания и не зависят от энергии считывателя, вследствие чего они читаются на дальнем расстоянии, имеют большие размеры и могут быть оснащены дополнительной электроникой. Однако такие метки наиболее дороги, а у батарей ограничено время работы. Активные метки в большинстве случаев более надежны и обеспечивают самую высокую точность считывания на максимальном расстоянии. Активные метки, обладая собственным источником питания, также могут генерировать выходной сигнал большего уровня, чем пассивные, позволяя применять их в более агрессивных для радиочастотного сигнала средах: воде (включая людей и животных, которые в основном состоят из воды), металлах (корабельные контейнеры, автомобили), для больших расстояний на воздухе. Большинство активных меток позволяет передать сигнал на расстояния в сотни метров при жизни батареи питания до 10 лет. Некоторые RFID-метки имеют встроенные сенсоры, например, для мониторинга температуры скоропортящихся товаров. Другие типы сенсоров в совокупности с активными метками могут применяться для измерения влажности, регистрации толчков/вибрации, света, радиации, температуры и газов в атмосфере. Активные метки обычно имеют гораздо больший радиус считывания (до 300 м) и объем памяти, чем пассивные, и способны хранить больший объем информации для отправки приемопередатчиком.

Полупассивные RFID-метки, также называемые полуактивными, очень похожи на пассивные метки, но оснащены батареей, которая обеспечивает чип энергопитанием. При этом дальность действия этих меток зависит только от чувствительности приемника считывателя и они могут функционировать на большем расстоянии и с лучшими характеристиками.

По типу используемой памяти RFID-метки делятся на:

- ❒ RO (Read Only) — данные записываются только один раз, сразу при изготовлении. Такие метки пригодны только для идентификации. Никакую новую информацию в них записать нельзя, и их практически невозможно подделать;
- ❒ WORM (Write Once Read Many) — кроме уникального идентификатора такие метки содержат блок однократно записываемой памяти, которую в дальнейшем можно многократно читать;

- RW (Read and Write) — такие метки содержат идентификатор и блок памяти для чтения/записи информации. Данные в них могут быть перезаписаны многократно.

По рабочей частоте RFID-метки делятся на:

- метки диапазона LF (125–134 кГц);
- метки диапазона HF (13,56 МГц);
- метки диапазона UHF (860–960 МГц);
- метки ближнего поля UHF Near-Field.

Пассивные системы диапазона LF (125–134 кГц) имеют низкую цену и в зависимости от физических характеристик используются для подкожных меток при чипировании животных, людей и рыб. Однако из-за длины волны у них существуют проблемы со считыванием на больших расстояниях, возможно также появление коллизий при считывании.

Метки диапазона HF (13,56 МГц) дешевы, не имеют экологических и лицензионных проблем, хорошо стандартизованы, имеют широкую линейку решений. Применяются в платежных системах, логистике, идентификации личности. Как и для диапазона LF, в системах, построенных в HF-диапазоне, существуют проблемы со считыванием на больших расстояниях, считыванием в условиях высокой влажности, при наличии на пути сигнала металла, возможно также появление коллизий при считывании.

Метки диапазона UHF (860–960 МГц) обладают наибольшей дальностью регистрации, во многих стандартах данного диапазона присутствуют антиколлизионные механизмы. Ориентированные изначально для нужд складской и производственной логистики, метки диапазона UHF не имели уникального идентификатора. Предполагалось, что идентификатором для метки будет служить EPC-номер (Electronic Product Code) товара, который каждый производитель будет заносить в метку самостоятельно при производстве. Однако скоро стало ясно, что помимо функции носителя EPC-номера товара хорошо бы возложить на метку еще и функцию контроля подлинности. То есть возникло требование, противоречащее самому себе, — одновременно обеспечить уникальность метки и позволить производителю записывать произвольный EPC-номер. На данный момент проблема решается так, что поле памяти TID (Tag ID), в которое при производстве обычно пишется код типа метки, разбито на две части. Первые 32 бита отведены под код производителя метки и ее марку, а вторые 32 бита — под уникальный номер самого чипа. Поле TID — неизменяемое, и, таким образом, каждая метка является уникальной. EPC-номер может быть записан производителем товара в момент маркировки. В UHF RFID-системах по сравнению с LF и HF ниже стоимость меток, при этом выше стоимость прочего оборудования. В настоящее время частотный диапазон УВЧ открыт для свободного использования в Российской Федерации в так называемом "европейском" диапазоне — 863–868 МГЦ.

Радиочастотные UHF-метки ближнего поля, не являясь непосредственно радиометками, поскольку используют магнитное поле антенны, позволяют решить проблему

считывания в условиях высокой влажности, присутствия воды и металла. С помощью данной технологии ожидается начало массового применения RFID-меток в розничной торговле фармацевтическими товарами (нуждающимися в контроле подлинности, учете, но при этом зачастую содержащими воду и металлические детали в упаковке).

Приборы, которые читают информацию с меток и записывают в них данные, называются ридерами (считывателями). Эти устройства могут быть постоянно подключенными к учетной системе или работать автономно.

18.2. Датчик считывания RFID-карт

Для создания считывателя RFID-меток на платформе Arduino применим датчик считывания RFID-карт компании Seeed Technology Inc., работающий на частоте 125 кГц (рис. 18.1). Датчик имеет высокую чувствительность расстояния срабатывания — 7 см. Датчик выдает информацию о карте в двух форматах данных: Uart и Wiegand. Wiegand — простой проводной интерфейс связи между устройством чтения идентификатора (карточки) и контроллером, широко применяемый в системах контроля доступа. Предназначен для передачи уникального кода карты в контроллер. Для переключения в формат Wiegand необходимо установить перемычку **JUMPER** на правый и средний контакт (рис. 18.2).

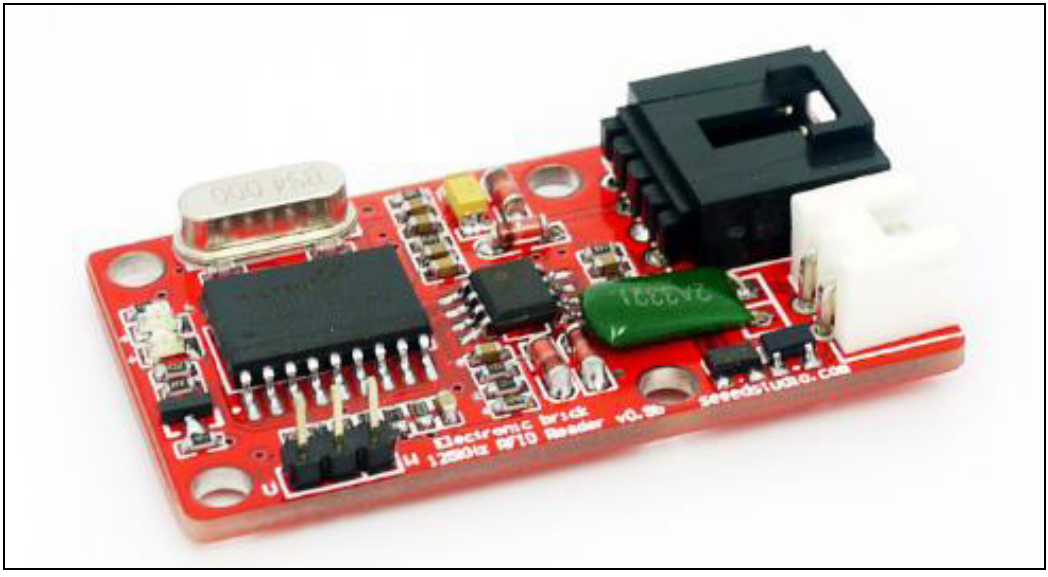


Рис. 18.1. Датчик считывания RFID-карт 125 кГц

Для подключения к плате Arduino подсоединяем выход **TX** к контакту 3 платы Arduino, выход **RX** к контакту 2 платы Arduino. В результате получается конструкция, приведенная на рис. 18.3. Для создания проекта были приобретены пассивные брелоки и карты RFID. Коды меток указаны на брелоках и картах.

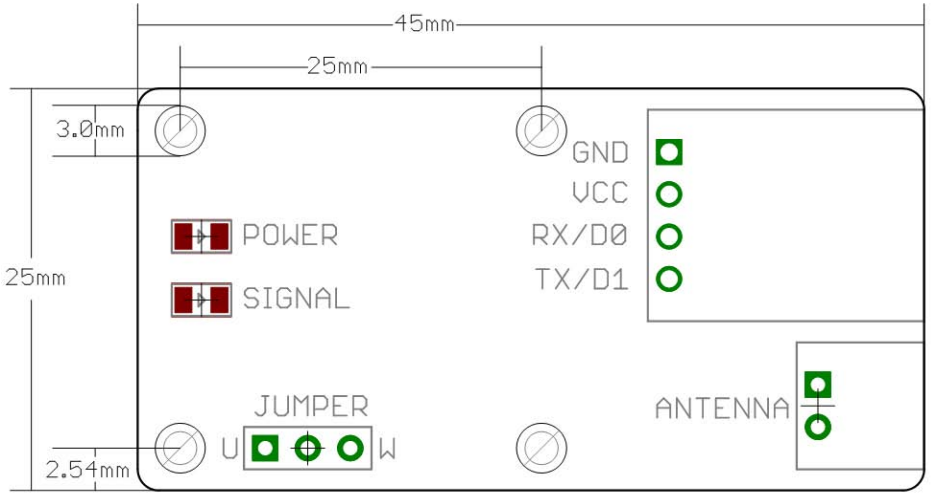


Рис. 18.2. Схема расположения контактов датчика считывания RFID-карт 125 кГц

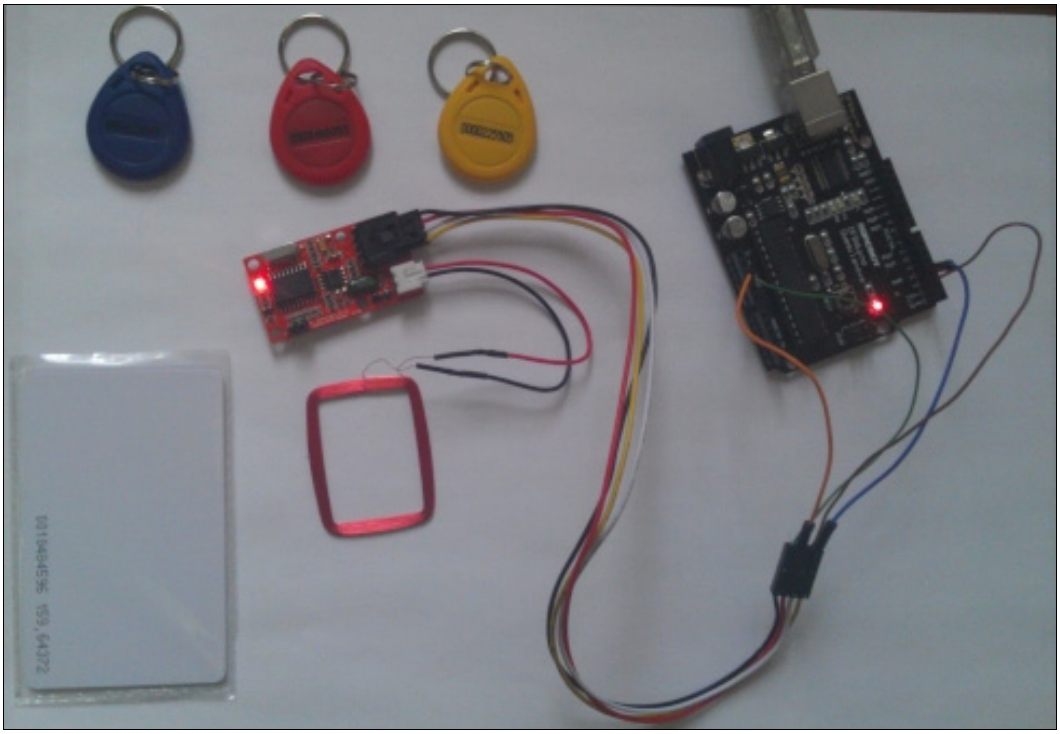


Рис. 18.3. Устройство для подключения датчика считывания RFID-карт 125 кГц к Arduino

18.3. Скетч для считывания RFID-карт

Скетч получает код от считывателя по протоколу Wiegand и выводит метку в монитор последовательного порта (рис. 18.4). Сравниваем с кодом, указанным на брелоках и картах. Код данного скетча приведен в листинге 18.1.

Листинг 18.1

```
byte RFIDcardNum[4];
byte evenBit = 0;
byte oddBit = 0;
byte isData0Low = 0;
byte isData1Low = 0;
int recvBitCount = 0;
byte isCardReadOver = 0;

void setup()
{
 Serial.begin(9600);
 attachInterrupt(0, ISRreceiveData0, FALLING );
 attachInterrupt(1, ISRreceiveData1, FALLING );
}
void loop()
{
 // чтение бита номера карты
 if(isData0Low||isData1Low){
  if(1 == recvBitCount){//even bit
   evenBit = (1-isData0Low)&isData1Low;
  }
  else if( recvBitCount >= 26){//odd bit
   oddBit = (1-isData0Low)&isData1Low;
   isCardReadOver = 1;
   delay(10);
  }
  else{
   //only if isData1Low = 1, card bit could be 1
   RFIDcardNum[2-(recvBitCount-2)/8] |= (isData1Low << (7-(recvBitCount-2)%8));
  }
  //reset data0 and data1
  isData0Low = 0;
  isData1Low = 0;
 }
 // печать метки карты
 if(isCardReadOver){
  if(checkParity()){
   Serial.println(*((long *)RFIDcardNum));
  }
```

```
  resetData();
 }
}
byte checkParity(){
 int i = 0;
 int evenCount = 0;
 int oddCount = 0;
 for(i = 0; i < 8; i++){
  if(RFIDcardNum[2]&(0x80>>i)){
   evenCount++;
  }
 }
 for(i = 0; i < 4; i++){
  if(RFIDcardNum[1]&(0x80>>i)){
   evenCount++;
  }
 }
 for(i = 4; i < 8; i++){
  if(RFIDcardNum[1]&(0x80>>i)){
   oddCount++;
  }
 }
 for(i = 0; i < 8; i++){
  if(RFIDcardNum[0]&(0x80>>i)){
   oddCount++;
  }
 }
 if(evenCount%2 == evenBit && oddCount%2 != oddBit){
  return 1;
 }
 else{
  return 0;
 }
}
void resetData(){
 RFIDcardNum[0] = 0;
 RFIDcardNum[1] = 0;
 RFIDcardNum[2] = 0;
 RFIDcardNum[3] = 0;
 evenBit = 0;
 oddBit = 0;
 recvBitCount = 0;
 isData0Low = 0;
 isData1Low = 0;
 isCardReadOver = 0;
}
```

```
// функция interrupt0
void ISRreceiveData0(){
 recvBitCount++;
 isData0Low = 1;
}
// функция interrupt1
void ISRreceiveData1(){
 recvBitCount++;
 isData1Low = 1;
}
```

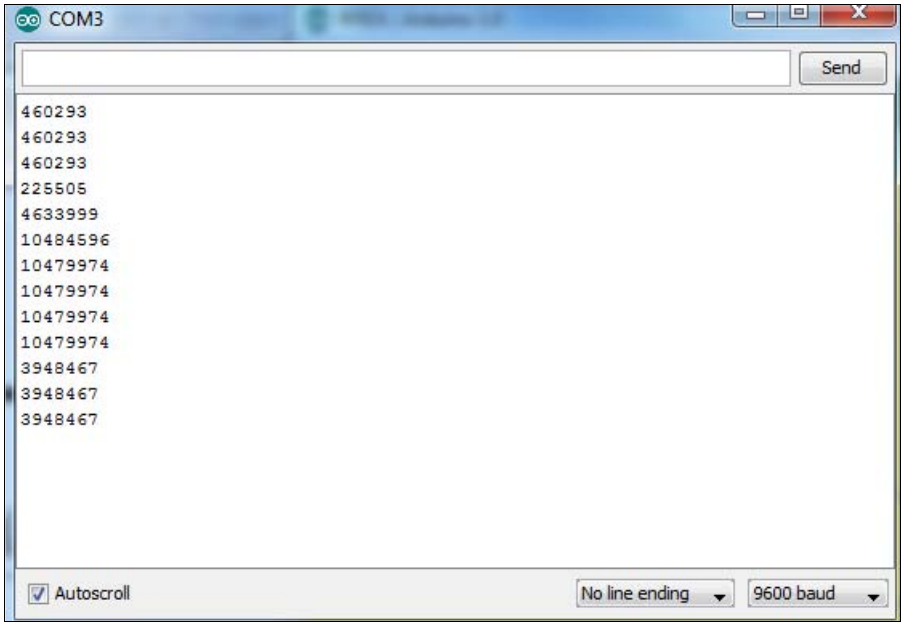


Рис. 18.4. Выдача кодов в монитор последовательного порта

Данный скетч можете найти в папке examples/_18_1 сопровождающего книгу электронного архива.

ГЛАВА 19

Arduino и датчики расстояния

Рассмотрим работу Arduino с датчиками расстояния, которые являются неотъемлемой частью любого робота. Обратим внимание на ультразвуковые датчики HC-SR04 и инфракрасные датчики расстояния Sharp.

19.1. Ультразвуковые дальномеры HC-SR04

Познакомимся с датчиками расстояния, которые пригодятся в проектах, рассматриваемых в следующих главах. Ультразвуковой дальномер HC-SR04 (рис. 19.1) — это помещенные на одну плату приемник и передатчик ультразвукового сигнала. Кроме самих приемника и передатчика на плате находится еще и необходимая обвязка, чтобы сделать работу с этим датчиком простой и непринужденной.

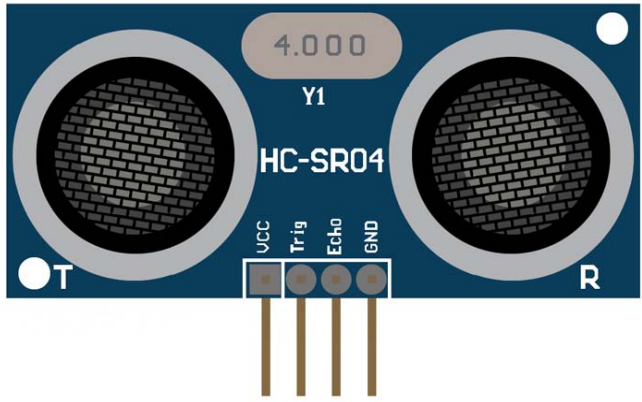


Рис. 19.1. Датчик HC-SR04

Датчик обладает низким энергопотреблением, что также является немаловажным преимуществом в случае с мобильными роботами, не привязанными к розетке. Питается датчик HC-SR04 от 5 В, что тоже удобно при подключении его к Arduino.

Характеристики ультразвукового дальномера HC-SR04:

- измеряемый диапазон — от 2-х до 500 см;
- точность — 0,3 см;
- угол обзора — < 15 °;
- напряжение питания — 5 В.

Датчик имеет 4 вывода стандарта 2,54 мм:

- VCC — питание +5 В;
- Trig (T) — вывод входного сигнала;
- Echo (R) — вывод выходного сигнала;
- GND — земля.

19.2. Принцип работы ультразвукового дальномера HC-SR04

В составе дальномера имеются два пьезоэлемента: один работает как излучатель сигнала, другой — как приемник. Излучатель генерирует сигнал, который, отразившись от препятствия, попадает на приемник. Измерив время, за которое сигнал проходит до объекта и обратно, можно оценить расстояние.

Последовательность действий следующая:

1. Подаем импульс продолжительностью 10 мкс на вывод Trig.
2. Внутри дальномера входной импульс преобразуется в 8 импульсов частотой 40 кГц и посылается вперед через излучатель T (см. рис. 15.1).
3. Дойдя до препятствия, посланные импульсы отражаются и принимаются приемником R (см. рис. 15.1), в результате получаем выходной сигнал на выводе Echo.
4. Непосредственно на стороне контроллера переводим полученный сигнал в расстояние по формуле:
   - ширина импульса (мкс) / 58 = дистанция (см);
   - ширина импульса (мкс) / 148 = дистанция (дюйм).

19.3. Библиотека *Ultrasonic*

Для работы Arduino с датчиком HC-SR04 имеется готовая библиотека — `Ultrasonic`. Конструктор `Ultrasonic` принимает два параметра: номера пинов, к которым подключены выводы Trig и Echo соответственно:

```
Ultrasonic ultrasonic(12,13);
```

здесь вывод датчика Trig подключен к 12-му пину Arduino, а Echo — к 13-му.

Библиотека имеет один метод `Ranging`, в качестве параметра которому задается, во что пересчитывать расстояние до объекта: в сантиметры или в дюймы:

```
#define CM 1
#define INC 0
```

Таким образом строчка `ultrasonic.Randing(CM)` вернет расстояние до объекта (типа `long`) в сантиметрах.

Файлы библиотеки вы можете найти в папке libraries/Ultrasonic сопровождающего книгу электронного архива. Для использования библиотеки в своих проектах поместим ее в папку libraries каталога установки Arduino.

Скетч, выдающий в последовательный порт расстояние до объекта в сантиметрах, представлен в листинге 19.1.

Листинг 19.1

```
#include "Ultrasonic.h"

// sensor connected to:
// Trig - 12, Echo - 13
Ultrasonic ultrasonic(12, 13);

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

void loop()
{
 float dist_cm = ultrasonic.Ranging(CM);
 Serial.println(dist_cm);

 delay(100);
}
```

Рассмотренный здесь датчик мы будем использовать в проектах, представленных в следующих главах.

19.4. Инфракрасные датчики расстояния Sharp

Для измерения расстояния до объекта существуют также и оптические датчики, основанные на методе триангуляции. Самые распространенные из них — это инфракрасные (Infra-Red, IR) датчики расстояния с выходным аналоговым напряжением, производимые фирмой Sharp (рис. 19.2).

В датчиках Sharp установлен инфракрасный (IR) светодиод (LED) с линзой, который излучает узкий световой луч. Отраженный от объекта луч направляется через

другую линзу на позиционно-чувствительный фотоэлемент (Position-Sensitive Detector, PSD). От местоположения падающего на PSD луча зависит его проводимость. Проводимость преобразуется в напряжение и, к примеру, оцифровывая его аналого-цифровым преобразователем микроконтроллера, можно вычислить расстояние. Рис. 19.3 показывает путь отраженного луча на различных расстояниях.

Рис. 19.2. Инфракрасные датчики расстояния Sharp

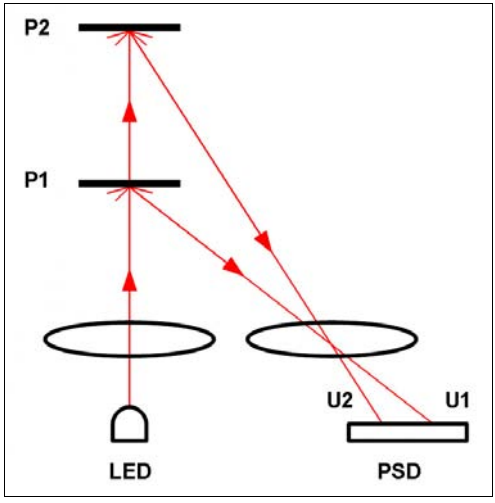


Рис. 19.3. Путь светового луча инфракрасного измерителя расстояния

Выход датчика расстояния Sharp обратно пропорциональный — с увеличением расстояния его значение медленно уменьшается. Вид графика зависимости между расстоянием и напряжением приведен на рис. 19.4. Датчики, в зависимости от их типа, имеют границы измерения, в пределах которых их выход может быть признан надежным. Измерение максимального реального расстояния ограничивают два фактора: уменьшение интенсивности отраженного света и невозможность PSD регистрировать незначительные изменения местоположения отображенного луча.

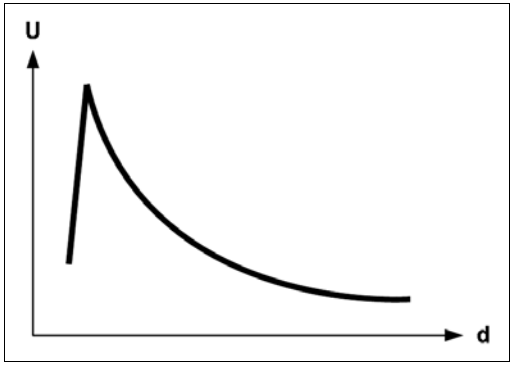


Рис. 19.4. График зависимости между расстоянием и напряжением датчиков Sharp

При измерении расстояния до сильно удаленных объектов выход датчика остается приблизительно таким же, как и при измерении минимально удаленных расстояний. Минимально измеряемое расстояние ограничено особенностями датчика Sharp, а именно — выходное напряжение при уменьшении расстояния (в зависимости от датчика — от 4-х до 20 см) начинает резко падать. По существу это означает, что одному значению выходного напряжения соответствуют два расстояния: очень близкое и очень далекое. Для предотвращения проблемы следует избегать слишком близкого приближения объектов к датчику.

В целом график зависимости между расстоянием и напряжением не является линейным, однако в пределах допустимых расстояний график обратной величины выходного напряжения и расстояния к линейности приближается достаточно близко, и с его помощью довольно просто получить формулу для преобразования напряжения в расстояние. Для нахождения такой формулы необходимо точки этого графика ввести в какую-либо программу обработки табличных данных и из них создать новый график. В программе обработки табличных данных на основе точек графика возможно автоматически вычислить линию тренда. На рис. 19.5 приведен график связи исправленной обратной величины между выходным напряжением инфракрасного датчика GP2Y0A21YK и расстоянием вместе с линейной линией тренда. Выходное напряжение для упрощения формулы уже переведено в 10-битное значение аналогово-цифрового преобразователя с опорным напряжением +5 В.

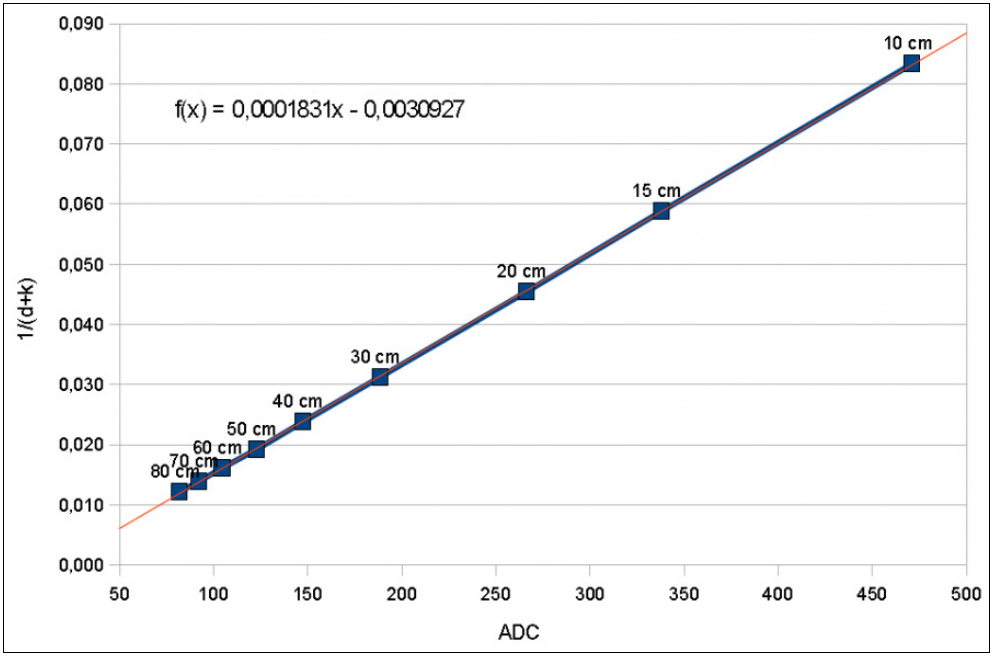


Рис. 19.5. График зависимости между выходным напряжением датчика GP2Y0A21YK и расстоянием

19.5. Подключение датчиков Sharp к Arduino

Работать с сенсорами Sharp очень просто — достаточно подключить к нему питание и завести вывод Vo на аналоговый вход Arduino. Значение получаемой функции `analogRead` представляет собой целое число от 0 до 1023. Таким образом, чтобы узнать напряжение на выходе сенсора, необходимо значение на аналоговом входе Arduino умножить на 0,0048828125 (5 В / 1024). Содержимое скетча, выдающего расстояние, измеряемое датчиком Sharp, в последовательный порт, представлено в листинге 19.2.

Листинг 19.2

```
//
// SHARP IR sensor
//
int IRpin = 0;  // аналоговый пин для подключения выхода Vo сенсора

void setup() {
 Serial.begin(9600);   // старт последовательного порта
}

void loop() {
 // 5V/1024 = 0.0048828125
 float volts = analogRead(IRpin)*0.0048828125;
   // считываем значение сенсора и переводим в напряжение
 Serial.println(volts);      // выдаем в порт
 delay(100);
}
```

ГЛАВА 20

Arduino и передача данных в инфракрасном диапазоне

Устройства инфракрасного (ИК) диапазона волн часто применяются в робототехнике. На ИК-трансиверах (приемопередатчиках) можно организовать полноценный обмен данными. Самое простое применение — использование ИК-пульта (думаю, найдется в каждом доме) для управления роботом.

20.1. Обмен данными в инфракрасном диапазоне

Для обеспечения надежного приема и гарантированной защиты от помех используется модуляция сигнала и кодирование. Передача данных производится в близком к видимому инфракрасном спектре. Длина волны в большинстве реализованных систем варьируется в пределах 800–950 нм. Самый простой способ избавиться от фонового шума — модулировать (заполнить) сигнал при передаче одной из стандартных частот: 30, 33, 36, 37, 38, 40, 56 кГц. Именно на эти частоты настроены все современные интегральные приемники.

Для обеспечения достаточной дальности при передаче кодовой последовательности необходимо сформировать мощный сигнал. Ток через ИК-светодиод может достигать 1 А — такие токи вполне допустимы в импульсном режиме, при этом средняя рассеиваемая мощность не должна превышать предельно допустимую, указанную в документации.

Разработано большое количество специализированных микросхем (SAA3010, GS8489, KS51840 и т. п), генерирующих готовую кодовую последовательность и потребляющих минимальный ток в ждущем режиме, что немаловажно при питании от батарей. Эти микросхемы существенно упрощают схему пультов дистанционного управления (ПДУ). Когда мы нажимаем кнопку пульта, микросхема передатчика активизируется и генерирует кодовую последовательность с заданным заполнением. Светодиод преобразуют эти сигналы в ИК-излучение. Излученный сигнал принимается фотодиодом, который снова преобразует ИК-излучение в электрические импульсы. Эти импульсы усиливаются и демодулируются микросхемой приемника. Затем они подаются на декодер. Декодирование обычно осуществляется программно с помощью микроконтроллера.

Приемник ИК ПДУ должен восстанавливать данные с двухфазным кодированием и реагировать на большие быстрые изменения уровня сигнала независимо от помех. Ширина импульсов на выходе приемника должна отличаться от номинальной не более чем на 10 %. Приемник также должен быть нечувствительным к постоянным внешним засветкам. Удовлетворить всем этим требованиям достаточно непросто. Старые реализации приемника ИК ДУ, даже с применением специализированных микросхем, содержали десятки компонентов. Такие приемники часто использовали резонансные контуры, настроенные на частоту заполнения. Все это делало конструкцию сложной в изготовлении и настройке, требовало применения хорошего экранирования.

В последнее время большое распространение получили трехвыводные интегральные приемники ИК ПДУ (SFH5110-xx, TSOP17xx, TFMS5xx0 и т. п.). В одном корпусе они объединяют фотодиод, предусилитель и формирователь. На выходе формируется обычный ТТЛ-сигнал без заполнения, пригодный для дальнейшей обработки микроконтроллером. Наиболее важный параметр при выборе приемника — частота заполнения.

Внутренний усилитель интегрального приемника имеет высокий коэффициент усиления, поэтому для исключения самовозбуждения и устранения влияния наводок по цепям питания необходимо использовать электролитический конденсатор емкостью не менее 4,7 мкФ, подключенный максимально близко к выводу VCC.

В ПДУ используется три вида модуляции:

- ❒ двухфазное кодирование (Bi-phase coding). Фронт импульса — логическая "1", спад импульса — логический "0" (рис. 20.1);
- ❒ модуляция длительностью пауз (Pulse Distance Modulation). Длина импульсов постоянна, паузы большей длительности — логическая "1", меньшей — логический "0" (рис. 20.2);
- ❒ модуляция длительностью импульса (Pulse Width Modulation). Длина пауз постоянна, импульсы большей длительности — логическая "1" , меньшей — логический "0" (рис. 20.3).

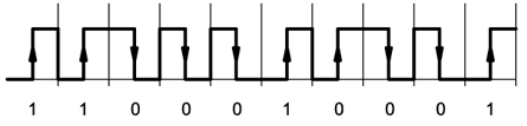


Рис. 20.1. Двухфазное кодирование

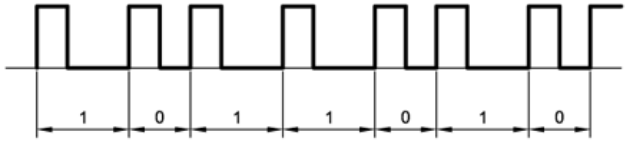


Рис. 20.2. Модуляция длительностью пауз

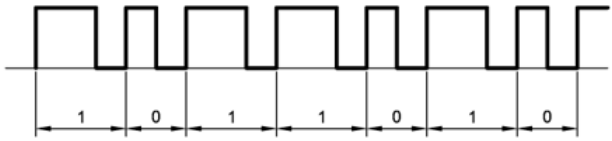


Рис. 20.3. Модуляция длительностью импульса

К сожалению, нет единого и универсального протокола для ИК-пультов дистанционного управления, хотя среди всего многообразия есть наиболее распространенные.

20.2. Протоколы для ИК-пультов

Наиболее распространенными протоколами для ИК-пультов дистанционного управления являются следующие:

- RC5;
- NEC;
- JVC;
- Sony.

20.2.1. Протокол RC5

Один из самых старых и распространенных протоколов. В свое время RC5, разработанный фирмой Philips для управления бытовой аппаратурой, получил широкое употребление. Сейчас он применяется реже, и в основном любителями, из-за своей простоты и широкой доступности недорогих компонентов. Позднее фирма Philips внедрила и стала использовать улучшенный протокол RC6. Основные характеристики протокола RC5:

- 5-битный адрес, 6-битные команды;
- модуляция Bi-phase coding;
- сначала идут старшие биты, потом младшие (MSB first);
- частота заполнения 36 кГц.

В основном этот стандарт используется в изделиях фирмы Samsung.

Формат посылки RC5, кодирование логического "0" и логической "1" приведены на рис. 20.4 и 20.5.

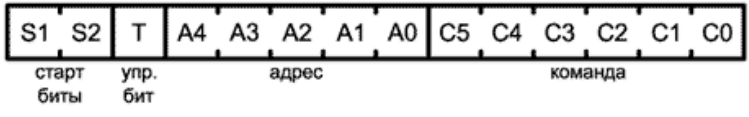


Рис. 20.4. Формат посылки RC5

Два первых бита на рис. 20.4 — это стартовые биты (всегда логическая "1"). Управляющий бит T изменяется только при новом нажатии на кнопку. При удержании кнопки посылка передается с интервалом 64 такта, что составляет 113,778 мс (рис. 20.6).

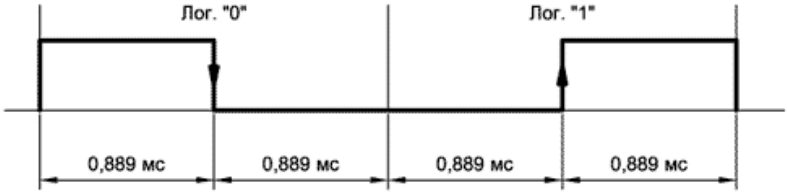


Рис. 20.5. Кодирование логического "0" и логической "1" протокола RC5

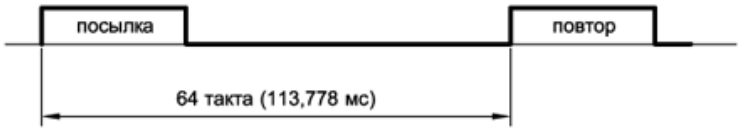


Рис. 20.6. Повторная передача

20.2.2. Протокол NEC

Очень распространенный, простой и универсальный протокол. Его используют многие корейские и японские производители бытовой техники — такие как Samsung, LG, NEC, Sanyo, Panasonic, Hitachi, Nokia, AverMedia. Сейчас сложно установить, кому из них принадлежит его разработка, но в Интернете он чаще всего упоминается как протокол NEC. Основные характеристики протокола:

- 8-битные адрес и команды;
- адрес и команды дублируются с инверсией;
- модуляция Pulse Distance Modulation;
- сначала идут младшие биты, потом старшие (LSB first);
- частота заполнения 38 кГц.

Формат посылки NEC, стартовая последовательность, кодирование логического "0" и логической "1" приведены на рис. 20.7, 20.8 и 20.9.

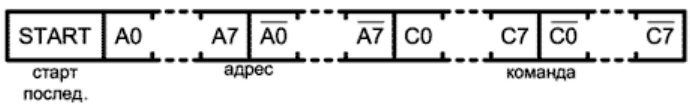


Рис. 20.7. Формат посылки NEC

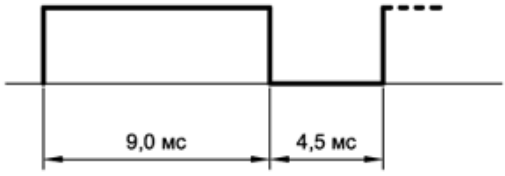


Рис. 20.8. Стартовая последовательность

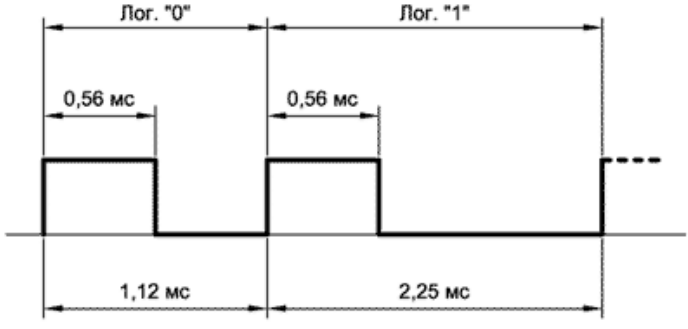


Рис. 20.9. Кодирование логического "0" и логической "1"

Основная посылка передается только один раз при нажатии на кнопку. При удержании кнопки передается последовательность повтора через каждые 110 мс (рис. 20.10 и 20.11).

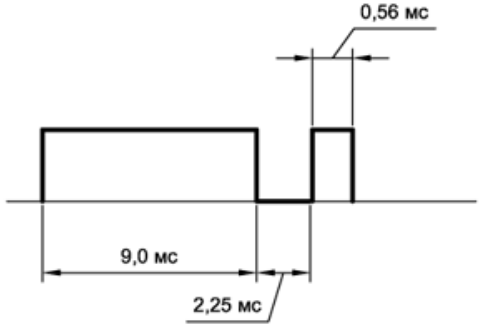


Рис. 20.10. Последовательность повтора

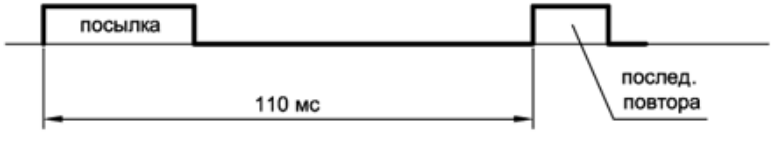


Рис. 20.11. Повторная передача

20.2.3. Протокол JVC

Протокол очень похож на NEC. Отличия заключаются только во временных интервалах, отсутствии дублирования адреса команд с инверсией и способе передачи состояния удержания кнопки. Основные характеристики протокола:

- 8-битные адрес и команды;
- модуляция Pulse Distance Modulation;
- сначала идут младшие биты, потом старшие (LSB first);
- частота заполнения 38 кГц.

Формат посылки JVC, стартовая последовательность, кодирование логического "0" и логической "1" приведены на рис. 20.12, 20.13 и 20.14.

Основная посылка передается только один раз при нажатии на кнопку. При удержании кнопки передается только команда через каждые 50–60 мс (рис. 20.15).

Рис. 20.12. Формат посылки JVC

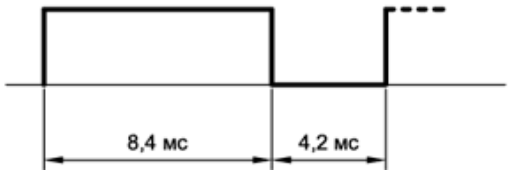


Рис. 20.13. Стартовая последовательность

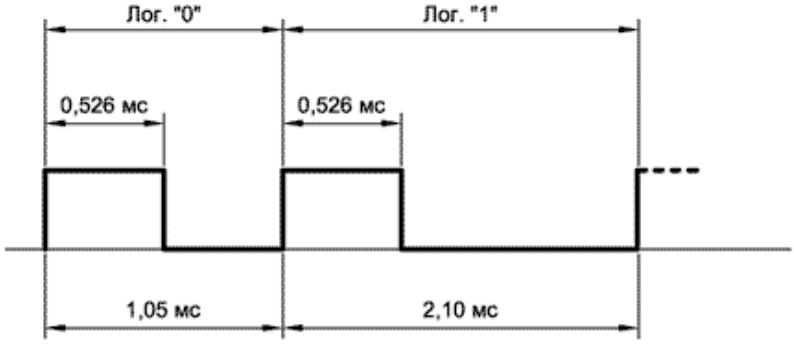


Рис. 20.14. Кодирование логического "0" и логической "1"

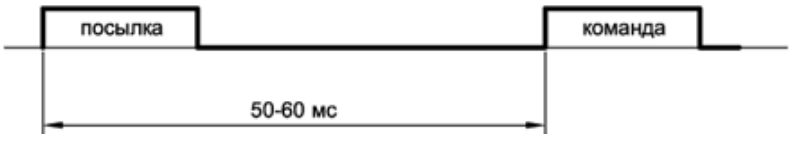


Рис. 20.15. Повторная передача

20.2.4. Протокол Sony

Еще один распространенный протокол. Основные его характеристики:

- 12-, 15- и 20-битные варианты протокола;
- модуляция Pulse Width Modulation;
- сначала идут младшие биты, потом старшие (LSB first);
- частота заполнения 40 кГц.

В 12-битном варианте: 7 битов команды и 5 битов адреса устройства. В 15-битном варианте — 8 и 7 битов соответственно. После окончания передачи удерживается состояние логического "0" до достижения интервала 45 мс с начала передачи. Формат посылки Sony, стартовая последовательность, кодирование логического "0" и логической "1" приведены на рис. 20.16, 20.17 и 20.18.

При удержании кнопки передается только команда через каждые 45 мс (рис. 20.19).

Рис. 20.16. Формат посылки Sony

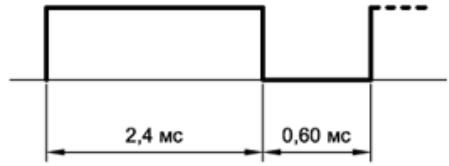


Рис. 20.17. Стартовая последовательность

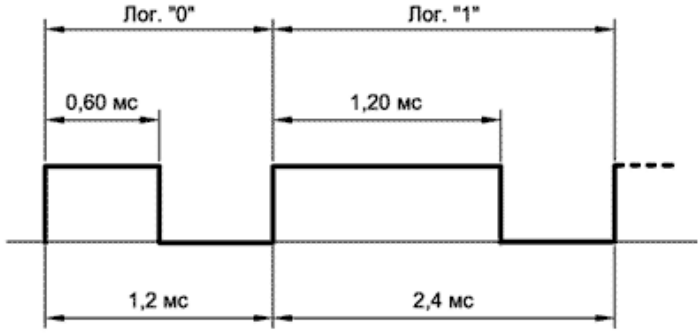


Рис. 20.18. Кодирование логического "0" и логической "1"

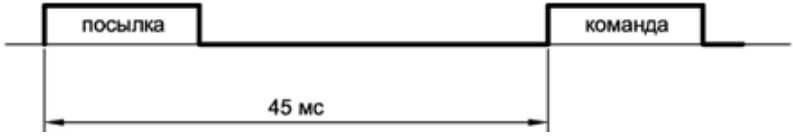


Рис. 20.19. Повторная передача

20.3. Подключение ИК-приемника

В качестве приемника ИК ПДУ применим микросхему TSOP31236. В одном корпусе она объединяют фотодиод, предусилитель и формирователь. На выходе формируется обычный ТТЛ-сигнал без заполнения, пригодный для дальнейшей обработки микроконтроллером. Несущая частота 36 кГц, выход инверсный, т. е. при отсутствии сигнала на пин приходит логическая "1", при появлении сигнала он посылает логический "0". Внешний вид микросхемы представлен на рис. 20.20.

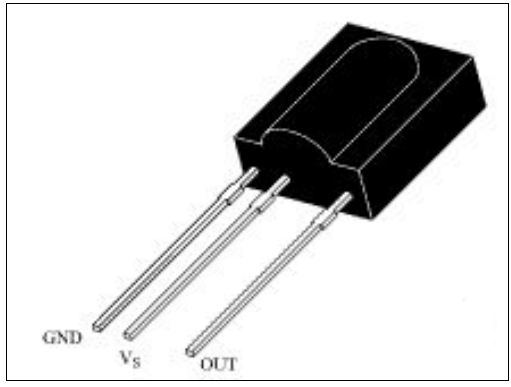


Рис. 20.20. Микросхема TSOP31236

Подключим приемник по схеме, представленной на рис. 20.21. Для исключения ложных срабатываний используем RC-фильтр.

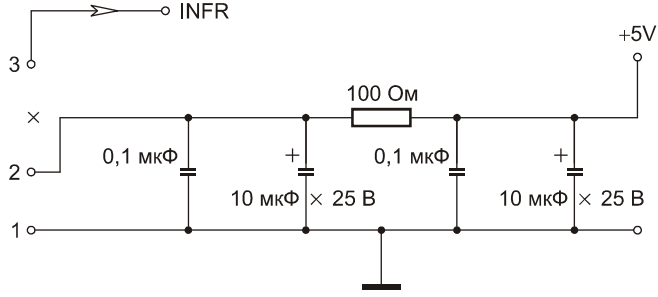


Рис. 20.21. Схема подключения

20.4. Библиотека *IRremote*

Можно узнать протокол вашего пульта и написать скетч для получения кодов, отправляемых с пульта. К счастью, уже написана универсальная библиотека для приема и обработки кодов с любого пульта — `IRremote`.

Файлы библиотеки вы можете найти в папке libraries/IrRemote сопровождающего книгу электронного архива. Для использования библиотеки в своих проектах поместим их в папку libraries каталога установки Arduino. Скетч для получения кода и отправки в последовательный порт представлен в листинге 20.1.

Листинг 20.1

```
#include <IRremote.h>

int RECV_PIN = 11;
IRrecv irrecv(RECV_PIN);
decode_results results;

void setup()
 {
 Serial.begin(9600);
 irrecv.enableIRIn(); // включить приемник
 }
void loop()
 {
 if (irrecv.decode(&results))
  {
  Serial.println(results.value, HEX);
  irrecv.resume(); // получить следующее значение
  }
 }
```

Можно и передавать ИК-команды. Поддерживаемые протоколы: NEC, Sony SIRC, Philips RC5, Philips RC6. Передающий ИК-светодиод должен быть подключен к pin 3 (рис. 20.22). Скетч для отправки ИК-кода представлен в листинге 20.2.

Листинг 20.2

```
#include <IRremote.h>
IRsend irsend;

void setup()
 {
 Serial.begin(9600);
 }
void loop()
 {
 if (Serial.read() != -1)
 {
 for (int i = 0; i < 3; i++)
  {
  irsend.sendSony(0xa90, 12); // Sony TV power code
  delay(100);
  }
 }
 }
```

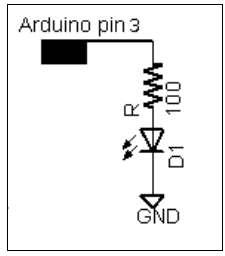


Рис. 20.22. Схема для отправки ИК-кода

20.5. Скетч для получения кодов ИК-пульта

В дальнейшем мы будем использовать пульт для управления роботом (см. *главу 21*). Поэтому первая задача — получить список кодов клавиш нашего пульта. Рассмотрим пульт Marmitek (ИК и радио 433 МГц), купленный в свое время для управления приборами X10 из серии "умный дом". Внешний вид пульта приведен на рис. 20.23.

Рис. 20.23. Пульт Marmitek

Определим список клавиш пульта для управления движением робота:

- <↑> — движение вперед;
- <↓> — движение назад;
- <←> — поворот влево;
- <→> — поворот вправо;
- <–CH> — увеличение скорости при движении вперед/назад;
- <CH+> — уменьшение скорости при движении вперед/назад;
- <–VOL> — круговое движение на месте влево;
- <VOL+> — круговое движение на месте вправо;
- <0> — остановка робота.

Запускаем скетч из листинга 20.3 и получаем коды нужных клавиш для вашего пульта. Значения кодов выводятся в последовательный порт.

Листинг 20.3

```
#include <IRremote.h>

// вход ИК-приемника
int RECV_PIN = 2;
IRrecv irrecv(RECV_PIN);
decode_results results;
unsigned long ir_dt, old_ir;
long ir_kod;
unsigned long ir_time1, ir_time2;

void setup()
 {
 // последовательный порт
 Serial.begin(9600);
 // включить приемник
 irrecv.enableIRIn();
 ir_time1=0;ir_time2=0;
 // прерывания для ИК
 // FALLING - вызов прерывания при изменении уровня напряжения
 // с высокого (HIGH) на низкое (LOW)
 attachInterrupt(0, get_ir_kod, FALLING);
 }
void loop()
 {
 // обработка кода нажатия
 if(ir_kod>0)
 {
 ir_go(ir_kod);
 Serial.println(ir_kod);
 ir_kod=0;
 }
}
// получить код, переданный с ИК-пульта
void get_ir_kod()
 {
 detachInterrupt(0);  // отключить прерывание 0
 if (irrecv.decode(&results))
 {
 if (results.value > 0 && results.value < 0xFFFFFFFF)
  {
  ir_dt = results.value;
  ir_time2=millis();
  // прошла 1 сек?
  if (ir_time2-ir_time1>1000)
   {ir_kod = ir_dt;ir_time1=ir_time2;}
```

```
  else
   ir_kod = 0;
  }
  irrecv.resume();
 }
 // активировать процедуру прерывания 0
 attachInterrupt(0, get_ir_kod, FALLING);
 }
```

Оформим их в виде констант (листинг 20.4).

Листинг 20.4

```
// коды клавиш ИК-пульта (marmitek)
#define FORWARD 1936    // ↑
#define BACK 3984     // ↓
#define SPEED_UP 144    //ch+
#define SPEED_DOWN 2192  //ch-
#define LEFT 3472     // ←
#define RIGHT 1424    // →
#define CIRCLE_LEFT 3216 //vol+
#define CIRCLE_RIGHT 1168 //vol-
#define STOP 2320     // 0 - стоп
```

Определять поступление команды с пульта мы будем по прерыванию 0 (на digital pin2). По прерыванию запускается процедура `get_ir_kod()`, которая определяет код, поступающий с пульта, и записывает его в переменную `ir_kod`. Процедура `loop()` проверяет переменную `ir_kod`, и в случае ненулевого значения переменной (получения кода с пульта) вызывает процедуру вывода действия `ir_go()`. На данном этапе — это вывод в последовательный порт предполагаемого по нажатию клавиши действия (рис. 20.24). Данный скетч представлен в листинге 20.5.

Рис. 20.24. Вывод на монитор последовательного порта результата команды с пульта

Листинг 20.5

```
#include <IRremote.h>

// вход ИК-приемника
int RECV_PIN = 2;
IRrecv irrecv(RECV_PIN);
decode_results results;
unsigned long ir_dt, old_ir;
long ir_kod;
unsigned long ir_time1, ir_time2;
// коды клавиш ИК-пульта (marmitek)
#define FORWARD 1936
#define BACK 3984
#define SPEED_UP 144   //ch+
#define SPEED_DOWN 2192  //ch-
#define LEFT 3472
#define RIGHT 1424
#define CIRCLE_LEFT 3216 //vol+
#define CIRCLE_RIGHT 1168 //vol-
#define STOP 2320     //0

void setup()
 {
 // последовательный порт
 Serial.begin(9600);
 // включить приемник
 irrecv.enableIRIn();
 ir_time1=0;ir_time2=0;
 // прерывания для ИК
 // FALLING - вызов прерывания при изменении уровня напряжения
 // с высокого (HIGH) на низкое (LOW)
 attachInterrupt(0, get_ir_kod, FALLING);
 }

void loop()
 {
 // обработка кода нажатия
 if(ir_kod>0)
 {
 ir_go(ir_kod);
 ir_kod=0;
 }
}
// получить код переданный с ИК-пульта
void get_ir_kod()
 {
 detachInterrupt(0);  // отключить прерывание 0
```

```
 if (irrecv.decode(&results))
 {
 if (results.value > 0 && results.value < 0xFFFFFFFF)
  {
  ir_dt = results.value;
  ir_time2=millis();
  // прошла 1 сек?
  if (ir_time2-ir_time1>1000)
   {ir_kod = ir_dt;ir_time1=ir_time2;}
  else
   ir_kod = 0;
  }
  irrecv.resume();
 }
 // активировать процедуру прерывания 0
 attachInterrupt(0, get_ir_kod, FALLING);
 }
// действие по полученному коду
void ir_go(kod)
 {
 switch(kod)
  {
  case FORWARD : // направление вперед
    Serial.print("forward\n");
    break;
  case BACK :  // направление назад
    Serial.print("back\n");
    break;
   case SPEED_UP : // скорость++
    Serial.print("speed++\n");
    break;
   case SPEED_DOWN : // скорость--
    Serial.print("speed--\n");
    break;
   case LEFT : // влево
    Serial.print("left\n");
    break;
  case RIGHT : // вправо
    Serial.print("right\n");
   break;
  case CIRCLE_RIGHT : // кружение вправо
    Serial.print("circle_right\n");
    break;
  case CIRCLE_LEFT : // кружение влево
    Serial.print("circle_left\n");
    break;
```

```
  case STOP : // стоп
    Serial.print("stop\n");
    break;
  default:
    break;
  }
}
```

Данный скетч можно найти в папке examples/_20_1 сопровождающего книгу электронного архива.

ГЛАВА 21

Создаем робота

В продолжение *главы 20* здесь мы приступим к созданию робота. Наш робот представляет собой движущуюся гусеничную платформу с внешним управлением через ИК-пульт. В автономном режиме робот будет двигаться по черной линии.

21.1. Ходовая часть

Для сборки ходовой части робота использовались:

- ❒ набор гусениц Tamiya;
- ❒ перфорированная пластиковая панель с крепежом Tamiya 70098 Universal Plate Set;
- ❒ двойной редуктор Tamiya.

Комплектация набора гусениц Tamiya (рис. 21.1) следующая:

- ❒ 2 гусеницы по 30 траков;
- ❒ 2 гусеницы по 10 траков;
- ❒ 2 гусеницы по 8 траков;
- ❒ 2 больших ведущих колеса;
- ❒ 2 больших гладких колеса;
- ❒ 6 роликов;
- ❒ 1 вал 3×100 мм шестигранный;
- ❒ 5 валов 3×100 мм.

Набор Tamiya 70098 Universal Plate Set (рис. 21.2) состоит из пластины со сквозными монтажными отверстиями, угловых скоб и крепежа, которые позволят смонтировать механические компоненты во множество позиций.

Двухмоторный редуктор Tamiya содержит два маленьких электродвигателя постоянного тока, которые вращают независимые 3-миллиметровые шестиугольные оси. Набор (рис. 21.3) позволяет собрать 4 различных варианта сдвоенных редукторов с разными передаточными числами: 12.7:1, 38:1, 115:1 и 344:1.

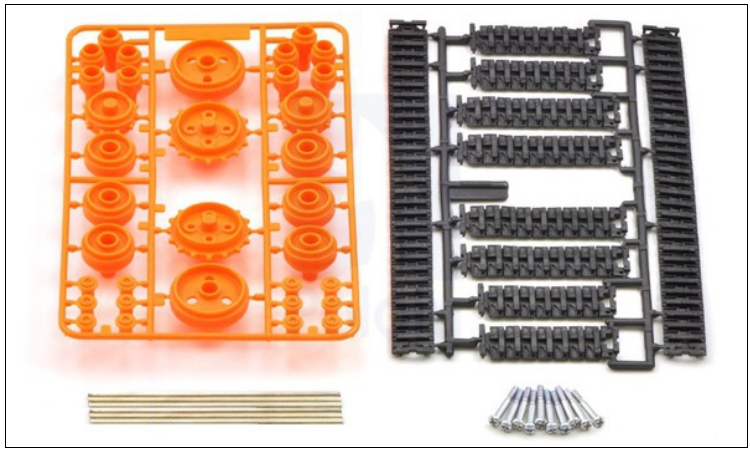


Рис. 21.1. Набор гусениц Tamiya

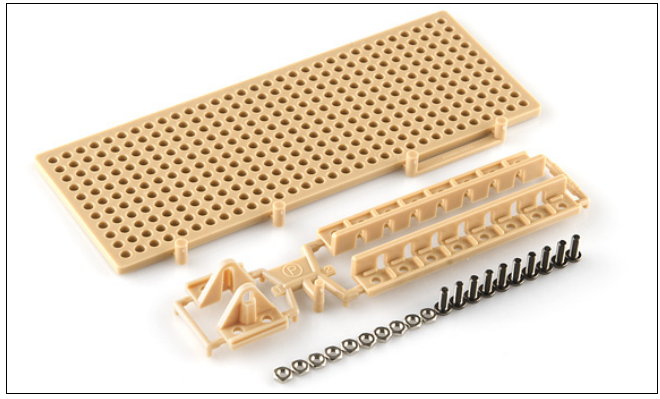


Рис. 21.2. Набор Tamiya 70098 Universal Plate Set

Рис. 21.3. Набор двойного редуктора Tamiya

В наборе используются 3-вольтовые моторы Mabuchi FA-130 (#18100), которые могут быть легко заменены на более мощные моторы Pololu. Размеры корпуса редуктора 70×60×23 мм.

Характеристики редуктора следующие:

- номинальное напряжение — 3 В;
- передаточное отношение — 58:1 или 203:1;
- скорость без нагрузки — 212 или 60 об/мин;
- ток без нагрузки — 150 мА;
- ток под нагрузкой — 2100 мА;
- крутящий момент — 2,0 или 7,3 кг/см;
- диаметр вала — 3 мм;
- размер (Д×Ш×В) — 75×56×23 мм;
- вес — 70 г;
- материал шестеренок — пластик.

К набору прилагается подробная инструкция и поэтому процесс сбора не вызывает затруднений (рис. 21.4).

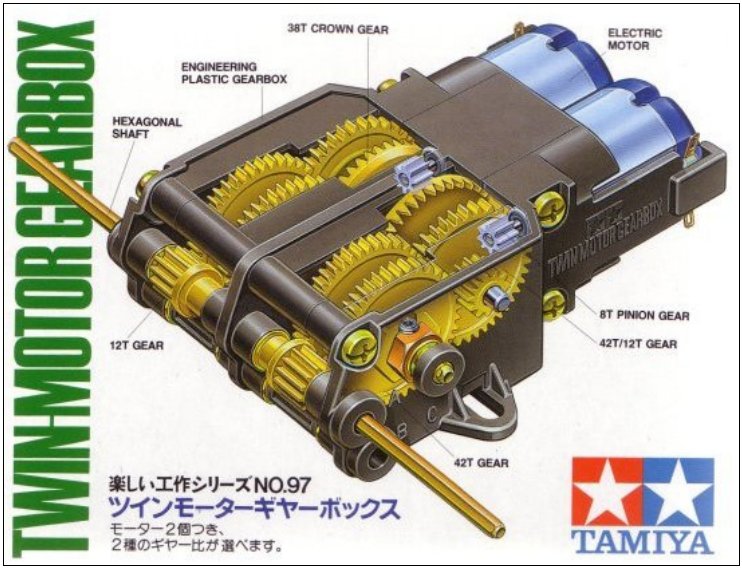


Рис. 21.4. Двойной редуктор Tamiya в сборе

В результате после сборки получается очень неплохая платформа с достаточно мягким ходом и хорошей управляемостью. Вид полученной платформы представлен на рис. 21.5. Ставим сверху комплект электронных компонентов робота и получаем конструкцию, изображенную на рис. 21.6.

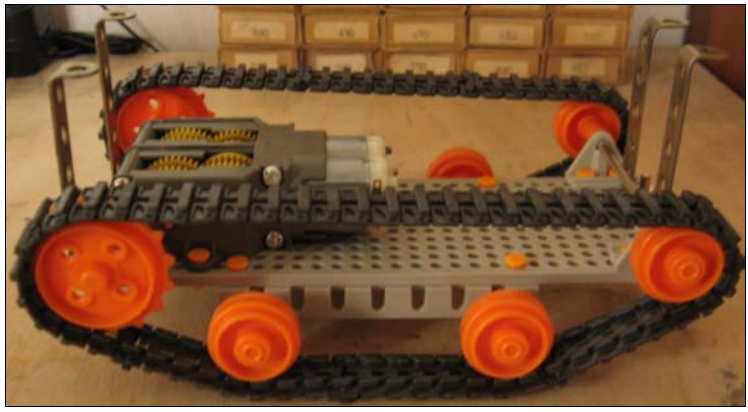


Рис. 21.5. Платформа Tamiya в сборе

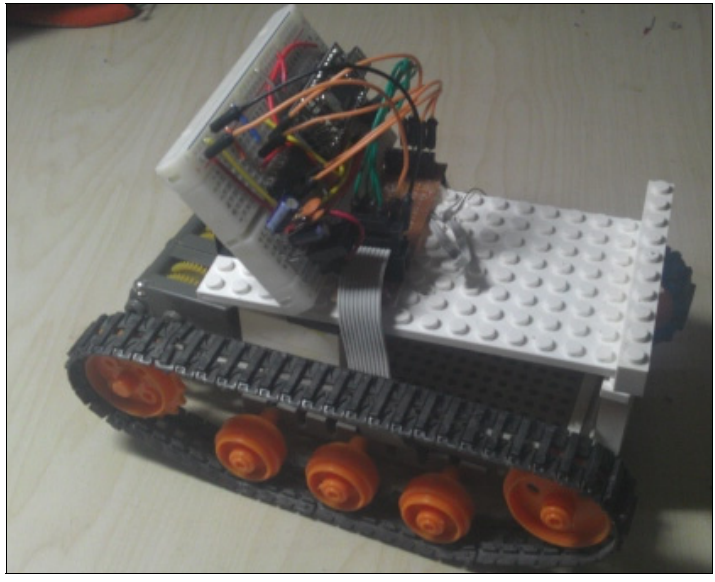


Рис. 21.6. Робот в сборе

21.2. Драйвер двигателей L293D

Нам необходимо реализовать следующие движения робота:

- движение вперед/назад — два мотора крутятся в одну сторону;
- движение влево/вправо — моторы крутятся либо в одну сторону с разной скоростью, либо в разные стороны;
- движение на месте по кругу — моторы крутятся в разные стороны с одной скоростью;
- остановка — оба мотора не крутятся.

Для управления двигателями робота необходимо устройство, которое бы преобразовывало управляющие сигналы малой мощности в токи, достаточные для управления моторами. Такое устройство называют *драйвером двигателей*.

Существует достаточно много самых различных схем для управления электродвигателями. Они различаются как мощностью, так и элементной базой, на основе которой они выполнены. В нашем роботе использовался самый простой драйвер управления двигателями, выполненный на полностью готовой к работе микросхеме. Эта микросхема называется L293D и является одной из самых распространенных микросхем, предназначенных для этой цели. Внешний вид микросхемы L293D представлен на рис. 21.7.

Рис. 21.7. Микросхема L293D

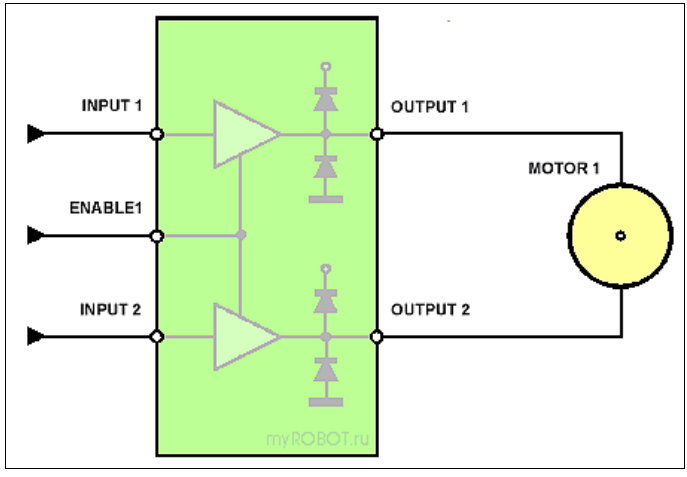


Рис. 21.8. Подключение микросхемы L293D

Микросхема L293D содержит сразу два драйвера для управления электродвигателями небольшой мощности (четыре независимых канала, объединенных в две пары). Она имеет две пары входов для управляющих сигналов и две пары выходов для подключения электромоторов. Кроме того, у L293D есть два входа для включения каждого из драйверов. Эти входы используются для управления скоростью вращения электромоторов с помощью широтно-модулированного сигнала (ШИМ). Микросхема L293D обеспечивает разделение электропитания для микросхемы и для управляемых ею двигателей, что позволяет подключить электродвигатели с большим напряжением питания, чем у самой микросхемы. Разделение электропитания микросхем и электродвигателей может быть также необходимо для уменьшения помех, вызванных бросками напряжения, связанными с работой моторов. Принцип работы каждого из драйверов, входящих в состав микросхемы, идентичен, поэтому рассмотрим его на примере одного из них (рис. 21.8).

К выходам `OUTPUT1` и `OUTPUT2` подключим электромотор `MOTOR1`. На вход `ENABLE1`, включающий драйвер, подадим сигнал (соединим с положительным полюсом источника питания +5 В). Если при этом на входы `INPUT1` и `INPUT2` не подаются сигналы, то мотор вращаться не будет.

Если вход INPUT1 соединить с положительным полюсом источника питания, а вход INPUT2 — с отрицательным, то мотор начнет вращаться. Теперь попробуем соединить вход INPUT1 с отрицательным полюсом источника питания, а вход INPUT2 — с положительным. Мотор начнет вращаться в другую сторону. Попробуем подать сигналы одного уровня сразу на оба управляющих входа INPUT1 и INPUT2 (соединить оба входа с положительным полюсом источника питания или с отрицательным) — мотор вращаться не будет. Если мы уберем сигнал с входа ENABLE1, то при любых вариантах наличия сигналов на входах INPUT1 и INPUT2 мотор вращаться не будет. Эти входы используются для управления скоростью моторов с помощью широтно-модулированного сигнала (ШИМ).

Назначение выводов микросхемы L293D представлено на рис. 21.9.

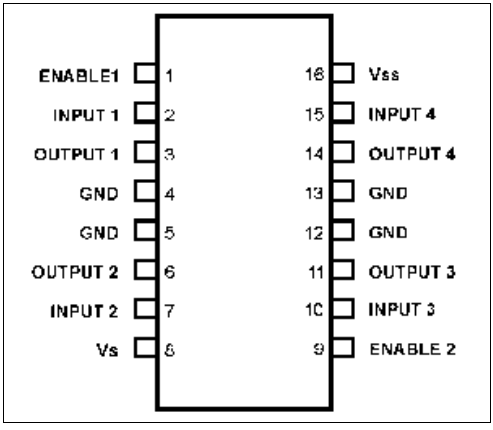


Рис. 21.9. Назначение выводов микросхемы L293D

Входы ENABLE1 и ENABLE2 отвечают за включение каждого из драйверов, входящих в состав микросхемы. Входы INPUT1 и INPUT2 управляют двигателем, подключенным к выходам OUTPUT1 и OUTPUT2. Входы INPUT3 и INPUT4 управляют двигателем, подключенным к выходам OUTPUT3 и OUTPUT4. Контакт Vs соединяем с положительным полюсом источника электропитания двигателей или просто с положительным полюсом питания, если питание схемы и двигателей единое. Проще говоря, этот контакт отвечает за питание электродвигателей. Контакт Vss соединяем с положительным полюсом источника питания. Этот контакт обеспечивает питание самой микросхемы. Четыре контакта GND соединяют с "землей" (общим проводом или отрицательным полюсом источника питания). Кроме того, с помощью этих контактов обычно обеспечивают теплоотвод от микросхемы, поэтому их лучше всего распаивать на достаточно широкую контактную площадку.

Характеристики микросхемы L293D следующие:

- напряжение питания двигателей (Vs) — 4,5–36 В;
- напряжение питания микросхемы (Vss) — 5 В;
- допустимый ток нагрузки — 600 мА (на каждый канал);
- пиковый (максимальный) ток на выходе — 1,2 А (на каждый канал);

- логический "0" входного напряжения — до 1,5 В;
- логическая "1" входного напряжения — 2,3...7 В;
- скорость переключений до 5 кГц;
- имеется защита от перегрева.

Примечание

При использовании аналогичной микросхемы L293E допустимый ток нагрузки на каждый канал будет равен 1 А (а пиковый ток — 2 А), но придется использовать внешние защитные диоды, которые у L293D встроены в саму микросхему.

21.3. Массив возможных состояний моторов

Номера контактов (пинов), к которым подключены моторчики, будем хранить в структуре `MOTOR` (листинг 21.1).

Листинг 21.1

```
struct MOTOR // структура для хранения номеров pinов
 {
 int in1; // INPUT1
 int in2; // INPUT2
 int enable; // ENABLE1
 };
// определяем порты, к которым подключены моторчики
MOTOR MOTOR1 = { 13, 12, 5 };
MOTOR MOTOR2 = { 7, 8, 6 };
```

Подключение платы Arduino к драйверу двигателей L293D производится согласно табл. 21.1.

Таблица 21.1. *Подключение Arduino и микросхемы L293D*

Контакты платы Arduino	Контакты микросхемы L293D
5	9
6	1
7	2
8	7
12	10
13	15

Входы ENABLE1 и ENABLE2 используются для управления скоростью моторов с помощью широтно-модулированного сигнала (ШИМ):

```
analogWrite(MOTOR1.enable, speed);
```

где speed = 0 ... 254.

Экспериментально установлено, что при значении speed < 130 моторы не могут сдвинуть робота с места. Поэтому будем ориентироваться на минимальное значение speed = 150 (при этом значении робот сдвигается с места). Для увеличения/уменьшения скорости выберем значение шага, равное 20. Для удобства управления роботом создадим структуру текущего состояния моторов робота. Создадим также массив состояний моторов для всех возможных состояний робота (листинг 21.2). Здесь требуются некоторые пояснения. Пусть состояние POZ12={m1,m2,d1}, т. е.:

- arrpoz[0][m1] — состояние входа in1 первого мотора;
- arrpoz[1][m1] — состояние входа in2 первого мотора;
- arrpoz[2][m1] — состояние входа enable первого мотора;
- arrpoz[3][m2] — состояние входа in1 второго мотора;
- arrpoz[4][m2] — состояние входа in2 второго мотора;
- arrpoz[5][m2] — состояние входа enable второго мотора.

Рассмотрим состояние POZ12={6,6,6} (см. листинг 21.2):

- arrpoz[0][6]=0; arrpoz[1][6]=0; arrpoz[2][6]=0;
- arrpoz[3][6]=0; arrpoz[4][6]=0; arrpoz[5][6]=0;

Рассмотрим состояние POZ12={10,10,10}:

- arrpoz[0][10]=1; arrpoz[1][10]=0; arrpoz[2][10]=210;
- arrpoz[3][10]=1; arrpoz[4][10]=0; arrpoz[5][10]=210.

В этом случае оба мотора крутятся в одну сторону — скорость 83 % (210/254) от максимальной, движение робота вперед.

Еще пример: состояние POZ12={10,7,10}:

- arrpoz[0][10]=1; arrpoz[1][10]=0; arrpoz[2][10]=210;
- arrpoz[3][7]=1; arrpoz[4][7]=0; arrpoz[5][7]=150.

Здесь первый мотор крутится вперед со скоростью 210, второй мотор крутится вперед со скоростью 150, т. е. обеспечивается поворот (достаточно крутой) вправо. Для увеличения крутизны нужно увеличивать разницу между значениями m1 и m2.

Для вращения робота на месте влево или вправо необходимо установить значения m1 и m2 на равном смещении от 6 в разные стороны — например, m1 = 2 и m2 = 10 (вращение вправо со скоростью 210) или m1 = 8 и m2 = 4 (вращение влево со скоростью 170). Значения arrpoz[6][], arrpoz[7][] и arrpoz[8][] мы разберем позднее.

Листинг 21.2

```
// состояние моторов робота
struct POZ // структура для хранения статусов моторов
{
 int poz1; // позиция для 1 мотора
 int poz2; // позиция для 2 мотора
 int direction12; // направление до поворота (позиция)
};
// массив возможных состояний моторов
// max down -> min down -> stop -> min up -> max up
int arrpoz[9][15]={
 // in1 первого мотора
 {0,0,0,0,0,0,0,1,1,1,1,1,1},
 // in2 первого мотора
 {1,1,1,1,1,1,0,0,0,0,0,0,0},
 // скорости (enable1) первого мотора
 {250,230,210,190,170,150,0,150,170,190,210,230,250},
 // in3 второго мотора
 {0,0,0,0,0,0,0,1,1,1,1,1,1},
 // in4 второго мотора
 {1,1,1,1,1,1,0,0,0,0,0,0,0},
 // скорости (enable2) второго мотора
 {250,230,210,190,170,150,0,150,170,190,210,230,250},
 // скорость++
 {0,-1,-1,-1,-1,-1,0,1,1,1,1,1,0},
 // скорость--
 {-1,-1,-1,-1,-1,-1,0,1,1,1,1,1,1},
 // кружение влево-вправо
 {12,10,8,6,4,2,0,-2,-4,-6,-8,-10,-12}
 };
// начальная позиция робота
// 1,2 мотор - выключены
POZ POZ12={6,6,6};
```

21.4. Разработка скетча движений робота

Приступим к разработке скетча движений робота по командам (при получении кодов) с ИК-пульта. Процедура `loop()` проверяет, получен ли код с ИК-пульта. Если получен, вызывается процедура `ir_go()`. Процедура изменяет значения переменных в структуре текущего состояния моторов робота `POZ`. Содержимое процедуры приведено в листинге 21.3. На данной стадии, для отладки, будем выводить в монитор последовательного порта текущие действия робота.

Рассмотрим назначение переменной `direction12` структуры `POZ`. При выборе движения робота вперед или назад (клавишами ↑ и ↓) по прямой и при увеличе-

нии/уменьшении скорости значение переменной `direction` равно значениям `poz1` и `poz2`.

Листинг 21.3

```
// обработка кода нажатия
void ir_go(long kod)
 {
 switch(kod)
 {
 case FORWARD : // направление вперед
 Serial.print("forward\n");
 POZ12.direction12=max(POZ12.direction12,8);
 POZ12.poz1=POZ12.direction12;
 POZ12.poz2=POZ12.direction12;
 go12(POZ12.poz1,POZ12.poz2);
 break;
 case BACK : // направление назад
 Serial.print("back\n");
 POZ12.direction12=min(POZ12.direction12,6);
 POZ12.poz1=POZ12.direction12;
 POZ12.poz2=POZ12.direction12;
 go12(POZ12.poz1,POZ12.poz2);
 break;
 case SPEED_UP : // скорость++
 Serial.print("speed++\n");
 POZ12.direction12=POZ12.direction12+arrpoz[6][POZ12.direction12];
 POZ12.poz1=POZ12.direction12;
 POZ12.poz2=POZ12.direction12;
 go12(POZ12.poz1,POZ12.poz2);
 break;
 case SPEED_DOWN : // скорость--
 Serial.print("speed--\n");
 POZ12.direction12=POZ12.direction12-arrpoz[7][POZ12.direction12];
 POZ12.poz1=POZ12.direction12;
 POZ12.poz2=POZ12.direction12;
 go12(POZ12.poz1,POZ12.poz2);
 break;
 case LEFT : // влево
 Serial.print("left\n");
 if(POZ12.direction12>6)
 {
 POZ12.poz1=POZ12.poz1-1;POZ12.poz1=max(POZ12.poz1,0);
 POZ12.poz2=POZ12.direction12;
 }
```

```
else
{
POZ12.poz1=POZ12.poz1+1;POZ12.poz1=min(POZ12.poz1,14);
POZ12.poz2=POZ12.direction12;
}
go12(POZ12.poz1,POZ12.poz2);
break;
case RIGHT : // вправо
Serial.print("right\n");
if(POZ12.direction12>6)
{
POZ12.poz2=POZ12.poz2-1;POZ12.poz2=max(POZ12.poz2,0);
POZ12.poz1=POZ12.direction12;
}
else
{
POZ12.poz2=POZ12.poz2+1;POZ12.poz2=min(POZ12.poz2,14);
POZ12.poz1=POZ12.direction12;
}
go12(POZ12.poz1,POZ12.poz2);
break;
case CIRCLE_RIGHT : // кружение вправо
Serial.print("circle_right\n");
POZ12.poz1=POZ12.direction12;
POZ12.poz2=POZ12.direction12+arrpoz[8][POZ12.direction12];
go12(POZ12.poz1,POZ12.poz2);
break;
case CIRCLE_LEFT : // кружение влево
Serial.print("circle_left\n");
POZ12.poz1=POZ12.direction12+arrpoz[8][POZ12.direction12];
POZ12.poz2=POZ12.direction12;
go12(POZ12.poz1,POZ12.poz2);
break;
case STOP : // стоп
Serial.print("stop\n");
POZ12.poz1=0;
POZ12.poz2=0;
POZ12.direction12=0;
go12(POZ12.poz1,POZ12.poz2);
break;
default:
break;
}
}
```

Полностью скетч движений робота при управлении от ИК-пульта находится в папке examples/_21_1 сопровождающего книгу электронного архива.

21.5. Движение робота по линии в автономном режиме

Чтобы наш проект превратился в робота, необходимо добавить для него режим осмысленных действий в автономном режиме. Предусмотрим переход робота из режима внешнего управления в автономный режим и обратно по нажатии клавиш <1> и <3> на ИК-пульте. Для этого добавим в скетч две константы:

```
#define EXT 16 // "1" - переход в режим внешнего управления
#define LOCAL 1040 // "3" - переход в автономный режим
```

При переходе в автономный режим робот перестает реагировать на все коды, поступающие с ИК-пульта, кроме кода перехода в режим внешнего управления.

В автономном режиме робот будет пока непрерывно двигаться по черной линии, нарисованной на поверхности движения. Для обеспечения движения робота по этой линии применим датчики "черное-белое" типа OR-BWSENS (рис. 21.10).

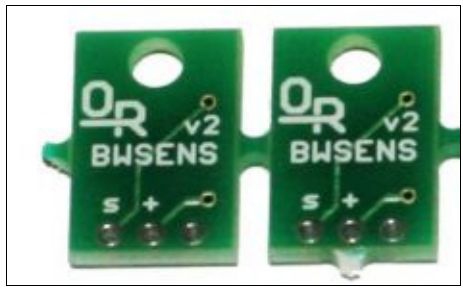


Рис. 21.10. Датчики OR-BWSENS

Датчики устанавливаются на робота, как показано на рис. 21.11. Высота датчиков от поверхности — 15 мм.

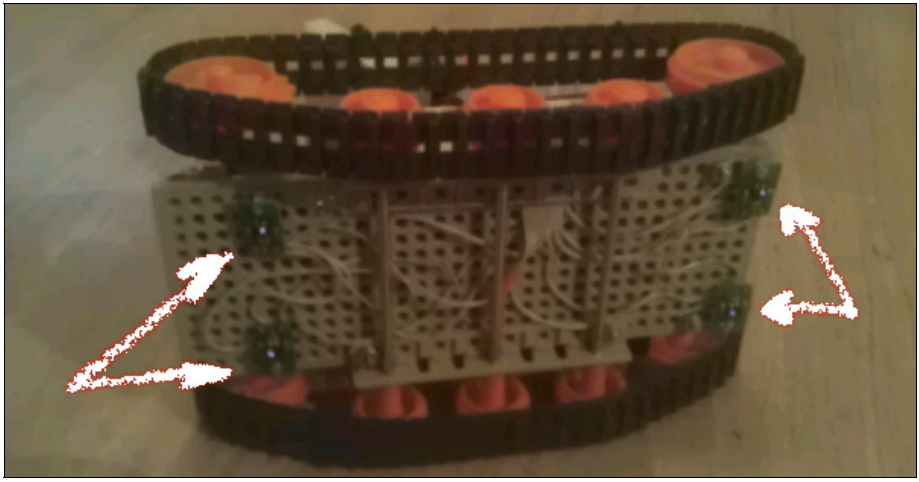


Рис. 21.11. Установка датчиков OR-BWSENS на платформу робота

Для работы с датчиками используем Arduino-библиотеку `PololuQTRSensors`, которую можно скачать со страницы **https://github.com/pololu/qtr-sensors-arduino**. Эта библиотека находится и в папке libraries/QTRSensors сопровождающего книгу электронного архива.

Итак, подключаем библиотеку к проекту:

```
// датчики черное-белое
#include <QTRSensors.h>
#define NUM_SENSORS 4 // кол-во сенсоров
#define TIMEOUT 2500 // 2500 мс
#define EMITTER_PIN QTR_NO_EMITTER_PIN
```

Прописываем константы: количество датчиков, время ожидания разрядки конденсаторов, параметр `EMITTER_PIN`.

Далее создаем экземпляр объекта. Для цифровых:

```
QTRSensorsRC qtrrc((unsigned char[]) {B1,...,Bn},NUM_SENSORS,
TIMEOUT,EMITTER_PIN);
unsigned int sensorValues[NUM_SENSORS];
```

где:

- `B1`, ..., `Bn` — пины Arduino для подключения сенсоров;
- `NUM_SENSORS` — количество сенсоров;
- `TIMEOUT` — время ожидания разрядки конденсатора датчика.

> ***Уточнение***
>
> "Принцип работы датчика: замыкаем на 500 мкс (для версии 1 датчика — на 2500 мкс) сигнальную линию датчика на землю, разряжая конденсатор. Делаем сигнальную линию входом для МК (т. е. перестаем ее подтягивать к «земле» или к питанию) и ждем сколько-то времени, пока через открытый, в зависимости от отражающей способности поверхности в спектре ИК-излучения и расстояния до поверхности, оптодатчик зарядится конденсатор; замеряем, что на выходе с датчика: 0 или 1" (из инструкции для датчиков OR-BWSENS).

- `EMITTER_PIN` — `QTR_NO_EMITTER_PIN`.

Далее в процедуре `setup()` — калибровка датчика:

```
for (i = 0; i < 400; i++)
{
qtrrc.calibrate();
}
```

И в основном цикле — считывание показаний датчиков:

```
unsigned char i;
for (i = 0; i < NUM_SENSORS; i++)
{
Serial.print(sensorValues[i] * 10 / 1001);
Serial.print(' ');
}
```

Черная линия выдавала показания — 24 `(sensorValues[i] * 10 / 1001)`, белый лист — 0–2.

Остается только реализовать движение по черной линии.

Создаем два режима работы робота:

- ❒ внешнее управление;
- ❒ автономный (движение по линии).

Для этого добавляем для ИК-пульта две клавиши:

```
#define EXT 16 //"1" - переход во внешнее управление
#define LOCAL 1040 //"3" - автономный режим
```

И два псевдокода (при наезде на черную линию передним левым и передним правым датчиком):

```
#define LOCAL_LEFT 9997 // для поворота по QTR
#define LOCAL_RIGHT 9998 // для поворота по QTR
```

При срабатывании датчиков выдаем в основной цикл программы срабатывание псевдокодов, и программа `ir_go()` переводит их в движения. Содержимое скетча приведено в листинге 21.4.

Листинг 21.4

```
// подключение библиотеки irremote
#include <IRremote.h>
//вход ИК-приемника
int RECV_PIN = 2;
IRrecv irrecv(RECV_PIN);
decode_results results;
unsigned long ir_dt, old_ir;
long ir_kod;
unsigned long ir_time1, ir_time2;
// коды клавиш ИК-пульта (marmitek)
#define FORWARD 1936
#define BACK 3984
#define SPEED_UP 144 //ch+
#define SPEED_DOWN 2192 //ch-
#define LEFT 3472
#define RIGHT 1424
#define CIRCLE_LEFT 3216 //vol+
#define CIRCLE_RIGHT 1168 //vol-
#define STOP 2320 //0
#define EXT 16 //1
#define LOCAL 1040 //3
#define LOCAL_LEFT 9997 // для поворота по QTR
#define LOCAL_RIGHT 9998 // для поворота по QTR
```

```
int local=0;
// состояние моторов робота
struct POZ // структура для хранения статусов моторов
{
 int poz1; // позиция для 1 мотора
 int poz2; // позиция для 2 мотора
 int direction12; // направление до поворота
 int qtr[4]; //
 long millis1[4]; // для датчиков черное-белое фиксация появления белого
};
// моторы
struct MOTOR // структура для хранения номеров pin-ов, к которым подключены
моторчики
{
 int in1; // INPUT1
 int in2; // INPUT2
 int enable; // ENABLE1
};
// определяем порты, к которым подключены моторчики
MOTOR MOTOR1 = { 13, 12, 5 };
MOTOR MOTOR2 = { 7, 8, 6 };
// начальная позиция робота
// 1,2 мотор - выключены
POZ POZ12={7,7,7,{0,0,0,0},{0,0,0,0}};
// массив возможных состояний моторов
// max down -> min down -> stop -> min up -> max up
int arrpoz[9][17]={
 // in1 первого мотора
 {0,0,0,0,0,0,0,0,1,1,1,1,1,1,1},
 // in2 первого мотора
 {1,1,1,1,1,1,1,0,0,0,0,0,0,0,0},
 // скорости (enable1) первого мотора
 {250,230,210,190,170,150,130,0,130,150,170,190,210,230,250},
 // in3 второго мотора
 {0,0,0,0,0,0,0,0,1,1,1,1,1,1,1},
 // in4 второго мотора
 {1,1,1,1,1,1,1,0,0,0,0,0,0,0,0},
 // скорости (enable2) второго мотора
 {250,230,210,190,170,150,130,0,130,150,170,190,210,230,250},
 // скорость++
 {0,-1,-1,-1,-1,-1,-1,0,1,1,1,1,1,1,0},
 // скорость--
 {-1,-1,-1,-1,-1,-1,-1,0,1,1,1,1,1,1,1},
 // кружение влево-вправо
 {14,12,10,8,6,4,2,0,-2,-4,-6,-8,-10,-12,-14}
 };
```

```
// датчики черное-белое
#include <QTRSensors.h>
#define NUM_SENSORS 4 // кол-во сенсоров
#define TIMEOUT 2500 // 2500 мс
#define EMITTER_PIN QTR_NO_EMITTER_PIN //
// выводы для датчиков
// перед левый 9, перед правый 4, зад левый 11, зад правый 10
QTRSensorsRC qtrrc((unsigned char[]) {4,9,10,11}, NUM_SENSORS, TIMEOUT,
EMITTER_PIN);
unsigned int sensorValues[NUM_SENSORS];

void setup()
 {
 Serial.begin(9600);
 delay(500);
 int i;
 pinMode(13, OUTPUT);
 digitalWrite(13, HIGH); // turn on LED to indicate we are in calibration mode
 for (i = 0; i < 400; i++) // make the calibration take about 10 seconds
 {
 qtrrc.calibrate(); // reads all sensors 10 times at 2500 us per read
                                                    (i.e. ~25 ms per call)
 }
 digitalWrite(13, LOW); // turn off LED to indicate we are through
                                                          with calibration
 for (i = 0; i < NUM_SENSORS; i++)
 {
 Serial.print(qtrrc.calibratedMinimumOn[i]);
 }
 // print the calibration maximum values measured when emitters were on
 for (i = 0; i < NUM_SENSORS; i++)
 {
 Serial.print(qtrrc.calibratedMaximumOn[i]);
 Serial.print(' ');
 }
 Serial.println();
 delay(1000);
 // включить приемник
 irrecv.enableIRIn();
 // прерывания для ИК
 ir_time1=0;ir_time2=0;
 attachInterrupt(0, get_ir_kod, FALLING);
 // настраиваем выводы для моторов
 pinMode(MOTOR1.in1, OUTPUT);
 pinMode(MOTOR1.in2, OUTPUT);
 pinMode(MOTOR2.in1, OUTPUT);
 pinMode(MOTOR2.in2, OUTPUT);
```

```
 // начальная позиция - робот стоит
 go12(POZ12.poz1,POZ12.poz2);
 }

void loop()
 {
 // обработка кода нажатия
 if(ir_kod>0)
 {
 ir_go(ir_kod);
 //Serial.println(ir_kod);
 ir_kod=0;
 }
 // если управление local - проверка датчиков
 if(local==1)
 {
 unsigned int position = qtrrc.readLine(sensorValues);
 qtrrc.read(sensorValues);
 for (int i = 0; i < NUM_SENSORS; i++)
 {
 int qtrv=0;
 if(sensorValues[i]*10/1001==24) qtrv=1;
 if(POZ12.qtr[i]!=qtrv)
 {POZ12.qtr[i]=qtrv;POZ12.millis1[i]=millis();}
 }
 if(POZ12.qtr[0]==1)
 ir_kod=LOCAL_RIGHT;
 else if(POZ12.qtr[1]==1)
 ir_kod=LOCAL_LEFT;
 else
 ir_kod=LOCAL;
 if(ir_kod>0)
 {detachInterrupt(0);local=0;
 ir_go(ir_kod);
 local=1;attachInterrupt(0, get_ir_kod, FALLING);}
 }
 delay(100);
 }

// получить код, переданный с ИК-пульта
void get_ir_kod()
 {
 detachInterrupt(0); // отключить прерывание 0
 if (irrecv.decode(&results))
 {
 if (results.value > 0 && results.value < 0xFFFFFFFF)
 {
 ir_dt = results.value;
```

```
 ir_time2=millis();
 ir_kod=ir_dt;
 Serial.println(ir_kod);
 }
 irrecv.resume();
 }
 attachInterrupt(0, get_ir_kod, FALLING);
 }
// переустановка моторов
void go12(int poz1,int poz2)
 {
 digitalWrite(MOTOR1.in1, arrpoz[0][poz1]);
 digitalWrite(MOTOR1.in2, arrpoz[1][poz1]);
 analogWrite(MOTOR1.enable, arrpoz[2][poz1]);
 digitalWrite(MOTOR2.in1, arrpoz[3][poz2]);
 digitalWrite(MOTOR2.in2, arrpoz[4][poz2]);
 analogWrite(MOTOR2.enable, arrpoz[5][poz2]);
}

// обработка кода нажатия
void ir_go(long kod)
 {
 if(kod!=EXT && local==1)
 return;
 switch(kod)
 {
 case LOCAL : // внешнее управление включить
 POZ12.direction12=10;POZ12.poz1=10;POZ12.poz2=10;
 go12(10,10);local=1;
 break;
 case LOCAL_LEFT : //
 POZ12.direction12=8;POZ12.poz1=7;POZ12.poz2=8;
 go12(POZ12.poz1,POZ12.poz2);
 break;
 case LOCAL_RIGHT : // по датчику вправо
 POZ12.direction12=8;POZ12.poz1=8;POZ12.poz2=7;
 go12(POZ12.poz1,POZ12.poz2);
 break;
 case EXT : // внешнее управление включить
 local=0;
 break;
 case FORWARD : // направление вперед
 POZ12.direction12=max(POZ12.direction12,8);
 POZ12.poz1=POZ12.direction12;
 POZ12.poz2=POZ12.direction12;
 go12(POZ12.poz1,POZ12.poz2);
 break;
```

```
case BACK : // направление назад
POZ12.direction12=min(POZ12.direction12,6);
POZ12.poz1=POZ12.direction12;
POZ12.poz2=POZ12.direction12;
go12(POZ12.poz1,POZ12.poz2);
break;
case SPEED_UP : // скорость++
POZ12.direction12=POZ12.direction12+arrpoz[6][POZ12.direction12];
POZ12.poz1=POZ12.direction12;
POZ12.poz2=POZ12.direction12;
go12(POZ12.poz1,POZ12.poz2);
break;
case SPEED_DOWN : // скорость--
POZ12.direction12=POZ12.direction12-arrpoz[7][POZ12.direction12];
POZ12.poz1=POZ12.direction12;
POZ12.poz2=POZ12.direction12;
go12(POZ12.poz1,POZ12.poz2);
break;
case LEFT : // влево
if(POZ12.direction12>7)
{
POZ12.poz1=POZ12.poz1-1;POZ12.poz1=max(POZ12.poz1,0);
POZ12.poz2=POZ12.direction12;
}
else if(POZ12.direction12<7)
{
POZ12.poz1=POZ12.poz1+1;POZ12.poz1=min(POZ12.poz1,14);
POZ12.poz2=POZ12.direction12;
}
go12(POZ12.poz1,POZ12.poz2);
break;
case RIGHT : // вправо
if(POZ12.direction12>7)
{
POZ12.poz2=POZ12.poz2-1;POZ12.poz2=max(POZ12.poz2,0);
POZ12.poz1=POZ12.direction12;
}
else if(POZ12.direction12<7)
{
POZ12.poz2=POZ12.poz2+1;POZ12.poz2=min(POZ12.poz2,14);
POZ12.poz1=POZ12.direction12;
}
go12(POZ12.poz1,POZ12.poz2);
break;
case CIRCLE_RIGHT : // кружение вправо
POZ12.poz1=POZ12.direction12;
POZ12.poz2=POZ12.direction12+arrpoz[8][POZ12.direction12];
```

```
go12(POZ12.poz1,POZ12.poz2);
break;
case CIRCLE_LEFT : // кружение влево
POZ12.poz1=POZ12.direction12+arrpoz[8][POZ12.direction12];
POZ12.poz2=POZ12.direction12;
go12(POZ12.poz1,POZ12.poz2);
break;
case STOP : // стоп
POZ12.poz1=7;
POZ12.poz2=7;
POZ12.direction12=7;
go12(POZ12.poz1,POZ12.poz2);
break;
default:
break;
}
}
```

Полностью скетч движений робота при ИК-управлении находится в папке examples/_21_2 сопровождающего книгу электронного архива.

ГЛАВА 22

Arduino и шаговые двигатели

Шаговые двигатели представляют собой электромеханические устройства, задачей которых является преобразование электрических импульсов в перемещение вала двигателя на определенный угол. Такие двигатели имеют существенные отличия от обычных, что и определяет их исключительные свойства при использовании в некоторых областях применения.

Шаговый двигатель (ШД) является бесколлекторным двигателем постоянного тока. Как и другие бесколлекторные двигатели, ШД высоконадежен и при надлежащей эксплуатации имеет длительный срок службы. ШД нашли широкое применение в области, где требуется высокая точность перемещений или скорости. Наглядными примерами устройств с ШД могут служить принтеры, факсы и копировальные машины, а также более сложные устройства: станки с ЧПУ (числовым программным управлением), фрезерные, гравировальные машины и т. д.

Достоинствами шаговых двигателей по сравнению с простыми являются:

- угол поворота ротора ШД зависит от числа поданных на двигатель пусковых импульсов;
- максимальный момент на валу ШД развивается в режиме останова (в случае, если обмотки двигателя запитаны);
- высокая точность позиционирования и повторяемости — качественные ШД имеют точность не хуже 2,5 % от величины шага, при этом данная ошибка не накапливается при последующих шагах;
- ШД может быстро стартовать, останавливаться и выполнять реверс;
- высокая надежность двигателя обусловлена отсутствием щеток, при этом срок службы двигателя ограничивается только лишь сроком службы подшипников;
- четкая взаимосвязь угла поворота ротора от количества входных импульсов (в штатных режимах работы) позволяет выполнять позиционирование без применения обратной связи;
- ШД обеспечивает получение сверхнизких скоростей вращения вала для нагрузки, подведенной непосредственно к валу двигателя, без использования редуктора;

- работают ШД в широком диапазоне скоростей, поскольку скорость напрямую зависит от количества входных импульсов.

Недостатки шаговых двигателей:

- ШД обладает явлением резонанса;
- возможен вариант выпадения двигателя из синхронизации с последующей потерей информации о положении при работе цепи обратной связи;
- при стандартных схемах подключения количество потребляемой энергии не уменьшается при отсутствии нагрузки;
- сложность управления при работе на высоких скоростях (на самом деле эффективная работа шагового двигателя на высоких скоростях возможна);
- низкая удельная мощность шагового привода;
- для обеспечения эффективного управления шаговым двигателем требуется очень сложная схема управления.

22.1. Управление шаговым двигателем

В самом общем виде управление шаговым двигателем сводится к задаче отработать определенное число шагов в нужном направлении и с нужной скоростью.

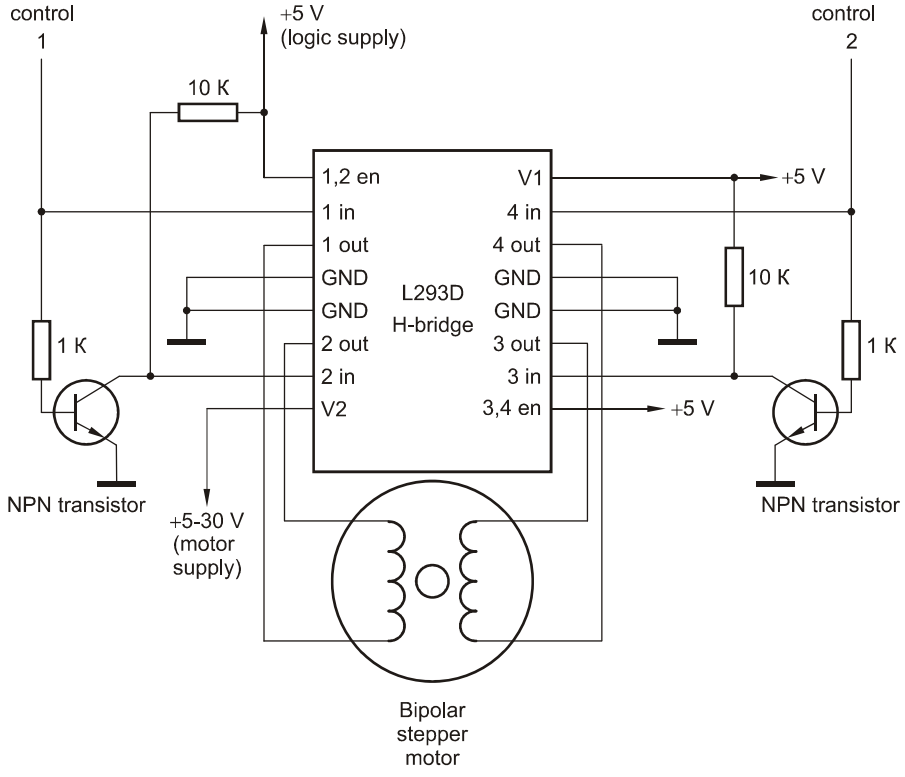


Рис. 22.1. Схема подключения шагового двигателя

На блок управления ШД (драйвер) подаются сигналы "сделать шаг" и "задать направление". Сигналы представляют собой импульсы 5 В. Такие импульсы можно получить от микроконтроллера Arduino. Для шагового двигателя необходимо отдельное питание — выводы шагового двигателя напрямую к выводам Arduino подключать нельзя. Подключение ШД осуществляется через Motor Shild либо с помощью микросхемы драйвера двигателей — например, L293. Схема подключения представлена на рис. 22.1.

22.2. Arduino-библиотека *Stepper*

Для управления шаговым двигателем в Arduino имеется стандартная библиотека `Stepper`. Набор функций у нее следующий:

- `Stepper()`;
- `setSpeed()`;
- `step()`.

22.2.1. Функция *Stepper()*

Функция `Stepper()` создает новый объект класса `Stepper`, привязанный к одному шаговому двигателю, подключенному к контроллеру Arduino. Конструктор следует использовать при объявлении переменной класса `Stepper`, обычно в самом начале — вне `setup()` и `loop()`. Количество параметров зависит от способа подключения — 2 или 4 выхода используются для управления двигателем.

Синтаксис:

```
Stepper(steps, pin1, pin2)
Stepper(steps, pin1, pin2, pin3, pin4)
```

Параметры:

- `steps` — количество шагов в полном обороте используемого двигателя. Если в документации к двигателю указан угол одного шага, то следует разделить 360° на этот угол, что даст нам искомое количество шагов;
- `pin1`, `pin2`, `pin3`, `pin4` — выходы Arduino для подключения шагового двигателя.

Возвращаемое значение: новый экземпляр объекта класса `Stepper`.

22.2.2. Функция *setSpeed(rpm)*

Функция `setSpeed(rpm)` устанавливает скорость вращения в оборотах в минуту. Эта функция не заставляет двигатель вращаться, а лишь устанавливает скорость вращения, которая будет использована при вызове функции `step()`.

Синтаксис:

```
Stepper.setSpeed(rpm)
```

Параметр: `rpm` — скорость, на которой будет производиться вращение шагового двигателя, выражается в оборотах в минуту.

Возвращаемого значения нет.

22.2.3. Функция *step(steps)*

Функция `step(steps)` вращает шаговый двигатель на определенное количество шагов на скорости, заданной функцией `setSpeed()`. Эта функция блокирующая, т. е. она ожидает окончания вращения двигателя, прежде чем передать управление в следующую строку кода. Во избежание длительной блокировки выполнения кода скетча, управление необходимо организовывать так, чтобы скорость была высокая, а за один вызов `step()` делалось всего несколько шагов.

Синтаксис:

```
Stepper.step(steps)
```

Параметр: `steps` — количество шагов:

- положительное число — вызывает вращение в одну сторону;
- отрицательное — в противоположную.

Возвращаемого значения нет.

22.3. Пример использования библиотеки *Stepper*

В листинге 22.1 представлен пример использования библиотеки `Stepper` — поворот шагового двигателя при нажатии кнопок "влево" и "вправо" на определенный угол, что можно представить как программную заготовку для поворота панорамной камеры в одной плоскости. В примере использованы шаговый двигатель и аналоговая клавиатура из *разд. 16.4*. Электрическая схема устройства представлена на рис. 22.2.

Листинг 22.1

```
#include <Stepper.h>
const int stepsPerRevolution=200; // количество шагов в полном обороте
двигателя
Stepper myStepper(stepsPerRevolution,8,9,10,11);

int minangle=15; // угол поворота на 1 нажатие (шаг камеры)
struct KEYS  // структура для хранения статусов клавиш
{
 int button;   // нажатая кнопка
 long millisbutton[7];   // millis для button
};
```

```
void setup() {;}

void loop() {
 int valbutton;
 // опрос клавиатуры
 valbutton=analogRead(A0);
 if(valbutton<1000)
  {buttonClick2(buttonClick1(valbutton));}
}
// обработка нажатия кнопки
int buttonClick1(int val)
 {
 if(val>650) {KEYS1.button=1;return 1;}
 if(val>600) {KEYS1.button=2;return 2;}
 if(val>530) {KEYS1.button=3;return 3;}
 if(val>450) {KEYS1.button=4;return 4;}
 if(val>300) {KEYS1.button=5;return 5;}
 if(val>200) {KEYS1.button=6;return 6;}
 return 0;
 }

 void buttonClick2(int val)
 {
 if(millis()-KEYS1.millisbutton[val]<100) // проверка на дребезг
  return;
 KEYS1.millisbutton[val]=millis();
 KEYS1.button=val;
 switch(val) {
  case 1: // left
    myStepper.step(stepsPerRevolution*minangle/360*(-1));
    break;
  case 2:  // up
   break;
  case 3:  // down
   break;
  case 4:  // right
    myStepper.step(stepsPerRevolution*minangle/360);
    break;
  case 5:  // white
   break;
  case 6:  // red
    // код на нажатие кнопки камеры
    break;
  default:
    break;
 }
 }
```

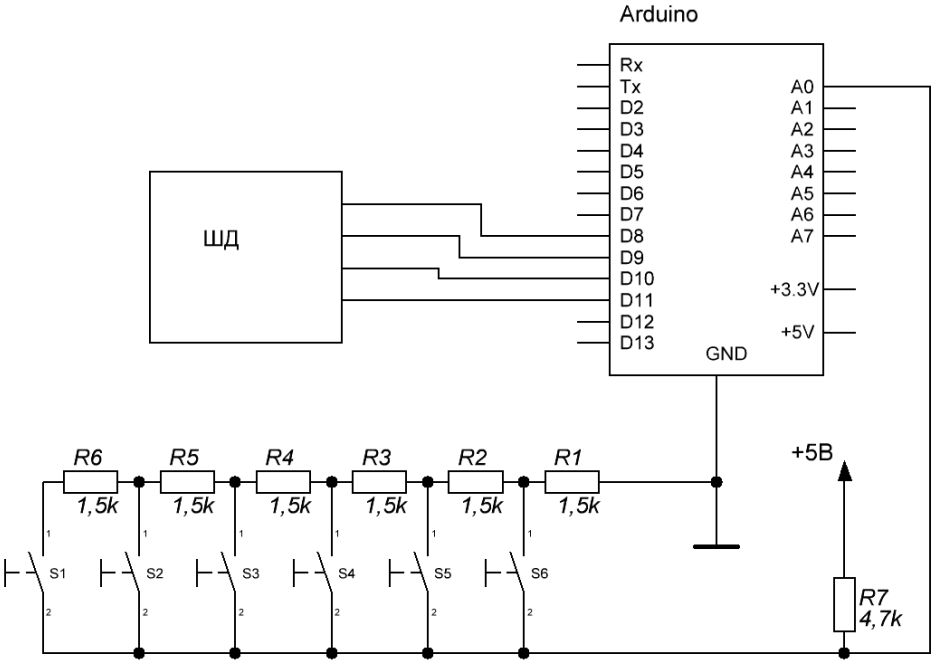


Рис. 22.2. Схема подключения для панорамной головки

Данный скетч можно найти в папке examples/_22_1 сопровождающего книгу электронного архива.

22.4. Arduino-библиотека *AccelStepper*

Стандартной библиотеки `Stepper` вполне достаточно для простого применения с одним двигателем. Библиотека `AccelStepper` значительно улучшает стандартную библиотеку `Stepper`, позволяя подключать несколько шаговых двигателей с управляемым ускорением и замедлением. Можно создать несколько объектов `AccelStepper`, давая уникальное имя каждому двигателю. Библиотеку можно скачать по адресу **http://www.open.com.au/mikem/arduino/AccelStepper**. Она также находится в папке libraries/AccelStepper сопровождающего книгу электронного архива.

ГЛАВА 23

Arduino и сервоприводы

23.1. Сервоприводы

Сервопривод, она же серва, servo, рулевая машинка — устройство, обеспечивающее преобразование сигнала в строго соответствующее этому сигналу перемещение (как правило, поворот) исполнительного устройства. Представляет собой прямоугольную коробку с мотором, схемой и редуктором внутри и выходным валом, который может поворачиваться на строго фиксированный угол, определяемый входным сигналом. Как правило, этот угол имеет предел в 60 градусов, иногда в 180. Бывают сервоприводы и постоянного вращения.

На вал надевается рычаг в форме круга, крестовины или перекладинки для передачи вращающего движения на рабочий орган. После поворота вал остается в том же положении, пока не придет иной управляющий сигнал. Смысл сервопривода в гарантированном выполнении заданной команды. Если внешняя сила не позволит выполнить поворот на нужный угол, сервопривод все равно закончит движение после окончания действия мешающего внешнего воздействия. Воспрепятствовать этому может лишь разрушение сервопривода, снятие внешнего управляющего сигнала или пропадание напряжения питания.

Большинство сервоприводов внешне похожи друг на друга (рис. 23.1) — как правило, это прямоугольный корпус (внутри которого прячутся мотор, шестерни и управляющая схема) с крепежными ушками по бокам и выходным валом, расположенным на верхней крышке. К валу, как уже отмечалось, крепится сменный (идущий в комплекте) передаточный элемент в виде диска (с отверстиями по кругу), крестовины с отверстиями на концах всех его перекладин или рычага, насаженного на выходной вал либо своим центром, либо одним из концов. Как правило, этот сменный элемент белого цвета.

Иногда сервопривод имеет форму цилиндра — тогда его верхняя часть при работе поворачивается вокруг своей оси. Некоторые сервоприводы вместо выходного вала оснащаются шкивом (катушкой). На такой шкив наматывается (или с него сматывается) тросик, с помощью которого управляют каким-то внешним приспособлением — например, парусом в моделях судов или яхт.

Рис. 23.1. Сервопривод

Сервопривод — электрическое исполнительное устройство. Он подключается с помощью трех проводов к управляющему устройству (драйверу или контроллеру) и источнику питания.

По способу управления сервоприводы подразделяются на аналоговые и цифровые. Аналоговые управляются аналоговым сигналом — буквально частотой, параметры которой задаются с помощью широтно-импульсной модуляции (ШИМ). Цифровые сервоприводы управляются цифровым сигналом, представляющим собой кодовые команды, передаваемые по последовательному интерфейсу. Аналоговые сервоприводы намного дешевле цифровых.

Сервопривод управляется с помощью импульсов переменной длительности. Для посылки импульсов используется сигнальный провод. Параметрами этих импульсов являются минимальная длительность, максимальная длительность и частота повторения. Из-за ограничений во вращении сервопривода нейтральное положение определяется как положение, в котором сервопривод обладает одинаковым потенциалом вращения в обоих направлениях. Важно отметить, что различные сервоприводы обладают разными ограничениями в своем вращении, но они все имеют нейтральное положение, и это положение всегда находится в районе длительности импульса в 1,5 миллисекунды (1,5 мс).

Угол поворота определяется длительностью импульса, который подается по сигнальному проводу. Это называется широтно-импульсной модуляцией. Сервопривод ожидает импульса каждые 20 мс. Длительность импульса определяет, насколько далеко должен поворачиваться мотор. Например, импульс в 1,5 мс диктует мотору поворот в положение 90 градусов (нейтральное положение).

Когда сервопривод получает команду на перемещение, его управляющий орган перемещается в это положение и удерживает его. Если внешняя сила действует на сервопривод, когда он удерживает заданное положение, сервопривод будет сопротивляться перемещению из этого положения. Максимальная величина силы, которую может выдерживать сервопривод, характеризует вращающий момент сервопривода. Однако сервопривод не навсегда удерживает свое положение, импульсы

позиционирования должны повторяться, информируя сервопривод о сохранении положения.

Когда импульс, посылаемый на сервопривод, становится короче 1,5 мс, сервопривод поворачивает выходной вал на несколько градусов против часовой стрелки и удерживает это положение. Когда импульс шире, чем 1,5 мс, выходной вал поворачивается на несколько градусов в противоположном направлении. Минимальная и максимальная ширина импульса, который управляет сервоприводом, является свойством конкретного сервопривода. Различные марки, и даже различные сервоприводы одной марки, обладают различным минимумом и максимумом. Как правило, ширина минимального импульса составляет примерно 1 мс и ширина максимального импульса — 2 мс.

Различаются сервоприводы и габаритами. Существуют так называемые *стандартные* сервоприводы. Их габариты и вес в общем модельном ряду соответствуют некоторым средним значениям. Они самые дешевые — в пределах 10–20 долларов. При уменьшении или увеличении размеров сервопривода в сторону от "стандартного" цена сервопривода возрастает пропорционально отклонению размеров. Как и самые маленькие (микросервы), так и самые большие (супермощные) сервоприводы, — это самые дорогие устройства, цена их может доходить до сотен долларов.

Сервоприводы различаются также материалом шестеренок. Самые дешевые сервоприводы оснащены шестернями из пластмассы. Более дорогие — с одной выходной шестерней из металла. Самые дорогие — с металлическими шестернями. Соответственно виду материала изменяется нагрузочная способность сервопривода. Самый слабый сервопривод — с пластиковыми шестернями, самый мощный — с металлическими.

Различия сервоприводов определяются и типом подшипников. Самые дешевые не имеют подшипников вообще. Пластмассовые шестерни на пластмассовых валах крутятся в отверстиях пластмассовых пластин, соединяющих шестерни в единый редуктор. Это самые недолговечные сервоприводы. Более дорогие сервоприводы имеют металлическую, обычно латунную, втулку на выходном валу. Эти сервы более долговечны. Еще более дорогие сервы имеют настоящий подшипник на выходном валу, на который приходится самая большая нагрузка. Подшипник может быть шариковым или роликовым. Шариковый дешевле, роликовый компактнее и легче. В самых дорогих сервоприводах на всех (металлических!) шестернях стоят подшипники. Это — самые долговечные и надежные устройства.

Сервоприводы различаются и по толщине. Она может сильно варьироваться при одинаковых размерах по высоте и длине. Чем меньше толщина, тем выше цена, поскольку в узком корпусе труднее разместить шестерни.

Наконец, сервоприводы различаются по фирме-производителю. Самые раскрученные бренды продают самые дорогие сервоприводы. При этом в их ассортименте будут и самые дорогие, и самые дешевые, но даже самый простенький и дешевый стандартный сервопривод крупного бренда стоит дороже, иногда существенно дороже, чем аналогичный сервопривод с наклейкой менее раскрученного имени и, тем более, с именем никому не известной фирмы.

Очень мощные и очень дорогие сервоприводы могут иметь любую внешнюю форму, определяемую назначением сервопривода. Так, сервоприводы для андроидов (человекоподобных роботов) могут быть шаровидными или дисковидными. Но обычно сервопривод — это все-таки черный параллелепипед с белой фитюлькой на верхней крышке.

Сервоприводы применяются в основном в промышленности (манипуляторы, станки и т. п.). В них установлены, как правило, мощные двигатели, в том числе шаговые, что позволяет использовать их в автоматических манипуляторах (в промышленных роботах).

Сервоприводы для моделирования имеют гораздо более худшие характеристики, чем промышленные. Основным их параметром является момент сил, приложенных к "качалке" на заданном расстоянии от оси. Обычно такой момент измеряется в кг/см. Самые "слабые" сервоприводы тянут 1–2 кг/см. Самые мощные не промышленные сервоприводы — более 100 кг/см. Момент также зависит от питающего напряжения.

Немаловажным параметром является и скорость вращения вала сервопривода. Как правило, она измеряется в сек/60°. Скорость сервопривода обычно важна, если он применяется совместно с гироскопом или акселерометром (например, в моделях вертолетов).

23.2. Arduino-библиотека *Servo* для управления сервоприводом

Для управления сервоприводом в Arduino имеется стандартная библиотека `Servo`. На платах, отличных от Mega, использование библиотеки отключает возможность использования `analogWrite()` (ШИМ) на пинах 9 и 10 (вне зависимости, подключены к этим пинам сервы или нет). На платах Mega до 12 сервоприводов могут использоваться без влияния на функциональность ШИМ, но использование от 12 до 23 сервомашинок отключит ШИМ на пинах 11 и 12. Сервопривод подключается тремя проводами (рис. 23.2): питание (Vcc), "земля" (Gnd) и сигнальный (C). Питание — красный провод, он может быть подключен к выводу +5 В на плате Arduino. Черный или коричневый провод — "земля" подключается к выводу Arduino GND, сигнальный (оранжевый/желтый/белый) провод подключается к цифровому выводу контроллера Arduino. Следует отметить, что мощные сервоприводы могут создавать большую нагрузку — в этом случае их следует запитывать отдельно (не через выход +5 В Arduino). То же самое верно для случая подключения сразу нескольких сервоприводов.

Набор функций библиотеки `Servo`:

- `attach(int);`
- `detach();`
- `write(int);`

- `writeMicroseconds(int)`;
- `read()`;
- `attached()`.

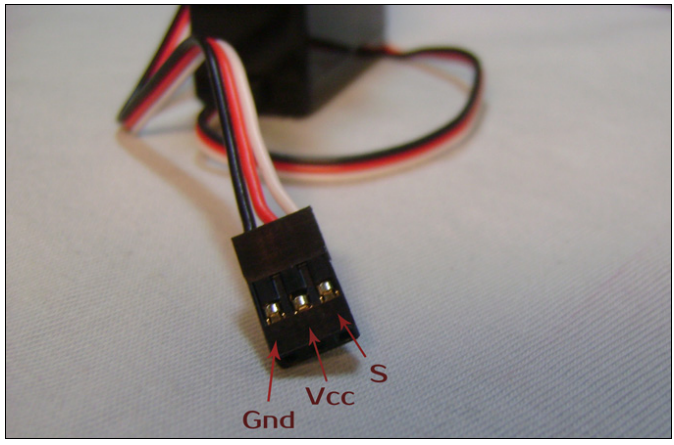


Рис. 23.2. Подключение сервопривода

23.2.1. Функция *attach()*

Функция `attach()` подключает переменную `servo` к указанному выходу, с которого осуществляется управление приводом.

Синтаксис:

```
servo.attach(pin)
servo.attach(pin, min, max)
```

Параметры:

- `pin` — номер выхода, к которому подключаем сервопривод и с которого осуществляется управление приводом;
- `min` — ширина импульса в микросекундах, соответствующая минимальному (угол 0 градусов) положению сервопривода;
- `max` — ширина импульса в микросекундах, соответствующая максимальному (угол 180 градусов) положению сервопривода.

Возвращаемого значения нет.

23.2.2. Функция *detach()*

Функция `detach()` отсоединяет переменную `servo` от подключенного функцией `attach()` выхода. Если все переменные `servo` отсоединены, то выходы 9 и 10 могут быть использованы в режиме ШИМ с помощью `analogWrite()`.

Синтаксис:

```
servo.detach()
```

Параметров нет.

Возвращаемого значения нет.

23.2.3. Функция *write(int)*

Функция `write(int)` передает значения для управления приводом. Для стандартного сервопривода это угол поворота. Для привода постоянного вращения функция задает скорость вращения: 0 — для максимальной скорости вращения в одну сторону, 180 — для максимальной скорости в другую сторону и около 90 для неподвижного состояния.

Синтаксис:

```
servo.write(angle)
```

Параметр: `angle` — значение, записываемое в `servo` (от 0 до 180).

Возвращаемого значения нет.

23.2.4. Функция *writeMicrosconds(int)*

Функция `writeMicrosconds(int)` передает значение для управления сервоприводом в микросекундах (uS), устанавливая угол поворота на это значение. Для стандартного привода: значение 1000 — максимальный поворот против часовой стрелки, 2000 — максимальный поворот по часовой стрелке, 1500 — посередине.

Синтаксис:

```
servo.writeMicrosconds (uS)
```

Параметр: `uS` — значение в микросекундах.

Возвращаемого значения нет.

23.2.5. Функция *read()*

Функция `read()` считывает значение текущего положения сервопривода (значение, записанное последним вызовом функции `write()`).

Синтаксис:

```
servo.read()
```

Параметров нет.

Возвращаемое значение: `angle` — положение (угол) сервопривода от 0 до 180.

23.2.6. Функция *attached()*

Функция `attached()` проверяет, подключена ли переменная `Servo` к выходу.

Синтаксис:

```
servo.attached()
```

Параметров нет.

Возвращаемые значения: `true` — если подключена, `false` — в противном случае.

23.3. Робот-паук на сервоприводах

Рассмотрим проект по созданию простейшего четвероногого робота-паука на сервоприводах. У робота будут два режима:

- автономный — робот движется вперед, при обнаружении препятствия (используется ультразвуковой датчик) поворачивается и движется дальше;
- внешнее управление с помощью ИК-пульта (вперед, назад, поворот влево, поворот вправо, остановка, засыпание).

Для проекта понадобятся 4 сервопривода. Мы используем сервомашинки Turnigy TGY-9025MG с металлическим редуктором (рис. 23.3), достаточно надежные и очень дешевые (цена в Китае с учетом пересылки в Россию порядка 130 руб.).

Рис. 23.3. Сервопривод Turnigy TGY-9025MG

В качестве ног робота были взяты заглушки для струйных картриджей, скрепленные с помощью поликапролактона (рис. 23.4 и 23.5). Корпус сделан из упаковочного материала для компьютерной техники (рис. 23.6).

Для сервомашинок требуется отдельное питание. В качестве источника питания взята литий-полимерная батарея Turnigy 2S 1600 mAh (рис. 23.7). Напряжение, выдаваемое батареей: 7,4–8,4 В. Поскольку для питания сервоприводов необходимо напряжение 4,8–6,0 В, применим стабилизатор напряжения 5 В, собранный на микросхеме L7805. Как выяснилось, одна микросхема постоянно перегревалась, проблема была решена установкой параллельно двух микросхем L7805.

Для обнаружения препятствий использован ультразвуковой датчик HC-SR04 (см. *главу 16*), который позволяет определять расстояние до объекта в диапазоне

от 2 до 500 см с точностью 0,3 см. Датчик излучает короткий ультразвуковой импульс (в момент времени 0), который отражается от объекта и принимается сенсором. Расстояние рассчитывается, исходя из времени до получения эха и скорости звука в воздухе. Если расстояние до препятствия меньше 10 см, робот делает поворот и движется дальше вперед.

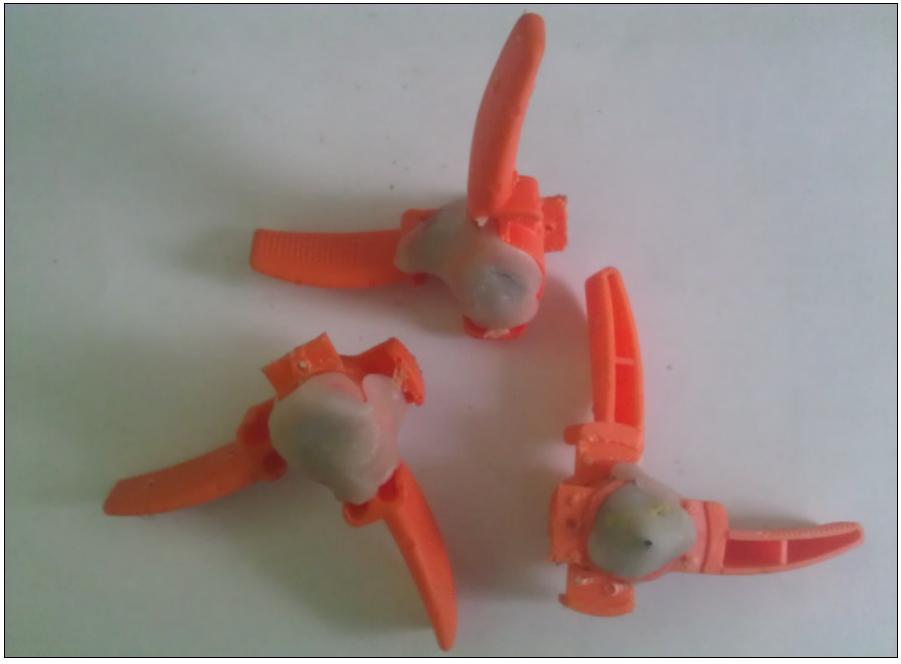


Рис. 23.4. Ноги робота, скрепленные поликапролактоном

Рис. 23.5. Ноги робота — крепление к сервомашинке

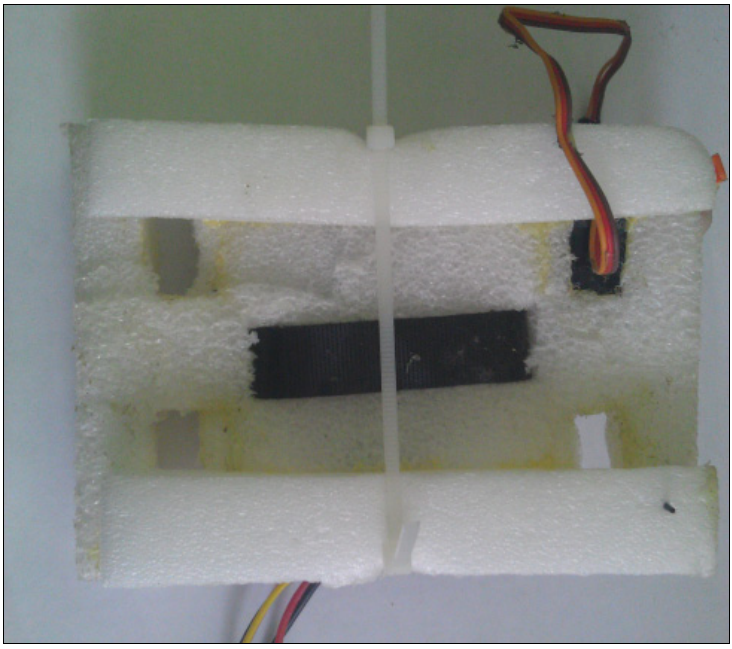


Рис. 23.6. Делаем корпус робота-паука

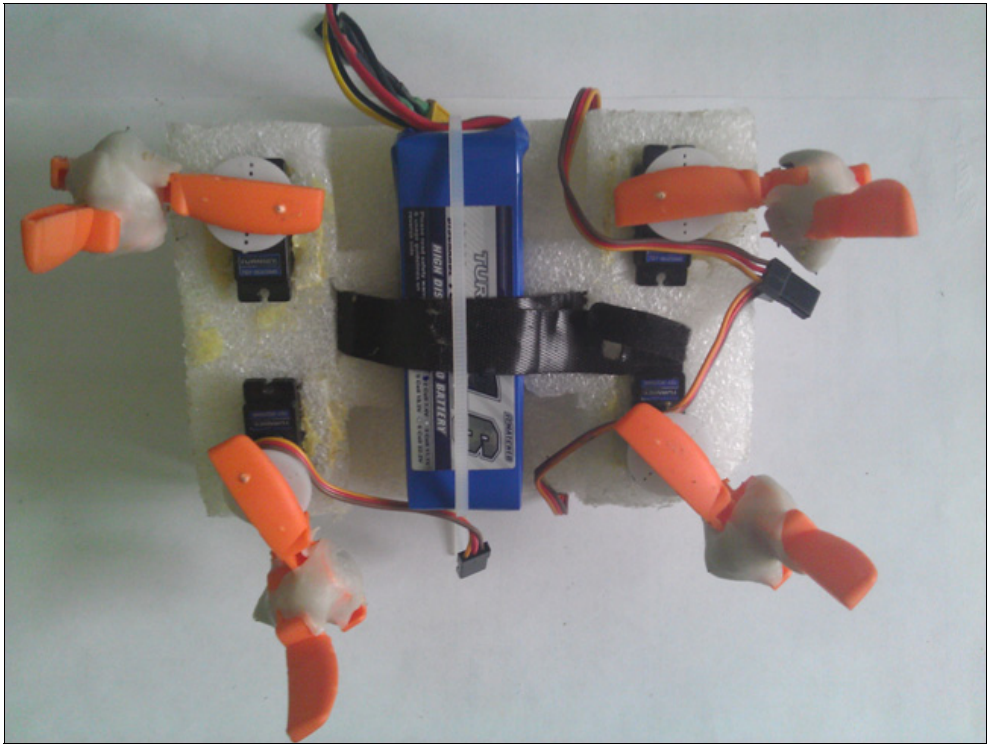


Рис. 23.7. Добавляем батарею для питания сервоприводов

В качестве пульта применен пульт LG, приемник ИК-сигналов — TSOP31238. Вид робота в сборе представлен на рис. 23.8. Его электрическая схема приведена на рис. 23.9.

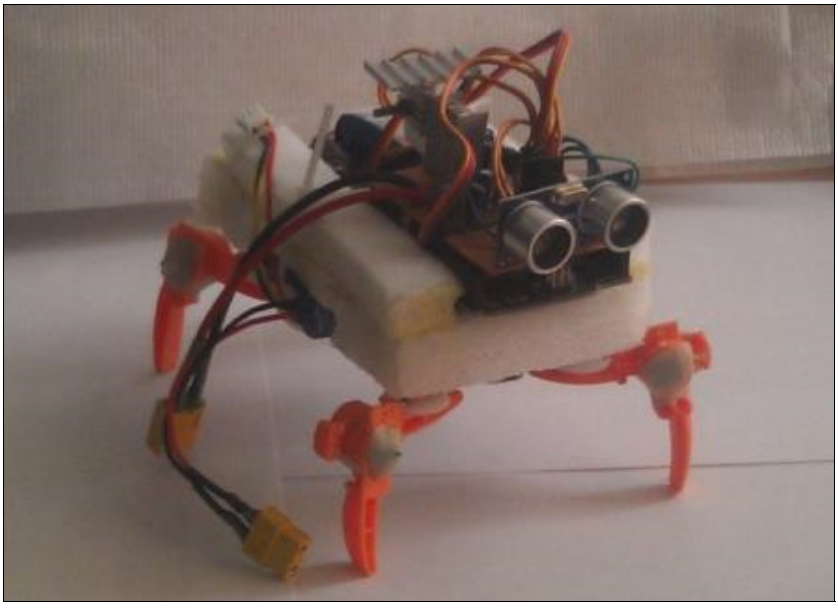


Рис. 23.8. Робот-паук в сборе

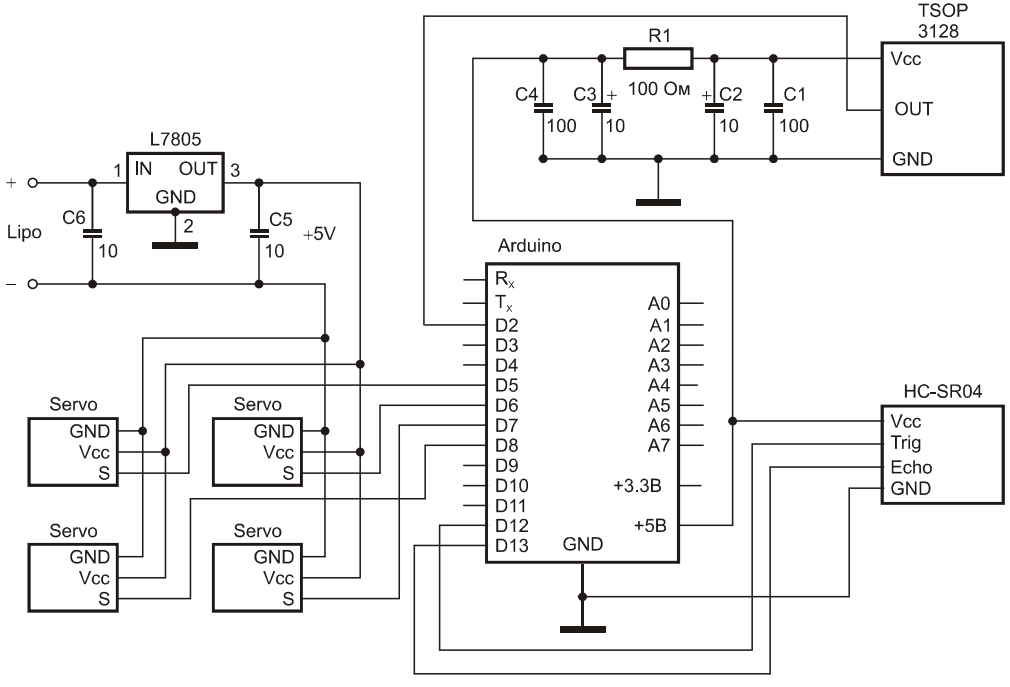


Рис. 23.9. Электрическая схема паука-робота

23.4. Скетч для управления роботом-пауком

Для управления сервоприводами используется Arduino-библиотека `Servo`. Нам необходимо реализовать настройку сервоприводов для движения робота-паука вперед, назад, поворота по часовой стрелке и поворота против часовой стрелки, функции остановки робота, а также — для экономии электроэнергии — предусмотрим режим засыпания (когда сервоприводы находятся в режиме `detach`) и пробуждения (перевод сервоприводов в режим `attach`). Поэтому каждое движение робота состоит из нескольких шагов.

Например, движение вперед состоит из следующих шагов:

- ❒ левая передняя нога вперед;
- ❒ правая передняя нога вперед;
- ❒ левая задняя нога вперед;
- ❒ правая задняя нога вперед;
- ❒ четыре ноги вместе назад (что приведет к перетаскиванию тела робота-паука).

Данные для угла поворота каждого сервопривода на каждом шаге для каждого движения робота-паука хранятся в трехмерном массиве `arr_pos` (листинг 23.1).

Листинг 23.1

```
int arr_pos[4][6][4]={
 { // forward
 {90,90,90,90},{45,90,90,90},{45,135,90,90},
 {45,135,45,90},{45,135,45,135},{135,45,135,45}
 },
 { // back
 {90,90,90,90},{90,90,90,45},{90,90,135,45},
 {90,45,135,45},{135,45,135,45},{45,135,45,135}
 },
 { // circle_left
 {90,90,90,90},{0,90,90,90},{0,0,90,90},
 {0,0,0,90},{0,0,0,0},{180,180,180,180}
 },
 { // circle_right
 {90,90,90,90},{180,90,90,90},{180,180,90,90},
 {180,180,180,90},{180,180,180,180},{0,0,0,0}
 }
 };
int pos_stop[1][4]={{90,90,90,90}};
```

Процедура `course(int variant)` реализует перемещение сервоприводов для каждого шага следующих движений робота-паука: вперед, назад, поворота по часовой стрелке и поворота против часовой стрелки (листинг 23.2).

Листинг 23.2

```
void course(int variant)
 {int i=0;;
 for(i=1;i<6;i++)
 {
 if(arr_pos[variant-1][i][0]!=arr_pos[variant-1][i-1][0])
 {myservo11.write(arr_pos[variant-1][i][0]);}
 if(arr_pos[variant-1][i][1]!=arr_pos[variant-1][i-1][1])
 {myservo12.write(arr_pos[variant-1][i][1]);}
 if(arr_pos[variant-1][i][2]!=arr_pos[variant-1][i-1][2])
 {myservo13.write(arr_pos[variant-1][i][2]);}
 if(arr_pos[variant-1][i][3]!=arr_pos[variant-1][i-1][3])
 {myservo14.write(arr_pos[variant-1][i][3]);}
 delay(200);
 }
 }
```

Для остановки, засыпания и пробуждения робота-паука существует процедура `go_hor_all()` (листинг 23.3).

Листинг 23.3

```
void go_hor_all()
 {
 myservo11.write(pos_stop[0][0]);
 myservo12.write(pos_stop[0][1]);
 myservo13.write(pos_stop[0][2]);
 myservo14.write(pos_stop[0][3]);
 delay(500);
 }
```

Реализуем простое ИК-управление с пульта. Выбираем семь клавиш, данные о кодах заносим в скетч в виде констант и в цикле `loop()` реализуем логику выбора движений робота-паука при нажатии клавиш ИК-пульта. Программа получения кода `get_ir_kod()` вызывается по прерыванию `CHANGE` на входе 2. Используется Arduino-библиотека `IRremote` (см. *главу 20*).

К режиму управления робота с ИК-пульта добавим автономный режим. В автономном режиме робот будет двигаться вперед, при достижении препятствия делать поворот и опять двигаться вперед. Примененный ультразвуковой дальномер HC-SR04, как уже отмечалось в *разд. 23.3*, позволяет определять расстояние до объекта в диапазоне от 2 до 500 см с точностью 0,3 см. Короткий ультразвуковой импульс, излученный датчиком в момент времени 0, отражается от объекта и принимается сенсором. Расстояние рассчитывается, исходя из времени до получения отраженного эха и скорости звука в воздухе. Если расстояние до препятствия меньше 10 см,

робот делает поворот и продолжает движение. Переход из режима ИК-управления в автономный режим осуществляется нажатием клавиш "желтая" и "синяя" на ИК-пульте.

Для работы с датчиком HC-SR04 используется Arduino-библиотека `Ultrasonic` (листинг 23.4).

Листинг 23.4

```
#include "Ultrasonic.h"
// trig -12, echo - 13
Ultrasonic ultrasonic(12, 13);
// коды клавиш ИК-пульта lg 6710v00090d
#define FORWARD 32   // pr +
#define BACK 33    // pr -
#define CIRCLE_LEFT 17 // vol-
#define CIRCLE_RIGHT 16 // vol+
#define STOP 54     // зеленая
#define SLEEP 55     // красная
#define AWAKE 37    // ок
#define EXT 50     // желтая
#define AUTO 52     // синяя

... .... .....

void loop()
 {
 delay(1000);
 if(ext==0)
 {
 float dist_cm = ultrasonic.Ranging(CM);
 Serial.print("dist_cm=");Serial.println(dist_cm);
 if(dist_cm<10.0)
  ir_kod=CIRCLE_LEFT;
 else
  ir_kod=FORWARD;
 }
 if(ir_kod!=0)
 {
 Serial.print("ir_kod=");Serial.println(ir_kod);
 switch(ir_kod)
  {
  case FORWARD :  // вперед
    course(1);
    Serial.print("forward\n");
    break;
  case BACK :  // назад
    course(2);
```

```
    Serial.print("back\n");
    break;
  case CIRCLE_LEFT:  // вращение влево
    course(3);
    Serial.print("circle_left\n");
    break;
  case CIRCLE_RIGHT :  // вращение вправо
    Serial.print("circle_right\n");
    course(4);
    break;
  case STOP :  // остановка
    ir_kod=0;
    go_hor_all();
    Serial.print("pause\n");
    break;
  case SLEEP :  // засыпание
    ir_kod=0;
    go_hor_all();
    myservo11.detach();myservo12.detach();
    myservo13.detach();myservo14.detach();
    digitalWrite(13,LOW);
    Serial.print("sleep\n");
    break;
  case AWAKE :  // пробуждение
    ir_kod=0;
    myservo11.attach(5);myservo12.attach(6);
    myservo13.attach(7);myservo14.attach(8);
    digitalWrite(13,HIGH);
    go_hor_all();
    Serial.print("awake\n");
    break;
  case AUTO :  // режим автономный
    //ir_kod=FORWARD;
    ext=0;
    myservo11.attach(5);myservo12.attach(6);
    myservo13.attach(7);myservo14.attach(8);
    Serial.print("auto\n");
    break;
  default:
    break;
  }
 }
 }
// получить код, переданный с ИК-пульта
void get_ir_kod()
 {
 detachInterrupt(0);  // отключить прерывание 0
```

```
if (irrecv.decode(&results))
{
if (results.value > 0 && results.value < 0xFFFFFFFF)
 {
 ir_dt = results.value;
 if(ir_dt==EXT && ext==0)
  {ir_kod = SLEEP;ext=1;}
 else if(ext==1)
  {
  if(ir_dt==FORWARD || ir_dt==BACK || ir_dt==CIRCLE_LEFT
   || ir_dt==CIRCLE_RIGHT || ir_dt==STOP || ir_dt==SLEEP
   || ir_dt==AWAKE || ir_dt==AUTO )
  ir_kod = ir_dt;
  }
 else
  ;
 }
 irrecv.resume();
}
attachInterrupt(0, get_ir_kod, CHANGE);
}
```

Полностью данный скетч можно найти в папке examples/_23_1 сопровождающего книгу электронного архива.

ГЛАВА 24

Arduino и Bluetooth

24.1. "Голубой зуб"

Bluetooth (с англ. — "голубой зуб") — одна из технологий беспроводной передачи данных. Спецификация была разработана в 1998 году компанией Ericsson, а позднее оформлена группой Bluetooth Special Interest Group (SIG), официально зарегистрированной 20 мая 1999 года.

Bluetooth позволяет объединять в локальные сети любую технику: от мобильного телефона и компьютера до холодильника. При этом, одним из немаловажных параметров новой технологии является низкая стоимость устройства связи (в пределах 20 долларов), его небольшие размеры (ведь речь идет о мобильных устройствах) и, что немаловажно — совместимость, простота встраивания в различные устройства.

24.2. Модуль Bluetooth HC-05

Модуль Bluetooth HC05, благодаря невысокой цене, получил широкое распространение. В российских интернет-магазинах его можно приобрести по цене 300–330 руб.

Модуль представляет собой плату размером 2,7×1,4 см, имеющую 34 вывода с шагом 1,5 мм, расположенных по периметру платы (рис. 24.1). На плате смонтирован чип BC417 от компании Cambridge Silicon Radio, который обеспечивает аппаратную поддержку стека Bluetooth 2.0+EDR (Enhaced Data Rate), а также флеш-память ES29LV800DB-70WGI от Excel Semiconductor на 8 Мбит (1 Мбайт), хранящая прошивку и настройки. Напряжение питания модуля: 3,3 В. Потребляемый ток: 50 мА.

Модуль может работать в 3-х режимах:

- Master (или сервер) — в этом режиме модуль может сам подключиться к какому-нибудь Bluetooth-устройству;
- Slave — в этом режиме другой мастер может подключиться к нашему модулю;
- Slave-loop — здесь модуль отправляет обратно все байты, которые ему прислали.

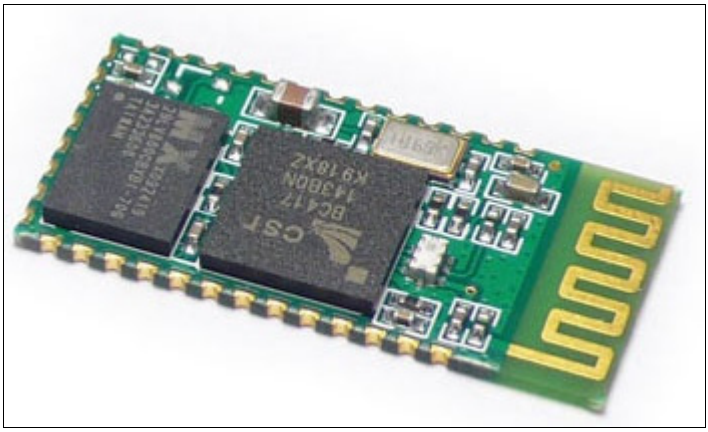


Рис. 24.1. Модуль Bluetooth HC05

Электрическая схема подключения модуля Bluetooth HC05 к Arduino приведена на рис. 24.2. Программируется модуль с помощью AT-команд через UART-интерфейс модуля. Чтобы перевести модуль в режим программирования, необходимо подать на вывод 34 логическую 1 (3,3 В). Затем подключиться к UART Bluetooth-модуля: выводы 2 (Rx) и 1 (Tx) — и передавать команды на модуль.

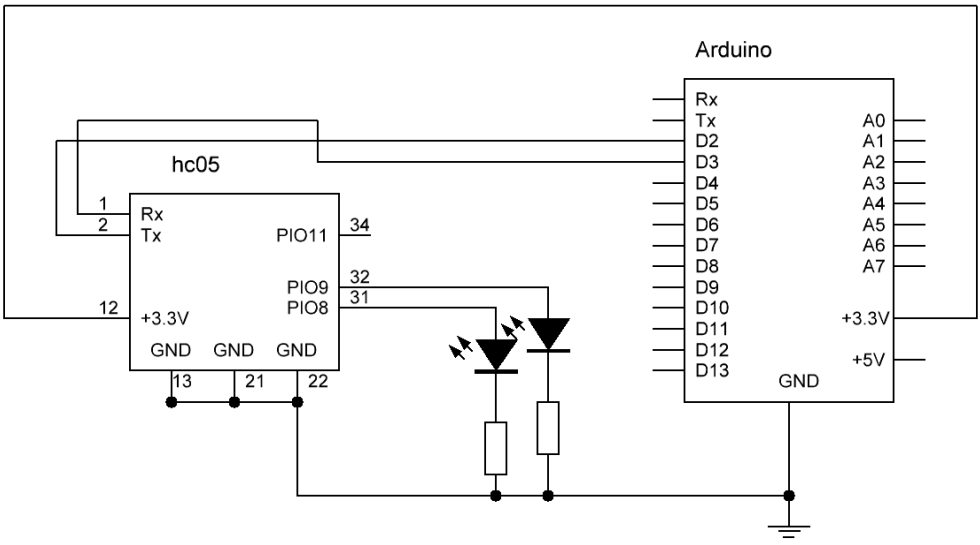


Рис. 24.2. Подключение модуля Bluetooth HC05 к Arduino

Подключимся к Arduino по UART, который эмулируем программно (для этого подключим библиотеку `SoftwareSerial`). Этим мы освобождаем последовательный порт Arduino, что дает возможность одновременного считывания данных через кабель и через Bluetooth-модуль. Смотрите внимательно: контакт Rx Bluetooth-модуля подключается к контакту Tx Arduino, а контакт Tx Bluetooth-модуля — к контакту Rx Arduino. Загружаем в Arduino скетч, представленный в листинге 24.1.

Листинг 24.1

```
#include <SoftwareSerial.h>
SoftwareSerial mySerial(2, 3); // указываем пины rx и tx соответственно

void setup()
{
 pinMode(2,INPUT);
 pinMode(3,OUTPUT);
 Serial.begin(38400);
 mySerial.begin(38400);
 Serial.println("38400");
 Serial.println("ok");
}

void loop()
{
 if (mySerial.available())
 {
  int c = mySerial.read(); // читаем из software-порта
  Serial.write(c); // пишем в hardware-порт
 }
 if (Serial.available())
 {
  int c = Serial.read(); // читаем из hardware-порта
  mySerial.write(c); // пишем в software-порт
 }
}
```

Открываем монитор последовательного порта и начинаем отправлять отправку команд в Bluetooth-модуль. Все команды имеют вид AT+КОМАНДА, AT+КОМАНДА? или AT+КОМАНДА=ПАРАМЕТРЫ и должны оканчиваться комбинацией CR+LF (символы с кодами 0x0D и 0x0A, '\r' и '\n'). На каждую команду Bluetooth-модуль шлет ответ.

Список основных AT-команд:

- AT — тестовая команда.

  Параметров нет.

  Ответ модуля: OK

- AT+VERSION? — получить версию прошивки модуля.

  Параметров нет.

  Ответ модуля: +VERSION: <Param>
  OK

  где <Param> — версия прошивки Bluetooth-модуля.

- AT+RESET — сброс настроек.

  Параметров нет.

  Ответ модуля: OK

- AT+ORGL — установка пользовательских настроек модуля.

  Параметров нет.

  Ответ модуля: OK

- AT+ADDR? — получить адрес модуля.

  Параметров нет.

  Ответ модуля: +ADDR:<Param>

  где <Param> — адрес Bluetooth-модуля NAP: UAP : LAP.

- AT+NAME? — получить имя модуля.

  Параметров нет.

  Ответ модуля: +NAME:<Param>

  где <Param> — имя Bluetooth-модуля.

- AT+NAME=<Param> — установить новое имя модуля.

  Параметр: <Param> — имя Bluetooth-модуля.

  Ответ модуля: +NAME:<Param>

  OK (или FAIL)

- AT+PSWD? — получить пин-код доступа к Bluetooth-модулю.

  Параметров нет.

  Ответ модуля: +PSWD:<Param>

  где <Param> — пин-код. По умолчанию 1234.

- AT+PSWD=<Param> — установить код доступа к Bluetooth-модулю.

  Параметр: <Param> — код доступа к модулю.

  Ответ модуля: OK (или FAIL).

- AT+CLASS=<Param> — установить режим работы модуля Bluetooth-модуля.

  Параметр: <Param> — класс. В документации модуля не приведены возможные значения данного параметра. По умолчанию он установлен в 0. Если предполагается использовать модуль в режиме master, значение не надо изменять. Если использовать модуль в режиме slave, при значении параметра, равном 0, он невидим для устройств с операционной системой Android. Для видимости необходимо установить значение параметра равным 7936.

  Ответ модуля: OK

- AT+CLASS? — получить класс модуля.

  Параметров нет.

Ответ модуля: `+CLASS:<Param>`

где `<Param>` — класс модуля.

- `AT+IAC` — получить код доступа к запросу GIAC (General Inquire Access Code).

  Различным физическим каналам связи, используемым в процессе установления соединения, соответствуют различные коды доступа к каналам. В каналах опроса, за исключением выделенных, используется одинаковый для всех устройств общий код доступа к запросу.

  Параметров нет.

  Ответ модуля: `+IAC:<Param>`

  где `<Param>` — код доступа к запросу.

- `AT+IAC=<Param>` — установить код доступа к запросу.

  Параметр: `<Param>` — код доступа к запросу. Значение по умолчанию 9e8b33.

  Ответ модуля: `OK` (или `FAIL`).

- `AT+ROLE?` — получить режим работы модуля.

  Параметров нет.

  Ответ модуля: `+ROLE:<Param>`

  где `<Param>` — режим работы модуля Bluetooth-модуля:

  - `0` — slave. В этом режиме другой мастер может подключиться к модулю;
  - `1` — master. В этом режиме модуль может сам подключиться к какому-нибудь Bluetooth-устройству;
  - `2` — slave-loop. Модуль отправляет обратно все байты, которые ему прислали.

- `AT+ROLE=<Param>` — установить режим работы Bluetooth-модуля.

  Параметр: `<Param>` — режим работы Bluetooth-модуля:

  - `0` — slave;
  - `1` — master;
  - `2` — slave-loop.

  Ответ модуля: `OK`

- `AT+UART=<Param1>,<Param2>,<Param3>` — установить модуль для последовательного порта.

  Параметры:

  - `<Param1>` — скорость обмена (9600,19200,38400,57600,115200);
  - `<Param2>` — стоп-бит:
    - `0` — нет;
    - `1` — есть;

- <Param3> — бит паритета:
  - 0 — нет;
  - 1 — есть.

Ответ модуля: OK (или FAIL).

- AT+UART? — получить параметры обмена модуля.

  Параметров нет.

  Ответ модуля: +UART:<Param1>,<Param2>,<Param3>

  где:

  - <Param1> — скорость обмена (9600,19200,38400,57600,115200);
  - <Param2> — стоп-бит;
  - <Param3> — бит паритета.

- AT+CMODE=<Param> — установить режим подключения Bluetooth-модуля.

  Параметр: <Param> — режим подключения Bluetooth-модуля:

  - 0 — модуль может подключаться только к определенному командой AT+BIND Bluetooth-устройству;
  - 1 — модуль может подключаться к любому Bluetooth-устройству;
  - 2 — режим slave-loop.

  Ответ модуля: OK

- AT+CMODE? — получить режим подключения модуля.

  Параметров нет.

  Ответ модуля: +CMODE:<Param>

  где <Param> — режим подключения Bluetooth-модуля:

  - 0 — модуль может подключаться только к определенному командой AT+BIND Bluetooth-устройству;
  - 1 — модуль может подключаться к любому Bluetooth-устройству;
  - 2 — режим slave-loop.

- AT+INQM=<Param1>,<Param2>,<Param3> — установить параметры для запроса поиска Bluetooth-устройств.

  Параметры:

  - <Param1>:
    - 0 — стандартный режим запроса;
    - 1 — запрос в режиме RSSI;
  - <Param2> — максимальное количество устройств, отвечающих на запрос;
  - <Param3> — тайм-аут ожидания (1–48: от 1,28 до 61,44 сек).

  Ответ модуля: OK (или FAIL).

- AT+INQM? — получить параметры для запроса поиска Bluetooth-устройств.

  Параметров нет.

  Ответ модуля: +UART:<Param1>,<Param2>,<Param3>
- AT+INQ — запуск поиска Bluetooth-устройств.

  Параметров нет.

  Ответ модуля — список найденных устройств.
- AT+BIND=<Param> — привязать Bluetooth-модуль к другому модулю.

  Параметр: <Param> — адрес авторизованного Bluetooth-модуля.

  Ответ модуля: OK (или FAIL).
- AT+BIND? — получить адрес устройства, привязанного к Bluetooth-модулю.

  Параметров нет.

  Ответ модуля: <Param> — адрес устройства, привязанного к Bluetooth-модулю.
- AT+FSAD=<Param> — поиск авторизированного Bluetooth-устройства.

  Параметр: <Param> — адрес авторизованного Bluetooth-модуля.

  Ответ модуля: OK (или FAIL).
- AT+RMSAD=<Param> — удалить устройство из списка авторизированных для нашего Bluetooth-модуля.

  Параметр: <Param> — адрес авторизованного Bluetooth-модуля.

  Ответ модуля: OK (или FAIL).
- AT+RMAAD — очистить список авторизированных устройств для нашего Bluetooth-модуля.

  Параметр: <Param> — адрес авторизованного Bluetooth-модуля.

  Ответ модуля: OK (или FAIL).
- AT+LINK=<Param> — соединиться с Bluetooth-устройством.

  Параметр: <Param> — адрес Bluetooth-устройства.

  Ответ модуля: OK (или FAIL).

Полный список AT-команд можно найти в файле datasheets/BluetoothHC05.pdf сопровождающего книгу электронного архива.

24.3. Управление роботом с Android-устройства по Bluetooth

Создадим проект управления роботом с Android-устройства по Bluetooth. В качестве платформы использовалось шасси 4WD, изображенное на рис. 24.3. Электрическая схема устройства представлена на рис. 24.4.

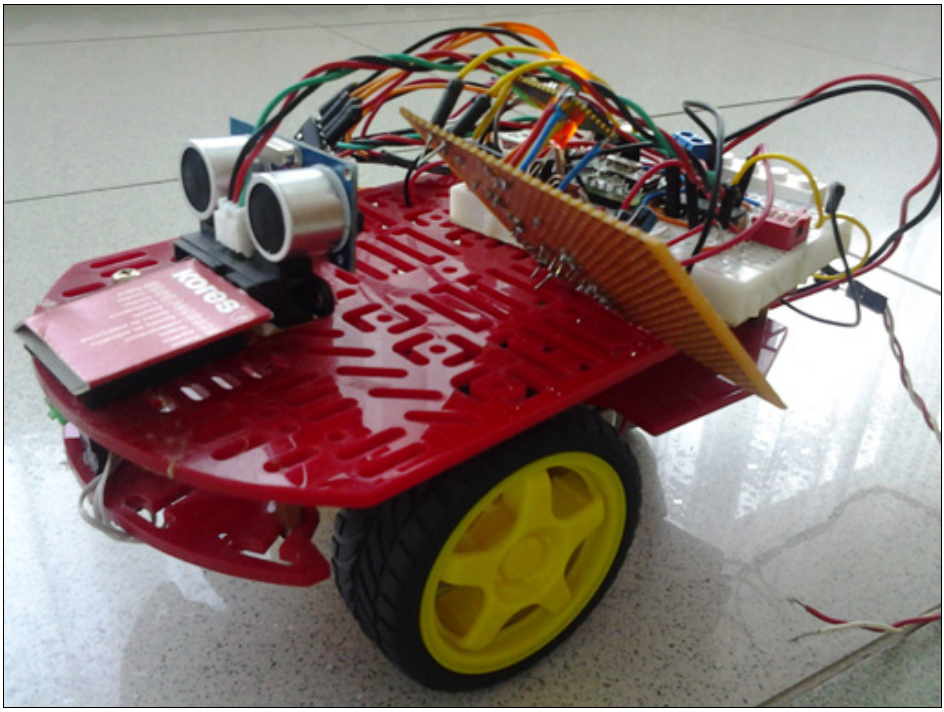


Рис. 24.3. Шасси для робота 4WD

Сначала необходимо запрограммировать модуль Bluetooth HC05 (см. *разд. 24.2*).

Прежде всего подключаемся по схеме, показанной на рис. 24.2. Затем переводим модуль в режим приема AT-команд, замкнув контакт PIO11 на +3,3 В. Отправляем тестовую команду:

```
AT\r\n
```

Ответ должен быть: `OK`. Теперь можно программировать.

Сначала даем команду сброса всех настроек:

```
AT+ORGL\r\n
```

Далее установим имя модуля:

```
AT+NAME=robot1\r\n
```

Установим пароль:

```
AT+PSWD=1111\r\n
```

Установим параметры UART:

```
AT+UART=38400,0,0\r\n
```

Переводим модуль в режим slave (ждет подключения):

```
AT+ROLE=0\r\n
```

Все. Переводим модуль HC05 в рабочий режим.

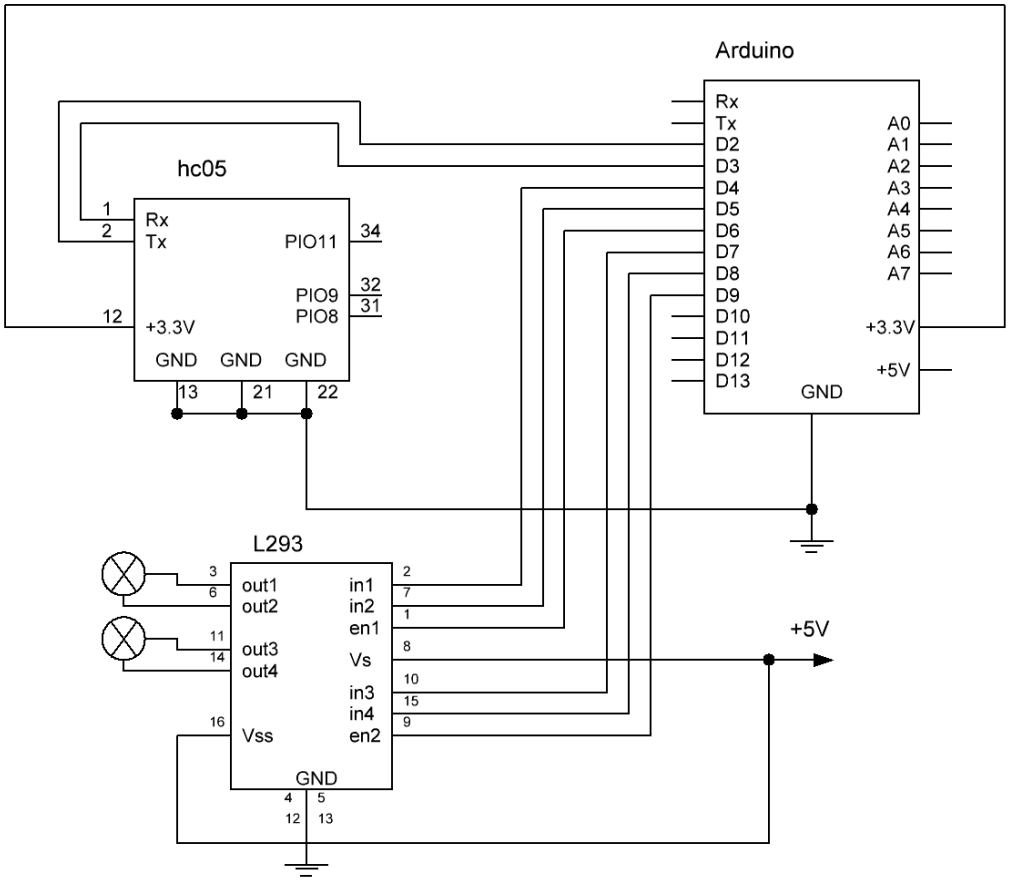


Рис. 24.4. Электрическая схема

Далее пишем скетч получения кодов из Android-устройства. За основу берем скетч из листинга 21.4 *главы 21*, описывающий управление гусеничным роботом с помощью инфракрасного пульта. Из Android-устройства для управления роботом будем передавать коды, аналогичные кодам пульта Marmitek, что позволит при необходимости перейти быстро на управление роботом по ИК. Код скетча представлен в листинге 24.2.

Листинг 24.2

```
// коды клавиш ИК-пульта (marmitek)
#define FORWARD 1936
#define BACK 3984
#define SPEED_UP 144   //ch+
#define SPEED_DOWN 2192  //ch-
#define LEFT 3472
#define RIGHT 1424
#define CIRCLE_LEFT 3216 //vol+
```

```
#define CIRCLE_RIGHT 1168 //vol-
#define STOP 2320     //0
#define EXT 16     //1
#define LOCAL 1040     //3

int local=0;
// состояние моторов робота
struct POZ  // структура для хранения статусов моторов
{
 int poz1;   // позиция для 1 мотора
 int poz2;   // позиция для 2 мотора
 int direction12;   // направление до поворота
};
// моторы
struct MOTOR  // структура для хранения номеров пинов, к которым подключены
моторчики
{
 int in1;   // INPUT1
 int in2;   // INPUT2
 int enable; // ENABLE1
};
// определяем порты, к которым подключены моторчики
MOTOR MOTOR1 = { 4, 5, 6 };
MOTOR MOTOR2 = { 7, 8, 9 };
// начальная позиция робота
// 1,2 мотор - выключены
POZ POZ12={5,5,5};
// массив возможных состояний моторов
// max down -> min down -> stop -> min up -> max up
int arrpoz[9][11]={
 // in1 первого мотора
 {0,0,0,0,0,0,1,1,1,1,1},
 // in2 первого мотора
 {1,1,1,1,1,0,0,0,0,0,0},
 // скорости (enable1) первого мотора
 {250,230,210,190,170,0,170,190,210,230,250},
 // in3 второго мотора
 {0,0,0,0,0,0,1,1,1,1,1},
 // in4 второго мотора
 {1,1,1,1,1,0,0,0,0,0,0},
 // скорости (enable2) второго мотора
 {250,230,210,190,170,0,170,190,210,230,250},
 // скорость++
 {0,-1,-1,-1,-1,0,1,1,1,1,0},
 // скорость--
 {0,-1,-1,-1,-1,0,1,1,1,1,0},
```

```
 // кружение влево-вправо
 {10,8,6,4,2,0,-2,-4,-6,-8,-10}
 };
//************** bluetootrh
#include <SoftwareSerial.h>
SoftwareSerial mySerial(2, 3); // указываем пины rx и tx соответственно
// rx-tx hc05 и tx-rx-rx hc05 соответственно
// для перевода строки в число
int pows[]={1,10,100,1000,10000};
int nums[10];
int index_nums=0;
int number=0;

void setup()
 {
 // последовательный порт
 pinMode(2,INPUT);
 pinMode(3,OUTPUT);
 Serial.begin(38400);
 mySerial.begin(38400);
 // настраиваем выводы для моторов
 pinMode(MOTOR1.in1, OUTPUT);
 pinMode(MOTOR1.in2, OUTPUT);
 pinMode(MOTOR2.in1, OUTPUT);
 pinMode(MOTOR2.in2, OUTPUT);
 // начальная позиция - робот стоит
 go12(POZ12.poz1,POZ12.poz2);
}
void loop()
 {
 // обработка кода нажатия
 while(mySerial.available())
 {
  int c = mySerial.read(); // читаем из software-порта
  if(c=='\n')
   {convert_to_number();
   index_nums=0;number=0;
   break;}
  else if(48<=int(c) && int(c)<=57)
   {
   nums[index_nums]=int(c)-48;
   index_nums++;
   }
  else
   {index_nums=0;number=0;
   Serial.println("error");
   break;}
  }
}
```

```
// получить число из массива цифр
void convert_to_number()
 {
 for(int i=index_nums;i>0;i--)
 //{number=number+nums[i-1]*10*(index_nums-i);}
 {number=number+nums[i-1]*pows[index_nums-i];
 }
 Serial.println(number,DEC);
 ir_go(number);
 }
// переустановка моторов
void go12(int poz1,int poz2)
 {
 digitalWrite(MOTOR1.in1, arrpoz[0][poz1]);
 digitalWrite(MOTOR1.in2, arrpoz[1][poz1]);
 analogWrite(MOTOR1.enable, arrpoz[2][poz1]);
 digitalWrite(MOTOR2.in1, arrpoz[3][poz2]);
 digitalWrite(MOTOR2.in2, arrpoz[4][poz2]);
 analogWrite(MOTOR2.enable, arrpoz[5][poz2]);
 }
// обработка кода нажатия
void ir_go(long kod)
 {
 Serial.print(kod);
 Serial.print("\n");
 if(kod!=EXT && local==1)
  return;
 switch(kod)
  {
  case LOCAL : // внешнее управление включить
    POZ12.direction12=10;
    POZ12.poz1=10;
    POZ12.poz2=10;
    go12(10,10);
   local=1;
   break;
  case EXT : // внешнее управление включить
   local=0;
   break;
  case FORWARD : // направление вперед
    POZ12.direction12=max(POZ12.direction12,10);
    POZ12.poz1=POZ12.direction12;
    POZ12.poz2=POZ12.direction12;
    go12(POZ12.poz1,POZ12.poz2);
   break;
  case BACK : // направление назад
    POZ12.direction12=min(POZ12.direction12,0);
```

```
   POZ12.poz1=POZ12.direction12;
   POZ12.poz2=POZ12.direction12;
   go12(POZ12.poz1,POZ12.poz2);
   break;
  case SPEED_UP : // скорость++
   POZ12.direction12=POZ12.direction12+arrpoz[6][POZ12.direction12];
   POZ12.poz1=POZ12.direction12;
   POZ12.poz2=POZ12.direction12;
   go12(POZ12.poz1,POZ12.poz2);
   break;
  case SPEED_DOWN : // скорость--
   POZ12.direction12=POZ12.direction12-arrpoz[7][POZ12.direction12];
   POZ12.poz1=POZ12.direction12;
   POZ12.poz2=POZ12.direction12;
   go12(POZ12.poz1,POZ12.poz2);
   break;
  case LEFT : // влево
   if(POZ12.direction12>6)
    {
    POZ12.poz1=POZ12.poz1-1;POZ12.poz1=max(POZ12.poz1,0);
    POZ12.poz2=POZ12.direction12;
    }
   else
    {
    POZ12.poz1=POZ12.poz1+1;POZ12.poz1=min(POZ12.poz1,14);
    POZ12.poz2=POZ12.direction12;
    }
   go12(POZ12.poz1,POZ12.poz2);
  break;
 case RIGHT : // вправо
   if(POZ12.direction12>6)
    {
    POZ12.poz2=POZ12.poz2-1;POZ12.poz2=max(POZ12.poz2,0);
    POZ12.poz1=POZ12.direction12;
    }
   else
    {
    POZ12.poz2=POZ12.poz2+1;POZ12.poz2=min(POZ12.poz2,14);
    POZ12.poz1=POZ12.direction12;
    }
   go12(POZ12.poz1,POZ12.poz2);
  break;
 case CIRCLE_RIGHT : // кружение вправо
   POZ12.direction12=10;
   POZ12.poz1=POZ12.direction12;
   POZ12.poz2=POZ12.direction12+arrpoz[8][POZ12.direction12];
   go12(POZ12.poz1,POZ12.poz2);
   break;
```

```
  case CIRCLE_LEFT : // кружение влево
    POZ12.direction12=0;
    POZ12.poz1=POZ12.direction12+arrpoz[8][POZ12.direction12];
    POZ12.poz2=POZ12.direction12;
    go12(POZ12.poz1,POZ12.poz2);
    break;
  case STOP : // стоп
    POZ12.poz1=5;
    POZ12.poz2=5;
    POZ12.direction12=5;
    go12(POZ12.poz1,POZ12.poz2);
    break;
  default:
    break;
  }
}
```

Данный скетч можно найти в папке examples/_24_1 сопровождающего книгу электронного архива.

Теперь необходимо написать приложение для Android. Внешний вид приложения приведен на рис. 24.5. При выборе пункта **Подключиться к роботу** происходит поиск Bluetooth-устройств (рис. 24.6). Выбираем наше устройство и попадаем в интерфейс управления роботом (рис. 24.7).

Архив с файлами проекта для среды Eclipse находится в файле examples/Android/_24_1.rar сопровождающего книгу электронного архива.

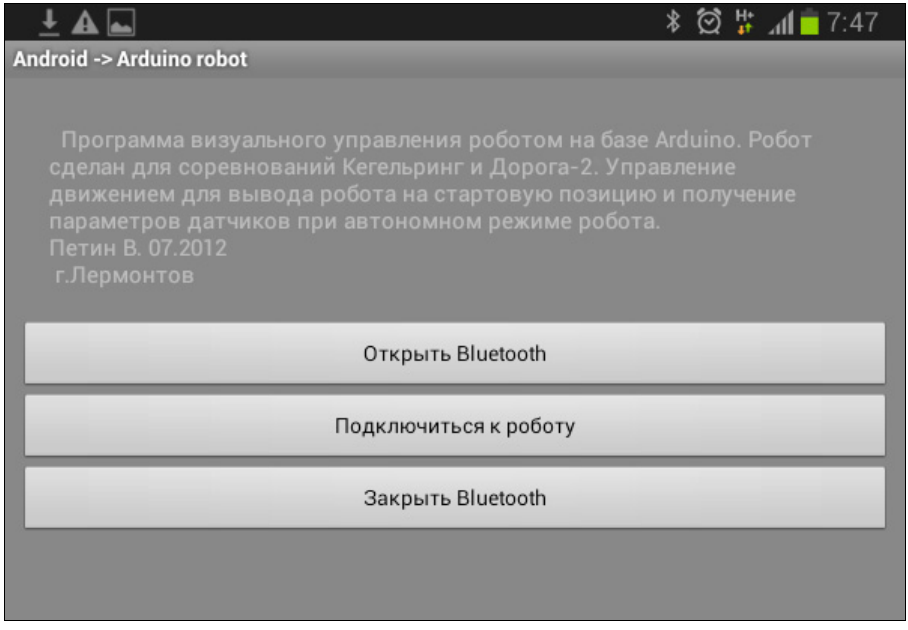


Рис. 24.5. Приложения Android для связи с роботом по Bluetooth

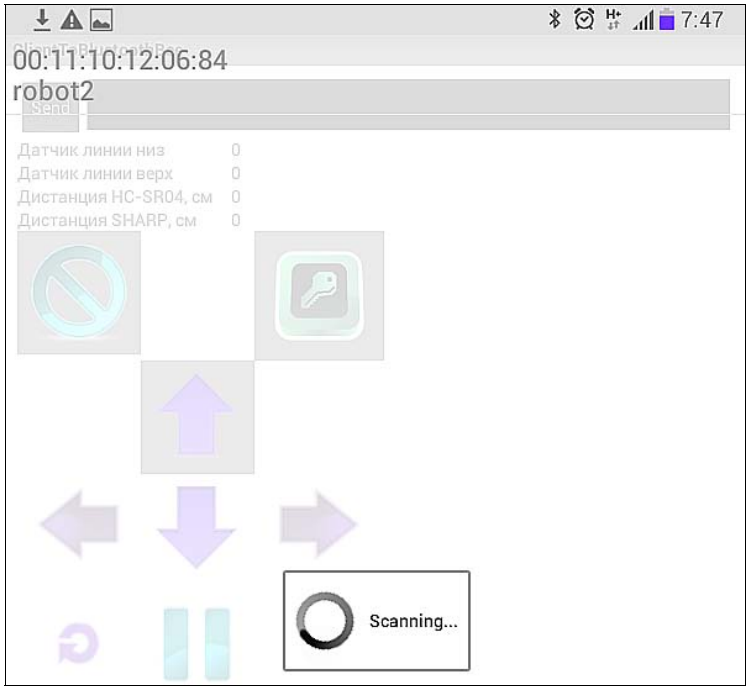


Рис. 24.6. Поиск Bluetooth-устройств

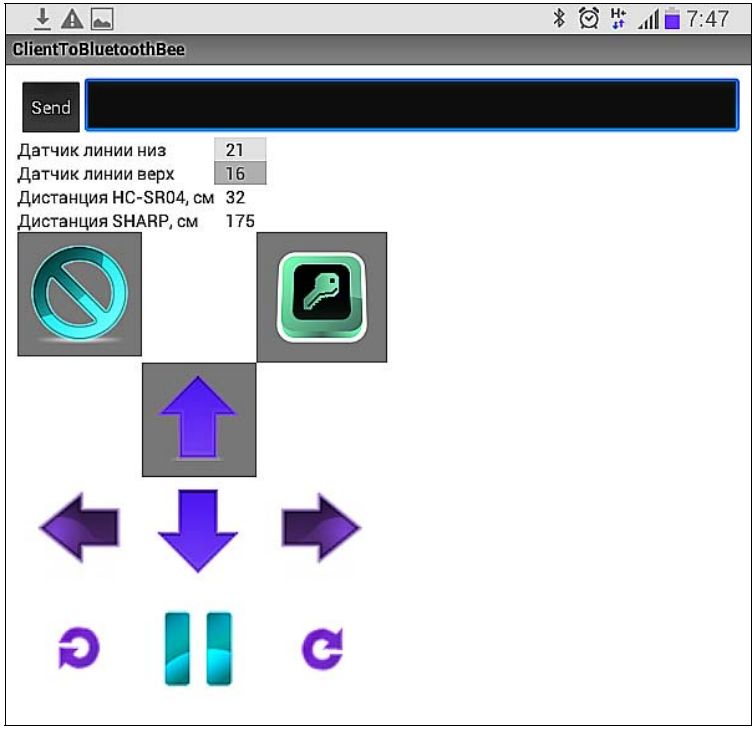


Рис. 24.7. Интерфейс управления роботом

ГЛАВА 25

TV-выход на Arduino

В этой главе мы рассмотрим, как можно использовать плату Arduino для вывода какой-либо информации на ТВ по НЧ-кабелю ("тюльпан"). Правда, изображение окажется черно-белым, однако и этого будет достаточно для большинства проектов. Для закрепления материала подключим к Arduino джойстик и создадим простенькую игру, используя в качестве устройства вывода обычный телевизор.

25.1. Схема подключения

Схема переходника (рис. 25.1) — простейшая.

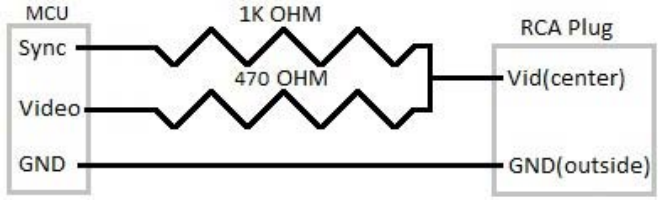


Рис. 25.1. Схема переходника

25.2. Arduino-библиотека *TVOut*

Для генерации монохромного композитного видеосигнала используется готовая программная библиотека `TVOut` для Arduino. Библиотеку можно скачать со страницы **http://code.google.com/p/arduino-tvout/downloads/list**. Библиотека совместима со следующими платами Arduino: Decimilia, Nano, Uno. Mega. Список используемых выводов для данных плат представлен в табл. 25.1.

Библиотека предоставляет большое количество функций для вывода графических примитивов, текста, доступа к отдельным точкам изображения телевизора, а также простейшие функции для вывода звука по аудиокабелю. По умолчанию рабочее разрешение устанавливается в 128×96 точек. Стоит отметить также, что не гаран-

тируется корректная работа данной библиотеки при ее применении в другой интегрированной среде разработки.

Для начала работы необходимо создать экземпляр класса `TVOut`, например:

```
TVOut TV;
```

Таблица 25.1. *Список выводов, используемых для плат Arduino*

Плата Arduino	SYNC	VIDEO	AUDIO
Decimilia, Uno, Nano	9	7	11
Mega	11	A7(D29)	10

Рассмотрим некоторые функции, предоставляемые классом `TVOut` для работы с изображением экрана.

25.2.1. Функция установки режима *begin()*

Функция `begin()` инициализирует вывод видеосигнала (разрешение экрана по умолчанию 128×96 точек).

Синтаксис:

```
TVOut.begin(mode)
TV.begin(mode, x, y)
```

Параметр: `mode` — стандарт видеосигнала:

- `_PAL` — режим PAL;
- `_NTSC` — режим NTSC.

Возвращаемые значения: `0` — в случае удачного соединения, `4` — в случае неудачи (недостаточно памяти для буфера вывода).

25.2.2. Функции задержки

Функция *delay()*

Функция `delay()` осуществляет задержку выведенного изображения.

Синтаксис:

```
TVOut.delay(ms)
```

Параметр: `ms` — задержка в мс с точностью: 20 мс для PAL и 16 мс для NTSC.

Функция *delay_frame()*

Функция `delay_frame()` осуществляет задержку выведенного изображения.

Синтаксис:

```
TVOut.delay_frame(frames)
```

Параметр: `frames` — количество кадров для задержки.

Функция полезна для сведения к минимуму или устранения мерцания экрана, вызванного его обновлением.

25.2.3. Функции получения параметров

Функция *hres()*

Функция `hres()` возвращает горизонтальное разрешение экрана.

Синтаксис:

```
TVOut.hres()
```

Параметров нет.

Возвращаемое значение: `unsigned char` — горизонтальное разрешение экрана.

Функция *vres()*

Функция `vres()` возвращает вертикальное разрешение экрана.

Синтаксис:

```
TVOut.vres()
```

Параметров нет.

Возвращаемое значение: `unsigned char` — вертикальное разрешение экрана.

Функция *char_line()*

Функция `char_line()` возвращает максимально возможное количество символов в одной строке при выводе текстовой информации.

Синтаксис:

```
TVOut. char_line()
```

Параметров нет.

Возвращаемое значение: `unsigned char` — количество символов.

25.2.4. Основные графические функции

Функция *set_pixel()*

Функция `set_pixel()` устанавливает цвет пиксела экрана в точке с заданными координатами.

Синтаксис:

```
TVOut.set_pixel(x,y,color)
```

Параметры:

- `x,y` — координаты пиксела;
- `color` — цвет пиксела:
  - `0` — черный;
  - `1` — белый;
  - `2` — инвертировать цвет.

Функция *get_pixel()*

Функция `get_pixel()` получает цвет пиксела экрана из точки с заданными координатами.

Синтаксис:

```
TVOut.get_pixel(x,y)
```

Параметры: `x,y` — координаты пиксела.

Возвращаемое значение:

- `color` — цвет пиксела:
  - `0` — черный;
  - `1` — белый;
  - `2` — инвертировать цвет.

Функция *fill()*

Функция `fill()` заполняет экран заданным цветом.

Синтаксис:

```
TVOut.fill(color)
```

Параметр: `color` — цвет заполнения:

- `0` — черный;
- `1` — белый;
- `2` — инвертировать цвет.

Функция *clear_screen()*

Функция `clear_screen()` очищает экран, заполняя его заданным цветом.

Синтаксис:

```
TVOut.clear_screen(color)
```

Параметр: `color` — цвет заполнения:

- `0` — черный;
- `1` — белый;
- `2` — инвертировать цвет.

Функция *invert()*

Функция `invert()` инвертирует содержимое экрана.

Синтаксис:

```
TVOut.invert()
```

Параметров нет.

Функция *shift_direction()*

Функция `shift_direction()` сдвигает содержимое экрана.

Синтаксис:

```
TVOut.shift_direction(distance, direction)
```

Параметры:

- `distance` — расстояние для сдвига содержимого экрана;
- `direction` — направление сдвига:
  - `UP`=0 — вверх;
  - `DOWN`=1 — вниз;
  - `LEFT`=2 — влево;
  - `RIGHT`=3 — вправо.

Функция *draw_line()*

Функция `draw_line()` соединяет на экране линией две точки.

Синтаксис:

```
TVOut.draw_line(x0,y0,x1,y1,color)
```

Параметры:

- `x0,y0` — координаты первой точки;
- `x1,y1` — координаты второй точки;
- `color` — цвет заполнения:
  - `0` — черный;
  - `1` — белый;
  - `2` — инвертировать цвет.

Функция *draw_row()*

Функция `draw_row()` заполняет указанным цветом строку между двумя точками строки.

Синтаксис:

```
TVOut.draw_row(row,x0,x1,color)
```

Параметры:

- `row` — вертикальная координата строки;
- `x1,x2` — горизонтальные координаты точек строки;
- `color` — цвет заполнения:
  - `0` — черный;
  - `1` — белый;
  - `2` — инвертировать цвет.

Функция *draw_column()*

Функция `draw_column()` заполняет указанным цветом строку между двумя точками столбца.

Синтаксис:

```
TVOut.draw_column(column,y0,y1,color)
```

Параметры:

- `column` — горизонтальная координата столбца;
- `y1,y2` — вертикальные координаты точек столбца;
- `color` — цвет заполнения:
  - `0` — черный;
  - `1` — белый;
  - `2` — инвертировать цвет.

Функция *draw_rect()*

Функция `draw_rect()` рисует на экране прямоугольник.

Синтаксис:

```
TVOut.draw_rect(x,y,w,h,color)
TVOut.draw_rect(x,y,w,h,color,fillcolor)
```

Параметры:

- `x,y` — координаты левой верхней точки;
- `w,h` — ширина и высота рисуемого прямоугольника;
- `color` — цвет границ прямоугольника:
  - `0` — черный;
  - `1` — белый;
  - `2` — инвертировать цвет;
- `fillcolor` — цвет заполнения прямоугольника:
  - `0` — черный;

- `1` — белый;
- `2` — инвертировать цвет.

Функция *draw_circle()*

Функция `draw_circle()` рисует на экране круг.

Синтаксис:

```
TVOut.draw_ circle(x,y,r,color)
TVOut.draw_ circle(x,y,r,color,fillcolor)
```

Параметры:

- `x,y` — координаты центра круга;
- `r` — радиус круга;
- `color` — цвет границ круга:
  - `0` — черный;
  - `1` — белый;
  - `2` — инвертировать цвет;
- `fillcolor` — цвет заполнения круга:
  - `0` — черный;
  - `1` — белый;
  - `2` — инвертировать цвет.

Функция *bitmap()*

Функция `bitmap()` выводит на экран растровое изображение.

Синтаксис:

```
TVOut.bitmap(x,y,bmp,w,h)
```

Параметры:

- `x,y` — координаты левого верхнего угла точки вывода;
- `bmp` — указатель на массив памяти, где хранится картинка;
- `w,h` — ширина, высота выводимого изображения;

В *разд. 25.4* мы рассмотрим процесс создания кода выводимых растровых изображений.

25.2.5. Функции вывода текстовой информации

Для применения функций вывода текстовой информации требуется подключение файлов с включенными в библиотеку пользовательскими шрифтами. Для подключения пользовательского набора шрифтов необходимо в скетче подключить заголовочный файл:

```
#include <fontALL.h>
```

В состав библиотеки включены следующие наборы шрифтов:

- `font4x6`;
- `font6x8`;
- `font8x8`;
- `font8x8ext`.

Создание пользовательских шрифтов мы рассмотрим в *разд. 25.5*.

Функция *select_font()*

Функция `select_font()` выбирает шрифт для вывода текстовой информации.

Синтаксис:

```
TVOut.select_font(font)
```

Параметр: `font` — шрифт, подключенный в скетче.

Функция *print_char()*

Функция `print_char()` выводит символ на экран.

Синтаксис:

```
TVOut.print_char(x,y,char)
```

Параметры:

- `x,y` — позиция на экране для вывода символа;
- `char` — символ из текущего шрифта.

Функция *set_cursor()*

Функция `set_cursor()` устанавливает позицию курсора для вывода текстовой информации на экран.

Синтаксис:

```
TVOut.set_cursor(x,y)
```

Параметры: `x,y` — координаты для курсора.

Функция *print()*

Функция `print()` выводит на экран строку, символ или число.

Синтаксис:

```
TVOut.print(x,y,string)
TVOut.print(x,y,char,base)
TVOut.print(x,y,int,base)
```

Параметры:

- `x,y` — координаты курсора;

- `base` — формат вывода:
  - `BYTE` = 0;
  - `DEC` = 10 (default);
  - `HEX` = 16.

Функция *println()*

Функция `println()` выводит на экран строку, символ или число и в конце символ перевода строки.

Синтаксис:

```
TVOut.println(x,y,string)
TVOut.println(x,y,char,base)
TVOut.println(x,y,int,base)
```

Параметры:

- `x,y` — координаты курсора;
- `base` — формат вывода:
  - `BYTE` = 0;
  - `DEC` = 10 (default);
  - `HEX` = 16.

25.2.6. Функции вывода аудио

Функции вывода звука позволяют отправлять на телевизор через аудиовыход сигнал определенной частоты.

Функция *tone()*

Функция `tone()` выдает аудиосигнал определенной частоты.

Синтаксис:

```
TVOut.tone(frequency,duration)
TVOut.tone(frequency)
```

Параметры:

- `frequency` — частота аудиосигнала;
- `duration` — длительность сигнала.

Функция *noTone()*

Функция `noTone()` прекращает выдачу аудиосигнала.

Синтаксис:

```
TVOut.noTone()
```

25.3. Создание пользовательских шрифтов

Рассмотрим процесс создания пользовательских шрифтов для библиотеки `TVOut`. Создадим пользовательский шрифт `myfont1`. Для пользовательского шрифта `myfont1` в папке TVOutfonts создаем 2 файла: myfont1.h и myfont1.cpp. Содержимое файла myfont1.h представлено в листинге 25.1.

Листинг 25.1

```
#ifndef MYFONT1_h
#define MYFONT1_h
#include <avr/pgmspace.h>
extern const unsigned char myfont1[];
#endif
```

Существуют два вида шрифтов: фиксированной и переменной ширины. Для шрифтов фиксированной ширины первые три байта массива содержат данные о ширине символа (4), высоте символа (6) и первый печатный символ (32). Затем идут данные для каждого последующего символа (листинг 25.2).

Листинг 25.2

```
#include "myfont1.h"

PROGMEM const unsigned char myfont1[] = {
4,6,32,
//
0b00000000,
0b00000000,
0b00000000,
0b00000000,
0b00000000,
0b00000000,
//
0b01000000,
0b01000000,
0b01000000,
0b00000000,
0b01000000,
0b00000000,
... ...
};
```

Для шрифтов переменной ширины в описании каждого символа первый байт определяет ширину данного символа (листинг 25.3).

Листинг 25.3

```
#include "myfont1.h"

PROGMEM const unsigned char myfont1[] = {
4,6,32,
//
2,
0b00000000,
0b00000000,
0b00000000,
0b00000000,
0b00000000,
0b00000000,
//
3,
0b01000000,
0b01000000,
0b01000000,
0b00000000,
0b01000000,
0b00000000,
... ...
};
```

Теперь для использования библиотекой `TVOut` нашего пользовательского шрифта `myfont1` в скетче необходимо перенести в папку libraries Arduino IDE и подключить файл myfont1.h:

```
#include "myfont1.h"
```

25.4. Создание графических примитивов

Библиотека `TVOut` позволяет загружать на экран растровые изображения. Рассмотрим создание кода для загрузки растрового изображения функцией `bitmap()` библиотеки `TVOut`.

Сначала необходимо создать однобитное (двухцветное) изображение — например, в графическом редакторе Paint (рис. 25.2).

Затем нам понадобится программа Image2Code, которая сконвертирует из нашего изображения код. Программу в версии для операционной системы Windows можно скачать по адресу **http://sourceforge.net/projects/image2code/files/**. Скачиваем, запускаем (рис. 25.3).

Нажимаем на кнопку **Convert** и получаем массив (рис. 25.4).

Далее создаем два файла. Первый с расширением h — например, MyBitmap1.h (листинг 25.4).

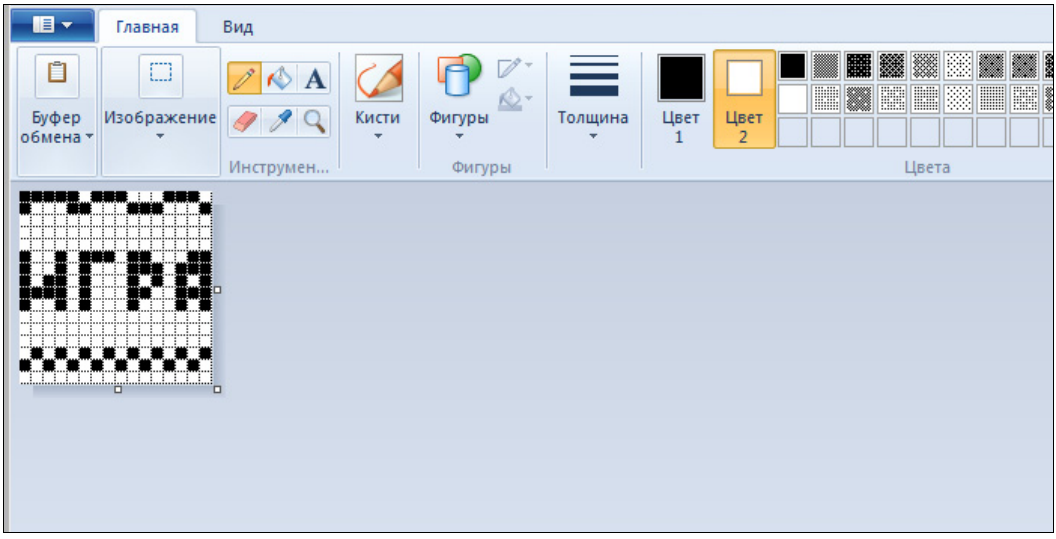


Рис. 25.2. Рисунок в редакторе Paint

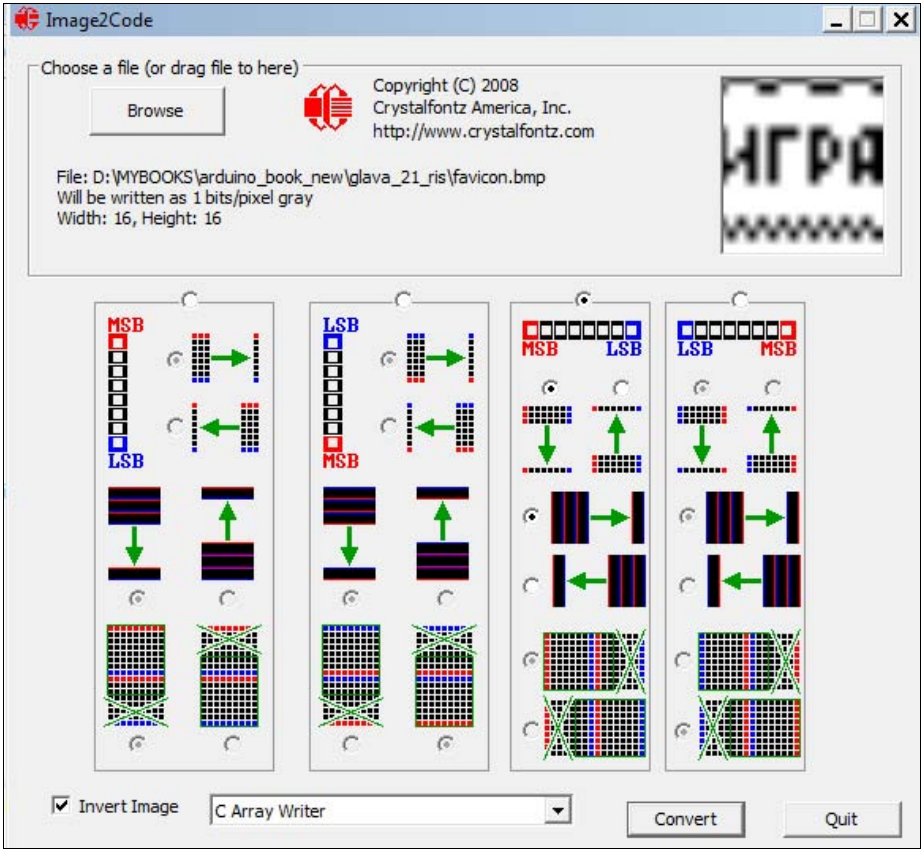


Рис. 25.3. Окно программы Image2Code

Рис. 25.4. Результат конвертации изображения программой Image2Code

Листинг 25.4

```
# include <avr/pgmspace.h>
# ifndef MYBITMAP1_H
# define MYBITMAP1_H
extern const unsigned char MyBitmap1[];
#endif
```

Затем создаем второй файл MyBitmap1.cpp (листинг 25.5), в него (в поле данных массива `unsigned char MyBitmap1[]`) копируем данные конвертации, убирая символы "{" и "}" и вставляя в начале ширину и высоту изображения в пикселах (листинг 25.6).

Листинг 25.5

```
#include " MyBitmap1.h"
PROGMEM const unsigned char MyBitmap1[] = {};
```

Листинг 25.6

```
#include " MyBitmap1.h"
PROGMEM const unsigned char MyBitmap1[] = {
16,16,
0x73,0x8E,
0x8C,0x71,
0x00,0x00,
0x00,0x00,
```

```
0x00,0x00,
0x97,0x63,
0x94,0x55,
0xB4,0x55,
0xD4,0x67,
0x94,0x45,
0x00,0x00,
0x00,0x00,
0x00,0x00,
0xAA,0xAA,
0x55,0x55,
0x00,0x00
};
```

Сохраняем файлы MyBitmap1.h и myBitmap.cpp в папку нашего скетча. Для вывода этого изображения на экран телевизора вызываем функцию `TVOut.bitmap()`:

```
#include " MyBitmap1.h"
TVOut.bitmap(x,y,MyBitmap);
```

25.5. Создание простейшей игровой консоли

Соберем простейшую игровую консоль. Нам потребуется переходник "тюльпан", перепаянный согласно схеме, приведенной на рис. 25.1, плата Arduino (в нашем случае Nano) и простейший двухкоординатный джойстик (рис. 25.5).

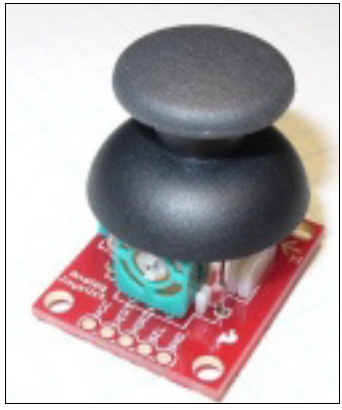


Рис. 25.5. Двухкоординатный джойстик

Выходы джойстика подключаем к аналоговым входам Arduino A0 и A1. Кабель-"тюльпан" подключаем в выводам 7, 9, 11 Arduino. Схема соединений представлена на рис. 25.6, консоль в сборе — на рис. 25.7.

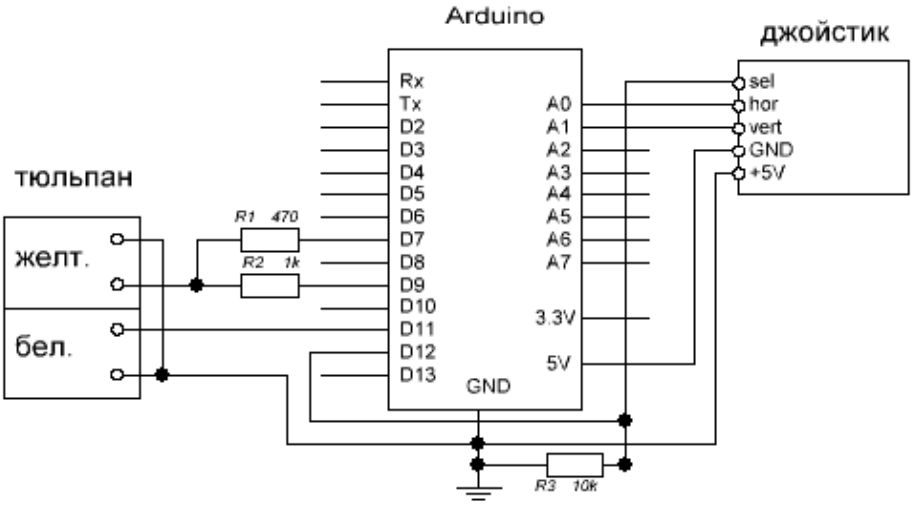


Рис. 25.6. Схема игровой консоли

Рис. 25.7. Внешний вид консоли

25.6. Разработка игры для игровой консоли

Приступим к созданию игры с условным названием "Арифметический сборщик". "Игрок" управляется джойстиком с возможностью перемещения по полю размером 128×90 точек. При нулевых отклонениях джойстика "игрок" находится в центре. Максимальное отклонение джойстика соответствует максимальному перемещению "игрока". С определенным интервалом времени генерируются объекты-цифры, которые движутся сверху вниз. По достижении нижнего положения экрана объект-цифра исчезает, уменьшая количество баллов "игрока" на величину данной цифры. Если "игрок" перехватывает объект-цифру на экране, то это приводит к исчезновению объекта-цифры и увеличению счетчика баллов "игрока". С определенной периодичностью значок "игрока" меняет свое значение с цифры 0 до 9. При перехвате объекта-цифры счетчик баллов "игрока" увеличивается на сумму, равную сумме объекта-цифры и "цифры" игрока. Если цифра "игрока" равняется цифре объекта-цифры, то счетчик баллов "игрока" увеличивается на произведение цифр. По достижении определенных порогов баллов происходит переход на более высокий уровень игры, что приводит к увеличению скорости движения и скорости генерации объектов-цифр. Кроме того, с 4-го уровня игры столкновение "игрока" с объектом-цифрой приводит не только к увеличению счетчика "игрока", но и к его уменьшению — если цифра "игрока" меньше цифры объекта-цифры. Если количество баллов "игрока" становится меньше 0, игра начинается сначала.

25.6.1. Создание переменных игры

Для управления игрой создадим объекты для хранения ее текущего положения. Символ, отображающий "игрока", и символы, отображающие объекты-цифры, выводятся как текстовая информация, поэтому поделим все поле игры на строки и все перемещения объектов будем осуществлять как вывод символа в знакоместо на поле. Переменные `MAX_X`=31 и `MAX_Y`=14 определяют размер поля игры по количеству знакомест по горизонтали и вертикали. Переменная `MAX_OBJ`=30 определяет максимальное количество одновременно находящихся на поле игры объектов-цифр. Массив `int FIGURA[30][3]` хранит информацию об объектах-цифрах, находящихся на поле игры следующим образом:

- `FIGURA[i][0]` — числовое значение объекта-цифры (0 — пустой объект);
- `FIGURA[i][1]` — текущая координата `x` объекта-цифры;
- `FIGURA[i][2]` — текущая координата `y` объекта-цифры.

Для хранения прочих переменных, описывающих текущее состояние игры, создадим структуру `GAME`. Список полей структуры:

- `xk` — координата `x`(/4) "игрока";
- `yk` — координата `y`(/6) "игрока";
- `tekCursor` — текущее значение курсора;
- `blinkCursor` — текущее состояние блинка (мерцания) курсора;
- `vblink` — скорость блинка (мерцания) в `vk`;

- `vk` — скорость движения "игрока" (проверка входов A0, A1);
- `vo_10` — скорость изменения цифры "игрока";
- `vo_11` — скорость появления объектов-цифр;
- `vo_12` — скорость движения объектов-цифр;
- `count_objects` — кол-во объектов-цифр на поле;
- `level` — уровень игры;
- `balls` — кол-во баллов.

Код структуры `GAME` представлен в листинге 25.7.

Листинг 25.7

```
int MAX_X=31;
int MAX_Y=14;
// структура данных игры
struct GAME // структура для данных игры
{
 int xk;   // координата x(/4) игрока
 int yk;   // координата y(/6) игрока
 int tekCursor; // текущее значение курсора
 int blinkCursor; // текущее состояние блинка курсора
 int vblink; // скорость blink в vk

 long vk;   // скорость движения игрока - проверка входов A0,A1
 long vo_10;  // скорость изменения цифры игрока
 long vo_11;  // скорость появления цифр
 long vo_12;  // скорость движения цифр

 int count_objects; // кол-во объектов на поле
 int level; // уровень игры
 int balls;   // кол-во баллов
};
int MAX_OBJ=30;
int FIGURA[30][3]={{0,0,0},{0,0,0},{0,0,0},{0,0,0},{0,0,0},
         {0,0,0},{0,0,0},{0,0,0},{0,0,0},{0,0,0},
         {0,0,0},{0,0,0},{0,0,0},{0,0,0},{0,0,0},
         {0,0,0},{0,0,0},{0,0,0},{0,0,0},{0,0,0},
         {0,0,0},{0,0,0},{0,0,0},{0,0,0},{0,0,0},
         {0,0,0},{0,0,0},{0,0,0},{0,0,0},{0,0,0}
         };
```

25.6.2. Управление положением "игрока" с помощью джойстика

Положение "игрока" на экране определяется отклонением джойстика. Выводы джойстика подключены к аналоговым портам A0 и A1 платы Arduino. Опрос пор-

тов происходит через время, определенное параметром GAME.vk, эти данные обрабатываются функцией map(), которая пропорционально переносит значение из текущего диапазона 0–124 в новый диапазон (значения ширины и высоты экрана). Затем это значение переводится в координаты знакоместа. В это знакоместо необходимо переместить изображение символа "игрока", предварительно поместив в предыдущее положение "игрока" символ пробела. Для отображения "игрока" используется мигающий символ — цифра и пробел. Все эти операции производит функция set_iam(), содержимое которой приведено в листинге 25.8.

Листинг 25.8

```
//****************** установка нового положения игрока
void set_iam()
 {
 TV.set_cursor(min(123,GAME1.xk*4),min(84,GAME1.yk*6));
 TV.print(" ");

 GAME1.xk=map(analogRead(A0), 0, 1024, 0, 128);
 GAME1.yk=map(analogRead(A1), 0, 1024, 0, 96);
 GAME1.xk=GAME1.xk/4;
 GAME1.yk=GAME1.yk/6;
 GAME1.vblink--;
 if(GAME1.vblink<0)
  {
  GAME1.blinkCursor=1-GAME1.blinkCursor;
  GAME1.vblink=5+GAME1.blinkCursor*5;
  }
 TV.set_cursor(min(123,GAME1.xk*4),min(84,GAME1.yk*6));
 if(GAME1.blinkCursor==1)
  TV.print(GAME1.tekCursor);
 else
  TV.print(" ");
 }
```

Символ, отображающий "игрока", меняется через время, определенное параметром GAME.vo_10, вызовом функции set_new_cursor(), изменение происходит случайным образом с помощью функции random(). Содержимое функции set_new_cursor() представлено в листинге 25.9.

Листинг 25.9

```
//****************** установка нового вида символа игрока
void set_new_cursor()
 {
 GAME1.tekCursor=random(0,10);
 }
```

25.6.3. Генерация и перемещение объектов-цифр

Объекты-цифры генерируются через время, определенное параметром GAME.vo11. Информация обо всех объектах-цифрах хранится в массиве FIGURA[30][3]. Вызывается функция set_new_object(). Программа ищет первый пустой индекс в массиве (FIGURA[30][3]=0) и помещает в него новый объект-цифру. Цифровое значение и горизонтальная координата нового объекта генерируются функцией random, а вертикальная координата устанавливается равной нулю. Символ, отображающий новый объект-цифру, выводится на экран. Содержимое функции set_new_object() представлено в листинге 25.10.

Листинг 25.10

```
//****************** появление нового объекта-цифры
void set_new_object()
 {
 int null_index=0;
 if(GAME1.count_objects<MAX_OBJ)
  {
  for(int i=0;i<MAX_OBJ;i++)
   {
   if(FIGURA[i][0]==0)
    {null_index=i;break;}
   }
  FIGURA[null_index][0]=random(1,9);
  FIGURA[null_index][1]=random(0,MAX_X);
  FIGURA[null_index][2]=0;
  // вывод на доску
  TV.set_cursor(FIGURA[null_index][1]*4,0);
  TV.print(FIGURA[null_index][0]);

  GAME1.count_objects++;
 }
 }
```

Функция движения объектов-цифр go_object() вызывается через время, определенное параметром GAME.vo12. Визуализация движения объектов-цифр происходит путем стирания символа из предыдущего положения объекта (запись символа пробела) и записи символа объекта в новое положение (вертикальная координата знакоместа объекта-цифры увеличивается на единицу). По достижении низа экрана происходит удаление объекта-цифры (запись 0 в элемент массива FIGURA[i][0]), а также уменьшение количества баллов игрока. Содержимое функции go_object() представлено в листинге 25.11.

Листинг 25.11

```
//****************** движение объекта-цифры
void go_object()
 {
 for(int i=0;i<MAX_OBJ;i++)
  {
  if(FIGURA[i][0]>0)
   {
   TV.set_cursor(FIGURA[i][1]*4,FIGURA[i][2]*6);
   TV.print(" ");
   if(FIGURA[i][2]<MAX_Y)
    {
    FIGURA[i][2]++;
    TV.set_cursor(FIGURA[i][1]*4,FIGURA[i][2]*6);
    TV.print(FIGURA[i][0]);
    }
   else
    {
    TV.tone(294,200);
    change_balls(FIGURA[i][0]*(-1));
    FIGURA[i][0]=0;
    GAME1.count_objects--;
    }
   }
  }
 }
```

25.6.4. Проверка столкновения "игрока" и объектов-цифр

При перемещении символа "игрока" по экрану необходимо проверять его столкновения с объектами-цифрами. Для этого используем функцию `collision()`. После установки символа, соответствующего "игроку", проверяем элементы массива `FIGURA` на соответствие координат объектов-цифр координатам символа "игрока". При совпадении координат выполняем следующие действия:

1. Уничтожается объект-цифра из массива `FIGURA` (запись `0` в `FIGURA[[i][0]`).
2. Уменьшается на 1 счетчик количества объектов-цифр (`GAME.count_objects`).
3. Производится изменение счетчика количества баллов "игрока" (вызов функции `change_balls()`).

Содержимое функции `collision()` представлено в листинге 25.12.

Листинг 25.12

```
//****************** проверка столкновения
void collision()
{
for(int i=0;i<MAX_OBJ;i++)
{
if(FIGURA[i][0]>0)
 {
 if(FIGURA[i][1]==GAME1.xk && FIGURA[i][2]==GAME1.yk)
 {
 TV.tone(740,200);
 if(FIGURA[i][0]==GAME1.tekCursor)
  change_balls(GAME1.tekCursor*GAME1.tekCursor);
 else if(FIGURA[i][0]>GAME1.tekCursor && GAME1.level>3)
  change_balls(FIGURA[i][0]*(-1));
 else
  change_balls(FIGURA[i][0]+GAME1.tekCursor);
 FIGURA[i][0]=0;
 GAME1.count_objects--;
 }
 }
}
}
```

В функцию изменения счетчика количества баллов "игрока" `change_balls()` в качестве аргумента передается число, на которое следует увеличить (уменьшить) счетчик баллов "игрока". Это число равно сумме объекта-цифры и цифры "игрока". Если цифра "игрока" равняется цифре объекта-цифры, то передается значение, равное произведению цифр. Как отмечалось ранее, для повышения сложности игры с 4-го уровня столкновение "игрока" с объектом-цифрой приводит не только к увеличению счетчика "игрока", но и к уменьшению его — если цифра "игрока" меньше цифры объекта-цифры.

25.6.5. Счетчик баллов "игрока"

Функция `change_balls()` производит изменение счетчика баллов "игрока" на величину входящего аргумента. Если счетчик количества баллов становится меньше 0, игра заканчивается, экран очищается и осуществляется переход на начало игры. Кроме того, при достижении счетчиком определенных значений, выполняется переход игры на новый, более высокий, уровень. Установку значений для нового уровня выполняет функция `new_level()`. Содержимое функции `change_balls()` представлено в листинге 25.13.

Листинг 25.13

```
//****************** изменение набранных баллов
void change_balls(int ball)
 {
 GAME1.balls=GAME1.balls+ball;
 if(GAME1.balls<0)
  start_game();
 if(GAME1.balls>(GAME1.level+1)*100)
  new_level(GAME1.level);
 set_data_tablo();
 }
```

25.6.6. Переход на новый уровень

При переходе на новый уровень игры осуществляется установка новых значений для скорости генерации и скорости перемещения объектов-цифр. Первоначально предполагалось менять и значение скорости изменения цифры "игрока" и скорости его движения, но в ходе испытания игры решено было от этого отказаться. Действия установки значений переменных для нового уровня игры производятся в функции `new_level()`, содержимое которой представлено в листинге 25.14.

Листинг 25.14

```
//****************** изменение уровня игры
void new_level(int tek_level)
 {
 GAME1.level++;
 GAME1.vo_10=5000;
 GAME1.vo_11=2000-(GAME1.level-1)*100;
 GAME1.vo_12=1000-(GAME1.level-1)*100;
 }
```

25.6.7. Отображение данных игры на табло

Табло с данными игры находится внизу экрана. Здесь отображаются текущие значения количества набранных баллов и уровень игры. При изменении счетчика баллов "игрока" происходит вызов функции `set_data_tablo()`, которая изменяет значение количества набранных баллов и уровень игры в переменных структуры `GAME` и на табло. Содержимое функции `set_data_tablo()` представлено в листинге 25.15.

Листинг 25.15

```
//****************** вывод данных на табло
void set_data_tablo()
 {
 TV.print(20,91," balls=  ");
```

```
 TV.print(70,91," level= ");
 TV.print(48,91,GAME1.balls);
 TV.print(98,91,GAME1.level);
 }
```

25.6.8. Звуковое сопровождение игры

Для звукового оформления игры мы используем функцию `tone(frequency, duration)` библиотеки `TVOut`. При достижении объектом-цифрой низа игрового поля (функция `go_object()`) воспроизводится звук `TV.tone(294,200)`, при столкновении игрока и объекта-цифры (функция `collision()`) — звук `TV.tone(740,200)`. Если вы захотите в проект вставить фрагмент из последовательного воспроизведения нескольких нот, после вывода каждой ноты командой `tone(frequency, duration)` необходимо вставлять командой `delay()` задержку длительностью не меньшей, чем параметр `duration`. Так, в листинге 25.16 организовано последовательное воспроизведение двух нот с паузой.

Листинг 25.16

```
TV.tone(294,200);
TV.delay(400);
TV.tone(740,200);
```

25.6.9. Основной цикл игры

Основной цикл программы состоит из вызова рассмотренных нами функций `set_iam()`, `collision()`, `go_object()`, `set_new_object()`, `set_new_cursor()` через промежуток времени, установленный в переменных `GAME.vk`, `GAME.vo_10`, `GAME.vo_11`, `GAME.vo_12`. Содержимое функции `game1()` представлено в листинге 25.17.

Листинг 25.17

```
int game1()
 {
 //while(GAME1.balls>0 && GAME1.level<6)
 //{
 long time2=millis();
 //TV.delay(GAME1.xk);
 if(time2-time11>GAME1.vk)
  {set_iam();
  collision();
  time11=time2;
  }
 if(time2-time12>GAME1.vo_12)
  {go_object();time12=time2;}
```

```
 if(time2-time13>GAME1.vo_11)
  {set_new_object();time13=time2;}
 if(time2-time14>GAME1.vo_10)
  {set_new_cursor();time14=time2;}
 TV.delay_frame(10);
 //}
 if(GAME1.balls<0)
  return 0;
 else if(GAME1.level>5)
  return 0;
 else
  return 1;
 }
```

Вид игры и сам ее процесс можно посмотреть в ролике по адресу: **http://www.youtube.com/watch?feature=player_embedded&v=tNJ_oxbT96Q#at=22**.

25.6.10. Добавляем меню для выбора игр

Добавим в скетч меню для вывода трех игр, которые вы можете написать самостоятельно. Содержимое основного цикла программы и функции `menu()` представлены в листинге 25.18.

Листинг 25.18

```
void loop() {
 switch(menu(pmenu))
  {
  case 1:start_game();
      while(game1()>0);
      break;
  default:
      break;
  }

 }
//***** меню для выбора игры
int menu(int poz)
 {
 TV.clear_screen();
 pmenu=max(poz,1);
 int pmenu1=pmenu;
 TV.println(60,30,"Game 1");
 TV.println(60,50,"Game 2");
 TV.println(60,70,"Game 3");
 TV.draw_rect(50,5+20*pmenu,40,10,WHITE,INVERT);
 TV.delay(500);
```

```
while(digitalRead(12)==LOW)
 {
 if(analogRead(A1)<100)
   pmenu=max(pmenu-1,1);
 else if(analogRead(A1)>900)
   pmenu=min(pmenu+1,3);
 else ;
 if(pmenu1!=pmenu)
  {
  TV.delay(500);
  TV.draw_rect(50,5+20*pmenu1,40,10,BLACK,INVERT);
  TV.draw_rect(50,5+20*pmenu,40,10,WHITE,INVERT);
  pmenu1=pmenu;
  }
 }
return pmenu;
}
```

Полностью данный скетч вы можете найти в папке examples/_25_1 сопровождающего книгу электронного архива. Там же находятся программа Image2Code (файл Image2Code.exe), файл для создания графического примитива favicon.bmp, папка TVOutfonts, в которой содержатся файлы шрифтов, и папка TVOut с файлами библиотеки TVOut.

ГЛАВА 26

Arduino и радиоуправление

Аппаратура радиоуправления используется для управления движущимися моделями. Передача команд от пилота к модели происходит по радиоканалу. Аппаратура радиоуправления состоит из передатчика, который находится у пилота, и размещенных на модели приемника исполнительных механизмов. Для управления исполнительными механизмами нередко используются микроконтроллеры Arduino. Микроконтроллер при этом должен получать и обрабатывать команды от приемника.

По конструкции органов управления, на которые, собственно, воздействуют пальцы пилота, передатчики делятся на джойстиковые (рис. 26.1) и пистолетного типа. В первых установлено, как правило, два двухкоординатных джойстика. Такие передатчики используются для управления летающими моделями.

Рис. 26.1. Передатчик HK-T6A

Для управления движущимися моделями требуется воздействие одновременно на несколько функций. Поэтому передатчики радиоуправления делают многоканальными. Так, для авто- и судомоделей требуются два канала: управление направлением движения и оборотами двигателя. Для полноценного управления самолетом нужно не менее четырех, а вертолетом — пяти каналов.

Для самолетов на два двухкоординатных джойстика выводятся функции управления рулем высоты, направления, элеронами и газом двигателя. Конкретная раскладка функций по джойстикам бывает двух типов: Mode 1 (рис. 26.2) — руль высоты слева по вертикали и руль направления по горизонтали, газ справа по вертикали и крен по горизонтали, а также Mode 2 (рис. 26.3) — газ слева по вертикали и руль направления по горизонтали, руль высоты справа по вертикали и крен по горизонтали. Есть еще типы раскладок Mode 3 и 4, но они мало распространены.

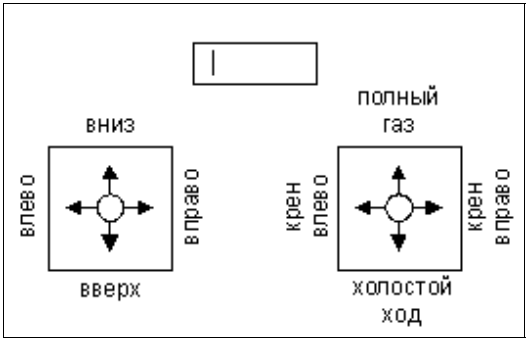


Рис. 26.2. Раскладка Mode 1

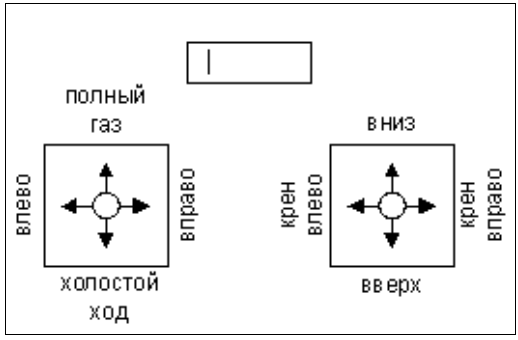


Рис. 26.3. Раскладка Mode 2

26.1. Принципы формирования радиосигнала

Для того чтобы излучаемый передатчиком радиосигнал мог переносить полезную информацию, он подвергается модуляции. То есть, управляющий сигнал изменяет параметры несущей радиочастоты. На практике нашли применение управление амплитудой и частотой несущей, обозначаемые буквами AM (Amplitude Modulation, амплитудная модуляция) и FM (Frequency Modulation, частотная модуляция).

В радиоуправлении используется только дискретная двухуровневая модуляция. В варианте AM несущая имеет либо максимальный, либо нулевой уровень. В варианте FM излучается сигнал постоянной амплитуды либо с частотой F, либо с чуть смещенной частотой F + df. Сигнал FM передатчика напоминает сумму сигналов двух AM-передатчиков, работающих в противофазе на частотах F и F + df соответственно. Из этого можно понять, даже не углубляясь в тонкости обработки радиосигнала в приемнике, что в одинаковых помеховых условиях FM-сигнал имеет принципиально большую помехозащищенность, чем AM-сигнал. AM-аппаратура, как правило, дешевле, однако разница не очень велика. В настоящее время использование AM-аппаратуры оправдано только для тех случаев, когда расстояние до

модели относительно невелико. Как правило, это справедливо для автомоделей, судомоделей и комнатных авиамоделей. Вообще, летать с использованием АМ-аппаратуры можно лишь с большой опаской и вдали от промышленных центров.

Модуляция позволяет наложить на излучаемую несущую полезную информацию. Однако в радиоуправлении используется только многоканальная передача информации. Для этого все каналы уплотняются в один посредством кодирования. Сейчас для этого используется только широтно-импульсная модуляция, обозначаемая буквами PPM (Pulse Phase Modulation), и импульсно-кодовая модуляция, обозначаемая буквами PCM (Pulse Code Modulation).

На рис. 26.4 приведен типовой PPM-сигнал пятиканальной аппаратуры. PPM-сигнал имеет фиксированную длину периода T = 20 мс. Это означает, что информация о положениях ручек управления на передатчике попадает на модель 50 раз в секунду, что определяет быстродействие аппаратуры управления. Как правило, этого хватает, поскольку скорость реакции пилота на поведение модели намного меньше. Все каналы пронумерованы и передаются по порядку номеров. Значение сигнала в канале определяется величиной временного промежутка между первым и вторым импульсом — для первого канала, между вторым и третьим — для второго канала и т. д.

Диапазон изменения величины временного промежутка при движении джойстика (ручки управления) из одного крайнего положения в другое определен от 1 до 2 мс. Значение 1,5 мс соответствует среднему (нейтральному) положению джойстика. Продолжительность межканального импульса составляет около 0,3 мс. Данная структура PPM-сигнала является стандартной для всех производителей аппаратуры радиоуправления.

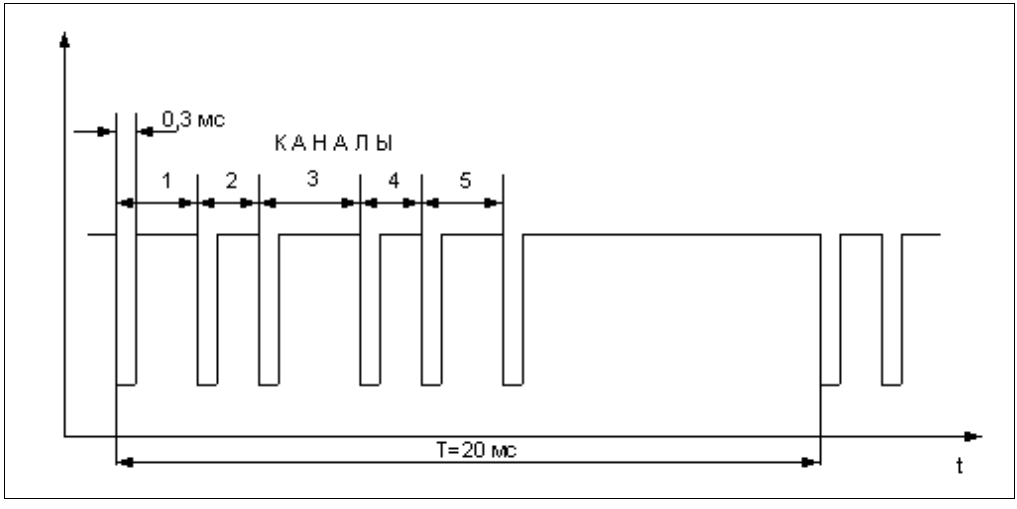


Рис. 26.4. Типовой PPM-сигнал пятиканальной аппаратуры

26.2. Установка связи приемника с передатчиком

На управляемой модели устанавливается приемник. В нашем проекте мы будем использовать приемник HK-T6A (рис. 26.5).

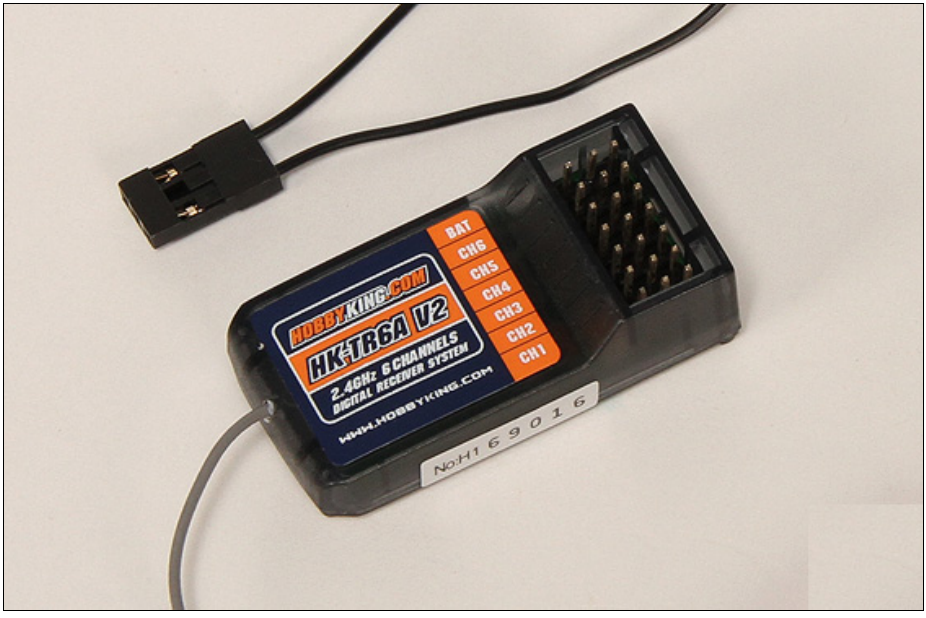


Рис. 26.5. Приемник HK-T6A

Как видно из рисунка, число каналов управления у этого приемника — шесть.

Для начала необходимо связать между собой передатчик и приемник в следующем порядке:

1. Установить батарею в передатчике.
2. Вставить шнур для кодирования (показан на рис. 26.5 вверху) в контакты BAT приемника.
3. Соединить батарею питания приемника с одним из портов канала — если светодиоды на приемнике и передатчике вспыхивают одновременно, значит приемник успешно включен.
4. Зажать (нажать и удерживать) кнопку поиска частоты на передатчике и включить питание — если светодиоды на приемнике не мигают, а просто горят, то связь установлена.
5. Отпустить кнопку на передатчике, отсоединить шнур на приемнике.
6. Установить сервомашинку в какой-нибудь из каналов и проверить работоспособность — при движении стиков на передатчике сервомашинка должна вращаться.

Теперь можно подключать приемник к Arduino и писать скетч для приема команд с передатчика.

26.3. Разработка скетча приема команд для Arduino

Приступим к разработке скетча для считывания данных, поступающих с передатчика на приемник. Подключаем сигнальные контакты приемника на выводы D11, D10, D9, D8 Arduino. Не забываем подать питание 5 В на приемник. Диапазон изменения величины временного промежутка при движении джойстика из одного крайнего положения в другое определен величиной от 1 до 2 мс. Для определения длительности сигнала, поступающего на входы, будем использовать функцию `pulseIn()`. Напомним, что функция считывает длину сигнала на заданном порту (`HIGH` или `LOW`). Например, если задано считывание `HIGH` функцией `pulseIn()`, функция ожидает, пока на заданном порту не появится `HIGH`. Когда `HIGH` получен, включается таймер, который будет остановлен, когда на порту входа/выхода будет `LOW`. Функция `pulseIn()` возвращает длину сигнала в микросекундах. Функция возвращает 0, если в течение заданного времени (тайм-аута) не был зафиксирован сигнал на порту.

Синтаксис функции `pulseIn()`:

```
pulseIn(pin, value)
pulseIn(pin, value, timeout)
```

Параметры:

- `pin` — номер порта входа/выхода, на котором будет ожидаться сигнал;
- `value` — тип ожидаемого сигнала (`HIGH` или `LOW`);
- `timeout` — время ожидания сигнала (тайм-аут) в микросекундах; по умолчанию — одна секунда.

Возвращаемое значение: длина сигнала в микросекундах или 0, если сигнал не получен до истечения тайм-аута.

Выбираем поочередно порт для 1, 2, 3, 4 каналов, параметр `value=HIGH`, `timeout=2 мсек`. Получаемое значение сигнала от 1 до 2 мсек. Данные, полученные с передатчика, выводим в последовательный порт (рис. 26.6). Как видно из рис. 26.6, при перемещении стиков передатчика изменяется значение считываемого сигнала. Код скетча представлен в листинге 26.1.

Листинг 26.1

```
// передатчик - HK-T6A
// приемник -
// приемник :
// Ch1 - Rudder (руль направления, рыскание, YAW)
// Ch2 - Elevator (тангаж, PITCH)
```

```
// Ch3 - Throttle (газ)
// Ch4 - Aileron (элероны, ROLL)

unsigned long Ch1Value,Ch2Value,Ch3Value,Ch4Value;
unsigned long last1,last2,last3,last4;
int pinCh1=11;
int pinCh2=10;
int pinCh3=9;
int pinCh4=8;

void setup()
 {
 Serial.begin(9600);
 Serial.println("Ready");
 pinMode (pinCh1, INPUT); // connect Rx channel 1
 pinMode (pinCh2, INPUT); // connect Rx channel 2
 pinMode (pinCh3, INPUT); // connect Rx channel 3
 pinMode (pinCh4, INPUT); // connect Rx channel 4
 last1 = pulseIn (pinCh1, HIGH); //read RC channel 1
 last2 = pulseIn (pinCh2, HIGH); //read RC channel 2
 last3 = pulseIn (pinCh3, HIGH); //read RC channel 3
 last4 = pulseIn (pinCh4, HIGH); //read RC channel 4
 }

void loop()
 {
 //
 Ch1Value = pulseIn (pinCh1, HIGH, 20000); //read RC channel 1
 if (Ch1Value == 0) {Ch1Value = last1;}
 else {last1 = Ch1Value;}
 Serial.print (" Ch1: ");Serial.print (Ch1Value);
 //
 Ch2Value = pulseIn (pinCh2, HIGH, 20000); //read RC channel 2
 if (Ch2Value == 0) {Ch2Value = last2;}
 else {last2 = Ch2Value;}
 Serial.print(" Ch2: ");Serial.print (Ch2Value);
 //
 Ch3Value = pulseIn (pinCh3, HIGH, 20000); //read RC channel 3
 if (Ch3Value == 0) {Ch3Value = last3;}
 else {last3 = Ch3Value;}
 Serial.print(" Ch3: ");Serial.print (Ch3Value);
 //
 Ch4Value = pulseIn (pinCh4, HIGH, 20000); //read RC channel 4
 if (Ch4Value == 0) {Ch4Value = last4;}
 else {last4 = Ch4Value;}
 Serial.print(" Ch4: ");Serial.print (Ch4Value);
 Serial.println("");

 //delay(1000);
}
```

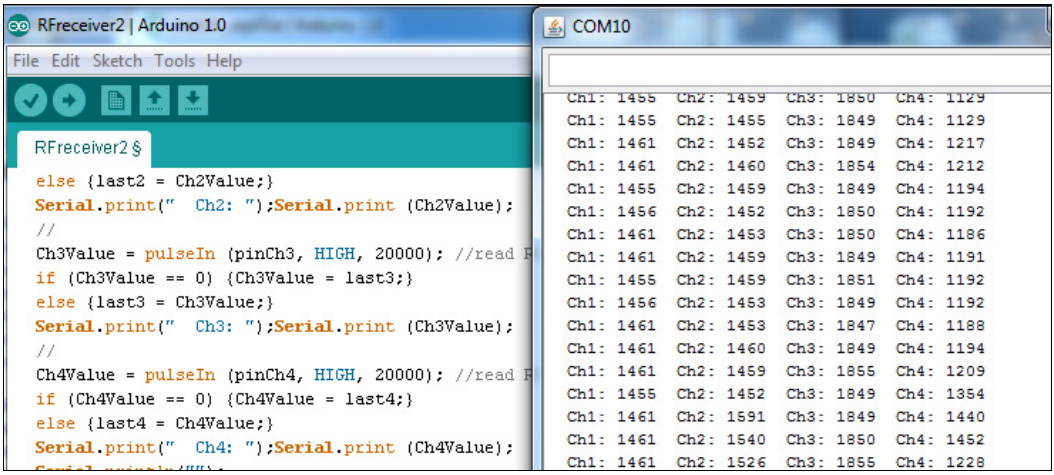


Рис. 26.6. Вывод данных с передатчика

Код данного скетча находится в папке examples/_26_1 сопровождающего книгу электронного архива.

ГЛАВА 27

Arduino и беспроводной радиомодуль NRF24L01

27.1. Радиомодуль NRF24L01

Если нам необходимо соединить два Arduino-устройства, очень популярным и бюджетным вариантом будет использование беспроводных модулей NRF24L01 (рис. 27.1). Малое энергопотребление, достойный радиус действия и низкая цена — вот основные качества радиомодуля NRF24L01. Это позволяет ему конкурировать с устройствамиXbee и Bluetooth.

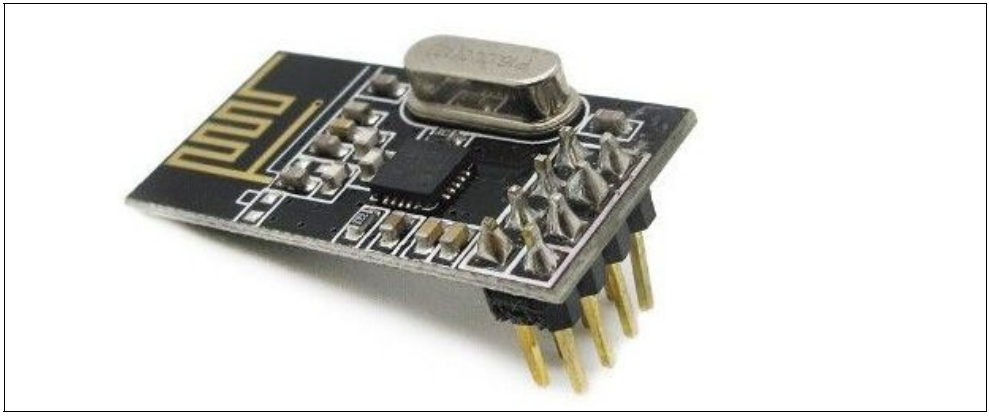


Рис. 27.1. Радиомодуль NRF24L01

Характеристики радиомодуля NRF24L01:

- напряжение питания — от 1,9 до 3,6 вольт DC;
- ток потребления:
  - пиковый ток при 2 Мбит — 12,5 мА;
  - рабочий ток — 11 мА;
  - режим ожидания — 32 мкА;
- скорость передачи — до 2 Мбит, интерфейс SPI;

- 125 каналов связи, скачкообразная перестройка частоты;
- поддержка многоточечной связи, аппаратный контроль ошибок;
- встроенная антенна 2,4 ГГц;
- встроенный стабилизатор напряжения;
- количество PIN — 10;
- радиус действия — 100 метров (на открытом пространстве);
- размер — 33×14 мм;
- вес — до 10 г.

Если вам нужно большее расстояние передачи, можно использовать модуль с внешней антенной (рис. 27.2), при этом расстояние передачи может составить до 1000 м со скоростью 250 Кбит/с (или 500 м со скоростью до 2 Мбит/с).

Назначение выводов модуля NRF24L01 представлено на рис. 27.3.

Рис. 27.2. Модуль NRF24L01 с антенной

Рис. 27.3. Назначение выводов модуля NRF24L01

27.2. Библиотека для работы с модулем NRF24L01

Набор функций, предоставляемых модулем NRF24L01, поддерживает библиотека `Mirf`. Скачать библиотеку можно с официального сайта Arduino по ссылке **http://playground.arduino.cc/InterfacingWithHardware/Nrf24L01**.

Используемые контакты модуля NRF24L01:

- 12 — MOSI;
- 11 — MISO;
- 13 — SCK;
- 8 — CE;
- 7 — CSN.

27.2.1. Параметры библиотеки *Mirf*

- Mirf.csnPin — пин CE (по умолчанию 9);
- Mirf.cePin — пин CSN (по умолчанию 7);
- Mirf.payload — размер буфера в байтах (по умолчанию 16, максимум 32);
- Mirf.channel — номер радиоканала 0–127 (по умолчанию 0).

27.2.2. Функции библиотеки *Mirf*

Функция *init()*

Функция init() инициализирует модуль, устанавливает значения для настраиваемых контактов и инициализирует модуль SPI.

Синтаксис:

```
Mirf.init()
```

Параметров нет.

Функция *setRADDR()*

Функция setRADDR() устанавливает адрес получателя.

Синтаксис:

```
Mirf.setRADDR(byte *addr)
```

Параметр: addr — адрес получателя.

Функция *setTADDR()*

Функция setTADDR() устанавливает адрес отправителя.

Синтаксис:

```
Mirf.setTADDR(byte *addr)
```

Параметр: addr — адрес получателя.

Функция *config()*

Функция config() устанавливает определенные в параметрах номер канала Mirf.channel и емкость буфера Mirf.payload.

Синтаксис:

```
Mirf.config().
```

Параметров нет.

Функция *dataReady()*

Функция `dataReady()` определяет, есть ли данные для получения.

Синтаксис:

```
Mirf.dataReady()
```

Параметров нет.

Возвращаемые значения:

- `true` — есть данные для получения;
- `false` — нет данных для получения.

Функция *getData()*

Функция `getData()` получает данные из буфера размером `Mirf.payload`.

Синтаксис:

```
Mirf.getData(byte *data)
```

Параметр: `data` — указатель на массив для получения данных.

Функция *send()*

Функция `send()` отправляет данные.

Синтаксис:

```
Mirf.send(byte *data)
```

Параметр: `data` — указатель на массив данных.

Функция *isSending()*

Функция `isSending()` определяет, отправляются данные или нет.

Синтаксис:

```
Mirf.isSending()
```

Параметров нет.

Возвращаемые значения:

- `true` — данные отправляются;
- `false` — данные не отправляются.

27.3. Пример соединения двух плат Arduino с помощью модуля NRF24L01

Радиомодули NRF24L01 подключаются к микроконтроллеру по SPI-интерфейсу. Для работы требуется напряжение в диапазоне от 1,8 до 3,6 вольт, впрочем, входы/выходы выдерживают до 5 вольт, поэтому при подключении к пятивольтовым устройствам дополнительные согласующие цепи ставить нет необходимости. Подключаем к платам Arduino по схеме, приведенной в табл. 27.1.

Таблица 27.1. *Подключение к Arduino и модуля NRF24L01*

Контакты модуля NRF24L01	Контакты платы Arduino
MISO	12
MOSI	11
SCK	13
CE	8
CSN	7
3,3 В	3,3 В
GND	GND

Напишем скетч, отправляющий данные, полученные по последовательному порту, из одной платы Arduino в другую через модуль NRF24l01. Код скетча представлен в листинге 27.1.

Листинг 27.1

```
#include <SPI.h>
#include <Mirf.h>
#include <nRF24L01.h>
#include <MirfHardwareSpiDriver.h>
#define MAX_BUFF 32 // Буфер приема-передачи

void setup(){
 Serial.begin(9600);
 Mirf.spi = &MirfHardwareSpi;
 Mirf.init();

 Mirf.setRADDR((byte *)"module1"); // Здесь задаем адрес
 Mirf.payload = MAX_BUFF; // Здесь задаем буфер
 Mirf.channel = 10;
// Это канал приема-передачи - должен
// быть одинаковым у устройств.
 Mirf.config();

 Serial.println("Start..");
}
```

```
char buff[MAX_BUFF];
int c_count = 0;

void loop(){
 int i;
 //sending
 if (c_count = Serial.available()) {
  if (c_count <= MAX_BUFF) {
   for (i=0; i<c_count; i++) {
    buff[i] = Serial.read();
   }
  } else {
   for (i=0; i<MX_BUFF; i++) {
    buff[i] = Serial.read();
   }
  }
  buff[i] = 0;
  Mirf.setTADDR((byte *)" module2"); //Адрес!
  Serial.print(">");
  Mirf.send((uint8_t *)buff);
  while(Mirf.isSending()){
  }
  Serial.println(buff);
 }
 delay(10);
 //receiving
 if(Mirf.dataReady()) {
  Mirf.getData((uint8_t *)buff);
  Serial.print("<");
  Serial.println(buff);
 }
 delay(100);
}
```

Загружаем скетч в одну из плат Arduino. Для другой платы в скетче меняем строки:

```
Mirf.setRADDR((byte *)"module1"); // Здесь задаем адрес
Mirf.setTADDR((byte *)" module2"); // Адрес!
```

на следующие:

```
Mirf.setRADDR((byte *)"module2"); // Здесь задаем адрес
Mirf.setTADDR((byte *)" module1"); // Адрес!
```

Запускаем монитор последовательного порта для первой платы и видим обмен данными по беспроводному соединению между ними (рис. 27.4 и 27.5).

Полностью данные скетчи можно найти в папках examples/_27_1 и examples/_27_2 сопровождающего книгу электронного архива.

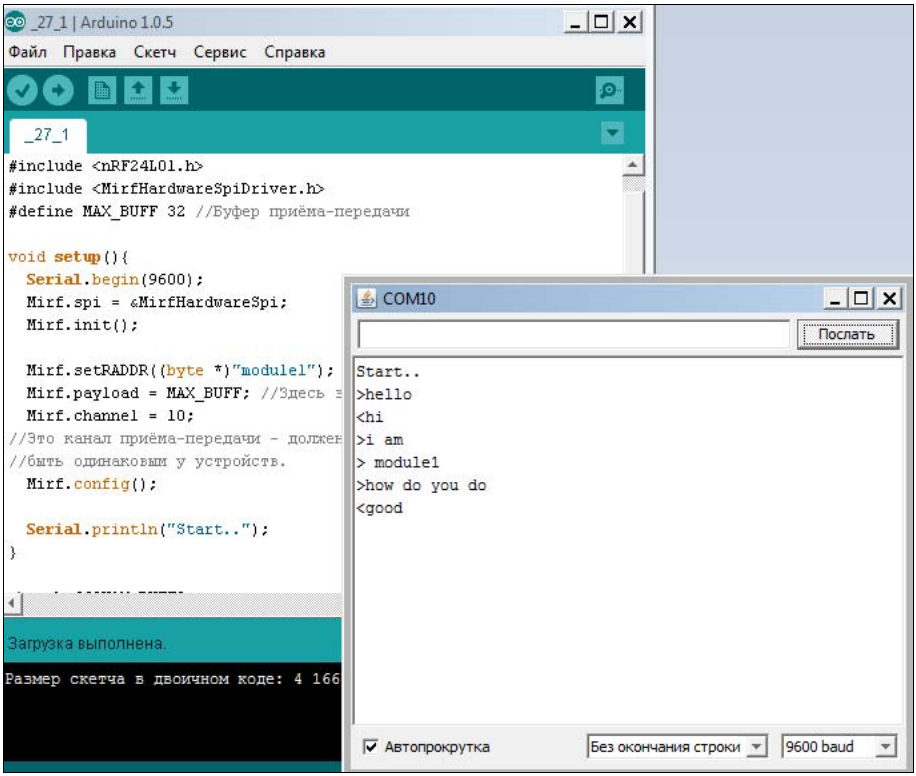


Рис. 27.4. Отправка/получение данных через NRF24L01 (первая плата)

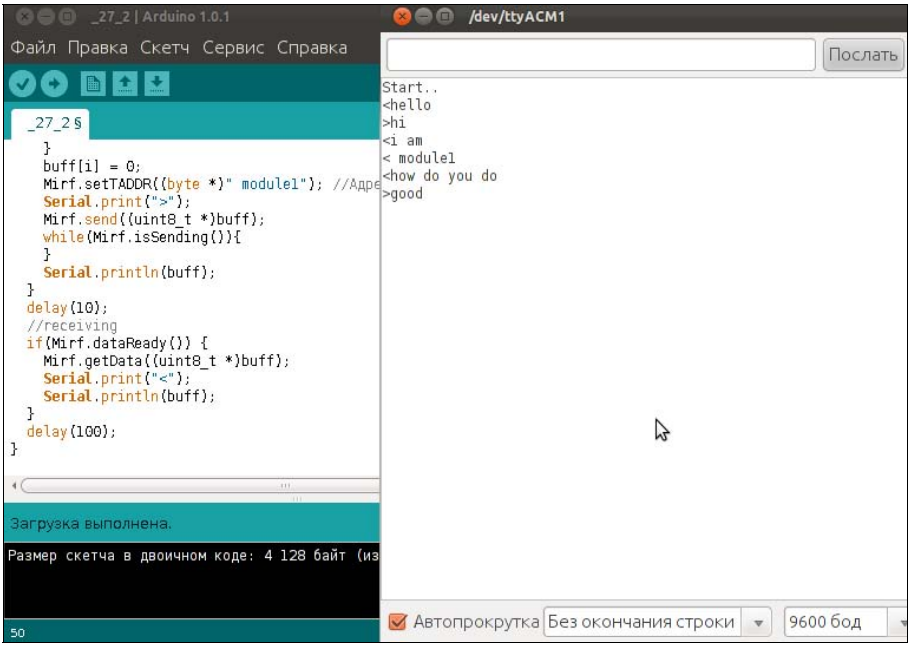


Рис. 27.5. Отправка/получение данных через NRF24L01 (вторая плата)

ГЛАВА 28

Работа Arduino с USB-устройствами

28.1. Интерфейс USB

Последовательный интерфейс USB используется для подключения периферийных устройств. Соответственно, существуют понятия "главное устройство" — *хост* (он управляет обменом данными через интерфейс, выступает инициатором обмена) и "периферийное устройство" — *клиент* (в процессе обмена данными он "подчиняется" хосту). Логика работы хоста и клиента принципиально отличаются, поэтому нельзя напрямую соединять устройства "хост — хост" и "клиент — клиент". Имеются специальные устройства — *хабы*, которые подключаются в качестве клиента к одному хосту и, в то же время, выступают хостом для других периферийных устройств. Хабы также применяются для "разветвления" шины USB.

Физически интерфейс USB (до версии 2.0) использует 4 провода (рис. 28.1):

- "земля" (GND);
- +5 В (VBUS);
- D–, D+ — линии приема/передачи данных (обозначения D+ и D– условны, с электрическими потенциалами это никак не связано).

Спецификация USB 1.0 определяла два типа разъемов: A — на стороне контроллера или концентратора USB и B — на стороне периферийного устройства. Впоследствии были разработаны миниатюрные разъемы для применения USB в переносных

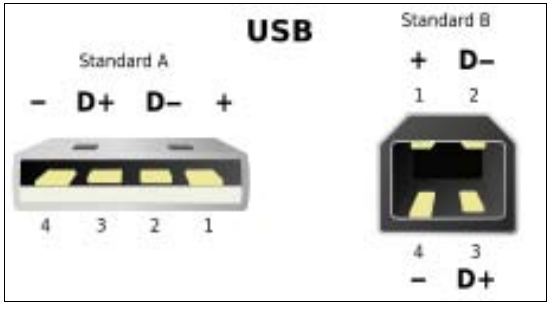


Рис. 28.1. Назначение контактов USB 1.0, USB 2.0

и мобильных устройствах, получившие название Mini-USB. Новая версия миниатюрных разъемов, называемых Micro-USB, была представлена USB Implementers Forum 4 января 2007 года.

Благодаря встроенным линиям питания USB позволяет подключать периферийные устройства без собственного источника питания (максимальная сила тока, потребляемого устройством по линиям питания шины USB, не должна превышать 500 мА, у USB 3.0 — 900 мА).

Стандарт USB поддерживает концепцию plug-and-play. Эта концепция подразумевает, что пользователю достаточно "воткнуть" устройство в соответствующий порт ПК. Дальше ОС автоматически определит тип подключенного устройства, найдет подходящий для данного устройства драйвер, сконфигурирует устройство и т. д. Для того чтобы это все работало, стандартом USB предусмотрены некие общие требования для всех устройств:

- каждое устройство содержит дескриптор (описание) устройства;
- есть общий для всех USB-устройств механизм, который позволяет ОС прочитать дескриптор устройства для того, чтобы идентифицировать устройство, узнать его характеристики;
- есть общий для всех USB-устройств механизм, который позволяет ОС выполнить первичную конфигурацию устройства (например, присвоить устройству новый адрес).

28.2. USB Host Shield

Очень соблазнительно иметь возможность подключать к Arduino USB-устройства, которых великое множество. Плата расширения USB Host Shield 2.0 (рис. 28.2) позволяет Arduino выступать в роли родительского USB-устройства для практически любой имеющейся USB-периферии. С этой платой открывается масса новых возможностей для создания интересных устройств. В настоящее время платой USB Host Shield 2.0 поддерживаются следующие классы устройств:

- HID-устройства, такие как клавиатуры, мыши, джойстики и др.;
- игровые устройства: Sony PS3, Nintendo Wii, Xbox360;
- USB преобразователи: FTDI, PL-2303, ACM, а также некоторые аппараты и GPS-приемники;
- Android -устройства;
- цифровые фотоаппараты: Canon (EOS, PowerShot), Nikon.

Для программирования USB Host Shield используется специальная библиотека, скачать которую можно со страницы **https://github.com/felis/USB_Host_Shield_2.0**. Библиотеку можно найти также в папке libraries/USB сопровождающего книгу электронного архива.

Спецификацию и примеры использования этой библиотеки можно найти на сайте Circuits@Home **http://www.circuitsathome.com/arduino_usb_host_shield_projects**.

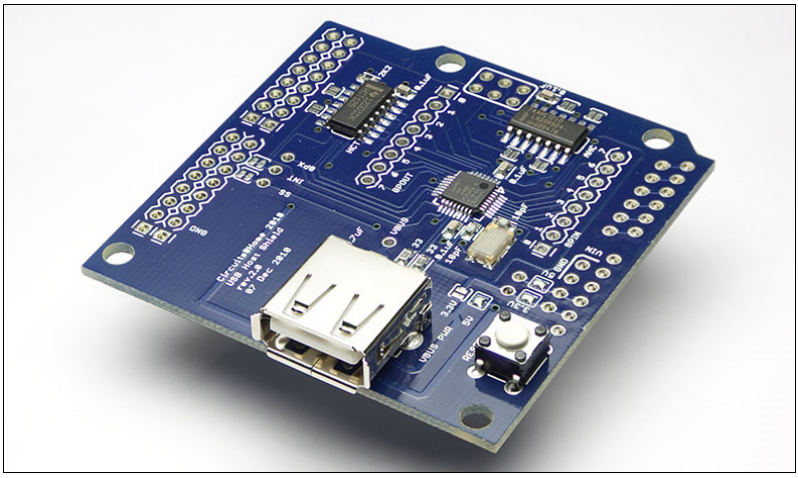


Рис. 28.2. USB Host Shield 2.0

28.3. HID-устройства USB

HID (Human Interface Device) — устройство, подключаемое к вычислительной технике, с тем, чтобы с ней мог работать человек. Говоря проще, HID — это устройство ввода информации. Устройства ввода необходимы для непосредственного участия человека в работе компьютера: для введения исходных данных для вычислений, для выбора параметров действия, для редактирования имеющихся данных и результатов и т. д.

HID-устройства ввода различаются по типу вводимой информации:

- для текстовой информации — это преимущественно *клавиатуры*. Они служат для управления техническими и механическими устройствами (компьютер, калькулятор, кнопочный телефон). Каждой клавише устройства соответствует один или несколько определенных символов. Возможно увеличить количество действий, выполняемых с клавиатуры, с помощью сочетаний клавиш. В клавиатурах такого типа клавиши сопровождаются наклейками с изображением символов или действий, соответствующих нажатию;
- для звуковой информации — это *микрофон*. Электроакустические приборы, преобразующие звуковые колебания в колебания электрического тока, используются во многих устройствах (телефоны и магнитофоны, приборы звукозаписи и видеозаписи на радио и телевидении, для радиосвязи);
- для графической информации:
  - *сканер* — устройство для считывания плоского изображения и представления его в растровой электронной форме;
  - *цифровая камера* — устройство (фотоаппарат), использующее массив полупроводниковых светочувствительных элементов (матрицу), на которую изображение фокусируется с помощью системы линз объектива. Полученное

изображение сохраняется в электронном виде в памяти самой камеры или же дополнительном цифровом носителе;

- *веб-камера* — цифровая видео- или фотокамера, способная в реальном времени фиксировать изображения, предназначенные для дальнейшей передачи по сети Интернет как в потоковом режиме, так и за определенные промежутки времени;
- *плата захвата* (тюнер) — электронное устройство для преобразования аналогового видеосигнала в цифровой видеопоток. Используется для захвата телесигнала, сигнала с камер видеонаблюдения и др.

HID-устройства управления различаются по функционалу:

- ❒ относительное позиционирование (обрабатывают информацию о перемещении):
  - *мышь* — манипулятор, преобразующий механические движения в движение курсора на экране. Различают механические, оптические, гироскопические, сенсорные мыши;
  - *трекбол* — манипулятор, чей принцип работы сходен с шариковой механической мышью и аналогичен мыши по принципу действия и по функциям. Однако пользователь не передвигает мышь, а управляет с помощью ладони или пальцев непосредственно шариком, закрепленным на специальном держателе с датчиками;
  - *трекпойнт* — миниатюрный тензометрический джойстик, применяемый в ноутбуках для замены мыши. Трекпойнт считывает направление и силу давления пальца пользователя;
  - *тачпад* — сенсорная панель, применяемая, в основном, в портативных компьютерах. В отличие от трекпойнта, считывающего давление пальца, тачпад считывает емкостные характеристики при соприкосновении пальца с поверхностью емкостных датчиков. Поэтому управление тачпадом с помощью непроводящих предметов (ручка, карандаш, стилус) достаточно проблематично;
  - *джойстик* — устройство ввода информации, которое представляет собой качающуюся в двух плоскостях ручку, боковое давление которой считывается датчиками в основании устройства. Используется как игровой гаджет, а так же как средство управления (например, роботизированной техникой на производстве);
- ❒ абсолютное позиционирование (высчитывают абсолютные координаты на плоскости, в качестве которой выступает устройство):
  - *графический планшет* — устройство для ввода графики (отдельных линий и рисунков) от руки непосредственно в компьютер. Состоит из пера и плоского планшета, чувствительного к нажатию пера;
  - *тачскрин* (сенсорный экран) — устройство ввода информации, представляющее собой экран, реагирующий на прикосновения к нему (используется в современных смартфонах, платежных и информационных терминалах).

Отдельно хочется выделить специальные HID-устройства для компьютерных игр:

- *игровые мыши* — отличаются от обычных компьютерных мышей высокой чувствительностью, настраиваемым весом, большим количеством программируемых кнопок;
- *кейпады* — специальные игровые клавиатуры-приставки, в которых кнопки скомбинированы для максимального удобства игрока (в современные модели встраивается мини-джойстик);
- *руль и педали* — манипуляторы для игр жанра "автогонки" (рэйсинг);
- *джойстики* — используются для игр жанра "авиасимуляторы";
- *геймпады* — специальные игровые манипуляторы, используемые в аркадных жанрах (перешли с игровых консолей);
- *танцевальные платформы* — специальные платформы с датчиками давления. Управление производится с помощью ног. Используются для игр жанра "танцевальные аркады";
- *музыкальные инструменты* (гитары, барабаны) — специальные манипуляторы в форме музыкальных инструментов с кнопками и датчиками давления. Используются для игр жанра "музыкальные аркады".

Операционные системы, как правило, имеют встроенные драйверы HID-класса, так что у разработчиков отпадает необходимость в трудоемкой собственной разработке драйвера для нового устройства. Чтобы определить устройство как HID, необходимо поддержать ряд структур, описывающих HID-интерфейс, а также написать алгоритм обмена по interrupt-каналу (каналу прерываний) передачи данных. Во многих отношениях устройства HID не имеют никаких особенных отличий от других USB-устройств. Однако кроме требований, которые относятся ко всем USB-устройствам, устройства HID выдвигают ряд дополнительных требований:

- HID-устройство должно иметь `Interrupt In` — конечную точку для выдачи данных в хост. `Interrupt Out` — конечная точка для получения периодических данных от хоста, является опциональной и может не использоваться;
- HID-устройство должно содержать дескриптор класса — `Device Class Descriptor` и один или более дескрипторов репорта `HID Report Descriptor`;
- HID-устройство должно поддерживать специфический для класса управляющий запрос `Get_Report`, а также опционально поддерживать дополнительный запрос `Set_Report`;
- для передачи `Interrupt In` (данные из устройства в хост) устройство должно положить данные репорта в FIFO соответствующей конечной точки и разрешить передачу;
- для передачи `Interrupt Out` (данные из хоста в устройство) устройство должно разрешить соответствующую конечную точку `Out`, а затем, после прихода пакета, забрать данные из FIFO.

28.4. Подключение HID-мыши USB

В библиотеке `USB Host Shield 2.0` имеется пример для подключения к Arduino HID-мыши USB — USBHIDBootMouse.pde. Загружаем его и смотрим в мониторе последовательного порта результат работы скетча (рис. 28.3).

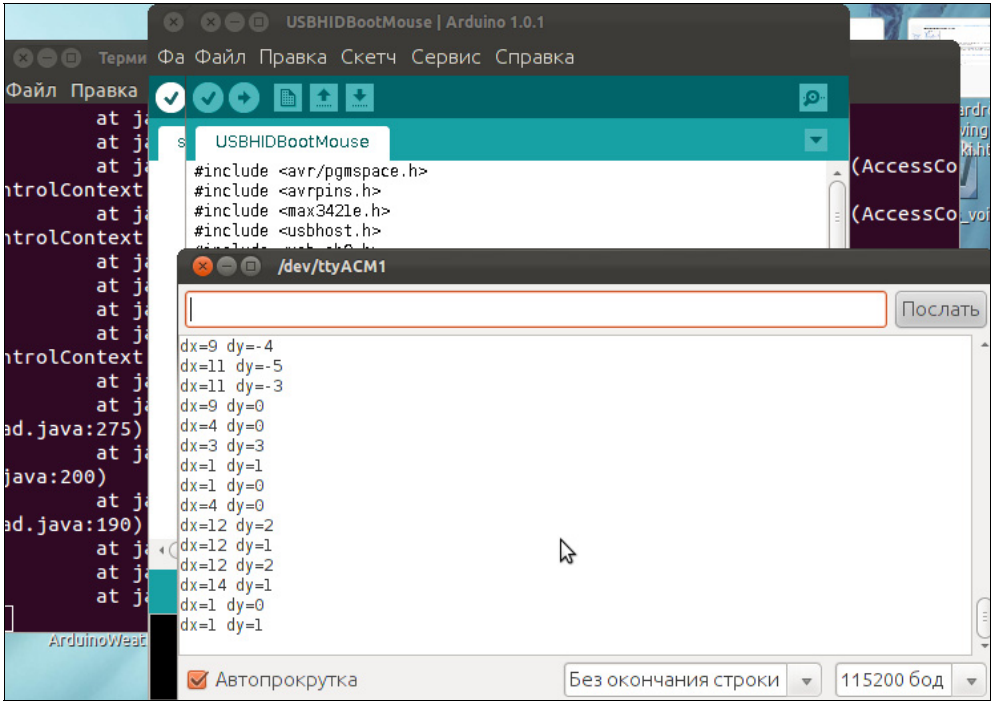


Рис. 28.3. Скетч для подключения HID-мыши USB

28.5. Использование HID-устройства (руль Defender) для управления роботом

Теперь создадим собственный проект. Будем управлять роботом с помощью игрового манипулятора — руля Defender Forsage Drift GT (рис. 28.4). Для этого необходимо организовать беспроводное управление моделью робота данным устройством. Для беспроводной передачи данных воспользуемся беспроводным радиомодулем NRF24L01, работу с которым мы рассмотрели в *главе 27*.

Руль подсоединен к плате Arduino через USB Host Shield. В качестве блока управления подсоединяем к Arduino модуль 2,4 ГГц NRF24L01. Блок робота выполнен в виде гусеничной платформы, рассмотренной нами в *главе 21*. Дополнительно к Arduino робота подключаем такой же радиомодуль 2,4 ГГц NRF24L01. Электрическая схема блока робота представлена на рис. 28.5.

Рис. 28.4. Руль Defender Forsage Drift GT

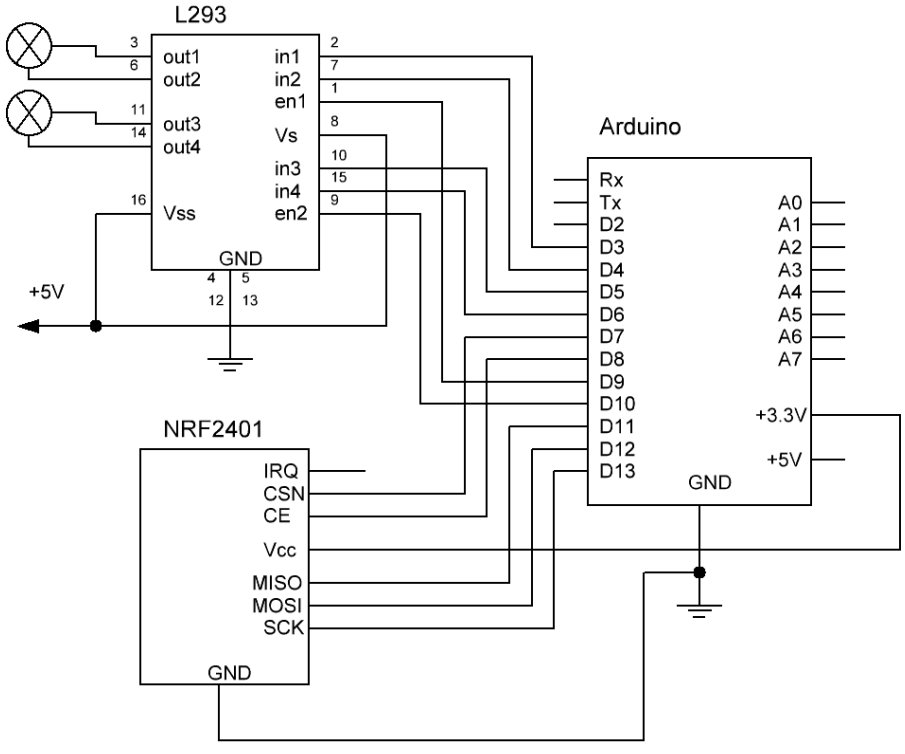


Рис. 28.5. Электрическая схема блока робота

Загружаем скетч USB_HID_Desc.pde, входящий в примеры библиотеки `USB_Host`, подсоединяем к Arduino через USB Host Shield наш руль и получаем `Descriptor Report` HID-устройства (рис. 28.6, листинг 28.1).

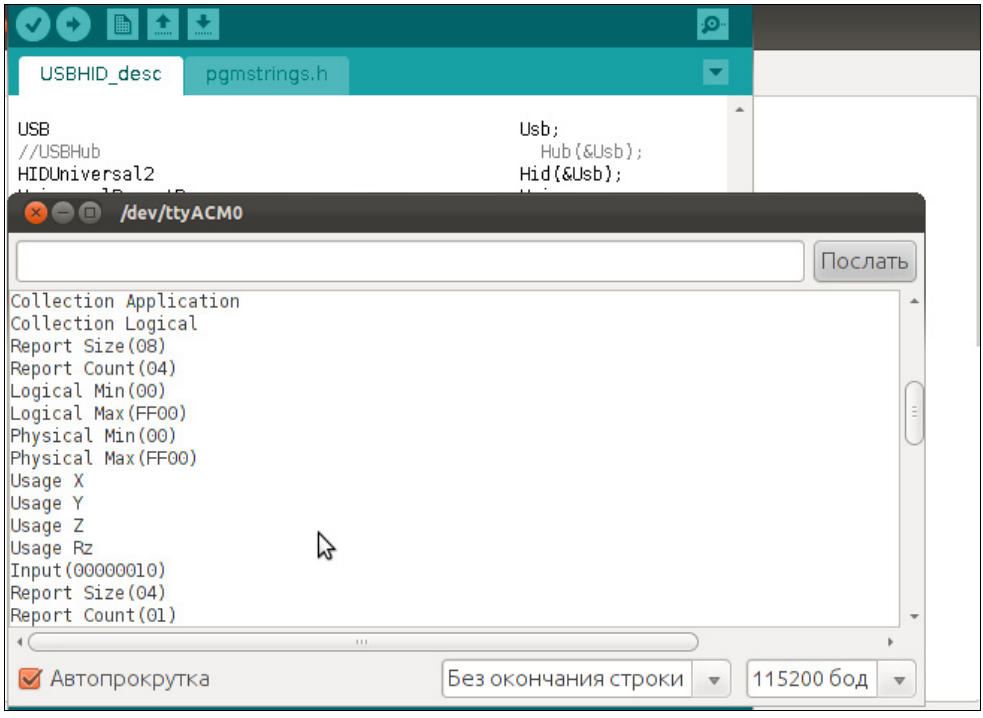


Рис. 28.6. Скетч получения `Descriptor Report` HID-устройства

Листинг 28.1

```
Usage Page Gen Desktop Ctrls(01)
Usage Game Pad
Collection Application
Collection Logical
Report Size(08)
Report Count(04)
Logical Min(00)
Logical Max(FF00)
Physical Min(00)
Physical Max(FF00)
Usage X
Usage Y
Usage Z
Usage Z
Usage Rz
Input(00000010)
Report Size(04)
```

```
Report Count(01)
Logical Max(07)
Physical Max(3B01)
Unit(14)
Usage Hat Switch
Input(01000010)
Unit(00)
Report Size(01)
Report Count(0C)
Logical Max(01)
Physical Max(01)
Usage Page Button(09)
Usage Min(01)
Usage Max(0C)
Input(00000010)
Usage Page Undef(00)
Report Size(08)
Report Count(02)
Logical Max(01)
Physical Max(01)
Usage
Input(00000010)
End Collection
Collection Logical
Report Size(08)
Report Count(04)
Physical Max(FF00)
Logical Max(FF00)
Usage
Output(00000010)
End Collection
End Collection
```

Для создания своего кода возьмем код примера, входящий в библиотеку `USB_Host` и реализующий функционал джойстика, а именно USBHIDJoystick.pde, и перепишем файлы hidjoystickrptparser.h (листинг 28.2) и hidjoystickrptparser.cpp (листинг 28.3) для сохранения данных в структуре. Далее добавим отправку данных руля по радиоканалу с помощью библиотеки для NRF24L01 (листинг 28.4). Отправку производим каждые 300 мс.

Листинг 28.2

```
#if !defined(__HIDJOYSTICKRPTPARSER_H__)
#define __HIDJOYSTICKRPTPARSER_H__

#include <inttypes.h>
#include <avr/pgmspace.h>
```

```
#include "avrpins.h"
#include "max3421e.h"
#include "usbhost.h"
#include "usb_ch9.h"
#include "Usb.h"

#if defined(ARDUINO) && ARDUINO >=100
#include "Arduino.h"
#else
#include <WProgram.h>
#endif

#include "printhex.h"
#include "hexdump.h"
#include "message.h"
#include "confdescparser.h"
#include "hid.h"

struct GamePadEventData
{
       uint8_t X, Y, Z1, Z2, Rz;
};

class JoystickEvents
{
public:
       virtual void OnGamePadChanged(const GamePadEventData *evt);
       virtual void OnHatSwitch(uint8_t hat);
       virtual void OnButtonUp(uint8_t but_id);
       virtual void OnButtonDn(uint8_t but_id);
    uint8_t X;
    uint8_t Y;
    uint8_t Z1;
    uint8_t Z2;
    uint8_t Rz;
};

#define RPT_GEMEPAD_LEN             5

class JoystickReportParser : public HIDReportParser
{
       JoystickEvents       *joyEvents;

       uint8_t                      oldPad[RPT_GEMEPAD_LEN];
       uint8_t                      oldHat;
       uint16_t                     oldButtons;
```

```
public:
        JoystickReportParser(JoystickEvents *evt);

        virtual void Parse(HID *hid, bool is_rpt_id, uint8_t len, uint8_t *buf);
};

#endif // __HIDJOYSTICKRPTPARSER_H__
```

Листинг 28.3

```
#include "hidjoystickrptparser.h"

JoystickReportParser::JoystickReportParser(JoystickEvents *evt) :
        joyEvents(evt),
        oldHat(0xDE),
        oldButtons(0)
{
        for (uint8_t i=0; i<RPT_GEMEPAD_LEN; i++)
                oldPad[i]      = 0xD;
}

void JoystickReportParser::Parse(HID *hid, bool is_rpt_id, uint8_t len, uint8_t
*buf)
{
        bool match = true;

        // Checking if there are changes in report since the method was last called
        for (uint8_t i=0; i<RPT_GEMEPAD_LEN; i++)
                if (buf[i] != oldPad[i])
                {
                        match = false;
                        break;
                }

        // Calling Game Pad event handler
        if (!match && joyEvents)
        {
                joyEvents->OnGamePadChanged((const GamePadEventData*)buf);

                for (uint8_t i=0; i<RPT_GEMEPAD_LEN; i++) oldPad[i] = buf[i];
        }

        uint8_t hat = (buf[5] & 0xF);

        // Calling Hat Switch event handler
        if (hat != oldHat && joyEvents)
```

```
        {
                joyEvents->OnHatSwitch(hat);
                oldHat = hat;
        }

        uint16_t buttons = (0x0000 | buf[6]);
        buttons <<= 4;
        buttons |= (buf[5] >> 4);
        uint16_t changes = (buttons ^ oldButtons);

        // Calling Button Event Handler for every button changed
        if (changes)
        {
                for (uint8_t i=0; i<0x0C; i++)
                {
                        uint16_t mask = (0x0001 << i);

                        if (((mask & changes) > 0) && joyEvents)
                                if ((buttons & mask) > 0)
                                        joyEvents->OnButtonDn(i+1);
                                else
                                        joyEvents->OnButtonUp(i+1);
                }
                oldButtons = buttons;
        }
}

void JoystickEvents::OnGamePadChanged(const GamePadEventData *evt)
{
    X=evt->X;
    Y=evt->Y;
    Z1=evt->Z1;
    Z2=evt->Z2;
    Rz=evt->Rz;
}

void JoystickEvents::OnHatSwitch(uint8_t hat)
{
        Serial.print("Hat Switch: ");
        PrintHex<uint8_t>(hat);
        Serial.println("");
}

void JoystickEvents::OnButtonUp(uint8_t but_id)
{
        Serial.print("Up: ");
        Serial.println(but_id, DEC);
}
```

```
void JoystickEvents::OnButtonDn(uint8_t but_id)
{
        Serial.print("Dn: ");
        Serial.println(but_id, DEC);
}
```

Листинг 28.4

```
#include <avr/pgmspace.h>
#include <avrpins.h>
#include <max3421e.h>
#include <usbhost.h>
#include <usb_ch9.h>
#include <Usb.h>
#include <usbhub.h>
#include <avr/pgmspace.h>
#include <address.h>
#include <hid.h>
#include <hiduniversal.h>
#include "hidjoystickrptparser.h"
#include <printhex.h>
#include <message.h>
#include <hexdump.h>
#include <parsetools.h>

USB   Usb;
USBHub Hub(&Usb);
HIDUniversal Hid(&Usb);
JoystickEvents JoyEvents;
JoystickReportParser Joy(&JoyEvents);

#include <SPI.h>
#include <Mirf.h>
#include <nRF24L01.h>
#include <MirfHardwareSpiDriver.h>
#define MAX_BUFF 32 // Буфер приема-передачи

void setup()
{
 Serial.begin( 115200 );
 Serial.println("Start");
 if (Usb.Init() == -1)
   Serial.println("OSC did not start.");
 delay( 200 );
 if (!Hid.SetReportParser(0, &Joy))
   ErrorMessage<uint8_t>(PSTR("SetReportParser"), 1 );
```

```
 Mirf.spi = &MirfHardwareSpi;
 Mirf.init();
 Mirf.setRADDR((byte *)"defender"); // Здесь задаем адрес
 Mirf.payload = MAX_BUFF; // Здесь задаем буфер
 Mirf.channel = 10;
// Это канал приема-передачи - должен
// быть одинаковым у устройств.
 Mirf.config();
 Serial.println("Start..");
}
char buff[MAX_BUFF];
int c_count = 0;

void loop()
{
  Usb.Task();
   buff[0]=map(JoyEvents.X,0,255,1,100);
   buff[1]=map(JoyEvents.Y,0,255,1,100);;
   buff[2]=map(JoyEvents.Z1,0,255,1,100);
   buff[3]=map(JoyEvents.Z2,0,255,1,100);
   buff[4]=JoyEvents.Rz+1;
   buff[5]=0;
  Mirf.setTADDR((byte *)"automobile1"); //Адрес!
  Serial.print(">");
  Mirf.send((uint8_t *)buff);
  while(Mirf.isSending()){
  }
  Serial.println(buff);
  delay(300);
}
```

Теперь необходимо написать скетч приема данных и отправки команд двигателям робота. Первый байт из буфера данных: влево (0–50) — вправо (51–100), второй байт: вперед (51–100) — назад (1–50). Надо только правильно перевести байты в данные для микросхемы L293. Полученный скетч представлен в листинге 28.5.

Листинг 28.5

```
#include <SPI.h>
#include <Mirf.h>
#include <nRF24L01.h>
#include <MirfHardwareSpiDriver.h>
#define MAX_BUFF 32 // Буфер приема-передачи

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

```
 Mirf.spi = &MirfHardwareSpi;
 Mirf.init();

 Mirf.setRADDR((byte *)"automobile1"); // Здесь задаем адрес
 Mirf.payload = MAX_BUFF; // Здесь задаем буфер
 Mirf.channel = 10;
// Это канал приема-передачи - должен
// быть одинаковым у устройств.
 Mirf.config();

 // настраиваем выводы для моторов
 pinMode(3, OUTPUT);
 pinMode(4, OUTPUT);
 pinMode(5, OUTPUT);
 pinMode(6, OUTPUT);
 Serial.println("Start..");
}

char buff[MAX_BUFF];

void loop(){
 delay(10);
 //receiving
 if(Mirf.dataReady()) {
  Mirf.getData((uint8_t *)buff);
  int fb1=buff[0];
  int lr=buff[1];
  go12(fb1,lr1);
 }
 delay(100);
}
//
void go12(int fb,int lr)
 {
 // вперед-назад
 if(fb>50)
  {digitalWrite(3,HIGH);digitalWrite(4,LOW);
   digitalWrite(5,HIGH);digitalWrite(6,LOW);}
 else if(fb<50)
  {digitalWrite(3,LOW);digitalWrite(4,HIGH);
   digitalWrite(5,LOW);digitalWrite(6,HIGH);}
 else
  {digitalWrite(3,LOW);digitalWrite(4,LOW);
   digitalWrite(5,LOW);digitalWrite(6,LOW);}
 // влево-вправо
 int lr1=map(abs(50-lr),1,50,1,255);
 int fb1=map(abs(50-fb),1,50,1,255);
```

```
if(lr<50)
 {int left=min(255,max(0,fb1-lr1/2));
 int right=min(255,fb1+lr1/2)}
else
 {int right=min(255,max(0,fb1-lr1/2));
 int left=min(255,fb1+lr1/2)}
analogWrite(9, left);
analogWrite(10, right);
}
```

Полностью данные скетчи можно найти в папках examples/_28_1 и examples/_28_2 сопровождающего книгу электронного архива.

28.6. Подключение к Arduino Android-устройства через USB Host Shield

В 2011 г. корпорация Google представила стандарт Android Open Accessory (AOA), который обеспечивает взаимосвязь между внешними USB-устройствами (Arduino-подобными платами и др.) и Android-устройством. Начиная с версии Android 3.1 (поддержка AOA портирована и в Android 2.3.4) по замыслу Google все устройства должны поддерживать USB-соединение в режиме "accessory mode". В этом режиме подключенное устройство (к примеру плата Arduino) является хостом (в т. ч. питает шину 5 В/500 мА), а Android-устройство — периферией.

Данная взаимосвязь между устройством на базе Android и Arduino-подобными платами открывает массу возможностей для разработчика и радиолюбителя: управление сервоприводами, двигателями, индикацией с Android-телефона, считывание состояния датчиков на Android-телефоне или планшете — таких, как гироскоп, акселерометр, компас, GPS, передача данных через модуль GPRS/HDSPA планшета и многое другое.

ГЛАВА 29

Arduino и ROS

29.1. Что такое ROS?

ROS (Robot Operating System, операционная система для роботов) — это структура программной системы (фреймворк), предоставляющая функционал для распределенной работы по программированию роботов. ROS (под названием Switchyard) была первоначально разработана в 2007 году в Лаборатории искусственного интеллекта Стэнфордского университета.

При разработке робота обычно приходится реализовывать свою архитектуру, свой протокол обмена сообщениями, драйвер пульта управления, логику навигации и пр. И даже если имеется возможность использовать для этих задач различные готовые библиотеки, то все равно остается серьезная проблема — объединить их для робота в единую систему. Разработчики ROS позиционируют свою систему именно как операционную — для программ взаимодействия и управления роботом ROS играет роль операционной системы, предоставляя программам управления свои интерфейсы, библиотеки и готовые приложения. ROS работает под уже готовой ОС (Ubuntu Linux), в которой реализует свой дополнительный слой абстракции — конкретно для управления роботами. ROS обеспечивает стандартные службы операционной системы, такие как аппаратную абстракцию, низкоуровневый контроль устройств, реализацию часто используемых функций, передачу сообщений между процессами и управление пакетами.

ROS развивается в двух направлениях: в качестве уже описанной здесь операционной системы и в виде поддерживаемых пользователями пакетов (ros-pkg), организованных в наборы (*стеки*), реализующие различные функции робототехники. Так, ROS содержит вспомогательные библиотеки и приложения для роботов: преобразование систем координат, утилиты для визуализации данных и распознавания объектов, стек навигации и многое другое. Реализованы для ROS и драйверы, позволяющие единым образом работать со многими устройствами: джойстиками, устройствами GPS, камерами, лазерными дальномерами и пр.

ROS основан на архитектуре *графов*, где обработка данных происходит в узлах, которые могут получать и передавать между собой сообщения (структурированные

данные). Комбинируя готовые узлы ROS и, по необходимости, дописывая собственные, можно существенно сократить время разработки и позволить себе сконцентрироваться только на тех задачах, которые действительно нужно решить.

На данный момент под управлением ROS работает уже много роботов. Вот неполный список: PR2, TurtleBot, PR1, HERB, STAIR I и II, Nao, Husky A200, iRobot Create, Lego Mindstorms NXT.

ROS выпускается в соответствии с условиями лицензии BSD и с открытым исходным кодом. Она бесплатна для использования как в исследовательских, так и в коммерческих целях. Пакеты из ros-pkg распространяются на условиях различных открытых лицензий.

29.2. Дистрибутивы ROS

Самым первым дистрибутивом ROS был Box Turtle — релиз от 2 марта 2010 года.

C Turtle (релиз от 2 августа 2010 года) — вторая версия дистрибутива ROS, содержащая обновление библиотек, входящих в ROS Box Turtle.

Diamondback (релиз от 2 марта 2011 года) — третий дистрибутив ROS. Содержит новые стеки, включая поддержку сенсора Kinect, стабильный релиз библиотеки `Point Cloud Library`. Diamondback стал меньше, проще и более конфигурируем, чем ROS C Turtle.

Electric Emys (релиз от 30 августа 2011 года) — четвертый дистрибутив ROS. Он содержит стабильные версии библиотек `arm_navigation` и `PCL`, а также расширяет поддержку новых платформ, таких как Android и Arduino.

ROS Fuerte Turtle (релиз от 23 апреля 2012 года) — стал пятым дистрибутивом ROS, в нем произведены значительные улучшения, облегчающие интегрирование с другими системами программного обеспечения.

29.3. Установка ROS

Шаги по установке ROS Fuerte Turtle под Ubuntu Linux расписаны на официальном сайте системы **http://www.ros.org/wiki/fuerte/Installation/Ubuntu**. Рассмотрим ее установку на компьютер с операционной системой Ubuntu 12.04 (Precise).

Добавляем адрес сервера ROS, чтобы менеджер пакетов знал откуда брать пакеты ROS:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main" >
/etc/apt/sources.list.d/ros-latest.list'
```

Получаем ключ:

```
wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
```

Обновляем список пакетов — тем самым сервер **ROS.org** будет проиндексирован:

```
sudo apt-get update
```

Отдаем команду установки ROS Fuerte (рекомендованная конфигурация Desktop-Full):

```
sudo apt-get install ros-fuerte-desktop-full
```

Разработчики ROS стремятся интегрировать в систему лучшие открытые робототехнические библиотеки, сохраняя при этом модульность системы, чтобы пользователь мог установить только те модули, которые ему действительно необходимы. Некоторые библиотеки вынесены из ROS и устанавливаются в ОС стандартным образом, что позволяет использовать эти библиотеки и без ROS. Установим необходимый нам пакет rosserial:

```
sudo apt-get install ros-fuerte-ros-comm
```

Отдельно необходимо установить и пакеты rosinstall и rosdep:

```
sudo apt-get install python-rosinstall python-rosdep
```

В начале новой сессии `bash` необходимо прописать установку переменных окружения ROS:

```
echo "source /opt/ros/fuerte/setup.bash" >> ~/.bashrc . ~/.bashrc
```

На этом установка закончена.

29.4. Узлы и темы в ROS

Узел — это исполняемый файл пакета ROS. Узлы ROS используют клиентские библиотеки ROS для связи с другими узлами. Клиентские библиотеки ROS позволяют реализовывать узлы ROS на различных языках программирования, например:

- `Rospy` — клиентская библиотека для Python;
- `Roscpp` — клиентская библиотека для C++;
- `Rosjava` — клиентская библиотека для Java.

Узлы могут публиковать сообщения по теме (publisher), а также подписаться на тему для приема сообщений (subscriber). *Сообщения* — тип данных, используемых для публикации или подписки на тему. Типы сообщений описываются в файлах сообщений msg — простых текстовых файлах с полем типа и полем имени в строке. Типы полей, которые можно использовать:

- `int8`;
- `int16`;
- `int32`;
- `int64`;
- `float32`;
- `float64`;
- `string`;
- `time`;
- `duration`;
- `other msg files`;
- `variable-length array[]`;
- `fixed-length array[C]`.

Файлы msg используются для генерации исходного кода для сообщений на разных языках. Файлы msg хранятся в подкаталоге msg каталога пакета.

Узлы могут также предоставлять или использовать службы (Service). Службы позволяют узлам послать запрос и получить ответ.

Файлы служб srv — такие же простые текстовые файлы, как и файлы msg, но они состоят из двух частей: запроса и ответа. Эти две части, разделяются линией: `- - -`. Пример файла srv:

```
int64 A
int64 B
---
int64 Sum
```

В этом примере, `A` и `B` — это запрос, а `Sum` — это ответ.

Файлы srv хранятся в подкаталоге srv каталога пакета.

29.5. Пакет *rosserial*

Библиотека `rosserial` устанавливает соединение точка-точка (point-to-point connection) через последовательный порт с недорогими контроллерами (типа Arduino) так, что вы можете посылать сообщения ROS туда и обратно.

Библиотека `rosserial` состоит из общего P2P-протокола, библиотеки для работы с Arduino и узлов для ПК.

Библиотека для работы с Arduino находится в папке проекта serial, в каталоге serial_arduino/libraries. Копируем папку ros_lib в библиотечный каталог libraries Arduino IDE (рис. 29.1). Библиотека `ros_lib` размещена также в папке libraries сопровождающего книгу электронного архива.

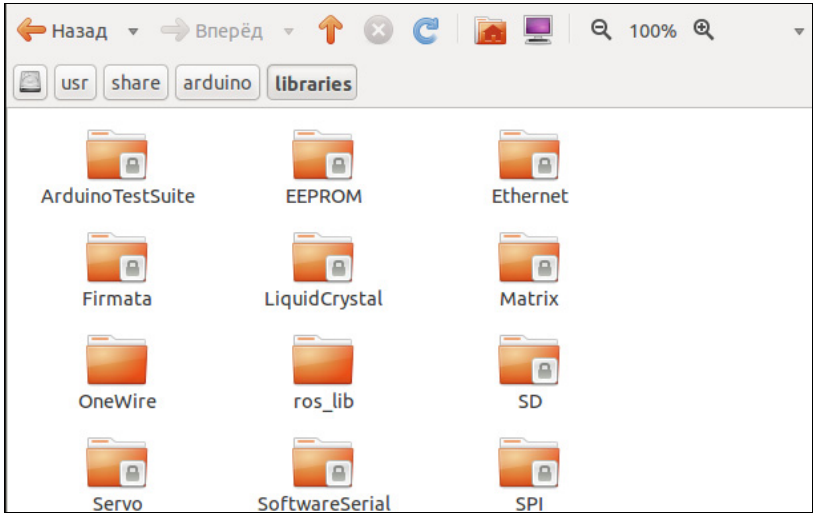


Рис. 29.1. Подключение библиотеки `ros_lib`

29.6. Подготовка сообщения (publisher) на Arduino

Создадим скетч на Arduino, демонстрирующий создание узла ROS, публикующего сообщения в тему. Соединяем Arduino с компьютером, на котором запущена ROS по последовательному порту (в рассматриваемом случае это порт `/dev/ttyUSB0`). К контроллеру подключен датчик температуры DS18B20. Будем отправлять в ROS значения температуры с датчика, используя библиотеки `OneWire.h` и `ros_lib`.

Работу с датчиком температуры, работающим по протоколу 1-Wire, мы подробно рассмотрели в предыдущих главах (см. например, *главы 11* и *12*), поэтому здесь остановимся на работе библиотеки `ros_lib`.

В каждую программу ROS Arduino необходимо включить заголовочный файл ros.h и файлы заголовков для всех типов сообщений, которые мы будем использовать, — в нашем случае это std_msgs/Float32.h:

```
#include <ros.h>
#include <std_msgs/Float32.h>
```

Далее нам необходимо создать экземпляр объекта узла `serial_node`, что позволит нашей программе выступать в качестве подписчика (subscriber), либо публиковать сообщения (publisher):

```
ros::NodeHandle nh;
```

Создаем экземпляр `publisher` для нашего узла `serial_node`, публикующий сообщения типа `std_msgs::Float32` в тему `temperature`:

```
std_msgs::Float32 float32_msg;
ros::Publisher chatter("temperature", &float32_msg);
```

В подпрограмме `setup()` необходимо инициализировать узел и объявить о роли узла `chatter` в качестве `publisher`:

```
nh.initNode();
nh.advertise(chatter);
```

В цикле `loop()` после считывания данных с датчика температуры публикуем сообщение в тему и вызываем `ros::spinOnce()`, где обрабатываются все функции обратного вызова соединения.

```
chatter.publish( &float32_msg );
nh.spinOnce();
```

Код данного скетча представлен в листинге 29.1.

Листинг 29.1

```
#include <OneWire.h>
OneWire ds(10); // линия 1-Wire будет на pin 10

#include <ros.h>
#include <std_msgs/Float32.h>
```

```
ros::NodeHandle nh;
std_msgs::Float32 float32_msg;
ros::Publisher chatter("temperature", &float32_msg);

void setup(void)
{
 nh.initNode();
 nh.advertise(chatter);
}
void loop(void)
{
 byte i;
 byte present = 0;
 byte data[12];
 byte addr[8];

 if ( !ds.search(addr)) {
   ds.reset_search();
   return;
 }
 ds.reset();
 ds.select(addr);
 ds.write(0x44,1);    // запускаем конвертацию
 delay(1000);  // скорее всего достаточно 750 ms
 present = ds.reset();
 ds.select(addr);
 ds.write(0xBE);    // считываем ОЗУ датчика

 for ( i = 0; i < 9; i++) {     // обрабатываем 9 байтов
  data[i] = ds.read();
 }
 Serial.print(" CRC=");
 Serial.print( OneWire::crc8( data, 8), HEX);
 Serial.println();
 // высчитываем температуру
 int HighByte, LowByte, Temp;
 float Tempf1,Tempf2;
 LowByte = data[0];
 HighByte = data[1];
 Temp = (HighByte << 8) + LowByte;
 Tempf1=Temp/16;
 Tempf2=(Temp%16)*100/16;
 float32_msg.data=Tempf1+Tempf2/100;
 // публикуем сообщение
 chatter.publish( &float32_msg );
 nh.spinOnce();
}
```

Теперь проверим работу данного скетча. Первое, что необходимо запустить, — это команда `roscore`. Команда `rosnode` отображает информацию об узлах ROS, которые работают в настоящий момент. Команда `rosnode list` выдает список этих активных узлов. В терминале увидим:

```
/rosout
```

Соответственно, есть только один работающий узел: `rosout`. Этот узел работает всегда, т. к. он собирает и логирует отладочные сообщения узлов.

Команда `rosrun` позволяет назначить имя пакета, чтобы непосредственно запустить его узел:

```
$ rosrun [package_name] [node_name]
```

Запускаем узел `serial_node.py` пакета `rosserial_python`, который соединяет нашу Arduino с остальной частью ROS. Необходимо выставить используемый последовательный порт:

```
rosrun rosserial_python serial_node.py /dev/ttyUSB0
```

В терминале набираем:

```
$ rosnode list
```

И смотрим список активных узлов:

```
/rosout
/serial_node
```

Утилита `rxgraph`, являющаяся частью пакета `rxtools`, позволяет визуально показать узлы и темы, запущенные в настоящий момент. Набираем в терминале:

```
$ rxgraph
```

Результат показан на рис. 29.2.

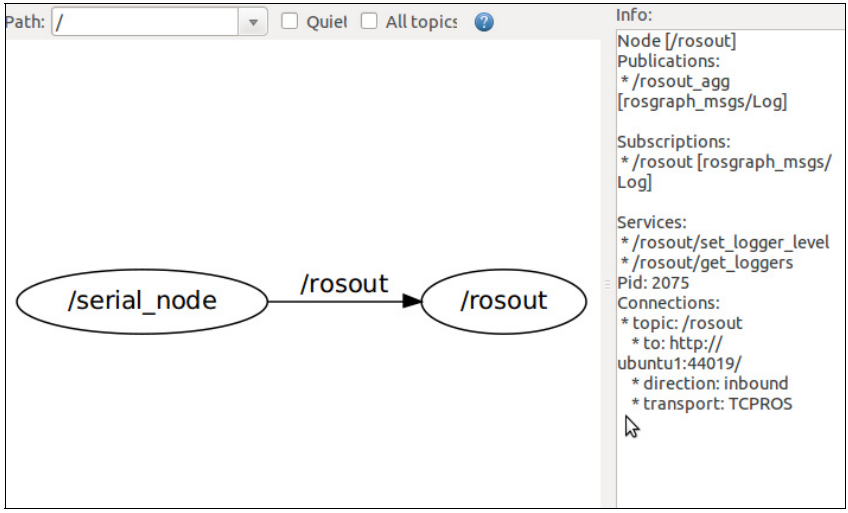


Рис. 29.2. Утилита `rxgraph` демонстрирует список активных узлов и тем

Утилита `rostopic` позволяет получить информацию о темах ROS. Команда `rostopic echo` показывает данные, опубликованные в теме. Набираем в терминале:

```
$ rostopic echo /temperature
```

и видим постоянно поступающие с Arduino данные датчика температуры (рис. 22.3).

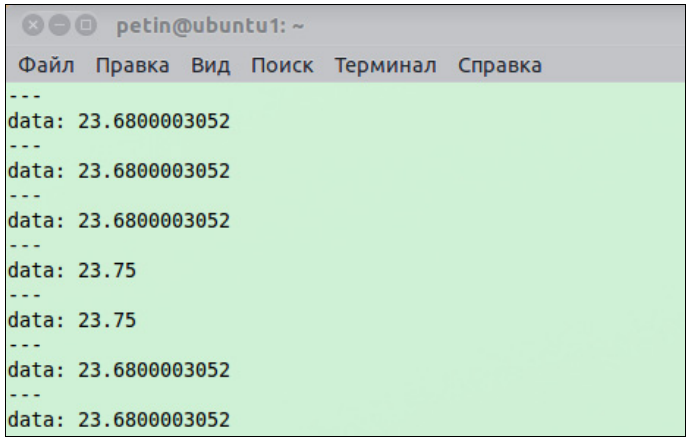


Рис. 22.3. Публикация сообщений из Arduino в тему temperature

Данный скетч находится в папке examples/_29_01 сопровождающего книгу электронного архива.

29.7. Создание подписки (subscriber) на Arduino

Теперь рассмотрим пример использования Arduino в качестве узла subscriber для приема сообщений. В этом примере мы будем включать/выключать светодиод, подключенный к выводу 13 Arduino, получая сообщения из ROS.

Включаем заголовочный файл ros.h и файлы заголовков для всех типов сообщений, которые мы будем использовать, — в нашем случае это std_msgs/Empty.h (для пустых сообщений):

```
#include <ros.h>
#include <std_msgs/Empty.h>
```

Далее необходимо создать экземпляр объекта узла `serial_node`, что позволит нашей программе выступать в качестве подписчика (subscriber), либо публиковать сообщения (publisher):

```
ros::NodeHandle nh;
```

Создаем экземпляр `subscriber` для нашего узла, публикующий пустые сообщения типа `std_msgs::Float32` в тему `toggle_led`:

```
ros::Subscriber<std_msgs::Empty> sub("toggle_led", &messageCb );
```

Создаем для нашего узла функцию обратного вызова `messageCb`. Эта функция должна постоянно получать сообщение в качестве аргумента. Для нашей функции обратного вызова `messageCb` назначим тип сообщения `std_msgs::Empty`. По получении сообщения функция инвертирует значение сигнала на выводе 13 Arduino, при этом включая/выключая светодиод.

```
void messageCb( const std_msgs::Empty& toggle_msg){
 digitalWrite(13, HIGH-digitalRead(13)); // blink the led
 }
```

В подпрограмме `setup()` необходимо инициализировать узел и объявить о роли узла в качестве подписчика на сообщения:

```
nh.initNode();
nh.subscribe(sub);
```

И наконец, в цикле `loop()` вызываем `ros::spinOnce()`, где обрабатываются все функции обратного вызова соединения:

```
nh.spinOnce();
```

Код данного скетча представлен в листинге 29.2.

Листинг 29.2

```
#include <ros.h>
#include <std_msgs/Empty.h>
ros::NodeHandle nh;
void messageCb( const std_msgs::Empty& toggle_msg){
 digitalWrite(13, HIGH-digitalRead(13)); // blink the led
 }
ros::Subscriber<std_msgs::Empty> sub("toggle_led", &messageCb );

void setup()
 {
 pinMode(13, OUTPUT);
 nh.initNode();
 nh.subscribe(sub);
 }

void loop()
 {
 nh.spinOnce();
 delay(1);
 }
```

Запускаем узел `serial_node.py` пакета `rosserial_python`, который соединяет нашу Arduino с остальной частью ROS. Необходимо выставить используемый последовательный порт (здесь использована другая плата Arduino, подключенная к порту `ttyACM0`):

```
rosrun rosserial_python serial_node.py /dev/ttyACM0
```

Переходим на компьютер с запущенной ROS и проверяем список активных узлов:

```
$ rosnode list
```

Смотрим результат:

```
/rosout
/serial_node
```

Наш узел запущен как подписчик на сообщения по теме `toggle_led`, но никаких сообщений он пока не получил. Связь по темам осуществляется путем отправки сообщений ROS между узлами. Для общения издателя и абонента издатель (publisher) и абонент (subscriber) должны отправлять и получать сообщения одинакового типа. Это означает, что тип темы определяется типом сообщений, которые в ней публикуются. Тип сообщения, отправляемого в тему, может быть определен с помощью команды `rostopic type`:

```
$ rostopic type toggle_led
```

Результат:

```
std_msgs/Empty
```

Теперь используем `rostopic` с сообщениями — `rostopic pub` публикует данные в тему:

```
rostopic pub [topic] [msg_type] [args]
```

Отправим единичное сообщение:

```
rostopic pub toggle_led std_msgs/Empty --once
```

Светодиод должен изменить значение на противоположное.

Для отправки сообщения в цикле (`-r`) с определенной частотой введем команду:

```
rostopic pub toggle_led std_msgs/Empty -r 1
```

Эта команда публикует сообщение с частотой 1 Гц в тему `toggle_led`. Светодиод будет мигать с частотой 2 раза в секунду.

Данный скетч находится в папке examples/_29_02 сопровождающего книгу электронного архива.

29.8. Связь через ROS двух плат Arduino

Теперь создадим пример передачи сообщений через ROS между двумя платами Arduino. На одной плате Arduino, соединенной с ROS, находится датчик температуры DS18B20 (см. *разд. 29.6*). Подключим к ROS по другому последовательному порту вторую плату Arduino, к которой подключен дисплей WH0802. На него мы и будем выводить показания температуры с датчика, расположенного на первой плате Arduino. Скетч для публикации показаний температуры у нас уже есть. Скетч для получения сообщений с показаниями температуры, публикуемыми в тему `temperature`, представлен в листинге 29.3.

Листинг 29.3

```
// подключить библиотеку LiquidCrystal
#include <LiquidCrystal.h>
// создание экземпляра объекта LiquidCrystal
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

// rosserial
#include <ros.h>
#include <std_msgs/Empty.h>
#include <std_msgs/Float32.h>

ros::NodeHandle nh;
void messageCb( const std_msgs::Float32& toggle_msg){
 digitalWrite(13, HIGH-digitalRead(13)); // blink the led
 lcd.setCursor(0, 0);
 lcd.print("Temp=");
 lcd.setCursor(0, 1);
 lcd.print(toggle_msg.data);}

ros::Subscriber<std_msgs::Float32> sub("temperature", &messageCb );

void setup() {
 lcd.begin(8, 2);
 pinMode(13, OUTPUT);
 nh.initNode();
 nh.subscribe(sub);
}

void loop() {
 nh.spinOnce();
 delay(1000);
}
```

Данный скетч находится в папке examples/_29_03 сопровождающего книгу электронного архива.

Теперь посмотрим, как реализовать передачу в ROS. Запустить два узла `serial_node` с одним именем не получится. Одна из особенностей ROS состоит в том, что вы можете переназначить имена (`Names`) узлов из командной строки:

```
$ rosrun rosserial_python serial_node.py /dev/ttyUSB0 __name:=serial1
$ rosrun rosserial_python serial_node.py /dev/ttyACM0 __name:=serial2
```

Теперь посмотрим список активных узлов командой `rosnode_list`:

```
/rosout
/serial1
/serial2
```

А командой `rxgraph` посмотрим узлы и темы (рис. 29.4). При этом показания температуры отображаются на дисплее WH0802.

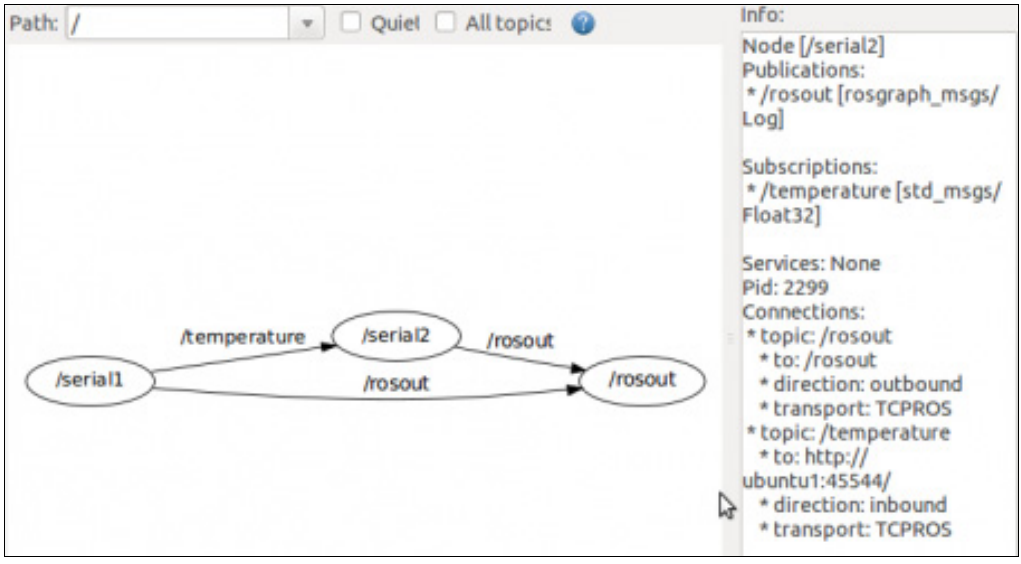


Рис. 29.4. Утилита `rxgraph` — список активных узлов и тем

ГЛАВА 30

Arduino и "умный дом" X10

Услышав словосочетание "умный дом", мы тут же представляем себе роскошный особняк миллионера, в котором холодильник заказывает продукты через Интернет, ужин разогревается сам, комнаты реагируют включением освещения на любое слово или взмах руки, а сам дом охраняется и контролируется удаленно. Тем не менее, если не гнаться за подобной фантастикой, вполне реально устроить в своей квартире филиал "умного дома". И конечно же, рассмотрим применение Arduino в этом деле.

30.1. Система домашней автоматизации X10

Существует несколько стандартов домашней автоматизации. Самой первой системой домашней автоматизации является X10, которая использует для передачи данных метод частотного уплотнения в обычной электросети квартиры. Плюсы данной технологии:

- ❒ легка в установке;
- ❒ никаких дополнительных проводов в квартире;
- ❒ распространенный стандарт, много исполнительных элементов;
- ❒ система может управляться дистанционно;
- ❒ система может программироваться;
- ❒ относительно низкая стоимость и простота компонентов;
- ❒ не требует дополнительного электропитания.

В системе X10 есть два основных компонента:

- ❒ модуль — принимает сигналы от X10-трансивера через электропроводку и управляет устройством, подключенным к нему;
- ❒ контроллер — посылает сигналы модулям.

Модули бывают разными. На рис. 30.1 показан стандартный модуль для управления светом. Модули также могут быть вмонтированы в электропроводку или встроены в прибор.

Рис. 30.1. Модуль LM12

Рис. 30.2. Трансивер X10

Существуют три типа контроллеров:

- трансивер (рис. 30.2) — подключается к розетке сети переменного тока, принимает сигналы от беспроводного пульта дистанционного управления (рис. 30.3) и отправляет команды модулям;

Рис. 30.3. Пульт управления X10

- настольный контроллер — подключается к настенной розетке сети переменного тока и посылает команды модулям по сети;
- универсальный пульт дистанционного управления — способен посылать как обычные инфракрасные сигналы, так и беспроводные сигналы формата X10.

Каждый модуль имеет два настроечных параметра: код группы (`A-P`) и код устройства (`1-16`). Существуют следующие правила конфигурации системы X10:

- все модули, управляемые одним трансивером или контроллером, должны использовать одинаковый код группы, переключатель кода группы имеет позиции A–P;
- трансивер или контроллер должны быть сконфигурированы так, чтобы использовать тот же самый код группы, что и модули, которыми они управляют;
- пульт дистанционного управления должен использовать тот же самый код группы, который использует трансивер и модули;
- каждый модуль, которым вы хотите управлять отдельно от остальных, должен иметь уникальный код устройства `1-16` (эти коды не обязательно должны быть последовательными).

Управлять системой X10 можно дистанционно, например с помощью телефонного контроллера или компьютерного интерфейса. Для объединения системы X10 с другими системами служит контроллер TW523. Его мы и будем использовать для управления приборами X10 с помощью Arduino.

30.2. Двусторонний интерфейс TW523

Отдельное направление развития технологий X10 — создание сторонними разработчиками и OEM-производителями собственных устройств (микрокомпьютерных систем) для управления модулями X10. Такие устройства подключаются к системам X10 с помощью специальных интерфейсных модулей. Примером такого модуля может служить TW523 (рис. 30.4), который вставляется в обычную электро-

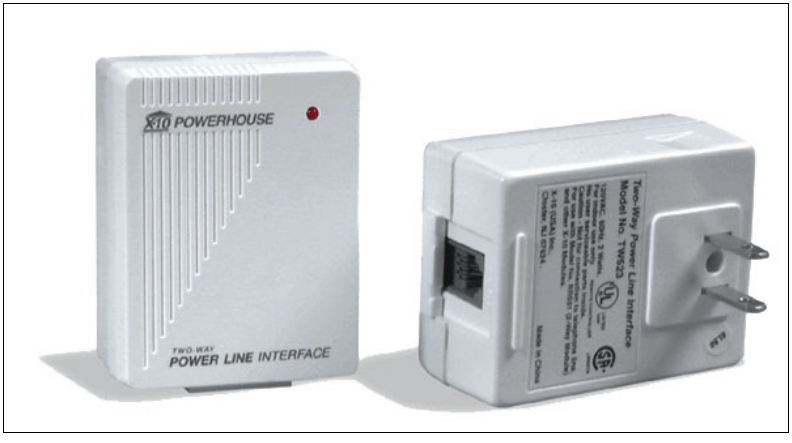


Рис. 30.4. Двусторонний интерфейс TW523

розетку, а внешние устройства подключаются к нему через оптически развязанный интерфейс с разъемом RJ11. Модуль TW523 преобразует генерируемые управляющим устройством команды в сигналы, передаваемые по электропроводке к модулям X10. Этот модуль может передавать сигналы и в обратном направлении — от устройств X10 к управляющим компьютерам.

30.3. Arduino-библиотека *X10*

Для взаимодействия с модулем TW523 написана библиотека `X10`, которую можно скачать по ссылке **https://github.com/tigoe/x10/downloads**. На данный момент доступна версия 0.4, совместимая с Arduino 1.0. Скачиваем архив и распаковываем его в папку libraries каталога установки Arduino. Файлы библиотеки вы также можете найти в папке libraries/X10 сопровождающего книгу электронного архива. В библиотеке реализована только отправка команд в модуль TW523.

Набор функций библиотеки:

- `begin()`;
- `write()`.

30.3.1. Функция *begin()*

Функция `begin()` инициализирует библиотеку `X10`.

Синтаксис:

```
begin(rxPin, txPin, zeroCrossingPin)
```

Параметры: `rxPin`, `txPin`, `zeroCrossingPin` — см. схему подключения (см. рис. 20.5). Сейчас `rxPin` не используется — зарезервирован для функций получения данных из сети X10.

Возвращаемое значение: экземпляр объекта `X10`.

30.3.2. Функция *write()*

Функция `write()` отправляет данные в сеть X10.

Синтаксис:

```
write(byte houseCode, byte numberCode, int numRepeats)
```

Параметры:

- `byte houseCode` — код группы;
- `byte numberCode` — код модуля;
- `int numRepeats` — количество повторений (отправок кода).

Возвращаемое значение: экземпляр объекта `X10`.

Схема подключения Arduino к модулю по кабелю RJ11 представлена на рис. 30.5.

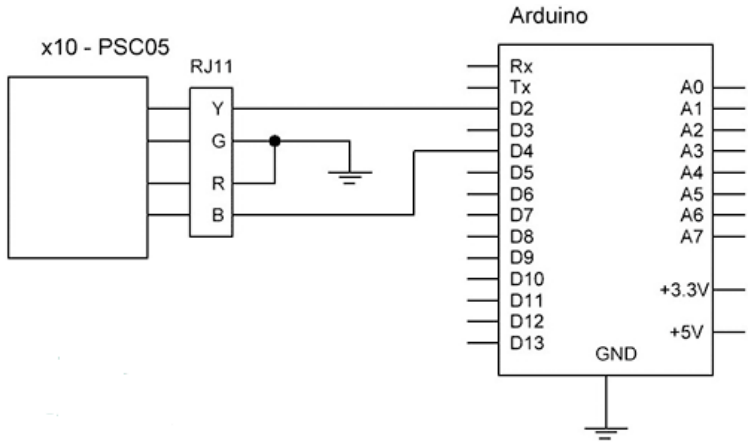


Рис. 30.5. Подключение Arduino к модулю TW523

Создадим пример управления двумя ламповыми модулями с кодами `A2` и `A4`. Скетч каждые 5 секунд попеременно включает/выключает модули. Содержимое скетча представлено в листинге 30.1.

Листинг 30.1

```
// подключение библиотеки X10
#include "x10.h"
#include "x10constants.h"

#define zcPin 2 // zero crossing pin
#define dataPin 4 // X10 data pin
#define repeatTimes 1 // количество повторов отправки команды

x10 myHouse = x10(zcPin, dataPin);

void setup() {
 // Выключить все
 myHouse.write(HOUSE_A, ALL_UNITS_OFF,repeatTimes);
}

void loop() {
 // Включение A2
 myHouse.write(HOUSE_A, UNIT_2,repeatTimes);
 myHouse.write(HOUSE_A, ON,repeatTimes);
 // Выключение A4
 myHouse.write(HOUSE_A, UNIT_4,repeatTimes);
 myHouse.write(HOUSE_A, OFF,repeatTimes);
 delay(5000);
 // Выключение A2
 myHouse.write(HOUSE_A, UNIT_2,repeatTimes);
 myHouse.write(HOUSE_A, OFF,repeatTimes);
```

```
 // Включение A4
 myHouse.write(HOUSE_A, UNIT_4,repeatTimes);
 myHouse.write(HOUSE_A, ON,repeatTimes);
 delay(5000);
}
```

30.4. Блок на Arduino для голосового управления приборами X10

Здесь использована связка модуля TW523 и Arduino для проекта голосового управления приборами X10 для "умного дома" на ROS. Проект можно посмотреть в Сети по следующим адресам:

- **http://cxem.net/house/1-310.php**;
- **http://cxem.net/house/1-311.php**;
- **http://cxem.net/house/1-312.php**;
- **http://cxem.net/house/1-314.php**;
- **http://cxem.net/house/1-326.php**.

Соединение Arduino с ROS мы рассматривали в *главе 29*. Система на ROS распознает голосовую фразу типа "светильник включить", "лампу выключить", "елочка гори" и отправляет сообщение типа X10 (три параметра: код группы, код устройства и количество повторений команды) в тему ROS. Слушателем темы является плата Arduino, соединенная по последовательному порту с компьютером. Arduino, получая сообщения из ROS, отправляет их далее в прибор TW523, который преобразует генерируемые управляющим устройством команды в сигналы, передаваемые по электропроводке к модулям X10. Для пользовательского типа сообщений `X10` в ROS сгенерируем заголовочный файл X10.h, который поместим в папку libraries/ros_lib/vp_x10_voice.

Содержимое скетча представлено в листинге 30.2.

Листинг 30.2

```
// подключение библиотеки X10
#include <x10.h>
#include <x10constants.h>

#define zcPin 2    // zero crossing pin
#define dataPin 4   // X10 data pin
#define repeatTimes 1 // количество повторов отправки команды
x10 myHouse = x10(zcPin, dataPin);

// подключение библиотеки ros_lib
#include <ros.h>
#include <vp_x10_voice/X10.h>
```

```
ros::NodeHandle nh;

vp_x10_voice::X10 x10_msg;
ros::Publisher pub_x10("data_from_x10", &x10_msg);

void messageCb( const vp_x10_voice::X10& toggle_msg){
 digitalWrite(13, HIGH-digitalRead(13));
 x10_msg.command1 = toggle_msg.command1;
 x10_msg.command2 = toggle_msg.command2;
 x10_msg.repeatTime = toggle_msg.repeatTime;
 pub_x10.publish(&x10_msg);
 int command1=toggle_msg.command1;
 int command2=toggle_msg.command2;
 int repeatTime=toggle_msg.repeatTime;
 myHouse.write(command1, command2,repeatTime);
 }
ros::Subscriber<vp_x10_voice::X10> sub_x10("data_to_x10", &messageCb );


void setup() {
 Serial.begin(9600);
 nh.initNode();
 nh.subscribe(sub_x10);
 nh.advertise(pub_x10);
}

void loop() {
 nh.spinOnce();
 delay(1000);
}
```

Данный скетч находится в папке examples/_30_1 сопровождающего книгу электронного архива.

ПРИЛОЖЕНИЕ 1

Список использованных источников

- **http://www.rlocman.ru/review/article.html?di=111906** — Дэвид Кушнер "Как разрабатывали и продвигали Arduino", "РАДИОЛОЦМАН", ноябрь 2011 г.
- **http://www.arduino.cc** — официальная документация проекта Arduino.
- **http://cxem.net** — авторские материалы с сайта "Паяльник".

ПРИЛОЖЕНИЕ 2

Описание электронного архива

Электронный архив с материалами, сопровождающими книгу, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977533379.zip**, а также со страницы книги на сайте **www.bhv.ru**.

В архиве находятся следующие папки:

- \examples — исходники примеров и проектов *глав 7–30* для Arduino IDE;
- \examples\Android — архивы с файлами проектов для среды Eclipse для Android из *глав 14* и *24*;
- \libraries — библиотеки Arduino, используемые в примерах и проектах книги, и не включенные в среду разработки Arduino IDE;
- \datasheets — документация производителей (data sheet) на рассматриваемые в книге микросхемы и модули;
- \cxem — электрические схемы в формате spl7.