Электроника

Дхананья Гадре
Нигул Мэлхотра

bhv

Занимательные проекты на базе микроконтроллеров tinyAVR

- 34 полноценных работающих проекта
- Пошаговые инструкции и доступный для скачивания исходный код
- Варианты дизайна и настройки проектов

Mc Graw Hill

УДК 621.382
ББК 32.85
Г13

Гадре, Д.

Г13 Занимательные проекты на базе микроконтроллеров tinyAVR / Дхананья Гадре, Нигул Мэлхотра: Пер. с англ. — СПб.: БХВ-Петербург, 2012. — 352 с.: ил. — (Электроника)

ISBN 978-5-9775-0728-8

На 34 занимательных практических примерах рассмотрены разработка и программирование электронных устройств на основе микроконтроллеров tinyAVR компании Atmel. Описаны устройство микроконтроллеров, их архитектура, электронные компоненты проектов и вопросы питания Рассмотрены инструменты для создания проектов и изготовления печатных плат, основы программирования и основные команды языка С для встроенных приложений. Приведены проекты с использованием светодиодов, графических дисплеев, датчиков, аудиопроекты и проекты на альтернативных источниках энергии. Материал сопровождается пошаговыми инструкциями, рисунками и фотографиями Приведены интернет-ссылки на исходные коды рассмотренных проектов.

Для радиолюбителей

УДК 621.382
ББК 32.85

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав редакцией	*Григорий Добин*
Перевод с английского	*Андрея Лашкевича*
Редактор	*Леонид Кочин*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Оформление обложки	*Марины Дамбиевой*
Зав производством	*Николай Тверских*

Подписано в печать 30.09.11.
Формат 70×100 1/16. Печать офсетная Усл печ. л 20,32.
Тираж 2000 экз Заказ № 1291
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77 99 60.953 Д.005770 05.09 от 26.05 2009 г выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12.

ISBN 978-0-07-174454-6 (англ.)
ISBN 978-5-9775-0728-8 (рус.)

Оглавление

Посвящается профессору Shailaja M. Karandikar (Шайлая Карандикару) (1920–1995), в просторном доме которого я всегда мог посмотреть и взять почитать любую книгу из его личной библиотеки, и профессору Нилу Гершенфельду (Neil Gershenfeld), который сделал возможным появление этой книги!

Дананья Гадре

Моим родителям, давшим мне индивидуальность, и моей сестре Ниха (Neha), которая так на меня похожа!

Нигул Мэлхотра

Об авторах

Дананья Гадре (Нью-Дели, Индия) получил степень магистра электронных наук в Университете Дели, а магистра по компьютерным технологиям — в Университете Айдахо. За 21 год своей профессиональной деятельности он преподавал в колледже SGTB Khalsa College, университете Дели, работал научным сотрудником в центре Inter University Centre for Astronomy and Astrophysics (IUCAA), а с 2001 года работает на факультете Electronics and Communication Engineering Division института Netaji Subhas Institute of Technology, Нью-Дели (в настоящее время на должности адъюнкт-профессора). Он состоит в международном сообществе Fablab и избран от своего факультета членом академии Fab Academy. Профессор Гадре является автором нескольких статей и трех книг. Одна из этих книг была переведена на китайский язык, а другая — на греческий. Он имеет лицензию радиолюбителя и позывной VU2NOX и надеется когда-нибудь спроектировать и собрать радиолюбительский спутник.

Нигул Мэлхотра (Нью-Дели, Индия) получил высшее образование по электронике и коммуникационным технологиям в институте Netaji Subhas Institute of Technology (Нью-Дели). Он работал в лаборатории профессора Гадре и энергично участвовал во многих проектах. Он также ведет учебный курс под названием LearnMicros. Однажды Нигул освободил джинна из найденной на морском берегу бутылки. В качестве вознаграждения он получил 30 часов в сутках. В настоящее время Нигул является аспирантом в институте Indian Institute of Management (Ахмедабад, Индия).

Благодарности

Мы начали работать с микроконтроллерами tinyAVR несколько лет назад. Создавать устройства на базе микроконтроллеров с ограниченным набором функций было очень интересно. Постепенно число проектов росло, и мы решили рассказать о них другим людям. Результатом стала эта книга.

Разрабатывать наши проекты помогали многие студенты: Anurag Chugh, Saurabh Gupta, Gaurav Minocha, Mayank Jain, Harshit Jain, Hashim Khan, Nipun Jindal, Prateek Gupta, Nikhil Kautilya, Kritika Garg и Lalit Kumar. Как всегда, большую помощь в изготовлении многих устройств нам оказал Satya Prakash из центра Centre for Electronics Design and Technology (CEDT).

Первоначально мы собирали схемы на макетных печатных платах либо заказывали печатные платы у сторонних изготовителей. Начиная с 2008 года, когда профессор Нил Гершенфельд из центра Center for Bits and Atoms (Media Labs) института Massachusetts Institute of Technology подарил нам станок MDX20, самостоятельное изготовление печатных плат значительно ускорилось и облегчилось. При помощи MDX20 мы можем выполнить макет схемы всего за несколько часов (а не за неделю, как это было раньше). Мы благодарим Нила Гершфельда за его щедрую помощь и множество сделанных им предложений. За поддержку наших усилий мы благодарим менеджера Sherry Lassiter из центра Center for Bits and Atoms.

С образцами устройств и инструментами нам помогали Lars Thore Aarrestaad, Marco Martin Joaquim и Imran Shariff из компании Atmel.

Я благодарю также Roger Stewart — главного редактора издательства McGraw-Hill — за его поддержку идеи этой книги, а Joya Anthony — координатора — за то, что он был настойчив, но мягок даже тогда, когда все сроки уже вышли. Большую работу по редактированию нашей книги проделал Vaishnavi Sundararajan. Спасибо вам, ребята!

Нигул Мэлхотра — студент, участвовавший в нескольких проектах, — внес существенный вклад в содержание книги и стал соавтором. Его настойчивость и способность работать прилежно и долго достойны подражания со стороны коллег.

Эта книга была бы невозможна без поддержки членов моей семьи Sangeeta и Chaitanya — они самые важные люди в моей жизни. Спасибо за ваше терпение и веру в успех!

Более десяти лет назад, когда я писал книгу о микроконтроллерах AVR, они были мало распространены, и знало об этих микросхемах совсем немного людей. Я решил опробовать эти новые устройства, поскольку мне надоели микроконтроллеры 8051, не обладавшие возможностями, достаточными для сложных приложений. Несмотря на то, что микросхемы AVR были новинками, программные инструменты компании Atmel оказались очень надежными и эффективными: за несколько дней я прочитал буквально все об этих микросхемах и запрограммировал свое первое приложение. Эти контроллеры только что появились, подходящих языков высокого уровня для них еще не было, поэтому все проекты в той книге были запрограммированы на языке ассемблера.

В настоящее время все совсем по-другому. Семейство микроконтроллеров AVR всем хорошо известно и на данный момент занимает второе место по объему продаж в мире! Для контроллеров AVR появилось множество качественных компиляторов с языка C. Семейство AVR поддерживается также компилятором AVRGCC, распространяемым по лицензии GNU, т. е. вам не придется тратить деньги на компилятор языка C.

Когда я начал использовать AVR более десяти лет назад, мое внимание привлекли несколько 8-контактных микросхем. До того момента интегральная схема с восемью выводами была либо операционным усилителем 741, либо таймером 555. А здесь в аналогичном корпусе помещался целый компьютер. Было просто восхитительно видеть такие маленькие компьютеры и еще интереснее — работать с ними. В течение всех последующих лет я не переставал восхищаться этими микросхемами. Компания Atmel тоже не сидела сложа руки — она расширила данную серию и дала ей новое название: микроконтроллеры tinyAVR, а также добавила множество новых микросхем (от 6- до 28-контактной). Это дешевые устройства, которые при оптовой покупке стоят по 25 центов за штуку.

Сегодня микроконтроллеры можно встретить везде, от пультов дистанционного управления телевизорами и микроволновых печей до мобильных телефонов. Чтобы научиться программировать и использовать эти устройства, были созданы самые разнообразные обучающие инструменты, наборы и среды. Одна из таких популярных сред — Arduino — основана на семействе микроконтроллеров AVR и предлагает

свой собственный язык, который изучить очень легко — можно начать пользоваться устройствами Arduino буквально за один день (вместо трудоемкого освоения ассемблера или C). Эта микроконтроллерная система специально разработана для начинающих. Самая простая платформа Arduino использует 28-контактный AVR микроконтроллер ATMega8 и стоит от 12 долларов. Однако если вы хотите управлять лишь несколькими светодиодами или для вашего проекта нужна всего пара контактов ввода/вывода, зачем вам 28-контактная микросхема? Тогда добро пожаловать в мир микроконтроллеров tinyAVR!

В этой книге описаны 34 законченных, полностью работоспособных устройства. Все проекты были реализованы на микроконтроллерах серии tinyAVR. Рассмотрено шесть тем: простые устройства с применением светодиодов, более сложные проекты со светодиодами, проекты с графическим дисплеем, проекты с датчиками, аудиопроекты и устройства на альтернативных источниках энергии. Некоторые из конструкций уже стали популярными и выпускаются в виде готовых продуктов. Поскольку в этой книге описаны все подробности этих проектов, она служит хорошим источником идей как для профессионалов, так и для энтузиастов-любителей. Описанные решения, безусловно, можно улучшить. Принципиальные схемы и файлы печатных плат для всех проектов доступны и пригодны для заказа печатных плат у изготовителей. Бóльшую часть компонентов можно заказать через Digikey или Farnell.

Схемы и файлы печатных плат для всех проектов, а также видеоруководства и фотографии есть на нашем сайте: **www.avrgenius.com/tinyavr1**.

В этой книге шесть глав и три приложения. Каждая глава посвящена определенной теме (например, светодиодам или датчикам). Главы можно читать в произвольном порядке. Однако если вы новичок, то мы рекомендуем вам изучать главы по порядку. В *главе 1* имеется вводная информация по разработке проектов, инструментам, источникам питания и т. д., поэтому мы настоятельно рекомендуем ее прочитать (даже если у вас есть необходимый опыт), чтобы получить представление о процессе разработки устройств, описанном в последующих главах.

В *главе 1* дан обзор устройств семейства Tiny. Рассмотрены следующие общие вопросы: архитектура tinyAVR, основные функциональные возможности микроконтроллеров tinyAVR, проектирование для микроконтроллеров, создание источника питания для портативных приложений, инструменты для изготовления устройств и печатных плат, начала программирования микроконтроллеров.

Глава 2 посвящена проектам с использованием светодиодов. Здесь рассмотрены типы светодиодов, их характеристики и управление светодиодами. Описаны четыре простые устройства: светодиодная свеча, смешивание цветов RGB-светодиодов, генератор случайных цветов и музыки, а также светодиодное перо.

В *главе 3* приведены более сложные конструкции со светодиодами. Здесь рассмотрены способы управления большим количеством светодиодов при помощи различных методов мультиплексирования. Описаны восемь проектов: лампа для создания настроения, волюметр с дисплеем из 20 светодиодов, вольтметр, частотомер с автоматической настройкой диапазона, термометр (с отсчетом в градусах

Цельсия и Фаренгейта), забавные часы, игральные кости и игра в крестики-нолики со светодиодами.

Глава 4 содержит проекты с графическим жидкокристаллическим дисплеем. Здесь описано управление жидкокристаллическими дисплеями, типы дисплеев, рассмотрен графический дисплей Nokia 3310. Приведено шесть проектов: регистратор температуры, игра Тэнгу, игра "Жизнь", крестики-нолики, забавные часы и школьный звонок.

В *главе 5* описаны различные типы датчиков света, температуры и магнитного поля, их работа и рассмотрены восемь оригинальных устройств: светодиод как датчик и индикатор, валентинка с датчиком близости, электронная спичка без огня, вращающийся волчок со светодиодами, бесконтактный тахометр, индуктивный датчик обнаружения и подсчета числа автомобилей, электронные свечи для дня рождения и сигнализация для холодильника.

В *главе 6* изложены способы генерирования музыки и звуков при помощи микроконтроллера и представлены четыре аудиопроекта: тональный генератор, усовершенствованная звуковая сигнализация для холодильника, проигрыватель рингтонов и музыкальная игрушка.

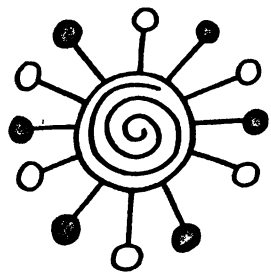
Глава 7 посвящена проектам с альтернативными источниками электроэнергии. Здесь рассмотрено генерирование напряжения по закону Фарадея и использование его для питания портативных приложений. Описаны три устройства, функционирующие без батареек: пульт дистанционного управления для телевизора, электронные игральные кости и игрушка, основанная на инерционности зрительного восприятия.

Приложение 1 представляет собой руководство, позволяющее читателю быстро освоить те команды языка C, которые необходимы во встроенных приложениях, и начать применять язык C для программирования микроконтроллеров tinyAVR.

В *приложении 2* приведена программа EAGLE для рисования схем и разводки печатных плат. Печатные платы всех устройств из этой книги были сделаны при помощи бесплатной версии программы EAGLE. Платы можно заказать у производителей или изготовить при помощи станка Modela (или аналогичного). Возможны и другие методы изготовления печатных плат.

Руководствуясь *приложением 3*, можно собрать замечательную лупу со светодиодной подсветкой, управляемой микроконтроллером.

Мы надеемся, что вы получите от изготовления этих устройств такое же удовольствие, как получили мы от того, что рассказали вам о них.



Глава 1

Обзор "малюток" Tiny

В соответствии с законом Мура степень интеграции микросхем по-прежнему увеличивается в два раза (ну, почти в два) каждые 18 месяцев. Это означает, что каждые полтора года изготовители интегральных полупроводниковых схем могут разместить на той же самой площади микросхемы в два раза больше транзисторов и прочих компонентов. Эта важная гипотеза была впервые высказана Гордоном Муром (одним из основателей компании Intel) в середине 1960 годов и, как это ни удивительно, по-прежнему остается верной (более или менее). Габариты персонального компьютера постоянно уменьшаются. Существуют различные модели компьютеров: настольные, переносные, карманные и т. д. Недавно появились так называемые компактные компьютеры (Small Form Factor PC). Это доступные потребителю небольшие универсальные компьютеры со стандартным программным обеспечением. Действие закона Мура распространяется не только на персональные компьютеры, но и на бытовые электронные устройства: мой нынешний мобильный телефон (который имеет гораздо больше функций, чем предыдущий) значительно компактнее своего предшественника!

Употребляя термин "компьютер", мы чаще всего имеем в виду обычное вычислительное устройство для работы с текстовым редактором, выхода в Интернет и т. д. Но в наши дни почти любое электронное устройство обладает определенными вычислительными способностями. Такие компьютеры называют "встроенными", поскольку они входят в состав более крупной системы и позволяют ей увеличить свои возможности.

В стремлении к малогабаритным изделиям наше внимание привлекли компьютеры еще более компактных размеров: Tiny ("малютка"). В отличие от прочих компьютеров, это миниатюрные специализированные компьютерные системы, которые могут поместиться в нагрудном кармане рубашки. Многие изготовители поставляют наборы для сборки таких компьютеров (лидеры здесь — компании Microchip и Atmel). По размерам микросхема соизмерима с рисовым зернышком и все, что ей нужно, — это подходящий источник питания и схема сопряжения. Запрограммируйте микросхему соответствующим образом, и у вас получится свое собственное персональное электронное устройство, которое может быть совершенно уникальным.

Что могут делать такие маленькие встроенные компьютеры? Есть ли от них хоть какая-то польза? Далее мы покажем, насколько маленькими они могут быть и что они могут делать.

Микроконтроллеры tinyAVR компании Atmel

Серия микроконтроллеров tinyAVR имеет много разновидностей. Число выводов может быть от 4 (у серии ATtiny4/5/9/10) и до 28 (у серии ATtiny48/88). Некоторые микросхемы серии ATtiny48/88 имеют только 24 контакта. Широко применяется схема ATtiny13, которая имеет 8 контактов: два для питания и шесть для ввода/вывода. Это не слишком много, но даже при помощи шести контактов доступны разнообразные возможности.

Из представленной далее в этой главе таблицы микросхем tinyAVR мы выбрали для большинства наших проектов следующие: ATtiny13, ATtiny25/45/85 и ATtiny261/461/861. Они представляют собой весь спектр семейства Tiny. Все эти микросхемы снабжены статической памятью (SRAM) для программирования на языке C. Схема Tiny13 имеет всего 1 Кбайт памяти для хранения программ, а схемы Tiny861 и Tiny85 — 8 Кбайт. Схемы Tiny13 и Tiny25/45/85 совместимы по цоколевке, но серия Tiny25/45/85 имеет больше памяти и функций. Если код не помещается в схеме Tiny13, то ее можно заменить схемой Tiny24/45/85 (в зависимости от требований к размеру памяти).

Почти все устройства, описанные в этой книге, отличаются привлекательным внешним видом благодаря большим светодиодным индикаторам. Новый метод управления большим числом светодиодов при помощи ограниченного числа управляющих выводов (Charlieplexing — "метод Чарли") позволяет мультиплексировать до 20 светодиодов, имея всего пять контактов ввода/вывода. Данный метод применялся для реализации привлекательных графических дисплеев и управления семисегментными индикаторами. В некоторых устройствах использованы графические жидкокристаллические дисплеи.

Любую описанную конструкцию можно собрать за один-два дня.

Микросхемы tinyAVR

Микросхемы tinyAVR отличаются друг от друга по нескольким признакам: по числу выводов, по размеру памяти, по типу корпуса (DIP — корпус с двумя рядами выводов по длинным сторонам; SOIC — то же для поверхностного монтажа; MLF — квадратный корпус для поверхностного монтажа), по периферийным функциям, по интерфейсам обмена и т. д. На рис. 1.1 показаны примеры микросхем tinyAVR в корпусах типа DIP, а на рис. 1.2 — в корпусах типа SOIC. Номенклатура микросхем постоянно меняется, поскольку компания Atmel регулярно добавляет новые устройства для замены старых. Последние изменения можно всегда посмотреть на сайте по ссылке: **www.avrgenius.com/tinyavr1**.

Большинство микросхем организовано так, что каждая схема из одной серии отличается от остальных всего несколькими функциями (размером памяти и т. п.).

Некоторые основные серии и схемы семейства tinyAVR приведены в табл. 1.1 и на рис. 1.1, 1.2.

Если в маркировке микросхемы имеется буква "A", значит, схема выполнена по технологии picoPower и снабжена функцией для снижения потребления электроэнергии.

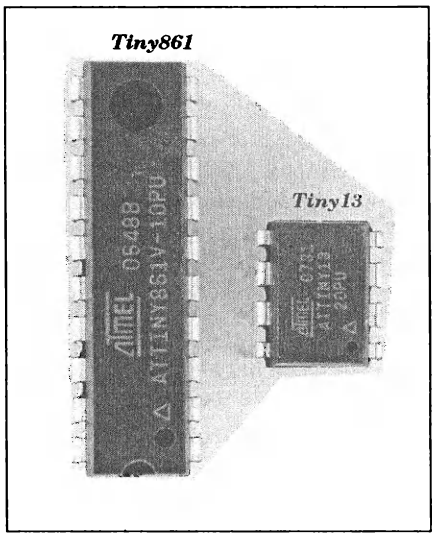


Рис. 1.1. Микроконтроллеры tinyAVR в корпусах DIP

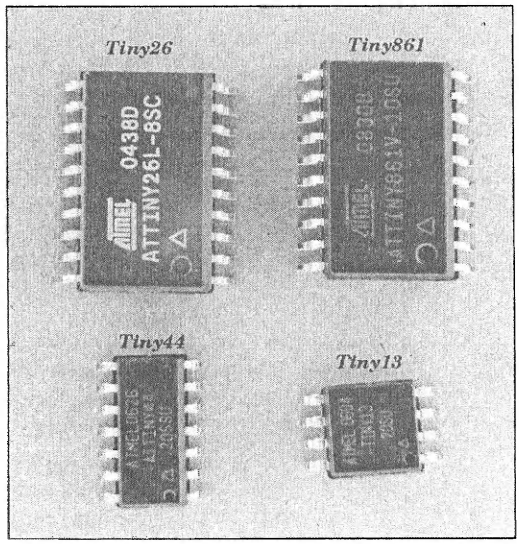


Рис. 1.2. Микроконтроллеры tinyAVR в корпусах SMD

Таблица 1.1. *Некоторые основные серии и микросхемы семейства tinyAVR*

Номер п/п	Серия/ Микросхема	Описание
1	ATtiny4/5/9/10	Максимум 4 контакта ввода/вывода, рабочее напряжение 1,8–5,5 В, 32 байта SRAM, производительность до 12 MIPS (на частоте 12 МГц), Flash-память для хранения программ (1 Кбайт в ATtiny9/10 и 512 байт в ATtiny4/5), аналого-цифровой преобразователь (в ATtiny9/10)
2	ATtiny13	Максимум 6 контактов ввода/вывода, рабочее напряжение 1,8–5,5 В, 64 байта SRAM, 64 байта EEPROM, производительность до 20 MIPS (на частоте 20 МГц), 1 Кбайт Flash-памяти для хранения программ, аналого-цифровой преобразователь (ADC)
3	ATtiny24/44/84	Максимум 12 контактов ввода/вывода, рабочее напряжение 1,8–5,5 В, 128/256/512 байт SRAM и 128/256/512 байт EEPROM (соответственно), производительность до 20 MIPS (на частоте 20 МГц), 2/4/8 Кбайт Flash-памяти для хранения программ (соответственно), аналого-цифровой преобразователь (ADC), температурный датчик (на кристалле), универсальный последовательный интерфейс (USI)

Таблица 1.1 (окончание)

Номер п/п	Серия/ Микросхема	Описание
4	ATtiny25/45/85	Максимум 6 контактов ввода/вывода, рабочее напряжение 1,8–5,5 В, 128/256/512 байт SRAM и 128/256/512 байт EEPROM (соответственно), производительность до 20 MIPS (на частоте 20 МГц), 2/4/8 Кбайт Flash-памяти для хранения программ (соответственно), аналого-цифровой преобразователь (ADC), универсальный последовательный интерфейс (USI)
5	ATtiny261/461/861	Максимум 16 контактов ввода/вывода, рабочее напряжение 1,8–5,5 В, 128/256/512 байт SRAM и 128/256/512 байт EEPROM (соответственно), производительность до 20 MIPS (на частоте 20 МГц), 2/4/8 Кбайт Flash-памяти для хранения программ (соответственно), аналого-цифровой преобразователь (ADC), универсальный последовательный интерфейс (USI)
6	ATtiny48/88	Максимум 24/28 контактов ввода/вывода (в зависимости от корпуса), рабочее напряжение 1,8–5,5 В, 256/512 байт SRAM (соответственно), 64 байта EEPROM, производительность до 12 MIPS (на частоте 12 МГц), 4/8 Кбайт Flash-памяти для хранения программ (соответственно), аналого-цифровой преобразователь (ADC), последовательный внешний интерфейс (SPI)
7	ATtiny43U	Максимум 16 контактов ввода/вывода, рабочее напряжение 0,7–1,8 В, 256 байт SRAM, 64 байта EEPROM, производительность до 1 MIPS на мегагерц, 4 Кбайт Flash-памяти для хранения программ, аналого-цифровой преобразователь (ADC), температурный датчик (на кристалле), универсальный последовательный интерфейс (USI). Микросхема с низким энергопотреблением, встроенный преобразователь автоматически генерирует стабильное напряжение питания 3 В от низковольтного источника питания (не ниже 0,7 В)

Архитектура tinyAVR

В этом разделе описывается внутреннее устройство микросхем семейства Tiny. Необходимо отметить, что здесь дан обзор только самых часто используемых функций серии Tiny. Некоторые функции в описании микросхем могут отсутствовать. Дополнительную информацию по этим функциям смотрите в спецификациях конкретных микросхем.

Память

В архитектуре AVR предусмотрено два основных адресных пространства: память данных и память программ. Кроме того, микросхемы имеют стираемую память типа EEPROM для хранения данных. Flash-память для хранения программ организована как линейный массив 16-разрядных ячеек (поскольку размер всех команд AVR равен 16 или 32 бита). Адресное пространство внутренней памяти SRAM, внутренних регистров и регистров ввода/вывода общее. Младшие 32 байта заняты внутренними регистрами, следующие 64 байта — регистрами ввода/вывода; затем адресация SRAM продолжается с адреса 0x60. Внутренняя память EEPROM предназначена для временного хранения данных. На рис. 1.3 показана карта памяти микроконтроллеров семейства Tiny.

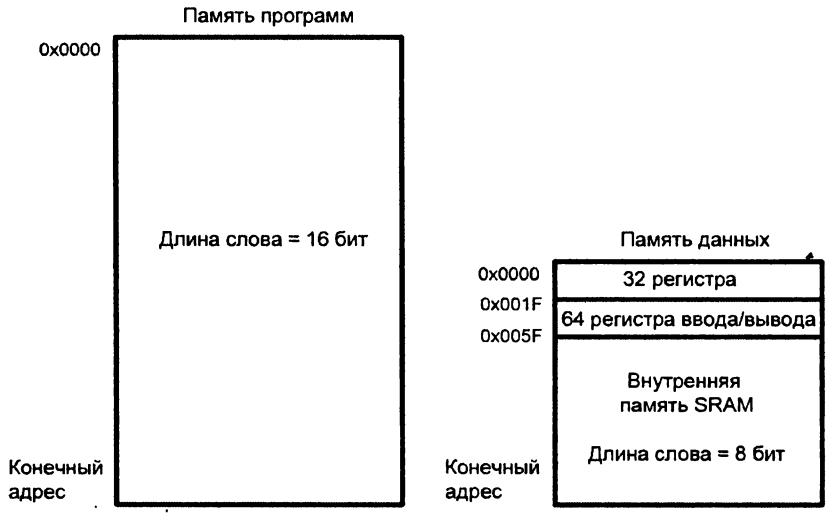


Рис. 1.3. Карта памяти микроконтроллеров семейства Tiny

Порты ввода/вывода

Порты ввода/вывода контроллеров AVR состоят из отдельных контактов, каждый из которых можно сконфигурировать для ввода или вывода. К любому входному контакту можно присоединить нагрузку. Это необходимо для подключения датчиков, которые не выдают электрического сигнала (например, микропереключателей). Каждый выходной буфер обеспечивает ток 40 мА, что позволяет напрямую подключать светодиоды. Все контакты ввода/вывода защищены диодами по шинам питания и земли. На рис. 1.4 показана блок-схема портов ввода/вывода контроллеров AVR.

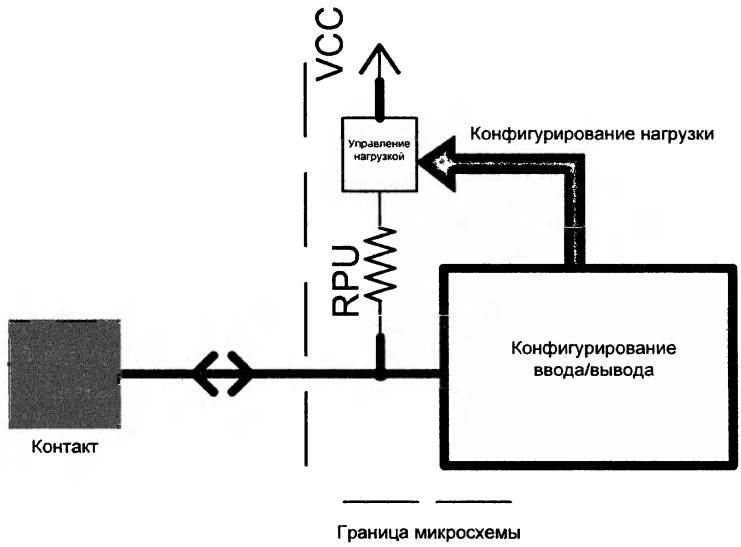


Рис. 1.4. Блок-схема порта ввода/вывода контроллеров семейства Tiny

Таймеры

В микросхемах tinyAVR обычно есть встроенные синхронные или асинхронные восьмиразрядные таймеры. Для синхронного тактирования служит сигнал от внутреннего тактового генератора (или от делителя частоты), для асинхронного — внешний тактовый сигнал либо цепь фазовой автоподстройки частоты (Phase Lock Loop, PLL), которая работает на частоте до 64 МГц.

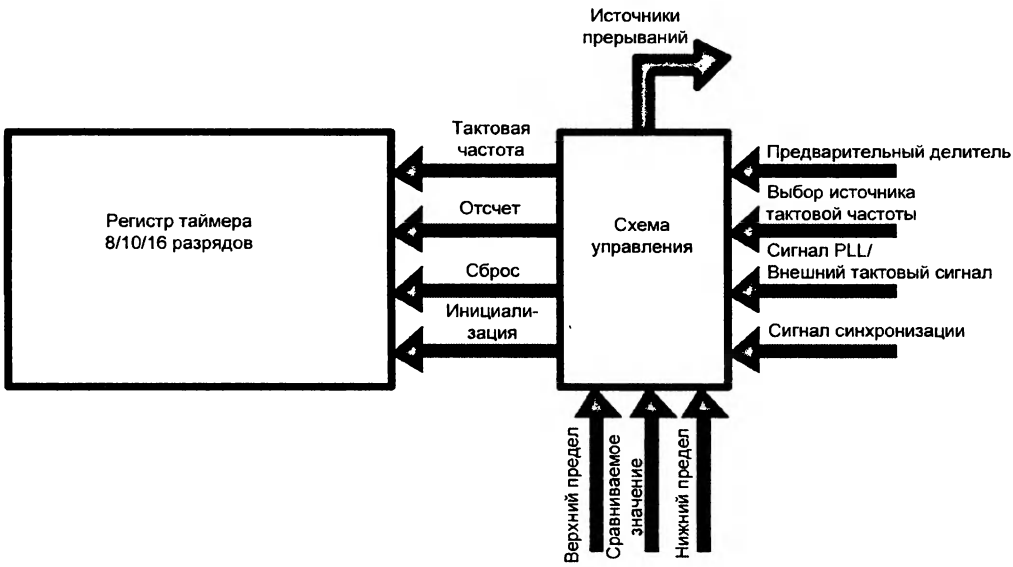


Рис. 1.5. Блок-схема таймера AVR

В состав некоторых контроллеров входят также 10- или 16-разрядные таймеры. Помимо счетчика, эти таймеры также имеют блоки сравнения, которые генерируют ШИМ-сигнал на контактах ввода/вывода. Таймеры могут работать в разных режимах (нормальный, захват, режим широтно-импульсной модуляции, сброс таймера по результату сравнения и т. д.). Каждый таймер имеет несколько связанных с ним источников прерываний, которые описываются в следующем разделе, посвященном прерываниям. На рис. 1.5 показана блок-схема таймера AVR.

Прерывания

В контроллерах AVR предусмотрено несколько различных источников прерываний, которым выделены соответствующие векторы в адресном пространстве программ. По умолчанию векторы прерываний занимают первые адреса в адресном пространстве программ. Самый младший адрес (0x0000) назначен вектору сброса, который вообще говоря, не является источником прерывания. Адрес прерывания определяет также и его приоритет. Чем ниже адрес, тем выше уровень приоритета прерывания. Поэтому сброс имеет самый высокий приоритет. Если одновременно происходит несколько прерываний, то первым выполняется прерывание с самым высоким приоритетом, за ним прерывание с более низким приоритетом и т. д. Прерывание приостанавливает нормальное выполнение основной программы и переставляет счетчик команд на подпрограмму обработки прерывания (Interrupt Service Routine, ISR). После обработки прерывания счетчик команд устанавливается снова на основную программу. На рис. 1.6 показано выполнение кода ISR.

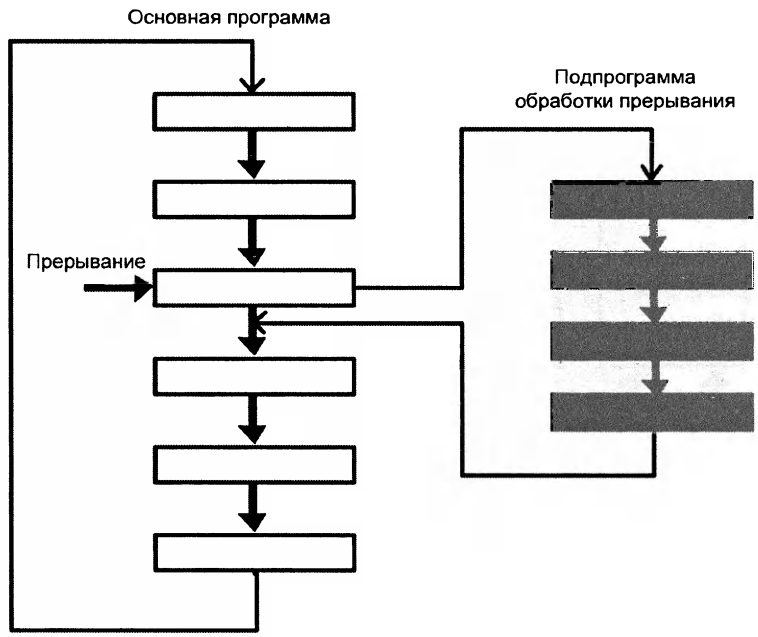


Рис. 1.6. Обработка прерывания

Каждому прерыванию присваивается свой бит разрешения, который для активизации прерывания должен быть установлен в логическую единицу (так же как и глобальный бит разрешения прерываний в регистре состояния). Глобальный бит разрешения прерываний при выполнении ISR по умолчанию сбрасывается, поэтому никакие другие прерывания произойти не могут (если только программа пользователя не выставила явным образом глобальный бит разрешения прерываний, чтобы разрешить вложенные прерывания (прерывания внутри другого прерывания)). Периферийные устройства AVR (таймеры, интерфейс USI, АЦП, аналоговые компараторы и т. д.) имеют разные источники прерываний для различных состояний или режимов.

USI: универсальный последовательный интерфейс

Интерфейс USI обеспечивает основные аппаратные ресурсы для последовательного обмена. Этот интерфейс можно сконфигурировать для работы либо по трехпроводному протоколу (который совместим с последовательным периферийным интерфейсом SPI), либо по двухпроводному протоколу (который совместим с двухпроводным интерфейсом TWI). При минимальном управляющем программном обеспечении интерфейс USI допускает значительно более высокие скорости передачи и требует меньше памяти для программ (чем чисто программные решения). Прерывания применяются для уменьшения нагрузки на процессор.

Аналоговый компаратор

Контроллеры AVR имеют компаратор, который измеряет аналоговое входное напряжение на двух входах и выдает цифровой выходной сигнал (0 или 1) в зависимости от того, на каком входе (положительном или отрицательном) есть напряжение.

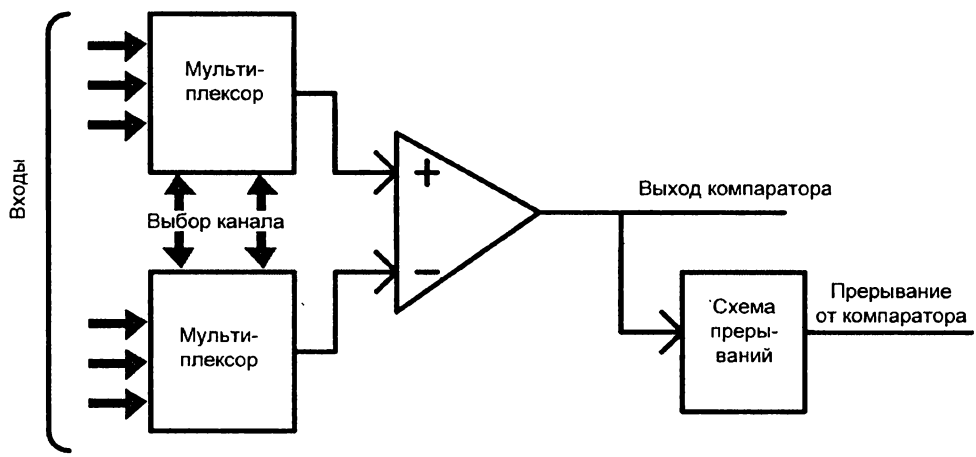


Рис. 1.7. Блок-схема аналогового компаратора

Положительный и отрицательный входы могут быть выбраны из контактов ввода/вывода. Изменение выхода компаратора можно испо источник прерывания. Выход компаратора можно увидеть на контакте логового компаратора (АСО). На рис. 1.7 показана блок-схема аналог ратора.

Аналого-цифровой преобразователь

Аналого-цифровой преобразователь (АЦП, ADC) представляет собой 10-разрядный преобразователь последовательного приближения с несколькими несимметричными входами. В некоторых микросхемах есть также дифференциальные входы (для преобразования разности аналоговых напряжений в двух точках в цифровой код). Для повышения точности измерений иногда усиливают входное напряжение (до преобразования). Опорное напряжение для измерения можно брать с контактов AREF, VCC и от внутреннего источника опорного напряжения. На рис. 1.8 показана блок-схема аналого-цифрового преобразователя.

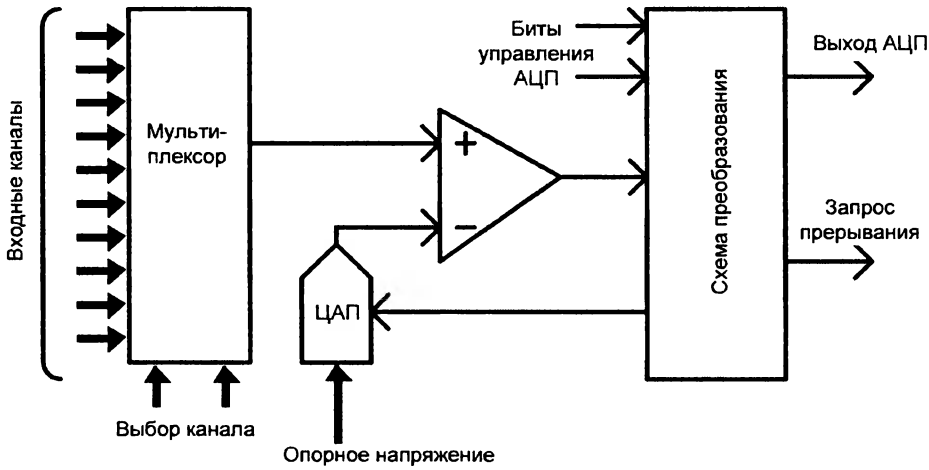


Рис. 1.8. Блок-схема аналого-цифрового преобразователя

Источники тактовых сигналов

Источники сигналов тактовой частоты: калиброванный RC-генератор, внешний тактовый генератор, кварцевый генератор, сторожевой генератор, низкочастотный кварцевый генератор, а также генератор с фазовой автоподстройкой частоты (PLL). Источник тактового сигнала можно задать (из этих вариантов) при помощи fuse-битов (конфигурационных ячеек). Частота сигнала от выбранного источника может быть впоследствии подвергнута предварительному делению (при помощи выставления битов в регистре предварительного деления) во время инициализации про-

граммного обеспечения пользователя. Тактовый сигнал поступает в разные модули микросхемы (CPU, I/O, Flash и ADC):

- CLK_CPU — синхронизирует те части системы, которые обеспечивают работу ядра AVR (внутренние регистры, регистр состояния и т. д.).
- CLK_I/O — используется большинством модулей ввода/вывода (таймеры/счетчики, интерфейс USI, синхронные внешние прерывания и т. д.).
- CLK_FLASH — управляет работой интерфейса Flash-памяти.
- CLK_ADC — в отличие от других модулей ввода/вывода, АЦП получает отдельный тактовый сигнал, чтобы во время работы АЦП можно было прервать другие тактовые сигналы (для снижения помех остальных цифровых цепей). Это позволяет получить более точные результаты аналого-цифрового преобразования. На рис. 1.9 показаны разные варианты подачи сигнала тактовой частоты.

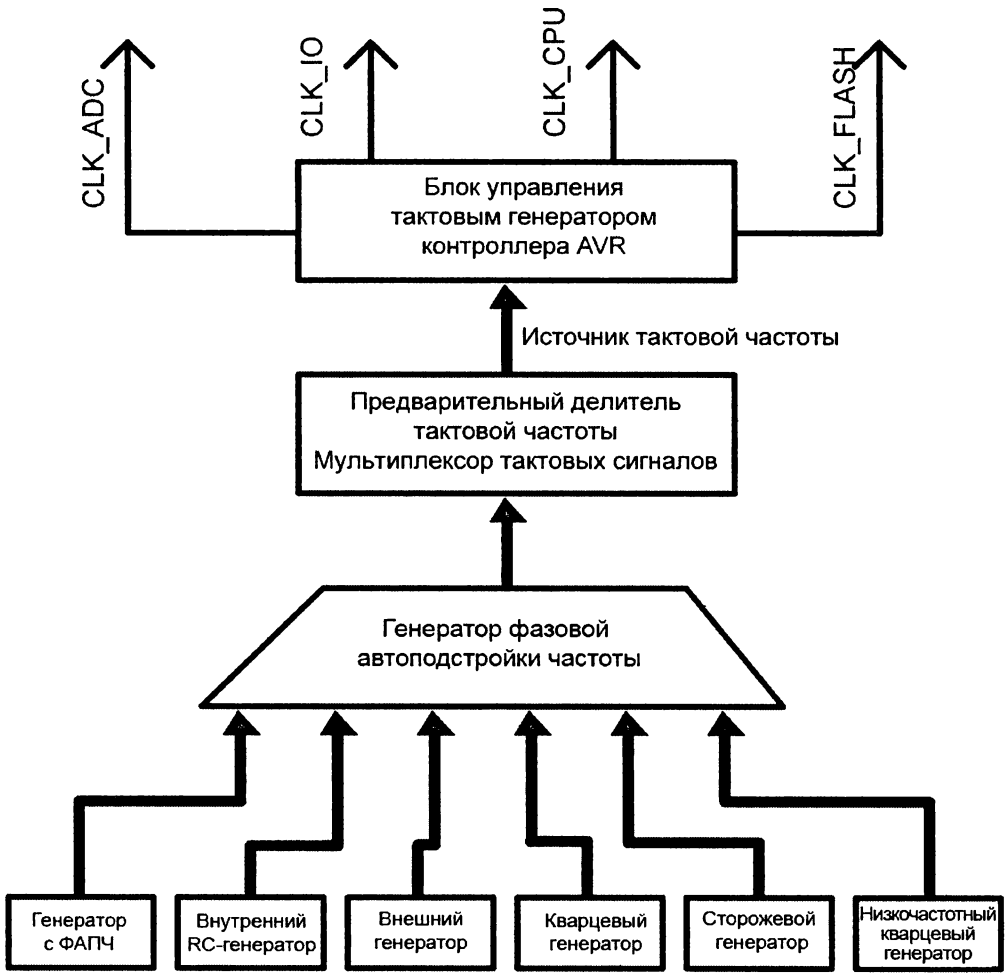


Рис. 1.9. Источники сигнала тактовой частоты

Управление электропитанием и режимы ожидания

В современных контроллерах, в том числе и в микросхемах AVR, предусмотрено самое эффективное управление электропитанием. Они поддерживают режимы ожидания, которые могут быть сконфигурированы пользовательским программным обеспечением, и позволяют отключать неиспользуемые модули (снижая тем самым энергопотребление).

Возможны следующие режимы ожидания: выключение, энергосбережение, простой, уменьшение шумов АЦП и др. Разные микросхемы поддерживают различные режимы, подробности относительно которых можно всегда найти в спецификациях.

Более того, каждый режим имеет свой набор источников пробуждения (для выхода из данного режима и перехода в полноценное рабочее состояние).

Сброс системы

Источники сигнала сброса контроллеров AVR:

- Сброс по включению питания — микроконтроллер сбрасывается, когда напряжение питания уменьшается до заданного порога срабатывания.
- Внешний сброс — когда на контакте RESET присутствует низкий уровень сигнала.
- Сброс по сторожевому таймеру — когда активизирован сторожевой таймер и его период ожидания истек.
- Сброс по падению напряжения питания — когда активизирован детектор падения напряжения и напряжение питания VCC оказывается ниже заданного порога срабатывания.

После сброса его источник может быть определен программно посредством проверки отдельных битов регистра состояния микроконтроллера. Во время сброса все регистры ввода/вывода устанавливаются в свои начальные значения, и программа начинает выполнение с вектора сброса. На рис. 1.10 показана блок-схема различных источников сигнала сброса.

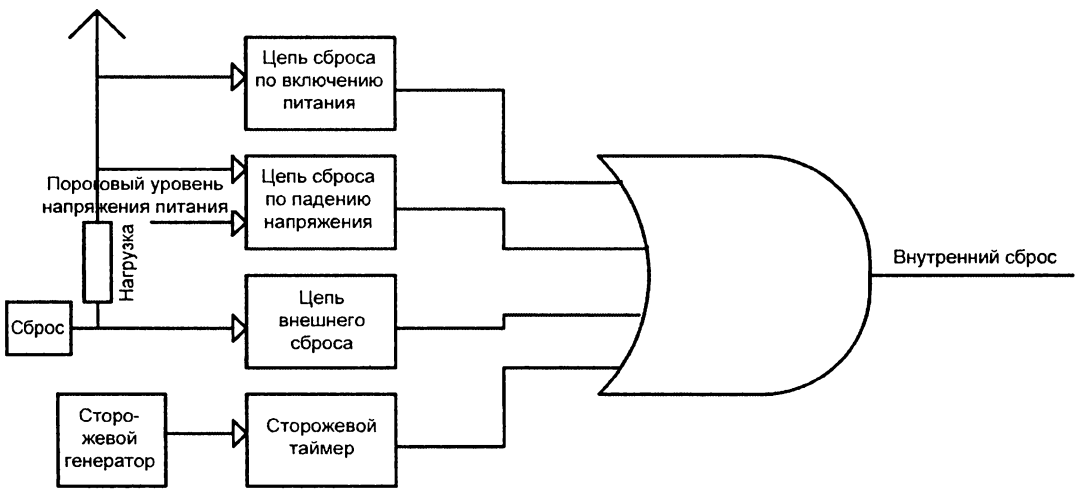


Рис. 1.10. Источники сигнала сброса

Программирование микроконтроллеров

Программирование контроллеров AVR включает в себя установку битов блокирования, fuse-битов, программирование Flash-памяти, а также программирование внутренней памяти EEPROM. Эти данные могут быть считаны с контроллера вместе с байтами идентификации устройства. Микросхемы семейства Tiny можно запрограммировать при помощи последовательного или параллельного способа. В этой книге (если не оговорено другое) мы применяли последовательное программирование микроконтроллеров семейства Tiny. Здесь тоже есть два варианта: системное программирование (ISP) и последовательное программирование при высоком напряжении (HVSP). HVSP применим (как альтернатива параллельному программированию) только для восьмиконтактных микроконтроллеров (поскольку эти микросхемы имеют слишком мало контактов для параллельного программирования).

Системное программирование использует внутренний последовательный периферийный интерфейс (SPI) контроллеров AVR для загрузки кода в память Flash и EEPROM. При этом также программируются биты блокирования и fuse-биты. Для такого программирования требуются только контакты VCC, GND, RESET и три сигнальных линии. В некоторых случаях для ввода/вывода (или других целей) может потребоваться контакт RESET. Если этот контакт сконфигурирован как контакт для ввода/вывода (при помощи бита RSTDISBL), то программирование в режиме ISP невозможно и микросхему следует программировать при помощи параллельного программирования или последовательного программирования при высоком напряжении.

Для программирования контроллеров AVR есть еще один метод — система отладки debugWIRE (описана в следующем разделе). Последняя серия шестиконтактных микросхем компании Atmel (ATtiny4/5/9/10) не поддерживает описанных ранее вариантов программирования и имеет новый встроенный интерфейс программирования TPI.

Биты блокирования служат для защиты программного обеспечения пользователя (во избежание дублирования), а fuse-биты применяются для начальной настройки контроллера, которая не может (и не должна) выполняться программным обеспечением пользователя. На рис. 1.11 показаны сигналы для последовательного программирования ISP.

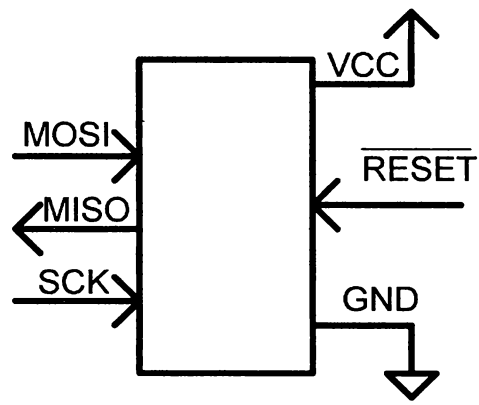


Рис. 1.11. Сигналы для последовательного программирования

Отладочная система debugWIRE

Отладочная система debugWIRE — это однопроводной интерфейс для аппаратной отладки и программирования памяти Flash и EEPROM. Этот интерфейс включается посредством программирования fuse-бита DWEN. После включения этого интерфейса обмен данными между микросхемой и эмулятором происходит через контакт RESET. Таким образом, при программировании через этот интерфейс внешний сброс не работает. Протокол программирования в данном случае аналогичен JTAG ICE mkII (популярный инструмент отладки компании Atmel). На рис. 1.12 показан отладочный интерфейс debugWIRE.

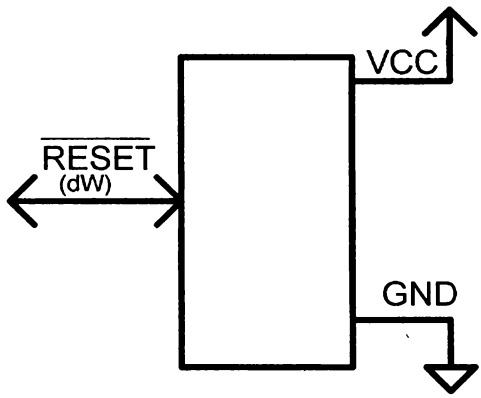


Рис. 1.12. Отладочный интерфейс debugWIRE

Составляющие проекта

В этой книге рассмотрены проекты, охватывающие широкий спектр идей и включающие в себя несколько прикладных областей. Описанные конструкции пригодны как для развлечения, так и для обучения. Однако важно рассмотреть и сам процесс проектирования и разработки.

Как же создается система или проект, который до сих пор никому не приходил в голову? Конечно, вы должны обдумать, что вам нужно. Иногда толчком может послужить другая разработка. Это абстрактный процесс, который можно проиллюстрировать примером. Предположим, что вы увидели использование светодиодов в какой-то системе: яркие и мигающие, они привлекли ваше внимание, и вы подумали, а что если я размещу эти веселенькие светодиоды на своей шапке и заставлю их мигать или менять интенсивность свечения? Самое главное — найти что-то оригинальное. На рис. 1.13 схематично изображен процесс проектирования и разработки.

После того как идея зародилась в вашей голове, вы можете начать развивать ее. Мы рекомендуем сразу же поискать по Интернету, чтобы убедиться в том, что она не пришла в голову кому-то еще. Не стоит повторно изобретать колесо. Если ваша идея уже реализована, то стоит подумать, как ее усовершенствовать. Если вы берете

готовую реализацию и улучшаете ее, то вам следует поделиться своей работой с автором исходного проекта, чтобы получить признание своей работы и зафиксировать сделанный вами вклад. Таким путем можно улучшить уже существующую конструкцию. Сказанное применимо к проектам, которые доступны в Интернете на условиях какой-нибудь бесплатной лицензии. В других случаях вам может понадобиться уточнить юридические нюансы. В большинстве ситуаций нарушения закона не будет в том случае, когда вы используете оригинальную разработку (или ее адаптацию) исключительно в личных целях. Однако при коммерческом применении нужно будет обязательно связаться с автором разработки (во избежание возникновения проблем в будущем).

В каждом проекте есть две отдельных составляющих (рис. 1.13): аппаратные компоненты и программное обеспечение. Аппаратную часть можно реализовать разными способами, но проще всего на основе микроконтроллеров. Поскольку наша книга посвящена применению микроконтроллеров, то именно на этом мы и сосредоточимся. Помимо микроконтроллера, для работы любого устройства нужен источник питания. Понадобятся и другие (специфичные для конкретного проекта) аппаратные компоненты (несмотря на то, что в современных микроконтроллерах интегрировано большое количество функций). Например, несмотря на то, что микроконтроллер имеет выходные контакты для управления семисегментными индикаторами, он не способен обеспечить большой ток, который может понадобиться, поэтому вам потребуются внешние формирователи тока.

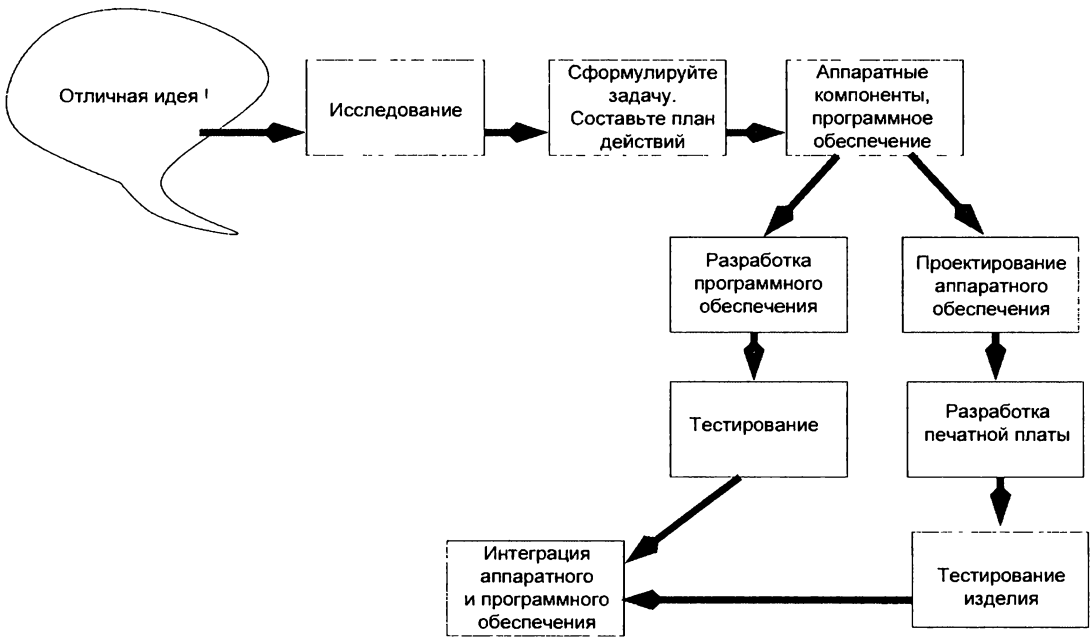


Рис. 1.13. Процесс проектирования и разработки

Еще пример: если вы хотите установить внешний датчик, который выдает аналоговое напряжение для измерения физического параметра, то диапазон напряжения этого датчика может оказаться неподходящим для аналого-цифрового преобразователя микроконтроллера и придется добавить внешний усилитель. На рис. 1.14 показаны элементы современного микроконтроллера.

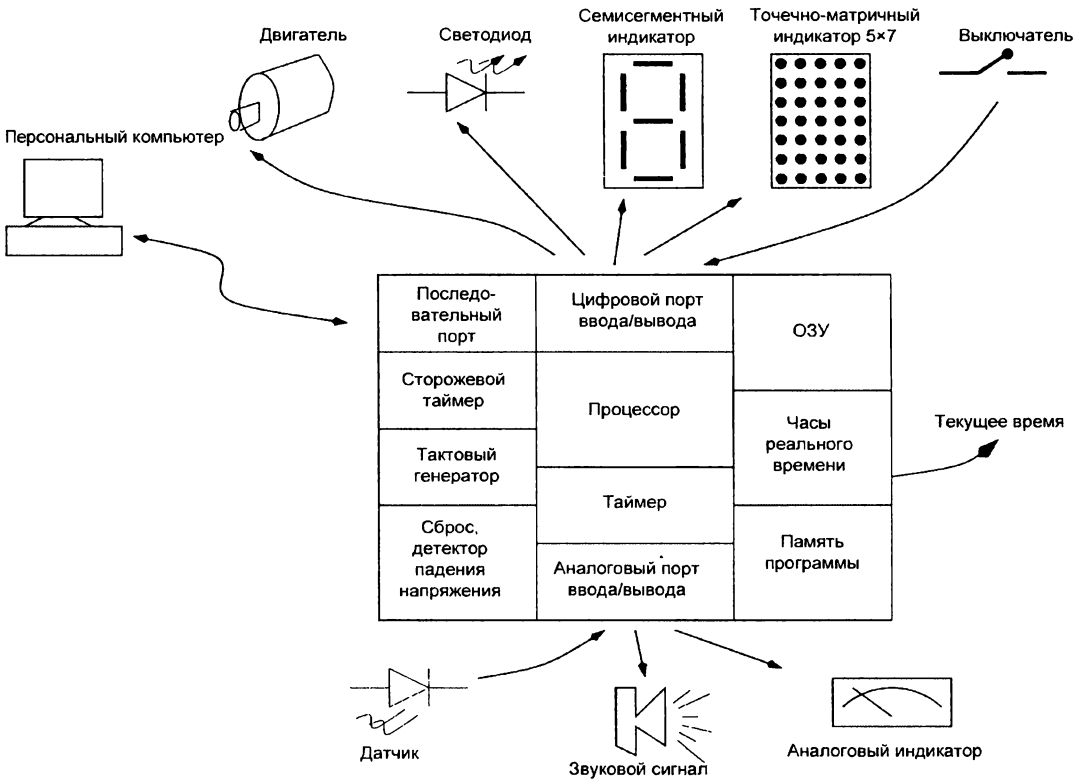


Рис. 1.14. Элементы современного проекта с использованием микроконтроллера

Программные компоненты — это прикладная программа, которая выполняется в микроконтроллере. Однако этим термином может также обозначаться и программа для обмена с микроконтроллером, работающая на персональном компьютере.

Разработка проекта требует параллельной одновременной работы над обоими составляющими: аппаратным и программным обеспечением. Программу для микроконтроллера можно создавать на персональном компьютере, причем большую часть кода можно разработать даже в отсутствие готового аппаратного прототипа. Программный код можно протестировать на персональном компьютере на отсутствие логических ошибок. Некоторые части кода (требующие внешних сигналов или синхронизации с аппаратными событиями) протестировать не удастся, поэтому такую проверку придется отложить до стадии интеграции программного и аппаратного обеспечения. После появления аппаратного прототипа его нужно совместить

с программным обеспечением и эту интегрированную реализацию проекта следует протестировать на соответствие требованиям. Процесс может оказаться не совсем гладким и потребовать нескольких итераций цикла разработки.

Помимо специфичных аппаратных компонентов и программного обеспечения для большинства проектов потребуются еще кое-какие стандартные блоки — источники питания и сигнала тактовой частоты (рис. 1.15). Источник питания и стабилизация питающего напряжения подробно описаны в одном из последующих разделов.

Для работы устройства очень важен источник сигнала тактовой частоты. К счастью, такой источник обычно есть в самом микроконтроллере. Обычно это RC-генератор, который не очень точен и частота которого зависит от рабочего напряжения, но для многих устройств этого вполне достаточно. Внешний источник тактовой частоты понадобится только для тех приложений, которые критичны к измерению времени. Все микроконтроллеры семейства AVR снабжены встроенным источником сигнала тактовой частоты и в большинстве проектов этой книги мы используем именно его. Скорость выполнения программы напрямую зависит от тактовой частоты. Однако высокая тактовая частота имеет и недостаток: система потребляет больше электроэнергии. Между тактовой частотой и потреблением энергии существует линейная зависимость. Если вы удвоите тактовую частоту, то потребление энергии также возрастет в два раза. Поэтому неразумно выбирать самую высокую рабочую частоту, лучше определить ее исходя из требуемой скорости выполнения программы. Как мы покажем в проекте 1 (далее в этой же главе), выбрав самую низкую тактовую частоту, мы можем снизить потребляемую мощность до минимума. Основные компоненты устройства показаны на рис. 1.15.

Помимо источников тактовой частоты и питания, а также стабилизатора напряжения потребуются устройства ввода/вывода и подходящий корпус.

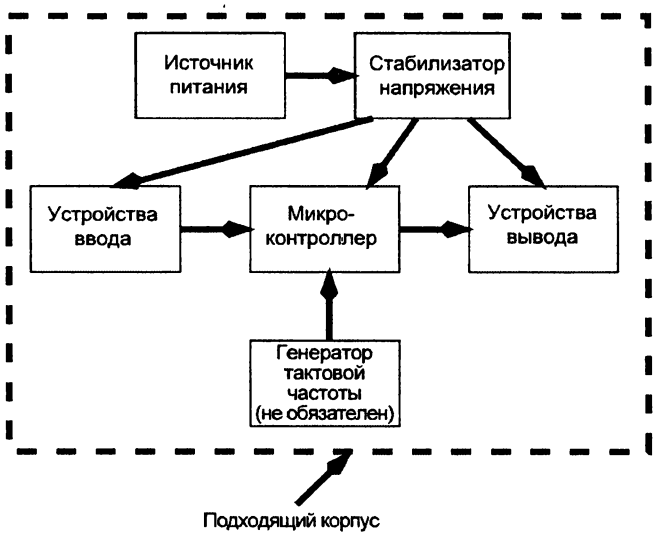


Рис. 1.15. Основные компоненты устройства

Источники питания

Для работы любой системы необходим источник энергии. Без него любая система — всего лишь "кусок железа". Очень важно правильно выбрать источник питания. Подключение малогабаритного переносного устройства к электрической сети привязывает его к розетке, и такое решение вряд ли можно считать приемлемым.

Батареи

Батареи — самый распространенный источник энергии для портативных электронных систем. Батареи различаются по типам, корпусам и энергетическим показателям. Энергетический показатель батареи характеризует количество хранящейся в ней энергии. Батареи бывают двух типов: одноразовые (гальванические элементы) и перезаряжаемые (аккумуляторы). Гальванические элементы дают энергию сразу же после их сборки и продолжают давать ее до тех пор, пока не разрядятся. Перезарядить их нельзя, поэтому после эксплуатации их приходится выбрасывать.

Аккумуляторы необходимо зарядить перед использованием. В течение срока эксплуатации их можно перезаряжать много раз, и поэтому они являются предпочтительными (по сравнению с обычными батарейками), хотя и более дорогими. Кроме того, удельная энергия аккумуляторной батареи хуже, чем у обычной. Удельная энергия — это количество энергии, хранящейся в батарее, на единицу ее массы. Обычная батарея может выдавать рабочее напряжение дольше, чем аккумуляторная такой же массы.

Широко распространен угольно-цинковый гальванический элемент. Его корпус сделан из цинка (который служит отрицательным полюсом). Корпус заполнен пастой из хлорида цинка и хлорида аммония (которая служит электролитом). Положительный полюс батареи — угольный или графитовый стержень, окруженный смесью диоксида марганца и угольным порошком. По мере использования цинковая оболочка становится тоньше (вследствие химической реакции, ведущей к окислению цинка) и в конечном итоге электролит начинает вытекать из корпуса. Угольно-цинковые батареи — самые дешевые. Выпускаются также щелочные батареи (рис. 1.16), которые похожи на угольно-цинковые, но в качестве электролита здесь применяется гидроксид калия. Номинальное напряжение угольно-цинковых и щелочных гальванических элементов равно 1,5 В.

Часто встречаются гальванические элементы на основе оксида серебра и лития. Батарейки на оксиде серебра (их напряжение равно 1,8 В) имеют гораздо более высокую удельную энергию, чем угольно-цинковые. В литиевых батареях используются различные химические соединения, в зависимости от комбинации которых напряжение может составлять от 1,5 до 3,7 В. На рис. 1.17 показаны литиевые и щелочные батарейки в форме таблеток.

Главная проблема с одноразовыми батареями состоит в том, что после разряда их необходимо сразу перерабатывать. В этом плане аккумуляторы гораздо привлекательнее: их можно несколько раз перезарядить (до того, как вам придется отправить их на переработку). Аккумуляторные батареи изготавливают как стандартных

размеров, так и на заказ. Самые распространенные — свинцово-кислотные, никель-кадмиевые, никель-металлогидридные, а также литий-ионные батареи. На рис. 1.18 показана литий-ионная батарея. Для зарядки аккумуляторных батарей требуются специальные зарядные устройства. Так, например, заряжать литий-ионную батарею зарядным устройством для никель-металлогидридных батарей нельзя, поскольку при этом батарея будет повреждена, что может привести даже к ее взрыву и пожару.

Рис. 1.16. Щелочные батареи типоразмеров 9 В и AAA

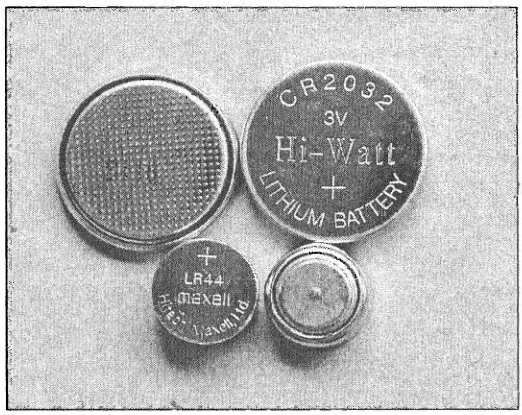


Рис. 1.17. Маленькая LR44 — щелочная батарея, большая CR2032 — литиевая батарея

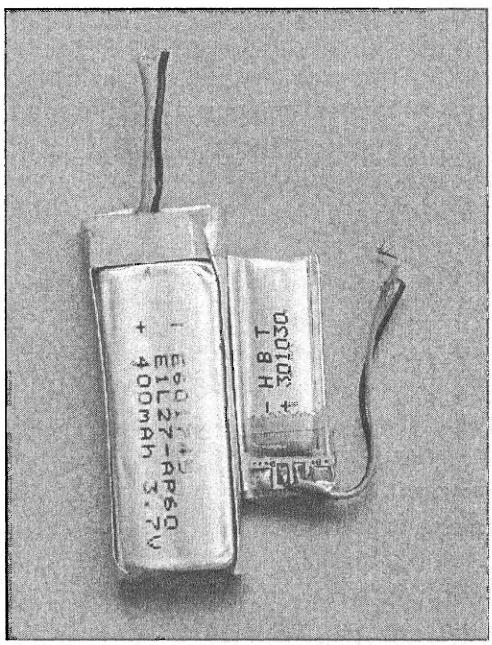


Рис. 1.18. Литий-ионная батарея

Батареи гальванических элементов и аккумуляторов выпускаются нескольких стандартных размеров. Некоторые из самых распространенных перечислены в табл. 1.2.

Таблица 1.2. *Обозначения батарей и их размеры*

Обозначение	Форма	Длина, мм	Диаметр/Длина, мм	Высота, мм
AAA	Цилиндр	44,5	12	–
AA	Цилиндр	50,5	14,5	–
9V	Прямоугольный параллелепипед	48,5	17,5	26,5
C	Цилиндр	50	26,2	–
D	Цилиндр	61,5	34,2	–

При выборе батареи необходимо учитывать следующие характеристики:

- Энергетическая емкость. Выражается в ампер-часах (или в миллиампер-часах). Это важная характеристика, которая показывает, как долго "продержится" батарея до полного разряда. Помните, что чем больше емкость батареи, тем больше ее размеры.
- Напряжение, выдаваемое батареей.
- Условия хранения (когда батарея не используется).
- Срок хранения. Показывает, сколько времени пройдет до полного саморазряда батареи. Не стоит покупать батареи про запас на десять лет вперед, если срок хранения составляет, к примеру, всего один год.
- Рабочая температура. Все химические источники тока имеют плохие температурные характеристики, т. к. скорость протекания химических реакций сильно зависит от температуры. При слишком низких (и высоких) температурах батареи работают очень плохо.
- Рабочий цикл. Некоторые батареи работают лучше тогда, когда они используются с перерывами. Рабочий цикл батареи показывает, можно ли включать батарею в непрерывном режиме без потери ее характеристик.

Батарея из фруктов

Для получения электричества можно использовать некоторые фрукты и овощи. Содержащиеся в них электролиты пригодны для изготовления простейших гальванических элементов. Для изготовления "фруктового" источника питания потребуется лимон, а также два электрода: медный и цинковый. Напряжение такого источника питания равно примерно 0,9 В. Величина тока зависит от площади электродов, контактирующих с электролитом, а также от качества самого электролита.

Для нашей "фруктовой" батареи потребуется несколько лимонов (это будет электролит) и несколько кусков меди и цинка (это будут электроды). В качестве

меди мы возьмем кусок фольгированного текстолита, а в качестве цинка — полоски, вырезанные из корпуса обычной батареи.

Последовательность изготовления батареи.

1. Начнем с куска текстолита. Его размер должен быть достаточным, чтобы на нем можно было создать три или четыре "острова", на каждом из которых поместится половинка разрезанного лимона.
2. Затем вскроем несколько батареек размера AA, сделаем из них цинковые полоски и зачистим их наждачной бумагой. Припаяем к каждой полоске по проводу. Вместо цинковых полосок подойдут даже гвозди. Гвозди обычно покрыты цинком, поэтому пригодны для изготовления батареи.
3. На медной печатной плате нужно напильником (или пилой) вырезать "островки" и припаять к каждому из них второй конец провода (от цинковой полоски). На одну ячейку понадобится половина лимона, один медный островок и одна цинковая полоска.
4. Положите половинки лимонов на медные островки. Сделайте на лимонах разрезы и вставьте в них цинковые полоски. На рис. 1.19 показана лимонная батарея из четырех элементов.

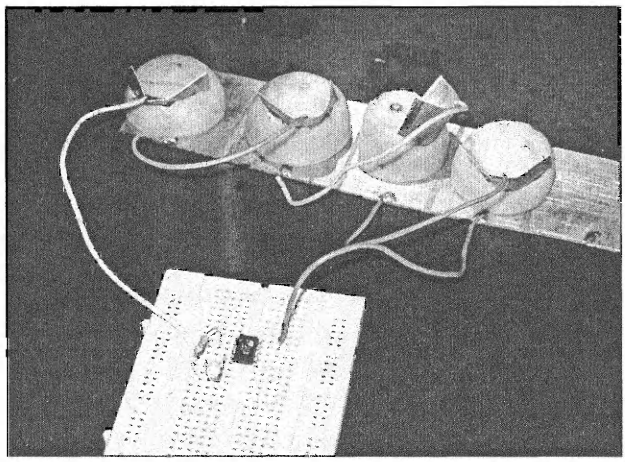


Рис. 1.19. "Лимонная" батарея

Адаптер переменного тока

Если вы используете адаптер переменного тока, то потребуется выпрямитель и емкостной фильтр (рис. 1.20). Выпрямитель можно собрать на отдельных диодах (например, 1N4001) либо взять готовый выпрямительный блок. Если источник питания выдает ток 500 мА, то диоды должны быть рассчитаны, по меньшей мере, на 1 А. Нужно учесть еще одну характеристику диода: максимальное обратное напряжение (Peak Inverse Voltage, PIV), которое диод может выдержать до пробоя. Так, например, максимальное обратное напряжение диода 1N4001 равно 50 В, а 1N4007 — 1000 В.

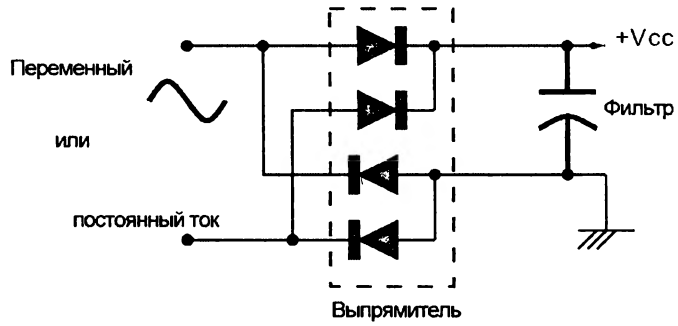


Рис. 1.20. Схема из выпрямителя и емкостного фильтра может использоваться как с переменным током, так и с постоянным

Амплитуда выпрямленного напряжения на конденсаторе фильтра в 1,4 раза выше среднеквадратического значения входного напряжения переменного тока. Значит, переменное напряжение в 10 В создаст на конденсаторе фильтра постоянное напряжение, равное примерно 14 В. Важно также правильно выбрать емкость и рабочее напряжение конденсатора фильтра. Чем больше емкость, тем меньше пульсации напряжения на выходе. При напряжении 14 В следует взять конденсатор с рабочим напряжением не менее 25 В. Схему, изображенную на рис. 1.20, можно использовать и с источником постоянного тока. При этом полярность подключенного источника тока не будет иметь значения.

Чтобы получить законченный блок питания, в схему необходимо добавить стабилизатор напряжения. В продаже имеются готовые микросхемы стабилизаторов. Мы рассмотрим интегральные линейные стабилизаторы напряжения с малым током покоя.

Широко распространены трехвыводные стабилизаторы типа 78XX. Они выпускаются многими компаниями и изготавливаются в разных корпусах. Для питания процессора AVR следует выбрать стабилизатор 7805 с выходным напряжением 5 В. Он может выдавать выходной ток до 1 А и питаться постоянным напряжением от 9 до 20 В. Подойдет также микросхема LM317, напряжение на выходе которой (1,25 В и выше) можно настроить при помощи двух резисторов.

Стабилизатор напряжения — это активный компонент, и при работе он потребляет некоторый ток. Этот ток называется током покоя и имеет порядок десятков миллиампер. Существуют специальные стабилизаторы напряжения с очень малым током покоя. Микросхемы LP2950 и LP2951 — это линейные микромощные стабилизаторы с низким током покоя (около 75 мА) компании National Semiconductor, имеющие очень малое падение напряжения (примерно 40 мВ при высокоомной нагрузке и 380 мВ при максимальном токе в 100 мА). Они идеально приспособлены для систем с питанием от батарей. Более того, ток покоя микросхем LP2950/LP2951 при повышении напряжения на них увеличивается очень незначительно. Это самые популярные трехконтактные стабилизаторы с низким током покоя, именно поэтому мы используем их во многих наших проектах.

Питание от разъема USB

USB — это популярный интерфейс, который есть и в персональных компьютерах, и в ноутбуках. В основном он служит для обмена данными между компьютером и периферийными устройствами (такими, как видеокамера, клавиатура и т. д.). USB — это четырехпроводной интерфейс с двумя контактами для питания и двумя для обмена данными. Питание на USB подается с компьютера. Номинальное напряжение равно +5 В, но по спецификации USB 2.0 оно может составлять от +4,4 до +5,25 В. Внешние устройства, подключаемые к компьютеру, например, мышь, могут получать питание от USB. Этим напряжением можно питать и другие внешние схемы, которые не связаны компьютером. Мы применяем питание от USB для устройств, расположенных недалеко от компьютера. С разъема USB можно взять ток до 100 мА (можно получить и больше, но для этого нужно дать запрос с устройства). В табл. 1.3 приведены питающие и сигнальные контакты порта USB.

Таблица 1.3. *Контакты разъема USB (для форм-факторов Mini и Micro)*

Контакт	Наименование	Цвет провода	Назначение
1	Vcc	Красный	+5 В
2	D	Белый	Сигнал данных –ve
3	D+	Зеленый	Сигнал данных +ve
4	ID	Бесцветный	Идентификация устройства
5	Gnd	Черный	Заземление

Солнечная энергия

Солнечную энергию можно преобразовать для питания электронных схем при помощи фотоэлементов. Солнечные батареи выдают мощность от долей ватта до нескольких сотен ватт. Выходная мощность солнечного элемента прямо пропорциональна интенсивности падающего света и обратно пропорциональна температуре элемента. Для максимально эффективной работы солнечный элемент должен быть расположен перпендикулярно падающему свету. Обычно выходную мощность солнечной батареи регулируют с помощью специальных схем. Чаще всего от солнечных элементов заряжают аккумуляторы, чтобы от них можно было постоянно получать электроэнергию. Подробности по использованию солнечных элементов описываются в одной из последующих глав.

Генератор на эффекте Фарадея

Рабочее напряжение для многих небольших встроенных систем можно получить при помощи интересного устройства, которое преобразует механическую энергию в электрическую. Подобная конструкция, действие которой основано на эффекте Фарадея, показана на рис. 1.21.

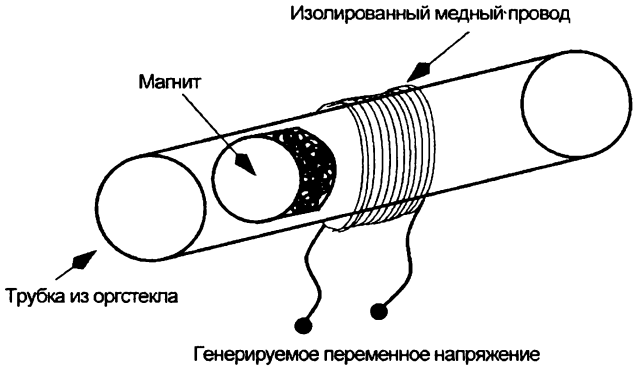


Рис. 1.21. Генератор напряжения на эффекте Фарадея

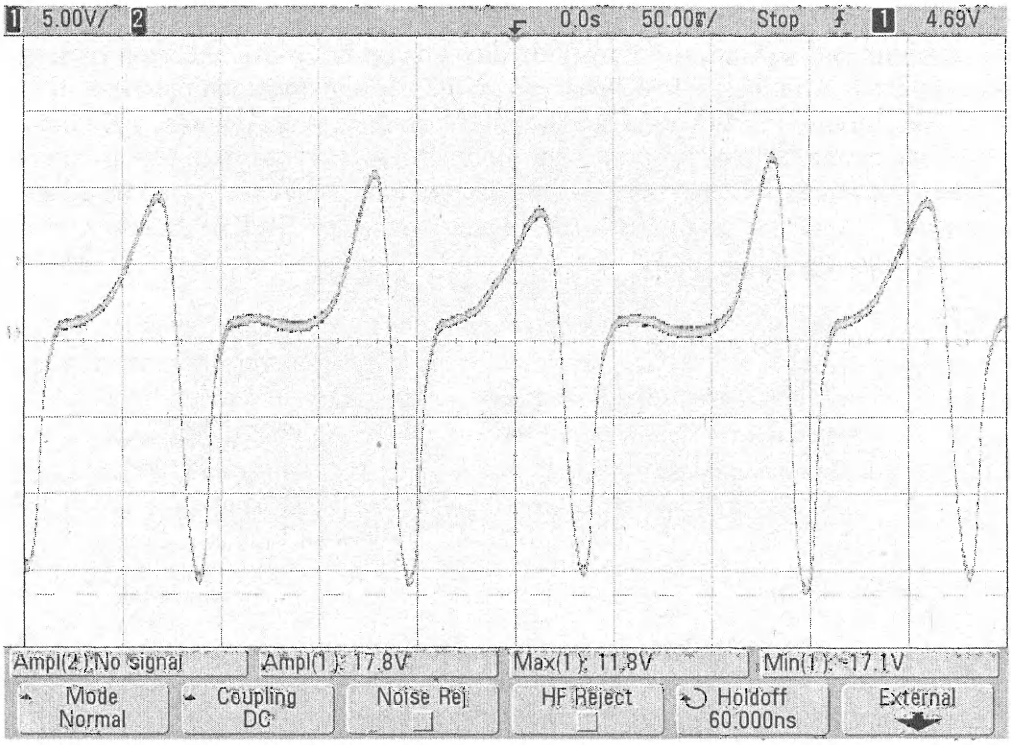


Рис. 1.22. Сигнал на выходе генератора Фарадея

Корпусом служит трубка из оргстекла подходящего диаметра и длины. Внутри нее находится магнит. Поверх трубки намотана катушка из нескольких сотен витков медного изолированного провода. Концы трубки закрыты заглушками. Для генерирования напряжения конструкцию встряхивают, магнит перемещается по трубке и создает в медном проводе переменное напряжение, которое можно выпрямить и отфильтровать, например, при помощи схемы, показанной на рис. 1.20. Единственная проблема состоит в том, что вам придется трясти эту трубку ровно столько времени, сколько вы хотите вырабатывать напряжение. Как только магнит останавливается, генерация напряжения прекращается и сохраняется только остаточное напряжение на конденсаторе. Но во многих случаях и этого оказывается достаточно. Одно из возможных решений — применение ионисторов. Однако для их зарядки до требуемого напряжения вам может понадобиться потратить много времени и сил.

В качестве стабилизатора к такому источнику рекомендуем использовать микросхему LP2950.

На рис. 1.22 показана осциллограмма сигнала на выходе генератора Фарадея. Ее размах превышает 17 В.

Питание от энергии радиоволн

Радиоволны вездесущи, и поэтому от них можно получать энергию (при помощи подходящей антенны) и преобразовывать ее в постоянное напряжение. К сожалению, для этого требуется либо значительная мощность источника, либо большая антенна, либо близость к радиопередатчику. Во многих коммерческих системах радиочастотная энергия для таких целей излучается специально. Одно из подобных применений — система радиочастотной идентификации (RFID), блок-схема которой показана на рис. 1.23.

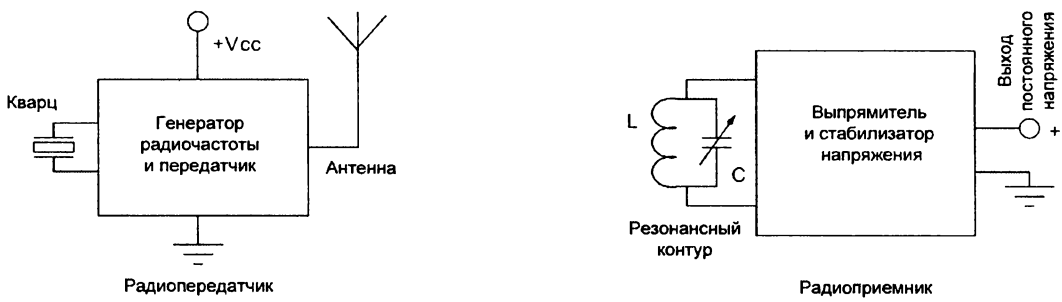


Рис. 1.23. Принцип питания от источника радиочастоты

Система состоит из передатчика немодулированного сигнала на подходящей частоте. Рабочая частота определяется кварцем. Чем выше частота, тем меньшая передающая антенна потребуется. Передатчик питается постоянным напряжением. Излучаемый сигнал принимается резонансным контуром (состоящим из катушки индуктивности и конденсатора переменной емкости), настроенным на частоту пе-

редатчика. Сигнал от резонансного контура поступает на диодный выпрямитель, фильтр и далее на стабилизатор напряжения. На выходе стабилизатора присутствует требуемое рабочее напряжение. Такая система может выдавать несколько милливатт мощности при расстоянии между передатчиком и приемником в несколько десятков сантиметров.

Выполненная по этому принципу реальная система описана в документе "Wireless battery energizes low-power devices": **www.edn.com/article/CA6501085.html**.

Инструменты для разработки аппаратного обеспечения

Для разработки и изготовления описанных в этой книге конструкций мы использовали некоторые общедоступные инструменты:

- Паяльник мощностью 35 Вт с насадкой для пайки мелких предметов. Еще лучше приобрести паяльную станцию, поскольку она обеспечивает изолированное питание нагревателя паяльника, что предотвращает ток утечки с жала паяльника.
- Проволочный припой. Для пайки мы рекомендуем тонкий проволочный припой (мы применяем припой марки 26 SWG). На рис. 1.24 изображены припой и паяльник.

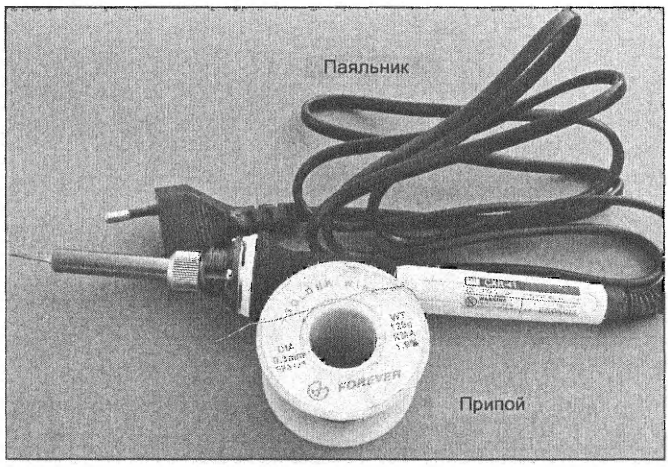


Рис. 1.24. Паяльник и проволочный припой

- Медная оплетка будет полезна для очистки компонентов от припоя.
- Увеличительное стекло пригодится для осмотра печатных плат, паяных соединений и т. д. Увеличительное стекло и медная оплетка показаны на рис. 1.25.
- Цифровой мультиметр (рис. 1.26) с возможностью измерения напряжения, тока и сопротивления полезен для тестирования и измерений.
- Тонкий пинцет для сгибания выводов компонентов.
- Кусачки для обрезки выводов компонентов.

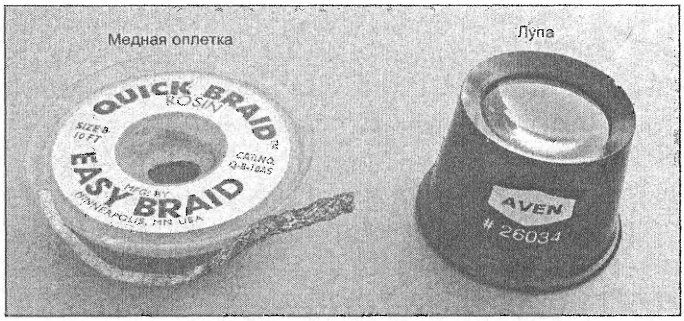


Рис. 1.25. Медная оплетка и увеличительное стекло

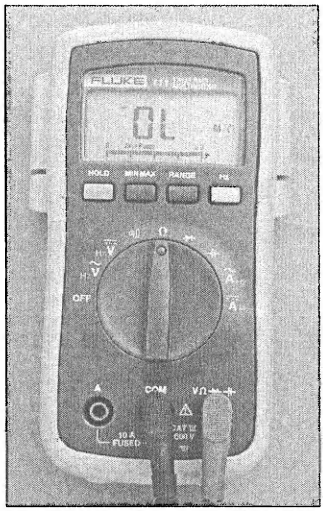


Рис. 1.26. Цифровой мультиметр

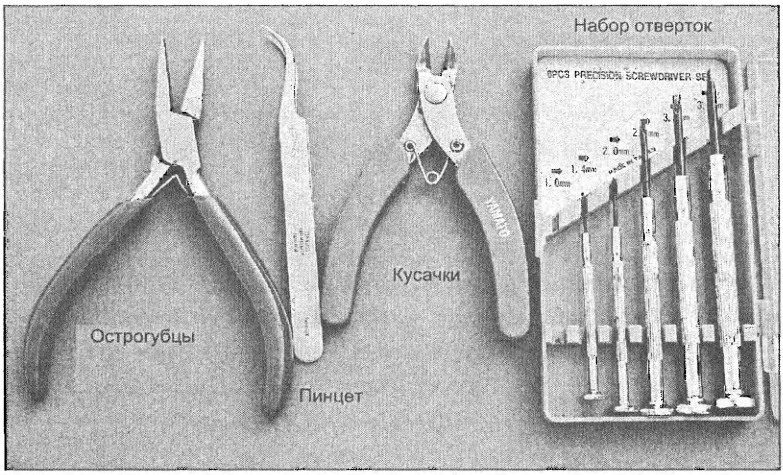


Рис. 1.27. Острогубцы, пинцет, кусачки и набор отверток

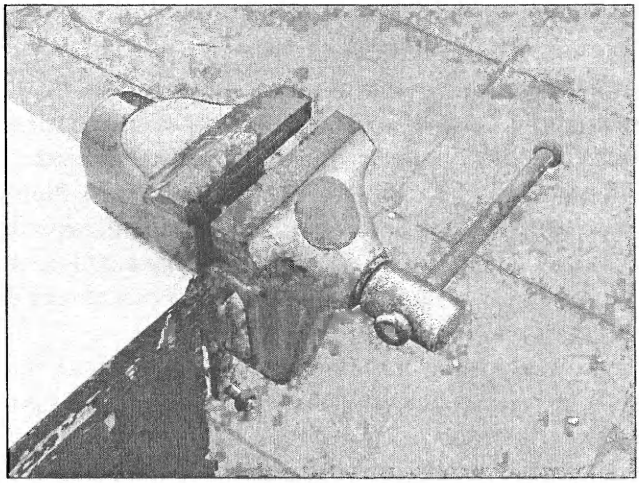


Рис. 1.28. Тиски

- Острогубцы для затягивания винтов и т. п.
- Набор отверток. Пинцет, кусачки, острогубцы и набор отверток показаны на рис. 1.27.
- Винты и гайки размера М3 для крепления кронштейнов и печатных плат.
- Дрель (ручная тоже подойдет) с набором сверел. Понадобится для сверления отверстий в печатных платах, корпусах и т. п.
- Верстачные тиски (рис. 1.28) с захватом в три дюйма (≈7,5 см) для закрепления печатных плат, отпиливания и т. д.

Разработка программного обеспечения

Преимущества программируемой системы (в данном случае микроконтроллера tinyAVR) невозможно реализовать без создания эффективного программного кода. В этой книге мы будем использовать для программирования язык С, синтаксис которого соответствует компилятору AVR-GCC (лицензия GNU).

Язык С — это язык программирования высокого уровня, поэтому написанный на нем код должен быть преобразован в тот машинный язык, который ваш контроллер понимает и может выполнять. Такое преобразование выполняет компилятор. Контроллеры Tiny "понимают" только двоичный формат и должны получать последовательность байтов. Подлежащие передаче в контроллер байты хранятся в виде файла, где они записаны в шестнадцатеричной системе счисления. Поэтому должен присутствовать такой инструмент, который преобразовывает код на языке С в шестнадцатеричный файл. Существует множество разных компиляторов с языка С для микроконтроллеров AVR, но по вполне понятным причинам мы остановились на AVR-GCC. WinAVR имеет хорошую интегрированную среду разработки (для операционной системы Windows).

Помимо хороших руководств по библиотеке AVR C, в WinAVR есть две основные программы:

- Programmer's Notepad — это универсальная интегрированная среда разработки для программирования на нескольких языках. Программа интегрирована с компилятором WinAVR. Для запуска Programmer's Notepad зайдите в меню **Windows | Programs | WinAVR (version) | Programmers Notepad**. На рис. 1.29 показано окно программы. Вы видите, что она имеет множество вкладок. Открыта самая важная вкладка — Tools. На этой вкладке три команды:
  - **Make All** — компилирует программу при помощи запуска файла MAKEFILE и генерирует шестнадцатеричный файл.
  - **Make Clean** — удаляет все шестнадцатеричные файлы и прочие зависимости. Обычно запускается перед повторной компиляцией программы.
  - **Make Program** — записывает ваш шестнадцатеричный файл в микроконтроллер, однако для этого требуется специальный программатор ISP.

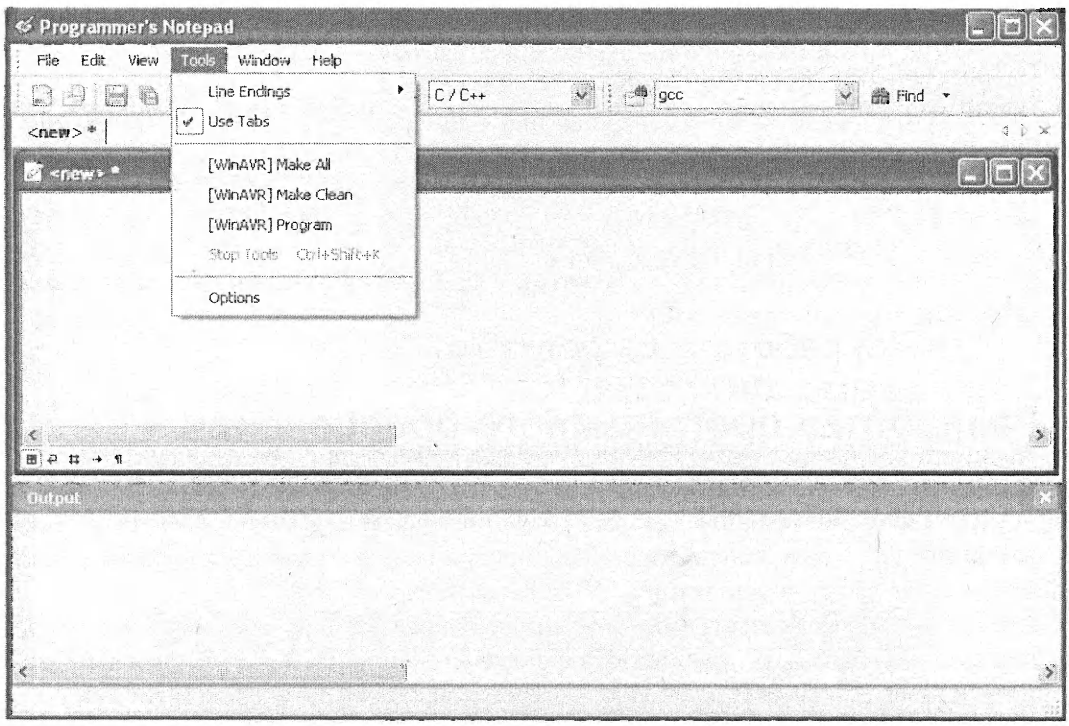


Рис. 1.29. Окно программы Programmer's Notepad

- MAKEFILE Template. Для преобразования вашего кода на языке C в шестнадцатеричные файлы нужно выполнить несколько задач: предварительную обработку, компиляцию, компоновку и загрузку. Для выполнения каждой из них компилятору GCC (компилятор GNU C) нужно выдать соответствующую команду. Вручную это делать весьма утомительно. В такой ситуации помогает утилита

MAKEFILE, которая собирает все команды и выдает их компилятору. WinAVR имеет базовый шаблон MAKEFILE, который вы можете подстроить под свои потребности. Для его запуска нужно зайти в меню **Windows | Programs | WinAVR (version) | mFile**. Выполните настройки и сохраните файл. Обратите внимание, что для начинающего написать файл MAKEFILE "с нуля" трудно. Если вы не чувствуете себя уверенно в опциях MAKEFILE, то лучше воспользоваться примером файла MAKEFILE, имеющимся в нашей книге, сделав в нем минимальные изменения под ваши потребности.

Работа с WinAVR и его компонентами на начальных стадиях может показаться несколько сложной. Программа AVR Studio компании Atmel хотя и позволяет легко управлять проектами на языке C и автоматически работать с командой `make` (которая нужна для компиляции написанного для компилятора GCC кода), но все равно требует WinAVR для компиляции кода на языке C (поскольку в ней нет собственного компилятора языка C, а есть только встроенный ассемблер). В результате, чтобы начать программировать, вам придется инсталлировать как WinAVR GCC, так и AVR Studio. Самую свежую версию AVR Studio можно скачать по ссылке: **http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=2725**, а WinAVR — по ссылке: **http://sourceforge.net/projects/winavr/files**. Проекты этой книги компилировались непосредственно из Programmer's Notepad программы WinAVR, а команды `make` писались вручную в файле MAKEFILE. Однако вы можете пользоваться любым из двух методов. Краткое введение в программирование встроенных систем на языке C (для микроконтроллеров AVR) приведено в приложении 1. Указания по обоим методам приведены далее.

Начинаем работать с проектом в программе AVR Studio

Для запуска AVR Studio зайдите в меню **Windows | Programs | Atmel AVR Tools | AVR Studio 4**.

1. Для создания нового проекта выберите опцию **New Project** в меню **Project** (рис. 1.30).

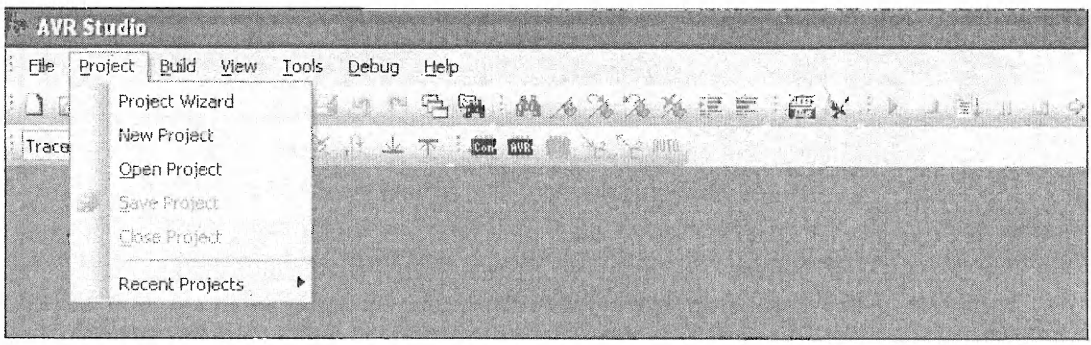


Рис. 1.30. Создание нового проекта в AVR Studio

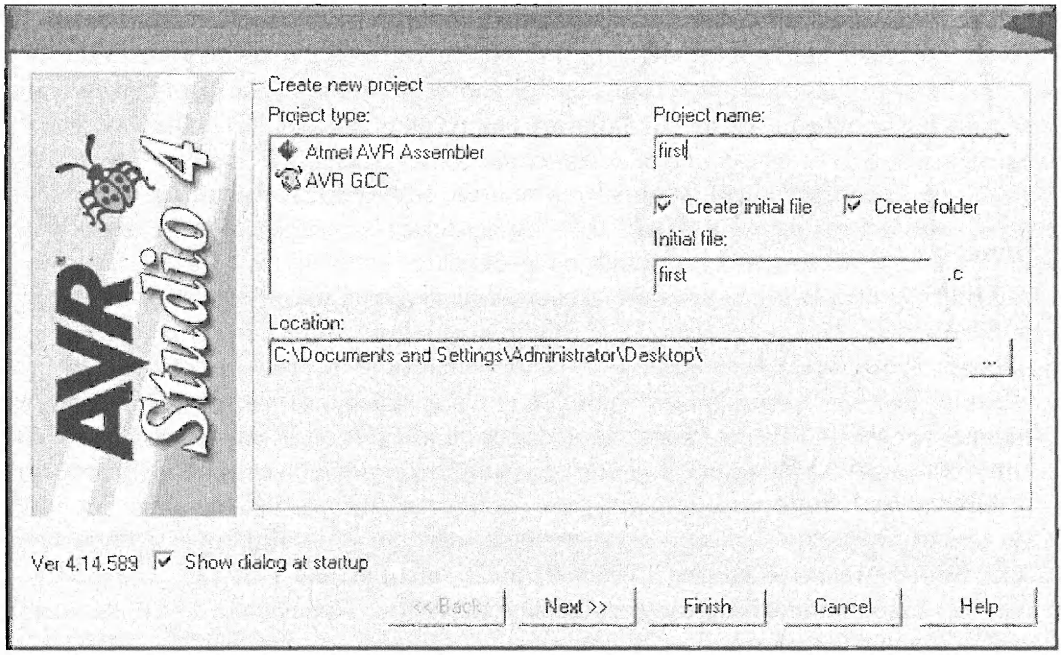


Рис. 1.31. Настройка проекта в AVR Studio

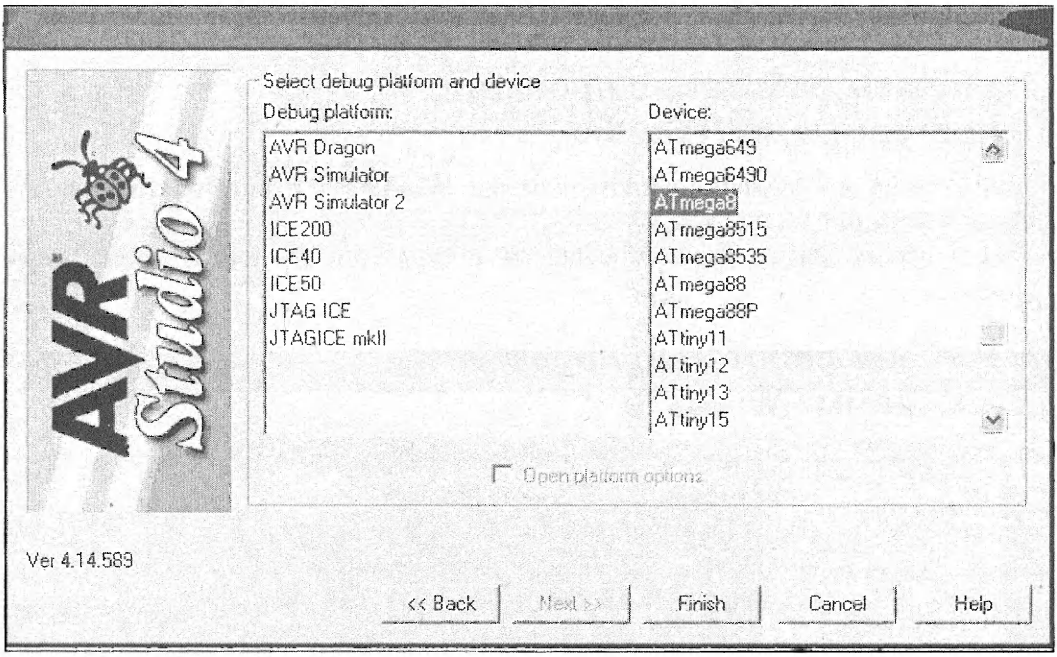


Рис. 1.32. Выбор контроллера в AVR Studio

2. Появится всплывающее меню (рис. 1.31). В поле **Project Type** выберите либо **AVR GCC**, либо **Atmel AVR Assembler** (в зависимости от используемого вами языка). Здесь показаны настройки для проекта на языке С. Установите флажки **Create Initial File** и **Create Folder** и дайте проекту подходящее название. Щелкните кнопку **Next**.
3. Появится следующее всплывающее меню (рис. 1.32). Выберите пункт **AVR Simulator** и в разделе **Device** укажите подходящий контроллер. Щелкните кнопку **Finish**, и вы увидите, что главный исходный файл открыт и в нем можно писать ваш код.
4. Обычно для удобства чтения вам потребуется разбивать код на части. В результате программа будет состоять из нескольких файлов. Чтобы присоединить дополнительные исходные файлы, щелкните правой кнопкой мыши пункт **Source Files** в разделе **AVR GCC** и выберите опцию либо **Add Existing Source File**, либо **Create New Source File**. Если вы используете существующие исходные файлы, убедитесь в том, что они скопированы в тот же каталог, что и ваш главный исходный файл (в тот каталог, который вы создали на шаге 2).
5. Напишите свой код в главном исходном файле.
6. В меню **Build** выберите команду **Build** (или нажмите клавишу <F7>), чтобы начать компиляцию вашей программы. Если в окне Build появилось сообщение "Build succeeded with 0 Warnings", то ошибок нет и ваш шестнадцатеричный файл создан. Если вы видите сообщение "Build succeeded" и несколько сообщений, то ваш шестнадцатеричный файл создан, но были выданы некие предупреждения. Рекомендуется изучить и по возможности устранить причины их появления. Шестнадцатеричный файл находится в подкаталоге "default" главного каталога проекта.
7. Для выполнения программы вы можете выбрать команду **Build and Run** меню **Build** (или нажать клавиши <CTRL>+<F7>). Пошаговое исполнение кода можно осуществить при помощи клавиши <F11>. После выполнения каждой команды можно отслеживать содержимое регистра контроллера, портов ввода/вывода и памяти.

Начинаем работать с проектом в программе WinAVR

Чтобы начать новый проект в WinAVR, необходимо выполнить следующие шаги:

1. Создать новый каталог на вашем персональном компьютере.
2. В этот каталог скопировать MAKEFILE любого проекта из этой книги (например, из *главы 1*). Опытный пользователь может написать свой собственный MAKEFILE. Шаблон MAKEFILE, приведенный в листинге 1.1, соответствует большинству ваших требований. Здесь указаны те места, где вам может понадобиться внести изменения в MAKEFILE. Строки, начинающиеся с символа #, считаются в MAKEFILE комментариями.

Листинг 1.1

```
# MCU name
MCU = ваш микроконтроллер
# Например,
# название MCU
MCU = attiny861
# здесь компилятору говорится о том, что нужно компилировать приложение под
микроконтроллер ATtiny861.
# формат выхода (формат может быть srec, ihex, binary)
FORMAT = ihex (выходной файл должен быть шестнадцатеричным)
# название целевого файла (без расширения)
TARGET = main (это название вашего главного файла)
# Список файлов на языке C (автоматически генерируемые зависимости языка C)
SRC = $(TARGET).c
# (эта строка указывает название исходного файла. Команда $(TARGET) заменяется
значением TARGET (которое равно main). Следовательно, ваш исходный файл должен
называться main.c).
# Если исходных файлов больше одного, то добавьте их выше или уберите символ
комментария из следующих строк:
#SRC += abc.c
SRC += def.c
```

Как уже упоминалось ранее, вам часто придется разбивать код на несколько файлов. Чтобы присоединить дополнительные исходные файлы, добавьте их, как показано здесь. В предыдущем примере файл abc.c не присоединяется (поскольку строка `SRC += abc.c` закомментирована), а def.c — присоединяется. Вы можете создать свои собственные исходные файлы и указать их здесь.

1. Затем создайте пустой текстовый документ и назовите его main.c (как указывалось ранее).
2. Измените MAKEFILE в соответствии с вашими потребностями.
3. Напишите свой код в файле main.c.
4. На вкладке **Tools** выберите пункт **Make All**. Если вы увидите код завершения 0, то ошибок нет и ваш шестнадцатеричный файл создан. Если вы увидели какой-то другой код завершения, то имеется ошибка и ее нужно устранить. Если код завершения равен 0, но вы видите какие-то предупреждения, то шестнадцатеричный файл был создан. Как уже говорилось ранее, постарайтесь по возможности устранить причины этих сообщений. Иногда предупреждения во время компиляции приводят к нестабильной работе проекта.

Язык ANSI C в сравнении со встроенным C

ANSI C — это документ, опубликованный институтом American National Standards Institute (ANSI) в качестве стандарта языка программирования C. Разработчики программного обеспечения обычно следуют этому стандарту при создании

кодов, которые работают на разных операционных системах. Даже Деннис Ритчи, создатель исходного языка C, во втором издании своей знаменитой книги C Programming Language (Prentice Hall, 1988) привел его к этому стандарту. Когда разработчик программного обеспечения пишет на языке C программу для персонального компьютера, она выполняется в среде операционной системы. После завершения программы управление процессором передается операционной системе, которая запускает другие (стоящие в очереди) программы. В случае многопроцессорной (или многопоточной) операционной системы на персональном компьютере выполняется много разных программ. Это делается при помощи квантования времени, т. е. каждой стоящей в очереди программе предоставляется (по очереди) доступ к процессору, памяти и вводу/выводу на некоторый промежуток времени (либо постоянной, либо переменной длительности).

По завершении программы она удаляется из очереди. Создается впечатление одновременного выполнения программ, но на самом деле процессор выполняет в каждый момент времени только одну последовательность команд (программу). Этим планированием занимается операционная система, которая постоянно поддерживает загрузку процессора.

Если же вы пишете код на языке C для микроконтроллера, то на нем выполняется только ваша программа, которая полностью управляет его ресурсами. В подобных приложениях операционные системы используются нечасто. Программа обычно работает в бесконечном цикле и не завершается.

Очевидно, что в двух рассмотренных случаях сам подход к программированию должен в определенной мере отличаться. Несмотря на этот факт, некоторые основные функции программирования на языке C (типы данных, циклы, операторы управления, функции, массивы и т. п.) аналогичны как в компиляторах ANSI C, так и во встроенных компиляторах языка C.

Изготовление печатной платы

Вы приняли решение относительно программных и аппаратных требований к вашему проекту, и это очень хорошо, но для настоящей реализации вашего устройства вам нужно изготовить схему со всеми ее компонентами (чтобы запрограммировать и протестировать ее). После того как количество компонентов превышает определенный предел, сборка макета устройства становится сложной. К тому же, схемы, выполненные на макетной плате, часто ведут себя непредсказуемо (из-за нежелательных замыканий или плохих контактов). Эти проблемы трудно отлаживать, и вы потратите много времени на их решение вместо того, чтобы тестировать ваше аппаратное решение и программное обеспечение. Другой способ проектирования схем — выполнение устройств на специально изготовленных печатных платах. Разработанные на печатных платах цепи более долговечны и менее подвержены сбоям (по сравнению с макетными платами). После изготовления и правильной пайки вы можете быть уверены во всех контактах и сосредоточиться на более важных вопросах: проектировании системы и разработке программного обеспечения. Существуют две разновидности проектирования и изготовления печатных плат, описанные в следующих разделах.

Использование макетной печатной платы

Этот подход обычно применяется энтузиастами и студентами, чтобы быстро спаять нужную цепь. Проектирование и изготовление нестандартной печатной платы занимает значительное количество времени, которое имеется не у всех. В таких случаях вполне разумно разместить компоненты на макетной печатной плате общего назначения и соединить их отрезками изолированных медных проводов. Мы рекомендуем одножильный облуженный медный провод в тефлоновой изоляции, который очень прочен и гибок. Универсальная макетная печатная плата имеет отверстия с шагом в 0,1 дюйма с контактными площадками. Такие платы бывают нескольких типов, два самых популярных: гетинаксовые (FR2) и стеклотекстолитовые (FR4). Первые дешевле, но они менее долговечные, чем вторые. Поэтому предпочтительнее текстолитовые платы. На рис. 1.33 показана пустая текстолитовая печатная плата, а на рис. 1.34 и 1.35 — две стороны макетной платы с собранной на ней схемой. Иногда полезно тестировать и проверять на такой плате небольшие части вашей системы до того, как начинать разрабатывать печатную плату для всего устройства. Ошибки подключения на такой плате легко исправить (поскольку подключения выполнены проводами, которые всегда можно перепаять с одной точки на другую). На готовой печатной плате это сделать очень трудно.

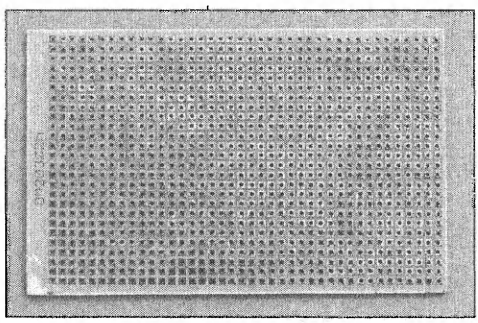


Рис. 1.33. Универсальная макетная печатная плата

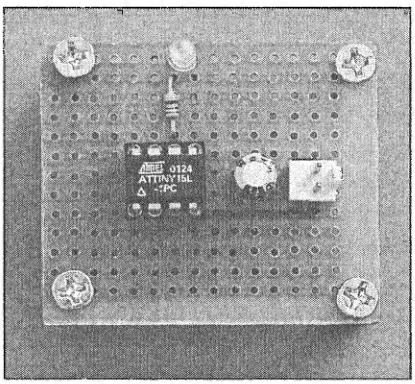


Рис. 1.34. Макет устройства (вид со стороны компонентов)

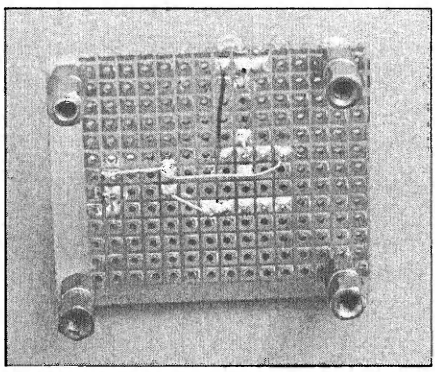


Рис. 1.35. Макет устройства (вид со стороны пайки)

Разработка собственной печатной платы

Это более "продвинутый" способ проектирования схем. Выполненные на печатной плате схемы имеют минимальную вероятность возникновения сбоев из-за плохих контактов (если плата разработана правильно). Первый шаг изготовления платы собственной конструкции — применение специального программного обеспечения для проектирования, компоновки и разводки платы. Имеется много программ для проектирования печатных плат, например: Proteus, Orcad и EAGLE. Для всех проектов этой книги мы использовали бесплатную версию программы EAGLE от компании CadSoft. Платы разводились так, чтобы максимальное количество дорожек шло в одном слое, т. к. изготовить одностороннюю плату гораздо дешевле и проще, чем двухслойную. Краткое руководство по проектированию печатных плат при помощи программы EAGLE приведено в приложении 2.

Спроектированную печатную плату можно изготовить многими способами. Наиболее распространены два — травление и фрезерование. При травлении используются химикаты, пленки, трафареты (и т. п.), которые помогают удалить ненужную медь с печатной платы. При фрезеровании лишнюю медь физически удаляют фрезой. Фрезерование (в отличие от травления) можно выполнять непосредственно из программного обеспечения для проектирования печатных плат. Бóльшая часть плат из этой книги была выполнена на фрезерном станке Roland Modela MDX-20, который изготавливает односторонние печатные платы, поэтому все наши проекты были адаптированы под его возможности. Для массового производства фрезерование слишком медленно (поскольку все экземпляры делаются последовательно), однако отдельные образцы изготавливаются очень быстро. Травление в массовом производстве быстрее (поскольку можно повторно использовать промежуточные пленки), но при небольшом количестве изделий оно становится слишком дорогим и медленным.

Поскольку это для вас всего лишь хобби, то возможно, что изготовление собственной печатной платы вам и не понадобится. Однако встречаются такие ситуации, когда вам нужно будет несколько экземпляров вашего устройства или когда сборка схемы на макетной печатной плате слишком утомительна (из-за сложности и большого числа соединений), либо вам может понадобиться отдать ваш проект на экспертизу. В таких ситуациях придется изготавливать печатную плату самостоятельно или заказывать ее у сторонних производителей.

Проект 1. Программа "Hello World!" в мире микроконтроллеров

После того как мы описали все элементы и компоненты проекта (и всю специфику создания проектов для микроконтроллеров AVR), предлагаем простой проект для иллюстрации. В нем есть все элементы, показанные на рис. 1.15. В схеме имеется два светодиода и две кнопки, а также кнопка сброса. Задача устройства — изменять состояние светодиодов при нажатии и отпускании кнопок. Проект имеет такое название потому, что вводит вас в мир микроконтроллеров tinyAVR.

На рис. 1.36 и 1.37 приведены два варианта принципиальной схемы этого устройства. Обе схемы идентичны, они иллюстрируют два самых популярных стиля изображения схем. На рис. 1.36 все соединения между компонентами показаны явно (соединительными линиями). На рис. 1.37 сигналам присвоены названия (например, PB3 — это контакт 2 микроконтроллера). Контакт 2 соединяется с LED1. Поэтому название сигнала PB3 присваивается и контакту 2, и катоду LED1. Похожие названия сигналов даны и остальным соединениям.

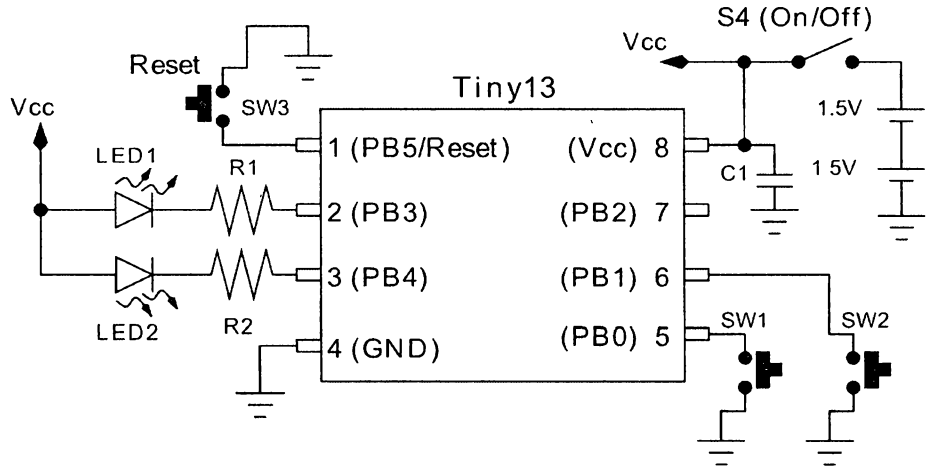


Рис. 1.36. Принципиальная схема проекта "Hello World!"

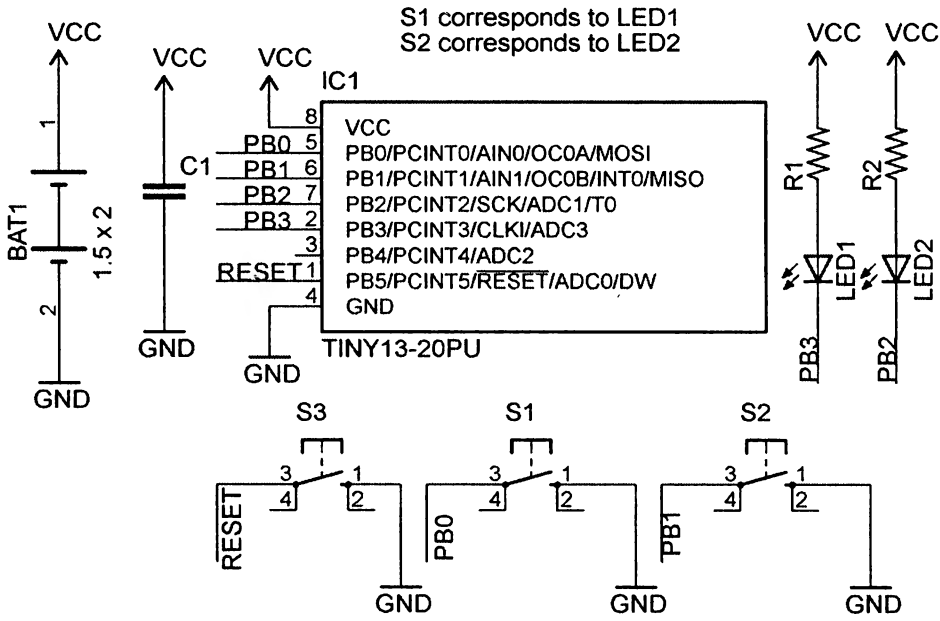


Рис. 1.37. Другой стиль схемы для "Hello World!" (S1 соответствует LED1, S2 — LED2)

Давайте рассмотрим элементы схемы рис. 1.36 (рис. 1.37). Схема питается от двух щелочных батареек размера AA. Как упоминалось ранее, такие батареи имеют номинальное напряжение на выводах 1,5 В. Две батареи дают напряжение 3 В. Рабочее напряжение Tiny13V составляет от 1,8 до 5,5 В, так что 3 В вполне подходит. По мере разряда батарей напряжение будет падать, но схема будет продолжать работать до тех пор, пока напряжение питания не снизится до 1,8 В. Светодиоды видимого света (в отличие от инфракрасных) имеют определенное напряжение включения, которое зависит от цвета (1,8 В — для красного и 3,5 В — для белого). Таким образом, для этого проекта следует выбрать красные светодиоды. На рис. 1.38 показана компоновка платы со стороны компонентов, а на рис. 1.39 — со стороны пайки. Вы видите, что плата в основном разведена в слое пайки (со стороны компонентов есть только одна перемычка). Ее легко изготовить при помощи фрезерования, описанного в предыдущем разделе. Внешний вид собранного образца устройства показан на рис. 1.40.

Компоновку платы в программе EAGLE (вместе со схемой) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

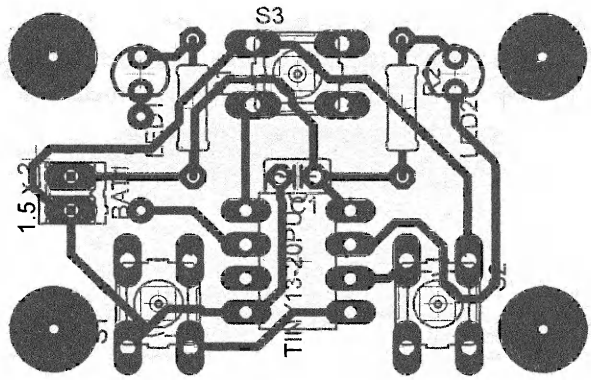


Рис. 1.38. Печатная плата проекта "Hello World!" со стороны пайки

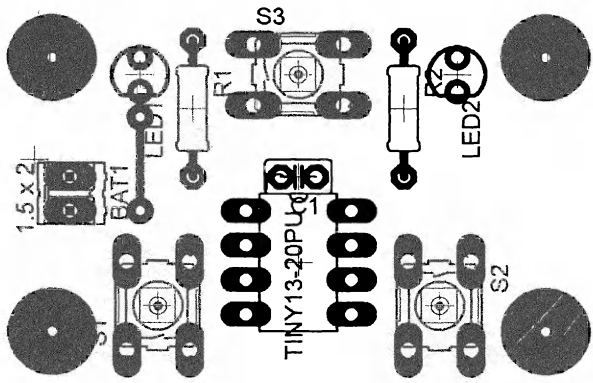


Рис. 1.39. Печатная плата проекта "Hello World!" со стороны компонентов

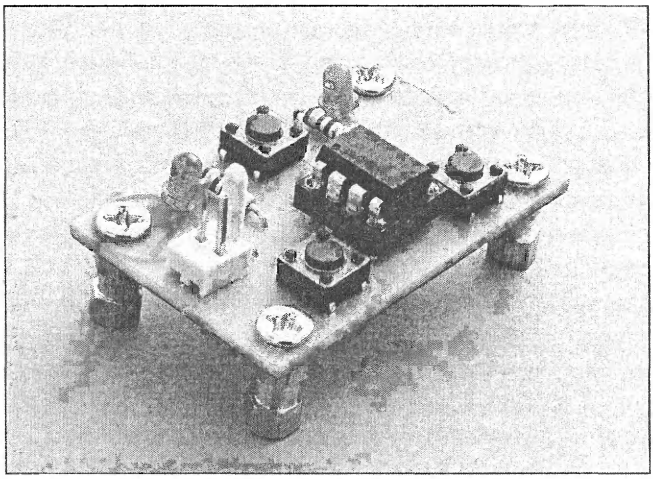


Рис. 1.40. Собранное устройство "Hello World!"

Код программы был написан таким образом, что левая кнопка включает левый светодиод, а правая — правый. То есть если правый светодиод выключен, и вы нажимаете и отпускаете правую кнопку, то она включает правый светодиод. Код программы для микроконтроллера Tiny13V на языке С приведен в листинге 1.2.

Листинг 1.2

```
//Include Files
#include<avr/io.h>
#define F_CPU 128000UL
#include<util/delay.h>

int main(void)
{
 DDRB |= 1<<2|1<<3;//Declare as outputs
 PORTB |= 1<<2|1<<3;
   //Выключить светодиоды
 DDRB &= ~(1<<0|1<<1);//Объявление ввода
 PORTB |= (1<<0|1<<1);//Включение нагрузки
 while(1)
 {
  //кнопка 1
  if(!(PINB&(1<<0))) //если есть нажатие
  {
   _delay_ms(10);//подавление дребезга
   while(!(PINB&(1<<0)));
      //ждем отпускания
```

```
    _delay_ms(10);//подавление дребезга
    PORTB^= (1<<3);//переключение
   }
   //кнопка 2
   if(!(PINB&(1<<1)))//если есть нажатие
   {
    _delay_ms(10);//подавление дребезга
    while(!(PINB&(1<<1)));
       //ждем отпускания
    _delay_ms(10);//подавление дребезга
    PORTB^= (1<<2);//переключение
   }
  }
}
```

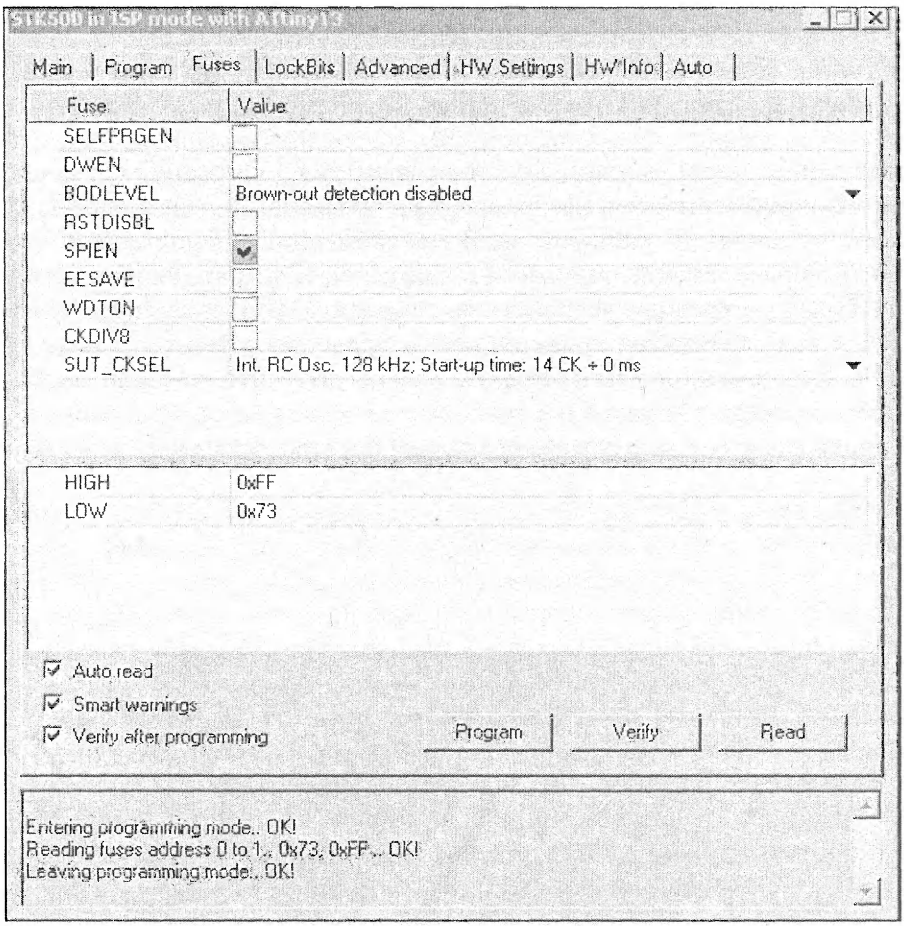


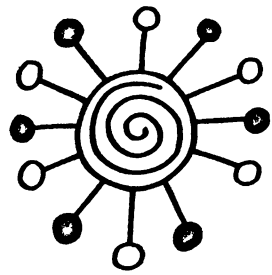
Рис. 1.41. Установка fuse-битов для микроконтроллера Tiny13

Листинг 1.2 демонстрирует общий стиль программирования нашей книги. Файлы заголовков специфичны для компилятора AVR-GCC. Макрос `F_CPU` используется для передачи в компилятор значения рабочей частоты. Программа работает в бесконечном цикле. Для каждой кнопки есть один блок `if`, который сначала проверяет нажатие кнопки. Если кнопка нажата, то происходит ожидание ее отпускания, и при отпускании выполняется нужное действие (переключение светодиода). Задержка в 10 мс после каждого нажатия и отпускания кнопки предназначена для подавления дребезга контактов. Тем, кто только начинает программировать на языке С для микроконтроллеров AVR, рекомендуется прочитать приложение 1, чтобы лучше понять приведенный код. Откомпилированный исходный код вместе с файлом MAKEFILE можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

AVR программируется через STK500 в режиме ISP. Установка fuse-битов показана на рис. 1.41. Tiny13 настроен для работы на частоте 128 кГц внутреннего RC-генератора. Идея состоит в том, чтобы тактировать AVR самой низкой частотой, поскольку потребление энергии цифровыми схемами AVR (схема типа CMOS) прямо пропорционально рабочей тактовой частоте, а мы хотим свести затраты энергии к минимуму.

Заключение

Мы завершили вводную часть книги и рассмотрели широкий круг вопросов, которые необходимо учитывать при разработке проектов. В этой главе заложены основы для понимания следующих глав. В ней также описан простейший проект, для которого мы предоставили полный исходный код и компоновку печатной платы. Создание более интересных устройств на основе контроллеров tinyAVR начнется в *главе 2*. Для всех остальных проектов мы не приводим в тексте полный исходный код и разводку печатной платы. Все это можно получить по ссылкам, указанным для каждого проекта. Однако для самых важных фрагментов кода даны пояснения. В следующей главе мы рассмотрим несколько простых проектов со светодиодами.



Глава 2

Простые устройства со светодиодами

В этой главе мы опишем несколько простых устройств с использованием светодиодов. Светодиоды — весьма популярные электронные компоненты, а последние технологические достижения позволили создать светодиоды, которые излучают свет практически во всем видимом спектре. Светодиоды были изобретены в начале 1960-х годов. Самые первые светодиоды были красными, следом за ними появились желтые. Синие светодиоды появились в начале 1970 годов, однако яркость их свечения была небольшой. Яркие синие светодиоды появились лишь в конце 1990-х годов. Сейчас внимание разработчиков сосредоточено на белых светодиодах, которые пригодны для освещения. Современные мощные и яркие белые светодиоды с потребляемой мощностью в 10 Вт выдают световой поток интенсивностью до 1000 люменов. Главные преимущества белых светодиодов для освещения — их долговечность (100 тысяч часов) и высокая эффективность. Таким образом, со светодиодами можно реализовать интересные проекты, и в этой главе мы рассмотрим некоторые несложные конструкции со светодиодами (в том числе многоцветными).

Общие сведения о светодиодах

Светодиоды (LED) — это замечательные электронные компоненты. Они бывают самых разных размеров и цветов (рис. 2.1). Многие светодиоды заключены в прозрачный корпус, поэтому цвет их свечения по внешнему виду определить невозможно. Светодиод излучает свет при прохождении тока в прямом направлении. На рис. 2.2 показан малогабаритный светодиод и его условное обозначение. Выводы светодиода разной длины: длинный — анод, короткий — катод.

Интенсивность излучаемого света пропорциональна электрическому току, проходящему через светодиод. Падение прямого напряжения на светодиоде зависит от ширины запрещенной зоны полупроводникового материала, т. е. можно сказать, что это напряжение связано с длиной волны излучаемого света. Красный светодиод имеет самую маленькую ширину запрещенной зоны, а синий — самую большую. Поэтому падение напряжения на красном светодиоде составляет 2 В, а на синем — 3,7 В. В табл. 2.1 приведены электрические и оптические характеристики различных светодиодов.

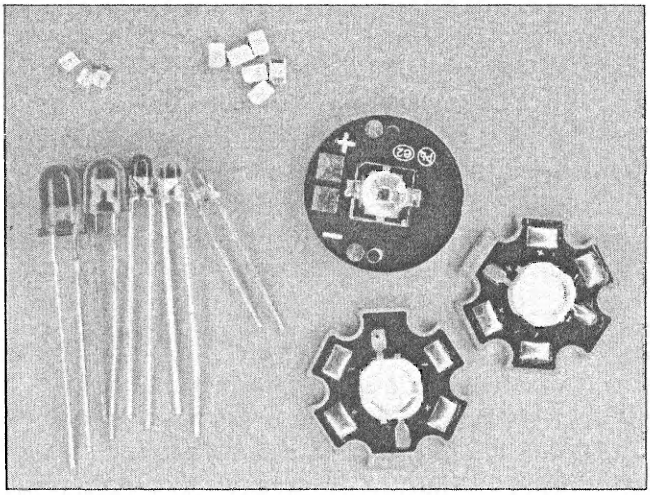


Рис. 2.1. Различные типы светодиодов

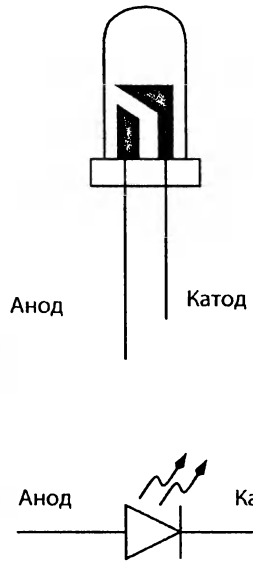


Рис. 2.2. Светодиод и его условное обозначение

Таблица 2.1. *Электрические и оптические характеристики светодиодов размером 5 мм (компания Lite-On Optoelectronics)*

Цвет свечения	Прямой ток $I_{пр}$, мА	Предельно допустимый ток I_{max}, мА	Типичное прямое напряжение $U_{пр}$, В	Угол видимости, град	Длина волны света, нм
Красный	20	120	2,0	30	635
Оранжевый	20	60	2,5	15	624
Желтый	20	90	2,0	20	591
Зеленый	20	100	3,5	15	504
Синий	20	100	3,7	20	470
Белый	20	100	3,5	20	Широкий спектр

Поскольку светодиод — это один из видов диода, то вполне естественно ждать от него таких же характеристик по напряжению и току, что и от обычного импульсного диода. Импульсные диоды изготавливают из кремния или германия и напряжение их включения составляет для кремниевых примерно 0,7 В, для германиевых — 0,2 В. Для изготовления светодиодов применяют арсенид галлия (GaAs), в который добавляют примеси для получения нужного цвета. Ширина запрещенной зоны GaAs составляет 1,45 эВ, а кремния — 1,1 эВ. Поэтому напряжение включения у диода из арсенида галлия должно быть выше, чем у кремниевого диода. На рис. 2.3 показана прямая ветвь вольт-амперной характеристики красного светодиода.

Рис. 2.3. Прямая ветвь вольт-амперной характеристики красного светодиода

Для включения светодиода нужен источник постоянного напряжения и дополнительная схема ограничения прямого тока. Прямой ток для конкретного светодиода подбирают исходя из требуемой интенсивности свечения и ориентируясь на предельно допустимый ток, который может выдержать этот светодиод. Нежелательна длительная работа при больших прямых токах, поскольку это может привести к повреждению светодиода. Максимально допустимое значение прямого тока вообще превышать нельзя. В самом простом варианте ограничителем тока служит резистор. При напряжении питания $U_{ип}$ (Vcc), падении напряжения $U_{пр}$ и желательном токе в 10 мА сопротивление резистора (в килоомах) должно быть равно

$$R = (U_{ип} - U_{пр})/10 \text{ мА}.$$

Пример: если наш светодиод имеет характеристику, показанную на рис. 2.3, а $U_{ип}$=5 В и падение напряжения равно $U_{пр}$=1,98 В, то величина сопротивления получится 302 Ом. Стандартные резисторы имеют номиналы 270 и 33 Ом, их можно соединить последовательно и получить 303 Ом, что нас вполне устроит. Если же в схеме будет один резистор в 270 Ом, ток через светодиод увеличится до 11 мА, а при резисторе в 330 Ом ток составит 9,15 мА.

Интенсивность свечения светодиода пропорциональна току, проходящему через него. Для иллюстрации этого свойства была собрана схема, показанная на рис. 2.4 и построен график. Результат этого небольшого эксперимента показан на рис. 2.5. Связь между интенсивностью свечения и током будет изучаться далее в этом же разделе и в последующих проектах.

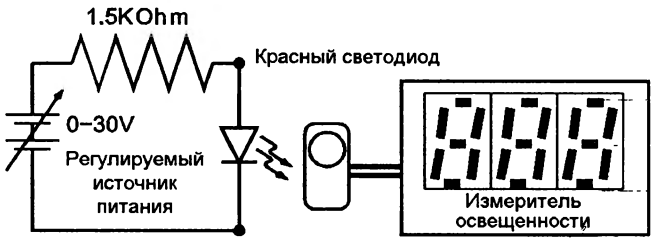


Рис. 2.4. Схема исследования светодиода

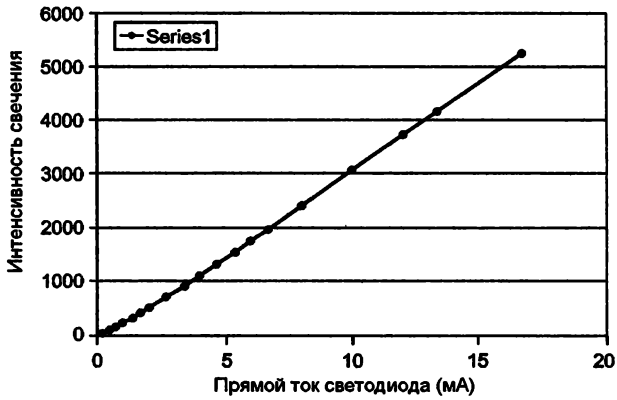


Рис. 2.5. График зависимости интенсивности свечения светодиода от тока через него

Типы светодиодов

Помимо изображенных на рис. 2.1, выпускаются двухцветные и трехцветные (RGB) светодиоды. На рис. 2.6 показан двухцветный светодиод с общим анодом и его условное обозначение. На рис. 2.7 приведена фотография двухцветных и трехцветных светодиодов.

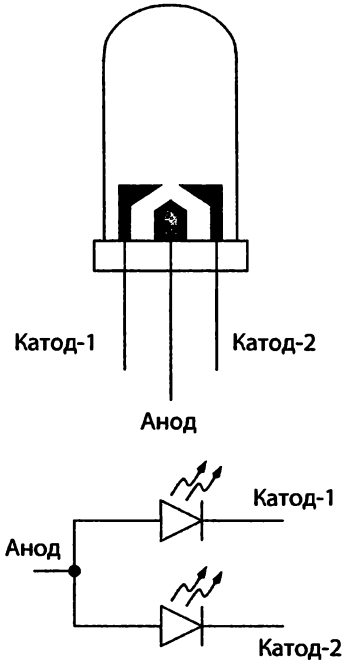


Рис. 2.6. Двухцветный светодиод с общим анодом и его условное обозначение

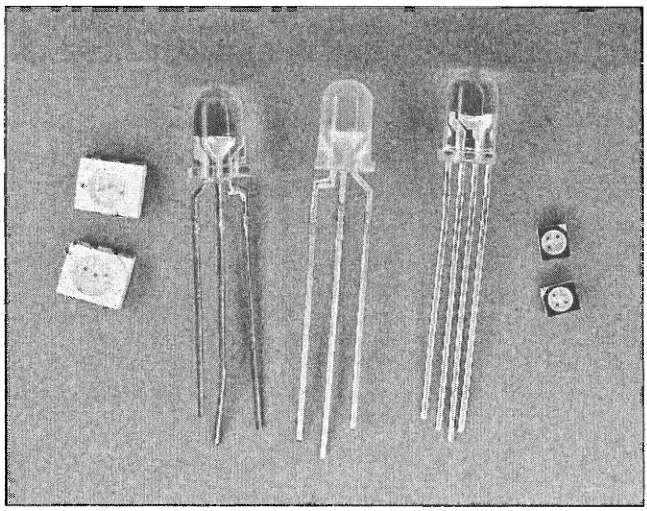


Рис. 2.7. Внешний вид двухцветных и трехцветных светодиодов

Бывают также двухцветные светодиоды с общим катодом. В одном корпусе могут совмещаться светодиоды любых цветов: красно-желтый, красно-зеленый, сине-желтый, сине-красный и т. д. Трехцветный светодиод объединяет в одном корпусе красный, зеленый и синий диоды. В таком приборе четыре вывода: общий анод и три катода (или общий катод и три анода). При помощи двухцветных светодиодов можно получать несколько цветов (одновременно включая оба светодиода). Так, при помощи двухцветного красно-зеленого светодиода можно получить желтый (включив одновременно красный и зеленый светодиоды). Если же красный светодиод включить "вполнакала", а зеленый полностью, то можно получить и другие оттенки желтого. Трехцветные светодиоды обеспечивают более широкую палитру цветов (путем управления интенсивностью трех основных цветов). В следующем разделе мы подробно обсудим, как управлять интенсивностью свечения светодиодов. На основе светодиодов изготавливают более сложные дисплеи: семисегментные, алфавитно-цифровые, столбчатые и точечно-матричные (о них речь пойдет в *главе 3*).

Управление светодиодами

Управление светодиодами заключается в их включении и выключении с помощью специальной схемы. Единственный способ включить и выключить светодиод в схеме на рис. 2.4 — подать и отключить напряжение питания. Однако светодиоды можно включать/выключать при помощи микроконтроллера и получать интересные световые узоры. На рис. 2.8 показана принципиальная схема, состоящая из микроконтроллера Tiny13 и пяти светодиодов.

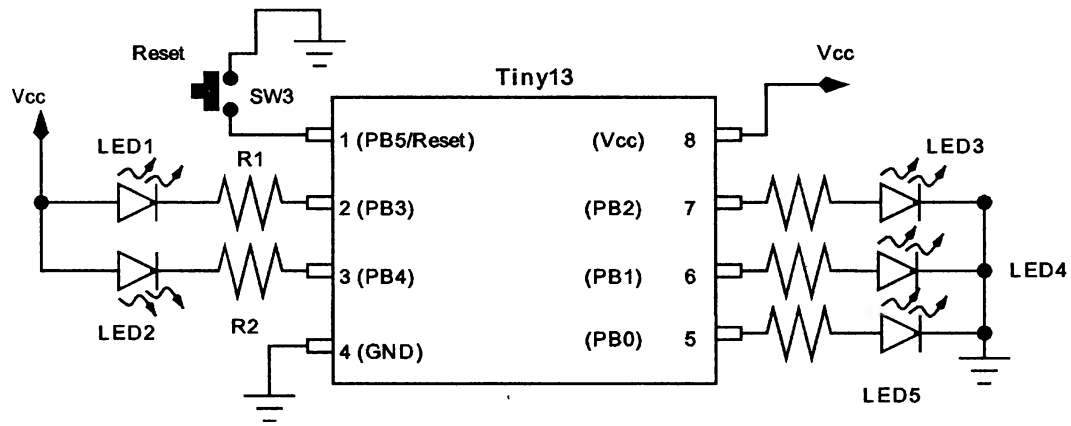


Рис. 2.8. Управление светодиодами от микроконтроллера

Все светодиоды подключены к контактам микроконтроллера через резисторы. Два светодиода (LED1 и LED2) подключены по схеме с общим анодом к источнику питания, а остальные три (LED1, LED2 и LED3) — по схеме с общим катодом к общей шине. Эти два разных способа использованы только для примера. Для мик-

росхем 74-й серии TTL-вентилей (например, 7400 или 74LS00) предпочтительнее подключение внешней нагрузки к источнику питания, чем к общей шине. Однако для современных микросхем КМОП (таких, как микроконтроллеры AVR) способ подключения светодиодов не имеет значения.

В схеме на рис. 2.8 светодиоды LED1 и LED2 загорятся, когда на соответствующем выходе контроллера будет логический нуль, а остальные светодиоды — когда на соответствующих выходах появится логическая единица. Величина сопротивления резистора зависит от требуемого тока через светодиод, который не должен превышать выходной ток микроконтроллера. Максимальный выходной ток микроконтроллеров AVR — 40 мА. Напряжение источника питания должно быть больше порогового напряжения включения светодиодов. Например, двух щелочных батареек (3 В) будет достаточно для включения красных светодиодов. Однако синие светодиоды работать не будут. Для их работы напряжение питания должно быть равно 5 В.

Микроконтроллер может переключать светодиоды с любой частотой. Однако если частота коммутации превышает 20 Гц, то светодиоды будут неприятно мерцать. При увеличении частоты, например до 100 Гц, мерцание исчезнет и светодиоды будут казаться постоянно включенными. На самом деле светодиоды будут включаться и выключаться с частотой 100 Гц, однако человеческий глаз не может реагировать на столь быстрое переключение. Это интересное явление (оно называется модуляцией интенсивности свечения) мы будем использовать для изменения интенсивности свечения светодиодов.

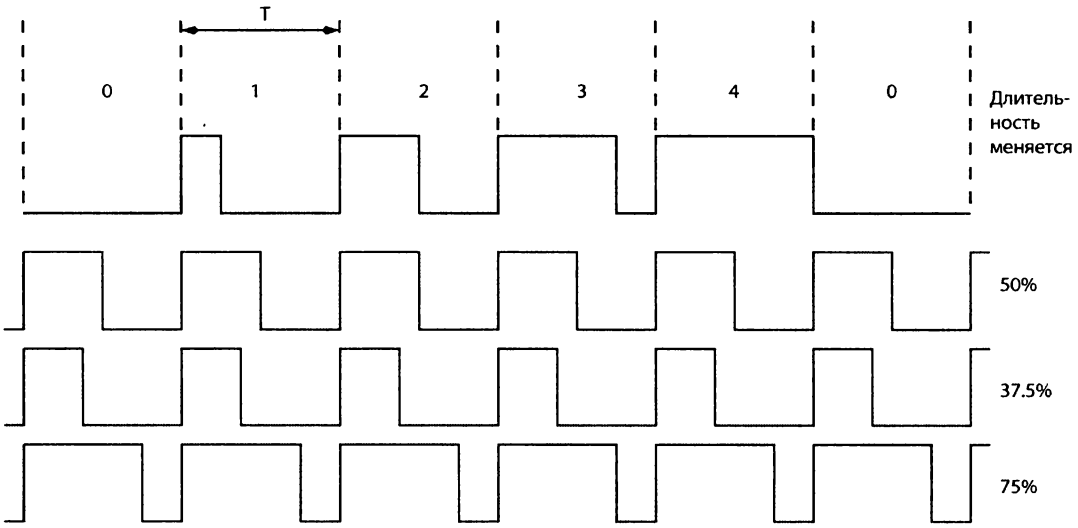


Рис. 2.9. Сигнал с широтно-импульсной модуляцией

На рис. 2.9 показан сигнал с широтно-импульсной модуляцией (ШИМ, PWM), имеющий частоту $F=1/T$ (показана на верхнем графике). Предположим, что частота F равна 100 Гц. При постоянной частоте сигнала меняем время нахождения сигнала

в состоянии логической единицы. Пусть это время равно $T1$. Отношение периода T к времени $T1$ называется скважностью сигнала. Сигналы, показанные на рис. 2.9, можно без труда сгенерировать при помощи микроконтроллерной схемы, приведенной на рис. 2.8. Если сигнал, обозначенный как 50%, подать на светодиод LED3 через контакт PB2 данной схемы, то наблюдатель увидит интенсивность свечения 50% от той, которая будет получена в случае подачи на контакт PB2 постоянной логической единицы (при котором яркость максимальна). Так происходит потому, что теперь средний ток через светодиод составляет 50% от максимального.

Если же на светодиод LED3 подать сигнал, помеченный как 75%, то интенсивность составит 75% от максимальной. Можно задать любое значение сигнала от 0% (минимальная интенсивность) до 100% (максимальная интенсивность). Сигнал с широтно-импульсной модуляцией можно сгенерировать либо программно, либо аппаратно (при помощи встроенных в микроконтроллер AVR таймеров). Использование аппаратных таймеров позволяет микроконтроллеру выполнять другие дополнительные задачи. Эти способы управления светодиодами будут встречаться во многих последующих проектах. Особенно удобно с помощью ШИМ управлять многоцветными светодиодами, создавая большое количество промежуточных цветов и оттенков.

Помимо управления интенсивностью свечения следует также обсудить способы соединения светодиодов. До настоящего момента мы рассматривали подключение только одного светодиода к одному контакту микроконтроллера. Кроме этого, светодиоды можно соединять последовательно или параллельно. Возможно последовательное соединение светодиодов с одним резистором (рис. 2.10).

Число светодиодов, последовательно подключаемых к выводу микроконтроллера, будет определяться напряжением включения светодиодов и напряжением питания. При напряжении питания, равном 5 В, можно последовательно подключить два красных светодиода. Но два синих светодиода подключить так не удастся, поскольку напряжение включения цепочки из двух синих светодиодов будет выше, чем напряжение питания +5 В. По той же причине не удастся последовательно подключить и три красных светодиода.

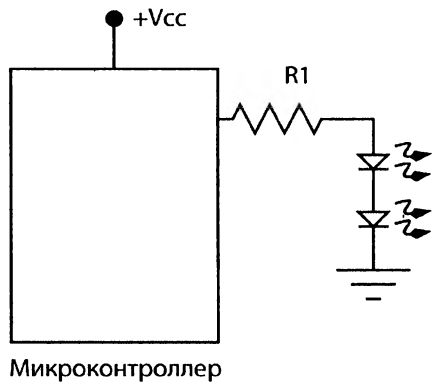


Рис. 2.10. Последовательное соединение светодиодов

Если нужно подключить три светодиода, то лучше их соединить параллельно, как показано на рис. 2.11.

Обратите внимание, что вместо одного резистора и параллельного подключения светодиодов мы выбрали вариант подключения по одному резистору на каждый светодиод и последующее параллельное соединение этих цепей. Так сделано потому, что светодиоды из одной партии могут иметь различные напряжения включения; и если при этом соединить параллельно несколько светодиодов, то светодиод с самым низким напряжением включения будет доминировать над остальными (потреблять больший ток и светиться ярче остальных). В худшем случае весь ток пойдет через один светодиод, а остальные вовсе не загорятся. При параллельном подключении нескольких светодиодов суммарный ток через все светодиоды должен быть меньше, чем максимальный ток выхода микроконтроллера. Если же ток через светодиоды превышает возможности микроконтроллера, то можно применить схему, показанную на рис. 2.12, где *n-p-n*-транзистор управляет несколькими последовательно соединенными светодиодами.

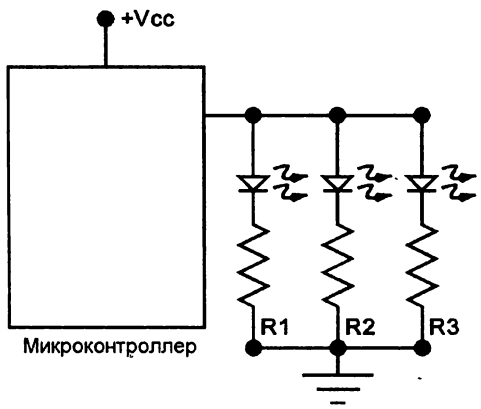


Рис. 2.11. Подключение параллельно соединенных светодиодов к контроллеру

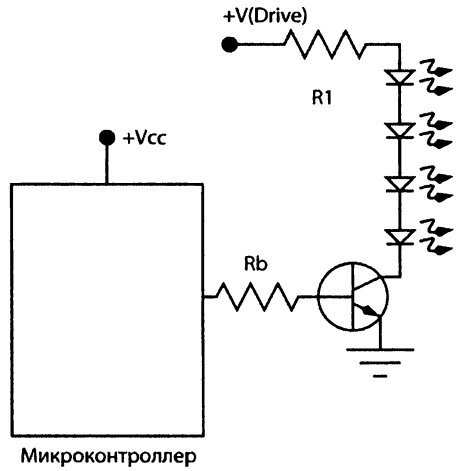


Рис. 2.12. Вариант подключения последовательно соединенных светодиодов через транзистор

Напряжение V(Drive), подаваемое на светодиоды, должно быть больше суммы напряжений включения всех последовательно включенных светодиодов. Резистор R1 определяет ток через светодиоды. Резистор Rb в цепи базы *n-p-n*-транзистора служит для ограничения тока базы; сопротивление Rb вычисляется по току коллектора (который течет и через светодиоды) и по коэффициенту усиления транзистора. Вот пример: предположим, что вы хотите соединить последовательно пять красных светодиодов и подать на них ток 30 мА. Из табл. 2.1 видно, что напряжение включения красного светодиода равно 2 В, следовательно, потребуется источник 10 В. Падение на выводах коллектора и эмиттера транзистора составит 0,5 В. Желательно получить напряжение V(Drive) в 15 В, поэтому сопротивление R1 = (15 – 10,5) В/30 мА = 150 Ом. Для дан-

ного случая подойдет транзистор малой мощности типа BC547. Обычно значение β для BC547 составляет 100, поэтому требуемый ток базы 30 мА/100 = 300 мкА. Если микроконтроллер питается напряжением +5 В, то за логическую единицу можно принять напряжение в 4,5 В. Падение напряжения V(be) на переходе "база-эмиттер" примерно равно 0,7 В. Таким образом, Rb = (4,5 – 0,7) В/300 мкА = 12,6 кОм. Следовательно, в качестве Rb вполне подойдет сопротивление 10 кОм. Для схемы, приведенной на рис. 2.13, необходимо, чтобы все последовательно соединенные светодиоды были одного цвета. При расчете нужно учесть суммарное падение напряжения на всех этих светодиодах, чтобы определить напряжение V(Drive) и значения R1 и Rb.

На рис. 2.13 показано, как можно соединить несколько светодиодов параллельно и подключить к контроллеру через *n-p-n*-транзистор. Такая схема нужна, когда суммарный ток через светодиоды превышает выходной ток микроконтроллера. Предположим, что вы хотите управлять десятью параллельными светодиодами (каждый с током в 20 мА). Потребуется ток в 200 мА, что гораздо больше, чем может дать один вывод микроконтроллера. Однако для управления этими светодиодами будет вполне достаточно *n-p-n*-транзистора средней мощности с максимальным током коллектора 1 А. Расчет сопротивления резистора для каждого из светодиодов (а также сопротивления Rb) аналогичен рассмотренному ранее.

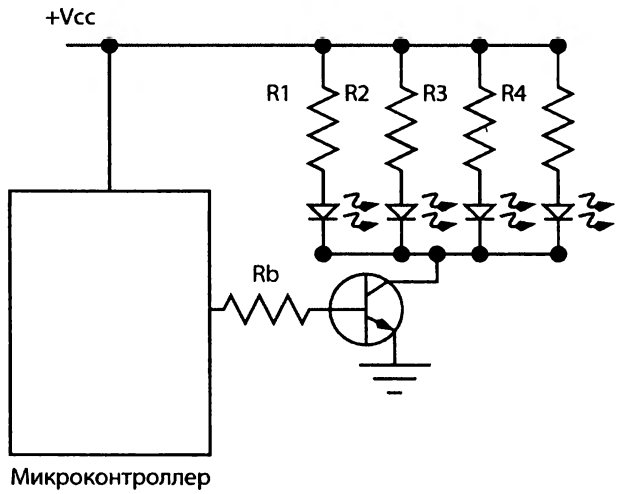


Рис. 2.13. Вариант подключения параллельно соединенных светодиодов через транзистор

Проект 2. Мерцающая светодиодная свеча

При всем разнообразии современных способов освещения свечи все равно продолжают привлекать человека. Ужин при свечах считается более романтичным, чем при обычном освещении (даже приглушенном). Возможно, что этот эффект создает именно мерцание свечи, поэтому стоит попробовать его воспроизвести.

В рассматриваемом проекте мы покажем, как можно сымитировать мерцающую свечу с помощью светодиода. На рис. 2.14 показана блок-схема мерцающей светодиодной свечи. Просто зажечь светодиод — не проблема. Секрет имитации свечи состоит в воспроизведении ее мерцания. Пламя свечи колеблется случайным образом, а иногда интенсивность света меняется от движения воздуха. При использовании светодиода заставить пламя колебаться не удастся, но можно добиться случайного изменения интенсивности свечения (даже при отсутствии движения воздуха). На блок-схеме показан генератор случайных чисел, который выдает сигнал в цепь управления интенсивностью свечения светодиода.

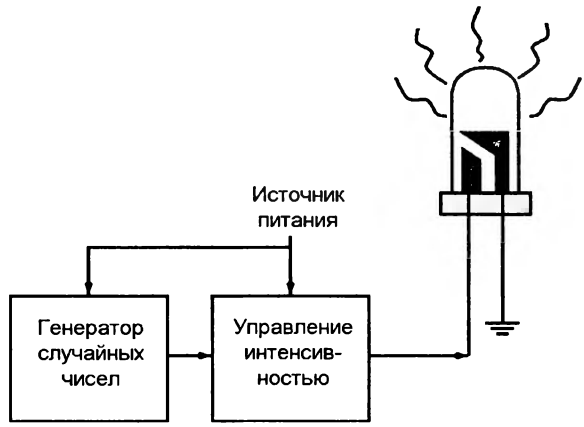


Рис. 2.14. Блок-схема светодиодной свечи

Получение случайных чисел всегда вызывало много споров. Долго обсуждали, может ли вообще какой-либо генератор чисел (или шумов) быть действительно случайным. Ответ прост — "нет". Любой генератор "случайных" чисел начинает повторяться через некоторый интервал времени. Если этот период достаточно большой, то реализация кажется полностью случайной. Поэтому мы называем такие источники генераторами псевдослучайных чисел. Обычно генераторы псевдослучайных чисел служат для привнесения в поведение системы некоторой неопределенности. Однако можно и без всякого дополнительного аппаратного обеспечения реализовать программный генератор псевдослучайных чисел — при помощи так называемых линейных сдвиговых регистров с обратной связью (Linear Feedback Shift Registers, LFSR).

На рис. 2.15 показана блок-схема такого LFSR-генератора. LFSR представляет собой сдвиговый регистр, на вход которого подается исключающее ИЛИ (XOR) по некоторым битам самого регистра. Используемые для операции XOR позиции называются отводами. LFSR должен быть инициализирован неким ненулевым начальным значением. Если начальное значение будет нулевым, то LFSR никогда не выйдет из своего начального состояния, поскольку XOR по любому количеству нулевых битов дает нуль. LFSR имеет интересное свойство: если тщательно выбрать отводы, то для n-разрядного LFSR значения на выходе начнут повторяться

через 2^n–1 тактов. На рис. 2.15 показан 10-разрядный регистр с длиной последовательности, равной 1023. 16-разрядный LFSR будет иметь длину последовательности 65 535 (и т. д.).

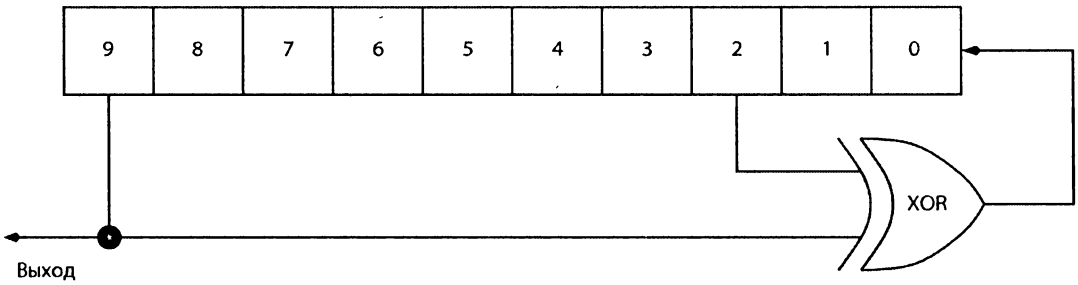


Рис. 2.15. Генератор случайных чисел

Такой LFSR называется также генератором Фибоначчи. Существует еще один тип LFSR — генератор Галуа, в котором разряды, не используемые в качестве отводов, сдвигаются без изменений. Отводы же подвергаются операции XOR с выходным разрядом перед сдвигом в следующую позицию. В рассматриваемом проекте мы реализовали генератор Галуа, а генератор Фибоначчи применяется в проекте 4 (далее в этой же главе).

Спецификация проекта

Цель — разработать питающуюся от батареи светодиодную свечу, которая как можно более точно имитирует настоящую. Интенсивность свечения светодиода может меняться при помощи псевдослучайных чисел, реализованных посредством LFSR-генератора. Разрядность LFSR будет определять длительность интервала, после которого картина освещения начнет повторяться. Генератор псевдослучайных чисел и управление интенсивностью свечения светодиода должны быть реализованы при помощи одного из микроконтроллеров семейства tinyAVR (с самым маленьким числом выводов). Окончательная блок-схема устройства приведена на рис. 2.16.

Генератор псевдослучайных чисел реализован при помощи микроконтроллера tinyAVR, а контакты портов микроконтроллера осуществляют управление интенсивностью свечения. Чтобы цвет был похож на свечу, нужно подобрать белый (или теплый белый) светодиод. Однако это означает, что потребуется напряжение питания 5 В (или больше). Микроконтроллеры AVR работают на напряжении 5,5 В, которое легко получить либо при помощи четырех батареек напряжением в 1,5 В размера AA (например, щелочных), либо при помощи никель-металлогидридных аккумуляторов (такого же размера) напряжением 1,2 В.

Плата, светодиод и батареи должны быть размещены в корпусе таким образом, чтобы конструкция напоминала свечу. Посмотрим, как исходя их этих требований, реализовать данный проект.

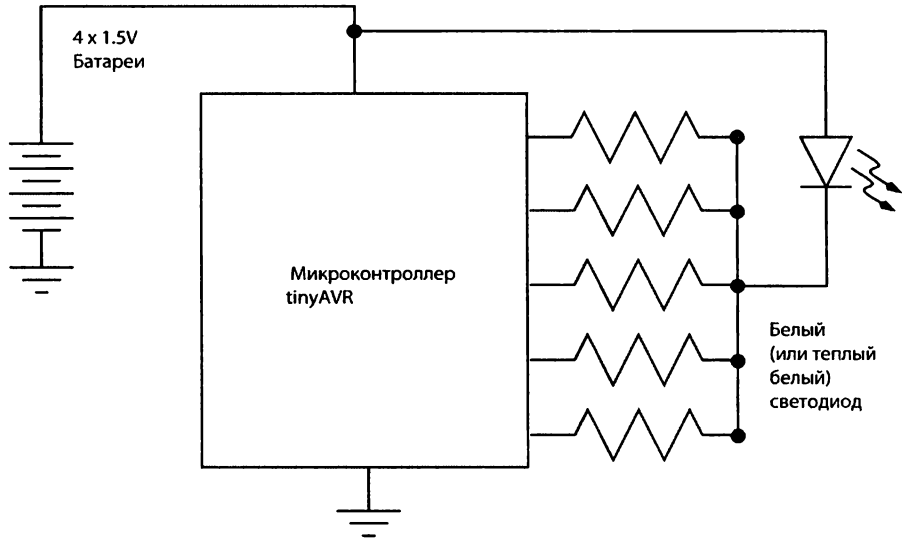


Рис. 2.16. Окончательная блок-схема мерцающей свечи

Описание устройства

На рис. 2.17 приведена принципиальная схема блока управления мерцающей светодиодной свечи на основе контроллера tinyAVR. Мы применили микроконтроллер Tiny13, который имеет восемь выводов. На рис. 2.18 показана схема подключения светодиода. Устройство содержит две платы: плату контроллера и плату светодиода. Идея состоит в том, чтобы закрепить плату со светодиодом над платой контроллера, чтобы уменьшить размер конструкции. Таким образом, она будет занимать меньше места и может быть упакована в трубку, напоминающую свечу.

На рис. 2.17 изображены разъемы SL1, SL2 и SL3, предназначенные для подключения платы светодиода. Светодиод подключен к источнику питания и размещен на второй плате (как показано на схеме рис. 2.18). Пять контактов ввода/вывода контроллера объединены, светодиод подключен к ним через последовательное сопротивление 100 Ом. Напряжение включения белого светодиода составляет 3,5 В, так что при напряжении питания в 5,5 В каждый контакт будет давать ток примерно 20 мА (с таким током контакт AVR справится без труда). Поскольку объединено пять выходов, то максимальный ток через светодиод составит 100 мА. Мы применили "теплый" белый светодиод высокой яркости с мощностью 1 Вт и максимальным током 300 мА. Плата контроллера имеет также разъем ISP для программирования микроконтроллера tinyAVR и разъем для подключения батарей. Выключатель SW1 коммутирует питание схемы.

На рис. 2.18 показана схема подключения светодиода. На плате светодиода имеется три разъема, каждому из которых соответствует разъем на плате контроллера. LED1 — это белый светодиод большой мощности (1 Вт).

При сборке устройства убедитесь в правильной стыковке разъемов SL 1, 2, 3 на платах контроллера и светодиода.

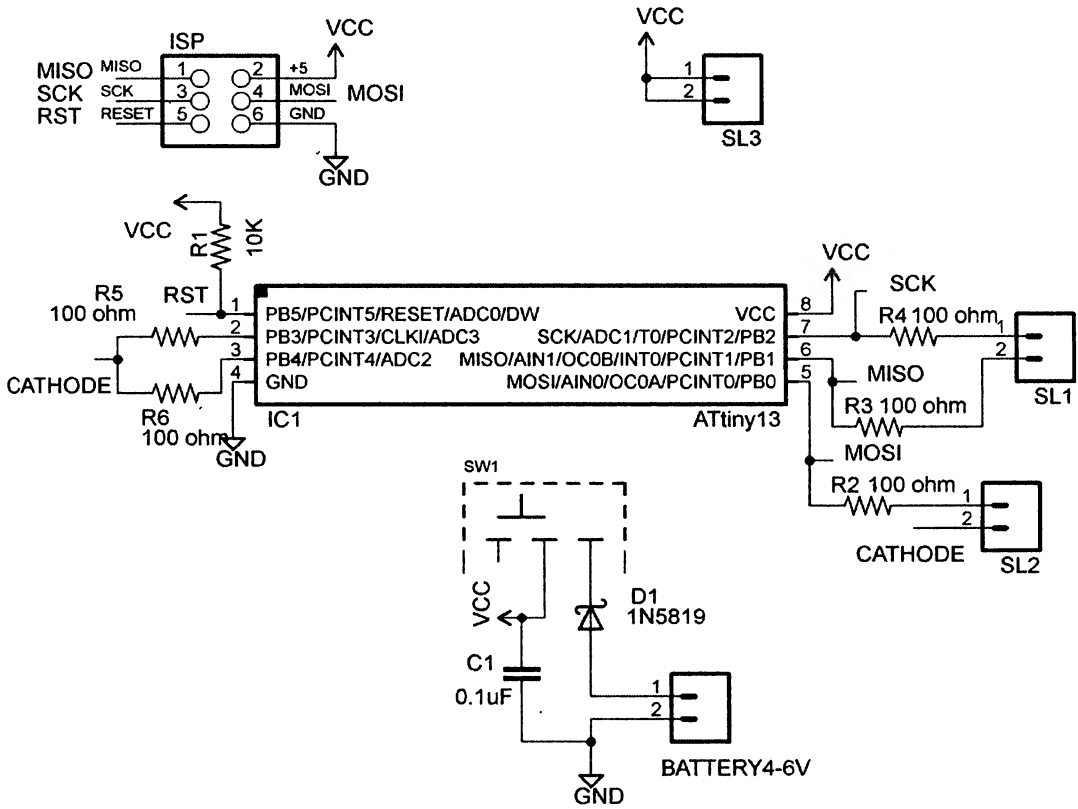


Рис. 2.17. Принципиальная схема блока управления светодиодной свечи

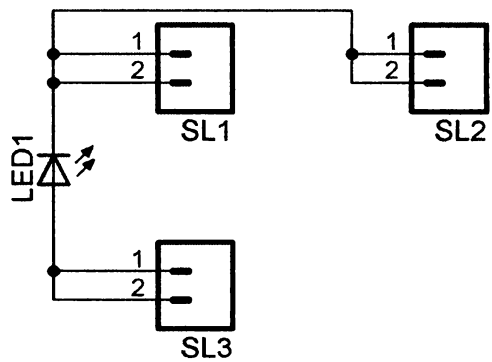


Рис. 2.18. Схема подключения светодиода

Конструкция

Компоновку обеих плат в программе EAGLE и принципиальные схемы можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Плата контроллера в основном разведена на стороне пайки (на стороне компонентов есть всего несколько перемычек). Плата светодиода, наоборот, разведена на стороне компонентов потому, что разъемы должны подключаться с другой стороны (чтобы должным образом совпасть с платой контроллера). На рис. 2.19–2.23 показаны фотографии устройства на разных стадиях. Обе схемы собраны на односторонних печатных платах. Микроконтроллер, резисторы и конденсатор выполнены в корпусах SMD.

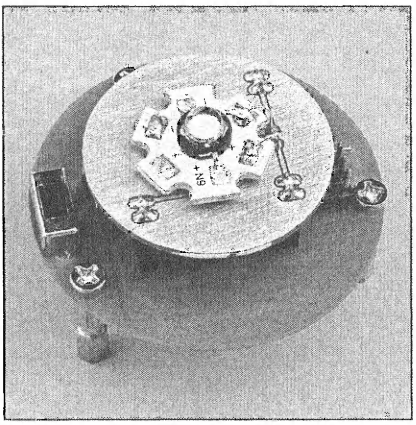


Рис. 2.19. Плата светодиода, смонтированная на плате контроллера

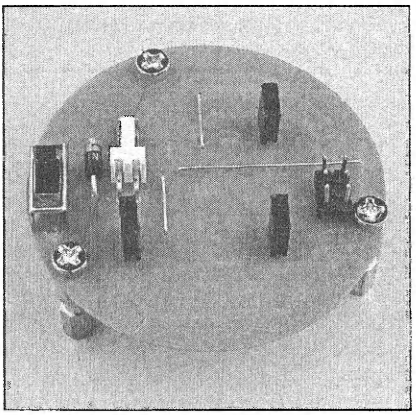


Рис. 2.20. Плата контроллера светодиода. Контроллер Tiny13 припаян с обратной стороны. Обратите внимание на три перемычки из провода

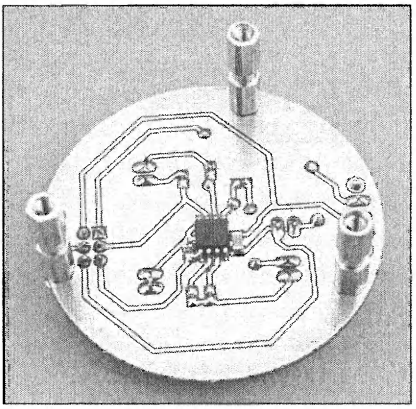


Рис. 2.21. Контроллер светодиода (сторона печатных проводников)

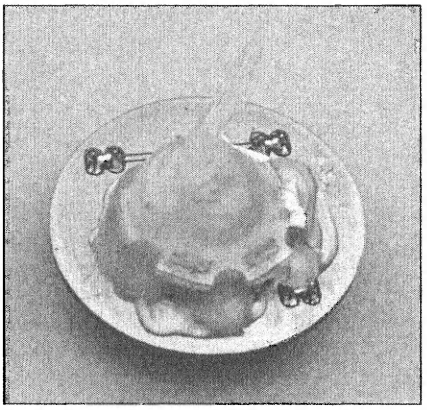


Рис. 2.22. Плата с белым светодиодом мощностью 1 Вт, покрытым термоклеем в форме пламени свечи

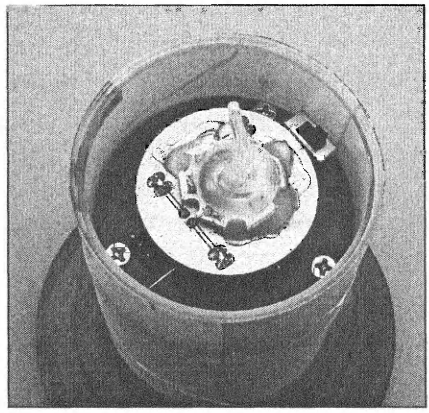


Рис. 2.23. Конструкция мерцающей свечи, смонтированная внутри трубки из оргстекла. Отсек для четырех щелочных батарей размера AA находится под платой контроллера светодиода

На рис. 2.22 показана фотография платы светодиода. Мощные светодиоды обычно снабжены радиатором. К выводам светодиода были припаяны провода, подключенные к печатной плате. После монтажа светодиода на него было нанесено изрядное количество прозрачного термоклея. После остывания термоклея ему была осторожно придана форма пламени свечи. На нашем сайте есть видеозапись мерцающей светодиодной свечи.

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 1,2 МГц. Контроллер запрограммирован при помощи STK500 в режиме программирования ISP. Во время программирования тактовая частота устанавливается в 1,2 МГц (выбирается частота генератора 9,6 МГц и программируется fuse-бит CKDIV8, для деления ее на 8). Управляющее программное обеспечение для мерцающей свечи очень простое. Генератор случайных чисел — это 32-разрядный генератор Галуа (на базе LFSR с отводами 32, 31, 29 и 1 (если разряды нумеровать справа)). В соответствии с генерируемыми случайными значениями включаются случайные выходы, к которым подключен светодиод. Между обновлениями делается случайная задержка. Длительность задержки также определяется по значению LFSR. Начальное значение LFSR равно единице. Полный исходный код приведен в листинге 2.1.

Листинг 2.1

```
#include<avr/io.h>
#define F_CPU 1200000UL
#include<util/delay.h>
```

```
int main(void)
{
 unsigned long lfsr = 1;
 unsigned char temp;
 DDRB= 0xff;
 while(1)
 {
  lfsr = (lfsr >> 1) ^ (-(lfsr & 1u) & 0xd0000001u);
   /* отводы 32 31 29 1 */
  temp = (unsigned char) lfsr;
     //берем младшие восемь битов
  DDRB = ~temp; //Declare those pins as
     //выдаем сигнал там, где temp равняется нулю
  PORTB = temp; //Присваиваем значение 0
     //тем контактам, которые объявлены как выходные
  temp = (unsigned char) (lfsr >> 24);
  _delay_loop_2(temp<<7);
 }
}
```

Переменная `lfsr` реализует генератор LFSR. Переменная `temp` получает младшие восемь битов LFSR и включает случайное количество выходов, дающих ток. Затем в нее записываются старшие восемь битов для формирования случайной задержки между обновлениями.

Проект 3. Смешивание цветов светодиода RGB

Известно, что все видимые цвета можно получить из трех основных цветов: красного, синего и зеленого. Рассматриваемый проект иллюстрирует, как можно смешивать первичные цвета в разных пропорциях и получать миллионы оттенков. Некоторые из вас, наверное, уже проверяли эту гипотезу, создавая цвета в программах для дизайна вроде Microsoft Paint, Adobe Photoshop и т. п. Дисплеи персональных компьютеров, ноутбуков и нетбуков характеризуются количеством отображаемых ими цветов. Простые дисплеи поддерживают 15-разрядное представление цвета (по 5 битов на каждый из основных цветов), они могут выдавать 2^5 комбинаций, что дает $2^5 \times 2^5 \times 2^5$ цветов. Хорошие дисплеи поддерживают 24-разрядную палитру цвет (и даже больше). В данном проекте мы продемонстрируем концепцию смешивания цветов на одном светодиоде типа RGB. Программное обеспечение генерирует 8 разрядов каждого цвета. Поэтому мы можем получить на одном светодиоде $2^8 \times 2^8 \times 2^8$ цветов, однако такое количество оттенков глаз человека не различает.

Спецификация проекта

Цель — разработать схему на основе светодиода типа RGB, которая позволит смешивать разные процентные соотношения красного, зеленого и синего цветов (чтобы получить множество оттенков). Пользователь должен иметь возможность задавать соотношение цветов. Для управления интенсивностью свечения каждого светодиода (т. е. соотношением цветов) используется широтно-импульсная модуляция (которая уже обсуждалась ранее). Блок-схема проекта приведена на рис. 2.24.

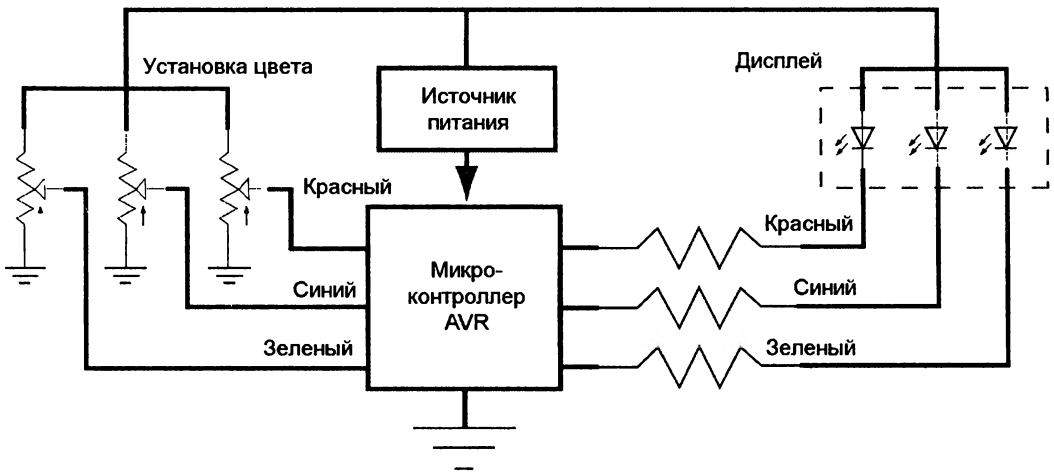


Рис. 2.24. Блок-схема устройства смешивания цветов RGB-светодиода

Описание устройства

На рис. 2.25 изображена принципиальная схема проекта со смешиванием цветов RGB. Входное напряжение может составлять от 5,5 до 20 В (стабилизатор LM2940 выдает постоянное выходное напряжение 5 В). Диод D1 — это диод Шоттки (1N5819) с падением напряжения примерно 0,2 В. Он защищает схему в случае подачи входного напряжения с обратной полярности. Конденсаторы C2 и C8 предназначены для фильтрации выбросов и нежелательных помех источника питания; C4 и C7 — для стабилизации выхода LM2940. C3 припаивается возле контактов питания микроконтроллера для развязки схемы. Основа устройства — микроконтроллер ATtiny13. Он имеет все необходимые нам элементы (таймеры, АЦП, контакты ввода/вывода и т. д.). Рассматриваемый далее код достаточно мал и легко помещается во Flash-память этого контроллера (размером 1 кБайт). Здесь используется RGB-светодиод с общим анодом, подключенным к источнику питания. Резисторы R1, R2 и R3 по 100 Ом каждый ограничивают ток красного, синего и зеленого светодиодов соответственно. SL2, SL3 и SL4 — это три потенциометра для установки интенсивности каждого цвета. Конденсаторы C1, C5 и C6 служат для фильтрации помех на выходах потенциометров. Выходы потенциометров поступают на входы АЦП.

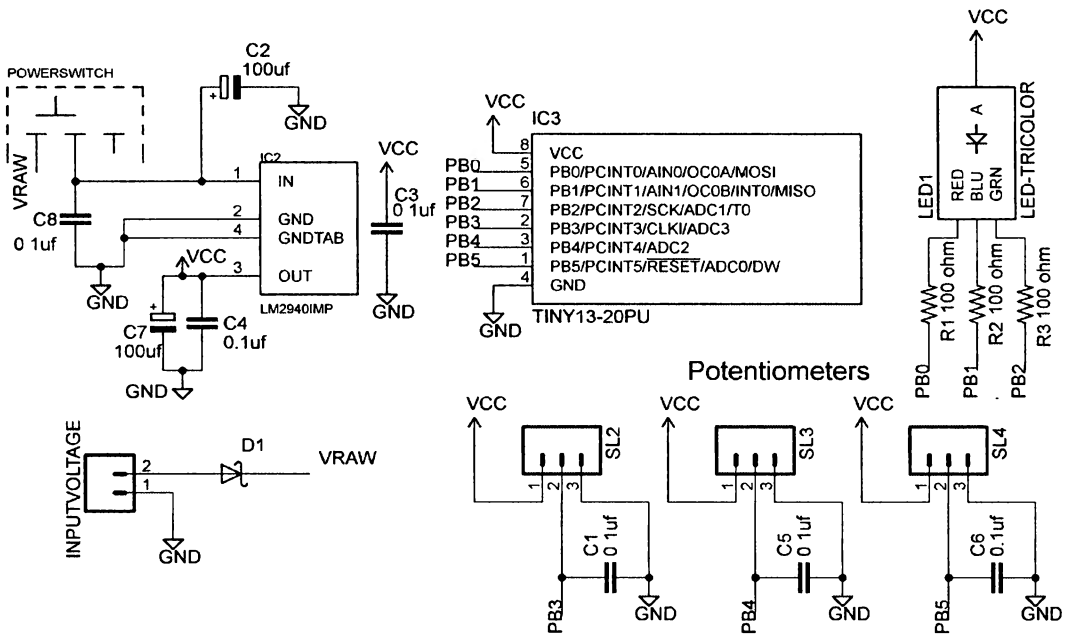


Рис. 2.25. Принципиальная схема устройства смешивания цветов RGB-светодиода

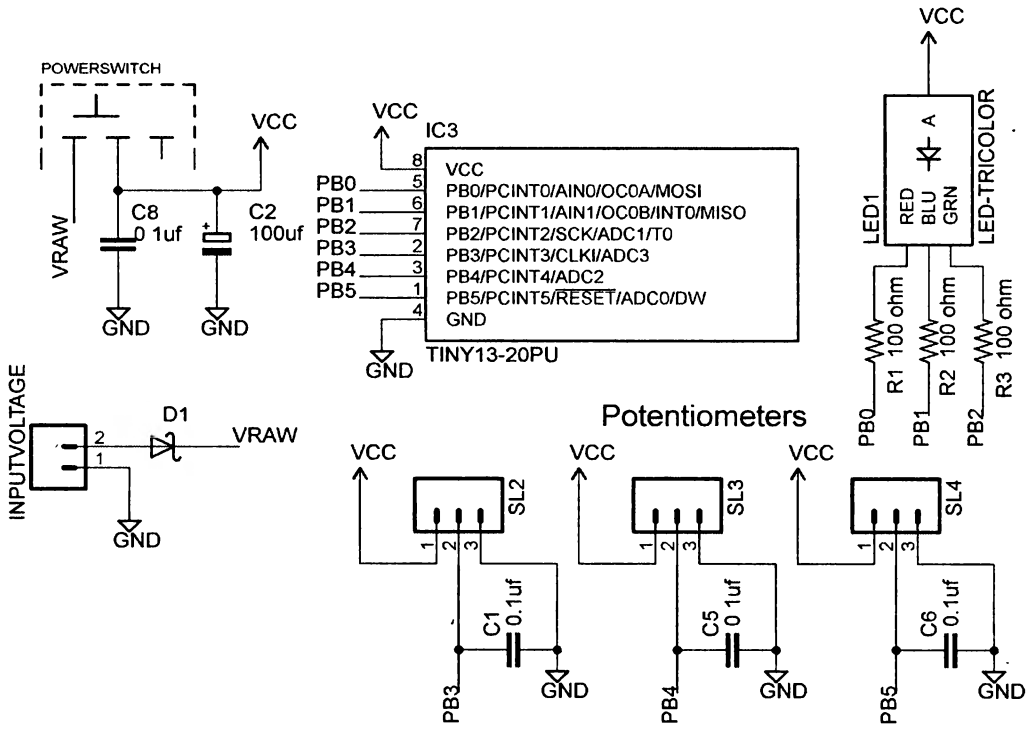


Рис. 2.26. Принципиальная схема устройства смешивания цветов RGB-светодиода без стабилизатора питания

Схему можно спроектировать и без стабилизатора напряжения (рис. 2.26), но тогда входное напряжение должно быть от 4,5 до 6 В. Здесь отсутствует микросхема LM2940 и конденсаторы на ее выходах.

В процессе выполнения программы сопротивления потенциометров считываются тремя АЦП контроллера ATtiny13 и соответствующая интенсивность свечения светодиодов устанавливается при помощи широтно-импульсной модуляции. Контроллер ATtiny13 имеет только два аппаратных ШИМ-канала, а нам для управления тремя светодиодами нужно три, поэтому реализована программная широтно-импульсная модуляция. Контроллеры tinyAVR имеют 10-разрядные АЦП, но в этом проекте задействовано только восемь разрядов. Каждый канал АЦП преобразует аналоговый сигнал от потенциометра в цифровое значение в диапазоне от 0 до 255 (8-битовое разрешение), которому может быть напрямую сопоставлен уровень яркости соответствующего светодиода.

Конструкция

Компоновки печатных плат и принципиальные схемы (обеих версий) можно скачать с адреса: **www.avrgenius.com/tinyavr1**.

Обе платы в основном разведены на стороне пайки (на стороне компонентов есть всего несколько перемычек). Нарисованный на платах круг диаметром 40 мм предназначен для резервирования места под шарик для пинг-понга, которым закрыт светодиод (для рассеивания и смешивания цветов). На рис. 2.27 и 2.28 изображены обе стороны платы (вариант со стабилизатором). На рис. 2.29 показана сторона компонентов (со светодиодом, закрытым шариком для пинг-понга).

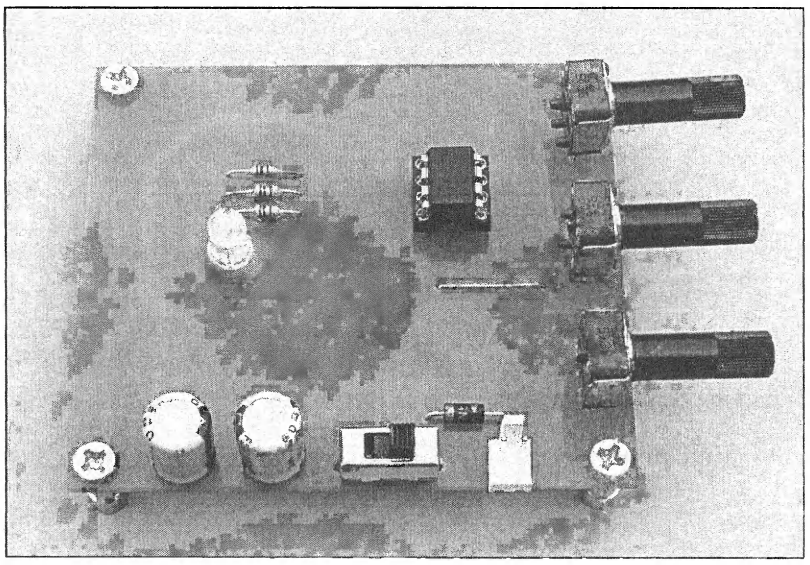


Рис. 2.27. Плата устройства смешивания цветов (сторона компонентов)

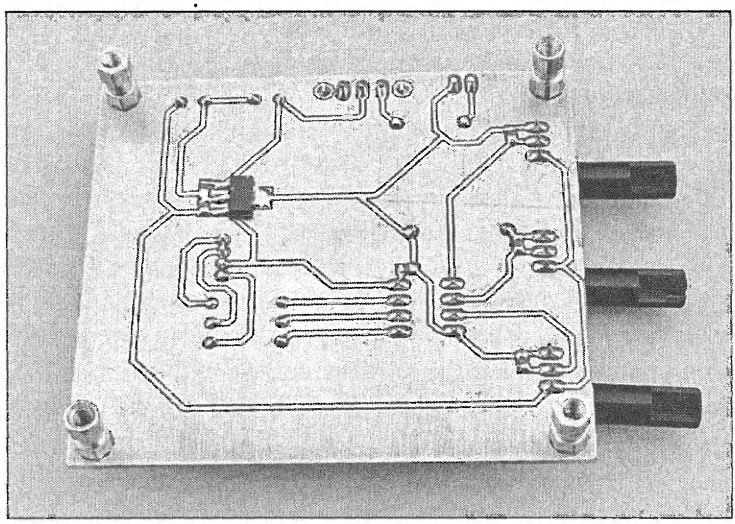


Рис. 2.28. Плата со стороны печатных проводников

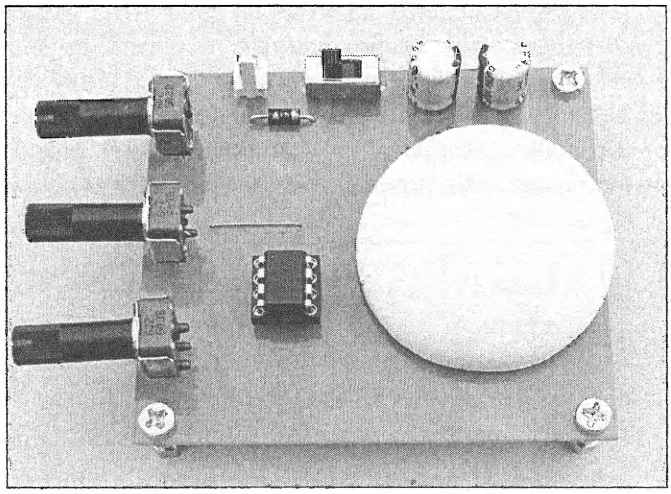


Рис. 2.29. Печатная плата устройства с установленным светорассеивателем

Программирование

Откомпилированный исходный код можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 9,6 МГц. Контакт RESET контроллера ATtiny13 задействован как вход АЦП, поэтому функцию сброса этого контакта следует отключить посредством программирования fuse-бита RSTDISBL. После установки этого бита интерфейс ISP становится недоступен и дальнейшее программирование контролле-

ра следует выполнять при помощи другого интерфейса. Поэтому контроллер программируется при помощи STK500 в режиме программирования HVSP. Поясним самые важные фрагменты кода.

Листинг 2.2

```
//Процедура переполнения для таймера Timer 0
ISR(TIM0_OVF_vect)
{
 //Значение e определяет количество уровней широтно-импульсной модуляции.
 //256 уровней - от 0 to 255
 //0 - полностью включен, а 255 полностью выключен
 if(e==255)
 {
  e=0;
  PORTB |= (1<<0)|(1<<1)|(1<<2);
 }
 abc(pwm[0],pwm[1],pwm[2],e);
 e++;
}
```

Листинг 2.2 — это процедура прерывания по переполнению таймера Timer0. Она обрабатывает три программных канала ШИМ и вызывается при переполнении Timer0. Переменная `e` хранит состояние ШИМ. Когда `e` получает значение 255, все светодиоды выключаются и `e` инициализируется значением 0. Массив `pwm` содержит текущее значение интенсивности каждого светодиода. Код написан таким образом, что значение 0 в любой переменной массива `pwm` соответствует максимальной интенсивности (светодиод полностью включен), а 255 — минимальной (светодиод полностью выключен). Функция `abc` сравнивает каждое значение массива `pwm` с переменной `e` и, если имеется совпадение, включает соответствующий светодиод.

Листинг 2.3

```
//Эта функция читает значение АЦП из выбранного канала
unsigned char read_ADC(unsigned char channel)
{
 unsigned char k;
 unsigned int ADCvalue=0;
 ADMUX = ADMUX&(0b11111100);
   //сбросить биты выбора канала
 ADMUX |= channel;
   //игнорируем первое считывание после смены канала
```

```
 ADCSRA |= 1<<ADSC;
 while(ADCSRA&(1<<ADSC));//Ждем
 ADCvalue=ADCH;
 ADCvalue=0;//игнорируем считывание
 for(k=0;k<=7;k++)
 {
  ADCSRA |= 1<<ADSC;
  while(ADCSRA&(1<<ADSC));//Ждем
  ADCvalue += ADCH;
 }
 return (ADCvalue>>3); //делим на 8
}
```

Процедура из листинга 2.3 преобразует значение из выбранного канала АЦП (номер которого передается в эту функцию через аргумент `channel`) в цифровое значение от 0 до 255 и возвращает его вызывающей функции. Функция сначала выбирает нужный канал АЦП. Она игнорирует первое считывание АЦП и возвращает в вызывающую функцию среднее значение последующих восьми считываний. АЦП используется в режиме однократного преобразования. После выбора нового канала АЦП всегда рекомендуется игнорировать первое считывание (во избежание ошибок преобразования).

Функция `main` работает в бесконечном цикле. Она берет (по одному) отсчеты трех каналов АЦП и присваивает их соответствующим переменным массива `pwm`.

Работа устройства

Интенсивность свечения каждого светодиода можно изменять от минимума до максимума (через 256 уровней) при помощи потенциометров. Разные интенсивности красного, зеленого и синего светодиодов дают разные оттенки цветов. Шарик для пинг-понга обеспечивает равномерное смешивание трех цветовых компонентов.

Проект 4. Случайный генератор цвета и звука

Мы успешно использовали LFSR-генератор псевдослучайных чисел устройства Tiny в проекте 2, но начальное значение LFSR было фиксированным. Это означает, что при каждом включении схемы она генерирует одну и ту же последовательность. Теперь покажем, как можно встроить 16-разрядный LFSR в такое небольшое устройство, как ATtiny13 (при этом канал АЦП будет задавать начальное значение). Это дает разные начальные значения для LFSR при каждом включении цепи, поэтому последовательности окажутся "более случайными". 16-разрядный LFSR может успешно генерировать случайные числа (за исключением 0) с периодичностью 65 535, однако для этого отводы должны быть в определенных позициях. N-разрядный регистр с периодом 2^n–1 называется максимальным LFSR, именно он применен в этом проекте. Рандомизация выхода LFSR хорошо заметна по разным цветам RGB-светодиода и звукам из динамика.

Спецификация проекта

Цель — реализовать LFSR на устройстве Tiny и продемонстрировать его случайную работу с помощью RGB-светодиода и динамика. Схема должна работать при напряжении в 3 В. Управление интенсивностью свечения светодиодов (для генерирования случайных цветов) осуществляется опять-таки при помощи широтно-импульсной модуляции (как в проекте 3). Случайный звук генерируется подачей в динамик сигнала звукового диапазона. Блок-схема устройства приведена на рис. 2.30.

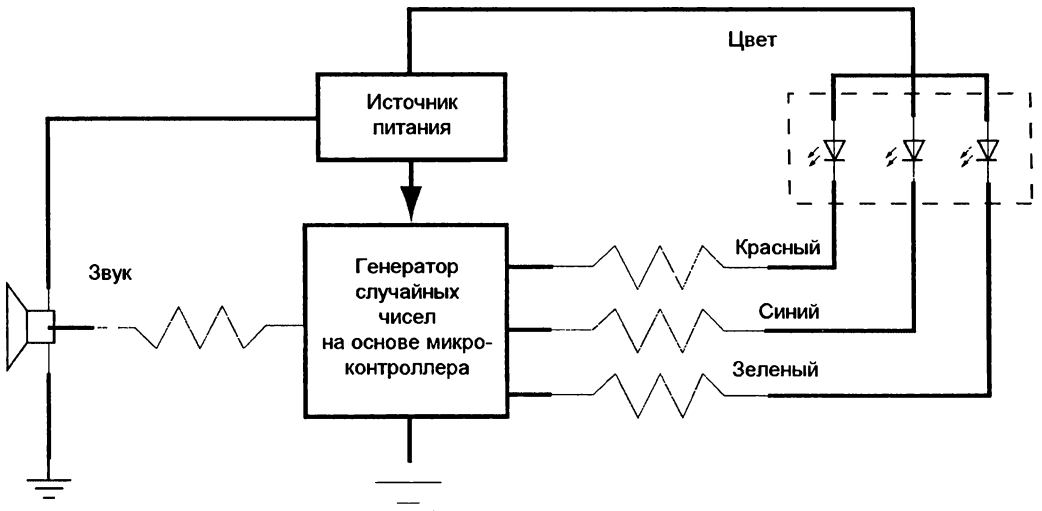


Рис. 2.30. Блок-схема генератора случайных цветов и звуков

Описание устройства

На рис. 2.31 показана принципиальная схема устройства. MAX756 — это повышающий преобразователь постоянного напряжения, который в данном случае выдает напряжение 5 В от источника питания 3 В. Если входное напряжение превысит 5 В, то выходное напряжение повысится в соответствии со входным. Это может привести к повреждению других частей схемы (включая и сам MAX756), поэтому нужно, чтобы входное напряжение никогда не превышало 5 В. Диод D1 и катушка L1 требуются для работы MAX756. Контроллер ATtiny13 имеет все функции, необходимые для нашего проекта. Используемый светодиод — это RGB-светодиод с общим анодом (в корпусе SMD), подключенным к источнику питания. R5, R6 и R7 работают как ограничивающие ток резисторы для красного, синего и зеленого светодиодов (соответственно) и имеют сопротивление по 100 Ом каждый. Элемент T1 (2SD789) — это *n-p-n*-транзистор, который дает сигнал на динамик. Он необходим, потому что контакты ввода/вывода устройства AVR могут дать только 40 мА, а этого для динамика недостаточно. Динамик, использованный нами, имеет мощность 0,5 Вт и сопротивление 8 Ом. Чтобы рассеиваемая динамиком мощность

находилась в допустимых пределах, последовательно с ним нужно включить резистор 10 Ом. Вывод PB4 контроллера работает как "плавающий" канал АЦП, чтобы получить начальное значение для LFSR.

Псевдослучайные числа формируются при помощи 16-разрядного LFSR-генератора Фибоначчи. Эти числа служат для изменения цвета RGB-светодиода и тона в динамике каждые полсекунды. Цвета создаются при помощи программно генерируемого ШИМ-сигнала с 10 уровнями (в отличие от 256 уровней, использованных в проекте 2). Это означает, что светодиод сможет выдавать максимум 10×10×10 цветов (но отображаться будут только 16 из них). Тон в динамике генерируется при помощи меандра переменной частоты. Было выбрано девять разных частот звука.

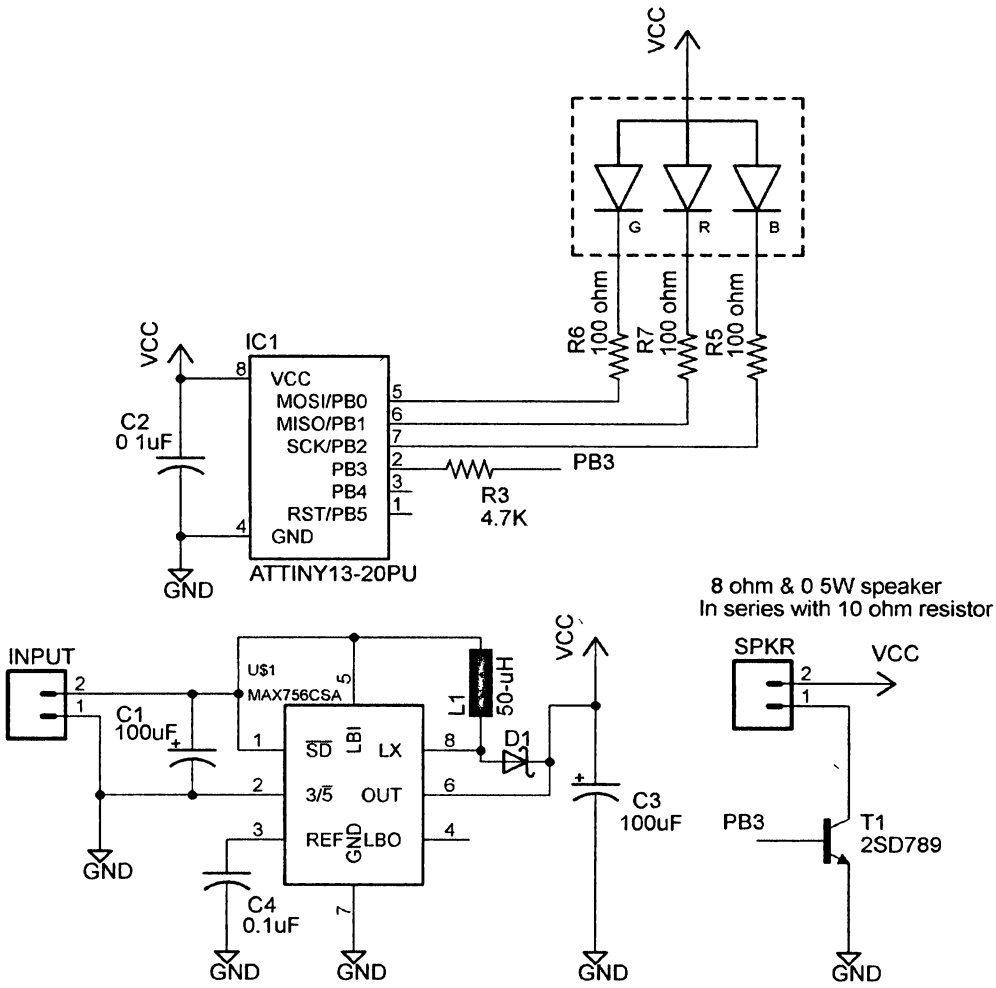


Рис. 2.31. Принципиальная схема генератора случайных цветов и звуков (динамик 8О м/0,5 Вт включен последовательно с резистором 10 Ом)

Конструкция

Компоновку платы (и принципиальную схему) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Печатная плата односторонняя (на стороне компонентов есть всего несколько перемычек). Размещение компонентов показано на рис. 2.32, а сторона пайки — на рис. 2.33. На распаянной плате RGB-светодиод смонтирован на стороне компонентов.

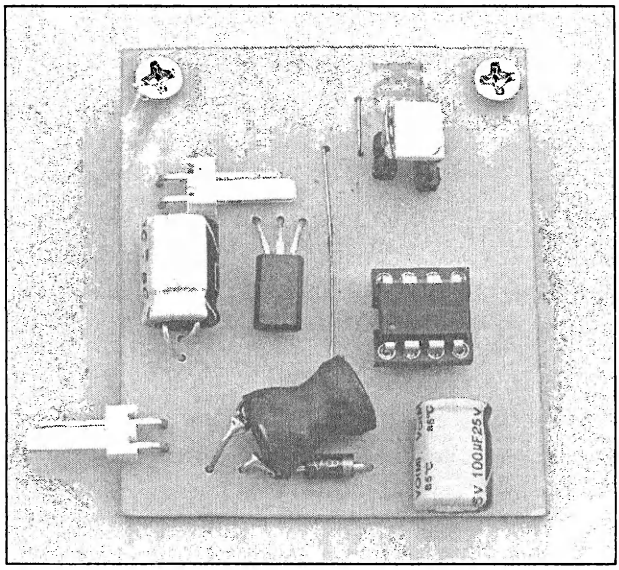


Рис. 2.32. Плата генератора случайных цветов и звуков (сторона компонентов)

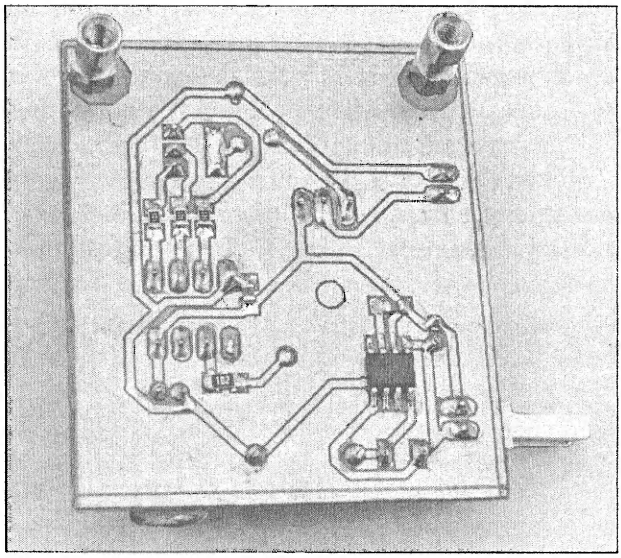


Рис. 2.33. Плата генератора случайных цветов и звуков (сторона печатных проводников)

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 9,6 МГц. Контроллер запрограммирован при помощи STK500 в режиме программирования ISP. Поясним самые важные фрагменты кода.

Листинг 2.4

```
while(1)
{
 //Ожидание начального значения от АЦП
 if(i==4)
 {
  //Это программный код для LFSR
  bit = (reg & 0x0001) ^((reg & 0x0004) >> 2)
       ^((reg & 0x0008) >> 3)^((reg & 0x0020) >> 5);
  reg = (reg >> 1) | (bit << 15);
 //Генерирующий звук код
  PORTB |= 1<<3;
  delay(t);
  PORTB &= ~(1<<3);
  delay(t);
 }
}
```

Листинг 2.4 — это основной бесконечный цикл программы. Он начинает выполняться только тогда, когда значение `i` равно 4, т. е. в качестве начального значения берется четвертое значение АЦП. Прерывание АЦП происходит четыре раза и при этом значение `i` каждый раз увеличивается на единицу. Когда значение `i` достигает четырех, значение плавающего канала АЦП присваивается переменной `reg` и прерывание АЦП отключается. Переменная `reg` — это 16-разрядное целое число, которое и реализует LFSR. Отводами для LFSR выбраны биты 16, 14, 13 и 11 (если считать от самого младшего бита). Код, генерирующий звук, просто формирует прямоугольный сигнал выбранной частоты.

Листинг 2.5

```
//Процедура обработки прерывания по переполнению Timer0
ISR(TIM0_OVF_vect)
{
 //for color
 if(e==9)
 {
```

```
 e=0;
 //Начало нового цикла
 PORTB = PORTB|(1<<2)|(1<<1)|(1<<0);
}
abc();
//Время изменения - примерно полсекунды
j++;
if(j==128)
{
 a = reg%9;//получить новое значение для звука
 t = pgm_read_word(d+a);
 a = reg%16;// получить новое значение для цвета
 blue = pgm_read_byte(k+a);
 red = pgm_read_byte(l+a);
 green = pgm_read_byte(m+a);
 j=0;
}
  }
```

Листинг 2.5 — это процедура обработки прерывания по переполнению Timer0. Она выполняет две основные функции. Во-первых, она реализует программную широтно-импульсную модуляцию (как в проекте 2), а во-вторых, после каждых 128 переполнений она выбирает новое значение цвета и тона звука (при помощи операции остатка от целочисленного деления переменной `reg`). В массиве `d` хранится девять звуков (в виде временных задержек для прямоугольного сигнала), поэтому взяв `modulo9` от переменной `reg`, мы присвоим переменной `a` случайное значение от 0 до 8. Затем соответствующая переменная массива `d` сохраняется в `t`, которая используется как переменная задержки в главном бесконечном цикле. Точно так же имеется 16 цветов, и операция `modulo16` от переменной `reg` задает переменной `a` случайное значение от 0 до 15. Соответствующие уровни интенсивности каждого цвета (из массивов `k`, `l` и `m`) сохраняются в соответствующих переменных `blue`, `red` и `green`. Timer0 имеет предварительный делитель, чтобы 128 прерываний происходили примерно за 0,5 с. Функции `pgm_read_byte` (для 8-битовых переменных) и `pgm_read_word` (для 16-битовых переменных) выбирают константы из памяти программ. Чтобы сэкономить память данных, нужно хранить неизменяемые значения в памяти программ (указав атрибут `PROGMEM` при их объявлении). Остальные подробности вы можете увидеть в полном исходном коде.

Проект 5. Светодиодное перо

Возможно, вы видели рекламные ролики изготовителей батареек, где в воздухе рисуют фигуры и фиксируют их при помощи фотоаппарата с длительной выдержкой. Это можно сделать и при помощи фонарика — рисовать в воздухе и фотографировать. Вместо фонарика рисовать фигуры можно многоцветным светодиодным

пером. Именно такое устройство описано в этом проекте. Некоторые из нарисованных таким пером фигур показаны на рис. 2.34. Подобные фигуры называют "световыми каракулями".

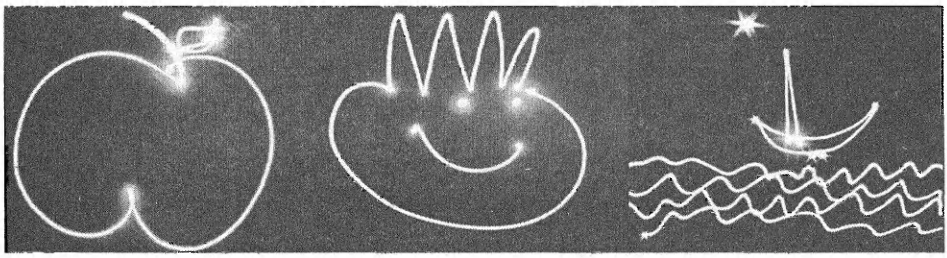


Рис. 2.34. Фотографии нарисованных в воздухе фигур (сделанные цифровым фотоаппаратом на длительной выдержке)

Спецификация проекта

Светодиодное перо должно быть удобно держать в руке, чтобы им можно было без труда рисовать и писать, поэтому размер этого устройства очень важен. Чтобы достичь этой цели, нужно было: применить самые маленькие батареи и предельно упростить схему. Светодиодное перо похоже на генератор случайного цвета и света, однако цель и реализация совершенно иные. В проекте приведены два примера светодиодного пера. Сначала мы реализовали первый вариант, блок-схема которого показана на рис. 2.35.

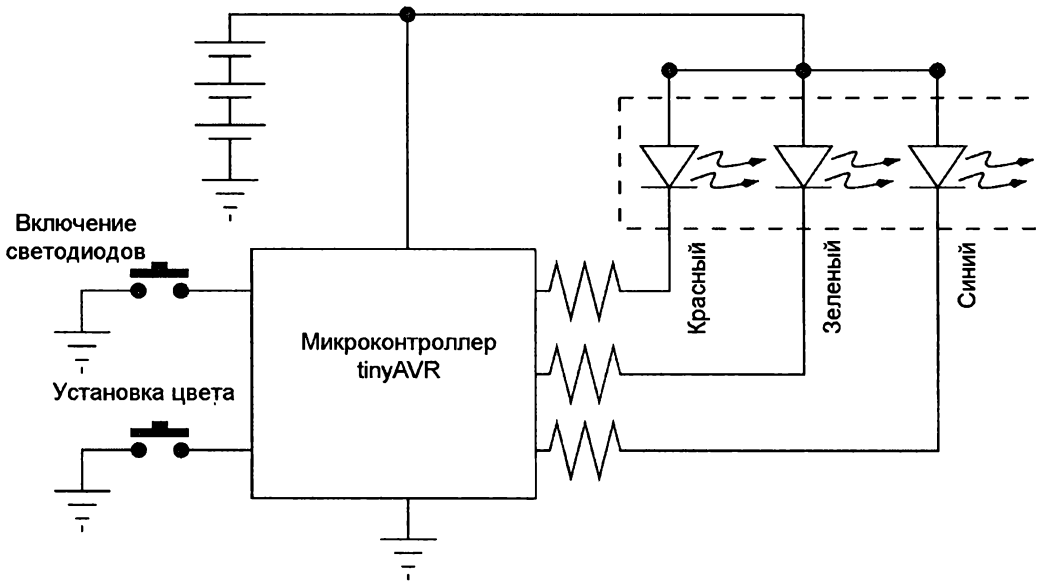


Рис. 2.35. Блок-схема светового пера

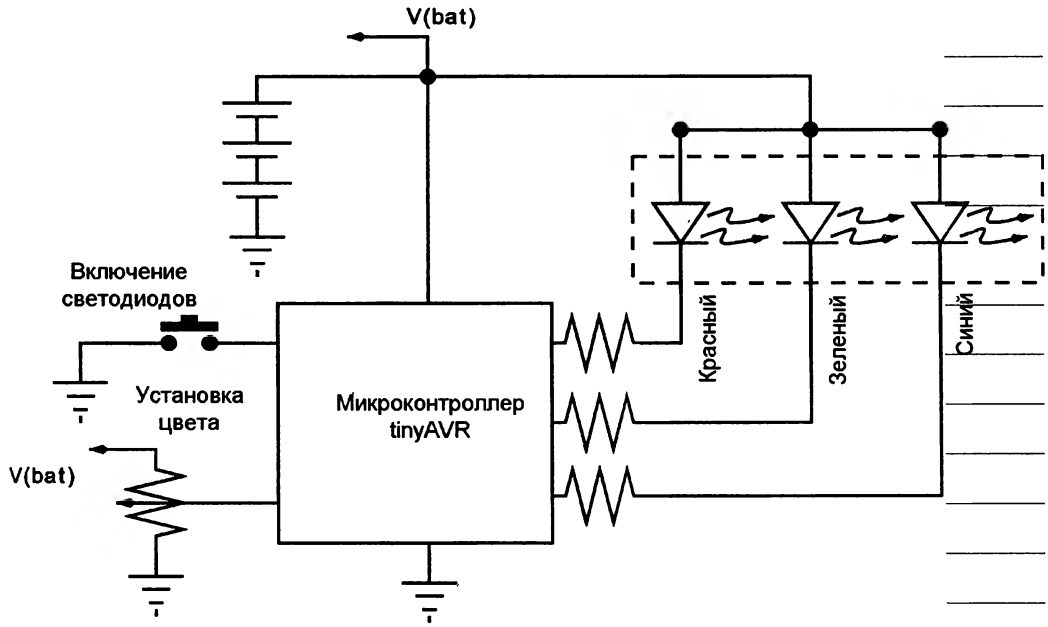


Рис. 2.36. Второй вариант светового пера

После изготовления первого устройства и использования его в течение некоторого времени мы поняли все связанные с ним проблемы и сделали вторую конструкцию (рис. 2.36).

Давайте сначала обсудим требования. Идея изготовить многоцветное светодиодное перо пришла после того, как мы увидели "световые каракули" на одном из Web-сайтов. Эти картинки рисовались при помощи простых светодиодных перьев, изготовленных из одноцветных светодиодов. Мы подумали, что вместо нескольких светодиодных перьев можно применить одно перо с RGB-светодиодом. Интенсивностью свечения светодиодов можно управлять при помощи микроконтроллера и получить при этом гораздо больше цветов, чем при наличии отдельных цветных светодиодов. Чтобы рисовать изображения, вам понадобится цифровой фотоаппарат с ручной установкой выдержки и диафрагмы. Максимальная выдержка фотоаппарата будет определять продолжительность вашего рисования светодиодным пером. Обычные компактные фотоаппараты дают выдержку порядка минуты, чего вполне достаточно для простых рисунков. Для более сложных понадобится зеркальная камера, которая позволяет делать очень длительные выдержки. При рисовании световых каракулей вы, возможно, захотите изменять цвет света. Для выбора цвета можно применить переключатель (который будет менять цвет при нажатии). Кроме того, при рисовании вам может понадобиться выключить светодиод, чтобы начать другую часть рисунка (поэтому нужен выключатель). Все это показано на первой блок-схеме нашего светодиодного пера. Перо имеет два переключателя: для включения светодиодов и для выбора цвета.

Первый вариант пера давал 16 разных цветов. Однако мы столкнулись с такой проблемой: если после выбора цвета вам опять понадобится предыдущий цвет, то придется перебрать все цвета, чтобы вернуться к нему. При этом теряется драгоценное время. Нужен способ быстрого выбора цвета (чтобы не перебирать все возможные цвета по очереди). Поэтому мы заменили кнопку "установка цвета" потенциометром, который можно быстро повернуть для получения нужного цвета. Так получился второй вариант устройства. Остальные функции точно такие же, как в первом варианте. Чтобы добиться минимального размера, мы решили применить для питания схемы батареи-таблетки и 8-контактный микроконтроллер tinyAVR в корпусе DIP. Такой корпус позволяет вынимать микросхему, поэтому на схеме отсутствует разъем для программирования (чтобы уменьшить размер схемы).

Описание устройства

На рис. 2.37 изображена принципиальная схема первого варианта светодиодного пера. Для считывания состояния кнопок S1 и S2 используется микроконтроллер Tiny13, а для отображения разных цветов применяется RGB-светодиод. Схема питается от трех батарей LR44 и не имеет выключателя питания. Устройство запрограммировано так, что при выключении светодиода (кнопкой S2) микроконтроллер входит в состояние пониженного энергопотребления (что обеспечивает экономичность). Обычно микроконтроллер AVR в таком состоянии потребляет ток всего несколько микроампер.

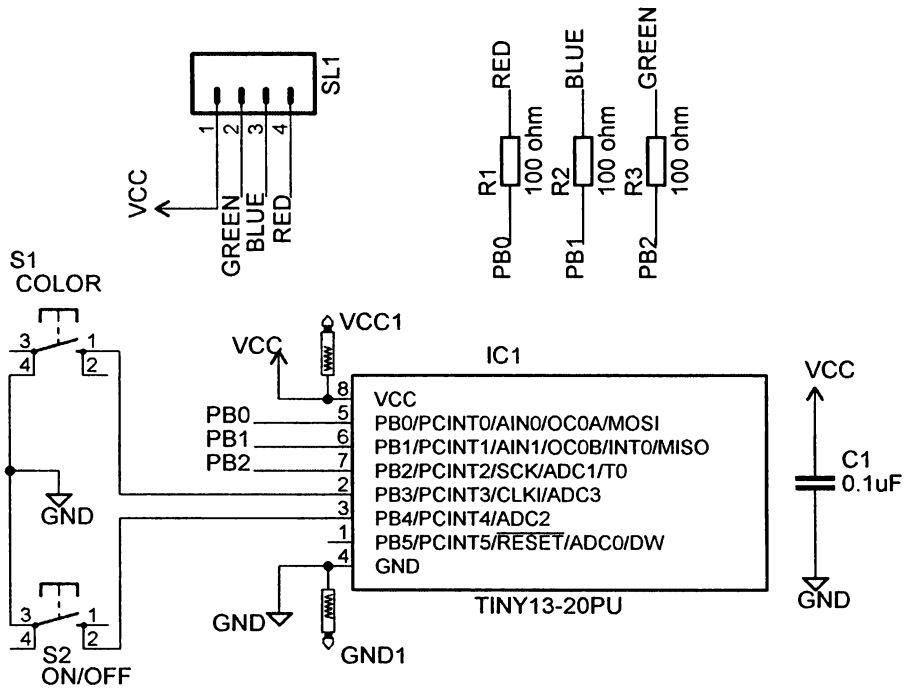


Рис. 2.37. Принципиальная схема светодиодного пера (вариант 1)

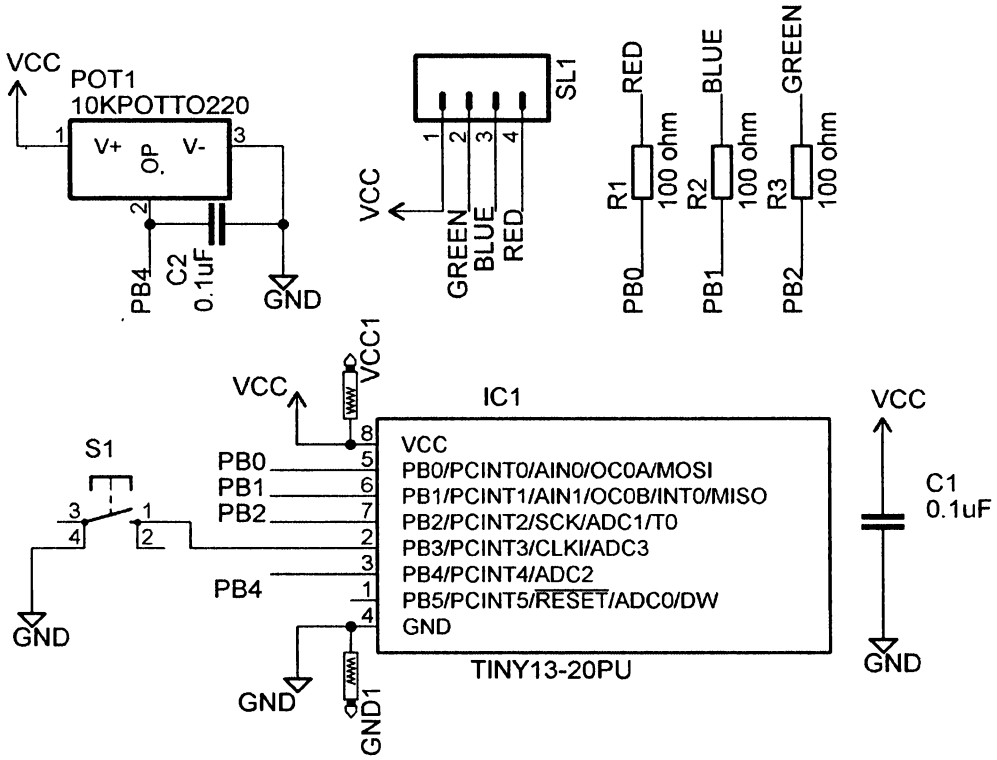


Рис. 2.38. Принципиальная схема светодиодного пера (вариант 2)

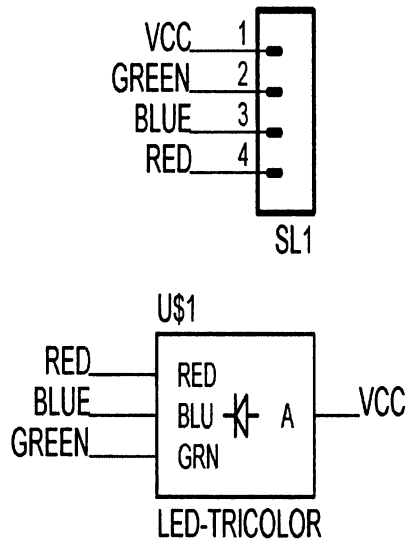


Рис. 2.39. Схема подключения RGB-светодиода

На рис. 2.38 показана принципиальная схема второго варианта светодиодного пера. Здесь кнопка S2 заменена потенциометром POT1. Микроконтроллер AVR13 имеет несколько каналов АЦП. Центральный вывод потенциометра подключен к входному каналу АЦП. Остальные два вывода потенциометра подключены к источнику питания и заземлению. Напряжение с движка потенциометра меняется от нуля до Vcc. По заданному напряжению микроконтроллер выдает соответствующие ШИМ-сигналы для красного, зеленого и синего светодиодов. Общий анод RGB-светодиода (SL1) подключен к источнику питания. В обеих версиях светодиодного пера разъем SL1 — это небольшая плата с RGB-светодиодом (рис. 2.39).

Конструкция

Компоновку платы (и ее схему) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Оба варианта светодиодного пера изготовлены на односторонней печатной плате. Готовая печатная плата достаточно мала для того, чтобы разместиться в трубке из оргстекла диаметром 20 мм. На рис. 2.40 и 2.41 изображены платы контроллера второго варианта светодиодного пера. На рис. 2.42 показана маленькая плата с RGB-светодиодом, смонтированная перпендикулярно главной плате контроллера. Схема питается от трех батареек LR44. На рис. 2.40 видна установка батареек. Батарейки крепятся небольшими магнитами, причем крайние батарейки припаяны к плате. Блок батареек с магнитами склеен резиновым клеем (чтобы они не разъединялись). На рис. 2.43 показана фотография первого варианта светодиодного пера.

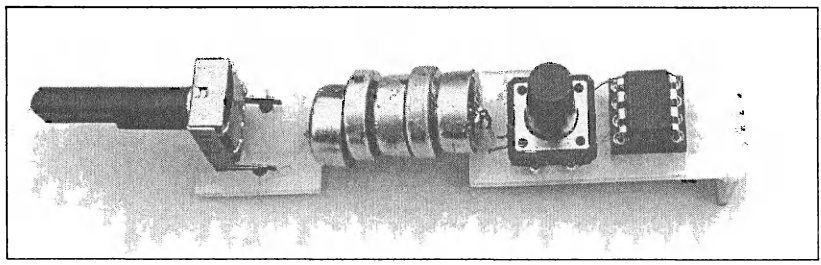


Рис. 2.40. Вид сверху второго варианта светодиодного пера

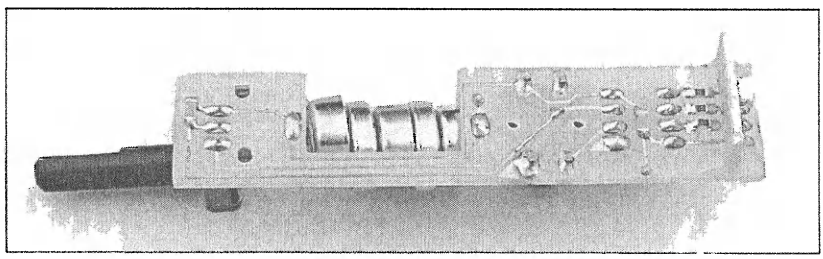


Рис. 2.41. Вид снизу второго варианта светодиодного пера

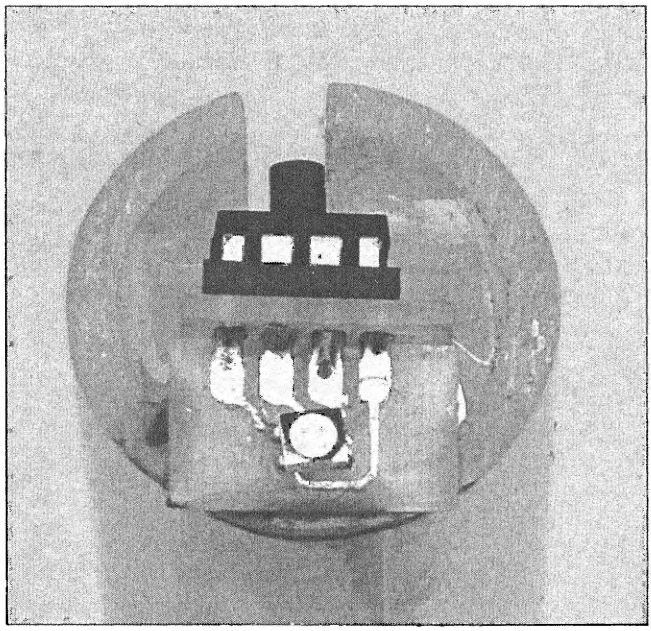


Рис. 2.42. Плата RGB-светодиода, присоединенная к основной плате под прямым углом

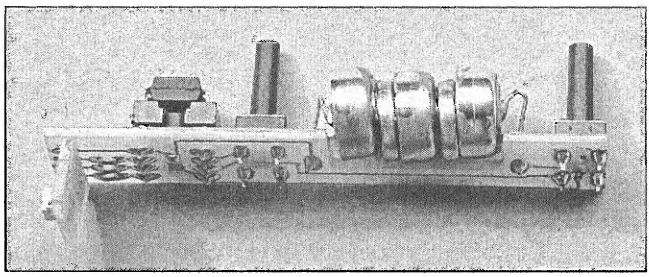


Рис. 2.43. Вид сбоку первого варианта светодиодного пера

Программирование

Откомпилированный исходный код обеих версий можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Оба устройства работают на частоте 1,2 МГц. Коды обеих версий похожи за исключением того, что в первой версии изменение цвета выполняется при помощи кнопки, а во второй версии — при помощи потенциометра. Программирование кода и fuse-битов осуществляется при помощи STK500 в режиме ISP. Разные цвета генерируются посредством программной широтно-импульсной модуляции. Для каждого цвета было выбрано по 9 уровней ШИМ. Из возможных 9×9×9 цветов было выбрано 16 (которые сохранены в памяти программ). Есть еще один

дополнительный режим, который называется "прогоном" — когда все цвета появляются один за другим с промежутком в 500 мс. Рассмотрим самые важные фрагменты кода.

Листинг 2.6

```
ISR(TIM0_OVF_vect)
{
 DDRB &=~(1<<0|1<<1|1<<2);
 if(e==8)
 {
  e=0;
  xyz();
 }
 abc(pwm[0],pwm[1],pwm[2],e);
 DDRB |=(1<<0|1<<1|1<<2);
 e++;
 if(i==15)//режим работы
 {
  counter++;
  if(counter == 390)//500ms
  {
   counter = 0;
   if(k==14)
   k=0;
   else k++;
   pwm[0] = pgm_read_byte(&Blue[k]);
   pwm[1] = pgm_read_byte(&Red[k]);
   pwm[2] = pgm_read_byte(&Green[k]);
  }
 }
}
```

Листинг 2.6 — процедура обработки прерывания по переполнению Timer0, которая управляет широтно-импульсной модуляцией с девятью уровнями (как и в предыдущих проектах). Если же выбирается "режим прогона", то он перебирает все цвета с временным промежутком в 500 мс. Эта процедура в обеих версиях одинакова.

Листинг 2.7

```
if('(PINB&(1<<3)))
{
 _delay_ms(30);
```

```
 while(!(PINB&(1<<3))); //ожидание
 _delay_ms(30);
 TIMSK0 &= ~(1<<TOIE0);
   //Сбросить прерывание таймера
 DDRB &=~(1<<0|1<<1|1<<2);
 GIFR |= 1<<PCIF;
   //Сбросить ожидающее прерывание
 GIMSK |= 1<<PCIE;
   //Активация прерывания по изменению сигнала на контакте
 MCUCR |= (1<<SE|1<<SM1);
   //Установка режима выключения питания
 sleep_cpu();
   //Процессор остановлен до прерывания
}
```

Фрагмент, приведенный в листинге 2.7, работает в бесконечном цикле (когда устройство активно). Он проверяет состояние переключателя на PB3. При нажатии и отпускании он отключает прерывания Timer0 и конфигурирует управляющие светодиодом контакты как плавающие. Это полностью отключает светодиод и включает прерывание по изменению сигнала на контакте. Несмотря на то, что это прерывание может быть выполнено на всех контактах ввода/вывода контроллера ATtiny13, при инициализации кода он настраивается так, чтобы это прерывание мог вызвать только контакт PB3. Затем значение регистра `MCUCR` выбирает режим ожидания и отправляет устройство в режим выключения (при помощи вызова функции `sleep_cpu()`), из которого его можно пробудить, только задав асинхронное прерывание по изменению сигнала на контакте (мы объясним это далее). Во время выключения выполнение кода останавливается. Оно запускается повторно после сигнала пробуждения, которым в данном случае является прерывание по изменению состояния контакта PB3.

Листинг 2.8

```
ISR(PCINT0_vect)
{
 _delay_ms(30);
 while(!(PINB&(1<<3))); //ожидание
 _delay_ms(30);
 MCUCR = 0x00; //отключение режима ожидания
 GIMSK &= ~(1<<PCIE);
   //Отключение прерывания по изменению состояния контакта
 TIMSK0 = 1<<TOIE0;
   //Включение прерывания по переполнению
}
```

Листинг 2.8 — это процедура обработки прерывания для изменения состояния контакта PB3. Исходный код был написан так, чтобы это прерывание было активно только тогда, когда устройство находится в состоянии выключения. Эта процедура отключает режим ожидания и прерывание по изменению состояния контакта. Она также включает снова таймер, чтобы начать генерирование цветов. Подобный метод перевода контроллера в режим выключения реализован в обеих версиях.

Цвета выбираются в бесконечном цикле (в первом устройстве — при помощи кнопки, а во втором — посредством считывания АЦП). Подробности смотрите в исходном коде.

Заключение

В этой главе мы изучили разные типы светодиодов, а также последовательное и параллельное управление ими. Мы также подробно обсудили управление интенсивностью свечения светодиодов при помощи широтно-импульсной модуляции. Эти концепции были подкреплены четырьмя проектами. Кроме того, светодиоды можно соединить так, что соотношение числа контактов ввода/вывода к числу светодиодов будет меньше единицы, т. е. число светодиодов может превышать имеющееся количество контактов ввода/вывода. Широтно-импульсная модуляция во всех проектах реализована программным путем. Устройства tinyAVR имеют и аппаратные каналы для широтно-импульсной модуляции (связанные с таймерами). Мы рассмотрим это в следующей главе.

Глава 3

Более сложные проекты со светодиодами

В предыдущей главе мы рассмотрели светодиоды и их использование в простых проектах. Мы привели примеры нескольких проектов с одним-тремя светодиодами и 8-контактными микроконтроллерами tinyAVR. Однако если мы хотим создать устройства с большим количеством светодиодов, то нам будут нужны микроконтроллеры с большим количеством выводов. Даже при наличии большого количества контактов, светодиодов часто оказывается еще больше, поэтому обычные способы подключения одного светодиода к одному выводу микроконтроллера уже недостаточны. В этой главе мы описываем другие проекты со светодиодами, демонстрирующие более сложные способы управления ими. В одних устройствах применены одноцветные светодиоды, в других — RGB-светодиоды. Для того чтобы управлять большим количеством светодиодов при помощи ограниченного набора контактов микроконтроллера, мы реализовали два способа: мультиплексирование и так называемый метод Чарли (разновидность мультиплексирования, весьма популярная в последнее время).

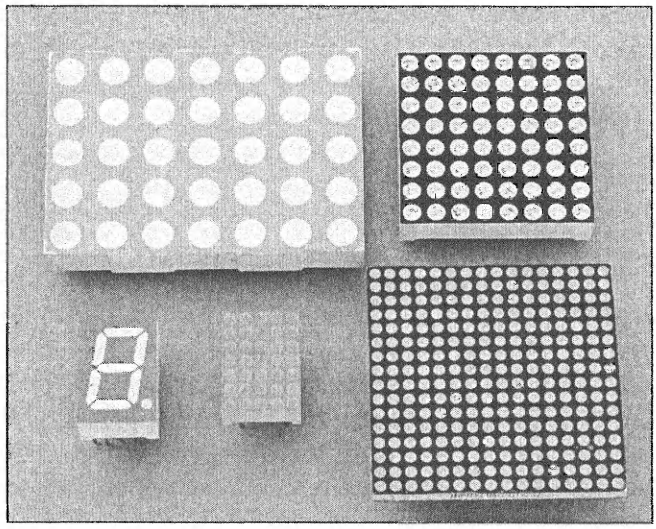


Рис. 3.1. Светодиодные дисплеи

Вам необязательно самому соединять светодиоды. В продаже имеется большое количество объединенных в различные конфигурации светодиодов (различных цветов, размеров и в разных корпусах). На рис. 3.1 показан двухцветный точечно-матричный дисплей 8×8 (в верхнем правом углу), дисплей 16×16 (в нижнем правом углу), семисегментный дисплей (нижний левый угол) и два точечно-матричных дисплея 5×7.

Размер дисплея влияет на номинальный ток светодиода; более крупный дисплей имеет светодиоды, которые выдерживают более высокий ток и поэтому они ярче.

Мультиплексирование светодиодов

Чтобы отображать изображения или текст, светодиодами необходимо управлять. В *главе 2* мы показали, как управлять светодиодами при помощи контактов микроконтроллера. Мы подключали по одному светодиоду к одному выводу. В зависимости от способа подключения светодиода (к источнику питания или к общей шине) светодиод можно было включать при помощи установки на контакте сигнала "1" или "0". Интенсивность света от светодиода регулировалась при помощи ШИМ-сигнала на контакте микроконтроллера. Однако подключение к одному выводу микроконтроллера только одного светодиода слишком расточительно. Вместо этого можно применять мультиплексирование — управление светодиодами при помощи разделения времени. На рис. 3.2 показан пример мультиплексирования: для управления девятью светодиодами используются три строки и три столбца. Каждый столбец и каждая строка подключены к выводу микроконтроллера. Поэтому при помощи шести контактов мы можем управлять девятью светодиодами. Эту схему можно распространить и на большее количество строк и столбцов. Однако для максимального использования контактов рекомендуется по возможности выдерживать одинаковое число строк и столбцов. До какой степени можно развивать этот подход? Можно ли управлять, например, 225 светодиодами при помощи матрицы размером 15×15? Ограничение — максимальный ток светодиодов, которые применяются для такого дисплея.

Давайте сначала рассмотрим работу мультиплексированного дисплея (рис. 3.2). Светодиоды соединены проводами в ряды и столбцы. Каждая строка и каждый столбец подключены к выводам микроконтроллера.

Для включения конкретного светодиода его столбец нужно подключить к источнику питания Vcc, а его строку — к заземлению Gnd. В цепи от Vcc до светодиода должны быть ограничивающие ток резисторы. После того, как столбец подключен к Vcc, светодиоды этого столбца можно включить путем подключения соответствующих строк к общей шине (при помощи микроконтроллера). Например, если подключить столбец 1 к Vcc, а строки 1 и 3 — к обшей шине, то будут включены светодиоды LED1 и LED7. Предположим, что нужно включить LED1, LED2 и LED5. Для этого следует подключить к Vcc столбец 1 (для LED1) и столбец 2 (для LED2 и LED5), а к общей шине — строку 1 (для LED1 и LED2) и строку 2 (для LED5). Однако если на эти строки и столбцы подать указанные напряжения, то загорится также и LED4, поскольку столбец 1 находится под напряжением Vcc, а строка 2 заземлена! Поэтому во избежание включения ненужных светодиодов напряжения на строки и столбцы подаются особым образом.

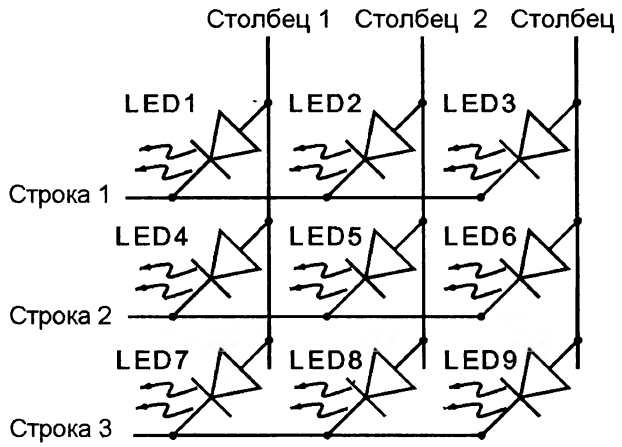


Рис. 3.2. Пример мультиплексирования светодиодов

На время T на столбец 1 подается Vcc, а столбцы 2 и 3 заземляются. В течение этого интервала времени на строку 1 подается заземление (если нужно включить LED1); в противном случае на нее подается Vcc. Если нужно включить LED4, то строка 2 заземляется; в противном случае на нее нужно подать Vcc. И наконец, если нужно включить LED7, то следует заземлить строку 3; в противном случае на строку 3 следует подать Vcc. По истечении первого периода времени T начинается второй интервал времени T, в течение которого столбец 1 заземляется, на столбец 2 подается Vcc, столбец 3 заземляется, а на строки 1, 2 и 3 подается Vcc или земля, в зависимости от того, что нужно сделать со светодиодами LED2, LED5 и LED8 (включить или выключить). Затем начинается третий интервал (опять продолжительностью T) и в этот период столбцы 1 и 2 заземляются, а на столбец 3 подается Vcc. Строки 1, 2 и 3 или заземляются, или на них подается Vcc (в зависимости от того, что нужно сделать с LED3, LED6 и LED9 — включить или отключить). После завершения интервала времени T весь цикл повторяется опять.

Какова продолжительность периода T? Это зависит от числа столбцов. Каждый столбец должен включаться каждые 100 Гц или чаще. В случае трех столбцов $3T$ = 10 мс (10 мс — период 100 Гц). Поэтому T = 3,3 мс.

Ограничивающий ток резистор может быть подключен либо к катоду (как на рис. 3.3), либо к аноду (как на рис. 3.4). Однако это размещение резистора определяет номинал переключателей на высокой (S1, S2 и S3) и низкой (S4, S5 и S6) сторонах. Для переключения Vcc (высокого потенциала) требуется биполярный *p-n-p*-транзистор или *p*-канальный полевой транзистор. Для переключения заземления (низкого потенциала) требуется *n-p-n*-транзистор или *n*-канальный транзистор типа MOSFET.

Давайте рассмотрим пример расчета транзисторных ключей высокого и низкого потенциалов, когда ограничивающие ток резисторы подключены последовательно с катодами светодиодов (см. рис. 3.3).

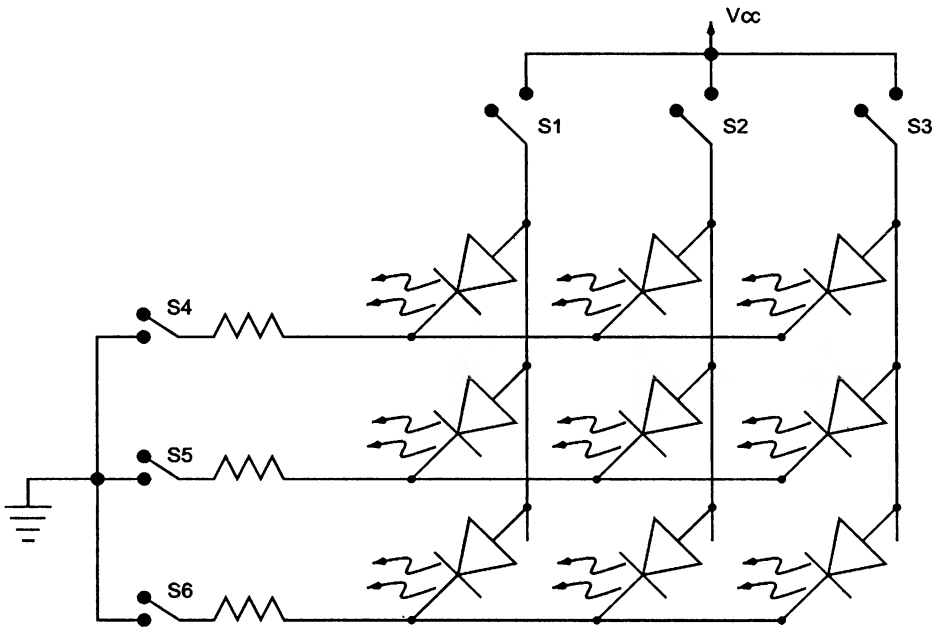


Рис. 3.3. Подключение токоограничительного резистора к катоду

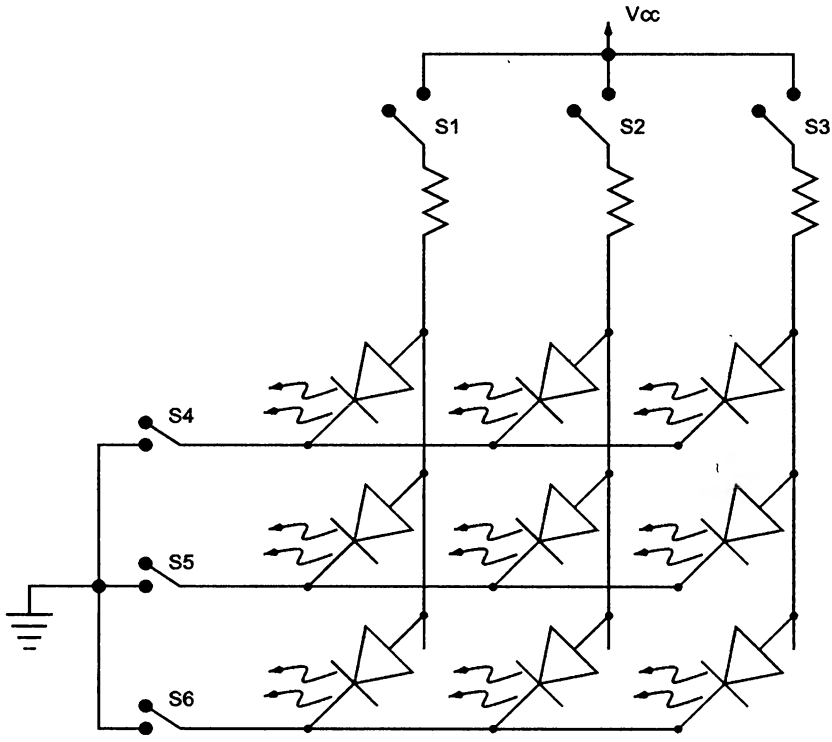


Рис. 3.4. Подключение токоограничительного резистора к аноду

На этой схеме ключ S1 включается на период T, во время которого включаются ключи S2 и S3. Ток через светодиод определяется напряжением питания $U_{ип}$ (Vcc), падением напряжения на светодиоде и последовательно включенным резистором (если падение напряжения на коммутирующих элементах пренебрежимо мало):

$$I_{LED} = (U_{ип} - U_{LED})/R,$$

где R — это последовательно включенное сопротивление. Однако это максимальный ток. Средний ток через светодиод меньше в N раз, где N — число столбцов (в нашем случае N=3). Для создания требуемого тока значение сопротивления нужно уменьшить в эти N раз (увеличив таким образом пиковый ток). Однако пиковый ток не может увеличиваться произвольно, поскольку в какой-то момент он превзойдет максимальный ток светодиода.

Обычно пиковый ток может быть от пяти до десяти раз больше, чем максимальный средний ток.

Поэтому число столбцов может быть не больше десяти. Если вы хотите уменьшить требуемый средний ток до величины меньшей максимального среднего номинала светодиода, то можно увеличить N. Давайте проиллюстрируем это примером. В табл. 2.1 из предыдущей главы приведены данные светодиодов. Красные светодиоды имеют рабочий средний ток 30 мА и максимальный прямой ток 120 мА.

Значение R следует выбирать таким, чтобы пиковый ток через светодиод никогда не превышал 120 мА. Если число столбцов равно десяти, то средний ток (благодаря мультиплексированию) составит 12 мА, что не превышает среднего значения тока. Мы можем увеличить число столбцов до 20 за счет снижения среднего тока (и следовательно, интенсивности свечения светодиодов). Влияние последовательно подключенного к катоду ограничивающего ток резистора (как показано на рис. 3.3) состоит в том, что ключи S4, S5 и S6 будут работать на таком максимальном токе, который равен пиковому току светодиода (в данном случае 120 мА). Ключи S1, S2 и S3 должны быть способны работать на 360 мА каждый. Обычно *n-p-n*-транзистор или *n*-канальный транзистор типа MOSFET могут обеспечить более высокий ток, чем соответствующий *p-n-p*-транзистор или *p*-канальный MOSFET. Вместо последовательно подключенного к катоду ограничивающего ток резистора, мы можем подключить резистор последовательно с анодом (рис. 3.4). Однако для этого потребуется соответствующее изменение в схеме мультиплексирования. Вместо подключения столбца к Vcc нам придется подключать строку к заземлению на период времени T и (в зависимости от того, какой светодиод нам нужно зажечь) подключать соответствующий столбец к Vcc. В следующий период времени T к заземлению подключается следующая строка и т. д. Переключатели столбцов (S1, S2 и S3) должны теперь выдерживать максимальный ток 120 мА, а переключатели строк — 360 мА. Однако ключи общей шины будут реализованы при помощи *n-p-n*-транзисторов (или *n*-канальных MOSFET), которые более доступны, чем аналогичные им *p-n-p* или *p*-канальные MOS-транзисторы.

А что, если число светодиодов гораздо больше, чем можно организовать в одну матрицу из строк и столбцов? Увеличение размера матрицы светодиодов — это проблема (как уже обсуждалось ранее), поскольку пиковый прямой ток светодиодов

не позволяет увеличивать размер дисплея сверх определенного предела. В такой ситуации можно реализовать несколько матриц из светодиодов, управляемых независимыми строками и столбцами (рис. 3.5).

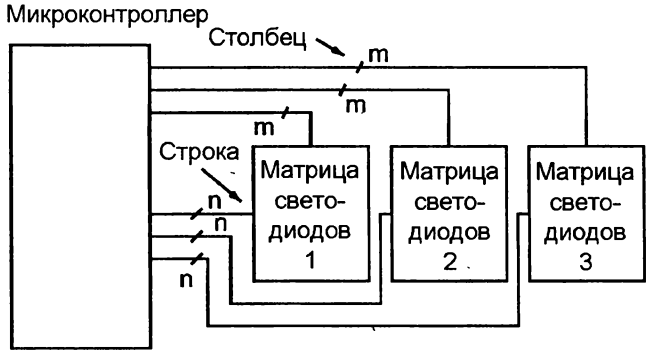


Рис. 3.5. Объединение матриц светодиодов

Теперь может не хватить контактов микроконтроллера. Вместо микроконтроллера с большим числом контактов можно добавить внешние сдвиговые регистры. На рис. 3.6 показана схема для увеличения числа управляющих выходов при помощи сдвигового регистра (например, 74HC164 или 74HC595).

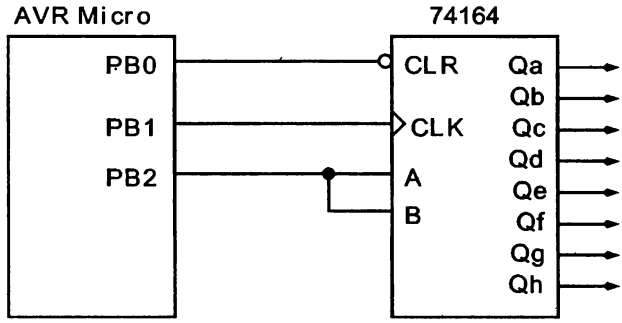


Рис. 3.6. Увеличение числа управляющих выходов с помощью регистра

Два 8-разрядных сдвиговых регистра обеспечивают восемь битов выходных данных. Кроме того, эти сдвиговые регистры можно каскадировать и получить 16 или 24 бита данных (рис. 3.7). Сдвиговые регистры позволяют подключить большое число светодиодов (либо напрямую к каждому контакту, либо при помощи мультиплексирования), но не могут поддерживать ток, для этого потребуются дополнительные транзисторы.

На рис. 3.7 показана схема, в которой три выходных контакта микроконтроллера использованы для получения 16 выходных контактов. Однако нужна программа для сдвига 16 битов данных в 16 выходных контактов. Эта программа должна начинать со сброса двух сдвиговых регистров (генерируя импульс на контакте PB0),

а затем выдавать нужные данные на вход верхнего сдвигового регистра. Нижний сдвиговый регистр получает свои данные из сигнала Qh верхнего сдвигового регистра. После установки нужного логического сигнала на том контакте микроконтроллера, который подключен к входу сдвигового регистра, данные в регистре сдвигаются (по тактовому импульсу на тактовом входе сдвигового регистра — это контакт PB1 микроконтроллера). Каждый тактовый импульс сдвигает данные из входа в Qa, из Qa в Qb и т. д. После 16 импульсов тактовой частоты первый бит данных появляется на контакте Qh нижнего сдвигового регистра. То есть для вывода любых данных нам нужно 16 импульсов тактовой частоты.

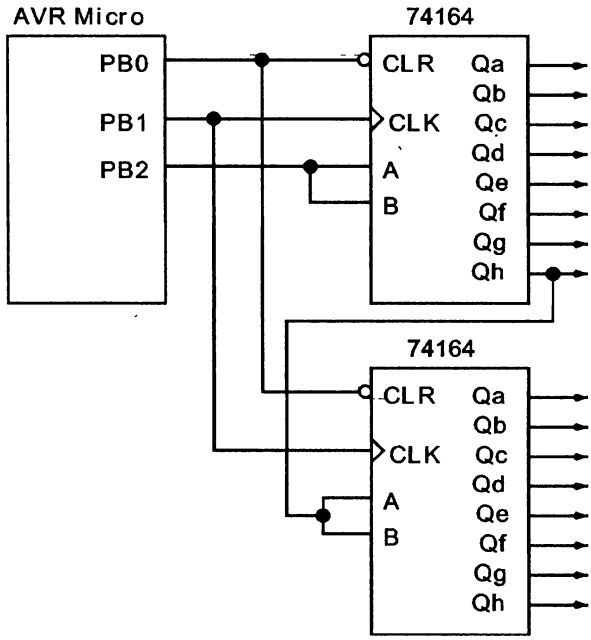


Рис. 3.7. Каскадирование сдвиговых регистров

Изменив конфигурацию подключения сдвиговых регистров и микроконтроллера (рис. 3.8), мы можем уменьшить число тактов с 16 до 8, но за счет дополнительного контакта микроконтроллера (как показано далее). Вход данных на каждый из сдвиговых регистров настраивается микроконтроллером по отдельности. Тактовая частота генерируется микроконтроллером для обоих сдвиговых регистров. Поэтому для вывода 16 битов данных в два сдвиговых регистра потребуется только восемь импульсов тактовой частоты.

По сравнению с рис. 3.7, теперь требуется восемь сигналов тактовой частоты (поскольку ввод данных в два сдвиговых регистра настраивается микроконтроллером по отдельности).

Давайте же применим полученные нами знания для разработки реальной схемы. Одна из самых популярных конфигураций светодиодов — точечно-матричный дисплей 5×7. Внутренняя структура такого дисплея показана на рис. 3.9.

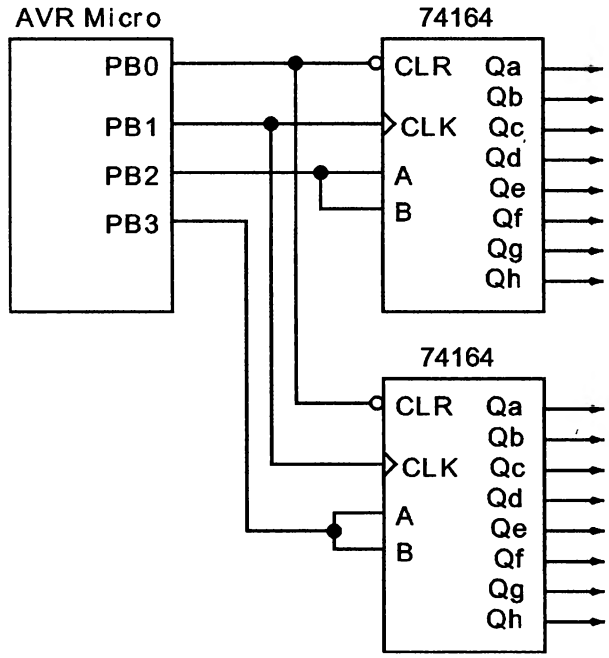


Рис. 3.8. Схема с уменьшенным числом тактов

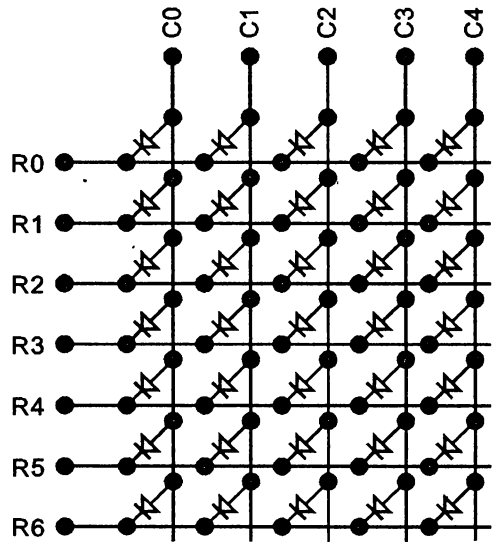


Рис. 3.9. Структура светодиодного дисплея 5×7

Аноды светодиодов подключены к столбцам, а катоды — к строкам. Однако есть и дисплеи 5×7 с подключением к анодам — строк, а к катодам — столбцов. При работе со светодиодными матрицами необходимо обращать внимание на их конфигурацию.

На рис. 3.10 показан способ подключения матрицы светодиодов к микроконтроллеру AVR. У дисплея анод подключен к столбцам, а катод — к строкам. В этой конструкции используется способность микроконтроллера AVR коммутировать ток до 40 мА. Аноды подключены к Vcc через *p-n-p*-транзисторы (по одному).

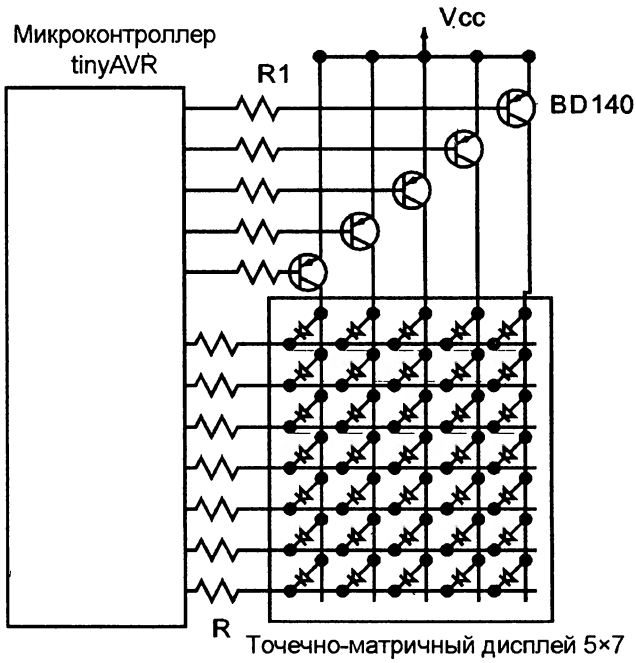


Рис. 3.10. Подключение матрицы светодиодов к микроконтроллеру

В зависимости от того, какой светодиод в данном столбце необходимо зажечь, соответствующая строка подключается к общей шине через контакт микроконтроллера. Резистор R подобран для ограничения тока через светодиод до 40 мА (поскольку именно такой ток выдерживает контакт микроконтроллера AVR). Для включения столбца на соответствующем контакте порта (подключенного к базе транзистора) выставляется логический 0. Это приводит к включению *p-n-p*-транзистора и позволяет току течь через светодиоды. Поскольку столбцов пять, то рабочий ток через светодиод столбца составляет 20%. Поэтому средний ток через светодиод составляет 20% от 40 мА, т. е. 8 мА. Такая схема подходит только для небольших дисплеев. Для более крупных дисплеев (у которых более высокие средние и пиковые токи) потребуется иная схема, обеспечивающая больший ток.

На рис. 3.11 изображена схема для увеличения пикового тока через светодиоды (с ключами на основе *n-p-n*-транзисторов). ULN2003 — это интегральная схема с семью *n-p-n*-каскадами и логическим входом. Каждый выход этой схемы может работать с током до 500 мА. Теперь величину сопротивления токоограничительного резистора R можно уменьшить, что обеспечит более высокий пиковый ток на каждый светодиод.

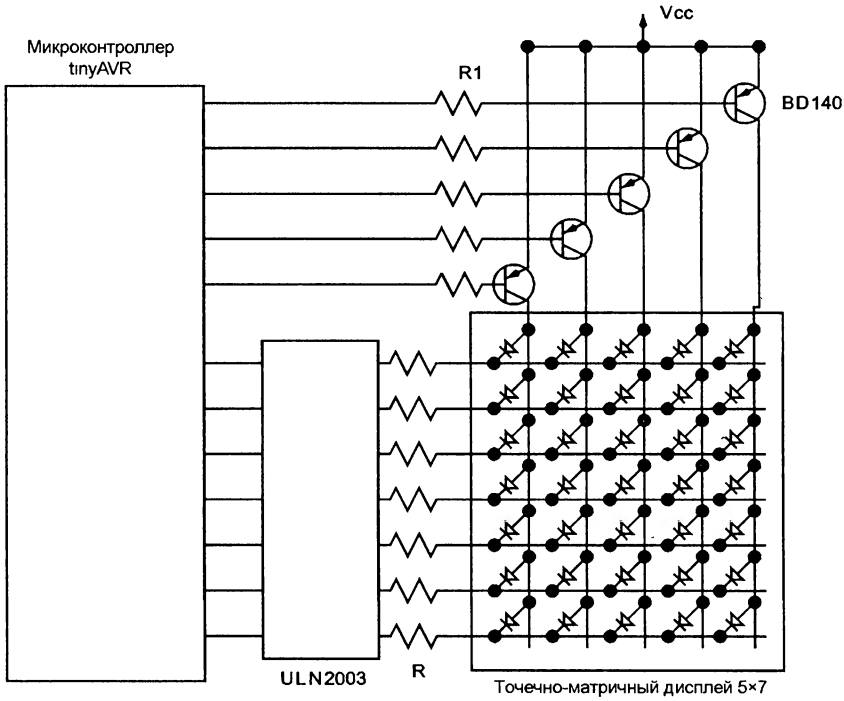


Рис. 3.11. Схема для увеличения тока через светодиоды

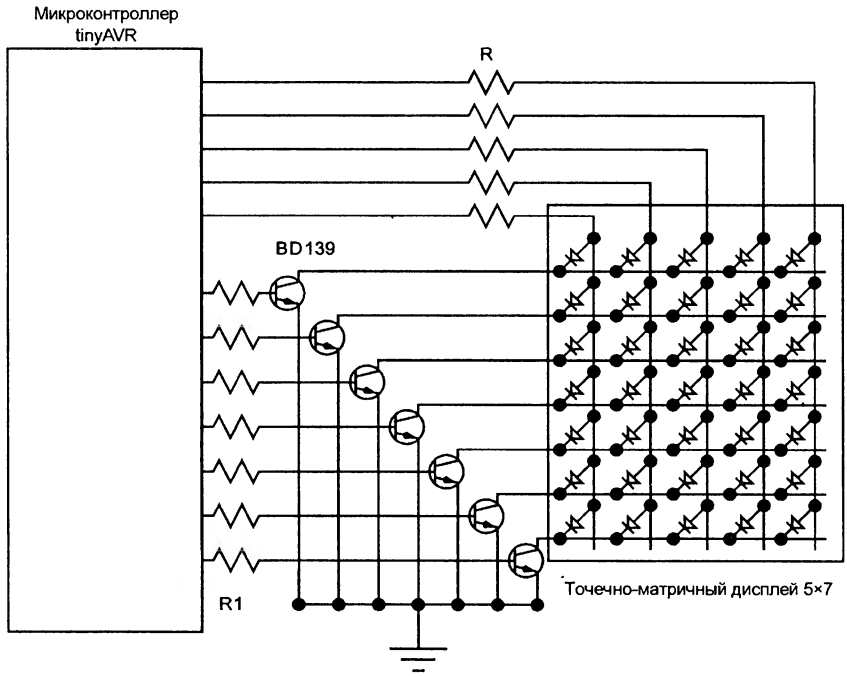


Рис. 3.12. Еще один пример схемы для увеличения тока светодиодов

На рис. 3.12 приведена аналогичная схема, однако здесь применяются *n-p-n*-транзисторы.

В этой схеме все строки активированы при помощи логической 1 на входе базы *n-p-n*-транзистора и в зависимости от того, какой светодиод этой строки должен быть включен в логическую 1, включаются соответствующие аноды. Номинал сопротивления по-прежнему подбирается так, чтобы ограничить ток до 40 мА (поскольку ток поступает напрямую с контактов микроконтроллера). Однако в этой схеме рабочий ток составляет 1/7 (поскольку строк семь) и средний ток через светодиод равен примерно 6 мА, что гораздо меньше, чем средний ток в показанной ранее схеме с *p-n-p*-транзисторами. Однако, поскольку здесь использованы *n-p-n*-транзисторы, схема подходит лишь для небольших дисплеев.

На рис. 3.13 приведена схема подключения к микроконтроллеру AVR точечно-матричного дисплея 16×16 со сдвиговыми регистрами и декодером. На блок-схеме не показаны усиливающие ток транзисторы, которые требуются для коммутации как высокого, так и низкого потенциалов.

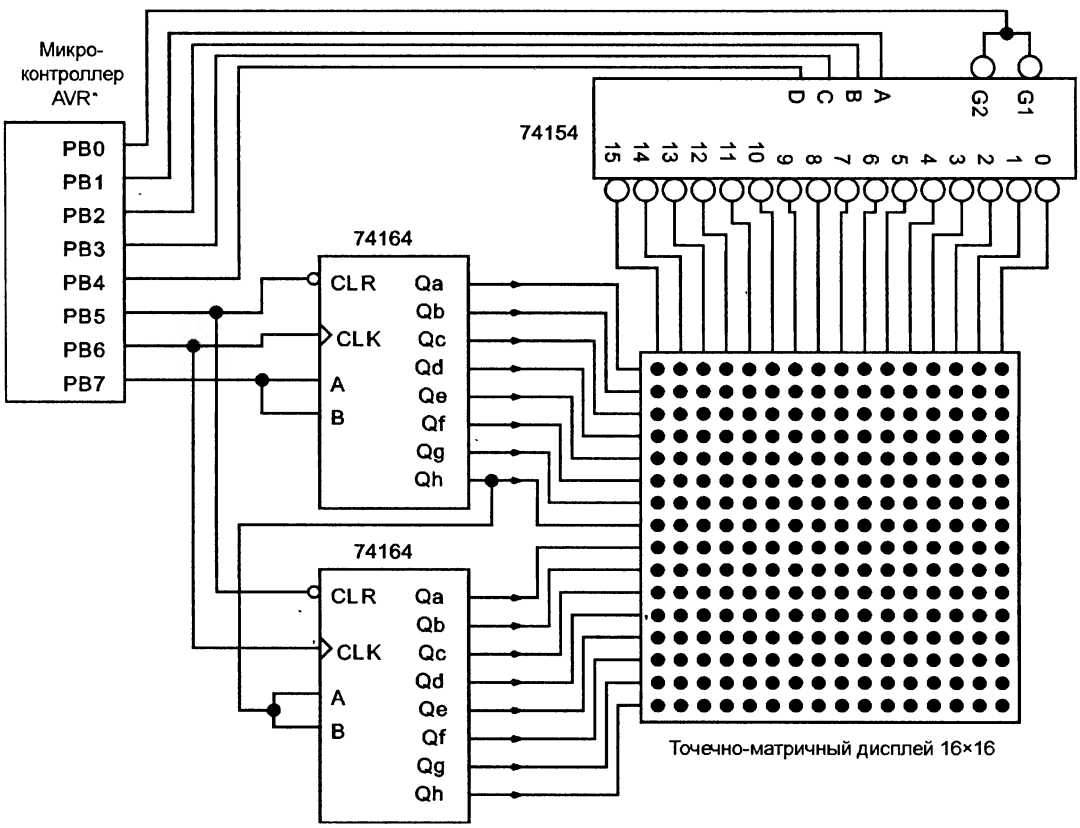


Рис. 3.13. Подключение к микроконтроллеру матричного дисплея 16×16

Особенность этой схемы — наличие декодера 74154 с 16 активными выходами, которые подходят для управления *p-n-p*-транзисторами или *p*-канальными MOS-транзисторами. Дополнительный сигнал с микроконтроллера AVR служит для отключения декодера (при помощи сигналов G1/G2). Пара каскадных сдвиговых регистров выдает 16 выходных сигналов (для управления 16 строками дисплея). Включение конкретных столбцов дисплея выполняется просто: необходимо установить входы декодера в требуемое состояние; если нужно активировать самый левый столбец (столбец номер 0), то на вход декодера подается логическая комбинация ABCD=0000. Для включения столбца 1 подается ABCD=0001 и т. д. Для включения декодера на вход G1/G2 подается логический 0. Если нужно отключить все столбцы, то на вход G1/G2 подается логическая 1. Для изменения столбцов нужно сначала отключить декодер (установив на входе G1/G2 логическую 1), а затем в сдвиговые регистры сдвинуть новые значения для 16 строк (помните, что для этого потребуется 16 импульсов тактовой частоты), а затем активировать новый столбец (при помощи установки входов декодера ABCD), после чего включить сам декодер (установив на входе G1/G2 логический 0). Для реализации рассмотренной схемы потребуется также 16 *p-n-p*-транзисторов (или *p*-канальных MOSFET) на выходе декодера и 16 *n-p-n*-транзисторов (или *n*-канальных MOSFET) на выходе сдвиговых регистров.

Несмотря на то, что на рис. 3.10–3.12 приведены методы управления точечно-матричными дисплеями 5×7 при помощи микроконтроллеров AVR, их можно использовать и для семисегментных, и для алфавитно-цифровых индикаторов (которые также часто встречаются). На рис. 3.14 изображен семисегментный дисплей (слева) и два типа алфавитно-цифровых дисплеев (в центре и справа). Каждый сегмент дисплея промаркирован буквой. Сегменты семисегментного дисплея помечены буквами от A до G, а десятичная точка — dp. Фактически в нем восемь сегментов (включая десятичную точку), но в литературе и спецификациях такой дисплей называется семисегментным. Алфавитно-цифровые дисплеи бывают двух типов: с 14 или 16 сегментами (не считая десятичной точки).

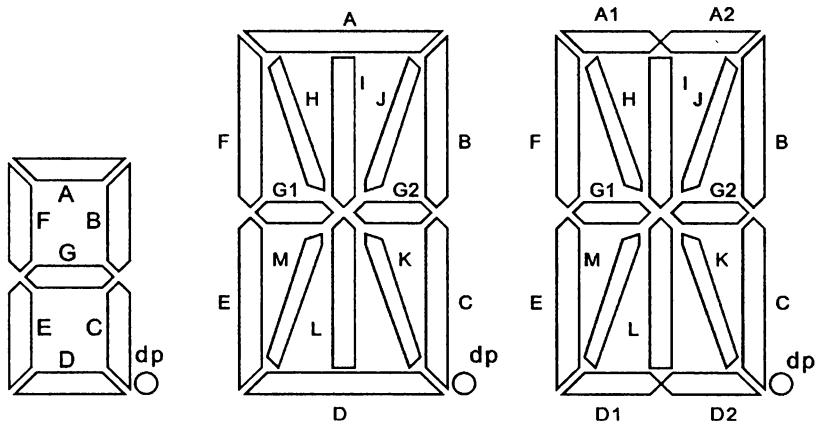


Рис. 3.14. Примеры светодиодных дисплеев

На рис. 3.15 показана организация светодиодов в семисегментном дисплее. Каждый дисплей имеет общий сигнал (либо анод, либо катод). То есть семисегментные дисплеи бывают либо с общим анодом, либо с общим катодом. Точно так же и алфавитно-цифровые дисплеи бывают либо с общим анодом, либо с общим катодом.

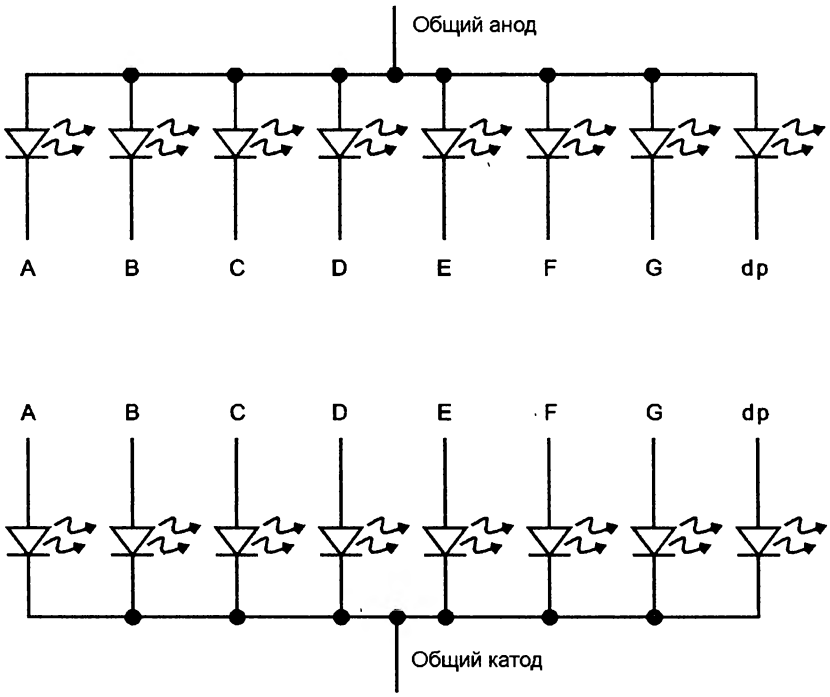


Рис. 3.15. Включение светодиодов семисегментного дисплея

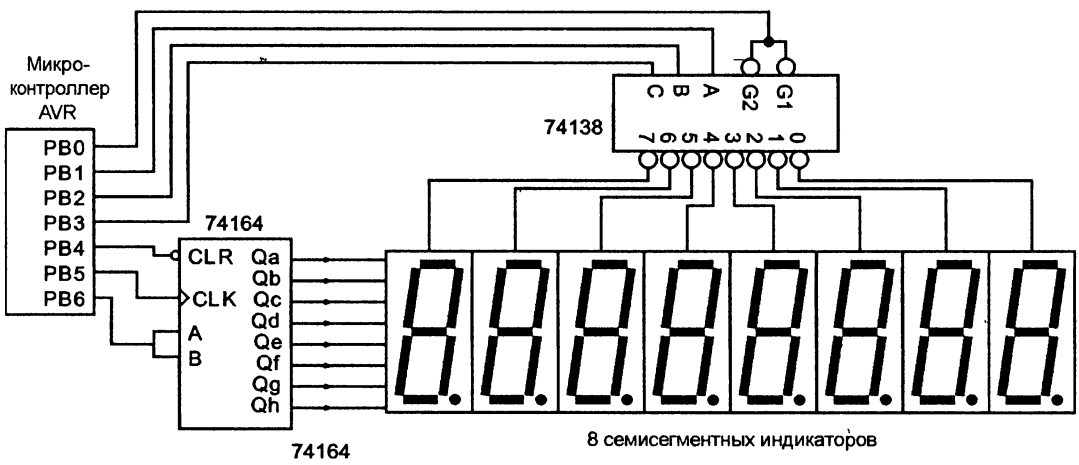


Рис. 3.16. Управление несколькими семисегментными дисплеями

Каждый точечно-матричный дисплей 5×7 можно заменить пятью семисегментными дисплеями. Способ управления восемью семисегментными дисплеями иллюстрирует рис. 3.16. В данной схеме сдвиговый регистр 74164 управляет сегментами семисегментного дисплея, а декодер 74138 — общими анодами дисплеев. Буферные транзисторы здесь опять не показаны.

Мультиплексирование по методу Чарли

В последнее время этот метод мультиплексирования светодиодных дисплеев стал очень популярным из-за того, что он позволяет управлять $N \cdot (N - 1)$ светодиодами при помощи N линий ввода/вывода. Заметим, что стандартное мультиплексирование (описанное в предыдущих разделах) управляет гораздо меньшим количеством светодиодов. В табл. 3.1 указано число светодиодов, которым можно управлять при помощи метода Чарли и стандартного мультиплексирования (распределив имеющиеся N линий ввода/вывода между строками и столбцами). В табл. 3.1 также приведен средний ток через светодиоды (в процентах от максимального тока).

Таблица 3.1. *Сравнение метода Чарли и обычного мультиплексирования*

N (число линий ввода/вывода)	Максимальное число светодиодов, управляемых при помощи мультиплексирования	Средний ток при мультиплексировании, %	Число светодиодов, управляемых при помощи метода Чарли	Средний ток при управлении по методу Чарли, %
2	2	100	2	50
3	3	100	6	16,67
4	4	50	12	8,33
5	6	50	20	5
6	9	33	30	3,33
7	12	33	42	2,4
8	16	25	56	1,78
9	20	25	72	1,38
10	25	20	90	1,11

Недостаток метода Чарли — уменьшение среднего рабочего тока, который идет через светодиоды, поэтому для поддержания нужной яркости необходимо пропорционально повышать пиковый ток через светодиоды (который при этом может быстро достичь максимального пикового значения для светодиодов). Тем не менее, метод Чарли вполне подходит при количестве линий ввода/вывода не более десяти (что позволяет управлять 90 светодиодами). Для управления таким же числом светодиодов при помощи стандартного мультиплексирования потребуется 19 линий ввода/вывода.

Метод Чарли использует высокоимпедансное состояние линий ввода/вывода ("состояние Z") современных микроконтроллеров. Чтобы понять работу дисплея, мультиплексируемого по методу Чарли, обратимся к рис. 3.17, на котором показан микроконтроллер с тремя контактами ввода/вывода и шестью подключенными светодиодами.

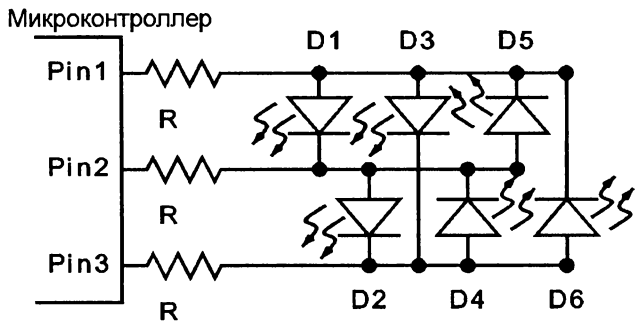


Рис. 3.17. Принцип управления по методу Чарли

Чтобы включить светодиод D1, на контакте Pin1 выставляется 1, на Pin2 — 0, а на Pin3 — состояние "Z" (третье, высокоимпедансное состояние). Большинство современных микроконтроллеров (таких, как AVR) позволяет выходному контакту работать в одном из трех состояний: логическая единица, логический нуль и "Z". Для включения светодиода D2: Pin2 выставляется в логическую 1, Pin3 — в 0, а Pin1 — в "Z". В табл. 3.2 показано состояние контактов для включения всех светодиодов.

Таблица 3.2. *Пример мультиплексирования по методу Чарли*

Номер светодиода	Pin1	Pin2	Pin3
D1	1	0	Z
D2	Z	1	0
D3	1	Z	0
D4	Z	0	1
D5	0	1	Z
D6	0	Z	1

В отличие от обычного дисплея, где активизируется целая строка (или столбец) светодиодов, в дисплее, мультиплексируемом по методу Чарли, включается по одному светодиоду. То есть средний ток через светодиоды равен I_{max}/X, где $X = N \cdot (N-1)$ — общее число светодиодов, N — число выводов. Кроме того, в такой дисплей трудно включить дополнительные усиливающие ток элементы, поэтому пиковый ток определяется возможностями микроконтроллера. Для микроконтроллеров AVR мак-

симальный ток (который может выдать или принять контакт) равен 40 мА. На рис. 3.29 ток светодиода идет через два резистора R. Если напряжение питания равно $U_{ип}$ и напряжение включения светодиода равно U_{LED}, то:

$$I_{max} = (U_{ип} - V_{LED})/2R.$$

Поскольку I_{max} = 40 мА, то R = $(U_{ип} - U_{LED})/2 \cdot I_{max}$. Если $U_{ип}$ = 5 В и светодиод красный (U_{LED} = 2 В), то значение R = 37,5 Ом и в качестве R можно взять резистор со стандартным сопротивлением 39 Ом.

Управление светодиодами по методу Чарли полезно, когда применяются маленькие микроконтроллеры с ограниченным числом выводов. Однако микроконтроллеры не предназначены для выдачи больших токов, поэтому данный метод не рекомендуется использовать больше чем для шести выводов (т. е. для управления больше, чем 30 светодиодами). При 30 светодиодах средний ток окажется примерно 1 мА, что годится только для небольших устройств. В некоторых проектах данной главы мы применяем метод Чарли.

Проект 6. Лампа для создания настроения

Здесь, как и в устройстве смешивания цветов из предыдущей главы (проект 3) также применяется RGB-светодиод. Однако цель описываемой лампы другая — создать свет любого требуемого оттенка, чтобы помочь вам медитировать и расслабиться или просто выбрать освещение под ваше настроение. В проекте смешивания цветов использовался один RGB-светодиод, а интенсивность излучения составляющих его светодиодов настраивалась потенциометрами (чтобы создать нужный цвет). Каждый потенциометр устанавливал интенсивность в значение между 0 и 100% (использовалось 256 уровней), так что можно было сгенерировать 16 миллионов цветов. В новом устройстве применен уже не один RGB-светодиод, а несколько, т. к. наша цель — обеспечить освещение. Лампа не имеет управления интенсивностью свечения каждого светодиода; вместо этого она позволяет выбрать цвет из таблицы цветов, записанной во внутренней энергонезависимой памяти. Каждый цвет в этой таблице представлен тремя значениями интенсивности (для красных, зеленых и синих светодиодов). Яркость свечения светодиода изменяется при помощи широтно-импульсной модуляции с разрядностью в пять битов (т. е. диапазон изменения интенсивности каждого цвета составляет 32 уровня). Блок-схема лампы показана на рис. 3.18.

Напряжение питания светодиодов должно составлять 12 В, почему именно так, мы объясним позже. В этом устройстве использованы имеющиеся в продаже ленты RGB-светодиодов (см. рис. 3.23), состоящие из каскадированных блоков. В метровой ленте светодиодов таких блоков примерно десять. В состав каждого блока входят наборы красных, зеленых и синих светодиодов. Каждый набор состоит их трех последовательно соединенных светодиодов и ограничительного резистора (рис. 3.19).

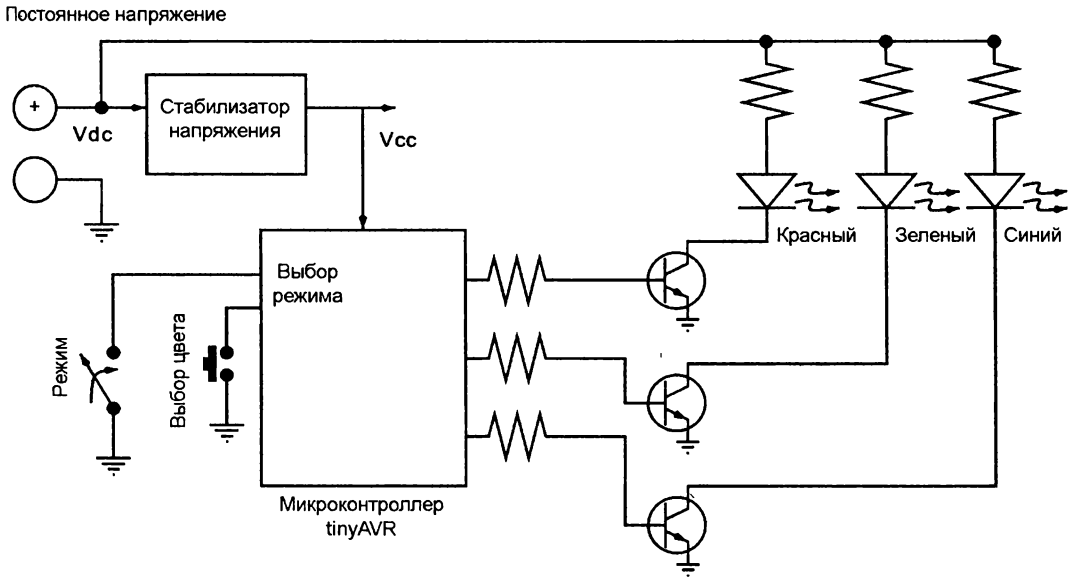


Рис. 3.18. Блок-схема лампы

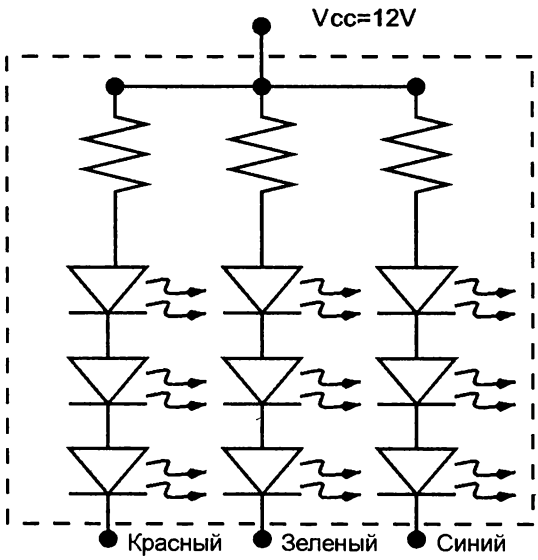


Рис. 3.19. Блок светодиодов

Лампа работает в двух режимах: изменяющегося или фиксированного цвета (выбирается переключателем "Режим"). В режиме фиксированного цвета при помощи переключателя "Выбор цвета" можно выбирать требуемый оттенок. Микроконтроллер питается от стабилизатора напряжения на 5 В, который подключен к источнику питания. В устройстве использованы готовые ленты светодиодов,

однако при необходимости можно применить отдельные мощные светодиоды с высокой яркостью. Удобной будет конфигурация из трех красных, трех зеленых и трех синих светодиодов мощностью по 1 Вт каждый. В этом случае придется так подобрать токоограничивающие резисторы, чтобы ток не превышал 300 мА.

Спецификация проекта

Цель данного проекта — разработать систему освещения на основе RGB-светодиодов, цвет которой будет выбирать пользователь. Суммарная мощность светодиодов каждого цвета должна составлять примерно 10 Вт. Нужно предусмотреть режим постепенного изменения цвета и перебора всех имеющихся цветов. Устройство должно питаться от внешнего источника постоянного напряжения 12 В. Конструкция должна предусматривать возможность установки наборов светодиодов или отдельных светодиодов высокой яркости мощностью по 1 Вт.

Описание устройства

Принципиальная схема устройства приведена на рис. 3.20. Опять использован стабилизатор напряжения LM2940 на 5 В. Входное напряжение может варьироваться от примерно 6 до 20 В. Диод D1 — это диод Шоттки (1N5819), работающий как защитный (как уже объяснялось ранее). Емкости C5 и C7 фильтруют выбросы и нежелательные помехи источника питания. C4 и C6 включены на выходе LM2940. C1 и C3 припаяны около контактов питания микроконтроллера для дополнительной развязки схемы по питанию.

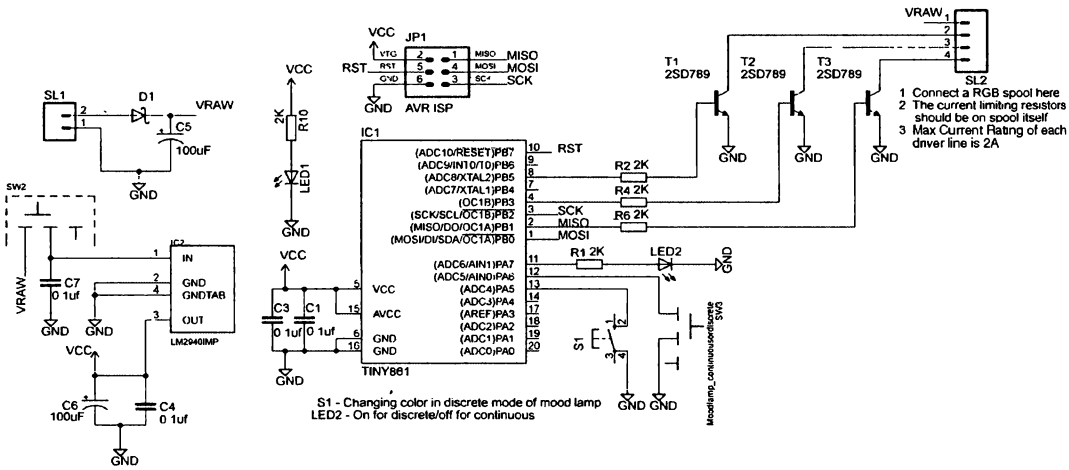


Рис. 3.20. Принципиальная схема устройства

Светодиод LED1 — это индикатор включения/выключения (красного цвета). Микроконтроллер — ATtiny861. У него три аппаратных канала широтно-импульсной модуляции на таймере Timer1 (необходимые для управления тремя транзисто-

рами T1, T2 и T3). *n-p-n*-транзисторы (2SD789) с максимально допустимым током коллектора 2 А включены по схеме с открытым коллектором. Переключатель SW3 предназначен для выбора одного из двух режимов: непрерывного или прерывистого, а кнопка S1 — для изменения цвета в прерывистом режиме. Светодиод LED(2) служит для индикации выбранного режима. SL2 — это 4-контактный разъем для блока светодиодов. Первый контакт используется для подключения анодов светодиодов, а остальные три контакта — для управления катодами красных, синих и зеленых светодиодов. Ограничивающие ток резисторы подключаются снаружи. На светодиоды напряжение VRAW поступает непосредственно от источника питания, чтобы получить большой ток. В противном случае ток был бы ограничен максимальным выходным током стабилизатора напряжения.

При выполнении программы происходит опрос состояния переключателей и если режим непрерывный, то выполняется постоянное изменение скважности на трех каналах аппаратной ШИМ. В прерывистом режиме программа ждет нажатия и отпускания кнопки S1 для обновления значений скважности ШИМ. Разрядность каждого канала ШИМ — пять битов.

Конструкция

Компоновку платы в программе EAGLE и принципиальную схему можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Печатная плата односторонняя (на стороне компонентов есть всего несколько перемычек). Распаянная плата показана на рис. 3.21 и 3.22.

Нам потребовалось 16 футов (около 5 м) ленты со светодиодами, накрутив ее на стеклянную трубку, мы получили лампу для создания настроения (рис. 3.23). На рис. 3.24 показан работающий светодиодный излучатель, подключенный к плате и отображающий одну из комбинаций цветов.

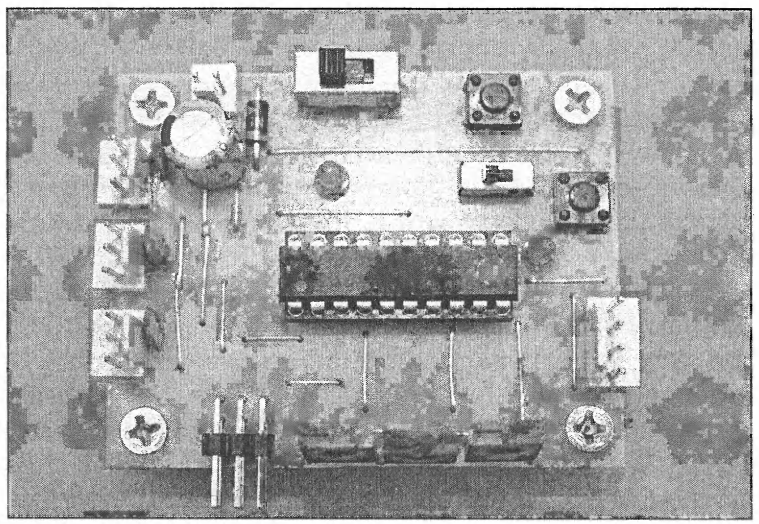
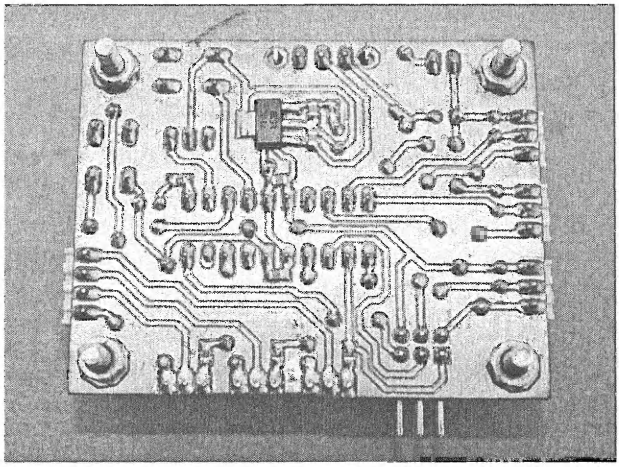


Рис. 3.21. Печатная плата (сторона компонентов)



ис. 3.22. Печатная плата (сторона печатных проводников)

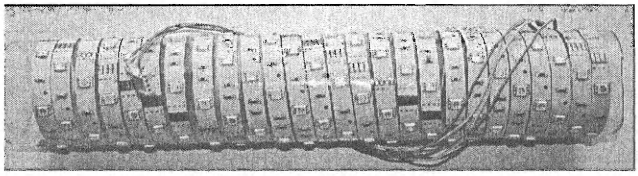


Рис. 3.23. Конструкция светодиодного излучателя

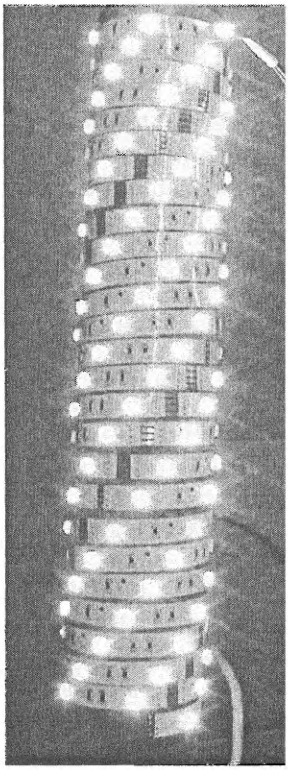


Рис. 3.24. Внешний вид работающей лампы

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно ть по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 8 МГц. Контроллер запрограммирован при по ГК500 в режиме программирования ISP. Рассмотрим самые важные фрагм да.

истинг 3.1

```
ile(1)

or(i=0;i<32;i++)

 OCR1A = i;
 for(j=0;j<32;j++)
```

```
 {
  OCR1B = j;
  for(k=0;k<32;k++)
  {
   OCR1D = k;
   if(mode==CONTINUOUS)
   _delay_ms(500);
   else if(mode==DISCRETE)
   {
    while((PINA&(1<<5))&&(mode==DISCRETE));
       //ждать нажатия кнопки
    _delay_ms(30);
    while((!(PINA&(1<<5)))&&(mode==DISCRETE));
       //ждать отпускания кнопки
    _delay_ms(30);
   }
  }
 }
}
```

Листинг 3.1 — это главный бесконечный цикл программы. Он имеет три вложенных цикла `for`, которые меняют сигналы всех аппаратных каналов ШИМ (при помощи регистров `OCR1A`, `OCR1B` и `OCR1D`). Режим (`mode`) может быть либо `DISCRETE` (прерывистый), либо `CONTINUOUS` (непрерывный); он изменяется при помощи прерывания по состоянию подключенного к переключателю контакта. Если режим прерывистый, то программа ждет нажатия и отпускания переключателя на пятом контакте PORTA (это делается в двух циклах `while`). Как только режим меняется с прерывистого на непрерывный, программа выходит из циклов `while`.

Листинг 3.2

```
ISR(PCINT_vect)
{
 _delay_ms(30);//устранение дребезга
 GIFR = 1<<PCIF;
   //Сброс установленного из-за дребезга переключателя флага
 mode =
   mode==CONTINUOUS?DISCRETE:CONTINUOUS;
 if(mode==CONTINUOUS)
 PORTA&=~(1<<7);//Выключить светодиод
 else
 PORTA|=(1<<7);//Включить светодиод
}
```

Листинг 3.2 — процедура обработки прерывания по изменению состояния контакта, которая вызывается при каждом изменении состояния переключателя. Она изменяет режим и бесконечный цикл (который мы уже обсуждали) прерывается. `CONTINUOUS` и `DISCRETE` — это макросы, которые объявлены в начале программы. В зависимости от режима изменяется и состояние светодиода.

Помимо этого, в остальной части кода есть инициализация таймера Timer1 и его аппаратных каналов ШИМ.

Работа устройства

По умолчанию задан непрерывный режим работы лампы: она постепенно меняет цвета подключенных светодиодов. Если яркость светодиодов достаточно велика, то вы увидите, как необычно поменяется освещение вокруг вас. В прерывистом режиме нужный оттенок цвета выбирают с помощью кнопки.

Проект 7. Волюметр с 20 светодиодами

Волюметры часто имеются на аудиоустройствах для индикации громкости. На старом оборудовании были стрелочные приборы, а на современном часто устанавливают светодиодные индикаторы. Назначение волюметра — дать представление о громкости сигнала. Кроме интенсивности аудиосигнала, подобный индикатор может показывать и другие величины. Светодиодные волюметры встречаются так часто и настолько популярны, что изготовители полупроводников предлагают специальные интегральные схемы для измерения внешних сигналов и отображения результата на светодиодах.

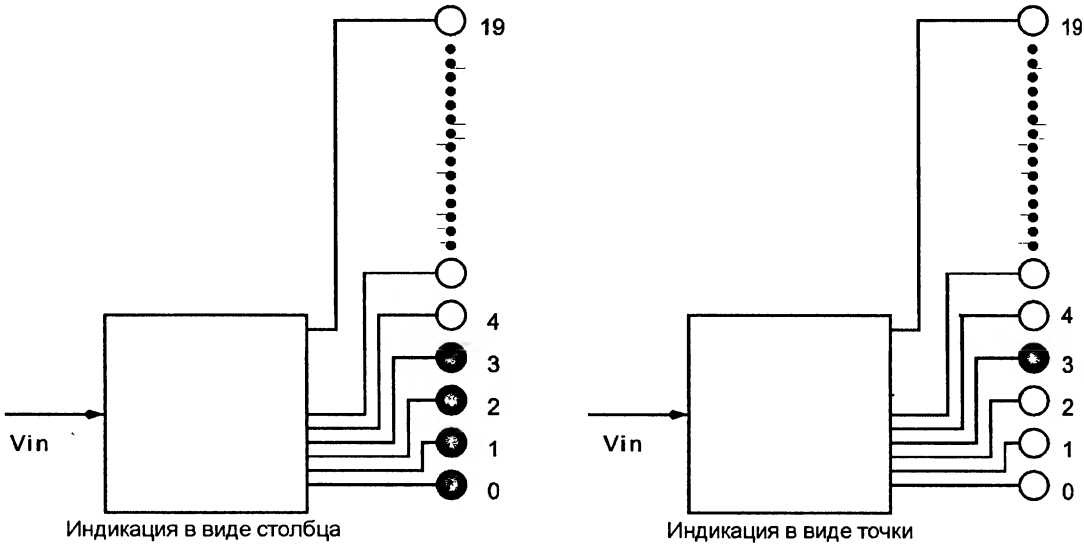


Рис. 3.25. Индикация в виде столбца и точки

Обычно светодиоды волюметра представляют собой один столбец или строку. Одна из таких популярных микросхем — LM3914 компании National Semiconductors — интересное решение для управления точечным дисплеем, для измерения уровня аналогового напряжения и отображения его на десяти светодиодах (организованных в линейный аналоговый индикатор). LM3914 выпускается уже более 20 лет. Но наша задача — создать индикатор с более чем десятью уровнями. Можно было бы каскадировать несколько схем LM3914, но даже после этого характер устройства не изменится — это будет всего лишь линейный индикатор входящего напряжения. Поэтому нам нужно было найти другое решение.

На рис. 3.25 показана работа волюметра в режиме столбца и точки. В режиме столбца при увеличении уровня входного сигнала загорается все больше светодиодов (снизу верх). В режиме точки по мере увеличения входного сигнала загорается один светодиод, соответствующий более сильному сигналу. Светодиоды расположены в столбик по вертикали: нижний светодиод означает более низкий уровень сигнала, чем верхний.

Спецификация проекта

Цель этого проекта — создать универсальный волюметр на 20 уровней, используя как можно меньший микроконтроллер. Сопряжение 20 светодиодов с микроконтроллером можно выполнить при помощи всего пяти контактов ввода/вывода. Таким образом, идеальным компонентом для реализации этого проекта будет восьмиконтактный микроконтроллер, имеющий шесть контактов ввода/вывода. Путем программирования можно настроить индикатор на любую зависимость между входным сигналом и выходными светодиодами. Например, на рис. 3.26 показана логарифмическая шкала. Возможна и линейная зависимость — для этого нужно перепрограммировать микроконтроллер. Чтобы отображать входное напряжение линейным образом, входной сигнал следует делить на постоянное число. Для логарифмической шкалы можно задать таблицу соответствия.

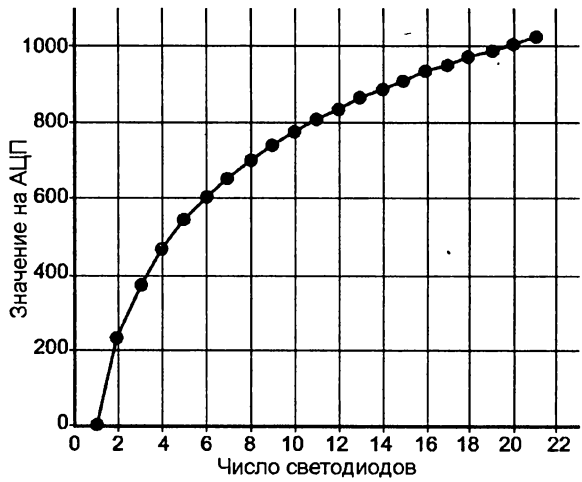


Рис. 3.26. Логарифмическая шкала

Описание проекта

На рис. 3.27 и 3.28 изображены принципиальные схемы проекта. В качестве защитного опять использован диод Шоттки (1N5819).

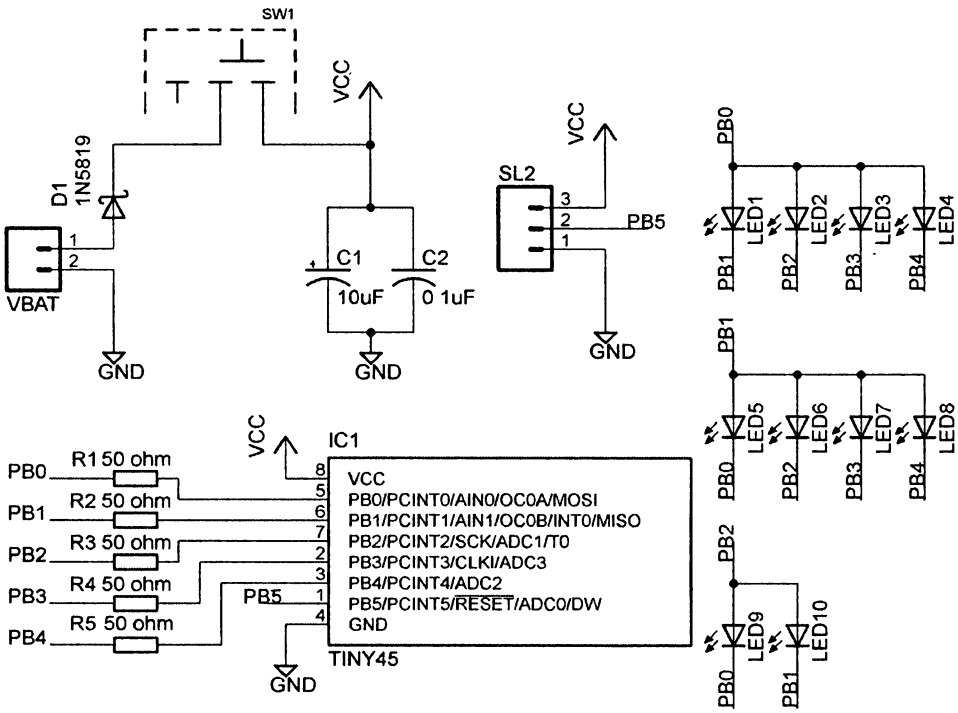


Рис. 3.27. Принципиальная схема устройства (часть 1)

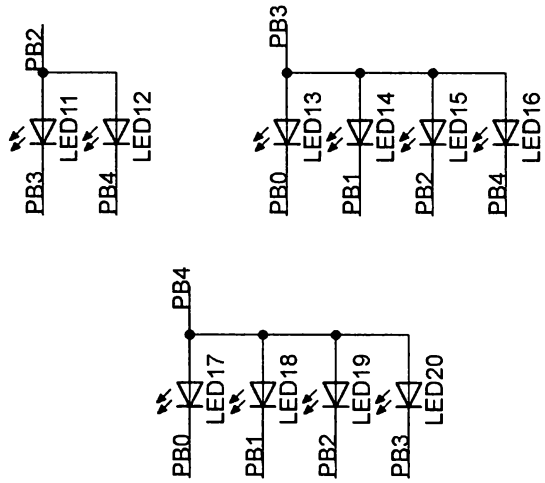


Рис. 3.28. Принципиальная схема устройства (часть 2)

Емкость C1 служит для фильтрации выбросов и нежелательных помех источника питания. C2 — это развязывающий конденсатор (как уже объяснялось ранее). Применен микроконтроллер ATtiny45. Стабилизатора напряжения в этой схеме нет, поэтому входное напряжение может варьироваться лишь от 4,5 до 5,5 В. Светодиоды (с первого по двадцатый) расположены в виде столбика. Поскольку размеры печатной платы под такую конструкцию превышают предельный размер, разрешенный в бесплатной версии программы EAGLE, схема была разбита на две части. Первая часть показана на рис. 3.27, она включает схему управления и десять светодиодов. Остальные десять светодиодов выполнены отдельно (рис. 3.28). Печатные платы этих двух схем проектировались отдельно с таким расчетом, чтобы соединенные вместе они образовали единый столбик из 20 светодиодов.

Двадцать светодиодов мультиплексируются по методу Чарли (при помощи пяти контактов ввода/вывода микроконтроллера). SL2 — это трехконтактный разъем для подачи сигнала на вход АЦП контроллера. Входной сигнал представляет собой постоянное напряжение, регулируемое при помощи потенциометра (контакты которого подключены к VCC, GND и PB5). Им может быть также сигнал, поданный на PB5 и GND. Если это внешний сигнал, то его минимальная и максимальная амплитуды должны быть ограничены 0 (GND) и $U_{ип}$ (VCC).

Программа вычисляет скользящее среднее по десяти последовательным отсчетам АЦП, делит его на 21 уровень (от 0 до 20) и включает соответствующее количество светодиодов.

Конструкция

Компоновку плат в программе EAGLE и принципиальные схемы можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Обе платы односторонние (на стороне компонентов есть всего несколько перемычек). Платы стыкуют, соединяя соответствующие удлиненные печатные дорожки. Полностью собранная конструкция показана на рис. 3.29.

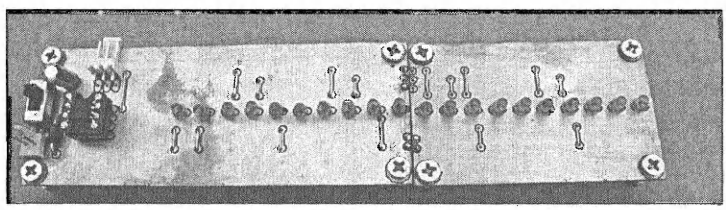


Рис. 3.29. Конструкция волюметра в сборе

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Программа выполняется на тактовой частоте 8 МГц. Вход сброса микроконтроллера (PB5) используется как входной контакт АЦП. Поэтому функцию сброса

этого контакта необходимо отключить (при помощи программирования fuse-бита RSTDISBL). После этого контроллер запрограммировать при помощи ISP будет уже нельзя. Поэтому для программирования контроллера применялся STK500 в режиме последовательного программирования под высоким напряжением (HVSP). Поясним самые важные фрагменты кода.

Листинг 3.3

```
while(1)
{
 //сдвинуть значения
 for(i=0;i<9;i++)
 {
  adcreading[i]=adcreading[i+1];
 }
 //сделать новый отсчет
 adcreading[9] = read_adc();
 //Вычислить сумму и выполнить дискретизацию
 adcsum = 0;
 for(i=0;i<10;i++)
 {
   adcsum = adcsum + adcreading[i];
 }
 //Разделить сумму 10 отсчетов ADC (от 0 до 2550)
 // на 21 уровень(от 0 до 20)
 adcsum = adcsum/122;
 if(level>adcsum)
 {
  for(i=adcsum;i<level;i++)
  {
   statusonoff[i]=0;
  }
 }
 else if(level<adcsum)
 {
  for(i=level;i<adcsum;i++)
  {
   statusonoff[i]=1;
  }
 }
 level=adcsum;
}
```

Листинг 3.3 — главный бесконечный цикл программы. Он удаляет предыдущий отсчет АЦП из буфера `adcreading`, сдвигает остальные значения и берет новый отсчет. АЦП в контроллере ATtiny45 8-разрядный. Затем вычисляется среднее, суммируются все отсчеты в буфере и сумма делится на 21 (это все равно, что сначала поделить сумму на 10, а затем поделить это среднее на 21 уровень). Затем нужное количество светодиодов либо включается, либо выключается (в зависимости от предыдущего уровня). И наконец, значение переменной `level` обновляется текущим отсчетом. Код мультиплексирования по методу Чарли (как объясняется в следующем разделе) был написан так, чтобы в любой момент, когда `statusonoff[p]` получает значение 1, загорался светодиод в позиции соответствующей `p` (и наоборот). Соответствие между уровнем сигнала и числом светящихся светодиодов таково, что при уровне 0 не светится ни один светодиод, а при уровне 20 (самом высоком) включаются все 20 светодиодов.

Листинг 3.4

```
//Процедура обработки переполнения таймера timer0
ISR(TIM0_OVF_vect)
{
 DDRB=0;
 PORTB=statusonoff[count])
  <<pgm_read_byte(&anode[count])
  |0<<pgm_read_byte(&cathode[count]);
 DDRB = 1<<pgm_read_byte(&anode[count])
  |1<<pgm_read_byte(&cathode[count]);
 count++;
 if(count==20)
 count=0;
}
```

Листинг 3.4 — процедура обработки прерывания по переполнению таймера Timer0. Поэтому она вызывается с определенной периодичностью. Она выполняет мультиплексирование методом Чарли всех 20 светодиодов. Счетчик `Count` отслеживает тот светодиод, с которым мы работали в предыдущий интервал времени. Хранящееся в `statusonoff [count]` значение выдается на его анод, а 0 — на катод. Затем контакты анода и катода объявляются выходами (посредством обновления регистра DDRB). После этого светодиод включается (если `statusonoff [count] = 1`) или наоборот. Этот цикл повторяется для каждого светодиода.

Остальной код состоит из инициализации АЦП и таймера Timer0.

Работа устройства

Мы протестировали устройство, припаяв к SL2 потенциометр в 10 кОм. Мы меняли значение постоянного напряжения и наблюдали за столбиком светодиодов.

Проект 8. Вольтметр

Этот проект (и следующие два также) построен на стандартной элементной базе, состоящей из семисегментного индикатора на две с половиной цифры и восьмиконтактного микроконтроллера. Семисегментные индикаторы часто применяются в приборах. Обычная конфигурация индикатора — три с половиной цифры (три полных цифры и "половинка", в которой при необходимости может высвечиваться "1"). Такой индикатор может показать значение от 0 до 1999. Если он отображает и символ "минус", то диапазон значений от –1999 до 1999. Индикаторы, имеющие четыре с половиной цифры, отображают диапазон от 0 до 19999 или от –19999 до 19999, что в десять раз превышает диапазон индикатора с тремя с половиной цифрами. Однако для многих приложений вполне достаточно индикатора в две с половиной цифры. Семисегментный индикатор можно собрать из отдельных светодиодов. Преимущество такого индикатора в том, что для него можно выбрать светодиоды любого цвета и размера. На рис. 3.30 показано, как можно сделать семисегментный индикатор из отдельных светодиодов. Каждый сегмент индикатора (за исключением десятичной точки) делается из трех параллельных светодиодов. Вспомним из предыдущей главы, что для управления параллельными светодиодами нужны сопротивления, ограничивающие ток. Однако мы не собираемся этого делать и подключим по три светодиода параллельно в каждом из семи сегментов. Было бы неплохо рассортировать эти светодиоды по интенсивности (хотя это придется делать вручную и займет много времени).

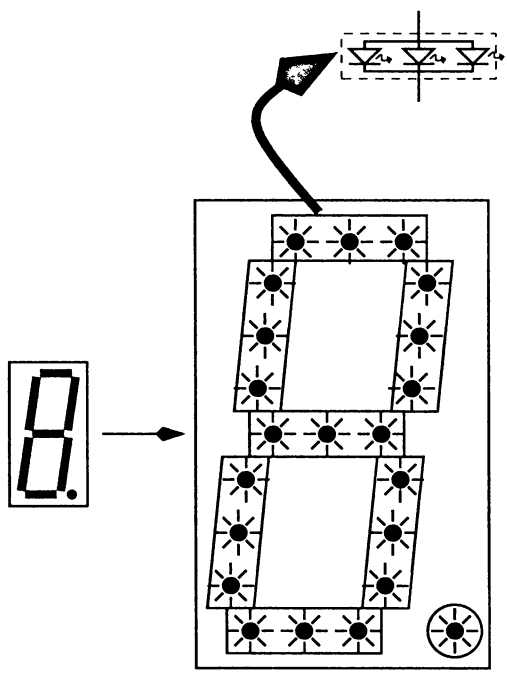


Рис. 3.30. Семисегментный индикатор, выполненный из отдельных светодиодов

Каждый семисегментный индикатор имеет восемь сегментов. Восьмиконтактный микроконтроллер имеет два контакта питания и шесть контактов ввода/вывода. При помощи пяти контактов мы можем управлять 20 светодиодами (по методу Чарли). То есть мы можем подключить наш индикатор на две с половиной цифры к восьмиконтактному микроконтроллеру при помощи пяти контактов ввода/вывода. Шестой контакт можно задействовать для других целей, например, для считывания внешнего аналогового напряжения или цифрового входного сигнала. Двадцать управляемых по методу Чарли светодиодов образуют два семисегментных индикатора (для этого нужно 16 светодиодов). Остальные четыре светодиода служат для отображения десятичной точки и знака "минус" (рис. 3.31).

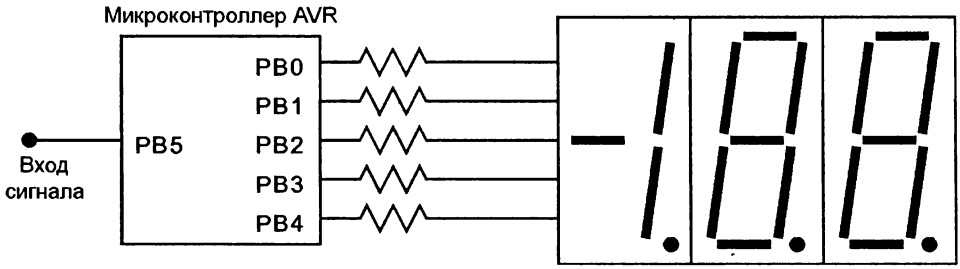


Рис. 3.31. Управление индикатором от микроконтроллера

Спецификация проекта

Цель проекта — создать однодиапазонный вольтметр на основе восьмиконтактного микроконтроллера. Индикатор в две с половиной цифры реализован при помощи мультиплексирования методом Чарли (как объяснялось в предыдущем разделе). Микроконтроллеры AVR серии Tiny имеют несколько 10-разрядных АЦП. Это позволит различить 1024 уровня. Индикатор в две с половиной цифры может отобразить лишь 200 значений. Поэтому разрядность АЦП не является ограничением для вольтметра. Мы выбрали диапазон от 0 до 12 В, что дает на нашем индикаторе точность в 0,1 В. Внутреннее опорное напряжение микроконтроллера составляет 2,56 В, поэтому для получения диапазона в 12 В добавлен внешний делитель напряжения 1:4,9. Коэффициент деления можно легко изменить, если вам требуется другой диапазон.

Описание устройства

На рис. 3.32 и 3.33 изображены принципиальные схемы устройства. В проектах 8, 9 и 10 используется эта же схема. C1 — развязывающий конденсатор для удаления помех, возникающих в цепи (он припаивается возле контактов питания контроллера). Выбранный микроконтроллер — ATtiny45. Стабилизатора напряжения нет, поэтому напряжение питания может варьироваться лишь от 4,5 до 5,5 В. Отсутствует и конденсатор для фильтрации выбросов напряжения, поэтому рекомендуется питать уст-

ройство от батарей. Двадцать светодиодов организованы в виде двух полных семисегментных индикаторов (по семь светодиодов для каждой цифры и по одному на десятичную точку), одного индикатора на полцифры (два светодиода для отображения 0 или 1 и один — для десятичной точки) и одной горизонтальной черточки (один светодиод), которая обозначает знак "минус". Поскольку сегменты цифр семисегментного индикатора длиннее, чем размер трехмиллиметрового светодиода, то для каждого сегмента параллельно соединены три светодиода. Это может привести к разному току светодиодов сегментов и точки. Для выравнивания тока добавлен резистор, подключенный последовательно к светодиоду точки.

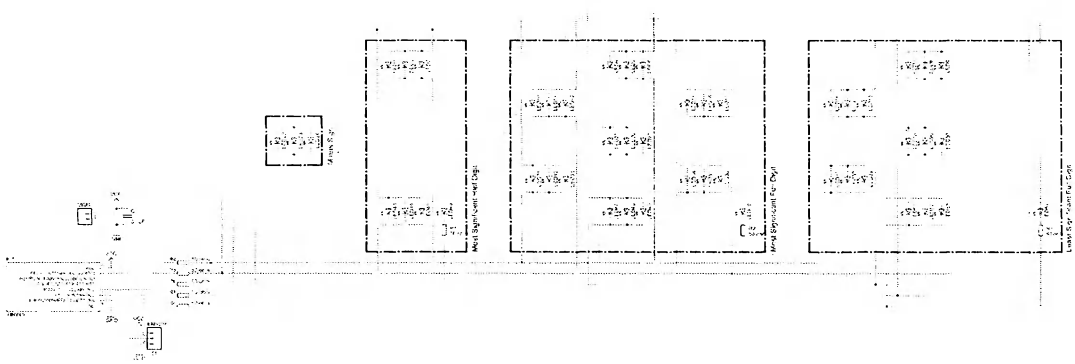


Рис. 3.32. Принципиальная схема вольтметра (термометра, частотомера) с автоматическим диапазоном (часть 1)

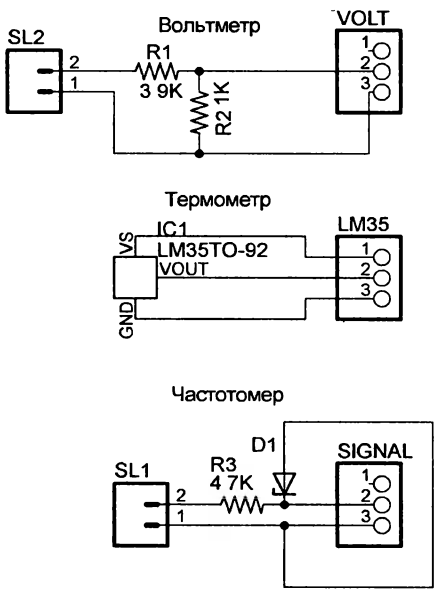


Рис. 3.33. Принципиальная схема вольтметра (термометра, частотомера) с автоматическим диапазоном (часть 2)

Двадцать светодиодов управляются посредством мультиплексирования по методу Чарли при помощи пяти контактов ввода/вывода микроконтроллера. SL1 — это трехконтактный разъем для подачи входного сигнала на АЦП контроллера. Входной сигнал в проектах 8, 9 и 10 разный. Вторая часть схемы (рис. 3.44) служит для подачи на контроллер разных входных сигналов (путем подключения разъема SL1 к клеммам VOLT, LM35 или SIGNAL). Для данного проекта на вход поступает напряжение с SL2. Затем оно снижается в 4,9 раз и подается на вход АЦП через разъем VOLT.

Программа считывает отсчет с выхода АЦП, преобразует его в напряжение, выполняет округление и отображает результат на семисегментном дисплее.

Конструкция

Компоновку платы в программе EAGLE (и принципиальную схему) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Основная (первая) печатная плата сделана двухсторонней. На второй плате перемычек нет, она односторонняя. На рис. 3.34 и 3.35 показаны верхняя и нижняя стороны основной платы. На рис. 3.36 изображена собранная плата разъемов (вторая).

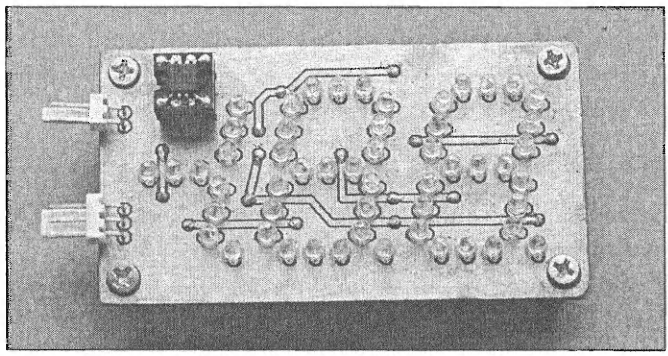


Рис. 3.34. Основная (первая) плата (сторона компонентов)

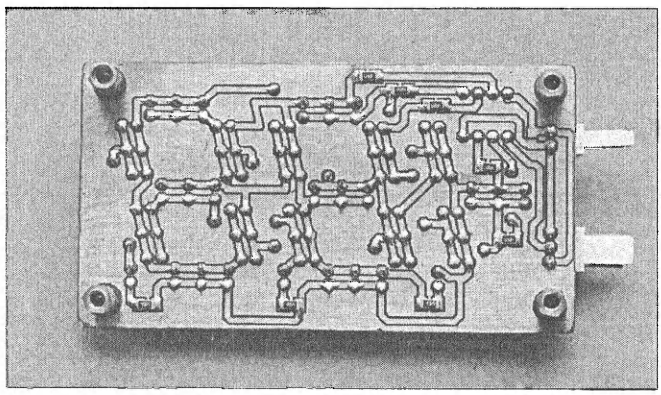


Рис. 3.35. Основная (первая) плата (сторона печатных проводников)

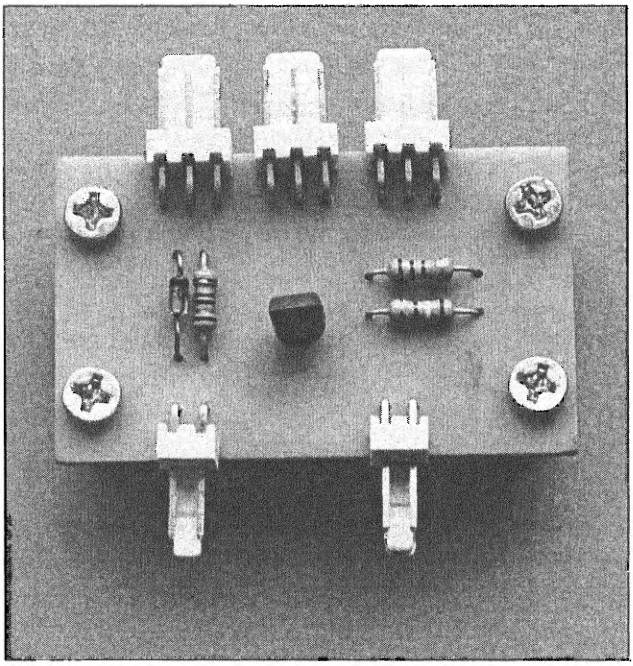


Рис. 3.36. Собранная плата входных разъемов (вторая)

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 8 МГц. В данном случае контакт сброса PB5 микроконтроллера служит входом АЦП. Поэтому функцию сброса этого контакта необходимо отключить, установив fuse-бит RSTDISBL. Для программирования контроллера был использован STK500 в режиме HVSP. Рассмотрим самые важные фрагменты кода.

Листинг 3.5

```
while(1)
 voltage = read_adc();
 //Здесь напряжение в 100 раз больше реального
 //Определение диапазона и масштабирование результата
 if(voltage<=199)
 {
  point = 1;
 }
 else if(voltage<1995)
```

```
{
 if(voltage%10>=5)
 {
  voltage = voltage +10;
 }
 voltage = voltage/10;
 point = 2;
}
else
{
 if(voltage%100>=50)
 {
  voltage = voltage+100;
 }
 voltage = voltage/100;
 point = 3;
}
c=voltage/100;
voltage = voltage%100;
d= voltage/10;
voltage = voltage%10;
e = voltage;
display(c,d,e,point,0);
_delay_ms(100);
}
```

Листинг 3.5 — это главный бесконечный цикл программы. Сначала он читает отсчет АЦП (при помощи функции `read_adc`, которая выдает значение, в 100 раз превышающее входное напряжение). Разрядность АЦП равна 10; опорным служит внутреннее напряжения 2,56 В (для повышения точности). Поскольку имеется делитель напряжения 4,9 : 1, то измеряемое входное напряжение может варьироваться в пределах от 0 до 2,56 · 4,9 = 12,544 В (примерно 12 В).

После того как считан результат АЦП, выполняется его округление (сначала нужно выбрать положение десятичной точки, а затем округлить значение до трех цифр). Затем цифры извлекаются и передаются в функцию отображения (которая ставит им в соответствие массив `statusonoff`). Назначение этого массива такое же, что и в проекте 7.

Работа устройства

Входное напряжение подается на клеммы VOLT, а его значение отображается на семисегментном дисплее.

Проект 9. Термометр

Аппаратная часть проекта аналогична предыдущему. Но вместо цепи внешнего делителя напряжения в этой схеме используется датчик температуры. Он преобразует температуру в напряжение, которое измеряется АЦП микроконтроллера и переводится в градусы Цельсия или Фаренгейта, отображаемые на дисплее. Существует много различных датчиков температуры. Самые распространенные: термистор, термопара, кремниевый датчик температуры. Термистор (термосопротивление) дешев и доступен. Простая схема преобразования температуры в напряжение приведена на рис. 3.37. Однако изменение сопротивления термистора не является линейной функцией от температуры. Для точного измерения температуры требуется сложное математическое уравнение Стейнхарт-Харта (Steinhart-Hart).

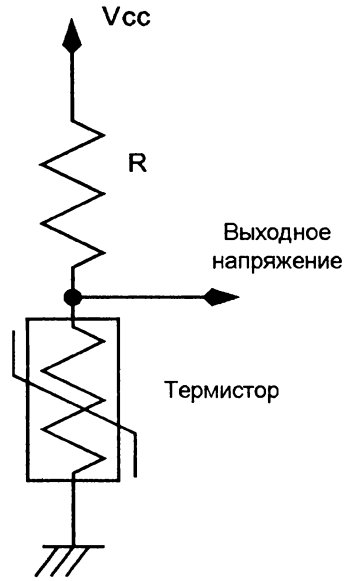


Рис. 3.37. Преобразование температуры в напряжение

Термопара хорошо подходит для измерения больших значений (порядка сотен градусов Цельсия). Термопара (подобно термистору) — также нелинейный датчик температуры. Хотя напряжение термопары зависит от температуры, но чтобы преобразовать выходное напряжение термопары в отсчет температуры, нужна аппроксимация полиномами. Кремниевый датчик температуры использовать проще всего. Он выдает напряжение (или ток), прямо пропорциональное температуре. В этом проекте мы выбрали датчик LM35 компании National Semiconductors, который совместим с датчиком TMP36 компании Analog Devices. Датчик LM35 выдает 10 мВ на один градус Цельсия. Микроконтроллер имеет 10-разрядный АЦП с опорным напряжением 2,56 В, который обеспечивает хорошую точность измерения температуры (доли градуса Цельсия).

Спецификация проекта

Цель проекта — измерить напряжение от датчика температуры и отобразить температуру в градусах Цельсия или Фаренгейта на семисегментном дисплее в две с половиной цифры. Источник питания на принципиальной схеме не показан, подойдет любой стабилизированный источник питания (линейный стабилизатор или адаптер постоянного тока). Рекомендуемое напряжение источника — 5 В. Можно также соединить последовательно четыре щелочные батареи по 1,5 В или даже четыре никель-металлогидридных аккумулятора по 1,2 В.

Описание устройства

Схема аналогична проекту 8, но входной сигнал поступает с температурного датчика LM35, который выдает 10 мВ напряжения на каждый градус Цельсия. Далее напряжение подается на вход АЦП, а результат отображается на дисплее (после автоматического выбора предела измерения).

Программа помимо отображения температуры выполняет также и преобразование из градусов Цельсия в градусы Фаренгейта. Режим отображения температуры (градусы Цельсия/градусы Фаренгейта) переключается каждые 5 секунд. Горизонтальная черточка включается для режима "Фаренгейт" и выключается для режима "Цельсий".

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 8 МГц. Функцию сброса контакта PB5 необходимо отключить (установив fuse-бит RSTDISBL). Рассмотрим самые важные фрагменты кода.

Листинг 3.6

```
While(1)
{
 temperature = read_adc();
 //Здесь температура в 100 раз больше реальной
 if(temperature<=199)
 {
  point = 1;
 }
 else if(temperature<1995)
 {
  if(temperature%10>=5)
  {
```

```
   temperature = temperature +10;
  }
  temperature = temperature/10;
  point = 2;
 }
 else
 {
  if(temperature%100>=50)
  {
   temperature = temperature+100;
  }
  temperature = temperature/100;
  point = 3;
 }
 c=temperature/100;
 temperature = temperature%100;
 d= temperature/10;
 temperature = temperature%10;
 e = temperature;
 display(c,d,e,point,mode==FAH);
 _delay_ms(100);
}
```

Листинг 3.6 — это главный бесконечный цикл программы. Он сначала читает отсчет АЦП (при помощи функции `adc_read`, которая выдает значение, в 100 раз превосходящее реальную температуру). Температура отображается либо в градусах Цельсия, либо Фаренгейта (в зависимости от режима). АЦП работает в 10-разрядном режиме; в качестве опорного выбрано внутреннее напряжение в 1,1 В (для повышения точности).

После получения отсчета АЦП выполняются вычисления: сначала выбирается место десятичной точки, а затем результат округляется до трех цифр. После этого цифры передаются в функцию отображения, которая устанавливает соответствие с массивом `statusonoff`. Назначение этого массива то же, что и в проектах 7 и 8. Таймер Timer0 помимо управления мультиплексированием по методу Чарли отсчитывает также и пять секунд для переключения режима отображения между шкалами Цельсия и Фаренгейта.

Работа устройства

Устройство постоянно показывает текущую температуру, причем каждые пять секунд режим отображения переключается. Шкалу Фаренгейта обозначает горизонтальная черточка (как объяснялось ранее).

Проект 10. Частотомер с автоматическим выбором диапазона

Частотомер — это прибор, который измеряет частоту внешнего сигнала. Входной сигнал может быть как аналоговым, так и цифровым. На рис. 3.38 показана блок-схема частотомера.

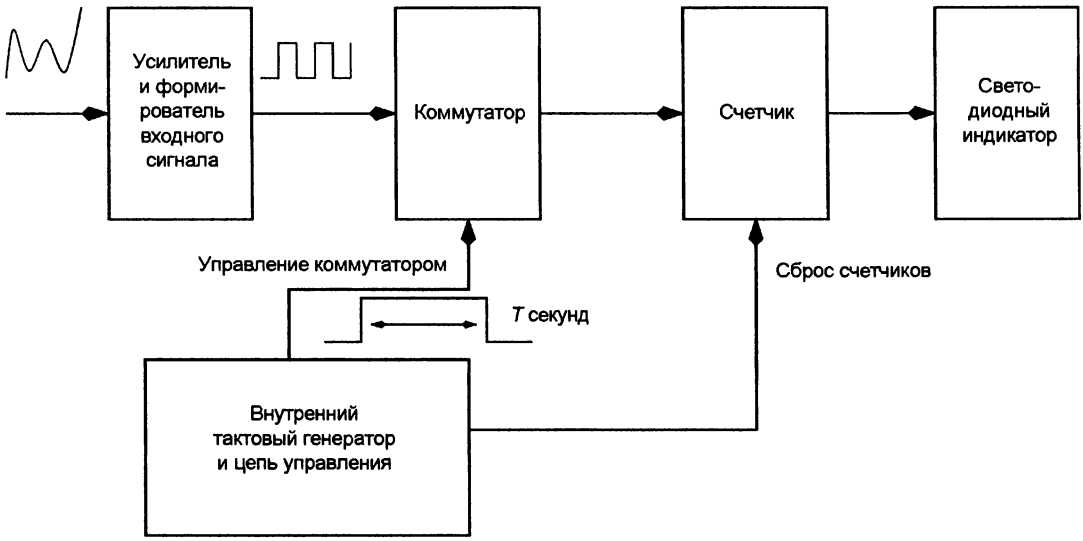


Рис. 3.38. Блок-схема частотомера

На вход поступает аналоговый сигнал. Входной усилитель и формирователь усиливают его и задают пороговое значение (для преобразования в цифровой сигнал). Устройство имеет точный внутренний тактовый генератор, который управляет коммутатором счетчика. Продолжительность времени, в течение которого входной сигнал поступает через коммутатор на счетчик (время отсчета), зависит от требуемой точности измерения. Например, для измерения сигнала с точностью до 1 Гц коммутатор должен включаться на одну секунду. Для измерения с точностью в 0,1 Гц время отсчета должно составлять десять секунд. Результат на выходе счетчика отображается на светодиодном индикаторе.

Если частота входящего сигнала очень низкая, то для ее измерения понадобится чрезвычайно длительное время отсчета. Один из способов решения этой проблемы — использовать в качестве сигнала на коммутаторе сам входной сигнал и замерять частоту, генерируемую внутренним тактовым генератором (рис. 3.39). Однако так мы получим период входного сигнала. Для преобразования периода в частоту нужно выполнить простой математический расчет.

На рис. 3.40 приведена временная диаграмма сигналов частотомера. Сигналы на счетчике периодов аналогичны.

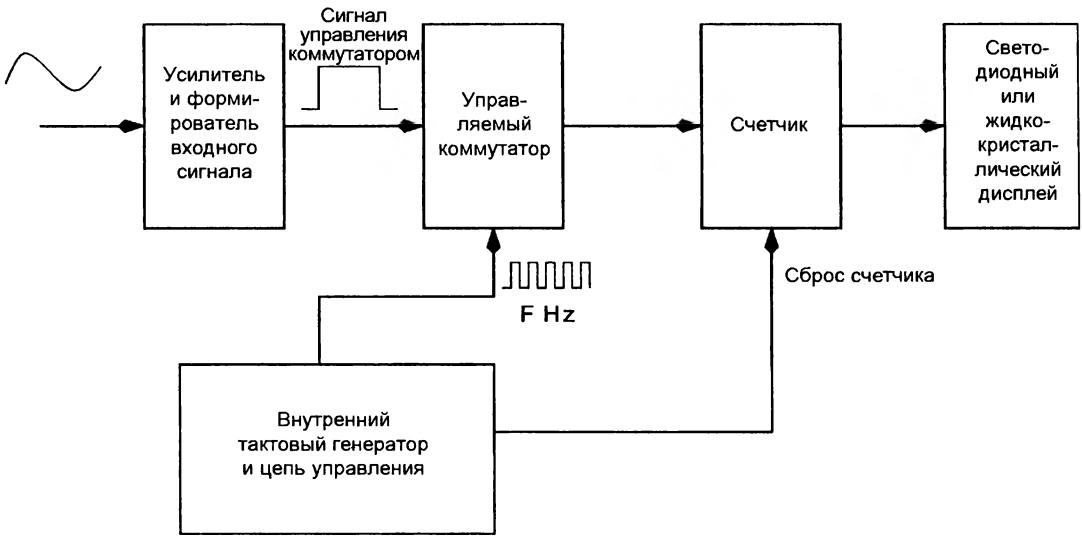


Рис. 3.39. Принцип измерения периода сигнала

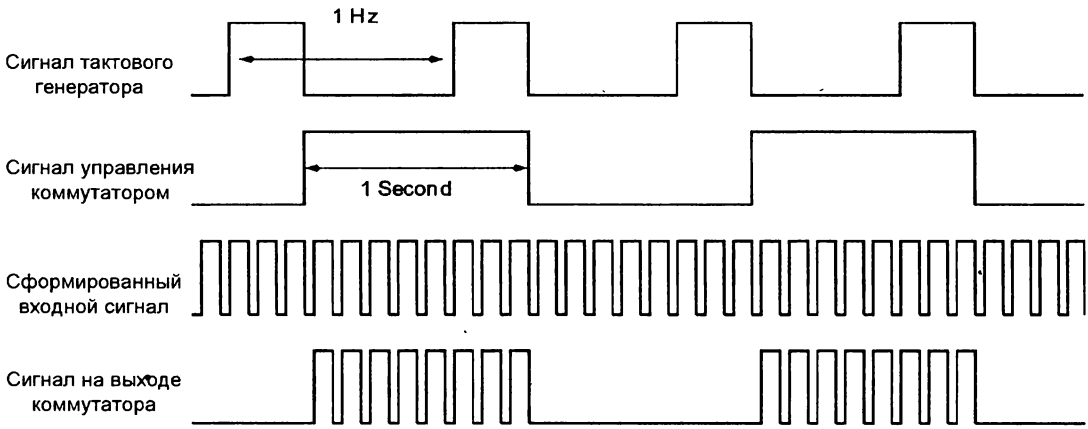


Рис. 3.40. Временная диаграмма сигналов частотомера

Спецификация проекта

Устройство должно измерять частоту входного сигнала и отображать результат на семисегментном дисплее в две с половиной цифры. Все блоки, показанные на рис. 3.38, имеются в самом микроконтроллере, который может также выполнить и необходимые вычисления для преобразования периода в частоту.

Частотомер с автоматическим определением диапазона может автоматически выбрать шкалу отсчета (по частоте входного сигнала). Наш частотомер имеет дисплей с диапазоном от 0 до 199, диапазоны отображаемых частот приведены в табл. 3.3.

Таблица 3.3. Диапазоны частотомера

Номер диапазона	Интервал частот, кГц	Время отсчета, с
1	0 – 0,199	1
2	0 – 1,99	0,1
3	0 – 19,9	0,01

Выбранный диапазон измерения указывают десятичные точки. Для первого диапазона измерения включается правая десятичная точка; для второго — средняя, для третьего — левая.

Описание устройства

Схема устройства аналогична проектам 8 и 9, но входной сигнал подается с разъема SL1 схемы, показанной на рис. 3.33. D1 — это стабилитрон на 5 В, который ограничивает амплитуду сигнала, а R3 — токоограничительный резистор. Входной сигнал должен быть однополярным (положительной полярности).

Программирование

Откомпилированный код проекта (вместе с файлом MAKEFILE) можно скачать с адреса: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 8 МГц. Поясним самые важные фрагменты кода.

Листинг 3.7

```
ISR(PCINT0_vect)
{
 edgecounter++;
}
```

Листинг 3.7 — процедура обработки прерывания по изменению состояния контакта PB5. Счетчик `edgecounter` удваивает частоту в конце периода работы коммутатора. Время отсчета установлено равным одной секунде. В конце каждого периода работы коммутатора число импульсов за секунду подсчитывается путем деления `edgecounter` на 2. Затем результат округляется (как в предыдущих двух проектах) и отображается на дисплее.

Работа устройства

Частота измеряется путем подачи входного сигнала на разъем SL1. Необходимо отметить, что показания обновляются только раз в секунду.

Проект 11. Забавные часы

Существует множество часов различных размеров и форм. В наших часах для индикации времени используются разные типы светодиодов. Блок-схема часов приведена на рис. 3.41.

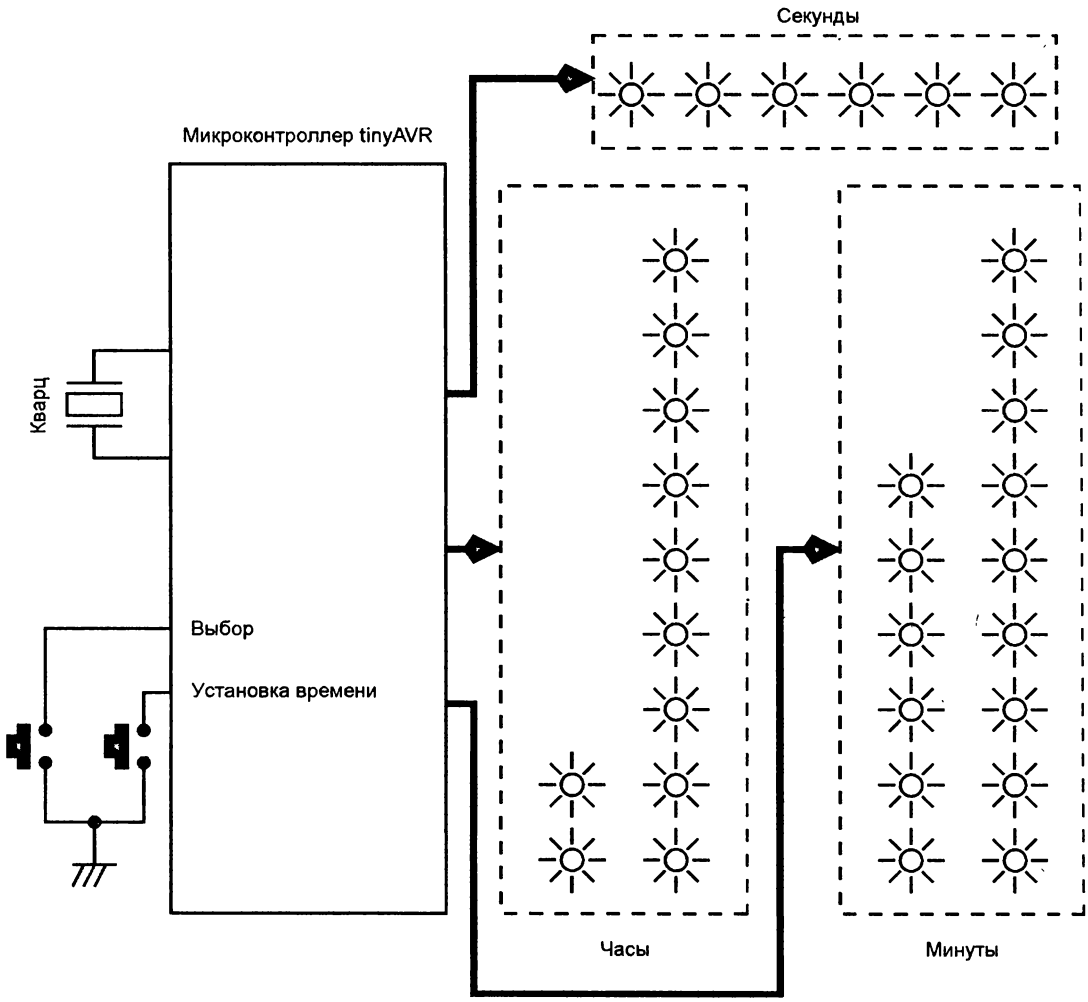


Рис. 3.41. Блок-схема часов

Эти часы могут показывать время с точностью до одной секунды. В них три блока светодиодов: для часов, минут и секунд. Число светодиодов в каждом блоке разное. Блок для индикации часов имеет два столбца: старшая и младшая цифра часов. Наши часы показывают время в 24-часовом формате, поэтому значения могут быть от 0 до 23. То есть в качестве старшей цифры часов могут светиться либо один, либо два светодиода (либо ни одного). В качестве младшей цифры часов мо-

жет светиться от нуля до девяти светодиодов. Точно так же (двумя блоками) отображаются и минуты. Шесть светодиодов отображают секунды. Мы назвали эти часы забавными потому, что светодиоды, отображающие часы и минуты переключаются случайным образом каждые пять секунд. Например, если время 15:35, то будет гореть любой светодиод старшей цифры часов. Будут светиться также любые пять светодиодов (из девяти) в младшей цифре часов. Через пять секунд загорятся другие пять светодиодов (из девяти). Таким образом, "огоньки" на часах будут все время меняться. Однако тот, кто знает принцип отображения времени на этих часах, сможет правильно понять показания.

На рис. 3.42 часы отображают время 15:35 (двумя разными способами).

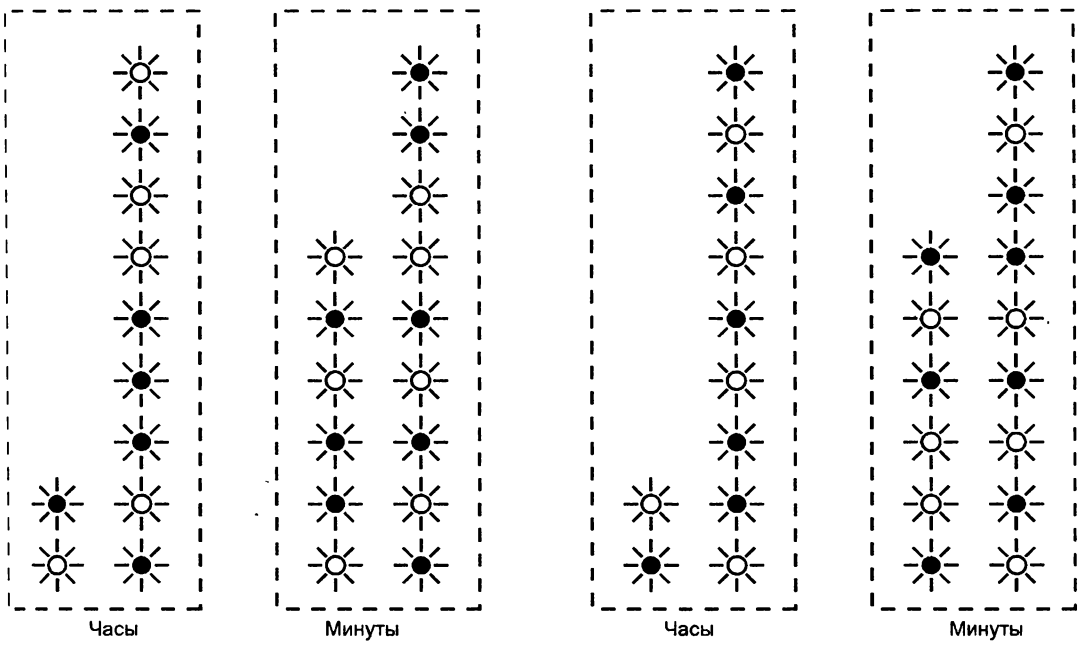


Рис. 3.42. Забавные часы показывают время 15:35

Спецификация проекта

Цель этого проекта — создать забавные часы, для считывания времени с которых нужно знать их секрет. Время (часы, минуты и секунды) отображается при помощи цветных светодиодов. Чтобы часы были точными, мы решили тактировать микроконтроллер от внешнего кварца.

Для подключения такого большого количества светодиодов мы применили мультиплексирование методом Чарли. Однако мы разбили имеющиеся контакты ввода/вывода на три блока: три контакта для управления шестью светодиодами индикации секунд, четыре контакта для управления 11 светодиодами для часов, пять контактов для мультиплексирования (методом Чарли) 14 светодиодов для минут.

Предусмотрены два переключателя для установки времени. Вторичный источник питания — линейный стабилизатор с низким падением напряжения LM2940, первичный источник — батареи.

Описание устройства

На рис. 3.43 показана принципиальная схема проекта. Поскольку есть стабилизатор напряжения LM2940 на 5 В, то входное напряжение может варьироваться от примерно 6 до 20 В. D1 — это диод Шоттки (1N5819), используемый в качестве защитного (это уже объяснялось ранее). Конденсатор C4 служит для фильтрации выбросов и нежелательных помех источника питания, а C3 подключен к выходу LM2940. Конденсаторы C1 и C2 припаяны около контактов питания микроконтроллера (чтобы дополнительно развязать возникающие в схеме помехи). Микроконтроллер — ATtiny261. Светодиоды мультиплексируются методом Чарли в трех группах: HOURS, MINUTES и SECONDS. Группа SECONDS имеет шесть светодиодов и отображает секунды в двоичном формате. LED6 — младший разряд, а LED1 — старший. Группа MINUTES состоит из 14 светодиодов. Светодиоды от LED7 до LED11 представляют десятки минут (от 0 до 5). Светодиоды от LED12 до LED20 представляют единицы минут.

В группе HOURS 11 светодиодов. Светодиоды LED21 и LED22 представляют десятки часов, а светодиоды от LED23 до LED31 — единицы часов. Рекомендуем для разных цифр выбрать светодиоды различных цветов.

Резисторы с R1 по R12 по 50 Ом ограничивают ток через светодиоды. Через R13 на контакт RESET микроконтроллера подается напряжение питания Vcc. Q1 — это кварц частотой 1,8432 МГц для тактирования микроконтроллера. В нашем устройстве микроконтроллер отсчитывает время, а для этого нужен очень точный генератор. Имеющийся внутри микроконтроллера RC-генератор недостаточно стабилен, поэтому требуется кварц.

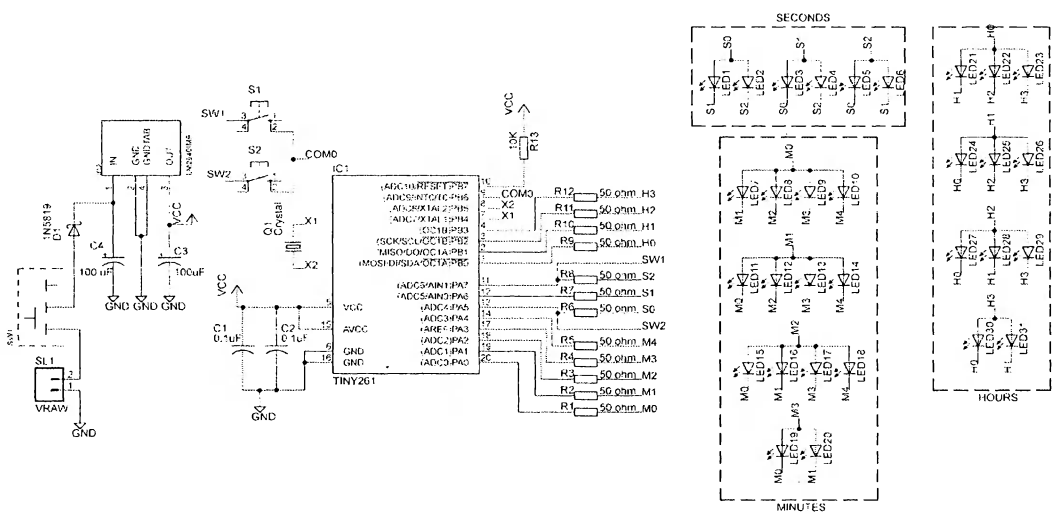


Рис. 3.43. Принципиальная схема часов

Две кнопки для установки времени подключены не так, как в предыдущих проектах. ATtiny261 имеет 15 контактов ввода/вывода (не считая сброса). Из них 12 контактов использованы для светодиодов, а два — для кварца. У нас остается один контакт и две кнопки. Однако мы знаем, что выходы, управляющие секундами, периодически находятся либо в состоянии 0, либо в состоянии 1. Более того, при мультиплексировании методом Чарли в любой момент включен только один светодиод (это уже объяснялось ранее). Первые контакты кнопок соединены с входом COM0 микроконтроллера, а вторые контакты кнопок подключены к выходам, управляющим светодиодами секунд. Если логический 0 присутствует на SW1, то изменение логического уровня COM0 рассматривается как нажатие кнопки S1, а если логический 0 присутствует на SW2, то изменение логического уровня COM0 рассматривается как нажатие кнопки S2.

Точное время отсчитывается при помощи таймеров. Кнопки S1 и S2 предназначены для коррекции времени.

Конструкция

Компоновку платы в программе EAGLE (вместе с принципиальной схемой) можно скачать по адресу: **www.avrgenius.com/tinyavr1**.

Плата односторонняя (на стороне компонентов всего несколько перемычек). Собранное устройство показано на рис. 3.44.

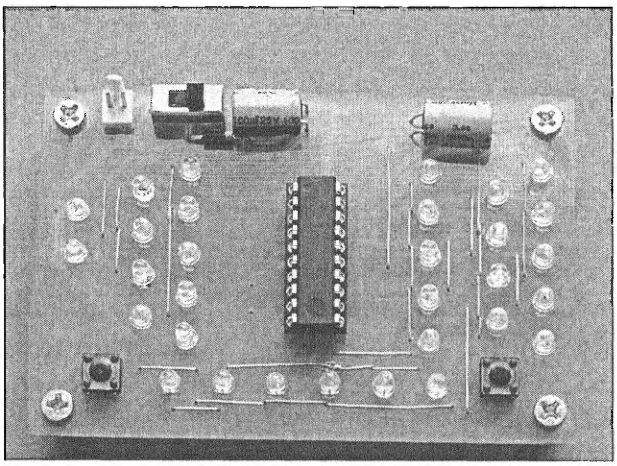


Рис. 3.44. Забавные часы в сборе

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по адресу: **www.avrgenius.com/tinyavr1**.

Программа выполняется на тактовой частоте 1,8432 МГц, которая поступает с внешнего кварца (поэтому нужно настроить fuse-биты). Поясним самые важные фрагменты кода.

Листинг 3.8

```
if((PORTA&0x80)==0)&&((DDRA&0x80)==0x80))
   //Если PA7 объявлен как выход и равен нулю
{
 if(!(PINB&0x40))
 {
  switch_1_pressed = 1;
 }
 if(((PINB&0x40)==0x40)&&(switch_1_pressed==1))
 {
  switch_1_released = 1;
 }
}

if(!(PORTA&0x20)&&(DDRA&0x20)==0x20)
   //Если PA5 объявлен как выход и равен нулю
{
 if(!(PINB&0x40))
 {
  switch_2_pressed = 1;
 }
 if(((PINB&0x40)==0x40)&&(switch_2_pressed==1))
 {
  switch_2_released = 1;
 }
}
```

Листинг 3.8 — часть процедуры обработки прерывания по переполнению таймера Timer0, которая используется для чтения переключателей. Для кнопки 1 сначала проверяется, равен ли нулю уровень на контакте PA7. Если да, то считывается PB6 (для обнаружения нажатия или отпускания кнопки). Если PB6 равен нулю, то `switch_1_pressed` получает значение `1` (чтобы указать состояние нажатия кнопки). Если PB6 равен единице и `switch_1_pressed=1`, значит произошло нажатие и отпускание кнопки 1 и переменной `switch_1_released` присваивается значение `1`. Основная программа выполняет соответствующую функцию. Аналогично все происходит и для кнопки 2 (когда на PA5 выставляется нуль).

Листинг 3.9

```
void display(void)
{
 display_on = 0;
```

```
u08 hour_ten,hour_one,min_ten,min_one;
hour_one = hour%10;
hour_ten = hour/10;
min_one = min%10;
min_ten = min/10;
//отобразить секунды
for(u08 h = 0;h<=5;h++)
{
 secondled[5-h] = ((sec&(1<<h)) == (1<<h));
}
if(display_on1==1)
  //время изменить случайную индикацию
{
 display_on1=0;
//Выключить все светодиоды часов и минут
 for(u08 o = 0;o<9;o++)
 {
  minuteled[o+5]=0;
    //сбросить цифру десятков минут
  hourled[o+2] = 0;
    //сбросить цифру десятков часов
  if(o<5)
  minuteled[o] = 0;
    //сбросить цифру единиц минут
  if(o<2)
  hourled[o] = 0;
    //сбросить цифру единиц часов
 }

 //отобразить цифру десятков часов
 for(u08 o = 0;o<hour_ten;o++)
 {
  //сгенерировать случайное число от 0 до 1
  random = TCNT0;
  random = random%2;
  while(hourled[random] == 1)
  {
   random++;
   if(random==2) random = 0;
  }
  hourled[random] = 1;
 }
```

```
//отобразить цифру единиц часов
 for (u08 o = 0;o<hour_one;o++)
{
 //сгенерировать случайное число от 2 до 10
 random = TCNT0;
 random = random%9+2;
 while(hourled[random] == 1)
 {
  random++;
  if(random == 11) random = 2;
 }
 hourled[random] = 1;
}

//отобразить цифру десятков минут
for(u08 o = 0;o<min_ten;o++)
{
 //сгенерировать случайное число от 0 до 4
 random = TCNT0;
 random = random%5;
 while(minuteled[random] == 1)
 {
  random++;
  if(random == 5) random = 0;
 }
 minuteled[random] = 1;
}

//отобразить цифру единиц минут
for(u08 o = 0;o<min_one;o++)
{
 //сгенерировать случайные числа от 5 до 13
 random = TCNT0;
 random = random%9+5;
 while(minuteled[random] == 1)
 {
  random++;
  if(random == 14) random = 5;
 }
 minuteled[random] = 1;
}
}
}
```

Функция отображения (листинг 3.9) — самый важный компонент исходного кода. Она вызывается каждую секунду для обновления светодиодов секунд. Однако если `display_on1=1`, то это означает, что нынешнее значение секунд кратно пяти и что нужно сгенерировать новое случайное отображение минут и часов. Значение `timer0` используется для вычисления случайных чисел. Код был написан так, чтобы в индикации разных цифр не было одинаковых случайных чисел.

Остальная часть кода отсчитывает время и реагирует на нажатие кнопок пользователем.

Работа устройства

В этом проекте нет требования сохранять отсчет времени, поэтому при каждом включении часов индикация начинается с 00:00. Вы можете изменить показания минут и секунд, нажимая кнопки S1 и S2. Обратите внимание, что кнопки не будут реагировать до тех пор, пока на подключенных к ним контактах ввода/вывода не появятся единицы. Время отображается в 24-часовом формате.

Проект 12. Разноцветные игральные кости

Если вы играете в настольные игры, то вам нужны кости. В этом проекте показано, как создать электронные кости, которые не только выдают случайное число при каждом нажатии кнопки, но и показывают его произвольным цветом. Это снова достигается при помощи RGB-светодиодов. Однако в отличие от устройства смешивания цветов или лампы для настроения (где для управления интенсивностью

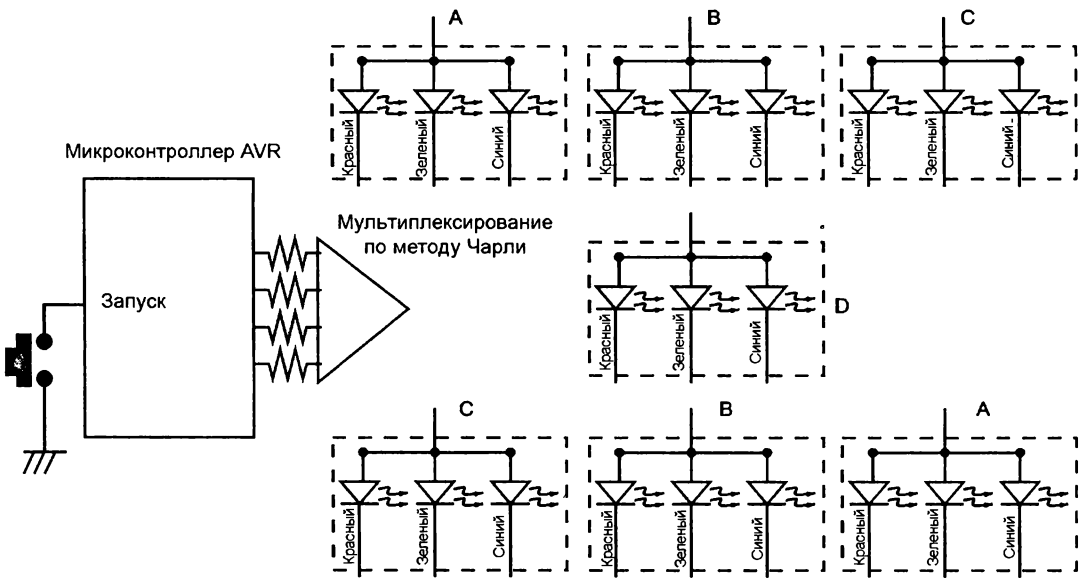


Рис. 3.45. Блок-схема устройства

красной, зеленой и синей составляющих цвета применяется широтно-импульсная модуляция, обеспечивающая множество оттенков), в данном проекте возможны только три первичных (красный, зеленый и синий) и три вторичных цвета (желтый, оранжевый и пурпурный). Блок-схема устройства показана на рис. 3.45.

Светодиоды организованы в обычную точечную матрицу. Кнопка нажимается и отпускается, при этом генерируется случайное число и загораются светодиоды. Их цвет выбирается случайным образом из шести возможных.

Спецификация проекта

Цель проекта — создать электронные игральные кости. Вместо точек на костях используются светодиоды. При каждом нажатии кнопки загораются светодиоды, показывающие число очков. Сами светодиоды также будут светиться разным цветом. Устройство питается от внешнего стабилизированного источника (хотя можно соединить последовательно четыре щелочные батареи по 1,5 В или четыре аккумулятора по 1,2 В.

Описание устройства

На рис. 3.46 показана принципиальная схема устройства. Стабилизатор напряжения LM3840 выдает 5 В, поэтому входное напряжение может варьироваться от 6 до 20 В. D1 — это диод Шоттки (1N5819) для защиты (как уже объяснялось ранее). Конденсатор C1 фильтрует выбросы и нежелательные помехи источника питания. C2 подключен к выходу LM3840. Микроконтроллер — ATtiny13. При помощи метода Чарли выполнено мультиплексирование четырех групп RGB-светодиодов. Всего имеется 12 светодиодов, управляемых четырьмя контактами ввода/вывода.

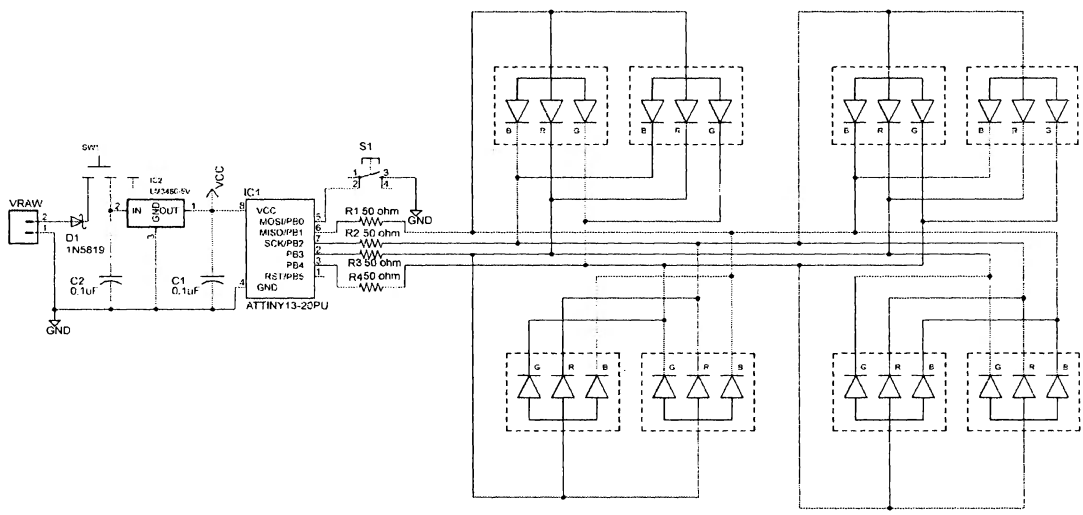


Рис. 3.46. RGB-кости: принципиальная схема

В каждой группе по два подключенных параллельно светодиода, которые включаются одновременно. Сама кость представлена семью точками. Для этого требуется три группы и еще один светодиод, но для того, чтобы выровнять ток через светодиоды, параллельно светодиоду центральной точки подключен "холостой" светодиод. Он смещен от центра и закрыт (чтобы его свет не был виден). Кнопка S1 предназначена для изменения числа и цвета светодиодов. Нажатие кнопки S1 изменяет число, а отпускание — цвет. Резисторы R1 и R4 ограничивают ток.

В программе работает счетчик и при каждом нажатии кнопки его значение считывается и генерируется новое число, а при каждом отпускании кнопки опять считывается значение счетчика и генерируется новый цвет. Всего имеется шесть цветов. Три первичных цвета формируются включением нужного числа RGB-светодиодов. Три вторичных цвета генерируются включением любых двух комбинаций нужных светодиодов. ШИМ в этом устройстве нет.

Конструкция

Компоновку платы в программе EAGLE (и ее принципиальную схему) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Плата односторонняя (на стороне компонентов есть всего несколько перемычек). Стороны печатной платы показаны на рис. 3.47 и 3.48.

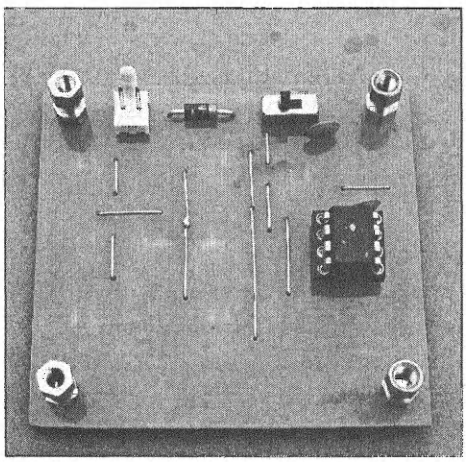


Рис. 3.47. Печатная плата (сторона компонентов)

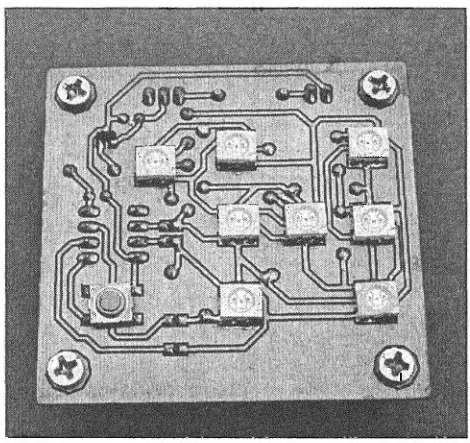


Рис. 3.48. Печатная плата (сторона печатных проводников)

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 9,6 МГц. Контроллер программируется при помощи STK500 в режиме ISP. Листинг 3.10 иллюстрирует наиболее важный фрагмент кода.

Листинг 3.10

```
//Процедура для нажатия кнопки
void switchpressed(void)
{
 unsigned char b = 1;
 while((PINB&(1<<0)))
   //ожидание нажатия кнопки
 {
  if(b==6)
  b = 1;
  else
  b = b+1;
 }
 _delay_ms(20); //для предотвращения дребезга
 statusonoff[1]=(b==4)||(b==5)||(b==6);
 statusonoff[2]=(b==4)||(b==5)||(b==6)||(b==3);
 statusonoff[3]=(b==2)||(b==6);
 statusonoff[4]=(b==1)||(b==3)||(b==5);
}

//процедура для отпускания кнопки
void switchreleased(void)
{
 unsigned char b = 1;
 while(!(PINB&(1<<0)))
   //ожидание отпускания кнопки
 {
  if(b==6)
  b = 1;
  else
  b = b+1;
 }
 _delay_ms(20); //для предотвращения дребезга
 coloronoff[0]=(b==2)||(b==4)||(b==6);
 coloronoff[1]=(b==3)||(b==4)||(b==5);
 coloronoff[2]=(b==1)||(b==6)||(b==5);
}
```

Функция `switchpressed` ждет нажатия кнопки, а функция `switchreleased` ждет отпускания уже нажатой кнопки. Обе функции имеют внутренний счетчик от 1 до 6. При нажатии или отпускании его значение увеличивается. В зависимости от этого значения загораются светодиоды.

Проект 13. Игра "крестики-нолики"

Это современная версия классической игры в крестики-нолики. Крестики и нолики представлены с помощью выбираемых пользователем цветов. Предусмотрено несколько цветов и каждый игрок может выбрать цвет для нолика (или крестика). После выбора цветов кнопки позволяют поставить крестик (или нолик) в любое незанятое место на поле размером 3×3. В устройстве девять RGB-светодиодов. В этом проекте светодиоды управляются путем обычного мультиплексирования. На рис. 3.49 изображена блок-схема устройства. Для подачи сигнала, что пользователь выиграл, есть зуммер. Микроконтроллер распознает, когда три светодиода одного цвета окажутся в строке, столбце или по диагонали и останавливает игру. После завершения (выигрыша, проигрыша или ничьей) можно сыграть новую партию.

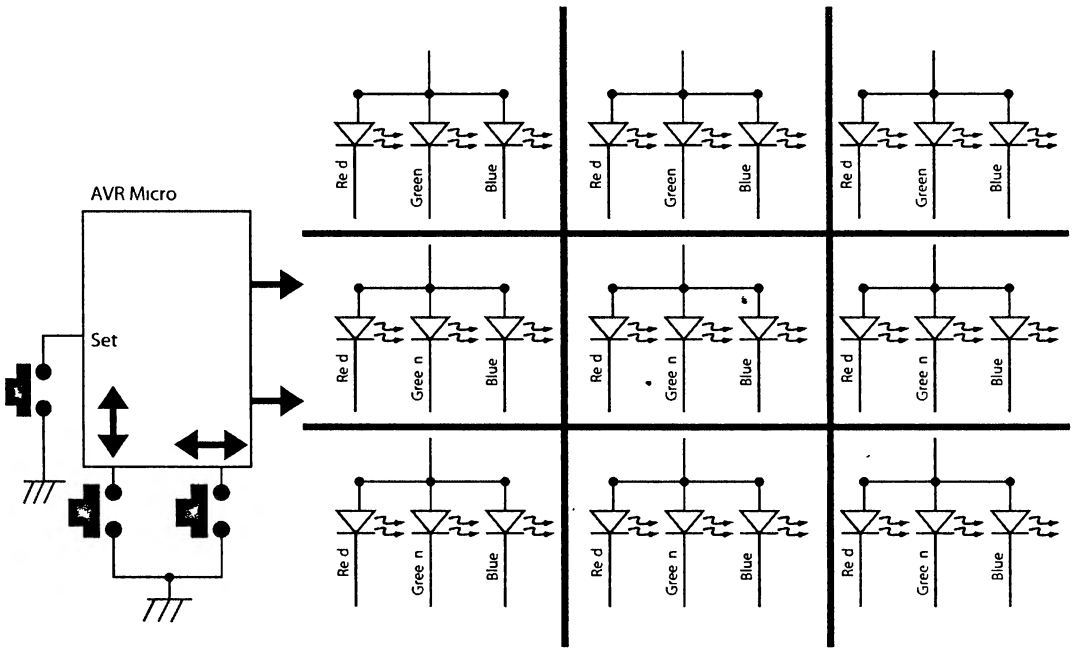


Рис. 3.49. Блок-схема игры "крестики-нолики"

Спецификация проекта

Цель проекта — создать современную версию классической игры в крестики-нолики. В устройстве применено десять RGB-светодиодов. Один светодиод указывает, чья очередь делать ход. Остальные девять светодиодов организованы в матрицу 3×3. Для управления размещением очередного знака есть несколько кнопок. Предусмотрен также зуммер, сигнализирующий о победе. Устройство работает от внешних батарей, в качестве вторичного источника встроен стабилизатор напряжения.

Описание устройства

До настоящего момента во всех проектах этой главы использовалось мультиплексирование по методу Чарли (чтобы управлять большим числом светодиодов при помощи ограниченного числа выводов). Здесь реализовано обычное мультиплексирование. Принципиальная схема проекта изображена на рис. 3.50. Благодаря стабилизатору напряжения LM2940 входное напряжение может варьироваться от 6 до 20 В. D1 — это диод Шоттки (1N5819), который работает как защитный (как уже объяснялось ранее). Конденсатор C3 фильтрует выбросы и нежелательные помехи источника питания. C2 подключен к выходу LM2940. C1 и C4 распаяны около контактов питания микроконтроллера (для дальнейшего развязывания помех, возникающих в схеме). Микроконтроллер — ATtiny861.

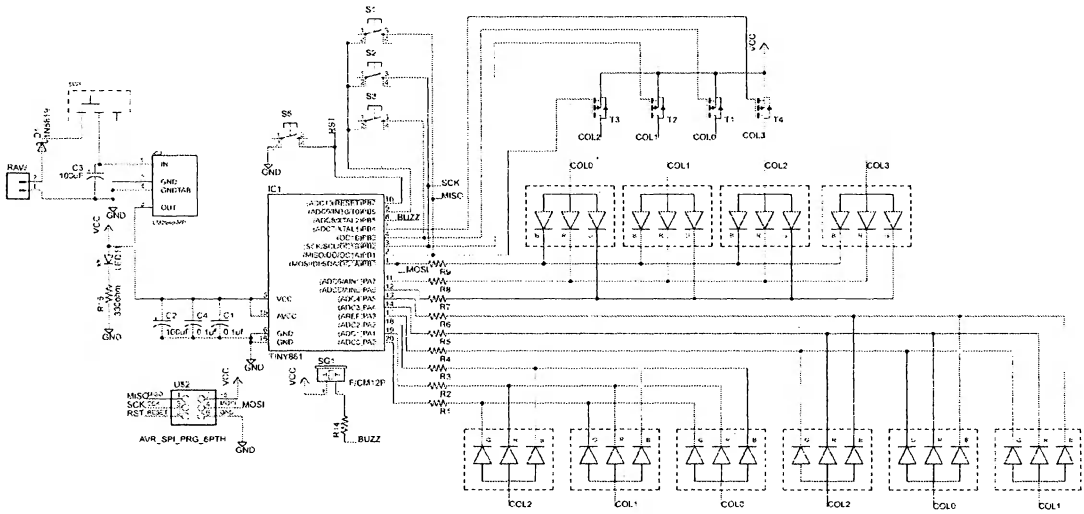


Рис. 3.50. Принципиальная схема игры "крестики-нолики"

Светодиоды объединены в четыре столбца и мультиплексированы. COL0 — это первый столбец матрицы размером 3×3, COL1 — второй, COL3 — третий. В каждом столбце по три RGB-светодиода. В столбце COL4 есть только один RGB-светодиод, который во время игры обозначает очередность ходов. T1, T2, T3 и T4 — это транзисторы *n*-канальные MOSFET (NDS355), работающие как источники тока для четырех столбцов. Есть три кнопки, подключенные как в проекте забавных часов. Они мультиплексированы со столбцами. SG1 — это зуммер, подающий сигнал о выигрыше.

Код этого проекта самый сложный из всех описанных до настоящего момента. Он обеспечивает программную широтно-импульсную модуляцию для всех светодиодов. Для каждого цветового компонента есть девять уровней интенсивности, следовательно, можно сгенерировать 9×9×9 цветов. Из этих цветов в программе записаны только 16 самых контрастных. После включения схемы игроки выбирают цвета, которыми они будут играть. После этого они ходят при помощи кнопок. Программа после каждого хода проверяет условия выигрыша. Если игрок выигры-

вает, то три выигравших символа начинают мигать и в течение полсекунды звучит звуковой сигнал. После нажатия на копку перезапуска игра продолжается теми же цветами, но первый ход предоставляется второму игроку (и т. д.).

Конструкция

Компоновку платы в EAGLE (вместе с принципиальной схемой) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Плата односторонняя (на стороне компонентов всего несколько перемычек). Стороны печатной платы показаны на рис. 3.51 и 3.52.

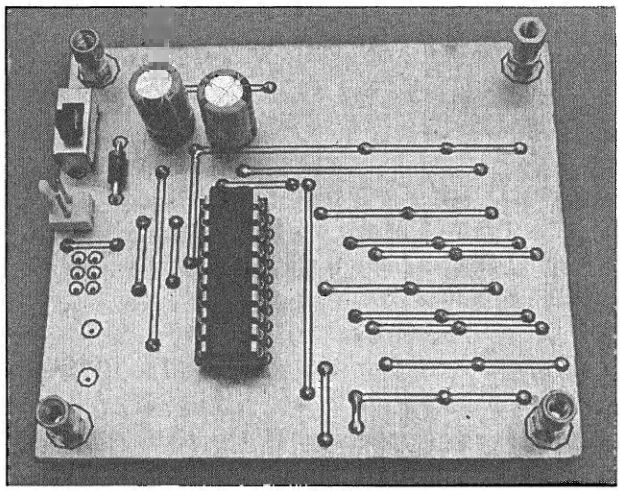


Рис. 3.51. Печатная плата (сторона компонентов)

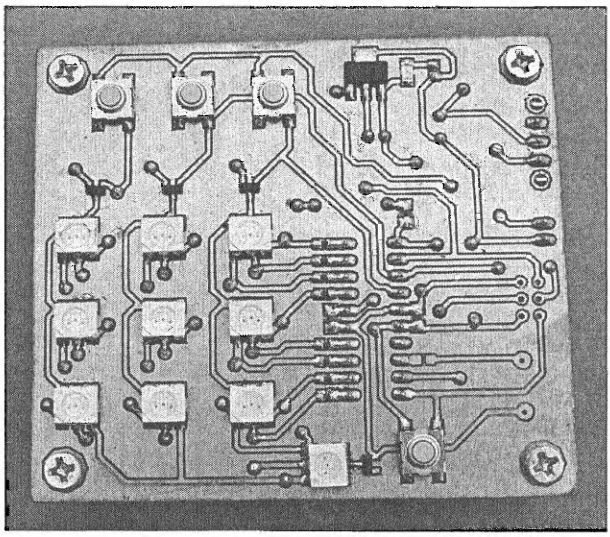


Рис. 3.52. Печатная плата (сторона печатных проводников)

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота — 8 МГц. Контроллер программируется при помощи STK500 в режиме программирования ISP. Листинг 3.11 содержит самый важный фрагмент кода.

Листинг 3.11

```
while(1)
{
 playerturn(b1,r1,g1,1);
 checkwin();
 playerturn(b2,r2,g2,2);
 checkwin();
}
```

Это главный бесконечный цикл программы. Первые три аргумента функции `playerturn` указывают интенсивность синего, красного и зеленого компонентов. Четвертый аргумент обозначает номер игрока. Функция `playerturn` отвечает за фиксацию выбранного игроком цвета. Функция `checkwin` проверяет условия выигрыша для обоих игроков (по правилу "магического" квадрата: суммы по всем его строкам, столбцам и диагоналям одинаковы). В нашем коде каждой позиции присваивается определенный номер. Когда игрок выставляет свой цвет на светодиоде, соответствующий номер заносится в буфер игрока. Если игрок занял светодиоды, сумма трех из которых равна магическому квадрату, значит, данный игрок выиграл. Перед этим циклом функция `start` позволяет игрокам выбрать цвет для игры.

Заключение

В этой главе мы изучили новые конфигурации светодиодов и управление ими способами мультиплексирования. Мы подробно обсудили также управление интенсивностью свечения светодиодов с помощью аппаратной ШИМ микроконтроллеров tinyAVR. В некоторых устройствах использовались восьмиконтактные микроконтроллеры, которые управляли семисегментными индикаторами в две с половиной цифры при помощи мультиплексирования по методу Чарли. Мы рассмотрели несколько проектов, которые можно модифицировать для других приложений. В следующей главе мы опишем несколько проектов с еще одним популярным устройством — графическим жидкокристаллическим дисплеем.

Глава 4

Проекты с графическими жидкокристаллическими дисплеями

В предыдущей главе мы рассмотрели применение светодиодных дисплеев. В этой главе мы расскажем про альтернативные (но такие же популярные) устройства — жидкокристаллические дисплеи. Между светодиодными и жидкокристаллическими дисплеями есть некоторые различия. Первое состоит в том, что в отличие от светодиодов, отдельных жидкокристаллических элементов не существует. Кроме того, светодиод сам излучает свет, а жидкокристаллический дисплей (LCD) лишь изменяет интенсивность окружающего света. Из-за этого светодиод лучше подходит в условиях плохого освещения, а LCD — при ярком дневном свете. Разработчик должен сделать выбор между светодиодом и ЖК-дисплеем. Если применяется LCD, и устройство будет работать в условиях малой освещенности, то необходимо предусмотреть подсветку. Интересно, что такую подсветку обычно делают на основе светодиодов. На рис. 4.1 показаны некоторые образцы жидкокристаллических дисплеев.

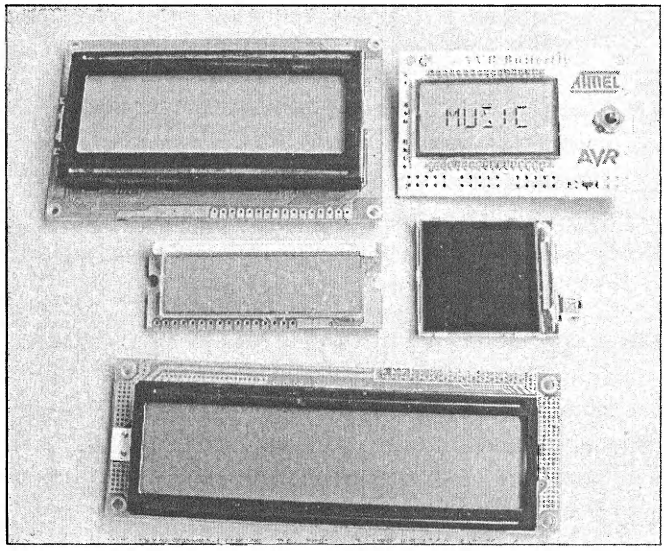


Рис. 4.1. Образцы жидкокристаллических дисплеев

ЖК-дисплеи бывают трех типов: отражающие, пропускающие и комбинированные. В следующем разделе мы рассмотрим работу этих устройств и воспользуемся ими для реализации нескольких проектов.

Принцип действия ЖК-дисплея

Как уже упоминалось, существуют три типа LCD: отражающие, пропускающие и комбинированные. Структура отражающего дисплея изображена на рис. 4.2. Снизу расположено зеркало, которое отражает падающий свет. Между поляризаторами (B и F) и двумя электродами (C и E) находится слой жидких кристаллов (D). При отсутствии напряжения на электродах жидкий кристалл пропускает свет, который отражается от зеркала и возвращается обратно. После приложения к электродам электрического потенциала (рис. 4.3) жидкие кристаллы ориентируются таким образом, что перекрывают прохождение света и наблюдателю становится виден черный квадрат, нанесенный на верхнем электроде. Отражающему дисплею для работы нужно внешнее освещение (или подсветка). Для подсветки над ЖК-дисплеем или рядом с ним размещают белые светодиоды, обеспечивающие равномерное освещение.

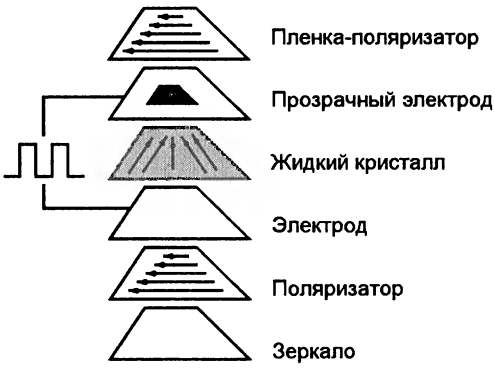


Рис. 4.2. Слои отражающего LCD в нормальном состоянии кристаллов

Рис. 4.3. Слои отражающего LCD в поляризованном состоянии кристаллов

В пропускающем ЖК-дисплее нижний слой прозрачен и зеркала нет. Такой дисплей нуждается в подсветке сзади. Пропускающие дисплеи очень хорошо работают в условиях плохого внешнего освещения (при включенной внутренней подсветке). Комбинированные дисплеи имеют частично прозрачные нижние электроды. У них также есть источник подсветки, который можно включить при недостатке внешнего освещения.

На рис. 4.4 приведена блок-схема стандартного контроллера LCD. Кроме матрицы из жидких кристаллов контроллер имеет память для хранения информации,

отображаемой на дисплее. Число битов памяти равно числу жидкокристаллических элементов. Для облегчения доступа память организована по байтам. Для управления жидким кристаллом контроллеру нужно напряжение смещения в диапазоне от 5 до 10 В. Обычно в контроллер встроен генератор напряжения смещения VLCD. Обмен информацией с внешними устройствами осуществляется через интерфейс. Это может быть параллельная 8-разрядная шина или более экономный интерфейс (SPI либо I2C). Внешний микроконтроллер через интерфейс обмена отправляет данные для отображения. Контроллер LCD получает данные, сохраняет их в своей внутренней памяти и осуществляет отображение информации.

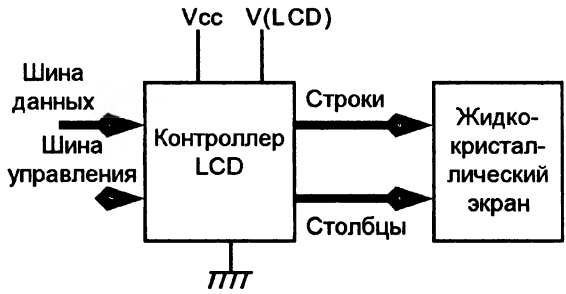


Рис. 4.4. Блок-схема контроллера LCD

Nokia 3310 — это весьма популярный мобильный телефон, поэтому многие его компоненты вполне доступны. Можно легко купить графический дисплей для такого телефона, хотя скорее всего не оригинальный, а сделанный в Китае. В следующем разделе мы опишем этот чрезвычайно популярный дисплей, который легко встроить в небольшие устройства.

Дисплей Nokia 3310

Это небольшой графический дисплей, подходящий для встраивания в различные устройства. Он довольно популярен, поскольку просто подключается к большинству микроконтроллеров и доступен. Размер самого дисплея 38×35 мм, размер активной зоны — 30×22 мм, разрешение — 84×48 пикселов. Дисплей поставляется с контроллером PCD8544, который управляет жидкокристаллической матрицей из 48 строк и 84 столбцов. Дисплей легко состыковать при помощи стандартного интерфейса SPI. Вот некоторые его особенности:

- Нужен только один внешний компонент — конденсатор емкостью от 1 до 10 мкФ между клеммами Vout и GND.
- Диапазон напряжения питания от 2,7 до 3,3 В.
- Низкое потребление энергии (хорошо подходит для систем с питанием от батарей).
- Диапазон температур от –25 до +70 °C.

Полностью спецификацию на дисплей можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Сопряжение дисплея Nokia 3310

Дисплей Nokia 3310 работает через интерфейс SPI, который имеется во многих микроконтроллерах tinyAVR, но отсутствует в микроконтроллерах других серий. Часто SPI занят обменом с другими устройствами. Однако это не означает, что данный дисплей нельзя использовать с этими устройствами. Мы можем состыковать такой дисплей при помощи программной реализации интерфейса SPI через любой вывод микроконтроллера. Для этой цели нам потребуется как минимум четыре контакта.

Вот назначение контактов дисплея Nokia 3310 (рис. 4.5):

- V_{CC} — входное напряжение питания, подаваемое на внутренний стабилизатор (от 2,7 до 3,3 В).
- SCK — входной тактовый сигнал (от 0 до 4,0 МГц). Подключается к контакту ввода/вывода микроконтроллера.
- SDI — вход последовательных данных. Подключается к контакту ввода/вывода микроконтроллера.
- D/C — вход выбора режима Data/Command. Подключается к контакту ввода/вывода микроконтроллера.
- SCE — выбор кристалла. Этот контакт может быть подключен к контакту ввода/вывода микроконтроллера или заземлен (чтобы дисплей был всегда активизирован).
- GND — общая шина.
- V_{OUT} — VLCD. Этот контакт подключен к GND через конденсатор емкостью 10 мкФ.
- RST — контакт сброса контроллера PCD8455. Подключается к контакту ввода/вывода микроконтроллера.

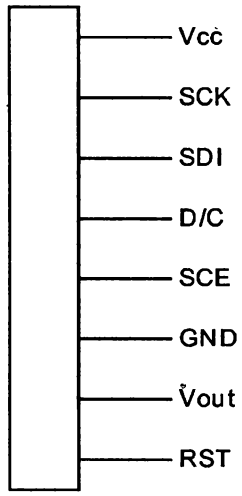


Рис. 4.5. Назначение контактов дисплея Nokia 3310

Функциональное описание контроллера PCD8455

PCD8455 — это контроллер, который управляет графическим дисплеем из 48 строк и 84 столбцов. Все требуемые для дисплея функции содержатся в одном чипе (в том числе генерирование необходимых напряжений), что минимизирует количество необходимых внешних компонентов и потребление энергии. На рис. 4.6 изображена блок-схема контроллера PCD8455.

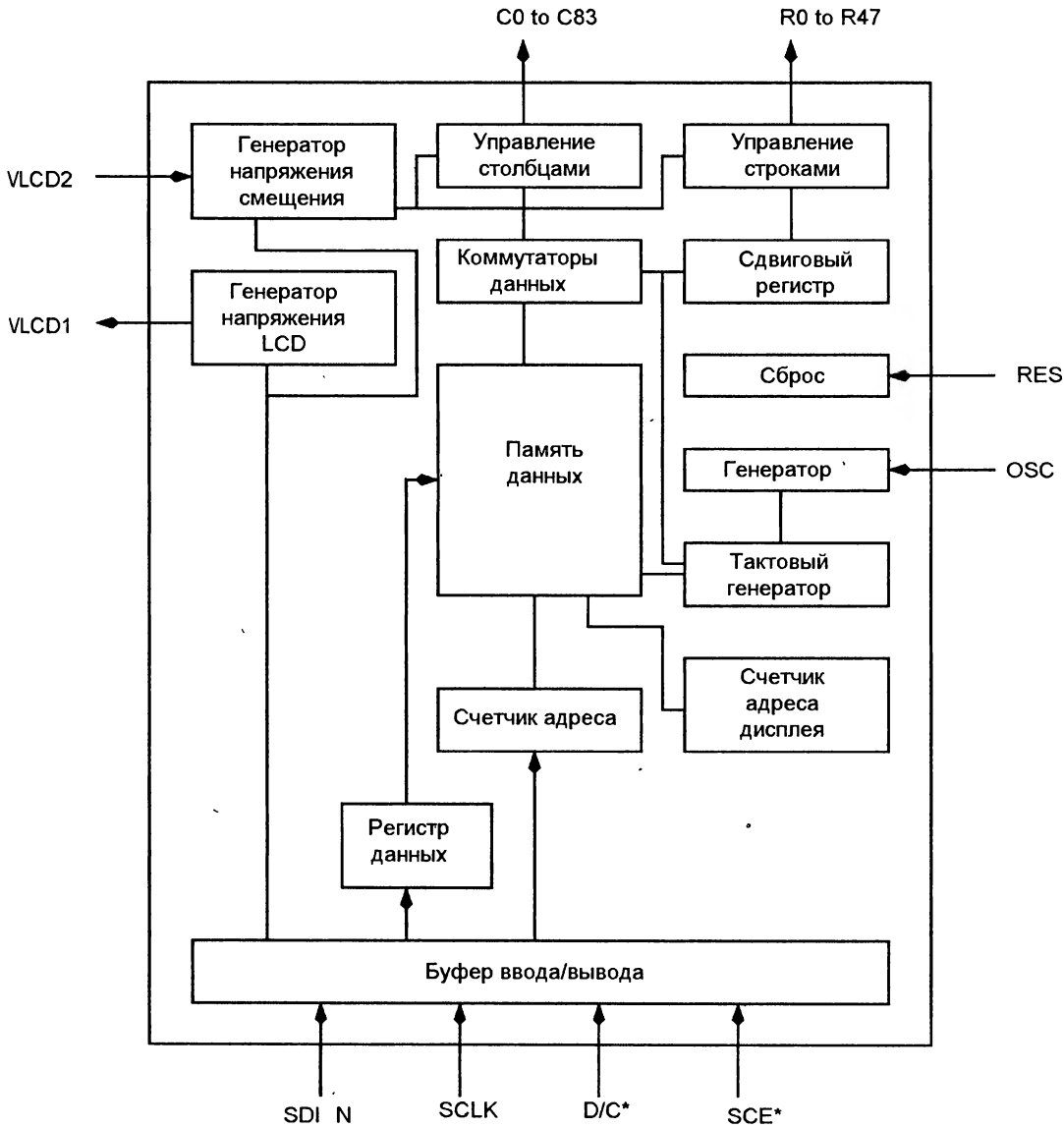


Рис. 4.6. Блок-схема контроллера PCD8455

Вот некоторые важные особенности этого контроллера:

- Счетчик адреса — содержит адрес в памяти отображаемых данных (для адресации столбца из 8 пикселов). Адреса по оси *X* (от 0 до 83) и адреса по оси *Y* (от 0 до 5) настраиваются отдельно.
- Память отображаемых данных — статическая память размером 48×84 бита, которая хранит отображаемые данные. Она разделена на шесть страниц по 84 байта (6×8×84 бита). Каждая страница адресуется одним адресом *Y*, а отдельные столбцы каждой страницы адресуются одним адресом *X*.
- Формат команд определяется двумя режимами — если на входе D/C присутствует низкий уровень (LOW), то текущий байт интерпретируется как байт команды; в противном случае (при высоком уровне — HIGH) текущий байт сохраняется как байт данных в памяти отображаемых данных.

  После каждого байта данных счетчик адреса автоматически увеличивается. Сигнал на входе D/C читается после передачи последнего бита (на восьмом импульсе SCLK). Если на входе SCE высокий уровень (HIGH), то сигнал SCLK игнорируется и выполняется инициализация последовательного интерфейса. Данные (SDIN) отсчитываются при положительном фронте тактового импульса. Если напряжение на SCE остается низким даже после передачи последнего бита байта команды/данных, то контроллер настраивается на прием следующего байта (предполагая, что входящий бит является седьмым битом следующего байта). Импульс RESET сбрасывает контроллер (прерывая передачу и очищая все регистры). Если SCE находится в состоянии LOW после положительного фронта импульса RESET, то контроллер готов принимать следующий байт.

Программа управления LCD

Необходимую для сопряжения с LCD библиотеку на языке C можно скачать по ссылке: **www.avrgenius.com/tinyavr1**. Эти функции интегрированы с исходными кодами всех проектов данной главы.

Листинг 4.1

```
void clockdata(char bits_in)
{
 int bitcnt;
 for (bitcnt=8; bitcnt>0; bitcnt--)
 {
  LCD_PORT = LCD_PORT&(~(1<<SCK));
  // Установить Clock Idle в уровень LOW.
  if ((bits_in&0x80)==0x80) {LCD_PORT |=1<<SDIN;}
  else {LCD_PORT &= ~(1<<SDIN);}
  LCD_PORT |=1<<SCK;
// Данные тактируются по нарастающему фронту SCK.
  bits_in=bits_in<<1;
```

```
// Логический сдвиг данных на 1 бит влево.
 }
}
```

Процедура, приведенная в листинге 4.1, передает байт данных из микроконтроллера в контроллер LCD. Для передачи данных (по одному биту) организован цикл. Сначала SCK переводится в состояние LOW, а затем передаваемый бит выдается на контакт SDIN (подключенный к LCD). После установки бита SCK переводится в HIGH (в течение того времени, пока данные передаются по нарастающему фронту тактового импульса). Этот процесс для передачи байта повторяется в цикле восемь раз. Обратите внимание, что мы заземлили контакт SCE дисплея.

Листинг 4.2

```
void writecom(char command_in)
{
LCD_PORT = LCD_PORT&(~(1<<D_C));
  // Выбрать регистр Select Command.
clockdata(command_in);
  // Тактировать биты команды.
}
void writedata(char data_in)
{
LCD_PORT = LCD_PORT|(1<<D_C);
  //Выбрать регистр данных.
clockdata(data_in);
  // Тактировать биты данных.
}
```

Подпрограмма из листинга 4.2 используют функцию `clockdata` для передачи либо байта команды после установки режима D/C (в функции `writecom`), либо байта данных (в функции `writedata`). Теперь мы знаем, как передать байт данных на дисплей, но для применения дисплея нам нужно инициализировать его (в соответствии с описанной в спецификации процедурой). Это показано в листинге 4.3.

Листинг 4.3

```
void initlcd(void)
{
  LCD_DDR |= 1<<SCK|1<<SDIN|1<<D_C|1<<RESET;
  LCD_PORT = LCD_PORT|1<<RESET;
  LCD_PORT = LCD_PORT&(~(1<<RESET));
  _delay_ms(200);
  LCD_PORT = LCD_PORT|1<<RESET;
```

```
  writecom(0x21);
    // Активировать чип и H=1.
  writecom(0xD3);
    // Установить напряжение LCD примерно в 9V.
  writecom(0x13);
    // Настроить напряжение смещения.
  writecom(0x20);
    // Горизонтальная адресация и H=0.
  writecom(0x09);
    // Активизировать все сегменты.
  clearram();
    // Стереть все пикселы в памяти.
  writecom(0x08);
    // Очистить дисплей.
  writecom(0x0C);
    // Режим дисплея Normal.
}
```

В этой процедуре мы сначала объявляем все четыре контакта ввода/вывода как выходы (при помощи соответствующего регистра). Затем мы сбрасываем LCD импульсом RESET длительностью 200 мс. Это очищает все предыдущие настройки LCD. Затем LCD инициализируется под наши требования (при помощи посылки последовательности командных слов). Сначала на LCD активизируется расширенная система команд (H=1). Следующий командный байт устанавливает рабочее напряжение на LCD. При помощи этой функции можно программно настроить контраст LCD. Это делается посредством вычисления подлежащего передаче восьмибитового значения при помощи описанного в спецификации Nokia 3310 соотношения. Затем настраивается напряжение смещения — задается коэффициент мультиплексирования 1:48. Следующий командный байт служит для выбора горизонтального адреса и для выставления H=0 (чтобы использовать команды типа "установить адрес *X*" или "установить адрес *Y*"). Для активизации дисплея сначала посылается командный байт для включения всех сегментов дисплея; следующий байт посылается для очистки дисплея, а последний — для перевода его в нормальный режим. Эта процедура должна вызываться при необходимости перевода дисплея в нормальный режим.

Сбои, наблюдающиеся в некоторых дисплеях

В некоторых образцах дисплеев мы иногда сталкиваемся с проблемой искажения первой страницы на экране размером в 48×84 пиксела. Дисплей Nokia 3310 содержит шесть страниц (от нулевой до пятой), но самая верхняя страница (с адресом `Yaddress=0`) бывает искаженной. Видны только пять пикселов каждого столбца этой страницы. На нижней стороне появляется новая страница с тремя пикселами в каждом столбце. Эта новая страница адресуется при помощи `Yaddress=6`. Чтобы

устранить эту ошибку, нужно изменить инициализирующий код LCD. Если в функции `initlcd` после настройки напряжения смещения написать команду `writecom(0x45)`, то произойдет выравнивание страниц и смещение вверх на пять пикселов. После этого искаженная верхняя страница скроется, а нижняя страница станет полностью видна. В таком случае адреса `Yaddress` меняются от 1 до 6.

На рис. 4.7–4.9 показаны дисплей и клавиатура Nokia, печатная плата, а также окончательный вид припаянного на плату дисплея.

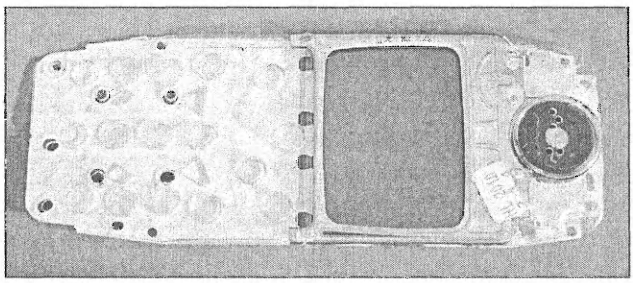


Рис. 4.7. Внешний вид дисплея Nokia 3310 с клавиатурой

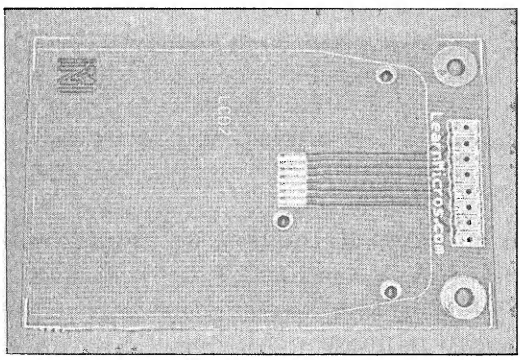


Рис. 4.8. Печатная плата дисплея Nokia 3310

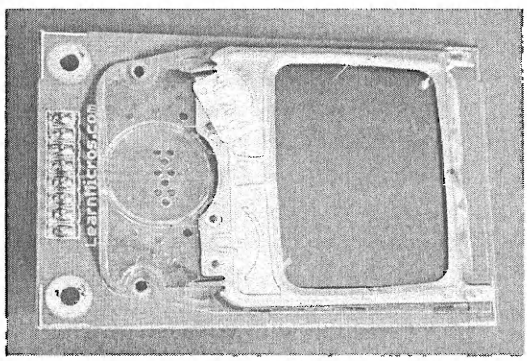


Рис. 4.9. Дисплей Nokia 3310 на печатной плате

Проект 14. Регистратор температуры

В этом проекте используется датчик температуры, который показывает на дисплее окружающую температуру в градусах Цельсия и Фаренгейта. Он также отображает минимальные и максимальные температуры, записанные регистратором. На рис. 4.10 приведена блок-схема проекта. Дисплей Nokia 3310 служит для отображения показаний. Кнопка на плате переключает режим индикации. В одном случае отображаются показания температуры (попеременно в градусах Цельсия и Фаренгейта), в другом — выводится график изменения температуры во времени. Устройство питается от батарей.

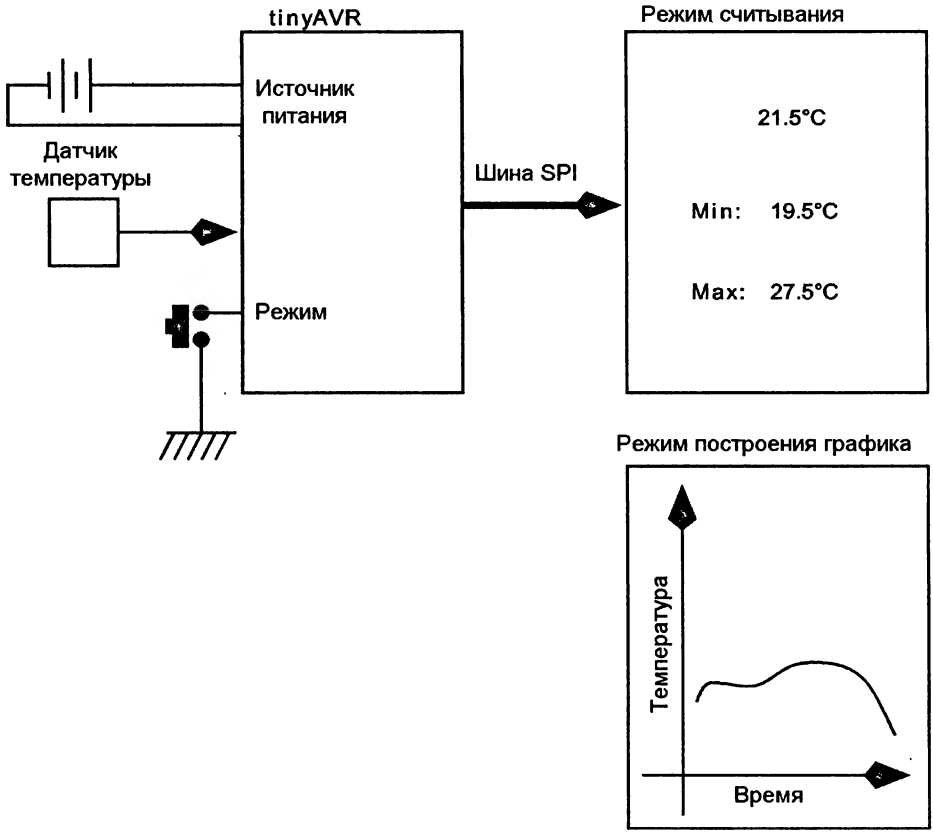


Рис. 4.10. Блок-схема регистратора температуры

Спецификации проекта

Цель проекта — создать систему регистрации температуры, которая будет показывать температуру окружающей среды в градусах Цельсия или Фаренгейта, а также ее минимальные и максимальные значения. Устройство также должно отображать температуру как функцию от времени. Чтобы переносить и устанавливать устройство в любом месте, следует предусмотреть питание от батарей.

Описание устройства

На рис. 4.11 изображена принципиальная схема устройства. Поскольку в проекте используется дисплей Nokia, то для него требуется напряжение питания от 2,7 до 3,3 В. Источник питания выполнен на основе повышающего преобразователя постоянного тока TPS61070, который выдает 3,3 В от одной батарейки в 1,5 В. Батарейка подключается к клеммам SL3. Поскольку защиты от неправильной полярности нет, следует тщательно следить за правильным подключением батарейки. Дисплей Nokia подключается при помощи шины SPI через разъем SL1.

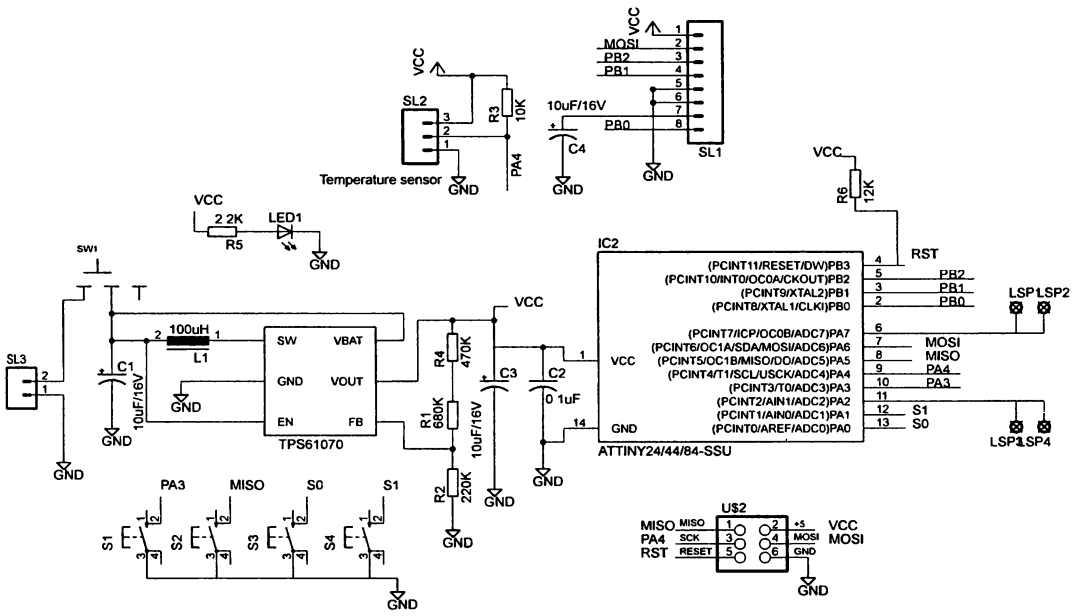


Рис. 4.11. Принципиальная схема регистратора температуры

Самый важный компонент системы — датчик температуры. Это может быть термистор, термопара или полупроводниковый датчик. Последний использовать проще всего.

Существуют разные полупроводниковые датчики температуры. Некоторые датчики выдают аналоговый сигнал (напряжение которого пропорционально температуре); другие датчики выдают цифровое значение в градусах Цельсия или Фаренгейта. Мы воспользовались температурным датчиком DS1820, который преобразует температуру в 9-разрядное число (представляющее температуру в градусах Цельсия или Фаренгейта). Точность отсчета температуры — 0,5 градуса Цельсия или 0,9 градуса Фаренгейта (в диапазоне от –55 до +125 °C или от –67 до +257 °F). Подробности смотрите в спецификации на DS1820, которую можно скачать с нашего Web-сайта по ссылке: **www.avrgenius.com/tinyavr1**.

Преобразованные данные можно считать с интерфейса датчика. На принципиальной схеме датчик DS1820 обозначен как SL2. На схеме показаны также четыре кнопки (S1–S4), но в данном проекте задействована только одна. Остальные кнопки потребуются в других проектах этой главы. В устройстве применен микроконтроллер Tiny44 в корпусе SMD с 14 контактами и 4 Кбайт памяти для программ. При включении питания или сбросе микроконтроллер инициализирует дисплей, опрашивает датчик DS1820 и отображает температуру (в градусах Цельсия или Фаренгейта) на дисплее. Он также хранит наблюдавшиеся минимальное и максимальное значения температуры. Пользователь может в любой момент нажать кнопку, и система переключится в другой режим, где на индикаторе вычерчиваются значения температуры как функции от времени. Система постоянно выполняет замеры температуры, но сохраняет только одно значение каждые десять минут (и вычерчивает график на дисплее). Предусмотрено хранение не более 40 значений,

поэтому изменения температуры отображаются за последние 400 минут. Данные, хранящиеся в буфере, постоянно сдвигаются (для размещения новых отсчетов), более старые значения стираются.

Конструкция

Компоновку платы в программе EAGLE (а также принципиальную схему) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Плата односторонняя (на стороне компонентов есть всего несколько перемычек). Стороны печатной платы показаны на рис. 4.12 и 4.13. Сначала следует смонтировать преобразователь напряжения и его навесные элементы. Паять микросхему TPS61070 нужно очень осторожно. Затем, до установки остальных компонентов необходимо протестировать напряжение на выходе TPS61070. Внешний вид дисплея регистратора в разных режимах показан на рис. 4.14 и 4.15.

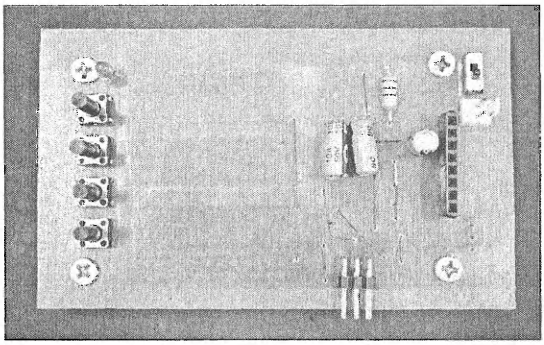


Рис. 4.12. Печатная плата регистратора (сторона компонентов)

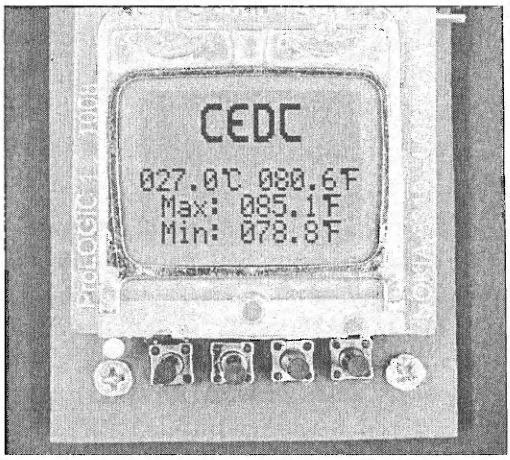


Рис. 4.14. Дисплей регистратора в режиме отсчета температуры

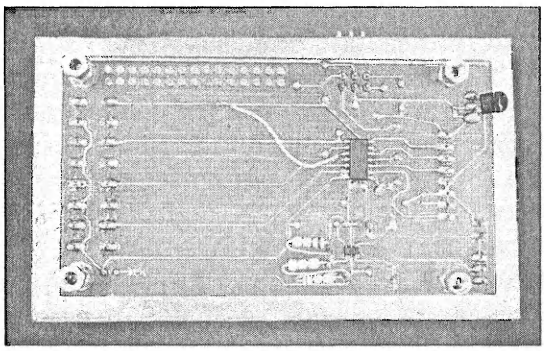


Рис. 4.13. Печатная плата регистратора (сторона печатных проводников)

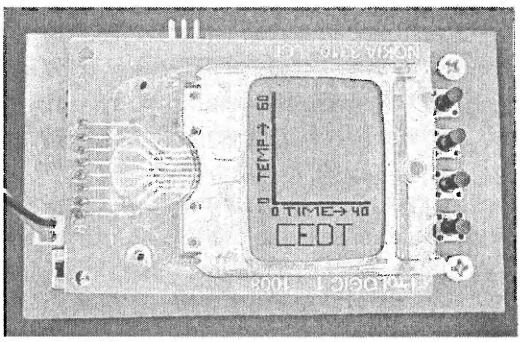


Рис. 4.15. Дисплей регистратора в режиме отображения графика

Программирование

Откомпилированный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 1 МГц. Микроконтроллер запрограммирован при помощи STK500 в режиме программирования ISP. Датчик температуры DS1820 выполняет операции чтения и записи через однопроводной интерфейс Далласа, реализованный программно. Чтобы понять суть, вы можете заглянуть в спецификацию датчика температуры. Самые важные фрагменты кода иллюстрирует листинг 4.4.

Листинг 4.4

```
int ds1820_read(void)
{
 char busy=0;
 unsigned char temp1,temp2;
 int result;
 onewire_reset();
 onewire_write(0xCC);//Пропустить команду Rom
 onewire_write(0x44);
   //Команда преобразования температуры
 while (busy == 0)
 busy = onewire_read();
 onewire_reset();
 onewire_write(0xCC);//Пропустить команду Rom
 onewire_write(0xBE);
   //Прочитать команду ScratchPad
 temp1 = onewire_read();
 temp2 = onewire_read();
 onewire_reset();
 result = temp1*5;
   //Точность 0.5 градусов Цельсия
   //Результат в десять раз больше, чем фактическая температура
 return result;
}
```

Функция `ds1820_read` опрашивает датчик DS1820 и возвращает значение, которое в десять раз больше фактической температуры в градусах Цельсия (после выполнения необходимого масштабирования). Главный бесконечный цикл программы работает в двух режимах. В первом режиме отображается текущая температура (вместе с ее минимальным и максимальным значениями) в градусах Цельсия и Фаренгейта. Во втором режиме отображается график изменения температуры. График рисуется при помощи функции `graph1`, которая извлекает значения для вычерчивания пикселов из массива `data`. Функция `setlcd` служит для рисования осей на экране

LCD. Остальные фрагменты кода выполняют инициализацию LCD и графического режима. Кнопка S4 (PA1) переключает режим. Переход из режима рисования графика к отображению температуры (и обратно) не приводит к удалению графических данных.

Работа устройства

Регистратор спроектирован для работы от одной или двух батареек размера AA/AAA. Можно использовать как щелочные, так и перезаряжаемые элементы (никель-металлогидридные или никель-кадмиевые). Достаточно подать напряжение питания, и на дисплее начинает отображаться значение температуры. Для переключения между режимами отображения нажмите кнопку.

Проект 15. Игрушка Тэнгу с графическим дисплеем

На самом деле Тэнгу — это сказочный персонаж из японского фольклора. Но Тэнгу — это еще и популярная игрушка, в которой общение с игроком происходит с помощью разнообразных звуков и шумов. Предлагаемое устройство является аудиоигрой и не имеет ничего общего со сверхъестественными существами. В нашей игрушке Тэнгу представляет собой лицо с глазами, носом и ртом. В зависимости от окружающих звуков выражение глаз и лица Тэнгу будет меняться. На рис. 4.16 изображена блок-схема этого устройства. Микрофон, подключенный к аудиоусилителю, предназначен для улавливания звуков. Лицо отображается на графическом дисплее Nokia. Микропроцессор tinyAVR анализирует звук и меняет выражение лица. Устройство питается от батареи, и его можно носить с собой.

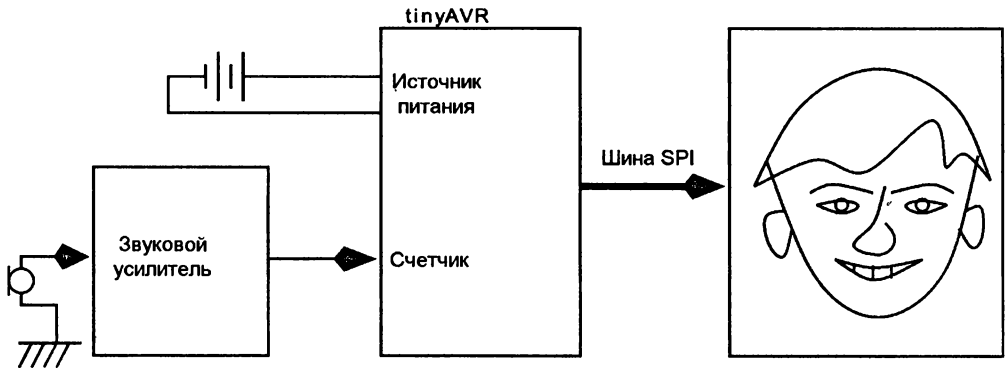


Рис. 4.16. Блок-схема игрушки Тэнгу

Спецификация проекта

Цель проекта — создать питающийся от батареи вариант игрушки Тэнгу, который реагирует на внешние звуки, музыку и шум (изменением выражения лица).

Описание устройства

На рис. 4.17 изображена схема игрушки Тэнгу. Устройство питается от батарейки 9 В, но будет хорошо работать даже от четырех щелочных батареек по 1,5 В. Напряжение подается через разъем SL2. Диод D2 обеспечивает защиту от неправильной полярности подключения батарей. Напряжение от источника питания подано на операционный усилитель LM358 и на вход преобразователя LP2950, который выдает напряжение Vcc (3,3 В) для работы микроконтроллера и дисплея Nokia. Дисплей подключен к схеме через разъем LCD1, микрофон — через SL4. При подключении конденсаторного микрофона необходимо соблюдать полярность. Питание на микрофон подается через резистор R7. Аналоговый сигнал с микрофона через конденсатор C5 емкостью 10 мкФ поступает на операционный усилитель.

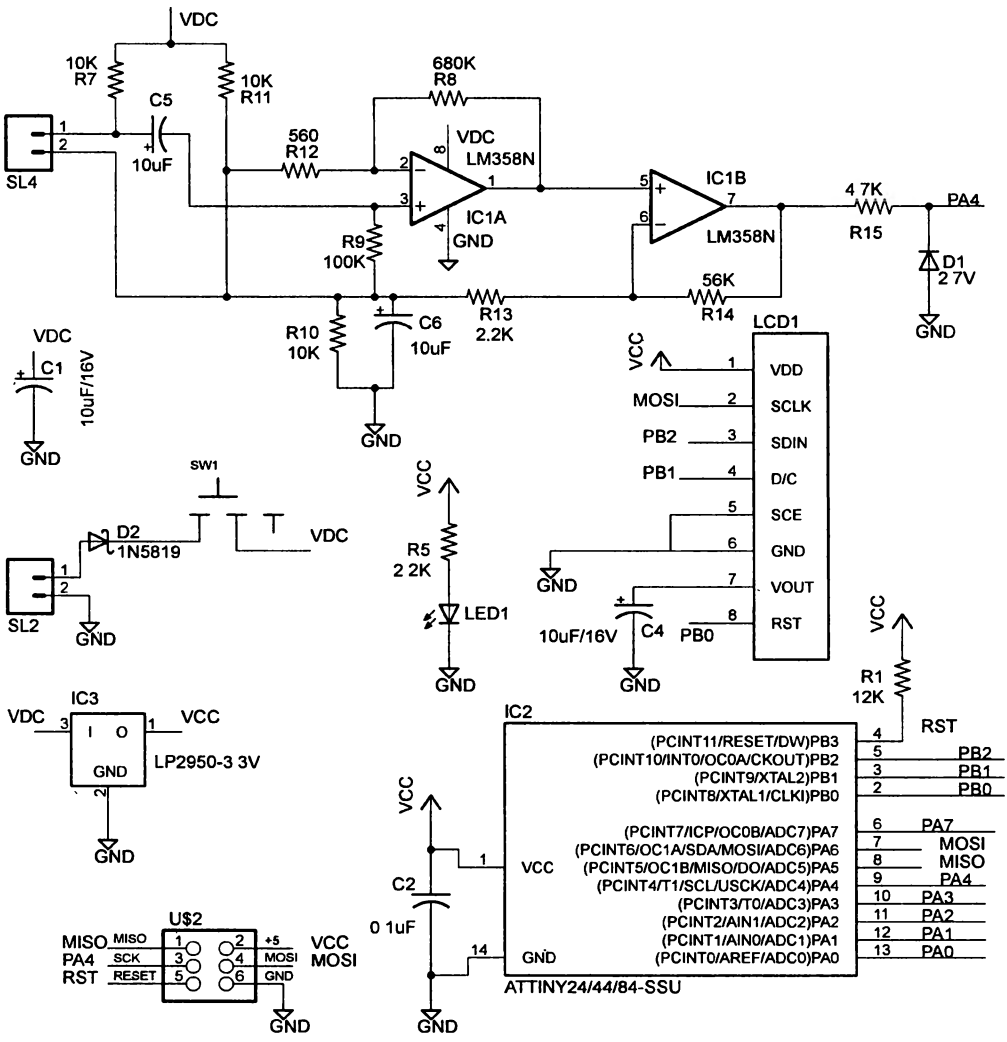


Рис. 4.17. Принципиальная схема игрушки Тэнгу

Микросхема включена как неинвертирующий усилитель постоянного тока с высоким усилением. Резисторы R10 и R11 обеспечивают постоянное смещение, равное половине входного напряжения. Поскольку резистор R10 зашунтирован конденсатором C6, то по переменному току точка соединения R10 и R11 оказывается заземлена. Полезный сигнал приложен ко входу усилителя (контакт 3).

За первым каскадом усилителя следует второй (собранный на второй половине операционного усилителя), включенный тоже как неинвертирующий. К выходу (контакт 7 микросхемы) подключен резистор R15 и стабилитрон D1 (2,7 В), поэтому выходное напряжение ограничивается до 2,7 В. Далее сигнал поступает на входной контакт PA4 микроконтроллера Tiny44. Усиление двух каскадов так велико (25 000), что сигнал на выходе представляет собой меандр с частотой исходного аудиосигнала. Задача микроконтроллера — определить частоту этого сигнала и затем изменить выражение лица на графическом дисплее. Микроконтроллер постоянно замеряет частоту аудиосигнала.

Операционный усилитель питается напрямую от батарей. Для LM358 нужно напряжение не менее 5 В, поэтому схема должна питаться либо от четырех батареек по 1,5 В, либо от батареи 9 В. Микроконтроллер и дисплей Nokia 3310 питаются от стабилизатора LP2950, который дает выходное напряжение 3,3 В.

Конструкция

Компоновку платы в программе EAGLE (и принципиальную схему) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Плата односторонняя (на стороне компонентов есть всего несколько перемычек). Стороны печатной платы изображены на рис. 4.18 и 4.19. Работающий образец устройства показан на рис. 4.20.

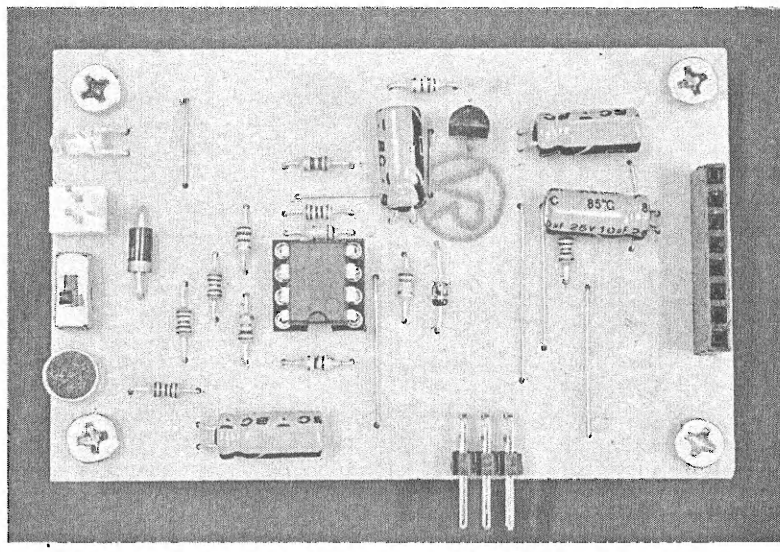


Рис. 4.18. Печатная плата игрушки Тэнгу (сторона компонентов)

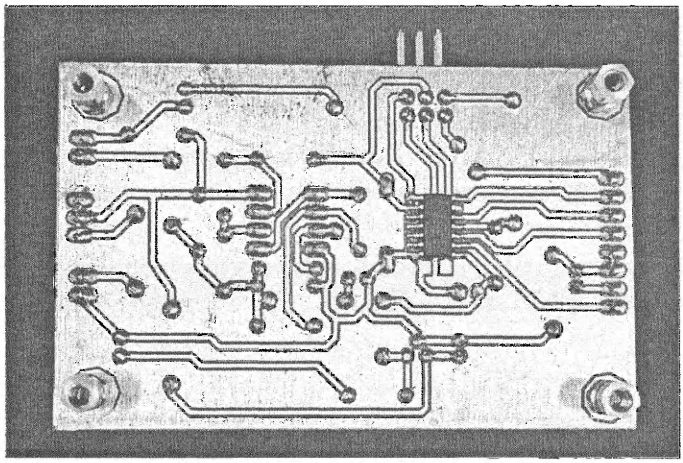


Рис. 4.19. Печатная плата игрушки Тэнгу (сторона печатных проводников)

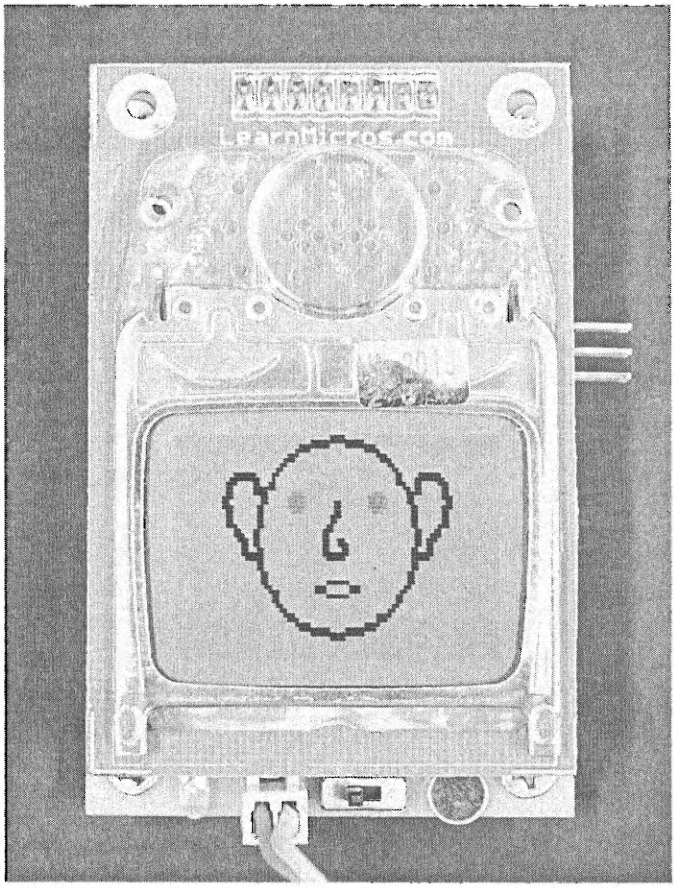


Рис. 4.20. Работающий образец игрушки Тэнгу

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 8 МГц. Контроллер запрограммирован при помощи STK500 в режиме программирования ISP. Самые важные фрагменты кода иллюстрирует листинг 4.5.

Листинг 4.5

```
void check(void)
{
 TIFR0 = 1<<TOV0;
 TCNT0 = 255-125;
 TCCR0B = 1<<CS00|1<<CS01;
 count = 0;
 freq = 0;
 stop_check=0;
 while(count<200)
 {
  while((PINA&(1<<4))&&(stop_check==0)
 );
 while((!(PINA&(1<<4)))&&(stop_check==0));
 if(PINA&(1<<4))
 {
  freq++;
 }
}
freq = freq*5;
if(freq>=1900)
face = 9;
else if(freq>=1700)
face = 7;
else if(freq>=1500)
face = 6;
else if(freq>=1200)
face = 1;
else if(freq>=900)
face = 0;
else if(freq>=650)
face = 5;
else if(freq>=350)
face = 2;
```

```
else if(freq>=200)
face = 4;
else if(freq<200)
face = 3;
TCCR0B = 0x00; //Остановить таймер
}
```

Это функция, которая измеряет частоту входного сигнала на контакте PA4 микроконтроллера. Она инициализирует Timer0 таким образом, что прерывание по его переполнению возникает каждую миллисекунду. Эта процедура увеличивает переменную `count` до 200 (это и есть интервал в 200 мс). В течение этого времени каждое изменение состояния контакта PA4 записывается в переменную `freq`. Она затем умножается на 5 (чтобы получить число изменений за одну секунду), что и дает частоту входного сигнала. Далее, исходя из полученного результата, методом проб и ошибок переменной `face` присваивается определенный номер лица, который функция `putface` использует для формирования выражения лица на дисплее LCD (извлекая отображаемые байты из памяти программ микроконтроллера). Она также отображает подмигивающие глаза (вызывая функцию `puteye`).

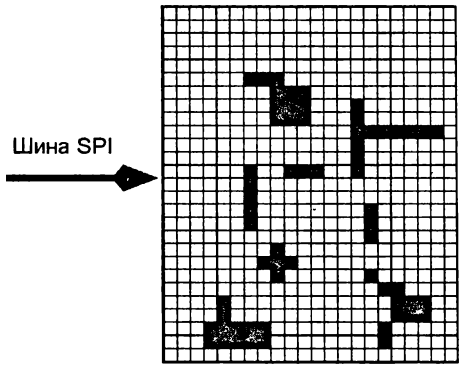
Главная процедура нашего кода сначала инициализирует LCD, а затем попеременно вызывает функции `check` и `putface` (в бесконечном цикле).

Работа устройства

Для работы с игрушкой Тэнгу вам нужно сначала подключить питание, а затем разместить ее так, чтобы микрофон не был закрыт. Вы увидите лицо на дисплее, а когда появится какой-нибудь звук, глаза мигнут, и изменится выражение рта. Если вы включите громкую музыку, то рот будет постоянно меняться — как будто Тэнгу поет.

Проект 16. Игра "Жизнь"

Эта математическая модель, изобретенная Джоном Конвеем — попытка смоделировать процессы реальной жизни при помощи простых правил. Игроки для нее не нужны, поэтому после настройки начального состояния не требуется никакого вмешательства. Подробности этой игры можно найти в Интернете и мы настоятельно рекомендуем вам почитать о ней. Предлагаемый проект позволяет пользователю смоделировать данную игру на дисплее Nokia при помощи микроконтроллера tinyAVR. В оригинальной игре используется двумерное поле бесконечного размера, но в нашем устройстве размер игрового пространства ограничен 16×16 ячеек. Пользователь может задать начальное состояние при помощи кнопок (как показано на рис. 4.21). После настройки начального состояния игру можно запустить и посмотреть на ее эволюцию.



"Жизнь"

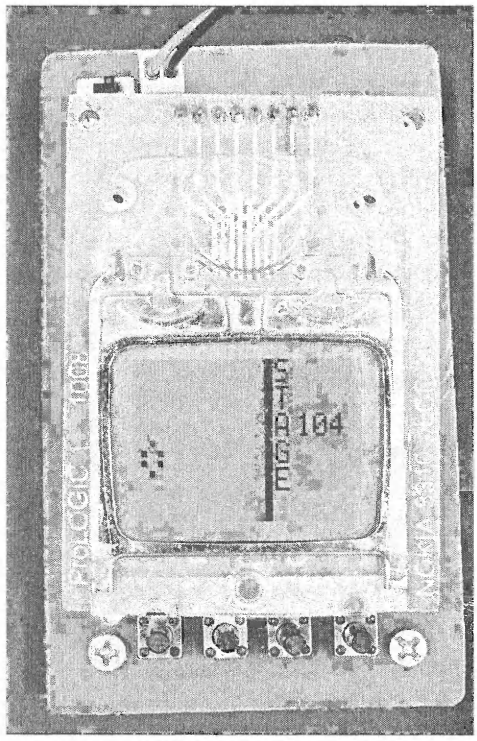


Рис. 4.22. Внешний вид игры

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 8 МГц. Контроллер запрограммирован при помощи STK500 в режиме программирования ISP. Поясним наиболее важные фрагменты кода (листинг 4.6).

Листинг 4.6

```
void place(void)
{
 row=0;
 column=0;
 while(1)
 {
  if(!(PINA&0b00000001))
  {
   _delay_ms(30);
   while(!(PINA&0b00000001));
   _delay_ms(30)
   TIMSK1 = 0x00;
   if(led[0][row]&(1<<column))
   {
    pix_light(row,column,1);
   }
   else if(!(led[0][row]&(1<<column)))
   {
    pix_light(row,column,0);
   }
    row++;
    if(row==(ROWMAX+1))
    row = 0;
    TIMSK1 = 0x01;
   }
   if(!(PINA&0b00000010))
   {
    _delay_ms(30);
    while(!(PINA&0b00000010));
    _delay_ms(30);
    TIMSK1 = 0x00;
    if(led[0][row]&(1<<column))
    {
```

```
  pix_light(row,column,1);
 }
 else if(!(led[0][row]&(1<<column)))
 {
  pix_light(row,column,0);
 }
 column++;
 if(column==(COLMAX+1))
 column = 0;
 TIMSK1 = 0x01;
}
if(!(PINA&0b00100000))
{
 _delay_ms(50);
 while(!(PINA&0b00100000));
 _delay_ms(50);
 TIMSK1 = 0x00;
 if(led[0][row]&(1<<column))
 led[0][row]&=~(1<<column);
 else led[0][row]|=1<<column;
 if(led[0][row]&(1<<column))
 {
  pix_light(row,column,1);
 }
 else if(!(led[0][row]&(1<<column)))
 {
  pix_light(row,column,0);
 }
 row++;
 if(row==(ROWMAX+1))
 row = 0;
 TIMSK1 = 0x01;
}
if(!(PINA&0b00001000))
{
 _delay_ms(30);
 while(!(PINA&0b00001000));
 _delay_ms(30);
 TIMSK1 = 0x00;
 if(led[0][row]&(1<<column)
 {
  pix_light(row,column,1);
```

```
    }
    else if(!(led[0][row]&(1<<column)))
    {
     pix_light(row,column,0);
    }
    TIMSK1 = 0x01;
    break;
  }
 }
}
```

Функция, приведенная в листинге 4.6, служит для установки начального состояния (комбинации "живых" и "мертвых" ячеек). Каждая ячейка на дисплее LCD состоит из девяти пикселов. Верхние левые 2×2 пиксела у активных ячеек светятся, а остальные пять пикселов всегда выключены (во избежание слияния соседних пикселов). Все игровое поле состоит из 16×16 таких ячеек. Для оптимальной загрузки оперативной памяти мы использовали для представления этой конфигурации целочисленные массивы (размером по 16 элементов). Каждая переменная массива соответствует одной строке (ее 16 битов означают 16 столбцов). Настройка начального состояния осуществляется четырьмя кнопками: две перемещают курсор по строкам и столбцам, третья переключает состояние текущей ячейки, а последняя запускает игру.

Листинг 4.7

```
while(1)
{
 clearram();
 stage=0;
 for(unsigned char j=0;j<=5;j++)
 {
  cursorxy(48,j);
  writedata(0xFF);
  writedata(0xFF);
  writedata(0xFF);
 }
 TIMSK1 = 0x01;
    //Вектор прерывания по переполнению
 place();
 TIMSK1 = 0x00;
    //Отключение прерывания по переполнению
 cursorxy(52,0);
 putcharacter('S');
```

```
cursorxy(52,1);
putcharacter('T');
cursorxy(52,2);
putcharacter('A');
cursorxy(52,3);
putcharacter('G');
cursorxy(52,4);
putcharacter('E');
generation=0;
generation1=1;
while(1)
{
 for(int row=0;row<=ROWMAX;row++)
 {
  for(int col=0;col<=COLMAX;col++)
  {
   if(led[generation][row]&(1<<col))
   led[generation1][row] |=1<<col;
   else led[generation1][row]
   &=~(1<<col);
   if(led[generation][row]&(1<<col))
     //включено
   {
    check_neighbors(generation,row,col);
    if(alive>3||alive<2)
    {
     led[generation1][row] &= ~(1<<col);
     pix_light(row,col,0);
    }
   }
   else
  if(!(led[generation][row]&(1<<col)))
   {
    check_neighbors(generation,row,col);
    if(alive == 3)
    {
     led[generation1][row] |= 1<<col;
     pix_light(row,col,1);
    }
   }
  }
```

```
  }
  generation = (generation==0)?1:0;
  generation1 = (generation==0)?1:0;
  //правила игры
  stage++;
  stage1=stage;
  cursorxy(60,2);putcharacter (stage1/100+48);
  stage1 = stage1%100;
  putcharacter(stage1/10+48);
  stage1 = stage1%10;
  putcharacter(stage1+48);
  _delay_ms(500);
  flag=0;
  for(unsigned char i=0;i<=15;i++)
  {
   if(!(actual[i]==0))
   {
   flag=1;
   break;
   }
  }
  if((flag==0)||(stage==255))
  {
   clearram();
   for(unsigned char i=0;i<=15;i++)
   {
    led[0][i]=0;
    led[1][i]=0;
    actual[i]=0;
    flag=0;
   }
   cursorxy(12,2);
   putstr("GAME OVER");
   while(PINA&0x01);
   _delay_ms(30);
   while(!(PINA&0x01));
   _delay_ms(30);
   break;
  }
 }
}
```

Листинг 4.7 — это главный бесконечный цикл программы. Он выполняет два основных действия: проверяет состояние каждой ячейки (в соответствии с правилами игры) и обновляет дисплей. Одновременно он заполняет следующее состояние каждой ячейки (в соответствующей позиции массива `led`). Он проверяет количество "живых" соседей ячейки: больше трех или меньше двух (максимально соседей может быть восемь). Если одно из этих условий верно, то ячейка выключается ("умирает"). В том случае, когда у ячейки ровно три соседа — она сохраняется (или включается, если она "мертвая"). После проверки всех ячеек следующее состояние становится текущим (и т. д.). Для итерации между текущим и следующим поколениями требуются только две переменные (`generation` и `generation1`). Игра заканчивается, когда "умирают" (выключаются) все ячейки или когда значение `stage` (число "поколений") становится равно 255.

Остальные части кода инициализируют различные переменные, функции для сопряжения с LCD и процедуру обработки прерывания (для переполнения TIMER1). Процедура обработки прерывания вступает в дело только при настройке начального состояния (для того, чтобы мигала текущая ячейка, на которую указывает курсор).

Работа устройства

Для включения устройства нужно подать питание на плату. Появится мигающий курсор в левом верхнем углу экрана. При помощи кнопок выбирайте нужные элементы и переключайте их состояние. Когда начальное состояние игры будет настроено, нажмите кнопку для запуска процесса. Справа на дисплее отображаются итерации.

Проект 17. Крестики-нолики

Это устройство позволяет двум людям играть в крестики-нолики. Его блок-схема изображена на рис. 4.23.

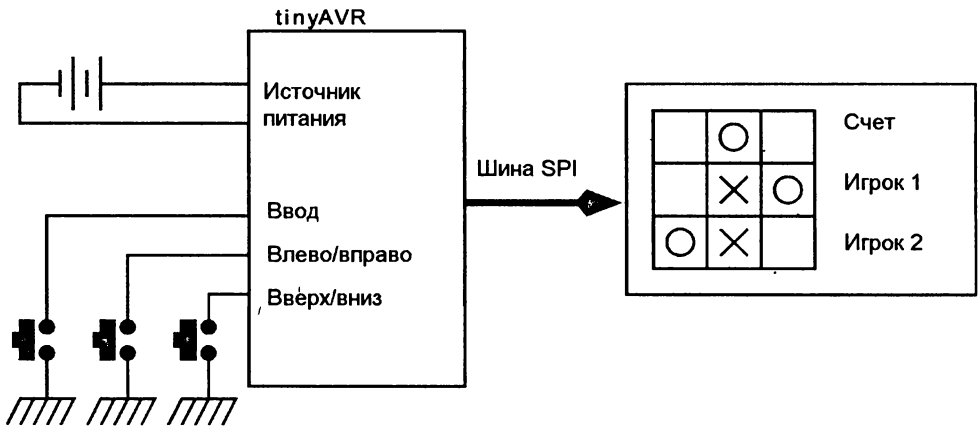


Рис. 4.23. Блок-схема устройства

Спецификация проекта

Цель проекта — создать графический интерфейс для игры в крестики-нолики. Пользователь размещает свой знак при помощи кнопок. Игра заканчивается либо ничьей, либо победой одного из игроков. Устройство отслеживает количество сыгранных игр и счет. Питание реализовано от батареек.

Описание устройства

Схема аналогична проекту температурного регистратора. В этом проекте задействовано три из имеющихся четырех кнопок: кнопки "вверх/вниз", "вправо/влево" и "ввод". На рис. 4.24 и 4.25 изображен дисплей, показывающий различные стадии игры.

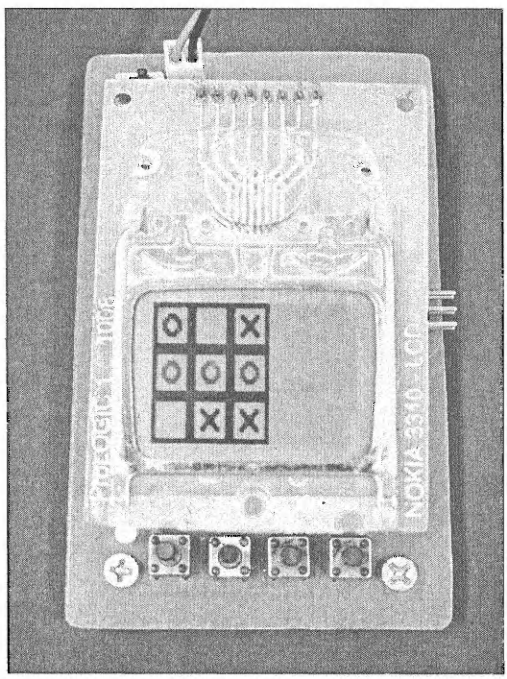


Рис. 4.24. Крестики-нолики: отображение хода игры

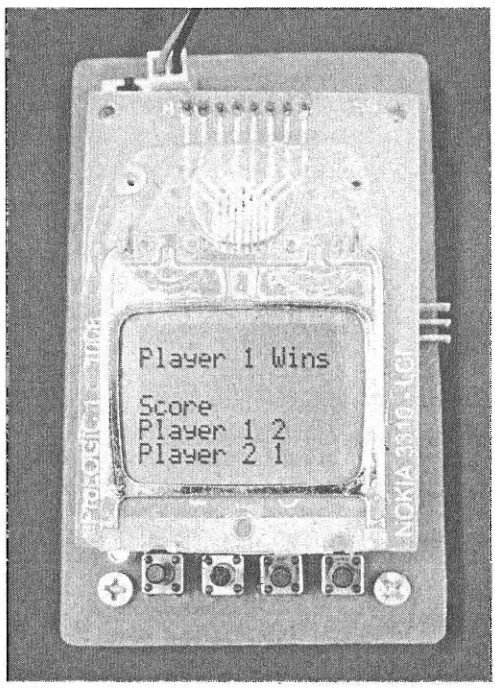


Рис. 4.25. Крестики-нолики: отображение результатов

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 8 МГц. Контроллер запрограммирован при помощи STK500 в режиме программирования ISP. Логика игры — та же самая, что и в игре, обсуждавшейся в проекте 13. Единственное отличие состоит в том, что нет началь-

ного выбора цветов. Одному пользователю назначаются крестики, другому — нолики. Шаблоны "крестик" и "нолик" хранятся в памяти программ. В отличие от предыдущей версии, здесь предусмотрено запоминание числа побед и проигрышей каждого пользователя (до десяти побед). В конце каждой игры результат и общий счет пользователей отображается на дисплее. Листинг 4.8 иллюстрирует наиболее важные фрагменты кода.

Листинг 4.8

```
while(1)
{
 cli();
 cursorxy(6,2);
 putstr(DISPLAY);
 while(PINA&0x01);
 _delay_ms(30);
 while('(PINA&0x01));
 _delay_ms(30);
 sei();
 TIMSK1 = 0x01;
   //Прерывание по переполнению активировано
 reset();
 while(1)
 {
  playerturn(1);
  checkwin();
  if(dis1==10)
  {
   dis1=0;
   dis2=0;
   break;
  }
  playerturn(2);
  checkwin();
  if(dis2==10)
  {
   dis1=0;
   dis2=0;
   break;
  }
 }
}
```

Это главный бесконечный цикл программы. Он выводит на экран слова "TIC TAC TOE" и ждет нажатия пользователем кнопки (для запуска игры). После начала игры вызывается функция `playerturn`, аргументом которой является номер игрока. Эта функция позволяет пользователю поместить его символ. Затем вызывается функция `checkwin`, чтобы проверить, не выиграл ли игрок, разместивший последний символ. Переменные `dis1` и `dis2` отслеживают число выигрышей игроков.

Работа устройства

Чтобы включить устройство, подайте питание на схему. Сначала возможность выставить свой символ получает первый игрок, потом — второй и т. д.

Проект 18. "Дурацкие" часы

Часы на базе микроконтроллера давно всем известны. Но предлагаемое устройство отличается от остальных. В этом проекте (вместо отображения цифр на LCD или светодиодном индикаторе) часы, минуты и секунды прокручиваются мимо вертикальной черты на экране. Именно поэтому эти часы и называются "дурацкими". Блок-схема приведена на рис. 4.26.

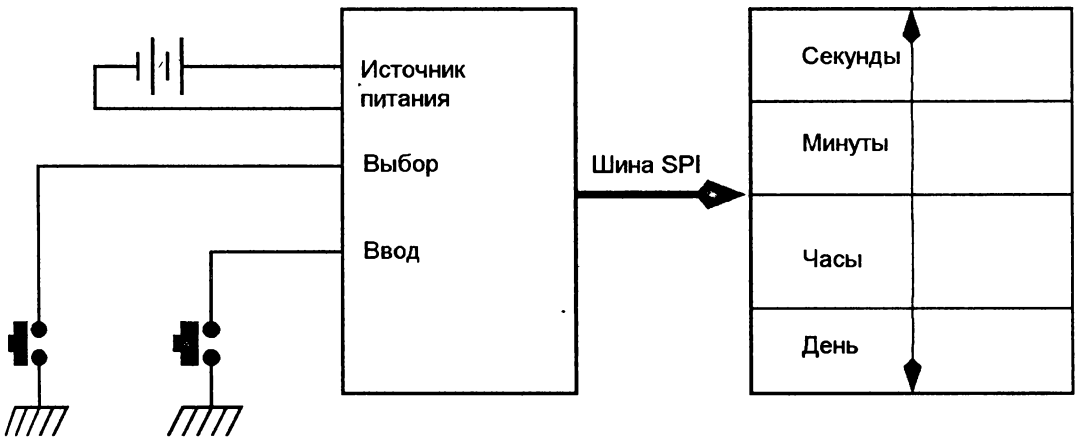


Рис. 4.26. Блок-схема часов

Спецификация проекта

Цель проекта — создать на основе микроконтроллера часы с необычным индикатором, на котором время проходит мимо отметки. Устройство питается от батарейки, чтобы его можно было переносить с места на место и не быть привязанным к розетке.

Описание устройства

На рис. 4.27 изображена принципиальная схема устройства. В ней использован микроконтроллер Tiny861 и дисплей Nokia. К микроконтроллеру подключен внешний кварц с частотой 7,3728 МГц. Та же самая тактовая частота служит и для отсчета реального времени. Устройство питается от батареи в 9 В (можно также взять четыре батарейки по 1,5 В). Напряжение от батарей стабилизирует микросхема LP2950-3.3 (для питания микроконтроллера и дисплея Nokia).

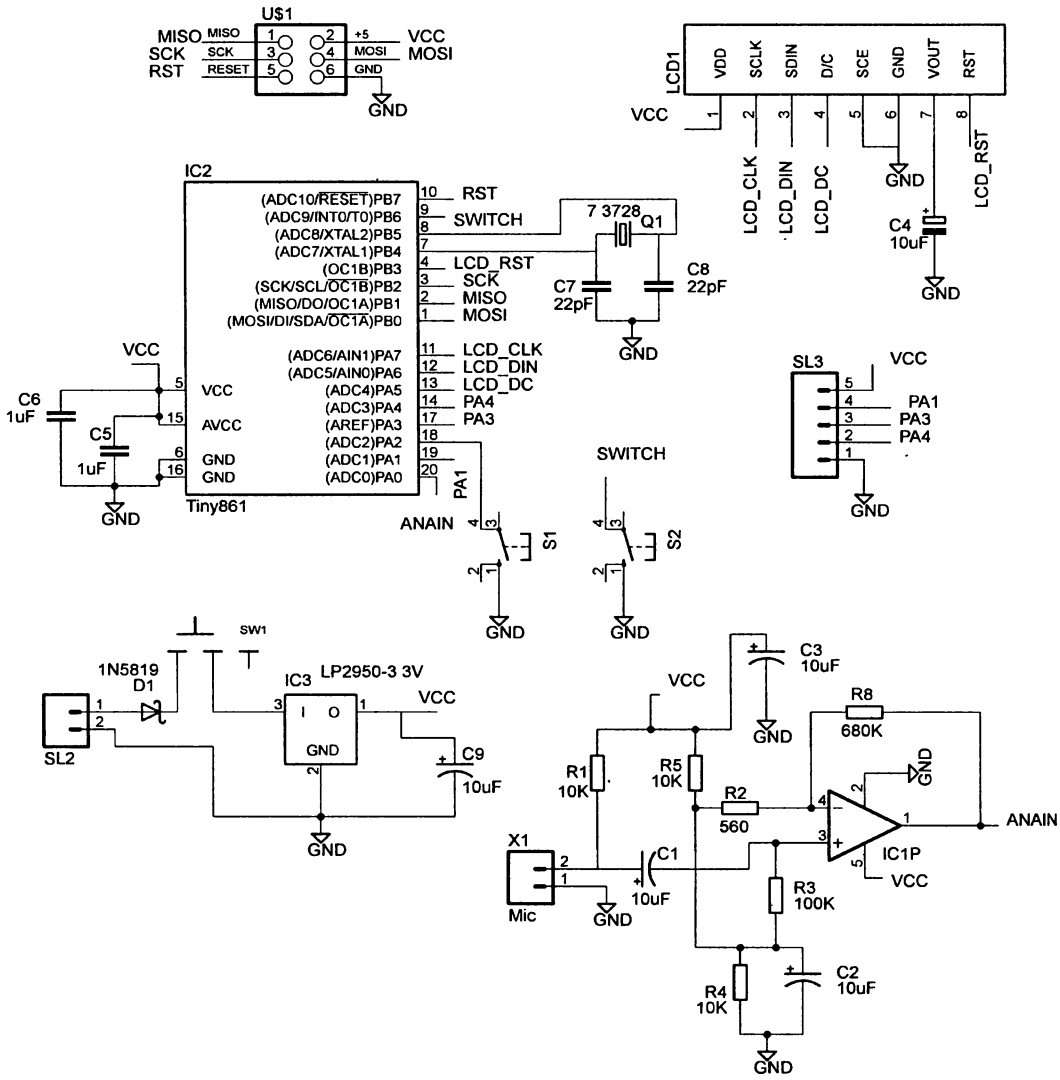


Рис. 4.27. Принципиальная схема часов

На схеме показаны еще некоторые компоненты (операционный усилитель и разъем для конденсаторного микрофона), но они к проекту часов отношения не имеют. Две кнопки S1 и S2 предназначены для настройки времени и прокручивания дисплея. На часах отображаются секунды, минуты, часы и дни недели. Однако из-за ограниченного размера дисплея одновременно можно видеть только три элемента. Кнопкой S2 можно прокручивать эти элементы.

Конструкция

Компоновку платы в программе EAGLE (и ее принципиальную схему) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Плата односторонняя (на стороне компонентов есть всего несколько перемычек). Стороны печатной платы приведены на рис. 4.28 и 4.29. Две фотографии работающего устройства показаны на рис. 4.30 и 4.31.

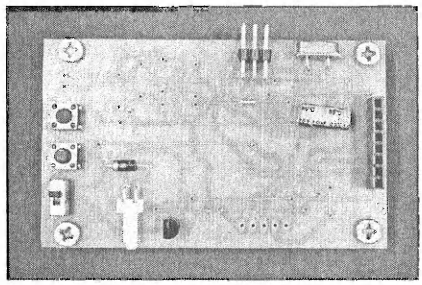


Рис. 4.28. Плата устройства (сторона компонентов)

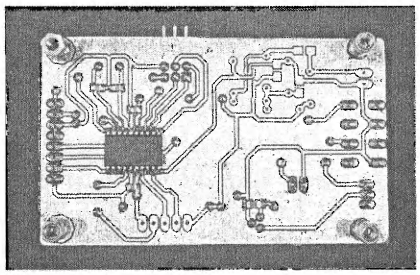


Рис. 4.29. Плата устройства (сторона печатных проводников)

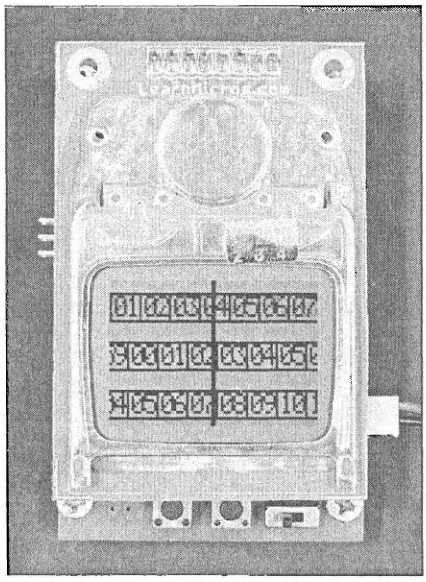


Рис. 4.30. Дисплей показывает три строки символов

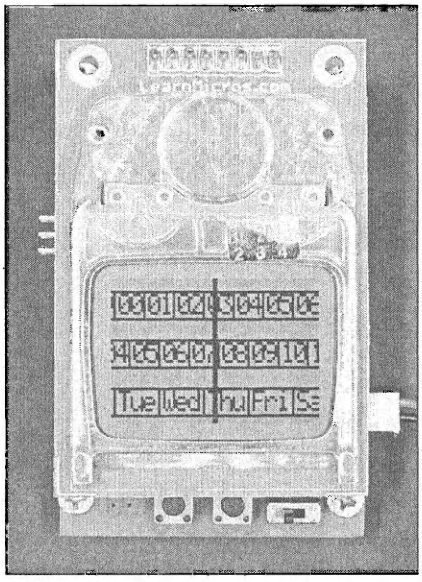


Рис. 4.31. Дисплей после прокрутки строк

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота выбрана равной 7,3728 МГц. Контроллер запрограммирован при помощи STK500 в режиме программирования ISP. Для работы устройства на внешней тактовой частоте необходимо перед программированием контроллера установить его fuse-биты CKSEL в 1101. Некоторые фрагменты кода такие же, как и в предыдущих проектах на основе дисплея Nokia 3310, поэтому мы включили в состав проекта библиотеку для работы с LCD (что избавляет от необходимости повторно писать код для сопряжения с LCD). Все необходимые функции для сопряжения с LCD и отображения данных взяты непосредственно из этой библиотеки.

Для формирования изображения мы создали две функции: `boxes` и `centerline`. Первая рисует символы отображения времени на нулевой, второй и четвертой страницах, а вторая — центральную линию, которая отмечает прошедшее время. Эта линия имеет координаты X, равные 41 и 42 — и присутствует на каждой странице. Данные (секунды, минуты, часы) хранятся в таблице в памяти программ. Для отображения символа или цифры нужно 12 столбцов, а таблица состоит из 60 таких символов — поэтому всего требуется 720 столбцов. В таблице `table2` хранятся символы дней недели. Для отображения символа одного дня недели требуется 18 столбцов, т. е. общий размер массива равен 18 × 7 = 126.

TIMER0 инициализируется частотой в 7200 Гц, получаемой путем деления системной частоты на 1024; при помощи выставления в 1 бита `TOIE0` регистра `TIMSK` активизируется прерывание по переполнению. Дальнейшее деление выполняется программно, как показано в процедуре обработки прерывания (листинг 4.9).

Листинг 4.9

```
ISR(TIMER0_OVF_vect)
    //процедура обработки прерывания таймера
{
  TCNT0L=(255-225);
  count++;
  if(count==8)
     //деление на 8
  {
    count=0;
    if(s.c==0)//число секунд
    s.c=1;
    if(m.c<20)//число минут
    m.c++;
    if(h.c<1200)//число часов
    h.c++;
```

```
    if(d.c<19200)//число дней
    d.c++;
  }
}
```

В этой процедуре регистр таймера инициализируется таким образом, что переполнение происходит при значении таймера 225 (а не 255). Таймер работает на частоте 7200 Гц, поэтому прерывание по переполнению происходит через 225/7200 секунд. Когда значение переменной `count` достигает 8, выполняется процедура обработки прерывания. Это означает, что тело процедуры выполняется каждые 225×8/7200 секунды, а это одна четверть секунды.

Секунда истекает после того, как 12 столбцов проходят мимо центральной линии. Поэтому на каждом восьмом прерывании таймера (т. е. каждую четверть секунды) мимо центральной линии должны проходить три столбца. Это делается в главном цикле программы. Бесконечный цикл `while` отслеживает секунды, которые нужно отображать на LCD. В этом бесконечном цикле есть четыре функции из библиотеки rtc library: `seconds()`, `minutes()`, `hours()` и `days()`. Функция `seconds()` приведена в листинге 4.10.

Листинг 4.10

```
if(s.end!=723)
  //Если end не равен пределу таблицы table1
{
cursorxy(0,s.row);
  //установить курсор на страницу секунд
for(i=s.start;i<s.end;i++)
  //записать содержимое start в end
{
column=pgm_read_byte((table1+i));
writedata(column);
}
centerline(); //отобразить центральную линию
if(s.c==1) //проверить count
{
s.start+=3;
s.end+=3;
s.c=0;
}
}
else if(s.end==723)
//если end достиг предела table1
{
```

```
centerline();
cursorxy(0,s.row);
for(i=s.start;i<(s.end-3);i++)
  //отобразить содержимое начиная со start
{
column=pgm_read_byte((table1+i));
writedata(column);
}
for(i=0;i<s.x;i++)
//отобразить с первого элемента таблицы table 1
{
column=pgm_read_byte((table1+i));
writedata(column);
}
centerline(); //временная линейка
if(s.c==1) //проверить count
{
s.start+=3;
s.x+=3;
s.c=0;
}
if(s.x==84)
//проверить – дошел ли start до end
{
s.end=84;
s.start=0;
s.x=3;
}
}
```

Все отображаемые на дисплее символы (секунды, минуты, часы и дни) имеют в структуре `rtcpara` свои наборы переменных:

- `start` — начальная X-координата подлежащих отображению данных;
- `end` — конечное значение X-координаты подлежащих отображению данных;
- `x` — смещение X, используемое для отображения данных из начала таблицы PROGMEM (когда `end` достигает своего предельного значения);
- `c` — счетчик числа, который увеличивается в процедуре обработки прерывания.
- `row` — страница соответствующего числа.

В предыдущей процедуре ее первая часть выполняется тогда, когда `s.end` не равно предельному значению (723). После этого данные выводятся на страницу `s.raw` от `s.start` до `s.end` (это 84 столбца, поэтому данные выводятся на полную

страницу). Затем включается `centerline` или `timeline` (для отображения прошедшего времени) и проверяется `s.c`. Если `s.c=1`, то `s.start` и `s.end` увеличиваются на 3 и `s.c` опять присваивается 0. Вторая часть выполняется тогда, когда `s.end` достигает максимального значения. Сначала данные отображаются от `s.start` до `s.end-3` (поскольку `s.end` в первой части была увеличена на 3) на странице `s.row`. Затем данные выводятся с начала таблицы `PROGMEM` до `s.x`. Это продолжается до тех пор, пока `s.x` не достигнет значения 84. В этот момент `s.start` присваивается 0, а `s.end` — значение 84. Происходит такая же проверка `s.c`, как и в первой части, и включается `centerline`.

Минуты, часы и дни отображаются при помощи аналогичных (определенных в `rtc.c`) процедур. Помимо этого, код содержит начальную установку времени (при помощи кнопок) и отслеживает прерывание по изменению состояния контакта для переключения между отображением дней и секунд.

Работа устройства

Для работы часов нужно подать напряжение и установить текущее время. Затем вы увидите, как "пролетают" по дисплею секунды. Минуты идут медленнее секунд, а часы — еще медленнее. Нажмите кнопку S2, и вы увидите день недели. Нажмите S2 еще раз, и вы вернетесь обратно.

Проект 19. Громкий будильник

Это устройство мы создали для местной школы. Нужно было получить очень громкий сигнал, который будил бы учеников утренней смены. Мы добавили также некоторые дополнительные функции. Специальная программа превращает сигнал подъема в школьный звонок. На рис. 4.32 показана блок-схема устройства. Питание предусмотрено от сети, но есть и резервное питание от батарей. Звуковой усилитель выдает мощный сигнал, который будит детей.

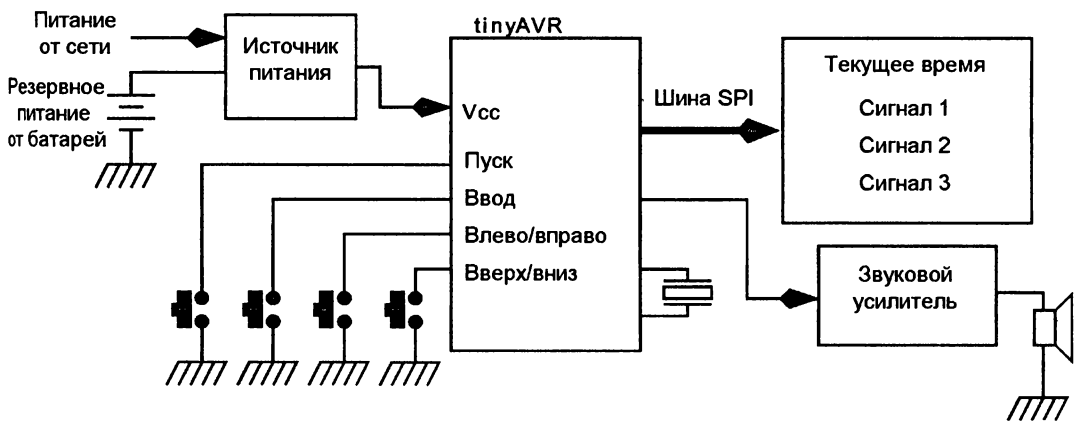


Рис. 4.32. Блок схема устройства

Спецификации проекта

Цель проекта — создать громкий сигнал подъема (с резервным питанием от батарей для непрерывного отсчета времени). Можно задать три момента времени срабатывания устройства.

Описание устройства

На рис. 4.33 показана принципиальная схема устройства, а на рис. 4.34 — подключение дополнительных кнопок.

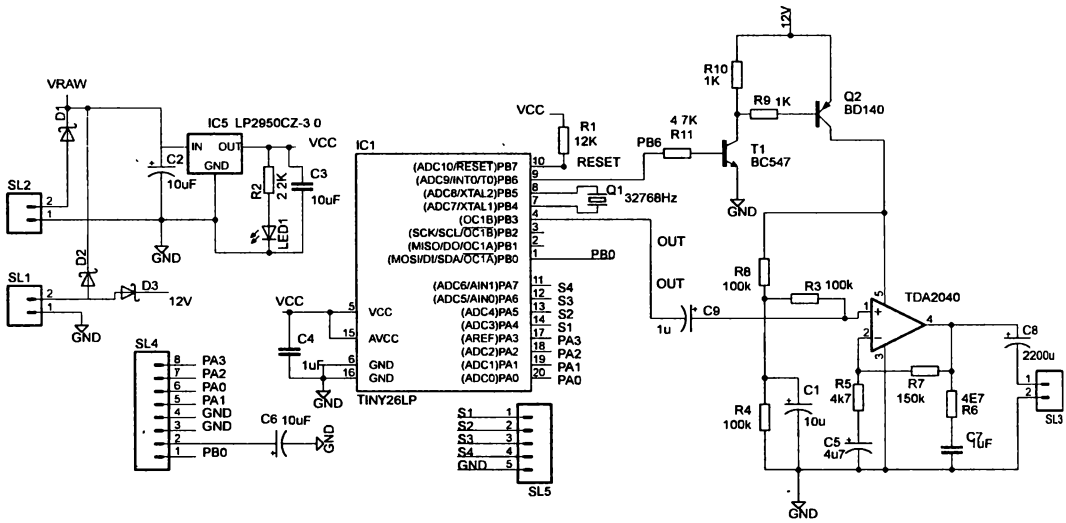


Рис. 4.33. Принципиальная схема устройства

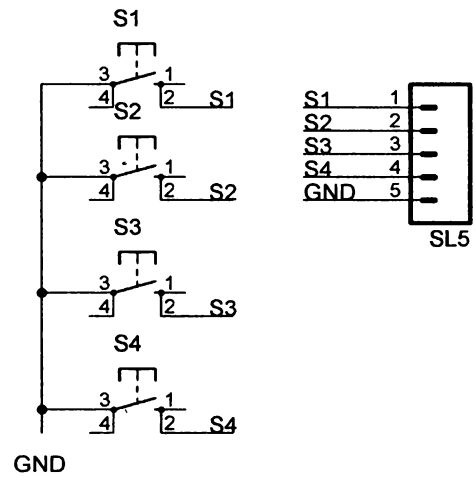


Рис. 4.34. Схема подключения кнопок

Устройство имеет два варианта питания: от сети и от батареи. Сеть предназначена для питания микроконтроллера и звукового усилителя, а батарея — только для микроконтроллера. К микроконтроллеру Tiny861 подключен кварц на 32,768 кГц в качестве источника тактовой частоты (и для выполнения программы, и для отсчета времени). Поэтому подача рабочего напряжения на микроконтроллер исключительна важна для работы устройства. Пользователь может установить три момента срабатывания, и когда текущее время совпадет со временем срабатывания, микроконтроллер сгенерирует сигнал на контакте PB3, который подается на усилитель мощностью 20 Вт. Микроконтроллер питается через стабилизатор LP2950-3.3.

Жидкокристаллический дисплей Nokia обеспечивает взаимодействие с пользователем. После подачи напряжения на схему в верхнем правом углу дисплея появляется текущее время (вместе с меню из шести пунктов: TIME, MODE, DISP, ALARM1, ALARM2 и ALARM3):

- TIME — устанавливает текущее время;
- MODE — включает/выключает время срабатывания;
- DISP — выключает дисплей;
- ALARM*x* — устанавливает время срабатывания.

Треугольный указатель показывает текущий пункт меню; его можно перемещать вверх/вниз при помощи кнопок S2/S3. После нажатия S1 отображается подменю для выбранного пункта меню:

- TIME — S1 обновляет текущее время и выходит из меню. S2 переводит указатель изменяемой цифры. Кнопки S3/S4 увеличивают/уменьшают текущую цифру.
- MODE — S1, S2 и S3 переключают время срабатывания Alarm1, Alarm2 и Alarm3, а кнопка S4 осуществляет выход из меню. Состояние сигнала показано в верхнем левом углу главного экрана.
- DISP — дисплей просто очищается. Дисплей можно включить нажатием любой кнопки. После включения дисплей возвращается в то же самое состояние, в котором он был перед выключением.
- ALARM*x* — то же самое, что и TIME, но обновляется время срабатывания (а не системное время).

Кнопка S4 останавливает сработавший сигнал.

Конструкция

Компоновку платы в программе EAGLE (и принципиальную схему) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Плата односторонняя (на стороне компонентов есть всего несколько перемычек). Стороны печатной платы показаны на рис. 4.35 и 4.36. Фотография готового устройства с проводами к громкоговорителю приведена на рис. 4.37.

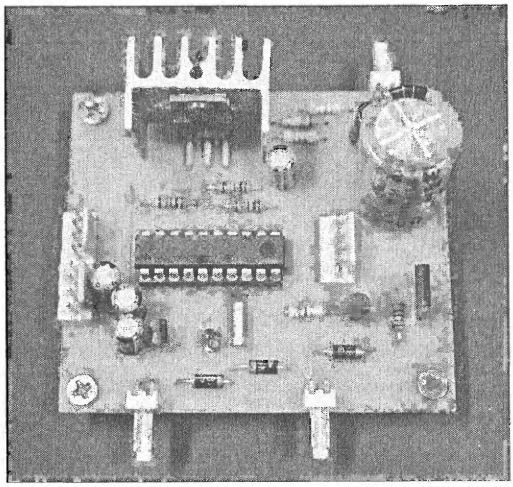


Рис. 4.35. Печатная плата устройства (сторона компонентов)

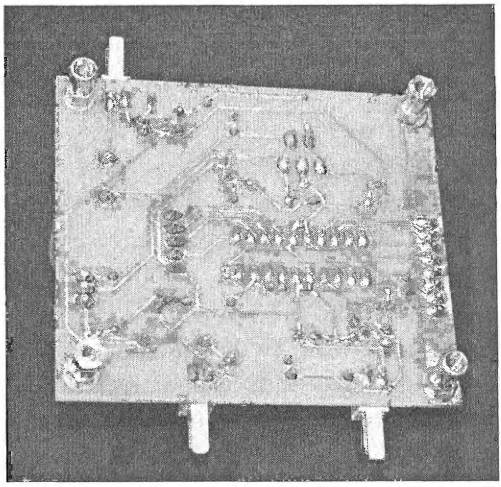


Рис. 4.36. Печатная плата устройства (сторона печатных проводников)

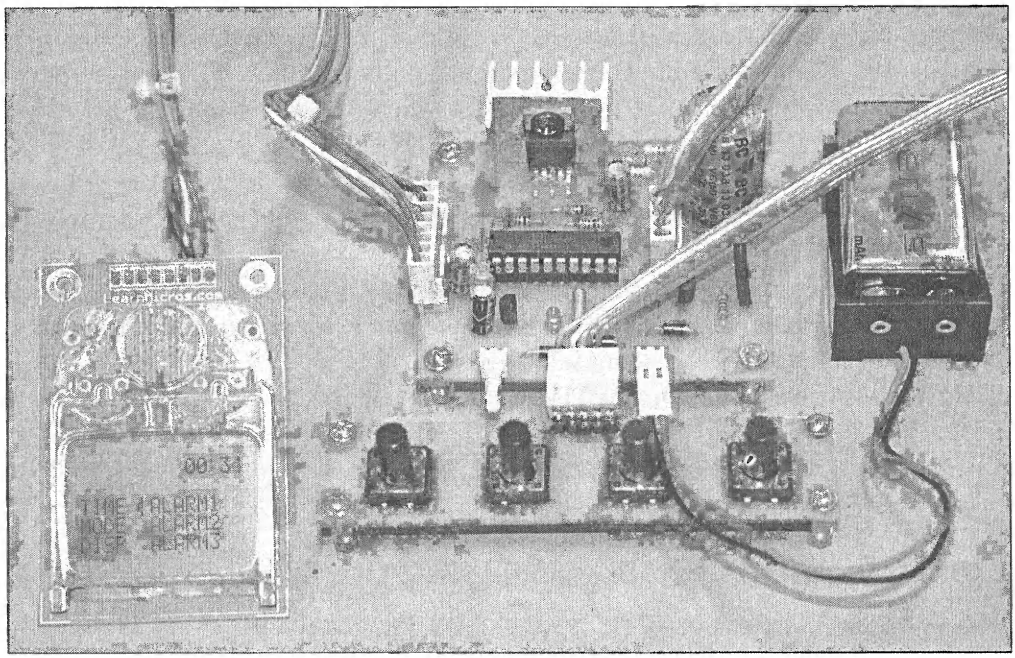


Рис. 4.37. Внешний вид устройства

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 32768 Гц (подается от внешнего кварца). Контроллер запрограммирован при помощи STK500 в режиме программирования ISP. Самые важные фрагменты кода иллюстрирует листинг 4.11. Программа управляется через меню, поэтому некоторые функции имеют бесконечные циклы `while`, которые позволяют пользователю изменять настройки и выходить из цикла (выбрав соответствующий пункт меню). Функция `setmode` дает возможность пользователю включать/выключать любое из трех срабатываний, а `setalarm` — установить время срабатывания. Важными фрагментами кода являются те, которые отображают текущее время, продолжают звучание при срабатывании, а также включают сигнал при наступлении времени срабатывания. Поэтому эти три фрагмента повторяются во всех циклах `while` внутри главной функции и внутри некоторых других функций. Функция `showTime_d` отображает текущее время, а `showTime` отображает время, которое передается в нее через массив `Time[3]`. Программа выполняется на низкой частоте, поэтому обновление дисплея занимает некоторое время, но в результате снижается общее энергопотребление.

Листинг 4.11

```
void showTime(unsigned char Time[3], unsigned char x, unsigned char y)
{
 cursorxy(x%84,y%6);
 timetext[0]=(Time[hour]/10+'0');
 timetext[1]=(Time[hour]%10+'0');
 timetext[2]=(':');
 timetext[3]=(Time[min]/10+'0'');
 timetext[4]=(Time[min]%10+'0');
 timetext[5]= '\0';
 putstr(timetext);
}

void showTime_d(unsigned char x, unsigned char y)
{
 cursorxy(x%84,y%6);
 putcharacter(Alarm[0][hour]/10+'0');
 putcharacter(Alarm[0][hour]%10+'0');
 putcharacter(':');
 putcharacter(Alarm[0][min]/10+'0');
 putcharacter(Alarm[0][min]%10+'0');
 if(((Alarm[0][sec])%2))
```

```
 {
  cursorxy((x%84)+2*6,y%6);
  putcharacter(' ');
 }
 if(al_on==0)
 {
  cursorxy(0,1);
  if(alarm1==1)
  {
   putstr("1 ");
  }
  else putstr(" ");
  if(alarm2==1)
  {
   putstr("2 ");
  }
  else putstr(" ");
  if(alarm3==1)
  {
   putstr("3 ");
  }
  else putstr(" ");
 }
}
```

Время отсчитывается при помощи внешнего кварца — это гораздо точнее, чем встроенный RC-генератор микроконтроллера. В остальных фрагментах кода запускается система автоподстройки частоты, инициализируются и сбрасываются значения времени срабатывания, а также настраиваются другие глобальные параметры.

Работа устройства

Работа устройства подробно описана в предыдущем разделе.

Заключение

В этой главе рассмотрены некоторые проекты на базе графического дисплея Nokia 3310. Многие приведенные здесь устройства можно легко модифицировать под другие цели. В следующей главе мы сосредоточимся на некоторых датчиках и обсудим проекты, выполненные с их использованием.

Глава 5

Проекты с датчиками

В этой главе мы рассмотрим несколько проектов, внимание в которых будет сконцентрировано на различных датчиках. Перечень датчиков огромен, и его невозможно охватить даже в целой книге (не говоря уже об одной главе). Однако мы предложим несколько проектов, основанных на часто встречающихся датчиках, причем некоторые из датчиков будут использованы весьма оригинально.

Основные виды датчиков

Светодиод в качестве датчика

Светодиоды часто используются для отображения информации. Однако современные светодиоды могут также работать и как фотогальванические детекторы. Убедиться в этом просто: нужно подключить светодиод к мультиметру и направить на него источник яркого света.

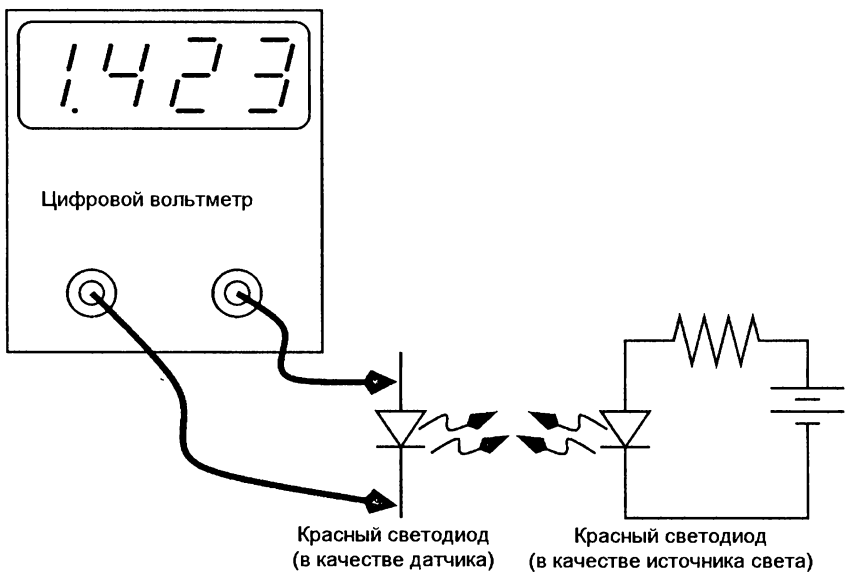


Рис. 5.1. Светодиод в качестве фотодетектора

Например, если подключить два контакта красного светодиода к мультиметру и осветить его другим таким же красным светодиодом (рис. 5.1), то мультиметр должен показать более 1,4 В. Однако для работы со светодиодом в качестве датчика потребуется аналого-цифровой преобразователь (АЦП), который достаточно дорог. Далее в этой же главе мы покажем, как можно включить светодиод в качестве датчика без АЦП, при помощи одного таймера (либо аппаратного, либо программного).

Термистор

Термистор — это резистор, сопротивление которого зависит от температуры. Это один из самых дешевых датчиков температуры. Термисторы бывают двух типов: с отрицательным температурным коэффициентом (NTC) и с положительным температурным коэффициентом (PTC). При повышении температуры сопротивление термисторов PTC возрастает, а термисторов NTC — падает. Единственный недостаток термисторов — это нелинейность, т. е. их сопротивление меняется не прямо пропорционально температуре. В узком диапазоне температур возможна линейная аппроксимация, в обычных условиях необходимо преобразование либо при помощи внешних аппаратных компонентов, либо программный пересчет по таблицам или по формуле Стейнхарт-Харта. Несмотря на эти сложности, термистор является превосходным датчиком температуры, имеющим малое время реакции и широко применяемым в термостатах. На рис. 5.2 показано изменение температуры NTC-термистора в зависимости от температуры, а на рис. 5.3 изображены различные термисторы.

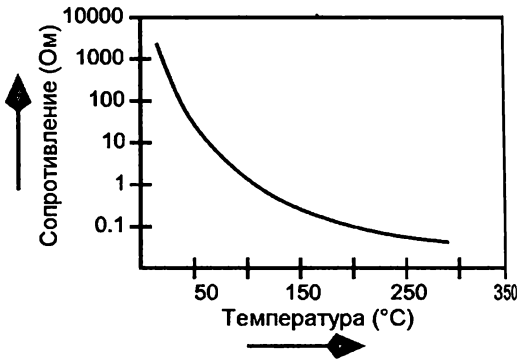


Рис. 5.2. Зависимость сопротивления NTC-термистора от температуры

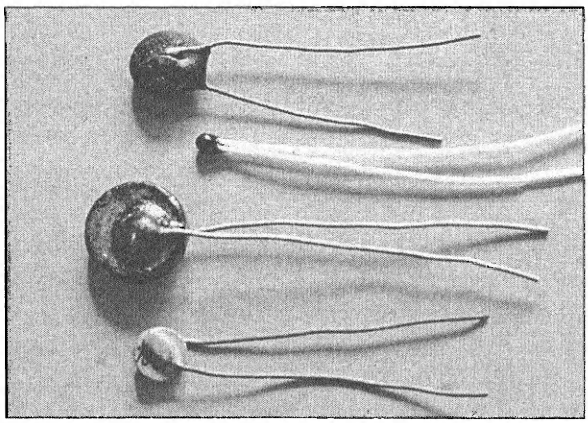


Рис. 5.3. Внешний вид различных термисторов

Фоторезистор

Фоторезисторы (LDR) предназначены для детектирования света. Фоторезистор — это полупроводниковый прибор, сопротивление которого уменьшается с повышением интенсивности падающего на него света. Сделанные из сульфида кадмия фоторезисторы весьма дешевы, но имеют малое быстродействие. На рис. 5.4 показана зависимость сопротивления фоторезистора от интенсивности падающего света. На рис. 5.5 изображены некоторые фоторезисторы.

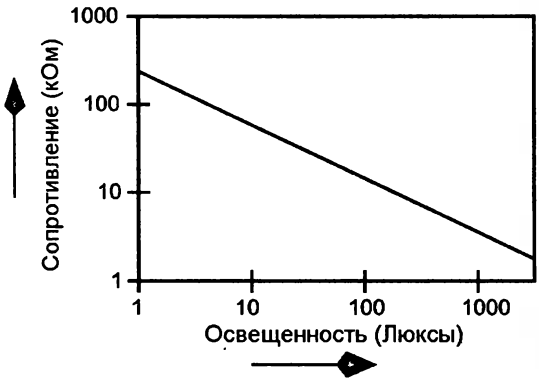


Рис. 5.4. Зависимость сопротивления фоторезистора от интенсивности света

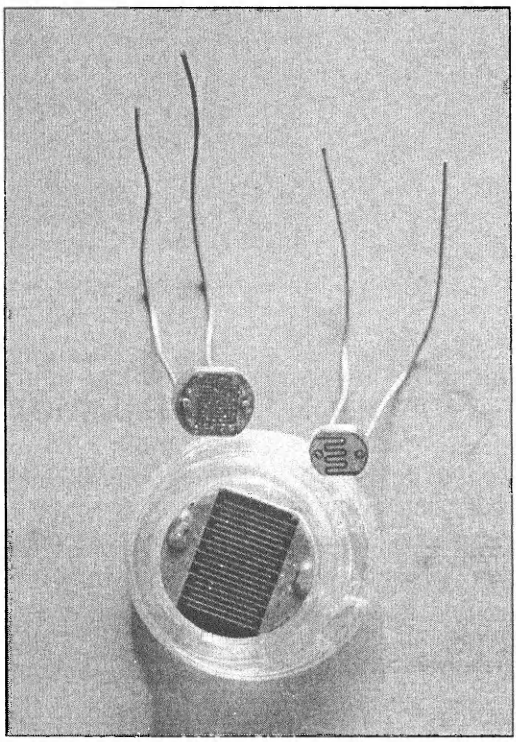


Рис. 5.5. Внешний вид различных фоторезисторов

Катушка индуктивности как датчик магнитного поля

Катушка индуктивности — это пассивный электрический компонент, который при прохождении через него тока сохраняет энергию за счет магнитного поля. Если катушку поместить в меняющееся магнитное поле, то на ее выводах (по закону Фарадея) возникнет напряжение. Катушка является простым датчиком, с помощью которого можно обнаруживать переменные магнитные поля. Мы будем использовать это свойство катушки индуктивности в нескольких проектах данной главы. На рис. 5.6 изображены различные типы катушек индуктивности.

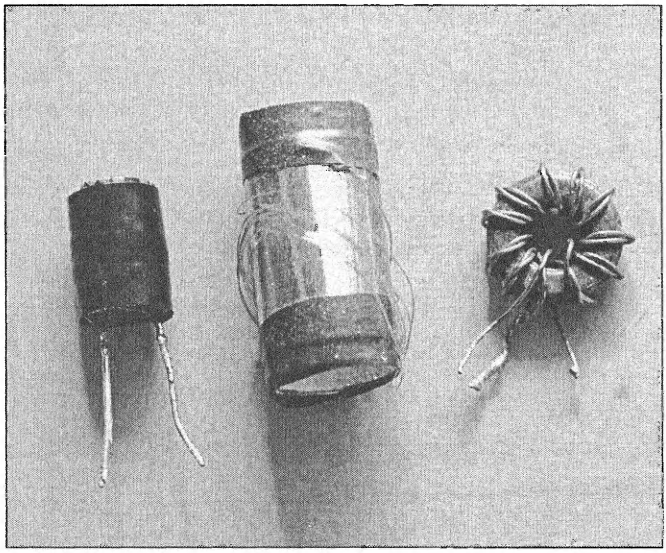


Рис. 5.6. Внешний вид катушек индуктивности

Проект 20. Светодиод как датчик и индикатор

Как уже упоминалось, светодиоды можно использовать в качестве датчиков. Фактически светодиоды могут менять режим работы (излучатель света или фотодетектор) так быстро, что создается впечатление, будто в одном корпусе находятся два устройства. Для включения светодиода как датчика, нужно подать на него обратное напряжение. При работе светодиода в обычном режиме нужно подать на него прямое напряжение. В этом проекте мы покажем, как включить светодиод для детектирования света без применения дорогого АЦП. Тот же самый светодиод будет служить индикатором обнаруженного света.

Светодиод при обратном смещении можно рассматривать как зависящий от света источник питания, включенный параллельно с конденсатором. Чем больше падает света, тем больше величина тока — а это быстрее разряжает конденсатор. На рис. 5.7 светодиод находится под обратным напряжением: его анод подключен к заземлению, а катод — к выводу микроконтроллера (контакт 2). Микроконтроллер подает на контакт 2 напряжение Vcc, которое заряжает конденсатор. Затем катод светодиода подключается к входному контакту (контакт 1) микроконтроллера. Конденсатор (который был заряжен до Vcc) теперь разряжается, и, когда напряжение на конденсаторе упадет ниже определенного уровня, контакт 1 микроконтроллера считает логический 0. Если интенсивность падающего света выше, то конденсатор разряжается быстрее, а если света меньше, то конденсатор разряжается дольше. Таким образом, путем измерения времени падения напряжения на контакте 1 до логического 0, наш микроконтроллер может оценить интенсивность света, падающего на светодиод.

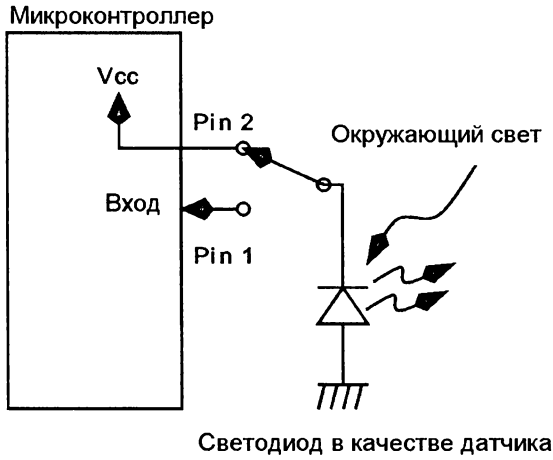


Рис. 5.7. Светодиод в качестве фотодетектора

Спецификация проекта

Цель проекта — использовать светодиод одновременно и как датчик, и как индикатор обнаруженного света. На рис. 5.8 показана блок-схема устройства. В ней один светодиод и несколько других компонентов. Светодиод мигает, а частота этого мигания пропорциональна интенсивности падающего света: если светодиод поместить под более яркий свет, он будет мигать быстрее.

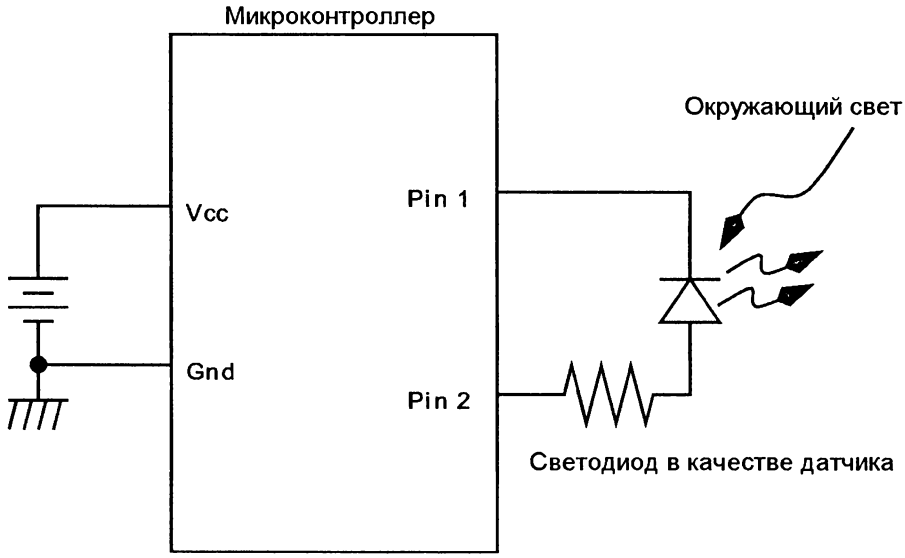


Рис. 5.8. Блок-схема устройства

Описание устройства

На рис. 5.9 изображена принципиальная схема устройства на базе микроконтроллера ATtiny15, в которой красный светодиод LED1 в прозрачном корпусе служит для определения и индикации (изменением частоты мигания) интенсивности окружающего света.

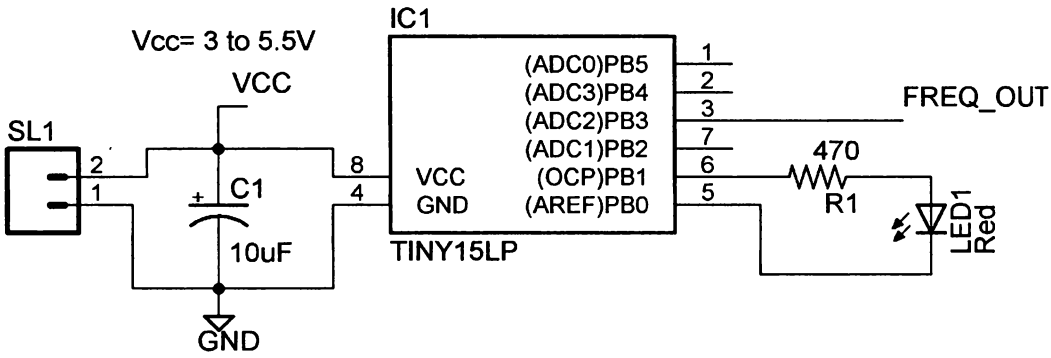


Рис. 5.9. Принципиальная схема устройства

Схема проста: в ней всего четыре компонента. Напряжение источника питания может быть от 3 до 5,5 В. Светодиод подключен к выводам PB0 и PB1 микроконтроллера. На выходе PB3 формируется меандр с частотой, пропорциональной интенсивности падающего света. Схема работает так: сначала на светодиод подается прямое смещение (на некоторое время). Затем на светодиод подается обратное смещение (для этого изменяются последовательности битов, подаваемые на PB0 и PB1). На следующем шаге PB0 конфигурируется как входной контакт. В цикле измеряется интервал времени, который требуется светодиоду для изменения напряжения на контакте PB0 с логической 1 на логический 0. Это время T обратно пропорционально падающему на светодиод свету. В результате светодиод начинает мигать с частотой, которая обратно пропорциональна времени T. При более слабом свете наш светодиод мигает медленнее, а по мере увеличения интенсивности падающего света частота мигания светодиода возрастает. Так обеспечивается визуальная индикация интенсивности падающего света.

Конструкция

Принципиальную схему проекта можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Схема изготовлена на небольшой печатной плате (рис. 5.10 и 5.11) и состоит всего из пяти компонентов.

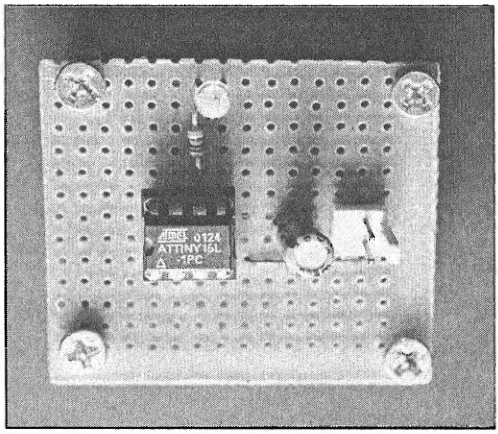


Рис. 5.10. Печатная плата устройства (сторона компонентов)

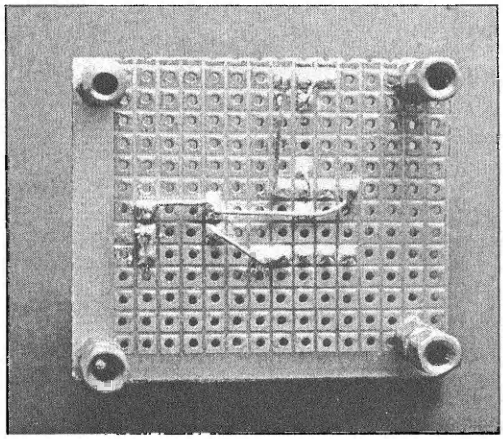


Рис. 5.11. Печатная плата устройства (сторона пайки)

Программирование

Код проекта написан на языке ассемблера (листинг 5.1). Сначала программа инициализирует контакты PB0 и PB1 как выходные и выставляет на PB0 логическую 1, а на PB1 — логический 0 (для обратного смещения светодиода). Затем контакт PB0 настраивается как входной и программа ждет, пока напряжение на PB0 не снизится до логического 0. Время хранится в регистре `r19`. Затем PB0 опять конфигурируется как выходной контакт и светодиод получает прямое смещение (чтобы включить его) на тот промежуток времени, который хранится в регистре `r19`. Таким образом, если микроконтроллер в первом цикле измерения фиксирует время *T*, за которое светодиод "разряжается", то он включает светодиод на то же время *T*. В итоге частота мигания светодиода будет пропорциональна падающему на него свету.

Листинг 5.1

```
.include "tn15def.inc"
.cseg
.org 0
;светодиод как датчик света...
main:
   ldi r16, 255
   out DDRB, r16
   ldi r16, 0
   out PORTB, r16
   ldi r19, 1
   rcall delay
   ldi r19, 1
```

```
new_main:
   sbi DDRB, 0
   nop
   nop
   sbi PORTB, 1 ; LED forward bias
   cbi PORTB, 0
   rcall delay
   sbi PORTB, 0
   cbi PORTB, 1 ; reverse bias
   cbi DDRB, 0 ; LED discharge
   cbi PORTB, 0
; настроить регистры для минимальной задержки
ldi r19, 1
wait_here:
   sbis PinB, 0
      rjmp its_one
      rcall min_delay
   inc r19
   brne dont_inc_r20
   rjmp over_flow
   dont_inc_r20: rjmp wait_here
over_flow:
its_one:
   in r16, PORTB
   ldi r17, 0b00001000
   eor r16, r17 ; переключить PB3 для генерирования
                ; пропорциональной свету частоты
   out PORTB, r16
   mov r2, r19
   rcall delay
   mov r19, r2
   rjmp new_main
delay:
   ldi r20, 0
dec_r20:
dec_r21: dec r20
    brne dec_r20
    dec r19
  brne dec_r20
  ret
min_delay: in r0, SREG
  ldi r18, 200
```

```
not_over:
  dec r18
  brne not_over
  out SREG, r0
  ret
```

Работа устройства

Схема была протестирована с помощью светодиода с известной интенсивностью свечения. При небольших значениях прямого тока через светодиод интенсивность излучаемого света практически линейно зависит от тока. Свет от тестового светодиода подавался на светодиод-датчик LED1. Никакой другой внешний свет на датчик не падал (оба светодиода были заключены в закрытую трубку, заклеенную черной клейкой лентой). Ток тестового светодиода изменялся от 0,33 до 2,8 мА. Регистрировалась частота переключения светодиода. На рис. 5.12 видно, что зависимость практически линейна.

Рис. 5.12. Частота мигания светодиода-датчика в зависимости от интенсивности падающего света

Микроконтроллер ATtiny15 имеет восемь выводов. Наша схема использует только три из шести контактов ввода/вывода. Остальные контакты можно применить для управления внешними устройствами или для обмена данными с ними. Эффективность работы светодиода в качестве датчика зависит от источника тока и емкостных характеристик светодиода. Мы оценили эти цифры для сравнения с теми, которые приводятся в литературе. Для оценки обратного фотоэлектрического тока мы подключили резистор сопротивлением 1 МОм параллельно светодиоду-датчику и замерили напряжение на резисторе. Датчик освещался постоянным светом, и регистрировалось напряжение на резисторе. Мы изменили значение сопротивления на 500 кОм, потом на 100 кОм и повторили измерения. В результате (при постоянном освещении) мы получили при всех измерениях фотоэлектрический ток примерно 25 мА. При этом же освещении датчика замерялась и частота мигания,

генерируемая схемой рис. 5.9, а время задержки, ток и напряжения подставлялись в уравнение dU/dt = I/C (для вычисления емкости обратно смещенного *p-n*-перехода). Полученные в итоге значения находятся в диапазоне от 25 до 60 пФ.

Проект 21. Валентинка с датчиком близости

В этом проекте мы продолжаем демонстрировать возможности применения светодиода в качестве датчика. Устройство состоит из нескольких светодиодов, расположенных в виде сердца, которое пульсирует с обычной (характерной для сердца) частотой. Однако если к нему поднести руку, то датчик улавливает уменьшение освещенности и микроконтроллер повышает частоту пульсации. Если руку поднести еще ближе, то датчик обнаруживает это и микроконтроллер начинает зажигать светодиоды необычным образом (чтобы продемонстрировать состояние счастья). Если руку убрать, то "сердце" снова пульсирует нормально. На рис. 5.13 показана блок-схема устройства.

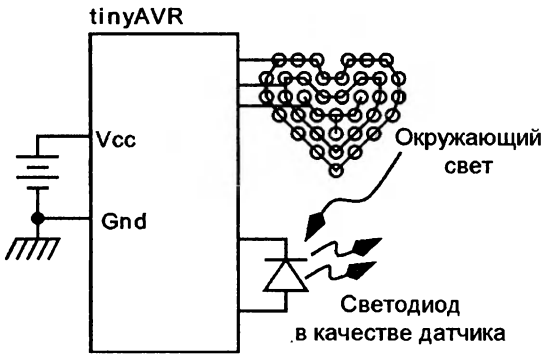


Рис. 5.13. Блок-схема устройства

Спецификация проекта

Цель — создать интерактивный проект, имеющий вид матрицы светодиодов в форме сердца. Функционирование устройства состоит в реакции на приближение руки (или любого другого объекта), при этом изменяется частота мигания светодиодов. Устройство питается от батарей, чтобы обеспечить портативность.

Описание устройства

Принципиальная схема устройства показана на рис. 5.14. Цепь состоит из 27 светодиодов, расположенных тремя концентрическими рядами. Внешний ряд состоит из 14 светодиодов. Средний ряд состоит из десяти светодиодов, а внутренний — из трех. В центре внутреннего ряда находится еще один светодиод (LED28), который не излучает, а воспринимает свет. Схема питается от батарей и снабжена стабилизатором напряжения на 3,3 В. Последовательно с каждым светодиодом включен

резистор сопротивлением 560 Ом, поэтому ток в цепи составляет примерно 3 мА. Для включения/выключения рядов предусмотрены три *n-p-n*-транзистора (BD139). Такой транзистор может управлять коллекторным током 1 А. Однако в данной схеме максимальный ток (через внешний ряд светодиодов) составляет примерно 45 мА, в принципе здесь сможет работать любой *n-p-n*-транзистор с током коллектора 100 мА.

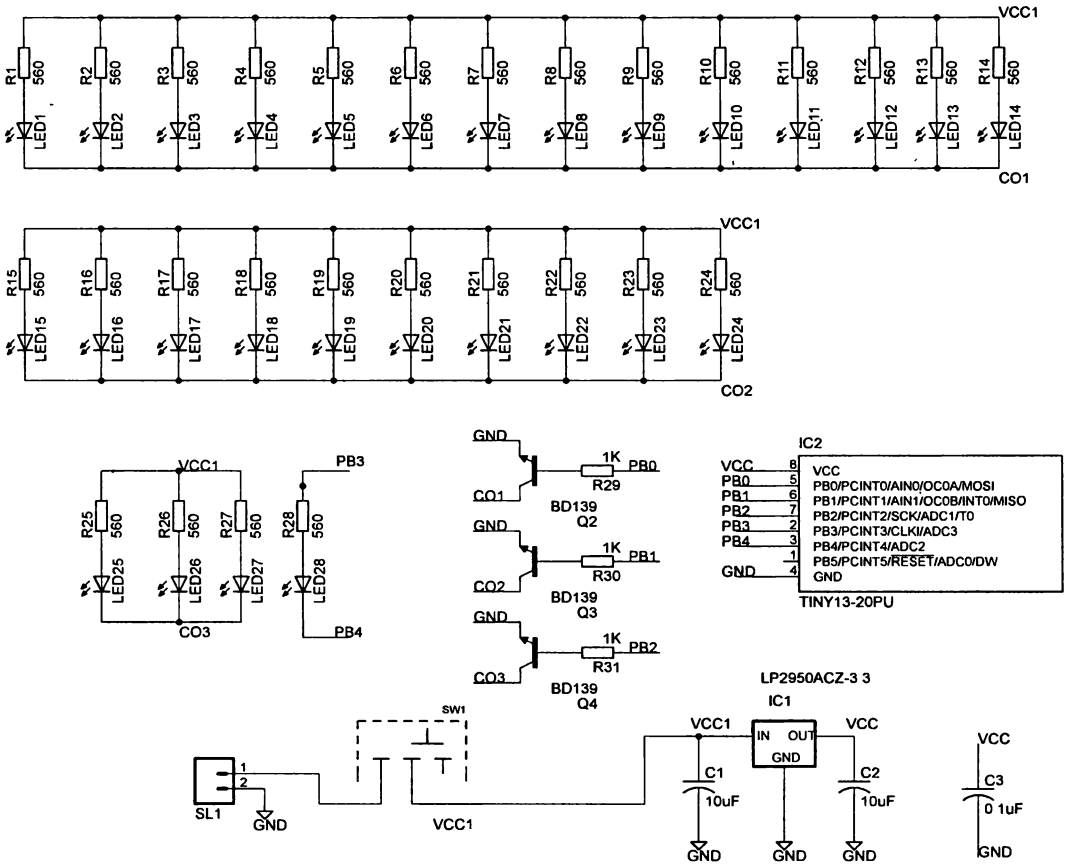


Рис. 5.14. Принципиальная схема светодиодного сердца

Интенсивность внешнего света измеряет светодиод LED28, подключенный к контактам микроконтроллера (PB3 и PB4). Микроконтроллер подает на светодиод обратное смещение и замеряет время, за которое состояние логической единицы превращается в состояние логического нуля. Схема может питаться от любого внешнего источника с напряжением от 5 до 10 В.

Конструкция

Компоновку платы в программе EAGLE (а также ее принципиальную схему) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

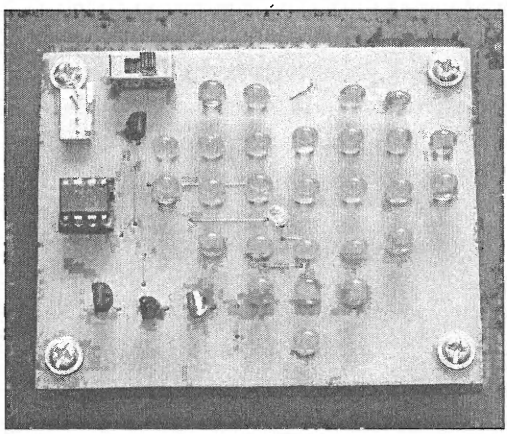


Рис. 5.15. Печатная плата светодиодного сердца (сторона компонентов)

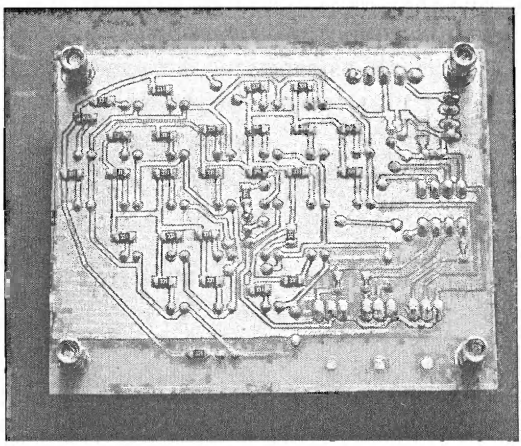


Рис. 5.16. Печатная плата светодиодного сердца (сторона печатных проводников)

Устройство было изготовлено на односторонней печатной плате. Фотографии собранного устройства показаны на рис. 5.15 и 5.16.

Сначала припаивались все компоненты в корпусах SMD (резисторы и т. п.), потом перемычки, затем все светодиоды, транзисторы, гнездо для микроконтроллера и остальные компоненты.

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 9,6 МГц. Контроллер запрограммирован при помощи STK500 в режиме программирования ISP. Устройство работает в одном из трех режимов: NONE, NORMAL или HAPPY, которые отличаются способом мигания светодиодов. В листинге 5.2 показан главный бесконечный цикл программы. При инициализации значение переменной `a` устанавливается равным 20.

Листинг 5.2

```
while(1)
{
 if(mode==NORMAL)
 {
  PORTB &=0b11111000;
    //выключение
  mydelay(a+a+a+a);
  PORTB|=0b00000111;
    //включение
  mydelay(a+a);
```

```
  PORTB &=0b11111000;
    //выключение
  mydelay(a+a);
  PORTB |=0b00000111;
    //включение
  mydelay(a+a);
  time=0;
 }
 else if(mode==HAPPY)
 {
  if(state==1)
  PORTB |=1<<2;
  else if(state==5)
  PORTB &= ~(1<<2);
  if(state==5)
  PORTB &=~(1<<1);
    //Средний выключен
  else if(state==2)
  PORTB |=1<<1;
    //Средний включен
  if(state==4)
  PORTB &=~(1<<0);
    //Внешний выключен
  else if(state==3)
  PORTB |=1<<0;
    //Внешний включен
  mydelay(5*state);
  state++;
  if(state==6)
  state=1;
  time++;
  if(time==100)
  {
   mode = NORMAL;
   previous_mode = NONE;
   time=0;
   a=20;
  }
 }
 check();
}
```

Если установлен режим NORMAL, то система начинает включать и выключать светодиоды с некоторой задержкой, зависящей от значения `a`. Сердце состоит из трех частей: внешней, средней и внутренней. В режиме HAPPY включаются/выключаются светодиоды одной из этих частей сердца (в зависимости от состояния переменной `state`).

Сначала значение переменной `mode` устанавливается в `NONE`. Для обнаружения близости система вызывает функцию `check()` (листинг 5.3), которая изменяет значение `mode`. Значение `count` из `TIMER0` сохраняется в переменной `i` до того момента, пока в третьем бите PINB (который подключен к датчику) не появится логическая 1 (при обнаружении близости любого непрозрачного объекта, который препятствует падающему на светодиод свету). В зависимости от расстояния до этого объекта (которое можно определить по значению `count`) и от предыдущего режима (в котором работала система), режим меняется либо на NORMAL, либо на HAPPY.

Листинг 5.3

```
void check(void)
{
  TCCR0B = 0;//остановить таймер
  TCNT0 = 0;//сбросить таймер
  i=0;
  DDRB = 0b00010000;
  PORTB &= ~(1<<3);
  i=0;
  TCCR0B = 1<<CS02|1<<CS00;
    //Предварительное деление на 1024
  while((PINB&(1<<3))&&(i<80))
  {
   i=TCNT0;
  }
  if(i>50)
  {
if((previous_mode==NORMAL))
  {
   mode = HAPPY;
  }
  else a=20;
 }
 else if(i<50)
 {
  if(i<40)
  previous_mode = NORMAL;
 }
```

```
  else previous_mode = NONE;
  mode = NORMAL;
  a=i/3;
 }
 PORTB |=1<<3;
 DDRB = 0b00011111;
}
```

В остальных фрагментах кода выполняется обычная инициализация различных параметров. В листинге 5.4 приведено определение функции `mydelay`; если система работает на частоте 9,6 МГц, то она обеспечивает задержку в 10 мс (когда ее аргумент равен 1).

Листинг 5.4

```
void mydelay(int var)
//задержка в 10 мс для var=1 при 9,6 МГц
{
 unsigned char i1, j1, k1;

 for (i1=0; i1<var; i1++)
 for (j1=0; j1<251;j1++)
 for (k1=0; k1<50; k1++)
 asm("NOP");
}
```

Работа устройства

Для демонстрации чувствительного пульсирующего сердца расположите схему возле своей груди (спрячьте батареи в карман) и включите ее. Светодиоды начнут переключаться с обычной частотой. Теперь приблизьте руку к мигающим светодиодам, частота пульсаций начнет увеличиваться. Поднесите руку еще ближе к схеме и коснитесь ее, а потом уберите руку. Когда вы уберете руку, светодиоды начнут мигать в режиме "счастье" (HAPPY), а через несколько секунд снова вернутся в нормальный режим.

Проект 22. Электронная спичка без огня

Представьте себе спичку, которая после чирканья по коробку вспыхивает, но не загорается. Какой прок от такой спички? Она пригодится в театральных постановках и ее можно давать детям (которые не должны играть с огнем). Электронная спичка — это именно такое устройство, поскольку вы должны чиркнуть по коробку и только тогда она "вспыхнет". Для этого в устройстве есть катушка индуктивности (на спичке) и скрытый магнит (внутри коробка). На рис. 5.17 изображена блок-схема нашей спички.

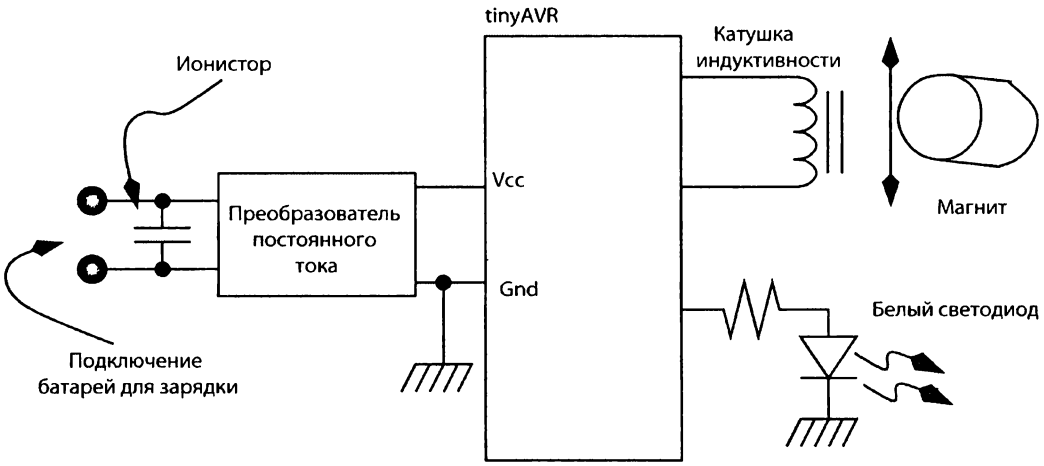


Рис. 5.17. Блок-схема электронной спички

Спецификация проекта

Цель — создать "перезаряжаемую спичку", для зажигания которой нужно чиркнуть по коробку. Электронная спичка, как и обычная, должна зажигаться на короткое время и затем гаснуть. Это, казалось бы, совершенно бесполезное устройство, тем не менее, находит множество применений (как в театре, так и для детей).

Описание устройства

Поскольку спичка должна загораться после чирканья по коробку, то возникают два вопроса: как подать питание на нее и как создать сигнал запуска, зажигающий спичку. Для питания спички есть несколько способов. Вполне естественным вариантом могут быть батарейки. Однако при этом потребуется включатель и, что еще важнее, решение проблемы, как спичку выключать? Разумеется, программа микроконтроллера может выключить спичку через определенное время, но от этого процесс горения спички лишится элемента случайности. Поэтому батарею мы решили заменить ионистором. Ионистор может запасти большое количество энергии, которой хватит для питания устройства. Преимущество такого подхода в том, что ионистор будет естественным образом разряжаться через схему, и таким образом спичка погаснет сама.

Второй вопрос: как запустить горение спички. Для этого используется простой трюк с катушкой индуктивности. Как мы уже знаем, перемещение катушки в магнитном поле создает напряжение на ее выводах. Если скорость изменения магнитного поля достаточно велика, то генерируемого напряжения будет достаточно для вызова прерывания микроконтроллера, а дальше микроконтроллер сделает все, что нам нужно.

На рис. 5.18 приведена принципиальная схема устройства. Для питания спички выбран ионистор на 10 Ф. Его напряжение рабочее 2,7 В, т. е. его можно зарядить

до этого напряжения, что позволяет запасти заряд в 27 кулонов. Для зарядки ионистора предусмотрена внешняя батарея (лучше всего подходят две последовательно соединенные щелочные или никель-металлогидридные батареи по 1,5 В). Ионистор подключен к преобразователю MAX756, который выдает 5 В. MAX756 можно настроить и для выдачи 3,3 В, но нам нужно управлять белым светодиодом, а для этого 3,3 В недостаточно. Для работы преобразователь использует катушку L1 (22 мкГн) и диод Шоттки (1N5819). После того как ионистор зарядится выше 1,2 В, преобразователь выдает напряжение 5 В на микроконтроллер Tiny13. Микроконтроллер получил питание и ждет внешнего запуска. Сигнал запуска приходит с катушки L2, индуктивность которой гораздо больше — 150 мГн. Когда спичку чиркают по коробку со скрытым магнитом, катушка выдает всплеск напряжения. Диод защищает микроконтроллер от выбросов отрицательной полярности. После запуска микроконтроллер выполняет код программы, которая случайным образом зажигает светодиод (точно так же, как в проекте светодиодной свечи). Светящийся светодиод потребляет гораздо больше энергии, чем микроконтроллер, поэтому ионистор быстро разряжается. После падения напряжения ионистора ниже 0,7 В преобразователь выключается, микроконтроллер прекращает работу и спичка "тухнет". В устройстве использован белый десятимиллиметровый светодиод, но подойдет и пятимиллиметровый.

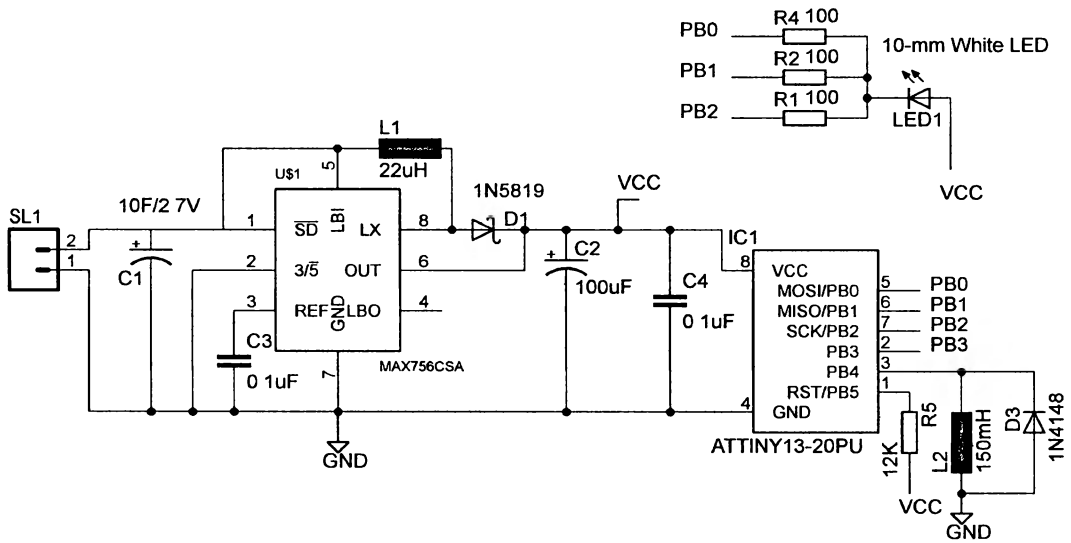


Рис. 5.18. Принципиальная схема электронной спички

Конструкция

Компоновку платы в программе EAGLE (и принципиальную схему) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Печатная плата для данного устройства нестандартная (из-за специфических требований). Плата (рис. 5.19) спроектирована так, чтобы она была длинной и узкой (напоминающей спичку) и заключена в прозрачную трубку из оргстекла (рис. 5.20). Сначала на плату были припаяны все компоненты в корпусах SMD, а потом все остальные компоненты. Длина трубки выбрана такой, чтобы светодиод выступал из нее. Светодиод залит термоклеем, который можно окрасить в красный цвет.

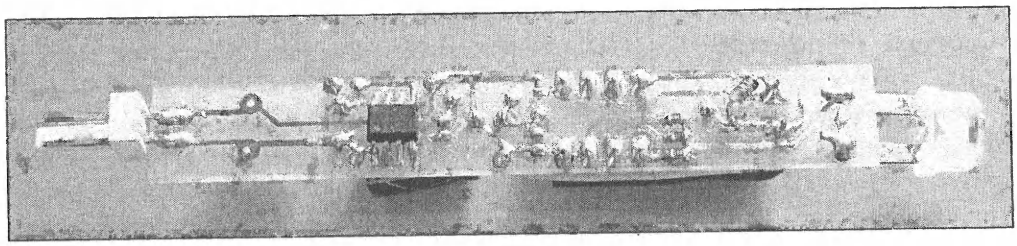


Рис. 5.19. Печатная плата электронной спички

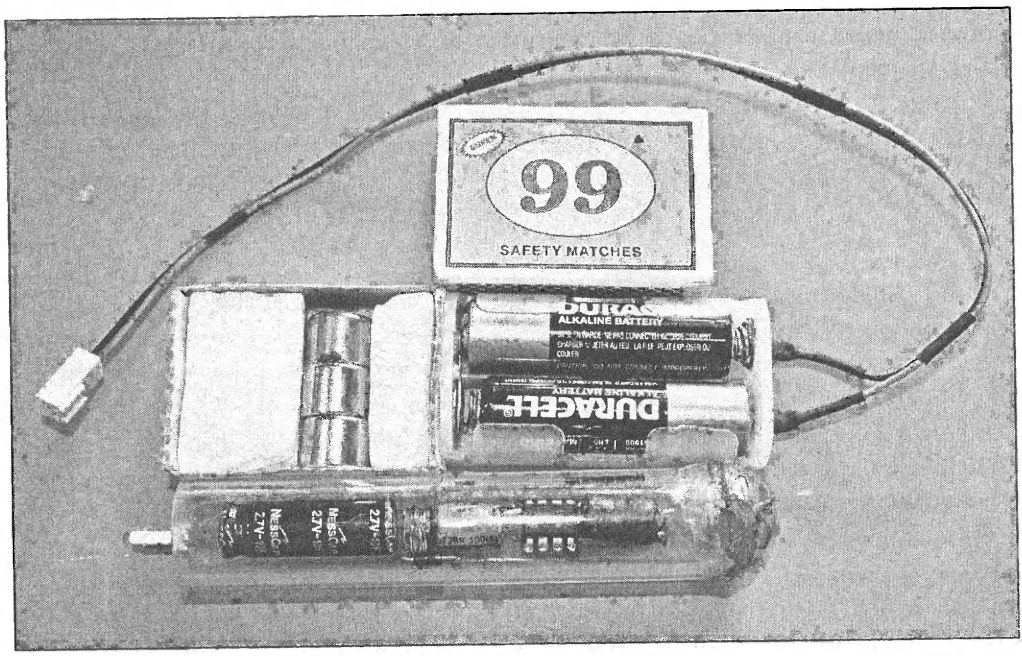


Рис. 5.20. Электронная спичка в сборе

Программирование

Откомпилированный код проекта (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 1,6 МГц. Главный бесконечный цикл программы приведен в листинге 5.5. Если переменная `mode` имеет значение `ON`, то система генерирует

псевдослучайную переменную lfsr (при помощи 32-разрядного сдвигового регистра LFSR с отводами от 32-го, 31-го, 29-го и первого разрядов). Это значение записывается в переменную temp (чтобы сохранить последнее состояние LFSR), а значение temp выводится на PORTB. Задержка системы тоже зависит от temp и поэтому она также псевдослучайна.

Листинг 5.5

```
while(1)
{
i=1;//Это сделано для игнорирования всех прерываний до этого
 if(mode==ON)
 {
  //Галуа
  lfsr = (lfsr >> 1) ^
  (-(lfsr & 1u) &
  0xd0000001u);
  /* отводы 32 31 29 1 */
  temp = (unsigned char)
  lfsr;
  DDRB= ~temp;
  PORTB = temp;

  temp = (unsigned char)
  (lfsr >> 24);
  _delay_loop_2(temp<<7);
 }
}
```

Значение переменной mode глобально устанавливается в OFF. Главная программа устанавливает переменную i в 1. Когда спичкой чиркают по коробку, в катушке возникает импульс напряжения, который прерывает процессор, и выполняется процедура обработки прерывания PCINT0. В коде этой процедуры значение mode устанавливается в ON, а маски GIMSK и PCMSK устанавливаются в 0x00 при помощи процедуры обработки прерывания (листинг 5.6). После возврата в главную программу в бесконечном цикле выполняется код LFSR, который зажигает светодиод случайным образом.

Листинг 5.6

```
ISR(PCINT0_vect)
{
 if(i==1)
 {
```

```
    mode = ON;
    GIMSK = 0x00;
    PCMSK = 0x00;
  }
}
```

Остальной код — различные инициализации, которые задают значения для используемых в программе масок и переменных.

Работа устройства

Для пользования спичкой нужно иметь специальный коробок со скрытым магнитом. Полярность магнита (какой полюс магнита направлен наружу) также важна. Ионистор в спичке нужно сначала зарядить. Для этого мы используем две соединенные последовательно батарейки размера AA. После подключения батареек к ионистору для его полной зарядки может потребоваться некоторое время. После зарядки ионистора (это можно проверить, измерив напряжение на нем, которое для нормальной работы спички должно быть не менее 2 В) можно чиркнуть спичкой по коробку. Как вы догадываетесь, не обязательно "физически" чиркать спичкой по коробку. Если вы быстро махнете спичкой рядом с коробком, в катушке появится всплеск напряжения и устройство сработает. Если вам не удается заставить спичку работать надлежащим образом, посмотрите видеозапись по ссылке: **www.avrgenius.com/tinyavr1**.

Проект 23. Волчок со светодиодами

Существует множество конструкций волчков со светодиодами. Обычно в них есть несколько светодиодов разного цвета и во время вращения эти цвета "размываются" на всю площадь волчка. Однако наше устройство отличается от всех других. Наш волчок при вращении отображает сообщение. И более того — если вы закрутите его в обратном направлении, то он покажет другое сообщение. Быстрое переключение светодиодов отображает сообщение, которое человеческий глаз способен уловить. Первоначальная идея была опубликована в журнале Elektor в декабре 2008 года (статья "LED Top with Special Effects"). Волчок из журнала Elektor мог показывать только одно сообщение. Мы изменили эту схему и внесли свои усовершенствования: наш волчок при вращении в разные стороны может показывать различные сообщения. Блок-схема волчка приведена на рис. 5.21. Устройство состоит из ряда светодиодов, смонтированных радиально от центра волчка к его периферии. Волчок питается от двух батареек размером AAA (никель-металлогидридных или щелочных), выдающих от 2,4 до 3 В. Поскольку микроконтроллер должен питаться от 5 В, то предусмотрен повышающий преобразователь (который поднимает напряжение от батареек до +5 В). Для определения факта вращения и его направления в схеме есть две катушки индуктивности.

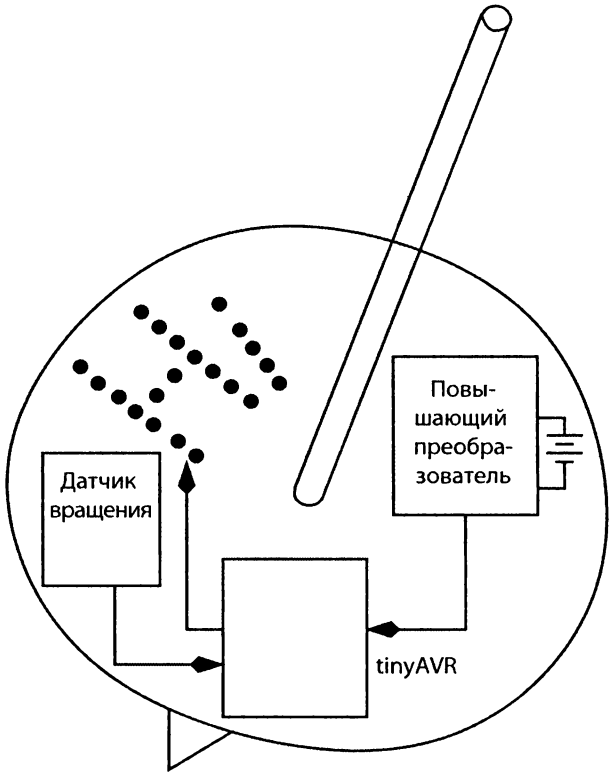


Рис. 5.21. Блок-схема волчка со светодиодами

Спецификация проекта

Цель проекта — создать вращающийся волчок со светодиодами, который показывает сообщения при вращении. Волчок работает от батареек и при вращении в разные стороны будет выдавать различные сообщения. В схеме восемь светодиодов, которые отображают как текст, так и графическую информацию.

Описание устройства

На рис. 5.22 изображена принципиальная схема волчка со светодиодами. Он питается от двух батареек размера AAA (на схеме — AAA1 и AAA2). Выключатель SW2 позволяет отключать питание. Учтите, что индикатора включения питания нет, поэтому весьма возможно, что вы забудете выключить питание и разрядите батарейки.

Напряжение от батареек подается на преобразователь MAX756, который обеспечивает напряжение +5 В для питания микроконтроллера AVR ATTiny44 (IC3), а также двух операционных усилителей LM358 (IC1 и IC2). Микроконтроллер управляет восемью светодиодами. Цепь обнаружения движения (состоящая из двух операционных усилителей) определяет факт вращения и направление поворота. Эта информация подается в микроконтроллер на сигнальные контакты INT (контакт PB2) и DIR (контакт PB1).

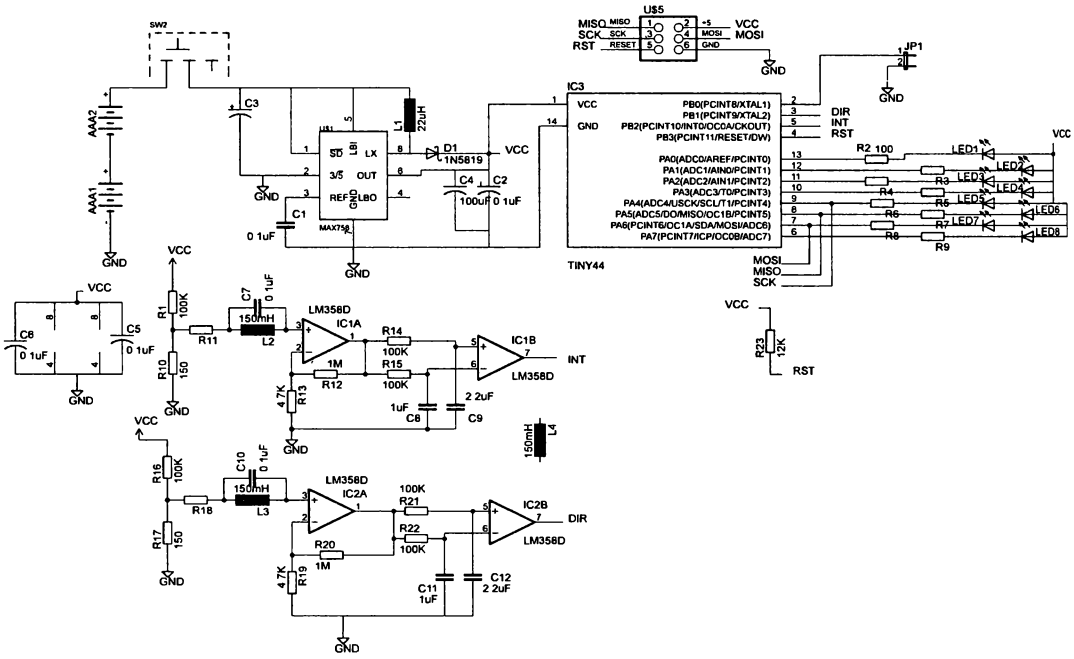


Рис. 5.22. Принципиальная схема волчка со светодиодами

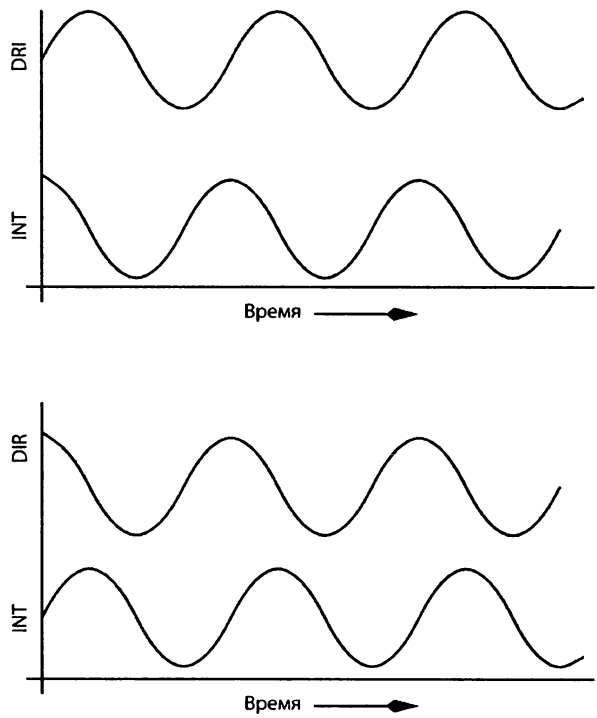


Рис. 5.23. Осциллограмма напряжения на катушках

Цепь обнаружения движения состоит из двух идентичных каналов (состоящих из катушек L2 и L3). Когда катушка двигается в магнитном поле, в ней возникает напряжение. Наша схема использует магнитное поле Земли, поэтому при вращении волчка генерируется очень небольшое переменное напряжение. Частота этого напряжения равна скорости вращения волчка. Две катушки размещены под углом 90° на краю волчка. Таким образом, сигнал от одной катушки запаздывает на 90° по сравнению с сигналом другой катушки. Если рассмотреть эти сигналы на осциллографе, то при вращении волчка в одну сторону первый сигнал будет опережать второй, а при вращении в другую сторону — отставать от него (рис. 5.23).

Синусоидальные сигналы от катушек усиливаются операционными усилителями, включенными как неинвертирующие усилители (IC1A и IC2A). Эти усиленные сигналы преобразуются при помощи RC-цепей задержки (R14, C9 и R15,C8 — в одном канале и R21, C12 и R22, C11 — в другом). Задержанные синусоидальные напряжения поступают на компараторы (IC1B и IC2B), чтобы получить прямоугольные сигналы той же частоты. Прямоугольные сигналы от операционных усилителей (INT и DIR) подаются на микроконтроллер. Сигнал INT поступает на контакт PB2 микроконтроллера Tiny44 (это контакт прерывания). Этот контакт программно сконфигурирован как вход по нарастающему фронту. Второй канал DIR подключен к PB1. При каждом прерывании микроконтроллера сигналом INT он выполняет процедуру обработки прерывания, которая читает состояние контакта DIR. Если контакт DIR равен 0, то это означает одно направление вращения, а если 1 — другое. Микроконтроллер также измеряет частоту сигнала INT и использует эту информацию для определения времени одного оборота волчка. Шаблоны свечения светодиодов (для обоих направлений) хранятся в памяти программ микроконтроллера. На основе информации о скорости вращения микроконтроллер решает, как долго демонстрировать шаблон свечения. Если скорость вращения падает, то время отображения шаблона свечения пропорционально увеличивается. Если скорость вращения высокая (как это бывает в начале вращения), то все шаблоны свечения демонстрируются в течение более короткого времени. Это обеспечивает постоянство отображаемого сообщения независимо от скорости вращения волчка.

Шестиконтактный разъем — ISP, который необходим для программирования микроконтроллера. Перемычка JP1 зарезервирована для последующего использования и в текущей версии программы состояние контакта PB0 не считывается. Катушка L4 установлена для балансировки волчка, к схеме она не подключена.

Конструкция

Компоновку платы в программе EAGLE (и принципиальную схему) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Волчок сделан на нестандартной печатной плате. На рис. 5.24 и 5.25 показаны обе ее стороны.

Все индикаторы размещены на нижней стороне платы и залиты термоклеем для защиты. Катушки L2 и L3 намотаны эмалированным медным проводом марки 42 SWG на сердечнике из феррита. Это очень тонкая проволока, поэтому при нама-

тывании катушки следует соблюдать осторожность. Ферритовый сердечник был полностью заполнен обмоткой, в результате чего индуктивность составила примерно 150 мГн. Подробности по сердечнику (высота 10 мм и внутренний диаметр 3 мм) и способ его намотки приведены на нашем Web-сайте. Катушка L1 намотана 20 витками провода 28 SWG и имеет индуктивность примерно 22 мкГн. Число витков катушки L4 не имеет значения, она просто должна иметь такой же вес, как остальные катушки. Все катушки размещаются на краю платы (через 90 друг от друга). Фотография вращающегося волчка приведена на рис. 5.26.

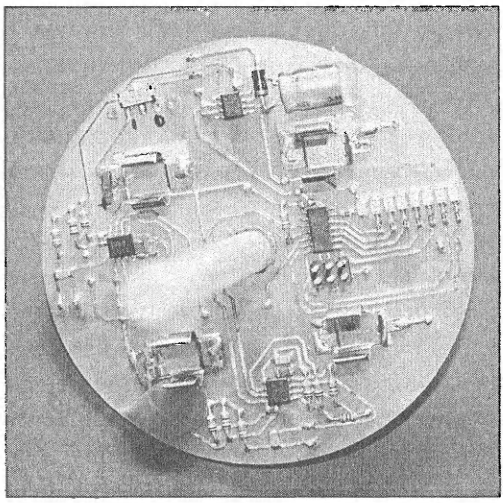


Рис. 5.24. Печатная плата устройства (верхняя сторона)

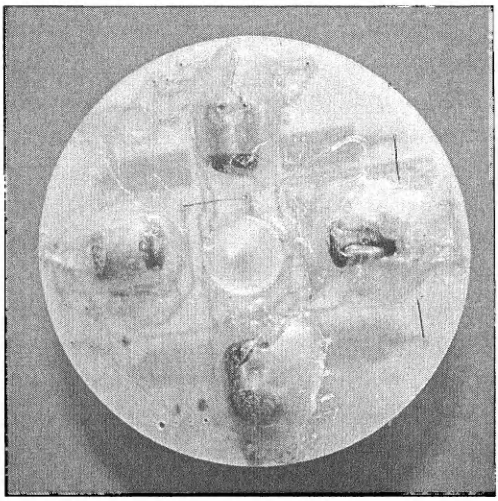


Рис. 5.25. Печатная плата устройства (нижняя сторона)

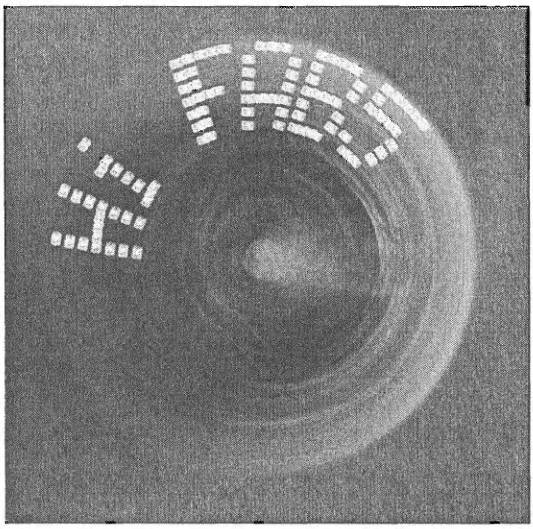


Рис. 5.26. Внешний вид вращающегося волчка

Сначала припаиваются все компоненты в корпусах SMD, за ними — все остальные компоненты, разъемы батарей и катушки (с обратной стороны). Пластмассовый стержень, выточенный под размер центрального отверстия, служит осью волчка.

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 1 МГц. Максимальное и минимальное время одного оборота задано заранее (как и те символы, которые будут отображаться при вращении в одну и в другую сторону). Главная программа инициализирует прерывания и входит в бесконечный цикл, из которого периодически вызывается функция `double_string_display` (листинг 5.7).

Листинг 5.7

```
void double_string_display (void)
{
  // Инициализируем дисплей
  construct_display_field();
  // Выполняется TOP
  while (running_condition() == TOP_TURNING)
  {
   i = current_column/2;
   if(current_column%2==1)
   {
    set_leds(LED_ALL_OFF);
   }
   else if(current_column%2==0)
   {
    if(mode==CLOCKWISE)
    {
     if(i<=(STRING_LENGTH1*6))
     set_leds((display_field_clock[i])&0x7F);
     else
     set_leds(LED_ALL_OFF);
    }
    else if(mode==ANTICLOCKWISE)
    {
     if(((STRING_LENGTH2*6)-i)>=0)
     set_leds((display_field_clock[(STRING_LENGTH2*6)-i])&0x7F);
     else
```

```
      set_leds(LED_ALL_OFF);
     }
    }
   }
   // TOP не выполняется
   while(running_condition() != TOP_TURNING)
   {
    // Отключить все светодиоды
    set_leds(LED_ALL_OFF);
   }
} /* double_string_display */
```

Нужный шаблон свечения на светодиодах (в соответствии с направлением вращения и с отображаемым столбцом) формирует функция `set_leds`. Функция `construct_display_function` заполняет массив `display_field_clock` (в соответствии с текстом и направлением вращения). Значения таймера и тактовой частоты (для правильного отображения строк) изменяются прерываниями INT0, TIMER0 и TIMER1 (как и направление вращения). Каждый второй столбец дисплея оставляется пустым (чтобы столбцы не сливались из-за высокой скорости вращения волчка). Отношение пустых столбцов к отображаемым определяет ширину выводимых символов.

Работа устройства

После изготовления волчка подключают батарейки и микроконтроллер программируется (при помощи интерфейса программирования ISP) кодом, который можно скачать с нашего Web-сайта. После программирования микроконтроллера кабель ISP убирают и включают питание. Раскрутите волчок и запустите его на ровной твердой поверхности. Вы увидите сообщения. Возможно, вам придется потренироваться в его раскручивании. Пусть волчок остановится, после чего закрутите его в другом направлении, и увидите другое сообщение. Не забудьте выключить питание, когда волчок не используется.

Проект 24. Бесконтактный тахометр

Для измерения скорости вращения мотора обычно требуется физический контакт тахометра с осью. Однако иногда весьма желательно измерять скорость вращения мотора без всякого физического контакта. В некоторых схемах бесконтактных тахометров для этого применяется зеркало, прикрепленное к вращающейся части мотора — оно отражает луч лазера, который потом регистрируется электронной схемой. Еще один способ бесконтактного измерения — это замер наведенного напряжения на проводе свечи зажигания двигателя.

В предлагаемой схеме бесконтактного тахометра есть небольшой перемещающийся магнит, и мы измеряем период напряжения, генерируемого им в неподвижной катушке. При таком способе измерительная цепь с мотором не связана (ни электрически, ни физически). Блок-схема устройства показана на рис. 5.27.

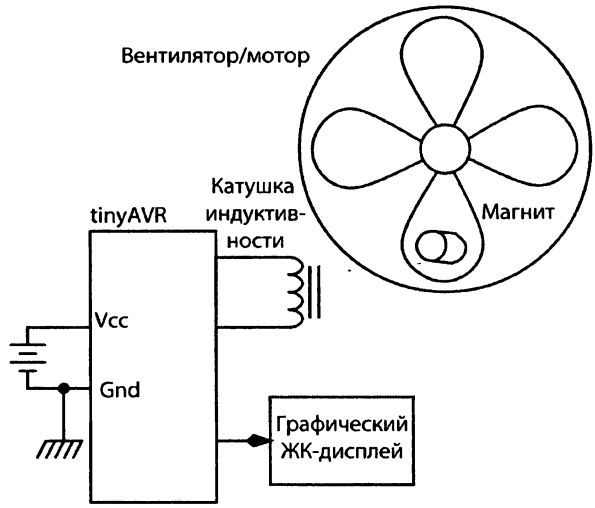


Рис. 5.27. Блок-схема тахометра

Спецификация проекта

Цель проекта — создать тахометр, который пригоден для измерения частоты вращения вентиляторов, моторов и т. п. без всякого механического или электрического контакта с объектом. Система должна работать от батареек и иметь графический дисплей для отображения необходимой информации.

Описание устройства

Схема состоит из катушки с большой индуктивностью (100 мГн) и операционного усилителя (аналогично схеме вращающегося волчка). Сигнал с усилителя проходит через два фильтра с разными постоянными времени, чтобы создать небольшую разницу фаз между отфильтрованными выходными сигналами. Эти два сигнала с разными фазами подаются на другой операционный усилитель для преобразования в импульсный выходной сигнал. Сигнал с катушки зависит от переменного магнитного поля. Если схему разместить рядом с закрепленным на валу мотора магнитом, то при его вращении магнитное поле в катушке начнет изменяться и на ее выводах появится синусоидальное напряжение.

На рис. 5.28 изображена принципиальная схема бесконтактного тахометра. Наведенное в катушке напряжение усиливается операционным усилителем IC3-A (LM358), а выходной сигнал проходит через два RC-фильтра (соединенные параллельно) с разными постоянными времени. Выходы этих фильтров соединены со входами усилителя IC3-B (LM358). Он преобразует синусоидальный сигнал с катушки в прямоугольные импульсы той же самой частоты (рис. 5.29). Далее импульсы подаются на микроконтроллер AVR Tiny45, который измеряет частоту сигнала на контакте 3 (сконфигурированном как входной). Остальные контакты микроконтроллера управляют точечно-матричным дисплеем от мобильного телефона Nokia 3310. Тахометр питается от батареи в 9 В, подключенной к разъему SL3.

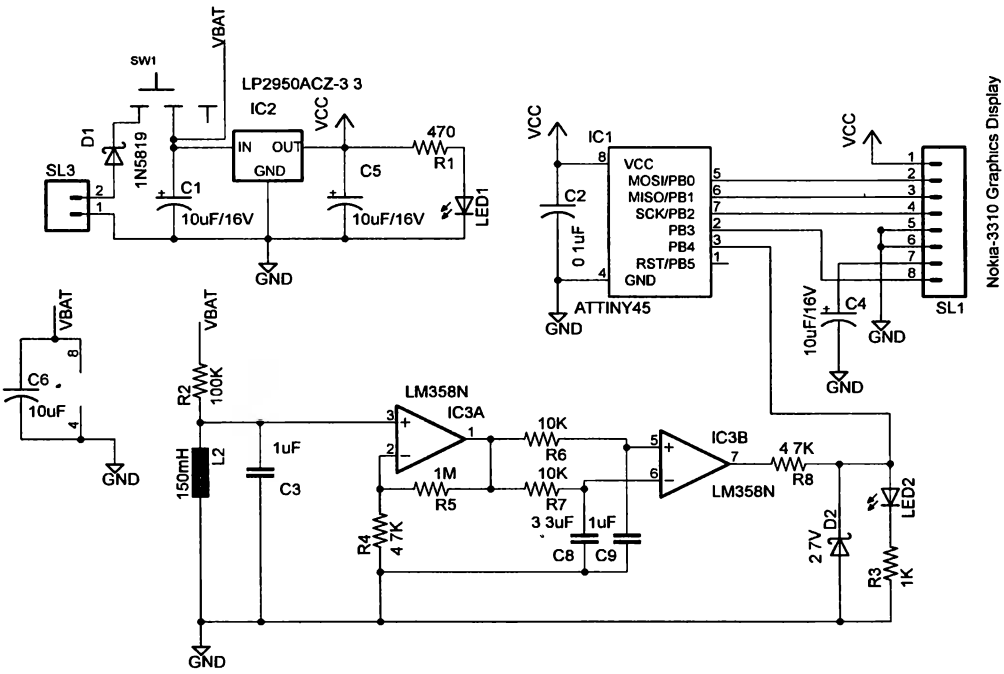


Рис. 5.28. Принципиальная схема бесконтактного тахометра

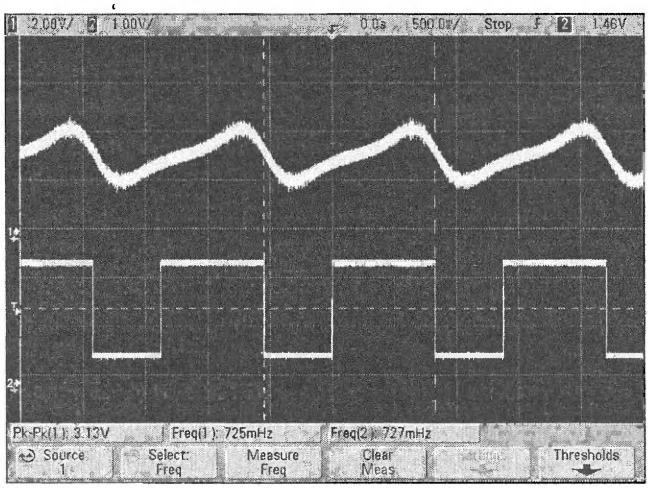


Рис. 5.29. Осциллограмма сигналов бесконтактного тахометра

Монохромный дисплей не имеет подсветки, поэтому по обеим сторонам дисплея установлены белые светодиоды (чтобы было видно при плохом освещении).

Схема питается от батарейки в 9 В, подключенной к разъему SL3; от этого же напряжения питается и операционный усилитель LM358. Для дисплея Nokia требуется напряжение от 3 до 3,3 В, поэтому питание для него (и для микроконтроллера)

обеспечивает стабилизатор LM2950-3.3V. Выходной сигнал с усилителя проходит через резистор и стабилитрон на 3 В, чтобы ограничить амплитуду сигнала, подаваемого на микроконтроллер.

Конструкция

Компоновку платы в программе EAGLE (а также ее принципиальную схему) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Обе стороны платы показаны на рис. 5.30 и 5.31. На рис. 5.32 изображено полностью готовое устройство.

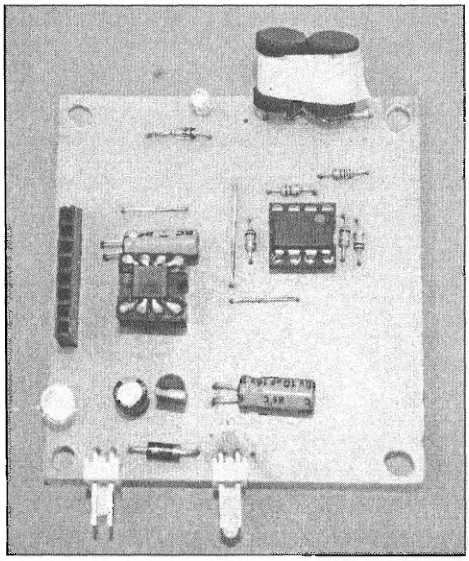


Рис. 5.30. Плата бесконтактного тахометра (сторона компонентов)

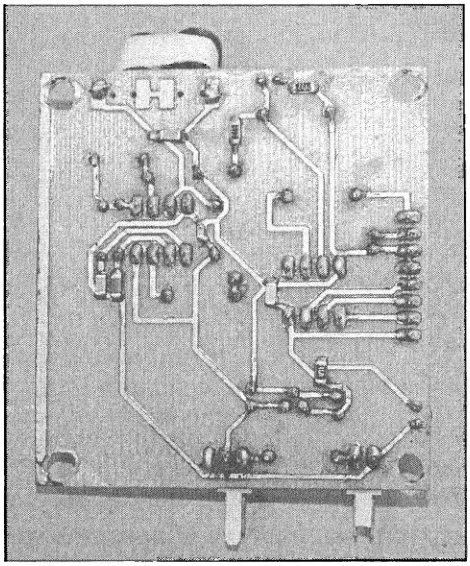


Рис. 5.31. Плата бесконтактного тахометра (сторона печатных проводников)

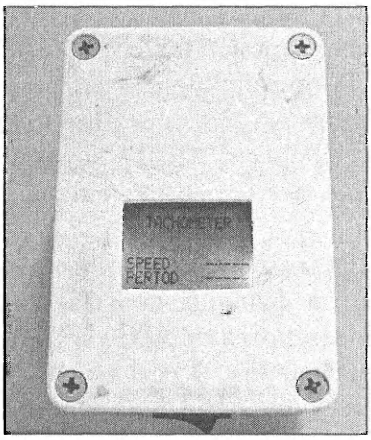


Рис. 5.32. Внешний вид готового устройства

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Максимальное время одного оборота определено заранее. Функция `running_condition` проверяет время одного оборота (при помощи `period_count`) и сравнивает с этим максимальным временем. Если максимальное время больше значения `period_count`, то двигатель вращается; если меньше — двигатель остановился. Значения `period_count` и `period_total` обновляются в прерывании по переполнению TIMER0 (листинг 5.8).

Листинг 5.8

```
ISR(TIM0_OVF_vect)
{
 //Каждые 9.984 мс
 TCNT0 = 255-78;
 period_total+=9.984;
 if(period_count<5500)
 period_count+=9.984;
 else
 period_count=5500;
}
```

Вектор прерывания PC0 обеспечивает основную работу тахометра (реальное измерение количества оборотов в минуту и длительности периода). Он проверяет присутствие низкого уровня на четвертом контакте PINB, и если этот уровень сохраняется в течение 20 мкс, то система считает, что один период прошел, и увеличивает глобальную переменную `no` на единицу. Когда значение `no` достигает 6 (прошло шесть периодов), контроллер вычисляет число оборотов в минуту и период вращения. Если значение переменной `period_total` меньше 360 мс, то программа не делает ничего. В противном случае осуществляется переход в тот блок, где при помощи значения `period_total` вычисляется число оборотов и период вращения (и включает дисплей). После выхода из этого блока программа опять сбрасывает значения `period_total` и `no` в нуль. Код этого блока приведен в листинге 5.9.

При помощи функции `display` тахометр отображает число оборотов в минуту и период вращения. Когда эта функция вызывается со значением переменной `what`, равным четырем, отображаются обороты в минуту, а когда `what` равна пяти — отображается период вращения. В остальной части кода выполняется присваивание начальных значений различным переменным (для правильной работы программы).

Листинг 5.9

```
if(!(PINB&(1<<4)))
{
 _delay_us(20);//Ждать стабилизации
 if(!(PINB&(1<<4))) //Проверить еще раз на низкий уровень
 {
  flag=0;//Режим отображения по истечении in
  no++;
  period_count=0;
  if(no==6) //Прошло 6 периодов
  {
   /*** ЭТОТ БЛОК ВЫЧИСЛЯЕТ СКОРОСТЬ И ЧИСЛО ОБОРОТОВ В МИНУТУ
    ПО РЕЗУЛЬТАТАМ ЗАМЕРА ***/
   if((period_total/(no))<60);
   //ничего не делать, если период меньше 60 мс.
   //Фильтрация 2000 оборотов в минуту.
   else
   {
    frequency = (12.0/period_total)*1000;
    rpm = frequency*60;//число оборотов в минуту
    period_actual = period_total/12;
    displayon=1;//дисплей включен
   }
   period_total=0;no=0;
  }
 }
}
```

Работа устройства

Для проверки работоспособности необходимо включить питание и поднести устройство к движущемуся объекту (вентилятору или валу мотора). К этому объекту должен быть заранее прикреплен небольшой магнит. Дисплей покажет частоту и период вращения.

Проект 25. Индуктивный датчик появления автомобиля и счетчик

Вас никогда не удивляло, что в момент приближения вашего автомобиля к светофору красный сигнал в нем меняется на зеленый? Или что при въезде на территорию автомобильного ресторана автоматически звучит приветственное сообщение?

Не удивляйтесь, поскольку в этом разделе тоже предлагается схема, которая позволит вам обнаружить приближающийся автомобиль. Существует много способов обнаружения автомобиля. Обычно автомобиль имеет большой металлический кузов и в присутствии такой большой массы металла электрические характеристики катушки индуктивности изменяются. Замерив это изменение характеристик, мы сможем обнаружить присутствие автомобиля. Блок-схема подобной системы приведена на рис. 5.33.

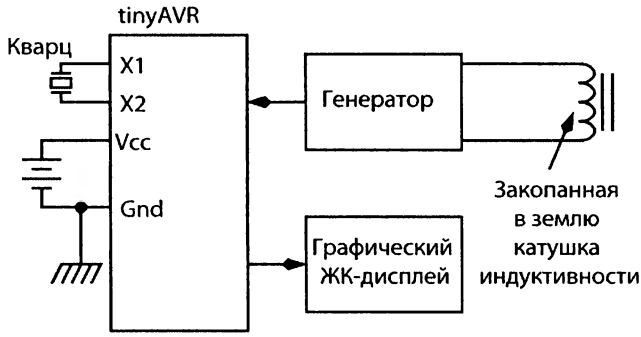


Рис. 5.33. Блок-схема датчика приближения автомобиля

Катушка индуктивности, выполненная из нескольких витков изолированного медного провода, укладывается в том месте, где будут проезжать машины. Катушку обычно закапывают под землю и закрывают асфальтом или бетоном. Катушка включена в цепь колебательного контура генератора. Когда большой металлический объект (такой, как машина или грузовик) проезжает над катушкой, ее индуктивность уменьшается, а частота колебаний увеличивается. Микроконтроллер обнаруживает повышение частоты колебаний и после превышения определенного предела выдает сигнал, что проехала машина.

Спецификация проекта

Цель проекта — создать устройство, которое обнаруживает присутствие автомобилей и подсчитывает их количество. Система питается от батареек; графический дисплей показывает число обнаруженных автомобилей.

Описание устройства

Устройство состоит из двух частей: генератора (рис. 5.34) и схемы с микроконтроллером и дисплеем (рис. 5.35). В состав генератора входят два конденсатора (C1 и C2) и катушка индуктивности (подключенная к контактам X1-1 и X1-2 показанного на рисунке разъема). Частота колебаний равна $F = 1/(2\pi\sqrt{LC})$. $C = C1 \cdot C2/(C1 + C2)$ — это эквивалентная емкость. Катушка индуктивности состоит из нескольких витков изолированного медного провода, заключенных в защитный

пластмассовый корпус, который затем закапывается в землю. Большое значение имеет размер катушки: она должна быть примерно шесть футов (≈1,8 м) в длину и четыре фута (≈1,2 м) в ширину. Десять витков медного провода дадут индуктивность от 500 до 1000 мкГн. Если выбрать емкости C1 = C2 = 1000 пФ, то частота генератора составит от 200 до 300 кГц. Такую частоту микроконтроллер AVR способен замерить без труда.

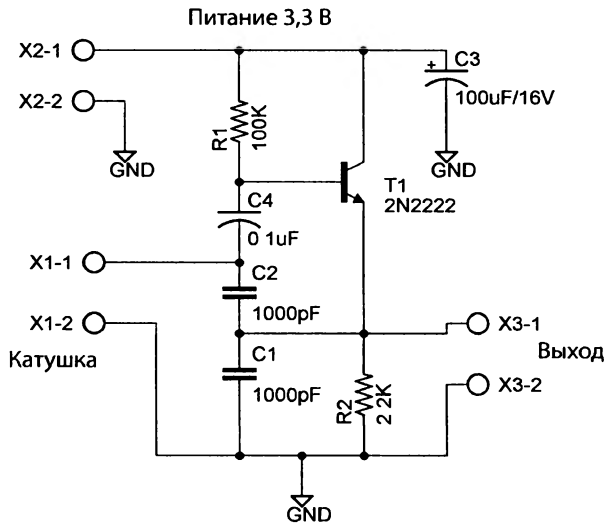


Рис. 5.34. Принципиальная схема генератора

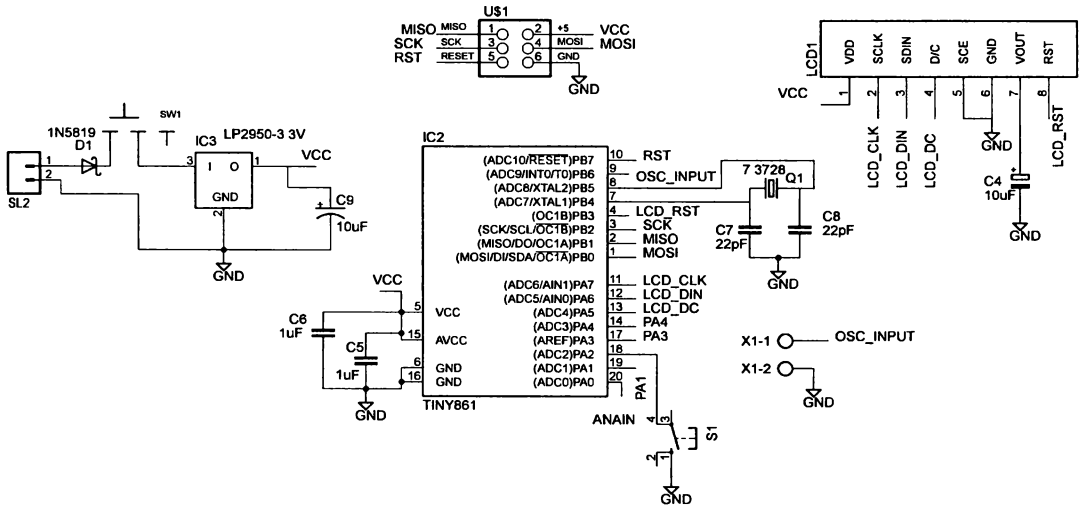


Рис. 5.35. Принципиальная схема блока обработки

Сигнал с генератора подается на микроконтроллер (рис. 5.32) Tiny861 с тактированием от кварца 7,37 МГц. Выбор такой частоты случаен. Просто такой кварц с частотой менее 8 МГц, оказался у нас под рукой (поскольку максимальная частота имевшейся у нас версии Tiny861 составляла 10 МГц). Устройство питается от батарейки и преобразователя LP2950-3.3V, поскольку для графического дисплея Nokia 3310 требуется напряжение питания 3,3 В. Помимо этого, предусмотрен разъем ISP для программирования микроконтроллера.

Конструкция

Компоновку плат в программе EAGLE (а также их принципиальные схемы) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Схема с микроконтроллером собрана на специально изготовленной печатной плате. Генератор собран на стандартной монтажной плате. Обе схемы соединены двумя разъемами (питание и сигнал генератора). На плате генератора предусмотрен еще разъем для подключения индуктивности. На рис. 5.36 и 5.37 показаны платы генератора и контроллера, на рис. 5.38 — внешний вид готового устройства вместе с катушкой индуктивности.

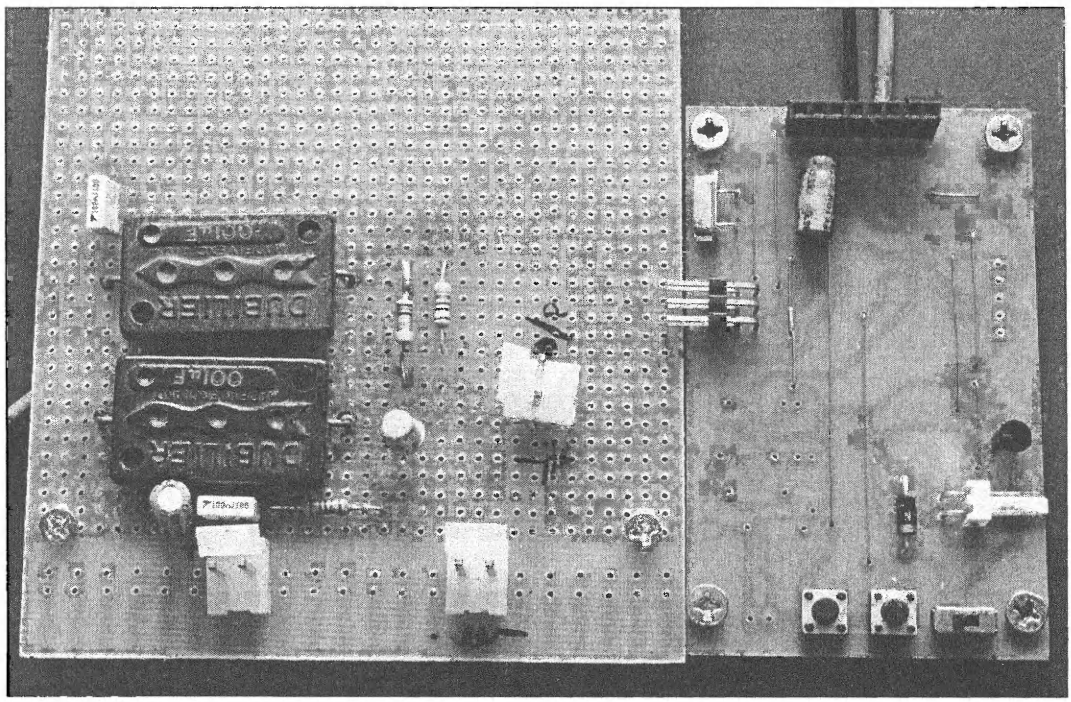


Рис. 5.36. Печатные платы генератора и микроконтроллера (сторона компонентов)

Рис. 5.37. Печатные платы генератора и микроконтроллера (сторона печатных проводников)

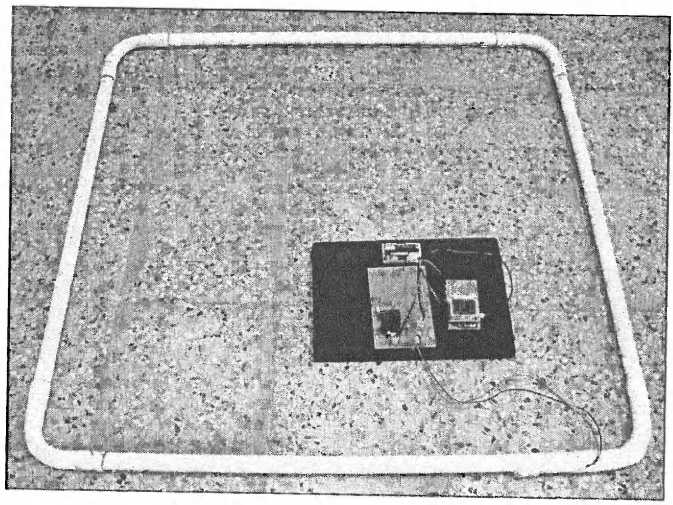


Рис. 5.38. Внешний вид готового детектора автомобилей

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Таймер Timer 1 инициализируется в режиме CLEAR TIMER ON COMPARE MATCH. Значение OCR1A устанавливается в 1 секунду, из которой 0,1 секунды выделяется для Timer 0 для подсчета внешних импульсов (приходящих из катушки) и 0,9 секунды система находится в состоянии конечного автомата (листинг 5.10).

Листинг 5.10

```
TIMSK |= (1<<OCIE1A); //активизировать прерывание сравнения
TCCR1B |= ((1<<WGM12) | (1<<CS12)); //предварительный делитель 256,
                                    // WGM для режима CTC
OCR1A = 31250; //для прерывания в 1 секунду
sei();         //активизировать глобальное прерывание
```

В ветви s0 (листинг 5.11) настраивается базовая частота в отсутствие автомобиля (когда пользователь нажимает кнопку 1 схемы).

В ветви s1 (листинг 5.12) проверяется появление автомобиля. Если частота превысит суммарное значение базовой частоты и верхнего предела частоты (который пользователь может настроить по своему усмотрению в файле detector.h (по умолчанию это 6 кГц)), то система считает, что появился автомобиль, и поэтому включает зеленый светодиод и выключает красный. После этого управление передается в ветвь s2.

Ветвь s2 (листинг 5.13) отслеживает отъезд автомобиля. Если частота падает ниже суммарного значения базовой частоты и нижнего предела частоты, который пользователь может настроить по своему усмотрению в файле detector.h (по умолчанию это 2 кГц), то система считает, что автомобиль уехал, и поэтому выключает зеленый светодиод и включает красный. Счетчик числа автомобилей увеличивается на единицу и после этого управление передается обратно в ветвь s1.

Листинг 5.11

```
Ветвь s0
if( !(SWITCH_PIN&(1<<2)) ) //При нажатии кнопки
{
base_freq = changing_freq; //установить базовую частоту
display_base_freq(base_freq); //показать ее на дисплее
select_case = s1; //Выбрана ветка s1
}
```

Листинг 5.12

Ветвь s1

```
while( changing_freq > (base_freq+upper_threshold_freq) )
//Если частота превышает предел
{
select_case = s2; //Выбрана ветка s2
LED_PORT &= ~(1<<LED_GREEN); //Включение зеленого светодиода
LED_PORT |= (1<<LED_RED); //Выключение красного светодиода
break;
}
```

Листинг 5.13

Ветвь s2

```
while( changing_freq < (base_freq+lower_threshold_freq) )
//Если частота вернулась в норму
{
 LED_PORT &= ~(1<<LED_RED); //Включение красного светодиода
 LED_PORT |= (1<<LED_GREEN); //Выключение зеленого светодиода

 car_counter = car_counter+1; //Увеличение счетчика машин
 display_car(car_counter); //Показать количество машин
 select_case = s1; //Выбрана ветка s1
 break;
}
```

В процедуре обработки прерывания для Timer 1 (листинг 5.14) происходит инициализация Timer 0, который срабатывает от внешних импульсов, приходящих на контакт T0. Timer 0 считает приходящие импульсы.

Мы подсчитываем число импульсов в течение 0,1 секунды, после чего вычисляем частоту и отображаем ее на экране графического дисплея.

Листинг 5.14

```
ISR(TIMER1_COMPA_vect)
  //Настраиваем процедуру обработки прерывания на режим Compare
{
  //Подсчитываем число импульсов за 0,1 секунды
   ovf_counter = 0;
   //Настраиваем Timer 0 для внешних импульсов
   TCCR0 = ((1<<CS02) | (1<<CS01) | (1<<CS00));
   //Активируем режим Normal, внешняя тактовая частота
   //увеличивает TCNT0
```

```
TCNT1=0;
TCNT0=0;
while((3125)>=(uint32_t)TCNT1)
{
 if(TCNT0==255)
 {
  ovf_counter += 1;
  TCNT0=0;
 }
 counts = TCNT0;
}
 //Вычисления для определения текущей частоты
 counts = (ovf_counter * 255) + counts;
 freq=(float)counts;
 freq=freq*10;
 counts=(uint32_t)freq;
 counts=counts/10;
 changing_freq = counts;
 LCD_partclear();
 display_changing_freq();
}
```

Фрагмент кода листинга 5.15 содержится в файле detector.h, и пользователь должен изменить его в соответствии со своими потребностями. Здесь заданы верхний и нижний пределы частот, которые используются для определения появления и отъезда машины.

Листинг 5.15

```
#define upper_threshold_freq 600
//В килогерцах, делим частоту на 10
//то есть в данном случае это 6 кГц (6000/10 = 600)
#define lower_threshold_freq 200
//В килогерцах, делим частоту на 10
//то есть в данном случае это 2 кГц (2000/10 = 200)
```

Работа устройства

Для запуска нашей системы необходимо уложить индуктивную петлю в землю. После ее укладки и подключения к генератору включаем контроллер. Он показывает значение счетчика (сброшенное в нуль), а когда к катушке индуктивности приближается автомобиль, на дисплее индицируется его присутствие. Когда автомобиль отъезжает, значение счетчика увеличивается на единицу.

Проект 26. Электронные свечи для дня рождения

Мы хотели создать светодиодные свечи для дня рождения, которые можно было бы задувать точно так же, как и обычные. Такая система хорошо подходит для маленьких детей (поскольку не огнеопасна). Для обнаружения дуновения использован термистор (показан на рис. 5.39). Микроконтроллер управляет светодиодами и может менять интенсивность их свечения случайным образом, что создает впечатление мерцания свечей. Устройство можно объединить с проигрывателем рингтонов (смотрите следующую главу), который после задувания всех свечей будет воспроизводить веселую мелодию.

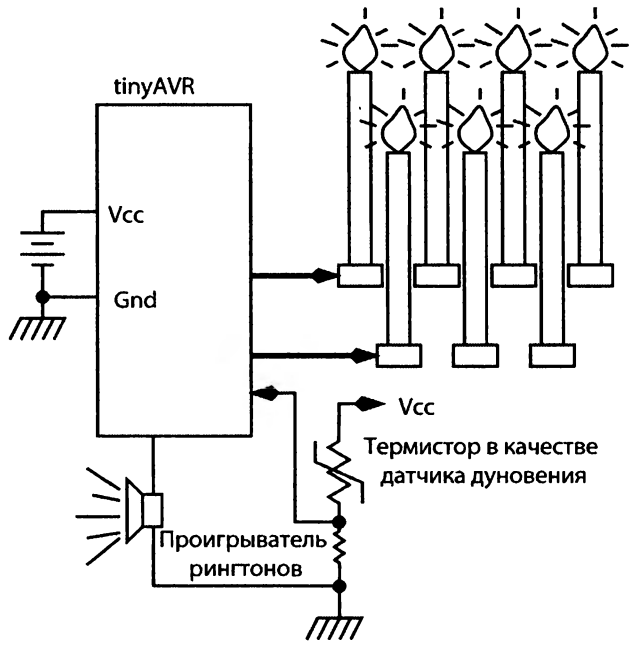


Рис. 5.39. Блок-схема электронных свечей

Спецификация проекта

Цель проекта — создать светодиодные свечи, которые можно задуть (точно так же, как обычные восковые). Главная задача — создать безопасное в пожарном отношении устройство для детей.

Описание устройства

Принципиальная схема устройства приведена на рис. 5.40.

Она состоит из микроконтроллера Tiny44, который имеет 12 контактов ввода/вывода, но поскольку один контакт предназначен для RESET, то остается 11 сво-

бодных контактов. Схема питается от внешних батарей (рекомендуем взять четыре никель-металлогидридных батареи размером AA) и не имеет стабилизаторов напряжения (для упрощения). Используется 20 светодиодов, собранных в матрицу 4×5. Аноды светодиодов подключены к источнику питания через *p-n-p*-транзисторы. Катоды соединены с выводами микроконтроллера. Светодиоды мультиплексируются на большой частоте. Чтобы добиться эффекта мерцания, средний ток через светодиоды меняется при помощи программной ШИМ. В системе предусмотрен делитель напряжения из резистора R10 (сопротивлением 150 Ом) и термистора R9 (с номинальным сопротивлением 150 Ом). Поблизости от термистора размещен небольшой резистор (для нагрева). Напряжение на R9, R10 постоянно отслеживается контактом PA7 микроконтроллера. Если подуть на термистор, то он остынет и напряжение на контакте PA7 уменьшится. Если падение напряжения больше запрограммированного предела, то микроконтроллер посчитает, что на схему кто-то дует, и начинает выключать светодиоды (случайным образом). Когда вы дуете на обычные свечи, одни из них гаснут сразу, а остальные лишь мерцают (и вам может даже не удастся задуть ни одной свечи). Поэтому таким случайным выключением мы имитируем поведение настоящих свечей.

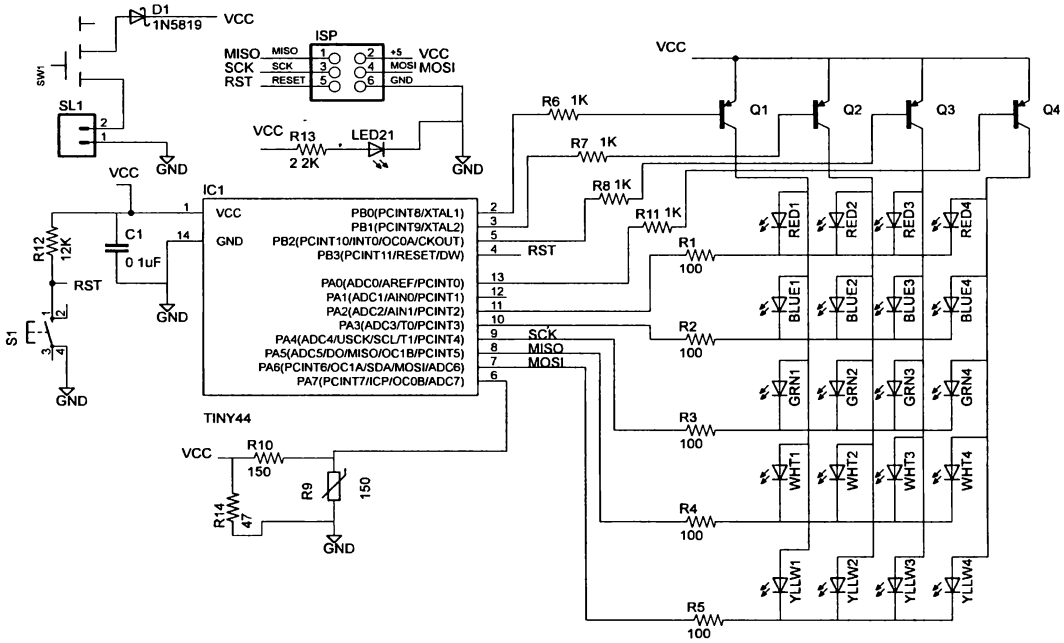


Рис. 5.40. Принципиальная схема устройства

Один из контактов микроконтроллера (PA1) не задействован и может применяться для любых целей. Например, к нему можно подключить проигрыватель рингтонов.

Конструкция

Компоновку платы в программе EAGLE (и ее принципиальную схему) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Плата разведена в двух слоях и поэтому ее нельзя изготовить при помощи станка Roland Modela MDX 20. Плата была специально вырезана в виде круга. На рис. 5.41 и 5.42 показаны обе стороны платы. На нашем сайте **www.avrgenius.com/tinyavr1** есть видеозапись, которая демонстрирует работу этой системы.

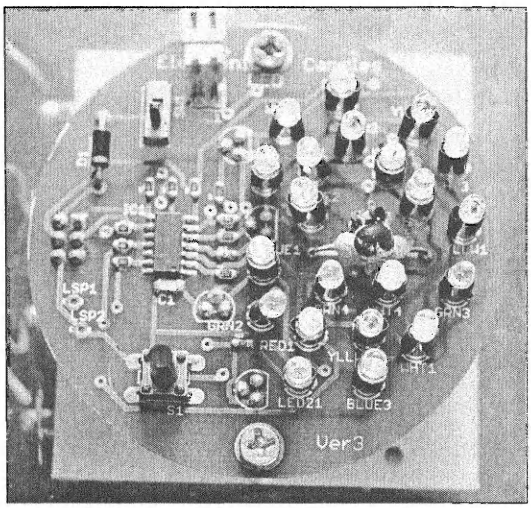


Рис. 5.41. Печатная плата устройства (сторона компонентов)

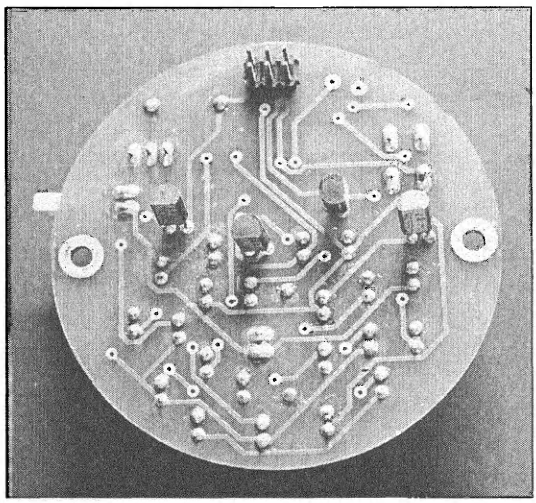


Рис. 5.42. Печатная плата устройства (сторона печатных проводников)

Программирование

Откомпилированный код проекта (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Светодиодные свечи для дня рождения работают как обычные свечи. Можно выбрать число горящих свечей (до 20 штук), а для их выключения на них нужно подуть. Как и в случае с обычными свечами, одно дуновение может погасить не все свечи, и, чтобы задуть их все, вам придется дуть несколько раз. 20 светодиодов мультиплексируется при помощи девяти линий; принцип мультиплексирования ясен из листинга 5.16. Переменные `e`, `e1`, `e2`, `e3` и `e4` управляют программным мультиплексированием. Функция `setrandom()` задает значения для массива `pwm`, обращение к которому производится из процедуры обработки прерывания по переполнению TIMER0. Массив `pwm` предназначен для задания случайных рабочих токов светодиодов (чтобы они мерцали подобно настоящим свечам).

Листинг 5.16

```
void setrandom(void)
{
 if(ab==0)
 {
  pwm[0]=3;pwm[1]=0;pwm[2]=0;pwm[3]=0;
  pwm[4]=0;pwm[5]=0;pwm[6]=2;pwm[7]=3;
  pwm[8]=3;pwm[9]=1;pwm[10]=1;pwm[11]=3;
  pwm[12]=3;pwm[13]=1;pwm[14]=1;pwm[15]=0;
  pwm[16]=3;pwm[17]=2;pwm[18]=4;pwm[19]=0;
 }
 if(ab==1)
 {
  pwm[0]=1;pwm[1]=1;pwm[2]=1;pwm[3]=1;
  pwm[4]=1;pwm[5]=3;pwm[6]=1;pwm[7]=2;
  pwm[8]=2;pwm[9]=3;pwm[10]=0;pwm[11]=2;
  pwm[12]=2;pwm[13]=1;pwm[14]=3;pwm[15]=1;
  pwm[16]=2;pwm[17]=1;pwm[18]=1;pwm[19]=1;
 }
 if(ab==2)
 {
  pwm[0]=2;pwm[1]=2;pwm[2]=2;pwm[3]=3;
  pwm[4]=2;pwm[5]=2;pwm[6]=0;pwm[7]=0;
  pwm[8]=1;pwm[9]=0;pwm[10]=1;pwm[11]=3;
  pwm[12]=3;pwm[13]=1;pwm[14]=0;pwm[15]=3;
  pwm[16]=1;pwm[17]=3;pwm[18]=2;pwm[19]=2;
 }
 if(ab==3)
 {
  pwm[0]=0;pwm[1]=3;pwm[2]=3;pwm[3]=2;
  pwm[4]=3;pwm[5]=1;pwm[6]=3;pwm[7]=1;
  pwm[8]=0;pwm[9]=2;pwm[10]=2;pwm[11]=0;
  pwm[12]=1;pwm[13]=1;pwm[14]=2;pwm[15]=2;
  pwm[16]=0;pwm[17]=0;pwm[18]=3;pwm[19]=3;
 }
 ab++;
 if(ab==4)
 ab=0;
}
```

Способ мультиплексирования светодиодов был описан в *главе 3*. Внутри главной функции `t` светодиоды разделены на восемь групп. Вся группа выключается одновременно, но группы выключаются случайным образом. Массив `z1` и переменная `z` служат для выключения светодиодов случайным образом (когда пользователь дует на свечи) посредством генерирования псевдослучайного числа от 1 до 8 (которое во время предыдущих попыток не встречалось). Выход с АЦП используется для управления двумя переменными: `present` и `past`. В зависимости от разницы между `present` и `past` (и от сгенерированного случайного числа) выключается соответствующая группа (ее элементы в массиве `statusonoff` выставляются в 0, что приводит к выключению соответствующих светодиодов). Процесс выключения светодиодов управляется фрагментом программы, приведенным в листинге 5.17.

Листинг 5.17

```
if((present-past)>=3)
{
 if(z==0)
 {
  statusonoff[0] = 0;
  statusonoff[5] = 0;
  statusonoff[9] = 0;
  statusonoff[12]=0;
  z1[0] = 0;
 }
 else if(z==1)
 {
  statusonoff[2] = 0;
  statusonoff[6] = 0;
  z1[1] = 1;
 }
 else if(z==2)
 {
  statusonoff[1] = 0;
  statusonoff[7] = 0;
  statusonoff[14]=0;
  z1[2] = 2;
 }
 else if(z==3)
 {
  statusonoff[11] = 0;
  z1[3] = 3;
 }
 else if(z==4)
```

```
 {
  statusonoff[15] = 0;
  statusonoff[18] = 0;
  z1[4] = 4;
 }
 else if(z==5)
 {
  statusonoff[10] = 0;
  statusonoff[13] = 0;
  z1[5] = 5;
 }
 else if(z==6)
 {
  statusonoff[3] = 0;
  statusonoff[4] = 0;
  statusonoff[8] = 0;
  z1[6] = 6;
 }
 else if(z==7)
 {
  statusonoff[17] = 0;
  statusonoff[16] = 0;
  statusonoff[19]=0;
  z1[7] = 7;
 }
}
```

Работа устройства

Для подготовки к работе нужно подключить схему к батареям (не более 5,5 В) и дать сопротивлению нагреть термистор. Через несколько минут он начнет реагировать на движение воздуха. Подуйте на схему и понаблюдайте за случайным выключением светодиодов.

Проект 27. Сигнализация для холодильника

Это простое и полезное устройство, которое предупреждает о том, что вы оставили открытой дверцу холодильника. Когда вы открываете холодильник, внутри него зажигается свет. Если вы не закрыли дверцу (или закрыли ее неплотно), то выключатель лампу освещения не отключает. Устройство состоит из небольшой (питающейся от батареи) схемы, в которой имеется светодиод (для обнаружения

света). Если все сделано правильно, то схема начнет выдавать звуковой сигнал после того, как дверца остается открытой более чем на девять секунд. Мы считаем, что девяти секунд вполне хватит, чтобы положить (или вытащить) продукты. Блок-схема устройства изображена на рис. 5.43.

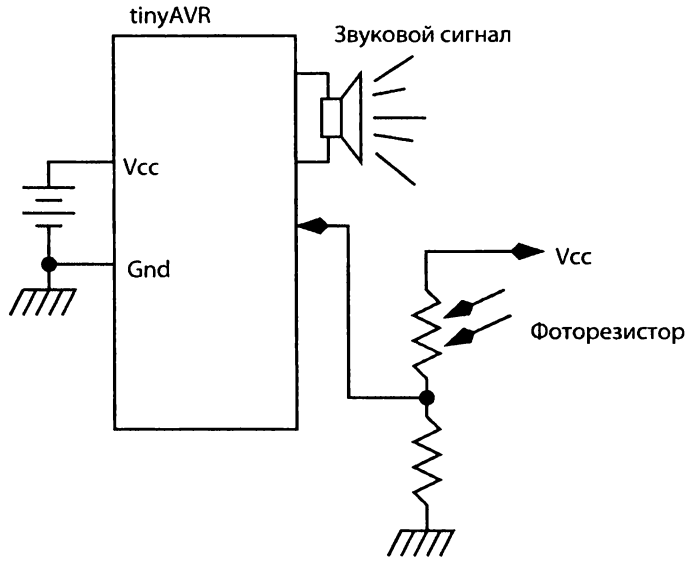


Рис. 5.43. Блок-схема сигнализатора для холодильника

Спецификация проекта

Цель проекта — создать простую компактную схему, которая обнаруживает свет от лампы освещения холодильника и отсчитывает время свечения этой лампы. Если время превышает девять секунд, устройство должно выдать звуковой сигнал, чтобы сообщить хозяину, что он, возможно, оставил открытой дверцу холодильника.

Описание устройства

На рис. 5.44 приведена принципиальная схема проекта. Схема не имеет защитного диода и стабилизатора напряжения, поэтому при подключении батарей нужно соблюдать осторожность. Чтобы схема была компактной, она питается от небольших батареек-таблеток LR44. Четыре последовательно соединенные батареи по 1,5 В питают микроконтроллер. Конденсатор C1 припаян возле контактов питания микроконтроллера (для развязывания возникающих в схеме помех). LED1 — это зеленый светодиод-индикатор открывания/закрывания дверцы. Выбран микроконтроллер ATtiny13, имеющий один аппаратный канал ШИМ (на таймере Timer0), необходимый для управления транзистором Q1. Динамик с сопротивлением 8 Ом подключен к коллектору этого же транзистора через токоограничительный рези-

стор. Фоторезистор LDR соединен с выводом PCINT через резистор сопротивлением 47 кОм. Это обеспечивает необходимый размах сигнала для прерывания микроконтроллера. Конденсатор C2 предназначен для фильтрации помех.

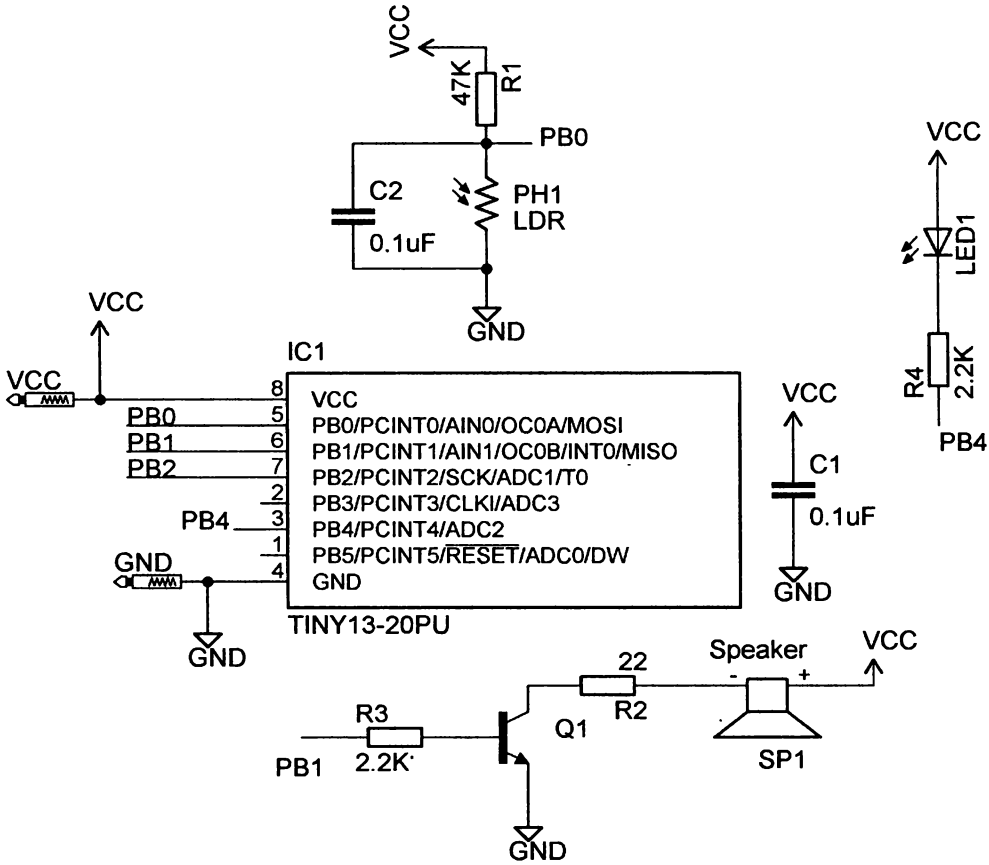


Рис. 5.44. Принципиальная схема сигнализатора для холодильника

Функционирование устройства зависит от напряжения на контакте прерывания микроконтроллера. При появлении света сопротивление фоторезистора уменьшается до нескольких килоом, поэтому падение напряжения на нем оказывается незначительным и напряжение на контакте прерывания близко к логическому нулю. Микроконтроллер выходит из режима энергосбережения. Затем микроконтроллер ждет заданное время, и, если логический уровень остается прежним, включается сигнал. Если дверь закрыта, логический уровень становится высоким, поскольку в отсутствие света сопротивление фоторезистора составляет несколько мегаом. Напряжение на фоторезисторе оказывается близким к логической единице, поэтому микроконтроллер переходит в режим энергосбережения.

Конструкция

Компоновку платы в программе EAGLE (вместе с принципиальной схемой) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Плата односторонняя (на стороне компонентов есть всего несколько перемычек). Обе стороны платы показаны на рис. 5.45 и 5.46.

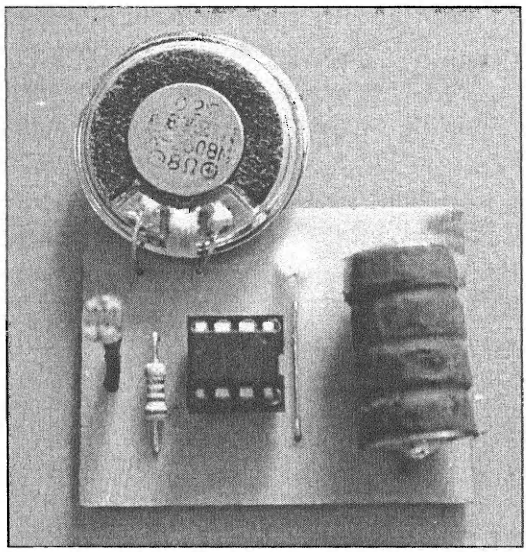


Рис. 5.45. Печатная плата сигнализатора для холодильника (сторона компонентов)

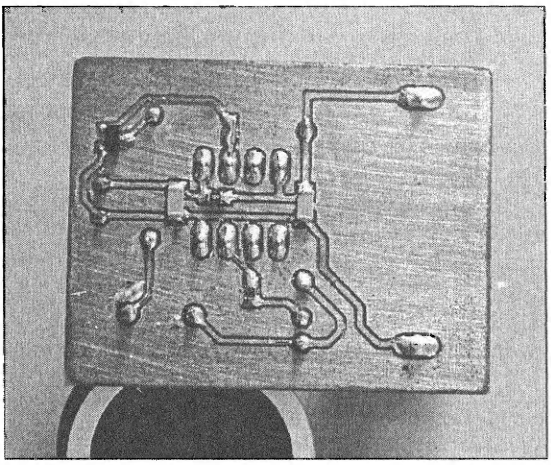


Рис. 5.46. Печатная плата сигнализатора для холодильника (сторона печатных проводников)

Фоторезистор припаян чуть повыше, чтобы на него падал свет от лампы освещения холодильника. Устройство, заключенное в корпус, изображено на рис, 5.47.

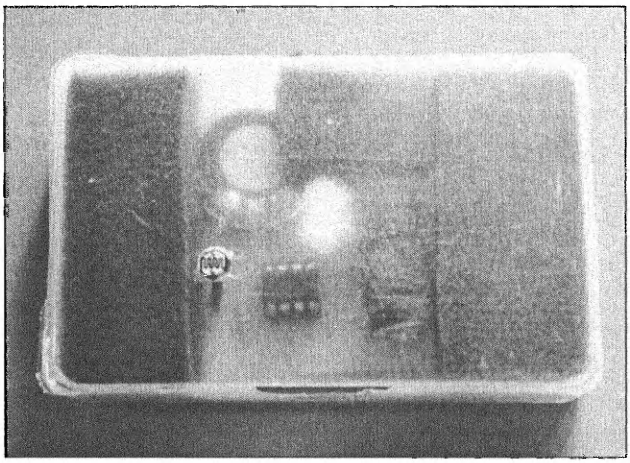


Рис. 5.47. Внешний вид сигнализатора для холодильника

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Код выполняется на тактовой частоте 8 МГц. Контроллер запрограммирован при помощи STK500 в режиме программирования ISP. Когда дверь холодильника закрыта, контроллер находится в режиме энергосбережения. В этом режиме контроллер потребляет ток всего 100 мА (что снижает потребление энергии и увеличивает время работы от батарей). Режим энергосбережения можно установить при помощи битов SE и SM1 регистра MCUCR. Контроллер выходит из режима энергосбережения после внешнего асинхронного прерывания. Это прерывание вызывает фоторезистор, подключенный к контакту PCINT. Прерывание возникает при изменении уровня напряжения. Наиболее важные фрагменты кода иллюстрируют листинги 5.18 и 5.19.

Листинг 5.18

```
while(1)
{
if(d==1)//дверь открыта после 9 секунд
{
d=0;
speaker_init();//speaker initiate
TIMSK0 &=~(1<<TOIE0);
  //прерывание таймера отключено
}
if(a==1)
{
OCR0B=0x01;//частота 1
PORTB&=~(1<<PB4);//светодиод включен
_delay_ms(200);
OCR0B=0x80;//частота 2
PORTB|=(1<<PB4);//светодиод выключен
_delay_ms(200);
}
}
```

Листинг 5.18 — это главный бесконечный цикл программы. Он состоит из двух операторов `if`, осуществляющих повторение двух фрагментов кода, которые управляют звуковым сигналом (если дверь открыта более девяти секунд). Первый фрагмент выполняется, когда процедура обслуживания прерывания по переполнению таймера делает управляющую переменную `d` равной 1. Затем в этом блоке `if`

главного бесконечного цикла включается звуковой сигнал и выключается прерывание по переполнению таймера. Второй фрагмент предназначен для воспроизведения различных частот через громкоговоритель и мигания светодиодом.

Листинг 5.19

```
ISR(PCINT0_vect)//процедура pc_int
{
pcint_init();//активация прерывания
sei();//установка бита активации в 1
if(!(PINB&(1)))//если PINB равен 0, то a=1
{
a=0;//начальное условие
TCNT0=0X00;//инициализировать таймер
timer_init();
sei();//активировать прерывание
DDRB|=(1<<PB4);//включить светодиод
PORTB&=~(1<<PB4);
}
else if((PINB&(1))==1)
{
a=0;//начальное условие
c=0;
d=0;
all_off();//отключить весь ввод/вывод
powerdown;//перейти в состояние выключения
sleep_cpu();
}
}
```

Листинг 5.19 — это процедура обработки прерывания по изменению состояния контакта, которая вызывается при каждом закрывании/открывании двери. Когда значение на этом контакте равно 0, это означает, что дверь открыта (поскольку падение напряжения на фоторезисторе близко к уровню логического 0). Это прерывание выводит микроконтроллер из режима выключения. Далее инициализируется таймер и включается светодиод. Когда дверь закрывается, на контакте оказывается уровень логической 1, что приводит к повторному выполнению процедуры обработки прерывания. Во время ее выполнения управляющие переменные сбрасываются в свои начальные значения и микроконтроллер переходит в режим выключения.

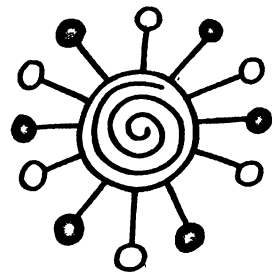
Кроме того, в программе имеются процедуры инициализации таймера и громкоговорителя.

Работа устройства

Фоторезистор должен находиться вблизи источника внутреннего освещения холодильника. Когда дверь закрыта, микроконтроллер находится в состоянии выключения и потребляет очень небольшой ток. Когда дверь открывается, включается светодиод, и если дверь остается открытой дольше указанного времени (девять секунд), то включается звуковой сигнал.

Заключение

В этой главе мы рассмотрели применение нескольких типов датчиков. Датчики — это важная часть любого проекта. Они обеспечивают взаимодействие между реальным миром и цифровым миром микроконтроллеров. При работе следует соблюдать все требования, имеющиеся в спецификациях конкретных датчиков. Теперь же настало время для музыки, поскольку мы переходим к звуковым проектам следующей главы.



Глава 6

Звуковые проекты

В этой главе мы рассмотрим несколько устройств, которые либо генерируют звук, либо взаимодействуют с пользователем при помощи звука. Звуковые сигналы нужны во многих случаях (например, зуммер может известить пользователя о каком-то событии), но в этой главе мы рассмотрим такие проекты, где звук играет главную роль.

Во многих проектах очень желательно иметь звуковые сигналы, которые могут означать успешное или неудачное завершение какой-то операции. Успех или неудача могут обозначаться соответственно коротким или длинным звуковым сигналом (либо сигналами разной частоты). На рис. 6.1 показано, как дополнить систему источником звуковых сигналов.

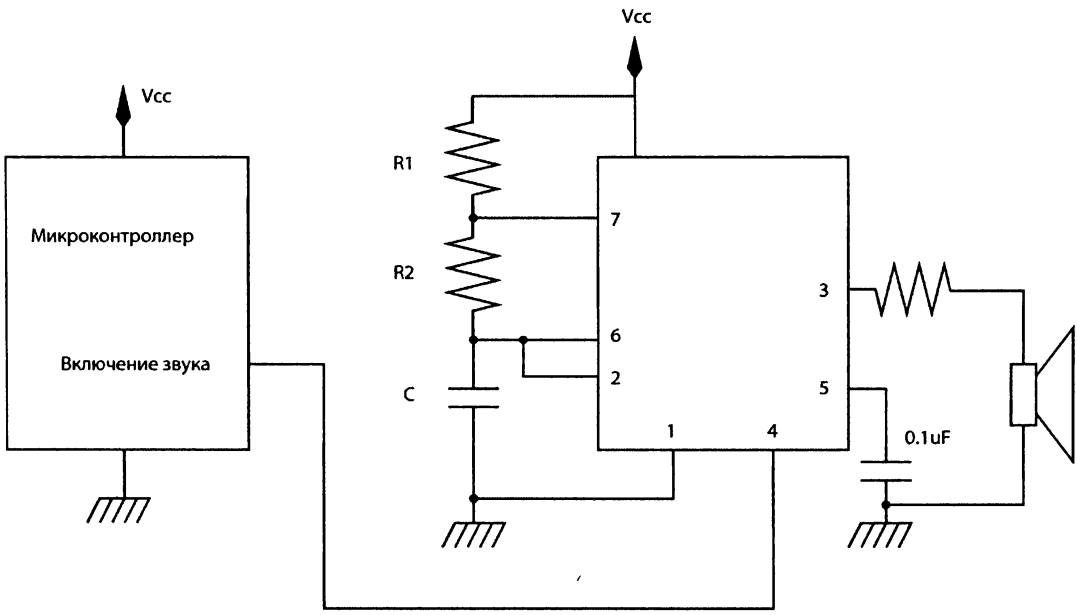


Рис. 6.1. Пример схемы звукового генератора

Звуковой генератор собран на базе микросхемы 555 Timer. Генератор работает, когда на контакт включения подано напряжение Vcc, если этот контакт заземлен, то устройство выключено. На рис. 6.2 показана еще одна схема — с управлением частотой звукового сигнала.

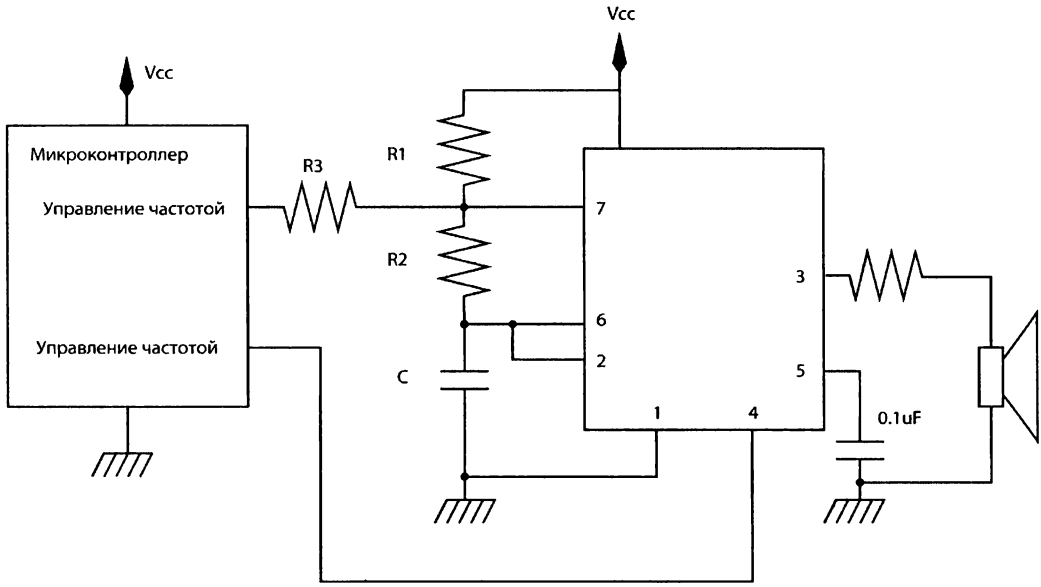


Рис. 6.2. Звуковой генератор с регулировкой частоты

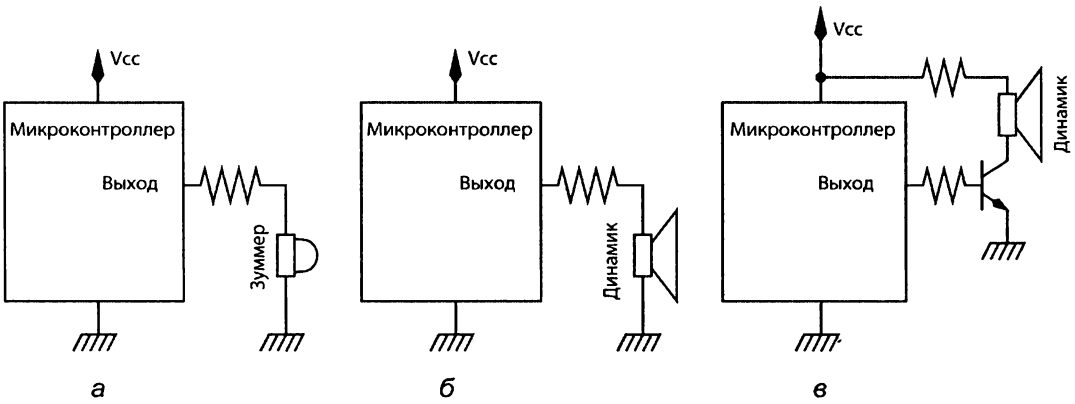


Рис. 6.3. Подключение к микроконтроллеру зуммера (*а*) и динамика (*б*, *в*)

Однако при наличии микроконтроллера дополнительная микросхема (555 Timer) не нужна. Микроконтроллер вполне способен сам генерировать звуковые сигналы. Необходим только подходящий излучатель звука. На рис. 6.3 показаны три схемы, генерирующие звук. Существуют зуммеры, которые нужно просто включить/выключить (рис. 6.3, *а*). Источником звука может быть небольшой дина-

мик, подключенный непосредственно к выводу микроконтроллера (рис. 6.3, *б*). Такие устройства широко распространены (они есть в мобильных телефонах). Для управления динамиком микроконтроллер должен сгенерировать меандр необходимой звуковой частоты (поскольку подобный сигнал получить легче всего). Меандр можно сформировать либо программным путем, либо при помощи внутреннего таймера. Кроме того, микроконтроллер позволяет изменять звуковую частоту и выдавать звуки разной продолжительности.

Микроконтроллер не может подать на динамик очень большой ток, поэтому последовательно с динамиком необходимо подключить резистор. Сопротивление резистора должно быть подобрано таким, чтобы ток через динамик не превосходил максимально допустимый ток через контакт микроконтроллера (35–40 мА). Последовательно включенный резистор будет ограничивать ток через динамик, но из-за этого звук будет негромким. Если вам нужен более громкий звук, подойдет вариант, показанный на рис. 6.3, *в*. Здесь динамик подключен к *n-p-n*-транзистору с последовательно включенным низкоомным сопротивлением (либо вообще без него).

На рис. 6.4 приведена схема, в которой для увеличения громкости использован звуковой усилитель. Так делают очень часто. Готовый усилитель TDA2020 обеспечит действительно весьма громкий сигнал, пригодный, например, для будильника. Но, поскольку это усилитель класса B, его КПД относительно мал. Усилитель, изображенный на рис. 6.4, собран по схеме H-моста и имеет КПД более 90% (класс D). Однако для его подключения придется задействовать два контакта микроконтроллера.

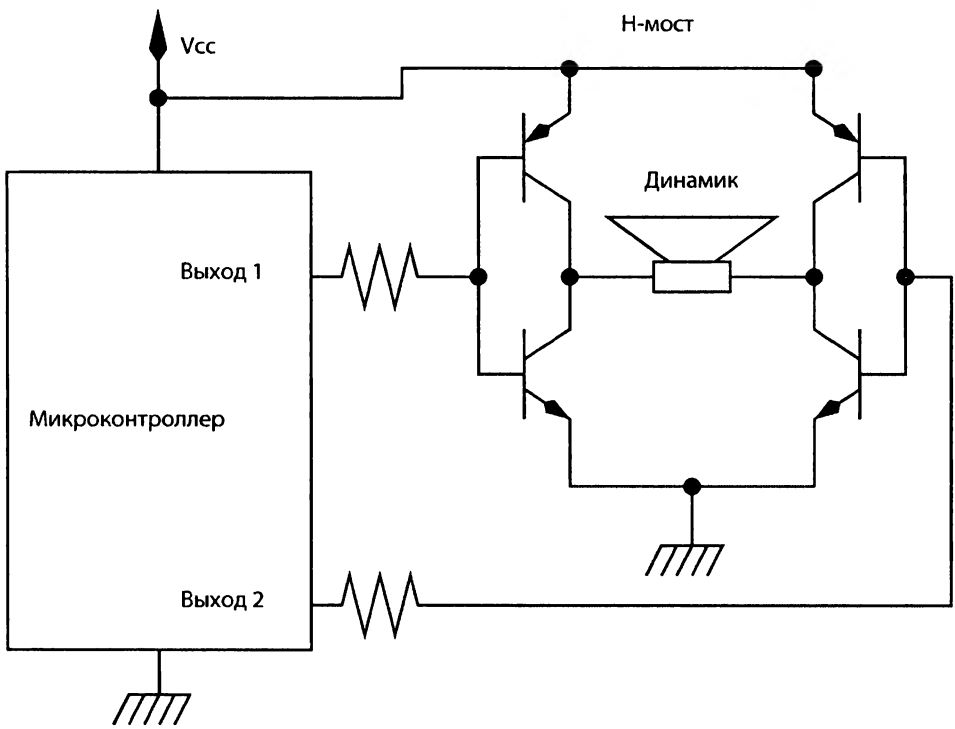


Рис. 6.4. Подключение динамика через звуковой усилитель

В проекте сигнализации для холодильника (проект 27 в *главе 5*) мы уже подключали динамик к микроконтроллеру через *n-p-n*-транзистор. В этой главе мы рассматриваем различные проекты, в которых применяются как готовые звуковые усилители, так и усилители типа Н-мост.

Проект 28. Тональный генератор

Этот проект демонстрирует способ получения звукового сигнала нужной частоты. Звуки генерируются при помощи внутреннего таймера микроконтроллера. Задается также продолжительность воспроизведения звукового сигнала. Когда микроконтроллер воспроизводит последовательность таких звуков, звучит мелодия.

Спецификация проекта

Цель проекта — создать схему на основе микроконтроллера TinyAVR, которая будет играть какую-нибудь мелодию (хранящуюся в памяти программ микроконтроллера). Мелодия определяется последовательностью пар значений "звук–продолжительность". Микроконтроллер читает пары и выдает указанный звук необходимой продолжительности. Из-за простоты схемы подобное устройство не может одновременно воспроизводить несколько звуков (как это необходимо для сложного музыкального произведения). Аппаратная часть проекта — схема на основе микроконтроллера Tiny45 и небольшой динамик. К микроконтроллеру подключен усилитель типа Н-мост, поэтому звук очень громкий. Наличие фоторезистора позволяет использовать устройство как тональный генератор и как усовершенствованную сигнализацию для холодильника. Блок-схема системы приведена на рис. 6.5.

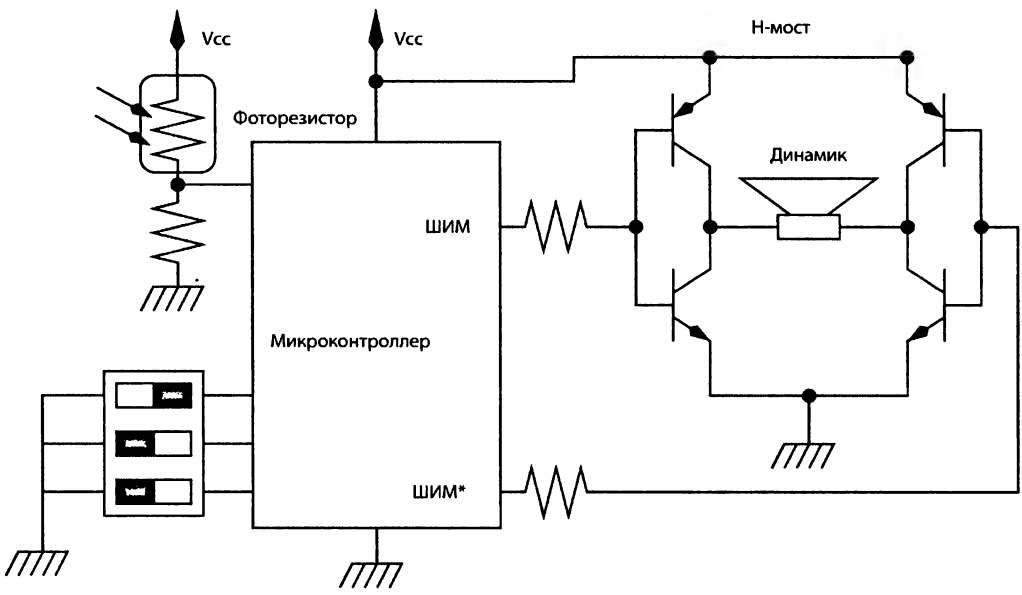


Рис. 6.5. Блок-схема тонального генератора

Описание устройства

На рис. 6.6 изображена принципиальная схема тонального генератора и усовершенствованной сигнализации для холодильника. Четыре транзистора слева — это Н-мост. На него поступают два выходных сигнала с микроконтроллера Tiny45 (PB0 и PB1), на которые подается ШИМ-сигнал. Причина выбора этих сигналов для возбуждения Н-моста состоит в том, что когда схема не должна выдавать никаких звуков — эти два выхода отключаются (устанавливаются либо в 0, либо в 1), Н-мост не потребляет тока. На выходе Н-моста установлен LC-фильтр (катушки L1 и L2 и конденсаторы C3 и C4). Небольшой динамик (8 Ом) подключен к выходу фильтров через разъем SL1.

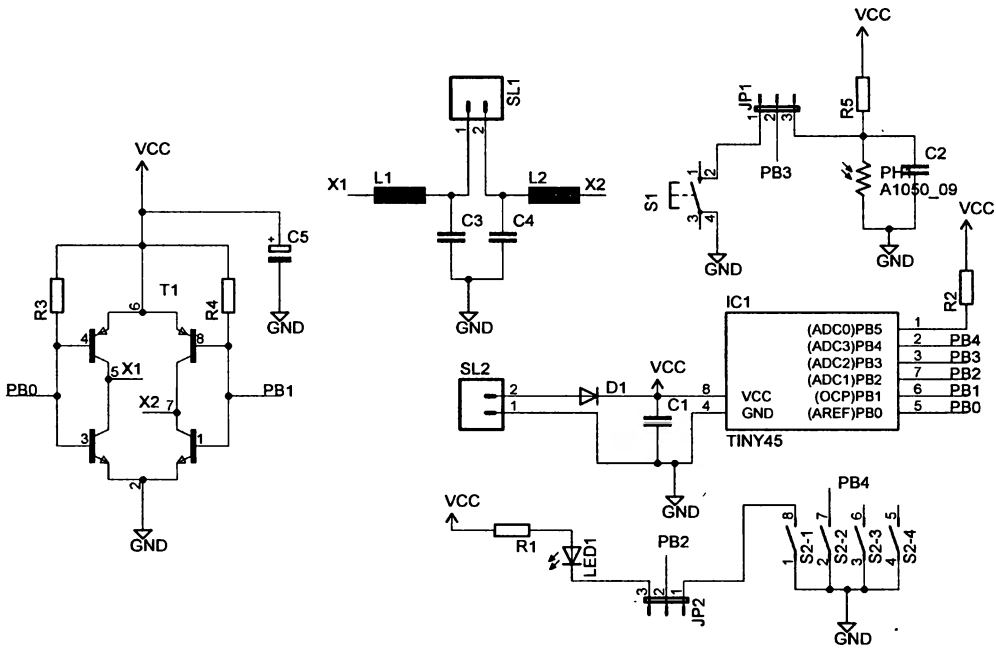


Рис. 6.6. Принципиальная схема тонального генератора (или сигнализации для холодильника)

Контакты микроконтроллера PB2 и PB3 коммутируются при помощи перемычек JP1 и JP2. Для тонального генератора кнопка S1 подключается к PB3, а DIP-переключатель S2-1 — к PB2. В проекте модифицированной сигнализации для холодильника фоторезистор (LDR) подключается к PB3, а светодиод (LED1) — к PB2.

При работе устройства как тональный генератор микроконтроллер ждет нажатия кнопки S1 и затем считывает состояние контактов PB2 и PB4. Они подключены к двум контактам DIP-переключателя, что позволяет пользователю указать на один из четырех (хранящихся в памяти программ) массивов данных. Каждый массив содержит информацию для описания звуков и их продолжительности (которые

составляют какую-либо мелодию). Микроконтроллер генерирует звук на контакте PB0 или PB1 (который подключен к H-мосту), формируя таким образом нужный звук. После того как весь массив воспроизведен, контроллер опять ждет нажатия кнопки. Чтобы выбрать другую мелодию, нужно (до нажатия кнопки S1) изменить положение DIP-переключателя.

Схема питается от внешнего источника постоянного напряжения (от 3 до 6 В). Две или три щелочные батарейки будут вполне подходящим источником питания. Диод D1 защищает схему от повреждения при неправильной полярности подключения источника питания.

Конструкция

Компоновку платы в программе EAGLE (а также ее принципиальную схему) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Печатная плата односторонняя (на стороне компонентов есть всего несколько перемычек). Обе стороны платы показаны на рис. 6.7 и 6.8.

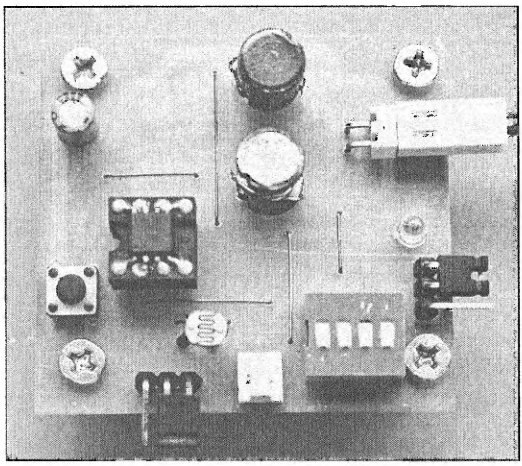


Рис. 6.7. Печатная плата тонального генератора (сторона компонентов)

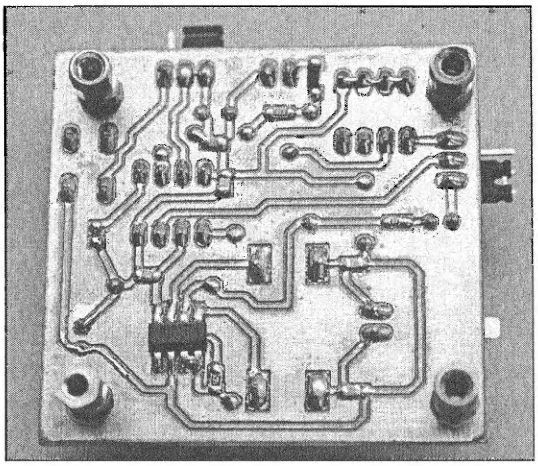


Рис. 6.8. Печатная плата тонального генератора (сторона печатных проводников)

Программирование

Самые важные фрагменты кода приведены в листингах 6.1–6.4. Звук определяется тоном и продолжительностью. Тон — это определенная частота, которую нужно подавать на динамик в течение определенного промежутка времени. Звуки в нашей программе записаны при помощи структуры, состоящей из элементов `duration` и `frequency` (тип данных — целое без знака).

Листинг 6.1

```
struct _Note
{
unsigned int durationMS;
unsigned int frequency;
};
```

Мелодии записаны во Flash-памяти микроконтроллера. Информация о мелодии хранится в макросе PROGMEM (листинг 6.2).

Листинг 6.2

```
struct _Note song1[] PROGMEM = {
  //День рождения
{ 360, 784},{ 120, 784},{ 480, 440},
{ 480, 784},{ 480, 1048},{ 960, 494},
{ 360, 784},{ 120, 784},{ 480, 440},
{ 480, 784},{ 480, 1176},{ 960, 1048},
{ 360, 784},{ 120, 784},{ 480, 1568},
{ 480, 1320},{ 480, 1048},{ 480, 494},
{ 480, 440},{ 240,1396},{ 480, 0},
{ 120, 1396},{ 480, 1320},{ 480, 1048},
{ 480, 1176},{ 960,1048 },{0,0}
};
```

Функция play_tone_P() принимает в качестве аргумента указатель *p, который перебирает массив мелодии _Note (выдавая тем самым значения продолжительности и частоты). Теперь нам нужен специальный макрос pgm_read_word для чтения Flash-памяти (листинг 6.3). Длина слова микропроцессора AVR равна двум байтам, как и у целых чисел.

Листинг 6.3

```
duration = pgm_read_word(p);
p++;
note = pgm_read_word(p);
p++;
top = (int)(31250/note);
```

После получения частоты и длительности звука пришло время создать меандр, соответствующий этим параметрам. Это делается при помощи двух таймеров. Timer0 функционирует в нормальном режиме определения длительности, а Timer1

работает в режиме ШИМ для формирования меандра нужной частоты (путем установки значения TOP регистра OUTPUT COMPARE REGISTER). Регистр OCR1C хранит верхнее значение, определяющее частоту. Если значение частоты равно нулю, то это пауза. Продолжительность паузы определяется соответствующей переменной. OCR1A — это регистр для установки среднего значения, которое постоянно равно 0,5.

Листинг 6.4

```
DDRB |= ((1<<PB0)|(1<<PB1));
   TCCR0B |= ((1<<CS02)|(1<<CS00)); //Предварительный делитель 1024
   TCCR0B &= ~(1<<WGM02); //Режим Normal
   TCCR0A &= ~((1<<WGM00) | (1<<WGM01));
//Воспроизведение звука
   if(note)
   {
TCCR1 |= ((1<<PWM1A) | (1<<COM1A0) | (1<<CS13) | (1<<CS10);
//предварительный делитель 256, режим ШИМ включен
   OCR1C = top; //Верхнее значение
   OCR1A = (OCR1C>>1); //Среднее значение
   TCNT0 = 0;
   for(;;)
   {
    if(!(PINB&(1<<PB4)))
    {
     flag1 = 1;
     return;
    }
    if(TCNT0 >= 78)
    {
     duration = duration - 10;
     TCNT0 = 0;
    }
    if(duration <= 0) break;
}
TCCR0B = 0x00;
}
else
{
   TCNT0 = 0;
for(;;)
{
   if(!(PINB&(1<<PB4)))
         {
```

```
            flag1 = 1;
            return;
           }
    if(TCNT0 >= 78)
           {
            duration = duration - 10;
            TCNT0 = 0;
           }
    if(duration <= 0) break;
}
TCCR0B = 0x00;
}
```

Работа устройства

Для пользования схемой необходимо подать внешнее напряжение питания. Убедитесь в том, что перемычки JP1 и JP2 выставлены правильно. Установив переключатели DIP, выберите нужную вам мелодию, а затем нажмите кнопку S1. Схема начнет воспроизводить песню. По окончании воспроизведения схема опять будет ждать нажатия S1. Устройство можно совместить с проектом светодиодных свечей (подключив контакт PA1 схемы со свечами к контакту 1 перемычки JP1 и соединив шину заземления обеих схем). Измененная программа проекта светодиодных свечей (ее код есть на нашем Web-сайте) запустит воспроизведение выбранной вами песни после задувания всех свечей. Схему соединения двух устройств вы можете увидеть по ссылке: **www.avrgenius.com/tinyavr1**.

Проект 29. Еще один проект сигнализации для холодильника

Это устройство — модификация предыдущего проекта сигнализации для холодильника (см. проект 27 в *главе 5*). Основная идея остается все той же, но цепь генерирования сигнала изменена. Раньше для работы динамика использовался один транзистор. Теперь мы подключили динамик к H-мосту. Результат — более громкий звук при том же источнике питания, но на другом микроконтроллере (ATtiny45).

Спецификация проекта

Цель — все та же, что и в предыдущем проекте, просто мы хотим получить более громкий сигнал.

Описание устройства

Принципиальная схема устройства была приведена на рис. 6.6. Стабилизатора напряжения нет, поэтому при подключении батарей следует соблюдать осторожность. Микроконтроллер питается от четырех батареек размера AAA напряжением

по 1,5 В. Диод D1 защищает схему от случайной подачи напряжения обратной полярности. Емкость C1 припаяна возле контактов питания микроконтроллера (для фильтрации помех). Схема (как мы уже упоминали) была сделана под два проекта, поэтому для этой цели есть две перемычки. Перемычки JP1 и JP2 предназначены для подключения фоторезистора и светодиода к микроконтроллеру (замыкаются контакты 2 и 3). LED1 — это трехмиллиметровый светодиод для индикации состояния двери холодильника (открыта/закрыта). Микроконтроллер — ATtiny45. Замена микроконтроллера требуется потому, что в Tiny45 есть два дополнительных выхода ШИМ, которые необходимы для управления звуковым усилителем в виде H-моста (в ATtiny13 их нет). Эти два аппаратных канала ШИМ находятся в таймере Timer1. К выходу H-моста подключен динамик. Для отфильтровывания высокочастотного ШИМ-сигнала (несущей) параллельно динамику включен фильтр низких частот из двух катушек по 11 мГн и конденсаторов по 0,1 мкФ. Фоторезистор LDR подключен к контакту PCINT микроконтроллера через резистор сопротивлением 47 кОм, что обеспечивает необходимый размах сигнала для прерывания микроконтроллера. Конденсатор C2 предназначен для фильтрации помех, возникающих при генерировании звукового сигнала.

Программа запускается в зависимости от уровня напряжения на контакте прерывания микроконтроллера. При наличии света сопротивление фоторезистора уменьшается до нескольких килоом, падение напряжения на нем незначительное и соответствует уровню логического 0. Это выводит микроконтроллер из спящего режима (поскольку возникает прерывание по изменению состояния контакта). Затем микроконтроллер ждет указанное количество времени, и, если логический уровень не меняется, срабатывает звуковой сигнал. Если дверца холодильника закрыта, то уровень логического сигнала становится высоким (поскольку в отсутствие света сопротивление фоторезистора порядка мегаом). Напряжение на фоторезисторе близко к логической 1, поэтому микроконтроллер опять переходит в выключенное состояние.

Конструкция

Компоновку платы в программе EAGLE (вместе с принципиальной схемой) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Печатная плата односторонняя (на стороне компонентов есть всего несколько перемычек). Стороны распаянной платы показаны на рис. 6.7 и 6.8. Фоторезистор припаян повыше, чтобы на него попадал свет от лампочки холодильника.

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 8 МГц. Контроллер запрограммирован при помощи STK500 в режиме программирования ISP. Программа работает так же, как и в предыдущем проекте (за исключением функционирования таймера). Микроконтроллер

ATtiny45 имеет дополнительные выходы ШИМ (на таймере TIMER1). Контакт OC1A и его дополнительный контакт используются для управления H-мостом. Частота работы TIMER1 устанавливается при помощи внутренней системы автоподстройки частоты (PLL), поэтому он работает на частоте 64 МГц. Частота на ШИМ-выходе равна 64/256 МГц (это примерно 250 кГц). Таким образом, генерируется высокочастотная несущая и сигнал модулируется при помощи ШИМ. Скважность высокочастотной несущей меняется в соответствии со звуковым сигналом, а продолжительность звука для конкретной мелодии устанавливается в соответствии с звуковым тоном. Это делается при помощи таблицы, в которой содержатся восемь значений:

```
const char sin_wav[8] PROGMEM=
   {128, 218, 255, 218, 128, 38, 1, 38};
```

Эти значения можно использовать в главном цикле `while` для кодирования синусоидальной огибающей высокочастотной несущей.

Листинг 6.5

```
if(d_alarm) //Цикл для воспроизведения звука
{
_delay_us(time);
   //задержка для воспроизведения мелодии
OCR1A = pgm_read_byte(sin_wav+sample);
   //Изменение скважности при помощи синусоидальных сигналов
sample++;
if( sample >= 7 )
sample = 0;
}
```

Фрагмент кода листинга 6.5 выполняется, когда управляющая переменная `d_alarm` равна 1 (аналогично предыдущей версии сигнализации для холодильника). Внутри оператора `if` переменная `sample` хранит смещение в таблице, а отсчет меняется путем изменения значения сравнения TIMER1 при смене содержимого регистра `OCR1A` (как уже было показано ранее). Период времени конкретной мелодии обусловлен частотой генерируемой волны. Устройство воспроизводит семь разных частот (каждую в течение примерно 330 мс). Это делается при помощи таблицы частот:

```
const int sin_freq[7] PROGMEM=
{200, 300, 400, 500, 600, 700, 800};
```

Изменение частоты и задержку мелодии рассчитывает процедура обработки прерывания таймера TIMER0, которая в предыдущем проекте вычисляла также задержку в девять секунд. Листинг 6.6 содержит код для манипулирования частотами.

Листинг 6.6

```
if((d_alarm)&&(count==10))
   //Отсчет времени воспроизведения звука
{count=0;
if(freq<7)
freq++;
else
freq=0;
freq_run=pgm_read_word((sin_freq+freq));
time= (125000/freq_run);
PORTB^=(1<<LED_PIN);
}
```

Оператор `if` — часть прерывания таймера TIMER0. Переменная `count` хранит время воспроизведения звука (330 мс). По истечении 330 мс прерывание таймера меняет воспроизводимую частоту путем смены значения переменной `freq_run`. Переменная `time`, задающая время воспроизведения мелодии, вычисляется по формуле `time=125000/freq_run`. Длительность воспроизведения получается в микросекундах, она используется в ранее показанном цикле `while` главной программы.

Работа устройства

Фоторезистор устанавливают возле лампочки холодильника. Когда дверца закрыта, микроконтроллер находится в выключенном состоянии и потребляет очень небольшой ток. Когда дверцу холодильника открывают, загорается светодиод, а если дверь остается открытой более девяти секунд, включается звуковой сигнал.

Проект 30. Проигрыватель рингтонов

RTTL — это популярный формат для записи рингтонов мобильных телефонов, созданный компанией Nokia. Рингтон записывается как текстовый файл при помощи кодов, которые указывают ноты и их продолжительность. Любой файл в формате RTTL содержит название рингтона, продолжительность (d), октаву (o), ритм (b) и сами ноты. Эта информация декодируется при воспроизведении рингтона. В данном проекте проигрыватель рингтонов реализован на микроконтроллере Tiny861. Динамик подключен к мощному звуковому усилителю TDA2020. Блок-схема проигрывателя показана на рис. 6.9.

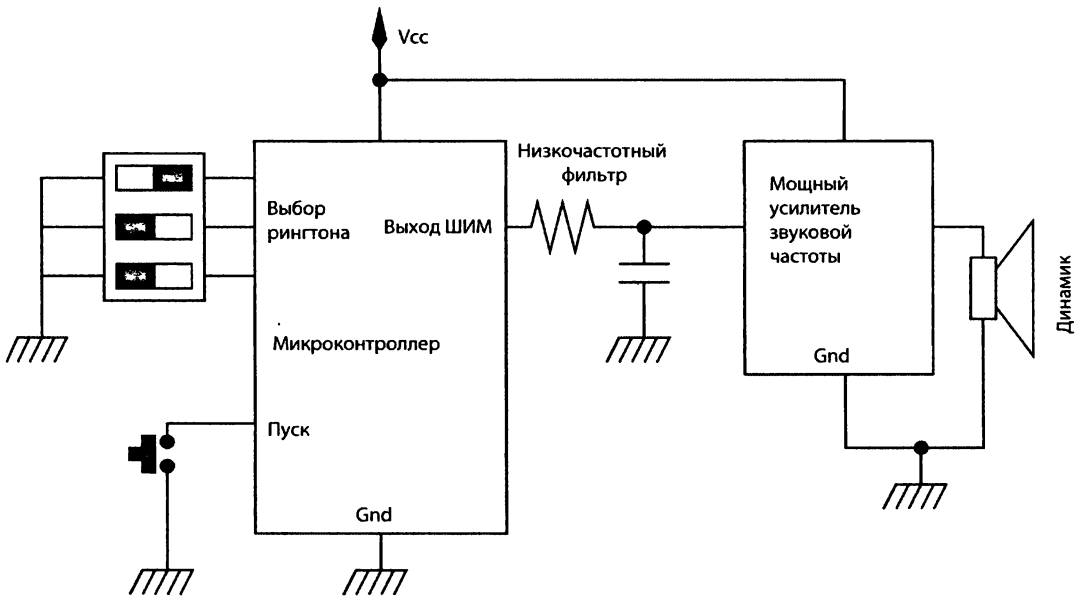


Рис. 6.9. Блок-схема проигрывателя рингтонов

Спецификация проекта

Цель проекта — создать декодер и проигрыватель RTTL на основе микроконтроллера AVR. Формат RTTL был выбран потому, что в Интернете имеется большое количество рингтонов, которые можно скачать, записать в память микроконтроллера и воспроизводить при помощи такого проигрывателя.

Описание проекта

Аппаратная часть проекта точно такая же, что и в проекте будильника для школьников (см. проект 19 из *главы 4*).

Принципиальная схема устройства изображена на рис. 6.10. В устройстве необходим DIP-переключатель, который будет выбирать мелодию из имеющихся в памяти микроконтроллера. Принципиальная схема соответствующей цепи приведена на рис. 6.11. DIP-переключатели (их изображено четыре, но используются только три) позволяют пользователю выбрать один из восьми рингтонов, хранящихся в памяти программ микроконтроллера. Для запуска воспроизведения выбранного рингтона необходимо нажать кнопку S1.

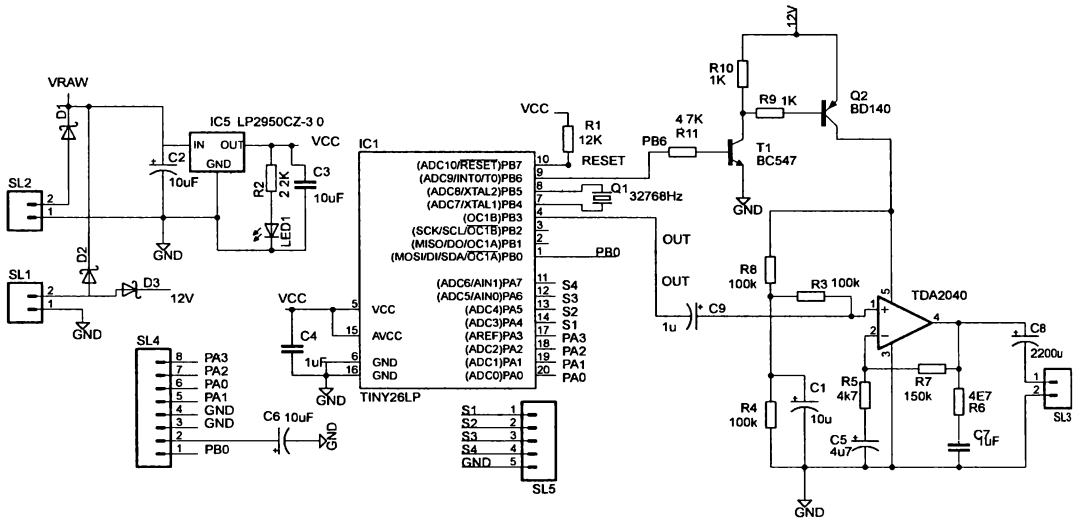


Рис. 6.10. Принципиальная схема проигрывателя рингтонов

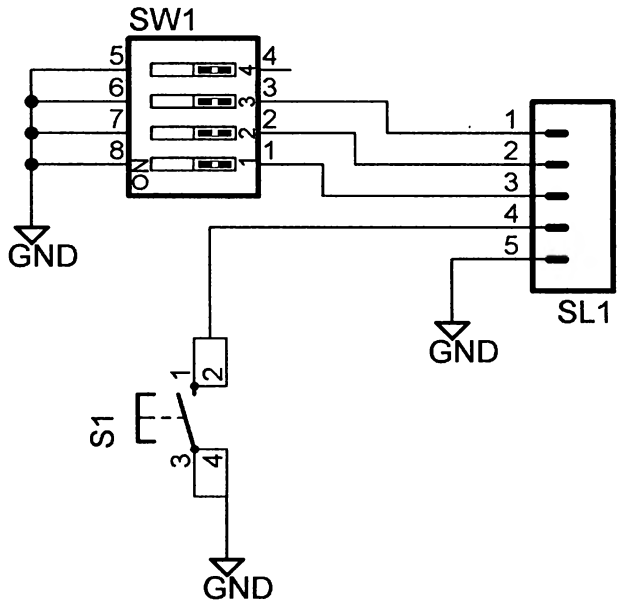


Рис. 6.11. Принципиальная схема DIP-переключателей для проигрывателя рингтонов

Конструкция

Компоновку платы в программе EAGLE (и принципиальную схему) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Печатная плата односторонняя (на стороне компонентов есть всего несколько перемычек). Обе стороны платы показаны на рис. 6.12 и 6.13.

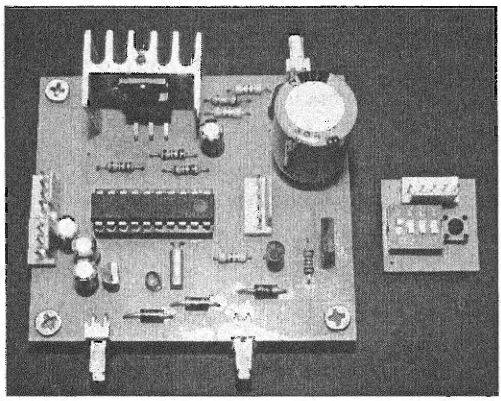


Рис. 6.12. Печатная плата проигрывателя рингтонов (сторона компонентов)

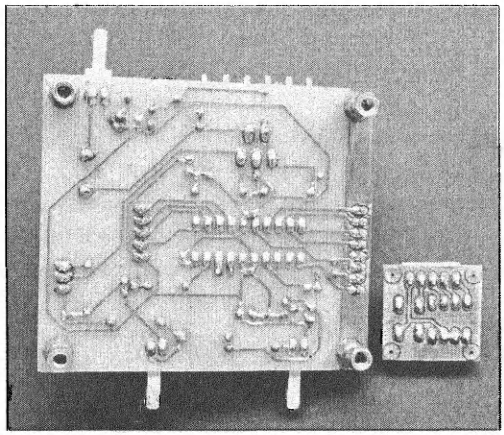


Рис. 6.13. Печатная плата проигрывателя рингтонов (сторона печатных проводников)

Программирование

Откомпилированный исходный код проекта (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 8 МГц. Контроллер запрограммирован при помощи STK500 в режиме программирования ISP. Самые важные фрагменты кода приведены в листингах 6.9–6.12. Как уже упоминалось, формат RTTL — это текстовый файл с информацией о характеристиках песни. В листинге 6.7 приведена простая мелодия в формате RTTL. В начале кода указана следующая информация: название песни, затем продолжительность по умолчанию (`d`), октава (`o`), ритм (`b`). После этого идет информация обо всех нотах.

Листинг 6.7

```
Happy Birthday Song:d=4,o=5,b=125:8g.,
  16g,a,g,c6,2b,8g.,16g,a,g,d6,2c6,
  8g.,16g,g6,e6,c6,b,a,8f6.,16f6,e6,
  c6,d6,2c6,8g.,16g,a,g,c6,2b,8g.,16g,
  a,g,d6,2c6,8g.,16g,g6,e6,c6,b,a,
  8f6.,16f6,e6,c6,d6,2c6
```

Песни сохраняются во Flash-памяти микроконтроллера при помощи макроса PROGMEM (листинг 6.8).

Листинг 6.8

```
char song1[] PROGMEM = "Happy Birthday
  Song:d=4,o=5,b=125:8g.,16g,a,g,c6,
  2b,8g.,16g,a,g,d6,2c6,8g.,16g,g6,e6,
  c6,b,a,8f6.,16f6,e6,c6,d6,2c6,8g.,
  16g,a,g,c6,2b,8g.,16g,a,g,d6,2c6,
  8g.,16g,g6,e6,c6,b,a,8f6.,16f6,e6,
  c6,d6,2c6";
```

Наша первая задача — расшифровать этот язык и получить нужную информацию о каждой ноте (т. е. ее продолжительность и скорость воспроизведения). Частоты нот хранятся в массиве top[]. В листинге 6.9 приведен код, который декодирует этот формат в соответствии со спецификациями RTTL.

Листинг 6.9

```
// формат: d=N,o=N,b=NNN:
// найти начало (пропустить название и тому подобное)
while(pgm_read_byte(p) != ':')
p++;          // пропустить ':'
p++;          // перейти к 'd'
// получить продолжительность по умолчанию
if(pgm_read_byte(p) == 'd')
{
 p++;     // пропустить "d"
 p++;     // пропустить "="
 num = 0;
 while(isdigit(pgm_read_byte(p)))
 {
  num = (num * 10) + (pgm_read_byte(p++) - '0');
 }
if(num > 0)
   default_dur = num;
p++;     // пропустить двоеточие
}
// получить октаву по умолчанию
if(pgm_read_byte(p) == 'o')
{
  p++; // пропустить "o"
```

```
  p++; // пропустить "="
  num = pgm_read_byte(p++) - '0';
  if(num >= 4 && num <=8)
  default_oct = num;
  p++; // пропустить двоеточие
}
// получить значение BPM
if(pgm_read_byte(p) == 'b')
{
  p++;   // пропустить "b="
  p++;   // пропустить "b="
  num = 0;
while(isdigit(pgm_read_byte(p)))
{
 num = (num * 10) + (pgm_read_byte(p++) - '0');
}
bpm = num;
p++;   // пропустить двоеточие
}
// BPM — это число четвертных нот в минуте
wholenote = (((60.0 * 1000.0) / (float)bpm) * 4.0);
// это время полной ноты (в миллисекундах)
// начало цикла ноты
while(pgm_read_byte(p))
{
  // сначала получить длительность ноты (если она есть)
  num = 0;
while(isdigit(pgm_read_byte(p)))
 {
  num = (num * 10) + (pgm_read_byte(p++) - '0');
 }
 if(num)
duration = wholenote / (float)num; //миллисекунды воспроизведения ноты
else
duration = wholenote / (float)default_dur;
// нам нужно будет проверить, не удлиненная ли это была нота
// теперь получим ноту
  note = 0;
  switch(pgm_read_byte(p))
  {
   case 'c':
   note = 1;
```

```
   break;
   case 'd':
   note = 3;
   break;
   case 'e':
   note = 5;
   break;
   case 'f':
   note = 6;
   break;
   case 'g':
   note = 8;
   break;
   case 'a':
   note = 10;
   break;
   case 'b':
   note = 12;
   break;
        case 'p':
        note = 0;
 }
 p++;
// теперь нужно сосчитать возможный диез '#'
if(pgm_read_byte(p) == '#')
 {
  note++;
  p++;
 }
octave = top[note];
// теперь нужно сосчитать возможную удлиненную ноту
if(pgm_read_byte(p) == '.')
{
 duration += duration/2;
 p++;
}
// теперь получим тональность
if(isdigit(pgm_read_byte(p)))
{
  scale = pgm_read_byte(p) - '0';
  p++;
}
```

```
else
{
scale = default_oct;
}
/* Обработать октаву */
switch (scale)
 {
  case 4 : /* Ничего не делаем */ // x>>y = x/2*y
  break;
  case 5 : /* %2 */
  octave = octave >> 1;
  break;
  case 6 : /* %4 */
  octave = octave >> 2;
  break;
 case 7 : /* %8 */
 octave = octave >> 4;
  break;
  case 8 : /* %16 */
  octave = octave >> 8;
  break;
 }
if(pgm_read_byte(p) == ',')
  p++; // пропустить запятую перед следующей нотой  (или мы уже в конце)
```

После того как мы получили тональность и продолжительность ноты — мы воспроизводим ее в течение указанной длительности. Это делается при помощи двух таймеров: Timer0 задает длительность ноты (в режиме переполнения), Timer1 (в режиме ШИМ) вырабатывает меандр нужной частоты при помощи установки значения `TOP` в регистре `OCR1C` (листинг 6.10).

Листинг 6.10

```
DDRB |= (1<<PB3);              //Настройка выходного контакта канала ШИМ
TCCR0A &= ~(1<<WGM00);                //Режим Normal
TCCR0B |= ((1<<CS02) | (1<<CS00));    //Предварительный делитель 1024
if(note) //Если это нота
{
TCCR1A |= ((1<<COM1B1) | (1<<PWM1B));//Неинвертирующий режим, быстрая ШИМ
TCCR1B |= ((1<<CS13) | (1<<CS10));    //Предварительный делитель 256
TCCR1C |= (1<<COM1B1);                //Очистить при совпадении
TCCR1D &=~((1<<WGM11) | (1<<WGM10));
```

```
OCR1C = octave;                          //настройка значения Top
OCR1B = (OCR1C>>1);                      //среднее значение 50%
TCNT0L = 0;
for(;;)
{
 if(TCNT0L >= 78)              //проверка длительности
 {
  duration = duration - 10.0;
  TCNT0L = 0;
 }
 if(duration <= 0.00) break;
}
TCCR0B = 0x00;
}
else                //Если случается пауза
{
TCNT0L = 0;
 for(;;)
 {
  if(TCNT0L >= 78) //Проверка длительности
 {
  duration = duration - 10.0;
  TCNT0L = 0;
 }
  if(duration <= 0.00)
  break;
 }
 TCCR0B = 0x00;
}
```

Работа устройства

В память программ микроконтроллера загружено восемь песен в формате RTTL. Выберите определенную мелодию с помощью DIP-переключателей и нажмите кнопку S1. Устройство начнет воспроизводить песню. Закончив воспроизведение, система останавливается. Для воспроизведения другой песни (или для повтора той же самой) нужно опять нажать кнопку S1.

Проект 31. Музыкальная игрушка

Этот проект — простая музыкальная игрушка, способная выдавать семь нот. В устройстве восемь кнопок: семь для нот и еще одна — для игры. Сначала игрушка выдает случайную ноту (если нажать восьмую кнопку) и загорается соответст-

вующий этой ноте светодиод. Затем вы нажимаете на кнопку, пытаясь угадать прозвучавшую ноту. Если вы угадали правильно, игра переходит на новый уровень и выдает две ноты, причем первая нота — угаданная вами, а вторая выбирается случайным образом. Вы правильно определяете эти ноты и т. д. Если вы ошиблись, то можете начать все заново. Если вы угадали, то переходите на следующий уровень. Подобная музыкальная игрушка — хороший тест для ваших музыкальных способностей. Если вы сможете запомнить и воспроизвести длительную последовательность случайных несвязанных нот, значит, вы настоящий виртуоз. Блок-схема музыкальной игрушки приведена на рис. 6.14.

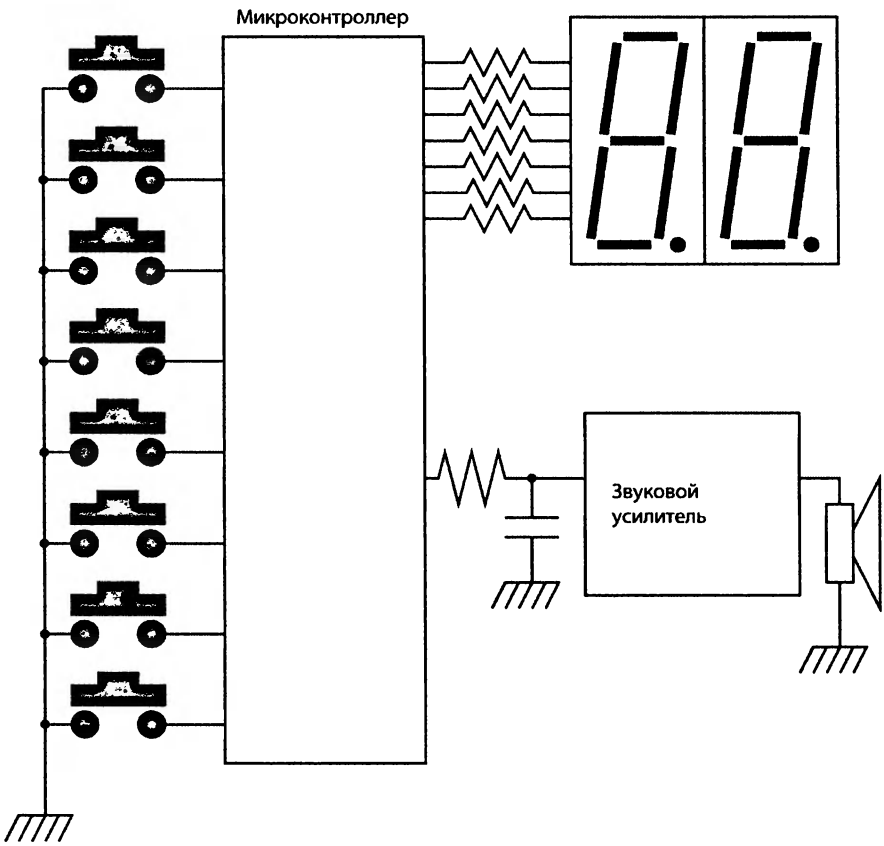


Рис. 6.14. Блок-схема музыкальной игрушки

Спецификация проекта

Цель проекта — создать музыкальную игрушку с пользовательским интерфейсом из восьми кнопок, семи светодиодов, а также семисегментного индикатора на две цифры. Устройство снабжено звуковым усилителем и динамиком для воспроизведения нот. Система питается от батареек (для портативности).

Описание устройства

Принципиальная схема устройства изображена на рис. 6.15. Питание возможно от внешних батарей или внешнего источника с напряжением от 6 до 12 В. В схеме предусмотрен стабилизатор напряжения на 5 В (LP2940). В устройстве используются: микроконтроллер Tiny861, семисегментный индикатор на две цифры (DIS1 и DIS2), а также семь светодиодов (LED2 – LED8). Индикаторы выполнены с общим анодом, коммутируются через MOSFET-транзисторы T1, T2 и T3 (NDS356A) и активизируются один за другим при помощи сигналов ROW1, ROW2 и ROW3 микроконтроллера.

Восемь кнопок собранны в матрицу 3×3. Микроконтроллер активизирует один из трех рядов кнопок при помощи контактов ROW1, ROW2 и ROW3 и проверяет нажатие кнопок путем чтения с контактов COL1, COL2 и COL3.

Схема формирует звуки при помощи ШИМ-выхода PB3 таймера Timer1, к которому подключен низкочастотный RC-фильтр (R9–C5). С выхода фильтра сигнал подается на звуковой усилитель LM386 через резистивный делитель (R10–R12). К усилителю через конденсатор C7 подключен небольшой динамик.

В схеме также предусмотрен разъем ISP (JP1) для программирования микроконтроллера.

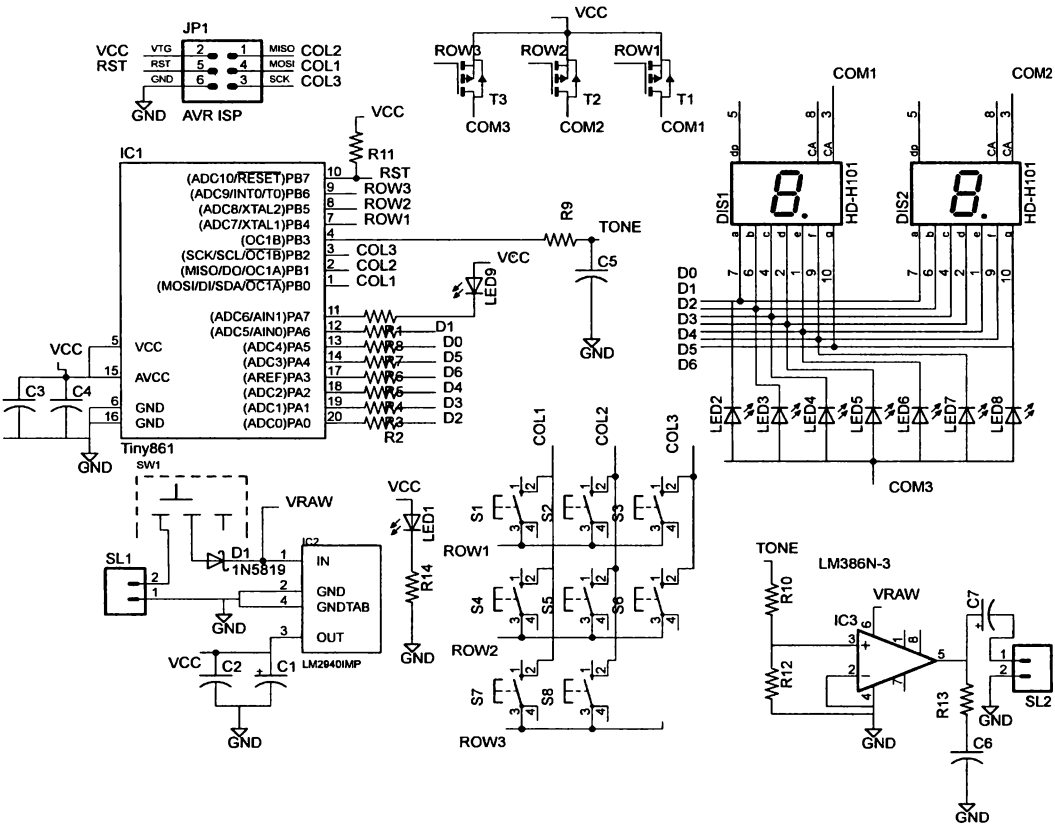


Рис. 6.15. Принципиальная схема музыкальной игрушки

Конструкция

Компоновку платы в программе EAGLE (а также принципиальную схему) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Печатная плата односторонняя (на стороне компонентов есть всего несколько перемычек). Стороны платы показаны на рис. 6.16 и 6.17.

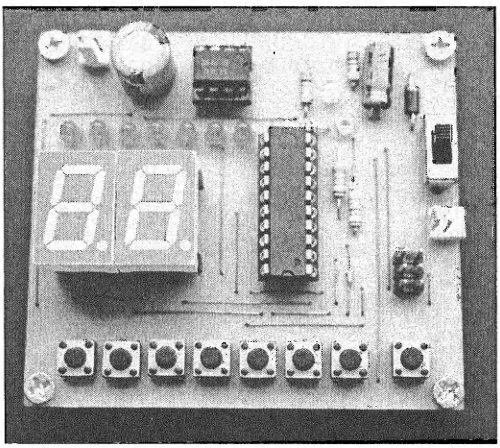


Рис. 6.16. Печатная плата музыкальной игрушки (сторона компонентов)

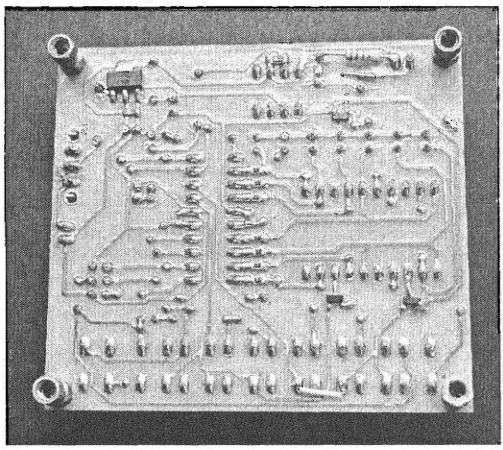


Рис. 6.17. Печатная плата музыкальной игрушки (сторона печатных проводников)

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Тактовая частота равна 8 МГц. Контроллер запрограммирован при помощи STK500 в режиме программирования ISP. Программа написана на языке C, откомпилирована при помощи компилятора AVRGCC в AVR Studio и разделена на следующие части:

- Инициализация регистров PORT и DDR.
- Инициализация дисплея.
- Прерывание по переполнению таймера.
- Мультиплексирование светодиодов и семисегментный дисплей.
- Генерирование случайных чисел.
- Генерирование звуков при помощи ШИМ.
- Сканирование кнопок.

Генерирование случайного числа — важная задача нашей программы. Нам нужно число от 0 до 6 (для этого используется 8-разрядный счетчик TCNT0L). От значения счетчика (которое может быть любым от 0 до 255) берется `mod7`, чтобы получить результат от 0 до 6. Случайное число генерируется каждый раз, когда

вызывается функция `randomize()`, а результат сохраняется в массиве `pattern[]`. Функция вызывается 50 раз (число уровней в игре). Листинг 6.11 содержит фрагмент кода.

Листинг 6.11

```
void randomize()
  // генерирование случайного числа
{
  k=k%21;
  pattern[0]=random[k];
  while(j<NUM)
  { k=TCNT0L%7;
  pattern[j]=k; // сохранение случайных чисел при каждом запуске игры
    j++;
  }
}
```

Timer1 служит для генерирования необходимых частот (см. табл. 6.1).

Таблица 6.1. *Таблица частот*

Нота	Частота, Гц	Значение OCR1C
До	240	130
Ре	270	116
Ми	300	104
Фа	320	98
Соль	360	87
Ля	400	78
Си	450	69

Меандр с уже упомянутой частотой, зависящей от значения `OCR1C`, генерируется с помощью ШИМ. Получаемую частоту можно рассчитать по формуле:

$$F = f_{clk}/N,$$

где f_{clk} — это масштабированная частота, поданная на Timer1. Отсюда можно получить N. Например, для ноты "До" с частотой 240 Гц и предварительным делением на 256 значение `OCR1C` равно:

$$\text{OCR1C} = (8\ \text{МГц}/256)/240 = 130.$$

Среднее значение фиксировано на уровне 0,75. Каждый звук генерируется в течение определенного ограниченного периода времени. Затем звук выключается путем отключения Timer1.

Код генерирования звуков иллюстрирует листинг 6.12.

Листинг 6.12

```
#ifndef tone
#define tone
#define sa 130 // 240 Гц
#define re 116 // 270 Гц
#define ga 104 // 300 Гц
#define ma 98 // 320 Гц
#define pa 87 // 360 Гц
#define dha 78 // 400 Гц
#define ni 69 // 450 Гц
#define dc 0.75 //среднее значение
void timer1init(void)
{
 TCCR1A|=(1<<PWM1B)|(1<<COM1B1);
   //активизация ШИМ
TCCR1B|=(1<<CS13)|(1<<CS10);
//предварительное деление на 256
}
void toneon(int N)
{  timer1init();
   switch(N)
   {
    case 0: OCR1C=sa;
      OCR1B=dc*OCR1C;
      break;
    case 1: OCR1C=re;
      OCR1B=dc*OCR1C;
      break;
    case 2: OCR1C=ga;
      OCR1B=dc*OCR1C;
      break;
    case 3: OCR1C=ma;
      OCR1B=dc*OCR1C;
      break;
    case 4: OCR1C=pa;
      OCR1B=dc*OCR1C;
```

```
      break;
    case 5: OCR1C=dha;
      OCR1B=dc*OCR1C;
      break;
    case 6: OCR1C=ni;
      OCR1B=dc*OCR1C;
      break;
   }
  }
  void toneoff()
  {
    TCCR1A=0x00;
  }
#endif
```

После начала игры микроконтроллер случайным образом выдает звук из таблицы. Пользователь отвечает нажатием кнопки, соответствующей звуку. Микроконтроллер проверяет, правильную ли кнопку нажал пользователь (листинг 6.13).

Листинг 6.13

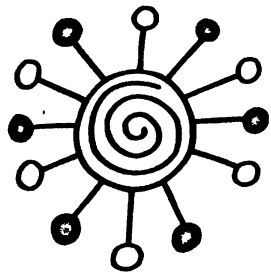
```
void checktone(int n)
  //проверяет правильность последовательности звуков
{
  toneon(n);
  if(n==pattern[count])
    // если нажата верная кнопка
  { ledinit(n,u,t);
    count++;
//подсчитывает число нажатых кнопок
 }
 else
 {
  ledoff();
  led[6]=led[13]=1;
  _delay_ms(200);
  toneoff();
 }
```

Работа устройства

Для включения игрушки необходимо подать на нее подходящее напряжение питания и нажать кнопку S8. Это запустит генерирование первой случайной ноты. Попробуйте угадать ту же самую ноту, нажав соответствующую кнопку (S1–S7), и продолжайте в том же духе.

Заключение

В этой главе мы описали несколько устройств, генерирующих звуки. Эти устройства можно использовать как отдельно, так и совместно с другими, например, для расширения функций светодиодных свечей ко дню рождения. Тональный генератор можно использовать совместно с проектом бьющегося сердца, чтобы при изменении частоты мерцания светодиодов менялся и звуковой сигнал. На основе устройств этой главы можно создавать и более интересные игрушки. В следующей главе мы рассмотрим несколько проектов, основанных на альтернативных источниках энергии.



Глава 7

Проекты с альтернативными источниками энергии

В *главе 1* мы упомянули некоторые альтернативные источники энергии, которые можно использовать для работы электронных устройств: солнечную энергию, генераторы на основе эффекта Фарадея и энергию радиоволн. В этой главе мы рассматриваем несколько проектов, основанных на альтернативных источниках энергии, базирующихся на эффекте Фарадея. Рабочее напряжение для многих компактных автономных конструкций можно получить при помощи интересного устройства, которое преобразует механическую энергию в электрическую (по закону Фарадея). Преобразователи механической энергии в электрическую (динамо-машины), работающие на эффекте Фарадея, хорошо известны. Однако устройство, изображенное на рис. 7.1, стало популярным благодаря фонарикам, которые тоже появились достаточно давно. Конструкция состоит из пластиковой трубки подходящего диаметра и длины, внутри которой находится магнит. На трубку намотана катушка с несколькими сотнями витков медного эмалированного провода, а концы трубки закрыты заглушками. Для генерирования напряжения трубку нужно просто потрясти. При перемещении магнита по трубке в катушке возникает переменное напряжение, которое можно выпрямить и отфильтровать. Величина напряжения определяется по закону Фарадея:

$$U = -N \times (d\varphi/dt),$$

где U — наведенное напряжение, N — число витков катушки, а $d\varphi/dt$ — скорость изменения магнитного потока через катушку. Магнитный поток через катушку равен: $\varphi = B \times S$, где B — это магнитное поле, S — площадь поперечного сечения катушки. Таким образом, катушка с более сильным магнитом и с большим поперечным сечением будет выдавать бóльший магнитный поток. Чтобы в катушке возникло напряжение, магнитное поле должно меняться, что и делается путем перемещения магнитов в трубке. Механическое перемещение магнита создает изменение магнитного потока через катушку, которое генерирует напряжение в ней. При быстром перемещении скорость изменения магнитного потока и напряжение будут больше. Когда магнит входит в катушку, полярность генерируемого напряжения будет противоположна той, которая возникает при выходе магнита из катушки. Следовательно, в катушке генерируется симметричное разнополярное напряжение.

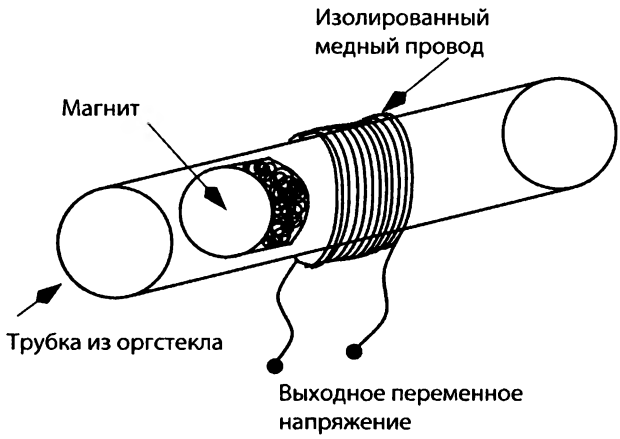


Рис. 7.1. Конструкция генератора на эффекте Фарадея

Такой генератор имеет множество преимуществ. Батарейки не потребуются, значит, их потребление уменьшится и это будет нашим вкладом в сохранение чистоты окружающей среды. К тому же вас никогда не подведет севшая в самый неподходящий момент батарейка.

Выбор подходящего стабилизатора напряжения

Единственная проблема устройства без батареек — вам придется трясти его ровно столько времени, сколько вы хотите подавать напряжение на схему. Если вы перестанете трясти трубку, то она перестанет выдавать напряжение и у вас будет только остаточное напряжение на конденсаторах. Для устройств, которые не должны функционировать постоянно, этого может оказаться вполне достаточно. В других случаях можно трясти трубку периодически и заряжать ионистор, от которого будет подаваться питание на схему. Ионистор — это конденсатор очень большой емкости. По сравнению с обычными электролитическими конденсаторами ионисторы имеют гораздо более низкое максимальное рабочее напряжение. Такой подход имеет свои недостатки: для зарядки ионистора до требуемого напряжения нужно много времени и усилий. Обычно максимальное рабочее напряжение такого конденсатора равно 2,5 или 2,7 В, хотя есть ионисторы и на 5 В. Ионистор емкостью 10 Ф напряжением 2,5 В хранит 25 кулонов заряда (при полной зарядке). Обычный электролитический конденсатор (10000 мкФ/25В) хранит только 0,25 кулона. Однако если в устройстве с ионистором потребуется рабочее напряжение 5 В, понадобится повышающий преобразователь напряжения.

На рис. 7.2 изображена блок-схема источника питания с ионистором и повышающим преобразователем MAX756. Напряжение с катушки выпрямляется при

помощи четырех диодов и подается на ионистор. Выходное напряжение MAX756 можно настроить на 3,3 или 5 В (в зависимости от состояния входного контакта 3/5*). Если этот контакт заземлен, то на выходе 5 В, если же на нем входное напряжение, то на выходе будет 3,3 В. Для схемы требуется очень мало внешних компонентов.

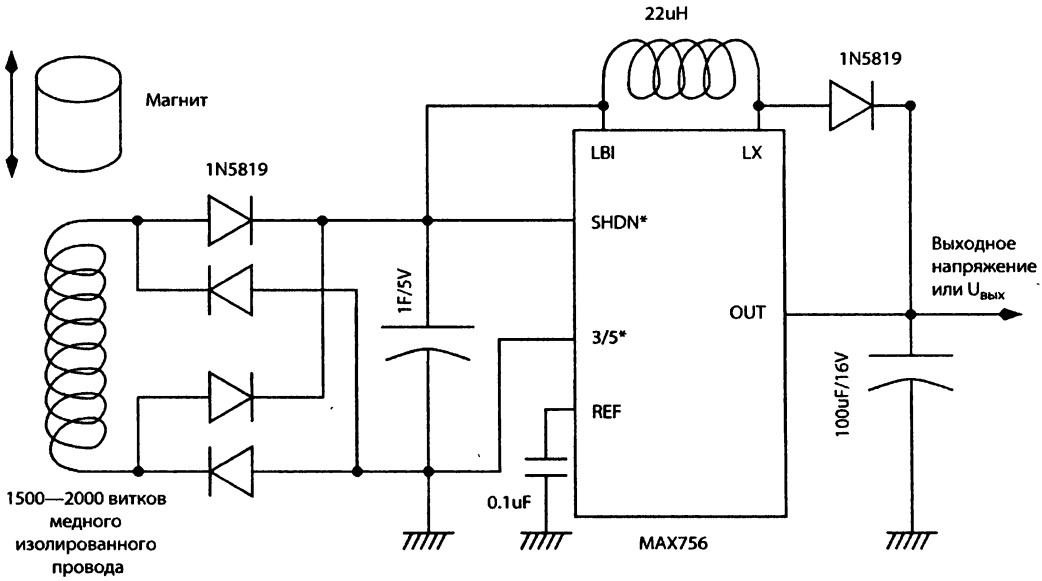


Рис. 7.2. Блок-схема источника питания с ионистором

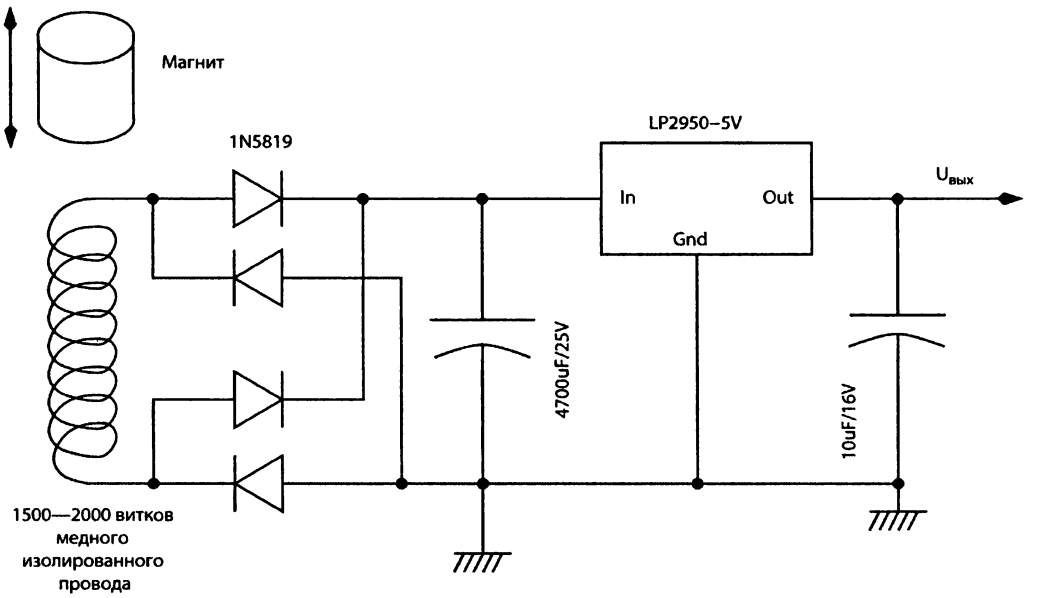


Рис. 7.3. Источник питания на базе генератора Фарадея со стабилизатором LDO

На рис. 7.3 показан источник питания с обычным электролитическим конденсатором соответствующего номинала (в данном случае 4700 мкФ/25 В) на выходе выпрямителя. Стабилизатор с низким падением напряжения (LDO) типа LD2950, который выпускается на несколько выходных напряжений (в данной схеме на 5 В).

В табл. 7.1 приведены некоторые повышающие преобразователи постоянного напряжения, которые можно использовать с генератором Фарадея, выпрямителем и ионистором. Преобразователь постоянного напряжения выбирают по току покоя и минимальному входному напряжению. Преобразователь с низким током покоя, безусловно, предпочтительнее. Однако часто выбор зависит от того, что есть под рукой или даже от того, какую микросхему легче припаять на плату.

Таблица 7.1. *Характеристики некоторых повышающих преобразователей постоянного напряжения*

Номер по порядку	Устройство	Изготовитель	Минимальное напряжение на преобразователе, В	Минимальное напряжение запуска, В	Ток покоя, мкА
1	MAX756	Maxim	0,7	1,1	60
2	TPS61070	Texas Instruments	0,9	1,1	19
3	ZXSC100	Zetex	0,92	1,01	150

Делаем генератор Фарадея

Сделать генератор Фарадея просто — для этого потребуется три основных компонента (рис. 7.4): трубка подходящего размера, цилиндрические магниты (которые легко скользят внутри трубки), а также медный изолированный провод (для намотки снаружи трубки).

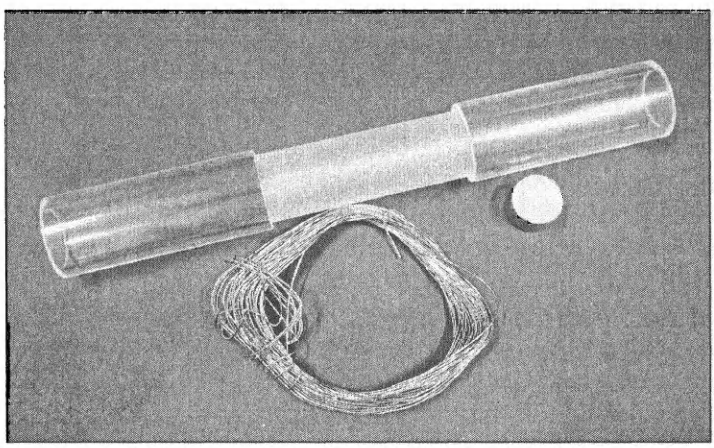


Рис. 7.4. Каркас из оргстекла, провод для намотки катушки и магнит

Мы взяли трубку с внутренним диаметром 15 мм, внешним — 20 мм и сильные неодимовые магниты диаметром 12 мм, которые могут легко скользить внутри трубки. На трубке толщиной в 2,5 мм была сделана проточка глубиной в 1,5 мм и длиной 5 см (для намотки провода). Для увеличения числа витков по обеим сторонам выемки были установлены кольца из оргстекла. В это пространство без труда помещается от 1500 до 2000 витков медного провода марки 36 SWG. Если диаметр провода будет меньше, то число витков возрастет и индуцированное напряжение повысится, однако при этом снизится максимальный ток. Здесь решение зависит от разработчика. После намотки провода на трубку внутрь нее помещают магниты и концы трубки закрывают круглыми заглушками (рис. 7.5).

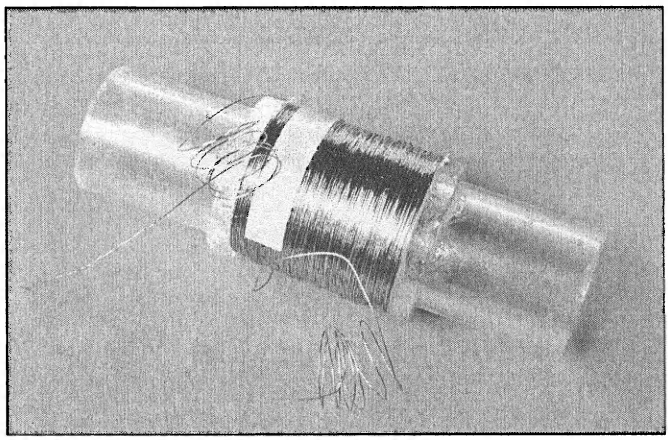


Рис. 7.5. Катушка, намотанная на каркасе из оргстекла

Экспериментальные результаты и их обсуждение

Экспериментальные результаты соответствуют уравнению Фарадея. Мы изготовили и протестировали несколько образцов генераторов. Напряжение на устройстве с 1800 витками провода марки 36 SWG было зафиксировано осциллографом (рис. 7.6). Ток короткого замыкания этой же катушки показан на рис. 7.7. Ток измерялся в цепи с низкоомным резистором (0,22 Ом), подключенным к выходу катушки; максимальное значение тока равно 180 мА. Такой генератор может зарядить ионистор до 3 В менее чем за 100 встряхиваний.

Тот же образец устройства был использован и для зарядки ионистора на 0,5 Ф (5 В), к которому в качестве нагрузки было подключено сопротивление в 1 кОм. Ионистор был заряжен до 3,75 В и затем разряжался через нагрузку. На рис. 7.8 показан график разрядки емкости через сопротивление в 1 кОм (как функция от времени).

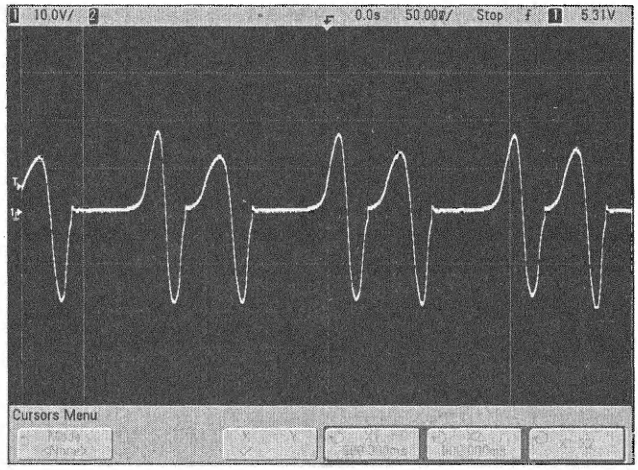


Рис. 7.6. Выходное напряжение на генераторе Фарадея

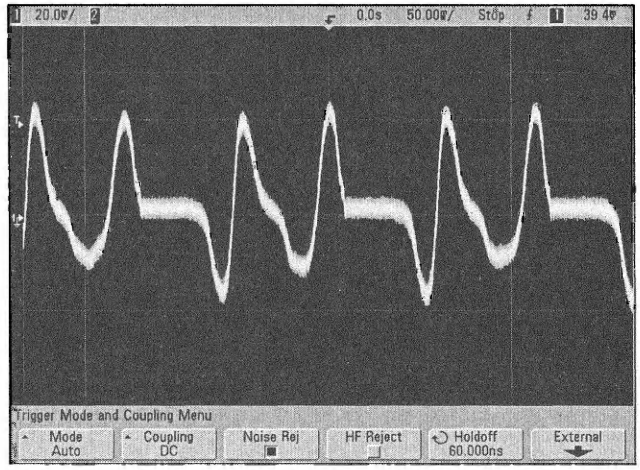


Рис. 7.7. Выходной ток на генераторе Фарадея

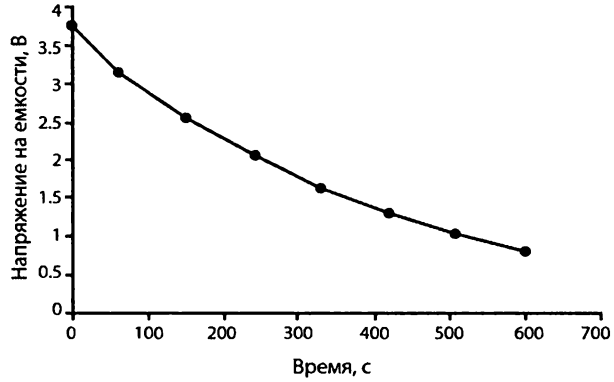


Рис. 7.8. Изменение напряжения на ионисторе при подключении активной нагрузки

Схема, приведенная на рис. 7.2, выдавала стабильный ток 1,6 мА в течение более шести минут.

Ионистор в 1 Ф был заряжен до 3 В. Выход преобразователя MAX756 был настроен на 3,3 В. Регистрировалась зависимость напряжения на емкости как функция от времени до тех пор, пока на выходе преобразователя оставалось равным 3,3 В. Результаты иллюстрирует рис. 7.9.

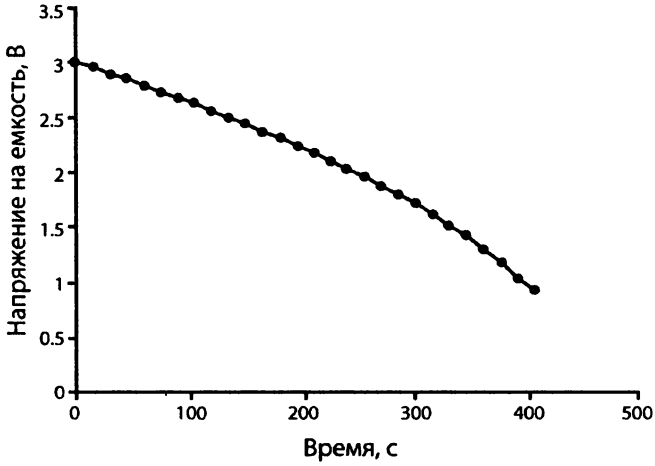


Рис. 7.9. Изменение напряжение на ионисторе при подключении преобразователя MAX756

Рис. 7.10. Зависимость квадрата напряжения на ионисторе от времени

Поскольку ионистор должен был теперь выдавать постоянную мощность, то напряжение на нем падало в соответствии со следующим уравнением:

$$U_c = \sqrt{U_{max}^2 - 2Pt/C},$$

где U_c, U_{max} — мгновенное и начальное напряжения на ионисторе, P — мощность, потребляемая нагрузкой, а C — емкость ионистора. Это означает, что если изобразить зависимость квадрата напряжения на ионисторе от времени, то вы получаете прямую линию (рис. 7.10). Следовательно, преобразователь напряжения является постоянной силовой нагрузкой для ионистора. Данное условие нужно учитывать при создании источников питания на принципе закона Фарадея с ионисторами. Уравнение показывает, как долго ионистор сможет подавать питание на нагрузку через преобразователь напряжения.

Теперь у нас есть источник питания на основе генератора Фарадея, который пригоден для питания портативного устройства. Давайте используем этот бесплатный источник питания в нескольких проектах.

Проект 32. Дистанционное инфракрасное управление без батарей

Дистанционное управление на инфракрасных лучах стало неотъемлемым элементом современного электронного оборудования. Подобный пульт управления прилагается почти к любым бытовым электронным устройствам: телевизорам, аудиопроигрывателям, кондиционерам и т. п. Большинство существующих пультов дистанционного управления питается от одной или двух батареек 1,5 В (размера AA или AAA). В таком устройстве смонтирован инфракрасный светодиод, который передает коды команд, соответствующие нажатой кнопке. Каждая кнопка имеет свой уникальный код команды. Инфракрасное излучение (невидимое для человеческого глаза, но регистрируемое фотодетектором) модулируется несущей частотой от 35 до 40 кГц, и этот сигнал используется для передачи кодов команд.

На рис. 7.11 показан сигнал SIRCS, передаваемый пультом дистанционного управления телевизором.

С инфракрасными устройствами есть одна серьезная проблема: здесь нет общих стандартов и каждый производитель создает свой собственный формат команд, частоту модуляции и т. д. Мы насчитали девять систем дистанционного управления: Daewoo, Samsung, Japan, Motorola, SIRCS (Sony), RC5 (Philips), Denon, NEC и RECS80 (Thomson). По мере расширения функций аппаратуры все более сложным становится и дистанционное управление ею. Теперь на пульте может быть более 50 кнопок. Однако наиболее интенсивно используется только часть этих кнопок. Возьмем к примеру дистанционное управление телевизором. Чаще всего на нем нажимают кнопки выключения/включения, выключения звука, переключения на следующий/предыдущий канал, повышения/понижения громкости.

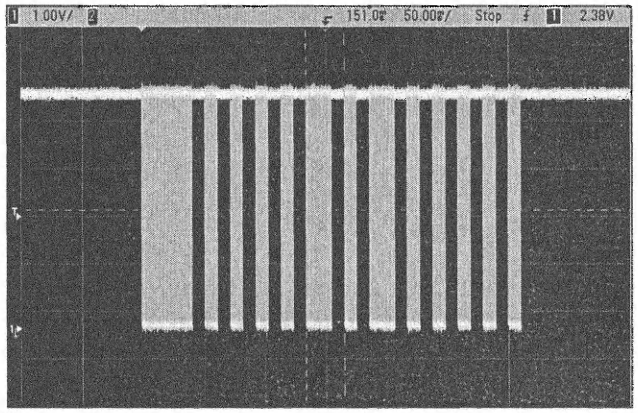


Рис. 7.11. Осциллограмма сигнала от пульта дистанционного управления телевизором

Для блоков дистанционного управления выпускаются специальные микросхемы. Если вы откроете любое устройство дистанционного управления, то вы обнаружите всего одну микросхему, к которой подключены кнопки и инфракрасный светодиод.

В этом проекте мы решили реализовать упрощенное дистанционное управление (без батареек) для телевизора, снабженное только шестью кнопками, причем пользователь может выбрать формат команд (NEC, SIRCS, RC5 или Samsung).

Спецификация проекта

Цель проекта — создать дистанционное управление с шестью кнопками (без батареек) для телевизора. На рис. 7.12 приведена блок-схема этого устройства.

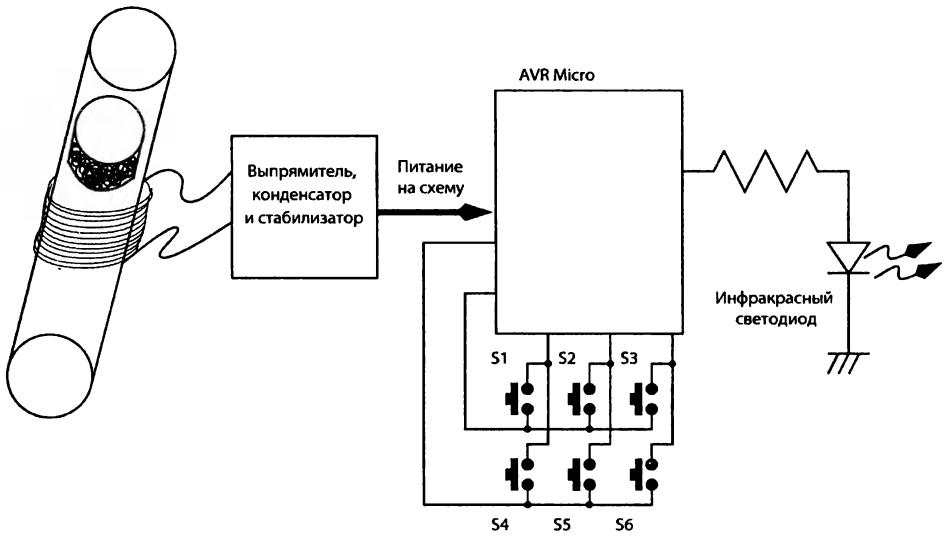


Рис. 7.12. Блок-схема пульта дистанционного управления (без батареек)

В устройстве использован восьмиконтактный микроконтроллер. Питание подается от генератора Фарадея. Шесть кнопок подключены в виде матрицы 3×2 с пятью выводами. У микроконтроллера семейства tinyAVR разработчику доступно до шести контактов ввода/вывода. В описываемом устройстве заняты все шесть контактов: пять контактов — для кнопок и еще один — для инфракрасного светодиода.

Описание устройства

Принципиальная схема нашего дистанционного управления изображена на рис. 7.13. Схема питается от генератора Фарадея, подключенного через разъем SL1. Диоды D1–D4 выпрямляют переменное напряжение; постоянное напряжение фильтруется и запасается в конденсаторах C1 и C3. Стабилизатор LP2950-3.3V выдает напряжение 3,3 В на микроконтроллер Tiny45. Для подачи команд управления предусмотрено шесть кнопок, которые подключены к микроконтроллеру через контакты PB0-PB4. Кнопка PB5 предназначена для управления инфракрасным светодиодом.

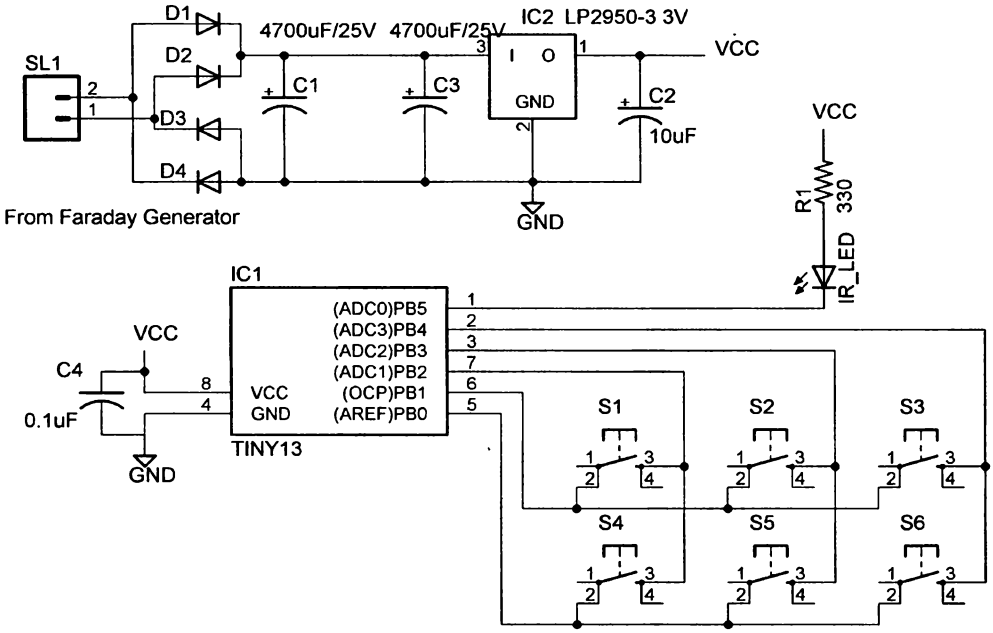


Рис. 7.13. Принципиальная схема пульта дистанционного управления (без батареек)

Микроконтроллер опрашивает кнопки и при обнаружении нажатия передает код соответствующей кнопки. Формат дистанционного управления (REC, NEC, Samsung или SIRCS) программируется в микроконтроллере. Несмотря на то, что все форматы отличаются, они все-таки имеют кое-что общее. Передача кода кнопки начинается со стартового бита, за которым следуют несколько адресных битов, а затем — несколько битов кода команды (т. е. собственно код нажатой кнопки).

Адрес обозначает устройство (телевизор, аудиопроигрыватель, проигрыватель DVD и т. д.). В некоторых форматах за стартовым битом может следовать код команды, а уже за ним — биты адреса. Кодирование и длительность битов в разных форматах также отличаются. Эти подробности приведены в файлах исходных кодов данного проекта.

Конструкция

Схема собрана на специально изготовленной печатной плате и заключена в небольшой корпус. Генератор Фарадея сделан под размер корпуса. На рис. 7.14–7.16 показана плата устройства, пульт в корпусе и со снятой крышкой.

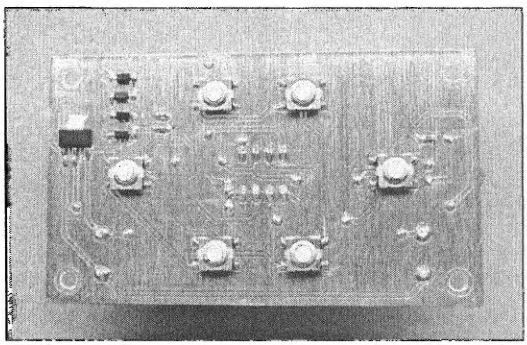


Рис. 7.14. Печатная плата пульта дистанционного управления

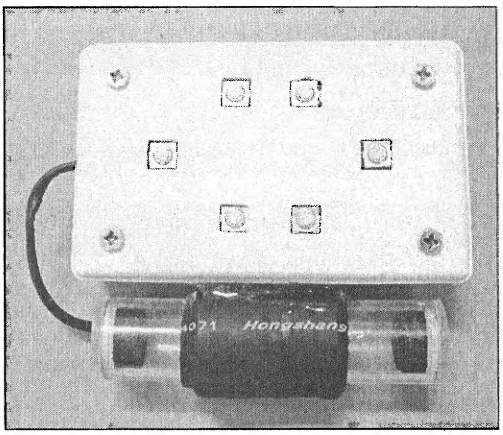


Рис. 7.15. Пульт дистанционного управления в корпусе

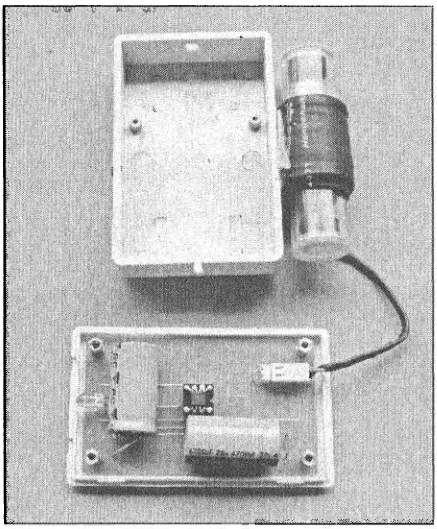


Рис. 7.16. Пульт дистанционного управления (крышка открыта)

Программирование

Откомпилированный код проекта (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

В главной программе выполняется цикл ожидания нажатия кнопки. До нажатия кнопки микроконтроллер находится в режиме ожидания (для экономии энергии). Нажатие кнопки вызывает прерывание по изменению состояния контакта, которое пробуждает микроконтроллер (листинг 7.1). Микроконтроллер сканирует кнопки и идентифицирует нажатую кнопку. Далее выполняется процедура передачи кода кнопки в соответствии с выбранным протоколом (форматом команды). После этого микроконтроллер снова переходит в состояние ожидания (до момента нажатия следующей кнопки).

Листинг 7.1

```
ISR(PCINT0_vect)
//Процедура обработки прерывания по изменению состояния контакта
{
MCUCR &= ~((1<<SE) | (1<<SM1));
 //Отключить режим ожидания
PCMSK &= ~((1<<PCINT4) | (1<<PCINT3) | (1<<PCINT2));
 //Прерывание по изменению состояния контакта отключено на всех контактах
New_Key_Pressed = 1;
}
```

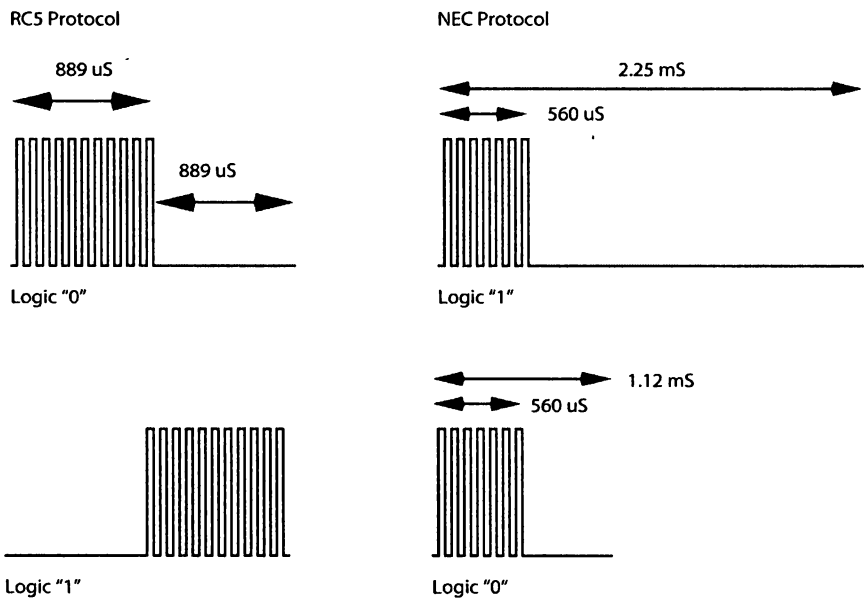


Рис. 7.17. Пример кодирования по протоколам RC5 и NEC

Код кнопки модулирует несущую частоту. Несущая частота зависит от выбранного протокола (рис. 7.17). Микроконтроллер, работающий на тактовой частоте 1 МГц, генерирует несущую частоту с помощью внутреннего 8-разрядного таймера (листинг 7.2). Несущая частота формируется переключением выходного разряда. Поэтому для получения несущей с частотой 36 кГц, частота прерываний должна быть равна 72 кГц.

Листинг 7.2

```
ISR(TIMER0_COMPA_vect)
  //Обработчик прерывания по совпадению при сравнении
{
PORTB ^= (1<<IR_LED);
  //Переключение бита для генерирования ШИМ
}
```

Инициализация таймера для протокола RC5 приведена в листинге 7.3.

Листинг 7.3

```
{
TCCR0A |= (1<<WGM01);
  //Включить прерывание по совпадению при сравнении
TCCR0B |= (1<<CS00);
  //Тактовая частота 8 МГц(предварительный делитель = 1), режим CTC
OCR0A = 14;
  //Прерывание с частотой примерно 72 кГц
TIMSK |= (1<<OCIE0A);
  //Активизировать прерывание CTC
sei();
}
```

Для передачи любого кода кнопки необходимо выключать и включать инфракрасный светодиод с частотой несущей, модулированной битами кода.

Листинг 7.4 иллюстрирует формирование реальных битов по протоколу RC5.

Листинг 7.4

```
void transmit_RC5(void)
{
 while(Tx == 1)
 {
  if(Tx_bit_RC5[i] == 0)
  {
```

```
    DDRB |= (1<<IR_LED);
      //Включить несущую
    _delay_us(RC5_ON_PERIOD_ZERO);
    DDRB &= ~(1<<IR_LED);
      //Выключить несущую
    _delay_us(RC5_OFF_PERIOD_ZERO);
}
if(Tx_bit_RC5[i] == 1)
 {
  DDRB &= ~(1<<IR_LED);
// Выключить несущую
  _delay_us(RC5_OFF_PERIOD_ONE);
  DDRB |= (1<<IR_LED);
    // Включить несущую
  _delay_us(RC5_ON_PERIOD_ONE);
  }
  i++;
  if(i == 14)
  {
   i=0;
   Tx = 0;
  }
 }
 PCMSK |= ((1<<PCINT4) | (1<<PCINT3) | (1<<PCINT2));
}
```

Работа устройства

Пользоваться пультом дистанционного управления очень просто. Потрясите его несколько раз и нажмите нужную кнопку!

Проект 33. Электронные игральные кости (без батареек)

Вместо обычных игральных костей очень интересно пользоваться электронными. Ранее мы уже рассматривали подобное устройство (см. проект 12 в *главе 3*), теперь давайте снова обсудим их поподробнее. Обычно электронные кости состоят из электронной схемы и светодиодного дисплея. Это может быть либо семисегментный индикатор, на котором отображаются числа от 1 до 6 (рис. 7.18), либо семь отдельных светодиодов (рис. 7.19).

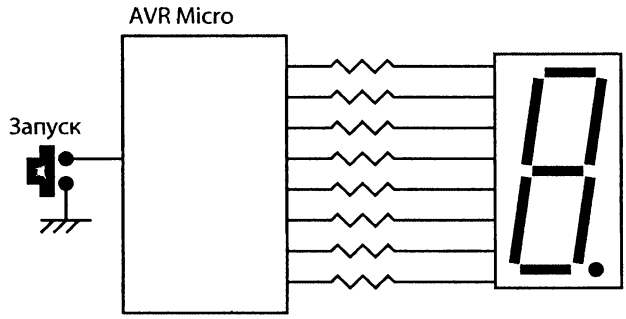


Рис. 7.18. Электронные игральные кости с семисегментным индикатором

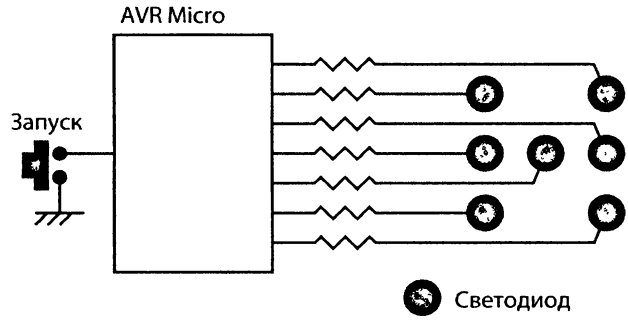


Рис. 7.19. Электронные игральные кости с отдельными светодиодами

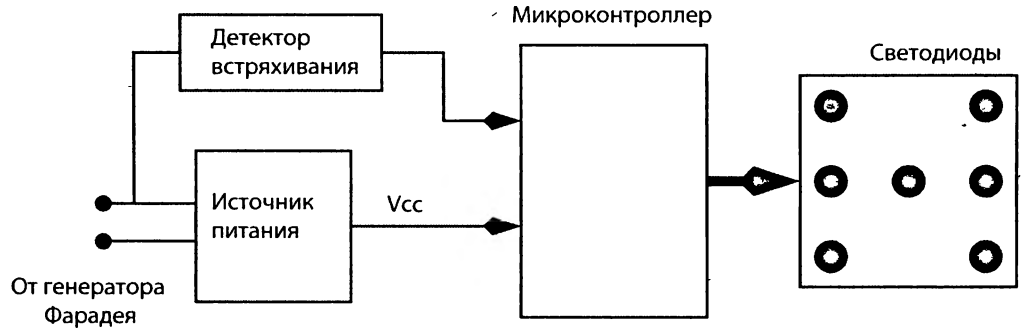


Рис. 7.20. Блок-схема электронных игральных костей (без батареек)

Обе схемы имеют кнопку, которую пользователь нажимает, когда он хочет "бросить кости". Кнопка запускает генератор случайных чисел, запрограммированный в микроконтроллере, после чего это число отображается на индикаторе. Когда пользователю требуется новое число, нужно еще раз нажать кнопку.

Для обеих этих схем необходим источник питания (например, сетевой адаптер с выпрямителем, конденсатором и стабилизатором на 5 В). Если кости должны быть переносными, то нужно предусмотреть питание от батареи (например, на 9 В). Можно взять и другие батарейки, но для работы устройства от одной батарейки (например, AA или AAA) понадобится повышающий преобразователь на 5 В.

Наконец, батарейки можно заменить генератором Фарадея. На рис. 7.20 приведена блок-схема таких электронных игральных костей.

Как уже неоднократно говорилось, чтобы получить энергию от генератора Фарадея, его нужно несколько раз встряхнуть. Можно создать "детектор встряхивания", который при помощи светодиодов будет выдавать случайное число. Поскольку питание имеется только тогда, когда вы трясете трубку, то необходим конденсатор, который продолжит питать в схему в течение некоторого времени и после встряхивания, когда на светодиодах отображается случайное число. После разряда конденсатора светодиоды выключаются. Увеличить время свечения светодиодов можно, повысив емкость конденсатора.

Спецификация проекта

Цель проекта — создать электронные игральные кости, которые будут показывать случайные числа при помощи светодиодов и без применения традиционных источников энергии (их заменит генератор Фарадея). Для некоторых настольных игр нужно две игральных кости, поэтому во втором варианте схемы предусмотрено два светодиодных индикатора.

Описание устройства

Принципиальная схема электронных игральных костей без батареек изображена на рис. 7.21. К разъему J1 подключен генератор Фарадея, вырабатывающий переменное напряжение, которое выпрямляется мостовой схемой на диодах D1–D4. Диоды 1N5819 — это диоды Шоттки с более низким напряжением включения, чем у обычных кремниевых выпрямительных диодов. Постоянное напряжение фильтруется и запасается в электролитическом конденсаторе C1 (4700 мкФ/25 В) и далее подается на вход стабилизатора LP2950-5V. Напряжение с выхода стабилизатора (5 В) служит для питания микроконтроллера и светодиодов нашей схемы.

Обнаружение факта встряхивания осуществляется при помощи диода D5, резистора R1 и стабилитрона D6. Входной переменный ток выпрямляется и через диод D5 проходят только положительные импульсы. Сигнал на выходе D5 показан на рис. 7.22.

Стабилитрон ограничивает импульсы напряжения до 4,7 В. Эти импульсы поступают на контакт микроконтроллера (PB0). Программа отслеживает наличие импульсов, и если вы перестанете трясти устройство, то импульсов больше не будет, микроконтроллер сделает вывод, что пользователь перестал трясти трубку, и выдаст случайное число, которое отображается на светодиодах.

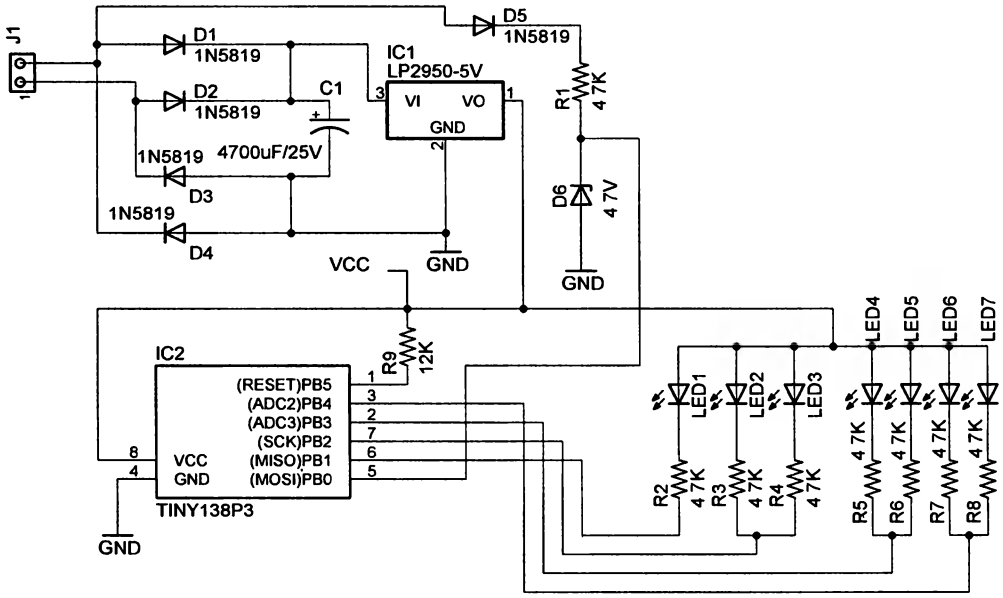


Рис. 7.21. Принципиальная схема электронных игральных костей (без батареек)

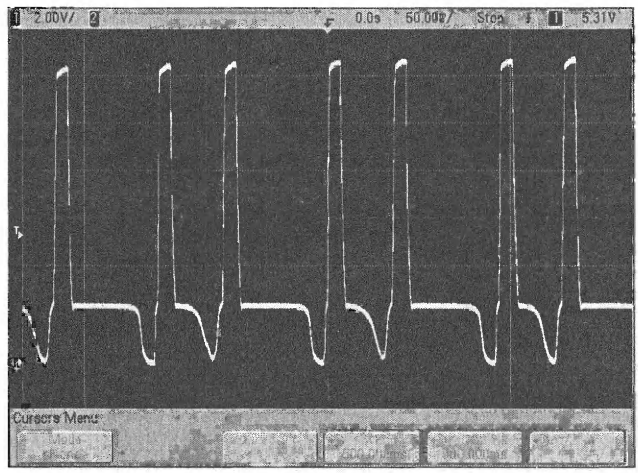


Рис. 7.22. Импульсы на выходе детектора встряхивания

Светодиоды LED1–LED7 сконфигурированы так (рис. 7.23), что для управления семью светодиодами нужно всего четыре контакта. Конечно, микроконтроллер не управляет всеми светодиодами по отдельности. Вместо этого четыре контакта микроконтроллера управляют группами: один, два, два и два светодиода. Когда пользователь начинает трясти трубку, напряжение на конденсаторе C1 растет и напряжение на выходе стабилизатора также возрастает. Когда напряжение питания стабилизируется, микроконтроллер начинает выполнять программу. Программа

инициализирует контакты и выключает все светодиоды. Она запускает также внутренний таймер Timer0. Значение таймера увеличивается при каждых восьми циклах тактовой частоты микроконтроллера. Затем микроконтроллер ждет, когда пользователь прекратит встряхивать трубку. После прекращения встряхивания микроконтроллер считывает значение таймера Timer0 и берет остаток от целочисленного деления этого значения на 6. Это дает значение от нуля до единицы. Результат этой операции (значение от нуля до пяти) преобразуется в значение от 1 до 6 и отображается на светодиодном дисплее. К этому моменту в конденсаторе остается достаточно заряда для того, чтобы светодиоды горели в течение примерно десяти секунд. После индикации числа микроконтроллер опять ждет встряхивания трубки. Чтобы получить новое случайное число, пользователь должен опять несколько раз встряхнуть трубку.

Поскольку работа таймера и встряхивание трубки никак не синхронизированы, полученное с таймера число получается случайным. В этом и состоит одна из основных идей устройства.

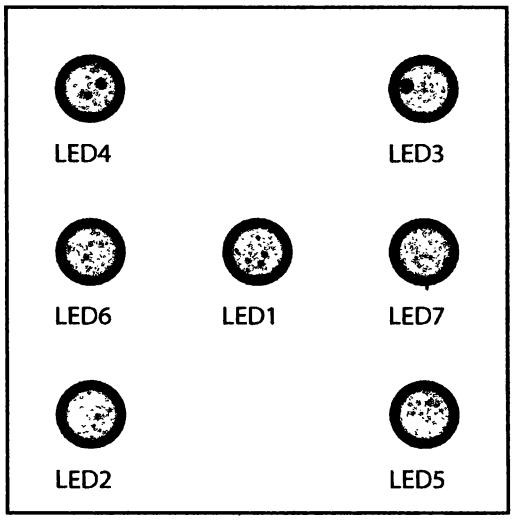


Рис. 7.23. Расположение светодиодов

Для многих настольных игр нужно две кости, поэтому мы видоизменили исходную схему и сделали два светодиодных дисплея. На рис. 7.24 показана принципиальная схема "двойных" костей. Здесь мультиплексируются два набора по семь светодиодов. Программа для микроконтроллера Tiny изменена так, чтобы две группы светодиодов обновлялись попеременно (с высокой частотой) и оба индикатора показывали случайные числа. Два случайных числа генерируются при помощи Timer0 (так же, как и в предыдущем случае, за исключением того, что первое число генерируется в момент начала сотрясения, а второе — по окончании сотрясения трубки). В остальном устройство функционирует аналогично предыдущему.

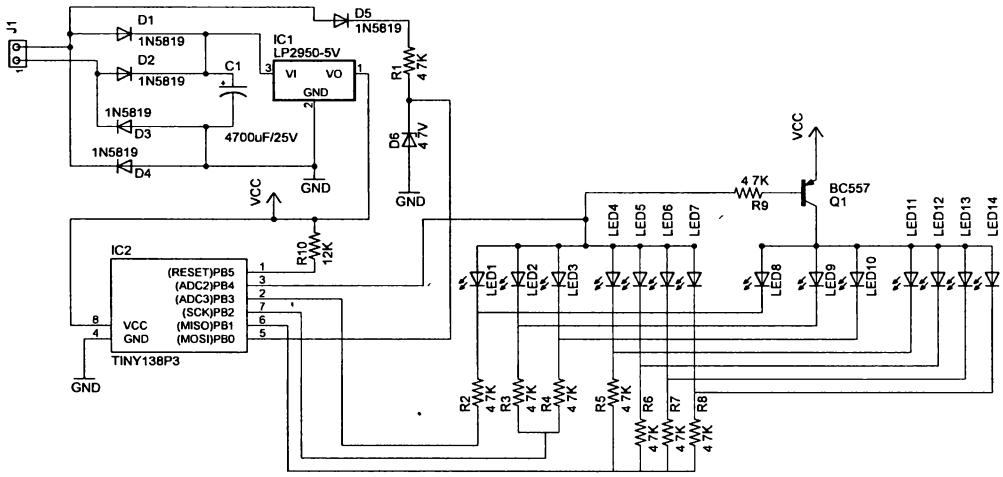


Рис. 7.24. Принципиальная схема игральных костей с двумя индикаторами

В обеих схемах микроконтроллер Tiny13 работает от внутреннего RC-генератора (запрограммированного для генерирования тактовой частоты в 128 кГц). Это минимальное значение тактовой частоты, которое может сгенерировать микроконтроллер Tiny13. Такой выбор частоты позволяет снизить энергопотребление микроконтроллера.

Конструкция

Первый вариант устройства был выполнен на печатной плате общего назначения (размерами примерно 2×10 см), как показано на рис. 7.25.

Готовая схема размещена внутри трубки из оргстекла. После сборки трубка со схемой и генератор Фарадея скрепляются вместе для удобства пользования. На рис. 7.26 изображено готовое устройство.

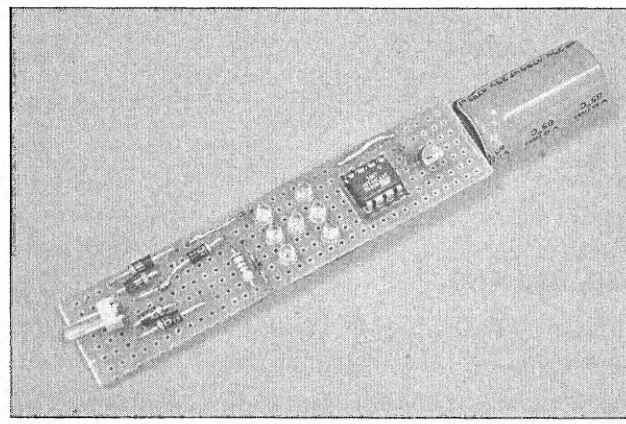


Рис. 7.25. Печатная плата игральных костей (вариант 1)

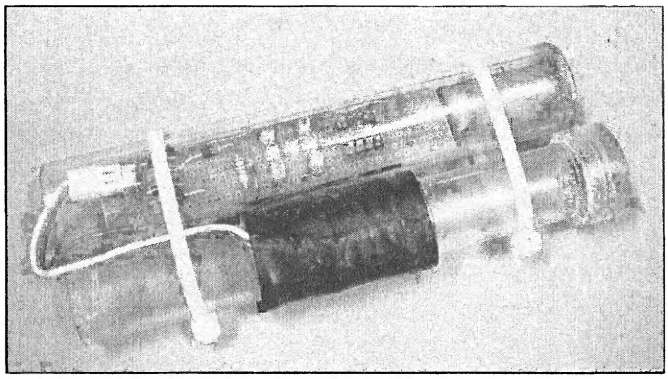


Рис. 7.26. Внешний вид игральных костей (вариант 1)

После тщательного тестирования этого образца мы решили заказать несколько печатных плат у стороннего изготовителя. На рис. 7.27 изображены устройства с одним и двумя индикаторами.

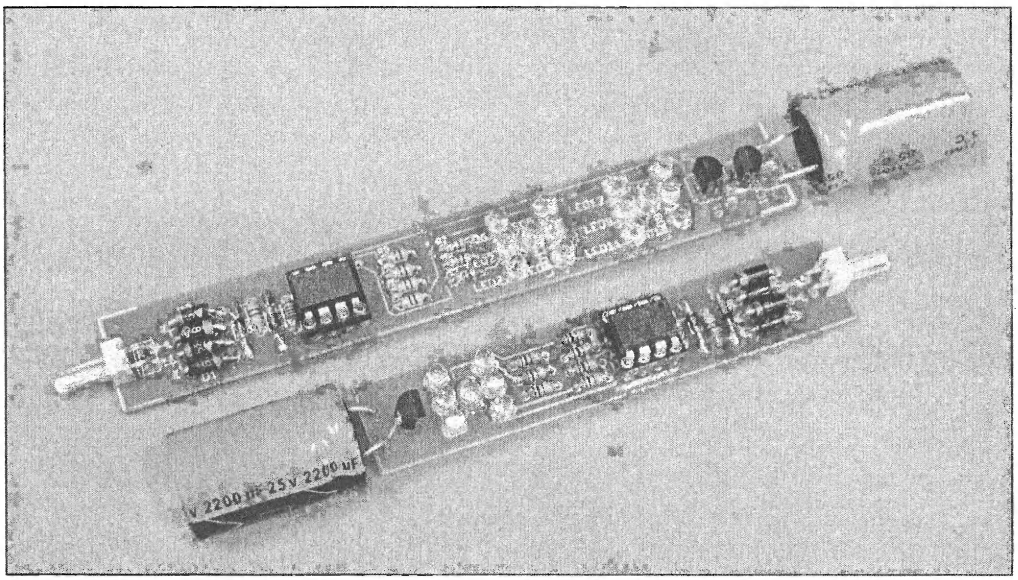


Рис. 7.27. Внешний вид игральных костей с одним и двумя индикаторами

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Один из важных фрагментов программы — главный бесконечный цикл, где микроконтроллер постоянно отслеживает импульсы на контакте PB0 (листинг 7.5).

Когда импульсы перестают появляться, он генерирует случайное число (при помощи Timer0) и отображает его на светодиодах. Такой же код имеется и для двойных костей. Задержка сформирована с помощью функции `_delay_loop_2` (в отличие от применявшихся ранее функций `_delay_ms` и `_delay_us`).

Листинг 7.5

```
const char ledcode[] PROGMEM= {0xfc, 0xee, 0xf8, 0xf2, 0xf0, 0xe2, 0xfe};
void main(void)
{
 unsigned char temp=0;
 int count=0;
 DDRB=0xfe; /* PB0 — входной контакт*/
 TCCR0B=2; /* делим на 8*/
 TCCR0A=0;
 TCNT0= 0;
 PORTB=254; /*выключаем все светодиоды*/
 while(1)
 {
  /* ждем, пока импульс не станет высоким */
  while ( (PINB & 0x01) == 0);
  _delay_loop_2(50);
  /* ждем, пока импульс не исчезнет */
  while ( (PINB & 0x01) == 0x01);
  _delay_loop_2(50);
  count=5000;
  while ( (count > 0) && ((PINB &0x01)==0))
  {
   count--;
  }
  if(count ==0) /* импульсов больше нет — отображаем случайное число */
   {PORTB=0xfe; /* выключаем все светодиоды */
  _delay_loop_2(10000);
  temp=TCNT0;
  temp= temp%6;
  temp =pgm_read_byte(&ledcode[temp]);
   PORTB=temp;
  }
 }
}
```

Микроконтроллер Tiny13 запрограммирован при помощи программатора STK500, а установка fuse-битов микроконтроллера показана на рис. 7.28.

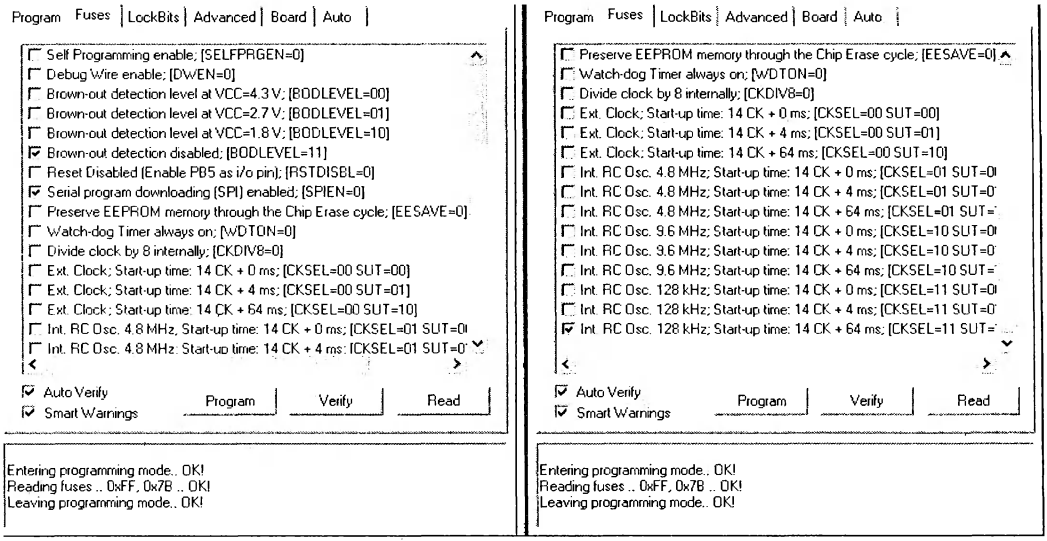


Рис. 7.28. Установка fuse-битов микроконтроллера

Проект 34. Игрушка, основанная на инерционности зрительного восприятия

Предлагаемое устройство основано на инерционном восприятии света человеческим глазом. Глаз "помнит" изображение в течение примерно 8 мс. Эта особенность человеческого зрения используется в некоторых игрушках, например в светодиодном волчке (см. проект 23 в *главе 5*).

В предлагаемом проекте мы создадим игрушку, которая показывает сообщения воздухе с помощью одного столбца светодиодов, причем она работает без батарей. Пользователь просто машет игрушкой в воздухе. Возвратно-поступательное ижение корпуса устройства обеспечивает генерирование рабочего напряжения свечение светодиодов. Шаблоны картинок (текст и графика) хранятся в памяти икроконтроллера.

Спецификация проекта

Цель проекта — создать безбатарейную игрушку, которая высвечивает текст картинки из памяти микроконтроллера. В игрушке должно быть семь диодов, сположенных в один столбец. Светодиоды управляются микроконтроллером, ерирующим световые шаблоны, которые наблюдатель воспринимает как сообние или картинку (из-за инерционности зрения). Рабочее напряжение поступает генератора Фарадея (описанного ранее в этой же главе). Блок-схема подобной рушки приведена на рис. 7.29.

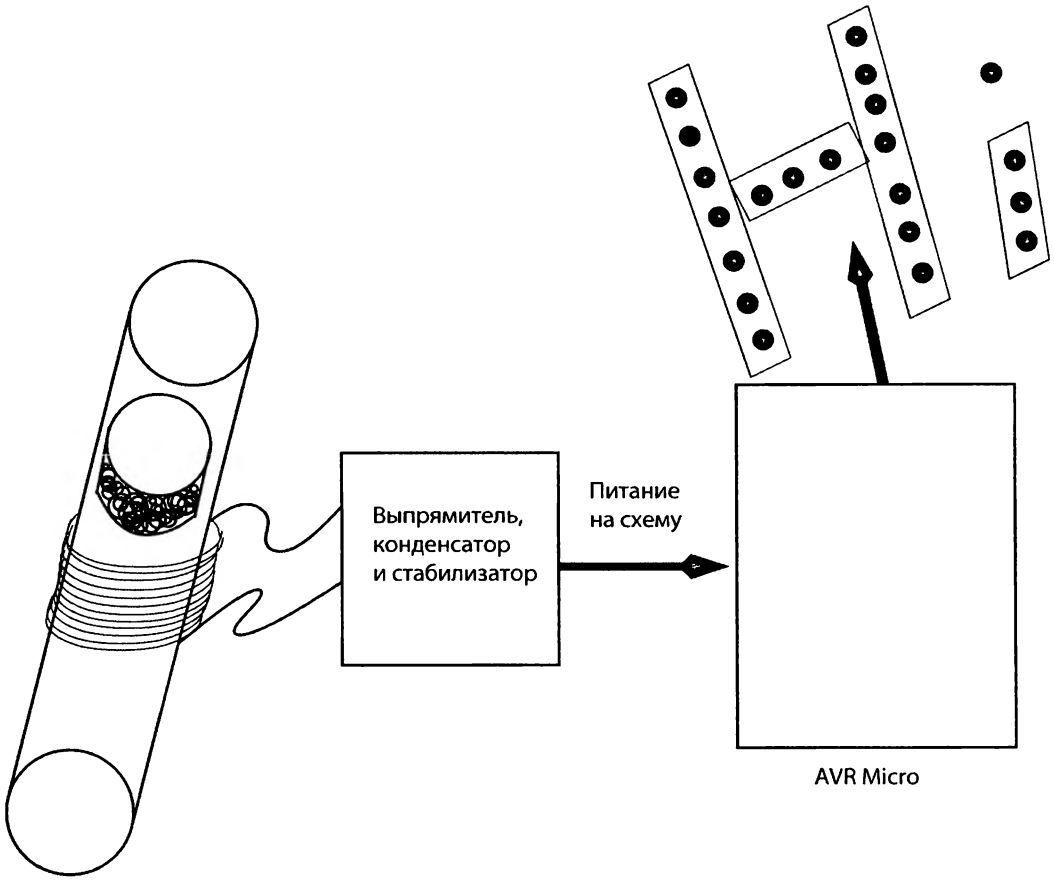


Рис. 7.29. Блок-схема игрушки, основанной на инерционности зрения

Описание устройства

Принципиальная схема нашей игрушки изображена на рис. 7.30. Устройство содержит мостовой выпрямитель на диодах D1–D4 и фильтрующий конденсатор 4700 мкФ/25 В. Стабилизатор напряжения LP2950-5V подает напряжение 5 В для питания микроконтроллера и светодиодов. Самый важный нюанс данной игрушки — способность воспроизводить один и тот же шаблон снова и снова. Для этого нужен сигнал синхронизации, который вырабатывает геркон, подключенный к трубке генератора Фарадея через разъем SL2. Генератор подключается к столбцу светодиодов. Когда игрушкой помахивают в воздухе, магниты двигаются по всей длине трубки. Геркон (находящийся на одном из концов трубки) замыкается при приближении магнитов. Микроконтроллер регистрирует это событие и синхронизирует включение светодиодов.

Аноды всех светодиодов подключены к источнику питания, т. е. для свечения светодиода на контакте микропроцессора должен присутствовать логический 0.

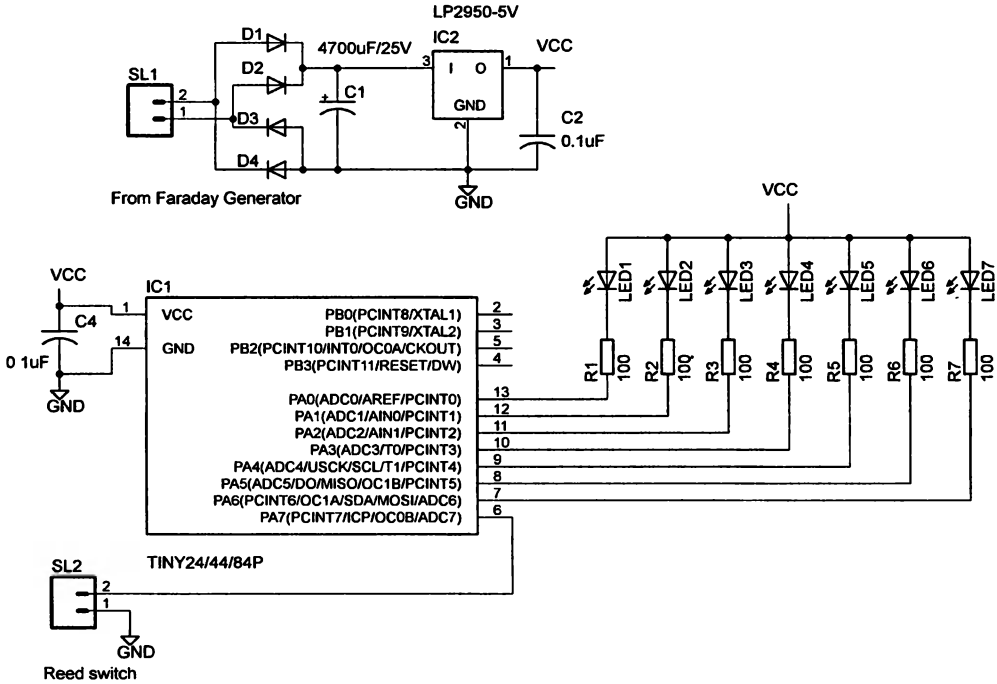


Рис. 7.30. Принципиальная схема игрушки, основанной на инерционности зрения

Конструкция

Игрушка собрана на печатной плате общего назначения (рис. 7.31). Печатная плата закреплена на пластиковой трубке. Пригодна любая трубка подходящего размера. Мы взяли упаковочную трубку для микросхем (кстати, на ней стоит маркировка Atmel). Перпендикулярно трубке мы закрепили генератор Фарадея. Затем мы приклеили геркон и залили его термоклеем (для дополнительной защиты). Геркон был закреплен на правой стороне трубки (если держать ее светодиодами к себе).

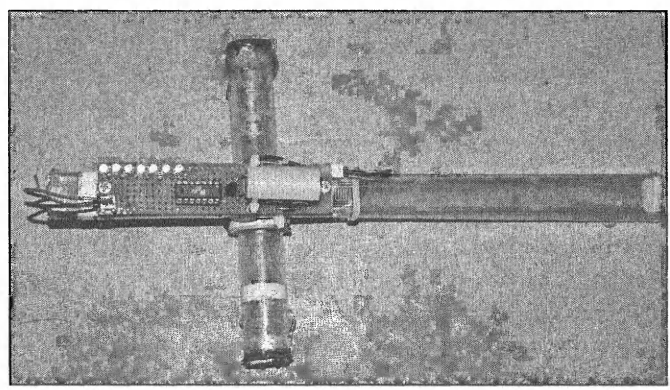


Рис. 7.31. Печатная плата игрушки, основанной на инерционности зрения

На рис. 7.32 показан геркон, приклеенный к концу трубки с генератором Фарадея. Геркон очень важен для нашей игрушки. Он состоит из двух контактов, покрытых магнитным материалом. В присутствии магнитного поля контакты замыкаются, когда поля нет — контакты размыкаются. На рис. 7.33 изображены несколько герконов.

Чтобы запустить игрушку, нужно быстро помахать ею в воздухе. Она начнет показывать запрограммированное сообщение (рис. 7.34).

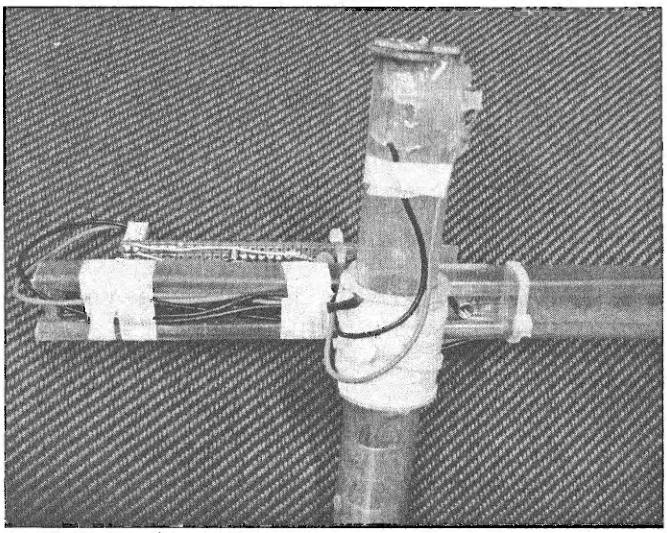


Рис. 7.32. Крепление геркона к трубке

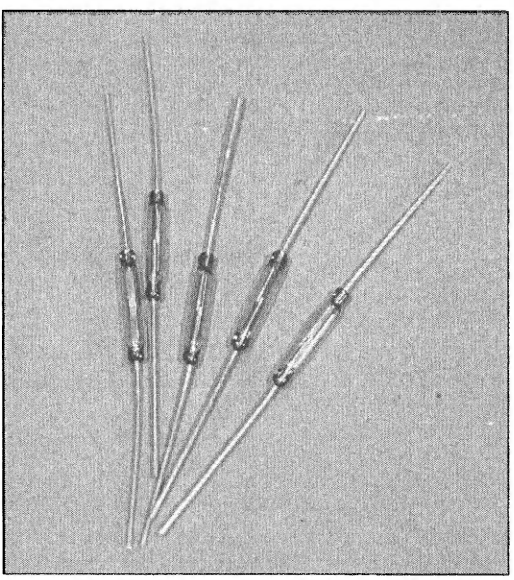


Рис. 7.33. Внешний вид герконов

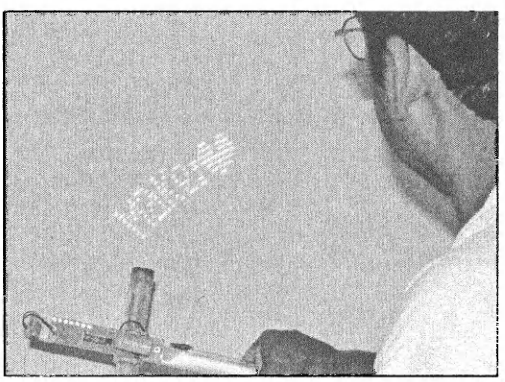


Рис. 7.34. Игрушка в действии

Программирование

Откомпилированный исходный код (вместе с файлом MAKEFILE) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Формирование отображаемой картинки — это важная функция программы. Шаблон кодируется побайтно и записывается в массив `MSG[]`. Размер массива (который зависит от длины отображаемого текста) определяется как константа `maxchar`. Наша программа формирует сообщение "Make:♥", поэтому размер `MSG` — 45 байтов. Поскольку в массиве равно 45 элементов, то константа `maxchar` устанавливается в значение 45. Если вы захотите, то сможете запрограммировать свое собственное сообщение, но тогда вам придется установить константу `maxchar` в соответствии с числом элементов массива `MSG[]`.

Программа ждет сигнала синхронизации от геркона и, когда он появляется, начинает отправлять элементы массива `MSG[]` на подключенные к PortA светодиоды. Каждый шаблон свечения длится несколько миллисекунд. Затем все светодиоды выключаются (перед отображением следующего байта массива). Так продолжается до полной выдачи массива на светодиоды. Затем программа ждет следующего сигнала синхронизации от геркона (листинг 7.6).

Листинг 7.6

```
//Make: H
const char MSG[] PROGMEM= {0x80, 0xfd,
  0xfb, 0xf7, 0xfb, 0xfd, 0x80, 0xff,
  0xdd, 0xae, 0xb6, 0xb6, 0xb9, 0xc3,
  0xbf, 0xff, 0x80, 0xf7, 0xeb, 0xdd,
  0xbe, 0xff, 0xe3, 0xcd, 0xad, 0xad,
  0xb3, 0xff, 0xff, 0x93, 0x93, 0xff,
  0xf3, 0xe1, 0xc0, 0xc0, 0xc1, 0x87,
  0x87, 0xc1, 0xc0, 0xc0, 0xe1, 0xf3,
  0xff, 0xff};
//размер: 45 байтов
#define maxchar 45
void main(void)
{
 unsigned char temp;
 DDRA= 0x7f;
 PORTA=255;
 while(1)
 {
  PORTA = 255;
  while( (PINA&0x80) == 0x80);
  while( (PINA&0x80) == 0);
```

```
  _delay_loop_2(3000);
  while( (PINA&0x80) == 0);
  _delay_loop_2(1000);
  for(temp=0; temp<maxchar; temp++)
  {
   PORTA= pgm_read_byte(&MSG[temp]);
   _delay_loop_2(150);
   PORTA=0xff;
   _delay_loop_2(50);
  }
 }
}
```

Работа устройства

Запустить игрушку несложно. Самое главное — правильно запрограммировать микроконтроллер, чтобы он отображал сообщение. Пример кодирования сообщения приведен на рис. 7.35. Бит PA7 соответствует входному сигналу от геркона, а PA6–PA0 управляют подключением светодиодов. Как уже упоминалось, для включения светодиода бит должен быть логическим 0. Если бит D7 порта равен 1, то остальные семь битов кодируются так, как показано на рис. 7.34, создавая изображение сообщения.

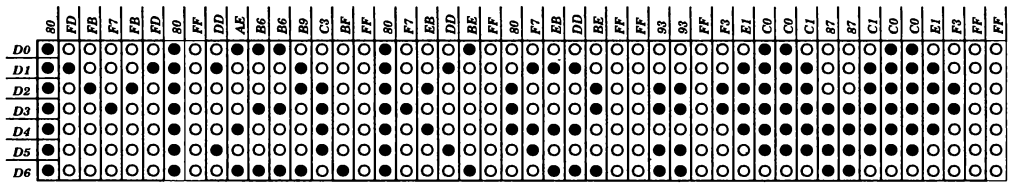
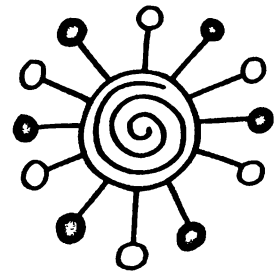


Рис. 7.35. Кодирование шаблонов светодиодов. Сообщение состоит из 45 байтов

Заключение

В этой главе мы рассмотрели несколько проектов, основанных на генераторе Фарадея. Как видим, этот генератор может дать достаточно энергии для питания различных устройств.

ПРИЛОЖЕНИЯ



Приложение 1

Программирование микроконтроллеров AVR на языке C

Прошли те дни, когда для каждой функции нужно было писать машинный код. Раньше для программирования микроконтроллеров использовался язык ассемблера. Код на этом языке нужно было писать примерно в таком виде: ADD Rd, Rs (для сложения регистров Rs и Rd). Затем ассемблер транслировал это в машинный код. Но при помощи ассемблера трудно писать сложные программы (из-за чрезмерной детализации). Ассемблер освоить трудно, и, кроме того, форма записи программы зависит от конкретного устройства.

С другой стороны, язык C (будучи языком высокого уровня) обеспечивает более высокий уровень абстракции и содержит богатую коллекцию библиотек. Язык C — это универсальный язык программирования, который может работать с любыми микроконтроллерами (при наличии для них кросс-компиляторов). При помощи языка C можно писать коды быстрее, создавать более легкие для понимания коды, а также достигать хорошего уровня переносимости программного обеспечения. Язык C всегда был и остается популярным и нужным языком программирования (благодаря его простоте, надежности и наличию компиляторов для него).

Широко применяется программирование встроенных процессоров мобильных телефонов, цифровых камер, медицинского оборудования, космических систем и домашней техники (микроволновых печей и стиральных машин) — вследствие их низкой стоимости и простоте кодирования. Интеллектом эти устройства обладают благодаря микроконтроллерам и встроенным в них программам. Эти программы должны работать не только быстро, но и при ограниченном объеме памяти. Семейство микроконтроллеров AVR состоит из множества микроконтроллеров с различными функциональными возможностями и разным объемом памяти. Несмотря на то, что программирование контроллеров похоже на любое другое программирование, все же есть отличия в стиле. При работе со встроенными устройствами требуется оптимизация. Следует помнить о тактировании и трудностях с размерами. Необходимо избегать объявления большого количества переменных. Более того, типы переменных необходимо тщательно подбирать (из-за ограниченного объема памяти в микроконтроллерах). Например, если переменная всегда имеет значение от 0 до 255, то следует объявить ее как `unsigned char` (а не `int` или `short`). Микросхемы семейства Tiny имеют ограниченное пространство памяти для программ и данных, поэтому при создании систем на их основе необходимо выполнять оптимизацию.

Нужно оптимизировать также и временную сложность кода. Необходимо помнить также и о тактировании (особенно при написании процедур для прерываний). При программировании на С следует постоянно держать в голове схему использования памяти и времени. Например, вычисления с плавающей точкой занимают больше времени, чем целочисленная арифметика, поэтому их лучше избегать и заменять вычислениями с фиксированной точкой.

Разница между ANSI C и встроенным C

В *главе 1* мы кратко описали языки ANSI C и встроенный C. Основные отличия этих языков обсуждаются в этом разделе.

Бесконечные и конечные программы

В среде операционной системы (ОС) задачи для процессора планирует операционная система; она же поддерживает его постоянную загрузку. Следовательно, когда ваша программа на языке С получает доступ к процессору, она выполняет свои задачи и после их завершения ОС забирает все те ресурсы, которые она выделила программе. В микроконтроллере операционной системы нет, поэтому в нем работает только ваша программа. Следовательно, она никогда не должна завершаться. Иначе говоря, программа должна поддерживать загрузку процессора. Для этого программа на языке С обычно работает в бесконечном цикле типа `while (1)` или `for (;;)`. Эти циклы никогда не завершаются, поэтому программа работает постоянно.

Включаем разные файлы для разных микроконтроллеров

`#include` — это директива препроцессора, которая дает указание компилятору включить определенные файлы заголовков, которые разрешают доступ к библиотечным функциям. Это стандартная функция языка С. При создании программ для контроллеров мы включаем файлы заголовков для конкретных микросхем (которые дают нам доступ к некоторым библиотечным функциям и рассказывают компилятору о различных функциональных возможностях микроконтроллера (на котором мы собираемся выполнять программу)). Директиву `#include` можно применять для определения различных макросов, которые мы будем использовать для доступа к специфическим функциям процессора из языка С (и к которым невозможно получить доступ при помощи стандартных конструкций языка С). Например, это могут быть названия регистров ввода/вывода и битов микроконтроллера AVR. Мы подробно объясним применение файлов заголовков и библиотечных функций для микропроцессоров AVR в последующих разделах.

Минимальное использование консольных функций

Стандартные функции ввода/вывода в программировании контроллеров обычно не используют, поскольку встроенные системы не имеют (по умолчанию) таких дисплеев, как персональные компьютеры. Функции консольного ввода/вывода

(printf, scanf, putchar, gets, puts и т. п.) в большинстве встроенных компиляторов существуют, но указывать их в исходном виде нельзя — сначала их нужно привязать к определенным утилитам контроллера.

Типы данных и операторы

Язык C предоставляет базовый, стандартный и минимальный наборы типов данных. Однако при помощи структур и объединений можно создавать более сложные типы данных. Переменная служит для хранения данных, попадающих в определенный диапазон (соответствующий типу объявленной переменной). Когда переменная объявляется, ей выделяется определенный объем памяти.

Вот основные типы данных языка C:

- char — тип данных размером в один байт, который обычно используется для представления символов из набора ASCII. Первоначально набор ASCII содержал 127 символов. Тип signed char имеет диапазон от –128 до 127, а unsigned char — от 0 до 255.
- short — целый тип, который можно записать также как short int. Обычно он имеет длину два байта. Signed short имеет диапазон от –32768 до 32767, а unsigned short — от 0 до 65535.
- int — размер целого может быть 16 или 32 бита (в зависимости от архитектуры). В контроллерах AVR тип int объявляет 16-битовую переменную с диапазоном значений от –32768 до 32767. Аналогичным образом, unsigned int имеет значения от 0 до 65535.
- long — длинное целое с размером 32 бита (чаще всего). Диапазон значений — от –2147483648 до 2147483647 (в формате со знаком) и от 0 до 4294967295 (в формате без знака). В переменных long обычно хранят большие значения.

Тип int может быть эквивалентен типам short или long (в зависимости от архитектуры). Однако short никогда не может быть больше, чем int, а int никогда не может быть больше, чем long.

Флаги микроконтроллера — это переменные для хранения состояния выполняемой программы. Обычно в них размещают ограниченные значения. Поэтому всегда лучше объявлять эти флаги как char, чтобы сэкономить память.

Типы с плавающей точкой

В языке C есть также типы данных float и double, которые хранят большие десятичные числа. В отличие от записи с фиксированной точкой, в этих типах данных положение десятичной точки не зафиксировано (отсюда и название). Число типа float имеет размер 4 байта и диапазон значений от –3.4e38 до +3.4e38. Для больших значений в языке C есть тип размером в 8 байтов (который называется double) с диапазоном от –1.7e308 до +1.7e308. Если и этого недостаточно, предусмотрен тип long double, который имеет еще больший диапазон значений. Однако в случае микроконтроллеров чисел с плавающей точкой следует избегать, поскольку большинство микросхем не имеет блока вычислений с плавающей точкой (и они выполняются при помощи программного эмулятора). Такой эмулятор приходится включать в состав кода, что увеличивает его размер и снижает производительность.

Переменные и константы

Переменная — это имя для зарезервированной области памяти, которая предназначена для хранения значения определенного типа. Синтаксически в языке C тип переменной предшествует ее имени. Вот пример объявления целочисленной переменной `count`, которая инициализируется значением 10 и неинициализированного целого числа с именем `number`:

```
int count = 10, number;
```

По умолчанию `int` представляет собой тип со знаком, следовательно, для переменной `count` допустимы значения от –32768 до +32767. Язык C чувствителен к регистру, то есть `a` и `A` — это разные переменные. Имена переменных могут содержать цифры и знаки подчеркивания, но не могут начинаться с цифры. Имена переменных должны отличаться от ключевых слов языка C. С другой стороны, объявление константы резервирует область памяти фиксированного размера (во время компиляции и во время выполнения). Константы могут быть типов `char`, `int` или `float`. Константа `A` типа `char` имеет десятичное значение 65. Целое число 120 в коде — это константа типа `int`. Суффикс `L` или `l` перед константой типа `int` (например, `1300L`) означает, что это `long int`, суффикс `u` или `U` означает, что константа не имеет знака. Таким образом, `300U` представляет собой константу типа `unsigned long int` со значением 300. Точно так же и для констант типа `float` — может быть добавлен суффикс `f` или `F`. Константы часто записывают в двоичной, восьмеричной или шестнадцатеричной системах, добавляя к значению приставку `0b` (`0B`), `0` или `0x` (`0X`). Так, `0xff` представляет десятичное значение 255.

Вот пример представления константы:

```
0b00001111 = =      017      = =       0x0f         = =     15
(двоичное)      (восьмеричное)     (шестнадцатеричное)     (десятичное)
```

Операторы

В следующих разделах описываются операторы языка C.

Оператор присваивания (=)

Это двуместный оператор, который копирует значение справа в значение слева. Слева от оператора присваивания не может стоять константа:

```
a = 2;
```

присваивает 2 переменной `a`

```
a = x + 1;
```

записывает значение `x+1` в переменную `a` (здесь `x` — другая переменная).

Математические операторы

В языке C есть как двуместные, так и одноместные математические операторы. Для использования математических операторов необходимо знать порядок выполнения этих операторов (который определяет последовательность их выполнения).

Во избежание путаницы можно указывать скобки. Математический оператор с целым числом и числом с плавающей запятой даст результат в виде числа типа `float`. Тип `int` сначала внутренним образом преобразуется во `float`, а затем вычисляется результат. Двуместные операции перечислены в следующей таблице:

Обозначение	Операция
+	Сложение
-	Вычитание
/	Деление
*	Умножение
%	Остаток

Одноместные операторы языка C описаны в следующей таблице:

Обозначение	Оператор
`count++`	Постинкремент
`count++`	Прединкремент
`count++`	Постдекремент
`count++`	Преддекремент

Оператор постинкремента (или постдекремента) увеличивает (уменьшает) значение переменной после вычисления выражения, а операторы прединкремента увеличивают его перед выполнением текущей команды.

Логические операции

В языке C есть логические операторы, которые перечислены в следующей таблице:

Обозначение	Операция
!	Логическое NOT (одноместная операция)
&&	Логическое AND (двуместная операция)
||	Логическое OR (двуместная операция)

Выражение

```
<condition 1> && <condition 2>
```

возвращает значение "истина" в том случае, когда истинны оба условия (`<condition 1>` и `<condition 2>`); в противном случае возвращается "ложь".

Точно так же выражение

```
<condition 1> | | <condition 2>
```

возвращает "ложь" в том случае, когда ложны оба условия (`<condition 1>` и `<condition 2>`); в противном случае возвращается "истина".

Оператор NOT инвертирует следующую за ним логику. В языке C любое ненулевое значение возвращает "истина", а нуль — "ложь".

Пример

```
int x=2;
if(!x)
{
 <некий код>
}
```

В этом примере, поскольку `x` не равен 0, то возвращается "истинно", а `!x` окажется равен "ложь" и блок `if` выполнен не будет.

Операции сравнения

В следующей таблице перечислены двуместные операции, которые возвращают либо 0 ("ложь"), либо 1 ("истина").

Обозначение	Операция
==	Равно
!=	Не равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно

Побитовые операции

Эти операции работают с отдельными битами и обычно используются с беззнаковыми типами. Подробнее они рассматриваются в последующих разделах. Самые распространенные битовые операции описываются в следующей таблице:

Обозначение	Операция
~	Отрицание (одноместная операция)
&	Побитовое AND
|	Побитовое OR
^	Побитовое XOR
>>	Сдвиг битов вправо (эквивалентно делению на 2)
<<	Сдвиг битов влево (эквивалентно умножению на 2)

Побитовые операции критичны для работы с портами ввода/вывода и регистрами микроконтроллеров AVR. Их использование подробно обсуждается в следующем разделе.

Эффективное управление портами ввода/вывода

С этого раздела мы начинаем писать код на языке C. Начнем мы с портов ввода/вывода, которые являются самыми критичными аспектами управления. Это такие объекты, которые принимают ввод от пользователя и отображают результаты вашей программы. Остальная часть обработки находится внутри контроллера.

Микроконтроллеры AVR имеют множество портов ввода/вывода с именами PORTA, PORTB, PORTC и т. д. Число контактов каждого порта ввода/вывода не более восьми. Каждый порт имеет три связанных с ним регистра:

- DDR*x* — регистр направления передачи данных;
- PORT*x* — регистр вывода данных;
- PIN*x* — регистр ввода данных,

где *x* — имя порта. Каждый из этих трех регистров 8-разрядный и предназначен для манипулирования восемью битами порта. Биты этих трех регистров (да и вообще всех регистров AVR) нумеруются от 0 (самый младший бит) до 7 (самый старший бит). DDR*x* конфигурирует контакты как вход или выход (в зависимости от значения бита — 1 (выход) или 0 (вход)). Регистр PORT*x* используется для вывода (если контакт объявлен как выход) или для включения (1)/выключения (0) нагрузочных резисторов (если контакт объявлен как вход). Регистр PIN*x* служит для чтения значения логического уровня на контактах (если они объявлены как вход).

В компиляторе WinAVR GCC все периферийные регистры AVR доступны через макросы и пишутся заглавными буквами. Таким образом, следующие эквивалентные строки кода объявляют все контакты PORTB выходами:

```
DDRB = 255; // В десятичной системе
DDRB = 0xff; // В шестнадцатеричной системе
DDRB = 0b11111111; // В двоичной системе
```

Обратите внимание, что для всех восьми контактов конкретного порта есть только один набор регистров, что создает определенную проблему в том случае, когда вы хотите работать с ними по отдельности.

Например, оператор:

```
PORTB = 0b00000010;
```

выставляет высокий уровень на контакте 1 порта PORTB. Теперь предположим, что мы хотим подать логическую 1 на контакт 3 того же самого порта. Вы можете написать:

```
PORTB = 0b00010000;
```

но при этом на контакт 1 подается логический уровень 0 (а вы не собирались этого делать). Следовательно, должен быть способ отслеживания отдельных битов — и именно здесь в игру вступают операторы &, ~, |, ^, << и >>.

Использование логических операторов

Все эти операторы являются двуместными, т. е. они работают с битами операндов и сохраняют результат в соответствующих битах указанной выходной переменной.

Битовый оператор *NOT*

Этот оператор инвертирует биты операнда. Это одноместный оператор:

```
a = 0b00001111;
b = ~a; // b = 0b11110000
```

Битовый оператор *OR*

Побитовый оператор OR выполняет операцию логического ИЛИ с битами операндов. Например, предположим, что у нас есть два байта данных: `a = 0b10101010` и `b = 0b01010101`. Тогда выражение `c=a|b` выполняет логическое ИЛИ по битам `a` и `b`, а затем сохраняет результат в `c`. Результат будет равен `0b11111111`.

Мы уже обсудили проблему при выставлении контакта 3 порта PORTB в логическую 1 (не затрагивая значения контакта 1 того же самого порта). Решить эту проблему можно при помощи побитового ИЛИ:

```
PORTB = PORTB|0b00001000;
```

Как видно, выполняется логическая операция ИЛИ по содержимому PORTB и числу с единицей в третьем бите. Мы знаем, что если какой-то из операндов оператора OR равен 1, то результат будет равен 1 (независимо от второго операнда). Поэтому данная операция выставляет третий бит порта PORTB в 1. Но что будет с остальными битами? Остальные биты порта PORTB будут подвергнуты побитовому OR с нулями, а при этом результат всегда равен второму операнду. Следовательно, мы добились желаемого. Этот оператор можно записать кратко:

```
PORTB |= 0b00001000;
```

Побитовый оператор *AND*

Побитовый *AND* выполняет логическую операцию И с битами операндов. Предположим, что у нас как и в предыдущем примере есть два байта данных `a = 0b10101010` и `b = 0b01010101`. Тогда выражение `c=a&b` выполняет логическое И по битам `a` и `b`, а затем сохраняет результат в `c`. Результат равен `0b00000000`.

Мы обсудили, как установить отдельные биты в 1 при помощи операции OR. Аналогично биты можно устанавливать в 0 при помощи оператора AND:

```
PORTB = PORTB&~ (0b00001000);
```

Оператор ~ дает обратный побитовый код, поэтому эквивалентным оператором будет:

```
PORTB = PORTB& (0b11110111);
```

Выполняется логическая операция И по содержимому порта PORTB и числу с нулем в третьем бите. Мы знаем, что если любой из операндов оператора AND ра-

вен 0, то результат тоже будет равен 0 (независимо от второго операнда). Эта операция выставляет третий бит порта PORTB в 0. А что же будет с остальными битами? Остальные биты PORTB подверглись операции AND с единицами, а если один из операндов операции AND равен 1, то результат равен второму операнду. Следовательно, мы получили решение и для этой проблемы.

Побитовый оператор *XOR*

Побитовый XOR (исключающее ИЛИ) выполняет логическую операцию XOR по битам двух операндов. Как и в предыдущих случаях предположим, что у нас есть два байта данных `a = 0b10101011` и `b = 0b01010101`. Тогда выражение `c=a^b` выполняет логическое XOR по битам `a` и `b`, а затем сохраняет результат в `c`. Значение `c` после этого будет равно `0b11111110`.

При программировании контроллеров AVR оператор XOR используется для переключения битов порта без проверки условий в операторе `if-else`. Если один из операндов этого оператора равен 1, то в результате получается обратный код второго операнда; а если один из операндов равен 0, то результат равен второму операнду. Поэтому контакт 4 порта PORTC можно переключить следующим образом:

```
PORTC ^= 0b00010000;
```

Использование операторов сдвига вправо и влево

Эти операторы сдвигают содержимое двоичной переменной влево или вправо на указанное число битов.

Предположим, что начальное содержимое переменной `a` показано в следующей таблице:

Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0
1	1	1	1	0	1	1	1

Мы выполняем операцию сдвига влево на четыре бита:

```
a = a<<4;
```

После операции содержимое `a` будет таким, как показано в следующей таблице:

Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0
0	1	1	1	0	0	0	0

Вы видите, что содержимое переменной `a` было сдвинуто влево на 4 бита и четырем младшим битам были присвоены нули.

Давайте рассмотрим другой пример:

```
b = 1<<2;
```

Этот оператор говорит компилятору о том, что нужно сдвинуть содержимое единицы влево на 2 бита. Первоначально единица представляется так, как показано в следующей таблице:

Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0
0	0	0	0	0	0	0	1

После операции содержимое переменной b будет таким:

Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0
0	0	0	0	0	1	0	0

Второй бит переменной b стал равен 1. Это стандартный способ манипуляции отдельными битами. Применение побитового XOR к любому порту и переменной b (с последующим сохранением результата обратно в порт) установит второй бит в 1 и не повлияет на другие биты (как это уже было разъяснено ранее). Например, рассмотрим следующие операторы для объявления седьмого бита порта PORTD как выходного:

```
DDRD |= 1<<7;
```

Все биты регистров ввода/вывода контроллеров AVR имеют имена, которые можно найти в спецификации контроллера. Эти имена затем транслируются компилятором WinAVR в макросы. Например, контакты регистра PORTD имеют имена от PD7 до PD0 и объявляются в компиляторе так:

```
#define PD7 7 and others alike
```

Таким образом, следующая строка кода также устанавливает третий бит порта PORTB в единицу:

```
PORTB |= 1<<PB3;
```

Изучив этот материал, вы сможете понять все коды проекта 1 из *главы 1*.

Несколько важных файлов заголовков

- io.h — файл находится в каталоге WinAVR/avr/include/avr, но по умолчанию компилятор в поиске файлов заголовков ведет просмотр только до каталога include. Следовательно, вы должны задать относительный путь и включить этот файл как avr/io.h. Этот файл считывает имя микроконтроллера из файла MAKEFILE и автоматически включает соответствующий файл заголовков (специфичный для соответствующего контроллера), в котором определяются различные адреса портов ввода/вывода, имена битов, адреса векторов и т. д.
- interrupt.h — файл включается как avr/interrupt.h; в нем хранятся различные определения для поддержки прерываний в вашем коде.
- pgmspace.h — файл включается как avr/pgmspace.h и служит для хранения и извлечения констант из памяти программ.

- delay.h — файл включается как util/delay.h и предоставляет две точных функции задержки: `_delay_us` и `_delay_ms`. Обратите внимание, что существует предел задержки, который может быть внесен этой функцией, зависящей от тактовой частоты. Подробности смотрите в самом файле. Чтобы присоединить этот файл, вы должны указать частоту, на которой работает ваша система (это делается путем объявления макроса F_CPU равным тактовой частоте).

Функции

Функция — это блок операторов, который выполняет некую изолированную задачу. Это похоже на обычную жизненную ситуацию — каждый человек делает свою особую работу. В реальной жизни мы зависим от других людей, которые выполняют обычные жизненно важные задачи. Точно так же и программу на языке C можно рассматривать как набор функций, которые выполняют частные задачи.

Для чего нужны функции?

Функции позволяют повторять выполнение кода. Например, если в программе нам нужно делать задержку между какими-то двумя действиями, то будет совершенно нерационально писать одни и те же команды снова и снова. Вместо этого можно создать функцию, команды которой реализуют нужную задержку. Разбиение программы на функции повышает ясность программы и облегчает отслеживание выполняемых задач. Разбиение программ на небольшие функции (выполняющие логически изолированные задачи) — хороший стиль программирования.

Как работают функции

Использование функций состоит из трех шагов:

- Объявление (называется также прототипом функции) — определяет вид функции. В объявление входят: тип возвращаемого значения, имя функции, принимаемые аргументы. После выполнения задачи функция может как возвратить, так и не возвращать значений (в зависимости от указанного типа возвращаемого значения). Имя функции может быть любым (кроме ключевых слов языка C). Аргументы функции — это переменные, которые передаются в функцию и используются ею для выполнения своей задачи.

Пример

```
int GetMax(int a, int b);
```

Этот оператор объявляет функцию, которая принимает два целочисленных аргумента и возвращает целочисленное значение. По названию функции видно, что она возвращает максимальное из значений `a` и `b`.

- Определение — это последовательность команд, определяющих поведение функции. Она должна начинаться с тех же самых имени и типа, которые содержатся в объявлении.

Пример

```
int GetMax(int a, int b)
 {
    if(a > b) return a;
    else return b;
 }
```

- Вызов — это фактическое использование функции в программе. При вызове функции управление передается в тело функции (ее определение). Функцию можно вызвать из любой другой функции; она может даже вызывать сама себя (рекурсия). Функция вызывается путем написания ее имени с именами тех переменных, которые должны ей передаваться.

В программах на языке С всегда присутствует функция с именем `main`, с которой всегда начинается выполнение программы. Функция может вызывать другие функции (или саму себя) столько раз, сколько это необходимо. Функции следует объявлять до того, как они вызываются программой. Однако если внутри файла функция была определена до ее вызова, тогда объявление данной функции не нужно.

Обработка прерываний

Прерывание — это механизм управления выполнением, который имеется в большинстве микроконтроллеров. Многие события внешнего мира происходят асинхронно (т. е. не связаны с тактовым сигналом): нажатие кнопки, посылка байта через последовательный порт, переполнение таймера и т. д. Прерывание говорит процессору о том, что произошло некое событие, чтобы процессору не нужно было постоянно опрашивать это событие. Например, существуют два способа, которыми процессор может узнать, нажата кнопка или нет. Один способ — постоянно сканировать состояние кнопки. Другой — сообщить процессору о нажатии кнопки путем прерывания выполнения главной программы.

Если устройство прерывает работу процессора, то выполнение главной программы останавливается и процессор переходит на соответствующую процедуру, которая называется ISR (процедура обработки прерывания). После выполнения необходимых действий прерванное выполнение программы возобновляется.

В контроллерах AVR есть множество прерываний (как синхронных, так и асинхронных). Следует разрешить необходимые прерывания и саму возможность прерывания. Во время прерываний и вызовов процедур в стеке сохраняется адрес возврата, который является адресом следующей команды в главной программе. Стек выделяется в памяти данных, следовательно, его размер ограничен только общим размером памяти и степенью ее загрузки. Все пользовательские программы должны инициализировать указатель стека SP (в подпрограмме сброса, которая выполняется до исполнения процедур или прерываний).

Прототип для прерывания

При программировании на языке С обрабатывать прерывания просто, поскольку для прерываний имеются процедуры под разными именами и компилятор сохраняет содержимое регистра состояния перед выполнением ISR. Когда процессор переходит на ISR — прерывания автоматически запрещаются (чтобы никакое другое прерывание не могло произойти). По завершении ISR прерывания автоматически разрешаются. В языке ассемблера возвращение из ISR выполняется при помощи команды `reti`, которая разрешает прерывания (и тем самым отличается от оператора `ret`). Далее показан пример включения таймера на микроконтроллере AVR (для прерывания по переполнению). Задействован 8-разрядный таймер Timer0.

Пример

```
// Инициализация Timer0

//разрешение прерываний
   sei();
//установка частоты таймера = fclk/1024
   TCCR0 = (1<<CS02) | (1<<CS01);
   TCNT0 = 0x00;
//разрешаем прерывание по переполнению таймера
   TIMSK | = (1<<TOIE0);
//Определение процедуры прерывания по переполнению таймера
ISR(TIMER0_OVF_vect)
{
   PORTD = 0xff; //обычные команды и операции
   <некий код>
}
```

Каждое прерывание имеет свое уникальное название вектора (которое пишется в скобках `ISR( )`); эти имена определяются в файле заголовков interrupt.h. Таким образом, необходимо присоединять этот файл заголовков при написании прерываний и работе с ними. Процессор вызывает процедуру `ISR(TIMER0_OVF_vect)` каждый раз, когда происходит переполнение Timer0 и прерывание.

Массивы

Массивы — это группы элементов данных одного типа. Массивы могут содержать элементы типов `char`, `int`, `float`, `double`, указатели, структуры, объединения и т. д. Они предоставляют способ объявления большого количества переменных (без необходимости давать имя каждой из них). В языке С массив переменных в памяти располагается последовательно. Такое расположение элементов важно для доступа к ним и передачи их в функции. При объявлении массива элементов

его размер (или число элементов) должен быть постоянным. Например, массив из 100 переменных типа `int` объявляется следующим образом:

```
int x[100] ;
```

Здесь `int` указывает тип переменной, а `x` — имя переменной. 100 — это число элементов типа `int` в данном массиве (его часто называют размерностью (размером) массива). Доступ к элементам массива производится при помощи индекса в квадратных скобках `[ ]`. Индекс говорит нам о положении элемента в массиве (начиная с нуля). То есть это переменные `x[0]`, `x[1]`, `x[2]` ... `x[99]`, с которыми можно работать так же, как и с любыми другими обычными переменными. Массив можно инициализировать значениями:

```
int marks[5] = { 90, 97, 94, 80, 91} ;
float weight[] = { 50.3, 55.4, 13.2, 20.0} ;
```

Следовательно, `marks[2]` — это 94, а `marks[3]` — 80. При инициализации массива компилятор самостоятельно вычисляет его размер (если он не указан). Обратите внимание, что в языке C при доступе к элементу нет метода для определения выхода индекса за границы массива. Это приведет к выходу за область памяти массива и будет иметь непредсказуемые результаты.

Утилиты языка C

В этом разделе мы обсуждаем некоторые дополнительные свойства языка C, которые облегчают программисту написание кодов и улучшают выполнение программы.

Препроцессор языка C

В полном соответствии со своим названием — это программа, которая обрабатывает исходную программу до компиляции. Итак, что же делает препроцессор? Он обрабатывает специальные команды языка C, которые называются командами препроцессора (их иногда называют также директивами). Эти директивы дают бóльшую гибкость и удобство при написании программ. Далее рассмотрим некоторые наиболее часто встречающиеся директивы программирования.

Подключение файлов

Вот директива `#include`, которая позволяет включить один файл в другой:

```
#include "filename"
```

Здесь `filename` — это имя существующего файла, который нужно подключить. Данный файл должен существовать в текущем каталоге (или в каталогах указанного пути). Обычно так подключают файлы заголовков (хотя точно так же можно подключать исходные файлы). Вот другой способ использования этой директивы:

```
#include <filename>
```

На этот раз препроцессор ищет файл `filename` только в тех каталогах, которые указаны в переменной среды PATH.

Макроподстановка

Теперь обсудим еще одну важную директиву: `#define`, которая предназначена для замены выражений, операторов или констант в программе.

Пример

```
#define PI 3.1415
void main ( )
{
  float circle_area, circle_circumference; int radius = 5;
  circle_area = PI * radius * radius;
  circle_circumference = 2 * PI * radius;
}
```

Здесь препроцессор просто заменяет в коде лексему `PI` значением 3,1415. Однако если поместить PI в кавычки ("PI"), то замещение не происходит. Макрос может использоваться с аргументом или заменять сложное выражение. Длинное выражение можно продолжить на следующую строку, если поместить в конце продлеваемой строки символ `\`. В следующем примере макрос имеет аргумент.

Пример

```
#define delay(x) for(i=0;i<100*x;i++) \
asm("nop");
void main( )
{
  int i;
  ......
  ......
  delay(10); //использование макроса с аргументом
  ......
  ......
}
```

Для чего же нужна директива `#define` в программах? Она облегчает чтение, модификацию и портирование программ. Например, многократно используемую в программе константу можно заменить на любое значение, и для этого не придется изменять ее во всех тех местах, где она была указана. Макросы — это хороший стиль программирования.

Макросы и функции

Как вы уже, вероятно, заметили, макросы с аргументами можно использовать подобно функциям. Но в отличие от функций макросы заменяются везде, где они

указаны (без вызовов и возвратов). Это, безусловно, ускоряет выполнение программы, но увеличивает размер выходного файла. Мораль такова: заменяйте функции макросами в том случае, когда они небольшие и встречаются в программе много раз.

Макросы для AVR

При создании программ на языке C для микроконтроллеров AVR существуют стандартные файлы заголовков. Если вы посмотрите на эти файлы, то увидите множество макросов для контактов ввода/вывода, регистров и названий битов. Например, файл заголовков для устройства ATtiny45 содержит макрос для АЦП. Регистр ADCSR имеет восемь битов (в позициях от 0 (самый младший) до 7 (самый старший)). Они определены в файле заголовков и могут использоваться в программе.

Перечислимые типы данных

Перечисление — это тип данных, состоящий из набора предопределенных значений (которые называются интегральными константами). Помимо определения и группирования интегральных констант — перечисления полезны для переменных с ограниченным числом возможных значений.

Синтаксис определения перечислимого типа данных

```
enum Days{
   Sunday = 0,
   Monday,
   Tuesday,
   Wednesday,
   Thursday,
   Friday,
   Saturday
};
```

Здесь `Sunday` имеет значение 0, а последующие дни имеют значения в возрастающем порядке, то есть: `Monday = 1`, `Tuesday = 2` и т. д. Можно присвоить значения и явным образом:

```
enum cards{
   CLUBS = 1,
   DIAMONDS = 2,
   HEARTS = 4,
   SPADES = 8
};
```

Описатель *volatile*

Тип данных может быть специфицирован описателем типа. Один из таких описателей — `volatile`. При объявлении переменной компилятор оптимизирует ее соответственно ситуации. Цель этого описателя — устранить потенциальную оптимизацию. Например, компилятор иногда загружает переменную из памяти данных в регистр и выполняет с ней некие операции. После этого новое значение переменной записывается в память не сразу. Когда эту переменную использует другая процедура, она получает старое значение (что приводит к ошибке). Переменная квалифицируется описателем `volatile` следующим образом:

```
volatile int temperature;
```

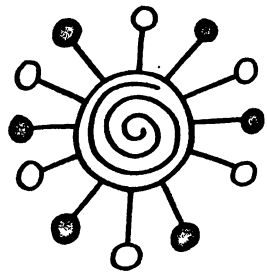
Лучше всего применять описатель `volatile` при совместном использовании переменных (вроде глобальной переменной, общей для многих функций и процедур обработки прерываний).

Описатель *const*

Еще один описатель языка C — `const`, который делает объекты постоянными (размещая их в памяти "только для чтения"). Такой объект должен инициализироваться во время объявления. Синтаксис таков:

```
const int my_marks;
```

В этом приложении мы кратко описали концепции языка C, необходимые для программирования микроконтроллеров AVR. Подробное изложение этой темы вы можете найти в любой книге по программированию на языке C.



Приложение 2

Проектирование и изготовление печатных плат

В *главе 1* мы разъясняли преимущества изготовления для наших проектов нестандартных печатных плат (по сравнению с платами общего назначения). Мы обсудили также несколько разных программ, имеющихся для проектирования печатных плат. Мы выбрали бесплатную версию EAGLE (Easily Applicable Graphical Layout Editor) компании CadSoft. Существует три стадии проектирования печатных плат: разработка схемы, компоновки и разводка. Компоновка и разводка часто подстраиваются под технологический процесс, при помощи которого плата будет изготавливаться. Мы изготавливали наши печатные платы на фрезерном станке Roland Modela MDX-20 и применяли опции программы EAGLE для этого оборудования. Настройки для разных проектов в основном одинаковы.

Версия EAGLE Light

Компания CadSoft предлагает три разных версии программы EAGLE: Professional, Standard и Light. Они отличаются друг от друга по максимальному числу страниц схемы и сигнальных уровней схемы, а также по максимальной площади платы для компоновки. Мы разрабатывали платы в версии Light. Ее функциональные возможности по сравнению с Professional несколько ограничены. Редактор Schematic Editor работает только с одним слоем, редактор Layout Editor имеет максимальный размер платы 4×3,2 дюйма (10×8 см), Autorouter разводит дорожки только в двух сигнальных уровнях (верхний и нижний). Таких возможностей вполне достаточно для печатных плат всех проектов этой книги. Печатную плату, спроектированную в версии Professional, в версии Light можно только смотреть, но не редактировать. Компания CadSoft разрешает бесплатно использовать версию Light для некоммерческих проектов. Вы можете скачать самую новую версию по ссылке: **http://cadsoft.de**. Компания CadSoft регулярно обновляет свое программное обеспечение. Когда мы начинали писать эту книги, существовала версия 5.6, а на данный момент доступна уже версия 5.10.

Программа EAGLE для Windows

Для проектирования плат в программе EAGLE вам понадобятся три окна: Control Panel, Schematic Editor и Layout Editor. Все они описаны далее.

Панель управления Control Panel

Панель управления (рис. П2.1) — это первое окно, которое появляется после запуска EAGLE.

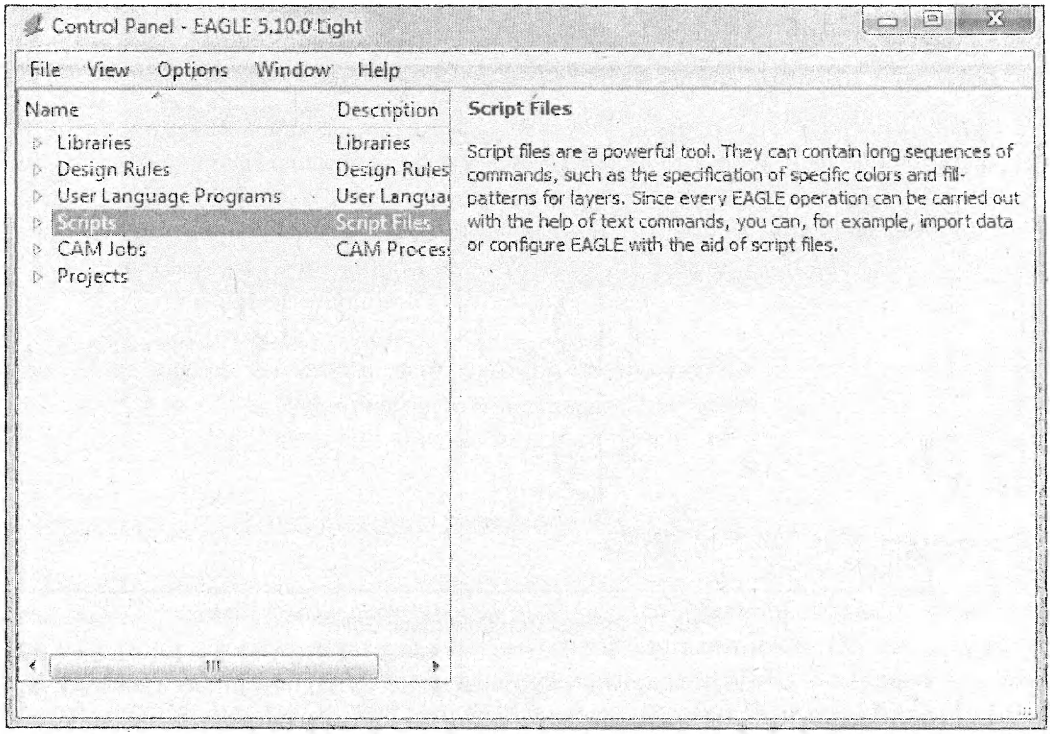


Рис. П2.1. Панель управления Control Panel

Панель управления — главное окно данной программы, в нем можно вводить команды управления. Чтобы начать в EAGLE новый проект, необходимо зайти в **File | New | Project**. Это создаст новый каталог в подкаталоге eagle. Каталог eagle — это каталог по умолчанию для хранения проектов; его местоположение указывается при инсталляции EAGLE. После этого вы можете создать в нем файлы схемы и платы (в меню **File | New | Schematic/Board**). При сохранении этих файлов в первый раз вам нужно будет указать каталог их местоположения (которым по умолчанию является созданный ранее каталог проекта). Однако вы можете хранить эти файлы и в другом месте. Вообще новая плата как таковая не создается. Сначала создается новая схема, а после ее проектирования плата создается по этой схеме.

Редактор схемы Schematic Editor

В этом окне проектируется схема, оно открывается при создании новой схемы или загрузке уже существующей (рис. П2.2).

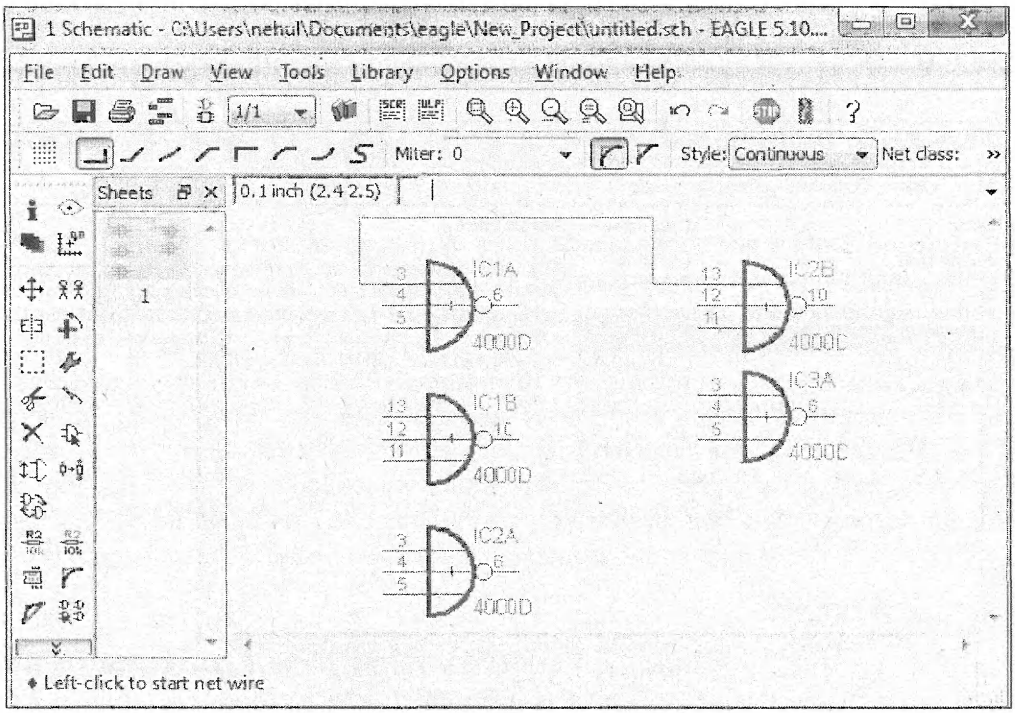


Рис. П2.2. Редактор Schematic Editor

Редактор компоновки Layout Editor

В этом окне (рис. П2.3) на плате размещают компоненты и разводят дорожки. Если оба окна (редакторов Schematic Editor и Layout Editor) открыты одновременно, то каждое изменение в схеме отражается и на плате. Это называется прямой и обратной корректировкой. Однако если окно компоновки не открыто во время редактирования схемы, то связь между ними теряется и программа не может отслеживать последующие изменения. Становится практически невозможно проектировать плату и единственный вариант — создать по схеме новую плату.

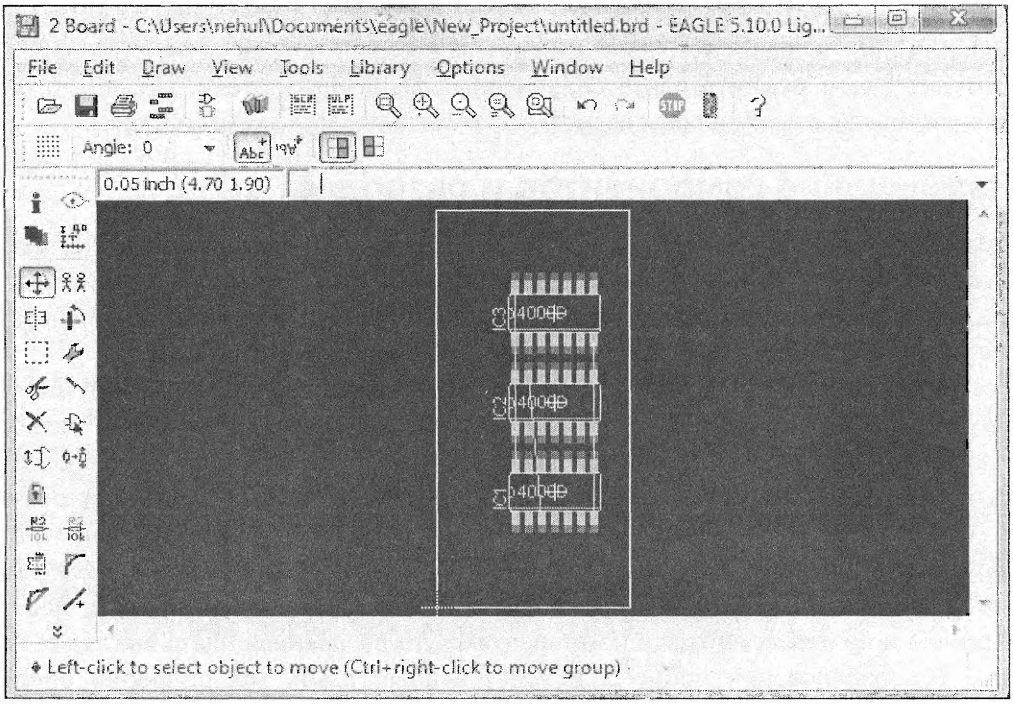


Рис. П2.3. Редактор компоновки Layout Editor

Руководство по программе EAGLE

В пакете программы EAGLE имеется руководство, в котором подробно описаны команды программы, ее настройки, добавление компонентов в схему, компоновка платы, разводка (ручная/автоматическая) и прочие сведения, необходимые для проектирования платы. Этот документ находится по адресу <каталог инсталляции>\EAGLE-5.10.0\doc\tutorial-en.pdf. В нем описываются все необходимые условия для проектирования обсуждавшихся в этой книге проектов. Если какая-то терминология EAGLE вам не понятна, она станет яснее после прочтения этого документа. Более подробное руководство можно найти по адресу: <каталог инсталляции>\ EAGLE-5.10.0\doc\manual-en.pdf. Мы рекомендуем вам прочитать это руководство перед тем, как переходить к следующему разделу.

Добавление новых библиотек

Для некоторых компонентов из этой книги мы самостоятельно создали пакеты и и добавили их в библиотеку. Для использования этих компонентов вам нужно присоединить библиотеку в панели управления. Файл библиотеки можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Вы можете сохранить этот файл в любой каталог, но лучше всего поместить его в <каталог инсталляции>\ EAGLE-5.10.0\lbr (это место по умолчанию для хранения

библиотек программы EAGLE). После сохранения этого файла зайдите в редактор Schematic Editor и щелкните меню **Library** (в верхней полосе меню). Затем выберите пункт **Use** и укажите ранее сохраненный файл библиотеки. После этого вы сможете добавлять компоненты из этой библиотеки в ваши схемы и платы.

Размещение компонентов и разводка

Перед компоновкой платы измените настройки DRC (в соответствии с вашим технологическим процессом). После этого разместите компоненты, учитывая все технологические ограничения. Если ваш производственный процесс имеет низкую точность, то компоненты придется размещать далеко друг от друга.

Разводка может выполняться либо вручную, либо автоматически. Если вы проектируете одностороннюю плату, то автоматическая разводка может не развести все дорожки, поэтому это придется делать вручную (либо в том же слое, либо в другом). Когда дорожки разводят в другом слое, их выполняют в виде перемычек. Соответствующие узлы соединяют либо луженными, либо изолированными медными проводами. Эти дорожки можно вообще не разводить, поскольку их все равно придется выполнять внешними проводниками. Если перемычки прямые, небольшой длины и не пересекаются с компонентами, то распаянная плата выглядит прилично. Технологический процесс, использованный нами, мы подробно опишем в следующем разделе.

Фрезерный станок Roland Modela MDX-20

Теперь мы переходим к изготовлению печатных плат при помощи фрезерного станка Roland Modela MDX-20. Мы опишем изготовление однослойной печатной платы, разведенной в нижнем слое программы EAGLE (обычные компоненты размещены на другой стороне разводки, а компоненты в корпусах SMD — на стороне разводки). Программное обеспечение было протестировано в среде Windows. Один из модулей (CAM.py) не может посылать команды в последовательный порт, поскольку он был написан под Linux. Однако его выходной файл можно отправить через последовательный порт при помощи другого программного обеспечения (как описано далее).

Шаг 1: изготовление схемы и компоновка платы в программе EAGLE

Для этого необходимо обладать базовыми знаниями о редакторах Schematic Editor и Layout Editor программы EAGLE. Поэтому мы рекомендуем вам прочитать упомянутое ранее руководство (если вы до сих пор этого не сделали). Первый шаг — проектирование схемы в соответствии с вашими требованиями. После того как схема будет готова и ошибки устранены (для проверки воспользуйтесь пунктом меню **Tools | Erc**), вы можете начинать компоновку печатной платы. В редакторе

Schematic Editor зайдите в меню **File** и выберите опцию **Switch To Board**. Откроется редактор Layout Editor. Прежде всего, убедитесь, что внешняя граница вашей платы начинается в координатах (0,0). Это нужно для того, чтобы избежать путаницы при указании смещений. Разместите компоненты любым желательным для вас образом.

Чтобы быть уверенным, что плату можно изготовить при помощи фрезерного станка Roland Modela, мы пользуемся набором опций проектирования, где указаны используемые слои (только нижний), минимальные расстояния от площадок и дорожек, диаметры отверстий, ширина медных дорожек и т. д. Откройте диалог Design Rules (при помощи пункта меню **Tools | Drc**) и загрузите файл modela.dru, имеющийся на сайте **www.avrgenius.com/tinyavr1** (кнопкой **Load**).

После загрузки опций проектирования выполните компоновку дорожек печатной платы (либо вручную, либо с помощью автоматического маршрутизатора программы EAGLE). Как уже упоминалось ранее, возможно, вам не удастся развести все дорожки в одном слое, поэтому придется задействовать перемычки или оставить их неразведенными. Когда компоновка будет готова, проверьте ее при помощи пункта меню **Tools | Drc | Check**. Если он выдаст ошибки, ликвидируйте их (изменив положение компонентов, дорожек и т. п.).

Шаг 2: создаем схему сверления

По компоновке платы мы можем подготовить данные для резки и сверления на станке Modela. На этом шаге мы создадим схему сверления отверстий в нашей плате.

Чтобы создать схему сигнальных дорожек и отверстий, нужно воспользоваться программой на языке EAGLE User Language Program (ULP), написанной Marc Boon (программа FabLabMill-n-Drill.ulp), она создаст схему для фрезерования дорожек и сверления отверстий. Однако полностью эти данные для фрезерования дорожек нам не понадобятся. Мы можем использовать уже разведенные в нижнем слое дорожки и площадки. Модифицированную версию программы ULP (настроенную под наши потребности) можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Перед запуском программы вы должны создать в редакторе Layout Editor два новых слоя:

- Layer no 111 Название: Roland_Milling;
- Layer no 112 Название: Roland_Drilling.

Для создания слоя щелкните мышью по значку слоя в панели команд, а затем нажмите кнопку **New**. После создания слоев запустите программу ULP (выбрав пункт **File | Run**) и откройте файл fablab-mill-n-drill.ulp. Вы должны увидеть окно, показанное на рис. П2.4.

Укажите диаметр инструмента фрезерного станка; для сверления отверстий рекомендуется 0,79375 мм, поскольку он соответствует сверлу диаметром в 1/32 дюйма. Затем укажите шину, которая не должна изолироваться от медного слоя. По умолчанию это общая шина GND. Если вы хотите изолировать все шины, убедитесь в том, что это поле пустое.

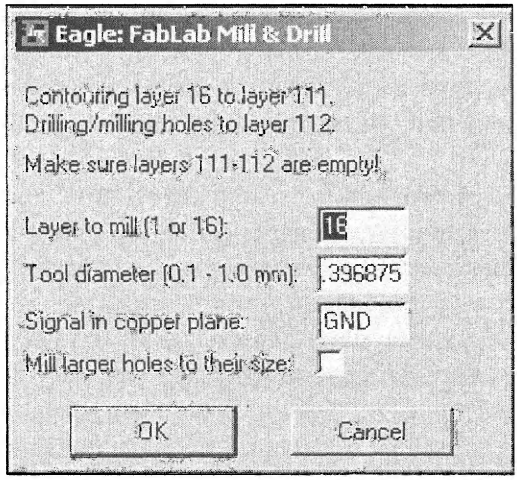


Рис. П2.4. Экран программы FabLab-Mill-n-Drill.ulp

И наконец, есть флажок, который необходимо установить, если вам нужны отверстия с диаметром, большим диаметра инструмента. Если вы не укажете этот флаг, то все отверстия будут просверлены с диаметром инструмента. Нажатие на кнопку **OK** создаст схему фрезерования в слое 111 и схему сверления в слое 112. Как уже объяснялось ранее — мы будем использовать только слой 112.

Шаг 3: создание файлов сверления и резки для управления станком Roland Modela Milling Machine

Для создания файлов CNC (Computer Numerical Control, числовое программное управление), которые будут управлять станком Roland Modela, мы воспользуемся процессором CAM Processor программы EAGLE. CAM Processor может создавать выходные файлы для многих плоттеров и печатающих устройств. Станок Roland Modela в их число не входит, однако мы можем описать его сами (указав все необходимые команды в файле eagle.def, который находится в подкаталоге bin инсталляционного каталога программы EAGLE). Добавив в этот файл несколько строк, мы опишем этот станок для процессора CAM Processor, и он будет узнавать станок как устройство вывода и сможет генерировать для него файлы числового программного управления.

Замените файл eagle.def (в подкаталоге bin каталога программы EAGLE) тем файлом eagle.def, который имеется по адресу: **www.avrgenius.com/tinyavr1**, и перезапустите EAGLE (сохранив вашу плату и схему).

Теперь опять откройте сохраненную компоновку и запустите CAM Processor (щелкнув мышью четвертую кнопку на главной панели инструментов). В процессоре загрузите задачу mill-n-drill.cam, которую можно скачать с нашего сайта по ссылке: **www.avrgenius.com/tinyavr1** при помощи пункта меню **File | Open | Job**. Вы увидите окно, показанное на рис. П2.5.

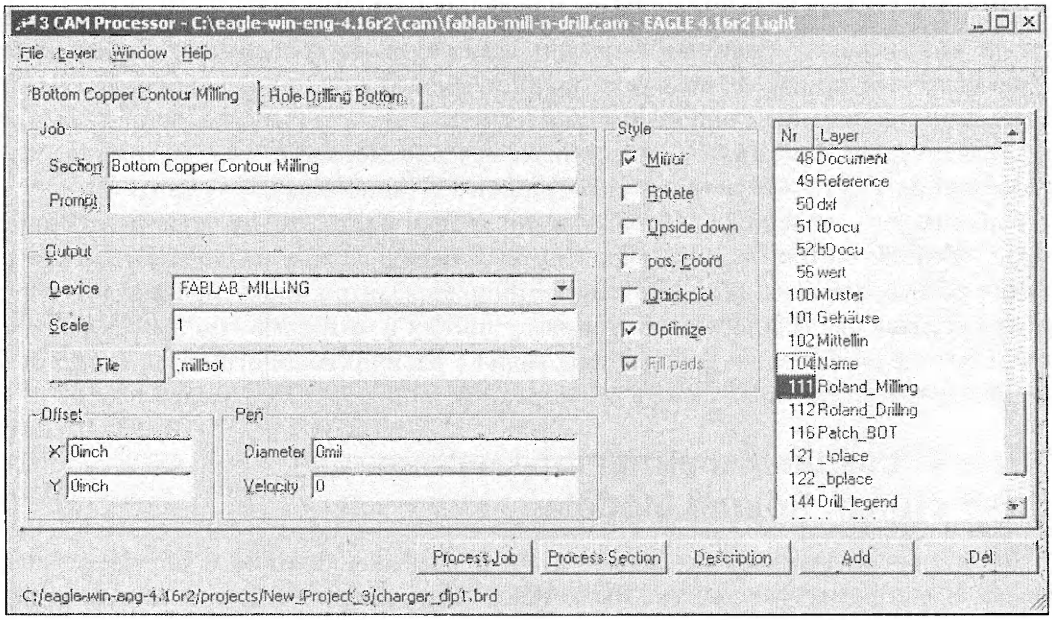


Рис. П2.5. Окно задачи Mill-n-Drill.cam

В этом окне содержится определение этой задачи процессора CAM. Здесь есть два раздела: **Bottom Copper Contour Milling** и **Hole Drilling Bottom**. В обоих разделах указано устройство (из модифицированного файла eagle.def), выходной файл, некоторые опции, а также используемые слои.

Все настройки уже предварительно указаны в задаче CAM и менять их не следует. Единственное исключение — смещение X и Y левого нижнего угла. Поскольку мы фрезеруем нижнюю сторону, то компоновка зеркальная и следует установить флажок **Mirror**. Однако зеркальное отражение компоновки означает, что все координаты как бы отражаются от оси Y, что преобразует положительные координаты X в отрицательные. Вам нужно будет сместить вашу компоновку (чтобы она оказалась в диапазоне системы координат фрезерного станка). Для этого можно установить флажок **pos. Coord**, но тогда смещение окажется слишком большим. Поэтому мы оставим этот флажок снятым и зададим смещение вручную.

Указываем смещения (это важно)

С указанием смещений связано много проблем. Во-первых, компоновка отражена зеркально (что дает отрицательные координаты X); и мы не хотим, чтобы наша плата фрезеровалась в координатах (0,0), поскольку это соответствует углу исходной платы (который может быть поврежден в этой точке). Следовательно, всегда очень полезно начинать плату в координатах (1,1) — в дюймах.

Теперь перейдем в редактор Layout Editor и снимем следующие координаты:

- Самая правая координата X (дает ширину платы);
- Самая нижняя координата Y.

Как мы уже упоминали ранее, если вы разместили вашу плату в (0,0), то самая нижняя координата Y будет 0. Мы взяли самую правую координату X (поскольку, когда плата отражена, ее правая сторона становится левой и наоборот). Теперь смещения вычисляются следующим образом:

- Смещение X = Самая правая координата X + 1 (в дюймах);
- Смещение Y = 1 – Самая нижняя координата Y (в дюймах).

Укажите эти смещения в обоих разделах. Убедитесь в том, что в разделе Bottom Copper Contour Milling выбран уровень 20-Dimension, а в разделе Hole Drilling Bottom выбран уровень 112 – Roland_Drilling. Нажмите кнопку **Process Job**. Программа создаст два файла с расширениями .millbot и .drillbot в том же самом каталоге, где содержатся ваши файлы *.brd. Файл с расширением drillbot предназначен для сверления отверстий, а файл с расширением millbot — для вырезки платы.

Шаг 4: создаем файлы фрезерования для станка Roland Modela

Для создания данных фрезерования мы берем информацию о дорожках, площадках и отверстиях из программы EAGLE. К сожалению, не существует одной задачи, которая (как на шаге 3) сможет сгенерировать данные фрезерования напрямую для управления станком. Поэтому мы делаем это в два этапа. Нужно зайти в CAM Processor и загрузить задачу gerb274x.cam, которая есть в программе EAGLE. Вы получите окно, показанное на рис. П2.6.

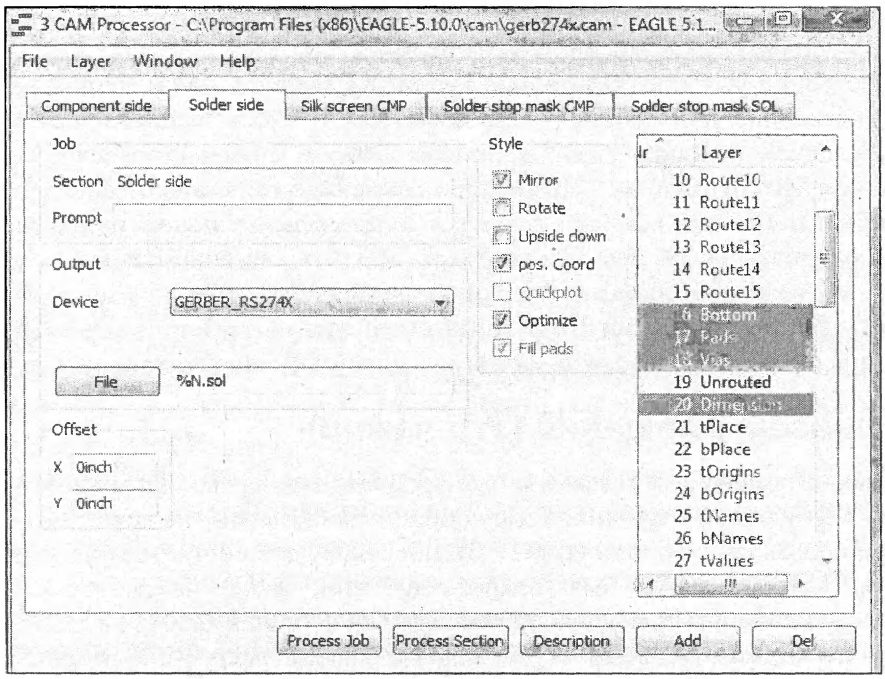


Рис. П2.6. Окно gerb274x.cam

В этом окне содержится определение задачи. Здесь пять разделов, но для плат с одним (нижним) слоем (которые производятся без идентификационного слоя) нужен только один раздел — **Solder Slide**. Убедитесь в том, что выделены все четыре слоя, указанных на рис. П2.6. Остальные настройки менять не следует (пусть остаются по умолчанию). Указывать смещения здесь не нужно, поскольку мы сделаем это на следующем шаге. Щелкнув вкладку **Process Tab**, вы получите выходные файлы, из которых нам нужен только один — с расширением sol.

Файл *.sol станок Modela читать не может, поэтому придется выполнить еще одно преобразование. Для этого мы воспользуемся программой под названием cam.py, которая написана на Python и следовательно является независимой от платформы. Однако она не может получать данные с последовательного порта компьютера под управлением Windows. Файл cam.py можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Файл cam.py (файл компьютеризированной обработки, написанный на языке программирования Python) сообщает станку Modela о том, что и как нужно вырезать; эта умная программа была разработана профессором Gershenfeld. Программа написана на языке Python и работает на всех платформах. Для выполнения этой программы в среде Windows на компьютере должен быть установлен Python (вы можете инсталлировать его с сайта **www.avrgenius.com/tinyavr1**). Для программы cam.py необходима библиотека Python Imaging Library (PIL). На сегодняшний день эта библиотека есть для версий Python ниже 2.7, поэтому следует выбрать версию Python ниже, чем 2.7. Библиотеку можно скачать по ссылке: **www.avrgenius.com/tinyavr1**.

Файл cam.py можно запустить в среде Windows из командной строки Python или из системной командной строки. Сохраните cam.py в том же каталоге, где инсталлирован Python. Откройте командную строку и перейдите в каталог Python. Затем введите команду python cam.py, и откроется окно программы (рис. П2.7).

В верхней части окна cam.py расположены некоторые пользовательские элементы управления (рис. П2.8).

Щелкните кнопку **Input File**, перейдите к вашему файлу с расширением sol и откройте его. Вы увидите вашу компоновку в левом нижнем углу окна. Вы можете увеличить ее размер, либо изменив число в окне **xy display size**, либо щелкнув по кнопке **Auto** (тогда заполнится все окно) (рис. П2.9).

Второй ряд кнопок и полей начинается с **x min** и **y min**. Это начало осей x и y на станине фрезерного станка. Поэтому точка с x = 0 и y = 0 находится в левом нижнем углу станины. Если вы начали фрезеровать с x = 0 и y = 0, то станок начнет работу в левом нижнем углу станины. Как уже упоминалось ранее, мы хотим начинать плату с точки (1,1) дюймов. Поэтому изменим значения **x min** и **y min** на 1 и 1. Следующий параметр — коэффициент масштабирования **xy scale factor**. Если плата должна быть такого же размера, как в программе EAGLE, то его следует установить в 1. Рядом с ним находятся **dx:** и **dy:** — размеры импортированного объекта. В нижней части окна cam.py есть и другие кнопки (рис. П2.10).

Чтобы указать тип станка и расширение файла, обратитесь к меню **Output Device** (под кнопкой **Output File**). Указав один из станков, вы автоматически выбираете целый набор параметров инструментов, которые относятся к нему. Если вы посмотрите в окно **Output File**, то увидите созданный файл (с правильным расширением).

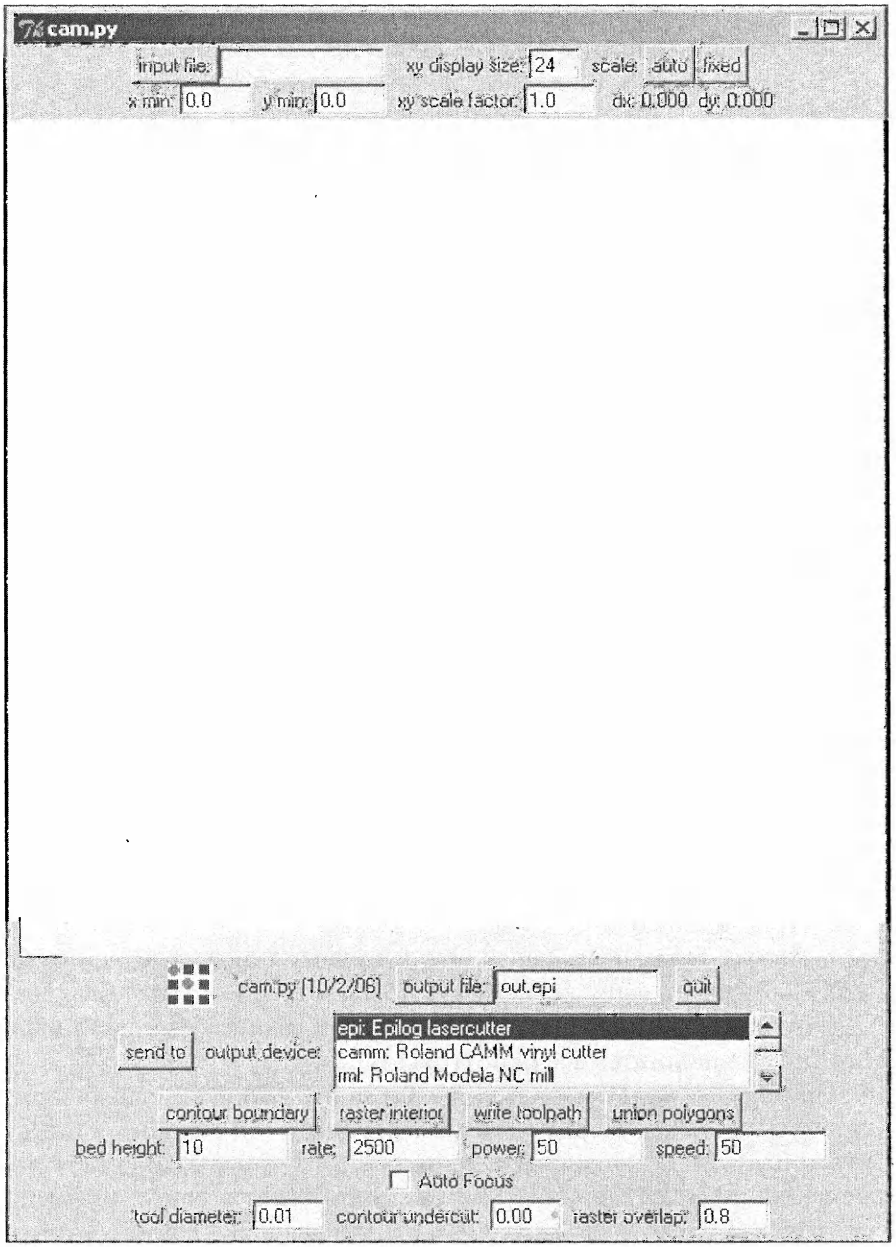


Рис. П2.7. Экран программы cam.py

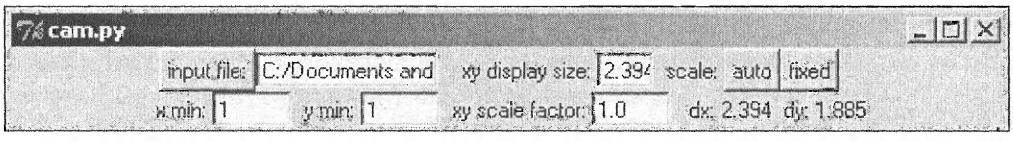


Рис. П2.8. Пользовательские элементы управления программы cam.py

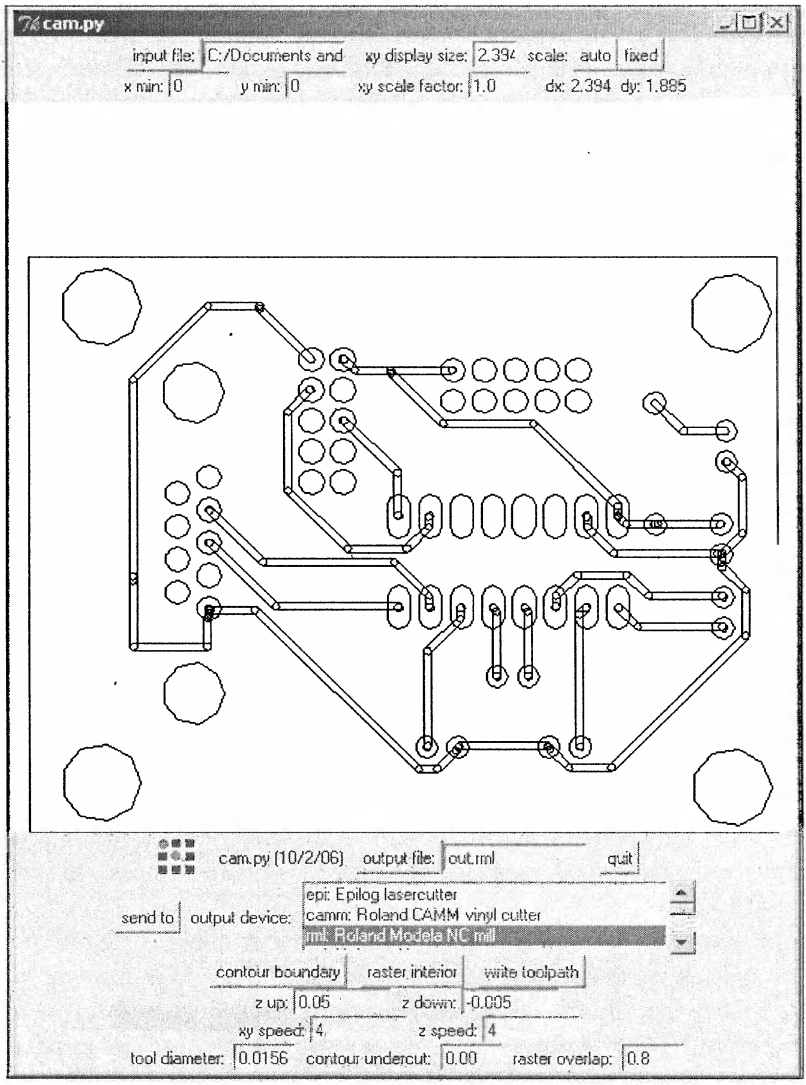


Рис. П2.9. Загрузка файла в программе cam.py

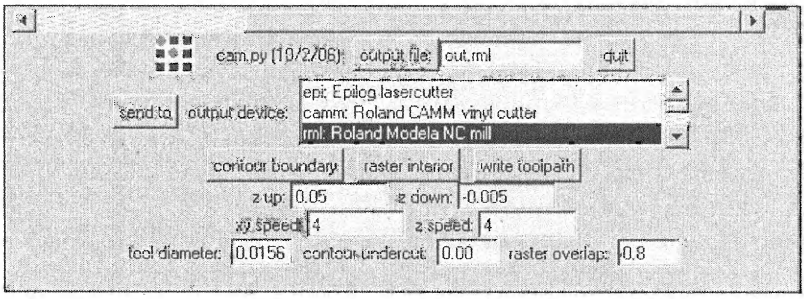


Рис. П2.10. Другие элементы управления программы cam.py

В окне **Output Device** выберите станок Roland Modela NC Mill, и в окне **Output File** появится расширение rml. Не забудьте нажать клавишу <Enter>, сообщив программе CAM о необходимости принять имя, набор инструментов и введенные вами до настоящего момента параметры. После этого вы должны опять установить **x min** и **y min** в 1, поскольку CAM не сохраняет эти числа при выборе нового станка.

В следующем ряду вы увидите кнопки **Contour Boundary**, **Raster Interior** и **Write Toolpath**. Если вы щелкнете кнопку **Contour Boundary**, то программа выделит красным все те области, которые мы хотим сохранить (эту кнопку нужно обязательно нажать для того, чтобы начать фрезерование платы). Если вы хотите удалить всю ненужную медь и оставить только те дорожки, которые необходимы для соединения компонентов, нажмите кнопку **Raster Interior**.

При помощи кнопки **Write Toolpath** вы можете сохранить файл со всеми выбранными вами параметрами. Это позволит вам в дальнейшем сразу получить тот же самый файл с точно такими же размерами, глубиной фрезерования, скоростью и т. д.

Параметр **z up** говорит о том, как высоко над материалом должна подниматься фреза при перемещении из одного места в другое (по умолчанию 0,05 дюйма). Параметр **z down** соответствует глубине прорезания материала (по умолчанию — 0,005 дюйма). Параметр **xy speed** — скорость движения фрезы по осям x и y. Параметр **z speed** — скорость движения фрезы по оси z. Значения по умолчанию для обоих этих параметров — 4. Параметр **Tool Diameter** — размер концевой фрезы. Значение по умолчанию — 1/64 дюйма (именно этой фрезой мы и пользовались для изготовления дорожек). Параметр **Contour Undercut** управляет положением центра фрезы относительно линий вашего проекта. Это способ для точного управления: резать по линии, внутри или снаружи линии. Если он вам не нужен, оставьте значение 0,0. Параметр **Raster Overlap** определяет точность срезания ненужной меди.

После того как все эти настройки сделаны, вы готовы отправить файл вашей платы на станок для ее изготовления. Кнопка **Send To** в Windows не работает, поскольку программа cam.py была разработана для Linux. Можно сгенерировать выходной файл *.rml и отправить его на станок при помощи программы Bray Terminal, как описано далее.

Шаг 5: фрезерование, сверление и вырезка печатной платы

Возьмите две заготовки платы класса FR2 (одну черновую и одну рабочую) и переверните одну из них (черновую) так, чтобы медная фольга была снизу. Приклейте к ней с обратной стороны двустороннюю клейкую ленту (рис. П2.11).

То место, где вы будете фрезеровать, должно быть надежно прикреплено к станине станка, поэтому не экономьте ленту.

На станине Modela есть сантиметровая сетка. Установите заготовку на два дюйма выше и на два дюйма правее, чтобы попасть под смещение по умолчанию в (1,1) дюйма. Обычно смещение в 1 дюйм слишком велико и если мы расположим нашу плату в двух сантиметрах, то у нас получится итоговое смещение в 2,54 см (1 дюйм),

что достаточно хорошо. После размещения на станине ненужного куска платы, прикрепите на него точно так же рабочий кусок. Теперь тряпкой (или рукой) прочно прижмите рабочую плату к черновой (рис. П2.12).

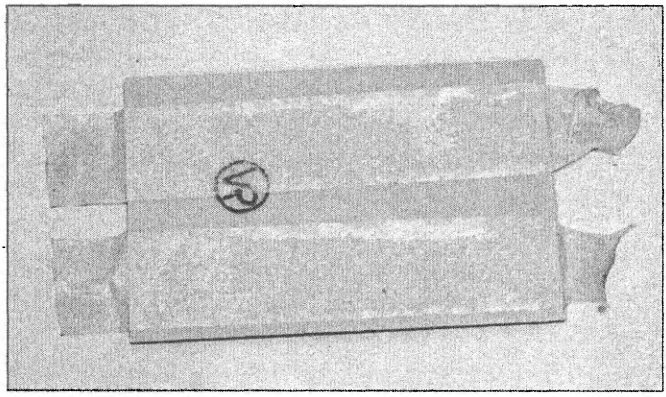


Рис. П2.11. Плата FR2 с двусторонней клейкой лентой

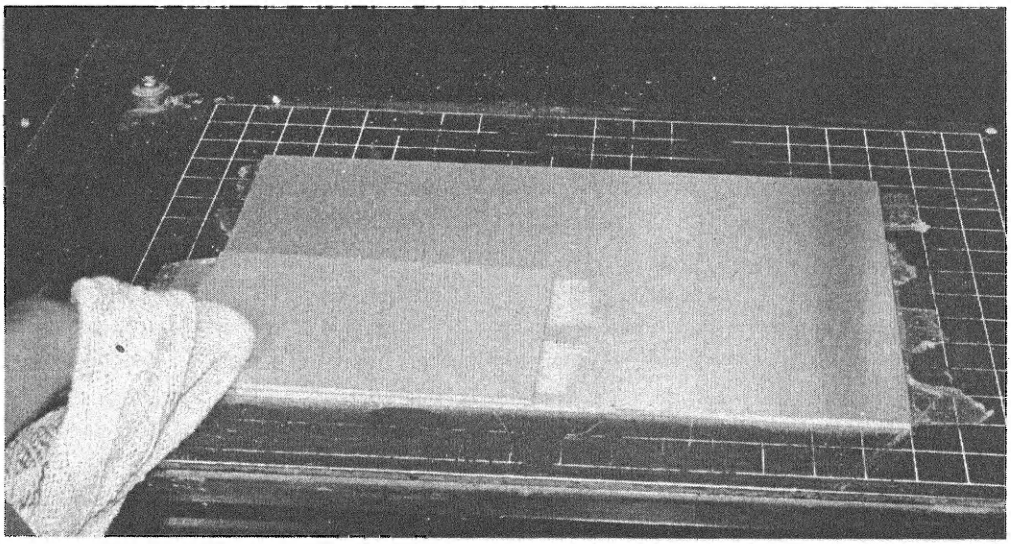


Рис. П2.12. Размещение заготовки на станине станка

Не прижимайте заготовку пальцами, потому что жир с вашей кожи может ухудшить проводимость медных дорожек. Черновая плата предохраняет станину Modela от повреждений при сверлении отверстий.

Затем нужно установить головку в правильное положение. Головка находится в задней части станка. Чтобы перевести ее в активный режим, нажмите кнопку **View** в панели управления **Control Panel** (рис. П2.13).

После этого головка станка переместится в начальную точку x=0, y=0 (рис. П2.14).

Рис. П2.13. Кнопка View на панели управления станка

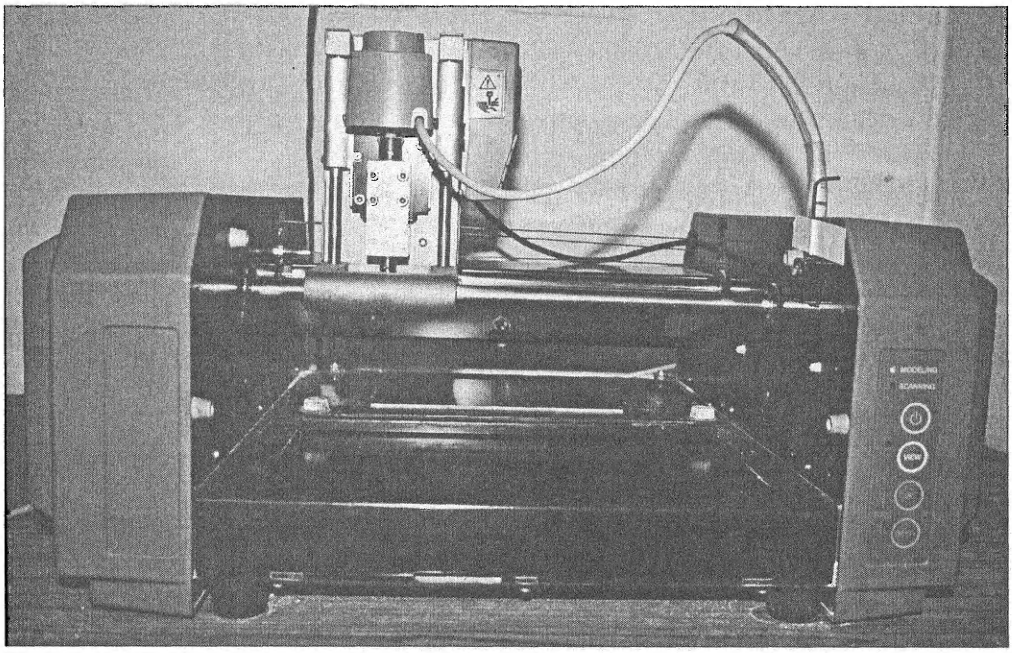


Рис. П2.14. Головка станка в исходной позиции (x=0, y=0)

Теперь вы хотите переместить фрезу в позицию x=1, y=1. Для этого вам нужно отправить на станок команды (через последовательный порт). Для этого мы используем программу Bray Terminal, которую можно скачать по ссылке: **www.avrgenius.com/tinyavr1**. Запустите программу и настройте ее опции (рис. П2.15).

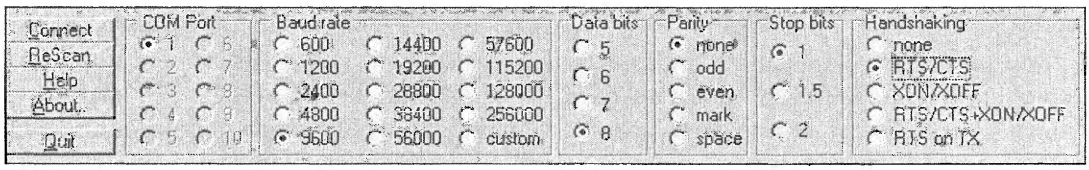


Рис. П2.15. Окно программы Bray Terminal

Убедитесь в том, что имя последовательного порта com1. Порт может быть на вашем компьютере другим, тогда измените его. После этого нажмите кнопку **Connect**. Теперь вы хотите переместить головку в позицию (1,1). Для этого на сайте **www.avrgenius.com/tinyavr1** есть файл move%d%d.txt. Откройте его, и вы увидите следующее содержимое:

PA;PA;!PZ0,400;VS10;!VZ10;!MC0;PU%d,%d;!MC0;

В этом тексте замените %d,%d координатами x и y той позиции, куда вы хотите переместить головку, но будьте внимательны, вы должны указывать координаты в миллидюймах, а не в дюймах. Поэтому для перемещения головки в (1,1) сделайте такие изменения:

PA;PA;!PZ0,400;VS10;!VZ10;!MC0;PU1000,1000;!MC0;

Скопируйте этот текст в раздел **Transmit** окна программы Bray Terminal (которое пока серое) и нажмите <Enter>. Вы увидите, как головка движется в позицию (1,1). Вернемся к станку. Теперь нам нужно установить в позицию инструмент. Для фрезерования используйте сверло в 1/16 дюйма, а для сверления и резки — сверло в 1/32 дюйма. Инструмент удерживается патроном с двумя винтами. Возьмите маленький торцевой ключ (рис. П2.16), который входит в комплект к станку, и ослабьте эти винты.

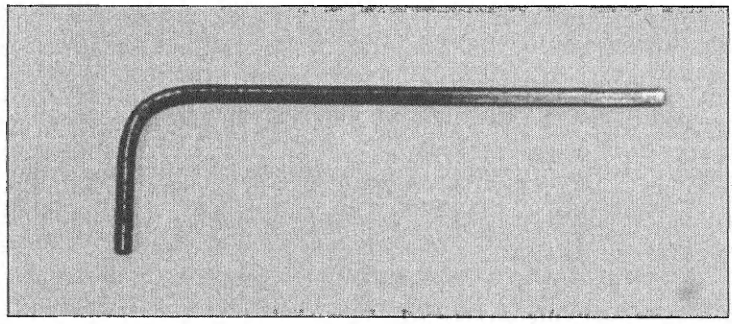


Рис. П2.16. Торцевой ключ

Крепко держите сверло, чтобы не уронить его и не сломать хрупкий кончик, и вставьте его как можно выше в патрон. Слегка затяните винты, чтобы зафиксировать инструмент, но не слишком сильно. Это делается просто для того, чтобы убрать инструмент на время позиционирования каретки.

Следующий шаг — переместить всю каретку в самую нижнюю точку, когда металл каретки соприкоснется с металлом основы. Сделайте это путем нажатия и удерживания кнопки **Down** на панели управления. Если вы хотите несколько сместиться от этого положения, нажмите и удерживайте кнопку **Up** до тех пор, пока каретка не поднимется немного вверх. Сейчас вы указываете станку, где находится точка Z=0. Если вы не сместитесь относительно самого нижнего положения, то инструмент не будет сверлить материал. Теперь опять придержите инструмент пальцем, ослабьте винты и осторожно поместите инструмент на поверхность заготовки (рис. П2.17 и П2.18). После этого затяните винты вручную, чтобы инструмент прочно удерживался на своем месте.

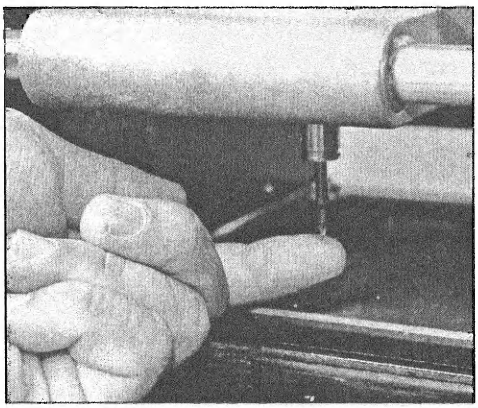


Рис. П2.17. Точная подгонка инструмента

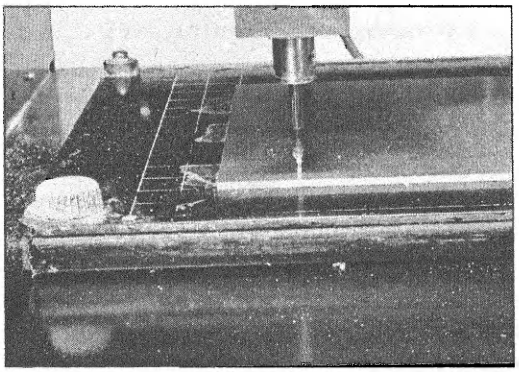


Рис. П2.18. Окончательное положение инструмента

Теперь все готово к изготовлению вашей платы. В качестве меры предосторожности поднимите инструмент над поверхностью в том месте, где вы собираетесь начать резать, т. е. в точке (1,1).

Для этого вернитесь в Bray Terminal и введите следующий текст:

PA;PA;!PZ0,400;VS10;!VZ10;!MC0;PU1000,1000;!MC0;

Теперь можно отправить rml-файл для фрезерования дорожек. Выберите опцию **Send File** и отправьте соответствующий файл. После завершения фрезерования снимите инструмент 1/16 дюйма и установите инструмент 1/32 дюйма. Повторите описанную ранее процедуру и отправьте в станок файлы *.drillbot и *.millbot. Плата готова к пайке и тестированию.

Приложение 3

Лупа со светодиодной подсветкой

В *главе 1* мы перечислили инструменты, которые потребуются для изготовления устройств этой книги. Один из таких инструментов — лупа, показанная на рис. П3.1.

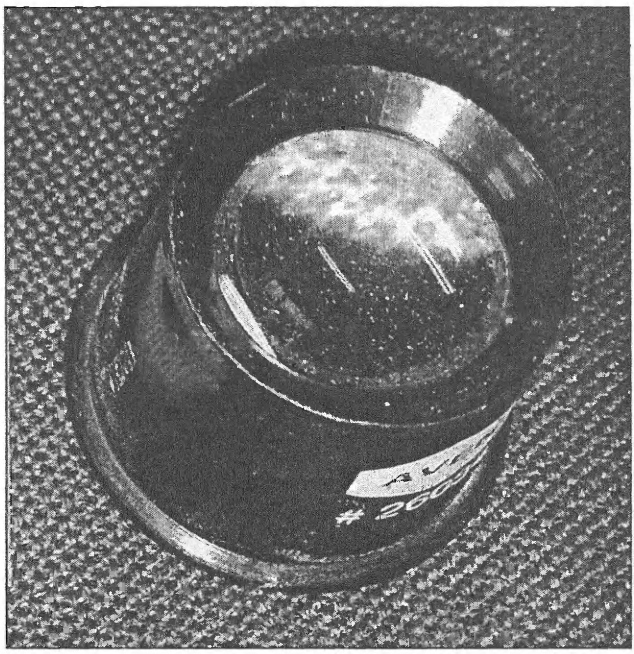


Рис. П3.1. Внешний вид лупы

Обычно лупу используют часовщики, а мы считаем ее очень полезной для осмотра компонентов, печатных плат и т. д. На рис. П3.1 показана лупа с увеличением в 10 раз. При разглядывании мелких деталей часто не хватает освещения. В этом приложении мы покажем, как усовершенствовать лупу, добавив белые светодиоды, которые будут освещать поле зрения. Мы опишем три варианта лупы с подсветкой. В первом используется батарейка 9 В и уровень освещения фиксированный; во втором установлена батарейка 1,5 В размером AAA и свет тоже

не регулируется; третья конструкция выполнена на базе микроконтроллера, который позволяет пользователю настроить освещение под свои потребности. Для работы третьего устройства требуется батарея 9 В.

Сначала нужно снабдить лупу восемью белыми светодиодами с последовательно включенными резисторами (рис. П3.2). Диаметр светодиодов 3 мм. Все они соединены параллельно.

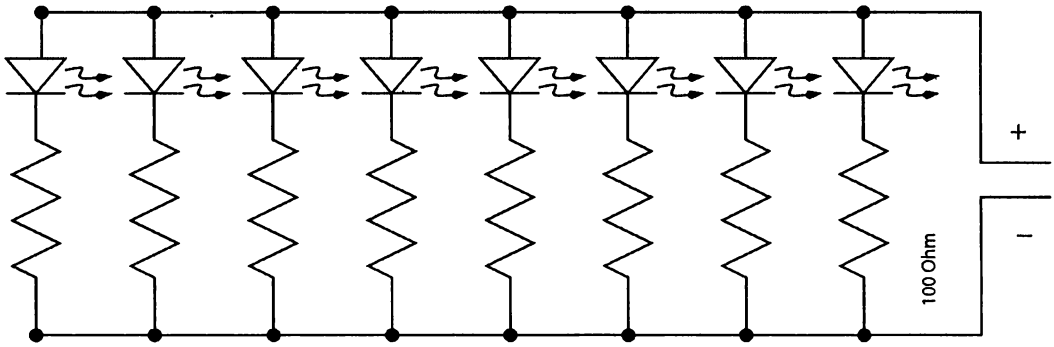


Рис. П3.2. Схема соединения светодиодов и резисторов

Для монтажа светодиодов и резисторов потребуется кусок печатной платы общего назначения (рис. П3.3).

Из заготовки вырезано кольцо (рис. П3.4) так, чтобы в него плотно входил тот конец лупы, где находится линза.

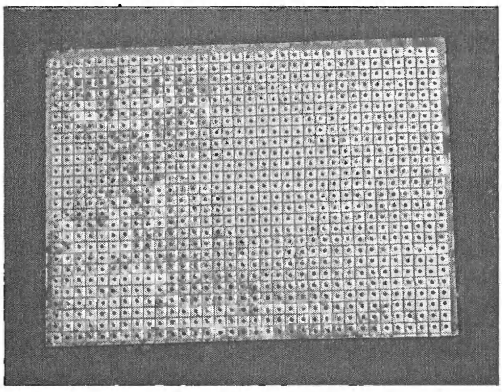


Рис. П3.3. Заготовка печатной платы

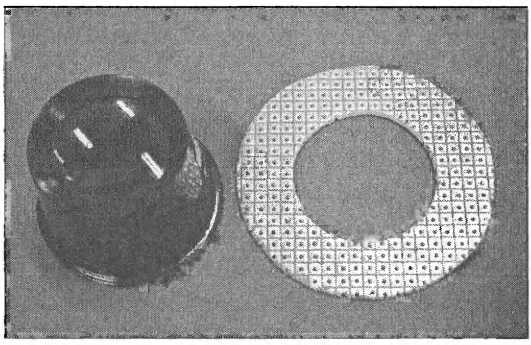


Рис. П3.4. Печатная плата в виде кольца

После вырезки печатной платы светодиоды равномерно распаиваются вокруг отверстия (рис. П3.5). Обратите внимание, что светодиоды вставляются со стороны пайки (со стороны медной фольги) печатной платы. Обычно компоненты вставляют с противоположной стороны и распаивают на стороне пайки, но здесь мы вставили светодиоды со стороны пайки. Для пайки светодиодов оставьте над поверхно-

стью печатной платы выводы длиной примерно 1 см. Припаяв светодиоды, их нужно немного подогнуть к центру платы.

После пайки светодиодов со стороны компонентов вставляют резисторы, которые припаивают к светодиодам (рис. П3.6).

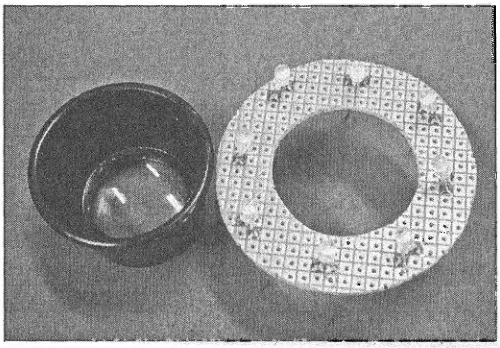


Рис. П3.5. Монтаж светодиодов

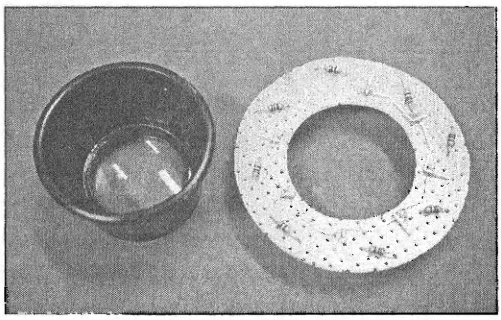


Рис. П3.6. Монтаж резисторов

После припаивания резисторов к светодиодам свободные выводы резисторов соединяют вместе. Также соединяют вместе и аноды светодиодов (рис. П3.7).

Следующий шаг — нужно припаять два провода (один — к анодам светодиодов, второй — к точке соединения всех резисторов). Светодиоды залиты термоклеем, который защищает их от физических повреждений (рис. П3.8).

Теперь наша плата готова и пришло время закрепить лупу в ее центре (рис. П3.9 и П3.10).

После припаивания к плате проводов, их подключают к батарейке 9 В через выключатель, чтобы можно было включать/выключать светодиоды по мере надобности. На рис. П3.11 и П3.12 показана работа подсветки.

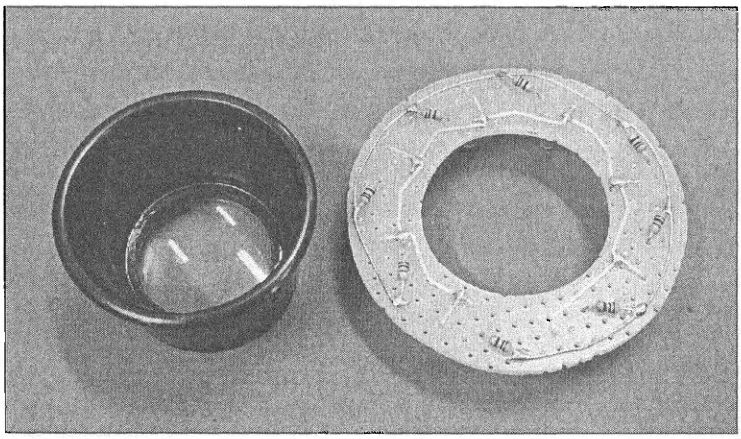


Рис. П3.7. Соединение выводов резисторов

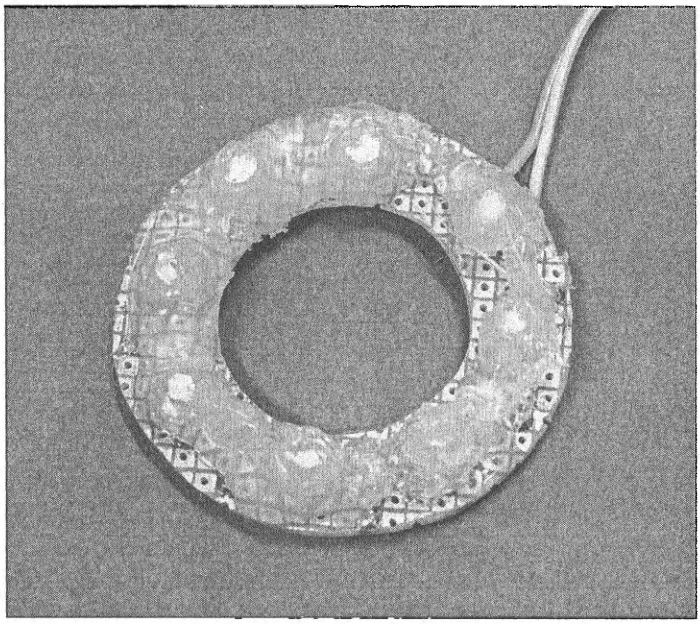


Рис. П3.8. Заливка светодиодов термоклеем

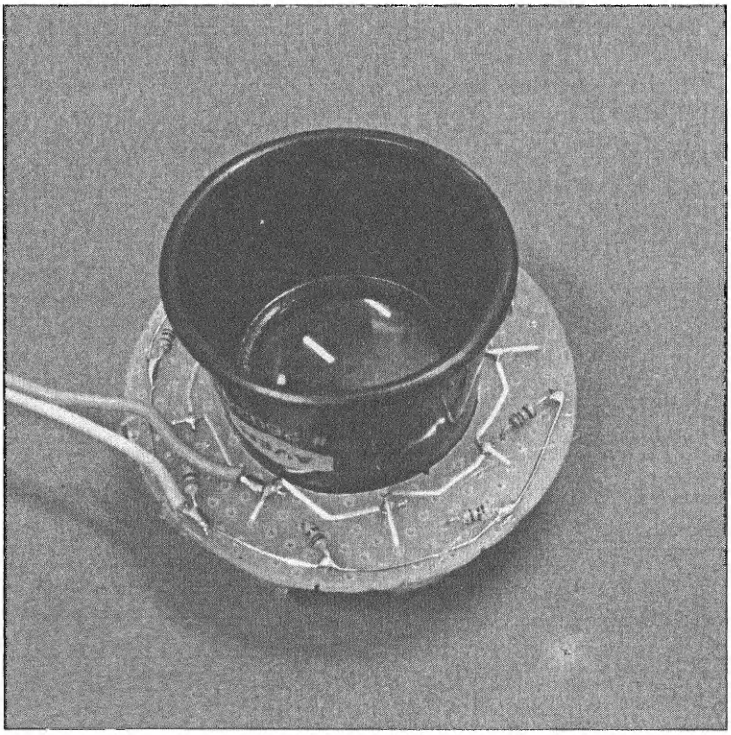


Рис. П3.9. Крепление лупы

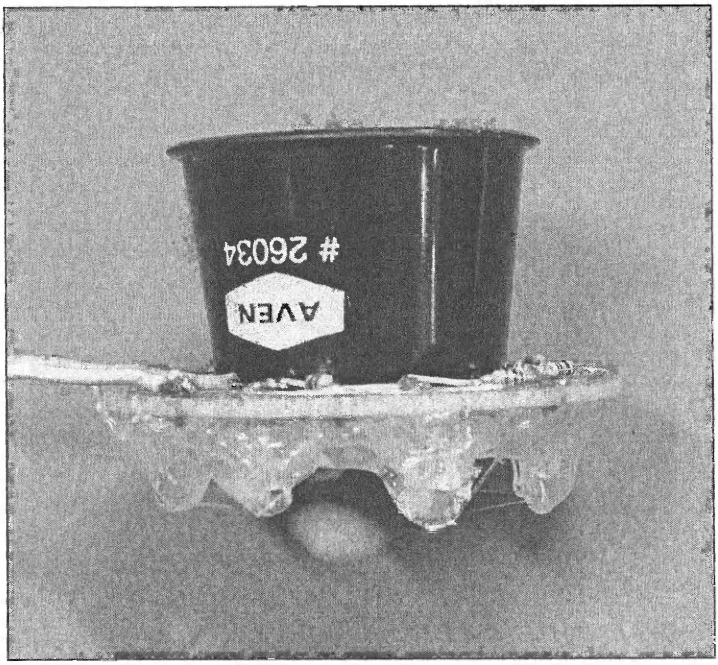


Рис. П3.10. Лупа, скрепленная с платой

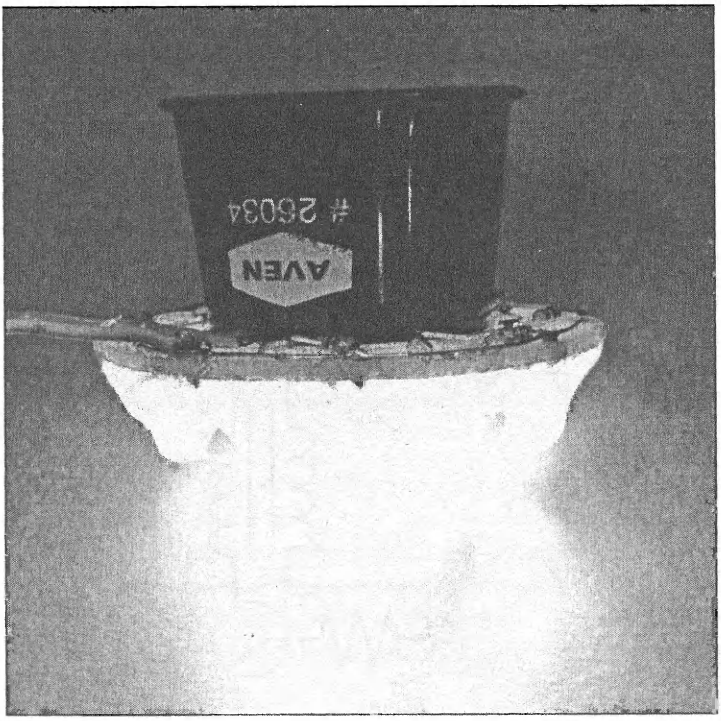


Рис. П3.11. Работа подсветки

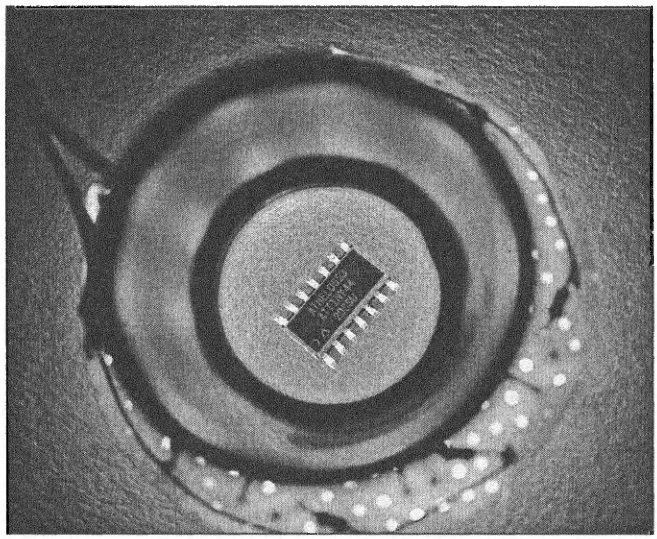


Рис. П3.12. Осмотр компонента через лупу с подсветкой

Второй вариант лупы с подсветкой

В предыдущем устройстве светодиоды питались от батарейки 9 В. Однако такие батареи дорогие и слишком большие. Лучше взять батарейку на 1,5 В, но для работы белых светодиодов нужно напряжение не менее 3,5 В. Поэтому потребуется электронная цепь, которая будет повышать напряжение с 1,5 В до, например 4 В. Это легко сделать при помощи повышающего преобразователя, однако стоимость такого конвертора высока. Несложную цепь для повышения напряжения можно собрать на простом транзисторном генераторе (так называемом релаксационном генераторе), показанном на рис. П3.13.

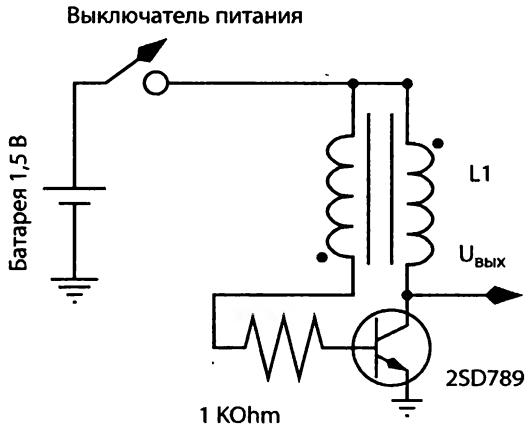


Рис. П3.13. Схема релаксационного генератора

В генераторе всего три компонента: *n-p-n*-транзистор, специальная катушка индуктивности и резистор. Выходной сигнал такого генератора изображен на рис. П3.14 — это однополярные импульсы с амплитудой более 10 В. Такой сигнал очень хорошо подходит для питания белых светодиодов нашей лупы.

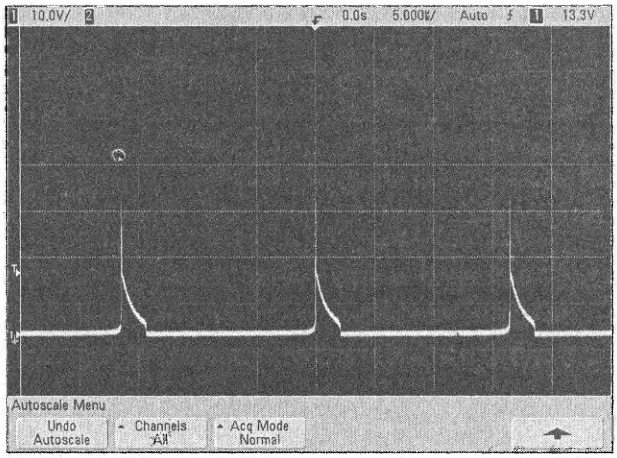


Рис. П3.14. Сигнал на выходе релаксационного генератора

Самый важный компонент — специальная катушка индуктивности. Она состоит из двух секций, выполненных эмалированным медным проводом (размера 36), намотанным на подходящий каркас (ферритовый стержень, тороид и т. п.). Точки возле выводов катушки (рис. П3.13) показывают начало обмотки. Изготовление катушки начните с отматывания достаточного (для 10–20 витков) количества провода. Сделайте два таких куска одинаковой длины (рис. П3.15).

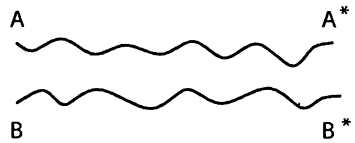


Рис. П3.15. Два провода для намотки катушки

Теперь сложите оба провода вместе. Убедитесь в том, что с концов проводов (A и A*, B и B*) снята изоляция (рис. П3.16).

Намотайте сложенные вместе отрезки проводов на выбранный вами сердечник. На рис. П3.17 показаны сложенные вдвое провода, намотанные на каркас.

Спаяйте вместе противоположные концы проводов (A и B* или B и A*). Таким образом, у вас остается три конца проводов, которые служат выводами катушки (рис. П3.18).

Собранная и распаянная схема показана на рис. П3.19. Разъем слева подключается к узлу со светодиодами и резисторами.

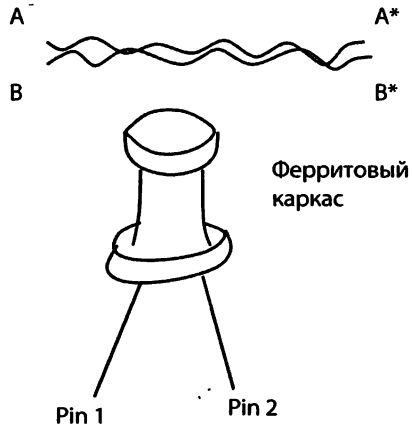


Рис. П3.16. Провода сложены вместе и готовы к намотке на каркас

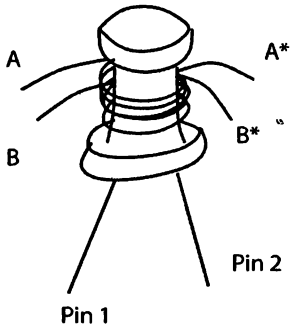


Рис. П3.17. Провода, намотанные на каркас

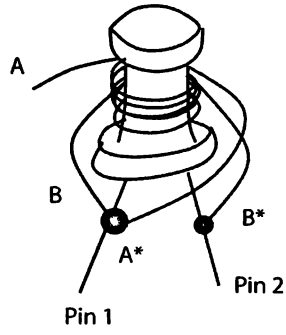


Рис. П3.18. Выводы катушки

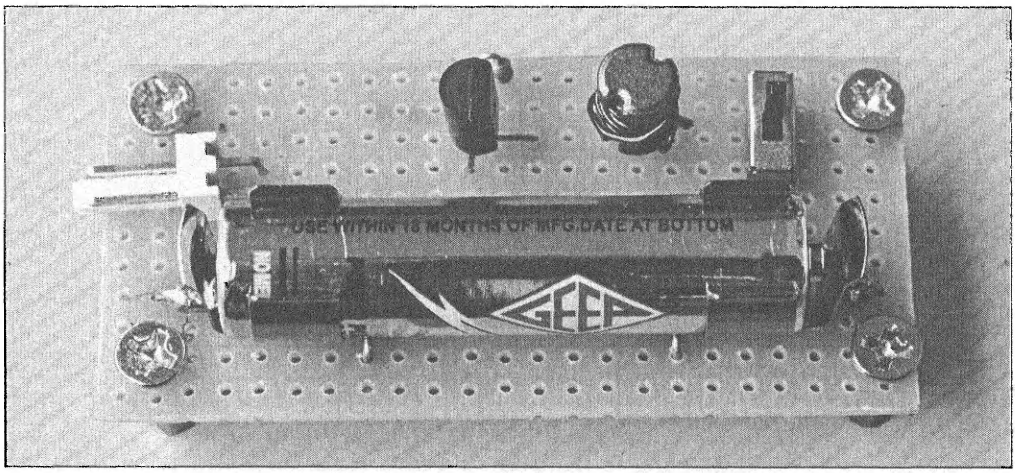


Рис. П3.19. Собранная схема генератора

Лупа с регулируемой светодиодной подсветкой

В предыдущих двух устройствах интенсивность подсветки не регулировалась. Однако иногда свет может оказаться слишком сильным, а в другом случае вам может потребоваться дополнительная подсветка. Новый вариант лупы соответствует таким запросам. В устройстве используется восьмиконтактный микроконтроллер tinyAVR. Здесь вполне подойдет Tiny13, хотя можно взять также Tiny24 или Tiny25. Принципиальная схема устройства изображена на рис. П3.20.

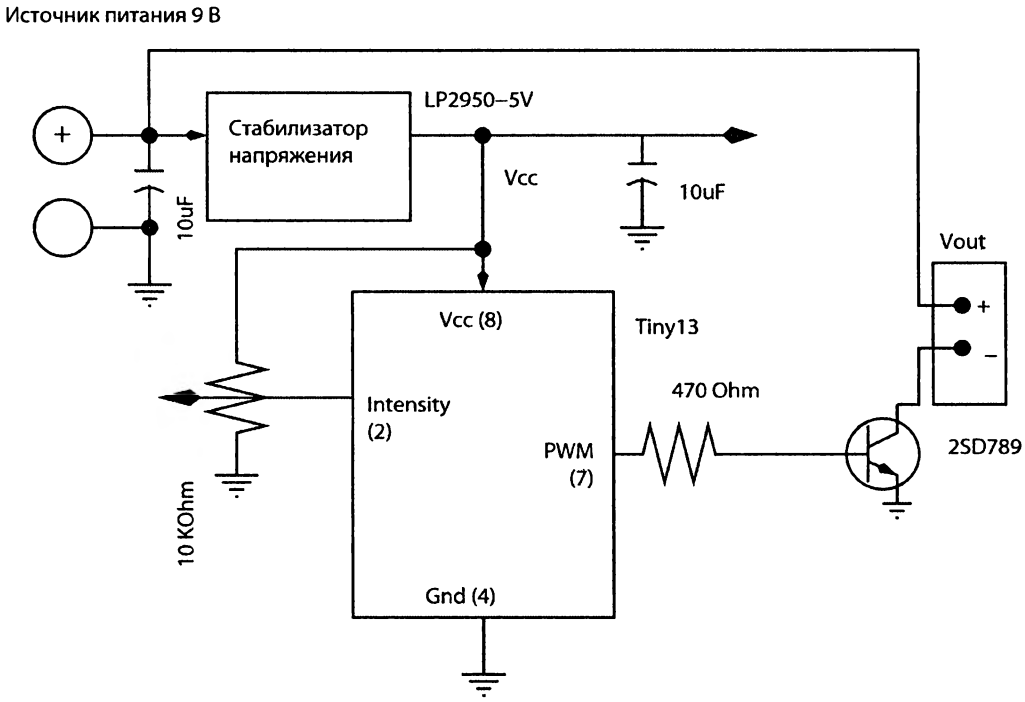


Рис. П3.20. Принципиальная схема лупы с регулируемой подсветкой

Схема питается от батарейки 9 В, а напряжение для микроконтроллера в ней вырабатывает стабилизатор LP2950-5V. Интенсивность свечения светодиодов лупы регулируется потенциометром 10 кОм. Положение движка потенциометра считывается АЦП микроконтроллера (контакт 2) и преобразуется в соответствующий ШИМ-сигнал на выходе микроконтроллера (контакт 7). Этот контакт управляет *n-p-n*-транзистором. Коллектор транзистора соединен с отрицательным выводом схемы, показанной на рис. П3.2. Положительный вывод схемы на рис. П3.2 подключен к батарее 9 В. ШИМ-сигнал меняет интенсивность от 0 до 100% (в зависимости от положения движка потенциометра).

На рис. П3.21 и П3.22 показан внешний вид готового устройства. Программное обеспечение для контроллера идентично использованному в проекте 3 *главы 2*.

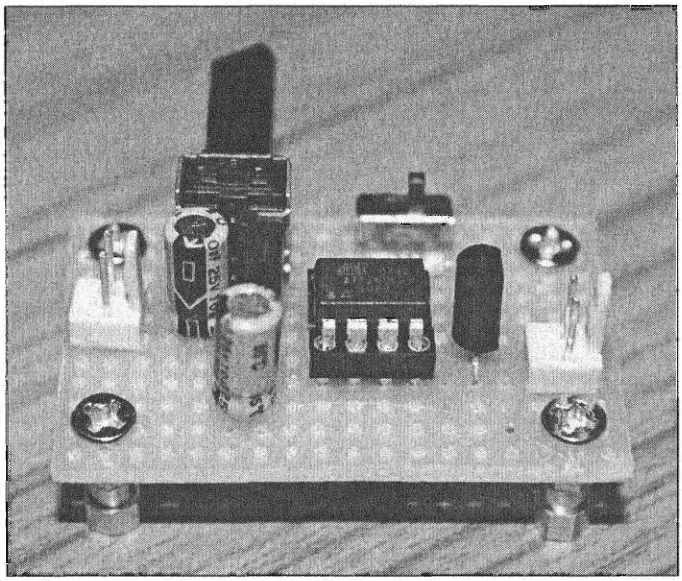


Рис. П3.21. Устройство в сборе (внешний вид компонентов)

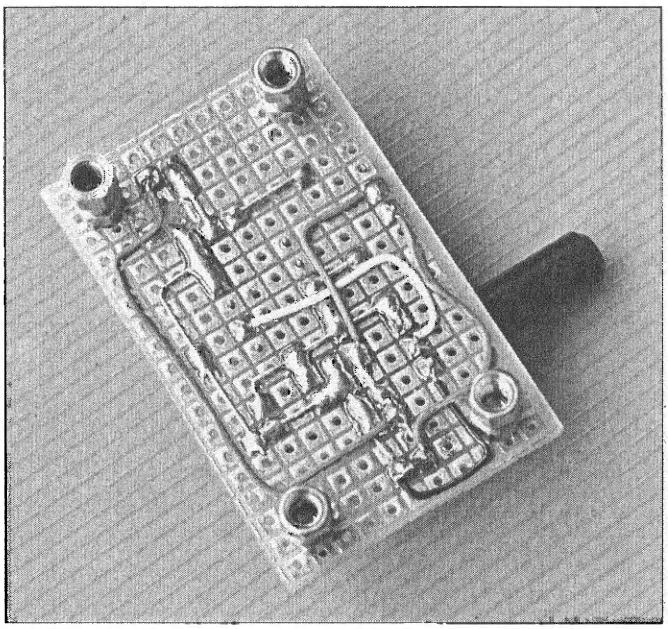


Рис. П3.22. Внешний вид обратной стороны печатной платы

Предметный указатель