Министерство образования и науки Российской Федерации
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

ИНСТИТУТ ЭНЕРГЕТИКИ И ЭЛЕКТРОТЕХНИКИ
(институт)
Промышленная электроника
(кафедра)
11.03.04 Электроника и наноэлектроника
(код и наименование направления подготовки, специальности)
Промышленная электроника
(направленность (профиль)

БАКАЛАВРСКАЯ РАБОТА

на тему **МИКРОПРОЦЕССОРНОЕ УСТРОЙСТВО ЗАПИСИ ТЕЛЕФОННЫХ РАЗГОВОРОВ**

Студент(ка) В.В. Бондаренко _____
(И.О. Фамилия) (личная подпись)

Руководитель к.т.н., доцент В.А. Медведев _____
(И.О. Фамилия) (личная подпись)

Допустить к защите

Заведующий кафедрой к.т.н., доцент А.А. Шевцов _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)

«_____»_____20_____г.

Тольятти 2017

Аннотация

УДК 621.314.572
ББК 32 852

Бакалаврская работа Бондаренко Вадима Вячелавовича по теме «Микропроцессорное устройство записи телефонных разговоров». Руководитель: Медведев Валерий Александрович. Защищена в Тольяттинском государственном университете в 2017 году.

Пояснительная записка: 96с., 3 разд., 49 рис., 21 табл., прил. 20 стр.

Графическая часть - 6 листов формата А1.

Ключевые слова: служба технической поддержки, система записи телефонных разговоров, микроконтроллер, принципиальная схема, печатная плата, программирование микроконтроллера.

Бакалаврская работа посвящена разработке микропроцессорному устройству записи телефонных разговоров. В ходе проектирования устройства разработана структурная схема, дано обоснование выбора элементной базы, разработаны принципиальная схема и печатная плата модуля. Выбраны средства программирования и отладки микроконтроллера.

Содержание

Введение

Для ряда компаний по определенным причинам нецелесообразно держать в штате собственных специалистов по комплексному техническому обслуживанию парка компьютеров и программного обеспечения. Но в тоже время им необходим высокий сервис и качественный ремонт компьютеров и оргтехники, своевременное обновление и настройка ПО.

Выходом из данной ситуации может служить сотрудничество со специализированными компаниями, оказывающими комплексную техническую поддержку.

Техническая поддержка – сервисная структура, решающая вопросы пользователей с компьютерами (как аппаратным, так и программным обеспечением) и оргтехникой. Техническая поддержка – неотъемлемая часть успешного бизнеса для любой современной организации.

Зачастую, клиентам, обращающимся в службу технической поддержки для решения возникших проблем с программным продуктом, достаточно консультации по телефону.

Оператор технической поддержки может проинструктировать клиента по вопросам работы, настройки и устранения мелких неполадок при работе с программными продуктами. При необходимости, оператор может подключиться удаленно и самостоятельно исправить возникшие неполадки с программным обеспечением.

Во время консультации по телефону или работе удаленно между клиентом и оператором службы технической поддержки может возникнуть недопонимание. Клиент может некорректно описать характер неисправности или оператор не в полном объеме выполнит свои обязанности по консультации или настройке программного обеспечения.

Для разрешения подобного рода спорных ситуаций выходом может послужить запись телефонных разговоров оператора службы технической поддержки с клиентом.

С целью повышения качества обслуживания пользователей, обращающихся в службу технической поддержки, а также повышения трудовой дисциплины сотрудников было решено разработать устройство, способное записывать на электронный носитель телефонные разговоры с клиентами.

Цель бакалаврской работы: разработать микропроцессорное устройство записи телефонных разговоров, способное взаимодействовать с ПК или работать автономно.

Задача бакалаврской работы: разработка устройства записи телефонных разговоров на основе микроконтроллера ATMEL.

1 Аналитическая часть. Анализ основных сфер применения и функционирования устройства записи телефонных разговоров

1.1 Анализ основных функций службы технической поддержки

В техническую поддержку входят следующие виды услуг:

а) консультации специалистов поддержки по телефону по вопросам эксплуатации, дополнительных настроек при изменениях условий эксплуатации.

б) прием запросов пользователя по электронной почте;

в) конфигурирование программных продуктов, диагностика или корректировка ошибок, выполняемая специалистами технической поддержки с использованием удаленного доступа;

г) исправление ошибок в программном обеспечении;

д) поставка пользователю корректирующих модулей, программ и файлов обновления, снимков комплекта ПО для устранения выявленных неисправностей в работе системы;

е) техническая помощь при возникновении нештатных ситуаций;

ж) в случае возникновения нештатной ситуации, которую невозможно устранить удаленно, выезд специалистов отдела технической поддержки на место возникновения нештатной ситуации;

з) консультации по добавлению новых функций систем;

и) уведомление о новых версиях и функциях программного обеспечения.

Служба технической поддержки на каждом предприятии строится разными способами. Существует несколько моделей службы поддержки (рисунок 1.1).

Служба технической поддержки на предприятии имеет большое значение при внедрении и практическом использовании современных ИТ - подходов и методик.

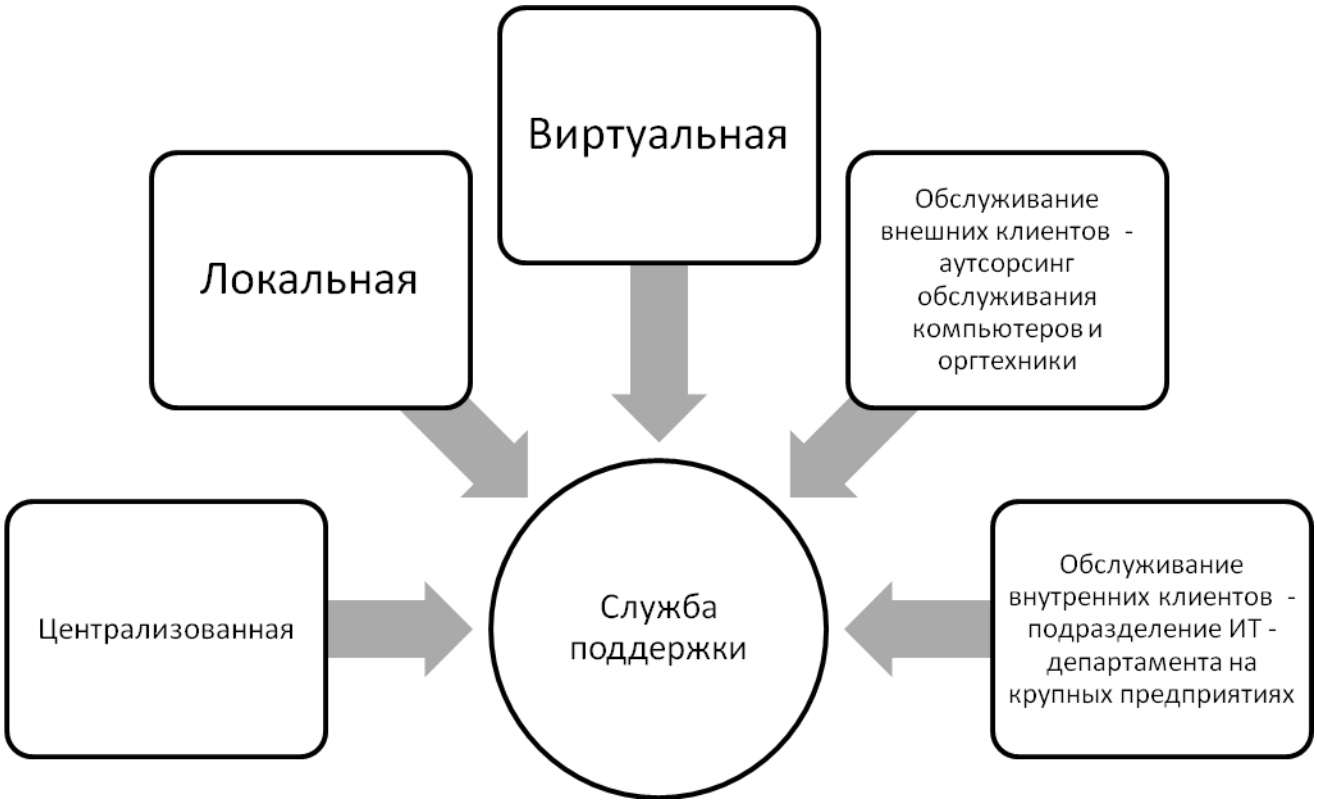

Рисунок 1.1 – Модели службы поддержки.

Правильно организованная техническая поддержка всегда начинается с регистрации всех обращений конечных пользователей, служит единой точкой для общения пользователя с ИТ - службой. Наиболее популярные решения по практической организации технической поддержки часто строятся на базе Call-center. Он является начальной точкой контактов конечных пользователей со службой технической поддержки и служит источником информации об их фактической удовлетворенности уровнем сервиса, что дополняет информацию о технических параметрах качества обслуживания компании-клиента (внешнего или внутреннего).

На больших предприятиях или в крупных компаниях - аутсорсерах служба технической поддержки часто организована по следующему многоуровневому принципу, представленному на рисунке 1.2.

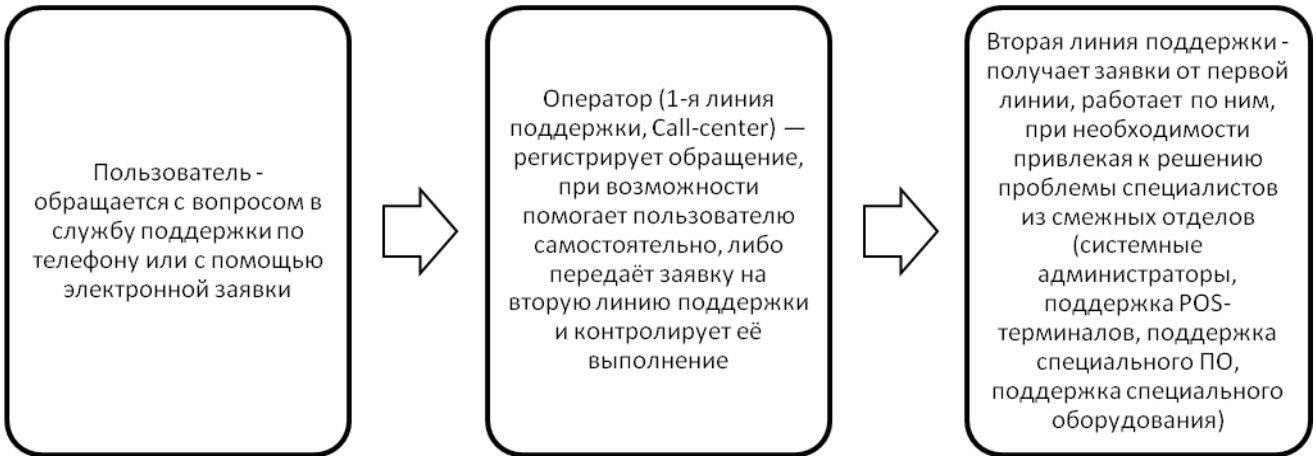

Рисунок 1.2 – Организация службы поддержки по многоуровневому принципу.

Сотрудники технической поддержки - опытные специалисты, способные в режиме реального времени ответить на большинство вопросов, возникших у пользователей при работе с программными продуктами. При необходимости пользователю подскажут порядок действий, чтобы он смог шаг за шагом выполнить их для решения возникшей проблемы. При наличии сложных случаев специалисты могут осуществить подключение к компьютеру пользователя удаленно и проверить настройки.

Для ряда предприятий нецелесообразно держать в штате собственных специалистов технической поддержки. Но в тоже время необходим высокий сервис и качественный ремонт компьютеров и оргтехники, установка и обслуживание всего необходимого ПО.

Прибегая к помощи служб, осуществляющих комплексную техническую поддержку, руководство подобных организаций освобождается от необходимости самостоятельно заботиться о ремонте компьютеров и оргтехники, подборе и настройке программного обеспечения, поиска оптимальных цен на нужные устройства.

Техническая поддержка и сопровождение поставляемого программного обеспечения в послегарантийный период осуществляется по договорам

технической поддержки и сопровождения. Порядок оказания услуг и стоимость послегарантийной поддержки определяются на договорной основе.

Благодаря этому сервису значительно повышается эффективность использования программных продуктов, что повышает продуктивность работы. Пользуясь услугами технической поддержки, организации получают значительную экономическую выгоду.

1.2 Обоснование для разработки микропроцессорного устройства записи телефонных разговоров

Общение с клиентами службы технической поддержки чаще всего сводится к консультации по телефону. Все входящие звонки клиентов поступают на один многоканальный телефонный номер, принимаются и фиксируются секретарем, затем переключаются на сотрудников технической поддержки. Коммутация телефонных линий осуществляется с помощью мини-АТС.

В результате разговора с клиентом, секретарь принимает решение, переключить его на отдел программистов 1С или инженеров-техников. В случае если все программисты и инженеры заняты, секретарь составляет заявку, в которой указывает краткое описание неисправности со слов клиента и его контактный телефон. Алгоритм работы службы технической поддержки показан на рисунке 1.3.

Возможны ситуации, когда при составлении заявки секретарь может совершить ошибку при внесении контактного телефона. В результате этого сотрудник отдела технической поддержки не сможет своевременно связаться с клиентом. Или секретарь может некорректно указать информацию о неполадках, в результате чего заявка попадет не в тот отдел технической поддержки, будет потрачено дополнительное время на её обработку.

При консультации пользователей технической поддержки по телефону могут возникнуть спорные ситуации. Пользователь может остаться недоволен качеством оказанных услуг или оператор технической поддержки может оказать услуги не в полном объеме.

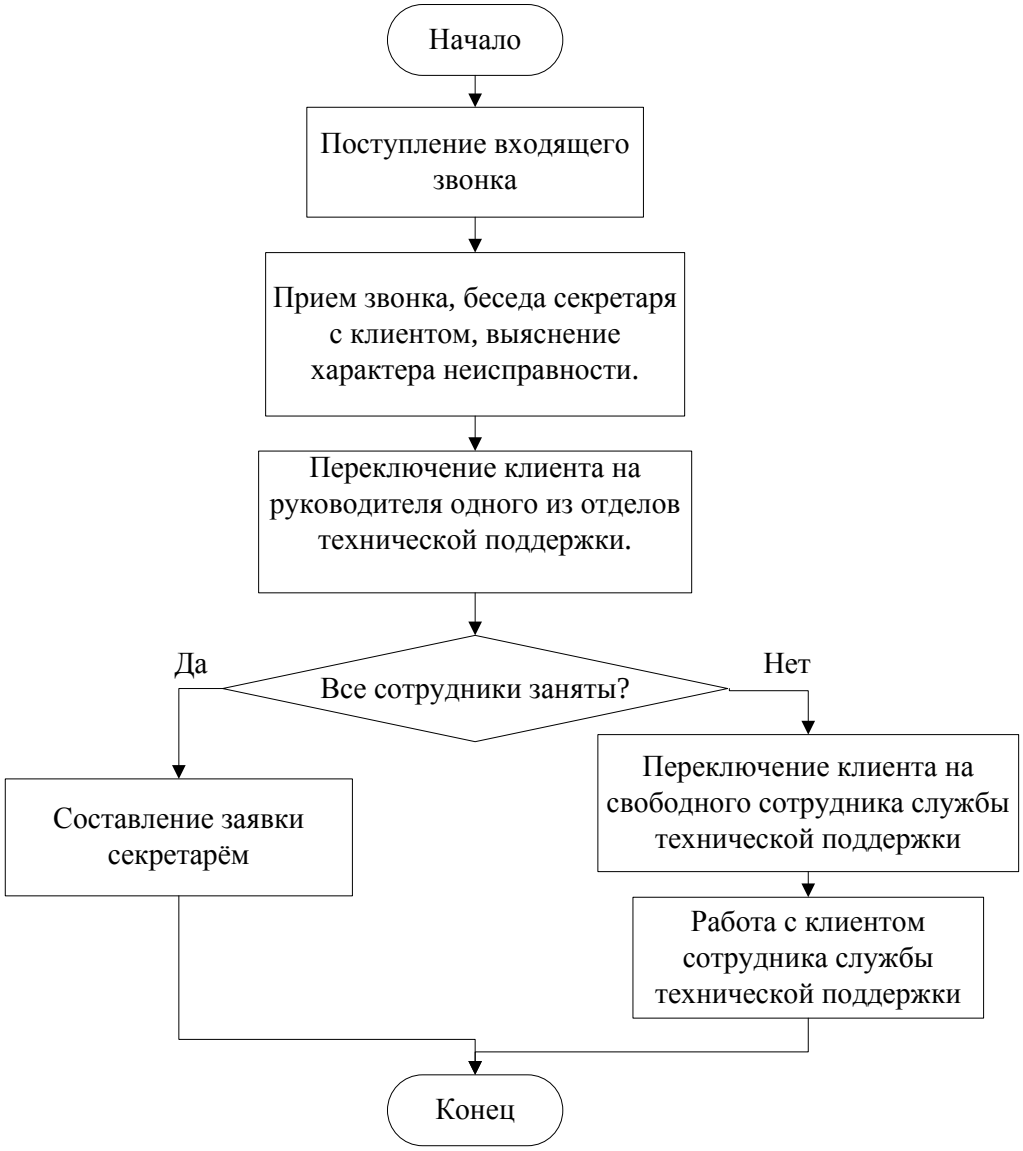

Рисунок 1.3– Алгоритм работы службы технической поддержки в компании.

Для предотвращения подобных ситуаций, повышения качества обслуживания клиентов и трудовой дисциплины в компании, было принято решение разработать устройство, способное записывать все телефонные разговоры для последующего их анализа.

Внедрение микропроцессорного устройства записи телефонных разговоров позволит:

а) регистрировать переговоров диспетчерских служб;

б) повысить трудовую дисциплину сотрудников службы технической поддержки;

в) снизить количество спорных ситуаций и ошибок, возникающих при составлении заявок;

г) вести базу телефонных заказов;

д) способствовать разрешению конфликтных ситуаций с клиентами службы технической поддержки;

е) вести запись важных телефонных звонков и конференций;

ж) повысить уровень безопасности, предупреждая преступные действия различного характера.

1.3 Сравнительный анализ устройств записи телефонных разговоров

1.3.1 Программно – аппаратный комплекс записи телефонных разговоров SpyRecord

Существует ряд способов записи телефонных разговоров для последующего их хранения и анализа. Запись может вестись на магнитную пленку, жесткий диск компьютера или другой цифровой носитель.

Для записи телефонных разговоров в компаниях, оказывающих различные услуги по телефону, на рынке существует большое количество специализированных устройств. Одним из наиболее распространенных является профессиональный программно-аппаратный комплекс записи телефонных разговоров SpyRecord.

SpyRecord – это комплекс многоканальной записи и регистрации телефонных переговоров на компьютер. Комплекс способен вести одновременную запись от 1 до 128 телефонных линий, поддерживает цифровое сжатие, автоматическое определение номера (АОН), определение набранного номера, функцию «электронный секретарь», также комплекс имеет возможность записи как обычных телефонных линий, так и цифровых.

Внешне устройство выполнено в виде отдельного блока с возможностью подключения одной, двух, четырех или восьми телефонных линий (рисунок 1.4). Подключение к ПК осуществляется через USB интерфейс.

В таблице 1.1 приведены основные технические характеристики комплекса записи телефонных разговоров SpyRecord A4.

Рисунок 1.4 - Внешний вид адаптеров SpyRecord.

Таблица 1.1 - Основные технические характеристики комплекса записи телефонных разговоров SpyRecord A4

Параметр	Значение
Количество телефонных линий одновременной записи	4шт.
Напряжение питания от USB-порта	5В
Потребляемая мощность	≤ 600 мВт
Предельный уровень напряжения на стыке с телефонной линией	230 В
Номинальный диапазон входного сигнала	-50 дБ … +10 дБ
Рабочий диапазон частот	250-3500 Гц
1 час записи с применением алгоритма сжатия	3,6 Мб
Рабочий диапазон температур	+5 °С…+40 °С

На рисунке 1.5 представлена схема включения комплекса SpyRecord A4.

В комплекте с данным комплексом поставляется специализированное программное обеспечение. Оно способно автоматически вести журнал и архивацию всех звонков и телефонных разговоров в фоновом режиме, что позволяет использовать ПК, на котором установлен данный комплекс, в

обычном режиме. На рисунке 1.6 представлено окно журнала телефонных разговоров программы SpRecord 3.

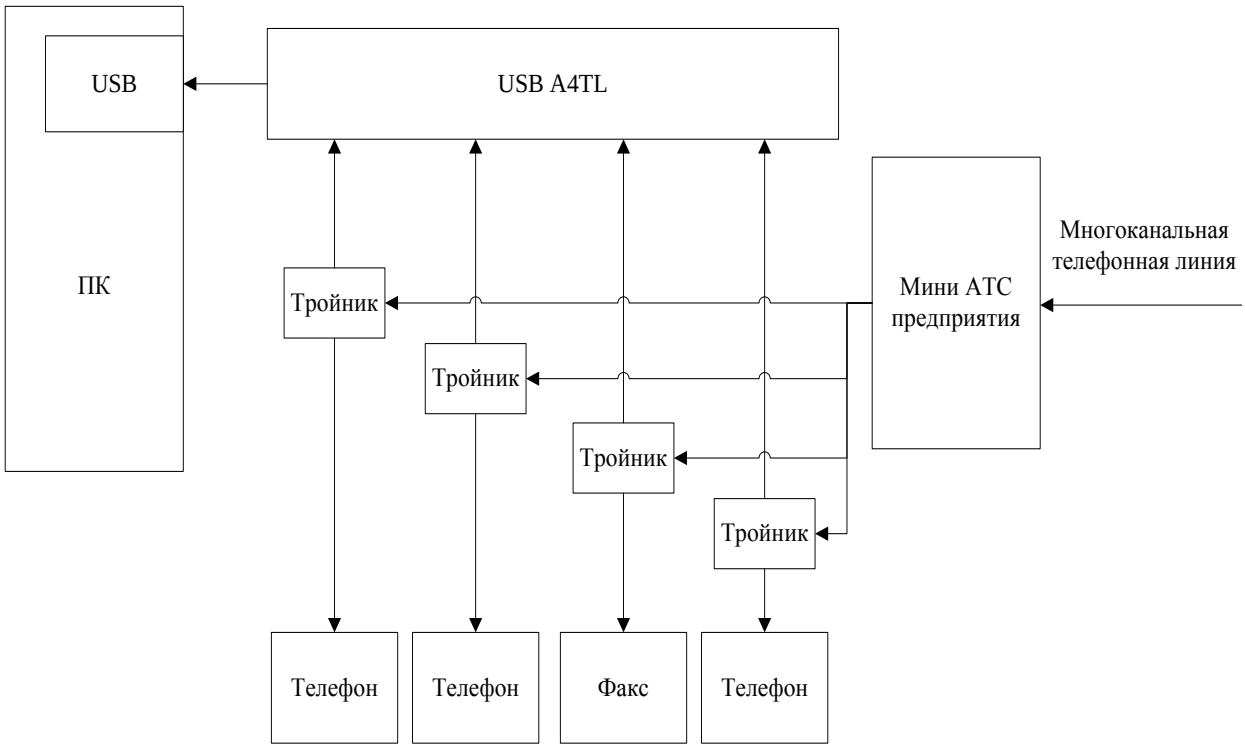

Рисунок 1.5 - Схема включения комплекса SpyRecord A4.

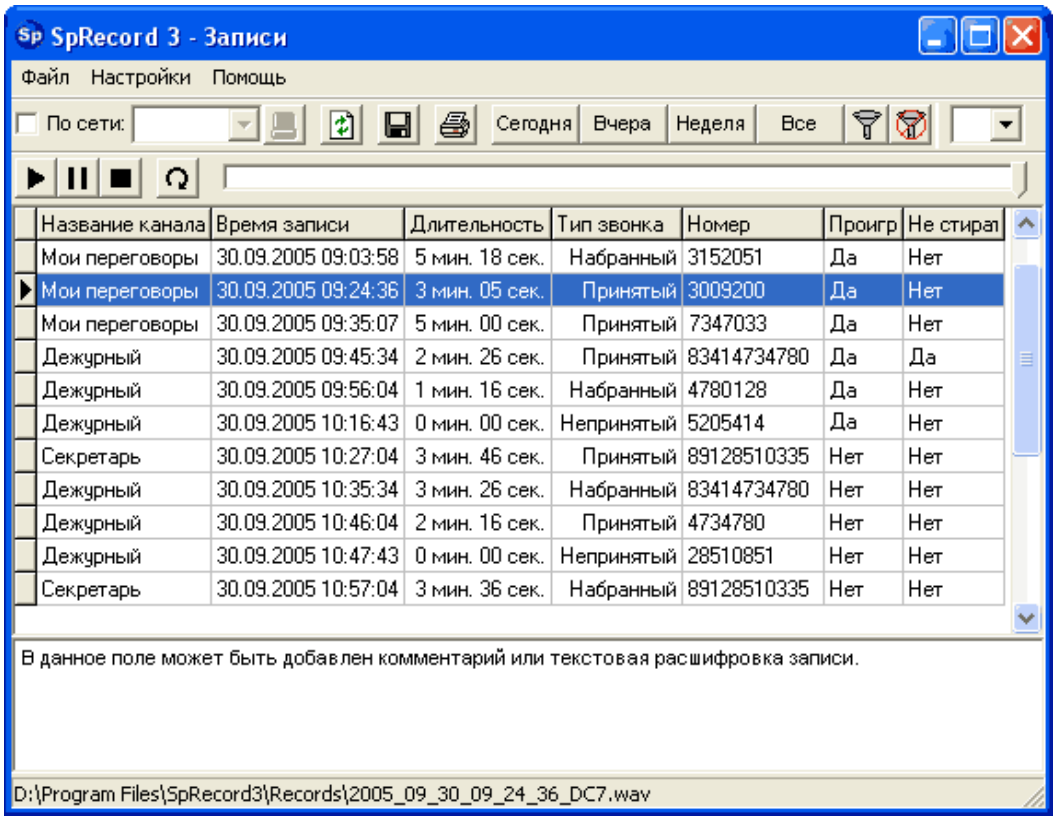

Рисунок 1.6 – Окно журнала телефонных разговоров программы SpRecord3.

В режиме реального времени возможно прослушивать телефонные переговоры любого количества каналов. На рисунке 1.7 представлено окно программы SpRecord 3, иллюстрирующее процесс записи телефонных разговоров.

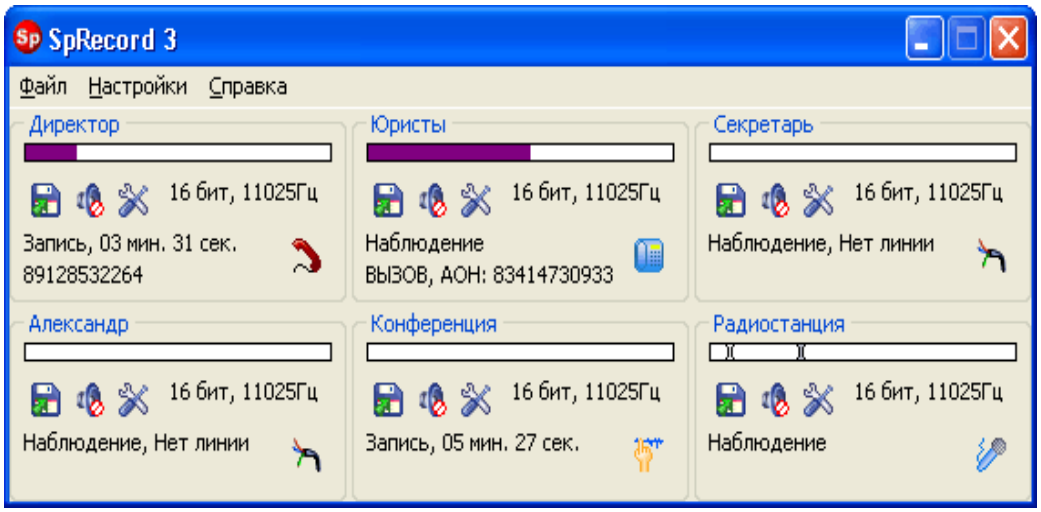

Рисунок 1.7 - Окно программы SpRecord 3.

Системы записи телефонных разговоров SpyRecord соответствуют нормам безопасности и техническим требованиям оборудования, подключаемого к единой сети электросвязи РФ.

Недостатком такого комплекса являются:

а) необходимость взаимодействия с компьютером (питание через USB, запись разговоров на жесткий диск ПК);

б) высокие системные требования к компьютеру;

в) высокая цена комплекса.

1.3.2 Система записи телефонных разговоров DogEar

В качестве альтернативы комплексу SpyRecord может выступать система DogEar, предназначена для организации многоканальной записи акустической информации на базе компьютера.

Система состоит из платы (или нескольких плат), устанавливаемых в стандартные ISA или PCI слота и специального программного обеспечения. На рисунке 1.8 представлен внешний вид печатной платы DogEar.

Рисунок 1.8 - Внешний вид печатной платы цифровой записи телефонных разговоров на две линии DogEar.

Гибкая модульная структура позволяет реализовывать многоканальные системы записи профессионального уровня с широким набором сервисных функций, включая организацию полностью автоматического и скрытого режима работы. На одном ПК возможна запись до 24 каналов. На рисунке 1.9 представлена схема включения платы записи телефонных разговоров на две линии DogEar.

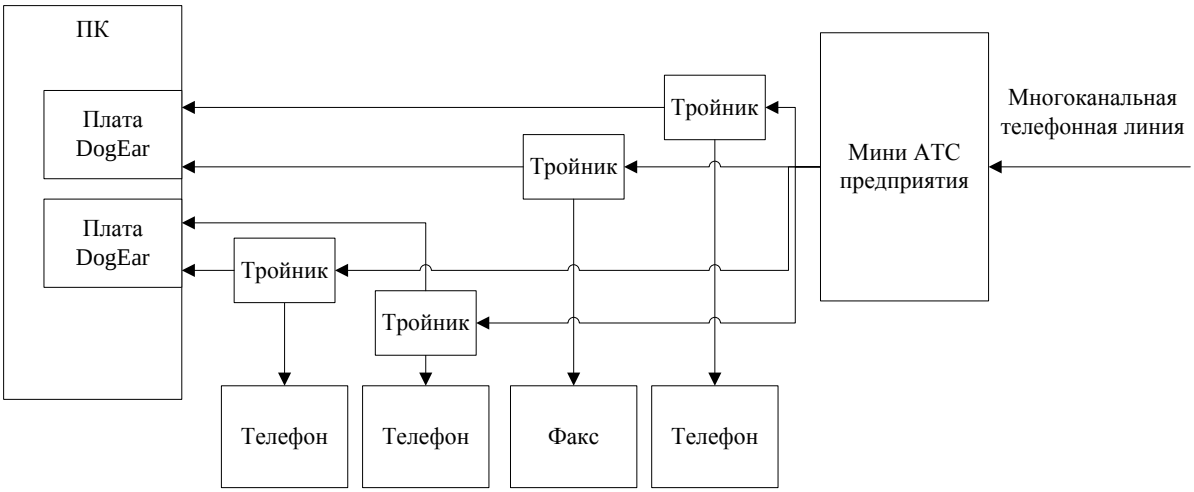

Рисунок 1.9 – Схема включения комплекса DogEar.

Особенностью системы DogEar является то, что она может работать параллельно с другими программами Windows. Исходя из этого, многоканальная система может быть реализована на базе практически любого современного компьютера.

Объем жесткого диска определяется исходя из требуемых задач. На диск объемом 10 ГГб может поместиться 10 суток непрерывной записи, при одновременной работе 8-и каналов со степенью сжатия 1.6 Кб/сек.

Основные функциональные возможности системы DogEar является:

а) автоматическое определение входящих и исходящих номеров, даты, времени и продолжительности сеанса связи;

б) данная информация позволяет сортировать записи в базе, удалять ненужные, осуществлять поиск и прослушать нужный разговор;

в) отображение текущего состояния каналов с указанием времени установления связи, длительности разговора, номера абонента, с которым происходит разговор, и комментария к разговору;

г) в момент разговора всегда виден номер звонящего, существует возможность задавать и сохранять комментарии к конкретному номеру телефона или записи разговора;

д) высокая степень сжатия аудиоинформации (5,58 МБ/Час) - за счет мощных алгоритмов сжатия экономится место на жестком диске, что позволяет хранить большее количество записей без увеличения затрат на дополнительный жесткий диск;

е) возможность прослушивания переговоров во время записи - позволяет оператору выборочно прослушивать любой из каналов без прерывания записи;

ж) просмотр и прослушивание информации, записанной в базе данных - любой зарегистрированный разговор можно неоднократно прослушать, независимо от времени его приема и обработки. Мгновенный доступ к любой по выбору оператора записи, независимо от очередности приема;

з) создание архивов переговоров и резервное копирование на случай потери или повреждения данных;

и) получение статистической информации по загрузке телефонных линий с возможностью разграничения междугородных и местных звонков

к) автоматический контроль за размером базы данных;

л) работа с операционными системами семейства Windows;

м) возможность осуществления удаленного доступа к базе телефонных разговоров;

н) дальнейшая расширяемость - система может работать с несколькими платами различных вариантов исполнения, при этом все каналы объединяются под общим программным интерфейсом.

Каждый канал может настраиваться на любой из режимов записи сигнала: с телефонной линии, микрофона, линейного выхода и т.п. Степень сжатия задается индивидуально для каждого канала и может принимать следующие значения: 16, 8, 4, 1.6, 1, 0.6 кБ/сек. При степени сжатия 1.6 кБ/сек, при воспроизведении записи сохраняются все тоновые особенности голоса.

Система имеет продуманный и интуитивно понятный программный интерфейс, обслуживание системы DogEar не требует привлечения специалистов высокого уровня.

К недостаткам данной системы можно отнести непосредственную зависимость от ПК и высокую цену системы.

1.3.3 Запись телефонных разговоров с помощью голосового модема ZyXEL OMNI 56K PRO EE

В качестве более дешевого варианта устройства для записи телефонных разговоров на предприятии может выступать аналоговый модем с поддержкой голосовых функций и специализированное программное обеспечение.

Голосовой модем - аналоговый телефонный модем с встроенной возможностью передачи и приема голосовых записей по телефонной линии. Современные модемы способны одновременно передавать голос и данные от чего эту группу назвали SVD.

Голосовой модем позволяет определять номер звонящего абонента, использовать автоответчик, системы автоматической рассылки речевых сообщений и т.п. Голосовой модем может быть реализован в виде печатной платы, устанавливающейся на материнскую плату ПК или в виде отдельного устройства.

Существуют две технологии голосовой передачи данных: ASVD и DSVD.

ASVD - аналоговый способ передачи данных, при котором звуковая информация вводится в поток данных в аналоговом виде на этапе модуляции. Скорость потока данных в канале при этом уменьшается. ASVD не позволяет предавать по голосовому каналу звук из компьютера без его преобразования в аналоговую форму.

DSVD - цифровой способ, при котором звук прозрачно внедряется в цифровой поток. При этом звук может как оцифровываться с микрофона на входе и подаваться на наушники с выхода, так и напрямую передаваться с компьютера на компьютер.

В качестве устройства записи телефонных разговоров может выступать аналоговый модем с поддержкой голосовых функций ZyXEL OMNI 56K PRO EE.

Данный модем выполнен в виде отдельного устройства, оснащенного графическим дисплеем и клавиатурой, функциями факса, автоответчика и определителя номера. На рисунке 1.10 изображен внешний вид голосового модема ZyXEL OMNI 56K PRO EE.

Рисунок 1.10 - Внешний вид модема ZyXEL OMNI 56K PRO EE.

При использовании голосового модема для записи телефонных разговоров следует учесть, что такой комплекс необходимо установить на каждое рабочее место сотрудника служб технической поддержки, оснащенное телефонным аппаратом.

Схема включения голосового модема ZyXEL OMNI 56K PRO EE представлена на рисунке 1.11.

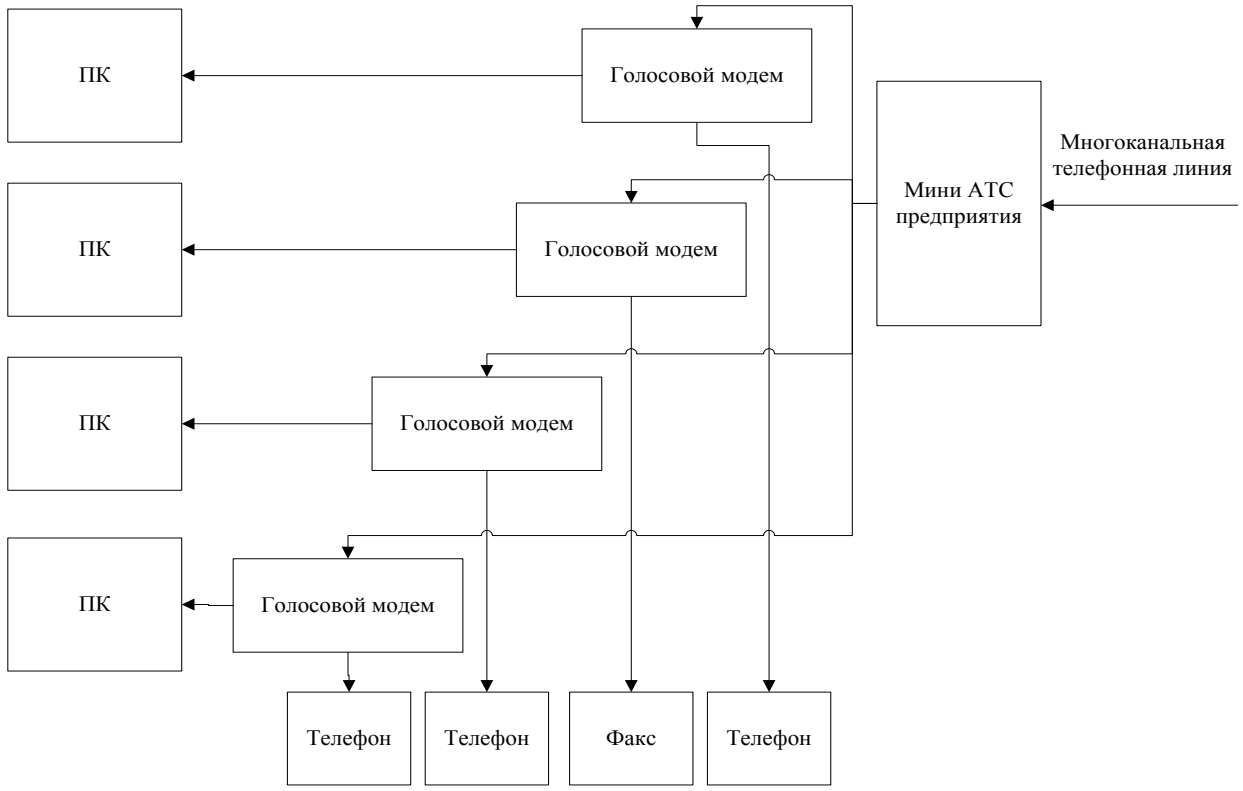

Рисунок 1.11 – Схема включения голосового модема ZyXEL OMNI 56K PRO EE.

Основные характеристики аппаратных и программных средств модема ZyXEL OMNI 56K PRO EE представлены на рисунке 1.12.

Запись телефонного разговора осуществляется с помощью специализированного ПО на жесткий диск компьютера через последовательный интерфейс.

В качестве программного обеспечения, с помощью которого возможно осуществлять запись разговоров на жесткий диск компьютера для данного модема, может выступать утилита «Modem Spy».

Modem Spy позволяет записывать все телефонные разговоры. Запись может выполняться в автоматическом режиме, при этом программа будет вести статистику телефонных разговоров в специальном журнале звонков. В процессе записи звук собеседника не будет звучать слишком тихо, поскольку программа использует автоматическую регулировку усиления сигнала. Диалоговое окно программы Modem Spy представлено на рисунке 1.13.

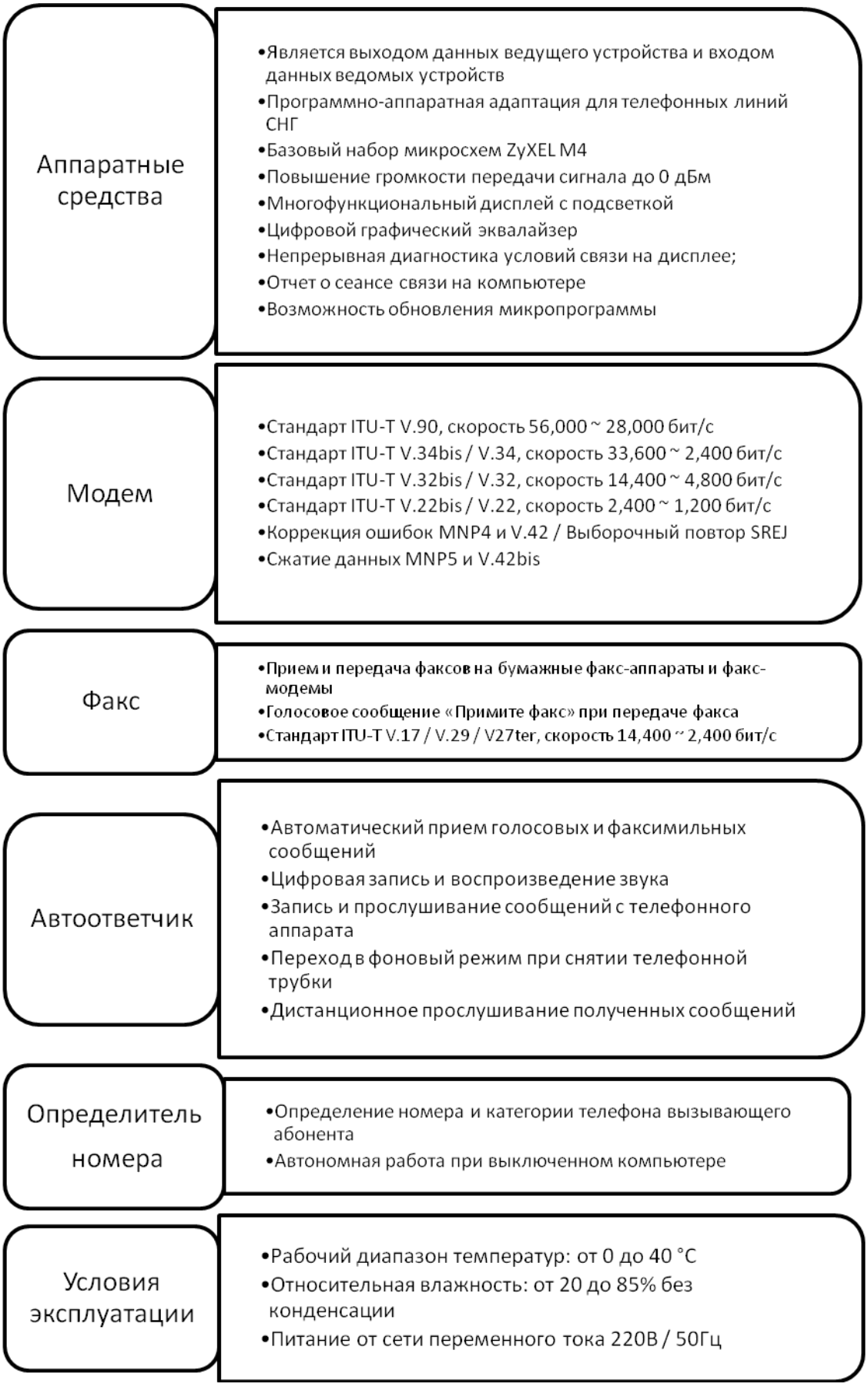

Рисунок 1.12 - Основные характеристики модема ZyXEL OMNI 56K PRO EE.

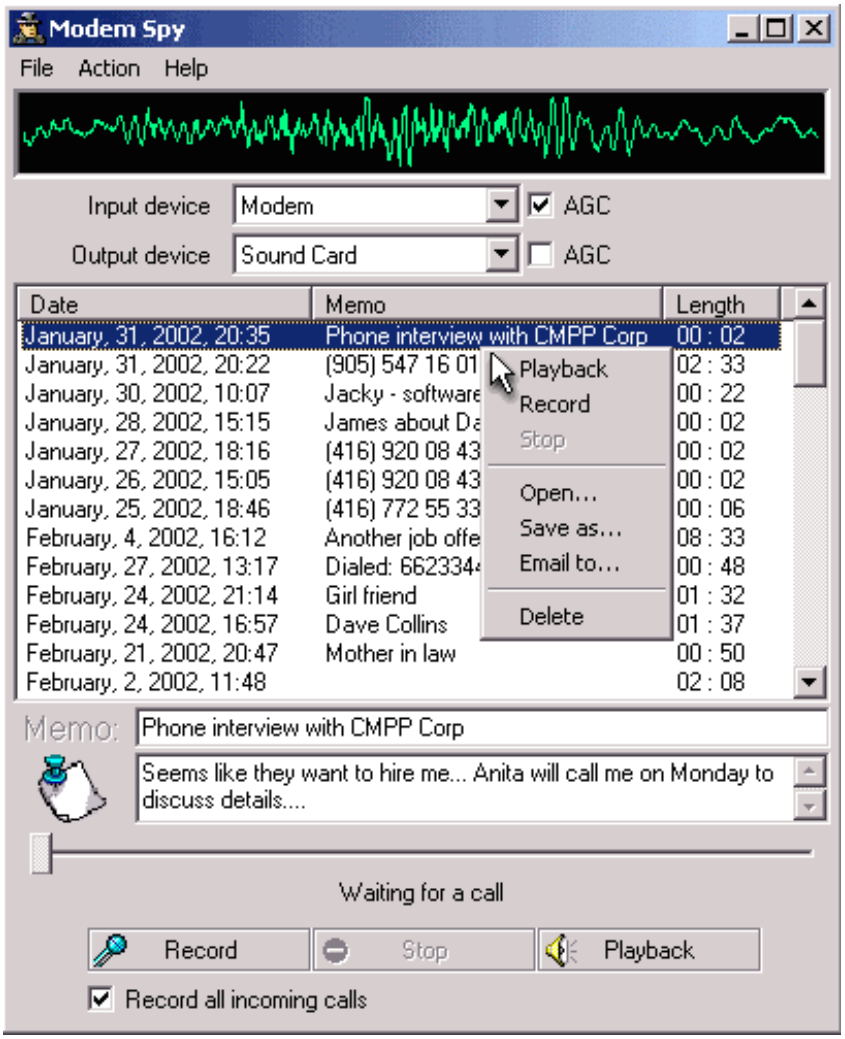

Рисунок 1.13 – Диалоговое окно программы Modem Spy.

Аудиозаписи можно сохранять в форматах MP3 и WAV. Записанные телефонные переговоры могут быть воспроизведены через звуковую карту или проиграны прямо в телефонную линию. Modem Spy может записывать разговоры, даже с использованием тех моделей модемов, которые поддерживает передачу только данных (data modem). Для этого необходимо соединить модем и линейный вход звуковой карты с помощью специального адаптера.

Преимуществами при использовании модема ZyXEL OMNI 56K PRO EE в связке с программой Modem Spy могут выступать:

а) простота в эксплуатации модема;

б) удобный, понятный интерфейс программы Modem Spy;

в) доступность на рынке данных технических средств.

К недостаткам использования данного комплекса для записи телефонных разговоров в компании относятся:

а) модем питается от сети, запись можно производить только на жесткий диск ПК. При отключении питания или выходе компьютера из строя комплекс становится неработоспособным;

б) стоимость модема с голосовыми функциями и лицензированная версия утилиты Modem Spy имеют достаточно высокую цену. Бесплатная версия программы позволяет вести запись разговоров не дольше двух минут.

1.3.4 Автономное устройство записи телефонных разговоров ICON-TR1

Автономное устройство записи телефонных разговоров ICON-TR1 позволяет вести запись телефонных разговоров на карту памяти формата SD в формате WAV. Устройство поддерживает карты памяти емкостью от 4 до 32ГГб, что позволяет записывать телефонные разговоры общей длительностью от 280 до 2200 часов. Каждый разговор записывается в отдельный файл. На карте выделяется лог - файл в формате HTML, в котором регистрируются:

а) время начала и продолжительность разговора;

б) исходящий/входящий номер;

в) ссылка на файл с записью разговора.

Просмотр лог - файла и прослушивание записей может осуществляться при помощи любого интернет - браузера. При переполнении карты новые разговоры записываются поверх самых старых записей.

Внешний вид устройства ICON TR1 представлен на рисунке 1.14.

Устройство ICON TR1 поддерживает работу с одной телефонной линией. Для ведения записи телефонных разговоров сотрудников службы технической поддержки необходима установка данного устройства на каждое рабочее место. Схема включения устройства ICON TR1 представлена на рисунке 1.15.

Рисунок 1.14 – Внешний вид автономного устройства записи телефонных разговоров ICON TR1.

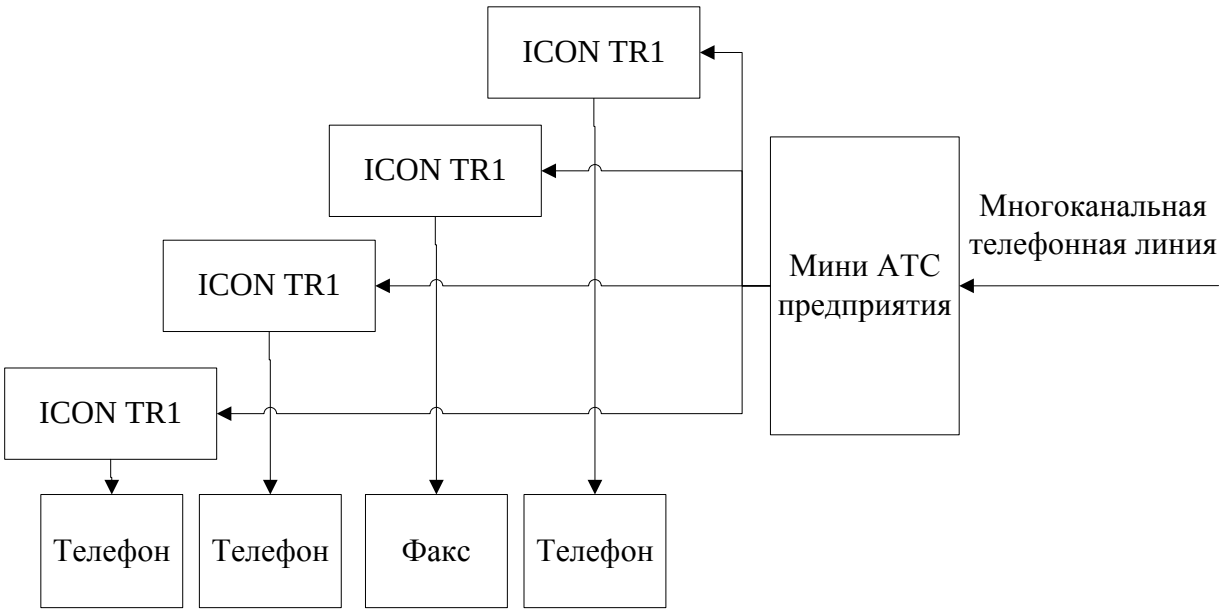

Рисунок 1.15 – Схема включения автономного устройства записи телефонных разговоров ICON TR1.

Основные характеристики устройства ICON TR1 представлены в таблице 12.

К преимуществам устройства ICON TR1следует отнести:

а) низкое энергопотребление и питание от телефонной линии;

б) поддержка карт памяти высокой ёмкости и алгоритмов сжатия аудио данных.

Таблица 1.2 – Основные технические характеристики автономного устройства записи телефонных разговоров ICON TR1

Характеристика	Значение
Количество линий	1
Подключение	В разрыв телефонной линии
Поддерживаемые типы карт памяти	SD, SDHC (до 32Gb)
Формат аудио файлов	WAV
Способ записи	16-bit PCM (1Gb – 18 часов записи)
	A-law (1Gb – 36 часов записи)
	ADPCM (1Gb – 70 часов записи)
Определение исходящего номера	Импульсный и тональный набор
Определение входящего номера	Российский АОН
	CallerID FSK
	CallerID DTMF
Сигнал предупреждения о записи	ГОСТ 28384-89
Питание	От телефонной линии

К недостаткам устройства можно отнести:

а) отсутствие интерфейса для взаимодействия с ПК для доступа к записям телефонных разговоров - для прослушивания и анализа записей необходимо каждый раз изымать карту памяти из устройства, также необходимо устройство считывания SD карт памяти для ПК;

б) высокую стоимость комплекса порядка 3000 рублей для одного рабочего места.

1.4 Требования к разрабатываемому устройству

Устройство должно записывать телефонные разговоры с высоким качеством на жесткий диск ПК через последовательный интерфейс RC-232. В случае выхода из строя ПК или отключения питания, устройство должно гарантировать надежную работу - оно должно переключиться на резервный источник питания и записывать данные на карту памяти формата MMC/SD.

Основой устройства должен выступать микроконтроллер семейства Atmega. Микроконтроллер должен:

а) обрабатывать данные, полученные из канала телефонной линии, управляющие сигналы со встроенной клавиатуры или полученные по последовательному интерфейсу;

б) выводить служебную информацию на LCD дисплей;

в) сохранять данные на карту памяти или жесткий диск компьютера.

Операционный усилитель совместно с трансформатором должен усиливать и разделять входной и выходной сигналы телефонной линии.

Усилитель звуковой частоты должен усиливать звуковой сигнал, направленный на встроенный динамик для прослушивания записей телефонных разговоров.

В системе резервного питания должен быть использован свой микроконтроллер. Микроконтроллер системы резервного питания совместно со стабилизатором напряжения должен регулировать напряжение, перераспределять его на заряд аккумуляторных батарей и выдавать напряжение в 5В для питания микроконтроллера и 3.3В для интерфейса MMC/SD карт памяти. В качестве источника резервного питания должны выступать 6 гальванических элементов АА с номинальным напряжением 1.5В.

В составе устройства необходимо применение микросхемы, способной преобразовывать логические уровни интерфейса RS-232.

Входными данными являются аналоговые данные из канала телефонной линии и управляющие команды с компьютера или встроенной клавиатуры.

Выходными данными являются цифровые данные, записанные на жесткий диск компьютера или карту памяти, служебная информация на ЖК-дисплее.

2 Расчетная часть. Проектирование и расчет основных параметров микропроцессорного устройства записи телефонных разговоров

2.1. Построение структурной схемы микропроцессорного устройства

При разработке структурной схемы МПУ необходимо в первую очередь учесть то, что микроконтроллер является достаточно чувствительным элементом для скачков напряжения. Поэтому, для надежного запуска микроконтроллера применена микросхема стабилизатора напряжения питания, которая контролирует и стабилизирует питание системы. При понижении напряжения питания в МПУ ниже установленного порога, микросхема стабилизации питания формирует сигнал «сброса» микроконтроллера. На рисунке 2.1 представлена структурная схема устройства. В качестве входных данных для устройства выступают аудио данные из канала телефонной линии и управляющие сигналы с ПК или встроенной клавиатуры. В связи с этим устройство должно обеспечить достаточное быстродействие для обработки данных в реальном времени. Получение управляющих сигналов с ПК и передача аудио данных на жесткий диск ПК осуществляется по последовательному интерфейсу. При отключении питания или выходе из строя компьютера устройство должно переключиться на резервный источник питании и обеспечить автономную работу. Запись аудио данных должна производиться на карту памяти. Взаимодействие пользователя с устройством должно осуществляться через интегрированный интерфейс: встроенную клавиатуру, LCD дисплей и динамик.

2.2 Выбор аппаратного и программного обеспечения

2.2.1 Выбор компонентов микропроцессорного устройства

2.2.1.1 Обоснование выбора микроконтроллера

ATmega32 – это 8-разрядный CMOS микроконтроллер, основанный на расширенной AVR RISK-архитектуре. За счет выполнения большинства инструкций за один машинный цикл ATmega32 достигает производительности 1млн. операций в секунду при тактовой частоте 1МГц, что позволяет

проектировщикам систем оптимизировать соотношение энергопотребления и быстродействия.

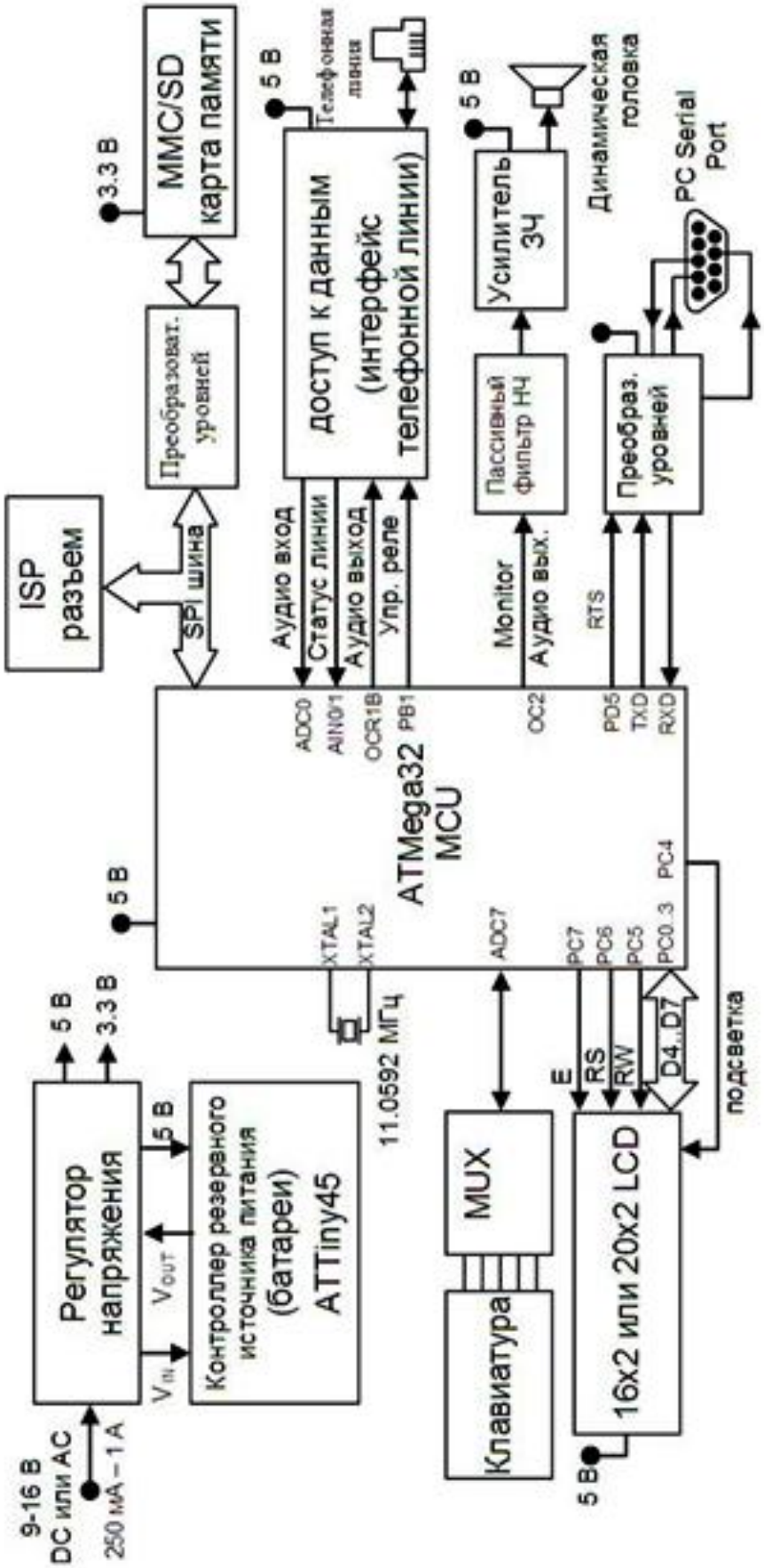

Рисунок 2.1 - Структурная схема микропроцессорного устройства записи телефонных разговоров.

Основные характеристики микроконтроллера ATmega32 приведены на рисунке 2.2.

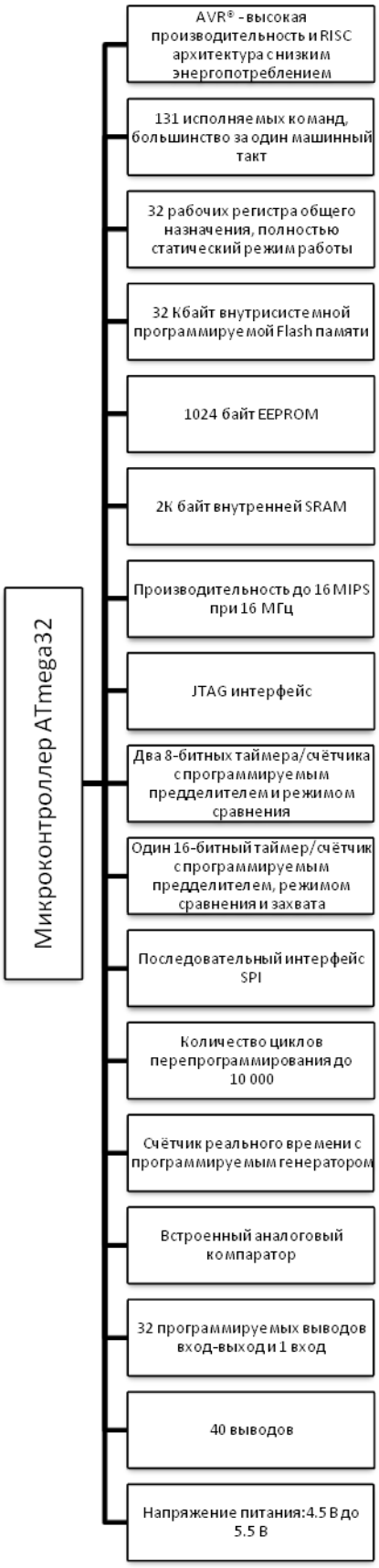

Рисунок 2.2 - Основные характеристики микроконтроллера ATmega32.

Структурная схема микроконтроллера ATmega32 представлена на рисунке 2.3.

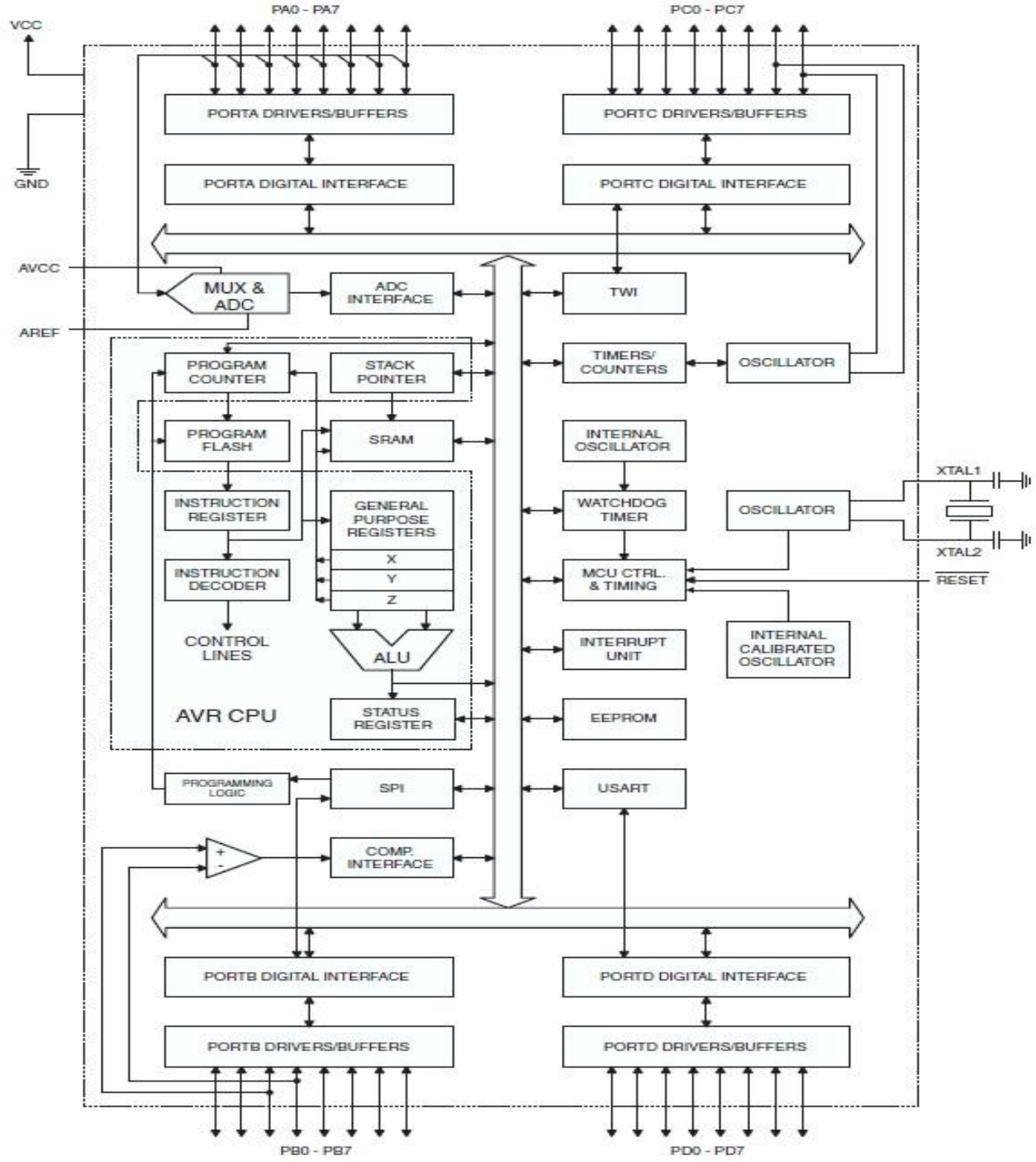

Рисунок 2.3 - Структурная схема микроконтроллера ATmega32.

Расположение выводов микроконтроллера ATmega32 показано на рисунке 2.4.

Назначения выводов микроконтроллера представлено в таблице 2.1.

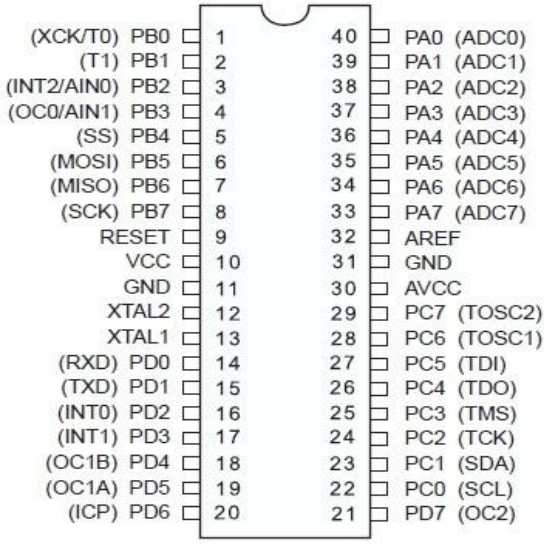

Рисунок 2.4 - Расположение и обозначение выводов микроконтроллера ATmega32.

Таблица 2.1 – Назначение выводов микроконтроллера ATmega32

Обозначение	Вывод	Описание
VCC	10	Напряжение питания цифровых элементов.
GND	11, 31	Земля.
Port A (PA7..PA0)	33-40	Аналоговый вход АЦП. 8-разрядный порт двунаправленного ввода/вывода с внутренними подтягивающими к полюсу резисторами.
Port B (PB7..PB0)	1-8	8-разрядный порт двунаправленного ввода/вывода с внутренними подтягивающими к полюсу резисторами.
Port C (PC7..PC0)	22-29	8-разрядный порт двунаправленного ввода/вывода с внутренними подтягивающими к полюсу резисторами.
Port D (PD7..PD0)	14-21	8-разрядный порт двунаправленного ввода/вывода с внутренними подтягивающими к полюсу резисторами.
RESET	9	Вход сброса.
XTAL1	13	Вход инвертирующего усилителя генератора и вход внешней синхронизации.
XTAL2	12	Выход инвертирующего усилителя генератора
AVCC	30	Вход питания для Port A и АЦП. Должен быть внешне связан с VCC, даже если АЦП не используется. Если АЦП задействован, то этот вывод должен быть связан с VCC через фильтр низких частот.
AREF	32	Вход подключения источника опорного напряжения АЦП.

2.2.1.2 Обоснование выбора микроконтроллера для системы резервного питания

ATtiny45 - экономичный 8-разрядный КМОП микроконтроллер, выполненный по усовершенствованной AVR RISC-архитектуре. За счет выполнения большинства инструкций за один машинный цикл

микроконтроллеры ATtiny45 достигает производительности 1млн. операций в секунду при тактовой частоте 1МГц, что позволяет разработчику оптимизировать потребляемую мощность и быстродействие.

Основные характеристики микроконтроллера ATtiny45 представлены на рисунке 2.5.

Рисунок 2.5 - Основные характеристики микроконтроллера ATtiny45.

Структурная схема микроконтроллера ATtiny45 представлена на рисунке 2.6.

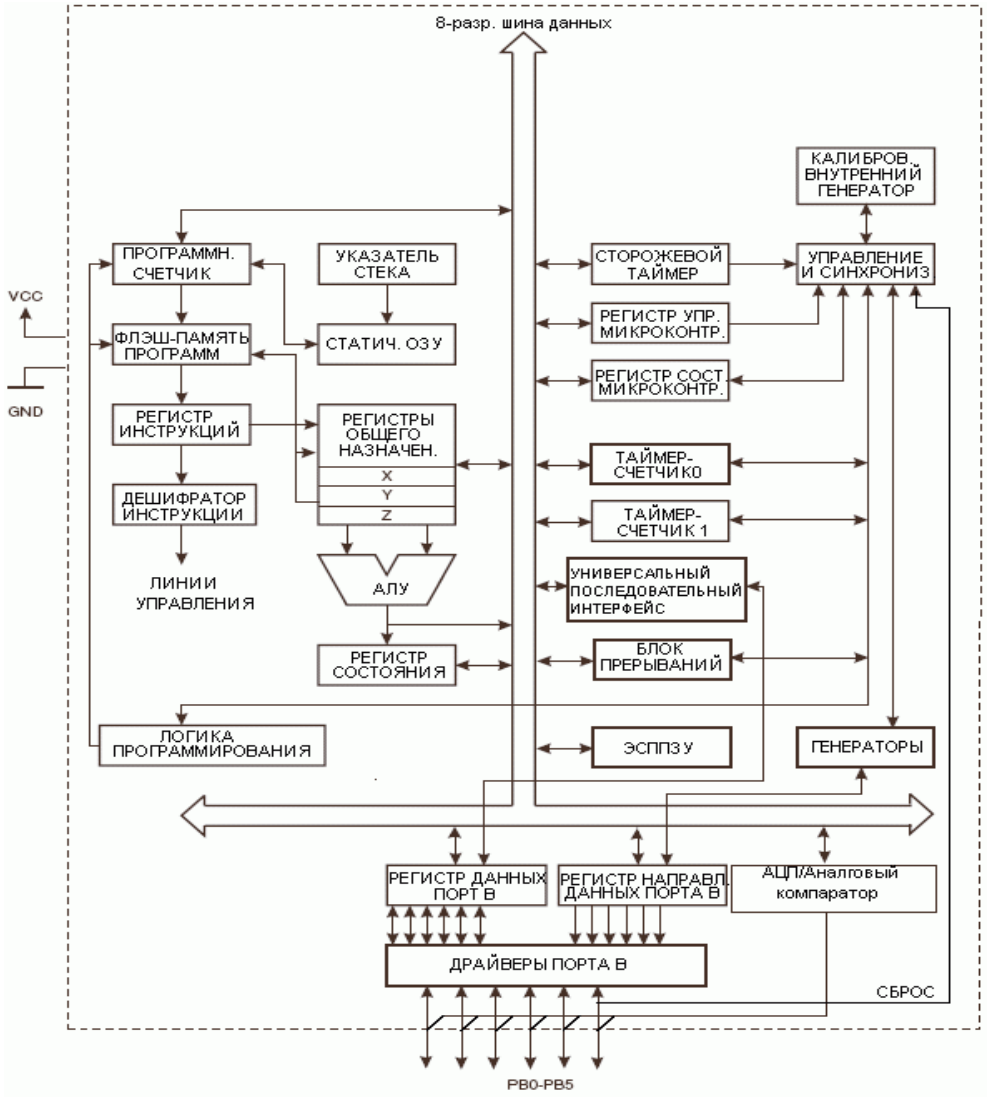

Рисунок 2.6 - Структурная схема микроконтроллера ATtiny45.

Расположение выводов микроконтроллера ATtiny45 представлено на рисунке 2.7.

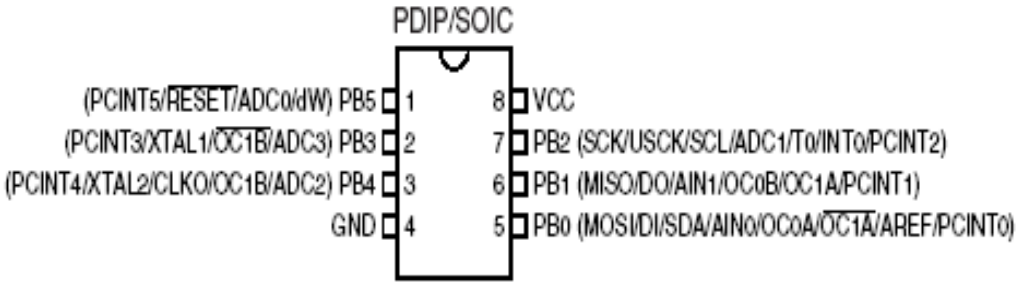

Рисунок 2.7 - Расположение и обозначение выводов микроконтроллера ATtiny45.

Назначения выводов микроконтроллера ATtiny45 представлено в таблице 2.2.

Таблица 2.2 – Назначения выводов микроконтроллера ATtiny45

Обозначение	Вывод	Функция
VCC	8	Напряжение питания цифровых элементов.
GND	4	Земля.
Port B (PB7..PB0)	1-8	6-разрядный порт двунаправленного ввода/вывода с внутренними подтягивающими к полюсу резисторами.
RESET	9	Вход сброса.

2.2.1.3 Обоснование выбора регулятора напряжения

LM317 – это монолитная интегральная схема, в корпусах TO-220, ISOWATT220, TO-3 или D2PAK. Представляет собой положительный регулятор напряжения с выходным током более 1.5A и диапазоном напряжения от 1.2 до 37 вольт. Номинал выходного напряжения регулируется переменным резистором, что делает устройство очень простым в применении.

Основные особенности регулятора напряжения:

а) диапазон выходного напряжения 1.2 - 37 V;

б) выходной ток более 1.5 A;

в) нестабильность выходного напряжения 0.1%;

г) защита от короткого замыкания;

д) защита от перегрева.

Электрические характеристики микросхемы LM317 представлены в таблице 2.3.

Таблица 2.3 – Электрические характеристики микросхемы LM317

Обоз.	Параметр	Условия	LM317			Ед. изм.
			Мин	Тип	Макс	
ΔV_o	Нестабильность входного напряжения	V_i-V_o=3 to 40V, T_j=25°C	-	0.01	0.04	%/V
		V_i-V_o=3 to 40V	-	0.02	0.07	%/V
ΔV_o	Нестабильность выходного напряжения	$V_o \leq 5V$, I_o от 10mA до I_{max}, T_j=25°C	-	5	25	mV
		$V_o \leq 5V$, I_o от 10mA до I_{max}	-	20	70	mV
		$V_o \geq 5V$, I_o от 10mA до I_{max}, T_j=25°C	-	0,1	0,5	%
		$V_o \geq 5V$, I_o от 10mA до I_{max}	-	0,3	1,5	%
I_{adj}	Ток регулиров. вывода	-		50	100	µA

V_{REF}	Опорное напряжение	V_i-V_o=2.5 to 40V, I_o от 10mA до I_{max}	1,2	1,25	1,3	V
$\Delta V_o/V_o$	Температурная стаб. вых. напряжения	-	-	-	1	%
$I_{o(min)}$	Минимальный выходной ток	V_i-V_o = 40V	-	3,5	10	mA
$I_{o(max)}$	Максимальный выходной ток	V_i-V_o ≤ 15V	1,5	2,2		A
		V_i-V_o=40V, T_j=25°C		0,4		A
e_N	Выходной уровень шумов	B = 10Hz to 10KHz, T_j=25°C	-	0,003	-	%

Внутренняя структура микросхемы регулятора напряжения представлено на рисунке 2.8.

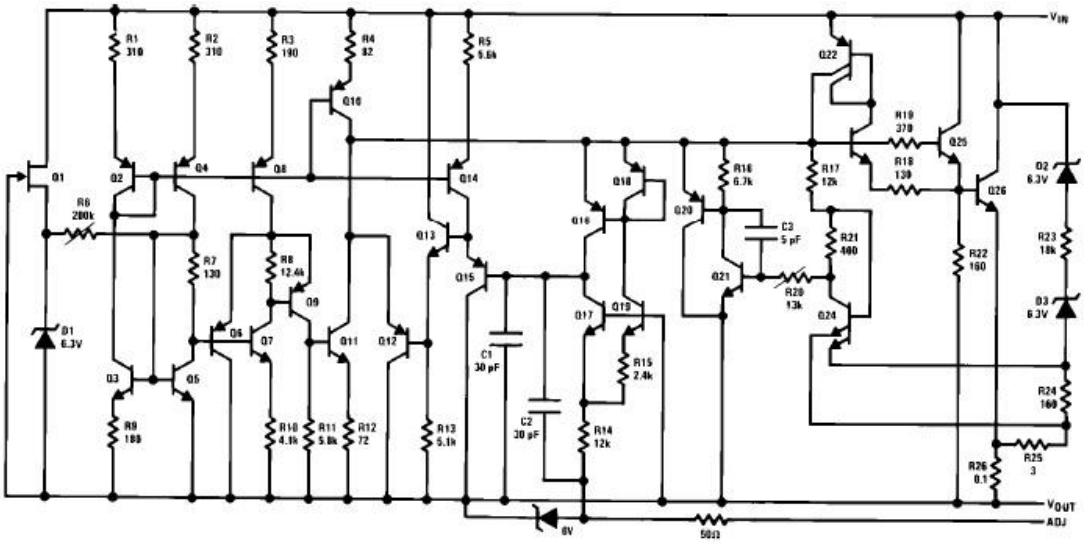

Рисунок 2.8 - Внутренняя структура микросхемы LM317.

Расположение выводов микросхемы представлено на рисунке 2.9.

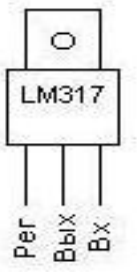

Рисунок 2.9 - Расположение выводов регулятора напряжения LM317.

Микросхема LM317 создает внутреннее опорное напряжение 1.25V между выходом и регулировочным выводом. Это сделано для того, чтобы установить постоянный ток через резисторный делитель. Регулятор сконструирован таким образом, чтобы уменьшить значение I_{ADJ} (максимум

100mA) и поддерживать его постоянным при изменении входного напряжения и нагрузки. В виду того что LM317 имеет плавный регулятор и определяет разницу напряжений между входом и выходом, питание высоким напряжением может продолжаться сколь угодно долго в максимально допустимых пределах. Благодаря этому, путем установки фиксированного резистора между регулировочным выводом и выходом, устройство можно использовать как прецизионный регулятор тока. Для уменьшения пульсаций регулятора можно добавить блокировочный конденсатор 0.1 μF на входе, и конденсаторы на выходе для подавления шумов на выходе. Также хорошей практикой является использование защитных диодов.

2.2.1.4 Обоснование выбора усилителя низкой частоты

Микросхема LM386 представляет собой усилитель низкой частоты (УНЧ) мощностью от 0,3 до 1 Ватта (в зависимости от индекса микросхемы). Стоит отметить, что полоса пропускания микросхемы составляет 300 кГц, что делает возможным применение этой микросхемы для усиления частоты не только звукового диапазона.

Характеристики микросхемы LM386 приведены в таблице 2.4.

Таблица 2.4 – Характеристики микросхемы LM386

Характеристика	Условия	Значение	Единица измерения
Напряжение питания	-	4 … 12	В
Входное сопротивление	-	50	кОм
Выходная мощность	Uпит = 5В, R нагр. = 8Ом, THD = 10%	0,325	Вт
Частотный диапазон	-	до 300	кГц

Внутренняя структура микросхемы LM386 представлена на рисунке 2.10.

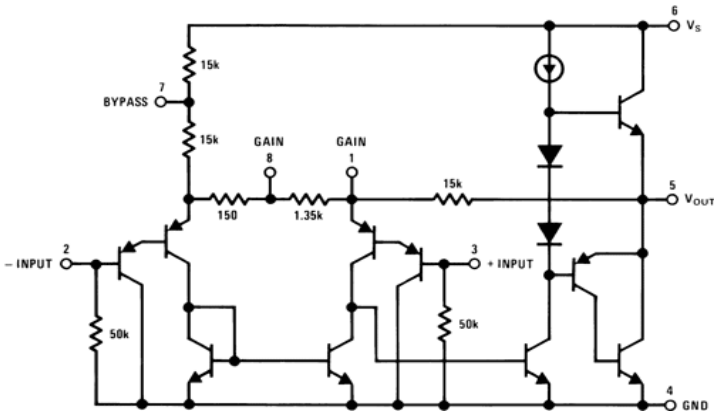

Рисунок 2.10 – Внутренняя структура микросхемы LM386.

Расположение выводов усилителя низкой частоты представлено на рисунке 2.11.

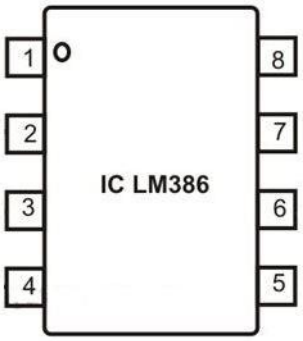

Рисунок 2.11 - Расположение выводов микросхемы LM386.

В таблице 2.5 приведено назначение выводов микросхемы LM386.

Таблица 2.5 - Назначение выводов усилителя низкой частоты LM386

Номер вывода	Вывод	Назначение
1,8	Gain	Управление усилением
2,3	Input(-) Input(+)	Дифференциальные входы
4	GND	Земля
5	V_{out}	Выход
6	V_s	Плюс питания
7	Bypass	

Выводы 1 и 8 позволяют управлять коэффициентом усиления. Если между этими выводами ничего не включено (точнее, включен только встроенный в микросхему резистор сопротивлением 1,35 кОм), коэффициент усиления равен 20 (26dB) (смотри рисунок 16). Если между этими выводами включить конденсатор, коэффициент усиления увеличивается до 200 (46dB). Последовательное включение резистора и конденсатора позволяет выбрать произвольный коэффициент усиления от 20 до 200. Если планируется использовать микросхему LM386 с высоким коэффициентом усиления (между выводом 1 и 8 включен конденсатор или резистор и конденсатор), следует зашунтировать неиспользуемые выводы путем подключения их к земле через конденсатор емкостью 0,1 мкФ. Это позволит исключить снижение коэффициента усиления и возможную неустойчивую работу (самовозбуждение)

усилителя.

Подключая внешние элементы к встроенным в микросхему резисторам в цепи обратной связи можно управлять усилением и частотной характеристикой усилителя.

2.2.1.5 Обоснование выбора операционного усилителя

Микросхема LM358N - это двухканальный операционный усилитель широкого применения для работы в бытовом диапазоне температур (0..+70°C).

Микросхема LM358 по функциональному назначению и расположению выводов аналогична таким микросхемам как LM158, LM258, LM2904, но отличается от них температурным диапазоном работы и незначительно другими параметрами.

Микросхема LM358N также может поставляться с маркировкой LM358P. Основные характеристики микросхемы приведены в таблице 2.6.

Таблица 2.6 - Основные характеристики микросхемы LM358N

Параметр	Мин.	Тип.	Макс.
Напряжение смещения	-	±2mV	±7mV
Синфазный входной ток	-	20nA	150nA
Дифференциальный входной ток	-	±2nA	±30nA
Выходной ток	20mA	40mA	60mA
Коэффициент ослабления синфазных помех	70dB	85dB	-
Коэффициент усиления по напряжению	-	50V/mV	100V/mV
Коэффициент гармонических искажений	-	0,02%	-
Ток потребления	-	0,7mA	2,0mA
Скорость нарастания	-	0,3V/µS	0,6V/µS
Граничная частота	-	0,7MHz	1,1MHz

Предельные режимы работы микросхемы представлены в таблице 2.7.

Таблица 2.7 - Предельные режимы работы микросхемы LM358N

Напряжение питания	+32V или ±16V
Входное напряжение	-0,3…+32V
Дифференциальное входное напряжение	32V
Выходной ток (ограничен внутренне)	40mA
Диапазон температур	0..+70°C

Расположение выводов микросхемы LM358N приведено на рисунке 2.12, описание выводов представлено в таблице 2.8.

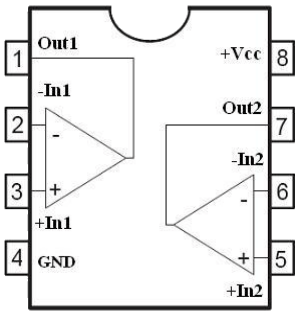

Рисунок 2.12 – Расположение выводов микросхемы LM358N.

Таблица 2.8 - Назначение выводов микросхемы LM358N

N	Наименование	Назначение
1	Out1	Выход 1
2	-In1	Инвертирующий вход 1
3	+In1	Не инвертирующий вход 1
4	GND	-Питания (общий)
5	+In2	Не инвертирующий вход 2
6	-In2	Инвертирующий вход 2
7	Out2	Выход 2
8	+VCC	+ Питание

2.2.1.6 Обоснование выбора стабилизатора напряжения для питания основного контроллера

Стабилизация напряжения — преобразование электрической энергии, позволяющее получить на выходе напряжение, находящееся в заданных пределах при значительных колебаниях входного напряжения и сопротивления нагрузки. По типу выходного напряжения стабилизаторы делятся на стабилизаторы постоянного тока и переменного тока.

Стабилизатор напряжения – это один из старейших и наиболее широко распространенных типов интегральных схем в электронной промышленности. За много лет применения технические характеристики этих интегральных схем были существенно улучшены. В данном МПУ требуется стабилизация напряжения на 5В также током до 1,5А. Для этого будет использован самый распространенный, надежный, а также недорогой стабилизатор LM7805C.

На рисунке 2.13 представлена внутренняя структура стабилизатора напряжения LM7805C.

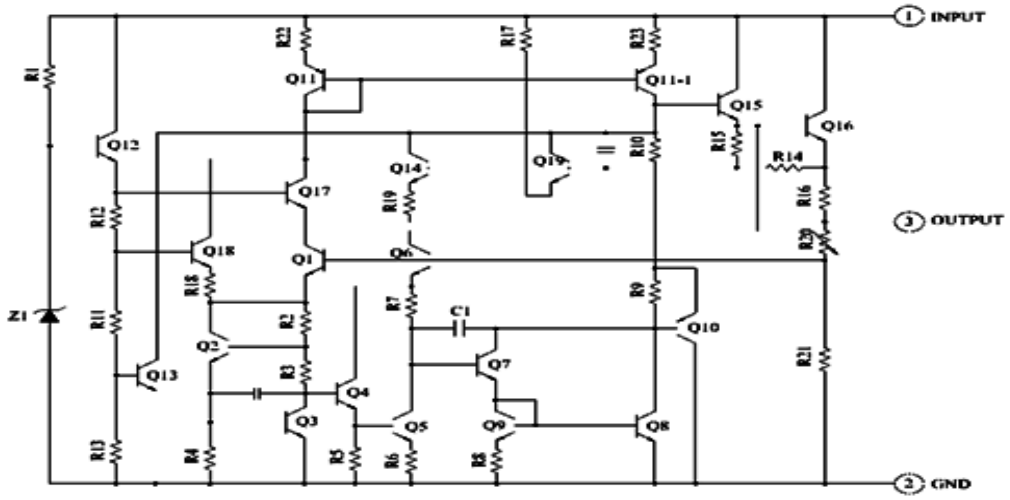

Рисунок 2.13 – Внутренняя структура стабилизатора напряжения LM7805C.

На рисунке 2.14 представлено расположение выводов стабилизатора напряжения.

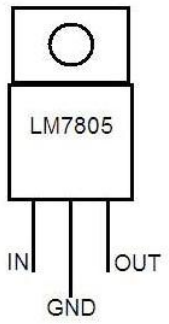

Рисунок 2.14 - Расположение выводов стабилизатора напряжения LM7805C.

Назначение выводов микросхемы LM7805C представлено в таблице 2.9.

Таблица 2.9 – Назначение выводов стабилизатора напряжения LM7805C

Номер вывода	Обозначение	Функция
1	IN	Входное напряжение
2	GND	Земля
3	OUTPUT	Выходное напряжение

Предельные допустимые параметры стабилизатора напряжения LM7805C:

а) рабочая температура: -50/+150°C;

б) выходной ток: 1.5A;

в) рабочее напряжение: 5В (от 4,8В до 5,2В).

2.2.1.7 Обоснование выбора стабилизатора напряжения для питания разъема карты памяти MMC/SD

LM2937-3.3 – это стабилизатор с малым падением напряжения и нагрузочной способностью 500мА, может быть выполнен в двух видах корпусов: TO-263 и SOT-223. Корпуса с расположением выводов стабилизатора напряжения представлены на рисунке 2.15.

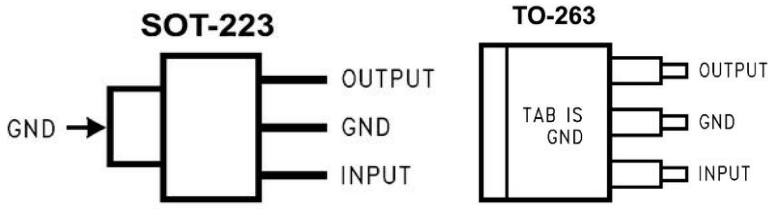

Рисунок 2.15 – Типы корпусов и выводы стабилизатора напряжения LM2937-3.3.

В таблице 2.10 приведено назначение выводов стабилизатора напряжения.

Таблица 2.10 – Назначение выводов микросхемы LM2937-3.3

Номер вывода	Обозначение	Функция
1	INPUT	Входное напряжение
2	GND	Земля
3	OUTPUT	Выходное напряжение

Основные параметры микросхемы LM2937-3.3:

а) входное напряжение: от 5,35В до 26В

б) выходное напряжение: 3.3В

в) выходной ток: 500 мА;

г) рабочая температура: -40/+125°С.

2.2.1.8 Архитектура последовательной шины передачи данных RS-232

В работе с электрическими приборами и сетями, с радиоизмерительной аппаратурой и другими видами техники необходимо наличие не только самого оборудования, но и специальных дополнительных элементов к нему, которые позволяют сделать работу более комфортной и удобной.

RS-232 (Recommended Standard 232) - стандарт последовательной асинхронной передачи двоичных данных между терминалом и коммуникационным устройством.

RS-232 - интерфейс передачи информации между двумя устройствами на расстоянии до 15 метров. Информация передается по проводам цифровым сигналом с двумя уровнями напряжения. Логическому "0" соответствует положительное напряжение от +5 до +15В, логической "1" отрицательное от -5 до -15В для передатчика. Асинхронная передача данных осуществляется с фиксированной скоростью при самосинхронизации фронтом стартового бита.

Интерфейс RS-232-C был разработан для простого применения, однозначно определяемого по его названию: «Интерфейс между терминальным оборудованием и связным оборудованием с обменом по последовательному двоичному коду».

Интерфейс RS-232 обеспечивает соединение двух устройств, одно из которых называется DTE - Оконечное Оборудование Данных (ООД), второе - DCE –Оборудование Передачи Данных (ОПД).

Как правило, DTE (ООД) - это компьютер, а DCE (ОПД) - это периферийные устройства – совместимая с ПК оргтехника. Чаще RS-232 используется в промышленном и узкоспециальном оборудовании, встраиваемых устройствах. На рисунке 2.16 представлена схема распайки управляющего кабеля RS-232 9pin.

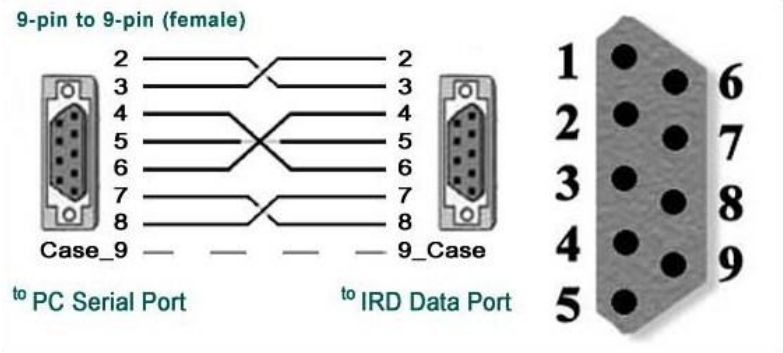

Рисунок 2.16 - Схема распайки кобеля управления RS-232.

2.2.1.9 Архитектура последовательной шины передачи данных ISP

ISP (In Serial Programming) – технология программирования электронных компонентов уже установленных в устройство. До появления этой технологии компоненты программировались перед установкой в устройство, для их перепрограммирования требовалось их извлечение из устройства.

Главным преимуществом технологии является возможность объединения процесса программирования и тестирования при производстве, исключив отдельную фазу программирования компонентов перед окончательной сборкой. Технология также позволяет производителям устройств обойтись без закупки заранее запрограммированных компонентов, выполняя программирование прямо в процессе производства. Это позволяет снизить стоимость производства и вносить изменения в программируемую часть устройства без остановки производства.

Существуют два основных способа ISP они представлены на рисунке 2.17.

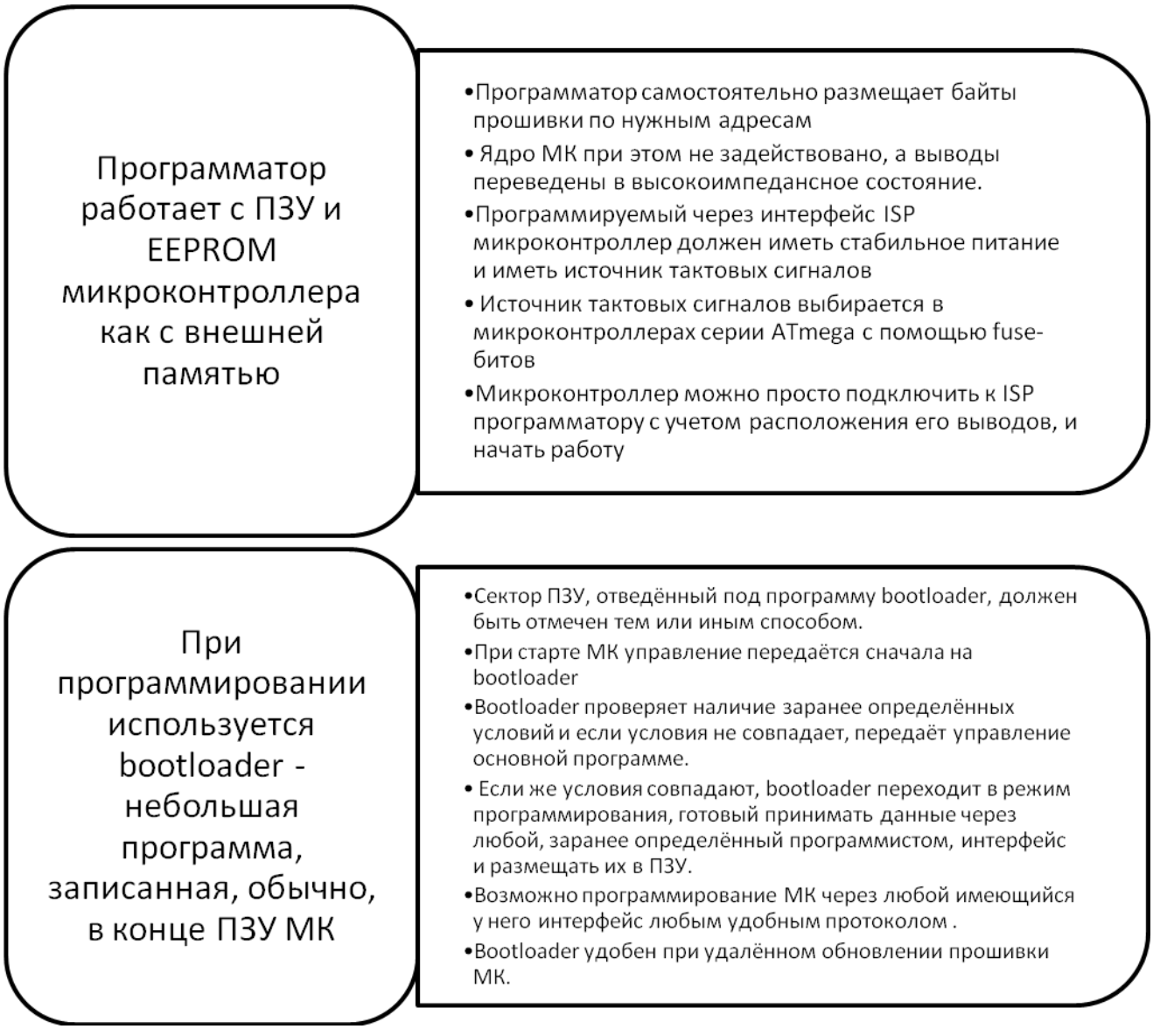

Рисунок 2.17 – Основные способы программирования через ISP.

Шина ISP (In Serial Programming) - это шина, предназначенная для программирования чипа уже подключенного в некоторую схему по последовательному протоколу. На рисунке 2.18 представлена схема разводки разъема шины ISP.

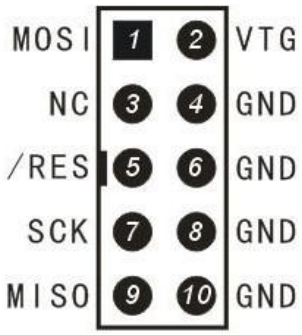

Рисунок 2.18 – Схема разводки разъема ISP.

Назначение выводов разъема ISP:

а) MISO - выход данных для чтения памяти;

б) VTG - вывод источника питания;

в) GND - земля;

г) Reset - вывод сигнала перезагрузки микропроцессора;

д) SCK - вход тактовой частоты для загрузки/чтения памяти;

е) MOSI - вход данных для загрузки памяти.

Выводы MOSI, MISO, SCK должны быть соединены с соответствующими контактами MCS-51 совместимого микроконтроллера.

2.2.1.10 Архитектура параллельной шины передачи данных для подключения LCD дисплея

Для отображения служебной информации устройства (статуса линии, времени, занятого места на карте памяти и т.д.) используется HD47780-совместимый LCD дисплей, который подключается к микроконтроллеру по 4-хбитному интерфейсу.

Для активизации четырехбитного режима надо программно сформировать сигналы управления согласно временным диаграммам, на рисунке 2.19.

По структуре они совпадают с диаграммой 8-ми разрядной шины за

исключением удвоенного числа импульсов "E". Линии связи проходят через старшие разряды шины данных DB4-DB7, младшие DB0-DB3 остаются не задействованными.

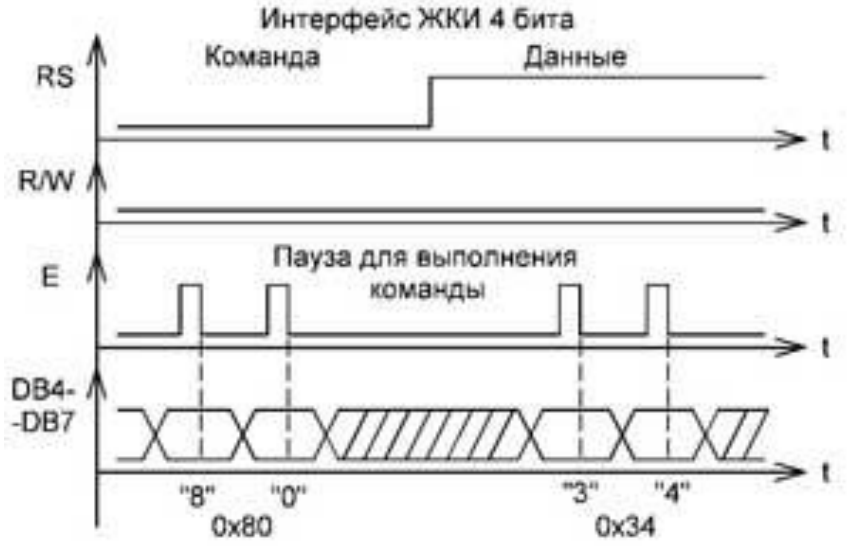

Рисунок 2.19 – Диаграмма сигналов управления для активизации четырехбитного режима.

Достоинство режима - малое число проводников, упрощение топологии печатной платы, экономия линий портов МК. Недостаток - пониженная скорость передачи данных в ЖКИ, так как информацию приходится передавать двумя порциями (нибблами или тетрадами) по 4 бита в каждой.

HD44780 имеет модификации исполнения с тремя основными вариантами зашивки знакогенератора:

а) латиница и европейские языки;

б) латиница и японские иероглифы;

в) латиница и кириллица.

Схема расположения выводов дисплея показаны на рисунке 2.20, их назначение приведены в таблице 2.11.

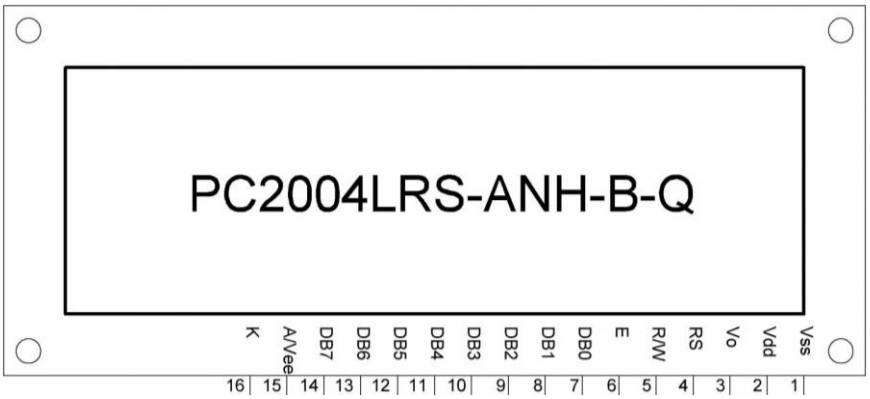

Рисунок 2.20 - Схема расположения выводов LCD дисплея.

Таблица 2.11 – назначение выводов LCD дисплея

Вывод ЖКИ	Обозначение	Функция
1	GND	Земля
2	VCC	Питание +5В
3	V_0	Напряжение фокусировки
4	RS	“0” запись команд, “1” запись данных
5	R/W	“0” запись в ЖКИ, “1” чтение из ЖКИ
6	E	Перепад уровня из “1” в “0” – ввод данных
7-14	DB0-DB7	Двунаправленная шина данных
15	A	Анод матрицы светодиодной подсветки
16	K	Катод матрицы светодиодной подсветки

Электрический интерфейс состоит из трех шин:

а) шина данных DB0-DB7;

б) шина управления RS, R/W, E;

в) шина питания VCC, GND, Vo, A, K.

2.2.1.11 Карта памяти MMC/SD

Данная система способна работать как в паре с персональным компьютером, так и без него, сохраняя при этом данные на карту памяти MMC или SD. Учитывая, что карты памяти SD/MMC поддерживают работу по протоколу SPI, подключение их к шине SPI микроконтроллера – очевидный выбор.

На рисунке 2.21 показаны контактные поверхности SDC и MMC. MMC имеет семь контактных площадок, а у SDC их девять, две из которых как бы добавлены к семи площадкам MMC. Три контакта у каждой карты заняты в качестве выводов источника питания, так что количество рабочих сигналов составляет соответственно четыре или шесть. Обмен данными между ведущим

контроллером и картой осуществляется в виде последовательной тактируемой передачи.

Расположение выводов на картах SD и MMC показано на рисунке 2.21.

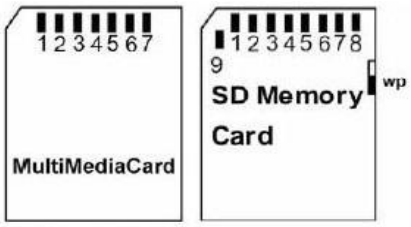

Рисунок 2.21 - Контактная поверхность SDC и MMC.

В таблицах 2.12, 2.13 описано назначение выводов карт памяти VVC/SD

Таблица 2.12 – Назначение контактов карт MMC.

Контакт	Режим MMC		Режим SPI	
	Обознач.	Функция	Обознач.	Функция
1	RSV	Зарезервированный контакт	CS	Сигнал выбора кристалла
2	CMD	Команда/Ответ	DI	Вход данных
3	VSS1	Земля	VSS1	Земля
4	VCC	Питание	VDD	Питание
5	CLK	Часы	SCLK	Часы
6	VSS2	Земля	VSS2	Земля
7	DAT	Передача данных	DO	Выход данных

Таблица 2.13 – Назначение контактов карт SDC

Контакт	Режим MMC		Режим SPI	
	Обознач.	Функция	Обознач.	Функция
1	CD/DAT3	Определение карты/ Передача данных [Bit3]	CS	Сигнал выбора кристалла
2	CMD	Команда/Ответ	DI	Вход данных
3	VSS1	Земля	VSS1	Земля
4	VCC	Питание	VDD	Питание
5	CLK	Часы	SCLK	Часы
6	VSS2	Земля	VSS2	Земля
7	DAT0	Передача данных [Bit0]	DO	Выход данных
8	DAT1	Передача данных [Bit1]	RSV	-
9	DAT2	Передача данных [Bit2]	RSV	-

2.2.2 Выбор программного обеспечения

Для полнофункциональной работы устройства необходимо разработать управляющую программу на языке Си.

Для языка Си характерны лаконичность, стандартный набор конструкций управления потоком выполнения, структур данных и обширный набор

операций. После компиляции каждому элементу программы соответствует небольшое число машинных команд, а использование базовых элементов языка не задействовало библиотеку времени выполнения.

Компиляторы Си разрабатываются сравнительно легко благодаря простоте языка и малому размеру стандартной библиотеки. Поэтому данный язык доступен на самых различных платформах

Си создавался с одной важной целью: сделать более простым написание больших программ с минимумом ошибок по правилам процедурного программирования, не добавляя на итоговый код программ лишних накладных расходов для компилятора, как это всегда делают языки очень высокого уровня С этой стороны Си имеет следующие важные особенности (рисунок 2.22).

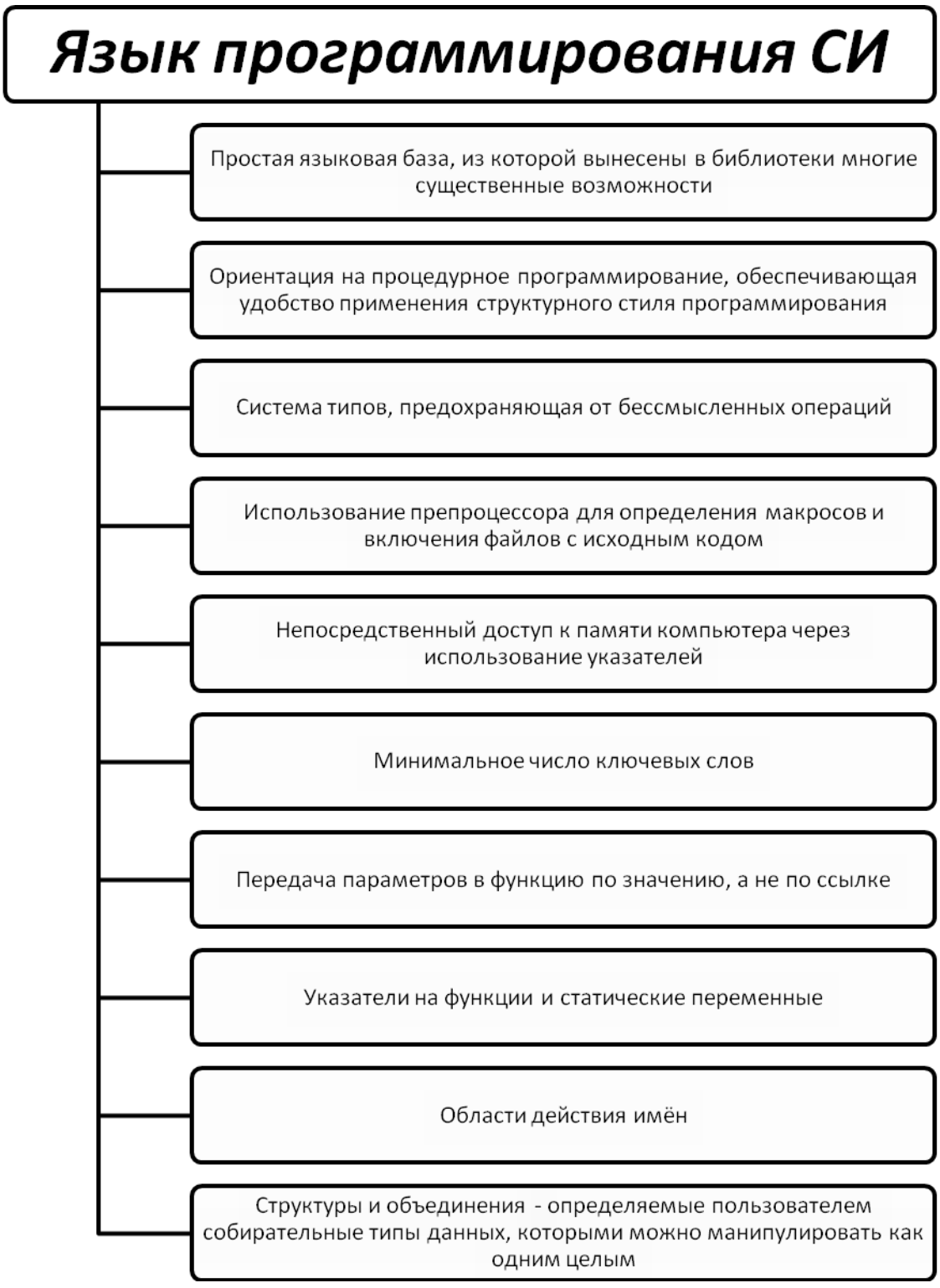

Рисунок 2.22 – Основные особенности языка программирования Си.

В Си есть три разных способа выделения памяти (классы памяти) для объектов (рисунок 2.23).

Рисунок 2.23 – Способы выделения памяти в языке программирования Си.

2.2.3 Оценка технических характеристик

2.2.3.1 Алгоритм Герцеля

Современные телефонные аппараты используют тональный набор номера DTMF, при котором каждая нажатая на клавиатуре телефона кнопка вызывает генерацию тонального сигнала, представленного сложением двух синусоидальных сигналов определенной частоты. В таблице 2.14 представлено соотношение кнопок телефонной клавиатуры и частот синусоидальных сигналов при тональном наборе номера.

Таблица 2.14 - Соотношение клавиш телефона и синусоидальных сигналов набора номера

Частота (Гц)	Клавиши телефона			
	1209	1336	1477	1633
697	1	2	3	A
770	4	5	6	B
852	7	8	9	C
941	*	0	#	B

Для того, чтобы узнать, какая цифра была введена, необходимо

определить интенсивность сигнала в определенный промежуток времени и одновременно с этим определить частоту сигнала.

При аппаратной реализации для этого возможно воспользоваться полосовыми фильтрами. При программной реализации эту задачу необходимо решить с помощью цифровых фильтров. Для измерения интенсивности сигнала на определенных частотах используется процедура, известная как алгоритм Герцеля, который позволяет определять частоту программно с большой точностью.

Алгоритм Гёрцеля - это специальная реализация дискретного преобразования Фурье (ДПФ) в форме рекурсивного фильтра. Данный алгоритм был предложен Джеральдом Герцелем в 1958 году. В отличие от быстрого преобразования Фурье, вычисляющего все частотные компоненты ДПФ, алгоритм Герцеля позволяет эффективно вычислить значение одного частотного компонента.

Для данного алгоритма требуется некоторые, предварительно вычисленные константы, и состоит он из двух частей: внутренняя часть, запускаемая для каждого отсчета и внешняя (финализация), запускаемая по окончанию каждого блока.

Основной параметр – размер блока. Чем больше блок, тем чувствительнее фильтр и тем меньше полоса пропускания. В реализуемом алгоритме используется блок размером N=206 выборок (аудио блок). Основной временной параметр в программном обеспечении для микроконтроллера - это частота дискретизации, генерируемая при помощи Таймера/счетчика 0 в режиме CTC. Код инициализации Таймера0 устанавливает делитель тактовой частоты CLK/8, при этом получаем частоту 1.3842 МГц. Прерывание Timer0 Compare Match устанавливает в регистре Output Compare 0 значение 171 каждые 5 раз или 172 в остальных случаях. (4×171+172)/5=172.8 – получим точно 8000 Гц. Отсюда минимальная длительность обнаружения сигнала: 206/8 кГц = 25.75 мс и полоса пропускания приблизительно 40 Гц.

Для каждой частоты необходимо вычислить коэффициенты *k* и *c*.

$$k = int\left(0.5 + f \times \frac{N}{S}\right); \quad (2.1)$$

$$c = 2 \times \cos\left(2 \times k \times \frac{pi}{N}\right); \quad (2.2)$$

где:

N- размер блока;

f – частота, для которой будет проводится измерение интенсивности;

S – частота дискретизации;

c – коэффициентом косинуса.

При реализации устройства используется арифметика с фиксированной точкой с 9-битными коэффициентами, интерпретируемыми как 1 бинарная цифра до десятичной точки и 8 цифр после. В таблице 2.15 представлены рассчитанные значения коэффициентов *c*, *k* для каждой частоты. Частота 440Гц – это частота сигналов вызова номера, занятого номера.

Таблица 2.15 - Значение коэффициентов *c*, *k* для каждой частоты

			Значение коэффициента *c* в формате с фиксированной точкой	
Частота (Гц)	*k*	*c*	decimal	hex
440	11	1.8885	483	1E3
697	18	1.7061	437	1B4
770	20	1.6393	420	1A3
852	22	1.5664	401	190
941	24	1.4876	381	17C
1209	31	1.1706	300	12B
1336	34	1.0176	260	104
1477	38	0.8004	205	0CC
1633	42	0.5714	146	092

Для внутренней части алгоритма Герцеля, для каждой из этих частот и для каждой выборки, мы должны вычислить:

y = d1 × c - d2 + s;

d2=d1;

d1=y;

где:

d1 и d2 – промежуточные результаты;

c – коэффициент косинуса (см. таблицу выше);

s – значение текущей выборки.

Первое вычисление по данному выражению называется «умножение и накопление». Конвертирование данного выражения в единицы с фиксированной точкой дает:

y=((d1×c)>>8)-d2+ss8,

где

ss8 – значение текущей выборки в 8-битном формате со знаком (это 8 старших значащих бита переменной sample16) и «>>» – оператор сдвига вправо. Переменные d1, d2, с должны быть 16-битными значениями со знаком и фактически будут массивами, потому что они нужны будут для каждой частоты. И в результате мы должны будем проделать следующее:

```
for (n=0;n<9;n++) {
y=((d1[n]*c[n])>>8)-d2[n]+ss8;
d2[n]=d1[n]; d1[n]=y;
}
```

В конце звукового блока мы должны будем произвести завершающие вычисления:

m = d1×d1 + d2×d2 - d1×d2×s

Результатом m является величина или интенсивность сигнала для данной частоты. Конвертируя в формат с фиксированной точкой, получим:

```
d1=d1>>8;
d2=d2>>8;
mag=d1*d1+d2*d2-d1*((d2*c[n])>>8);
d1=d2=0;
```

Алгоритм Герцеля позволяет определить интенсивность сигнала на заданной частоте, произведя одно умножение 16-битных значений, одно сложение, одно вычитание и одну 8-битную операцию сдвига вправо за одну выборку для обнаруженной частоты. В конце блока из 206 выборок проводятся

дополнительные вычисления: три 8-битных сдвига вправо, три перемножения 16-битных значений, одно сложение и одно вычитание. Благодаря встроенному в микроконтроллер модулю аппаратного умножения, это происходит очень быстро (перемножение двух 8-битных операндов за два машинных цикла).

При попытке организации данных вычислений с использованием языка Си, это приводит к умножению 16-битного числа (d1) на 16-битное число (d2) и результат 32-битное число. Но не только в этом расточительство, так как требуется четыре операции 8-битного умножения, теряются младшие значащие 8 бит.

```
if (nn switch (cc) {
case 0: d1[n]>>=8; break;
case 1: d2[n]>>=8; break;
case 2: mag =d1[n]*d1[n]; break;
case 3: mag+=d2[n]*d2[n]; break;
case 4: coef=c[n]; break;
case 5: coef*=d2[n]; coef>>=8; break;
case 6: coef*=d1[n]; break;
case 7: mag-=coef; break;
case 8: // results interpretation,
// omitted for brevity
case 9: d1[n]=d2[n]=0; break;
}
cc++; if (cc==10) cc=0,n++;
```

В итоге завершающая часть алгоритма отнимет менее 60 машинных циклов в результате чего будет получено равномерное распределение рабочей нагрузки, но при этом используется вдвое больше памяти (72 Байта).

Следующий шаг – оптимизация внутренней части алгоритма. С этой целью мы объединили два массива d1 и d2 в один массив d, упростив т.о. индексацию, и смогли перемещаться по массиву с помощью одной 8-битной операции инкремента (это также означает, что массив не может пересечь 256-байтную границу). Операции умножения в этой части алгоритма были реализованы с помощью ассемблера. Таким образом внутренняя часть алгоритма Герцеля заняла 42 машинных цикла, вместо 100.

2.2.3.2 Разработка кода для MMC/SD интерфейса

Существует много доступных библиотек для работы с MMC/SD картами памяти, но они не подходят для использования с разрабатываемым устройством, так как накладываются жесткие ограничения по времени. Поэтому был разработан собственный код для работы с картами памяти с применением техники распределения нагрузки (в ходе циклов оцифровки аудио сигнала) с применением множества статических конечных автоматов, контролируемых глобальными переменными.

При программной реализации интерфейса MMC/SD используется около 80% доступной памяти SRAM. Размер блока MMC/SD составляет 512 Байт и используются три таких блока памяти: два для двойного буфера записи/воспроизведения и один для таблицы указателей на начало кластеров для каждой записи.

Это ограничение памяти исключило возможность использования файловых систем, таких как FAT. В связи с этим было принято решение обратится к простой схеме, где карта памяти рассматривается как огромный кольцевой буфер.

При заполнении его старые записи будут заменяться новыми (перезаписываются).

Чтобы разместить 230 указателей в единственном индексном секторе размером 512 Байт, была разработана система адресации, при которой карту разделили на кластеры с 16 секторами (8 Кбайт или приблизительно 1 с записи) и каждый номер кластера имеет 16-битную ширину. Таким образом обеспечивается поддержка карт памяти объемом 512 МБайт.

Преимущество двойного буфера в том, что появляется возможность одновременно производить запись с телефонной линии и воспроизведение через встроенный динамик.

2.3 Алгоритм работы микропроцессорного устройства записи телефонных разговоров

После подачи питания система стабилизирует входное напряжение и проводит контроль его уровня. При достижении питания устройства нужного

уровня происходит запуск микроконтроллера.

При включении система в фоновом режиме инициализирует карту памяти. Микроконтроллер ожидает получения сигнала готовности от карты памяти. Если сигнал получен, микроконтроллер считывает дескриптор статуса карты и идентификационное поле для определения емкости карты и серийного номера. Затем считывается первый сектор и проверяется наличие «своей» индексной таблицы.

Если карта памяти используется на устройстве впервые, создается новая индексная таблица.

Устройство отслеживает состояние телефонной линии и отображает статус на дисплее: “Disconnected” – линия не подключена (отсутствует напряжение на линии), “Idle” – в ожидании, “In Use” – линия занята.

При помощи клавиатуры пользователь может выбрать запись для прослушивания (клавиши «влево», «впрво»). Центральная клавиша «Enter» может использоваться для прерывания воспроизведения, а также для выбора режима отслеживания: всегда, только когда линия в состоянии «трубка снята», выключен.

Алгоритм работы микропроцессорного устройства представлен на рисунке 2.24.

При обнаружении входящего звонка устройство переходит в режим записи, на второй строке индикатора отображается определенный номер. На первой строке отображается счетчик количества звонков и таймер.

Запись продолжается до момента освобождения линии, если звонок принят. Если звонок не принят, запись прекращается через несколько секунд в отсутствии активности.

При исходящем звонке запись начинается с момента поднятия трубки и прекращается при освобождении линии. В этом случае на первой строке индикатора отображаются цифры набранного номера, на второй – таймер и статус.

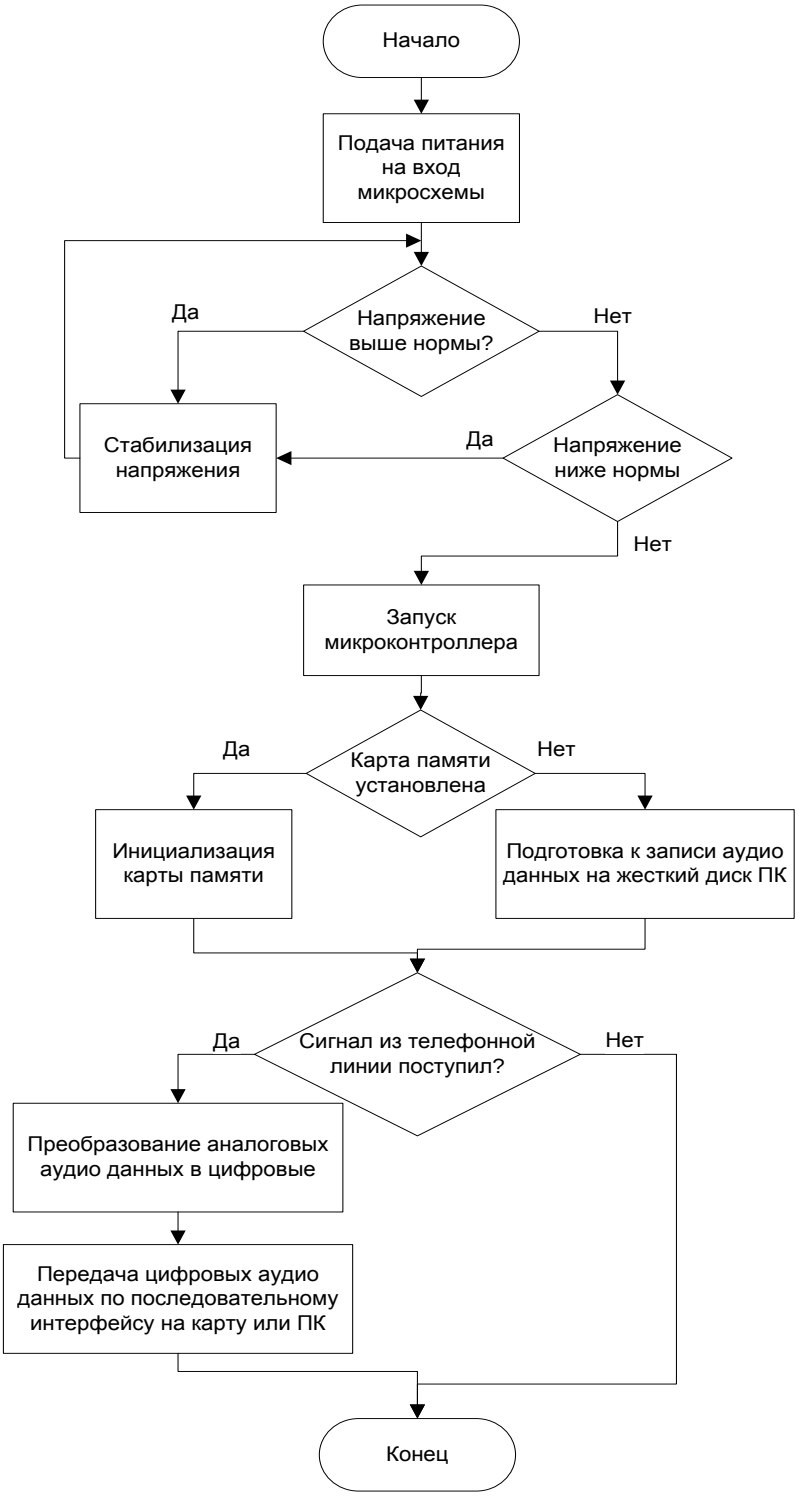

Рисунок 2.24 – Алгоритм работы микропроцессорного устройства записи телефонных разговоров.

2.4 Методы обеспечения качества МПУ

Понятие надежности наиболее близко к понятию качества, поэтому проблемы управления качеством непосредственно отражаются в представлении о надежности.

Под надежностью изделия понимается свойство сохранять свое качество при определенных условиях эксплуатации в течение заданного промежутка времени. Количественно надежность характеризуется рядом интервальных, интегральных и точечных показателей.

Надежность - объективное свойство изделия, ее можно измерить. С этой целью вводят понятия - «отказ», «вероятность безотказной работы», «интенсивность отказов» и другие. Понятия об отказе и безотказности являются одними из основных в теории надежности. Обычно под безотказностью понимают свойство изделий сохранять работоспособность в течение определенного интервала времени. Отказ - это полная или частичная утрата изделием работоспособности. Но и само понятие отказа оказалось непростым.

При расчете надежности в данной дипломной работе используем точечный показатель надежность – интенсивность отказов. Интенсивность отказов определяется как вероятность невосстанавливаемого отказа изделия в единицу времени после некоторого момента времени при условии, что до этого момента отказ не возникал. Будем полагать, что при выходе из строя хотя бы одного элемента, вся схема перестает работать. Тогда при расчете надежности условно можно представить все устройство в виде последовательно соединенных блоков.

Расчет показателей надежности по ориентировочному методу для разрабатываемого микропроцессорного устройства представлены в таблице 2.16.

Таблица 2.17 - Результаты ориентировочного расчета показателей надежности устройства записи телефонных разговоров

Компонент	Кол-во	Интенсивность отказов, $\lambda_{0i} \cdot 10^{-6}$	Интенсивность отказов группы $\lambda_{\Sigma} \cdot 10^{-6}$
Микросхемы аналоговые и цифровые	9	0,01	0,9
Резистор постоянный	49	0,5	24,5
Резистор переменный	1	1,7	1,7
Конденсатор неполярный	15	0,4	6
Конденсатор полярный	10	1,4	14
Диод	6	0,02	0,12
Транзистор	3	0,5	1,5

Трансформатор	1	0,5	0,5
Разъем	8	0,05	0,4

$\lambda_{\Sigma} = 48,81$

Вероятность безотказной работы на 1000 часов:

$P_{изд} = exp(-t/T_o) = 0,9523$, вероятность безотказной работы 95%.

Наработка на отказ:

$T_1 0 = 1/(= 1/(48,81 * [10]\ ^{\uparrow}(-6) =)$ 20487, наработка на отказ 20 тысяч часов.

Принципиальная схема микропроцессорного устройства записи телефонных разговоров представлена в Приложении А.

3 Конструкторская часть. Реализация микропроцессорного устройства записи телефонных разговоров

3.1 Конструкторско-техническая документация устройства

3.1.1 Описание процесса компоновки модулей МПУ

Появление печатных плат (ПП) в их современном виде совпадает с началом использования полупроводниковых приборов в качестве элементной базы электроники. Переход на печатный монтаж даже на уровне одно- и двухсторонние плат стал в свое время важнейшим этапом в развитии конструирования и технологии электронной аппаратуры.

Разработка очередных поколений элементной базы (интегральная, затем функциональная микроэлектроника), ужесточение требований к электронным устройствам, потребовали развития техники печатного монтажа и привели к созданию многослойных печатных плат (МПП), появлению гибких, рельефных печатных плат.

Многообразие сфер применения электроники обусловило совместное существование различных типов печатных плат (рисунок 3.1).

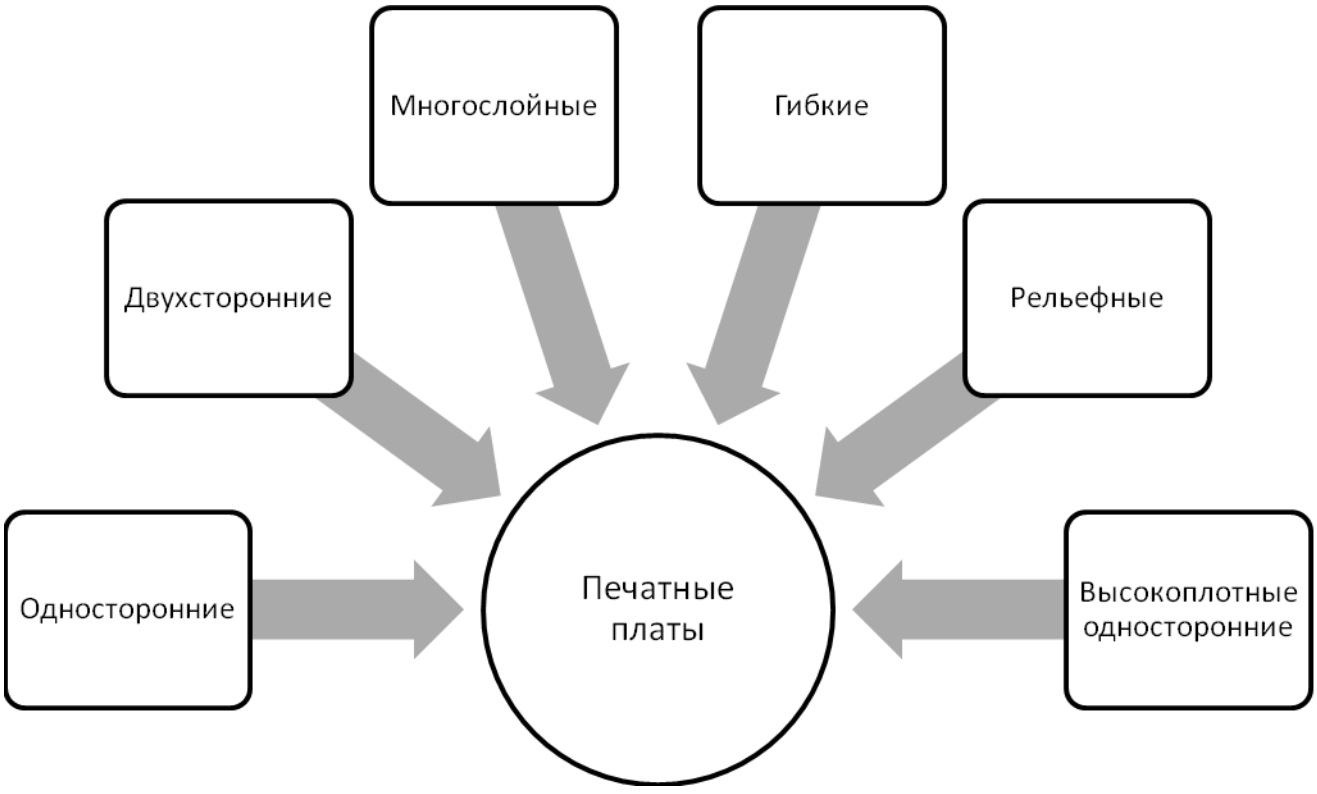

Рисунок 3.1 – Типы печатных плат.

Двухсторонние платы составляют в настоящее время примерно половину от всего объема выпуска плат. Столь значительное внимание разработчиков к этому виду плат объясняется своеобразным компромиссом между их относительно малой стоимостью и достаточно высокими возможностями.

На рисунке 3.2 представлена схема расположения слоев двухсторонней печатной платы.

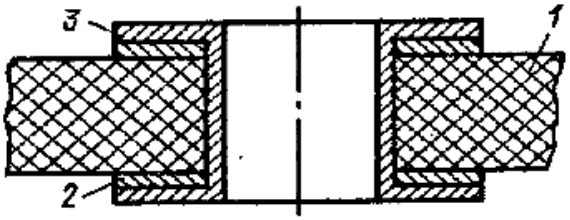

Рисунок 3.2 – Схема расположения слоев двухсторонней печатной платы (1 – диэлектрик; 2 – медная фольга; 3 – металлический слой).

В последние годы заметна тенденция резкого сокращения сроков проектирования новых изделий пи одновременно возрастание требованиям к их качественным характеристикам. Проектирование любого устройства включает в себя следующие этапы (рисунок 3.3).

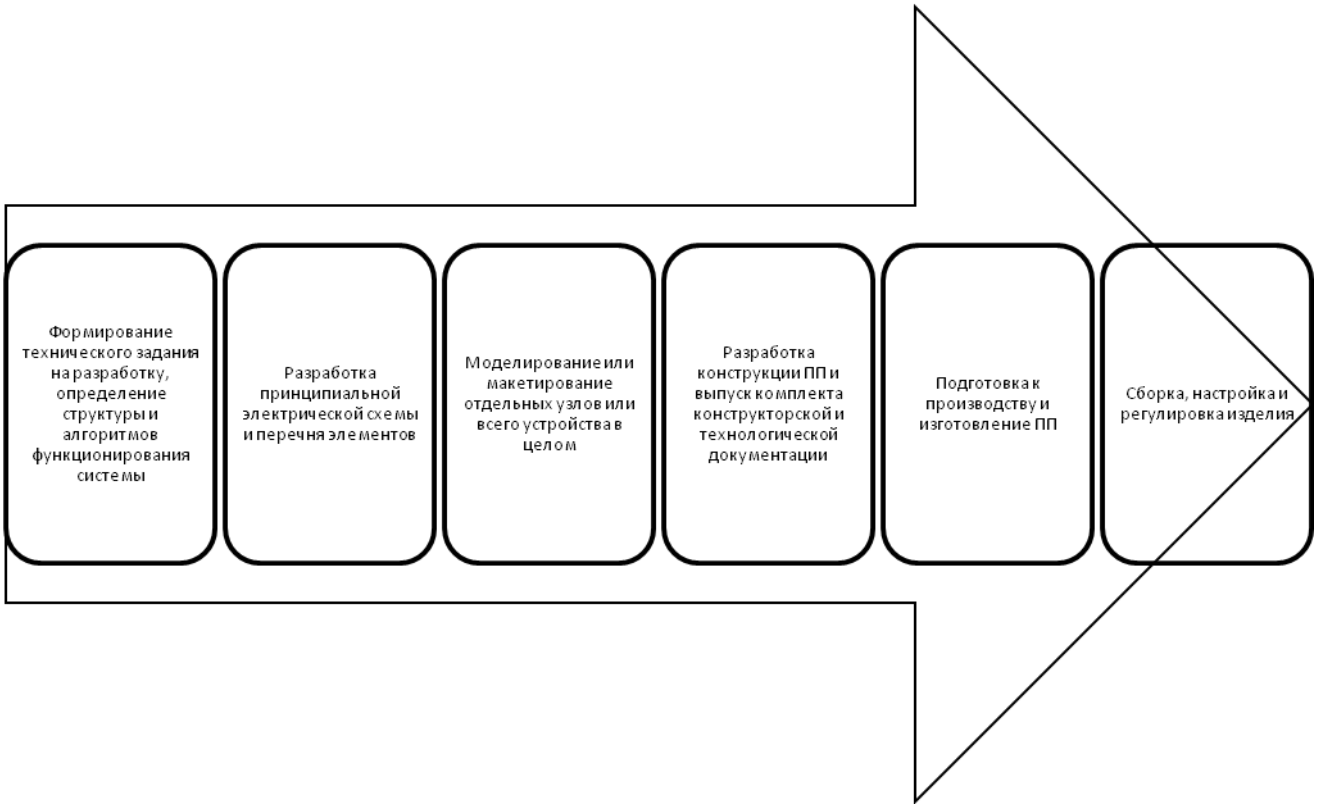

Рисунок 3.3 – Этапы проетирования устройства.

В условиях сжатых сроков проектирования и высоких требований к разрабатываемому устройству на помощь разработчикам пришли CAD системы. Данные системы позволяют реализовать следующие технические возможности (рисунок 3.4).

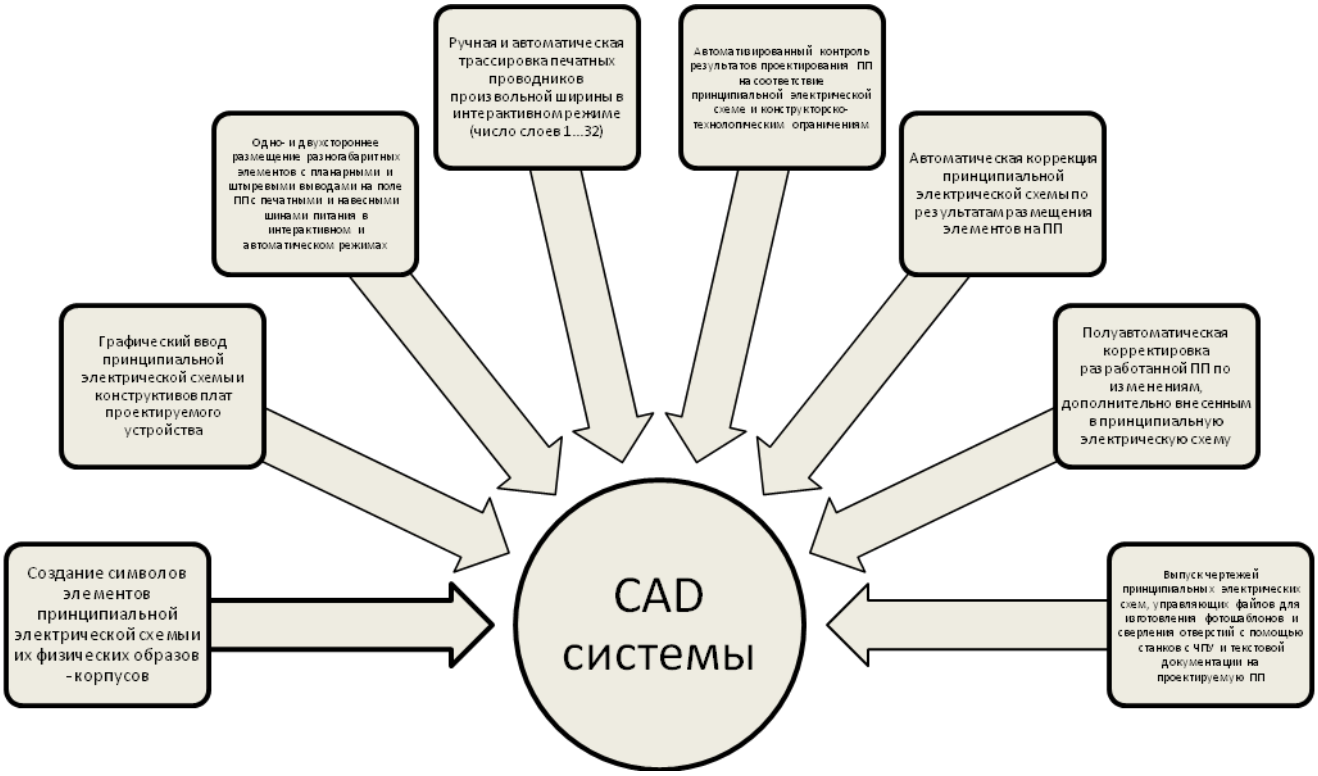

Рисунок 3.4 – Основные технические возможности CAD систем.

Одной из распространенных CAD-систем является DipTrace.

DipTrace – это профессиональная система автоматизированного проектирования и разработки принципиальных схем и печатных плат (САПР). В состав системы входят четыре компонента:

а) PCB Layout – менеджер для проектирования печатных плат с интерактивной автоматической трассировкой;

б) Schematic - менеджер проектирования принципиальных схем, которые впоследствии можно перевести в платы;

в) ComEdit - редактор корпуса для печатных плат;

г) SchemEdit - редактор компонентов, позволяющий рисование символов схемотехники в связке с корпусами.

САПР DipTrace удобна в использовании благодаря интуитивному интерфейсу и поддержки русского языка. Библиотеки программы содержат более 100000 компонентов.

Бесплатная версия программы имеет ограничение до 300 выводов на одной схеме.

С помощью САПР DipTrace разработана печатная плата устройства записи телефонных разговоров. На рисунке 3.5 представлен вид верхнего слоя печатной платы устройства.

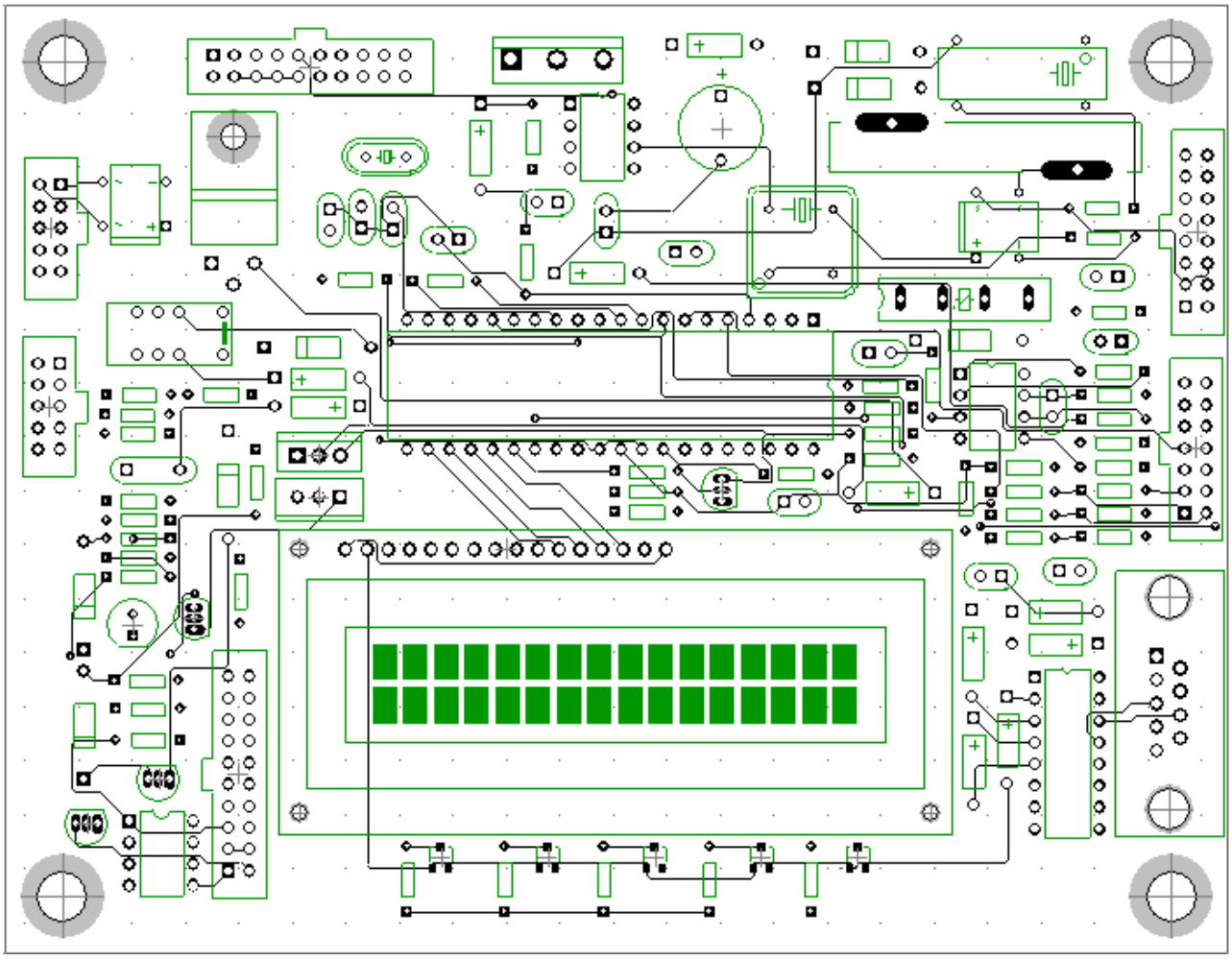

Рисунок 3.5– Верхний слой печатной платы устройства записи телефонных разговоров.

На рисунке 3.6 представлен вид нижнего слоя печатной платы устройства.

Существует несколько технологий изготовления печатных плат (рисунок 3.7).

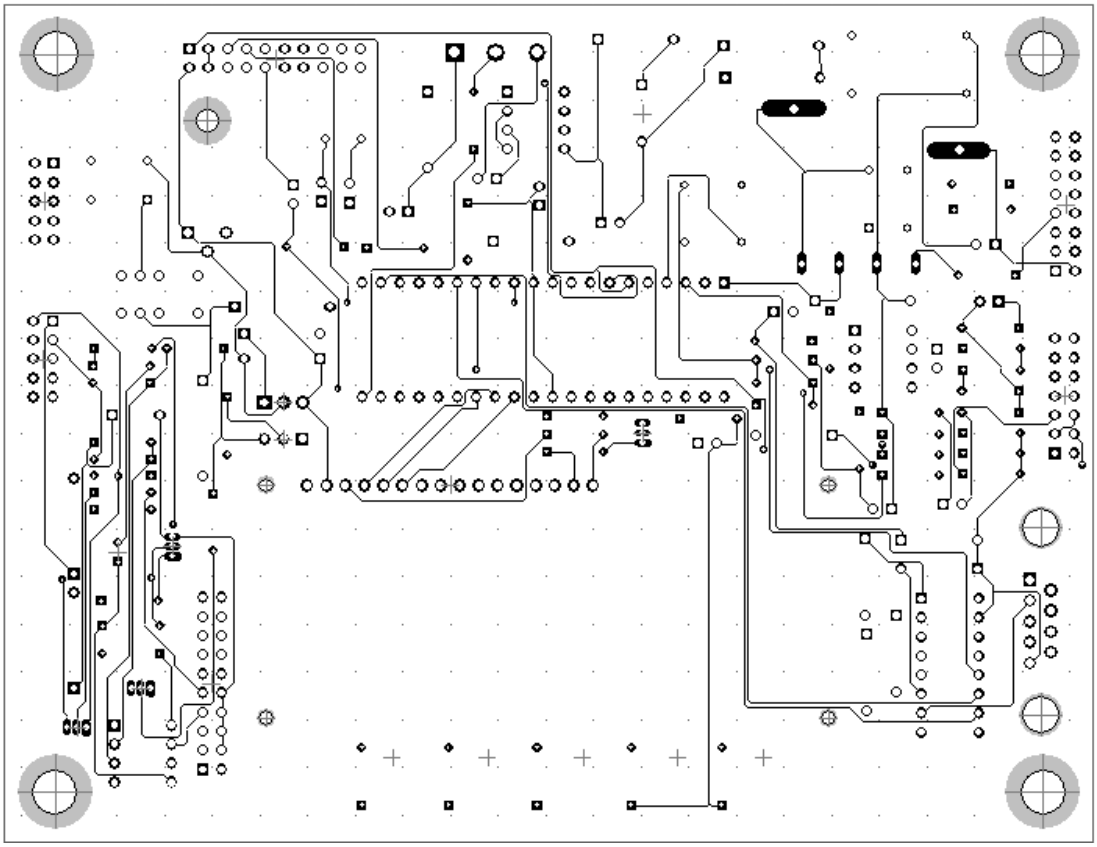

Рисунок 3.6 – Нижний слой печатной платы устройства записи телефонных разговоров.

Рисунок 3.7 - Технологии изготовления печатных плат.

Наибольшее распространение получила субтрактивная (классическая) технология, при которой лишние участки меди с поверхности платы удаляются путем химического травления. Помимо этого, возможно, удаление меди путем фрезерования или с использованием электроискровой установки. Однако эти способы не получили широкого распространения ни в радиолюбительской среде, ни в промышленности.

Процесс изготовления печатной платы можно условно разделить на пять основных этапов (рисунок 3.8).

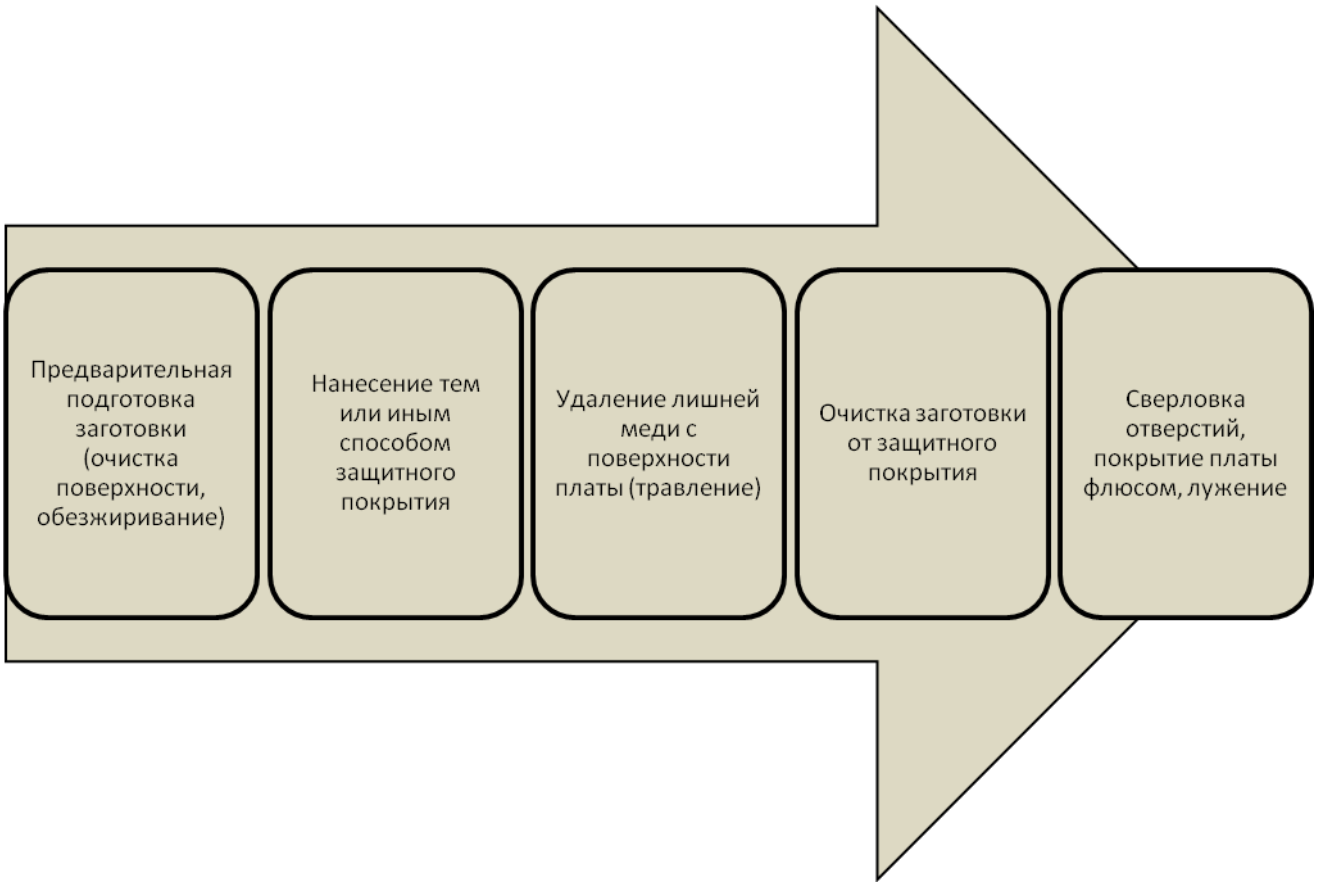

Рисунок 3.8 – Основные этапы изготовления печатной платы.

3.1.2 Описание особенностей программирования

Для того чтобы запрограммировать микроконтроллеры устройства записи телефонных разговоров использован программатор AVRISP mkII.

Программатор AVRISP mkII совместим с программой AVR Studio и способен программировать все 8-разрядные микроконтроллеры с RISC-архитектурой серии AVR, обладающие возможностью внутрисхемного программирования.

Программатор выполнен в виде блока с USB и ISP интерфейсами на боковых панелях. На рисунке 3.9 представлен внешний вид программатора AVRISP mkII.

Рисунок 3.9 - Внешний вид программатора AVRISP mkII.

Характеристики программатора AVRISP mkII:

а) совместим с AVR Studio 4.12 и более поздними версиями;

б) поддерживает все устройства серии AVR (ATmega32 и ATtiny45 используемые в устройстве) с возможностью внутрисхемного программирования (интерфейсом ISP);

в) поддержка программирования Flash и EEPROM;

г) поддержка программирования битов конфигурации (fuses) и битов блокировки;

д) возможность обновления прошивки для поддержки новых устройств;

е) поддержка микроконтроллеров с напряжением питания от 1.8В до 5.5В;

ж) регулируемая скорость программирования (от 50 Гц до 8 МГц частоты SCK);

з) совместим с USB 2.0 (Full-speed, 12 Мбит/с);

и) питание от шины USB, не требует внешнего источника питания;

к) защита интерфейсных линий программируемого устройства, в т.ч. защита от короткого замыкания.

Для индикации текущего состояния программатор имеет трехцветный светодиод. В таблице 3.1 приведены соотношение цвета светодиода и состояния программатора.

Дополнительный зелёный светодиод, расположенный внутри корпуса AVRISP mkII рядом с USB-разъёмом, индицирует обмен данными по шине USB.

Таблица 3.1 – Соотношение цвета светодиода и состояния программатора AVRISP mkII

Цвет светодиода	Описание
Красный	Состояние простоя – Питание программируемого микроконтроллера не обнаружено
Зелёный	Состояние простоя – Питание программируемого микроконтроллера обнаружено

Оранжевый	Занят – Идёт программирование
Оранжевый мигающий	Кабель подключен к программируемому устройству неправильно.
Красный мигающий	Короткое замыкание на программируемом устройстве
Красный/оранжевый мигающий	Режим обновления прошивки

Программное обеспечение для микроконтроллера ATmega32 написано на языке Си и скомпилировано с использованием AVR-GCC 3.4.3 и libc-1.2.3. Исходный код программа находится в Приложении Б.

3.2 Настройка и тестирование микропроцессорного устройства

Микропроцессорная система - это система реального времени, корректность ее функционирования зависит от времени выполнения отдельных программ и скорости работы аппаратуры. Поэтому система считается отлаженной после того, как рабочие программы правильно функционируют на действительной аппаратуре системы в реальных условиях.

Тенденция развития средств отладки микропроцессорных систем состоит в объединении свойств нескольких приборов в одном комплексе, в создании универсальных средств, пригодных для автономной отладки аппаратуры, генерации и автономной отладки программ и комплексной отладки системы. Эти средства позволяют вести разработку и отладку, постепенно усложняя аппаратуру и программы. При этом разработка, изготовление и отладка планируются поэтапно с нарастанием сложности. Новая, не отлаженная аппаратура и программа вводятся в создаваемую систему, присоединяются к проверенной ее части.

Если отладка программ ведется с использованием эмуляционного ОЗУ, а затем изготовляются микросхемы ПЗУ, то микропроцессорная система должна быть протестирована.

Средства отладки на последних этапах не должны влиять на правильность функционирования системы, вносить задержки, дополнительные нагрузки.

При комплексной отладке наряду с детерминированным используется статистическое тестирование, при котором МПС проверяется при изменении исходных переменных в соответствии со статистическими законами работы источников информации. Полнота контроля работоспособности проектируемой

системы возрастает за счет расширения диапазона возможных сочетаний переменных и соответствующих им логических маршрутов обработки информации.

Существуют пять основных приемов комплексной отладки микропроцессорной системы (рисунок 3.10).

Рисунок 3.10 - Основные приемы комплексной отладки микропроцессорной системы.

Комплексная отладка завершается приемосдаточными испытаниями, показывающими соответствие спроектированной системы техническому заданию. Для проведения комплексной отладки МПС используют логические анализаторы, а также оценочные и отладочные комплексы, комплексы развития микропроцессоров и диагностирования.

3.3 Описание эксплуатационной документации

После подключения устройства к телефонной линии, компьютеру и включения устройства в сеть, оно инициализирует карту памяти, отслеживает состояние телефонной линии и отображает его на ЖК-дисплее. При

поступлении входящего звонка устройство автоматически начинает запись на жесткий диск ПК с момента поднятия трубки, отображает номер абонента и состояние линии «IN USE» - линия занята. Помимо этого, на дисплее отображается количество поступивших звонков и продолжительность последнего звонка. При исходящем звонке запись начинается с момента поднятия трубки и прекращается при освобождении линии. При этом на дисплее отображается набранный номер, продолжительность звонка и статус линии.

При отключении устройства от ПК или отключении электроснабжения в офисе устройство автоматически переходит на резервный источник питания и запись производится на карту памяти.

При помощи встроенной клавиатуры пользователь может выбрать запись для прослушивания. Клавиши «Влево» и «Вправо» служат для перебора записей, центральная клавиша - «Ввод» служит для начала/прерывания прослушивания. Воспроизведение звука осуществляется через встроенный динамик.

Устройство способно распознавать несколько команд, подаваемых с помощью DTMF сигналов. Принятые команды подтверждаются пятью короткими звуковыми сигналами в линию.

В таблице 3.2 приведены основные технические характеристики микропроцессорного устройства записи телефонных разговоров.

Таблица 3.2 – Основные технические характеристики устройства

Наименование параметра	Значение
Напряжение	9В
Потребляемая мощность	0,5А
Емкость карты памяти, время записи	512 МБ, около 17 часов
Температура эксплуатации	0^{o}C/ +55^{o}C;
Влажность	10%-90% без образования конденсата

Заключение

В результате выполнения бакалаврской работы было спроектировано микропроцессорное устройство записи телефонных разговоров на базе

микроконтроллера ATmega32. Устройство обеспечивает автоматическую запись телефонных разговоров между сотрудником службы технической поддержки и клиентом из канала телефонной линии на жесткий диск компьютера по последовательному интерфейсу. При перебоях электропитания или выходе из строя компьютера устройство способно работать автономно от резервного источника питания, производить запись данных на карту памяти и воспроизводить аудио записи через встроенный динамик.

Был проведен сравнительный анализ алгоритмов работы, основных особенностей и характеристик различных комплексов записи телефонных разговоров. На основании проведенного анализа были сформулированы требования к проектируемому микропроцессорному устройству записи телефонных разговоров.

На этапе проектирования была построена структурная схема устройства, выбраны основные исполнительные узлы, алгоритмы преобразования аналоговых данных и алгоритмы управления памятью карты ММС. Был создан алгоритм работы устройства и рассчитаны его качественные характеристики

На этапе реализации устройства была подготовлена конструкторско-техническая документация устройства, описаны процессы компоновки модулей и особенностей программирования микропроцессорных устройств. Были описаны основные способы тестирования микропроцессорных устройств и подготовлена эксплуатационная документация МПУ записи телефонных разговоров.

На основании полученной информации было спроектировано микропроцессорное устройство, имеющее в своем составе ряд возможностей нескольких комплексов записи телефонных разговоров.

При выполнении бакалаврской работы были изучены материалы и получены практические навыки по проектированию микропроцессорных систем и разработке программного обеспечения к ним, разработке и изготовлению печатных плат, способов контроля качества изделий и безопасности жизнедеятельности.

Были приобретены навыки программирования в среде разработки AVR Studio, пакетом программ для разработки печатных плат – DipTrace.

Список литературы

1. Каган Б.М., Сташин В.В. Основы проектирования микропроцессорных устройств в автоматике. Москва, Энергоатомиздат, 1987. - 304 с.: ил.

2. Сташин В.В. и др. Проектирование цифровых устройств на однокристальных микроконтроллерах. – Москва: Энергоатомиздат, 1987. – 304 с.: ил.

3. Бродин В. Г., Шагурин И. И. Микроконтроллеры. Архитектура, програмирование, интерфейс. – М.: ЭКОМ, 1999. - 400с.: ил.

4. Медведев А.Д. Технология производства печатных плат. – Москва: Техносфера, 2005. - 360 с.: ил.

5. Гультяев А.К. MS Office Project 2007. Управление проектами - СПб.: КОРОНА-Век, 2008. – 480 с.: ил.

6. Шпак Ю.А. Программирование на языке Си для AVR и PIC микроконтроллеров. Киев: МК-Пресс, СПб Корона-Век, 2011. – 544.: ил.

7. http://microelectronic.at.ua/publ/avrstudio_nachalo/1-1-0-3

8. http://www.recordphone.ru/#37

9. http://dread-red.livejournal.com/18884.html

10. http://www.ats-telecom.ru/2_equipment/2_3_other/2_3_11_dogear/2_3_11_2.htm

11. http://www.modemspy.com/ru/

12. http://www.radioland.net.ua/contentid-397-page2.html

13. http://xreferat.ru/33/3252-1-harakteristiki-chislovyh-pokazateleiy-nerezervirovannogo-ustroiystva.html

14. http://controllersystems.com/programmi-sapr/programmirovanie-avr-avr-studio.html

15. http://www.adp.ru/icon/tr1.html

16. http://www.novosoft.by/Ency/rs-232.htm

17. http://bgd.alpud.ru/_private/Svet_pr/Raschet_6/IV_6_Raschet_isk.htm

18. http://slavapril.narod.ru/raschet_light.html

19. http://www.chipdip.ru/

20. http://www.kosmodrom.com.ua/avr/atmega32.php

21. http://www.atmel.com/Images/doc2503.pdf

22. http://www.gaw.ru/html.cgi/txt/ic/Atmel/micros/avr/ATtiny25_45_85.htm

23. http://ru.wikipedia.org

24. http://www.elkomponent.ru/catalog/tsepi_avr/programmator_avr_isp_1.html

25. http://dfe.karelia.ru/koi/posob/microcpu/proekt3.html

Приложение А

Поз. обозначение	Наименование	Кол.	Примечание
	Конденсаторы		
C1	B41142A7477M F 35v 470uF 5%	1	
C2	B41142A7477M F 16v 470uF 5%	1	
C3	A70D10M06ATE015 6V 100uF 10%	1	
C4, C24	ECAP 0.1 uF полярн.	2	
C5, C6	ECAP 22 pF полярн.	2	
C7, C22	B32652-A 473-J 47nF 1kVdc 5%	2	
C8	B32653A104J 0.1uF 250V 10%	1	
C9 - C12	CTL010G 10uf 35v 10%	4	
C13, C14	ECAP 0.1 uF неполярн.	2	
C15, C18, C25	K10-17 100pF 16V 5%	3	
C16	150D475X0006A2B 4.7 uF 6V 20%	1	
C19	TAJB 4.7uF 16V 5%	1	
C17	C1206M5V9B304 0.30mF 50V 20%	1	
C20	K10-17 100pF 16V 5%	1	
C21	B32924E325MCSD 2.2mF 50V 20%	1	
C23	B32653A104J 2.2uF 50V 10%		
	Микросхемы		
IC1	LM7805C	1	
IC2	LM2937z-3.3	1	

				ВУиТ.1.230101.65.09.12			
Изм. Лист	№ докум.	Подп.	Дата				
Разраб.	Курнаев В.Е			МПУ записи телефонных разговоров	Лит	Лист	Листов
Пров.	Калягина Н.В.					1	3
Н.контр.	Калягина				ИС – 501		

IC3	*ATmega32*	*1*	
IC4	*Max232*	*1*	
IC5	*LM386*	*1*	
IC6	*LM358N*	*1*	
IC7	*ATtiny45*	*1*	
IC8	*LM317*	*1*	
	Резисторы		
R1, R16, R33, R38, R42, R47	*RS09-R-30-1K 5%*	*6*	
R2	*RS09-R-30-150R 5%*	*1*	
R3	*CR-12 470R 5%*	*1*	
R4, R25	*MFR 12FTF52 39K 5%*	*2*	
R5, R41	*MCHMV LV5 3R3 J 5W 5%*	*2*	
R6-R8, R29	*CFR-12JT-52- 2K7 5%*	*4*	
R9-R11, R31, R50	*12CS-4K7-J 5 %*	*5*	
R12, R40	*CR-12 560R 5%*	*2*	
R13, R15, R48, R49	*R16K6-B 10K 5%*	*4*	
R14, R36, R43, R45, R46	*R16K6-B 220K 5%*	*5*	
R17	*CR-12 39R 5%*	*1*	
R18	*CR-12 100R 5%*	*1*	
R19	*CR-12 330K 5%*	*1*	
R20, R35	*CR-12 120K 5%*	*2*	
R21, R27	*CR-12 15K 5%*	*2*	
R22	*CR-12 47K 5%*	*1*	
R23	*CR-12 470K 5%*	*1*	
R24	*CR-12 150K 5%*	*1*	
R26	*CR-12 68K 5%*	*1*	

R28	*CR-12 1M 5%*	*1*	
R32	*CR-12 33K 5%*	*1*	
R34	*CR-12 82K 5%*	*1*	

R37	CR-12 1K5 5%	1	
R39	R1212N-A100K 10% Переменный	1	
	Диоды		
D1, D4	1N4004	2	
D2, D3	4V3-20	2	
D5, D7	1N4148	2	
D6	1N5158	1	
B1, B2	DF04M D70	2	
	Транзисторы		
Q1, Q3	BC557	2	
Q2	BC548	1	
Q4	BD330	1	
Y1	KXO-97T 11.0592 MHz	1	Тактовый генератор
	Фототранзистор		
OK1	PC817 5kV 50mA	1	
	Разъемы		
CN2, CN5	Разъемы ISP		
CN3	RS232 последовательный порт		
CN4	Разъем LCD		
CN6	Разъем MMC|SD		

Приложение Б

Листинг основной программы

```
#ifndef __AVR_ATmega32__
#error Hey, this code was only tested in an ATmega32
#endif
#include <avr/io.h>
#include <avr/sleep.h>
#include <avr/signal.h>
#include <avr/pgmspace.h>
#include <avr/interrupt.h>
#include "commands.h"
#define F_CPU 11059200L
#include <avr/delay.h>
#define HAVE_LCD 1
#ifdef HAVE_LCD
#include "lcd.h"
#endif // HAVE_LCD
#define RTS_DDR DDRD
#define RTS_PORT PORTD
#define RTS_BIT PD5
#define HOOK_DDR DDRB
#define HOOK_PORT PORTB
```

```
#define HOOK_BIT                    PB1
#ifdef HAVE_LCD
#define LCD_BACKLIGHT_DDR           DDRC
#define LCD_BACKLIGHT_PORT          PORTC
#define LCD_BACKLIGHT_BIT           PC4
#endif
#define HAVE_GOERTZEL
#ifdef HAVE_GOERTZEL
#include "goertzel.h"
#endif // HAVE_GOERTZEL
#define HAVE_MMC
#define USE_PROGMEM_CONSTANTS
#define ADPCM_USES_LONGS
#ifdef  USE_PROGMEM_CONSTANTS
#define _PROGMEM PROGMEM
#else
#define _PROGMEM
#endif
#define
ALLOW_SIMULTANEOUS_RECORD_AND_PLAYB
ACK
/* -------------------- Macro-Defined Constants --------- */

#define ADC_CHANNELS                 8
#define MAXTXFIFOLENGTH             16
#define MAXRXFIFOLENGTH              8
#define TRUE                         1
#define FALSE                        0
#define MAXDTMFSTRINGLENGTH         16
#define MAXVISIBLEDTMFSTRINGLENGTH 16
#define STEADY_COUNT                 7
#define MMC_CLUSTER_BITSHIFT         4
#define MMC_CLUSTER_SIZE
(1<<MMC_CLUSTER_BITSHIFT)
#define MMC_CLUSTER_SIZE_MASK
(MMC_CLUSTER_SIZE-1)
#define MMC_BLOCK_SIZE_BITSHIFT      9
#define MMC_BLOCK_SIZE
(1<<MMC_BLOCK_SIZE_BITSHIFT)
#define MMC_GO_IDLE_STATE            0
#define MMC_SEND_OP_COND             1
#define MMC_SEND_CSD                 9
#define MMC_SEND_CID                10
#define MMC_GO_INACTIVE_STATE       15
#define MMC_SET_BLOCKLEN            16
#define MMC_READ_SINGLE_BLOCK       17
#define MMC_WRITE_BLOCK             24
#define MMC_CRC_ON_OFF              59
#define MMC_STARTBLOCK_READ       0xFE
#define MMC_STARTBLOCK_WRITE      0xFE
#define MMC_DR_MASK               0x1F
#define MMC_DR_ACCEPT             0x05
/* Audio statuses */
#define AS_PLAYBACK_IN_PROGRESS      1
#define AS_RECORDING_IN_PROGRESS     2
#define AS_PLAYBACK_SHUTTING_DOWN    4
#define AS_RECORDING_SHUTTING_DOWN   8
#define DEFERRED_INDEX_WRITE      0x80
/* ---------------- Macro-Defined Functions ---------------- */
#define nop()    asm __volatile__("nop")
/* -------- Global Variables and Declarations ---------- */
unsigned char txfifo[MAXTXFIFOLENGTH];
volatile unsigned char fifo_head,fifo_tail;
unsigned char rxfifo[MAXRXFIFOLENGTH];
volatile unsigned char rxhead,rxtail;
unsigned short count=AUDIO_BLOCK_LENGTH;
#ifdef USE_ADPCM_COMPRESSION

short stepsizeTable[89] _PROGMEM = {
   7, 8, 9, 10, 11, 12, 13, 14, 16, 17,
   19, 21, 23, 25, 28, 31, 34, 37, 41, 45,
   50, 55, 60, 66, 73, 80, 88, 97, 107, 118,
   130, 143, 157, 173, 190, 209, 230, 253, 279, 307,
   337, 371, 408, 449, 494, 544, 598, 658, 724, 796,
   876, 963, 1060, 1166, 1282, 1411, 1552, 1707, 1878,
2066,
   2272, 2499, 2749, 3024, 3327, 3660, 4026, 4428, 4871,
5358,
   5894, 6484, 7132, 7845, 8630, 9493, 10442, 11487,
12635, 13899,
   15289, 16818, 18500, 20350, 22385, 24623, 27086,
29794, 32767};

char indexTable[16] _PROGMEM = {
   -1, -1, -1, -1, 2, 4, 6, 8,
   -1, -1, -1, -1, 2, 4, 6, 8,};

#ifdef ADPCM_USES_LONGS
long inpred;
char idx;
unsigned char pending;
#else
short inpred;
char idx;
unsigned char pending;
#endif
#endif
enum hook_status_t {
   HookUnknown, OnHook, OffHook, Disconnected};
#ifdef HAVE_LCD
//                          1234567890123456
char PROGMEM       on_hook_msg[]="IDLE
%h:%m:%s";
char PROGMEM     off_hook_msg[]="IN USE    %T%t";
char PROGMEM     disconn_msg[]="DISCONNECTED
";
char PROGMEM    ring_number_msg[]="RING #%r
%T%t";
char PROGMEM  incoming_call_msg[]="INCOMING ";
char PROGMEM
tens[]="00000000001111111111222222222333333333
4444444444"
"55555555556666666666777777777788888888899999
99999";
prog_char *main_msg;
unsigned char msg_index;
#endif // HAVE_LCD
unsigned char PROGMEM wav880[]={

0x80,0xAD,0xC6,0xBE,0x9A,0x6A,0x43,0x39,0x4F,0x0
0 };
const prog_char *wavp,*wavp0;
unsigned char wav_cnt,wav_rep,wav_mask,wav_rep0;
volatile unsigned char steady;
volatile enum hook_status_t
last_hook,last_reported_hook;
```

```
volatile unsigned char ring_count;
volatile unsigned char has_sample;
unsigned char last_pwm=0;
unsigned char age=0;
unsigned char ring_age=255;
static unsigned char digit_age;

#ifdef HAVE_LCD
unsigned char l1;
unsigned char gchrs[4],gchrs_index;
#endif // HAVE_LCD
unsigned short macroframe;
unsigned char  microframe;
volatile unsigned char now_hour,now_min,now_sec;
volatile unsigned char
now_day=1,now_month=1,now_year;
unsigned char PROGMEM mdays[]={
31,28,31,30,31,30,31,31,30,31,30,31};

unsigned char timer_hour,timer_min,timer_sec;
enum timer_status_t {
   TimerStopped, TimerRunning
} timer_status;
enum call_status_t {
   CallIdle, CallIncoming, CallOutgoing
} call_status=CallIdle;

char dtmf_string[MAXDTMFSTRINGLENGTH];
unsigned char dtmf_msg_pos,dtmf_string_length;
unsigned char dialing_digits;
char dtmf_digit;
unsigned char
ringlike,last_ringdet,ringdet_his,ringdet_los,ringdet_mids;
unsigned char ringdet_quell;

unsigned char *mmc_ptr;
unsigned short mmc_ptr_count;
unsigned char mmc_read_index_state;
unsigned char mmc_read_state;
unsigned char mmc_write_index_state;
unsigned char mmc_write_state;
unsigned char has_index,has_index_count;
unsigned char  buffer_a[512],buffer_b[512];
unsigned short
mmc_write_cluster,mmc_read_cluster,mmc_read_stop
_at;
unsigned char
mmc_write_cluster_sector,mmc_read_cluster_sector;
unsigned char
mmc_write_buf_state,mmc_read_buf_state;
unsigned char audio_status;
unsigned char *audio_buf,*io_buf;
unsigned short audio_cnt;
unsigned char  current_playback_call;
unsigned char
call_number[MAXDTMFSTRINGLENGTH];
unsigned char
call_year,call_month,call_day,call_hour,call_min,call_
sec;
unsigned char call_number_already_copied;

#endif // HAVE_MMC
#ifdef DO_HEXDUMP
unsigned char cmd_cnt,pending_cmd;
unsigned char monitor_mode;
unsigned char gci;
unsigned char kbd_read_state;
unsigned char kbd_key,kbd_has_key,kbd_last_key;
unsigned char PROGMEM
kbd_tbl[]="RDEEUUULLLL";

#ifdef HAVE_MMC
unsigned char spi_buf[8];
unsigned char spi_buf_tail,spi_buf_len,spi_ff;
unsigned char spi_has_reply,spi_reply,spi_autofinish;
unsigned char mmc_csd_index;
unsigned char mmc_init_state=6,mmc_retry;
unsigned char mmc_read_csd_state;
unsigned short mmc_last_cluster;
unsigned char mmc_psn[4];

#define MAXCALLS 230
#define MAGIC1   0x4A55
#define MAGIC2   0x4D45
struct mmc_index_t {
   unsigned short magic1;
   unsigned char  mmc_psn[4];
   unsigned short first_cluster;
   unsigned short last_cluster;
   unsigned short free;
   unsigned char  first_call;
   unsigned char  last_call;
   unsigned char  reserved1[12];
   unsigned short start[MAXCALLS];
   unsigned char  reserved2[510-26-2*MAXCALLS];
   unsigned short magic2;} mmc_index;


   unsigned short hexdump_count;
   unsigned char *hexdump_ptr;
   #endif // DO_HEXDUMP
   //#define SHOW_DEBUG_COUNT

   #ifdef SHOW_DEBUG_COUNT
   unsigned short dcnt;
   #define set_dcnt(x) (dcnt=(x))
   #else
   #define set_dcnt(x)
   #endif // SHOW_DEBUG_COUNT

   /* ---------------- Function Prototypes ------------------ */
   unsigned char inline has_rx_data(void);
   unsigned char recv(void);
   void inline xmit(unsigned char c);
   void inline do_xmit(void);
   void process_sample(void);
   void update_hook_status(enum hook_status_t hook);
   void lcd_timer_print(void);
   void dtmf_string_clear(void);
   void dtmf_string_putc(char c);
   void inline call_timer_reset(void);
   void inline call_timer_stop(void);
   void copy_call_number(void);
   unsigned char get_int2(char *p);
   void u8to2d(unsigned char val);
   void inline spi_start_xmit(void);
```

```
void inline spi_next_xmit(void);
void inline spi_buf_reset(void);
void inline spi_buf_putc(unsigned char c);
void inline spi_get_next(void);
void inline spi_finish(void);
void mmc_command(unsigned char cmd, unsigned
long arg);

/* ----------------- Main Program ----------------------- */
int main(void)
{   DDRD    |= _BV(PD1);      // UART TX is output
   RTS_DDR  |= _BV(RTS_BIT);  // CTS is output
   RTS_PORT |= _BV(RTS_BIT);  // We can't accept
data right now
DDRD  |= _BV(PD4) | _BV(PD7);
   DDRC=0;
   HOOK_DDR |= _BV(HOOK_BIT);
TCCR0 = _BV(WGM01) | 2;
OCR0   = 172;
TIMSK |= _BV(OCIE0);
TCCR1A = (0<<COM1A0) | (2<<COM1B0) |
(3<<WGM10);
TCCR1B = (2<<WGM12) | (1<<CS10);
OCR1A=255;
TCCR2 = _BV(WGM20) | _BV(WGM21)
          | _BV(COM21) | _BV(COM20)
          | _BV(CS20);

UCSRC  = 0x80|(3<<UCSZ0);
UCSRB  =  _BV(TXEN)
        |  _BV(RXEN)
        |  _BV(RXCIE);
UBRRL  = 5;
UBRRH  = 0;

ADCSRA = _BV(ADEN) | (6<<ADPS0) | _BV(ADIE)
| _BV(ADATE);
ADMUX  = 0 | _BV(ADLAR);
ADMUX |= _BV(REFS0) | _BV(REFS1);
SFIOR  = (SFIOR & 0x1F)|(3<<ADTS0);
SFIOR &= ~_BV(ACME);
 ACSR  &= ~_BV(ACBG);
ACSR  &= ~_BV(ACD);

#ifdef HAVE_MMC
PORTB |= _BV(PB7);
 DDRB  |= _BV(PB7) | _BV(PB5) | _BV(PB4); // SCK,
MOSI and SS are outputs
   SPCR  |= _BV(MSTR)
           |  _BV(SPE);
   audio_buf=buffer_a;
   io_buf=buffer_b;

#endif
          fifo_head=fifo_tail=0;
#ifdef HAVE_GOERTZEL
    d=q1; q=q2;
#endif // HAVE_GOERTZEL

#ifdef HAVE_LCD
   LCD_BACKLIGHT_DDR |=
_BV(LCD_BACKLIGHT_BIT);
   LCD_BACKLIGHT_PORT &=
~_BV(LCD_BACKLIGHT_BIT);
  lcd_init(LCD_DISP_ON);
#endif
   has_sample=FALSE;

   sei();
   for (;;) {    //set_sleep_mode(0);
             //sleep_mode();
      if (has_sample) {
         has_sample=FALSE;
         process_sample();}}
   return 0;}

#ifdef HAVE_MMC
void inline do_block_mmc_io(void)
{    if (mmc_read_state==3) {
       if (mmc_ptr_count) {
          //dtmf_string[4]='-';
          if (spi_has_reply) {
             //dtmf_string[4]='+';
             *mmc_ptr++=spi_reply;
             mmc_ptr_count--;
             spi_get_next(); }}
            else {//dtmf_string[1]='O';
          mmc_read_state++;
          spi_finish();}  } else if (mmc_write_state==3) {
       if (mmc_ptr_count) {
          //dtmf_string[10]='+';
          SPDR=*mmc_ptr++;
          mmc_ptr_count--;
       } else { mmc_write_state++;
          spi_buf_putc(0xFF);          spi_start_xmit();
          spi_autofinish=FALSE;   } }}

void stop_playback(void)
{    if (mmc_read_cluster)
       audio_status |=
AS_PLAYBACK_SHUTTING_DOWN;
#ifdef
ALLOW_SIMULTANEOUS_RECORD_AND_PLAY
BACK
   if (has_index & DEFERRED_INDEX_WRITE) {
      has_index &= ~DEFERRED_INDEX_WRITE;
      mmc_write_index_state=1;}
#endif //
ALLOW_SIMULTANEOUS_RECORD_AND_PLAY
BACK}

void start_playback(unsigned short from_cluster,
unsigned short to_cluster)
{   if (mmc_read_cluster) {
      mmc_read_cluster=from_cluster;
      mmc_read_stop_at=to_cluster;
      mmc_read_cluster_sector=0;
   } else {#ifdef
ALLOW_SIMULTANEOUS_RECORD_AND_PLAY
BACK
      if (mmc_write_cluster) {
         mmc_read_cluster=1;
         //dtmf_string[11]='C';} else
```

```
#endif //
ALLOW_SIMULTANEOUS_RECORD_AND_PLAY
BACK
      {if (!mmc_init_state) {
            mmc_read_buf_state=1;
            mmc_read_cluster=from_cluster;
            mmc_read_stop_at=to_cluster;
            //dtmf_string[11]='P';
            mmc_init_state=6;
            mmc_read_cluster_sector=0;
            audio_cnt=0;}}}}

void stop_recording(void)
{    if (audio_status &
AS_RECORDING_IN_PROGRESS)
      audio_status |=
AS_RECORDING_SHUTTING_DOWN;
   xmit(CMD_BLOCK_START);
   xmit(CMD_RECORDING_STOPPED);}

void start_recording(unsigned short cluster)
{ if (!mmc_init_state && !mmc_write_cluster) {
#ifndef
ALLOW_SIMULTANEOUS_RECORD_AND_PLAY
BACK
      if (mmc_read_cluster) {}
#endif //
ALLOW_SIMULTANEOUS_RECORD_AND_PLAY
BACK
      if (!cluster) cluster=mmc_index.last_cluster;
      mmc_write_cluster=cluster;
      mmc_index.start[mmc_index.last_call++]=cluster;
      if (mmc_index.last_call==MAXCALLS)
mmc_index.last_call=0;
      current_playback_call=mmc_index.last_call;
      if (mmc_index.last_call==mmc_index.first_call) {
         mmc_index.first_call++;
         if (mmc_index.first_call==MAXCALLS)
            mmc_index.first_call=0;}
      audio_status |=
AS_RECORDING_IN_PROGRESS;
      if (!mmc_read_cluster) {
        audio_cnt=0;}
      mmc_write_cluster_sector=0;}
   xmit(CMD_BLOCK_START);
   xmit(CMD_RECORDING_STARTED);
   call_number_already_copied=FALSE;
   call_year  = now_year;
   call_month = now_month;
   call_day   = now_day;
   call_hour  = now_hour;
   call_min   = now_min;
   call_sec   = now_sec;}

#endif // HAVE_MMC
#define MMC_MANY_IO_BLOCKS
void process_sample(void)
{#ifdef HAVE_MMC
#ifdef MMC_MANY_IO_BLOCKS
   do_block_mmc_io();
#endif // MMC_MANY_IO_BLOCKS
#endif // HAVE_MMC
  RTS_PORT &= ~_BV(RTS_BIT);
unsigned char pwm=0;
   if (has_rx_data()) {
      pwm=recv();
      if (cmd_cnt>0) {
         if (cmd_cnt==1) {
            if (pwm==CMD_BLOCK_START) {
               cmd_cnt=0;}
#ifdef HAVE_LCD
            else if
(pwm==CMD_LCD_BACKLIGHT_ON) {
               cmd_cnt=0;
               LCD_BACKLIGHT_PORT &=
~_BV(LCD_BACKLIGHT_BIT);
            } else if
(pwm==CMD_LCD_BACKLIGHT_OFF) {
               cmd_cnt=0;
               LCD_BACKLIGHT_PORT |=
_BV(LCD_BACKLIGHT_BIT);  }
#endif // HAVE_LCD
            else if (pwm==CMD_GO_OFF_HOOK) {
               cmd_cnt=0;
               HOOK_PORT |=  _BV(HOOK_BIT); //
Turn on hook relay
            } else if (pwm==CMD_GO_ON_HOOK) {
               cmd_cnt=0;
               HOOK_PORT &= ~_BV(HOOK_BIT); //
Turn off hook relay
            } else if (pwm==CMD_SET_TIME) {
               cmd_cnt++; pending_cmd=pwm;}
         } else if (pending_cmd==CMD_SET_TIME) {
            if (cmd_cnt==2) now_hour= pwm;
            if (cmd_cnt==3) now_min = pwm;
            if (cmd_cnt==4) now_sec = pwm;
            cmd_cnt++; if (cmd_cnt==5) cmd_cnt=0;
         } else if (pending_cmd==CMD_SET_DATE) {
            if (cmd_cnt==2) now_year = pwm;
            if (cmd_cnt==3) now_month = pwm;
            if (cmd_cnt==4) now_day   = pwm;
            cmd_cnt++; if (cmd_cnt==5) cmd_cnt=0;}
         pwm=last_pwm;
      } else {if (pwm==CMD_BLOCK_START) {
            cmd_cnt++; pwm=last_pwm;
         } else {age=0;
            if (pwm>248) pwm=248;
            last_pwm=pwm;}}
   } else {if (age>10) {
         age=0; pwm=0; last_pwm=0;
      } else {pwm=last_pwm; age++;}}
   if (wav_cnt) {
      pwm=pgm_read_byte(wavp++);
      if (!pwm) {
         wavp=wavp0;
         pwm=pgm_read_byte(wavp++);
         if (!--wav_rep) {
            wav_rep=wav_rep0;
            wav_cnt--;}}
      if (wav_cnt & wav_mask) pwm=128;}

   OCR1B = pwm;
#ifdef HAVE_MMC
   if (audio_status && !audio_cnt) {
      unsigned char *tmp=audio_buf;
      audio_buf=io_buf;
```

```
      io_buf=tmp;
      //dtmf_string[9]='*';
      if (mmc_read_buf_state) {
         //dtmf_string[5]='^';}
      if (mmc_write_cluster) {
         //dtmf_string[10]='r';
         mmc_init_state=1;
         mmc_write_buf_state=1;
         if (audio_status &
AS_RECORDING_SHUTTING_DOWN) {
            if (mmc_write_cluster_sector==15) {
               unsigned char i,*p=io_buf+
                  MMC_BLOCK_SIZE-7-
MAXDTMFSTRINGLENGTH;
               *p++=1; // Version info
               for
(i=0;i<MAXDTMFSTRINGLENGTH+6;i++)
                  *p++=call_number[i];}  }
      } else if (mmc_read_cluster) {
         //dtmf_string[9]='p';
         mmc_read_buf_state=10;
         //mmc_init_state=6;}}
   unsigned char *p=audio_buf+audio_cnt;
   if (audio_status &
AS_PLAYBACK_IN_PROGRESS) {
      OCR2=*p;
      //dtmf_string[9]='$';
   } else { //dtmf_string[9]=' ';
#endif // HAVE_MMC
      if (monitor_mode) {
         OCR2=(monitor_mode==2 ||
last_reported_hook==OffHook)?sample8:0;
      } else {OCR2=0;}
#ifdef HAVE_MMC       }
   if (audio_status &
AS_RECORDING_IN_PROGRESS) {
      *p=sample8;}
   if (audio_status) {audio_cnt++;
      audio_cnt&=(MMC_BLOCK_SIZE-1);}
#endif // HAVE_MMC

unsigned char out=sample8;
#ifdef USE_ADPCM_COMPRESSION
#ifdef ADPCM_USES_LONGS
unsigned char sign;
unsigned char delta;
   long step;
   long diff;
   long vpdiff;
#else // ADPCM_USES_LONGS
   unsigned char sign;
   unsigned char delta;
   short step;
   short diff;
   short vpdiff;
#endif // ADPCM_USES_LONGS
#endif // USE_ADPCM_COMPRESSION
   enum hook_status_t hook=HookUnknown;
#ifdef USE_ADPCM_COMPRESSION
#ifdef USE_PROGMEM_CONSTANTS
   step = pgm_read_word(stepsizeTable+idx);
#else
   step = stepsizeTable[idx];
#endif
          diff = sample16 - inpred;
          sign = (diff < 0) ? 8 : 0;
          if ( sign ) diff = (-diff);
delta = 0;
          vpdiff = (step >> 3);
          if ( diff >= step ) {
             delta = 4;
             diff -= step;
             vpdiff += step;}
          step >>= 1;
          if ( diff >= step  ) {
             delta |= 2;
             diff -= step;
             vpdiff += step;}
          step >>= 1;
          if ( diff >= step ) {
             delta |= 1;
             vpdiff += step;}
#ifdef ADPCM_USES_LONGS
          /* Step 3 - Update previous value */
          if ( sign )
            inpred -= vpdiff;
          else
            inpred += vpdiff;
/* Step 4 - Clamp previous value to 16 bits */
          if ( inpred > 32767 )
            inpred = 32767;
          else if ( inpred < -32768 )
            inpred = -32768;
#else
/* Step 3 - Update previous value */
   if (sign) {
      if (vpdiff>0 && inpred<vpdiff-32767)
         inpred=-32768;
      else  inpred-=vpdiff;
   } else {if (vpdiff>0 && inpred>32767-vpdiff)
         inpred=32767;
      else  inpred+=vpdiff;}
#endif // ADPCM_USES_LONGS
/* Step 5 - Assemble value, update index and step
values */
          delta |= sign;
#ifdef USE_PROGMEM_CONSTANTS
   idx += (char)pgm_read_byte(indexTable+delta);
#else
          idx += indexTable[delta];
#endif
          if ( idx < 0 )  idx = 0;
          if ( idx > 88 ) idx = 88;
/* Step 6 - Output value */
   if (pending & 128) {
      out = (pending<<4)|delta;
      pending=0;
      if (!--count) {
         count=AUDIO_BLOCK_LENGTH;
         xmit(CMD_BLOCK_START);
         xmit(CMD_ADPCM_STATE);
         xmit_short(inpred);
         xmit(idx);

/* On-hook/Off-hook/Disconnected detection logic &
debounce */
```

```
        if (ACSR & _BV(ACO)) {
           hook=Disconnected;
        } else {if (PINB & _BV(PB3)) {
              hook=OnHook;
           } else {hook=OffHook;}}
        if (inverts<0) inverts++;
        if (ring_age>0) {
           if (ring_age<255) ring_age--;
        } else {
           ring_age=255;
           call_timer_stop();
           call_status=CallIdle;
           stop_recording();
           ring_count=0; }
        if (hook != last_hook) {
           steady=0; last_hook=hook;
           if (inverts>=0) {
              inverts++;
              if (inverts==4) {
                 lcd_home();
                 ring_count++;
                 if (ring_count==1) {
                    if (call_status!=CallIncoming) {
                       call_timer_reset();
                       call_status=CallIncoming;
                       start_recording(0);}}
                 inverts=-70;
                 ring_age=250;}}
        } else {if (steady==STEADY_COUNT) {
              if (inverts>0) inverts=0;
              if (last_reported_hook!=hook) {
                 xmit(CMD_BLOCK_START);
                 xmit(hook);
                 steady=0;
                 last_reported_hook=hook;
                 if (hook==OffHook) {
                    if (call_status==CallIdle) {
                       call_timer_reset();
                       call_status=CallOutgoing;
                       dtmf_string_clear();
dialing_digits=TRUE;
                       digit_age=0;
                       start_recording(0); }}
                 if (hook==OnHook) {
                    call_timer_stop();
                    call_status=CallIdle;
                    stop_recording();
                    ring_count=0;
                    dtmf_string_clear();}}
           } else { steady++;}} }
     if (out==CMD_BLOCK_START) xmit(out);
     xmit(out);
   } else {pending = (delta&15)|128; }

#else   // USE_ADPCM_COMPRESSION
#ifdef HAVE_MMC
   spi_next_xmit();
#endif // HAVE_MMC

if (!ringdet_quell) {
      unsigned char ringdet=2;
      if (sample8>250) ringdet=1;
      if (sample8<5)   ringdet=0;
      if (ringdet==1) {
         ringdet_mids=0;
         if (last_ringdet) {
            ringdet_his++;
         } else {ringdet_his=0;
            if (ringdet_los>=60 || ringdet_los<=140) {
               ringlike++;
            } else {  ringlike--;}}
      } else if (ringdet==0) {
         ringdet_mids=0;
         if (last_ringdet) {
            ringdet_los=0;
            if (ringdet_his>=60 || ringdet_his<=140)
           {ringlike++;
            } else {ringlike--;}
         } else { ringdet_los++;}
      } else { ringdet_mids++;
         if (ringdet_mids>100)
ringdet_his=ringdet_los=ringlike=0;}
      last_ringdet=ringdet;  }

if (!--count) {  count=AUDIO_BLOCK_LENGTH;
#define DO_HOOK_LOGIC
#ifdef DO_HOOK_LOGIC
      if (ACSR & _BV(ACO)) {
         hook=Disconnected;
      } else {if (PINB & _BV(PB3)) {
            hook=OnHook;
         } else { hook=OffHook;}}
      if (hook != last_hook) {
         steady=0; last_hook=hook;
      } else {if (steady==STEADY_COUNT) {
            if (last_reported_hook!=hook) {
               xmit(CMD_BLOCK_START);
               xmit(hook);
               steady=0;
               last_reported_hook=hook;
               if (hook==OffHook) {
                  if (call_status==CallIdle) {
                     call_timer_reset();
call_status=CallOutgoing;
                     dtmf_string_clear();
dialing_digits=TRUE;
#ifdef HAVE_LCD
                     digit_age=0;
#endif // HAVE_LCD
#ifdef HAVE_MMC
                     start_recording(0);
#endif // HAVE_MMC
                  } else if (call_status==CallIncoming) {
                     ring_age=255;}
               } else if (hook==OnHook) {
                  call_timer_stop();
                  call_status=CallIdle;
#ifdef HAVE_MMC
                  stop_recording();
#endif // HAVE_MMC
                  dtmf_string_clear();
                  ring_age=255;}}
         } else {steady++;}}
#endif // DO_HOOK_LOGIC
      xmit(CMD_BLOCK_START);
      xmit(CMD_RAW_BLOCK);}
```

```
#endif
   do_xmit();
#ifdef HAVE_MMC
#ifdef MMC_MANY_IO_BLOCKS
   spi_next_xmit();
   do_block_mmc_io();
#endif // MMC_MANY_IO_BLOCKS
#endif // HAVE_MMC

#ifdef HAVE_GOERTZEL
      asm("lds    r30,(%0)+1  \n\t"
        "eor   r31,r31     \n\t"
        "sbrc  r30,7       \n\t"
        "com   r31         \n\t": : "m"(sample16) :
"r30","r31");

   asm("push r1\n\t");
   asm("lds  r28,%0     \n\t"
       "lds  r29,(%0)+1 \n\t"
        : : "m" (d) : "r28", "r29");

   goertzel_inner_16(GCOS0440);
   goertzel_inner_16(GCOS0697);
   goertzel_inner_16(GCOS0770);
   goertzel_inner_16(GCOS0852);
   goertzel_inner_16(GCOS0941);
   goertzel_inner_16(GCOS1209);
   goertzel_inner_00(GCOS1336);
   goertzel_inner_08(GCOS1477);
   goertzel_inner_08(GCOS1633);

   do_xmit();
   asm("pop r1\n\t");
#ifdef HAVE_MMC
   spi_next_xmit();
#ifdef MMC_MANY_IO_BLOCKS
    do_block_mmc_io();
#endif // MMC_MANY_IO_BLOCKS
#endif // HAVE_MMC

if (++gcount==GOERTZEL_BLOCK_SIZE) {
      gcount=cc=nn=0;
      volatile short *aux=d;
      d=q; q=aux; }
   if (nn<MAXGOERTZELITEMS) {
      unsigned char one=nn<<1;
      unsigned char two=one+1;
      switch (cc) {
         case 0: if (!nn) {
               at_least_one=FALSE; dtmf_digit=0; }
             q[one]>>=8; break;
         case 1: q[two]>>=8; break;
         case 2: mag =q[one]*q[one]; break;
         case 3: mag+=q[two]*q[two];break;
         case 4: coef=pgm_read_word(gcos+nn); break;
         case 5: coef*=q[two]; >>=8; break;
         case 6: coef*=q[one]; break;
         case 7: mag-=coef; break;
         case 8:if (nn>0) {
                  if (mag>10) {
                     at_least_one=TRUE; dtmf_frames++;
                     accum_flags |= ( (nn>4) ? 2 : 1 );
                     accum[nn-1] += mag;}
                  } else {break;
         case 9: q[one]=q[two]=0;
//#define GOERTZEL_DEBUG
#ifdef GOERTZEL_DEBUG
                  xmit(CMD_BLOCK_START);
                  xmit(CMD_GOERTZEL_DEBUG);
                  xmit(nn);
                  xmit_short(mag);
#endif // GOERTZEL_DEBUG
                  break;}
    } else {if (nn==MAXGOERTZELITEMS) {
          switch (cc) {
             case 0:
                if (!at_least_one) {
                   if (accum_flags==7 &&
dtmf_frames>1) {                    char  ri=-1,ci=-1;
                     unsigned char n;
                     unsigned short rv=0,cv=0;
                     for (n=0;n<4;n++) {
                    if (accum[n]>rv) rv=accum[n],ri=n;}
                     for (n=4;n<8;n++) {
                     if (accum[n]>cv) cv=accum[n],ci=n;}
            if (ri!=-1 && ci!=-1 && cv>100 && rv>100)

dtmf_digit=pgm_read_byte(dtmf_digits+(ri*4+ci-4));}
                   accum_flags=0; dtmf_frames=0;}
                if (accum_flags==3) {
                   accum_flags=7;
                } else {if (accum_flags!=7) {
                     unsigned char n;
                     for (n=0;n<8;n++) accum[n]=0;}}
                break;
             case 1:
                if (dtmf_digit) {
                   xmit(CMD_BLOCK_START);
                   xmit(CMD_DTMF_DETECTED);
                   xmit(dtmf_digit);
                   if (last_reported_hook==OnHook) {
                      if (dtmf_digit=='A' || dtmf_digit=='B')
{  call_status=CallIncoming; dtmf_msg_pos=0;
                         dtmf_string_clear();
call_timer_reset();
#ifdef HAVE_MMC
                         start_recording(0);
#endif // HAVE_MMC
                      } else {
                         if (call_status==CallIncoming) {
                            dtmf_msg_pos++;
                            if (dtmf_msg_pos>1) {
                               if (dtmf_digit!='C') {

dtmf_string_putc(dtmf_digit);
                                  if (dtmf_msg_pos==3 ||
                                     dtmf_msg_pos==7)
                                        dtmf_string_putc('-
');}}}}
                   } else {if (call_status==CallOutgoing) {
                         if (digit_age==200) {
                            dtmf_string_clear();
dialing_digits=FALSE;}
                         dtmf_string_putc(dtmf_digit);
                         if (dialing_digits) {
```

```
                        char c=dtmf_string[0];
                        if (c=='0') {
                           if (dtmf_string_length==1 ||
                              dtmf_string_length==4 ||
                              dtmf_string_length==7 ||
                              dtmf_string_length==11)
                                 dtmf_string_putc('-');
                        } else {if (c!='*' && c!='#') {
                              if (dtmf_string_length==4)
                                 dtmf_string_putc('-');}}}
                     digit_age=0; dtmf_digit=0;}}}
               else if (digit_age<200)
                  digit_age++;
               else if (digit_age==200) {
                  digit_age++;
#ifdef HAVE_MMC
                  copy_call_number();
#endif // HAVE_MMC    }
               break;
            case 2:
               if (dtmf_string[0]=='*' &&
dtmf_string[1]=='*') {
                  if (dtmf_string[2]=='0' &&
dtmf_string[3]=='1' &&
                     dtmf_string_length==10) {

now_hour=get_int2(dtmf_string+4);
                        now_min
=get_int2(dtmf_string+6);
                        now_sec =get_int2(dtmf_string+8);
                        dtmf_string_clear();
                  } else {if (dtmf_string[2]=='#' &&
dtmf_string[3]=='#' &&
                        dtmf_string_length==4) {
#ifdef HAVE_MMC
                           stop_recording();
#endif // HAVE_MMC
                           wavp=wavp0=wav880;
wav_rep=wav_rep0=75;
                           wav_cnt=10; wav_mask=1;
                           dtmf_string_clear();}}}
               break;
            case 3:
               if (ring_age>0) {
                  if (ring_age<255) ring_age--;
               } else {ring_age=255;
                  call_timer_stop();
                  call_status=CallIdle;
#ifdef HAVE_MMC
                  stop_recording();
#endif // HAVE_MMC}
               if (ringdet_quell>0) ringdet_quell--;
               if (ringlike==9)
               {if (last_reported_hook==OnHook) {
                     ring_count++;
                     if (ring_count==1) {
                        if (call_status==CallIdle) {
                           call_timer_reset();
                           call_status=CallIncoming;
#ifdef HAVE_MMC
                           start_recording(0);
#endif // HAVE_MMC}}
                     ring_age=250;
                     ringdet_quell=50;
                     ringlike=0;}}
               break;}}
#ifdef HAVE_MMC
      do_block_mmc_io();
#endif // HAVE_MMC}
   cc++;
   if (cc==10) {cc=0; nn++;}
#endif // HAVE_GOERTZEL

   if (count>45 && count<100) {
#ifdef HAVE_MMC
      //dtmf_string[0]=mmc_init_state+'0';
      if (mmc_init_state) {
         if (mmc_init_state==6) {
            if (count>88 && count<99) SPDR=0xFF;
            if (count==88) {

mmc_command(MMC_GO_IDLE_STATE,0);
               mmc_init_state--; }
         } else if (mmc_init_state==5) {
            if (spi_has_reply) {
               if (spi_reply!=0x01)
               {mmc_init_state++;
               } else {mmc_init_state--;}}
         } else if (mmc_init_state==4) {

mmc_command(MMC_SEND_OP_COND,0);
            mmc_init_state--;
            mmc_retry=0;
         } else if (mmc_init_state==3) {
            if (spi_has_reply) {
               if (spi_reply) {mmc_retry++;
                  if (mmc_retry==100) {
                     mmc_init_state=6;
                  } else {mmc_init_state++;}
               } else {]mmc_init_state--;}}
         } else if (mmc_init_state==2) {
            if (count==56) {

mmc_command(MMC_CRC_ON_OFF,0);
               mmc_init_state--;}
         } else if (mmc_init_state==1) {
            if (count==46) {

mmc_command(MMC_SET_BLOCKLEN,512);
               mmc_init_state--;
               if (mmc_write_buf_state) {
                  mmc_write_buf_state=2;
               } else if (mmc_read_buf_state) {
                  mmc_read_buf_state=2;
               } else {mmc_read_csd_state=1;}}}
      } else if (mmc_read_csd_state) {
         unsigned char state=mmc_read_csd_state&15;
         //dtmf_string[4]=mmc_read_csd_state+'0';
         if (state==1) {
            if (count==46) {
               mmc_read_csd_state++;
               unsigned char cmd;

cmd=mmc_read_csd_state<16?MMC_SEND_CSD:M
MC_SEND_CID;
               mmc_command(cmd,0);
```

```
            spi_autofinish=FALSE; mmc_retry=0; }
        } else if (state==2) {
           if (spi_has_reply) {
              if
(spi_reply==MMC_STARTBLOCK_READ) {
                 mmc_read_csd_state++;
                 mmc_csd_index=0;
                 spi_get_next();
              } else {
                 //dtmf_string[13]=spi_reply+'0';
                 mmc_retry++;
                 //dtmf_string[15]=mmc_retry+'0';
                 if (mmc_retry==20) {
                    mmc_read_csd_state&=0xF0;
                    mmc_read_csd_state|=1;
                    spi_finish();
                 } else {
                    spi_get_next();}}}
        } else if (state==3) {
           if (spi_has_reply) {
              buffer_a[mmc_csd_index++]=spi_reply;
              if (mmc_csd_index==16) {
                 spi_finish();
                 mmc_read_csd_state++;
              } else {spi_get_next();}
           } else {spi_get_next();}
        } else if (state==4) {
           if (mmc_read_csd_state<16) {
              mmc_read_csd_state=17;
              unsigned char
c_size_mult=buffer_a[10]>>7;
              c_size_mult |= (buffer_a[9]&0x3)<<1;
              if (c_size_mult<7) {

mmc_last_cluster=(buffer_a[6]&0x3)<<10;
                 mmc_last_cluster|=buffer_a[7]<<2;
                 mmc_last_cluster|=buffer_a[8]>>6;
                 mmc_last_cluster++;
                 if (c_size_mult<2) {
                    mmc_last_cluster>>=2-c_size_mult;
                 } else if (c_size_mult>2) {
                    mmc_last_cluster<<=c_size_mult-2;
}
              } else {mmc_last_cluster=0; }
           } else {
              mmc_read_csd_state=0;
              mmc_psn[0]=buffer_a[9];
              mmc_psn[1]=buffer_a[10];
              mmc_psn[2]=buffer_a[11];
              mmc_psn[3]=buffer_a[12];
              mmc_read_index_state=1;
              mmc_retry=0;
              //dtmf_string[1]='0';}}
     } else if (mmc_read_index_state && !has_index)
{
        if (mmc_read_index_state==1) {
           //if (count==46) {
              mmc_read_index_state++;

mmc_command(MMC_READ_SINGLE_BLOCK,0);
              spi_autofinish=FALSE;
              mmc_ptr=(unsigned char *)&mmc_index;
              mmc_read_state=1; }
     } else if (mmc_write_index_state) {
        if (mmc_write_index_state==1) {
           if (count==46) {
              mmc_write_index_state++;

mmc_command(MMC_WRITE_BLOCK,0);
              spi_autofinish=FALSE;
              mmc_ptr=(unsigned char *)&mmc_index;
              mmc_write_state=1;
              mmc_retry=0;}}}
#endif // HAVE_MMC
     // Keyboard scan
     if (count==99) {
        DDRA  |=  _BV(PA7);
        PORTA &= ~_BV(PA7); // Capacitor is pulled
to ground
        kbd_read_state=1;
        kbd_has_key=FALSE;
     } else if (kbd_read_state==1) {
        DDRA  &= ~_BV(PA7); // Port is now input
        kbd_read_state++;
     } else if (kbd_read_state==16) {
        kbd_read_state=0;
        if (!kbd_has_key) {
           kbd_last_key=kbd_key;
           kbd_key=0; }
     } else {if (PINA & _BV(PA7)) {
           if (!kbd_has_key) {

kbd_key=pgm_read_byte(kbd_tbl+kbd_read_state-2);
              //dtmf_string[0]=kbd_key;
              kbd_has_key=TRUE;}}
        kbd_read_state++;}
     if (kbd_last_key=='E') {
#ifdef HAVE_MMC
        if (mmc_read_cluster) {
           stop_playback();
           current_playback_call=mmc_index.last_call;
           if (current_playback_call>0)
current_playback_call--;
        } else {
#endif // HAVE_MMC
           monitor_mode++; if (monitor_mode==3)
monitor_mode=0;
#ifdef HAVE_MMC
           //dtmf_string[2]=monitor_mode+'0';
           mmc_read_cluster=mmc_write_cluster=0;
           //dtmf_string[0]=dtmf_string[1]=' ';
           mmc_read_buf_state=0;
           mmc_read_state=0;
           mmc_write_buf_state=0;
           mmc_write_state=0;
           audio_status=0;
           spi_finish();
           mmc_init_state=6;
           kbd_last_key=0; }
#endif // HAVE_MMC}
     else if (kbd_last_key=='U') {
        wavp=wavp0=wav880;
wav_rep=wav_rep0=64; wav_cnt=10; wav_mask=1;
        kbd_last_key=0;}
#ifdef DO_HEXDUMP
     else if (kbd_last_key=='D') {
```

```
          hexdump_count=16;
          kbd_last_key=0;
       } else if (kbd_last_key=='U') {
          hexdump_ptr=(unsigned char *)&buffer_a;
          kbd_last_key=0; }
#endif // DO_HEXDUMP
#ifdef HAVE_MMC
#ifdef DEBUG_RECORDING_TEST
       else if (kbd_last_key=='L') {
          start_playback(1,32);
          kbd_last_key=0;
       } else if (kbd_last_key=='R') {
          start_recording(1);
          kbd_last_key=0; }
#else
       else if (kbd_last_key=='L') {
          unsigned short start_cluster;
          if
(current_playback_call>mmc_index.first_call)
             current_playback_call--;

start_cluster=mmc_index.start[current_playback_call];
          if (start_cluster>0) {
             unsigned short end_cluster;
             if
(current_playback_call==mmc_index.last_call) {
                end_cluster=mmc_index.last_cluster;
             } else {
end_cluster=mmc_index.start[current_playback_call+1
];}
             start_playback(start_cluster,end_cluster);}
          kbd_last_key=0;
       } else if (kbd_last_key=='R') {
          unsigned short start_cluster;
          if (current_playback_call<mmc_index.last_call)
             current_playback_call++;

start_cluster=mmc_index.start[current_playback_call];
          if (start_cluster>0) {
             unsigned short end_cluster;
             if
(current_playback_call==mmc_index.last_call) {
                end_cluster=mmc_index.last_cluster;
             } else {
end_cluster=mmc_index.start[current_playback_call+1
];}
             start_playback(start_cluster,end_cluster); }
          kbd_last_key=0;
       }
#endif // DEBUG_RECORDING_TEST
#endif // HAVE_MMC
   } // if (count>45...)

#ifdef HAVE_MMC
   if (mmc_write_buf_state) {
      if (mmc_write_buf_state>1 &&
mmc_write_buf_state<10) {
          mmc_write_buf_state++;
       } else if (mmc_write_buf_state==10) {
          mmc_write_buf_state++;
          unsigned long mmc_address=(unsigned
long)mmc_write_cluster;
mmc_address<<=MMC_CLUSTER_BITSHIFT;
mmc_address+=mmc_write_cluster_sector;
mmc_address<<=MMC_BLOCK_SIZE_BITSHIFT;
mmc_command(MMC_WRITE_BLOCK,mmc_addres
s);
          spi_autofinish=FALSE;
          mmc_ptr=(unsigned char *)io_buf;
          mmc_write_state=1;
          mmc_retry=0;}
    } else if (mmc_read_buf_state) {
       if (mmc_read_buf_state>1 &&
mmc_read_buf_state<10) {
          mmc_read_buf_state++;
       } else if (mmc_read_buf_state==10) {
          mmc_read_buf_state++;
          unsigned long mmc_address=(unsigned
long)mmc_read_cluster;
mmc_address<<=MMC_CLUSTER_BITSHIFT;
mmc_address+=mmc_read_cluster_sector;
mmc_address<<=MMC_BLOCK_SIZE_BITSHIFT;
mmc_command(MMC_READ_SINGLE_BLOCK,mm
c_address);
          spi_autofinish=FALSE;
          mmc_ptr=(unsigned char *)io_buf;
          mmc_read_state=1;
          mmc_retry=0;}}

   if (mmc_read_state) {
      //dtmf_string[0]=mmc_read_state+'0';
      if (mmc_read_state==1) {
         if (spi_has_reply) {
            if (spi_reply) {
               mmc_retry++;
               if (mmc_retry==8) {
                  mmc_read_state=0;
                  spi_finish();
                  if (mmc_read_index_state)
                     mmc_read_index_state=1;
                  else if (mmc_read_buf_state==11)
                     mmc_read_buf_state=10;
               } else spi_get_next();
            } else {mmc_read_state++;
               spi_get_next();
               mmc_retry=0;}}
      } else if (mmc_read_state==2) {
         if (spi_has_reply) {
            if
(spi_reply==MMC_STARTBLOCK_READ) {
               mmc_ptr_count=MMC_BLOCK_SIZE;
               mmc_read_state++;
               //mmc_psn[3]=spi_reply;
               spi_get_next();
            } else {mmc_retry++;
               if (mmc_retry==8) {
                  mmc_read_state=0;
                  spi_finish();
                  if (mmc_read_index_state)
                     mmc_read_index_state=1;
                  else if (mmc_read_buf_state==11)
                     mmc_read_buf_state=10;
               } else spi_get_next();}}
      } else if (mmc_read_state==3) {
```

```
            if (mmc_ptr_count) {
               //dtmf_string[3]='-';
               if (spi_has_reply) {
                  //dtmf_string[4]='+';
                  *mmc_ptr++=spi_reply;
                  mmc_ptr_count--;
                  spi_get_next();}
            } else {mmc_read_state++;
            } else if (mmc_read_state==4) {
            spi_get_next();
            mmc_read_state++;
         } else if (mmc_read_state==5) {
            mmc_read_state=0;
            spi_finish();
            //dtmf_string[8]=mmc_read_buf_state+'@';
            if (!has_index) {
               has_index=1;
               mmc_read_index_state=0;
            } else if (mmc_read_buf_state) {

set_dcnt((mmc_read_cluster<<4)|mmc_read_cluster_se
ctor);
               mmc_read_cluster_sector++;

mmc_read_cluster_sector&=MMC_CLUSTER_SIZE_
MASK;
               mmc_read_buf_state=0;
               if (!mmc_read_cluster_sector) {
                  mmc_read_cluster++;
                  if
(mmc_read_cluster==mmc_read_stop_at) {
audio_status |=
AS_PLAYBACK_SHUTTING_DOWN;}}
               if (audio_status &
AS_PLAYBACK_SHUTTING_DOWN) {
                  mmc_read_cluster=0;
                  //dtmf_string[5]='!';
                  audio_status &= ~(
 AS_PLAYBACK_SHUTTING_DOWN |
 AS_PLAYBACK_IN_PROGRESS);
               } else {audio_status |=
AS_PLAYBACK_IN_PROGRESS;}}}
      } else if (mmc_write_state) {
         //dtmf_string[3]=mmc_write_state+'@';
         if (mmc_write_state==1) {
            if (spi_has_reply) {
               if (spi_reply) {
                  mmc_retry++;
                  if (mmc_retry==20) {
                     mmc_write_state=0;
                     spi_finish();
                     if (mmc_write_index_state)
                        mmc_write_index_state=1;
                     else if (mmc_write_buf_state==11)
                        mmc_write_buf_state=10;
                  } else spi_get_next();
               } else {mmc_retry=0;
                  mmc_write_state++;
                  spi_get_next();}}
         } else if (mmc_write_state==2) {
            mmc_write_state++;
            SPDR=MMC_STARTBLOCK_WRITE;
            mmc_ptr_count=MMC_BLOCK_SIZE;
         } else if (mmc_write_state==3) {
            if (mmc_ptr_count) {
               //dtmf_string[10]='+';
               SPDR=*mmc_ptr++;
               mmc_ptr_count--;
            } else {mmc_write_state++;
               spi_buf_putc(0xFF); // First byte of the
dummy 16-bit CRC
               spi_start_xmit();
               spi_autofinish=FALSE;   }
         } if (mmc_write_state==4) {
            if (spi_has_reply) {
               if ( (spi_reply & MMC_DR_MASK) !=
MMC_DR_ACCEPT) {
                  //mmc_psn[1]=spi_reply;
                  //dtmf_string[4]='F';
                  mmc_retry++;
                  if (mmc_retry==20) {
                     mmc_write_state=0;
                     spi_finish();
                     if (mmc_write_index_state)
                        mmc_write_index_state=1;
                     else if (mmc_write_buf_state==11)
                        mmc_write_buf_state=10;
                  } else spi_get_next();
               } else {mmc_write_state++;
                  spi_get_next();}   }
         } if (mmc_write_state==5) {
            if (spi_has_reply) {
               if (spi_reply) {
                  //dtmf_string[4]='S';
                  mmc_ptr=0;
                  mmc_write_state=0;
                  spi_finish();
                  if (mmc_write_index_state) {
                     has_index=4;
                     mmc_write_index_state=0;
                  } else if (mmc_write_buf_state) {
set_dcnt((mmc_write_cluster<<4)|mmc_write_cluster_
sector);
                     mmc_write_cluster_sector++;
mmc_write_cluster_sector&=MMC_CLUSTER_SIZE
_MASK;
                     if (!mmc_write_cluster_sector) {
                        mmc_write_cluster++;
         if (mmc_write_cluster==mmc_last_cluster)
                           mmc_write_cluster=1;
                        if (audio_status &
AS_RECORDING_SHUTTING_DOWN) {
mmc_index.last_cluster=mmc_write_cluster;
                           mmc_write_cluster=0;
                           audio_status &= ~(

AS_RECORDING_IN_PROGRESS |
AS_RECORDING_SHUTTING_DOWN);
#ifdef
ALLOW_SIMULTANEOUS_RECORD_AND_PLAY
BACK
                           if (!mmc_read_cluster)
                              mmc_write_index_state=1;
                           else
                              has_index |=
DEFERRED_INDEX_WRITE;
```

```
#endif }}
                mmc_write_buf_state=0;
#ifdef
ALLOW_SIMULTANEOUS_RECORD_AND_PLAY
BACK
                if (mmc_read_cluster) {
                   mmc_read_buf_state=10; }
#endif          } else spi_get_next();}}}
   if (has_index==2) {
      mmc_index.start[has_index_count++]=0;
      if (has_index_count==MAXCALLS) {
         has_index=3;
         mmc_write_index_state=1;}}
   if (count==39) {
      if (has_index==1) {
         if (mmc_index.magic1==MAGIC1 &&
mmc_index.magic2==MAGIC2 &&
            mmc_index.mmc_psn[0]==mmc_psn[0] &&
            mmc_index.mmc_psn[1]==mmc_psn[1] &&
            mmc_index.mmc_psn[2]==mmc_psn[2] &&
            mmc_index.mmc_psn[3]==mmc_psn[3]) {
               has_index=3;
               //dtmf_string[7]='R';

current_playback_call=mmc_index.last_call;
               if (current_playback_call>0)
current_playback_call--;
         } else {
            mmc_index.magic1=MAGIC1;
            mmc_index.magic2=MAGIC2;
            mmc_index.mmc_psn[0]=mmc_psn[0];
            mmc_index.mmc_psn[1]=mmc_psn[1];
            mmc_index.mmc_psn[2]=mmc_psn[2];
            mmc_index.mmc_psn[3]=mmc_psn[3];
            mmc_index.first_cluster=1;
            mmc_index.last_cluster=1;
            mmc_index.first_call=0;
            mmc_index.last_call=0;
            mmc_index.free=mmc_last_cluster-2;
            has_index++;
            has_index_count=0;
            //dtmf_string[1]='T';}} } else
#endif // HAVE_MMC
#ifdef HAVE_LCD
   if (count==40) {
#ifdef HAVE_MMC
#ifdef SHOW_DEBUG_COUNT
      unsigned short x=dcnt;
      unsigned char c;
      c=(x&15)+'0'; if (c>'9') c+=7;
      dtmf_string[15]=c; x>>=4;
      c=(x&15)+'0'; if (c>'9') c+=7;
      dtmf_string[14]=c; x>>=4;
      c=(x&15)+'0'; if (c>'9') c+=7;
      dtmf_string[13]=c; x>>=4;
      c=(x&15)+'0'; if (c>'9') c+=7;
      dtmf_string[12]=c;
#endif // SHOW_DEBUG_COUNT
#endif // HAVE_MMC
      msg_index=0;
      if (call_status==CallIncoming) {
         main_msg=(prog_char *)ring_number_msg;
         l1=1; lcd_gotoxy(0,0);
      } else if (call_status==CallIdle) {
         if (last_reported_hook==Disconnected) {
            main_msg=(prog_char *)disconn_msg; l1=1;
         } else {
            main_msg=(prog_char *)on_hook_msg;
ring_count=0; l1=1;  }
         lcd_gotoxy(0,0);
      } else if (call_status==CallOutgoing) {
         main_msg=(prog_char *)off_hook_msg; l1=0;
         lcd_gotoxy(0,1);}
   } else if (count>=20 && count<40) {
      unsigned char c=0;
      if (gci) {
         c=gchrs[gci-1]; gci++;}
      if (!c) { gci=0;
         c=pgm_read_byte(main_msg+msg_index);
         msg_index++; }
      if (c) {
         if (c=='%') {

c=pgm_read_byte(main_msg+msg_index++);
            gci=1;
            if (c=='h') { u8to2d(now_hour); gchrs[2]=0;
            } else if (c=='m') {
               u8to2d(now_min); gchrs[2]=0;
            } else if (c=='s') {
               u8to2d(now_sec); gchrs[2]=0;
            } else if (c=='T') {
               if (timer_hour<1) {
                  u8to2d(timer_min);
                  gchrs[2]='m'; gchrs[3]=0;
               } else {u8to2d(timer_hour);
                  gchrs[2]='h'; gchrs[3]=0; }
            } else if (c=='t') {
               if (timer_hour<1) {
                  u8to2d(timer_sec);
                  gchrs[2]='s'; gchrs[3]=0;
               } else {u8to2d(timer_min);
                  gchrs[2]='m'; gchrs[3]=0;}
            } else if (c=='r') {
               u8to2d(ring_count); gchrs[2]=0;}
         } else {  lcd_putc(c);}}
   } else if (count==19) {
      lcd_gotoxy(0,l1); msg_index=0;
   } else if (count>1 && count<19) {
      unsigned char c=dtmf_string[msg_index];
      if
(msg_index<MAXVISIBLEDTMFSTRINGLENGTH)
{
         msg_index++;
         lcd_putc(c);}}
#endif // HAVE_LCD
#ifdef DO_HEXDUMP
   if (hexdump_count) {
      hexdump_count--;
      unsigned char z,zz;
      z=*hexdump_ptr++;
      zz=(z>>4)+'0';
      if (zz>'9') zz+=7;
      xmit(zz);
      z&=15; z+='0';
      if (z>'9') z+=7;
      xmit(z);
```

```
      if ((hexdump_count&15)==0) { xmit(0xD);
xmit(0x0A); }}
#endif // DO_HEXDUMP
#ifdef HAVE_MMC
   do_block_mmc_io();
#endif // HAVE_MMC
#ifdef DEBUG_TIMING
   out=TCNT0;
#endif // DEBUG_TIMING

   if (out==CMD_BLOCK_START) xmit(out);
   xmit(out);
   do_xmit();}

/* ------------- Call Timer Auxiliary Routines --------- */
void inline call_timer_reset(void)
{  timer_hour=timer_min=timer_sec=0;
   timer_status=TimerRunning;}

void inline call_timer_stop(void)
{    timer_status=TimerStopped;}
/* ------ Serial Transmission Helper Routines ------- */
void inline xmit(unsigned char c)
{  txfifo[fifo_head++]=c;
   fifo_head &= 15;}

#ifdef HAVE_XMIT2
void inline xmit2(unsigned char c)
{   txfifo[fifo_head++]=c;
   fifo_head &= 15;}
#endif // HAVE_XMIT2

void inline do_xmit(void)
{ if (fifo_tail==fifo_head) return;
  if (!(UCSRA & _BV(UDRE))) return;
  UDR = txfifo[fifo_tail++];
  fifo_tail &= MAXTXFIFOLENGTH-1;}

unsigned char inline has_rx_data()
{    return rxhead!=rxtail;}
unsigned char recv(void)
{    char c; c=rxfifo[rxhead++];
   rxhead &= MAXRXFIFOLENGTH-1;
          return c;}

/* -------------------- SPI Routines ---------------------- */
#ifdef HAVE_MMC
void inline spi_start_xmit(void)
{   if (!spi_buf_len) return;
   PORTB &= ~_BV(PB4); // Assert CS (Chip Select)
   spi_buf_tail=1;
   spi_ff=0;
   spi_has_reply=FALSE;
   spi_autofinish = _BV(PB4);
   SPDR = spi_buf[0];}
void inline spi_buf_reset(void)
{    spi_buf_len=0;}
void inline spi_buf_putc(unsigned char c)
{    spi_buf[spi_buf_len++]=c;}

void inline spi_next_xmit(void)
{    if (!spi_buf_len) return;
   spi_reply=SPDR;
   if (spi_buf_tail<spi_buf_len) {
      SPDR = spi_buf[spi_buf_tail++];
   } else { if (spi_reply!=0xFF || spi_ff==8) {
         spi_has_reply=TRUE;
         spi_buf_len=0;
         PORTB |= spi_autofinish;  // Negate CS
      } else { if (spi_ff>0) SPDR = 0xFF;
         spi_ff++;}}}

void inline spi_get_next(void)
{   spi_ff=8; SPDR=0xFF;
   spi_buf_len=spi_buf_tail=1;
   spi_has_reply=FALSE;}

void inline spi_finish(void)
{   PORTB |= _BV(PB4);
   SPDR=0xFF; spi_has_reply=FALSE;}

void mmc_command(unsigned char cmd, unsigned
long arg)
{  unsigned char *p=(unsigned char *)&arg;
   spi_buf_reset();
   spi_buf[0]=0xFF;
   spi_buf[1]=cmd|0x40;
   spi_buf[2]=p[3];
   spi_buf[3]=p[2];
   spi_buf[4]=p[1];
   spi_buf[5]=p[0];
   spi_buf[6]=0x95;
   spi_buf_len=7;
   spi_start_xmit();}
#endif // HAVE_MMC

/* ----------- Number Conversion Routines ----------- */
#ifdef HAVE_LCD
void u8to2d(unsigned char val)
{  unsigned char c=pgm_read_byte(val+tens);
   gchrs[0]=c;
   c-='0';
   c=(c<<3)+(c<<1);
   c=val-c+'0';
   gchrs[1]=c;}

#endif // HAVE_LCD
unsigned char get_int2(char *p)
{    return (p[0]-'0')*10+(p[1]-'0');}
void dtmf_string_clear(void)
{  unsigned char n,*p;
   for
(p=dtmf_string,n=0;n<MAXDTMFSTRINGLENGTH;
n++) *p++=' ';
   dtmf_string_length=0;
//#define DEBUG
#ifdef DEBUG
   unsigned short x=dcnt;
   unsigned char c;
   c=(x&15)+'0'; if (c>'9') c+=7;
   dtmf_string[15]=c; x>>=4;
   c=(x&15)+'0'; if (c>'9') c+=7;
   dtmf_string[14]=c; x>>=4;
   c=(x&15)+'0'; if (c>'9') c+=7;
   dtmf_string[13]=c; x>>=4;
   c=(x&15)+'0'; if (c>'9') c+=7;
```

```
   dtmf_string[12]=c;
   unsigned char cc,i=3;
   for (n=0;n<4;n++) {
      cc=mmc_psn[n];
      c=(cc>>4)+'0'; if (c>'9') c+=7;
      dtmf_string[i++]=c;
      c=(cc&15)+'0'; if (c>'9') c+=7;
      dtmf_string[i++]=c;}
#endif}
#ifdef HAVE_MMC
void copy_call_number(void)
{   if (call_number_already_copied) return;
   unsigned char n,*p,*q;
   for
(p=dtmf_string,q=call_number,n=0;n<MAXDTMFST
RINGLENGTH;n++)
      *q++=*p++;
   call_number_already_copied=TRUE;}

#endif // HAVE_MMC
void dtmf_string_putc(char c)
{   if
(dtmf_string_length<MAXDTMFSTRINGLENGTH)
{
      dtmf_string[dtmf_string_length++]=c;}}
SIGNAL(SIG_ADC)
{   union {
      unsigned short as_u16;
      unsigned char as_u8[2];
   } adc_result;
   adc_result.as_u8[0]=ADCL;
   adc_result.as_u8[1]=ADCH;
   sample8=adc_result.as_u8[1];
   sample16=adc_result.as_u16-32768;}
SIGNAL(SIG_UART_RECV)
{   rxfifo[rxtail++] = UDR;
   rxtail &= MAXRXFIFOLENGTH-1;
   RTS_PORT |= _BV(RTS_BIT);}
SIGNAL(SIG_OUTPUT_COMPARE0)
{  has_sample=TRUE;
   microframe++;
   if (microframe==5) {
      microframe=0; OCR0=171;
   } else {OCR0=172;}
   if (macroframe==7999) {
      macroframe=0;
      if (now_sec==59) {
         now_sec=0;
         if (now_min==59) {
            now_min=0;
            if (now_hour==23) {
               now_hour=0;
               unsigned char
max=pgm_read_byte(now_month-1+mdays);
               if ( (now_month==2) &&
((now_year&3)==0) ) max++;
               if (now_day==max) {
                  now_day=1;
                  if (now_month==13) {
                     now_month=1;
                     if (now_year==99) {
                        now_year=0;
                     } else now_year++;
                  } else now_month++;
               } else now_day++;
            } else now_hour++;
         } else now_min++;
      } else now_sec++;
      if (timer_status==TimerRunning) {
         if (timer_sec==59) {
            timer_sec=0;
            if (timer_min==59) {
               timer_min=0;
               timer_hour++;
            } else timer_min++;
         } else timer_sec++;}
   } else {macroframe++;}}
```

Листинг программы управления памятью карты MMC

```
#include <time.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <termios.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include "commands.h"
#define BAUD_RATE    B115200
#define FALSE        0
#define TRUE         1

/* ----------- Macro-declared Constants ------------ */
#define MAXLINELENGTH 80
/* ----------- Declarations and Global Variables ------ */
char *device=NULL;
char timestamp=FALSE;
int raw_mode=0;
static char indexTable[16] = {
   -1, -1, -1, -1, 2, 4, 6, 8,
   -1, -1, -1, -1, 2, 4, 6, 8,};
static int stepsizeTable[89] = {
   7, 8, 9, 10, 11, 12, 13, 14, 16, 17,
   19, 21, 23, 25, 28, 31, 34, 37, 41, 45,
   50, 55, 60, 66, 73, 80, 88, 97, 107, 118,
   130, 143, 157, 173, 190, 209, 230, 253, 279, 307,
   337, 371, 408, 449, 494, 544, 598, 658, 724, 796,
   876, 963, 1060, 1166, 1282, 1411, 1552, 1707, 1878,
2066,
   2272, 2499, 2749, 3024, 3327, 3660, 4026, 4428,
4871, 5358,
   5894, 6484, 7132, 7845, 8630, 9493, 10442, 11487,
12635, 13899,
   15289, 16818, 18500, 20350, 22385, 24623, 27086,
29794, 32767};

int abuf_tail;
char      abuf[AUDIO_BLOCK_LENGTH];
short     sbuf[AUDIO_BLOCK_LENGTH*2];

struct adpcm_state {
   short  valprev;
   char   index;} state;
```

```
/* ------------ Function Prototypes ---------------------- */
void handle_values(char *buf);
void save_random_data(FILE *fp,char *buf);
void print_timestamp();
void audio_decode(char c);
void adpcm_decoder(char indata[], short outdata[], int
len,
           struct adpcm_state *state);

/* -------------------- Main Program ----------------------
*/
main(int argc, char **argv)
{ FILE *fp=NULL;
   int fd,res,a,n;
   struct termios oldtio,newtio;
   unsigned char c;
   int in_cmd,adpcm_cnt,in_sync,has_state=0;
   struct adpcm_state mystate;
   char *cmds=NULL;
   char *pbfilename=NULL;
   int recording_in_progress=TRUE;
   if (argc<2) {fprintf(stderr,"usage: read <device>\n");
      exit(1); }
   for (a=0,n=1;n<argc;n++) {
      if (argv[n][0]=='-') {
         switch(argv[n][1]) { case 'c':
               cmds=argv[n][2] ? argv[n]+2 : argv[n++];
               break;
            case 'p':
               pbfilename=argv[n][2] ? argv[n]+2 :
argv[n++]; break;} } else {switch(a)
           { case 0: device=argv[n]; break; }
         a++;}}

fd = open(device, O_RDWR | O_NOCTTY |
O_NONBLOCK);
   if (fd < 0) {
      fprintf(stderr,"Can't open %s\n",device);
      exit(1);}

   FILE *infp=fopen(pbfilename,"rb");
   unsigned char *outdata=NULL;
   int outsize,outindex=0;
   if (infp) {fseek(infp,0,SEEK_END);
      outsize=ftell(infp);
      fprintf(stderr,"size=%d\n",outsize);
      outdata=malloc(outsize);
      if (!outdata) {
         fclose(infp);
         fprintf(stderr,"unable to alloc all that,
aborting\n"); exit(1); }
      fseek(infp,0,SEEK_SET);
      fread(outdata,1,outsize,infp);
      fclose(infp);}

tcgetattr(fd,&oldtio); /* save current port settings */

   bzero(&newtio, sizeof(newtio));
   newtio.c_cflag = BAUD_RATE | CS8 | CLOCAL |
CREAD | CRTSCTS;
   newtio.c_iflag = IGNPAR;
   newtio.c_oflag = 0;
   cfsetospeed(&newtio,BAUD_RATE);
   cfsetispeed(&newtio,BAUD_RATE);
   newtio.c_lflag = 0;
   newtio.c_cc[VTIME]=0;
   newtio.c_cc[VMIN] =1;
   tcflush(fd, TCIFLUSH);
   tcsetattr(fd,TCSANOW,&newtio);

   if (cmds) {char tst[5];
      tst[0]=CMD_BLOCK_START;
      char *p=cmds;
      for (;*p;p++) { switch (*p) {case 'b':
tst[1]=CMD_LCD_BACKLIGHT_OFF;
                  write(fd,tst,2);
                  break;
            case 'B':
tst[1]=CMD_LCD_BACKLIGHT_ON;
                  write(fd,tst,2);
                  break;
            case 'h': tst[1]=CMD_GO_OFF_HOOK;
                  write(fd,tst,2);
                  break;
            case 'H': tst[1]=CMD_GO_ON_HOOK;
                  write(fd,tst,2);
                  break;
            case 'c': {time_t tt;
                     struct tm *t;
                     time(&tt); t=localtime(&tt);
                     fprintf(stderr,"Setting time to "
               "%02d:%02d:%02d... ",t->tm_hour,
                           t->tm_min,t->tm_sec);
                     tst[1]=CMD_SET_TIME;
                     tst[2]=t->tm_hour;
                     tst[3]=t->tm_min;
                     tst[4]=t->tm_sec;
                     write(fd,tst,5);
                     fprintf(stderr,"done\n");}
                  break;
            case 'C': {time_t tt;
                     struct tm *t;
                     time(&tt); t=localtime(&tt);
                     fprintf(stderr,"Setting date to "
            "%02d-%02d-%02d... ",t->tm_year-100,
                           t->tm_mon+1,t->tm_mday);
                     tst[1]=CMD_SET_DATE;
                     tst[2]=t->tm_year-100;
                     tst[3]=t->tm_mon+1;
                     tst[4]=t->tm_mday;
                     write(fd,tst,5);
                     fprintf(stderr,"done\n");}
                  break;}}}

in_cmd=0; adpcm_cnt=0; in_sync=0;
   long count=0,divcnt=0;
   for (;;) {
      if (!divcnt) {
         if (outdata) {
            char buf[8000];
            int n,buflen=0;
            for (n=0;n<1;n++) {
               unsigned char c=outdata[outindex++];
               if (outindex>=outsize) outindex=0;
               buf[buflen++]=c;
```

```
                if (c==CMD_BLOCK_START)
buf[buflen++]=c; }
              write(fd,buf,buflen); }
           divcnt=1;} else { divcnt--;}
        res = read(fd,&c,1);
        if (res>0) {count++;
           if (adpcm_cnt) {
              switch(adpcm_cnt) {
                 case 1: mystate.valprev=c;
                       adpcm_cnt=2; break;
                 case 2: mystate.valprev|=c<<8;
                       adpcm_cnt=3; break;
                 case 3: mystate.index=c;
                       if (!has_state) {state=mystate;
              fprintf(stderr,"at %d got state %d %d\n ",
count,mystate.valprev,mystate.index); }
           adpcm_cnt=0; in_cmd=0; has_state=1;
in_sync=1;
                       break;}
                 continue;}
           if (in_cmd>0) {
              switch (c) {
                 case CMD_BLOCK_START:
                    if (in_sync) audio_decode(c);
                    in_cmd=0;
                    break;
                 case CMD_ADPCM_STATE:
                    adpcm_cnt=1;
                    break;
                 case CMD_NOTIFY_ON_HOOK:
                    in_cmd=0;
                    fprintf(stderr,"Now on hook\n");
                    break;
                 case CMD_NOTIFY_OFF_HOOK:
                    in_cmd=0;
                    fprintf(stderr,"Now off hook\n");
                    break;
                 case CMD_NOTIFY_DISCONNECTED:
                    in_cmd=0;
                    fprintf(stderr,"DISCONNECTED\n");
                    break;
                 case CMD_RAW_BLOCK:
                    if (!raw_mode) fprintf(stderr,"** Raw
mode\n");
                    raw_mode=1; in_sync=1; in_cmd=0;
                    break;
                 case CMD_DTMF_DETECTED:
                    in_cmd=0;
                    res = read(fd,&c,1);
                    if (res>0)
          fprintf(stderr,"-- DTMF detected = '%c'\n",c);
                    break;
                 case CMD_LOCAL_RING_DETECTED:
                    in_cmd=0;
                    res = read(fd,&c,1);
                    if (res>0)
          fprintf(stderr,"** LOCAL RING #%d\n",c);
                    break;
                 case CMD_RECORDING_STARTED:
                    in_cmd=0;
                    recording_in_progress=TRUE;
          fprintf(stderr,">> Recoding now in
progress\n",c);
                    break;
                 case CMD_RECORDING_STOPPED:
                    in_cmd=0;
                    recording_in_progress=FALSE;
          fprintf(stderr,">> Recoding stopped\n",c);
                    break;
                 case CMD_GOERTZEL_DEBUG:
                    in_cmd=0;
                    unsigned char n;
                    res = read(fd,&n,1);
                    unsigned short x=0;
                    res = read(fd,&x,1);
                    res = read(fd,&c,1);
                    x|=c<<8;
                    fprintf(stderr,"%6d ",x);
                    if (n==9) putc('\n',stderr);
                    break;}} else {
              if (c==CMD_BLOCK_START) {
                 in_cmd=1;} else {
                 if (in_sync) audio_decode(c);}}}}
           tcsetattr(fd,TCSANOW,&oldtio);
     fclose(fp);}
void audio_decode(char c)
{  int n,s;
     if (raw_mode) { putc(c,stdout); return;}
     abuf[abuf_tail++]=c;
     if (abuf_tail==AUDIO_BLOCK_LENGTH)
    { abuf_tail=0;

adpcm_decoder(abuf,sbuf,AUDIO_BLOCK_LENGTH
*2,&state);
        for (n=0;n<2*AUDIO_BLOCK_LENGTH;n++) {
           s=sbuf[n]+32768;
           putc((s>>8)&0xFF,stdout);}}}

/* ----------------- Auxiliary Functions ----------------- */
void
adpcm_decoder(char indata[], short outdata[], int len,
              struct adpcm_state *state)
{  signed char *inp;
     short *outp;
     int sign;
     int delta;
     int step;
     int valpred;
     int vpdiff;
     int index;
     int inputbuffer;
     int bufferstep;
     outp = outdata;
     inp = (signed char *)indata;
     valpred = state->valprev;
     index = state->index;
     step = stepsizeTable[index];
     bufferstep = 0;
      for ( ; len > 0 ; len-- ) {
            if ( bufferstep ) {
               delta = inputbuffer & 0xf;
            } else {inputbuffer = *inp++;
```

```
            delta = (inputbuffer >> 4) & 0xf;}
        bufferstep = !bufferstep;
        index += indexTable[delta];
        if ( index < 0 ) index = 0;
        if ( index > 88 ) index = 88;
        sign = delta & 8;
        delta = delta & 7;
        vpdiff = step >> 3;
        if ( delta & 4 ) vpdiff += step;
        if ( delta & 2 ) vpdiff += step>>1;
if ( delta & 1 ) vpdiff += step>>2;
        if ( sign )
        valpred -= vpdiff;
        else
          valpred += vpdiff;
        if ( valpred > 32767 )
          valpred = 32767;
        else if ( valpred < -32768 )
          valpred = -32768;
        step = stepsizeTable[index];
            *outp++ = valpred; }
    state->valprev = valpred;
    state->index = index;}
```