



Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 Информатика и вычислительная техника

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по дисциплине «Микропроцессорные системы»
на тему:

Синтезатор речи

Студент

ИУ6-73Б
(Группа)

(Подпись, дата)

В.К. Залыгин
(И.О. Фамилия)

Руководитель

(Подпись, дата)

И.Б. Трамов
(И.О. Фамилия)

2025 г.

**Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ
Заведующий кафедрой ИУ6
_____ А.В. Пролетарский
« 2 » сентября 2025 г.

**З А Д А Н И Е
на выполнение курсовой работы**

по дисциплине Микропроцессорные системы

Студент группы ИУ6-73Б

Залыгин В.К.

Тема курсовой работы: Синтезатор речи

Направленность курсовой работы: учебная

Источник тематики (кафедра, предприятие, НИР): кафедра

График выполнения работы: 25% – 4 нед., 50% – 8 нед., 75% – 12 нед., 100% – 16 нед.

Техническое задание:

Разработать на основе микроконтроллера устройство озвучивания передаваемого текста. Обеспечить поддержку текста на русском и английском языках. Текст на устройство должен передаваться по протоколу UART.

Разработать схему, алгоритмы и программу. Отладить проект в симуляторе или на макете. Оценить потребляемую мощность. Описать принципы и технологию программирования используемого микроконтроллера.

Оформление курсовой работы:

1. Расчетно-пояснительная записка на 30-35 листах формата А4.
2. Перечень графического материала:
 - а) схема электрическая функциональная;
 - б) схема электрическая принципиальная.

Дата выдачи задания: «2» сентября 2025 г.

Руководитель курсовой работы

Студент

02.09.2025
(Подпись, дата)
02.09.2025
(Подпись, дата)

И.Б. Трамов
(И.О.Фамилия)
В.К. Залыгин
(И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах; один выдается студенту, второй хранится на кафедре.

РЕФЕРАТ

РПЗ 174 с., 16 рис., 1 табл., 10 источн., 2 прил.

МИКРОКОНТРОЛЛЕР, ATMEGA128A, СИНТЕЗ РЕЧИ, TTS, UART, SAM

Курсовая работа посвящена разработке проекта TTS (text-to-speech) устройства на основе микроконтроллера ATMega128A семестра AVR и программы SAM для генерации звуковой речи на русском и английском языках на основе передаваемого на устройство по UART текста.

Основная цель курсовой работы состоит в формировании навыков разработки и проектирования микропроцессорных систем путем освоения современных технологий проектирования систем на основе микроконтроллеров, а также программируемых систем на кристалле.

В результате работы были разработаны функциональная и принципиальная схемы устройства, выполнено портирование программы сам под SAM аппаратные средства микроконтроллера с написанием необходимых модулей обвязки. По завершении проектирования создан макет для проверки работоспособности схемы и программного обеспечения.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1 Конструкторская часть	8
1.1 Анализ требований ТЗ	8
1.2 Описание структурной схемы	9
1.3 Описание функциональной схемы	10
1.3.1 Описание архитектуры и характеристики ATMega128A	12
1.3.2 Структура и организация памяти микроконтроллера	13
1.4 Описание принципиальной электрической схемы	14
1.4.1 Подключения МК и программатора	15
1.4.2 Подключение микросхемы памяти	16
1.4.3 Подключение модуля СН340С	18
1.4.4 Подключение усилителя РАМ8403, RC-фильтра и динамика. . . .	19
1.5 Оценка потребляемой мощности	20
2 Технологическая часть	23
2.1 Выбор инструментов и средств разработки	23
2.2 Программирование МК	23
2.3 Общая модель исполнения	24
2.4 Портирование SAM под МК ATMega128A	27
2.4.1 Устранение ошибок сборки	28
2.4.2 Работа с памятью программы	29
2.5 Модуль трансляции	32
2.6 Асинхронные операции ввода-вывода	34
2.6.1 Модуль асинхронного буфера входного потока	35
2.6.2 Модуль асинхронного проигрывателя звукового файла	37
ЗАКЛЮЧЕНИЕ	40
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	41
ПРИЛОЖЕНИЕ А ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ	42
ПРИЛОЖЕНИЕ Б СПЕЦИФИКАЦИЯ РАДИОЭЛЕМЕНТОВ СХЕМЫ . . .	173

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

МК – микроконтроллер, микросхема для программного управления электронными устройствами

ОЗУ – оперативное запоминающее устройство, память с произвольным доступом для хранения рабочих данных программы

ПК – персональный компьютер, компьютер индивидуального пользования

Программатор – специальное устройство, позволяющее записывать и читать память программ микроконтроллера, а также конфигурировать его состояние

Тулчейн – (англ. toolchain) программный комплекс, включающий компилятор, компоновщик и вспомогательные утилиты для сборки и отладки программы

Callback-функция – (англ. callback – обратный вызов) функция обратного вызова, указатель на которую передаётся как параметр и которая вызывается по завершении асинхронной операции

TTS – (англ. Text-To-Speech) технология синтеза речи на основе текста

UART – (англ. Universal Asynchronous Receiver-Transmitter) универсальный асинхронный приёмопередатчик, протокол последовательной передачи данных

USB - (англ. Universal Serial Bus) универсальная последовательная шина, интерфейс подключения периферийных устройств к вычислительной технике

SRAM – (англ. Static Random-Access Memory) статическое ОЗУ с произвольным доступом, не требующее периодического обновления ячеек памяти

Libc – стандартная библиотека языка C

Malloc – функция стандартной библиотеки языка C для динамического выделения области памяти из кучи

XMEM – (англ. eXternal MEMory) параллельный интерфейс подключения внешней памяти к микроконтроллерам серии AVR

ВВЕДЕНИЕ

Курсовая работа «Синтезатор речи» выполняется на основании учебного плана кафедры ИУ6.

К одному из наиболее широко используемых в учебных и практических разработках семейств микроконтроллеров относятся 8-разрядные микроконтроллеры AVR фирмы Atmel (в настоящее время – Microchip), которые основаны на RISC-архитектуре с гарвардской организацией памяти, обладают развитой системой периферийных модулей, низким энергопотреблением и сравнительно простой системой программирования. AVR-микроконтроллеры применяются в измерительной технике, системах автоматизации и управления, в встраиваемой аудио- и видеоаппаратуре, а также в образовательных макетах и учебных стендах.

Целевым микроконтроллером в данной работе является ATMega128A, относящийся к семейству высокопроизводительных 8-битных AVR. Он обладает большим объемом программной памяти (128 кБайт Flash), внутренней ОЗУ (4 кБайт) и энергонезависимой EEPROM-памятью. К архитектурным особенностям данного микроконтроллера относятся наличие нескольких таймеров/счетчиков с поддержкой режима ШИМ, универсальных последовательных интерфейсов (USART, SPI, TWI), встроенного АЦП, а также внешнего интерфейса памяти XMEM.

В основе проекта лежит программа SAM (Software Automatic Mouth), предназначенная для синтеза речи (TTS) на английском языке, созданная компанией Don't ask software в 1982 году для компьютеров Commodore C64, Atari 8-bit, Apple II. SAM является одной из первых коммерческих программ синтеза речи, выпущенных на рынок. Изначальная программа в процессе работы генерирует готовый к воспроизведению файл с речью (в формате .wav), а также имеет ряд входных параметров для модификации выходного голоса и набор функций для отладки. Изначально написанная на ассемблере под целевые компьютеры, в 2015 году благодаря стараниям Stefan Maske программа была дизассемблирована в код на языке C и выложена в открытый доступ [1] (лицензия программы не была указана в

связи с неуспешными попытками связаться с создателями, компания Don't ask software ныне не существует). Программа обладает весьма малыми размерами – автономная версия программы под x86 занимает менее 39 кБ дискового пространства и требует менее 128 кБ оперативной памяти для работы (в основном занимаемой выходным .wav файлом). Данная особенность программы позволяет портировать ее даже на самые маленькие МК системы.

В рамках данной курсовой работы ставится задача портирования SAM под МК ATMega128A.

С точки зрения проекта микроконтроллер ATMega128A обладает всеми необходимыми свойствами:

- наличие 128 кБ памяти программы;
- ОЗУ размером в 4 кБ, которое расширяется до 64 кБ при использовании внешней памяти и интерфейса XMEM;
- наличие нескольких таймеров, поддерживающих ШИМ;
- наличие интерфейсов передачи данных, таких как UART;
- высокая производительность МК на частоте 8 МГц;
- простота и открытость архитектуры МК семейства AVR для настройки.

1 Конструкторская часть

1.1 Анализ требований ТЗ

Согласно техническому заданию, необходимо разработать на основе микроконтроллера устройство озвучивания текста на русском и английском языках. Текст к микроконтроллеру должен передаваться по протоколу UART.

Микроконтроллер должен уметь работать с протоколом UART и считывать входящий текст, выполнять необходимые преобразования, а затем озвучивать его. Как и оригинальная программа SAM, устройство должно поддерживать флаги, передаваемые во входящем тексте (флаг обозначается при помощи тире на месте первого символа). Также устройство должно уметь подавать сигнал о готовности к вводу после инициализации, передавать информационные и отладочные сообщения по UART передающему устройству.

Поскольку встроенной в МК ОЗУ объемом 4 кБ не хватает для генерации звукового потока, необходимо использовать внешнюю память, которая расширяет адресное пространство данных вплоть до 64 кБ, чего уже достаточно для работы алгоритмов синтеза речи. Согласно даташиту микроконтроллера, работа с внешней памятью должна осуществляться при помощи вспомогательного регистра-защелки, который необходим для хранения младших бит адреса до окончания цикла чтения или записи [2].

Для генерации звукового потока сигнал ШИМ, генерируемый МК, должен проходить несколько этапов обработки: сначала он поступает на RC-фильтр, который сглаживает высокочастотную составляющую, затем усиливается усилителем мощности, повышающим уровень сигнала по напряжению и току и обеспечивающим необходимую громкость динамика.

Для быстрого и простого соединения МК с компьютером устройство снабжается преобразователем интерфейсов UART-USB.

Питание устройство можно организовать при помощи того же самого USB-порта. Согласно спецификации интерфейса USB 2.0 напряжение на линии питания составляет 5 вольт [3], что подходит микроконтроллеру.

1.2 Описание структурной схемы

По результатам анализа требований к устройству можно сформулировать ряд компонентов, необходимых для работы устройства:

- микроконтроллер;
- UART-USB преобразователь;
- микросхема памяти;
- регистр-защелка;
- RC-фильтр;
- усилитель;
- динамик.

На рисунке 1 представлена структурная схема устройства.

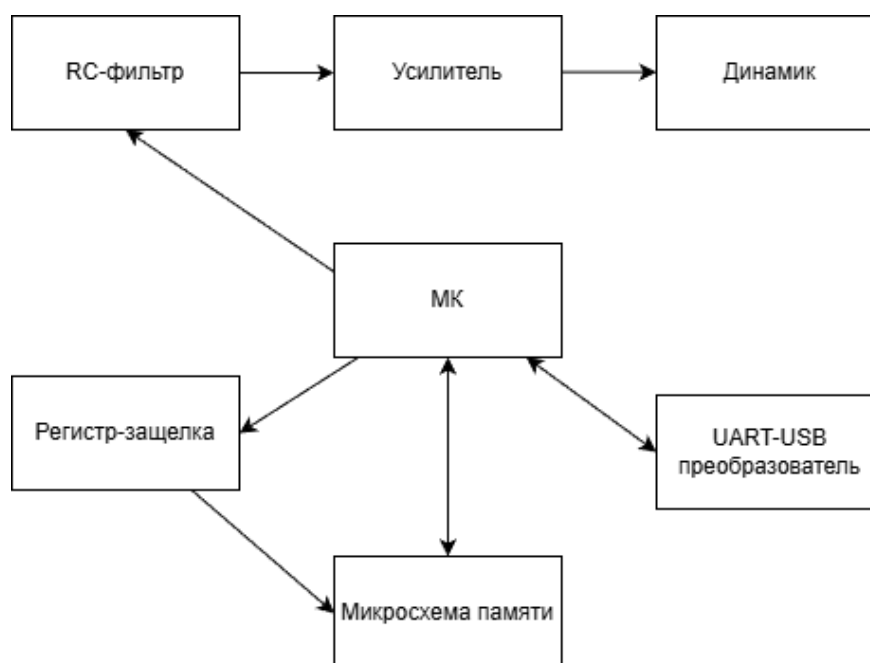


Рисунок 1 — Структурная схема

Микроконтроллер играет центральную роль в устройстве и потому расположен в центре схемы. Подключение к компьютеру, передача и приём данных от него производится через UART-USB преобразователь. Звуковой сигнал формируется на основе сигнала ШИМ, проходящего обработку в RC-фильтре и усилителе. Внешняя память и регистр-защёлка образуют подсистему расширения ОЗУ микроконтроллера.

1.3 Описание функциональной схемы

На основе структурной схемы выполнена функциональная схема устройства, которая уточняет состав внутренних модулей микроконтроллера и логические связи между ними и внешними блоками. Функциональная схема представлена на рисунке 2.

На функциональной схеме микроконтроллер ATmega128A изображён в развернутом виде: показаны ядро AVR, блок программной Flash-памяти, внутренняя SRAM, контроллер внешней памяти, таймеры/счётчики, интерфейсы USART, SPI, модуль тактирования и система прерываний.

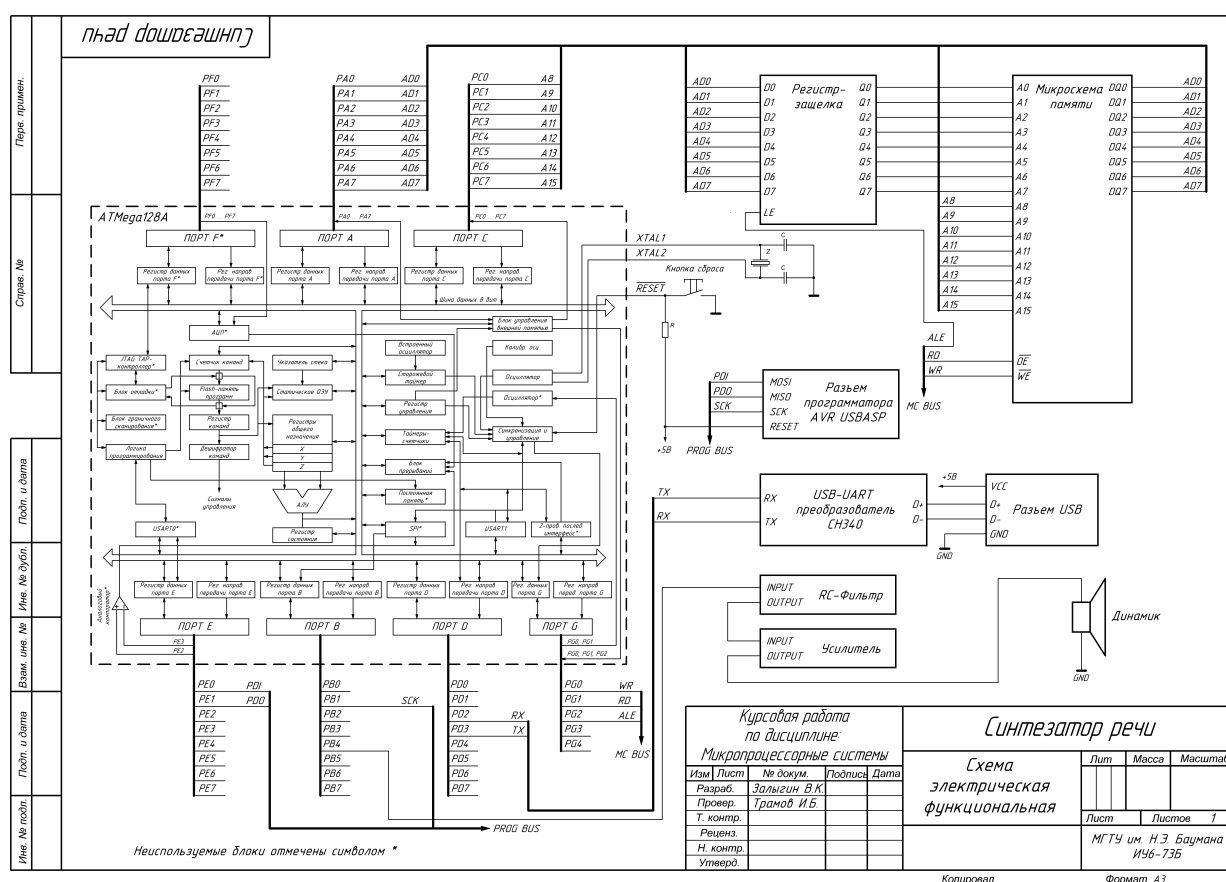


Рисунок 2 — Функциональная схема

Передача текста от персонального компьютера осуществляется по интерфейсу USB-UART. На функциональной схеме этот путь представлен как цепочка: USB разъем — USB-UART преобразователь — интерфейс USART1 микроконтроллера. USART1 принимает байтовый поток, на основе которого выполняются задачи: разбор текстовых флагов, преобразование текста и синтеза речи. В обратном направлении по тому же каналу передаются служебные и отладочные сообщения.

Для хранения промежуточных данных и выходного звукового буфера используется внешняя статическая память. В функциональном представлении подсистема памяти состоит из контроллера внешней памяти внутри микроконтроллера, шины адреса, шины данных (совмещена с младшими адресами шины адреса), регистра-защёлки и микросхемы SRAM. Контроллер внешней памяти формирует последовательность сигналов на шинах адреса и данных, а также управляющие сигналы чтения и записи в рамках параллельного интерфейса XMEM. Регистр-защёлка выполняет функцию временного хранения младшей части адреса. С точки зрения программы внешняя память воспринимается как продолжение внутренней SRAM, доступ к которой осуществляется обычными операциями чтения и записи по нужным адресам (с учетом необходимой конфигурации контроллера внешней памяти и разметки память данных для программы).

Генерация звукового сигнала представлена как последовательность модулей «ШИМ-выход таймера» — «RC-фильтр» — «Усилитель мощности» — «Динамик». Таймер/счётчик 0 работает в режиме генерации ШИМ и получает от программной части массив значений, соответствующих дискретному уровню звукового сигнала. Контроллер прерываний обеспечивает регулярную загрузку новых значений в регистр таймера 0 с требуемой частотой дискретизации (более подробное описание представлено в технологической части РПЗ), RC-фильтр сглаживает высокочастотную составляющую сигнала — таким образом реализован простой цифро-аналоговый преобразователь. Усилитель мощности обеспечивает повышение мощности сигнала для подачи на динамик (мощность выхода GPIO-порта микроконтроллера слишком мала и при прямом подключении динамик не было бы слышен).

Отдельным представлена изображена схема тактирования. Кварцевый резонатор и схема генератора формируют опорную частоту 8 МГц. Наличие внешнего кварцевого резонатора продиктовано недостаточной точностью внутреннего механизма тактирования для работы с UART.

Наконец, показан интерфейс с программатором по SPI. Модуль SPI микроконтроллера используется для внутрисхемного программирования

памяти программ и конфигурационных фьюзов. На схеме это отражено как отдельный канал взаимодействия с внешним модулем «Программатор».

1.3.1 Описание архитектуры и характеристики ATMega128A

Микроконтроллер входит в семейство AVR, имеет гарвардскую архитектуру (программа и данные находятся в разных адресных пространствах) и систему команд, близкую к идеологии RISC. Процессор AVR имеет 32 8-битных регистра общего назначения, объединённых в регистровый файл.

В основном микроконтроллер можно встретить в корпусе TQFP64, распиновка которого показана на рисунке 3.

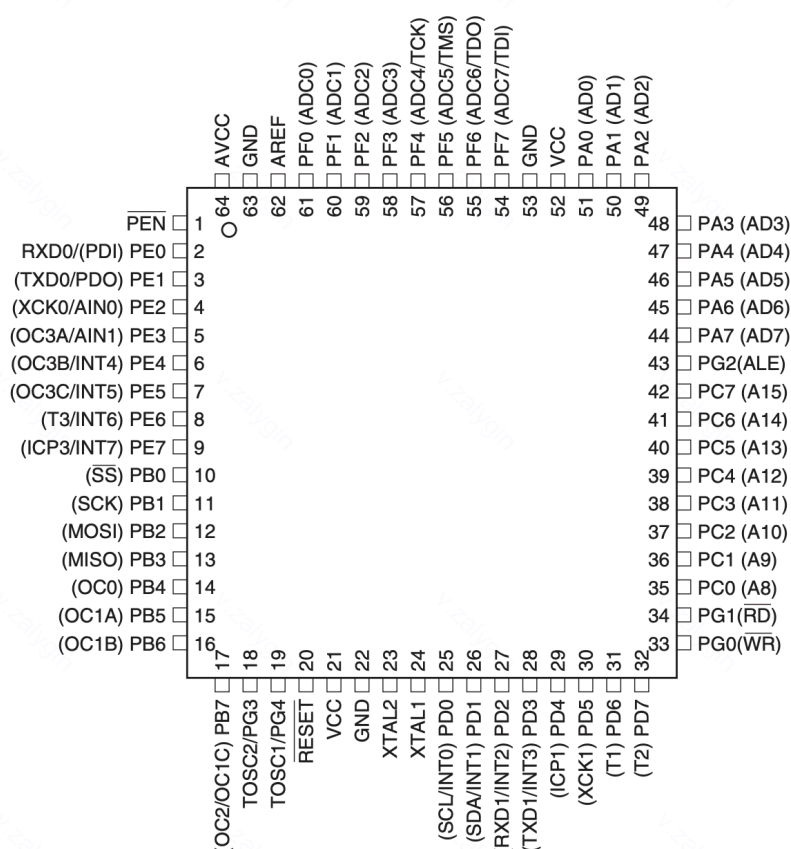


Рисунок 3 — Распиновка ATMega128A в корпусе TQFP64

Большинство ножек имеют несколько назначений: помимо 7 портов ввода-вывода, к ним также подключено множество периферии: 2 8-битных и 2 16-битных счётчика с поддержкой ШИМ, АЦП, Two-wire, два интерфейса USART, интерфейс SPI (с возможностью внутрисхемного программирования), сторожевой таймер, аналоговый компаратор, JTAG.

1.3.2 Структура и организация памяти микроконтроллера

В ATMega128A используется несколько типов памяти, каждый из которых имеет собственное адресное пространство и назначение. Программная память представляет собой встроенную Flash-память объёмом 128 Кбайт, организованную в виде слов по 16 бит. В ней размещается основной код программы, векторы прерываний, таблицы констант и, при необходимости, загрузчик. Доступ к Flash из программы осуществляется через специальные инструкции чтения программной памяти, а запись возможна только в режиме самопрограммирования под управлением бутлоадера.

Графическое описание пространства данных представлено на рисунке 4. Цветные области обозначают секции памяти, используемые устройством.

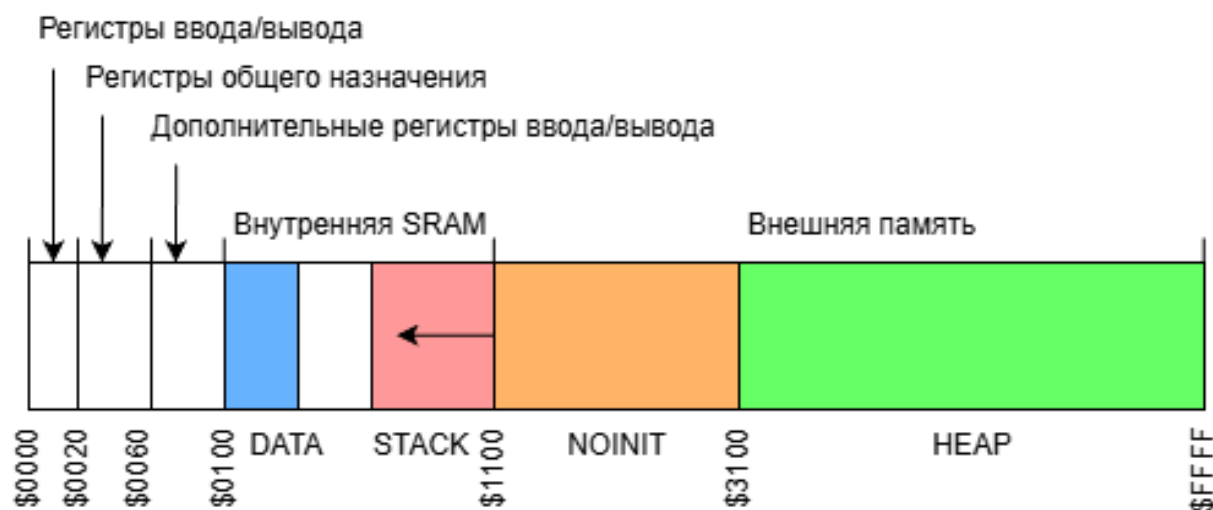


Рисунок 4 — Адресное пространство данных

Адресное пространство данных имеет объём 64 Кбайт и объединяет несколько областей. В самом начале располагаются 32 регистра общего назначения R0-R31. Далее следуют регистры ввода-вывода, в которых отображены все периферийные модули микроконтроллера: таймеры/счётчики, интерфейсы USART, SPI и TWI, АЦП, портовые регистры, регистры прерываний и конфигурации. Начиная с адреса, следующего за расширенной областью регистров ввода-вывода, располагается внутренняя SRAM объёмом 4 Кбайт.

При включённом интерфейсе XMEM верхняя часть адресного пространства данных отводится под внешнюю память. В рассматриваемом проекте во внешнюю область проецируется микросхема статической ОЗУ.

Алгоритм синтеза речи SAM из-за батчевого характера обработки данных активно используют массивы промежуточных данных и большой выходной буфер звукового сигнала. В данной реализации эти массивы размещаются именно во внешней ОЗУ, тогда как внутренняя память микроконтроллера резервируется под стек, системные переменные и небольшие служебные буферы.

Отдельно в микроконтроллере присутствует энергонезависимая EEPROM-память объёмом 4 Кбайт. В курсовой работе она не используется.

Скорость работы микросхемы памяти также вносит ограничение на частоту микропроцессора – 8 МГц. При более высоких частотах микросхема памяти не успевает отвечать микроконтроллеру, что ломает логику обработки данных.

1.4 Описание принципиальной электрической схемы

Принципиальная схема устройства представлена на рисунке 5.

На принципиальной схеме центральным элементом является микроконтроллер ATMega128A (DD1), к выводам которого подведены цепи питания, тактирования, сброса, программирования, интерфейса внешней памяти, интерфейса связи с ПК и звуковой тракт.

Питание устройства осуществляется от линии +5 В, поступающей с разъёма USB. Вблизи выводов питания микроконтроллера и других микросхем размещены развязывающие керамические конденсаторы. Тактовый генератор построен на кварцевом резонаторе Z1 частотой 8 МГц, подключённом к выводам XTAL1 и XTAL2 микроконтроллера. Два конденсатора небольшой ёмкости (20 нФ) согласно требованиям из документации к микроконтроллеру (C13 и C14) образуют с резонатором пьезоэлектрический генератор.

Цепь сброса включает подтягивающий резистор R2, соединяющий вывод RESET с линией +5 В, и кнопку SB1, замыкающую этот вывод на

Имя, дата

Подп. и дата

Имя, ин. №

Взам. инв. №

Имя, дата

Подп. и дата

Имя, ин. №

план довершения

Справа, примен.

Слева, №

Имя, дата

Подп. и дата

Имя, ин. №

Взам. инв. №

Имя, дата

Подп. и дата

Имя, ин. №

Синтезатор речи

Схема электрическая принципиальная

Изм.	Лист	№ докум.	Подпись	Дата
Разработчик	Залыгин В.К.	Транов И.Б.		
Проектировщик	Т. контр.			
Рецензент	Н. контр.			
Утвердил				

Справа от микроконтроллера размещён узел внешней памяти, включающий регистр-защёлку SN74HC573A (DD2) и микросхему SRAM AS6C1008 (DD3) объемом 128 кБ. Шестнадцатиразрядная адресная шина микроконтроллера позволяет адресовать до 64 кБ внешней памяти, поэтому используется только младшая часть объёма микросхемы SRAM – старший адресный бит A16 притянут к нулю.

1.4.1 Подключения МК и программатора

15

соответственно к выводам PB3, PB2 и PB1 микроконтроллера, которые совмещают функции интерфейса SPI и портовых линий.

Вывод RESET микроконтроллера подключён к разъёму XP1 и используется программатором для перевода МК в режим программирования. В штатном режиме этот вывод подтянут к питанию резистором R2 и может принудительно замыкаться на землю кнопкой SB1. Такая схема обеспечивает как аппаратный сброс пользователем, так и корректное управление режимами микроконтроллера со стороны программатора.

1.4.2 Подключение микросхемы памяти

Подключение внешней памяти реализует стандартный параллельный интерфейс XMEM, поддерживаемый ATMega128A. Линии порта A микроконтроллера используются как мультиплексированная шина AD0-AD7: они подключены одновременно ко входам D0-D7 регистра-защёлки SN74HC573A и к двунаправленной шине данных D0-D7 микросхемы SRAM AS6C1008. Сигнал ALE, формируемый выводом PG2, подаётся на вход LE регистра-защёлки как того требует документация микросхемы [4]. В момент начала цикла обращения по шине на порт A выводится младший байт адреса, и при помощи стробирующего сигнала ALE фиксируется выходах Q0-Q7, которые подключены к адресным входам A0-A7 микросхемы SRAM [5].

Старшие адресные биты A8-A15 формируются выводами порта C микроконтроллера и напрямую подключены к соответствующим выводам памяти. Управляющие сигналы чтения и записи формируются выводами порта G: сигнал /RD (PG1) подаётся на вход /OE ОЗУ, а сигнал /WR (PG0) — на вход /WE. Вывод /CE микросхемы памяти подключён к общему проводу, а вывод CE2 — к шине +5 В, что обеспечивает постоянно разрешённое состояние микросхемы [6]; фактический выбор режима работы определяется только сигналами /OE и /WE. Адресный вход A16 притянут к земле, поэтому используется младшая половина доступного адресного пространства ОЗУ. В цепях питания регистра-защёлки и памяти установлены развязывающие керамические конденсаторы C4 и C5.

Выбор микросхемы памяти случайного доступа AS6C1008, построенной на низковольтных КМОП-транзистрах, обусловлен ее объемом, скоростями работы и требуемым питанием, а также ее интерфейсом, совместимым с параллельным интерфейсом ХМЕМ. Микросхема может хранить 128 кБайт информации, адресуемые при помощи 17 битов (в рамках устройство используется 16 бит, контакт старшего бита притянут к земле). Тайминги ответа микросхемы не превышают 55нс [6] (что позволяет обращаться к микросхеме на частоте 8 МГц), а сама микросхема работает при диапазоне напряжений от 2,7 до 5,5 вольт [6]. Набор перечисленных свойств делает ее подходящей под использование в курсовой работе.

Для поддержки интерфейса ХМЕМ микросхеме памяти необходим регистр-защелка на 8 бит, коим выступает SN74HC573A. Регистр аналогично построен на технологии низковольтных КМОП-транзисторов и питается напряжением в диапазоне от 2 до 6 вольт [4]. Для регистра скорость распространения сигнала составляет в среднем 21нс [4].

Таким образом микросхемы были подобраны на основе необходимого напряжения питания, разрядности, скорости ответа, чтобы подходить под режим работы ATMega128A на частоте 8 МГц без дополнительных согласований напряжений.

Для конфигурации работы с внешней памятью используются регистры MCUCR, XMCRA, XMCRB, в которых указываются следующие биты:

- SRE – включает/выключает интерфейс внешней памяти ХМЕМ;
- SRWn1 и SRWn0 – биты выбора числа состояний ожидания (wait-states) при доступе к внешней памяти для нижнего/верхнего сектора;
- SRL2, SRL1, SRL0 – задают разбиение адресного пространства внешней памяти на нижний и верхний сектор (адрес границы сектора);
- XMBK – разрешение работы устройства запоминания состояния шины на линиях AD7:0, при XMBK=1 шина удерживает последнее состояние, когда переведена в третье состояние;
- XMM2, XMM1, XMM0 – маска старших адресных разрядов A15:8; определяют, сколько линий порта C используется как старший адресный байт

внешней памяти, а сколько остаётся обычными линиями ввода-вывода (тем самым ограничивается максимальный объём адресуемой RAM);

Регистры MCUCR, XMCRA представлены на рисунке 6.

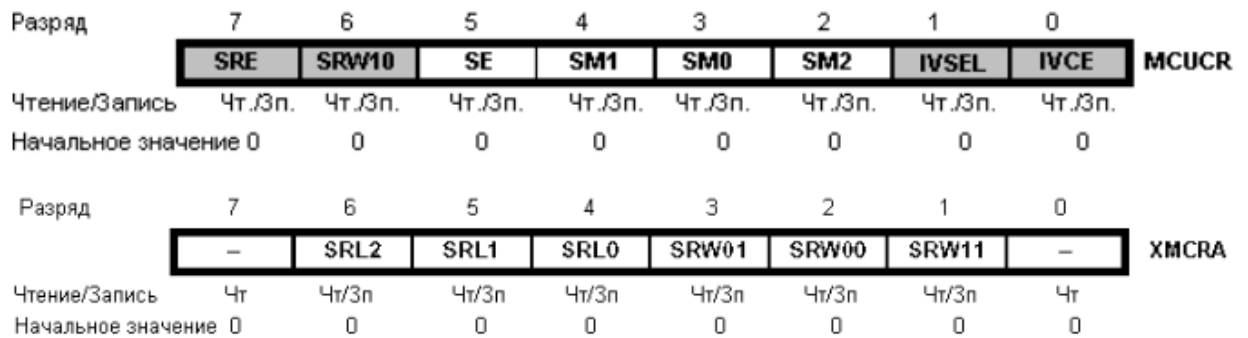


Рисунок 6 — Регистры MCUCR и XMCRA

Примеры временных диаграмм циклов чтения и записи показаны на рисунке 7. В устройстве используются короткие циклы без дополнительных тактов задержек, что возможно благодаря быстродействию микросхем регистра-защелки и памяти.

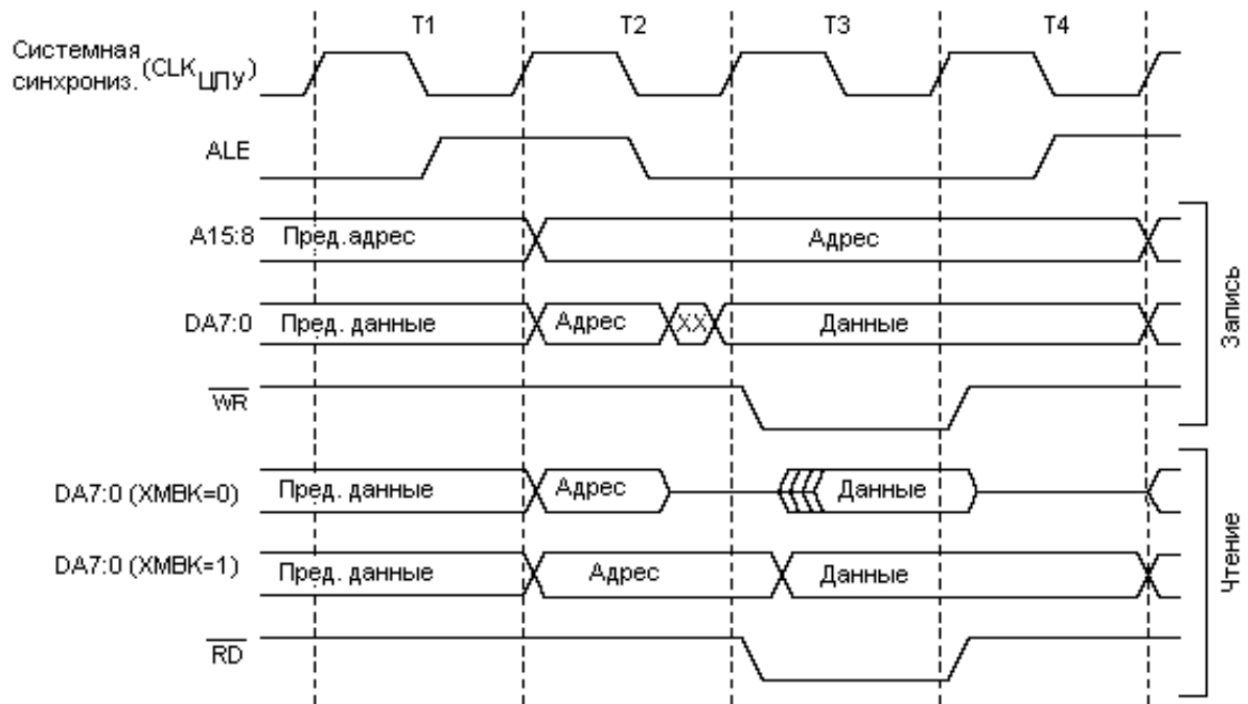


Рисунок 7 — Временная диаграмма XMEM циклов чтения и записи без дополнительных задержек

1.4.3 Подключение модуля CH340C

Сопряжение микроконтроллера с персональным компьютером по протоколу UART реализовано на микросхеме USB-UART преобразователя

CH340C (DD5), которая с одной стороны имеет интерфейс UART (TTL уровни напряжений), а с другой стороны – интерфейс USB. На её выводы UD+ и UD- подведены линии D+ и D- разъёма USB XP2, а вывод VCC подключён к линии питания +5 В. Встроенный стабилизатор ядра микросхемы питается от линии +5 В. Пин на питание +3 В развязан через керамический конденсатор на землю в соответствии с требованиями из документации [7] – в питании микросхемы пин не участвует.

Последовательный интерфейс UART со стороны микроконтроллера реализуется линиями USART1. Вывод PD3 (TXD1) подключён к входу RXD микросхемы CH340C, а вывод PD2 (RXD1) — к её выходу TXD. Остальные выводы микросхемы CH340C, связанные с аппаратным управлением модемными сигналами (CTS, DSR, DTR и др.), в схеме не используются и остаются свободными или подтянутыми к фиксированным уровням согласно рекомендациям документации к микросхеме [7].

1.4.4 Подключение усилителя PAM8403, RC-фильтра и динамика.

Формирование аналогового звукового сигнала начинается на выводе PB4 микроконтроллера, который использует OC0 — выход таймера/счётчика 0 в режиме ШИМ. Прямоугольный ШИМ-сигнал с частотой несущей (работает на частоте микроконтроллера) значительно выше максимальной звуковой частоты поступает на RC-фильтр, образованный резистором R1 и конденсатором C3. Фильтр сглаживает высокочастотную составляющую звукового сигнала.

С выхода RC-фильтра (разрыв D на схеме) сигнал подаётся на вход INR усилителя мощности PAM8403 (DD4). Усилитель подобран согласно требуемому напряжению питания (5 вольт), а также выходной мощности (1,5 ватт на динамик) [8]. Усилитель питается от линии +5 В, к его выводам питания подключены конденсаторы C2, C9 и C10, обеспечивающие развязку питания микросхемы согласно документации [8]. В схеме используется только правый канал усилителя, пины второго канала притянуты к земле. Выход усилителя соединён к маленькому динамику FBS1850 (BA1) с сопротивлением 8 Ом [9]. Такой динамик подходит к усилителю, тк его входная мощность согласуется с мощностью усилителя.

Расчет RC-фильтра выполняется на основе следующих данных. Сопротивление резистора подбирается на основе напряжения питания и максимального тока на выходе GPIO-порта микроконтроллера ($I_{\max} = 20$ мА согласно даташиту на микроконтроллер [2]), тогда минимальное сопротивление резистора определяется законом Ома, описанной в формуле (1). Для наличия запаса удобно использовать резистор $R_1 = 1$ кОм. Для речи достаточно взять частоту среза около 3-3.5 кГц (передача голоса по телефонной связи работает на полосе частот 300-3400 Гц [10]). Тогда можно вычислить емкость конденсатора при сопротивлении резистора $R_1 = 1$ кОм и частоте среза $f_c = 3.2$ кГц. Вычисления представлены в формуле (2).

$$R_{\min} = \frac{U_{\text{питания}}}{I_{\max}} = \frac{5 \text{ В}}{0.02 \text{ А}} = 250 \text{ Ом}, \quad (1)$$

где R_{\min} – минимальное сопротивление резистора,
 $U_{\text{питания}}$ – напряжение питания устройства,
 I_{\max} – максимальный ток GPIO-порта микроконтроллера.

$$C_3 = \frac{1}{2\pi \cdot R_1 \cdot f_c} = \frac{1}{2\pi \cdot 1000 \text{ Ом} \cdot 3200 \text{ Гц}} \approx 50 \text{ нФ}, \quad (2)$$

где C_3 – искомая емкость конденсатора, входящего в RC-фильтр,
 R_1 – сопротивление резистора, входящего в RC-фильтр,
 f_c – требуемая частота среза.

Таким образом получены параметры компонент RC-фильтра: сопротивление резистора $R_1 = 1$ кОм и емкость конденсатора $C_3 = 50$ нФ.

1.5 Оценка потребляемой мощности

Для оценки энергопотребления устройства требуется определить суммарный ток, потребляемый всеми узлами при работе в типовом и в наиболее тяжёлом режимах.

Мощность RC-фильтра можно вычислить по формуле (3)

$$P_{RC} = \frac{U_{питания}^2}{R_1} = \frac{(5 \text{ В})^2}{1000} = 25 \text{ мВт}, \quad (3)$$

где P_{RC} – искомая мощность,

$U_{питания}$ – напряжение питания,

R_1 – сопротивление резистора, состоящего в RC-фильтре.

Для динамика мощность можно взять из даташита усилителя: $P_{динамик} = 1.5 \text{ Вт}$ [8].

Токи питания используемых микросхем приведены в таблице 1.

Таблица 1 — Таблица токов питания микросхем при напряжении 5 В

Обозначение	Микросхема	Ток питания, мА
DD1	ATMega128A	10
DD2	SN74HC573A	0.08
DD3	AS6C1008	10
DD4	PAM8403	3.5
DD5	CH340C	20

Мощность микросхем можно установить по формуле (4).

$$P = I_{питания} \cdot U_{питания}, \quad (4)$$

где P – потребляемая мощность микросхемы,

$I_{питания}$ – потребляемый ток,

$U_{питания}$ – напряжение питания.

Общая потребляемая мощность устройства будет определяться как суммарная мощность компонентов устройства и показано в формуле (5).

Таким образом суммарная мощность устройства составляет $P_{sum} \approx 1.570 \text{ Вт}$.

$$\begin{aligned}
P_{\text{sum}} &= \sum_i P_i = \sum_i I_i \cdot U_{\text{питания}} + P_{\text{RC}} + P_{\text{динамик}} = \\
&= (10 + 0.08 + 10 + 3.5 + 20) \text{ мА} \cdot 5 \text{ В} + 25 \text{ мВт} + 1.5 \text{ Вт} \approx \\
&\approx 1.570 \text{ Вт},
\end{aligned} \tag{5}$$

где P_{sum} – суммарная мощность всего устройства,
 P_i – мощность i -го устройства.

2 Технологическая часть

2.1 Выбор инструментов и средств разработки

При разработке программной части устройства на языке C используется компилятор `avr-gcc` (версия тулчейна 3.7.0_1796) и стандартной библиотеки `avr-libc`, которые обеспечивают доступ к регистрам периферии и специализированным функциям работы с программной памятью и внешним ОЗУ микроконтроллера ATmega128A. Работа с исходным кодом выполняется в среде Visual Studio Code как наиболее дружелюбной к всем остальным средствам разработки благодаря многочисленным расширениям и интеграциям.

Сборка проекта выполняется утилитой `make`, которая последовательно запускает компиляцию модулей, линковку и генерацию прошивки в формате Intel HEX (содержимое Makefile представлено в приложении А). В Makefile выделяются отдельные цели для компиляции, записи прошивки и очистки файлов сборки. Для программирования микроконтроллера используется утилита `avrdude`, вызываемая из `make` с заранее подготовленным набором параметров интерфейса программирования и сигнатуры целевого МК.

2.2 Программирование МК

На рисунке 8 показана структурная схема программной части устройства.

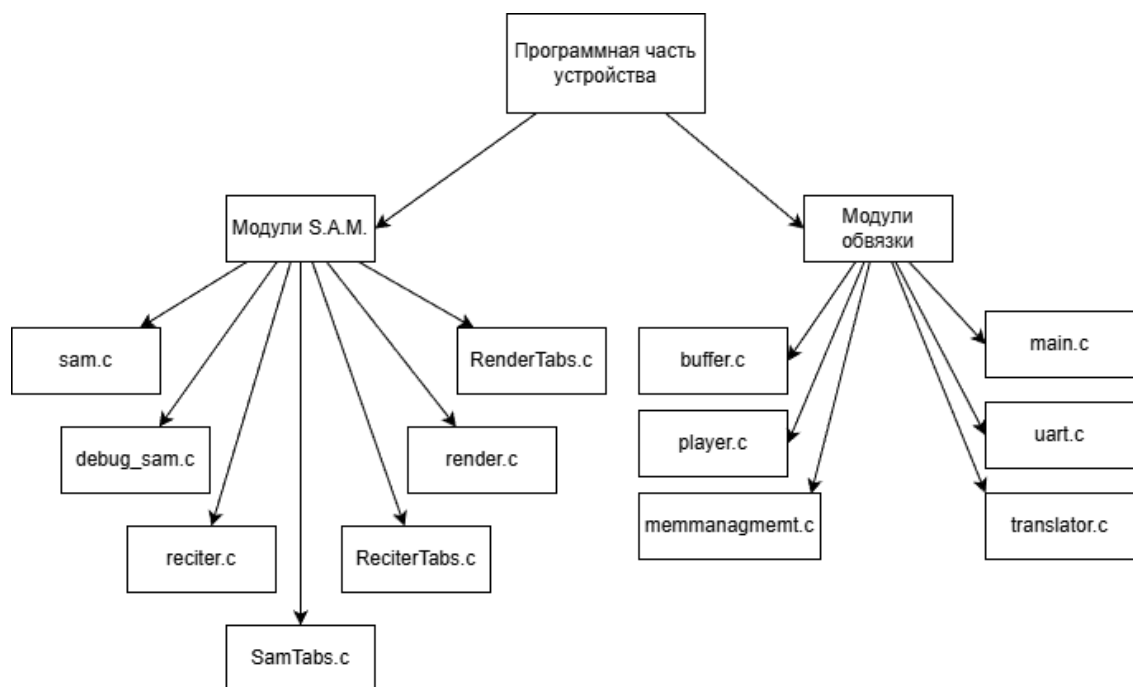


Рисунок 8 — Структурная схема

Логика программы делится на два крупных блока. Первый блок составляют модули оригинального синтезатора речи SAM: файлы «sam.c», «debug_sam.c», «reciter.c», «ReciterTabs.c», «render.c», «RenderTabs.c», «SamTabs.c». Данные файлы реализуют алгоритмы преобразования текста в фонемы, последующую обработку фонем и формирование дискретизированного звукового сигнала по таблицам частот и амплитуд. Эти файлы по сути представляют собой перенесённое ядро синтезатора и в минимальной степени зависят от аппаратной платформы (с поправкой на необходимые манипуляции при портировании на микропроцессорную платформу, описанные в дальнейших разделах).

Второй блок — модули «обвязки», которые выполняют функцию окружения для синтезатора и связывают его с аппаратными средствами микроконтроллера. В «main.c» находится основной цикл работы устройства и обработка команд пользователя. Модуль «uart.c» настраивает интерфейс USART1 микроконтроллера и перенаправляет стандартные потоки ввода-вывода stdin и stdout на последовательный порт. Модуль «buffer.c» реализует кольцевой буфер для асинхронного приёма данных по UART, а «player.c» отвечает за воспроизведение сгенерированного звукового буфера через ШИМ-выход. В файлах «memmanagement.h» и «memmanagement.c» сосредоточены вспомогательные средства работы с программной и внешней памятью, а модуль «translator.c» содержит алгоритмы транслитерации русского текста в английский.

2.3 Общая модель исполнения

Рисунок 9 иллюстрирует основной цикл работы устройства: после аппаратного сброса выполняется инициализация основных подсистем устройства: настраивается контроллер внешней памяти, интерфейс UART, модуль проигрывателя и глобальные структуры синтезатора речи. Затем разрешаются прерывания, и управление передаётся в основной бесконечный цикл.

Основной цикл организован как обработчик входного потока слов.

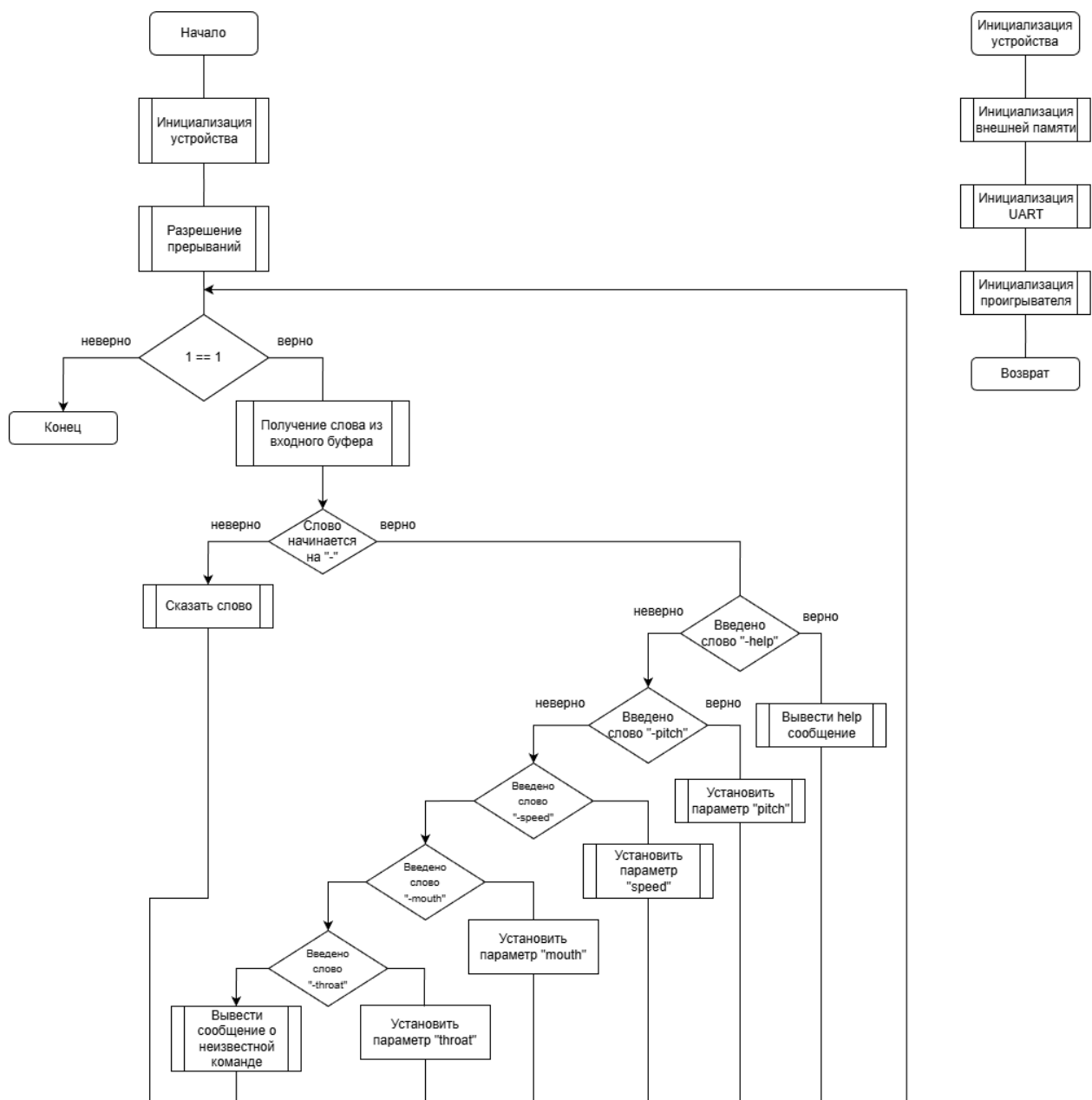


Рисунок 9 — Основной цикл работы

Текст от ведущего устройства по интерфейсу UART помещается в асинхронный кольцевой буфер, откуда цикл по мере готовности извлекает очередное слово. По первому символу определяется тип слова. Если оно начинается с «-», слово интерпретируется как команда изменения параметров синтеза: команды настраивают высоту тона, скорость и тембр звучания, возвращают справочную информацию. Неизвестные команды игнорируются с формированием диагностического сообщения. Если слово не имеет префикса «-», оно рассматривается как элемент произносимого текста и передаётся в функцию генерации речи.

Функция «Сказать слово», алгоритм которой изображён на рисунке 10, связывает между собой модули трансляции, синтеза и проигрывателя. На вход она получает слово или короткую фразу, выделенную из входного потока. Сначала выполняется нормализация текста: символы переводятся в верхний регистр и передаются модулю трансляции, который заменяет буквы кириллицы соответствующими латинскими последовательностями. Результирующая строка полностью соответствует требованиям синтезатора SAM и передаётся в его ядро.

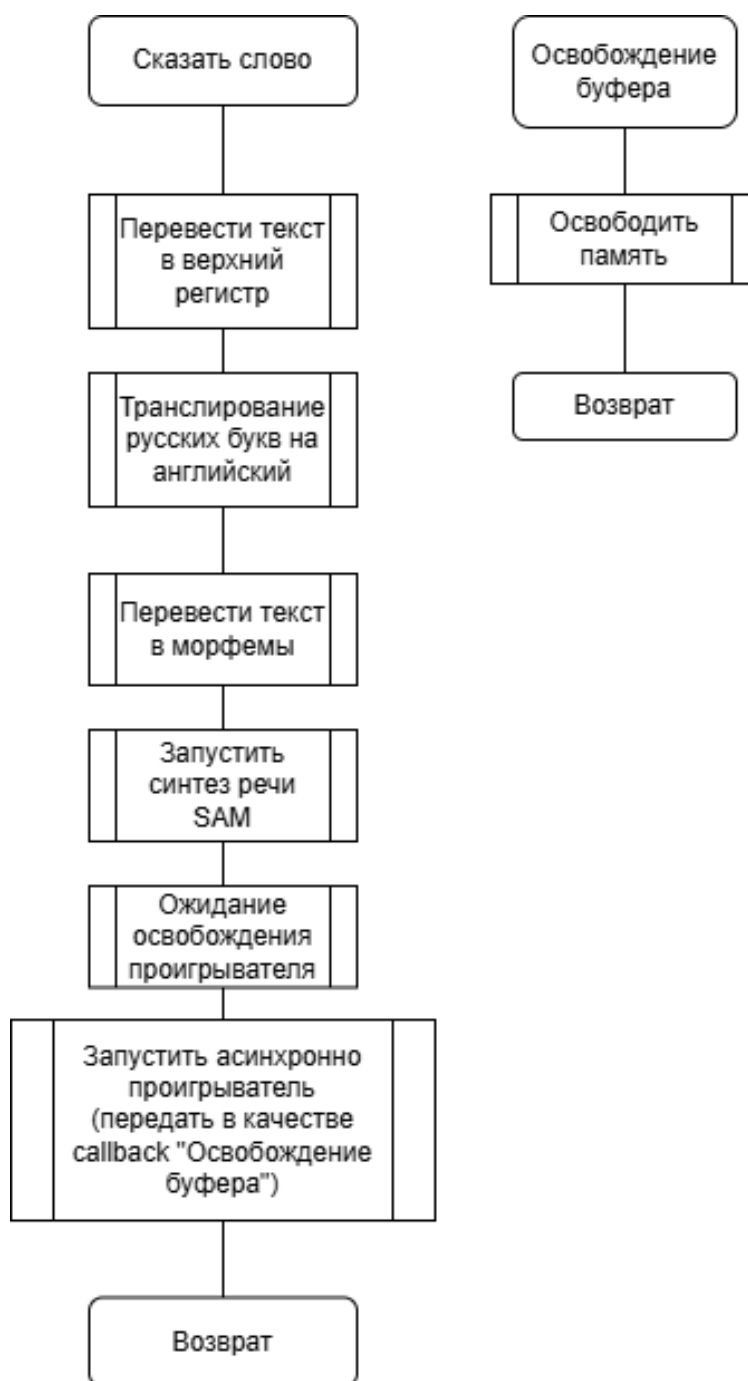


Рисунок 10 — Схема алгоритма основной функции генерации речи

В ходе работы синтезатора во внешней памяти формируется буфер дискретизированного звукового сигнала. После завершения синтеза функция проверяет состояние проигрывателя, при необходимости ожидает его освобождения и затем запускает асинхронное воспроизведение, передавая структуру с описанием буфера и callback-функцию освобождения памяти. После старта проигрывателя управление немедленно возвращается в основной цикл. Таким образом, общая модель исполнения строится по принципу: основной цикл управляет приёмом и разбором текста, а ресурсоёмкий синтез речи и вывод звука выполняются в специализированных модулях и обработчиках прерываний.

2.4 Портирование SAM под МК ATmega128A

Алгоритм синтеза SAM, схематично показанный на рисунке 11, в портированной версии сохраняет исходную структуру. Сначала выполняется инициализация внутренних буферов синтезатора. Затем вызывается первый проход парсера фонем, который преобразует входной текст в последовательность фонем с базовыми характеристиками. Второй проход уточняет фонемы, добавляет вставки и подготавливает таблицы длительностей. Далее накладываются эффекты удара и дыхания, после чего модуль рендера формирует выборки звукового сигнала, комбинируя несколько гармоник с различными амплитудами и формами волн. Полученный буфер передаётся в проигрыватель.

Программа SAM изначально создавалась как настольное приложение и ориентируется на особенности персональных ПК: прямой доступ к оперативной памяти и сравнительно большой объём ОЗУ. При переносе на ATmega128A возникает несколько ограничений: объём внутренней SRAM составляет всего 4 Кбайт, отсутствуют стандартные средства файлового ввода-вывода. Эти особенности требуют переработки модели использования памяти.

В процессе портирования часть работы связана с устранением зависимостей от POSIX-совместимой среды. Функции вывода в файл заменяются на формирование буфера в оперативной памяти, а интерфейс командной строки — на набор функций, вызываемых из основного цикла

микроконтроллера. Все обращения к динамической памяти и глобальным массивам адаптируются к разметке внешнего ОЗУ.



Рисунок 11 — Схема алгоритма синтеза речи SAM

2.4.1 Устранение ошибок сборки

Исходный код SAM организован в несколько файлов, но многие переменные и функции предполагаются глобальными и не имеют явных прототипов. Для инкрементальной сборки с разделением на объектные файлы код должен удовлетворять более строгим требованиям, поэтому на первом этапе портирования возникают многочисленные предупреждения и ошибки линковки. Для их устранения модули синтезатора разделяются на заголовочные и исходные файлы: в заголовках объявляются внешние функции и глобальные массивы с указанием принадлежности к определенной секции памяти, а единственное определение каждой сущности остаётся в одном .с-файле.

Глобальные таблицы фонем и параметров синтеза, к которым обращаются несколько модулей, объявляются в заголовках с ключевым словом `extern`, а в одном исходнике получают фактическое определение. Вспомогательные функции, используемые только внутри одного файла, помечаются как `static`, что предотвращает конфликт имён на этапе линковки и позволяет оптимизатору выполнять их развёртку. Такой пересмотр структуры проекта делает код более модульным и облегчает последующую отладку.

2.4.2 Работа с памятью программы

Весь код (макросы и одна вспомогательная функция) для удобства вынесены в отдельный заголовочный файл `memmanagment.h`, который затем подключается к остальным файлам проекта и обеспечивает поддержку работы с памятью данных. Содержимое `memmanagment.h` показано в листинге 1.

Листинг 1 — Заголовочный файл `memmanagment.h`

```
#ifndef MEMMANAGMENT_H
#define MEMMANAGMENT_H

#include <avr/pgmspace.h>
#include <stdio.h>
#include <string.h>

#include "uart.h"

#define DATAMEM __attribute__((section(".data")))
#define NOINITMEM __attribute__((section(".noinit")))

#define printf(...) _PRINTF_IMPL(__VA_ARGS__)(__VA_ARGS__)
#define _PRINTF_IMPL(...) \
    \
    _GET_MACRO(__VA_ARGS__, _PRINTF_IMPL_1, _PRINTF_IMPL_1, \
    _PRINTF_IMPL_1, \
    _PRINTF_IMPL_1, _PRINTF_IMPL_1, _PRINTF_IMPL_1, \
    _PRINTF_IMPL_1, \
    _PRINTF_IMPL_1, _PRINTF_IMPL_1, _PRINTF_IMPL_0)
#define _GET_MACRO(_fmt, _1, _2, _3, _4, _5, _6, _7, _8, _9, \
NAME, ...) NAME
#define _PRINTF_IMPL_0(fmt) \
    do { \
        static const char MSG[] PROGMEM = fmt; \
        printf(copy_pgm_str(MSG)); \
    } while (0)
#define _PRINTF_IMPL_1(fmt, ...) \
```

Продолжение листинга 1

```
do {
    static const char MSG[] PROGMEM = fmt; \
    printf(copy_pgm_str(MSG), __VA_ARGS__); \
} while (0)

extern char COPY_PGM_BUFFER[256] NOINITMEM;
const char* copy_pgm_str(const char* pgm_str);

#endif
```

Макросы `DATAMEM` и `NOINITMEM` служат для явного указания секции размещения глобальных данных. Атрибут `DATAMEM` принудительно помещает переменную в стандартную инициализируемую секцию `.data`, которая загружается во внутреннюю SRAM при старте программы. Атрибут `NOINITMEM` размещает объект в секции `.noinit`, которая не очищается и не инициализируется кодом старта, так как сдвинута в область внешней памяти (конфигурация внешней памяти осуществляется уже после инициализации переменных).

Особое внимание уделяется переопределению функции «`printf`». Проблем заключается в необходимости перемещения всех строковых литералов (по большей части используемых в функции «`printf`») в флеш-память для освобождения памяти данных. Макрос «`printf(...)`» разворачивается препроцессором в последовательность макросов «`_PRINTF_IMPL`» и «`_GET_MACRO`», которые по числу аргументов выбирают один из двух вариантов реализации: с параметрами форматирования или без них. В обоих случаях строка формата объявляется как `static const` с атрибутом «`PROGMEM`», то есть сохраняется во флеш-памяти микроконтроллера и не занимает оперативное ОЗУ. Только во время непосредственного использования той или иной строки функция «`copy_pgm_str`» копирует строку формата из программной памяти во временный буфер «`COPY_PGM_BUFFER`», размещённый в секции «`.noinit`» во внешнем ОЗУ, и возвращает указатель на этот буфер. Благодаря этому стандартная функция «`printf`» из `libc`, ожидающая строку в адресном пространстве данных, может использоваться

без модификации (что значительно снижает количество изменений в исходном коде SAM), а постоянные строки не расходуют объём внутренней SRAM.

Реализация функции «copy_pgm_str» использует функцию «strcpy_P» из заголовка «avr/pgmspace.h» для побайтного копирования строки из программной памяти в буфер «COPY_PGM_BUFFER» и возвращает указатель на него. Размер буфера выбран равным 256 байтам.

Благодаря указанным выше макросам можно гибко и явно определять структуру памяти. На рисунке 12 показано адресное пространство программы, подписаны различные секции программы и переменные линкера, которые отвечают за разметку.

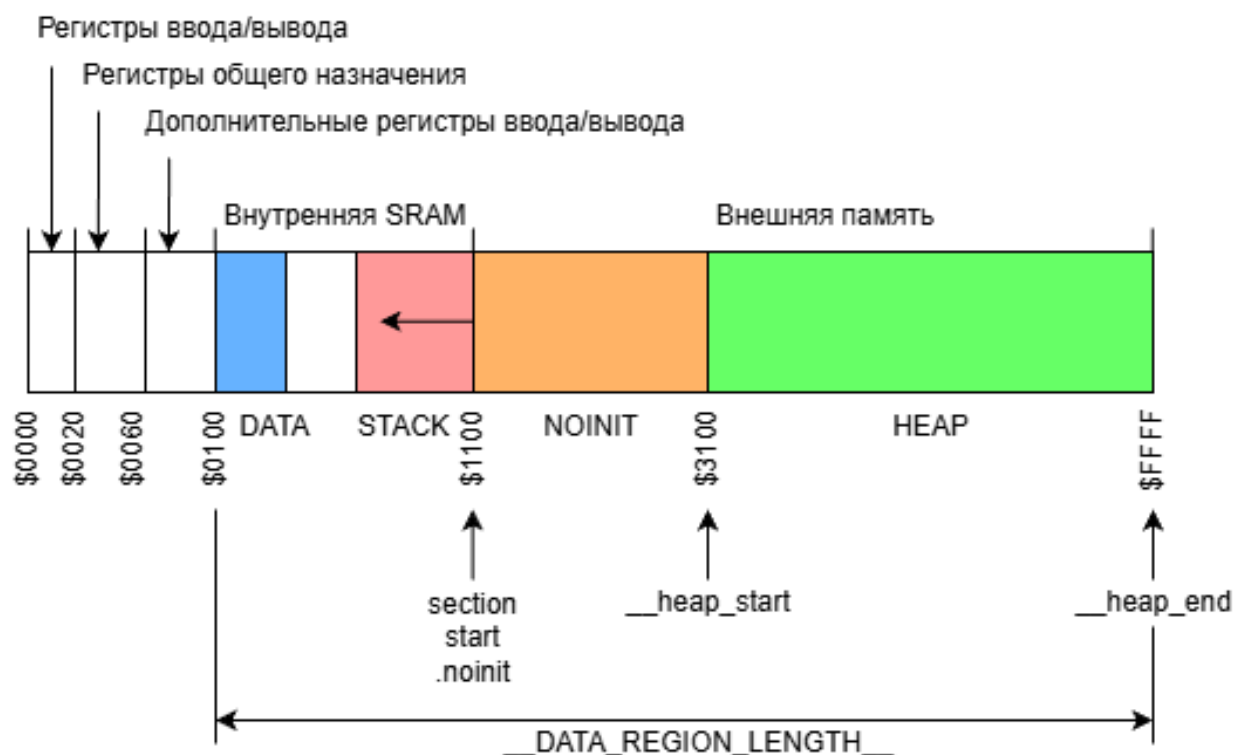


Рисунок 12 — Разметка адресного пространства данных

Адресное пространство данных микроконтроллера ATmega128A делится на регистровую область, область регистров ввода-вывода и оперативную память. В нижней части пространства расположены регистры общего назначения и регистры периферии, далее следует внутренняя SRAM объёмом 4 Кбайт. На рисунке эта область объединена в секцию DATA, где размещаются инициализируемые глобальные переменные, и в область стека STACK, растущего навстречу данным. Начиная с адреса 0x1100 доступна

внешняя SRAM вплоть до конца адресного пространства. В этой области последовательно располагаются секция «.noinit» (8 кБайт) и динамическая куча HEAP (52 кБайт), используемая реализацией «malloc» для аллокации памяти под звуковой буфер.

Помимо разметки данных по секциям, для непосредственного назначения диапазонов адресов для секций данных необходимо передать несколько аргументов линкеру через переменные сборочного файла. Фрагмент файла с аргументами представлен в листинге 2.

Листинг 2 — Выдержка из файла Makefile с флагами линкера

```
LDFLAGS += -Wl,--defsym,__DATA_REGION_LENGTH__=0xffa0,--  
section-start=.noinit=0x801100,--  
defsym=__heap_start=0x803100,--defsym=__heap_end=0x80ffff
```

Переменная линкера «__DATA_REGION_LENGTH__» задаёт суммарную длину доступной области данных. В проекте для неё выбирается значение 0xffa0, что соответствует использованию всего диапазона от начала внутренней SRAM до конца внешней памяти, за вычетом служебной области. Параметр «--section-start=.noinit=0x801100» указывает начальный адрес секции «.noinit» в расширенной области данных; фактический физический адрес соответствует 0x1100. Символы «__heap_start» и «__heap_end» определяют границы кучи, из которой выделяется память под выходные звуковые буферы.

2.5 Модуль трансляции

Модуль трансляции необходим для перевода русских букв в английские транскрипции, так как алгоритмы синтеза работают только с английским языком и ожидают на входе текст в латинском алфавите. Заголовочный файл транслятора показан в листинге 3.

Функция translate принимает указатель на выходной буфер, исходную строку и максимально допустимый размер результата (для корректной обработки ошибки переполнения). На вход подаётся уже приведённый к верхнему регистру текст. В процессе работы функция последовательно просматривает символы исходной строки, для каждого символа определяет, относится ли он к русскому алфавиту, и в зависимости от результата либо копирует символ напрямую, либо заменяет его на одну или

несколько латинских букв согласно таблице транслитерации. Результат всегда завершается нулевым байтом, а запись не выходит за пределы указанного размера буфера.

Листинг 3 — Заголовочный файл translator.h

```
#ifndef TRANSLATOR_H
#define TRANSLATOR_H

void translate(char* dst, char* src, int dstsize);

#endif
```

Алгоритм модуля трансляции представлен на рисунке 13. Для каждого символа входного слова проверяется, является ли он русской буквой. Если это не так, символ без изменений добавляется в выходную строку. Если символ относится к кириллице, по таблице соответствий выбирается его транскрипция, после чего в выходную строку поочерёдно добавляются все символы этой транскрипции.

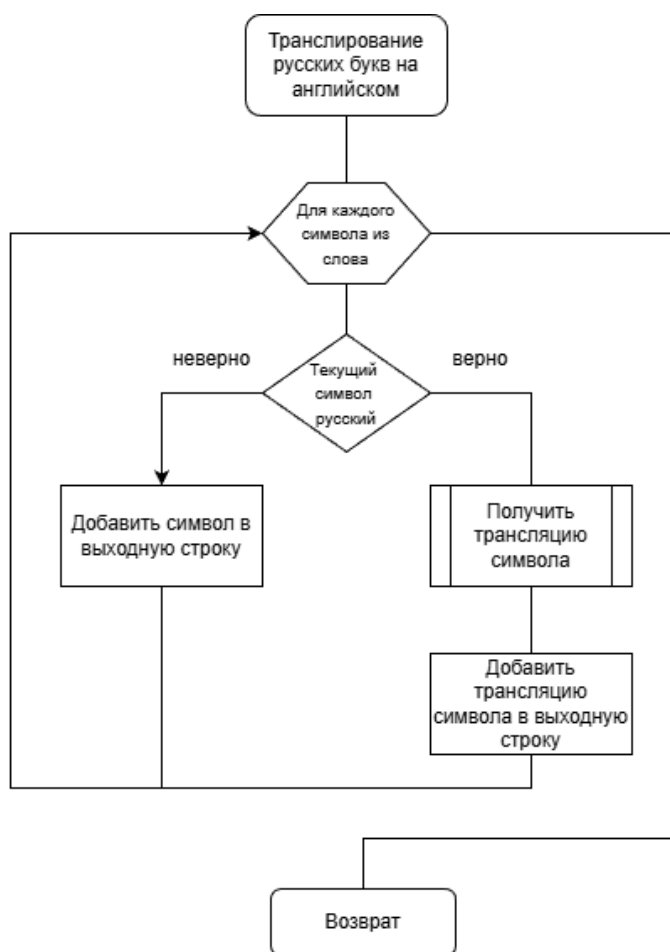


Рисунок 13 — Схема алгоритмов модуля трансляции

Модуль трансляции используется в функции «Сказать слово» перед вызовом синтезатора. В результате SAM получает на вход строку, содержащую лишь латинские буквы, а особенности произношения русских слов моделируются с помощью заранее подобранных комбинаций английских букв.

2.6 Асинхронные операции ввода-вывода

Так как синтез речи является тяжёлой задачей, встаёт вопрос о грамотном использовании ограниченных ресурсов. Для увеличения полезного процента во времени утилизации вычислительного ядра активно применяются асинхронные алгоритмы ввода-вывода, реализованные при помощи прерываний. Таким образом удаётся минимизировать время простаивания ядра в циклах ожидания, а освободившиеся ресурсы направить на вычислительно сложные этапы работы синтезатора. Рисунок 14 иллюстрирует, как используется процессорное время.

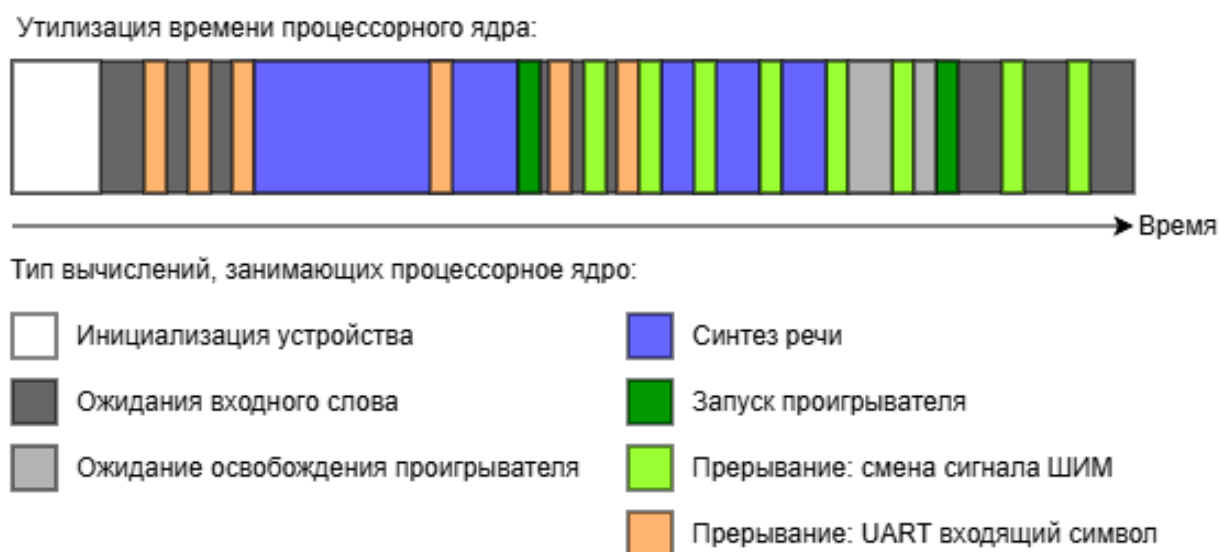


Рисунок 14 — Использование процессорного времени

На диаграмме можно выделить несколько характерных фаз. В начале работы ядро кратковременно занято инициализацией устройства: настройкой внешней памяти, интерфейса UART, таймеров. Затем, после ожидания ввода слова для озвучивания, следует продолжительный интервал, в течение которого выполняется синтез речи; в это время основное ядро полностью загружено арифметическими вычислениями и обработкой таблиц фонем, звуков. Параллельно с этим приходящие по UART символы обрабатываются

в прерывании и попадают в кольцевой буфер: таким образом входящие слова принимаются микроконтроллеров в любой момент времени, а не только когда микроконтроллер свободен.

После завершения синтеза запускается проигрыватель звукового файла. С этого момента основное ядро почти не участвует в выводе звука: обновление значения регистра сравнения ШИМ происходит в прерывании таймера, причём обработчик прерывания выполняется очень быстро и занимает лишь небольшие фрагменты времени между остальными задачами. Параллельно с проигрыванием текущего слова в случае готовности буфера начинается обработка следующего слова с прерываниями на обновления ШИМ. Временные интервалы ожидания освобождения проигрывателя и получения нового слова сведены к минимуму, так как вместо активного опроса периферии используются прерывания и программные флаги состояния. Такое распределение нагрузки позволяет максимально использовать процессорное время для тяжёлых вычислений синтеза речи, а операции ввода-вывода выполняются фрагментарно и не мешают синтезу.

2.6.1 Модуль асинхронного буфера входного потока

Принято решение реализовать получение входных данных при помощи специального входного буфера, который позволяет не терять данные, если они приходят уже во время активной вычислительной работы, а также снижает долю времени ожидания процессора. Данные в буфер попадают асинхронно из обработчика прерывания по приёму UART, а при готовности микроконтроллера взять в работу следующее слово считываются из буфера и передаются в обработку. Заголовочный файл в листинге 4 показывает интерфейс буфера.

Структура `buffer_t` описывает кольцевой буфер фиксированного размера `BUFFER_SIZE`. Поле `buf` хранит указатель на область памяти, выделенную динамически при вызове `make_buffer`. Поля `begin` и `end` содержат индексы начала и конца данных в буфере и изменяются по модулю `BUFFER_SIZE`, что образует кольцевую структуру. Функция `make_buffer` выделяет память

под массив символов, инициализирует индексы нулём и возвращает готовую структуру.

Функция `buffer_is_empty` вычисляет разность индексов с учётом циклического переполнения и возвращает признак пустоты буфера. Запись символа выполняется функцией `buffer_write`: символ помещается в позицию `end`, после чего индекс сдвигается вперёд по модулю `BUFFER_SIZE`. Функция `buffer_read` реализует блокирующее чтение: если в буфере нет символа для выдачи, выполняется ожидание, затем символ берётся из позиции `begin`, а индекс начала данных увеличивается. Объявление аргумента как `volatile buffer_t*` гарантирует, что компилятор не оптимизирует обращения к полям структуры, поскольку они могут изменяться как в основном коде, так и в обработчике прерывания.

Листинг 4 — Заголовочный файл `buffer.h`

```
#ifndef BUFFER_H
#define BUFFER_H

#define BUFFER_SIZE 256

typedef struct buffer_t {
    char* buf;
    int begin;
    int end;
} buffer_t;

buffer_t make_buffer();

int buffer_is_empty(volatile buffer_t* buffer);

void buffer_write(volatile buffer_t* buffer, char ch);

char buffer_read(volatile buffer_t* buffer);

#endif
```

Логическая модель работы буфера показана на рисунке 15. При чтении сначала проверяется, пуст ли буфер; если данных нет, алгоритм остаётся в состоянии ожидания до появления новых символов. При записи, согласно схеме, производится проверка наличия свободного места, затем в случае наличия свободного места – запись в буфер и инкрементация поля смещения.

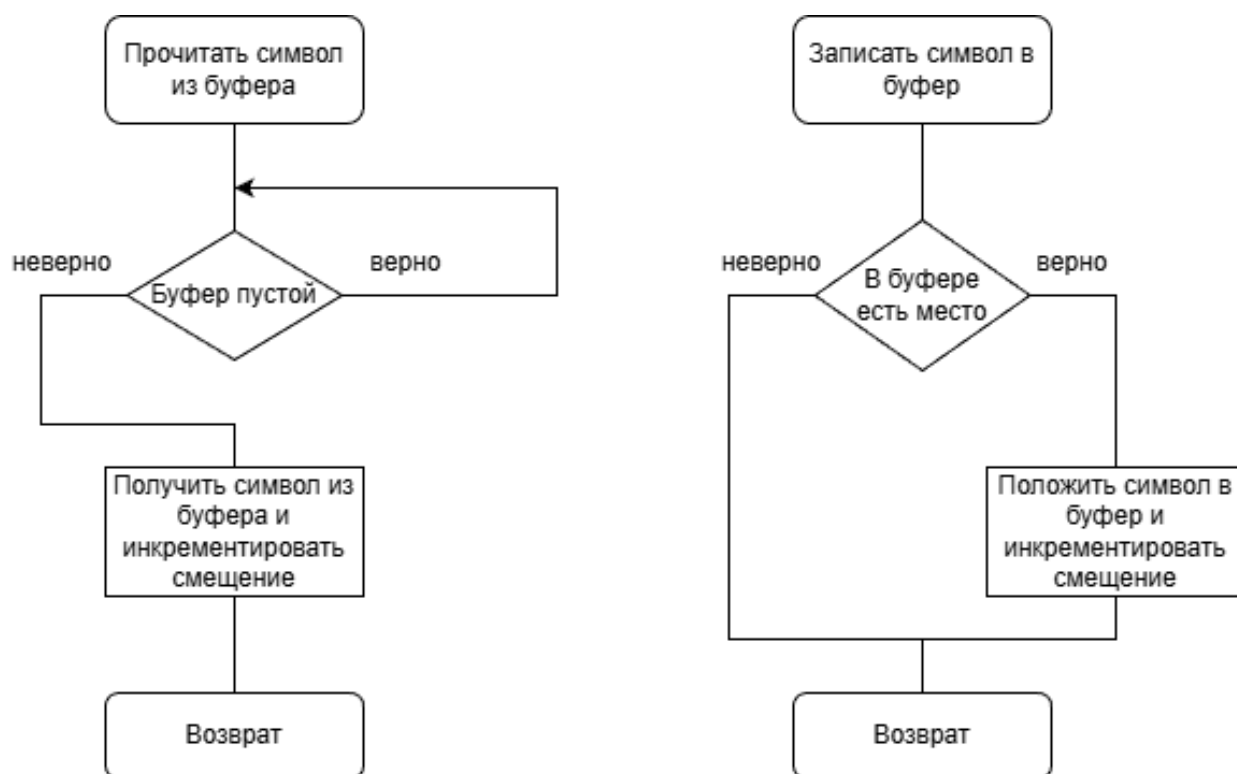


Рисунок 15 — Схема алгоритмов буфера входного потока

2.6.2 Модуль асинхронного проигрывателя звукового файла

Асинхронный проигрыватель, интерфейс которого представлен в листинге 5, снимает задачу тяжёлого вывода и, совместно с асинхронным входным буфером, позволяет процессору проводить минимальное количество времени в состоянии ожидания.

Структура `player_data_t` описывает звуковой файл в памяти и содержит указатель на буфер выборок и его размер в байтах. Тип `callback_func_t` задаёт прототип функции обратного вызова, которая вызывается после завершения воспроизведения (функция нужна, так как запуск плеера не блокирует поток исполнения, при этом возможно должна выполняться какая-то работа после окончания проигрывания); в контексте устройства – в `callback`-функции выполняется освобождение памяти из-под использованного звукового буфера. Функция `init_player` готовит периферию к работе проигрывателя: настраивает вывод ОС0 в режим быстрый ШИМ, задаёт параметры таймера 1 и включает прерывание по совпадению с регистром OCR1A.

Функция `make_player_data` формирует структуру `player_data_t` из указателя и размера, упрощая передачу данных между модулями. Основная функция `play` является неблокирующей, принимает структуру с описанием

буфера и указатель на callback. Внутри модуля происходят сохранение этих параметров во внутренние статические переменные, установка флага блокировки проигрывателя и сброс счётчика позиции, после чего запускается таймер. С этого момента воспроизведение происходит полностью в обработчике прерывания.

Листинг 5 — Заголовочный файл player.h

```
#ifndef PLAYER_H
#define PLAYER_H

typedef struct player_data_t {
    char* buffer;
    unsigned int size;
} player_data_t;

typedef void (*callback_func_t)(volatile player_data_t* data);

void init_player();

player_data_t make_player_data(char* buffer, unsigned int
size);

void play(player_data_t data, callback_func_t callback);

int is_player_blocked();

void wait_player();

#endif
```

Функция `is_player_blocked` возвращает текущее состояние флага блокировки и используется для проверки готовности проигрывателя к приёму нового буфера. Функция `wait_player` реализует простое ожидание окончания воспроизведения, пока флаг блокировки не сброшен; она вызывается, чтобы гарантировать, что предыдущая фраза полностью прозвучала перед запуском следующей.

Внутренняя логика проигрывателя иллюстрируется на рисунке 16. Во время воспроизведения таймер генерирует прерывания с частотой, соответствующей частоте дискретизации звукового сигнала (22050 Гц). В обработчике прерывания очередной байт звукового буфера записывается в регистр OCR0, что приводит к изменению коэффициента заполнения ШИМ

и, после фильтрации, к формированию аналогового звука на выходе. Затем увеличивается смещение в буфере; когда оно достигает размера звукового файла, таймер и ШИМ выключаются, флаг блокировки сбрасывается и вызывается callback-функция, освобождающая ресурсы.

Такая организация позволяет основному коду не заниматься непосредственной выдачей выборков: процессор лишь изредка выполняет короткий обработчик прерывания и остаётся свободен для синтеза следующего фрагмента речи или обработки команд, что в совокупности обеспечивает высокую утилизацию вычислительного ядра при ограниченных ресурсах микроконтроллера.

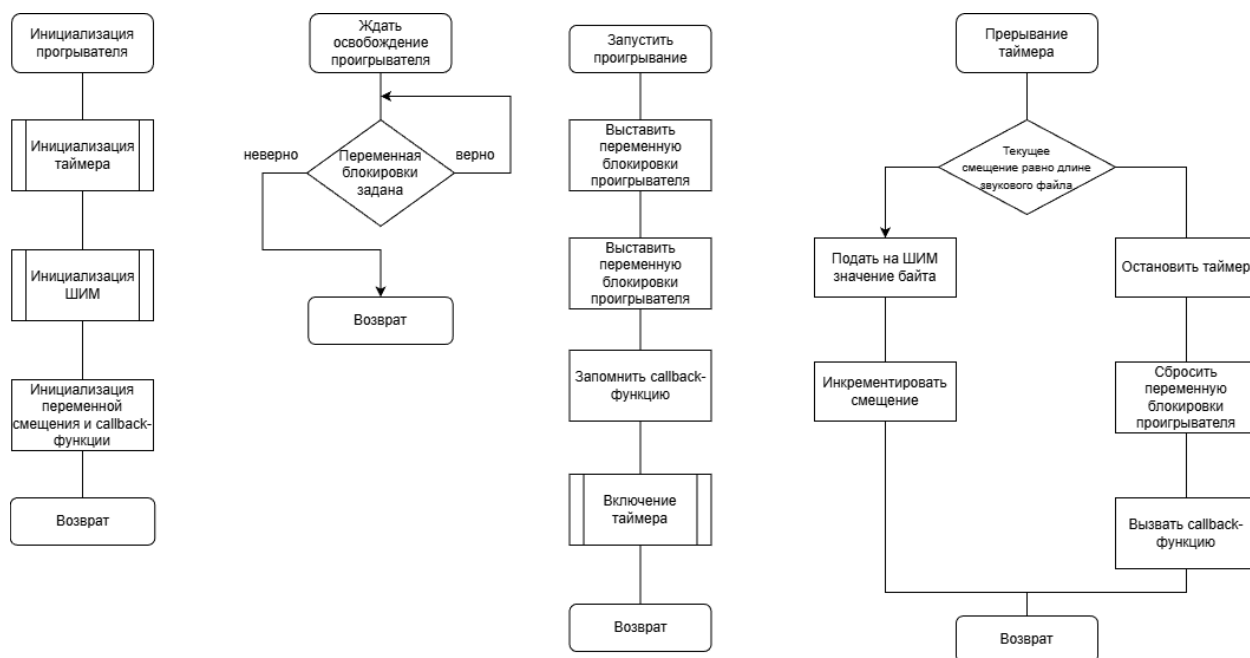


Рисунок 16 — Схема алгоритмов проигрывателя

Таким образом реализуется программная часть устройства.

ЗАКЛЮЧЕНИЕ

В рамках курсовой работы разработано устройство синтеза речи на основе TTS программы SAM для русского и английского языков на основе микроконтроллера ATMega128A. Устройство может быть подключено и использовано при помощи протокола UART.

В процессе проектирования созданы функциональная и принципиальная схемы устройства, портирован код SAM под платформу AVR, написаны дополнительные модули-обвязки для корректной интеграции TTS-алгоритмов и аппаратных средств.

Разработанное устройство соответствует требованиям технического задания и выполняют функцию синтеза речи на русском и английском языках.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Репозиторий с исходным кодом TTS-программы SAM [Электронный ресурс]. URL: <https://github.com/s-macke/SAM> (дата обращения: 30.09.2025).
2. Datasheet ATmega128A [Электронный ресурс]. URL: https://www1.microchip.com/downloads/en/DeviceDoc/Atmel-8151-8-bit-AVR-ATmega128A_Datasheet.pdf (дата обращения: 10.10.2025).
3. USB 2.0 Specification [Электронный ресурс]. URL: <https://www.usb.org/document-library/usb-20-specification> (дата обращения: 09.10.2025).
4. Datasheet SNx4HC573A [Электронный ресурс]. URL: <https://static.chipdip.ru/lib/973/DOC054973478.pdf> (дата обращения: 10.10.2025).
5. Статья об интерфейсе внешней памяти XMEM [Электронный ресурс]. URL: http://www.gaw.ru/html.cgi/txt/doc/micros/avr/arh128/2_1.htm (дата обращения: 15.10.2025).
6. Datasheet AS6C1008 [Электронный ресурс]. URL: <https://static.chipdip.ru/lib/708/DOC011708899.pdf> (дата обращения: 10.10.2025).
7. Datasheet CH340 [Электронный ресурс]. URL: <https://www.mpja.com/download/35227cpdata.pdf> (дата обращения: 10.10.2025).
8. Datasheet PAM8403 [Электронный ресурс]. URL: <https://static.chipdip.ru/lib/050/DOC035050562.pdf> (дата обращения: 10.10.2025).
9. Datasheet FBS1850 [Электронный ресурс]. URL: <https://static.chipdip.ru/lib/610/DOC061610482.pdf> (дата обращения: 10.10.2025).
10. Электронный учебно-методический комплекс по дисциплине ЦСП [Электронный ресурс]. URL: <https://biik.ru/uchebnik/csp/page29.html> (дата обращения: 20.10.2025).

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ
Листов 130

Листинг А.1 — Содержимое файла Makefile

```
#####  
# Makefile for the project handheld-sam #  
#####  
  
## General Flags  
PROJECT=handheld-sam  
MCU=atmega128a  
MCU_SHORT=m128  
F_CPU=8000000UL  
  
#Dirs  
BUILDDIR=build  
SRCDIR=src  
INCLUDEDIR=include  
  
# Source files  
SOURCES += $(SRCDIR)/main.c  
SOURCES += $(SRCDIR)/player.c  
SOURCES += $(SRCDIR)/uart.c  
SOURCES += $(SRCDIR)/buffer.c  
SOURCES += $(SRCDIR)/memmanagment.c  
SOURCES += $(SRCDIR)/translator.c  
  
# SAM source files  
SOURCES += $(SRCDIR)/sam.c  
SOURCES += $(SRCDIR)/debug_sam.c  
SOURCES += $(SRCDIR)/reciter.c  
SOURCES += $(SRCDIR)/ReciterTabs.c  
SOURCES += $(SRCDIR)/render.c  
SOURCES += $(SRCDIR)/RenderTabs.c  
SOURCES += $(SRCDIR)/SamTabs.c  
  
# Objects and deps  
TARGET=$(BUILDDIR)/$(PROJECT).elf  
TARGET_HEX=$(BUILDDIR)/$(PROJECT).hex  
OBJECTS=$(patsubst $(SRCDIR)/%.c,$(BUILDDIR)/%.o,$(SOURCES))  
DEPS=$(OBJECTS:.o=.d)  
  
## Compilers  
CC =avr-gcc  
CPP=avr-g++  
  
## Options common to compile, link and assembly rules  
COMMON=-mmcu=$(MCU)  
  
## Compile options  
CFLAGS=$(COMMON)
```

Продолжение листинга A.1

```
CFLAGS += -Wall -gdwarf-2 -std=gnu99 -DF_CPU=$(F_CPU) -Os \
        -funsigned-char -funsigned-bitfields -fpack-struct -
fshort-enums
CFLAGS += -MD -MP -MF $(BUILDDIR)/$(F).d
CFLAGS += -I$(INCLUDEDIR)

## Assembly specific flags
ASMFLAGS=$(COMMON) $(CFLAGS) -x assembler-with-cpp -Wa,-
gdwarf2

## Linker flags
LDFLAGS=$(COMMON)
LDFLAGS += -Wl,--defsym,__DATA_REGION_LENGTH__=0xffa0,--
section-start=.noinit=0x801100,--
defsym=__heap_start=0x803100,--defsym=__heap_end=0x80ffff

## Hex/Eep flags
HEX_FLASH_FLAGS=-R .eeprom -R .fuse -R .lock -R .signature

HEX_EEPROM_FLAGS=-j .eeprom \
        --set-section-flags=.eeprom="alloc,load" \
        --change-section-lma .eeprom=0 \
        --no-change-warnings

## Build all
all: $(TARGET) \
    $(TARGET:.elf=.hex) \
    $(TARGET:.elf=.eep) \
    $(TARGET:.elf=.lss) \
    size

## Ensure build directory exists
$(BUILDDIR):
    mkdir $(BUILDDIR)

## Compile all .c → build/*.o
$(BUILDDIR)/%.o: $(SRCDIR)/%.c | $(BUILDDIR)
    $(CC) $(INCLUDES) $(CFLAGS) -c $< -o $@

## Link
$(TARGET): $(OBJECTS)
    $(CC) $(LDFLAGS) -Wl,-Map,$(TARGET:.elf=.map) $(OBJECTS)
    $(LIBDIRS) $(LIBS) -o $@

## Make HEX
$(BUILDDIR)/%.hex: $(BUILDDIR)/%.elf
```

Продолжение листинга A.1

```
avr-objcopy -O ihex $(HEX_FLASH_FLAGS) $< $@

## EEPROM HEX
$(BUILDDIR)/%.eep: $(BUILDDIR)/%.elf
    -avr-objcopy $(HEX_EEPROM_FLAGS) -O ihex $< $@ || exit 0

## LSS
$(BUILDDIR)/%.lss: $(BUILDDIR)/%.elf
    avr-objdump -h -S $< > $@

size: $(TARGET)
    @echo
    @avr-size ${TARGET}

flash: all
    avrdude -c usbasp -p $(MCU_SHORT) -U flash:w:$(TARGET_HEX):i

## Clean
.PHONY: clean
clean:
    ifeq ($(OS),Windows_NT)
        rmdir /S /Q $(BUILDDIR)
    else
        rm -rf $(BUILDDIR)
    endif

## Include dependency files
-include $(DEPS)
```

Листинг A.2 — Содержимое файла include/buffer.h

```
#ifndef BUFFER_H
#define BUFFER_H

#define BUFFER_SIZE 256

typedef struct buffer_t {
    char* buf;
    int begin;
    int end;
} buffer_t;

buffer_t make_buffer();

int buffer_is_empty(volatile buffer_t* buffer);
```

Продолжение листинга A.2

```
void buffer_write(volatile buffer_t* buffer, char ch);

char buffer_read(volatile buffer_t* buffer);

#endif
```

Листинг A.3 — Содержимое файла include/debug_sam.h

```
#ifndef DEBUG_H
#define DEBUG_H

#include <stdio.h>
#include <string.h>

#include "SamTabs.h"
#include "memmanagment.h"

extern int debug DATAMEM;

void PrintPhonemes(unsigned char* phonemeindex, unsigned char*
phonemeLength,
                  unsigned char* stress);
void PrintOutput(unsigned char* flag, unsigned char* f1,
unsigned char* f2,
                  unsigned char* f3, unsigned char* a1,
unsigned char* a2,
                  unsigned char* a3, unsigned char* p);

void PrintRule(int offset);

#endif
```

Листинг A.4 — Содержимое файла include/help.h

```
#ifndef HELP_H
#define HELP_H

#include <inttypes.h>
#include <stdio.h>

#include "debug_sam.h"
#include "memmanagment.h"

static void print_usage() {
    printf("          VOWELS                                VOICED
CONSONANTS      \n");
```

Продолжение листинга A.4

```

    printf("IY          f(ee) t          R
red          \n");
    printf("IH          p(i) n          L
allow        \n");
    printf("EH          beg          W
away         \n");
    printf("AE          Sam          W
whale        \n");
    printf("AA          pot          Y
you          \n");
    printf("AH          b(u) dget      M
Sam          \n");
    printf("AO          t(al) k        N
man          \n");
    printf("OH          cone          NX
so(ng)       \n");
    printf("UH          book          B
bad          \n");
    printf("UX          l(oo) t        D
dog          \n");
    printf("ER          bird          G
again        \n");
    printf("AX          gall(o) n      J
judge        \n");
    printf("IX          dig(i) t        Z
zoo          \n");
    printf("          ZH          plea(s) ure
\n");
    printf("    DIPHTHONGS          V
seven        \n");
    printf("EY          m(a) de          DH
(th) en      \n");
    printf("AY          h(igh)          \n");
    printf("OY          boy          \n");
    printf("AW          h(ow)          UNVOICED
CONSONANTS   \n");
    printf("OW          slow          S
Sam          \n");
    printf("UW          crew          Sh
fish         \n");
    printf("          F
fish         \n");
    printf("          TH
thin         \n");
    printf(" SPECIAL PHONEMES          P
poke         \n");

```

Продолжение листинга A.4

```
        printf("UL          sett(le) (=AXL)          T
talk          \n");
        printf("UM          astron(omy) (=AXM)        K
cake          \n");
        printf("UN          functi(on) (=AXN)         CH
speech        \n");
        printf("Q           kitt-en (glottal stop)    /H
a(h)ead      \n");
    }

#endif
```

Листинг A.5 — Содержимое файла include/memmanagment.h

```
#ifndef MEMMANAGMENT_H
#define MEMMANAGMENT_H

#include <avr/pgmspace.h>
#include <stdio.h>
#include <string.h>

#include "uart.h"

#define DATAMEM __attribute__((section(".data")))
#define NOINITMEM __attribute__((section(".noinit")))

#define printf(...) _PRINTF_IMPL(__VA_ARGS__)(__VA_ARGS__)
#define _PRINTF_IMPL(...)
\
    _GET_MACRO(__VA_ARGS__, _PRINTF_IMPL_1, _PRINTF_IMPL_1,
    _PRINTF_IMPL_1, \
        _PRINTF_IMPL_1, _PRINTF_IMPL_1, _PRINTF_IMPL_1,
    _PRINTF_IMPL_1, \
        _PRINTF_IMPL_1, _PRINTF_IMPL_1, _PRINTF_IMPL_0)
#define _GET_MACRO(_fmt, _1, _2, _3, _4, _5, _6, _7, _8, _9,
NAME, ...) NAME
#define _PRINTF_IMPL_0(fmt) \
    do { \
        static const char MSG[] PROGMEM = fmt; \
        printf(copy_pgm_str(MSG)); \
    } while (0)
#define _PRINTF_IMPL_1(fmt, ...) \
    do { \
        static const char MSG[] PROGMEM = fmt; \
        printf(copy_pgm_str(MSG), __VA_ARGS__); \
    } while (0)
```


Продолжение листинга А.5

```
extern char COPY_PGM_BUFFER[256] NOINITMEM;
const char* copy_pgm_str(const char* pgm_str);

#endif
```

Листинг А.6 — Содержимое файла include/player.h

```
#ifndef PLAYER_H
#define PLAYER_H

typedef struct player_data_t {
    char* buffer;
    unsigned int size;
} player_data_t;

typedef void (*callback_func_t)(volatile player_data_t* data);

void init_player();

player_data_t make_player_data(char* buffer, unsigned int
size);

void play(player_data_t data, callback_func_t callback);

int is_player_blocked();

void wait_player();

#endif
```

Листинг А.7 — Содержимое файла include/reciter.h

```
#ifndef RECITER_C
#define RECITER_C

#include "ReciterTabs.h"
#include "debug_sam.h"
#include "memmanagment.h"

//int TextToPhonemes(char *input, char *output);

int TextToPhonemes(unsigned char *input);

#endif
```

Листинг А.8 — Содержимое файла include/ReciterTabs.h

```
#ifndef RECITERTABS_H
#define RECITERTABS_H

#include "memmanagment.h"

extern const unsigned char tab36376[] PROGMEM;

extern const unsigned char rules[] PROGMEM;

extern const unsigned char rules2[] PROGMEM;

extern const unsigned char tab37489[] PROGMEM;

extern const unsigned char tab37515[] PROGMEM;

#endif
```

Листинг А.9 — Содержимое файла include/RenderTabs.h

```
#ifndef RENDERTABS_H
#define RENDERTABS_H

#include "memmanagment.h"

extern const unsigned char tab48426[5] PROGMEM;

extern const unsigned char tab47492[] PROGMEM;

extern const unsigned char amplitudeRescale[] PROGMEM;

extern const unsigned char blendRank[] PROGMEM;

extern const unsigned char outBlendLength[] PROGMEM;

extern const unsigned char inBlendLength[] PROGMEM;

extern unsigned char sampledConsonantFlags[] DATAMEM;

extern unsigned char freq1data[] DATAMEM;

extern unsigned char freq2data[] DATAMEM;

extern const unsigned char freq3data[] PROGMEM;

extern const unsigned char ampl1data[] PROGMEM;
```

Продолжение листинга A.9

```
extern const unsigned char ampl2data[] PROGMEM;

extern const unsigned char ampl3data[] PROGMEM;

extern const signed char sinus[256] PROGMEM;

extern const unsigned char rectangle[] PROGMEM;

extern const unsigned char sampleTable[0x500] PROGMEM;

#endif
```

Листинг A.10 — Содержимое файла include/sam.h

```
#ifndef SAM_H
#define SAM_H

#include "SamTabs.h"
#include "debug_sam.h"
#include "render.h"
#include "reciter.h"
#include "memmanagment.h"

#define SAMPLE_RATE 22050

extern int singmode DATAMEM;

extern

void SetInput(char *_input);
void SetSpeed(unsigned char _speed);
void SetPitch(unsigned char _pitch);
void SetMouth(unsigned char _mouth);
void SetThroat(unsigned char _throat);
void EnableSingmode();
void EnableDebug();

int SAMMain();

char* GetBuffer();
uint32_t GetBufferLength();

//char input[]={"/HAALAOAO MAYN NAAMAEAE IHSTT SAEBAASTTIHAAN
\x9b\x9b\0"};
//unsigned char input[]={"/HAALAOAO \x9b\0"};
```

Продолжение листинга А.10

```
//unsigned char input[]={"AA \x9b\0"};
//unsigned char input[] = {"GUH5DEHN TAEG\x9b\0"};

//unsigned char input[]={"AY5 AEM EY TAO4LXKIHNX
KAX4MPYUX4TAH. GOW4 AH/HEH3D PAHNK.MEYK MAY8 DEY.\x9b\0"};
//unsigned char input[]={"/HEH3LOW2, /HAW AH YUX2 TUXDEY. AY /
HOH3P YUX AH FIYLIHNX OW4 KEY.\x9b\0"};
//unsigned char input[]={"/HEY2, DHIHS IH3Z GREY2T. /HAH /
HAH /HAH.AYL BIY5 BAEK.\x9b\0"};
//unsigned char input[]={"/HAH /HAH /HAH \x9b\0"};
//unsigned char input[]={"/HAH /HAH /HAH.\x9b\0"};
//unsigned char input[]={" .TUW BIY5Y3,, OHR NAA3T - TUW
BIY5IYIY., DHAE4T IHZ DHAH KWEH4SCHAHN.\x9b\0"};
//unsigned char input[]={"/HEY2, DHIHS \x9b\0"};

//unsigned char input[]={" IYIHEHAEEAAHAOOHUHUXERAXIX
\x9b\0"};
//unsigned char input[]={" RLWWYMNNXBDGJZZHVDH \x9b\0"};
//unsigned char input[]={" SSHFTHPTKCH/H \x9b\0"};

//unsigned char input[]={" EYAYOYAWOWUW ULUMUNQ YXWXRXLX/
XDX\x9b\0"};

#endif
```

Листинг А.11 — Содержимое файла include/SamTabs.h

```
#ifndef SAMTABS_H
#define SAMTABS_H

#include "memmanagment.h"

extern const unsigned char stressInputTable[] PROGMEM;

extern const unsigned char signInputTable1[] PROGMEM;

extern const unsigned char signInputTable2[] PROGMEM;

extern const unsigned char sam_flags[] PROGMEM;

extern const unsigned char sam_flags2[] PROGMEM;

extern const unsigned char phonemeStressedLengthTable[]
PGMEM;
```

Продолжение листинга A.11

```
extern const unsigned char phonemeLengthTable[] PROGMEM;

#endif
```

Листинг A.12 — Содержимое файла include/translator.h

```
#ifndef TRANSLATOR_H
#define TRANSLATOR_H

void translate(char* dst, char* src, int dstsize);

#endif
```

Листинг A.13 — Содержимое файла include/uart.h

```
#ifndef UART_H
#define UART_H

#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdio.h>

#include "memmanagment.h"

#define BAUD_RATE 9600UL
#define UBRR_VALUE ((F_CPU / (16UL * BAUD_RATE)) - 1)

extern FILE uart_stdout;
extern FILE uart_stdin;

void init_uart();

int putchar_uart(char c, FILE* stream);
int getchar_uart(FILE* stream);

#endif
```

Листинг A.14 — Содержимое файла include/xmem.h

```
#ifndef XMEM_H
#define XMEM_H

#include <avr/interrupt.h>
#include <avr/io.h>

static void init_xmem() {
    // XMCRA = (1 << SRW11) | (1 << SRW10) | (1 << SRW01) | (1
```

Продолжение листинга A.14

```
<< SRW00);  
    XMCRA = 0;  
    XMCRB = 0;  
    MCUCR = (1 << SRE);  
}  
  
#endif
```

Листинг A.15 — Содержимое файла src/buffer.c

```
#include "buffer.h"  
  
#include <stdlib.h>  
#include <stdio.h>  
  
buffer_t make_buffer() {  
    buffer_t buffer;  
    buffer.buf = (char*)malloc(BUFFER_SIZE);  
    buffer.begin = 0;  
    buffer.end = 0;  
    return buffer;  
}  
  
int buffer_is_empty(volatile buffer_t* buffer) {  
    return (buffer->end - buffer->begin + BUFFER_SIZE) %  
    BUFFER_SIZE == 0;  
}  
  
void buffer_write(volatile buffer_t* buffer, char ch) {  
    if ((buffer->end - buffer->begin + BUFFER_SIZE) %  
    BUFFER_SIZE == -1) {  
        return; // писать нельзя, буфер заполнен  
    }  
  
    buffer->buf[buffer->end] = ch;  
    buffer->end = (buffer->end + 1) % BUFFER_SIZE;  
}  
  
char buffer_read(volatile buffer_t* buffer) {  
    char ch;  
    // блокируемся, пока в буфер кто-нибудь не напишет  
    while (buffer_is_empty(buffer));  
    ch = buffer->buf[buffer->begin];  
    buffer->begin = (buffer->begin + 1) % BUFFER_SIZE;  
    return ch;  
}
```

Листинг A.16 — Содержимое файла src/main.c

```
#include <ctype.h>
#include <stdlib.h>
#include <string.h>

#include "help.h"
#include "player.h"
#include "sam.h"
#include "translator.h"
#include "uart.h"
#include "xmem.h"

#define INPUT_SIZE 255

static volatile char* current_word NOINITMEM;

// void init_led() { DDRB |= (1 << PB0); }
// void led_on() { PORTB &= ~(1 << PB0); }
// void led_off() { PORTB |= (1 << PB0); }

void init() {
    // xmem должен стоять первым
    init_xmem();
    // init_led();
    init_uart();
    init_player();
}

void player_callback(volatile player_data_t* data) {
    free(data->buffer);
    printf("%s\n", current_word);
}

void say(char* input, int phonetic) {
    strncat(input, " ", INPUT_SIZE);
    char translated[INPUT_SIZE] = "\0";

    if (debug) printf("input: %s\n", input);
    for (int i = 0; input[i] != 0; i++) input[i] =
toupper((int)input[i]);
    translate(translated, input, INPUT_SIZE);
    input = translated;
    if (debug) printf("translated: %s\n", input);

    if (debug) {
        if (phonetic) {
            printf("phonetic input: %s\n", input);
        }
    }
}
```

Продолжение листинга A.16

```
        } else {
            printf("text input: %s\n", input);
        }
    }

    if (!phonetic) {
        strncat(input, "[", INPUT_SIZE);
        if (!TextToPhonemes((unsigned char*)input)) {
            return;
        }
        if (debug) {
            printf("phonetic input: %s\n", input);
        }
    } else {
        strncat(input, "\\x9b", INPUT_SIZE);
    }

    // led_on();
    SetInput(input);
    if (!SAMMain()) {
        print_usage();
    }
    // led_off();

    if (debug) printf("length: %d\n", GetBufferLength() / 50);

    wait_player();
    current_word = input;
    play(make_player_data(GetBuffer(), GetBufferLength() /
50),
        player_callback);
}

void loop() {
    unsigned char number;
    int phonetic = 0;
    char input[INPUT_SIZE];
    int i;
    for (i = 0; i < INPUT_SIZE; ++i) input[i] = 0;
    debug = 0;

    printf("READY\n> ");

    while (1) {
        scanf("%255s", input);
        if (input[0] != '-') {
```


Продолжение листинга А.16

```
        say(input, phonetic);
    } else if (strcmp(input, "-help") == 0) {
        print_usage();
    } else if (strcmp(input, "-debug") == 0) {
        if (!debug) {
            debug = 1;
            printf("-debug enabled\n");
        } else {
            debug = 0;
            printf("-debug disabled\n");
        }
    } else if (strcmp(input, "-pitch") == 0) {
        scanf("%hhu", &number);
        SetPitch(number);
        printf("-pitch %hhu\n", number);
    } else if (strcmp(input, "-speed") == 0) {
        scanf("%hhu", &number);
        SetSpeed(number);
        printf("-speed %hhu\n", number);
    } else if (strcmp(input, "-mouth")) {
        scanf("%hhu", &number);
        SetMouth(number);
        printf("-mouth %hhu\n", number);
    } else if (strcmp(input, "-throat")) {
        scanf("%hhu", &number);
        SetThroat(number);
        printf("-throat %hhu\n", number);
    } else if (strcmp(input, "-sing")) {
        if (!singmode) {
            singmode = 1;
            printf("-sing enabled\n");
        } else {
            singmode = 0;
            printf("-sing disabled\n");
        }
    } else {
        printf("unknown command %s\n", input);
    }
}

int main() {
    stdin = &uart_stdin;
    stdout = &uart_stdout;

    init();
}
```

Продолжение листинга A.16

```
sei();

loop();

while (1);
}
```

Листинг A.17 — Содержимое файла src/translator.c

```
#include "translator.h"

#include <string.h>

#include "memmanagment.h"

#define RULE_SIZE 3

const char RU_TABLE[][RULE_SIZE] PROGMEM = {
    "AE", // ?
    "B",  // ?
    "V",  // ?
    "G",  // ?
    "D",  // ?
    "A",  // ?
    "ZH", // ?
    "Z",  // ?
    "EE", // ?
    "EI", // ?
    "K",  // ?
    "L",  // ?
    "M",  // ?
    "N",  // ?
    "O",  // ?
    "P",  // ?
    "R",  // ?
    "S",  // ?
    "T",  // ?
    "U",  // ?
    "F",  // ?
    "H",  // ?
    "C",  // ?
    "CH", // ?
    "SH", // ?
    "SH", // ?
}
```

Продолжение листинга А.17

```

    "",      // ?
    "E",     // ?
    "",      // ?
    "A",     // ?
    "U",     // ?
    "IYA"    // ?
};

int is_russian_upper(char ch) { return (ch >= '?' && ch <=
'?') || ch == '?'; }

int is_russian_lowest(char ch) { return (ch >= '?' && ch <=
'?') || ch == '?'; }

int is_russian(char ch) {
    return is_russian_lowest(ch) || is_russian_upper(ch);
}

const char* get_translation(char ch) {
    if (is_russian_upper(ch)) {
        if (ch == '?') ch = '?';
        return copy_pgm_str(RU_TABLE[ch - '?']);
    } else {
        if (ch == '?') ch = '?';
        return copy_pgm_str(RU_TABLE[ch - '?']);
    }
}

void translate(char* dst, char* src, int dstsize) {
    int srctime = strlen(src);
    int size = 0;
    for (int i = 0; i < srctime && size - RULE_SIZE <=
dstsize; ++i) {
        char ch = src[i];
        if (is_russian(ch)) {
            const char* translated = get_translation(ch);
            strncat(dst, translated, RULE_SIZE);
            size += strlen(translated);
        } else {
            strncat(dst, &ch, 1);
            size++;
        }
    }
}

```

Листинг А.18 — Содержимое файла src/debug_sam.c

```
#include "debug_sam.h"

int debug DATAMEM = 0;

void PrintPhonemes(unsigned char* phonemeindex, unsigned char*
phonemeLength,
                    unsigned char* stress) {
    int i = 0;
    printf("=====\n");

    printf("Internal Phoneme presentation:\n\n");
    printf(" idx      phoneme  length  stress\n");
    printf("-----\n");

    while ((phonemeindex[i] != 255) && (i < 255)) {
        if (phonemeindex[i] < 81) {
            printf(" %3i      %c%c      %3i      %i\n",
phonemeindex[i],
pgm_read_byte(&signInputTable1[phonemeindex[i]]),
pgm_read_byte(&signInputTable2[phonemeindex[i]]),
phonemeLength[i], stress[i]);
        } else {
            printf(" %3i      ??      %3i      %i\n",
phonemeindex[i],
phonemeLength[i], stress[i]);
        }
        i++;
    }
    printf("=====\n");
    printf("\n");
}

void PrintOutput(unsigned char* flag, unsigned char* f1,
unsigned char* f2,
                unsigned char* f3, unsigned char* a1,
unsigned char* a2,
                unsigned char* a3, unsigned char* p) {
    printf("=====\n");
    printf("Final data for speech output:\n\n");
    int i = 0;
    printf(" flags ampl1 freq1 ampl2 freq2 ampl3 freq3
pitch\n");
    printf("-----
\n");
}
```

Продолжение листинга A.18

```
        while (i < 255) {
            printf("%5i %5i %5i %5i %5i %5i %5i %5i\n", flag[i],
a1[i], f1[i],
                a2[i], f2[i], a3[i], f3[i], p[i]);
            i++;
        }
        printf("=====\n");
    }

extern unsigned char GetRuleByte(unsigned short mem62,
unsigned char Y);

void PrintRule(int offset) {
    int i = 1;
    unsigned char A = 0;
    printf("Applying rule: ");
    do {
        A = GetRuleByte(offset, i);
        if ((A & 127) == '=')
            printf(" -> ");
        else
            printf("%c", A & 127);
        i++;
    } while ((A & 128) == 0);
    printf("\n");
}
```

Листинг A.19 — Содержимое файла src/memmanagment.c

```
#include "memmanagment.h"

char COPY_PGM_BUFFER[256] NOINITMEM;

const char* copy_pgm_str(const char* pgm_str) {
    strcpy_P(COPY_PGM_BUFFER, pgm_str);
    return COPY_PGM_BUFFER;
}
```

Листинг A.20 — Содержимое файла src/player.c

```
#include "player.h"

#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdio.h>
#include <stdlib.h>
```

Продолжение листинга А.20

```
static volatile player_data_t data;
static volatile unsigned int pos;
static volatile callback_func_t callback;
static volatile int blocked;

void stop_pwm0() { OCR0 = 0; }

// прескейлер /1
void start_timer1() { TCCR1B |= (1 << CS10); }

void stop_timer1() { TCCR1B &= ~(1 << CS12) | (1 << CS11) |
(1 << CS10)); }

void init_pwm0() {
    DDRB |= (1 << PB4);

    // Fast PWM, non-inverting, prescaler = 1
    // Частота ШИМ  $\approx 8$  МГц / (1 * 256)  $\sim 31.25$  кГц
    TCCR0 = (1 << WGM00) | (1 << WGM01) | // Fast PWM
            (1 << COM01) | // non-inverting
            (1 << CS00); // предделитель = 1

    stop_pwm0();
}

void init_timer1() {
    // CTC по OCR1A
    TCCR1A = 0;
    TCCR1B = (1 << WGM12); // CTC mode
    OCR1A = 362; // 8 МГц и 22050 Гц прескейлер /1

    // Разрешаем прерывание по совпадению с OCR1A
    TIMSK |= (1 << OCIE1A);

    stop_timer1();
}

ISR(TIMER1_COMPA_vect) {
    if (pos == data.size) {
        stop_pwm0();
        stop_timer1();
        blocked = 0;
        if (callback != NULL) callback(&data);
        return;
    }
}
```

Продолжение листинга A.20

```
        OCR0 = data.buffer[pos++];
    }

    void init_player() {
        pos = 0;
        callback = NULL;
        init_pwm0();
        init_timer1();
    }

    player_data_t make_player_data(char* buffer, unsigned int
size) {
        player_data_t data;
        data.buffer = buffer;
        data.size = size;
        return data;
    }

    void play(player_data_t _data, callback_func_t _callback) {
        blocked = 1;
        data = _data;
        callback = _callback;
        pos = 0;
        start_timer1();
    }

    int is_player_blocked() { return blocked; }

    void wait_player() { while (is_player_blocked()); }
```

Листинг A.21 — Содержимое файла src/reciter.c

```
#include <stdio.h>
#include <string.h>

#include "reciter.h"

#include "debug_sam.h"

unsigned char A NOINITMEM, X NOINITMEM, Y NOINITMEM;

static unsigned char inputtemp[256] NOINITMEM;    // secure
copy of input tab36096

void Code37055(unsigned char mem59)
{
```

Продолжение листинга A.21

```
    X = mem59;
    X--;
    A = inputtemp[X];
    Y = A;
    A = pgm_read_byte(&tab36376[Y]);
    return;
}

void Code37066(unsigned char mem58)
{
    X = mem58;
    X++;
    A = inputtemp[X];
    Y = A;
    A = pgm_read_byte(&tab36376[Y]);
}

unsigned char GetRuleByte(unsigned short mem62, unsigned char
Y)
{
    unsigned int address = mem62;

    if (mem62 >= 37541)
    {
        address -= 37541;
        return pgm_read_byte(&rules2[address+Y]);
    }
    address -= 32000;
    return pgm_read_byte(&rules[address+Y]);
}

int TextToPhonemes(unsigned char *input) // Code36484
{
    //unsigned char *tab39445 = &mem[39445];    //input and
output
    //unsigned char mem29;
    unsigned char mem56;           //output position for phonemes
    unsigned char mem57;
    unsigned char mem58;
    unsigned char mem59;
    unsigned char mem60;
    unsigned char mem61;
    unsigned short mem62;          // memory position of current
rule
    unsigned char mem64;           // position of '=' or current
```


Продолжение листинга A.21

```
character
    unsigned char mem65;        // position of ')'
    unsigned char mem66;        // position of '('
    unsigned char mem36653;

    inputtemp[0] = 32;

    // secure copy of input
    // because input will be overwritten by phonemes
    X = 1;
    Y = 0;
    do
    {
        //pos36499:
        A = input[Y] & 127;
        if ( A >= 112) A = A & 95;
        else if ( A >= 96) A = A & 79;

        inputtemp[X] = A;
        X++;
        Y++;
    } while (Y != 255);

    X = 255;
    inputtemp[X] = 27;
    mem61 = 255;

pos36550:
    A = 255;
    mem56 = 255;

pos36554:
    while(1)
    {
        mem61++;
        X = mem61;
        A = inputtemp[X];
        mem64 = A;
        if (A == '[')
        {
            mem56++;
            X = mem56;
            A = 155;
```

Продолжение листинга А.21

```
        input[X] = 155;
        //goto pos36542;
        //          Code39771();      //Code39777();
        return 1;
    }

    //pos36579:
    if (A != '.') break;
    X++;
    Y = inputtemp[X];
    A = pgm_read_byte(&tab36376[Y]) & 1;
    if(A != 0) break;
    mem56++;
    X = mem56;
    A = '.';
    input[X] = '.';
} //while

//pos36607:
A = mem64;
Y = A;
A = pgm_read_byte(&tab36376[A]);
mem57 = A;
if((A&2) != 0)
{
    mem62 = 37541;
    goto pos36700;
}

//pos36630:
A = mem57;
if(A != 0) goto pos36677;
A = 32;
inputtemp[X] = ' ';
mem56++;
X = mem56;
if (X > 120) goto pos36654;
input[X] = A;
goto pos36554;

// -----

//36653 is unknown. Contains position
pos36654:
```

Продолжение листинга A.21

```
    input[X] = 155;
    A = mem61;
    mem36653 = A;
    // mem29 = A; // not used
    // Code36538(); das ist eigentlich
    return 1;
    //Code39771();
    //go on if there is more input ???
    mem61 = mem36653;
    goto pos36550;

pos36677:
    A = mem57 & 128;
    if(A == 0)
    {
        //36683: BRK
        return 0;
    }

    // go to the right rules for this character.
    X = mem64 - 'A';
    mem62 = pgm_read_byte(&tab37489[X]) |
    (pgm_read_byte(&tab37515[X])<<8);

    // -----
    // go to next rule
    // -----

pos36700:

    // find next rule
    Y = 0;
    do
    {
        mem62 += 1;
        A = GetRuleByte(mem62, Y);
    } while ((A & 128) == 0);
    Y++;

    //pos36720:
    // find '('
    while(1)
    {
        A = GetRuleByte(mem62, Y);
        if (A == '(') break;
        Y++;
    }
```

Продолжение листинга А.21

```
    }
    mem66 = Y;

    //pos36732:
    // find ')'
    do
    {
        Y++;
        A = GetRuleByte(mem62, Y);
    } while(A != ')');
    mem65 = Y;

    //pos36741:
    // find '='
    do
    {
        Y++;
        A = GetRuleByte(mem62, Y);
        A = A & 127;
    } while (A != '=');
    mem64 = Y;

    X = mem61;
    mem60 = X;

    // compare the string within the bracket
    Y = mem66;
    Y++;
    //pos36759:
    while(1)
    {
        mem57 = inputtemp[X];
        A = GetRuleByte(mem62, Y);
        if (A != mem57) goto pos36700;
        Y++;
        if(Y == mem65) break;
        X++;
        mem60 = X;
    }

    // the string in the bracket is correct

    //pos36787:
    A = mem61;
    mem59 = mem61;
```

Продолжение листинга A.21

```
pos36791:
    while(1)
    {
        mem66--;
        Y = mem66;
        A = GetRuleByte(mem62, Y);
        mem57 = A;
        //36800: BPL 36805
        if ((A & 128) != 0) goto pos37180;
        X = A & 127;
        A = pgm_read_byte(&tab36376[X]) & 128;
        if (A == 0) break;
        X = mem59-1;
        A = inputtemp[X];
        if (A != mem57) goto pos36700;
        mem59 = X;
    }

//pos36833:
    A = mem57;
    if (A == ' ') goto pos36895;
    if (A == '#') goto pos36910;
    if (A == '.') goto pos36920;
    if (A == '&') goto pos36935;
    if (A == '@') goto pos36967;
    if (A == '^') goto pos37004;
    if (A == '+') goto pos37019;
    if (A == ':') goto pos37040;
    // Code42041(); //Error
    //36894: BRK
    return 0;

    // -----

pos36895:
    Code37055(mem59);
    A = A & 128;
    if(A != 0) goto pos36700;
pos36905:
    mem59 = X;
    goto pos36791;

    // -----

pos36910:
    Code37055(mem59);
```

Продолжение листинга А.21

```
A = A & 64;
if(A != 0) goto pos36905;
goto pos36700;

// -----

pos36920:
Code37055(mem59);
A = A & 8;
if(A == 0) goto pos36700;
pos36930:
mem59 = X;
goto pos36791;

// -----

pos36935:
Code37055(mem59);
A = A & 16;
if(A != 0) goto pos36930;
A = inputtemp[X];
if (A != 72) goto pos36700;
X--;
A = inputtemp[X];
if ((A == 67) || (A == 83)) goto pos36930;
goto pos36700;

// -----

pos36967:
Code37055(mem59);
A = A & 4;
if(A != 0) goto pos36930;
A = inputtemp[X];
if (A != 72) goto pos36700;
if ((A != 84) && (A != 67) && (A != 83)) goto pos36700;
mem59 = X;
goto pos36791;

// -----

pos37004:
Code37055(mem59);
A = A & 32;
```

Продолжение листинга А.21

```
        if(A == 0) goto pos36700;

pos37014:
    mem59 = X;
    goto pos36791;

    // -----

pos37019:
    X = mem59;
    X--;
    A = inputtemp[X];
    if ((A == 'E') || (A == 'I') || (A == 'Y')) goto pos37014;
    goto pos36700;
    // -----

pos37040:
    Code37055(mem59);
    A = A & 32;
    if(A == 0) goto pos36791;
    mem59 = X;
    goto pos37040;

//-----

pos37077:
    X = mem58+1;
    A = inputtemp[X];
    if (A != 'E') goto pos37157;
    X++;
    Y = inputtemp[X];
    X--;
    A = pgm_read_byte(&tab36376[Y]) & 128;
    if(A == 0) goto pos37108;
    X++;
    A = inputtemp[X];
    if (A != 'R') goto pos37113;
pos37108:
    mem58 = X;
    goto pos37184;
pos37113:
    if ((A == 83) || (A == 68)) goto pos37108;    // 'S' 'D'
    if (A != 76) goto pos37135; // 'L'
    X++;
    A = inputtemp[X];
```

Продолжение листинга А.21

```
    if (A != 89) goto pos36700;
    goto pos37108;

pos37135:
    if (A != 70) goto pos36700;
    X++;
    A = inputtemp[X];
    if (A != 85) goto pos36700;
    X++;
    A = inputtemp[X];
    if (A == 76) goto pos37108;
    goto pos36700;

pos37157:
    if (A != 73) goto pos36700;
    X++;
    A = inputtemp[X];
    if (A != 78) goto pos36700;
    X++;
    A = inputtemp[X];
    if (A == 71) goto pos37108;
    //pos37177:
    goto pos36700;

    // -----

pos37180:

    A = mem60;
    mem58 = A;

pos37184:
    Y = mem65 + 1;

    //37187: CPY 64
    // if(? != 0) goto pos37194;
    if(Y == mem64) goto pos37455;
    mem65 = Y;
    //37196: LDA (62),Y
    A = GetRuleByte(mem62, Y);
    mem57 = A;
    X = A;
    A = pgm_read_byte(&tab36376[X]) & 128;
    if(A == 0) goto pos37226;
    X = mem58+1;
    A = inputtemp[X];
```


Продолжение листинга А.21

```
    if (A != mem57) goto pos36700;
    mem58 = X;
    goto pos37184;
pos37226:
    A = mem57;
    if (A == 32) goto pos37295;    // ' '
    if (A == 35) goto pos37310;    // '#'
    if (A == 46) goto pos37320;    // '.'
    if (A == 38) goto pos37335;    // '&'
    if (A == 64) goto pos37367;    // ''
    if (A == 94) goto pos37404;    // ''
    if (A == 43) goto pos37419;    // '+'
    if (A == 58) goto pos37440;    // ':'
    if (A == 37) goto pos37077;    // '%'
    //pos37291:
    // Code42041(); //Error
    //37294: BRK
    return 0;

    // -----
pos37295:
    Code37066(mem58);
    A = A & 128;
    if(A != 0) goto pos36700;
pos37305:
    mem58 = X;
    goto pos37184;

    // -----

pos37310:
    Code37066(mem58);
    A = A & 64;
    if(A != 0) goto pos37305;
    goto pos36700;

    // -----

pos37320:
    Code37066(mem58);
    A = A & 8;
    if(A == 0) goto pos36700;

pos37330:
    mem58 = X;
```

Продолжение листинга А.21

```
        goto pos37184;

        // -----

pos37335:
    Code37066(mem58);
    A = A & 16;
    if(A != 0) goto pos37330;
    A = inputtemp[X];
    if (A != 72) goto pos36700;
    X++;
    A = inputtemp[X];
    if ((A == 67) || (A == 83)) goto pos37330;
    goto pos36700;

    // -----

pos37367:
    Code37066(mem58);
    A = A & 4;
    if(A != 0) goto pos37330;
    A = inputtemp[X];
    if (A != 72) goto pos36700;
    if ((A != 84) && (A != 67) && (A != 83)) goto pos36700;
    mem58 = X;
    goto pos37184;

    // -----

pos37404:
    Code37066(mem58);
    A = A & 32;
    if(A == 0) goto pos36700;
pos37414:
    mem58 = X;
    goto pos37184;

    // -----

pos37419:
    X = mem58;
    X++;
    A = inputtemp[X];
    if ((A == 69) || (A == 73) || (A == 89)) goto pos37414;
    goto pos36700;
```

Продолжение листинга A.21

```
// -----  
  
pos37440:  
  
    Code37066(mem58);  
    A = A & 32;  
    if(A == 0) goto pos37184;  
    mem58 = X;  
    goto pos37440;  
pos37455:  
    Y = mem64;  
    mem61 = mem60;  
  
    if (debug)  
        PrintRule(mem62);  
  
pos37461:  
    //37461: LDA (62),Y  
    A = GetRuleByte(mem62, Y);  
    mem57 = A;  
    A = A & 127;  
    if (A != '=')  
    {  
        mem56++;  
        X = mem56;  
        input[X] = A;  
    }  
  
    //37478: BIT 57  
    //37480: BPL 37485 //not negative flag  
    if ((mem57 & 128) == 0) goto pos37485; //???  
    goto pos36554;  
pos37485:  
    Y++;  
    goto pos37461;  
}
```

Листинг A.22 — Содержимое файла src/ReciterTabs.c

```
#include "ReciterTabs.h"  
  
//some flags  
const unsigned char tab36376[] PROGMEM =  
{  
    0, 0, 0, 0, 0, 0, 0, 0, // 0-7
```

Продолжение листинга А.22

```

    0, 0, 0, 0, 0, 0, 0, 0, // 8-15
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 2, 2, 2, 2, 2, 2, 130, // ' ', '!'
    0, 0, 2, 2, 2, 2, 2, 2,
    3, 3, 3, 3, 3, 3, 3, 3,
    3, 3, 2, 2, 2, 2, 2, 2,
    2, 192, 168, 176, 172, 192, 160, 184, // '@', 'A'
    160, 192, 188, 160, 172, 168, 172, 192,
    160, 160, 172, 180, 164, 192, 168, 168,
    176, 192, 188, 0, 0, 0, 2, 0, // 'X', 'Y', 'Z', '[',
    32, 32, 155, 32, 192, 185, 32, 205,
    163, 76, 138, 142
};

const unsigned char rules[] PROGMEM =
{
    ']', 'A'|0x80,
    ' ', '(', 'A', '.', ')',
    '=', 'E', 'H', '4', 'Y', '.', ' '|0x80,
    '(', 'A', ')', ' ',
    ' ', '(', 'A', 'R', 'E', ')', ' ',
    ' ', '(', 'A', 'R', ')', 'O',
    '(', 'A', 'R', ')', '#',
    0x80,
    ' ', '^', '(', 'A', 'S', ')', '#',
    0x80,
    '(', 'A', ')', 'W', 'A',
    '(', 'A', 'W', ')',
    ' ', ':', '(', 'A', 'N', 'Y', ')',
    '=', 'E', 'H', '4', 'N', 'I', 'Y'|0x80,
    '(', 'A', ')', '^', '+', '#',
    '#', ':', '(', 'A', 'L', 'L', 'Y', ')',
    0x80,
    ' ', '(', 'A', 'L', ')', '#',
    '(', 'A', 'G', 'A', 'I', 'N', ')',
    '=', 'A', 'X', 'G', 'E', 'H', '4', 'N'|0x80,
    '#', ':', '(', 'A', 'G', ')', 'E',
    '(', 'A', ')', '^', '%',
    '(', 'A', ')', '^', '+', ':', '#',
    ' ', ':', '(', 'A', ')', '^', '+', ' ',
    ' ', '(', 'A', 'R', 'R', ')',
    '(', 'A', 'R', 'R', ')',
    0x80,
    ' ', '^', '(', 'A', 'R', ')', ' ',
    0x80,
    '=', 'A', 'H'|0x80,
    '=', 'A', 'A', 'R'|0x80,
    '=', 'A', 'X', 'R'|0x80,
    '=', 'E', 'H', '4', 'R'|
    0x80,
    '=', 'E', 'Y', '4', 'S'|
    0x80,
    '=', 'A', 'X'|0x80,
    '=', 'A', 'O', '5'|0x80,
    '=', 'E', 'Y', '5'|0x80,
    '=', 'U', 'L', 'I', 'Y'|
    0x80,
    '=', 'U', 'L'|0x80,
    '=', 'I', 'H', 'J'|0x80,
    '=', 'E', 'Y'|0x80,
    '=', 'A', 'E'|0x80,
    '=', 'E', 'Y', '4'|0x80,
    '=', 'A', 'X', 'R'|0x80,
    '=', 'A', 'E', '4', 'R'|
    0x80,
    '=', 'A', 'A', '5', 'R'|
    0x80,

```

Продолжение листинга A.22

```

'(', 'A', 'R', ')',
0x80,
'(', 'A', 'I', 'R', ')',
0x80,
'(', 'A', 'I', ')',
'(', 'A', 'Y', ')',
'(', 'A', 'U', ')',
'#', ':', '(', 'A', 'L', ')', ' ',
'#', ':', '(', 'A', 'L', 'S', ')', ' ',
'(', 'A', 'L', 'K', ')',
0x80,
'(', 'A', 'L', ')', '^',
' ', ':', '(', 'A', 'B', 'L', 'E', ')',
'=', 'E', 'Y', '4', 'B', 'U', 'L' | 0x80,
'(', 'A', 'B', 'L', 'E', ')',
'=', 'A', 'X', 'B', 'U', 'L' | 0x80,
'(', 'A', ')', 'V', 'O',
'(', 'A', 'N', 'G', ')', '+',
'=', 'E', 'Y', '4', 'N', 'J' | 0x80,
'(', 'A', 'T', 'A', 'R', 'I', ')',
'=', 'A', 'H', 'T', 'A', 'A', '4', 'R', 'I', 'Y' | 0x80,
'(', 'A', ')', 'T', 'O', 'M',
'(', 'A', ')', 'T', 'T', 'I',
' ', '(', 'A', 'T', ')', ' ',
' ', '(', 'A', ')', 'T',
'(', 'A', ')',

'=', 'A', 'A', '5', 'R' |
'=', 'E', 'H', '4', 'R' |
'=', 'E', 'Y', '4' | 0x80,
'=', 'E', 'Y', '5' | 0x80,
'=', 'A', 'O', '4' | 0x80,
'=', 'U', 'L' | 0x80,
'=', 'U', 'L', 'Z' | 0x80,
'=', 'A', 'O', '4', 'K' |
'=', 'A', 'O', 'L' | 0x80,
'=', 'E', 'Y', '4' | 0x80,
'=', 'A', 'E' | 0x80,
'=', 'A', 'E' | 0x80,
'=', 'A', 'E', 'T' | 0x80,
'=', 'A', 'H' | 0x80,
'=', 'A', 'E' | 0x80,

']', 'B' | 0x80,
' ', '(', 'B', ')', ' ',
0x80,
' ', '(', 'B', 'E', ')', '^', '#',
'(', 'B', 'E', 'I', 'N', 'G', ')',
'=', 'B', 'I', 'Y', '4', 'I', 'H', 'N', 'X' | 0x80,
' ', '(', 'B', 'O', 'T', 'H', ')', ' ',
'=', 'B', 'O', 'W', '4', 'T', 'H' | 0x80,
' ', '(', 'B', 'U', 'S', ')', '#',
'=', 'B', 'I', 'H', '4', 'Z' | 0x80,
'(', 'B', 'R', 'E', 'A', 'K', ')',
'=', 'B', 'R', 'E', 'Y', '5', 'K' | 0x80,
'(', 'B', 'U', 'I', 'L', ')',
'=', 'B', 'I', 'H', '4', 'L' | 0x80,
'(', 'B', ')',

'=', 'B' | 0x80,

']', 'C' | 0x80,
' ', '(', 'C', ')', ' ',
0x80,

'=', 'S', 'I', 'Y', '4' |

```

Продолжение листинга А.22

```

' ', '(', 'C', 'H', ')', '^',
'^', 'E', '(', 'C', 'H', ')',
'(', 'C', 'H', 'A', ')', 'R', '#',
0x80,
'(', 'C', 'H', ')',
' ', 'S', '(', 'C', 'I', ')', '#',
0x80,
'(', 'C', 'I', ')', 'A',
'(', 'C', 'I', ')', 'O',
'(', 'C', 'I', ')', 'E', 'N',
'(', 'C', 'I', 'T', 'Y', ')',
'=', 'S', 'I', 'H', 'T', 'I', 'Y' | 0x80,
'(', 'C', ')', '+',
'(', 'C', 'K', ')',
'(', 'C', 'O', 'M', 'M', 'O', 'D', 'O', 'R', 'E', ')', '=', 'K', 'A', 'A', '4', 'M', 'A',
0x80,
'(', 'C', 'O', 'M', ')',
0x80,
'(', 'C', 'U', 'I', 'T', ')',
0x80,
'(', 'C', 'R', 'E', 'A', ')',
'=', 'K', 'R', 'I', 'Y', 'E', 'Y' | 0x80,
'(', 'C', ')',
'=', 'K' | 0x80,

']', 'D' | 0x80,
' ', '(', 'D', ')', ' ',
0x80,
' ', '(', 'D', 'R', '.', ')', ' ',
'=', 'D', 'A', 'A', '4', 'K', 'T', 'E', 'R' | 0x80,
'#', ':', '(', 'D', 'E', 'D', ')', ' ',
'=', 'D', 'I', 'H', 'D' |
0x80,
'.', 'E', '(', 'D', ')', ' ',
'=', 'D' | 0x80,
'#', ':', '^', 'E', '(', 'D', ')', ' ',
'=', 'T' | 0x80,
' ', '(', 'D', 'E', ')', '^', '#',
'=', 'D', 'I', 'H' | 0x80,
' ', '(', 'D', 'O', ')', ' ',
'=', 'D', 'U', 'W' | 0x80,
' ', '(', 'D', 'O', 'E', 'S', ')',
'=', 'D', 'A', 'H', 'Z' |
0x80,
'(', 'D', 'O', 'N', 'E', ')', ' ',
'=', 'D', 'A', 'H', '5', 'N' | 0x80,
'(', 'D', 'O', 'I', 'N', 'G', ')',
'=', 'D', 'U', 'W', '4', 'I', 'H', 'N', 'X' | 0x80,
' ', '(', 'D', 'O', 'W', ')',
'=', 'D', 'A', 'W' | 0x80,
'#', '(', 'D', 'U', ')', 'A',
'=', 'J', 'U', 'W' | 0x80,
'#', '(', 'D', 'U', ')', '^', '#',
'=', 'J', 'A', 'X' | 0x80,
'(', 'D', ')',
'=', 'D' | 0x80,

```

Продолжение листинга А.22

```
' ]', 'E' | 0x80,
' ', '(', 'E', ')', ' ', ' ',
'= ', 'I', 'Y', 'I', 'Y', '4' | 0x80,
'#', ':', '(', 'E', ')', ' ', ' ', '= ' | 0x80,
'\', ':', '^', '(', 'E', ')', ' ', ' ', '= ' | 0x80,
' ', ':', '(', 'E', ')', ' ', ' ',
'#', '(', 'E', 'D', ')', ' ', ' ',
'#', ':', '(', 'E', ')', 'D', ' ', ' ', '= ' | 0x80,
'(', 'E', 'V', ')', 'E', 'R',
0x80,
'(', 'E', ')', '^', '%',
'(', 'E', 'R', 'I', ')', '#',
'= ', 'I', 'Y', '4', 'R', 'I', 'Y' | 0x80,
'(', 'E', 'R', 'I', ')',
'= ', 'E', 'H', '4', 'R', 'I', 'H' | 0x80,
'#', ':', '(', 'E', 'R', ')', '#',
'(', 'E', 'R', 'R', 'O', 'R', ')',
'= ', 'E', 'H', '4', 'R', 'O', 'H', 'R' | 0x80,
'(', 'E', 'R', 'A', 'S', 'E', ')',
'= ', 'I', 'H', 'R', 'E', 'Y', '5', 'S' | 0x80,
'(', 'E', 'R', ')', '#',
'(', 'E', 'R', ')',
' ', '(', 'E', 'V', 'E', 'N', ')',
'= ', 'I', 'Y', 'V', 'E', 'H', 'N' | 0x80,
'#', ':', '(', 'E', ')', 'W', '= ' | 0x80,
'@', '(', 'E', 'W', ')',
'(', 'E', 'W', ')',
'(', 'E', ')', 'O',
'#', ':', '&', '(', 'E', 'S', ')', ' ', ' ',
'#', ':', '(', 'E', ')', 'S', ' ', ' ', '= ' | 0x80,
'#', ':', '(', 'E', 'L', 'Y', ')', ' ', ' ',
'#', ':', '(', 'E', 'M', 'E', 'N', 'T', ')',
'= ', 'M', 'E', 'H', 'N', 'T' | 0x80,
'(', 'E', 'F', 'U', 'L', ')',
0x80,
'(', 'E', 'E', ')',
'(', 'E', 'A', 'R', 'N', ')',
0x80,
' ', '(', 'E', 'A', 'R', ')', '^',
'(', 'E', 'A', 'D', ')',
'#', ':', '(', 'E', 'A', ')', ' ', ' ',
0x80,
'(', 'E', 'A', ')', 'S', 'U',
'(', 'E', 'A', ')',
'(', 'E', 'I', 'G', 'H', ')',
'(', 'E', 'I', ')',
'= ', 'I', 'Y', '4' | 0x80,
'= ', 'D' | 0x80,
'= ', 'E', 'H', '4', 'V' |
'= ', 'I', 'Y', '4' | 0x80,
'= ', 'E', 'R' | 0x80,
'= ', 'E', 'H', 'R' | 0x80,
'= ', 'E', 'R' | 0x80,
'= ', 'U', 'W' | 0x80,
'= ', 'Y', 'U', 'W' | 0x80,
'= ', 'I', 'Y' | 0x80,
'= ', 'I', 'H', 'Z' | 0x80,
'= ', 'L', 'I', 'Y' | 0x80,
'= ', 'F', 'U', 'H', 'L' |
'= ', 'I', 'Y', '4' | 0x80,
'= ', 'E', 'R', '5', 'N' |
'= ', 'E', 'R', '5' | 0x80,
'= ', 'E', 'H', 'D' | 0x80,
'= ', 'I', 'Y', 'A', 'X' |
'= ', 'E', 'H', '5' | 0x80,
'= ', 'I', 'Y', '5' | 0x80,
'= ', 'E', 'Y', '4' | 0x80,
'= ', 'I', 'Y', '4' | 0x80,
```

Продолжение листинга A.22

```
' ', '(', 'E', 'Y', 'E', ')',
'(', 'E', 'Y', ')',
'(', 'E', 'U', ')',
0x80,
'(', 'E', 'Q', 'U', 'A', 'L', ')',
'=', 'I', 'Y', '4', 'K', 'W', 'U', 'L' | 0x80,
'(', 'E', ')',

'J', 'F' | 0x80,
' ', '(', 'F', ')', ' ', ' ',
0x80,
'(', 'F', 'U', 'L', ')',
0x80,
'(', 'F', 'R', 'I', 'E', 'N', 'D', ')',
'=', 'F', 'R', 'E', 'H', '5', 'N', 'D' | 0x80,
'(', 'F', 'A', 'T', 'H', 'E', 'R', ')',
'=', 'F', 'A', 'A', '4', 'D', 'H', 'E', 'R' | 0x80,
'(', 'F', ')', 'F', '=' | 0x80,
'(', 'F', ')',

'J', 'G' | 0x80,
' ', '(', 'G', ')', ' ', ' ',
0x80,
'(', 'G', 'I', 'V', ')',
'=', 'G', 'I', 'H', '5', 'V' | 0x80,
' ', '(', 'G', ')', 'I', '^',
'(', 'G', 'E', ')', 'T',
0x80,
'S', 'U', '(', 'G', 'G', 'E', 'S', ')',
'=', 'G', 'J', 'E', 'H', '4', 'S' | 0x80,
'(', 'G', 'G', ')',
' ', 'B', '#', '(', 'G', ')',
'(', 'G', ')', '+',
'(', 'G', 'R', 'E', 'A', 'T', ')',
'=', 'G', 'R', 'E', 'Y', '4', 'T' | 0x80,
'(', 'G', 'O', 'N', ')', 'E',
'=', 'G', 'A', 'O', '5', 'N' | 0x80,
'#', '(', 'G', 'H', ')', '=' | 0x80,
' ', '(', 'G', 'N', ')',
'(', 'G', ')',

'J', 'H' | 0x80,
' ', '(', 'H', ')', ' ', ' ',
'=', 'E', 'Y', '4', 'C', 'H' | 0x80,
' ', '(', 'H', 'A', 'V', ')',
'=', '/', 'H', 'A', 'E', '6', 'V' | 0x80,
```

```
'=', 'A', 'Y', '4' | 0x80,
'=', 'I', 'Y' | 0x80,
'=', 'Y', 'U', 'W', '5' |
'=', 'E', 'H' | 0x80,
'=', 'E', 'H', '4', 'F' |
'=', 'F', 'U', 'H', 'L' |
'=', 'F' | 0x80,
'=', 'J', 'I', 'Y', '4' |
'=', 'G' | 0x80,
'=', 'G', 'E', 'H', '5' |
'=', 'G' | 0x80,
'=', 'G' | 0x80,
'=', 'J' | 0x80,
'=', 'N' | 0x80,
'=', 'G' | 0x80,
```


Продолжение листинга А.22

```

' ', '(', 'H', 'E', 'R', 'E', ')',
'= ', '/', 'H', 'I', 'Y', 'R' | 0x80,
' ', '(', 'H', 'O', 'U', 'R', ')',
'= ', 'A', 'W', '5', 'E', 'R' | 0x80,
'(', 'H', 'O', 'W', ')',
0x80,
'(', 'H', ')', '#',
'(', 'H', ')', '=' | 0x80,

']', 'I' | 0x80,
' ', '(', 'I', 'N', ')',
' ', '(', 'I', ')', ' ',
'(', 'I', ')', ' ',
'(', 'I', 'N', ')', 'D',
0x80,
'S', 'E', 'M', '(', 'I', ')',
' ', 'A', 'N', 'T', '(', 'I', ')',
'(', 'I', 'E', 'R', ')',
0x80,
'#', ':', 'R', '(', 'I', 'E', 'D', ')', ' ',
'(', 'I', 'E', 'D', ')', ' ',
0x80,
'(', 'I', 'E', 'N', ')',
'= ', 'I', 'Y', 'E', 'H', 'N' | 0x80,
'(', 'I', 'E', ')', 'T',
'= ', 'A', 'Y', '4', 'E', 'H' | 0x80,
'(', 'I', '\', ')',
' ', ':', '(', 'I', ')', '^', '%',
' ', ':', '(', 'I', 'E', ')', ' ',
'(', 'I', ')', '%',
'(', 'I', 'E', ')',
' ', '(', 'I', 'D', 'E', 'A', ')',
'= ', 'A', 'Y', 'D', 'I', 'Y', '5', 'A', 'H' | 0x80,
'(', 'I', ')', '^', '+', ':', '#',
'(', 'I', 'R', ')', '#',
'(', 'I', 'Z', ')', '%',
'(', 'I', 'S', ')', '%',
'I', '^', '(', 'I', ')', '^', '#',
'+', '^', '(', 'I', ')', '^', '+',
'#', ':', '^', '(', 'I', ')', '^', '+',
'(', 'I', ')', '^', '+',
'(', 'I', 'R', ')',
'(', 'I', 'G', 'H', ')',
'(', 'I', 'L', 'D', ')',
'= ', 'A', 'Y', '5', 'L', 'D' | 0x80,
' ', '(', 'I', 'G', 'N', ')',

'= ', '/', 'H', 'A', 'W' |
'= ', '/', 'H' | 0x80,

'= ', 'I', 'H', 'N' | 0x80,
'= ', 'A', 'Y', '4' | 0x80,
'= ', 'A', 'Y' | 0x80,
'= ', 'A', 'Y', '5', 'N' |
'= ', 'I', 'Y' | 0x80,
'= ', 'A', 'Y' | 0x80,
'= ', 'I', 'Y', 'E', 'R' |
'= ', 'I', 'Y', 'D' | 0x80,
'= ', 'A', 'Y', '5', 'D' |
'= ', 'A', 'Y', '5' | 0x80,
'= ', 'A', 'Y', '5' | 0x80,
'= ', 'A', 'Y', '4' | 0x80,
'= ', 'I', 'Y' | 0x80,
'= ', 'I', 'Y', '4' | 0x80,
'= ', 'I', 'H' | 0x80,
'= ', 'A', 'Y', 'R' | 0x80,
'= ', 'A', 'Y', 'Z' | 0x80,
'= ', 'A', 'Y', 'Z' | 0x80,
'= ', 'I', 'H' | 0x80,
'= ', 'A', 'Y' | 0x80,
'= ', 'I', 'H' | 0x80,
'= ', 'A', 'Y' | 0x80,
'= ', 'E', 'R' | 0x80,
'= ', 'A', 'Y', '4' | 0x80,

'= ', 'I', 'H', 'G', 'N' |

```

Продолжение листинга A.22

```

0x80,
'(', 'I', 'G', 'N', ')', ' ', ' ',
'=', 'A', 'Y', '4', 'N' |
0x80,
'(', 'I', 'G', 'N', ')', '^',
'=', 'A', 'Y', '4', 'N' |
0x80,
'(', 'I', 'G', 'N', ')', '%',
'=', 'A', 'Y', '4', 'N' |
0x80,
'(', 'I', 'C', 'R', 'O', ')',
'=', 'A', 'Y', '4', 'K', 'R', 'O', 'H' | 0x80,
'(', 'I', 'Q', 'U', 'E', ')',
'=', 'I', 'Y', '4', 'K' |
0x80,
'(', 'I', ')',
'=', 'I', 'H' | 0x80,

']', 'J' | 0x80,
' ', '(', 'J', ')', ' ', ' ',
'=', 'J', 'E', 'Y', '4' |
0x80,
'(', 'J', ')',
'=', 'J' | 0x80,

']', 'K' | 0x80,
' ', '(', 'K', ')', ' ', ' ',
'=', 'K', 'E', 'Y', '4' |
0x80,
' ', '(', 'K', ')', 'N', '=' | 0x80,
'(', 'K', ')',
'=', 'K' | 0x80,

']', 'L' | 0x80,
' ', '(', 'L', ')', ' ', ' ',
'=', 'E', 'H', '4', 'L' |
0x80,
'(', 'L', 'O', ')', 'C', '#',
'=', 'L', 'O', 'W' | 0x80,
'L', '(', 'L', ')', '=', | 0x80,
'#', ':', '^', '(', 'L', ')', '%',
'=', 'U', 'L' | 0x80,
'(', 'L', 'E', 'A', 'D', ')',
'=', 'L', 'I', 'Y', 'D' |
0x80,
' ', '(', 'L', 'A', 'U', 'G', 'H', ')',
'=', 'L', 'A', 'E', '4', 'F' | 0x80,
'(', 'L', ')',
'=', 'L' | 0x80,

']', 'M' | 0x80,
' ', '(', 'M', ')', ' ', ' ',
'=', 'E', 'H', '4', 'M' |
0x80,
' ', '(', 'M', 'R', '.', ')', ' ', ' ',
'=', 'M', 'I', 'H', '4', 'S', 'T', 'E', 'R' | 0x80,
' ', '(', 'M', 'S', '.', ')',
'=', 'M', 'I', 'H', '5', 'Z' | 0x80,
' ', '(', 'M', 'R', 'S', '.', ')', ' ', ' ',
'=', 'M', 'I', 'H', '4', 'S', 'I', 'X', 'Z' | 0x80,
'(', 'M', 'O', 'V', ')',

```

Продолжение листинга А.22

```
'=', 'M', 'U', 'W', '4', 'V' | 0x80,
'(', 'M', 'A', 'C', 'H', 'I', 'N', ')',
'=', 'M', 'A', 'H', 'S', 'H', 'I', 'Y', '5', 'N' | 0x80,
'M', '(', 'M', ')', '=' | 0x80,
'(', 'M', ')', '=', 'M' | 0x80,

']', 'N' | 0x80,
' ', '(', 'N', ')', ' ',
0x80,
'E', '(', 'N', 'G', ')', '+',
'(', 'N', 'G', ')', 'R',
'(', 'N', 'G', ')', '#',
'(', 'N', 'G', 'L', ')', '%',
'=', 'N', 'X', 'G', 'U', 'L' | 0x80,
'(', 'N', 'G', ')',
'(', 'N', 'K', ')',
' ', '(', 'N', 'O', 'W', ')', ' ',
0x80,
'N', '(', 'N', ')', '=' | 0x80,
'(', 'N', 'O', 'N', ')', 'E',
'=', 'N', 'A', 'H', '4', 'N' | 0x80,
'(', 'N', ')', '=', 'N' | 0x80,

']', 'O' | 0x80,
' ', '(', 'O', ')', ' ',
0x80,
'(', 'O', 'F', ')', ' ',
' ', '(', 'O', 'H', ')', ' ',
'(', 'O', 'R', 'O', 'U', 'G', 'H', ')',
'=', 'E', 'R', '4', 'O', 'W' | 0x80,
'#', ':', '(', 'O', 'R', ')', ' ',
'#', ':', '(', 'O', 'R', 'S', ')', ' ',
'(', 'O', 'R', ')',
' ', '(', 'O', 'N', 'E', ')',
0x80,
'#', '(', 'O', 'N', 'E', ')', ' ',
0x80,
'(', 'O', 'W', ')',
' ', '(', 'O', 'V', 'E', 'R', ')',
'=', 'O', 'W', '5', 'V', 'E', 'R' | 0x80,
'P', 'R', '(', 'O', ')', 'V',
'(', 'O', 'V', ')',
0x80,
'(', 'O', ')', '^', '%',
'(', 'O', ')', '^', 'E', 'N',
'(', 'O', ')', '^', 'I', '#',
'=', 'E', 'H', '4', 'N' |
'=', 'N', 'J' | 0x80,
'=', 'N', 'X', 'G' | 0x80,
'=', 'N', 'X', 'G' | 0x80,
'=', 'N', 'X' | 0x80,
'=', 'N', 'X', 'K' | 0x80,
'=', 'N', 'A', 'W', '4' |
'=', 'N' | 0x80,
'=', 'O', 'H', '4', 'W' |
'=', 'A', 'H', 'V' | 0x80,
'=', 'O', 'W', '5' | 0x80,
'=', 'E', 'R' | 0x80,
'=', 'E', 'R', 'Z' | 0x80,
'=', 'A', 'O', 'R' | 0x80,
'=', 'W', 'A', 'H', 'N' |
'=', 'W', 'A', 'H', 'N' |
'=', 'O', 'W' | 0x80,
'=', 'U', 'W', '4' | 0x80,
'=', 'A', 'H', '4', 'V' |
'=', 'O', 'W', '5' | 0x80,
'=', 'O', 'W' | 0x80,
'=', 'O', 'W', '5' | 0x80,
```

Продолжение листинга A.22

'(', 'O', 'L', ')', 'D',	'=', 'O', 'W', '4', 'L'
0x80,	
'(', 'O', 'U', 'G', 'H', 'T', ')',	'=', 'A', 'O', '5', 'T'
0x80,	
'(', 'O', 'U', 'G', 'H', ')',	'=', 'A', 'H', '5', 'F'
0x80,	
' ', '(', 'O', 'U', ')',	'=', 'A', 'W' 0x80,
'H', '(', 'O', 'U', ')', 'S', '#',	'=', 'A', 'W', '4' 0x80,
'(', 'O', 'U', 'S', ')',	'=', 'A', 'X', 'S' 0x80,
'(', 'O', 'U', 'R', ')',	'=', 'O', 'H', 'R' 0x80,
'(', 'O', 'U', 'L', 'D', ')',	'=', 'U', 'H', '5', 'D'
0x80,	
'(', 'O', 'U', ')', '^', 'L',	'=', 'A', 'H', '5' 0x80,
'(', 'O', 'U', 'P', ')',	'=', 'U', 'W', '5', 'P'
0x80,	
'(', 'O', 'U', ')',	'=', 'A', 'W' 0x80,
'(', 'O', 'Y', ')',	'=', 'O', 'Y' 0x80,
'(', 'O', 'I', 'N', 'G', ')',	
'=', 'O', 'W', '4', 'I', 'H', 'N', 'X' 0x80,	
'(', 'O', 'I', ')',	'=', 'O', 'Y', '5' 0x80,
'(', 'O', 'O', 'R', ')',	'=', 'O', 'H', '5', 'R'
0x80,	
'(', 'O', 'O', 'K', ')',	'=', 'U', 'H', '5', 'K'
0x80,	
'F', '(', 'O', 'O', 'D', ')',	'=', 'U', 'W', '5', 'D'
0x80,	
'L', '(', 'O', 'O', 'D', ')',	'=', 'A', 'H', '5', 'D'
0x80,	
'M', '(', 'O', 'O', 'D', ')',	'=', 'U', 'W', '5', 'D'
0x80,	
'(', 'O', 'O', 'D', ')',	'=', 'U', 'H', '5', 'D'
0x80,	
'F', '(', 'O', 'O', 'T', ')',	'=', 'U', 'H', '5', 'T'
0x80,	
'(', 'O', 'O', ')',	'=', 'U', 'W', '5' 0x80,
'(', 'O', '\\', ')',	'=', 'O', 'H' 0x80,
'(', 'O', ')', 'E',	'=', 'O', 'W' 0x80,
'(', 'O', ')', ' ',	'=', 'O', 'W' 0x80,
'(', 'O', 'A', ')',	'=', 'O', 'W', '4' 0x80,
' ', '(', 'O', 'N', 'L', 'Y', ')',	
'=', 'O', 'W', '4', 'N', 'L', 'I', 'Y' 0x80,	
' ', '(', 'O', 'N', 'C', 'E', ')',	
'=', 'W', 'A', 'H', '4', 'N', 'S' 0x80,	
'(', 'O', 'N', '\\', 'T', ')',	
'=', 'O', 'W', '4', 'N', 'T' 0x80,	
'C', '(', 'O', ')', 'N',	'=', 'A', 'A' 0x80,

Продолжение листинга А.22

```
'(','O',')','N','G',
' ',' ':'^','(','O',')','N',
'I','(','O','N',')',
'#',' ':'(','O','N',')',
'#','^','(','O','N',')',
'(','O',')','S','T',
'(','O','F',')','^',
0x80,
'(','O','T','H','E','R',')',
'=','A','H','5','D','H','E','R'|0x80,
'R','(','O',')','B',
'^','R','(','O',')',' ':'#',
'(','O','S','S',')',' ',
0x80,
'#',' ':'^','(','O','M',')',
'(','O',')',

']','P'|0x80,
' ','(','P',')',' ',
0x80,
'(','P','H',')',
'(','P','E','O','P','L',')',
'=','P','I','Y','5','P','U','L'|0x80,
'(','P','O','W',')',
0x80,
'(','P','U','T',')',' ',
0x80,
'(','P',')','P','='|0x80,
'(','P',')','S','='|0x80,
'(','P',')','N','='|0x80,
'(','P','R','O','F','.',')',
'=','P','R','O','H','F','E','H','4','S','E','R'|0x80,
'(','P',')',

']','Q'|0x80,
' ','(','Q',')',' ',
'=','K','Y','U','W','4'|0x80,
'(','Q','U','A','R',')',
'=','K','W','O','H','5','R'|0x80,
'(','Q','U',')',
'(','Q',')',
']','R'|0x80,
' ','(','R',')',' ',
0x80,
' ','(','R','E',')','^','#',
'(','R',')','R','='|0x80,

'=','A','O'|0x80,
'=','A','H'|0x80,
'=','U','N'|0x80,
'=','U','N'|0x80,
'=','U','N'|0x80,
'=','O','W'|0x80,
'=','A','O','4','F'|
'=','R','A','A'|0x80,
'=','O','W','5'|0x80,
'=','A','O','5','S'|
'=','A','H','M'|0x80,
'=','A','A'|0x80,

'=','P','I','Y','4'|
'=','F'|0x80,
'=','P','A','W','4'|
'=','P','U','H','T'|
'=','P'|0x80,

'=','K','W'|0x80,
'=','K'|0x80,
'=','A','A','5','R'|
'=','R','I','Y'|0x80,
```

Продолжение листинга А.22

'(','R',')',	'=', 'R' 0x80,
']', 'S' 0x80,	
' ', '(','S',')', ' ',	'=', 'E', 'H', '4', 'S'
0x80,	
'(','S', 'H',')',	'=', 'S', 'H' 0x80,
'#', '(','S', 'I', 'O', 'N',')',	'=', 'Z', 'H', 'U', 'N'
0x80,	
'(','S', 'O', 'M', 'E',')',	'=', 'S', 'A', 'H', 'M'
0x80,	
'#', '(','S', 'U', 'R',')', '#',	'=', 'Z', 'H', 'E', 'R'
0x80,	
'(','S', 'U', 'R',')', '#',	'=', 'S', 'H', 'E', 'R'
0x80,	
'#', '(','S', 'U',')', '#',	'=', 'Z', 'H', 'U', 'W'
0x80,	
'#', '(','S', 'S', 'U',')', '#',	'=', 'S', 'H', 'U', 'W'
0x80,	
'#', '(','S', 'E', 'D',')',	'=', 'Z', 'D' 0x80,
'#', '(','S',')', '#',	'=', 'Z' 0x80,
'(','S', 'A', 'I', 'D',')',	'=', 'S', 'E', 'H', 'D'
0x80,	
'^', '(','S', 'I', 'O', 'N',')',	'=', 'S', 'H', 'U', 'N'
0x80,	
'(','S',')', 'S', '=' 0x80,	
'.', '(','S',')', ' ',	'=', 'Z' 0x80,
'#', ':', '.', 'E', '(','S',')', ' ',	'=', 'Z' 0x80,
'#', ':', '^', '#', '(','S',')', ' ',	'=', 'S' 0x80,
'U', '(','S',')', ' ',	'=', 'S' 0x80,
' ', ':', '#', '(','S',')', ' ',	'=', 'Z' 0x80,
'#', '#', '(','S',')', ' ',	'=', 'Z' 0x80,
' ', '(','S', 'C', 'H',')',	'=', 'S', 'K' 0x80,
'(','S',')', 'C', '+', '=' 0x80,	
'#', '(','S', 'M',')',	'=', 'Z', 'U', 'M' 0x80,
'#', '(','S', 'N',')', '\',	'=', 'Z', 'U', 'M' 0x80,
'(','S', 'T', 'L', 'E',')',	'=', 'S', 'U', 'L' 0x80,
'(','S',')',	'=', 'S' 0x80,
']', 'T' 0x80,	
' ', '(','T',')', ' ',	'=', 'T', 'I', 'Y', '4'
0x80,	
' ', '(','T', 'H', 'E',')', ' ', '#',	'=', 'D', 'H', 'I', 'Y'
0x80,	
' ', '(','T', 'H', 'E',')', ' ',	'=', 'D', 'H', 'A', 'X'
0x80,	
'(','T', 'O',')', ' ',	'=', 'T', 'U', 'X' 0x80,

Продолжение листинга A.22

```
' ', '(', 'T', 'H', 'A', 'T', ')',
'= ', 'D', 'H', 'A', 'E', 'T' | 0x80,
' ', '(', 'T', 'H', 'I', 'S', ')', ' ',
'= ', 'D', 'H', 'I', 'H', 'S' | 0x80,
' ', '(', 'T', 'H', 'E', 'Y', ')',
0x80,
' ', '(', 'T', 'H', 'E', 'R', 'E', ')',
'= ', 'D', 'H', 'E', 'H', 'R' | 0x80,
'(', 'T', 'H', 'E', 'R', ')',
0x80,
'(', 'T', 'H', 'E', 'I', 'R', ')',
'= ', 'D', 'H', 'E', 'H', 'R' | 0x80,
' ', '(', 'T', 'H', 'A', 'N', ')', ' ',
'= ', 'D', 'H', 'A', 'E', 'N' | 0x80,
' ', '(', 'T', 'H', 'E', 'M', ')', ' ',
'= ', 'D', 'H', 'A', 'E', 'N' | 0x80,
'(', 'T', 'H', 'E', 'S', 'E', ')', ' ',
'= ', 'D', 'H', 'I', 'Y', 'Z' | 0x80,
' ', '(', 'T', 'H', 'E', 'N', ')',
'= ', 'D', 'H', 'E', 'H', 'N' | 0x80,
'(', 'T', 'H', 'R', 'O', 'U', 'G', 'H', ')',
'= ', 'T', 'H', 'R', 'U', 'W', '4' | 0x80,
'(', 'T', 'H', 'O', 'S', 'E', ')',
'= ', 'D', 'H', 'O', 'H', 'Z' | 0x80,
'(', 'T', 'H', 'O', 'U', 'G', 'H', ')', ' ',
0x80,
'(', 'T', 'O', 'D', 'A', 'Y', ')',
'= ', 'T', 'U', 'X', 'D', 'E', 'Y' | 0x80,
'(', 'T', 'O', 'M', 'O', ')', 'R', 'R', 'O', 'W', '= ', 'T', 'U', 'M', 'A', 'A', '5' |
0x80,
'(', 'T', 'O', ')', 'T', 'A', 'L',
0x80,
' ', '(', 'T', 'H', 'U', 'S', ')',
'= ', 'D', 'H', 'A', 'H', '4', 'S' | 0x80,
'(', 'T', 'H', ')',
'#', ': ', '(', 'T', 'E', 'D', ')',
0x80,
'S', '(', 'T', 'I', ')', '#', 'N',
'(', 'T', 'I', ')', 'O',
'(', 'T', 'I', ')', 'A',
'(', 'T', 'I', 'E', 'N', ')',
0x80,
'(', 'T', 'U', 'R', ')', '#',
0x80,
'(', 'T', 'U', ')', 'A',
0x80,
'= ', 'D', 'H', 'E', 'Y' |
'= ', 'D', 'H', 'E', 'R' |
'= ', 'D', 'H', 'O', 'W' |
'= ', 'T', 'O', 'W', '5' |
'= ', 'T', 'H' | 0x80,
'= ', 'T', 'I', 'X', 'D' |
'= ', 'C', 'H' | 0x80,
'= ', 'S', 'H' | 0x80,
'= ', 'S', 'H' | 0x80,
'= ', 'S', 'H', 'U', 'N' |
'= ', 'C', 'H', 'E', 'R' |
'= ', 'C', 'H', 'U', 'W' |
```

Продолжение листинга А.22

```
' ', '(', 'T', 'W', 'O', ')',
'&', '(', 'T', ')', 'E', 'N', ' ', '=', |0x80,
'(', 'T', ')',
'=', 'T', |0x80,

']', 'U', |0x80,
' ', '(', 'U', ')', ' ', ' ',
0x80,
' ', '(', 'U', 'N', ')', 'I',
0x80,
' ', '(', 'U', 'N', ')',
' ', '(', 'U', 'P', 'O', 'N', ')',
'=', 'A', 'X', 'P', 'A', 'O', 'N', |0x80,
'@', '(', 'U', 'R', ')', '#',
0x80,
'(', 'U', 'R', ')', '#',
'=', 'Y', 'U', 'H', '4', 'R', |0x80,
'(', 'U', 'R', ')',
'(', 'U', ')', '^', ' ',
'(', 'U', ')', '^', '^',
'(', 'U', 'Y', ')',
' ', 'G', '(', 'U', ')', '#', '=' |0x80,
'G', '(', 'U', ')', '%', '=' |0x80,
'G', '(', 'U', ')', '#',
'#', 'N', '(', 'U', ')',
'@', '(', 'U', ')',
'(', 'U', ')',
'=', 'E', 'R', |0x80,
'=', 'A', 'H', |0x80,
'=', 'A', 'H', '5', |0x80,
'=', 'A', 'Y', '5', |0x80,

']', 'V', |0x80,
' ', '(', 'V', ')', ' ', ' ',
0x80,
'(', 'V', 'I', 'E', 'W', ')',
'=', 'V', 'Y', 'U', 'W', '5', |0x80,
'(', 'V', ')',
'=', 'W', |0x80,
'=', 'Y', 'U', 'W', |0x80,
'=', 'U', 'W', |0x80,
'=', 'Y', 'U', 'W', |0x80,

']', 'V', |0x80,
' ', '(', 'V', ')', 'I', 'Y', '4', |
0x80,
'(', 'V', 'I', 'E', 'W', ')',
'=', 'V', 'Y', 'U', 'W', '5', |0x80,
'(', 'V', ')',
'=', 'V', |0x80,

']', 'W', |0x80,
' ', '(', 'W', ')', ' ', ' ',
'=', 'D', 'A', 'H', '4', 'B', 'U', 'L', 'Y', 'U', 'W', |0x80,
' ', '(', 'W', 'E', 'R', 'E', ')',
'(', 'W', 'A', ')', 'S', 'H',
'(', 'W', 'A', ')', 'S', 'T',
'(', 'W', 'A', ')', 'S',
'(', 'W', 'A', ')', 'T',
'(', 'W', 'H', 'E', 'R', 'E', ')',
'=', 'W', 'H', 'E', 'H', 'R', |0x80,
'(', 'W', 'H', 'A', 'T', ')',
'=', 'W', 'H', 'A', 'H', 'T', |0x80,
'=', 'W', 'E', 'R', |0x80,
'=', 'W', 'A', 'A', |0x80,
'=', 'W', 'E', 'Y', |0x80,
'=', 'W', 'A', 'H', |0x80,
'=', 'W', 'A', 'A', |0x80,
```


Продолжение листинга А.22

```
'(','W','H','O','L',')',
'=', '/' , 'H','O','W','L'|0x80,
'(','W','H','O',')',
0x80,
'(','W','H',')',
'(','W','A','R',')', '#',
0x80,
'(','W','A','R',')',
0x80,
'(','W','O','R',')', '^',
'(','W','R',')',
'(','W','O','M',')', 'A',
0x80,
'(','W','O','M',')', 'E',
0x80,
'(','W','E','A',')', 'R',
'(','W','A','N','T',')',
'=', 'W','A','A','5','N','T'|0x80,
'A','N','S','(','W','E','R',')',
'(','W',')',

']','X'|0x80,
' ','(','X',')', ' ',
'=', 'E','H','4','K','R'|0x80,
' ','(','X',')',
'(','X',')',

']','Y'|0x80,
' ','(','Y',')', ' ',
0x80,
'(','Y','O','U','N','G',')',
'=', 'Y','A','H','N','X'|0x80,
' ','(','Y','O','U','R',')',
0x80,
' ','(','Y','O','U',')',
' ','(','Y','E','S',')',
0x80,
' ','(','Y',')',
'F','(','Y',')',
'P','S','(','Y','C','H',')',
'#', ':', '^', '(', 'Y', ')',
'#', ':', '^', '(', 'Y', ')', 'I',
' ',':','(','Y',')', ' ',
' ',':','(','Y',')', '#',
' ',':','(','Y',')', '^', '+', ':', '#',
' ',':','(','Y',')', '^', '#',

'=', '/' , 'H','U','W'|
'=', 'W','H'|0x80,
'=', 'W','E','H','R'|
'=', 'W','A','O','R'|
'=', 'W','E','R'|0x80,
'=', 'R'|0x80,
'=', 'W','U','H','M'|
'=', 'W','I','H','M'|
'=', 'W','E','H'|0x80,
'=', 'E','R'|0x80,
'=', 'W'|0x80,

'=', 'Z'|0x80,
'=', 'K','S'|0x80,

'=', 'W','A','Y','4'|
'=', 'Y','O','H','R'|
'=', 'Y','U','W'|0x80,
'=', 'Y','E','H','S'|
'=', 'Y'|0x80,
'=', 'A','Y'|0x80,
'=', 'A','Y','K'|0x80,
'=', 'I','Y'|0x80,
'=', 'I','Y'|0x80,
'=', 'A','Y'|0x80,
'=', 'A','Y'|0x80,
'=', 'I','H'|0x80,
'=', 'A','Y'|0x80,
```

```

'(', 'Y', ')',                                     '=', 'I', 'H' | 0x80,

']', 'Z' | 0x80,

' ', '(', 'Z', ')', ' ', ' ',                       '=', 'Z', 'I', 'Y', '4' |
0x80,
'(', 'Z', ')',                                     '=', 'Z' | 0x80,
'j' | 0x80
};

const unsigned char rules2[] PROGMEM =
{
'(', 'A', ')',                                     '=', | 0x80,
'(', '!', ')',                                     '=', ' .' | 0x80,
'(', '"', ')', ' ', ' ',
'=', '-', 'A', 'H', '5', 'N', 'K', 'W', 'O', 'W', 'T', '-' | 0x80,
'(', '"', ')',
'=', 'K', 'W', 'O', 'W', '4', 'T', '-' | 0x80,
'(', '#', ')',                                     '=', ' ',
', 'N', 'A', 'H', '4', 'M', 'B', 'E', 'R' | 0x80,
'(', '$', ')',                                     '=', ' ',
', 'D', 'A', 'A', '4', 'L', 'E', 'R' | 0x80,
'(', '%', ')',                                     '=', ' ',
', 'P', 'E', 'R', 'S', 'E', 'H', '4', 'N', 'T' | 0x80,
'(', '&', ')',                                     '=', ' ',
', 'A', 'E', 'N', 'D' | 0x80,
'(', '\', ')',                                     '=', | 0x80,
'(', '*', ')',                                     '=', ' ',
', 'A', 'E', '4', 'S', 'T', 'E', 'R', 'I', 'H', 'S', 'K' | 0x80,
'(', '+', ')',                                     '=', ' ',
', 'P', 'L', 'A', 'H', '4', 'S' | 0x80,
'(', ' ', ')',                                     '=', ' ', | 0x80,
' ', '(', '-', ')', ' ', ' ',                       '=', '-' | 0x80,
'(', '-', ')',                                     '=', | 0x80,
'(', '.', ')',                                     '=', ' ',
', 'P', 'O', 'Y', 'N', 'T' | 0x80,
'(', '/', ')',                                     '=', ' ',
', 'S', 'L', 'A', 'E', '4', 'S', 'H' | 0x80,
'(', '0', ')',                                     '=', ' ',
', 'Z', 'I', 'Y', '4', 'R', 'O', 'W' | 0x80,
' ', '(', '1', 'S', 'T', ')',
'=', 'F', 'E', 'R', '4', 'S', 'T' | 0x80,
' ', '(', '1', '0', 'T', 'H', ')',
'=', 'T', 'E', 'H', '4', 'N', 'T', 'H' | 0x80,
'(', '1', ')',                                     '=', ' ',
', 'W', 'A', 'H', '4', 'N' | 0x80,
' ', '(', '2', 'N', 'D', ')',

```

Продолжение листинга A.22

```

'=', 'S', 'E', 'H', '4', 'K', 'U', 'N', 'D' | 0x80,
'(', '2', ')', '=', '
', 'T', 'U', 'W', '4' | 0x80,
' ', '(', '3', 'R', 'D', ')',
'=', 'T', 'H', 'E', 'R', '4', 'D' | 0x80,
'(', '3', ')', '=', '
', 'T', 'H', 'R', 'I', 'Y', '4' | 0x80,
'(', '4', ')', '=', '
', 'F', 'O', 'H', '4', 'R' | 0x80,
' ', '(', '5', 'T', 'H', ')',
'=', 'F', 'I', 'H', '4', 'F', 'T', 'H' | 0x80,
'(', '5', ')', '=', '
', 'F', 'A', 'Y', '4', 'V' | 0x80,
' ', '(', '6', '4', ')', ' ',
'=', 'S', 'I', 'H', '4', 'K', 'S', 'T', 'I', 'Y', ' ', 'F', 'O', 'H', 'R' |
0x80,
'(', '6', ')', '=', '
', 'S', 'I', 'H', '4', 'K', 'S' | 0x80,
'(', '7', ')', '=', '
', 'S', 'E', 'H', '4', 'V', 'U', 'N' | 0x80,
' ', '(', '8', 'T', 'H', ')',
'=', 'E', 'Y', '4', 'T', 'H' | 0x80,
'(', '8', ')', '=', '
', 'E', 'Y', '4', 'T' | 0x80,
'(', '9', ')', '=', '
', 'N', 'A', 'Y', '4', 'N' | 0x80,
'(', ':', ')', '=', ' ', ' | 0x80,
'(', ';', ')', '=', ' ', ' | 0x80,
'(', '<', ')', '=', '
', 'L', 'E', 'H', '4', 'S', ' ', 'D', 'H', 'A', 'E', 'N' | 0x80,
'(', '=', ')', '=', '
', 'I', 'Y', '4', 'K', 'W', 'U', 'L', 'Z' | 0x80,
'(', '>', ')', '=', '
', 'G', 'R', 'E', 'Y', '4', 'T', 'E', 'R', ' ', 'D', 'H', 'A', 'E', 'N' |
0x80,
'(', '?', ')', '=', ' ', '? ' | 0x80,
'(', '@', ')', '=', '
', 'A', 'E', '6', 'T' | 0x80,
'(', '^', ')', '=', '
', 'K', 'A', 'E', '4', 'R', 'I', 'X', 'T' | 0x80,
']', 'A' | 0x80
};

//26 items. From 'A' to 'Z'
// positions for mem62 and mem63 for each character

```

Продолжение листинга A.22

```
const unsigned char tab37489[] PROGMEM =
{
0, 149, 247, 162, 57, 197, 6, 126,
199, 38, 55, 78, 145, 241, 85, 161,
254, 36, 69, 45, 167, 54, 83, 46,
71, 218
};

const unsigned char tab37515[] PROGMEM =
{
125, 126, 126, 127, 128, 129, 130, 130,
130, 132, 132, 132, 132, 132, 133, 135,
135, 136, 136, 137, 138, 139, 139, 140,
140, 140
};
```

Листинг A.23 — Содержимое файла src/render.c

```
#include "render.h"

unsigned char wait1 DATAMEM = 7;
unsigned char wait2 DATAMEM = 6;

extern unsigned char A, X, Y;
extern unsigned char mem44;
extern unsigned char mem47;
extern unsigned char mem49;
extern unsigned char mem39;
extern unsigned char mem50;
extern unsigned char mem51;
extern unsigned char mem53;
extern unsigned char mem56;

extern unsigned char speed;
extern unsigned char pitch;
extern int singmode;

extern unsigned char phonemeIndexOutput[60]; // tab47296
extern unsigned char stressOutput[60]; // tab47365
extern unsigned char phonemeLengthOutput[60]; // tab47416

// contains the final soundbuffer
extern uint32_t bufferpos;
extern char* buffer;

unsigned char pitches[256]; // tab43008
```

Продолжение листинга A.23

```
unsigned char frequency1[256] NOINITMEM;
unsigned char frequency2[256] NOINITMEM;
unsigned char frequency3[256] NOINITMEM;

unsigned char amplitude1[256] NOINITMEM;
unsigned char amplitude2[256] NOINITMEM;
unsigned char amplitude3[256] NOINITMEM;

unsigned char sampledConsonantFlag[256] NOINITMEM; //
tab44800

void AddInflection(unsigned char mem48, unsigned char phase1);
unsigned char trans(unsigned char mem39212, unsigned char
mem39213);

// timetable for more accurate c64 simulation
const int timetable[5][5] PROGMEM = {{162, 167, 167, 127,
128},
                                     {226, 60, 60, 0, 0},
                                     {225, 60, 59, 0, 0},
                                     {200, 0, 0, 54, 55},
                                     {199, 0, 0, 54, 54}};

static unsigned oldtimetableindex = 0;
void Output8BitAry(int index, unsigned char ary[5]) {
    int k;
    bufferpos += pgm_read_byte(&timetable[oldtimetableindex]
[index]);
    oldtimetableindex = index;
    // write a little bit in advance
    for (k = 0; k < 5; k++) buffer[bufferpos / 50 + k] =
ary[k];
}
void Output8Bit(int index, unsigned char A) {
    unsigned char ary[5] = {A, A, A, A, A};
    Output8BitAry(index, ary);
}

// written by me because of different table positions.
// mem[47] = ...
// 168=pitches
// 169=frequency1
// 170=frequency2
// 171=frequency3
// 172=amplitude1
```

Продолжение листинга А.23

```
// 173=amplitude2
// 174=amplitude3
unsigned char Read(unsigned char p, unsigned char Y) {
    unsigned char res;
    switch (p) {
        case 168:
            res = pitches[Y];
            return res;
        case 169:
            res = frequency1[Y];
            return res;
        case 170:
            res = frequency2[Y];
            return res;
        case 171:
            res = frequency3[Y];
            return res;
        case 172:
            res = amplitude1[Y];
            return res;
        case 173:
            res = amplitude2[Y];
            return res;
        case 174:
            res = amplitude3[Y];
            return res;
    }
    printf("Error reading to tables");
    return 0;
}

void Write(unsigned char p, unsigned char Y, unsigned char
value) {
    switch (p) {
        case 168:
            pitches[Y] = value;
            return;
        case 169:
            frequency1[Y] = value;
            return;
        case 170:
            frequency2[Y] = value;
            return;
        case 171:
            frequency3[Y] = value;
            return;
    }
}
```

Продолжение листинга A.23

```

        case 172:
            amplitude1[Y] = value;
            return;
        case 173:
            amplitude2[Y] = value;
            return;
        case 174:
            amplitude3[Y] = value;
            return;
    }
    printf("Error writing to tables\n");
}

//
-----
// Code48227
// Render a sampled sound from the sampleTable.
//
//   Phoneme      Sample Start      Sample End
//   32: S*        15                  255
//   33: SH        257                  511
//   34: F*        559                  767
//   35: TH        583                  767
//   36: /H        903                  1023
//   37: /X       1135                  1279
//   38: Z*        84                   119
//   39: ZH        340                  375
//   40: V*        596                  639
//   41: DH        596                  631
//
//   42: CH
//   43: **        399                  511
//
//   44: J*
//   45: **        257                  276
//   46: **
//
//   66: P*
//   67: **        743                  767
//   68: **
//
//   69: T*
//   70: **        231                  255
//   71: **
//
// The SampledPhonemesTable[] holds flags indicating if a

```

Продолжение листинга A.23

```

phoneme is
// voiced or not. If the upper 5 bits are zero, the sample is
// voiced.
//
// Samples in the sampleTable are compressed, with bits being
// converted to
// bytes from high bit to low, as follows:
//
//   unvoiced 0 bit    -> X
//   unvoiced 1 bit    -> 5
//
//   voiced 0 bit      -> 6
//   voiced 1 bit      -> 24
//
// Where X is a value from the table:
//
//   { 0x18, 0x1A, 0x17, 0x17, 0x17 };
//
// The index into this table is determined by masking off the
// lower
// 3 bits from the SampledPhonemesTable:
//
//       index = (SampledPhonemesTable[i] & 7) - 1;
//
// For voices samples, samples are interleaved between voiced
// output.

// Code48227()
void RenderSample(unsigned char* mem66) {
    int tempA;
    // current phoneme's index
    mem49 = Y;

    // mask low three bits and subtract 1 get value to
    // convert 0 bits on unvoiced samples.
    A = mem39 & 7;
    X = A - 1;

    // store the result
    mem56 = X;

    // determine which offset to use from table { 0x18, 0x1A,
    0x17, 0x17, 0x17 }
    // T, S, Z                0                0x18
    // CH, J, SH, ZH          1                0x1A
    // P, F*, V, TH, DH       2                0x17

```


Продолжение листинга А.23

```
// /H                                3                0x17
// /X                                4                0x17

// get value from the table
mem53 = pgm_read_byte(&tab48426[X]);
mem47 = X; // 46016+mem[56]*256

// voiced sample?
A = mem39 & 248;
if (A == 0) {
    // voiced phoneme: Z*, ZH, V*, DH
    Y = mem49;
    A = pitches[mem49] >> 4;

    // jump to voiced portion
    goto pos48315;
}

Y = A ^ 255;
pos48274:

// step through the 8 bits in the sample
mem56 = 8;

// get the next sample from the table
// mem47*256 = offset to start of samples
A = pgm_read_byte(&sampleTable[mem47 * 256 + Y]);
pos48280:

// left shift to get the high bit
tempA = A;
A = A << 1;
// 48281: BCC 48290

// bit not set?
if ((tempA & 128) == 0) {
    // convert the bit to value from table
    X = mem53;
    // mem[54296] = X;
    // output the byte
    Output8Bit(1, (X & 0x0f) * 16);
    // if X != 0, exit loop
    if (X != 0) goto pos48296;
}

// output a 5 for the on bit
```

Продолжение листинга А.23

```
    Output8Bit(2, 5 * 16);

    // 48295: NOP
pos48296:

    X = 0;

    // decrement counter
    mem56--;

    // if not done, jump to top of loop
    if (mem56 != 0) goto pos48280;

    // increment position
    Y++;
    if (Y != 0) goto pos48274;

    // restore values and return
    mem44 = 1;
    Y = mem49;
    return;

    unsigned char phasel;

pos48315:
    // handle voiced samples here

    // number of samples?
    phasel = A ^ 255;

    Y = *mem66;
    do {
        // pos48321:

        // shift through all 8 bits
        mem56 = 8;
        // A = Read(mem47, Y);

        // fetch value from table
        A = pgm_read_byte(&sampleTable[mem47 * 256 + Y]);

        // loop 8 times
        // pos48327:
        do {
            // 48327: ASL A
            // 48328: BCC 48337
```

Продолжение листинга A.23

```
        // left shift and check high bit
        tempA = A;
        A = A << 1;
        if ((tempA & 128) != 0) {
            // if bit set, output 26
            X = 26;
            Output8Bit(3, (X & 0xf) * 16);
        } else {
            // timetable 4
            // bit is not set, output a 6
            X = 6;
            Output8Bit(4, (X & 0xf) * 16);
        }

        mem56--;
    } while (mem56 != 0);

    // move ahead in the table
    Y++;

    // continue until counter done
    phasel++;

} while (phase1 != 0);
// if (phase1 != 0) goto pos48321;

// restore values and return
A = 1;
mem44 = 1;
*mem66 = Y;
Y = mem49;
return;
}

// RENDER THE PHONEMES IN THE LIST
//
// The phoneme list is converted into sound through the steps:
//
// 1. Copy each phoneme <length> number of times into the
frames list,
//     where each frame represents 10 milliseconds of sound.
//
// 2. Determine the transitions lengths between phonemes, and
linearly
//     interpolate the values across the frames.
```

Продолжение листинга A.23

```
//  
// 3. Offset the pitches by the fundamental frequency.  
//  
// 4. Render the each frame.  
  
// void Code47574()  
void Render() {  
    unsigned char phase1 = 0; // mem43  
    unsigned char phase2 = 0;  
    unsigned char phase3 = 0;  
    unsigned char mem66 = 0;  
    unsigned char mem38 = 0;  
    unsigned char mem40 = 0;  
    unsigned char speedcounter = 0; // mem45  
    unsigned char mem48 = 0;  
    int i;  
    if (phonemeIndexOutput[0] == 255) return; // exit if no  
data  
  
    A = 0;  
    X = 0;  
    mem44 = 0;  
  
    // CREATE FRAMES  
    //  
    // The length parameter in the list corresponds to the  
number of frames  
    // to expand the phoneme to. Each frame represents 10  
milliseconds of time.  
    // So a phoneme with a length of 7 = 7 frames = 70  
milliseconds duration.  
    //  
    // The parameters are copied from the phoneme to the frame  
verbatim.  
  
    // pos47587:  
    do {  
        // get the index  
        Y = mem44;  
        // get the phoneme at the index  
        A = phonemeIndexOutput[mem44];  
        mem56 = A;  
  
        // if terminal phoneme, exit the loop  
        if (A == 255) break;
```

Продолжение листинга А.23

```

        // period phoneme *.
        if (A == 1) {
            // add rising inflection
            A = 1;
            mem48 = 1;
            // goto pos48376;
            AddInflection(mem48, phase1);
        }
        /*
        if (A == 2) goto pos48372;
        */

        // question mark phoneme?
        if (A == 2) {
            // create falling inflection
            mem48 = 255;
            AddInflection(mem48, phase1);
        }
        // pos47615:

        // get the stress amount (more stress = higher pitch)
        phase1 = pgm_read_byte(&tab47492[stressOutput[Y] +
1]);

        // get number of frames to write
        phase2 = phonemeLengthOutput[Y];
        Y = mem56;

        // copy from the source to the frames list
        do {
            frequency1[X] = freq1data[Y]; //
F1 frequency
            frequency2[X] = freq2data[Y]; //
F2 frequency
            frequency3[X] = pgm_read_byte(&freq3data[Y]); //
F3 frequency
            amplitude1[X] = pgm_read_byte(&ampl1data[Y]); //
F1 amplitude
            amplitude2[X] = pgm_read_byte(&ampl2data[Y]); //
F2 amplitude
            amplitude3[X] = pgm_read_byte(&ampl3data[Y]); //
F3 amplitude
            sampledConsonantFlag[X] =
                sampledConsonantFlags[Y]; // phoneme data for
sampled
                                                    // consonants

```

Продолжение листинга А.23

```

        pitches[X] = pitch + phase1;    // pitch
        X++;
        phase2--;
    } while (phase2 != 0);
    mem44++;
} while (mem44 != 0);
// -----
// pos47694:

// CREATE TRANSITIONS
//
// Linear transitions are now created to smoothly connect
the
// end of one sustained portion of a phoneme to the
following
// phoneme.
//
// To do this, three tables are used:
//
// Table          Purpose
// =====
=====
// blendRank      Determines which phoneme's blend values
are used.
//
// blendOut       The number of frames at the end of the
phoneme that
//
//                will be used to transition to the
following phoneme.
//
// blendIn        The number of frames of the following
phoneme that
//
//                will be used to transition into that
phoneme.
//
// In creating a transition between two phonemes, the
phoneme
// with the HIGHEST rank is used. Phonemes are ranked on
how much
// their identity is based on their transitions. For
example,
// vowels are and diphthongs are identified by their
sustained portion,
// rather than the transitions, so they are given low
values. In contrast,
// stop consonants (P, B, T, K) and glides (Y, L) are

```

Продолжение листинга A.23

```

almost entirely
    // defined by their transitions, and are given high rank
values.
    //
    // Here are the rankings used by SAM:
    //
    //      Rank      Type                      Phonemes
    //      2         All vowels                IY, IH, etc.
    //      5         Diphthong endings         YX, WX, ER
    //      8         Terminal liquid consonants LX, WX, YX, N,
NX
    //      9         Liquid consonants         L, RX, W
    //      10        Glide                     R, OH
    //      11        Glide                     WH
    //      18        Voiceless fricatives      S, SH, F, TH
    //      20        Voiced fricatives         Z, ZH, V, DH
    //      23        Plosives, stop consonants P, T, K, KX,
DX, CH
    //      26        Stop consonants           J, GX, B, D, G
    //      27-29     Stop consonants (internal) **
    //      30        Unvoiced consonants       /H, /X and Q*
    //      160       Nasal                     M
    //
    // To determine how many frames to use, the two phonemes
are
    // compared using the blendRank[] table. The phoneme with
the
    // higher rank is selected. In case of a tie, a blend of
each is used:
    //
    //      if blendRank[phoneme1] == blendRank[phoneme2]
    //          // use lengths from each phoneme
    //          outBlendFrames = outBlend[phoneme1]
    //          inBlendFrames = outBlend[phoneme2]
    //      else if blendRank[phoneme1] > blendRank[phoneme2]
    //          // use lengths from first phoneme
    //          outBlendFrames = outBlendLength[phoneme1]
    //          inBlendFrames = inBlendLength[phoneme1]
    //      else
    //          // use lengths from the second phoneme
    //          // note that in and out are SWAPPED!
    //          outBlendFrames = inBlendLength[phoneme2]
    //          inBlendFrames = outBlendLength[phoneme2]
    //
    // Blend lengths can't be less than zero.
    //

```

Продолжение листинга A.23

```

// Transitions are assumed to be symmetrical, so if the
transition
// values for the second phoneme are used, the
inBlendLength and
// outBlendLength values are SWAPPED.
//
// For most of the parameters, SAM interpolates over the
range of the last
// outBlendFrames-1 and the first inBlendFrames.
//
// The exception to this is the Pitch[] parameter, which
is interpolates the
// pitch from the CENTER of the current phoneme to the
CENTER of the next
// phoneme.
//
// Here are two examples. First, For example, consider the
word "SUN" (S AH
// N)
//
//      Phoneme      Duration      BlendWeight      OutBlendFrames
InBlendFrames
//      S              2              18              1
3
//      AH              8              2              4
4
//      N              7              8              1
2
//
// The formant transitions for the output frames are
calculated as follows:
//
//      flags ampl1 freq1 ampl2 freq2 ampl3 freq3 pitch
//      -----
// S
//      241      0      6      0      73      0      99      61      Use
S (weight 18) for
//      transition instead of AH (weight 2) 241      0      6
0      73      0
//      99      61      <-- (OutBlendFrames-1) = (1-1) = 0 frames
// AH
//      0      2      10      2      66      0      96      59 * <--
InBlendFrames = 3
//      frames 0      4      14      3      59      0      93      57
* 0      8      18
//      5      52      0      90      55 * 0      15      22      9

```


Продолжение листинга A.23

```

44      1      87 53
      //      0      15      22      9      44      1      87      53
      //      0      15      22      9      44      1      87      53      Use
N (weight 8) for
      //      transition instead of AH (weight 2). 0      15      22
9      44      1
      //      87      53      Since N is second phoneme, reverse the
IN and OUT values.
      //      0      11      17      8      47      1      98      56 * <--
(InBlendFrames-1)
      //      = (2-1) = 1 frames
      // N
      //      0      8      12      6      50      1      109      58 * <--
OutBlendFrames = 1
      //      0      5      6      5      54      0      121      61
      //      0      5      6      5      54      0      121      61
      //      0      5      6      5      54      0      121      61
      //      0      5      6      5      54      0      121      61
      //      0      5      6      5      54      0      121      61
      //      0      5      6      5      54      0      121      61
      //
      // Now, consider the reverse "NUS" (N AH S):
      //
      //      flags ampl1 freq1 ampl2 freq2 ampl3 freq3 pitch
      //      -----
      // N
      //      0      5      6      5      54      0      121      61
      //      0      5      6      5      54      0      121      61
      //      0      5      6      5      54      0      121      61
      //      0      5      6      5      54      0      121      61
      //      0      5      6      5      54      0      121      61
      //      0      5      6      5      54      0      121      61      Use N
(weight 8) for
      //      transition instead of AH (weight 2) 0      5      6
5      54      0
      //      121      61      <-- (OutBlendFrames-1) = (1-1) = 0
frames
      // AH
      //      0      8      11      6      51      0      110      59 * <--
InBlendFrames = 2
      //      0      11      16      8      48      0      99      56 *
      //      0      15      22      9      44      1      87      53      Use S
(weight 18) for
      //      transition instead of AH (weight 2) 0      15      22
9      44      1
      //      87      53      Since S is second phoneme, reverse the

```

Продолжение листинга A.23

```

IN and OUT values.
    //      0      9      18      5      51      1      90      55 * <--
(InBlendFrames-1) =
    //      (3-1) = 2 0      4      14      3      58      1      93
57 *
    // S
    //      241      2      10      2      65      1      96      59 * <--
OutBlendFrames = 1
    //      241      0      6      0      73      0      99      61

A = 0;
mem44 = 0;
mem49 = 0; // mem49 starts at as 0
X = 0;
while (1) // while No. 1
{
    // get the current and following phoneme
    Y = phonemeIndexOutput[X];
    A = phonemeIndexOutput[X + 1];
    X++;

    // exit loop at end token
    if (A == 255) break; // goto pos47970;

    // get the ranking of each phoneme
    X = A;
    mem56 = pgm_read_byte(&blendRank[A]);
    A = pgm_read_byte(&blendRank[Y]);

    // compare the rank - lower rank value is stronger
    if (A == mem56) {
        // same rank, so use out blend lengths from each
phoneme
        phase1 = pgm_read_byte(&outBlendLength[Y]);
        phase2 = pgm_read_byte(&outBlendLength[X]);
    } else if (A < mem56) {
        // first phoneme is stronger, so us it's blend
lengths
        phase1 = pgm_read_byte(&inBlendLength[X]);
        phase2 = pgm_read_byte(&outBlendLength[X]);
    } else {
        // second phoneme is stronger, so use it's blend
lengths
        // note the out/in are swapped
        phase1 = pgm_read_byte(&outBlendLength[Y]);
        phase2 = pgm_read_byte(&inBlendLength[Y]);
    }
}

```

Продолжение листинга A.23

```

    }

    Y = mem44;
    A = mem49 + phonemeLengthOutput[mem44]; // A is mem49
+ length
    mem49 = A; // mem49 now holds length + position
    A = A + phase2; // Maybe Problem because of carry
flag
    // 47776: ADC 42
    speedcounter = A;
    mem47 = 168;
    phase3 = mem49 - phase1; // what is mem49
    A = phase1 + phase2; // total transition?
    mem38 = A;

    X = A;
    X -= 2;
    if ((X & 128) == 0)
        do // while No. 2
        {
            // pos47810:

            // mem47 is used to index the tables:
            // 168 pitches[]
            // 169 frequency1
            // 170 frequency2
            // 171 frequency3
            // 172 amplitude1
            // 173 amplitude2
            // 174 amplitude3

            mem40 = mem38;

            if (mem47 == 168) // pitch
            {
                // unlike the other values, the pitches[]
interpolates from
                // the middle of the current phoneme to
the middle of the
                // next phoneme

                unsigned char mem36, mem37;
                // half the width of the current phoneme
                mem36 = phonemeLengthOutput[mem44] >> 1;
                // half the width of the next phoneme
                mem37 = phonemeLengthOutput[mem44 + 1] >>

```

Продолжение листинга A.23

```

1;
                                // sum the values
                                mem40 = mem36 + mem37;    // length of both
halves
                                mem37 += mem49;           // center of next
phoneme
                                mem36 = mem49 - mem36;    // center index
of current phoneme
                                A = Read(mem47, mem37);   // value at
center of next phoneme
                                // - end
interpolation value

                                Y = mem36;    // start index of
interpolation
                                mem53 =
                                    A - Read(mem47,
                                                mem36); // value to center
of current phoneme
                                } else {
                                    // value to interpolate to
                                    A = Read(mem47, speedcounter);
                                    // position to start interpolation from
                                    Y = phase3;
                                    // value to interpolate from
                                    mem53 = A - Read(mem47, phase3);
                                }

                                // Code47503(mem40);
                                // ML : Code47503 is division with remainder,
and mem50 gets
                                // the sign

                                // calculate change per frame
                                signed char m53 = (signed char)mem53;
                                mem50 = mem53 & 128;
                                unsigned char m53abs = abs(m53);
                                mem51 = m53abs % mem40;    // abs((char)m53) %
mem40;
                                mem53 = (unsigned char)((signed char)(m53) /
mem40);

                                // interpolation range
                                X = mem40;    // number of frames to
interpolate over
                                Y = phase3;    // starting frame

```

Продолжение листинга A.23

```
        // linearly interpolate values

mem56 = 0;
// 47907: CLC
// pos47908:
while (1) // while No. 3
{
    A = Read(mem47, Y) + mem53; // carry
always cleared

    mem48 = A;
    Y++;
    X--;
    if (X == 0) break;

    mem56 += mem51;
    if (mem56 >= mem40) //???
    {
        mem56 -= mem40; // carry? is set
        // if ((mem56 & 128)==0)
        if ((mem50 & 128) == 0) {
            // 47935: BIT 50
            // 47937: BMI 47943
            if (mem48 != 0) mem48++;
        } else
            mem48--;
    }
    // pos47945:
    Write(mem47, Y, mem48);
} // while No. 3

// pos47952:
mem47++;
// if (mem47 != 175) goto pos47810;
} while (mem47 != 175); // while No. 2
// pos47963:
mem44++;
X = mem44;
} // while No. 1

// goto pos47701;
// pos47970:

// add the length of this phoneme
mem48 = mem49 + phonemeLengthOutput[mem44];
```

Продолжение листинга А.23

```
// ASSIGN PITCH CONTOUR
//
// This subtracts the F1 frequency from the pitch to
create a
// pitch contour. Without this, the output would be at a
single
// pitch level (monotone).

// don't adjust pitch if in sing mode
if (!singmode) {
    // iterate through the buffer
    for (i = 0; i < 256; i++) {
        // subtract half the frequency of the formant 1.
        // this adds variety to the voice
        pitches[i] -= (frequency1[i] >> 1);
    }
}

phase1 = 0;
phase2 = 0;
phase3 = 0;
mem49 = 0;
speedcounter = 72; // sam standard speed

// RESCALE AMPLITUDE
//
// Rescale volume from a linear scale to decibels.
//

// amplitude rescaling
if (debug) printf("amplitude rescaling\n");
for (i = 255; i >= 0; i--) {
    amplitude1[i] =
pgm_read_byte(&amplitudeRescale[amplitude1[i]]);
    amplitude2[i] =
pgm_read_byte(&amplitudeRescale[amplitude2[i]]);
    amplitude3[i] =
pgm_read_byte(&amplitudeRescale[amplitude3[i]]);
}

Y = 0;
A = pitches[0];
mem44 = A;
X = A;
mem38 = A - (A >> 2); // 3/4*A ???
```

Продолжение листинга A.23

```
    if (debug) {
        PrintOutput(sampledConsonantFlag, frequency1,
frequency2, frequency3,
                    amplitude1, amplitude2, amplitude3,
pitches);
    }

    if (debug) printf("process the frames\n");

    // PROCESS THE FRAMES
    //
    // In traditional vocal synthesis, the glottal pulse
drives filters, which
    // are attenuated to the frequencies of the formants.
    //
    // SAM generates these formants directly with sin and
rectangular waves.
    // To simulate them being driven by the glottal pulse, the
waveforms are
    // reset at the beginning of each glottal pulse.

    // finally the loop for sound output
    // pos48078:
    while (1) {
        // get the sampled information on the phoneme
        A = sampledConsonantFlag[Y];
        mem39 = A;

        // unvoiced sampled phoneme?
        A = A & 248;
        if (A != 0) {
            // render the sample for the phoneme
            RenderSample(&mem66);

            // skip ahead two in the phoneme buffer
            Y += 2;
            mem48 -= 2;
        } else {
            // simulate the glottal pulse and formants
            unsigned char ary[5];
            unsigned int p1 =
                phase1 *
                256; // Fixed point integers because we need
to divide later on
                unsigned int p2 = phase2 * 256;
```

Продолжение листинга А.23

```

        unsigned int p3 = phase3 * 256;
        int k;
        for (k = 0; k < 5; k++) {
            signed char sp1 =
                (signed char)pgm_read_byte(&sinus[0xff &
(p1 >> 8)]);
            signed char sp2 =
                (signed char)pgm_read_byte(&sinus[0xff &
(p2 >> 8)]);
            signed char rp3 =
                (signed char)pgm_read_byte(&rectangle[0xff
& (p3 >> 8)]);
            signed int sin1 = sp1 * ((unsigned
char)amplitude1[Y] & 0x0f);
            signed int sin2 = sp2 * ((unsigned
char)amplitude2[Y] & 0x0f);
            signed int rect = rp3 * ((unsigned
char)amplitude3[Y] & 0x0f);
            signed int mux = sin1 + sin2 + rect;
            mux /= 32;
            mux += 128; // Go from signed to unsigned
amplitude
            ary[k] = mux;
            p1 += frequency1[Y] * 256 /
                4; // Compromise, this becomes a shift
and works well
            p2 += frequency2[Y] * 256 / 4;
            p3 += frequency3[Y] * 256 / 4;
        }
        // output the accumulated value
        Output8BitAry(0, ary);
        speedcounter--;
        if (speedcounter != 0) goto pos48155;
        Y++; // go to next amplitude

        // decrement the frame count
        mem48--;
    }

    // if the frame count is zero, exit the loop
    if (mem48 == 0) {
        if (debug) printf("OUTPUT READY\n");
        return;
    }
    speedcounter = speed;
pos48155:

```


Продолжение листинга A.23

```
        // decrement the remaining length of the glottal pulse
        mem44--;

        // finished with a glottal pulse?
        if (mem44 == 0) {
        pos48159:
            // fetch the next glottal pulse length
            A = pitches[Y];
            mem44 = A;
            A = A - (A >> 2);
            mem38 = A;

            // reset the formant wave generators to keep them
in        // sync with the glottal pulse
            phase1 = 0;
            phase2 = 0;
            phase3 = 0;
            continue;
        }

        // decrement the count
        mem38--;

        // is the count non-zero and the sampled flag is zero?
        if ((mem38 != 0) || (mem39 == 0)) {
        pulse
            // reset the phase of the formants to match the

            phase1 += frequency1[Y];
            phase2 += frequency2[Y];
            phase3 += frequency3[Y];
            continue;
        }

        // voiced sampled phonemes interleave the sample with
the        // glottal pulse. The sample flag is non-zero, so
render    // the sample for the phoneme.
        // printf("2 RenderSample(mem66: %hhu)\n", mem66);
        RenderSample(&mem66);
        goto pos48159;
    } // while

    // The following code is never reached. It's left over
```

Продолжение листинга А.23

```
from when
    // the voiced sample code was part of this loop, instead
of part
    // of RenderSample();

    // pos48315:
    int tempA;
    phase1 = A ^ 255;
    Y = mem66;
    do {
        // pos48321:

        mem56 = 8;
        A = Read(mem47, Y);

        // pos48327:
        do {
            // 48327: ASL A
            // 48328: BCC 48337
            tempA = A;
            A = A << 1;
            if ((tempA & 128) != 0) {
                X = 26;
                // mem[54296] = X;
                bufferpos += 150;
                buffer[bufferpos / 50] = (X & 15) * 16;
            } else {
                // mem[54296] = 6;
                X = 6;
                bufferpos += 150;
                buffer[bufferpos / 50] = (X & 15) * 16;
            }

            for (X = wait2; X > 0; X--); // wait
            mem56--;
        } while (mem56 != 0);

        Y++;
        phase1++;

    } while (phase1 != 0);
    // if (phase1 != 0) goto pos48321;
    A = 1;
    mem44 = 1;
    mem66 = Y;
    Y = mem49;
```

Продолжение листинга А.23

```
    if (debug) printf("OUTPUT READY NEVER REACHED???\n");
    return;
}

// Create a rising or falling inflection 30 frames prior to
// index X. A rising inflection is used for questions, and
// a falling inflection is used for statements.

void AddInflection(unsigned char mem48, unsigned char phasel)
{
    // pos48372:
    //   mem48 = 255;
    // pos48376:

    // store the location of the punctuation
    mem49 = X;
    A = X;
    int Atemp = A;

    // backup 30 frames
    A = A - 30;
    // if index is before buffer, point to start of buffer
    if (Atemp <= 30) A = 0;
    X = A;

    // FIXME: Explain this fix better, it's not obvious
    // ML : A =, fixes a problem with invalid pitch with '.'
    while ((A = pitches[X]) == 127) X++;

pos48398:
    // 48398: CLC
    // 48399: ADC 48

    // add the inflection direction
    A += mem48;
    phasel = A;

    // set the inflection
    pitches[X] = A;
pos48406:

    // increment the position
    X++;

    // exit if the punctuation has been reached
    if (X == mem49) return; // goto pos47615;
```

Продолжение листинга A.23

```
    if (pitches[X] == 255) goto pos48406;
    A = phasel;
    goto pos48398;
}

/*
    SAM's voice can be altered by changing the frequencies of
    the
        mouth formant (F1) and the throat formant (F2). Only the
    voiced
        phonemes (5-29 and 48-53) are altered.
*/
void SetMouthThroat(unsigned char mouth, unsigned char throat)
{
    unsigned char initialFrequency;
    unsigned char newFrequency = 0;
    // unsigned char mouth; //mem38880
    // unsigned char throat; //mem38881

    // mouth formants (F1) 5..29
    unsigned char mouthFormants5_29[30] = {
        0, 0, 0, 0, 0, 10, 14, 19, 24, 27, 23, 21, 16,
20, 14,
        18, 14, 18, 18, 16, 13, 15, 11, 18, 14, 11, 9, 6, 6,
6};

    // throat formants (F2) 5..29
    unsigned char throatFormants5_29[30] = {
        255, 255, 255, 255, 255, 84, 73, 67, 63, 40, 44, 31,
37, 45, 73,
        49, 36, 30, 51, 37, 29, 69, 24, 50, 30, 24, 83,
46, 54, 86};

    // there must be no zeros in this 2 tables
    // formant 1 frequencies (mouth) 48..53
    unsigned char mouthFormants48_53[6] = {19, 27, 21, 27, 18,
13};

    // formant 2 frequencies (throat) 48..53
    unsigned char throatFormants48_53[6] = {72, 39, 31, 43,
30, 34};

    unsigned char pos = 5; // mem39216
    // pos38942:
    // recalculate formant frequencies 5..29 for the mouth
    (F1) and throat (F2)
```

Продолжение листинга А.23

```
while (pos != 30) {
    // recalculate mouth frequency
    initialFrequency = mouthFormants5_29[pos];
    if (initialFrequency != 0)
        newFrequency = trans(mouth, initialFrequency);
    freq1data[pos] = newFrequency;

    // recalculate throat frequency
    initialFrequency = throatFormants5_29[pos];
    if (initialFrequency != 0)
        newFrequency = trans(throat, initialFrequency);
    freq2data[pos] = newFrequency;
    pos++;
}

// pos39059:
// recalculate formant frequencies 48..53
pos = 48;
Y = 0;
while (pos != 54) {
    // recalculate F1 (mouth formant)
    initialFrequency = mouthFormants48_53[Y];
    newFrequency = trans(mouth, initialFrequency);
    freq1data[pos] = newFrequency;

    // recalculate F2 (throat formant)
    initialFrequency = throatFormants48_53[Y];
    newFrequency = trans(throat, initialFrequency);
    freq2data[pos] = newFrequency;
    Y++;
    pos++;
}

// return = (mem39212*mem39213) >> 1
unsigned char trans(unsigned char mem39212, unsigned char
mem39213) {
    // pos39008:
    unsigned char carry;
    int temp;
    unsigned char mem39214, mem39215;
    A = 0;
    mem39215 = 0;
    mem39214 = 0;
    X = 8;
    do {
```

Продолжение листинга A.23

```
        carry = mem39212 & 1;
        mem39212 = mem39212 >> 1;
        if (carry != 0) {
            /*
                39018: LSR 39212
                39021: BCC 39033
            */

            carry = 0;
            A = mem39215;
            temp = (int)A + (int)mem39213;
            A = A + mem39213;
            if (temp > 255) carry = 1;
            mem39215 = A;
        }
        temp = mem39215 & 1;
        mem39215 = (mem39215 >> 1) | (carry ? 128 : 0);
        carry = temp;
        // 39033: ROR 39215
        X--;
    } while (X != 0);
    temp = mem39214 & 128;
    mem39214 = (mem39214 << 1) | (carry ? 1 : 0);
    carry = temp;
    temp = mem39215 & 128;
    mem39215 = (mem39215 << 1) | (carry ? 1 : 0);
    carry = temp;

    return mem39215;
}
```

Листинг A.24 — Содержимое файла src/RenderTabs.c

```
#include "RenderTabs.h"

const unsigned char tab48426[5] PROGMEM = { 0x18, 0x1A, 0x17,
0x17, 0x17 };

const unsigned char tab47492[] PROGMEM =
{
    0 , 0 , 0xE0 , 0xE6 , 0xEC , 0xF3 , 0xF9 , 0 ,
    6 , 0xC , 6
};

const unsigned char amplitudeRescale[] PROGMEM =
{
```

Продолжение листинга A.24

```

    0 , 1 , 2 , 2 , 2 , 3 , 3 , 4 ,
    4 , 5 , 6 , 8 , 9 , 0xB , 0xD , 0xF, 0 //17 elements?
};

// Used to decide which phoneme's blend lengths. The candidate
// with the lower score is selected.
// tab45856
const unsigned char blendRank[] PROGMEM =
{
    0 , 0x1F , 0x1F , 0x1F , 0x1F , 2 , 2 , 2 ,
    2 , 2 , 2 , 2 , 2 , 2 , 5 , 5 ,
    2 , 0xA , 2 , 8 , 5 , 5 , 0xB , 0xA ,
    9 , 8 , 8 , 0xA0 , 8 , 8 , 0x17 , 0x1F ,
    0x12 , 0x12 , 0x12 , 0x12 , 0x1E , 0x1E , 0x14 , 0x14 ,
    0x14 , 0x14 , 0x17 , 0x17 , 0x1A , 0x1A , 0x1D , 0x1D ,
    2 , 2 , 2 , 2 , 2 , 2 , 0x1A , 0x1D ,
    0x1B , 0x1A , 0x1D , 0x1B , 0x1A , 0x1D , 0x1B , 0x1A ,
    0x1D , 0x1B , 0x17 , 0x1D , 0x17 , 0x17 , 0x1D , 0x17 ,
    0x17 , 0x1D , 0x17 , 0x17 , 0x1D , 0x17 , 0x17 , 0x17
};

// Number of frames at the end of a phoneme devoted to
// interpolating to next phoneme's final value
//tab45696
const unsigned char outBlendLength[] PROGMEM =
{
    0 , 2 , 2 , 2 , 2 , 4 , 4 , 4 ,
    4 , 4 , 4 , 4 , 4 , 4 , 4 , 4 ,
    4 , 4 , 3 , 2 , 4 , 4 , 2 , 2 ,
    2 , 2 , 2 , 1 , 1 , 1 , 1 , 1 ,
    1 , 1 , 1 , 1 , 1 , 1 , 2 , 2 ,
    2 , 1 , 0 , 1 , 0 , 1 , 0 , 5 ,
    5 , 5 , 5 , 5 , 4 , 4 , 2 , 0 ,
    1 , 2 , 0 , 1 , 2 , 0 , 1 , 2 ,
    0 , 1 , 2 , 0 , 2 , 2 , 0 , 1 ,
    3 , 0 , 2 , 3 , 0 , 2 , 0xA0 , 0xA0
};

// Number of frames at beginning of a phoneme devoted to
// interpolating to phoneme's final value
// tab45776
const unsigned char inBlendLength[] PROGMEM =
{
    0 , 2 , 2 , 2 , 2 , 4 , 4 , 4 ,

```

Продолжение листинга A.24

```

    4 , 4 , 4 , 4 , 4 , 4 , 4 , 4 ,
    4 , 4 , 3 , 3 , 4 , 4 , 3 , 3 ,
    3 , 3 , 3 , 1 , 2 , 3 , 2 , 1 ,
    3 , 3 , 3 , 3 , 1 , 1 , 3 , 3 ,
    3 , 2 , 2 , 3 , 2 , 3 , 0 , 0 ,
    5 , 5 , 5 , 5 , 4 , 4 , 2 , 0 ,
    2 , 2 , 0 , 3 , 2 , 0 , 4 , 2 ,
    0 , 3 , 2 , 0 , 2 , 2 , 0 , 2 ,
    3 , 0 , 3 , 3 , 0 , 3 , 0xB0 , 0xA0
};

// Looks like it's used as bit flags
// High bits masked by 248 (11111000)
//
// 32: S*      241      11110001
// 33: SH      226      11100010
// 34: F*      211      11010011
// 35: TH      187      10111011
// 36: /H      124      01111100
// 37: /X      149      10010101
// 38: Z*       1      00000001
// 39: ZH       2      00000010
// 40: V*       3      00000011
// 41: DH       3      00000011
// 43: **      114      01110010
// 45: **       2      00000010
// 67: **      27      00011011
// 70: **      25      00011001
// tab45936
unsigned char sampledConsonantFlags[] DATAMEM =
{
    0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
    0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
    0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
    0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
    0xF1 , 0xE2 , 0xD3 , 0xBB , 0x7C , 0x95 , 1 , 2 ,
    3 , 3 , 0 , 0x72 , 0 , 2 , 0 , 0 ,
    0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
    0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
    0 , 0 , 0 , 0x1B , 0 , 0 , 0x19 , 0 ,
    0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
};

//tab45056

```


Продолжение листинга А.24

```

unsigned char freq1data[] DATAMEM =
{
    0x00 ,0x13 ,0x13 ,0x13 ,0x13 , 0xA , 0xE ,0x12
    , 0x18 ,0x1A ,0x16 ,0x14 ,0x10 ,0x14 , 0xE ,0x12
    , 0xE ,0x12 ,0x12 ,0x10 , 0xC , 0xE , 0xA ,0x12
    , 0xE ,0xA , 8 , 6 , 6 , 6 , 6 ,0x11
    , 6 , 6 , 6 , 6 ,0xE , 0x10 , 9 ,0xA
    , 8 ,0xA , 6 , 6 , 6 , 5 , 6 , 0
    , 0x12 , 0x1A , 0x14 , 0x1A , 0x12 ,0xC , 6 , 6
    , 6 , 6 , 6 , 6 , 6 , 6 , 6 , 6
    , 6 , 6 , 6 , 6 , 6 , 6 , 6 , 6
    , 6 ,0xA ,0xA , 6 , 6 , 6 , 0x2C , 0x13
};

//tab451356
unsigned char freq2data[] DATAMEM =
{
    0x00 , 0x43 , 0x43 , 0x43 , 0x43 , 0x54 , 0x48 , 0x42 ,
    0x3E , 0x28 , 0x2C , 0x1E , 0x24 , 0x2C , 0x48 , 0x30 ,
    0x24 , 0x1E , 0x32 , 0x24 , 0x1C , 0x44 , 0x18 , 0x32 ,
    0x1E , 0x18 , 0x52 , 0x2E , 0x36 , 0x56 , 0x36 , 0x43 ,
    0x49 , 0x4F , 0x1A , 0x42 , 0x49 , 0x25 , 0x33 , 0x42 ,
    0x28 , 0x2F , 0x4F , 0x4F , 0x42 , 0x4F , 0x6E , 0x00 ,
    0x48 , 0x26 , 0x1E , 0x2A , 0x1E , 0x22 , 0x1A , 0x1A ,
    0x1A , 0x42 , 0x42 , 0x42 , 0x6E , 0x6E , 0x6E , 0x54 ,
    0x54 , 0x54 , 0x1A , 0x1A , 0x1A , 0x42 , 0x42 , 0x42 ,
    0x6D , 0x56 , 0x6D , 0x54 , 0x54 , 0x54 , 0x7F , 0x7F
};

//tab45216
const unsigned char freq3data[] PROGMEM =
{
    0x00 , 0x5B , 0x5B , 0x5B , 0x5B , 0x6E , 0x5D , 0x5B ,
    0x58 , 0x59 , 0x57 , 0x58 , 0x52 , 0x59 , 0x5D , 0x3E ,
    0x52 , 0x58 , 0x3E , 0x6E , 0x50 , 0x5D , 0x5A , 0x3C ,
    0x6E , 0x5A , 0x6E , 0x51 , 0x79 , 0x65 , 0x79 , 0x5B ,
    0x63 , 0x6A , 0x51 , 0x79 , 0x5D , 0x52 , 0x5D , 0x67 ,
    0x4C , 0x5D , 0x65 , 0x65 , 0x79 , 0x65 , 0x79 , 0x00 ,
    0x5A , 0x58 , 0x58 , 0x58 , 0x58 , 0x52 , 0x51 , 0x51 ,
    0x51 , 0x79 , 0x79 , 0x79 , 0x70 , 0x6E , 0x6E , 0x5E ,
    0x5E , 0x5E , 0x51 , 0x51 , 0x51 , 0x79 , 0x79 , 0x79 ,
    0x65 , 0x65 , 0x70 , 0x5E , 0x5E , 0x5E , 0x08 , 0x01
};

const unsigned char ampl1data[] PROGMEM =
{

```

Продолжение листинга А.24

```

    0 , 0 , 0 , 0 , 0 , 0xD , 0xD , 0xE ,
    0xF , 0xF , 0xF , 0xF , 0xF , 0xC , 0xD , 0xC ,
    0xF , 0xF , 0xD , 0xD , 0xD , 0xE , 0xD , 0xC ,
    0xD , 0xD , 0xD , 0xC , 9 , 9 , 0 , 0 ,
    0 , 0 , 0 , 0 , 0 , 0 , 0xB , 0xB ,
    0xB , 0xB , 0 , 0 , 1 , 0xB , 0 , 2 ,
    0xE , 0xF , 0xF , 0xF , 0xF , 0xD , 2 , 4 ,
    0 , 2 , 4 , 0 , 1 , 4 , 0 , 1 ,
    4 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
    0 , 0xC , 0 , 0 , 0 , 0 , 0xF , 0xF
};

const unsigned char ampl2data[] PROGMEM =
{
    0 , 0 , 0 , 0 , 0 , 0xA , 0xB , 0xD ,
    0xE , 0xD , 0xC , 0xC , 0xB , 9 , 0xB , 0xB ,
    0xC , 0xC , 0xC , 8 , 8 , 0xC , 8 , 0xA ,
    8 , 8 , 0xA , 3 , 9 , 6 , 0 , 0 ,
    0 , 0 , 0 , 0 , 0 , 0 , 3 , 5 ,
    3 , 4 , 0 , 0 , 0 , 5 , 0xA , 2 ,
    0xE , 0xD , 0xC , 0xD , 0xC , 8 , 0 , 1 ,
    0 , 0 , 1 , 0 , 0 , 1 , 0 , 0 ,
    1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
    0 , 0xA , 0 , 0 , 0xA , 0 , 0 , 0
};

const unsigned char ampl3data[] PROGMEM =
{
    0 , 0 , 0 , 0 , 0 , 8 , 7 , 8 ,
    8 , 1 , 1 , 0 , 1 , 0 , 7 , 5 ,
    1 , 0 , 6 , 1 , 0 , 7 , 0 , 5 ,
    1 , 0 , 8 , 0 , 0 , 3 , 0 , 0 ,
    0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 ,
    0 , 0 , 0 , 0 , 0 , 1 , 0xE , 1 ,
    9 , 1 , 0 , 1 , 0 , 0 , 0 , 0 ,
    0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
    0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
    0 , 7 , 0 , 0 , 5 , 0 , 0x13 , 0x10
};

//tab42240
const signed char sinus[256] PROGMEM =
{0,3,6,9,12,16,19,22,25,28,31,34,37,40,43,46,49,51,54,57,60,63,65,68,7

```

Продолжение листинга А.24

```
//tab42496
const unsigned char rectangle[] PROGMEM =
{
    0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 ,
    0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 ,
    0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 ,
    0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 ,
    0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 ,
    0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 ,
    0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 ,
    0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 ,
    0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 ,
    0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 ,
    0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 ,
    0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 ,
    0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 ,
    0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 ,
    0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 , 0x90 ,
    0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 ,
    0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 ,
    0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 ,
    0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 ,
    0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 ,
    0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 ,
    0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 ,
    0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 ,
    0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 ,
    0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 ,
    0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 ,
    0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 ,
    0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 ,
    0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 ,
    0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 , 0x70 ,
};

//random data ?
const unsigned char sampleTable[0x500] PROGMEM =
{
    //00

    0x38 , 0x84 , 0x6B , 0x19 , 0xC6 , 0x63 , 0x18 , 0x86
    , 0x73 , 0x98 , 0xC6 , 0xB1 , 0x1C , 0xCA , 0x31 , 0x8C
    , 0xC7 , 0x31 , 0x88 , 0xC2 , 0x30 , 0x98 , 0x46 , 0x31

```

Продолжение листинга А.24

```

, 0x18 , 0xC6 , 0x35 , 0xC , 0xCA , 0x31 , 0xC , 0xC6
//20
, 0x21 , 0x10 , 0x24 , 0x69 , 0x12 , 0xC2 , 0x31 , 0x14
, 0xC4 , 0x71 , 8 , 0x4A , 0x22 , 0x49 , 0xAB , 0x6A
, 0xA8 , 0xAC , 0x49 , 0x51 , 0x32 , 0xD5 , 0x52 , 0x88
, 0x93 , 0x6C , 0x94 , 0x22 , 0x15 , 0x54 , 0xD2 , 0x25
//40
, 0x96 , 0xD4 , 0x50 , 0xA5 , 0x46 , 0x21 , 8 , 0x85
, 0x6B , 0x18 , 0xC4 , 0x63 , 0x10 , 0xCE , 0x6B , 0x18
, 0x8C , 0x71 , 0x19 , 0x8C , 0x63 , 0x35 , 0xC , 0xC6
, 0x33 , 0x99 , 0xCC , 0x6C , 0xB5 , 0x4E , 0xA2 , 0x99
//60
, 0x46 , 0x21 , 0x28 , 0x82 , 0x95 , 0x2E , 0xE3 , 0x30
, 0x9C , 0xC5 , 0x30 , 0x9C , 0xA2 , 0xB1 , 0x9C , 0x67
, 0x31 , 0x88 , 0x66 , 0x59 , 0x2C , 0x53 , 0x18 , 0x84
, 0x67 , 0x50 , 0xCA , 0xE3 , 0xA , 0xAC , 0xAB , 0x30
//80
, 0xAC , 0x62 , 0x30 , 0x8C , 0x63 , 0x10 , 0x94 , 0x62
, 0xB1 , 0x8C , 0x82 , 0x28 , 0x96 , 0x33 , 0x98 , 0xD6
, 0xB5 , 0x4C , 0x62 , 0x29 , 0xA5 , 0x4A , 0xB5 , 0x9C
, 0xC6 , 0x31 , 0x14 , 0xD6 , 0x38 , 0x9C , 0x4B , 0xB4
//A0
, 0x86 , 0x65 , 0x18 , 0xAE , 0x67 , 0x1C , 0xA6 , 0x63
, 0x19 , 0x96 , 0x23 , 0x19 , 0x84 , 0x13 , 8 , 0xA6
, 0x52 , 0xAC , 0xCA , 0x22 , 0x89 , 0x6E , 0xAB , 0x19
, 0x8C , 0x62 , 0x34 , 0xC4 , 0x62 , 0x19 , 0x86 , 0x63
//C0
, 0x18 , 0xC4 , 0x23 , 0x58 , 0xD6 , 0xA3 , 0x50 , 0x42
, 0x54 , 0x4A , 0xAD , 0x4A , 0x25 , 0x11 , 0x6B , 0x64
, 0x89 , 0x4A , 0x63 , 0x39 , 0x8A , 0x23 , 0x31 , 0x2A
, 0xEA , 0xA2 , 0xA9 , 0x44 , 0xC5 , 0x12 , 0xCD , 0x42
//E0
, 0x34 , 0x8C , 0x62 , 0x18 , 0x8C , 0x63 , 0x11 , 0x48
, 0x66 , 0x31 , 0x9D , 0x44 , 0x33 , 0x1D , 0x46 , 0x31
, 0x9C , 0xC6 , 0xB1 , 0xC , 0xCD , 0x32 , 0x88 , 0xC4
, 0x73 , 0x18 , 0x86 , 0x73 , 8 , 0xD6 , 0x63 , 0x58
//100
, 7 , 0x81 , 0xE0 , 0xF0 , 0x3C , 7 , 0x87 , 0x90
, 0x3C , 0x7C , 0xF , 0xC7 , 0xC0 , 0xC0 , 0xF0 , 0x7C
, 0x1E , 7 , 0x80 , 0x80 , 0 , 0x1C , 0x78 , 0x70
, 0xF1 , 0xC7 , 0x1F , 0xC0 , 0xC , 0xFE , 0x1C , 0x1F
//120
, 0x1F , 0xE , 0xA , 0x7A , 0xC0 , 0x71 , 0xF2 , 0x83
, 0x8F , 3 , 0xF , 0xF , 0xC , 0 , 0x79 , 0xF8
, 0x61 , 0xE0 , 0x43 , 0xF , 0x83 , 0xE7 , 0x18 , 0xF9
, 0xC1 , 0x13 , 0xDA , 0xE9 , 0x63 , 0x8F , 0xF , 0x83

```

Продолжение листинга А.24

```
//140
, 0x83 , 0x87 , 0xC3 , 0x1F , 0x3C , 0x70 , 0xF0 , 0xE1
, 0xE1 , 0xE3 , 0x87 , 0xB8 , 0x71 , 0xE , 0x20 , 0xE3
, 0x8D , 0x48 , 0x78 , 0x1C , 0x93 , 0x87 , 0x30 , 0xE1
, 0xC1 , 0xC1 , 0xE4 , 0x78 , 0x21 , 0x83 , 0x83 , 0xC3
//160
, 0x87 , 6 , 0x39 , 0xE5 , 0xC3 , 0x87 , 7 , 0xE
, 0x1C , 0x1C , 0x70 , 0xF4 , 0x71 , 0x9C , 0x60 , 0x36
, 0x32 , 0xC3 , 0x1E , 0x3C , 0xF3 , 0x8F , 0xE , 0x3C
, 0x70 , 0xE3 , 0xC7 , 0x8F , 0xF , 0xF , 0xE , 0x3C
//180
, 0x78 , 0xF0 , 0xE3 , 0x87 , 6 , 0xF0 , 0xE3 , 7
, 0xC1 , 0x99 , 0x87 , 0xF , 0x18 , 0x78 , 0x70 , 0x70
, 0xFC , 0xF3 , 0x10 , 0xB1 , 0x8C , 0x8C , 0x31 , 0x7C
, 0x70 , 0xE1 , 0x86 , 0x3C , 0x64 , 0x6C , 0xB0 , 0xE1
//1A0
, 0xE3 , 0xF , 0x23 , 0x8F , 0xF , 0x1E , 0x3E , 0x38
, 0x3C , 0x38 , 0x7B , 0x8F , 7 , 0xE , 0x3C , 0xF4
, 0x17 , 0x1E , 0x3C , 0x78 , 0xF2 , 0x9E , 0x72 , 0x49
, 0xE3 , 0x25 , 0x36 , 0x38 , 0x58 , 0x39 , 0xE2 , 0xDE
//1C0
, 0x3C , 0x78 , 0x78 , 0xE1 , 0xC7 , 0x61 , 0xE1 , 0xE1
, 0xB0 , 0xF0 , 0xF0 , 0xC3 , 0xC7 , 0xE , 0x38 , 0xC0
, 0xF0 , 0xCE , 0x73 , 0x73 , 0x18 , 0x34 , 0xB0 , 0xE1
, 0xC7 , 0x8E , 0x1C , 0x3C , 0xF8 , 0x38 , 0xF0 , 0xE1
//1E0
, 0xC1 , 0x8B , 0x86 , 0x8F , 0x1C , 0x78 , 0x70 , 0xF0
, 0x78 , 0xAC , 0xB1 , 0x8F , 0x39 , 0x31 , 0xDB , 0x38
, 0x61 , 0xC3 , 0xE , 0xE , 0x38 , 0x78 , 0x73 , 0x17
, 0x1E , 0x39 , 0x1E , 0x38 , 0x64 , 0xE1 , 0xF1 , 0xC1
//200
, 0x4E , 0xF , 0x40 , 0xA2 , 2 , 0xC5 , 0x8F , 0x81
, 0xA1 , 0xFC , 0x12 , 8 , 0x64 , 0xE0 , 0x3C , 0x22
, 0xE0 , 0x45 , 7 , 0x8E , 0xC , 0x32 , 0x90 , 0xF0
, 0x1F , 0x20 , 0x49 , 0xE0 , 0xF8 , 0xC , 0x60 , 0xF0
//220
, 0x17 , 0x1A , 0x41 , 0xAA , 0xA4 , 0xD0 , 0x8D , 0x12
, 0x82 , 0x1E , 0x1E , 3 , 0xF8 , 0x3E , 3 , 0xC
, 0x73 , 0x80 , 0x70 , 0x44 , 0x26 , 3 , 0x24 , 0xE1
, 0x3E , 4 , 0x4E , 4 , 0x1C , 0xC1 , 9 , 0xCC
//240
, 0x9E , 0x90 , 0x21 , 7 , 0x90 , 0x43 , 0x64 , 0xC0
, 0xF , 0xC6 , 0x90 , 0x9C , 0xC1 , 0x5B , 3 , 0xE2
, 0x1D , 0x81 , 0xE0 , 0x5E , 0x1D , 3 , 0x84 , 0xB8
, 0x2C , 0xF , 0x80 , 0xB1 , 0x83 , 0xE0 , 0x30 , 0x41
//260
```

Продолжение листинга А.24

```
, 0x1E , 0x43 , 0x89 , 0x83 , 0x50 , 0xFC , 0x24 , 0x2E
, 0x13 , 0x83 , 0xF1 , 0x7C , 0x4C , 0x2C , 0xC9 , 0xD
, 0x83 , 0xB0 , 0xB5 , 0x82 , 0xE4 , 0xE8 , 6 , 0x9C
, 7 , 0xA0 , 0x99 , 0x1D , 7 , 0x3E , 0x82 , 0x8F
//280
, 0x70 , 0x30 , 0x74 , 0x40 , 0xCA , 0x10 , 0xE4 , 0xE8
, 0xF , 0x92 , 0x14 , 0x3F , 6 , 0xF8 , 0x84 , 0x88
, 0x43 , 0x81 , 0xA , 0x34 , 0x39 , 0x41 , 0xC6 , 0xE3
, 0x1C , 0x47 , 3 , 0xB0 , 0xB8 , 0x13 , 0xA , 0xC2
//2A0
, 0x64 , 0xF8 , 0x18 , 0xF9 , 0x60 , 0xB3 , 0xC0 , 0x65
, 0x20 , 0x60 , 0xA6 , 0x8C , 0xC3 , 0x81 , 0x20 , 0x30
, 0x26 , 0x1E , 0x1C , 0x38 , 0xD3 , 1 , 0xB0 , 0x26
, 0x40 , 0xF4 , 0xB , 0xC3 , 0x42 , 0x1F , 0x85 , 0x32
//2C0
, 0x26 , 0x60 , 0x40 , 0xC9 , 0xCB , 1 , 0xEC , 0x11
, 0x28 , 0x40 , 0xFA , 4 , 0x34 , 0xE0 , 0x70 , 0x4C
, 0x8C , 0x1D , 7 , 0x69 , 3 , 0x16 , 0xC8 , 4
, 0x23 , 0xE8 , 0xC6 , 0x9A , 0xB , 0x1A , 3 , 0xE0
//2E0
, 0x76 , 6 , 5 , 0xCF , 0x1E , 0xBC , 0x58 , 0x31
, 0x71 , 0x66 , 0 , 0xF8 , 0x3F , 4 , 0xFC , 0xC
, 0x74 , 0x27 , 0x8A , 0x80 , 0x71 , 0xC2 , 0x3A , 0x26
, 6 , 0xC0 , 0x1F , 5 , 0xF , 0x98 , 0x40 , 0xAE
//300
, 1 , 0x7F , 0xC0 , 7 , 0xFF , 0 , 0xE , 0xFE
, 0 , 3 , 0xDF , 0x80 , 3 , 0xEF , 0x80 , 0x1B
, 0xF1 , 0xC2 , 0 , 0xE7 , 0xE0 , 0x18 , 0xFC , 0xE0
, 0x21 , 0xFC , 0x80 , 0x3C , 0xFC , 0x40 , 0xE , 0x7E
//320
, 0 , 0x3F , 0x3E , 0 , 0xF , 0xFE , 0 , 0x1F
, 0xFF , 0 , 0x3E , 0xF0 , 7 , 0xFC , 0 , 0x7E
, 0x10 , 0x3F , 0xFF , 0 , 0x3F , 0x38 , 0xE , 0x7C
, 1 , 0x87 , 0xC , 0xFC , 0xC7 , 0 , 0x3E , 4
//340
, 0xF , 0x3E , 0x1F , 0xF , 0xF , 0x1F , 0xF , 2
, 0x83 , 0x87 , 0xCF , 3 , 0x87 , 0xF , 0x3F , 0xC0
, 7 , 0x9E , 0x60 , 0x3F , 0xC0 , 3 , 0xFE , 0
, 0x3F , 0xE0 , 0x77 , 0xE1 , 0xC0 , 0xFE , 0xE0 , 0xC3
//360
, 0xE0 , 1 , 0xDF , 0xF8 , 3 , 7 , 0 , 0x7E
, 0x70 , 0 , 0x7C , 0x38 , 0x18 , 0xFE , 0xC , 0x1E
, 0x78 , 0x1C , 0x7C , 0x3E , 0xE , 0x1F , 0x1E , 0x1E
, 0x3E , 0 , 0x7F , 0x83 , 7 , 0xDB , 0x87 , 0x83
//380
, 7 , 0xC7 , 7 , 0x10 , 0x71 , 0xFF , 0 , 0x3F
```

Продолжение листинга А.24

```

, 0xE2 , 1 , 0xE0 , 0xC1 , 0xC3 , 0xE1 , 0 , 0x7F
, 0xC0 , 5 , 0xF0 , 0x20 , 0xF8 , 0xF0 , 0x70 , 0xFE
, 0x78 , 0x79 , 0xF8 , 2 , 0x3F , 0xC , 0x8F , 3
//3a0
, 0xF , 0x9F , 0xE0 , 0xC1 , 0xC7 , 0x87 , 3 , 0xC3
, 0xC3 , 0xB0 , 0xE1 , 0xE1 , 0xC1 , 0xE3 , 0xE0 , 0x71
, 0xF0 , 0 , 0xFC , 0x70 , 0x7C , 0xC , 0x3E , 0x38
, 0xE , 0x1C , 0x70 , 0xC3 , 0xC7 , 3 , 0x81 , 0xC1
//3c0
, 0xC7 , 0xE7 , 0 , 0xF , 0xC7 , 0x87 , 0x19 , 9
, 0xEF , 0xC4 , 0x33 , 0xE0 , 0xC1 , 0xFC , 0xF8 , 0x70
, 0xF0 , 0x78 , 0xF8 , 0xF0 , 0x61 , 0xC7 , 0 , 0x1F
, 0xF8 , 1 , 0x7C , 0xF8 , 0xF0 , 0x78 , 0x70 , 0x3C
//3e0
, 0x7C , 0xCE , 0xE , 0x21 , 0x83 , 0xCF , 8 , 7
, 0x8F , 8 , 0xC1 , 0x87 , 0x8F , 0x80 , 0xC7 , 0xE3
, 0 , 7 , 0xF8 , 0xE0 , 0xEF , 0 , 0x39 , 0xF7
, 0x80 , 0xE , 0xF8 , 0xE1 , 0xE3 , 0xF8 , 0x21 , 0x9F
//400
, 0xC0 , 0xFF , 3 , 0xF8 , 7 , 0xC0 , 0x1F , 0xF8
, 0xC4 , 4 , 0xFC , 0xC4 , 0xC1 , 0xBC , 0x87 , 0xF0
, 0xF , 0xC0 , 0x7F , 5 , 0xE0 , 0x25 , 0xEC , 0xC0
, 0x3E , 0x84 , 0x47 , 0xF0 , 0x8E , 3 , 0xF8 , 3
//420
, 0xFB , 0xC0 , 0x19 , 0xF8 , 7 , 0x9C , 0xC , 0x17
, 0xF8 , 7 , 0xE0 , 0x1F , 0xA1 , 0xFC , 0xF , 0xFC
, 1 , 0xF0 , 0x3F , 0 , 0xFE , 3 , 0xF0 , 0x1F
, 0 , 0xFD , 0 , 0xFF , 0x88 , 0xD , 0xF9 , 1
//440
, 0xFF , 0 , 0x70 , 7 , 0xC0 , 0x3E , 0x42 , 0xF3
, 0xD , 0xC4 , 0x7F , 0x80 , 0xFC , 7 , 0xF0 , 0x5E
, 0xC0 , 0x3F , 0 , 0x78 , 0x3F , 0x81 , 0xFF , 1
, 0xF8 , 1 , 0xC3 , 0xE8 , 0xC , 0xE4 , 0x64 , 0x8F
////460
, 0xE4 , 0xF , 0xF0 , 7 , 0xF0 , 0xC2 , 0x1F , 0
, 0x7F , 0xC0 , 0x6F , 0x80 , 0x7E , 3 , 0xF8 , 7
, 0xF0 , 0x3F , 0xC0 , 0x78 , 0xF , 0x82 , 7 , 0xFE
, 0x22 , 0x77 , 0x70 , 2 , 0x76 , 3 , 0xFE , 0
//480
, 0xFE , 0x67 , 0 , 0x7C , 0xC7 , 0xF1 , 0x8E , 0xC6
, 0x3B , 0xE0 , 0x3F , 0x84 , 0xF3 , 0x19 , 0xD8 , 3
, 0x99 , 0xFC , 9 , 0xB8 , 0xF , 0xF8 , 0 , 0x9D
, 0x24 , 0x61 , 0xF9 , 0xD , 0 , 0xFD , 3 , 0xF0
//4a0
, 0x1F , 0x90 , 0x3F , 1 , 0xF8 , 0x1F , 0xD0 , 0xF
, 0xF8 , 0x37 , 1 , 0xF8 , 7 , 0xF0 , 0xF , 0xC0

```

Продолжение листинга A.24

```
, 0x3F , 0 , 0xFE , 3 , 0xF8 , 0xF , 0xC0 , 0x3F
, 0 , 0xFA , 3 , 0xF0 , 0xF , 0x80 , 0xFF , 1
//4c0
, 0xB8 , 7 , 0xF0 , 1 , 0xFC , 1 , 0xBC , 0x80
, 0x13 , 0x1E , 0 , 0x7F , 0xE1 , 0x40 , 0x7F , 0xA0
, 0x7F , 0xB0 , 0 , 0x3F , 0xC0 , 0x1F , 0xC0 , 0x38
, 0xF , 0xF0 , 0x1F , 0x80 , 0xFF , 1 , 0xFC , 3
//4e0
, 0xF1 , 0x7E , 1 , 0xFE , 1 , 0xF0 , 0xFF , 0
, 0x7F , 0xC0 , 0x1D , 7 , 0xF0 , 0xF , 0xC0 , 0x7E
, 6 , 0xE0 , 7 , 0xE0 , 0xF , 0xF8 , 6 , 0xC1
, 0xFE , 1 , 0xFC , 3 , 0xE0 , 0xF , 0 , 0xFC
};
```

Листинг A.25 — Содержимое файла src/sam.c

```
#include "sam.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char input[256] NOINITMEM; // tab39445
// standard sam sound
unsigned char speed DATAMEM = 72;
unsigned char pitch DATAMEM = 64;
unsigned char mouth DATAMEM = 128;
unsigned char throat DATAMEM = 128;
int singmode DATAMEM = 0;

extern int debug;

unsigned char mem39 NOINITMEM;
unsigned char mem44 NOINITMEM;
unsigned char mem47 NOINITMEM;
unsigned char mem49 NOINITMEM;
unsigned char mem50 NOINITMEM;
unsigned char mem51 NOINITMEM;
unsigned char mem53 NOINITMEM;
unsigned char mem56 NOINITMEM;

unsigned char mem59 DATAMEM = 0;

unsigned char B NOINITMEM;
unsigned char R NOINITMEM;
unsigned char S NOINITMEM;
```


Продолжение листинга А.25

```
unsigned char stress[256] NOINITMEM;          // numbers from 0
to 8
unsigned char phonemeLength[256] NOINITMEM;   // tab40160
unsigned char phonemeindex[256] NOINITMEM;

unsigned char phonemeIndexOutput[60] NOINITMEM; // tab47296
unsigned char stressOutput[60] NOINITMEM;       // tab47365
unsigned char phonemeLengthOutput[60] NOINITMEM; // tab47416

// contains the final soundbuffer
uint32_t bufferpos DATAMEM = 0;
char* buffer DATAMEM = NULL;

void SetInput(char* _input) {
    int i, l;
    if (debug) {
        printf("set input:\n");
        for (i = 0; i <= strlen(_input); ++i) printf("%hhu ",
_input[i]);
        printf("\n");
    }
    l = strlen(_input);
    if (l > 254) l = 254;
    for (i = 0; i < l; i++) input[i] = _input[i];
    input[l] = 0;
}

void SetSpeed(unsigned char _speed) { speed = _speed; }
void SetPitch(unsigned char _pitch) { pitch = _pitch; }
void SetMouth(unsigned char _mouth) { mouth = _mouth; }
void SetThroat(unsigned char _throat) { throat = _throat; }
void EnableSingmode() { singmode = 1; }
char* GetBuffer() { return buffer; }
uint32_t GetBufferLength() { return bufferpos; }

int Init();
int Parser1();
void Parser2();
int SAMMain();
void CopyStress();
void SetPhonemeLength();
void AdjustLengths();
void Code41240();
void Insert(unsigned char position, unsigned char mem60,
unsigned char mem59,
```

Продолжение листинга A.25

```
        unsigned char mem58);
void InsertBreath();
void PrepareOutput();
void SetMouthThroat(unsigned char mouth, unsigned char
throat);

// 168=pitches
// 169=frequency1
// 170=frequency2
// 171=frequency3
// 172=amplitude1
// 173=amplitude2
// 174=amplitude3

int Init() {
    int i;
    SetMouthThroat(mouth, throat);

    bufferpos = 0;
    // TODO, check for free the memory, 10 seconds of output
    should be more than
    // enough Max HEAP size is ~59KB, so 22050*1 = ~21.5KB
    buffer = malloc(SAMPLE_RATE * 1u);
    if (buffer == NULL) {
        printf("Cannot alloc memory for buffer :(\n");
        return 1;
    }

    /*
    freq2data = &mem[45136];
    freq1data = &mem[45056];
    freq3data = &mem[45216];
    */
    // pitches = &mem[43008];
    /*
    frequency1 = &mem[43264];
    frequency2 = &mem[43520];
    frequency3 = &mem[43776];
    */
    /*
    amplitude1 = &mem[44032];
    amplitude2 = &mem[44288];
    amplitude3 = &mem[44544];
    */
    // phoneme = &mem[39904];
    /*
```

Продолжение листинга А.25

```
    ampl1data = &mem[45296];
    ampl2data = &mem[45376];
    ampl3data = &mem[45456];
    */

    for (i = 0; i < 256; i++) {
        stress[i] = 0;
        phonemeLength[i] = 0;
    }

    for (i = 0; i < 60; i++) {
        phonemeIndexOutput[i] = 0;
        stressOutput[i] = 0;
        phonemeLengthOutput[i] = 0;
    }
    phonemeindex[255] = 255; // to prevent buffer overflow //
ML : changed from                // 32 to 255 to stop freezing
with long inputs
    return 0;
}

// int Code39771()
int SAMMain() {
    if (Init()) {
        return 1;
    }
    phonemeindex[255] = 32; // to prevent buffer overflow

    if (!Parser1()) return 0;
    if (debug) {
        PrintPhonemes(phonemeindex, phonemeLength, stress);
    }
    Parser2();
    CopyStress();
    SetPhonemeLength();
    AdjustLengths();
    Code41240();
    do {
        B = phonemeindex[R];
        if (B > 80) {
            phonemeindex[R] = 255;
            break; // error: delete all behind it
        }
        R++;
    } while (R != 0);
```

Продолжение листинга A.25

```
// pos39848:
InsertBreath();

// mem[40158] = 255;
if (debug) {
    PrintPhonemes(phonemeindex, phonemeLength, stress);
}

PrepareOutput();

// free(buffer);
return 1;
}

// void Code48547()
void PrepareOutput() {
    B = 0;
    R = 0;
    S = 0;

    // pos48551:
    while (1) {
        B = phonemeindex[R];
        if (B == 255) {
            B = 255;
            phonemeIndexOutput[S] = 255;
            Render();
            return;
        }
        if (B == 254) {
            R++;
            int temp = R;
            // mem[48546] = X;
            phonemeIndexOutput[S] = 255;
            Render();
            // X = mem[48546];
            R = temp;
            S = 0;
            continue;
        }

        if (B == 0) {
            R++;
            continue;
        }
    }
}
```

Продолжение листинга A.25

```

        phonemeIndexOutput[S] = B;
        phonemeLengthOutput[S] = phonemeLength[R];
        stressOutput[S] = stress[R];
        R++;
        S++;
    }
}

// void Code48431()
void InsertBreath() {
    unsigned char mem54;
    unsigned char mem55;
    unsigned char index; // variable Y
    mem54 = 255;
    R++;
    mem55 = 0;
    unsigned char mem66 = 0;
    while (1) {
        // pos48440:
        R = mem66;
        index = phonemeindex[R];
        if (index == 255) return;
        mem55 += phonemeLength[R];

        if (mem55 < 232) {
            if (index != 254) // ML : Prevents an index out
of bounds problem
            {
                B = pgm_read_byte(&sam_flags2[index]) & 1;
                if (B != 0) {
                    R++;
                    mem55 = 0;
                    Insert(R, 254, mem59, 0);
                    mem66++;
                    mem66++;
                    continue;
                }
            }
            if (index == 0) mem54 = R;
            mem66++;
            continue;
        }
        R = mem54;
        phonemeindex[R] = 31; // 'Q*' glottal stop
        phonemeLength[R] = 4;
    }
}

```

Продолжение листинга A.25

```
        stress[R] = 0;
        R++;
        mem55 = 0;
        Insert(R, 254, mem59, 0);
        R++;
        mem66 = R;
    }
}

// Iterates through the phoneme buffer, copying the stress
// value from
// the following phoneme under the following circumstance:

//      1. The current phoneme is voiced, excluding plosives
// and fricatives
//      2. The following phoneme is voiced, excluding plosives
// and fricatives,
//      and
//      3. The following phoneme is stressed
//
// In those cases, the stress value+1 from the following
// phoneme is copied.
//
// For example, the word LOITER is represented as LOY5TER,
// with as stress
// of 5 on the diphtong OY. This routine will copy the stress
// value of 6 (5+1)
// to the L that precedes it.

// void Code41883()
void CopyStress() {
    // loop through all the phonemes to be output
    unsigned char pos = 0; // mem66
    while (1) {
        // get the phoneme
        S = phonemeindex[pos];

        // exit at end of buffer
        if (S == 255) return;

        // if CONSONANT_FLAG set, skip - only vowels get
stress
        if ((pgm_read_byte(&sam_flags[S]) & 64) == 0) {
            pos++;
            continue;
        }
    }
}
```

Продолжение листинга А.25

```

        // get the next phoneme
        S = phonemeindex[pos + 1];
        if (S == 255) // prevent buffer overflow
        {
            pos++;
            continue;
        } else
            // if the following phoneme is a vowel, skip
            if ((pgm_read_byte(&sam_flags[S]) & 128) == 0) {
                pos++;
                continue;
            }

        // get the stress value at the next position
        S = stress[pos + 1];

        // if next phoneme is not stressed, skip
        if (S == 0) {
            pos++;
            continue;
        }

        // if next phoneme is not a VOWEL OR ER, skip
        if ((S & 128) != 0) {
            pos++;
            continue;
        }

        // copy stress from prior phoneme to this one
        stress[pos] = S + 1;

        // advance pointer
        pos++;
    }
}

// void Code41014()
void Insert(unsigned char position /*var57*/, unsigned char
mem60,
            unsigned char mem59, unsigned char mem58) {
    int i;
    for (i = 253; i >= position;
        i--) // ML : always keep last safe-guarding 255
    {
        phonemeindex[i + 1] = phonemeindex[i];
        phonemeLength[i + 1] = phonemeLength[i];
    }
}

```

Продолжение листинга A.25

```
        stress[i + 1] = stress[i];
    }

    phonemeindex[position] = mem60;
    phonemelength[position] = mem59;
    stress[position] = mem58;
    return;
}

// The input[] buffer contains a string of phonemes and stress
// markers along
// the lines of:
//
//      DHAX KAET IHZ AH5GLIY. <0x9B>
//
// The byte 0x9B marks the end of the buffer. Some phonemes
// are 2 bytes
// long, such as "DH" and "AX". Others are 1 byte long, such
// as "T" and "Z".
// There are also stress markers, such as "5" and ".".
//
// The first character of the phonemes are stored in the table
// signInputTable1[]. The second character of the phonemes are
// stored in the
// table signInputTable2[]. The stress characters are arranged
// in low to high
// stress order in stressInputTable[].
//
// The following process is used to parse the input[] buffer:
//
// Repeat until the <0x9B> character is reached:
//
//      First, a search is made for a 2 character match for
// phonemes that do
//      not end with the '*' (wildcard) character. On a
// match, the index of
//      the phoneme is added to phonemeIndex[] and the
// buffer position is
//      advanced 2 bytes.
//
//      If this fails, a search is made for a 1 character
// match against all
//      phoneme names ending with a '*' (wildcard). If this
// succeeds, the
//      phoneme is added to phonemeIndex[] and the buffer
// position is advanced
```


Продолжение листинга A.25

```
//          1 byte.
//
//          If this fails, search for a 1 character match in the
//          stressInputTable[]. If this succeeds, the stress
value is placed in
//          the last stress[] table at the same index of the
last added phoneme,
//          and the buffer position is advanced by 1 byte.
//
//          If this fails, return a 0.
//
// On success:
//
// 1. phonemeIndex[] will contain the index of all the
phonemes.
// 2. The last index in phonemeIndex[] will be 255.
// 3. stress[] will contain the stress value for each
phoneme

// input[] holds the string of phonemes, each two bytes wide
// signInputTable1[] holds the first character of each phoneme
// signInputTable2[] holds the second character of each phoneme
// phonemeIndex[] holds the indexes of the phonemes after
parsing input[]
//
// The parser scans through the input[], finding the names of
the phonemes
// by searching signInputTable1[] and signInputTable2[]. On a
match, it
// copies the index of the phoneme into the
phonemeIndexTable[].
//
// The character <0x9B> marks the end of text in input[]. When
it is reached,
// the index 255 is placed at the end of the
phonemeIndexTable[], and the
// function returns with a 1 indicating success.
int Parser1() {
    int i;
    unsigned char sign1;
    unsigned char sign2;
    unsigned char position = 0;
    R = 0;
    B = 0;
    S = 0;
```

Продолжение листинга A.25

```

// CLEAR THE STRESS TABLE
for (i = 0; i < 256; i++) stress[i] = 0;
// THIS CODE MATCHES THE PHONEME LETTERS TO THE TABLE
// pos41078:
while (1) {
    // GET THE FIRST CHARACTER FROM THE PHONEME BUFFER
    sign1 = input[R];
    // TEST FOR 155 (◆) END OF LINE MARKER
    if (sign1 == 155) {
        // MARK ENDPOINT AND RETURN
        phonemeindex[position] = 255; // mark endpoint
        // REACHED END OF PHONEMES, SO EXIT
        return 1; // all ok
    }

    // GET THE NEXT CHARACTER FROM THE BUFFER
    R++;
    sign2 = input[R];

    // NOW sign1 = FIRST CHARACTER OF PHONEME, AND sign2 =
SECOND CHARACTER
    // OF PHONEME

    // TRY TO MATCH PHONEMES ON TWO TWO-CHARACTER NAME
    // IGNORE PHONEMES IN TABLE ENDING WITH WILDCARDS

    // SET INDEX TO 0
    S = 0;
pos41095:

    // GET FIRST CHARACTER AT POSITION Y IN signInputTable
    // --> should change name to PhonemeNameTable1
    B = pgm_read_byte(&signInputTable1[S]);

    // FIRST CHARACTER MATCHES?
    if (B == sign1) {
        // GET THE CHARACTER FROM THE
PhonemeSecondLetterTable
        B = pgm_read_byte(&signInputTable2[S]);
        // NOT A SPECIAL AND MATCHES SECOND CHARACTER?
        if ((B != '*') && (B == sign2)) {
            // STORE THE INDEX OF THE PHONEME INTO THE
phomeneIndexTable
            phonemeindex[position] = S;

            // ADVANCE THE POINTER TO THE

```

Продолжение листинга A.25

```

phonemeIndexTable
    position++;
    // ADVANCE THE POINTER TO THE
phonemeInputBuffer
    R++;

    // CONTINUE PARSING
    continue;
}
}

// NO MATCH, TRY TO MATCH ON FIRST CHARACTER TO
WILDCARD NAMES (ENDING
// WITH '*')

// ADVANCE TO THE NEXT POSITION
S++;
// IF NOT END OF TABLE, CONTINUE
if (S != 81) goto pos41095;

// REACHED END OF TABLE WITHOUT AN EXACT (2 CHARACTER)
MATCH.
// THIS TIME, SEARCH FOR A 1 CHARACTER MATCH AGAINST
THE WILDCARDS

// RESET THE INDEX TO POINT TO THE START OF THE
PHONEME NAME TABLE
S = 0;
pos41134:
// DOES THE PHONEME IN THE TABLE END WITH '*'?
if (pgm_read_byte(&signInputTable2[S]) == '*') {
    // DOES THE FIRST CHARACTER MATCH THE FIRST LETTER
OF THE PHONEME
    if (pgm_read_byte(&signInputTable1[S]) == sign1) {
        // SAVE THE POSITION AND MOVE AHEAD
        phonemeindex[position] = S;

        // ADVANCE THE POINTER
        position++;

        // CONTINUE THROUGH THE LOOP
        continue;
    }
}
S++;
if (S != 81) goto pos41134; // 81 is size of PHONEME

```

Продолжение листинга А.25

```

NAME table

        // FAILED TO MATCH WITH A WILDCARD. ASSUME THIS IS A
STRESS
        // CHARACTER. SEARCH THROUGH THE STRESS TABLE

        // SET INDEX TO POSITION 8 (END OF STRESS TABLE)
        S = 8;

        // WALK BACK THROUGH TABLE LOOKING FOR A MATCH
        while ((sign1 != pgm_read_byte(&stressInputTable[S]))
&& (S > 0)) {
            // DECREMENT INDEX
            S--;
        }

        // REACHED THE END OF THE SEARCH WITHOUT BREAKING OUT
OF LOOP?
        if (S == 0) {
            // mem[39444] = X;
            // 41181: JSR 42043 //Error
            // FAILED TO MATCH ANYTHING, RETURN 0 ON FAILURE
            return 0;
        }
        // SET THE STRESS FOR THE PRIOR PHONEME
        stress[position - 1] = S;
    } // while
}

// change phonemelength depedendent on stress
// void Code41203()
void SetPhonemeLength() {
    unsigned char B;
    int position = 0;
    while (phonemeindex[position] != 255) {
        B = stress[position];
        // 41218: BMI 41229
        if ((B == 0) || ((B & 128) != 0)) {
            phonemeLength[position] =

pgm_read_byte(&phonemeLengthTable[phonemeindex[position]]);
        } else {
            phonemeLength[position] = pgm_read_byte(

&phonemeStressedLengthTable[phonemeindex[position]]);
        }
    }
}

```

Продолжение листинга А.25

```
        position++;
    }
}

void Code41240() {
    unsigned char pos = 0;

    while (phonemeindex[pos] != 255) {
        unsigned char index; // register AC
        R = pos;
        index = phonemeindex[pos];
        if ((pgm_read_byte(&sam_flags[index]) & 2) == 0) {
            pos++;
            continue;
        } else if ((pgm_read_byte(&sam_flags[index]) & 1) ==
0) {
            Insert(pos + 1, index + 1,
                pgm_read_byte(&phonemeLengthTable[index +
1]), stress[pos]);
            Insert(pos + 2, index + 2,
                pgm_read_byte(&phonemeLengthTable[index +
2]), stress[pos]);
            pos += 3;
            continue;
        }

        do {
            R++;
            B = phonemeindex[R];
        } while (B == 0);

        if (B != 255) {
            if ((pgm_read_byte(&sam_flags[B]) & 8) != 0) {
                pos++;
                continue;
            }
            if ((B == 36) || (B == 37)) {
                pos++;
                continue;
            } // '/H' '/X'
        }

        Insert(pos + 1, index + 1,
            pgm_read_byte(&phonemeLengthTable[index + 1]),
stress[pos]);
        Insert(pos + 2, index + 2,
```

Продолжение листинга A.25

```

                                pgm_read_byte(&phonemeLengthTable[index + 2]),
stress[pos]);
    pos += 3;
};
}

// Rewrites the phonemes using the following rules:
//
//      <DIPHTONG ENDING WITH WX> -> <DIPHTONG ENDING WITH
WX> WX
//      <DIPHTONG NOT ENDING WITH WX> -> <DIPHTONG NOT ENDING
WITH WX> YX
//      UL -> AX L
//      UM -> AX M
//      <STRESSED VOWEL> <SILENCE> <STRESSED VOWEL> ->
<STRESSED VOWEL>
//      <SILENCE> Q <VOWEL> T R -> CH R D R -> J R <VOWEL> R
-> <VOWEL> RX
//      <VOWEL> L -> <VOWEL> LX
//      G S -> G Z
//      K <VOWEL OR DIPHTONG NOT ENDING WITH IY> -> KX <VOWEL
OR DIPHTONG NOT
//      ENDING WITH IY> G <VOWEL OR DIPHTONG NOT ENDING WITH
IY> -> GX <VOWEL
//      OR DIPHTONG NOT ENDING WITH IY> S P -> S B S T -> S D
S K -> S G S KX
//      -> S GX <ALVEOLAR> UW -> <ALVEOLAR> UX CH -> CH
CH' (CH requires two
//      phonemes to represent it) J -> J J' (J requires two
phonemes to
//      represent it) <UNSTRESSED VOWEL> T <PAUSE> ->
<UNSTRESSED VOWEL> DX
//      <PAUSE> <UNSTRESSED VOWEL> D <PAUSE> -> <UNSTRESSED
VOWEL> DX <PAUSE>

// void Code41397()
void Parser2() {
    unsigned char pos = 0; // mem66;
    unsigned char mem58 = 0;

    // Loop through phonemes
    while (1) {
        // SET X TO THE CURRENT POSITION
        R = pos;
        // GET THE PHONEME AT THE CURRENT POSITION
        B = phonemeindex[pos];
    }
}

```

Продолжение листинга A.25

```

        // DEBUG: Print phoneme and index
        if (debug && B != 255)
            printf("%d: %c%c\n", R,
pgm_read_byte(&signInputTable1[B]),
                pgm_read_byte(&signInputTable2[B]));

        // Is phoneme pause?
        if (B == 0) {
            // Move ahead to the
            pos++;
            continue;
        }

        // If end of phonemes flag reached, exit routine
        if (B == 255) return;

        // Copy the current phoneme index to Y
        S = B;

        // RULE:
        //      <DIPHTONG ENDING WITH WX> -> <DIPHTONG ENDING
WITH WX> WX
        //      <DIPHTONG NOT ENDING WITH WX> -> <DIPHTONG
NOT ENDING WITH WX>
        //      YX
        // Example: OIL, COW

        // Check for DIPHTONG
        if ((pgm_read_byte(&sam_flags[B]) & 16) == 0) goto
pos41457;

        // Not a diphthong. Get the stress
        mem58 = stress[pos];

        // End in IY sound?
        B = pgm_read_byte(&sam_flags[S]) & 32;

        // If ends with IY, use YX, else use WX
        if (B == 0)
            B = 20;
        else
            B = 21; // 'WX' = 20 'YX' = 21
        // pos41443:
        // Insert at WX or YX following, copying the stress

```

Продолжение листинга A.25

```

        if (debug)
            if (B == 20)
                printf(
                    "RULE: insert WX following diphtong NOT
ending in IY "
                    "sound\n");
        if (debug)
            if (B == 21)
                printf(
                    "RULE: insert YX following diphtong ending
in IY sound\n");
        Insert(pos + 1, B, mem59, mem58);
        R = pos;
        // Jump to ???
        goto pos41749;

pos41457:

        // RULE:
        //         UL -> AX L
        // Example: MEDDLE

        // Get phoneme
        B = phonemeindex[R];
        // Skip this rule if phoneme is not UL
        if (B != 78) goto pos41487; // 'UL'
        B = 24; // 'L'                //change 'UL' to 'AX
L'

        if (debug) printf("RULE: UL -> AX L\n");

pos41466:
        // Get current phoneme stress
        mem58 = stress[R];

        // Change UL to AX
        phonemeindex[R] = 13; // 'AX'
        // Perform insert. Note code below may jump up here
with different
        // values
        Insert(R + 1, B, mem59, mem58);
        pos++;
        // Move to next phoneme
        continue;

pos41487:

```


Продолжение листинга A.25

```

// RULE:
//      UM -> AX M
// Example: ASTRONOMY

// Skip rule if phoneme != UM
if (B != 79) goto pos41495; // 'UM'
// Jump up to branch - replaces current phoneme with
AX and continues
B = 27; // 'M' //change 'UM' to 'AX M'
if (debug) printf("RULE: UM -> AX M\n");
goto pos41466;
pos41495:

// RULE:
//      UN -> AX N
// Example: FUNCTION

// Skip rule if phoneme != UN
if (B != 80) goto pos41503; // 'UN'

// Jump up to branch - replaces current phoneme with
AX and continues
B = 28; // 'N' //change UN to 'AX N'
if (debug) printf("RULE: UN -> AX N\n");
goto pos41466;
pos41503:

// RULE:
//      <STRESSED VOWEL> <SILENCE> <STRESSED VOWEL> -
> <STRESSED VOWEL>
//      <SILENCE> Q <VOWEL>
// EXAMPLE: AWAY EIGHT

S = B;
// VOWEL set?
B = pgm_read_byte(&sam_flags[B]) & 128;

// Skip if not a vowel
if (B != 0) {
    // Get the stress
    B = stress[R];

    // If stressed...
    if (B != 0) {
        // Get the following phoneme

```

Продолжение листинга A.25

```

        R++;
        B = phonemeindex[R];
        // If following phoneme is a pause

        if (B == 0) {
            // Get the phoneme following pause
            R++;
            S = phonemeindex[R];

            // Check for end of buffer flag
            if (S == 255) // buffer overflow
                // ??? Not sure about these flags
                B = 65 & 128;
            else
                // And VOWEL flag to current phoneme's
flags
                B = pgm_read_byte(&sam_flags[S]) &
128;

            // If following phonemes is not a pause
            if (B != 0) {
                // If the following phoneme is not
stressed
                B = stress[R];
                if (B != 0) {
                    // Insert a glottal stop and move
forward
                    if (debug)
                        printf(
                            "RULE: Insert glottal stop
between two "
                            "stressed vowels with
space between "
                            "them\n");
                    // 31 = 'Q'
                    Insert(R, 31, mem59, 0);
                    pos++;
                    continue;
                }
            }
        }

// RULES FOR PHONEMES BEFORE R
//      T R -> CH R

```

Продолжение листинга А.25

```
// Example: TRACK

// Get current position and phoneme
R = pos;
B = phonemeindex[pos];
if (B != 23) goto pos41611;  // 'R'

// Look at prior phoneme
R--;
B = phonemeindex[pos - 1];
// pos41567:
if (B == 69)  // 'T'
{
    // Change T to CH
    if (debug) printf("RULE: T R -> CH R\n");
    phonemeindex[pos - 1] = 42;
    goto pos41779;
}

// RULES FOR PHONEMES BEFORE R
//          D R -> J R
// Example: DRY

// Prior phonemes D?
if (B == 57)  // 'D'
{
    // Change D to J
    phonemeindex[pos - 1] = 44;
    if (debug) printf("RULE: D R -> J R\n");
    goto pos41788;
}

// RULES FOR PHONEMES BEFORE R
//          <VOWEL> R -> <VOWEL> RX
// Example: ART

// If vowel flag is set change R to RX
B = pgm_read_byte(&sam_flags[B]) & 128;
if (debug) printf("RULE: R -> RX\n");
if (B != 0) phonemeindex[pos] = 18;  // 'RX'

// continue to next phoneme
pos++;
continue;

pos41611:
```

Продолжение листинга A.25

```

// RULE:
//      <VOWEL> L -> <VOWEL> LX
// Example: ALL

// Is phoneme L?
if (B == 24)  // 'L'
{
    // If prior phoneme does not have VOWEL flag set,
move to next
    // phoneme
    if ((pgm_read_byte(&sam_flags[phonemeindex[pos -
1]]) & 128) == 0) {
        pos++;
        continue;
    }
    // Prior phoneme has VOWEL flag set, so change L
to LX and move to
    // next phoneme
    if (debug) printf("RULE: <VOWEL> L -> <VOWEL>
LX\n");
    phonemeindex[R] = 19;  // 'LX'
    pos++;
    continue;
}

// RULE:
//      G S -> G Z
//
// Can't get to fire -
//      1. The G -> GX rule intervenes
//      2. Reciter already replaces GS -> GZ

// Is current phoneme S?
if (B == 32)  // 'S'
{
    // If prior phoneme is not G, move to next phoneme
    if (phonemeindex[pos - 1] != 60) {
        pos++;
        continue;
    }
    // Replace S with Z and move on
    if (debug) printf("RULE: G S -> G Z\n");
    phonemeindex[pos] = 38;  // 'Z'
    pos++;
    continue;
}

```

Продолжение листинга A.25

```

    }

    // RULE:
    //           K <VOWEL OR DIPHTONG NOT ENDING WITH
IY> -> KX <VOWEL OR
    //           DIPHTONG NOT ENDING WITH IY>
    // Example: COW

    // Is current phoneme K?
    if (B == 72)  // 'K'
    {
        // Get next phoneme
        S = phonemeindex[pos + 1];
        // If at end, replace current phoneme with KX
        if (S == 255)
            phonemeindex[pos] =
                75;  // ML : prevents an index out of
bounds problem
        else {
            // VOWELS AND DIPHTONGS ENDING WITH IY SOUND
flag set?

            B = pgm_read_byte(&sam_flags[S]) & 32;
            if (debug)
                if (B == 0)
                    printf(
                        "RULE: K <VOWEL OR DIPHTONG NOT
ENDING WITH IY> -> "
                        "KX <VOWEL OR DIPHTONG NOT ENDING
WITH IY>\n");
            // Replace with KX
            if (B == 0) phonemeindex[pos] = 75;  // 'KX'
        }
    } else

        // RULE:
        //           G <VOWEL OR DIPHTONG NOT ENDING
WITH IY> -> GX <VOWEL
        //           OR DIPHTONG NOT ENDING WITH IY>
        // Example: GO

        // Is character a G?
        if (B == 60)  // 'G'
        {
            // Get the following character
            unsigned char index = phonemeindex[pos + 1];

```

Продолжение листинга A.25

```

        // At end of buffer?
        if (index == 255) // prevent buffer overflow
        {
            pos++;
            continue;
        } else
            // If diphtong ending with YX, move
continue processing next
            // phoneme
            if ((pgm_read_byte(&sam_flags[index]) &
32) != 0) {
                pos++;
                continue;
            }
        // replace G with GX and continue processing
next phoneme
        if (debug)
            printf(
                "RULE: G <VOWEL OR DIPHTONG NOT ENDING
WITH IY> -> GX "
                "<VOWEL OR DIPHTONG NOT ENDING WITH
IY>\n");
        phonemeindex[pos] = 63; // 'GX'
        pos++;
        continue;
    }

    // RULE:
    //      S P -> S B
    //      S T -> S D
    //      S K -> S G
    //      S KX -> S GX
    // Examples: SPY, STY, SKY, SCOWL

    S = phonemeindex[pos];
    // pos41719:
    // Replace with softer version?
    B = pgm_read_byte(&sam_flags[S]) & 1;
    if (B == 0) goto pos41749;
    B = phonemeindex[pos - 1];
    if (B != 32) // 'S'
    {
        B = S;
        goto pos41812;
    }
    // Replace with softer version

```

Продолжение листинга A.25

```

        if (debug)
            printf("RULE: S* %c%c -> S* %c%c\n",
                pgm_read_byte(&signInputTable1[S]),
                pgm_read_byte(&signInputTable2[S]),
                pgm_read_byte(&signInputTable1[S - 12]),
                pgm_read_byte(&signInputTable2[S - 12]));
        phonemeindex[pos] = S - 12;
        pos++;
        continue;

pos41749:

        // RULE:
        //      <ALVEOLAR> UW -> <ALVEOLAR> UX
        //
        // Example: NEW, DEW, SUE, ZOO, THOO, TOO

        //      UW -> UX

        B = phonemeindex[R];
        if (B == 53) // 'UW'
        {
            // ALVEOLAR flag set?
            S = phonemeindex[R - 1];
            B = pgm_read_byte(&sam_flags2[S]) & 4;
            // If not set, continue processing next phoneme
            if (B == 0) {
                pos++;
                continue;
            }
            if (debug) printf("RULE: <ALVEOLAR> UW ->
<ALVEOLAR> UX\n");
            phonemeindex[R] = 16;
            pos++;
            continue;
        }
pos41779:

        // RULE:
        //      CH -> CH CH' (CH requires two phonemes to
represent it)
        // Example: CHEW

        if (B == 42) // 'CH'
        {
            //      pos41783:

```

Продолжение листинга A.25

```

        if (debug) printf("CH -> CH CH+1\n");
        Insert(R + 1, B + 1, mem59, stress[R]);
        pos++;
        continue;
    }

pos41788:

    // RULE:
    //      J -> J J' (J requires two phonemes to
represent it)
    // Example: JAY

    if (B == 44) // 'J'
    {
        if (debug) printf("J -> J J+1\n");
        Insert(R + 1, B + 1, mem59, stress[R]);
        pos++;
        continue;
    }

    // Jump here to continue
pos41812:

    // RULE: Soften T following vowel
    // NOTE: This rule fails for cases such as "ODD"
    //      <UNSTRESSED VOWEL> T <PAUSE> -> <UNSTRESSED
VOWEL> DX <PAUSE>
    //      <UNSTRESSED VOWEL> D <PAUSE> -> <UNSTRESSED
VOWEL> DX <PAUSE>
    // Example: PARTY, TARDY

    // Past this point, only process if phoneme is T or D

    if (B != 69) // 'T'
        if (B != 57) {
            pos++;
            continue;
        } // 'D'
    // pos41825:

    // If prior phoneme is not a vowel, continue
processing phonemes
    if ((pgm_read_byte(&sam_flags[phonemeindex[R - 1]]) &
128) == 0) {
        pos++;
    }

```


Продолжение листинга A.25

```

        continue;
    }

    // Get next phoneme
    R++;
    B = phonemeindex[R];
    // pos41841
    // Is the next phoneme a pause?
    if (B != 0) {
        // If next phoneme is not a pause, continue
processing phonemes
        if ((pgm_read_byte(&sam_flags[B]) & 128) == 0) {
            pos++;
            continue;
        }
        // If next phoneme is stressed, continue
processing phonemes
        // FIXME: How does a pause get stressed?
        if (stress[R] != 0) {
            pos++;
            continue;
        }
        // pos41856:
        // Set phonemes to DX
        if (debug)
            printf(
                "RULE: Soften T or D following vowel or ER
and preceding a "
                "pause -> DX\n");
        phonemeindex[pos] = 30; // 'DX'
    } else {
        B = phonemeindex[R + 1];
        if (B == 255) // prevent buffer overflow
            B = 65 & 128;
        else
            // Is next phoneme a vowel or ER?
            B = pgm_read_byte(&sam_flags[B]) & 128;
        if (debug)
            if (B != 0)
                printf(
                    "RULE: Soften T or D following vowel
or ER and "
                    "preceding a pause -> DX\n");
        if (B != 0) phonemeindex[pos] = 30; // 'DX'
    }
}

```

Продолжение листинга A.25

```

        pos++;

    }    // while
}

// Applies various rules that adjust the lengths of phonemes
//
//          Lengthen <FRICATIVE> or <VOICED> between <VOWEL>
and <PUNCTUATION>
//          by 1.5 <VOWEL> <RX | LX> <CONSONANT> - decrease
<VOWEL> length by 1
//          <VOWEL> <UNVOICED PLOSIVE> - decrease vowel by
1/8th
//          <VOWEL> <UNVOICED CONSONANT> - increase vowel by
1/2 + 1
//          <NASAL> <STOP CONSONANT> - set nasal = 5, consonant
= 6
//          <VOICED STOP CONSONANT> {optional silence} <STOP
CONSONANT> - shorten
//          both to 1/2 + 1 <LIQUID CONSONANT> <DIPHTONG> -
decrease by 2

// void Code48619()
void AdjustLengths() {
    // LENGTHEN VOWELS PRECEDING PUNCTUATION
    //
    // Search for punctuation. If found, back up to the first
vowel, then
    // process all phonemes between there and up to (but not
including) the
    // punctuation. If any phoneme is found that is a either a
fricative or
    // voiced, the duration is increased by (length * 1.5) + 1

    // loop index
    R = 0;
    unsigned char index;

    // iterate through the phoneme list
    unsigned char loopIndex = 0;
    while (1) {
        // get a phoneme
        index = phonemeindex[R];

        // exit loop if end on buffer token
        if (index == 255) break;
    }
}

```

Продолжение листинга A.25

```

        // not punctuation?
        if ((pgm_read_byte(&sam_flags2[index]) & 1) == 0) {
            // skip
            R++;
            continue;
        }

        // hold index
        loopIndex = R;

        // Loop backwards from this point
pos48644:

        // back up one phoneme
        R--;

        // stop once the beginning is reached
        if (R == 0) break;

        // get the preceding phoneme
        index = phonemeindex[R];

        if (index != 255) // inserted to prevent access
overrun
            if ((pgm_read_byte(&sam_flags[index]) & 128) == 0)
                goto pos48644; // if not a vowel, continue
looping

        // pos48657:
        do {
            // test for vowel
            index = phonemeindex[R];

            if (index != 255) // inserted to prevent access
overrun
                // test for fricative/unvoiced or not voiced
                if (((pgm_read_byte(&sam_flags2[index]) & 32)
== 0) ||
                    ((pgm_read_byte(&sam_flags[index]) & 4) !=
0)) // nochmal überprüfen
                {
                    // A = pgm_read_byte(&sam_flags[Y]) & 4;
                    // if(A == 0) goto pos48688;

                    // get the phoneme length

```

Продолжение листинга A.25

```

        B = phonemeLength[R];

        // change phoneme length to (length * 1.5)
+ 1
        B = (B >> 1) + B + 1;
        if (debug)
            printf(
                "RULE: Lengthen <FRICATIVE> or
<VOICED> between "
                "<VOWEL> and <PUNCTUATION> by
1.5\n");
        if (debug) printf("PRE\n");
        if (debug)
            printf("phoneme %d (%c%c) length
%d\n", R,
pgm_read_byte(&signInputTable1[phonemeindex[R]]),
pgm_read_byte(&signInputTable2[phonemeindex[R]]),
                phonemeLength[R]);

        phonemeLength[R] = B;

        if (debug) printf("POST\n");
        if (debug)
            printf("phoneme %d (%c%c) length
%d\n", R,
pgm_read_byte(&signInputTable1[phonemeindex[R]]),
pgm_read_byte(&signInputTable2[phonemeindex[R]]),
                phonemeLength[R]);
    }
    // keep moving forward
    R++;
} while (R != loopIndex);
// if (X != loopIndex) goto pos48657;
R++;
} // while

    // Similar to the above routine, but shorten vowels under
    some circumstances

    // Loop throught all phonemes
    loopIndex = 0;
    // pos48697

```

Продолжение листинга A.25

```

while (1) {
    // get a phoneme
    R = loopIndex;
    index = phonemeindex[R];

    // exit routine at end token
    if (index == 255) return;

    // vowel?
    B = pgm_read_byte(&sam_flags[index]) & 128;
    if (B != 0) {
        // get next phoneme
        R++;
        index = phonemeindex[R];

        // get sam_flags
        if (index == 255)
            mem56 = 65; // use if end marker
        else
            mem56 = pgm_read_byte(&sam_flags[index]);

        // not a consonant
        if ((pgm_read_byte(&sam_flags[index]) & 64) == 0)
        {
            // RX or LX?
            if ((index == 18) || (index == 19)) // 'RX' &
            'LX'
            {
                // get the next phoneme
                R++;
                index = phonemeindex[R];

                // next phoneme a consonant?
                if ((pgm_read_byte(&sam_flags[index]) &
                64) != 0) {
                    // RULE: <VOWEL> RX | LX <CONSONANT>

                    if (debug)
                        printf(
                            "RULE: <VOWEL> <RX | LX>
                <CONSONANT> - "
                            "decrease length by 1\n");
                    if (debug) printf("PRE\n");
                    if (debug)
                        printf(

```

Продолжение листинга A.25

```

                                "phoneme %d (%c%c) length
%d\n", loopIndex,
                                pgm_read_byte(
&signInputTable1[phonemeindex[loopIndex]]),
                                pgm_read_byte(
&signInputTable2[phonemeindex[loopIndex]]),
                                phonemeLength[loopIndex]);

                                // decrease length of vowel by 1 frame
                                phonemeLength[loopIndex]--;

                                if (debug) printf("POST\n");
                                if (debug)
                                    printf(
                                        "phoneme %d (%c%c) length
%d\n", loopIndex,
                                        pgm_read_byte(
&signInputTable1[phonemeindex[loopIndex]]),
                                        pgm_read_byte(
&signInputTable2[phonemeindex[loopIndex]]),
                                        phonemeLength[loopIndex]);
                                }
                                // move ahead
                                loopIndex++;
                                continue;
                            }
                            // move ahead
                            loopIndex++;
                            continue;
                        }

                        // Got here if not <VOWEL>

                        // not voiced
                        if ((mem56 & 4) == 0) {
                            // Unvoiced
                            // *, .*, ?*, ,*, -*, DX, S*, SH, F*, TH, /
H, /X, CH, P*, T*,
                            // K*, KX

                            // not an unvoiced plosive?
                            if ((mem56 & 1) == 0) {

```

Продолжение листинга A.25

```

        // move ahead
        loopIndex++;
        continue;
    }

    // P*, T*, K*, KX

    // RULE: <VOWEL> <UNVOICED PLOSIVE>
    // <VOWEL> <P*, T*, K*, KX>

    // move back
    R--;

    if (debug)
        printf(
            "RULE: <VOWEL> <UNVOICED PLOSIVE> -
decrease vowel by "
            "1/8th\n");
    if (debug) printf("PRE\n");
    if (debug)
        printf("phoneme %d (%c%c) length %d\n", R,
pgm_read_byte(&signInputTable1[phonemeindex[R]]),
pgm_read_byte(&signInputTable2[phonemeindex[R]]),
                phonemeLength[R]);

    // decrease length by 1/8th
    mem56 = phonemeLength[R] >> 3;
    phonemeLength[R] -= mem56;

    if (debug) printf("POST\n");
    if (debug)
        printf("phoneme %d (%c%c) length %d\n", R,
pgm_read_byte(&signInputTable1[phonemeindex[R]]),
pgm_read_byte(&signInputTable2[phonemeindex[R]]),
                phonemeLength[R]);

    // move ahead
    loopIndex++;
    continue;
}

// RULE: <VOWEL> <VOICED CONSONANT>

```

Продолжение листинга A.25

```

        // <VOWEL> <WH, R*, L*, W*, Y*, M*, N*, NX, DX,
Q*, Z*, ZH, V*, DH,
        // J*, B*, D*, G*, GX>

        if (debug)
            printf(
                "RULE: <VOWEL> <VOICED CONSONANT> -
increase vowel by 1/2 "
                "+ 1\n");
        if (debug) printf("PRE\n");
        if (debug)
            printf("phoneme %d (%c%c) length %d\n", R - 1,
pgm_read_byte(&signInputTable1[phonemeindex[R - 1]]),
pgm_read_byte(&signInputTable2[phonemeindex[R - 1]]),
                phonemeLength[R - 1]);

        // decrease length
        B = phonemeLength[R - 1];
        phonemeLength[R - 1] = (B >> 2) + B + 1; // 5/4*A
+ 1

        if (debug) printf("POST\n");
        if (debug)
            printf("phoneme %d (%c%c) length %d\n", R - 1,
pgm_read_byte(&signInputTable1[phonemeindex[R - 1]]),
pgm_read_byte(&signInputTable2[phonemeindex[R - 1]]),
                phonemeLength[R - 1]);

        // move ahead
        loopIndex++;
        continue;
    }

    // WH, R*, L*, W*, Y*, M*, N*, NX, Q*, Z*, ZH, V*, DH,
J*, B*, D*, G*,
    // GX

    // pos48821:

    // RULE: <NASAL> <STOP CONSONANT>
    //         Set punctuation length to 6
    //         Set stop consonant length to 5

```



```

// nasal?
if ((pgm_read_byte(&sam_flags2[index]) & 8) != 0) {
    // M*, N*, NX,

    // get the next phoneme
    R++;
    index = phonemeindex[R];

    // end of buffer?
    if (index == 255)
        B = 65 & 2; // prevent buffer overflow
    else
        B = pgm_read_byte(&sam_flags[index]) &
            2; // check for stop consonant

    // is next phoneme a stop consonant?
    if (B != 0)

        // B*, D*, G*, GX, P*, T*, K*, KX

        {
            if (debug)
                printf(
                    "RULE: <NASAL> <STOP CONSONANT> - set
nasal = 5, "
                    "consonant = 6\n");
            if (debug) printf("POST\n");
            if (debug)
                printf("phoneme %d (%c%c) length %d\n", R,
pgm_read_byte(&signInputTable1[phonemeindex[R]]),
pgm_read_byte(&signInputTable2[phonemeindex[R]]),
                    phonemeLength[R]);
            if (debug)
                printf("phoneme %d (%c%c) length %d\n", R
- 1,
pgm_read_byte(&signInputTable1[phonemeindex[R - 1]]),
pgm_read_byte(&signInputTable2[phonemeindex[R - 1]]),
                    phonemeLength[R - 1]);

            // set stop consonant length to 6
            phonemeLength[R] = 6;

```

Продолжение листинга A.25

```

        // set nasal length to 5
        phonemeLength[R - 1] = 5;

        if (debug) printf("POST\n");
        if (debug)
            printf("phoneme %d (%c%c) length %d\n", R,
pgm_read_byte(&signInputTable1[phonemeindex[R]]),
pgm_read_byte(&signInputTable2[phonemeindex[R]]),
                phonemeLength[R]);
        if (debug)
            printf("phoneme %d (%c%c) length %d\n", R
- 1,
pgm_read_byte(&signInputTable1[phonemeindex[R - 1]]),
pgm_read_byte(&signInputTable2[phonemeindex[R - 1]]),
                phonemeLength[R - 1]);
    }
    // move to next phoneme
    loopIndex++;
    continue;
}

// WH, R*, L*, W*, Y*, Q*, Z*, ZH, V*, DH, J*, B*, D*,
G*, GX

// RULE: <VOICED STOP CONSONANT> {optional silence}
<STOP CONSONANT>
//          Shorten both to (length/2 + 1)

// (voiced) stop consonant?
if ((pgm_read_byte(&sam_flags[index]) & 2) != 0) {
    // B*, D*, G*, GX

    // move past silence
    do {
        // move ahead
        R++;
        index = phonemeindex[R];
    } while (index == 0);

    // check for end of buffer
    if (index == 255) // buffer overflow

```

Продолжение листинга A.25

```

        {
            // ignore, overflow code
            if ((65 & 2) == 0) {
                loopIndex++;
                continue;
            }
        } else if ((pgm_read_byte(&sam_flags[index]) & 2)
== 0) {
            // if another stop consonant, move ahead
            loopIndex++;
            continue;
        }

        // RULE: <UNVOICED STOP CONSONANT> {optional
silence} <STOP
        // CONSONANT>
        if (debug)
            printf(
                "RULE: <UNVOICED STOP CONSONANT> {optional
silence} <STOP "
                "CONSONANT> - shorten both to 1/2 + 1\n");
        if (debug) printf("PRE\n");
        if (debug)
            printf("phoneme %d (%c%c) length %d\n", R,

pgm_read_byte(&signInputTable1[phonemeindex[R]]),

pgm_read_byte(&signInputTable2[phonemeindex[R]]),
                phonemeLength[R]);
        if (debug)
            printf("phoneme %d (%c%c) length %d\n", R - 1,

pgm_read_byte(&signInputTable1[phonemeindex[R - 1]]),

pgm_read_byte(&signInputTable2[phonemeindex[R - 1]]),
                phonemeLength[R - 1]);
        // X gets overwritten, so hold prior X value for
debug statement
        int debugX = R;
        // shorten the prior phoneme length to (length/2 +
1)

        phonemeLength[R] = (phonemeLength[R] >> 1) + 1;
        R = loopIndex;

        // also shorten this phoneme length to (length/2
+1)

```

Продолжение листинга А.25

```

        phonemeLength[loopIndex] =
        (phonemeLength[loopIndex] >> 1) + 1;

        if (debug) printf("POST\n");
        if (debug)
            printf("phoneme %d (%c%c) length %d\n",
debugX,

pgm_read_byte(&signInputTable1[phonemeindex[debugX]]),

pgm_read_byte(&signInputTable2[phonemeindex[debugX]]),
            phonemeLength[debugX]);
        if (debug)
            printf(
                "phoneme %d (%c%c) length %d\n", debugX -
1,

pgm_read_byte(&signInputTable1[phonemeindex[debugX - 1]]),

pgm_read_byte(&signInputTable2[phonemeindex[debugX - 1]]),
            phonemeLength[debugX - 1]);

        // move ahead
        loopIndex++;
        continue;
    }

    // WH, R*, L*, W*, Y*, Q*, Z*, ZH, V*, DH, J*, **,

    // RULE: <VOICED NON-VOWEL> <DIPHTONG>
    //         Decrease <DIPHTONG> by 2

    // liquic consonant?
    if ((pgm_read_byte(&sam_flags2[index]) & 16) != 0) {
        // R*, L*, W*, Y*

        // get the prior phoneme
        index = phonemeindex[R - 1];

        // prior phoneme a stop consonant>
        if ((pgm_read_byte(&sam_flags[index]) & 2) != 0) {
            // Rule: <LIQUID CONSONANT> <DIPHTONG>

            if (debug)
                printf(
                    "RULE: <LIQUID CONSONANT> <DIPHTONG> -

```

Продолжение листинга A.25

```

decrease by "
                "2\n");
        if (debug) printf("PRE\n");
        if (debug)
            printf("phoneme %d (%c%c) length %d\n", R,
pgm_read_byte(&signInputTable1[phonemeindex[R]]),
pgm_read_byte(&signInputTable2[phonemeindex[R]]),
                phonemeLength[R]);

                // decrease the phoneme length by 2 frames (20
ms)
                phonemeLength[R] -= 2;

                if (debug) printf("POST\n");
                if (debug)
                    printf("phoneme %d (%c%c) length %d\n", R,
pgm_read_byte(&signInputTable1[phonemeindex[R]]),
pgm_read_byte(&signInputTable2[phonemeindex[R]]),
                    phonemeLength[R]);
            }
        }

        // move to next phoneme
        loopIndex++;
        continue;
    }
    //          goto pos48701;
}

//
-----
// ML : Code47503 is division with remainder, and mem50 gets
the sign
void Code47503(unsigned char mem52) {
    S = 0;
    if ((mem53 & 128) != 0) {
        mem53 = -mem53;
        S = 128;
    }
    mem50 = S;
    B = 0;
    for (R = 8; R > 0; R--) {

```

Продолжение листинга A.25

```
        int temp = mem53;
        mem53 = mem53 << 1;
        B = B << 1;
        if (temp >= 128) B++;
        if (B >= mem52) {
            B = B - mem52;
            mem53++;
        }
    }

    mem51 = B;
    if ((mem50 & 128) != 0) mem53 = -mem53;
}
```

Листинг A.26 — Содержимое файла src/SamTabs.c

```
#include "SamTabs.h"

const unsigned char stressInputTable[] PROGMEM =
{
    '*', '1', '2', '3', '4', '5', '6', '7', '8'
};

//tab40682
const unsigned char signInputTable1[] PROGMEM = {
    ' ', '.', '?', ',', '-', 'I', 'I', 'E',
    'A', 'A', 'A', 'A', 'U', 'A', 'I', 'E',
    'U', 'O', 'R', 'L', 'W', 'Y', 'W', 'R',
    'L', 'W', 'Y', 'M', 'N', 'N', 'D', 'Q',
    'S', 'S', 'F', 'T', '/', '/', 'Z', 'Z',
    'V', 'D', 'C', '*', 'J', '*', '*', '*',
    'E', 'A', 'O', 'A', 'O', 'U', 'B', '*',
    '*', 'D', '*', '*', 'G', '*', '*', 'G',
    '*', '*', 'P', '*', '*', 'T', '*', '*',
    'K', '*', '*', 'K', '*', '*', 'U', 'U',
    'U'
};

//tab40763
const unsigned char signInputTable2[] PROGMEM =
{
    '*', '*', '*', '*', '*', 'Y', 'H', 'H',
    'E', 'A', 'H', 'O', 'H', 'X', 'X', 'R',
    'X', 'H', 'X', 'X', 'X', 'X', 'H', '*',
    '*', '*', '*', '*', '*', 'X', 'X', '*',
    '*', 'H', '*', 'H', 'H', 'X', '*', 'H',
}
```

Продолжение листинга А.26

```

        '*', 'H', 'H', '*', '*', '*', '*', '*',
        'Y', 'Y', 'Y', 'W', 'W', 'W', '*', '*',
        '*', '*', '*', '*', '*', '*', '*', 'X',
        '*', '*', '*', '*', '*', '*', '*', '*',
        '*', '*', '*', 'X', '*', '*', 'L', 'M',
        'N'
};

//loc_9F8C
const unsigned char sam_flags[] PROGMEM ={
    0x00 , 0x00 , 0x00 , 0x00 , 0x00 , 0xA4 , 0xA4 , 0xA4 ,
    0xA4 , 0xA4 , 0xA4 , 0x84 , 0x84 , 0xA4 , 0xA4 , 0x84 ,
    0x84 , 0x84 , 0x84 , 0x84 , 0x84 , 0x84 , 0x44 , 0x44 ,
    0x44 , 0x44 , 0x44 , 0x4C , 0x4C , 0x4C , 0x48 , 0x4C ,
    0x40 , 0x40 , 0x40 , 0x40 , 0x40 , 0x40 , 0x44 , 0x44 ,
    0x44 , 0x44 , 0x48 , 0x40 , 0x4C , 0x44 , 0x00 , 0x00 ,
    0xB4 , 0xB4 , 0xB4 , 0x94 , 0x94 , 0x94 , 0x4E , 0x4E ,
    0x4E , 0x4E , 0x4E , 0x4E , 0x4E , 0x4E , 0x4E , 0x4E ,
    0x4E , 0x4E , 0x4B , 0x4B , 0x4B , 0x4B , 0x4B , 0x4B ,
    0x4B , 0x4B , 0x4B , 0x4B , 0x4B , 0x4B , 0x80 , 0xC1 ,
    0xC1
};

//??? flags overlap flags2
//loc_9FDA
const unsigned char sam_flags2[] PROGMEM =
{
    0x80 , 0xC1 , 0xC1 , 0xC1 , 0xC1 , 0x00 , 0x00 , 0x00 ,
    0x00 , 0x00 , 0x00 , 0x00 , 0x00 , 0x00 , 0x00 , 0x00 ,
    0x00 , 0x00 , 0x00 , 0x00 , 0x00 , 0x00 , 0x00 , 0x10 ,
    0x10 , 0x10 , 0x10 , 0x08 , 0x0C , 0x08 , 0x04 , 0x40 ,
    0x24 , 0x20 , 0x20 , 0x24 , 0x00 , 0x00 , 0x24 , 0x20 ,
    0x20 , 0x24 , 0x20 , 0x20 , 0x00 , 0x20 , 0x00 , 0x00 ,
    0x00 , 0x00 , 0x00 , 0x00 , 0x00 , 0x00 , 0x00 , 0x00 ,
    0x00 , 0x04 , 0x04 , 0x04 , 0x00 , 0x00 , 0x00 , 0x00 ,
    0x00 , 0x00 , 0x00 , 0x00 , 0x00 , 0x04 , 0x04 , 0x04 ,
    0x00 , 0x00 , 0x00 , 0x00 , 0x00 , 0x00 , 0x00
};

//tab45616???
const unsigned char phonemeStressedLengthTable[] PROGMEM =
{

```

Продолжение листинга А.26

```

    0x00 , 0x12 , 0x12 , 0x12 , 8 , 0xB , 9 , 0xB ,
    0xE , 0xF , 0xB , 0x10 , 0xC , 6 , 6 , 0xE ,
    0xC , 0xE , 0xC , 0xB , 8 , 8 , 0xB , 0xA ,
    9 , 8 , 8 , 8 , 8 , 8 , 3 , 5 ,
    2 , 2 , 2 , 2 , 2 , 2 , 6 , 6 ,
    8 , 6 , 6 , 2 , 9 , 4 , 2 , 1 ,
    0xE , 0xF , 0xF , 0xF , 0xE , 0xE , 8 , 2 ,
    2 , 7 , 2 , 1 , 7 , 2 , 2 , 7 ,
    2 , 2 , 8 , 2 , 2 , 6 , 2 , 2 ,
    7 , 2 , 4 , 7 , 1 , 4 , 5 , 5
};

//tab45536???
const unsigned char phonemeLengthTable[] PROGMEM =
{
    0 , 0x12 , 0x12 , 0x12 , 8 , 8 , 8 , 8 ,
    8 , 0xB , 6 , 0xC , 0xA , 5 , 5 , 0xB ,
    0xA , 0xA , 0xA , 9 , 8 , 7 , 9 , 7 ,
    6 , 8 , 6 , 7 , 7 , 7 , 2 , 5 ,
    2 , 2 , 2 , 2 , 2 , 2 , 6 , 6 ,
    7 , 6 , 6 , 2 , 8 , 3 , 1 , 0x1E ,
    0xD , 0xC , 0xC , 0xC , 0xE , 9 , 6 , 1 ,
    2 , 5 , 1 , 1 , 6 , 1 , 2 , 6 ,
    1 , 2 , 8 , 2 , 2 , 4 , 2 , 2 ,
    6 , 1 , 4 , 6 , 1 , 4 , 0xC7 , 0xFF
};

/*

Ind   | phoneme | flags   |
-----|-----|-----|
0     | *       | 00000000 |
1     | .*      | 00000000 |
2     | ?*      | 00000000 |
3     | ,*      | 00000000 |
4     | -*      | 00000000 |

VOWELS
5     | IY      | 10100100 |
6     | IH      | 10100100 |
7     | EH      | 10100100 |
8     | AE      | 10100100 |
9     | AA      | 10100100 |
10    | AH      | 10100100 |

```


Продолжение листинга A.26

11		AO		10000100	
17		OH		10000100	
12		UH		10000100	
16		UX		10000100	
15		ER		10000100	
13		AX		10100100	
14		IX		10100100	

DIPHTONGS

48		EY		10110100	
49		AY		10110100	
50		OY		10110100	
51		AW		10010100	
52		OW		10010100	
53		UW		10010100	

21		YX		10000100	
20		WX		10000100	
18		RX		10000100	
19		LX		10000100	
37		/X		01000000	
30		DX		01001000	

22		WH		01000100	
----	--	----	--	----------	--

VOICED CONSONANTS

23		R*		01000100	
24		L*		01000100	
25		W*		01000100	
26		Y*		01000100	
27		M*		01001100	
28		N*		01001100	
29		NX		01001100	
54		B*		01001110	
57		D*		01001110	
60		G*		01001110	
44		J*		01001100	
38		Z*		01000100	
39		ZH		01000100	
40		V*		01000100	
41		DH		01000100	

unvoiced CONSONANTS

Продолжение листинга А.26

32		S*		01000000	
33		SH		01000000	
34		F*		01000000	
35		TH		01000000	
66		P*		01001011	
69		T*		01001011	
72		K*		01001011	
42		CH		01001000	
36		/H		01000000	

43		**		01000000	
45		**		01000100	
46		**		00000000	
47		**		00000000	

55		**		01001110	
56		**		01001110	
58		**		01001110	
59		**		01001110	
61		**		01001110	
62		**		01001110	
63		GX		01001110	
64		**		01001110	
65		**		01001110	
67		**		01001011	
68		**		01001011	
70		**		01001011	
71		**		01001011	
73		**		01001011	
74		**		01001011	
75		KX		01001011	
76		**		01001011	
77		**		01001011	

SPECIAL

78		UL		10000000	
79		UM		11000001	
80		UN		11000001	
31		Q*		01001100	

* /

Листинг А.27 — Содержимое файла src/uart.c

```
#include "uart.h"

#include <avr/interrupt.h>
#include <avr/io.h>

#include "buffer.h"

static volatile buffer_t INPUT_BUFFER NOINITMEM;

void init_uart() {
    UBRR1H = (uint8_t)(UBRR_VALUE >> 8);
    UBRR1L = (uint8_t)(UBRR_VALUE);

    UCSR1A = 0; // U2X1 = 0
    (обычный режим)
    UCSR1C = (1 << UCSZ11) | (1 << UCSZ10); // 8 бит, без
    чётности, 1 стоп
    UCSR1B = (1 << RXEN1) | (1 << TXEN1) |
              (1 << RXCIE1); // включить RX и TX, прерывание
    по чтению
    INPUT_BUFFER = make_buffer();
}

int putchar_uart(char c, FILE* stream) {
    while (!(UCSR1A & 1 << UDRE1));
    if (c == '\n') {
        putchar_uart('\r', NULL);
    }
    UDR1 = c;
    return 0;
}

int getchar_uart(FILE* stream) {
    // while (!(UCSR1A & 1 << RXC1));
    // return UDR1;
    return buffer_read(&INPUT_BUFFER);
}

ISR(USART1_RX_vect) {
    // PORTB ^= (1 << PB0);
    buffer_write(&INPUT_BUFFER, UDR1);
}

FILE uart_stdout = FDEV_SETUP_STREAM(putchar_uart, NULL,
    _FDEV_SETUP_WRITE);
```

Продолжение листинга A.27

```
FILE uart_stdin = FDEV_SETUP_STREAM(NULL, getchar_uart,  
_FDEV_SETUP_READ);
```

ПРИЛОЖЕНИЕ Б
СПЕЦИФИКАЦИЯ РАДИОЭЛЕМЕНТОВ СХЕМЫ
Листов 1

Поз. обознач.		Наименование			Кол.	Примечание		
		<u>Динамики</u>						
BA1		FBS1850			1			
		<u>Конденсаторы</u>						
C1-C2		K50-68-25B-10 мкФ			2			
C3		KM5B-H90-50 нФ			1			
C4-C12		KM5B-H90-0,1 мкФ			9			
C13-14		KM5B-H90-20 нФ			2			
		<u>Микросхемы</u>						
DD1		ATMega128A			1			
DD2		SN74HC573A			1			
DD3		AS6C1008			1			
DD4		РАМ8403			1			
DD5		CH340C			1			
		<u>Резисторы</u>						
R1-R2		МЛТ 0,125 - 10 кОм			2			
		<u>Соединители контактные</u>						
XP1		DS-1023-2x3			1			
XP2		DS-1023-1x4			1			
		<u>Резонаторы</u>						
Z1		HC-49U 8 МГц			1			