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. Зальгин (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	02.09.2025	И.Б. Трамов
	(Подпись, дата)	(И.О.Фамилия)
Студент	02.09.2025	В.К. Залыгин
	(Подпись, дата)	(И.О.Фамилия)

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

РЕФЕРАТ

РПЗ 38 с., 16 рис., 1 табл., 0 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.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

МК – микроконтроллер

МК система – микропроцессорная система

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

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

UART –

тулчейн –

ПК

callback-функция

ОЗУ

SRAM

libc

malloc

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, что делает их удобной основой для реализации курсовых и дипломных проектов.

Целевым микроконтроллером в данной работе является ATMega128A, относящийся к семейству высокопроизводительных 8-битных AVR. Он обладает расширенным объемом программной памяти (128 кБайт Flash), внутренней ОЗУ 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. Наличие двух независимых USART, развитой системы прерываний и широкого диапазона тактовых частот обеспечивает гибкость при организации обмена с внешними устройствами и генерации звука.

В основе проекта лежит S.A.M. (The Software Automatic Mouth – TODO ссылка) – программа синтеза речи (TTS) для английского языка, созданная компанией Don’t ask software в 1982 году для компьютеров Commodore C64, Atari 8-bit, а также Apple II. S.A.M. является одной из первых коммерческих программ синтеза речи, выпущенных на рынок. Программа включает в себя преобразователь текста в фонемы, который называется reciter, функцию преобразования фонем в речь для окончательного вывода (в формате .wav

файла), а также набор функций для отладки. Изначально написанная на ассемблере под целевые компьютеры, в 2015 году благодаря стараниям Stefan Macke программа была дизассемблирована в код на языке С и выложена в открытый доступ (лицензия программы не была указана в связи с неуспешными попытками связаться с создателями, компания Don’t ask software ныне не существует). Программа обладает поразительно малыми размерами – «боевая» 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 для самых маленьких МК систем. В рамках данной курсовой работы ставится задача портирования S.A.M. 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 протокола XMEM;

– наличие нескольких таймеров, поддерживающих ШИП;

– наличие интерфейсов передачи данных, таких как UART;

– высокая производительность МК на частоте 8 МГц;

– простота и открытость архитектуры МК семейства AVR для настройки.

По завершении проектирования была выполнена проверка работоспособности схемы и программного обеспечения на макете.

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 UART.

Микроконтроллер должен уметь работать с 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 объёмом 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 осуществляется при помощи вспомогательного регистра-защёлки, который необходим для хранения младших бит адреса до окончания цикла чтения или записи.

Для создания качественного звучания сигнал ШИМ, генерируемый МК, проходит 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 UART-USB.

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.

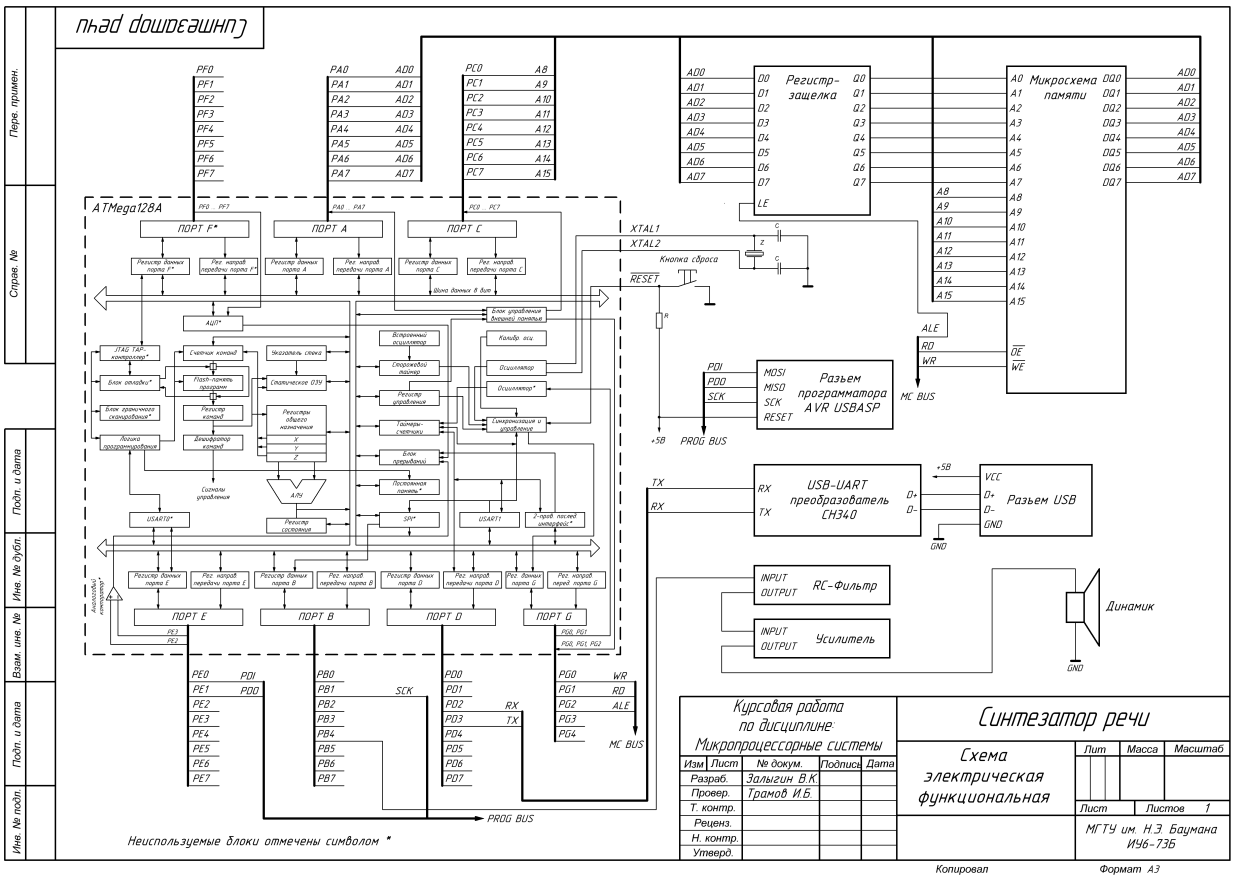

Рисунок 2 — Функциональная схема

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.

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 заданной 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.

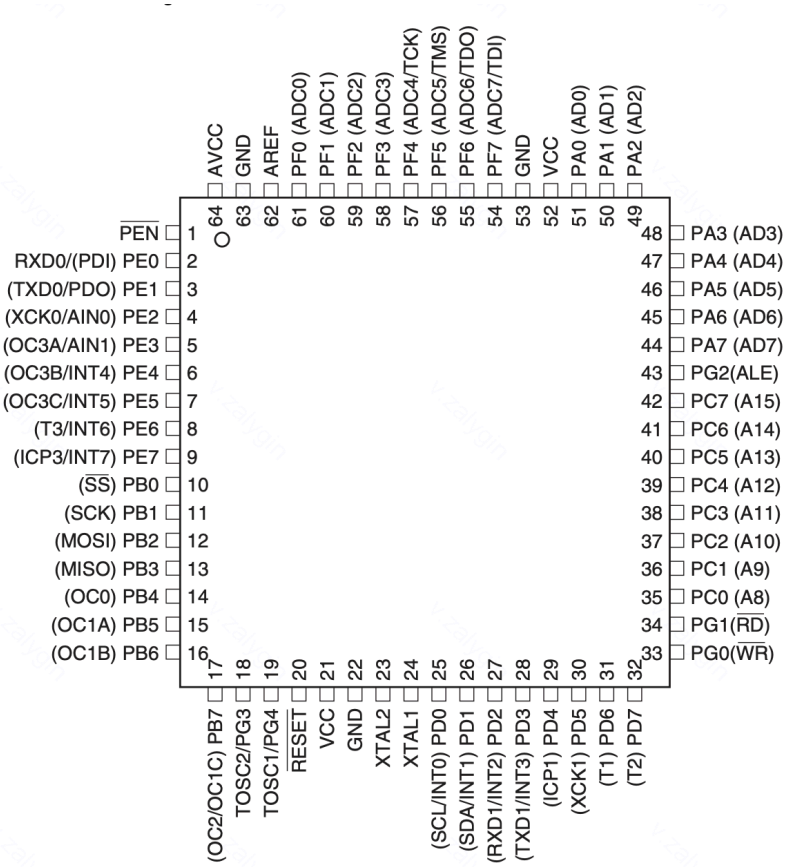

Рисунок 3 — Распиновка ATMega128A в корпусе TQFP64

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.

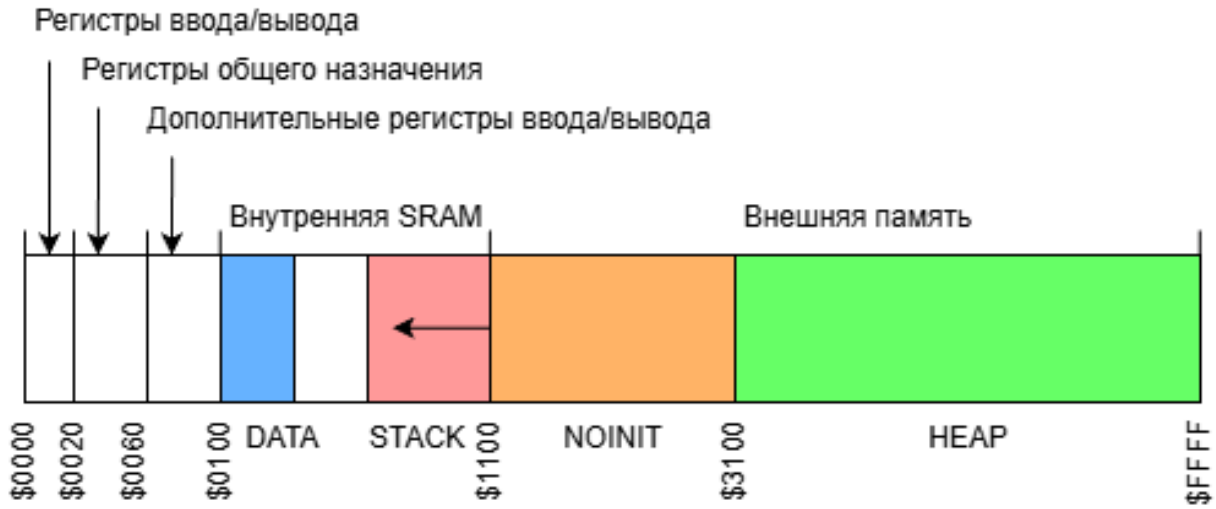

Рисунок 4 — Адресное пространство данных

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 данной 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 логиу обработки данных.

1.4 Описание принципиальной электрической схемы

Принципиальная схема устройства представлена на рисунке 5.

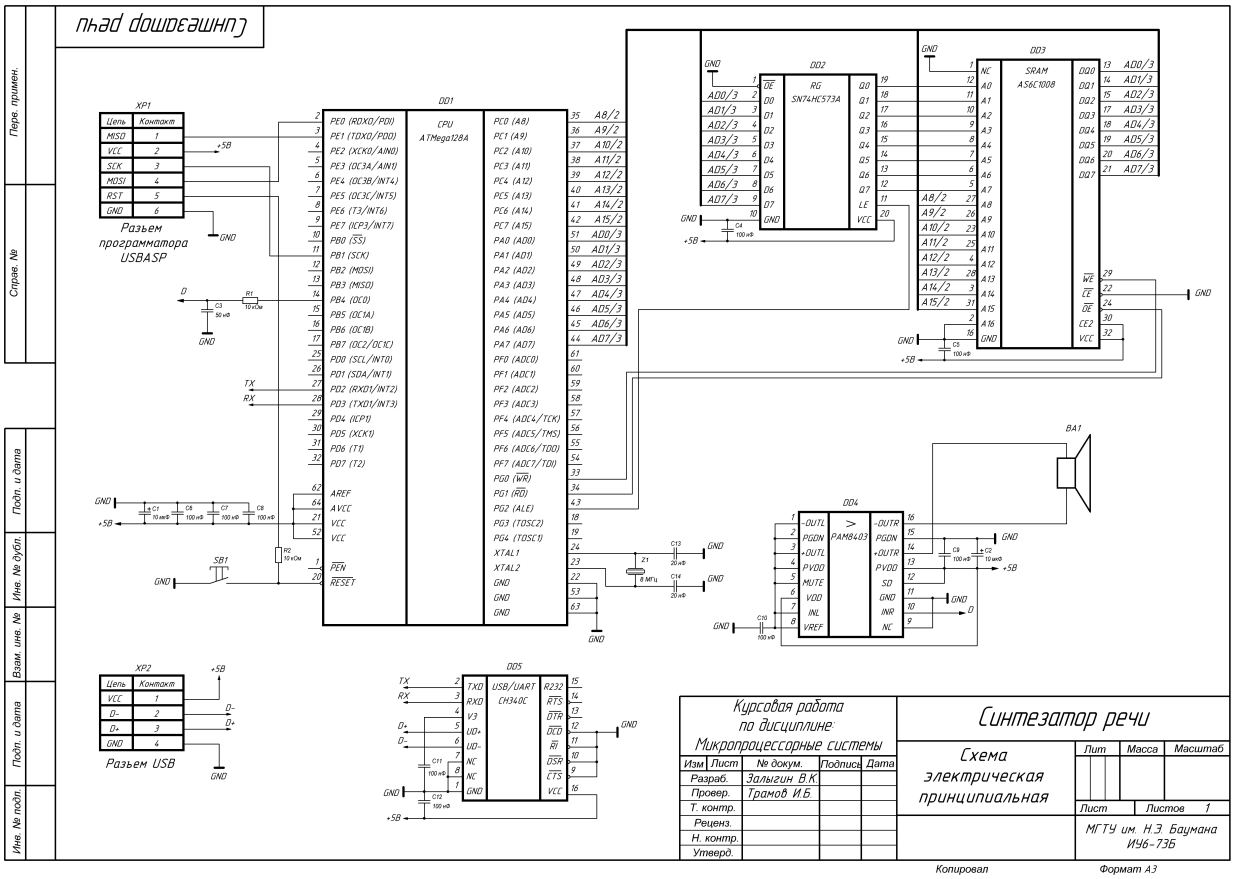

Рисунок 5 — Принципиальная схема

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 общую землю; при нажатии кнопки микроконтроллер переходит в состояние аппаратного сброса.

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.

Снизу показан интерфейс USB-UART на микросхеме CH340C, связанный с разъёмом USB и выводами USART1 микроконтроллера. В нижней правой части схемы расположен «аудиотракт»: RC-фильтр, усилитель мощности PAM8403 и разъём динамика XP3.

1.4.1 Подключения МК и программатора

Для удобства разработки и отладки микроконтроллер поддерживает внутрисхемное программирование по SPI. На схеме показан разъём программатора USBasp XP1. Его выводы MISO, MOSI и SCK подключены 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. 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.

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; 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.

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

– SRE – включает/выключает интерфейс внешней памяти XMEM;

– 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.

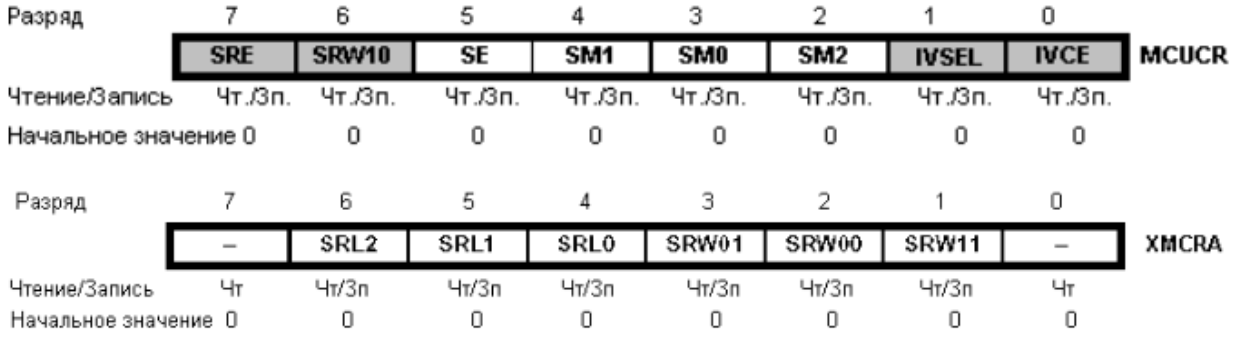

Рисунок 6 — Регистры MCUCR и XMCRA

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

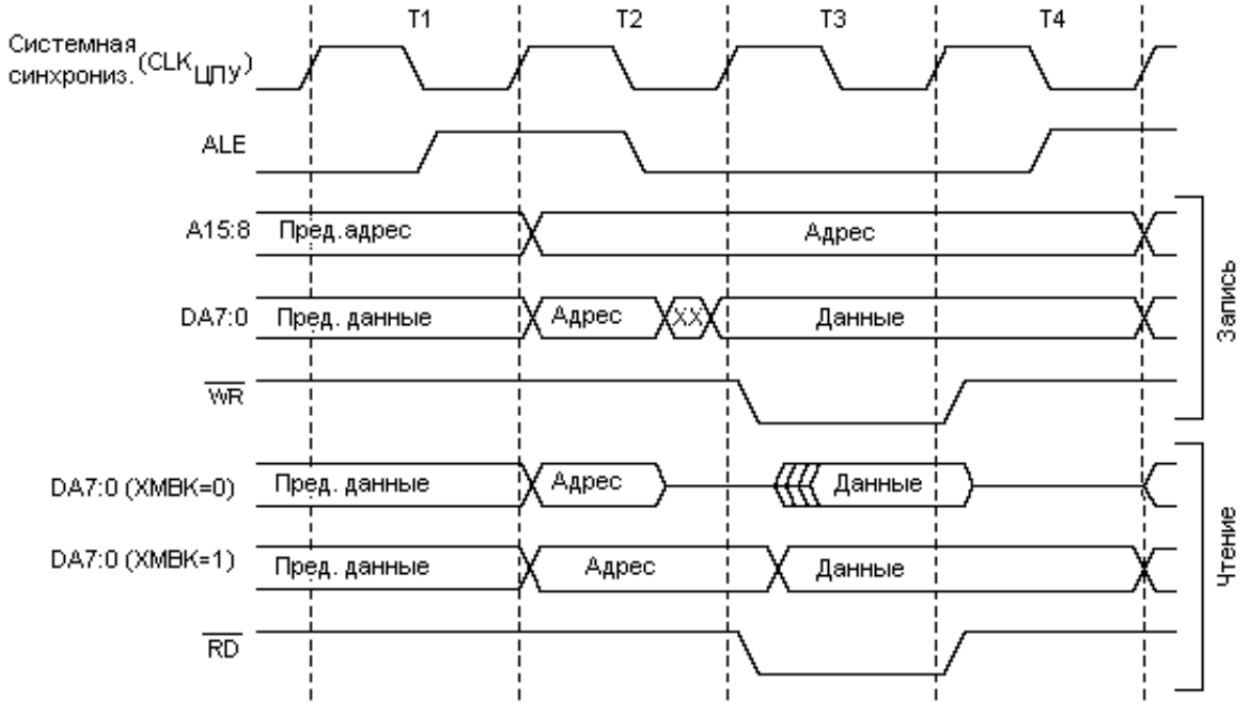

Рисунок 7 — Временная диаграмма циклов чтения и записи без дополнительных задержек

1.4.3 Подключение модуля CH340C

Сопряжение микроконтроллера с персональным компьютером по USB реализовано на микросхеме USB-UART преобразователя CH340C (DD5). На её выводы UD+ и UD– подведены линии D+ и D– разъёма USB XP2, а вывод VCC подключён к шине +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.

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 данной схеме не используются и остаются свободными или подтянутыми к фиксированным уровням согласно рекомендациям документации к микросхеме.

1.4.4 Подключение усилителя PAM8403 и RC-фильтра

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 питается от линии +5 В, к его выводам питания подключены конденсаторы C2, C9 и C10, обеспечивающие развязку питания микросхемы согласно документации. В данной схеме используется только правый канал усилителя, пины второго кана притянуты к земле. Выход усилителя соединён с разъёмом XP3, к которому подключается динамик.

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), 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 0,3-3,4 кГц). 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).

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

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

$$C_3 = \frac{1}{2\pi * R_1 * f_c} = \frac{1}{2\pi * 1000 \text{ Ом} * 3200 \text{ Гц}} \approx 50 \text{ нФ}, \tag{2}$$

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.

1.4.5 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)

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

где P_{RC} – искомая мощность,

$U_{\text{питания}}$ – напряжение питания,

R_1 – сопротивление резистора, состоящего в RC-фильтре.

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).

$$P = I_{\text{питания}} * U_{\text{питания}}, \tag{4}$$

где P – потребляемая мощность микросхемы,

$I_{\text{питания}}$ – потребляемый ток,

$U_{\text{питания}}$ – напряжение питания.

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

$$\begin{gathered} P_{\text{sum}} = \sum_i P_i = \sum_i I_i * U_{\text{питания}} + P_{\text{RC}} = \\ = (10 + 0.08 + 10 + 3.5 + 20)\ \text{мА} * 5\text{В} + 25\ \text{мВт} \approx 70\ \text{мВт}, \end{gathered} \tag{5}$$

где P_{sum} – суммарная мощность всего устройства,

P_i – мощность i-го устройства.

Таким образом суммарная мощность устройства составляет $P_{\text{sum}} \approx 70$ мВт.

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. Редактирование исходного кода выполняется в среде Visual Studio Code.

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 промежуточных файлов. 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.

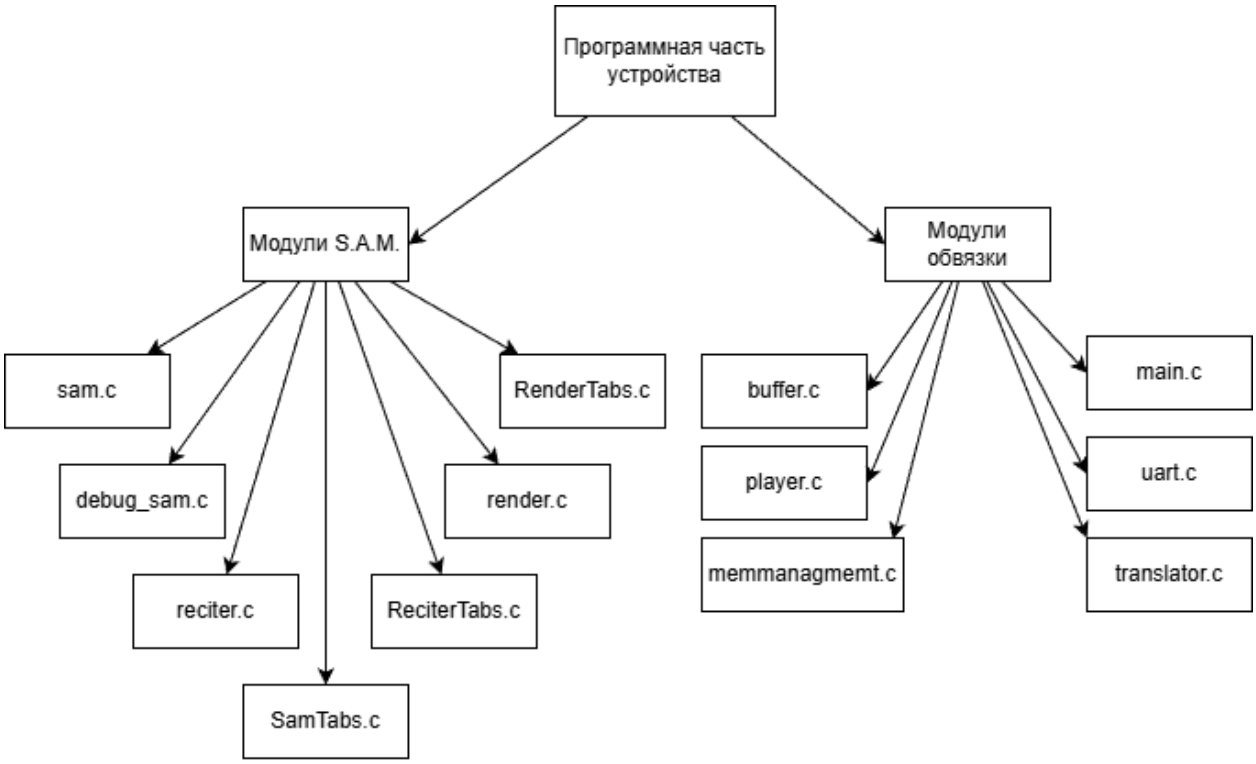

Рисунок 8 — Структурная схема

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 «memmanagment.h» и «memmanagment.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.

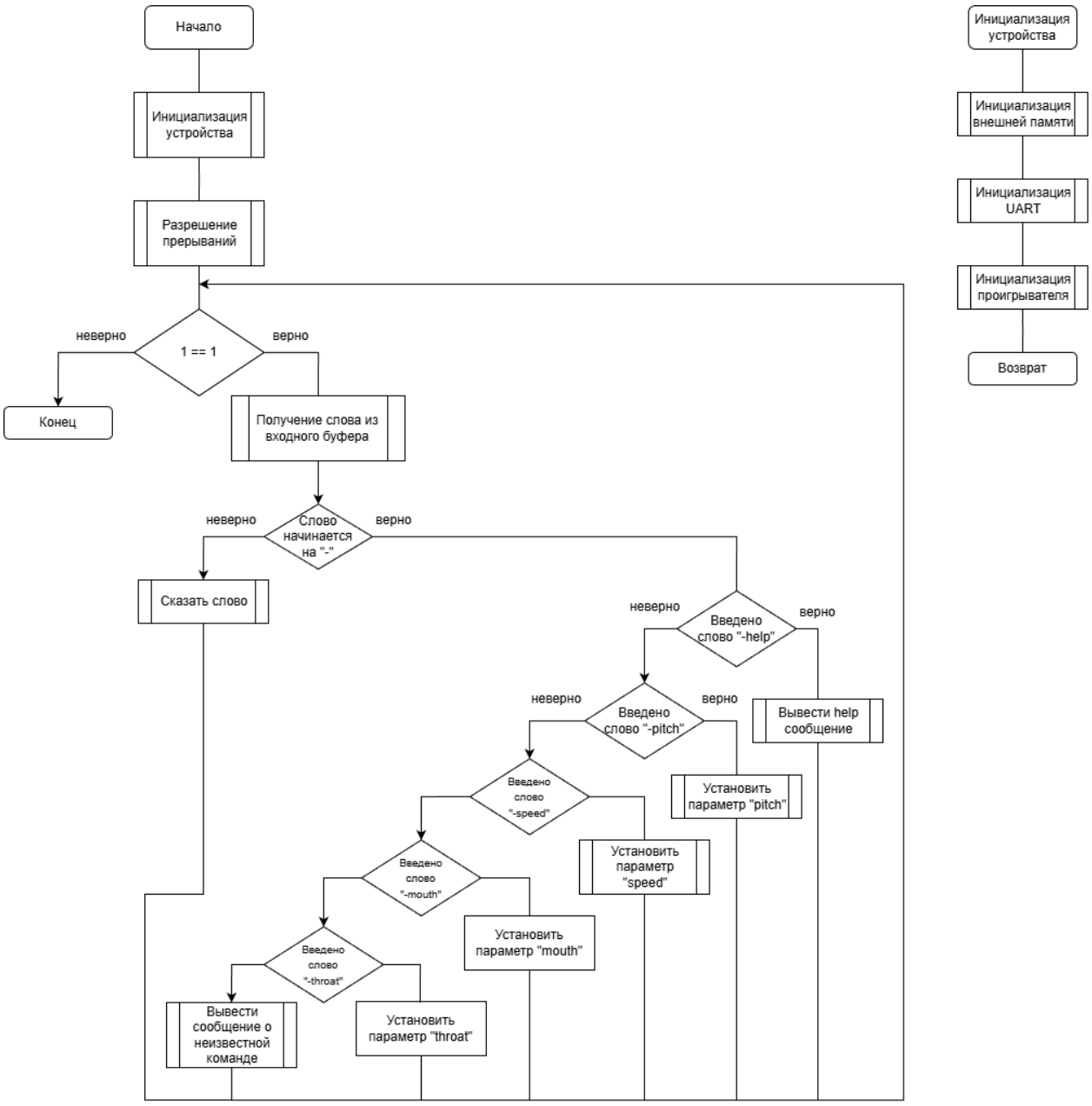

Рисунок 9 — Основной цикл работы

Основной цикл организован как обработчик входного потока слов. 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 S.A.M. 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 S.A.M. 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 ударения 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.

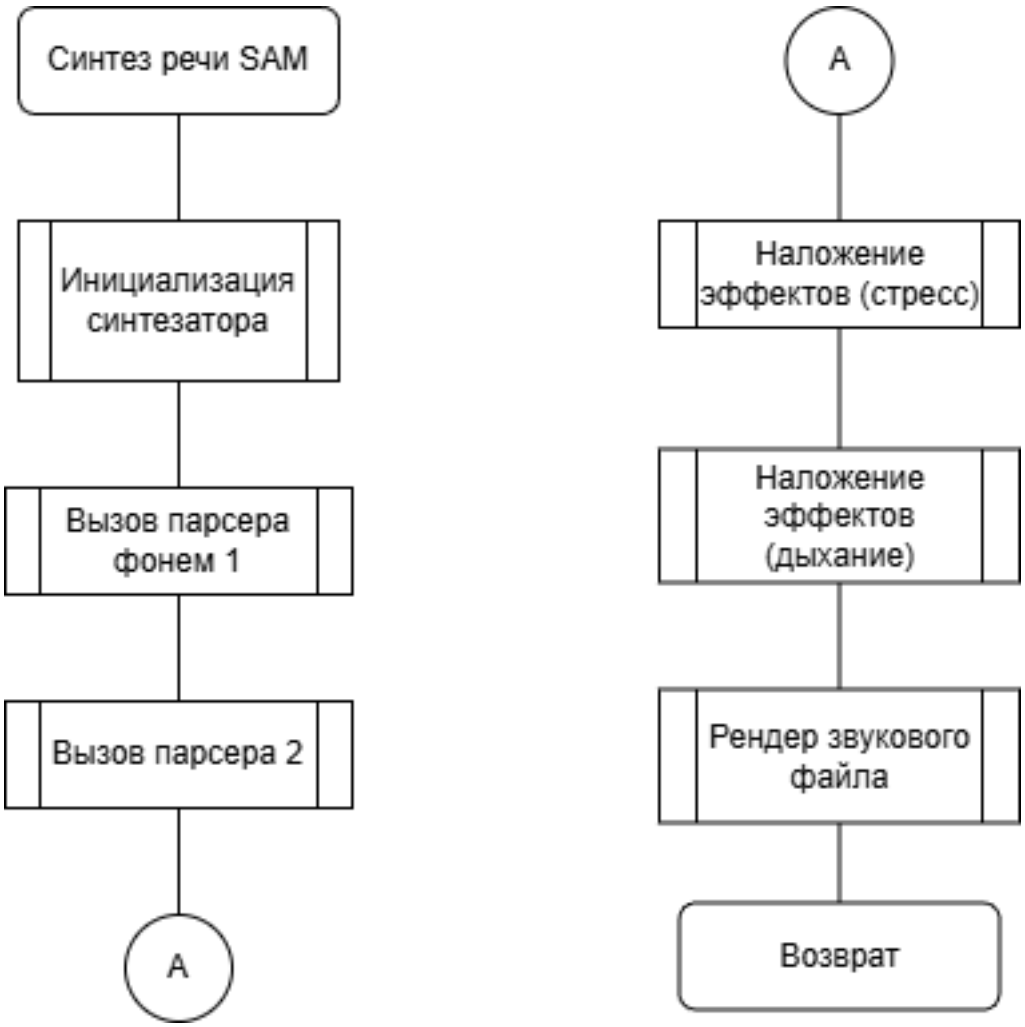

Рисунок 11 — Схема алгоритма синтеза речи S.A.M.

2.4.1 Устранение ошибок сборки

Исходный код S.A.M. организован в несколько файлов, но многие переменные и функции предполагаются глобальными и не имеют явных прототипов. Для инкрементальной сборки с разделением на объектные файла 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 .c-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 большой 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 ценный объём внутренней SRAM.

Реализация функции «copy_pgm_str» приведена в файле «memmanagment.c». Она 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, что достаточно для диагностических и сервисных сообщений, используемых в проекте.

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.

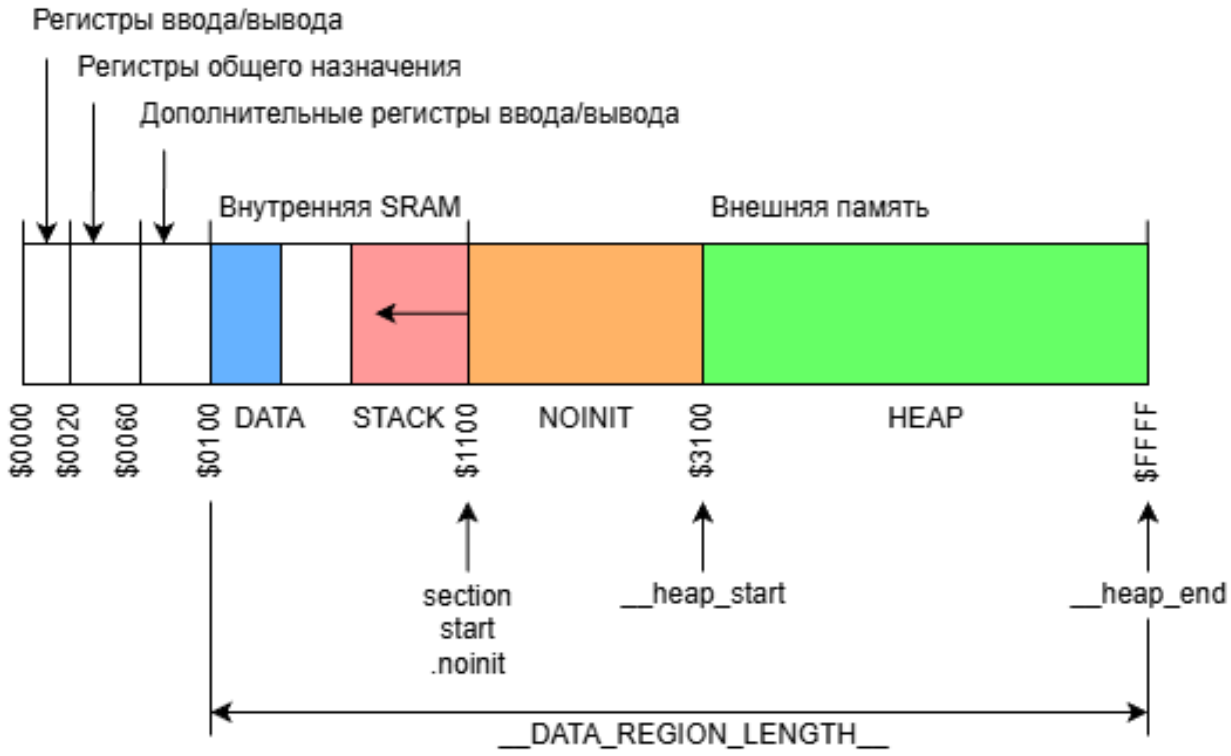

Рисунок 12 — Разметка адресного пространства данных

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.

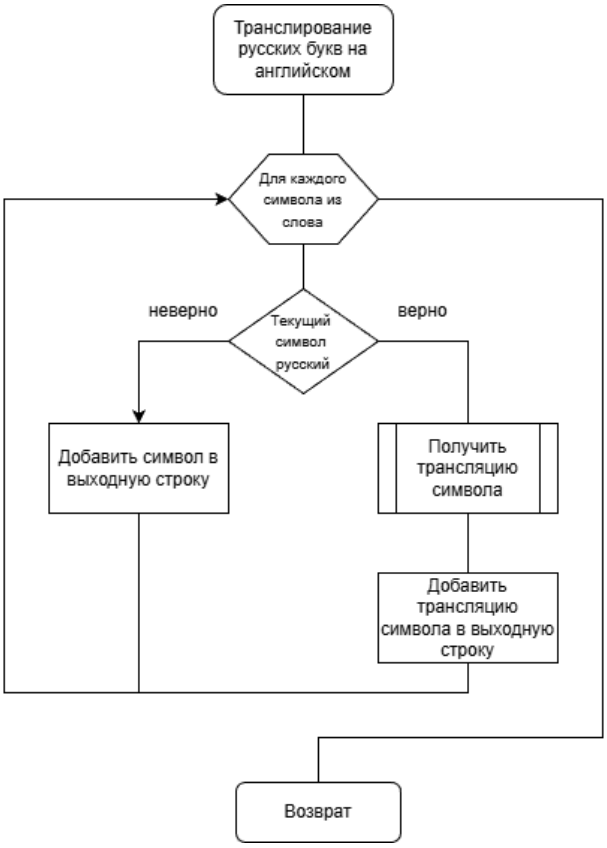

Рисунок 13 — Схема алгоритмов модуля трансляции

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

2.6 Достижение максимальной утилизации вычислительного ядра

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, 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.

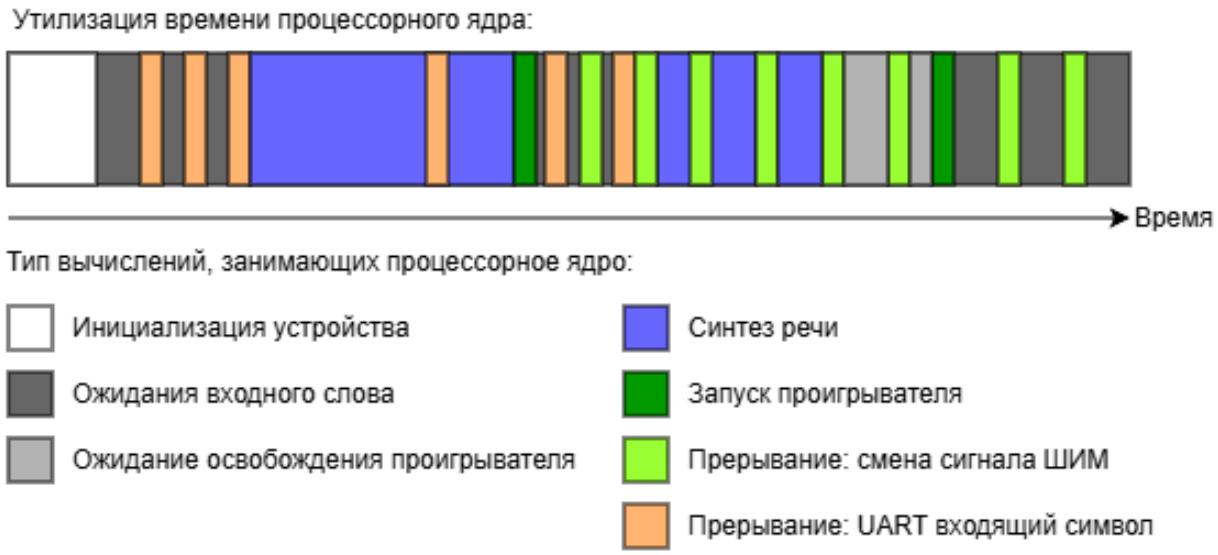

Рисунок 14 — Использование процессорного времени

На временной 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 смещеения.

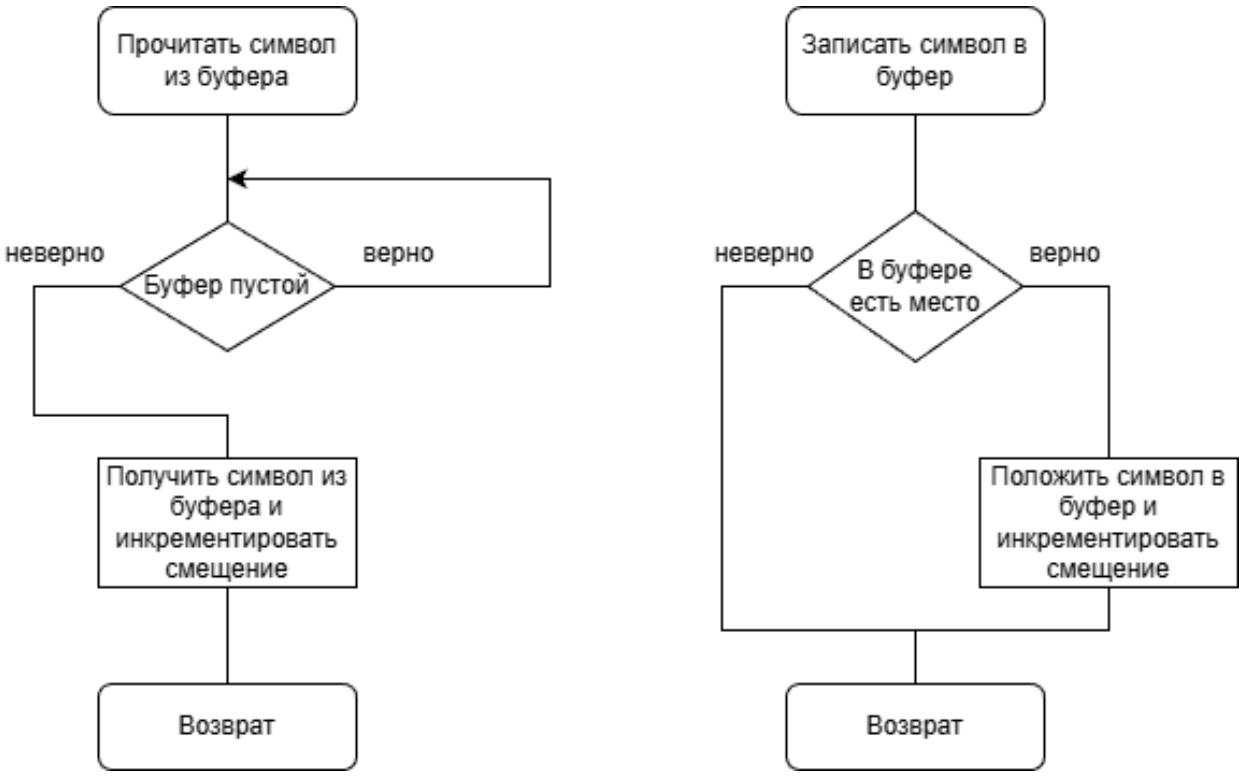

Рисунок 15 — Схема алгоритмов буфера входного потока

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 прогрывания); в контексте устройства – в callback-функции выполняется освобождение памяти из-под использованного звукового буфера. Функция init_player готовит периферию к работе проигрывателя: настраивает вывод OC0 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 выборок: 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.

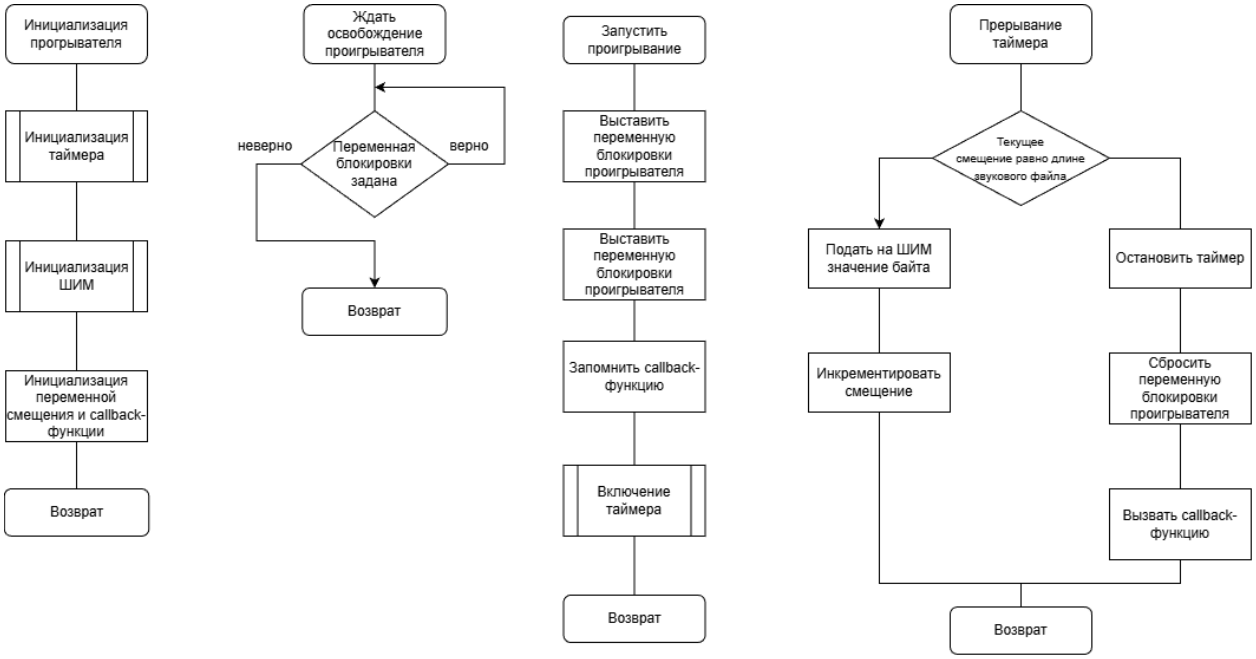

Рисунок 16 — Схема алгоритмов проигрывателя

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ

Листов 1

TODO

ПРИЛОЖЕНИЕ Б

СПЕЦИФИКАЦИЯ РАДИОЭЛЕМЕНТОВ СХЕМЫ

Листов 1

C13-14	КМ5Б-Н90-20 нФ	2	
	Микросхемы		
DD1	ATMega128A	1	
DD2	SN74HC573A	1	
DD3	AS6C1008	1	
DD4	PAM8403	1	
DD5	CH340C	1	
	Резисторы		
R1-R2	МЛТ 0,125 – 10 кОм	2	
	Соединители контактные		
XP1	DS-1023-2x3	1	
XP2	DS-1023-1x4	1	
	Резонаторы		
Z1	HC-49U 8 МГц	1	

Подп. и дата

Инв. № дубл.

Инв. №