|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное 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:***

***\_\_\_\_\_Устройство речевого ввода – вывода \_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-73Б |  |  | А.А. Смирнов |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Руководитель |  |  |  | С.А. Хохлов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |

*2019 г.*

**Задание (2 экземпляра)**

# РЕФЕРАТ

Записка 46 стр., 8 таб., 20 рис., 6 источников, 2 прил.

МИКРОКОНТРОЛЛЕР, МИЛАНДР, Miladnr, АЦП, ЦАП, ЗВУК

В ходе работы над данным курсовым проектом были исследованы МК К1986ВЕ92QI, его отладочная плата, модули АЦП, ЦАП, 16-разрядные таймеры, для реализации функции записи, хранения и воспроизведения речевых сообщений. В целях хранения речевых сообщений используется энергонезависимая память EEPROM.

Программа протестирована, отлажена с помощью среды разработки Keil uVision 5.

Материалы по курсовой работе представлены в виде графической части, приложений со схемами и отлаженным программным кодом для микроконтроллера и расчетно-пояснительной записки.

# СПИСОК СОКРАЩЕНИЙ

МК – микроконтроллер

ADC - Analog-to-digital converter (аналого-цифровой преобразователь)

DAC – Digital-to-analog converter (цифро-аналоговый преобразователь)

RAM – Random access memory (оперативное запоминающее устройство)

EEPROM - Electrically Erasable Programmable Read-Only Memory (электрически стираемое перепрограммируемое ПЗУ)

LCD – liquid crystal display (жидкокристаллический дисплей)

USB - Universal Serial Bus (универсальная последовательная шина)

ПО – программное обеспечение

# СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 7](#_Toc27938621)

[1 Конструкторская часть 8](#_Toc27938622)

[1.1 Описание архитектуры микроконтроллера 8](#_Toc27938623)

[1.2 Описание отладочной платы 18](#_Toc27938624)

[1.3 Описание контроллера АЦП 21](#_Toc27938625)

[1.4 Описание контроллера ЦАП 24](#_Toc27938626)

[1.5 Описание контроллера Flash-памяти MDR\_EEPROM 26](#_Toc27938627)

[1.6 Описание микрофонного модуля 27](#_Toc27938628)

[1.7 Алгоритмы работы программ 28](#_Toc27938629)

[1.7.1 Алгоритм главной программы 28](#_Toc27938630)

[1.7.2 Алгоритм записи 30](#_Toc27938631)

[1.7.3 Алгоритм воспроизведения 33](#_Toc27938632)

[1.8 Расчет потребляемой мощности 35](#_Toc27938633)

[2 Технологическая часть 36](#_Toc27938634)

[2.1 Программа разработки и отладки 36](#_Toc27938635)

[2.2 Отладка и тестирование программ 38](#_Toc27938636)

[2.3 Оценка времени работы модулей 40](#_Toc27938637)

[2.4 Способ программирования памяти программ 43](#_Toc27938638)

[ЗАКЛЮЧЕНИЕ 45](#_Toc27938639)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 46](#_Toc27938640)

[ПРИЛОЖЕНИЕ А – Спецификация 47](#_Toc27938641)

[ПРИЛОЖЕНИЕ Б – текст программы 48](#_Toc27938642)

# ВВЕДЕНИЕ

С развитием технологий разработки микросхем, сложные вычислительные задачи могут быть решены с помощью микроконтроллеров, имеющих очень компактные размеры.

АО "ПКК Миландр" — ведущий российский разработчик и производитель изделий микроэлектроники (микроконтроллеры, микропроцессоры, микросхемы памяти, микросхемы приемопередатчиков, микросхемы преобразователей напряжения, радиочастотные схемы), универсальных электронных модулей и приборов промышленного и коммерческого назначения, разработки ПО для современных информационных систем и изделий микроэлектроники. Устройства данной компании пользуются достаточно высокой популярностью.  
 Цель работы – разработка и реализация устройства речевого ввода и вывода на основе микроконтроллера. Устройство должно предоставлять возможность записи речи, ее хранения и воспроизведения. Управление осуществляется с помощью пульта оператора.

Работа выполняется на основе микросхемы производства выше описанной АО "ПКК Миландр", а именно - К1986ВЕ92QI, имеющий все необходимые модули для реализации устройства речевого ввода-вывода.

# 1 Конструкторская часть

# Описание архитектуры микроконтроллера

Микроконтроллер К1986ВЕ92QI российской компании «Ми-ландр» (АО «ПКК Миландр») имеет следующие характеристики:

1. Ядро:

* ARM 32-битное RISC-ядро Cortex™-M3 ревизии 2.0, тактовая частота до 80 МГц;
* производительность 1.25 DMIPS/МГц (Dhrystone 2.1) при нулевой задержке памяти,
* блок аппаратной защиты памяти MPU,
* умножение за один цикл, аппаратная реализация деления.

1. Память:

* встроенная энергонезависимая Flash-память программ размером 128 Кбайт,
* встроенное ОЗУ размером 32 Кбайт,
* контроллер внешней шины с поддержкой микросхем памяти СОЗУ, ПЗУ, NAND Flash.

1. Питание и тактовая частота:

* внешнее питание 2,2 ÷ 3,6 В,
* встроенный регулируемый стабилизатор напряжения на 1,8 В для питания ядра,
* встроенные схемы контроля питания,
* встроенный домен с батарейным питанием,
* встроенные подстраиваемые RC генераторы 8 МГц и 40 кГц,
* внешние кварцевые резонаторы на 2 ÷ 16 МГц и 32 кГц,
* встроенный умножитель тактовой частоты PLL для ядра,
* встроенный умножитель тактовой частоты PLL для USB.

1. Режим пониженного энергопотребления:

* режимы Sleep, Deep Sleep и Standby,
* батарейный домен с часами реального времени и регистрами аварийного сохранения.

1. Аналоговые модули:

* два 12-разрядных АЦП (до 16 каналов),
* температурный датчик,
* двухканальный 12-разрядный ЦАП,
* встроенный компаратор.

1. Периферия:

* контроллер DMA с функциями передачи Периферия-Память, Память-Память,
* два контроллера CAN интерфейса,
* контроллер USB интерфейса с функциями работы Device и Host,
* контроллеры интерфейсов UART, SPI, I2C,
* три 16-разрядных таймер-счетчика с функциями ШИМ и регистрации событий.

до 96 пользовательских линий ввода-вывода.

1. Отладочные интерфейсы:

* последовательные интерфейсы SWD и JTAG,
* Тип корпуса - LQFP64,
* Ближайший аналог - STM32F103x,
* Температурный диапазон – минус 45 °С ...+85°С.

Таким образом, микроконтроллер может быть использован для решения широкого спектра задач, так как обладает внушительными характеристиками.

На рисунке 1 представлена схема расположения выводов данного микроконтроллера. При этом почти все выводы с портов ввода/вывода имеют альтернативные функции.

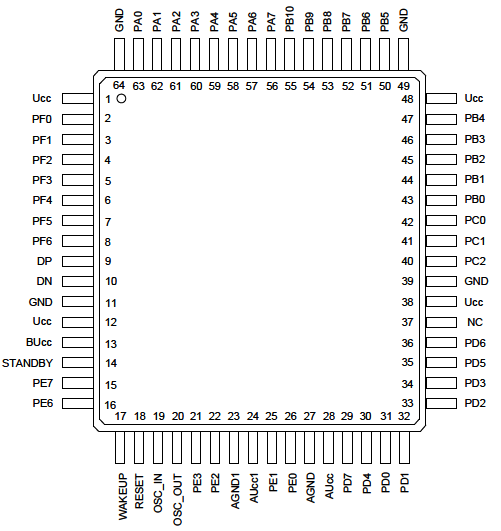


Рисунок 1 – Схема выводов микроконтроллера

Назначение линий портов микроконтроллера приведено в таблице 1. Для того, чтобы линии порта перешли под управление того или иного периферийного блока, необходимо задать для выбранных линий выполняемую функцию и настройки.

Таблица 1 - Функции линий портов микроконтроллера (часть 1)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Линия** | **Вывод** |  | **Цифровая функция** |  | **Аналоговая функция** |
| **Основная** | **Альтернат.** | **Переопред.** |
| **1** | **2** | **3** | **4** | **5** | **6** |
|  | | | Порт A |  | |
| PA0 | 63 | DATA0 | EXT\_INT1 | – | – |
| PA1 | 62 | DATA1 | TMR1\_CH1 | TMR2\_CH1 | – |
| PA2 | 61 | DATA2 | TMR1\_CH1N | TMR2\_CH1N | – |
| PA3 | 60 | DATA3 | TMR1\_CH2 | TMR2\_CH2 | – |
| PA4 | 59 | DATA4 | TMR1\_CH2N | TMR2\_CH2N | – |
| PA5 | 58 | DATA5 | TMR1\_CH3 | TMR2\_CH3 | – |
| PA6 | 57 | DATA6 | CAN1\_TX | UART1\_RXD | – |
| PA7 | 56 | DATA7 | CAN1\_RX | UART1\_TXD | – |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | Порт B |  | |
| PB0 | 43 | DATA16 | TMR3\_CH1 | UART1\_TXD | – |
| PB1 | 44 | DATA17 | TMR3\_CH1N | UART2\_RXD | – |

Таблица 1 - Функции линий портов микроконтроллера (часть 2)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Линия** | **Вывод** |  | **Цифровая функция** |  | **Аналоговая функция** | |
|  |  | **Основная** | **Альтернат.** | **Переопред.** |  | |
| **1** | **2** | **3** | **4** | **5** | **6** | |
| PB4 | 47 | DATA20 | TMR3\_BLK | TMR3\_ETR | – | |
| PB5 | 50 | DATA21 | UART1\_TXD | TMR3\_CH3 | – | |
| PB6 | 51 | DATA22 | UART1\_RXD | MR3\_CH3N | – | |
| PB7 | 52 | DATA23 | nSIROUT1 | TMR3\_CH4 | – | |
| PB8 | 53 | DATA24 | COMP\_OUT | TMR3\_CH4N | – | |
| PB9 | 54 | DATA25 | nSIRIN1 | EXT\_INT4 | – | |
| PB10 | 55 | DATA26 | EXT\_INT2 | nSIROUT1 | – | |
|  |  |  | Порт C |  | | |
| PC0 | 42 | – | SCL1 | SSP2\_FSS | – | |
| PC1 | 41 | OE | SDA1 | SSP2\_CLK | – | |
| PC2 | 40 | WE | TMR3\_CH1 | SSP2\_RXD | – | |
|  |  |  | Порт D |  | | |
| PD0 | 31 | TMR1\_CH1N | UART2\_RXD | TMR3\_CH1 | ADC0\_REF+ | |
| PD1 | 32 | TMR1\_CH1 | UART2\_TXD | TMR3\_CH1N | ADC1\_REF- | |
| PD2 | 33 | BUSY1 | SSP2\_RXD | TMR3\_CH2 | ADC2 | |
| PD3 | 34 | – | SSP2\_FSS | TMR3\_CH2N | ADC3 | |
| PD4 | 30 | TMR1\_ETR | nSIROUT2 | TMR3\_BLK | ADC4 |
| PD5 | 35 | CLE | SSP2\_CLK | TMR2\_ETR | ADC5 |
| PD6 | 36 | ALE | SSP2\_TXD | TMR2\_BLK | ADC6 |
| PD7 | 29 | TMR1\_BLK | nSIRIN2 | UART1\_RXD | ADC7 |
|  |  |  | Порт E |  | |
| PE0 | 26 | ADDR16 | TMR2\_CH1 | CAN1\_RX | DAC2\_OUT |
| PE1 | 25 | ADDR17 | TMR2\_ CH1N | CAN1\_TX | DAC2\_REF |
| PE2 | 22 | ADDR18 | TMR2\_CH3 | TMR3\_CH1 | COMP\_IN1 |
| PE3 | 21 | ADDR19 | TMR2\_CH3N | TMR3\_CH1N | COMP\_IN2 |
| PE6 | 16 | ADDR22 | CAN2\_RX | TMR3\_CH3 | OSC\_IN32 |
| PE7 | 15 | ADDR23 | CAN2\_TX | TMR3\_CH3N | OSC\_OUT32 |
|  |  |  | Порт F |  | |
| PF0 | 2 | ADDR0 | SSP1\_TXD | UART2\_RXD | – |
| PF1 | 3 | ADDR1 | SSP1\_CLK | UART2\_TXD | – |
| PF2 | 4 | ADDR2 | SSP1\_FSS | CAN2\_RX | – |
| PF3 | 5 | ADDR3 | SSP1\_RXD | CAN2\_TX | – |
| PF4 | 6 | ADDR4 | – | – | – |
| PF5 | 7 | ADDR5 | – | – | – |
| PF6 | 8 | ADDR6 | TMR1\_CH1 | – | – |

На рисунке 2 изображена структурная блок-схема микроконтроллера К1986ВЕ92QI, наглядно представляющая периферийные устройства и их взаимодействие. Используемые обозначения представлены в таблице 2.

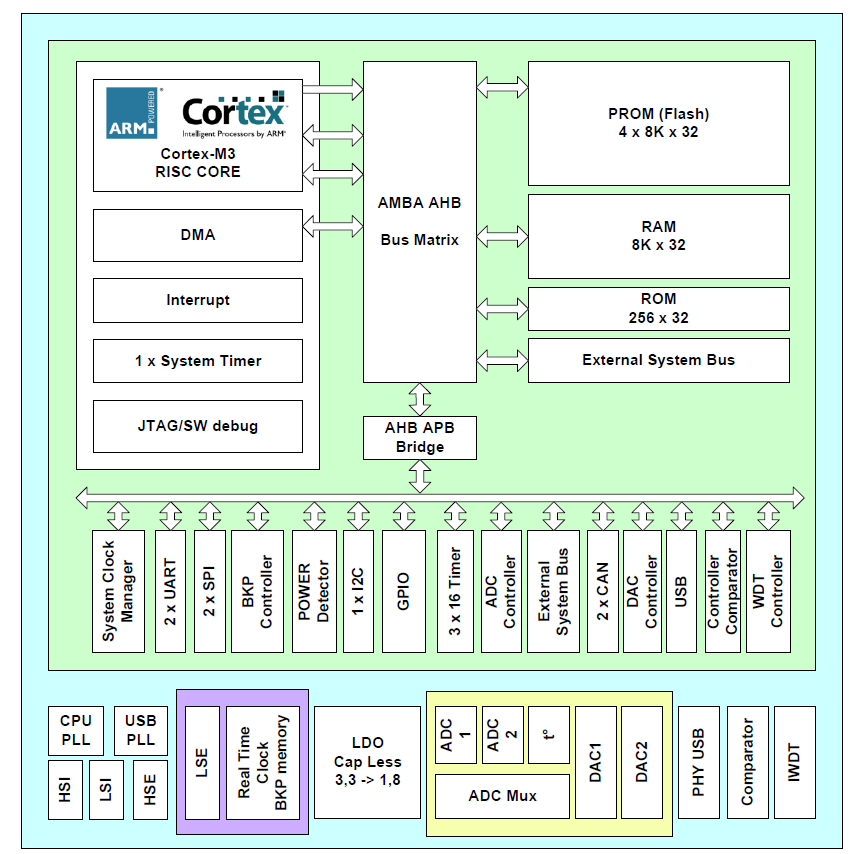


Рисунок 2 – Структурная блок-схема микроконтроллера К1986ВЕ92QI

Таблица 2 - Обозначение функциональных блоков (часть 1)

|  |  |
| --- | --- |
| **Блок** | **Описание** |
| Cortex-M3 RISC CORE | Процессорное ядро ARM Cortex-M3 архитектуры RISC |
| DMA | Контроллер прямого доступа в память |
| Interrupt | Контроллер прерываний |
| System timer | Системный таймер |
| JTAG/SW debug | Отладочный модуль через интерфейс JTAG/SW |
| AMBA AHB Bus Matrix | Шинная матрица для связи высокоскоростных внутренних компонентов |
| AHB APB Bridge | Мост для связи с периферией |
| Flash | Модуль памяти Flash |
| RAM | Модуль памяти RAM |
| ROM | Модуль памяти ROM |
| External System Bus | Внешняя системная шина |
| System Clock Manager | Модуль системного тактирования |
| UART | Контроллер UART |
| SPI | Контроллер SPI |
| BKP Controller | Контроллер резервных данных |
| Power Detector | Модуль управления питанием |
| I2C | Контроллер I2C |
| GPIO | Интерфейс ввода/вывода общего назначения |
| 16 Timer | 16-разрядный таймер |
| ADC Controller | Контроллер аналого-цифрового преобразователя |
| CAN | Контроллер CAN |
| DAC Controller | Контроллер цифро-аналогового преобразователя |
| USB | Контроллер USB |
| Controller Comparator | Контроллер компаратора |
| WDT Controller | Контроллер сторожевого таймера |

В рамках реализации устройства речевого ввода-вывода для управления некоторыми функциональными блоками (такими как GPIO, ADC Controller, DAC Controller, Flash) используются библиотеки, предоставляемые и поддерживаемыми компанией АО "ПКК Миландр". К таким библиотекам относятся MDR32F9Qx\_eeprom, MDR32F9Qx\_port, MDR32F9Qx\_adc, MDR32F9Qx\_dac, подключаемые с помощью директивы #include в начале главной программы. Это позволяет допускать меньше ошибок в процессе разработки за счет встроенной проверки данных на корректность, таким образом значительно ускоряется процесс написания и отладки программ.

Таблица 2 - Обозначение функциональных блоков (часть 2)

|  |  |
| --- | --- |
| CPU PLL | Фазовая автоподстройка частоты для процессорного ядра |
| USB PLL | Фазовая автоподстройка частоты для USB |
| HSI | Высокоскоростной внутренний генератор тактовой частоты |
| LSI | Низкоскоростной внутренний генератор тактовой частоты |
| HSE | Высокоскоростной внешний генератор тактовой частоты |
| LSE | Низкоскоростной внешний генератор тактовой частоты |
| Real Time Clock BKP memory | Резервная память |
| LDO Cap Less | Регулятор напряжения |
| ADC | АЦП |
| DAC | ЦАП |
| PSY USB | Дескриптор USB |
| Comparator | Компаратор |
| IWDT | Независимый сторожевой таймер |

Процессорное ядро имеет три системных шины:

* I Code – шина выборки инструкций,
* D Code – шина выборки данных, расположенных в коде программы;
* S Bus – шина выборки данных, расположенных в области ОЗУ.

Также в микроконтроллере реализован контроллер прямого доступа в память (DMA), который осуществляет выборку через шину DMA Bus. Ниже представлена структурная схема организации памяти микроконтроллера.

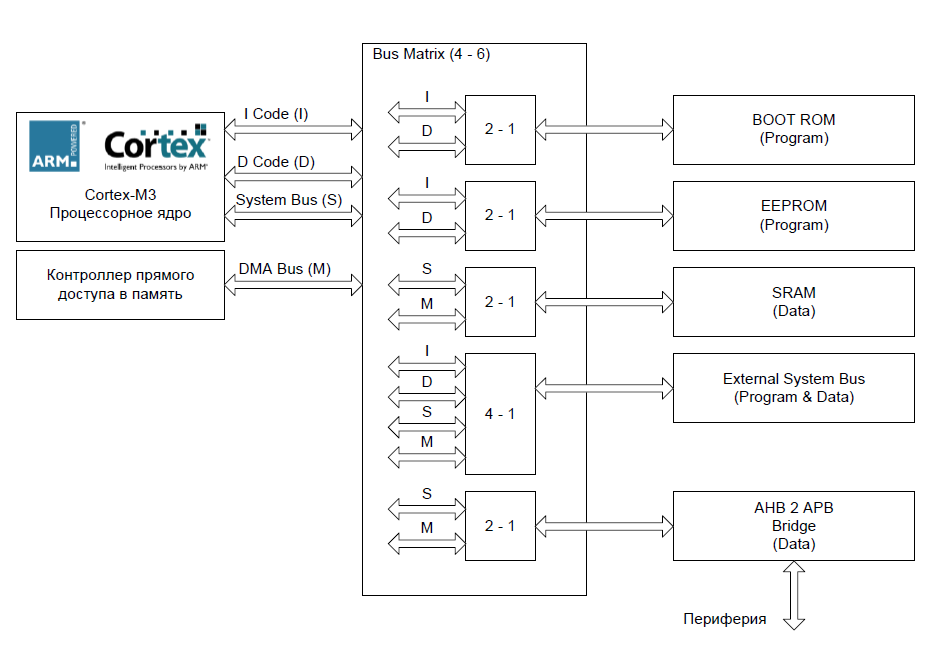


Рисунок 3 - Структурная схема организации памяти

По умолчанию для записи программ и другой информации используется область памяти начиная с 0x08000000 и размером в 0x20000 EEPROM памяти (128К).

После включения питания и снятия внутренних (POR) и внешних (RESET) сигналов сброса, микроконтроллер начинает выполнять программу из загрузочной области ПЗУ BOOT ROM. В загрузочной программе микроконтроллер определяет, в каком из режимов он будет функционировать, и переходит в этот режим. Режим функционирования определяется внешними выводами MODE[2:0], при этом перед опросом состояния этих выводов, для них включается внутренняя подтяжка к шине «Общий» (встроенные резисторы подтяжки к шине «Общий» имеют сопротивление ~50 кОм). Также устанавливается бит FPOR в регистре BKP\_REG\_0E, который может быть сброшен только при отключении основного питания UCC. После перезапуска микроконтроллера уровни на выводах MODE[2:0] не влияют на режим функционирование микроконтроллера, если установлен бит FPOR.

На рисунке 4 представлена подробная схема организации адресного пространства микроконтроллера.

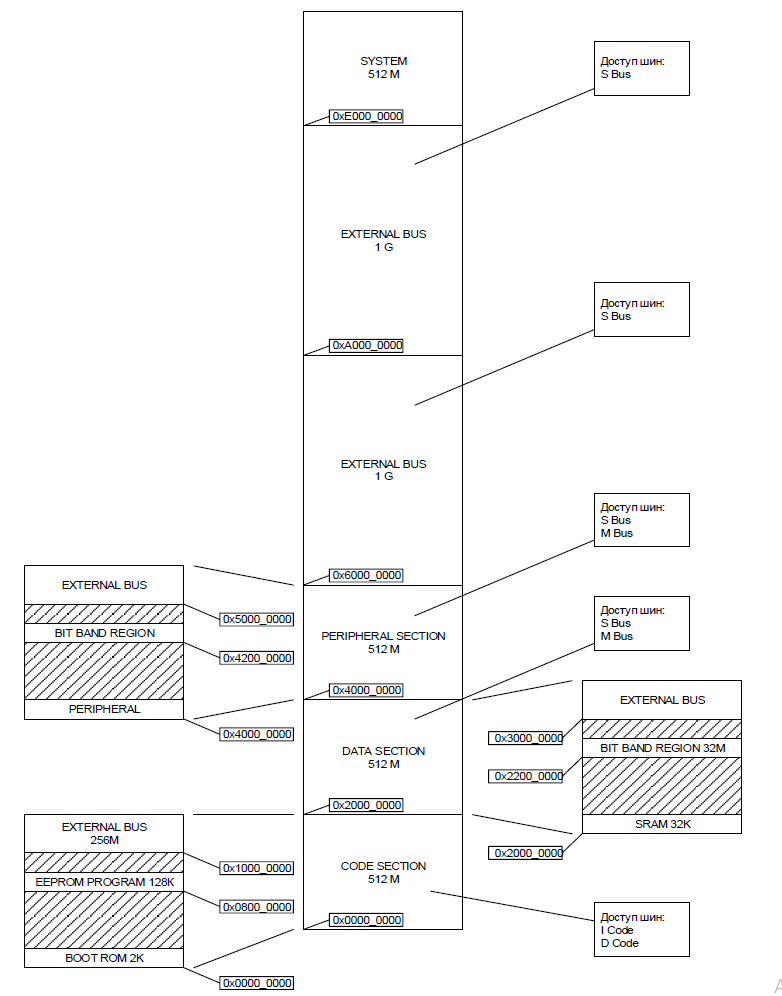


Рисунок 4 - Схема организации адресного пространства

При работе с отладочной платой BOOT SELECT (MODE[2:0]) устанавливается с помощью трех переключателей на плате. Описание режимов MODE[2:0] = 000 (JTAG\_B) и MODE[2:0] = 001 (JTAG\_A) представлены в таблице 3.

Таблица 3 – Режимы BOOT SELECT (первоначального запуска микроконтроллера)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MODE**  **[2:0]** | **Стартовый адрес** | **Описание** | **Порты** | **Описание выводов интерфейса** |
| 000 | 0х0800\_0000 | Процессор начинает выполнять программу из внутренней Flash-памяти программ. При этом установлен отладочный интерфейс JTAG\_B | PD2/JB\_TRST  PD1/JB\_TCK  PD0/JB\_TMS  PD3/JB\_TDI  PD4/JB\_TDO | В качестве выводов интерфейса используются выводы порта D, совмещенные с каналами АЦП, выводами каналов Таймера 1 и 3, UART2 и SSP2, использование которых при отладке запрещено |
| 001 | 0х0800\_0000 | Процессор начинает выполнять программу из внутренней Flash-памяти программ. При этом разрешается работа отладочного интерфейса JTAG\_А | PB4/JA\_TRST  PB2/JA\_TCK  PB1/JA\_TMS  PB3/JA\_TDI  PB0/JA\_TDO | В качестве выводов интерфейса используются выводы порта B, совмещенные с выводами данных внешней системной шины, выводами таймера 3, выводами UART1 и UART2 и CAN1, использование которых при отладке запрещено |

# Описание отладочной платы

Для разработки и отладки устройства речевого ввода-вывода использовалась отладочная плата для микросхемы К1986ВЕ92QI. На рисунке 5 представлена схема отладочной платы.

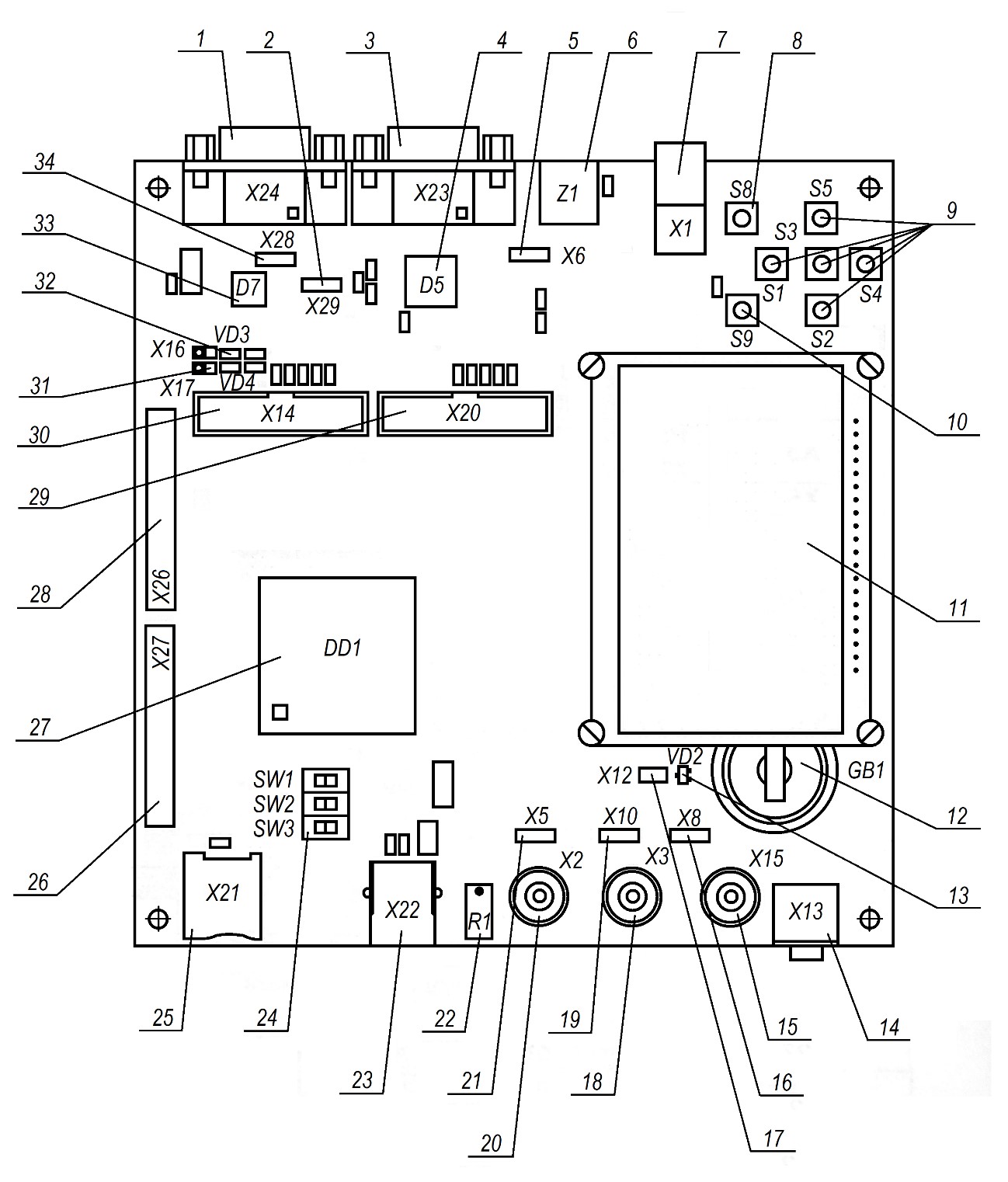


Рисунок 5 - Схема отладочной платы

Кнопки S1 «UP», S2 «LEFT», S3 «SELECT», S4 «DOWN», S5 «RIGHT» могут быть нами запрограммированы. Кнопка S1 подключена к линии PB5 порта B, кнопка S2 – к линии PE3 порта E, кнопка S3 – к линии PC2 порта C, кнопка S4 – к линии PE1 порта E, кнопка S5 – к линии PB6 порта B.

Кнопка S8 «RESET» предназначена для аппаратного сброса.

Кнопка S9 «WAKEUP» служит для выхода микроконтроллера из режима пониженного энергопотребления STANDBY.

Светодиоды VD3 и VD4 (поз. 32 на рисунке 1.3) могут служить для простейшей индикации. Элементы отладочной платы представлены в таблице 4.

Таблица 4 – Элементы отладочной платы (часть 1)

|  |  |  |
| --- | --- | --- |
| **Обозначение** | **Описание** | **Поз.** |
| DD1 | Контактное устройство для микроконтроллера | 27 |
| D5 | Приемопередатчик RS-232 | 4 |
| D7 | Приемопередатчик CAN | 33 |
| GB1 | Батарейный отсек | 12 |
| R1 | Подстроечный резистор канала 7 АЦП | 22 |
| SW1SW3 | Переключатели | 24 |
| S1-S5 | Кнопки UP, LEFT, SELECT, DOWN, RIGHT | 9 |
| S8 | Кнопка RESET | 8 |
| S9 | Кнопка WAKEUP | 10 |
| VD2 | Транзистор для подключения батарейного отсека | 13 |
| VD3, VD4 | Набор светодиодов для порта С | 32 |
| X1 | Разъем питания 5В | 7 |
| X2 | Разъем BNC внешнего сигнала канала 7 АЦП | 20 |
| X3 | Разъем BNC внешнего сигнала на 1-м входе компаратора | 18 |
| X5 | Разъем для установки конфигурационных перемычек | 21 |
| X6 | 5 |
| X8 | 16 |
| X10 | 19 |
| X12 | 17 |

Таблица 4 - Элементы отладочной платы (часть 2)

|  |  |  |  |
| --- | --- | --- | --- |
| **Обозначение** | | **Описание** | **Поз.** |
| X13 | | Разъем Audio 3,5 мм выхода ЦАП1 через звуковой усилитель | 14 |
| X14 | | Разъем отладки JTAG-A | 30 |
| X15 | | Разъем BNC выхода ЦАП-1 | 15 |
| X16, X17 | | Разъемы для установки конфигурационных перемычек | 31 |
| X20 | | Разъем отладки JTAG-B | 29 |
| X21 | | Разъем карты памяти micro-SD | 25 |
| X22 | Разъем USB-B | | 23 |
| X23 | Разъем интерфейса RS-232 | | 3 |
| X24 | Разъем интерфейса CAN | | 1 |
| X26 | Разъем портов B, C, D микроконтроллера | | 28 |
| X27 | Разъем портов A, E, F микроконтроллера | | 26 |
| X28 | Разъем для установки конфигурационных перемычек | | 34 |
| X29 | 2 |
| Z1 | Фильтр питания | | 6 |
| – | Жидкокристаллический модуль | | 11 |

# Описание контроллера АЦП

В микроконтроллере реализовано два 12-разрядных АЦП. С помощью АЦП можно оцифровать сигнал от 16 внешних аналоговых выводов порта D и от двух внутренних каналов, на которые выводятся датчик температуры и источник опорного напряжения. Скорость выборки составляет до 512 тысяч преобразований в секунду для каждого АЦП.

В качестве опорного напряжения преобразования могут выступать:

– питание АЦП с выводов AUCC и AGND,

– внешние сигналы с выводов ADC0\_REF+ и ADC\_REF-.

Контроллер АЦП позволяет:

* + оцифровать один из 16 внешних каналов,
  + оцифровать значение встроенного датчика температуры,
  + оцифровать значение встроенного источника опорного напряжения,
  + осуществить автоматический опрос заданных каналов,
  + выработать прерывание при выходе оцифрованного значения за заданные пределы,
  + запускать два АЦП синхронно для увеличения скорости выборки.

Для осуществления преобразования требуется не менее 28 тактов синхронизации CLK. В качестве синхросигнала может выступать частота процессора CPU\_CLK, либо частота ADC\_CLK, формируемая в блоке «Сигналы тактовой частоты». Выбор частоты осуществляется с помощью бита Cfg\_REG\_CLKS. Частота CPU\_CLK формируется из частоты процессорного ядра делением на коэффициент Cfg\_REG\_DIVCLK[3:0].

На рисунке 6 представлена структурная схема контроллера АЦП.

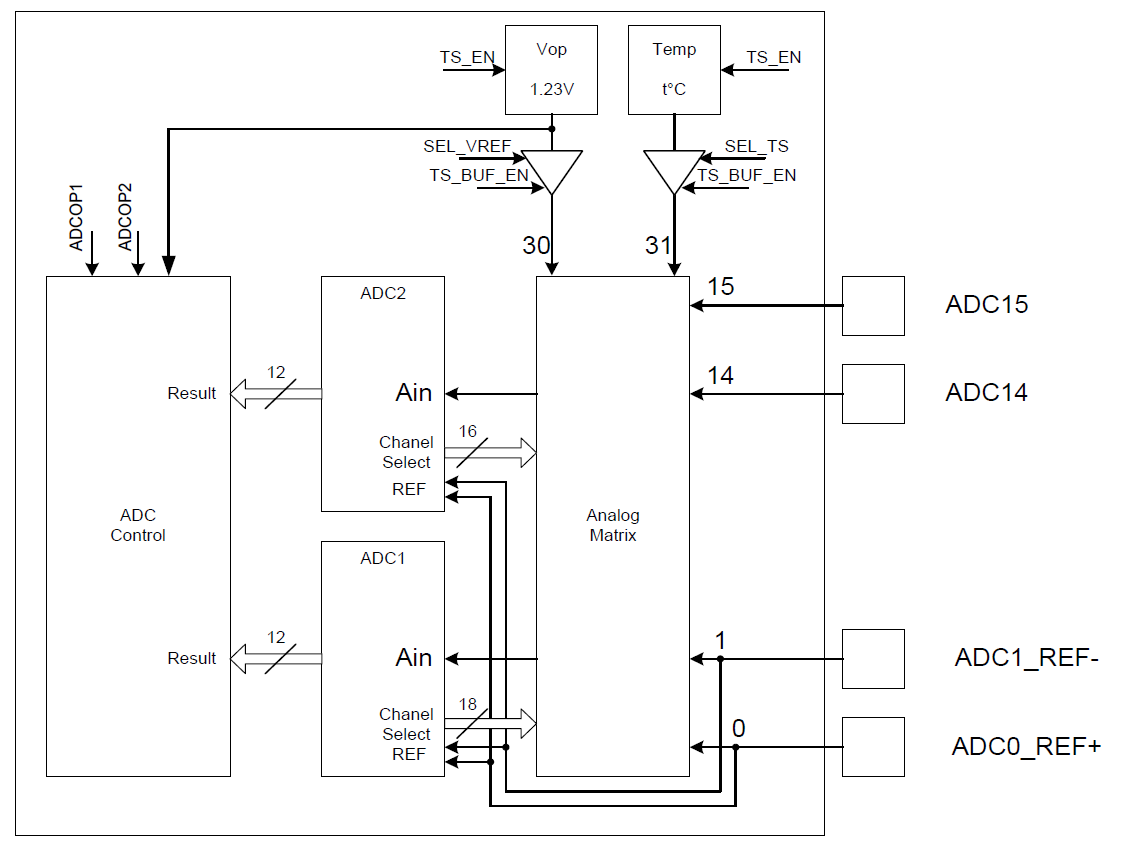


Рисунок 6 - Структурная схема контроллера АЦП

Для включения АЦП необходимо установить бит Cfg\_REG\_ADON.

Следует учитывать, что процесс преобразования состоит из двух этапов: сначала происходит заряд внутренней емкости до уровня внешнего сигнала, и затем происходит преобразование уровня заряда внутренней емкости в цифровой вид. Таким образом, для точного преобразования внешнего сигнала в цифровой вид, за время первого этапа внутренняя емкость должна зарядиться до уровня внешнего сигнала. Это время определяется соотношением номинальной внутренней емкости, входным сопротивлением тракта АЦП и выходным сопротивлением источника сигнала.

АЦП может работать как в режиме одиночного преобразования, так и в режиме последовательного преобразования.

В таблице 5 представлено описание регистров блока контроллера АЦП.

Таблица 5 - Описание регистров блока контроллера АЦП (часть 1)

|  |  |  |
| --- | --- | --- |
| **Базовый Адрес** | **Название** | **Описание** |
| 0x4008\_8000 | MDR\_ADC | Контроллер ADC |
| Смещение |  | |
| 0x00 | MDR\_ADC-> ADC1\_CFG | Регистр управления ADC1 |
| 0х04 | MDR\_ADC-> ADC2\_CFG | Регистр управления ADC2 |
| 0x08 | ADC1\_H\_LEVEL | Регистр MDR\_ADC->ADCx\_H\_LEVEL верхней границы ADC1 |
| 0x0C | ADC2\_H\_LEVEL | Регистр MDR\_ADC->ADCx\_H\_LEVEL верхней границы ADC2 |
| 0x10 | ADC1\_L\_LEVEL | Регистр MDR\_ADC->ADCx\_L\_LEVEL нижней границы ADC1 |
| 0x14 | ADC2\_L\_LEVEL | Регистр MDR\_ADC->ADCx\_L\_LEVEL нижней границы ADC2 |

Таблица 5 – Описание регистров блока контроллера АЦП (часть 2)

|  |  |  |
| --- | --- | --- |
| **Смещение** | **Название** | **Описание** |
| 0x18 | ADC1\_RESULT | Регистр MDR\_ADC->ADCx\_RESULT результата ADC1 |
| 0x1C | ADC2\_RESULT | Регистр MDR\_ADC->ADCx\_RESULT результата ADC2 |
| 0x20 | ADC1\_STATUS | Регистр MDR\_ADC->ADCx\_STATUS статуса ADC1 |
| 0x24 | ADC2\_STATUS | Регистр MDR\_ADC->ADCx\_STATUS статуса ADC2 |
| 0x28 | ADC1\_CHSEL | Регистр MDR\_ADC->ADCx\_CHSEL выбора каналов перебора ADC1 |
| 0x2C | ADC2\_CHSEL | Регистр MDR\_ADC->ADCx\_CHSEL выбора каналов перебора ADC2 |

# Описание контроллера ЦАП

В микроконтроллере реализовано два ЦАП. Для включения ЦАП необходимо установить бит Cfg\_ON\_DACx в 1, используемые выводы ЦАП порта Е были сконфигурированы как аналоговые и были отключены какие-либо внутренние подтяжки. Оба ЦАП могут работать независимо или совместно. При независимой работе ЦАП (бит Cfg\_SYNC\_A=0) после записи данных в регистр данных DACx\_DATA на выходе DACx\_OUT формируется уровень напряжения, соответствующий записанному значению. При синхронной работе (бит Cfg\_SYNC\_A=1) данные обоих ЦАП могут быть обновлены одной записью в один из регистров DACx\_DATA. ЦАП может работать от внутренней опоры Cfg\_M\_REFx=0, тогда ЦАП формирует выходной сигнал в диапазоне от 0 до напряжения питания AUCC. В режиме работы с внешней опорой Cfg\_M\_REFx=1 ЦАП формирует выходное напряжение в диапазоне от 0 до значения DACx\_REF.

На рисунке 7 представлена структурная схема контроллера ЦАП.

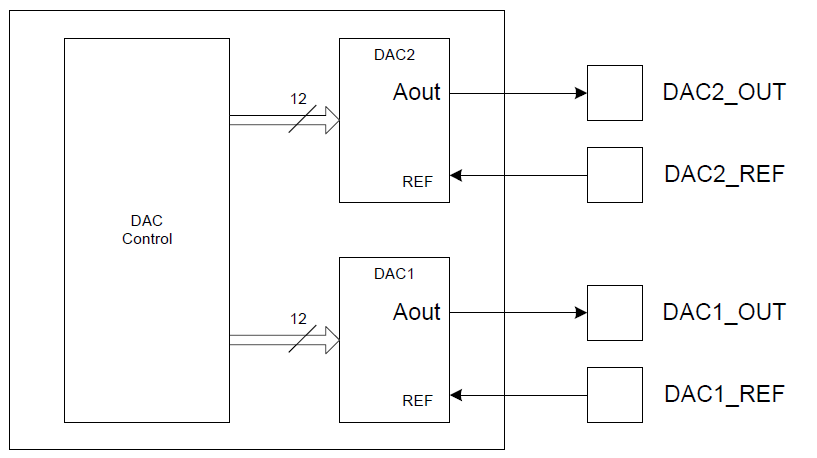


Рисунок 7 - Структурная схема контроллера ЦАП

Таблица 6 - Описание регистров блока контроллера ЦАП

|  |  |  |
| --- | --- | --- |
| **Базовый Адрес** | **Название** | **Описание** |
| 0x4009\_0000 | MDR\_DAC | Контроллер DAC |
| Смещение |  | |
| 0x00 | MDR\_DAC->CFG | Регистр управления DAC |
| 0х04 | MDR\_DAC->DAC1\_DATA | Регистр данных DAC1 |
| 0x08 | MDR\_DAC->DAC2\_DATA | Регистр данных DAC2 |

В таблице 6 представлено описание регистров блока контроллера ЦАП.

# Описание контроллера Flash-памяти MDR\_EEPROM

Микроконтроллер содержит встроенную Flash-память программ с объемом 128 Кбайт основной памяти программ и 4 Кбайта информационной памяти.

В обычном режиме (бит CON = 0, регистр EEPROM\_CMD) доступна основная память программ через системные шины I Code и D code для выборки инструкций и данных кода программы.

В режиме программирования (бит CON = 1, регистр ЕEPROM\_CMD) ,который и используется в рамках реализации заданного устройства, основная и информационная память доступны как периферийные устройства и могут быть использованы для нужд разработчика приложения. В режиме программирования программный код должен выполняться из области системной шины или ОЗУ. Выполнение программного кода из Flash-памяти программ в режиме программирования невозможно.

Память поддерживает до 10 000 циклов перезаписи, при этом не допускается повторять циклы стирания – записи – стирания одной ячейки памяти с периодом менее 4мс.

# Описание микрофонного модуля

Микросхема MAX9812 – это микрофонный усилитель с постоянным усилением 20dB. Имеет полосу пропускания 500КГц, небольшое энергопотребление. Напряжение питания – от 2.7В до 3.6В. Подробные электрические характеристики опущены.

* + – сигнал выключения, активный уровень – логическая единица,
  + GND – земля,
  + OUT – выходной сигнал,
  + VCC – питание (3.3В),
  + BIAS – вход напряжения смещения,
  + IN – вход, сигнал с которого усиливается и подается на OUT.

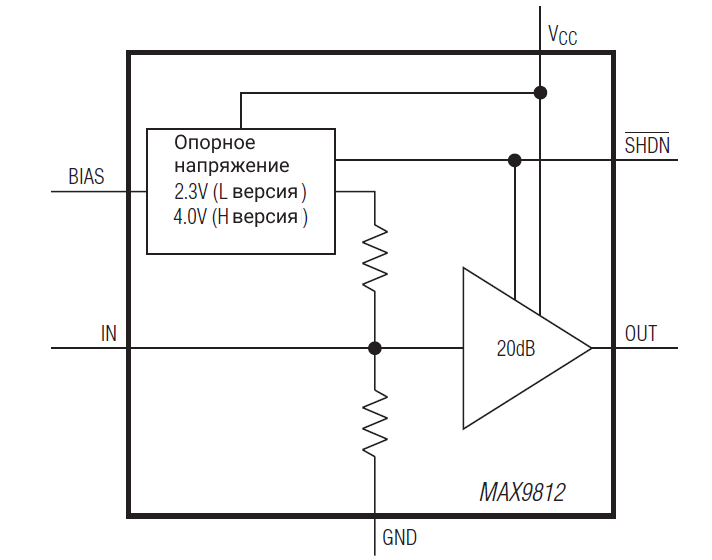


Рисунок 8 - Схема усилителя MAX9812

Для корректной работы микрофонного модуля необходимо сделать обвязку по питанию микросхемы MAX9812, добавить микрофон. Схема представлена на рисунке 9.

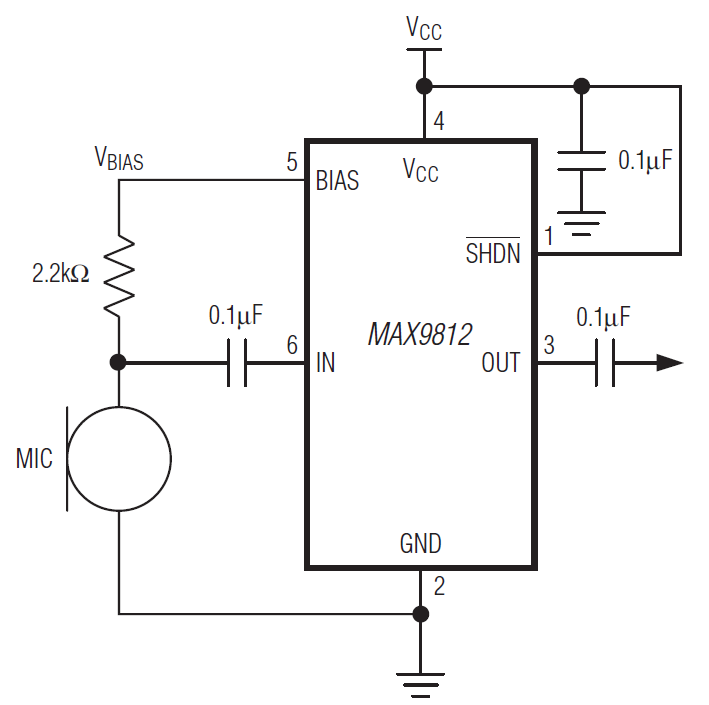


Рисунок 9 - Схема микрофонного модуля на основе усилителя MAX9812

Выход усилителя подается на линию АЦП (порт PD7) микроконтроллера.

# Алгоритмы работы программ

# Алгоритм главной программы

Главная функция int main(void) состоит из 2 частей. В первой части содержится определение всех необходимых переменных и инициализация модулей ADC, DAC, LCD, портов ввода-вывода, таймеров, устанавливается частота тактирования и на экран выводится фамилия, инициалы и группа студента. Вторая часть – это цикл while(1){…}, в котором происходит опрос кнопок, выполнение обработки их нажатия, вывод на экран дополнительной информации о назначении кнопок на пульте оператора (используются кнопки RIGHT, LEFT, SELECT, UP и DOWN отладочной платы). На рисунке 10 приведена схема алгоритма основной программы. Выбор трека осуществляется с помощью кнопок UP и DOWN (всего 2 трека).



Рисунок 10 - Схема алгоритма основной программы

Далее подробнее разберем вызываемые во время обработки нажатия кнопок подпрограммы записи и воспроизведения звука. Схемы алгоритмов других подпрограмм опущены, поскольку являются достаточно простыми (линейными).

На рисунке 11 приведено изображения экрана отладочной платы после запуска МК.

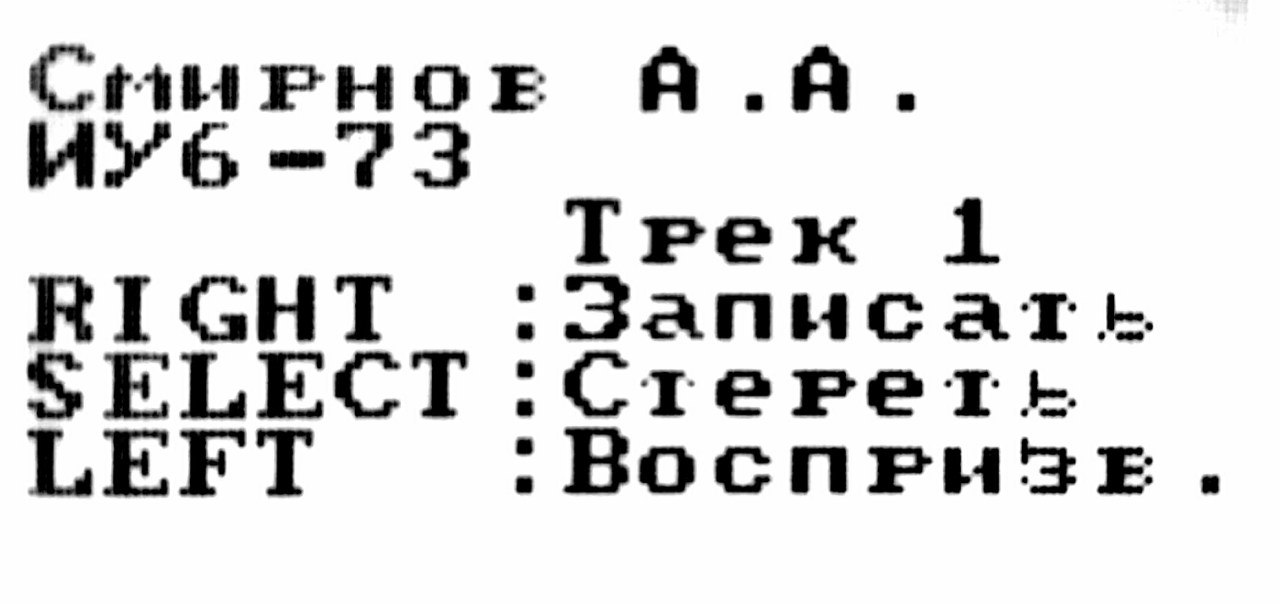


Рисунок 11 - Изображение ЖК экрана после запуска МК

# Алгоритм записи

Запись звука осуществляется с помощью вызова подпрограммы wirte\_track(). Сразу после вызова запускается 16-разрядный таймер Timer1 с помощью команды TIMER\_Cmd(MDR\_TIMER1, ENABLE). Таймер настроен на выработку прерывания каждый раз, когда он досчитывает от указанного значения (Period) до нуля. При этом устанавливается соответствующий флаг TIMER\_STATUS\_CNT\_ZERO и происходит вызов обработчика прерывания Timer1\_IRQHandler(), в котором сначала происходит сброс флага и старт следующего преобразования, а затем запись преобразованного значения в память.

Для записи сигнала используется ADC1, на вход ADC7 (PD7) поступает сигнал с микрофонного модуля. Сигнал имеет диапазон , сдвиг на 1.65В получен с помощью делителя напряжения. ADC1 настроен работу в режиме одиночных преобразований. Вызов преобразования происходит с помощью команды ADC1\_Start(), после чего необходимо дождаться установления флага конца преобразования - ADC1\_FLAG\_END\_OF\_CONVERSION. На одно преобразование уходит от 28 тактов процессора. При чтении результата данный флаг автоматически сбрасывается. Если чтения не произошло, но вызвано повторное преобразование – результат затирается и устанавливается флаг перезаписи.

Обработчик вызывается каждые Period тактов процессора. Значение Period вычисляется на основе требуемой частоты дискретизации и значения тактовой частоты процессора во время инициализации таймера. Ниже приведена формула вычисления значения Period, оно должно быть достаточным, чтобы успевать записывать очередное значение в память перед выполнением следующего преобразования (запись занимает 1514 тактов процессора).

Таймер ведет счет до того момента, когда будет заполнена вся выделенная для записи выбранного трека EEPROM память. После этого внутри обработчика прерывания таймер останавливается. На каждый трек выделено по 13 страниц памяти, одна страница – 4К. Под трек 1 выделены адреса памяти: 0x08006000 – 0x08012FFF, под трек 2: 0x08013000 – 0x08020000 (53248 байт под каждый трек).

После записи данных по начальному адресу выбранного трека записывается слово 0xABCDEFAB, которое является меткой, показывающей факт того, что запись выполнена. Эта метка проверяется при выполнении функции track\_is\_empty(), возвращающей 1, если память пуста и разрешается записывать данные, и 0, если память уже занята, пользователю об этом сообщается с помощью ЖК дисплея.

Схема алгоритма подпрограммы записи и обработчика прерывания от таймера 1 представлена на рисунке 12.



Рисунок 12 - Схема алгоритма подпрограмм записи и обработки прерывания от таймера 1

# Алгоритм воспроизведения

Для реализации вывода звука выбран ЦАП2, поскольку его вывод может быть подключен к аудио-усилителю через установку перемычки DAC\_OUT\_SEL на значение AMP. Сигнал снимается с линии PE0. Как и для других модулей, для управления контроллером используется официальная библиотека (MDR32F9Qx\_dac), а именно – функции DAC2\_Init(DAC2\_AVCC), которая устанавливает указанный источник опорного напряжения и DAC2\_Cmd(ENABLE), которая включает ЦАП и DAC2\_SetData(Data), которая отвечает за установку напряжения, равного (Data/4069)\*DAC2\_AVCC на выходе.

Воспроизведение звука реализуется с помощью функции read\_track(). Чтение данных из памяти во время произведения происходит по адресу текущего трека. Так же, как и в случае записи – для воспроизведения используется таймер (Timer 2), который генерирует прерывание каждые Period тактов процессора (совпадает с Period для записи). Обработчик прерывания от таймера 2 сначала устанавливает очередное значение напряжения на выходе ЦАП, а затем считывает новое значение из памяти и сохраняет в ту же переменную, после чего добавляет ранее отброшенные 4 бита (нулевые).

На рисунке 13 представлена схема алгоритма подпрограммы воспроизведения и обработчика прерывания от таймера 2.



Рисунок 13 - Схема алгоритма подпрограмм воспроизведения и обработки прерывания от таймера 2

# Расчет потребляемой мощности

Определим мощность, потребляемую устройством на основе справочных данных по микроконтроллеру, микрофонному модулю и принципиальной электрической схеме устройства. На рисунке 14 представлена зависимость тока потребления МК от частоты.

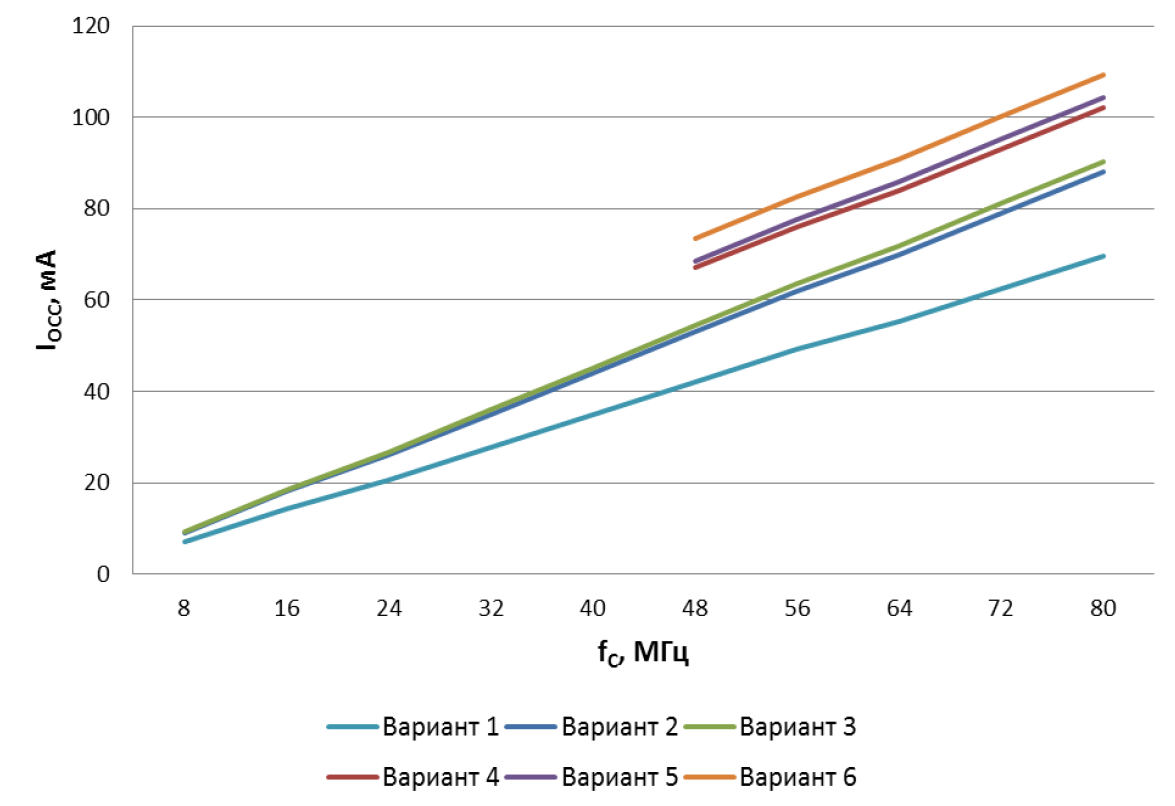


Рисунок 14 - Зависимость тока потребления от используемых периферийных устройств и частоты тактирования

* Вариант 1 – процессорное ядро,
* Вариант 2 – процессорное ядро + 1xUART+2xSPI+ 3xTIMER (используемый вариант),
* Вариант 3 – процессорное ядро + 1xUART+2xSPI+ 3xTIMER + 1xCAN,
* Вариант 4 – процессорное ядро + 1xUART+2xSPI+ 3xTIMER + USB,
* Вариант 5 – процессорное ядро + 1xUART+2xSPI+ 3xTIMER + USB + 1xCAN,
* Вариант 6 – процессорное ядро + 1xUART+2xSPI+ 3xTIMER + USB + 1xCAN + 1xADC + 1xDAC.

Из графика видно, что при частоте в 20МГц ток потребления составляет ~ 24мА, поскольку кроме процессорного ядра работают 2 таймера.

Ниже представлена таблица некоторых электрических параметров используемых микросхем для подсчета суммарной потребляемой мощности. Данные взяты из документации по соответствующим микросхемам.

Таблица 7 – Таблица некоторых электрических параметров микросхем

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Микросхема** | **Название** | **, мкА** | **, В** | **, мВт** |
| К1986ВЕ92QI | МК | 2400 | 5 | 12 |
| MAX9812 | Усилитель (на входе) | 230 | 3.3 | 0.75 |
| МТ–12864A | ЖК экран | 400 | 3.3 | 1.32 |
| MC34119 | Усилитель (на выходе) | 2700 | 3.3 | 8.9 |

Из таблицы 7 можно посчитать суммарную потребляемую мощностью с помощью формулы

# Технологическая часть

# Программа разработки и отладки

В ходе разработки устройства речевого ввода-вывода было задействовано следующее программное обеспечение: Keil uVision 5 – среде разработки, а также подключены дополнительные библиотеки «Standard Peripherals Library + software pack для Keil MDK 5», официально предоставляемые компанией АО «ПКК Миландр». Данная среда разработки поддерживает работу со многими микроконтроллерами, в том числе с МК архитектуры AVR, ARM.

Keil uVision позволяет работать с проектами любой степени сложности, начиная с введения и правки исходных текстов и заканчивая внутрисхемной отладкой кода и программированием ПЗУ микроконтроллера. От разработчика скрыта большая часть второстепенных функций, что сильно разгружает интерфейс и делает управление интуитивно понятным. Однако при возрастании сложности реализуемых задач, всегда можно задействовать весь потенциал модулей, функционирующих под управлением единой оболочки.

Преимуществом данной среды является удобство отладки. Редактирование кода остается доступным и во время отладки программы, что позволяет сразу исправлять ошибки или отмечать проблемные участки кода.

На рисунке 15 представлен интерфейс среды разработки Keil uVision 5.

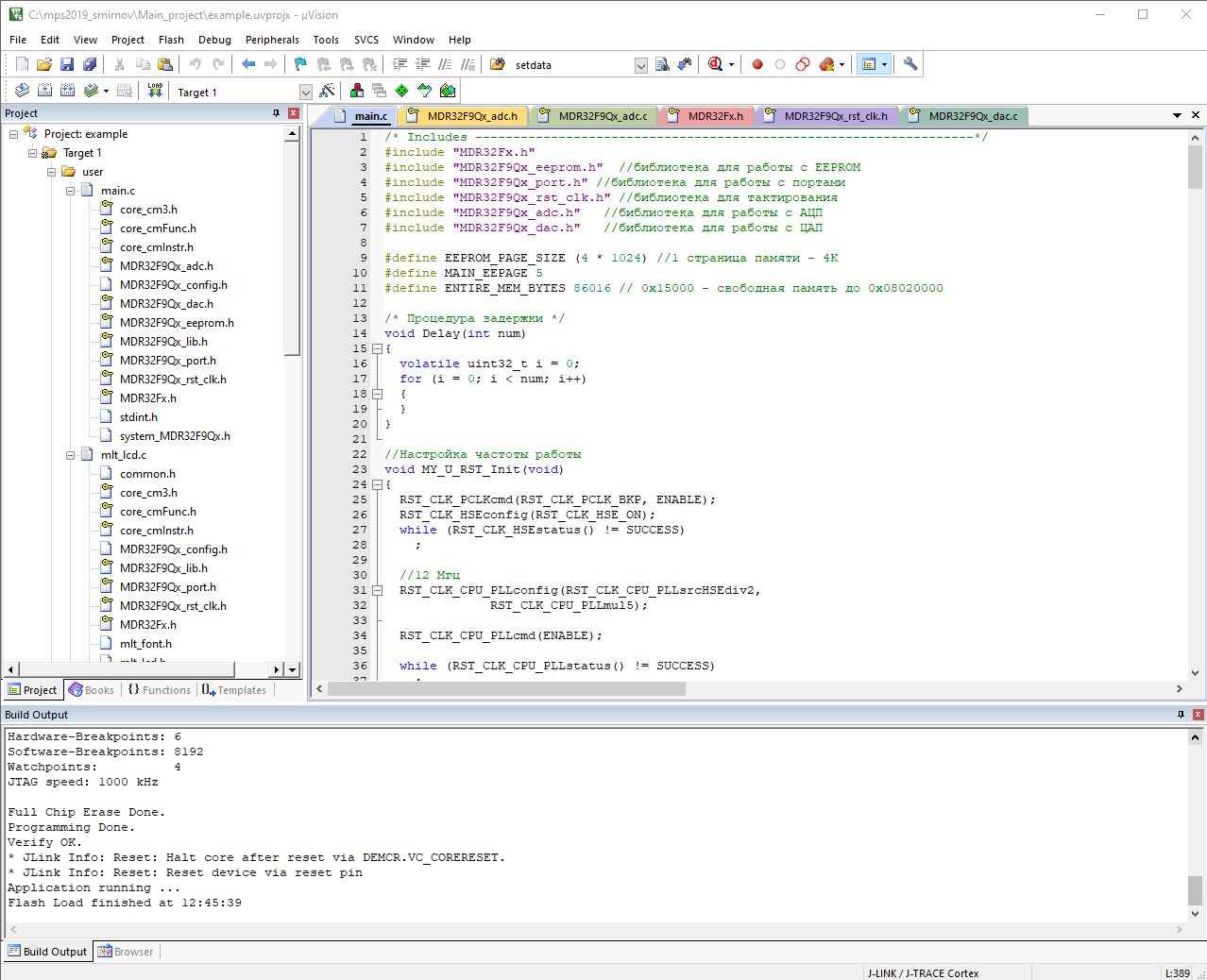


Рисунок 15 - Интерфейс среды разработки Keil uVision 5

# Отладка и тестирование программ

Тестирование программ выполняется путем ручного прохождения возможных сценариев использования. Если во всех случаях не было выявлено ошибок, то будем считать, что программа работает верно. На рисунке 16 представлена диаграмма состояний интерфейса.

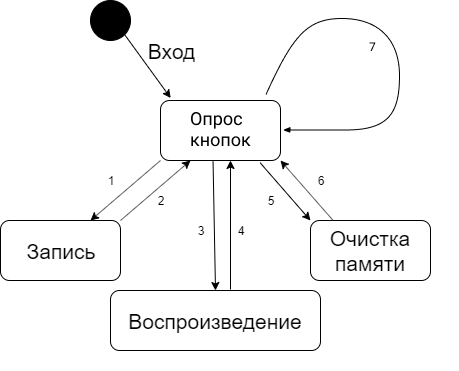


Рисунок 16 - Диаграмма состояний интерфейса

Название переходов:

* 1,2 – запись речи и возврат в состояние опроса,
* 3,4 – воспроизведение текущего трека и возврат в состояние опроса,
* 5,6 – стирание текущего трека и возврат в состояние опроса,
* 7 – смена текущего трека.

Из рисунка можно выделить 6 основных сценариев использования программы (последовательности переходов по диаграмме состояний интерфейса для каждого из треков):

* Запись – воспроизведения – очистка,
* Запись – очистка – воспроизведение,
* Воспроизведение – очистка – запись,
* Воспроизведение – запись – очистка,
* Очистка – запись – воспроизведение,
* Очистка – воспроизведение – очистка.

Все данные сценарии проверены вручную. Тестирование показало, что все возможные исключения обработаны и ошибок не возникает.

В процессе разработки программы активно использовался режим отладки в среде разработки. На рисунке 17 представлено окно отладки.

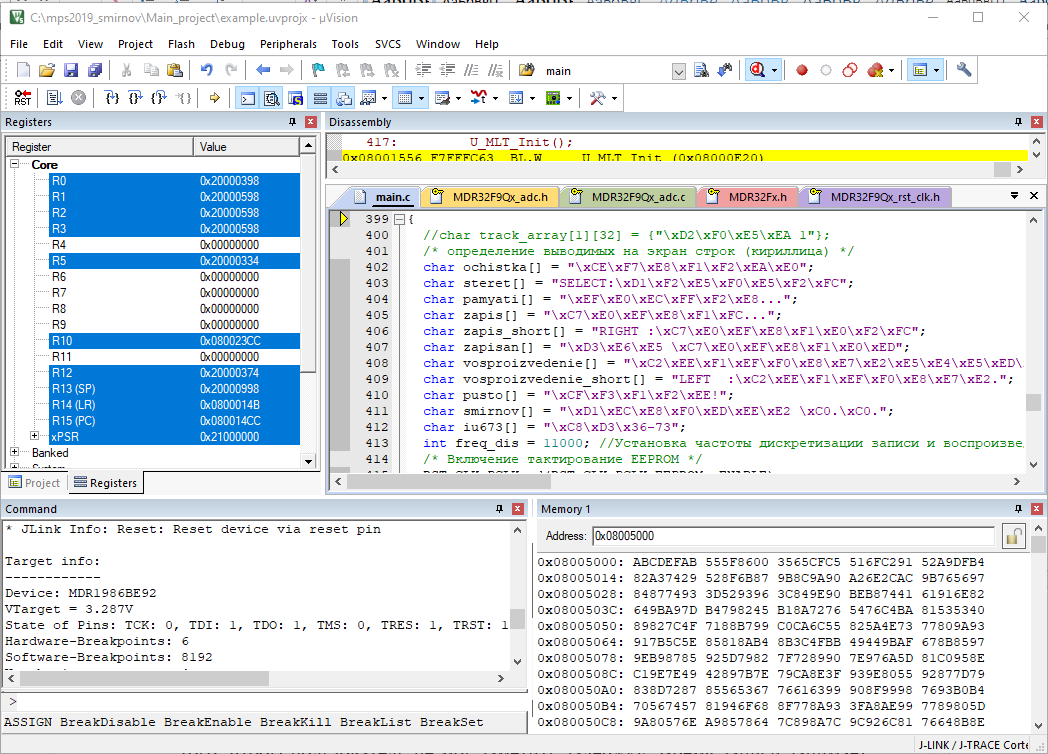
=

Рисунок 17 - Окно отладки

Для просмотра содержимого памяти после записи речи предназначено окно «Memory» в режиме отладки, для просмотра текущих действий и ошибок используется окно «Command». В окне «Registers» можно просмотреть текущее значение регистров. Поскольку во время отладки доступны как пошаговый режим, так и режим автоматического выполнения до указанного места – не составляет труда проверить все участки кода как последовательно, так и по отдельности.

# Оценка времени работы модулей

Оценим время выполнения разных блоков программы. Частота процессора установлена 20МГц, используя режим отладки вычислим время, затраченное на:

* Инициализацию всех используемых модулей и переменных,
* Одну итерацию опроса кнопок,
* Выполнение подпрограммы очистки памяти.

Результаты замеров представлены в таблице 8.

Таблица 8 - Время выполнения модулей

|  |  |  |
| --- | --- | --- |
| **Выполняемый блок** | **Количество тактов** | **Время (при F = 20Мгц), с** |
| Инициализация переменных, модулей. | 2 520 752 | 0.12 |
| Одна итерация опроса кнопок. | 892 102 | 0.04 |
| Выполнение подпрограммы очистки памяти. | 12 190 797 | 0.61 |

Как видно из таблицы 8, время инициализации переменных практически не заметно для пользователя – 0.12с, частота опроса кнопок достаточна для того, чтобы пользователь не мог заметить задержку. Время очитки программы хоть и заметно, но достаточно мало для удобной работы. Чтобы пользователь успел прочитать выведенный на экран статус «Очистка памяти…» была добавлена дополнительная задержка после завершения выполнения процедуры очистки.

Время выполнения подпрограмм записи и воспроизведения оценим отдельно, построим для этого диаграммы, демонстрирующие порядок и длительность выполняемых процедур в тактах.

На рисунке 18 представлена временная диаграмма выполнения обработчика прерывания при записи.

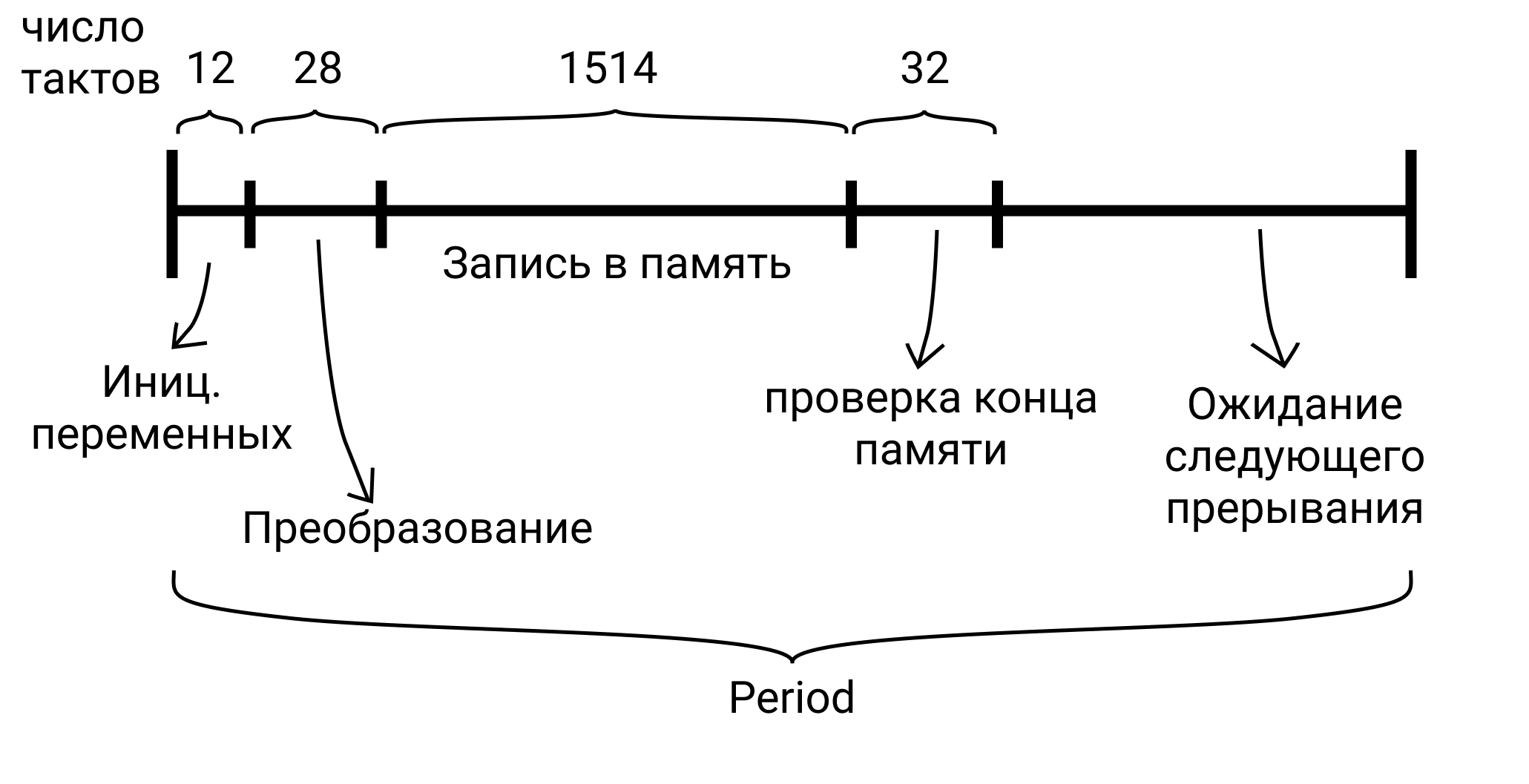


Рисунок 18 - Диаграмма выполнения записи одного значения с АЦП в память

На рисунке 19 представлена диаграмма выполнения обработчика прерывания при воспроизведении.

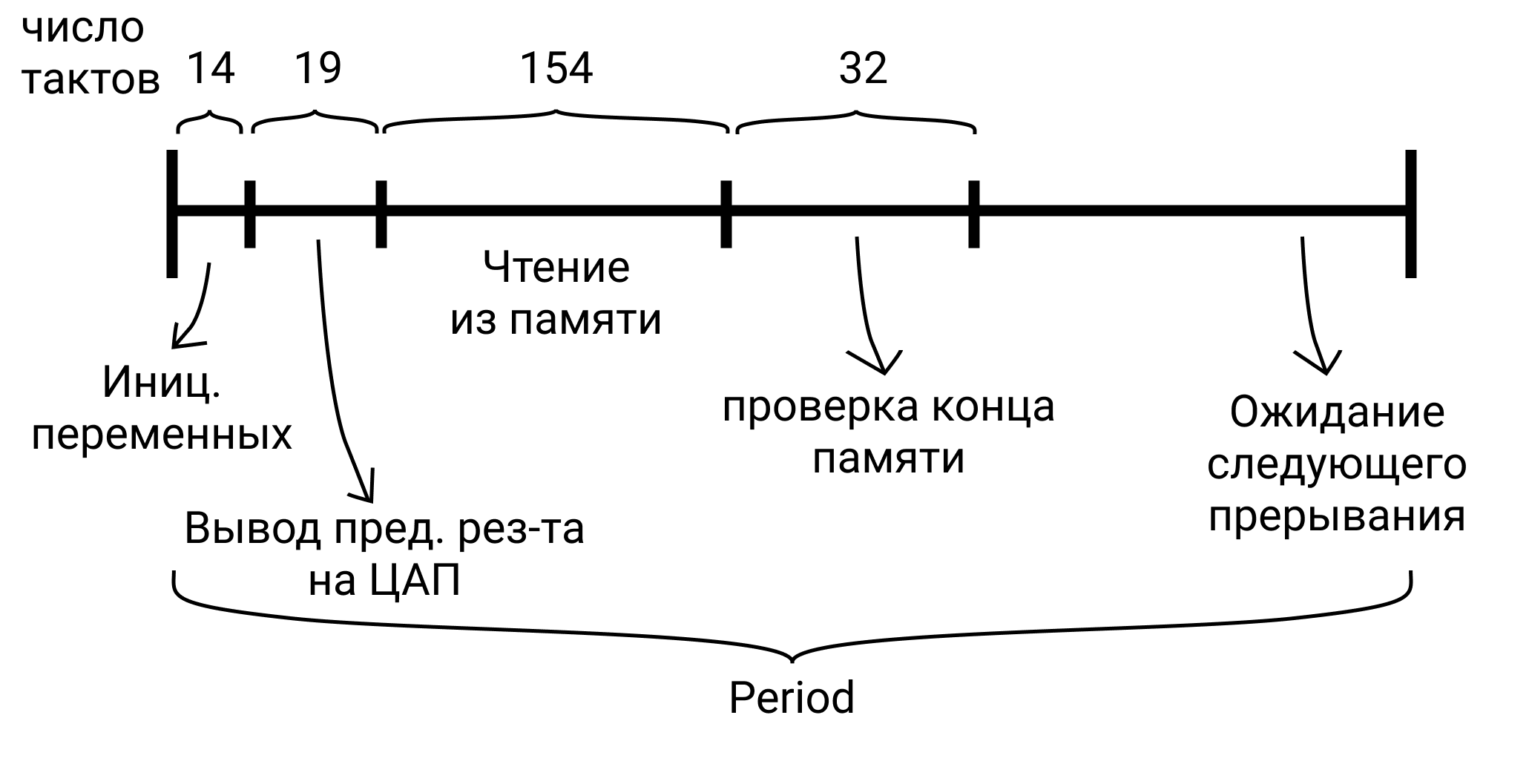


Рисунок 19 –Диаграмма выполнения вывода одного значения из памяти на ЦАП

Period на рисунках 18 и 19 означает количество отсчетов таймера перед выработкой прерывания. Таким образом, при частоте дискретизации в 10КГц период будет составлять 2000 тактов, а длительность записи одного трека будет составлять: . Таким образом на основании диаграмм можно подсчитать предельное значение частоты дискретизации на заданной частоте тактирования процессора (20МГц):

Время выполнения подпрограммы записи одного значения в тактах:

Время выполнения подпрограммы воспроизведения одного значения в тактах:

Таким образом для обеспечения нормальной работы рекомендуется устанавливать Period не меньше 1800, такое значение позволяет получить частоту дискретизации:

Поскольку частотный диапазон человеческой речи составляет 300Гц – 3КГц, частоты дискретизации около 6КГц будет достаточно для точного сохранения голосового сигнала, максимальная частота дискретизации (11.1КГц) удовлетворяет данному требованию. Установим частоту дискретизации 10КГц.

# Способ программирования памяти программ

Программирование Flash-памяти микроконтроллера производится с помощью JTAG/SW программатора. В режиме отладки разрешается работа отладочного интерфейса JTAG/SW. При этом к микроконтроллеру может быть подключен JTAG/SW адаптер, с помощью которого программные средства разработки позволяют работать с микроконтроллером в отладочном режиме. Линии JTAG должны быть подтянуты к питанию сопротивлениями не менее 10К с учетом, чтобы эти подтяжки не влияли на работу системы. Для программирования и отладки в рамках реализации устройства речевого ввода-вывода можно использовать как интерфейс JTAG\_B (линии PD0 - PD4), так и JTAG\_A (линии PB0 - PB4). При этом используется USB JTAG адаптер JEM-ARM-V2 (из состава отладочного комплекта для микроконтроллера К1986ВЕ92QI).

После включения питания, если сняты сигналы сброса (POR и RESET), МК выполняет код из загрузочной области ПЗУ - BOOT ROM, где по сигналам переключателей MODE[2:0] определяется режим функционирования. В данном случае используется режим MODE[2:0] = 000, то есть процессор переходит на выполнение программы из Flash (EEPROM) памяти программ (по адресу 0x08000000).

Схема подключения при отладке представлена на рисунке 20.

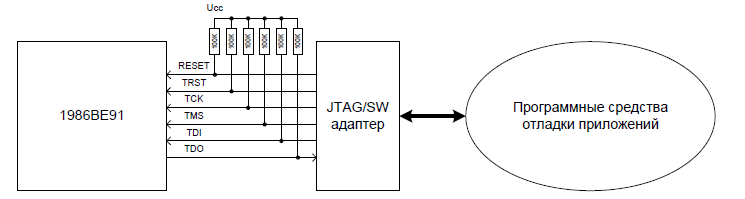


Рисунок 20 - Схема подключения в режиме отладки

В отладочном режиме среда разработки позволяет:

* стирать, записывать, считывать внутреннюю Flash-память программ,
* считывать и записывать содержимое ОЗУ, периферии;
* останавливать программу по точкам останова,
* выполнять программу в пошаговом режиме,
* запускать программу в нормальном режиме,
* проводить трассировку хода выполнения программного обеспечения,
* просматривать переменные выполняемой программы.

# ЗАКЛЮЧЕНИЕ

В рамках выполнения данного проекта были изучены и использованы следующие блоки микроконтроллера К1986ВЕ92QI для реализации устройства речевого ввода-вывода: АЦП, ЦАП, 16-разрыдне таймеры, EEPROM. Также была рассмотрена отладочная плата, написана основная программа, реализовано ручное тестирование системы в целом, построена диаграмма состояний интерфейса. Тестирование показало, что программа работает верно.

Для разработки и отладки программ, а также для программирования микроконтроллера использовалась среда разработки Keil uVision 5 и программатор JTAG/SW.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация на усилитель MAX9812 [Электронный ресурс]. – URL: https://datasheets.maximintegrated.com/en/ds/3859.pdf (дата обращения 2.11.2019).
2. Документация на МК серии 1986ВЕ9х [Электронный ресурс]. – URL:  [https://ic.milandr.ru/upload/iblock/214/2140b755c6493c4c80e575698d642549.pdf](http://www.elec.uow.edu.au/avr/avr_documentation/STK500_Schematics.pdf) (дата обращения 5.10.2019).
3. Хартов, В.Я. Микропроцессорные системы: учебник для студ. технических ВУЗов. – Издательство: Академия, 2014. – 368с.
4. Оцифровка звука на STM32 (АЦП+DMA) и кодирование в Speex для передачи [Электронный ресурс]. – URL: <https://m.habr.com/ru/post/323598/> (дата обращения 30.09.2019).
5. Работа с флеш памятью (MDR\_EEPORM) [Электронный ресурс]. – URL: http://forum.milandr.ru/viewtopic.php?t=624 (дата обращения 1.12.2019).
6. Программирование микроконтроллера K1986BE92QI компании «Миландр» [Электронный ресурс]. – URL: https://edu.milandr.ru/upload/iblock/8bd/8bda469e07e2dd755e1880d3543bf613.pdf (дата обращения 15.11.2019).

# ПРИЛОЖЕНИЕ А – Спецификация

Лист 1

# ПРИЛОЖЕНИЕ Б – текст программы

/\* Includes ------------------------------------------------------------------\*/

#include "MDR32Fx.h"

#include "MDR32F9Qx\_eeprom.h"  //библиотека для работы с EEPROM

#include "MDR32F9Qx\_port.h" //библиотека для работы с портами

#include "MDR32F9Qx\_rst\_clk.h" //библиотека для тактирования

#include "MDR32F9Qx\_adc.h"   //библиотека для работы с АЦП

#include "MDR32F9Qx\_dac.h"   //библиотека для работы с ЦАП

#include "MDR32F9Qx\_timer.h"   // библиотека для работы с таймером

#define EEPROM\_PAGE\_SIZE (4 \* 1024) //1 страница памяти - 4К

#define MAIN\_EEPAGE 6

#define TRACK\_MEM\_BYTES 53248 // свободная память для одного трека

#define PEGES 26              //всего доступно страниц для записи

#define CPU\_FREQ 20000000    //20МГц

int global\_byte\_counter = 4;       //начиная с 4 бита

uint8\_t global\_adc\_result = 0x08;  //результат преобразования на АЦП

uint16\_t global\_dac\_result = 0x80; //результат преобразования на ЦАП

int global\_current\_track = 0;     // текущий трек

//===============================================================================

//====================  ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ И ПРОЦЕДУРЫ  ====================

//===============================================================================

/\* Процедура задержки \*/

void Delay(int num)

{

    volatile uint32\_t i = 0;

    for (i = 0; i < num; i++)

    {

    }

}

/\* функция вычисления сдвига для записи трека в память \*/

int track\_offset(void)

{

    return global\_current\_track \* TRACK\_MEM\_BYTES;

}

/\* Функция преобразования частоты дискретизации в число тактов отсчета таймера \*/

int freq\_to\_tact(int freq\_dis)

{

    return (int)((1 / (double)freq\_dis) / (1 / (double)CPU\_FREQ));

}

/\* функция считывания результата преобразования.

После считывания флаг ADC1\_STATUS->Flg\_REG\_EOCIF будет сброшен

следующее преобразование начнется автоматически через 2 такта\*/

uint16\_t ADC\_Receive\_Word()

{

    uint16\_t result;

    ADC1\_Start();

    while (ADC1\_GetFlagStatus(ADC1\_FLAG\_END\_OF\_CONVERSION) == 0)

        ;

    result = ADC1\_GetResult() & 0x00000FFF; //получение 12-разрядного результата

    return result;

}

/\* Функция определения статуа (нажата или нет) кнопок (на пульте оператора)\*/

//btn\_name = 0 - кнопка SELECT (Очистка памяти)

//btn\_name = 1 - кнопка UP

//btn\_name = 2 - кнопка RIGHT (Запись трека)

//btn\_name = 3 - кнопка LEFT (Воспроизведение трека)

//btn\_name = 4 - кнопка DOWN

int current\_btn\_status(int btn\_name)

{

    int status = 0xA;

    //SELECT

    if (btn\_name == 0)

    {

        status = PORT\_ReadInputDataBit(MDR\_PORTC, PORT\_Pin\_2);

    }

    //UP

    if (btn\_name == 1)

    {

        status = PORT\_ReadInputDataBit(MDR\_PORTB, PORT\_Pin\_5);

    }

    //RIGHT

    if (btn\_name == 2)

    {

        status = PORT\_ReadInputDataBit(MDR\_PORTB, PORT\_Pin\_6);

    }

    //LEFT

    if (btn\_name == 3)

    {

        status = PORT\_ReadInputDataBit(MDR\_PORTE, PORT\_Pin\_3);

    }

    //DOWN

    if (btn\_name == 4)

    {

        status = PORT\_ReadInputDataBit(MDR\_PORTE, PORT\_Pin\_1);

    }

    //Инвертирование результата (1 - нажата, 0 - не нажата)

    if (status == 1)

    {

        return 0;

    }

    if (status == 0)

    {

        return 1;

    }

    return 0xA;

}

//===============================================================================

//=============  ПРОЦЕДУРЫ ИНИЦИАЛИЗАЦИИ ТАЙМЕРОВ И ОБРАБОТЧИКИ ПРЕР.  ==========

//===============================================================================

/\* Процедура настройки таймера 1 \*/

void Timer1\_init(int freq)

{

    int period = freq\_to\_tact(freq);

    //Включение тактирования

    RST\_CLK\_PCLKcmd(RST\_CLK\_PCLK\_TIMER1, ENABLE);

    //Инициализация структур

    TIMER\_CntInitTypeDef timerCnt;

    //Установка настроек по умолчанию

    TIMER\_CntStructInit(&timerCnt);

    //Установка предделителя

    TIMER\_BRGInit(MDR\_TIMER1, TIMER\_HCLKdiv1);

    //Установка числа отсчетов до прерывания

    timerCnt.TIMER\_Period = period;

    //Применение настроек

    TIMER\_CntInit(MDR\_TIMER1, &timerCnt);

    //Установка наибольшего приоритета

    NVIC\_SetPriority(Timer1\_IRQn, 0);

    //Установка прерывания по окончанию счета таймера (достижения 0)

    TIMER\_ITConfig(MDR\_TIMER1, TIMER\_STATUS\_CNT\_ZERO, ENABLE);

    //Разрешение прерываний

    \_\_enable\_irq();

    //Разрешение прерываний

    NVIC\_EnableIRQ(Timer1\_IRQn);

    NVIC\_ClearPendingIRQ(Timer1\_IRQn);

}

/\* Обработчик прерывания таймера 1 \*/

void Timer1\_IRQHandler()

{

    uint16\_t Data;

    uint32\_t Address = 0;

    uint32\_t BankSelector = 0;

    Address = 0x08000000 + EEPROM\_PAGE\_SIZE \* MAIN\_EEPAGE + track\_offset();

    BankSelector = EEPROM\_Main\_Bank\_Select;

    if (TIMER\_GetITStatus(MDR\_TIMER1, TIMER\_STATUS\_CNT\_ZERO))

    {

        TIMER\_ClearITPendingBit(MDR\_TIMER1, TIMER\_STATUS\_CNT\_ZERO);

        /\* Начало логики обработки прерывания\*/

        //Запуск преобразования и получение нового результата

        Data = ADC\_Receive\_Word();

        //отбрасывание младших 4 разрядов

        global\_adc\_result = (uint8\_t)(Data >> 4);

        //запись очередного значения в память (1514 тактов)

        EEPROM\_ProgramByte(Address + global\_byte\_counter, BankSelector, global\_adc\_result);

        //Инкремент счетчика бит

        global\_byte\_counter += 1;

        //Остановка таймера, если счетчик выходит на пределы доступной памяти

        if (global\_byte\_counter >= TRACK\_MEM\_BYTES)

        {

            TIMER\_Cmd(MDR\_TIMER1, DISABLE);

        }

        /\* Конец логики обработки прерывания\*/

    }

}

/\* Процедура настройки таймера 2 \*/

void Timer2\_init(int freq)

{

    int period = freq\_to\_tact(freq);

    //Включение тактирования

    RST\_CLK\_PCLKcmd(RST\_CLK\_PCLK\_TIMER2, ENABLE);

    //Инициализация структур

    TIMER\_CntInitTypeDef timerCnt;

    //Установка настроек по умолчанию

    TIMER\_CntStructInit(&timerCnt);

    //Установка предделителя

    TIMER\_BRGInit(MDR\_TIMER2, TIMER\_HCLKdiv1);

    //Установка числа отсчетов до прерывания

    timerCnt.TIMER\_Period = period;

    //Применение настроек

    TIMER\_CntInit(MDR\_TIMER2, &timerCnt);

    //Установка наибольшего приоритета

    NVIC\_SetPriority(Timer2\_IRQn, 0);

    //Установка прерывания по окончанию счета таймера (достижения 0)

    TIMER\_ITConfig(MDR\_TIMER2, TIMER\_STATUS\_CNT\_ZERO, ENABLE);

    //Разрешение прерываний

    \_\_enable\_irq();

    //Разрешение прерываний

    NVIC\_EnableIRQ(Timer2\_IRQn);

    NVIC\_ClearPendingIRQ(Timer2\_IRQn);

}

/\* Обработчик прерывания таймера 2 \*/

void Timer2\_IRQHandler()

{

    uint16\_t Data;

    uint8\_t Data\_8bit;

    uint32\_t Address = 0;

    uint32\_t BankSelector = 0;

    Address = 0x08000000 + EEPROM\_PAGE\_SIZE \* MAIN\_EEPAGE + track\_offset();

    BankSelector = EEPROM\_Main\_Bank\_Select;

    if (TIMER\_GetITStatus(MDR\_TIMER2, TIMER\_STATUS\_CNT\_ZERO))

    {

        TIMER\_ClearITPendingBit(MDR\_TIMER2, TIMER\_STATUS\_CNT\_ZERO);

        /\* Начало логики обработки прерывания\*/

        //Вывод предыдущего значения на ЦАП

        DAC2\_SetData(global\_dac\_result);

        //Чтение очередного значения

        Data\_8bit = (EEPROM\_ReadByte(Address + global\_byte\_counter, BankSelector));

        Data = (((uint16\_t)Data\_8bit) << 4) + 0x7;

        global\_dac\_result = Data;

        //Инкремент счетчика бит

        global\_byte\_counter += 1;

        //Остановка таймера, если счетчик выходит на пределы доступной памяти

        if (global\_byte\_counter >= TRACK\_MEM\_BYTES)

        {

            TIMER\_Cmd(MDR\_TIMER2, DISABLE);

        }

        /\* Конец логики обработки прерывания\*/

    }

}

//===============================================================================

//=========  ФУНКЦИИ ЧТЕНИЯ, ЗАПИСИ, ОЧИСТКИ    =================================

//===============================================================================

//Процедура очистки памяти

void erise\_mem()

{

    uint32\_t Address = 0;

    uint32\_t BankSelector = 0;

    uint32\_t i = 0;

    Address = 0x08000000 + EEPROM\_PAGE\_SIZE \* MAIN\_EEPAGE;

    BankSelector = EEPROM\_Main\_Bank\_Select;

    if (global\_current\_track == 0)

    {

        //очистка трека 1 (13 странц)

        for (i = 0; i < 13; i++)

        {

            EEPROM\_ErasePage(Address + i \* EEPROM\_PAGE\_SIZE, BankSelector);

        }

    }

    if (global\_current\_track == 1)

    {

        //очистка трека 2 (13 странц)

        for (i = 13; i < 26; i++)

        {

            EEPROM\_ErasePage(Address + i \* EEPROM\_PAGE\_SIZE, BankSelector);

        }

    }

}

/\*Процедура считывания значения напряжения на АЦП и записи в память \*/

void write\_track()

{

    uint32\_t Address = 0;

    uint32\_t BankSelector = 0;

    Address = 0x08000000 + EEPROM\_PAGE\_SIZE \* MAIN\_EEPAGE + track\_offset();

    BankSelector = EEPROM\_Main\_Bank\_Select;

    //Запуск таймера

    TIMER\_Cmd(MDR\_TIMER1, ENABLE);

    //ожидание окончания записи

    while (global\_byte\_counter < TRACK\_MEM\_BYTES)

    {

    }

    global\_byte\_counter = 4;

    //создание метки

    EEPROM\_ProgramWord(Address, BankSelector, 0xABCDEFAB);

}

/\* Процедура считывания памяти и вывода на ЦАП (AUDIO) \*/

void read\_track()

{

    uint32\_t size\_in\_bytes = TRACK\_MEM\_BYTES;

    //Запуск таймера

    TIMER\_Cmd(MDR\_TIMER2, ENABLE);

    //Ожидание завершения воспроизведения

    while (global\_byte\_counter < size\_in\_bytes)

    {

    }

    global\_byte\_counter = 4;

}

/\* Функция проверки, записан ли трек под номером num или пуст \*/

int track\_is\_empty()

{

    uint32\_t Address = 0;

    uint32\_t BankSelector = 0;

    Address = 0x08000000 + EEPROM\_PAGE\_SIZE \* MAIN\_EEPAGE + track\_offset();

    BankSelector = EEPROM\_Main\_Bank\_Select;

    if (EEPROM\_ReadWord(Address, BankSelector) == 0xABCDEFAB)

    {

        return 0; //трек пока не записан

    }

    else

    {

        return 1; //трек записан

    }

}

//===============================================================================

//=========  ПРОЦЕДУРЫ ИНИЦИАЛИЗАЦИИ МОДУЛЕЙ  ===================================

//===============================================================================

//Настройка частоты тактирования процессора

void MY\_U\_RST\_Init(void)

{

    RST\_CLK\_PCLKcmd(RST\_CLK\_PCLK\_BKP, ENABLE);

    RST\_CLK\_HSEconfig(RST\_CLK\_HSE\_ON);

    while (RST\_CLK\_HSEstatus() != SUCCESS)

        ;

    //20 Мгц

    RST\_CLK\_CPU\_PLLconfig(RST\_CLK\_CPU\_PLLsrcHSEdiv2,

                          RST\_CLK\_CPU\_PLLmul5);

    RST\_CLK\_CPU\_PLLcmd(ENABLE);

    while (RST\_CLK\_CPU\_PLLstatus() != SUCCESS)

        ;

    RST\_CLK\_CPUclkPrescaler(RST\_CLK\_CPUclkDIV1);

    RST\_CLK\_CPU\_PLLuse(ENABLE);

    RST\_CLK\_CPUclkSelection(RST\_CLK\_CPUclkCPU\_C3);

}

/\* Настройка АЦП \*/

void MY\_ADC\_Init(void)

{

    RST\_CLK\_PCLKcmd(RST\_CLK\_PCLK\_ADC, ENABLE);

    ADC\_InitTypeDef ADC\_Nastroyka;

    ADC\_StructInit(&ADC\_Nastroyka);

    ADC\_Init(&ADC\_Nastroyka);

}

/\* Настройка АЦП1 \*/

void MY\_ADC\_1\_Init(void)

{

    ADCx\_InitTypeDef ADC\_1\_Nastroyka;

    ADCx\_StructInit(&ADC\_1\_Nastroyka);

    ADC\_1\_Nastroyka.ADC\_ChannelNumber = ADC\_CH\_ADC7; //channel 7 connected to BNC

    ADC1\_Init(&ADC\_1\_Nastroyka);

    ADC1\_Cmd(ENABLE);

}

/\* настройка ЦАП  \*/

void MY\_DAC2\_Init(void)

{

    //Настройка порта для вывода аналогового сигнала на AUDIO выход

    //Тактирование порта вывода - PORTE и ЦАП

    RST\_CLK\_PCLKcmd(RST\_CLK\_PCLK\_PORTE | RST\_CLK\_PCLK\_DAC, ENABLE);

    PORT\_InitTypeDef PORT\_InitStructure;

    PORT\_InitStructure.PORT\_Pin = PORT\_Pin\_0; //PE0

    PORT\_InitStructure.PORT\_OE = PORT\_OE\_OUT;

    PORT\_InitStructure.PORT\_MODE = PORT\_MODE\_ANALOG;

    PORT\_Init(MDR\_PORTE, &PORT\_InitStructure);

    DAC2\_Init(DAC2\_AVCC); //выбор опорного напряжения (3.3V)

    DAC2\_Cmd(ENABLE);    //включение АЦП

}

/\*  Настройка портов ввода-вывода для кнопок  \*/

void BUTTONS\_Init(void)

{

    RST\_CLK\_PCLKcmd(RST\_CLK\_PCLK\_PORTB | RST\_CLK\_PCLK\_PORTE | RST\_CLK\_PCLK\_PORTC, ENABLE); //taktirovanie porta B, C, E

    //Настройка PB5 (UP) и PB6 (RIGHT)

    PORT\_InitTypeDef Nastroyka\_b;

    PORT\_StructInit(&Nastroyka\_b);

    Nastroyka\_b.PORT\_Pin = PORT\_Pin\_5 | PORT\_Pin\_6;

    Nastroyka\_b.PORT\_MODE = PORT\_MODE\_DIGITAL;

    Nastroyka\_b.PORT\_OE = PORT\_OE\_IN;

    Nastroyka\_b.PORT\_SPEED = PORT\_SPEED\_SLOW;

    Nastroyka\_b.PORT\_PD = PORT\_PD\_DRIVER;

    PORT\_Init(MDR\_PORTB, &Nastroyka\_b);

    //Настройка PC2 (SELECT)

    PORT\_InitTypeDef Nastroyka\_c;

    PORT\_StructInit(&Nastroyka\_c);

    Nastroyka\_c.PORT\_Pin = PORT\_Pin\_2;

    Nastroyka\_c.PORT\_MODE = PORT\_MODE\_DIGITAL;

    Nastroyka\_c.PORT\_OE = PORT\_OE\_IN;

    Nastroyka\_c.PORT\_SPEED = PORT\_SPEED\_SLOW;

    Nastroyka\_c.PORT\_PD = PORT\_PD\_DRIVER;

    PORT\_Init(MDR\_PORTC, &Nastroyka\_c);

    //Настройка PE3 (LEFT) и PE1 (DOWN)

    PORT\_InitTypeDef Nastroyka\_e;

    PORT\_StructInit(&Nastroyka\_e);

    Nastroyka\_e.PORT\_Pin = PORT\_Pin\_3 | PORT\_Pin\_1;

    Nastroyka\_e.PORT\_MODE = PORT\_MODE\_DIGITAL;

    Nastroyka\_e.PORT\_OE = PORT\_OE\_IN;

    Nastroyka\_e.PORT\_SPEED = PORT\_SPEED\_SLOW;

    Nastroyka\_e.PORT\_PD = PORT\_PD\_DRIVER;

    PORT\_Init(MDR\_PORTE, &Nastroyka\_e);

}

/\* Главная функция \*/

int32\_t main(void)

{

    /\* Список треков \*/

    char track\_array[2][32] = {"       \xD2\xF0\xE5\xEA 1", "       \xD2\xF0\xE5\xEA 2"};

    /\* количество треков \*/

    int num\_of\_tracks = 2;

    /\* определение выводимых на экран строк (кириллица) \*/

    char ochistka[] = "\xCE\xF7\xE8\xF1\xF2\xEA\xE0";

    char steret[] = "SELECT:\xD1\xF2\xE5\xF0\xE5\xF2\xFC";

    char pamyati[] = "\xEF\xE0\xEC\xFF\xF2\xE8...";

    char zapis[] = "\xC7\xE0\xEF\xE8\xF1\xFC...";

    char zapis\_short[] = "RIGHT :\xC7\xE0\xEF\xE8\xF1\xE0\xF2\xFC";

    char zapisan[] = "\xD3\xE6\xE5 \xC7\xE0\xEF\xE8\xF1\xE0\xED";

    char vosproizvedenie[] = "\xC2\xEE\xF1\xEF\xF0\xE8\xE7\xE2\xE5\xE4\xE5\xED\xE8\xE5...";

    char vosproizvedenie\_short[] = "LEFT  :\xC2\xEE\xF1\xEF\xF0\xE8\xE7\xE2.";

    char pusto[] = "\xCF\xF3\xF1\xF2\xEE!";

    char smirnov[] = "\xD1\xEC\xE8\xF0\xED\xEE\xE2 \xC0.\xC0.";

    char iu673[] = "\xC8\xD3\x36-73";

    int freq\_dis = 10000; //Установка частоты дискретизации записи и воспроизведения (не больше 10к)

    /\* Включение тактирование EEPROM \*/

    RST\_CLK\_PCLKcmd(RST\_CLK\_PCLK\_EEPROM, ENABLE);

    /\* вызов инициализирующих функций \*/

    U\_MLT\_Init();

    BUTTONS\_Init();

    MY\_ADC\_Init();

    MY\_ADC\_1\_Init();

    MY\_DAC2\_Init();

    Timer1\_init(freq\_dis);

    Timer2\_init(freq\_dis);

    //Установка частоты тактирования

    MY\_U\_RST\_Init();

    /\* Вывод имени, фамилии и группы\*/

    U\_MLT\_Put\_String("", 0);

    U\_MLT\_Put\_String(smirnov, 1);

    U\_MLT\_Put\_String(iu673, 2);

    /\* переменные текущего статуса кнопок \*/

    int current\_status\_select = 0; //кнопка очистка памяти

    int current\_status\_right = 0;  //кнопка записи трека

    int current\_status\_left = 0;   //кнопка воспроизведения трека

    int current\_status\_up = 0;   //кнопка перехода к следующему треку

    int current\_status\_down = 0;   //кнопка перехода к предыдущему треку

    /\* Основной цикл проверки кнопок \*/

    while (1)

    {

        /\* проверка нажатия кнопок \*/

        //UP -> Переход к предыдущему треку

        if (current\_btn\_status(1) == 1 && current\_status\_up == 0)

        {

            //обработка нажатия (начало)

            current\_status\_up = 1;

            global\_current\_track += 1;

            if (global\_current\_track == num\_of\_tracks)

                global\_current\_track = 0;

            //обработка нажатия (конец)

        }

        current\_status\_up = current\_btn\_status(1);

        //DOWN -> переход к следующему треку

        if (current\_btn\_status(4) == 1 && current\_status\_down == 0)

        {

            //обработка нажатия (начало)

            current\_status\_down = 1;

            global\_current\_track -= 1;

            if (global\_current\_track == -1)

                global\_current\_track = num\_of\_tracks - 1;

            //обработка нажатия (конец)

        }

        current\_status\_down = current\_btn\_status(4);

        //SELECT -> Очистка памяти

        if (current\_btn\_status(0) == 1 && current\_status\_select == 0)

        {

            //обработка нажатия (начало)

            if (track\_is\_empty() == 1)

            {

                U\_MLT\_Put\_String(pusto, 4);

                U\_MLT\_Put\_String("", 5);

                U\_MLT\_Put\_String("", 6);

                U\_MLT\_Put\_String("", 7);

                Delay(5000000);

            }

            else

            {

                current\_status\_select = 1;

                U\_MLT\_Put\_String(ochistka, 4);

                U\_MLT\_Put\_String(pamyati, 5);

                U\_MLT\_Put\_String("", 6);

                U\_MLT\_Put\_String("", 7);

                erise\_mem();

            }

            //обработка нажатия (конец)

        }

        current\_status\_select = current\_btn\_status(0);

        //RIGHT -> Запись трека

        if (current\_btn\_status(2) == 1 && current\_status\_right == 0)

        {

            //обработка нажатия (начало)

            if (track\_is\_empty() == 1)

            {

                U\_MLT\_Put\_String(zapis, 4);

                U\_MLT\_Put\_String("", 5);

                U\_MLT\_Put\_String("", 6);

                U\_MLT\_Put\_String("", 7);

                write\_track();

            }

            else

            {

                U\_MLT\_Put\_String(zapisan, 4);

                U\_MLT\_Put\_String("", 5);

                U\_MLT\_Put\_String("", 6);

                U\_MLT\_Put\_String("", 7);

                Delay(5000000);

            }

            //обработка нажатия (конец)

        }

        current\_status\_right = current\_btn\_status(2);

        //LEFT -> Воспроизведение трека

        if (current\_btn\_status(3) == 1 && current\_status\_left == 0)

        {

            //обработка нажатия (начало)

            if (track\_is\_empty() == 1)

            {

                U\_MLT\_Put\_String(pusto, 4);

                U\_MLT\_Put\_String("", 5);

                U\_MLT\_Put\_String("", 6);

                U\_MLT\_Put\_String("", 7);

                Delay(5000000);

            }

            else

            {

                U\_MLT\_Put\_String(vosproizvedenie, 4);

                U\_MLT\_Put\_String("", 5);

                U\_MLT\_Put\_String("", 6);

                U\_MLT\_Put\_String("", 7);

                read\_track(); //воспроизведение трека

            }

            //обработка нажатия (конец)

        }

        current\_status\_left = current\_btn\_status(3);

        //Вывод селект-листа треков на экран

        U\_MLT\_Put\_String(track\_array[global\_current\_track], 3);

        U\_MLT\_Put\_String(zapis\_short, 4);

        U\_MLT\_Put\_String(steret, 5);

        U\_MLT\_Put\_String(vosproizvedenie\_short, 6);

        U\_MLT\_Put\_String("", 7);

        Delay(500);

    }

}