

Министерство образования и науки Российской Федерации
ФГАОУ ВО «Национальный исследовательский Томский
политехнический университет»
ФГАОУ ВО «Национальный исследовательский Томский
государственный университет»
ФГУИ Институт оптики атмосферы им. В.Е. Зуева Сибирского
отделения Российской академии

**С.Н. Торгаев, И.С. Мусоров,
А.А. Солдатов, П.В. Сорокин**

**ПРОГРАММИРОВАНИЕ
МИКРОКОНТРОЛЛЕРОВ С ЯДРОМ
CORTEX-M3 В ЗАДАЧАХ ДИАГНОСТИКИ
И КОНТРОЛЯ**

*Рекомендовано в качестве учебного пособия
Редакционно-издательским советом
Томского политехнического университета*

Scientific & Technical Translations

ИЗДАТЕЛЬСТВО
Томск – 2017

УДК 681.322 (075.8)
ББК 32.973.26-04я73
Т60

T60 Торгаев С.Н., Мусоров И.С., Солдатов А.А.,
Сорокин П.В. **Программирование микроконтроллеров с ядром Cortex-M3 в задачах диагностики и контроля** : учебное пособие. – Томск : STT, 2017. – 104 с.

ISBN 978-5-93629-598-0

Методическое пособие предназначено для студентов старших курсов, изучающих многоразрядные микроконтроллеры и построение систем неразрушающего контроля на их основе. В пособии рассмотрены вопросы программирования микроконтроллеров K1986BE92QI. Представлено большое количество примеров программ по настройке основных периферийных устройств для данных микроконтроллеров.

Пособие предназначено для студентов, обучающихся по направлениям 11.04.04 «Электроника и наноэлектроника», 12.04.04 “Биотехнические системы и технологии”.

УДК 681.322 (075.8)
ББК 32.973.26-04я73

Рецензенты:

- | | |
|-----------------|---|
| Нариманова Г.Н. | – канд. физ.-мат. наук, доцент кафедры УИ ТУСУР; |
| Шульгин Е.М. | – начальник отдела управления персоналом
АО «НПЦ "Полюс"». |

ISBN 978-5-93629-598-0

© С.Н. Торгаев, И.С. Мусоров,
А.А. Солдатов, П.В. Сорокин, 2017
© ФГАОУ ВО НИ ТПУ, 2017
© Оформление. STT™, 2017

ОГЛАВЛЕНИЕ

Введение	4
Глава 1. МИКРОКОНТРОЛЛЕР K1986BE92QI.	
НАСТРОЙКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.....	5
Описание процессорного ядра Cortex-M3	5
Основные характеристики микроконтроллера K1986be92qi	100
Описание отладочного макета	111
Настройка программы Iar Embedded Workbench	155
Создание проекта в программе Iar Embedded Workbench	177
Создание проема в программном продукте Keil Mvision 5.....	30
Глава 2. НАСТРОЙКА ПОРТОВ ВВОДА ВЫВОДА. ПРИМЕРЫ	411
Порты ввода-вывода Mdr_Portx.....	411
Пример программ настройки портов ввода-вывода	48
Лабораторная работа №1. Порты ввода-вывода	59
Глава 3. ТАЙМЕРЫ МИКРОКОНТРОЛЛЕРА K1986BE92QI.....	600
Описание таймеров микроконтроллера	600
Пример программ настройки таймера 1	644
Лабораторная работа №2. Таймеры	711
Глава 4. АНАЛОГОВО-ЦИФРОВЫЕ ПРЕОБРАЗОВАТЕЛИ	
МИКРОКОНТРОЛЛЕРА K1986BE92QI	733
Описание АЦП микроконтроллера	733
Пример программ настройки АЦП	777
Лабораторная работа №3. Аналогово-цифровой преобразователь	866
Глава 5. ЦИФРО-АНАЛОГОВЫЙ ПРЕОБРАЗОВАТЕЛЬ	
МИКРОКОНТРОЛЛЕРА K1986BE92Qi.....	88
Описание ЦАП микроконтроллера	88
Пример программ настройки ЦАП	889
Лабораторная работа №3. Цифро-аналоговый преобразователь	900
Глава 6. ИНТЕРФЕЙС UART МИКРОКОНТРОЛЛЕРА K1986BE92QI	922
Описание ЦАП микроконтроллера	922
Пример программ настройки Uart	933
Лабораторная работа №3. Интерфейс Uart	98
Литература	99

Введение

В настоящее время особой популярностью у разработчиков электронных устройств пользуются так называемые встраиваемые микроконтроллеры. В мире выпускается большое количество семейств микроконтроллеров, в основном на базе приборов с 8-битной шиной данных и процессорными RISC- и CISC-ядрами. Их производительность и объем памяти вполне достаточны для решения множества бытовых и промышленных задач, а архитектура ядер многих из этих семейств хорошо изучена пользователями. Для 8-битных микроконтроллеров написано огромное количество программ, которые, наряду с многочисленными русифицированными описаниями самих этих устройств, свободно доступны в сети Интернет.

16- и 32-битные микроконтроллеры, которые обеспечивают более высокую производительность, быстро набирают популярность. Их применение обусловлено повышенной сложностью решаемых задач, жесткими требованиями к производительности встраиваемых контроллеров управления, необходимостью иметь в электронных устройствах развитые пользовательские интерфейсы, предназначенные для отображения информации, управления, индикации и т.д. Типичные для 16- и 32-битных микроконтроллеров приложения (сотовые телефоны, дисководы, модемы и т.п.) предъявляют к встраиваемым управляющим контроллерам непрерывно возрастающие требования. Особенно важно обеспечить их высокую эффективность, сохранив при этом низкую стоимость, отличавшую 8-битные микроконтроллеры.

Значительных успехов в области создания 16/32-битных микропроцессорных (микроконтроллерных) ядер добилась британская фирма Advanced RISC Machines (ARM), специализирующаяся на разработке микропроцессоров и периферии к ним, и продающая лицензии на их производство другим фирмам-производителям. 32-битные микроконтроллеры, использующие процессорное ядро с архитектурой ARM, приобрели широкую популярность у разработчиков. Благодаря высокой производительности и выгодному соотношению «цена/качество», микроконтроллеры с ядром ARM представляются многим потребителям элементной базы весьма перспективными.

Данное учебное пособие посвящено вопросам настройки периферийных устройств микроконтроллера *K1986BE92QI* и его программированию. В пособии представлено большое количество примеров программ для микроконтроллера, написанных на языке *C*.

Глава 1. МИКРОКОНТРОЛЛЕР K1986BE92QI. НАСТРОЙКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Описание процессорного ядра Cortex-M3

Cortex-M3 – это новое 32-разрядное ARM RISC ядро с гарвардской архитектурой. Микроконтроллеры на основе этого ядра выпускаются многими фирмами (ST Microelectronics, NXP, TI и др.), что позволяет разработчику выбрать наиболее подходящий микроконтроллер. Данные процессоры спроектированы для достижения высокой производительности всей системы в недорогих высокономичных встроенных приложениях, таких как системы промышленного контроля, автомобильная электроника, проводные и беспроводные телекоммуникационные системы, системы управления электроприводами и т.п.

Процессоры на основе архитектуры CortexM3 являются дальнейшим и наиболее прогрессивным развитием классической архитектуры ARM и имеют более высокую производительность, менее сложную модель программирования, прекрасную систему обработки прерываний и низкую цену. Некоторые преимущества процессоров на базе архитектуры Cortex-M3 перед процессорами на базе классической архитектуры приведены в таблице 1.1 [1].

Таблица 1.1
Сравнение ядер ARM7TDMI-S и Cortex-M3

Параметр	Ядро	
	ARM7TDMI-S	Cortex-M3
Ядро	ARMv4T	ARMv7-M
Архитектура	фон Неймана	Гарвардская
Поддерживаемые системы команд	Thumb	Thumb / Thumb-2
Прерывания	FIQ/IRQ	NMI + от 1 до 240 физических прерываний
Время обработки прерываний	24..42 тактов	12 тактов
Пошаговый режим	Нет	Интегрирован
Защита памяти	Нет	8 региональных устройств защиты памяти
Производительность	0,95 DMIPS/МГц (в режиме ARM)	1,25 DMIPS/МГц
Потребляемая мощность	0,28 мВт/МГц	0,19 мВт/МГц
Площадь кристалла, мм ²	0,62 (только ядро)	0,86 (ядро и стандартная периферия)

Процессор на базе архитектуры Cortex-M3 в своей основе имеет иерархическую структуру. Она включает в себя ядро CM3Core с развитой периферией, включающей в себя механизмы управления прерываниями, защиты памяти и внутрисхемной отладки и другие. Архитектура процессорного ядра Cortex-M3 и интегрированные в него компоненты представлены на рисунке 1.1.

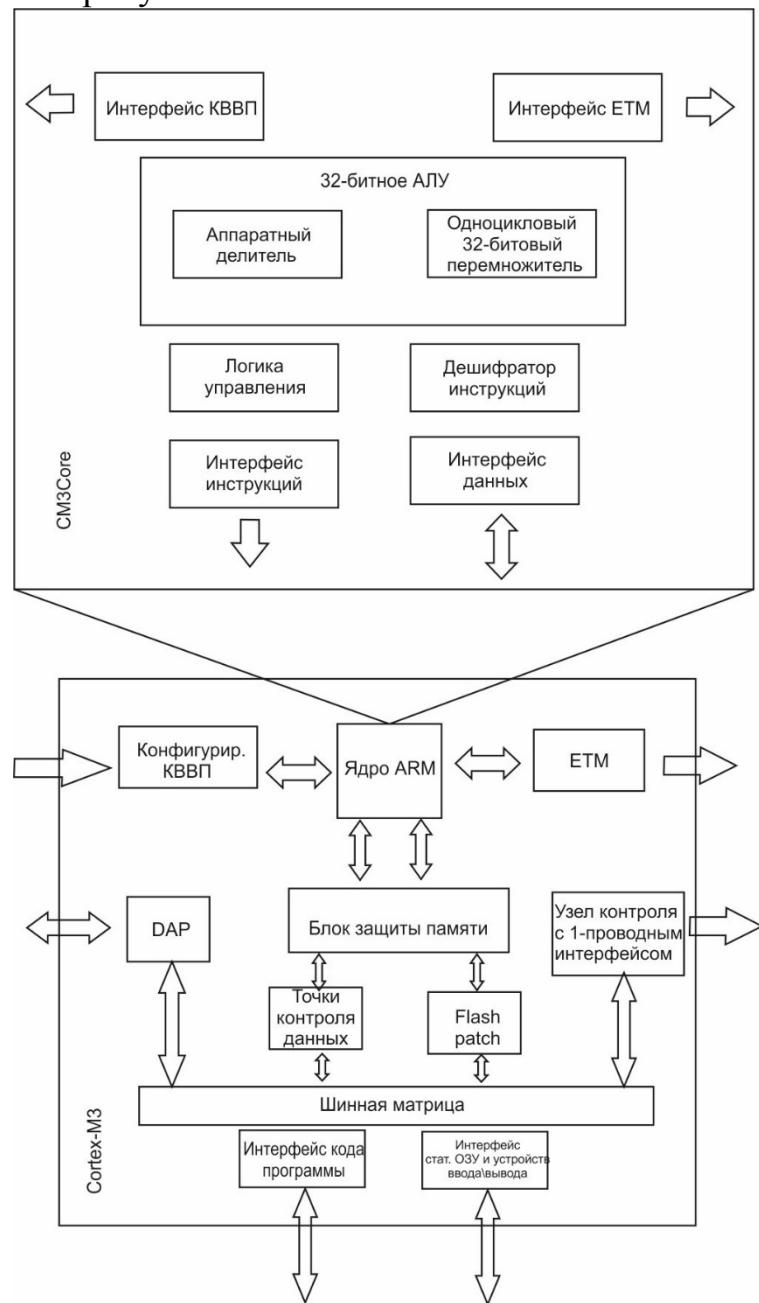


Рис. 1.1. Архитектура ядра Cortex-M3

Ядро Cortex-M3 базируется на Гарвардской архитектуре и имеет раздельные шины для команд и для данных в отличие от стандартных ARM процессоров, использующих фон Неймановскую архитектуру и совмещенные шину и память как для данных, так и для команд. Поскольку процессоры Cortex-M3 считывают данные и команды одновременно, это позволяет производить некоторые операции одновременно и таким образом обеспечить конвейерную обработку программ [3].

Ядро Cortex-M3 может работать в одном из двух режимов работы: «Thread» и «Handler», и поддерживает два уровня доступа к коду программы: привилегированный и непривилегированный, что облегчает реализацию сложных и открытых систем, не жертвуя при этом защищенностью системы. Код программы, исполняемый в непривилегированном режиме, имеет ограниченные возможности по доступу к некоторым ресурсам и специфическим областям памяти. Режим «Thread» является типичным режимом работы, в котором код программы может быть как привилегированным, так и непривилегированным. Переход в режим «Handler» происходит при возникновении исключительной ситуации (exception); в данном режиме весь код программы выполняется как привилегированный. Также предусматривается такое понятие, как рабочее состояние ядра. Их два: Thumb, в котором выполнение инструкций идет обычным путем, и Debug, в котором активизируются встроенные отладочные возможности ядра.

В отличие от прежних ядер ARM, в ядре Cortex-M3 стандартизовано не только ЦПУ, но и контроллер прерываний, системный таймер и карта памяти. Например, 4 Гбайта адресного пространства Cortex-M3 разделено на строго определенные области для кода приложения, SRAM, периферийных и системных устройств (рис. 1.2).

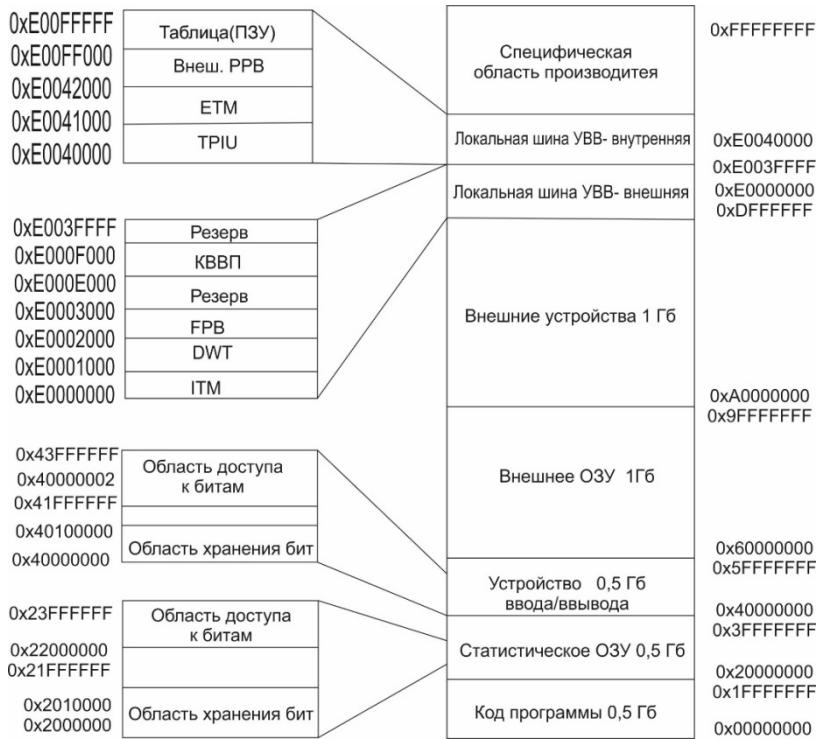


Рис.1.2. Адресное пространство процессора Cortex-M3

Традиционные процессоры на базе архитектуры ARM7 поддерживают только доступ к выровненным данным, что подразумевает, что сохраняемые и вычитываемые данные должны быть выровнены по границе слова. Процессоры же на базе архитектуры Cortex-M3 позволяют обращаться к невыровненным данным и сводят к минимуму временные задержки, связанные с доступом к данным. В случае, если происходит обращение к невыровненным данным, это обращение разбивается на несколько параллельных обращений к выровненным данным, но этот процесс является прозрачным для программы пользователя, поскольку происходит автоматически внутри ядра.

Помимо этого, процессоры с архитектурой Cortex-M3 поддерживают операции 32-разрядного умножения за 1 такт, а также знаковое и беззнаковое деление, которое требует от 2 до 12 тактов в зависимости от размера операндов. Операция деления происходит быстрее, в случае, если делимое и делитель имеют небольшой размер.

Cortex поддерживает одну систему команд, ARM Thumb-2. Эта система содержит и 16- и 32-битные команды, сочетая в себе производительность 32-битного ARM с плотностью кода 16-битного Thumb.

Процессор Cortex-M3 содержит более совершенную систему прерываний (относительно ARM7), характеризующуюся задержкой вызова процедуры обработки прерывания всего лишь 12 машинных циклов (для сравнения, ARM7TDMI-S требует 24–42 цикла). Основу системы

прерываний составляет Контроллер Вложенных Векторных Прерываний, который в стандартной реализации поддерживает одно немаскируемое прерывание и 32 прерывания общего назначения с 8 уровнями приоритетов (в общем случае число прерываний может достигать 240 при 256 уровнях приоритета). С момента получения прерывания до начала выполнения первой команды обработчика прерывания затрачивается только двенадцать циклов тактового сигнала. Это достигается частично за счет автоматического размещения информации в стеке, осуществляемого внутри ЦПУ микроконтроллера. В случае, когда прерывания следуют впритык друг за другом, КВВП использует метод «постановки в очередь», позволяющий осуществлять обработку каждого из последующих прерываний всего за шесть циклов тактового сигнала. На этапе обращения к стеку, прерывание с более высоким приоритетом может получить преимущество перед прерыванием с низким приоритетом, что не влечет за собой затрат дополнительных циклов ЦПУ.

Блок защиты памяти является опциональным компонентом ядра Cortex-M3. Он позволяет повысить надежность встраиваемых систем за счет защиты критичной информации, используемой операционной системой, от действия пользовательских программ.

Доступ к встроенной отладочной системе реализован посредством порта доступа к отладочной системе (DAP), который с внешней средой связывается по одному из последовательных интерфейсов: 2-выводной последовательный отладочный порт SW или стандартный 5-выводной последовательный порт JTAG. Появление 2-выводного интерфейса делает возможным появление 32-битных микроконтроллеров с числом выводов менее 10 и существенно упрощает электрическое подключение к отлаживаемому устройству. Для микроконтроллеров с масочным ПЗУ предусмотрен специальный блок Flash Patch, который во время отладки позволяет осуществлять выборку инструкций не из ПЗУ, а из статического ОЗУ, тем самым существенно упрощая процедуру отладки программного кода для таких микроконтроллеров.

Микроконтроллеры с архитектурой Cortex-M3 стали значительным явлением на рынке ARM микроконтроллеров. Ядро микроконтроллера, построенное на базе Гарвардской архитектуры с использованием 3-уровневого конвейера, использует новые решения, такие как предсказание переходов в командах ветвления, однотактное умножение и деление, показывает впечатляющий уровень производительности, равный 1,25 DMIPS/МГц.

Немаловажным плюсом архитектуры Cortex-M3 является широкая ее поддержка ведущими производителями программного обеспечения. В таблице 3 перечислены основные производители программных и про-

граммно-аппаратных комплексов для создания и отладки программного кода для микроконтроллеров с архитектурой Cortex-M3.

Таблица 1.2

Некоторые производители программных продуктов для ARM MCU Cortex-M3

Производитель	Продукты
Keil Software	Полный комплекс разработки и отладки программного кода, а также аппаратные средства программирования и отладки
IAR Systems	Полный программный комплекс разработки и отладки программного кода
CodeSousery	Полнофункциональный комплекс разработки и отладки GNU G++
Rowley Associates	Полнофункциональный пакет разработки и отладки CrossWorks
FreeRTOS.org	Встраиваемые операционные системы реального времени RTOS
Pumpkin	RTOS
Express Logic	RTOS, TCP/IP стеки, файловые системы, USB стеки
Micrium	RTOS, TCP/IP стеки, файловые системы, USB стеки, библиотеки для работы с протоколами CAN и Modbus
CMX Systems	RTOS, TCP/IP стеки, файловые системы, USB стеки
SEGGER Microcontroller Systeme	RTOS, GUI, файловые системы, USB стеки, JTAG эмуляторы
Interniche Technologies	RTOS, сетевые стеки и файловые системы

Основные характеристики микроконтроллера K1986BE92QI

Ядро:

- ARM 32-битное RISC-ядро Cortex™-M3, тактовая частота до 80 МГц;
- производительность 1,25 DMIPS/МГц;

- блок аппаратной защиты памяти MPU;
- умножение за один цикл, аппаратная реализация деления.

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.

Питание и тактовая частота:

- внешнее питание $2,2 \div 3,6$ В;
- встроенный регулируемый стабилизатор напряжения на 1,8 В для питания ядра;
- встроенные схемы контроля питания;
- встроенные подстраиваемые RC генераторы 8 МГц и 40 кГц;
- внешние кварцевые резонаторы на $2 \div 16$ МГц и 32 кГц;
- встроенный умножитель тактовой частоты PLL для ядра;
- встроенный умножитель тактовой частоты PLL для USB.

Аналоговые модули:

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

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.

Отладочные интерфейсы:

- последовательные интерфейсы SWD и JTAG.

Описание отладочного макета

На рисунке 1.3 представлен внешний вид отладочного макета микроконтроллера K1986BE92QI.

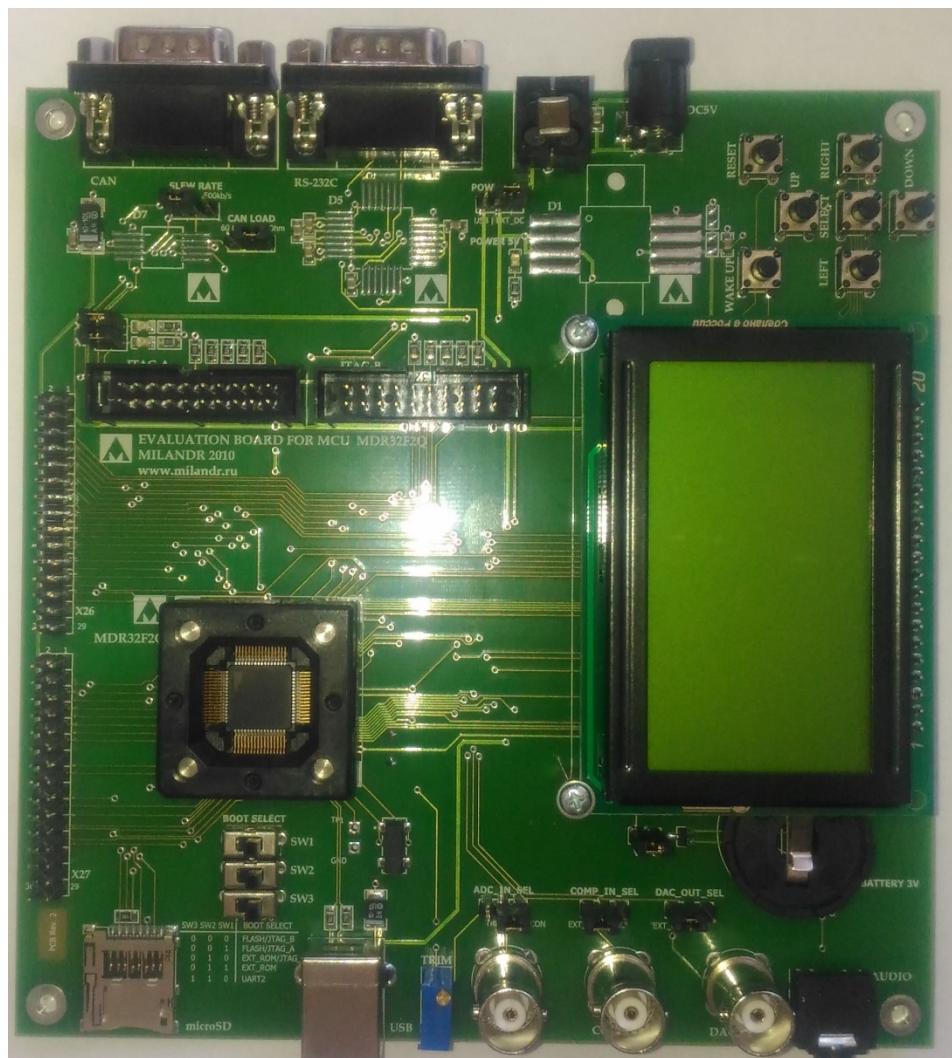


Рис. 1.3. Внешний вид отладочного макета

Модуль Блок коммутации включает в себя 7 кнопок – RESET, WAKEUP, UP, DOWN, LEFT, RIGHT и SELECT. Схемы подключения кнопок к соответствующим портам микроконтроллера представлены на рисунке 1.4.

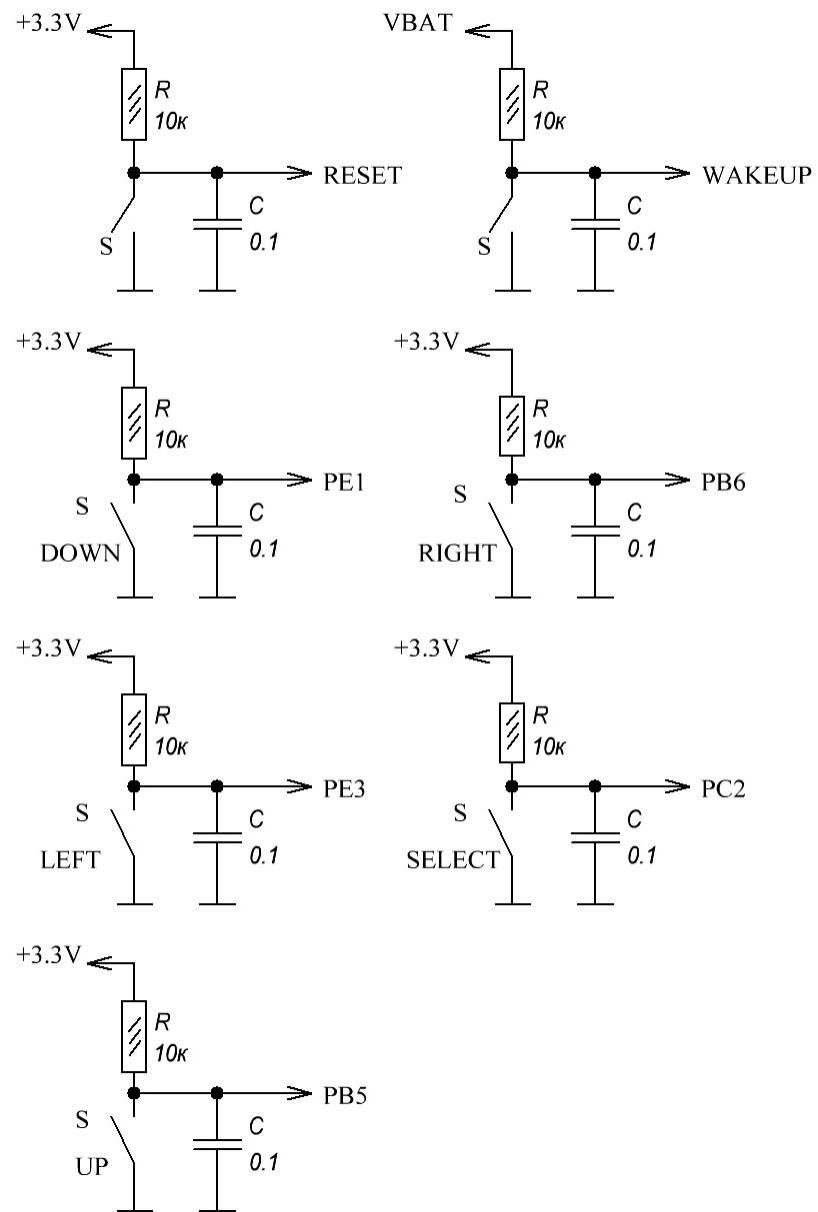


Рис. 1.4. Схемы подключения кнопок отладочной платы

На отладочном макете имеется два пользовательских светодиода, подключенные к порту С (рис. 1.5).

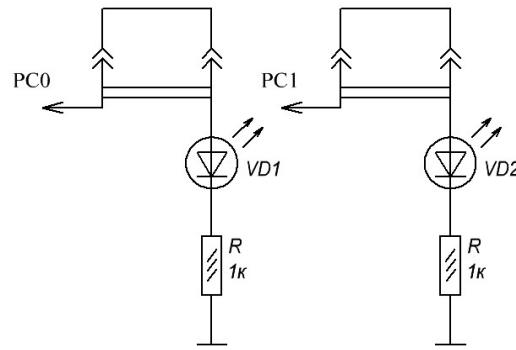


Рис. 1.5. Схемы подключения светодиодов на отладочной плате

Для подключения внешних устройств на плате присутствуют 2 разъема с выводами на них линий портов микроконтроллера (рис. 1.6).

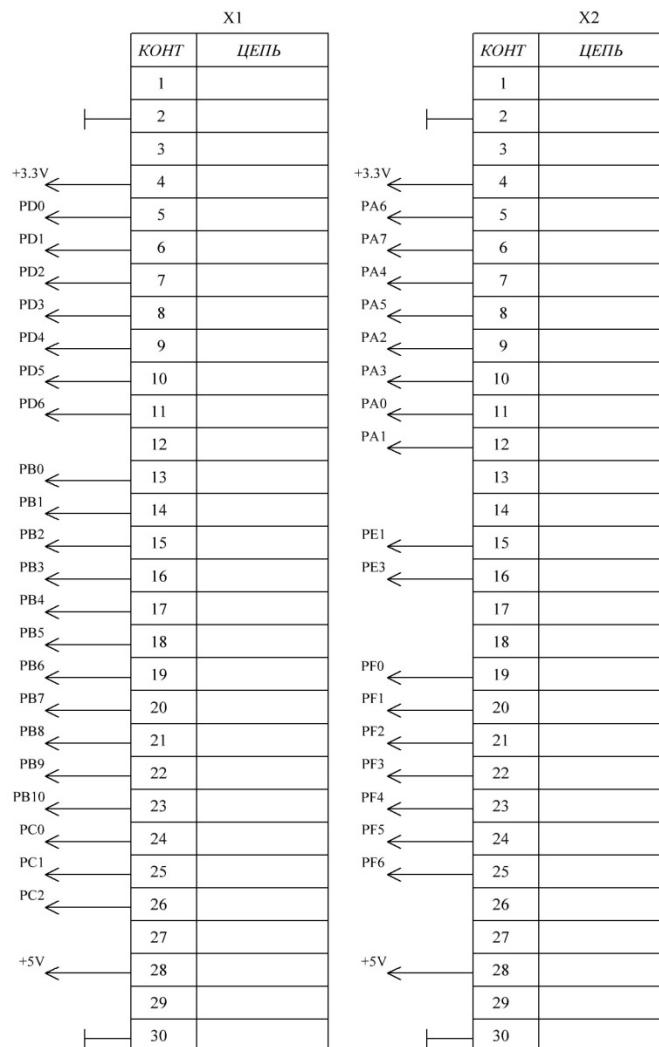


Рис. 1.6. Схема подключения разъемов

Также на отладочной плате присутствует разъем для подключения microSD карточки (рис. 1.7).

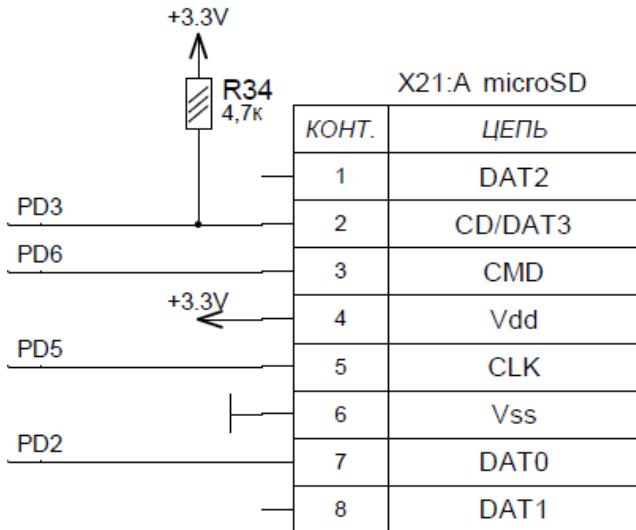


Рис. 1.7. Схема подключения разъемов

Настройка программы IAR Embedded Workbench

Архив iar_arm.rar содержит папку arm, эту папку надо скопировать в папку <Где установлен IAR>. Там уже есть такая папка, т.е. просто должны появиться дополнительные файлы и папки в ней.

/arm/src/flashloadert/Milandr – исходники загрузчика флеш

/arm/inc/Milandr – заголовочные файлы для 1986BE9x

/examples/Milandr/coremark_iar – пример, программа CoreMark

/config/linker/Milandr – настройки линкера

/config/flashloader/Milandr – скомпилированный загрузчик с настройками

/config/devices/Milandr – собственно, само описание МК для IAR

/config/debugger/Milandr – файл с описанием периферии МК для отладки

1 Настройки IAR EWB

В настройках проекта необходимо произвести настройку параметров согласно рис. 1.8–1.10.

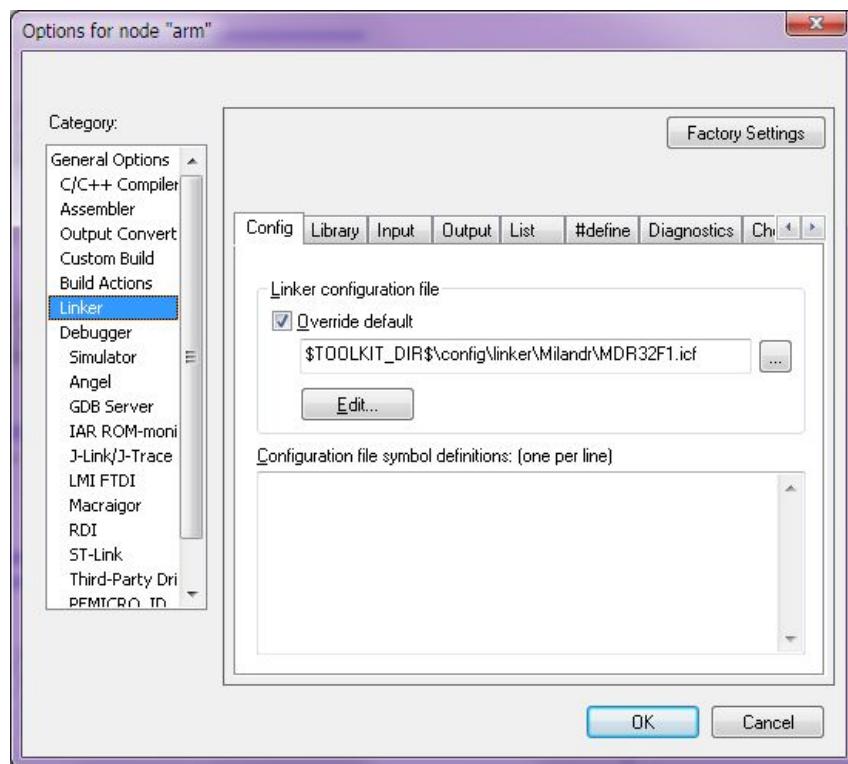


Рисунок 1.8. Настройки линкера

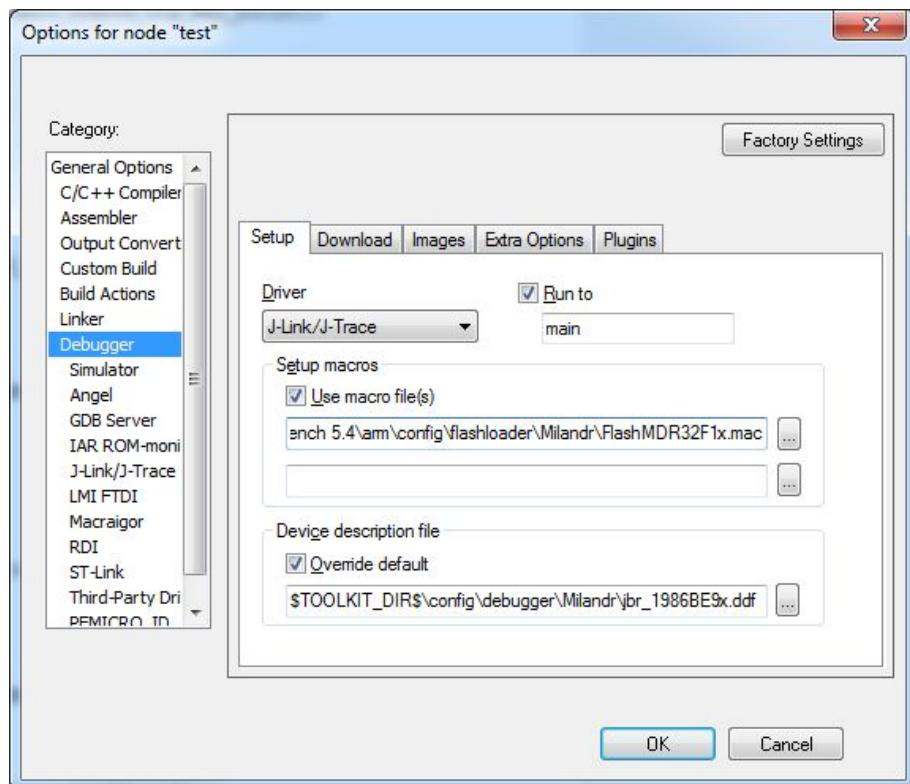


Рисунок 1.9. Настройки отладчика

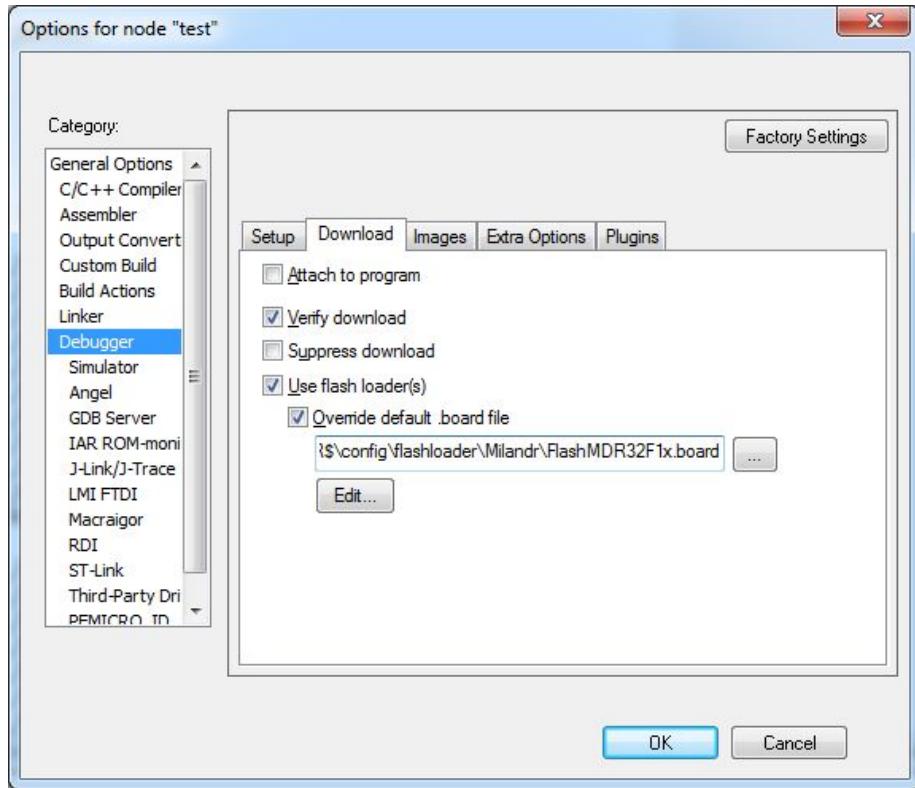


Рисунок 1.10. Настройки отладчика (программирование)

Создание проекта в программе *IAR Embedded Workbench*

Создание проекта в среде IAR Embedded Workbench осуществляется по следующему алгоритму.

1. Запускаем среду программирования IAR Embedded Workbench for ARM. На рисунке 1.11 представлен внешний вид стартового окна программы.

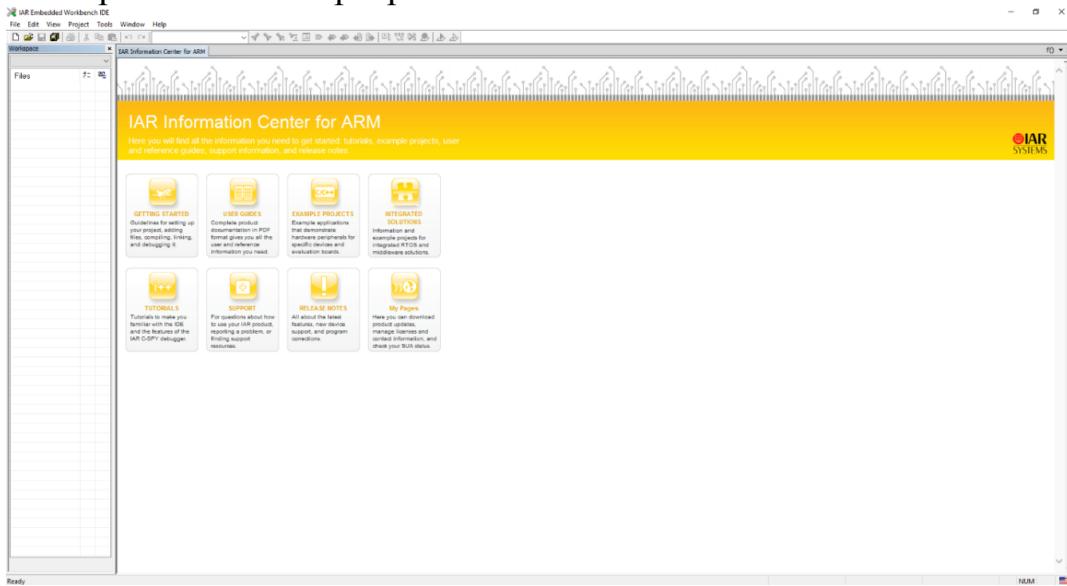


Рис.1.11. Стартовое окно программы

2. Для создания нового проекта необходимо зайти в меню «*Project*» и выбрать пункт «*Create new project...*» (рис. 1.12).

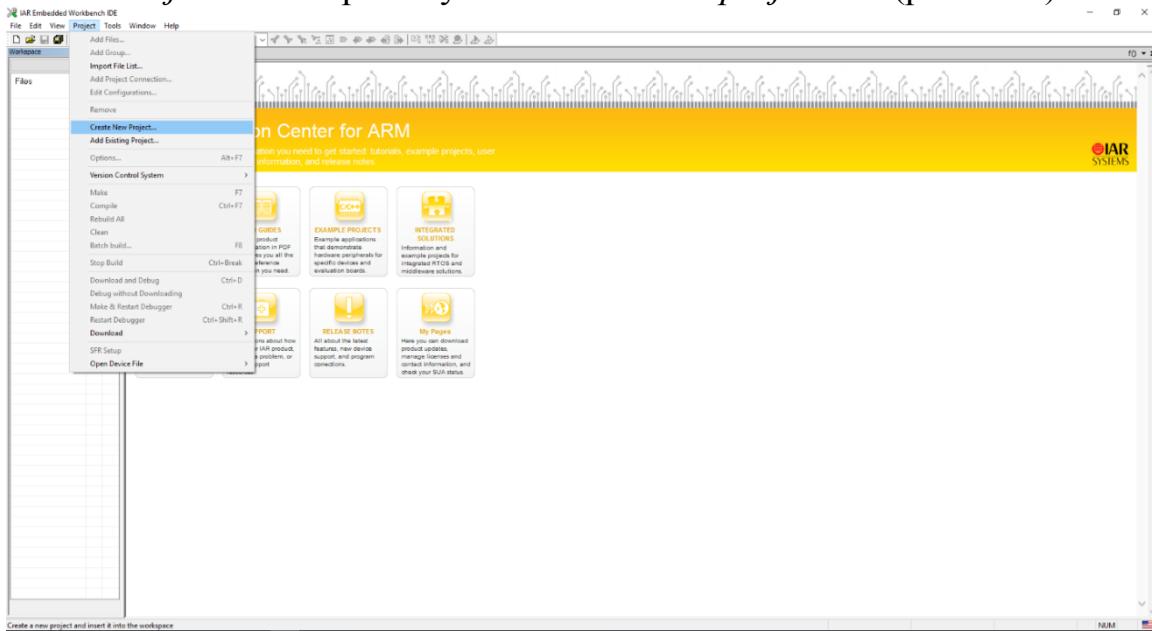


Рис. 1.12. Окно создания нового проекта

3. В появившемся окне (рис. 1.13) необходимо выбрать шаблон для языка С и тип микроконтроллера (ARM). Далее следует сохранить рабочую область *Workspace* (рис. 1.14).

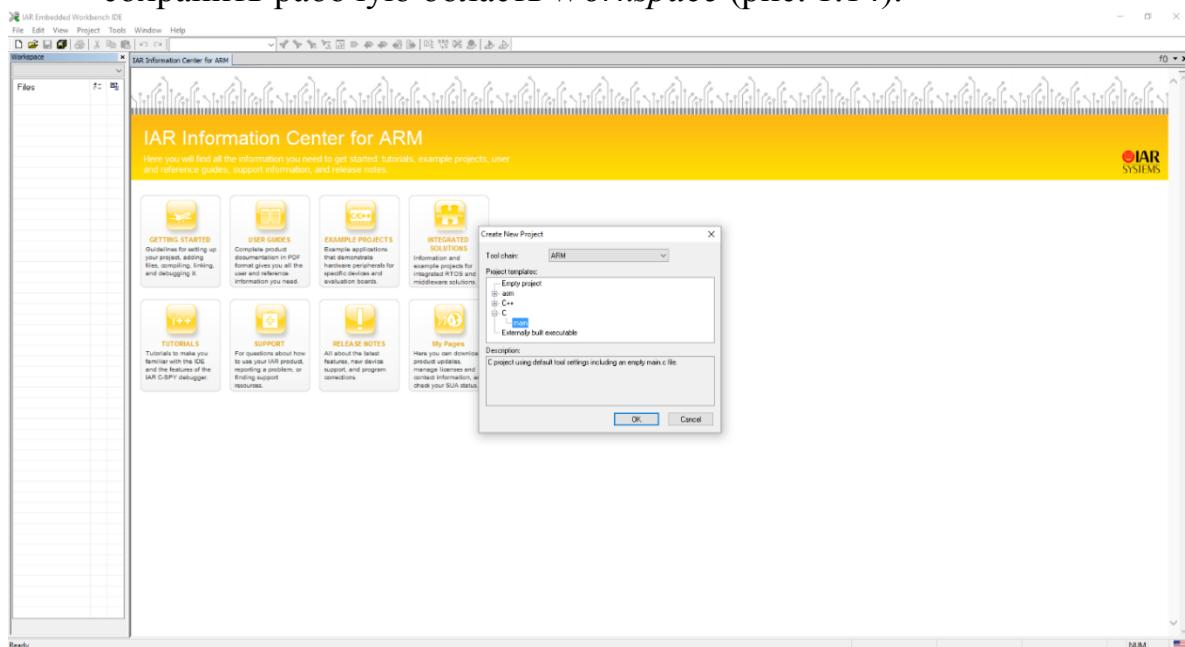


Рис. 1.13. Окно выбора языка программирования и микроконтроллера

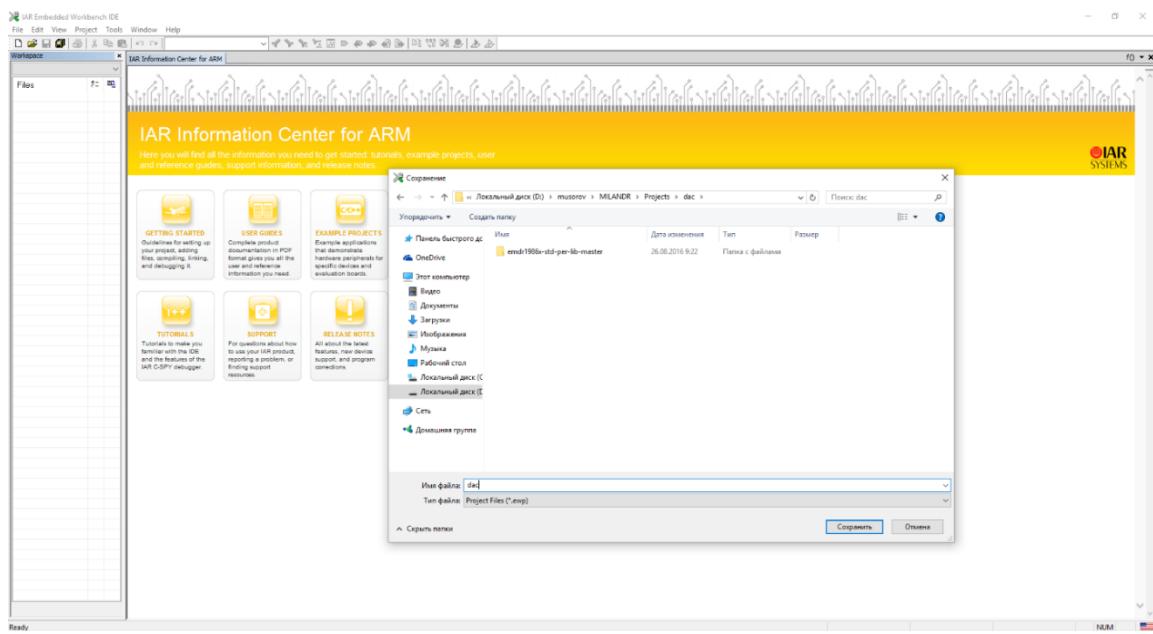


Рис. 1.14. Окно сохранения проекта

4. После сохранения проекта будет открыто его рабочее окно (рис. 1.15).

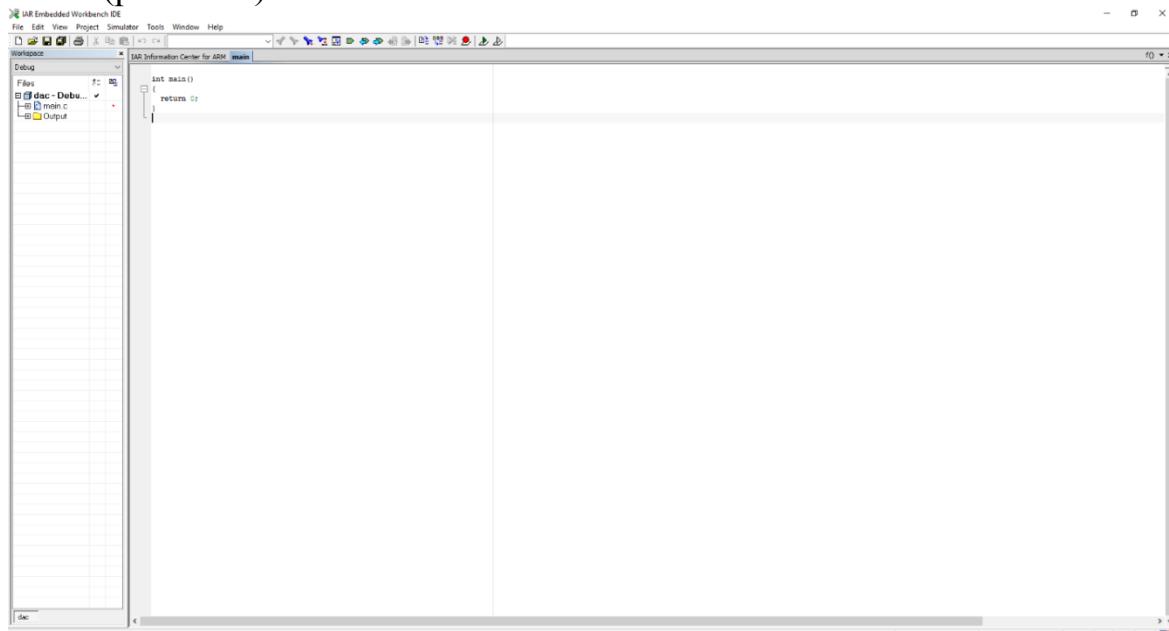


Рис.1.15. Рабочее окно проекта

5. Далее необходимо настроить проект. Для этого в окне «*Workspace*» выбирается пункт контекстного меню «*Options*» (рис. 1.16).

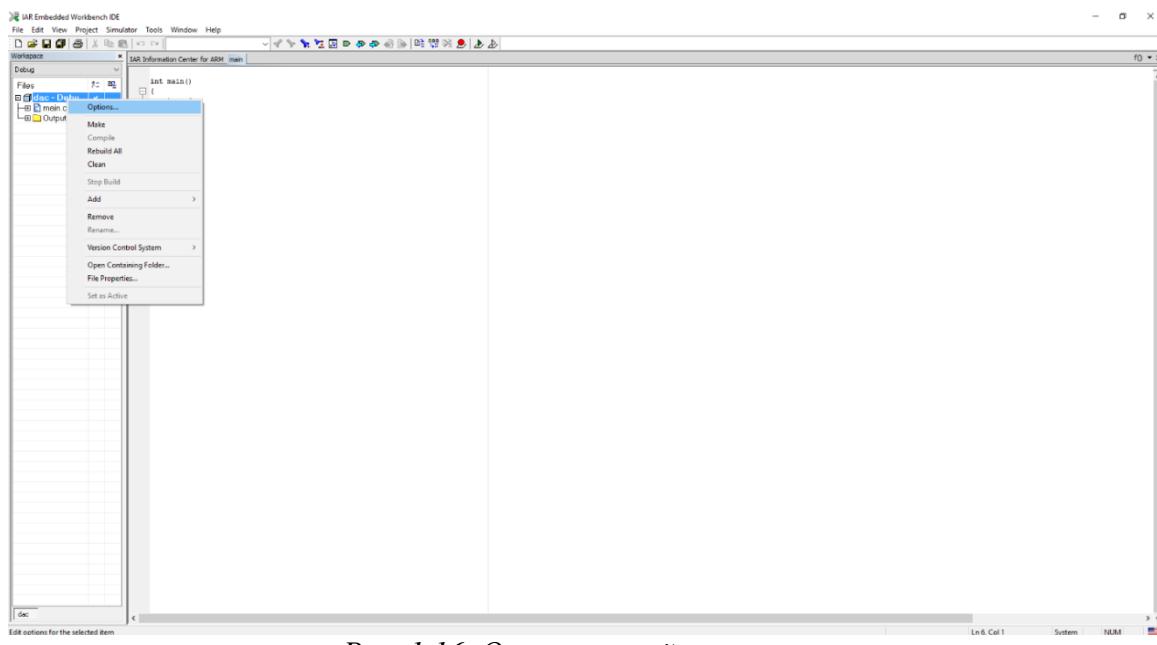


Рис. 1.16. Окно настройки проекта

6. В меню *General Options* во вкладке *Target* выбирается модель контроллера: *Device*—>*Milandr*—>*Milandr 1986BE9x* (рис. 1.17).

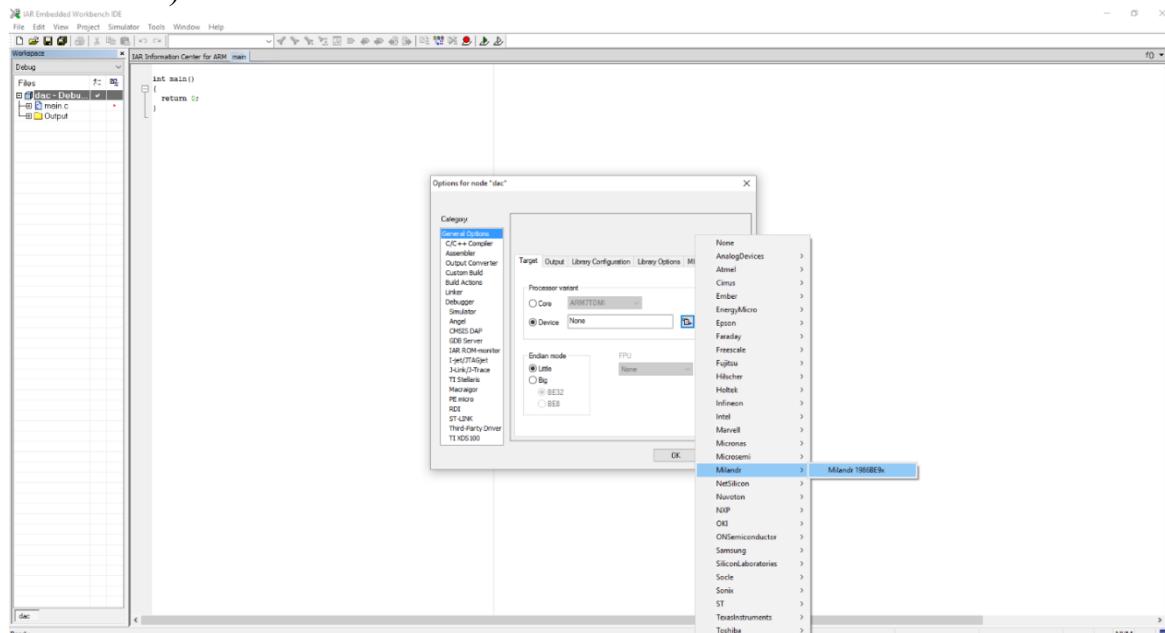


Рис. 1.17. Окно выбора микроконтроллера

7. Для дальнейшей работы с микроконтроллером необходимо подключить библиотеку, содержащую описание регистров, масок и битов. Существует стандартные библиотеки ядра микроконтроллеров серии Cortex – CMSIS. Данная аббревиатура расшифровывается как *Cortex Microcontroller Software Interface*.

Помимо этого, существует еще одна библиотека для микроконтроллеров *MDR32F9Qx* под названием *Standard Peripherals Library (SPL)*. Библиотека *SPL* может использоваться в дополнение к *CMSIS*, обеспечивая более быстрый и удобный доступ к периферии. Библиотеку *SPL* часто называют набором драйверов к периферийным модулям.

Для загрузки библиотек нужно перейти на страницу *C/C++ Compiler*, выбрать вкладку *Preprocessor* и в соответствующем окне указать пути к следующим файлам библиотеки *CMSIS* (рис. 1.18):

```
$PROJ_DIR$  
$PROJ_DIR$\emdr1986x-std-per-lib-master\Config  
$PROJ_DIR$\emdr1986x-std-per-lib-master\CMSIS\CM3\CoreSupport  
$PROJ_DIR$\emdr1986x-std-per-lib-  
master\CMSIS\CM3\DeviceSupport\MDR32F9Qx\inc  
$PROJ_DIR$\emdr1986x-std-per-lib-  
master\CMSIS\CM3\DeviceSupport\MDR32F9Qx\startup  
$PROJ_DIR$\emdr1986x-std-per-lib-master\MDR32F9Qx_StdPeriph_Driver\inc  
$PROJ_DIR$\emdr1986x-std-per-lib-master\MDR32F9Qx_StdPeriph_Driver\src
```

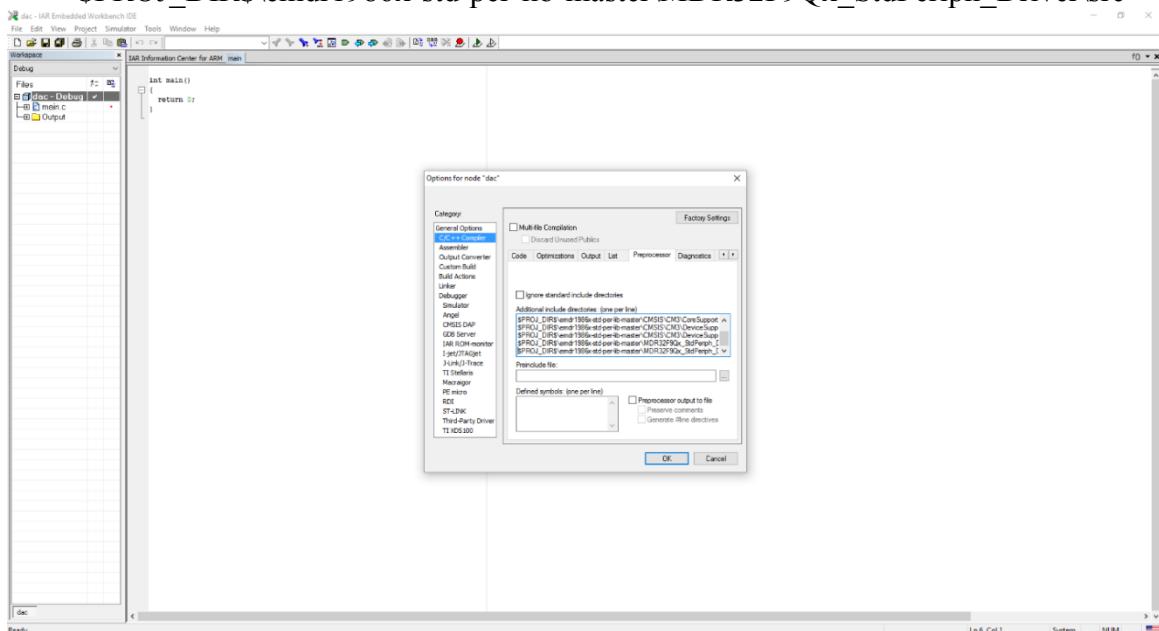


Рис. 1.18. Содержимое окна *C/C++ Compiler* вкладки *Preprocessor*

8. Далее нужно перейти в меню *Linker*, где следует выбрать пункт *Override default* и выбрать *Linker* как показано на рис. 1.19 и 1.20.

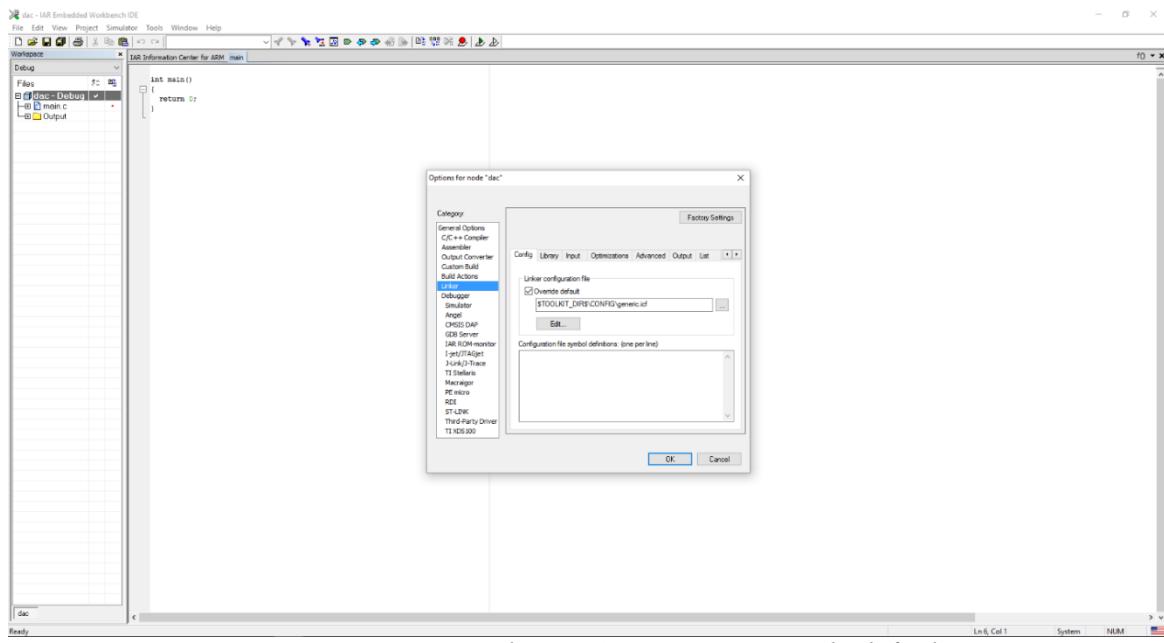


Рис. 1.19. Меню Linker выбор опции Override default

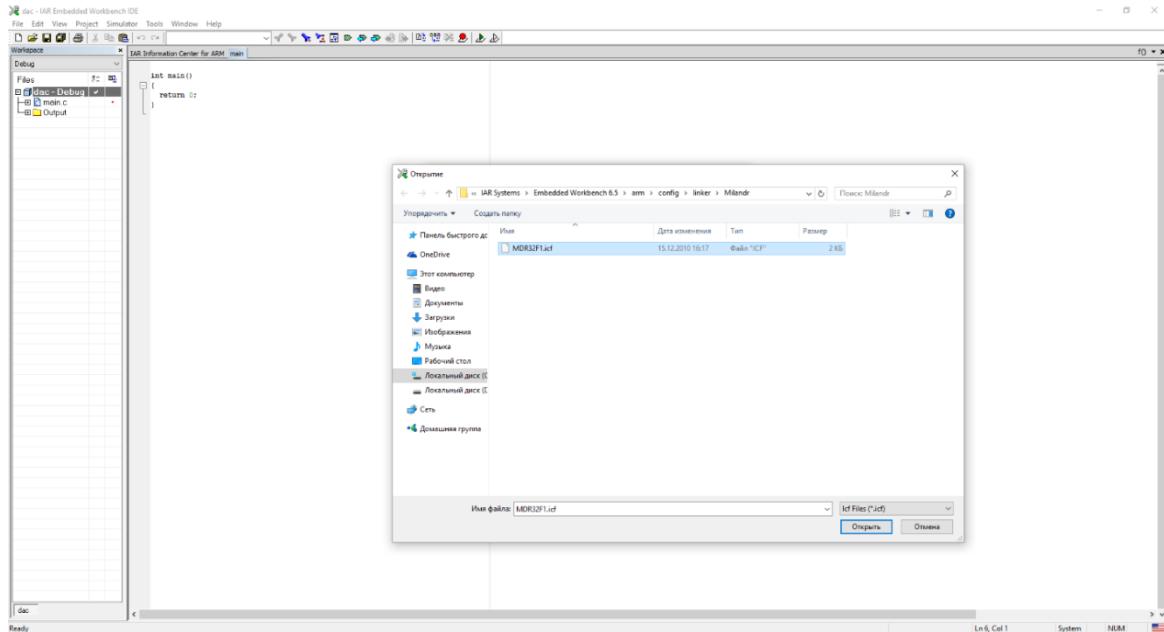


Рис. 1.20 Окно выбора Linker фирмы «Миландр»

9. Далее необходимо перейти в меню *Debugger*. Во вкладке *Setup* расположено поле *Driver*, в котором следует выбрать *J-Link/J-Trace* (рис. 1.21).

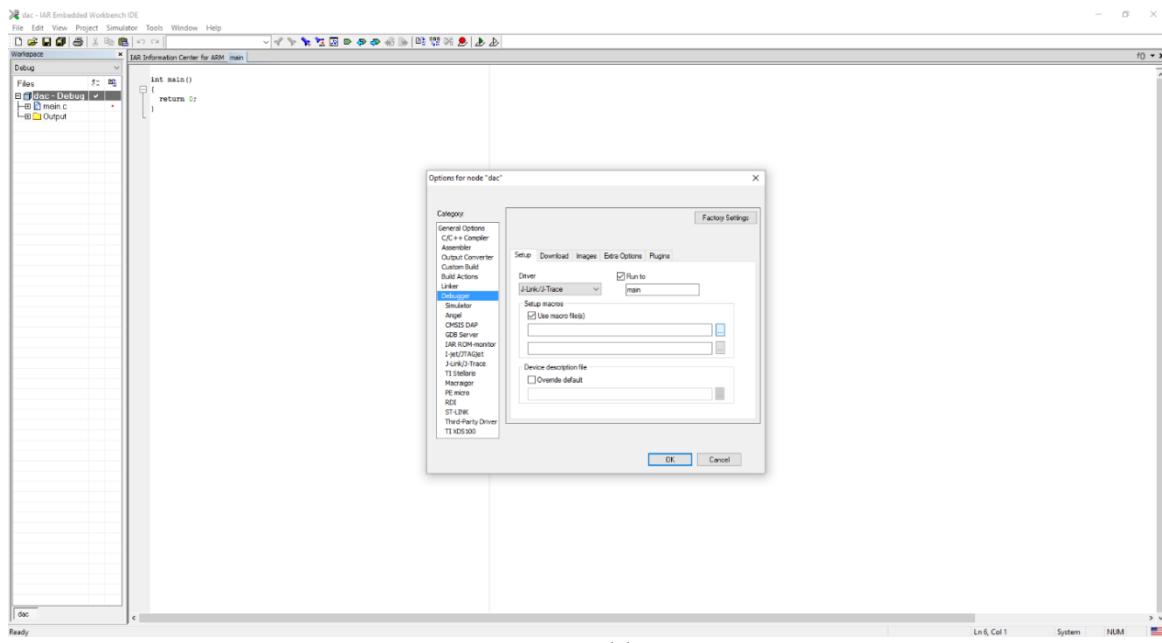


Рис. 1.21. Страница Debugger, вкладка Setup

10. В поле use macro file(s) необходимо добавить файл FlashMDR32F1x (рис. 1.22).

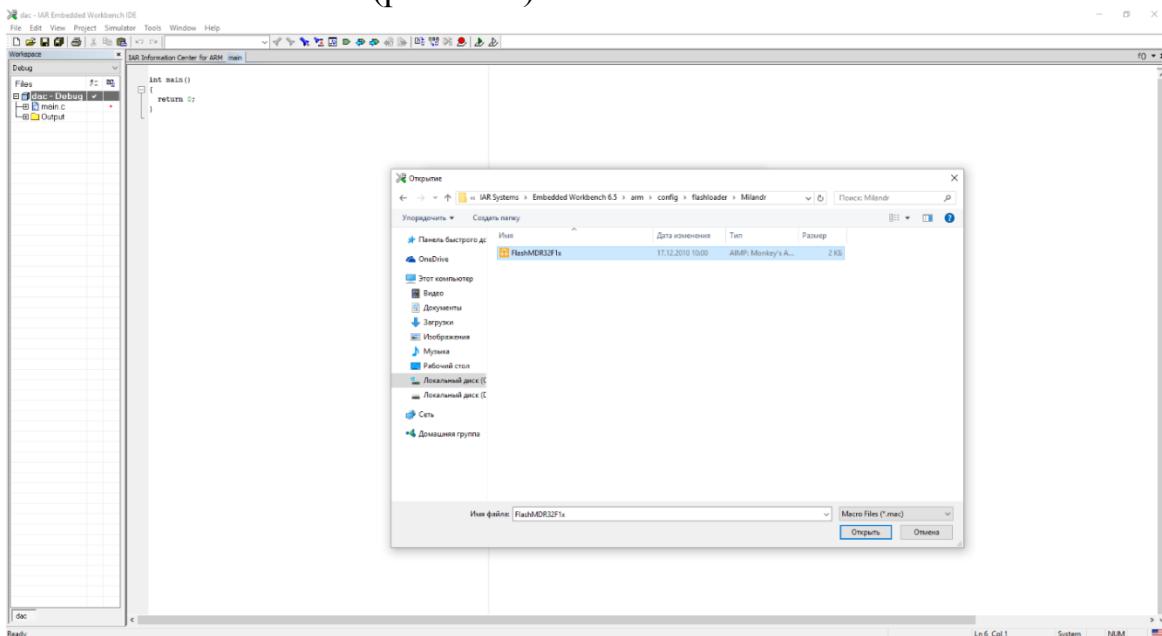


Рис. 1.22. Окно добавления макро файла FlashMDR32F1x

11. Во вкладке Download меню Debugger следует выбрать необходимые пункты, как показано на рис. 1.23 и выбрать flashloader фирмы «Миландр» (рис. 1.24).

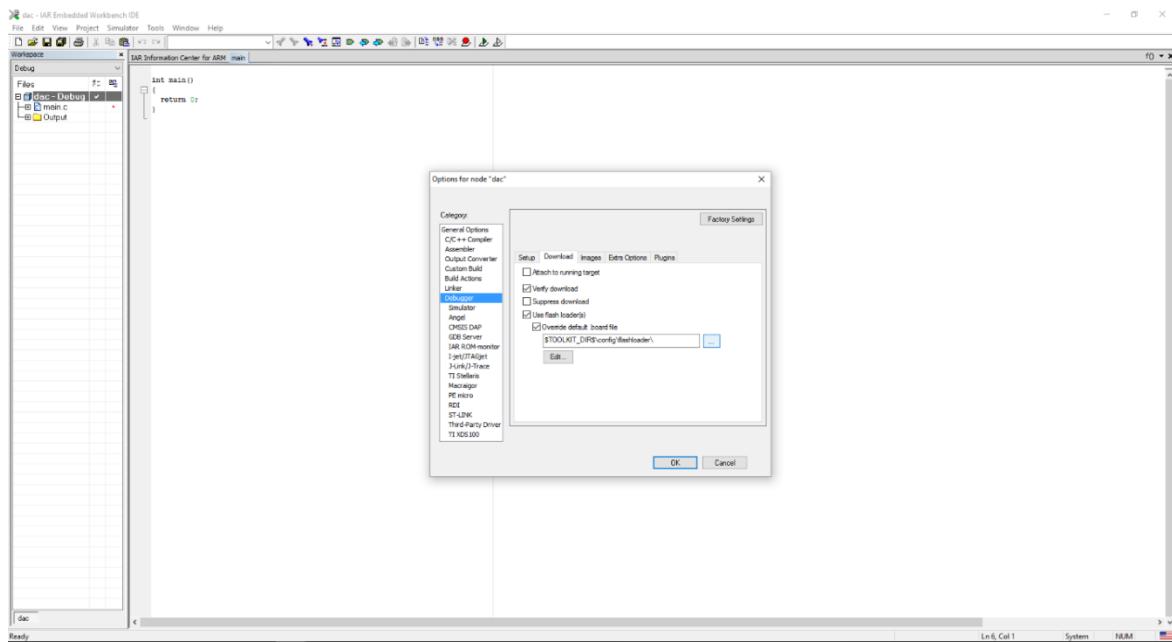


Рис. 1.23. Меню Debugger, вкладка Download

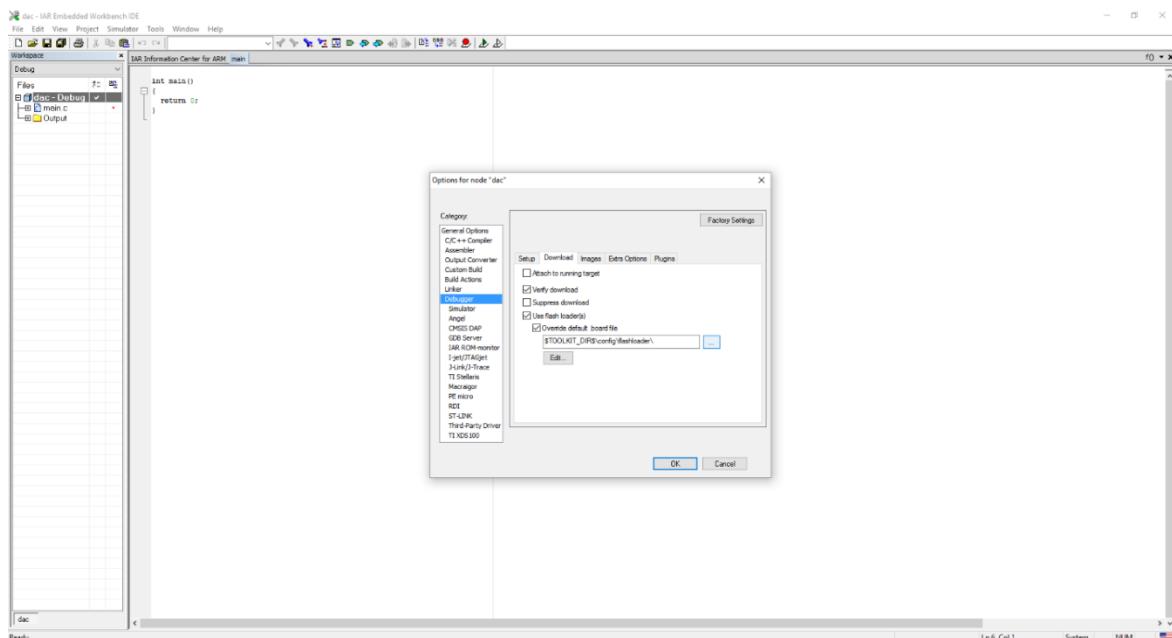


Рис. 1.24. Окно выбора flashloader фирмы «Миландр»

12. Далее в меню J-Link/J-Trace во вкладке connection необходимо выбрать SWD (рис. 1.25).

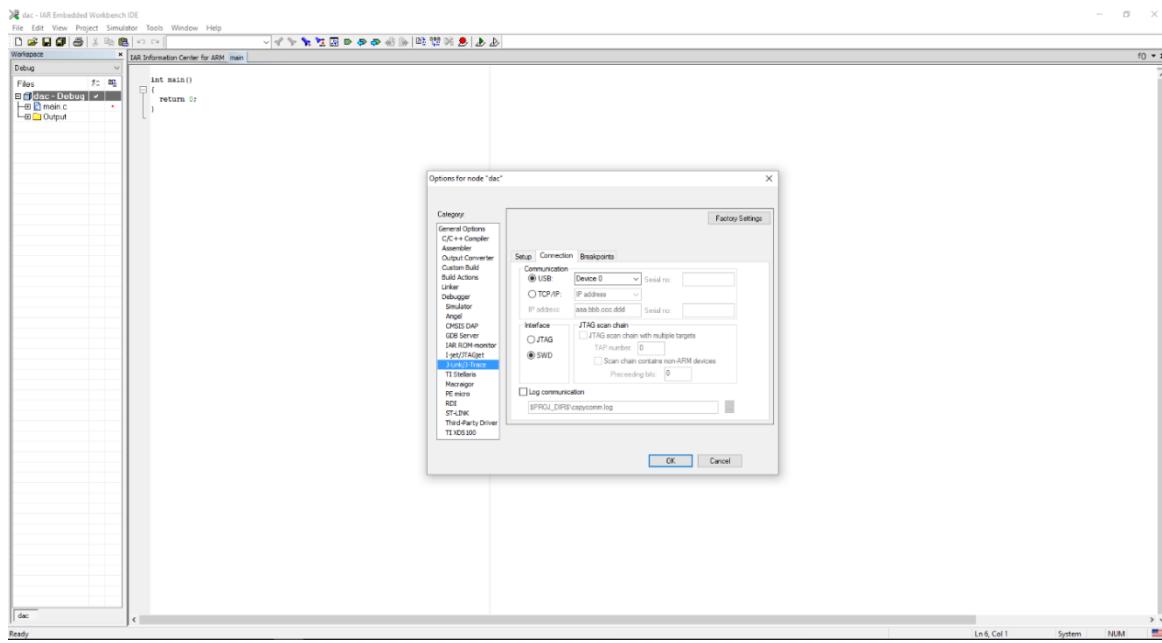


Рис. 1.25. Меню J-Link/J-Trace, вкладка Connection

13. После осуществления всех настроек в контекстном меню «*Options*» необходимо нажать правой кнопкой мыши на окно «*Workspace*» и добавить файлы стандартных библиотек: *Add—>Add Files* (рис. 1.26):

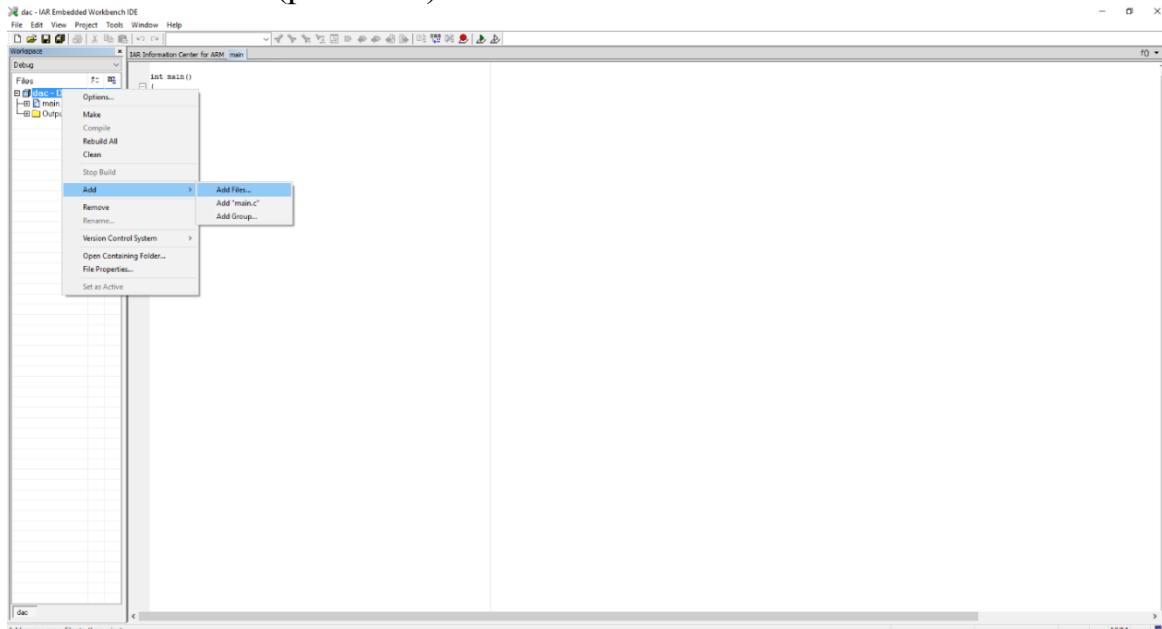


Рис. 1.26. Меню J-Link/J-Trace, вкладка Connection

14. Необходимо добавить файлы как показано на рис. 1.27 – 31.

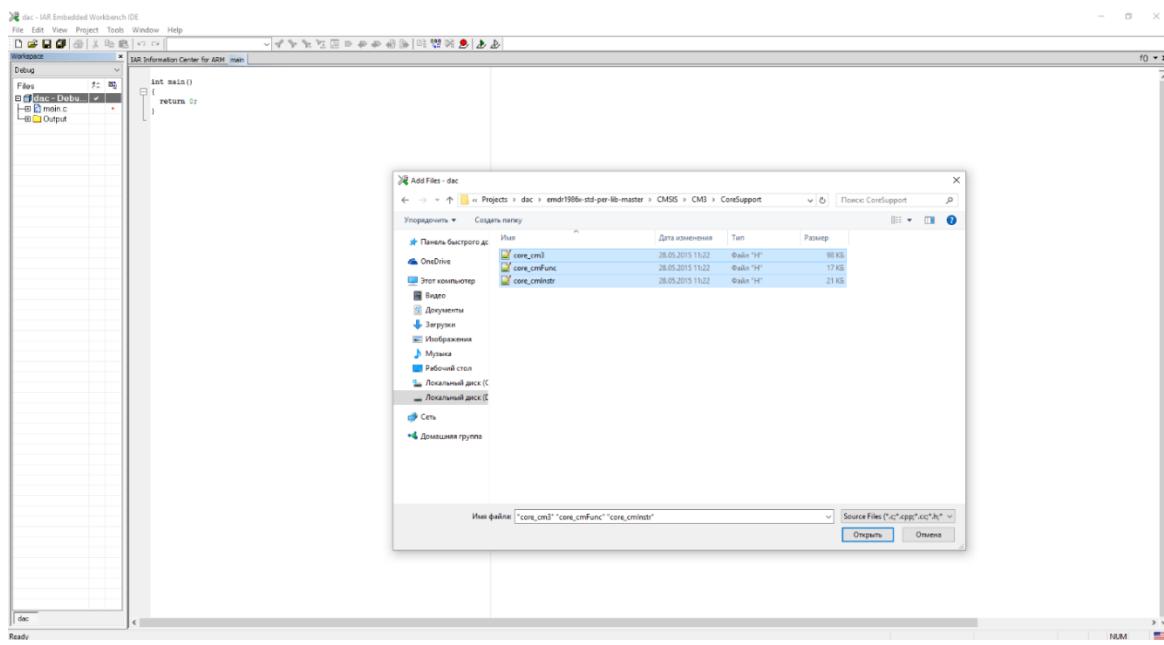


Рис. 1.27. Окно добавления файлов: core_cm3.h, core_cmfunc.h и core_cminstr.h

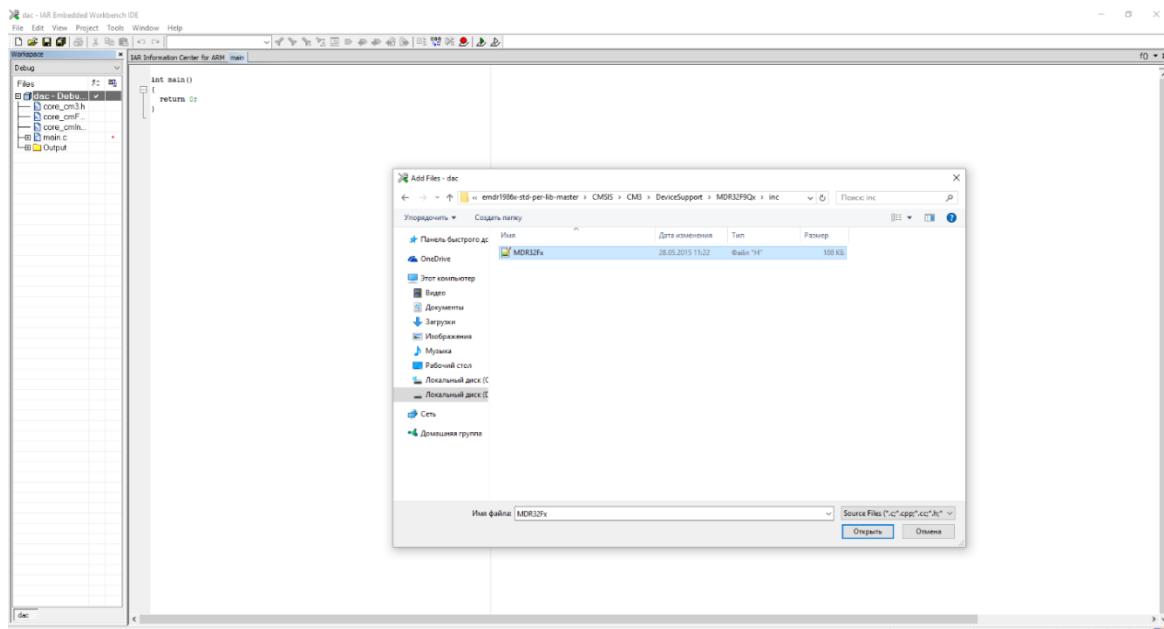


Рис. 1.28. Окно добавления файла MDR32Fx.h

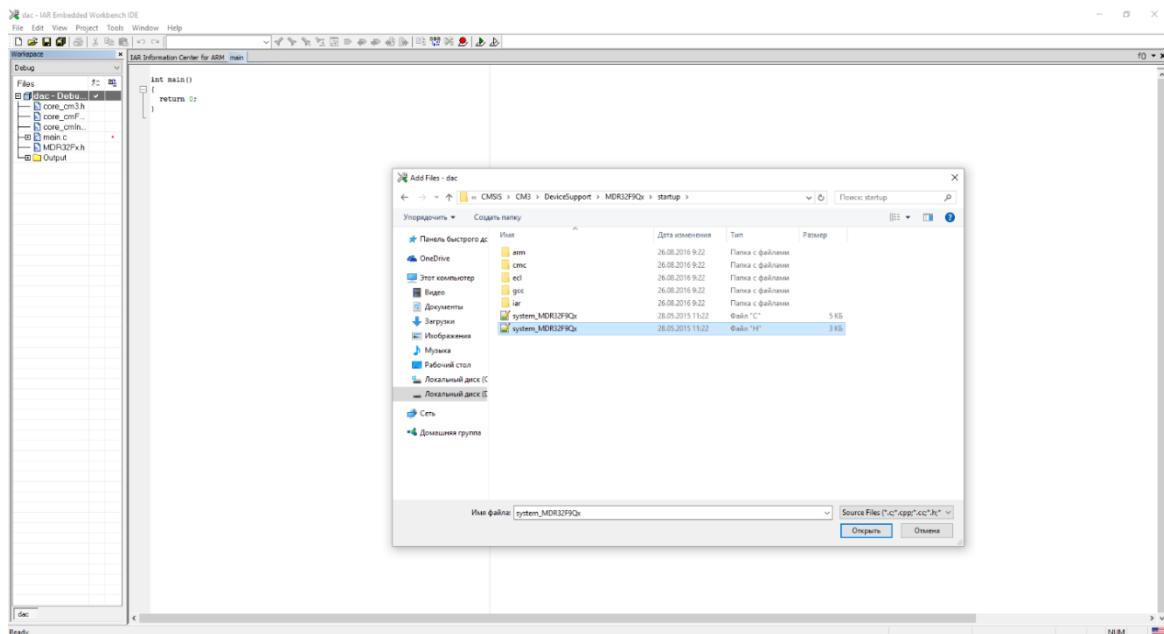


Рис. 1.29. Окно добавления файла *system_MDR32F9Qx.h*

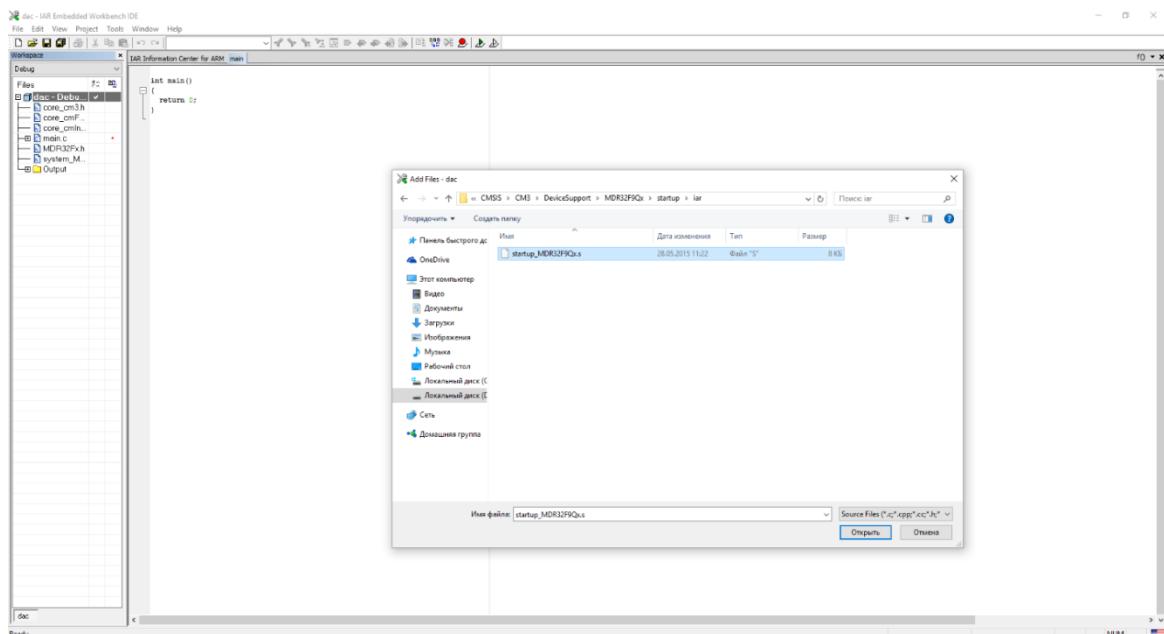


Рис. 1.30. Окно добавления файла *startup_MDR32F9Qx.s*

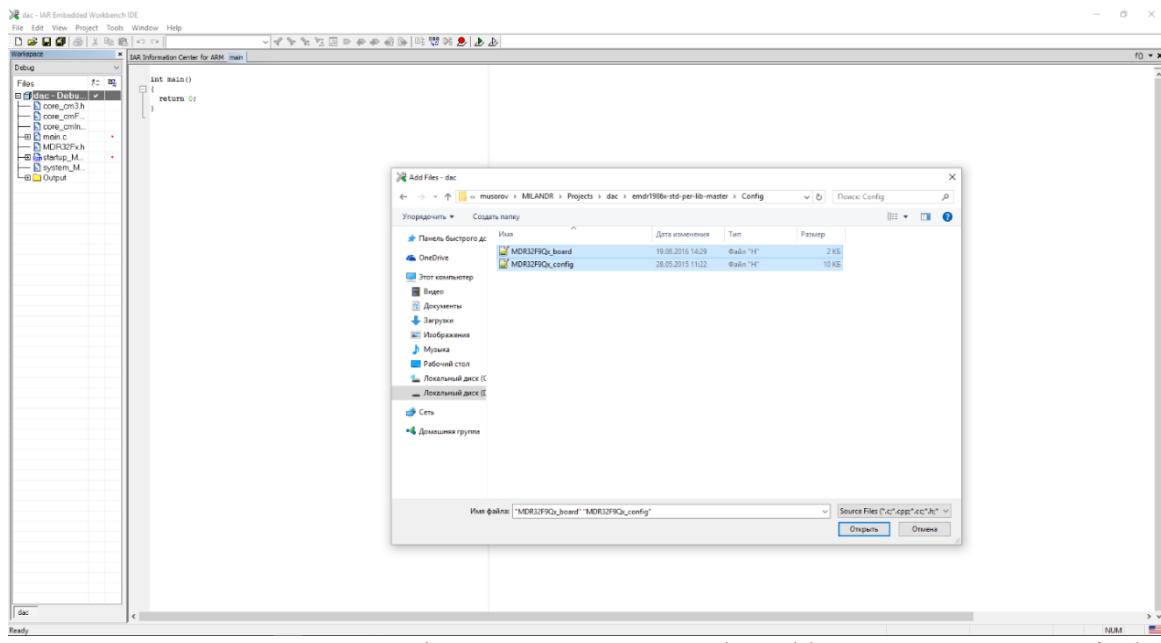


Рис. 1.31. Окно добавления файлов: *MDR32F9Qx_board.h* и *MDR32F9Qx_config.h*

15. Загрузка программы в микроконтроллер осуществляется в три этапа (рис. 1.32): компиляция (*Compile*), создание (*Make*), загрузка и отладка (*Download and Debug*)

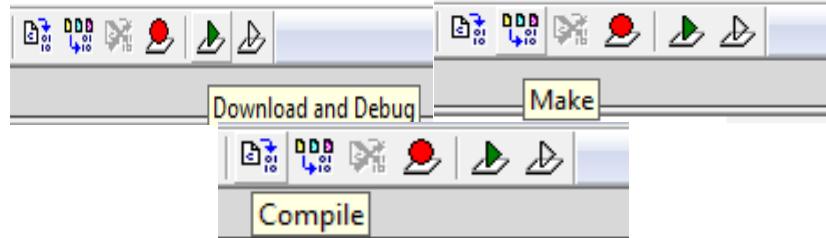


Рис. 1.32. Панели компиляции и загрузки программы

16. Далее можно запускать программу, как показано на рис. 1.33.

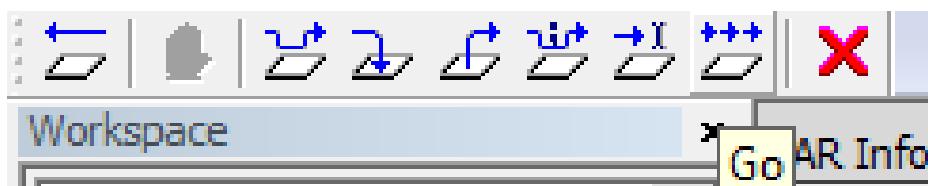


Рис. 1.33. Запуск программы

Настройка программного продукта Keil µVision 5

1. Необходимо скачать с официального [сайта](#): «Software pack для Keil MDK 5 (MDR32F9Qx, MDR1986VE1T, MDR1986VE3T)», распаковать и установить. С keil 4 версии данный файл не работает. Только keil µVision 5.
2. Далее нужно установить 2 файла: «Setup_JLinkARM_V468a» и «MT-Link». Это драйвера для программатора MT-LINK. Их нет на CD диске.
3. После этих действий подключаем MT-LINK к компьютеру, устройство должно определиться, но на всякий случай идем в диспетчер устройств и проверяем, определилось ли устройство.

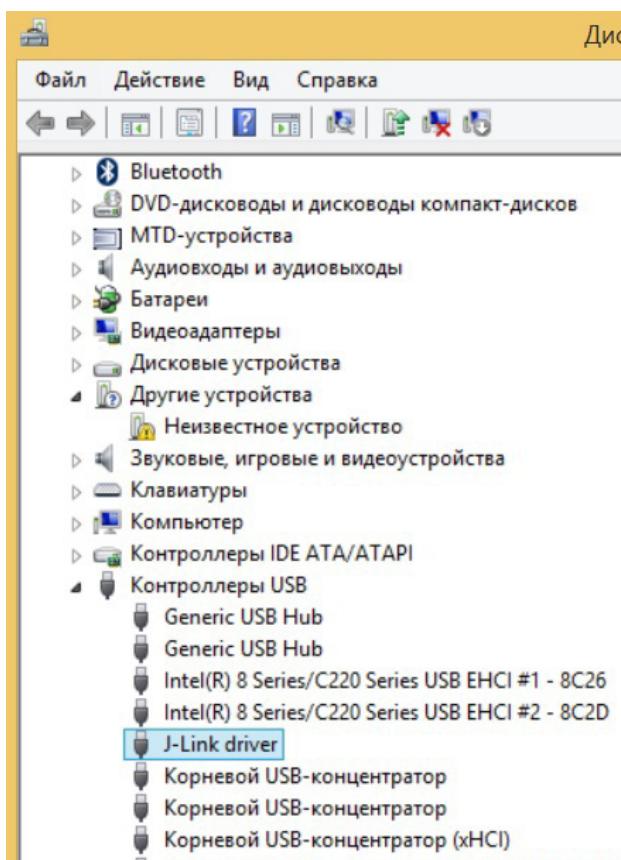


Рис.1.34. Диспетчер устройств

3. Скопировать файл «MDR32F9x.FLM» из CD диска, в папку «Flash» в директории с keil 5. По умолчанию путь такой «C:\Keil_v5\ARM\Flash».

Создание проекта в программном продукте Keil5

Внимание!, если в пути файлов есть русские символы, то проект не скомпилируется.

Подключим программатор к JTAG_A (верхнему порту) и выставляем соответствующее положения на рычагах BOOT загрузки (Самый верхний влево, нижние два — вправо). Подключаем питание. Не забываем переключить рычаг выбора источника питания.

Запускаем программный продукт Keil μVision 5. Переходим на вкладку Project->New μVision Project... Сохраняем проект.

Далее необходимо выбрать микроконтроллер. Микроконтроллер фирмы Milandr с ядром Cortex-M3, называние K1986BE92. Находим нужный, справа в окошке можно увидеть основные характеристики микроконтроллера. Нажимаем кнопку «ok».

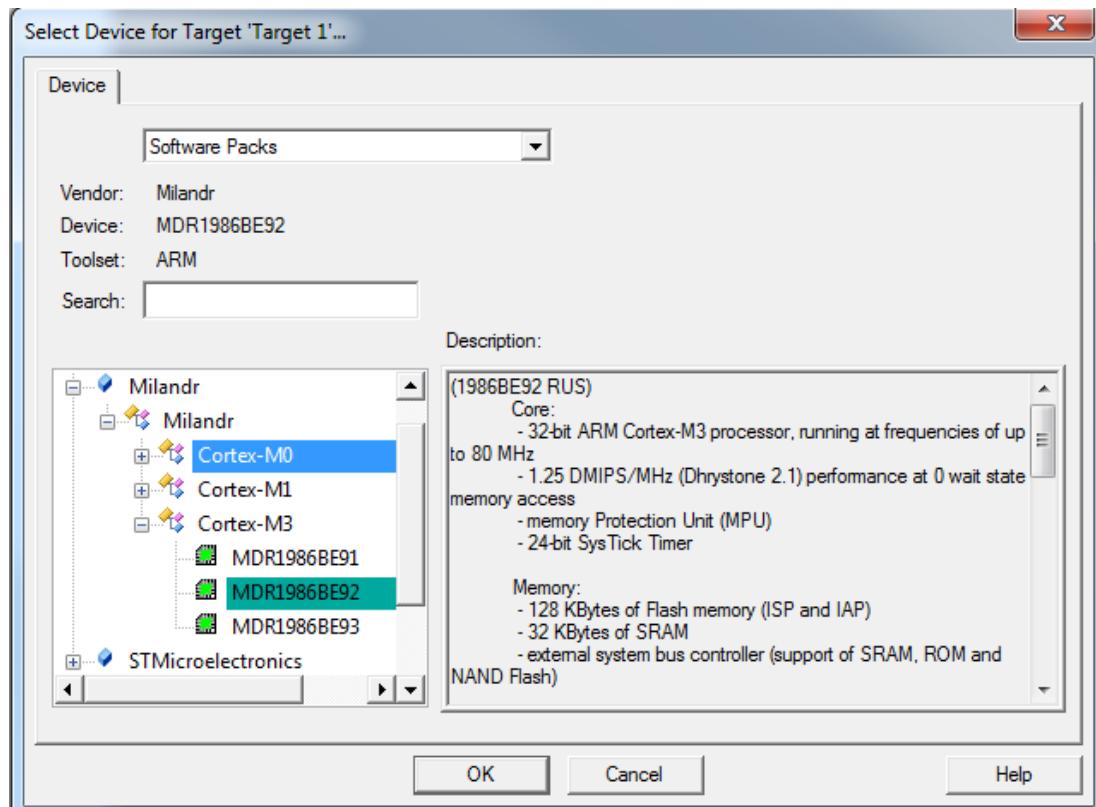
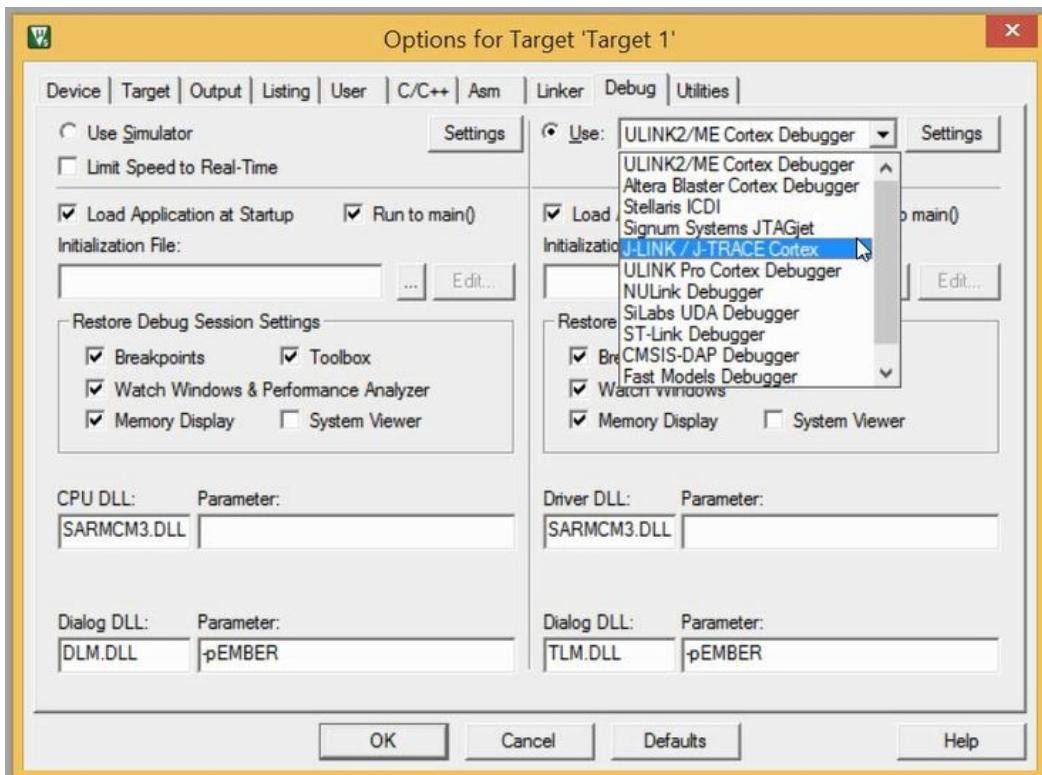


Рис.1.35. Выбор микроконтроллера

Далее открывается окно «Manage Run-Time Environment». В данном окне можно выбрать интересующие стандартные библиотеки.

Настраиваем проект. Переходим на вкладку Project->Options for Target ‘Target 1’... Откроется диалоговое окно настроек.

Переходим на вкладку Debug. Вкладка Debug разбита на 2 колонки: «Use Simulator» – режим симуляции и «Use» – выбор отладочного инструмента «эмулятора». Выбираем «Use» и выбираем «J-Link» эмулятор.



Rис.1.36. Выбор эмулятора

После выбора эмулятора нажимем кнопку «Setting» - настройки. Настроим эмулятор. Переходим на вкладку Debug. В поле «SN:» должен быть сразу показан номер эмулятора. Если его нет, то что-то не так с драйверами. Далее в списке «PORT» нужно сменить JTAG на SW и выбрать частоту в списке 1MHz. После этого справа должен появиться код микроконтроллера, как на (рисунке 1.36). Если ничего не появилось, то нужно проверить правильность выбранного канала эмулятора «A» или «B». После нажать кнопку «RESET» и еще раз выбрать частоту. Если не помогло, то проверьте, как установлен контроллер в кроватку. Часто достаточно всего лишь нажать на нее, чтобы ножки «отошли» и «встали» обратно. После чего снова нажать «RESET» выбрать частоту.

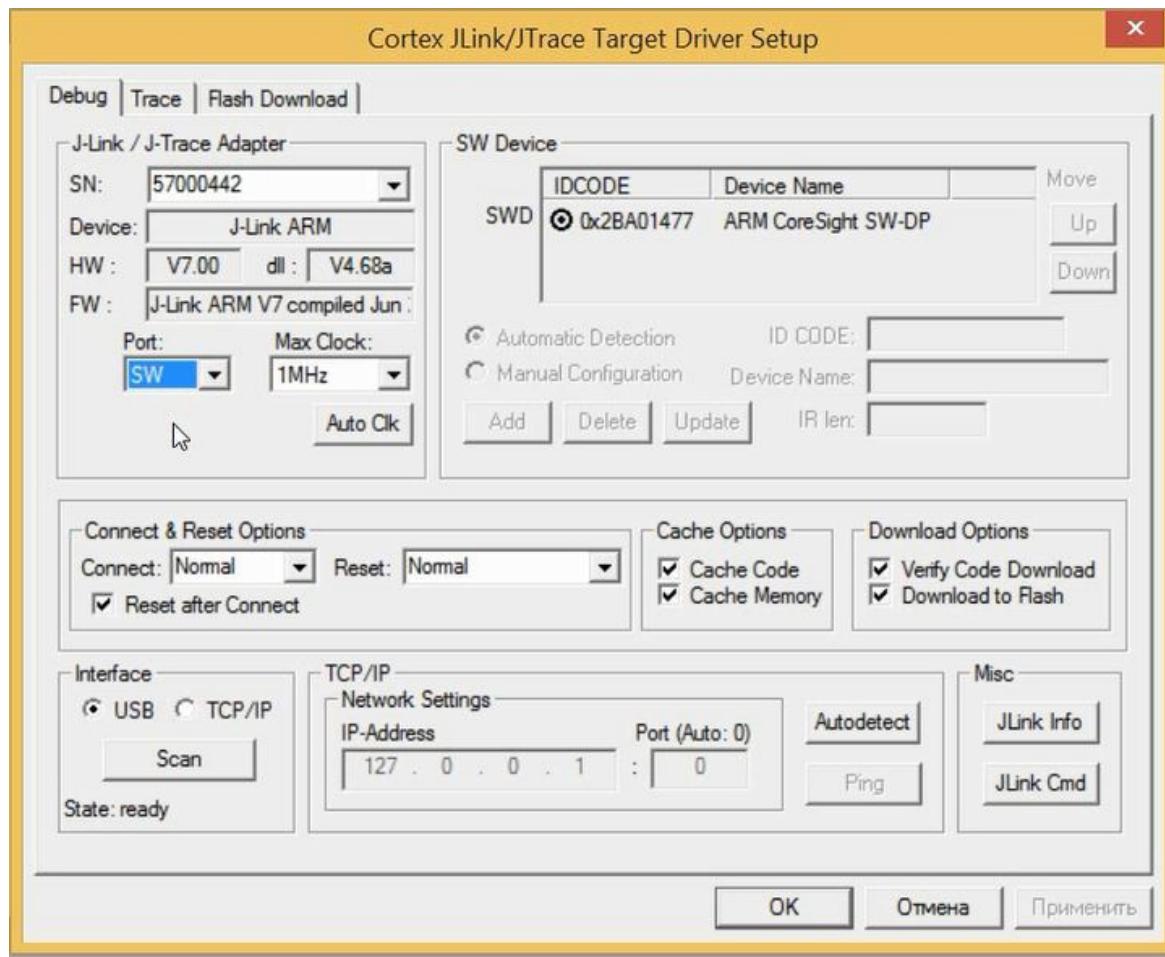


Рис.1.37. Настройка эмулятора

После успешного распознавания контроллера – переходим во вкладку «Flash Download». Там ставим птичку около «Erase Full Chip», как на рисунке, и жмем Add.

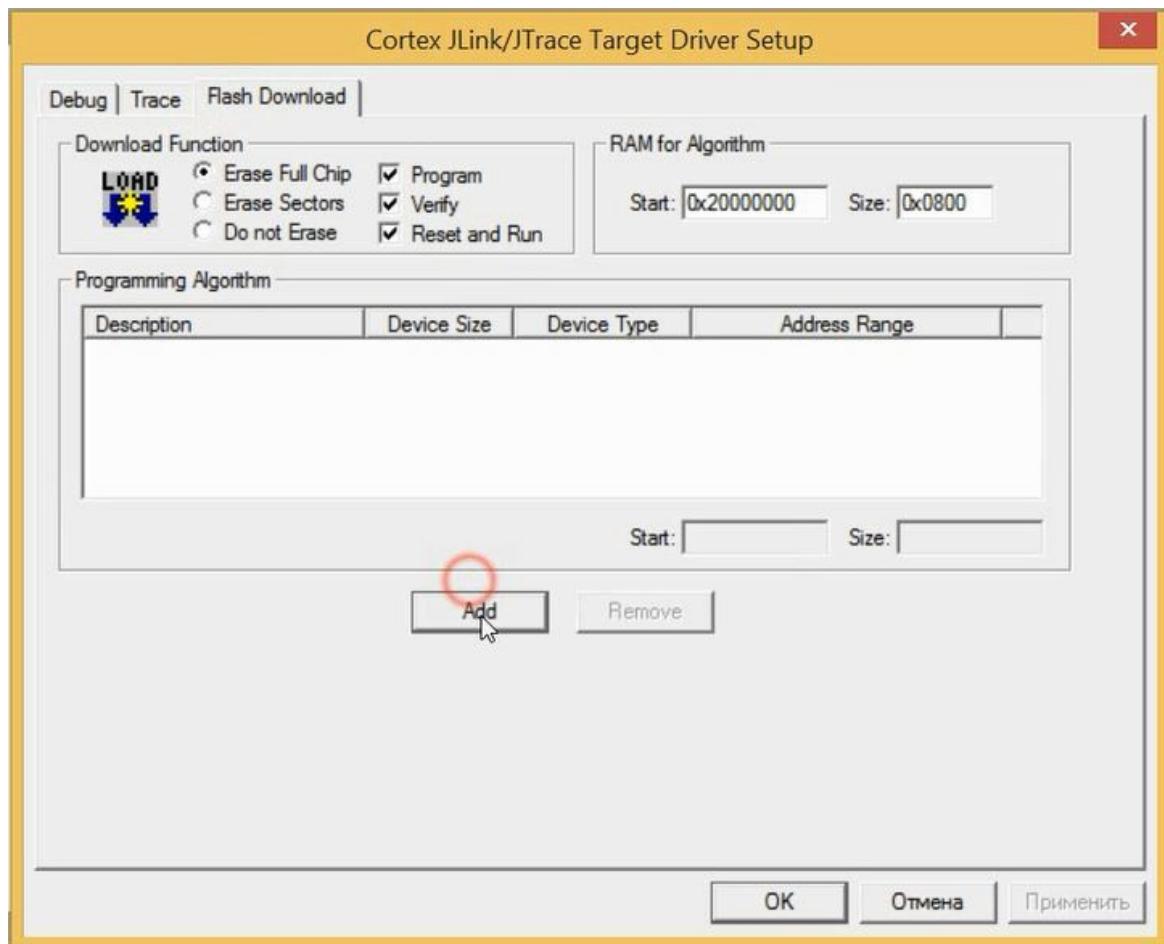


Рис.1.38. Добавление нужного микроконтроллера

Из этого списка выбираем наш микроконтроллер и жмем ОК. Если микроконтроллера нет, то это значит, что вы не скопировали FLM файл в папку Flash описанную в разделе «Настройка программного продукта Keil μ Vision 5».

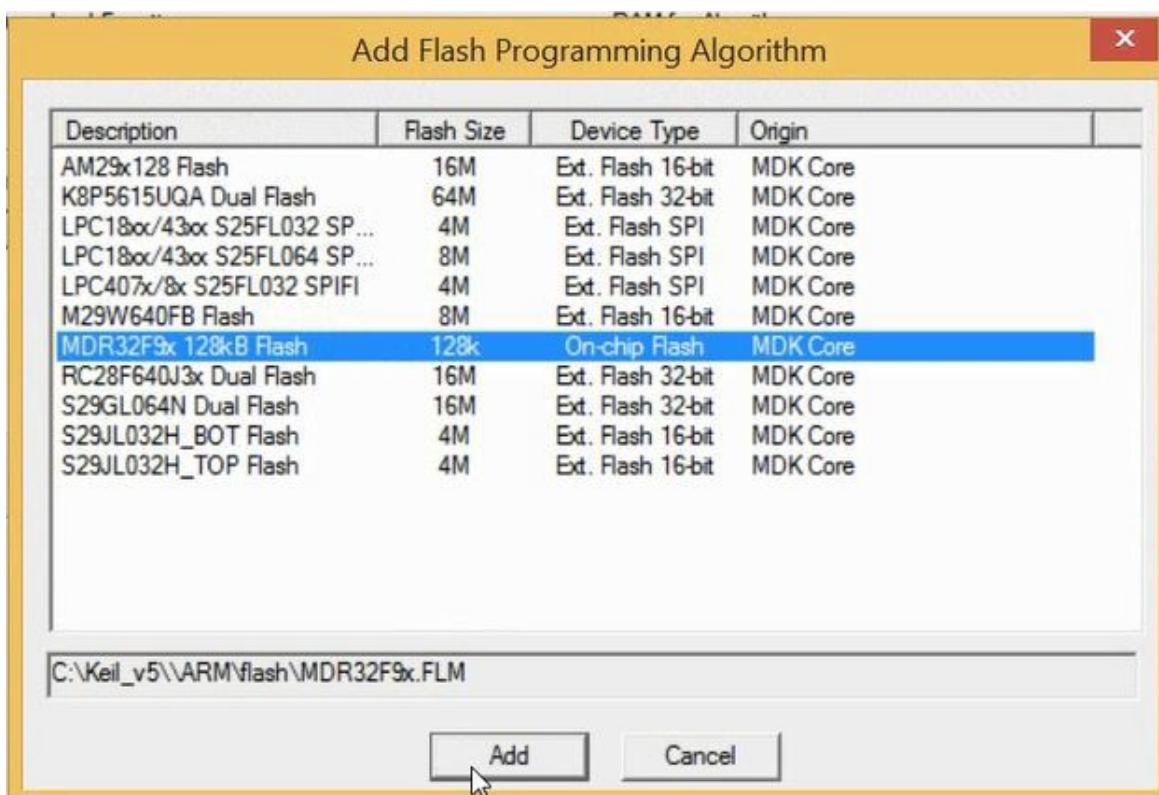


Рис.1.39. Выбор микроконтроллера

Нажимем кнопку «Add», настройка эмулятора закончена.

Переходим во вкладку «Utilities». Там ставим точку слева от Use Target Driver for Flash Programming, после выбираем наш J-LINK. Ну и на последок переходим в «Settings». Там все должно быть уже настроено так же, как и в предыдущем меню. Но все равно проверяем.

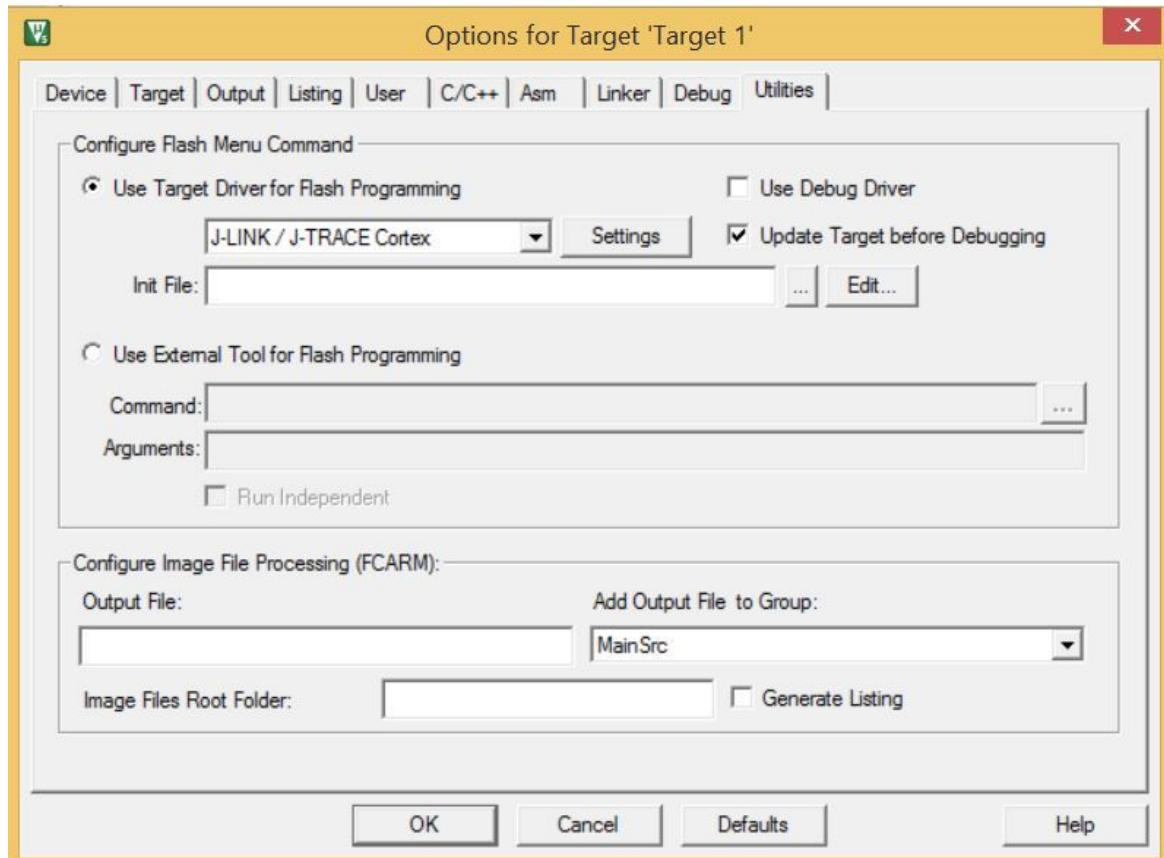


Рис.1.40. Настройка проекта

Теперь необходимо добавить к проекту файл в котором будем присать программу. Правой кнопкой нажимаем на «Source Group1» и выбираем «Add New Item Group ...». Выбираем тип файла «C File(.c)» сохраняес в рабочей папке.

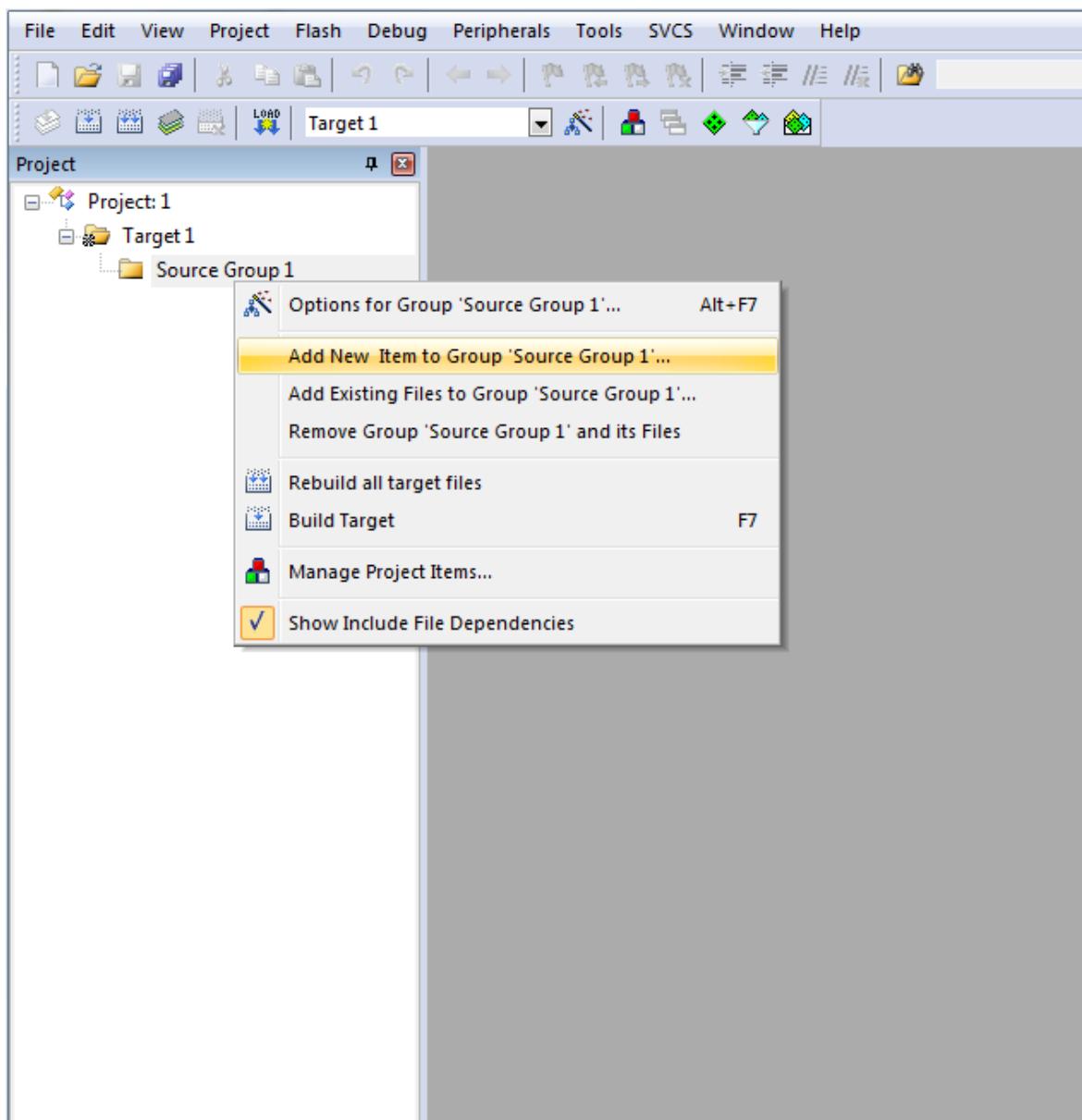


Рис.1.41. Добавление в проект основного файла программы *main.c*

Далее необходимо добавить файлы библиотеки в проект. Добавление происходит следующим образом. Щелкаем правой кнопкой по нужной папке в дереве проекта и выбираем «Add Existing Files to Group ‘Имя группы’ ...».

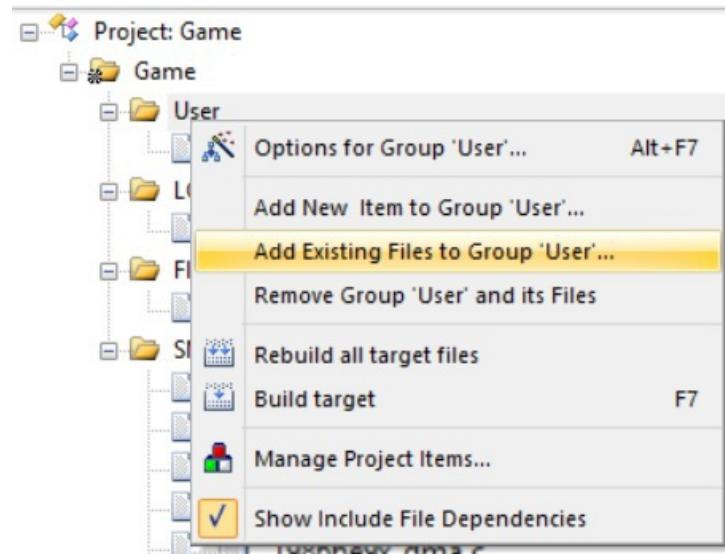


Рис.1.42. Добавление библиотек

В открывшемся окне нужно выбрать тип файлов «All files (*.*)». После чего выбрать 1 или выделить несколько файлов.

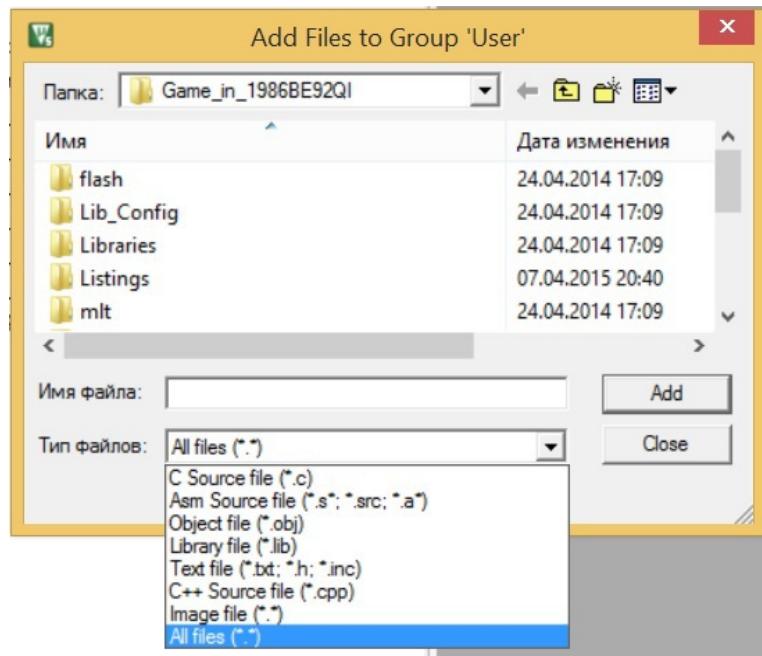


Рис.1.43. Выбор библиотечных файлов

Добавить нужно:

1. В папку SMSIS_and_Drivers -> «Libraies\1986BE9x_StdPeriph_Driver\src -> все файлы .c». Тут хранится аналог SPL у STM32. Проще говоря, это «обертки»,

которые позволяют не вдаваясь в структуру контроллера управлять его периферией (выводы, uart и т.д.).

2. В папку SMSIS_and_Drivers -> «Libraries\CMSIS\CM3\DeviceSupport\1986BE9x\startup\arm -> startup_1986be9x.s». Это файл «стартап». Тут прописаны все «вектора переходов». Иначе говоря, по любому прерыванию (к примеру, нажатие кнопки) контроллер возвращается к этой таблице и смотрит, куда ему перейти, чтобы выполнять код дальше.

3. В папку SMSIS_and_Drivers -> «Libraries\CMSIS\CM3\DeviceSupport\1986BE9x\startup\arm -> system_1986BE9x.c».

Итогом должно стать такое дерево.

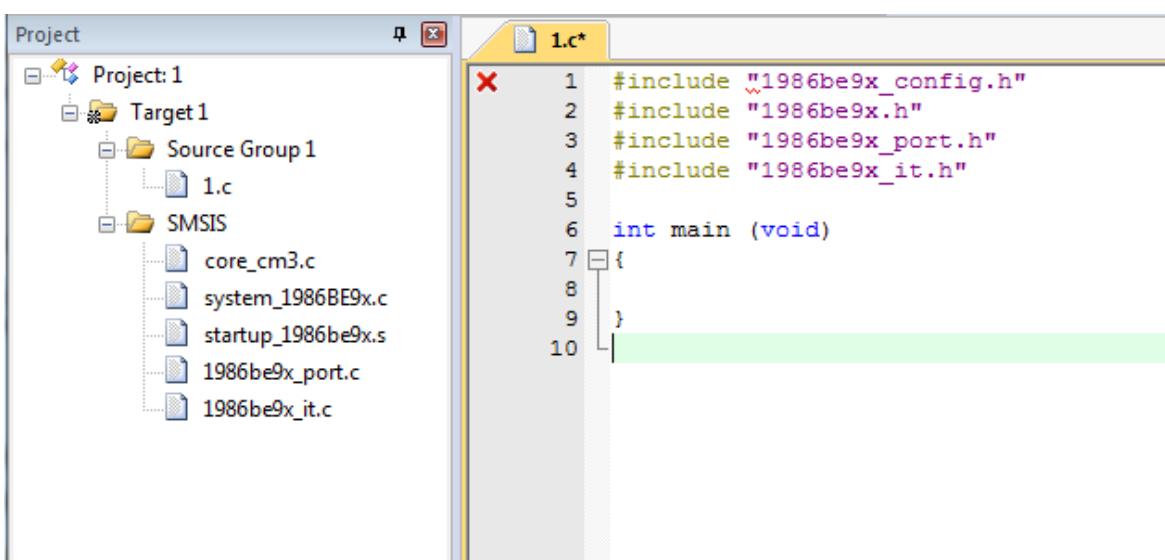


Рис.1.44. Дерево проекта

Как можно было заметить, около самого верхнего #include файла стоит крестик. Keil просто не видит данного файла. Для того, чтобы исправить это, мы должны указать ему, где ему брать этот файл. Для этого жмем Alt+F7. В открывшемся окне переходим во вкладку C/C++. Для того, чтобы исправить это, нужно нажать галочку около надписи «C99 Mode». Это даст возможность писать на более совершенном стандарте языка Си, чем это можно было делать изначально. Далее следует нажать

на прямоугольник с «...» внутри. Справа около строчки с подписью «Include Paths».

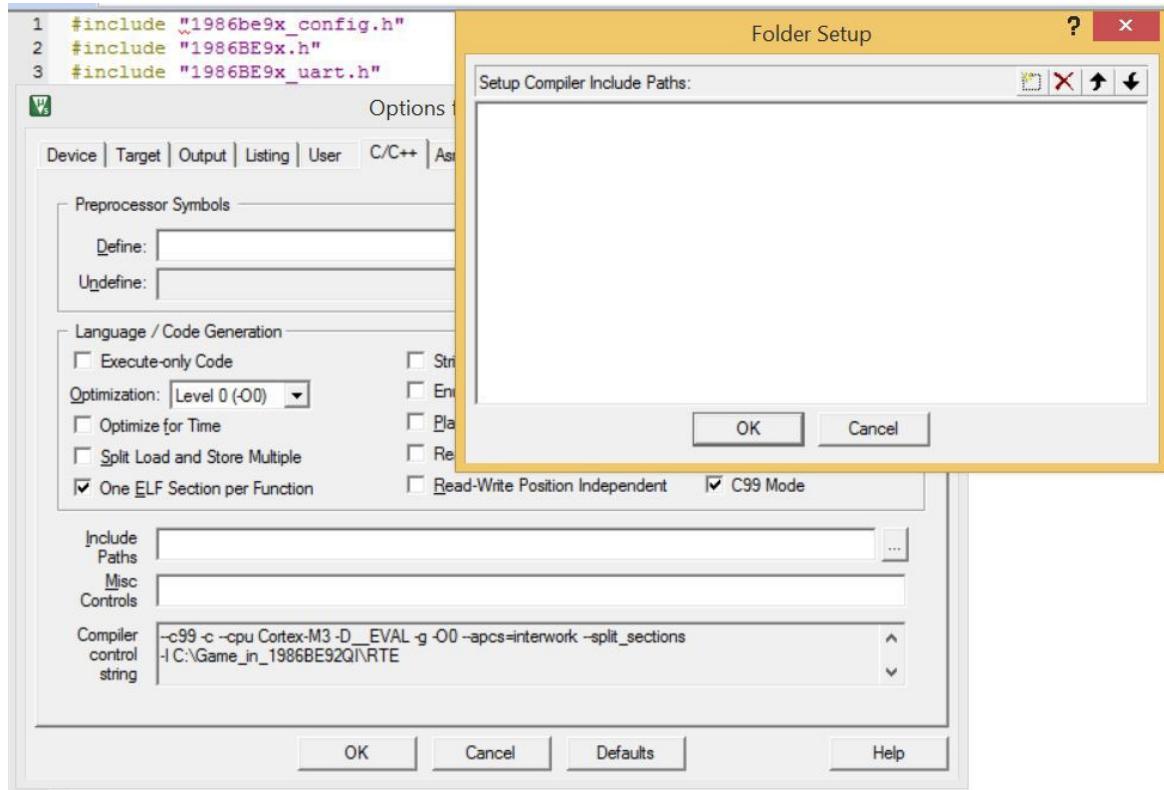


Рис.1.45. Окно добавления пути библиотечных файлов

В открывшемся окне нажимаем на иконку с прямоугольником, слева от крестика. Это создаст пустую строку. В правом углу созданной строчки жмем на «...». После чего указываем нужную папку, в которой лежат интересующие нас файлы. После этого жмем «ОК». Папка будет добавлена. Необходимо добавить все эти пути. Если вы заметили, то все ссылки кроме одной – относительные. То есть идут от корневого каталога. Но 1 идет начиная от «C:\». Это ссылка на сам каталог с проектом. Его тоже следует указать.

Жмем «OK» и переходим в файл main.c.

Компилируем простейшую программу.

```

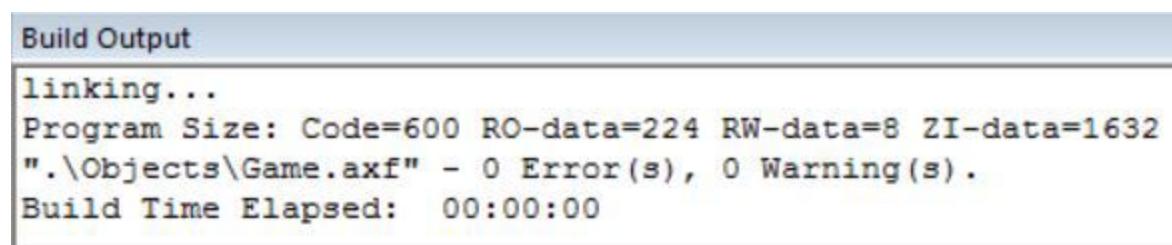
1 #include "1986be9x_config.h"
2 #include "1986BE9x.h"
3 #include "1986BE9x_uart.h"
4 #include "1986BE9x_port.h"
5 #include "1986BE9x_rst_clk.h"
6 #include "1986BE9x_it.h"
7 #include "mlt_lcd.h"
8 #include "MilFlash.h"
9
10 uint32_t Loop = 0;
11
12 int main (void)
13 {
14     while (1)
15     {
16         Loop++;
17     }
18 }
19

```

Рис.1.46. Пример простой программы

Не забываем о том, что в конце каждого файла должна быть пустая строка. Keil считает это как предупреждение, которое частенько действует на нервы. Конечно, компилироваться будет, но сам факт предупреждения – настораживает.

На этом настройка закончена. Мы можем перекомпилировать наш проект нажав F7. Если все прошло хорошо, ты мы увидим это.



```

Build Output
linking...
Program Size: Code=600 RO-data=224 RW-data=8 ZI-data=1632
".\Objects\Game.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:00

```

Рис.1.47. Информационное окно проекта

Глава 2. НАСТРОЙКА ПОРТОВ ВВОДА ВЫВОДА. ПРИМЕРЫ

Порты ввода-вывода MDR_PORTx

Микроконтроллер имеет 6 портов ввода/вывода. Порты 16-разрядные и их выводы мультиплексируются между различными функциональными блоками, управление для каждого вывода отдельное. Для того, чтобы выводы порта перешли под управление того или иного периферийного блока, необходимо осуществить настройку портов.

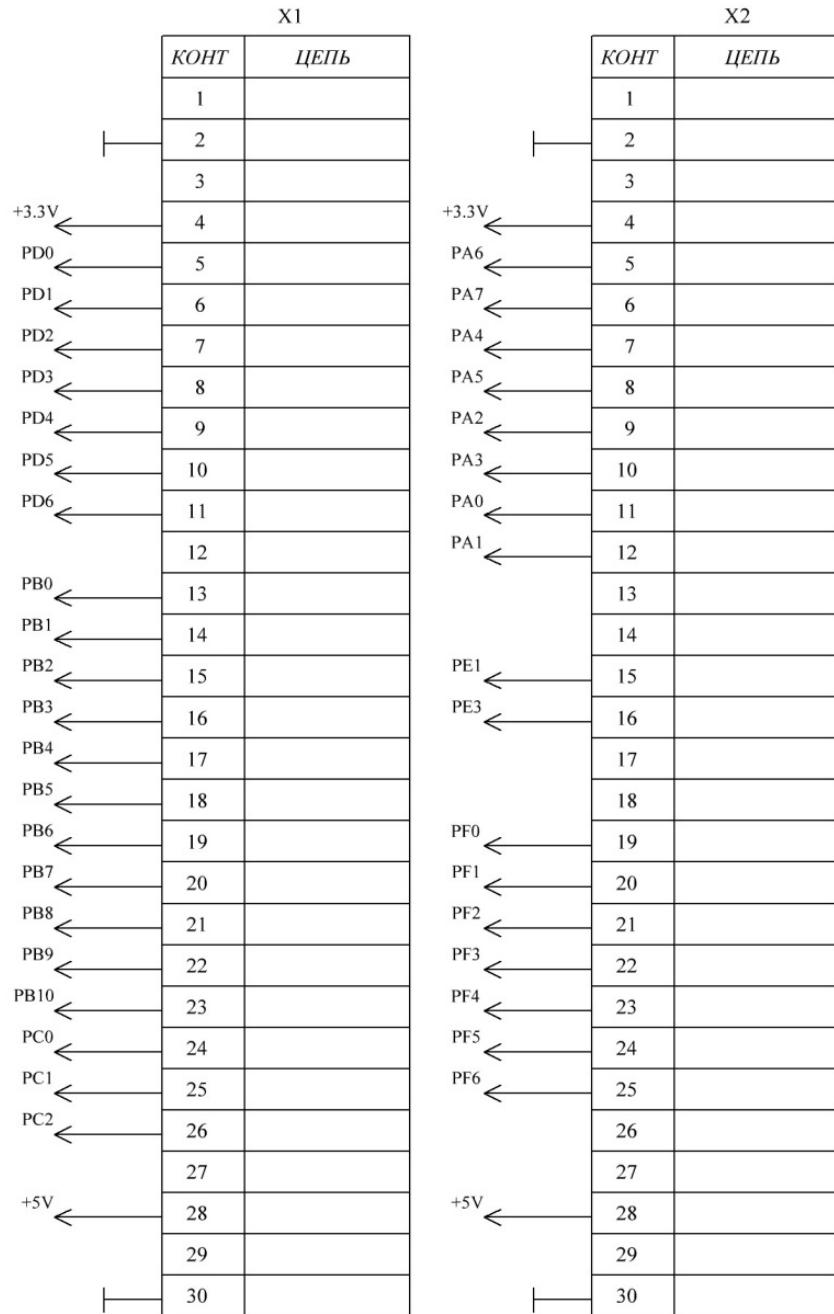


Рис. 2.1. Схема подключения разъемов

Таблица 2.1

Альтернативные функции

Вывод	Аналоговая функция ANALOG_EN=0	Цифровая функция			
		Порт IO MODE[1:0]=00	Основная MODE[1:0]=01	Альтернативная MODE[1:0]=10	Переопреде- ленная MODE [1:0]=11
Порт А					
PA0		PA0	DATA0	1) EXT_INT1	9) -
PA1		PA1	DATA1	TMR1_CH1	
PA2		PA2	DATA2	TMR1_CH1N	
PA3		PA3	DATA3	TMR1_CH2	
PA4		PA4	DATA4	TMR1_CH2N	
PA5		PA5	DATA5	TMR1_CH3	
PA6		PA6	DATA6	CAN1_TX	2) UART1_RXD
PA7		PA7	DATA7	CAN1_RX	UART1_TXD
PA8		PA8	DATA8	TMR1_CH3N	
PA9		PA9	DATA9	TMR1_CH4	
PA10		PA10	DATA10	nUART1DTR	10) TMR2_CH4N
PA11		PA11	DATA11	nUART1RTS	
PA12		PA12	DATA12	nUART1RI	
PA13		PA13	DATA13	nUART1DCD	
PA14		PA14	DATA14	nUART1DSR	
PA15		PA15	DATA15	nUART1CTS	
Порт В					
PB0		PB0 JA_TDO	DATA16	1) TMR3_CH1	UART1_TXD
PB1		PB1 JA_TMS	DATA17	TMR3_CH1N	UART2_RXD
PB2		PB2 JA_TCK	DATA18	TMR3_CH2	CAN1_TX
PB3		PB3 JA_TDI	DATA19	TMR3_CH2N	CAN1_RX
PB4		PB4 JA_TRST	DATA20	TMR3_BLK	TMR3_ETR
PB5		PB5	DATA21	UART1_TXD	10) TMR3_CH3
PB6		PB6	DATA22	UART1_RXD	
PB7		PB7	DATA23	nSIROUT1	
PB8		PB8	DATA24	COMP_OUT	7) TMR3_CH4N
PB9		PB9	DATA25	nSIRIN1	10) EXT_INT4
PB10		PB10	DATA26	EXT_INT2	9) nSIROUT1
PB11		PB11	DATA27	EXT_INT1	COMP_OUT

Продолжение таблицы 2.1

PB12		PB12	DATA28		SSP1_FSS		SSP2_FSS
PB13		PB13	DATA29		SSP1_CLK		SSP2_CLK
PB14		PB14	DATA30		SSP1_RXD		SSP2_RXD
PB15		PB15	DATA31		SSP1_TXD		SSP2_TXD

Порт C

PC0		PC0	READY 17)	1) 2)	SCL1	11)	SSP2_FSS
PC1		PC1	OE		SDA1	12)	SSP2_CLK
PC2		PC2	WE		TMR3_CH1		SSP2_RXD
PC3		PC3	BE0		TMR3_CH1N		SSP2_TXD
PC4		PC4	BE1		TMR3_CH2	12)	TMR1_CH1
PC5		PC5	BE2		TMR3_CH2N		TMR1_CH1N
PC6		PC6	BE3		TMR3_CH3		TMR1_CH2
PC7		PC7	CLOCK		TMR3_CH3N		TMR1_CH2N
PC8		PC8	CAN1_TX		TMR3_CH4		TMR1_CH3
PC9		PC9	CAN1_RX		TMR3_CH4N		TMR1_CH3N
PC10		PC10			TMR3_ETR	13)	TMR1_CH4
PC11		PC11			TMR3_BLK		TMR1_CH4N
PC12		PC12			EXT_INT2		TMR1_ETR
PC13		PC13			EXT_INT4	13)	TMR1_BLK
PC14		PC14			SSP2_FSS		CAN2_RX
PC15		PC15			SSP2_RXD		CAN2_TX

Порт D

PD0	ADC0_REF+	5	PD0 JB_TMS	TMR1_CH1N	3) 4) 3) 4)	UART2_RXD	14)	TMR3_CH1
PD1	ADC1_REF-		PD1 JB_TMS	TMR1_CH1		UART2_TXD	13)	TMR3_CH1N
PD2	ADC2		PD2 JB_TMS	BUSY1		SSP2_RXD		TMR3_CH2
PD3	ADC3		PD3 JB_TMS			SSP2_FSS		TMR3_CH2N
PD4	ADC4		PD4 JB_TMS	TMR1_ETR		nSIROUT2	14)	TMR3_BLK
PD5	ADC5		PD5	CLE		SSP2_CLK	13)	TMR2_ETR
PD6	ADC6		PD6	ALE		SSP2_TXD	13)	TMR2_BLK
PD7	ADC7		PD7	TMR1_BLK		nSIRIN2	14)	UART1_RX_D
PD8	ADC8		PD8	TMR1_CH4N		TMR2_CH1		UART1_TXD
PD9	ADC9		PD9	CAN2_TX		TMR2_CH1N		SSP1_FSS
PD10	ADC10		PD10	TMR1_CH2		TMR2_CH2		SSP1_CLK
PD11	ADC11		PD11	TMR1_CH2N		TMR2_CH2N		SSP1_RXD
PD12	ADC12		PD12	TMR1_CH3		TMR2_CH3		SSP1_TXD
PD13	ADC13		PD13	TMR1_CH3N		TMR2_CH3N		CAN1_TX
PD14	ADC14		PD14	TMR1_CH4		TMR2_CH4		CAN1_RX
PD15	ADC15		PD15	CAN2_RX		BUSY2	1)	EXT_INT3

Окончание таблицы 2.1

Порт E							
PE0	DAC2_OUT	6	PE0	ADDR16	1)	TMR2_CH1	¹⁵) CAN1_RX
PE1	DAC2_REF		PE1	ADDR17		TMR2_CH1N	CAN1_TX
PE2	COMP_IN1	7	PE2	ADDR18		TMR2_CH3	TMR3_CH1
PE3	COMP_IN2		PE3	ADDR19		TMR2_CH3N	TMR3_CH1N
PE4	COMP_REF+		PE4	ADDR20		TMR2_CH4N	TMR3_CH2
PE5	COMP_REF-		PE5	ADDR21		TMR2_BLK	TMR3_CH2N
PE6	OSC_IN32	8	PE6	ADDR22	4)	CAN2_RX	TMR3_CH3
PE7	OSC_OUT32		PE7	ADDR23		CAN2_TX	TMR3_CH3N
PE8	COMP_IN3		PE8	ADDR24	15)	TMR2_CH4	TMR3_CH4
PE9	DAC1_OUT		PE9	ADDR25		TMR2_CH2	TMR3_CH4N
PE10	DAC1_REF		PE10	ADDR26		TMR2_CH2N	TMR3_ETR
PE11			PE11	ADDR27		nSIRIN1	TMR3_BLK
PE12			PE12	ADDR28	16)	SSP1_RXD	UART1_RXD
PE13			PE13	ADDR29		SSP1_FSS	UART1_TXD
PE14			PD14	ADDR30	15)	TMR2_ETR	SCL1
PE15			PD15	ADDR31	9)	EXT_INT3	SDA1
Порт F							
PF0			PF0	ADDR0	1)	SSP1TXD	¹⁶) UART2_RXD
PF1			PF1	ADDR1		SSP1CLK	UART2_TXD
PF2			PF2	ADDR2		SSP1FSS	CAN2_RX
PF3			PF3	ADDR3		SSP1RXD	CAN2_TX
PF4	MODE[0]		PF4 MODE[0]	ADDR4		-	
PF5			PF5 MODE[1]	ADDR5			
PF6			PF6 MODE[2]	ADDR6	3)	TMR1_CH1	
PF7			PF7	ADDR7		TMR1_CH1N	TMR3_CH1
PF8			PF8	ADDR8		TMR1_CH2	TMR3_CH1N
PF9			PF9	ADDR9		TMR1_CH2N	TMR3_CH2
PF10			PF10	ADDR10		TMR1_CH3	TMR3_CH2N
PF11			PF11	ADDR11		TMR1_CH3N	TMR3_ETR
PF12			PF12	ADDR12		TMR1_CH4	SSP2_FSS
PF13			PF13	ADDR13		TMR1_CH4N	SSP2_CLK
PF14			PF14	ADDR14		TMR1_ETR	SSP2_RXD
PF15			PF15	ADDR15		TMR1_BLK	SSP2_TXD

На рис. 2.2 представлена функциональная схема линии порта ввода-вывода.

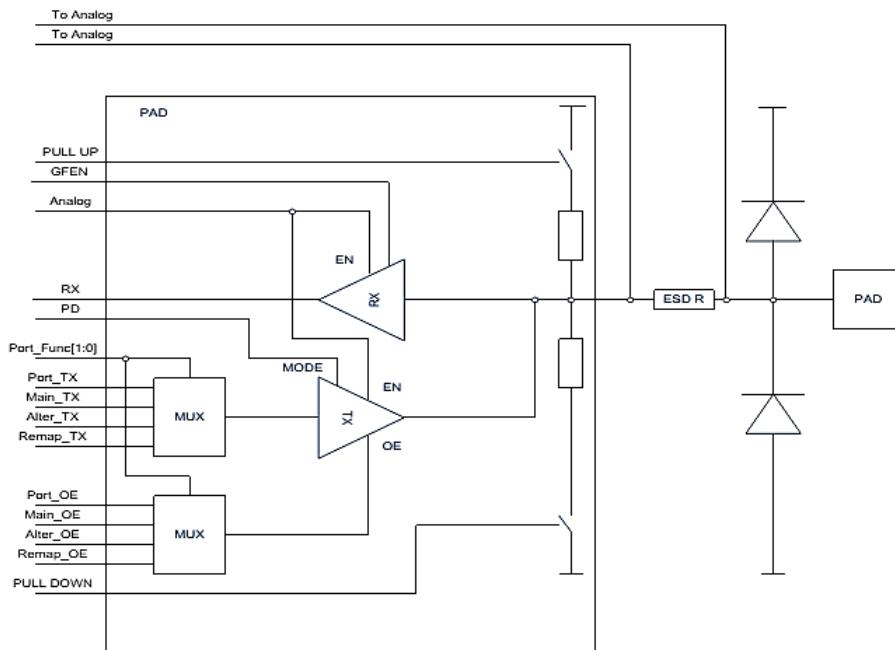


Рис. 2.2. Функциональная схема линии порта

Для настройки и работы портов ввода-вывода используются регистры, представленные в таблице 2.2.

Таблица 2.2

Описание регистров портов ввода-вывода

Название	Описание	
MDR_PORTA	Порт A	
MDR_PORTB	Порт B	
MDR_PORTC	Порт C	
MDR_PORTD	Порт D	
MDR_PORTE	Порт E	
MDR_PORTF	Порт F	
RXTX[15:0]	MDR_PORTx->RXTX	Данные порта
OE[15:0]	MDR_PORTx->OE	Направление порта
FUNC[31:0]	MDR_PORTx->FUNC	Режим работы порта
ANALOG[15:0]	MDR_PORTx->ANALOG	Аналоговый режим работы порта
PULL[31:0]	MDR_PORTx->PULL	Подтяжка порта
PD[31:0]	MDR_PORTx->PD	Режим работы выходного драйвера
PWR[31:0]	MDR_PORTx->PWR	Режим мощности передатчика
GFEN[15:0]	MDR_PORTx->GFEN	Режим работы входного фильтра

В таблицах 2.3–2.10 представлены назначения регистров портов ввода вывода.

Таблица 2.3

Данные порта MDR_PORTx->RXTX

Номер бита	31..16	15..0
Функциональное имя бита	-	RXTX[15:0]
Назначение	Зарезервировано	Данные для выдачи на выводы порта и для чтения

Таблица 2.4
Направление порта MDR_PORTx->OE

Номер бита	31..16	15..0
Функциональное имя бита	-	OE [15:0]
Назначение	Зарезервировано	Направление передачи данных на выводах порта: 1 – выход; 0 – вход

Таблица 2.5

Режим работы порта MDR_PORTx->FUNC

Номер бита	31..2	1..0
Функциональное имя бита	MODEx	MODE0[1:0]
Назначение	Аналогично MODE0 для остальных бит порта	Режим работы вывода порта: 00 –порт; 01 – основная функция; 10- альтернативная функция; 11- переопределённая функция

Таблица 2.6. Аналоговый режим работы порта MDR_PORTx->ANALOG

Номер бита	31..16	15..0
Функциональное имя бита		EN [15:0]
Назначение	Зарезервировано	Режим работы контроллера: 1 – аналоговый; 0 – цифровой

Таблица 2.7.

Подтяжка порта MDR_PORTx->PULL

Номер бита	31..16	15..0
Функциональное имя бита	UP[15:0]	DOWN[15:0]
Назначение	Разрешение подтяжки вверх: 0- подтяжка в питание выключена; 1- подтяжка в питание включена (есть подтяжка ~50 кОм)	Разрешение подтяжки вверх: 0- подтяжка в ноль выключена; 1- подтяжка в ноль включена (есть подтяжка ~50 кОм)

Таблица 2.8.

Режим работы выходного драйвера MDR_PORTx->PD

Номер бита	31..16	15..0
Функциональное имя бита	SHM[15:0]	PD[15:0]
Назначение	Режим работы входа: 0- триггер Шмитта включен гистерезис 200 мВ; 1- триггер Шмитта включен гистерезис 400 мВ	Режим работы выхода: 0- управляемый драйвер; 1- открытый сток

Таблица 2.9

Режим мощности передатчика MDR_PORTx->PWR

Номер бита	31..2	1..0
Функциональное имя бита	PWRx	PWR0[1:0]
Назначение	Аналогично PWR0 для остальных бит порта	Режим работы вывода порта: 00 – зарезервировано (передатчик отключен); 01 – медленный фронт (порядка 100 нс); 10 – быстрый фронт (порядка 20 нс); 11 – максимально быстрый фронт (порядка 10 нс)

Таблица 2.10

Режим работы входного фильтра MDR_PORTx->GFEN

Номер бита	31..16	15..0
Функциональное имя бита		GFEN[15:0]
Назначение	Зарезервировано	Режим работы входного фильтра: 1 – фильтр выключен; 0 – фильтр включен (фильтрация импульсов до 10 нс)

Пример программ настройки портов ввода-вывода

Пример 1. Мерцание светодиодов

Программа осуществляет мерцание светодиодов на выводах Pin 0 и Pin 1 порта C.

```
#include "MDR32Fx.h"
#include "MDR32F9Qx_port.c"
```

```

#include "MDR32F9Qx_rst_clk.c"
#include "system_MDR32F9Qx.c"

PORT_InitTypeDef PORT_InitStructure; //Объявление структуры
                                    //инициализации портов
int time;                         //переменная, необходимая для
                                    //формирования времени
                                    //свечения светодиодов

int main(void)
{
    //Разрешение тактирования периферии
    RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);

    //Настройка выводов для светодиодов
    //*****
    PORT_InitStructure.PORT_Pin=(PORT_Pin_0|PORT_Pin_1);
                                    //Выбор настраиваемого вывода
    PORT_InitStructure.PORT_OE = PORT_OE_OUT;
                                    //Настройка вывода порта на
                                    //режим выход (может быть
                                    //вход-PORT_OE_IN)
    PORT_InitStructure.PORT_FUNC = PORT_FUNC_PORT;
                                    //Функция порта - порт (так же
                                    //может быть: основная
                                    //функция-PORT_FUNC_MAIN,
                                    //альтернативная функция-
                                    //PORT_FUNC_ALTER,
                                    //переопределенная функция-
                                    //PORT_FUNC_OVERRIDE)
    PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL;
                                    //Настройка режима работы
                                    //вывода цифровой (может быть
                                    //аналоговый-
                                    //PORT_MODE_ANALOG)
    PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
                                    //Выбор скорости медленный
                                    //фронт(100нс) (может так же
                                    //быть отключен-
                                    //PORT_OUTPUT_OFF,
                                    //быстрый фронт(20нс)-
                                    //PORT_SPEED_FAST,
                                    //максимально быстрый
                                    //фронт(10нс)-
                                    //PORT_SPEED_MAXFAST)

    PORT_Init(MDR_PORTC, &PORT_InitStructure);
}

```

```

//Применение заполненной
//структурой к выбранному пор-
ту
/*************
while(1)
{
    time=2;                                //Переменная, отвечающая за
                                              //скорость мигания
    PORT_SetBits(MDR_PORTC, PORT_Pin_0);      //Установить лог "1" на 0
                                              //выводе порта С
    PORT_ResetBits(MDR_PORTC, PORT_Pin_1);     //Установить лог "0" на 1
                                              //выводе порта С
    delay(time);                            //Вызов функции временной
                                              //задержки
    PORT_ResetBits(MDR_PORTC, PORT_Pin_0);      //Установить лог "0" на 0
                                              //выводе порта С
    PORT_SetBits(MDR_PORTC, PORT_Pin_1);        //Установить лог "1" на 1
                                              //выводе порта С
    delay(time);                            //Вызов функции временной
                                              //задержки
}
int delay(int a) {for(int i=0; i<=100000*a; i++);}
                                              //Функция программной
                                              //временной задержки

```

Пример 2. Мерцание светодиодов с разной скоростью по опросу кнопок UP и DOWN

Программа осуществляет зажигание светодиодов на выводах Pin 0 и Pin 1.

```

#include "MDR32Fx.h"
#include "MDR32F9Qx_port.c"
#include "MDR32F9Qx_RST_CLK.c"
#include "system_MDR32F9Qx.c"

```

```

PORT_InitTypeDef PORT_InitStructure; //Обявление структуры инициа-
                                              //лизации портов

```

```

int time;                                //переменная, необходимая для
                                         //формирования времени
                                         //свечения светодиодов

int main(void)
{
    RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTB |
                      RST_CLK_PCLK_PORTC | RST_CLK_PCLK PORTE, ENABLE); //Разрешение тактирования
                                         //периферии

    //Настройка выводов для светодиодов
    //*****
    PORT_InitStructure.PORT_Pin = (PORT_Pin_0 | PORT_Pin_1); //Выбор настраиваемого вывода
    PORT_InitStructure.PORT_OE = PORT_OE_OUT; //Настройка вывода порта на
                                         //режим выход (может быть
                                         //вход-PORT_OE_IN)
    PORT_InitStructure.PORT_FUNC = PORT_FUNC_PORT; //Функция порта - порт (так же
                                         //может быть: основная
                                         //функция-PORT_FUNC_MAIN,
                                         //альтернативная функция-
                                         //PORT_FUNC_ALTER,
                                         //переопределенная функция-
                                         //PORT_FUNC_OVERRIDE)
    PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL; //Настройка режима работы
                                         //вывода цифровой (может быть
                                         //аналоговый-
                                         //PORT_MODE_ANALOG)
    PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW; //Выбор скорости медленный
                                         //фронт(100нс) (может так же
                                         //быть отключен-
                                         //PORT_OUTPUT_OFF,
                                         //быстрый фронт(20нс)-
                                         //PORT_SPEED_FAST,
                                         //максимально быстрый
                                         //фронт(10нс)-
                                         //PORT_SPEED_MAXFAST)
    PORT_Init(MDR_PORTC, &PORT_InitStructure); //Применение заполненной
                                         //структурьы к выбранному
                                         //порту
    //*****
}

```

```

//Настройка выводов для кнопок Up и Down
//*****************************************************************************
PORT_InitStructure.PORT_Pin = (PORT_Pin_5);
                                //Выбор настраиваемого вывода
PORT_InitStructure.PORT_OE = PORT_OE_IN;
                                //Настройка вывода порта на
                                //режим выход (может быть
                                //вход-PORT_OE_IN)
PORT_InitStructure.PORT_FUNC = PORT_FUNC_PORT;
                                //Функция порта - порт (так же
                                //может быть: основная
                                //функция-PORT_FUNC_MAIN,
                                //альтернативная функция-
                                //PORT_FUNC_ALTER,
                                //переопределенная функция-
                                //PORT_FUNC_OVERRIDE)
PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL;
                                //Настройка режима работы вы-
                                //вода цифровой (может быть
                                //аналоговый-
                                PORT_MODE_ANALOG)
PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
                                //Выбор скорости медленный
                                //фронт(100нс) (может так же
                                //быть отключен-
                                //PORT_OUTPUT_OFF,
                                //быстрый фронт(20нс)-
                                //PORT_SPEED_FAST,
                                //максимально быстрый
                                //фронт(10нс)-
                                //PORT_SPEED_MAXFAST)
PORT_Init(MDR_PORTB, &PORT_InitStructure);
                                //Применение заполненной
                                //структуры к выбранному
                                //порту
PORT_InitStructure.PORT_Pin = (PORT_Pin_1);
                                //Выбор настраиваемого
                                //вывода
PORT_InitStructure.PORT_OE = PORT_OE_IN;
                                //Настройка вывода порта на
                                //режим выход (может быть
                                //вход-PORT_OE_IN)
PORT_InitStructure.PORT_FUNC = PORT_FUNC_PORT;
                                //Функция порта - порт (так же
                                //может быть: основная
                                //функция-PORT_FUNC_MAIN,

```

```

//альтернативная функция-
//PORT_FUNC_ALTER,
//переопределенная функция-
//PORT_FUNC_OVERRIDE

PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL;
//Настройка режима работы
//вывода цифровой (может быть
//аналоговый-
//PORT_MODE_ANALOG)

PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
//Выбор скорости медленный
//фронт(100нс) (может так же
//быть отключен-
//PORT_OUTPUT_OFF,
//быстрый фронт(20нс)-
//PORT_SPEED_FAST,
//максимально быстрый
//фронт(10нс)-
//PORT_SPEED_MAXFAST)

PORT_Init(MDR_PORTE, &PORT_InitStructure);
//Применение заполненной
//структуры к выбранному
//порту

//*****
while(1)
{
    time=3; //Средняя скорость мигания
    if (PORT_ReadInputDataBit(MDR_PORTB, PORT_Pin_5) ==
        Bit_RESET)
        //Если выполнение команды
        //чтения вывода 5 порта В
        //возвращает 0
    {
        time=1; //Высокая скорость мигания
    }
    if (PORT_ReadInputDataBit(MDR_PORTE, PORT_Pin_1) ==
        Bit_RESET)
        //Если выполнение команды
        //чтения вывода 1 порта Е
        //возвращает 0
    {
        time=6; //Низкая скорость мигания
    }
}

```

```

PORT_SetBits(MDR_PORTC, PORT_Pin_0);
    //Установить лог "1" на 0
    //выводе порта С
PORT_ResetBits(MDR_PORTC, PORT_Pin_1);
    //Установить лог "0" на 1
    //выводе порта С
delay(time);
    //Вызов функции временной
    //задержки
PORT_ResetBits(MDR_PORTC, PORT_Pin_0);
    //Установить лог "0" на 0
    //выводе порта С
PORT_SetBits(MDR_PORTC, PORT_Pin_1);
    //Установить лог "1" на 1
    //выводе порта С
delay(time);
    //Вызов функции временной за-
    //держки
}
}

int delay(int a) {for(int i=0; i<=100000*a; i++);}

//Функция программной
//временной задержки

```

Пример 3. Изменение скорости мерцания светодиодов с помощью внешних прерываний

Прерывания в данном МК генерируются по уровню. Для проверки работоспособности программы, необходимо выводы PA0 и PB10 подключить к кнопкам.

```

#include "MDR32Fx.h"
#include "MDR32F9Qx_port.c"
#include "MDR32F9Qx_rst_clk.c"
#include "MDR32F9Qx_it.c"
#include "system_MDR32F9Qx.c"

PORT_InitTypeDef PORT_InitStructure; //Обявление структуры
                                    //инициализации портов
int time;                         //переменная, необходимая для
                                    //формирования времени
                                    //свечения светодиодов
void delay(int a) {for(int i=0; i<=10000*a; i++);}

//Функция программной
//временной задержки

```

```

int main(void)
{
    RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTA |
RST_CLK_PCLK_PORTB | RST_CLK_PCLK_PORTC |
RST_CLK_PCLK PORTE, ENABLE); //Разрешение тактирования
периферии
    SCB->VTOR = 0x08000000; // Включение таблицы векторов

    //Настройка выводов для светодиодов
    //*****
    PORT_InitStructure.PORT_Pin = (PORT_Pin_0 | PORT_Pin_1);
                                //Выбор настраиваемого вывода
    PORT_InitStructure.PORT_OE = PORT_OE_OUT;
                                //Настройка вывода порта на
                                //режим выход (может быть
                                //вход-PORT_OE_IN)
    PORT_InitStructure.PORT_FUNC = PORT_FUNC_PORT;
                                //Функция порта - порт (так же
                                //может быть: основная
                                //функция-PORT_FUNC_MAIN,
                                //альтернативная функция-
                                //PORT_FUNC_ALTER,
                                //переопределенная функция-
                                //PORT_FUNC_OVERRIDE)
    PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL;
                                //Настройка режима работы
                                //вывода цифровой (может быть
                                //аналоговый-
                                //PORT_MODE_ANALOG)
    PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
                                //Выбор скорости медленный
                                //фронт(100нс) (может так же
                                //быть отключен-
                                //PORT_OUTPUT_OFF,
                                //быстрый фронт(20нс)-
                                //PORT_SPEED_FAST,
                                //максимально быстрый
                                //фронт(10нс)-
                                //PORT_SPEED_MAXFAST)
    PORT_Init(MDR_PORTC, &PORT_InitStructure);
                                //Применение заполненной
                                //структурь к выбранному пор-
                                //ту
    //*****

```

```

//Настройка вывода 0 порта А (альтернативная функция EXT_INT1)
//*****
PORT_InitStructure.PORT_Pin = (PORT_Pin_0);
//Выбор настраиваемого вывода
PORT_InitStructure.PORT_OE = PORT_OE_IN;
//Настройка вывода порта на
//режим вход (может быть
//выход-PORT_OE_OUT)
PORT_InitStructure.PORT_FUNC = PORT_FUNC_ALTER;
//Функция порта -
//альтернативная функция (так
//же может быть: порт -
//PORT_FUNC_PORT, основная
//функция-PORT_FUNC_MAIN,
//переопределенная функция-
//PORT_FUNC_OVERRIDE)
PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL;
//Настройка режима работы
//вывода цифровой (может быть
//аналоговый-
//PORT_MODE_ANALOG)
PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
//Выбор скорости медленный
//фронт(100нс) (может так же
//быть отключен-
//PORT_OUTPUT_OFF,
//быстрый фронт(20нс)-
//PORT_SPEED_FAST,
//максимально быстрый
//фронт(10нс)-
//PORT_SPEED_MAXFAST)
PORT_Init(MDR_PORTA, &PORT_InitStructure);
//Применение заполненной
//структуры к выбранному пор-
//ту
//*****

//Настройка вывода 10 порта В (альтернативная функция EXT_INT2)
//*****
PORT_InitStructure.PORT_Pin = (PORT_Pin_10);
//Выбор настраиваемого вывода
PORT_InitStructure.PORT_OE = PORT_OE_IN;
//Настройка вывода порта на
//режим вход (может быть
//выход-PORT_OE_OUT)
PORT_InitStructure.PORT_FUNC = PORT_FUNC_ALTER;
//Функция порта -

```

```

        //альтернативная функция (так
        //же может быть: порт -
        //PORT_FUNC_PORT, основная
        //функция-PORT_FUNC_MAIN,
        //переопределенная функция-
        //PORT_FUNC_OVERRIDE)

PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL;
        //Настройка режима работы
        //вывода цифровой (может быть
        //аналоговый-
        //PORT_MODE_ANALOG)

PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
        //Выбор скорости медленный
        //фронт(100нс) (может так же
        //быть отключен-
        //PORT_OUTPUT_OFF,
        //быстрый фронт(20нс)-
        //PORT_SPEED_FAST,
        //максимально быстрый
        //фронт(10нс)-
        //PORT_SPEED_MAXFAST)

PORT_Init(MDR_PORTB, &PORT_InitStructure);
        //Применение заполненной
        //структуре к выбранному порту

//*****
NVIC_ClearPendingIRQ(EXT_INT1 IRQn);
        //Очистка бита ожидания
        //внешнего прерывания
        //EXT_INT1

NVIC_ClearPendingIRQ(EXT_INT2 IRQn);
        //Очистка бита ожидания
        //внешнего прерывания
        //EXT_INT2

__enable_irq();           //Глобальное разрешение
                        //прерываний

NVIC_EnableIRQ(EXT_INT1 IRQn); //Разрешения внешнего
                            //прерывания EXT_INT1

NVIC_EnableIRQ(EXT_INT2 IRQn); //Разрешения внешнего
                            //прерывания EXT_INT2

time=60;                  //Задание начального значения
                        //временной задержки

while(1)
{
    PORT_SetBits(MDR_PORTC, PORT_Pin_0);
}

```

```

        //Установить лог "1" на 0
        //выводе порта С
    PORT_ResetBits(MDR_PORTC, PORT_Pin_1);
        //Установить лог "0" на 1
        //выводе порта С
    delay(time); //Вызов функции временной
                  //задержки
    PORT_ResetBits(MDR_PORTC, PORT_Pin_0);
        //Установить лог "0" на 0
        //выводе порта С
    PORT_SetBits(MDR_PORTC, PORT_Pin_1);
        //Установить лог "1" на 1
        //выводе порта С
    delay(time); //Вызов функции временной
                  //задержки
}
}

void EXT_INT1_IRQHandler(void) //Вектор внешнего прерывания
//EXT_INT1
{
time=time+10; //Увеличиваем время задержки,
//тем самым уменьшаем
//скорость мигания светодиодов
if(time==120) time=60; //Возвращение в исходное
//состояние
NVIC_ClearPendingIRQ(EXT_INT1 IRQn); //Очистка бита ожидания
//внешнего прерывания
//EXT_INT1
}

void EXT_INT2_IRQHandler(void) //Вектор внешнего прерывания
//EXT_INT2
{
time=time-10; //Уменьшаем время задержки,
//тем самым увеличиваем
//скорость мигания светодиодов
if(time==0) time=60; //Возвращение в исходное
//состояние
NVIC_ClearPendingIRQ(EXT_INT2 IRQn); //Очистка бита ожидания
//внешнего прерывания
//EXT_INT2
}

```

Лабораторная работа №1. Порты ввода-вывода

Цель работы

Целью работы является изучение основ работы портов ввода-вывода микроконтроллера K1986BE92QI.

Оборудование

1. Компьютер IBM PC.
2. Отладочная плата микроконтроллера K1986BE92QI.
3. Лабораторный макет.
4. Источник питания 12В, 1А.

Программа работы

1. Создать проект в среде IAR Embedded System. Проект создается в каталоге D:\OMT\(*номер группы*).
2. Написать и проверить работу программы мерцания светодиодов с переменной частотой. Изменение частоты осуществлять кнопками UP и DOWN.
3. Подключить лабораторный макет к отладочной плате.
4. Организовать на светодиодах лабораторного макета “бегущий 0”.
5. Организовать на светодиодах лабораторного макета “бегущую 1”.
6. Организовать на светодиодах лабораторного макета светофор.
7. Вывести на светодиоды числа в двоичном коде от 0÷32 с задержкой в 1 с.
8. Организовать на светодиодах лабораторного макета “эквалайзер”.
9. Организовать на выводе порта импульсы с частотой 5 кГц и коэффициентом заполнения 0,25.
10. Организовать на выводе порта импульсы с частотой 50 кГц и коэффициентом заполнения 0,15.
11. Организовать на выводе порта импульсы с частотой 100 кГц и коэффициентом заполнения 0,75.
12. Вывести на ЖК дисплей отладочного макета свою.
13. Вывести на ЖК дисплей отладочного макета название кафедры.
14. Вывести на ЖК дисплей отладочного макета фамилию и инициалы преподавателя.
15. Вывести на ЖК дисплей отладочного макета название университета.
16. Организовать на светодиодах лабораторного макета “бегущий 0” с изменением скорости по внешним прерываниям.
17. Организовать на светодиодах лабораторного макета “бегущую 1” с изменением скорости по внешним прерываниям.

Глава 3. ТАЙМЕРЫ МИКРОКОНТРОЛЛЕРА K1986BE92QI

Описание таймеров микроконтроллера

Все блоки таймеров выполнены на основе 16-битного перезагружаемого счетчика, который синхронизируется с выхода 16-битного предделителя. Перезагружаемое значение хранится в отдельном регистре. Счет может быть прямой, обратный или двунаправленный (сначала прямой до определенного значения, а затем обратный).

Каждый из трех таймеров микроконтроллера содержит 16-битный счетчик, 16-битный предделитель частоты и 4-канальный блок захвата/сравнения. Их можно синхронизировать системной синхронизацией, внешними сигналами или другими таймерами. Помимо составляющего основу таймера счетчика, в каждый блок таймера также входит четырехканальный блок захвата/сравнения. Данный блок выполняет как стандартные функции захвата и сравнения, так и ряд специальных функций. Таймеры с 4 каналами схем захвата и ШИМ с функциями формирования «мертвой зоны» и аппаратной блокировки. Каждый из таймеров может генерировать прерывания и запросы DMA.

Особенности таймеров микроконтроллера:

- 16-битный счетчик; счёт прямой, обратный или двунаправленный;
- 16-разрядный программируемый предварительный делитель частоты;
- до четырех независимых 16-битных каналов захвата на один таймер. Каждый из каналов захвата может захватить (скопировать) текущее значение таймера при изменении некоторого входного сигнала. В случае захвата имеется дополнительная возможность генерировать прерывание и/или запрос DMA;
- четыре 16-битных регистра сравнения (совпадения), которые позволяют осуществлять непрерывное сравнение, с дополнительной возможностью генерировать прерывание и/или запрос DMA при совпадении;
- имеется до четырех внешних выводов, соответствующих регистрам совпадения со следующими возможностями: сброс в НИЗКИЙ уровень при совпадении; установка в ВЫСОКИЙ уровень при совпадении; переключение (инвертирование) при совпадении; при совпадении состояние выхода не изменяется; переключение при некотором условии.

На рисунке 3.1 представлена структурная схема таймеров общего назначения.

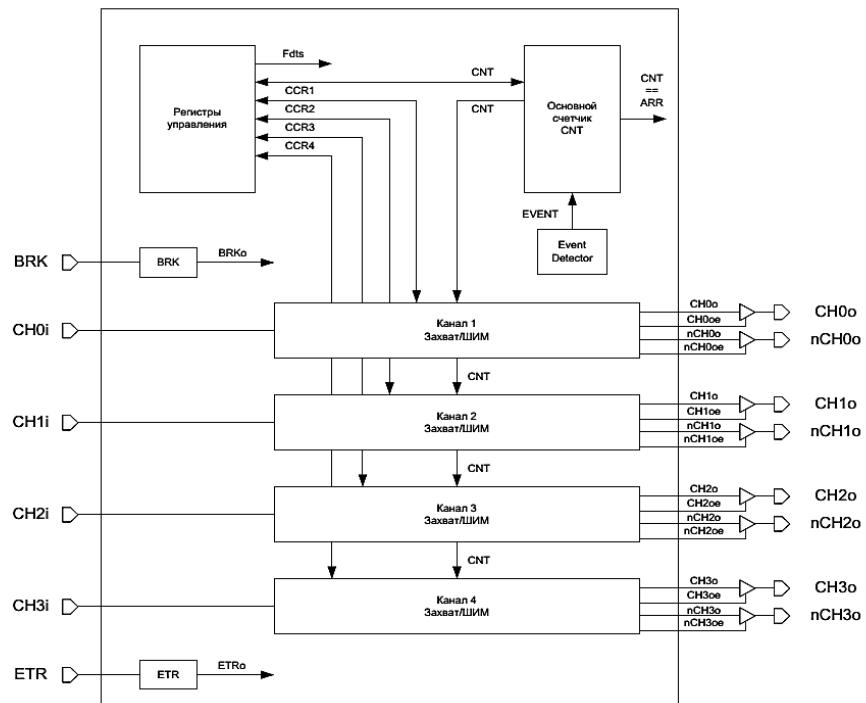


Рис. 3.1. Структурная схема таймеров общего назначения

В таблице 3.1 представлены регистры управления и настройки таймеров.

Таблица 3.1. Регистры портов ввода-вывода

Название	Описание
MDR_TIMER1	Контроллер Timer1
MDR_TIMER 2	Контроллер Timer2
MDR_TIMER 3	Контроллер Timer3
MDR_TIMERx->CNT[15:0]	MDR_TIMERx->CNT Основной счетчик таймера
MDR_TIMERx->PSG[15:0]	MDR_TIMERx->PSG Делитель частоты при счете основного счетчика
MDR_TIMERx->ARR[15:0]	MDR_TIMERx->ARR Основание счета основного счетчика
MDR_TIMERx->CNTRL[7:0]	MDR_TIMERx->CNTRL Регистр управления основного счетчика
CCR1[15:0]	MDR_TIMERx->CCRY Регистр сравнения, захвата для 1 канала таймера
CCR2[15:0]	MDR_TIMERx->CCRY Регистр сравнения, захвата для 2 канала таймера
CCR3[15:0]	MDR_TIMERx->CCRY Регистр сравнения, захвата для 3 канала таймера
CH1_CNTRL[15:0]	MDR_TIMERx->CHy_CNTRL Регистр управления для 1 канала таймера
CH2_CNTRL[15:0]	MDR_TIMERx->CHy_CNTRL Регистр управления для 2 канала таймера

Продолжение таблицы 3.1

CH3_CNTRL[15:0]	MDR_TIMERx->CHy_CNTRL Регистр управления для 3 канала таймера
CH4_CNTRL[15:0]	MDR_TIMERx->CHy_CNTRL Регистр управления для 4 канала таймера
CH1_CNTRL1[15:0]	MDR_TIMERx->CHy_CNTRL1 Регистр управления 1 для 1 канала таймера
CH2_CNTRL1[15:0]	MDR_TIMERx->CHy_CNTRL1 Регистр управления 1 для 2 канала таймера
CH3_CNTRL1[15:0]	MDR_TIMERx->CHy_CNTRL1 Регистр управления 1 для 3 канала таймера
CH4_CNTRL1[15:0]	MDR_TIMERx->CHy_CNTRL1 Регистр управления 1 для 4 канала таймера
CH1_DTG[15:0]	MDR_TIMERx->CHy_DTG Регистр управления DTG для 1 канала таймера
CH2_DTG[15:0]	MDR_TIMERx->CHy_DTG Регистр управления DTG для 2 канала таймера
CH3_DTG[15:0]	MDR_TIMERx->CHy_DTG Регистр управления DTG для 3 канала таймера
CH4_DTG[15:0]	MDR_TIMERx->CHy_DTG Регистр управления DTG для 4 канала таймера
BRKETR_CNTRL[15:0]	MDR_TIMERx->BRKETR_CNTRL Регистр управления входом BRK и ETR
STATUS[15:0]	MDR_TIMERx->STATUS Регистр статуса таймера
IE[15:0]	MDR_TIMERx->IE Регистр разрешения прерывания таймера
DMA_RE[15:0]	MDR_TIMERx->DMA_RE Регистр разрешения запросов DMA от прерываний таймера
CH1_CNTRL2[15:0]	MDR_TIMERx->CHy_CNTRL2 Регистр управления 2 для 1 канала таймера
CH2_CNTRL2[15:0]	MDR_TIMERx->CHy_CNTRL2 Регистр управления 2 для 2 канала таймера
CH3_CNTRL2[15:0]	MDR_TIMERx->CHy_CNTRL2 Регистр управления 2 для 3 канала таймера
CH4_CNTRL2[15:0]	MDR_TIMERx->CHy_CNTRL2 Регистр управления 2 для 4 канала таймера
CCR11[15:0]	MDR_TIMERx->CCRy1 Регистр сравнения 1, захвата для 1 канала таймера
CCR21[15:0]	MDR_TIMERx->CCRy1 Регистр сравнения 1, захвата для 2 канала таймера
CCR31[15:0]	MDR_TIMERx->CCRy1 Регистр сравнения 1, захвата для 3 канала таймера
CCR41[15:0]	MDR_TIMERx->CCRy1 Регистр сравнения 1, захвата для 4 канала таймера

Режимы счета таймеров общего назначения

- **Счет вверх:** CNT_MODE = 00, DIR = 0 (пример: счет вверх от 0 до 0x13, стартовое значение 0x04)

```
MDR_TIMERx->CNTRL=0x00000000;      //Настраиваем работу основного
                                         //счетчика
MDR_TIMERx->CNT=0x00000004;          //Начальное значение счетчика
MDR_TIMERx->PSG=0x00000000;          //Предделитель частоты
MDR_TIMERx->ARR=0x00000013;          //Период счета
MDR_TIMERx->CNTRL=0x00000001;          //Счет вверх по TIM_CLK.
```

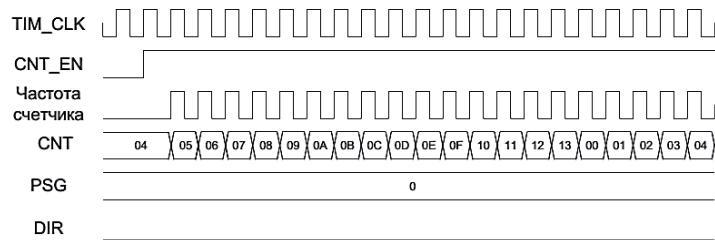


Рис. 3.2. Диаграмма счёта от 0 до 0x13

- **Счет вниз:** CNT_MODE = 00, DIR = 1 (пример: счет вниз от 0x13 до 0, стартовое значение 0x04)

```
MDR_TIMERx->CNTRL=0x00000000;      //Настраиваем работу основного
                                         //счетчика
MDR_TIMERx->CNT=0x00000004;          //Начальное значение счетчика
MDR_TIMERx->PSG=0x00000000;          //Предделитель частоты
MDR_TIMERx->ARR=0x00000013;          //Период счета
MDR_TIMERx->CNTRL=0x00000009;          //Счет вниз по TIM_CLK.
```

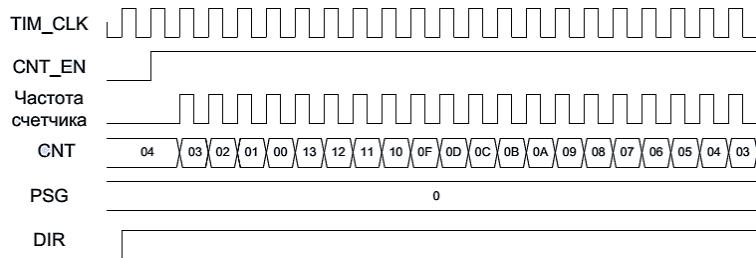


Рис. 3.3. Диаграмма счёта от 0x13 до 0

- **Счет вверх/вниз:** CNT_MODE = 01, DIR = 0

```
MDR_TIMERx->CNTRL=0x00000000;      //Настраиваем работу основного
                                         //счетчика
MDR_TIMERx->CNT=0x00000004;          //Начальное значение счетчика
```

```

MDR_TIMERx->PSG=0x00000000;      //Предделитель частоты
MDR_TIMERx->ARR=0x00000013;      //Период счета
MDR_TIMERx->CNTRL=0x00000041;    //Счет вверх/вниз по TIM_CLK.

```

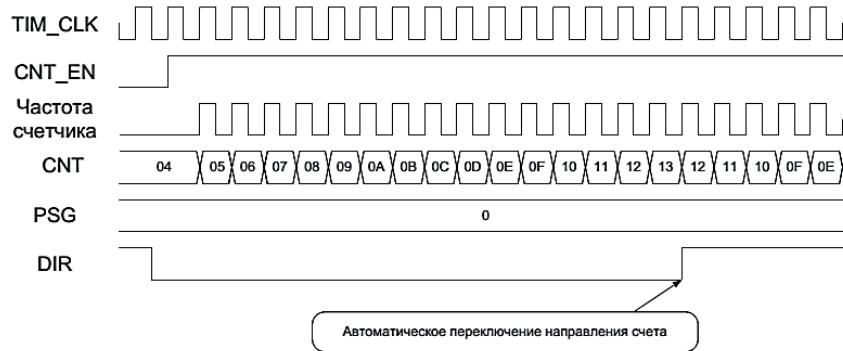


Рис.3.4. Диаграмма счёта вверх/вниз

- **Счет вверх/вниз: CNT_MODE = 01, DIR = 1**

```

MDR_TIMERx->CNTRL=0x00000000;      //Настраиваем работу основного
                                         //счетчика
MDR_TIMERx->CNT=0x00000004;        //Начальное значение счетчика
MDR_TIMERx->PSG=0x00000000;        //Предделитель частоты
MDR_TIMERx->ARR=0x00000013;        //Период счета
MDR_TIMERx->CNTRL=0x00000049;    //Счет вверх/вниз по TIM_CLK.

```

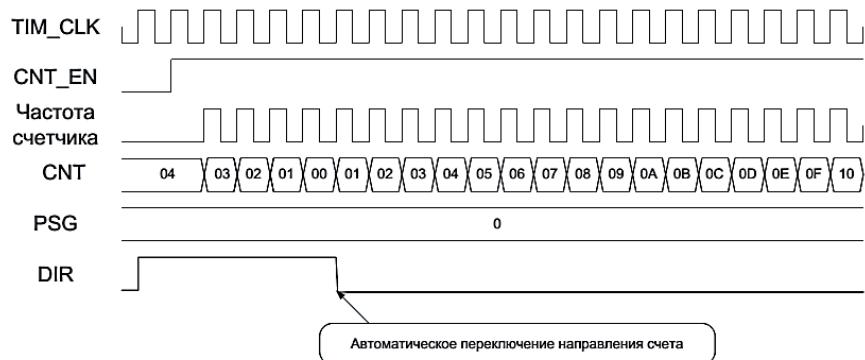


Рис.3.5. Диаграмма счёта вверх/вниз

Пример программ настройки таймера 1

Пример 1. Прерывание по таймеру

Программа осуществляет мерцание светодиодов по прерыванию таймера.

```
#include "MDR32Fx.h"  
#include "MDR32F9Qx_port.c"  
#include "MDR32F9Qx_timer.c"  
#include "MDR32F9Qx_rst_clk.c"  
#include "MDR32F9Qx_it.c"  
#include "system_MDR32F9Qx.c"
```

PORT_InitTypeDef PORT_InitStructure; //Обявление структуры инициализации портов

TIMER_CntInitTypeDef TIM_CntInitStructure; //Обявление структуры инициализации таймеров

```
int main(void)
```

```
{
```

```
    RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTA |  
    RST_CLK_PCLK_PORTB | RST_CLK_PCLK_PORTC |  
    RST_CLK_PCLK PORTE | RST_CLK_PCLK_TIMER1, ENABLE);  
    //Разрешение тактирования периферии
```

```
    SCB->VTOR = 0x08000000; //Включение таблицы векторов
```

```
    MDR_RST_CLK->TIM_CLOCK = 0x01000000; //Включение тактирования таймера
```

```
//Настройка выводов для светодиодов
```

```
//*****
```

```
    PORT_InitStructure.PORT_Pin = (PORT_Pin_0 | PORT_Pin_1); //Выбор настраиваемого вывода
```

```
PORT_InitStructure.PORT_OE = PORT_OE_OUT; //Настройка вывода порта на режим выход (может быть вход-PORT_OE_IN)
```

```
PORT_InitStructure.PORT_FUNC = PORT_FUNC_PORT; //Функция порта - порт (так же может быть: основная функция-PORT_FUNC_MAIN, альтернативная функция-PORT_FUNC_ALTER, переопределенная функция-PORT_FUNC_OVERRIDE)
```

```
PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL; //Настройка режима работы вывода цифровой (может быть аналоговый-PORT_MODE_ANALOG)
```

```
PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW; //Выбор скорости медленный фронт(100нс) (может так же быть отключен-PORT_OUTPUT_OFF, быстрый фронт(20нс)-PORT_SPEED_FAST, максимально быстрый фронт(10нс)-PORT_SPEED_MAXFAST)
```

```
PORT_Init(MDR_PORTC, &PORT_InitStructure); //Применение заполненной структуры к выбранному порту
```

```
*****
```

```
//Настройка таймера
```

```
*****
```

```
TIM_CntInitStructure.TIMER_Prescaler = 0xFFFF; //Предварительный делитель
```

```
TIM_CntInitStructure.TIMER_Period = 0x1F; //Период счета
```

```
TIM_CntInitStructure.TIMER_CounterMode = TIMER_CntMode_ClkFixedDir; //Режим счета
```

```
TIM_CntInitStructure.TIMER_CounterDirection = TIMER_CntDir_Up; //Счет вверх
```

```
TIMER_CntInit (MDR_TIMER1,&TIM_CntInitStructure);
```

```
*****
```

```

NVIC_ClearPendingIRQ(TIMER1_IRQn); //Очистка бита ожидания
внешнего прерывания TIMER1

__enable_irq(); //Глобальное разрешение прерываний

NVIC_EnableIRQ(TIMER1_IRQn); //Разрешения внешнего прерывания
TIMER1

TIMER_ITConfig(MDR_TIMER1, TIMER_STATUS_CNT_ARR, ENABLE); //Настройка прерываний таймера

TIMER_Cmd(MDR_TIMER1, ENABLE); //Включение таймера

while(1)

{

}

void Timer1_IRQHandler(void)

{

    NVIC_ClearPendingIRQ(TIMER1_IRQn); //Очистка бита ожидания
внешнего прерывания TIMER1

    TIMER_ClearFlag(MDR_TIMER1, TIMER_STATUS_CNT_ARR);
//Очистка флага достижения счетчиком значения периода

    MDR_PORTC->RXTX ^= 0x03; //Инверсия состояния 0 и 1 выводов
порта С

}

```

Пример 2. Настройка режима работы TIMER1 на работу в режиме широтно-импульсной модуляции (ШИМ)

```
#include "MDR32F9Qx_config.h"
```

```

#include "MDR32Fx.h"
#include "MDR32F9Qx_timer.h"
#include "MDR32F9Qx_rst_clk.h"
#include "MDR32F9Qx_port.h"
#include "MDR32F9Qx_it.c"
#include "system_MDR32F9Qx.c"

*****Объявление структур*****
TIMER_CntInitTypeDef sTIM_CntInit;
TIMER_ChnInitTypeDef sTIM_ChnInit;
TIMER_ChnOutInitTypeDef sTIM_ChnOutInit;
PORT_InitTypeDef PORT_InitStructure;
TIMER_CntInitTypeDef TIM_CntInitStructure;
*****



void main(void)

{
    *****Настройка тактирования*****
    RST_CLK_DeInit();                                //Сброс системы
                                                    //тактирования
    RST_CLK_CPU_PLLconfig(RST_CLK_CPU_PLLsrcHSIdiv2,0); //Инициализация системы
                                                    //тактирования
    RST_CLK_PCLKcmd((RST_CLK_PCLK_RST_CLK |
    RST_CLK_PCLK_TIMER1),ENABLE); //Тактирование TIMER1
    RST_CLK_PCLKcmd((RST_CLK_PCLK_PORTA), ENABLE); //Тактирование PORTA
    SCB->AIRCR = 0x05FA0000 | ((uint32_t)0x500); //Настройка регистра
                                                    //управления
                                                    //прерываниями и
                                                    //программного сброса
    SCB->VTOR = 0x08000000; // Включение таблицы векторов
    MDR_RST_CLK->TIM_CLOCK = 0x01000000; //Включение
                                                    //тактирования таймера
    *****



    *****Настройка вывода PA1*****
    PORT_InitStructure.PORT_Pin = PORT_Pin_1;
}

```

```

                //Выбор настраиваемого
                //вывода
PORT_InitStructure.PORT_OE = PORT_OE_OUT;
                //Настройка вывода на
                //выход
PORT_InitStructure.PORT_FUNC = PORT_FUNC_ALTER;
                //Функция вывода -
                //альтернативная
PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL;
                //Режим работы - цифровой
PORT_InitStructure.PORT_SPEED = PORT_SPEED_FAST;
                //Высокая скорость работы
                //короткие фронты)
PORT_Init(MDR_PORTA, &PORT_InitStructure);
                //Заполнение структуры
                //настройки PORTA
/*****************************************/

```

TIMER_DeInit(MDR_TIMER1); //Сброс всех настроек
 //TIMER1
 TIMER_BRGInit(MDR_TIMER1,TIMER_HCLKdiv1);

 /******Настройка конфигураций TIMER1*****/
 sTIM_CntInit.TIMR_Prescaler = 0x20; //Выбор предделителя
 //частоты
 sTIM_CntInit.TIMR_Period = 0xFF; //Период счета таймера
 sTIM_CntInit.TIMR_CounterMode = TIM-
 ER_CntMode_ClkFixedDir; //Направление счета не-
 изменно
 sTIM_CntInit.TIMR_CounterDirection = TIMR_CntDir_Up;
 //Режим счета вверх
 sTIM_CntInit.TIMR_EventSource = TIMR_EvSrc_None;
 //Выбор источника
 //прерываний - отсутствует
 sTIM_CntInit.TIMR_FilterSampling = TIM-
 ER_FDTS_TIMER_CLK_div_1;
 sTIM_CntInit.TIMR_ETR_FilterConf = TIM-
 ER_Filter_1FF_at_TIMER_CLK;
 sTIM_CntInit.TIMR_ARR_UpdateMode= TIM-
 ER_ARR_Update_On_CNT_Overflow; //Обновление ARR при
 //переполнении CNT

```

sTIM_CntInit.TIMR_ETR_Prescaler = TIM-
ER_ETR_Prescaler_None;           //Без деления входной
                                  //частоты
sTIM_CntInit.TIMR_ETR_Polarity = TIM-
ER_ETRPolarity_NonInverted;     //Нет инверсии входного
                                  //ETR
sTIM_CntInit.TIMR_BRK_Polarity = TIM-
ER_BRK_Polarity_NonInverted;    //Нет инверсии входного
                                  //BRK
TIMR_CntInit(MDR_TIMR1,&sTIM_CntInit); //Заполнение
                                         //структуры настройки
                                         //таймера
/****************************************/

*****Настройка TIMER1 на режим ШИМ*****
TIMER_ChnStructInit(&sTIM_ChnInit); //Объявление структуры
                                         //настройки TIMER1
sTIM_ChnInit.TIMR_CH_Number = TIMER_CHANNEL1;
                                         //Выбор канала (PA1)
sTIM_ChnInit.TIMR_CH_Mode = TIMER_CH_MODE_PWM;
                                         //Выбор режима ШИМ
sTIM_ChnInit.TIMR_CH_REF_Format = TIM-
ER_CH_REF_Format6;                  //Выбор режима работы
                                         //ШИМ - если счетный
                                         //регистр < регистра
                                         //сравнения, то на PA1
                                         //висит логическая "1",
                                         //иначе - "0"
TIMR_ChnInit(MDR_TIMR1, &sTIM_ChnInit);
                                         //Заполнение структуры
                                         //настройки TIMER1
/****************************************/

TIMR_SetChnCompare(MDR_TIMR1, TIMER_CHANNEL1,
                    0x10); // Кладём в регистр
                                         //сравнения число, которое
                                         //будет отвечать за
                                         //длительность импульса

*****Настройка режима ШИМ*****

```

```

TIMER_ChnOutStructInit(&sTIM_ChnOutInit); //Объявление
                                         //структурой настройки
                                         //TIMER1
sTIM_ChnOutInit.TIMER_CH_Number = TIMER_CHANNEL1;
                                         //Выбор канала ШИМ
sTIM_ChnOutInit.TIMER_CH_DirOut_Polarity = TIM-
ER_CHOPolarity_NonInverted;      //Без инверсии
sTIM_ChnOutInit.TIMER_CH_DirOut_Source = TIM-
ER_CH_OutSrc_REF;                //На канале Ch1 будет
                                         //дублироваться значение
                                         //REF
sTIM_ChnOutInit.TIMER_CH_DirOut_Mode = TIM-
ER_CH_OutMode_Output;           //Настройка канала Ch1 на
                                         //выход
                                         TIM-
ER_ChnOutInit(MDR_TIMER1, &sTIM_ChnOutInit);
                                         //Заполнение структуры
                                         //настройки TIMER1
/*****************************************/

```

TIMER_Cmd(MDR_TIMER1,ENABLE); //Включение таймера

```

while(1)
{
}
}

```

Лабораторная работа №2. Таймеры

Цель работы

Целью работы является изучение основ работы таймеров микроконтроллера K1986BE92QI.

Оборудование

1. Компьютер IBM PC.
2. Отладочная плата микроконтроллера K1986BE92QI.
3. Лабораторный макет.
4. Источник питания 12В, 1А.

Программа работы

1. Написать и проверить работу программы, представленные в примерах.
2. Подключить лабораторный макет к отладочной плате.

3. Организовать на светодиодах лабораторного макета “бегущий 0”. Временная задержка должна быть организована по прерыванию таймера.
4. Организовать на светодиодах лабораторного макета “бегущую 1”. Временная задержка должна быть организована по прерыванию таймера.
5. Вывести на светодиоды числа в двоичном коде от $0 \div 255$ с задержкой в 1 с. Временная задержка должна быть организована по прерыванию таймера.
6. Организовать на светодиодах лабораторного макета “эквалайзер”. Временная задержка должна быть организована по прерыванию таймера.
7. Организовать на выводе порта импульсы с частотой 5 кГц и коэффициентом заполнения 0.25. Длительности должны определяться таймером.
8. Организовать на выводе порта импульсы с частотой 50 кГц и коэффициентом заполнения 0.15. Длительности должны определяться таймером.
9. Организовать на выводе порта импульсы с частотой 100 кГц и коэффициентом заполнения 0.75. Длительности должны определяться таймером.
10. Организовать на светодиодах лабораторного макета “бегущий 0” с изменением скорости по внешним прерываниям. Временная задержка должна быть организована по прерыванию таймера.
11. Организовать на светодиодах лабораторного макета “бегущую 1” с изменением скорости по внешним прерываниям. Временная задержка должна быть организована по прерыванию таймера.
12. Организовать на светодиодах лабораторного макета свечение светодиодов с изменяемой яркостью посредством кнопок.
13. Организовать на выводе порта импульсы с частотой от 10 до 100 кГц и коэффициентом заполнения 0.5. Изменение частоты осуществлять по внешним прерываниям.
14. Организовать на выводе порта импульсы с частотой от 1 до 500 кГц и коэффициентом заполнения 0.5. Изменение частоты осуществлять по внешним прерываниям.
15. Написать программы вывода на дисплей результата измерения частоты входного импульсного сигнала.

Глава 4. АНАЛОГОВО-ЦИФРОВЫЕ ПРЕОБРАЗОВАТЕЛИ МИКРОКОНТРОЛЛЕРА K1986BE92QI

Описание АЦП микроконтроллера

В микроконтроллере K1986BE92QI 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. 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]. Максимальная частота CLK не может превышать 14 МГц.

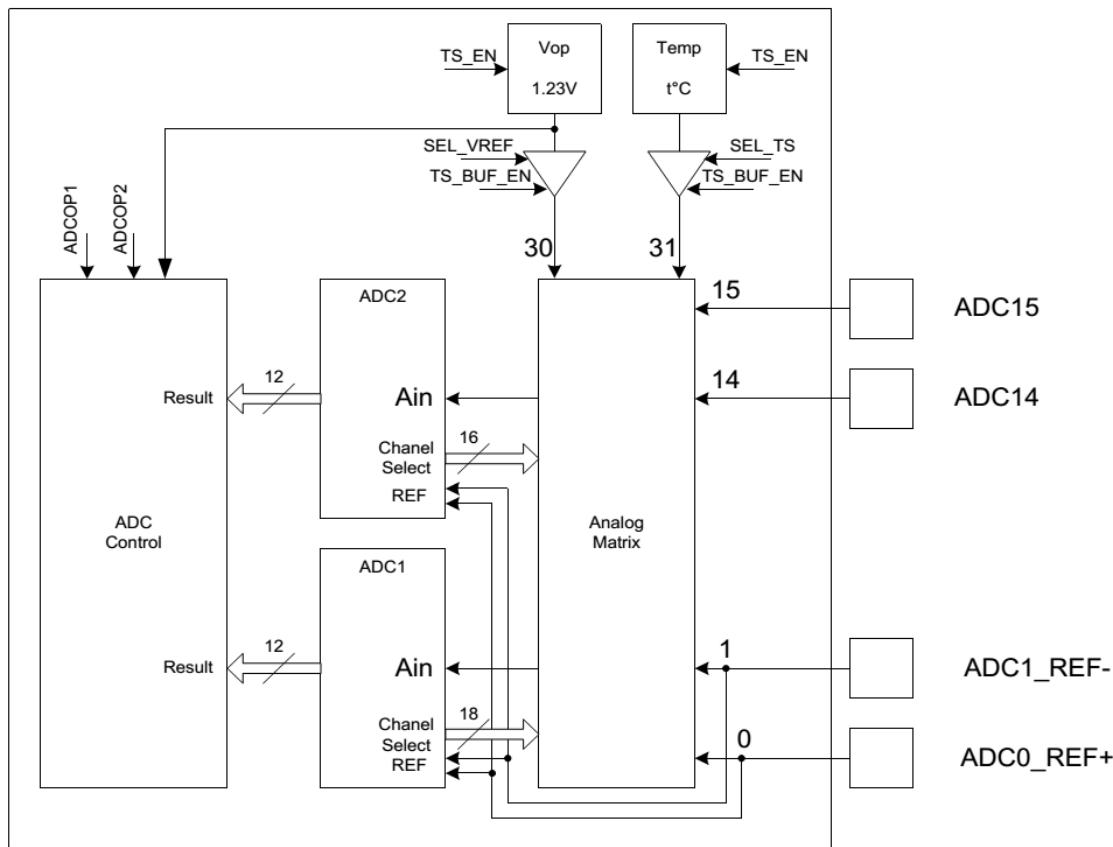


Рис. 4.1. Структурная схема аналогово-цифрового преобразователя

Для включения АЦП необходимо установить бит Cfg_REG_ADON. Для снижения тока потребления вместо собственного источника опорного напряжения в АЦП может использоваться источник датчика температуры. Для этого необходимо включить блок датчика температуры и источник опорного напряжения, установив бит TS_EN в 1. После включения можно использовать источник опорного напряжения для первого и второго АЦП вместо их собственных. Для этого необходимо установить биты ADCx_OP в единицу. Для преобразования необходимо, чтобы выводы, используемые АЦП у порта D, были сконфигурированы как аналоговые и были отключены какие-либо внутренние подтяжки.

Преобразование внешнего канала

В регистре ADCx_CFG в битах Cfg_REG_CHS[4:0] необходимо задать соответствующий выводу номер канала. Преобразование может осуществляться при внутренней опоре бит Cfg_M_REF = 0 и внешней Cfg_M_REF = 1, в этом случае опора берется с выводов

ADC0_REF+ и ADC1_REF-. Биты Cfg_REG_CHCH, Cfg_REG_RNGC, Cfg_REG_SAMPLE, TS_BUF_EN, SEL_VREF, SEL_TS и Cfg_Sync_Conver должны быть сброшены.

Для начала преобразования необходимо записать 1 в бит Cfg_REG_GO.

После завершения преобразования будет введен бит Flg_REG_EOCIF в регистре ADCx_STATUS, а в регистре ADCx_RESULT будет результат преобразования.

После считывания результата бит Flg_REG_EOCIF сбросится.

Если после первого преобразования результат не был считан, и было выполнено второе преобразование, то в регистре результата ADCx_RESULT будет значение от последнего преобразования, и помимо бита Flg_REG_EOCIF будет введен бит Flg_REG_OVERWRITE. Флаг Flg_REG_OVERWRITE может быть сброшен только записью в регистр ADCx_STATUS.

Последовательное преобразование нескольких каналов

Для автоматического последовательного преобразования нескольких каналов или одного канала в регистре ADCx_CHSEL необходимо установить единицы в битах, соответствующих выбранным для преобразования каналам. Преобразование может осуществляться при внутренней опоре бит Cfg_M_REF = 0 и внешней Cfg_M_REF = 1. В этом случае опора берется с выводов ADC0_REF+ и ADC1_REF-. Биты Cfg_REG_RNGC, TS_BUF_EN, SEL_VREF, SEL_TS и Cfg_Sync_Conver должны быть сброшены, а биты Cfg_REG_CHCH должны быть установлены. С помощью бит Delay_GO можно задать паузу между преобразованиями при переборе каналов. Эта определяется в тактах CPU_CLK, независимо от того на какой частоте ADC_CLK или CPU_CLK идет само преобразование. Для начала преобразования необходимо записать 1 в бит Cfg_REG_SAMPLE.

После завершения преобразования будет введен бит Flg_REG_EOCIF в регистре ADCx_STATUS, а в регистре ADCx_RESULT будет результат преобразования.

После считывания результата бит Flg_REG_EOCIF сбросится.

Если после первого преобразования результат не был считан, и было выполнено второе преобразование, то в регистре результата ADCx_RESULT будет значение от последнего преобразования, и помимо бита Flg_REG_EOCIF будет введен бит Flg_REG_OVERWRITE.

Флаг Flg_REG_OVERWRITE может быть сброшен только записью в регистр ADCx_STATUS.

Для последовательного преобразования одного и того же канала можно в регистре ADCx_CHSEL выбрать только один канал и установить бит Cfg_REG_CHCH в 1, либо установить номер канала в битах Cfg_REG_CHS[4:0] и сбросить бит Cfg_REG_CHCH в 0. В этом случае процесс последовательного преобразования будет выполняться только для данного канала. Последовательное преобразование значения датчика температуры и источника опорного напряжения могут выполняться только в режиме последовательного преобразования одного канала.

Преобразование с контролем границ

При необходимости отслеживать нахождение оцифрованных значений в допустимых пределах можно задать нижнюю и верхнюю допустимые границы в регистрах ADCx_L_LEVEL и ADCx_H_LEVEL. При этом если установлен бит Cfg_REG_RNGC, то в случае, если результат преобразования выходит за границы, выставляется флаг Flg_REG_AWOIFEN, а в регистре результата будет полученное значение.

Синхронный запуск двух АЦП

Для ускорения оцифровки одного канала можно использовать оба АЦП, запускаемые с задержкой одного относительно другого по времени. Время задержки запуска второго АЦП относительно первого задается битами Delay_ADC. При этом задержка Delay_ADC определяется в тактах CPU_CLK, независимо от того на какой частоте ADC_CLK или CPU_CLK идет само преобразование. Для одновременного запуска процесса преобразования необходимо установить бит Cfg_Sync_Conver и запустить процесс преобразования установкой бита Cfg_REG_GO. Синхронный запуск двух АЦП может работать также и в режиме последовательного преобразования нескольких каналов.

В таблице 4.1 представлены регистры управления и настройки АЦП.

Таблица 4.1.

Регистры АЦП

Название	Описание
MDR_ADC	Контроллер ADC
MDR_ADC->ADC1_CFG	Регистр управления ADC1
MDR_ADC->ADC2_CFG	Регистр управления ADC2
ADC1_H_LEVEL	Регистр MDR_ADC->ADCx_H_LEVEL верхней границы ADC1
ADC2_H_LEVEL	Регистр MDR_ADC->ADCx_H_LEVEL верхней границы ADC2

Продолжение таблицы 4.1

ADC1_L_LEVEL	Регистр MDR_ADC->ADCx_L_LEVEL нижней границы ADC1
ADC2_L_LEVEL	Регистр MDR_ADC->ADCx_L_LEVEL нижней границы ADC2
ADC1_RESULT	Регистр MDR_ADC->ADCx_RESULT результата ADC1
ADC2_RESULT	Регистр MDR_ADC->ADCx_RESULT результата ADC2
ADC1_STATUS	Регистр MDR_ADC->ADCx_STATUS статуса ADC1
ADC2_STATUS	Регистр MDR_ADC->ADCx_STATUS статуса ADC2
ADC1_CHSEL	Регистр MDR_ADC->ADCx_CHSEL выбора каналов перевода ADC1
ADC2_CHSEL	Регистр MDR_ADC->ADCx_CHSEL выбора каналов перевода ADC2

Пример программ настройки АЦП

Пример 5. Программа настройки АЦП1 микроконтроллера

Программа реализует мерцание светодиода с переменной частотой. Частота изменяется посредством переменного резистора, подключенного к входу АЦП микроконтроллера.

```
#include "MDR32Fx.h"
#include "MDR32F9Qx_port.c"
#include "MDR32F9Qx_rst_clk.c"
#include "MDR32F9Qx_it.c"
#include "MDR32F9Qx_adc.c"
#include "system_MDR32F9Qx.c"
PORT_InitTypeDef PORT_InitStructure; //Объявление структуры
                                     //инициализации портов
ADC_InitTypeDef ADC_InitStructure;   //Объявление структуры
                                     //инициализации АЦП
ADCx_InitTypeDef ADCx_InitStructure; //Объявление структуры
                                     //инициализации АЦП
int res=0,i;                      //Объявление переменных

int main(void)
{
    RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTA|
                      RST_CLK_PCLK_PORTB|RST_CLK_PCLK_PORTC|
                      RST_CLK_PCLK_PORTD|RST_CLK_PCLK_ADC, ENABLE);
                                     //Разрешение тактирования
                                     //периферии
    SCB->VTOR = 0x08000000;          // Включение таблицы векторов
```

```

MDR_RST_CLK->ADC_MCO_CLOCK=0x2000;//Включение
//тактирования АЦП

//Настройка выводов для светодиодов
PORT_InitStructure.PORT_Pin=(PORT_Pin_0 | PORT_Pin_1); //Выбор
//настраиваемого вывода
PORT_InitStructure.PORT_OE = PORT_OE_OUT; //Настройка вывода
//порта на режим выход (может
//быть вход-PORT_OE_IN)
PORT_InitStructure.PORT_FUNC = PORT_FUNC_PORT; //Функция
//порта - порт (так же может
//быть: основная функция-
//PORT_FUNC_MAIN,
//альтернативная функция-
//PORT_FUNC_ALTER,
//переопределенная функция-
//PORT_FUNC_OVERRIDE)
PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL;
//Настройка режима работы
//вывода цифровой (может быть
//аналоговый-
//PORT_MODE_ANALOG)
PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW; //Выбор
//скорости медленный
//фронт(100нс) (может так же
//быть отключен-
//PORT_OUTPUT_OFF,
//быстрый фронт(20нс)-
//PORT_SPEED_FAST,
//максимально быстрый
//фронт(10нс)-
//PORT_SPEED_MAXFAST)
PORT_Init(MDR_PORTC, &PORT_InitStructure); //Применение
//заполненной структуры к
//выбранному порту

//Настройка вывода для АЦП
PORT_InitStructure.PORT_Pin = PORT_Pin_7; //Выбор настраиваемого
//вывода
PORT_InitStructure.PORT_OE = PORT_OE_IN; //Настройка вывода
//порта на режим вход (может
//быть выход-PORT_OE_OUT)
PORT_InitStructure.PORT_MODE = PORT_MODE_ANALOG;
//Настройка режима работы
//вывода аналоговый (может

```

```

        //быть цифровой -
        //PORT_MODE_DIGITAL)
PORT_Init(MDR_PORTD, &PORT_InitStructure); //Применение
                                                //заполненной структуры к
                                                //выбранному порту

//Настройка АЦП
ADC_InitStructure.ADC_SynchronousMode =
ADC_SyncMode_Independent;           //Выбор независимого режима
работы
ADC_InitStructure.ADC_StartDelay = 0; //Нулевая задержка при старте
                                         //преобразования
ADC_InitStructure.ADC_TempSensor = C_TEMP_SENSOR_Disable;
                                         //Выключение датчика
                                         //температуры
ADC_InitStructure.ADC_TempSensorAmplifier =
                                         ADC_TEMP_SENSOR_AM
                                         PLIFIER_Disable;
                                         //Выключение усилителя
                                         //датчика температуры
ADC_InitStructure.ADC_TempSensorConversion =
ADC_TEMP_SENSOR_CONVERSION_Disable;
                                         //Выключение конвертации
                                         //датчика температуры
ADC_InitStructure.ADC_IntVRefConversion =
ADC_VREF_CONVERSION_Disable;
                                         //Выключение преобразования
                                         //опорного напряжения
ADC_InitStructure.ADC_IntVRefTrimming = 0; //Коэффициент для
                                         //опорного напряжения
ADC_Init(&ADC_InitStructure);

ADCx_InitStructure.ADC_ClockSource =
ADC_CLOCK_SOURCE_CPU;
                                         //Выбор источника тактирова-
                                         //ния
ADCx_InitStructure.ADC_SamplingMode=
ADC_SAMPLING_MODE_CICLIC_CONV;
                                         //Непрерывный режим
                                         //преобразований
ADCx_InitStructure.ADC_ChannelSwitching =
ADC_CH_SWITCHING_Disable;
                                         //Запрет переключения каналов
ADCx_InitStructure.ADC_ChannelNumber = ADC_CH_ADC7;
                                         //Номер канала

```

```

ADCx_InitStructure.ADC_Channels = 1; //Количество каналов
ADCx_InitStructure.ADC_LevelControl =
ADC_LEVEL_CONTROL_Disable;
                                //Отключение контроля уровня
                                //входного сигнала
ADCx_InitStructure.ADC_VRefSource =
ADC_VREF_SOURCE_INTERNAL;
                                //Внутренний источник
                                //опорного напряжения
ADCx_InitStructure.ADC_Prescaler = ADC_CLK_div_32768;
                                //Предварительный делитель
ADCx_InitStructure.ADC_DelayGo = 250; //Задержка между
                                //преобразованиями
ADC1_Init (&ADCx_InitStructure);

//Настройка прерываний
ADC1_ITConfig(ADCx_IT_END_OF_CONVERSION, ENABLE);
                                //Разрешение прерываний по
                                //окончанию преобразования
__enable_irq(); //Глобальное разрешение
                //прерываний
NVIC_EnableIRQ(ADC IRQn); //Разрешения внешнего
                //прерывания ADC
ADC1_Cmd (ENABLE); //Запуск АЦП

while(1)
{
for (i=0;i<463000-res;i++); //Задержка
PORT_ResetBits(MDR_PORTC, PORT_Pin_1); //Установить лог "0" на
                                //1 выводе порта С
for (i=0;i<463000-res;i++); //Задержка
PORT_SetBits(MDR_PORTC, PORT_Pin_1); //Установить лог "0" на 1
                                //выводе порта С
}
}

void ADC_IRQHandler(void)
{
ADC1_Cmd (DISABLE); //Остановка АЦП
res=ADC1_GetResult(); //Считывание результата
                    //преобразования
NVIC_ClearPendingIRQ(ADC IRQn); //Очистка бита ожидания
                                //внешнего прерывания EXTI1
}

```

```
    ADC1_Cmd (ENABLE); //Запуск АЦП
}
```

Пример 5. Программа настройки АЦП и ШИМ микроконтроллера

```
#include "MDR32F9Qx_config.h"
#include "MDR32Fx.h"
#include "MDR32F9Qx_timer.h"
#include "MDR32F9Qx_rst_clk.h"
#include "MDR32F9Qx_port.h"
#include "MDR32F9Qx_it.c"
#include "system_MDR32F9Qx.c"
#include "MDR32F9Qx_adc.c"

*****Объявление структур*****
TIMER_CntInitTypeDef sTIM_CntInit;
TIMER_ChnInitTypeDef sTIM_ChnInit;
TIMER_ChnOutInitTypeDef sTIM_ChnOutInit;
PORT_InitTypeDef PORT_InitStructure;
TIMER_CntInitTypeDef TIM_CntInitStructure;
ADC_InitTypeDef ADC_InitStructure;
ADCx_InitTypeDef ADCx_InitStructure;
*****



int res; //Объявление переменной, в которой будет содержаться значение
преобразования АЦП

*****Вектор прерывания АЦП*****
void ADC_IRQHandler(void)
{
    ADC1_Cmd (DISABLE); //Остановка АЦП
    res=ADC1_GetResult(); //Считывание результата преобразования
    NVIC_ClearPendingIRQ(ADC_IRQn); //Очистка бита ожидания внеш-
    него прерывания EXTI1
    ADC1_Cmd (ENABLE); //Запуск АЦП
}
*****



void main(void)
{
    *****Настройка тактирования*****
```

```

RST_CLK_DeInit(); //Сброс системы тактирования
RST_CLK_CPU_PLLconfig (RST_CLK_CPU_PLLsrcHSIdiv2,0);
//Инициализация системы тактирования
RST_CLK_PCLKcmd((RST_CLK_PCLK_RST_CLK |
RST_CLK_PCLK_TIMER1| RST_CLK_PCLK_ADC),ENABLE);
//Тактирование TIMER1 и ADC
RST_CLK_PCLKcmd((RST_CLK_PCLK_PORTA), ENABLE);
//Тактирование PORTA
SCB->AIRCR = 0x05FA0000 | ((uint32_t)0x500); //Настройка регистра
управления прерываниями и программного сброса
SCB->VTOR = 0x08000000; // Включение таблицы векторов
MDR_RST_CLK->TIM_CLOCK = 0x01000000; //Включение тактиро-
вания таймера
MDR_RST_CLK->ADC_MCO_CLOCK = 0x2000; //Включение такти-
рования АЦП
/***********************/

//Настройка вывода для АЦП
/***********************/
PORT_InitStructure.PORT_Pin = PORT_Pin_7; //Выбор настраиваемого
вывода
PORT_InitStructure.PORT_OE = PORT_OE_IN; //Настройка вывода
порта на режим вход (может быть выход-PORT_OE_OUT)
PORT_InitStructure.PORT_MODE = PORT_MODE_ANALOG;
//Настройка режима работы вывода аналоговый (может быть цифровой -
PORT_MODE_DIGITAL)
PORT_Init(MDR_PORTD, &PORT_InitStructure); //Применение запол-
ненной структуры к выбранному порту
/***********************/

*****Настройка вывода PA1*****
PORT_InitStructure.PORT_Pin = PORT_Pin_1; //Выбор настраиваемого
вывода
PORT_InitStructure.PORT_OE = PORT_OE_OUT; //Настройка вывода
на выход
PORT_InitStructure.PORT_FUNC = PORT_FUNC_ALTER; //Функция
вывода - альтернативная
PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL; //Режим
работы - цифровой
PORT_InitStructure.PORT_SPEED = PORT_SPEED_FAST; //Высокая
скорость работы короткие фронты)

```

```

PORT_Init(MDR_PORTA, &PORT_InitStructure); //Заполнение структу-
ры настройки PORTA
/***********************/

TIMER_DeInit(MDR_TIMER1); //Сброс всех настроек TIMER1
TIMER_BRGInit(MDR_TIMER1,TIMER_HCLKdiv1);

*****Настройка конфигураций TIMER1*****
sTIM_CntInit.TIMR_Prescaler      = 0x20; //Выбор предделителя
частоты
sTIM_CntInit.TIMR_Period         = 0xFFFF; //Период счета тайме-
ра
sTIM_CntInit.TIMR_CounterMode    =
TIMR_CntMode_ClkFixedDir; //Направление счета неизменно
sTIM_CntInit.TIMR_CounterDirection = TIMR_CntDir_Up;
//Режим счета вверх
sTIM_CntInit.TIMR_EventSource    = TIMR_EvSrc_None;
//Выбор источника прерываний - отсутствует
sTIM_CntInit.TIMR_FilterSampling = TIM-
ER_FDTS_TIMER_CLK_div_1;
sTIM_CntInit.TIMR_ETR_FilterConf = TIM-
ER_Filter_1FF_at_TIMER_CLK;
sTIM_CntInit.TIMR_ARR_UpdateMode = TIM-
ER_ARR_Update_On_CNT_Overflow; //Обнавление ARR при перепол-
нении CNT
sTIM_CntInit.TIMR_ETR_Prescaler   = TIM-
ER_ETR_Prescaler_None; //Без деления входной частоты
sTIM_CntInit.TIMR_ETR_Polarity    = TIM-
ER_ETRPolarity_NonInverted; //Нет инверсии входного ETR
sTIM_CntInit.TIMR_BRK_Polarity    = TIM-
ER_BRKPolarity_NonInverted; //Нет инверсии входного BRK
TIMR_CntInit(MDR_TIMER1,&sTIM_CntInit); //Заполнение структу-
ры настройки таймера
/***********************/

*****Настройка TIMER1 на режим ШИМ*****
TIMER_ChnStructInit(&sTIM_ChnInit); //Объявление структуры на-
стройки TIMER1
sTIM_ChnInit.TIMR_CH_Number       = TIMER_CHANNEL1;
//Выбор канала (PA1)

```

```

sTIM_ChnInit.TIMER_CH_Mode      = TIMER_CH_MODE_PWM;
//Выбор режима ШИМ
sTIM_ChnInit.TIMER_CH_REF_Format = TIM-
ER_CH_REF_Format6; //Выбор режима работы ШИМ - если счетный ре-
гистр < регистра сравнения, то на PA1 висит логическая "1", иначе - "0"
    TIMER_ChnInit(MDR_TIMER1, &sTIM_ChnInit); //Заполнение струк-
туры настройки TIMER1
/*****************/

```

```

    TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL1, 0x1FF);
// Ложим в регистр сравнения число, которое будет отвечать за ачаль-
ную длительность импульса

```

```

/****************Настройка режима ШИМ*******/
    TIMER_ChnOutStructInit(&sTIM_ChnOutInit); //Объявление структуры
настройки TIMER1
    sTIM_ChnOutInit.TIMER_CH_Number      = TIM-
ER_CHANNEL1; //Выбор канала ШИМ
    sTIM_ChnOutInit.TIMER_CH_DirOut_Polarity = TIM-
ER_CHOPolarity_NonInverted; //Без инверсии
    sTIM_ChnOutInit.TIMER_CH_DirOut_Source   = TIM-
ER_CH_OutSrc_REF; //На канале Ch1 будет дублироваться значение
REF
    sTIM_ChnOutInit.TIMER_CH_DirOut_Mode     = TIM-
ER_CH_OutMode_Output; //Настройка канала Ch1 на выход
    TIMER_ChnOutInit(MDR_TIMER1, &sTIM_ChnOutInit); //Заполнение
структурь настройки TIMER1
/*****************/

```

```

//Настройка АЦП
/******************
ADC_InitStructure.ADC_SynchronousMode =
ADC_SyncMode_Independent; //Выбор независимого режима работы
ADC_InitStructure.ADC_StartDelay = 0; //Нулевая задержка при
старте преобразования
ADC_InitStructure.ADC_TempSensor      =
ADC_TEMP_SENSOR_Disable; //Выключение датчика температуры
ADC_InitStructure.ADC_TempSensorAmplifier =
ADC_TEMP_SENSOR_AMPLIFIER_Disable; //Выключение усилителя
датчика температуры

```

```

ADC_InitStructure.ADC_TempSensorConversion =
ADC_TEMP_SENSOR_CONVERSION_Disable; //Выключение конвер-
тации датчика температуры
ADC_InitStructure.ADC_IntVRefConversion =
ADC_VREF_CONVERSION_Disable; //Выключение преобразования
опорного напряжения
ADC_InitStructure.ADC_IntVRefTrimming = 0; //Коэффициент для
опорного напряжения
ADC_Init (&ADC_InitStructure);

ADCx_InitStructure.ADC_ClockSource =
ADC_CLOCK_SOURCE_CPU; //Выбор источника тактирования
ADCx_InitStructure.ADC_SamplingMode =
ADC_SAMPLING_MODE_CICLIC_CONV; //Непрерывный режим пре-
образований
ADCx_InitStructure.ADC_ChannelSwitching =
ADC_CH_SWITCHING_Disable; //Запрет переключения каналов
ADCx_InitStructure.ADC_ChannelNumber = ADC_CH_ADC7; //Номер
канала
ADCx_InitStructure.ADC_Channels = 1; //Количество каналов
ADCx_InitStructure.ADC_LevelControl =
ADC_LEVEL_CONTROL_Disable; //Отключение контроля уровня вход-
ного сигнала
ADCx_InitStructure.ADC_VRefSource =
ADC_VREF_SOURCE_INTERNAL; //Внутренний источник опорного
напряжения
ADCx_InitStructure.ADC_Prescaler = ADC_CLK_div_32768;
//Предварительный делитель
ADCx_InitStructure.ADC_DelayGo = 250; //Задержка между пре-
образованиями
ADC1_Init (&ADCx_InitStructure);
//*****
//Настройка прерываний
//*****
ADC1_ITConfig(ADCx_IT_END_OF_CONVERSION, ENABLE);
//Разрешение прерываний по окончанию преобразования
__enable_irq(); //Глобальное разрешение прерываний
NVIC_EnableIRQ(ADC IRQn); //Разрешения внешнего прерывания
ADC
//*****

```

```

ADC1_Cmd (ENABLE); //Запуск АЦП
TIMER_Cmd(MDR_TIMER1,ENABLE);//Запуск таймера

while(1)
{
    TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL1, res); //
    Ложим в регистр сравнения значение АЦП, которое отвечает за длительность импульса (коэф. заполнения)
}
}

```

Лабораторная работа №3. Аналогово цифровой преобразователь

Цель работы

Целью работы является изучение основ настройки и работы аналогово-цирового преобразователя микроконтроллера K1986BE92QI.

Оборудование

1. Компьютер IBM PC.
2. Отладочная плата микроконтроллера K1986BE92QI.
3. Лабораторный макет.
4. Источник питания 12В, 1А.

Программа работы

1. Написать и проверить работу программы, представленные в примерах.
2. Подключить лабораторный макет к отладочной плате.
3. Организовать на светодиодах лабораторного макета “бегущий 0”. Временная задержка должна быть организована по прерыванию таймера, а скорость должна определяться АЦП.
4. Организовать на светодиодах лабораторного макета “бегущую 1”. Временная задержка должна быть организована по прерыванию таймера, а скорость должна определяться АЦП.
5. Вывести на светодиоды числа в двоичном коде от 0÷255 с задержкой в 1 с. Временная задержка должна быть организована по прерыванию таймера, а скорость должна определяться АЦП.
6. Организовать на светодиодах лабораторного макета “эквалайзер”. Временная задержка должна быть организована по прерыванию таймера, а скорость должна определяться АЦП.
7. Организовать на выводе порта импульсы с частотой 5 кГц и коэффициентом заполнения от 0.1 до 0.9. Длительности должны определяться таймером и изменяться посредством АЦП.

8. Организовать на выводе порта импульсы с частотой 50 кГц и коэффициентом заполнения от 0.2 до 0.8. Длительности должны определяться таймером и изменяться посредством АЦП.
9. Организовать на выводе порта импульсы с частотой 100 кГц и коэффициентом заполнения от 0.1 до 0.7. Длительности должны определяться таймером и изменяться посредством АЦП.
10. Написать программу вывода на дисплей результата преобразования АЦП и измеренного напряжения.
11. Написать программу реализующую изменение яркости двух светодиодов с помощью АЦП. Причем при увеличении напряжения на входе АЦП яркость одного светодиода должна увеличиваться, а яркость другого уменьшаться.
12. Написать программу вывода на дисплей величины напряжений, поданных на два АЦП.
13. Написать программу вывода на дисплей результата измерения амплитуды синусоидального сигнала.
14. Написать программу вывода на дисплей результата измерения частоты синусоидального сигнала. Измерение частоты осуществлять по преобразованию АЦП.

Глава 5. ЦИФРО-АНАЛОГОВЫЙ ПРЕОБРАЗОВАТЕЛЬ МИКРОКОНТРОЛЛЕРА K1986BE92QI

Описание ЦАП микроконтроллера

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 рис. 5.1 представлена структурная схема ЦАП микроконтроллера.

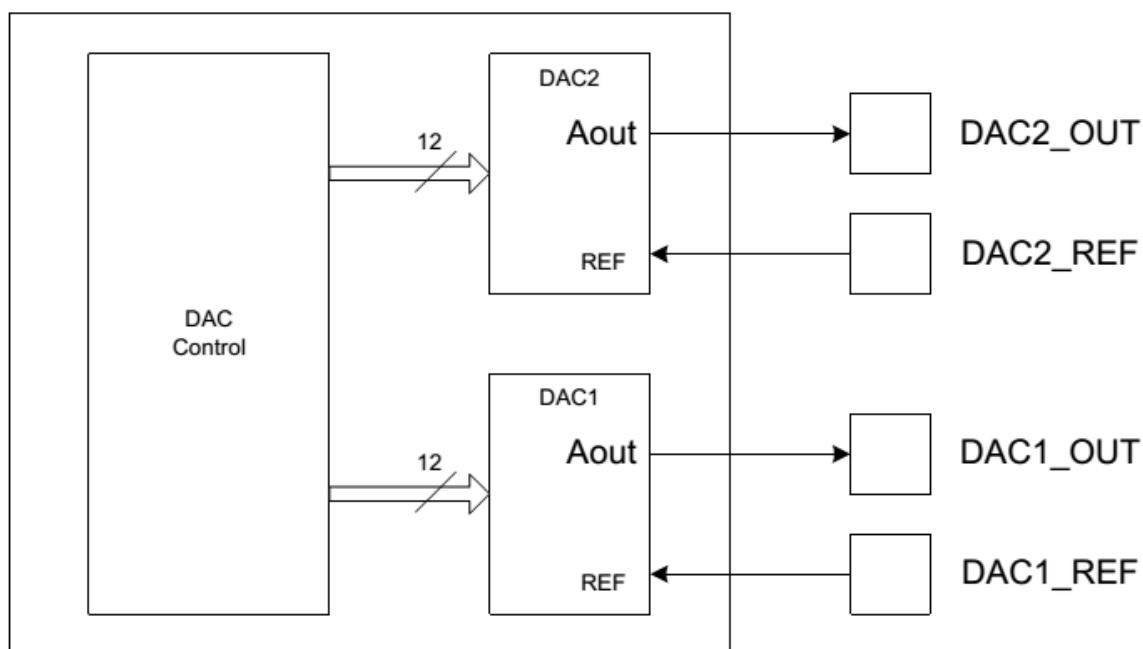


Рис. 5.1. Структурная схема ЦАП микроконтроллера

В таблице 5.1. представлено описание регистров ЦАП.

Таблица 5.1
Регистры ЦАП

Название	Описание
MDR_DAC	Контроллер DAC
MDR_DAC->CFG	Регистр управления DAC
MDR_DAC->DAC1_DATA	Регистр данных DAC1
MDR_DAC->DAC2_DATA	Регистр данных DAC2

Пример программ настройки ЦАП

Пример программы настройки ЦАП

Программа позволяет получать на выходе ЦАП сигнал треугольной формы.

```
#include "MDR32Fx.h"
#include "MDR32F9Qx_port.c"
#include "MDR32F9Qx_rst_clk.c"
#include "MDR32F9Qx_dac.c"
#include "system_MDR32F9Qx.c"

PORT_InitTypeDef PORT_InitStructure; //Обявление структуры
                                      //инициализации портов
int res=0,i;                         //Обявление переменных

int main(void)
{
    RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTA|
                      RST_CLK_PCLK_PORTB | RST_CLK_PCLK_PORTC |
                      RST_CLK_PCLK_PORTD | RST_CLK_PCLK PORTE |
                      RST_CLK_PCLK_DAC, ENABLE);           //Разрешение тактирования
                                              //периферии

//Настройка вывода для ЦАП
    PORT_InitStructure.PORT_Pin=PORT_Pin_0; //Выбор
                                          //настраиваемого вывода
    PORT_InitStructure.PORT_OE = PORT_OE_OUT; //Настройка
                                              //вывода порта на режим
                                              //выход (может быть вход-
                                              //PORT_OE_IN)
    PORT_InitStructure.PORT_MODE = PORT_MODE_ANALOG;
                                  //Настройка режима работы
```

```

        //вывода аналоговый
        //(может быть цифровой -
        //PORT_MODE_DIGITAL)
PORT_Init(MDR_PORTE, &PORT_InitStructure); //Применение
                                                //заполненной структуры к
                                                //выбранному порту

DAC2_Init(DAC2_AVCC);                      //Выбор опорного
                                                //напряжения
DAC2_Cmd(ENABLE);                          //Включение ЦАП

while(1)
{
for (i=0;i<4000;i++)
{
    DAC2_SetData(i);                      //Инкрементировать
                                            //переменную
}
for (i=4000;i>0;i--)
{
    DAC2_SetData(i);                      //Декрементировать
                                            //переменную
}
}
}
}

```

Лабораторная работа №3. Цифро-аналоговый преобразователь

Цель работы

Целью работы является изучение основ настройки и работы цифро-аналогово преобразователя микроконтроллера K1986BE92QI.

Оборудование

1. Компьютер IBM PC.
2. Отладочная плата микроконтроллера K1986BE92QI.
3. Лабораторный макет.
4. Источник питания 12В, 1А.

Программа работы

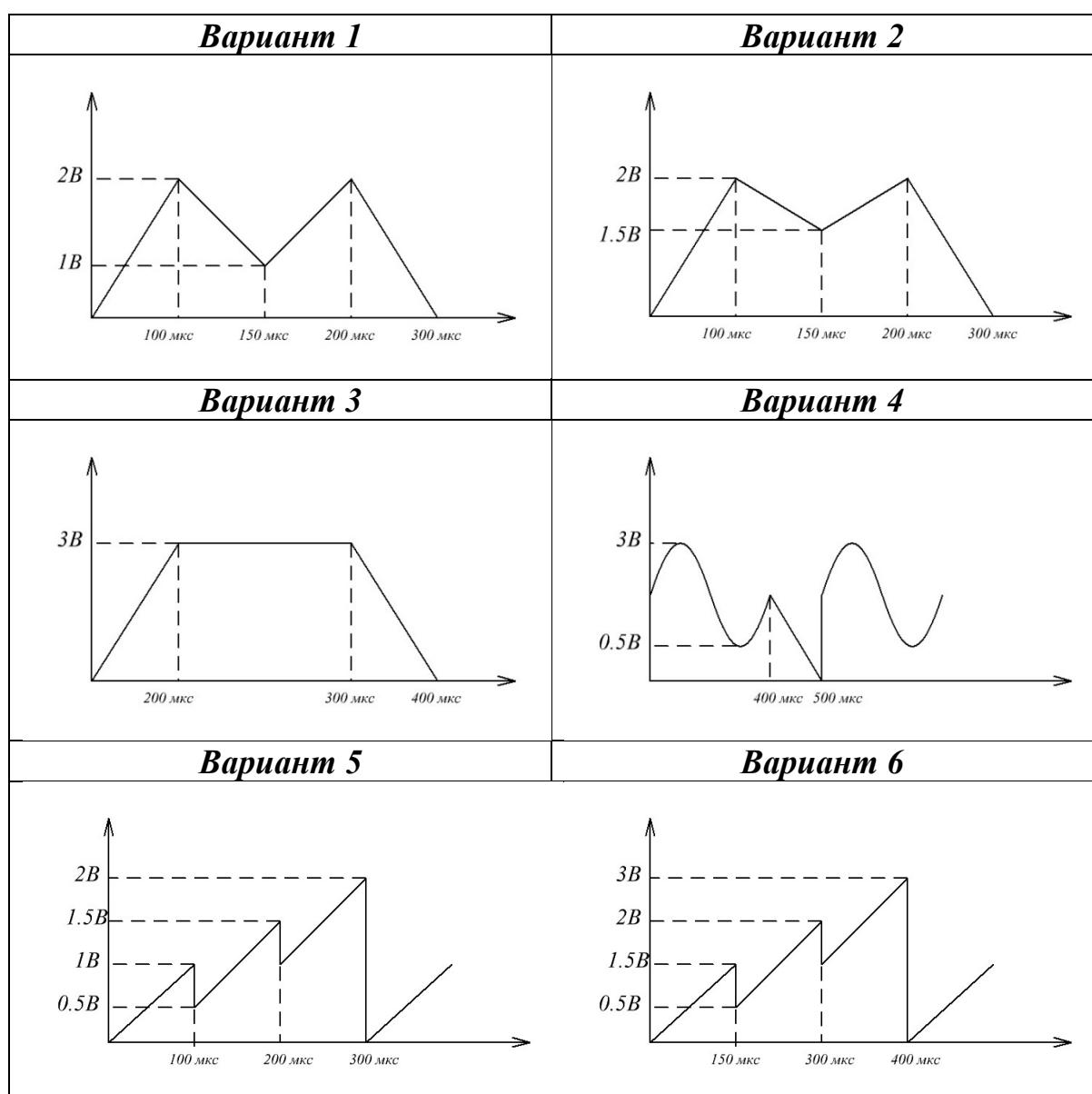
1. Написать и проверить работу программы, представленные в примерах.
2. Подключить лабораторный макет к отладочной плате.

3. Написать программы выхода в ЦАП сигнала синусоидальной формы с частотой 5 кГц и переменной амплитудой (от 0.5 до 3 В). Регулировку амплитуды осуществлять посредством АЦП.

4. Написать программы выхода в ЦАП сигнала синусоидальной формы с пересекающей частотой частотой от 1 до 5 кГц. Регулировку частоты осуществлять посредством АЦП.

5. Написать программы выхода в ЦАП пилообразного сигнала с пересекающей частотой частотой от 1 до 5 кГц и амплитудой от 0.2 до 2 В. Регулировку частоты осуществлять посредством АЦП, а регулировку амплитуды посредством внешних прерываний.

6. Согласно варианту, организовать на выходе ЦАП сигналы произвольной формы.



Глава 6. ИНТЕРФЕЙС UART МИКРОКОНТРОЛЛЕРА K1986BE92QI

Описание ЦАП микроконтроллера

Модуль UART микроконтроллера может быть запрограммирован для использования, как в качестве универсального асинхронного приемопередатчика, так и для инфракрасного обмена данными (SIR). Он содержит независимые буферы приема (16x12) и передачи (16x8) типа FIFO (First In First Out), что позволяет снизить интенсивность прерываний. Программное отключение FIFO позволяет ограничить размер буфера одним байтом.

Программное управление скоростью обмена. Обеспечивается возможность деления тактовой частоты опорного генератора в диапазоне (1x16 – 65535x16). Допускается использование нецелых коэффициентов деления частоты, что позволяет использовать любой опорный генератор с частотой более 3.6864 МГц.

Основные свойства модуля UART микроконтроллера:

- Поддержка стандартных элементов асинхронного протокола связи – стартового и стопового бит, а так же бита контроля четности.
- Независимое маскирование прерываний от буфера FIFO передатчика, буфера FIFO приемника, по таймауту приемника, по изменению линий состояния модема, а также в случае обнаружения ошибки.
- Поддержка прямого доступа к памяти.
- Обнаружение ложных стартовых бит.
- Формирование и обнаружения сигнала разрыва линии.
- Поддержка функция управления модемом (линии CTS, DCD, DSR, RTS, DTR и RI).
- Возможность организации аппаратного управления потоком данных.

Программируемые параметры модуля UART следующие:

- скорость передачи данных – целая и дробная часть числа;
- количество бит данных;
- количество стоповых бит;
- режим контроля четности;

- разрешение или запрет использования буферов FIFO (глубина очереди данных – 32 элемента или один элемент, соответственно);
- порог срабатывания прерывания по заполнению буферов FIFO (1/8, 1/4, 1/2, 3/4 и 7/8);
- частота внутреннего тактового генератора (номинальное значение - 1.8432 МГц) может быть задана в диапазоне 1.42 – 2.12 МГц для обеспечения возможности формирования бит данных с укороченной длительностью в режиме пониженного энергопотребления;
- режим аппаратного управления потоком данных.

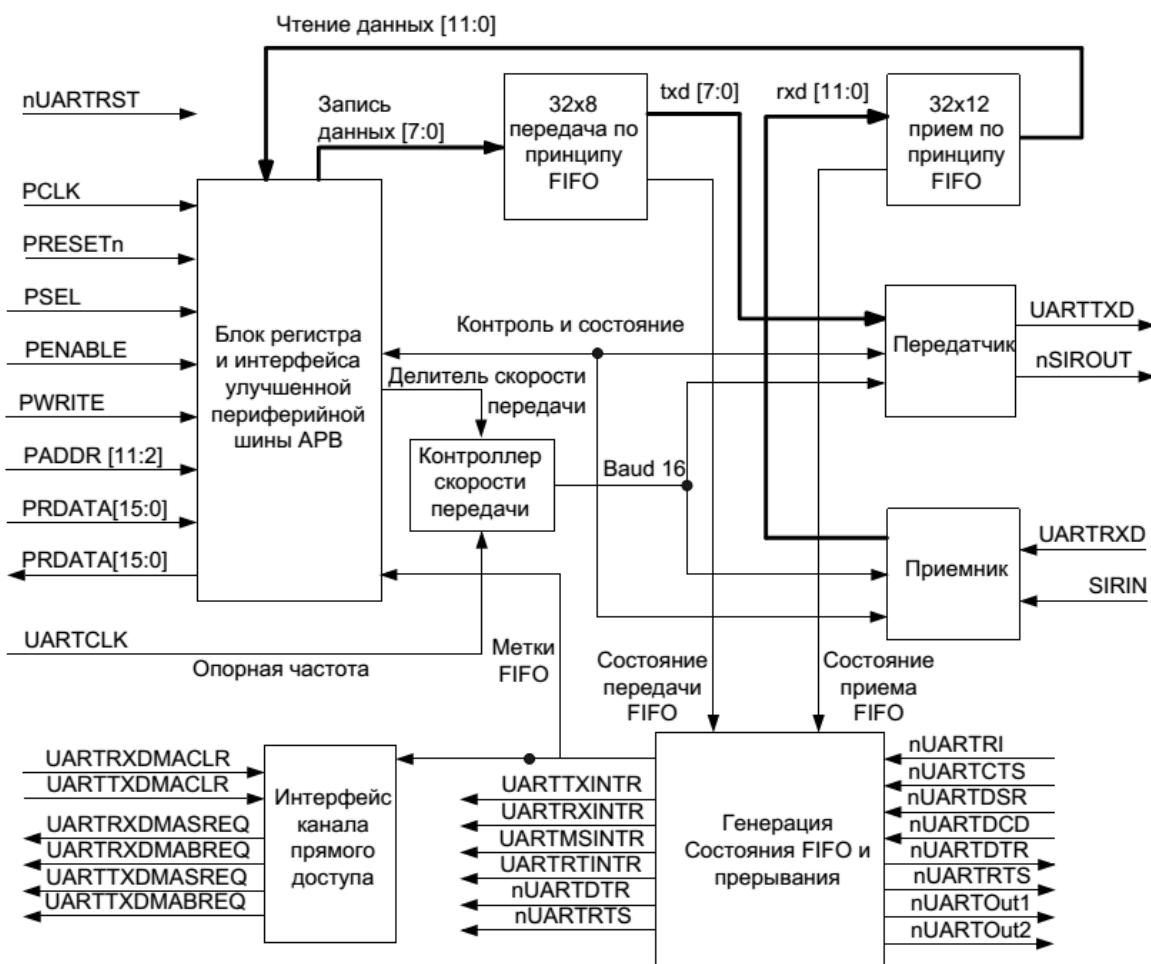


Рис. 6.1. Функциональная схема асинхронного приемо-передатчика

Пример программ настройки UART

Программа выводит на экран ПК выражение « $2+2=?$ », и принимает ответ. При неправильном ответе на экране появляется фраза «*Neverno!*», а при правильном «*Verno!*».

```

#include "MDR32Fx.h"
#include "MDR32F9Qx_port.c"
#include "MDR32F9Qx_rst_clk.c"
#include "MDR32F9Qx_it.c"
#include "MDR32F9Qx_uart.c"
#include "system_MDR32F9Qx.c"

PORT_InitTypeDef PORT_InitStructure; //Обявление структуры
                                    //инициализации портов
UART_InitTypeDef UART_InitStructure; //Обявление структуры
                                    //инициализации UART
int i;                                //Обявление переменных
void UART_SendBait (int data);
void vopros (void);

int main(void)
{
    RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTA |
    RST_CLK_PCLK_PORTB | RST_CLK_PCLK_PORTC |
    RST_CLK_PCLK_PORTD | RST_CLK_PCLK_UART1, ENABLE); //Разрешение тактирования
                                                       //периферии
    SCB->VTOR = 0x08000000;           // Включение таблицы векторов
    RST_CLK_PCLKcmd(RST_CLK_PCLK_UART1, ENABLE); //Включение тактирования
                                                   //UART

//Настройка выводов для UART PB5 - UART1_TX, PB6 -
//UART1_RX
    PORT_InitStructure.PORT_Pin = PORT_Pin_5; //Выбор
                                              //настраиваемого вывода
    PORT_InitStructure.PORT_OE = PORT_OE_OUT; //Настройка
                                              //вывода порта на режим
                                              //выход (может быть вход-
                                              ////PORT_OE_IN)
    PORT_InitStructure.PORT_FUNC = PORT_FUNC_ALTER; //Функция порта -
                                              //альтернативная функция
                                              //((так же может быть: порт
                                              //// PORT_FUNC_PORT,
                                              //основная функция-

```

```

        //PORT_FUNC_MAIN,
        //переопределенная
        //функция-
        //PORT_FUNC_OVERRIDE)
PORT_InitStructure.PORT_SPEED = PORT_SPEED_MAXFAST;
        //Выбор скорости
        //максимально быстрый
        //фронт(10нс) (может так
        //же быть отключен-
        //PORT_OUTPUT_OFF,
        //быстрый фронт(20нс)-
        //PORT_SPEED_FAST,
        //медленный фронт(100нс)-
        //PORT_SPEED_SLOW)
PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL;
        //Настройка режима работы
        //вывода цифровой (может
        //быть аналоговый-
        //PORT_MODE_ANALOG)
PORT_Init(MDR_PORTB, &PORT_InitStructure); //Применение
        //заполненной структуры к
        //выбранному порту

PORT_InitStructure.PORT_Pin = PORT_Pin_6; //Выбор
        //настраиваемого вывода
PORT_InitStructure.PORT_OE = PORT_OE_IN; //Настройка вывода
        //порта на режим вход
        //((может быть выход-
        //PORT_OE_OUT)
PORT_InitStructure.PORT_FUNC = PORT_FUNC_ALTER;
        //Функция порта -
        //альтернативная функция
        //((так же может быть: порт
        //PORT_FUNC_PORT,
        //основная функция-
        //PORT_FUNC_MAIN,
        //переопределенная
        //функция-
        //PORT_FUNC_OVERRIDE)
PORT_InitStructure.PORT_SPEED = PORT_SPEED_MAXFAST;

```

```

//Выбор скорости
//максимально быстрый
//фронт(10нс) (может так
//же быть отключен-
//PORT_OUTPUT_OFF,
//быстрый фронт(20нс)-
//PORT_SPEED_FAST,
//медленный фронт(100нс)-
//PORT_SPEED_SLOW)
PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL;
//Настройка режима работы
//вывода цифровой (может
//быть аналоговый-
//PORT_MODE_ANALOG)
PORT_Init(MDR_PORTB, &PORT_InitStructure); //Применение
//заполненной структуры к
//выбранному порту

//Настройка UART
UART_InitStructure.UART_BaudRate = 9600; //Скорость обмена
UART_InitStructure.UART_WordLength = UART_WordLength8b;
//Длина слова
UART_InitStructure.UART_StopBits = UART_StopBits1;
//Количество стоп бит
UART_InitStructure.UART_Parity = UART_Parity_No;
//проверка четности
UART_InitStructure.UART_FIFOMode = UART_FIFO_OFF;
//Отключение буфера
UART_InitStructure.UART_HardwareFlowControl =
UART_HardwareFlowControl_RXE |
UART_HardwareFlowControl_TXE; //Направление
UART_Init (MDR_UART1,&UART_InitStructure); //Применение
//заполненной структуры

//Настройка прерываний
__enable_irq(); //Глобальное разрешение
//прерываний
UART_ITConfig (MDR_UART1, UART_IT_RX, ENABLE);
//Разрешение прерываний
//по приему данных

```

```

NVIC_EnableIRQ(UART1_IRQn);      //Разрешения внешнего
                                  //прерывания ADC

UART_Cmd(MDR_UART1,ENABLE);
vopros();

while(1)
{
}

void UART1_IRQHandler(void)
{
    i=UART_ReceiveData(MDR_UART1);
    if (i==0x34)
    {
        UART_SendBait (' ');
        UART_SendBait ('V');
        UART_SendBait ('e');
        UART_SendBait ('r');
        UART_SendBait ('n');
        UART_SendBait ('o');
        UART_SendBait ('!');
        UART_SendBait ('\n');
    }
    else
    {
        UART_SendBait (' ');
        UART_SendBait ('N');
        UART_SendBait ('e');
        UART_SendBait (' ');
        UART_SendBait ('v');
        UART_SendBait ('e');
        UART_SendBait ('r');
        UART_SendBait ('n');
        UART_SendBait ('o');
        UART_SendBait ('!');
        UART_SendBait ('\n');
    }

    vopros();
}

```

```

void UART_SendBait (int data)
{
    UART_SendData (MDR_UART1,data); //Отправка байта
    while (!UART_GetFlagStatus(MDR_UART1, UART_FLAG_TXFE))
        //Ожидания завершения
    {}
}

void vopros (void)
{
    UART_SendBait ('2');
    UART_SendBait ('+');
    UART_SendBait ('2');
    UART_SendBait ('=');
    UART_SendBait ('?');
}

```

Лабораторная работа №3. Интерфейс UART

Цель работы

Целью работы является изучение основ настройки и работы цифро-аналогово преобразователя микроконтроллера K1986BE92QI.

Оборудование

1. Компьютер IBM PC.
2. Отладочная плата микроконтроллера K1986BE92QI.
3. Лабораторный макет.
4. Источник питания 12В, 1А.

Программа работы

1. Написать и проверить работу программы, представленные в примерах.
2. Подключить лабораторный макет к отладочной плате.
3. Подключить преобразователь RS232 к ПК.
4. Открыть программное обеспечение Terminal
5. Написать программу приема и отправки данных. Отправить в Terminal свою фамилию, а принять необходимо число соответствующее количеству раз мигания (инверсии) светодиода.

Литература

1. STlife.augmented [Электронный ресурс] / User manual – Режим доступа:http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00040810.pdf, свободный. – Загл. с экрана. – Яз. англ. (дата обращения 05/04/2015).
2. ST-LINK/V2 [Электронный ресурс] / ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32 – Режим доступа: <http://lib.chipdip.ru/163/DOC001163688.pdf>, свободный. – Загл. с экрана. – Яз. англ. (дата обращения 05/05/2015).
3. STlife.augmented [Электронный ресурс] / User manual – Режим доступа:http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/CD00267113.pdf, свободный. – Загл. с экрана. – Яз. англ. (дата обращения 06/05/2015).
4. Торгаев С.Н. Основы микропроцессорной техники: микроконтроллеры STM8S: учебное пособие / С.Н. Торгаев и др. – Томск : Изд-во ТПУ, 2014. – 130 с.
5. STlife.augmented [Электронный ресурс] / User manual – Режим доступа:http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/CD00283778.pdf, свободный. – Загл. с экрана. – Яз. англ. (дата обращения 09/05/2015).
6. STlife.augmented [Электронный ресурс] / User manual – Режим доступа:http://www.st.com/st-web-ui/static/active/en/resource/technical/document/technical_note/DM00039768.pdf, свободный. – Загл. с экрана. – Яз. англ. (дата обращения 15/05/2015).
7. STlife.augmented [Электронный ресурс] / User manual – Режим доступа:http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00039084.pdf, свободный. – Загл. с экрана. – Яз. англ. (дата обращения 15/05/2015).
8. Winstar Display Co., Ltd. [Электронный ресурс] / Specification – Режим доступа: <http://aquacontrol.narod.ru/spravka/WH1602A-YGH-CTK.pdf>, свободный. – Загл. с экрана. – Яз. англ. (дата обращения 20/05/2015).
9. Dallas Semiconductor. [Электронный ресурс] / DS18B20 – Режим доступа:
<http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Temp/DS18B>

20.pdf, свободный. – Загл. с экрана. – Яз. англ. (дата обращения 25/05/2015).

SUMMARY

This text-book is intended for senior students of universities, who is studying multi-bit microcontrollers and development of non-destructive testing systems based on them. The authors describe systematically the issues of programming the microcontrollers K1986BE92QI. A large number of examples of programs for configuring the main peripheral devices for these microcontrollers are presented.

For the students getting education on directions 11.04.04 "Electronics and nanoelectronics", and 12.04.04 "Biotechnical systems and technologies".

Издательство "STT" является лидером научного книгоиздания в Сибирском регионе, консультирует по вопросам защиты авторских прав, организации выпуска научной периодики и распространению научных книг и журналов в России и за рубежом. С 2014 года является официальным представителем британского издательства *Red Square Scientific*, специально ориентированного на российских авторов и российское научное содержание. Это облегчает российским ученым публикации за рубежом и делает их работы широко доступными для мирового научного сообщества.

Лучшие книги, выпущенные Издательством "STT", находятся в крупнейших библиотеках мира – National Library of Medicine (USA), The British Library (UK), Library of Congress (USA) и в The US Patent Bureau (USA), что обеспечивает их размещение в мировых базах данных.



Россия, 634028, г. Томск, проспект Ленина 15Б-1
Тел.: (3822) 421-455
E-mail: stt@sttonline.com

МИР ЖДЕТ ВАШИ КНИГИ!

Учебное издание

Торгаев Станислав Николаевич
Мусоров Илья Сергеевич
Солдатов Андрей Алексеевич
Сорокин Павел Владимирович

**ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ С ЯДРОМ
CORTEX-M3 В ЗАДАЧАХ ДИАГНОСТИКИ И КОНТРОЛЯ**

Учебное пособие
Опубликовано в авторской редакции

Издательство "STT"
Россия, 634028, г. Томск, проспект Ленина, 15^Б-1
Тел.: (3822) 421-455
E-mail: stt@sttonline.com



Усл. печ. л. 11,78. Уч.-изд. л. 2,51.
Бумага для офисной техники. Гарнитура Times.
Формат 60x84/16
Тираж 100 экз. Заказ № 598.