МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВПО «КОВРОВСКАЯ ГОСУДАРСТВЕННАЯ ТЕХНОЛОГИЧЕСКАЯ
АКАДЕМИЯ ИМЕНИ В.А.ДЕГТЯРЕВА»

Проектирование микропроцессорных систем

Учебно-методическое пособие к выполнению лабораторных работ

Составители:
А.С. Карпенков,

Ковров 2015

Введение

32-битная микропроцессорная архитектура с сокращенным набором команд (RISC) фирмы ARM Limited является одной из самых популярных архитектур микропроцессорных систем, поэтому все большее число компаний разрабатывают микроконтроллеры на этом ядре. Одна из последних версий ядра — Cortex-M3. В настоящее время на российском рынке в основном представлены микроконтроллеры с ядром ARM Cortex-M3 компаний STMicroelectronics (семейство STM32Fxxx), NXP (семейство LPC17xx) и отечественный дизайн-центр ЗАО «ПКК Миландр» [1, 2].

Микроконтроллеры ЗАО «ПКК Миландр» на базе архитектуры Cortex-M были разработаны для аппаратуры специального назначения, требующей большого запаса по стойкости (то есть способности микросхемы работать в агрессивной окружающей среде), и с широким рабочим температурным диапазоном.

Основные характеристики микроконтроллеров серии 1986BE9x

Ярким представителем отечественных микроконтроллеров, построенных на архитектуре ARM является микроконтроллер 1986BE91T — это микроконтроллер ЗАО «ПКК Миландр», основанный на ядре ARM Cortex-M3 (официально лицензированного). Внутри — 128 КБ флеш-памяти, 32 КБ статической памяти, аппаратный USB и 80Мгц ядро, изготовлено по технологии 180нм.

Устройства серии 1986BE9x являются микроконтроллерами со встроенной Flash-памятью программ и построены на базе высокопроизводительного процессорного RISC-ядра ARM Cortex-M3 (производительность — 1,25 DMIPS/МГц (Dhrystone 2.1) при нулевой задержке между обращениями к памяти). Максимальная тактовая частота работы микроконтроллера — 80 МГц, в то время как максимальная частота выборки данных (в данном случае команд) из Flash-памяти составляет 28,6 МГц (соответствует 35 нс). Для обеспечения максимального быстродействия при существующих технологических возможностях необходимо применять аппаратные решения для ускорения процесса обращения к Flash-памяти. В ряде микроконтроллеров, например, тех, что производят фирмы ЗАО «ПКК Миландр» и STMicroelectronics, для этого реализован специальный буфер «шириной» 64 бита. Но и в него выборка 32-разрядных команд осуществляется фактически с частотой 28,6×2 = 57,2 МГц. Таким образом, для работы при частоте свыше 57,2 МГц необходимо искусственно вводить задержку между обращениями к памяти. При частоте ниже 57,2 МГц задержка между считываниями из памяти является нулевой.

В ядре Cortex-M3 реализованы следующие функции:

- блок аппаратной защиты памяти от несанкционированного доступа;

- умножение за один цикл;
- аппаратная реализация деления (32 бита/32 бита).

Микроконтроллеры работают на тактовой частоте до 80 МГц и содержат 128 кбайт Flash-памяти программ и 32 Кбайта ОЗУ. Контроллер внешней системной шины позволяет работать с внешними микросхемами статического ОЗУ и ПЗУ, NAND Flash памятью и другими периферийными устройствами [3].

Встроенные RC-генераторы HSI (8 МГц) и LSI (40 кГц) и внешние генераторы HSE (2–16 МГц) и LSE (32 кГц), а также две схемы умножения тактовой частоты PLL для ядра и USB-интерфейса позволяют гибко настраивать скорость работы периферийных блоков микроконтроллеров.

Процессор Cortex-M3 выполнен по Гарвардской архитектуре, которая подразумевает использование раздельных шин данных и инструкций. Они называются шиной Dcode и Icode соответственно. Также имеется дополнительная системная шина, которая предоставляет доступ к области системного управления. У встроенной отладочной системы процессора Cortex имеется еще одна дополнительная шинная структура, которая называется локальной шиной устройств ввода/вывода. Системная шина и шина данных ядра подключаются к внешним (относительно ядра) блокам микроконтроллера через набор высокоскоростных шин, называемых матрицей шин. Для разрешения конфликтов при запросе на доступ к шине используется арбитраж. Блоки, подключенные к шине, могут быть активными задатчиками шины — «мастерами». Матрица шин образует несколько параллельных соединений между шинами ядра Cortex и другими внешними шинными «мастерами», такими как каналы DMA, статическое ОЗУ и устройства ввода/вывода. Если два шинных «мастера» (например, ядро Cortex и канал DMA) предпринимают попытку доступа к одному и тому же устройству ввода/вывода, то вступит в действие внутренний арбитр, который разрешит конфликт, предоставив доступ к шине тому, кто имеет наивысший приоритет.

Таким образом, архитектура системы памяти за счет матрицы системных шин позволяет минимизировать возможные конфликты при работе системы и повысить общую производительность.

Контроллер DMA дает возможность ускорить обмен информацией между ОЗУ и периферией без участия процессорного ядра.

Встроенный регулятор для формирования питания внутренней цифровой части формирует напряжение 1,8 В и не требует дополнительных внешних элементов. Таким образом, для работы микроконтроллеров достаточно одного внешнего напряжения питания в диапазоне от 2,2 до 3,6 В. Также в микроконтроллерах серии 1986ВЕ9х реализована возможность работы кристалла от внешней батареи, что позволяет иметь резервированное питание, на которое

микроконтроллер автоматически переключается при пропадании основного напряжения питания. При этом будут сохранены специальные флаги и не произойдет нарушения работы часов реального времени. Встроенные детекторы напряжения питания могут отслеживать уровень внешнего основного питания и уровень напряжения питания на батарее. Аппаратные схемы сброса по снижению уровня питания позволяют исключить некорректное выполнение кода программы и неправильное поведение микроконтроллера в целом при выходе уровня напряжения питания за допустимые пределы.

Для возможности реализации приложений, критичных к уровню энергопотребления, в микроконтроллерах существуют следующие режимы:

- Sleep;
- Deep sleep;
- Standby.

Микроконтроллеры обладают богатой периферией, набор которой зависит от модели МК (табл. 1).

Таблица 1. Сводная таблица микроконтроллеров серии 1986ВЕ9х

Периферия	**1986ВЕ91Т**	**1986ВЕ92У**	**1986ВЕ93У**
Корпус	4229.132-3 (132 вывода)	H18.64-1B (64 вывода)	H16.48-1B (48 выводов)
Ядро	ARM Cortex-M3		
ПЗУ	128 кбайт Flash		
ОЗУ	32 кбайт		
Питание	2,2–3,6 В		
Частота	80 МГц		
Температура	–60...+125 °С		
USER IO	96	45	31
USB	Device и Host FS (до 12 Мбит/с), встроенный PHY		
UART	2	2	2
CAN	2	2	1
SPI	2	2	1
I^2C	1	1	Нет
ADC 12 разрядов 1 Мвыб/с	16 каналов	8 каналов	5 каналов
DAC 12 разрядов	2	1	1
Компаратор	3 входа	2 входа	2 входа
Внешняя шина	32 разряда	8 разрядов	8 разрядов

Максимальным количеством периферийных устройств обладает МК 1986BE91T:

1. Цифровые модули:
    - контроллер прямого доступа в память с функциями передачи периферия – память, память – память;
    - два контроллера CAN-интерфейса;
    - контроллер USB-интерфейса с режимами работы Device и Host;
    - контроллеры интерфейсов USART, SPI, I^2C;
    - до 96 пользовательских линий ввода/вывода;
    - три 16-разрядных таймера с 4 каналами схем захвата и ШИМ с функциями формирования «мертвой зоны» и аппаратной блокировки;
    - системный 24-разрядный таймер;
    - два сторожевых таймера.

Реализация двух сторожевых таймеров позволяет расширить возможности по определению сбоя в выполнении кода программы МК. Один из них — оконный сторожевой таймер, который необходимо обновлять с определенной частотой. Другой — независимый сторожевой таймер, который синхронизируется отдельным генератором, не связанным с основной системной синхронизацией.

2. Аналоговые модули:

- два 12-разрядных АЦП (до 16 каналов), измеряемый диапазон напряжений от 0 до 3,6 В;
- температурный сенсор;
- двухканальный 12-разрядный ЦАП;
- встроенный компаратор.

Для отладки устройств на базе МК 1986BE91T в них реализованы 2 интерфейса:

- последовательный отладочный интерфейс SWD;
- последовательный отладочный интерфейс JTAG.

Средства разработки и отладки

Для разработки ПО для МК серии 1986BE9x можно использовать три разных пакета инструментальных средств:

- CodeMaster-ARM — интегрированная среда разработки российской компании «Фитон».
- Keil uVision — интегрированная среда разработки компании Keil.
- IAR Embedded Workbench — интегрированная среда разработки компании IAR Systems.

Программирование осуществляется либо с помощью стандартного программатора фирмы Keil (ULINK2), либо с помощью программатора, разработанного фирмой «Фитон».

Также для ознакомления с МК 1986BE91T разработана отладочная плата, позволяющая максимально использовать периферию и возможности МК [4]. Внешний вид отладочной платы и USB JTAG адаптера J-LINK (Segger), идущем в комплекте поставки, показан на рис. 1. Описание элементов платы представлено на рис. 2.

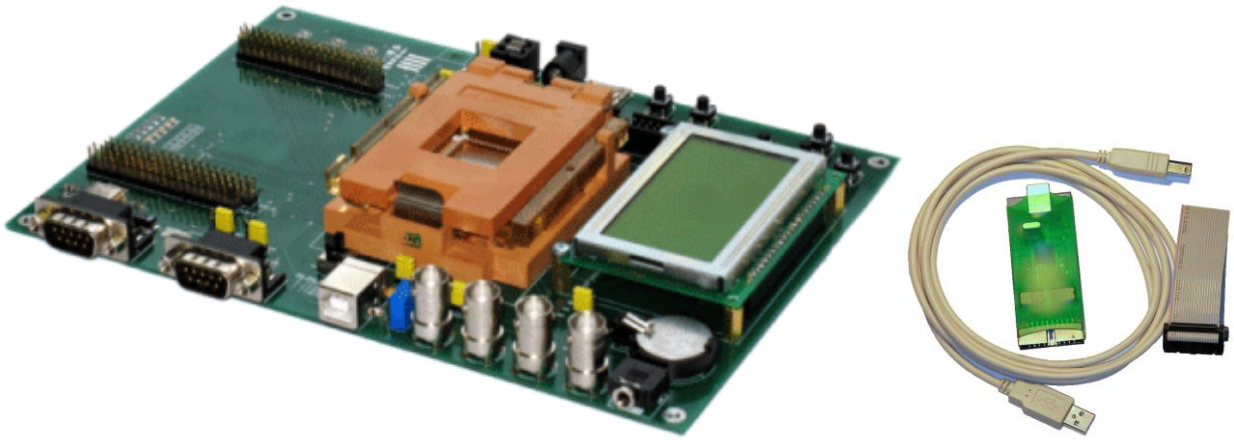


Рис. 1. Внешний вид отладочной платы для МК серии 1986BE9x

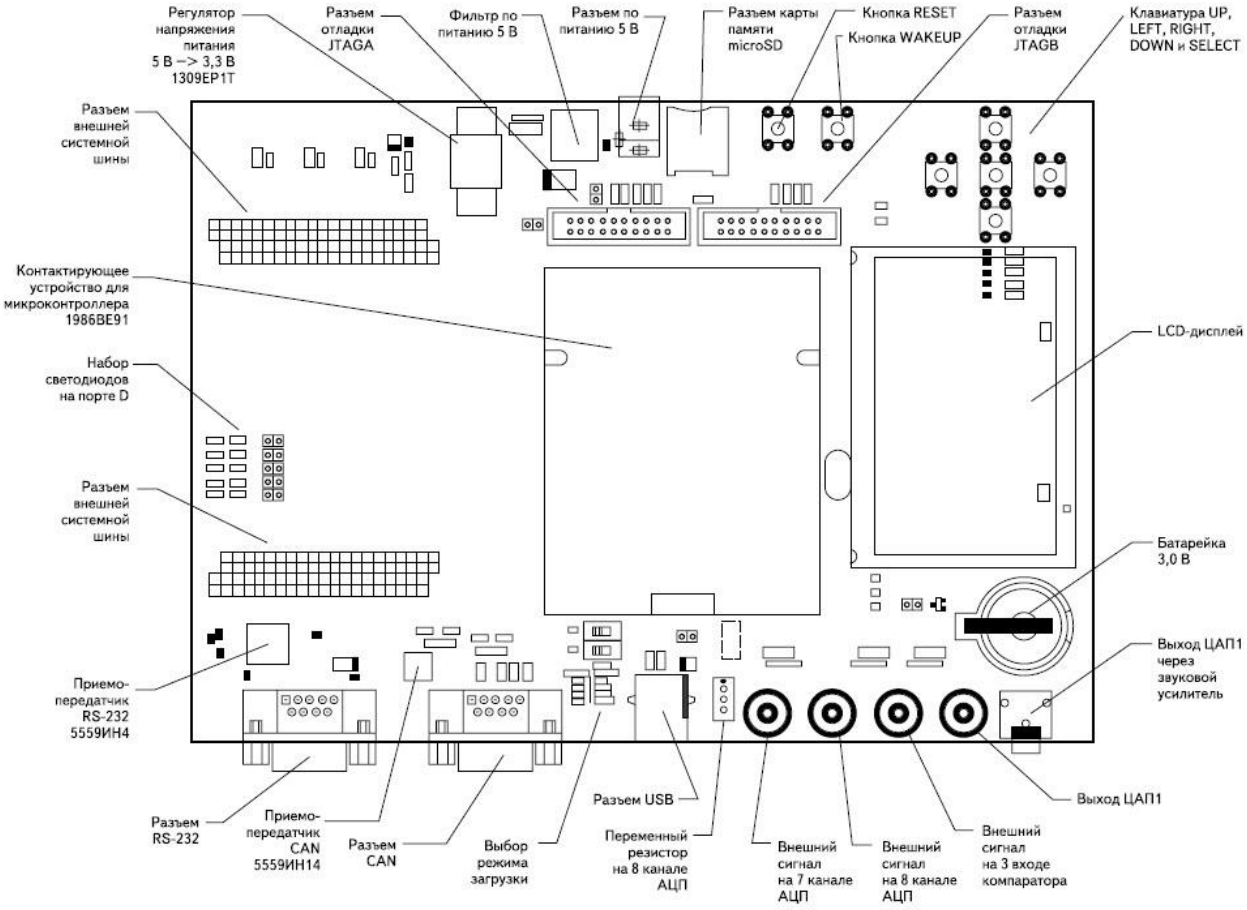


Рис. 2. Описание элементов отладочной платы

CodeMaster-ARM

CodeMaster-ARM — набор программно-аппаратных средств, предназначенный для разработки и отладки систем на базе микроконтроллеров ARM7/ARM9/Cortex-Mx, в том числе систем на базе первых отечественных микроконтроллеров с ядром Cortex-M3 серии 1986ВЕ9x, разработанных компанией ЗАО «ПКК Миландр» [5, 6, 7].

Среда разработки CodeMaster-ARM обладает всеми необходимыми средствами для реализации проектов на микроконтроллерах с ядром ARM. Особое внимание следует уделить тому факту, что среда разработана в России, поэтому пользователь в любой момент может получить качественную техническую поддержку и задать интересующие его вопросы. Неоспоримым достоинством также следует считать возможность выбора языка интерфейса между русским и английским.

Более подробная информация о среде CodeMaster-ARM, а также руководство пользователя могут быть получены на сайте фирмы «Фитон» [8].

CodeMaster-ARM - это интуитивная и лёгкая в использовании интегрированная среда разработки (IDE). CodeMaster-ARM содержит редактор, менеджер проектов, компилятор языка Си, макроассемблер, линкер, программный симулятор и драйвера аппаратных JTAG эмуляторов JEM-ARM, JEM-ARM-V2.

Встроенный язык скриптов позволяет автоматизировать рутинные задачи при отладке программ или тестировании оборудования.

Интерфейс JTAG или **JTAG** - специальный интерфейс к целевому микроконтроллеру, обеспечивающий связь внешнего аппаратного отладчика (эмулятора) с встроенными в микроконтроллер средствами отладки. Каждый микроконтроллер, имеющий интерфейс JTAG, обязательно содержит в своем составе специальные аппаратные средства, обеспечивающие реализацию отладочных функций.

Target JTAG connector – разъём установленный на целевую плату для отладки и/или программирования целевого микроконтроллера через интерфейс JTAG.

FLASH память - FLASH память целевого микроконтроллера, которая может быть запрограммирована/перепрограммирована средствами Phyton JEM-ARM через интерфейс JTAG.

Интегрированная среда разработки или **IDE** – набор программных средств разработки и отладки микропроцессорных систем, управляемых единой программной оболочкой.

CodeMaster или **CodeMaster-ARM** – торговая марка ООО "Фирма Фитон" для интегрированных сред микроконтроллеров ARM.

CodeMaster Компилятор или **CMC** – Си компилятор для ARM микроконтроллеров.

CodeMaster Simulator или **Instruction-Set Simulator or Command-Set Simulator** – логическая модель микроконтроллера ARM и модель внутренней памяти, реализованная как компьютерная программа для имитации команд микроконтроллера. Не поддерживает симуляцию периферийных устройств микроконтроллера.

JTAG emulator или **JTAG debugger** – аппаратное устройство, работающее под управлением CodeMaster IDE и предназначенное для отладки целевой микроконтроллерной системы через JTAG интерфейс.

JEM-ARM – торговая марка ООО "Фирма Фитон" для JTAG-эмулятора.

JEM JTAG разъем – разъём, установленный на плате JEM для подключения к JTAG интерфейсу целевой системы.

JTAG кабель - плоский кабель для соединения разъёма JTAG на целевой плате и JEM-ARM.

Тестовые целевые платы – небольшие платы, поставляемые с JEM-ARM и обычно используемые как базовые прототипы микроконтроллерных систем.

Файлы исходного кода - текстовые файлы(ASCII) или программы на Си или ассемблере с комментариями. Программы в виде файлов исходного кода могут быть загружены в среду CodeMaster для компиляции.

Абсолютный объектный файл - выход компоновщика. Компоновщик обрабатывает выход компилятора и сохраняет результат в двоичных кодах команд микроконтроллера ARM в абсолютной объектной форме. JEM-ARM загружает эти машинные коды в память целевого микроконтроллера для отладки или запуска на оборудовании пользователя после завершения отладки. Различный компиляторы и компоновщики могут иметь собственные соглашения по именованию и расширениям файлов.

Лабораторная работа №1. Знакомство со средой проектирования CodeMaster-ARM.

Цель: Изучение среды проектирования CodeMaster-ARM на примере запуска тестового проекта.

В комплекте со средой CodeMaster-ARM поставляются примеры проектов для МК 1986BE9x, что, несомненно, облегчает разработку собственных проектов.

При первом запуске САПР CodeMaster-ARM появляется окно с предложением запустить один из примеров проектов. Примеры для микроконтроллера 1986BE91T находятся в разделе “Cortex-M3/Milandr 1986” (см. рисунок 3).

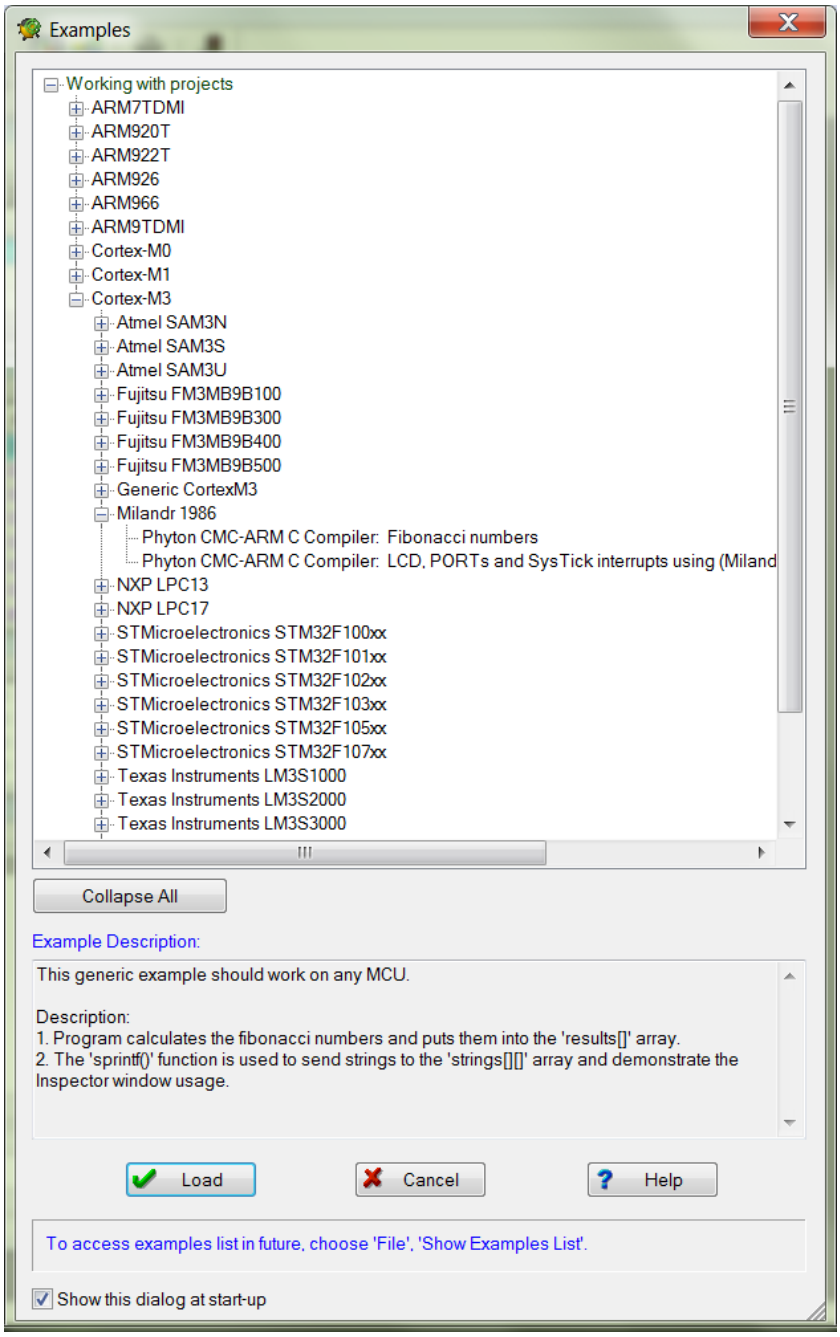
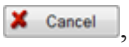


Рисунок 3 – Окно мастера запуска предустановленных примеров

Для загрузки примера проекта необходимо выбрать его и нажать кнопку 

. Если нажать кнопку 

, то появиться типичное окно среды программирования CodeMaster-ARM (рисунок 4).

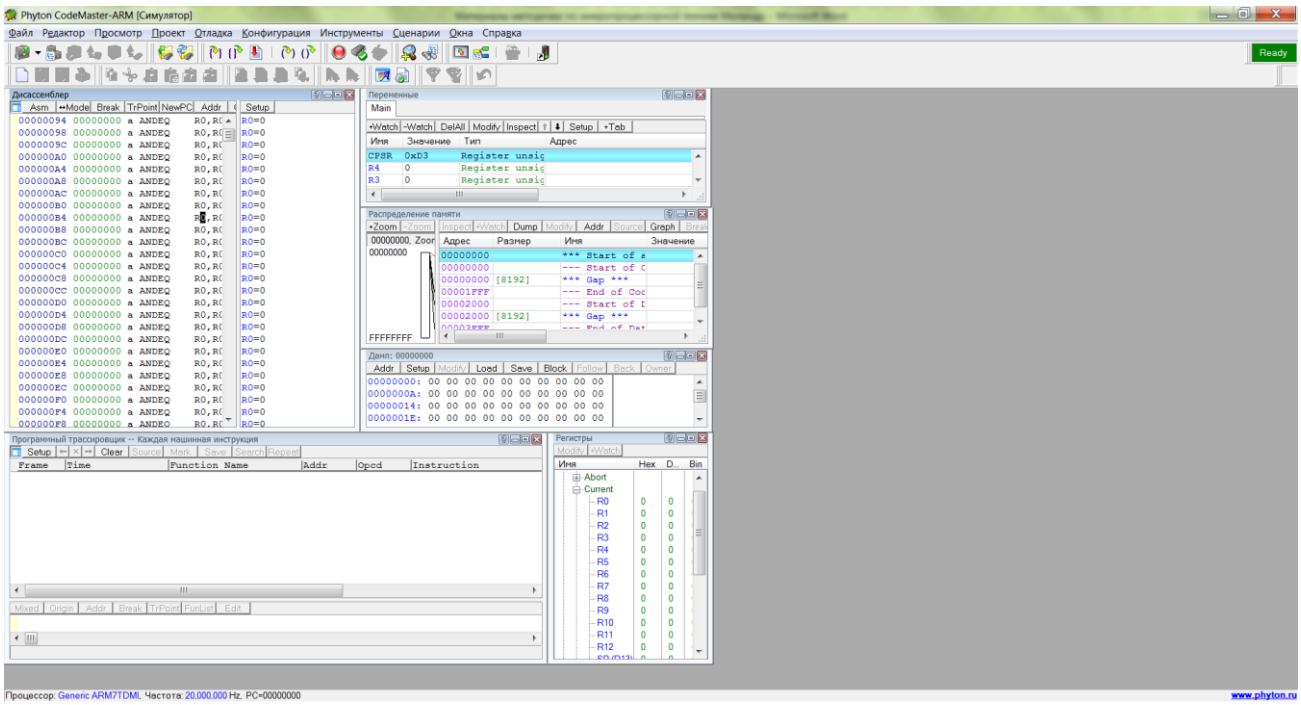


Рисунок 4 – Графическая среда САПР CodeMaster-ARM

Для выполнения данной лабораторной работы необходимо загрузить проект «Phyton CMC-ARM C Compiler: LCD, PORTs and SysTick interrupts using (Milandr 1986BE91 evaluation board needed)». Пример использует следующие периферийные узлы микроконтроллера:

- Порт ввода/вывода PORTA, PORTC, PORTD, PORTE,
- Контроллер внешней шины EXT_BUS_CNTRL,
- Системный таймер SysTick, формирующий прерывания.

Пример использует установленные по умолчанию (после Сброса) параметры, определяющие источник и значение тактовой частоты.

Пример использует следующие устройства отладочной платы:

- LCD дисплей,
- КнопкиUP, DOWN, LEFT, RIGHT, SEL (джойстик),
- 5 светодиодовVD7..VD11.

После загрузки проекта в графической среде слева можно увидеть окно «Проект» (рисунок 5). Окно «Проект» отображает содержимое проекта (файлы исходного текста, библиотеки, листинги, файлы препроцессора, и т.д.) в форме дерева. Файлы сгруппированы в

пределах их подкаталога и изображены в виде веток дерева (файловых групп). Древообразная структура содержит следующие элементы для каждого исходного файла в проекте:

- Иконки и имена файлов.
- Описание, если есть, в квадратных скобках (смотрите пример).
- Состояние при последней компиляции (общее количество ошибок, если они случились, предупреждений и замечаний).
- Файлы зависимостей, если есть.

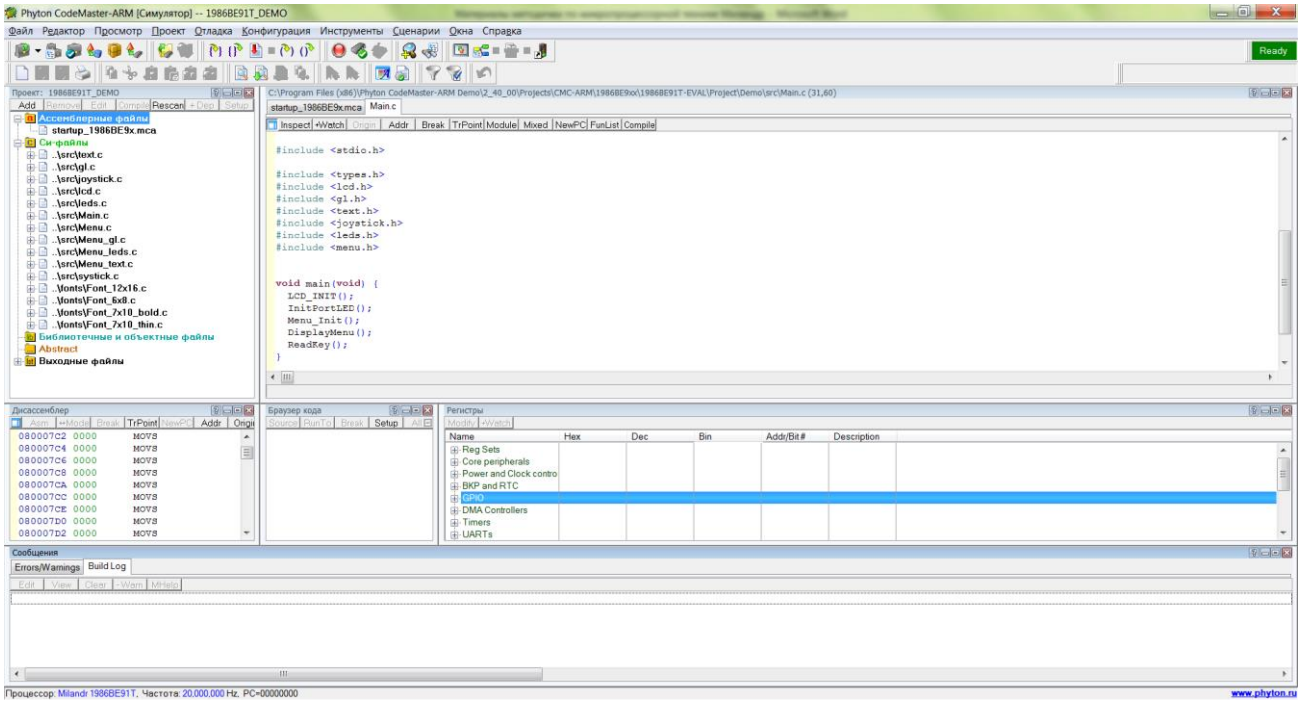


Рисунок 5 – Графическая среда САПР CodeMaster-ARM с загруженным примером проекта

Справа от окна «Проект» находится окно «Редактор». Окно Редактора является основным окном в CodeMaster IDE. Его основное назначение - отображать и редактировать исходный текст программы. В действительности, в этом окне можно полноценно редактировать любой ASCII–файл. С этой точки зрения, это - окно редактора. Можно открыть столько окон Редактора, сколько нужно.

Окно Редактора предоставляет большое количество функций редактирования, доступных через меню Редактор:

- Базовые (стандартные) функции редактирования, включающие: отмену сделанных действий, поиск фразы, поиск с заменой, повтор последнего поиска (во всех этих функциях можно использовать обычные текстовые строки и выражения поиска).

Окно поддерживает постоянные (persistent) блоки и выполняет полный набор действий с блоками со стандартными потоковыми (stream), вертикальными (column) и строчными блоками текста.

- Чтобы сделать процесс разработки проще и удобнее.
- Клавиши редактирования («горячие клавиши») для обращения ко всем обычным функциям редактирования и некоторым дополнительным функциям.
- Расширенные возможности в режиме редактора в виде назначения «горячих клавиш» редактора и создания новых команд. Можно по своему усмотрению переназначить комбинации клавиш некоторых собственных (встроенных) команд редактора. Можно расширить возможности самого редактора: создать новые, собственные (пользовательские) команды редактора на основе встроенного языка сценариев, которые повысят Вашу производительность в процессе разработки. Подробнее об этом—в главе Закладка Назначение клавиш (в диалоге Опции редактора).
- Панель инструментов редактора, общую для всех окон Редактора, и индивидуальную панель инструментов в каждом окне Текст.
- Правую панель для отображения дополнительных данных для левой панели.

При работе с исходным текстом, правая панель отображает список автоматического дописывания слов. В режиме отладки, правая панель отображает значения переменных, находящихся в соответствующих строках левой (основной) панели. Чтобы включить/выключить правую панель, нажмите кнопку Вкл/выкл правую панель на панели инструментов этого окна.

Окно Редактора является частью интегрированной среды разработки. После завершения редактирования исходного текстового файла, Вы можете и дальше работать с этим окном; от компиляции открытого файла до отладки программ.

В самом низу графической среды CodeMaster располагается окно «Сообщения». После каждой компиляции окно «Сообщения» отображает сообщения об ошибках, сгенерированные компилятором и/или компоновщиком. По умолчанию CodeMaster открывает и закрывает это окно в зависимости от результатов компиляции. Если компиляция прошла без ошибок, то CodeMaster закроет это окно, даже если оно было оставлено открытым. Если во время компиляции произошла ошибка, CodeMaster оставит это окно открытым или откроет его если оно было закрыто.

Закладка Ошибки/Предупреждения открывает окно вывода сообщений, что поможет вам скорректировать ваш исходный файл. Каждое сообщение отображается в окне одной строкой.

Чтобы выбрать сообщение используйте мышь или клавиши со стрелками. Для горизонтальной прокрутки используйте клавиши с левой и правой стрелками. Нажатие клавиш Home (End) переместит курсор в начало (конец) выбранного сообщения.

Если вы выбрали сообщение об ошибке компилятора, то строка исходного кода, вызвавшая это сообщение, будет подсвечена в окне Редактора. Это срабатывает только для открытых файлов, если файл с ошибкой не был открыт в окне Редактора, то он не откроется автоматически. Если открытое окно Редактора закрыто другими окнами, CodeMaster автоматически переместит его вверх, а окно Сообщения останется активным. Если сразу несколько окон Редактора с текстом модулей Вашей программы, то окно с модулем, породившим это сообщение, будет помещено на передний план.

Чтобы определить строку исходного кода, в которой произошла ошибка, выберите сообщение в соответствующем окне Сообщения нажмите кнопку Edit (на панели инструментов окна Сообщения) или используйте команду «Редактировать исходный текст» его локального меню или просто дважды щёлкните кнопкой мыши на строке сообщения. Окно Редактора отобразит соответствующую строку текста и станет активным. Если ваш компилятор выдаёт информацию о номере строки и номере символа в строке, то курсор установится точно на ошибке. Если соответствующий исходный файл ещё не открыт в окне Редактора, то CodeMaster автоматически откроет его.

Закладка Журнал компиляции отображает сеанс компиляции: командные строки, запустившие компилятор, и выходные сообщения компилятора, ассемблера и линкера на консоль. Окно стирает текст предыдущего сеанса.

Примечание. Когда вы добавляете или удаляете строки в окне Редактора, чтобы исправить ошибку, окно Сообщения не обновляет номер ошибочной строки. Например, окно Сообщения сообщает об ошибке в строке 20. если вы отредактировали файл и добавили 3 строки после 20-й, то когда вы выберете сообщение, связанное с 20-й строкой, окно Редактора подсветит строку 20, хотя ошибка сдвинулась на строку 23.

После загрузки проекта, необходимо выбрать отладчик. Для этого в среде CodeMaster-ARM необходимо открыть диалог Отладка > Выбор отладчика > Выберите отладчик. В

появившемся окне необходимо выбрать JTAG-отладчик J-Link и нажать кнопку OK. (рисунок 6).

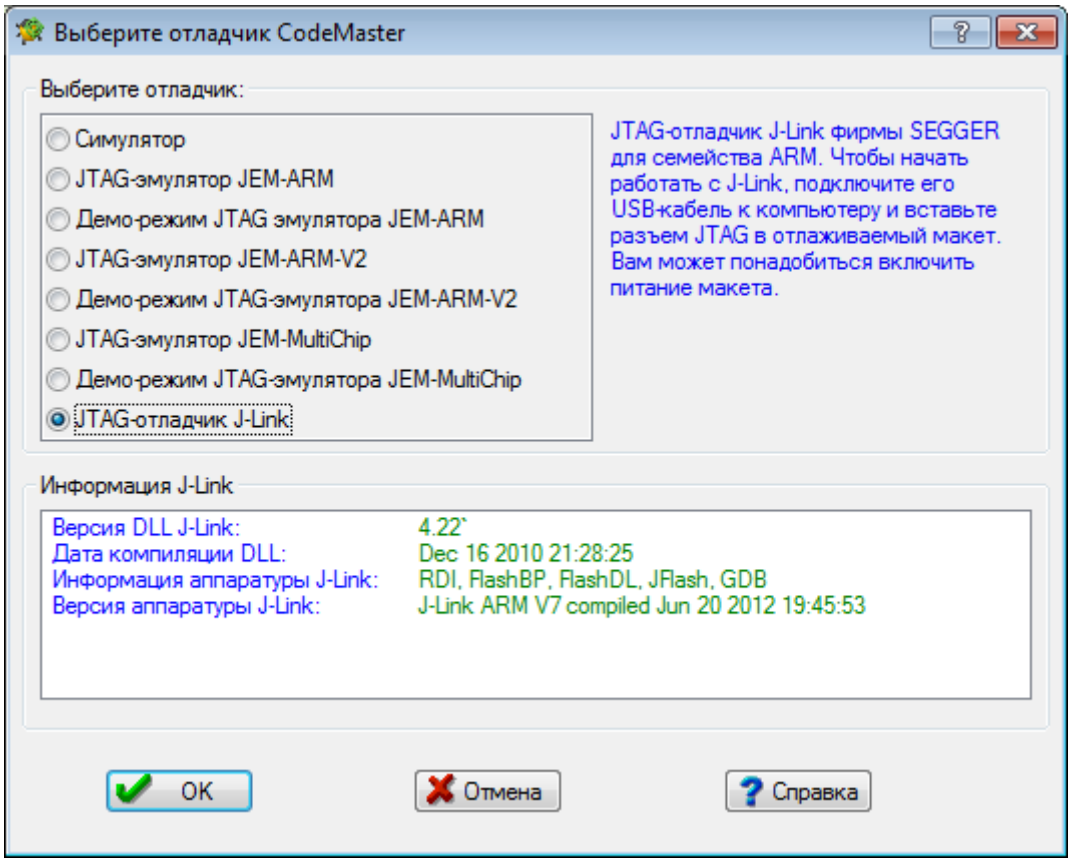


Рисунок 6 – Графическое окно выбора отладчика

При первом запуске в настройках Опции Отладки > Настройки J-Link необходимо сконфигурировать J-Link адаптер как показано на рисунке 7.

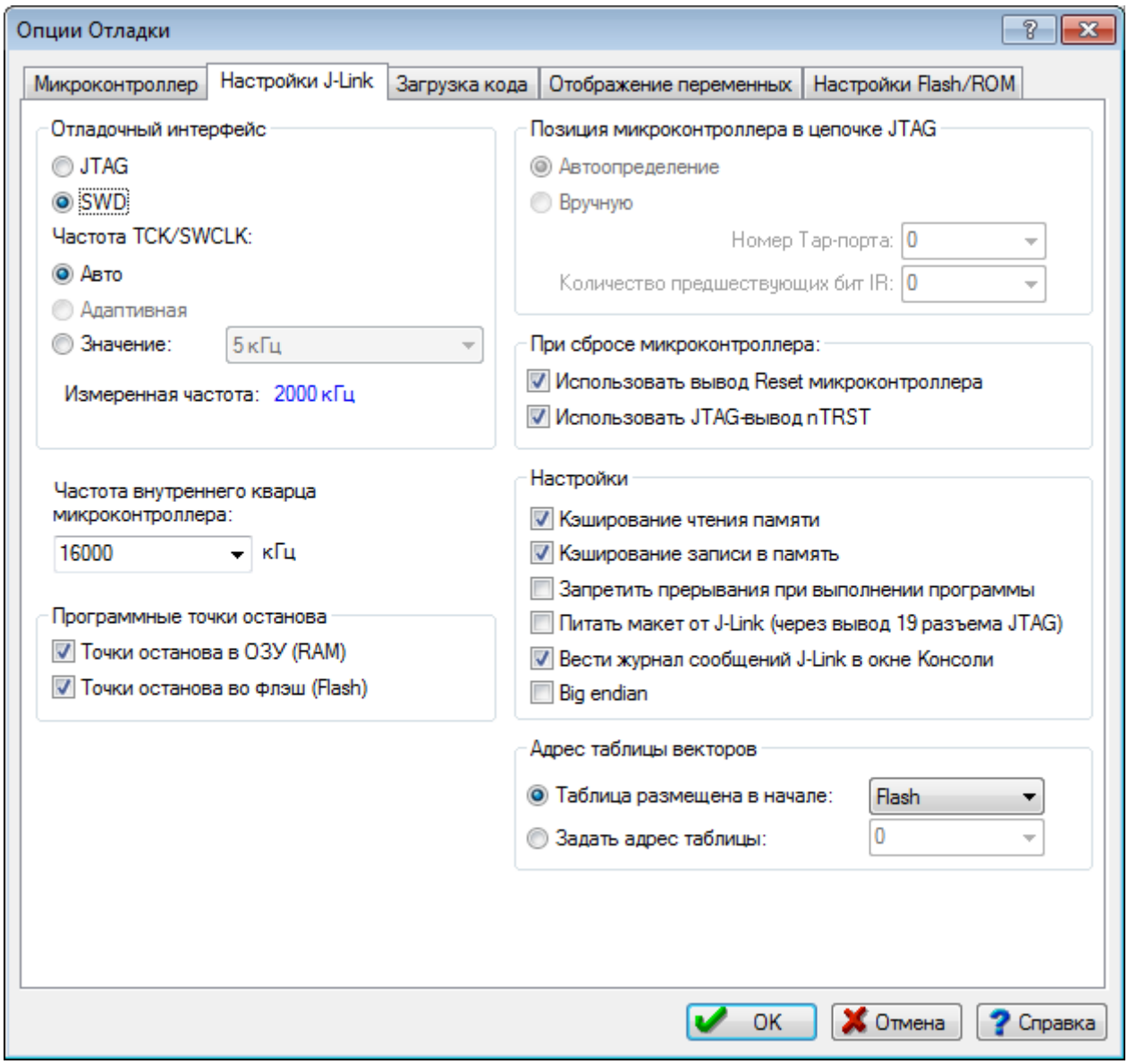


Рисунок 7 – Графическое окно настроек опций отладки

Для работы с отладочной платой необходимо в закладке «Микроконтроллер» (рисунок 8) выбрать микроконтроллер Milandr 1986BE91T1 и нажать кнопку OK.

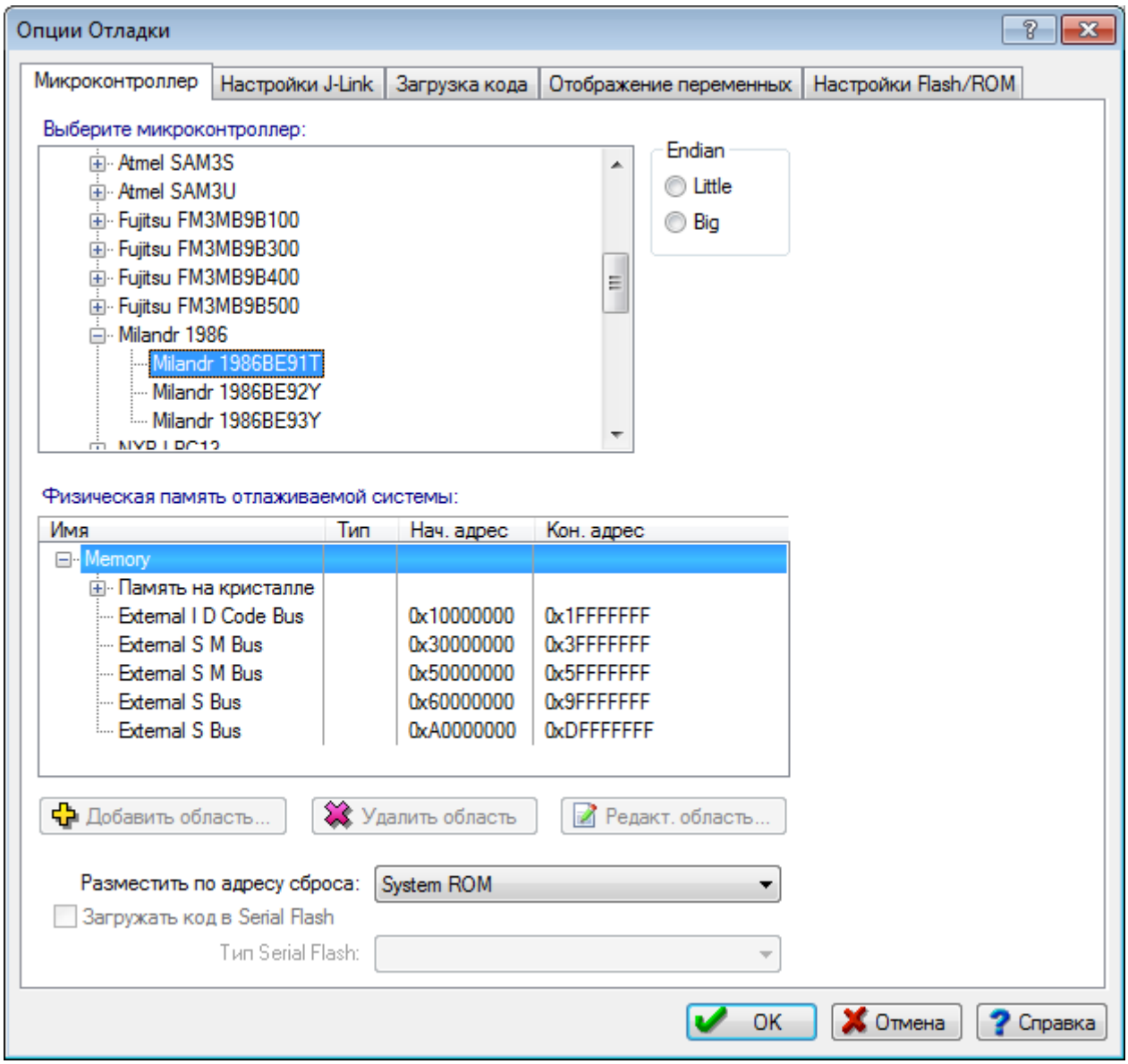


Рисунок 8 – Выбор микроконтроллера в окне настроек опций отладки

После старта программа выводит на дисплей меню. Нажимая на кнопки джойстика можно совершать переходы по меню и выбирать примеры вывода графической информации и управления светодиодами.

Нажмите на кнопку  (Собрать проект и запустить отладку) или клавишу[Shift + F7]. Проект будет откомпилирован с использованием кросс-средств, исполняемый файл будет загружен в память микроконтроллера. С этого момента начнется процесс отладки – на микроконтроллер будет подан сигнал сброса, после чего микроконтроллер остановится на первой инструкции после сброса по адресу0x0008. Для запуска программы на выполнение нажмите кнопку  (Запуск/Останов) или клавишу[F5]. Убедитесь, что пример работает– с

помощью кнопок на отладочной плате последовательно выбирайте команды меню и следите за отработкой команд на ЖКИ дисплее. В любой момент вы можете остановить выполнение программы кнопкой  (Запуск/Останов). Для продолжения отладки используйте любые команды в меню Отладка, самые популярные из них следующие:

Для продолжения отладки используйте любые команды в меню Отладка, самые популярные из них следующие:

Таблица. Команды отладки среды проектирования CodeMaster-ARM

Кнопка	Команда	Описание
	**Выполнить шаг**	Выполняет один шаг высокого уровня, т.е., одну команду исходного текста. Если не загружено ни одной программы или отсутствует информация о номерах строк программы, то эта команда выполняет одну машинную инструкцию, подобно команде **Выполнить инструкцию процессора**.
	**Выполнить шаг без захода в подпрограммы**	Выполняет один шаг высокого уровня без захода в функцию, если текущий оператор вызывает функцию. Если не загружено ни одной программы или отсутствует информация о номерах строк программы, то эта команда выполняет одну машинную инструкцию, подобно команде **Выполнить инструкцию без захода в подпрограммы**.
	**Выполнить инструкцию процессора**	Выполняет один шаг низкого уровня, т.е., одну машинную инструкцию.
	**Выполнить инструкцию процессора без захода в подпрограммы**	Выполняет один шаг низкого уровня без захода в подпрограмму, если текущая инструкция вызывает подпрограмму. Ставит точку останова на инструкции, следующей после текущей, и запускает непрерывное выполнение до точки останова. Так одной командой можно не только вызвать подпрограмму, но и выполнить циклы. Если текущей инструкцией является инструкция перехода или возврата из подпрограммы, то будет выполнена только эта инструкция.
	**Запуск/Останов**	Запускает или останавливает непрерывное выполнение программы.
	**Сброс процессора**	Выполняет сброс эмуляционного микроконтроллера. Кроме этого, эмуляционный микроконтроллер сбрасывается автоматически при загрузке программы для отладки. Чтобы отключить такой сброс используйте диалог **Опции отладки**.
	**Обновить экран**	Один раз обновляет изображение на экране, т.е. содержимое всех окон. Действует при непрерывном выполнении программы.

Для возврата в режим редактора достаточно начать редактирование текста в окне Редактора. Затем цикл сборки проекта и отладки может быть повторен.

Лабораторная работа №2. Знакомство с портами ввода вывода

Цель: Создание простого проекта в среде CodeMaster-ARM для ознакомления с функциональными возможностями микроконтроллера 1986BE91T и с демонстрационной платой.

В данной лабораторной работе необходимо зажигать светодиоды в определенной последовательности (определяется заданием преподавателя к данной лабораторной работе).

Работа с периферийными блоками ввода/вывода семейства Cortex-M (и других семейств микроконтроллеров) заключается в строго определенной последовательности действий:

1. Задать и подключить нужную периферийную частоту к выбранному блоку. Это достигается функциями модуля Сброса и тактовых частот (см. таблицу 78 на стр.162 технической документации на микроконтроллер 1986BE91T) [3].

   Пример:

```
#include <types.h>
#include    <C:\Program    Files\Phyton    CodeMaster-ARM    Demo\2_40_00\CMC-ARM\Inc\Milandr\1986\1986be91.h>


#define mask_PCLK_RST_CLK (1 << 4)
#define mask_PCLK_PORTD (1 << 24)

void main(void) {
RST_CLK->PER_CLOCK = mask_PCLK_RST_CLK | mask_PCLK_PORTD;
// далее следуют функции пользователя
}
```

2. Настроить порты ввода-вывода, соответствующие выбранному периферийному модулю. см. таблицу 2 (Описание выводов микроконтроллеров серии 1986BE9x стр. 10 технической документации на микроконтроллер 1986BE91T)
3. Сконфигурировать модуль в соответствии с желаемым режимом работы, используя регистры нужного модуля.
4. Сконфигурировать режимы прерывания, если будет использован данный вид обмена информацией.
5. Сконфигурировать режим прямого доступа к памяти, если будет использован данный вид обмена информацией.
6. Если по логике работы программы используемый блок ввода-вывода больше не нужен, его можно отключить, сняв генерацию тактовой частоты этого блока. При этом

потребление электроэнергии МК сократиться. Кроме этого сократиться «цифровой шум», что может быть важным для работы аналоговой периферии.

Для работы с микроконтроллером 1986BE91T понадобится библиотека 1986be91.h, находящаяся по умолчанию в папке «C:\Program Files\Phyton CodeMaster-ARM Demo\2_40_00\CMC-ARM\Inc\Milandr\1986\»

1986be91.h — заголовочный файл, стандартная библиотека, созданная компанией Фитон на языке Си для микроконтроллеров семейства Cortex-M производства Миландр. Содержит функции, структуры и макросы для облегчения работы с блоками микроконтроллеров 1986be91. Данный заголовочный файл необходимо подключать в каждом проекте.

Для отладки программ можно использовать 2 варианта работы с CodeMaster:

1. Внешний отладчик.

Программа компилируется внешним компилятором (Keil, IAR, ...) не из под среды CodeMaster, а в CodeMaster передается только результат компиляции – файл, программируемый в микроконтроллер. Если это HEX или BIN файл, то символьная информация в нем отсутствует и можно "шагать" только по ассемблерным инструкциям. Для других (поддерживаемых CodeMaster) вариантов символьная информация имеется и можно вести полноценную отладку. Для загрузки файла нужно выбрать меню «Файл / Загрузить программу для отладки...», или кликнуть по значку  на панели инструментов.

2. Работа с проектом.

Для работы с проектом необходимо создать (или загрузить имеющийся) проект.

Подробнее о создании проекта смотрите раздел «Разработка проекта» руководства пользователя CodeMaster-ARM.

После того как проект создан, необходимо:

1. добавить (создать) файлы проекта (*.c и *.h);
2. добавить ассемблерный файл *.mca (создается после компиляции проекта в рабочей директории проекта);
3. выбрать целевой микроконтроллер (кнопка  , закладка Микроконтроллер);
4. запустить компиляцию программы и, в случае успеха, ее запрограммировать микроконтроллер (кнопка ).

Пример исходного кода для выполнения данной лабораторной работы включает в себя исходный код главного исполняемого модуля и два заголовочных файла (библиотеки). Исходный код данных файлов приведен ниже:

```
/*=================================================================
 *  (C) 2015, ФГБОУ ВПО «КГТА им. В.А.Дегтярева»
 *  Демонстрационный проект для 1986BE91
 *
 *  Данное ПО предоставляется "КАК ЕСТЬ", т.е. исключительно как
 *  пример, призванный облегчить
 *  студентам выполнение лабораторной работы с использованием процессоров Milandr
 *  1986BE91T1. ФГБОУ ВПО «КГТА им. В.А.Дегтярева»
 *  не несет никакой ответственности за возможные последствия
 *  использования данного, или
 *  разработанного пользователем на его основе, ПО.
 *-----------------------------------------------------------------
 *  Файл main.c: главный файл проекта
 *=================================================================*/
#include <Inc\leds.h>
#include <C:\Program Files\Phyton CodeMaster-ARM Demo\2_40_00\CMC-
ARM\Inc\Milandr\1986\1986be91.h>

#define mask_PCLK_RST_CLK (1 << 4)
#define mask_PCLK_PORTD (1 << 24)

/* Маска включенных светодиодов */
u32 CurrentLights;

/* Сервисные утилиты */

void InitPortLED(void) {
    PORTD->FUNC &= ~((0x3FF << (LED0_OFS << 1)));   /* Port */
    PORTD->ANALOG |= LEDS_MASK;                     /* Digital */
    PORTD->PWR |= (0x155 << (LED0_OFS << 1));       /* Slow */
    PORTD->RXTX &= ~LEDS_MASK;
    PORTD->OE |= LEDS_MASK;
}

void ShiftLights(void) {
    u32 ovf;
    PORTD->RXTX = (PORTD->RXTX & ~LEDS_MASK) | (CurrentLights & LEDS_MASK);
    ovf = (CurrentLights & (1UL << 31)) != 0;
    CurrentLights <<= 1;
    CurrentLights |= ovf;
}

void LightsOnFunc(void) {
  u32 tck, tck_led;
  /* Запускаем "спецэффект" */
  CurrentLights = 0x00000001;
  tck = 0; tck_led = 0;
  while(1) {
    if (tck == tck_led) {
      tck_led += 350000;
      ShiftLights();
    }
    tck++;
  }
}

/* Главная функция */

void main(void) {
  RST_CLK->PER_CLOCK = mask_PCLK_RST_CLK | mask_PCLK_PORTD;
  InitPortLED();
```

```
  LightsOnFunc();
}
/*=====================================================================
 * Конец файла main.c
 *====================================================================*/
/*=====================================================================
 *
 *  (C) 2010, Phyton
 *
 *  Демонстрационный проект для 1986BE91 testboard
 *
 *  Данное ПО предоставляется "КАК ЕСТЬ", т.е. исключительно как
 *  пример, призванный облегчить
 *  пользователям разработку их приложений для процессоров Milandr
 *  1986BE91T1. Компания Phyton
 *  не несет никакой ответственности за возможные последствия
 *  использования данного, или
 *  разработанного пользователем на его основе, ПО.
 *
 *---------------------------------------------------------------------
 *
 *  Файл leds.h: Работа со светодиодами
 *
 *====================================================================*/

#ifndef __LEDS_H
#define __LEDS_H

#include <types.h>

/* Смещения в регистре и маски для конкретных светодиодов */
typedef enum {
    LED0_OFS    = 10,
    LED1_OFS    = 11,
    LED2_OFS    = 12,
    LED3_OFS    = 13,
    LED4_OFS    = 14,
    LED0_MASK   = (1 << LED0_OFS),
    LED1_MASK   = (1 << LED1_OFS),
    LED2_MASK   = (1 << LED2_OFS),
    LED3_MASK   = (1 << LED3_OFS),
    LED4_MASK   = (1 << LED4_OFS),
    LEDS_MASK   = (LED0_MASK | LED1_MASK | LED2_MASK | LED3_MASK | LED4_MASK)
} LEDS_Masks;


/* Маска включенных светодиодов */
extern u32 CurrentLights;

/* Сервисные утилиты */
void InitPortLED(void);
void ShiftLights(void);

#endif /* __LEDS_H */

/*=====================================================================
 * Конец файла leds.h
 *====================================================================*/


/*=====================================================================
 *
 *  (C) 2010, Phyton
 *
```

```
 *  Демонстрационный проект для 1986BE91 testboard
 *
 *  Данное ПО предоставляется "КАК ЕСТЬ", т.е. исключительно как
 *  пример, призванный облегчить
 *  пользователям разработку их приложений для процессоров Milandr
 * 1986BE91T1. Компания Phyton
 *  не несет никакой ответственности за возможные последствия
 * использования данного, или
 *  разработанного пользователем на его основе, ПО.
 *
 *------------------------------------------------------------------
 *
 *  Файл type.h: алиасы базовых типов и макросы
 *
 *================================================================*/

#ifndef __TYPES_H
#define __TYPES_H

typedef signed long s32;
typedef signed short s16;
typedef signed char s8;

typedef signed long const sc32;
typedef signed short const sc16;
typedef signed char const sc8;

typedef volatile signed long vs32;
typedef volatile signed short vs16;
typedef volatile signed char vs8;

typedef volatile signed long const vsc32;
typedef volatile signed short const vsc16;
typedef volatile signed char const vsc8;

typedef unsigned long u32;
typedef unsigned short u16;
typedef unsigned char u8;

typedef unsigned long const uc32;
typedef unsigned short const uc16;
typedef unsigned char const uc8;

typedef volatile unsigned long vu32;
typedef volatile unsigned short vu16;
typedef volatile unsigned char vu8;

typedef volatile unsigned long const vuc32;
typedef volatile unsigned short const vuc16;
typedef volatile unsigned char const vuc8;

typedef enum {FALSE = 0, TRUE = !FALSE} bool;

#endif /* __TYPES_H */

/*================================================================
 * Конец файла types.h
 *================================================================*/
```

Лабораторная работа №3. Знакомство с ШИМ

Цель: изучить основы формирования ШИМ сигналов с помощью микроконтроллеров

Цифровые устройства, например микроконтроллер, может работать только с двумя уровнями сигнала, т.е. ноль и единица или выключено и включено. Таким образом, можно легко использовать его для контроля состояния нагрузки, например, включить или выключить светодиод. Так же можно использовать его для управления любым электрическим прибором, используя соответствующие драйверы (транзистор, симистор, реле и т.д.). Но иногда нужно больше, чем просто "включить" и "выключить" устройство. Если необходимо управлять яркостью светодиода (или лампы) или скоростью двигателя постоянного тока или каким-либо нагревательным элементом, то цифровые сигналы просто не могут этого сделать. Эта ситуация очень часто встречается в цифровой технике и называется Широтно-Импульсной Модуляцией (рус. ШИМ, англ. PWM).

Почти все современные микроконтроллеры имеют специализированные аппаратные средства для генерации ШИМ-сигнала.

Такие цифровые устройства, как микроконтроллер, могут формировать только два уровня на выходных линиях, высокий = 5В или 3,3В и низкий = 0В. Для получения других значений среднего напряжения в диапазоне от низкого уровня до высокого, на выходе микроконтроллера формируется последовательность импульсов (рисунок 9).

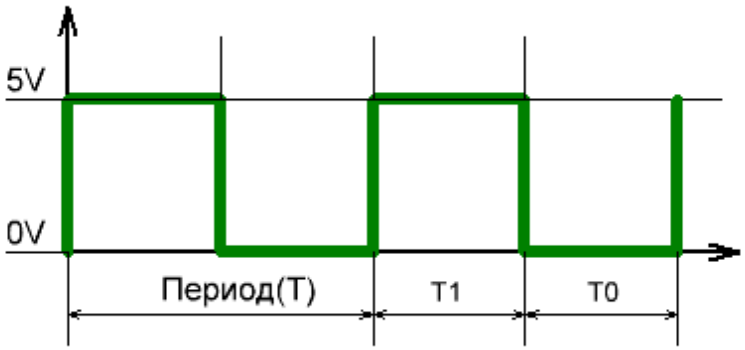


Рисунок 9 – Импульсный сигнал, формируемый со скважностью 50% (меандр)

Из рисунка видно, что сигнал на некоторое время остается поочередно на низком и высоком уровне. T0 - низкий уровень, T1 - высокий уровень. Период сигнала будет равен T = T0+T1. Период импульсов - это промежуток времени, между двумя характерными точками двух соседних импульсов. Обычно период измеряют между двух фронтов или двух спадов соседних импульсов и обозначают заглавной латинской буквой T.

Период следования импульсов напрямую связан с частотой импульсной последовательности F, и его можно вычислить по формуле: T = 1/F

Если длина импульса T1 точно равна половине периода T, то такой сигнал часто называют "меандр".

Скважностью импульсов называется отношение периода следования импульсов к их длительности и обозначается буквой S: S = T/T1

Скважность - безразмерная величина и не имеет единиц измерения, но может быть выражена в процентах. Часто в англоязычных текстах встречается термин Duty cycle, это так называемый коэффициент заполнения или величина рабочего цикла ШИМ. Коэффициент заполнения D является величиной, обратной скважности.

Коэффициент заполнения обычно выражается в процентах и вычисляется по формуле: D=1/S или так D = T1/T*100%

На рисунке 9 можно увидеть, что T1 = T0, это равно половине периода времени. Так величина рабочего цикла ШИМ составляет 50%. Если частота таких импульсов достаточно велика (например, 5000 Гц), то на выходе микроконтроллера формируется среднее напряжение 5В/2=2,5В. Таким образом, если выход контроллера связан с двигателем (с помощью соответствующих драйверов) он будет работать на 50% от его максимальной скорости. Ниже приведены некоторые примеры ШИМ сигнала различной скважности.

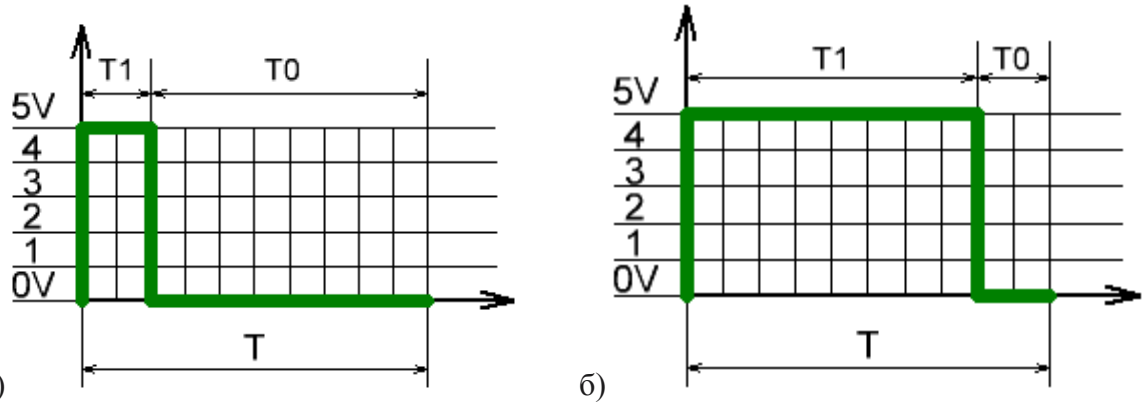


Рисунок 10

На рисунке 10а коэффициент заполнения равен 20%. Напряжение на выходе будет равно 20% от 5В, т.е. 1В.

На рисунке 10б Коэффициент заполнения равен 80%. Напряжение на выходе будет равно 80% от 5В, т.е. 4В.

Если на выходе поставить R/C фильтр, то можно получить чистый DC уровень сигнала, а не квадратные волны. Но это не требуется для коллекторных двигателей или для управления яркостью светодиодов. Для этого можно подавать ШИМ сигнал непосредственно на драйвер (например, биполярный транзистор, MOSFET и т.д.).

В данной лабораторной работе будет изучаться использование ШИМ сигнала для управления яркостью светодиода. Программный код написанный для микроконтроллера так же может быть пригоден для управления двигателем постоянного тока путем замены светодиода на электродвигатель M1, который управляется через мощный N канальный полевой транзистор (Power MOSFET) типа IRF510 (или мостовой схемы), на затвор которого подаются импульсы управления с вывода микроконтроллера (рисунок 11).

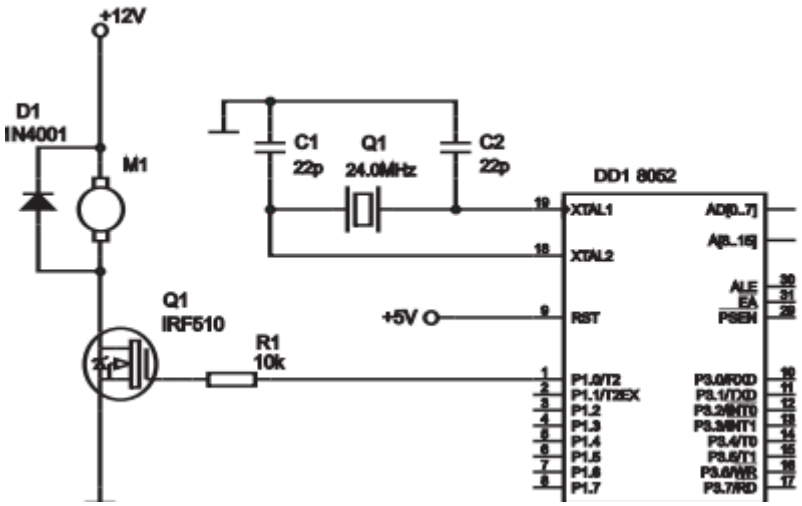


Рисунок 11 – Примерная схема реализации системы управления двигателем постоянного тока с использованием ШИМ сигнала, формируемого микроконтроллером

Микроконтроллеры позволяют довольно легко реализовать ШИМ, причем это решение очевидно. Оно основано на применении счетчика или таймера. Рассмотрим на практическом примере построение ШИМ, используя счетчик микроконтроллера.

```
/*=====================================================================
 *  (C) 2015, ФГБОУ ВПО «КГТА им. В.А.Дегтярева»
 *  Формирование ШИМ сигнала с использованием счетчика для 1986BE91
 *
 *  Данное ПО предоставляется "КАК ЕСТЬ", т.е. исключительно как
 *  пример, призванный облегчить
 *  студентам выполнение лабораторной работы с использованием процессоров Milandr
 *  1986BE91T1. ФГБОУ ВПО «КГТА им. В.А.Дегтярева»
 *  не несет никакой ответственности за возможные последствия
 *  использования данного, или
 *  разработанного пользователем на его основе, ПО.
 *---------------------------------------------------------------------
```

```
 *  Файл main.c: главный файл проекта
 *=====================================================================*/
    #include <Inc\leds.h>
    #include <C:\Program Files\Phyton CodeMaster-ARM Demo\2_40_00\CMC-
ARM\Inc\Milandr\1986\1986be91.h>

    #define mask_PCLK_RST_CLK (1 << 4)
    #define mask_PCLK_PORTD (1 << 24)

    /* Маска включенных светодиодов */
    u32 CurrentLights;

    /* Сервисные утилиты */

    void InitPortLED(void) {
        PORTD->FUNC &= ~((0x3FF << (LED0_OFS << 1)));   /* Port */
        PORTD->ANALOG |= LEDS_MASK;                     /* Digital */
        PORTD->PWR |= (0x155 << (LED0_OFS << 1));       /* Slow */
        PORTD->RXTX &= ~LEDS_MASK;
        PORTD->OE |= LEDS_MASK;
    }

    void LightsOnFunc(void) {
      u32 tck;
      /* Запускаем "спецэффект" */
      CurrentLights = LED0_MASK; //В начале горит только светодиод LED0
      tck = 0; //Период импульсного сигнала
      while(1) { // бесконечный цикл
        if (tck <= 500) // Время, в течение которого светодиод LED1 горит
          CurrentLights = CurrentLights | LED1_MASK;
        else
        {
          //Светодиод LED1 гаснет, светодиод LED0 горит в полную силу
          CurrentLights = LED0_MASK;
    /*  Сброс  счетчика  перед  началом  формирования  нового  периода  импульсного
сигнала  */
          if (tck == 1000) tck =0;
        }
    /*   Вывод   на   физический   порт   микроконтроллера   содержимого   внутренних
переменных */
        PORTD->RXTX = (PORTD->RXTX & ~LEDS_MASK) | (CurrentLights & LEDS_MASK);
        tck++;
      }
    }

    /* начало работы программы*/
    void main(void) {
      // инициализация тактовых сигналов для внутренних блоков микроконтроллера
      RST_CLK->PER_CLOCK = mask_PCLK_RST_CLK | mask_PCLK_PORTD;
      //Запуск функции инициализации порта PORTD
      InitPortLED();
      //Запуск функции управления яркостью светодиода
      LightsOnFunc();
    }
/*=====================================================================
 * Конец файла main.c
 *=====================================================================*/

/*=====================================================================
 *
 *  (C) 2010, Phyton
 *
 *  Демонстрационный проект для 1986BE91 testboard
 *
```

```
*  Данное ПО предоставляется "КАК ЕСТЬ", т.е. исключительно как
*  пример, призванный облегчить
*  пользователям разработку их приложений для процессоров Milandr
*  1986BE91T1. Компания Phyton
*  не несет никакой ответственности за возможные последствия
*  использования данного, или
*  разработанного пользователем на его основе, ПО.
*
*-----------------------------------------------------------------

    #ifndef __LEDS_H
    #define __LEDS_H

    #include <types.h>

    /* Смещения в регистре и маски для конкретных светодиодов */
    typedef enum {
        LED0_OFS    = 10,
        LED1_OFS    = 11,
        LED2_OFS    = 12,
        LED3_OFS    = 13,
        LED4_OFS    = 14,
        LED0_MASK   = (1 << LED0_OFS),
        LED1_MASK   = (1 << LED1_OFS),
        LED2_MASK   = (1 << LED2_OFS),
        LED3_MASK   = (1 << LED3_OFS),
        LED4_MASK   = (1 << LED4_OFS),
        LEDS_MASK   = (LED0_MASK | LED1_MASK | LED2_MASK | LED3_MASK | LED4_MASK)
    } LEDS_Masks;

    /* ROL */
    #define __SHLC(val, cnt) ((val << (cnt & 31)) | (val >> ((32 - cnt) & 31)))

    /* Маска включенных светодиодов */
    extern u32 CurrentLights;

    /* Сервисные утилиты */
    void InitPortLED(void);

    #endif /* __LEDS_H */

/*=================================================================
 * Конец файла leds.h
 *================================================================*/
```

При выполнении данной лабораторной работы студент также должен самостоятельно исследовать влияние периода формируемого импульсного сигнала на качество системы управления яркостью светодиода и сделать вывод по итогам выполнения данной лабораторной работы.

Лабораторная работа №4. Знакомство с прерываниями (Сбрасывание счетчика?)

Цель: ознакомиться с работой прерываний на примере микроконтроллера 1986ВЕ91Т.

В ядро Cortex-M3 микроконтроллера 1986ВЕ91Т входит 24-битный вычитающий счетчик с функциями автоматической перезагрузки и генерации прерывания. Он называется таймером SysTick и предназначен для использования в качестве стандартного таймера во всех Cortex-микроконтроллерах. Таймер SysTick может использоваться для формирования шкалы времени в операционных систем реального времени (ОСРВ) или для генерации периодических прерываний для обработки запланированных задач. С помощью регистра управления и статуса таймера SysTick, который расположен в области системных ресурсов процессора Cortex-M3, пользователь может выбрать источник синхронизации таймера. Если установить бит CLKSOURCE, то таймер SysTick будет работать на тактовой частоте центрального процессорного устройства (ЦПУ). Если же его сбросить, то таймер будет работать на частоте, равной 1/8 тактовой частоты ЦПУ.

У таймера SysTick имеется три регистра. Для задания периодичности счета необходимо инициализировать регистр текущего значения и регистр перезагружаемого значения. В регистре управления и статуса имеются биты ENABLE, позволяющий активизировать работу таймера, и TICKINT, управляющий активностью линии прерывания таймера. Далее мы будем рассматривать структуру прерываний Cortex и использование таймера SysTick для генерации первой исключительной ситуации в микроконтроллере 1986ВЕ91Т.

Одними из главных усовершенствований ядра Cortex по сравнению с предшествующими ЦПУ ARM являются структура прерываний и механизм обработки исключительных ситуаций. ЦПУ ARM7 и ARM9 использовали две линии прерывания: быстродействующая линия прерывания и линия прерывания общего назначения. Данные линии поддерживали все источники прерываний в рамках микроконтроллера конкретного производителя. Однако, несмотря на использование казалось бы одинаковых подходов, конкретная реализация могла отличаться между разными производителями МК. Структуре прерываний ARM7 и ARM9 свойственны еще две проблемы. Во-первых, она недетерминистическая, т.к. время, которое требуется на завершение текущей инструкции при возникновении прерывания непостоянно. Тем не менее, во многих приложениях это не создает проблем, но в системах реального времени могут возникнуть большие трудности. Во-вторых, поддержка вложенных прерываний в архитектуре ARM7 и ARM9 реализована неэффективно и требует написания дополнительных кодов программы в виде ассемблерных макросов или ОСРВ. Таким образом, ключевой задачей, которая стояла перед разработчиками ядра Cortex, являлась преодоление всех этих ограничений и разработка стандартной структуры прерываний, отличающейся предельным быстродействием и детерминизмом.

Контроллер вложенных векторизованных прерываний (КВВП) является стандартным блоком ядра Cortex. Это означает, что у любого Cortex-микроконтроллера будет присутствовать одна и та же структура прерываний, независимо от его производителя. Таким образом, прикладной код и операционные системы можно легко переносить с одного МК на любой другой и, при этом, программисту не потребуется изучение нового набора регистров. При разработке КВВП также учитывалось, что задержка реагирования на прерывание должна быть очень малой. Данная задача решена двояким образом: собственно возможностями КВВП, а также набором инструкций Thumb-2, многоцикловые инструкции которого, как например, инструкции многократного чтения/записи, являются прерываемыми. Задержка реагирования на

прерывание то же является детерминистичной, что важно для систем реального времени. Как следует из наименования КВВП, им поддерживаются вложенные прерывания и, в частности, у микроконтроллера 1986ВЕ91Т используется 16 уровней приоритетов. Структура прерываний КВВП разработана с учетом программирования полностью на Си и исключает потребность в написании каких-либо ассемблерных макросов или специальных, несовместимых с ANSI, директив.

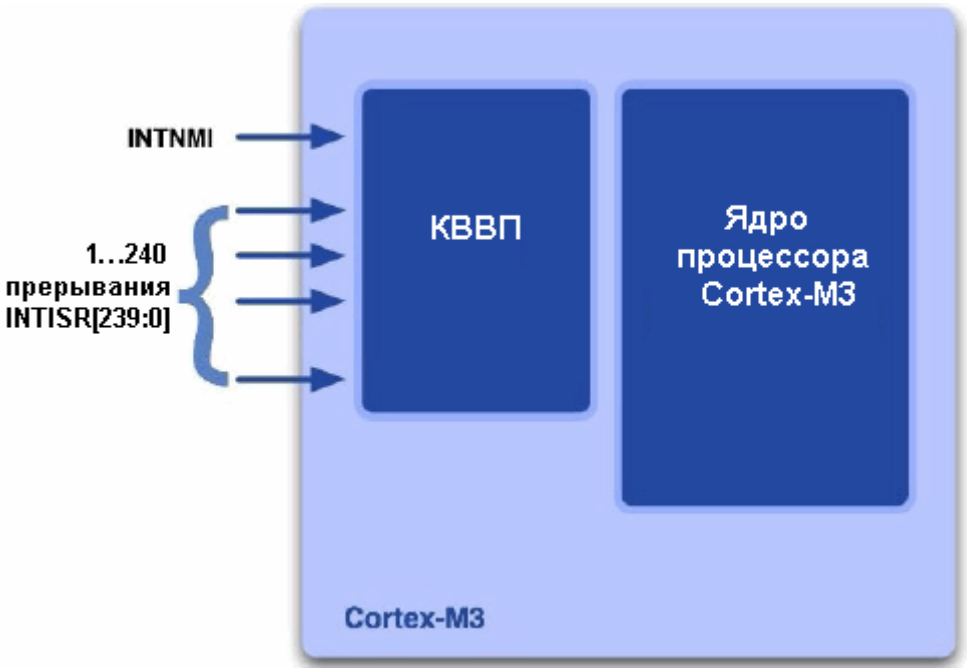


Рисунок 12 – В микроконтроллер 1986ВЕ91Т входит контроллер вложенных векторизованных прерываний, поддерживающий до 240 внешних УВВ

Несмотря на то, что КВВП является стандартным блоком ядра Cortex, в целях минимизации количества логических вентилей, разработчик микроконтроллера (МК) может сконфигурировать количество линий прерываний, идущих к КВВП. Контроллер поддерживает одно немаскируемое прерывание и еще до 240 внешних линий прерывания, которые можно подключить к пользовательским УВВ. В ядре Cortex поддерживается еще 15 дополнительных источников прерываний, использующихся для обработки внутренних исключительных ситуаций ядра Cortex. Максимальное число маскируемых линий прерывания КВВП микроконтроллеров 1986ВЕ91Т равно 43.

Если прерывание инициируется УВВ, то КВВП подключит ЦПУ Cortex к обработке прерывания. После перехода ЦПУ Cortex в режим прерывания, он помещает набор регистров в стек. Эта операция выполняется с помощью специального микрокода, что упрощает прикладной код. В процессе записи данных в стек на шине инструкций осуществляется выборка начального адреса процедуры обработки прерывания. Благодаря этому, с момента возникновения прерывания до выполнения первой инструкции его обработки проходит всего лишь 12 циклов. В них входит выполнение микрокода, который автоматически помещает набор регистров в стек.

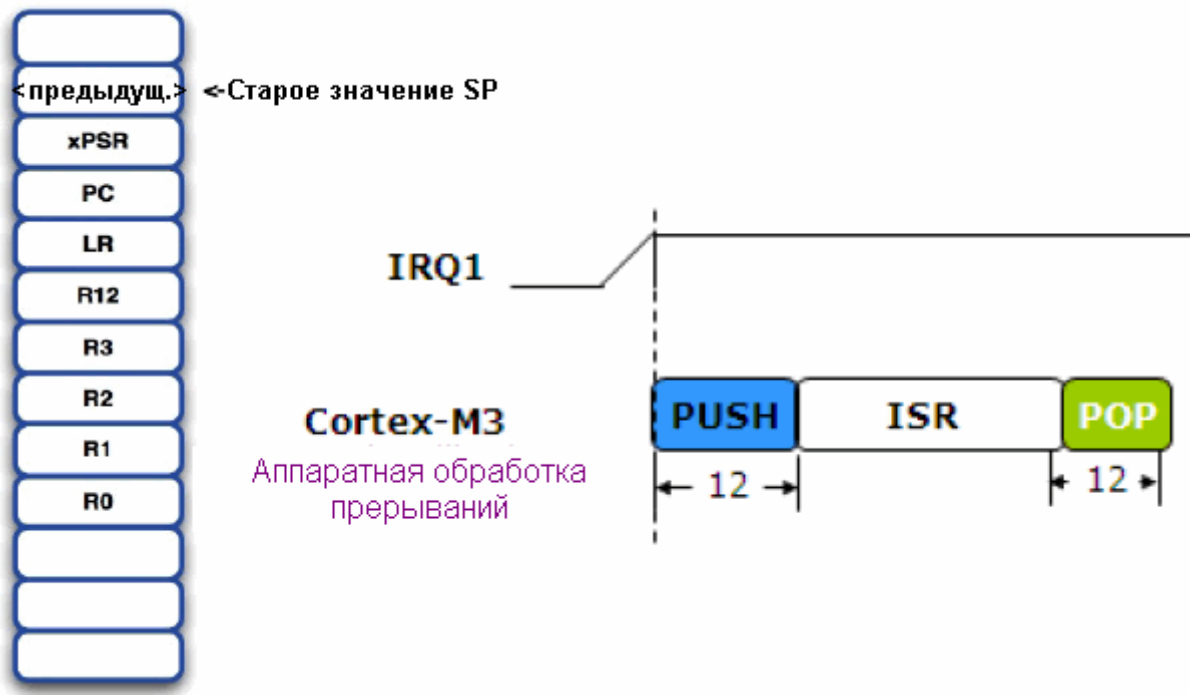


Рисунок 13 – КВВП реагирует на обработку прерывания с задержкой всего лишь 12 циклов

К числу помещаемых в стек данных относятся регистр статуса программы, счетчик программы и регистр связи. Благодаря этому, запоминается состояние, в котором находилось ЦПУ Cortex CPU. Кроме того, также сохраняются регистры R0 - R3. Эти регистры широко используются в инструкциях для передачи параметров, поэтому, помещение в стек делает возможным их использование в процедуре обработке прерывания. Замыкает список помещаемых в стек регистров - R12. Он выступает в роли рабочего регистра внутри подпрограммы. Например, если в компиляторе активизировать проверку стека, то будет генерироваться дополнительный код, который при потребности в регистре ЦПУ будет использовать R12. По завершении обработки прерывания все действия выполнятся в обратном порядке: с помощью микрокода извлекается содержимое стека и, параллельно с этим, осуществляется выборка адреса возврата, таким образом, для возобновления выполнения фоновой программы потребуется 12 циклов.

Чтобы включить в работу КВВП необходимо выполнить три действия:

1. Сконфигурировать таблицу векторов используемых прерываний.
2. Настроить регистры КВВП с целью активизации и установки уровней приоритета прерываний КВВП.
3. Настроить УВВ и разрешить поддержку ими прерываний.

Таблица векторов Cortex начинается с нижней части адресного пространства. Однако таблица векторов начинается не с нулевого адреса, а с адреса 0x00000004, т.к. первые четыре байта используются для хранения начального адреса указателя стека.

Таблица 1. Таблица векторов исключительных ситуаций содержит адреса, которые загружаются в счетчик программы, когда ЦПУ переходит в исключительную ситуацию.

Номер	Тип исключительной ситуации	Приоритет	Тип приоритета	Описание
1	Reset	-3 (высший)	фиксированнный	Сброс
2	NMI	-2	фиксированнный	Немаскируемое прерывание
3	Hard Fault	-1	фиксированнный	Обработчик аварийный состояний по умолчанию, если другой не реализован
4	MemManageFault	0	устанавливаемый	Сбой в блоке защите памяти или доступ по несуществующему адресу
5	BusFault	1	устанавливаемый	Ошибки в интерфейсе AHB
6	UsageFault	2	устанавливаемый	Исключительные ситуации, вызванные программными ошибками
7-10	Reserved	N.A.	N.A.	
11	SVCall	3	устанавливаемый	Вызов системных служб
12	DebugMonitor	4	устанавливаемый	Точки прерывания, контрольные точки, внешняя отладка
13	Reserved	N.A.	N.A.	
14	PendSV	5	устанавливаемый	Отправляемый запрос системному устройству
15	SYSTICK	6	устанавливаемый	Срабатывание системного таймера
16	Прерывание 0	7	устанавливаемый	Внешнее прерывание 0
.....	.....	.....	устанавливаемый	.....
256	Прерывание 240	247	устанавливаемый	Внешнее прерывание 240

Каждый из векторов прерываний занимает 4 байта и указывает на начальный адрес каждой конкретной процедуры обработки прерывания. Первые 15 векторов - адреса обработки исключительных ситуаций, возникающих в ядре Cortex. К ним относятся вектор сброса, немаскируемое прерывание, управление авариями и ошибками, исключительные ситуации отладочной системы и прерывание таймера SysTick. Набором инструкций Thumb-2 также поддерживается инструкция, выполнение которой приводит к генерации исключительной ситуации. Начиная с 16 вектора, следуют адреса обработки прерываний пользовательских УВВ. Их назначение зависит от каждого конкретного производителя. В программе таблица векторов обычно приводится в отдельном файле и содержит адреса процедур обработки прерываний.

Например, если используется прерывание таймера SysTick, то объявление на Си процедуры обработки прерывания выполняется следующим образом:

```
void SysTick_Handler (void)

{
```

```
}
```

Теперь, когда сконфигурирована таблица векторов и объявлена процедура обработки прерываний, можно настроить КВВП на обработку прерывания таймера SysTick. Обычно, для этого выполняют две операции: задается приоритет прерывания, а затем разрешается источник прерывания. Регистры КВВП расположены в области системных ресурсов Cortex-M3 и доступ к ним возможен при работе ЦПУ только в привилегированном режиме

Настройка внутренних исключительных ситуаций процессора Cortex выполняется с помощью регистров системного управления и системных приоритетов, а пользовательских УВВ - с помощью регистров IRQ. Прерывание SysTick является внутренней исключительной ситуацией процессора Cortex и, поэтому, управляется через системные регистры. Некоторые внутренние исключительные ситуации постоянно разрешены. К ним относятся прерывание по сбросу, немаскированное прерывание, а также и прерывание таймера tSysTick, поэтому, никаких действий с КВВП по разрешению этого прерывания делать не нужно. Для настройки прерывания SysTick нам необходимо активизировать сам таймер и его прерывание с помощью соответствующего регистра управления:

```
SysTickCurrent = 0x9000;          //Начальное значение счетчика SysTick
SysTickReload = 0x9000;           //Перезагружаемое значение
SysTickControl = 0x07;           //Запуск счета и разрешение прерывания
```

Приоритет каждой внутренней исключительной ситуации Cortex можно задать в системных регистрах приоритета. У исключительных ситуаций Reset, NMI (NON MASKABLE INTERRUPT, немаскируемое прерывание) и hard fault он фиксированный. Этим гарантируется, что ядро всегда будет переходить к обработке известной исключительной ситуации. У всех остальных исключительных ситуаций имеется восьмибитное поле, которое расположено в трех системных регистрах приоритета. МК 1986ВЕ91Т используют только 16 уровней приоритета, поэтому, у них активно только 4 бита этого поля. Однако важно запомнить, что приоритет устанавливается четырьмя старшими битами.

Каждое пользовательское УВВ управляется через блоки регистров IRQ. У каждого такого УВВ имеется бит разрешения прерывания. Все эти биты находятся в пределах двух 32-битных регистров установки разрешения прерываний. Для отключения источника прерывания предусмотрены отдельные регистры отмены разрешения прерываний. У КВВП также имеются регистры отправленных и активных прерываний, которые позволяют отследить состояние источника прерывания.

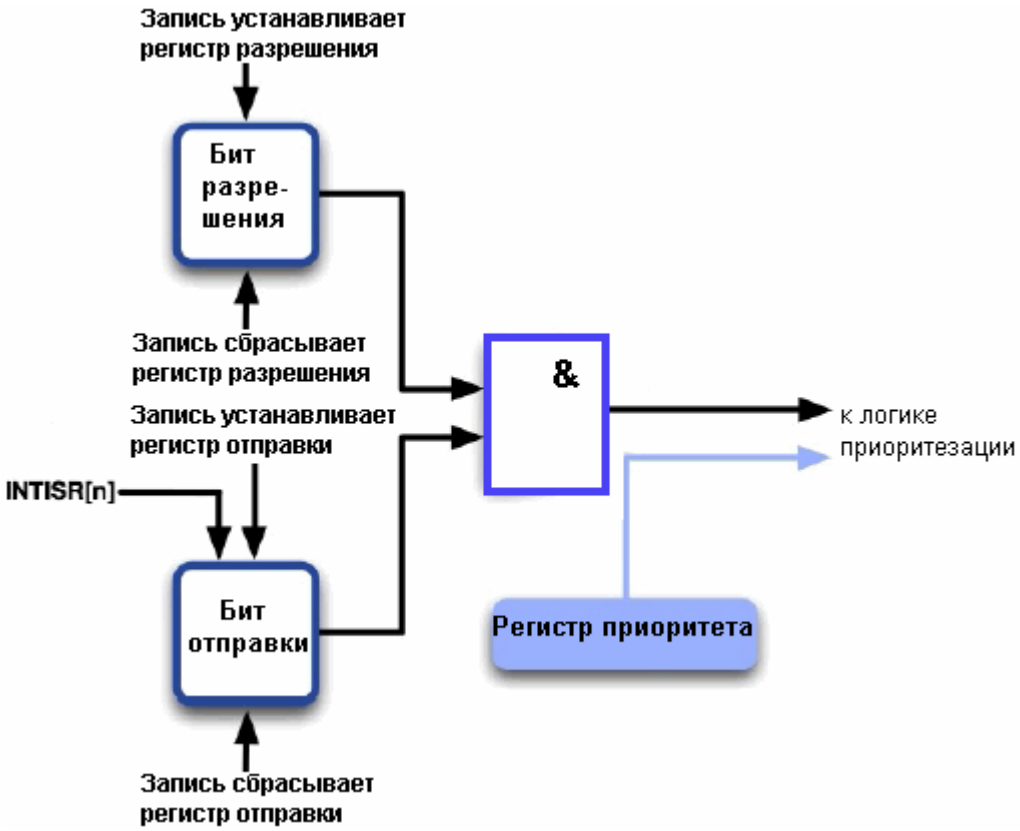


Рисунок 14 – У каждого источника прерывания имеется бит разрешения, как в КВВП, так и в УВВ. У МК STM32 используется 16 уровней приоритетов

Всего предусмотрено 16 регистров приоритета. Каждый из них разделен на четыре 8-битных поля для задания приоритета. Каждое поле связано с конкретным вектором прерывания. У МК 1986ВЕ91Т используется только половина такого поля, т.к. реализовано только 16 уровней приоритета. Однако необходимо помнить, что активные биты приоритета находятся в старшей тетраде поля. По умолчанию поле приоритета определяет 16 уровней приоритета, причем уровень 0 - наивысший приоритет, а 15 – самый низший. Поле приоритета также можно представить в виде групп и подгрупп приоритета. Это не добавляет дополнительных уровней приоритета, просто облегчает управление ими при необходимости задания в поле PRIGROUP регистра прикладных прерываний и управления сбросом большого числа прерываний.

Таблица 2. Поле PRIGROUP разделяет уровни приоритетов на группы и подгруппы

PRIGROUP (3 бита)	Положение запятой в двоичном числе (группа.подгруппа)		Группа приоритета		Подгруппа приоритета	
			Кол-во бит	Кол-во уровней	Кол-во бит	Кол-во уровней
011	4.0	гггг	4	16	0	0
100	3.1	гггп	3	8	1	2
101	2.2	ггпп	2	4	2	4
110	1.3	гппп	1	2	3	8
111	0.4	пппп	0	0	4	16

Трехбитное поле PRIGROUP управляет разделением 4-битных полей приоритета на группы и подгруппы. Например, запись в PRIGROUP числа 3 приведет к созданию двух групп с

4 уровнями приоритетов в каждой. После этого, вы можете в программе выполнить определения высокоприоритетной и низкоприоритетной групп прерываний. В рамках каждой группы можно задавать подуровни, в т.ч. низкий, средний, высокий и очень высокий. Ранее уже говорилось, что это позволяет более абстрактно смотреть на структуру прерываний и помогает программисту управлять большим числом прерываний. Конфигурация прерываний УВВ очень похожа на конфигурацию внутренних исключительных ситуаций процессора Cortex. Если взять в качестве примера прерывание АЦП, то вначале необходимо установить вектор прерывания и создать процедуру обработки прерываний:

```
DCD        ADC_IRQHandler ;
void ADC_Handler void
{
}
```

Затем необходимо инициализировать АЦП и разрешить прерывание в регистрах УВВ и КВВП:

```
ADC1->CR2          = ADC_CR2;              //Включение АЦП в режиме непрерывных преобразований
ADC1->SQR1     = sequence1;       //Выбор номеров каналов в очереди преобразования
ADC1->SQR2     = sequence2;       //и выбор каналов для преобразования
ADC1->SQR3     = sequence3;
ADC1->CR2      |= ADC_CR2;        //Перезапись бита включения
ADC1->CR1      = ADC_CR1;         //Запуск группы каналов, разрешение прерывания АЦП
GPIOB->CRH     = 0x33333333;      //Настройка светодиодных выводов на выход
NVIC->Enable[0] = 0x00040000;     //Разрешение прерывания АЦП
NVIC->Enable[1] = 0x00000000;
```

Задание к работе: Создать систему управления яркостью светодиодов с использованием прерываний и таймеров для реализации широтно-импульсной модуляции.

Пример выполнения задания:

Файл leds.h необходимо взять из лабораторной работы №3.

```
/*=================================================================
 *  (C) 2015, ФГБОУ ВПО «КГТА им. В.А.Дегтярева»
 *  Формирование ШИМ сигнала с использованием счетчика для 1986BE91
 *
 *  Данное ПО предоставляется "КАК ЕСТЬ", т.е. исключительно как
 *  пример, призванный облегчить
 *  студентам выполнение лабораторной работы с использованием процессоров Milandr
 *  1986BE91T1. ФГБОУ ВПО «КГТА им. В.А.Дегтярева»
 *  не несет никакой ответственности за возможные последствия
 *  использования данного, или
 *  разработанного пользователем на его основе, ПО.
 *-----------------------------------------------------------------
 *  Файл main.c: главный файл проекта
 *================================================================*/
#include     <C:\Program     Files\Phyton     CodeMaster-ARM     Demo\2_40_00\CMC-ARM\Inc\Milandr\1986\1986be91.h>
#include <Inc\leds.h>
#include <Inc\systick.h>

#define mask_PCLK_RST_CLK (1 << 4)
#define mask_PCLK_PORTD (1 << 24)

/* Маска включенных светодиодов */
u32 CurrentLights;
```

```
/* Сервисные утилиты */

void InitPortLED(void) {
    PORTD->FUNC &= ~((0x3FF << (LED0_OFS << 1)));   /* Port */
    PORTD->ANALOG |= LEDS_MASK;                     /* Digital */
    PORTD->PWR |= (0x155 << (LED0_OFS << 1));       /* Slow */
    PORTD->RXTX &= ~LEDS_MASK;
    PORTD->OE |= LEDS_MASK;
}

void LightsOnFunc(void) {
  u32 pwm_hi, pwm_total;
  /* Запускаем "спецэффект" */
  CurrentLights = LED0_MASK; // зажигаем светодиод LED0
  pwm_total = 200;   // период формирования импульсного сигнала
  pwm_hi=50;         // длительность высокого состояния
  while(1) {
    // включение светодиода LED1
    CurrentLights = CurrentLights | LED1_MASK; PORTD->RXTX = (PORTD->RXTX &
~LEDS_MASK) | (CurrentLights & LEDS_MASK);
    SysTickDelay(pwm_hi);
    // выключение светодиода LED1
    CurrentLights = LED0_MASK; PORTD->RXTX = (PORTD->RXTX & ~LEDS_MASK) |
(CurrentLights & LEDS_MASK);
    SysTickDelay(pwm_total-pwm_hi);
  }
}

void main(void) {
  RST_CLK->PER_CLOCK = mask_PCLK_RST_CLK | mask_PCLK_PORTD;
  InitPortLED();
  LightsOnFunc();
}
/*=====================================================================
 * Конец файла main.c
 *=====================================================================*/
/*=====================================================================
 *  (C) 2010, Phyton
 *  Демонстрационный проект для 1986BE91 testboard
 *  Данное ПО предоставляется "КАК ЕСТЬ", т.е. исключительно как пример,
призванный облегчить
 *  пользователям разработку их приложений для процессоров Milandr 1986BE91T1.
Компания Phyton
 *  не несет никакой ответственности за возможные последствия использования
данного, или
 *  разработанного пользователем на его основе, ПО.
 *---------------------------------------------------------------------------
 *
 *  Файл systick.c: Утилиты нижнего уровня для работы с системным таймером
 *
 *===========================================================================*/
#include <Inc\systick.h>
#include <C:\Program Files\Phyton CodeMaster-ARM Demo\2_40_00\CMC-
ARM\Inc\Milandr\1986\1986be91.h>

/* Счетчик ожидания */
static vu32 TimerCounter = 0;

/* Управление системным таймером */
void SysTickStart(u32 ticks) {
    SYS->STRVR = ticks;
    SYS->STCSR = mask_SYS_STCSR_ENABLE | mask_SYS_STCSR_TICKINT |
mask_SYS_STCSR_CLKSOURCE;
}
```

```
void SysTickStop() {
    SYS->STCSR &= ~mask_SYS_STCSR_ENABLE;
}


/* Обработчик прерывания SYSTICK */
void SysTick_Handler(void) {
    if (TimerCounter)
        TimerCounter--;
}


/* Функция задержки (на базе systick) */
void SysTickDelay(u32 ticks) {
    if (ticks) {
        TimerCounter = ticks;
        SysTickStart(ticks);
        while (TimerCounter);
        SysTickStop();
    }
}


/*============================================================================
 * Конец файла systick.c
 *============================================================================*/
```

Список использованных источников:

1. Шумилин С. Новая серия отечественных 32-разрядных высокопроизводительных микроконтроллеров семейства 1986 на базе процессорного ядра ARM Cortex-M3 // Компоненты и технологии. 2008. № 10.
2. Шумилин С. Характеристики производительности микроконтроллеров на базе ядра ARM Cortex-M3 // Электронные компоненты. 2009. № 8.
3. Предварительный вариант спецификации. Серия 1986ВЕ9х высокопроизводительных 32-разрядных микроконтроллеров на базе процессорного ядра ARM Cortex-M3.
4. Руководство по эксплуатации отладочной платы для МК 1986ВЕ91Т.
5. Комплект инструментальных средств для микроконтроллеров ЗАО «ПКК Миландр» 1986ВЕ91. Быстрый старт.
6. Пакет инструментальных средств CodeMaster-ARM для микроконтроллеров ПКК «Миландр» 1986ВЕ91хх с ядром Cortex-M3.
7. Инструментальные средства для разработки и отладки систем на базе микроконтроллеров Cortex-M3/M1/M0, ARM7/ARM9. Руководство пользователя.
8. www.phyton.ru
9. Спецификация. Жидкокристаллический модуль MT-12864J.
10. www.milandr.ru
11. Trevor Martin, "The Insider's Guide to the STM32 ARM Based Microcontroller, An Engineer's Introduction to the STM32 Series", Hitex Development Tools, Hitex (UK) Ltd., Coventry, United Kingdom, Apr. 21, 2008.