Уральский федеральный университет
имени первого Президента России Б.Н.Ельцина
Физико-технологический институт

И. Н. ОГОРОДНИКОВ

МИКРОПРОЦЕССОРНАЯ ТЕХНИКА: ВВЕДЕНИЕ В CORTEX-M3

Учебное пособие

Министерство образования и науки Российской Федерации

Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина

И. Н. Огородников

МИКРОПРОЦЕССОРНАЯ ТЕХНИКА: ВВЕДЕНИЕ В CORTEX-M3

Учебное пособие

*Рекомендовано методическим советом УрФУ
для студентов, обучающихся по направлениям подготовки
140801.65 «Электроника и автоматика физических установок»,
201000.62 «Биотехнические системы и технологии»,
140800.62 «Ядерные физика и технологии»*

Екатеринбург
Издательство Уральского университета
2015

УДК 004.31:004.42(075.8)
ББК 32.973.26-04я73
О-39

Рецензенты:
Институт химии твердого тела УрО РАН
(гл. науч. сотр., канд. физ.-мат. наук,
д-р хим. наук М. В. Кузнецов);
Д. И. Бескрестнов, директор, ООО «Авитек-плюс»

Огородников, И. Н.

О-39 Микропроцессорная техника: введение в Cortex-M3 : учеб. пособие / И. Н. Огородников. – Екатеринбург : Изд-во Урал. ун-та, 2015. – 116 с.

ISBN 978-5-7996-1499-7

Учебное пособие нацелено на формирование у студентов практических навыков разработки и программирования микропроцессорных устройств автоматики физических установок, приборов радиационной безопасности человека и окружающей среды, а также различных приборов биофизического и медицинского назначения.

Предназначено для студентов технических специальностей Физико-технологического института всех уровней обучения.

Библиогр.: 10 назв. Табл. 9. Рис. 44.

УДК 004.31:004.42 (075.8)
ББК 32.973.26-04я73

ISBN 978-5-7996-1499-7

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Предлагаемое учебное пособие основано на семестровом курсе по основам микропроцессорной техники, читаемом в Уральском федеральном университете для студентов Физико-технологического института, специализирующихся в областях электроники и автоматики физических установок, приборов для применения в области радиационной безопасности человека и окружающей среды, защиты от излучений, радиационной экологии, биомедицинской инженерии. Учебное пособие может быть полезно и для студентов других родственных специальностей. Рассмотрены практические вопросы программирования микропроцессорных устройств. Обсуждаются также некоторые вопросы проектирования устройств на их основе. Отметим, что данное учебное пособие касается лишь практической части учебного курса по основам микропроцессорной техники. Оно содержит дополнительный теоретический материал и техническую информацию, которые необходимы студентам при программировании микропроцессорных устройств на платформе Cortex-M3 во время практических занятий, лабораторного практикума, а также при выполнении индивидуального домашнего задания и курсовой работы. Отсюда название – введение в Cortex-M3. Теоретические основы микропроцессорной техники, излагаемые в лекционном курсе, содержатся в учебнике по микропроцессорной технике [1]. Детальное описание возможностей платформы Cortex-M3 можно найти в монографиях [2, 3].

Следует отметить, что микропроцессорная техника представляет собой весьма обширную, динамично развивающуюся область знаний. В данное пособие включен лишь ограниченный круг вопросов, выбор и глубина освещения которых продиктованы в первую очередь требованиями федеральных государственных образовательных стандартов высшего профессионального образования по направлениям подготовки 140800.62 «Ядерные физика и технологии», 201000.62 «Биотехнические системы и технологии» и специальности 140801.65 «Электроника и автоматика физических установок».

В первой главе приведены необходимые сведения об организации микропроцессорного ядра Cortex-M3. Глава является кратким введением в основы организации данного микропроцессорного ядра. После общей характеристики и описания возможностей и перспектив ядра Cortex-M3 обсуждаются наиболее важные особенности ядра: конвейер, регистровый файл, регистр статуса программы, режимы

работы микропроцессорного ядра, организация адресного пространства платформы Cortex-M3 и системный интерфейс. Далее идет обсуждение архитектурных особенностей микроконтроллеров STM32, выполненных на основе микропроцессорного ядра Cortex-M3. Уделено внимание организации внутренних шин, распределению памяти. Обсуждаются счетчики-таймеры общего и специального назначений, а также средства для эффективной работы в режиме реального времени, такие как схемы входного захвата и выходного сравнения. Специальное внимание уделено организации обработки прерываний, возможностям контроллера прерываний, программированию векторов прерываний. В завершении главы обсуждается схема тактирования и организация внутренних и внешних тактовых генераторов.

Вторая глава посвящена краткой характеристике отечественных микроконтроллеров с ядром Cortex-M3. Обсуждаются архитектура микроконтроллеров серии 1986BE9x, функциональная схема, режимы энергопотребления, цифровые интерфейсные узлы (параллельные порты ввода-вывода и последовательные интерфейсы UART, I2C, SPI, CAN, USB) и встроенные аналоговые блоки (аналого-цифровые и цифроаналоговые преобразователи, аналоговый компаратор), а также режимы работы микроконтроллера. Отдельно обсуждается организация демонстрационно-отладочной платы, ее общая характеристика, компоновка и интерфейс для подключения отладчика. В завершении главы обсуждаются средства разработки программного обеспечения. На примере интегрированной среды разработки Keil uVision продемонстрирована методика выбора и инсталляции среды разработки, а также организация программного проекта. Уделено внимание характеристике стандартных библиотек периферийных устройств.

Третья глава посвящена разработке авторских учебных программ, основное назначение которых – закрепить полученные теоретические знания по микроконтроллерам с ядром Cortex-M3. Основные разделы этой главы связаны с программированием периферийных устройств стенда (светодиодные индикаторы, графический дисплей, цифроаналоговый и аналого-цифровой преобразователи, встроенный аналоговый компаратор), программированием часов реального времени в домене батарейного питания микроконтроллера, программированием счетчика-таймера для работы в режиме прерываний. Все данные приведены в объеме, достаточном для их последующего практического использования.

Отметим, что рассматриваемый семестровый курс основ микропроцессорной техники является вводным курсом перед последующим изучением более сложного профессионально-ориентированного курса «Микропроцессорные системы».

1. ВВЕДЕНИЕ В ПЛАТФОРМУ CORTEX-M3

1.1. Введение

Период времени, прошедший с 2005 г., характеризуется значительными успехами в области разработки микроконтроллеров общего назначения на основе микропроцессорного ядра ARM7/ARM9, имеющего архитектуру с сокращенным набором команд (RISC) фирмы ARM Ltd. В настоящее время различными производителями выпускается почти две с половиной сотни различных типов ARM-микроконтроллеров.

Из зарубежных производителей необходимо прежде всего отметить компанию *ST Microelectronics*. Этой фирмой выпущено семейство микроконтроллеров STM32, выполненное на основе микропроцессорного ядра `ARM Cortex-M3`, которое считается эталоном по балансу рабочих характеристик и стоимости. Эти контроллеры изначально ориентированы на применение в приборах с малым энергопотреблением и жесткими требованиями к характеристикам управления в режиме реального времени.

В 2008 г. компания ЗАО «*ПКК Миландр*» (г. Зеленоград) приобрела у компании ARM лицензию на процессорное ядро ARM Cortex-M3. На базе данного процессорного ядра фирмой «Миландр» были разработаны высокопроизводительные 32-разрядные микроконтроллеры серии 1986ВЕ9х индустриального применения. Отечественные микроконтроллеры серии 1986ВЕ9х выпускаются в различных модификациях, отличающихся количеством выводов и некоторыми функциями.

Детальное изложение всех особенностей организации семейства Cortex-M3 можно найти в [2, 3]. Ниже мы обсудим только минимальный набор информации, необходимой для программирования платформы Cortex-M3.

1.2. Общая характеристика Cortex-M3

Семейство ARM Cortex – это новое поколение процессоров, которые выполнены по стандартной архитектуре и отвечают различным технологическим требованиям. В отличие от других микропроцессоров ARM, семейство Cortex является завершенным изделием, кото-

рое объединяет стандартное микропроцессорное ядро и системную архитектуру. Микроконтроллеры STM32 выполнены на основе ядра Cortex-M3, которое специально разработано для применений, где необходимы развитые системные ресурсы и при этом малое энергопотребление. Они характеризуются настолько низкой стоимостью, что могут конкурировать с традиционными 8- и 16-битными микроконтроллерами. Cortex-M3 является стандартизованным микроконтроллерным ядром, которое помимо 32-битного микропроцессорного ядра содержит систему прерываний, системный таймер SysTick, отладочную систему и предопределенную организацию памяти. Адресное пространство Cortex-M3 объемом 4 Гб разделено на четко распределенные области кода программы, статического ОЗУ, устройств ввода-вывода и системных ресурсов. В отличие от ядра ARM7, Cortex-M3 выполнен по гарвардской архитектуре и поэтому имеет несколько шин, позволяющих выполнять операции параллельно. Семейство Cortex имеет возможность оперировать с фрагментированными данными (*unaligned data*), что также отличает его от предшествующих архитектур ARM. Этим гарантируется максимальная эффективность использования внутреннего статического ОЗУ. Семейство Cortex также поддерживает возможности установки и сброса бит в пределах двух областей памяти размером 1 Мб по методу *bit banding*.

Еще одним ключевым компонентом ядра Cortex-M3 является контроллер векторизованных вложенных прерываний NVIC (англ. *Nested Vector Interrupt Controller*). Контроллер NVIC предоставляет стандартную структуру прерываний для всех Cortex-микроконтроллеров и способы их обработки.

Несмотря на то, что ядро Cortex-M3 разрабатывалось как недорогое ядро, оно остается 32-битным микропроцессорным ядром и, в связи с этим, поддерживает два режима работы: потоковый режим (*Thread*) и режим обработчика (*Handler*), для каждого из которых можно сконфигурировать свои собственные стеки. Благодаря этому появляется возможность разработки более интеллектуального программного обеспечения и поддержки операционных систем реального времени (ОСРВ). В ядро Cortex также входит 24-битный автоматически перезагружаемый таймер, предназначенный для генерации периодических прерываний и используемый ядром ОСРВ. Если у микропроцессоров ARM7 и ARM9 имеется два набора инструкций (32-битный ARM и 16-битный Thumb), то у семейства Cortex предусмотрена поддержка набора инструкций ARM Thumb-2. Этот набор представляет собой смесь 16- и 32-битных инструкций, которые позволяют добиться производительности 32-битного набора инструкций ARM и плотности кода, свойственной 16-битному набору инструк-

ций Thumb. Thumb-2 – обширный набор инструкций, ориентированный на компиляторы языков C/C++. Это означает, что программа для Cortex-микроконтроллера может быть полностью написана на языке программирования C/C++.

Семейство STM32 состоит из двух групп. Группа *Performance Line* работает на тактовых частотах до 72 МГц и оснащена полным набором устройств ввода-вывода (УВВ, англ. I/O), а группа *Access Line* работает на частотах до 36 МГц и интегрирует ограниченный набор устройств ввода-вывода.

Встроенные УВВ (два аналого-цифровых преобразователя АЦП, таймеры общего назначения, последовательные интерфейсы I2C, SPI, CAN, USB и часы реального времени RTC) гораздо сложнее аналогичных устройств для простых микроконтроллеров. Например, 12-битный АЦП оборудован датчиком температуры и поддерживает несколько режимов преобразования, а микроконтроллеры с двумя АЦП могут использовать их совместно еще в девяти режимах преобразования. По аналогии с этим каждый из четырех таймеров, оснащенных блоками захвата и сравнения, может использоваться как отдельно, так и совместно, образую более сложные счетные массивы таймеров. У расширенного таймера (advanced timer) добавлена поддержка управления электродвигателями. Для этого у него предусмотрено 6 комплементарных выходов от широтно-импульсных модуляторов (ШИМ) с программируемой паузой неперекрытия и вход экстренного останова, который переводит ШИМ-выходы в предварительно запрограммированное безопасное состояние. У интерфейса SPI предусмотрен аппаратный генератор контрольных сумм (CRC) для 8 и 16 слов, что упрощает реализацию интерфейса карт флэш-памяти SD и MMC.

В отличие от простых микроконтроллеров, у STM32 также предусмотрен многоканальный блок прямого доступа к памяти (ПДП). Каждый канал может использоваться для передачи данных между регистрами любого из УВВ и запоминающими устройствами 8/16 или 32-битными словами. Каждое из УВВ может выполнять роль потокового контроллера ПДП, при необходимости отправляя данные или посылая запрос на их получение. Арбитр внутренней шины и матрица шин минимизируют потребность в арбитраже доступа к шинам со стороны микропроцессорного ядра и каналов ПДП. Это означает, что блок ПДП является универсальным, простым в применении и реально автоматизирует передачу потоков данных внутри микроконтроллера.

Микроконтроллеры STM32 отличаются сочетанием характеристик малого энергопотребления и высокой производительности. Они способны работать от источника питания 2 В на тактовой частоте 72 МГц и потреблять при этом ток, с учетом нахождения в активном состо-

нии всех встроенных ресурсов, всего лишь 36 мА. Если же использовать поддерживаемые ядром Cortex экономичные режимы работы, то потребляемый ток можно снизить до 2 мкА в режиме STANDBY. Для быстроты возобновления активной работы микроконтроллера используется внутренний RC-генератор на частоту 8 МГц. Его активность сохраняется на время запуска внешнего генератора. Благодаря быстроте перехода в экономичный режим работы и выхода из них результирующая средняя потребляемая мощность электропитания еще больше снижается.

В STM32 интегрирован ряд аппаратных блоков, отвечающих за безопасность работы микроконтроллера, в т. ч. маломощный супервизор питания, система защиты синхронизации и два отдельных сторожевых таймера. Первый сторожевой таймер относится к оконному типу (windowed watchdog, WWDT). Его необходимо обновлять только в пределах отведенных временных рамок. Если это сделать слишком рано или слишком поздно, то он сгенерирует сигнал срабатывания. Другой сторожевой таймер полностью независим от первого. Он синхронизируется от отдельного внутреннего генератора, который не связан с главной системной синхронизацией. У микроконтроллера также используется система защиты синхронизации, которая может выявлять перебои в работе основного внешнего генератора и безопасно переключаться на работу от внутреннего RC-генератора частотой 8 МГц.

Флэш-память STM32 оснащена программируемой блокировкой чтения через отладочный порт. После активизации этой блокировки будет также невозможно записать что-либо во флэш-память, что исключает возможность внесения изменений в таблицу векторов прерываний. В остальной части флэш-памяти может быть активирована блокировка записи. У микроконтроллера STM32 также имеются часы реального времени и небольшая область энергонезависимого статического ОЗУ, которые питаются от отдельного резервного батарейного источника электропитания. В этой области имеется вход реагирования на внешнее вмешательство. При изменении состояния на данном входе генерируется прерывание и обнуляется содержимое энергонезависимого статического ОЗУ.

Иногда фирма ARM ссылается на свои процессоры по наименованию версии архитектуры. В этих обозначениях версия архитектуры процессора Cortex-M3 определяется как ARMV7M. Иными словами, процессор Cortex-M3 выполнен по архитектуре ARMV7M и поддерживает исполнение инструкций Thumb-2.

1.3. Микропроцессорное ядро Cortex

Основой процессора Cortex является 32-битное микропроцессорное ядро, имеющее RISC-систему команд. Данное ядро использует упрощенную модель программирования ARM7/9, но при этом более обширный набор инструкций с хорошей поддержкой целочисленной арифметики, улучшенными битовыми операциями и более строгими реально-временными характеристиками.

1.3.1. Конвейер

Микропроцессорное ядро Cortex способно выполнять большинство инструкций за один цикл, что достигается с помощью трехступенчатого конвейера: *выборка–дешифрация–выполнение*. Cortex поддерживает предсказание переходов для минимизации количества перезагрузок конвейера. Во время выполнения одной инструкции следующая инструкция дешифрируется, а третья инструкция считывается из памяти. Этот механизм отлично работает с линейным кодом, но если требуется выполнить переход, то, прежде чем продолжить выполнение кода программы, потребуется очистка и перезагрузка конвейера. Трехступенчатый конвейер микропроцессора Cortex оснащен логикой предсказания переходов. Это означает, что при достижении инструкции условного перехода выполняется упреждающая выборка. В результате оба назначения инструкции условного перехода будут доступны для исполнения и не произойдет снижения производительности. Хуже обстоят дела с инструкциями косвенного перехода, т. к. в этом случае упреждающую выборку выполнить нельзя и перезагрузка конвейера может оказаться неизбежной. Конвейер – это инструмент, от которого зависит результирующая производительность микропроцессора Cortex и который не требует каких-либо действий со стороны кода программы.

1.3.2. Регистровый файл

Микропроцессорное ядро Cortex-M3 содержит регистровый файл, состоящий из шестнадцати 32-битных регистров (R0–R15). У регистров R13–R15 имеются особые функции.

R13 выступает в роли указателя стека (SP). Данный регистр является банковым, что делает возможной работу Cortex в двух режимах работы, в каждом из которых используется свое собственное пространство стека. Данная возможность обычно используется ОСРВ, которые

могут выполнять свой *системный* код в защищенном режиме. У двух стеков Cortex имеются собственные наименования: основной стек и стек процесса.

R14 – регистр связи (LR). Он используется для хранения адреса возврата из подпрограммы. Благодаря этому регистру Cortex быстро переходит к подпрограмме и выходит из нее. Если же в программе используется несколько уровней вложений подпрограмм, то компилятор будет автоматически сохранять R14 в стек.

R15 – счетчик программы (PC). Он является частью центрального регистрового файла, его чтение и обработка могут выполняться аналогично любым другим регистрам.

1.3.3. Регистр статуса программы

Помимо регистрового файла, имеется отдельный регистр XPSR, который называется регистром статуса программы. Он не входит в основной регистровый файл, а доступ к нему возможен с помощью двух специальных инструкций. В XPSR хранятся значения полей, влияющих на исполнение инструкций Cortex. Регистр статуса программы содержит поля статуса, от которых зависит исполнение инструкций. Данный регистр разделен еще на три поля: статуса прикладной программы, исполнения программы и прерываний.

Регистр статуса программы содержит поля, от которых зависит исполнение инструкций. Данный регистр разделен на три поля: статуса прикладной программы, исполнения программы и прерываний (рис. 1.1). Биты регистра XPSR разделены на три группы, к каждой из которых возможен доступ по собственному наименованию.

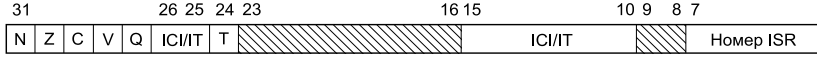


Рис. 1.1. Регистр статуса программы: ICI – возобновляемая прерыванием инструкция; IT – поле 'if then'; ISR – процедура обработки прерывания

Верхние пять бит (флаги кода условия) именуются полем статуса прикладной программы. Первые четыре флага кода условия N, Z, C, V (индикация отрицательного (N) или нулевого (Z) результата, переноса (C) и переполнения (V)) устанавливаются и сбрасываются по итогам выполнения инструкции обработки данных. Пятый бит Q используется при выполнении математических инструкций с насыщением алгоритмов цифровой обработки сигналов для индикации достижения переменной своего максимального или минимального значения. Так же как и 32-битные инструкции ARM, некоторые инструкции Thumb-2

выполняются только при условии совпадения кода условия инструкции и состояния флагов регистра статуса прикладной программы. Если коды условия инструкции не совпадают, то инструкция проходит по конвейеру как NOP (нет операции). Этим гарантируется равномерность прохождения инструкций по конвейеру и минимизируется число перезагрузок конвейера. У Cortex данный способ расширен полем статуса исполнения программы, который связан с битами [26:8][1] регистра XPSR. Это поле состоит из трех полей: поле 'if then' (IT), поле возобновляемой прерыванием инструкции и поле инструкции Thumb. Набор инструкций Thumb-2 реализует эффективный метод выполнения компактных блоков инструкций типа 'if then'. Если проверяемое условие истинно, записью значения в поле IT можно сигнализировать микропроцессору о необходимости выполнения до четырех следующих инструкций. Если же проверяемое условие – ложное, то данные инструкции пройдут по конвейеру как NOP.

Несмотря на то, что большинство инструкций Thumb-2 выполняются за один цикл, некоторые инструкции (например, инструкции чтения/записи) требуют для выполнения несколько циклов. Чтобы точно знать время отклика микропроцессорного ядра Cortex на прерывания, данные инструкции должны быть прерываемыми. В случае преждевременного прекращения исполнения инструкции в поле возобновляемых прерываниями инструкций запоминается номер следующего регистра, подлежащего обработке инструкцией многократного чтения или записи. Таким образом, сразу после завершения процедуры обработки прерывания выполнение инструкции многократного чтения/записи может быть восстановлено. Последнее поле Thumb предусмотрено для совместимости с предшествующими версиями микропроцессорного ядра ARM. Данное поле сигнализирует, что в настоящий момент микропроцессорное ядро выполняет инструкцию ARM или Thumb. У микропроцессорного ядра Cortex-M3 данный бит всегда равен единице. Наконец, в поле статуса прерывания хранится информация о любых приостановленных запросах прерывания.

1.3.4. Режимы работы микропроцессора

Процессор Cortex разрабатывался как быстродействующее, простое в использовании и с малым числом логических элементов микроконтроллерное ядро, в нем была учтена поддержка для работы ОСРВ. Процессор Cortex поддерживает два режима работы: режим *Thread* (или потоковый режим) и режим *Handler* (или режим обработчика).

[1] В квадратных скобках будем указывать диапазон битов для обсуждаемого регистра.

Микропроцессор запускается в режиме Thread при непрерываемом, фоновом выполнении инструкций и переключается в режим Handler при обработке исключительных ситуаций. Кроме того, микропроцессор Cortex может выполнять код программы в привилегированном или непривилегированном режиме. В привилегированном режиме ЦПУ имеет доступ ко всему набору инструкций, а в непривилегированном некоторые инструкции отключаются (например, инструкции MRS и MSR, осуществляющие доступ к регистру XPSR и его битовым группам). В этом режиме также отключается доступ к большинству регистров управления системными ресурсами процессора Cortex. Можно сконфигурировать использование стека. Основной стек (R13) может использоваться в обоих режимах (Thread и Handler). Режим Handler можно настроить на использование стека процесса (банковый регистр R13). Процессор Cortex-M3 может использоваться в обычном режиме ('flat'), а также поддерживает ОСРВ. У него предусмотрены режимы Handler и Thread с возможностями выбора используемого стека (основной стек или стек процесса) и привилегированного доступа к регистрам управления системными ресурсами Cortex. Сразу после сброса процессор Cortex запускается в конфигурации 'flat'. В обоих режимах (Thread и Handler) инструкции выполняются в привилегированном режиме, какие-либо ограничения на доступ к процессорным ресурсам отсутствуют. В режимах Thread и Handler используется основной стек. Чтобы начать выполнение инструкций, достаточно указать процессору вектор сброса и стартовый адрес стека, после чего можно выполнять код программы. Однако если используется ОСРВ или выполняется разработка критичного к безопасности приложения, процессор может использоваться в более изощренной конфигурации: режим Handler (используется при обработке исключительных ситуаций и ОСРВ) с привилегированными операциями и использованием основного стека и режим Thread при исполнении прикладного кода с непривилегированным доступом и использованием стека процесса. При таком подходе весь код программы разделяется на системный и прикладной, и поэтому ошибки в прикладном коде не вызывают сбоев ОСРВ.

1.3.5. Организация памяти Cortex-M3

Процессор Cortex-M3 является стандартизованным микроконтроллерным ядром, и поэтому его карта памяти четко расписана (рис. 1.2). Несмотря на использование нескольких внутренних шин, адресное пространство является линейным и имеет размер 4 Гб. Первый гигабайт памяти разделен равномерно между областью кода программы

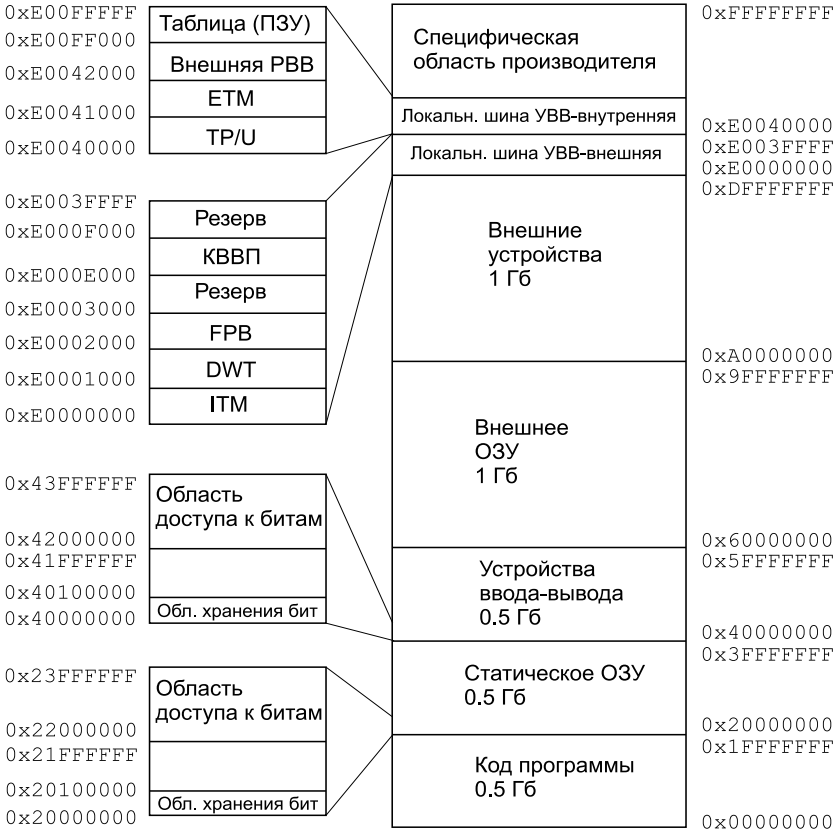


Рис. 1.2. Организация памяти микропроцессорного ядра Cortex-M3

и областью статического ОЗУ. Пространство кода программы оптимизировано для работы с шиной I-Code. Аналогично пространство статического ОЗУ доступно через шину D-code. Несмотря на то, что в области статического ОЗУ поддерживается загрузка и исполнение инструкций, их выборка осуществляется через системную шину, что требует дополнительного состояния ожидания. Таким образом, выполнение кода программы из статического ОЗУ будет более медленным, чем из встроенной флэш-памяти, расположенной в области кода программы. Следующие полгигабайта памяти – область встроенных УВВ. В этой области находятся все предоставляемые пользователю

производителем микроконтроллера УВВ. Первый мегабайт в каждой области памяти (статическое ОЗУ и УВВ) является битноадресуемым. Для этого используется метод *bit banding*. Таким образом, все данные, хранящиеся в этих областях, могут обрабатываться как пословно, так и побитно. Следующие два гигабайта адресного пространства выделены для внешних статического ОЗУ и УВВ. Последние полгигабайта зарезервированы для системных ресурсов и будущих расширений процессора Cortex. Все регистры процессора Cortex расположены по фиксированным адресам во всех Cortex-микроконтроллерах. Благодаря этому облегчается портирование программ между различными микроконтроллерами STM32 и Cortex-микроконтроллерами других производителей. Для процессора Cortex-M3 определена фиксированная карта памяти размером 4 Гб, в которой выделены конкретные области для хранения кода программы, статического ОЗУ, устройств ввода-вывода, внешней памяти и устройств, а также системных регистров. Карта памяти одинакова для всех Cortex-микроконтроллеров.

1.3.6. Системный интерфейс

Процессор Cortex-M3 выполнен по гарвардской архитектуре, которая подразумевает использование раздельных шин данных (D-bus) и инструкций (I-bus). Обе эти шины могут осуществлять доступ к инструкциям и данным в диапазоне адресов 0x00000000–0x1FFFFFFF. Также имеется дополнительная системная шина (System), которая предоставляет доступ к области системного управления по адресам

0x20000000–0xDFFFFFFF и 0xE0100000–0xFFFFFFFF.

У встроенной отладочной системы процессора Cortex имеется еще одна дополнительная шинная структура, которая называется локальной шиной УВВ (*Private Peripheral Bus*, PPB).

Системная шина и шина данных подключаются к внешнему микроконтроллеру через набор высокоскоростных шин, называемых матрицей шин. Она образует несколько параллельных соединений между шинами Cortex и другими внешними шинными мастерами, как, например, каналы ПДП к встроенным ресурсам, статическое ОЗУ и УВВ. Если два шинных мастера (например, микропроцессорное ядро Cortex и канал ПДП) предпринимают попытку доступа к одному и тому же УВВ, то вступит в действие внутренний арбитр, который разрешит конфликт, предоставив доступ к шине тому, кто имеет наивысший приоритет. Однако, благодаря тесной взаимосвязи блоков ПДП с микропроцессорным ядром Cortex, необходимость арбитража во многих случаях исключается.

1.4. Архитектура микроконтроллеров STM32

Фирма *ST Microelectronics* выпускает семейство микроконтроллеров STM32, которые выполнены на основе микропроцессорного ядра `ARM Cortex-M3`. Эти контроллеры считаются эталоном по балансу рабочих характеристик и стоимости. Они изначально ориентированы на применение в приборах с малым энергопотреблением и жесткими требованиями к характеристикам управления в режиме реального времени.

1.4.1. Организация внутренних шин

Микроконтроллеры STM32 выполнены на основе ядра Cortex-M3, которое подключено к флэш-памяти по отдельной шине инструкций (I-bus) (рис. 1.3). Поэтому обсудим лишь дополнительные детали, добавленные в STM32 к ядру Cortex-M3. Шина данных (D-bus) и системная шина (System) Cortex-M3 подключены к матрице высокоскоростных шин (*Advanced high-performance bus*, AHB). Внутреннее статическое ОЗУ (ведомое устройство, *slave*) подключено напрямую к матрице шин AHB, с которой также связан блок ПДП. Подключение встроенных УВВ распределено между двумя периферийными шинами (*Advanced peripheral bus*, APB). Каждая из этих шин связана с матрицей шин AHB посредством шинных формирователей. Матрица шин AHB синхронизируется той же частотой, что и ядро Cortex-M3. Однако у шин AHB имеются отдельные предделители, и поэтому в целях снижения энергопотребления их можно синхронизировать более низкими частотами. Важно обратить внимание, что шина APB2 может ра-

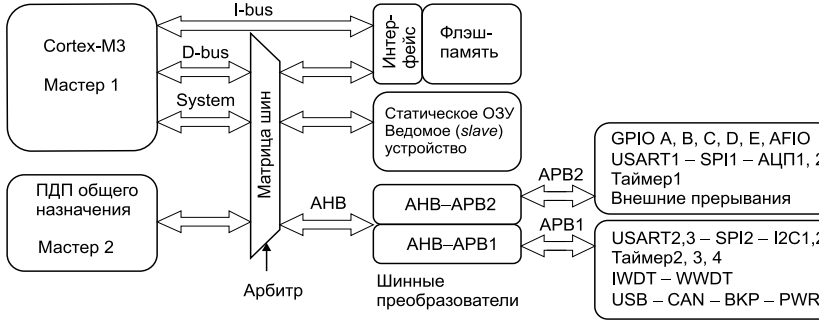


Рис. 1.3. Организация внутренних шин STM32

ботать с максимальным быстродействием 72 МГц, а быстродействие шины APB1 ограничено частотой 36 МГц (см. рис. 1.3).

Шина APB2 обслуживает следующие периферийные устройства:
– порты I/O общего назначения (General purpose I/O, GPIO);
– альтернативные функции портов I/O (Alternate function I/O, AFIO);
– контроллеры последовательных интерфейсов (USART1 и SPI1);
– контроллер аналого-цифровых преобразователей (ADC1 и ADC2);
– таймер-счетчик (TC1);
– внешние прерывания (EXTI).

Шина APB1 обслуживает следующие периферийные устройства:
– контроллеры интерфейсов (USART2,3, SPI2, I2C1,2);
– таймеры-счетчики (TC2, 3, 4);
– сторожевые таймеры – IWDT (независимый) и WWDT (оконный);
– контроллеры последовательных интерфейсов (USB и CAN);
– контроллер управления электропитанием (PWR);
– регистры для резервного копирования данных (BKP).

В качестве шинных мастеров могут выступать как CPU Cortex, так и блок ПДП. Благодаря свойственному матрице шин параллелизму, необходимость в арбитраже возникает только в случае попыток одновременного доступа обоих мастеров к статическому ОЗУ, шине APB1 или APB2. Тем не менее шинный арбитр гарантированно предоставляет 2/3 времени доступа для блока ПДП и 1/3 для CPU Cortex. В структуре внутренних шин предусмотрены отдельная шина инструкций и матрица шин, которая предоставляет несколько каналов передачи данных для CPU Cortex и блока ПДП.

1.4.2. Распределение памяти

Микроконтроллеры STM32 имеют линейное адресное пространство размером 4 Гб (рис. 1.4). Распределение памяти соответствует стандартному распределению памяти Cortex-M3 (см. рис. 1.2). Память программ начинается с адреса 0x00000000. Встроенное статическое ОЗУ стартует с адреса 0x20000000. Все ячейки статического ОЗУ расположены в области хранения бит. Регистры УВВ представлены в карте памяти, начиная с адреса 0x40000000, и также расположены в области хранения бит УВВ. Наконец, регистры Cortex находятся в их стандартном месте, начиная с адреса 0xE0000000.

Карта памяти STM32 выполнена по стандарту Cortex. Первые 2 Кб памяти могут быть связаны с флэш-памятью, системной памятью или статическим ОЗУ, в зависимости от состояния выводов управления загрузкой (табл. 1.1).

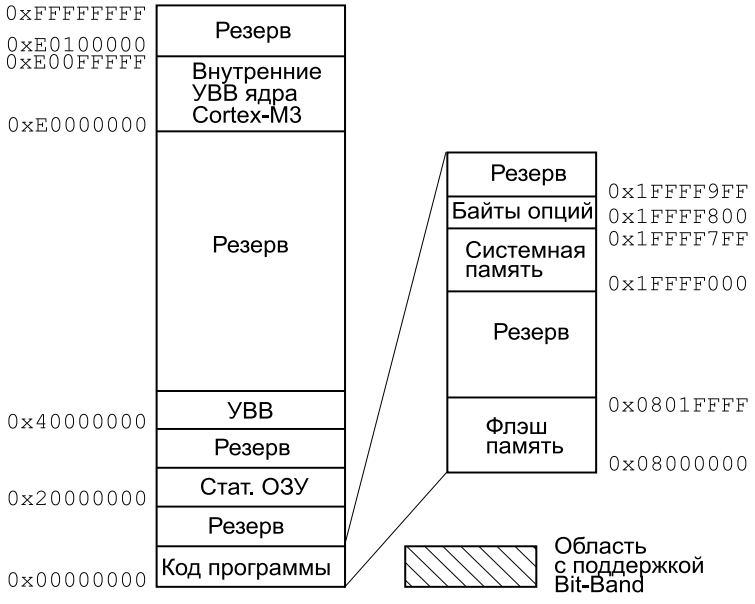


Рис. 1.4. Организация памяти STM32

Таблица 1.1

Выбор режима загрузки STM32

Вывод выбора режима загрузки		Режим загрузки
BOOT1	BOOT0	
*	0	Флэш-память пользователя
0	1	Системная память
1	1	Встроенное статическое ОЗУ

Область флэш-памяти разделена на три секции:

– флэш-память пользователя, начинается с адреса 0x0000000;

– большой информационный блок, который также называется «системная память». Блок представляет собой флэш-память размером 4 Кб, которая может быть запрограммирована производителем кодом программы загрузчика;

– малый информационный блок, начинается с адреса 0x1FFFF800. В нем находится группа опциональных байт, с помощью которых можно повлиять на некоторые системные настройки микроконтроллера STM32.

Выводы управления загрузкой и внутрисистемное программирование. Микроконтроллер может начать свою работу в одном из трех различных режимов загрузки. Эти режимы выбираются с помощью выводов BOOT0 и BOOT1. От выбранного режима загрузки зависит, какую область карты памяти микроконтроллер будет считать началом памяти. Микроконтроллер может исполнять код программы из флэш-памяти, внутреннего статического ОЗУ или системной памяти. Если выбирается загрузка из системной памяти, то STM32 начнет свою работу с выполнения запрограммированной производителем загрузочной программы, которая позволяет пользователю перепрограммировать флэш-память внутрисистемно.

Режимы загрузки. Для работы в обычном режиме вывод BOOT0 необходимо соединить с GND. Если же планируется использование других режимов, необходимо предусмотреть джамперы для задания различных состояний на выводах управления загрузкой. Выводы управления загрузкой позволяют указать, какая область памяти будет использоваться как первые 2 Кб памяти. В их качестве могут выступать флэш-память, встроенная программа загрузчика или первые 2 Кб статического ОЗУ. Обычно потребность в этом возникает при обновлении ПО уже на фазе эксплуатации продукции. Программа загрузчика для получения кода программы от ПК по умолчанию использует последовательный интерфейс УСАПП1, поэтому, если планируется ее использование, в схеме необходимо предусмотреть микросхему приемопередатчика RS232.

Программа загрузчика позволяет через интерфейс USART1 загрузить код программы и запрограммировать его во флэш-память пользователя. Чтобы перевести микроконтроллер STM32 в режим загрузчика, нужно на внешних выводах BOOT0 и BOOT1 установить низкий и высокий уровни соответственно. Если установить именно такие состояния на выводах управления загрузкой, то блок системной памяти начнется с адреса 0x00000000. После сброса микроконтроллера STM32 вместо выполнения прикладного кода из флэш-памяти пользователя начнется выполнение программы загрузчика. Чтобы пользователь имел возможность стирать и перепрограммировать флэш-память на компьютере, необходимо запустить еще одну программу загрузчика, которую можно скачать с сайта компании STM. Программа для ПК также доступна в виде DLL-файла, что позволяет создавать собствен-

ное ПО для программирования микроконтроллеров на фазе производства или эксплуатации продукции. С помощью выводов управления загрузкой адрес 0x00000000 вместо флэш-памяти пользователя может быть также связан со статическим ОЗУ. Поскольку загрузка статического ОЗУ осуществляется более быстро, то эта возможность может оказаться полезной на фазе проектирования для исполнения кода программы из статического ОЗУ. Кроме того, появляется возможность сократить частоту перепрограммирования флэш-памяти.

1.4.3. Таймеры общего и специального назначения

Любой микроконтроллер должен содержать несколько встроенных таймеров-счетчиков (ТС). У STM32 имеются системный таймер SysTick и четыре блока таймеров. Системный таймер SysTick является частью микропроцессорного ядра Cortex-M3. Таймер 1 – расширенный таймер, предназначенный для управления электродвигателем. Остальные таймеры являются таймерами общего назначения. Все таймеры выполнены по общей архитектуре, а расширенный таймер отличается лишь добавлением специальных аппаратных блоков.

Системный таймер SysTick. Микропроцессорное ядро Cortex-M3 содержит 24-битный вычитающий счетчик с функциями автоматической перезагрузки и генерации прерывания, который называется таймером SysTick. Он предназначен для использования в качестве стандартного таймера во всех Cortex-микроконтроллерах. Таймер SysTick может использоваться для формирования шкалы времени в ОСРВ или для генерации периодических прерываний для обработки запланированных задач. У таймера SysTick имеется три регистра. 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 микропроцессора (CPU). Если же его сбросить, таймер будет работать на частоте, равной 1/8 тактовой частоты CPU. 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.

Таймеры общего назначения. Все блоки таймеров (рис. 1.5) 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). Вход синхронизации счетчика можно связать с одним из восьми различных источников. В их число входят: специальный сигнал синхронизации, производный от сигнала главной системной синхронизации; выходной сигнал синхронизации одного из других таймеров или внешний сигнал синхронизации, связанный с выводами захвата/сравнения. Каждый из четырех таймеров микроконтроллера STM32 содержит 16-битный счетчик, 16-битный предделитель частоты и 4-канальный блок захвата/сравнения. Их можно синхронизировать системной синхронизацией, внешними сигналами или другими таймерами

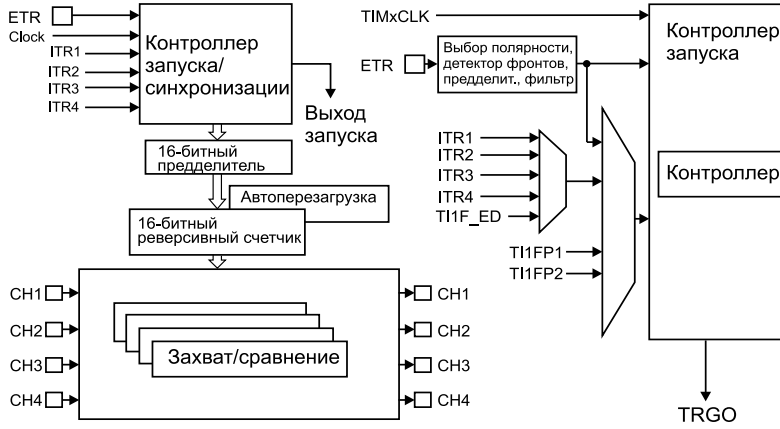


Рис. 1.5. Организация блока таймера общего назначения

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. Каждый из таймеров может генерировать прерывания и поддерживает ПДП.

1.4.4. Блок захвата/сравнения

Каждый канал захвата/сравнения (блок входного захвата, англ. *Input capture*, *IC*; блок выходного сравнения, англ. *Output compare*, *OC*) управляется через один регистр (рис. 1.6). Данный регистр имеет несколько функций, которые зависят от установок бит выбора. В

режиме захвата данный блок выполняет фильтрацию на входах, поддерживает специальный режим измерения внешнего ШИМ-сигнала, а также имеет входы для подключения внешнего энкодера. В режиме сравнения блок выполняет стандартные функции сравнения, генерации ШИМ-сигналов, а также поддерживает опциональную функцию одновибратора. У каждого канала захвата/сравнения имеется один регистр для задания режима работы.

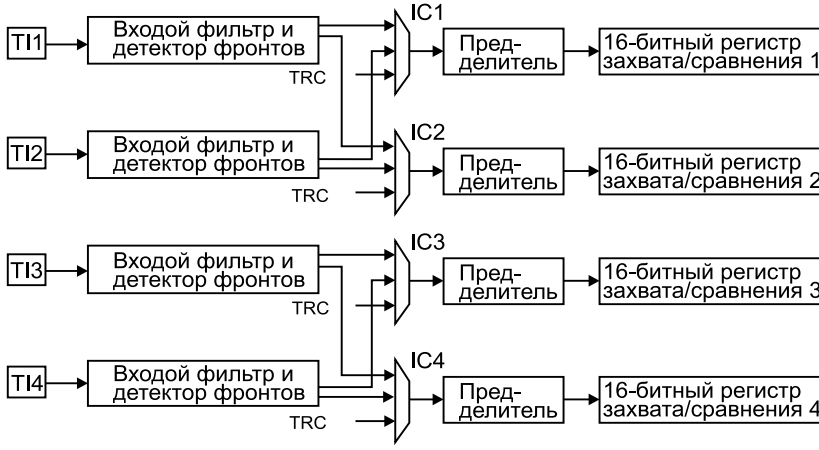


Рис. 1.6. Организация блока захвата/сравнения

Базовый блок захвата имеет 4 канала, подключенных к конфигурируемым детекторам фронтов. При обнаружении нарастающего или падающего фронта текущее значение счетчика записывается в 16-битный регистр захвата/сравнения. Когда возникает событие захвата, счетчик таймера может быть сброшен или приостановлен. Кроме того, одновременно с этим может быть запущено прерывание или ПДП-передача. Каждый из четырех блоков захвата имеет входной фильтр и детектор фронтов. При возникновении события захвата может быть запущено прерывание или ПДП-передача.

Каждый таймер микроконтроллеров STM32 имеет 4 канала схем сравнения. В базовом режиме сравнения генерируется событие захвата, если обнаруживается совпадение состояния счетчика с 16-битным значением, хранящимся в регистре захвата/сравнения. Данное событие может использоваться для изменения состояния, связанного с каналом захвата/сравнения вывода, генерации сброса таймера, прерывания или ПДП-передачи.

Блок входного захвата. Каждый блок захвата имеет возможность использования двух каналов захвата для автоматического измерения параметров внешнего ШИМ-сигнала, в т. ч. заполнения импульсов и периода следования импульсов. В режиме измерения параметров ШИМ-сигнала два канала могут использоваться для автоматического измерения периода и заполнения импульсов ШИМ-сигнала. В этом режиме входной сигнал соединен с двумя каналами захвата. В начале цикла ШИМ основной счетчик сбрасывается через второй канал захвата (нарастающий фронт ШИМ-сигнала) и начинает прямой счет. При возникновении падающего фронта ШИМ-сигнала срабатывает первый канал захвата, что приводит к фиксации значения заполнения импульсов. При обнаружении следующего нарастающего фронта вторым каналом захвата в начале следующего цикла ШИМ сбрасывается таймер и фиксируется значение периода ШИМ-сигнала.

Блок захвата всех таймеров также поддерживает возможность непосредственного подключения к внешнему энкодеру, который обычно используется для контроля угловых перемещений и частоты вращения электродвигателей. Каждый таймер имеет возможность подключения к линейному или поворотному энкодеру для контроля положения, скорости и направления В данной конфигурации выводы захвата выступают в роли входов синхронизации счетчика. Состояние счетчика в дальнейшем используется для определения положения. Для контроля скорости необходимо использовать еще один таймер. Он нужен для измерения интервала времени между импульсами сигнала энкодера.

Схема выходного сравнения. Каждый таймер микроконтроллеров STM32 имеет 4 канала схем сравнения. В базовом режиме сравнения генерируется событие захвата, если обнаруживается совпадение состояния счетчика с 16-битным значением, хранящимся в регистре захвата/сравнения. Данное событие может использоваться для изменения состояния, связанного с каналом захвата/сравнения вывода, генерации сброса таймера, прерывания или ПДП-передачи.

Помимо базового режима сравнения, каждый таймер поддерживает специальный режим генерации ШИМ-сигналов. В этом режиме период ШИМ задается с помощью регистра автоматической перезагрузки таймера. Значение заполнения импульсов задается через регистр захвата/сравнения канала. Таким образом, каждый таймер может генерировать до четырех независимых ШИМ-сигналов. Позже мы увидим, что таймеры могут работать и синхронизированно, позволяя генерировать до 16 синхронизированных ШИМ-сигналов.

Режим одновибратора. Как в базовом режиме сравнения, так и в режиме ШИМ блоки таймеров непрерывно генерируют прямоуголь-

ные импульсы. Однако, при необходимости, в каждом из таймеров можно задействовать опциональный режим одновибратора для генерации одиночного импульса. Данный режим, по сути, является особой версией режима ШИМ, в котором генерация ШИМ-сигнала инициируется внешним сигналом запуска (внешний вывод или выход запуска любого другого таймера) и длится один цикл. В результате генерируется один импульс с программируемой задержкой и длительностью.

Расширенный таймер. Расширенным таймером является блок таймера 1. Этот таймер дополнен рядом аппаратных узлов, предназначенных для управления электродвигателем. В каждом из трех каналов этого таймера предусмотрены два противофазных выхода. Таким образом, он представляет собой 6-канальный ШИМ-блок. Поскольку данный блок предназначен для управления трехфазным электродвигателем, у него имеется возможность программирования в каждом канале паузы неперекрытия и общий вход экстренного отключения. Кроме того, в дополнение к интерфейсу энкодера здесь предусмотрен интерфейс подключения датчиков Холла.

Расширенный таймер имеет возможность перевести свои ШИМ-выходы и их противофазные выходы в заранее запрограммированное состояние по сигналу на входе экстренного отключения. Источником данного сигнала может быть специальный внешний вывод экстренного отключения или система защиты синхронизации, которая контролирует работу внешнего высокочастотного генератора. После активизации функция экстренного отключения работает полностью автономно и гарантирует перевод ШИМ-выходов в безопасное состояние в случае отказа системы синхронизации микроконтроллера STM32 или в случае повреждения внешней схемы.

1.5. Обработка прерываний

Главными усовершенствованиями ядра Cortex по сравнению с предшествующими процессорами ARM являются аппаратная обработка прерываний и механизм обработки исключительных ситуаций.

1.5.1. Контроллер прерываний

Контроллер вложенных векторизованных прерываний NVIC 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 портировать с

одного микроконтроллера на любой другой, и при этом программисту не потребуется изучение нового набора регистров. При разработке NVIC также учитывалось, что задержка реагирования на прерывание должна быть очень малой. Данная задача решена двояким образом: собственно возможностями NVIC, а также набором инструкций Thumb-2, многоцикловые инструкции которого, такие как, например, инструкции многократного чтения/записи, являются прерываемыми. Задержка реагирования на прерывание тоже является детерминистичной, что важно для систем реального времени. Как следует из наименования NVIC, им поддерживаются вложенные прерывания (в частности, у микроконтроллера STM32 используется 16 уровней приоритетов). Структура прерываний NVIC разработана с учетом программирования полностью на Си и исключает потребность в написании каких-либо ассемблерных макросов или специальных, несовместимых с ANSI, директив. В процессор STM32 входит контроллер вложенных векторизованных прерываний, поддерживающий до 240 внешних УВВ.

Несмотря на то, что NVIC является стандартным блоком ядра Cortex, в целях минимизации количества логических вентилей разработчик МК может сконфигурировать количество линий прерываний, идущих к NVIC. 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 NVIC микроконтроллеров STM32 равно 43.

Если прерывание инициируется УВВ, то NVIC подключит микропроцессорное ядро Cortex к обработке прерывания. После перехода микропроцессора 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. Иными словами, NVIC реагирует на обработку прерывания с задержкой всего лишь 12 циклов. В них входит выполнение микрокода, который автоматически помещает набор регистров в стек. К числу помещаемых в стек данных относятся регистр статуса программы, счетчик программы и регистр связи. Благодаря этому запоминается состояние, в котором находился микропроцессор. 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 обработки 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 микрорпоцессора 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.

Помимо высокого быстродействия обработки одного прерывания, NVIC также характеризуется эффективной обработкой нескольких прерываний, что важно для высокобыстродействующих систем реального времени. Для этого у NVIC поддерживается несколько оригинальных методов, позволяющих обрабатывать несколько запросов прерываний с минимальными задержками между прерываниями и с гарантированием приоритетности обработки прерываний.

NVIC имеет возможность приостановки находящегося на обработке прерывания, если возникает прерывание с более высоким приоритетом. В этом случае обработка активного прерывания прекращается, в течение последующих 12 циклов выполняется сохранение в стек нового набора данных и запускается обработка высокоприоритетного прерывания. По завершении его обработки содержимое стека автоматически извлекается и обработка низкоприоритетного прерывания возобновляется.

Если на обработке находится высокоприоритетное прерывание и при этом возникает низкоприоритетное, NVIC Cortex использует метод непрерывной обработки с исключением внутренних операций над стеком, который гарантирует минимальность задержки при переходе к обработке следующего прерывания. Несколько прерываний обрабатываются непрерывно с исключением внутренних операций над стеком, поэтому задержка после завершения обработки одного прерывания и перед началом обработки следующего минимальна.

Если возникает два прерывания, первым со стандартной задержкой в 12 циклов обслуживается прерывание с более высоким приоритетом. Однако по окончании его обработки микропроцессорное ядро Cortex не возвращается к выполнению фоновой программы и содержимое стека не извлекается. Вместо этого осуществляется выборка адреса процедуры обработки следующего прерывания с учетом приоритета. Таким образом, задержка перехода к обработке следующего

прерывания составит всего лишь 6 циклов. По завершении обработки последнего прерывания извлекается содержимое стека и выполняется выборка адреса возврата. Таким образом, через 12 циклов возобновляется выполнение фоновой программы. Если же во время выхода из активного прерывания возникает еще одно низкоприоритетное прерывание, то операция извлечения из стека прекращается и указатель стека вернется к своему исходному значению, а выполнение обработки нового прерывания начнется через 6 дополнительных циклов. В итоге задержка вызова процедуры обработки нового прерывания будет в пределах 7–18 циклов.

В системах реального времени часто возникают ситуации, когда во время перехода к обработке низкоприоритетного прерывания возникает еще одно прерывание с более высоким приоритетом. Если такая ситуация возникнет на фазе загрузки стека, то по его завершении NVIC инициирует переход к обработке высокоприоритетного прерывания. Загрузка стека будет продолжаться еще минимум 6 циклов с момента возникновения высокоприоритетного прерывания, после чего будет выполнена выборка нового адреса процедуры обработки прерывания. Высокоприоритетное прерывание будет обработано первым, даже если оно возникнет уже на фазе перехода к обработке низкоприоритетного прерывания, при этом дополнительные операции над стеком будут исключены.

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

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. Набор инструкций Thumb-2 также поддерживает инструкцию, выполнение которой приводит к генерации исключительной ситуации. Начиная с 16 вектора, следуют адреса обработки прерываний пользовательских УВВ. Их назначение зависит от каждого конкретно-

го производителя. В программе таблица векторов обычно приводится в отдельном файле и содержит адреса процедур обработки прерываний (рис. 1.7).

```
/*-------------------------------------------------------------*/
          AREA RESET, DATA, READONLY
          EXPORT __Vectors
__Vectors DCD __initial_sp ; верхняя граница стека;
          DCD Reset_Handler ; сброс;
          DCD NMI_Handler ; немаскируемое прерывание;
          DCD HardFault_Handler ; авария типа HardFault;
          DCD MemManage_Handler ; авария блока защиты памяти;
          DCD BusFault_Handler ;авария типа BusFault;
          DCD UsageFault_Handler ; авария типа UsageFault;
          DCD 0 ; резерв;
          DCD 0 ; резерв;
          DCD 0 ; резерв;
          DCD 0 ; резерв;
          ; обработчик программно-сгенерированного прерывания;
          DCD SVC_Handler
         ; обработчик прерывания встроенной отладочной системы;
          DCD DebugMon_Handler
          DCD 0 ; резерв;
          DCD PendSV_Handler ; обработчик PendSV
          DCD SysTick_Handler ; прерывание от таймера SysTick
/*-------------------------------------------------------------*/
```

Рис. 1.7. Программирование векторов прерываний

Например, если используется прерывание таймера SysTick, то объявление на Си процедуры обработки прерывания выполняется следующим образом (рис. 1.8).

```
/*-------------------------------------------------------------*/
void SysTick_Handler (void)
{}
/*-------------------------------------------------------------*/
```

Рис. 1.8. Программирование обработчика прерываний

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

Настройка внутренних исключительных ситуаций процессора 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 прерывание таймера SysTick, поэтому никаких действий с NVIC по разрешению этого прерывания делать не нужно. Для настройки прерывания SysTick нам необходимо активизировать сам таймер и его прерывание с помощью соответствующего регистра управления.

Приоритет каждой внутренней исключительной ситуации Cortex можно задать в системных регистрах приоритета. У исключительных ситуаций Reset, NMI 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. Микроконтроллеры STM32 используют только 16 уровней приоритета, поэтому у них активны 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 NVIC также имеются регистры отправленных и активных прерываний, которые позволяют отследить состояние источника прерывания.

У каждого источника прерывания имеется бит разрешения, как в NVIC, так и в УВВ. У микроконтроллера 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 микроконтроллера STM32 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 – высший приоритет, а 15 – 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. Поле PRIGROUP разделяет уровни приоритетов на группы и подгруппы. Это необходимо для повышения программной абстракции при работе с большим числом прерываний. 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.

1.6. Тактовые генераторы

Микроконтроллер STM32 имеет внутренние RC-генераторы, способные синхронизировать встроенную схему фазовой автоподстройки частоты (ФАПЧ). Высокочастотный (HSI) генератор имеет частоту 8 МГц, а низкочастотный (LSI) генератор работает на частоте около 32 кГц. Благодаря их совместной работе микроконтроллеры могут синхронизироваться частотой до 72 МГц (9×8 МГц). Внутренние генераторы, по сравнению с кварцевыми, не отличаются такой же высокой точностью или стабильностью, поэтому во многих применениях может потребоваться использование как минимум одного внешнего кварцевого генератора.

Внешний высокочастотный генератор – это основной внешний источник синхронизации, который используется для тактирования процессора и УВВ. Совместно с генератором может использоваться кварцевый/керамический резонатор или отдельный источник синхронизации. Сигнал внешнего источника может иметь прямоугольную, синусоидальную или треугольную форму, но при этом заполнение импульсов должно быть 50 %-ным, а частота не более 25 МГц.

Частота внешнего кварцевого/керамического резонатора должна лежать в пределах 4–16 МГц. Чтобы добиться работы микроконтроллера на его максимальной частоте 72 МГц, необходимо выбрать частоту внешней синхронизации, которая бы нацело делила максимальную рабочую частоту. Это связано с тем, что внутренняя схема ФАПЧ умножает частоту HSE-генератора на целое число.

Внешний низкочастотный генератор. Микроконтроллеры STM32 могут иметь еще один внешний генератор, который называется внешним низкочастотным генератором (LSE-генератор). Он предназначен для синхронизации часов реального времени и оконного сторожевого таймера. Так же как и HSE-, LSE-генератор может работать совместно с кварцевым резонатором или внешним сигналом синхронизации прямоугольной, синусоидальной или треугольной формы и с заполнением импульсов 50 %. В каждом из этих случаев частота LSE-генератора должна быть равна 32768 Гц, что необходимо для точной работы часов реального времени. Часы реального времени также могут синхронизироваться внутренним низкочастотным генератором, однако вследствие его недостаточной точности обычно для реализации функций часов используется LSE-генератор.

Выход синхронизации. Одна из линий ввода-вывода может быть настроена как выход синхронизации (MCO). В этом режиме вывод MCO может генерировать один из четырех внутренних источников синхронизации. Об этом более детально пойдет речь при рассмотрении настроек внутренней системы синхронизации.

2. ОТЕЧЕСТВЕННЫЕ МИКРОКОНТРОЛЛЕРЫ С ЯДРОМ CORTEX-M3

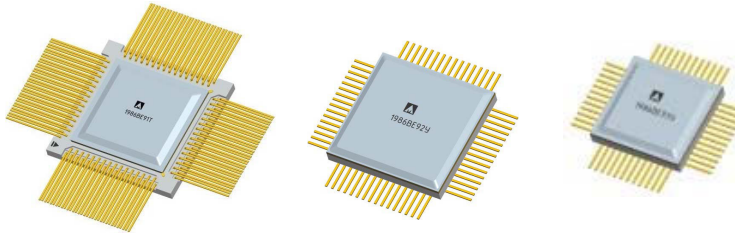
Приведены данные по отечественным микроконтроллерам с архитектурой Cortex-M3 и инструментальным средствам, необходимым для программирования платформы Cortex. Вначале рассмотрены особенности архитектуры микроконтроллеров 1986ВЕ9х фирмы «Миландр» (г. Зеленоград), пример организации демонстрационно-отладочной платы (Evalution board) типа 1986EvBrd_64 («Миландр»), а затем обсуждается интегрированная среда разработки Keil uVision (ARM).

2.1. Архитектура микроконтроллеров 1986ВЕ9х

2.1.1. Общая характеристика

В 2008 году «Дизайн-центр Миландр» (г. Зеленоград) приобрел исходные *Verilog* коды процессорного ядра ARM Cortex-M3. На базе данного процессорного ядра фирмой «Миландр» были разработаны высокопроизводительные 32-разрядные микроконтроллеры серии 1986ВЕ9х индустриального применения. Ближайший зарубежный аналог – STM32F103x. Отечественные микроконтроллеры серии 1986ВЕ9х выпускаются в различных модификациях, отличающихся количеством выводов и некоторыми функциями.

Существует несколько основных модификаций корпусов микроконтроллеров серии 1986ВЕ9х, рассчитанных на 132, 64 или 48 внешних выводов (рис. 2.1). Основой контроллеров 1986ВЕ9х, К1986ВЕ9х, К1986ВЕ92QI, К1986ВЕ92QC, К1986ВЕ91Н4 является один и тот же кристалл, помещенный в корпуса с различным количеством выводов. В связи с этим контроллеры имеют единую спецификацию [4]. Наибольшей функциональностью из них обладает микроконтроллер 1986ВЕ91Т, размещенный в корпусе со 132 выводами. В корпусах с 64 и 48 выводами соответствующее количество контактных площадок кристалла остаются неподключенными к внешним выводам. В табл. 2.1 подытожены основные функциональные отличия микроконтроллеров 1986ВЕ91Т (132 вывода), 1986ВЕ92У (64 вывода), 1986ВЕ93У (48 выводов). Более детальная информация содержится в спецификации [4].



а *б* *в*

Рис. 2.1. Основные модификации корпусов микроконтроллеров 1986ВЕ9х с количеством внешних выводов: *а* – 132; *б* – 64; *в* – 48

Таблица 2.1

Серия микроконтроллеров 1986ВЕ9х

Элемент	1986ВЕ91Т 1986ВЕ94Т	1986ВЕ92У 1986ВЕ92QI	1986ВЕ93У
Корпус	4229.132-3	Н18.64-1В	Н16.48-1В
Внешние выводы	132 вывода	64 вывода	48 выводов
Ядро	ARM Cortex-M3		
ПЗУ	128 Кб флэш-память		
ОЗУ	32 Кб		
Питание	2.2–3.6 В		
Частота	80 МГц		
Температура	от −60 до +125 °С*		
USER I/O	96	43	30
USB	Device и Host (Full Speed и Low Speed) встроенный PHY		
UART	2	2	2
CAN	2	2	2
SPI	2	2	2
I2C	1	1	1
RC генераторы	8 МГц и 40 кГц	8 МГц и 40 кГц	8 МГц и 40 кГц
Внешние генераторы	2–16 МГц и 32 кГц	2–16 МГц и 32 кГц	2–16 МГц
ADC	16 каналов	8 каналов	4 канала
DAC	2	1	1
Компаратор	3 входа	2 входа	2 входа
Внешняя шина	32 разряда	8 разрядов	—

* Только для металлокерамического корпуса.

2.1.2. Функциональная схема

Функциональная схема микроконтроллеров 1986BExx показана на рис. 2.2. На схеме выделены нижеследующие функциональные блоки.

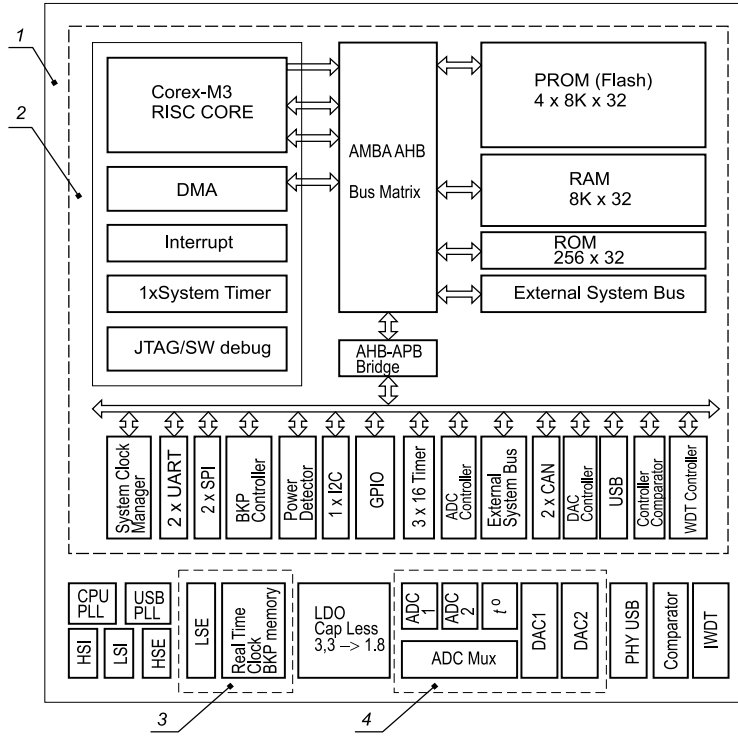


Рис. 2.2. Функциональная схема микроконтроллеров 1986BExx. Цифрами обозначены зоны электропитания: 1 – основное питание Ucc = 2.2–3.6 В; 2 – питание ядра DUcc = 1.8 В формируется встроенным LDO; 3 – батарейное питание BUcc = 1.8–3.6 В; 4 – аналоговое питание AUcc = 2.4–3.6 В

В зоне питания ядра (DUcc = 1.8 В) расположены:

[Core-M3 RISC CORE] – микропроцессорное ядро Cortex-M3;
[DMA] – многоканальный контроллер прямого доступа к памяти;
[Interrupt] – контроллер векторизованных прерываний;
[1×System Timer] – системный таймер SysTick;
[JTAG/SW debug] – отладчик с интерфейсами JTAG и SW;
[PROM (флэш-память)] – резидентная флэш-память программ 128 Кб;

[RAM] – оперативное запоминающее устройство 32 Кб;

[ROM] – постоянное запоминающее устройство 512 б;

[External System Bus] – внешняя системная шина;

[AMBA AHB Bus Matrix] – матрица шин в стандарте AMBA (Advanced Microcontroller Bus Architecture). AMBA является открытым протоколом для соединения функциональных блоков и управления ими. Протокол AMBA был разработан фирмой ARM в 1996 году. Для 32-разрядных микроконтроллеров он является *de facto* стандартом, поскольку хорошо документирован и может быть использован без финансовых отчислений правообладателю. Шины AHB и APB являются частью протокола AMBA.

[AHB–APB Bridge] – мост между шинами AHB и APB.

Периферийные устройства ядра: System Clock Manager – контроллер системы тактирования; контроллеры последовательных интерфейсов, первая цифра указывает их количество (2×UART; 2×SPI; 1×I2C; 2×CAN; 1×USB); BKP Controller – контроллер регистров резервного копирования данных; POWER detector – контроллер электропитания; GPIO (General purpose I/O) – порты ввода-вывода общего назначения; 3×16 Timer – три 16-битных таймера счетчика общего назначения; контроллеры АЦП (ADC), ЦАП (DAC), аналогового компаратора (Comparator) и сторожевого таймера (WDT).

В зоне основного питания (Ucc = 2.2–3.6 В) расположены:

умножители частоты с фазовой автоподстройкой (Phase Locked Loop, PLL) для системы тактирования центрального процессора (CPU) и последовательного интерфейса (USB);

внутренние RC-генераторы системы тактирования: LSI – низкочастотный (Low speed internal) и HSI – высокочастотный (High speed internal);

HSE – внешний высокочастотный тактовый генератор (High speed external);

[LDO Cap Less 3.3→1.8] – безконденсаторный регулятор напряжения (Capacitor-less low drop-out voltage);

PHY USB – чип PHY (Physical layer) интегрирован в большинстве контроллеров USB в хостах или встраиваемых системах и обеспечивает мост между цифровыми и модулированными частями интерфейса;

Comparator – аналоговый компаратор;

IWDT – независимый (independent WDT) сторожевой таймер.

В зоне батарейного питания (BUcc = 1.8–3.6 В) расположены:

LSE – внешний низкочастотный тактовый генератор (Low speed external);

RTC – часы реального времени (Real Time Clock);

[BKP memory] – регистры для резервного копирования данных.

В зоне аналогового питания (AUcc = 2.4–3.6 В) расположены:

ADC и DAC – преобразователи АЦП и ЦАП;
ADC Mux – мультиплексор входных каналов АЦП;
[t°] – температурный сенсор.

2.1.3. Режимы энергопотребления

Максимальное энергопотребление (120 мА). При этом включены все периферийные цифровые и аналоговые блоки; тактовая частота ядра и периферии 80 МГц; тактовая частота USB 48 МГц.

Режим *SLEEP* (до 40 мА). При этом могут быть включены все периферийные цифровые и аналоговые блоки; не тактируются ядро, флэш-память, ПДП (DMA); тактовая частота USB 48 МГц; пробуждение от прерываний.

Режим *SLEEPDEEP* (до 2 мА). При этом могут быть включены только АЦП, ЦАП, Компаратор, PWD, NVIC с тактированием от LSI; не тактируются ядро, флэш-память, ПДП (DMA), ОЗУ и внешняя шина; пробуждение от прерываний.

Режим *STANDBY* (до 15 мкА). При этом выключено питание DUcc; работают только LSI, LSE, RTC и BKP блоки; пробуждение от сигналов WAKEUP или ALARM RTC; время запуска не более 20 мкс.

Режим *BATTERY ONLY* (до 5 мкА). При этом выключены зоны питания DUcc и Ucc; работают только LSE, RTC и BKP блоки; пробуждение по появлению питания Ucc; время запуска не более 6 мс.

2.1.4. Цифровые интерфейсы

К цифровым интерфейсам относятся следующие функциональные узлы микроконтроллера:

– 6×портов ввода-вывода (до 96 User I/O);
– 2×UART (до 5 Мбит/с);
– 2×SSP (до 40 Мбит/с – ведущий *master*, до 6 Мбит/с – ведомый *slave*);
– 1×I2C (только режим ведущего *master*);
– 2×CAN (до 1 Мбит/с, 32 буфера сообщений);
– 1×USB (до 12 Мбит/с, встроенный PHY);
– 3×Timer (4 канала на таймер, функции ШИМ и захват);
– 1×DMA (Работа с ОЗУ, периферией и внешней шиной).

Порты ввода-вывода. 6 портов ввода-вывода, разрядность 16 бит:
– индивидуальное управление каждым выводом;
– универсальный цифровой пользовательский порт ввода-вывода

Iload = 4 мА Uil = 0.8 В Uol = 0.4 В;
Uimax = 5.5 В Uih = 2.0 В Uoh = 2.4 В;

– аналоговый режим работы вывода Uimax= Ucc* (порты PD[15:0] и PE[10:0] не толерантны к 5 В);
– управляемая мощность выходного драйвера

PowerTX = 01 – фронт 200 нс;
PowerTX = 10 – фронт 30 нс;
PowerTX = 11 – фронт 10 нс;

– встроенные резисторы подтяжки вывода

PullUp – 50 К; PullDown – 50 К;

– управляемый входной гистерезис

SHM= 0 (200 мВ) SHM= 1 (400 мВ);

– режим работы с открытым стоком;
– основная, альтернативная и переопределенная функции;
– для CAN, UART, SSP, Timer более 6 вариантов расположения.

Интерфейс UART. Два контроллера UART:
– FIFO очередь приема и передачи до 16 позиций;
– поддержка сигналов управления модемом:
CTS, DCD, DSR, RTS, DTR и RI;
– размерность данных 5, 6, 7 и 8 бит;
– бит четности;
– 1 или 2 стоп-бита;
– обнаружение ложного старт-бита;
– скорость до 5 Мбит/с;
– поддержка IRDA канала;
– переопределяемые выводы.

Интерфейс SSP. Два контроллера SPI:
– FIFO 16 б×8;
– режимы: младший бит вперед, старший бит вперед;
– режимы: Motorola SPI, Microwire, TI SPI;
– переопределяемые выводы.

Интерфейс USB. Встроенный аналоговый приемопередатчик:
– встроенные подтяжки линий D+ и D–;
– контроллеры Function и Host;
– скорость Low Speed (1.5 Мбит/с) и Full Speed (12 Мбит/с).

Интерфейс CAN. Два контроллера CAN:
– 32 буфера сообщений на контроллер;
– собственный фильтр для каждого буфера;
– скорость 1 Мбит/с;
– переопределяемые выводы.

Интерфейс I2C. Один контроллер I2C:
– только Master-режим;
– встроенный в вывод фильтр «иголок».

Внешняя системная шина. В состав внешней шины входят:
– внешняя память СОЗУ и ПЗУ;
– внешняя флэш-память и NAND флэш-память данных;
– внешние синхронные устройства;
– раздельная шина Данных и Адреса;
– nWE, nOE, nBE[3:0] сигналы;
– адресуемое пространство до 2.768 Гб.

Таймеры. 3 периферийных 16-разрядных таймера:
– встроенный в процессор Системный Таймер;
– режим каскадного объединения;
– каждый таймер содержит 4 канала Захвата/ШИМ;
– ШИМ-сигнал прямой и инверсный;
– формирования «мертвой» зоны сигналов ШИМ;
– внешний сброс ШИМ.

DMA. Контроллер прямого доступа в память:
– 32 канала ПДП;
– аппаратные запросы DMA от UART, SSP, Timer и ADC;
– программные запросы;
– передача между ОЗУ, периферией и внешней системной шиной;
– передача данных различной разрядности;
– различные механизмы расположения данных;
– основная и альтернативная структуры управления;
– различные приоритеты каналов;
– конфигурация DMA хранится в ОЗУ.

2.1.5. Аналоговые блоки

Аналоговые блоки представлены следующим списком функциональных узлов:
– встроенный регулятор напряжения (питание DUcc 1.8 В);
– 2×АЦП на 16 каналов (12 бит на 1 Мвыб/с);
– датчик температуры (в составе АЦП);
– датчик опорного напряжения (в составе АЦП);
– 2×ЦАП (12 бит, скорость преобразования 10 мкс);
– компаратор (3 внешних входа, внутренняя шкала напряжений);
– 2×PLL для умножения частоты (для ядра и USB);
– встроенные генераторы 8 МГц и 40 кГц (с подстройкой);
– внешние генераторы 2–16 МГц и 32.768 кГц;

– батарейный домен с часами реального времени;
– детектор напряжений питания (Ucc и BUcc).

Подсистема питания. Внешнее питание 2.2–3.6 В:
– внешнее питание 2.4–3.6 В при использовании АЦП и ЦАП;
– внешнее питание 3.0–3.6 В при использовании USB;
– внешнее батарейное питание 1.8–3.6 В;
– аппаратная схема сброса по включению и выключению питания;
– автоматический переход на батарейное питание.

Встроенный регулятор напряжения. Формирование напряжения питания для цифровой части 1.8 В:
– большая нагрузочная способность до 150 мА;
– не требует внешних конденсаторов;
– подстройка выходного напряжения;
– эффективная обработка появления нагрузки и сброса нагрузки;
– режим STANDBY.

Аналого-цифровой преобразователь. Два встроенных АЦП:
– разрядность до 12 бит;
– скорость преобразования до 1 Мвыб/с;
– внутренние и внешние опорные напряжения;
– 16 внешних каналов;
– автоматический последовательный опрос каналов;
– автоматический контроль диапазона;
– встроенный датчик опорного напряжения;
– встроенный датчик температуры.

Цифроаналоговый преобразователь. Два встроенных ЦАП:
– разрядность до 12 бит;
– внутренние и внешние опорные напряжения.

Аналоговый компаратор. Один встроенный аналоговый компаратор:
– три внешних входа (в некоторых корпусах – два входа);
– внутренний генератор шкалы напряжений.

Умножитель частоты (PLL). Умножитель снабжен фазовой автоподстройкой частоты:
– входная частота 2–16 МГц;
– выходная частота 2–100 МГц;
– коэффициенты умножения $\times 1$ – $\times 16$;
– сигнал готовности PLL_RDY;
– возможность формирования некратных частот.

Подсистема тактирования. Состав подсистемы тактирования:
– встроенный RC генератор HSI 8 МГц (с подстройкой);
– встроенный RC генератор LSI 40 кГц (с подстройкой);
– внешний генератор HSE 2–16 МГц;
– внешний генератор LSE 32 кГц (в батарейном домене).

Батарейный домен. 56 байт пользовательской памяти:
– регистры настройки микроконтроллера;
– внешний часовой генератор LSE 32768 Гц;
– часы реального времени с калибровкой от 0 до минус 256 ppm;
– пробуждение процессора из STANDBY режима по будильнику;
– формирование прерываний по событиям: «секунда», «будильник» и «переполнение»;
– программное преобразование в формат
Sec:Min:Hour:Day:Month:Year.

Детектор напряжения питания. Анализ Ucc и BUcc:
– анализ уровня основного питания Ucc с точностью 200 мВ;
– анализ уровня батарейного питания BUcc с точностью 400 мВ;
– прерывание при переходе заданного уровня.

2.1.6. Режимы работы микроконтроллера

В микроконтроллере доступны следующие режимы работы:
– режим «микроконтроллер» (отладка через JTAG/SW):
MODE[2:0]= 0 0 0; отладка через порт PD;
MODE[2:0]= 0 0 1; отладка через порт PB;
запуск из внутренней флэш-памяти;
– режим «микропроцессор» (отладка через JTAG/SW):
MODE[2:0]= 0 1 0; отладка через порт PD;
MODE[2:0]= 0 1 1; без отладки;
запуск из внешней памяти с адреса 0x10000000;
– режим «UART загрузчик» (доступен только в 1986BE91 и 1986BE92):
MODE[2:0]= 1 0 1; UART загрузчик на порте PD;
MODE[2:0] =1 1 0; UART загрузчик на порте PF;

– 2 порта (PD[1:0] и PF[1:0]);
– стандартный RS-232 интерфейс (скорость 9600);
– реализует команды:
CMD_LOAD – загрузить данные в память МК;
CMD_VFY – считать данные из памяти МК;
CMD_RUN – передать управление;
CMD_BAUD – задать скорость связи;
CMD_CR – запрос приглашения;
CMD_SYNC – пустая команда (для синхронизации).

На рис. 2.3 показано схематическое представление режимов загрузки микроконтроллеров 1986ВЕхх: микроконтроллер (связь через 5-выводной JTAG), микропроцессор (связь через 2-выводной SWD), UART-загрузчик (связь через RS-232).

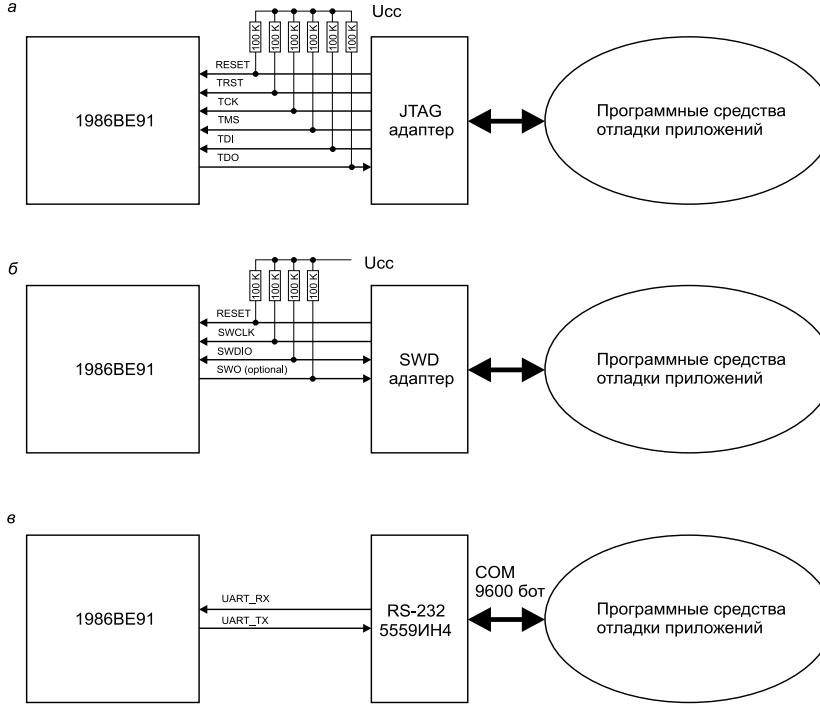


Рис. 2.3. Режимы работы: *а* – микроконтроллер; *б* – микропроцессор; *в* – UART загрузчик

2.2. Демонстрационно-отладочные платы

ЗАО «ПКК Миландр» выпускает широкий ассортимент демонстрационно-отладочных плат для ознакомления с микроконтроллерами серии 1986. Отметим лишь некоторые отладочные комплекты для 32-битных микроконтроллеров 1986ВЕ91Т(94Т), 1986ВЕ92У, а также для микроконтроллеров К1986ВЕ92QI, К1986ВЕ92QI, 1986ВЕ93У, 1986ВЕ1Т. В данном параграфе будет обсуждаться лишь демонстрационно-отладочная плата для микроконтроллеров 1986ВЕ92У. Более детальная информация о составе выпускаемых отладочных средств размещена на веб-сайте «ПКК Миландр»: `http://milandr.ru`.

2.2.1. Общая характеристика платы 1986EvBrd_64

Официальными источниками технической информации по демонстрационно-отладочной плате (Evalution board) 1986EvBrd_64 (далее – плата 1986EvBrd_64) являются ее техническое описание, представленное в документе [5], и принципиальная электрическая схема, представленная в [6]. Ниже мы приводим выдержки из описания [5] в объеме, необходимом для программирования. Фрагменты принципиальной электрической схемы [6] будут обсуждаться далее при анализе примеров программ.

Плата (стенд) 1986EvBrd_64 предназначена:

– для демонстрации функционирования и оценки производительности микроконтроллера 1986ВЕ92У и его основных периферийных модулей;

– демонстрации функционирования интерфейсных микросхем CAN и COM (RS-232) интерфейсов;

– отладки собственных проектов с применением установленных на плате блоков;

– программирования резидентной памяти программ микроконтроллеров 1986ВЕ92У.

В демонстрационном режиме плата 1986EvBrd_64 подключается:

– к COM порту персонального компьютера;

– CAN или COM (RS-232) интерфейсу дополнительного внешнего устройства, например аналогичной демонстрационно-отладочной плате 1986EvBrd_64;

– источнику питания +5 В.

Для программирования резидентной памяти программ микроконтроллеров 1986ВЕ92У применяется внешний внутрисхемный программатор ULINK2 (Keil) или JEM-ARM-V2(Phyton).

Питание 1986EvBrd_64 осуществляется от адаптера постоянного тока напряжением +5 В или от шины USB.

Комплектация демонстрационно-отладочной платы:

– печатная плата 1986EvBrd_64;

– образец микроконтроллера 1986ВЕ92У;

– нуль-модемный кабель для COM (RS-232) интерфейса;

– кабель USB-A/USB-B;

– блок питания для отладочной платы;

– диск с программным обеспечением, документацией, схемотехническими файлами и исходными кодами программ.

Дополнительно поставляются следующие продукты третьих фирм:

– USB JTAG адаптер J-LINK (Segger), предназначенный для работы с интегрированными средами разработки IAR Embedded Workbench и Keil uVision;

– ознакомительная 30-дневная полнофункциональная версия интегрированной среды разработки CodeMaster-ARM-TL (Phiton).

2.2.2. Компоновка платы 1986EvBrd_64

На рис. 2.4 показан внешний вид платы 1986EvBrd_64. Установленные на плату 1986EvBrd_64 компоненты показаны на рис. 2.5, а их описание содержится в табл. 2.2. При описании использован термин «BNC разъем», который обозначает коаксиальный радиочастотный разъём (СР) . BNC (*Bayonet Neill-Concelman*) — электрический разъем с байонетной фиксацией, служит для подключения коаксиального кабеля с волновым сопротивлением 50 или 75 Ом и диаметром до 8 мм.

На плате 1986EvBrd_64 установлены конфигурационные перемычки для управления выбором источников сигнала и настройкой скорости передачи данных CAN-интерфейса:

POWER_SEL – выбор источника питания для платы между разъемом USB и внешним источником питания;

SLEW RATE – выбор скорости передачи данных для интерфейса CAN;

CAN LOAD – выбор нагрузки линии CAN;

ADC_INP_SEL – выбор источника сигнала для 7-го канала АЦП между подстроечным резистором «TRIM» и BNC разъемом «ADC»;

COMP_INP_SEL – выбор источника сигнала на 1-м входе компаратора между BNC разъемом «COMP_INP» и выходом ЦАП2;

DAC_OUT_SEL – выбор назначения сигнала с выхода ЦАП2 между BNC разъемом «DAC_OUT» и звуковым усилителем.

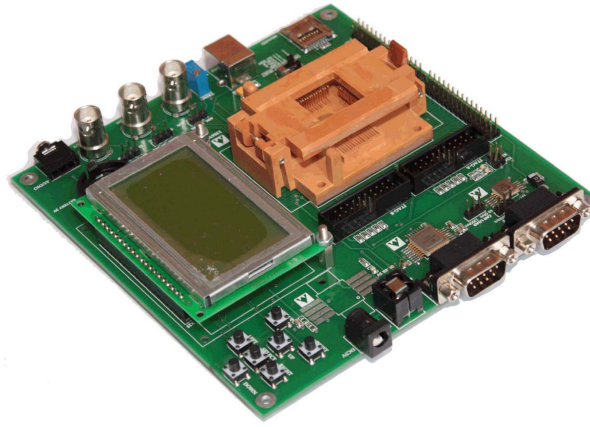


Рис. 2.4. Внешний вид демонстрационно-отладочной платы 1986EvBrd_64
(фотография взята из технического описания платы [5])

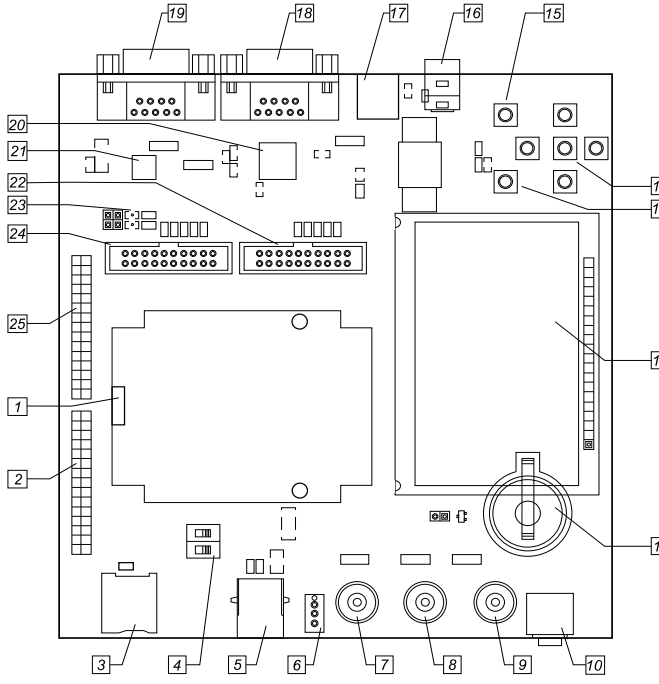


Рис. 2.5. Компоненты, установленные на плату 1986EvBrd_64

Таблица 2.2

Описание компонентов платы 1986EvBrd_64

Номер на рис. 2.5	Описание компонентов платы 1986EvBrd_64
1	Контактирующее устройство для микроконтроллера 1986ВЕ92У. Микроконтроллер должен быть установлен в спутник-держатель
2	Разъем X27 портов A, E, F микроконтроллера
3	Разъем карты памяти microSD
4	Переключатели выбора режима загрузки
5	Разъем USB-B
6	Подстроечный резистор на 7-м канале АЦП
7	Разъем BNC внешнего сигнала на 7-м канале АЦП
8	Разъем BNC внешнего сигнала на 1-м входе компаратора
9	Разъем BNC выхода ЦАП2
10	Разъем Audio 3.5 мм выхода ЦАП2 через звуковой усилитель
11	Батарея 3.0 В
12	Графический ЖК дисплей 128×64
13	Кнопка WAKEUP
14	Кнопки UP, DOWN, LEFT, RIGHT, SELECT
15	Кнопка RESET
16	Разъем питания 5 В
17	Фильтр питания
18	Разъем RS-232
19	Разъем CAN
20	Приемопередатчик RS-232 5559ИН4
21	Приемопередатчик CAN 5559ИН14
22	Разъем отладки JTAG-B
23	Набор светодиодов на порте C
24	Разъем отладки JTAG-A
25	Разъем X26 портов B, C, D микроконтроллера

Назначение установленных на плате переключателей и клавиш:

– SW1, SW2, SW3 — переключатели выбора режима загрузки (Boot Select). Назначение переключателей приведено в табл. 2.3;

– UP, DOWN, LEFT, RIGHT, SELECT — программируемые пользователем клавиши;

– RESET — сигнал аппаратного сброса микроконтроллера;

– WAKEUP — сигнал внешнего выхода из режима Standby.

Плата 1986EvBrd_64 содержит два 30-контактных разъема X26 и X27 (см. табл. 2.2, позиции *2* и *25*), на которые выведены линии всех портов микроконтроллера, общий провод (GND) и питание (+3.3 В, +5 В). Подключение портов микроконтроллера к разъемам X26 и X27 показано в табл. 2.4. Контакты каждого разъема сгруппированы в два ряда: четные и нечетные номера выводов.

Таблица 2.3

Назначение переключателей выбора режима загрузки

SW1	SW2	SW3	Режим работы
0	0	0	Режим микроконтроллера, код исполняется из флэш-памяти, начиная с адреса 0x0800_0000. Загрузка через разъем JTAG_B
0	0	1	Режим микроконтроллера, код исполняется из флэш-памяти, начиная с адреса 0x0800_0000. Загрузка через разъем JTAG_A
0	1	0	Режим микропроцессора, код исполняется из внешней памяти (EXT_ROM), начиная с адреса 0x1000_0000. Загрузка через разъем JTAG_B
0	1	1	Режим микропроцессора, код исполняется из внешней памяти (EXT_ROM), начиная с адреса 0x1000_0000. Загрузка через разъем JTAG_A
1	1	0	Режим микропроцессора, код исполняется из внешней памяти, начиная с адреса 0x1000_0000. Загрузка через UART2

Таблица 2.4

Подключение портов микроконтроллера к разъемам X26, X27

Контакт	Вывод МК/питание		Контакт	Вывод МК/питание	
	X26	X27		X26	X27
1	GND	GND	2	GND	GND
3	+3.3 В	+3.3 В	4	+3.3 В	+3.3 В
5	PD0	PA6	6	PD1	PA7
7	PD2	PA4	8	PD3	PA5
9	PD4	PA2	10	PD5	PA3
11	PD6	PA0	12	–	PA1
13	PB0	–	14	PB1	–
15	PB2	PE1	16	PB3	PE3
17	PB4	–	18	PB5	–
19	PB6	PF0	20	PB7	PF1
21	PB8	PF2	22	PB9	PF3
23	PB10	PF4	24	PC0	PF5
25	PC1	PF6	26	PC2	–
27	+5 В	+5 В	28	+5 В	+5 В
29	GND	GND	30	GND	GND

2.2.3. Интерфейс для подключения отладчика

Микроконтроллер снабжен встроенным интерфейсом JTAG, который можно использовать двояко: для занесения прошивки в резидентную память программ (флэш-память) и для отладки программы. Для практической работы с этим интерфейсом рекомендуется внешний программатор-отладчик MT-Link, который относится к классу эмуляторов J-Link. Уточним содержание новых терминов.

JTAG (сокращение от англ. *Joint Test Action Group*; произносится «джей-та́г») — название рабочей группы по разработке стандарта IEEE 1149. Позднее это сокращение стало прочно ассоциироваться с разработанным этой группой специализированным аппаратным интерфейсом на базе стандарта IEEE 1149. Официальное название стандарта **Standard Test Access Port and Boundary-Scan Architecture**. Интерфейс предназначен для подключения сложных цифровых микросхем или устройств уровня печатной платы к стандартной аппаратуре тестирования и отладки. JTAG-порт содержит пять линий.

На текущий момент интерфейс стал промышленным стандартом. Практически все сколько-нибудь сложные цифровые микросхемы оснащают этим интерфейсом:

– для выходного контроля микросхем при производстве;

– тестирования собранных печатных плат;

– прошивки микросхем с памятью;

– отладочных работ при проектировании аппаратуры и программного обеспечения.

Метод тестирования, реализованный в стандарте, получил название *Boundary Scan* (граничное сканирование). Название отражает первоначальную идею процесса: в микросхеме выделяются функциональные блоки, входы которых можно отсоединить от остальной схемы, подать заданные комбинации сигналов и оценить состояние выходов каждого блока. Весь процесс производится специальными командами по интерфейсу JTAG, при этом никакого физического вмешательства не требуется. Разработан стандартный язык управления данным процессом – *Boundary Scan Description Language* (BSDL).

SW (сокращение от англ. *Serial wire*). Порт SW содержит 2 линии. Serial Wire Debug (SWD) – отладчик в стандарте SW имеется в наличии как часть CoreSight Debug Access Port. Является альтернативой 5-pin JTAG интерфейсу.

J-Link – это JTAG эмулятор с питанием от шины USB, поддерживающий большое количество ядер CPU. Основанный на 32-разрядном RISC CPU, он может с высокой скоростью обмениваться данными со

всеми поддерживаемыми CPU. J-Link используется в десятках тысяч мест по всему миру для целей разработки и производств (программирования флэш-памяти). Поддержка J-Link интегрирована в большинство профессиональных IDE, таких как IAR, Keil, Rowley и многие другие. Наряду с OEM версиями (такими как IAR J-Link, ATMEL SAM-Ice и другие) были проданы более чем 60 000 экземпляров J-Link, что позволяет говорить о J-Link как наиболее популярном эмуляторе для ARM ядер и, де-факто, промышленном стандарте.

На рис. 2.6 показан внешний вид JTAG-эмулятора `MT-Link`. В комплект входят эмулятор `MT-Link` (плата, покрытая зеленым пластиком), шлейф FC 20P для подключения к порту JTAG и кабель USB-A/USB-B для подключения к стандартному USB-разъему компьютера.

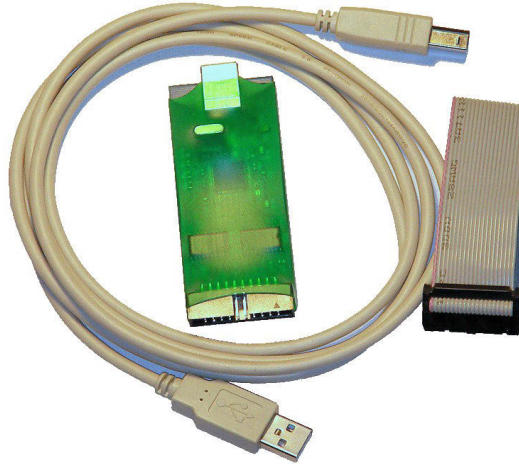


Рис. 2.6. Внешний вид JTAG-эмулятора `MT-Link` (аналог J-Link) в комплекте (фотография взята с сайта http://starter-kit.ru)

Микроконтроллер имеет два порта для подключения отладчика: JTAG-A и JTAG-B (см. табл. 2.2). Это сделано из-за того, что выводы для подключения отладчика совмещены с выводами различных устройств, встроенных в микроконтроллер. В зависимости от того, какие из них используются в том или ином проекте, выбирается тот вариант подключения отладчика, который позволит не занимать его разъёмом нужные для конструкции выводы микроконтроллера. Производители микроконтроллера рекомендуют подключать отладчик только к плате, на которую уже подано напряжение питания.

Если в программе настроить для работы на выход один или несколько выводов, используемых для подключения отладчика, его работа станет невозможной. В таком случае можно использовать другой разъём для отладчика, если он имеется в целевой схеме. Более серьезная проблема возникает, если перенастроить выводы, используемые для подключения сразу обоих отладчиков (JTAG_A и JTAG_B). Однако и в этом случае существуют опробованные решения по выводу микроконтроллера в нормальный режим работы [7].

2.3. Средства разработки

Для разработки программного обеспечения применяют различные средства разработки. Вот лишь некоторые из них, рекомендуемые разработчиками микроконтроллеров.

«CodeMaster-ARM» – интегрированная среда разработки компании Phyton. Поддерживает разработку программ для 1986ВЕ9xx. Укомплектована С-компилятором, ассемблером, отладчиком, средствами для трассировки и внутрисхемного программирования, а также поддерживает USB JTAG адаптер JEM-ARM-V2. Имеется ознакомительная 30-дневная полнофункциональная версия и бесплатная демо-версия с ограничением кода программы в 8 Кб.

«IAR Embedded Workbench» – интегрированная среда разработки компании IAR Systems. Поддерживает разработку программ для 1986ВЕ9xx. Укомплектована C/C++ компилятором, ассемблером, отладчиком, средствами для трассировки и внутрисхемного программирования, а также поддерживает USB JTAG адаптер J-Link.

«Keil uVision» – интегрированная среда разработки (IDE) компании Keil (An ARM Company). IDE Keil uVision поддерживает разработку программ для различных ARM-микроконтроллеров (версия uVision 3.0 и выше), в том числе для микроконтроллеров серии 1986 фирмы «Миландр» (версия uVision 4.2 и выше). Укомплектована C/C++ компилятором, ассемблером, отладчиком, средствами для трассировки и внутрисхемного программирования, а также поддерживает USB JTAG адаптеры J-Link и ULINK2. Имеется бесплатная полнофункциональная версия с ограничением кода программы в 32 Кб.

В дополнение к этим средствам можно использовать свободное программное обеспечение на основе компилятора GCC, среды Eclipse и им подобные сборки.

Мы рассмотрим лишь одно из рекомендованных средств разработки, а именно интегрированную среду разработки Keil uVision.

2.3.1. Интегрированная среда разработки Keil uVision

Инсталляция среды разработки. На момент написания данного материала на сайте фирмы Keil выложен дистрибутивный архив интегрированной среды разработки Keil uVision5 (версия 5.13, Декабрь 2014) в виде файла `mdk_513.exe` (346 Мб). Этот программный продукт имеет бесплатную лицензию вида Lite/Evaluation Edition, которая устанавливает следующие ограничения в сравнении с полнофункциональной коммерческой версией продукта. Для Lite/Evaluation версии объем исполняемого модуля в двоичных кодах не может превышать 32 Кб (ARM-платформа) и 2 Кб (x51-платформа). Есть еще некоторые менее существенные ограничения. Рассмотрим основные этапы процесса инсталляции интегрированной среды Keil uVision5.

1. Загрузить с сайта http://www.keil.com/download/product/ дистрибутивный архив в виде файла, например `mdk_513.exe`. Установить программу в Keil uVision5 в подходящую директорию. В нашем случае была выбрана директория `D:\Program Files\Keil`. Общий объем файлов в этой директории после установки интегрированной среды разработки Keil uVision5 составил около около 1.20 Гб.

2. Установить дополнительные архивы (Packs) от производителей оборудования. Эти архивы обеспечивают поддержку оборудования конкретных производителей. Архивы можно установить при инсталляции основной программы либо в любой другой момент времени, используя вкладку `Pack Installer`, которая расположена на панели `Toolbar`. Мы выбрали два дополнительных архива:

"`Milandr.MDR1986BExx.1.4.0.pack`" (2.41 Мб) – пакет от фирмы «Миландр» для работы с контроллерами MDR1986BE;

"`Keil.STM32F1xx_DFP.1.0.5.pack`" (48 Мб) – пакет для работы с контроллерами `STM32`.

3. Создать директорию "`D:\Program Files\Keil\ARM\INC`" для самостоятельного размещения `*.h` файлов, которые не входят в состав стандартных библиотек.

4. К разъемам внешнего программатора J-Link (плата, покрытая пластиком зеленого цвета) подключаем кабели:

– USB(B)-кабель, другой конец которого [USB(A)] подключаем к компьютеру. Светодиод под зеленым пластиком начинает быстро мигать, пока USB-драйверы J-Link не установлены. Следует отметить, что требуемые драйверы уже предустановлены инсталлятором Keil uVision5. После подключения USB-кабеля начнется их установка, и как только драйверы установятся, светодиод начнет светится ровным светом;

– FC 20P-шлейф, другой конец которого подключаем к разъему JTAG B на стенде. Жидкокристаллический дисплей начнет подсвечиваться в половину обычной яркости, получая питание от USB через шлейф.

Необходимо отметить, что если в резидентной флэш-памяти программ есть исполняемая программа, то она может начать выполняться в контроллере сразу после включения питания. При этом оба кабеля (USB-кабель и шлейф) должны быть подключены, как это описано выше, либо шлейф JTAG должен быть отключен от стенда. Причина заключается в том, что не подключенный к USB J-Link может препятствовать выполнению программы контроллером.

5. Все три переключателя [SW1, SW2, SW3] на стенде ставим в положение 0. Они выбирают источник для загрузки программ. Указанная комбинация (0 0 0) соответствует JTAG B.

6. Подключаем источник питания к стенду.

Создание нового проекта Keil. Программный проект в интегрированной среде разработки Keil uVision будет содержать множество файлов. Помимо файлов с программой пользователя там будут создаваться многочисленные служебные файлы. В связи с этим для каждого проекта необходимо сначала создать отдельную директорию. В качестве примера создадим директорию "Мой проект Keil\Project_LED". В этой директории разместим исходный модуль нашей программы – файл main.c. Создание проекта включает несколько этапов, каждый из которых будет начинаться с вкладки [Project]. В этой связи вкладку [Project] указывать не будем, а приведем только последующие вложенные вкладки.

1. Объявление нового проекта. Для этого внутри вкладки [Project] выбираем вкладку [New uVision Project]. После этого вводим имя проекта, например LED_1. В дальнейшем для указания последовательности действий будем использовать краткие обозначения без приведения вкладки [Project], в которых рассмотренный выше пример имеет вид [New uVision Project] – LED_1.

2. Выбор типа микроконтроллера из базы данных Keil uVision.
[Select Device for Target 'Target1'] –
– [Milandr/Milandr/Cortex-M3/MDR1986BE92].

3. Далее выбираем стартовый файл для микроконтроллера.
[Manage/Run-Time Environment] –
– [Device/Startup_MDR1986BE9x].

4. Выбираем файл, содержащий исходный текст нашей программы, например файл "main.c".

[Manage/Components, Environments, Books/
/Manage Project Items/Add/Files/Add] – main.c.

5. Указываем тактовую частоту микроконтроллера 8 МГц.

[Options for Target 'Target1'/Target/Xtal] – 8.0 MHz.

6. Назначаем пути поиска для файлов-заголовков (драйверы *.h).

[Options for Target 'Target1'/C/C++/Include Path].
В качестве примера укажем два маршрута. Во-первых, для *.h файлов стандартной библиотеки периферийных устройств:

"D:\Program Files\Keil\ARM\Pack\Keil\MDR1986BExx\
\1.4\Libraries\MDR32F9Qx_StdPeriph_Driver\inc".

Во-вторых, для дополнительных *.h файлов, не входящих в состав стандартной библиотеки периферийных устройств:

"D:\Program Files\Keil\ARM\INC".

Маршрутные имена набирают в двойных кавычках и разделяют символом ';',

7. Настройка внешнего программатора (JTAG-эмулятор) для загрузки программ пользователя производится на вкладке [Debug/]:

[Options for Target 'Target1'/Debug/]. Далее выполнить:

[/Use/J-LINK/J-TRACECortex];

[/Settings/Debug/Port/SW];

[/Settings/Debug/Connect/with Pre-reset];

[/Settings/Debug]: 0-Cashe Code; 0-Cashe Memory;
1-Verify Code Download; 1-Download to Flash; [OK];

[/Settings/Flash Download/Programming Algorithm/] –
– 1986BE IAP 128KB Flash.

[/Settings/Flash Download/]: 1 – Erase Full Chip;
1 – Reset and Run; [OK].

8. Компиляция программы [Rebuild all target Files].

9. Подготовка флэш-памяти программ [Flash\Erase].

10. Загрузка программы в микроконтроллер [Flash\Download].

2.3.2. Стандартная библиотека периферийных устройств

MDR32F9Qx_StdPeriph_Driver – стандартная библиотека ввода-вывода, созданная компанией «Фитон» на языке Си для микроконтроллеров семейства Cortex-M производства фирмы «Миландр». Содержит функции, структуры и макросы для облегчения работы с периферийными блоками микроконтроллеров. Библиотека поддерживает CMSIS (Cortex Microcontroller Software Interface Standard) и предоставляется компанией «Миландр» бесплатно. Основные файлы библиотеки распространяются в виде архива "`Standard Peripheral Library MDR32F9Qx, MDR1986VE1T, MDR1986VE3T.rar`". Файловый архив имеет объем около 3.4 Мб.

Библиотека хорошо документирована, содержит примеры по каждому периферийному устройству и имеет интерактивную справку [8] в виде отдельного файла в стандартном формате справки Windows "`MDR32F9Qx_Standard_Peripherals_Library.chm`" (9.2 Мб).

Необходимо отметить, что использование стандартной библиотеки ввода-вывода или стандарта CMSIS не является обязательным при разработке программ для микроконтроллеров семейства Cortex-M. Так, в книге [3] примеры программ написаны без использования таких библиотек, в них приводится традиционный подход системного программирования прямого обращения к нужным регистрам микроконтроллера. Создание CMSIS разработчиком процессора и создание библиотеки ввода-вывода производителем микроконтроллеров призвано лишь сократить затраты на разработку нового программного обеспечения, и наличие этих библиотек является ничем иным, как важным конкурентным преимуществом по сравнению со многими другими микроконтроллерными разработками.

Мы также будем стараться обходиться без использования стандартных библиотек ввода-вывода. В большинстве случаев достаточно лишь двух файлов: общего файла `MDR32Fx.h` с описанием ресурсов микроконтроллеров семейства Cortex-M и файла конфигурации `MDR32F9Qx_config.h` для настройки на конкретное периферийное оборудование. Подробное описание и спецификация данных файлов содержится в документе [4].

3. ПРОГРАММИРОВАНИЕ ПЛАТФОРМЫ CORTEX-M3

Приведены необходимые данные по программированию на языке C/C++ микроконтроллеров семейства Cortex-M3. Вначале рассмотрены общие вопросы программирования периферийных устройств на языке Си, а затем приведены примеры конкретных программ и разобраны типовые задачи, предлагаемые студентам при выполнении практических занятий.

3.1. Программирование периферийных устройств стенда

3.1.1. Светодиодные индикаторы

Микроконтроллер имеет 6 портов ввода/вывода по 16 разрядов каждый. Выводы портов могут быть переключены на различные функциональные блоки, возможна индивидуальная настройка и управление каждой битовой линией порта. На рис. 3.1 приведен фрагмент принципиальной схемы стенда, касающийся только светодиодных индикаторов.

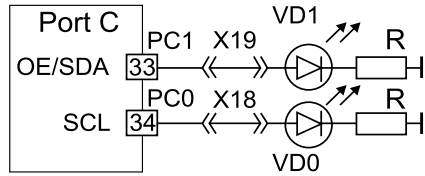


Рис. 3.1. Схема подключения светодиодных индикаторов

Для того чтобы выводы порта перешли под управление того или иного периферийного блока, необходимо задать для нужных выводов выполняемую функцию и настройки. Все адреса приведены с использованием условного обозначения `База->смещение`, в квадратных скобках указан диапазон номеров битов каждого порта. Рассмотрим несколько вариантов построения программы в порядке усложнения.

Программа 1. Вывести статическую информацию на светодиодные индикаторы. Решение заключается в выводе единичных значений для линий `PC0` и `PC1` порта `PORTC`. В программе содержится мини-

мальный набор параметров настройки, необходимых для вывода статической информации. Листинг программы приведен на рис. 3.2.

```
/*-------------------------------------------------------------*/
#include <MDR32F9Qx_config.h>          // Device Startup
#include <MDR32Fx.h>                   // Device Header
/*-------------------------------------------------------------*/
int main (void)
{
    MDR_RST_CLK->PER_CLOCK |=0x800000; // Clock for PortC
    MDR_PORTC->OE          |=0x0003;   // Output for both
    MDR_PORTC->ANALOG      |=0x0003;   // Analog for both
    MDR_PORTC->PWR         |=0x05;     // Slow front (100ns)
    MDR_PORTC->RXTX        |=0x03;     // Two light indicators
    while (1);                         // Infinite loop
}
/*-------------------------------------------------------------*/
```

Рис. 3.2. Программа 1 для вывода статической информации на светодиодные индикаторы

Рассмотрим спецификацию регистров настройки порта PORT C.

MDR_RST_CLK->PER_CLOCK [31:0], смещение 0x1C – регистр управления тактовой частотой периферийных блоков. Биты разрешают тактирование периферийных блоков (0 – запрещено, 1 – разрешено). Порту PORTC соответствует бит 23.

MDR_PORTC, базовый адрес 0x400B_8000 – порт PORTC.

MDR_PORTC->RXTX [15:0], смещение 0x0 – данные порта PORTC.

MDR_PORTC->OE [15:0], смещение 0x04 – направление потока данных порта (0 – ввод, 1 – вывод).

MDR_PORTC->ANALOG [15:0], смещение 0x0C – режим работы порта (0 – аналоговый, 1 – цифровой).

MDR_PORTC->PWR [31:0], смещение 0x18 – управление мощностью порта вывода: 00 – порт отключен;
01 – медленный фронт (100 нс);
10 – быстрый фронт (20 нс);
11 – короткий фронт (10 нс).

Проведем структурирование Программы 1, выделив начальную инициализацию порта в отдельную подпрограмму-функцию io_init. Одна из целей структурирования заключается в возможности последующего использования созданных подпрограмм-функций при разработке новых программ. Листинг программы приведен на рис. 3.3.

```
/*-------------------------------------------------------------*/
#include <MDR32F9Qx_config.h>              // Device Startup
#include <MDR32Fx.h>                       // Device Header
/*-------------------------------------------------------------*/
void io_init (void)
{
    MDR_RST_CLK->PER_CLOCK |=0x800000;
    MDR_PORTC->OE          |=0x0003;
    MDR_PORTC->ANALOG      |=0x0003;
    MDR_PORTC->PWR         |=0x05;
    MDR_PORTC->RXTX=0;             // Clear all data in Port C
}
int main (void)
{
    io_init();
    MDR_PORTC->RXTX        |=0x03;
    while (1);
}/*-------------------------------------------------------------*/
```

Рис. 3.3. Структурирование Программы 1 (версия 1)

Рекомендуется головную программу main() всегда располагать в начале файла, а подпрограммы-функции – вслед за ней. Для этого необходимо в начале файла поместить прототипы функций, т. е. декларировать типы данных, связанные с этими функциями. В дальнейшем будем использовать только такой способ декларирования функций, т. е. будем указывать их прототипы в начале программы перед головной функцией main(). Листинг программы приведен на рис. 3.4.

```
/*-------------------------------------------------------------*/
#include <MDR32F9Qx_config.h>      // Device Startup
#include <MDR32Fx.h>               // Device Header
void io_init (void);               // Declaration for io_init
/*-------------------------------------------------------------*/
int main (void)
{
    io_init();
    MDR_PORTC->RXTX        |=0x03;
    while (1);
}
/*-------------------------------------------------------------*/
void io_init (void)
{
    MDR_RST_CLK->PER_CLOCK |=0x800000;
    MDR_PORTC->OE          |=0x0003;
    MDR_PORTC->ANALOG      |=0x0003;
    MDR_PORTC->PWR         |=0x05;
    MDR_PORTC->RXTX=0;
}/*-------------------------------------------------------------*/
```

Рис. 3.4. Структурирование Программы 1 (версия 2)

Программа 2. Вывести динамическую информацию на оба светодиодных индикатора. На рис. 3.5 приведен фрагмент функциональной схемы контроллера тактовой частоты. Микроконтроллер имеет два встроенных тактовых генератора (HSI и LSI) и две линии для подключения внешнего тактирования (HSE и LSE), а также специализированный блок формирования частоты тактовой синхронизации. Оба встроенных генератора (HSI и LSI) автоматически запускаются при появлении питающего напряжения Ucc.

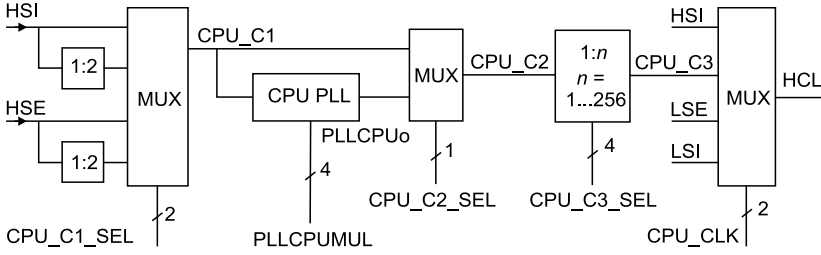


Рис. 3.5. Фрагмент функциональной схемы контроллера тактовой частоты

Встроенный RC-генератор HSI вырабатывает тактовую частоту 8 МГц, первоначально процессорное ядро всегда запускается на тактовой частоте HSI. При дальнейшей работе генератор HSI может быть отключен или подстроен.

Встроенный RC-генератор LSI вырабатывает тактовую частоту около 32 КГц, которая первоначально используется для формирования дополнительной задержки t_{por}. При дальнейшей работе генератор LSI может быть отключен.

Внешние генераторы HSE и LSE предназначены для выработки стабильной тактовой частоты от 2 до 16 МГц (HSE) и 32 КГц (LSE). Каждый из них может работать в двух режимах: внутренний тактовый генератор с внешним резонатором или внешний генератор, т. е. внешние сигналы тактирования, подаваемые на линию HSE или LSE.

В Программе 2 (рис. 3.6) мы не будем использовать умножитель частоты CPU PLL, а будем применять только внешний тактовый генератор HSE, работающий на тактовой частоте 8 МГц. Функция `io_init` здесь взята из Программы 1, она подробно обсуждалась ранее. В этой связи обсудим только настройки контроллера тактовой частоты в новой функции `frq_init`.

На рис. 3.7 приведена функция `frq_init_pll`, использующая PLL (умножитель частоты). Рассмотрим назначение регистров.

```
/*-------------------------------------------------------------*/
#include <MDR32F9Qx_config.h>              // Device Startup
#include <MDR32Fx.h>                       // Device Header
/*-------------------------------------------------------------*/
void frq_init(void);              // Declaration for frq_init
void io_init (void);              // Declaration for io_init
/*-------------------------------------------------------------*/
int main (void)
{
    unsigned TIME_LIGHT=0x9FFFF;
    unsigned int t=0;
    frq_init();
    io_init();
    while (1){
        for (t = 0; t < TIME_LIGHT; t++)
        MDR_PORTC->RXTX = 0x01;
        for (t = 0; t < TIME_LIGHT; t++)
        MDR_PORTC->RXTX =0x02;
    }
}
/*-------------------------------------------------------------*/
void frq_init(void)
{
  MDR_RST_CLK->HS_CONTROL = 0x1;      // Enable HSE oscillator
  /* wait while HSE startup    */
  while (MDR_RST_CLK->CLOCK_STATUS == 0x00) __NOP();
  MDR_RST_CLK->CPU_CLOCK = 0x102;     // switch to HSE (8 MHz)
  SystemCoreClockUpdate();
}
/*-------------------------------------------------------------*/
```

Рис. 3.6. Программа 2 для вывода динамической информации на индикаторы (без использования PLL)

```
/*-------------------------------------------------------------*/
void frq_init_pll(void)
{
    MDR_RST_CLK->HS_CONTROL = 0x1;
    while (MDR_RST_CLK->CLOCK_STATUS == 0x00) __NOP();
MDR_RST_CLK->PLL_CONTROL |=0x500;
    /* PLL=6: CPU CLK= 48 MHz */
    MDR_RST_CLK->PLL_CONTROL |=0x04;  // Enable PLL for CPU
    while (!(MDR_RST_CLK->CLOCK_STATUS & 0x02)) __NOP();
    MDR_RST_CLK->CPU_CLOCK = 0x176;   // switch to HSE (6 MHz)
    SystemCoreClockUpdate();
}
/*-------------------------------------------------------------*/
```

Рис. 3.7. Функция frq_init_pll, использующая PLL

MDR_RST_CLK, базовый адрес 0x4002_0000 – контроллер тактовой частоты;

MDR_RST_CLK->HS_CONTROL [31:0], смещение 0x08 – регистр для управления высокочастотным генератором и осциллятором. [31:2] – не используются; бит [1] – управление HSE осциллятором (0 – режим осциллятора, 1 – режим внешнего генератора); бит [0] – управление HSE осциллятором (0 – выключен, 1 – включен);

MDR_RST_CLK->CLOCK_STATUS [31:0], смещение 0x0 – регистр состояния блока управления тактовой частоты. Биты [31:3] – не используются; бит 2 – флаг выхода в рабочий режим осциллятора HSE (0 – осциллятор не запущен или не стабилен; 1 – осциллятор запущен и стабилен); бит 1 – флаг выхода в рабочий режим CPU PLL; бит 0 – флаг выхода в рабочий режим USB PLL;

MDR_RST_CLK->CPU_CLOCK [31:0], смещение 0x0C – регистр управления тактовой частотой процессорного ядра. Назначение битов:

[31:10] – не используются;

[9:8] (CPU_CLK) – выбор источника HCLK
(00 – HSI; 01 – CPU_C3; 10 – LSE; 11 – LSI);

[7:4] (CPU_C3_SEL) – выбор делителя для получения CPU_C3:
CPU_C3 = CPU_C2 / n (0xxx – n = 1; 1000 – n = 2; 1001 – n = 4; 1010 – n = 8 … 1111 – n = 256);

[3] – не используется. Рекомендуется записать нулевое значение;

[2] (CPU_C2_SEL) – выбор источника для CPU_C2
(0 – CPU_C1; 1 – PLLCPUo);

[1:0] (CPU_C1_SEL) – выбор источника для CPU_C1
(00 – HSI; 01 – HSI/2; 10 – HSE; 11 – HSE/2).

В Программе 2 был сделан следующий выбор для битов [9:0] регистра MDR_RST_CLK->CPU_CLOCK = 0x102 = 0100000010:

01 (CPU_C3) – 0000 (n = 1) – 0 (x) – 0 (CPU_C1) – 10 (HSE).

Информация о текущей тактовой частоте содержится в системной переменной SystemCoreClock, а обновление этой переменной осуществляет системная функция SystemCoreClockUpdate(). Рекомендуется все настройки тактовой частоты завершать вызовом системной функции SystemCoreClockUpdate(), чтобы всегда поддерживать в актуальном состоянии системную переменную SystemCoreClock.

Обсудим теперь спецификацию регистров управления умножителем частоты PLL CPU, которые были использованы в соответствующей версии функции frq_init (см. рис. 3.7):

MDR_RST_CLK->PLL_CONTROL [31:0], смещение 0x04 – регистр управления блоками умножения частоты. Назначение битов:

[31:12] – биты не используются;

[11:8] PLLCPUMUL – коэффициент умножения частоты для CPU:
PLLCPUUo = CPU_C3 × PLLCPUMUL + 1;
[7:4] PLLUSBMUL – коэффициент умножения частоты для USB;
[3] PLLCPU – бит перезапуска;
[2] PLLCPU – бит включения;
[1] PLLUSB – бит перезапуска;
[0] PLLUSB – бит включения.

Программа 3. Эта программа предназначена для вывода информации о состоянии клавишного джойстика на светодиодные индикаторы. Схема подключения клавиш джойстика представлена на рис. 3.8. Джойстик состоит из пяти клавиш, обозначенных UP (вверх), DOWN (вниз), LEFT (влево), RIGHT (вправо) и SEL (select, выбор). Для управления двумя светодиодными индикаторами программа будет использовать следующие клавиши: LEFT для включения светодиода VD1, RIGHT для включения светодиода VD0 и SEL для одновременного включения обоих индикаторов VD0 и VD1.

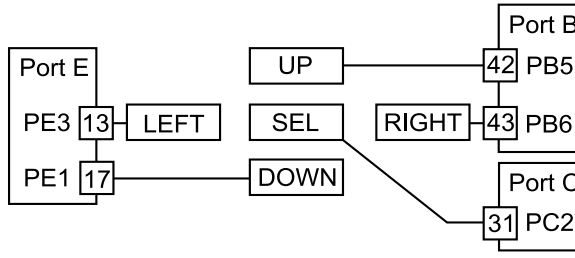


Рис. 3.8. Схема подключения клавишного джойстика

Головная программа для вывода состояний джойстика на светодиодные индикаторы показана на рис. 3.9. Для инициализации тактового генератора эта программа использует функцию frq_init_pll(), которая была описана ранее (см. рис. 3.7). Действие программы состоит в непрерывном сканировании линий портов, соответствующих трем клавишам джойстика LEFT, RIGHT и SEL. При опросе каждой линии производится проверка на нулевое значение уровня, поскольку исходное состояние каждой клавиши – уровень логической единицы. Процедуры включения и выключения светодиодных индикаторов оформлены в виде отдельных функций led_on() и led_off(). Параметром этих функций является номер светодиодного индикатора 0 или 1. Для определения констант, соответствующих линиям портов, использованы возможности препроцессора: команда `#define`

для определения базовых констант и операция "<<" для сдвига влево. Все остальные константы получены из базовых констант путем логических операций.

```
/*-------------------------------------------------------------*/
#include <MDR32F9Qx_config.h>              // Device Startup
#include <MDR32Fx.h>                       // Device Header
#define RIGHT (1<<6)                       // PB6
#define LEFT  (1<<3)                       // PE3
#define SEL   (1<<2)                       // PC2
#define VD1   (1<<1)                       // PC1
#define VD0   0x001                        // PC0
/*-------------------------------------------------------------*/
void frq_init_pll(void);      // Declaration for frq_init_pll
void joystick_init(void);     // Declaration for joystick_init
void led_on(unsigned short);  // Declaration for led_on
void led_off(unsigned short); // Declaration for led_off
/*-------------------------------------------------------------*/
int main (void)
{
    frq_init_pll();
    joystick_init();
    while (1){
      if (!(MDR_PORTB->RXTX & RIGHT))                 // RIGHT
            led_on(0);
      else  led_off(0);
      if (!(MDR_PORTC->RXTX & SEL)) {                 // SEL
        led_on(0);
        led_on(1);
      }
      if (!(MDR_PORTE->RXTX & LEFT))
            led_on(1);                                // LEFT
      else  led_off(1);
    }
}
/*-------------------------------------------------------------*/
void led_on(unsigned short num)
{
   if(num == 0) MDR_PORTC->RXTX |= VD0;
   else         MDR_PORTC->RXTX |= VD1;
}
void led_off(unsigned short num)
{
   if(num == 0) MDR_PORTC->RXTX &= ~VD0;
   else         MDR_PORTC->RXTX &= ~VD1;
}
/*-------------------------------------------------------------*/
```

Рис. 3.9. Программа 3 для вывода состояний клавиш джойстика на светодиодные индикаторы

Подпрограмма-функция `joystick_init` для начальной инициализации линий трех портов, которые используются для считывания информации с клавиш джойстика, приведена на рис. 3.10. Инициализация линий каждого порта здесь выполнена так же, как и в рассмотренной ранее функции `io_init`. Все линии портов, которые мы обсуждали ранее (см. рис. 3.8), программируются на ввод в аналоговом режиме с медленными фронтами (до 100 нс). Включено тактирование всех трех портов B, C, E. Особенностью является одновременное использование двух линий порта PORT C для вывода информации на светодиодные индикаторы. Поэтому линии PC0 и PC1 запрограммированы на вывод информации аналогично тому, как это было сделано в функции `io_init`.

```
/*-------------------------------------------------------------*/
void joystick_init (void)
{
   MDR_RST_CLK->PER_CLOCK |=(0x0B<<22);       // PORTs: B, C, E
/* PORT C */
   MDR_PORTC->OE          &= ~SEL;            // SEL (PC2)
   MDR_PORTC->OE          |=                  0x003;
   MDR_PORTC->ANALOG      |=  (SEL          | 0x003);
   MDR_PORTC->PWR         &= ~(((2+1)<<SEL) | 0x00A);
   MDR_PORTC->PWR         |=  (( 2   <<SEL) | 0x005);
   MDR_PORTC->RXTX=0;
/* PORT B */
   MDR_PORTB->OE          &= ~RIGHT;          // RIGHT (PB6)
   MDR_PORTB->ANALOG      |=  RIGHT;
   MDR_PORTB->PWR         &= ~((6+1)<<RIGHT);
   MDR_PORTB->PWR         |=   (6   <<RIGHT);
   MDR_PORTB->RXTX=0;
/* PORT E */
   MDR_PORTE->OE          &= ~LEFT;           // LEFT (PE3)
   MDR_PORTE->ANALOG      |=  LEFT;
   MDR_PORTB->PWR         &= ~((3+1)<<LEFT);
   MDR_PORTB->PWR         |=   (3   <<LEFT);
   MDR_PORTE->RXTX=0;
}/*-------------------------------------------------------------*/
```

Рис. 3.10. Подпрограмма-функция joystick_init для начальной инициализации клавиш джойстика

3.1.2. Графический дисплей

Вывод информации на графический дисплей (LCD) в отладочных платах фирмы «Миландр» осуществляется при помощи жидкокристаллического (ЖК) дисплея MT-12864J-2YLG-3V0 (производство

компании «МЭЛТ», г. Москва). Контроллер управления графического дисплея К145ВГ10 (производство ОАО «Ангстрем», г. Зеленоград) аналогичен контроллеру KS0108 фирмы SAMSUNG. Полная техническая информация на модуль графического дисплея MT-12864J содержится в спецификации [9]. Ниже приведены назначения программируемых выводов дисплея (табл. 3.1) и их спецификация при обращении к графическому модулю (табл. 3.2).

Таблица 3.1

Программируемые внешние выводы модуля LCD MT-12864j

LCD MT-12864j			Микроконтроллер	
Номер вывода	Обозначение	Назначение вывода	Линия порта	Номер вывода
4	DB0	Шина данных DB0	PA0	55
5	DB1	Шина данных DB1	PA1	54
6	DB2	Шина данных DB2	PA2	53
7	DB3	Шина данных DB3	PA3	52
8	DB4	Шина данных DB4	PA4	51
9	DB5	Шина данных DB5	PA5	50
10	DB6	Шина данных DB6	PF2	60
11	DB7	Шина данных DB7	PF3	61
12	E1	Выбор кристалла 1	PB7	44
13	E2	Выбор кристалла 2	PB8	45
14	RES	Сброс (начальная установка)	PB9	46
15	R/W	Выбор: Чтение/Запись	PB10	47
16	A0	Выбор: Команды/Данные	PC0	34
17	E	Стробирование данных	PC1	33

Таблица 3.2

Описание команд LCD MT-12864j

Команда	A0	R/W	Шина данных DB								Описание
			7	6	5	4	3	2	1	0	
On/Off Power	0	0	0	0	1	1	1	1	1	0/1	1 – включить питание; 0 – выключить питание
Start line	0	0	1	1	(0 – 63)						Верхняя строка LCD
Set Page	0	0	1	0	1	1	1	(0 – 7)			Установить страницу
Set Address	0	0	0	1	(0 – 63)						Установить адрес ОЗУ
Status	0	1	F1	0	F2	F3	0	0	0	0	Флаги состояния*
Write	1	0	Данные для вывода								Вывод данных
Read	1	1	Данные для ввода								Ввод данных

*Назначение флагов: F1–BUSY; F2–On/Off; F3–Reset.

Графический модуль дисплея содержит два независимых контроллера, поскольку видимая область (отображаемое поле точек) графического дисплея (128×64 точек) состоит из двух областей (64×64 точки). Каждая область дисплея работает под управлением своего контроллера. Эти контроллеры при программировании обозначают терминами «Кристалл 1» и «Кристалл 2» (Chip 1 и Chip 2). Для хранения данных, выводимых на ЖК-дисплей, модуль содержит ОЗУ размером 1024 байта: 64×64×2 бит (по 64×64 бит на каждый кристалл). Для выбора нужного кристалла используются выводы E1, E2. ОЗУ каждого кристалла разбито на 8 страниц (Page 0 – 8) размером по 64×8 бит каждая. Каждой светящейся точке на ЖК-дисплее соответствует логическая "1" в ячейке ОЗУ модуля.

Для начальной установки модуля необходимо подать сигнал RES (логический "0") длительностью не менее 1 мкс. После деактивации сигнала RES (переключение в логическую "1" с временем фронта не более 200 нс) необходимо выдержать паузу не менее 10 мкс.

При программировании каждая страница представлена в виде матрицы 64×8 бита. Запись информации в модуль осуществляется по страницам (64 байта данных) (рис. 3.11). Для записи байта данных по произвольному адресу необходимо предварительно установить страницу ОЗУ и установить адрес внутри страницы ОЗУ. Это осуществляется командами Set Page и Set Address соответственно. После этого можно записать байт данных. Запись каждого байта должна сопровождаться подачей строба (переключение в логическую "1" с временем фронта не более 200 нс) на вход E. Модуль поддерживает непрерывную последовательность операций записи: после записи одного байта счетчик адреса автоматически увеличивается на 1 и модуль готов к новой операции записи по следующему адресу без предварительной установки страницы ОЗУ и адреса. Счетчик адреса считает только внутри одной страницы. При достижении адреса 63 следующим значением счетчика будет 0 и т. д. Между любыми двумя передачами данных или команд необходимо выдержать паузу не менее 8 мкс или ожидать сброса флага BUSY в регистре состояния того кристалла, к которому будет обращение. Возможна скорость потока данных до 100–130 тыс. байтов / с.

Графический модуль может также работать в режиме чтения ОЗУ. Линия R/W служит для переключения режима чтение/запись.

Линия A0 служит для переключения между режимами «Команды» (A0 = 0) или «Данные» (A0 = 1) (см. табл. 3.2).

Рутинные процедуры для работы с графическим модулем дисплея собраны в библиотеку, которая представлена в виде отдельного файла `lcd_mlt.lib` (13.3 Кб). Для организации взаимодействия с графиче-

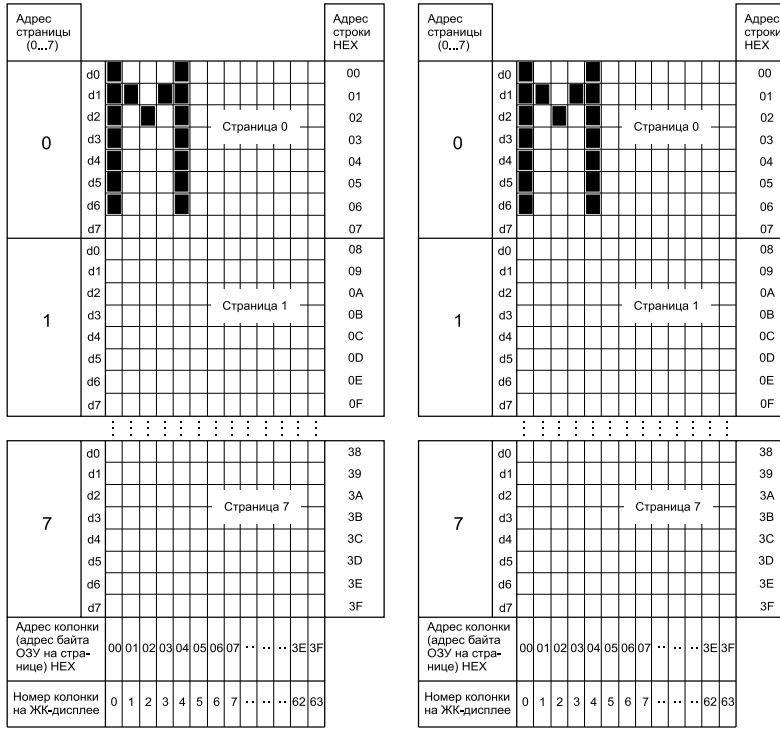


Рис. 3.11. Соответствие между адресами ОЗУ модуля и отображаемыми точками на ЖК-дисплее: *а* – левая половина отображаемого поля точек (кристалл 1, E1 = 1); *б* – правая половина отображаемого поля точек (кристалл 2, E2 = 1)

ским ЖК-модулем данную библиотеку следует подключить к проекту посредством команд [`Project/Manage/Components, Environment, Books.../Project items/Files/Add Files`].

Прототипы всех библиотечных функций собраны в отдельный файл `lcd_mlt.h`, который необходимо включить в программу посредством оператора #include. Рассмотрим несколько программ, которые используют библиотечные функции для работы с графическим модулем ЖК-дисплея.

Программа 4. Вывод графической информации на ЖК-дисплей организован в виде программы (рис. 3.12). Библиотека `lcd_mlt.lib` была подключена к проекту с помощью команд [`Project/Manage`...].

```
/*-------------------------------------------------------------*/
#include <MDR32F9Qx_config.h>         // Device Startup
#include <MDR32Fx.h>                  // Device Header
#include <.\mlt\lcd_mlt.h>            // LCD module functions
#include <.\mlt\logo_mlt.h>           // Picture for output
/*-------------------------------------------------------------*/
void frq_init(void);
/*-------------------------------------------------------------*/
int main (void)
{
   frq_init();
   MltPinCfg();                    //Pins configuration for LCD
   LcdInit();                            //LCD module init
   LcdClearChip(1);                      //Clear chip 1
   LcdClearChip(2);                      //Clear chip 2
   LcdPutImage(*LogoMlt[0],0,0,7,15);    //Draw picture for LCD
   while(1) ;
}
/*-------------------------------------------------------------*/
```

Рис. 3.12. Программа для вывода изображения (128×64 точки) на ЖК-дисплей

Секция include содержит типовые файлы (`MDR32F9Qx_config.h` и `MDR32Fx.h`) и два новых файла для работы с ЖК-дисплеем. Файл `mlt_lcd_mlt.h` содержит прототипы для всех функций библиотеки `lcd_mlt.lib`. Файл `logo_mlt.h` представляет собой заранее подготовленное изображение (128×64 точки) для вывода на ЖК-дисплей. Формат изображения полностью соответствует представлению информации в ОЗУ графического модуля (см. рис. 3.11) – 8 групп по 128 байтов. В секции main расположены пять подпрограмм-функций:

`frq_init()` – подключает HSE-генератор (внешний кварцевый резонатор и внутренние электронные схемы) и устанавливает частоту тактирования 8 МГц без использования PLL. Эта программа была использована в предыдущих примерах, и ее следует включить в состав исходного модуля программы. Исходный текст подпрограммы-функции `frq_init()` был приведен ранее (см. рис. 3.6).

`MltPinCfg()` – библиотечная функция, конфигурирует линии портов A, B, C и F микроконтроллера для работы с графическим модулем ЖК-дисплея в соответствии с табл. 3.1.

`LcdInit()` – библиотечная функция, конфигурирует графический модуль ЖК-дисплея и выполняет его инициализацию. Процедура инициализации состоит из формирования сигнала «Сброс» на линии Res и включения обоих кристаллов при помощи библиотечной функции `DispOn()`. Перед выполнением каждого действия считывают флаги состояния графического модуля (функция `ReadStatus(Chip)`)

и ожидают готовность ЖК-дисплея к выполнению очередного действия. Аргумент функции (uint8_t Chip, однобайтовое целое число без знака) может принимать одно из двух значений, соответствующих номеру кристалла (1 или 2). После выполнения этой процедуры графический модуль ЖК-дисплея готов к работе.

`LcdClearChip(uint8\_t Chip)` – библиотечная функция для очистки области ОЗУ, соответствующей отображаемому полю точек 64×64. Процедура выполняется отдельно для каждого кристалла.

`LcdPutImage(uint8\_t* array, int Ypos1, int Xpos1, int Ypos2, int Xpos2)` – библиотечная функция для вывода графического файла на экран ЖК-дисплея.

Первый аргумент функции представляет собой указатель на массив array, содержащий графическую информацию для вывода. Название массива может быть другим. Пример организации такого массива приведен в файле logo_mlt.h. Основу массива составляют 8 групп из 128 байтов. Каждая группа соответствует одной странице, каждый байт содержит информацию для восьми отображаемых точек. Общее описание графического массива 128×64 точки имеет вид

uint8_t LogoMlt[8][8][128]

Четыре последующих аргумента задают две точки, определяющие левый верхний и правый нижний углы прямоугольной области для отображения точек на экране дисплея. Координаты точек заданы в единицах групп из 8 битов таким образом, что координаты по вертикали (Ypos1 и Ypos2) могут изменяться от 0 до 7 и соответствуют номерам страниц. Координаты по горизонтали (Xpos) могут изменяться от 0 до 15.

Программа 5. Вывод одиночных символов в произвольную часть отображаемой области точек. На рис. 3.13 представлена программа, реализующая вывод одиночных символов. Вывод каждого символа производится путем вызова библиотечной функции LcdPutChar():

```
LcdPutChar (uint8_t* array, int Xpos, int Ypos).
```

Это библиотечная функция для вывода графического образа символа. Первый аргумент представляет собой указатель на массив точек, соответствующих образу символа. Графические образы символов подготовлены в файле font.h. Каждому символу (8×8 точек) соответствует группа из 8 байтов в виде массива `uint8_t symbol[8]`. Второй и третий аргументы функции задают местоположение выводимого символа. Численные значения этих аргументов могут изменяться от 0 до 7 (Ypos) и от 0 до 15 (Xpos).

```
/*-------------------------------------------------------------*/
#include <MDR32F9Qx_config.h>         // Device Startup
#include <MDR32Fx.h>                  // Device Header
#include <.\mlt\lcd_mlt.h>            // LCD module functions
#include <.\mlt\font.h>               // Fonts
/*-------------------------------------------------------------*/
void frq_init(void);
/*-------------------------------------------------------------*/
int main (void)
{
   frq_init();
   MltPinCfg();                    //Pins configuration for LCD
   LcdInit();                            //LCD module init
   LcdClearChip(1);                      //Clear chip 1
   LcdClearChip(2);                      //Clear chip 2
   LcdPutChar (cyr_F, 5, 3);
   LcdPutChar (cyr_I, 6, 3);
   LcdPutChar (cyr_Z, 7, 3);
   LcdPutChar (cyr_T, 8, 3);
   LcdPutChar (cyr_E, 9, 3);
   LcdPutChar (cyr_KH, 10, 3);
while(1) ;
}
/*-------------------------------------------------------------*/
```

Рис. 3.13. Программа для вывода одиночных символов на ЖК-дисплей

Программа 6. Вывод строки символов осуществляется при помощи функции `LcdPutString (uint8_t** array, int Ypos)`.

Первый аргумент представляет собой указатель на строку символов. Второй аргумент задает местоположение строки по вертикали. Подпрограмма-функция будет пытаться вывести ровно 16 символов, чтобы заполнить одну страницу ОЗУ графического модуля. Пример программы приведен на рис. 3.14. Строка символов задана в формате файла font.h.

Программа 7. Вывод строки символов в ASCII кодировке. Эта программа читает системную переменную SystemCoreClock; осуществляет форматированный вывод информации о текущей тактовой частоте микроконтроллера в строку ch_str (ASCII кодировка); при помощи подпрограммы-функции `ascii()` производит сопоставление каждого ASCII-символа строки с графическим образом этого символа в файле font.h; при помощи подпрограммы-функции LcdPutString() выводит графический образ строки на экран.

Листинг программы приведен на рис. 3.15. Обсудим детали программы, которых не было в предыдущих примерах. Секция include содержит дополнительный файл `stdio.h`. В нем размещены прото-

```
/*-------------------------------------------------------------*/
#include <MDR32F9Qx_config.h>         // Device Startup
#include <MDR32Fx.h>                  // Device Header
#include <.\mlt\lcd_mlt.h>            // LCD module functions
#include <.\mlt\font.h>               // Fonts
/*-------------------------------------------------------------*/
void frq_init(void);
/*-------------------------------------------------------------*/
int main (void)
{
uint8_t *array[16] = {sym_sp,sym_sp,sym_sp,sym_sp,sym_sp,\
                        cyr_F,cyr_I,cyr_Z,cyr_T,cyr_E,cyr_KH,\
                      sym_sp,sym_sp,sym_sp,sym_sp,sym_sp};
   frq_init();
   MltPinCfg();                     //Pins configuration for LCD
   LcdInit();                             //LCD module init
   LcdClearChip(1);                       //Clear chip 1
   LcdClearChip(2);                       //Clear chip 2
   LcdPutString (array, 2);
while(1) ;
}
/*-------------------------------------------------------------*/
```

Рис. 3.14. Программа для вывода строки символов на ЖК-дисплей

типы стандартных функций ввода-вывода, некоторые из них мы будем использовать ниже по тексту. Секция прототипов функций дополнена декларацией функции `ascii()`. Длина строки для вывода задана константой `STR_LENGTH`. В секции main объявлены два массива одинаковой длины. Массив `ch_str` служит для размещения строки вывода, символы которой представлены в кодировке ASCII. Этот массив заполняется при помощи стандартной функции `sprintf()`. Массив `array` служит для размещения указателей на графические образы этих символов в файле font.h. Каждый элемент этого массива заполняется при помощи функции `ascii()` в цикле по индексу "*i*".

Функция `ascii()` выполняет роль знакогенератора. Она принимает ASCII-символ в качестве входного аргумента и возвращает указатель на графический образ этого символа в файле font.h. Основу функции `ascii()` составляет оператор switch, осуществляющий многовариантный выбор. Для минимизации объема программы он содержит только символы, используемые в данной программе. Однако его структура достаточно простая, и ее легко можно дополнить другими символами, графические образы которых содержатся в файле font.h.

```
/*-------------------------------------------------------------*/
#include <MDR32F9Qx_config.h>         // Device Startup
#include <MDR32Fx.h>                  // Device Header
#include <.\mlt\lcd_mlt.h>            // LCD module functions
#include <.\mlt\font.h>               // Fonts
#include <stdio.h>
#define STR_LENGTH 16          //Length of the string in bytes
/*-------------------------------------------------------------*/
void frq_init_pll(void);
uint8_t * ascii(char symbol);
/*-------------------------------------------------------------*/
int main (void)
{
   char ch_str[STR_LENGTH];
   uint8_t *array[STR_LENGTH], i;
   frq_init_pll();
   MltPinCfg();                    //Pins configuration for LCD
   LcdInit();                                 //LCD module init
   LcdClearChip(1); LcdClearChip(2);          //Clear chip 1 & 2
   sprintf(ch_str,"    %u %c%c%c",SystemCoreClock/1000,\
                                                 'k','H','z');
   for(i = 0;i < STR_LENGTH;i++) {array[i] = ascii(ch_str[i]);}
   LcdPutString(array, 3);
   while(1) ;
}
/*-------------------------------------------------------------*/
uint8_t* ascii(char symbol)
{
   uint8_t* ch_ptr;
   switch (symbol){
         case '0': ch_ptr=dig_0; break;
         case '1': ch_ptr=dig_1; break;
         case '2': ch_ptr=dig_2; break;
         case '3': ch_ptr=dig_3; break;
         case '4': ch_ptr=dig_4; break;
         case '5': ch_ptr=dig_5; break;
         case '6': ch_ptr=dig_6; break;
         case '7': ch_ptr=dig_7; break;
         case '8': ch_ptr=dig_8; break;
         case '9': ch_ptr=dig_9; break;
         case 'H': ch_ptr=lat_H; break;
         case 'V': ch_ptr=lat_V; break;
         case 'k': ch_ptr=lat_k; break;
         case 'z': ch_ptr=lat_z; break;
         case '.': ch_ptr=sym_pt;break;
         default:  ch_ptr=sym_sp;break;
   }
   return(ch_ptr);
}
/*-------------------------------------------------------------*/
```

Рис. 3.15. Программа для вывода строки символов в ASCII кодировке

3.1.3. Цифроаналоговый преобразователь

Для вывода сигналов звукового диапазона используется линия PE0, которая может быть задействована не только как линия вывода порта PORTE, но и как выход встроенного цифроаналогового преобразователя (ЦАП). На рис. 3.16 приведена схема подключения аудиовыхода к линии выхода PE0 микроконтроллера. В зависимости от по-

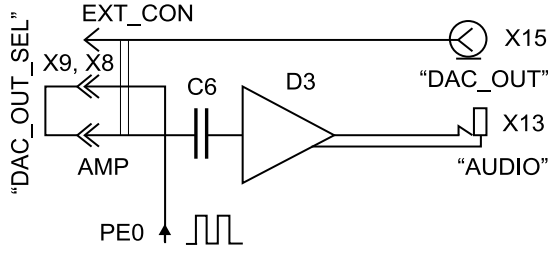


Рис. 3.16. Схема подключения аудиовыхода

ложения перемычки X9, X8 (DAC_OUT_SEL) линия PE0 может быть подключена к выходному разъему X15 (DAC_OUT) или ко входу усилителя звука D3. В первом случае можно вывести сигнал на внешний осциллограф и анализировать форму импульсов. Во втором случае можно использовать разъем AUDIO для подключения звуковоспроизводящей аппаратуры. Следует отметить, что звуковой усилитель ограничивает амплитуду сигнала на уровне около 0.7 В, поэтому он не может быть использован с цифровым сигналом.

Программа 8. Получение звука без использования цифроаналогового преобразователя (ЦАП). В программе 8 мы будем использовать линию PE0 в режиме порта вывода. Звуковой сигнал в цифровом виде будет сформирован полностью программно. Листинг программы приведен на рис. 3.17.

В секции прототипов описаны три функции: `frq_init()` – обеспечивает тактирование частотой 8 МГц с использованием внешнего кварцевого резонатора, листинг этой функции был приведен ранее (см. рис. 3.6); `sound_init()` – конфигурирует линии порта PORTE; `SoundDelay` формирует задержку `DELTA` на половину периода звуковых колебаний. Период колебаний задан константой `SOUND_DELAY`. Значение этой константы подобрано экспериментально для получения частоты колебаний около 440 Гц (период $T = 2.3$ мс). Программа генерирует меандр путем инверсного переключения состояния бита P0E через временной интервал $T/2$.

```
/*-------------------------------------------------------------*/
#include <MDR32F9Qx_config.h>                // Device Startup
#include <MDR32Fx.h>                         // Device Header
/*-------------------------------------------------------------*/
void frq_init(void);
void sound_init (void);
void SoundDelay (uint32_t value);
#define SOUND_DELAY 2600                     //440 Hz meandr
#define DELTA SOUND_DELAY/2
/*-------------------------------------------------------------*/
void main (void)
{
   frq_init();                               // Core CLK=8 MHz
   sound_init();
   while(1) {
     MDR_PORTE->RXTX ^= 0x01;                // Port_Bit_PE0
     SoundDelay(DELTA);
   }
}
/*-------------------------------------------------------------*/
void sound_init (void)
{
   MDR_RST_CLK->PER_CLOCK |=(0x01<<25);       // PORT: E
   MDR_PORTE->OE          |= 0x01;            // PE0
   MDR_PORTE->ANALOG      |= 0x01;
   MDR_PORTE->FUNC        &= 0x00;
   MDR_PORTE->PWR         &= 0xFFFD;
   MDR_PORTE->PWR         |= 0x01;
   MDR_PORTE->RXTX        |= 0x01;
}
/*-------------------------------------------------------------*/
void SoundDelay (uint32_t value)
{
while ((value--)!=0) {
      __NOP(); __NOP();
    }
}
/*-------------------------------------------------------------*/
```

Рис. 3.17. Программа для вывода звукового сигнала без использования ЦАП

Программа 9. Получение звука с использованием цифроаналогового преобразователя. В данной программе осуществляется цифровой синтез синусоидальных колебаний частотой около 440 Гц с помощью встроенного ЦАП. В программе 9 звуковой сигнал будет сформирован посредством ЦАП, поэтому мы будем использовать линию PE0 в режиме выхода ЦАП. Листинг программы приведен на рис. 3.18. Рассмотрим более детально организацию этой программы.

```
/*-------------------------------------------------------------*/
#include <MDR32F9Qx_config.h>                // Device Startup
#include <MDR32Fx.h>                         // Device Header
#include <math.h>                            // Device Header
#define SOUND_DELAY 2000                     //440 Hz sinus
#define DIGITIZER 128
#define AMPLITUDE 400
#define DELTA SOUND_DELAY/DIGITIZER
#define DELTA_RAD 2*3.141592/DIGITIZER
/*-------------------------------------------------------------*/
void frq_init(void);
void SoundDelay (uint32_t value);
void sound_init_dac (void);
void signal_sin(uint16_t *signal);
/*-------------------------------------------------------------*/
int main (void)
{
   uint8_t i;
   uint16_t signal[DIGITIZER];
   frq_init();                                // Core CLK=8 MHz
   sound_init_dac();
   MDR_DAC->CFG = 0x08;                            //DAC2 -- on
   while(1) {
     for(i=0;i<DIGITIZER;i++){
        MDR_DAC->DAC2_DATA = signal[i];
        SoundDelay(DELTA);
     }
   }
}
/*-------------------------------------------------------------*/
void signal_sin(uint16_t* signal)
{
   uint8_t i;
   for(i=0;i<DIGITIZER;i++){
      signal[i] = (uint16_t) (AMPLITUDE*(1 + sin(i*DELTA_RAD)));
   }
}
/*-------------------------------------------------------------*/
void sound_init_dac (void)
{                                              // PORT: E, DAC
   MDR_RST_CLK->PER_CLOCK |=((0x01<<25) | (0x01<<18));
   MDR_PORTE->OE          |= 0x01;             // PE0 - output
   MDR_PORTE->ANALOG      &= ~0x01;            // Analog output
}
/*-------------------------------------------------------------*/
```

Рис. 3.18. Программа для вывода звукового сигнала с использованием ЦАП

Секция Include содержит не только стадартные файлы описания аппаратуры MDR32F*.h, но также дополнительный файл `math.h`, в котором описаны прототипы математических функций.

В секции Define определены константы, необходимые для цифрового синтеза синусоидального сигнала: SOUND_DELAY задает период повторения синусоидального сигнала; DIGITIZER – количество временных каналов в расчете на один период синусоиды, т. е. интервалов времени, на которые разбит один период синусоиды; DELTA – цена одного канала для шкалы времени; DELTA_RAD – цена одного канала для шкалы в радианах. Значение константы SOUND_DELAY подобрано экспериментально для получения периода $T = 2.3$ мс (частота 440 Гц). Константа AMPLITUDE также подобрана экспериментально для получения амплитуды выходного синусоидального сигнала 0.6 В, поскольку усилитель ограничивает аудиосигнал на уровне 0.7 В.

В секции прототипов функций декларированы четыре функции. Функции frq_init() и SoundDelay() полностью соответствуют предыдущему примеру. Функция sound_init_dac() осуществляет следующие настройки: включение тактирования порта PORTE и ЦАП; конфигурирование линии PE0 для работы на вывод в аналоговом режиме; signal_sin() выполняет расчет одного периода синусоиды для последующего вывода через ЦАП.

Рассмотрим спецификацию регистров настройки ЦАП:

MDR_DAC, базовый адрес 0x4009_0000 – контроллер ЦАП.
MDR_DAC->CFG [31:0], смещение 0x00 – регистр управления ЦАП.

Назначение битов:

[31:5] –не используются;
[4] – синхронизация DAC1 и DAC2 (0 – асинхронные,
1 – синхронные);
[3] – включение DAC2 (0 – выключен, 1 – включен);
[2] – включение DAC1 (0 – выключен, 1 – включен);
[1] – выбор источника опорного напряжения DAC2 (0 – AUcc,
1 – в качестве источника используется линия DAC2_REF);
[0] – выбор источника опорного напряжения DAC1 (0 – AUcc,
1 – в качестве источника используется линия DAC1_REF);

MDR_DAC->DAC1_DATA [31:0], смещение 0x04 – регистр данных.

Назначение битов:
[31:28] – не используются;
[27:16] – данные DAC2 при работе в синхронном режиме (W);
[15:12] – не используются;
[11:0] – данные DAC1.

MDR_DAC->DAC2_DATA [31:0], смещение 0x08 – регистр данных.

Назначение битов:

[31:28] – не используются;

[27:16] – данные DAC1 при работе в синхронном режиме (W);
[15:12] – не используются;
[11:0] – данные DAC2.

Отметим, что не все микроконтроллеры серии `1986BExx` имеют полный набор внешних выводов, соответствующих спецификации [4]. В частности, микроконтроллер `1986BE92У` имеет внешние выводы, соответствующие только одному ЦАП (12 разрядов) – DAC2.

3.1.4. Встроенный аналоговый компаратор

В микроконтроллере реализована схема компаратора (рис. 3.19), обеспечивающая следующие режимы работы:

– сравнение двух сигналов с трех различных выводов микросхемы;

– сравнение сигнала с трех различных выводов с внутренней шкалой напряжений;

– сравнение сигнала с вывода IN1 с внутренним источником опорного напряжения;

– формирование внутренней шкалы напряжений от питания микроконтроллера и от внешних выводов.

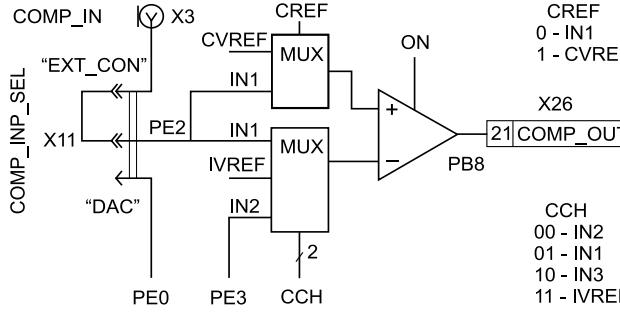


Рис. 3.19. Схема подключения встроенного компаратора

Спецификация регистров настройки контроллера компаратора:

`MDR_COMP`, базовый адрес `0x4009_8000` – контроллер аналогового компаратора;

`MDR_COMP->CFG` [31:0], смещение 0x00 – регистр управления аналогового компаратора.

Назначение битов регистра:

[31:14] – не используются;

[13] (CMP IE) – флаг разрешения генерации прерывания по событию `Rst_lch` (0 – запрещено, 1 – разрешено);

[12] (Ready) – сигнал готовности аналогового компаратора при включении: (0 – компаратор не включен или не готов к работе, 1 – компаратор готов к работе);
[11] (INV) – инверсия выхода компаратора (1 – инверсный);
[10:9] (CCH) – биты управления мультиплексором канала:
00 – на вход компаратора "–" сигнал подается с IN2;
01 – на вход компаратора "–" сигнал подается с IN1;
10 – на вход компаратора "–" сигнал подается с IN3;
11 – на вход компаратора "–" сигнал подается с выхода внутреннего источника опорного напряжения 1.2 В (IVREF);
[8] (CREF) – бит управления мультиплексором канала:
0 – на вход компаратора "+" сигнал подается с IN1;
1 – на вход компаратора "+" сигнал подается с CVREF;
[7:4] (CVR) – биты управления мультиплексором выбора CVREF при формировании внутренней шкалы напряжений;
[3] (CVREN) – разрешение работы CVREF (0 – запрет; 1 – разрешение);
[2] (CVRSS) – база CVREF (0–AUcc : AGND, 1 – Vref+ : Vref–);
[1] (CVRR) – выбор диапазона CVREF (0 – верхний, 1 – нижний);
[0] (ON) – включение компаратора (0 – выключен, 1 – включен).

`MDR_COMP->RESULT`, смещение 0x04 – регистр результата:
[31:3] – не используются;
[2] (Rst_lch) – результат работы компаратора хранится до момента считывания из регистра `MDR_COMP->RESULT_LATCH`, затем сбрасывается. Взводится по переднему фронту сигнала с аналогового компаратора;
[1] – результат работы компаратора получают непосредственно из аналогового компаратора;
[0] – результат работы компаратора синхронизирован с частотой HCLK.

`MDR_COMP->RESULT_LATCH`, смещение 0x08 – регистр-защелка для фиксирования результата работы аналогового компаратора;
[31:1] – не используются;
[0] (Rst_lch) – результат работы компаратора хранится до момента считывания из регистра `MDR_COMP->RESULT_LATCH`, затем сбрасывается. Взводится по переднему фронту сигнала с компаратора.

Внутренняя шкала напряжений формируется на резистивном делителе (рис. 3.20), который включается битом CVREN = 1. При этом в качестве опорного напряжения делителя может выступать питание микросхемы AUcc (CVRSS = "0") или напряжение "COMP_VREF+"

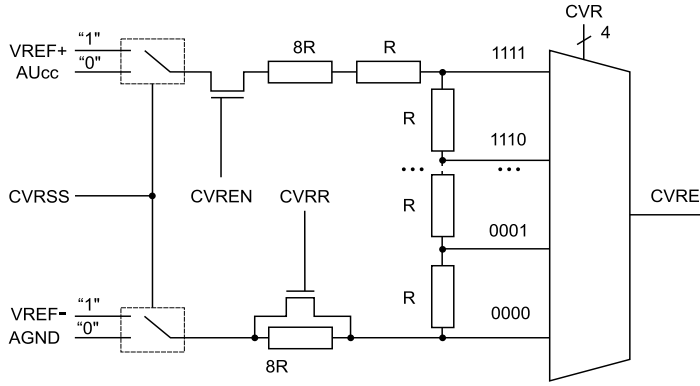


Рис. 3.20. Структура блока формирования CVREF

Таблица 3.3

Формирование внутренней шкалы напряжений CVREF

CVR[3:0]	CVRR = 0		CVRR = 1	
	Отношение резисторов	U(CVREF), В при Ucc = 3.3 В	Отношение резисторов	U(CVREF), В при Ucc = 3.3 В
0000	8/32	0.83	0/24	0.00
0001	9/32	0.93	1/24	0.14
0010	10/32	1.03	2/24	0.28
0011	11/32	1.13	3/24	0.41
0100	12/32	1.24	4/24	0.55
0101	13/32	1.34	5/24	0.69
0110	14/32	1.44	6/24	0.83
0111	15/32	1.55	7/24	0.96
1000	16/32	1.65	8/24	1.10
1001	17/32	1.75	9/24	1.24
1010	18/32	1.86	10/24	1.38
1011	19/32	1.96	11/24	1.51
1100	20/32	2.06	12/24	1.65
1101	21/32	2.17	13/24	1.79
1110	22/32	2.27	14/24	1.93
1111	23/32	2.37	15/24	2.06

Примечание. Значения напряжения U(CVREF) приведены для справки, реальные значения напряжений в конкретном кристалле могут отличаться за счет технологического разброса параметров.

(CVRSS = "1"). Нижнее опорное напряжение компаратора задается на выводе "COMP_VREF–". Напряжение на выводе CVREF задается комбинацией битов CVRR и CVR (табл. 3.3).

Результат работы компаратора (сигнал `Out`) может быть проинвертирован с помощью бита INV и выдан на вывод микросхемы OUT_COMP. Также результат сравнения доступен внутри микроконтроллера. Комбинационный сигнал с компаратора отображается в бите Rslt_As (при чтении может быть считан как "1", но сигнал прерывания при этом может отсутствовать). Зафиксированный в триггере по тактовой частоте HCLK сигнал сравнения отображается в бите Rslt_Sy. Флаг Rst_lch фиксирует событие появления положительного сигнала сравнения и устанавливается в "1" до тех пор, пока не будет считан регистр COMP_RESULT_LATCH.

Отметим, что не все микроконтроллеры серии `1986BExx` имеют полный набор внешних выводов, приведенный в спецификации [4]. В частности, микроконтроллер `1986BE92У` не имеет выводов "IN3", "Vref+" и "Vref–", поскольку корпус содержит всего 64 вывода.

Программа 10. На рис. 3.21 приведена программа, демонстрирующая работу встроенного компаратора. На вход "–" компаратора подается напряжение IVREF от встроенного источника опорного напряжения 1.2 В (см. рис. 3.19). Напряжение на входе "+" компаратора формируется схемой CVREF (см. рис. 3.20), которая настроена на два уровня напряжения ($U_1 < U(\text{IREF})$ и $U_2 > U(\text{IREF})$) с возможностью программного выбора одного из них с помощью кнопок джойстика "LEFT" и "RIGHT" (см. рис. 3.9 и 3.10). Результат работы компаратора отображается с помощью двух светодиодных индикаторов, а сигнал COMP_OUT выводится на внешний разъем X26 (ножка *21*) для наблюдения с помощью осциллографа.

Рассмотрим более детально организацию программы (см. рис. 3.21). Секция Include содержит дополнительный файл `MPT_comp.h`, который включает определения констант и прототипов функций, подробно обсуждавшихся ранее, а их исходные тексты не присутствуют на рис. 3.21.

Секция `Define` содержит новую константу `COMP_OUT`, соответствующую линии PB8 микроконтроллера, через которую осуществляется выдача выходного сигнала компаратора. Константы `RIGHT` и `LEFT` используются для программирования кнопок джойстика, они обсуждались ранее (см. рис. 3.9 и 3.10).

В начале секции `main` осуществляется инициализация используемого оборудования: тактовый генератор (frq_init), клавиши джойстика и светодиодные индикаторы (joystick_init), встроенный компаратор

```
/*-------------------------------------------------------------*/
#include <MDR32F9Qx_config.h>                // Device Startup
#include <MDR32Fx.h>                         // Device Header
#include <MPT_comp.h>                        // Local settings
#define COMP_OUT (1<<8)                      // PB8 "COMP_OUT"
#define RIGHT (1<<6)                         // PB6
#define LEFT  (1<<3)                         // PE3
/*-------------------------------------------------------------*/
int main (void)
{
   uint8_t cvref;
   frq_init();                              // Core CLK = 8 MHz
   joystick_init();
   comp_init();              // Activation of the comparator
   led_off(0);
   led_off(1);
   while (1){
     cvref = 0x00;
     if (!(MDR_PORTB->RXTX & RIGHT)) cvref = 0x03;   // RIGHT
     if (!(MDR_PORTE->RXTX & LEFT )) cvref = 0x04;   // LEFT
     if (cvref != 0x0) {
       MDR_COMP->CFG &= ~(0x0f<<4);
       MDR_COMP->CFG |= (cvref<<4);
       if ((MDR_COMP->RESULT & 0x02)) {led_off(0); led_on(1);}
       else {led_off(1); led_on(0);}
     }
   }
}
/*-------------------------------------------------------------*/
void comp_init(void)
{
   MDR_RST_CLK->PER_CLOCK |= (0x01<<19);     // Comp: CLC On
   MDR_COMP->CFG  = 0x0;
   MDR_COMP->CFG |= ((0x07<<8) | 0x09);      // Comp: On
   while (!(MDR_PORTE->RXTX & (0x01<<12)));  // Ready
   comp_osc_init();         // Activation of extern "COMP_OUT"
}
/*-------------------------------------------------------------*/
void comp_osc_init(void)
{  /* PORT B8 = COMP_OUT */
   MDR_PORTB->OE     |=   COMP_OUT;          // COMP_OUT (PB8)
   MDR_PORTB->ANALOG |=   COMP_OUT;
   MDR_PORTB->PWR    &= ~(COMP_OUT<<(8+1));
   MDR_PORTB->PWR    |=  (COMP_OUT<< 8   );
   MDR_PORTB->FUNC   |=  (0x01<<17);
   MDR_PORTB->FUNC   &= ~(0x01<<16);         //Alternative func
}
/*-------------------------------------------------------------*/
```

Рис. 3.21. Программа, демонстрирующая работу встроенного компаратора

(comp_init), линия порта PB8 для вывода сигнала с выхода компаратора на внешний разъем X26 (ножка *21*), а также выключение светодиодных индикаторов.

Основной бесконечный цикл `while` формирует и поддерживает переменную cvref, соответствующую параметру компаратора `CVREF`. В начале каждой итерации цикла `while` переменная cvref принимает нулевое значение, которое далее может быть изменено нажатием клавиши RIGHT ($U_1 < U(\text{IREF})$) или LEFT ($U_2 > U(\text{IREF})$). Каждая итерация цикла `while` завершается программной проверкой компаратора и отображением результата работы на светодиодных индикаторах.

Функция `comp_init()` осуществляет начальную настройку компаратора: подачу тактовой частоты на компаратор (`PER_CLOCK`, линия *19*), включение компаратора (ON), выбор источника внутреннего опорного напряжения IVREF (биты *9* и *10*), установку битов CVREN и CVREF (биты *3* и *8*), сброс битов CVRR и CVRSS (биты *1* и *2*). После ожидания установки флага готовности «Ready» (бит *12*) происходит вызов функции `comp_osc_init()`.

Функция `comp_osc_init()` осуществляет начальную настройку линий порта PB8 для вывода выходного сигнала компаратора на внешний разъем X26 (ножка *21*). Линия PB8 программируется для цифрового вывода в режиме медленных фронтов (до 100 нс). Особенностью является выбор альтернативной функции порта, соответствующей выходному сигналу COMP_OUT компаратора.

3.1.5. Аналого-цифровой преобразователь

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

В качестве опорного напряжения при работе аналого-цифрового преобразователя 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 ускорения выборки.

Для осуществления преобразования требуется не менее 28 тактов синхронизации CLK. В качестве синхросигнала может выступать частота процессора CPU_CLK либо частота ADC_CLK, формируемая в блоке "Сигналы тактовой частоты". Выбор частоты осуществляется с помощью бита Cfg_REG_CLKS. Частота CPU_CLK формируется из частоты процессорного ядра делением на коэффициент Cfg_REG_DIVCLK[3:0]. Максимальная частота CLK не может превышать 14 МГц (рис. 3.22).

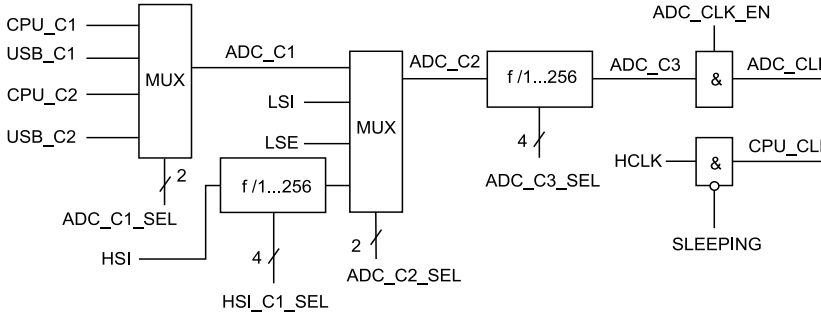


Рис. 3.22. Схема формирования тактовой частоты АЦП

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 (рис. 3.23). Для этого необходимо установить биты ADCx_OP в "1". Для преобразования необходимо выводы порта D (АЦП) сконфигурировать как аналоговые и отключить 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 (Cfg_M_REF= 0) или внешней (Cfg_M_REF= 1), в этом случае опора берется с выводов "ADC0_REF+" и "ADC1_REF–". Биты Cfg_REG_RNGC, Cfg_REG_SAMPLE, TS_BUF_EN, SEL_VREF, SEL_TS и Cfg_Sync_Conver для краткости обсуждения назовем *битами первой группы*. Все биты первой группы в этом режиме должны

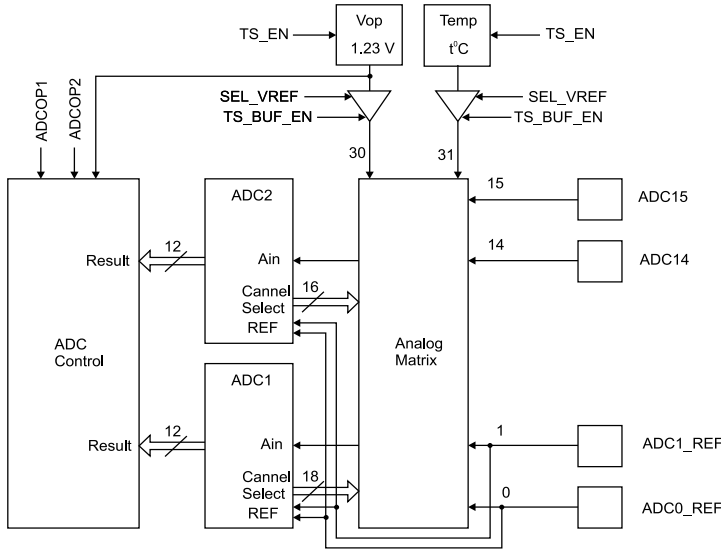


Рис. 3.23. Структурная схема контроллера АЦП

быть сброшены. Необходимо также сброить бит Cfg_REG_CHCH. Для начала преобразования в бит Cfg_REG_GO следует записать логическую единицу. Это соответствует запуску АЦП.

Состояние после завершения преобразования – будет установлен бит Flg_REG_EOCIF в регистре ADCx_STATUS, а результат преобразования будет размещен в регистре ADCx_RESULT. 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 Flg_REG_EOCIF, будет установлен дополнительный флаг – бит Flg_REG_OVERWRITE. Флаг Flg_REG_OVERWRITE может быть сброшен только записью в регистр ADCx_STATUS. После считывания результата будет автоматически сброшен бит Flg_REG_EOCIF.

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. Как и в предыдущем случае, последовательное преобразование нескольких каналов может осуществляться при внутренней опоре (Cfg_M_REF= 0) или внешней (Cfg_M_REF= 1), в этом случае опо-

ра берется с выводов "ADC0_REF+" и "ADC1_REF–". Биты первой группы должны быть сброшены, а биты Cfg_REG_CHCH должны быть установлены. С помощью бит Delay_GO можно задать паузу между преобразованиями при переборе каналов. Это определяется в единицах тактов 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.

Состояние после завершения преобразования полностью идентично предыдущему случаю.

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 может 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 выхода результата преобразования за границы выставляется флаг Flg_REG_AWOIFEN, а в регистре результата будет полученное значение.

Датчик опорного напряжения. С помощью первого АЦП можно осуществить преобразования источника опорного напряжения. 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, что позволяет снизить ток потребления. Для этого необходимо установить биты ADCx_OP в единицу. Для выбора источника опорного напряжения в качестве источника для преобразования необходимо в битах Cfg_REG_CHS установить значение 30-го канала и биты TS_BUF_EN и SEL_VREF, после чего можно запустить процесс преобразования. Для запуска преобразования необходимо записать "1" в бит Cfg_REG_GO. Состояние после завершения преобразования первого АЦП не отличается от рассмотренных ранее случаев.

Для последовательного преобразования только источника опорного напряжения можно в регистре ADC1_CHSEL выбрать только 30-й канал и установить бит Cfg_REG_CHCH в "1" либо установить номер

30-го канала в битах Cfg_REG_CHS[4:0] и сбросить Cfg_REG_CHCH в "0". В этом случае процесс последовательного преобразования будет выполняться только для данного канала. При этом должны быть также установлены биты TS_BUF_EN и SEL_VREF. Датчик опорного напряжения не может быть использован для задания опорного напряжения преобразования.

Датчик температуры. С помощью первого АЦП можно осуществить преобразования датчика опорного напряжения. Для этого необходимо включить блок датчика температуры и источник опорного напряжения, установив бит TS_EN в единицу. После включения можно использовать источник опорного напряжения для первого и второго АЦП вместо их собственных, что позволяет снизить ток потребления. Для этого необходимо установить биты ADCx_OP в единицу. Для выбора датчика температуры в качестве источника для преобразования необходимо в битах Cfg_REG_CHS установить значение 31-го канала, установить биты TS_BUF_EN и SEL_TS, после чего можно запустить процесс преобразования. Для начала преобразования необходимо записать "1" в бит Cfg_REG_GO.

После завершения преобразования будет взведен Flg_REG_EOCIF в регистре ADC1_STATUS, а в регистре ADC1_RESULT будет размещен результат преобразования. После считывания результата будет автоматически сброшен флаг Flg_REG_EOCIF.

Если после первого преобразования результат не был считан и было выполнено второе преобразование, то в регистре результата ADC1_RESULT будет размещено значение от последнего преобразования, а помимо бита Flg_REG_EOCIF будет установлен дополнительный флаг Flg_REG_OVERWRITE. Флаг Flg_REG_OVERWRITE может быть сброшен только записью в регистр ADC1_STATUS.

Для последовательного преобразования только датчика температуры можно в регистре ADC1_CHSEL выбрать только 31-й канал и установить бит Cfg_REG_CHCH в "1" либо установить номер 31-го 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. При этом должны быть также установлены биты TS_BUF_EN и SEL_TS.

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 пре-

образование. 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. Для корректного задания режимов работы АЦП все изменения в регистре ADCx_CFG необходимо сделать до задания бита Go или Cfg_REG_SAMPLE, иначе новая конфигурация будет действовать со следующего преобразования.

Назначение регистров контроллера АЦП

`MDR_RST_CLK->ADC_MCO_CLOCK`, смещение 0x14 – регистр управления тактовой частотой АЦП (см. рис. 3.22):

[31:14] – биты не используются;

[13] (ADC_CLK_EN) – бит разрешения выдачи тактовой частоты ADC_CLK (1 – разрешен, 0 – запрещен);

[12] – бит не используется;

[11:8] (ADC_C3_SEL) – выбор делителя для ADC_C3:
0xxx – ADC_C3 = ADC_C2;
1000 – ADC_C3 = ADC_C2 / 2;
1001 – ADC_C3 = ADC_C2 / 4;
1010 – ADC_C3 = ADC_C2 / 8;
… …
1111 – ADC_C3 = ADC_C2 / 256;

[7:6] – не используются;

[5:4] (ADC_C2_SEL) – выбор источника для ADC_C1:
00 – LSE;
01 – LSI;
10 – ADC_C1;
11 – HSI_C1;

[3:2] – не используются;

[1:0] (ADC_C1_SEL) – выбор источника для ADC_C1:
00 – CPU_C1 (до умножителя PLL);
01 – USB_C1 (до умножителя PLL);
10 – CPU_C2 (после умножителя PLL);
11 – USB_C2 (после умножителя PLL);

`MDR_RST_CLK->PER_CLOCK`, смещение 0x1c – регистр управления тактовой частотой периферийных блоков, бит [17] отвечает за включение АЦП;

`MDR_ADC`, базовый адрес `0x4008_8000` – контроллер АЦП;

`MDR_ADC->ADC1_CFG`, смещение 0x00 – регистр управления ADC1;

[31:28] (Delay_ADC) – задержка между началом преобразования ADC1 и ADC2 при последовательном переборе либо работе на один канал:
0000 – 1 такт CPU_CLK;
0001 – 2 такта CPU_CLK;
… …
1111 – 16 тактов CPU_CLK.

[27:25] (Delay_Go) – дополнительная задержка перед началом преобразования после выбора канала:
000 – 1 такт CPU_CLK;
001 – 2 такта CPU_CLK;
… …
111 – 8 тактов CPU_CLK.

[24:21] (TR) – подстройка опорного напряжения;

[20] (SEL_VREF) – выбор для оцифровки источника опорного напряжения на 1.23 В (0 – не выбран; 1 – выбран), должен использоваться совместно с выбором канала Cfg_REG_CHS= 30;

[19] (SEL_TS) – выбор для оцифровки датчика температуры (0 – не выбран; 1 – выбран), должен использоваться совместно с выбором канала Cfg_REG_CHS = 31;

[18] (TS_BUF_EN) – включение выходного усилителя для датчика температуры и источника опорного напряжения (0 – выключен; 1 – включен), используется при TS_EN= 1;

[17] (TS_EN) – включение датчика температуры и источника опорного напряжения (0 – выключен; 1 – включен); при включении датчика температуры и источника опорного напряжения выходной сигнал стабилизируется в течение 1 мс;

[16] (Cfg_Sync_Conver) – запускает работу двух АЦП одновременно, при этом биты конфигурации ADC2 (Cfg_REG_DIVCLK, Cfg_REG_ADON, Cfg_M_REF и Cfg_REG_CHS) берутся из регистра конфигурации ADC1 (0 – независимые АЦП; 1 – синхронные АЦП);

[15:12] (Cfg_REG_DIVCLK) – выбор коэффициента деления частоты процессора:
0000 – CPU_CLK = HCLK;
0001 – CPU_CLK = HCLK/2;
0010 – CPU_CLK = HCLK/4;
0011 – CPU_CLK = HCLK/8;
… …
1011 – CPU_CLK = HCLK/2048.
Остальные CPU_CLK = HCLK;

[11] (Cfg_M_REF) – выбор источника опорных напряжений:
0 – внутреннее опорное напряжение (от AUcc и AGND);
1 – внешнее опорное напряжение (от "ADC0_REF+" и newline "ADC1_REF–");

[10] (Cfg_REG_RNGC) – разрешение автоматического контроля уровней (0 – не разрешено; 1 – разрешена выработка флага при выходе за диапазон в регистрах границы);

[9] (Cfg_REG_CHCH) – выбор переключения каналов: 0 – используется только выбранный канал; 1 – переключение включено (перебираются каналы, выбранные в регистре выбора канала);

[8:4] (Cfg_REG_CHS) – выбор аналогового канала, по которому поступает сигнал для преобразования:
00000 – 0-й канал;
00001 – 1-й канал;
… …
11111 – 31-й канал.

[3] (Cfg_REG_SAMPLE) – выбор способа запуска АЦП (0 – одиночное; 1 – последовательное), автоматический запуск после завершения предыдущего преобразования;

[2] (Cfg_REG_CLKS) – выбор источника синхросигнала CLK работы ADC (0 – CPU_CLK; 1 – ADC_CLK);

[1] (Cfg_REG_GO) – начало преобразования; запись "1" начинает процесс преобразования, сбрасывается автоматически;

[0] (Cfg_REG_ADON) – включение АЦП (0 – выключено; 1 – включено);

`MDR_ADC->ADC2_CFG`, смещение 0x04 – регистр управления ADC2;

[31:28] – не используются;

[27:25] (Delay_Go) – задержка перед началом следующего преобразования после завершения предыдущего при последовательном переборе каналов:
000 – 0 тактов CPU_CLK;
001 – 1 такт CPU_CLK;
… …
111 – 7 тактов CPU_CLK.

[24:19] – не используются;

[18] (ADC2_OP) – выбор источника опорного напряжения 1.23 В: 0 – внутренний (неточный); 1 – от датчика температуры (точный). Необходимо, чтобы бит TS_EN был равен 1;

[17] (ADC1_OP) – выбор источника опорного напряжения 1.23 В: 0 – внутренний (неточный); 1 – от датчика температуры (точный). Необходимо чтобы бит TS_EN был равен 1;

[16] – не используется;

[15:12] (Cfg_REG_DIVCLK) – выбор коэффициента деления частоты процессора:
0000 – CPU_CLK = HCLK;
0001 – CPU_CLK = HCLK/2;
0010 – CPU_CLK = HCLK/4;
0011 – CPU_CLK = HCLK/8;
… …
1011 – CPU_CLK = HCLK/2048;
Остальные CPU_CLK = HCLK;

[11] (Cfg_M_REF) – выбор источника опорных напряжений:
0 – внутреннее опорное напряжение (от AUcc и AGND);
1 – внешнее опорное напряжение (от "ADC0_REF+" и "ADC1_REF–");

[10] (Cfg_REG_RNGC) – разрешение автоматического контроля уровней (0 – не разрешена; 1 – разрешена выработка флага при выходе за диапазон в регистрах границы);

[9] (Cfg_REG_CHCH) – выбор переключения каналов: 0 – используется только выбранный канал; 1 – переключение включено (перебираются каналы, выбранные в регистре выбора канала);

[8:4] (Cfg_REG_CHS) – выбор аналогового канала, по которому поступает сигнал для преобразования:
00000 – 0-й канал;
00001 – 1-й канал;
… …
11111 – 31-й канал.

[3] (Cfg_REG_SAMPLE) – выбор способа запуска АЦП (0 – одиночное; 1 – последовательное), автоматический запуск после завершения предыдущего преобразования;

[2] (Cfg_REG_CLKS) – выбор источника синхросигнала CLK работы ADC (0 – CPU_CLK; 1 – ADC_CLK);

[1] (Cfg_REG_GO) – начало преобразования; запись "1" начинает процесс преобразования, сбрасывается автоматически;

[0] (Cfg_REG_ADON) – включение АЦП (0 – выключено; 1 – включено);

`MDR_ADC->ADC1_H_Level`, смещение 0x08 – регистр верхней границы ADC1;

`MDR_ADC->ADC2_H_Level`, смещение 0x0c – регистр верхней границы ADC2;

[31:12] – не используются;

[11:0] (REG_H_LEVEL) – верхняя граница зоны допуска;

`MDR_ADC->ADC1_L_Level`, смещение 0x10 – регистр нижней границы аналого-цифрового преобразователя ADC1;

`MDR_ADC->ADC2_L_Level`, смещение 0x14 – регистр нижней границы аналого-цифрового преобразователя ADC2;

[31:12] – не используются;

[11:0] (REG_L_LEVEL) – нижняя граница зоны допуска;

`MDR_ADC->ADC1_RESULT`, смещение 0x18 – регистр данных ADC1;

`MDR_ADC->ADC2_RESULT`, смещение 0x1c – регистр данных ADC2;

[31:21] – не используются;

[20:16] (CHANNEL) – канал результата преобразования;

[15:12] – не используются;

[11:0] (RESULT) – значение результата преобразования;

`MDR_ADC->ADC1_STATUS`, смещение 0x20 – регистр статуса ADC1;

`MDR_ADC->ADC2_STATUS`, смещение 0x24 – регистр статуса ADC2;

[31:5] – не используются;

[4] (ECOIF_IE) – флаг разрешения генерирования прерывания по событию Flg_REG_ECOIF (0 – прерывание не генерируется; 1 – прерывание генерируется);

[3] (AWOIF_IE) – флаг разрешения генерирования прерывания по событию Flg_REG_AWOIFEN (0 – прерывание не генерируется; 1 – прерывание генерируется);

[2] (Flg_REG_EOCIF) – флаг выставляется, когда закончено преобразование и данные еще не считаны. Очищается считыванием результата из регистра ADCx_RESULT (1 – есть готовый результат преобразования; 0 – нет результата);

[1] (Flg_REG_AWOIFEN) – флаг выставляется, когда результат преобразования выше верхней или ниже нижней границы автоматического контроля уровней. Сбрасывается только при записи нуля в данный регистр флагов (0 – результат в допустимой зоне; 1 – вне допустимой зоны);

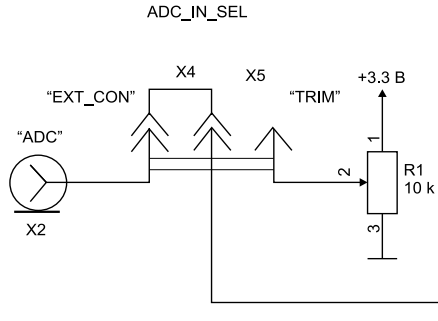
[0] (Flg_REG_OVERWRITE) – данные в регистре результата были перезаписаны, данный флаг сбрасывается только при записи нуля в данный бит регистра флагов (0 – не было события перезаписи несчитанного результата; 1 – был результат преобразования, который не был считан);

`MDR_ADC->ADC1_CHSEL`, смещение 0x28 – регистр выбора каналов перебора аналого-цифрового преобразователя ADC1;

`MDR_ADC->ADC2_CHSEL`, смещение 0x2c – регистр выбора каналов перебора аналого-цифрового преобразователя ADC2;

[31:0] (Sl_Ch_Ch_REF) – выбор каналов автоматического перебора (0 – в соответствующем бите, если канал не участвует в переборе; 1 – канал участвует в переборе);

Программа 11. На рис. 3.24 приведен фрагмент принципиальной схемы входных цепей встроенного АЦП. Перемычка ADC_IN_SEL (X5) должна быть установлена в положение TRIM. Входной сигнал для преобразования при этом поступает с многооборотного переменного резистора TRIM на линию PD7 порта PORTD. Выбран режим работы с одним АЦП (ADC1), который подключен к седьмому каналу



Пин	Порт	Функция
23	PD0	ADC0_REF+
24	PD1	ADC1_REF–
25	PD2	ADC2
26	PD3	ADC3
22	PD4	ADC4
27	PD5	ADC5
28	PD6	ADC6
21	PD7	ADC7

Рис. 3.24. Схема входных линий АЦП на отладочной плате `1986EvBrd_64` (микроконтроллер 1986ВЕ92У, 64 вывода, тип корпуса Н16.64-1В)

и тактируется частотой CPU_CLK. Все задержки выбраны минимально возможные в 1 машинный такт. В качестве опорного напряжения выбран источник AUcc : GND. Реализован однократный запуск АЦП с последующей проверкой флага Flg_REG_EOCIF в регистре состояния MDR_ADC->ADC1_STATUS. Вывод результатов работы осуществляется на графический дисплей в три строки: первая строка содержит информацию о тактовой частоте микроконтроллера; вторая строка содержит 12-разрядный цифровой код, являющийся результатом преобразования; третья строка содержит информацию о величине напряжения, поступающего на вход АЦП. На рис. 3.25 приведена программа, демонстрирующая работу встроенного АЦП. Для доступа к функциям графического дисплея к проекту подключена библиотека lcd_mlt.lib. Все измерения проводят в бесконечном цикле `while`, поэтому можно наблюдать изменение показаний дисплея в режиме реального времени при изменении сопротивления переменного резистора TRIM.

```
/*-------------------------------------------------------------*/
#include <MDR32F9Qx_config.h>          // Device Startup
#include <MDR32Fx.h>                   // Device Header
#include <.\mlt\lcd_mlt.h>             // LCD module functions
#include <.\mlt\font.h>                // Fonts
#include <stdio.h>
#define STR_LENGTH 16          //Length of the string in bytes
/*-------------------------------------------------------------*/
void frq_init(void);                          // Core CLK = 8 MHz
void adc_init(void);
uint8_t * ascii(char symbol);
void print_mlt(char* ch_str, int str_num);
/*-------------------------------------------------------------*/
int main (void)
{
   char ch_str[STR_LENGTH];
   uint16_t adc_result;
   uint32_t frq_result;
   float adc_result_volt;
   frq_init();                               // Core CLK = 8 MHz
   adc_init();
   MltPinCfg();                    //Pins configuration for LCD
   LcdInit();                                  //LCD module init
   LcdClearChip(1);                               //Clear chip 1
   LcdClearChip(2);                               //Clear chip 2
   frq_result = SystemCoreClock/1000;
   sprintf(ch_str,"    %u %c%c%c",frq_result, 'k','H','z');
   print_mlt(ch_str,2);
   while(1) {
     MDR_ADC->ADC1_CFG |= 0x02;                    // ADC Start
     while(!(MDR_ADC->ADC1_STATUS & 0x04)) __NOP();
     adc_result = MDR_ADC->ADC1_RESULT&0x0FFF;
     adc_result_volt = 3.3*adc_result/0x0FFF;
     sprintf(ch_str,"    %u",adc_result);
     print_mlt(ch_str,4);
     sprintf(ch_str,"    %1.3f V",adc_result_volt);
     print_mlt(ch_str,6);
   }
}
/*-------------------------------------------------------------*/
```

Рис. 3.25. Программа, демонстрирующая работу встроенного АЦП

Секция Include содержит ссылки на два стандартных файла описания оборудования (MDR32F*.h), файл описания функций доступа к графическому дисплею (lcd_mlt.h), файл с отображаемыми шрифтами для вывода на графический дисплей (font.h) и файл описания стандартной библиотеки ввода-вывода (stdio.h). Секция Define содержит определение константы STR_LENGTH, задающей размер строки для вывода на графический дисплей.

В секции прототипов описаны следующие функции:
frq_init – настройка тактирования микроконтроллера (частота 8 МГц);
adc_init – начальная инициализация АЦП;
ascii – перекодировка входного символа в ASCII кодировке
в указатель на массив с графическим образом этого символа;
print_mlt – перекодировка строки символов в ASCII кодировке
в строку указателей на графические образы этих символов.

Ранее уже обсуждались исходные тексты первых двух функций frq_init (см. рис. 3.6) и ascii (см. рис. 3.15).

В секции main декларированы следующие переменные: ch_str – указатель на строку длиной STR_LENGTH для символов в ASCII кодировке; adc_result – двухбайтовое целое для записи результата работы АЦП в двоичном коде; adc_result – четырехбайтовое целое для записи частоты тактирования; adc_result_volt – тип float для записи результата работы АЦП в вольтах. Далее идет инициализация оборудования: тактовый генератор (frq_init), АЦП (adc_init), порты контроллера для подключения графического дисплея (MltPinCfg), микросхема графического дисплея (LcdInit). Далее идут очистка графического дисплея и получение системной переменной SystemCoreClock, содержащей информацию о тактовой частоте.

Вывод информации на графический дисплей включает два этапа: (1) использование стандартной функции sprintf для форматного вывода в строку ASCII символов ch_str; (2) использование функции print_mlt для вывода этой строки на экран дисплея. Информация о тактовой частоте выводится до начала бесконечного цикла `while`. В теле цикла `while` осуществляется пуск АЦП, ожидание флага готовности результата и вывод результата в двух видах: двоичный код и напряжение в вольтах.

На рис. 3.26 приведены исходные тексты новых функций, использованных для работы с АЦП. Эти функции ранее не обсуждались, поэтому обсудим их более подробно.

Функция adc_init осуществляет начальную инициализацию аналого-цифрового преобразователя, заключающуюся в выборе частоты CPU_CLK и подаче ее для тактирования АЦП. Все настройки для однократного запуска одного АЦП сосредоточены в регистре MDR_ADC->ADC1_CFG и состоят в выборе седьмого канала для подачи на вход АЦП и включения АЦП.

Функция print_mlt имеет два входных параметра – указатель на текстовую строку ch_str в ASCII кодировке и позиционный номер str_num для размещения строки на графическом дисплее. Алгоритм работы заключается в преобразовании каждого ASCII-символа строки ch_str в указатель на графический образ этого символа. Преоб-

```
/*-------------------------------------------------------------*/
void adc_init(void)
{
   uint32_t ch_tmp;
   MDR_RST_CLK->PER_CLOCK |= (0x01<<17);          // ADC CLK On
   ch_tmp = 0x00;
   ch_tmp |= (0x01<<5);
   MDR_RST_CLK->ADC_MCO_CLOCK = ch_tmp;
   ch_tmp = 0x00;
   ch_tmp |= ((0x07<<4) | 0x01);          // Channel=7; ADC1 On
   MDR_ADC->ADC1_CFG = ch_tmp;
}
/*-------------------------------------------------------------*/
void print_mlt(char* ch_str, int str_num)
{
   uint8_t *array[STR_LENGTH], i, FLAG = 0;
   for(i = 0;i < STR_LENGTH;i++){
     if(ch_str[i] != NULL) {
       if(!FLAG) array[i] = ascii(ch_str[i]);
       else array[i] = sym_sp;
     }
     else {FLAG = 1; array[i] = sym_sp;}
   }
   LcdPutString(array, str_num);
}
/*-------------------------------------------------------------*/
```

Рис. 3.26. Подпрограммы-функции для начальной инициализации и вывода результатов работы АЦП

разование каждого символа осуществляется при помощи вспомогательной функции ascii. В результате преобразования получают строку array из указателей графических образов символов, соответствующих исходной строке ASCII-символов. В заключение вызывают функцию LcdPutString для вывода строки графических образов на экран дисплея. Особенностью функции print_mlt является дополнение количества элементов каждой строки до значения STR_LENGTH. Каждый дополнительный элемент соответствует символу пробела.

3.2. Программирование часов реального времени

Блок батарейного домена (зона батарейного питания) предназначен для обеспечения функций часов реального времени (Real time clock, RTC) и сохранения некоторого набора пользовательских данных при отключении основного источника питания. При снижении напряжения питания Ucc в блоке SW происходит автоматическое пе-

реключение питания BDUcc с Ucc на BUcc. Если на BUcc имеется отдельный источник питания (батарея), то батарейный домен остается включенным и может выполнять свои функции.

Часы реального времени. Часы реального времени позволяют организовать механизм отсчета времени в кристалле, в том числе при отключении основного источника питания. Включение часов реального времени осуществляется битом RTCEN. В качестве источника тактовой частоты часов реального времени могут выступать низкочастотные генераторы LSI, LSE или высокочастотные блоки HSE, HSI с дополнительным делителем до 256 (HSE и HSI формируются в блоке управления тактовыми частотами и могут быть выбраны только при наличии питания DUcc, LSI может быть выбран при наличии питания Ucc, LSE может быть выбран при наличии Ucc или BUcc). Выбор между источниками осуществляется битами RTCSEL. При возможном отключении основного источника питания Ucc в качестве источника тактовой частоты должен использоваться осциллятор LSE, так как он также имеет питание BDUcc. Биты управления осциллятором LSE расположены в батарейном домене и, таким образом, при отключении основного питания они не сбрасываются.

Для калибровки тактовой частоты используются биты CAL[6:0]. Значение CAL определяет, какое число тактов из 220 будет замаскировано. Таким образом, с помощью бит CAL[6:0] производится замедление хода часов. Изменение значения бит CAL может быть осуществлено в ходе работы часов реального времени.

Регистр RTC_DIV выступает в роли 20-битного предварительного делителя входной тактовой частоты таким образом, чтобы на его выходе была тактовая частота в 1 Гц. Для задания коэффициента деления регистра RTC_DIV используется регистр RTC_PRL. Регистр RTC_CNT предназначен для отсчета времени в секундах и работает на выходной частоте делителя RTC_DIV. Регистр RTC_ALR предназначен для задания времени, при совпадении с которым вырабатывается флаг прерывания и пробуждения процессора. Таким образом, бит STANDBY, отключающий внутренний регулятор напряжения, автоматически сбрасывается при совпадении RTC_CNT и RTC_ALR.

Бит STANDBY также может быть сброшен с помощью вывода WAKEUP.

Регистры аварийного сохранения и блока RTC. Батарейный домен имеет шестнадцать встроенных 32-разрядных регистров аварийного сохранения (BKP). 16-й и 15-й регистры служат для хранения битов управления батарейным доменом, оставшиеся 14 регистров могут быть использованы разработчиком программы.

Для работы с часами реального времени используют пять регистров RTC_CNT, RTC_DIV, RTC_PRL, RTC_ALRM, RTC_CS. Особенностью этих регистров является сравнительно медленная запись. Обращение к этим регистрам рекомендуется проводить по технологии «чтение–модификация–запись». В рамках этой технологии предусмотрен флаг готовности к записи (WEC= RTC_CS [6]). Его необходимо проверять перед каждым актом записи («чтение–модификация–запись») в регистр блока RTC. Отметим, что при непосредственной записи в регистр флаг WEC может работать некорректно.

Описание регистров блока батарейного домена. `MDR_BKP`, базовый адрес `0x400D_8000` – контроллер батарейного домена и часов реального времени;

`MDR_BKP->REG_00`, смещение 0x00 – BKP-регистр 00 [31:0];
`MDR_BKP->REG_01`, смещение 0x04 – BKP-регистр 01 [31:0];
`MDR_BKP->REG_02`, смещение 0x08 – BKP-регистр 02 [31:0];
…
`MDR_BKP->REG_0D`, смещение 0x34 – BKP-регистр 0D [31:0];
`MDR_BKP->REG_0E`, смещение 0x38 – BKP-регистр 0E и биты управления батарейным доменом:
[31:15] – не используются;
[14:12] (MODE[2..0]) – режим работы микроконтроллера, определенный при включении питания по выводам MODE[2:0] (PF[6:4]):
000 – микроконтроллер, с отладкой через JTAG_B;
001 – микроконтроллер, с отладкой через JTAG_A;
010 – микропроцессор, с отладкой через JTAG_B;
011 – микропроцессор без отладки;
100 – зарезервировано;
101 – UART загрузчик;
110 – UART загрузчик;
111 – зарезервировано (режим тестирования кристалла);
[11] (FPOR) – флаг срабатывания POR;
[10:8] (Trim[2:0]) – коэффициент настройки опорного напряжения встроенного регулятора напряжения DUcc. С помощью Trim осуществляется подстройка напряжения DUcc:
000 – DUcc +0.10 В; 100 – DUcc –0.01 В;
001 – DUcc +0.06 В; 101 – DUcc –0.04 В;
010 – DUcc +0.04 В; 110 – DUcc –0.06 В;
011 – DUcc +0.01 В; 111 – DUcc –0.10 В;
[7] (JTAG B) – разрешение работы порта JTAG B: (0 – запрещен, 1 – разрешен);

[6] (JTAG A) – разрешение работы порта JTAG A: (0 – запрещен, 1 – разрешен);
[5:3] (SelectRI[2:0]) – выбор дополнительной стабилизирующей нагрузки для встроенного регулятора напряжения DUcc. Детали выбора представлены в спецификации [4];
[2:0] (LOW[2:0]) – выбор режима работы встроенного регулятора напряжения DUcc. Значение LOW должно совпадать со значением SelectRI и выставляться в зависимости от тактовой частоты микроконтроллера. Детали выбора представлены в спецификации [4];

`MDR_BKP->REG_0F`, смещение 0x3C – BKP-регистр 0F и биты управления блоками RTC, LSE, LSI и HSI:

[31] (RTC_RESET) – сброс RTC (0 – часы не сбрасываются, 1 – сброс часов реального времени);
[30] (STANDBY) – режим отключения регулятора DUcc (0 – регулятор включен и выдает напряжение, 1 – регулятор выключен). Триггер сбрасывается флагом ALRF или по низкому уровню на выводе WAKEUP;
[29:24] (HSI_TRIM[5:0]) – коэффициент подстройки частоты генератора HSI. Детали выбора представлены в спецификации [4];
[23] (HSI_RDI) – флаг выхода генератора HSI в рабочий режим (0–генератор не запущен или не вышел в режим, 1 – генератор работает в рабочем режиме);
[22] (HSI_ON) – бит управления генератором HSI (0 – генератор выключен, 1 – генератор включен);
[21] (LSI_RDY) – флаг выхода генератора LSI в рабочий режим (0 – генератор не запущен или не вышел в режим, 1 – генератор находится в рабочем режиме);
[20:16] (LSI_TRIM[4:0]) – коэффициент подстройки частоты генератора LSI. Детали выбора представлены в спецификации [4];
[15] (LSI_ON) – бит управления генератором LSI (0 – генератор выключен, 1 – генератор включен);
[14] – не используется;
[13] (LSE_RDY) – флаг выхода генератора LSE в рабочий режим (0 – генератор не запущен или не вышел в режим, 1 – генератор работает в рабочем режиме);
[12:5] (CAL[7:0]) – коэффициент подстройки тактовой частоты часов реального времени (замедление хода). Детали выбора представлены в спецификации [4];
[4] (RTC_EN) – бит разрешения работы часов реального времени (0 – работа запрещена, 1 – работа разрешена);

[3:2] (RTC_SEL[1:0]) – биты выбора источника тактовой синхронизации часов реального времени:
00 – LSI; 10 – HSIRTC (формируется в блоке CLKRST);
01 – LSE; 11 – HSERTC (формируется в блоке CLKRST);

[1] (LSY_BYP) – бит управления генератором LSE (0 – режим осциллятора, 1 – режим работы на проход (внешний генератор));

[0] (LSE_ON) – бит управления генератором LSE (0 – генератор выключен, 1 – генератор включен).

`MDR_BKP->RTC_CNT`, смещение 0x40, [31:0] (RTC_CNT) – значение основного счетчика часов реального времени;

`MDR_BKP->RTC_DIV`, смещение 0x44,

[31:20] – не используется;

[19:0] (RTC_DIV) – значение счетчика предварительного делителя часов реального времени;

`MDR_BKP->RTC_PRL`, смещение 0x48,

[31:20] – не используется;

[19:0] (RTC_PRL) – значение основания для счета счетчика предварительного делителя часов реального времени;

`MDR_BKP->RTC_ALRM`, смещение 0x4C, [31:0] (RTC_ALRM) – значения для сравнения основного счетчика и выработки ALRF;

`MDR_BKP->RTC_CS`, смещение 0x50,

[31:7] – не используется;

[6] (WEC) – запись завершена (0 – можно записывать в регистры RTC, 1 – запись запрещена, идет запись в регистры RTC);

[5] (ALRF_IE) – флаг разрешения прерывания по совпадению основного счетчика и регистра RTC_ALRM (0 – нет совпадения, 1 – есть совпадение);

[4] (SECF_IE) – флаг разрешения прерывания по разрешению счета основного счетчика от счетчика предварительного деления (0 – нет разрешения счета, 1 – разрешение счета);

[3] (OWF_IE) – флаг разрешения прерывания по переполнению основного счетчика RTC_CNT (0 – нет переполнения, 1 – было достигнуто переполнение);

[2] (ALRF) – флаг совпадения регистра RTC_ALRM и основного счетчика (0 – нет совпадения, 1 – есть совпадение);

[1] (SECF) – флаг разрешения счета основного счетчика от счетчика предварительного деления (0 – нет разрешения счета, 1 – есть разрешение счета);

[0] (OWF) – флаг переполнения основного счетчика RTC_CNT (0 – нет переполнения, 1 – было переполнение).

Программа 12. На рис. 3.27 приведена программа, демонстрирующая работу часов реального времени в батарейном домене микроконтроллера. Программа инициализирует систему, состоящую из часов реального времени (RTC), расположенных в батарейном домене микроконтроллера и графического дисплея. На экране дисплея программа отображает статическую информацию о тактовой частоте микроконтроллера. После запуска часов программа каждую секунду отображает на экране дисплея динамическую информацию о текущем времени в формате «часы, минуты, секунды». Благодаря тому, что часы расположены в батарейном домене, работоспособность часов сохраняется после выключения основного питания контроллера. При повторном включении контроллера не происходит повторной инициализации часов, вместо этого программа просто выводит на экран дисплея динамическую информацию о текущем времени.

Секция Include содержит ссылки на два стандартных файла описания оборудования (MDR32F*.h), файл описания функций доступа к графическому дисплею (lcd_mlt.h), файл с отображаемыми шрифтами для вывода на графический дисплей (font.h) и файл описания стандартной библиотеки ввода-вывода (stdio.h).

Секция Define содержит определения констант: STR_LENGTH – размер строки для вывода на графический дисплей, RTC_RESET – бит сброса часов реального времени, RTC_EN – бит разрешения работы часов, WEC – флаг готовности регистров RTC к записи, LSE_RDY – флаг выхода низкочастотного тактового генератора LSE на стабильный режим работы. Кроме того, в этой секции определен новый тип данных RTC_Time, представляющий собой структуру для хранения информации о текущем времени в формате «часы, минуты, секунды».

В секции прототипов описаны следующие функции: frq_init – настройка тактирования микроконтроллера на частоту 8 МГц; ascii – для входного символа в ASCII кодировке возвращает указатель на массив с графическим образом этого символа; print_mlt – перевод строки символов в ASCII кодировке в строку указателей на графические образы этих символов. Исходные тексты этих трех функций обсуждались ранее и приведены на рис. 3.6 (frq_init), рис. 3.15 (ascii) и рис. 3.26 (print_mlt). Новые функции rtc_init – инициализация часов и RtcToTime – преобразование содержимого счетчика часов в стандартный формат «часы, минуты, секунды». Исходные тексты новых функций приведены на рис. 3.28.

В секции main декларированы следующие переменные: ch_str – указатель на строку длиной STR_LENGTH для символов в ASCII кодировке; ch_frm – строка, задающая формат вывода на графический дисплей; frq_result – информация о тактовой частоте, две переменные для

```
/*-------------------------------------------------------------*/
#include <MDR32F9Qx_config.h>                  // Device Startup
#include <MDR32Fx.h>                            // Device Header
#include <.\mlt\lcd_mlt.h>              // LCD module functions
#include <.\mlt\font.h>                                 // Fonts
#include <stdio.h>
#define STR_LENGTH 16           //Length of the string in bytes
#define RTC_RESET (uint32_t) 0x80000000
#define RTC_EN  (uint32_t) 0x10
#define WEC     (uint32_t) (1<<6)
#define LSE_RDY (uint32_t) (1<<13)
typedef struct{uint8_t hour;
               uint8_t min;
               uint8_t sec;} RTC_Time;
/*-------------------------------------------------------------*/
void frq_init(void);
uint8_t * ascii(char symbol);
void print_mlt(char* ch_str, int str_num);
unsigned char rtc_init (void);
void RtcToTime(uint32_t cnt, RTC_Time* time);
/*-------------------------------------------------------------*/
int main (void)
{
   char  ch_str[STR_LENGTH];
   char* ch_frm= "    %02u.%02u.%02u";
   uint32_t frq_result, oldTime, newTime;
   RTC_Time time;
   frq_init();                                 // Core CLK = 8 MHz
   rtc_init();                             // Real-time clock init
   MltPinCfg();                       //Pins configuration for LCD
   LcdInit();                                    //LCD module init
   LcdClearChip(1);                                 //Clear chip 1
   LcdClearChip(2);                                 //Clear chip 2
   frq_result = SystemCoreClock/1000;
   sprintf(ch_str,"    %u %c%c%c",frq_result, 'k','H','z');
   print_mlt(ch_str,2);
   oldTime = 0x0;
   while(1) {
     newTime = MDR_BKP->RTC_CNT;
     if(newTime > oldTime) {
       RtcToTime(newTime,&time);
       sprintf(ch_str,ch_frm,time.hour,time.min,time.sec);
       print_mlt(ch_str,4);
     }
     oldTime = newTime;
   }
}
/*-------------------------------------------------------------*/
```

Рис. 3.27. Программа для демонстрации работы часов реального времени в батарейном домене микроконтроллера

```
/*-------------------------------------------------------------*/
unsigned char rtc_init (void)
{
   if((MDR_BKP->REG_0F & RTC_EN) != RTC_EN) {          //RTC_EN
     MDR_RST_CLK->PER_CLOCK |=  ((1<<25) | (1<<27));// BKP & PE
     MDR_PORTE->ANALOG &= ~((1<<6) | (1<<7)); //ANALOG PE6, PE7
     MDR_BKP->REG_0F  |= RTC_RESET;                  //RTC_RESET
     MDR_BKP->REG_0F   = 0x0;                 // Clear Register
     MDR_BKP->REG_0F  |= 0x04;                   // LSE select
     while((MDR_BKP->RTC_CS & WEC) != 0) {__NOP();}
     MDR_BKP->RTC_CS  |= 0x02;           //F:20bit is available
     while((MDR_BKP->RTC_CS & WEC) != 0) {__NOP();}
     MDR_BKP->RTC_PRL = 0x0;
     MDR_BKP->RTC_PRL |= (uint32_t) 0x7FFF; // 32767 +1 = 32768
     MDR_BKP->REG_0F  |= 0x01;                     //LSE ON
     while((MDR_BKP->REG_0F & LSE_RDY)== 0){__NOP();}  //LSE OK
     MDR_BKP->REG_0F  |= RTC_EN;                  // RTC ON
     while((MDR_BKP->REG_0F & RTC_EN) == 0) {__NOP();} //RTC OK
     return(1);
   }
   return(0);
}
/*-------------------------------------------------------------*/
void RtcToTime(uint32_t cnt, RTC_Time* time )
{
   time->sec = cnt % 60;
   cnt /= 60;
   time->min = cnt % 60;
   cnt /= 60;
   time->hour = cnt % 24;
}
/*-------------------------------------------------------------*/
```

Рис. 3.28. Подпрограммы-функции для начальной инициализации и преобразования данных счетчика часов реального времени

хранения информации о прошедшем и текущем времени в формате счетчика часов oldTime и newTime, а также time – структура для хранения информации о времени в стандартном формате «часы, минуты, секунды».

Далее идет инициализация оборудования: frq_init – тактовый генератор, rtc_init – часы реального времени, MltPinCfg – порты контроллера для подключения дисплея, LcdInit – микросхема дисплея. Затем идут очистка графического дисплея и получение системной переменной SystemCoreClock, содержащей информацию о тактовой частоте.

Вывод информации на графический дисплей включает два этапа: (1) использование стандартной функции sprintf для форматного вывода в строку ASCII символов ch_str; (2) использование функции

print_mlt для вывода этой строки на экран дисплея. Информация о тактовой частоте выводится до начала бесконечного цикла `while`. В теле цикла `while` осуществляется преобразование содержимого счетчика часов в информацию о времени в стандартном формате «часы, минуты, секунды» посредством функции RtcToTime и вывод результата на графический дисплей. Особенностью этого вывода является использование двух переменных oldTime и newTime, чтобы выводить информацию не чаще одного раза в секунду. Обсудим организацию двух новых подпрограмм-функций rtc_init и RtcToTime (см. рис. 3.28).

Начальная инициализация часов реального времени включает в себя следующие основные этапы:

(1) проверка состояния бита RTC_EN. Если бит RTC_EN установлен, то инициализация не проводится. Это соответствует случаю, когда часы были запущены раньше, и после включения основного питания микроконтроллера требуется только вывод на графический дисплей информации о текущем времени. Если бит RTC_EN сброшен, то выполняются последующие этапы инициализации;

(2) включение тактирования регистров аварийного сохранения данных (BKP) и порта PORTE, к выводам которого (PE6 и PE7) подключен часовой кварц генератора LSE;

(3) назначение аналогового режима работы выводов PE6 и PE7;

(4) сброс часов путем установки бита RTC_RESET;

(5) очистка всего регистра REG_0F от случайных данных;

(6) выбор генератора LSE в качестве источника тактирования;

(7) разрешение тактирования счетчика часов от 20-битного предварительного делителя частоты. Перед обращением к любому регистру блока RTC производится ожидание (while) готовности к записи путем проверки флага WEC;

(8) настройка предварительного делителя путем загрузки числа в регистр основания счета RTC_PRL. Значение основания счета выбрано следующим образом: f(LSE)[Гц] – 1= 32768–1= 32767= 0x7FFF. Это позволяет получить на выходе предварительного делителя тактовую частоту 1 Гц (период следования импульсов составляет 1 с);

(9) включение генератора LSE и ожидание его выхода на стабильный рабочий режим;

(10) включение часов реального времени (RTC) и ожидание установления флага RTC_EN.

Необходимо отметить, что очень важную роль при инициализации часов реального времени играет правильная последовательность инициализации регистров. Изменение последовательности (см. рис. 3.28) может привести, например, к срыву генерации LSE или сбою в работе предварительного делителя.

Функция RtcToTime преобразует содержимое счетчика часов (cnt – количество секунд с начала счета) в стандартный формат «часы, минуты, секунды» и выдает его в формате заданной структуры данных (тип данных RTC_Time) [10]. Алгоритм преобразования базируется на последовательном применении операции целочисленного деления (%) с основаниями 60, 60 и 24 (см. рис. 3.28).

Для начальной установки времени в счетчике часов использована отдельная программа (рис. 3.29). Необходимо отметить, что эта программа предназначена только для демонстрационных целей.

Секция Include содержит ссылки на два стандартных файла описания оборудования (MDR32F*.h).

Секция Define содержит определения следующих констант:

WEC – флаг готовности регистров RTC к записи;

TIME_HOUR, TIME_MIN и TIME_SEC – для начальной установки времени в счетчике часов.

В секции main происходит ожидание готовности флага WEC, вычисление начального содержимого для счетчика часов реального времени и запись этого содержимого в регистр RTC_CNT.

```
/*-------------------------------------------------------------*/
#include <MDR32F9Qx_config.h>                  // Device Startup
#include <MDR32Fx.h>                            // Device Header
#define WEC     (uint32_t) (1<<6)
#define TIME_HOUR 18
#define TIME_MIN  12
#define TIME_SEC   3
/*-------------------------------------------------------------*/
int main (void)
{
   while((MDR_BKP->RTC_CS & WEC) != 0) {__NOP();}
   MDR_BKP->RTC_CNT = (uint32_t) (TIME_HOUR * 3600 + \
                                  TIME_MIN * 60 +    \
                                  TIME_SEC);
}
/*-------------------------------------------------------------*/
```

Рис. 3.29. Программа для начальной установки времени в счетчике часов реального времени

3.3. Программирование счетчика-таймера и прерываний

Каждый из трех таймеров микроконтроллера выполнен 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).

Инициализация таймера. Перед началом работы с таймерами в первую очередь должны быть включены тактовые сигналы. Параметры задаются в блоке «Сигналы тактовой частоты». Для задания тактовой частоты блока необходимо установить бит разрешения тактирования блока (бит [14] для таймера 1, бит [15] для таймера 2, бит [16] для таймера 3 регистра PER_CLOCK). В регистре TIM_CLOCK необходимо установить бит TIMxCLKEN, чтобы разрешить тактовую частоту для определенного таймера, и задать коэффициент деления тактовой частоты HCLK для каждого таймера. После подачи тактового сигнала на блок таймера можно приступать к работе с ним. Для того чтобы запустить работу основного счетчика, следует задать:

– начальное значение основного счетчика таймера TIMx_CNT;

– значение предварительного делителя счетчика TIMx_PSG, при этом основной счетчик будет считать на частоте

$$CLK = TIMx\_CLK / (PSG + 1);$$

– значение основания счета для основного счетчика TIMx_ARR;

– режим работы счетчика TIMx_CNTRL;

– режим счета основного счетчика CNT_MODE (значения 00 и 01 при тактировании внутренней частотой, значения 10 и 11 при тактировании внешними сигналами);

– направление счета основного счетчика DIR;

– источник события переключения счетчика EVENT_SEL;

– разрешить работу счетчика CNT_EN.

По событиям совпадения значения основного счетчика со значением нуля или значением основания счета генерируется прерывание и запрос DMA, которые могут быть замаскированы.

Описание регистров блока таймера. Рассмотрим только регистры, непосредственно относящиеся к программированию счетчика-таймера. Описание регистров для настройки схем входного захвата и выходного сравнения содержится в спецификации [4].

`MDR_TIMERx`, базовые адреса 0x4007_0000 (Timer1), 0x4007_8000 (Timer2), 0x4008_0000 (Timer3) – контроллер счетчика-таймера;

`MDR_TIMERx->CNT`, смещение 0x00 – основной счетчик таймера:

[31:16] – не используется;

[15:0] (CNT[15:0]) – значение основного счетчика таймера;

`MDR_TIMERx->PSG`, смещение 0x04 – делитель частоты при счете основного счетчика:

[31:16] – не используется;

[15:0] (PSG[15:0]) – значение предварительного делителя счетчика для получения частоты счета CLK = TIM_CLK / (PSG + 1);

MDR_TIMERx->ARR, смещение 0x08 – основание счета основного счетчика CNT:

[31:16] – не используется;

[15:0] (ARR[15:0]) – основание счета для основного счетчика, динамический диапазон счетчика CNT = [0…ARR];

MDR_TIMERx->CNTRL, смещение 0x0C – регистр управления основного счетчика CNTRL:

[31:11] – не используется;

[11:8] (EVENT_SEL[3:0]) – биты выбора источника событий:
0000 – всегда нуль, другие детали выбора представлены в [4];

[7:6] (CNT_MODE[1:0]) – режим счета основного счетчика:

00 – счетчик вверх при DIR = 0 (при PSG = 0); счетчик вниз при DIR = 1 (при PSG = 0);

01 – счетчик вверх/вниз с автоматическим изменением DIR при PSG = 0;

10 – счетчик вверх при DIR = 0 (при EVENT = 1); счетчик вниз при DIR = 1 (при EVENT = 1);

11 – счетчик вверх/вниз с автоматическим изменением DIR (при EVENT = 1);

[5:4] (FDTS[1:0]) – частота семплирования данных FDTS:

00 – каждый TIM_CLK;
01 – каждый второй TIM_CLK;
10 – каждый третий TIM_CLK;
11 – каждый четвертый TIM_CLK;

[3] (DIR) – направление счета основного счетчика (0 – вверх, от 0 до ARR, 1 – вниз, от ARR до 0);

[2] (WR_CMPL) – окончание записи, при задании нового значения регистров CNT, PSG и ARR (0 – новые данные можно записывать, 1 – данные не записаны и идет запись);

[1] (ARRB_EN) – разрешение обновления значения ARR (0 – ARR будет перезаписан в момент записи в ARR, 1 – ARR будет перезаписан при завершении счета CNT);

[0] (CNT_EN) – разрешение работы таймера (0 – таймер отключен, 1 – таймер включен).

MDR_TIMERx->STATUS, смещение 0x54 – регистр статуса таймера. Сбрасывается записью «0», если запись одновременно с новым событием совпадения (0 – нет события, 1 – есть событие). Приоритет у нового события;

[31:17] – не используются;

[16:13] (CCR_CAP1_EVENT[3:0]) – событие переднего фронта на входе CAP1 каналов таймера;

[12:9] (CCR_REF_EVENT[3:0]) – событие переднего фронта на выходе REF каналов таймера;
[8:5] (CCR_CAP_EVENT[3:0]) – событие переднего фронта на входе CAP каналов таймера;
[4] (BRK_EVENT) – состояние входа BRK, синхронизированное по PCLK;
[3] (ETR_FE_EVENT) – событие заднего фронта на входе ETR;
[2] (ETR_RE_EVENT) – событие переднего фронта на входе ETR;
[1] (CNT_ARR_EVENT) – событие совпадения CNT = ARR;
[0] (CNT_ZERO_EVENT) – событие совпадения CNT с нулем.

`MDR_TIMERx->IE`, смещение 0x58 – регистр разрешения прерывания таймера:
[31:17] – не используются;
[16:13] (CCR_CAP1_EVENT_IE [3:0]) – флаг разрешения прерывания по событию переднего фронта на выходе CAP1 каналов таймера (0 – нет прерывания, 1 – прерывание разрешено; биты [0] – [3] соответствуют каналам с первого по четвертый);
[12:9] (CCR_REF_EVENT_IE[3:0]) – флаг разрешения прерывания по событию переднего фронта на выходе REF каналов таймера (0 – нет прерывания, 1 – прерывание разрешено; биты [0] – [3] соответствуют каналам с первого по четвертый);
[8:5] (CCR_CAP_EVENT_IE [3:0]) – флаг разрешения прерывания по событию переднего фронта на выходе CAP каналов таймера (0 – нет прерывания, 1 – прерывание разрешено; биты [0] – [3] соответствуют каналам с первого по четвертый);
[4] (BRK_EVENT_IE) – флаг разрешения по состоянию входа BRK, синхронизированному по PCLK (0 – нет прерывания, 1 – прерывание разрешено);
[3] (ETR_FE_EVENT_IE) – флаг разрешения прерывания по заднему фронту на входе ETR (0 – нет прерывания, 1 – прерывание разрешено);
[2] (ETR_RE_EVENT_IE) – флаг разрешения прерывания по переднему фронту на входе ETR (0 – нет прерывания, 1 – прерывание разрешено);
[1] (CNT_ARR_EVENT_IE) – флаг разрешения прерывания по событию совпадения CNT и ARR (0 – нет прерывания, 1 – прерывание разрешено);
[0] (CNT_ZERO_EVENT_IE) – флаг разрешения прерывания по событию совпадения CNT и нуля (0 – нет прерывания, 1 – прерывание разрешено).

`MDR_TIMERx->DMA_RE`, смещение 0x5C – регистр разрешения запросов ПДП (англ. DMA) от прерываний таймера;

[31:17] – не используются;
[16:13] (CCR_CAP1_EVENT_RE [3:0]) – флаг разрешения запроса DMA по событию переднего фронта на выходе CAP1 каналов таймера (0 – нет прерывания, 1 – прерывание разрешено; биты [0] – [3] соответствуют каналам с первого по четвертый);
[12:9] (CCR_REF_EVENT_RE[3:0]) – разрешение запроса DMA по событию переднего фронта на выходе REF каналов таймера (0 – нет прерывания, 1 – прерывание разрешено; биты [0] – [3] соответствуют каналам с первого по четвертый);
[8:5] (CCR_CAP_EVENT_RE [3:0]) – флаг разрешения запроса DMA по событию переднего фронта на выходе CAP каналов таймера (0 – нет прерывания, 1 – прерывание разрешено; биты [0] – [3] соответствуют каналам с первого по четвертый);
[4] (BRK_EVENT_RE) – флаг разрешения запроса DMA по состоянию входа BRK, синхронизированному по PCLK (0 – нет прерывания, 1 – прерывание разрешено);
[3] (ETR_FE_EVENT_RE) – флаг разрешения запроса DMA по заднему фронту на входе ETR (0 – нет прерывания, 1 – прерывание разрешено);
[2] (ETR_RE_EVENT_RE) – флаг разрешения запроса DMA по переднему фронту на входе ETR (0 – нет прерывания, 1 – прерывание разрешено);
[1] (CNT_ARR_EVENT_RE) – флаг разрешения запроса DMA по событию совпадения CNT и ARR (0 – нет прерывания, 1 – прерывание разрешено);
[0] (CNT_ZERO_EVENT_RE) – флаг разрешения запроса DMA по событию совпадения CNT и нуля (0 – нет прерывания, 1 – прерывание разрешено).

Векторы и обработчики прерываний. Таблица векторов прерываний микроконтроллеров Cortex начинается с адреса 0x00000004. Каждый из векторов прерываний занимает 4 байта и указывает на начальный адрес каждой конкретной процедуры обработки прерывания. Первые 15 векторов (номера 1–15) – адреса обработки исключительных ситуаций, возникающих в ядре Cortex. Начиная с 16-го вектора, следуют адреса обработки прерываний пользовательских УВВ. Их назначение зависит от каждого конкретного производителя. В программе пользователя таблица векторов приводится в отдельном файле и содержит адреса процедур обработки прерываний (см. рис. 1.7).

В интегрированной среде разработки Keil uVision имеются файлы поддержки оборудования для микроконтроллеров 1986 серии. В этих файлах предопределены векторы и имена обработчиков преры-

ваний. Фрагмент этого списка для системного таймера и IRQ0–IRQ31 приведен в табл. 3.4. Некоторые позиции оставлены незанятыми для использования в прикладных программах пользователя. Напомним, что внесение каких-либо изменений в этот список возможно только в привилегированном режиме.

Таблица 3.4

Таблица имен обработчиков векторных прерываний

IRQ	Обработчик прерывания	Примечание
SysTick	SysTick_Handler	Системный таймер
0	CAN1_IRQHandler	Контроллер CAN1
1	CAN2_IRQHandler	Контроллер CAN2
2	USB_IRQHandler	Контроллер USB
5	DMA_IRQHandler	Контроллер ПДП
6	UART1_IRQHandler	Контроллер UART1
7	UART2_IRQHandler	Контроллер UART2
8	SSP1_IRQHandler	Контроллер SSP1
10	I2C_IRQHandler	Контроллер I2C
11	POWER_IRQHandler	Контроллер PWR
12	WWDG_IRQHandler	Оконный WDT
14	Timer1_IRQHandler	Счетчик/таймер1
15	Timer2_IRQHandler	Счетчик/таймер2
16	Timer3_IRQHandler	Счетчик/таймер3
17	ADC_IRQHandler	АЦП
19	COMPARATOR_IRQHandler	Аналоговый компаратор
20	SSP2_IRQHandler	Контроллер SSP2
27	BACKUP_IRQHandler	Контроллер BKP
28	EXT_INT1_IRQHandler	Внешний INT1
29	EXT_INT2_IRQHandler	Внешний INT2
30	EXT_INT3_IRQHandler	Внешний INT3
31	EXT_INT4_IRQHandler	Внешний INT4

Программа 13. Программа (рис. 3.30) для демонстрации работы счетчика таймера CT1 в режиме прерываний выполняет инициализацию оборудования (тактовый генератор, порты вывода, счетчик-таймер CT1 и контроллер векторных прерываний NVIC). При этом тактовый генератор и счетчик-таймер CT1 программируются для выработки временного интервала длительностью примерно 1 с. Контроллер прерываний необходимо запрограммировать на обработку прерывания IRQ14, соответствующего заданному событию в счетчике-таймере CT1. Обработчик прерывания (Timer1_IRQHandler) каждую секунду осуществляет перезапуск счетчика-таймера с выводом состояния на светодиодные индикаторы.

Секция Include содержит только ссылки на два стандартных файла описания оборудования (MDR32F*.h).

```
/*-------------------------------------------------------------*/
#include <MDR32F9Qx_config.h>                 // Device Startup
#include <MDR32Fx.h>                           // Device Header
#define VD0           0x001                              // PC0
#define WR_CMPL       0x004
#define CNT_ARR_EVENT 0x002
/*-------------------------------------------------------------*/
void frq_init(void);
void Timer1_init(void);
void io_init (void);
void led_on(unsigned short);          // Declaration for led_on
void led_off(unsigned short);        // Declaration for led_off
/*-------------------------------------------------------------*/
int main (void)
{
   frq_init();                               // Core CLK = 8 MHz
   io_init ();
   timer_init();
   while(1) {__NOP();}
}
/*-------------------------------------------------------------*/
void Timer1_init(void)
{
   MDR_RST_CLK->PER_CLOCK |= (1<<14);             // Timer1 CLK
   MDR_RST_CLK->TIM_CLOCK  = 0x0;
   MDR_RST_CLK->TIM_CLOCK |= (1<<24);            // TIM1_CLK_EN
   MDR_RST_CLK->TIM_CLOCK |=  0x07;                 // HCLK/128
   MDR_TIMER1->CNTRL = 0x00000000;                 //Timer1 OFF
   MDR_TIMER1->CNT   = 0x00000000;                //Timer1 count
   MDR_TIMER1->PSG   = 0x040;                  //Pre-counter f/64
   while((MDR_TIMER1->CNTRL & WR_CMPL) != 0) {__NOP();}
   MDR_TIMER1->ARR   = 0x0400;          //Count base 0--ARR=1024
   while((MDR_TIMER1->CNTRL & WR_CMPL) != 0) {__NOP();}
   MDR_TIMER1->IE    = 0x00000002;              //(CNT==ARR)->IE
   NVIC->ISER[0]     = (1<<14);          // Global EN for IRQ14
   MDR_TIMER1->CNTRL = 0x00000001;                   //Timer1 ON
}
/*-------------------------------------------------------------*/
void Timer1_IRQHandler(void)
{
   MDR_TIMER1->CNTRL    = 0x00000000;               //Timer1 OFF
   MDR_TIMER1->STATUS  &= ~CNT_ARR_EVENT;            //IE FLAG=0
   if((MDR_PORTC->RXTX & VD0) == 0){led_on(0); led_off(1);}
   else {led_off(0); led_on(1);}
   MDR_TIMER1->CNT      = 0x00000000;             //Timer1 count
   MDR_TIMER1->IE       = 0x00000002;           //(CNT==ARR)->IE
   MDR_TIMER1->CNTRL    = 0x00000001;                //Timer1 ON
}/*-------------------------------------------------------------*/
```

Рис. 3.30. Программа для демонстрации работы счетчика таймера CT1 в режиме прерываний с выводом состояния на светодиодные индикаторы

Секция Define содержит определения трех констант:
VD0 – линия порта PORTC, к которой подключен светодиодный индикатор VD0;
WR_CMPL – флаг готовности регистров CNT, PSG и ARR к записи;
CNT_ARR_EVENT – флаг разрешения прерывания по событию совпадения CNT и ARR.

В секции прототипов описаны следующие функции: frq_init – настройка тактирования (HSE-генератор) микроконтроллера на частоту 8 МГц (8388608 Гц); Timer1_init – инициализация счетчика-таймера CT1 и контроллера прерываний; io_init – инициализация портов ввода-вывода; led_on и led_off – функции управления (включение и выключение соответственно) светодиодными индикаторами.

В секции main осуществляется инициализация тактового генератора, портов ввода-вывода, счетчика-таймера CT1 и контроллера прерываний. Далее следует бесконечный цикл while. Все остальные заданные действия будут выполнены в обработчике прерываний.

Функция Timer1_init() (см. рис. 3.30) предназначена для инициализации работы счетчика-таймера CT1 в режиме прерываний. Она включает в себя следующие этапы:

– включение тактирования CT1 (бит 14 в регистре PER_CLOCK);

– выбор частоты HCLK/128 (8388608 / 128= 65536 Гц) в качестве источника счетного сигнала (бит 24 и биты 2:0 в регистре TIM_CLOCK);

– останов CT1 (бит 0 в регистре TIMER1->CNTRL) и обнуление счетного регистра CNT;

– выбор коэффициента деления f/64 для предделителя (регистр TIMER1->PSG). Частота счетного сигнала на входе предделителя при этом составляет 65536 / 64= 1024 Гц. Запись в регистр PSG сопровождается проверкой бита WR_CMPL;

– выбор базы основного счетчика 1024 (регистр TIMER1->ARR). При частоте счетного сигнала 1024 Гц такая база соответствует периоду счета в 1 с. Запись в регистр ARR сопровождается проверкой бита WR_CMPL;

– выбор события «CNT==ARR» для генерирования сигнала «Запрос на прерывание» от счетчика-таймера CT1 (бит 1 регистра счетчика-таймера TIMER1->IE);

– разрешение прерывания IRQ14 в контроллере прерываний NVIC (бит 14 регистра ISER[0]);

– пуск счетчика-таймера CT1 (бит 0 в регистре TIMER1->CNTRL).

Обработчик прерывания Timer1_IRQHandler предназначен для периодического (один раз в секунду) выполнения следующих действий:

– останов счетчика-таймера CT1 (бит 0 в регистре состояния счетчика-таймера TIMER1->CNTRL);

– сброс флага прерывания CNT_ARR_EVENT в регистре состояния счетчика-таймера TIMER1->STATUS;

– организация поочередного включения светодиодных индикаторов путем анализа состояния бита VD0;

– обнуление счетного регистра CNT;

– выбор события «CNT==ARR» для генерирования сигнала «Запрос на прерывание» от счетчика-таймера CT1 (бит 1 управляющего регистра TIMER1->IE);

– пуск счетчика-таймера CT1 (бит 0 в регистре TIMER1->CNTRL).

В заключение отметим, что в главе 3 были подробно рассмотрены авторские программы (с 1-й по 13-ю). Эти программы охватывают достаточно широкий круг возможностей, которые предоставляют платформа Cortex-M3 и демонстрационно-отладочная плата фирмы «ПКК Миландр». Все рассмотренные программы носят учебный характер, и их обсуждение проводится в порядке от простого к более сложному. Так, первая программа касается элементарных действий по коммутации светодиодных индикаторов. В завершение рассмотрены вопросы программирования часов реального времени в зоне батарейного питания и счетчика-таймера в режиме прерываний. Тщательное изучение учебных программ с привлечением теоретического материала первых двух глав дает возможность детально разобраться в сути обсуждаемых вопросов.

Необходимо подчеркнуть два важных факта, обусловленные учебной направленностью данного материала.

Во-первых, все учебные программы написаны без использования стандартных библиотек ввода-вывода (SPL) или стандарта CMSIS. Во всех примерах использован традиционный подход системного программирования, заключающийся в прямом обращении к нужным регистрам микроконтроллера. Во всех случаях оказалось достаточно использовать два дополнительных файла: общий файла `MDR32Fx.h` с описанием ресурсов микроконтроллеров семейства Cortex-M3 и файл конфигурации `MDR32F9Qx_config.h` для настройки на конкретное периферийное оборудование.

Во-вторых, за рамками обсуждаемых вопросов остался обширный круг дополнительных возможностей, которые предоставляют платформа Cortex-M3 и демонстрационно-отладочная плата фирмы «ПКК Миландр». Так, например, мы ничего не говорили о программировании интерфесных контроллеров (USART, I2C/SPI, CAN, USB) или использовании операционной системы реального времени (ОСРВ). Эта обширная тема требует отдельного обсуждения. Упомянем лишь некоторые возможности, предоставлемые в распоряжение пользователя.

Библиотеки и протокольные стеки. Для того чтобы помочь разработчику в ускорении разработки кода программы, компания STM разработала библиотеку программ для микроконтроллеров STM32, которую можно свободно загрузить с веб-сайта компании. Библиотека программ поддерживает функции драйверов низкого уровня всех встроенных устройств ввода-вывода (УВВ). Пользователю предоставляются базовые составные блоки, из которых он может создать собственный проект. USB-контроллер является наиболее сложным устройством ввода-вывода. Для облегчения реализации наиболее распространенных USB-классов компания ST предлагает бесплатный набор для разработки USB-устройств. Этот набор, так же как библиотеку программ, можно скачать с веб-сайта компании STM. В комплект набора для разработки USB-устройств входят USB-библиотека и демонстрационные программы для классов HID, Mass Storage, Audio и Device Field Upgrade.

Операционные системы реального времени. Контроллер Cortex-M3 обладает достаточно большой вычислительной мощностью по сравнению с другими сопоставимыми по стоимости микроконтроллерами и разработан с учетом работы под управлением ОСРВ, занимающими небольшое место в памяти. При освоении этого микроконтроллера важно уделить ее изучению особое внимание. Использование ОСРВ дает преимущества более абстрактной разработки кода программы, более широких возможностей по повторному использованию программ, более простого управления проектом и более широких возможностей отладки. Использование ОСРВ также позволяет структурировать программу. Многие поставщики компиляторов предлагают свои собственные ОСРВ, однако наибольшую популярность среди операционных систем с открытым исходным кодом имеет *FreeRTOS*. Ее можно загрузить с сайта http://www.freertos.org. Коммерческая версия FreeRTOS называется *SafeRTOS*. Она протестирована на соответствие стандарту безопасности IEC 61508 и доступна на том же сайте.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Огородников И.Н. Микропроцессорная техника : учебник /И. Н. Огородников. Екатеринбург : УГТУ-УПИ, 2007. 380 с.

2. Ю Дж. Ядро Cortex-M3 компании ARM. Полное руководство /Дж. Ю. М. : Додэка-XXI, 2012. 552 с.

3. Мартин Т. Микроконтроллеры фирмы STMicroelectronics на базе ядра Cortex-M3. Серия STM32 /Т. Мартин. М. : Техносфера, 2009. 168 с.

4. Серия 1986ВЕ9х, К1986ВЕ9х, К1986ВЕ92QI, К1986ВЕ92QC, К1986ВЕ91Н4 высокопроизводительных 32-разрядных микроконтроллеров на базе процессорного ядра ARM Cortex-M3. Спецификация (версия 3.7.0 от 20.11.2014). М. : ЗАО «ПКК Миландр», 2014. 542 с. [Электронный ресурс]. Режим доступа: http://milandr.ru/uploads/Products/product_80 (файл spec_seriya_1986BE9x.pdf). Загл. с экрана.

5. Демонстрационно-отладочная плата 1986EvBrd_64. Техническое описание (версия 1.0 от 25.05.2010). М. : ЗАО «ПКК Миландр», 2010. 9 с. [Электронный ресурс]. Режим доступа: http://milandr.ru/uploads/Products/product_80 (файл 1986EvBrd_64_тех_описание.pdf). Загл. с экрана.

6. Отладочная плата 1986EvBrd_64 (Rev. 3). Схема электрическая принципиальная (версия от 24.04.2014). М. : ЗАО «ПКК Миландр», 2014. 2 с. [Электронный ресурс]. Режим доступа: http://milandr.ru/uploads/Products/product_80 (файл 1986EvBrd_64_Rev3.pdf). Загл. с экрана.

7. Голубцов М. Микроконтроллер MDR32F9Q2I. Часть 1. Первое знакомство с микроконтроллером и средствами разработки для него /М. Голубцов //Современная электроника. 2012. N 3. С. 18–21.

8. MDR32F9Qx Standard Peripherals Library (версия 1.3.0 от 27.11.2013). М. : ЗАО «ПКК Миландр», 2014. [Электронный ресурс]. Режим доступа: http://milandr.ru/uploads/Products/product_80 (файл MDR32F9Qx_Standard_Peripherals_Library.chm). Загл. с экрана.

9. Жидкокристаллический модуль MT-12864J. Спецификация (версия 1.5 от 29.05.2008). М. : ООО «МЭЛТ», 2008. 9 с. [Электронный ресурс]. Режим доступа: http://www.melt.com.ru/docs (файл MT-12864J.pdf). Загл. с экрана.

10. Вальпа О. Современные 32-разрядные ARM-микроконтроллеры серии STM32: часы реального времени RTC /О. Вальпа //Современная электроника. 2014. N 2. С. 22–26.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Учебное издание

Огородников Игорь Николаевич

Микропроцессорная техника: введение в Cortex-M3

Редактор *Л. Ю. Козяйчева*
Компьютерная верстка *И. Н. Огородникова*

План выпуска 2015 г. Подписано в печать 04.09.2015. Формат 60×84 1/16.
Бумага писчая. Плоская печать. Усл. печ. л. 6,7.
Уч.-изд. л. 6,73. Тираж 100 экз. Заказ 346.

Издательство Уральского университета
Редакционно-издательский отдел ИПЦ УрФУ
620049, Екатеринбург, ул. С. Ковалевской, 5
Тел. +7 (343) 375-48-25, 375-46-85, 374-19-41
E-mail: rio@urfu.ru

Отпечатано в Издательско-полиграфическом центре УрФУ
620075, Екатеринбург, ул. Тургенева, 4
Тел.: +7 (343) 350-56-64, 350-90-13
Факс: +7 (343) 358-93-06
E-mail: press.info@mail.ru