
1.Введение

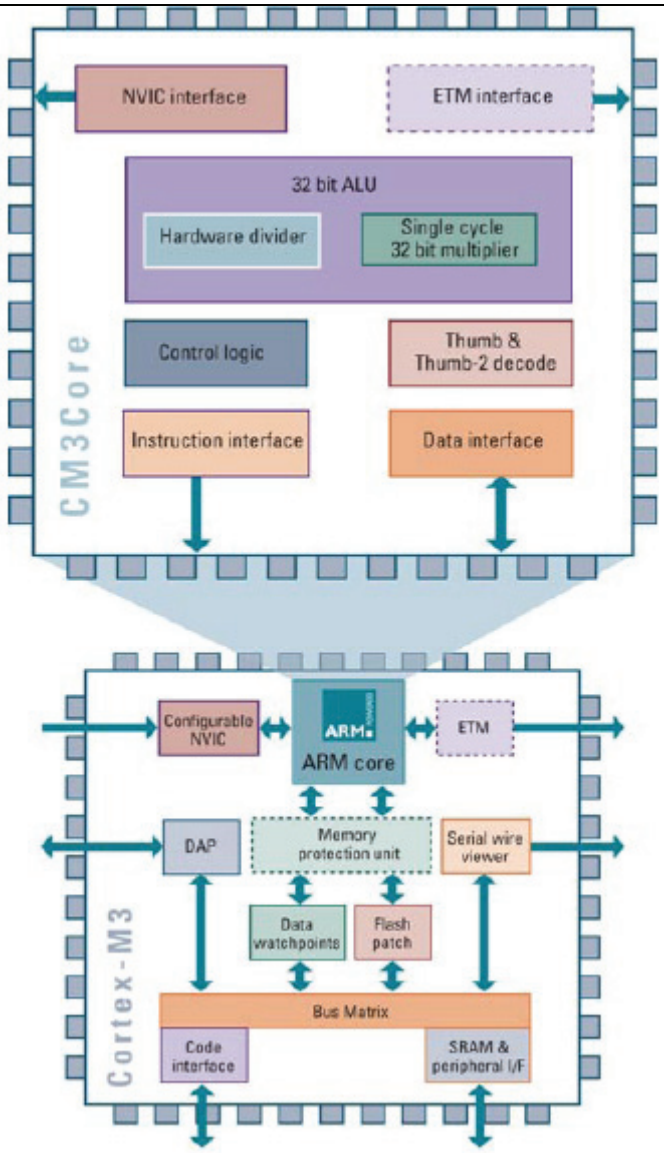
На протяжении последних шести или семи лет одним из главных направлений в области разработки микроконтроллеров являлась адаптация ARM7 и ARM9 как ядер для микроконтроллеров общего применения. На сегодняшний день доступно около 240 ARM-микроконтроллеров от многочисленных производителей. Компания

ST Microelectronics выпустила первые микроконтроллеры на базе нового ядра ARM Cortex-M3, STM32. Эти устройства устанавливают новые стандарты производительности и цены, способности выполнения алгоритмов управления жесткого реального времени при низком энергопотреблении.

## 1.1 Так что же такое Cortex?

Семейство ARM Cortex – это новое поколение процессоров со стандартизованной архитектурой для решения широкого круга технологических задач. В отличие от других ядер ARM, семейство Cortex представляет собой законченное процессорное ядро со стандартным ЦПУ и системной архитектурой. Семейство Cortex имеет три основных профиля: профиль A для высокопроизводительных приложений, R – для приложений реального времени, M – для бюджетных приложений. В STM32 используется профиль Cortex-M3, разработанный специально для систем, сочетающих в себе высокую производительность и низкое энергопотребление. Цена этого семейства достаточно низка, чтобы конкурировать с 8 и 16-разрядными микроконтроллерами.

Несмотря на то, что ядра ARM7 и ARM9 были успешно интегрированы в стандартные микроконтроллеры, в них прослеживается наследственность от Систем на Кристалле (SoC). Особенно это проявляется при обработке исключительных ситуаций и прерываний, так как каждый производитель микроконтроллеров разрабатывал свое собственное решение. Cortex-M3 – это стандартизованное микроконтроллерное ядро, выходящее за пределы ЦПУ, представляющее собой законченное сердце микроконтроллера (включает систему прерываний, системный таймер, систему отладки и карту памяти). 4 Гбайта адресного пространства Cortex-M3 разделено на строго определенные области для кода приложения, SRAM, периферийных и системных устройств. В отличие от ARM7, Cortex-M3 построен на базе Гарвардской архитектуры памяти и, следовательно, содержит множество шин, позволяющих выполнять операции параллельно, увеличивая общую производительность. По сравнению с ранними архитектурами ARM, семейство Cortex может осуществлять доступ к невыровненным данным, что гарантирует более эффективное использование встроенной SRAM. Семейство Cortex также поддерживает установку и сброс битов в пределах 1 Мбайта памяти с помощью метода, называемого «сегментация битов», при помощи которого осуществляется эффективный доступ к регистрам периферийных устройств и флагам, размещенным в SRAM, без привлечения полноценных булевых операций.




**Сердце STM32 – это ядро Cortex-M3. Cortex-M3 – это стандартизованный микроконтроллер, включающий 32-битное ЦПУ, систему шин, модуль вложенных прерываний, систему отладки и стандартно размещенную память.**

Перевод к рисунку:

*NVIC interface – NVIC интерфейс, ETM interface – ETM интерфейс, 32 bit ALU – 32-битное АЛУ, Hardware divider – Аппаратный делитель, Single cycle 32 bit multiplier – 32 битный умножитель за один цикл, Control logic – Управляющая логика, Thumb & Thumb-2 decode – декодер инструкций Thumb и Thumb-2, Instruction interface – Интерфейс команд, Data interface – Интерфейс данных.*

*Configurable NVIC – Конфигурируемый NVIC, ARM Core – Ядро ARM, Memory protection unit – Модуль защиты памяти, Serial wire viewer - , Data watchpoints – Точки просмотре данных, Flash patch - , Bus Matrix – Шинная матрица, Code interface – Интерфейс команд, SRAM & peripheral I/F – Интерфейс SRAM и периферийных устройств.*

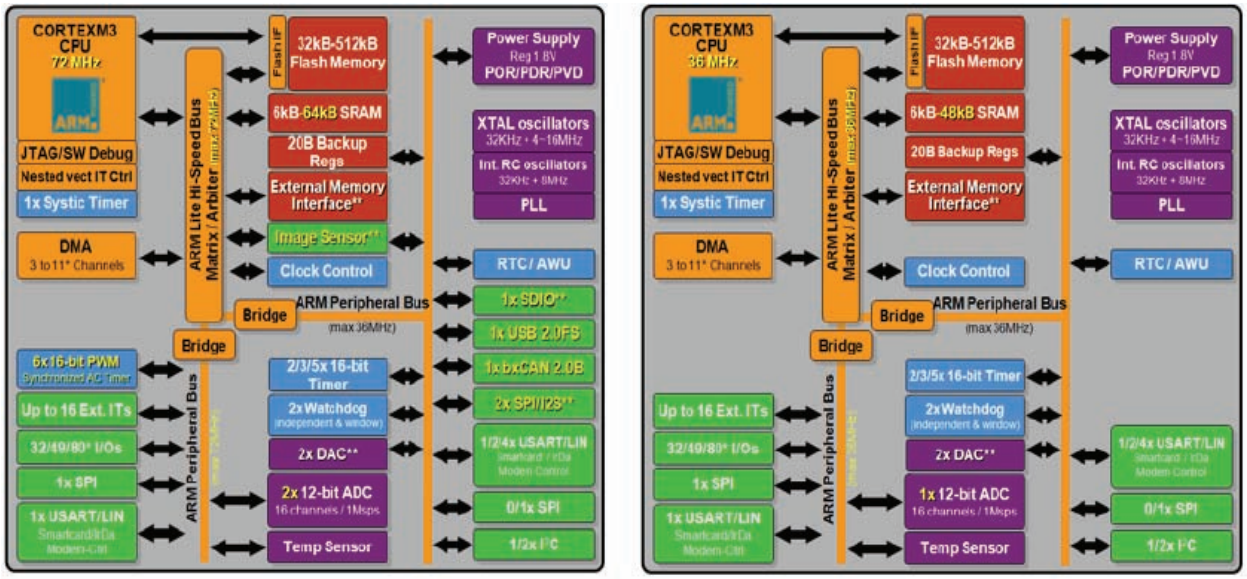
Одним из ключевых компонентов Cortex-M3 является Контроллер Вложенных Векторных Прерываний (NVIC). NVIC имеет стандартизованную структуру прерываний для всех микроконтроллеров на основе Cortex и отличается непревзойденными возможностями обработки прерываний. NVIC содержит

вектора прерываний для 240 источников, причем каждому источнику прерывания можно индивидуально назначить приоритет. Контроллер NVIC разрабатывался для очень быстрой обработки прерываний. С момента получения прерывания до начала выполнения первой команды обработчика прерывания затрачивается только двенадцать циклов тактового сигнала. Это достигается частично за счет автоматического размещения информации в стеке, осуществляемого внутри ЦПУ микроконтроллера. В случае, когда прерывания следуют впритык друг за другом, NVIC использует метод «постановки в очередь», позволяющий осуществлять обработку каждого из последующих прерываний всего за шесть циклов тактового сигнала. На этапе обращения к стеку, прерывание с более высоким приоритетом может получить преимущество перед прерыванием с низким приоритетом, что не влечет за собой затрат дополнительных циклов ЦПУ. Логика прерывания жестко сопряжена с режимами сниженного энергопотребления. ЦПУ может автоматически переходить в режим низкого энергопотребления при выходе из прерывания. То есть, ядро остается в «спящем» режиме до возникновения следующей исключительной ситуации.

Несмотря на то, что ядро Cortex-M3 разработано для бюджетных решений, оно является 32-битным и поддерживает два режима работы: Поток и Обработчик, которые могут конфигурироваться со своими собственными стеками. Все это позволяет реализовывать устройства со сложным программным обеспечением и поддержкой операционных систем реального времени. Cortex содержит 24-битный таймер с автоматической перезагрузкой, предназначенный для генерации периодических прерываний ядра RTOS. В отличие от ARM7 и ARM9, работающих с двумя наборами команд (32-битный ARM и 16-битный Thumb), семейство Cortex поддерживает одну систему команд, ARM Thumb-2. Эта система содержит и 16- и 32-битные команды, сочетая в себе производительность 32-битного ARM с плотностью кода 16-битного Thumb. Thumb-2 – это богатая система команд, разработанная для работы с Си/Си++ компиляторами. Это означает, что приложение под Cortex может быть полностью написано на Си.

## 1.2 Взгляд на STM32

Компания ST производит уже четыре семейства микроконтроллеров на базе ядер ARM7 и ARM9, но STM32 – это серьезный шаг вперед в развитии микроконтроллеров. С ценой один Евро при серийных количествах, STM32 ставит под вопрос дальнейшее существование 8-битных микроконтроллеров… Первоначально STM32 были представлены в четырнадцати различных вариантах. Они разделялись на две группы: Performance с тактовой частотой ЦПУ до 72 МГц и Access – до 36 МГц. Микроконтроллеры обоих групп, аппаратно и программно совместимые друг с другом, содержат до 128K Flash и 20K SRAM. С момента своего появления, семейство STM32 расширилось и на сегодняшний день содержит микроконтроллеры с еще большими объемами RAM, FLASH и более сложными периферийными устройствами.




**STM32 делятся на две отдельные группы. Группа Performance, с частотой ядра до 72 МГц, полным набором периферийных устройств и группа Access, с частотой ядра 36 МГц и урезанным набором периферийных устройств.**

*Перевод к рисунку: CPU – ЦПУ, 72MHz – 72 МГц, Nested vect IT Ctrl – Контроллер Прерываний, 1x Systic Timer – Системный Таймер, 3 to 11 Channels – от 3 до 11 каналов, 6x16-bit PWM – 6x16-бит ШИМ, Up to 16 Ext. Its – До 16 внешн. прерываний, Smartcard/IrDA – И/Ф Смарткарт/IrDA, Modem-Ctrl – Контроллер Модема, ARM Lite Hi-Speed Bus – Высокочастотная Шина ARM Lite, Matrix / Arbiter – Матрица / Арбитр, max 72 MHz – до 72 МГц, Bidge - Мост, ARM Peripheral Bus – Периферийная Шина ARM, Flash IF – И/Ф Flash, 32kB-512kB Flash Memory – 32КБ – 512КБ Flash Памяти, 6kB-64kB SRAM – 6КБ-64КБ SRAM, 20B Backup Regs – 20Б Регистров Резервир., External Memory Interface – Интерфейс Внешней Памяти, Image Sensor – Датчик Изображения, Clock Control – Система Синхронизации, max 36 MHz – до 36 МГц, 16-bit Timer – 16-битный Таймер, 2xWatchdog (independent&window) – 2 Сторож. Таймера (независимый и оконный), 2x DAC – 2x ЦАП, 2x12-bit ADC – 2x 12-бит АЦП, 16 channels / 1 MSPS – 16 каналов / 1 MSPS, Temp Sensor - Термодатчик, Power Supply – Источник Питания, Reg 1.8V – Рег. 1.8 В, XTAL oscillators – Внешн. осцилляторы, 32KHz + 4-16MHz – 32кГц + 4-16МГц, Int. RC oscillators – Внутр. осцилляторы, 32KHz + 8MHz – 32кГц + 8МГц, PLL - ФАПЧ, 36 MHz – 36 МГц, max 36MHz – до 36 МГц.*

### 1.2.1 Многофункциональные Периферийные Устройства

На первый взгляд, набор периферийных устройств выглядит как у обычного небольшого микроконтроллера. В его состав входит двойной АЦП, таймера общего применения, I2C, SPI, CAN, USB и часы реального времени. Однако каждое из этих периферийных устройств имеет много особенностей. Например, 12-битный АЦП содержит встроенный температурный датчик и поддерживает множество режимов преобразования, а устройства с двойным АЦП могут перестраивать оба АЦП сразу на работу в одном из девяти режимов преобразования. Точно так же, каждый из четырех таймеров содержит четыре устройства захвата/сравнения, и каждый таймер может объединяться с другими для образования сложных массивов таймеров. Таймер с расширенными функциями предназначен для приложений управления приводами, он содержит шесть комплементарных выходов ШИМ с программируемым «мертвым временем» и входной линией экстренного отключения, которая переводит выводы ШИМ в предопределенное безопасное состояние. Модуль SPI содержит аппаратный генератор избыточного циклического кода (CRC) для 8- и 16-битных слов, использующийся при подключении к SD и MMC картам памяти.

STM32 содержит модуль прямого доступа к памяти (DMA) на семь каналов, что вообще необычно для небольших микроконтроллеров. Каждый канал может быть использован для передачи 8/16 или 32-битных данных в и из любого регистра периферийных устройств. Каждое из периферийных устройств, в зависимости от задач, может быть либо отправителем, либо получателем данных от DMA контроллера. Встроенные арбитр шины и матрица шин минимизируют арбитраж между ЦПУ и DMA. Все это свидетельствует о гибкости DMA, о том, что он прост в использовании и действительно автоматизирует передачу данных в пределах микроконтроллера.

Если попытаться охарактеризовать STM32 в двух словах, то это малопотребляющий, но в то же время высокопроизводительный микроконтроллер. Он может работать от источника напряжения 2 В с тактовой частотой ядра 72 МГц с включенныйми всеми периферийными устройствами и при этом потреблять только 36 мА. В режиме сниженного энергопотребления STM32 потребляет 2 мкА. Встроенный RC-осциллятор на 8 МГц позволяет быстро выходить из режимов сниженного энергопотребления пока внешний осциллятор стабилизируется. Быстрый вход и выход из режимов сниженного потребления сокращает общую энергозатратность системы.

### 1.2.2 Надежность

Многие современные устройства помимо обеспечения высокой производительности и функциональности, должны удовлетворять особым требованиям обеспечения безопасности. STM32 имеет несколько аппаратных особенностей для поддержания целостности системы. В их число входит детектор падения напряжения питания, система безопасности системы синхронизации и два отдельных сторожевых таймера. Первый сторожевой таймер оконного типа.

Он должен обновляться в определенный временной интервал. Если обновление произойдет раньше или позже, сторожевой таймер сработает, т.е. сгенерирует прерывание. Второй сторожевой таймер является независимым, имеет свой собственный внешний осциллятор, отдельный от основной системы синхронизации. Система синхронизации поддерживает функции детектирования неисправностей основного внешнего осциллятора, в случае которых происходит переключение на встроенный 8 МГц RC-осциллятор.

### 1.2.3 Безопасность

Одним из требований к современным устройствам является обеспечение сохранности программного кода от несанкционированного доступа. Для Flash памяти STM32 может быть установлена защита от чтения через отладочный порт. Когда защита от чтения включена, Flash память также защищена от записи, чтобы предотвратить возможность размещения некорректного кода в таблице векторов прерываний. Микроконтроллеры STM32 также содержат часы реального времени и небольшую обаласть SRAM с питанием от батареи. Содержимое этой области автоматически стирается по прерыванию от модуля предотвращения вмешательства в устройство.

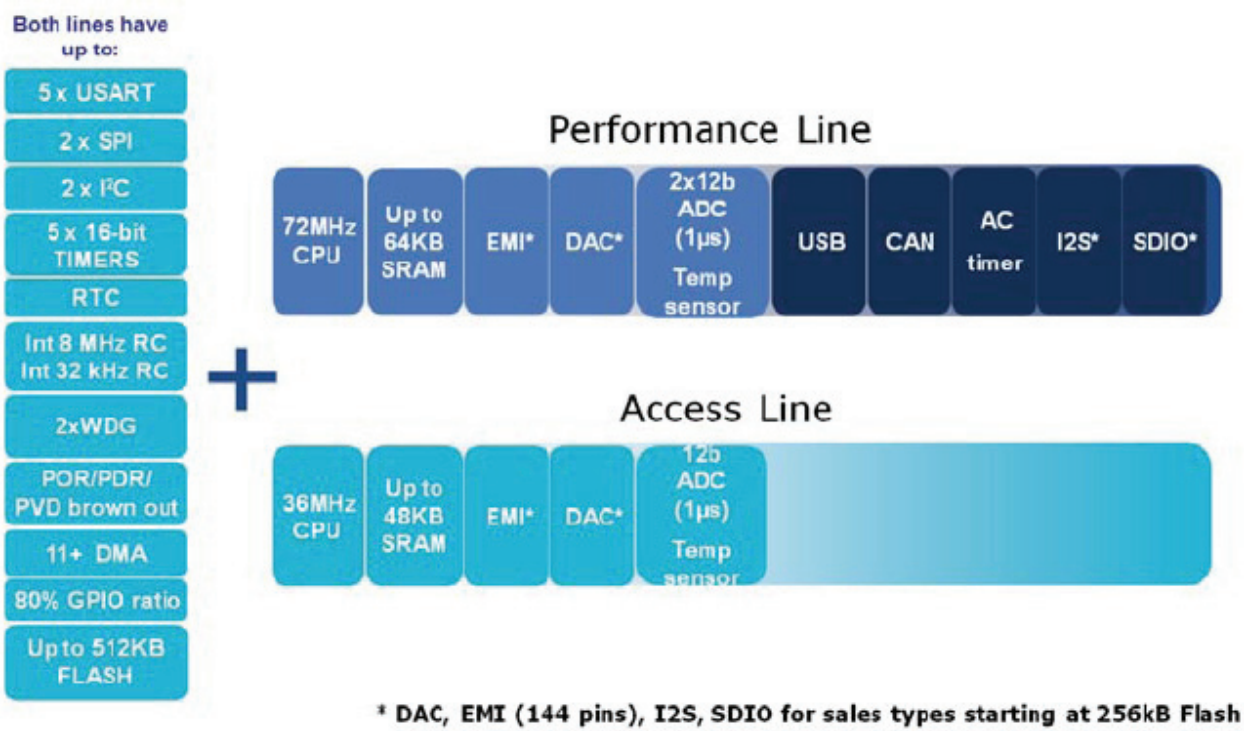
### 1.2.4 Разработка Программного Обеспечения

Если вы уже используете ARM-микроконтроллеры, есть шанс, что ваши отладочные средства поддерживают систему команд Thumb-2 и ядро Cortex. В худшем случае придется только обновить программное обеспечение. Компания ST предоставляет библиотеку драйверов периферийных устройств, библиотеку для USB (как ANSI Си библиотеку) и исходные коды, совместимые с предыдущими библиотеками для микроконтроллеров STR7 и STR9. Уже доступны адаптированные версии этих библиотек для популярных компиляторов. Кроме этого, для семейства Cortex доступно множество открытых и коммерческих RTOS и связующего ПО (TCP/IP, файловые системы и т.д.). В Cortex-M3 используется совершенно новая система отладки под названием CoreSight. Доступ к системе CoreSight осуществляется через порт Dedug Access, который поддерживает соединение либо через стандартный JTAG, либо через последовательный двухвыводной интерфейс serial wire. Помимо управления ходом отладки, система CoreSight отвечает за точки просмотра данных и трассирову. Трассировщик передает определенную информацию о приложении в среду отладки, которая может быть использована во время тестирования программного обеспечения.

### 1.2.5 STM32 Performance Line и Access Line

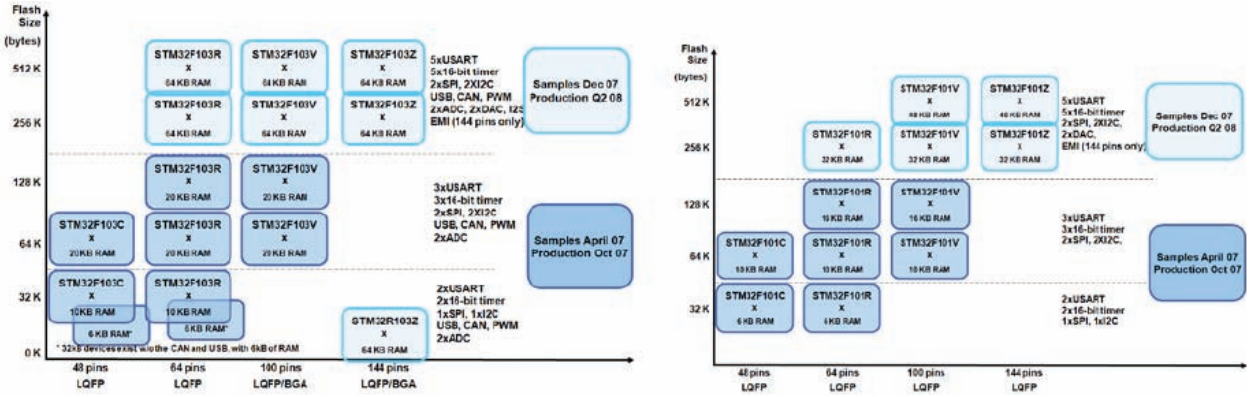
Семейство STM32 разделяется на две группы: Performance и Access. Группа Performance содержит полный набор периферийных устройств и работает на тактовой частоте до 72 МГц. Группа Access содержит сокращенный набор

периферийных устройств и работает на чатотах до 32 МГц. Важно отметить, что типы корпусов и расположение выводов микрокотроллеров групп Access и Performance совпадают. Это позволяет взаимозаменять различные версии STM32 в устройстве без переработки схемы приложения.




*Перевод к рисунку: Both lines have up to – Обе группы содержат до, 5x16-bit TIMERS – 5 16-битных Таймеров, Int 8 MHz RC – Внутр. 8МГЦ RC, Int 32 kHz RC – Внутр 32 кГ RC, 2xWDG – 2 Сторожевых Таймера, 80% GPIO ratio – 80% GPIO, Up to 512KB FLASH – до 512 КБ FLASH, 72MHz CPU – 72 МГц ЦПУ, Up to 64KB SRAM – до 64 КБ SRAM, DAC – ЦАП, ADC (1мюs) – АЦП (1 мкс), Temp sensor - Термодатчик, AC timer – Таймер AC, 36 MHz CPU – 36 МГц ЦПУ, Up to 48KB SRAM – до 48 КБ SRAM,*

*DAC, EMI (144 pins), I2S, SDIO for sales types starting at 256 kB Flash – ЦАП, Интрефейс Внешней Памяти, I2S, SDIO в микроконтроллерах с объемом Flash от 256 КБ.*




(данные обновить?)

# 2. Обзор Cortex

Как мы видели в предыдущей главе, процессоры Cortex – это новое поколение встраиваемых ядер ARM. В какой-то степени они отличаются от предыдущих ЦПУ ARM в том, что это уже полное процессорное ядро, состоящее из ЦПУ Cortex и окружающего его набора системных периферийных устройств, реализующее в себе сердце встраиваемой системы. Так как существует большое количество типов встраиваемых систем, доступны процессоры Cortex для различных профилей приложений. На профиль указывает буква в названии. На сегодняшний момент существуют следующие три профиля:

**Cortex-A**, процессоры для приложений со сложными операционными системами ОС и алгоритмами пользователя.
Поддерживает системы команд ARM, Thumb и Thumb-2.

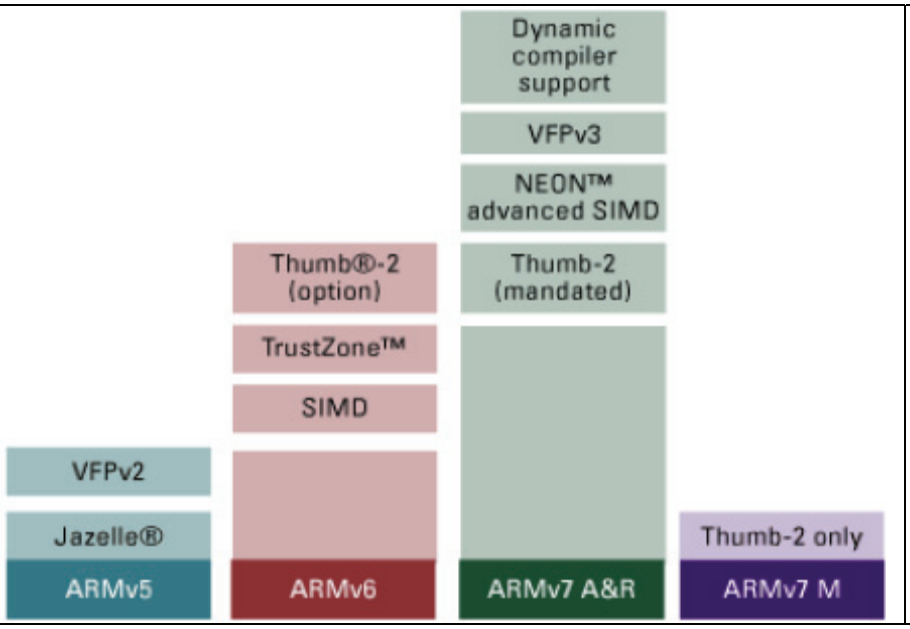
**Cortex-R**, профиль для систем реального времени.
Поддерживает системы команд ARM, Thumb и Thumb-2.

**Cortex-M**, профиль для микроконтроллеров, оптимизированных по цене.
Поддерживает только систему команд Thumb-2.

Число в конце названия Cortex обозначает соответствующий уровень производительности: от 1 - для минимального уровня, до 8 - для максимального. В настоящий момент, уровень производительности 3 является самым высоким уровнем, доступным для профиля микроконтроллеров. Микроконтроллеры STM32 построены на базе процессоров Cortex-M3.

## 2.1 Ревизия Архитектуры ARM

Компания ARM маркирует каждый из своих процессоров в соответствии с ревизией архитектуры (ARMV6, ARMV7 и т.д.). Cortex-M3 имеет ревизию архитектуры ARMV7 M.




**Процессоры Cortex-M3 построены на базе архитектуры ARMV7 и поддерживают систему команд Thumb-2**

*Перевод к рисунку: Dynamic compiler support – Поддержка динамической компиляции, only –только.*

Документация для Cortex-M3 представлена Техническим Справочным Руководством Cortex-M3 и Справочным Руководством Архитектуры ARMV7 M. Оба документа доступны на сайте компании ARM: www.arm.com

## 2.2 Процессоры Cortex и ЦПУ Cortex

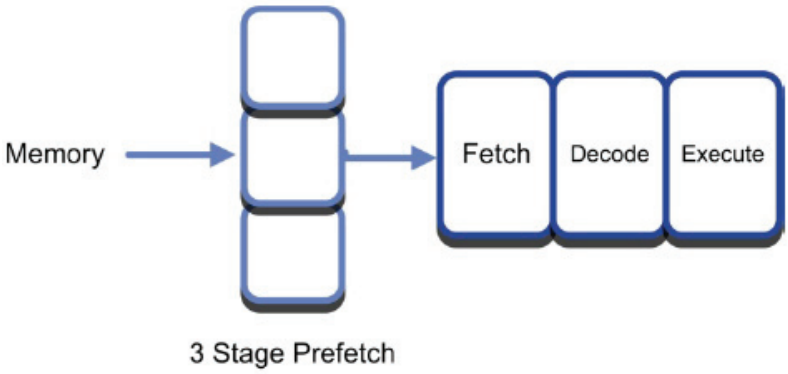
На протяжении оставшейся части книги определения «процессор Cortex» и «ЦПУ Cortex» будут использоваться для различия между полноценным встраиваемым ядром Cortex и внутренним RISC ЦПУ. В следующей главе мы рассмотрим ключевые отличительные особенности ЦПУ Cortex, а затем и системных периферийных устройств процессоров Cortex.

## 2.3 ЦПУ Cortex

Сердцем процессоров Cortex является 32-битное RISC ЦПУ. Это ЦПУ имеет упрощенную версию программной модели ARM7/9, но расширенную систему команд с хорошей поддержкой целочисленной арифметики, лучшей битовой манипуляцией и производительностью с более «жестким» реальным временем.

### 2.3.1 Конвейер

ЦПУ Cortex способно выполнять большую часть команд за один цикл. Как в ARM7 и ARM9 это достигается с помощью трехступенчатого конвейера.



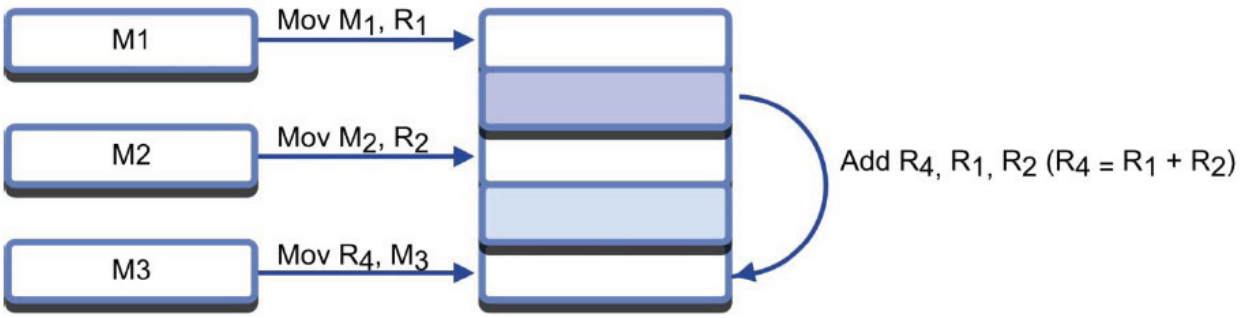

**Как в ARM7 и ARM9, Cortex-M3 содержит трехступенчатый конвейер. Однако Cortex-M3 поддерживает прогнозирование ветвлений для минимизации числа сбросов конвейера.**

*Перевод к рисунку: Memory - Память, 3 Stage Prefetch – Трехуровневая предвыборка, Fetch - Выборка, Decode - Декодирование, Execute - Исполнение.*

Пока первая команда выполняется, вторая декодируется, и третья извлекается из памяти. Это работает очень хорошо при линейном коде, но когда встречается ветвление, конвейер сбрасывается и загружается заново перед тем, как программный код продолжит выполняться. В ARM7 и ARM9 ветвления очень затратные с точки зрения производительности. В ЦПУ Cortex трехступенчатый конвейер дополнен функцией прогнозирования ветвлений. Это значит, что когда встречается команда условного перехода, осуществляется извлечение следующих предполагаемых команд. Таким образом, оба варианта условного перехода доступны для исполнения, что не влечет за собой снижение производительности. Худший случай – косвенный переход, при котором предполагаемый переход не может быть определен и единственным выходом из ситуации является сброс конвейера. Так как именно конвейер определяет полную производительность ЦПУ Cortex, дополнительные рассуждения относительно кода приложения были бы лишними.

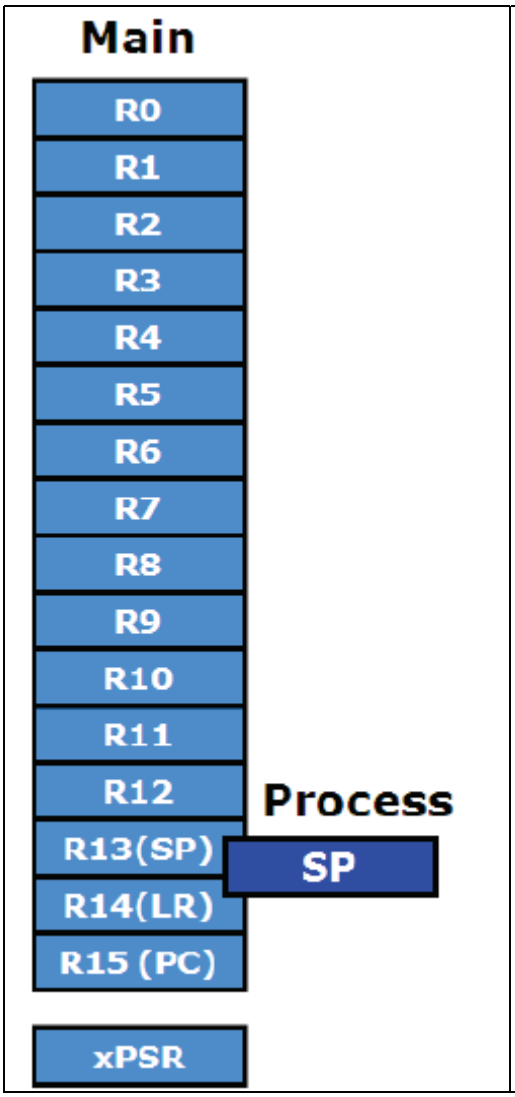
### 2.3.2 Программная Модель

ЦПУ Cortex – это RISC процессор с архитектурой load-store. Перед выполнением команд обработки данных, операнды должны быть загружены в центральный регистровый файл. Операции с данными должны выполняться в этих регистрах и результаты сохраняться в памяти.



**Cortex-M3 имеет архитектуру load-store. Все данные должны передаваться в центральный регистровый файл перед обработкой в соответствии с командой.**

Следовательно, вся программная деятельность сосредотачивается на регистровом файле ЦПУ. Этот регистровый файл состоит из шестнадцати 32-битных регистров. Регистры R0-R12 являются простыми регистрами, которые могут быть использованы для хранения переменных. Регистры R13-R15 выполняют специализированные функции внутри ЦПУ Cortex. Регистр R13 используется как указатель стека. Это сгруппированный регистр, что позволяет ЦПУ Cortex иметь отдельные стеки для двух режимов работы. Обычно это используется RTOS, которая выполняет свой «системный» код в защищенном режиме. В ЦПУ Cortex эти два стека называются основной стек и рабочий стек. Следующий регистр, R14, называется регистром связи. Он используется для хранения адреса возврата при переходе в процедуру. Это позволяет ЦПУ Cortex осуществлять быстрый вход и выход из процедуры. Если программный код содержит несколько уровней вложенных подпрограмм, компилятор будет автоматически сохранять значение R14 в стеке при переходе на следующий уровень. Последний регистр, R15, является счетчиком команд (PC); так как он входит в состав регистрового файла, значение R15 можно считывать и изменять как в случае с любым другим регистром.

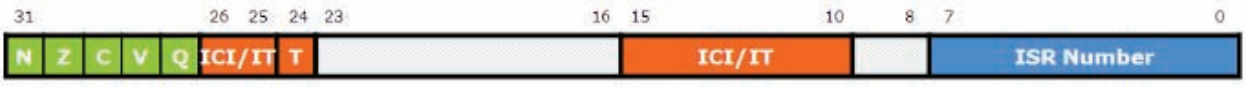



**Cortex-M3 содержит регистровый файл ЦПУ из 16-ти 32-битных регистров. Как в ARM7/9, R13 является указателем стека. R14 – регистр связи и R15 – счетчик команд. R13 сгруппирован, что позволяет Cortex-M3 работать с двумя стеками: рабочим и основным.**

*Перевод к рисунку: Main - Основной, Process - Рабочий.*

2.3.2.1 xPSR

В дополнение к регистровому файлу существует отдельный регистр под названием Program Status Register. Он не входит в состав главного регистрового файла и доступен только через две специальные команды. Значения в битовых областях регистра xPSR влияют на работу ЦПУ Cortex.




**Области Program Status Register, влияющие на исполнение команд. Этот регистр также называют Application, Execution и Interrupt Status Register.**

Регистр xPSR может быть доступен также через три специальных псевдоимени, позволяющие обращаться к определенным областям xPSR. Пять верхних битов

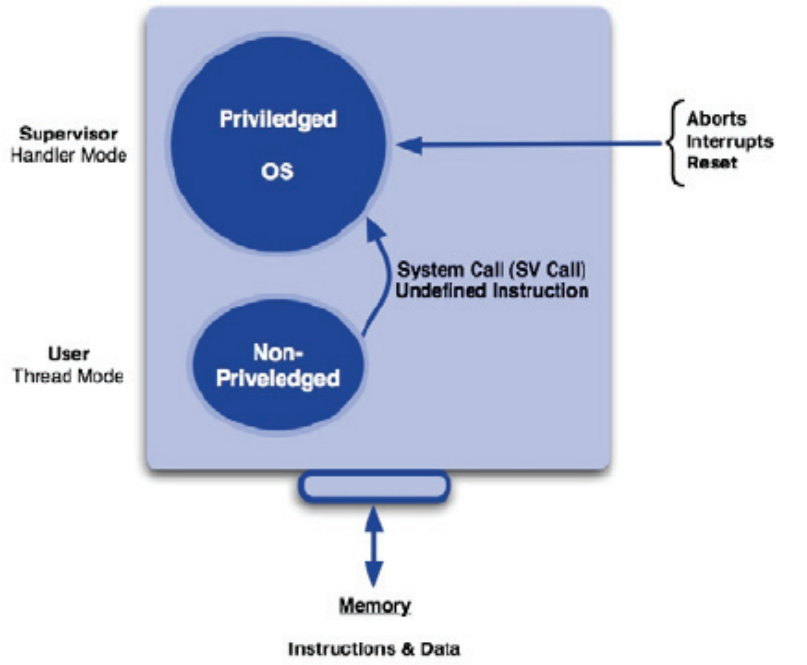
являются флагами кода условия и доступны под псевдоименем Application Program Status Register. Первые четыре флага кода условия, N, Z, C, V (Negative, Zero, Carry и Overflow) устанавливаются и сбрасываются в зависимости от результата выполнения команды. Бит Q используется DSP-командами, чтобы показать, что переменная достигла своего максимального или минимального значения. Как и в 32-битной системе команд ARM, команды Thumb-2 выполняются только, если код условия команды соответствует состоянию флагов регистра Application Program Status Register. Если соответствия нет, команд проходит через конвейер как пустая команд NOP. Это обеспечивает «плавное течение» команд через конвейер и минимизирует число сбросов конвейера. В ЦПУ Cortex эта технология разширена при помощи регистра Execution Program Status Register. Это псевдоимя битов 26-8 регистра xPSR. Он включает три области: “If then”, «команда продолжаемая после прерывания» и «команда Thumb». Система команд Thumb-2 поддерживает эффективный метод выполнения блока команд “if then”. Если условие выполняется, устанавливается значение в области “if then”, которое заставляет ЦПУ выполнять до четырех следующих команд. Если условие не выполняется, эти команды проходят через конвейер как NOP. Таким образом, типичная строка на языке Си будет компилироваться как показано ниже:

```
If(r0 == 0)
        CMP r0, #0    сравниваем r0 с 0
        ITTEE EQ      если “истина”, выполняем следующие две команды
Then r0 = *r1 + 2;
        LDR r0, [r1]  загружаем содержимое из памяти в r0
        ADDr0, #2     прибавляем 2
```

Несмотря на то, что большинство команд Thumb-2 выполняются за один цикл, некоторые (такие как команды загрузки и сохранения) требуют несколько циклов. Таким образом, для того, чтобы ЦПУ Cortex могло иметь детерменированное время реакции на прерывание, такие команды должны быть прерываемыми. Когда команда прерывается, в области «команда продолжаемая после прерывания» сохраняется число последующих регистров, которые нужно обработать согласно команде. Соответственно, после завершения обработки прерывания, многоцикловая команда может продолжать исполнение. Последняя область, «команда Thumb», унаследована из предыдущих ЦПУ ARM. Эта облать показывает к какой системе принадлежит выполняемая ЦПУ команда: ARM или Thumb. В Cortex-M3 этот бит всегда установлен в единицу. В заключение, область статуса прерывания содержит информацию о запросах прерывания, которые были отложены.

2.3.3 Режимы Работы ЦПУ

Несмотря на то, что Cortex разрабатывался как процессор с малым числом вентилей, простым в применении, он поддерживает использование операционных систем реального времени. Cortex имеет режима работы: Поток и Обработчик. ЦПУ будет работать в режиме Поток пока оно исполняет код в непрерываемом фоновом режиме и переключится в режим Обработчик при обработке исключительных ситуаций. Кроме того, ЦПУ Cortex может исполнять код в привелегированном или непривелегированном режиме. В привелегированном режиме ЦПУ имеет доступ ко всей системе команд. В непривелегированном режиме некоторые команды недоступны (такие как MRS и MSR, через которые возможен доступ к xPSR и его псевдоименам). Также, запрещен доступ к большинству системных регистров управления Cortex. Основной стек (R13) может использоваться в обоих режимах: Поток и Обработчик. Режим Обработчик может быть сконфигурирован на использование рабочего стека (сгруппированный с R13 регистр).




**Cortex-M3 может использоваться в простом режиме, но он также разработан поддерживать операционные системы реального времени. Он имеет режимы Обработчик и Поток, которые могут быть сконфигурированы для работы с основным и рабочим стеком и иметь привелегированный доступ с системным регистрам управления.**

*Перевод к рисунку: Supervisor - Супервизор, Handler Mode – Режим Обработчик, User - Пользователь, Thread Mode – Режим Поток, Memory - Память, Instructions & Data – Команды и Данные, Priviledged - Привелегированный, OS - ОС, Non-Priveledged - Непривелегированный, System Call (SV Call) – Системный Вызов (SV Call), Undefined Instruction – Неопределенная Команда, Aborts – Преждевременные Прекращения, Interrupts - Прерывания, Reset - Сброс.*

| | | Operations (privilege out of reset) | Stacks (Main out of reset) |
|---|---|---|---|
| Modes (Thread out of reset) | **Handler**<br>- An exception is being processed | Privileged execution<br>Full control | Main Stack Used by OS and Exceptions |
| | **Thread**<br>- No exception is being processed<br>- Normal code is executing | Privileged/Unprivileged | Main/Process |

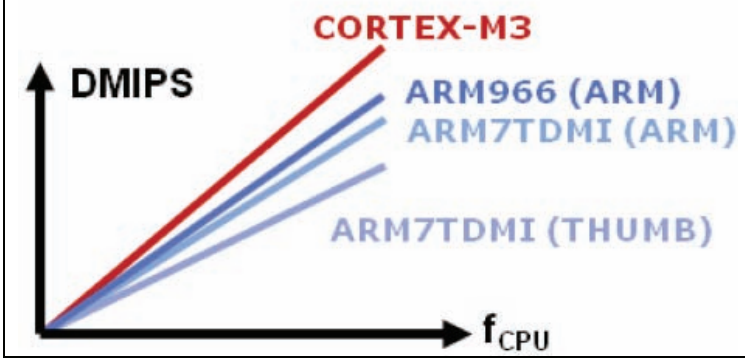
*Перевод к таблице: Operations (privilege out of reset) – Функционирование (привелегированное после сброса), Stacks (Main out of reset) – Стек (Основной после сброса), Modes (Thread out of reset) – Режимы (Поток после сброса), Handler - Обработчик, An exception is being processed – Обрабатывается исключение, Thread - Поток, No exception is being processed – Обрабатывается не исключение, Normal code is executing – Выполняется обычный код, Privileged execution – Привелегированное исполнение, Full control – Полный контроль, Privileged/Unprivileged – Привелегированное/Непривелегированное, Main Stack Used by OS and Exceptions – Основной стек используется ОС и исключениями, Main/Process – Основной/Рабочий.*

После сброса процессор Cortex работает в простой конфигурации. И Поток и Обработчик выполняются в привилегированном режиме, так что нет ограничений по доступу к ресурсам процессора. И Поток и Обработчик используют основной стек. Для запуска процессора Cortex нужно только сконфигурировать вектор сброса и начальный адрес стека, после чего может исполняться код приложения пользователя. Однако если вы используете RTOS или разрабатываете приложение с особыми требованиями к обеспечению безопасности, можно использовать расширенный режим. При этом исключения и RTOS обрабатываются в режиме Обработчик с привилегированным доступом и с использованием основного стека, а код приложения исполняется в режиме Поток с непривилегированным доступом и использует рабочий стек. Таким образом, системный код и код приложения разделяются и ошибки кода приложения не повлияют на работу RTOS.

### 2.3.4 Система Команд Thumb-2

ЦПУ ARM7 и ARM9 могут исполнять две системы команд: 32-битную систему команд ARM и 16-битную систему команд Thumb. Это позволяет разработчику оптимизировать программу, выбирая нужную систему команд под определенную процедуру: 32-битные команды для высокой скорости и 16-битные для высокой плотности кода. Система команд Thumb-2 имеет преимущество по плотности

кода над 32-битной ARM на 26% и по производительности над 16-битной Thumb на 25%. Thumb-2 содержит несколько улучшенных команд умножения, исполняемых за один цикл и команду аппаратного деления, требующую от 2 до 7 циклов на исполнение.




**Производительность процессора Cortex оценивается как 1.2 DMIPS/МГц, что эквивалентно 1.2 циклам на команду.**

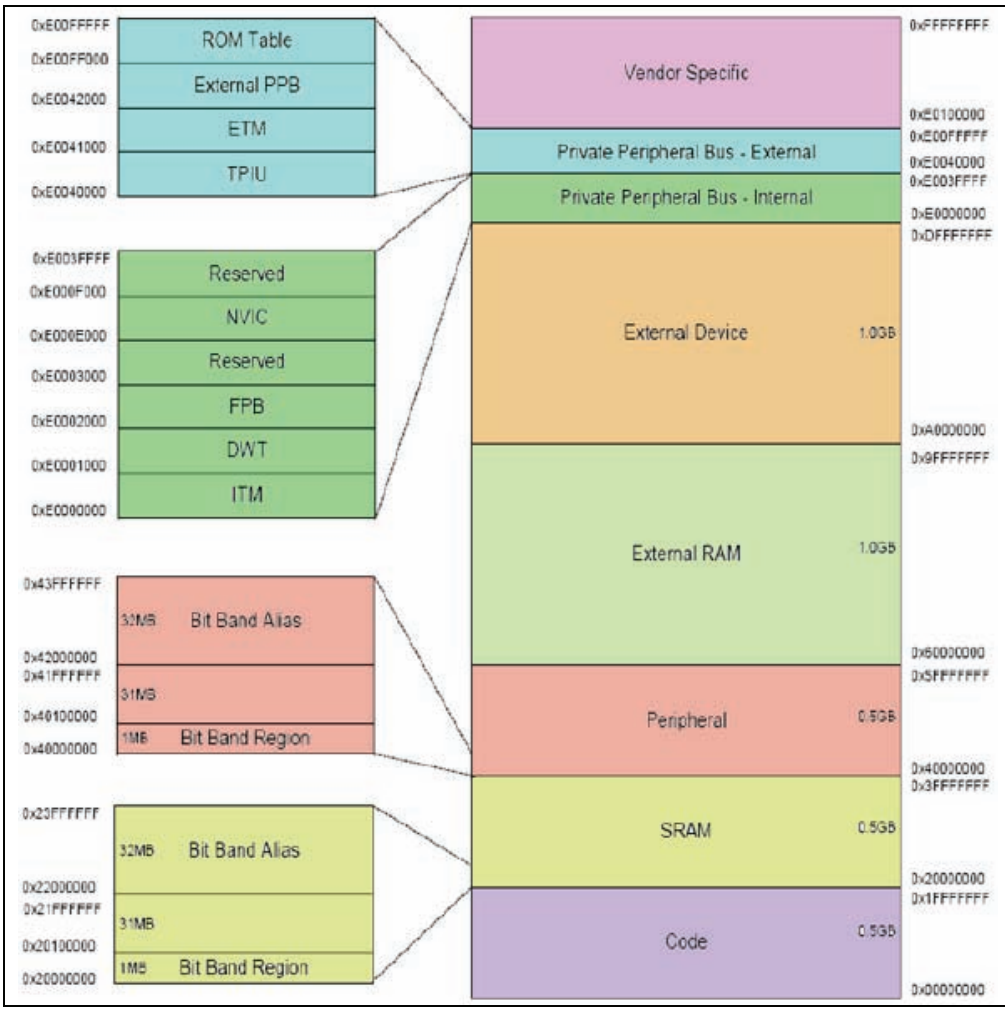
| Source | Destination | Cycles |
|---|---|---|
| 16b x 16b | 32b | 1 |
| 32b x 16b | 32b | 1 |
| 32b x 32b | 32b | 1 |
| 32b x 32b | 64b | 3-7* |

*Перевод к таблице: Sourse - Операнды, Destination - Результат, Cycles - Циклы.*

Система команд Thumb-2 также имеет: улучшенные команды ветвления, включая тест и сравнение; условно исполняемые блоки if/then; для манипуляции с данными команды упорядочения байтов и команды извлечения байтов и полуслов. Так как ЦПУ Cortex – это все еще RISC процессор, он имеет богатый набор команд, специально разработанных под Си компилятор. Обычная программа для Cortex-M3 пишется полностью на ANSI Си с минимальным количеством зарезервированных слов (не ANSI). Единственное исключение – таблица векторов прерываний, которая пишется на Ассемблере.

### 2.3.5 Карта Памяти

Процессор Cortex-M3 – это стандартное микроконтроллерное ядро, поэтому содержит четко определенную карту памяти. Несмотря на множетво внутренних шин, карта памяти представляет собой линейное 4 Гбайтное адресное пространство.




**Cortex-M3 имеет фиксированную 4 Гб карту памяти с указанными областями для кода, SRAM, периферийных устройств, внешней памяти и устройств и системных регистров Cortex. Эта карта памяти общая для всех устройств на базе Cortex.**

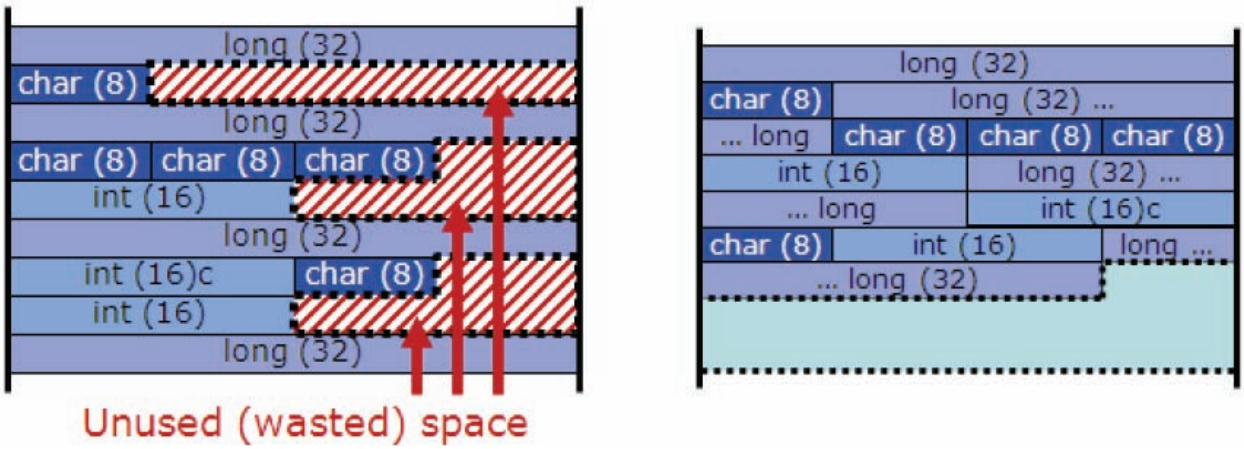
Перевод к рисунку: Reserved - Зарезервированно, Bit Band Alias – Область Псевдоимен Битовых Сегментов, Bit Band Region – Область Битовых Сегментов, Vendor Specific – Определяется Производителем, Private Peripheral Bus – External - Приватная Шина Периферийных Устройств - Внешняя, Private Peripheral Bus – Internal – Приватная Шина Периферийных Устройств - Внутренняя, External Device – Внешние Периферийные Устройства, External RAM – Внешняя RAM, Peripheral – Периферийные Устройства, Code – Программный Код.

Первый 1 Гбайт памяти разделен поровну между областями для программного кода и для SRAM. Область для кода доступна через шину I-Code, а область SRAM через шину D-code. Несмотря на то, что программный код может загружаться и исполняться из SRAM, команды в таком случае будут извлекаться с использованием системной шины, что приводит к дополнительным задержкам. И вероятнее всего код будет исполняться медленнее из SRAM, чем из встроенной Flash памяти, расположенной в области для программного кода. Следующие 0.5 Гбайт памяти выделены под встроенные периферийные устройства. Все периферийные устройства, предоставляемые производителем, будут расположены в этой области. Первый 1 Мбайт областей для SRAM и периферийных устройств является побитово адресуемым, что достигается при помощи техники, называемой

«битовая сегментация». Так как вся SRAM и пользовательские периферийные устройства STM32 расположены как раз на этом участке, вся память STM32 может быть доступна в формате слов или бит. Следующие 2 Гбайта адресного пространства выделены под внешнюю SRAM и внешние периферийные устройства. И последние 0.5 Гбайт предназначены для внутренних процессорных периферийных устройств Cortex и для дальнейших специфических улучшений процессора Cortex. Все регистры процессора Cortex имеют фиксированное местоположение во всех микроконтроллерах на базе Cortex. Это позволяет легко переносить код между различными вариантами STM32 и даже на Cortex-микроконтроллеры других производителей. Один процессор для изучения, один набор инструментария для покупки и можете легко оптимизировать приложение, выбирая подходящий по производительности и функциональности микроконтроллер.

### 2.3.6 Невыровненный Доступ к Памяти

Системы команд ARM7 и ARM9 способны оперировать со знаковыми и беззнаковыми переменными в формате байта, полуслова и слова. Это позволяет ЦПУ свободно поддерживать целочисленные переменные без потребности в дополнительных библиотеках, как это обычно требуется для 8- и 16-битных микроконтроллеров. Однако ранние версии ЦПУ ARM имеют недостаток в том, что они могут осуществлять доступ к памяти с выравниванием только по словам и полусловам. Это ограничивает возможности компоновщика при уплотнении данных в SRAM, что приводит к дополнительным затратам памяти (в зависимости от числа используемых переменных, дополнительные затраты памяти могут достигать 25%).



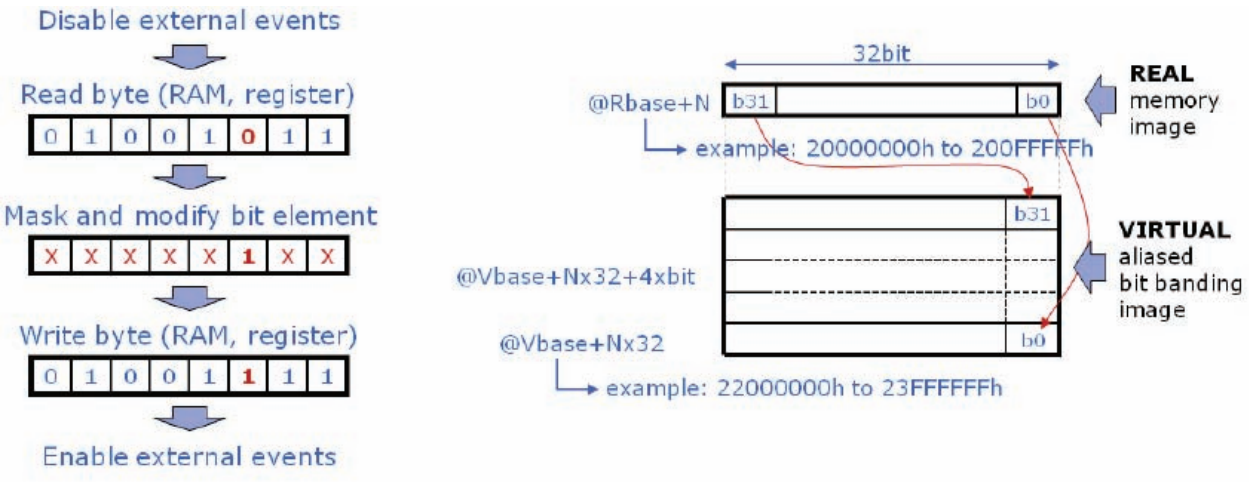

**Cortex-M3 может осуществлять невыровненный доступ к памяти, что увеличивает эффективность использования SRAM.**

*Перевод к рисунку: Unused (wasted) space – Неиспользуемое (потеряное) пространство*

ЦПУ Cortex имеет режимы адресации для слов, полуслов и байт, но он может осуществлять и невыровненный доступ к памяти. Это дает компоновщику дополнительную свободу при размещении данных программы в памяти. За счет поддержки битовой сегментации, ЦПУ Cortex может упаковывать программные флаги в переменные формата полуслова и слова и не использовать отдельный байт для каждого флага.

### 2.3.7 Битовая сегментация

Предыдущие версии ARM7 и ARM9 могли осуществлять битовые манипуляции в SRAM и областях памяти для периферийных устройствт только с использованием операций И и ИЛИ. При этом требовалось выполнять команду «ЧТЕНИЕ ИЗМЕНЕНИЕ ЗАПИСЬ», что очень затратно с точки зрения числа циклов на установку и сброс битов и числа команд, необходимых для каждой манипуляции с битами.

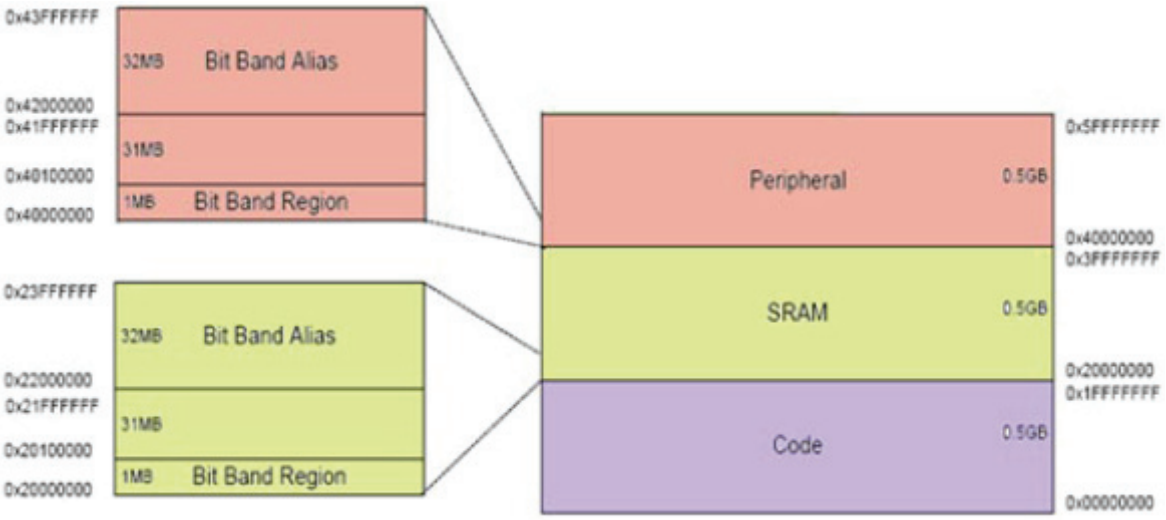



**Технология битовой сегментации позволяет осуществлять атомарные битовые манипуляции.**

*Перевод к рисунку: Disable external events – Запрещение внешних событий, Read byte (RAM, register) – Чтение байта (ОЗУ, регистр), Mask and modify bit element – Маскирование и изменение битов, Write byte (RAM, register) – Запись байта (ОЗУ, регистр), Enable external events – Разрешение внешних событий, REAL memory image – РЕАЛЬНАЯ память, VIRTUAL aliased bit banding image – ВИРТУАЛЬНЫЕ псевдоимена битовых сегментов, example - пример.*

Для преодоления этого ограничения, можно бы было ввести специальные команды установки и сброса битов, или полный Булев процессор, но это бы привело к увеличению размера и сложности ЦПУ. Внедрение технологии битовой сегментации в областях памяти SRAM и периферийных устройств позволяет осуществлять битовую манипуляцию без необходимости в дополнительных командах. Область с побитовой адресацией карты памяти Cortex состоит из

области битовых сегментов (размером до 1 МБ реальной памяти и регистров периферийных устройств) и области псевдоимен битовых сегментов, которая занимает до 32 МБ карты памяти. При битовой сегментации каждому биту в области битовых сегментов соответствует адрес слова из области псевдоимен. Таким образом, устанавливая или сбрасывая значения по адресам слов псевдоимен, можно устанавливать и сбрасывать биты в реальной памяти.




**Битовая сегментация работает в пределах первого 1 Мбайта областей SRAM и Периферийные устройства. В этих областях укладываются все ресурсы STM32.**

*Перевод к рисунку: Bit Banding Alias – Область Псевдоимен Битовых Сегментов, Bit Band Region –Область Битовых Сегментов , Periferal – Периферийные устройства, Code – Программный Код.*

Такая технология позволяет нам осуществлять битовые манипуляции без необходимости в специальных командах и увеличении размеров ядра Cortex. На практике, нам необходимо вычислять адрес псевдоимени битового сегмента для заданной ячейки памяти области SRAM или периферийных устройств. Формула для вычисления адреса псевдоимени приведена ниже:

Адрес в области псевдоимен = Начальный адрес области псавдоимен + смещение слова
Смещение слова = Смещение в байтах от начала битовых сегментов x 0x20 + номер бита x 4

Все это намного проще, чем кажется на первый взгляд. Для примера рассмотрим, как осуществляется запись в выходной регистр данных портов ввода/вывода общего назначения (GPIO) для установки и сброса отдельных линий ввода/вывода. Физический адрес выходного регистра Порта B 0x40010C0C. В этом примере будем устанавливать и сбрасывать восьмой бит в слове, используя вышеприведенную формулу.

Адрес слова =0x40010C0C
Начало битовых сегментов Периферийных устройств =0x40000000
Начало псевдоимен битовых сегментов Периферийных устройств =0x42000000
Байтовое смещение от начала битовых сегментов =0x40010C0C–0x40000000=10C0C
Смещение слова =(0x10C0C x 0x20)+(8x4)=0x2181A0
Адрес псевдоимени =0x42000000+0x2181A0=0x422181A0

Теперь мы можем создать указатель на этот адрес, использую следующую строку языка Си:

```
#define PortBbit8        (*((volatile unsigned long *) 0x422181A0 ))
```

Этот указатель затем может использоваться для установки и сброса бита порта ввода/вывода:

```
PB8 = 1;        //включить светодиод
```

В результате чего генерируются следующие ассемблерные команды:

```
MOVS        r0, #0x01
LDR         r1, [pc, #104]
STR         r0, [r1, 0x00]
```

Выключение светодиода:

```
PB = 0;         //выключить светодиод
```

Генерируются следующие ассемблерные команды:

```
MOVS        r0, #0x00
LDR         r1, [pc, #88]
STR         r0, [r1, 0x00]
```

Каждая из операций, установка и сброс, занимают три 16-битные команды и на STM32, работающем с частотой 72 МГц, эти команды выполнятся за 80 нс. Любое слово в битово сегментированной области периферийных устройств и памяти SRAM может также напрямую адресоваться в формате слова, так что мы можем осуществить ту же самую установку и сброс, используя традиционный подход с операциями И и ИЛИ:

```
GPIOB -> ODR  |=  0x00000100;                    //включить светодиод
 LDR        r0, [pc, #68]
 ADDS               r0, r0, #0x08
 LDR        r0, [r0, #0x00]
 ORR        r0, r0, #0x100
 LDR        r0, [pc, #64]
 STR        r0, [r1, #0xC0C]
```

```
GPIOB -> ODR  &= ! 0x00000100;          //выключить светодиод
 LDR          r0, [pc, #40]
 ADDS               r0, r0, #0x08
 LDR          r0, [r0, #0x00]
 MOVS               r0,  #0x00
 LDR          r1, [pc, #40]
 STR          r0, [r1, #0xC0C]
```

Каждая из операций, установка и сброс, требует группы из 16 и 32-битных команд, что потребляет как минимум 14 байт памяти для каждой операции и 180 нс времени при той же тактовой частоте. В обычном встраиваемом приложении очень часто используются операции установки и сброса битов в регистрах периферийных устройств, семафоров и флагов в SRAM. И если принять во внимание преимущества технологии битовой сегментации, то можно значительно сократить объем кода приложения и время его исполнения. Тем более, в заголовочных файлах STM32 уже проведены все необходимые расчеты для использования этой технологии.

## 2.4 Процессор Cortex

### 2.4.1 Шины

Процессор Cortex-M3 построен по Гарвардской архитектуре памяти с раздельными шинами для кода и данных. Эти шины носят названия Icode и Dcode. Обе шины имеют доступ к программному коду и данным в диапазоне адресов 0x00000000 – 0x1FFFFFFF. Дополнительная системная шина используется для доступа к системной управляющей области Cortex в диапазоне адресов 0x20000000 – 0xDFFFFFFF и 0xE0100000 – 0xFFFFFFFF. Встроенная система отладки Cortex имеет дополнительную структуру шин, называемую Private Peripheral Bus.

### 2.4.2 Матрица Шин

Системная шина и шина данных подключаются к внешнему микроконтроллеру через совокупность высокоскоростных шин, организованных в виде матрицы. Это позволяет реализовывать множество параллельных каналов связи между шинами Cortex и внешними устройствами управления передачей данных по шине, например, между DMA и SRAM. Если два устройства управления передачей данных по шине (то есть, ЦПУ Cortex и канал DMA) пытаются получить доступ к одному и тому же периферийному устройству, встроенный арбитр разрешит конфликт и предоставит доступ устройству с высшим приоритетом. Как мы увидим при рассмотрении модуля DMA, STM32 построен таким образом, что DMA и ЦПУ Cortex работают согласованно.

### 2.4.3 Системный Таймер SysTick

Ядро Cortex содержит 24-битный вычитающий счетчик, с функциями автоперезагрузки и генерации прерывания по завершению счета. Это стандартный таймер для всех микроконтроллеров на базе Cortex. Таймер SysTick может использоваться для отсчета системных интервалов RTOS, или для генерации периодических прерываний запланированных задач. Источник тактирования для таймера SysTick настраивается с помощью регистра SysTick Control и регистра статуса в области управления Cortex-M3 System. При установленном бите CLKSOURSE, таймер SysTick будет работать на частоте ЦПУ. Если этот бит сброшен, таймер будет работать на частоте в 8 раз меньшей частоты ЦПУ.

| | |
|---|---|
|  | **Таймер SysTick – это 24-битный автоматически перезагружающийся таймер в составе процессора Cortex-M3. Он предназначен для отсчета интервалов Операционной Системы Реального Времени.** |

*Перевод к рисунку: Control & Status – Управление и Статус, Reload Value – Значение Перезагрузки, Current Value – Текущее Значение.*

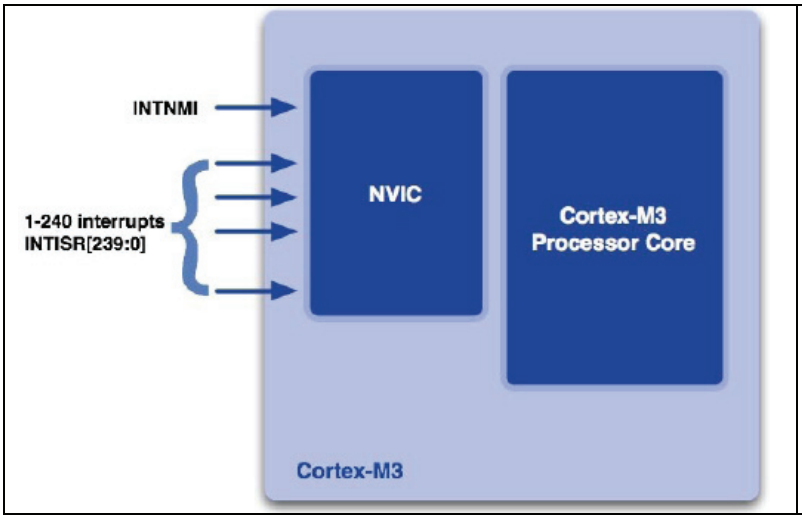
Таймер SysTick содержит три регистра. Текущее значение и значение перезагрузки должны устанавливаться вместе с периодом счета. Регистр управления и статуса содержит бит ENABLE для запуска таймера и бит TICKINT для включения линии прерываний. В следующей главе мы рассмотрим структуру прерываний Cortex и использование таймера SysTick для генерации первой исключительной ситуации STM32.

### 2.4.4 Обработка Прерываний

Одним из ключевых улучшений ядра Cortex по сравнению с предыдущими ЦПУ ARM заключается в структуре прерываний и обработке исключений. ARM7 и ARM9 содержат два вида прерываний: быстрые прерывания и прерывания общего пользования. Эти два вида прерывания должны обслуживать все источники прерываний внутри микроконтроллера. Структура прерываний ARM7 и ARM9 имеет две проблемы. Первое, она не детерминирована; время, затрачиваемое на остановку или отмену выполняемой команды при возникновении прерывания, непостоянно. Это может не быть проблемой для многих приложений, но это очень серьезный момент для управления в реальном времени. Второе, структура прерываний ARM7 и ARM9 не поддерживает вложенных прерываний; требуется дополнительное программное обеспечение: либо макросы Ассемблера, либо RTOS. В ядре Cortex этих проблем нет, структура прерываний является детерминированной и невероятно быстрой.

### 2.4.5 Контроллер Вложенных Векторных Прерываний NVIC

Контроллер Вложенных Векторных Прерываний (NVIC) – это стандартный модуль ядра Cortex. Это значит, что любой микроконтроллер на базе Cortex будет иметь идентичную структуру прерываний, независимо от производителя. Таким образом, код приложения и операционная система могут легко переноситься с одного микроконтроллера на другой и программисту не придется осваивать новый набор регистров. NVIC разработан обеспечивать очень малые задержки прерывания. Это достигается с помощью самого NVIC и системы команд Thumb-2, которая позволяет прерывать многоцикловые команды. Задержка прерывания также детерминированна, имеются специальные возможности обработки прерываний для поддержки приложений реального времени. Как отражено в названии, Контроллер Вложенных Векторных Прерываний поддерживает вложенные прерывания и в STM32 имеются 16 уровней приоритета. Структура прерываний программируется полностью на C и не требует никаких макросов Ассемблера или специальных не-ANSI директив.



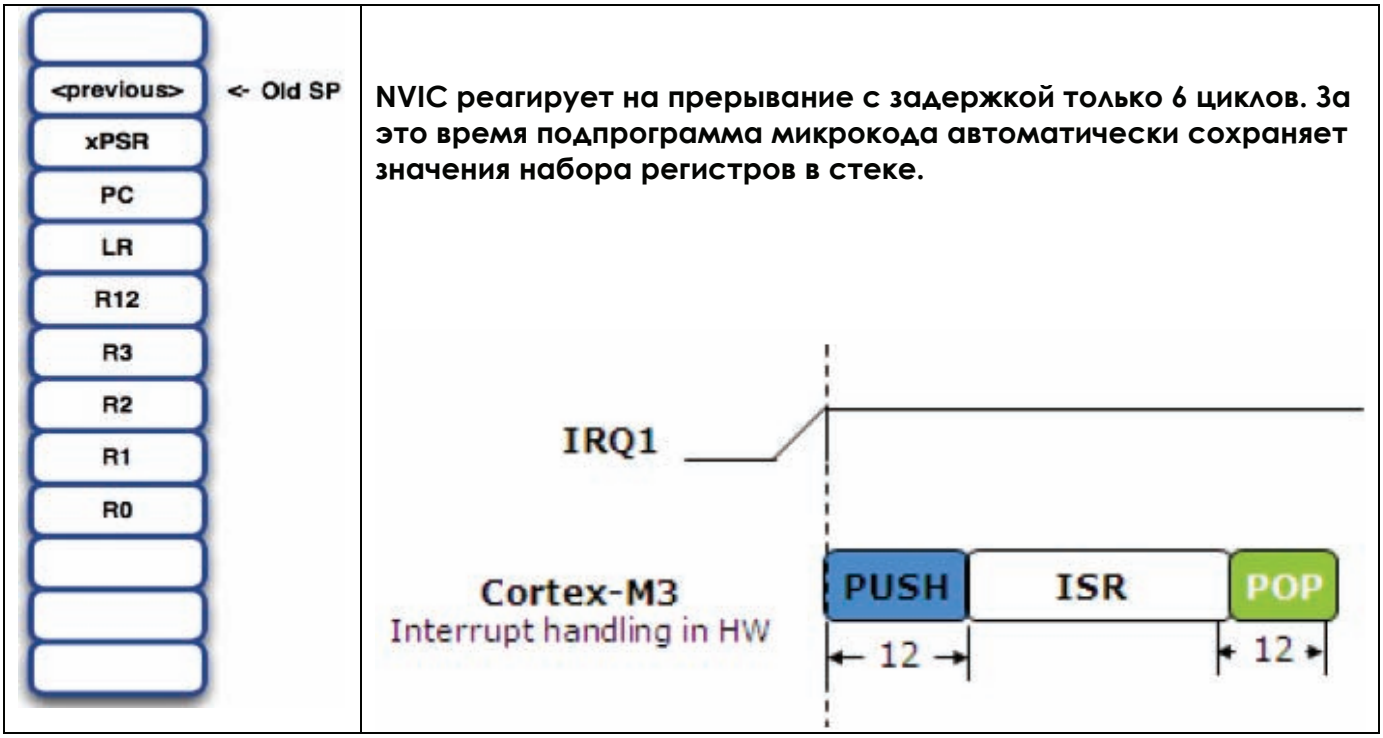

**Процессор STM32 содержит Контроллер Вложенных Векторных Прерываний, который может обслуживать до 240 внешних периферийных устройств.**

*Перевод к рисунку: 1-240 interrupts – 1-240 прерываний, Processor Core – Процессороное Ядро.*

Несмотря на то, что NVIC является стандартным модулем в составе ядра Cortex, для того, чтобы число вентилей оставалось минимальным, число линий прерываний, поступающих на вход NVIC, конфигурируется на этапе разработки микроконтроллера. NVIC содержит одно немаскируемое прерывание и до 240 линий внешних прерываний, которые могут быть соединены с пользовательскими периферийными устройствами. Также имеются дополнительные 15 источников прерываний внутри ядра Cortex, используемые для обработки внутрених исключительных ситуаций самого ядра Cortex. NVIC STM32 был синтезирован с максимальным числом линий маскируемых прерываний равным 43.

2.4.5.1 NVIC: Функционирование, Исключения, Вход и Выход

При возникновении прерывания от периферийного устройства, NVIC начинает перевод ЦПУ Cortex в режим обработки прерывания. Как только ЦПУ Cortex входит в режим обработки прерывания, оно сохраняет значения набора регистров в стек. Важно то, что это осуществляет микрокод, то есть не требуется дополнительных команд в коде приложения. В то время как в стеке сохраняются данные, извлекается начальный адрес процедуры обработки прерываний. Таким образом, с момента возникновения прерывания до начала выполнения первой команды процедуры обработки прерывания проходит только 12 циклов.




*Перевод к рисунку: previous – предыдущий, Old SP – Старое значение SP, Interrupt handling in HW – Аппаратная обработка прерывания.*

В стеке сохраняются значения регистра Program Status Register, счетчика команд и значение регистра связи, чтобы запомнить состояние ЦПУ Cortex на момент возникновения прерывания. Кроме того, значения регистров R0 – R3 тоже сохраняются. В стандарте бинарного интерфейса ARM эти регистры используются для передачи данных, поэтому, сохранив их значения, мы увеличиваем число регистров, которые могут использоваться в процедуре обработки прерываний. И последнее, значение регистра R12 также сохраняется; это рабочий регистр внутренних вызовов. Он используется каждой программой, которая выполняется во время вызова функции. Например, если вы активировали проверку стека во время компиляции, сгенерированный код будет использовать регистр R12, если будет необходимость в регистре ЦПУ. По завершению обработки прерывания, все

действия выполняются в обратном порядке, микрокод восстанавливает информацию из стека, одновременно с этим извлекается адрес возврата. Выполнение программного кода возобновляется через 12 циклов.

### 2.4.5.2 Расширенный Режим Обработки Прерываний

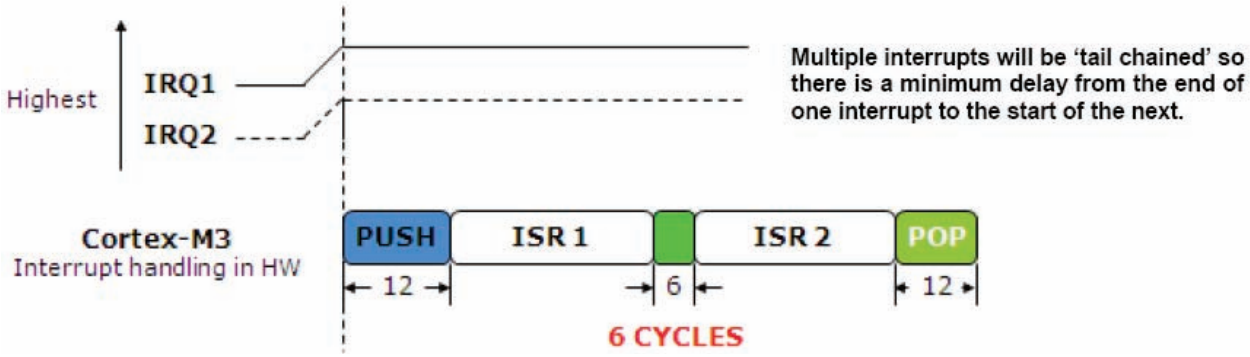
Кроме быстрой обработки одиночных прерываний, в NVIC на этапе разработки закладывалась возможность эффективной обработки нескольких прерываний приложения реального времени. В NVIC реализовано несколько методов обработки прерываний от множества источников с минимальной задержкой между прерываниями, гарантирующих, что прерывание с высшим приоритетом будет обработано в первую очередь.

#### 2.4.5.2.1 Приоритетное обслуживание прерываний

NVIC разработан таким образом, что прерывание с большим приоритетом будет обработано без очереди, даже если в этот момент уже обрабатывается прерывание с меньшим приоритетом. При этом, обработка прерывания с меньшим приоритетом останавливается, новый стековый фрейм сохраняется за стандартные 12 циклов, после чего запускается обработчик прерывания с большим приоритетом. После завершения обработки прерывания с большим приоритетом, данные из стека автоматически выталкиваются, и возобновляется обработка прерывания с меньшим приоритетом.

#### 2.4.5.2.2 Постановка Прерываний в Очередь

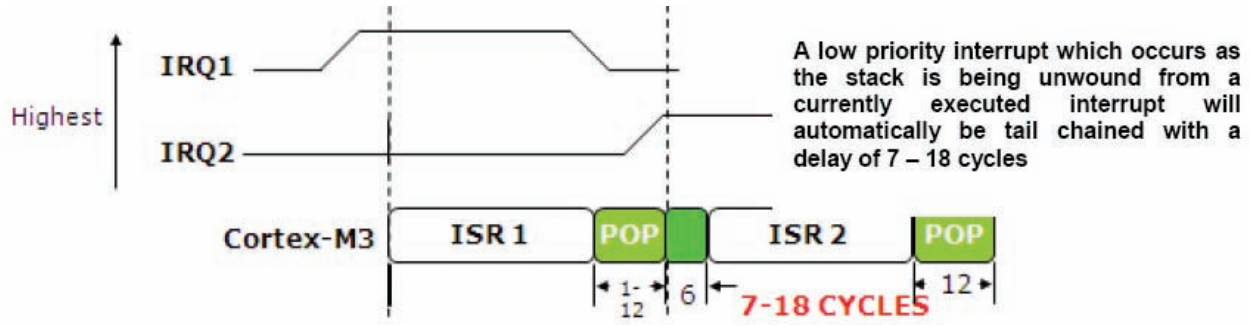
Если во время обработки прерывания с большим приоритетом возникает прерывание с меньшим приоритетом, NVIC использует метод под названием «постановка в очередь» для обеспечения минимальной задержки входа в обработчик прерывания.




*Перевод к рисунку: Highest - Высший, Interrupt handling in HW – Обработка прерывания, 6 CYCLES – 6 ЦИКЛОВ,* ***Multiple interrupts will be "tail chained" so there is***

***a minimum delay from the end of one interrupt to the start of the next – Для того, чтобы сократить задержку между завершением обработки одного и началом обработки другого прерывания, формируется очередь из нескольких пришедших прерываний.***

При возникновении двух прерываний, прерывание с большим приоритетом начнет обрабатываться первым через 12 циклов тактового сигнала. Однако в момент завершения процедуры обработки прерывания, ЦПУ Cortex не возвращается к работе в фоновом режиме. Стековый фрейм не восстанавливается, а только извлекается начальный адрес процедуры обработки следующего прерывания. Это занимает всего шесть циклов, после чего начинается выполнение функции обработки следующего прерывания. После обработки всей очереди прерываний, восстанавливается информация из стека и извлекается адрес возврата в программу. Таким образом, происходит переход к работе в фоновом режиме за 12 циклов. Если прерывание с меньшим приоритетом возникает в момент завершения обработки прерывания с большим приоритетом, выталкивание данных из стека останавливается, и указатель стека возвращается на исходную позицию. В этом случае затрачиваются дополнительные 6 циклов, пока извлекается начальный адрес процедуры обработки возникшего прерывания. Полное время задержки до входа в процедуру обработки прерывания получается в районе между 7 и 18 циклами.

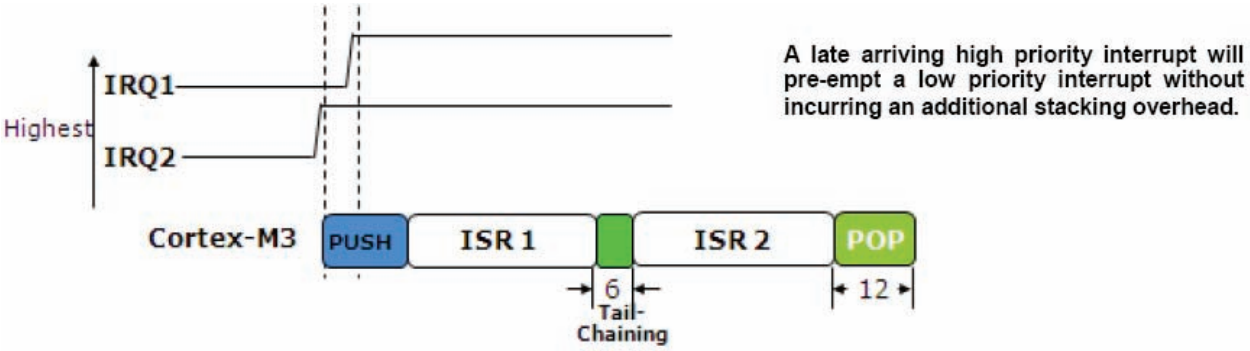



*Перевод к рисунку: Highest – Высшее, 7-18 CYCLES – 7-18 ЦИКЛОВ,* ***A low priority interrupt… 7-18 cycles – Прерывание с меньшим приоритетом, возникшее в момент восстановления информации из стека по завершению обработки предыдущего прерывания, будет автоматически поставлено в очередь с задержкой 7-18 циклов.***

2.4.5.2.3 Позднее Появление

В системах реального времени часто возникает ситуация, когда приходит прерывание с большим приоритетом, а обработка прерывания с меньшим приоритетом уже начата. Если это случается во время начального сохранения информации в стеке, NVIC переключается на обработку прерывания с большим приоритетом. Сохранение информации в стеке продолжается и пройдет еще

минимум 6 циклов, за время которых извлекается начальный адрес процедуры обработки прерываний.




*Перевод к рисунку: Highest - Высшее, Tail-chaining – постановка в очередь,* ***A late arriving…stacking overhead – Прерывание с большим приоритетом, пришедшее позднее, прервет обработку прерывания с меньшим приоритетом без дополнительной перезагрузки стека****.*

Прерывание с меньшим приоритетом ставится в очередь и начнет обрабатываться через 6 циклов после завершения обработки прерывания с большим приоритетом.

### 2.4.5.3 Конфигурирование и Использование NVIC

Для использования NVIC нам нужно сделать три вещи. Первое, сконфигурировать таблицу векторов для источников прерываний, которые мы хотим использовать. Второе, установить приоритеты и разрешить прерывания, сконфигурировав регистры NVIC. И третье, настроить периферийные устройства и разрешить их прерывания.

#### 2.4.5.3.1 Таблица Векторов Прерываний

Таблица векторов прерываний Cortex находится в низу адресного пространства. Однако вместо того, чтобы начинаться с нулевого адреса, первым адресом таблицы векторов прерывания является 0x00000004. Первые четыре байта используются для хранения начального адреса указателя стека.

| No. | Exception Type | Priority | Type of Priority | Descriptions |
|---|---|---|---|---|
| 1 | Reset | -3 (Highest) | fixed | Reset |
| 2 | NMI | -2 | fixed | Non-Maskable Interrupt |
| 3 | Hard Fault | -1 | fixed | Default fault if other hander not implemented |
| 4 | MemManage Fault | 0 | settable | MPU violation or access to illegal locations |
| 5 | Bus Fault | 1 | settable | Fault if AHB interface receives error |
| 6 | Usage Fault | 2 | settable | Exceptions due to program errors |
| 7-10 | Reserved | N.A. | N.A. | |
| 11 | SVCall | 3 | settable | System Service call |
| 12 | Debug Monitor | 4 | settable | Break points, watch points, external debug |
| 13 | Reserved | N.A. | N.A. | |
| 14 | PendSV | 5 | settable | Pendable request for System Device |
| 15 | SYSTICK | 6 | settable | System Tick Timer |
| 16 | Interrupt #0 | 7 | settable | External Interrupt #0 |
| ..... | ..... | ..... | settable | ..... |
| 256 | Interrupt#240 | 247 | settable | External Interrupt #240 |

**The Cortex exception table contains the start address or an ISR which is loaded into the Program counter as the CPU enters the exception.**

Перевод рисунка:

***The Cortex exception… CPU enters the exception – Таблица исключений Cortex содержит начальные адреса или адреса процедур обработки прерываний, которые загружаются в счетчик команд при возникновении исключений.***

| № | Тип Исключения | Приоритет | Тип Приоритета | Описание |
|---|---|---|---|---|
| 1 | Reset | -3 (Высший) | фиксирован | Сброс |
| 2 | NMI | -2 | Фиксирован | Немаскируемое прерывание |
| 3 | Hard Fault | -1 | фиксирован | Ошибка по умолчанию, если другие пункты непременимы |
| 4 | MemManage Fault | 0 | задается | Нарушение МПУ или обращение к запрещенной области |
| 5 | Bus Fault | 1 | задается | Интерфейс AHB принял ошибку |
| 6 | Usage Fault | 2 | задается | Исключение из-за программной ошибки |
| 7-10 | Зарезервировано | недоступно | | |
| 11 | SVCall | 3 | задается | Вызов Системного Сервиса |
| 12 | Dedug Monitor | 4 | задается | Точки останова, точки просмотра, внешняя отладка |
| 13 | Зарезервировано | Недоступно | | |
| 14 | PendSV | 5 | задается | «Подвешиваемый» запрос для Системного Устройства |
| 15 | SYSTICK | 6 | задается | Таймер System Tick |
| 16 | Прерывание #0 | 7 | задается | Внешнее прерываний №0 |
| …… | …… | …… | задается | …… |
| 256 | Прерывание #240 | 247 | задается | Внешнее прерывание №240 |

Каждый вектор прерывания имеет размер четыре байта и содержит начальный адрес той процедуры обработки прерывания, с которой связано данное прерывание. Первые 15 векторов предназначены для исключений, происходящих внутри самого ядра Cortex. В их число входит вектор сброса, немаскируемое прерывание, ошибка управления, исключения отладочной системы, а также прерывание от таймера SysTick. Система команд Thumb-2 также содержит

команды вызова системных сервисов, при исполнении которых генерируются прерывания. Прерывания от пользовательских периферийных утройств начинаются с 16-го вектора и связаны с периферийными устройствами так, как определил производитель. В программном коде, таблица векторов прерываний, обычно в начале, заполняется адресами процедур обработки прерываний.

```
                AREA        RESET, DATA, READONLY
                EXPORT              __Vectors
__Vectors       DCD         __initial_sp            ; Вершина стека
                DCD         Reset_Handler           ; Обработчик Сброса
                DCD         NMI_Handler             ; Обработчик NMI
                DCD         HardFault_Handler       ; Обработчик аппаратной ошибки
                DCD         MemManage_Handler       ; Обработчик ошибки МПУ
                DCD         BusFault_Handler        ; Обработчик ошибки шины
                DCD         UsageFault_Handler      ; Обработчик ошибки использования
                DCD         0                       ; Зарезервировано
                DCD         0                       ; Зарезервировано
                DCD         0                       ; Зарезервировано
                DCD         0                       ; Зарезервировано
                DCD         SVC_Handler             ; Обработчик SVCall
                DCD         DebugMon_Handler        ; Обработчик монитора отладки
                DCD         0                       ; Зарезервировано
                DCD         PendSV_Handler          ; Обработчик PendSV
                DCD         SysTick_Handler         ; Обработчик SysTick
```

В случае с таймером SysTick мы можем создать процедуру обработки прерывания, объявив Си-функцию с идентичным символьным именем:

```
void SysTick_Handler (void)

{

....

}
```

Теперь, с заданной таблицей векторов прерываний и с объявленным прототипом функции обработки прерываний, мы можем сконфигурировать NVIC на обработку прерывания от таймера SysTick. В целом, нам нужно проделать две вещи: установить приоритет прерывания и включить источник прерывания. Регистры NVIC расположены в системной области управления.

| | |
|---|---|
|  | **Регистры NVIC расположены в системной области управления Cortex-M3 и доступны только когда ЦПУ работает в привилегированном режиме**. |

*Перевод к рисунку: Interrupt Control – Управление Прерываниями, IRQ SET and CLEAR – УСТАНОВКА и СБРОС запросов, IRQ Pending – Ожидающие Запросы, IRQ ACTIVE – Активные Запросы, IRQ PRIORITY – ПРИОРИТЕТ ПРЕРЫВАНИЙ, System Control – Системное Управление, System Priority – Системный Приоритет, Fault Status – Статус Ошибок*.

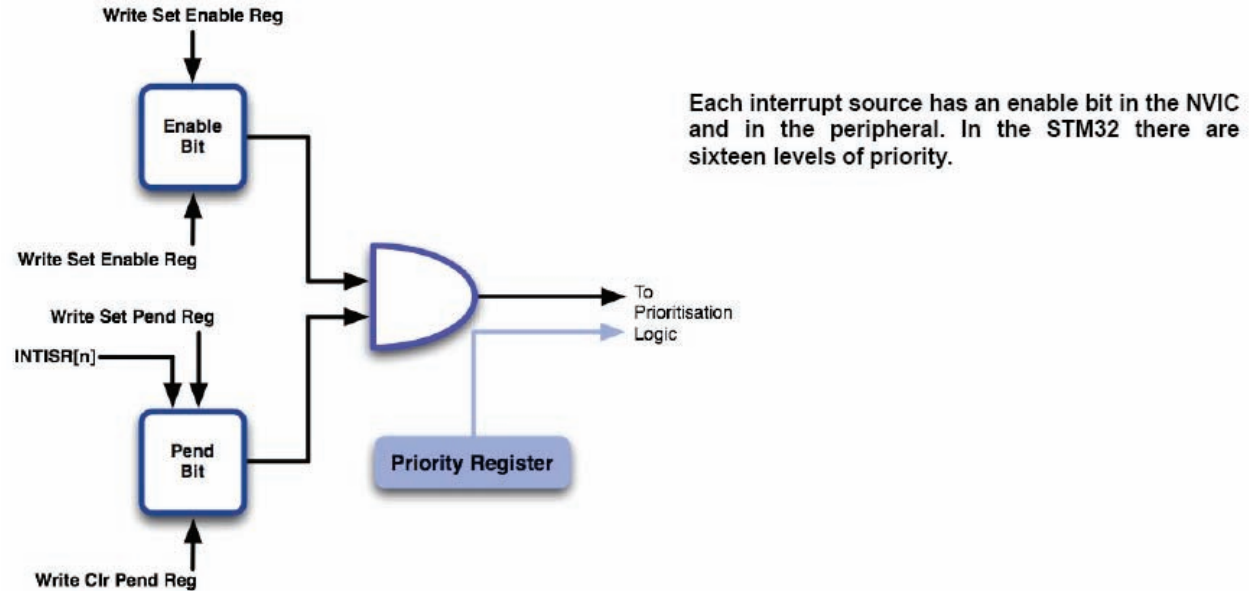
Внутренние исключения Cortex конфигурируются с помощью системного регистра управления и системного регистра прерываний, в то время как исключения пользовательских периферийных устройств – с помощью регистров запросов прерываний IRQ. Прерывание таймера SysTick является внутренним исключением Cortex и настраивается в системных регистрах. Некоторые из внутренних исключений постоянно включены; в их число входят сброс, немаскируемое прерывание NMI и таймер SysTick. Так что не требуется никаких явных действий для активации прерывания таймера SysTick. Для конфигурации прерывания таймера SysTick нам нужно включить таймер и разрешить прерывание в самом периферийном устройстве:

```
SysTickCurrent  = 0x9000;          // Начальное значение системного таймера
SysTickReload = 0x9000;            // Значение перезагрузки
SysTickControl= 0x07;              // Запуск таймера и разрешение прерывания
```

Приоритет каждого из внутренних исключений Cortex может быть установлен в системном регистре прерываний. Приоритет сброса, немаскируемого прерывания и аппаратной ошибки фиксированы для того, чтобы ядро безошибочно приходило к нужному исключению. Каждое из оставшихся исключений имеет восьмибитное поле в одном из трех системных регистров прерываний. В STM32 используются 16 уровней приоритетов прерываний, так что активны только четыре бита восьмибитного поля. Очень важно отметить, что приоритет задается установкой четырех старших битов.

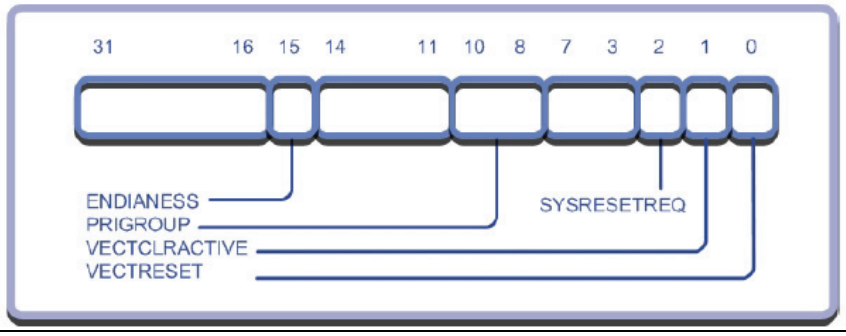
Каждое из пользовательских периферийных устройств управляется через блок регистров запросов прерываний IRQ. Каждое пользовательское периферийное устройство имеет бит разрешения прерывания Interrupt Enable bit. Эти биты расположены в двух 32-битных регистрах IRQ Set Enable. Соответственно существуют регистры IRQ Clear Enable, использующиеся для запрещения

прерываний. NVIC содержит также регистры, которые отображают ожидающие и активные прерывания, для того, чтобы пользователь мог определить текущее состояние и источник прерываний.




*Перевод к рисунку: Write Set Enable Reg –Установка Бита в Регистре Разрешения, Enable Bit – Бит Разрешения, Write Set Enable Reg – Сброс Бита в Регистре Разрешения, Write Set Pend Reg – Установка Бита в Регистре Ожидания, Pend Bit – Бит Ожидания, Write Clr Pend Reg – Сброс Бита в Регистре Ожидания, Priority Register – Регистр Приоритетов, To Prioritisation Logic – В логику установки приоритетов,* ***Each interrupt sourse … levels of priority – Каждый источник прерываний имеет бит разрешения в NVIC и в периферийном устройстве. В STM32 используются 16 уровней приоритетов прерываний***.

Имеются шестнадцать регистров приоритетов. Каждый из регистров разделен на четыре 8-битных поля. Каждое из полей предназначено для отдельного вектора прерываний. Для организации 16 уровней прерываний в STM32 используется только половина каждого поля. Важно отметить, что для установки приоритетов используются старшие полубайты полей. По умолчанию, нулевой уровень соответствует высшему уровню приоритета, а 15-ый – низшему. Уровни приоритетов можно также разделить на группы и подгруппы. Это не обеспечит дополнительных уровней приоритетов, но облегчит управление, если в системе используется большое число прерываний. Для этого нужно запрограммировать поле PRIGROUP в регистре Application Interrupt and Reset Control Register.




**Поле PRIGROUP разделяет уровни прерываний на группы и подгруппы. Это полезно для программной абстракции в приложениях с большим числов прерываний.**

| PRIGROUP (3 Bits) | Binary Point (group.sub) | | Preempting Priority (Group Priority) | | Sub-Priority | |
|---|---|---|---|---|---|---|
| | | | Bits | Levels | Bits | Levels |
| 011 | 4.0 | gggg | 4 | 16 | 0 | 0 |
| 100 | 3.1 | gggs | 3 | 8 | 1 | 2 |
| 101 | 2.2 | ggss | 2 | 4 | 2 | 4 |
| 110 | 1.3 | gsss | 1 | 2 | 3 | 8 |
| 111 | 0.4 | ssss | 0 | 0 | 4 | 16 |

*Перевод к рисунку: 3 Bits – 3 Бита, Binory Point (group.sub) – Двоичная Запятая (группа.подгр), Preempting Priority (Group Priority) – Приоритет (Приоритет Группы), Bits - Биты, Levels - Уровни, Sub-Priority - Подприоритет, Bits - Биты, Levels - Уровни.*

Трехбитное поле PRIGROUP позволяет разделить 4-битные поля на группы и подгруппы. Например, записав 3 в PRIOGROUP, мы создаем две группы, каждая из которых имеет четыре уровня приоритетов. Теперь в приложении мы можем определить группу прерываний с высоким приоритетом и группу прерываний с низким приоритетом. Внутри каждой группы мы можем распределить приритеты подргрупп: низкий, средний, высокий и самый высокий приоритет. Как сказано ранее, это не обеспечит что-либо «экстра», но придает более абстрактный вид структуре прерываний, что полезно при управлении большим числом прерываний. Конфигурирование прерываний периферийных устройств осуществляется по подобию с конфигурированием внутренних исключений Cortex. В случае с прерыванием от АЦП, сначала нужно определить вектор прерывания и написать процедуру обработки прерывания:

```
DCD             ADC_IRQHandler ;

void    ADC_Handler   void

{

}
```

Затем нужно инициализировать АЦП и разрешить прерывание в периферийном устройстве и в NVIC:

```
ADC1 -> CR2= ADC_CR2;  // Включаем АЦП в режиме непрерывного преобразования
ADC1 -> SQR1        = sequence1;        // Выбираем число каналов в последовательности преобразования
ADC1 -> SQR2        = sequence2;        // выбираем канал для преобразования
ADC1 -> SQR3        = sequence3;
ADC1 -> CR2 |= ADC_CR2;         // Перезаписываем бит
ADC -> CR1  = ADC_CR1;  // Стартуем группу каналов, разрешаем прерывание АЦП

GPIOB -> CRH        = 0x33333333;       // Настраиваем светодиодный вывод на выход

NVIC -> Enable[0] = 0x00040000;  // Разрешаем прерывание АЦП
NVIC -> Enable[1] = 0x00000000;
```

### 2.5 Режимы Энергопотребления

Мы рассмотрим все возможности управления энергопотреблением STM32 позднее. В этой главе мы сосредоточимся на режимах энергопотребления самого ядра Cortex. ЦПУ Cortex поддерживает «спящий» режим, при котором ядро переходит в режим наименьшего энергопотребления и останавливается выполнение команд в ЦПУ. Небольшая часть NVIC остается активной для того, чтобы пришедшие от периферийных устройств STM32 прерывания могли «разбудить» ядро.

#### 2.5.1 Вход в Режим Низкого Энергопотребления

Ядро Cortex может быть переведено в спящий режим с помощью выполнения команд:  Wait For Interrupt (WFI) или Wait For Event (WFE). В случае с командой WFI, ядро Cortex возобновит работу в активном режиме и обработает возникшее прерывание. После завершения процедуры обработки прерывания возможны два варианта развития событий. Первый, ядро Cortex продолжает выполнение кода в нормальном режиме. Второе, если установлен бит SLEEPON EXIT в регистре System Control Register, ядро Cortex автоматически перейдет в спящий режим снова. Эта возможность позволяет строить приложения, полностью управляемые прерываниями, то есть, ядро будет автоматически просыпаться, выполнять определенный код и засыпать снова.

Команда WFE позволяет ядру Cortex возобновить выполнени программы с того места, где оно было переведено в спящий режим. При этом не происходит перехода в процедуру обработки исключения. Событие, выводящее ядро из режима сна, представляет собой простое периферийное прерывание, которое не было включено в регистре NVIC. Это позволяет периферийному устройству

разбудить ядро для продолжения выполнения кода без входа в процедуру обработки прерывания. Команды WFI и WFE недоступны из языка Си, компилятор для системы команд Thumb-2 предоставляют внутренние макросы, которые могут быть использованы вместе со стандартными командами Си:

__WFI

__WFE

В дополнение к режимам низкого энергопотребления SLEEPNOW и SLEEPONEXIT, ядро Cortex может выдавать сигнал SLEEPDEEP для остальной части микроконтроллера.

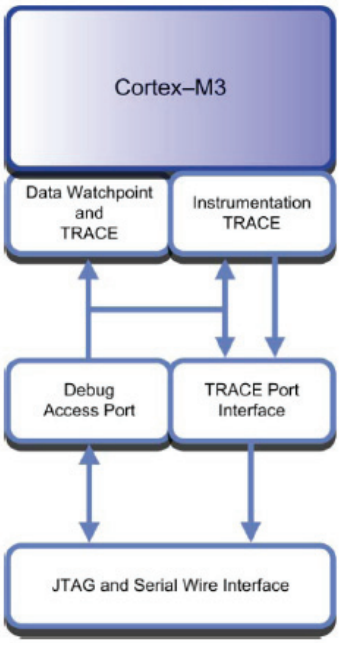

**Режимы низкого энергопотребления процессора Cortex конфигурируются в системном регистре управления. STM32 имеет дополнительный режим низкого энергопотребления, в котором используется сигнал DeepSleep, выдаваемый процессором Cortex.**

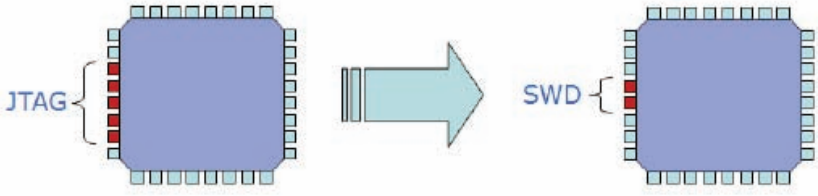
Это позволяет останавливать работу других функциональных модулей, таких как ФАПЧ и пользовательских периферийных устройств. То есть STM32 может входить в свои собственные режимы низкого энергопотребления.

### 2.5.2 Отладочная система CoreSight

Все ЦПУ ARM имеют свою собственную встроенную отладочную систему. ARM7 и ARM9 содержат как минимум JTAG порт, который позволяет при помощи стандартных отладочных средств подключаться к ЦПУ и загружать программный код во встроенное ОЗУ или Flash память. JTAG порт поддерживает базовые функции отладки (пошаговое исполнение программы, постановка точек останова и т.д.), а также позволяет просматривать содержимое областей памяти. ЦПУ ARM7 и ARM9 могут также поддерживать трассировку в реальном времени через дополнительное отладочное периферийное устройство, называемое встроенная трассирующая макро ячейка (ETM). Несмотря на то, что эта система работает хорошо, она имеет некоторые ограничения. JTAG отладчики могут передавать отладочную информацию отладочным средствам только когда ЦПУ ARM остановлено. То есть нет возможности обновления отладочной информации в реальном времени. Также ограничено число аппартных точек останова, хотя

системы команд ARM7 и ARM9 содержат команды останова, вставляемые в программный код отладочными средствами (программные точки останова). Аналогично, для трассировки в реальном времени, нужно, чтобы производитель микроконтроллера встроил ETM модуль, что увеличивает стоимость кристалла. Вследствие чего, это не всегда реализуется. В новом ядре Cortex представлена новая отладочная система, называемая CoreSight.




**Отладочная система CoreSight использует JTAG или последовательный интерфейс serial wire. CoreSight выполняет функции трассировки и управления выполнения программы. Дополнительное преимущество новой системы в том, что она может оставаться работающей во время пребывания STM32 в «спящем» режиме. Это большой шаг в развитии отладочных систем.**

*Перевод к рисунку: Data Watchpoint and TRACE – Точки просмотра данных и ТРАССИРОВЩИК, Instrumentation TRACE – Инструментальный ТРАССИРОВЩИК, Debug Access Port – Порт Доступа Отладчика, TRACE Port Interface – Порт ТРАССИРОВЩИКА, JTAG and Serial Wire Interface – Интерфейсы JTAG и Serial Wire.*

Полноценная отладочная система CoreSight содержит порт доступа отладчика, через который можно подключаться к микроконтроллеру через JTAG. Отладочное средство подключается с помощью стандартного 5-выводного JTAG интерфейса или последовательного 2-проводного интерфейса. В дополнение к возможностям JTAG, полная отладочная система CoreSight содержит трассировщик просмотра данных и встроенную трассирующую макро ячейку ETM. Для программного тестирования имеется инструментальный трассировщик и блок Flash patch. В STM32 применяется отладочная система CoreSight без ETM.

В действительности, структура отладочной системы CoreSight STM32 обеспечивает улучшенную «реально временную» версию стандартного JTAG. CoreSight STM32 имеет 8 аппаратных точек останова, которые могут ставиться и убираться не изменяя режима работы ЦПУ Cortex. Трассировщик просмотра данных позволяет видеть содержимое областей памяти не изменяя режима работы ЦПУ. Отладочная система CoreSight может оставаться активной во время пребывания ядра Cortex в режиме низкого энергопотребления или спящем режиме. Это имеет огромное значение при отладке низкопотребляющего приложения. Кроме того, система CoreSight может останавливать таймера STM32 в те моменты, когда ЦПУ находится в состоянии ожидания. Это позволяет осуществлять пошаговое

исполнение программы и оставлять таймера синхронными с выполением команд. Отладочная система CoreSight значительно улучшает реально-временные характеристики отладки STM32 по сравнению с предыдущими ARM7 и ARM9, не увеличивая стоимость микроконтроллера.

## 3. Включение Микроконтроллера

Минимальный дизайн STM32 действительно может быть минимальным. Для того, чтобы STM32 заработал нужен только источник питания. Микроконтроллер содержит свой собственный RC-осциллятор и внутреннюю схему сброса. В этой главе мы рассмотрим основные вопросы построения аппаратной части, которые придется решать при проектировании устройства на базе STM32.

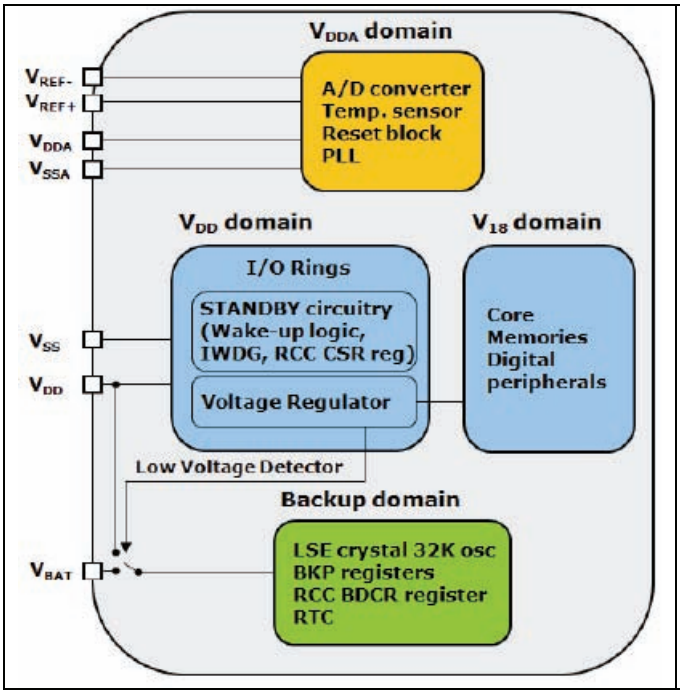
### 3.1 Типы Корпусов

Микроконтроллеры STM32 в вариантах Access и Performance имеют одинаковые типы корпусов, что облегчает возможность аппаратного обновления устройства

без переработки печатной платы. STM32 доступны в корпусах LQFP с числом выводов от 48 до 144.

3.2 Источник Питания

Для STM32 требуется один источник питания с напряжением в диапазоне от 2.0 В до 3.6 В. Встроенный регулятор используется для генерации напряжения 1.8 В для ядра Cortex. STM32 может имееть два дополнительных источника питания. Часы реального времени и несколько регистров расположены в отдельной области, которая может подключаться к батареи для сохранения данных при переходе микроконтроллера в выключенный режим. Если в устройстве эта функция не используется, $V_{BAT}$ должен быть подключен к $V_{DD}$.

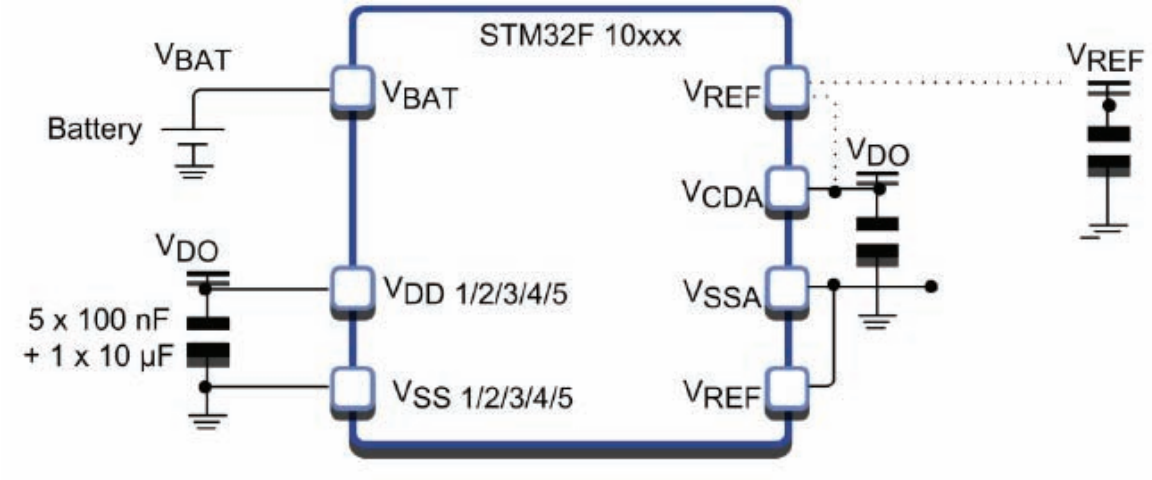



**STM32 работает от одного источника питания 2.0 В – 3.6 В. Небольшая область кристалла имеет возможность подключения к батареи, а также для АЦП нужен отдельный источник питания (только в 144-выводных корпусах).**

*Перевод к рисунку: Vdda domain – Питание от Vdda, A/D converter - АЦП, Temp.sensor - Термодатчик, Reset block – Схема сброса, PLL - ФАПЧ, Vdd domain – Питание от Vdd, I/O Rings – I/O, STANDBY circuitry (Wake-up logic, IWDG, RCC CSR reg.) – Схема STANDBY (Логика «пробуждения», регистры IWDG, RCC CSR), Voltage Regulator – Регулятор напряжения, V18 domain – Питание от V18, Core - Ядро, Memories - Память, Digital peripherals – Цифровые периферийные устройства, Low Voltage Detector – Детектор Падения Напряжения, Backup domain – Питание от резервного источника, LSE crystal 32K osc – внешний кварц 32 КГц, BKP register – BKP регистры, RCC BDCR register – регистр RCC BDCR, RTC – Часы реального времени.*

Второй дополнительный источник питания используется для АЦП. При использовании АЦП, напряжение источника питания $V_{DD}$ должно находиться в диапазоне от 2.4 В до 3.6 В. В 100-выводных корпусах есть дополнительные выводы, $V_{REF+}$ и $V_{REF-}$, для подключения опорного сигнала. Вывод $V_{REF-}$ подключается к $V_{DD}$, а

$V_{REF+}$ должен принимать значения от 2.4 В до $V_{DDA}$. Во всех других корпусах опорный сигнал врутренне соединен с выводом источника питания АЦП. Каждый источник напряжения должен иметь стабилизирующие конденсаторы, как показано ниже.

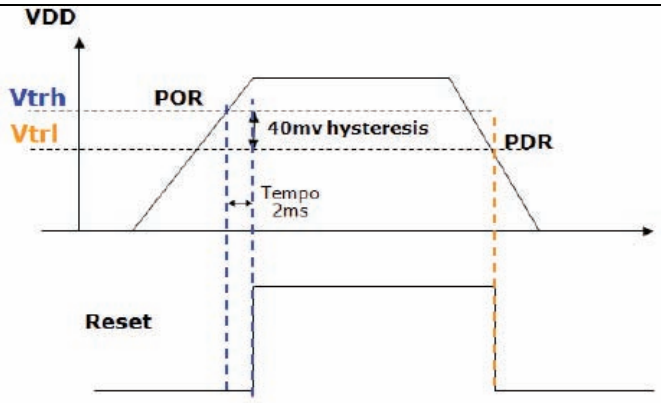



**Со встроенной схемой сброса и регулятором напряжения, для STM32 требуются только внешние конденсаторы.**

*Перевод к рисунку: Battery – Батарея.*

3.3 Схема Сброса

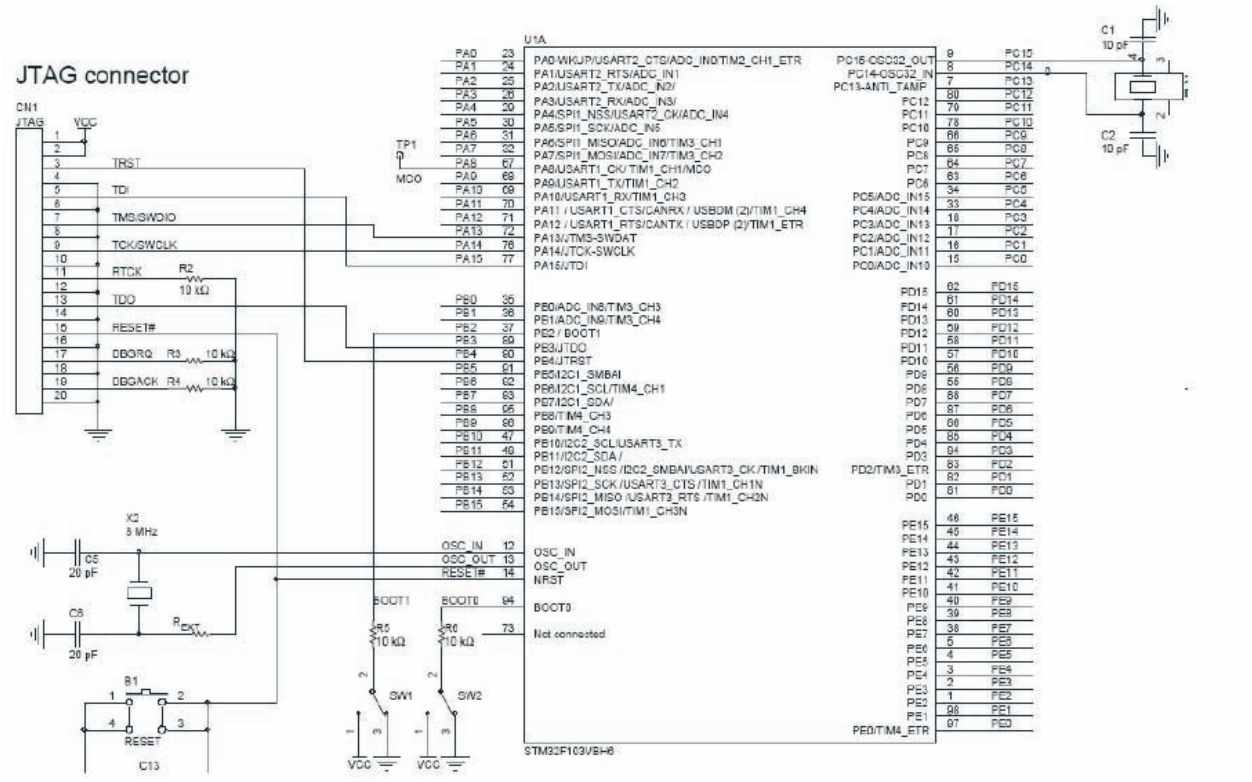
STM32 содержит встроенную схему сброса, которая держит микроконтроллер в режиме сброса пока $V_{DD}$ ниже 2.0 В с гистерезисом 40 мВ.




**Встроенные схемы сброса при включении питания и сброса при падении напряжения питанния гарантируют, что процессор будет работать только со стабильным источником питания. Внешних схем сброса не требуется.**

*Перевод к рисунку: Reset – Сброс, 40mv hysteresis – 40 мВ гистерезис.*

3.3.1.1.1 Базовая Принципиальная Схема




Собственно говоря, внешняя схема сброса не является необходимой частью дизайна STM32. Однако на этапе разработки, к выводу nRST можно подключить простую кнопку перезапуска. nRST соединяется также с портом JTAG, так что через отладочное устройство тоже можно осуществлять сброс микроконтроллера. STM32 содержит множество внутренних источников сброса, которые могут детектировать ошибки при выполнении различных условий, мы подробно рассмотрим это позднее.

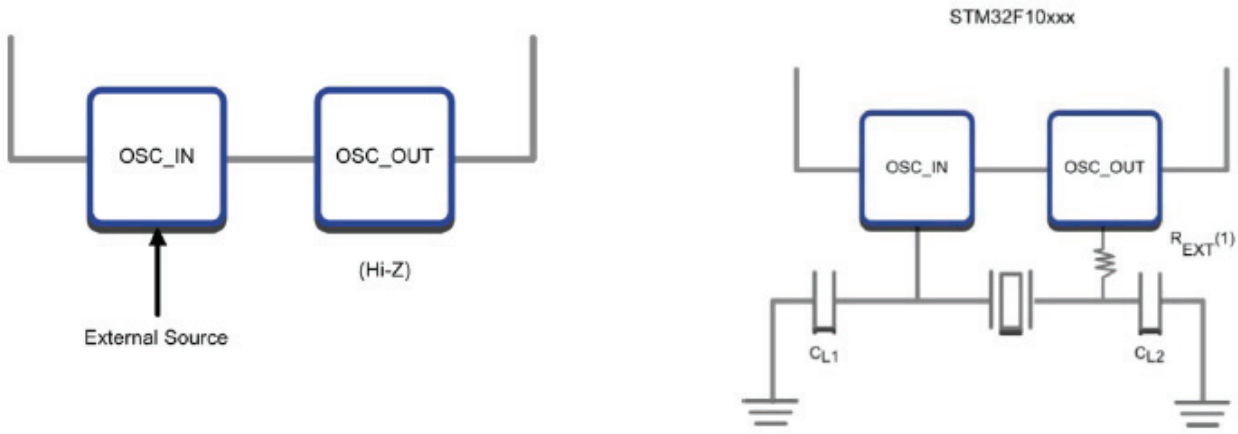
## 3.4 Осцилляторы

STM32 содержит встроенный RC-осциллятор, который может выступать в качестве источника тактового сигнала для внутренней ФАПЧ. Это позволяет микроконтроллеру работать с максимальной тактовой частотой 72 МГц. Встроенные осцилляторы не такие точные и стабильные, как внешние кварцы; следовательно, для большинства приложений потребуется хотябы один внешний источник тактового сигнала.

### 3.4.1 Высокочастотный Внешний Осциллятор

Основной внешний источник тактового сигнала ипользуется для получения сигнала синхронизации процессора Cortex и периферийных устройств STM32. Этот источник тактового сигнала называется High Speed External Oscillator (HSE Osc) и может быть кварцевым/керамическим резонатором или сигналом,

генерируемым пользователем. Если используется пользовательский сигнал, он должен иметь форму меандра, синуса или треугольную, длительность импульса 50% от периода, и максимальную частоту 25 МГц.




**External Oscillator может работать от кварца или внешнего сигнала синхронизации.**

*Перевод рисунка: External Source – Внешний Источник*.

Если используется внешний кварцевый/керамический резонатор, его частота должна быть в диапазоне от 4 МГц до 16 МГц. Так как внутренняя ФАПЧ умножает частоту HSE Osc на целое значение, частота внешнего сигнала должна иметь такую величину, чтобы при умножении на коэффициент ФАПЧ можно было легко получить 72 МГц.

### 3.4.2 Низкочастотный Внешний Осциллятор

STM32 может иметь второй внешний низкочастотный осциллятор, называемый Low Speed Oscillator(LSE Osc). Он используется для тактирования чаов реального времени и сторожевого таймера оконного типа.

Как и HSE Osc, LSE Osc может представлять собой внешний кварц или обеспечиваемый пользователем сигнал формы меандра, синуса или треугольной, с длительностью импульса 50% от периода. В обоих случаях LSE Osc должен работать с частотой 32.768 КГц, для того, чтобы обеспечить точную работу часов реального времени. Внутренний низкочастотный осциллятор также может использоваться для часов реального времени, но точность при этом будет ниже. Так что, если в приложении планируется использовать часы реального времени, нужно правильно определиться с низкочастотным осциллятором.

### 3.4.3 Вывод Тактового Сигнала

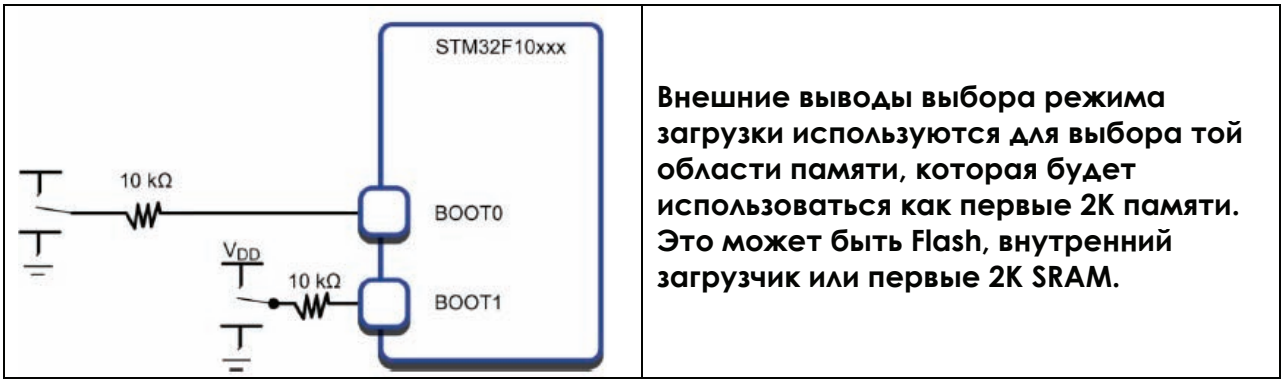
Один из выводов портов ввода/вывода общего назначения может быть сконфигурирован как вывод тактового сигнала микроконтроллера (MCO). В этом режиме вывод MCO может выдавать один из четырех внутренних тактовых сигналов. Мы коснемся этого подробнее при рассмотрении конфигурации внутренней системы синхронизации.

### 3.4.4 Выводы Выбора Режима Загрузки

STM32 может стартовать в одном из трех различных режимов. Эти режимы выбираются с помощью двух внешних выводов, BOOT0 и BOOT1. При изменении режима загрузки, микроконтроллер устанавливает определенные области карты памяти как начало карты памяти. Это позволяет выполнять код из пользовательской FLASH, внутренней SRAM или системной памяти. Если выбрана системная память, STM32 начнет выполнение загрузчика, запрограммированного на этапе изготовления микроконтроллера, через который перепрограммируется пользовательская flash память.

### 3.4.5 Режимы Загрузки

Для работы в обычном режиме BOOT0 должен быть заземлен. Если вы хотите использовать другие режимы, необходимо предусмотреть перемычки для выводов выбора режима загрузки, чтобы переключаться между различными режимами.




**Внешние выводы выбора режима загрузки используются для выбора той области памяти, которая будет использоваться как первые 2K памяти. Это может быть Flash, внутренний загрузчик или первые 2K SRAM.**

Обычно это требуется, если вы хотите осуществить обновление программного кода, используя встроенный загрузчик. По умолчанию, USART1 используется для загрузки данных из компьютера в микроконтроллер, поэтому, если планируется задействовать загрузчик, необходимо добавить в устройство драйвер RS232.

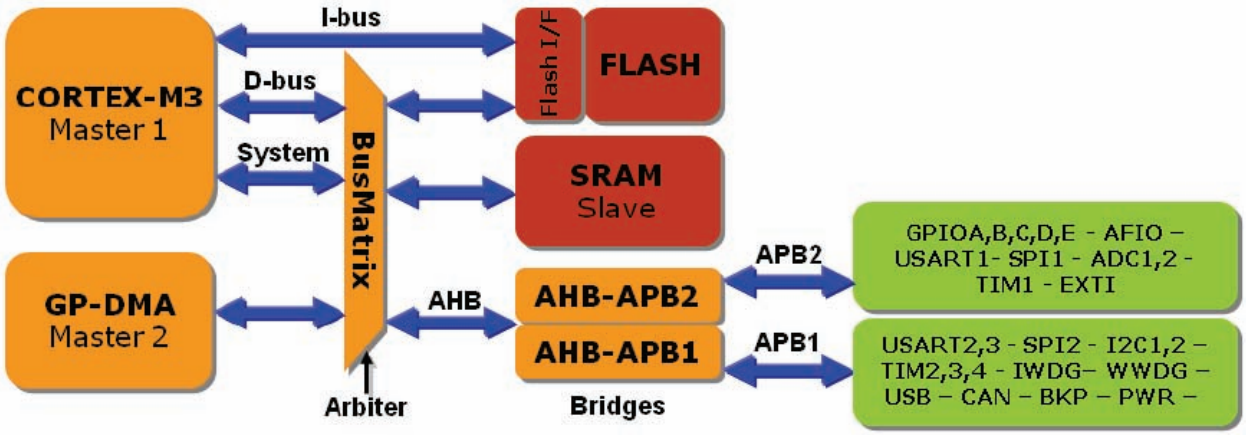
### 3.4.6 Отладочный Порт

Для того, чтобы подключать STM32 к отладочным средствам, нам необходимо добавить в схему отладочный порт. Отладочная система CoreSight поддерживает два стандарта коннекторов: 5-выводной JTAG и 2-выводной Cortex serial wire. В

обоих вариантах придется пожертвовать выводами микрокотнроллера для средств отладки. После сброса, ЦПУ Cortex инициализирует эти выводы как отладочный порт. С помощью регистров альтернативных функций можно переконфигурировать эти выводы отладочного порта обратно в выводы общего назначения. 5-выводной JTAG интерфейс подключается через 20-выводной коннектор IDC типа, который имеет стандартное расположение выводов для всех отладочных средств JTAG. Последовательный интерфейс serial wire использует Port A 13 для передачи данных и Port A 14 для передачи синхросигнала.

4. Системная Архитектура STM32

STM32 состоит из ядра Cortex, которое соединяется с Flash памятью через специальную Шину Команд. Шина Данных и Системная Шина Cortex соединяются с матрицей высокоскоростных шин ARM Advanced High Speed Busses (AHB). Внутренняя SRAM подключается напрямую к матрице шин AHB, так же как и модуль прямого доступа к памяти (DMA). Периферийные устройства располагаются на двух шинах ARM Advanced Peripheral Bussess (APB). Каждая из APB шин замыкается на матрицу шин AHB. Матрица шин тактируется с той же частотой, что и ядро Cortex. Однако шины AHB имеют отдельные делители частоты и могут тактироваться со сниженной частотой для снижения энергопотребления. Стоит отметить, что APB2 может работать с частотой 72 МГц, в то время как APB1 с частотой 36 МГц. И ЦПУ Cortex и DMA могут быть «хозяином» шины. Благодаря параллельным принципам организации матрицы шин, арбитраж необходим только в случае одновременного доступа к SRAM, шине APB1 или APB2. Как мы увидим в разделе, посвященному модулю DMA, арбитр шин предоставляет 2/3 времени доступа для DMA и 1/3 для ЦПУ Cortex.

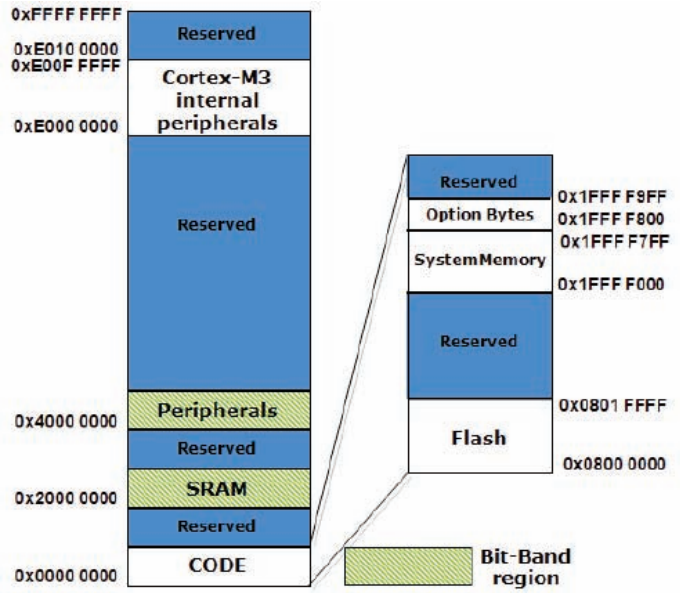



**Структура встроенных шин обеспечивает отдельную шину для команд и матрицу шин для организации доступа ядра Cortex и модуля DMA к ресурсам микроконтроллера.**

*Перевод к рисунку: Master 1 – Ведущий 1, Master 2 – Ведущий 2, GP-DMA – DMA общего использования, BusMatrix – Матрица Шин, Flash I/F – Интерфейс Flash, Slave - Ведомый, Arbiter - Арбитр, Bridges - Мосты.*

## 4.1 Распределение Памяти

Несмотря на то, что STM32 содержит множество внутренних шин, программисту отображается линейное 4 Гбайтное адресное пространство. Так как STM32 – это микроконтроллер на базе Cortex, карта памяти соответствует стандартному распределению памяти Cortex. Следовательно, память программ начинается с адреса 0x00000000. Встроенная SRAM начинается с 0x20000000 и вся она располагается в области с битовой сегментацией. Память пользовательских периферийных устройств начинается с адреса 0x40000000 и также расположена в области с битовой сегментацией. Регистры Cortex расположены начиная с с адреса 0xE0000000.




| BOOT Mode Selection Pins | | Boot Mode | Aliasing |
|---|---|---|---|
| BOOT1 | BOOT0 | | |
| x | 0 | User Flash | User Flash is selected as boot space |
| 0 | 1 | SystemMemory | SystemMemory is selected as boot space |
| 1 | 1 | Embedded SRAM | Embedded SRAM is selected as boot space |

**The STM32 memory map follows the Cortex standard. The first 2K of memory is mapped from FLASH, System Memory or SRAM depending on the condition of the boot pins.**

*Перевод к рисунку: Reserved - Зарезервированно, Cortex-M3 internal peripherals – Внутренние периферийные утройства Cortex, Peripherals – Периферийные устройства, CODE – ПРОГРАММНЫЙ КОД, Option Bytes – Опциональные Байты, System Memory – Системная Память, Bit-Band region – Область с Битовой Сегментацией.*

***The STM32 memory map follows … of the boot pins - Карта памяти STM32 соответствует стандартам Cortex. Первые 2 К памяти адресуются во Flash, Системную Память или SRAM в зависимости от состояния выводов выбора режима загрузки.***

*Перевод таблицы:*

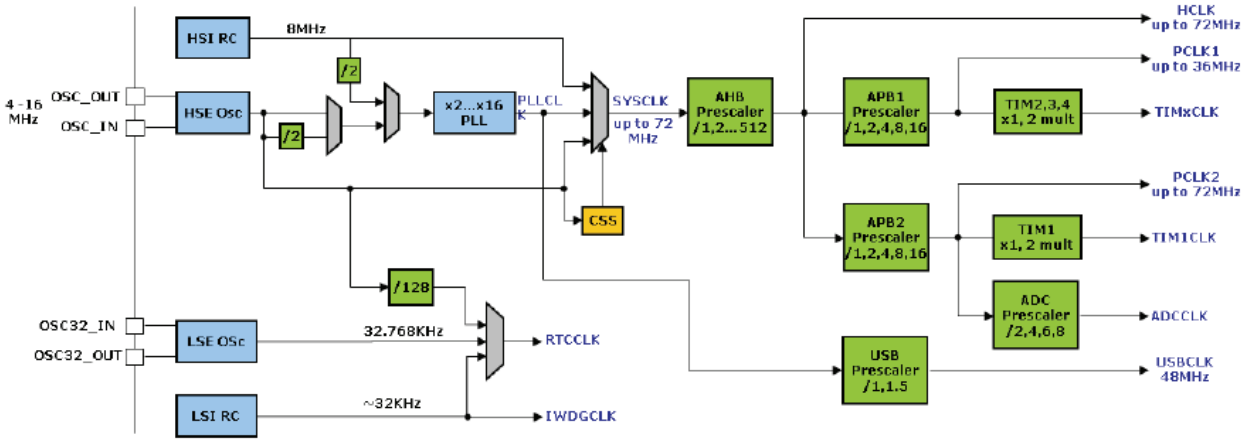
| | | |
|---|---|---|
| Выводы выбора | | |

| режима загрузки | | Режим Загрузки | Совмещение имен |
|---|---|---|---|
| BOOT1 | BOOT0 | | |
| x | 0 | Flash | Пользовательская Flash выбрана как загрузочная область |
| 0 | 1 | Системная Память | Системная Память выбрана как загрузочная область |
| 1 | 1 | Встроенная SRAM | Встроенная SRAM выбрана как загрузочная область |

Область FLASH памяти состоит из трех секций. Первая - это Пользовательская Flash память, начинается с адреса 0x00000000. Вторая – Системная Память, также называемая большим информационным блоком. В эти 4 K Flash памяти на этапе производства микроконтроллера запрограммирован загрузчик. Третья секция, начинаемая с адреса 0x1FFFF800, называемая малым информационным блоком, содержит группу опциональных байтов, которые позволяют осуществлять системные настройки STM32. Загрузчик разработан с целью загрузки кода через USART1 в Пользовательскую FLASH память. Для перевода STM32 в режим загрузчика, внешние выводы BOOT0 и BOOT1 должны быть переведены соответственно в низкое и высокое состояние. В таком состоянии выводов блок системной памяти отображается по адресу 0x00000000. После сброса, вместо выполнения программы пользователя из Пользовательской Flash, STM32 начнет выполнять код загрузчика. Приложение загрузчика для ПК доступно для скачивания на сайте компании ST. Это приложение используется для обмена информацией с загрузчиком, стирания и перепрошивки Пользовательской Flash памяти. Приложение для ПК также доступно в виде DLL, что позволяет написать свой собственный загрузчик для обновления областей памяти и для программирования микроконтроллеров на этапе серийного производства устройств. Выводы выбора режима загрузки позволяют также отображать встроенную SRAM по адресу 0x00000000 вместо Пользовательской Flash. Это дает возможность на этапе разработки загружать и исполнять программный код непосредственно из SRAM. При этом ускоряется процесс загрузки программы и уменьшается число циклов записи/стирания FLASH памяти, что увеличивает срок ее службы .

4.2 Максимизация Производительности

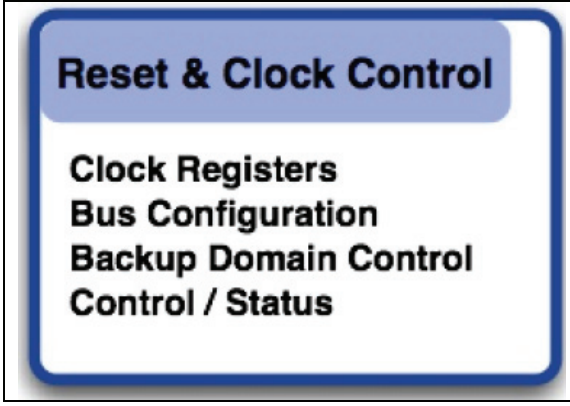
В дополнение к двум внешним осцилляторам, STM32 содержит два внутренних RC-осциллятора. После сброса, первоначальным источником тактового сигнала для ядра Cortex является Высокочастотный Внутренний Осциллятор, с номинальной частотой 8 МГц. Второй внутренний осциллятор представляет собой Низкочастотный Осциллятор, работающий на частоте 32.768 кГц. Этот осциллятор предназначен для часов реального времени и сторожевых таймеров.




**STM32 содержит сложную систему синхронизации с двумя внешними и двумя внутренними осцилляторами, а также системой ФАПЧ. Внешний Высокочастотный Осциллятор может контролироваться системой безопасности системы синхронизации**.

*Перевод к рисунку: Osc - Осц, KHz - кГц, MHz - МГц, PLL - ФАПЧ, up to – до, Prescaler – Делитель Частоты*.

Процессор Cortex может тактироваться от Внутреннего или Внешнего Высокочастотного Осциллятора или от внутренней ФАПЧ. ФАПЧ, в свою очередь, тактируется от Внутреннего, либо от Внешнего Высокочастотного Осциллятора. То есть STM32 может работать с частотой 72 МГц без использования внешних осцилляторов. Минус в том, что встроенные осцилляторы не такие точные и стабильный, как внешние. Поэтому для организации последовательных интерфейсов или реализации точных временных функций необходимо использовать внешний осциллятор. Независимо от того какой осциллятор используется, для получения 72 МГц для ядра Cortex нужно задействовать ФАПЧ. Все регистры конфигурации ФАПЧ и шин расположены в группе регистров Reset and Clock Control (RCC).

| | |
|---|---|
|  | **Модуль Reset And Clock Control управляет мостами между шинами системы синхронизации и резервной областью.** |



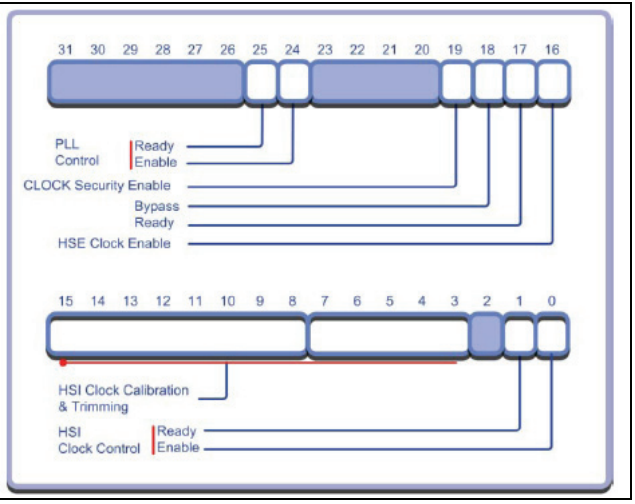
*Перевод к рисунку: Clock Registers – Регистры Системы Синхронизации, Bus Configuration – Конфигурация Шин, Backup Domain Control –Управление Резервной Областью , Control / Status – Управление / Статус*.

4.2.1 ФАПЧ

После сброса ЦПУ STM32 тактируется от осциллятора HSI. В этот момент внешний осциллятор выключен. Первое что нужно сделать для запуска STM32 на полной скорости заключается во включении внешнего осциллятора и ожидании пока он стабилизируется.

**После сброса STM32 работает от Внутреннего Высокочастотного Осциллятора. Внешний Осциллятор нужно включать.**

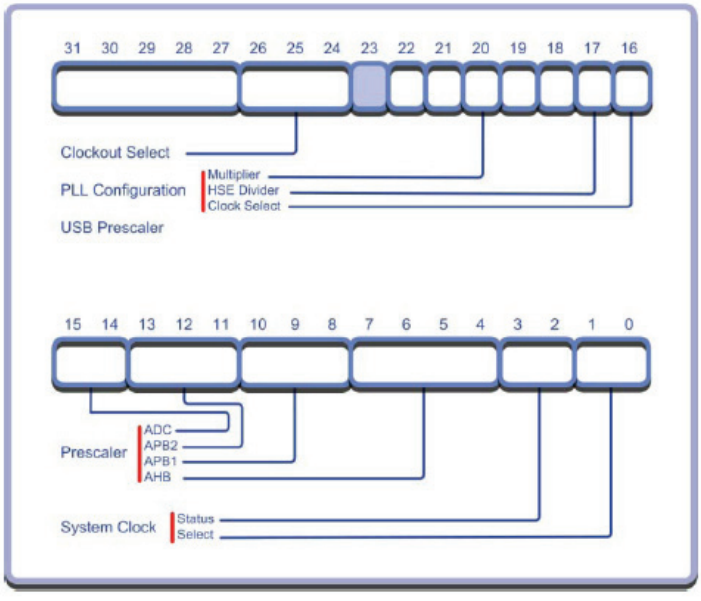
```
RCC -> CR |= 0x10000;   // Включаем HSE

// Ждемпока HSE стабилизируется
While (!(RCC -> CR &0x00020000))
{
   ;
}
```




Внешний осциллятор включается в регистре RCC_Control. Бит готовности сигнализирует о том, что осциллятор стабилизировался. Как только это случается, он может быть выбран в качестве тактового сигнала для ФАПЧ. Частота выходного сигнала ФАПЧ определяется значением коэффициента умножителя в регистре RCC_PLL_configuration. В случае 8 МГц-ового осциллятора, ФАПЧ должна умножать частоту входного сигнала на 9 для получения максимальных 72 МГц. Как только произведены настройки умножителя, ФАПЧ может быть включена с помощью регистра управления. После стабилизации ФАПЧ устанавливается бит готовности и выходной сигнал ФАПЧ может быть выбран как тактовый сигнал ЦПУ Cortex.

**После включения осциллятора HSE он может использоваться в качестве источника тактового сигнала для ФАПЧ. После стабилизации ФАПЧ, ее выходной сигнал может выбираться в качестве системного синхросигнала.**

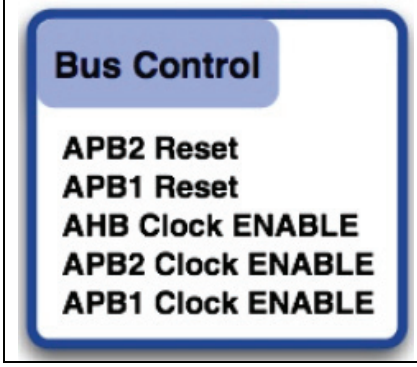
```
//HSE осциллятор, PLLx9
RCC -> CFGR = 0x001D0000; // Включаем ФАПЧ
RCC -> CR |= 0x01000000;
while (! (RCC -> CR & 0x02000000))
{
   ;
}

//Установка оставшихся битов управления
RCC -> CR |= 0x00000001;
```



| | |
|---|---|
| //Установка оставшихся конфигурационных битов<br>RCC -> CFGR \|= 0x005D0402; | |

4.2.1.1.1 Конфигурация Шин

Как только ФАПЧ выбрана в качестве источника тактового сигнала, ЦПУ Cortex будет работать на частоте 72 МГц. Для того, чтобы остальная часть микроконтроллера работала на нужной тактовой частоте, нужно сконфигурировать шины AHB и APB через их регистры.

| | |
|---|---|
|  | //Включение синхросигналов для шин AHB, APB1 и APB2<br>AHBENR = 0x00000014;<br>RCC -> APB2ENR = 0x00005E7D;<br>RCC -> APB1ENR = 0x1AE64807;<br><br>//Освобождение линий сброса периферийных устройств на шинах<br>// APB1 и APB2<br>RCC -> APB2RSTR = 0x00000000;<br>RCC -> APB1RSTR = 0x00000000; |

**После сброса многие периферийные устройства остаются в состоянии сброса с выключенными тактовыми сигналами. Перед использованием периферийного устройства включите его тактовый сигнал и выведите из состояния сброса.**

*Перевод к рисунку: Bus Control – Управдение Шинами, APB2 Reset – Сброс APB2, AHB Clock ENABLE – ВКЛЮЧЕНИЕ Тактирования AHB и т.д.*

4.2.2 Буфер FLASH

Если посмотреть на системную архитектуру STM32, можно увидеть, что ядро Cortex-M3 подключается к внутренней FLASH через отдельную шину I-Bus. Эта шина тактируется с той же частотой, что и ЦПУ, это значит, что при включенной ФАПЧ, ядро будет пытаться работать на 72 МГц. Так как ЦПУ Cortex является по существу одноцикловой машиной, оно будет пытаться обращаться ко Flash каждые 1.3 нс. При включении STM32, он работает от внутреннего осциллятора 8 МГц, так что время доступа к внутренней FLASH в этом случае не является поблемой. Однако при включенной ФАПЧ, когда она становится источником тактового сигнала, время доступа ко FLASH очень велико (35 нс), чтобы позволить ЦПУ Cortex работать на максимальной частоте. Для того, чтобы позволить ЦПУ Cortex работать на 72 МГц без задержек, FLASH память имеет буфер предварительной выборки, состоящий из двух 64-битных буферов. Каждый из этих буферов может осуществлять 64-битную выборку из FLASH памяти и затем передавать отдельные 16 или 32-битные команды в ЦПУ Cortex для исполнения. Эта технология очень хорошо работает в связке с условным выполнением команд системы Thumb-2 и прогнозированием

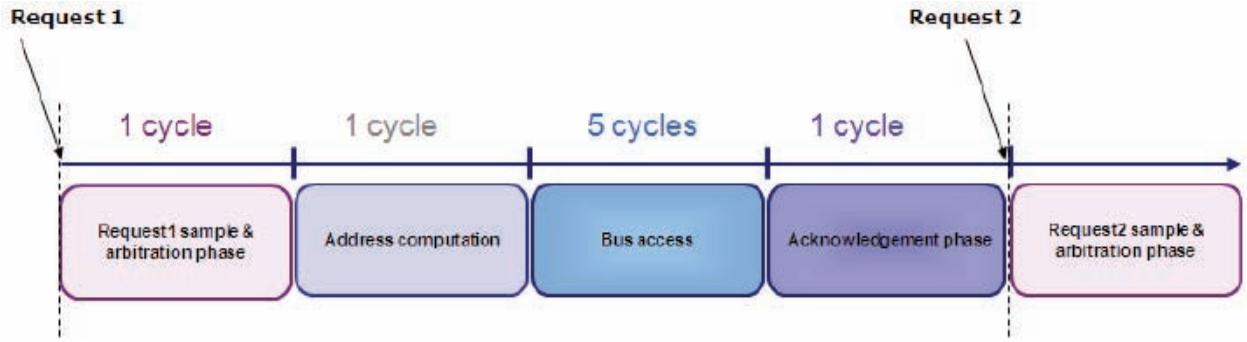
ветвления в конвейере Cortex. Во время нормальной работы, программисту не нужно предпринимать дополнительных мер предосторожности из-за буфера Flash. Однако нужно убедиться, что он включен перед активацией ФАПЧ как основоного источника тактового сигнала. Буфер FLASH управляется через регистр управления доступом FLASH. Кроме включения буфера предварительной выборки, нужно подстроить число циклов ожидания, требующихся для выборки 8 байт команд из FLASH памяти в буфер. Установить задержку можно следующим образом:

| | |
|---|---|
| 0 < SYSCLK < 24 МГц | 0 циклов ожидания |
| 24 < SYSCLK < 48 МГц | 1 цикл ожидания |
| 48 < SYSCLK < 72 МГц | 2 цикла ожидания |

В то время, пока ЦПУ исполняет команды из первой части буфера, идет заполение второй части буфера. Таким образом, на производительность ЦПУ не влияют задержки чтения из FLASH в буфер, и оно может работать на оптимальной частоте.

### 4.2.3 Прямой Доступ к Памяти

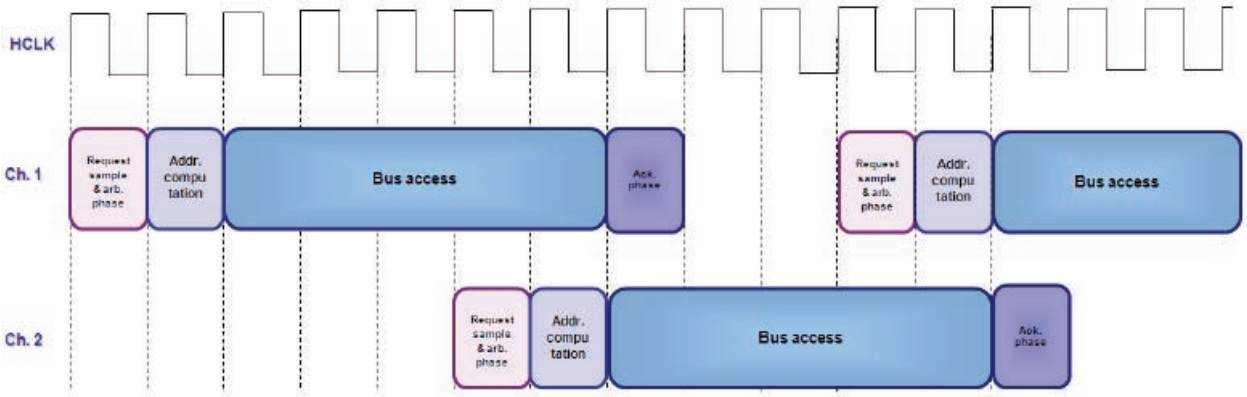
Несмотря на то, что ЦПУ Cortex может самостоятельно передавать данные между периферийными устройствами и внутренней SRAM, это может быть автоматизировано с помощью встроенного модуля Прямого Доступа к Памяти (DMA). Модуль DMA STM32 содержит семь независимо конфигурируемых каналов, которые могут осуществлять автономную передачу данных из памяти в память, из периферийного устройства в память, из памяти в периферийное устройство и из периферийного устройства в периферийное устройство. Передача данных из памяти в память осуществляется на столько быстро, насколько канал DMA может передавать данные. В случае с периферийным устройством, DMA находится под контролем выбранного периферийного устройства и данные передаются по требованию в или из периферийного устройства. При передачи массивов данных, модуль DMA может последовательно передавать данные в кольцевой буфер. Так как большая часть коммуникационных периферийных устройств не содержит FIFO буферов, модуль DMA может работать с буфером в SRAM. Модуль DMA был специально разработан для STM32 и поэтому оптимизирован для коротких, но частых посылок данных, что типично для микроконтроллерных приложений.




**DMA-посылка из памяти в память состоит из четырех фаз, каждая из которых выполняется за 1 цикл, кроме фазы доступа к шине, которая занимает 5 циклов для передачи одного слова**.

*Перевод к рисунку: 1 cycle – 1 цикл, 5 cycles – 5 циклов, Request - Запрос, Request1 sample & arbitration phase – Фаза выборки Запроса 1 и арбитража, Address computation – Вычисление адреса, Bus access – Доступ к шине, Acknowledgement phase – Фаза подтверждения,Request 2 – Запрос 2, Request2 sample & arbitration phase - Фаза выборки Запроса 2 и арбитража.*

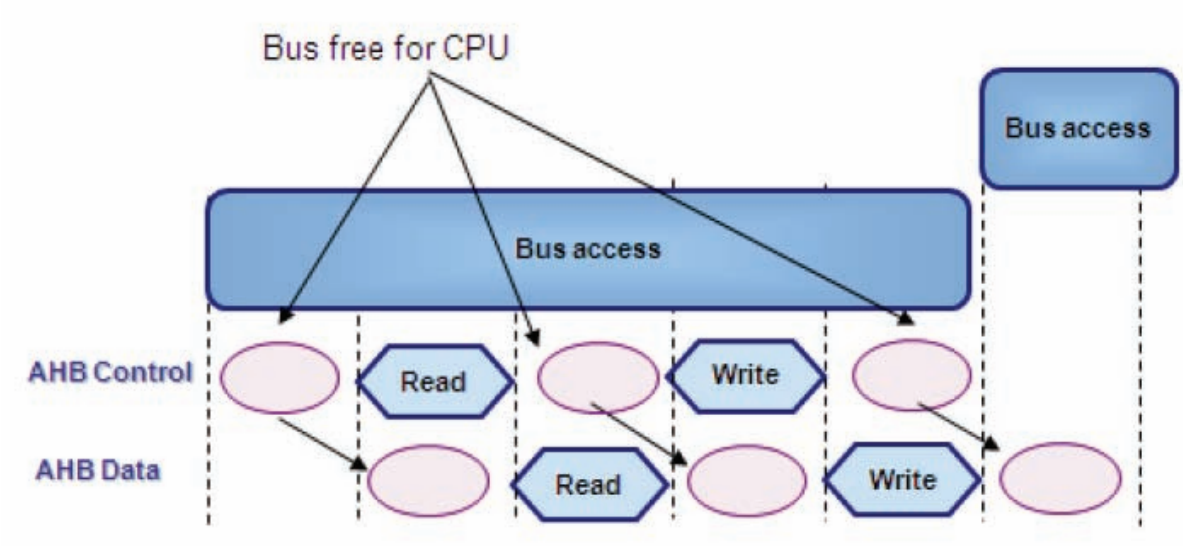
Каждая посылка DMA состоит из четырех фаз: фаза выборки и арбитража, фаза вычисления адреса, фаза доступа к шине и фаза подтверждения. Каждая фаза занимает один цикл, кроме фазы доступа к шине. Во время фазы доступа к шине (когда непосредственно осуществляется передача данных) для передачи одного слова требуется три цикла. Модуль DMA и ЦПУ Cortex работают чередуясь друг с другом, то есть DMA не будет блокировать ЦПУ и наоборот. Мы рассмотрим как это работает чуть позднее. Необходимо только задать приоритеты для различных каналов DMA. Каждому каналу DMA можно программно назначать один из четырех уровней приоритета. Во время фазы арбитража канал с большим уровнем приоритета получает доступ к шине. При возникновения двух запросов DMA с одинаковым приоритетом, доступ к шине получит канал с меньшим номером.



**Модуль DMA разработан для быстрых, но коротких посылок данных, что типично для малых встраиваемых систем. Модуль DMA занимает шину только во время фазы доступа к шине.**

*Перевод к рисунку доступен выше*.

Модуль DMA может осуществлять арбитраж и вычисление адресов в то время, когда один из каналов DMA производит доступ к шине. Как только активный канал заканчивает передачу данных по внутренней шне, следующий канал DMA готов начать передачу, в то время как предыдущий канал находится в фазе подтверждения. Таким образом, DMA не только передает данные быстрее ЦПУ, но также работает перемежаясь с ЦПУ и занимает шину только непосредственно для передачи данных.




**Во время фазы доступа к шине, три цикла свободны для ЦПУ. Таким образом, при организации передачи данных из памяти в память ЦПУ Cortex-M3 владеет шиной 60% времени, даже если DMA работает в режиме непрерывной передачи (это касается только передачи данных). Для извлечения команд у Cortex отдельная шина I-code**.

*Перевод к рисунку: AHB Control – Управление AHB, AHB Data – Данные AHB, Bus fre for CPU – Шина свободна для ЦПУ, Bus access – Доступ к шине, Read - Чтение, Write – Запись*.

В случае передачи данных из памяти в память, каждый канал DMA занимает шину данных только во время фазы доступа к шине и требует только пять циклов на передацу каждого слова данных. Один цикл на чтение данных и один цикл на запись данных, чередуются с циклами ожидания, когда шина освобождается для ЦПУ Cortex. Это означает, что модуль DMA поребляет максимум 40% пропускной

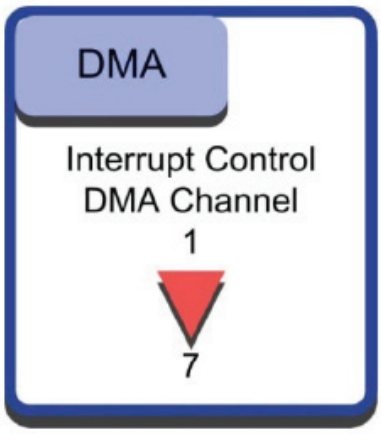
способности шины данных, даже в момент непрерывной передачи данных. В случае передачи данных из периферийного устройства в периферийное устройства и из периферийного устройства в память, ситуация немного сложнее. Передача по шине AHB занимает два цикла тактового сигнала AHB и передача по шине APB занимает два цикла тактового сигнала шины и 2 дополнитеьных цикла тактового сигнала AHB. Каждая посылка DMA состоит из двух посылок шин и свободного цикла. Например, посылка из SPI в SRAM состоит из посылки из SPI, посылки в SRAM и одного свободного цикла:

DMA посылка из SPI в SRAM = посылка из SPI (APB)+посылка в SRAM (AHB)+свободный цикл (AHB)

= (2 цикла APB + 2 цикла AHB) + 2 цикла AHB + 1 цикл AHB

= 2 цикла APB + 5 циклов AHB

Не забывайте, что все это связано только с передачей данных, все инструкции Cortex извлекаются с помощью отдельной шины I-Bus.



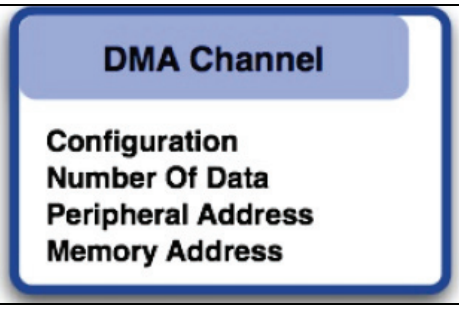

**Модуль DMA имеет семь каналов. Все каналы полностью ортоганальны.**

*Перевод к рисунку: Interrupt Control – Управление Прерываниями, DMA Channel – Канал DMA.*

Следующая хорошая новость относительно модуля DMA заключается в том, что он очень прост в использовании. Первое, что нужно помнить делать – переключать модуль DMA на свой тактовый сигнал и выводить из состояния сброса. Это производится через регистр включения тактового сигнала AHB в модуле управления сбросом и системой синхронизации.

```
RCC -> AHBENR |= 0x00000001;   // включение тактового сигнала DMA
```

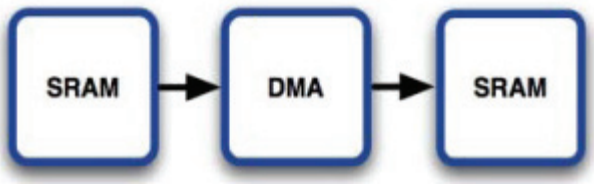
После включения модуля DMA, каждый из его каналов управляется четырьмя регистрами. Два регистра содержат адрес источника и адрес назначения для регистра периферийного устройства и области памяти. Размер посылки содержится в регистре «количество данных». Конфигурационный регистр определяет общие характеристики передачи данных через DMA.




**Каждый канал DMA управляется четырьмя регистрами и имеет три источника прерываний: передача завершена, передача наполовину завершена и ошибка передачи.**

*Перевод к рисунку: DMA Channel – DMA Канал, Configuration - Конфигурация, Number Of Data – Количество Данных, Peripheral Addres – Адрес Периферийного Устройства, Memory Address – Адрес Области Памяти.*

Каждому каналу DMA может быть назначен один из следующих уровней приоритета: «очень высокий», «высокий», «средний» и «низкий». Размер передаваемого слова может задаваться отдельно для памяти и периферийного устройств. Например, мы можем передать 32-битное слово в канал DMA из памяти (3 цикла) и затем эти данные в виде четырех 8-битных слов в регистр данных UART (всего 35 циклов вместо 64 циклов, если бы все передавалось в 8-битном формате). Можно также инеркментировать адреса памяти и периферийного устройства, так, чтобы данные передавались в массив памяти для обработки. Бит Transfer Direction Bit определяет направление передачи, то есть из памяти в периферийное устройство или из периферийного устройства в память. В случае передачи из памяти в память мы должны установить бит 14, для того, чтобы активировать «быструю на сколько возможно» передачу данных между двумя буферами SRAM. Несмотря на то, что каналы DMA могут работать в режиме по запросу, каждый из каналов поддерживает три вида прерываний: передача завершена, передача наполовину завершена и ошибка передачи. После того, как DMA полностью сконфигурирован, нужно установить бит Channel Enable Bit и начинатся передача данных. Передача данных из памяти в память может быть организована следующим образом:

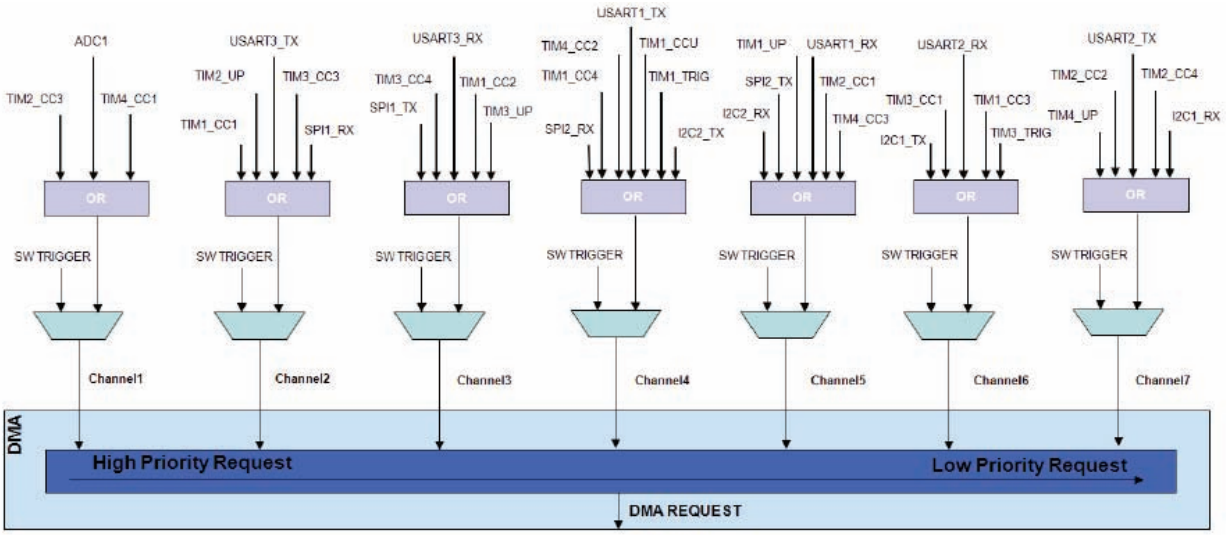



**Программный код осуществляет простую передачу по каналу DMA из памяти в память с использованием встроенного таймера для подстчета числа затрачиваемых циклов тактового сигнала.**

```
DMA_Channell -> CCR      = 0x00007AC0;          // задаем передачу из памяти в память
DMA_Channell -> CPAR     = (unsigned int) src_arry; // устанавливаем источник и назначение
DMA_Channell -> CMAR  = (unsigned int) arry_dest;
DMA_Channell -> CNDTR = 0x000A;                 // устанавливаем размер посылки
```

```
TIM2 -> CR                = 0x00000001;              // запускаем таймер
DMA_Channell -> CRR       |= 0x00000001;                  // начинаем передачу DMA
while (!(DMA -> ISR & 0x00000001))                   //ждем завершения передачи
{
;
}
TIM2 -> CR1 = 0;                                     // останавливаем таймер
TIM2 -> CNT = 0;                                     // сбрасываем счетчик
TIM2 -> CR1 = 0;                                     // перезапускаем таймер
 for (index = 0; index < 0xA; index++)                   // повтор операции с
использованием ЦПУ
{
arry_dest [index] = arry_src [index];
}
TIM2 -> CR1 = 0;                                     // останавливаем таймер
```

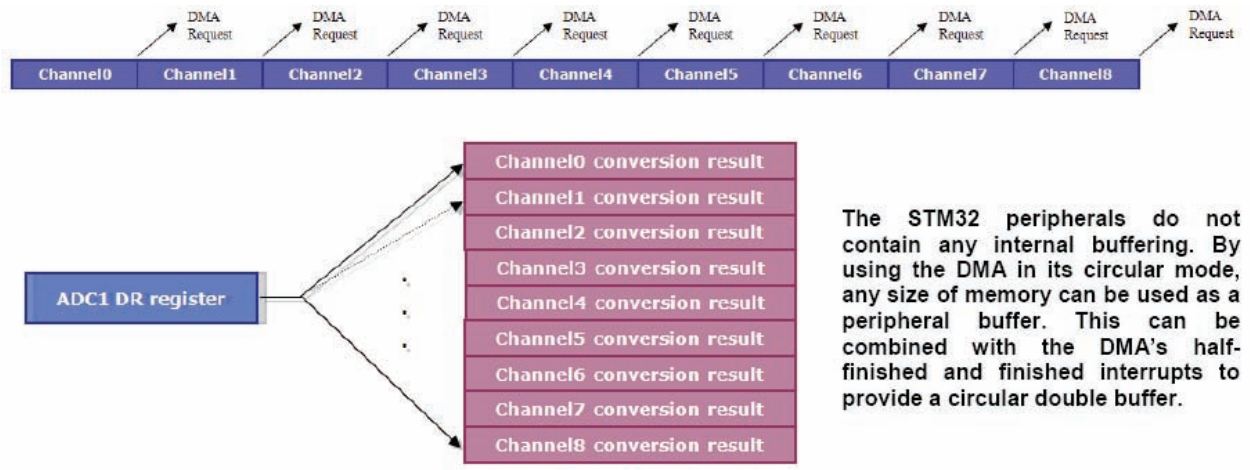
Вышеприведенная программа осуществляет передачу десяти слов данных между двумя массивами памяти SRAM, сначала с использованием DMA, затем с использованием ЦПУ Cortex. В обоих случаях в начале передачи запускается таймер. В этом примере передачу с использованием DMA выполнилась за 220 циклов, а с использованием ЦПУ – за 536.




**Каждому периферийному устройству с поддержкой DMA назначается определенный канал. При включении периферийного устройства, оно начинает управлять передачей данных через DMA. Таким образом, периферийное устройство принимает и передает данные без привлечения ЦПУ**.

*Перевод к рисунку: High Priority Request – Запрос с Высоким Приоритетом, Low Priority Request – Запрос с Низким приоритетом, Channel - Канал, DMA Request – DMA Запрос.*

Передача из памяти в память осуществляется при инициализации областей памяти и  копирования массивов данных, но большую часть времени DMA каналы используются для передачи данных между памятью и различными периферийными устройствами. В этом случае, каждому каналу DMA соответствует определенный набор периферийных устройств. Сначала необходимо инициализировать периферийное устройство и разрешить для него поддержку DMA, затем мы должны настроить определенный для этого периферийного устройства канал DMA для передачи данных по запросу от этого периферийного устройства. Мы подробно рассмотрим АЦП позднее, здесь просто отметим, что в режиме простого преобразования оно может осуществлять непрерывное 10-битное преобразование с сохранением результата в один регистр. Без модуля DMA ЦПУ Cortex пришлось бы постоянно обрабатывать прерывания по завершению преобразования АЦП. Т.е. нам бы пришлось затрачивать на это ценный ресурс ЦПУ. Однако при использовании DMA, по завершению преобразования каждой выборки, АЦП запрашивает DMA для передачи данных. DMA передает данные АЦП по инкрементирующемуся адресу в SRAM. Таким образом, данные АЦП могут быть обработаны после накопления в памяти нужного числа выборок.




*Перевод к рисунку: Channel – Канал, DMA Request – Запрос DMA, Channel0 conversion result – Результат преобразования Канала 1,* ***The STM32 peripherals … a circular double buffer – Периферийные устройства STM32 не содержат внутренних буферов. Благодаря использованию DMA в кольцевом режиме, можно организовать буфер любого размера в памяти. Прерывания DMA по завершению передачи и завершению на половину можно использовать для организации двойного кольцевого буфера***.

Для того, чтобы сделать этот процесс более эффективным, мы можем реализовать круговой буфер для непрерывной записи в него данных АЦП. Используя прерывания по завершению передачи и завершению наполовину, мы

можем организовать двойной буфер. По заполнению первой части буфера, будет сгенерировано прерывание и мы можем приступить к обработке этих данных, в то время как DMA заполняет вторую половину буфера. Как только заполнится вторая половина, мы обрабатываем данные из нее, пока DMA обновляет содержимое первой части буфера. Все остальные периферийные устройства с поддержкой DMA работают аналогичным образом. Стоит отметить, что коммуникационные периферийные устройства имеют отдельные DMA каналы для приема и передачи. Например, SPI может одновременно передавать данные в двух направлениях.

## 5. Периферийные Устройства

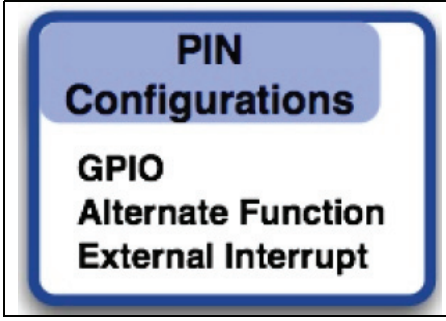
В этой главе мы проведем обзор периферийных устройств существующих версий STM32. Для удобства разделим периферийные устройства на две группы: периферийные устройства общего назначения и коммуникационные периферийные устройства. Все периферийные устройства STM32 имеют высокий уровень сложности и тесно сопряжены с DMA модулем. Каждое периферийное устройство реализует в себе дополнительные аппаратные функции, позволяющие минимизировать участие ЦПУ в управлении этим периферийным устройством. Другими словами, имеется множество «умных» функций для автоматизации управления периферийными устройствами и снижения нагрузки на ЦПУ.

### 5.1 Периферийные Устройства Общего Назначения

Периферийные устройства общего назначения STM32 включают: порты ввода/вывода общего назначения, контроллер внешних прерываний, аналого-цифровой преобразователь, таймеры общего назначения и с расширенными функциями, часы реального времени с регистром резервирования и выводом оповещения о вскрытии корпуса устройства.

#### 5.1.1 Порты Ввода/Вывода Общего Назначения GPIO

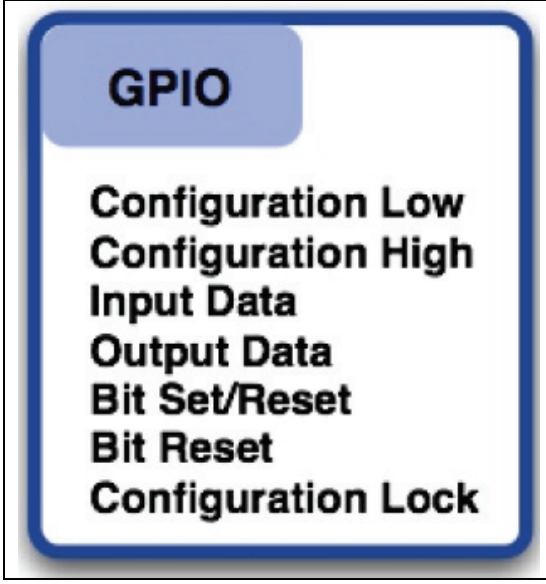
STM32 хорошо оснащен портами ввода/вывода общего назначения GPIO и может иметь до 80 двунаправленных выводов. Они сгруппированны как пять портов, каждый из которых содержит 16 линий ввода/вывода.




**Каждый цифровой вывод может конфигурироваться как вывод GPIO или как вывод с альтернативной функцией. Каждый вывод одновременно может работать как одна из 16 линий внешних прерываний**.

*Перевод к рисунку: PIN Configurations – Конфигурация Выводов, GPIO - GPIO, Alternate Function – Альтернативная Функция, External Interrupt – Внешнее Прерывание*.

Порты обозначаются символами латинского алфавита от A до E и все толерантны к напряжению 5 В. Многие из внешних выводов микроконтроллера вместо выполнения функций ввода/вывода могут быть переключены на обслуживание пользовательских периферийных устройств, таких как USART или I2C. Кроме этого, модуль внешних прерываний позволяет распределить 16 линий внешних прерываний между линиями портов ввода/вывода общего назначения.



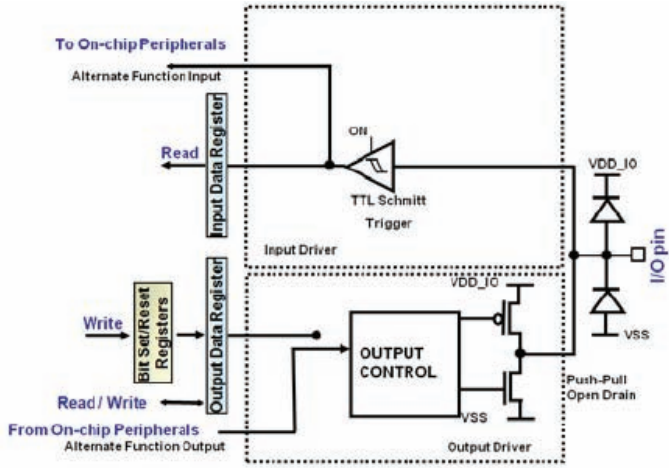

**Отдельные выводы каждого порта GPIO можно конфигурировать как вход или выход различных задающих устройств. Порты содержат регистры, в которые можно записывать информацию в формате слов или манипулировать с их битовыми полями. После осуществления настроек, регистры можно заблокировать**.

Перевод к рисунку: Configuration Low Configuration High – Конфигурационные Регистры, Input Data – Входные Данные, Output Data – Выходные Данные, Bit Set/Reset – Сброс/Установка Битов, Bit Reset – Сброс Битов, Configuration Lock – Блокировка Конфигурации.

Каждый порт GPIO содержит два 32-битных конфигурационных регистра. Эти два регистра объединяются для получения 64-битного конфигурационного регистра. Среди этих 64 бит для каждого вывода выделяется четыре бита, задающие его характеристики. Четырехбитовое поле состоит из двухбитового поля выбора режима и двухбитового конфигурационного поля. Поле выбора режима позволяет пользователю определять вывод как вход или выход, в то время как конфигурационное поле задает управляющую характеристику вывода.

Вывод порта можно определить как вход или выход, а также выбрать его нагрузочную характеристику. В случае, если вывод определен как вход, с помощью встроенного регистра можно «подтягивать» вывод к «земле» или напряжению питания. В случае, если вывод определен как выход, он может конфигурироваться как двухтактный или с открытым стоком. Каждый выход может также быть сконфигурирован для работы на частоте до 2 МГц, 10 МГц или 50 МГц.

| Configuration Mode | CNF1 | CNF0 | MOD1 | MOD0 |
|---|---|---|---|---|
| Analog Input | 0 | 0 | 00 | |
| Input Floating (Reset State) | 0 | 1 | | |
| Input Pull-Up | 1 | 0 | | |
| Input Pull-Down | 1 | 0 | | |
| Output Push-Pull | 0 | 0 | 00: Reserved<br>01: 10 MHz<br>10: 2 MHz<br>11: 50 MHz | |
| Output Open-Drain | 0 | 1 | | |
| AF Push-Pull | 1 | 0 | | |
| AF Open-Drain | 1 | 1 | | |




*Перевод рисунка: To On-chip Peripherals – В Периферийные Устройства, Alternate Function Input – Вход Альтернативной Функции, Read – Чтение, Input Data Register – Входной Регистр Данных, Write - Запись, Bit Set/Reset Register – Регистр Установки/Сброса Битов, Output Data Register – Регистр Выходных Данных, Read/Write – Чтение/Запись, From On-chip Peripherals – От Периферийных Устройств, Alternate Function Output – Выход Альтернативной Функции, ON - ВКЛ, TTL Shmitt Trigger – Триггер Шмидта, Input Driver – Входной Драйвер, OUTPUT COTROL – УПРАВЛЕНИЕ ВЫХОДОМ, Output Driver – Выходной Драйвер, Push-Pull - Двухтактный, Open Drain – С Открытым Стоком, I/O pin – Вывод I/O.*

Перевод таблицы:

| Конфигурация | CNF1 | CNF0 | MOD1 | MOD0 |
|---|---|---|---|---|
| Аналоговый вход | 0 | 0 | 00 | |
| Плавающий вход (состояние сброса) | 0 | 1 | | |
| Подтянуты вход | 1 | 0 | | |
| Зазаемленный вход | 1 | 0 | | |
| Двухтактный выход | 0 | 0 | 00: Зарезервировано<br>01: 10 МГц<br>10: 2 МГц<br>11: 50 МГц | |
| Выход с открытым соком | 0 | 1 | | |
| Двухтактный выход альтернативной функции | 1 | 0 | | |
| Выход с открытым стоком альтернативной функции | 1 | 1 | | |

После того, как конфигурация порта закончена, установленные параметры могут быть защищены от изменения. В регистре блокировки конфигурации каждому выводу соответствует один блокировочный бит, устанавливая который, мы предотвращаем запись в поля режима и конфигурации соответствующего

вывода. Когда все необходимые блокировочные биты установлены, блокировка активируется записью последовательности 1, 0, 1 в 16-ый бит регистра блокировки конфигурации. После этого, если блокировка была успешно активирована, при считывании этого бита, должны быть получены значения 0 и 1. Регистры входных и выходных данных позволяют получить доступ к выводам портов. Атомарная битовая манипуляция осуществляется при помощи технологии битовой сегментации Cortex через регистры входных и выходных данных, либо при помощи двух специальных регистров. Регистр установки/сброса битов представляет собой 32-битный регистр. Старшие 16 бит соответствуют выводам порта. Запись логической единицы в определенную битовую область сбросит соответствующий вывод порта. Аналогично, запись логической единицы в любой из младших 16 битов регистра приведет к установке соответствующего вывода порта в высокое состояние. Второй регистр манипуляции с битами позволяет сбрасывать биты. Это 16-битный регистр. Запись логической единицы в любой из младших битов приведет к сбросу соответствующего вывода порта. Комбинация регистров портов, технология битовой сегментации и регистры манипуляции с битами позволяют легко управлять портами STM32 и строить эффективные приложения с интенсивным вводом/выводом данных.

#### 5.1.1.1 Альтернативные Функции

Регистры альтернативных функций позволяют переназначать выводы портов GPIO для выполнения альтернативных функций периферийных устройств. Для гибкости проектирования аппаратной части, имеющиеся функции периферийных устройств могут назначаться одному или нескольким выводам.

| | |
|---|---|
|  | ??????? не отсюда, явно |

**A late-arriving high priority interrupt will pre-empt a low priority interrupt without incurring an additional stacking overhead.**

*Перевод к рисунку: EVENT CONTROL – УПРАВЛЕНИЕ СОБЫТИЯМИ, Remap & Debug I/O Config – Переназначение и Конфигурация Отладочных Выводов.*

Альтернативные функции STM32 управляются через регистр переназначения и отладки портов ввода/вывода. Каждое из пользовательских периферийных устройств (USART, CAN, таймера, I2C и SPI) имеет одно или два битовых поля для выбора комбинации выводов микроконтроллера с которыми оно будет работать. После выброра альтернативной функции вывода, нужно в конфигурационном регистре GPIO произвести переключение с ввода/вывода на альтернативную функцию. Регистр переназначения также управляет конфигурацией выводов JTAG.

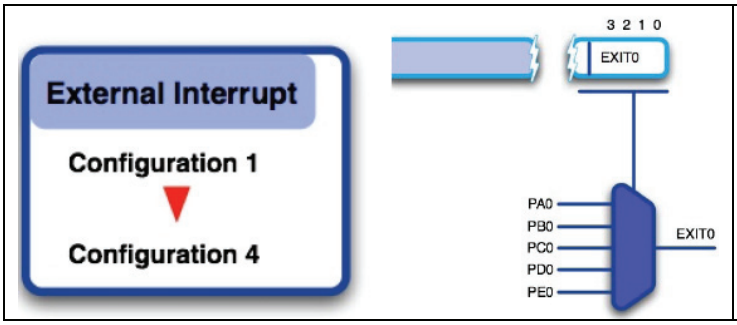
После сброса, порт JTAG включается с отключенной функцией трассировки данных. JTAG может быть переключен на seial wire (2-выводный отладчик) или выключен, а неиспользующиеся выводы в любом из случаев могут работать как GPIO.

### 5.1.1.2 Прерывание Для Внешних Устройств

Процессор Cortex может генерировать импульс для выведения другого микроконтроллера из спящего режима. Обычно вывод, на который подается этот импульс, подключается к выводу «пробуждения» второго микроконтроллера STM32. Прерывание для внешних устройств генерируется через выполнение команды SEV Thumb-2. Регистр управления событиями STM32 используется для маршрутизации этого сигнала на нужный вывод GPI. Регистр содержит поля для выбора порта и вывода внутри порта. После того, как порт и вывод выбраны, устанавливается бит разрешения генерирования прерывания для внешних устройств.

### 5.1.2 Внешние Прерывания

Модуль внешних прерываний содержит 19 линий прерываний, соединенных с векторами прерываний через NVIC. Шестнадцать из этих линий соединяются с выводами GPIO и могут генерировать прерывания по переднему, заднему или обоим фронтам входного сигнала. Остальные три линии соединяются с часами реального времени, USB и модулем детектирования напряжения питания. NVIC имеет отдельные вектора прерываний для линий EXTI 0-4, часов реального времени, USB и модуля детектирования напряжения питания. Остальные линии объединены в группы: линии 5-9 и линии 10-15; и соединены с двумя векторами прерываний. Внешние прерывания широко используются при управлении напряжением питания STM32, например, для вывода микроконтроллера из режима STOP, в котором оба основных осциллятора выключены. Модуль EXTI может генерировать прерывания, как для выхода из режима WFI, так и для WFE.




**STM32 содержит 16 линий внешних прерываний, каждая из которых может соединяться с одним из выводов микроконтроллера.**

*Перевод к рисунку: External Interrupt –Внешнее Прерывание, Configuration 1 – Конфигурация 1, Configuration 4 – Конфигурация 4.*

16 линий внешних прерываний EXTI могут соединяться с выводами GPIO в различных комбинациях. Настройка осуществляется через четыре конфигурационных регистра. В этих регистрах для каждой линии EXTI выделено четырехбитовое поле. Эти поля позволяют соединять каждую из линии EXTI с любым из пяти портов. Например, нулевая линия может соединяться с выводом 0 портов A, B, C, D или E. Такая схема подключения позволяет соединять линию прерывания с любым внешним выводом. Внешнее прерывание может использоваться вместе с альтернативной функцией вывода.


```
// Распределение внешних прерываний между вводами
AFIO -> EXTICR [0] = 0x00000000;
// Разрешение Внешних прерывание
EXTI -> IMR        = 0x00000001;
// Разрешение «пробуждающих» событий
EXTI -> EMR        = 0x00000000;
// Выбор срабатывания по заднему фронту
EXTI -> FTSR       = 0x00000001;
// Выбор срабатывания по переднему фронту
EXTI -> RTSR       = 0x00000000;
// Разрешение прерываний в NVIC
NVIC -> Enable [0] = 0x00000040;
NVIC -> Enable [1] = 0x00000000;
```

**Внешние прерывания могут генерироваться по переднему, заднему или обоим фронтам импульсов на выводах GPIO**.

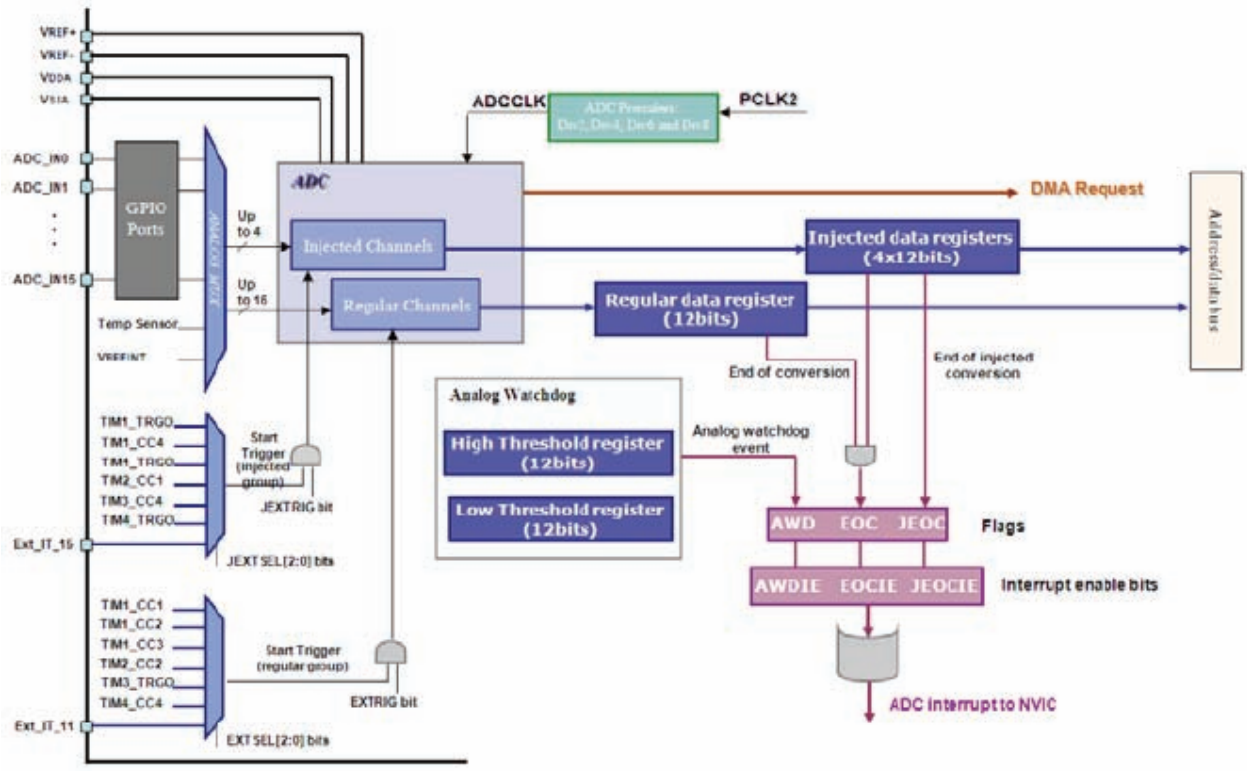
*Перевод к рисунку: Interrupt mask – Маскирование Прерываний, Event Mask – Маскирование Событий, Rising Trigger Selection – Выбор Срабатывания по Переднему Фронту, Falling Trigger Selection – Выбор Срабатывания по Заднему Фронту, Software Interrupt Event – Програмное Прерывание, Pending – Ожидание*.

После того как конфигурация регистров EXTI произведена, любое из внешних прерываний можно настроить на генерацию по переднему или заднему фронту. Также можно программно вызвать EXTI прерывание, установив соответствующий бит в регистре прерываний.

### 5.1.3 АЦП

STM32, в зависимости от версии, может содержать до двух независимых аналогово-цифровых преобразователей. АЦП имеет отдельный источник питания с напряжением в диапазоне от 2.4 В до 3.6 В, в зависимости от типа корпуса. Опорное напряжение АЦП подключается внутри к напряжению питания АЦП, или подается на специальный вывод. АЦП обладает разрешающей способностью 12-бит и скоростью преобразования 1МГц. АЦП содержать до 18 мультиплексированных каналов, 16 из которых выводятся наружу. Остальные два

используются для подключения встроенного температурного датчика и внутреннего опорного сигнала.

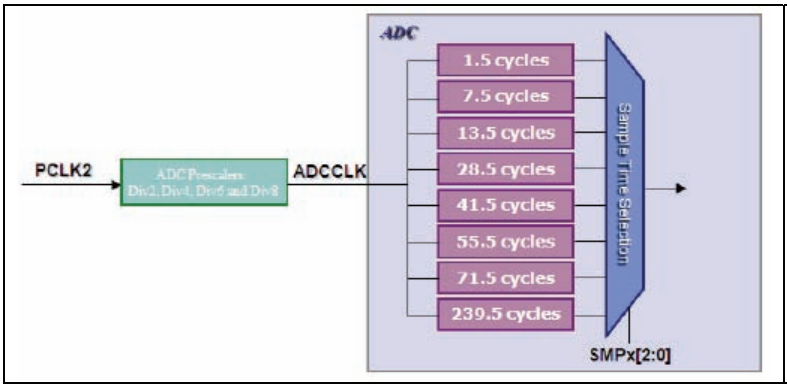



**STM32 содержит 12-битный АЦП со скоростью 1МГц, со встроенной аналоговой сторожевой схемой и температурным датчиком.**

*Перевод к рисунку: GPIO Ports – Порты GPIO, Up to - До, ADC - АЦП, Injected Channels – Инжектированные Каналы, Regular Channels – Регулярные Каналы, Analog Watchdog – Аналоговая Сторожевая Схема, High Threshold register – Регистр Верхней Границы, Low Threshold register – Регистр Нижней Границы, 12bits – 12 бит, Injected data registers – Регистры Инжектированных Каналов, Regular data registers – Регистры Регулярных Каналов, End of conversion – Завершение Преобразования, End of injected conversion – Завершение Преобразования Инжектированных Каналов, Analog watchdog event – Прерывание от сторожевой схемы, Flags - Флаги, interrupt enable bits – биты разрешения прерываний, ADC interrupt to NVIC – Прерывание от АЦП в NVIC, Start Trigger (injected group) – Запуск (инжектированной группы), Start Trigger (regular group) – Запуск (регулярной группы), ADC Prescalers – Делители Частоты АЦП, DMA Request – Запрос DMA, Adress/data bus – Шина Адреса/Данных.*

### 5.1.3.1 Время Преобразования и Группы Преобразования

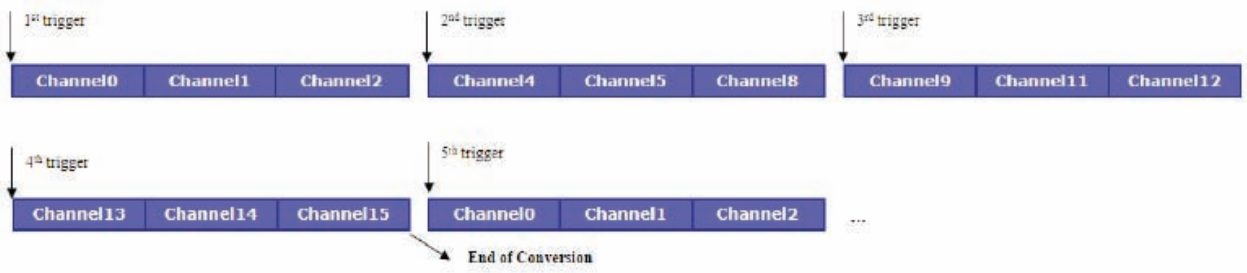
После того, как конфигурация АЦП произведена, можно задать время преобразования отдельно для каждого канала. Доступны восемь дискретных значений времени преобразования в диапазоне от 1.5 до 239.5 циклов.




**Для каждого канала можно индивидуально задать время преобразования.**

*Перевод к рисунку: ADC Pescalers – Делители Частоты АЦП, cycles – циклов, Sample Time Selection – Выбор Времени Преобразования*.

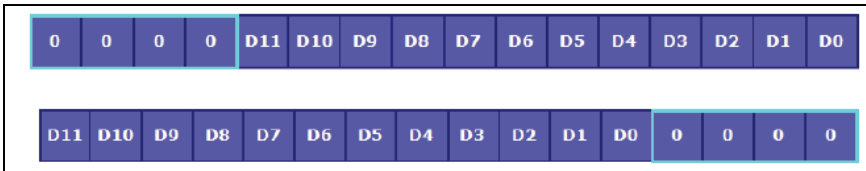
АЦП имеет два основных режима преобразования: регулярный и инжектированный. В регулярном режиме осуществляется аналогово-цифровое преобразование одного или группы каналов в цикле. В группе может быть до 16 каналов. Кроме того, можно задавать порядок следования каналов в цикле преобразования. За один цикл преобразования один и тот же канал может быть задействован несколько раз. Преобразование группы каналов в регулярном режиме может быть запущено программно или аппаратно с помощью прерываний от таймеров или первой линии внешних прерываний. После запуска, групповое преобразование может осуществляться непрерывно, либо в прерывистом режиме, когда производится преобразование выбранной группы каналов, затем преобразование останавливается до прихода следующего сигнала запуска группового преобразования.




**Преобразование группы каналов в регулярных каналов осуществляется в непрерывном циклическом режиме, либо в прерывистом режиме, когда группа выбранных каналов обрабатывается по приходу сигнала запуска**.

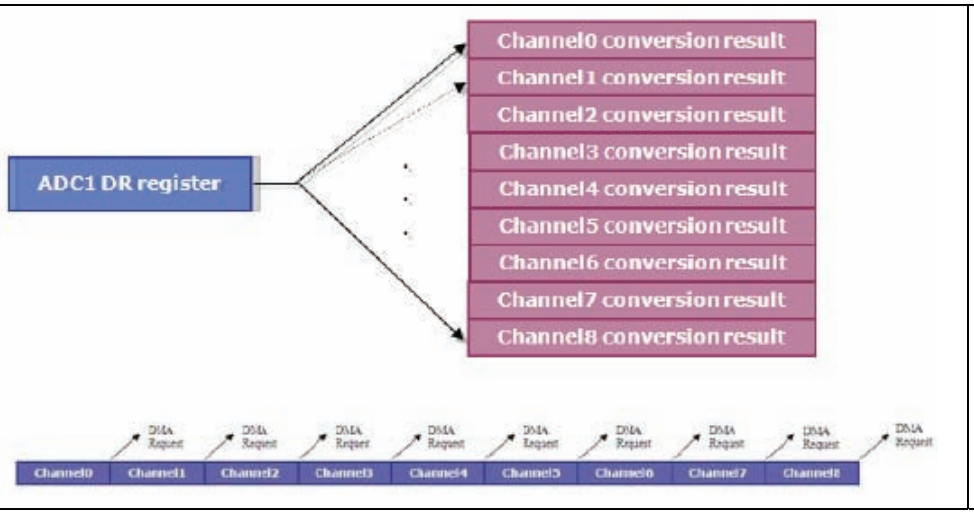
*Перевод к рисунку: Channel - Канал, trigger - запуск, End of Conversion – Завершение преобразования*.

Каждый раз после обработки группы каналов, результаты преобразования сохраняются в регистре результата и может генерироваться прерывание. 12-битный результат сохраняется в 16-битном регистре и может выравниваться по левому или правому краю.




**12-битный результат может выравниваться по левому или правому краю в 16-битном регистре результата**.

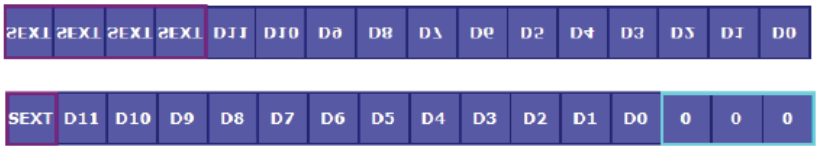
Для АЦП1 выделен специальный канал DMA, который может использоваться для переноса результата преобразования из регистра результата в буфер памяти. Таким образом, результаты преобразования группы каналов копируются в память и генерируется DMA прерывание по завершению цикла преобразования группы каналов. Правильное решение - сделать буфер двойного размера и использовать DMA в режиме генерации прерывания по завершению передачи наполовину. DMA при этом можно использовать в режиме кольцевого буфера для размещения множества результатов преобразования под аппаратным управлением.




**АЦП1 имеет поддержку DMA, которая позволяет автоматически передавать результаты преобразования в определенный пользователем буфер в SRAM**.

*Перевод к рисунку: Channel0 conversion result – Результат преобразования Канала 1, DMA Request – Запрос DMA, Channel – Канал.*

Инжектированная группа каналов - это последовательность максимум из четырех каналов, преобразование которых может запускаться программно или аппаратно. При запуске инжектированной группы останавливается преобразование регулярных каналов. После завершения преобразования группы инжектированных, продолжается преобразование регулярных каналов. Как и в случае с регулярными каналами, инжектированные каналы могут обрабатываться в любой последовательности, причем любой из каналов может обрабатываться несколько раз за один цикл. В отличие от регулярных, каждый из инжектированных каналов имеет свой регистр результата и регистр смещения.

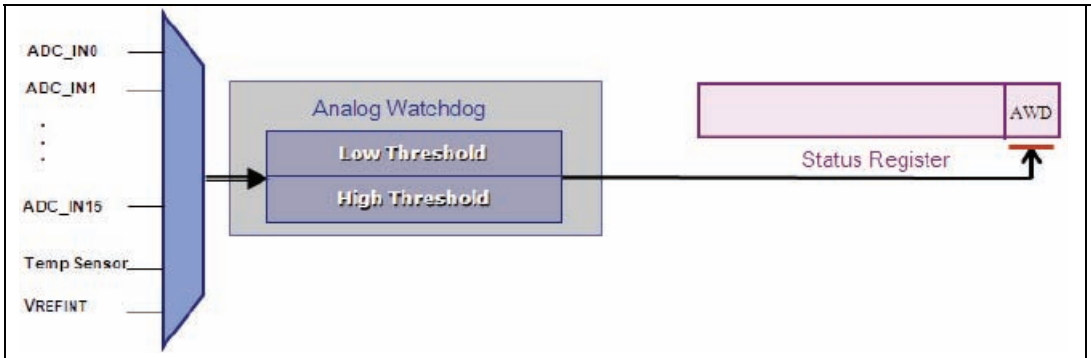



**Результаты преобразования группы инжектированных каналов имеют знак и могут выравниваться по левому или правому краю**.

В регистр смещения записывается 16-битное числовое значение, которое автоматически вычитается из результата преобразования АЦП. Если результат представляет собой отрицательное значения, в регистре результата инжектированного канала один бит отводится для знака. Результат можно выровнять по левому или правому краю.

### 5.1.3.2 Аналоговая Сторожевая Схема

В дополнение к двум режимам преобразования, АЦП содержит аналоговую сторожевую схему. Для сторожевой схемы задаются верхний и нижний пороги для детектирования выхода напряжения из заданных условий. В результате чего сторожевая схема может генерировать прерывание. Сторожевая схема также может использоваться для мониторинга отдельных регулярных или инжектированных каналов, либо всех сразу каналов. Кроме отслеживания напряжения, аналоговая сторожевая схема может использоваться как детектор пересечения нулевого напряжения.



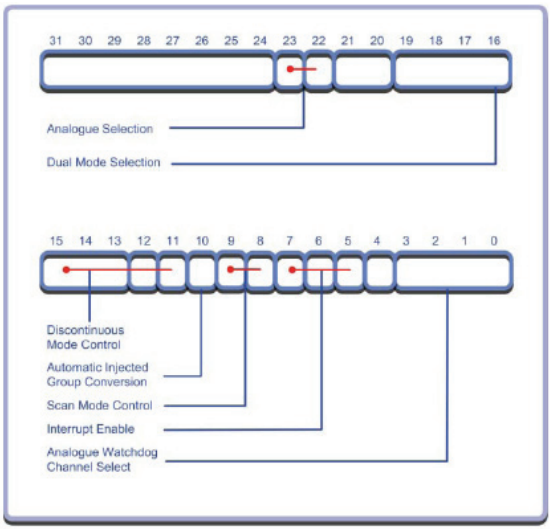

**Аналоговая сторожевая схема может отслеживать напряжение определенного канала или всех каналов и детектировать выход из заданных границ напряжения.**

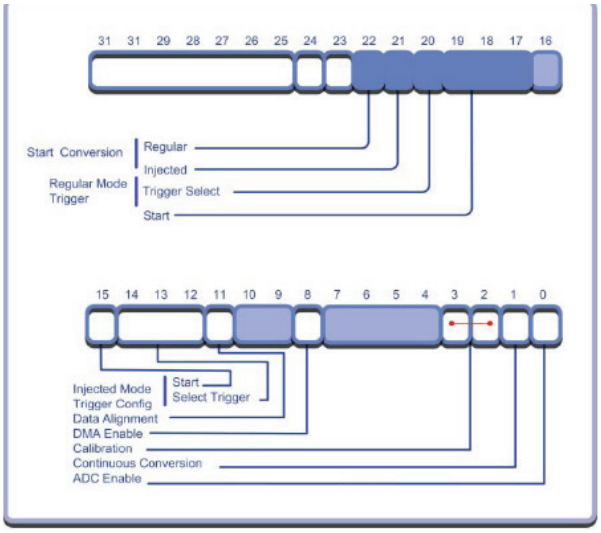
*Перевод к рисунку: Temp Sensor - Термодатчик, Analog Watchdog – Сторожевая схема, Low Threshold – Нижний Порог, High Threshold – Верхний Порог, Status Register – Регистр Статуса.*

### 5.1.3.3 Базовая Конфигурация АЦП


**Регистры АЦП делятся на шесть групп. Регистры статуса и управления определяются конфигурацией АЦП**.

АЦП содержит блоки регистров для конфигурации времени преобразования отдельных каналов, последовательности преобразования регулярных и инжектированных каналов, значение смещения в инжектированной группе и порогов сторожевой схемы. Общие настройки АЦП производятся через регистры статуса и управления.




**Два регистра управления определяют режим работы АЦП. Режим преобразования одного канала, управляемый через прерывание показан ниже**.

```
ADC1 -> CR2= 0x005E7003;          //Включение АЦП в режиме непрерывного
преобразования
ADC1 -> SQR1      = 0x0000;       // установка длины последовательности в
единицу
ADC1 -> SQR2      = 0x0000;       // выбор нулевого канала для
преобразования
ADC1 -> SQR3      = 0x0001;
ADC1 -> CR2!=0x005E7003;          // перезапись бита включения
```

```
ADC1 -> CR1=0x000100;              // Запуск преобразования regular каналов,
                разрешение
                                   // прерывания АЦП
NVIC -> Enable [0] = 0x00040000; // Разрешение прерывания АЦП
NVIC -> Enable [1] = 0x00000000;
```

В обработчике прерывания АЦП считывается регистр результата преобразования и копируется в регистр портов ввода/вывода.

```
 void ADC_IRQHandler (void)

{

GPIO -> ODR = ADC -> DR << 5; // Копирование результата АЦП в регистр порта

}
```

Если прерывания не используются, DMA канал может передавать результат АЦП напрямую в регистр порта ввода/вывода.

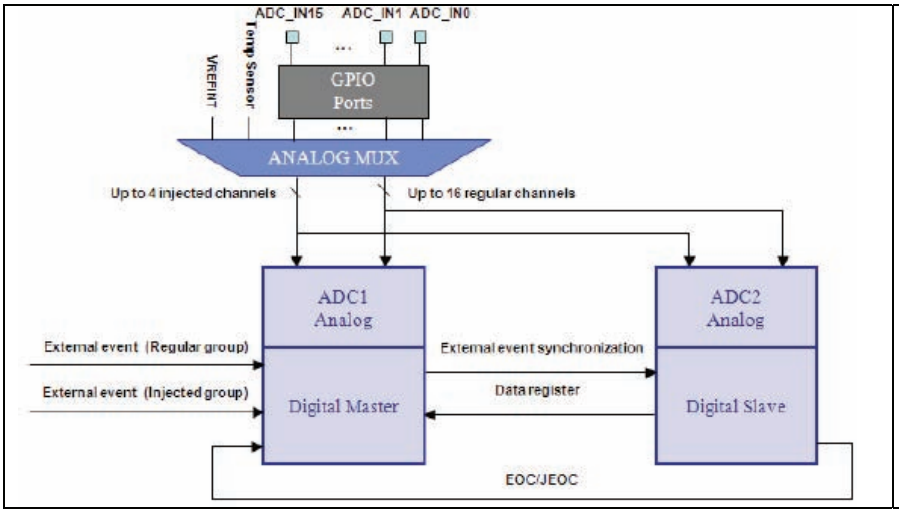
```
DMA_Channel1 -> CCR     = 0x00003A28;         // Кольцевой режим,
                        инкрементирование адресов
                                   // отключено
// Загрузка адреса назначения в регистр периферийного устройства, регистр данных порта GPIO
DMA_Channel1 -> CPAR = (unsigned int) 0x4001244C;
// Загрузка адреса источника в регистр памяти
DMA_Channel1 -> CMAR  = (unsigned int) 0x40010C0C;
DMA_Channel1 -> CNDTR = 0x1; // Загрузка количества передаваемых слов
DMA_Channel1 -> CCR       != 0x00000001; // Включение передачи DMA
```

В АЦП должна быть включена поддержка DMA.

```
ADC -> CR2  != 0x0100;
```

### 5.1.3.4 Режим Двойного Преобразования

Несмотря на то, что STM32 представляет собой не дорогой микроконтроллер общего применения, его АЦП является многофункциональным. Потребуется время, чтобы разобраться со всеми его функциями. АЦП STM32 может аппаратно выполнять такие операции, для осуществления которых на традиционном АЦП потребовалось бы дополнительное программное обеспечение. Кроме этого, в вариантах STM32 с двумя АЦП поддерживается режим двойного преобразования.

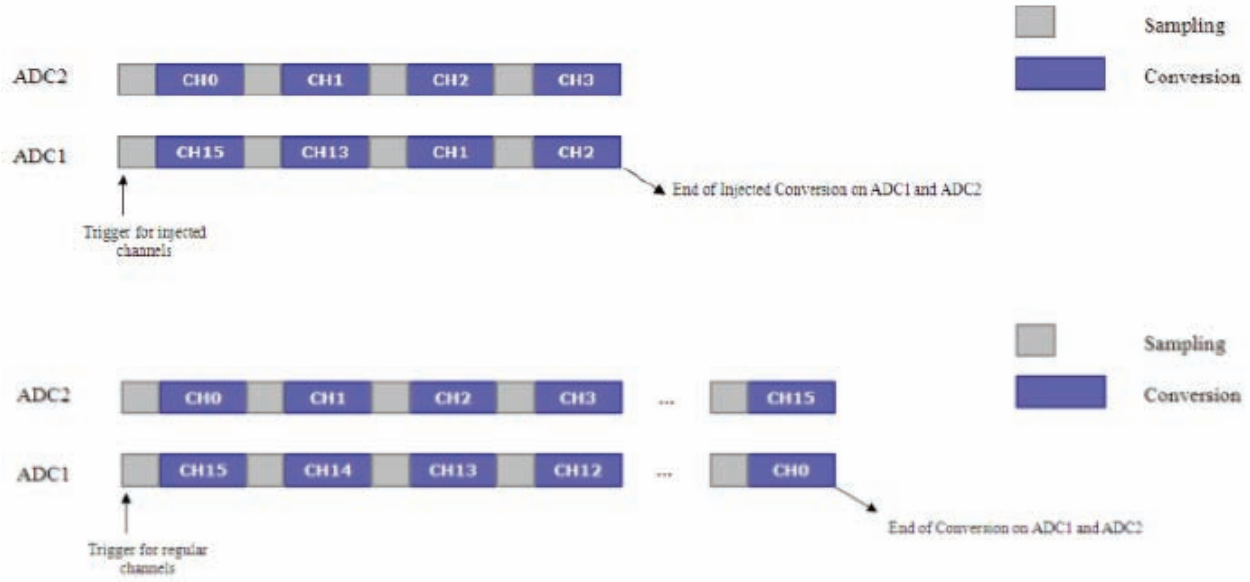



**В режиме двойного преобразования АЦП производится синхронизация двух конвертеров. При этом возможны восемь дополнительных режимов**.

Перевод к рисунку: Up to 4 injected channels – До 4 инжектированных каналов, Up to 16 regular channels – До 16 регулярных каналов, External event (Regular group) – Внешнее событие (Регулярная группа), External event (Injected group) – Внешнее событие (Инжектированная группа), ADC1 Analog – Аналоговая часть АЦП1, ADC2 Analog – Аналоговая часть АЦП2, Digital Master – Цифровая часть Ведущего, Digital Slave – Цифровая часть Ведомого, External event synchronization – Синхронизация внешних событий, Data register – Регистр данных.

В режиме двойного преобразования АЦП1 управляет АЦП2 в одном из восьми дополнительных режимов.

5.1.3.4.1 Режимы Одновременного Преобразования Инжектированных Каналов и Регулярных Каналов

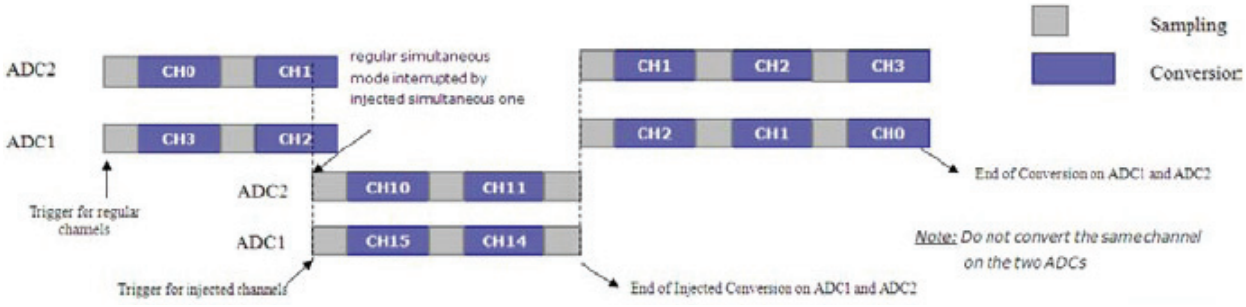



*Перевод к рисунку: ADC1 – АЦП1, ADC2 – АЦП2, Trigger for injected channels – Запуск преобразования инжектированных каналов, Trigger for regular channels – Запуск преобразования регулярных каналов, End of Injected Conversion on ADC1*

*and ADC2 – Завершение преобразования инжектированных каналов АЦП1 и АЦП2, End of Conversion on ADC1 and ADC2 – Завершение преобразования регулярных каналов АЦП1 и АЦП2, Sampling - Выборка, Conversion – Преобразование*.

В первом дополнительном режиме двойного преобразования производится синхронизация преобразования инжектированных каналов двух АЦП, во втором - синхронизация преобразования регулярных каналов. Эти два режима широко используются при одновременном измерении значений двух величин, например, напряжения и тока.

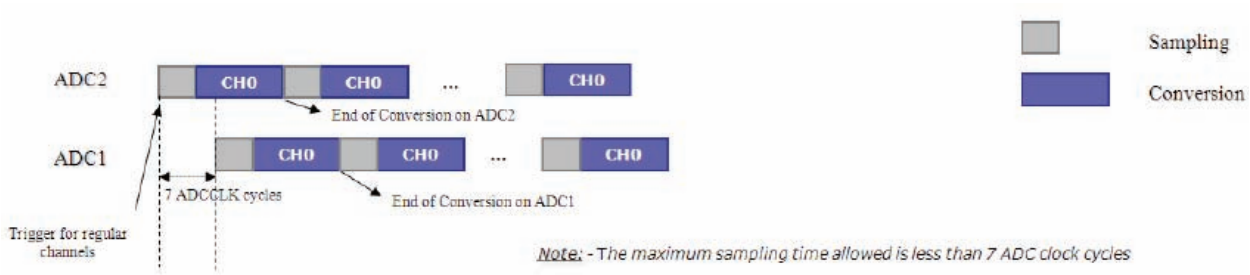
5.1.3.5 Комбинированный Режим Одновременного Преобразования Регулярных/Инжектированных Каналов

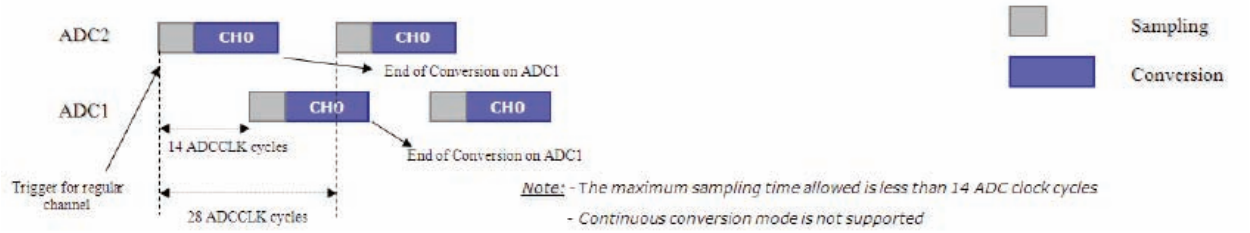



*Перевод к рисунку: ADC1 – АЦП1, ADC2 – АЦП2, Trigger for injected channels – Запуск преобразования инжектированных каналов, Trigger for regular channels – Запуск преобразования регулярных каналов, regular simultaneous mode interrupted by injected simultaneous one – одновременное преобразование регулярных каналов прерывается одновременным преобразованием инжектированных каналов, End of Injected Conversion on ADC1 and ADC2 – Завершение преобразования инжектированных каналов АЦП1 и АЦП2, End of Conversion on ADC1 and ADC2 – Завершение преобразования регулярных каналов АЦП1 и АЦП2, Sampling - Выборка, Conversion – Преобразование, Note: Do not convert the same channel on the two ADCs – Замечание: Нельзя обрабатывать один канал сразу двумя АЦП.*

В этом объединенном режиме синхронизируется преобразование регулярных и инжектированных групп каналов обоих АЦП.

5.1.3.6 Режимы Преоразования с Больним и Малым Смещением Каналов

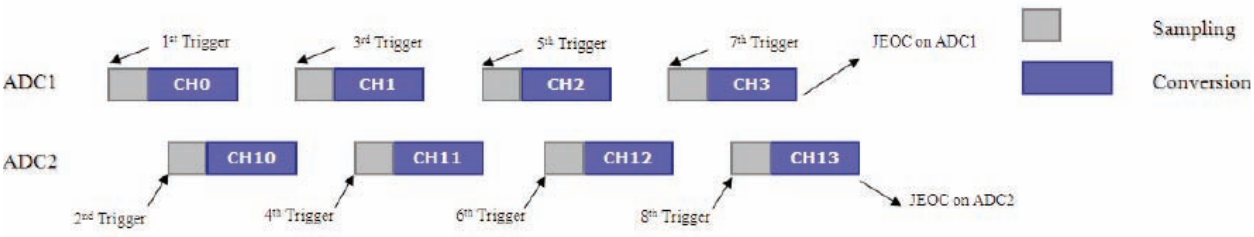



*Перевод к рисунку: ADC1 – АЦП1, ADC2 – АЦП2, Trigger for regular channels – Запуск преобразования регулярных каналов, End of Conversion on ADC1 – Завершение преобразования АЦП1, End of Conversion on ADC2 – Завершение преобразования АЦП2, Sampling - Выборка, Conversion – Преобразование, 7 ADCCLK cycles – 7 циклов ADCCLK, 14 ADCCLK cycles -14 циклов ADCCLK , 28 ADCCLK cycles – 28 циклов ADCCLK, Note: The maximum sampling time allowed is less than 7 ADC cycles – Замечание: Максимально допустимое время преобразования меньше 7 циклов тактового сигнала АЦП, Note: The maximum sampling time allowed is less than 14 ADC cycles – Continuous conversion mode is not supported - Замечание: Максимально допустимое время преобразования меньше 14 циклов тактового сигнала АЦП – Режим непрерывного преобразования не поддерживается.*

В режимах преобразования с большим и малым смещением каналов преобразование регулярных групп каналов осуществляется синхронно, но в отличие от режимов одновременного преобразования, запуск АЦП1 производится с задержкой. В режиме с малым смещением задержка составляет семь циклов тактового сигнала АЦП, в режиме большим смещением –четырнадцать циклов. Оба этих режима могут использоваться для увеличения общей скорости преобразования с помощью объединения двух АЦП.

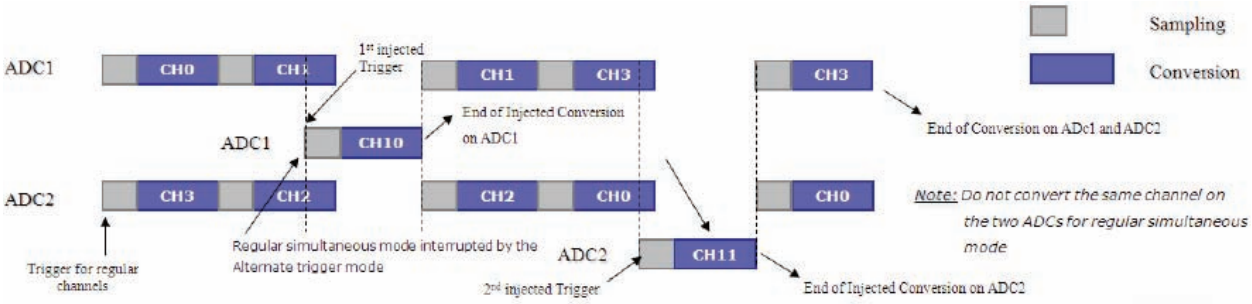
5.1.3.7 Режим Поочередного Запуска




*Перевод к рисунку: ADC1 – АЦП1, ADC2 – АЦП2, Trigger - Запуск, JEOC on ADC1 – Завершение преобразования инж. группы АЦП1, JEOC on ADC2 – Завершение преобразования инж. группы АЦП2, Sampling - Выборка, Conversion – Преобразование.*

В режиме поочередного запуска сначала аппартно запускается преобразование инжектированного канала АЦП1, затем преобразование инжектированного кана АЦП2 и т.д. до завершения преобразования всей группы АЦП1 и АЦП2.

5.1.3.8 Объединенный режим одновременного преобразования регулярных каналов и поочередного запуска.

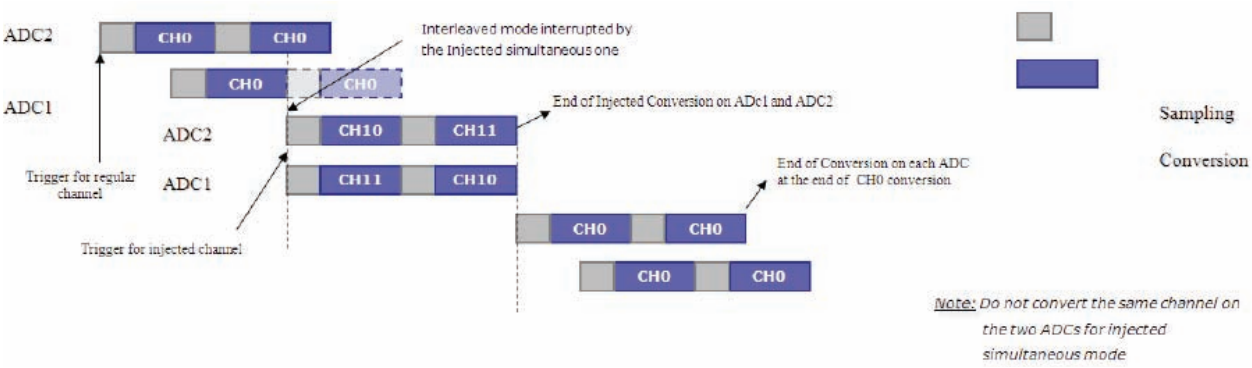



*Перевод к рисунку: ADC1 – АЦП1, ADC2 – АЦП2, Trigger for regular channel – Запуск для регулярного канала, 1st injected Trigger – Запуск для первого инжектированного канала, Regular simultaneous mode interrupted by the Alternate trigger mode – Режим одновременного преобразования прерывается режимом поочередного запуска, End of Injected Conversion on ADC1 – Завершение преобразования инжектированных каналов АЦП1, 2nd injected Trigger – Запуск для второго инжектированного канала, End of Injected Conversion on ADC2 - Завершение преобразования инжектированных каналов АЦП2, End of Conversion on ADC1 and ADC2 – Завершение преобразования АЦП1 и АЦП2, Sampling - Выборка, Conversion - Преобразование, Note: Do not convert the same channel on two ADCs for regular simultaneous mode – Замечание: Нельзя обрабатывать один канал сразу двумя АЦП в режиме одновременного преобразования регулярных каналов.*

Режим поочередного запуска может быть объединен с режимом одновременного преобразования регулярных каналов. При этом

синхронизируется преобразование регулярных каналов обоих АЦП и поочередно запускается преобразование инжектированных каналов.

5.1.3.9 Объединенный Режим Одновременного Преобразования Инжектированных Каналов и Преобразования со Смещением Регулярных Каналов




*Перевод к рисунку: ADC2 – АЦП2, ADC1 – АЦП1, Trigger for regular channel – Запуск для регулярного канала, Trigger for injected channel – Запуск для инжектированного канала, Interleaved mode interrupt by the Injected simultaneous one – Режим преобразования со смещением прерывается режимом одновременного преобразования инжектированных каналов, End of Injected Conversion on ADC1 and ADC2 – Завершение преобразования инжектированных каналов АЦП1 и АЦП2, End of Conversion on each ADC at the end of CH0 conversion – Преобразование каждого АЦП завершается преобразованием канала CH0, Sampling - Выборка, Conversion - Преобразование, Note: Do not convert the same channel on the two ADCs for injected simultaneous mode - Замечание: Нельзя обрабатывать один канал сразу двумя АЦП в режиме одновременного преобразования инжектированных каналов.*

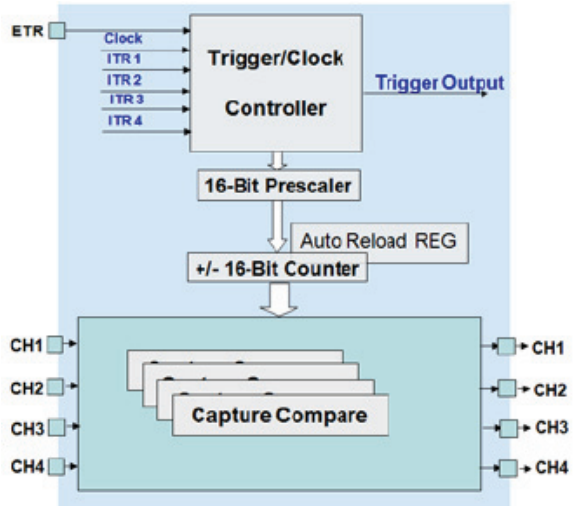
В этом режиме преобразование регулярных каналов осуществляется со смещением во времени и одновременное преобразование двух групп инжектированных каналов.

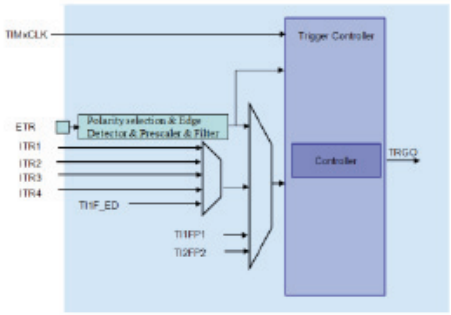
5.1.4 Таймеры Общего Назначения и с Расширенными Функциями

STM32 содержит четыре таймера. Таймер 1 – это таймер с расширенными функциями, предназначенный для управления приводами. Остальные таймеры являются таймерами общего назначения. Все таймеры имеют общую архитектуру; таймер с расширенными функциями содержит только дополнительные аппаратные функциональные блоки. В этой главе мы сначала рассмотрим таймеры общего назначения, а затем перейдем к таймеру с расширенными функциями.

5.1.4.1 Таймеры Общего Назначения

Все таймеры построены на базе 16-битного счетчика с 16-битным делителем частоты и автоматически перезагружаемым регистром. Счетчик таймера может считать в прямом направлении, в обратном направлении или до середины в прямом и затем в обратном направлении. На вход тактового сигнала может подаваться сигнал от одного из восьми источников. В их число входят: специальный сигнал, генерируемый основной системой синхронизации, выходной сигнал других таймеров и внешний сигнал, подаваемы на выводы захвата сравнения. Вход запуска таймера и входы для внешнего тактового сигнала имеют стробируются с помощью вывода для внешнего триггера ETR.



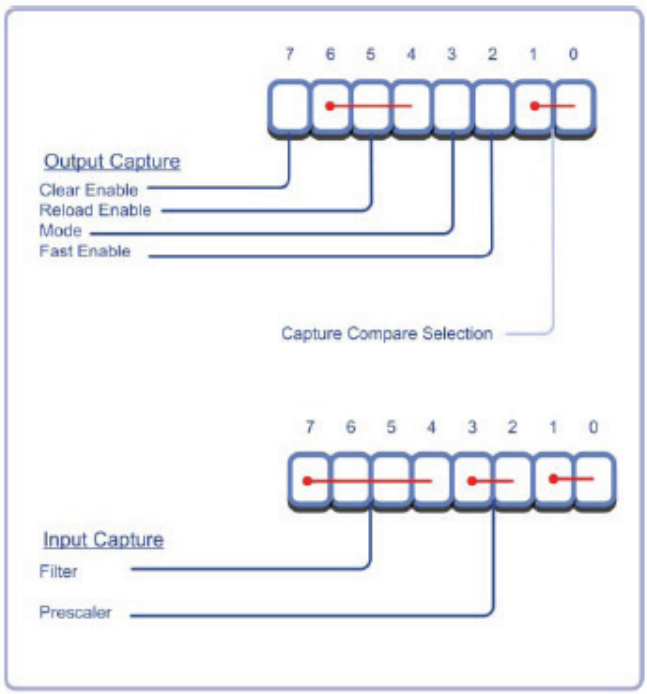

**Каждый из четырех таймеров STM32 содержит 16-битный счетчик с 16-битным делителем частоты и четыре канала захвата сравнения. Таймеры могут тактироваться от системного синхросигнала, внешних импульсов и других таймеров**.

*Перевод к рисунку: Trigger/Clock Controller – Контроллер Запуска/Синхронизации, Trigger Output – Сигнал Запуска Для Внешнего Устройства, 16-Bit Prescaler – 16-Битный Делитель Частоты, Auto Reload REG – Автоперезагружаемый Регистр, +/- 16-Bit Counter - +/- 16-Битный Счетчик, Capture Compare – Захват Сравнение*, *Trigger Controller – Контроллер Запуска, Controller - Контроллер, Polarity selection & Edge Detector & Prescaler & Filter – Выбор полярности & Детектор фронта импульса & Делитель частоты & Фильтр*.

Кроме базового счетчика, каждый таймер содержит четырехканальный модуль захвата сравнения. Этот модуль может осуществлять простые функции захвата и сравнения, а также поддерживает специальные режимы, позволяющие осуществлять стандартные функции аппаратно. Каждый из таймеров работает с прерываниями и поддерживает передачу по DMA.

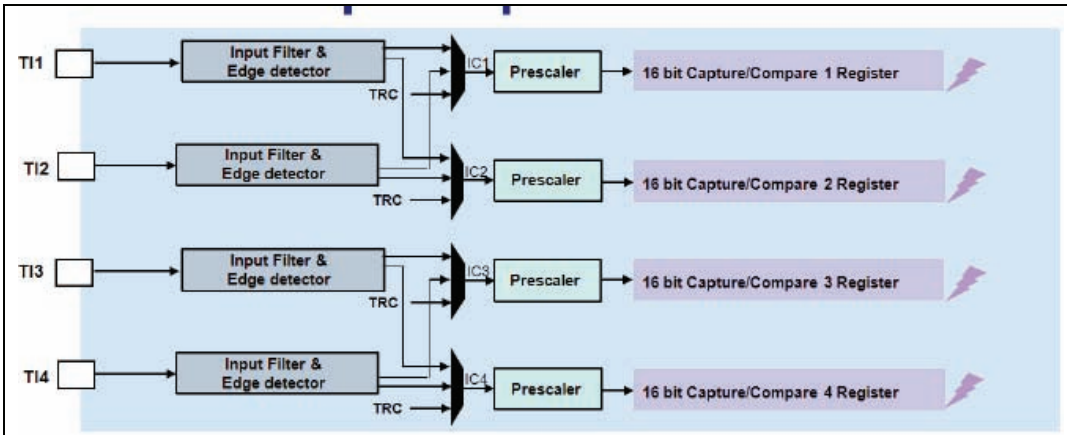
5.1.4.1.1 Модуль Захвата Сравнения

Каждый канал захвата сравнения управляется одним регистром. Этот регистр выполняет различные функции, в зависимости от конфигурации битов. В режиме захвата этот регистр осуществляет фильтрацию входного сигнала, измерение параметров ШИМ, а также прием сигналов кодера. В режиме сравнения регистр выполняет стандартные функции сравнения, задает параметры ШИМ и импульсного режима.

| | |
|---|---|
|  | **Каждый канал захвата/сравнения имеет один регистр. Биты выбора определяют режим канала захвата/сравнения**. |

5.1.4.1.2 Модуль Захвата

Базовый модуль захвата имеет четыре канала, соединенных с конфигурируемыми детекторами фронта импульса. При детектировании переднего или заднего фронта импульса текущее значение счетчика таймера сохраняется в 16-битном регистре захвата/сравнения канала. В момент захвата, счетчик таймера может быть сброшен или остановлен. Кроме того, может быть запущена передача по DMA или сгенерировано прерывание.

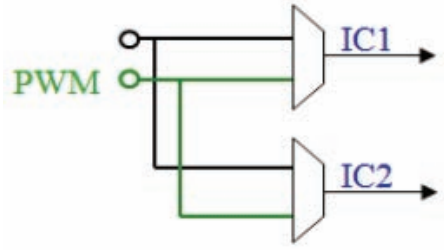



**Четыре канала захвата содержат входной фильтр и детектор фронта импульса. В момент захвата может запускаться передача по DMA или генерироваться прерывание**.

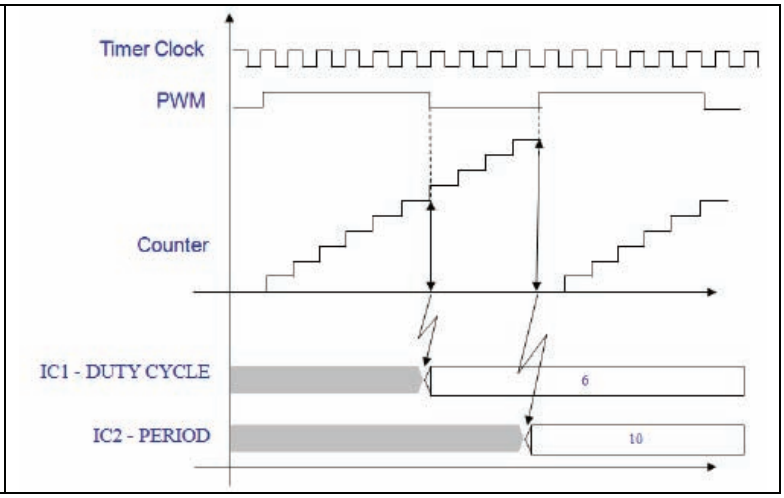
*Перевод к рисунку: Input Filter & Edge detector – Входной Фильтр и Детектор Фронта Импульса, Prescaler – Делитель Частоты, 16 bit Capture/Compare 1 Register – 16-битный Регистр Захвата/Сравнения 1, 16 bit Capture/Compare 2 Register - 16-битный Регистр Захвата/Сравнения 2, 16 bit Capture/Compare 3 Register - 16-битный Регистр Захвата/Сравнения 3, 16 bit Capture/Compare 4 Register - 16-битный Регистр Захвата/Сравнения 4.*

5.1.4.1.3 Режим Измерения Параметров ШИМ

Модуль захвата сравнения может конфигурироваться для работы с входным ШИМ сигналом. При этом два канала захвата автоматически измеряют длительность импульса и период ШИМ сигнала.




**В режиме измерения ШИМ, два канала используются для автоматического захвата периода и длительности импульса ШИМ.**




*Перевод к рисунку: Timer Clock – Синхросигнал Таймера, PWM - ШИМ, Counter - Счетчик, IC1-DUTY CYCLE – IC1-ДЛИТЕЛЬНОСТЬ ИМПУЛЬСА, IC2-PERIOD – IC2-ПЕРИОД.*

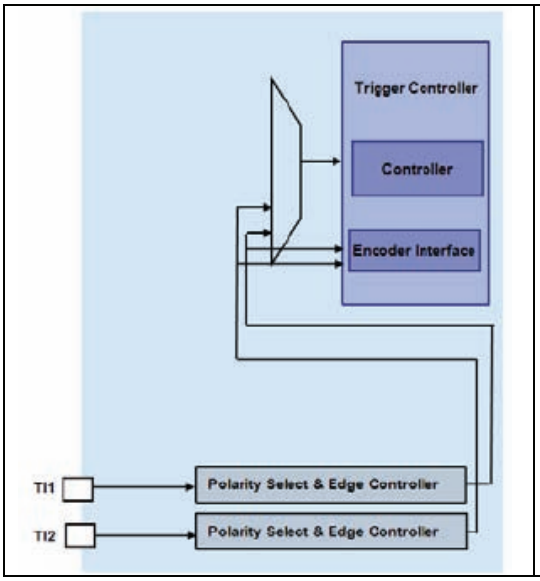
```
M3 -> CR1     =  0x00000000;        // по умолчанию
TIM3 -> PSC   =  0x000000FF;              // установка максимального значения делителя частоты
```

```
TIM3 -> ARR    =  0x00000FFF;                  // установка максимального значения
перезагрузки счетчика
TIM3 -> CCMR1        =  0x00000001;        // Вход IC1 соответствует TI1
TIM3 -> CCER  |=0x00000000;        // IC1 срабатывает по переднему фронту
импульса
TIM3 -> CCMR1  |= 0x00000200;    // Вход IC2 соответствует TI1
TIM3CCER      |= 0x00000020;        // IC1 срабатывает по заднему фронту импульса
TIM3 -> SMCR            = 0x00000054;        // Выбор TI1FP1 как входа, по переднему
фронту
                                                // импульса счетчик сбрасывается
TIM3 -> CCER   |= 0x00000011;      // включение каналов захвата
TIM3 -> CR1       = 0x00000001;       // включение таймера
```

В режиме ШИМ входной сигнал подается на входы двух каналов захвата. В начале периода ШИМ сигнала основной счетчик сбрасывается вторым каналом захвата (используется передний фронт импульса ШИМ) и начинает считать в прямом направлении. По заднему фронту ШИМ сигнала срабатывает первый канал захвата, получая таким образом длительность импульса ШИМ. По следующему переднему фронту импульса вначале следующего периода ШИМ снова срабатывает второй канал захвата, получая значение периода ШИМ и сбрасывая таймер.

5.1.4.1.4 Интерфейс Кодера

С помощью модуля захвата каждого из таймеров может организовывать интерфейс с внешним кодером. Кодер обычно используется при оценке скорости вращения и угла поворота двигателя.



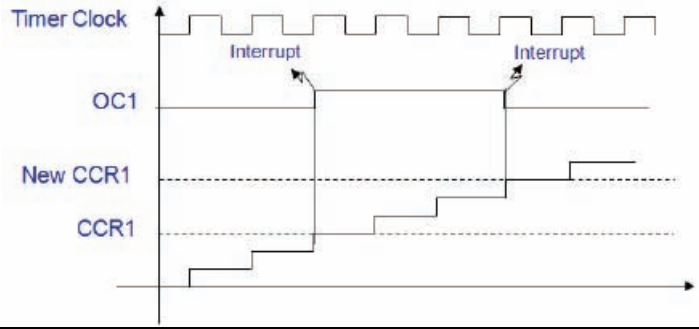

**Любой из таймеров может служить интерфейсом для линейного или углового кодера, захватывая значения положения, скорости и направления**.

*Перевод к рисунку: Trigger Controller – Контроллер Запуска, Controller - Контроллер, Encoder Interface – Интерфейс Кодера, Polarity Select & Edge Controller – Контроллер Выбора Полярности и Фронта Импульса*.

В этой конфигурации через выводы захвата подается тактовый сигнал для счетчика таймера. Значение счетчика затем используется для вычисления положения. Для определения скорости нужно использовать второй таймер.

#### 5.1.4.1.5 Режим Сравнения

Каждый из таймеров STM32 поддерживает четыре канала сравнения. Когда значение счетчика таймера совпадает со значением 16-битного регистра захвата/сравнения, генерируется событие. Это событие может использоваться для изменения состояния вывода захвата/сравнения соответствующего канала, сброса таймера, прерывания или запуска передачи DMA.

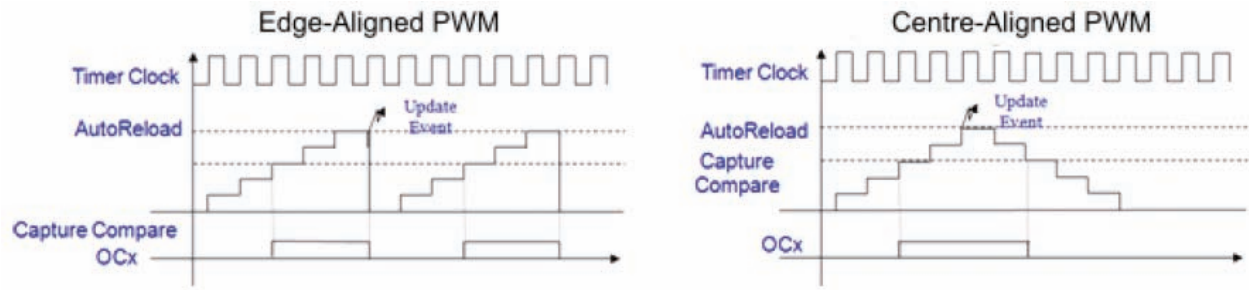



**В режиме сравнения любой из каналов может использоваться для генерации прерывания или изменения состояния вывода захвата/сравнения, при совпадении содержимого регистра сравнения со счетчиком таймера**.

*Перевод к рисунку: Timer Clock – Синхросигнал Таймера, Interrupt - Прерывание, New CCR1 - Новое CCR1.*

#### 5.1.4.1.6 Режим Генерирования ШИМ

Кроме базового режима сравнения, каждый таймер поддерживает специальный режим генерации ШИМ. В этом режиме период ШИМ задается значением, хранящимся в автоперезагружающемся регистре таймера. Длительность импульса определяется значением, хранящимся в регистре захвата/сравнения соответствующего канала. Таким образом, каждый таймер может генерировать до четырех независимых ШИМ сигналов. Как мы увидим позже, таймера могут работать синхронно друг с другом и генерировать до 16 синхронных ШИМ сигналов.




**Каждый таймер имеет специальный режим ШИМ, в котором генерируется выровненный по фронту или по центру ШИМ сигнал**.

*Перевод к рисунку: Edge-Aligned PWM – Выровненный по Фронту ШИМ, Centre-Aligned PWM – Выровненный по Центру ШИМ, Timer Clock – Синхросигнал Таймера, AutoReload - Автоперезагрузка, Capture Compare – Захват Сравнение, Update Event – Событие Обновления*.

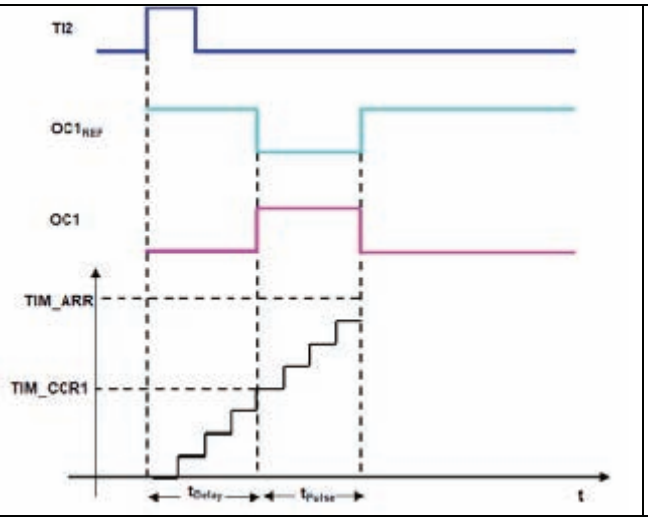
Каждый канал может генерировать выровненный по фронту, либо по центру ШИМ сигнал. В режиме генерирования выровненного по фронту ШИМ задний фронт всегда совпадает с обновлением таймера. Изменение значения регистра захвата/сравнения просто модулирует передний фронт ШИМ сигнала. В режиме с выравниванием по центру счетчик таймера конфигурируется как считающий сначала в прямом, затем в обратном направлении. При совпадении значения регистра захвата/сравнения канала со значением счетчика, изменяется состояние вывода канала.

```
TIM2 -> CR1          = 0x00000000;        // по умолчанию
TIM2 -> PSC          = 0x000000FF;        // установка максимального значения
делителя частоты
TIM2 -> ARR          = 0x00000FFF;// установка максимального значения
перезагрузки счетчика
TIM2 -> CCMR1              = 0x00000068;        // установка режима ШИМ
TIM2 -> CCR1         = 0x000000FF;        // установка начального значения ШИМ
TIM2 -> CCER         = 0x00000101;        // Включение выхода канала CH1
TIM2 -> DIER         = 0x00000000;        // включение прерывания обновления
TIM2 -> EGR          = 0x00000001;        // включение обновления
TIM2 -> CR1          = 0x00000001;        // включение таймера
```

5.1.4.1.7 Режим Генерации Одиночного Импульса

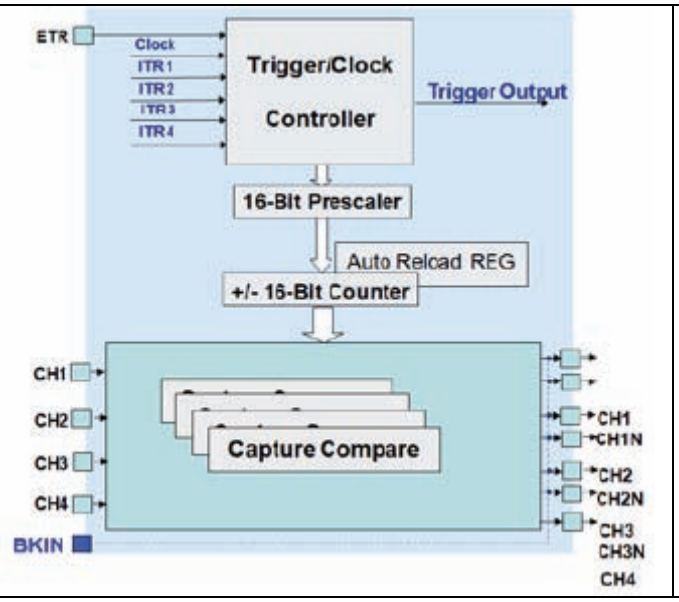
В базовом режиме сравнения и в режиме ШИМ таймер генерирует непрерывную последовательность сигналов. Каждый из таймеров также поддерживает режим одиночного импульса. По сути, это частный случай режима ШИМ, в котором

внешнее событие запускает генерирование одного импульса ШИМ. При этом можно задавать начальную задержку и длительность импульса.




**Режим одиночного импульса позволяет генерировать одиночный короткий импульс с задаваемой начальной задержкой и длительностью**.

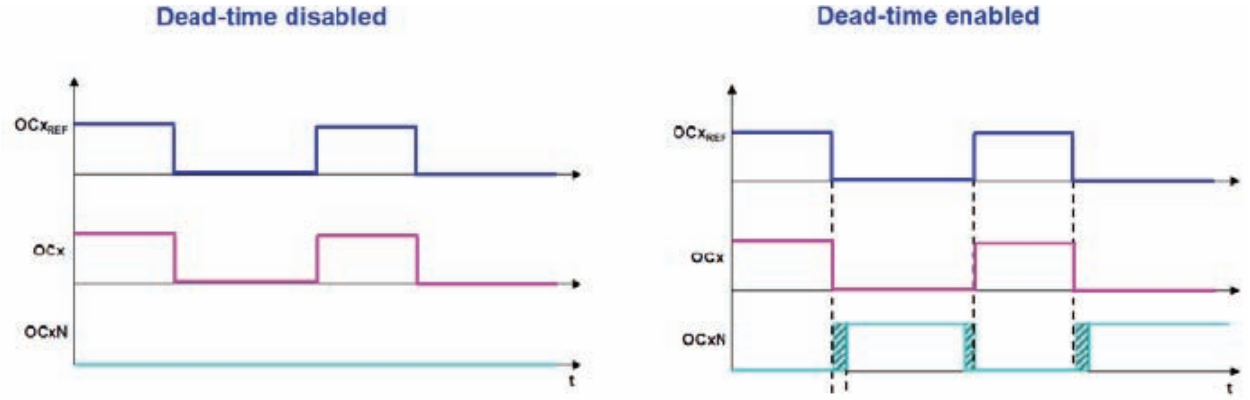
Таймер 1 является таймером с расширенными функциями. Он содержит дополнительные аппаратные функции, предназначенные для управления приводом. Три выходных канала таймера с расширенными функциями имеют комплементарные выводы. То есть, получается шесть каналов ШИМ. Так как этот таймер предназначен для управления трехфазными приводами, для каждого из каналов задается «мертвое время», а также имеется входная линия экстренного отключения. В дополнение к интерфейсу кодера имеется также интерфейс датчика Холла.




**Таймер с расширенными функциями имеет такую же базовую структуру, как и таймеры общего назначения. Три выходных канала содержат комплементарные выводы. Также имеется дополнительный вход экстренного отключения и интерфейс датчика Холла**.

Перевод к рисунку: *Trigger/Clock Controller – Контроллер Запуска/Синхронизации, Trigger Output – Сигнал Запуска Внешнего Устройства, 16-Bit Prescaler – 16-Битный Делитель Частоты, Auto Reload REG – Автоперезагружаемый Регистр, +/- 16-Bit Counter - +/- 16-Битный Счетчик, Capture Compare – Захват Сравнение*

Для каждого из трех комплементарных каналов ШИМ задается «мертвое время», которое вводит задержку между моментом сброса в ноль вывода ШИМ и моментом установки в единицу комплементарного вывода.




**Комплементарные выводы ШИМ таймера с расширенными функциями имеют программируемое мертвое время для управления двигателями**.

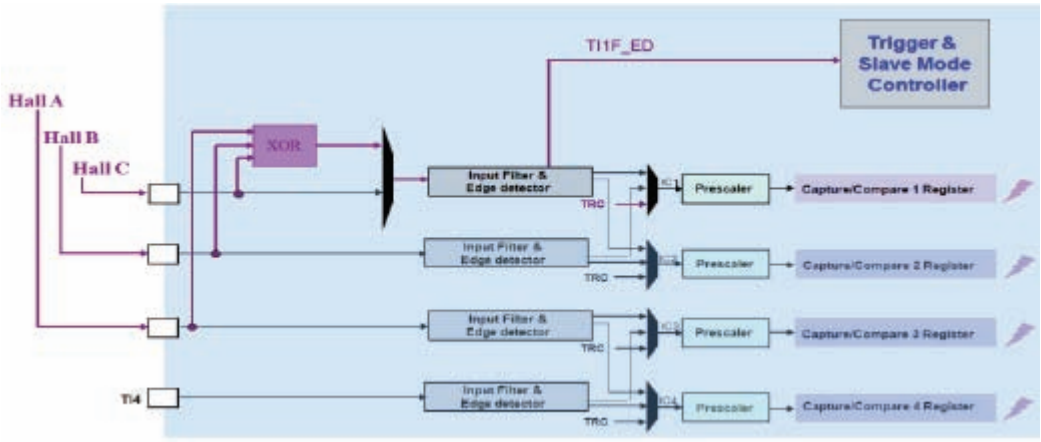
*Перевод к рисунку: Dead-time disabled – Мертвое время выключено, Dead-time enabled – Мертвое время включено*.

5.1.4.2.1 Функция Экстренного Отключения

Таймер с расширенными функциями может переводить выводы ШИМ (вместе с комплементарными) в предопределенное пользователем состояние по сигналу линии экстренного отключения. Сигнал на эту линию может подаваться с внешнего вывода экстренного отключения или из системы безопасности системы синхронизации, которая отслеживает состояние внешнего высокочастотного осциллятора. Функция экстренного отключения работает полностью на аппаратном уровне и гарантирует перевод выводов ШИМ в безопасное состояние при нарушении в системе синхронизации STM32 или внешних аппаратных устройств.

5.1.4.2.2 Интерфейс Датчика Холла

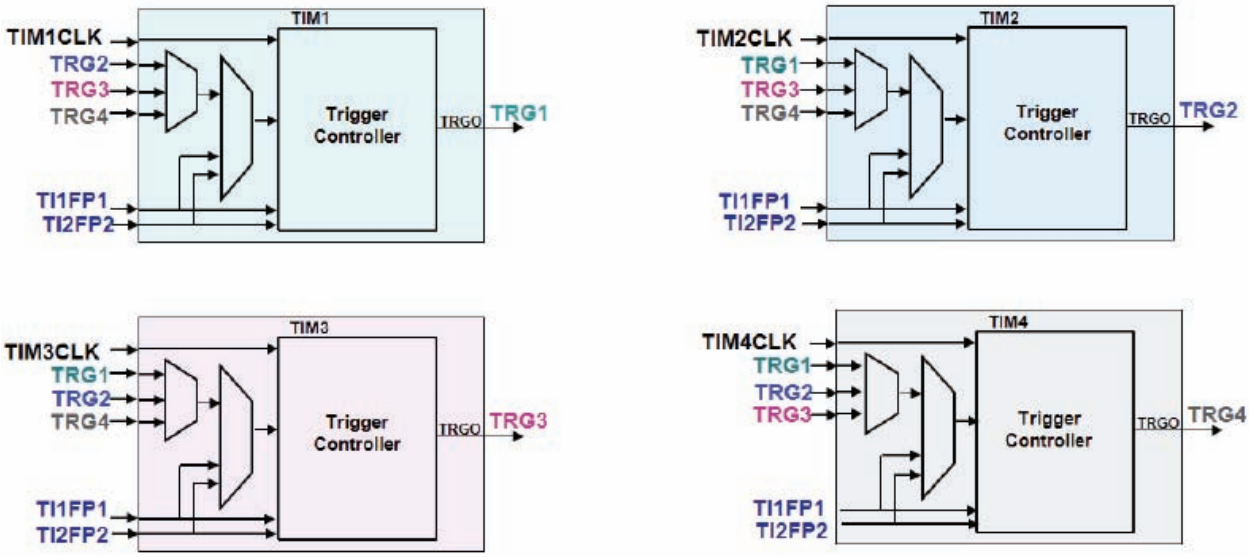
Каждый из таймеров, включая таймер с расширенными функциями обеспечивает простой интерфейс с датчиками Холла для измерения угловой скорости привода. Первые три вывода каждого таймера могут подключаться к первому каналу через исключающее ИЛИ. При вращении привода и прохождении мимо каждого датчика в первом канале будет генерироваться событие захвата. В такие моменты содержимое счетчика таймера передается в регистр захвата первого канала и сбрасывается таймер. По значению в регистре захвата оценивается угловая скорость привода.




Перевод к рисунку: Trigger & Slave Mode Controller – Контроллер Запуска и Режима Ведомого, *Input Filter & Edge detector – Входной Фильтр и Детектор Фронта Импульса, Prescaler – Делитель Частоты, Capture/Compare 1 Register – Регистр Захвата/Сравнения 1, Capture/Compare 2 Register - Регистр Захвата/Сравнения 2, Capture/Compare 3 Register - Регистр Захвата/Сравнения 3, Capture/Compare 4 Register - Регистр Захвата/Сравнения 4*.

### 5.1.4.3 Синхронизация Таймеров

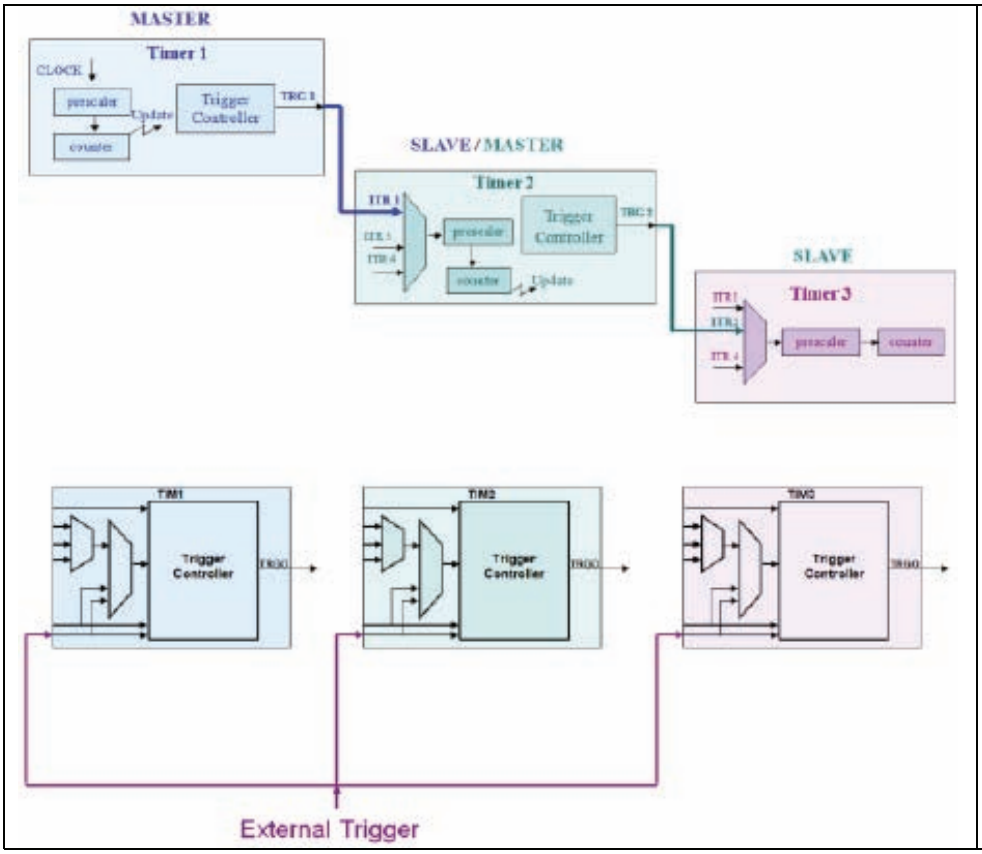
Несмотря на то, что каждый таймер является независимым, они могут синхронизироваться между собой. Эта возможность позволяет аппаратно организовывать массивы таймеров, сокращая тем самым объем программного кода, требующегося для реализации сложных временных функций.




**Каждый из таймеров содержит вход запуска от других трех таймеров, а также внешние входы, связанные с выводами захвата/сравнения**.

Перевод к рисунку: Trigger Controller – Контроллер Запуска.

Каждый таймер содержит выход для запуска внешнего устройства, сигнал с которого подается на входы остальных трех таймеров. Кроме того, выводы входа захвата таймера 1 и таймера 2 (TI1FP1 и TIFP2) подключаются к контроллеру запуска каждого из таймеров. Таймера могут синхронизироваться в нескольких различных режимах. Ниже показано несколько возможных конфигураций.




**Все таймера могут каскадироваться в различных конфигурациях для образования массивов таймеров**.

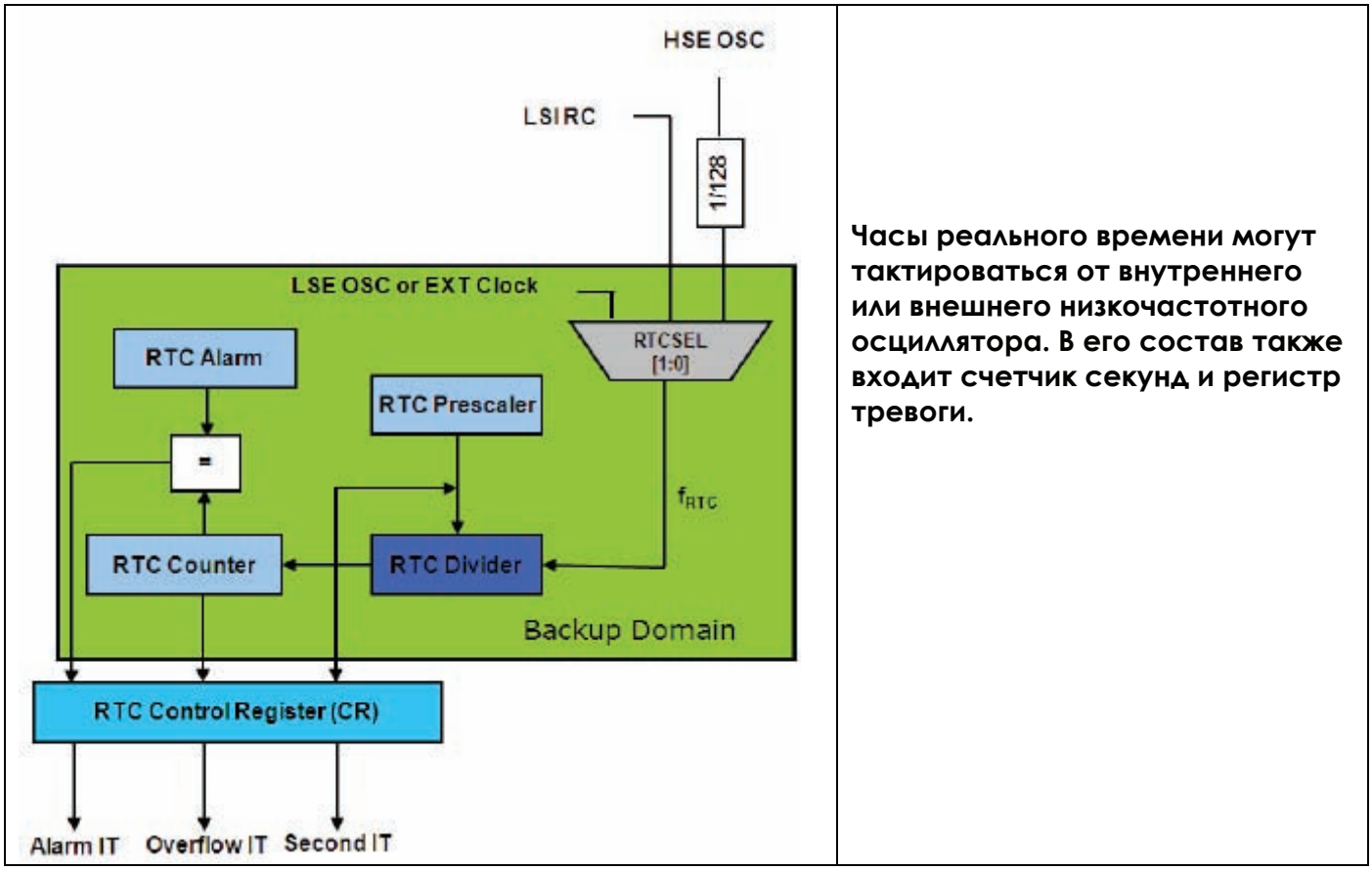
*Перевод к рисунку: Master - Ведущий, Slave/Master – Ведомый/Ведущий, Slave - Ведомый, prescaler - делитель, counter - счетчик, Trigger Controller – Контроллер Запуска, External Trigger – Внешний Сигнал Запуска*.

Один таймер функционирует как ведущий с двумя ведомыми таймерами. Ведущий может выдавать тактовый сигнал для двух ведомых, в результате чего получается один большой таймер. Точно также, ведущий может вводить временную задержку запуска или отключения двух ведомых. Аналогично может быть использоваться внешний сигнал для запуска каждого из таймеров.

### 5.1.5 Часы реального времени RTC и Регистры Резервирования

STM32 содержит две системы питания: для основной части системных и периферийных устройств STM32, и для области резервирования. В области резервирования расположены десять 16-битных регистров, часы реального времени RTC и независимый сторожевой таймер. Регистры резервирования представляют собой просто десять ячеек памяти, использующиеся для сохранения важных данных в те моменты времени, когда STM32 находится в режиме ожидания с выключенным основным питанием. В режиме сниженного энергопотребления RTC и независимый сторожевой таймер могут оставаться включенными и использоваться для вывода STM32 из спящего режима или для сброса микроконтроллера.

STM32 содержит базовые часы реального времени. Это 32-битный счетчик, который инкрементируется каждую секунду, если тактируется от осциллятора 32.768 кГц. При конфигурации системы синхронизации для RTC может быть выбран один из следующих источников тактового сигнала: низкочастотный внутренний осциллятор, низкочастотный внешний осциллятор или высокочастотный внешний осциллятор с фиксированным делителем частоты на 128. Делитель частоты самого модуля RTC позволяет точнее настроить интервал счета. Счетчик часов реального времени может генерировать три вида прерываний: секундное инкрементирование, переполнение счетчика и тревога. Прерывание по тревоге генерируется когда счечик RTC досчитывает до значения, хранящегося в регистре тревоги.




**Часы реального времени могут тактироваться от внутреннего или внешнего низкочастотного осциллятора. В его состав также входит счетчик секунд и регистр тревоги.**

*Перевод к рисунку: RTC Alarm – RTC Тревога, RTC Prescaler – RTC Делитель Частоты, RTC Counter – RTC Счетчик, RTC Divider – RTC Делитель, Backup Domain – Облась с*

*Резервным Питанием, RTC Control Register (CR) – Регистр Управления RTC (CR), Alarm IT – Прерывание по тревоге, Overflow IT – Прерывание по переполнению , Second IT – Ежесекундное прерывание*.

RTC расположен в области с резервным питанием от источника напряжения $V_{BAT}$. Прерывание по тревоге выводится на линию внешних прерываний EXTI 17. Это значит, что когда основное питание STM32 выключено и микроконтроллер находится в режиме сниженного энергопотребления, RTC остается включенным. Через внешнее прерывание, часы реального времени могут выводить STM32 из спящего режима. Такая конфигурация RTC осень важна для обеспечения низкого энергопотребления приложения, когда устройство находится большую часть времени в глубоком «спящем» режиме и для выхода из него нужно какое-либо прерывание.

### 5.1.6 Регистры Резервирования и Вывод Вскрытия Устройства

Область с резервным источником питания также содержит десять 16-битных регистров, которые работают как SRAM с питанием от батареи. Данные, содержащиеся в этих регистрах, могут стираться через регистр управления RCC. Внешний вывод вскрытия устройства также активируется в этом регистре. При включении микроконтроллера этот вывод может быть сконфигурирован для перехода в состояние логической единицы или нуля. Изменение логического уровня этого вывода приведет к стиранию информации регистров резервирования. Также можно активировать генерирование прерывания при изменении логического уровня вывода, что позволяет в обработчике прерывания предпринимать нужные защитные действия при вмешательстве в устройство.

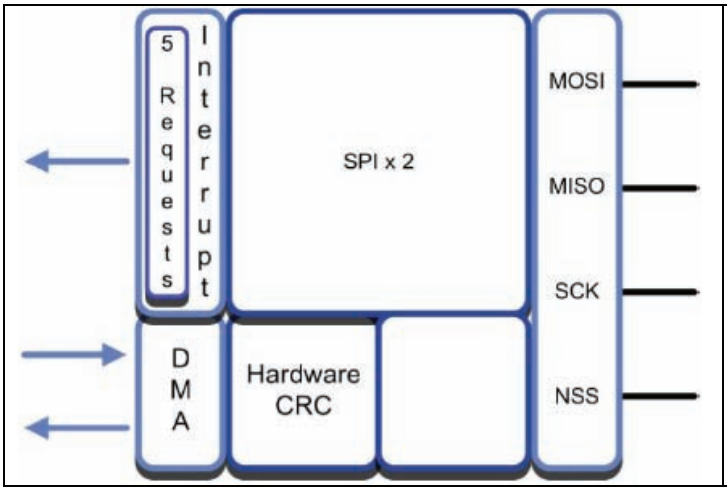
## 5.2 Коммуникационные Интерфейсы

Кроме превосходного набора периферийных устройств общего назначения, STM32 содержит пять различных типов коммуникационных периферийных устройств. Для обмена информацией между компонентами на печатной плате STM32 содержит SPI и I2C интерфейсы. Для связи между различными модулями устройства имеется шина CAN, а для коммуникаций с ПК – USB device интерфейс. Также в STM32 используется популярный USART интерфейс.

### 5.2.1 SPI

Для быстрого обмена данными между компонентами печатной платы, STM32 содержит два SPI модуля, которые обеспечивают полнодуплексную передачу данных на частотах до 18 МГц. Очень важно отметить, что один из SPI модулей

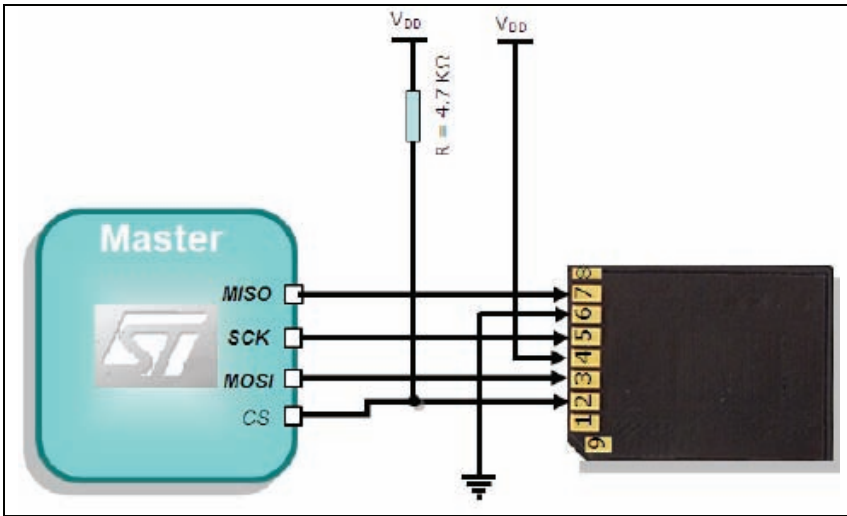
расположен на высокоскоростной шине периферийных устройств APB2, которая работает на тактовой частоте до 72 МГц. Второй SPI расположен на низкоскоростной шине APB1, работающей на тактовой частоте до 37 МГц. Для каждого SPI можно задать полярность и фазу тактового сигнала. Данные могут передаваться как 8 или 16-битные слова, старшим или младшим битом вперед. Это позволяет обоим SPI модулям работать как ведущий или ведомый и обмениваться информацией с любым другим SPI устройством.




**Каждый SPI интерфейс может работать как ведущий или ведомый с частотой до 18 МГц. Для увеличения эффективности передачи данных выделены два канала DMA**.

*Перевод к рисунку: 5 Requests – 5 Запросов, Interrupt - Прерывание, Hardware CRC – Аппаратный CRC*.

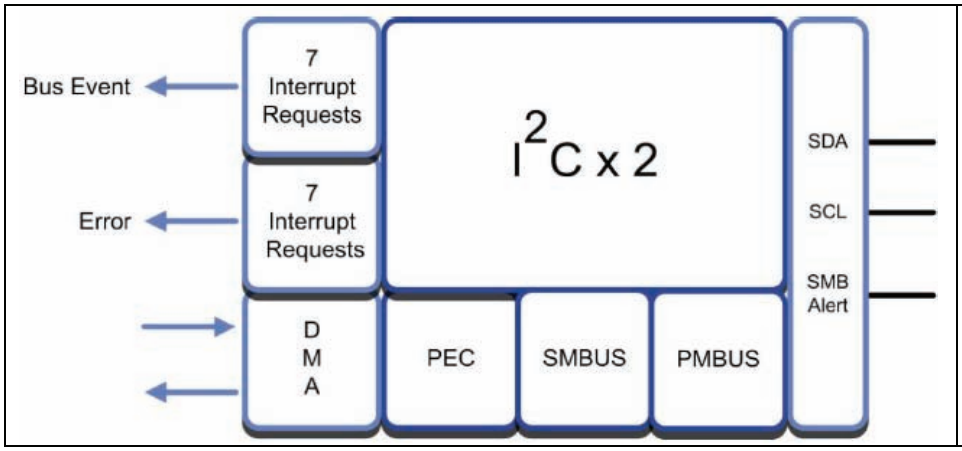
Для организации высокоскоростного обмена данными, каждый SPI модуль содержит два канала DMA: один для передачи данных и второй для сохранения принимаемых данных в память. Использование DMA позволяет обмениваться данными на высокой скорости в двух направлениях под аппаратным управлением. Кроме стандартных функций, SPI STM32 содержит два аппаратных блока вычисления циклического избыточного кода CRC. Один CRC блок используется для передаваемых данных, а второй – для принимаемых. Оба блока могут вычислять циклический избыточный код для 8 и 16-битных данных. Эта функция особенно необходима при подключении к SPI карт памяти MMC/SD.




**Модуль SPI содержит аппаратный блок вычисления циклического избыточного кода CRC, который необходим для обмена информацией с мультимедийными и SD картами памяти**.

5.2.2 I2C

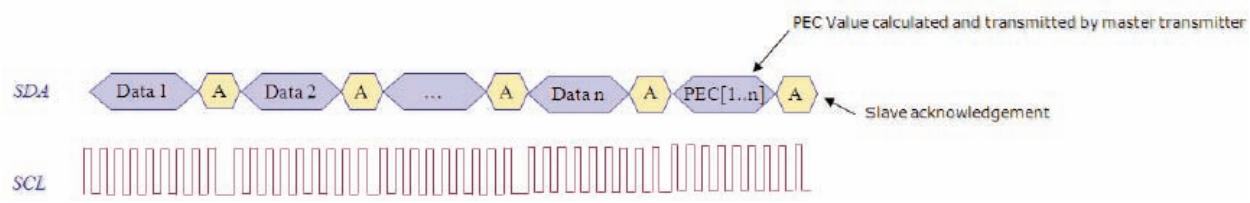
STM32 может обмениваться данными с другими компонентами на печатной плате через интерфейс I2C. I2C интерфейс может работать как ведомый или ведущий, а также осуществлять арбитраж в системе с несколькими ведущими. SPI интерфейс поддерживает стандартные частоты передачи данных до 100 кГц и быструю передачу данных на частотах до 400 кГц. Модуль I2C поддерживает 7-битный и 10-битный режимы адресации. По сути, модуль I2C просто передает и принимает данные по шине. Программное обеспечение пользователя должно управлять модулем I2C для организации нужного протокола обмена информацией с различными устройствами на шине. Модуль I2C генерирует два прерывания для процессора Cortex: одно для ограничения распространения ошибки и второе для управления адресами и передачей данных. Кроме того, выделяются два канала DMA по которым можно считывать из и записывать данные в передающий буфер I2C. Таким образом, после согласования адресов устройств в сети и данных для передачи, обмен информацией может производиться под аппаратным управлением STM32.




**Два модуля I2C имеют дополнительные функции для управления шиной и управления энергопотреблением шины. Они также содержат аппаратный блок коррекции ошибок**.

*Перевод к рисунку: Bus Event – Событие Шины, Error - Ошибка, 7 Interrupt Requests – 7 Запросов Прерываний*.

Все перечисленные отличительные особенности позволяют назвать модуль I2C STM32 быстрым и эффективным шинным интерфейсом. Однако есть еще несколько дополнительных особенностей, выходящих за рамки базовых функций I2C. Модуль I2C STM32 содержит аппаратный блок проверки пакетных ошибок (PEC). Во включенном состоянии, PEC генерирует 8-битный циклический избыточный код CRC. Этот CRC-код автоматически размещается в конце передающейся посылки. PEC также проверяет принимаемые данные на соответствие CRC-кодов.

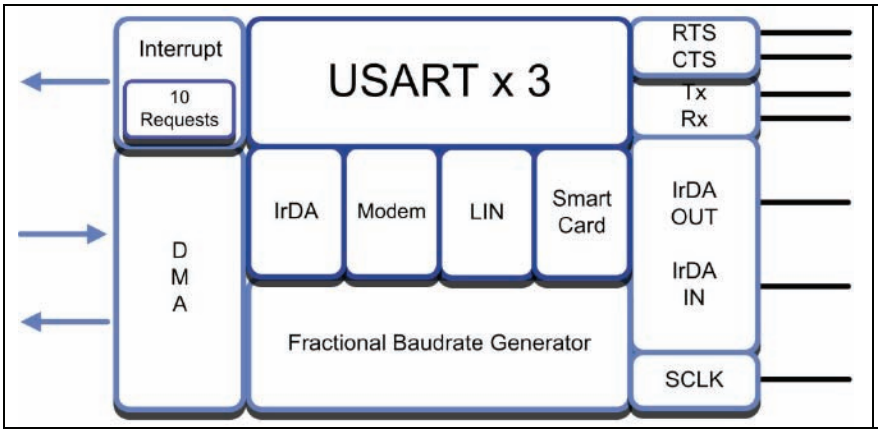



*Перевод к рисунку: Data - Данные, PEC Value calculated and transmitted by master transmitter – Значение PEC, вычисленное и переданное ведущим устройством, Slave acknowledgement – Подтверждение от ведомого устройства.*

Модуль I2C STM32 поддерживает два расширенных протокола, SMBus и PMB. Системная Управляющая Шина SMBus – это протокол, разработанный компанией Intel в 1995 году для использования в ПК и серверах. SMBus определяет канальный уровень в который входит использование проверки пакетных ошибок и дополнительный сетевой уровень, стандартизующий конфигурацию взаимодействия БИОСа ПК и устройств различных производителей. Во время работы в режиме SMBus, модуль I2C, кроме проверки пакетных ошибок, поддерживает несколько других функций SMBus. В их число входит SMN address resolution protocol, протокол уведомления хоста и сигнал SMBALERT. PMBus протокол представляет собой версию SMBus, разработанную для использования в системах преобразования мощности. PMBus позволяет конфигурировать, программировать и контролировать в реальном времени такие системы.

### 5.2.3 USART

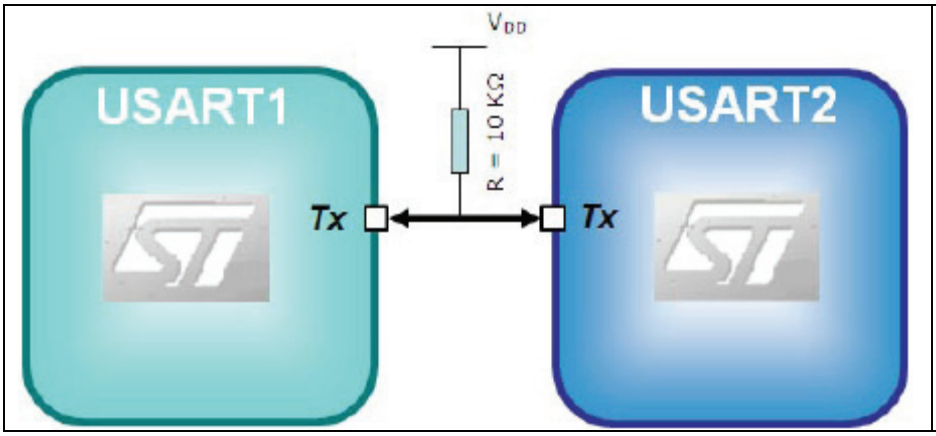
Несмотря на то, что последовательные порты уходят из ПК, они до сих пор широко применяются во встраиваемых приложениях как простые последовательные коммуникационные интерфейсы. Благодаря их практичности и простоте использования, мы будем с ними работать еще много лет. STM32 содержит до трех USART, каждый из которых поддерживает несколько расширенных режимов для работы в составе современных устройств с последовательной передачей данных. Каждый из трех USART может обмениваться данными со скоростью до 4 Мбит/с. Для каждого USART можно задавать длину посылки данных (8 или 9 бит), стоповый бит четности и скорость передачи. Один USART расположен на шине APB2, которая работает на тактовой частоте до 72 МГц, в то время как остальные на шине APB1, с тактовой частотой до 36 МГц.




**USART может обмениваться данными в асинхронном режиме с UART-, LIN-, IrDA-устройствами, а также с модемами и смарт-картами**.

*Перевод к рисунку: Interrupt - Прерывание, 10 Request – 10 Запросов, Modem - Модем, Smart Card – Смарт Карта, Fractional Baudrate Generator – Генератор Дробных Скоростей Передачи*.

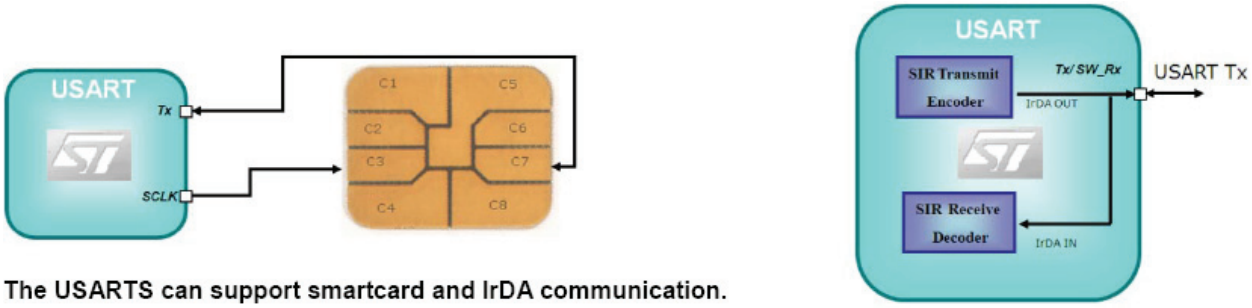
Задающий генератор каждого USART представляет собой генератор дробных скоростей передачи, что намного сложнее, чем простой делитель частоты. Он позволяет получать стандартные скорости передачи из любой частоты шины. Как все остальные последовательные коммуникационные периферийные устройства, каждый USART поддерживает два канала DMA для передачи данных в память и из памяти. Когда USART используется как UART, он поддерживает несколько специальных режимов передачи данных. USART может передавать данные в полудуплексном режиме по одному проводу, используя только вывод Tx. Для подключения модема, а также для аппаратного управления передачей данных, каждый USART содержит дополнительные линии управления CTS и RTS.




**USART модули поддерживают обмен данными в полудуплексном режиме по одному проводу**.

Каждый USART может использоваться для подключения к шине LIN. Эта шина используется в автомобильных приложениях для построения не дорогих микроконтроллерных сетей. Любой из модулей USART может работать как кодер/декодер последовательного инфракрасного сигнала в соответствии со стандартом IrDA. При этом обеспечивается скорость передачи до 115200 бит/с, используется полудуплексная NRZ модуляция, поддерживается режим низкого энергопотребления при работе модуля USART на частотах от 1.4 МГц до 2.12 МГц.

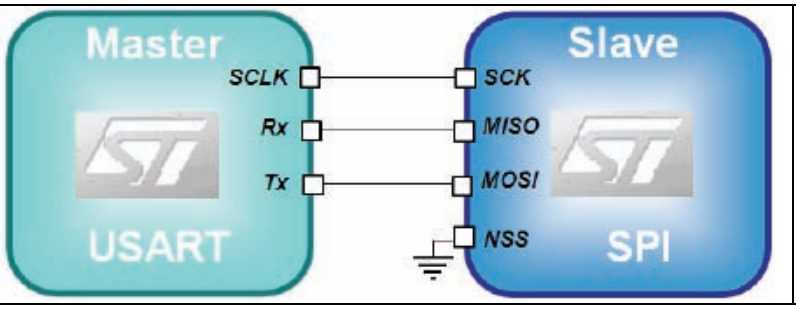
Все модули USART имеют дополнительный режим для работы со смарт-картами в соответствии со стандартом ISO 7618-3.




**The USARTS can support smartcard and IrDA communication.**

*Перевод к рисунку: The USARTS can … communication – Модули USART могут обмениваться данными со смарт-картами и через IrDA.*

Кроме работы в режиме UART, USART может использоваться для синхронной передачи данных, что позволяет подключаться к другим устройствам через трехпроводной SPI. В этом режиме модуль USART работает как ведущее SPI-устройство, для которого можно задавать полярность и фазу тактового сигнала, так, чтобы оно могло обмениваться данными с любым ведомым SPI-устройством.



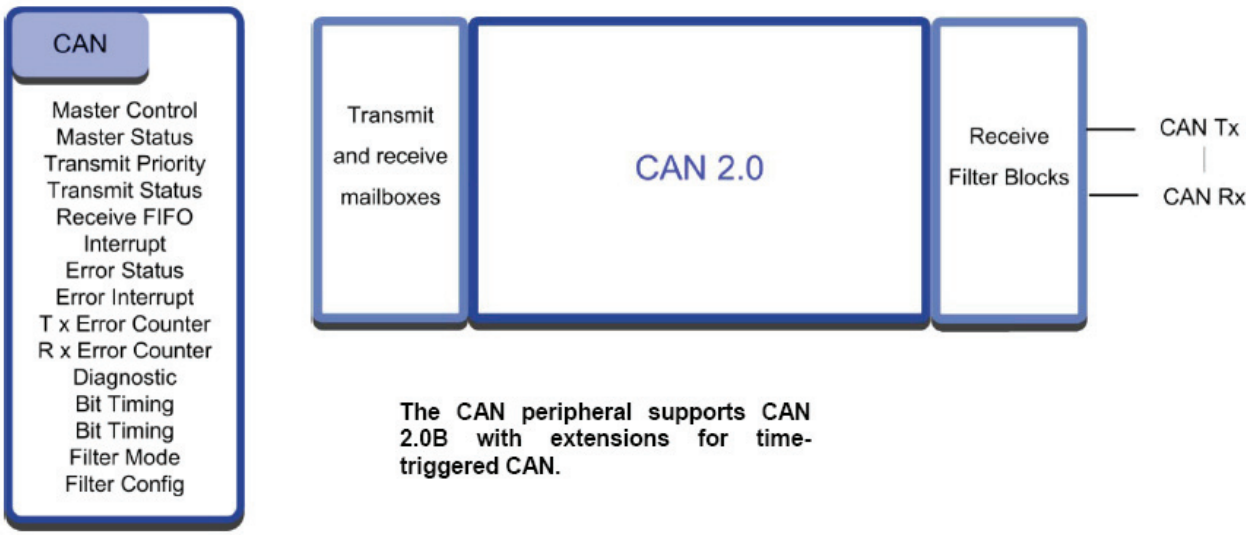

**При работе в синхронном режиме, USART может использоваться как ведущий SPI.**

*Перевод к рисунку: Master - Ведущий, Slave - Ведомый.*

Оставшиеся два коммуникационных периферийных устройства в составе STM32: CAN контроллер и USB full speed device. Оба протокола являются достаточно сложными. Если вы не имеет их опыта использования, вам лучше начать с прочтения специальной литературы по CAN и USB. Для работы обоих устройств требуется относительно большой объем SRAM, который используется для фильтрации сообщений и буферов сообщений. STM32 содержит специальную область из 512 байт SRAM, которая совместно используется модулями CAN и USB. Эта область памяти доступна только для этих двух модулей. Кроме того, CAN и USB могут обращаться к этой области только поочередно.

### 5.3.1 CAN Контроллер

CAN контроллер STM32 представляет собой полноценный модуль CAN, поддерживающий стандарты CAB 2.0A и 2.0B, активную и пассивную передачу данных со скоростью до 1 Мбит/с. CAN контроллер также имеет расширение для поддержки полностью детерминированного обмена данными, согласно протоколу TTCAN. Когда расширение TTCAN включено, поддерживается автоматическая повторная передача сообщения и размещение отметки времени сообщения в двух последних байтах пакета сообщения CAN. Расширение TTCAN делает возможным использование модуля CAN программным обеспечением приложения для управления в жестком реальном времени.

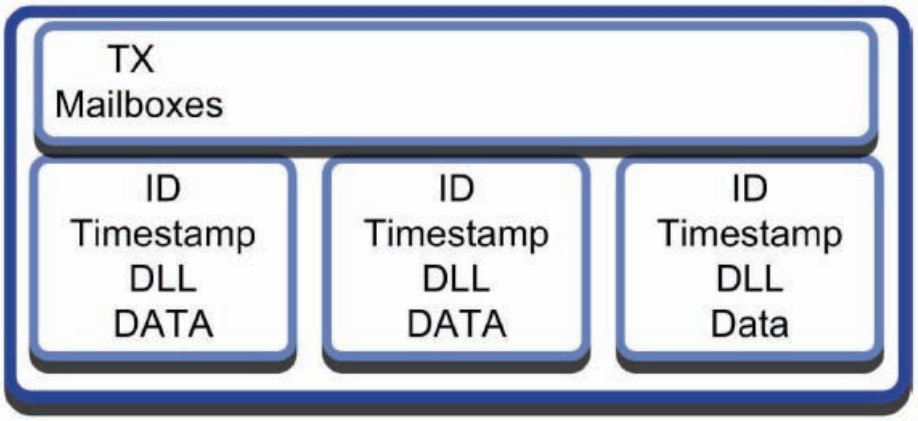



**The CAN peripheral supports CAN 2.0B with extensions for time-triggered CAN.**

*Перевод к рисунку: Transmit and receive mailboxes – Почтовые ящики для приема и передачи, Receive Filter Blocks – Блоки Приемных Фильтров, Master Control – Управление Ведущим, Master Status – Статус Ведущего, Transmit Priority – Приоритет Передачи, Transmit Status – Статус Передачи, Receive FIFO – Приемный FIFO, Interrupt - Прерывание, Error Status – Статус Ошибки, Error Interrupt – Прерывание Ошибки, Tx Error Counter – Счетчик Ошибок Tx, Rx Error Counter – Счетчик Ошибок Rx, Diagnostic - Диагностика, Bit Timing – Тактовая Синхронизация, Filter Mode – Режим Фильтрации, Filter Config – Конфигурация Фильтрации.*

**The CAN peri …. Trigger CAN – Модуль CAN поддерживает стандарт CAN 2.0B с расширением для TTCAN.**

Полное название CAN контроллера – периферийное устройство bxCAN, где bx значит базисное продолжение (basic extended). Базовый CAN содержит один буфер приема и передачи, в то время как расширенный CAN содержит множество буферов приема и передачи. bxCAN – это гибрид двух архитектур CAN. bxCAN имеет три почтовых ящика для передачи и два для приема. Каждый из приемных почтовых ящиков содержит FIFO на три сообщения. Это компромиссный дизайн между CAN с низкой производительностью, но малой занимаемой площадью на кристалле и высокопроизводительным CAN с большой занимаемой площадью.




**Модуль CAN содержит три передающих почтовых ящика с автоматической расстановкой отметок времени для TTCAN.**

*Перевод к рисунку: Mailboxes – Почтовые ящики, Timestamp – Отметка времени.*

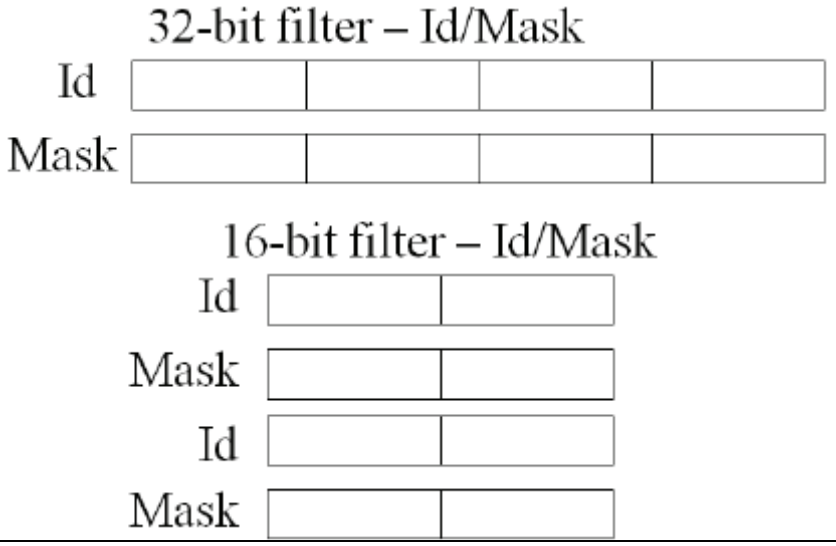
Следующая очень важная особенность CAN контроллера – фильтрация принимаемых сообщений. Так как CAN представляет собой широковещательную сеть, каждое переданное сообщение принимается всеми устройствами в сети. В сети CAN достаточной сложности по шине CAN будет передаваться большое количество сообщений. ЦПУ пришлось бы тратить все время на обработку этих сообщений. Чтобы этого избежать, CAN контроллер содержит фильтр сообщений, который блокирует копирование в приемный буфер ненужных сообщений. CAN контроллер STM32 содержит 14 блоков фильтров, которые пропускают только сообщения с определенным идентификатором.


**14 фильтров сообщений имеют две конфигурации, которые могут использоваться для фильтрации одиночных сообщений**.

Перевод к рисунку: 32-bit filter – 32-битный фильтр, 16-bit filter – 16-битный фильтр.

Каждый блок фильтров состоит из двух 32-битных регистров. Каждый блок может работать в одном из четырех режимов. В обычном режиме, в каждый регистр блока фильтров записывается идентификатор сообщения (ID). Когда приходит

новое сообщение, его ID должен совпасть с записанным в регистре. Если не совпадает – сообщение не принимается. У этого режима есть две конфигурации. В первой конфигурации используются 32 разряда регистров блоков фильтров. При этом могут фильтроваться 11-битные и 29-битные ID-поля сообщений, а также биты RTR и IDE в 16-битном режиме.




**Те же самые блоки фильтров могут использоваться для фильтрации групп сообщений**.

*Перевод к рисунку: 32-bit filter – 32-битный фильтр, 16-bit filter – 16-битный фильтр, Mask - Маска*

Во второй конфигурации в первый 32-битный регистр записывается ID сообщения, а второй регистр используется как маска для сообщений. Этот масочный регистр маркирует биты ID-регистра как «важный» или «неважный». Это позволяет фильтровать группу сообщений через один блок фильтров. При прохождении сообщения через приемный фильтр, указатель фильтра сохраняется в приемном FIFO вместе с сообщением. Это позволяет программе приложения ускорить получение данных сообщения без необходимости считывать и декодировать идентификатора пакета сообщения.

Все CAN контроллеры поддерживают два режима работы: нормальный режим, для приема и передачи пакетов сообщений, и инициализационный режим, для установки коммуникационных параметров. STM32 может переходить в «спящий» режим. В такие моменты тактовый сигнал bxCAN отключается, но регистры почтовых ящиков могут оставаться доступными. Модуль bxCAN выйдет из «спящего» режима при обнаружении активности на шине CAN; его работу можно возобновить также из программного кода приложения. В нормальном режиме имеются два подрежима работы. Первый подрежим – режим молчания, в котором CAN контроллер принимает, но не передает сообщения, а также не генерирует ошибку или подтверждение приема сообщения. Этот режим предназначен для пассивного мониторинга сети CAN. Второй подрежим – режим обратной связи, в котором все переданные сообщения возвращаются в приемный буфер. Этот режим предназначен для самотестирования и также используется во время отладки программного кода приложения. Оба подрежима могут объединяться, что идеально для самотестирования устройства в составе работающей сети.

5.3.2 USB

STM32 содержит full speed (12 Мбит/с) device USB интерфейс, который может использоваться для подключения устройства к ПК. Модуль USB реализует в себе физический уровень USB и уровень передачи данных с проверкой пакетных ошибок и повторной передачей. При разработке USB-приложения требуется знание спецификации USB и его классов приложений. На сайте компании ST доступен полный отладочный набор для разработки USB-приложений. В него входит программный стек для инициализации USB интерфейса и поддержка широко применяемых USB классов, таких как Human Interface Device (HID), массовая память, аудио и традиционный коммуникационный порт. Использование этого стека, или аналогичного стека других компаний, значительно сокращает время разработки и позволяет обойтись без «изобретения колеса».


**Модуль USB работает как USB 2.0 device.**

*Перевод к рисунку: Endpoint Buffers – Буфера конечных точек, Analogue Transceiver – Аналоговый Трансивер.*

USB интерфейс поддерживает до восьми конечных точек, которые пользователь настраивает как конечные точки, работающие в режиме управления, прерываний, передачи больших массивов данных и изохронной передачи. Буфера конечных точек расположены в 512 байтной области SRAM памяти, совместно используемой с CAN контроллером. При инициализации устройства, программный код приложения разделяет SRAM на группы буферов.

Область SRAM памяти разделяется на буфера конечных точек с помощью специальной таблицы, находящейся в начале SRAM. Для каждой конечной точки в таблице хранится начальный адрес и размер буфера. Для конечных точек, работающих в режиме управления, прерываний и передачи больших массивов данных выделяется по одному буферу, в то время как для изохронных конечных точек выделяется двойной буфер. Это позволяет принимать данные в один буфер и одновременно обрабатывать данные из другого. При приеме нового пакета, данные сохраняются во втором буфере, в то время как обрабатывается первый буфер. Такой подход с двойным буфером необходим при потоковой передачи данных в реальном времени, например, при передаче аудио.

# 6. Режимы Сниженного Энергопотребления

Оставаясь микроконтроллером с высокой производительностью, STM32 кроме обычного активного режима работы, поддерживает несколько режимов сниженного энергопотребления. Если обдуманно использовать эти режимы (SLEEP, STOP, STANDBY), приложение может работать без замены батареи очень долгое время. Можно сказать, что STM32 является низкопотребляющим микроконтроллером с высокой производительностью процессора. Во время обзора Cortex мы видели, что процессор Cortex может входить в режимы сниженного энергопотребления в которых ЦПУ и периферийные устройства Cortex останавливаются. Во время входа процессора Cortex в режим сниженного потребления, он может выдавать сигнал SLEEPDEEP для остальной части микроконтроллера, сигнализируя о своем переходе. ЦПУ Cortex переводится в режим сниженного энергопотребления через выполнение команд WFI и WFE. Режим, в который при этом входит STM32, зависит от настроек в регистре управления режимом энергопотребления. В этой главе мы поочередно рассмотрим каждый из режимов и сравним их.

## 6.1 Активный Режим RUN

В активном режиме STM32 выполняет программный код и потребляет больше всего энергии. В этой части мы рассмотрим различные способы сокращения общего потребления системы во время работы в активном режиме. Важно отметить, что все рассматриваемые способы можно использовать динамически. Это означает, что в те моменты, когда не нужна высокая производительность при выполнении кода, система может быть сконфигурирована для низкого энергопотребления, а в те моменты, когда возникает необходимость быстро выполнить код, например, при возникновении прерывания, система может перестроиться в сторону высокой производительности, но большего энергопотребления.

Процессор Cortex и большая часть микроконтроллеров STM32 могут работать на частотах до 72 МГц. На полной скорости STM32 потребляет более 30 мА. Первое, что можно сделать для снижения энергопотребления STM32 – отключить тактирование неиспользующихся периферийных устройств. Тактовые сигналы периферийных устройств могут включаться и выключаться динамически через модуль управления сбросом и системой синхронизации. Кроме этого, значительно снизить энергозатратность можно уменьшив частоту системного тактового сигнала. Если не требуется высокая скорость обработки программного кода, ФАПЧ может быть выключена, а STM32 тактироваться напрямую от внешнего выскокочастотного осциллятора HSE. Энергопотребление будет еще меньше, если вместо HSE-осциллятора использовать внутренний HSI-осциллятор. Аналогично, если сторожевой таймер оконного типа и часы реального времени не используются, можно выключить LSI-осциллятор.

### 6.1.1 Буфер Предвыборки и Полупериодный Режим

Если микроконтроллер работает напрямую от HSE-осциллятора на частоте 8 МГц, можно отключить буфер предвыборки FLASH и активировать полупериодный режим. Это приведет к возникновению дополнительных задержек, но сократит энергопотребление в активном режиме.

| APB1 | APB2 | Peripheral | Frequency | Prefetch | Half Cycle | WFI | Oscillator | Typical consumption at 25 °C in mA |
|---|---|---|---|---|---|---|---|---|
| DIV4 | DIV2 | ALL_ON | 72 MHz | ON | OFF | OFF | HSE | 33.15 |
| DIV 8 | DIV 8 | ALL_ON | 72 MHz | ON | OFF | OFF | HSE | 27.75 |
| DIV 8 | DIV 8 | USART | 72 MHz | ON | OFF | OFF | HSE | 23.65 |
| DIV4 | DIV2 | USART | 8 MHz | ON | OFF | OFF | HSE | 8.65 |
| DIV4 | DIV2 | USART | 8 MHz | OFF | OFF | OFF | HSE | 8.48 |
| DIV4 | DIV2 | USART | 8 MHz | OFF | OFF | ON | HSE | 1.68 |
| DIV4 | DIV2 | USART | 8 MHz | OFF | OFF | ON | HSI | 0.9 |

**Энергопотребление на полной скорости составляет около 34 мА, но на 8 МГц (9.6 DMIPS) это значении ниже 1 мА**.

*Перевод к рисунку: Peripheral – Периферийные устройства, Frequency - Частота, Prefetch - Предвыборка, Half Cycle – Полупериод, Oscillator – Typical consumption at 25°C in mA – Уровень энергопотребления при 25°C в мА , ALL_ON – ВСЕ_ВКЛ, MHz - МГц, ON - ВКЛ, OFF - ВЫКЛ.*

## 6.2 Режимы Сниженного Энергопотребления

При правильном использовании активного режима STM32 можно добиться снижения энергопотребления до 8.5 мА. Но чтобы разработать действительно экономичное приложение, нужно использовать режимы сниженного энергопотребления STM32.

### 6.2.1 Режим SLEEP

Первый из режимов сниженного энергопотребления – «спящий» режим SLEEP. По умолчанию, при выполнении команы WFE или WFI, процессор Cortex останавливает свой внутренний тактовый сигнал и прекращает исполнение программного кода приложения. В режиме SLEEP остальная часть STM32 продолжает функционировать. STM32 выходит из «спящего» режима по приходу прерывания от периферийного устройства. Когда STM32 находится в режиме SLEEP с включенными периферийными устройствами и тактированием с частотой 72 МГц от внешнего осциллятора HSE через ФАПЧ, энергопотребление составляет около 14.4 мА. Однако если STM32 подготовить для работы в режиме низкого энергопотребления, то есть, сначала выключить тактирование всех периферийных устройств (кроме тех, которые будут использоваться для вывода процессора Cortex из «спящего» режима), затем переключиться на внутренний осциллятор HSI (частота которого может быть уменьшена до 1 МГц с помощью делителя), мы можем снизить энергопотребление до 0.5 мА.

| Conditions | $f_{HCLK}$ | All APB peripherals enabled | All peripherals disabled | Unit |
|---|---|---|---|---|
| Running on HSE, AHB prescaler used to reduce the frequency | 72 MHz | 14.4 | 5.5 | mA |
| | 48 MHz | 9.9 | 3.9 | |
| | 36 MHz | 7.6 | 3.1 | |
| | 24 MHz | 5.3 | 2.3 | |
| | 16 MHz | 3.8 | 1.8 | |
| | 8 MHz | 2.1 | 1.2 | |
| | 4 MHz | 1.6 | 1.1 | |
| | 2 MHz | 1.3 | 1 | |
| | 1 MHz | 1.11 | 0.98 | |
| | 500 kHz | 1.04 | 0.96 | |
| | 125 kHz | 0.98 | 0.95 | |
| Running on high speed internal RC (HSI), AHB prescaler used to reduce the frequency | 64 MHz | 12.3 | 4.4 | |
| | 48 MHZ | 9.3 | 3.3 | |
| | 36 MHz | 7 | 2.5 | |
| | 24 MHz | 4.8 | 1.8 | |
| | 16 MHz | 3.2 | 1.2 | |
| | 8 MHz | 1.6 | 0.6 | |
| | 4 MHz | 1 | 0.5 | |
| | 2 MHz | 0.72 | 0.47 | |
| | 1 MHz | 0.56 | 0.44 | |
| | 500 kHz | 0.49 | 0.42 | |
| | 125 kHz | 0.43 | 0.41 | |

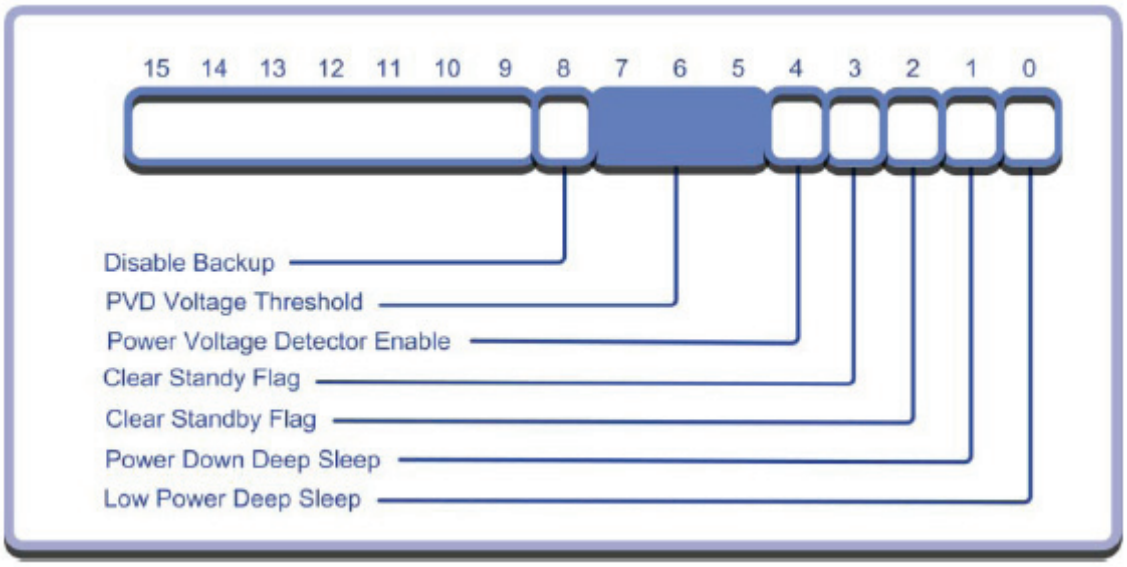
**В режиме SLEEP энергопотребление может быть снижено до 0.14 мА**.

*Перевод к рисунку: Conditions – Состояние; Running on HSE, AHB prescaler used to reduce the frequency – Тактирование от HSE, используется делитель частоты AHB; Running on high speed internal RC (HSI), AHB prescaler used to reduce the frequency – Тактирование от внутреннего высокочастотного RC осциллятора (HSI); All APB peripherals enabled – Все APB периферийные устройства включены; All peripherals*

*disabled – Все периферийные устройства выключены; Unit - Единица измерения; MHz - МГц; kHz - кГц; mA - мА.*

### 6.2.2 Режим STOP

Для того чтобы перевести STM32 в STOP режим, нужно установить бит SLEEPDEEP в регистре управления питанием Cortex и сбросить бит Power Down Deep Sleep (PDDS) в регистре управления питанием STM32.




Если осуществлена конфигурация для STOP режима, выполнение команды WFI или WFE приведет к выключению осцилляторов HSI и HSE. FLASH, SRAM и периферийные устройства остаются включенными, то есть состояние STM32 сохранится. Из STOP режима, также как и из SLEEP, можно выйти по прерыванию от периферийного устройства STM32. В режиме STOP тактирование всех периферийных устройств, за исключением модуля EXTI, останавливается. Использование прерывания EXTI позволяет вывести STM32 из режима STOP по изменению состояния вывода GPIO. Кроме того, EXTI содержит линию, по которой может уходить, или, наоборот, приходить запрос прерывания от часов реального времени. Так как RTC содержит свой собственный осциллятор (LSI или LSE осциллятор), он может генерировать периодические прерывания для вывода STM32 из режима STOP.

При входе STM32 в STOP режим, энергопотребление падает с единиц или десятков мА до уровня 24 мкА. Дальнейшее снижение энергопотребления достигается за счет перевода встроенного регулятора напряжения в специальный экономичный режим. Режим энергопотребления регулятора напряжения выбирается установкой

бита LPDS в регистре управления энергопотреблением STM32. При установке этого бита в момент входа STM32 в режим STOP, энергопотребление микроконтроллера снижается до 14 мкА. Включенный RTC будет потреблять дополнительные 1.4 мкА.

| Conditions | $V_{DD}/V_{BAT}$ = 2.4 V | $V_{DD}/V_{BAT}$ = 3.3 V | Unit |
|---|---|---|---|
| Regulator in Run mode, low-speed and high speed internal RC oscillators and high-speed oscillator OFF (no independent watchdog) | NA | 24 | µA |
| Regulator in Low Power mode, low speed and high speed internal RC oscillators and high speed oscillator OFF (no independant watchdog) | NA | 14 | |

| Symbol | Parameter | Conditions | Type | Unit |
|---|---|---|---|---|
| $t_{WUSTOP}$ | Wakeup from Stop mode (regulator in run mode) | Wakeup on HSI RC clock | 3.52 | µs |
| | Wakeup from Stop mode (regulator in run mode + WFI) | | 5.42 | |
| | Wakeup from Stop mode (regulator in Low power mode + WFE) | | 5.32 | |
| | Wakeup from Stop mode (regulator in Low power mode + WFI) | | 7.21 | |

Перевод таблиц:

| Условия | $V_{DD}/V_{BAT}$ = 2.4 В | $V_{DD}/V_{BAT}$ = 3.3 В | Единицы измерения |
|---|---|---|---|
| Регулятор в активном режиме, низкочастотный и высокочастотный встроенные и высокочастотный внешний осцилляторы выключены (независимый сторожевой таймер выключен) | неопределенно | 24 | мкА |
| Регулятор в экономичном режиме, низкочастотный и высокочастотный встроенные и высокочастотный внешний осцилляторы выключены (независимый сторожевой таймер выключен) | неопределенно | 14 | |

| Символ | Параметры | Условия | Значение | Единицы измерения |
|---|---|---|---|---|
| $t_{WUSTOP}$ | Выход из STOP режима (регулятор в активном режиме) | Переход в активный режим по тактовому сигналу от HSI RC | 3.52 | мкс |
| | Выход из STOP режима (регулятор в активном режиме + WFI) | | 5.42 | |
| | Выход из STOP режима (регулятор в | | 5.32 | |

| | | | | |
|---|---|---|---|---|
| | режиме энергосбережения + WFE) | | | |
| | Выход из STOP режима (регулятор в режиме энергосбережения + WFE) | | 7.21 | |

Время выхода из режима STOP в худшем случае составит 5.5 мкс с регулятором напряжения в своем активном режиме и 7.3 мкс с регулятором в режиме сниженного энергопотребления.

6.3 Режим STANDBY

Для того чтобы перевести STM32 в STANDBY режим, нужно установить бит SLEEPDEEP в регистре управления питанием Cortex и бит Power Down Deep Sleep (PDDS) в регистре управления питанием STM32. При выполнении команды WFI или WFE, STM32 перейдет в режим наименьшего энергопотребления. В режиме STANDBY микроконтроллер действительно выключается. Выключается встроенный регулятор напряжения и осцилляторы HSE и HSI. STM32 потребляет в этом режиме 2 мкА.

| Conditions | $V_{DD}/V_{BAT}$ = 2.4 V | $V_{DD}/V_{BAT}$ = 3.3 V | Unit |
|---|---|---|---|
| Low-speed internal oscillator and independent watchdog OFF, low speed oscillator and RTC OFF | NA | 2 | µA |
| Low-speed oscillator and RTC ON | 1.08 | 1.4 | |

| Symbol | Parameter | Conditions | Type | Unit |
|---|---|---|---|---|
| $t_{WUSTDBY}$ | Wakeup from Standby mode | Wakeup on HSI RC clock | 50 | µs |

**In Standby mode power consumption is 2uA with a wakeup time of 50uS.**

Перевод таблиц и надписи:

**In Standby … time of 50uS – В режиме Standby энергопотребление составляет 2 мкА, а время перехода в активный режим – 50 мкс**.

| Условия | $V_{DD}/V_{BAT}$ = 2.4 В | $V_{DD}/V_{BAT}$ = 3.3 В | Единицы измерения |
|---|---|---|---|
| Низкочастотный внутренний осциллятор | неопределенно | 2 | мкА |

| | | | |
|---|---|---|---|
| и независимый сторожевой таймер выключены, низкочастотный осциллятор и RTC выключены | | | |
| низкочастотный осциллятор и RTC включены | 1.08 | 1.4 | |

| Символ | Параметры | Условия | Значение | Единицы измерения |
|---|---|---|---|---|
| $t_{WUSTOP}$ | Выход из режима Standby | Переход в активный режим по тактовому сигналу от HSI RC | 50 | мкс |

Из режима STANDBY, точно так же как и из STOP, можно выйти по прерыванию от часов реального времени RTC. Можно выходить из режима по внешнему сбросу STM32, либо по переднему фронту на выводе 0 порта А. Этот вывод должен быть сконфигурирован для перевода микроконтроллера в активный режим (как вывод WKUP). Для этого нужно установить бит EWUP в регистре статуса и управления энергопотреблением. Как самый энергоэкономичный, режим Standby требует наибольшего времени выхода. Требуется около 50 мкс перед тем как ЦПУ Cortex начнет выполнение команд. В режиме STANDBY все данные SRAM, ЦПУ и регистрах STM32 теряются. Выход из режима STANDBY это практически как программный сброс.

### 6.4 Энергопотребление Области Резервирования

Область резервирования, в которую входит часть ОЗУ и RTC с питанием от батареи, остается активной во время всех режимов сниженного энергопотребления. Эта область потребляет около 1.4 мкА на 3.3 В.

### 6.5 Поддержка Отладки

В традиционных микроконтроллерных системах отладка приложения, использующего режимы сниженного энергопотребления, является очень болезненным занятием. Как только микроконтроллер входит в режим энергосбережения, он перестает отвечать на запросы отладчика, который затем выдает ошибку, или прекращает работу. В STM32 возможно конфигурировать режимы сниженного энергопотребления таким образом, чтобы HSI осциллятор

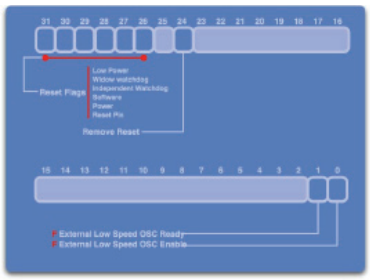
оставался включенным и выдавал специальный тактовый сигнал для отладочной системы CoreSight. Это означает, что вы можете полноценно отлаживать приложение, не избегая входов в режимы энергосбережения. Расширенные функции отладки STM32 конфигурируются в регистре DBG_MCU.

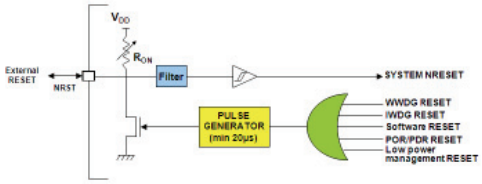
## 7. Обеспечение Надежности

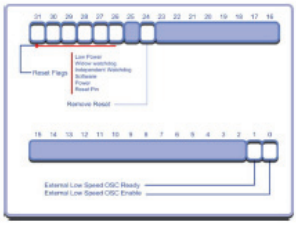
В STM32 заложено множество функций самотестирования и детектирования ошибок при исполнении кода приложения или работы самого STM32. Для контроля напряжения питания STM32, содержит свою собственную схему сброса, которая срабатывает, если напряжение питания опускается ниже допустимого значения $V_{DD}$. Кроме того, имеется программируемая схема детектирования напряжения питания, которая обнаруживает ошибки в работе источника питания до того, как сработает схема сброса. При этом генерируется прерывание, по которому микроконтроллер переводится в безопасное состояние. Система синхронизации содержит систему безопасности CSS, которая мониторит HSE-осциллятор. При нарушениях в работе осциллятора, CSS переключает STM32 на HSI-осциллятор. Корректное выполнение программного кода контролируется с помощью двух встроенных сторожевых таймеров. Первый сторожевой таймер – оконного типа, должен обновляться в определенный интервал времени. Второй является независимым сторожевым таймером, тактируется от отдельного осциллятора. Время хранения данных во внутренней FLASH памяти составляет 30 лет при температуре 85°C (лучший показатель среди микроконтроллеров общего применения). Все эти функции не подходят для систем с самыми высокими требованиями к обеспечению безопасности (в таких системах требуется внешний сторожевой таймер). Однако на базе STM32 можно строить самокорректирующиеся системы с использованием тех же самых техник, что используются в авиа- и автомобильных приложениях, но применяя гораздо более дешевые компоненты.

### 7.1 Управление Сбросом

Помимо внешней линии сброса, STM32 содержит множество источников сброса, в число которых входят: внутренние сторожевые таймеры, программный сброс через NVIC, встроенная схема сброса по включению/выключению питания и схема детектирования падения напряжения питания. При возникновении сброса, в регистре RCC control and status register устанавливается флаг, по которому в дальнейшем можно определить источник сброса. Состояние флагов остается неизменным до следующего сброса по включению питания. Установленный флаг также можно сбросить, записав логическую единицу по его адресу.



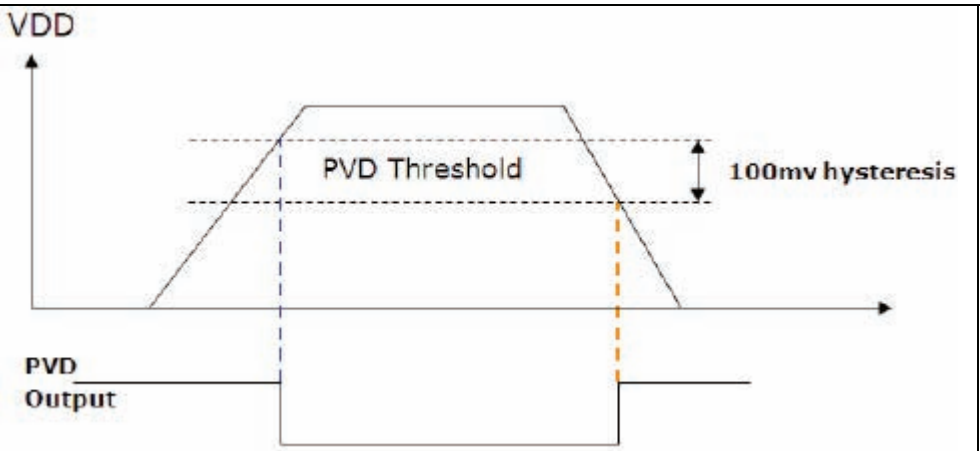


**STM32 имеет множество источников сброса. С помощью флагов в регистре RCC control and status register можно определить источник последнего сброса**.

7.2 Детектирование Падения Напряжения

В состав встроенного супервизора напряжения питания STM32 входит модуль детектирования падения напряжения питания PVD. Порог PVD может быть задан в диапазоне от 2.2 В до 2.9 В с шагом 0.1 В. Этот порог задается в регистре управления энергопотреблением.




**Детектор падения напряжения питания может генерировать прерывание, если напряжение источника питания опускается ниже заданного уровня**.

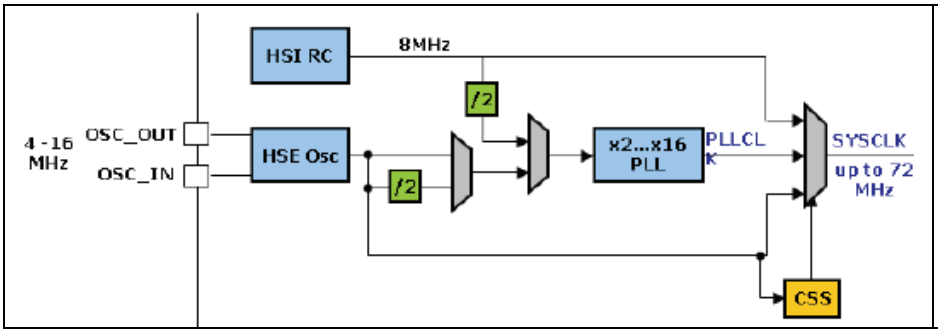
*Перевод к рисунку: PVD Threshold – Порог PVD, 100mv hysteresis – Гистерезис 100 мВ, PVD Output – Выход PVD*.

Выход PVD соединен с линией 16 модуля внешних прерываний. Так как линии EXTI могут генерировать прерывание по переднему и заднему фронтам входного сигнала, модуль PVD может использоваться для отслеживания превышения и падения напряжения ниже заданного порога.

7.3 Система Безопасности Системы Синхронизации

В большинстве приложений на базе STM32 основной тактовый сигнал, использующийся для процессора Cortex и периферийных устройств STM32, будет

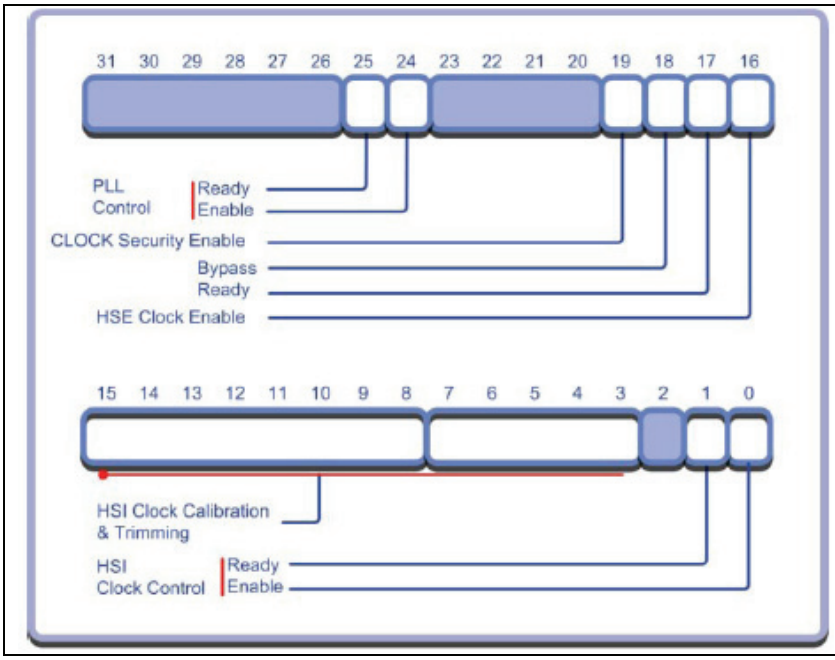
генерироваться внешним кварцем, подключенным к выводам HSE. Тракт передачи этого тактового сигнала содержит Систему Безопасности Системы Синхронизации, которая отслеживает внешний кварц. При каких-либо нарушениях система синхронизации переключится на встроенный осциллятор 8 МГц.




**Система Безопасности Системы Синхронизации генерирует прерывание при нарушениях в работе внешнего осциллятора и переключает систему синхронизации на встроенный RC-осциллятор.**

Перевод к рисунку: PLL – ФАПЧ, Osc – Осц, MHz – МГц.

Система Безопасности Системы Синхронизации включается через установку бита Clock Security Enable в регистре управления RCC.



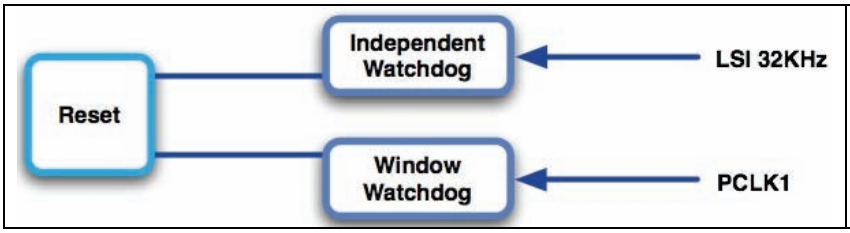

**Система Безопасности Системы Синхронизации включается через установку бита CLOCK Security Enable в регистре управления RCC.**

Система Безопасности Системы Синхронизации содержит линию прерываний, соединяющую ее с входом таймера с расширенными функциями, который в свою очередь подключается к линии немаскируемых прерываний NVIC. Это сделано для того, чтобы в случае выхода из строя основного осциллятора, выходы

ШИМ таймера с расширенными функциями сразу же аппаратно переводились в преопределенное безопасное состояние. Таким образом, гарантируется то, что силовая часть не будет самопроизвольно управляться ШИМ выводами. Это особенно важно для приложений управления приводами.

## 7.4 Сторожевые Таймера

STM32 содержит два отдельных сторожевых таймера. Независимый сторожевой таймер полностью отделен от основной части STM32. Он расположены в области с резервным питанием и тактируется от внутреннего низкочастотного осциллятора Low Speed Oscillator (LSI). Сторожевой таймер оконного типа входит в состав основной системы STM32 и тактируется от шины периферийных устройств 1. Оба таймера включаются индивидуально и могут использоваться одновременно.

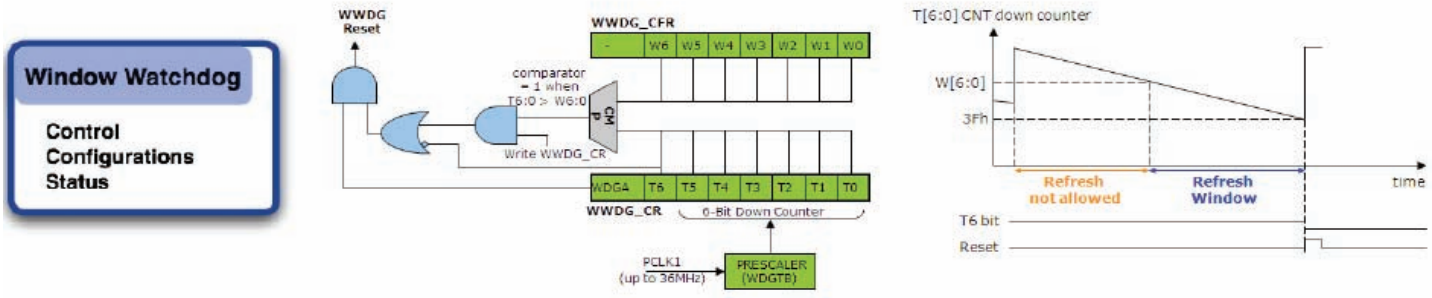



**STM32 содержит два встроенных сторожевых таймера, один из которых имеет свой собственный осциллятор**.

Перевод к рисунку: Reset - Сброс, Independent Watchdog – Независимый Сторожевой Таймер, Window Watchdog – Сторожевой Таймер Оконного Типа.

### 7.4.1 Сторожевой Таймер Оконного Типа

Сторожевой таймер оконного типа – это расширенная версия традиционного сторожевого таймера. После включения, он начинает считать в обратном направлении и сгенерирует сброс при переходе от 0x40 к 0x3F, если бит T6 сброшен. В конфигурационном регистре хранится еще одно дополнительное числовое значение. Если ПО приложения обновит счетчик сторожевого таймера в тот момент, когда значение счетчика больше дополнительного числового значения, будет сгенерирован сброс. Таким образом, сторожевой таймер оконного типа можно обновлять только в определенный временной интервал (окно), что придает больше уверенности в том, что приложение работает в заданных рамках.




*Перевод к рисунку: Window Watchdog – Оконный Сторожевой Таймер, Control - Управление, Configuration - Конфигурация, Status - Статус, Reset – Сброс; comparator = 1 when T6:0 > W6:0 – компаратор = 1 когда T6:0 > W6:0; Write WWDG_CR – Запись WWDG_CR; 6-Bit Down Counter – Вычитающий Счетчик; up to 36*

*MHz – до 36 МГц, PRESCALER – ДЕЛИТЕЛЬ ЧАСТОТЫ, down counter – вычитающий счетчик, Refresh not allowed – Обновление запрещено, Refresh Window – Окно Обновления, time - время, T6 bit – бит T6, Reset - Сброс.*

Сторожевой таймер оконного типа представляет собой шестибитный вычитающий счетчик, тактирующийся от PCLK1 через 12-битный делитель частоты, который снижает частоту PCLK1 в 4096 раз. Четыре оставшихся бита делителя частоты могут использоваться пользователем, чтобы дополнительно снижать частоту в 1, 2, 4 и 8 раза. Биты делителя частоты – это 6-й и 7-й биты регистра управления.


Перевод к рисунку: *Window Watchdog – Оконный Сторожевой Таймер, Control - Управление, Configuration - Конфигурация, Status – Статус.*

Период сторожевого таймера оконного типа рассчитывается по формуле:

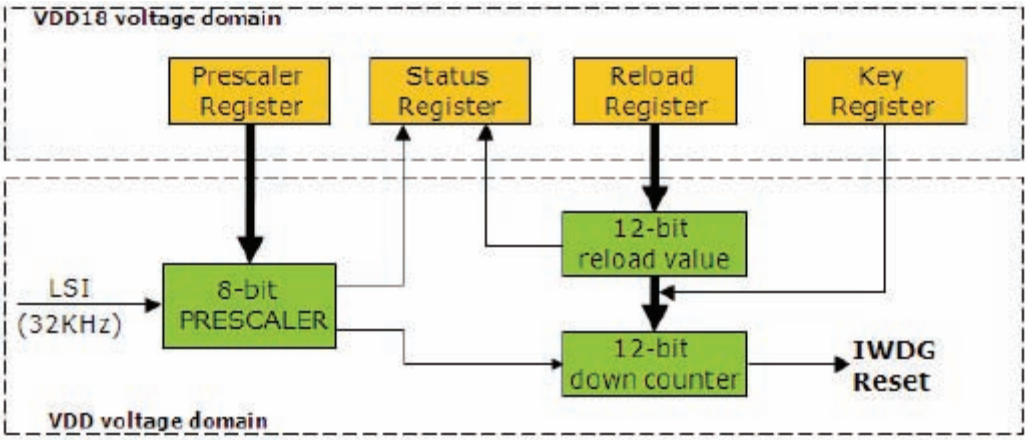
Twwdg = Tpclk1 x 4096 x 2POW (WDGTB) x (перезагрузочное значение + 1)

Если Pclk1 работает на максимальной тактовой частоте 36 МГц, минимальный период сторожевого таймера оконного типа составляет 910 мкс, а максимальный – 58.25 мс.

После конфигурации сторожевого таймера оконного типа, его можно включить, установив активационный бит в регистре управления. После программной активации сторожевого таймера, он не может быть выключен, если только с помощью сброса.

7.4.2 Независимый Сторожевой Таймер

Несмотря на то, что независимый сторожевой таймер находится на одном кристалле с основной частью STM32, он содержит свой собственный осциллятор. Независимый сторожевой таймер подключен к напряжению питания $V_{DD}$, которое остается активным даже в режимах сниженного энергопотребления STOP и STANDBY.




*Перевод к рисунку: VDD18 voltage domain – Питание от напряжения VDD18, Prescaler Register – Регистр Делителя Частоты, Status Register – Регистр Статуса, Reload Register – Регистр Перезагрузки, Key Register – Регистр Ключа, 8-bit PRESCALER – 8-битный ДЕЛИТЕЛЬ ЧАСТОТЫ, 12-bit reload value – 12-битное перезагрузочное значение, 12-bit down counter – 12-битный вычитающий счетчик, 32 KHz – 32 кГц, VDD voltage domain – Питание от напряжения VDD, Reset - Сброс.*

Независимый сторожевой таймер представляет собой 12-битный вычитающий счетчик, сбрасывающий STM32, когда досчитывает до нуля. Он тактируется от низкочастотного внутреннего осциллятора через 8-битный делитель частоты. Несмотря на то, что номинальная частота LSI-осциллятора составляет 32.768 кГц, на практике она может варьироваться в диапазоне между 30 кГц и 60 кГц. При инициализации независимого сторожевого таймера сначала устанавливается значение регистра делителя частоты, чтобы снизить частоту LSI-осциллятора в 4, 8, 16… или 256 раз. Минимальный период независимого сторожевого таймера составляет 0 с и максимальный – немного более 26 с. Значение периода задается напрямую в регистре перезагрузки.


**Независимый сторожевой таймер – это таймер обратного счета со своим собственным осциллятором. Он расположен в области с резервным напряжением питания, следовательно, остается включенным во время режимов работы STOP и STANDBY.**

*Перевод к рисунку: Independent Watchdog – Независимый Сторожевой Таймер, Key Register – Регистр Ключа, Prescaler Register – Регистр Делителя, Reload – Значение Перезагрузки, Status - Статус*

С помощью опциональных байтов малого информационного блока FLASH памяти можно настроить независимый сторожевой таймер на запуск сразу после сброса, или по команде из программного кода. Из приложения независимый сторожевой таймер можно запустить, записав значение 0xCCCC в регистр ключа. Таймер начнет считать в обратном направлении от начального значения 0xFFF. Для

обновления независимого сторожевого таймера нужно записать 0xAAAA в регистр ключа. При этом в регистр вычитающего счетчика копируется перезагрузочное значение.

Обычно очень трудно отлаживать приложение с работающим сторожевым таймером. Как только ЦПУ останавливается, сторожевой таймер не может быть обновлен, что приводит к сбросу системы и завершению сессии отладки. В таком случае приходится отключать сторожевой таймер, чтобы не прерывать отладку. Следовательно, очень сложно протестировать работу полноценного приложения, отследить корректное обновление счетчиков сторожевых таймеров. В STM32 эта проблема решается с помощью регистра MCUDBG, через который можно настроить чтобы независимый сторожевой таймер и сторожевой таймер оконного типа останавливались в те моменты времени, когда ЦПУ Cortex-M3 находится под управлением системы отладки CoreSight. Эта возможность позволяет исполнять программный код в пошаговом режиме с работающими сторожевыми таймерами, которые будут инкрементироваться синхронно с тактовыми циклами ЦПУ.

## 7.5 Особенности Периферийных Устройств

Пользовательские периферийные устройства имеют ряд особенностей для обеспечения безопасное функционирование STM32. Они подробно описаны в соответствующих разделах, ниже мы их просто перечислим.

### 7.5.1 Блокировка Портов GPIO

Во время инициализации портов GPIO, каждая линия ввода/вывода может быть сконфигурирована как вход или выход. После чего конфигурацию портов можно блокировать, что предотвращает случайное изменение настроек в ходе выполнения программы. Каждый порт можно блокировать индивидуально.

### 7.5.2 Аналоговые Сторожевые Схемы

Каждый аналогово-цифровой преобразователь содержит две аналоговых сторожевых схемы. Эти сторожевые схемы можно использовать для генерирования прерывания при выходе напряжения за пределы диапазона АЦП.

### 7.5.3 Вход Экстренного Отключения

В приложениях управления приводами вход экстренного отключения таймера с расширенными функциями используется для перевода трех комплементарных выводов ШИМ в предопределенное состояние по изменению состояния вывода

экстренного отключения или при нарушениях в работе основного осциллятора STM32.

## 8. Модуль FLASH памяти

Встроенная FLASH память STM32 разделена на три области. Первая область – это основная FLASH память, предназначенная для хранения программного кода. Эта область памяти имеет разрядность 64-бит для эффективности использования буфера предвыборки и разделена на страницы по 4K для операций записи и стирания. Обеспечивается 10000 циклов перезаписи и сохранность данных в течение 30 лет при температуре 85°C. Длительность сохранности данных большинства FLASH-микроконтроллеров других производителей оценивается для 25°C, так что STM32 имеют серьезное преимущество. Кроме основной области памяти для программного кода, имеются две области поменьше: большой информационный блок и малый информационный блок. Большой информационный блок, размером 2K, содержит загрузчик, запрограммированный на этапе изготовления, предназначенный для загрузки программного кода через USART 1. Малый информационный блок содержит шесть конфигурационных байтов, использующихся для настройки режима сброса STM32 и защиты памяти.

### 8.1 Безопасность и Программирование Встроенной FLASH

Записывать информацию во FLASH память можно через встроенный загрузчик, JTAG, или во время исполнения программного кода через набор регистров, называемых контроллером записи и стирания FLASH (FPEC). Контроллер FPEC используется также для программирования опционных байтов в малом информационном блоке.


**Модуль FPEC используется для программирования FLASH памяти во время работы приложения. FLASH память можно защитить от чтения и записи через средства отладки**.

*Перевод к рисунку: Flash Program & Erase – Запись и Стирание Flash, Access Control – Контроль Доступа, Key Register – Регистр Ключа, Option Key Byte – Опциональный Байт Ключа, Status - Статус, Control - Управление, Address - Адрес, Option Byte – Опциональный Байт, Write Protect – Защита от Записи*.

8.2 Стирание и Запись

После сброса, регистры FPEC заблокированы. Для того чтобы их разблокировать, нужно записать в Регистр Ключа последовательность чисел 0x45670123, 0xCDEF89AB. Если при записи допустить ошибку, FPEC останется заблокированным до следующего сброса. Как только FPEC разблокирован, можно осуществлять стирание и запись основной FLASH памяти. В пределах основного блока FLASH памяти можно производить массовое стирание или стирание страниц по 4K. Массовое стирание выполняется просто через установку битов mass erase и start в регистре управления. Если бит busy в этом же регистре сброшен, все ячейки памяти будут сброшены в состояние 0xFFFF. Стирание страницы осуществляет также просто, нужно записать начальный адрес страницы FLASH памяти в регистр адреса, затем установить биты page erase и start в регистре управления. Опять же, если бит busy не установлен, страница будет стерта. Новые данные можно записывать в ячейку FLASH памяти, только когда она стерта. Чтобы осуществить запись, нужно установить бит program в регистре управления и затем выполнять запись полуслов по нужному адресу. Если ячейка памяти по этому адресу стерта, и защита от записи не установлена, контроллер FPEC запишет новое значение в ячейку.

8.3 Опционные Байты

Малый информационный блок содержит восемь опционных байтов, конфигурируемых пользователем. Четыре из них используются для установки защиты от записи основной FLASH памяти. Пятый байт устанавливает защиту от чтения, которая необходима для предотвращения доступа к областям памяти на

этапе отладки микроконтроллера. Шестой байт используется для конфигурации режимов сброса и сниженного энергопотребления. И два последних байта представляют собой две обычные ячейки FLASH памяти, доступные для пользователя. Перед тем как записывать в опционные байты, нужно разблокировать FPEC, как описано выше. Затем необходимо разблокировать опционные байты записав те же самые два числа в регистр option key register. Опционные байты имеют отдельные от основной FLASH памяти процедуры записи и стирания. Малый информационный блок стирается с помощью установки бита OPTER в регистре управления и затем бита STRT. Если бит BSY сброшен, малый информационный блок стирается. Для записи в опционный байт нужно установить бит OPTPG в регистре управления FLASH и производить запись полуслов в опционный байт. Каждый опционный байт хранится как полуслово в младшем байте слова, в то время как старшем байте хранится его комплементарное значение, которое автоматически вычисляет контроллер FPEC.

### 8.3.1 Защита от Записи

Каждый бит в байте защиты от записи позволяет заблокировать запись в определенную страницу FLASH памяти. Отключить защиту можно стерев малый информационный блок.

### 8.3.2 Защита от Чтения

Если включена защита от чтения, все обращения на чтение FLASH памяти блокируются во время нахождения микроконтроллера в режиме отладки. Доступ к SRAM остается разрешенным, поэтому код может загружаться и исполняться из нее. Таким образом, отключить защиту от чтения можно с помощью кода, исполняемого из SRAM. Однако при отключении защиты от чтения происходит массовое стирание FLASH памяти, чтобы предотвратить несанкционированный доступ к программному коду приложения. При включении защиты от чтения, включается защита от записи во FLASH, предотвращая тем самым запись некорректных команд в область, содержащую таблицу векторов прерываний. FLASH память STM32 защищена, если байт защиты от чтения и его комплементарный байт содержат значения 0xFF. Чтобы снять защиту, нужно записать 0xFA и дополнительный код этого числа как полуслова в опционный байт защиты от чтения.

### 8.3.3 Конфигурационный Байт

Конфигурационный Опционный байт содержит три активных бита. Два бита управляют процессом входа STM32 в режимы STANDBY и STOP. Можно задать, чтобы при входе в тот и другой режим генерировался сброс. В этом случае цифровые порты ввода/вывода будут сконфигурированы как входы, снижая тем самым энергопотребление STM32. ФАПЧ и внешний осциллятор тоже отключатся и

кристалл переключится на внутренний высокоскоростной RC-осциллятор. Третий бит конфигурационного Опционного байта конфигурирует процесс активации независимого сторожевого таймера. Этот сторожевой таймер имеет режим аппаратного сторожевого таймера, который включается сразу после сброса микроконтроллера и режим программного сторожевого таймера, в котором таймер запускается под управлением ПО.

## 9. Средства Отладки

Становление ARM7 и ARM9 как одних из самых популярных ядер стандартных микроконтроллеров послужило причиной массового производства отладочных средств для них. Все ведущие производители инструментария, такие как GCC, Greenhills, Keil, IAR и Tasking предлагают компиляторы для ARM ядер. С появлением процессоров Cortex, все эти средства отладки получили расширение для поддержки системы команд Thumb-2. Если вы уже используете ARM-микроконтроллеры, есть шансы, что среда разработки может генерировать программный код для STM32. В худшем случае потребуется просто запросить обновление у поставщика.

Если вы собираетесь использовать ARM-микроконтроллер впервые, у вас есть возможность подобрать средства разработки в соответствии с вашими предпочтениями. Несмотря на то, что в настоящее время очень трудно найти плохие отладочные средства, о двух компиляторах все же стоит упомянуть. Первый, GCC или GNU компилятор представляет собой открытое средство разработки, свободно доступное для скачивания и использования. GCC компилятор интегрирован во многие коммерческие среды разработки и отладчики в составе недорогих отладочных и оценочных наборов. Хотя GCC компилятор надежный и стабильный, наш опыт показывает, что эффективность генерируемого им кода ниже по сравнению с коммерческими компиляторами. Также не существует прямой линии поддержки для GCC компиляторов, то есть если вы столкнетесь с проблемой, сроки разработки могут сильно затянуться. Второй, коммерческий компилятор ARM Real View является оригинальным и самым проработанным Си-компилятором, разработанным компанией ARM для своих ядер. Компилятор RealView поставляется в отладочном наборе ARM RealView. Этот набор предназначен для разработчиков систем на кристалле и не подходит для микроконтроллерных проектов. Однако с 2006 года RealView компилятор был интегрирован в набор Keil Microcontroller Development Kit (MDK-ARM). Название этого набора говорит о том, что это полноценное инструментальное средство, предназначенное исключительно для ARM-микроконтроллеров. MDK прост в использовании (четырьмя опциями конфигурируется весь проект) представляет собой полноценный инструментальный набор от одного производителя.

Выбор между GCC и коммерческим компилятором частично зависит от бюджета проекта. При реализации простого проекта не принято затрачиваться на коммерческие инструментальные средства. Однако если вы планируете и в дальнейшем использовать ARM-микроконтроллеры, «дорогой» отладочный

комплект очень быстро себя окупит, сократив время разработки. Очень важно принять во внимание относительный уровень вашего опыта. Если вы продвинутый разработчик встроенных систем, вероятнее всего вы сможете разработать весь проект с помощью GCC компилятора. Однако если вы новичок в этом деле, можете сильно запутаться.

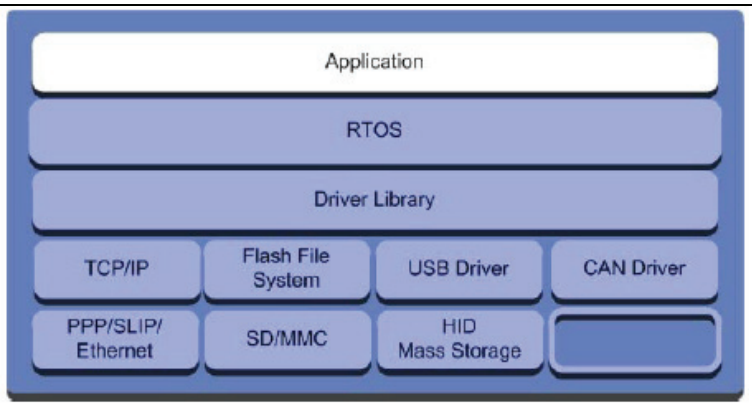
### 9.1.1 Оценочные Наборы

Многие производители компиляторов также предлагают оценочные наборы или стартер киты. Обычно это отладочная плата и ограниченная (по времени или функциональности) версия среды разработки. Полный список оценочных наборов можно найти на сайте компании ST. Одним из лучших оценочных наборов является Hitex STM32 Performance Stick, по цене около 50 Евро. Он выполнен в формфакторе USB-стика, позволяет разрабатывать и отлаживать неограниченный по объему программный код при помощи компилятора GCC или Tasking, через среду разработки HiTOP. Кроме STM32, на оценочной плате размещен второй микроконтроллер STR750. АЦП и таймера этого микроконтроллера используются для измерения энергопотребления STM32 и задержек прерываний. Эта информация передается в «инструментальную панель» на ПК. Инструментальная панель позволяет экспериментировать с различными функциями STM32, получая достоверные данные по энергопотреблению, времени выхода из режимов сниженного энергопотребления и др.


**Hitex Performance Stick – это очень дешевый оценочный комплект для STM32. В его состав входит неограниченная среда разработки на базе отладчика HiTOP и компилятора GCC. Эта же среда разработки и компилятор входят в комплект JTAG-отладчика Hitex Tantino, который можно использовать для создания полноценного устройства**.

### 9.1.2 Библиотеки и Протоколы Стеков

Для ускорения разработки программного кода приложения, компания ST предлагает программные библиотеки для STM32, которые можно свободно скачать на сайте компании. Эти программные библиотеки содержат драйвера для всех встроенных периферийных устройств микроконтроллера. Это предоставляет вам базу для построения вашего приложения. Самым сложным периферийным устройством в существующих на сегодняшний день версиях STM32 является USB device контроллер. Для того чтобы помочь разработать USB-устройство, ST также предлагает бесплатный USB developer's kit. Его тоже можно скачать на сайте компании. В состав USB developer's kit входит библиотека USB и демонстрационные приложения для HID, Mass Storage, Audio и Device Field Upgrade.




**В связи с возрастающей сложностью периферийных устройств микроконтроллера, очень важно выбирать средства разработки с хорошей поддержкой протоколов и программных стеков приложения**.

*Перевод к рисунку: Application - Приложение, Driver Library – Библиотека Драйверов, Flash File System – Файловая Система Flash, USB Driver – Драйвер USB, CAN Driver – Драйвер CAN.*

Новые версии STM32 будут иметь в своем составе еще более сложные периферийные устройства (Ethernet MAC, TFT интерфейс и др.). С повышением сложности, разработка всего кода приложения с нуля станет практически невозможной. Таким образом, при выборе средств разработки, очень важно оценить доступность стеков протоколов (TCP/IP и др.) и другого ПО (GUI и др.), которые могут потребоваться для реализации проекта. Идеальный вариант, если все это будет от одного производителя и интегрировано в среду разработки.

### 9.1.3 RTOS

Если вы до этого времени использовали восьми и шестнадцатиразрядные микроконтроллеры, скорее всего вы не работали с RTOS. Как мы видели ранее, Cortex-M3 отличается большей вычислительной мощностью по сравнению с микроконтроллерами своего ценового диапазона и поддерживает RTOS. При использовании RTOS разработка кода приложения становится более абстрактной, облегчается повторное использование кода, упрощается управление проектом и отладка. Использование RTOS также придает структурный вид коду приложения, что

подвигает вас к планированию приложения до начала написания кода. Для ARM и Cortex доступно больше RTOS, чем для других ЦПУ. Многие производители микроконтроллеров предлагают свои собственные, а также портированные RTOS от третьих партнеров. Самой популярной открытой операционной системой является “FreeRTOS”, которая доступна на сайте www.freertos.org. Коммерческая версия “FreeRTOS” называется “SafeRTOS”, она протестирована на соответствие стандарту IEC 61508 и тоже доступна на этом сайте.

## 10. Заключение

Если вы прочитали эту книгу, я думаю вы согласитесь, что Cortex STM32 – это новое поколение недорогих микроконтроллеров общего применения. Содержащие высокопроизводительный процессор с детерминистической системой прерываний и сложными периферийными устройствами, микроконтроллеры STM32 подходят для множества промышленных и бытовых применений. Режимы сниженного энергопотребления позволяют рассматривать STM32 как решения для портативных устройств с питанием от батареи.

## 11. Список Использованной Литературы


| | |
|---|---|
| Cortex-M3 Technical reference manual | ARM Ltd |
| ARMv7-M architectural reference manual | ARM Ltd |
| ARM Architectural reference manual Thumb2 supplement | ARM Ltd |
| STM32F103xx User Manual | ST Microelectronics |
| STM32F10xxx FLASH Programming manual | ST Microelectronics |


Эта книга рекомендуется, как практическое руководство для всех, кто использует Cortex-M3 микроконтроллеры семейства STM32 компании ST Microelectronics.

На протяжении последних шести или семи лет одним из главных направлений в области разработки микроконтроллеров являлась адаптация ARM7 и ARM9 как ядер для микроконтроллеров общего применения. На сегодняшний день доступно около 240 ARM-микроконтроллеров от многочисленных производителей. Компания ST Microelectronics выпустила первые микроконтроллеры на базе нового ядра ARM Cortex-M3, STM32. Эти устройства устанавливают новые стандарты производительности и цены, способности выполнения алгоритмов управления жесткого реального времени при низком энергопотреблении.