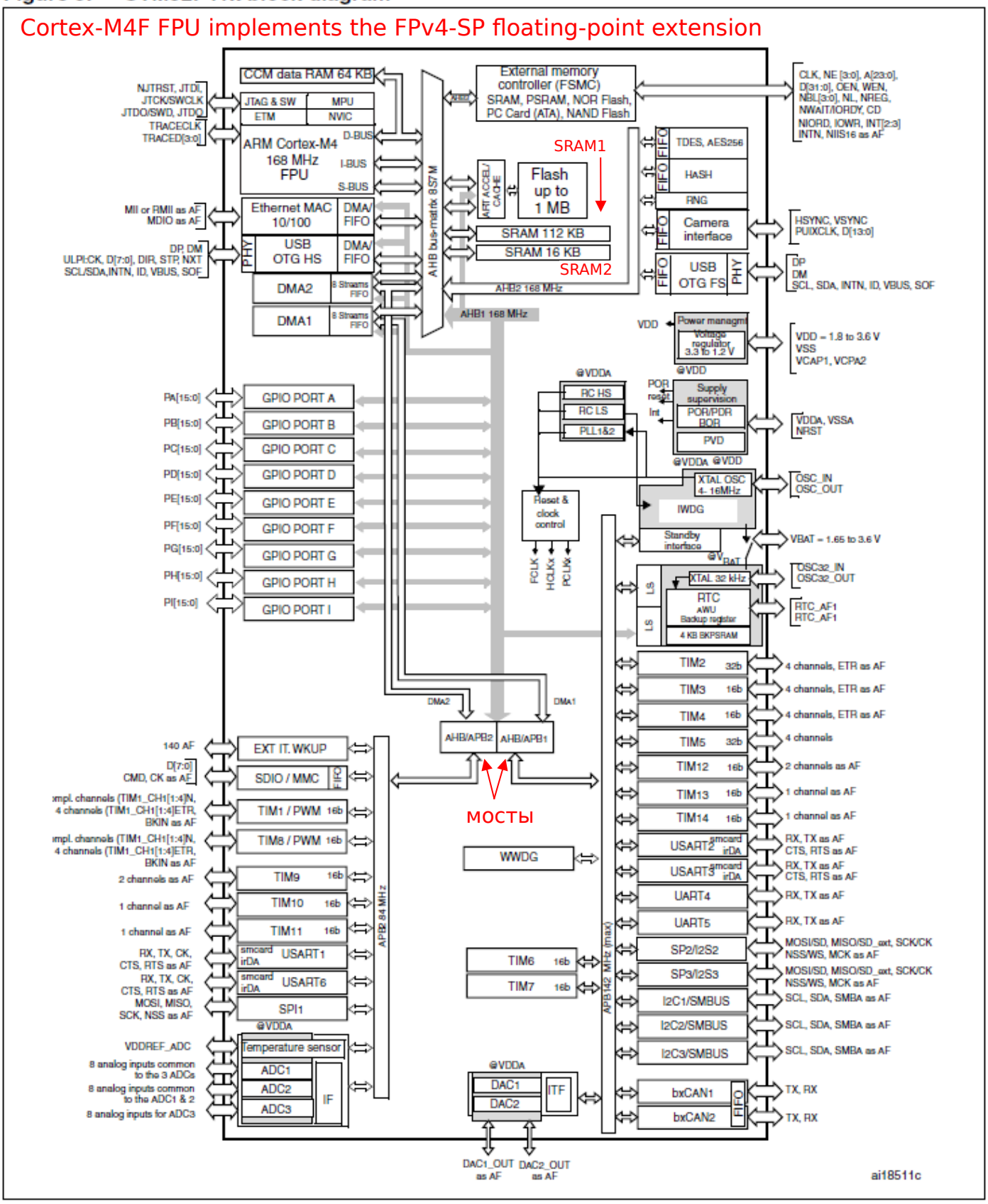
Содержание

1. БЛОК-СХЕМА STM32F417

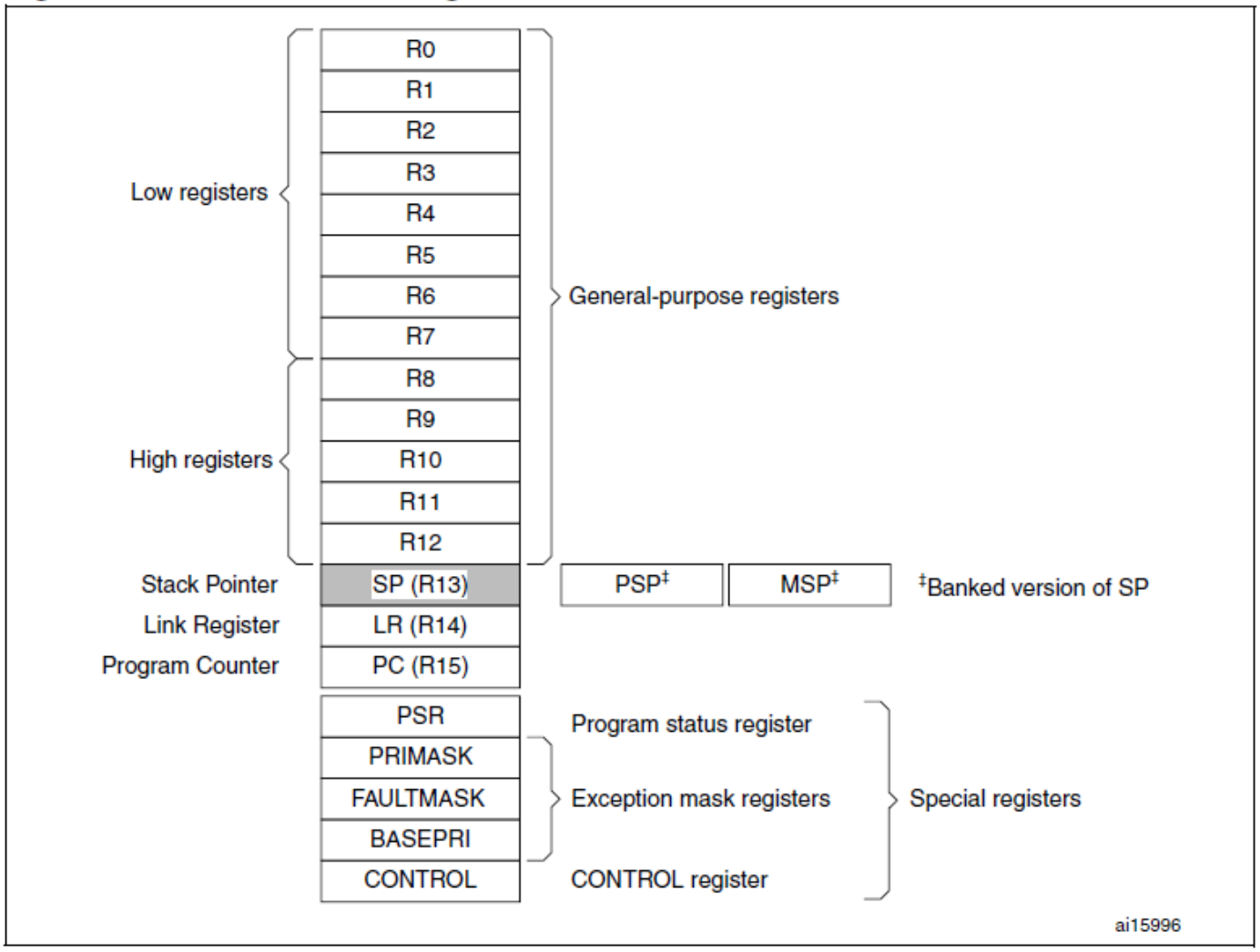
Figure 5. STM32F41x block diagram



Протоколы внутренних шин (разработано ARM Limited):
AHB – Advanced High Performance Bus (высокоскоростной)
APB – Advanced Peripheral Bus (низкоскоростной)

2. РЕГИСТРЫ ЯДРА ПРОЦЕССОРА

Figure 2. Processor core registers



R0-R12	Регистры общего назначения
SP	Указатель стека. Какой в данный момент указатель стека используется (PSP или MSP) определяется режимом процессора и регистром CONTROL. PSP – Process SP (используется прикладной программой) MSP – Main SP (по умолчанию привилегированный доступ к памяти) Можно всегда использовать только MSP
LR	Связующий регистр. Содержит информацию о возврате (адрес и т. п.) из подпрограмм, функций и исключений (прерываний).
PC	Счётчик программы. Содержит текущий адрес программы. После reset’а процессор загружает в PC значение вектора сброса (reset vector). Вектор сброса расположен с адреса 0x00000004.
PSR	Регистр статуса программы. Комбинирует в себе

	регистры: **APSR** (Application Program Status Register) – содержит текущее состояние условных флагов (флаг переноса, флаг переполнения и т. п.) после выполнения предыдущей инструкции **IPSR** (Interrupt Program Status Register) – содержит номер типа исключения текущей ISR (Interrupt Service Routine) **EPSR** (Execution Program Status Register) – содержит Thumb state bit и биты состояния выполнения условной инструкции IT (If-Then) и инструкции ICI (Interruptible-Continuable Instruction).
PRIMASK	Предотвращает активацию всех исключений с конфигурируемым приоритетом. Этот регистр содержит всего 1 бит разрешающий или запрещающий генерацию исключения от исключений с конфигурируемым приоритетом.
FAULTMASK	Предотвращает активацию всех исключений кроме NMI (Non-Maskable Interrupt). Содержит только 1 бит разрешающий или запрещающий.
BASEPRI	Определяет минимальный приоритет для обработки исключений. Предотвращает все исключения с приоритетом равным значению регистра BASEPRI и ниже.
CONTROL	Управляет какой использовать стек и какой использовать уровень привилегии для программы. Индицирует активен ли FPU.

Processor mode and privilege levels for software execution (Режимы процессора и уровни привилегии для выполнения программ):

1) Thread mode: выполнение прикладной программы.

В этом режиме процессор сразу после reset'a. Регистр CONTROL определяет уровень привилегии для выполнения прикладной программы

2) Handler mode: выполнение исключений (прерываний).

После выполнения этого режима процессор обратно возвращается в Thread mode.

1) Unprivileged:

Непривилегированный уровень. Прикладная программа:

- Имеет ограниченный доступ к инструкциям MSR, MRS и CPS,
- Не имеет доступа к system timer, NVIC, system control block.
- Может иметь ограниченный доступ к памяти и периферии.
- Должна использовать инструкцию SVC (super visor call) для передачи управления привилегированной программе.

2) Privileged:

Прикладная программа имеет полный доступ к ресурсам микроконтроллера.

СТЕК

Процессор использует полный ниспадающий стек. SP указывает на последний объект помещенный в стек. При помещении нового объекта в стек: SP декрементируется, а затем объект помещается в стековую память.

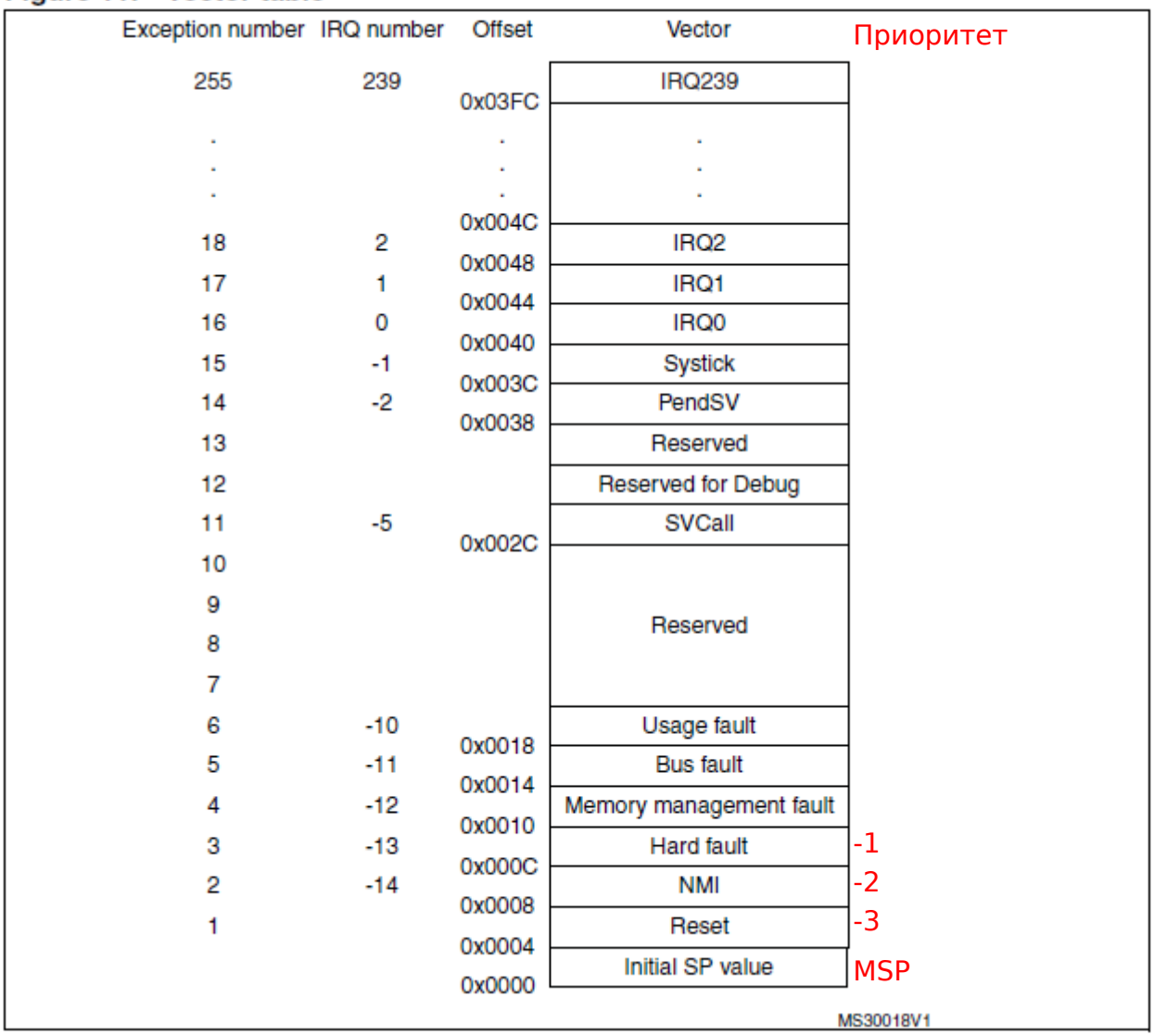
Процессор реализует два стека: Main stack и Process stack с независимыми копиями указателя стека: MSP и PSP соответственно.

Processor mode	Used to execute	Privilege level	Stack used
Thread	Applications	Privileged or unprivileged*	Main stack or Process stack*
Handler	Exception handlers	Always privileged	Main stack

* что именно используется определяется в регистре CONTROL.

3. NVIC

Figure 11. Vector table



Состояния исключений:

- Inactive
- Pending (исключение ожидает обработки процессором)
- Active (исключение обрабатывается процессором и не завершено)
- Active and Pending (исключение обрабатывается процессором и в очереди есть такое же исключение, ожидающее обработки)

Типы исключений:

Reset	Вызывается при запуске процессора или по сигналу сброс. После Reset выполнение начинается как Privileged в Thread mode с адреса, предоставленным в ячейке Reset (адрес 0x00000004 в Vector table).
NMI	Немаскируемое прерывание. Вызывается периферией или программно. NMI не м. б. маскировано или отложено другим прерыванием!!! NMI не м. б. вытеснено другим исключением, кроме

	Reset.
Hard fault	Возникает в случае: - ошибка во время обработки какого-нибудь исключения - какое-то исключение не м. б. обработано никаким механизмом обработки исключений.
Memory management fault	MPU (Memory Protection Unit) генерирует это исключение для защиты памяти. Например, когда идёт обращение к области памяти, помеченной как XN (execute never).
Bus fault	Ошибка на шине памяти (как инструкций (I-bus), так и данных (D-bus)).
Usage fault	Ошибка выполнения инструкции: - неизвестная инструкция - неправильный невыравненный доступ - неправильное состояние при выполнении инструкции - ошибка при возврате из исключения - деление на ноль.
SVCall	SuperVisor Call – исключение, вызываемое инструкцией SVC. В среде операционных систем приложения м. использовать SVC для доступа к функциям ядра, драйверам и т. п.
PendSV	Request for system-level service (Запрос на службу системного уровня). В среде операционных систем, PendSV используется для переключения контекста, когда ни одно другое исключение не активно (состояние Active).
SysTick	Вызывается, когда системный таймер достигает 0. Приложения м. использовать это исключение. Или в среде операционных систем, процессор м. использовать это исключение, как системный тик.
Interrupt (IRQ)	Прерывание – исключение, генерируемое периферией или программой.

Приоритет исключений.
Lower priority number indicates a higher priority (Чем ниже номер приоритета – тем выше сам приоритет).

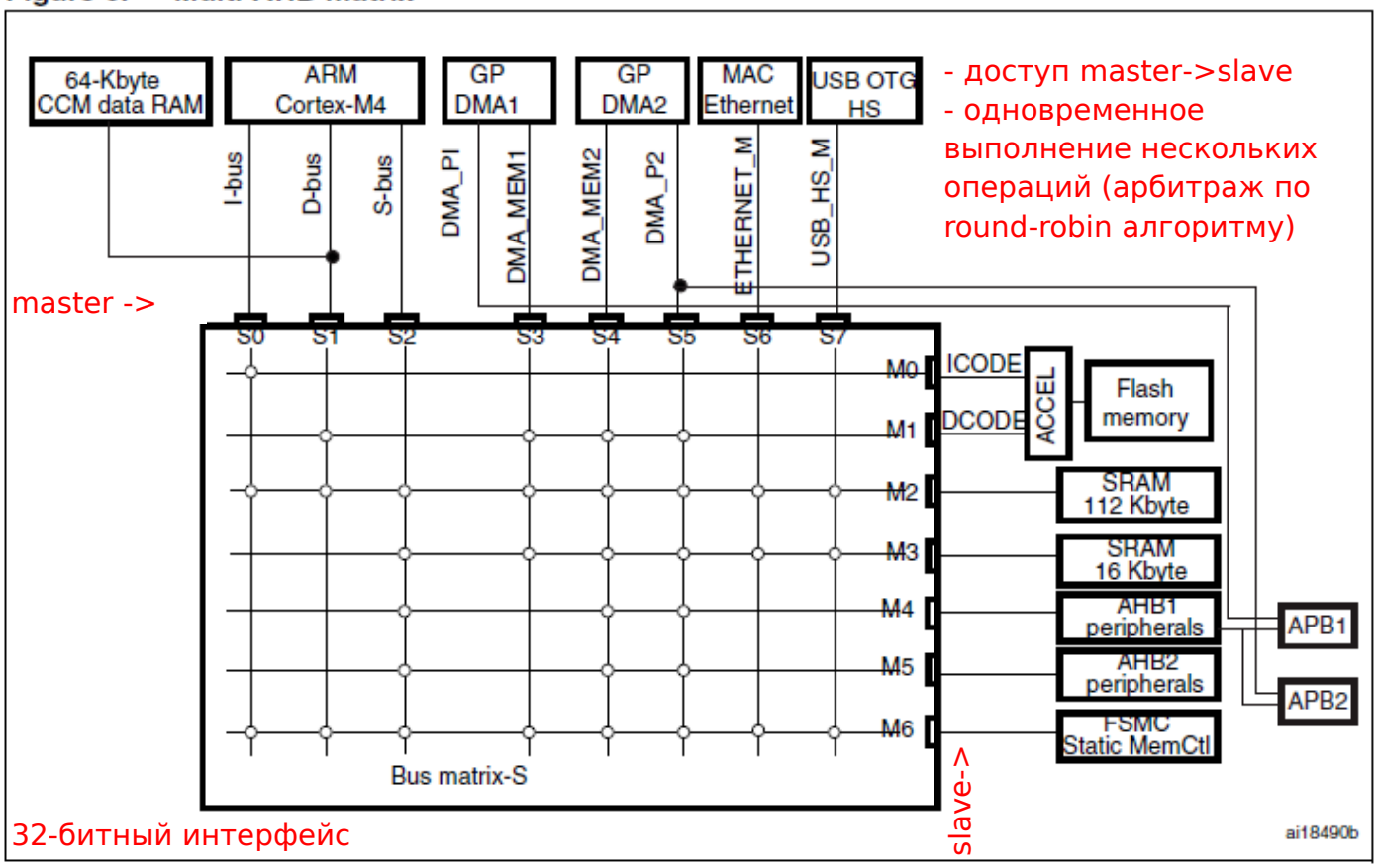
Приоритет программируется в пределах 0-15. По умолчанию приоритет равен 0. Приоритеты со знаком «-» фиксированы.

Если процессор обрабатывает исключение, то поступающее исключение с высшим приоритетом вытесняет его!

Если у двух одновременно ожидающих обработку исключений одинаковые приоритеты, первым на обработку идёт исключение с низшим порядковым номером.

4. MULTU-AHB matrix

Figure 6. Multi-AHB matrix

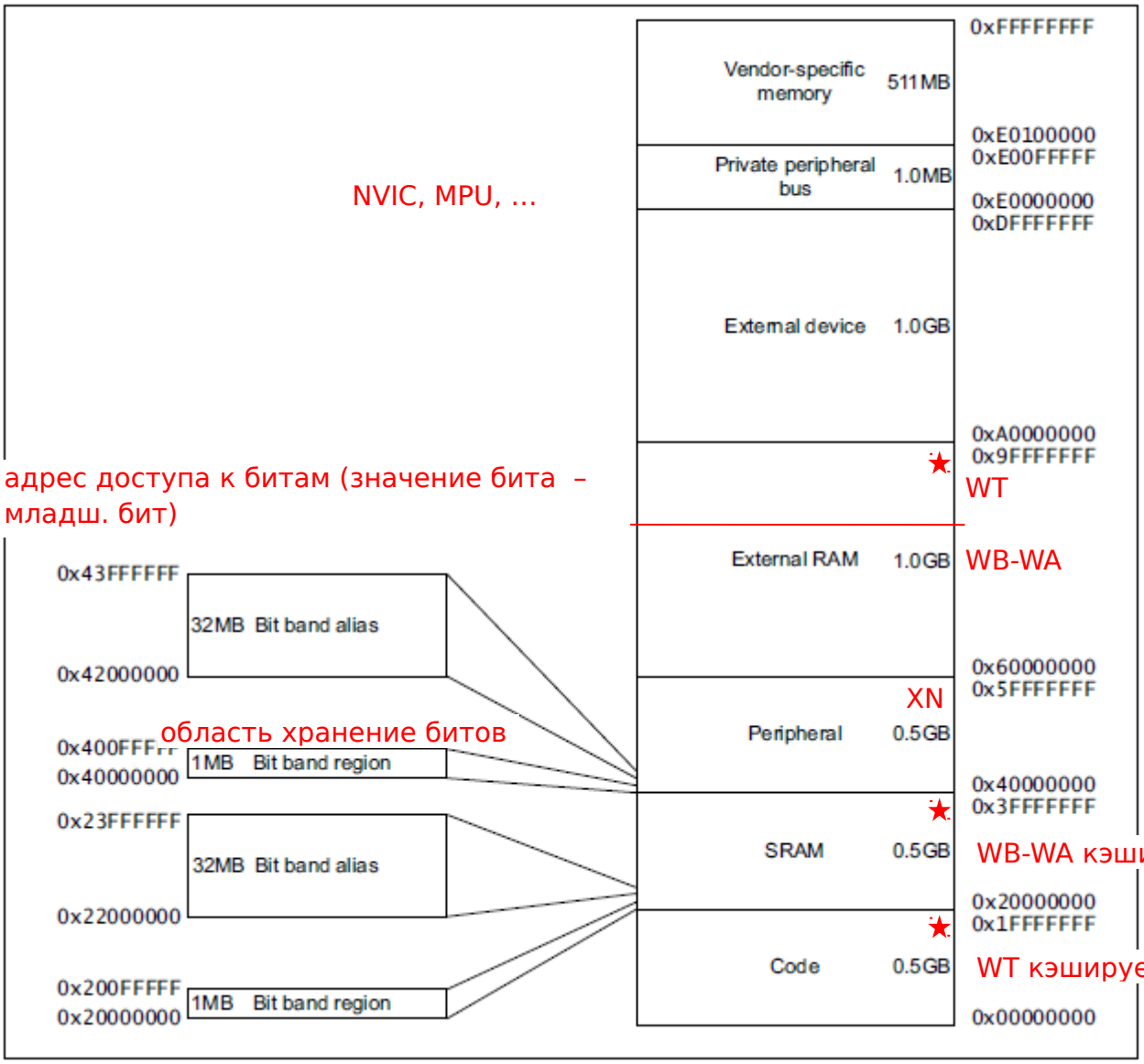


I-bus: Instruction bus - шина инструкций. Выборка инструкций из памяти Flash, SRAM, FSMC.
D-bus: Data bus – шина данных. Используется ядром для загрузки констант (literal load) и для отладки.
S-bus: System bus – системная шина. Используется для доступа к данным, расположенным в периферии или SRAM. Так же м. исп-ся для выборки инструкций, но менее эффективно, чем I-bus.

5. ПАМЯТЬ

5.1. КАРТА ПАМЯТИ ARM Cortex-M4

Figure 8. Memory map



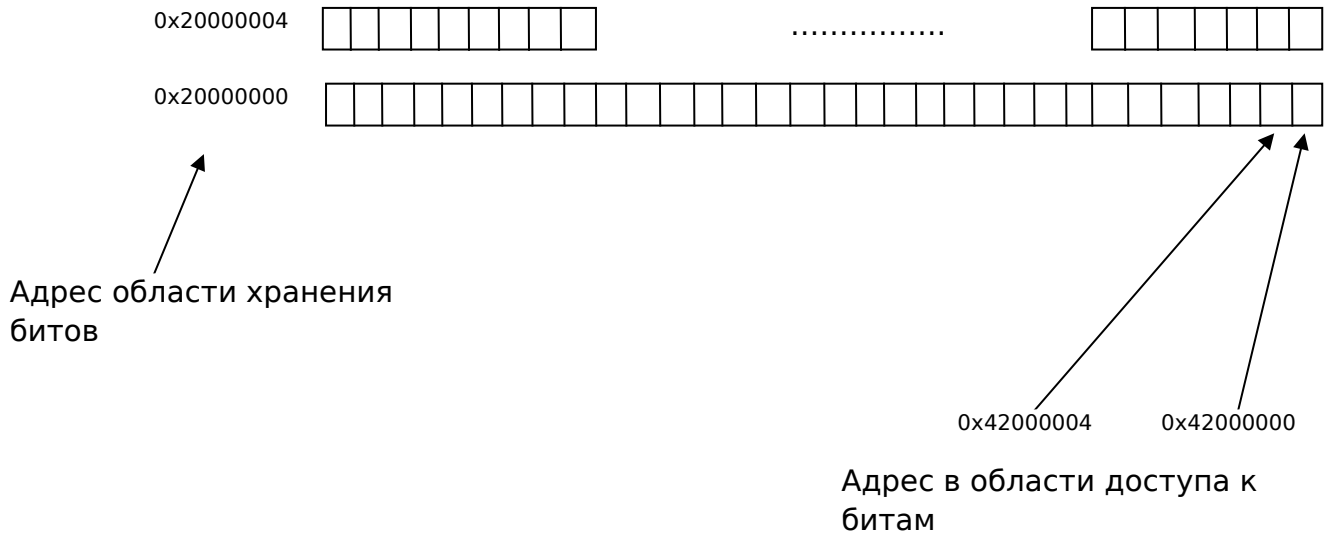
★ - программный код м. б. расположен здесь

XN - execute never

WT – Write through

WB-WA – write back-write a????

Сущность bit-band:

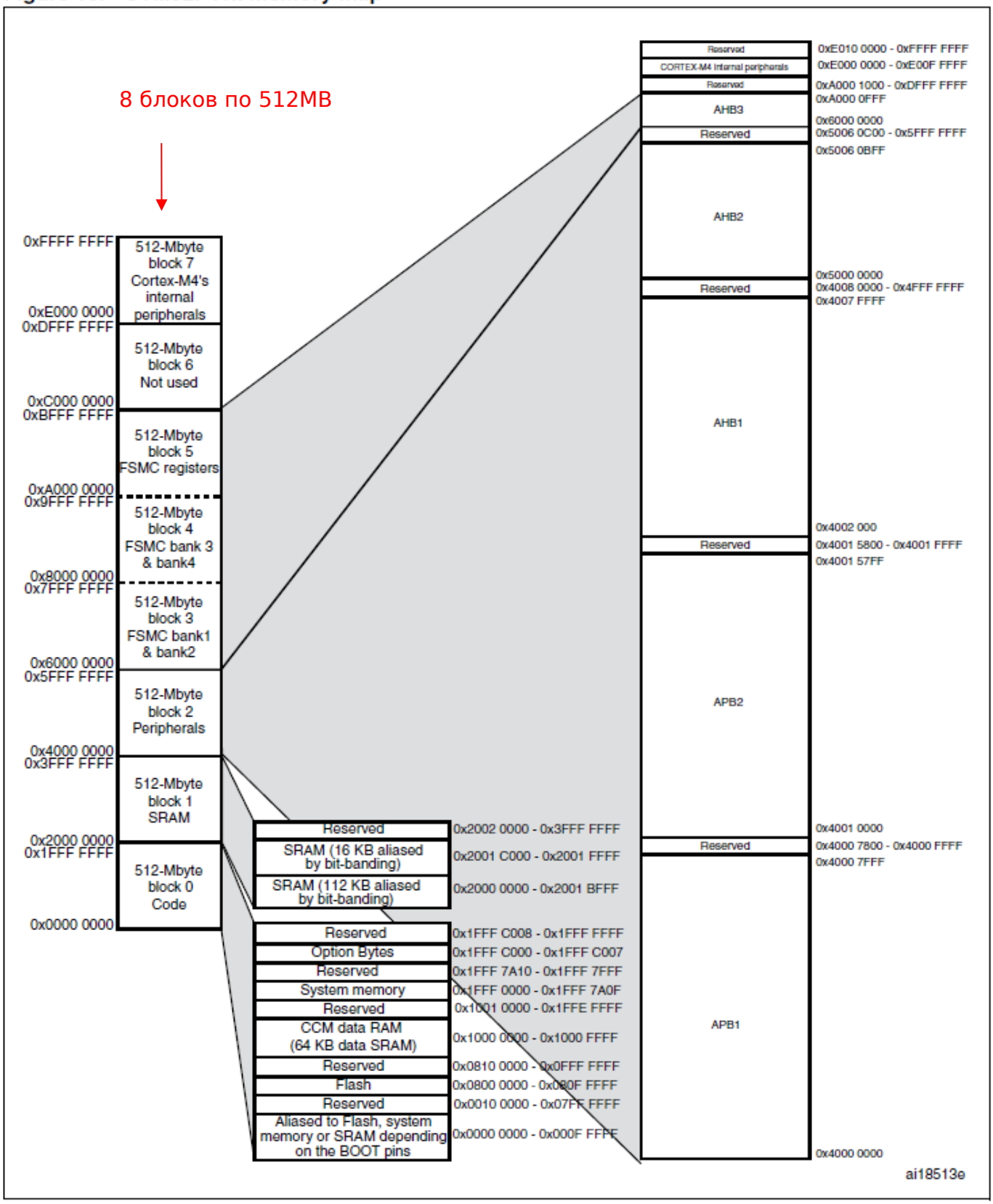


Преимущества Bit-band:

1. Ускоряет выполнение битовых операций (1 инструкция вместо 3)
2. Атомарность операции с битами

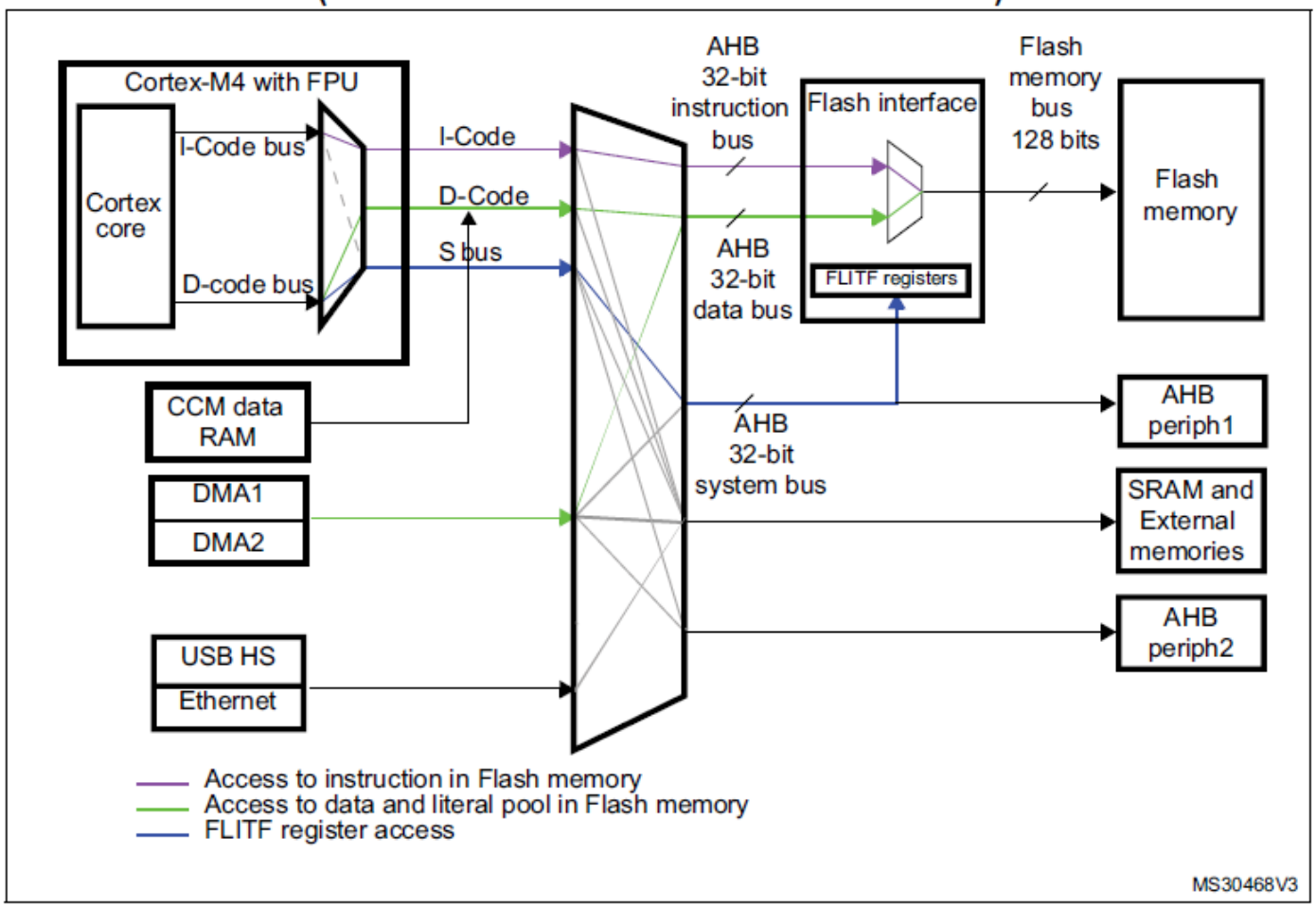
5.2. КАРТА ПАМЯТИ STM32F417

Figure 16. STM32F41x memory map



5.3. FLASH-ПАМЯТЬ

Figure 3. Flash memory interface connection inside system architecture (STM32F405xx/07xx and STM32F415xx/17xx)



Основные свойства:

1. Предвыборка 128 бит по шине I-Code (У Cortex-M4F 3-х стадийный конвейер).
2. 64 строки кэша по 128 бит по шине I-Code
3. 8 строк кэша по 128 бит по шине D-Code
4. Задержка на чтение из Flash при частоте ядра 168 MHz и напряжении 3,6V составляет 6 машинных циклов.

! После запуска МК, рекомендуется указать эту задержку (LATENCY) в регистре FLASH_ACR.

5. STM32F4 использует проприетарный ART flash accelerator. (Adaptive Real Time). Благодаря этому состояние ожидания выполнения программы сводится к 0 даже на частоте 168MHz.
6. RWW (read while write) операция не разрешена.

Организация Flash:

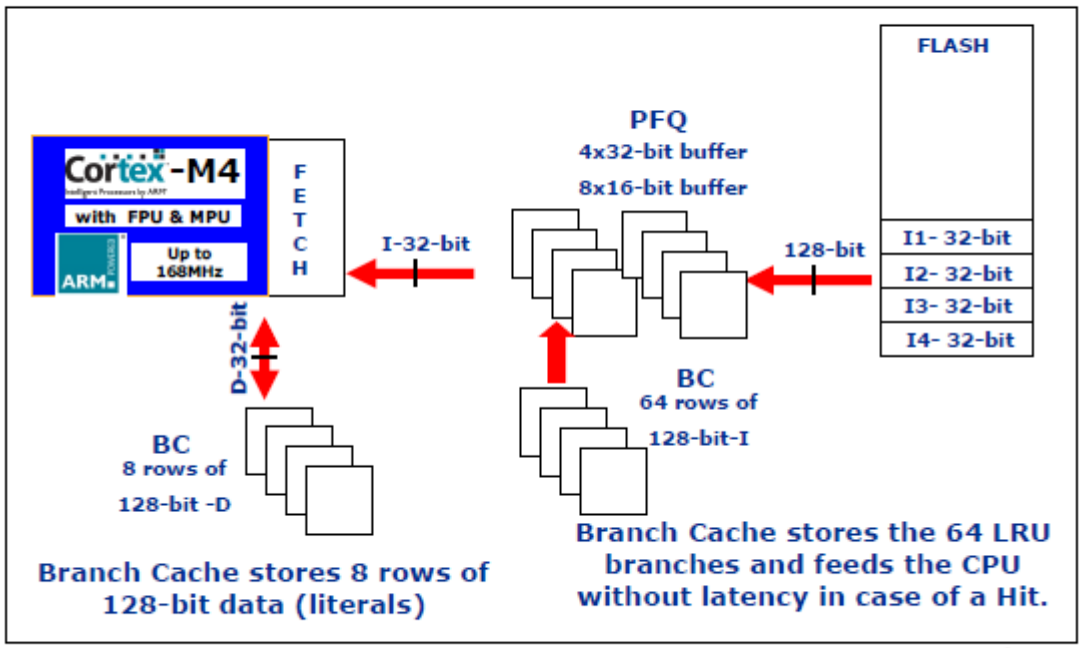
Table 6. Flash module organization (STM32F40x and STM32F41x)

Block	Name	Block base addresses	Size
Main memory	Sector 0	0x0800 0000 - 0x0800 3FFF	16 Kbytes
	Sector 1	0x0800 4000 - 0x0800 7FFF	16 Kbytes
	Sector 2	0x0800 8000 - 0x0800 BFFF	16 Kbytes
	Sector 3	0x0800 C000 - 0x0800 FFFF	16 Kbytes
	Sector 4	0x0801 0000 - 0x0801 FFFF	64 Kbytes
	Sector 5	0x0802 0000 - 0x0803 FFFF	128 Kbytes
	Sector 6	0x0804 0000 - 0x0805 FFFF	128 Kbytes
	. . .	. . .	. . .
	Sector 11	0x080E 0000 - 0x080F FFFF	128 Kbytes
System memory		0x1FFF 0000 - 0x1FFF 77FF	30 Kbytes
OTP area		0x1FFF 7800 - 0x1FFF 7A0F	528 bytes
Option bytes		0x1FFF C000 - 0x1FFF C00F	16 bytes

System memory: отсюда МК загружается в режиме system memory boot mode. Содержит встроенный BootLoader.
OTP area: one-time programmable область. Программируется один раз и не м. б. стерта в последствии. Используется для пользовательских данных. Например, сюда м. записать серийный номер изделия.
Option bytes: байты настройки. Read/write защита; BOR-level, watchdog, управление сбросом Reset, когда МК в режиме Standby или Stop.

ART accelerator:

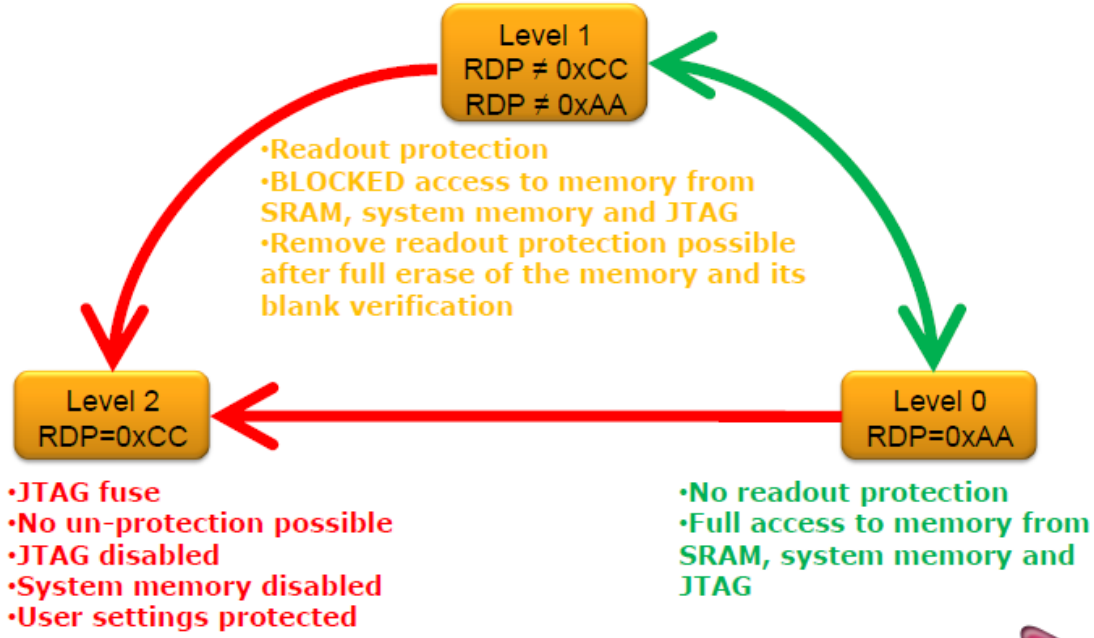
System Architecture – Role of the ART accelerator



STMicroelectronics

Защита Flash:

Flash Protections

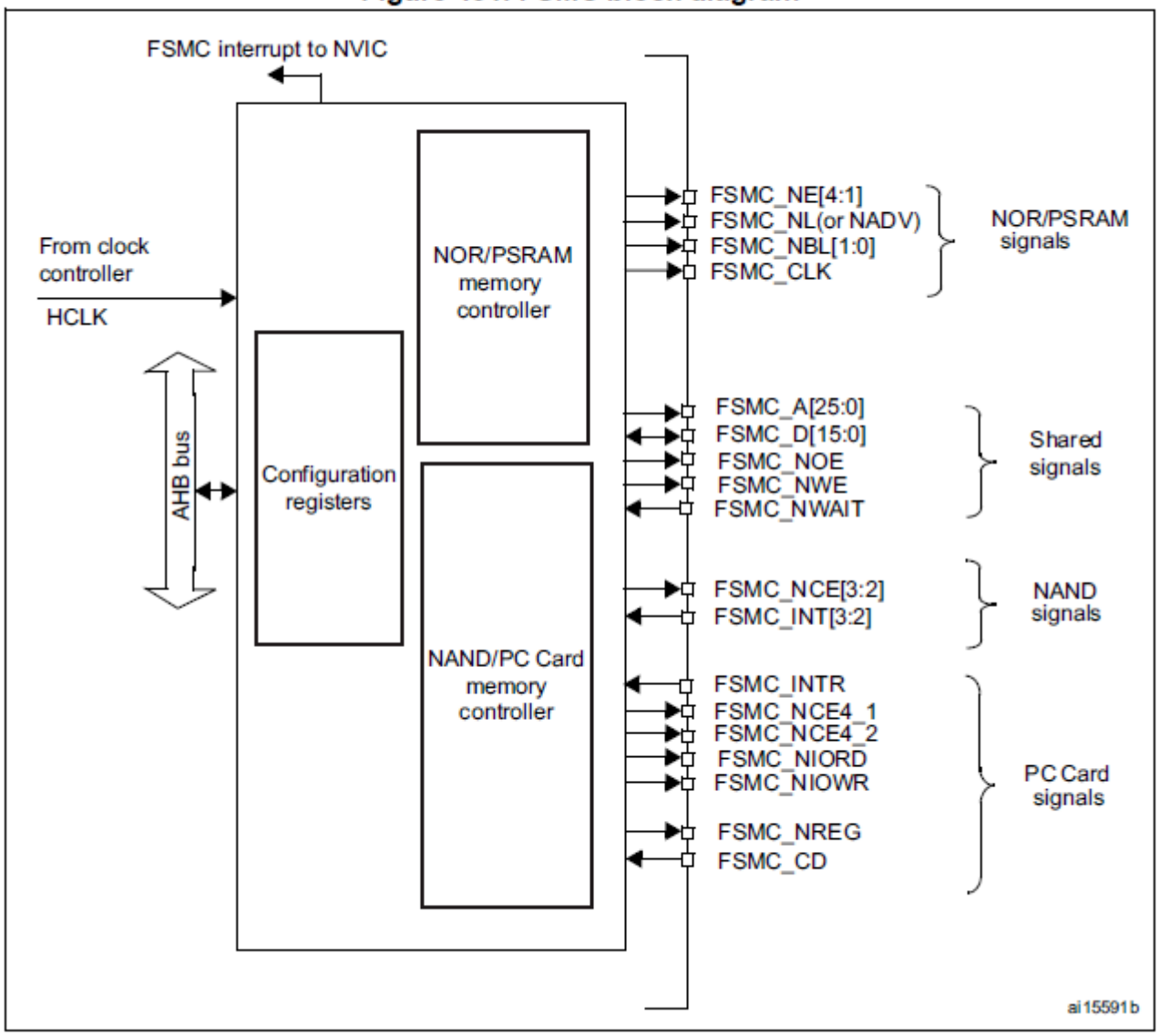


STMicroelectronics

Защита от несанкционированной вычитки кода или данных.

5.4. FSMC

Figure 431. FSMC block diagram



Назначение FSMC:

1. Трансляция AHB-транзакций в протокол внешнего устройства.
2. Удовлетворение timing requirements для внешнего устройства.

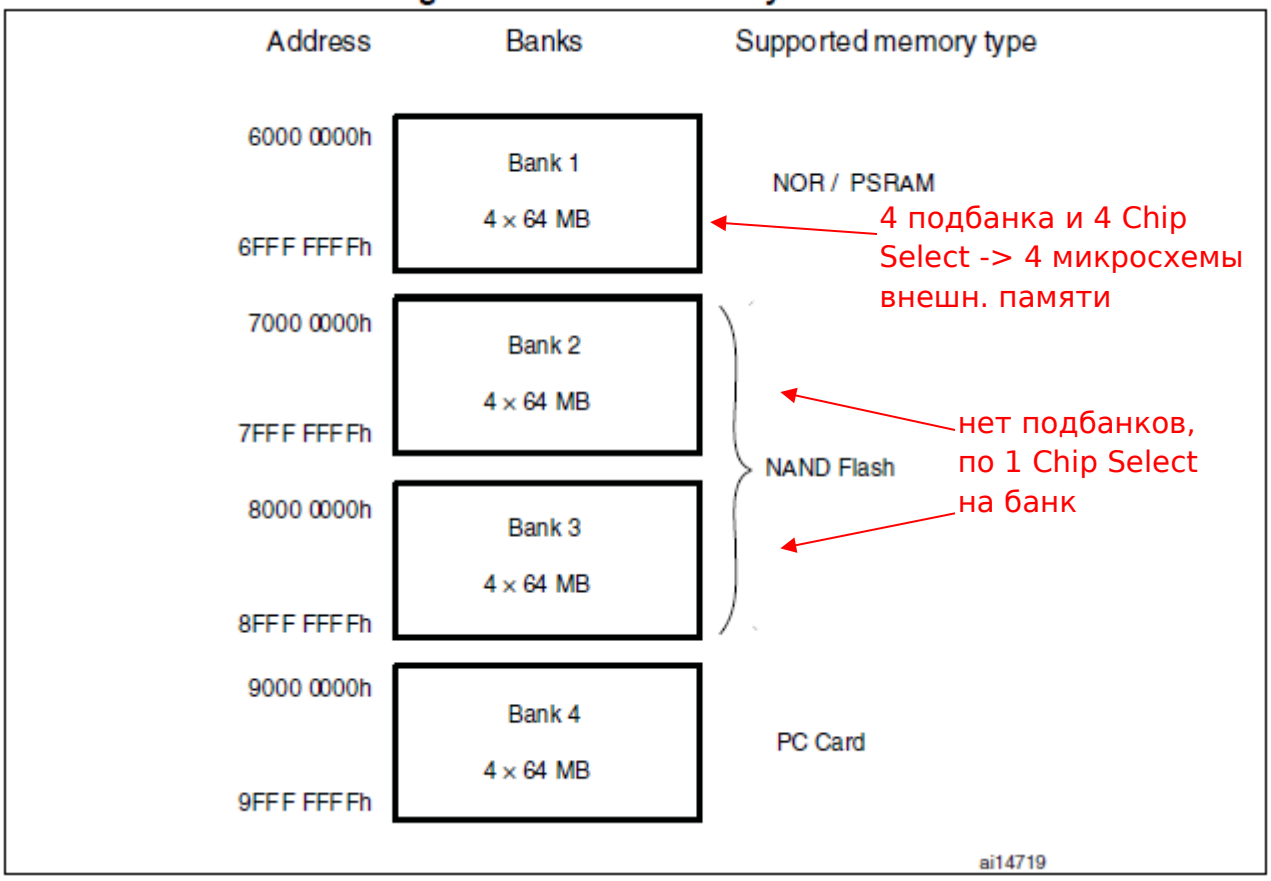
Особенности FSMC:

1. Взаимодействие с static memory-mapped devices (SRAM, NOR/NAND Flash, PSRAM).

1a. Взаимодействие в PC Card.

2. 8-бит или 16-бит шина данных.
3. Независимая конфигурация для каждого банка памяти.
4. Трансляция 32-битных AHB-транзакций, в общем случае, в последовательные 8-бит/16-бит данных для внешней памяти. Причем транзакция AHB 32 бита, а данные содержащиеся в ней м. б. 8/16/32 бита. Как они транслируются в 8/16 бит данных для внешней памяти – смотри reference_manual.
5. FIFO буфер записи на 2 слова (32-бит).

Figure 432. FSMC memory banks



PSRAM – pseudo SRAM. На самом деле это DRAM, имеющая встроенный модуль обновления ячеек и микросхему управления адресами, вместе позволяющие работать как будто это SRAM.

ToDo
Burst mode – пакетный режим обмена данными:
1. Linear burst mode
2. Wrap-around burst mode (STM32F417 не поддерживает wrap burst mode для синхронной памяти).

5.5. BOOT CONFIGURATION

По умолчанию:

Code area	0x00000000	Flash; I-code/D-code
Data area	0x20000000	SRAM; S-code
Reset vector 0x00000000	т. е. указывает на начало Flash	

Следовательно, нужен специальный механизм выбора места откуда загружаться.

Вариант 1.

Boot mode выбирается по BOOT[1:0] пинам:

x0	Flash-память
01	Bootloader, хранящийся в системной памяти
11	внутренняя SRAM

Boot-пины выбираются на 4-ом фронте SYSCLK после сброса.

BootLoader интерфейсы:
- USART1
- USART3
- CAN2
- USB OTG FS in Device mode through DFU (device firmware update)

Вариант 2.

Физическое перераспределение памяти (physical remap) в обход BOOT[1:0] пинов.

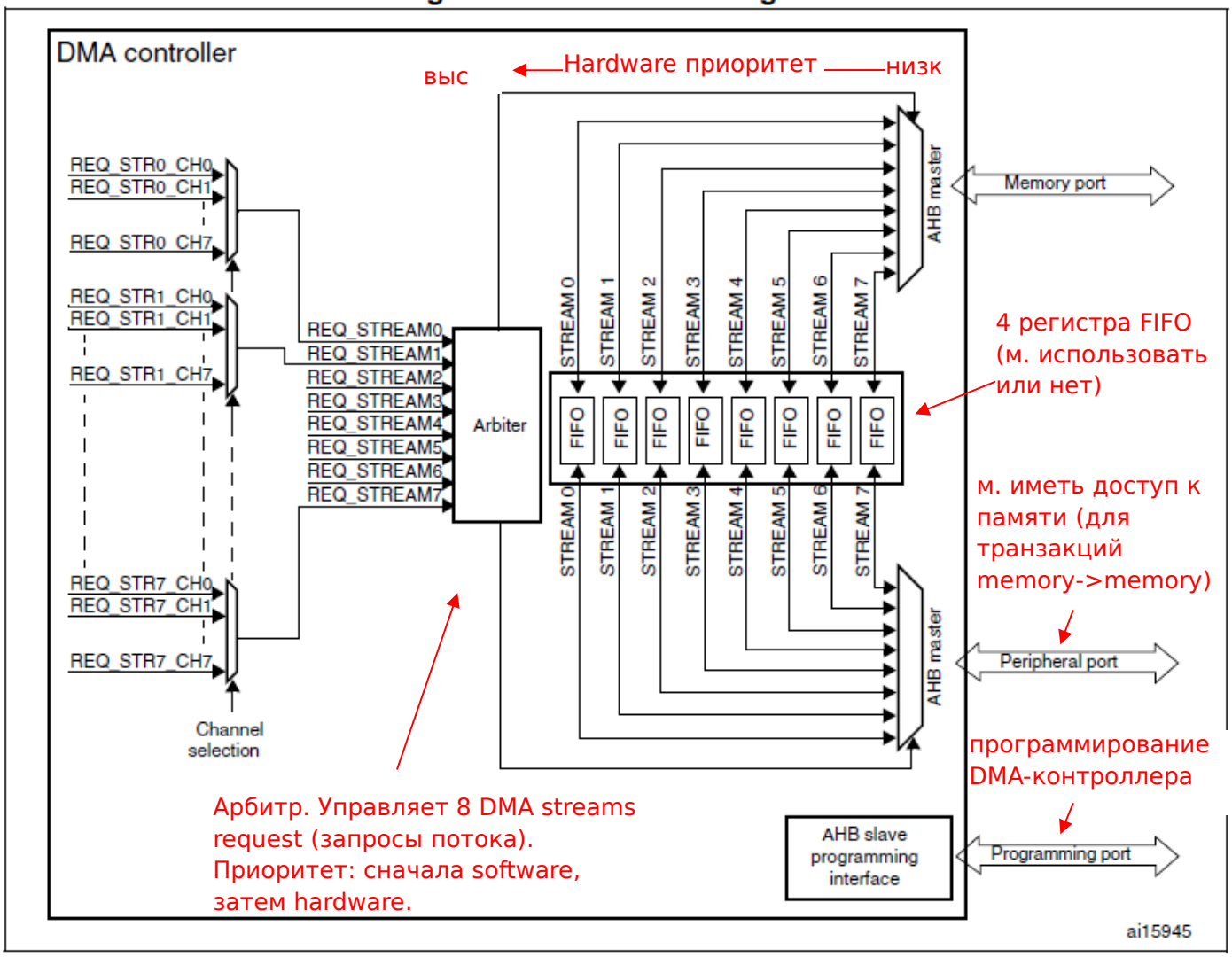
Значения SW в регистре SYSCFG_MEMRMP

SW:	00	11	01	10

	BOOT/REMAP in Main Flash memory	BOOT/REMAP in Embedded SRAM	BOOT/REMAP in System memory	REMAP in FSMC
0x2001 C000 - 0x2001 FFFF	SRAM2 (16kB)	SRAM2 (16kB)	SRAM2 (16kB)	SRAM2 (16kB)
0x2000 0000 - 0x2001 BFFF	SRAM1 (112kB)	SRAM1 (112kB)	SRAM1 (112kB)	SRAM1 (112kB)
0x1FFF 0000 - 0x1FFF 77FF	System memory	System memory	System memory	System memory
0x1000 0000 - 0x1000 FFFF	CCM Data RAM (64KB)	CCM Data RAM (64KB)	CCM Data RAM (64KB)	CCM Data RAM (64KB)
0x0810 0000 - 0x0FFF FFFF	Reserved	Reserved	Reserved	Reserved
0x0800 0000 - 0x080F FFFF	FLASH (1MB)	FLASH (1MB)	FLASH (1MB)	FLASH (1MB)
0x0010 0000 - 0x07FF FFFF	Reserved	Reserved	Reserved	FSMC NOR/SRAM 2 Bank1 (Aliased)
0x0000 0000 - 0x000F FFFF	FLASH (1MB) Aliased	SRAM1 (112kB) Aliased	System memory (30KB) Aliased	FSMC NOR/SRAM 1 Bank1 (Ali[illegible])

6. DMA-КОНТРОЛЛЕР

Figure 32. DMA block diagram



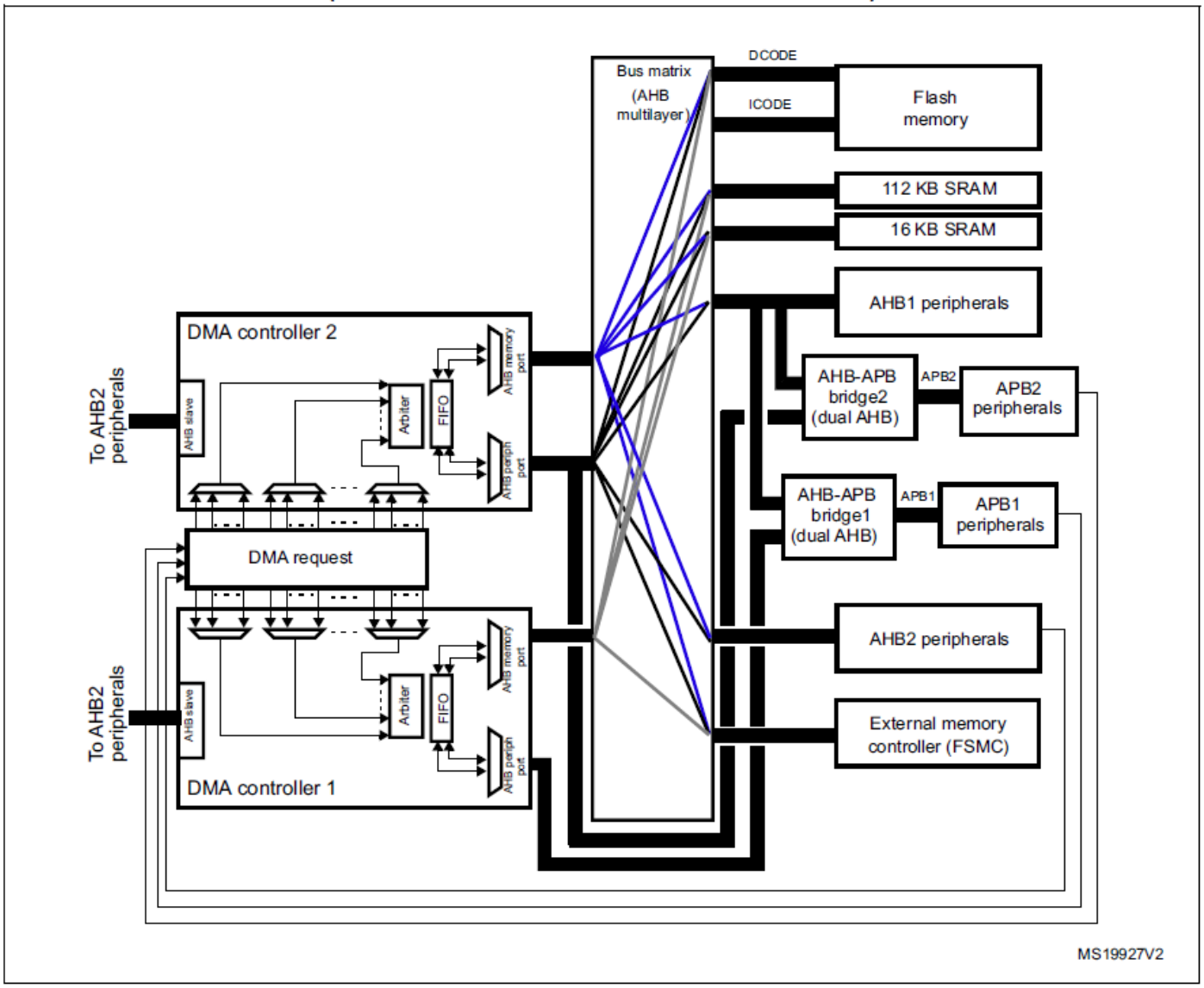
DMA-контроллер: 8 потоков, каждый по 8 каналов (запросов).

Типы DMA-транзакций:

1. Regular – обычные:
   - memory -> peripheral
   - peripheral -> memory
   - memory -> memory
2. Double-buffer.

Работает, как обычная транзакция, за исключением того, что 2 указателя адреса. Пока DMA работает с одной областью памяти (первый указатель), ПО м. работать с другой областью памяти (второй указатель).

Figure 33. System implementation of the two DMA controllers (STM32F405xx/07xx and STM32F415xx/17xx)



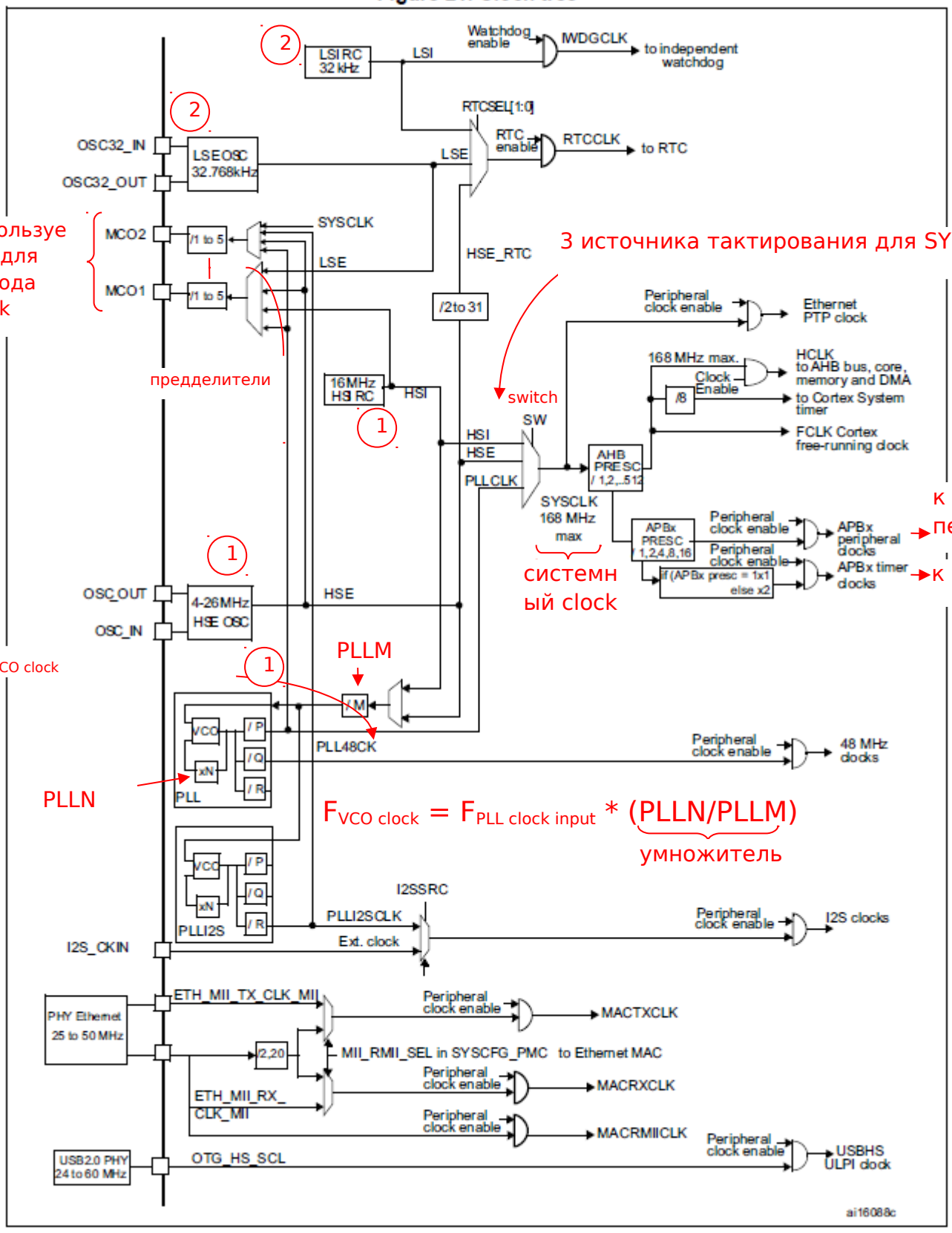
Каждый DMA-трансфер:
1. Загрузка из периф. регистра или памяти адреса памяти данных откуда читать.
2. Загрузка адреса назначения данных: периф. регистр или память.
3. Количество DMA-транзакций, которые необходимо выполнить.

Надежность DMA-транзакций:
1. Периферия отправляет request signal в DMA контроллер.
2. DMA-контроллер обрабатывает запрос на основе приоритета каналов.
3. Как только DMA-контроллер получил доступ к периферии, сигнал ACK отправляется периферии.
4. Периферия отменяет сигнал request signal, как только получила сигнал ACK от DMA.

5. После отмены request signal от периферии, DMA-контроллер отменяет сигнал ACK.
6. Если есть еще запросы периферия м. выставить сигнал request signal. И далее с п. 1.

7. ТАКТИРОВАНИЕ

Figure 21. Clock tree



1 - первичные источники тактирования (используется для SYSCLK)

2 - вторичные источники тактирования (используется для watchdog, RTC)

LSE, LSI – Low Speed External/Internal

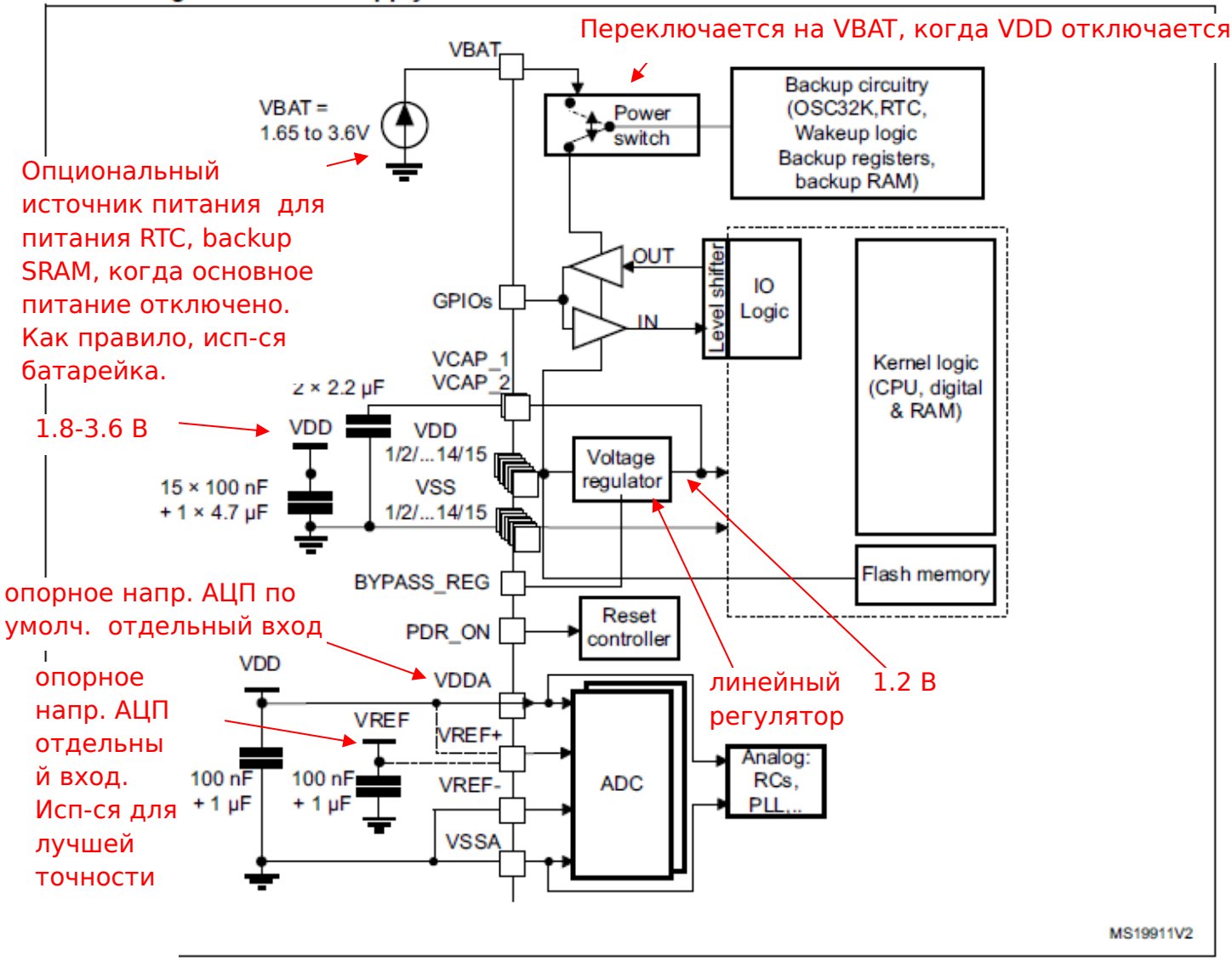
HSE, HSI – High Speed External/Internal

SYSCLK:
1. После reset: HSI clock выбирается для SYSCLK.
2. Переключение с одного источника тактирования на другой возможно, только если целевой clock готов (стабилен). Регистр RCC_CR показывает какой clock в данный момент выбран для SYSCLK и какие clock’и готовы.

CSS – Clock Security System (активируется программным обеспечением): Если HSE clock дает сбой, то этот осциллятор автоматически отключается и генерируется NMI.

8. PWR.

Figure 9. Power supply overview for STM32F405xx/07xx and STM32F415xx/17xx



1. V_{DDA} and V_{SSA} must be connected to V_{DD} and V_{SS}, respectively.

http://encyclobeamia.solarbotics.net/articles/vxx.html

BJT	FET	"Vxx" meaning
Vcc	Vdd	Positive supply voltage
Vee	Vss	Negative supply, ground

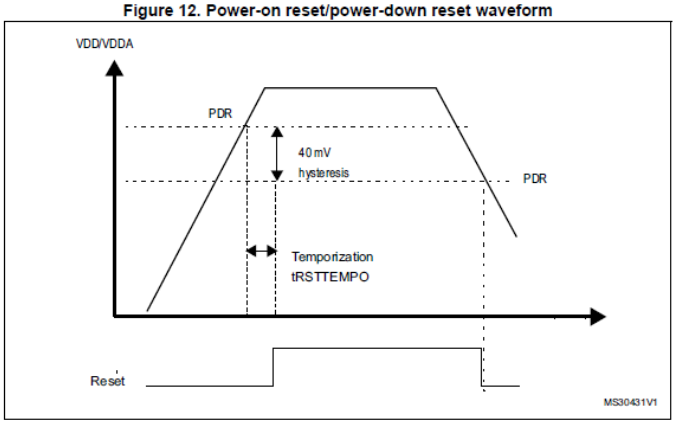
BJT – Bipolar Junction Transistor
FET – Field Effect Transistor

Battery Backup Domain – бэкап домен микроконтроллера (RTC, 4KB SRAM), работающий от питания батарейки VBAT, когда основное питание VDD отключено.

Режимы работы МК и Voltage Regulator:
- Run mode. Полное снабжение энергией микроконтроллера. МК м. потреблять меньше, когда работает на частоте меньше 168MHz.
- Stop mode. Полное снабжение энергией микроконтроллера. М. б. переведен в режим пониженного энергопотребления.
- StandBy mode. Voltage Regulator отключен.

Супервизор питания STM32F417:
1. POR/PDR – Power On Reset / Power Down Reset. Встроенная микросхема – запуск МК, если VDD > 1.8В. Если меньше – МК в состоянии reset.

Figure 12. Power-on reset/power-down reset waveform



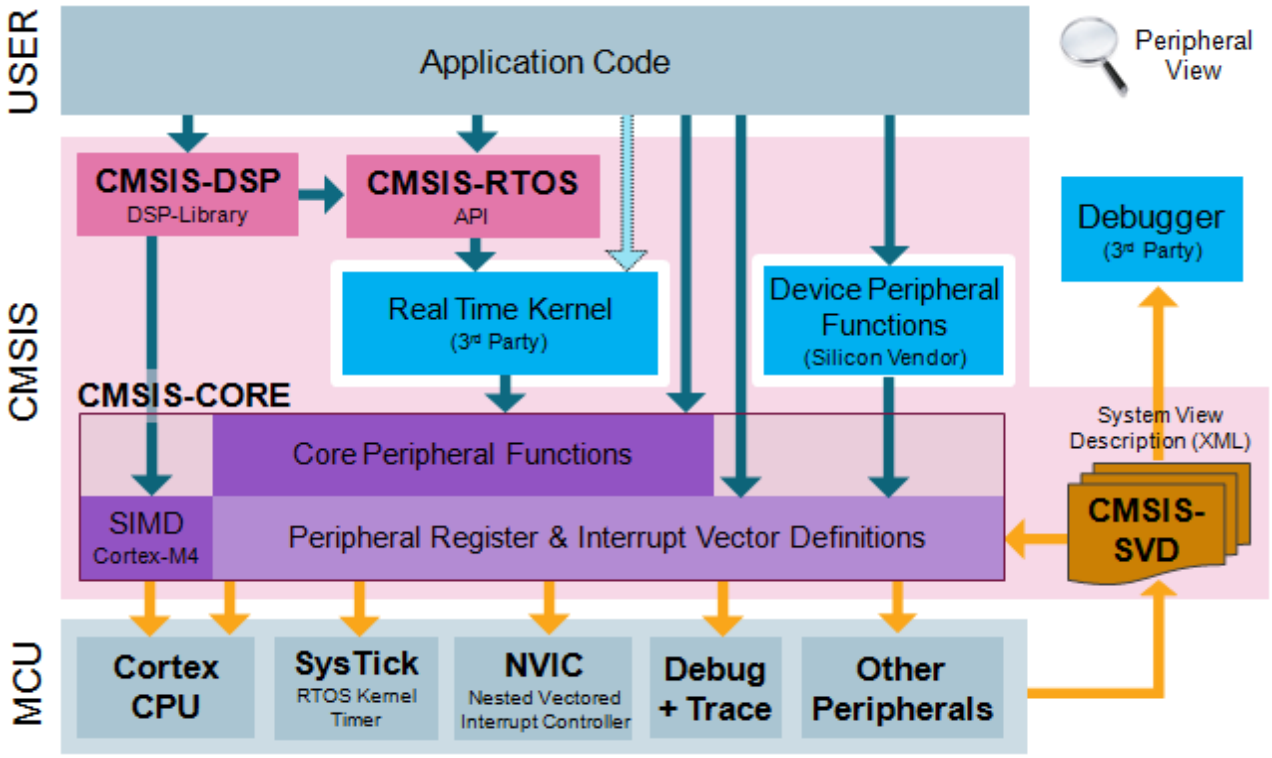
2. BOR – Brownout Reset. Если в процессе работы МК, уровень VDD опустится ниже уровня BOR, генерируется прерывание reset. Brownout reset м. отключить в option bytes.

9. TODO

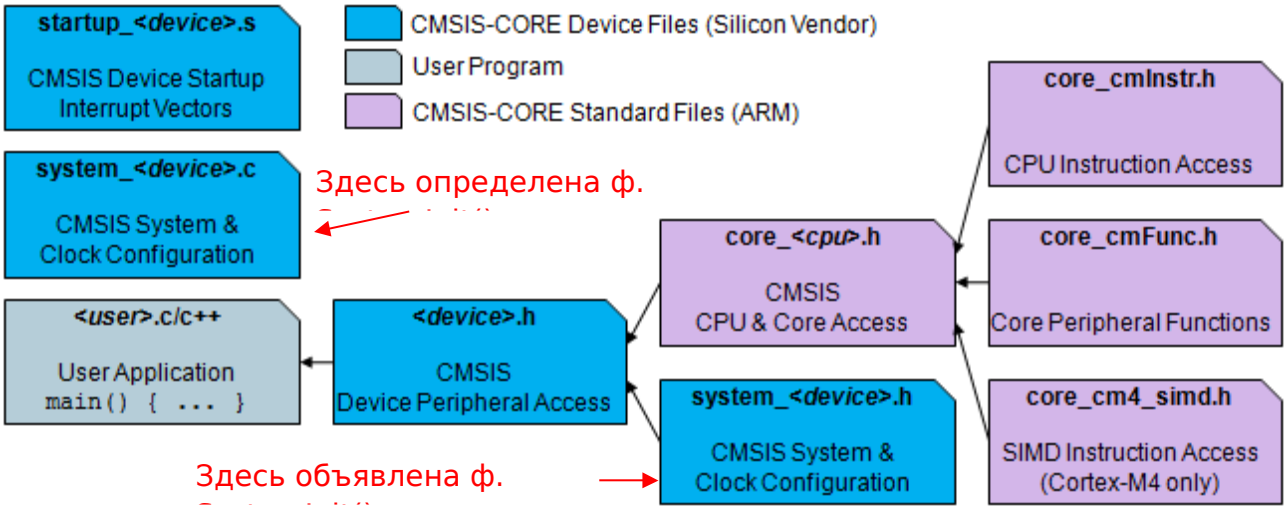
10. CMSIS И ПРОГРАММИРОВАНИЕ

10.1. CMSIS

CMSIS – Cortex Microcontroller software interface standard - определения регистров, функций, векторов прерываний. С-подобная надстройка над уровнем ассемблера, созданная для унификации переносимости между различными Cortex'ами и упрощения разработки.

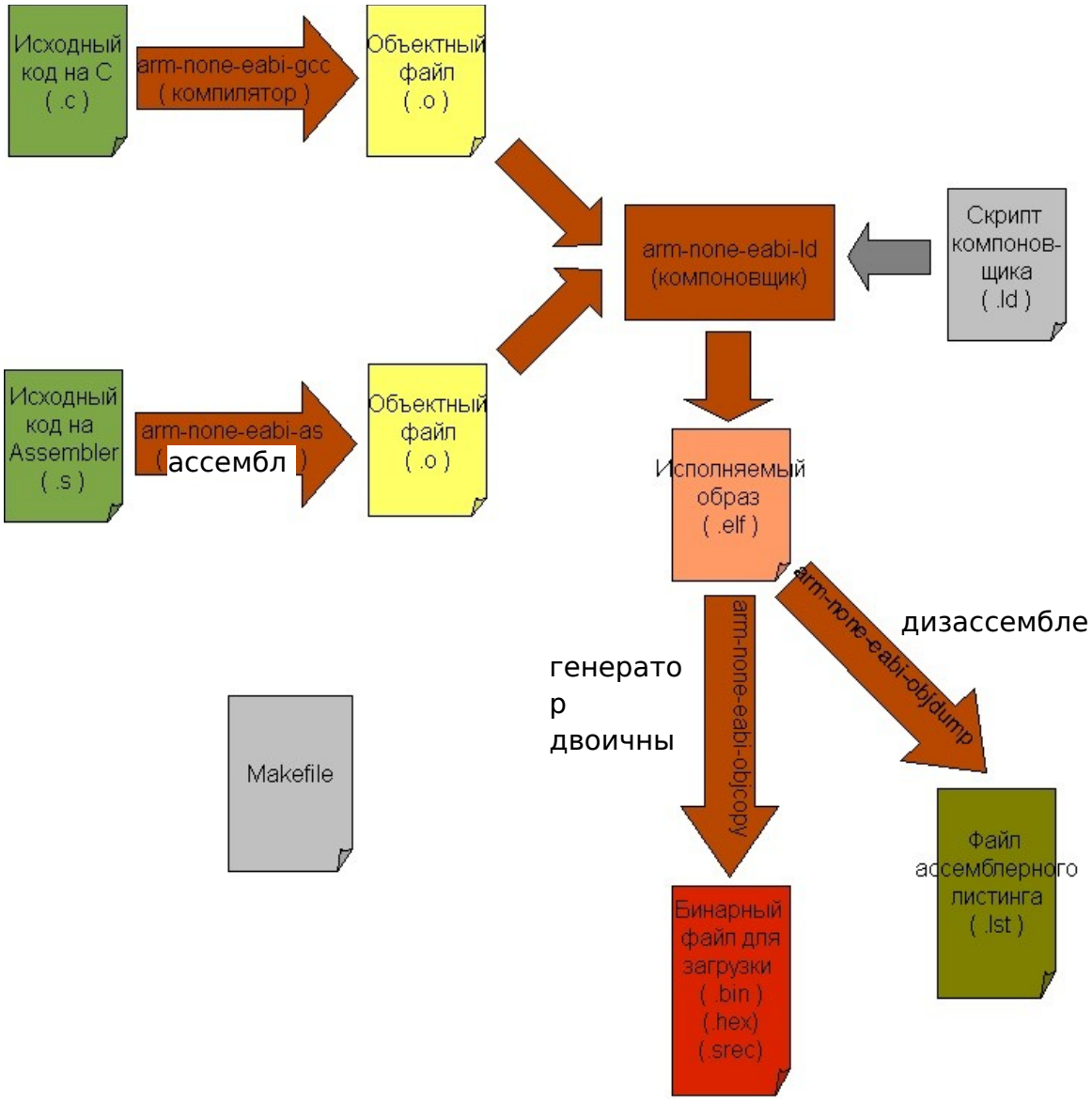


Структура файлов CMSIS-CORE:



Эти файлы д. б. в <u>любом</u> проекте для STM32.

10.2. Процесс разработки.



EABI – Embedded Application Binary Interface. Набор соглашений для форматов файлов, типов данных, использования регистров,

организации стека и передачи параметров функций во встроенном ПО.

Компиляторы, поддерживающие EABI м. генерировать код, совместимы друг с другом (возможность линковки).

10.3. НАСТРОЙКА СРЕДЫ РАЗРАБОТКИ ECLIPSE.

Состав среды разработки:
- Eclipse
- GNU Tools ARM Embedded
- Eclipse ARM plug-in
- Embedded systems register view plug-in

11. GNU TOOLCHAIN

11.1. GNU AS

GNU as – семейство ассемблеров для каждой конкретной архитектуры. Каждая версия имеет много общего с другими: формат объектных файлов, ассемблерные директивы (псевдооператоры) и синтаксис.

GNU as – однопроходной ассемблер. Препроцессор не выполняет обработку макросов и включаемых файлов.

Символы – набор знаков, букв и символов: ‘_.$’. Символ не м. начинаться с цифры. Регистр имеет значение.

Оператор – оканчивается на символ новой строки ‘\n’ или на символ разделителя строк ‘;’. Последний знак входного файла д. б. символ новой строки ‘\n’.

http://www.opennet.ru/docs/RUS/gas/gas-4.html

Сущность	**Описание**
.section .weak .type Reset_Handler	Символы, начинающиеся с ‘.’ – директива ассемблера (псевдооператор)
ldr r3,[r3,r1] str r3,[r0,r1]	Символы, начинающиеся с буквы – инструкция ассемблера для конкретной архитектуры
adds r1,r1,#4	#4 – прямой операнд (непосредственная адресация). М. использовать #4, $4, 4
.byte 0x4A .ascii “Ring the bell” .octa 0x12345678	Константы и переменные

.float 0f-314.0E-2 .word _data	
/* comment1 comment2 */ @1-line comment for as	Комментарии
.section	

11.2. GNU GCC

11.3. GNU LD

Линковка – процесс выдирания секций из объектных файлов, раскладывание их по указанным адресам и исправление перекрестных ссылок.

Секция	**Описание**
.text	скомпилированный машинный код
.data	глобальные и статические переменные (при старте МК копируются из Flash в RAM)
.rodata	глобальные и статические константы (при старте МК остаются во Flash)
.bss	глобальные и статические переменные, которые при старте содержат нулевые значения (шуточное расшифровка better save space). Не надо выделять под них Flash-память в целях ее экономии (бессмысленно хранить нули), но при старте программы место в RAM выделять надо. Размер секции .bss хранится в объектном файле.
.comment	инфо о версии компилятора
.ARM.attributes	ARM-специфичные атрибуты

Секция .data – данные физически хранятся во Flash, а работать предстоит из RAM => необходим загрузочный код, который будет копировать переменные секции .data в RAM. Это делает стартовый код на ассемблере.

У секции 2 адреса:

LMA – Load Memory Address	**VMA** – Virtual Memory Address
Откуда загружается секция, т. е. где она окажется в бинарном файле	Куда будут перенаправлены символы секции, т. е. указатель на символ в коде будет ссылаться на VMA-адрес.

http://www.opennet.ru/docs/RUS/gnu_ld/gnuld-3.html

Команда	**Описание, пример**
ENTRY	Точка входа в программу ENTRY(Reset_Handler) Точку входа м. указать (в порядке уменьшения приоритета): 1. Опция к.с. ‘-e’ 2. Команда ENTRY(<символ>) в скрипте линкера 3. Значение символа старт 4. Адрес первого байта в секции ‘.text’

MEMORY	Описывает доступную память в целевой архитектуре. По умолчанию доступна вся память.
	
	MEMORY
	{
	FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 1M
	RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 128K
	CCMRAM (xrw) : ORIGIN = 0x10000000, LENGTH = 64K
	}
	
	- FLASH, RAM, CCMRAM – произвольные имена, используемые внутри скрипта линкера для ссылки на регионы.
	- (xrw) и т. п. – необязательный список атрибутов LIRWX, использующийся для совместимости с линкером AT&T. GNU ld его не использует.
	- ORIGIN – нач. адрес региона физич. памяти.
	- LENGTH – размер региона физич. памяти.

SECTIONS	Определяет «карту» выходного файла. Распределение секций по LMA. SECTIONS { .isr_vector : { . = ALIGN(4); KEEP(*(.isr_vector)) /* Startup code */ . = ALIGN(4); } >FLASH

	.text : { . = ALIGN(4); *(.text) /* .text sections (code) */ *(.text*) /* .text* sections (code) */ *(.rodata) /* .rodata sections (constants, strings, etc.) */ *(.rodata*) /* .rodata* sections (constants, strings, etc.) */ . = ALIGN(4); } >FLASH .denis : { *(.some_section) } >RAM .denis – произвольное имя секции, которая после линкера будет в выходном файле .elf. Аналогично имя .text и .isr_vector, стоящие до фигурных скобок {}. Имена секций, стоящих внутри фигурных скобок {} – стандартные и генерируются ассемблером и компилятором. Например, для функции main() будет сгенерирована секция .text.main. И чтобы включить ее в выходную секцию .text стоит выражение *(.text*). Аналогично для переменных.
>FLASH	Ключевое слово сообщает, что выходная секция д. б. помещена в регион FLASH, описанный ранее в команде MEMORY
.	Счетчик позиции. Всегда содержит текущую позицию вывода – адрес LMA. Счетчик позиций автоинкрементируется и никогда не должен уменьшаться.
.text .bss .rodata	Пример имен секций. В данном случае ‘.’ относится к имени секции.
ALIGN(4)‘*’ - ссылка на файл**ы** в к. с. ld. В скобках – имя секции в этих файл**ах** (они известны из объектных файлов с расширением .o) COMMON – указывает на все неинициализированные данные из	Функция ALIGN возвращает значение счетчика позиций, выровненное на границу следующих 4 байт. Функция ALIGN сама не изменяет счетчик позиций. Чтобы это сделать, нужно сделать явное присвоение возвращаемое значение счетчику позиций. . = ALIGN(4);

всех входных файлов *(.text) *(COMMON)	
KEEP(*(.isr_vector))	KEEP – инструкция «оставить». Не дает линкеру оптимизировать «за ненадобностью». В данном примере – не дает оптимизировать все секции .isr_vector
PROVIDE(etext = .)	Инструкция используется для определения символа, если он не используется в программе и если он не определен другим объектом, включенным в линковку. В этом случае, если программа определяет etext, то будет использовано определение etext в программе. Если программа ссылается на etext, но не определяет etext, то будет использована etext, определенная в скрипте линковщинка.
PROVIDE_HIDDEN(etext = .)	Аналогично PROVIDE для ELF (executable linkable file)

12. STEMWIN LIBRARY

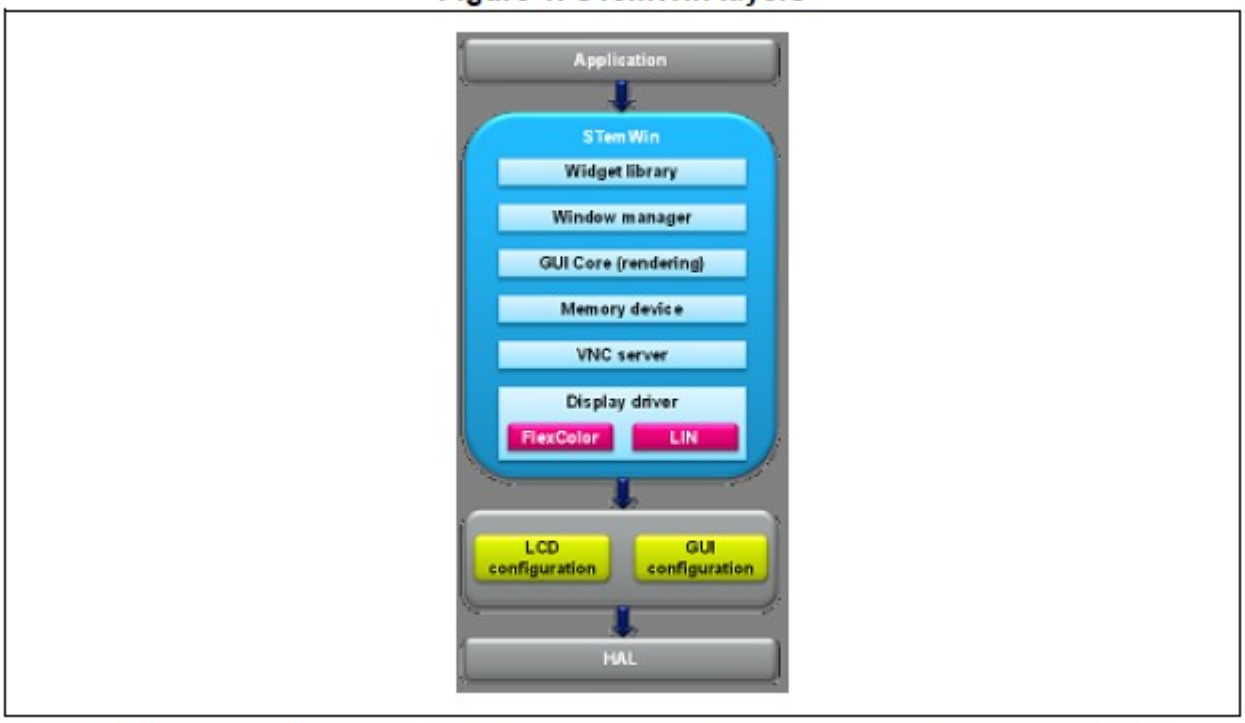
Процесс инициализации emWin:

GUI_Init()	
----GUI_X_Config()	
----GUI_ALLOC_AssignMemory()	Выделение памяти для STemWin в куче
----GUI_DEVICE_CreateAndLink()Создание display driver и выбор color conversion. Установка display size, конфигурация touchscreen. ----LCD_X_Config()	
----LCD_SetVSizeEx()	
----LCD_SetSizeEx()	
----LCD_SetVRAMAddrEx()	
----GUI_TOUCH_SetOrientation() ----GUI_TOUCH_Calibrate()	
----LCD_X_DisplayDriver()	display driver вызывает эту функцию. Запуск операций display controller. + доп. функции: сглаживание, вирт. экраны и т. д.

Файлы run-time конфигурации:

GUIConf.c	конфигурирование включаемых модулей (memory device, window manager, ...) и доступной памяти (куча - heap)
LCDConf.c	конфигурирование размера дисплея, display driver и color conversion routines
GUI_X.c	конфигурирование timing routines. Создание таймера OS_TimeMS с периодом 1мс. для нужд STemWin.

Figure 1. STemWin layers

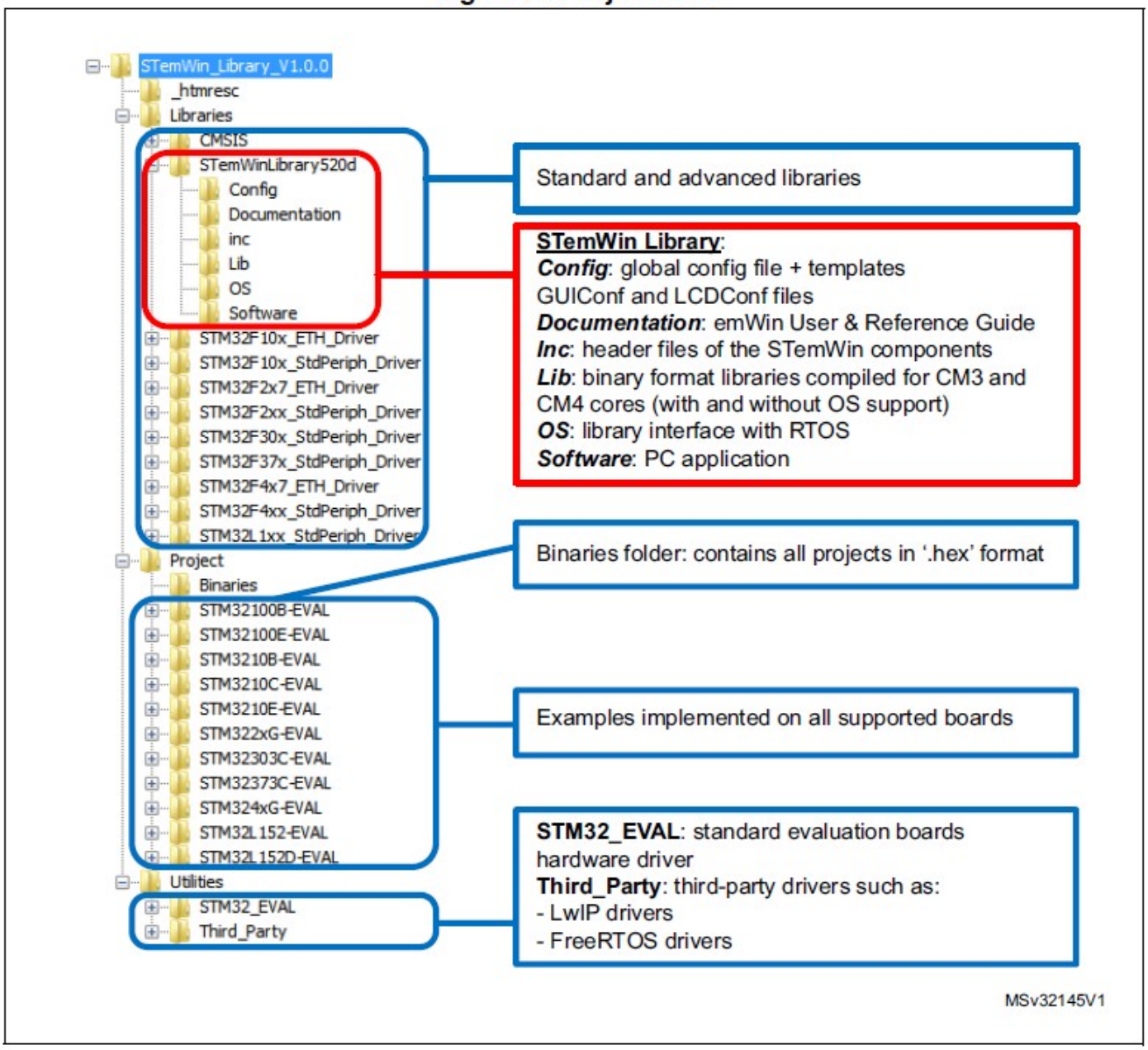


1. The CRC module (in RCC peripheral clock enable register) should be enabled before using the library.

Два display driver в STemWin:

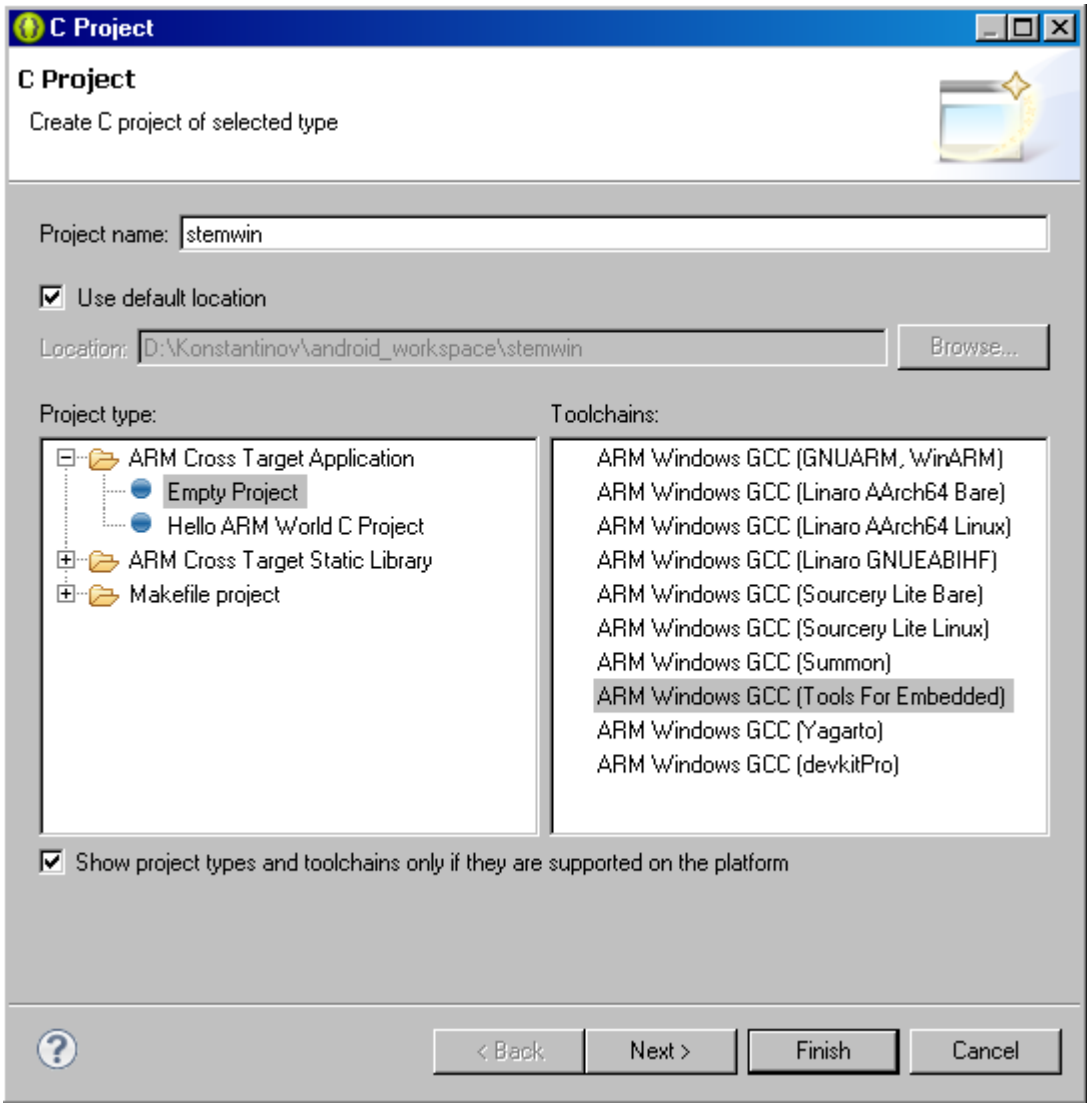
LIN	Direct Linear access. Для внутреннего TFT-LCD контроллера. Например, в STM32F429.
FlexColor	Indirect access. Для последовательной и параллельной шины. Для внешнего LCD-контроллера. Например, контроллер STM32F417 и внешний LCD-контроллер Solomon SSD1963.

Figure 2. Project tree



12.1. Настройка и компиляция проекта в Eclipse, использующего STemWin Library, для демо-платы STM3241G_EVAL.

1) Создать новый проект с именем stemwin. File -> New -> C Project.



2) Нажать Finish.

3) Пока проект выглядит так.

В «Includes» содержаться пути к включаемым каталогам GNU Tools ARM Embedded.

4) Создать в проекте каталог CMSIS.
Поместить в каталог CMSIS 9 файлов:

core_cm4_simd.h
core_cm4.h
core_cmInstr.h core_cmFunc.h
startup_stm32f4xx.S

Расширение .S обязательно с большой буквы! Иначе компилятор не поймет.

stm32f4xx_conf.h
stm32f4xx.h
system_stm32f4xx.c
system_stm32f4xx.h

stm32f4xx_conf.h на самом деле не принадлежит CMSIS-CORE, но д. б. в этом каталоге т. к. на него ссылается stm32f4xx.h, если определена USE_STD_PERIPH_DRIVER.
В stm32f4xx_conf.h – включения всех стандартных заголовочных файлов периферии.

Все файлы м. взять, например, из STM32F4xx_DSP_StdPeriph_Lib_V1.1.0. Свободно загружается с официального сайта st.com

5) Создать в проекте каталог StdPeripheralDriver.
Поместить во вновь созданный каталог StdPeripheralDriver каталоги inc и src из каталога STM32F4xx_StdPeriph_Driver – стандартные драйверы периферии микроконтроллера. М. взять, например, из STM32F4xx_DSP_StdPeriph_Lib_V1.1.0. Свободно загружается с официального сайта st.com.

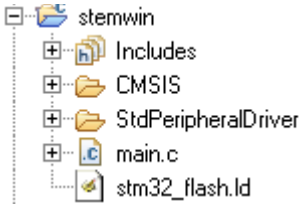
inc	заголовочные файлы драйверов
src	исходные файлы драйверов

6) Добавить к проекту новый С-файл main.c
Содержание файла main.c минималистичное:

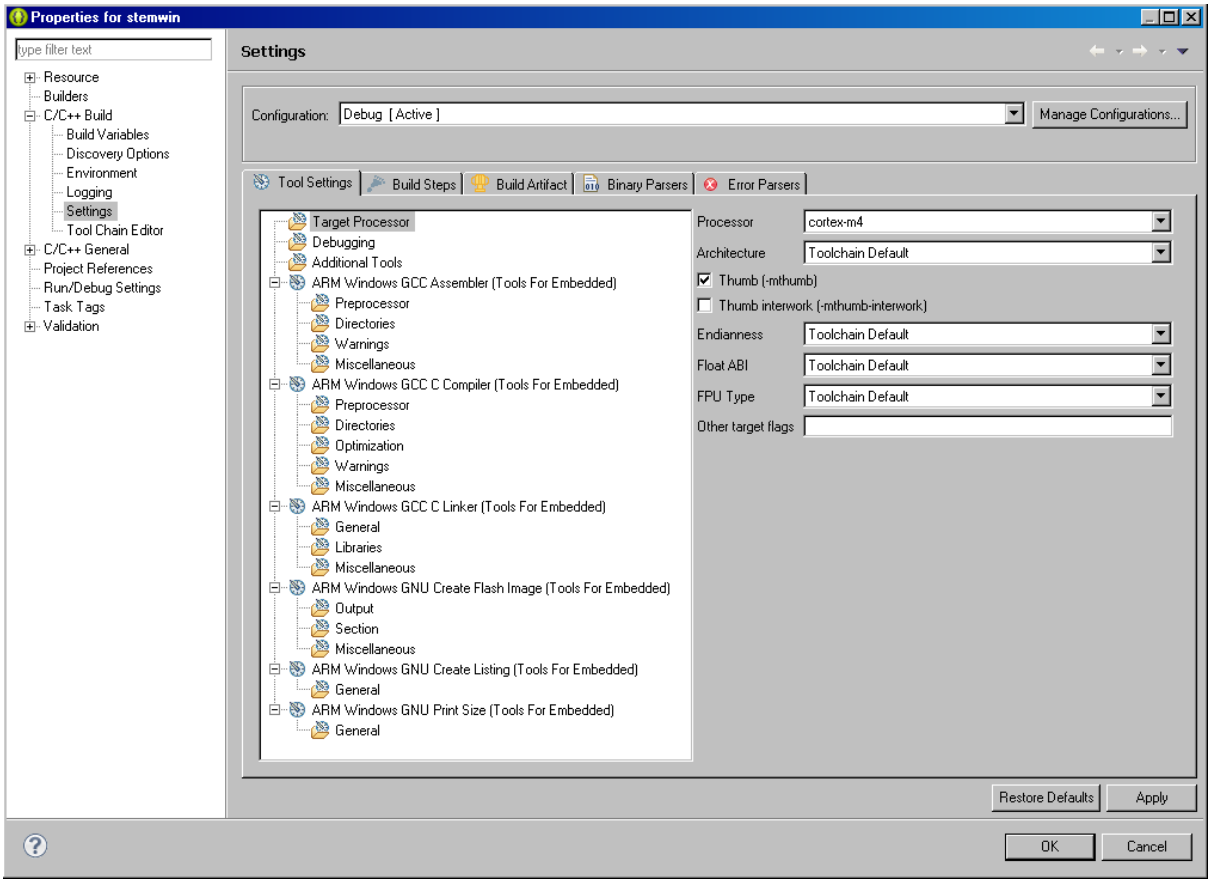
```
int main(void)
{
    return 0;
}
```

Функция main – точка входа в программу.

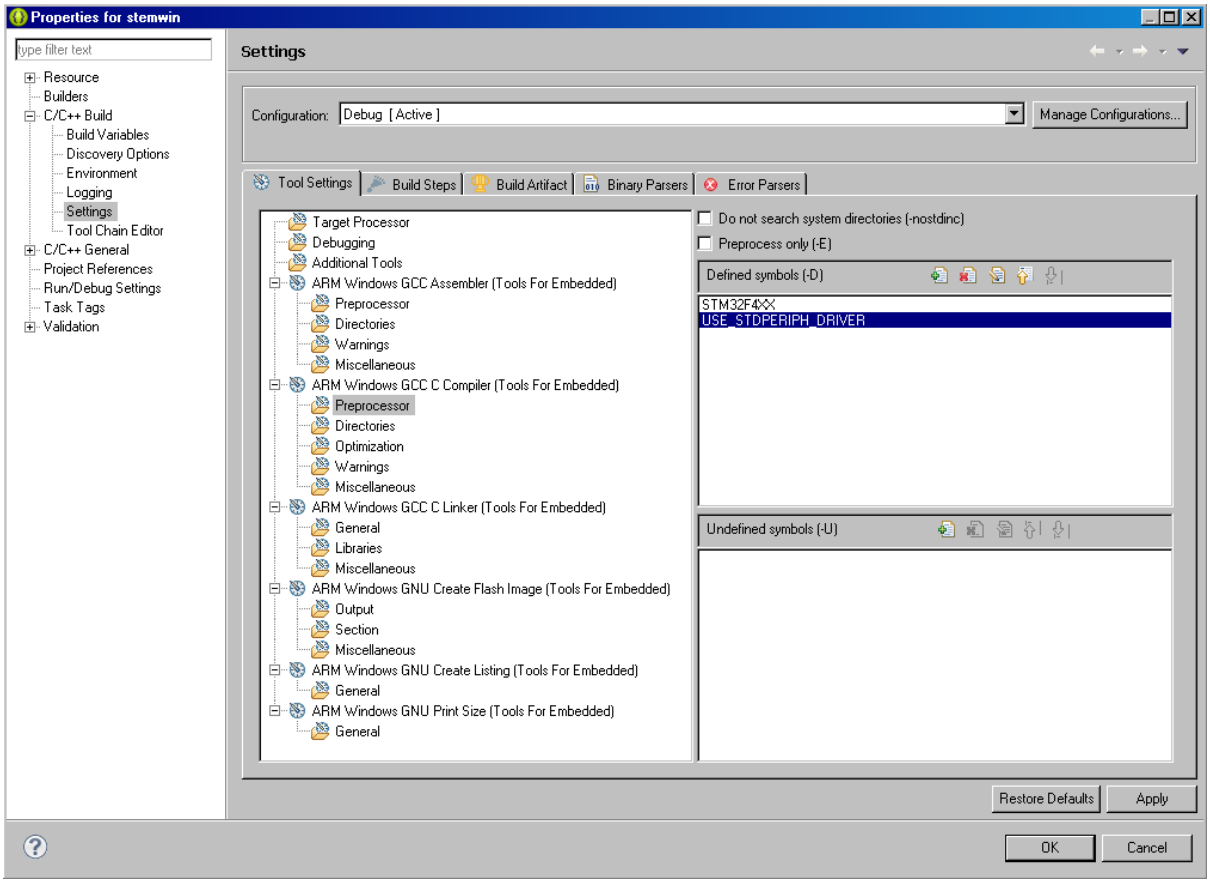
7) Добавить к проекту файл линкера stm32_flash.ld
После этого шага проект д. выглядеть так:



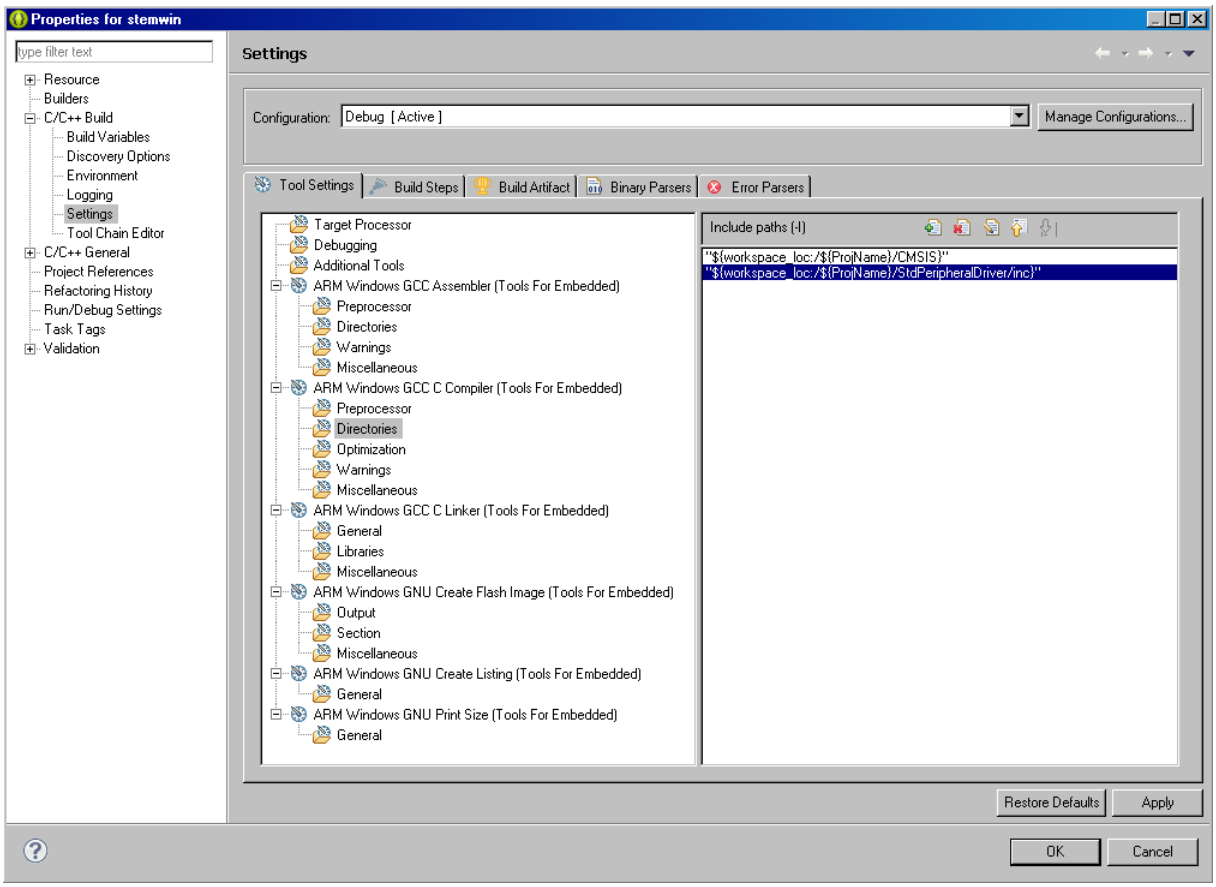
8) Настроить проект
Project->Properties
Выбрать процессор Cortex-M4



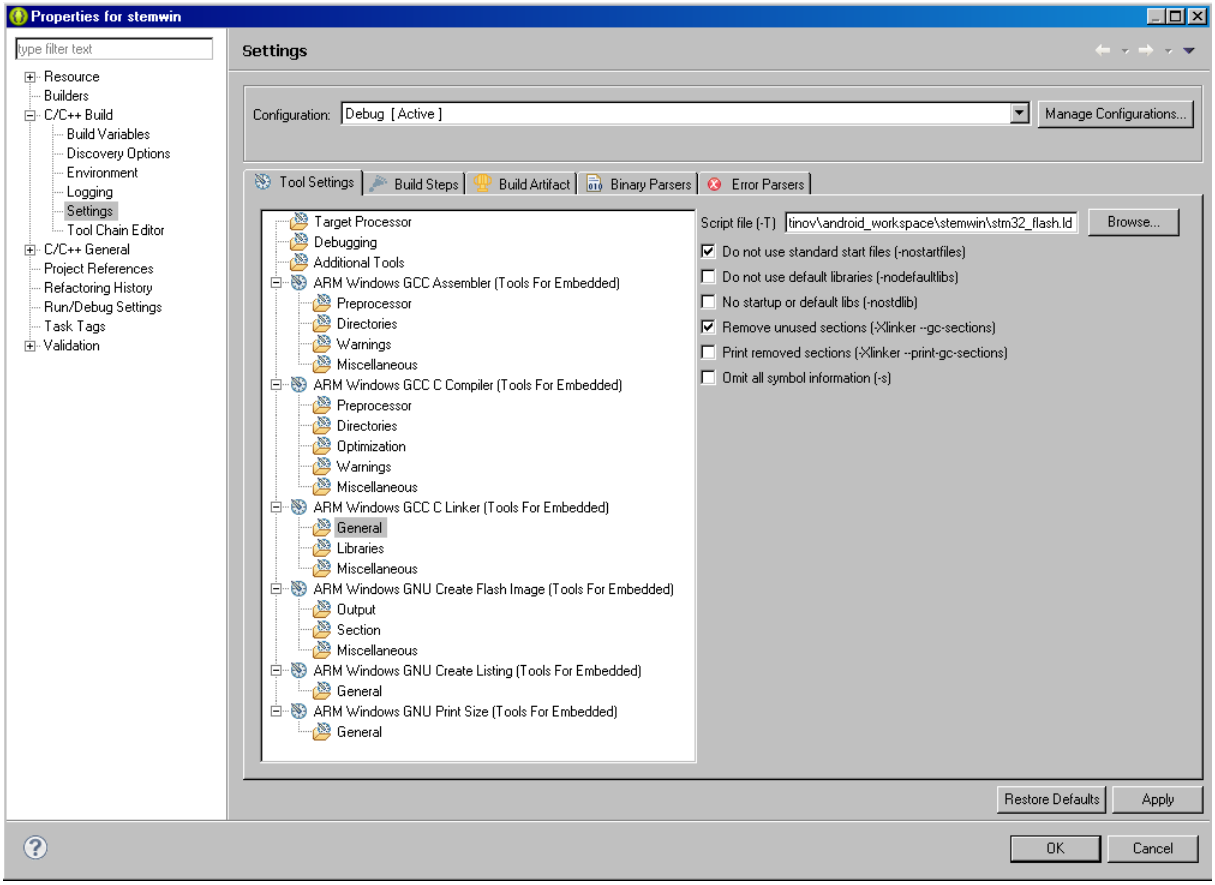
Добавить «определенные» символы компилятора STM32F4XX и USE_STDPERIPH_DRIVER



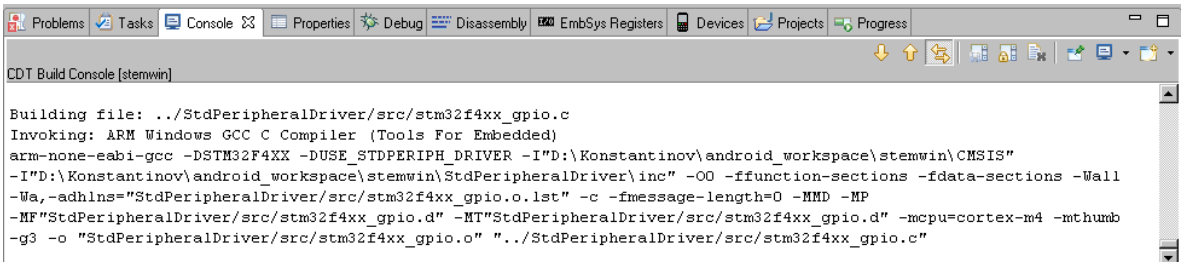
Добавить включаемые каталоги компилятора: CMSIS и StdPeripheralDriver/inc



Указать скрипт линкера `stm32_flash.ld`. Не использовать стандартные стартовые файлы (у нас свой - `startup_stm32f4xx.S`)

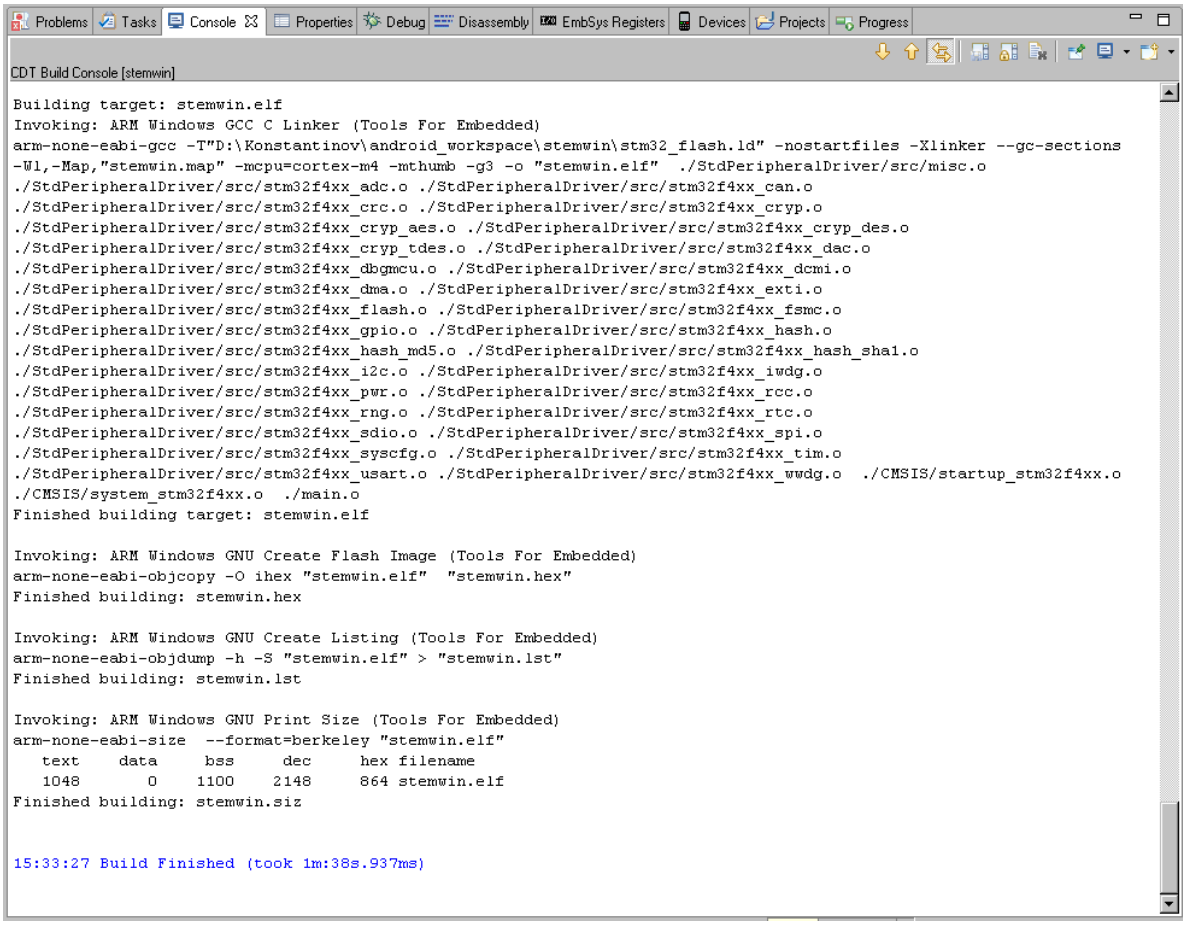


9) Выполнить компиляцию проекта
Project -> Build Project



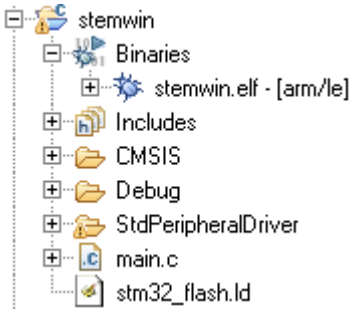
В процессе компиляции было показано предупреждение о не используемой переменной tmp_size. Не обращаем на это предупреждение внимание.

```
Building file: ../StdPeripheralDriver/src/stm32f4xx_flash.c
Invoking: ARM Windows GCC C Compiler (Tools For Embedded)
arm-none-eabi-gcc -DSTM32F4XX -DUSE_STDPERIPH_DRIVER -I"D:\Konstantinov\android_workspace\stemwin\CMSIS"
-I"D:\Konstantinov\android_workspace\stemwin\StdPeripheralDriver\inc" -O0 -ffunction-sections -fdata-sections -Wall
-Wa,-adhlns="StdPeripheralDriver/src/stm32f4xx_flash.o.lst" -c -fmessage-length=0 -MMD -MP
-MF"StdPeripheralDriver/src/stm32f4xx_flash.d" -MT"StdPeripheralDriver/src/stm32f4xx_flash.d" -mcpu=cortex-m4 -mthumb
-g3 -o "StdPeripheralDriver/src/stm32f4xx_flash.o" "../StdPeripheralDriver/src/stm32f4xx_flash.c"
../StdPeripheralDriver/src/stm32f4xx_flash.c: In function 'FLASH_EraseAllSectors':
../StdPeripheralDriver/src/stm32f4xx_flash.c:434:12: warning: variable 'tmp_psize' set but not used
[-Wunused-but-set-variable]
Finished building: ../StdPeripheralDriver/src/stm32f4xx_flash.c
```



Компиляция прошла успешно, создан исполняемый файл `stemwin.elf`.

10) Структура проекта



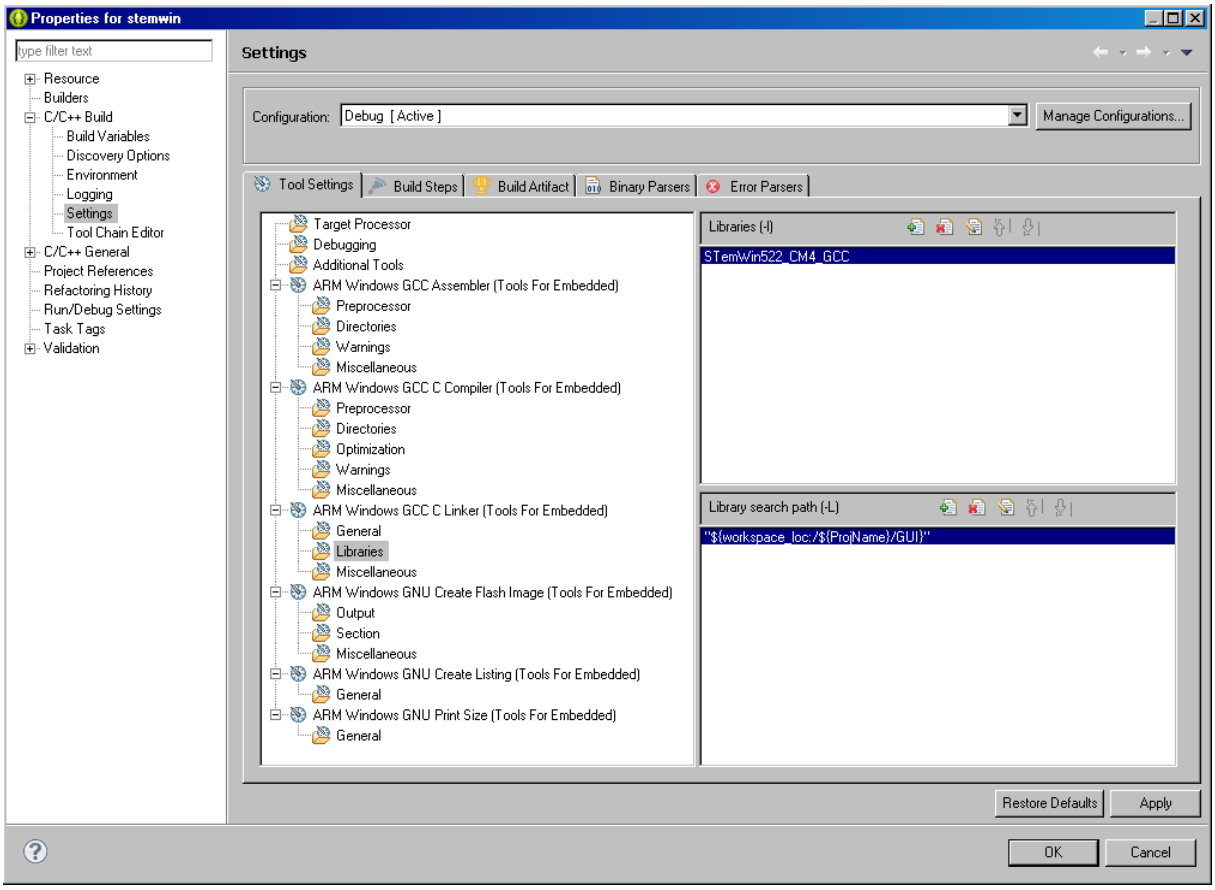
Далее включим в проект библиотеку STemWin.

11) Создать в проекте каталог GUI, куда будем складывать все, относящееся к Graphical User Interface.

12) Поместить в каталог GUI библиотеку `STemWin522_CM4_GCC.a` из каталога STemWin_Library_V1.1.0\Libraries\STemWinLibrary522\Lib\

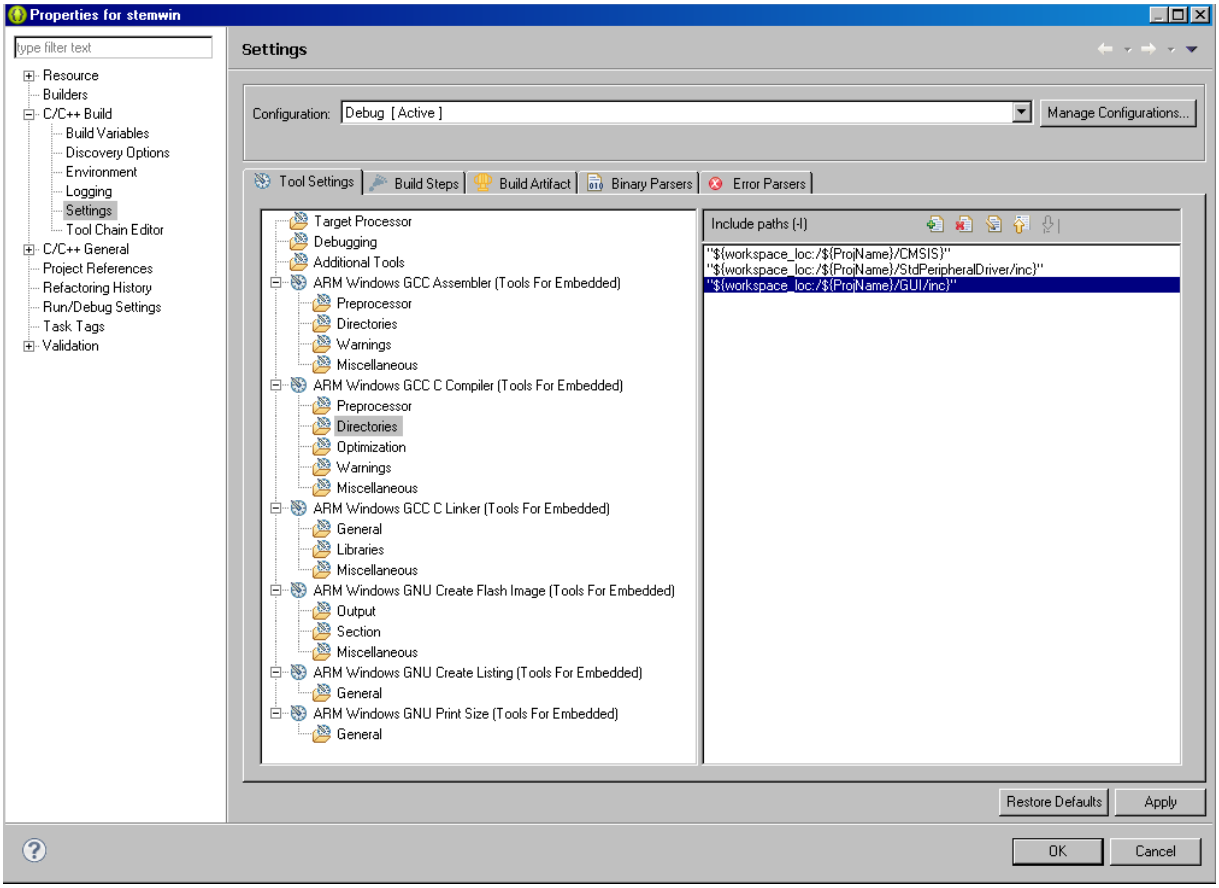
Добавить префикс lib к имени библиотеки: таким образом переименовать библиотеку из `STemWin522_CM4_GCC.a` в `libSTemWin522_CM4_GCC.a`
Это нужно для особенностей компилятора GCC - у библиотеки д. б. префикс lib.

Добавить включение библиотеки `libSTemWin522_CM4_GCC.a` и путь к ней в путях поиска библиотек линкера.



13) Поместить в каталог GUI каталог STemWin_Library_V1.1.0\Libraries\STemWinLibrary522\inc, содержащий заголовочные файлы библиотеки STemWin.

Добавить каталог в пути компилятора



14) Создать каталог STM32_EVAL в проекте
В этом каталоге будет все, что относиться к самой отладочной плате STM3241G_EVAL.

15) Добавить в каталог STM32_EVAL каталог STemWin_Library_V1.1.0\Utilities\STM32_EVAL\STM3240_41_G_EVAL, содержащий файлы настройки периферии для отладочной платы STM3241G_EVAL. Хотя некоторые файлы из него и не нужны, но поскольку он стандартный, оставим все как есть.

Добавить в пути компилятора путь к каталогу STM3241G_EVAL.

16) Добавить в каталог STM32_EVAL каталог STemWin_Library_V1.1.0\Utilities\STM32_EVAL\Common, содержащий общие настройки для отладочных плат STM32_EVAL.

В каталоге Common нужно переименовать файл `lcd_log_conf_template.h` в `lcd_log_conf.h` и в строке

```
#include "stm32xxx_eval_lcd.h" /* replace 'stm32xxx' with your
                                  EVAL board name, ex:
                                  stm3210c_eval_lcd.h */
```

Заменить `stm32xxx_eval_lcd.h` на `stm324xg_eval_lcd.h`

Добавить в пути компилятора путь к каталогу Common.

17) Скопировать в каталог GUI каталог STemWin_Library_V1.1.0\Project\STM324xG-EVAL\Standalone\Config, содержащий файлы

GUIConf_stm3240g_eval.h/.c
LCDConf_stm3240g_eval.h/.c
global_includes.h

переименовать файлы

GUIConf_stm3240g_eval.h/.c в GUIConf.h/.c

LCDConf_stm3240g_eval.h/.c в LCDConf.h/.c

Добавить в пути компилятора путь к каталогу Config.

18) Скопировать в каталог проекта каталог STemWin_Library_V1.1.0\Project\STM324xG-EVAL\Standalone\Demo, содержащий исполняемые файлы демонстрационных экранов.

Добавить в пути компилятора путь к каталогу Demo.

19) Скопировать из каталога STemWin_Library_V1.1.0\Project\STM324xG-EVAL\Standalone\User следующие файлы:

в каталог Demo:

bsp.h

bsp.c

в каталог проекта:

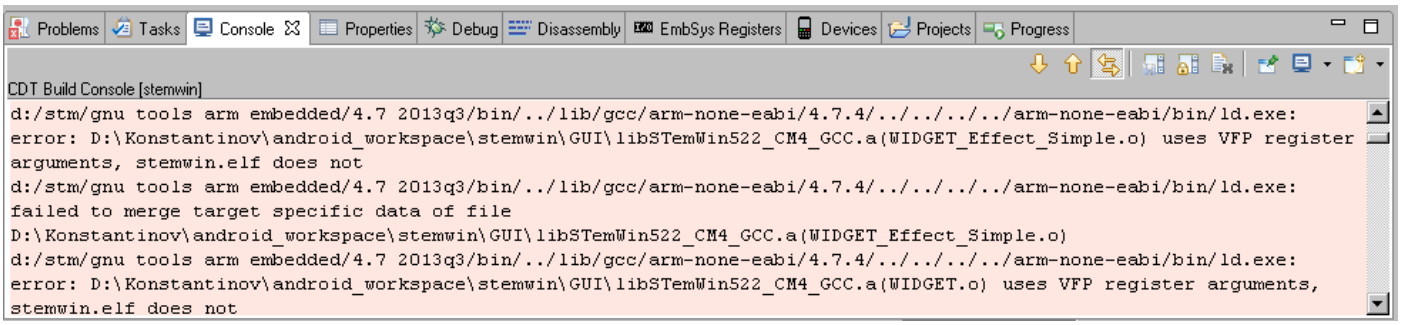
main.c

stm32xxx_it.c

stm32xxx_it.h

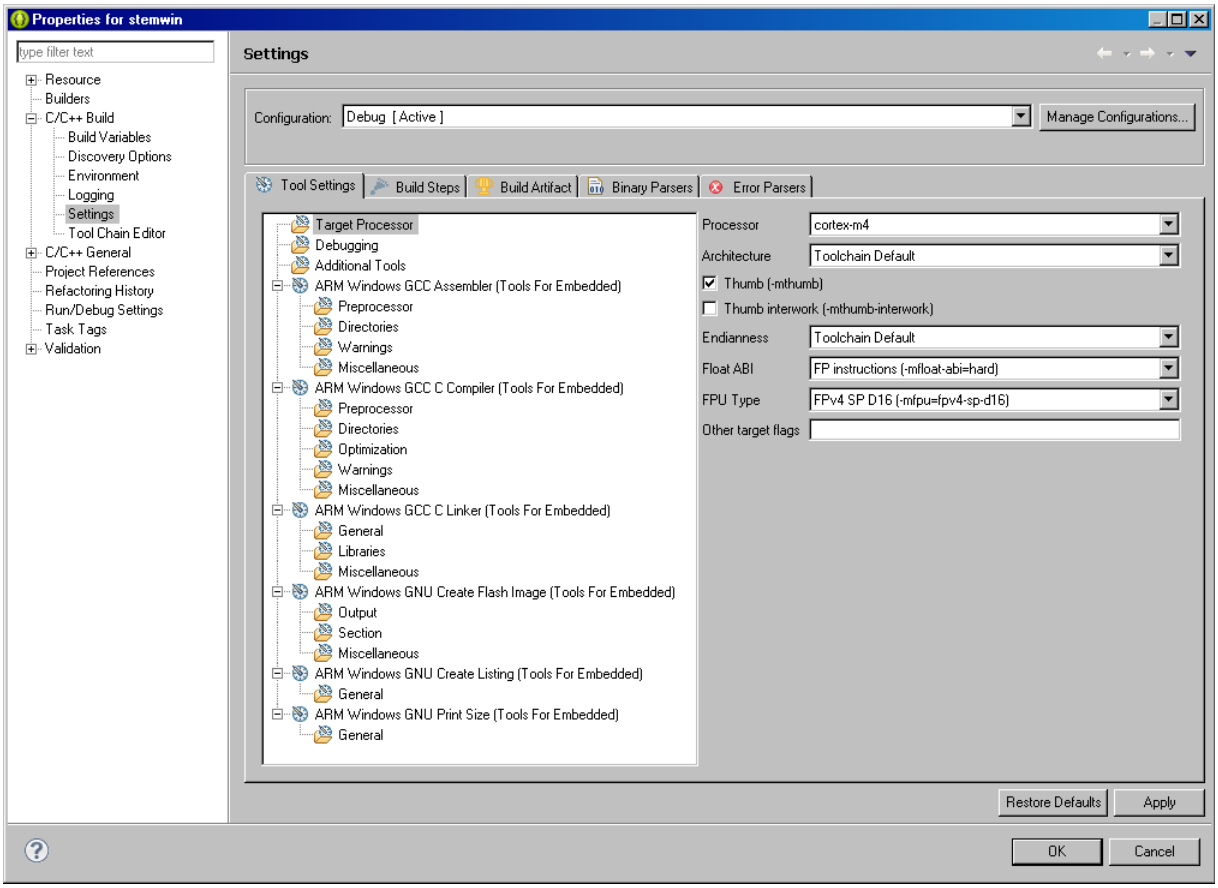
Будет предложено заменить main.c. Согласиться.

20) Начать компиляцию проекта.

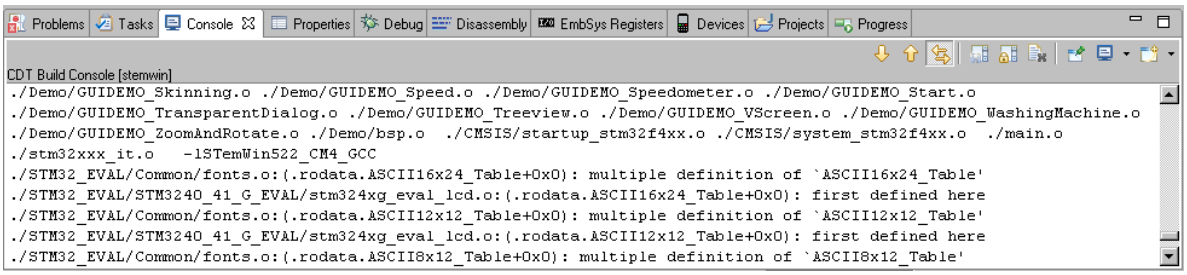


Компилятор сообщит об ошибке libSTemWin522_CM4_GCC.a uses VFP register arguments, stemwin.elf does not. Что говорит о том, что библиотека STemWin использует регистры FPU (Floating Point Unit), так как при компиляции библиотеки разработчиками была указана поддержка FPU. Но в нашем проекте мы не указали использовать FPU. Сделаем это сейчас.

Нужно указать использование FPU. Float ABI: FP instructions. FPU Type: FPv4 SP D16. Именно эти параметры указаны, так как STM32F417 имеет FPU и этот FPU реализует FPv4-SP расширение.



21) Компилируем еще раз и снова ошибка



Ошибка multiple definition of “ASCII16x24_Table” и т. д. из-за двойного включения файла `font.c`:

I) `fonts.c` компилируется из каталога STM32_EVAL/Common

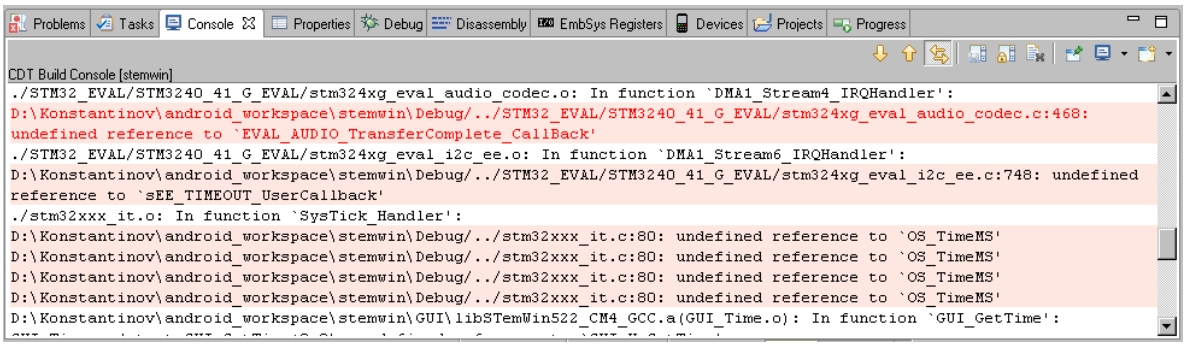
II) `fonts.c` включается в файле `stm324xg_eval_lcd.c`

Следовательно, нужно исключить файл из компиляции, но чтобы его содержимое было в файле `stm324xg_eval_lcd.c`.

Для этого переименовываем `fonts.c` в `fonts_c.h` и делаем такое же переименование внутри файла `stm32xg_eval_lcd.c`

`#include` `"../Common/fonts.c"` -> **`#include`** `"../Common/fonts_c.h"`

22) Компилируем еще раз и снова ошибки



I) undefined reference to EVAL_AUDIO_TransferComplete_CallBack - Неопределенная ссылка на функцию EVAL_AUDIO_TransferComplete_CallBack. Открываем файл `stm324xg_eval_audio_codec.c` и вставляем свою функцию-заглушку, как написано в заголовочном файле `stm324xg_eval_audio_codec.h`.

```
      /* Manage the remaining file size and new address
offset: This function should be coded by user (its prototype
is already declared in stm32_eval_audio_codec.h) */

void EVAL_AUDIO_TransferComplete_CallBack(uint32_t pBuffer,
uint32_t Size)
{}
```

Поскольку функция связана с аудио, а мы подключаем графическую библиотеку, то не особо заботимся о ее содержимом.

II) undefined reference to EVAL sEE_TIMEOUT_UserCallback()- неопределенная ссылка на функцию sEE_TIMEOUT_UserCallback().
Открываем файл `stm324xg_eval_i2c_ee.h` и раскомментируем #define USE_DEFAULT_TIMEOUT_CALLBACK, как и написано в самом файле:

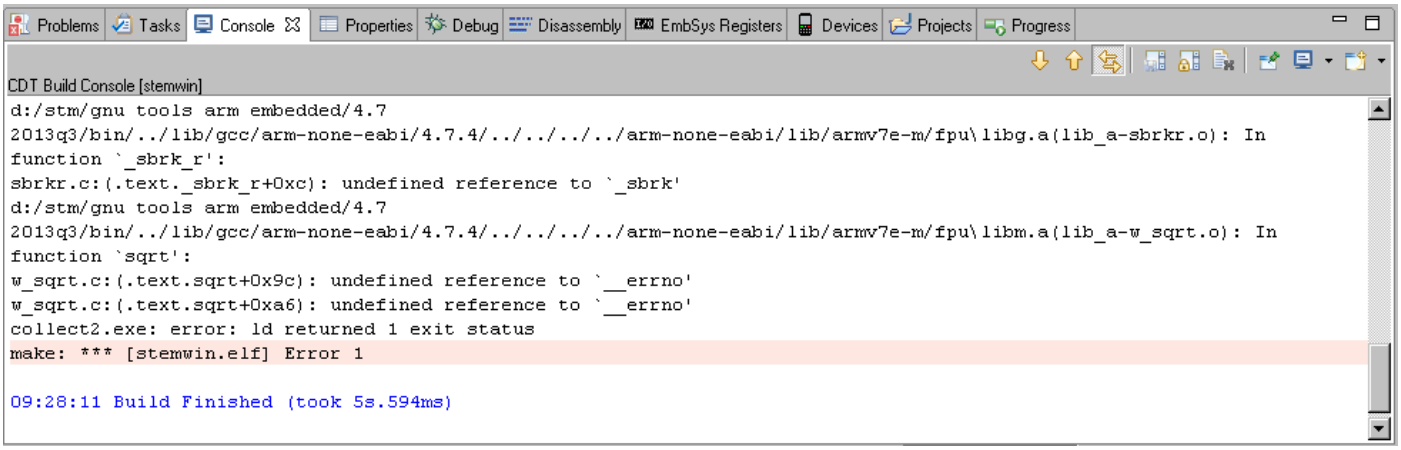
```
 /* Uncomment the following line to use the default
sEE_TIMEOUT_UserCallback() function implemented in
stm32_evel_i2c_ee.c file. sEE_TIMEOUT_UserCallback() function
is called whenever a timeout condition occure during
communication (waiting on an event that doesn't occur, bus
errors, busy devices ...). */
/* #define USE_DEFAULT_TIMEOUT_CALLBACK */
```

III) undefined reference to OS_TimeMS - неопределенная ссылка на OS_TimeMS.
В самом файле `stm32xxx_it.c` эта переменная объявлена как:

```
extern __IO int32_t OS_TimeMS;
```

Истинное определение в файле `GUI_X.c`, которого в нашем проекте пока нет.
Скопировать из каталога STemWin_Library_V1.1.0\Libraries\STemWinLibrary522\OS файл `GUI_X.c` в каталог GUI проекта.

23) Компилируем еще раз и снова ошибки



Здесь начинается самое интересное.

I) undefined reference to '_sbrk' – неопределенная ссылка на _sbrk.

_sbrk – функция выделения памяти в куче (heap).

Напишем свою функцию sbrk в файле `main.c`:

```
extern int  __HEAP_START;
caddr_t _sbrk ( int incr )
{
  static unsigned char *heap = 0;
  unsigned char *prev_heap;

  if (heap == 0) {
    heap = (unsigned char *)&__HEAP_START;
  }
  prev_heap = heap;
  /* check removed to show basic approach */

  heap += incr;

  return (caddr_t) prev_heap;
}
```

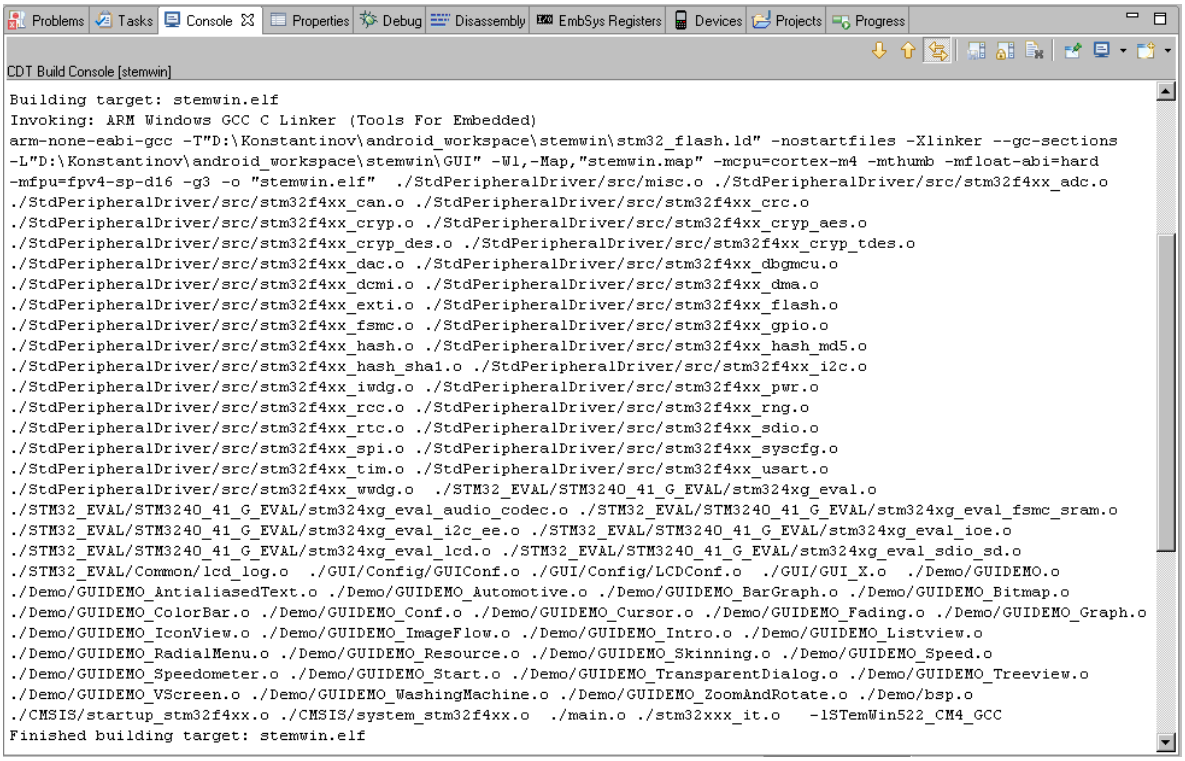
и добавим __HEAP_START в файл линкера в секцию ._user_heap_stack:

```
  /* User_heap_stack section, used to check that there is
enough RAM left */
  ._user_heap_stack :
  {
    . = ALIGN(4);
    PROVIDE(__HEAP_START = . );   /* !!! */
    . = . + _Min_Heap_Size;
    . = . + _Min_Stack_Size;
    . = ALIGN(4);
  } >RAM
```

II) undefined reference to __errno – неопределенная ссылка на __errno.
Напишем функцию-заглушку в файле `main.c`

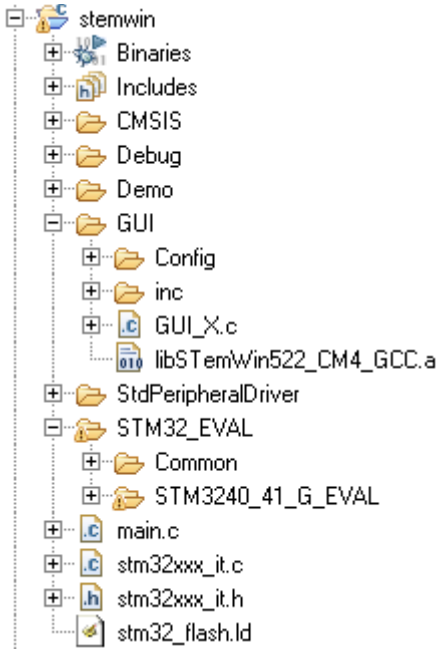
```
void __errno(void)    //dummy
{

}
```

24) На этом все. Компилируем и загружаем `stemwin.elf` в демо-плату и вуаля.

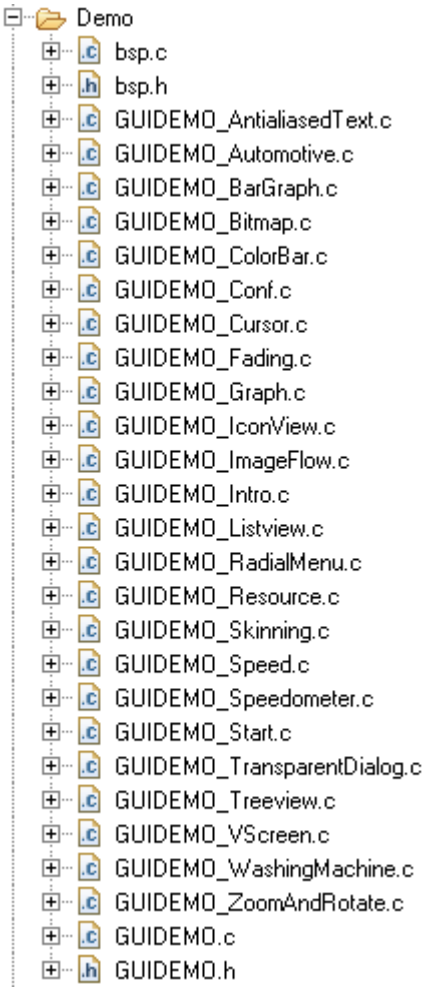


И еще, у меня получилось так, что символ определен, программа компилируется, а Eclipse подчеркивает его как symbol could not be resolved. Решение данной проблемы – проверьте, чтобы в настройках проекта в C/C++ general -> Indexer стояли все галочки.

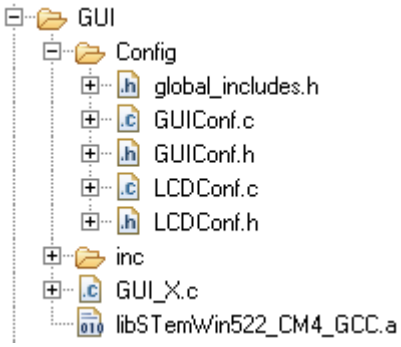
Окончательная структура проекта:



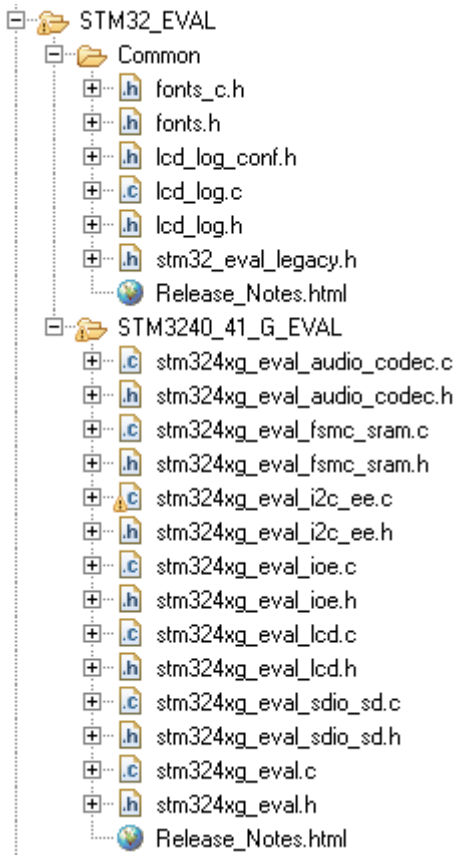
Содержание каталога Demo:



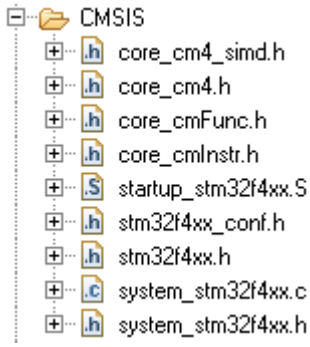
Содержание каталога GUI:



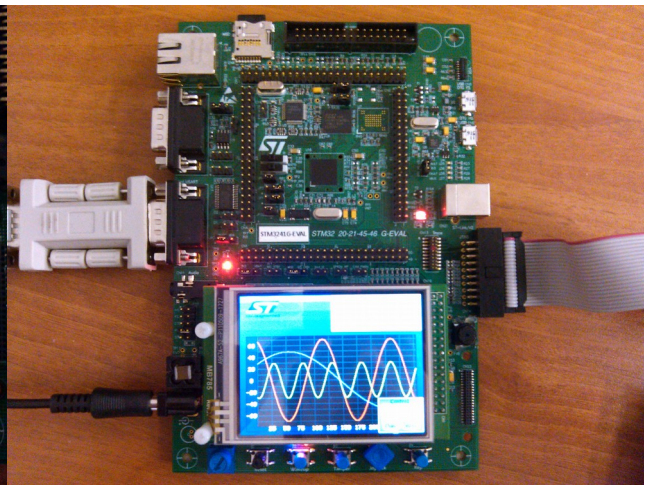
Содержание каталога STM32_EVAL:



Содержание каталога CMSIS:

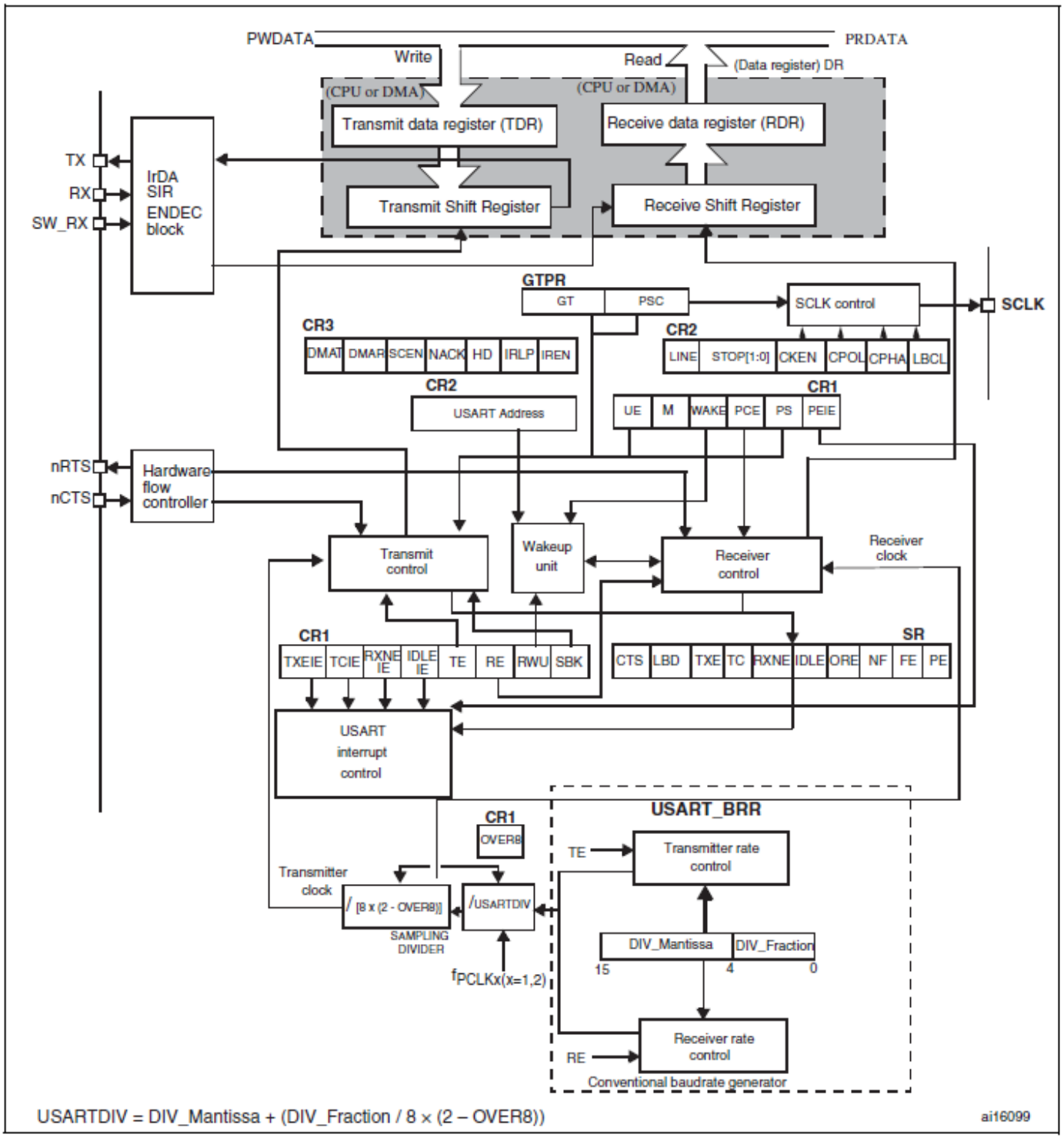


Фотографии отладочной платы STM3241G_EVAL, на которой запущено демонстрационное приложение, использующее библиотеку STemWin.



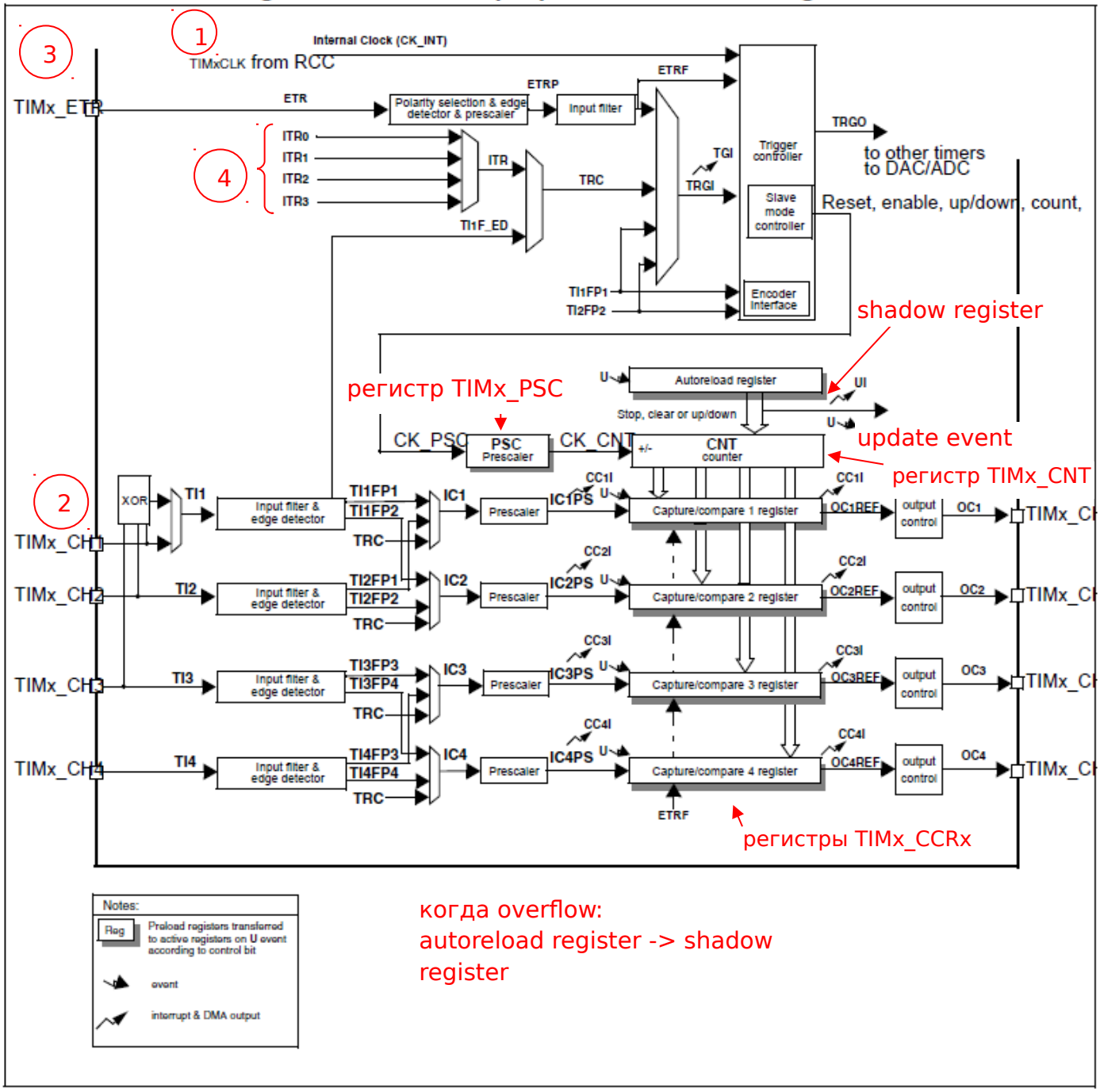
12. USART

Figure 296. USART block diagram



13. TIMER

Figure 134. General-purpose timer block diagram



Clock для таймера:

1. Внутренний clock (CK_INT)
2. Внешний clock (TIx)
3. Внешний триггерный вход (ETR)
4. Внутренние триггерные входы (ITRx). Используется один таймер, как предделитель для другого.

Модуль таймера включает в себя:

1. Counter Register (TIMx_CNT)

2. Prescaler Register (TIMx_PSC)
3. Auto-Reload Register (TIMx_ARR)

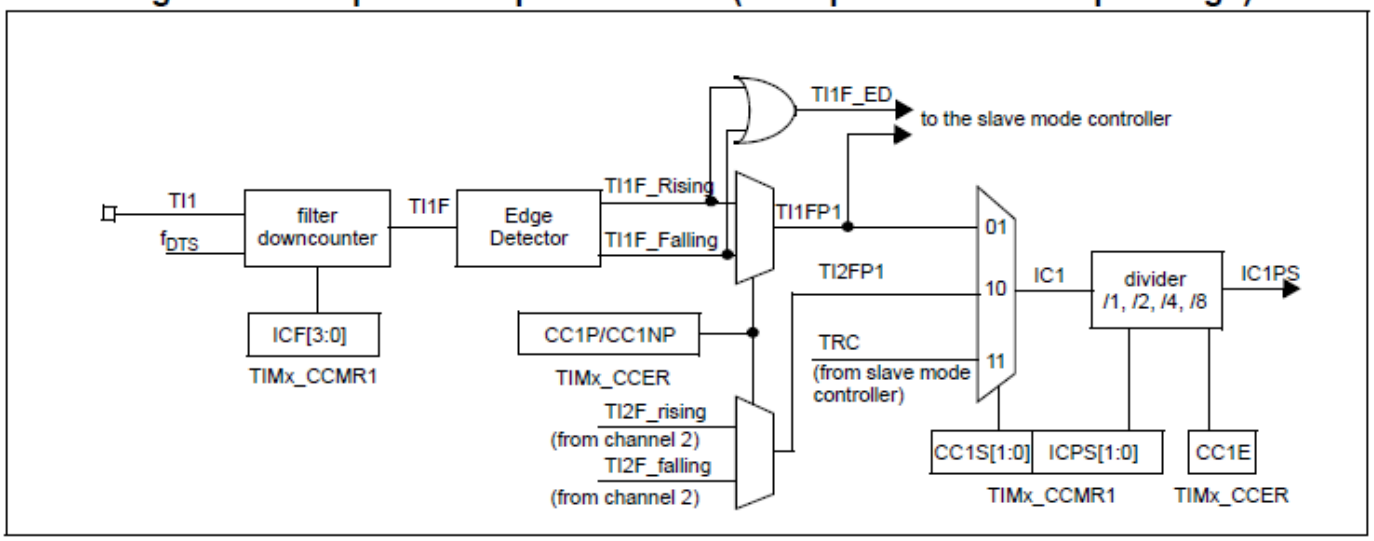
Auto-Reload Register
Содержимое Auto-Reload Register помещается в Shadow Register постоянно или в каждом Update Event (UEV), в зависимости от конфигурации. Update Event генерируется при переполнении таймера.

Prescaler Register
Предделитель делит тактирующую частоту таймера на любое значение от 1 до 65536 (16 бит). Значение предделителя можно менять на ходу, т. к. этот регистр буферизирован. Значение вступает в эффект при следующем Update Event.

Режимы работы таймера:
1) Upcounting mode
В этом режиме таймер считает от 0 до значения в Auto-Reload Register, затем перезапускается заново с 0 и генерирует событие «переполнение счётчика».
Когда происходит Update Event:
- Все регистры обновляются
- устанавливается флаг UIF
- буфер предделителя обновляется значением в регистре TIMx_PSC.
- Auto-Reload Shadow Register обновляется значением регистра TIMx_ARR.
! Все эти буферы и скрытые регистры нужны для того, чтобы м. было менять значение «на ходу», не останавливая таймер, при каждом Update Event (например, при переполнении, или совпадении и т. п.). Это и есть основное преимущество таймеров.
2) Downcounting mode
3) Center-aligned mode (up/down counting)

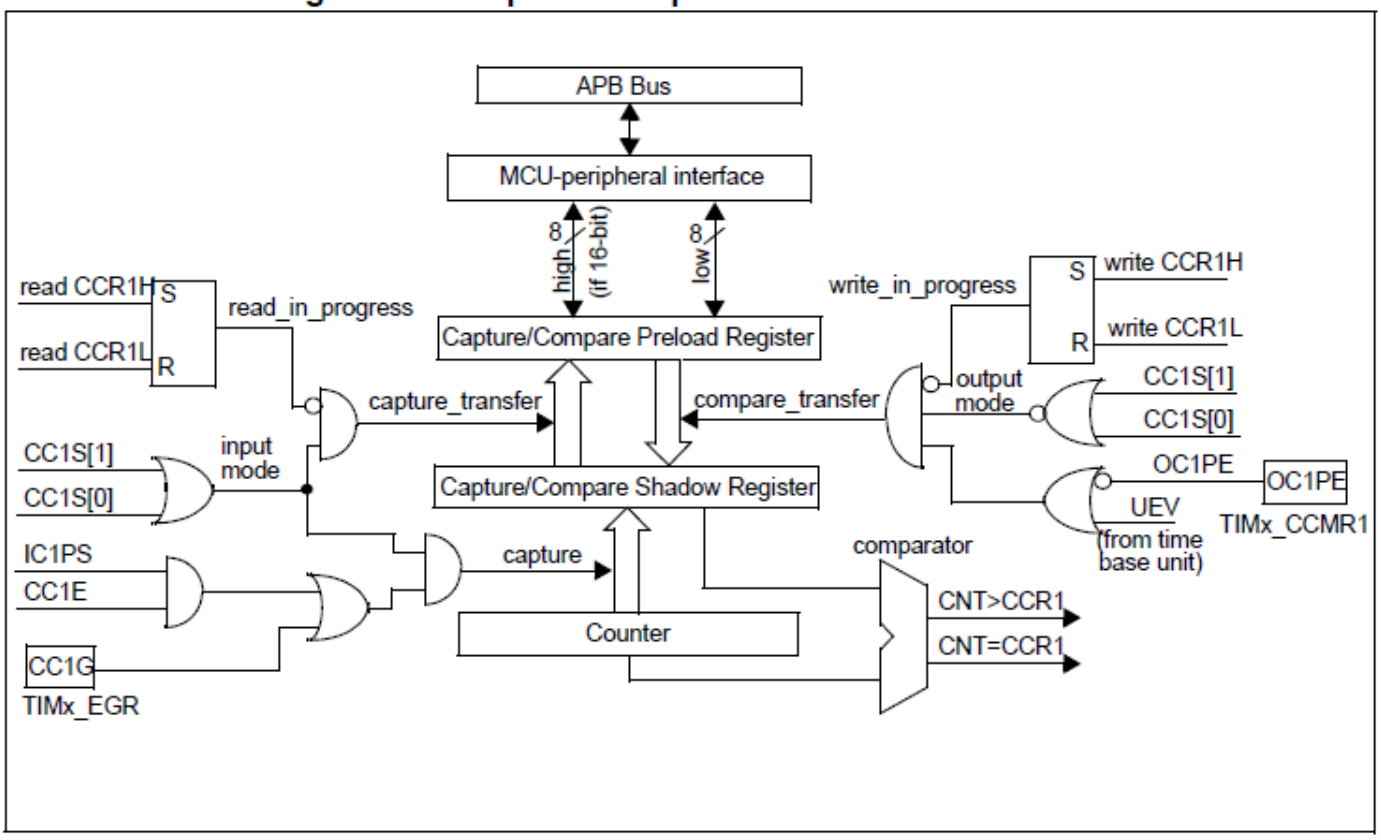
Capture/Compare channels (Каналы Захвата/Сравнения)
Основные компоненты канала захвата/сравнения:
1) Capture/compare register (включая Shadow Register)
2) Входной каскад для захвата (с цифровым фильтром, мультиплексированием и предделителем)
3) Выходной каскад для сравнения (с компаратором)

Figure 159. Capture/compare channel (example: channel 1 input stage)



Входной каскад семплирует вход TI1 для получения фильтрованного сигнала TI1F. Затем детектор фронта генерирует сигнал TI1FP1, который м. б. команда на захват или как триггерный вход для slave mode controller. TI1FP1 делится предделителем для получения значения регистра захвата IC1PS.

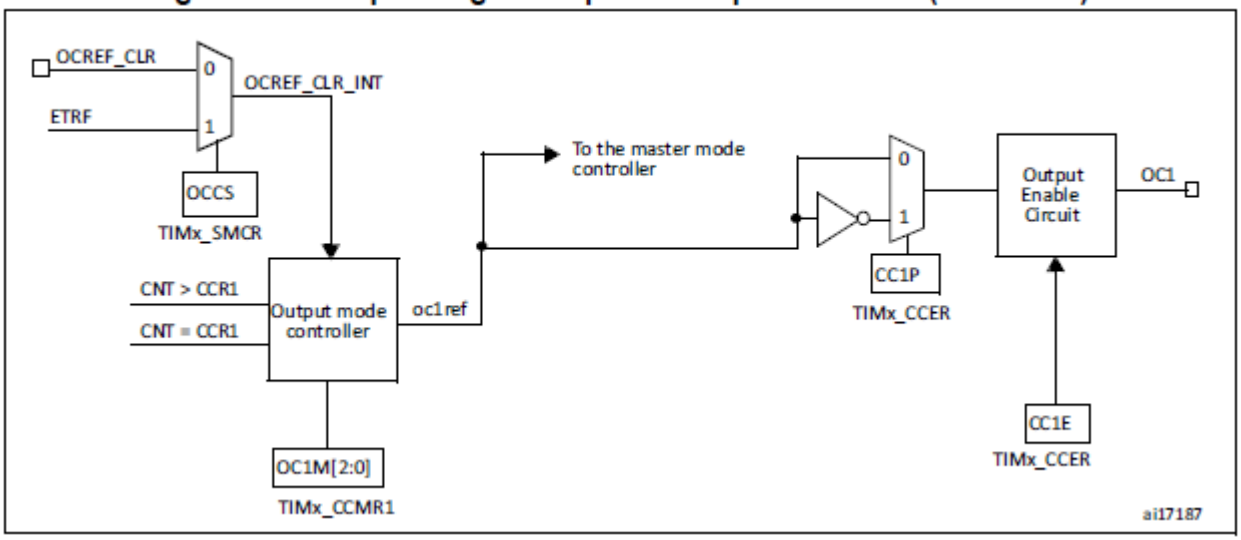
Figure 160. Capture/compare channel 1 main circuit



Модуль захвата/сравнения сделан из одного Preload Register и одного Shadow Register. Запись или чтение всегда имеет доступ только к Preload Register.

В режиме захвата, захват на самом деле происходит в Shadow Register, который затем копируется в Preload Register.

Figure 161. Output stage of capture/compare channel (channel 1)



В режиме сравнения, содержимое Preload Register копируется в Shadow Register, который затем сравнивается с Counter.

14. USB OTG FS и USB OTG HS.

14.1. ОБЩАЯ ИНФОРМАЦИЯ

USB требует программный стек. USB организован по топологии «звезда», где одно устройство – host, а остальные – device. Host организует трафик.

Скорость USB:

HS – High Speed	480 Mbits/s
FS – Full Speed	12 Mbits/s
LS – Low Speed	1.5 Mbits/s

Классы device:

OTG	On the Go. Способность менять функциональность с device на host и обратно «на лету»
HID	Human Interface Device – мыши, клавиатуры…
CDC	Communication Device Class
и множество других	…

Разъемы USB:
- A на стороне хоста
- B на стороне устройства

Устройство м. иметь несколько **endpoint** (номер оконечной точки). Каждая транзакция приема или передачи данных содержит в себе номер endpoint.
Канал (pipe) – совокупность endpoint и структуры данных (скорость канала, класс канала, номер endpoint и т. д.) для нее в ядре ОС.
При подключении устройства драйверы в ядре ОС создают для него канал(ы).
1 канал – 1 endpoint

Классы endpoint и pipe:
1) Bulk (поточный)

Дает гарантию доставки каждого пакета, поддерживает автоматическую приостановку передачи данных по нежеланию устройства (переполнение или опустошение буфера), но не дает гарантий скорости и задержки доставки. Используется, например, в принтерах и сканерах.

2) Control (управляющий)

Предназначен для обмена с устройством короткими пакетами «вопрос-ответ». Любое устройство имеет управляющий канал 0, который позволяет программному обеспечению ОС прочитать краткую информацию об устройстве, в том числе коды производителя и модели, используемые для выбора драйвера, и список других оконечных точек.

3) Isoch (изохронный)

Позволяет доставлять пакеты без гарантии доставки и без ответов/подтверждений, но с гарантированной скоростью доставки в N пакетов на один период шины (1 КГц у low и full speed, 8 КГц у high speed). Используется для передачи аудио- и видеоинформации.

4) Interrupt (прерывание)

Позволяет доставлять короткие пакеты и в том, и в другом направлении, без получения на них ответа/подтверждения, но с гарантией времени доставки — пакет будет доставлен не позже, чем через N миллисекунд. Например, используется в устройствах ввода (клавиатуры/мыши/джойстики).

Низкоскоростные устройства (например, мышь) не м. иметь поточные и изохронные каналы.

Время шины делится на периоды, в начале периода контроллер передает всей шине пакет «начало периода». Далее в течение периода передаются пакеты прерываний, потом изохронные в требуемом количестве, в оставшееся время в периоде передаются управляющие пакеты и в последнюю очередь поточные.

Активной стороной шины всегда является контроллер, передача пакета данных от устройства к контроллеру реализована как короткий вопрос контроллера и длинный, содержащий данные, ответ устройства. Расписание движения пакетов для каждого периода шины создается совместным усилием аппаратуры контроллера и ПО драйвера, для этого многие контроллеры используют крайне сложный DMA со сложной DMA-программой, формируемой драйвером.

Размер пакета для оконечной точки есть вшитая в таблицу оконечных точек устройства константа, изменению не подлежит. Он выбирается разработчиком устройства из числа тех, что поддерживаются стандартом USB.

14.2. РЕАЛИЗАЦИЯ USB OTG FS в STM32F417

USB OTG FS поддерживает протоколы:

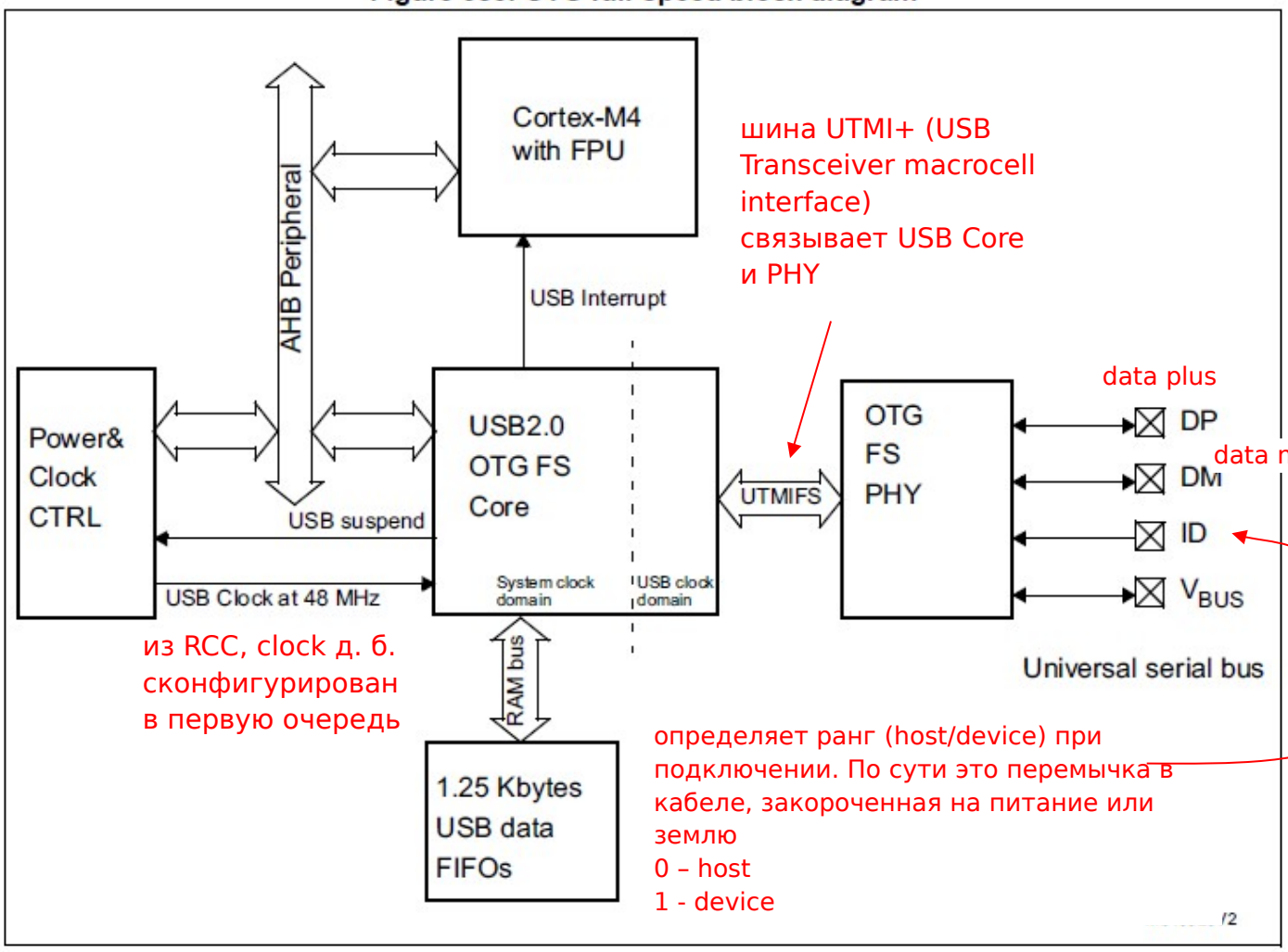
- HNP (Host Negotiation Protocol) – возможность двум USB OTG обмениваться ролями host/device во время работы друг с другом.

- SRP (Session Request Protocol) – запрос устройства начала сессии у хоста. В ответ хост подает питание и т. д.

Host-mode:
- До 8 каналов (pipes); каждый м. б. сконфигурирован на поддержку входа/выхода и любой класс канала.

Device-mode:
- 1 двунаправленная Control endpoint0
- 3 входных endpoint: bulk, interrupt, isoch
- 3 выходных endpoint: bulk, interrupt, isoch
- поддержка soft disconnect

Figure 385. OTG full-speed block diagram



Чтобы гарантировать корректность работы USB OTG FS, частота AHB д. б. выше 14.2MHz.

USB OTG FS имеет модуль динамической подстройки периода SOF. Это нужно для более точной синхронизации с устройством.

14.3. USB OTG FS В РОЛИ ХОСТА

! USB OTG FS не имеет аппаратного Vbus, поэтому роль Vbus играет любой GPIO на выбор. Когда программа решает включить питание Vbus, то д.:
1) Выставить GPIO в 1
2) Указать в USB контроллере, что питание подано (PPWR бит в регистре OTG_FS_HPRT)

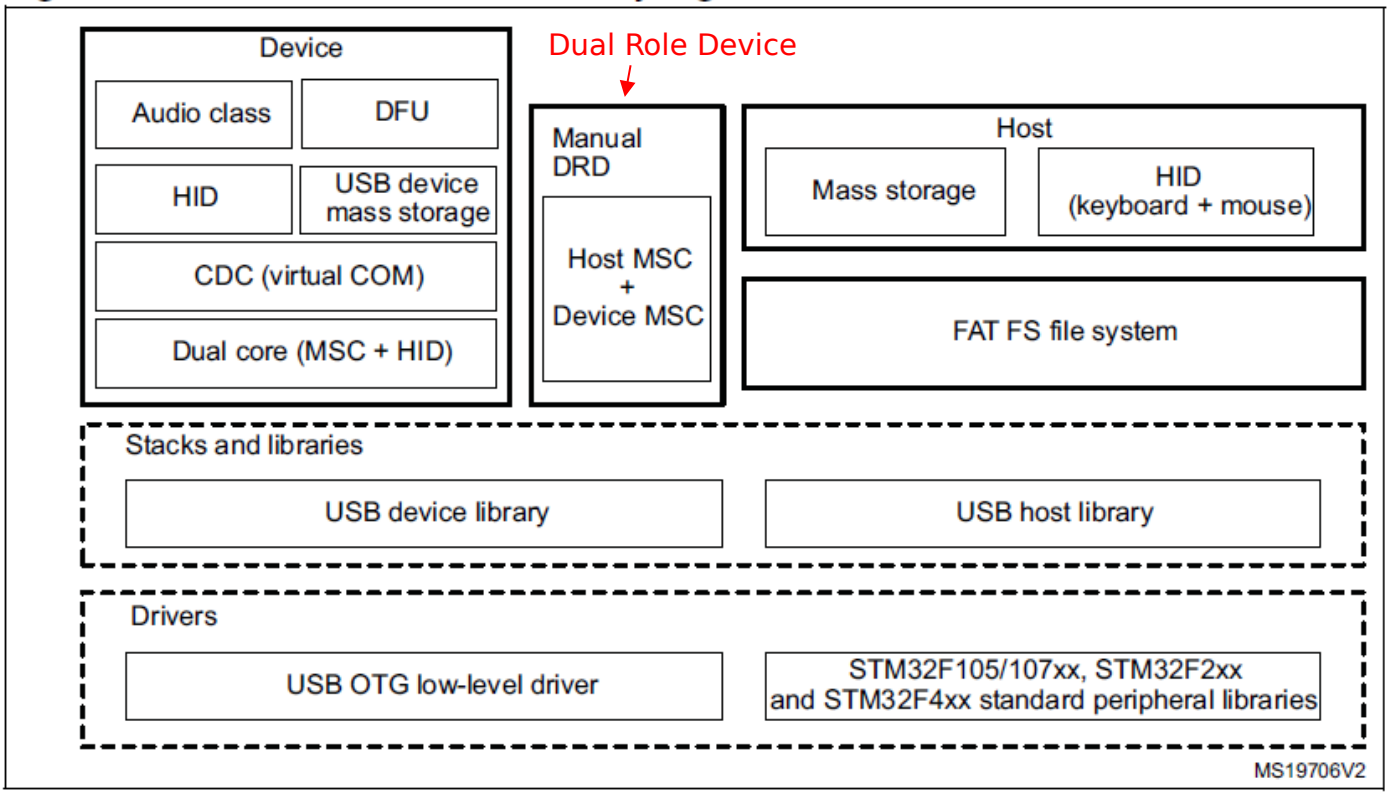
! Когда активированы функции HNP и SRP, то пин PA9 д. б. подключен к Vbus, чтобы измерять напряжение на шине (для реакции на отсутствие напряжения, превышение тока и т. д.). Если напряжение и ток удовл. требованиям (5В и 100мА), возможно подключение device.

Инициализация устройства, выполняемая хостом:
1) Устройство физически подключается, хост генерирует внутреннее прерывание, что подключено новое устройство.
2) Хост удостоверяется, что питание стабильно.
3) Хост делает USB reset (10-20ms)
4) Хост прерывается по биту port enable/disable; теперь скорость устройства м. б. прочитана из статусного регистра хоста; устройство готово принимать SOF (для FS) и Keep alives (для LS).
5) Хост заканчивает инициализацию устройства, отправляя конфигурационные команды устройству.

У хоста есть свой планировщик, планируемый данные по 8 каналам.

14.2. БИБЛИОТЕКА USB от STM.

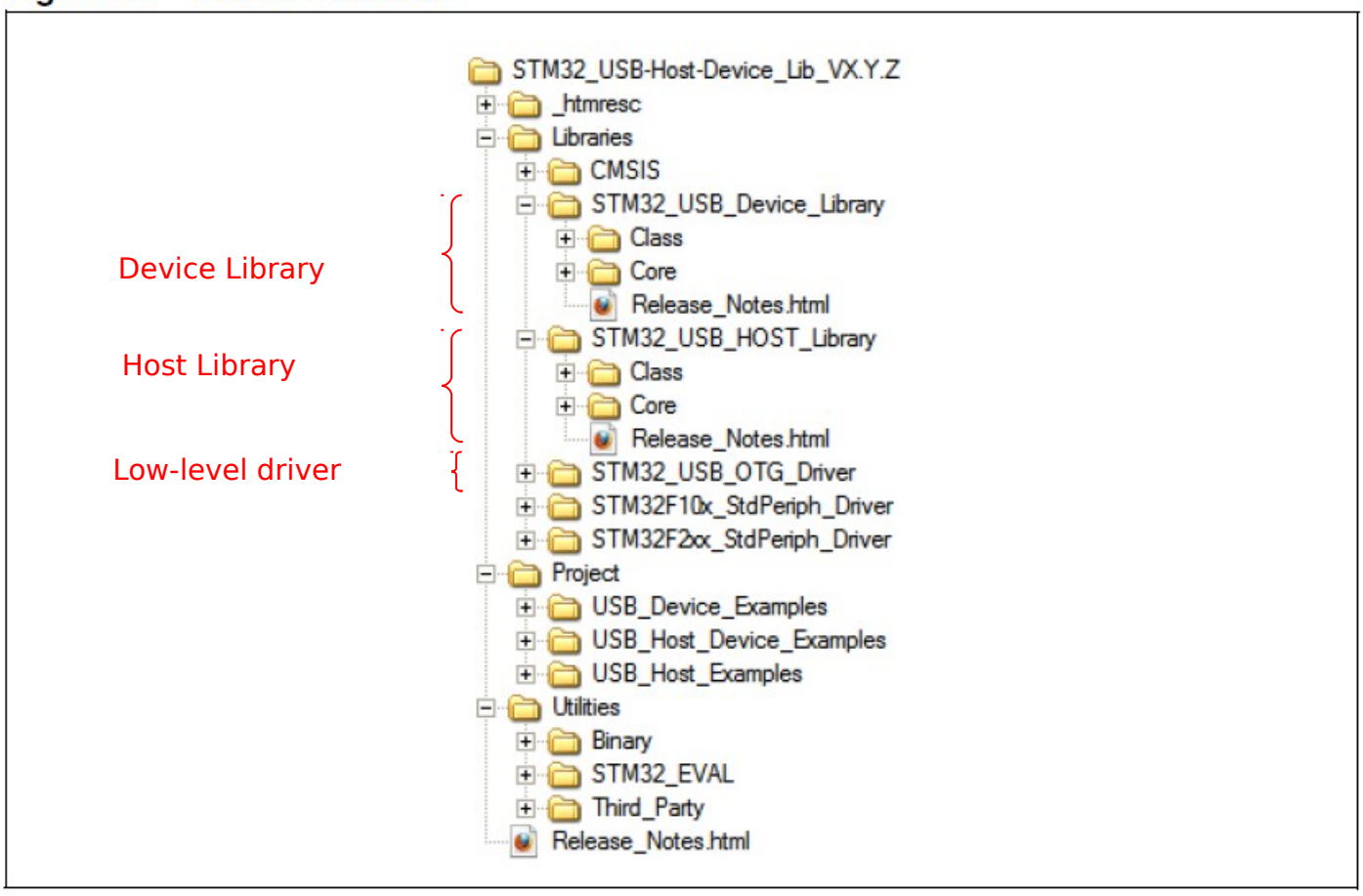
Figure 1. USB host and device library organization overview



The USB host and device libraries are built around the common STM32 USB OTG low level driver and the USB device and host libraries.

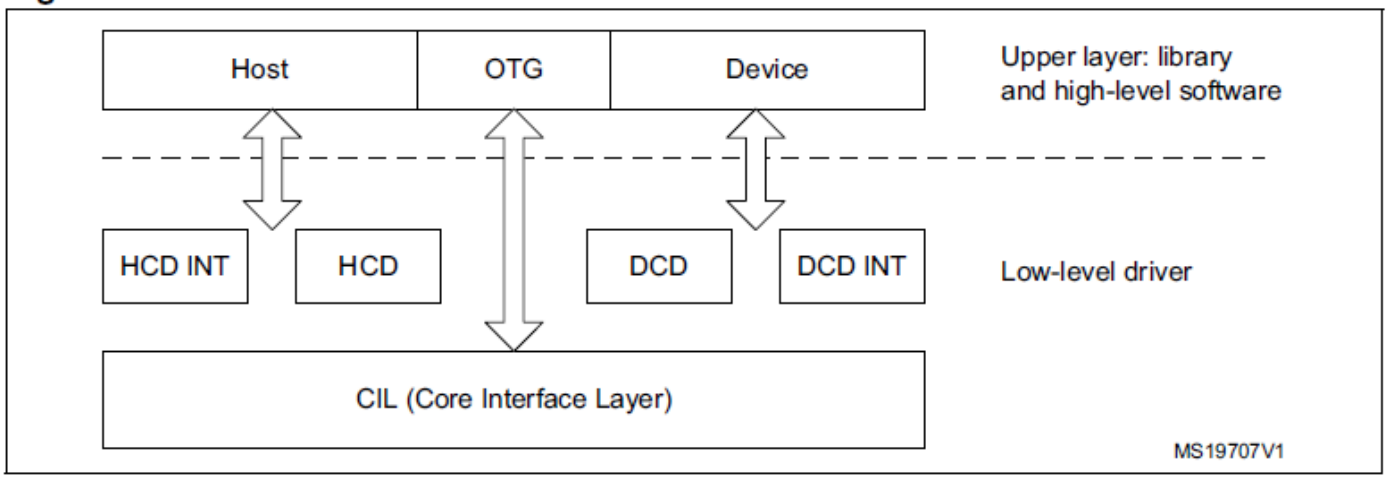
Структура файлов в библиотеке:

Figure 2. Folder structure



Архитектура Low-Level Driver

Figure 3. Driver architecture overview



DCD – Device Core Driver
HCD – Host Core Driver

Глобальная структура USB core, хранит все переменные, состояния и буферы, используемые core для управления своим внутренним состоянием и пересылки данных:

```
typedef struct USB_OTG_handle
{
  USB_OTG_CORE_CFGS    cfg;
  USB_OTG_CORE_REGS    regs;
#ifdef USE_DEVICE_MODE
  DCD_DEV     dev;
#endif
#ifdef USE_HOST_MODE
  HCD_DEV     host;
#endif
#ifdef USE_OTG_MODE
  OTG_DEV     otg;
#endif
}
USB_OTG_CORE_HANDLE , *PUSB_OTG_CORE_HANDLE;
```

!Внимание у USB OTG HS нет встроенного на чипе HS PHY (надо исп. внешний PHY), есть только FS PHY.

15 SDIO

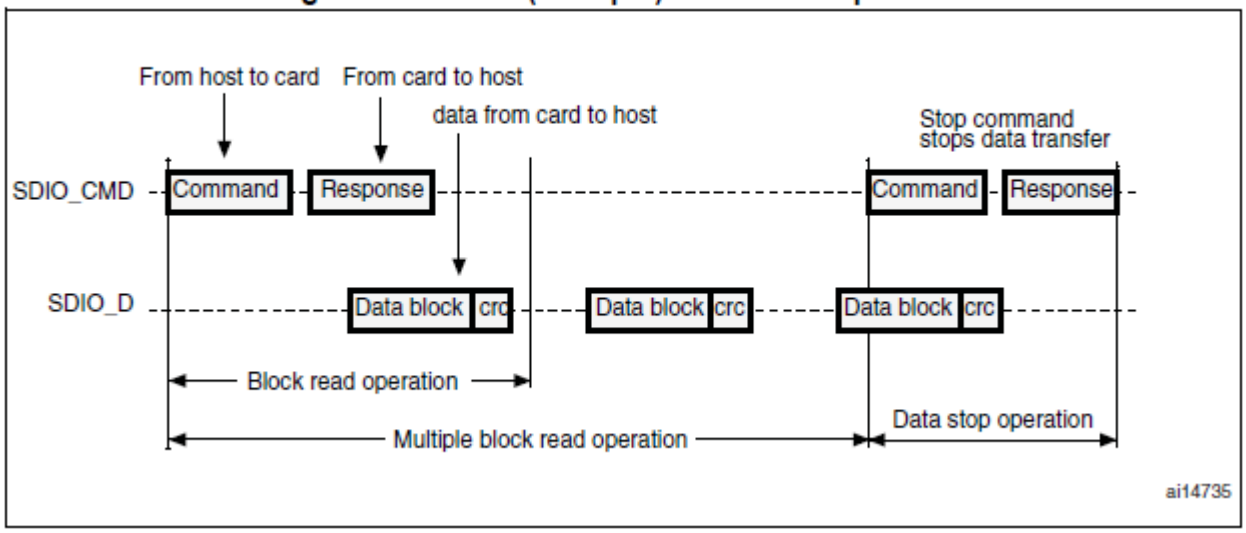
15.1. ОБШАЯ ИНФОРМАЦИЯ

SDIO host interface предоставляет интерфейс между периферийной шиной APB2 и MMC (Multi Media Card), SD memory card, SDIO cards и CE-ATA устройства (потребительская электроника).

Особенности SDIO:
- Полная совместимость с MMC System Specification v4.2. Поддержка 1-бит, 4-бит и 8-бит режима шины данных.
- Полная совместимость с SD Memory Card Specification v2.0
- Полная совместимость с SD I/O Card Specification v2.0: карта поддерживает 2 режима шины данных: 1-бит и 4-бит.
- Полная поддержка цифрового протокола CE-ATA.
- SDIO не имеет поддержки SPI-совместимого коммуникационного режима.

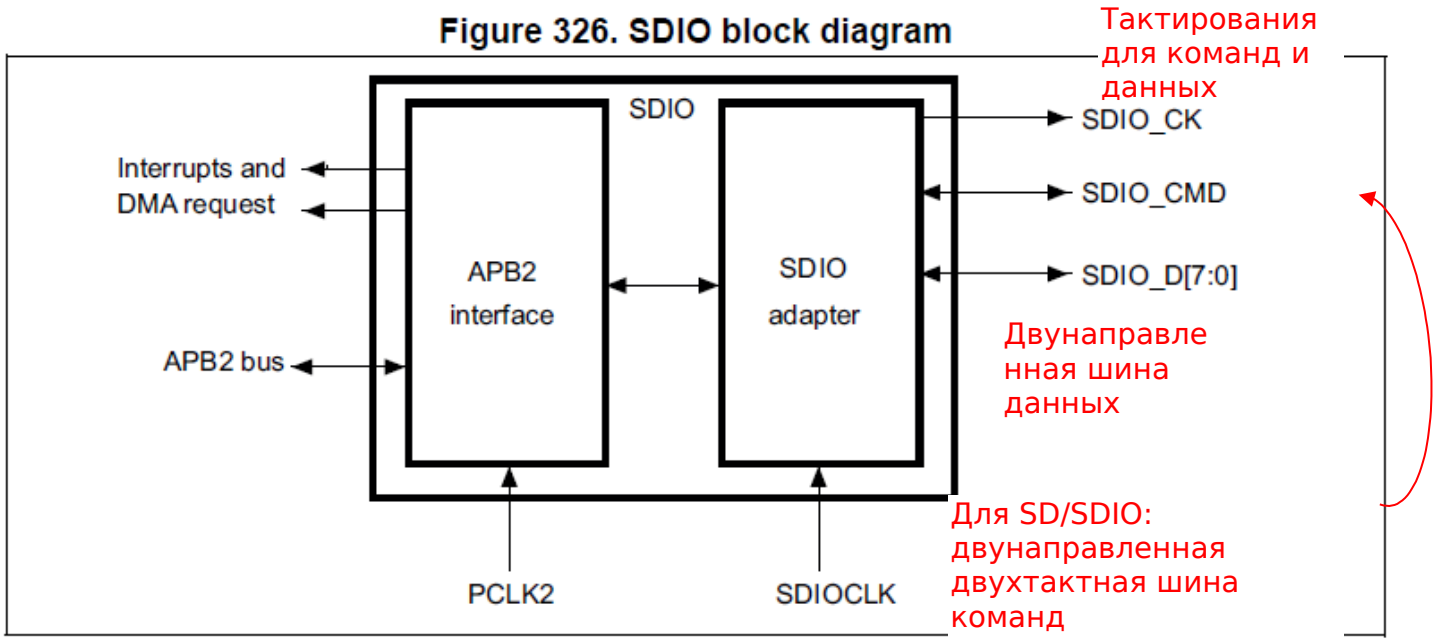
Коммуникация основана на передаче команд и данных.
Основная транзакция – транзакция команда/ответ.
Передача данных в/из SD/SDIO делается только блоками данных.

Figure 322. SDIO (multiple) block read operation



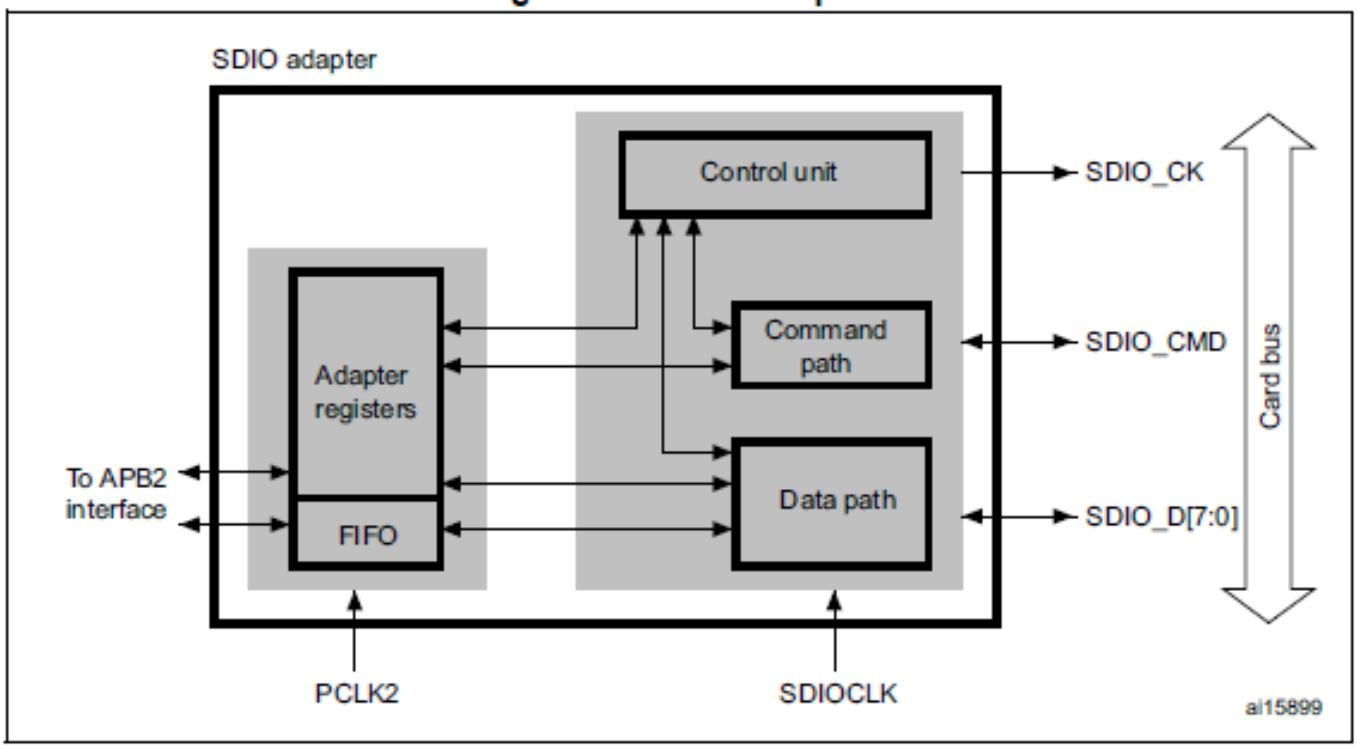
Архитектура модуля SDIO:

Figure 326. SDIO block diagram



По умолчанию для передачи данных используется SDIO_D0

Figure 327. SDIO adapter



Control unit – управление питанием и делитель clock'а для SDIO_CK.
Command path – отправляет команды и принимает ответы от карты. Состояние контролируется через CPSM (Command Path State Machine). Имеет встроенные модуля формирования команды, расчета CRC, обнаружения таймаута.

Data path – отправляет и принимает данные из карты. Состояние контролируется через DPSM (Data Path State Machine). Имеет встроенный модуль расчета CRC, обнаружения таймаута.
FIFO – буфер данных: 32-битный, глубиной 32 слова.

15.2. ФУНКЦИОНАЛЬНОЕ ОПИСАНИЕ КАРТ

а) Card identification mode.
Хост сбрасывает все карты, проверяет напряжение, идентифицирует карты и устанавливает относительные адреса для всех карт (RCA – relative card address).
б) Card reset.
Установка карты в состояние IDLE путем отсылки команды GO_IDLE_STATE.
в) Card identification process.
Начинается, когда SDIO_CMD шина переходит из третьего состояния в двухтактное состояние (push-pull).
Для SD-карт:
- шина команд активируется
- хост отправляет широковещательное сообщение SD_APP_OP_CMD
- карты отвечают хосту содержимым своих регистров состояний
- для не подходящих карт устанавливает неактивное состояние
- хост отправляет широковещательное сообщение ALL_SEND_CID всем активным картам
- карты отправляют обратно их уникальные идентификационные номера карт (CID – card ID) и входят в состояние Identification State
- хост устанавливает SET_RELATIVE_CID для всех активных карт. Новый адрес идентифицирует карту - RCA (relative card address), короче чем CID.
г) Защита карт:
- внутренняя защита карты на запись (ответственность карты)
- механическая внешняя защита карты на запись (ответственность хоста!!!)
- защита карты паролем

15.3. ДОПОЛНИТЕЛЬЫЕ СВЕДЕНИЯ

В модуле SDIO STM32F417 реализована функция Hardware Flow Control – аппаратное управление потоком. Основная причина – исключить переполнение внутренних буферов SDIO. Суть метода: остановка SDIO_CK (и остановка SDIO state machine), но SDIOCLK

шины APB2 по-прежнему работает, освобождая внутренние буферы модуля SDIO.

16 АЦП

16.1. ОБЩЕЕ ОПИСАНИЕ

АЦП использует метод последовательного приближения.
Суть метода последовательного приближения:
последовательное сравнение измеряемой величины с ½, ¼, 1/8 и т.д. от возможного максимального значения её.

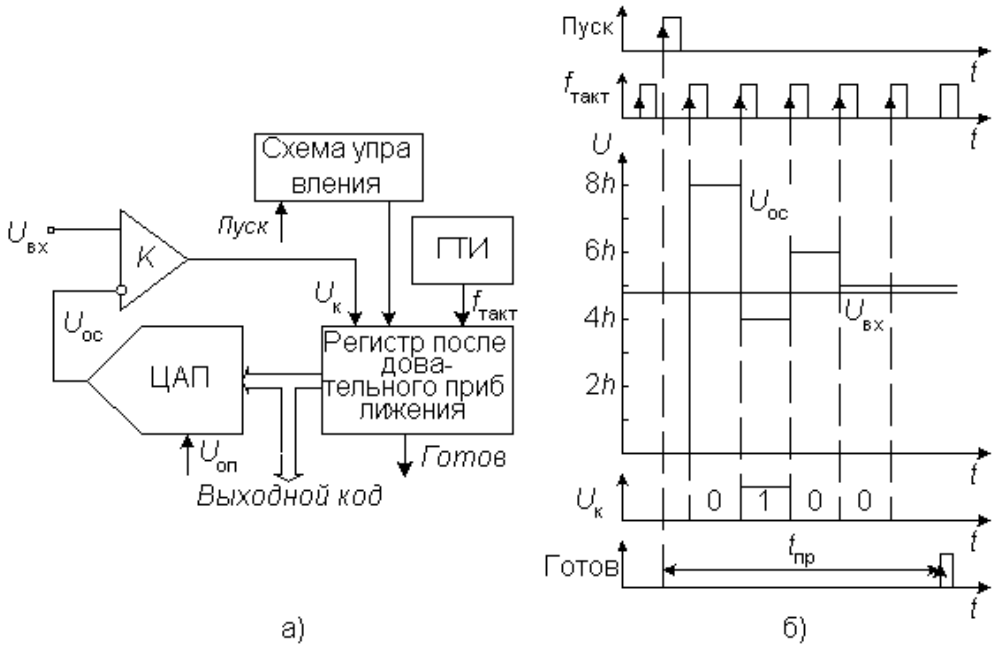


Рис. 9. Структурная схема и временные диаграммы АЦП последовательного приближения

АЦП м. б. работать в режимах:
- Single conversion (Одиночный – АЦП делает одно преобразование, выдает результат, АЦП останавливается)
- Continuous conversion (Продолжительный – преобразование АЦП выполняется одно за другим, постоянно, с выдачей результата)
- Scan mode (Сканирующий – по сути это тоже самое, что и continuous conversion, но идёт сканирование группы)
- Discontinuous mode (Непродолжительный режим –до N <= 8 преобразований. Например, N=3: преобразование каналов 1, 2, 3; затем каналов 4, 5, 6. И т. д.)

АЦП:
- 12-битный

- Результат сохраняется в 16-битном регистре
- 16 мультиплексированных каналов.
- организация преобразований в две группы:
    - regular (в группе до 16 каналов) (опрос всех каналов по очереди)
    - injected (в группе до 4 каналов) (более высокий приоритет, чем у regular, поэтому м. прерывать regular преобразование, по внешнему триггеру (например, событие таймера или внешний сигнал на пине)).
- Имеется внутренний температурный сенсор, подсоединенный к ADC1_IN16
- Имеется внутренний вольтметр питания МК, подсоединенный к ADC1_IN17
- Время преобразования (где 3 – sampling time – время необходимое для зарядки конденсатора. Его также возможно варьировать):
    12 бит: 3 + 12 = 15 циклов ADCCLK
    10 бит: 3 + 10 = 13 циклов ADCCLK
    8 бит: 3 + 8 = 11 циклов ADCCLK
    6 бит: 3 + 6 = 9 циклов ADCCLK

Figure 44. Single ADC block diagram