

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОСТОВСКОЙ ОБЛАСТИ  
ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
РОСТОВСКОЙ ОБЛАСТИ «САЛЬСКИЙ ИНДУСТРИАЛЬНЫЙ ТЕХНИКУМ»  
(ГБПОУ РО «СИТ»)

УТВЕРЖДАЮ:

Заместитель директора по учебной работе

 Т.В. Якимова

« \_\_\_\_ » \_\_\_\_\_ 2025 г.

**КУРС ЛЕКЦИЙ ПО ПРОФЕССИОНАЛЬНОМУ МОДУЛЮ**

**ПМ.02 ПРОЕКТИРОВАНИЕ УПРАВЛЯЮЩИХ ПРОГРАММ**

**КОМПЬЮТЕРНЫХ СИСТЕМ И КОМПЛЕКСОВ**

**МДК.02.02. ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ**

**ЧАСТЬ 1**

**(базовый уровень)**

**профиль обучения: технологический**

для специальности 09.02.01 Компьютерные системы и комплексы



г. Сальск

2025

Курс лекций по МДК.02.02. Программирование микроконтроллеров разработан на основе ФГОС СПО по специальности 09.02.01 Компьютерные системы и комплексы, утвержденного приказом Министерства просвещения Российской Федерации от 25 мая 2022 г. №362 (ред. от 03.07.2024), с учетом рабочей программы профессионального модуля «Проектирование управляющих программ компьютерных систем и комплексов».

Организация-разработчик: ГБПОУ РО «СИТ»

Разработчик: Халилова А.В., преподаватель ГБПОУ РО «СИТ»

Глазунов О.А., преподаватель ГБПОУ РО «СИТ»

Рекомендована (одобрена) цикловой комиссией общепрофессиональных дисциплин

Председатель \_\_\_\_\_ / Халилова А.В./

*подпись*

Протокол № 1 от « 29 » 08 2025 г.

## СОДЕРЖАНИЕ

МДК.02.02. Программирование микроконтроллеров .....	3
Тема 2.1. Архитектура микроконтроллеров STM32 .....	3
Лекция на тему «Архитектура микроконтроллеров STM32» .....	3
Лекция на тему «Специальные функции различных серий STM32» .....	6
Лекция на тему «Ядро Cortex и процессоры на базе Cortex-M» .....	26
Лекция на тему «Отладочная плата Nucleo. Основные характеристики». ....	34
Тема 2.2. Особенности программирования микроконтроллеров STM32 .....	43
Лекция на тему «Принципы построения программ для микроконтроллеров. Средства программирования и отладки микроконтроллеров» .....	43
Лекция на тему «Правила составления алгоритмов. Типы алгоритмов» .....	48
Лекция на тему «Диаграммы состояний. Конечный автомат» .....	52
Лекция на тему «Особенности синтаксиса для программ на МК» .....	57
Тема 2.3. Модульное программирование микроконтроллеров STM32 .....	61
Лекция на тему «Высокоуровневые библиотеки HAL. Синтаксис и шаблоны программ и программных модулей» .....	61
Лекция на тему «Среда программирования CubeIDE. Структура проекта» .....	67
Лекция на тему «Память МК. Работа с модулем МК в программе» .....	78
Лекция на тему «Подсистема ввода/вывода МК» .....	84
Лекция на тему «Последовательные интерфейсы МК» .....	91
Лекция на тему «Система прерываний МК» .....	101
Лекция на тему «Таймеры счетчики МК» .....	109
Лекция на тему «Модуль DMA» .....	120
Лекция на тему «Синхронные интерфейсы МК» .....	135
Лекция на тему «Режимы потребления МК» .....	142
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ .....	152

## **МДК.02.02. Программирование микроконтроллеров**

### **Тема 2.1. Архитектура микроконтроллеров STM32**

#### **Лекция на тему «Архитектура микроконтроллеров STM32»**

##### **Рассматриваемые вопросы:**

1. Введение в процессоры на базе ARM.

##### **Теоретический материал:**

##### **Введение в процессоры на базе ARM**

Под термином ARM в настоящее время мы понимаем как множество семейств архитектур с сокращенным набором команд (Reduced Instruction Set Computing, RISC), так и несколько семейств готовых ядер, являющихся строительными блоками (отсюда и термин ядро) процессоров, представленных многими производителями интегральных схем. При работе с процессорами на базе ARM может возникнуть много путаницы из-за того, что существует много разных версий архитектуры ARM (ARMv6, ARMv6-M, ARMv7-M, ARMv7-A и так далее) и многих архитектур ядра, которые в свою очередь основаны на версии архитектуры ARM. Для ясности, например, процессор на базе ядра Cortex-M4 разработан на архитектуре ARMv7-M.

Архитектура ARM представляет собой совокупность спецификаций, касающихся системы команд, модели выполнения, организации и распределения памяти, тактовых циклов на команду и т. д. Эти спецификации точно описывают машину, которая будет реализовывать указанную архитектуру. Если ваш компилятор способен генерировать ассемблерные инструкции для данной архитектуры, он может генерировать машинный код для всех тех реальных машин (то есть процессоров), реализующих эту архитектуру.

Cortex-M – это семейство физических ядер, предназначенных для дальнейшей интеграции с полупроводниковыми устройствами, определяемыми производителем, для формирования готового микроконтроллера. Принцип работы ядра определяется не только соответствующей архитектурой ARM (например, ARMv7-M), но и встроенными периферийными устройствами, а также аппаратными возможностями, заложенными производителем интегральных схем. Например, архитектура ядра Cortex-M4 разработана для поддержки операций доступа к битовым данным в двух определенных областях памяти с использованием функции, называемой битовыми лентами (bit-banding), но при фактической реализации такая функция добавляется или не

добавляется. STM32F4 – это семейство микроконтроллеров на базе ядра Cortex-M4, которое реализует технологию битовых лент. На рисунке 1 четко показана связь между микроконтроллером на базе Cortex-M3 и его ядром Cortex-M3.

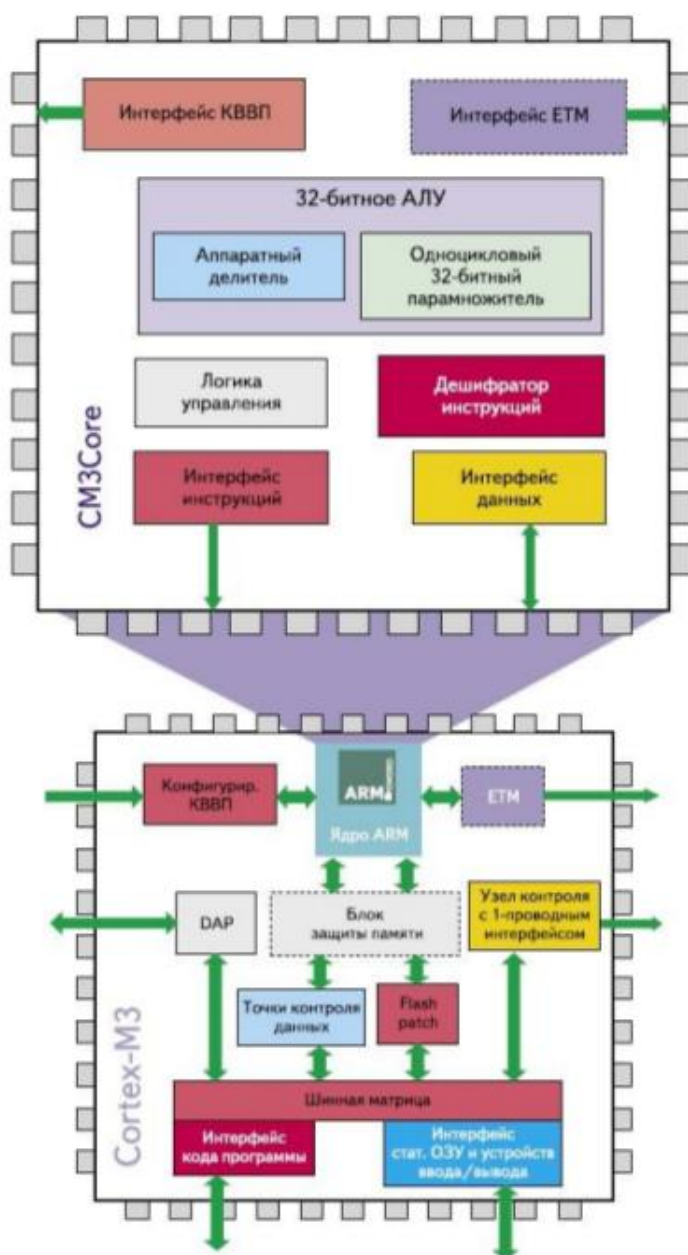


Рисунок 1 Соотношение между ядром Cortex-M3 и микроконтроллером на базе Cortex-M3

ARM Holdings – британская компания, которая разрабатывает систему команд и архитектуру для продуктов на базе ARM, но не производит устройства. Это довольно важный аспект мира ARM, и по этой причине существует множество производителей интегральных схем, которые разрабатывают, производят и продают микроконтроллеры на базе архитектур и ядер ARM. ST Microelectronics является одним из них, и в

настоящее время это единственный производитель, продающий полный ассортимент процессоров на базе Cortex-M.

ARM Holdings не производит и не продает процессорные устройства, основанные на собственных разработках, а скорее лицензирует архитектуру процессора для заинтересованных сторон. ARM предлагает разные условия лицензирования, различающиеся по стоимости и итоговому результату. Говоря о ядрах Cortex-M, также часто говорят о ядрах интеллектуальной собственности (Intellectual Property, IP), имея в виду макет конструкции чипа, который считается интеллектуальной собственностью одной из сторон, а именно ARM Holdings.

Благодаря данной бизнес-модели и довольно интересным функциям, таким как пониженное энергопотребление, низкая стоимость производства некоторых архитектур и т. д., ARM является наиболее широко используемой архитектурой системы команд с количественной точки зрения. Продукты на базе ARM стали чрезвычайно популярными. Многие крупносерийные и популярные 64-разрядные и многоядерные процессоры, используемые в устройствах и ставшие иконами в электронной промышленности (например, iPhone от Apple), основаны на архитектуре ARM (ARMv8-A).

Будучи своего рода широко распространенным стандартом, существует множество компиляторов и инструментов, а также операционных систем (Linux является наиболее используемой ОС на процессорах Cortex-A), которые поддерживают данные архитектуры, предлагая разработчикам множество возможностей для создания своих приложений.

## **Лекция на тему «Специальные функции различных серий STM32»**

### **Рассматриваемые вопросы:**

1. Краткий обзор подсемейств STM32.
2. Серия F0.
3. Серия F1.
4. Серия F2.
5. Серия F3.
6. Серия F4.
7. Серия F7.
8. Серия H7.
9. Серия L0.
10. Серия L1.
11. Серия L4.
12. Серия L4+.
13. Серия STM32WB.

### **Теоретический материал:**

#### **Краткий обзор подсемейств STM32**

STM32 представляет собой довольно сложную линейку продуктов, охватывающую более десяти подсемейств продуктов. Диаграммы объединяют подсемейства в четыре большие группы: High-performance, Mainstream, Wireless и Ultra Low-Power микроконтроллеры.

High-performance микроконтроллерами являются те микроконтроллеры STM32, которые предназначены для приложений с большим объемом вычислений и мультимедиа. Это микроконтроллеры на базе Cortex-M3/4F/7 с максимальными тактовыми частотами в диапазоне от 120МГц (F2) до 400МГц (H7). Все микроконтроллеры в данной группе поддерживают ускоритель ART<sup>™</sup> Accelerator – технологию ST, которая позволяет выполнять операции с Flash-памятью с состоянием 0-ожиданий (0-wait).


Mainstream микроконтроллеры разрабатываются для чувствительных к стоимости приложений, где стоимость микроконтроллера должна быть даже менее 1 \$/шт, а пространство является серьезным ограничением. В данной группе мы можем найти микроконтроллеры на базе Cortex-M0/3/4 с максимальными тактовыми частотами в диапазоне от 48 МГц (F0) до более 72 МГц (F1/F3).

Wireless микроконтроллеры – это новая линейка двухъядерных микроконтроллеров STM32 со встроенным 2,4 ГГц радиомодулем, подходящая для приложений с беспроводной сетью и Bluetooth-приложений. Данные микроконтроллеры имеют ядро Cortex-M0+ (называемое Сетевым процессором, англ. Network Processor), предназначенное для управления радиосвязью (сопутствующий стек BLE 5.0 также предоставляется ST), и программируемое пользователем ядро Cortex-M4 (называемое Прикладным процессором, англ. Application Processor) для основного встроенного приложения.

Группа Ultra Low-Power включает в себя семейства микроконтроллеров STM32, предназначенные для приложений с пониженным энергопотреблением, использующихся в устройствах с батарейным питанием, которым необходимо снизить общее энергопотребление до низкого уровня, обеспечивая более длительный срок службы батареи. В данной группе мы можем найти как микроконтроллеры на базе Cortex-M0+ для чувствительных к стоимости приложений, так и микроконтроллеры на базе Cortex-M4F с Динамическим изменением напряжения (Dynamic Voltage Scaling, DVS) – технологией, которая позволяет оптимизировать внутреннее напряжение процессора в соответствии с его частотой.

## Серия F0

Таблица 1 Возможности STM32F0

Common to all STM32F0		Core: Cortex-M0 Instruction set: Thumb subset, Thumb-2 subset Internal RC oscillators: HSI=8MHz, LSI=40KHz External clocks: HSE=4 - 32MHz, LSE=32.768 - 1000 KHz Maximum Core Frequency: 48MHz Low power modes: Sleep, Stop and Standby Year of commercialization: 2012 Available Packages: LQFP(32,48,64,100), TSSOP20, UFBGA(64,100), UQFPN(28,32,48), WLCSP(25,36,49,64)										
		Product Line	FLASH (KB)	RAM (KB)	Operating Voltage	Backup Memory	DAC	Touch Sense	Up to 2xSPI/I <sup>2</sup> S, 2xI <sup>2</sup> C	USART	CAN	USB 2.0
		STM32F0x0 Value Line	16 to 256	4 to 32	2.4 to 3.6 V				*	6		*
		STM32F0x1 Access Line	16 to 256	4 to 32	2.0 to 3.6 V	*	*	*	*	8	*	
		STM32F0x2 USB Line	16 to 128	4 to 32	2.0 to 3.6 V	*	*	*	*	8	*	*
		STM32F0x8 Low-Voltage Line	32 to 256	4 to 32	1.8 V ±8%	*	*	*	*	8	*	*
MAINSTREAM	<ul style="list-style-type: none"><li>•Reset POR/PDR</li><li>•2xWDT</li><li>•Hardware CRC</li><li>•Crystal oscillators</li><li>•PLL</li><li>•RTC calendar/clock</li><li>•16/32-bit timers</li><li>•1x12-bit ADC</li><li>•Up to 12-channel DMA</li><li>•Temperature sensor</li><li>•Multiple channel DMA</li><li>•SWD</li><li>•Unique ID</li></ul>											

Серия STM32F0 — это знаменитые 32-разрядные за 32 цента линейки микроконтроллеров из ассортимента STM32. Они рассчитаны на рыночную цену, способную конкурировать с 8/16-разрядными микроконтроллерами других производителей, предлагая более продвинутую и мощную платформу.

Наиболее важные особенности данной серии:



1. Ядро:
  - Ядро ARM Cortex-M0 с максимальной тактовой частотой 48 МГц.
  - В дополнение к Cortex-M0 включают в себя таймер SysTick.
2. Память:
  - Статическое ОЗУ (SRAM) от 4 до 32 КБ.
  - Flash-память от 16 до 256 КБ.
  - Каждый чип имеет запрограммированный на заводе 96-разрядный уникальный идентификационный номер устройства.
3. Периферийные устройства:
  - Каждое устройство серии F0 имеет ряд периферийных устройств, которые разнятся от одной линейки к другой (краткий обзор см. в таблице 4).
4. Генераторы состоят из внутреннего RC (8 МГц, 40 кГц), дополнительного внешнего HSE (от 4 до 32 МГц), LSE (от 32,768 до 1000 кГц).
5. Корпусы ИС: LQFP, TSSOP2020 , UFBGA, UFQFPN, WLCSP (детали см. в таблице 4).
6. Диапазон рабочего напряжения от 2,0В до 3,6В с возможностью понижения до 1,8В  $\pm$  8%.

## Серия F1

Таблица 2 Возможности STM32F1

Common to all STM32F1		Core: Cortex-M3 Instruction set: Thumb, Thumb-2, Saturated Math Internal RC oscillators: HSI=8MHz, LSI=40KHz External clocks: HSE=4-24Mhz(F100), 4-16Mhz(F101/2/3), 3-25MHz (F105/7), LSE=32.768 - 1000 KHz Low power modes: Sleep, Stop and Standby Year of commercialization: 2007 Available Packages: LFBGA(100,144), LQFP(48,64,100,144), UFBGA(100), UFQFPN(36,48), WLCSP(64)											
		Product Line	FLASH (KB)	RAM (KB)	F <sub>clk</sub> (MHz)	USB 2.0 FS	USB 2.0 OTG FS	FSMC	3-phase MC Timer	I <sup>2</sup> S	CAN 2.0B	SDIO	Ethernet
		STM32F100 Value Line	16 to 512	4 to 32	24			*	*				
		STM32F101 Access Line	16 to 1024	4 to 80	36			*					
		STM32F102 USB Line	16 to 128	4 to 16	48	*							
		STM32F103 Performance Line	16 to 1024	6 to 96	72	*		*	*	*	*	*	
		STM32F105 STM32F107 Connectivity Line	64 to 256	64	72		*	*	*	*	*		*

MAINSTREAM	<ul style="list-style-type: none"><li>• -40 to +105° range</li><li>• USART, SPI, I<sup>2</sup>C</li><li>• 16/32-bit timers</li><li>• Temperature sensor</li><li>• Up to 3x12-bit DAC</li><li>• Dual 12-bit ADC</li><li>• Up to 12-channels DMA</li><li>• Low voltage 2.0 to 3.6V</li><li>• 5V tolerant I/Os</li><li>• Up to 80 fast I/Os</li><li>• Reset POR/PDR</li><li>• 2xWDT</li><li>• Hardware CRC</li><li>• Backup Memory</li><li>• RTC calendar/clock</li><li>• SWD</li><li>• Unique ID</li></ul>												
------------	--	--	--	--	--	--	--	--	--	--	--	--	--

Серия STM32F1 была первыми микроконтроллерами на базе ARM от ST. Представленные на рынке в 2007 году, они по-прежнему являются самыми распространенными микроконтроллерами из ассортимента STM32. На рынке доступно множество отладочных плат, выпускаемых ST и другими производителями, и вы

найдете в Интернете множество примеров для микроконтроллеров F1. Если вы новичок в мире STM32, возможно, линейка F1 – лучший выбор для начала работы с данной платформой.

Серия F1 развивалась с течением времени за счет увеличения скорости, объема внутренней памяти, разнообразных периферийных устройств. Существует пять линеек F1: Connectivity (STM32F105/107), Performance (STM32F103), USB Access (STM32F102), Access (STM32F101), Value (STM32F100).

Наиболее важные особенности данной серии:

1. Ядро:

– Ядро ARM Cortex-M3 с максимальной тактовой частотой от 24 до 72 МГц.

2. Память:

– Статическое ОЗУ от 4 до 96 КБ.

– Flash-память от 16 до 256 КБ.

– Каждый чип имеет запрограммированный на заводе 96-разрядный уникальный идентификационный номер устройства.

3. Периферийные устройства:

– Каждое устройство серии F1 имеет ряд периферийных устройств, которые разнятся от одной линейки к другой.

4. Генераторы состоят из внутреннего RC (8 МГц, 40 кГц), дополнительного внешнего HSE (4-24 МГц (F100), 4-16 МГц (F101/2/3), 3-25 МГц (F105/7), LSE (32,768 – 1000 кГц).

5. Корпусы ИС: LFBGA, LQFP, UFBGA, UFQFPN, WLCSP.

6. Диапазон рабочего напряжения от 2,0В до 3,6В

7. Несколько вариантов обмена данными, включая Ethernet, CAN и USB 2.0 OTG.

## **Серия F2**

Таблица 3 Возможности STM32F2

Common to all STM32F2		Core: Cortex-M3 with ART™ Accelerator Instruction set: Thumb, Thumb-2, Saturated Math Internal RC oscillators: HSI=16MHz, LSI=32KHz External clocks: HSE=1 - 26MHz, LSE=32.768 - 1000 KHz Maximum Core Frequency: 120MHz Low power modes: Sleep, Stop and Standby Year of commercialization: 2010 Available Packages: BGA(176), LQFP(64,100,144,176), UFBGA(100), WLCSP(66)								
		Product Line	FLASH (KB)	RAM (KB)	Hardware Crypto/Hash	USB 2.0 OTG FS	FSMC	Camera I/F	SDIO	Ethernet
		STM32F205 STM32F215	128 to 1024	Up to 128	*	*	*	*	*	*
		STM32F207 STM32F217	512 to 1024	Up to 128	*	*	*	*	*	*

HIGH PERFORMANCE	<ul style="list-style-type: none"><li>• -40 to +105° range</li><li>• USART, SPI, I²C</li><li>• 16/32-bit timers</li><li>• Temperature sensor</li><li>• 12-bit DAC up to 24 channels</li><li>• Dual 12-bit ADC</li><li>• 16-channels DMA</li><li>• Low voltage 1.8 to 3.6V</li><li>• 5V tolerant I/Os</li><li>• Up to 136 fast I/Os</li><li>• Reset POR/PDR</li><li>• 2xWDT</li><li>• Hardware CRC</li><li>• Backup Memory</li><li>• RTC calendar/clock</li><li>• SWD</li><li>• Unique ID</li></ul>		Core: Cortex-M3 with ART™ Accelerator Instruction set: Thumb, Thumb-2, Saturated Math Internal RC oscillators: HSI=16MHz, LSI=32KHz External clocks: HSE=1 - 26MHz, LSE=32.768 - 1000 KHz Maximum Core Frequency: 120MHz Low power modes: Sleep, Stop and Standby Year of commercialization: 2010 Available Packages: BGA(176), LQFP(64,100,144,176), UFBGA(100), WLCSP(66)								
			Product Line	FLASH (KB)	RAM (KB)	Hardware Crypto/Hash	USB 2.0 OTG FS	FSMC	Camera I/F	SDIO	Ethernet
			STM32F205 STM32F215	128 to 1024	Up to 128	*	*	*	*	*	*
			STM32F207 STM32F217	512 to 1024	Up to 128	*	*	*	*	*	*

Серия STM32F2 микроконтроллеров STM32 является экономически эффективным решением в сегменте High-performance. Это самые новые и самые быстрые микроконтроллеры на базе Cortex-M3 с эксклюзивным ускорителем ART™ Accelerator от ST. F2 совместим с выводами серии STM32 F4. STM32F2 был микроконтроллером, выбранным разработчиками популярных часов Pebble для своих первых умных часов.



Рисунок 1 Первые часы Pebble с микроконтроллером STM32F205 внутри

Наиболее важные особенности данной серии:

- Ядро:
  - Ядро ARM Cortex-M3 с максимальной тактовой частотой 120 МГц.
- Память:
  - Статическое ОЗУ от 64 до 128 КБ. 4 КБ с батарейным питанием, 80 Байт с батарейным питанием и стиранием при обнаружении несанкционированного доступа.
  - Flash-память от 128 до 1024 КБ.
  - Каждый чип имеет запрограммированный на заводе 96-разрядный уникальный идентификационный номер устройства.
- Периферийные устройства:
  - Каждое устройство серии F2 имеет ряд периферийных устройств, которые разнятся от одной линейки к другой.

- Генераторы состоят из внутреннего RC (16 МГц, 32 кГц), дополнительного внешнего HSE (от 1 до 26 МГц), LSE (от 32,768 до 1000 кГц).
- Корпусы ИС: BGA, LQFP, UFBGA, WLCSP.
- Диапазон рабочего напряжения от 1,8В до 3,6В.

### Серия F3

Таблица 4 Возможности STM32F3

Common to all STM32F3



Core: Cortex-M4F  
Instruction set: Thumb, Thumb-2, Saturated Math, DSP, FPU  
Internal RC oscillators: HSI=8MHz, LSI=40KHz  
External clocks: HSE=4-32MHz, LSE=32.768 - 1000 KHz  
Maximum Core Frequency: 72MHz  
Low power modes: Sleep, Stop and Standby  
Year of commercialization: 2012  
Available Packages: LQFP(32,48,64,100,144), UFBGA(100), UFQFPN(32), WLCSP(49,66,100)

Product Line	FLASH (KB)	RAM (KB)	CCM SRAM	ADC 12-bit 16-bit	12-bit DAC	Fast Comparator	OpAmp (PGA)	Advanced 16-bit timer	Hig resolution Timer
STM32F301	32 to 64	16		Up to 2	1	3	1	1	
STM32F302 - USB & CAN	32 to 512	16 to 64		Up to 2	1	Up to 4	Up to 2	1	
STM32F303 - Performance	32 to 512	16 to 80	•	Up to 4	Up to 3	Up to 7	Up to 4	Up to 3	
STM32F3x4 Digital Power	32 to 512	16	•	2	3	2x Ultra fast	1	1	• 10 ch
STM32F373 Precision measurement	16 to 64	32		1 3	3	2			
STM32F3x8 1.8V ±8%	64 to 512	16 to 64	•	Up to 4	Up to 3	Up to 7	Up to 4	Up to 3	

MAINSTREAM

STM32F3 – это самая мощная серия микроконтроллеров в сегменте Mainstream, основанная на ядре ARM Cortex-M4F. Она спроектирована так, чтобы быть практически совместимой с выводами серии STM32F1, несмотря на то, что она не предоставляет такое же разнообразие периферийных устройств. STM32F3 был микроконтроллером, выбранным разработчиками игрушки BB-8 droid от Sphero.



Рисунок 2 BB-8 droid с микроконтроллером STM32F3

Отличительной особенностью данной серии является наличие встроенных аналоговых периферийных устройств, что ведет к снижению затрат на уровне приложения и упрощает разработку приложений, в том числе:

- Сверхбыстрые компараторы (25 нс).
- Операционный усилитель с программируемым усилением.
- 12-разрядные ЦАП.
- Сверхбыстрые 12-разрядные АЦП с 5 МВыборок/с (миллион выборок в секунду) на канал (до 18 МВыборок/с в режиме чередования (Interleaved mode)).
- Точные 16-разрядные сигма-дельта АЦП (21 канал).
- 16-разрядный таймер расширенного управления с широтно-импульсной модуляцией в 144 МГц (разрешение < 7 нс) для приложений управления; таймер высокого разрешения (217 пикосекунд), самокомпенсация против дрейфов источника питания и температурного.

Еще одной интересной особенностью данной серии является наличие памяти, связанной с ядром (Core Coupled Memory, CCM) – специфической архитектуры памяти, которая связывает некоторые области памяти с ядром ЦПУ, позволяя добиться состояния 0-ожиданий. Она может быть использована для ускорения выполнения критичных ко времени процедур, повышая производительность до 40%. Например, процедуры ОС для переключения контекста могут быть сохранены в данной области для ускорения действий ОСРВ.


Наиболее важные особенности данной серии:

- Ядро:
  - Ядро ARM Cortex-M4F с максимальной тактовой частотой 72 МГц.
- Память:
  - SRAM от 16 до 80КБ общего назначения с аппаратным контролем четности. 64 / 128 Байт с батарейным питанием и стиранием при обнаружении несанкционированного доступа.
  - До 8 КБ Core Coupled Memory (CCM) с аппаратным контролем четности.
  - Flash-память от 32 до 512 КБ.
  - Каждый чип имеет запрограммированный на заводе 96-разрядный уникальный идентификационный номер устройства.
- Периферийные устройства:
  - Каждое устройство серии F3 имеет ряд периферийных устройств, которые разнятся от одной линейки к другой.

- Генераторы состоят из внутреннего RC (8 МГц, 40 кГц), дополнительного внешнего HSE (от 4 до 32 МГц), LSE (от 32,768 до 1000 кГц).
- Корпусы ИС: LQFP, UFBGA, UFQFPN, WLCSP.
- Диапазон рабочего напряжения составляет от  $1,8V \pm 8\%$  до 3,6V.

### Серия F4

Таблица 5 Возможности STM32F4

Common to all STM32F4		Core: Cortex-M4F with ART™ Accelerator Instruction set: Thumb, Thumb-2, Saturated Math, DSP, FPU Internal RC oscillators: HSI=16MHz, LSI=32KHz External clocks: HSE=4-26MHz, LSE=32.768KHz Low power modes: Sleep, Stop and Standby Year of commercialization: 2011 Available Packages: LQFP(64,100,144,176,208), TFBGA(216), UFBGA(64,100,144,169,176), UFQFPN(48), WLCSP(36,49,64,81,90,143,168)									
		Product Line	FLASH (KB)	RAM (KB)	F <sub>core</sub> (MHz)	Ethernet I/F	Camera I/F	SDRAM I/F	SAI <sup>1</sup> I/F	Chrom-ART™	TFT Controller
		Advanced lines									
		STM32F469	512 to 2048	384	180	*	*	*	*	*	*
		STM32F429	512 to 2048	256	180	*	*	*	*	*	*
		STM32F427	1024 to 2048	256	180	*	*	*	*	*	*
		Foundation lines									
		STM32F446	256 to 512	128	180	*	*	*	*	*	*
		STM32F407	512 to 1024	192	168	*	*	*	*	*	*
		STM32F405	512 to 1024	192	168	*	*	*	*	*	*
HIGH PERFORMANCE		Product Line	FLASH (KB)	RAM (KB)	F <sub>core</sub> (MHz)	Dynamic Efficiency	Run Current (µA/MHz)	STOP Current (µA)	QSPI	CAN 2.0B	USB 2.0 OTG FS
		Access lines									
		STM32F401	128 to 512	96	84	*	Down to 128	Down to 10	*	*	*
		STM32F410	64 to 128	32	100	*	Down to 89	Down to 6	*	*	*
		STM32F411	256 to 512	128	100	*	Down to 100	Down to 12	*	*	*
		STM32F412	512 to 1024	256	100	*	Down to 112	Down to 18	*	*	*
		STM32F413	1024 to 1536	320	100	*	Down to 115	Down to 18	*	*	*

Серия STM32F4 – самая распространенная группа микроконтроллеров на базе CortexM4F в сегменте High-performance. Серия F4 также является первой серией STM32 с инструкциями DSP и форматом с плавающей запятой одинарной точности. F4 совместима с выводами серии STM32F2, добавляя более высокую тактовую частоту, 64 КБ статического ОЗУ типа CCM, полнодуплексный I<sup>2</sup>S, улучшенные часы реального времени и более быстрые АЦП. Серия STM32F4 также предназначена для мультимедийных приложений, а некоторые микроконтроллеры предлагают специальную поддержку LCD-TFT.

Наиболее важные особенности данной серии:

- Ядро:




- Ядро ARM Cortex-M4F с максимальной тактовой частотой от 84 до 180 МГц.
- Память:
  - Статическое ОЗУ от 128 до 384 КБ.
  - 4 КБ с батарейным питанием, 80 Байт с батарейным питанием и стиранием при обнаружении несанкционированного доступа.
  - 64 КБ Core Coupled Memory (CCM).
  - Flash-память от 256 до 2048 КБ.
  - Каждый чип имеет запрограммированный на заводе 96-разрядный уникальный идентификационный номер устройства.
- Периферийные устройства:
  - Каждое устройство серии F4 имеет ряд периферийных устройств, которые разнятся от одной линейки к другой.
  - Генераторы состоят из внутреннего RC (16 МГц, 32 кГц), дополнительного внешнего HSE (от 4 до 26 МГц), LSE (от 32,768 до 1000 кГц).
  - Корпусы ИС: BGA, LQFP, TFBGA, UFBGA, UFQFPN, WLCSP.
  - Диапазон рабочего напряжения от 1,8В до 3,6 В.

### Серия F7

Таблица 6 Возможности STM32F7

Common to all STM32F7



Core: Cortex-M7 with ART™ Accelerator  
Instruction set: Thumb, Thumb-2, Saturated Math, DSP, FPU, SIMD  
Internal RC oscillators: HSI=16MHz, LSI=40KHz  
External clocks: HSE=4-26MHz, LSE=32.768KHz  
Maximum Core Frequency: 216MHz  
Low power modes: Sleep, Stop and Standby  
Year of commercialization: 2015  
Available Packages: LQFP(64,100,144,176,208), TFBGA(100,216), UFBGA(144,176), WLCSP(100,143,180)

Product Line	FLASH (KB)	RAM (KB)	L1 Cache (I/D)	FPU	Ethernet	FMC	CAN	JPEG Codec	TFT Controller
Advanced lines									
STM32F7x9	1024 to 2048	512	16K+16K	Double Precision	*	*	3	*	*
STM32F7x8	1024 to 2048	512	16K+16K	Double Precision	*	*	3	*	*
STM32F7x7	1024 to 2048	512	16K+16K	Double Precision	*	*	3	*	*
STM32F7x6	512 to 1024	320	4k+4k	Single Precision	*	*	2		*
STM32F765	1024 to 2048	512	16K+16K	Double Precision	*	*	3		
STM32F745	512 to 1024	320	4k+4k	Single Precision	*	*	2		
Foundation lines									
Product Line	FLASH (KB)	RAM (KB)	L1 Cache (I/D)	FPU	PC-RDP	FMC	CAN	USB PHY	TFT Controller
STM32F7x3	512 to 1024	256	8k+8k	Single Precision	*	*	1	*	
STM32F7x2	512 to 1024	256	8k+8k	Single Precision	*	*	1		

- Chrom-ART™ Accelerator
- USART, SPI, I²C
- I²S + audio PLL
- 2xSAI¹
- SDIO
- 2xCAN
- 2xUSB OTG FS/HS
- HDMI-CEC
- Ethernet
- 16/32-bit timers
- Temperature sensor
- Up to 2x12-bit DAC
- Up to 3x12-bit ADC
- 16-channels DMA
- Low voltage 1.7 to 3.6V
- 5V tolerant I/Os
- Up to 164 fast I/Os
- Reset POR/PDR
- 2xWDT
- Hardware CRC
- Backup Memory
- RTC calendar/clock
- SWD
- Unique ID

HIGH PERFORMANCE

Таблица 9: Возможности STM32F7

Серия STM32F7 – это новейшие сверхвысокопроизводительные микроконтроллеры в сегменте High-performance, и это были первые микроконтроллеры

на базе Cortex-M7, представленные на рынке. Благодаря ускорителю ART™ Accelerator от ST и кэш-памяти L1 устройства STM32F7 обеспечивают максимальную теоретическую производительность Cortex-M7 независимо от кода, выполняемого из встроенной Flash-памяти или внешней памяти: 1082 CoreMark/462 DMIPS на частоте 216 МГц. STM32F7 явно ориентирован на тяжелые мультимедийные приложения. Благодаря программе долговечности STM32 (10 лет) можно разрабатывать мощные встроенные приложения, не беспокоясь о доступности микроконтроллеров на рынке в далеком будущем. Cortex-M7 обратно совместим с системой команд Cortex-M4, а серия STM32F7 совместима с выводами серии STM32F4.


Наиболее важные особенности данной серии:


- Ядро:
  - Ядро ARM Cortex-M7 с максимальной тактовой частотой 216 МГц.
- Память:
  - Статическое ОЗУ до 512 КБ с рассеянной архитектурой (scattered architecture).
  - Кэш L1 (I/D до 16 КБ + 16 КБ).
  - Flash-память от 512 до 2048 КБ.
  - Каждый чип имеет запрограммированный на заводе 96-разрядный уникальный идентификационный номер устройства.
- Периферийные устройства:
  - Каждое устройство серии F4 имеет ряд периферийных устройств, которые разнятся от одной линейки к другой.
- Генераторы состоят из внутреннего RC (16 МГц, 32 кГц), дополнительного внешнего HSE (от 4 до 26 МГц), LSE (от 32,768 до 1000 кГц).
- Корпусы ИС: LQFP, TFBGA, UFBGA, WLCSP.
- Диапазон рабочего напряжения от 1,7В до 3,6 В.

### **Серия H7**

Таблица 7 Возможности STM32H7



Common to all STM32H7		Core: Cortex-M7 with ART™ Accelerator Instruction set: Thumb, Thumb-2, Saturated Math, DSP, FPU, SIMD Internal RC oscillators: HSI=16MHz, LSI=40KHz External clocks: HSE=4-26Mhz, LSE=32.768KHz Maximum Core Frequency: 400MHz Low power modes: Sleep, Stop and Standby Year of commercialization: 2017 Available Packages: LQFP(100,144,176,208), TFBGA(240), UFBGA(176), XQFPFN0(168)							
		Product Line	FLASH (KB)	RAM (KB)	Ethernet I/F	Camera I/F	FMC	JPEG Codec	TFT Controller
		STM32F7x7	Available in late 2017						
		STM32F7x5							
		STM32H7x3	2048	1024	*	*	*	*	*

HIGH PERFORMANCE		Core: Cortex-M7 with ART™ Accelerator Instruction set: Thumb, Thumb-2, Saturated Math, DSP, FPU, SIMD Internal RC oscillators: HSI=16MHz, LSI=40KHz External clocks: HSE=4-26Mhz, LSE=32.768KHz Maximum Core Frequency: 400MHz Low power modes: Sleep, Stop and Standby Year of commercialization: 2017 Available Packages: LQFP(100,144,176,208), TFBGA(240), UFBGA(176), XQFPFN0(168)							
		Product Line	FLASH (KB)	RAM (KB)	Ethernet I/F	Camera I/F	FMC	JPEG Codec	TFT Controller
		STM32F7x7	Available in late 2017						
		STM32F7x5							
		STM32H7x3	2048	1024	*	*	*	*	*

В октябре 2016 года ST объявила о новом семействе микроконтроллеров STM32: STM32H7. Это Cortex-M7, созданный по 40-нм техпроцессу, способный работать до 400 МГц. Он также обеспечивает 1 МБ SRAM с той же рассеянной архитектурой, что и в серии STM32F7. По мнению автора, данное семейство микроконтроллеров STM32 откроет двери для двухъядерных микроконтроллеров STM32, благодаря 40-нм процессу производства.

Характеристики серии STM32H7:

- Ядро:
  - Ядро ARM Cortex-M7 с максимальной тактовой частотой 400 МГц.
- Память:
  - Статическое ОЗУ до 1024 КБ с рассеянной архитектурой.
  - Кэш L1 (I/D до 16 КБ + 16 КБ).
  - Flash-память от 512 до 2048 КБ.
  - Каждый чип имеет запрограммированный на заводе 96-разрядный уникальный идентификационный номер устройства.
- Периферийные устройства:
  - Несколько новых периферийных устройств, таких как 14-разрядный АЦП и новый SAI.
- Корпусы ИС: LQFP, TFBGA
- Совместимость с выводами серии STM32F7.

### Серия L0

Таблица 8 Возможности STM32L0

<div>Common to all STM32L0</div> <div>LOW POWER</div>	<ul style="list-style-type: none"> <li>• -40 +125°C range</li> <li>• Low voltage 1.65 to 3.6V</li> <li>• Dynamic Voltage Scaling</li> <li>• 5 clock sources</li> <li>• USART, SPI, I<sup>2</sup>C</li> <li>• 16-bit timers</li> <li>• Temperature sensor</li> <li>• DMA</li> <li>• 5V tolerant I/Os</li> <li>• Up to 51 fast I/Os</li> <li>• Reset POR/PDR</li> <li>• 2xWDT</li> <li>• Hardware CRC</li> <li>• Backup Memory</li> <li>• RTC calendar/clock</li> <li>• SWD</li> <li>• Unique ID</li> </ul>		Core: Cortex-M0+ with MPU Instruction set: Thumb subset, Thumb-2 subset Internal RC oscillators: HSI=16MHz, LSI=37KHz External clocks: HSE=1-24MHz, LSE=32.768KHz Maximum Core Frequency: 32MHz Low power modes: Low-power run, Sleep, Low-power sleep, Stop with RTC, stop without RTC, Standby with RTC and Standby without RTC Year of commercialization: 2014 Available Packages: LQFP(32,48,64), TFBGA(64), UFQFPN(32), WLCSP(36)										
		Product Line	FLASH (KB)	RAM (KB)	EEPROM (KB)	12-bit ADC	Low Power UART	Low Power 16-bit timer	12-bit DAC	Touch Sense	True RNG	USB 2.0 Crystal-less	Segment LCD Driver
		STM32L0x1 Access	Up to 64	8	2	*	*	*					
		STM32L0x2 USB	Up to 64	8	2	*	*	*	*	*	*	*	
		STM32L0x3 USB & LCD	Up to 64	8	2	*	*	*	*	*	*	*	Up to 8x28 or 4x32

Серия STM32L0 является экономически эффективным решением сегмента Ultra LowPower. Комбинация ядра ARM Cortex-M0+ и функций сверхнизкого энергопотребления делает STM32L0 наиболее подходящей для приложений, работающих от батареи или от аккумулятора, предлагая самое низкое в мире потребление энергии при 125°C. STM32L0 предоставляет динамическое изменение напряжения, тактовый генератор со сверхнизким энергопотреблением, интерфейс ЖК-дисплея, компаратор, ЦАП и аппаратное шифрование.

Текущие значения потребления:

- Динамический рабочий режим: до 87 мкА/МГц.
- Режим сверхнизкого энергопотребления + полное ОЗУ + таймер с пониженным энергопотреблением: 440 нА (16 линий пробуждения).
- Режим сверхнизкого энергопотребления + резервный регистр: 250 нА (3 линии пробуждения).
- Время пробуждения: 3,5 мкс.

Наиболее важные особенности данной серии:

- Ядро:
  - Ядро ARM Cortex-M0+ с максимальной тактовой частотой 32 МГц.
- Память:
  - 8 КБ статического ОЗУ.
  - 20 Байт с батарейным питанием и стиранием при обнаружении несанкционированного доступа.
  - Flash-память от 32 до 64 КБ.
  - EEPROM до 2 КБ (вместе с Error Code Correction (ECC))
  - Каждый чип имеет запрограммированный на заводе 96-разрядный уникальный идентификационный номер устройства.

- Периферийные устройства:
  - Каждое устройство серии L0 имеет ряд периферийных устройств, которые разнятся от одной линейки к другой.
- Генераторы состоят из внутреннего RC (16 МГц, 37 кГц), дополнительного внешнего HSE (от 1 до 24 МГц), LSE (32,768 кГц).
- Корпусы ИС: LQFP, TFBGA, UFQFPN, WLCSP.
- Диапазон рабочего напряжения от 1,65В до 3,6В.

### Серия L1

Таблица 9 Возможности STM32L1

Common to all STM32L1		Core: Cortex-M3 Instruction set: Thumb, Thumb-2, Saturated Math Internal RC oscillators: HSI=16MHz, LSI=37KHz External clocks: HSE=1-24Mhz, LSE=32.768KHz Maximum Core Frequency: 32MHz Low power modes: Low-power run, Sleep, Low-power sleep, Stop with RTC, stop without RTC, Standby with RTC and Standby without RTC Year of commercialization: 2010 Available Packages: LQFP(48,64,100,144), TFBGA(64), UFBGA(100,132), UQFPN(48), WLCSP(63,64,104)									
		Product Line	FLASH (KB)	RAM (KB)	EEPROM (KB)	Memory I/F	OpAmp	Temperature Sensor	AES 128-bit	Touch Sense	Segment LCD Driver
		STM32L100 Value line	32 to 256	4 to 16	2						Up to 8x28
		STM32L151 STM32L152	32 to 512	16 to 80	4 to 16	SDIO FSMC	•	•		•	Up to 8x28
		STM32L162	256 to 512	8 to 16	8 to 16	SDIO FSMC	•	•	•	•	Up to 8x40

Серия STM32L1 – это решение среднего класса сегмента Ultra Low-Power. Сочетание ядра ARM Cortex-M3 с FPU и функциями сверхнизкого энергопотребления делает STM32L1 оптимальным для приложений, работающих от батареи, которые также требуют достаточной вычислительной мощности. Как и серия L0, STM32L1 предоставляет динамическое изменение напряжения, тактовый генератор со сверхнизким энергопотреблением, интерфейс ЖК-дисплея, компаратор, ЦАП и аппаратное шифрование.

Текущие значения потребления:

- Режим сверхнизкого энергопотребления: 280 нА с резервными регистрами (3 вывода для пробуждения)
- Режим сверхнизкого энергопотребления + RTC: 900 нА с резервными регистрами (3 вывода для пробуждения)
- Рабочий режим с пониженным энергопотреблением: до 9 мкА
- Динамический рабочий режим: до 177 мкА/МГц STM32L1 совместим с выводами нескольких микроконтроллеров из серии STM32F.

Наиболее важные особенности данной серии:

- Ядро:

- Ядро ARM Cortex-M3 с FPU с максимальной тактовой частотой 32 МГц.

- Память:

- Статическое ОЗУ от 4 до 80 КБ.

20 Байт с батарейным питанием и стиранием при обнаружении несанкционированного доступа.

- Flash-память от 32 до 512 КБ.

- EEPROM до 2 КБ (вместе с ECC).

- Каждый чип имеет запрограммированный на заводе 96-разрядный уникальный идентификационный номер устройства.

- Периферийные устройства:

- Каждое устройство серии L1 имеет ряд периферийных устройств, которые разнятся от одной линейки к другой.

- Генераторы состоят из внутреннего RC (16 МГц, 37 кГц), дополнительного внешнего HSE (от 1 до 24 МГц), LSE (32,768 кГц).

- Корпусы ИС: LQFP, TFBGA, UFBGA, UFQFPN, WLCSP

- Диапазон рабочего напряжения от 1,65 до 3,6 В, включая программируемый детектор провала напряжения (brownout detector).

### **Серия L4**

Таблица 10 Возможности STM32L4

Common to all STM32L4		Core: Cortex-M4F with ART™ Accelerator Instruction set: Thumb, Thumb-2, DSP, FPU Internal RC oscillators: HSI=16MHz, LSI=37KHz External clocks: HSE=1-24MHz, LSE=32.768KHz Maximum Core Frequency: 80MHz Low power modes: Low-power run, Sleep, Low-power sleep, Stop with RTC, stop without RTC, Standby with RTC and Standby without RTC Year of commercialization: 2015 Available Packages: LQFP(64,100,144), UFBGA(132), WLCSP(72,81)									
		Product Line	FLASH (KB)	RAM (KB)	Memory I/F	Op-Amp	CAN	12-bit ADC 3Msp/s	USB 2.0 FS Crystall-less	Segment LCD Driver	Chrom-ART
		STM32L4x6 - USB OTG + Segment LCD lines									
		STM32L496	512 to 1024	320	•	2	2	3	•	Up to 8x40	•
		STM32L476	256 to 1024	128	•	2	1	3	•	Up to 8x40	
		STM32L4x5 - USB OTG lines									
		STM32L475	256 to 1024	128	•	2	1	3	•		
		STM32L4x3 - USB Device + Segment LCD lines									
		STM32L433	128 to 256	64		1	1	1	USB Device		
		STM32L4x2 - USB Device lines									
		STM32L452	256 to 512	160		1	1	1	USB Device		
		STM32L432	128 to 256	64		1	1	1	USB Device		
		STM32L4x1 - Access lines									
		STM32L471	512 to 1024	128	•	2	1	3			
LOW POWER		STM32L451	256 to 512	160		1	1	1			
		STM32L431	128 to 256	64		1	1	1			

Серия STM32L4 – одни из лучших микроконтроллеров в своем классе в сегменте Ultra Low-Power. Комбинация ядра ARM Cortex-M4 с FPU и функциями сверхнизкого энергопотребления делает STM32L4 наиболее подходящим для приложений, требующих высокой производительности при работе от батареи или от запасенной энергии (energy harvesting). Как и серия L1, STM32L4 предоставляет динамическое изменение напряжения и тактовый генератор со сверхнизким энергопотреблением.

Текущие значения потребления:

- Режим сверхнизкого энергопотребления: 30 нА с резервными регистрами без RTC.
- Режим сверхнизкого энергопотребления + RTC: 330 нА с резервными регистрами (5 линий пробуждения).
- Режим сверхнизкого энергопотребления + 32 КБ ОЗУ: 360 нА.
- Режим сверхнизкого энергопотребления + 32 КБ ОЗУ + RTC: 660 нА.
- Динамический рабочий режим: до 100 мкА/МГц.
- Время пробуждения: 5 мкс.

STM32L4 совместима с выводами нескольких микроконтроллеров из серии STM32F.

Наиболее важные особенности данной серии:

- Ядро:

– Ядро ARM Cortex-M4F с FPU с максимальной тактовой частотой 80 МГц.

- Память:

– Статическое ОЗУ до 320 КБ.

20 Байт с батарейным питанием и стиранием при обнаружении несанкционированного доступа.

– Flash-память от 256 до 1024 КБ.

– Поддержка интерфейсов SDMMC и FSMC.

– Каждый чип имеет запрограммированный на заводе 96-разрядный уникальный идентификационный номер устройства.

- Периферийные устройства:

– Каждое устройство серии L4 имеет ряд периферийных устройств, которые разнятся от одной линейки к другой.

- Генераторы состоят из внутреннего RC (16 МГц, 37 кГц), дополнительного внешнего HSE (от 1 до 24 МГц), LSE (32,768 кГц).

- Корпусы ИС: LQFP, UFBGA, WLCSP.

- Диапазон рабочего напряжения от 1,7В до 3,6 В.

#### **Серия L4+**

Серия STM32L4+, представленная на рынке в конце 2017 года, является новой, лучшей в своем классе, серией микроконтроллеров в сегменте Ultra Low-Power. Серия STM32L4+ разрушает пределы возможностей обработки в мире сверхнизкого энергопотребления, предоставляя 150 DMIPS/409 CoreMark баллов, одновременно выполняясь из внутренней Flash-памяти и встраивая 640 КБ SRAM, обеспечив более продвинутые потребительские, медицинские и промышленные приложения и устройства с пониженным энергопотреблением.

Микроконтроллеры STM32L4+ предоставляют динамическое изменение напряжения, позволяя сбалансировать энергопотребление при потребности в обработке, периферийные устройства с пониженным энергопотреблением (LP UART, LP-таймеры), доступные в режиме Останова, функции сохранности и безопасности, интеллектуальные и многочисленные периферийные устройства, продвинутые и с пониженным энергопотреблением аналоговые периферийные устройства, такие как операционные усилители, компараторы, 12-разрядные ЦАП и 16-разрядные АЦП (аппаратная передискретизация (oversampling)).

Новая серия STM32L4+ также включает в себя передовые графические функции, обеспечивающие современные графические пользовательские интерфейсы. Chrom-ART Accelerator™ – собственный аппаратный графический ускоритель от ST,

который эффективно обрабатывает повторяющиеся графические операции, высвобождая основные возможности ЦПУ для обработки в реальном времени или даже для более сложных графических операций. Ускоритель Chrom-ART Accelerator в сочетании с большой встроенной памятью SRAM, оптимизатором циклической памяти дисплея (round display memory optimizer) Chrom-GRC™, высокопроизводительным интерфейсом Octo-SPI и современными контроллерами TFT и DSI позволяет получить «смартфоноподобную» графику и пользовательские интерфейсы в одном чипе при сверхнизком энергопотреблении.

Таблица 11 Возможности STM32L4+

Common to all STM32L4+		Core: Cortex-M4F with ART™ Accelerator Instruction set: Thumb, Thumb-2, DSP, FPU Internal RC oscillators: HSI=16MHz, LSI=37KHz External clocks: HSE=1-24MHz, LSE=32.768KHz Maximum Core Frequency: 120MHz Low power modes: Low-power run, Sleep, Low-power sleep, Stop with RTC, stop without RTC, Standby with RTC and Standby without RTC Year of commercialization: 2017 Available Packages: LQFP(100,144), UFBGA(132,144,169), WLCSP(144)										
		Product Line	FLASH (KB)	RAM (KB)	Memory I/F	Op-Amp	Comp.	12-bit ADC 5Msps	USB 2.0 OTG	TFT Display	MIPI-DSI	AES 128/256
		STM32L4R5/S5										
		STM32L4R5	1024 to 2048	640	•	2	2	1	•			
		STM32L4S5	2048	640	•	2	2	1	•			•
		STM32L4R7/S7										
		STM32L4R7	1024 to 2048	640	•	2	1	1	•	•		
		STM32L4S7	2048	640	•	2	1	1	•	•		•
		STM32L4R9/S9										
		STM32L4R9	1024 to 2048	640	•	2	1	1	•	•	•	
		STM32L4S9	1024 to 2048	640	•	2	1	1	•	•	•	•

Текущие значения потребления:

- Режим сверхнизкого энергопотребления: 20 нА с резервными регистрами без RTC.
- Режим сверхнизкого энергопотребления + RTC: 200 нА с резервными регистрами (5 линий пробуждения).
- Режим сверхнизкого энергопотребления + 64 КБ ОЗУ: 800 нА.
- Режим сверхнизкого энергопотребления + 64 КБ ОЗУ + RTC: 1 мкА.
- Динамический рабочий режим: до 43 мкА/МГц.
- Время пробуждения: 5 мкс. STM32L4+ совместим с выводами нескольких микроконтроллеров из серии STM32F.


Наиболее важные особенности данной серии:



- Ядро:
  - Ядро ARM Cortex-M4F с FPU с максимальной тактовой частотой 120 МГц.
- Память:
  - Статическое ОЗУ до 640 КБ.
  - 64 Байт с батарейным питанием и стиранием при обнаружении несанкционированного доступа.
  - Flash-память от 1024 до 2048 КБ.
  - Поддержка интерфейсов SDMMC и FSMC.
  - Каждый чип имеет запрограммированный на заводе 96-разрядный уникальный идентификационный номер устройства.
- Периферийные устройства:
  - Каждое устройство серии L4+ имеет ряд периферийных устройств, которые разнятся от одной линейки к другой.
- Генераторы состоят из внутреннего RC (16 МГц, 37 кГц), дополнительного внешнего HSE (от 1 до 24 МГц), LSE (32,768 кГц).
- Корпусы ИС: LQFP, UFBGA, WLCSP.
- Диапазон рабочего напряжения от 1,7В до 3,6 В.

### Серия STM32WB

Таблица 12 Возможности STM32WB

Common to all STM32WB	 <ul style="list-style-type: none"><li>• Low voltage 1.71 to 3.6V</li><li>• Dynamic Voltage Scaling</li><li>• Multiprotocol 2.4GHz radio</li><li>• USART, LPUART, SPI, I2C</li><li>• ART™ accelerator</li><li>• 7x 16/32-bit timers</li><li>• 2xDMA</li><li>• Built-in DC/DC converter</li><li>• Temperature sensor</li><li>• 5V tolerant I/Os</li><li>• 2xWDT</li><li>• Hardware CRC</li><li>• Backup Memory</li><li>• RTC calendar/clock</li><li>• SWD</li><li>• Unique ID</li></ul>	<p>Cores: Cortex-M4F with ART™ Accelerator (Application Processor) + Cortex-M0+ (Network Processor) Instruction set: Thumb, Thumb-2, DSP, FPU Internal RC oscillators: HSI=16MHz, LSI1/LSI2=32KHz, USB=100KHz to 48MHz External clocks: HSE=1-24MHz, LSE=32.768KHz Maximum Core Frequency for Application Processor: 64MHz Low power modes: Sleep, Stop with RTC, stop without RTC, Standby with RTC and Standby without RTC Year of commercialization: 2018 Available Packages: UFQFPN(48), VFQFPN(64), WLCCSP(100)</p>													
		Product Line	FLASH (KB)	RAM (KB)	BLE 5.0	IEEE802.15.4	FW OTA	External PA Support	HW Security	Quad-SPI	16-bit ADC	USB 2.0 FS Crystal-less	Segment LCD	Capacitive Touch	AES 128/256
		STM32WB													
WIRELESS	STM32WB55	256 to 1024	Up to 256	*	*	*	*	*	*	*	*	*	*	*	

Серия STM32WB, представленная на рынке в начале 2018 года, является новой серией микроконтроллеров в сегменте Wireless. STM32WB – это линейка двухъядерных микроконтроллеров STM32 со встроенным радиомодулем 2,4 ГГц, подходящих для беспроводных приложений и приложений с Bluetooth 5.0. Данные микроконтроллеры имеют ядро Cortex-M0+, работающее на частоте 32 МГц (называемое Сетевым процессором, англ. Network Processor), предназначенное для управления радиосвязью (сопутствующий стек BLE 5.0 также предоставляется ST), и



программируемое пользователем ядро Cortex-M4, работающее на частоте 64 МГц (названное Прикладным процессором, англ. Application Processor) для основного встроенного приложения.

Платформа STM32WB является эволюцией серии Ultra Low-Power микроконтроллеров STM32L4. Она предоставляет те же цифровые и аналоговые периферийные устройства, подходящие для приложений, требующих увеличения срока службы батареи и сложных функций. STM32WB объединяет несколько периферийных устройств связи, удобный бескварцевый (crystal-less) интерфейс USB 2.0 FS, поддержку звука, драйвер ЖК-дисплея, до 72 GPIO, интегрированный SMPS для оптимизации энергопотребления и несколько режимов пониженного энергопотребления для увеличения срока службы батареи.

Помимо беспроводных функций и функций с пониженным энергопотреблением, особое внимание было уделено внедрению аппаратных функций безопасности, таких как 256-разрядный AES, PCROP, JTAG Fuse, PKA (механизм шифрования эллиптических кривых) и Root Secure Services (RSS). RSS позволяет аутентифицировать связь OTA, независимо от стека радиосвязи или приложения.

STM32WB55 является устройством, сертифицированным Bluetooth 5.0, и предлагает поддержку программного обеспечения Mesh 1.0, несколько профилей и гибкость для интеграции фирменных стеков BLE. Также доступен пакет программного обеспечения, сертифицированный OpenThread. Радиомодуль также может запускать протоколы BLE/OpenThread одновременно. Встроенный универсальный MAC позволяет использовать другие собственные стеки IEEE 802.15.4, такие как ZigBee®, или собственные протоколы, предоставляя еще больше возможностей для подключения устройств к Интернету вещей (IoT).

Наиболее важные особенности данной серии:

- Ядра:
  - Ядро ARM Cortex-M4F с FPU на максимальной тактовой частоте 64 МГц (Прикладной процессор, англ. Application Processor).
  - Ядро ARM Cortex-M0+ с максимальной тактовой частотой 32 МГц (Сетевой процессор, англ. Network Processor).
- Память:
  - Статическое ОЗУ до 256 КБ.
  - Flash-память до 1024 КБ.
  - Поддержка интерфейса Quad-SPI.

– Каждый чип имеет запрограммированный на заводе 96-разрядный уникальный идентификационный номер устройства.

- Радиомодуль:

- BLE 5.0-совместимый верхний уровень (front-end) и стек радио.

- IEEE 802.15.4-совместимый верхний уровень (front-end) радио.

- Возможность обновления микропрограммы по воздуху.

- Поддержка внешнего усилителя мощности.

- Периферийные устройства:

- Каждое устройство серии WB имеет ряд периферийных устройств, которые разнятся от одной линейки к другой.

- Генераторы состоят из нескольких внутренних RC (16 МГц, 32 кГц), дополнительного внешнего HSE (от 1 до 24 МГц), LSE (32,768 кГц).

- Корпусы ИС: WLCSP, UFQFPN, VFQFPN.

- Диапазон рабочего напряжения от 1,7В до 3,6 В.

## **Лекция на тему «Ядро Cortex и процессоры на базе Cortex-M»**

### **Рассматриваемые вопросы:**

1. Cortex и процессоры на базе Cortex-M.
2. Регистры ядра.
3. Карта памяти.
4. Обработка прерываний и исключений.

### **Теоретический материал:**

#### **Cortex и процессоры на базе Cortex-M**

ARM Cortex является обширным набором 32/64-разрядных архитектур и ядер, довольно популярных в мире встраиваемых систем. Микроконтроллеры Cortex делятся на три основных подсемейства:

– Cortex-A, что означает Application – прикладной, представляет собой серию процессоров, предоставляющих широкий спектр решений для устройств, выполняющих сложные вычислительные задачи, такие как хостинг платформы операционной системы (ОС) мобильных устройств (rich OS) (наиболее распространенными являются Linux и его производные Android) и поддержка нескольких программных приложений. Ядрами Cortex-A оснащены процессоры большинства мобильных устройств, таких как смартфоны и планшеты. В данном сегменте рынка мы можем найти несколько производителей интегральных схем: от тех, кто продает каталог компонентов (TI или Freescale) до тех, кто производит процессоры для других лицензиатов. Среди наиболее распространенных ядер в этом сегменте можно выделить 32-разрядные процессоры Cortex-A7 и Cortex-A9, а также новейшие высокопроизводительные 64-разрядные ядра Cortex-A53 и Cortex-A57.

– Cortex-M, что означает eMbedded – встраиваемый, представляет собой линейку масштабируемых, совместимых, энергоэффективных и простых в использовании процессоров, предназначенных для недорогого встраиваемого рынка. Семейство Cortex-M оптимизировано для чувствительных к стоимости и энергопотреблению микроконтроллеров, подходящих для таких приложений, как Интернет вещей (Internet of Things, IoT), связь, управление двигателем, интеллектуальный учет, устройства взаимодействия с человеком (human interface devices, HID), автомобильные и промышленные системы управления, домашняя бытовая техника, потребительские товары и медицинские инструменты. В данном

сегменте рынка мы можем найти многих производителей интегральных схем, которые производят процессоры Cortex-M: ST Microelectronics является одним из них.

– Cortex-R, что означает Real-Time – реального времени, представляет собой серию процессоров, предлагающих высокопроизводительные вычислительные решения для встраиваемых систем, где необходимы надежность, высокая доступность, отказоустойчивость, ремонтпригодность и детерминированный отклик в реальном времени. Процессоры серии Cortex-R обеспечивают быструю и детерминированную обработку и высокую производительность при одновременном решении сложных задач в режиме реального времени. Они объединяют эти функции в корпусе, оптимизированном по производительности, энергопотреблению и занимаемой площади, что делает их верным выбором в надежных системах, требовательных к отказоустойчивости.

### **Регистры ядра**

Как и все архитектуры RISC, процессоры Cortex-M являются машинами загрузки/хранения, которые выполняют операции только с регистрами ЦПУ, за исключением двух инструкций: load и store, используемых для передачи данных между регистрами ЦПУ и ячейками памяти.

На рисунке 2 показаны основные регистры Cortex-M. Некоторые из них доступны только в высокопроизводительных сериях, таких как M3, M4 и M7. R0-R12 являются регистрами общего назначения и могут использоваться в качестве операндов для инструкций ARM. Однако некоторые регистры общего назначения могут использоваться компилятором в качестве регистров со специальными функциями. R13 – регистр указателя стека (Stack Pointer, SP), который также считается банковым. Это означает, что содержимое регистра изменяется в соответствии с текущим режимом ЦПУ (привилегированным или непривилегированным). Данная функция обычно используется операционными системами реального времени (OSPB) для переключения контекста.

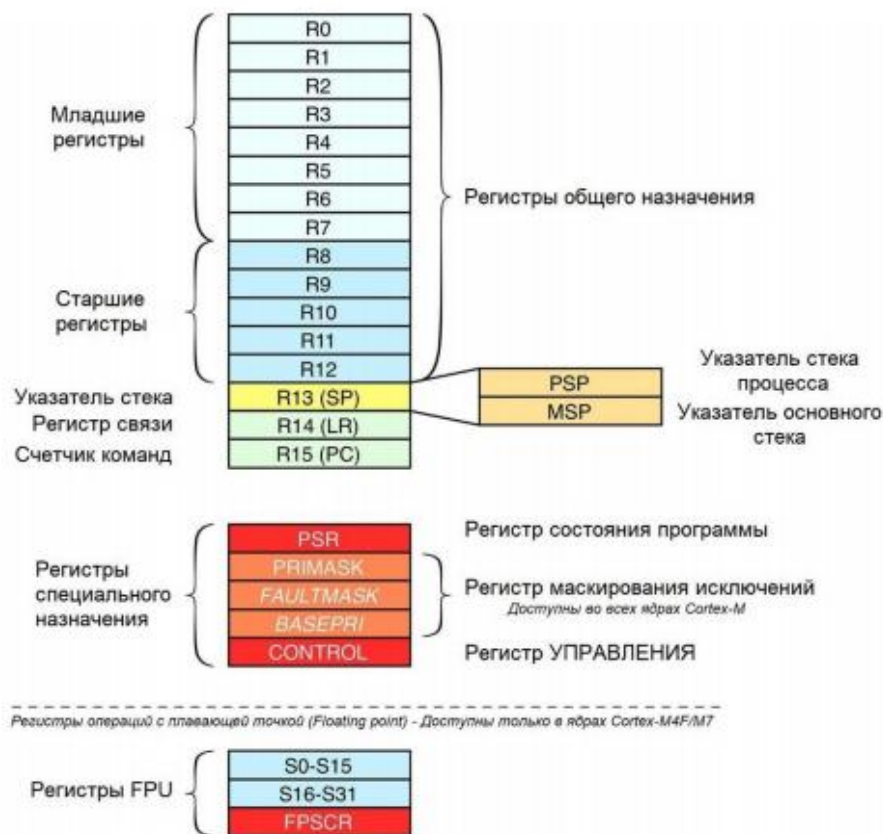


Рисунок 2 Регистры ядра ARM Cortex-M

Например, рассмотрим следующий код Си с использованием локальных переменных

“a”, “b”, “c”:

...

**uint8\_t** a,b,c;

a = 3;

b = 2;

c = a \* b;

...

Компилятор сгенерирует следующий ассемблерный код ARM:

```

1  movs  r3, #3          ;поместить "3" в регистр r3
2  strb  r3, [r7, #7]    ;сохранить содержимое r3 в "a"
3  movs  r3, #2          ;поместить "2" в регистр r3
4  strb  r3, [r7, #6]    ;сохранить содержимое r3 в "b"
5  ldrbr r2, [r7, #7]    ;загрузить содержимое "a" в r2
6  ldrb  r3, [r7, #6]    ;загрузить содержимое "b" в r3

```

7    smulbb    r3, r2, r3    ;перемножить "a" на "b" и сохранить результат в r3

8    strb    r3    , [r7, #5]    ;сохранить результат в "с"

Как мы видим, все операции всегда выполняются с регистром. Команды в строках 1-2 перемещают число 3 в регистр r3 и затем сохраняют его содержимое (то есть число 3) в ячейке памяти, заданной регистром r7 (который является указателем стекового кадра (frame pointer), как мы увидим в Главе 20) плюс смещение в 7 ячеек памяти – это ячейка, в которой хранится переменная. То же самое происходит для переменной b в строках 3-4. Затем строки 5-7 загружают содержимое переменных a и b и выполняют умножение. Наконец, строка 8 сохраняет результат в ячейке памяти переменной c.

### Карта памяти

ARM определяет стандартизированное адресное пространство памяти, общее для всех ядер Cortex-M, что обеспечивает переносимость кода между различными производителями интегральных схем. Адресное пространство размером 4 ГБ и состоит из нескольких секций с различными логическими функциями. На рисунке 3 показана карта памяти (или схема распределения памяти) процессора Cortex-M.

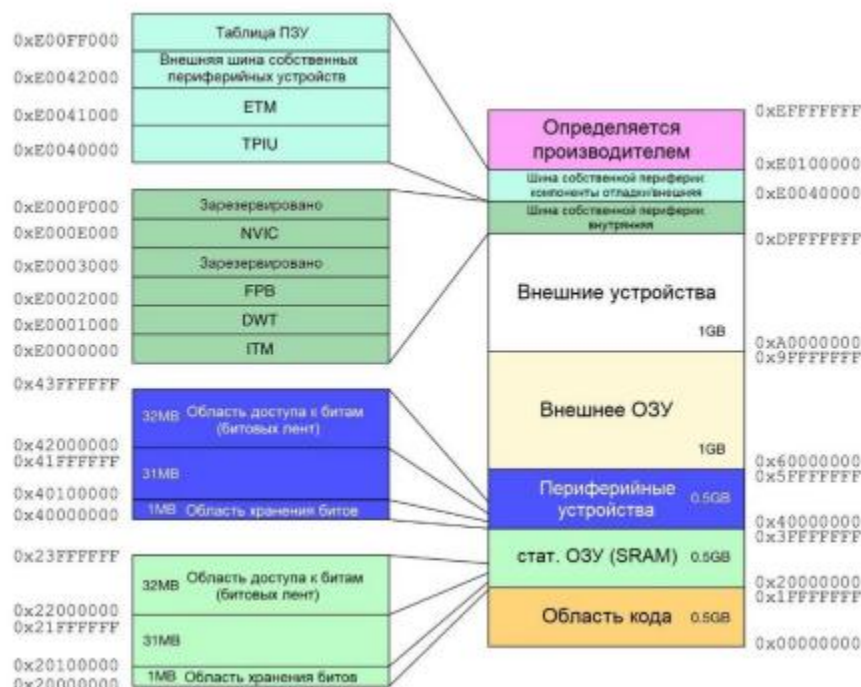


Рисунок 3 Фиксированное адресное пространство памяти Cortex-M

Первые 512 МБ выделены для области кода. Устройства STM32 дополнительно делят эту область на несколько секций, как показано на рисунке 4. Давайте кратко рассмотрим их.

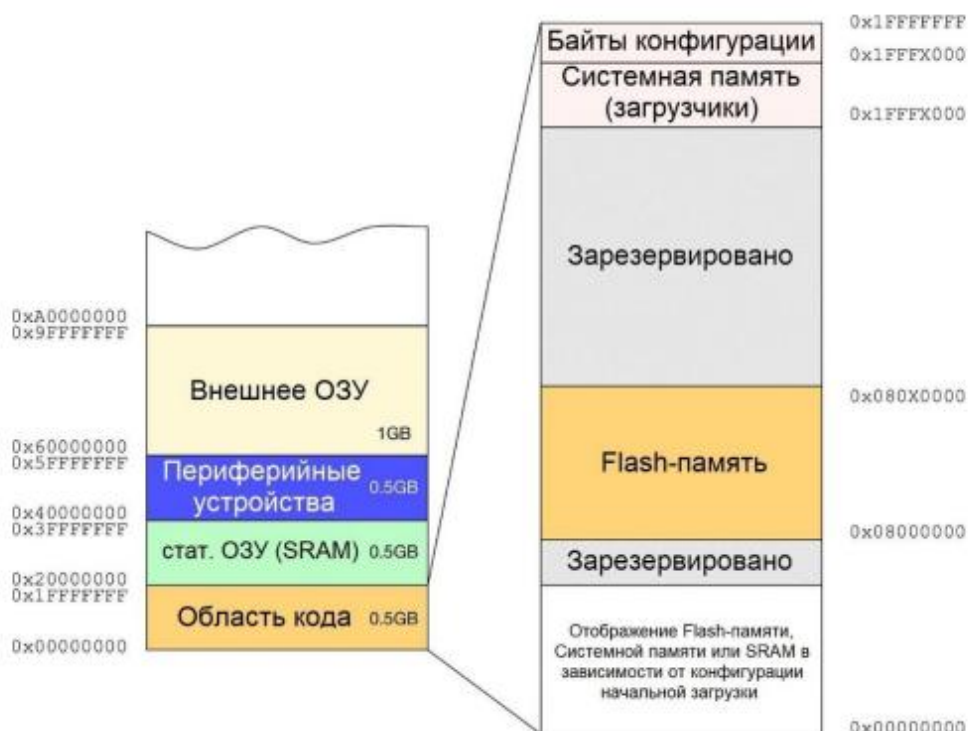


Рисунок 4 Карта памяти области кода на микроконтроллерах STM32

Все процессоры Cortex-M отображают область кода, начиная с адреса 0x0000 0000. Данная область также включает указатель на начало стека (обычно помещается в SRAM) и таблицу векторов. Расположение области кода стандартизировано среди всех других производителей Cortex-M, несмотря на то, что архитектура ядра достаточно гибкая, чтобы позволить им организовать данную область по-другому. Фактически, для всех устройств STM32 область, начинающаяся с адреса 0x0800 0000, связана с внутренней Flash-памятью микроконтроллера и является областью, в которой находится программный код. Тем не менее, благодаря определенной конфигурации начальной загрузки, данная область также отражается (aliased) на адрес 0x0000 0000. Это означает, что вполне возможно сослаться на содержимое Flash-памяти, начиная с адреса 0x0800 0000 и 0x0000 0000 (например, процедура, расположенная по адресу 0x0800 16DC, также доступна из 0x0000 16DC).

Последние две секции выделены под Системную память и Байты конфигурации (Option bytes). Первая – это область ПЗУ, зарезервированная для загрузчиков. Каждое семейство STM32 (и их подсемейства – low density, medium density и т. д.) предоставляет загрузчик, предварительно запрограммированный в

микросхему во время производства. Данный загрузчик можно использовать для загрузки кода с нескольких периферийных устройств, включая USART, USB и CAN-шину. Область Байтов конфигурации содержит последовательность битовых флагов, которые могут использоваться для конфигурации некоторых аспектов микроконтроллера (таких как защита от чтения Flash-памяти, аппаратный сторожевой таймер, режим начальной загрузки и т. д.) и связаны с конкретным микроконтроллером STM32.

Возвращаясь ко всему 4 Гб адресному пространству, следующая основная область – это область, ограниченная внутренним статическим ОЗУ (SRAM) микроконтроллера. Она начинается с адреса 0x2000 0000 и потенциально может расширяться до 0x3FFF FFFF. Однако фактический конечный адрес зависит от действующего количества внутреннего SRAM. Например, в случае микроконтроллера STM32F103RB с 20 Кб SRAM конечный адрес 0x2000 4FFF. Попытка получить доступ к ячейке за пределами данной области вызовет исключение отказа шины Bus Fault.

Следующие 0,5 Гб памяти предназначены для отображения периферийных устройств. Каждое периферийное устройство, предоставляемое микроконтроллером (таймеры, интерфейсы I<sup>2</sup>C и SPI, USART и т. д.), имеет отображение в данной области. Организация данного пространства памяти зависит от конкретного микроконтроллера.

### **Обработка прерываний и исключений**

Обработка прерываний и исключений – одна из самых мощных функций процессоров на базе Cortex-M. Прерывания и исключения являются асинхронными событиями, которые изменяют ход программы. Когда происходит исключение или прерывание, ЦПУ приостанавливает выполнение текущей задачи, сохраняет свой контекст (т. е. свой указатель стека) и начинает выполнение процедуры, предназначенной для обработки события прерывания. Эта процедура называется Обработчиком исключений (Exception Handler) в случае исключений и Процедурой обслуживания прерывания (Interrupt Service Routine, ISR) в случае события прерывания. После обработки исключения или прерывания ЦПУ возобновляет предыдущий поток выполнения, и предыдущая задача может продолжить свое выполнение.

В архитектуре ARM прерывания являются одним из типов исключений. Прерывания обычно генерируются от встроенных периферийных устройств (например, таймера) или внешних входов (например, тактильного переключателя, подключенного к GPIO), и в некоторых случаях они могут запускаться программно. Исключения, напротив, связаны с выполнением программного обеспечения, а само



ЦПУ может быть источником исключений. Это могут быть события отказов, такие как попытка доступа к неверной ячейке памяти, или события, сгенерированные операционной системой, если таковые имеются.

Каждое исключение (а, следовательно, и прерывание) имеет номер, который однозначно идентифицирует его. Каждому исключению может быть назначен уровень приоритета, который определяет порядок обработки в случае одновременных прерываний: чем меньше число, тем выше приоритет. Например, предположим, что у нас есть две процедуры прерывания, связанные с внешними входами А и В. Мы можем назначить прерывание с более высоким приоритетом (с более низким числом) для входа А. Если прерывание, связанное с А, поступает при обработке процессором прерывания со входа В, то выполнение В приостанавливается, что позволяет немедленно выполнить процедуру обработки прерывания с более высоким приоритетом.

И исключения, и прерывания обрабатываются отдельным модулем, который называется Контроллер вложенных векторных прерываний (Nested Vectored Interrupt Controller, NVIC). Контроллер NVIC обладает следующими особенностями:

- Гибкое управление исключениями и прерываниями: NVIC может обрабатывать как сигналы/запросы прерываний, поступающие от периферийных устройств, так и исключения, поступающие от ядра процессора, что позволяет нам разрешать/запрещать их в программном обеспечении (кроме NMI).

- Поддержка вложенных исключений/прерываний: NVIC позволяет назначать уровни приоритета исключениям и прерываниям (кроме первых трех типов исключений), предоставляя возможность классифицировать прерывания в зависимости от потребностей пользователя.

- Векторный переход к исключению/прерыванию: NVIC автоматически определяет расположение обработчика исключений, связанного с исключением/прерыванием, без необходимости в дополнительном коде.

- Маскирование прерываний: разработчики могут приостанавливать выполнение всех обработчиков исключений (кроме NMI) или приостанавливать некоторые из них на основе уровня приоритета благодаря набору отдельных регистров. Это позволяет выполнять критические задачи безопасным способом, исключив асинхронные прерывания.

- Детерминированное время реакции на прерывание: еще одна интересная особенность NVIC – детерминированная задержка обработки прерывания, которая равна 12 тактовым циклам для всех ядер Cortex-M3/4, 15 тактовым циклам для

Cortex-M0, 16 тактовым циклам для Cortex-M0+ независимо от текущего состояния процессора.

– Перемещение обработчиков исключений: как мы рассмотрим далее, обработчики исключений могут быть перемещены в другие ячейки Flash-памяти, а также в совершенно другую, даже внешнюю память, не ПЗУ (ROM). Это обеспечивает большую степень гибкости для продвинутых приложений.

## Лекция на тему «Отладочная плата Nucleo. Основные характеристики».

### Рассматриваемые вопросы:

1. Основные характеристики отладочной платы Nucleo-F401RE
2. Программируемый комплекс «Электроника»

### Теоретический материал:

#### Основные характеристики отладочной платы Nucleo-F401RE

Каждый практический текст об электронном устройстве требует отладочной платы (development board, также известной как kit) для начала работы с ним. В мире STM32 одной из самых распространенных отладочных плат является STM32 Discovery. Компания ST разработала более 20 различных плат Discovery, полезных для тестирования микроконтроллеров STM32 и их возможностей.



Рисунок 19 Отладочная плата STM32L0538 Discovery (Discovery kit), представленная ST в 2015 году

Например, плата STM32L0538DISCOVERY (рисунок 19) позволяет тестировать как микроконтроллер STM32L053, так и e-ink дисплей. Вы можете найти множество руководств в Интернете, посвященных платам линейки Discovery.

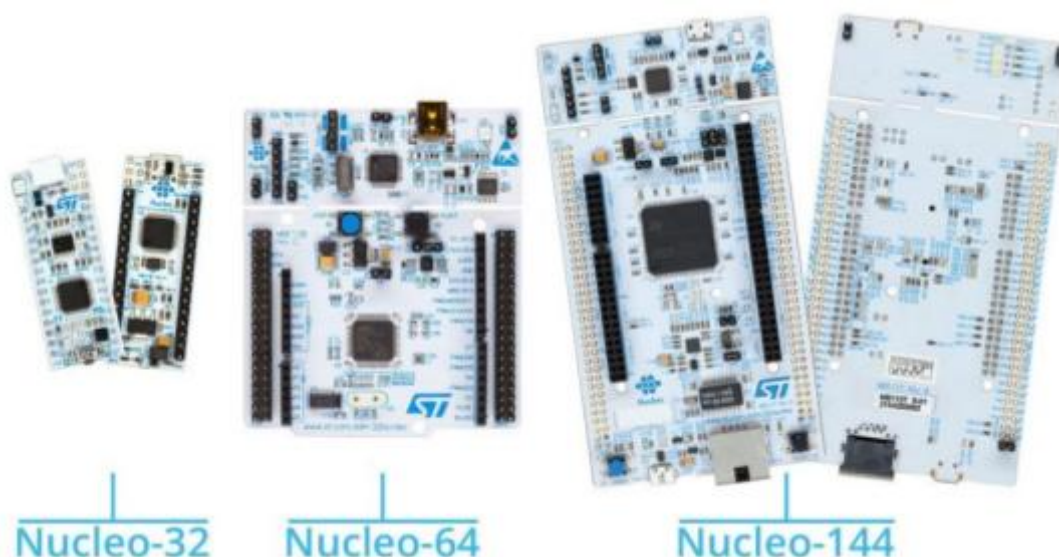


Рисунок 20 Отладочная плата Nucleo

Линейка Nucleo разделена на три основные группы: Nucleo-32, Nucleo-64 и Nucleo-144 (см. рисунок 20). Название каждой группы происходит от используемого типа корпуса микроконтроллера: Nucleo-32 использует STM32 в корпусе LQFP-32; Nucleo-64 использует LQFP64; Nucleo-144 использует LQFP-144. Nucleo-64 была первой линейкой, представленной на рынке, и в ней 16 различных плат, каждая с определенным микроконтроллером STM32. Nucleo-144 была представлена в январе 2016 года, и это первая недорогая отладочная плата, оснащенная мощным STM32F746. Она также предоставляет Ethernet pyther и порт LAN.

Nucleo состоит из двух частей, как показано на рисунке 21. Часть с разъемом mini-USB представляет собой встроенный отладчик ST-LINK 2.1, который используется для загрузки микропрограммы в целевой микроконтроллер и выполнения пошаговой отладки. Интерфейс ST-LINK также предоставляет виртуальный COM-порт (Virtual COM Port, VCP), который можно использовать для обмена данными и сообщениями с хостПК. Одна из главных особенностей плат Nucleo заключается в том, что интерфейс STLINK можно легко отделить от остальной части платы (двое красных ножниц на рисунке 21 показывают, где можно ломать плату). Таким образом, ее можно использовать в качестве автономного программатора ST-LINK. Однако ST-LINK предоставляет дополнительный интерфейс SWD, который можно использовать для программирования другой платы без отсоединения интерфейса ST-LINK от Nucleo (как это уже происходит с платами Discovery) путем удаления двух перемычек, помеченных как ST-LINK. Остальная часть платы содержит целевой микроконтроллер, англ. target MCU (микроконтроллер, который мы будем использовать для разработки наших



F103RB (оснащенной популярным микроконтроллером STM32F103), а затем адаптировать ее к более мощной Nucleo (например, STM32Nucleo-F401RE), если вам требуется больше вычислительной мощности. В дополнение к Arduino-совместимым гнездовым разъемам (типа «female»), Nucleo предоставляет свои собственные разъемы расширения. Это два штыревых разъема (типа «male») 2x19, 2,54 мм. Они называются Morpho-разъемами и являются удобным способом доступа к большинству выводов микроконтроллера. На рисунке 23 показаны периферийные устройства STM32 и GPIO, связанные с Morpho-разъемом.

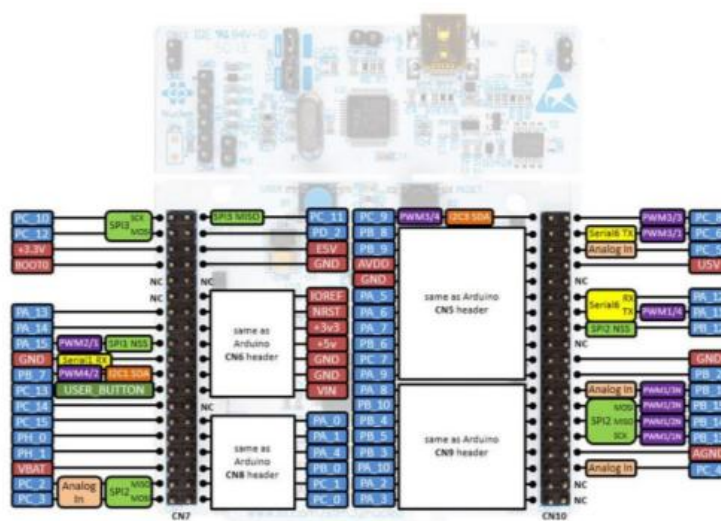


Рисунок 23 Периферийные устройства и GPIO, связанные со штыревыми Morpho-разъемами

На рисунке 24 показана плата Nucleo с платой расширения X-NUCLEO-IDB04A1 – «шилд» BlueNRG с монолитным сетевым процессором Bluetooth Low Energy 4.0.



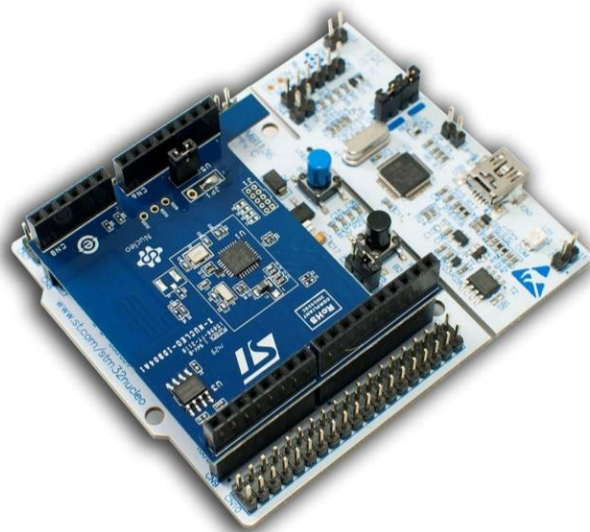


Рисунок 24 Готовая плата расширения BlueNRG

В таблице 16 приведены основные и общие характеристики для всех плат Nucleo.

Таблица 16 Список доступных плат Nucleo и их характеристики

Common to all Nucleo boards:					
Integrated ST-Link debugger that can be also used as stand-alone debugger					
Virtual COM port integrated in ST-Link interface					
64-pin LQFP target MCU					
2x(2x19) 2.54mm Morpho extension headers					
Arduino UNO extension headers					
1 LED and 1 Tactile switch freely available to programmer					
	Nucleo P/N	STM32 MCU	RAM (KB)	FLASH (KB)	F <sub>cpu</sub> (MHz)
HIGH PERFORMANCE	NUCLEO-F446RE	STM32F446RET6	128	512	180
	NUCLEO-F411RE	STM32F411RET6	128	512	100
	NUCLEO-F410RB	STM32F410RBT6	32	128	100
	NUCLEO-F401RE	STM32F401RET6	96	512	84
MAINSTREAM	NUCLEO-F334R8	STM32F334R8T6	16	64	72
	NUCLEO-F303RE	STM32F303RET6	64	512	72
	NUCLEO-F302R8	STM32F302R8T6	16	64	72
	NUCLEO-F103RB	STM32F103RBT6	20	128	72
	NUCLEO-F091RC	STM32F091RCT6	32	128	48
	NUCLEO-F072RB	STM32F072RBT6	16	128	48
	NUCLEO-F070RB	STM32F070RBT6	16	128	48
	NUCLEO-F030R8	STM32F030R8T6	8	64	48
	NUCLEO-L476RG	STM32L476RGT6	96	1024	80
	NUCLEO-L152RE	STM32L152RET6	80	512 + 16KB EEPROM	32
LOW POWER	NUCLEO-L073RZ	STM32L073RZT6	20	192 + 6KB EEPROM	32
	NUCLEO-L053R8	STM32L053R8T6	8	64+2K EEPROM	32

## Программируемый комплекс «Электроника»

Предназначен для программирования и изучения функций микроконтроллера STM32. Внешний вид устройства представлен на рисунке 1.



Рисунок 1 Программируемой комплекс «Электроника»

### Характеристики программируемого комплекса «Электроника»:

- Содержит микроконтроллер STM32F411RE в составе отладочной платы NUCLEO-64. Его расположение на плате представлено в соответствии с рисунком 2;

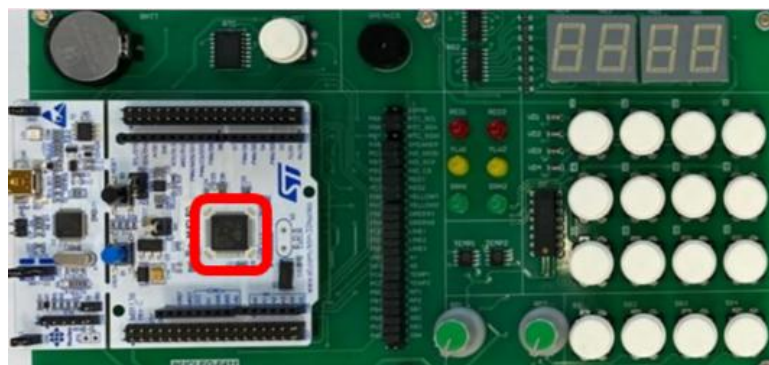


Рисунок 2 микроконтроллер STM32F411RE в составе ПМК «Электроника»

- Содержит четыре кнопки SB1..SB4, предназначенные для формирования коротких сигналов управления для МК. Его расположение на плате представлено в соответствии с рисунком 3;





Рисунок 3 четыре кнопки SB1..SB4 в составе ПМК «Электроника»

- Содержит пьезоизлучатель с диапазоном резонансных частот 4.1 ... 500 Гц; служит для формирования звукового сигнала. Его расположение на плате представлено в соответствии с рисунком 4;



Рисунок 4 Пьезоизлучатель в составе ПМК «Электроника»

- Содержит 6 светодиодов, аноды которых через токоограничивающие резисторы выведены на соответствующие разъемы. Светодиоды предназначены для осуществления световой индикации. Его расположение на плате представлено в соответствии с рисунком 5;

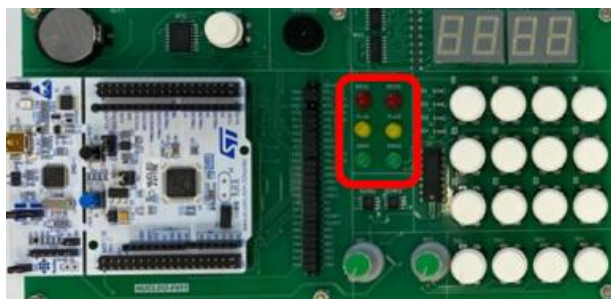


Рисунок 5 6 светодиодов в составе ПМК «Электроника»

– Оснащен двумя аналоговыми датчиками температуры TMP36, напряжение на выходе которых пропорционально температуре. Его расположение на плате представлено в соответствии с рисунком 6;

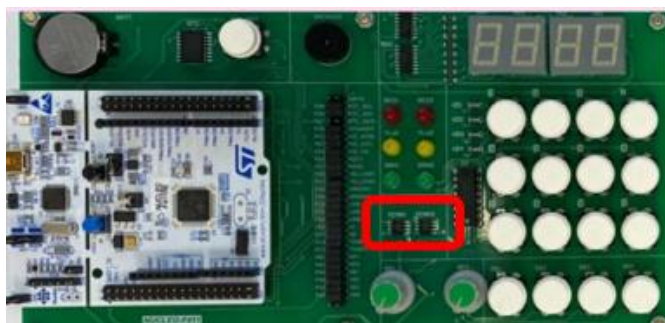


Рисунок 6 Аналоговые датчики температуры TMP36 в составе ПМК «Электроника»

– Содержит микросхему часов реального времени DS3231 с автономным питанием от батарейки CR2032 и возможностью сброса параметров без отключения питания. Микросхема связывается с микроконтроллером по интерфейсу I2C. Его расположение на плате представлено в соответствии с рисунком 7;



Рисунок 7 Микросхема часов реального времени DS3231 в составе ПМК «Электроника»

– Содержит 4 семисегментных индикатора, соединенных по схеме с общим катодом. Сегменты и катоды индикаторов подключены к выходам двух сдвиговых регистров 74НС595. Для включения любого сегмента индикатора необходимо подать логические единицы на соответствующую клемму сегмента (А..Н) и катод (HG0.. HG3). Его расположение на плате представлено в соответствии с рисунком 8.



Рисунок 8 4 семисегментных индикатора, соединенных по схеме с общим катодом в составе ПМК «Электроника»

## Тема 2.2. Особенности программирования микроконтроллеров STM32

Лекция на тему «Принципы построения программ для микроконтроллеров. Средства программирования и отладки микроконтроллеров»

### Рассматриваемые вопросы:

1. Виды компиляторов.
2. Виды программ-трансляторов.
3. Языки программирования для микроконтроллеров.

### Теоретический материал:

#### Языки программирования для микроконтроллеров

Программирование для микроконтроллеров, как и программирование для универсальных компьютеров, прошло большой путь развития — от использования машинных кодов до применения современных интегрированных сред, предоставляющих встроенный текстовый редактор, обеспечивающих трансляцию программ, их отладку и загрузку во внутреннюю память микроконтроллеров. В настоящее время исходный текст программы пишется на каком-либо из языков программирования, относящемся к одной из двух групп:

- языки программирования «низкого» уровня;
- языки программирования «высокого» уровня.

Классификация языков программирования приведена на рисунке 7.1.

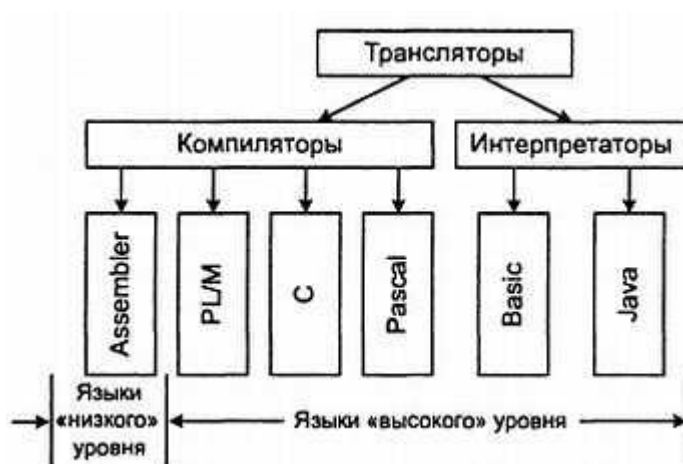


Рисунок 7.1 Классификация языков программирования

В языках «низкого» уровня каждому оператору соответствует не более одной машинной команды. В их состав обязательно входит набор машинных команд каждого конкретного процессора. Языки программирования низкого уровня в настоящее время называются ассемблерами (старое название «автокоды»). Для каждого процессора существует своя группа ассемблеров. Ассемблеры для одного и того же процессора различаются между собой дополнительными возможностями, облегчающими программирование.

Языки программирования «высокого» уровня позволяют заменять один оператор несколькими машинными командами. Это увеличивает производительность труда программистов. Кроме того, языки «высокого» уровня позволяют писать программы, которые могут выполняться на различных микропроцессорах. (Естественно, что при этом необходимо использовать программы-трансляторы для соответствующего процессора.) В настоящее время для микроконтроллеров широко используются такие языки программирования высокого уровня, как C, Си++, Python и другие Си-подобные языки программирования.

О преимуществах и недостатках языков высокого и низкого уровней говорилось достаточно много. Выбор языка программирования зависит от состава аппаратуры, для которой пишется программа, наличия тех или иных средств разработки программного обеспечения, а также от требуемого быстродействия всего программно-аппаратного комплекса в целом.

В тех случаях, когда объем ОЗУ и ПЗУ мал (в районе нескольких килобайтов), альтернативы ассемблеру нет. Именно этот язык программирования позволяет получать самый короткий и самый быстродействующий код программы (при прочих равных условиях).

Языки программирования высокого уровня позволяют значительно сократить время создания программы, но при этом увеличивается ее размер, поэтому использование такого языка программирования для микропроцессорных систем требует достаточно большого объема памяти программ (несколько десятков килобайтов). Увеличение объема программы связано с несколькими факторами:

1. Язык программирования рассчитывается на все случаи жизни, поэтому в большинстве случаев человек мог бы написать программу короче, исключив ненужные в данном конкретном случае проверки или защиты.

2. Если программист не знает, к чему приводит использование тех или других операторов языка программирования, то он может выбирать операторы,

неоптимальные как с точки зрения длины машинного кода программы, так и с точки зрения быстродействия программы.

3. Программист не использует подпрограммы там, где они могли бы сократить объем программы, так как на языке программирования высокого уровня это всего один или несколько операторов.

Первый из этих пунктов постепенно утрачивает свое значение с появлением все более совершенных трансляторов. Третий пункт тоже решается тем же путем при применении различных видов оптимизаторов, входящих в состав компилятора. Однако в большинстве случаев оптимизатор не может определить одинаковые действия, если они отличаются хотя бы одной командой. Кроме того, оптимизатор работает только в пределах одного программного модуля.

### **Виды программ-трансляторов**

Процесс преобразования операторов исходного языка программирования в машинные коды микропроцессора называется трансляцией исходного текста. В настоящее время ручная трансляция программ практически не используется. Трансляция производится специальными трансляторами.

Существуют два больших класса таких программ: компиляторы и интерпретаторы. При использовании компиляторов весь исходный текст программы преобразуется в машинные коды, и именно эти коды записываются в память микропроцессора. При использовании интерпретатора в память микропроцессора записывается исходный текст программы, а трансляция производится при считывании очередного оператора. Естественно, что быстродействие интерпретаторов намного ниже, чем у компиляторов, так как, например, при использовании оператора в цикле он транслируется многократно.

Применение интерпретатора может обеспечить выигрыш только в случае его разработки для языка программирования «высокого» уровня. В этом случае может быть сэкономлена внутренняя память программ, а также облегчен процесс отладки или облегчен перенос программ с одного типа процессора на другой.

При программировании на ассемблере применение интерпретатора приводит к проигрышу по всем параметрам, поэтому для языков программирования низкого уровня применяются только программы-компиляторы.

Для программирования микроконтроллеров как на языке программирования «низкого» уровня, так и на языке программирования «высокого» уровня чаще используются компиляторы, поэтому рассмотрим подробнее виды этих трансляторов.

### **Виды компиляторов**

Программы-компиляторы бывают оценочные и профессиональные.

Оценочные компиляторы позволяют написать простейшие программы для конкретного процессора и определить, подходит ли процессор для тех задач, которые предстоит решать в процессе разработки устройства.

Конечно, если программа очень проста, то можно весь программный продукт выполнить на оценочном компиляторе. Оценочные компиляторы позволяют транслировать одиночный файл исходного текста программы. Иногда такие компиляторы позволяют включать в процесс трансляции содержимое отдельных файлов специальной директивой.

В результате работы оценочного компилятора сразу получается исполняемый или загрузочный модуль программы, поэтому такие компиляторы называются компиляторы с единой трансляцией.

Профессиональные трансляторы позволяют производить компиляцию исходного текста программы по частям. Это позволяет значительно сократить время трансляции, так как нужно компилировать не весь текст программы, а только ту ее часть, которая менялась после предыдущей трансляции.

Кроме того, каждый программный модуль может писать отдельный программист. Это позволяет сократить время написания программы. Даже в том случае, если программу пишет один человек, время написания программы сокращается за счет использования готовых отлаженных и оттранслированных программных модулей.

При многомодульном программировании процесс получения исполняемого кода программы разбивается на два этапа: компиляция исходного текста модулей (некоторых или всех) и связывание (компоновка) полученных объектных файлов модулей в единую программу. Такой процесс получил название раздельная компиляция-компоновка.

Оценочные компиляторы обычно предлагаются бесплатно фирмами — производителями микроконтроллеров.

Профессиональные компиляторы разрабатываются и продаются независимыми фирмами — разработчиками программного обеспечения.

В состав современных профессиональных средств написания и отладки программ для микроконтроллеров обычно входят эмуляторы процессоров или отладочные платы, текстовый редактор, компиляторы языка высокого уровня (чаще всего «С») и ассемблера, редактор связей (компоновщик) и загрузчик программы в отладочную плату. Все программы обычно объединены интегрированной средой

разработки программного проекта, которая позволяет поддерживать один или несколько программных проектов.



## **Лекция на тему «Правила составления алгоритмов. Типы алгоритмов»**

### **Рассматриваемые вопросы:**

1. Пример разработки блок-схемы алгоритма программы для заданного микроконтроллерного устройства.
2. Особенности построения блок-схем алгоритмов программ для микроконтроллеров.

### **Теоретический материал:**

#### **Особенности построения блок-схем алгоритмов программ для микроконтроллеров**

Между первоначальным замыслом создания устройства на базе микроконтроллера (МК) и разработкой его программы есть важный этап - составление блок-схемы алгоритма. О нем нередко забывают или, что еще хуже, им пренебрегают. Программирование ведется "эвристически", по существу, - методом проб и ошибок. Результат - кое-как работающая громоздкая программа, не до конца понятная даже ее создателю, и плохо поддающаяся модернизации. Однако давно известны и используются программистами достаточно простые методы, позволяющие, начав со словесной формулировки алгоритма, грамотно спроектировать его блок-схему.

Взаимодействие любой МК-системы с оператором и объектом управления можно представить показанной на рис. 1 схемой. В общем случае объект управления снабжен исполнительными устройствами и датчиками. Человек-оператор воздействует на МК с помощью задающих устройств и получает информацию о состоянии объекта из показаний устройств индикации. Первые представляют собой переключатели, кнопки, переменные резисторы, вторые - световые (в том числе графические и буквенно-цифровые) индикаторы, звукоизлучающие и другие сигнальные устройства.



Рисунок 1 Взаимодействие любой МК-системы с оператором и объектом управления

Все показанные на схеме функциональные узлы и связи обязательны лишь в комплексных диалоговых системах контроля и управления. В так называемых разомкнутых системах управления МК работает "вслепую", не получая никакой информации о состоянии объекта. Иногда он даже не выдает оператору никаких сведений о работе (как своей, так и объекта), особенно? если имеется возможность оценивать результаты управления, наблюдая за самим объектом. В замкнутых системах управления МК корректирует управляющие воздействия на объект в зависимости от показаний датчиков, но устройства индикации не обязательны и здесь. Системы контроля не содержат исполнительных устройств, а с помощью задающих оператор лишь выбирает контролируемые параметры или переключает режимы работы индикаторов.

Методика проектирования систем на МК включает в себя постановку и анализ задачи, ее инженерную интерпретацию, разработку блок-схемы алгоритма и текста прикладной программы. В подобных системах максимальное число функций стремятся возложить именно на программные средства. От эффективности их реализации зависят в конечном счете требуемый объем памяти, быстродействие и надежность работы системы в целом.

### **Пример разработки блок-схемы алгоритма программы для заданного микроконтроллерного устройства**

Исходные данные:

– в резервуар Р через управляемую задвижку 1 поступает жидкость (управляемая задвижка 2 при этом - закрыта). При достижении уровня воды в

резервуаре «верхнего» уровня (определяемого датчиком D1), задвижка 1 – закрывается, и одновременно открывается задвижка 2. По мере расхода жидкости из резервуара и достижении «нижнего» уровня (определяемого датчиком D2) происходит обратный процесс: задвижка 1 – открывается, задвижка 2 - закрывается и жидкость вновь начинает наполнять резервуар. Далее процесс повторяется. Т.е. фактически система позволяет поддерживать уровень жидкости в резервуаре в пределах, устанавливаемых датчиками D1, D2. Пример автоматизированной системы управления уровнем жидкости представлен на рисунке 2.

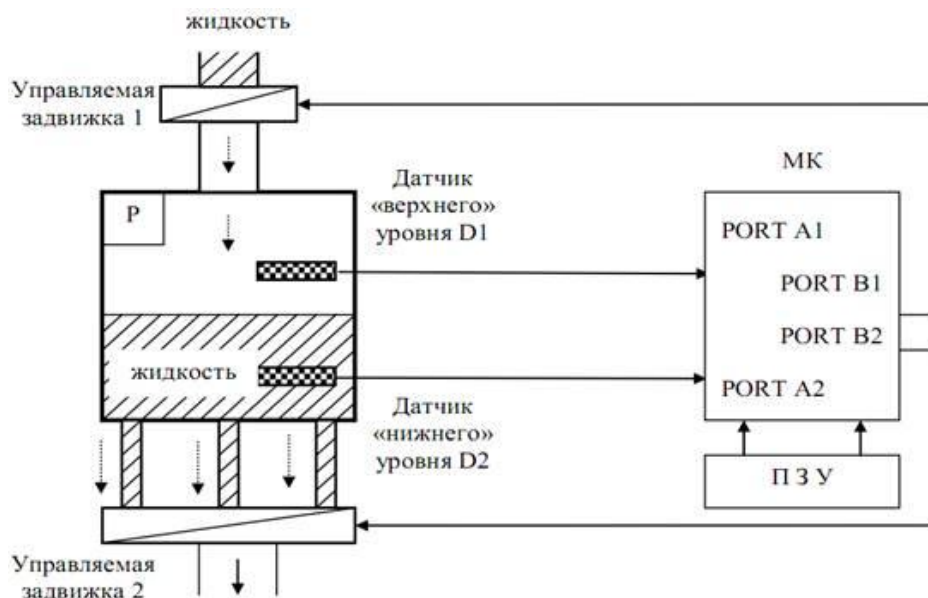


Рисунок 2 Пример автоматизированной системы управления уровнем жидкости

– сигналы с датчиков – цифровые, причем если датчик находится в воде, то на его выходе формируется сигнал логической «1», в противном случае – «0», для открывания задвижки необходимо сформировать сигнал логической «1», для закрывания – «0».

Вначале, учитывая двунаправленный характер контактов МК, необходимо установить требуемый режим их работы в соответствии с рис. 4. Далее, учитывая неопределенность исходного состояния системы, необходимо установить некоторое устойчивое ее состояние: в данном случае наиболее целесообразно закрыть обе задвижки, т.к. в таком состоянии система может находиться сколь угодно долго.

Затем необходимо проверить текущий уровень жидкости в резервуаре и в зависимости от него включить один из возможных режимов: слив или наполнение. В данном случае это сделано по состоянию «нижнего» датчика D2. Если он в воде, то открывается задвижка 2 и начинается слив до снижения уровня жидкости ниже D2,

если нет – открывается задвижка 1 и начинается наполнение до достижения уровня D1. Далее указанные режимы чередуются с учетом текущего уровня жидкости. Обратите внимание, что в программе отсутствует точка выхода, свойственная блок-схемам программного обеспечения вычислительного характера, то есть микроконтроллер функционирует циклически.

Таким образом, можно сформулировать два этапа разработки блок-схем, свойственные ВСЕМ практическим системам, использующим МК:

- инициализация портов МК и фиксация исходного состояния его управляющих выходов, независимо от состояния всех входных сигналов;
- разработка блок-схемы с учетом установленного исходного состояния;
- степень детализации блок-схемы определяется в каждом конкретном случае самим разработчиком с учетом возможности реализации требуемых операций данной архитектурой микроконтроллера (то есть наличием соответствующих команд).

Блок-схема алгоритма работы МК представлена на рисунке 3.

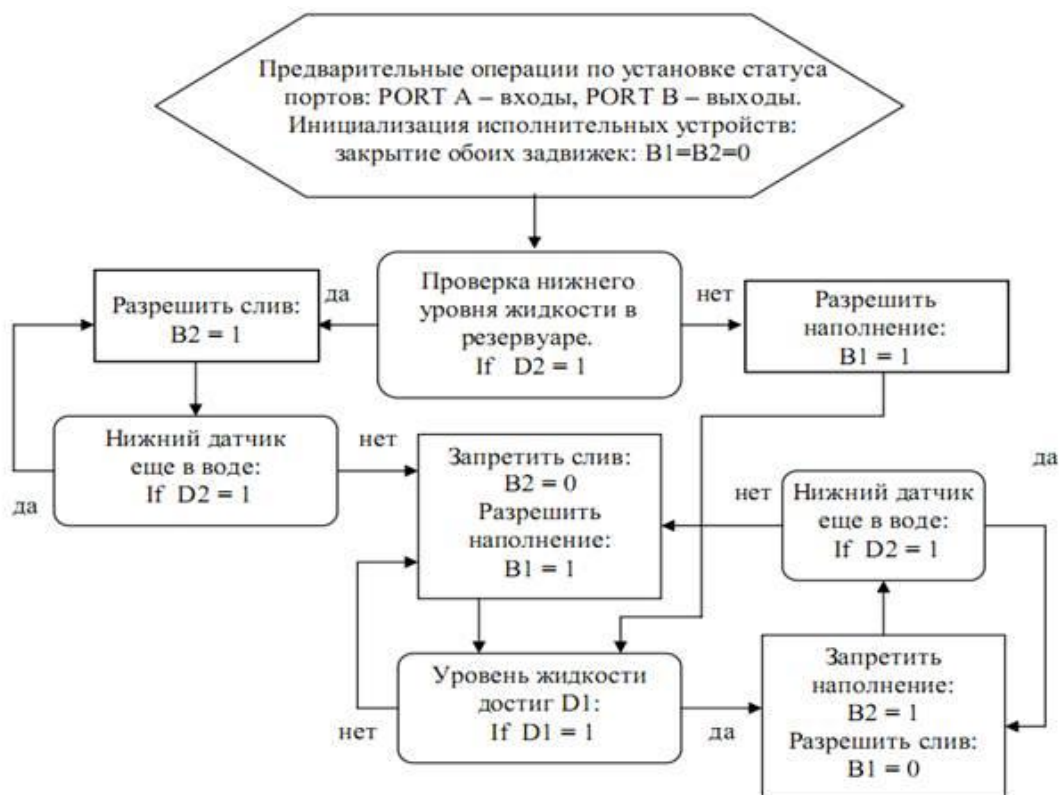


Рисунок 3 Блок-схема алгоритма работы МК в системе управления уровнем жидкости

## Лекция на тему «Диаграммы состояний. Конечный автомат»

### Рассматриваемые вопросы:

1. Конечные автоматы.
2. Общая структура программы в Switch-технологии.
3. Конечные автоматы и автоматное программирование.

### Теоретический материал:

#### Конечные автоматы и автоматное программирование

Рассмотрим такую концепцию программирования микроконтроллеров как автоматное программирование. Благодаря ней многие задачи, поставленные перед программистом, решаются гораздо легче и проще, избавляя программиста от многих сложностей. Автоматное программирование часто также называют *Switch-технологией*.

**Конечный автомат** (в англ. – **Finite State Machine, FSM**) *определяется как абстрактная машина, которая может находиться ровно в одном из конечного числа состояний в любой момент времени. Конечный автомат может переходить из одного состояния в другое в ответ на некоторые внешние входные данные и/или при выполнении заданного условия; переход из одного состояния в другое называется переходом (в англ. – transition). FSM определяется списком его состояний, его начальным состоянием и условиями для каждого перехода.*

Каким же образом конечные автоматы упрощают написание программ для микроконтроллеров (да и не только для них)? Дело в том, что традиционный способ обработки в программе переходов из одного состояния в другое в системе с ограниченным числом состояний заключается в использовании операторов if или switch (или их комбинации). В системах с небольшим количеством состояний этот подход вполне успешно работает, но если состояний много, то традиционный код для их обработки становится ужасно запутанным и трудно читаемым, он очень сложно поддается модификации.

Программирование в конечных автоматах избавляет программу от этих недостатков. Программа становится проще, ее легко модифицировать. Программа, написанная в автоматном стиле похожа на переключатель, который в зависимости от условий переключается в то или иное состояние. Количество состояний программисту изначально известно.

В грубом представлении это как выключатель освещения. Есть два состояния – включено и выключено, и два условия включить и выключить.

Также программы, написанные по Switch-технологии, легко документировать. В рамках этого стиля программирования программа представляет собой совокупность конечных автоматов, взаимодействующих друг с другом и с «внешним миром». При этом в наглядной графической форме могут быть выражены как связи между автоматами, так и их внутренняя структура.

Еще одним существенным преимуществом Switch-технологии является возможность повторного использования кода. Фактически, программа в данном случае состоит из компонентов, являющихся в высокой степени автономными «сущностями». Компонент имеет ограниченное количество связей с остальной программой, его можно разрабатывать и тестировать отдельно, а применять многократно, и именно это свойство подобных систем делает разработку быстрой и удобной.

Автомат в предлагаемом стиле программирования также является своего рода «кирпичиком», автономной единицей программы. Его связи с остальными автоматами сведены к минимуму и унифицированы. Автомат можно разработать отдельно, а затем применять в различных программных проектах.

Из всего изложенного вытекает и такое достоинство Switch-технологии как легкость модификации программ, написанных с ее помощью. Так как количество связей между автоматами минимально, изменения в одном из них чаще всего не влекут за собой необходимость коррекции кода в других автоматах.

### **Общая структура программы в Switch-технологии**

Суть описываемого нами стиля автоматного программирования кратко можно сформулировать следующим образом: программа представляет собой совокупность конечных автоматов, выполняющихся параллельно и обменивающихся между собой сообщениями. Другим ключевым свойством предлагаемой концепции является активное использование таймеров, которые предназначены для привязки работы программы к реальному времени.

Итак, автоматы должны выполняться параллельно. Как достичь данного эффекта без использования многозадачной операционной системы? Все дело в особой структуре автоматной программы.

Рассмотрим структуру программ рассматриваемого класса более подробно. Будем для простоты считать, что каждый конечный автомат (КА) описан в отдельном модуле программы и имеет, как минимум, две внешние функции:

```
void InitFSM(void);  
void ProcessFSM(void);
```

Вместо букв FSM в объявлении представленных функций необходимо подставлять имя автомата. В данном случае, к примеру, функции:

```
void InitPasswordEditor(void);
```

```
void ProcessPasswordEditor (void);
```

будут принадлежать автомату PasswordEditor.

При этом функция InitFSM, как следует из ее названия, будет инициализировать автомат (это что-то вроде конструктора в объектно-ориентированном программировании), а функция ProcessFSM будет отвечать за работу автомата. Главная особенность данной функции: она не должна выполнять продолжительных во времени действий, связанных с ожиданием какого-либо флага или с истечением временного интервала – то есть в ней не должно быть конструкций типа:

```
while(flag==0);
```

или

```
for(i=0;i<delay;i++);
```

В автоматном программировании в таких конструкциях просто нет необходимости.

Задачей главного цикла программы является поочередный вызов функций ProcessFSM всех автоматов, составляющих программу:

```
//main.c
```

```
#include «messages.h» //модуль обработки сообщений
```

```
#include «timers.h» //модуль таймеров
```

```
#include «fsm1.h» //модуль автомата fsm1
```

```
#include «fsm2.h» //модуль автомата fsm2
```

```
#include «fsm3.h» //модуль автомата fsm3
```

```
void main()
```

```
{
```

```
    InitTimers(); // инициализация таймеров
```

```
    InitMessages(); // инициализация механизма обработки сообщений
```

```
    InitFSM1(); // инициализация автомата FSM1
```

```
    InitFSM2(); // инициализация автомата FSM2
```

```
    InitFSM3(); // инициализация автомата FSM3
```

```
    SendMessage(MSG_FSM1_ACTIVATE); // активируем автомат FSM1
```

```
    //главный цикл программы
```

```
    while(1)
```

```

    {
        ProcessFSM1(); // итерация автомата FSM1
        ProcessFSM2(); // итерация автомата FSM2
        ProcessFSM3(); // итерация автомата FSM3
        ProcessMessages(); // обработка сообщений
    };
}

```

Теперь становится понятно, каким образом обеспечивается «многопоточность» системы: в каждой итерации главного цикла поочередно вызываются Process-функции каждого автомата – каждому автомату выделяется время для выполнения какого-либо элементарного действия (или, что тоже может быть, просто для передачи управления далее по списку). То есть если какой либо из автоматов «зависает», то «зависает» вся система. Именно для предотвращения подобной ситуации вводится явный запрет на действия в автоматах, которые занимают продолжительное или неопределенное время.

### Конечные автоматы

Существует три наиболее распространенных типов конечных автоматов (КА, в англ. Finite State Machine – FSM): **автомат Мура**, **автомат Мили** и **смешанный автомат**. Автомат Мура – это КА, выход которого является функцией состояния: выходные воздействия определены в состояниях. Автомат Мили – это автомат, у которого выходные воздействия определены для переходов между состояниями. А смешанный автомат – это автомат, у которого выходные воздействия могут быть определены как в состояниях, так и на переходах.

Теория КА активно развивалась в 50-е – 60-е годы XX века и нашла широкое применение во многих областях техники. Особенно плодотворной она оказалась при разработке трансляторов.

При этом КА рассматривается как своеобразное средство для «перевода» цепочек символов языка А в цепочки символов языка В, а правила перевода задаются структурой КА. Такое применение КА привело к тому, что до настоящего времени наиболее полное изложение теории КА можно найти в учебниках по компиляторам и математической лингвистике.

Достаточно широкое применение КА нашли также в задачах синтеза цифровых устройств, в задачах описания поведения сложных систем (в отрасли связи даже был создан специальный язык описания телекоммуникационных систем, базирующийся на КА).



Прежде чем перейти к рассмотрению программных реализаций конечных автоматов, укажем, что КА имеет входы, выходы и переменную состояния. Схематично представим структуру КА на рисунке 1.

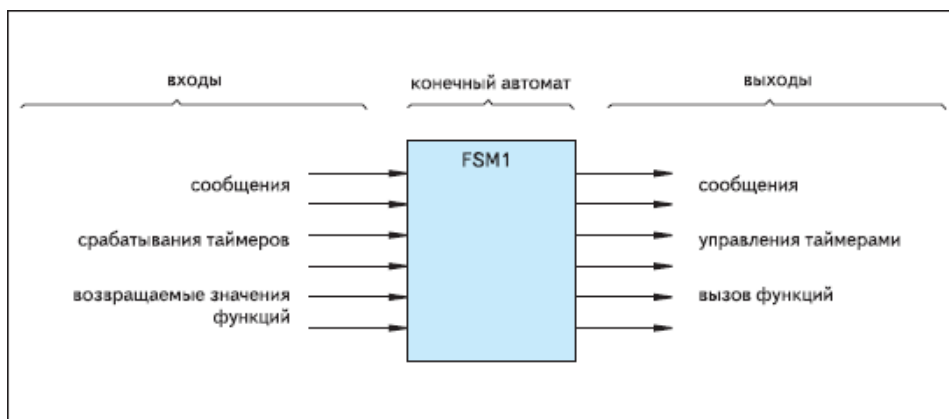


Рисунок 1 Компоненты конечного автомата

Под входом понимается сообщение, срабатывание таймера, результат выражения, которое имеет логическое значение (в том числе возвращаемое функцией логическое значение). Доступ к аппаратным ресурсам микроконтроллера (регистрам периферийных устройств, портам ввода–вывода и тому подобное) выполняется также путем вызова функций.

Под выходом автомата понимается отправка сообщения; запуск, останов или сброс таймера; вызов функции.

При этом выделяется особая переменная состояния – переменная, которая определяет текущее состояние автомата. Эта переменная должна быть доступна только своему автомату, ее изменение из других автоматов недопустимо.

Автомат может осуществлять действия (action) и деятельности (activity) автомата. И деятельность, и действие автомата относятся к его выходной активности.

При этом действием называется выходное воздействие, выполняемое однократно при входе в состояние или на переходе, а деятельностью – выходное воздействие, выполняющееся непрерывно в течение всего времени нахождения автомата в определенном состоянии. Также возможна реализация действий, выполняющихся на выходе из состояния.

Автомат Мура и автомат смешанного типа могут выполнять как действия, так и деятельности, а автомат Мили может выполнять только действия.

## Лекция на тему «Особенности синтаксиса для программ на МК»

### Рассматриваемые вопросы:

1. Базовые понятия для программирования на СИ.
2. Особенности синтаксиса для программ на МК.

### Теоретический материал:

#### Базовые понятия для программирования на СИ

Операторы — символы, при помощи которых производятся какие-либо операции над переменными:

- + - сложение;
- - вычитание;
- \* - умножение;
- / - деление;
- = - присвоение переменной значения.

Например, выражение  $a = b + c$  означает необходимость присвоить переменной  $a$  результат вычисления суммы значений переменных  $b$  и  $c$ .

- ++ - инкремент (увеличение значения переменной на 1);
- - декремент (уменьшение значения переменной на 1);
- == - сравнение, знак «равно» (не путать с присвоением);
- != - сравнение, знак «не равно»;
- < - сравнение, знак «меньше»;
- <= - сравнение, знак «меньше или равно»;
- > - сравнение, знак «больше»;
- >= - сравнение, знак «больше или равно».
- % - остаток от деления;

Например, если  $a = 5$  и  $b = 3$ , то результат вычисления выражения  $a \% b$  будет равно 2 (так как  $5 / 3 = 1$ , а остаток 2).

- << - побитовый сдвиг влево;
- >> - побитовый сдвиг вправо;
- & - логическое «И»;
- | - логическое «ИЛИ»;
- ~ — инвертирование

Циклы с предпроверкой условий (основной цикл программы).

while(условие)

```
{
    тело цикла
}
```

Тело цикла (всё, что в фигурных скобках) выполняется, когда условие истинно (пока условие не станет ложным).

Цикл со счетчиком выполняется определенное количество раз с определенным шагом переменной.

```
for (начальное_значение; цикл_выполняется_до; шаг)
{
    тело цикла,
}
```

где:

**начальное\_значение** – начальное значение счетчика;

**цикл\_выполняется\_до** – указывается до достижения какого значения выполняется цикл;

**шаг** – с каким шагом счетчик считает.

Например:

```
for (i = 0; i < 10; i++)
{
    тело цикла
}
```

Здесь начальное значение переменной *i* равно 0, цикл выполняется, пока значение переменной *i* меньше 10, в каждой итерации цикла значение *i* увеличивается на 1. Так же значение переменной можно изменять прямо в цикле.

Условный переход.

```
if (условие)
{
    тело 1
} else
{
    тело 2
}
```

При условном переходе «тело 1» выполняется, если условие истинно, иначе (если условие ложно) выполняется «тело 2». Существует также следующий вариант условного перехода:

```

if (условие)
{
    тело 1
} elseif (условие 2)
{
    тело 2
}

```

### **Особенности синтаксиса для программ на МК**

Синтаксис программ для микроконтроллеров (МК) на языке Си (C) имеет особенности, связанные с спецификой управления реальным объектом. Эти особенности касаются структуры программы, работы с переменными, использования операторов и написания комментариев. Рассмотрим их подробнее:

#### **1. Структура программы.**

- обязательная главная функция - `main()`. Выполнение программы начинается всегда с неё;
- функции. Каждая функция может вызывать для выполнения другие функции. Отличительный признак — круглые скобки после имени функции. В скобках содержится набор параметров, которые могут передаваться в функцию, и переменных, через которые функция может передавать результаты своей работы во внешнюю функцию;
- фигурные скобки. Отмечают начало последовательности операторов, образующих тело функции, и конец этой последовательности. Фигурные скобки также используются для того, чтобы объединить несколько операторов программы в составной оператор или блок;
- инициализационная часть. Место для размещения действий по подготовке периферии микроконтроллера к работе с заданными параметрами - настройки портов на вход или выход, начальной инициализации таймеров и так далее.

#### **2. Переменные.**

- переменные должны быть объявлены до их использования. Обычно объявления переменных производятся в начале функций, но также можно их объявить в любом месте тела функции;
- при объявлении переменной обязательно нужно указать её тип. В зависимости от типа переменной компилятор резервирует ей место в памяти;
- переменную можно инициализировать - присвоить ей начальное значение.

Например:

```
int a = 100;
```

- переменную можно объявить как немодифицируемую - значение которой в дальнейшем невозможно будет изменить (переменная, доступная только для чтения). Для этого нужно добавить ключевое слово `const` к спецификатору типа.

### 3. Операторы.

- операторы выполняются последовательно в том порядке, в котором они записаны. Но структура программы не обязательно должна быть линейной — при наличии циклов и условий возможны пропуски выполнения отдельных операторов (условия) или многократное выполнение одних и тех же операторов (циклы);

- круглые скобки используются для определения порядка выполнения действий над операндами;

- фигурные скобки следует использовать для объединения некоторого множества операторов в законченный смысловой блок, например в функцию или цикл.

### 4. Комментарии.

- текст комментариев записывается между значками `/* */`. То, что написано между этими значками, компилятору неважно — он игнорирует это;

- комментарии можно размещать как на одной строке с операторами, так и на разных (обычно ниже или выше строки с оператором);

- комментарии могут занимать несколько строк и для них не обязательно наличие в конце точки с запятой;

- внутри комментариев нельзя использовать символы, определяющие начало и конец комментариев.

## **Тема 2.3. Модульное программирование микроконтроллеров STM32**

### **Лекция на тему «Высокоуровневые библиотеки HAL. Синтаксис и шаблоны программ и программных модулей»**

#### **Рассматриваемые вопросы:**

1. Управление GPIO.
2. Отображение периферийных устройств STM32 и дескрипторы HAL.
3. Список основных функций библиотек HAL и CMSIS.

#### **Теоретический материал:**

##### **Управление GPIO**

С появлением «инициативы STCube» компания ST решила полностью обновить уровень аппаратной абстракции (Hardware Abstraction Layer, HAL) для своих микроконтроллеров STM32. До выпуска HAL STCube официальной библиотекой для разработки приложений STM32 долгое время была Стандартная библиотека для работы с периферией (Standard Peripheral Library, SPL). Несмотря на то, что она до сих пор широко распространена среди разработчиков STM32, и вы сможете найти множество примеров использования данной библиотеки в Интернете, HAL STCube является отличным усовершенствованием старой SPL. Фактически, будучи первой библиотекой, разработанной ST, не все части SPL были совместимы между различными семействами STM32, а ранние версии библиотеки содержали множество ошибок. Это стало причиной появления различных альтернативных библиотек.

Поэтому ST полностью переработала HAL и, несмотря на то, что ему все еще необходима небольшая подстройка, ST обеспечивает ему официальную поддержку в будущем. Более того, новый HAL значительно упрощает портирование кода между подсемействами STM32 (F0, F1 и др.), сокращая затраты усилий, необходимых для адаптации вашего приложения к другому микроконтроллеру (без хорошего уровня абстракции совместимость между выводами — это всего лишь маркетинговое преимущество).

##### **Отображение периферийных устройств STM32 и дескрипторы HAL**

Каждое периферийное устройство STM32 соединяется с ядром микроконтроллера некоторым набором шин, как показано на рисунке 1.

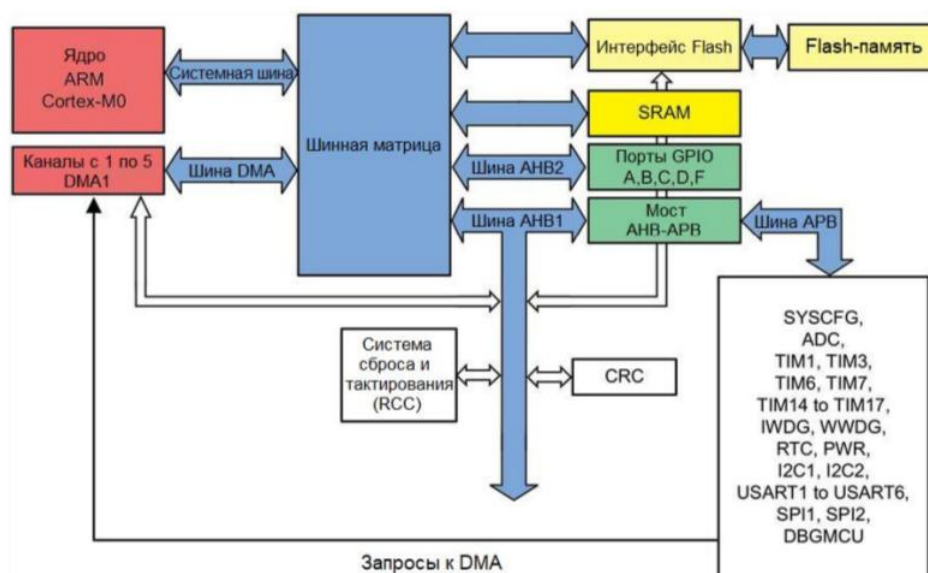


Рисунок 1 Архитектура шин микроконтроллера STM32F030

Системная шина соединяет ядро Cortex-M с шинной матрицей, которая выполняет роль арбитра между ядром и контроллером DMA. И ядро, и контроллер DMA действуют как ведущие (masters).

Шина DMA соединяет ведущий интерфейс DMA продвинутой высокопроизводительной шины (Advanced High-performance Bus, AHB) с шинной матрицей, которая управляет доступом к ЦПУ и доступом контроллера DMA к SRAM, Flash-памяти и периферийным устройствам.

Шинная матрица (BusMatrix) управляет последовательностью доступа между системной шиной ядра и ведущей шиной DMA. Арбитраж производится по алгоритму циклического перебора Round Robin. Шинная матрица состоит из двух ведущих (шины ЦПУ и DMA) и четырех ведомых (slaves): интерфейс Flash-памяти, SRAM, шина AHB1 с мостом AHB-APB (где APB – Advanced Peripheral Bus – продвинутая периферийная шина) и шина AHB2. Периферийные устройства шины AHB подключены к системной шине через шинную матрицу для обеспечения доступа к ядру и к контроллеру DMA.

Мост AHB-APB обеспечивает полностью синхронные соединения между шиной AHB и шиной APB, к которой подключена большая часть периферийных устройств.

Каждая из этих шин подключена к разным источникам тактового сигнала, которые определяют максимальную скорость периферийного устройства, подключенного к какой-либо шине.

Периферийные устройства отображаются на определенную область 4 ГБ адресного пространства, начиная с 0x4000 0000 и продолжая до 0x5FFF FFFF.

Данная область дополнительно разделена на несколько подобластей, на каждую из которых отображается определенное периферийное устройство, как показано на рисунке 2.

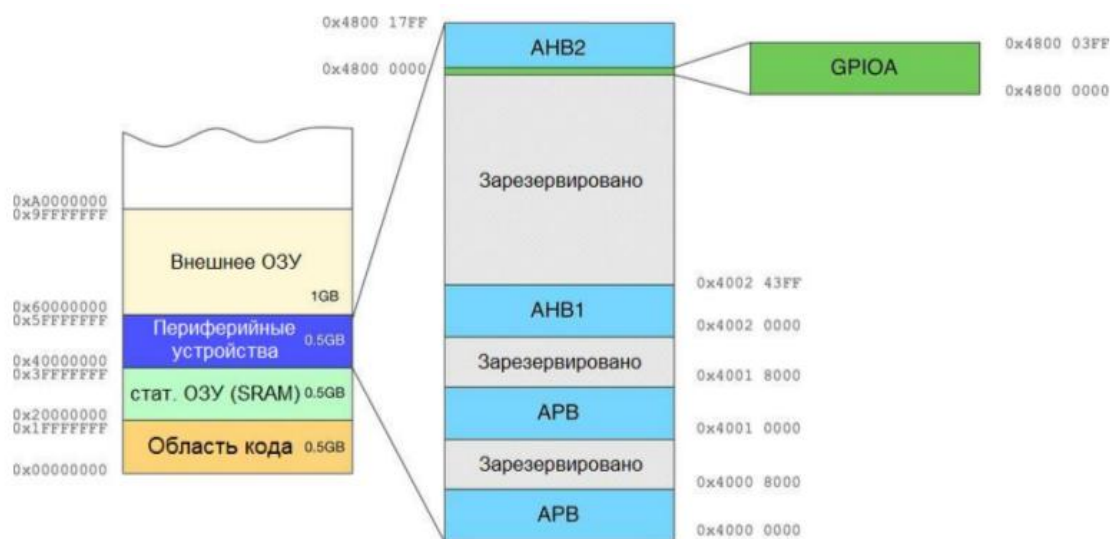


Рисунок 2 Карта памяти областей периферийных устройств для микроконтроллера STM32F030

Способ организации данного пространства, а, следовательно, и способ отображения периферийных устройств зависит от конкретного микроконтроллера STM32. Например, в микроконтроллере STM32F030 шина АНВ2 отображается на область, расположенную между 0x4800 0000 и 0x4800 17FF. Это означает, что данная область размером 6144 Байт. Данная область дополнительно разбита на несколько подобластей, каждая из которых соответствует определенному периферийному устройству. Следуя предыдущему примеру, периферийное устройство GPIOA (которое управляет всеми выводами, подключенными к порту PORT-A) отображается на область, расположенную между 0x4800 0000 и 0x4800 03FF, что означает, что она занимает 1 КБ отображенной периферийной памяти. То, как организовано это отображаемое в памяти пространство, зависит от конкретного периферийного устройства. Таблица 1 показывает организацию памяти периферийного устройства GPIO.

Таблица 1 Карта памяти периферийного устройства GPIO микроконтроллера STM32F030



Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x00	GPIOA_MODER	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]		MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]		
	Reset value	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x00	GPIOx_MODER (where x = B .. F)	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]		MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x04	GPIOx_OTYPER (where x = A .. F)																	OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x08	GPIOx_OSPEEDR (where x = B .. F)	OSPEEDR15[1:0]		OSPEEDR14[1:0]		OSPEEDR13[1:0]		OSPEEDR12[1:0]		OSPEEDR11[1:0]		OSPEEDR10[1:0]		OSPEEDR9[1:0]		OSPEEDR8[1:0]		OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1[1:0]		OSPEEDR0[1:0]		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x0C	GPIOA_PUPDR	PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]		PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]		
	Reset value	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x10	GPIOx_IDR (where x = A .. F)																	IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0	
	Reset value																	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
0x14	GPIOx_ODR (where x = A .. F)																	ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	GPIOx_BSRR (where x = A .. F)	BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0	BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1C	GPIOx_LCKR (where x = A .. B)																LCKK	LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0	
	Reset value																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x20	GPIOx_AFR1 (where x = A .. B)	AFRLAFR7 [3:0]			AFRLAFR6 [3:0]			AFRLAFR5 [3:0]			AFRLAFR4 [3:0]			AFRLAFR3 [3:0]			AFRLAFR2 [3:0]			AFRLAFR1 [3:0]			AFRLAFR0 [3:0]											
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x24	GPIOx_AFRH (where x = A .. B)	AFRHAFR15 [3:0]			AFRHAFR14 [3:0]			AFRHAFR13 [3:0]			AFRHAFR12 [3:0]			AFRHAFR11 [3:0]			AFRHAFR10 [3:0]			AFRHAFR9 [3:0]			AFRHAFR8 [3:0]											
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x28	GPIOx_BRR (where x = A .. F)																	BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0	



HAL_TIM_Base_Start_IT(&htim)	Запуск таймера
HAL_TIM_Base_Stop_IT(&htim)	Остановка таймера
TIMx ->ARR = value	Установка значения value в регистр переполнения таймера
HAL_ADC_Start(&hadc)	Запуск преобразования АЦП
HAL_ADC_PollForConversion(&hadc,Timeout)	Ожидание преобразования АЦП
HAL_ADC_GetValue(&hadc)	Получение результата преобразования АЦП
HAL_ADC_Stop(&hadc)	Остановка АЦП
ADCx -> SQR3 = n	Смена канала преобразования АЦП на n
HAL_SPI_Transmit(&hspi, *pData, Size, TimeOut)	Отправка данных (массив pData) в количестве Size по SPI
HAL_I2C_Master_Transmit(&hi2c, Address, *pData, Size, Timeout)	Отправка данных (массив pData) в количестве Size по адресу Address по I2C
HAL_I2C_Master_Receive(&hi2c, Address, *pData, Size, Timeout)	Чтение данных (с сохранением в массив pData) по I2C

## **Лекция на тему «Среда программирования CubeIDE. Структура проекта»**

### **Рассматриваемые вопросы:**

1. Элементы интерфейса STM32CubeIDE.
2. Этапы создания проекта.
3. Структура сгенерированного проекта.

### **Теоретический материал:**

#### **Элементы интерфейса STM32CubeIDE**

STM32CubeIDE - это продвинутая платформа разработки C / C ++ с IP-конфигурацией микроконтроллера STM32, генерацией кода, компиляцией кода и функциями отладки.

Главные особенности:

##### **1. Интегрированный STM32CubeMX:**

CubeMX – это инструмент, используемый для конфигурации микроконтроллера, выбранного для проекта. Он используется как для выбора правильных аппаратных подключений, так и для генерации кода, необходимого для конфигурации HAL от ST.

CubeMX является приложением, ориентированным на микроконтроллеры. Это означает, что все действия, выполняемые инструментом, основаны на:

- семействе микроконтроллеров STM32 (F0, F1 и другие);
- типе корпуса, выбранного для вашего устройства (LQFP48, BGA144 и другие);
- аппаратной периферии, которая нам нужна в проекте (USART, SPI и так далее);
- общих конфигурациях микроконтроллера (например, тактирование, управление питанием, контроллер NVIC и так далее).

В дополнение к функциям, касающимся аппаратного обеспечения, CubeMX также может работать со следующими программными аспектами:

- управление HAL от ST для выбранного семейства микроконтроллеров (CubeF0, CubeF1 и так далее);
- дополнительные функциональные возможности библиотек программного обеспечения, необходимые в нашем проекте (библиотека FatFs, FreeRTOS и так далее).

2. Основан на Eclipse™ / CDT, поддерживает плагин ECLIPSE™, GNU C / C++ в цепочке инструментов ARM® и отладчик GDB.

3. Другие расширенные функции отладки:

- ядро ЦП и просмотр памяти;
- просмотр переменных в реальном времени;
- системный анализ и отслеживание в реальном времени (SWV);
- инструмент анализа отказов ЦП.

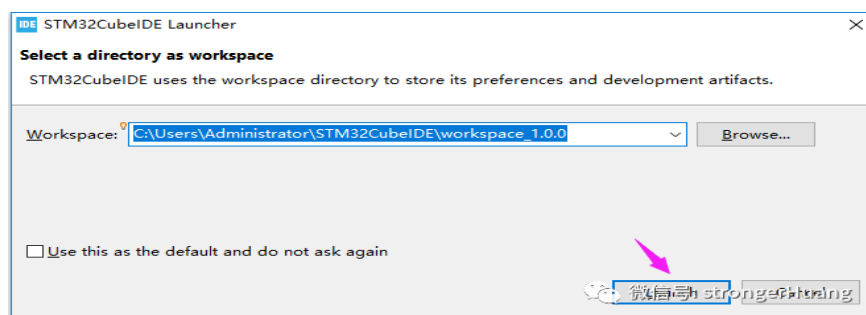
4. Поддержка отладки ST-LINK и J-Link.

5. Импортируйте проекты из TrueSTUDIO®.

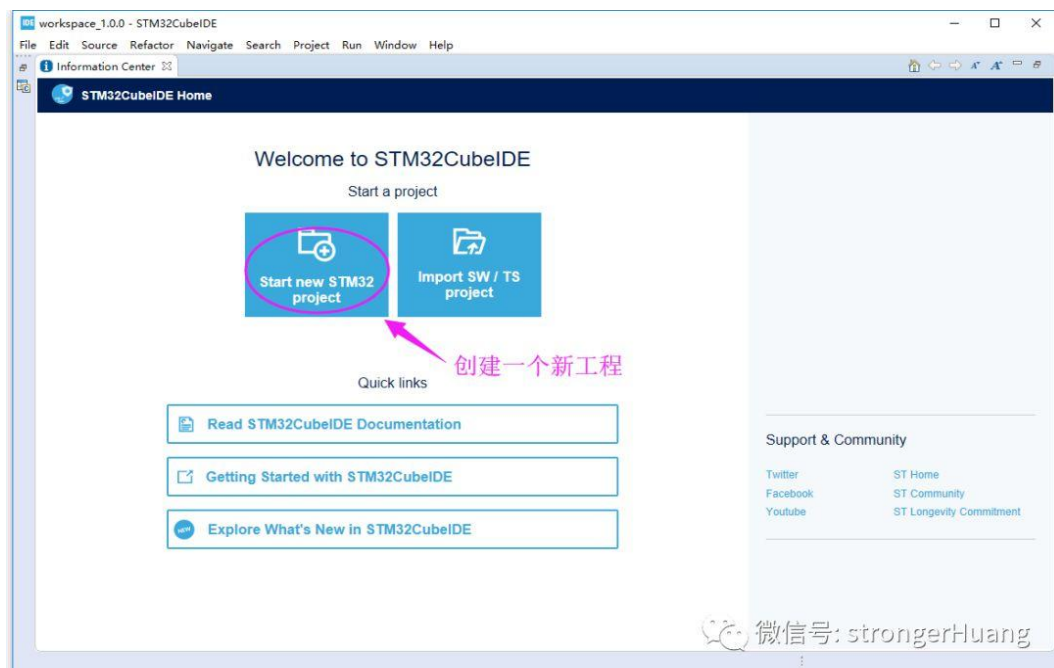
6. Поддержка операционных систем: Windows®, Linux® и MacOS®.

### Этапы создания проекта:

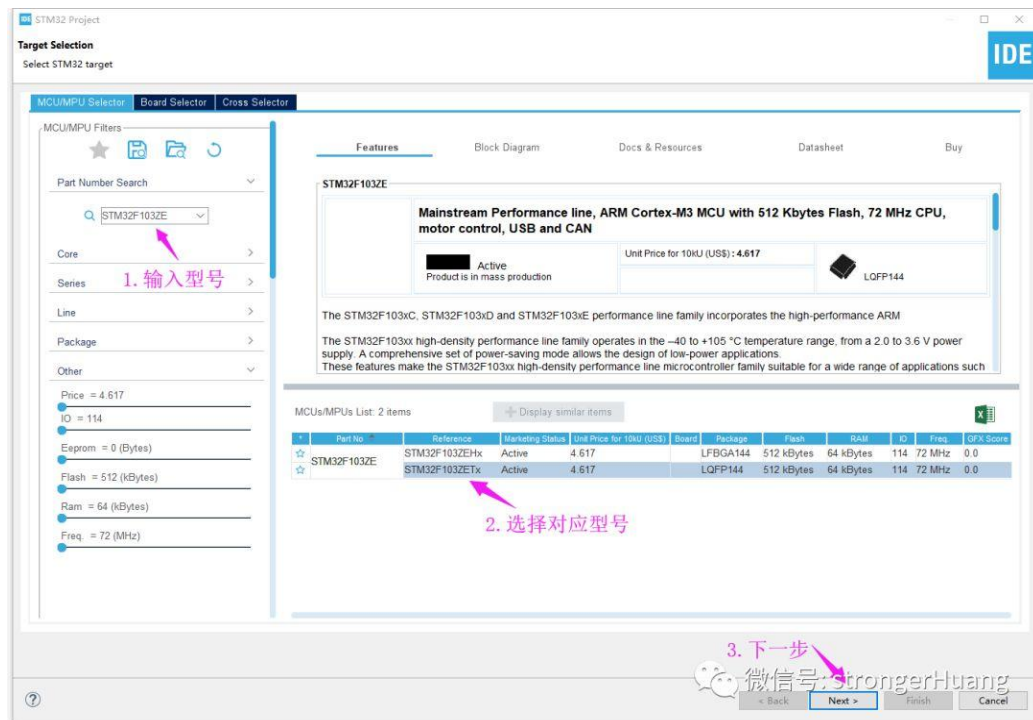
1. Откройте STM32CubeIDE, выберите путь к рабочей области для сохранения (по умолчанию ОК):



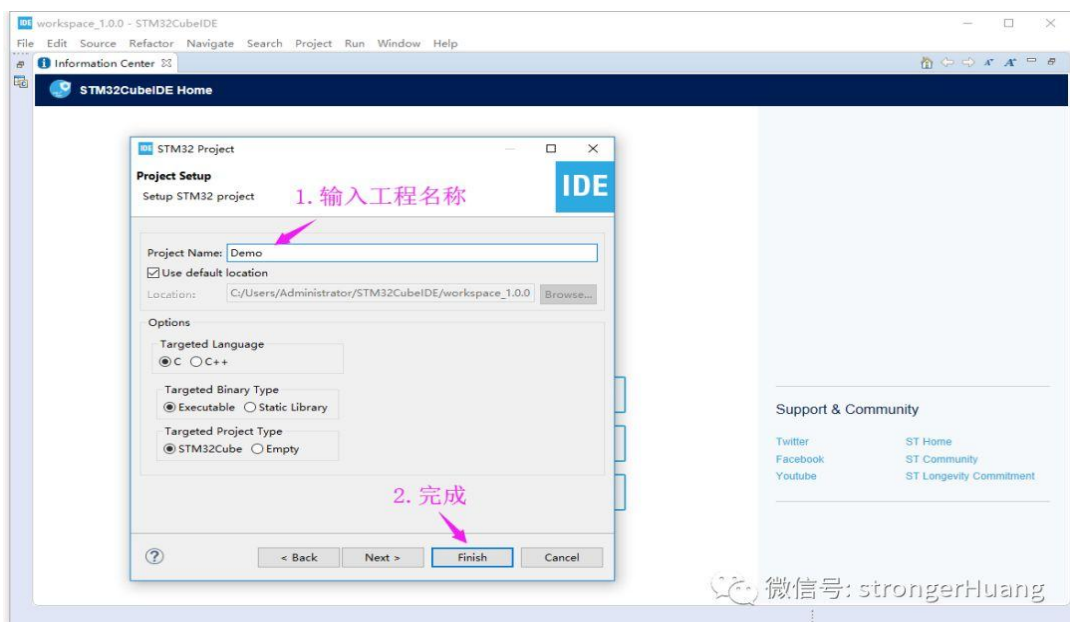
2. Создайте новый проект.



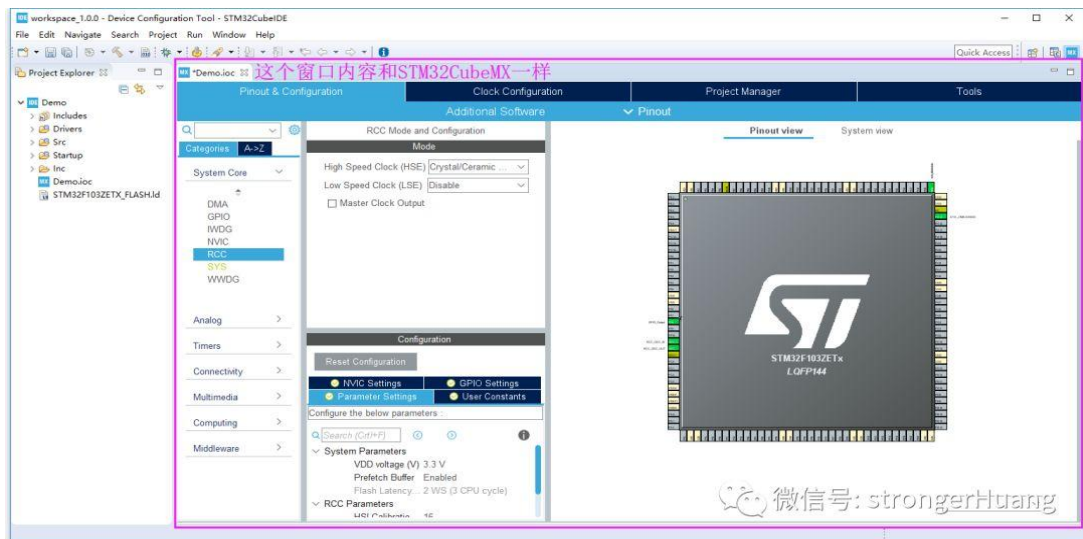
### 3. Выберите модель



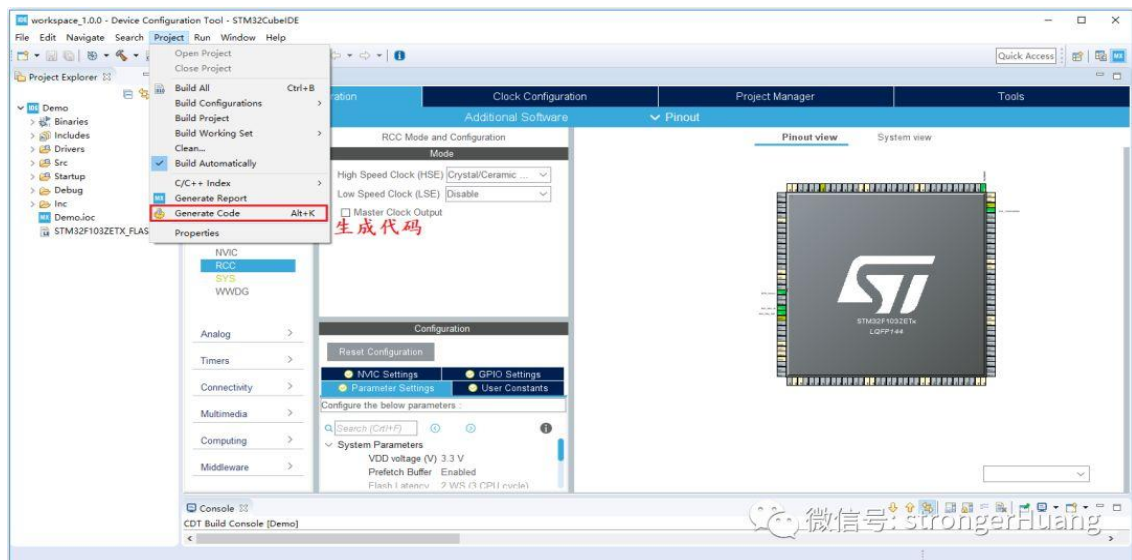
### 4. Введите название проекта.



### 5. Настройте STM32CubeMX.

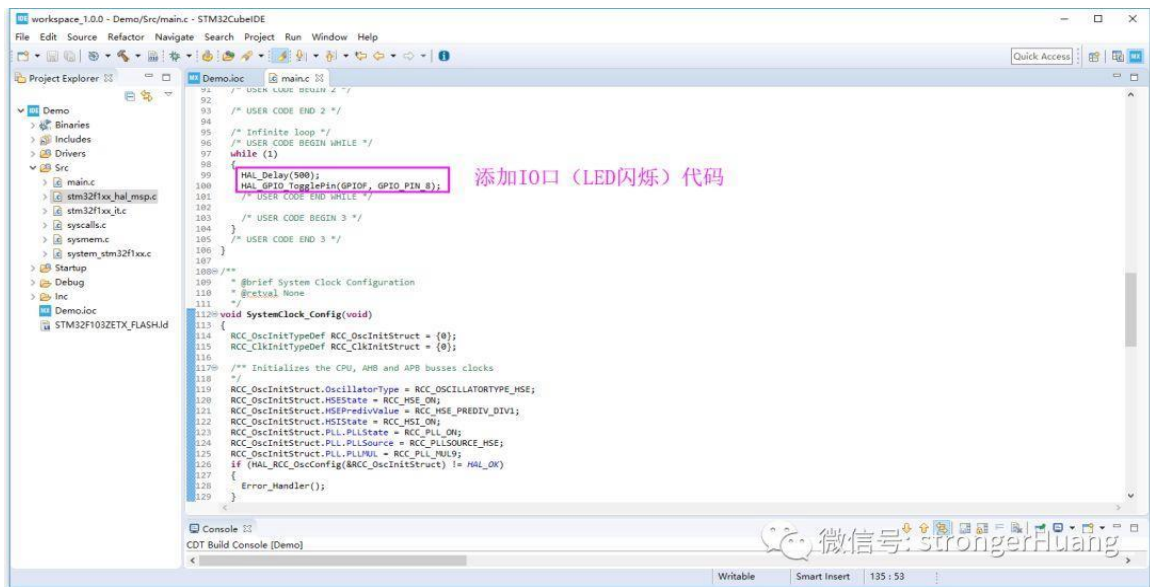


6. Сгенерируйте код с помощью кнопки быстрого доступа или меню.

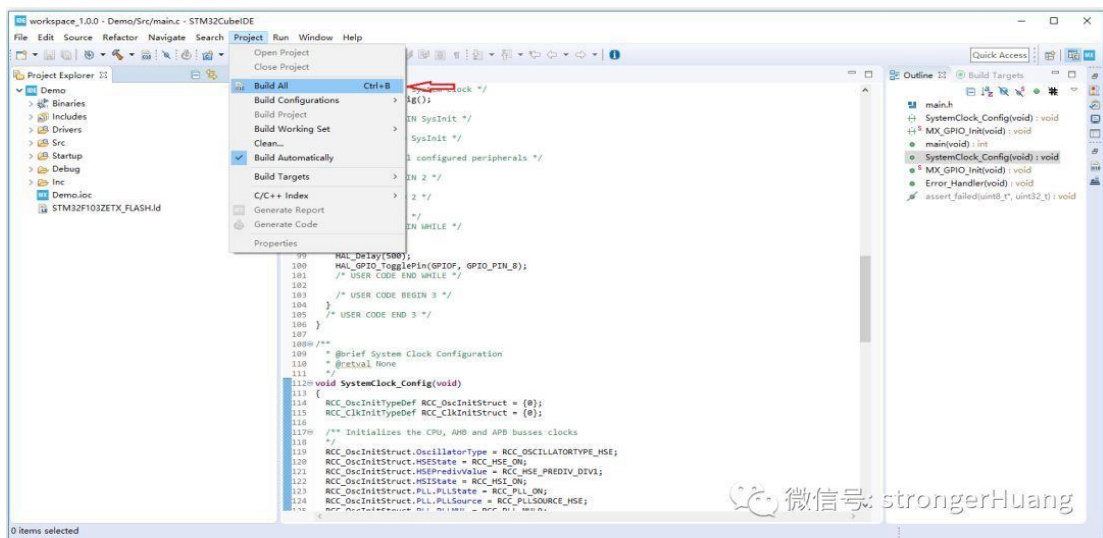


7. Добавьте код.

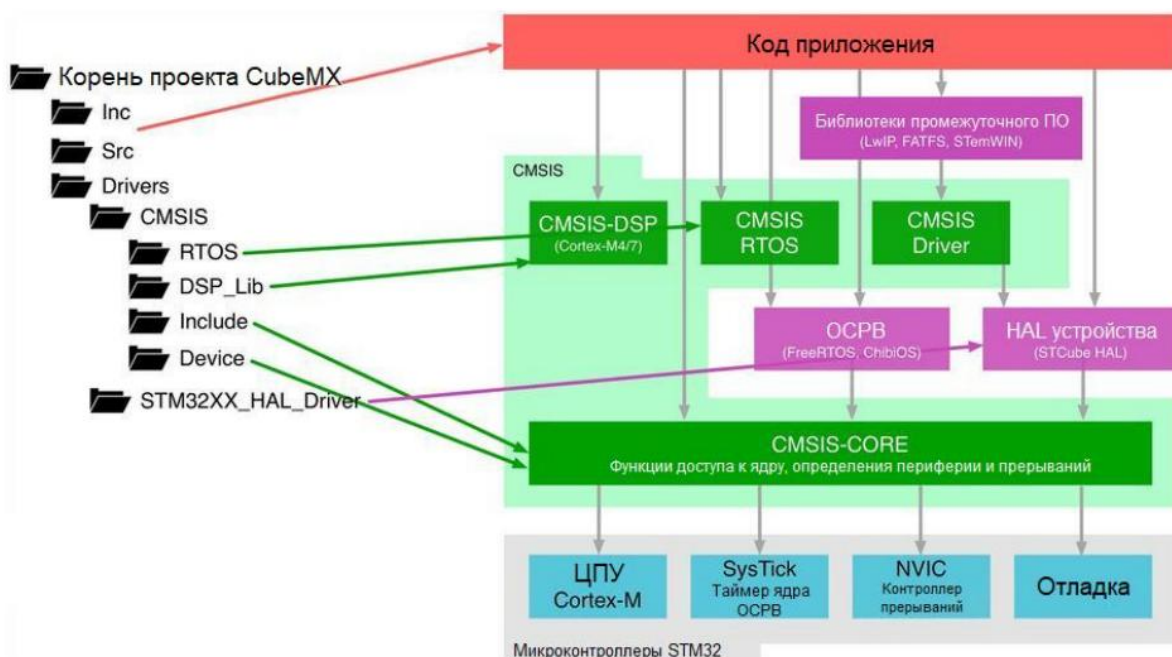




## 8. Скомпилируйте проект.



## Структура сгенерированного проекта



CMSIS-CORE реализует базовую систему поддержки исполнения программ (run-time system) для устройства Cortex-M и предоставляет пользователю доступ к ядру процессора и периферии устройства. Если подробнее, он определяет:

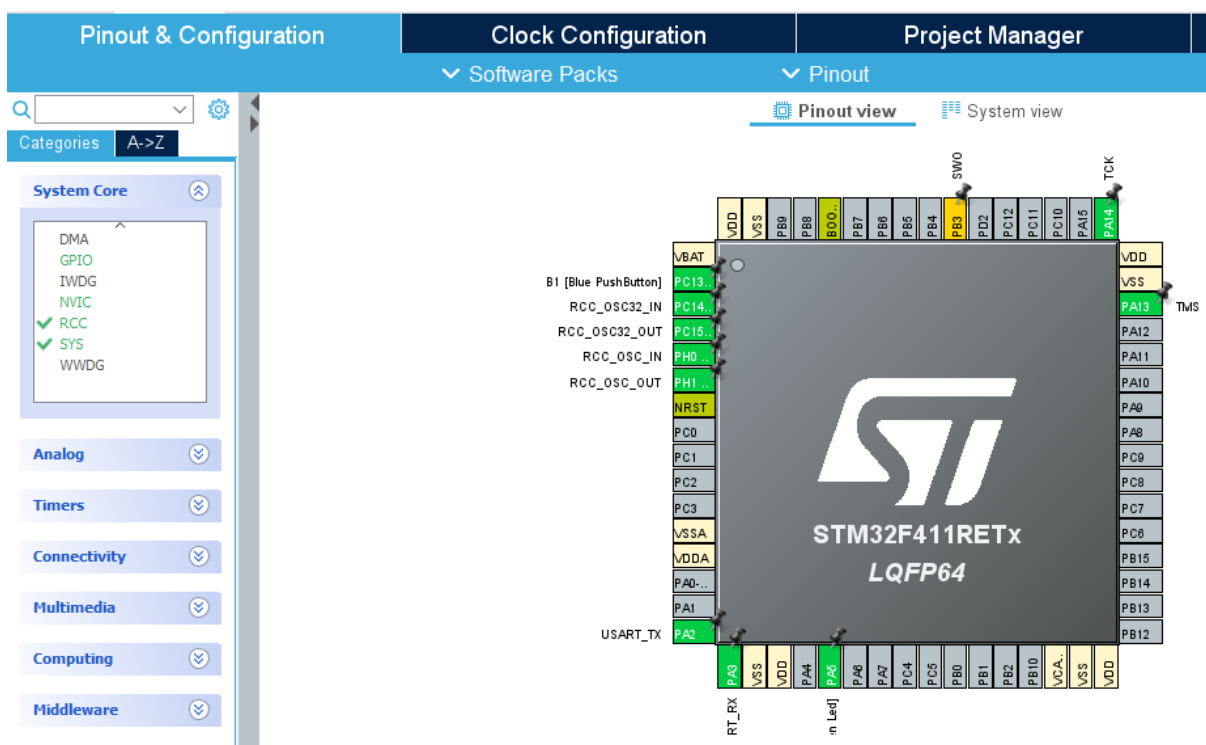
- HAL для регистров процессора Cortex-M со стандартизованными определениями для регистров SysTick, регистров NVIC, регистров блока управления системой (SCB), регистров MPU, регистров FPU и функций доступа к ядру;
- имена системных исключений для взаимодействия с системными исключениями без проблем с совместимостью;
- методы организации заголовочных файлов, облегчающие изучение новых микроконтроллеров Cortex-M и улучшающие переносимость программного обеспечения. Они включают в себя соглашения об именах для прерываний, специфичных для устройства;
- методы инициализации системы, которые будут использоваться каждым производителем микроконтроллеров. Например, стандартизированная функция SystemInit() необходима для конфигурации системы тактирования при запуске устройства;
- встроенные функции, используемые для генерации инструкций ЦПУ, которые не поддерживаются стандартными функциями Си;
- глобальную переменную с именем SystemCoreClock, позволяющую легко определять частоту системного тактового сигнала.

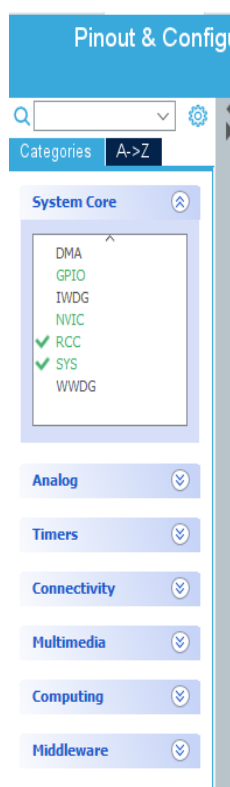
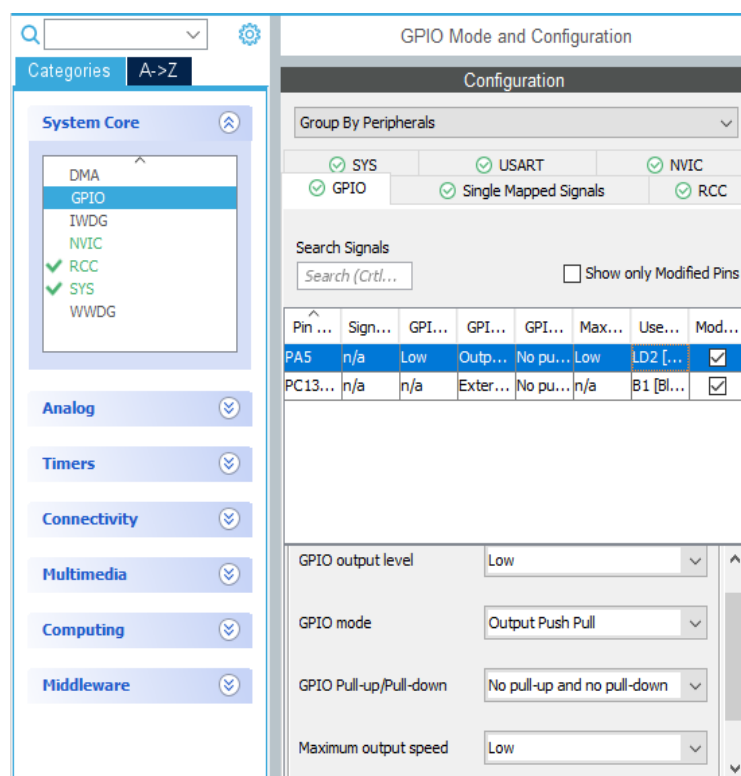
Папка Include содержит несколько файлов core\_.h (где заменяется на stm0, stm3 и т. д.). Данные файлы определяют основные периферийные устройства и предоставляют вспомогательные функции, которые обращаются к основным регистрам.

Папка Device содержит специфическую информацию об устройстве для всех микроконтроллеров STM32F/L, такую как номера запросов прерываний (IRQn) для всех исключений и прерываний устройства, определения для периферийного доступа (Peripheral Access) ко всей периферии устройства (все структуры данных и отображение адресов для периферийных устройств конкретного устройства) – файл system\_.h. Она также содержит дополнительные вспомогательные функции для упрощения программирования периферийных устройств. Кроме того, существует также несколько ассемблерных startup-файлов startup\_.s: они содержат код начального запуска и код конфигурации системы.

Папки Inc и Src в корневом каталоге проекта содержат заголовочные файлы и файлы с исходным кодом «скелета» приложения, сгенерированного CubeMX, а папка STM32xxxx\_HAL\_Driver, находящаяся внутри папки Drivers, является всем HAL от ST для используемой серии микроконтроллеров.

Представление Pinout («Распиновка») разделено на две части:





Выводы, окрашенные в ярко-зеленый цвет, включены. Это означает, что CubeMX сгенерирует необходимый код для конфигурации данного вывода в соответствии с его функциональными возможностями.



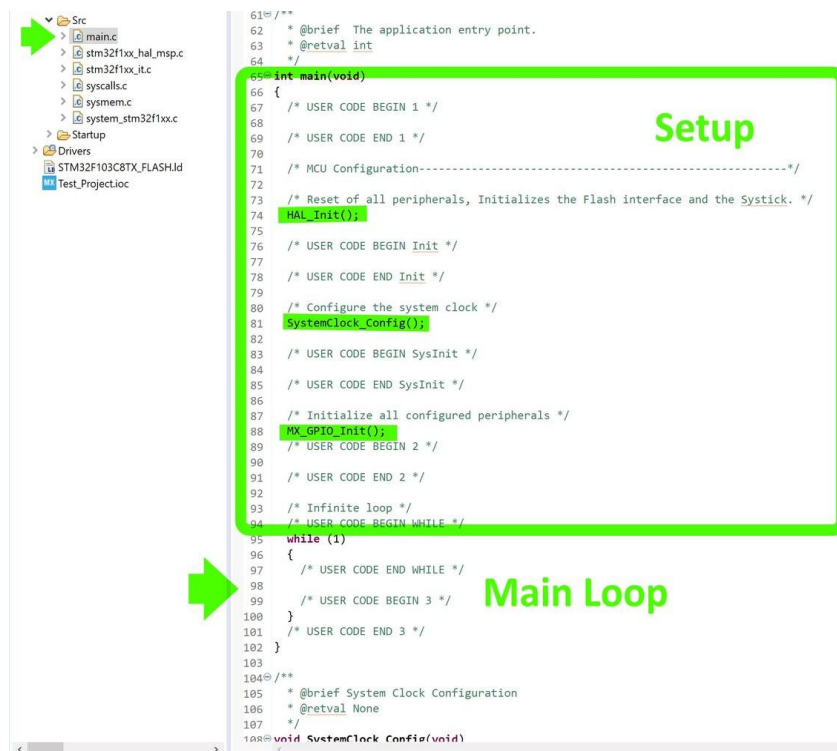
```

138 /**
139  * @brief GPIO Initialization Function
140  * @param None
141  * @retval None
142  */
143 static void MX_GPIO_Init(void)
144 {
145     GPIO_InitTypeDef GPIO_InitStructure = {0};
146
147     /* GPIO Ports Clock Enable */
148     __HAL_RCC_GPIOC_CLK_ENABLE();
149     __HAL_RCC_GPIOD_CLK_ENABLE();
150
151     /*Configure GPIO pin Output Level */
152     HAL_GPIO_WritePin(LED13_GPIO_Port, LED13_Pin, GPIO_PIN_RESET);
153
154     /*Configure GPIO pin : LED13_Pin */
155     GPIO_InitStructure.Pin = LED13_Pin;
156     GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
157     GPIO_InitStructure.Pull = GPIO_NOPULL;
158     GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;
159     HAL_GPIO_Init(LED13_GPIO_Port, &GPIO_InitStructure);
160 }
161
162 /* USER CODE BEGIN 4 */
163
164 /* USER CODE END 4 */
165
166 /**
167  * @brief This function is executed in case of error occurrence.
168  * @retval None
169  */
170

```

CubeMX

Функция main (void) и бесконечный цикл в ней while (1) – основной раздел для написания программного кода.



```

61 /**
62  * @brief The application entry point.
63  * @retval int
64  */
65 int main(void)
66 {
67     /* USER CODE BEGIN 1 */
68
69     /* USER CODE END 1 */
70
71     /* MCU Configuration-----*/
72
73     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
74     HAL_Init();
75
76     /* USER CODE BEGIN Init */
77
78     /* USER CODE END Init */
79
80     /* Configure the system clock */
81     SystemClock_Config();
82
83     /* USER CODE BEGIN SysInit */
84
85     /* USER CODE END SysInit */
86
87     /* Initialize all configured peripherals */
88     MX_GPIO_Init();
89     /* USER CODE BEGIN 2 */
90
91     /* USER CODE END 2 */
92
93     /* Infinite loop */
94     /* USER CODE BEGIN WHILE */
95     while (1)
96     {
97         /* USER CODE END WHILE */
98
99         /* USER CODE BEGIN 3 */
100
101         /* USER CODE END 3 */
102     }
103
104 /**
105  * @brief System Clock Configuration
106  * @retval None
107  */
108 void SystemClock_Config(void)

```

Setup

Main Loop

MX\_UART6\_Init(), инициализирует USART6 (взаимодействие с периферийными устройствами) в соответствии с настройками.

```

static void MX_USART6_UART_Init(void)
{
    huart6.Instance = USART6;
    huart6.Init.BaudRate = 115200;
    huart6.Init.WordLength = UART_WORDLENGTH_8B;
    huart6.Init.StopBits = UART_STOPBITS_1;
    huart6.Init.Parity = UART_PARITY_NONE;
    huart6.Init.Mode = UART_MODE_TX_RX;
    huart6.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart6.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart6) != HAL_OK)
    {
        Error_Handler();
    }
}

```

## **Лекция на тему «Память МК. Работа с модулем МК в программе»**

### **Рассматриваемые вопросы:**

1. Модель организации памяти в STM32.
2. Основы процессов компиляции и компоновки.

### **Теоретический материал:**

#### **Модель организации памяти в STM32**

Микроконтроллеры STM32 организуют адресное пространство памяти объемом 4 ГБ. Первые 0,5 ГБ памяти выделяются под область кода. В свою очередь, данная область подразделяется на несколько подобластей. Наиболее важная из них, начинающаяся с адреса 0x0800 0000, предназначена для отображения внутренней Flash-памяти. Вместо этого внутренняя память SRAM начинается с адреса 0x2000 0000 и состоит из нескольких подобластей, предназначенных для определенных задач, которые мы вскоре увидим.

На рисунке 1 показана типовая организация Flash-памяти и памяти SRAM (статическое ОЗУ) в микроконтроллере STM321. Начальные байты Flash памяти выделены под указатель основного стека (Main Stack Pointer, MSP) и таблицу векторов. MSP содержит адрес начала стека. Архитектура Cortex-M дает максимальную свободу в размещении стека в памяти SRAM, а также в других типах внутренней (например, RAM CCM, доступной в некоторых микроконтроллерах STM32) или внешней (подключенной к контроллеру FSMC) памяти. Это объясняет необходимость MSP.

Flash-память также может использоваться для хранения данных только для чтения (данные RO), также известных как неизменяемые данные (const data), поскольку переменные, объявленные как const, автоматически размещаются в этой памяти. Наконец, Flash память содержит ассемблерный код, сгенерированный из исходного кода Си.



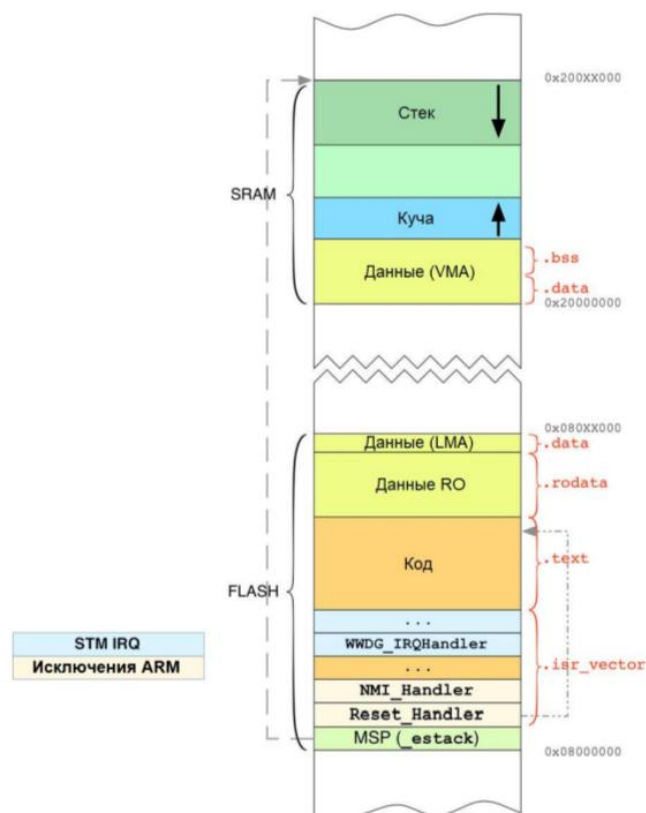


Рисунок 1 Типовая организация Flash-памяти и памяти SRAM

Память SRAM также организована в виде нескольких подобластей. Область переменного размера, начинающаяся с конца SRAM и растущая вниз (то есть ее базовый адрес – наибольший адрес в SRAM), выделяется под стек (stack). Это происходит потому, что ядра Cortex-M используют модель стековой памяти, которая называется нисходящим стеком с указателем на занятый элемент (full-descending stack). Базовый указатель стека, также называемый указателем основного стека (MSP), вычисляется на этапе компиляции и сохраняется по адресу 0x0800 0000 во Flash-памяти. Как только мы вызываем функцию, в стек помещается новый кадр стека (stack frame). Это означает, что указатель на текущий кадр стека (SP) автоматически декрементируется при каждом вызове функции (например, инструкция push ассемблера ARM автоматически декрементирует его).

SRAM также используется для хранения изменяемых данных (variable data), и данная область обычно начинается в начале SRAM (0x2000 0000). В свою очередь, данная область делится между инициализированными и неинициализированными данными. Для понимания разницы между ними, давайте рассмотрим следующий фрагмент кода:

```
...
uint8_t var1 = 0xEF;
```

```
uint8_t var2;
```

```
...
```

`var1` и `var2` – две глобальные переменные. `var1` является инициализированной переменной (мы фиксируем ее начальное значение во время компиляции), в то время как значение `var2` неинициализировано: среда выполнения может инициализировать ее как ноль. По той же причине у нас есть две секции `.data`: одна хранится во Flash-памяти, а другая – в оперативной памяти, как мы увидим далее.

Наконец, память SRAM может содержать еще одну растущую область: кучу (heap). В ней хранятся переменные, которые выделяются динамически во время выполнения микропрограммы (с помощью процедуры Си `malloc()` или аналогичной). Данная область, в свою очередь, может быть организована в несколько подобластей в соответствии с используемым распределителем памяти или аллокатором, англ. `allocator`. Куча растет вверх (то есть базовый адрес является наименьшим в своей области) и имеет фиксированный максимальный размер.

С точки зрения компилятора, эти секции традиционно называются по-разному внутри бинарного файла приложения. Например, секция, содержащая ассемблерный код, называется `.text`, `.rodata` – секция, содержащая неизменяемые переменные и строки, а секция с инициализированными данными – `.data`. Данные имена также являются общими для других компьютерных архитектур, таких как x86 и MIPS. Другие же специфичны для «мира микроконтроллеров». Например, секция `.isr_vector` предназначена для хранения таблицы векторов в микроконтроллерах на базе Cortex-M.

Поскольку каждый микроконтроллер STM32 имеет свое собственное количество SRAM и Flash-памяти, и поскольку каждая программа имеет разное количество команд и переменных, то размеры и расположение этих секций в памяти различаются. Прежде чем мы увидим, как дать указание компилятору сгенерировать бинарный файл для конкретного микроконтроллера, мы должны понять все шаги и инструменты, задействованные во время генерации объектных файлов (object files).

### **Основы процессов компиляции и компоновки**

Процесс, начинающийся с компиляции исходного кода Си и заканчивающийся генерацией конечного бинарного образа для загрузки на наш микроконтроллер, включает в себя несколько шагов и инструментов, предоставляемых инструментарием GCC. Рисунок 2 пытается обрисовать в общих чертах данный процесс. Все начинается с файлов с исходным кодом Си. Обычно они содержат следующие программные структуры:

- глобальные переменные: их можно в свою очередь разделить на неинициализированные и инициализированные переменные; глобальная переменная также может быть определена как статическая (`static`), то есть ее видимость ограничена текущим файлом с исходным кодом;
- локальные переменные: их можно разделить между простыми локальными (также называемыми автоматическими) переменными и статическими локальными переменными (то есть теми переменными, время жизни которых длится все время выполнения программы);
- неизменяемые (постоянные) данные: они могут быть в свою очередь разделены на константные типы данных (например, `const int c = 5`) и строковые константы (например, `"Hello World!"`);
- процедуры (подпрограммы): они формируют программу и будут транслированы в ассемблерные инструкции;
- внешние ресурсы: это глобальные переменные (объявленные как `extern`) и процедуры, определенные в других исходных файлах. Задачей компоновщика будет «связать» ссылки на их символьные имена, определенные в других файлах с исходным кодом, и объединить секции, поступающие из соответствующих бинарных файлов.

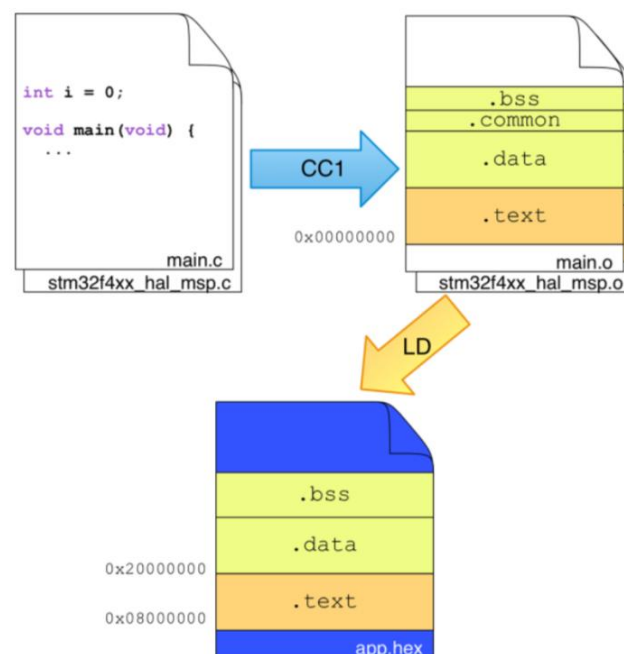


Рисунок 2 Процесс компиляции из файла с исходным кодом в конечный бинарный образ

Как только файл с исходным кодом скомпилирован, вышеприведенные программные структуры отображаются в определенных секциях бинарного файла. В таблице 1 приведены наиболее важные из них.

Таблица 1 Отображение структур программы и секций бинарных файлов

Структура языка	Секция бинарного файла	Область памяти при выполнении
Глобальные неинициализированные переменные	.common	Данные (SRAM)
Глобальные инициализированные переменные	.data	Данные (SRAM+Flash)
Глобальные статические неинициализированные переменные	.bss	Данные (SRAM)
Глобальные статические инициализированные переменные	.data	Данные (SRAM+Flash)
Локальные переменные	<не определена>	Стек или куча (SRAM)
Локальные статические неинициализированные переменные	.bss	Данные (SRAM)
Локальные статические инициализированные переменные	.data	Данные (SRAM+Flash)
Неизменяемые (постоянные) типы данных	.rodata	Код (Flash)
Неизменяемые (постоянные) строки	.rodata.1	Код (Flash)
Процедуры	.text	Код (Flash)

Для каждого файла с исходным кодом (.c), составляющего наше приложение, компилятор сгенерирует соответствующий объектный файл (.o), который содержит секции из таблицы 1. Объектный файл – это тип бинарного файла, который придерживается широко известному стандарту. Существует множество стандартов для

бинарных файлов (PE, COFF, ELF и т. д.). GCC ARM использует ELF32 – достаточно популярный открытый стандарт благодаря его использованию в операционных системах на основе Linux и широко поддерживаемый прочими инструментами, такими как OpenOCD и STM32CubeProgrammer. Однако файлы, оканчивающиеся на .o, представляют собой особый тип объектных файлов. Они также известны как перемещаемые файлы (relocatable files). Это название происходит от того факта, что все адреса в памяти, содержащиеся в этом типе файла, относительные и начинаются с адреса 0x0000 0000. Это также означает, что секция .text будет начинаться с этого адреса, а мы знаем, что он отличается от начального адреса Flash-памяти (0x0800 0000) в микроконтроллере STM32.

Начиная с последовательности перемещаемых файлов (плюс некоторые другие конфигурационные файлы, которые мы увидим через некоторое время), компоновщик будет собирать их содержимое в один общий объектный файл, который будет представлять собой нашу микропрограмму для загрузки в микроконтроллер. В этом процессе, называемом компоновкой (linking), компоновщик переместит (relocate) все относительные адреса в фактические адреса в памяти. Этот тип файла также известен как абсолютный файл (absolute file), потому что все адреса являются абсолютными и специфичными для используемого микроконтроллера STM32.

Как компоновщик знает, где разместить в памяти секции, содержащиеся в абсолютном файле? Именно благодаря скриптам компоновщика, англ. linker scripts, (файлы, оканчивающиеся на .ld) мы можем выстроить содержимое абсолютного файла в соответствии с фактической организацией памяти. CubeMX также встраивает правильный скрипт компоновщика для нашего микроконтроллера в сгенерированный проект Си (он содержится во вложенной папке SW4STM32). Однако довольно сложно изучить содержимое этих скриптов, если мы не освоим несколько концепций, оговоренных ранее. Поэтому лучше всего начать постепенно создавать голый фундамент приложения STM32.

## Лекция на тему «Подсистема ввода/вывода МК»

### Рассматриваемые вопросы:

1. Конфигурация GPIO.
2. Режимы работы GPIO.
3. Управление GPIO.
4. Деинициализация GPIO.

### Теоретический материал:

#### Конфигурация GPIO

Каждый микроконтроллер STM32 имеет разное количество программируемых вводов/выводов общего назначения. Точное их число зависит от:

- типа выбранного корпуса (LQFP48, BGA176 и так далее);
- семейства микроконтроллеров (F0, F1 и другие);
- использования внешних генераторов HSE и LSE.

Вводы/выводы общего назначения (General Purpose Input/Output, GPIO) являются способом связи микроконтроллера с внешним миром. Каждая плата использует разное число вводов/выводов (I/O) для управления внешними периферийными устройствами (например, светодиодом) или для обмена данными посредством нескольких типов коммуникационных периферийных устройств (UART, USB, SPI и др.). Каждый раз, когда нам нужно сконфигурировать периферийное устройство, использующее выводы микроконтроллера, нам необходимо сконфигурировать соответствующие GPIO, используя модуль HAL\_GPIO.

Как было сказано ранее, HAL спроектирован таким образом, что он абстрагируется от конкретного отображения периферийной памяти. В то же время он также предоставляет общий и более удобный для пользователя способ конфигурации периферийного устройства без необходимости программисту вникать в детали конфигурации его регистров.

Для конфигурации GPIO мы используем функцию HAL\_GPIO\_Init(GPIO\_TypeDef \*GPIOx, GPIO\_InitTypeDef \*GPIO\_Init).

GPIO\_InitTypeDef – это структура Си, используемая для конфигурации GPIO, и она определена следующим образом:

```
typedef struct {  
    uint32_t Pin;  
    uint32_t Mode;
```

```

uint32_t Pull;
uint32_t Speed;
uint32_t Alternate;
} GPIO_InitTypeDef;

```

Назначение каждого поля структуры:

- Pin: это номера выводов, начиная с 0, которые мы собираемся сконфигурировать. Например, для вывода PA5 оно принимает значение GPIO\_PIN\_5 . Мы можем использовать один и тот же экземпляр структуры GPIO\_InitTypeDef для одновременной конфигурации нескольких выводов, выполняя побитовое ИЛИ (например, GPIO\_PIN\_1 | GPIO\_PIN\_5 | GPIO\_PIN\_6);
- Mode: это режим работы вывода, и он может принимать одно из значений из таблицы 2. Подробнее об этом поле ниже;
- Pull: определяет, активировать ли подтяжку к питанию (Pull-up) или к земле (PullDown) для выбранных выводов в соответствии с таблицей 3;
- Speed: определяет скорость выводов;
- Alternate: определяет, какое периферийное устройство связать с выводом.

Таблица 2 Доступные режимы GPIO\_InitTypeDef.Mode для GPIO

Режим вывода	Описание
GPIO_MODE_INPUT	Режим плавающего входа (Input Floating)
GPIO_MODE_OUTPUT_PP	Режим двухтактного выхода (Output Push Pull)
GPIO_MODE_OUTPUT_OD	Режим выхода с открытым стоком (Output Open Drain)
GPIO_MODE_AF_PP	Режим двухтактной альтернативной функции (Alternate Function Push Pull)
GPIO_MODE_AF_OD	Режим альтернативной функции с открытым стоком (Alternate Function Open Drain)
GPIO_MODE_ANALOG	Режим аналогового вывода (Analog)
GPIO_MODE_IT_RISING	Режим внешнего прерывания при обнаружении перепада переднего (нарастающего) фронта сигнала (External Interrupt Mode with Rising edge trigger detection)
GPIO_MODE_IT_FALLING	Режим внешнего прерывания при

	обнаружении перепада заднего (спадающего) фронта сигнала (External Interrupt Mode with Falling edge trigger detection)
GPIO_MODE_IT_RISING_FALLING	Режим внешнего прерывания при обнаружении перепада переднего или заднего фронта сигнала (External Interrupt Mode with Rising/Falling edge trigger detection)
GPIO_MODE_EVT_RISING	Режим события при обнаружении перепада переднего (нарастающего) фронта сигнала (External Event Mode with Rising edge trigger detection)
GPIO_MODE_EVT_FALLING	Режим события при обнаружении перепада заднего (спадающего) фронта сигнала (External Event Mode with Falling edge trigger detection)
GPIO_MODE_EVT_RISING_FALLING	Режим события при обнаружении перепада переднего или заднего фронта сигнала (External Event Mode with Rising/Falling edge trigger detection)

Таблица 3 Доступные режимы GPIO\_InitTypeDef.Pull для GPIO

Режим вывода	Описание
GPIO_NOPULL	Отключить подтяжку вывода (Pull-up или Pull-down)
GPIO_PULLUP	Активация подтяжки к питанию (Pull-up)
GPIO_PULLDOWN	Активация подтяжки к земле (Pull-down)

### Режимы работы GPIO

Микроконтроллеры STM32 обеспечивают по-настоящему гибкое управление GPIO. На рисунке 4 показано аппаратное устройство одиночного вывода микроконтроллера STM32F030.



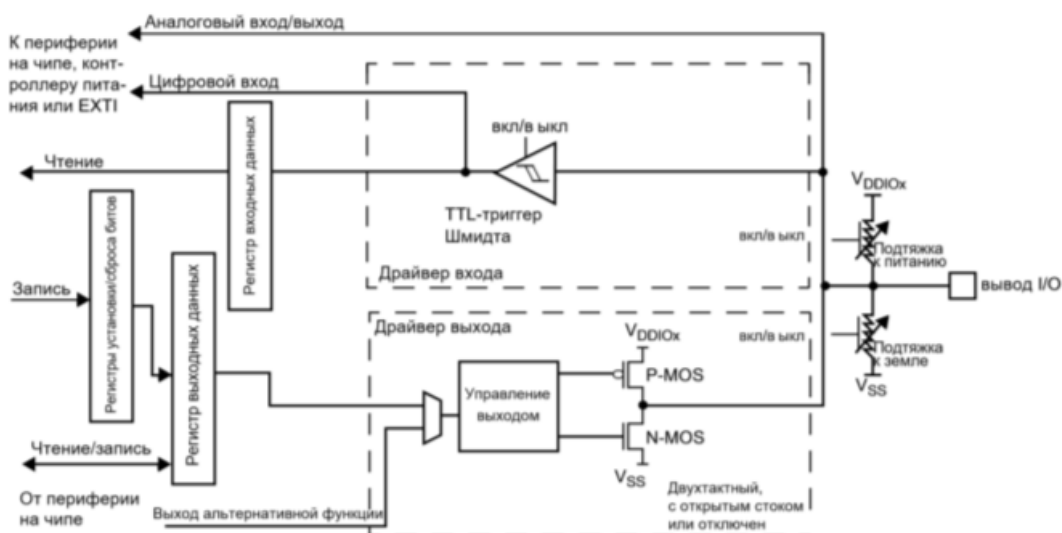


Рисунок 4 Базовая структурная схема вывода порта I/O

В зависимости от поля GPIO GPIO\_InitTypeDef.Mode микроконтроллер изменяет способ аппаратной работы I/O. Давайте рассмотрим основные режимы.

Когда вывод сконфигурирован в режиме GPIO\_MODE\_INPUT:

1. Буфер выходных данных отключен.
2. Триггер Шмидта на входе активирован.
3. Подтягивающие резисторы активированы в зависимости от значения поля Pull.
4. Данные, поступающие на вывод, защелкиваются в регистре входных данных (Input Data Register, IDR) на каждом тактовом цикле шины AHB.

5. Чтение регистра IDR отображает состояние вывода. Когда вывод сконфигурирован в режиме GPIO\_MODE\_ANALOG:

6. Буфер выходных данных отключен.
7. Триггер Шмидта на входе отключен и не потребляет ток для каждого аналогового значения вывода.
8. Подтягивающие резисторы отключены аппаратно.
9. Чтение регистра IDR возвращает 0.

Когда вывод сконфигурирован в режиме выхода:

1. Буфер выходных данных в одном из следующих режимов:
  - если режим GPIO\_MODE\_OUTPUT\_OD: 0 в регистре выходных данных (ODR) активирует n-канальный полевой транзистор, в то время как 1 переводит порт в состояние высокого импеданса – Hi-Z (p-канальный полевой транзистор всегда бездействует);

– если режим `GPIO_MODE_OUTPUT_PP`: 0 в регистре `ODR` активирует n-канальный полевой транзистор, в то время как 1 активирует p-канальный полевой транзистор.

2. Триггер Шмидта на входе активирован.

3. Подтягивающие резисторы активированы в зависимости от значения поля `Pull`.

4. Данные, поступающие на вывод, защелкиваются в регистре `IDR` на каждом тактовом цикле шины АНВ.

5. Чтение регистра `IDR` возвращает текущее состояние вывода.

6. Чтение регистра `ODR` возвращает последнее записанное значение.

Когда вывод сконфигурирован в режиме альтернативной функции:

1. Буфер выходных данных можно сконфигурировать в режим открытого стока или двухтактный.

2. Буфер выходных данных управляется сигналами, поступившими от периферийного устройства (разрешенным передатчиком и данными [`transmitter enable and data`]).

3. Триггер Шмидта на входе активирован.

4. Подтягивающие резисторы активированы в зависимости от значения поля `Pull`.

5. Данные, поступающие на вывод, защелкиваются в регистре `IDR` на каждом тактовом цикле шины АНВ.

6. Чтение регистра `IDR` возвращает текущее состояние вывода. Режимы `GPIO` `GPIO_MODE_EVT_*` относятся к спящим режимам. Когда вывод сконфигурирован на работу в одном из этих режимов, ЦПУ пробуждается (при условии, что он был переведен в спящий режим инструкцией `WFE`), если соответствующий вывод изменяет свое состояние, не генерируя при этом соответствующее прерывание. Режимы `GPIO` `GPIO_MODE_IT_*` относятся к управлению прерываниями.

Однако имейте в виду, что данная схема реализации может варьироваться в зависимости от семейств `STM32`, особенно для серии с пониженным энергопотреблением. Всегда обращайтесь к справочному руководству по вашему микроконтроллеру, в котором точно описаны режимы I/O и их влияние на работу микроконтроллера и его энергопотребление.

Также важно отметить, что такая гибкость представляет собой преимущество при проектировании платы. Например, если вам нужны внешние подтягивающие резисторы в вашем приложении, нет необходимости использовать внешние и

специализированные, поскольку соответствующие GPIO могут быть сконфигурированы с установкой `GPIO_InitTypeDef.Mode = GPIO_MODE_OUTPUT_PP` и `GPIO_InitTypeDef.Pull = GPIO_PULLUP`. Это экономит место на печатной плате и упрощает спецификацию компонентов.

В конце концов режим I/O можно сконфигурировать с помощью инструмента CubeMX, как показано на рисунке 5. Диалоговое окно Pin Configuration можно открыть в представлении Configuration, нажав кнопку GPIO.

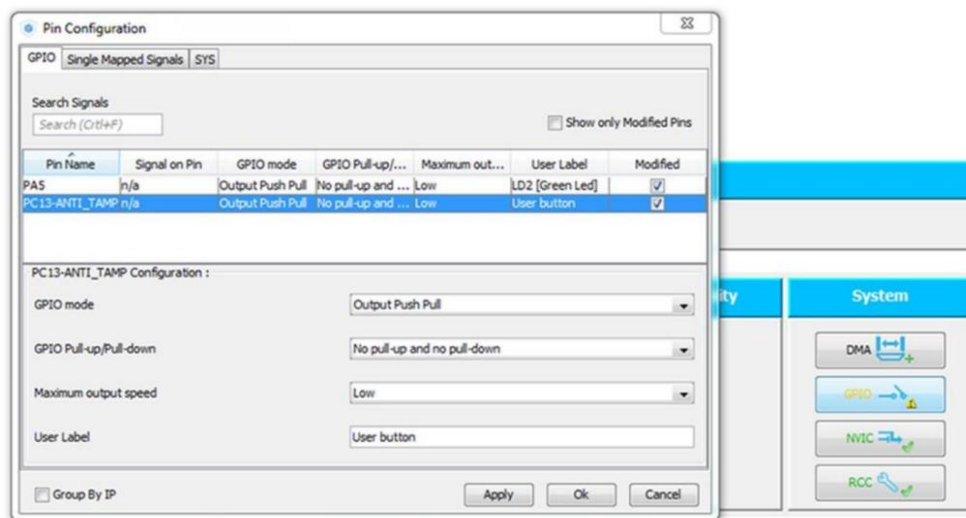


Рисунок 5 Диалоговое окно Pin Configuration может использоваться для конфигурации режима I/O

## Управление GPIO

CubeHAL предоставляет четыре процедуры манипуляции для чтения, изменения и блокировки состояния I/O. Чтобы считать состояние I/O, мы можем использовать функцию:

```
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
```

которая принимает дескриптор порта GPIO и номер вывода. Она возвращает `GPIO_PIN_RESET`, когда на выводе низкий уровень сигнала, или `GPIO_PIN_SET` – когда высокий. И наоборот, чтобы изменить состояние вывода, у нас есть функция:

```
void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
```

которая принимает дескриптор порта GPIO, номер вывода и желаемое состояние. Если мы хотим просто инвертировать состояние вывода, мы можем использовать эту удобную процедуру:

```
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin).
```

Наконец, одной из особенностей портов GPIO является возможность заблокировать конфигурацию их вводов/выводов. Любая последующая попытка изменить их конфигурацию будет неудачной, пока не произойдет сброс. Чтобы заблокировать конфигурацию вывода, мы можем использовать эту процедуру:

```
HAL_StatusTypeDef HAL_GPIO_LockPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin).
```

### **Деинициализация GPIO**

Можно установить вывод GPIO в состояние по умолчанию (то есть в режиме плавающего входа). Функция:

```
void HAL_GPIO_DeInit(GPIO_TypeDef *GPIOx, uint32_t GPIO_Pin)
```

выполняет эту работу автоматически для нас. Данная функция очень удобна, если нам больше не нужна какая-либо периферия или чтобы избежать потерь энергии при переходе процессора в спящий режим.

## **Лекция на тему «Последовательные интерфейсы МК»**

### **Рассматриваемые вопросы:**

1. Введение в UART и USART.
2. Конфигурация UART с использованием CubeMX.
3. UART-связь в режиме опроса.
4. Прерывания, относящиеся к UART.
5. Обработка ошибок.

### **Теоретический материал:**

В настоящее время в электронной промышленности существует очень большое количество протоколов последовательной связи и аппаратных интерфейсов. Большинство из них ориентированы на высокую пропускную способность, например, новейшие стандарты USB 2.0 и 3.0, Firewire (IEEE 1394) и другие. Некоторые из этих стандартов пришли из прошлого, но все еще широко распространены, особенно в качестве интерфейса связи между модулями на плате. Одним из них является интерфейс универсального синхронно-асинхронного приемопередатчика (Universal Synchronous/Asynchronous Receiver/Transmitter), также известного как USART.

Почти каждый микроконтроллер имеет как минимум одно периферийное устройство UART. Почти все микроконтроллеры STM32 предоставляют по крайней мере два интерфейса UART/USART, но большинство из них предоставляют более двух интерфейсов (иногда до восьми интерфейсов) в зависимости от количества I/O, предоставляемых корпусом микроконтроллера.

### **Введение в UART и USART**

Прежде чем приступить к анализу функций, предоставляемых HAL для управления универсальными устройствами последовательной связи, лучше всего кратко взглянуть на интерфейс UART/USART и его протокол связи.

Когда нам нужно обмениваться данными между двумя (или даже более) устройствами в обе стороны, у нас есть два возможных варианта: мы можем передавать их параллельно, то есть, используя определенное количество линий связи, равное размеру каждого слова данных (например, восемь независимых линий для слова, состоящего из восьми битов), или мы можем передавать каждый бит, составляющий наше слово, один за другим. UART/USART – это устройство, передающее параллельную последовательность битов (обычно сгруппированных в байтах) в непрерывном потоке сигналов, протекающих по одному проводу.

Когда информация передается между двумя устройствами по общему каналу, оба устройства (здесь для простоты мы будем называть их отправителем и получателем) должны договориться о временной выдержке (timing), т.е. о том, сколько времени требуется для передачи каждого отдельного бита информации. При синхронной передаче отправитель и получатель совместно используют общие синхронизирующие импульсы, сгенерированные одним из двух устройств (обычно это устройство, действующее как ведущее (master) данной системы взаимосвязи).

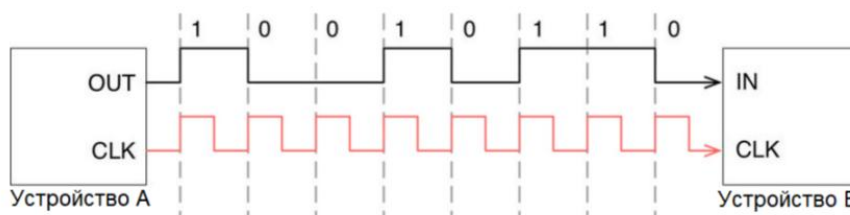


Рисунок 1 Последовательная связь между двумя устройствами с использованием общего источника синхронизирующих импульсов

На рисунке 1 приведена типовая временная диаграмма, показывающая отправку Устройством А одного байта данных (0b01101001) последовательно на Устройство В, используя общий опорный тактовый сигнал. Общие синхронизирующие импульсы также используются для согласования того, когда начинать отсчет (выборку, англ. sampling) последовательности битов: когда ведущее устройство начинает тактировать специальную линию, это означает, что оно собирается отправить последовательность битов.

В синхронной передаче скорость и продолжительность передачи определяются синхронизирующими импульсами: их частота определяет, насколько быстро мы можем передать один байт по каналу связи. Но если оба устройства, участвующие в передаче данных, договорились о том, сколько времени требуется для передачи одного бита и когда начинать и заканчивать отсчет передаваемых битов, мы можем избежать использования специальной отдельной линии синхронизации (CLK). В этом случае мы имеем асинхронную передачу.

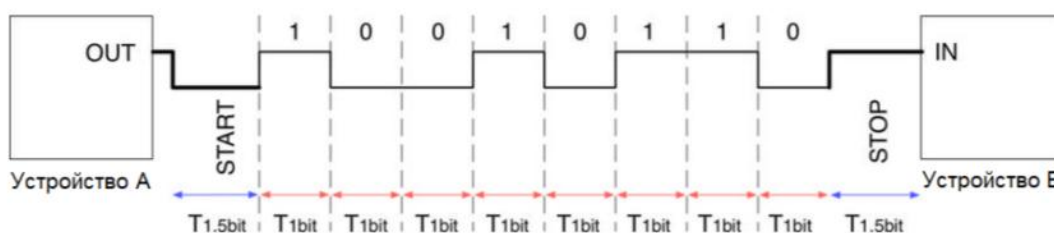


Рисунок 2 Временная диаграмма последовательной связи без специальной линии синхронизации

На рисунке 2 показана временная диаграмма асинхронной передачи. Пассивное состояние (то есть передача не происходит) представлено высоким сигналом. Передача начинается со START-бита, который представлен низким уровнем. При обнаружении приемником отрицательного перепада фронта через 1,5 периода битов (обозначен  $T_{1.5bit}$  на рисунке 2) начинается отсчет битов данных. Младший значащий бит (Least Significant Bit, LSB) обычно передается первым. Затем передается необязательный бит четности (для проверки битов данных на ошибки). Часто этот бит опускается, если предполагается, что канал передачи свободен от шума, или если на уровнях протоколов проверка ошибок выполняется выше. Передача заканчивается STOP-битом, который длится 1,5 бита.

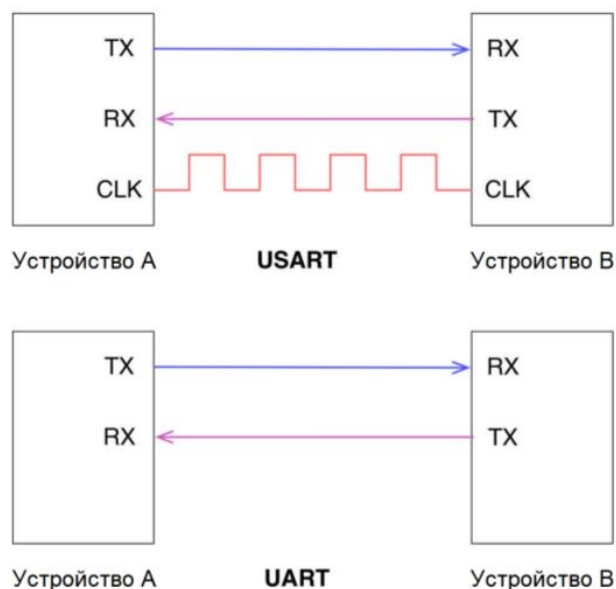


Рисунок 3 Разница в сигналах между USART и UART

Интерфейс универсального синхронного приемопередатчика – это устройство, способное передавать слово данных последовательно с использованием двух I/O, один из которых выступает в качестве передатчика (TX), а другой – в качестве приемника (RX), плюс один дополнительный вывод в качестве линии синхронизации, в то время как универсальный асинхронный приемопередатчик использует только два вывода RX/TX (см. рисунок 3). Традиционно мы ссылаемся на первый интерфейс термином USART, а на второй – UART.

UART/USART определяют метод передачи сигнала, но ничего не говорят о его уровнях напряжения. Это означает, что UART/USART микроконтроллера STM32 будут

использовать уровни напряжения выводов I/O микроконтроллера, которые практически равны VDD (эти уровни напряжения также принято называть уровнями напряжения TTL). То, как эти уровни напряжения переводятся для обеспечения последовательной связи за пределами платы, зависит от других стандартов связи. Например, EIA-RS232 или EIA-RS485 являются двумя очень популярными стандартами, которые определяют напряжения сигналов в дополнение к их временной выдержке и значению, а также физические размеры и цоколевку разъемов. Кроме того, интерфейсы UART/USART могут использоваться для обмена данными с использованием других физических и логических последовательных интерфейсов. Например, FT232RL является очень популярной интегральной схемой, которая позволяет совместить интерфейс UART с интерфейсом USB, как показано на рисунке 4.



Рисунок 4 Типовая схема на базе FT232RL, используемая для преобразования 3,3 В TTL интерфейса UART в интерфейс USB

Наличие отдельной линии синхронизации или общего соглашения о частоте передачи не гарантирует, что приемник потока байтов сможет обрабатывать их с той же скоростью передачи, что и у ведущего устройства. По этой причине некоторые стандарты связи, такие как RS232 и RS485, предоставляют возможность использовать отдельную линию аппаратного управления потоком данных (Hardware Flow Control). Например, два устройства, обменивающиеся данными с использованием интерфейса RS232, могут совместно использовать две дополнительные линии, называемые Запросом на передачу (Request To Send, RTS) и Готовностью к передаче (Clear To Send, CTS): отправитель устанавливает свою RTS, которая сообщает получателю о необходимости начать мониторинг своей входной линии данных. Когда данные будут готовы, получатель установит свою дополнительную линию CTS, которая сообщает отправителю начать отправку данных, а отправителю следует начать мониторинг выходной линии данных ведомого устройства.



Микроконтроллеры STM32 предоставляют различное количество USART, которые можно сконфигурировать для работы как в синхронном, так и в асинхронном режиме. Некоторые микроконтроллеры STM32 также предоставляют интерфейсы, которые могут работать только как UART. В таблице 1 перечислены UART/USART, предоставляемые микроконтроллерами STM32, размещенными на всех платах Nucleo. Большинство USART также могут автоматически реализовывать аппаратное управление потоком, как для стандартов RS232, так и для RS485.

Таблица 1 Список доступных USART и UART на всех платах Nucleo

	Nucleo P/N	USARTs + UARTs	USART#	HW Flow Control RS232	HW Flow Control RS485
	NUCLEO-F446RE	4 + 2	USART1/2/3	Y	-
			USART6	-	-
			UART4/5	Y	-
	NUCLEO-F411RE NUCLEO-F410RB NUCLEO-F401RE	3 + 0	USART1/2	Y	-
			USART6	-	-
	NUCLEO-F334R8	3 + 0	USART1/2/3	Y	Y
	NUCLEO-F303RE	3 + 2	USART1/2/3	Y	Y
			UART4/5	-	-
	NUCLEO-F302R8	3 + 0	USART1/2/3	Y	Y
	NUCLEO-F103RB	3 + 0	USART1/2/3	Y	-
	NUCLEO-F091RC	8 + 0	USART1/2/3/4	Y	Y
			USART5	-	Y
			USART6/7/8	-	-
	NUCLEO-F072RB NUCLEO-F070RB	4 + 0	USART1/2/3/4	Y	Y
	NUCLEO-F030R8	2 + 0	USART1/2	Y	Y
	NUCLEO-L476RG	3 + 2	USART1/2/3	Y	Y
			UART4/5	Y	Y
	NUCLEO-L152RE	3 + 2	USART1/2/3	Y	-
			UART4/5	-	-
	NUCLEO-L073RZ	4 + 2	USART1/2/4	Y	Y
			UART5	RTS Only	Y
	NUCLEO-L053R8	2 + 0	USART1/2	Y	Y

Все платы Nucleo-64 спроектированы таким образом, что USART2 целевого микроконтроллера связан с интерфейсом ST-LINK. Когда мы устанавливаем драйверы ST-LINK, также устанавливается дополнительный драйвер для виртуального COM-порта (Virtual COM Port, VCP): он позволяет нам получить доступ к USART2 целевого

микроконтроллера через интерфейс USB без использования специального конвертера TTL/USB. Используя программу эмуляции терминала, мы можем обмениваться сообщениями и данными с нашей Nucleo.

CubeHAL разделяет API для управления интерфейсами UART и USART. Все функции и дескрипторы, используемые для обработки USART, начинаются с префикса HAL\_USART и содержатся в файлах stm32xxx\_hal\_usart.{c,h}, а связанные с управлением UART начинаются с префикса HAL\_UART и содержатся в файлах stm32xxx\_hal\_uart.{c,h}.

### Конфигурация UART с использованием CubeMX

При первой конфигурации USART2 для нашей Nucleo лучше всего использовать CubeMX. Первым шагом является разрешение периферийного устройства USART2 в представлении Pinout, выбирая запись Asynchronous в выпадающем списке Mode, как показано на рисунке 5.

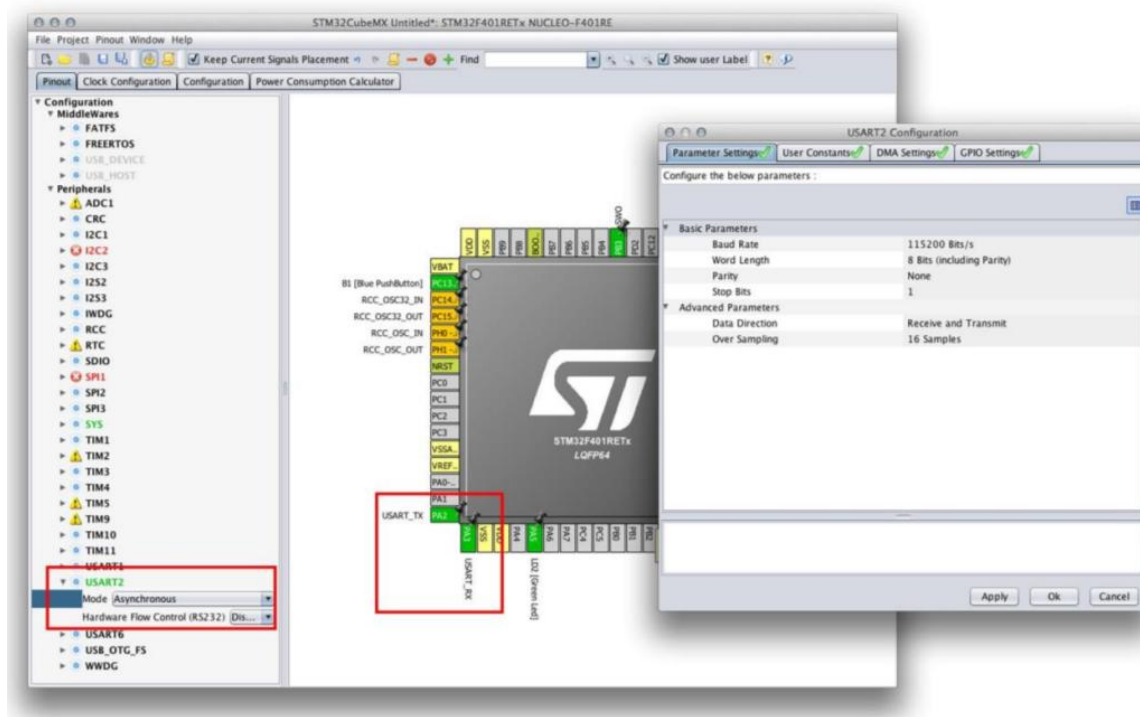


Рисунок 5 CubeMX можно использовать для облегчения конфигурации интерфейса USART2

Выводы PA2 и PA3 будут автоматически выделены зеленым цветом. Затем перейдите в раздел Configuration и нажмите кнопку USART2. Появится диалоговое окно конфигурации, как показано на рисунке 5 справа. Оно позволяет нам конфигурировать параметры USART, такие как BaudRate, длину слова и так далее.

После конфигурации интерфейса USART мы можем сгенерировать код Си. Вы заметите, что CubeMX помещает весь код инициализации USART2 в

MX\_USART2\_UART\_Init() (который содержится в файле main.c). И напротив, весь код, связанный с конфигурацией GPIO, помещается в функцию HAL\_UART\_MspInit(), которая находится в файле stm32xxxx\_hal\_msp.c.

### **UART-связь в режиме опроса**

Микроконтроллеры STM32 и, следовательно, CubeHAL предоставляют три способа обмена данными между узлами по каналу связи UART: режимы опроса, прерываний и DMA. С этого момента важно подчеркнуть, что данные режимы представляют собой не только три разных варианта обращения с UART-связью. Это три разных программных подхода к одной и той же задаче, которые дают несколько преимуществ как с точки зрения разработки, так и с точки зрения производительности. Кратко представим их.

В режиме опроса (polling mode), также называемом блокирующим режимом (blocking mode), основное приложение или один из его потоков синхронно ожидает передачу и прием данных. Это наиболее простая форма передачи данных с использованием данного периферийного устройства, и ее можно использовать, когда скорость передачи не слишком низкая, при этом UART не используется в качестве критически важного периферийного устройства в нашем приложении (классическим примером является использование UART в качестве консоли вывода при отладочной деятельности).

В режиме прерываний (interrupt mode), также называемом неблокирующим режимом (non-blocking mode), основное приложение освобождается от ожидания завершения передачи и приема данных. Процедуры передачи данных завершаются, как только они завершают конфигурацию периферийного устройства. Когда передача данных заканчивается, последующее прерывание будет сигнализировать об этом основному коду. Данный режим больше подходит, когда скорость обмена данными низкая (ниже 38400 бит/с) или когда передача/прием происходят «изредка», по сравнению с другими действиями, выполняемыми микроконтроллером, и мы не хотим «застрять» на ожидании передачи данных.

Режим DMA обеспечивает наилучшую пропускную способность передачи благодаря прямому доступу периферийного устройства UART к внутренней памяти микроконтроллера. Данный режим лучше всего подходит для высокоскоростной связи, и когда мы полностью хотим освободить микроконтроллер от накладных расходов при передаче данных. Без режима DMA практически невозможно достичь самых высоких скоростей передачи данных, которые способны обрабатывать периферийные устройства USART.

Для передачи последовательности байтов по USART в режиме опроса HAL предоставляет функцию

```
HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData,
                                     uint16_t Size, uint32_t Timeout);
```

где:

- `huart`: это указатель на экземпляр структуры `UART_HandleTypeDef`, рассмотренной нами ранее, который идентифицирует и конфигурирует периферийное устройство UART;
- `pData`: указатель на массив, длина которого равна параметру `Size`, содержащий последовательность байтов, которые мы собираемся передать;
- `Timeout`: максимальное время, выраженное в миллисекундах, в течение которого мы будем ждать завершения передачи. Если передача не завершается в течение заданного времени ожидания, функция прерывает свое выполнение и возвращает значение `HAL_TIMEOUT`; в противном случае она возвращает значение `HAL_OK`, если не возникает других ошибок. Кроме того, мы можем передать тайм-аут, равный `HAL_MAX_DELAY` (`0xFFFF FFFF`), чтобы неопределенно долго ждать завершения передачи.

И, наоборот, для получения последовательности байтов по USART в режиме опроса HAL предоставляет функцию

```
HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData,
                                    uint16_t Size, uint32_t Timeout);
```

где:

- `huart`: это указатель на экземпляр структуры `UART_HandleTypeDef`, рассмотренной нами ранее, который идентифицирует и конфигурирует периферийное устройство UART;
- `pData`: указатель на массив, длина которого по крайней мере равна параметру `Size`, содержащий последовательность байтов, которую мы собираемся получить. Функция будет блокироваться, пока не будут получены все байты, указанные параметром `Size`;
- `Timeout`: максимальное время, выраженное в миллисекундах, а течение которого мы будем ждать завершения приема. Если передача не завершается в течение заданного времени ожидания, функция прерывает свое выполнение и возвращает значение `HAL_TIMEOUT`; в противном случае она возвращает значение `HAL_OK`, если не возникает других ошибок. Кроме того, мы можем передать тайм-аут, равный

HAL\_MAX\_DELAY (0xFFFF FFFF), чтобы неопределенно долго ждать завершения получения.

### Прерывания, относящиеся к UART

Каждое периферийное устройство USART микроконтроллера STM32 предоставляет прерывания, перечисленные в таблице 6. Эти прерывания включают в себя IRQ, связанные и с передачей данных, и с ошибками связи. Их можно разделить на две группы:

- IRQ, генерируемые во время передачи: «передача завершена» (Transmission Complete), «готов к отправке» (Clear to Send) или «регистр передаваемых данных пуст» (Transmit Data Register Empty);
- IRQ, сгенерированные во время приема: «обнаружение незанятой линии» (Idle Line detection), «ошибка вследствие переполнения» (Overrun error), «регистр принимаемых данных не пуст» (Receive Data Register not Empty), «ошибка при проверке четности» (Parity error), «обнаружение разрыва линии связи» (LIN break detection), «флаг шума», англ. Noise Flag (только в многобуферной связи) и «ошибка кадрирования», англ. Framing Error (только в многобуферной связи).

Таблица 6 Список прерываний, относящихся к USART

Событие прерывания	Флаг события	Бит управления разрешением
Регистр передачи данных пуст	TXE	TXEIE
Установка флага	Clear To Send (CTS)	CTS CTSIE
Передача завершена	TC	TCIE
Принятые данные готовы к чтению	RXNE	RXNEIE
Обнаружена ошибка переполнения	ORE	RXNEIE
Обнаружена незанятая линия	IDLE	IDLEIE
Ошибка при проверке четности	PE	PEIE
Установка флага разрыва линии	LBD	LBDIE
Установка флага шума, возникновение ошибок	NF или ORE или FE	EIE

переполнения и кадрирования в многобуферной связи		
--	--	--

Эти события генерируют прерывание, если установлен соответствующий бит управления разрешением прерывания, англ. Enable Control Bit (третий столбец таблицы 6). Однако микроконтроллеры STM32 спроектированы таким образом, чтобы все эти IRQ были связаны только с одной ISR для каждого периферийного устройства USART.

### Обработка ошибок

При работе с внешними подключениями обработка ошибок является аспектом, который мы должны строго учитывать. Периферийное устройство UART в STM32 предлагает несколько флагов ошибок, относящихся к ошибкам обмена данными. Более того, возможно, что соответствующее ошибке прерывание не будет обработано при ее возникновении.

CubeHAL предназначен для автоматического обнаружения условий ошибок и предупреждения нас о них. Нам нужно только реализовать функцию HAL\_UART\_ErrorCallback() внутри кода нашего приложения. HAL\_UART\_IRQHandler() автоматически вызовет ее в случае возникновения ошибки. Чтобы понять, какая именно произошла ошибка, мы можем проверить значение поля UART\_HandleTypeDef->ErrorCode. Список кодов ошибок приведен в таблице 7.

Таблица 7 Список возможных значений UART\_HandleTypeDef->ErrorCode

Код ошибки UART	Описание
HAL_UART_ERROR_NONE	Ошибка не произошла
HAL_UART_ERROR_PE	Ошибка при проверке четности
HAL_UART_ERROR_NE	Ошибка вследствие зашумления
HAL_UART_ERROR_FE	Ошибка кадрирования данных
HAL_UART_ERROR_ORE	Ошибка вследствие переполнения
HAL_UART_ERROR_DMA	Ошибка передачи посредством DMA

HAL\_UART\_IRQHandler() разработан таким образом, что нам не нужно вдаваться в подробности реализации обработки ошибок UART. Код HAL автоматически выполнит все необходимые шаги для обработки ошибки (например, сброс флагов событий, бита отложенного состояния и так далее), оставляя нам ответственность за обработку ошибки на уровне приложения (например, мы можем попросить другой узел повторно отправить поврежденный кадр данных).

## **Лекция на тему «Система прерываний МК»**

### **Рассматриваемые вопросы:**

1. Обработка прерываний.
2. Контроллер NVIC.
3. Таблица векторов в STM32.
4. Разрешение прерываний в CubeMX.
5. Жизненный цикл прерываний.

### **Теоретический материал:**

#### **Обработка прерываний**

Управление аппаратными средствами связано с асинхронными событиями. Большинство из них приходит от аппаратной периферии. Например, таймер достигает заданного значения периода или UART, который предупреждает о поступлении данных. Другие порождаются «внешним окружением» нашей платы. Его примером может быть нажатие пользователем переключателя, заставляющего плату «зависать».

Все микроконтроллеры предоставляют функцию, называемую прерываниями. Прерывание – это асинхронное событие, которое вызывает остановку выполнения текущего кода в приоритетном порядке (чем важнее прерывание, тем выше его приоритет; и более приоритетное прерывание приведет к приостановке прерывания с более низким приоритетом). Код, который обслуживает прерывание, называется Процедурой обслуживания прерывания (Interrupt Service Routine, ISR).

Прерывания являются источником многозадачности: аппаратное обеспечение знает о них и отвечает за сохранение текущего контекста исполнения (т. е. стекового кадра, текущего счетчика команд (Program Counter, PC) и некоторых других вещей) перед вызовом ISR. Они используются операционными системами реального времени для введения понятия задач. Без помощи аппаратного обеспечения невозможно создать настоящую вытесняющую многозадачность, которая позволяет переключаться между несколькими контекстами исполнения без необратимой потери текущего исполняемого потока.

Прерывания могут возникать как от аппаратного, так и от программного обеспечения. Архитектура ARM различает два типа исключений: прерывания, вызываемые аппаратным обеспечением, и исключения, вызываемые программным обеспечением (например, доступ к неверной ячейке памяти). В терминологии ARM прерывание – это тип исключения.

Процессоры Cortex-M предоставляют модуль, предназначенный для управления исключениями. Он называется Контроллером вложенных векторных прерываний (Nested Vectored Interrupt Controller, NVIC).

### Контроллер NVIC

Контроллер NVIC – это специальный аппаратный модуль внутри микроконтроллеров на базе Cortex-M, отвечающий за обработку исключений. На рисунке 1 показана взаимосвязь между модулем NVIC, ядром процессора и периферийными устройствами. Здесь мы должны различать два типа периферийных устройств: внешние по отношению к ядру Cortex-M, но внутренние по отношению к микроконтроллеру STM32 (например, таймеры, интерфейсы UART и т. д.), и периферийные устройства, внешние по отношению к микроконтроллеру. Источником прерываний, поступающих от последнего вида периферийных устройств, является вывод микроконтроллера, который может быть сконфигурирован как I/O общего назначения (например, тактильный переключатель, подключенный к выводу, сконфигурированному как вход) или для управления внешним продвинутым периферийным устройством (например, I/O, сконфигурированные для обмена данными с ethernet phyther через интерфейс RPII). Как мы увидим далее, специальный программируемый контроллер, называемый Контроллером внешних прерываний/событий (External Interrupt/Event Controller, EXTI), отвечает за взаимосвязь между внешними сигналами I/O и контроллером NVIC.

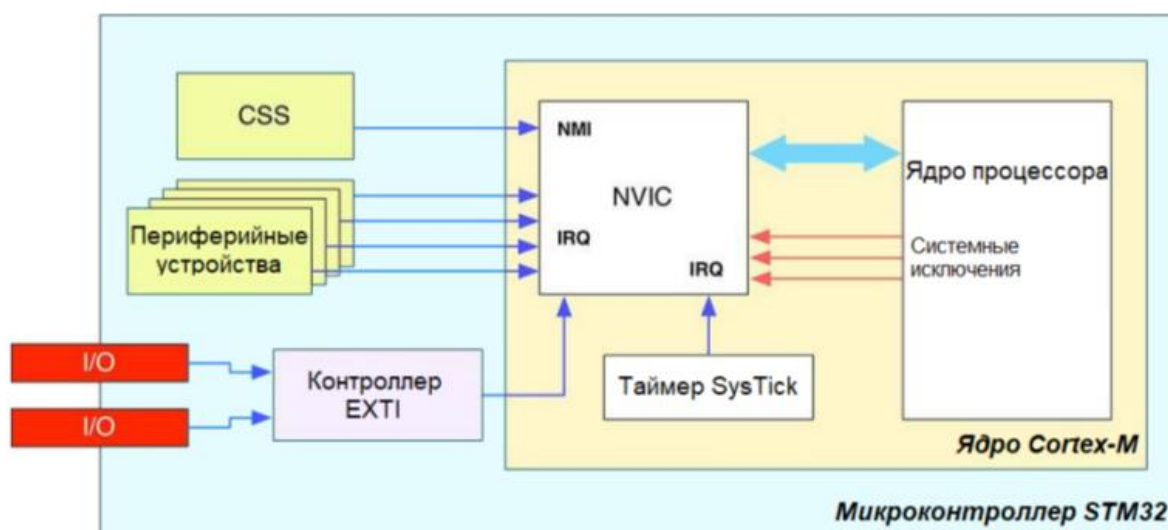


Рисунок 1 Взаимосвязь между контроллером NVIC, ядром Cortex-M и периферийными устройствами STM32



Как указывалось ранее, ARM различает системные исключения, возникающие внутри ядра ЦПУ, и аппаратные исключения, поступающие от внешних периферийных устройств, также называемые запросами прерываний (Interrupt Requests, IRQ). Программисты управляют исключениями посредством использования специальных ISR, которые кодируются на более высоком уровне (чаще всего с использованием языка Си). Процессор знает, где найти эти процедуры благодаря косвенной таблице, содержащей адреса процедур обслуживания прерываний в памяти. Данная таблица обычно называется таблицей векторов, и каждый микроконтроллер STM32 определяет свою собственную таблицу. Давайте проанализируем ее подробнее.

### Таблица векторов в STM32

Все процессоры Cortex-M определяют фиксированный набор исключений (15 для ядер Cortex-M3/4/7 и 13 для ядер Cortex-M0/0+), общий для всех семейств Cortex-M и, следовательно, общий для всех серий STM32.

Таблица 1 Типы исключений Cortex-M

Номер	Тип исключения	Приоритет	Описание
1	Reset	-3	Сброс
2	NMI	-2	Немаскируемое прерывание
3	Hard Fault	-1	Любой отказ, если отказ не может быть активирован из-за приоритета или программируемый обработчик отказа не разрешен.
4	Memory Management	Программируемый	Несоответствие MPU, включая нарушение прав доступа и отсутствие совпадений. Используется, даже если модуль MPU отключен или отсутствует.
5	Bus Fault	Программируемый	Отказ предвыборки, отказ доступа к памяти и другие, связанные с адресом/памятью.
6	Usage Fault	Программируемый	Ошибка в программе, такая как выполнение неопределенной инструкции или недопустимая

			попытка перехода между состояниями.
7-10	—	—	ЗАРЕЗЕРВИРОВАНО
11	SVCall	Программируемый	Вызов системной службы командой SVC.
12	Debug monitor	Программируемый	Монитор отладки – для программной отладки.
13	—	—	ЗАРЕЗЕРВИРОВАНО
14	PendSV	Программируемый	Отложенный запрос для системной службы
15	SYSTICK	Программируемый	Срабатывание системного таймера
16- [47/240]	IRQ	Программируемый	Вход внешнего прерывания

Рассмотрим некоторые из них подробнее:

- Reset: исключение сброса возникает сразу после сброса ЦПУ. Его обработчик является реальной точкой входа работающей микропрограммы. В приложении STM32 все начинается с этого исключения. Обработчик содержит некоторые ассемблерные функции, предназначенные для инициализации среды выполнения, такие как основной стек, сегмент .bss и так далее;

- NMI: немаскируемое прерывание – это особое исключение, имеющее самый высокий приоритет после сброса. Как и исключение сброса Reset, оно не может быть маскировано (запрещено) и может быть связано с критическими и неотложными действиями. В микроконтроллерах STM32 оно связано с Системой защиты тактирования (Clock Security System, CSS). CSS является периферийным устройством для самодиагностики, обнаруживающим отказ HSE. Если он происходит, HSE автоматически отключается (это означает, что внутренний HSI автоматически включается) и возникает прерывание NMI, чтобы сообщить программному обеспечению, что что-то случилось с HSE;

- Hard Fault: тяжелый отказ является общим исключением отказа, и, следовательно, связан с программными прерываниями. Когда другие исключения отказов запрещены, он действует как накопитель для всех типов исключений (например, обращение к недопустимой ячейке памяти вызывает исключения тяжелого отказа, если отказ шины (Bus Fault) не разрешен);

- Memory Management Fault: отказ системы управления памятью происходит при попытке обращения выполняющимся кодом к некорректному адресу или нарушения им правил доступа Модуля защиты памяти (Memory Protection Unit, MPU);
- Bus Fault: отказ шины происходит, когда интерфейс шины АНВ получает ответ об ошибке от ведомой шины (также называемой отказом предвыборки при выборке команд или отказом данных при чтении данных). Также может быть вызван другими некорректными доступами (например, чтение несуществующей ячейки памяти SRAM);
- Usage Fault: отказ программы происходит, когда есть программная ошибка, такая как выполнение неопределенной инструкции, проблема выравнивания или попытка получить доступ к отсутствующему сопроцессору;
- SVCCall: данное исключение не является условием отказа, оно возникает при вызове инструкции «Вызов системной службы» (Supervisor Call, SVC). Она используется операционными системами реального времени для выполнения инструкций в привилегированном состоянии (задача, требующая выполнения привилегированных операций, выполняет инструкцию SVC, а ОС выполняет запрошенные операции – то же самое поведение у системного вызова в иной ОС);
- Debug Monitor: исключение монитора отладки возникает, когда происходит событие отладки программного обеспечения при нахождении ядра процессора в режиме мониторинга отладки (Monitor Debug-Mode). Оно также используется в качестве исключения для событий отладки, таких как точки останова и наблюдения, когда используется программное решение отладки (software based debug solution);
- PendSV: запрос системной службы – еще одно исключение, связанное с ОСПВ. В отличие от исключения SVCCall, которое выполняется сразу после выполнения инструкции SVC, PendSV может быть отложено. Это позволяет ОСПВ выполнять задачи с более высокими приоритетами;
- SysTick: исключение системного таймера также обычно связано с действиями ОСПВ. Каждой ОСПВ необходим таймер, чтобы периодически прерывать выполнение текущего кода и переключаться на другую задачу. Все микроконтроллеры STM32 оснащены таймером SysTick, встроенным в ядро Cortex-M. Несмотря на то что любой другой таймер может использоваться для планирования системных действий, наличие специализированного таймера обеспечивает переносимость среди всех семейств STM32 (из-за причин оптимизации, связанных с внутренней площадкой для

монтажа кристалла микроконтроллера, не все таймеры могут быть доступны в качестве внешнего периферийного устройства). Более того, даже если мы не используем OCPB в нашей микропрограмме, важно помнить, что CubeHAL от ST использует таймер SysTick для выполнения внутренних действий, связанных со временем (а это также предполагает, что таймер SysTick сконфигурирован генерировать прерывание каждые 1 мс).

Остальные исключения, которые могут быть определены для какого-либо микроконтроллера, связаны с обработкой IRQ. Ядра Cortex-M0/0+ допускают до 32 внешних прерываний, в то время как ядра Cortex-M3/4/7 позволяют производителям интегральных схем определять до 240 прерываний.

### Разрешение прерываний в CubeMX

CubeMX можно использовать для облегчения разрешения IRQ и автоматической генерации кода ISR. Первым шагом является разрешение соответствующей линии запроса прерывания EXTI, используя представление Chip, как показано на рисунке 5.

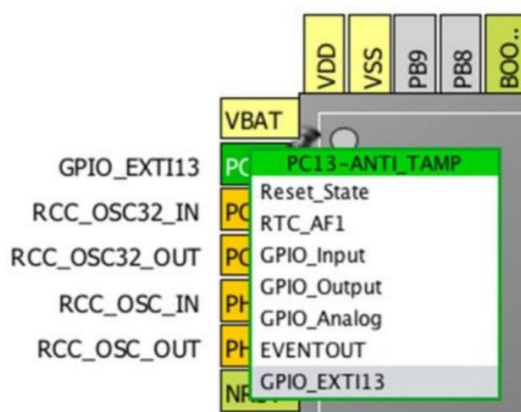


Рисунок 5 Как GPIO может быть привязан к линии прерывания EXTI с помощью CubeMX

Как только мы разрешили IRQ, нам нужно дать CubeMX команду генерировать соответствующую ISR. Данная конфигурация выполняется в представлении Configuration, нажав кнопку NVIC. Появится список ISR, которые можно разрешить, как показано на рисунке 6.

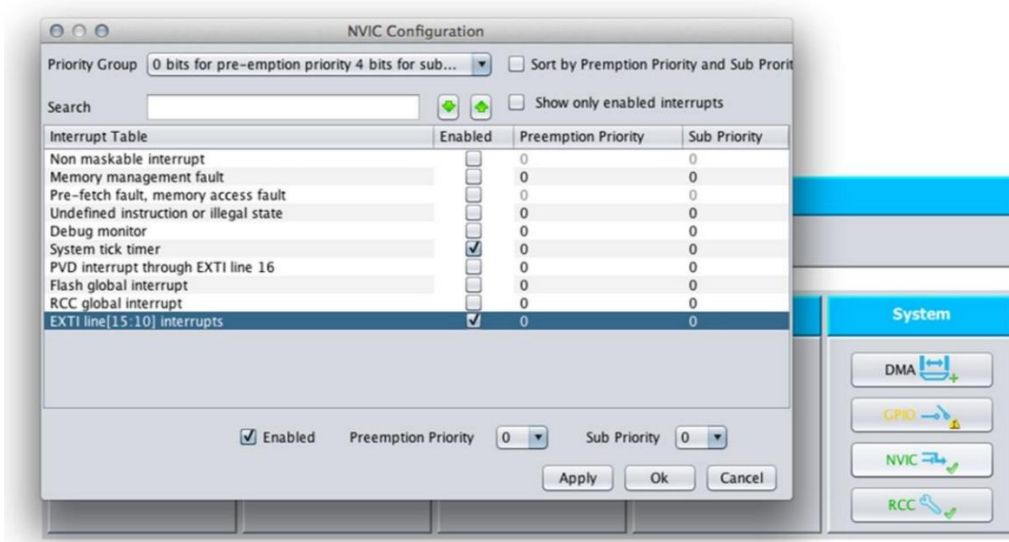


Рисунок 6 Представление NVIC Configuration позволяет разрешить соответствующую ISR

CubeMX автоматически добавит разрешенные ISR в файл `src/stm32fxxx_it.c` и позаботится о разрешении IRQ. Кроме того, он добавляет для нас соответствующую процедуру обработчика HAL для вызова.

### Жизненный цикл прерываний

Тому, кто имеет дело с прерываниями, очень важно понимать их жизненный цикл. Хотя ядро Cortex-M автоматически выполняет за нас большую часть работы, мы должны обратить внимание на некоторые аспекты, которые могут стать источником путаницы при управлении прерываниями. Однако в данном параграфе рассматривается жизненный цикл прерываний с «точки зрения HAL». Если вы заинтересованы в более глубоком рассмотрении данного вопроса, серия книг Джозефа Ю является опять же лучшим источником.

Прерывание может быть:

1. Либо запрещено (поведение по умолчанию) или разрешено.  
– мы разрешаем/запрещаем его, вызывая функцию `HAL_NVIC_EnableIRQ()/ HAL_NVIC_DisableIRQ()`.
2. Либо отложено (запрос ожидает обработки) или не отложено.
3. Либо в активном (обслуживаемом) или неактивном состоянии.

Важно изучить, что происходит при срабатывании прерывания.

Когда срабатывает прерывание, оно помечается как отложенное (pending), пока процессор не сможет его обслужить. Если никакие другие прерывания в настоящее

время не обрабатываются, его отложенное состояние автоматически сбрасывается процессором, который почти сразу начинает обслуживать его.

## **Лекция на тему «Таймеры счетчики МК»**

### **Рассматриваемые вопросы:**

1. Введение в таймеры.
2. Категории таймеров в микроконтроллере STM32.
3. Доступность таймеров в ассортименте STM32.
4. Базовые таймеры.

### **Теоретический материал:**

Встраиваемые устройства выполняют некоторые действия с учетом времени. Для достаточно простых и неточных задержек цикл активного ожидания (busy loop) мог бы выполнить эту задачу, однако использование ядра ЦПУ для выполнения зависимых от времени действий никогда не является разумным решением. По этой причине все микроконтроллеры предоставляют отдельную аппаратную периферию: таймеры. Таймеры являются не только генераторами временного отсчета (timebase generators), но и также предоставляют несколько дополнительных функций, используемых для взаимодействия с ядром Cortex-M и другими периферийными устройствами, как внутренними, так и внешними по отношению к микроконтроллеру.

В зависимости от семейства и используемого корпуса, микроконтроллеры STM32 реализуют различное количество таймеров, каждый из которых имеет определенные характеристики. Некоторые номера устройств по каталогу (P/N) могут содержать до 14 независимых таймеров. В отличие от других периферийных устройств таймеры имеют практически одинаковую реализацию во всех сериях STM32 и сгруппированы в девять различных категорий. Наиболее важными из них являются: таймеры базовые (basic), общего назначения (general purpose) и расширенного управления (advanced).

Таймеры STM32 – это продвинутое периферийные устройства, которые предлагают широкий спектр применения. Более того, некоторые из их функций являются специфическими для области приложения. Все это требует полноценной отдельной книги, чтобы углубиться тему (вы должны учитывать, что обычно более 250 страниц типового технического описания STM32 посвящено таймерам). В этой главе, которая, несомненно, является самой длинной в книге, делается попытка сформировать наиболее актуальные концепции, касающиеся базовых таймеров и таймеров общего назначения в микроконтроллерах STM32, рассматривая относящийся к их программированию модуль CubeHAL.

## Введение в таймеры

Таймер – это автономный счетчик с частотой отсчета, составляющей часть его источника тактового сигнала. Частота отсчета может быть уменьшена с помощью отдельного делителя для каждого таймера. В зависимости от типа таймера, он может тактироваться внутренним тактовым сигналом (который получен от шины, к которой подключен таймер), внешним источником тактового сигнала или другим таймером, используемым в качестве «ведущего».

Обычно таймер считает от нуля до заданного значения, которое не может быть выше максимального беззнакового целого значения разрядности таймера (например, 16-разрядный таймер переполняется, когда счетчик достигает 65535), при этом он также может отсчитывать в обратную сторону и другими способами, которые мы увидим далее.

Наиболее значимые таймеры в микроконтроллере STM32 обладают несколькими возможностями:

1. Они могут использоваться в качестве генератора временного отсчета (эта функция является общей для всех таймеров STM32).
2. Их можно использовать для измерения частоты возникновения внешнего события (режим захвата входного сигнала).
3. Для управления формой выходного сигнала или для индикации истечения определенного периода времени (режим сравнения выходного сигнала).

– одноимпульсный режим (One pulse mode, OPM) является частным случаем режима захвата входного сигнала и режима сравнения выходного сигнала. Он позволяет запускать счетчик в ответ на внешнее воздействие и генерировать импульс программируемой длительности после программируемой задержки.

4. Для генерации сигналов ШИМ (PWM) в режиме выравнивания по фронтам или по центру периода независимо на каждом канале (режим ШИМ).

В некоторых микроконтроллерах STM32 (особенно от STM32F3 и последних серий STM32L4) некоторые таймеры могут генерировать центрированные ШИМ-сигналы с программируемой задержкой и фазовым сдвигом. В зависимости от типа таймера, он может генерировать прерывания или запросы к DMA при возникновении следующих событий:

### 4.1. События обновления

- Переполнение/опустошение (overflow/underflow) счетчика
- Завершение инициализации счетчика
- Другие



## 4.2. События триггерной цепи

- Запуск/останов счетчика
- Инициализация счетчика
- Другие

## 4.3. Захват входного сигнала/Сравнение выходного сигнала

### **Категории таймеров в микроконтроллере STM32**

Таймеры STM32 можно сгруппировать в девять категорий. Давайте кратко рассмотрим каждую из них:

1. Базовые таймеры: таймеры из этой категории являются самым простым видом таймеров в микроконтроллерах STM32. Это 16-разрядные таймеры, используемые для генерации временного отсчета, и они не имеют выводов I/O. Базовые таймеры также используются для подачи питания на периферийное устройство ЦАП, так как их событие обновления (update event) может вырабатывать запросы к DMA для ЦАП (по этой причине они обычно доступны в микроконтроллерах STM32, предоставляющих как минимум ЦАП). Базовые таймеры также могут быть использованы в качестве «ведущих» для других таймеров.

2. Таймеры общего назначения: это 16/32-разрядные таймеры (в зависимости от серии STM32), обеспечивающие классические функции, которые предполагается реализовать в таймере современного встраиваемого микроконтроллера. Они используются в любом приложении для сравнения выходного сигнала (синхронизация и генерация задержки), одноимпульсного режима, захвата входного сигнала (для измерения частоты внешнего сигнала), интерфейса датчика (энкодера, датчика Холла) и так далее. Очевидно, что таймер общего назначения может быть использован в качестве генератора временного отсчета, как и базовый таймер. Таймеры данной категории предоставляют четыре программируемых входных/выходных канала.

– 1-канальные/2-канальные: это две подгруппы таймеров общего назначения, обеспечивающие только один/два входных/выходных канала;

– 1-канальные/2-канальные с одним комплементарным выходом: аналогичны предыдущим типам, но с генератором мертвого времени (dead time) на одном канале. Это позволяет получать комплементарные сигналы с временной задержкой независимо от таймеров расширенного управления.

3. Таймеры расширенного управления: являются наиболее полными в микроконтроллере STM32. В дополнение к функциям таймера общего назначения, они включают в себя несколько функций, относящихся к приложениям управления

двигателем и цифрового преобразования энергии: три комплементарных сигнала с введением мертвого времени, вход аварийного отключения.

4. Таймер высокого разрешения: Таймер высокого разрешения (HRTIM1) – это специальный таймер, предоставляемый некоторыми микроконтроллерами серии STM32F3 (серии, предназначенной для управления двигателем и преобразования энергии). Он позволяет генерировать цифровые сигналы с высокоточными временными выдержками, такими как ШИМ или сдвинутые по фазе импульсы. Он состоит из 6 вспомогательных таймеров: 1 ведущего и 5 ведомых, всего 10 выходов с высоким разрешением, которые могут быть связаны попарно для введения мертвого времени. Он также имеет 5 входов сбоя для целей защиты и 10 входов для обработки внешних событий, таких как ограничение тока, отключение при нуле напряжения или нуле тока (zero voltage or zero current switching). Таймер HRTIM1 состоит из цифрового ядра (kernel) с тактовой частотой 144 МГц, за которым следуют линии задержки (delay lines). Линии задержки с управлением по замкнутому контуру гарантируют разрешение в 217 пс независимо от напряжения, температуры или отклонения производственного процесса. Высокое разрешение доступно на 10 выходах во всех режимах работы: при переменном коэффициенте заполнения, при переменной частоте и постоянном времени включения (constant ON time).

Таблица 1 Наиболее значимые функции каждой категории таймеров

Timer Type	Counter resolution	Counter type	DMA	Channels	Complimentary channels	Synchronization	
						Master	Slave
Advanced	16-bit	up, down and center aligned	Yes	4	3	Yes	Yes
General purpose	16/32-bit	up, down and center aligned	Yes	4	0	Yes	Yes
Basic	16-bit	up	Yes	0	0	Yes	No
1-channel	16-bit	up	No	1	0	Yes (OC signal)	No
2-channels	16-bit	up	No	2	0	Yes	Yes
1-channel with one complementary output	16-bit	up	Yes	1	1	Yes (OC signal)	No
2-channel with one complementary output	16-bit	up	Yes	2	1	Yes	Yes
High-resolution	16-bit	up	Yes	10	10	Yes	Yes
Low-power	16-bit	up	No	1	0	No	No

5. Таймеры с пониженным энергопотреблением: таймеры из данной группы специально разработаны для приложений с пониженным энергопотреблением. Благодаря разнообразию источников тактового сигнала эти таймеры могут работать во

всех режимах питания (кроме режима Ожидания). Учитывая эту возможность – работать даже без внутреннего источника тактового сигнала, таймеры с пониженным энергопотреблением могут использоваться в качестве «счетчиков импульсов», что может быть полезно в некоторых приложениях. В них также присутствует возможность вывести систему из режима пониженного энергопотребления.

В таблице 1 приведены наиболее значимые функции для каждой категории таймеров, которые чаще всего используются.

### **Доступность таймеров в ассортименте STM32**

Не все типы таймеров доступны во всех микроконтроллерах STM32. Они зависят главным образом от серии STM32, вида поставки (sale type) и используемого корпуса. В таблице 2 сведено распределение 22 таймеров во всех семействах STM32.

Важно отметить некоторые моменты касательно таблицы 2:

1. При наличии конкретного таймера (например, TIM1, TIM8 и т. д.) его реализация (функции, количество и тип регистров, генерируемые прерывания, запросы к DMA, межсоединение периферийных устройств и т. д.) одинакова во всех микроконтроллерах STM32. Это гарантирует вам, что микропрограмма, написанная для использования данных специфических таймеров, переносима на другие микроконтроллеры или серию STM32, имеющие такие же таймеры.

2. Наличие таймера в микроконтроллере, принадлежащем конкретному семейству, зависит от вида поставки и используемого корпуса (корпусы с большим количеством выводов могут обеспечить все таймеры, реализованные этим семейством).

3. Таблица была взята, расширена и переорганизована с AN4013.

Таблица 2 Какие таймеры реализованы в каждой серии STM32

Timer Type		STM32F0 series	STM32F100 line	STM32F101 /102/103/105/107 lines	STM32F30x and STM32F3x8	STM32F37x line	STM32F2/ F4/F7 series	STM32L0 series	STM32L1 series	STM32L4 series
Advanced		TIM1	TIM1	TIM1	TIM1		TIM1			TIM1
				TIM8	TIM8		TIM8			TIM8
					TIM20					
General purpose	16-bit		TIM2	TIM2				TIM2	TIM2	
		TIM3	TIM3	TIM3	TIM3	TIM3	TIM3	TIM3	TIM3	TIM3
			TIM4	TIM4	TIM4	TIM4	TIM4	TIM21	TIM4	TIM4
			TIM5	TIM5		TIM19		TIM22		
	32-bit	TIM2			TIM2	TIM2	TIM2			TIM2
						TIM5	TIM5		TIM5	TIM5
Basic		TIM6	TIM6	TIM6	TIM6	TIM6	TIM6	TIM6	TIM6	TIM6
		TIM7	TIM7	TIM7	TIM7	TIM7	TIM7	TIM7	TIM7	TIM7
						TIM18				
1-channel			TIM10				TIM10		TIM10	
			TIM11				TIM11		TIM11	
			TIM13	TIM13		TIM13	TIM13			
		TIM14	TIM14	TIM14		TIM14	TIM14			
2-channels			TIM9				TIM9		TIM9	
			TIM12	TIM12		TIM12	TIM12			
1-channel with one complementary output		TIM15		TIM15	TIM15	TIM16				TIM16
						TIM17				TIM17
2-channel with one complementary output		TIM16		TIM16	TIM16	TIM15				TIM15
		TIM17		TIM17	TIM17					
High-resolution					HRTIM1					
Low-power							LPTIM1	LPTIM1		LPTIM1
										LPTIM2

В таблице 3 приведен список всех таймеров, реализованных микроконтроллерами, оснащающими шестнадцать плат Nucleo, которые мы рассматриваем в данной книге. Важно подчеркнуть некоторые моменты, представленные в таблице 3:

1. STM32F411RE, STM32F401RE и STM32F103RB не предоставляют базовый таймер.
2. В столбце «MAX clock speed» указывается максимальная тактовая частота для всех таймеров в конкретном микроконтроллере STM32. Это означает, что максимальная тактовая частота таймера зависит от шины, к которой он подключен. Обязательно сверяйтесь с техническим описанием, чтобы определить, к какой шине подключен таймер, и используйте представление CubeMX Clock Configuration для определения сконфигурированной частоты шины.
3. Микроконтроллер STM32F410RB, представленный на рынке в начале 2016 года, реализует функцию, характерную для серии STM32L0/L4: таймер с пониженным энергопотреблением.

При работе с таймерами важно иметь прагматичный подход. В противном случае достаточно легко потеряться в их параметрах и в соответствующих процедурах HAL (модули HAL\_TIM и HAL\_TIM\_EX являются одними из наиболее четко сформулированных в CubeHAL).

Таблица 3 Какие таймеры реализованы в каждом микроконтроллере STM32 из шестнадцати плат Nucleo

	Nucleo P/N	Basic Timers	General Purpose Timers	Advanced Timers	High-resolution Timers	Low-power Timers	MAX clock speed
	NUCLEO-F446RE	TIM6-7	TIM2-5 TIM9-14	TIM1 TIM8	-	-	90/180MHz
	NUCLEO-F411RE	-	TIM2-5 TIM9-11	TIM1	-	-	100MHz
	NUCLEO-F410RB	TIM6	TIM5 TIM9 TIM11	TIM1	-	LPTIM1	100MHz
	NUCLEO-F401RE	-	TIM2-5 TIM9-11	TIM1	-	-	84MHz
	NUCLEO-F334R8	TIM6-7	TIM2-3 TIM15-17	TIM1	HRTIM1	-	72/144MHz
	NUCLEO-F303RE	TIM6-7	TIM2-4 TIM15-17	TIM1 TIM8 TIM20	-	-	72/144MHz
	NUCLEO-F302R8	TIM6	TIM2 TIM15-17	TIM1	-	-	72/144MHz
	NUCLEO-F103RB	-	TIM3-4	TIM1	-	-	64/72MHz
	NUCLEO-F091RC	TIM6-7	TIM2-3 TIM14-17	TIM1	-	-	48MHz
	NUCLEO-F072RB	TIM6-7	TIM2-3 TIM14-17	TIM1	-	-	48MHz
	NUCLEO-F070RB	TIM6-7	TIM3 TIM14-17	TIM1	-	-	48MHz
	NUCLEO-F030R8	TIM6	TIM3 TIM14-17	TIM1	-	-	48MHz
	NUCLEO-L476RG	TIM6-7	TIM2-5 TIM15-17	TIM1 TIM8	-	LPTIM1 LPTIM2	80MHz
	NUCLEO-L152RE	TIM6-7	TIM2-5 TIM9-11	-	-	-	32MHz
	NUCLEO-L073RZ	TIM6-7	TIM2-3 TIM21-22	-	-	LPTIM1	32MHz
	NUCLEO-L053R8	TIM6	TIM2-3 TIM21-22	-	-	LPTIM1	32MHz

### Базовые таймеры

Базовые таймеры TIM6, TIM7 и TIM188 – самые простые таймеры, доступные в ассортименте STM32. Даже если они не предоставляются всеми микроконтроллерами STM32, важно подчеркнуть, что таймеры STM32 спроектированы так, что более продвинутые таймеры реализуют те же функции (таким же образом), что и менее мощные, как показано на рисунке 1. Это означает, что вполне возможно использовать таймер общего назначения так же, как и базовый таймер. CubeHAL также отражает эту

аппаратную реализацию: базовые операции для всех таймеров выполняются с использованием функций HAL\_TIM\_Base\_XXX.



Рисунок 1 Отношение между тремя основными категориями таймеров

На каждый таймер ссылаются с использованием экземпляра структуры Си TIM\_HandleTypeDef, которая определена следующим образом:

```
typedef struct {  
    TIM_TypeDef *Instance; /* Указатель на дескриптор таймера */  
    TIM_Base_InitTypeDef Init; /* Требуемые параметры TIM для генерации  
временного отсчета */  
    HAL_TIM_ActiveChannel Channel; /* Активные каналы */  
    DMA_HandleTypeDef *hdma[7]; /* Массив дескрипторов DMA */  
    HAL_LockTypeDef Lock; /* Блокировка объекта TIM */  
    __IO HAL_TIM_StateTypeDef State; /* Состояние работы TIM */  
} TIM_HandleTypeDef;
```

Давайте более подробно рассмотрим наиболее важные поля данной структуры.

1. Instance (экземпляр): указатель на дескриптор таймера (TIM), который мы будем использовать. Например, TIM6 является одним из базовых таймеров, доступных в большинстве микроконтроллеров STM32.

2. Init: это экземпляр структуры Си TIM\_Base\_InitTypeDef, которая используется для конфигурации базовых функций таймера. Мы рассмотрим ее более подробно в ближайшее время.

3. Channel: определяет количество активных каналов в таймерах, обеспечивающих один или несколько входных/выходных каналов (это не относится к

базовым таймерам). Оно может принимать одно или несколько значений из перечисления enum HAL\_TIM\_ActiveChannel.

4. \*hdma[7]: это массив, содержащий указатели на дескрипторы DMA\_HandleTypeDef для запросов к DMA, связанных с таймером. Как мы увидим позже, таймер может генерировать до семи запросов к DMA, используемых для управления его функциями.

5. State (состояние): используется внутри HAL для отслеживания состояния таймера. Все действия по конфигурации таймера выполняются с использованием экземпляра структуры Си TIM\_Base\_InitTypeDef, которая определена следующим образом:

```
typedef struct {  
    uint32_t Prescaler; /* Задаёт значение предделителя, используемое для  
деления тактового сигнала TIM. */  
    uint32_t CounterMode; /* Задаёт режим отсчёта. */  
    uint32_t Period; /* Задаёт значение периода, которое будет загружено в  
активный регистр автоперезагрузки (ARR) при следующем событии обновления.  
*/  
    uint32_t ClockDivision; /* Задаёт делитель тактового сигнала таймера. */  
    uint32_t RepetitionCounter; /* Задаёт значение кратности отсчётов  
счетчика. */  
} TIM_Base_InitTypeDef;
```

6. Prescaler (предделитель): делит тактовый сигнал таймера на коэффициент в диапазоне от 1 до 65535 (это означает, что регистр предделителя имеет 16-разрядное разрешение). Например, если шина, к которой подключен таймер, работает на частоте 48 МГц, то значение предделителя, равное 48, понижает частоту отсчета до 1 МГц.

7. CounterMode (режим отсчета): задает направление отсчета таймера и может принимать одно из значений из таблицы 4. Некоторые режимы отсчета доступны только для таймеров общего назначения и расширенного управления. Для базовых таймеров определяется только TIM\_COUNTERMODE\_UP.

8. Period (период): задает максимальное значение счетчика таймера, прежде чем он повторно перезапустит отсчет. Он может принимать значение от 0x1 до 0xFFFF (65535) для 16-разрядных таймеров и от 0x1 до 0xFFFF FFFF для таймеров TIM2 и TIM5 в микроконтроллерах, реализующих их как 32-разрядные таймеры. Если Period установлен в 0x0, таймер не запускается.

9. ClockDivision (делитель тактового сигнала таймера): это битовое поле задает соотношение деления между внутренним тактовым сигналом таймера и тактовой частотой дискретизации, используемых цифровыми фильтрами на выводах ETRx и T1x. Может принимать одно из значений из таблицы 5 и доступно только для таймеров общего назначения и расширенного управления. Мы будем изучать цифровые фильтры на входных выводах таймера позже в этой главе. Это поле также используется генератором мертвого времени.

10. RepetitionCounter (кратность отсчетов счетчика): каждый таймер имеет специальный регистр обновления (update register), отслеживающий состояние переполнения/опустошения счетчика таймера. При этом также может генерироваться определенный IRQ, как мы увидим далее. RepetitionCounter сообщает, сколько раз таймеры переполняют/опустошат счетчик до того, как установится регистр обновления, и будет вызвано соответствующее событие (если разрешено). RepetitionCounter доступен только для таймеров расширенного управления.

Таблица 4 Доступные режимы отсчета таймера

Режим отсчета	Описание
TIM_COUNTERMODE_UP	Таймер считает от нуля до значения Period (которое не может быть выше, чем разрешение таймера – 16/32-разрядного), а затем генерирует событие переполнения.
TIM_COUNTERMODE_DOWN	Таймер считает вниз от значения Period до нуля, а затем генерирует событие опустошения.
TIM_COUNTERMODE_CENTERALIGNED1	В режиме выравнивания по центру счетчик считает от 0 до значения Period – 1, генерирует событие переполнения, затем считает от значения Period до 1 и генерирует событие опустошения счетчика. Затем он возобновляет отсчет с 0. Флаг прерывания сравнения выходного сигнала (Output compare interrupt flag) каналов, сконфигурированных в режиме выхода, устанавливается при отсчете счетчика вниз.



TIM_COUNTERMODE_CENTRALIGNED2	То же, что и TIM_COUNTERMODE_CENTRALIGNED1, но флаг прерывания сравнения выходного сигнала каналов, сконфигурированных в режиме выхода, устанавливается при отсчете счетчика вверх.
TIM_COUNTERMODE_CENTRALIGNED3	То же, что TIM_COUNTERMODE_CENTRALIGNED1, но флаг прерывания сравнения выходного сигнала каналов, сконфигурированных в режиме выхода, устанавливается, когда счетчик отсчитывает вверх и вниз.

Таблица 5 Доступные режимы ClockDivision для таймеров общего назначения и расширенного управления

Режимы делителя тактового сигнала таймера	Описание
TIM_CLOCKDIVISION_DIV1	Вычисляет 1 выборку входного сигнала на выводах ETRx и TIx
TIM_CLOCKDIVISION_DIV2	Вычисляет 2 выборки входного сигнала на выводах ETRx и TIx
TIM_CLOCKDIVISION_DIV4	Вычисляет 4 выборки входного сигнала на выводах ETRx и TIx

## **Лекция на тему «Модуль DMA»**

### **Рассматриваемые вопросы:**

1. Необходимость DMA и роль внутренних шин.
2. Контроллер DMA.
3. Реализация DMA в микроконтроллерах F0/F1/F3/L1.
4. Реализация DMA в микроконтроллерах F2/F4/F7.
5. Реализация DMA в микроконтроллерах L0/L4.

### **Теоретический материал:**

Каждое встроенное приложение должно обмениваться данными с внешним миром или управлять внешними периферийными устройствами. Например, наш микроконтроллер может обмениваться сообщениями с другими модулями на печатной плате, используя UART, или он может хранить данные во внешней Flash-памяти, используя один из доступных интерфейсов SPI. Все это включает в себя передачу определенного объема данных между внутреннего SRAM или Flash-памятью и периферийными регистрами, и для выполнения передачи требуется определенное количество тактовых циклов ЦПУ. Это приводит к потере вычислительной мощности (процессор занят процессом передачи), к снижению общей производительности и, в конечном итоге, к потере важных асинхронных событий.

Контроллер прямого доступа к памяти (Direct Memory Access, DMA) – это специализированный и программируемый аппаратный модуль, позволяющий периферийным устройствам микроконтроллера получать доступ к внутренней памяти без вмешательства ядра Cortex-M. ЦПУ полностью освобождается от накладных расходов (или, как говорят, от «оверхеда», overhead), порождаемых передачей данных (за исключением накладных расходов, связанных с конфигурацией DMA), благодаря чему он может выполнять другие действия параллельно.

DMA спроектирован так, чтобы работать обоими способами (то есть позволяет передавать данные из памяти в периферийные устройства и наоборот), при этом все микроконтроллеры STM32 предоставляют как минимум один контроллер DMA, но большинство из них имеют два независимых контроллера DMA. DMA является «продвинутой» функцией современных микроконтроллеров, и начинающие пользователи склонны считать ее слишком сложной в использовании. Вместо этого, концепции, лежащие в основе DMA, действительно просты, и как только вы их поймете, будет довольно легко их использовать. Более того, хорошая новость

заключается в том, что CubeHAL предназначен для абстрагирования от большинства этапов конфигурации DMA для используемого периферийного устройства, оставляя пользователю ответственность за предоставление лишь нескольких базовых конфигураций.

Прежде чем мы сможем проанализировать функции, предлагаемые модулем HAL\_DMA, важно понять некоторые фундаментальные концепции, лежащие в основе контроллера DMA. Попытаемся обобщить наиболее важные аспекты, которые следует учитывать при изучении данного периферийного устройства. Кроме того, они пытаются устранить различия в реализации между STM32F2/4/7 и другими семействами STM32.

### **Необходимость DMA и роль внутренних шин**

Каждое периферийное устройство в микроконтроллере STM32 должно обмениваться данными с внутренним ядром Cortex-M. Некоторые из них преобразуют эти данные в электрические сигналы I/O для обмена ими с внешним миром в соответствии с заданным протоколом связи (например, в случае интерфейсов UART или SPI). Другие просто спроектированы так, что доступ к их регистрам внутри отображаемой области периферийной памяти (от 0x4000 0000 до 0x5FFF FFFF) вызывает изменение их состояния (например, регистр GPIOx->ODR управляет состоянием всех подключенных к порту I/O). Однако имейте в виду, что с точки зрения процессора это также подразумевает передачу памяти между ядром и периферией.

Ядро микроконтроллера теоретически может быть спроектировано таким образом, чтобы каждое периферийное устройство имело свою собственную область хранения, и она, в свою очередь, могла бы быть тесно связана с ядром ЦПУ, чтобы минимизировать затраты, связанные с передачами памяти. Однако это усложняет архитектуру микроконтроллера, требуя много кремния и больше «активных компонентов», потребляющих энергию. Таким образом, подход, используемый во всех встроенных микроконтроллерах, заключается в использовании некоторых частей внутренней памяти (в том числе и SRAM) в качестве временных областей хранения для различных периферийных устройств. Пользователь сам решает, сколько места уделить данным областям. Например, давайте рассмотрим этот фрагмент кода:

```
uint8_t buf[20];
```

```
...
```

```
HAL_UART_Receive(&huart2, buf, 20, HAL_MAX_DELAY);
```

Здесь мы собираемся считать двадцать байт по интерфейсу UART2, и, следовательно, мы выделяем массив (временное хранилище) того же размера внутри

SRAM. Функция HAL\_UART\_Receive() будет двадцать раз считывать регистр данных huart2.Instance->DR для передачи байт из периферийного устройства во внутреннюю память, а также будет опрашивать флаг RXNE интерфейса UART, чтобы определить, когда новые данные готовы к отправке. Процессор будет задействован во время этих операций (см. рисунок 1), несмотря на то что его роль «ограничена» перемещением данных из периферийного устройства в SRAM4.

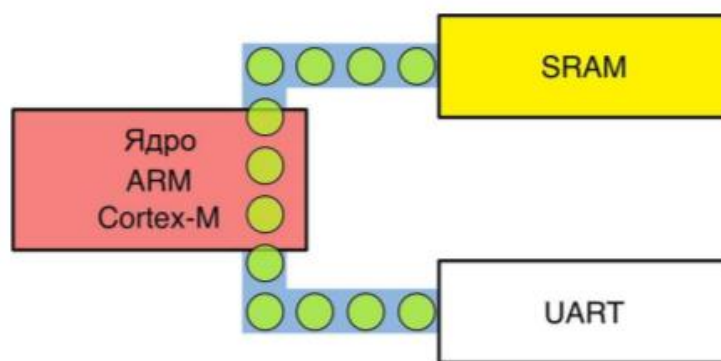


Рисунок 1 Поток данных при передаче из периферийного устройства в SRAM

Хотя данный подход с одной стороны упрощает проектирование аппаратного обеспечения, с другой стороны он вводит накладные расходы на производительность. Ядро Cortex-M «отвечает» за загрузку данных из периферийной памяти в SRAM, и это блокирующая операция, которая не только не позволяет ЦПУ выполнять другие действия, но и также требует, чтобы ЦПУ ждало «более медленные» модули, пока они не завершат свою работу (некоторые периферийные устройства STM32 подключаются к ядру с помощью более медленных шин). По этой причине высокопроизводительные микроконтроллеры предоставляют аппаратные модули, предназначенные для передачи данных между периферийными устройствами и централизованным буферным хранилищем, то есть SRAM.

Прежде чем углубляться в подробности DMA, лучше рассмотреть все компоненты, участвующие в процессе передачи данных с периферийного устройства в память SRAM и наоборот. Для удобства архитектура шин микроконтроллера STM32F030 показана на рисунке 2. Она сильно отличается от других более производительных семейств STM32.

Рисунок говорит нам о нескольких важных моментах:

1. И ядро Cortex-M, и контроллер DMA1 взаимодействуют с другими периферийными устройствами микроконтроллера через последовательность шин. Если это все еще неясно, важно отметить, что Flash-память и память SRAM также являются

компонентами, внешними по отношению к ядру микроконтроллера, и поэтому они должны взаимодействовать друг с другом через межшинные соединения.

2. И ядро Cortex-M, и контроллер DMA1 являются ведущими (masters). Это означает, что они являются единственными устройствами, которые могут начать транзакцию по шине. Однако доступ к шине должен регулироваться таким образом, чтобы они не могли получить доступ к одному и тому же ведомому (slave) периферийному устройству одновременно.

3. Шинная матрица (BusMatrix) управляет последовательностью доступа между ядром Cortex-M и контроллером DMA1. Арбитраж производится по алгоритму циклического перебора Round Robin для управления доступом к шине. Шинная матрица состоит из двух ведущих шин (ЦПУ, DMA) и четырех ведомых шин: интерфейс Flash-памяти, SRAM, АНВ1 с мостом АНВ-АРВ (где АРВ – Advanced Peripheral Bus – продвинутая периферийная шина) и АНВ2. Шинная матрица также позволяет автоматически соединять между собой несколько периферийных устройств.

4. Системная шина соединяет ядро Cortex-M с шинной матрицей.

5. Шина DMA соединяет ведущий интерфейс DMA продвинутой высокопроизводительной шины (Advanced High-performance Bus, АНВ) с шинной матрицей.

6. Мост АНВ-АРВ обеспечивает полностью синхронные соединения между шиной АНВ и шиной АРВ, куда подключена большая часть периферийных устройств.

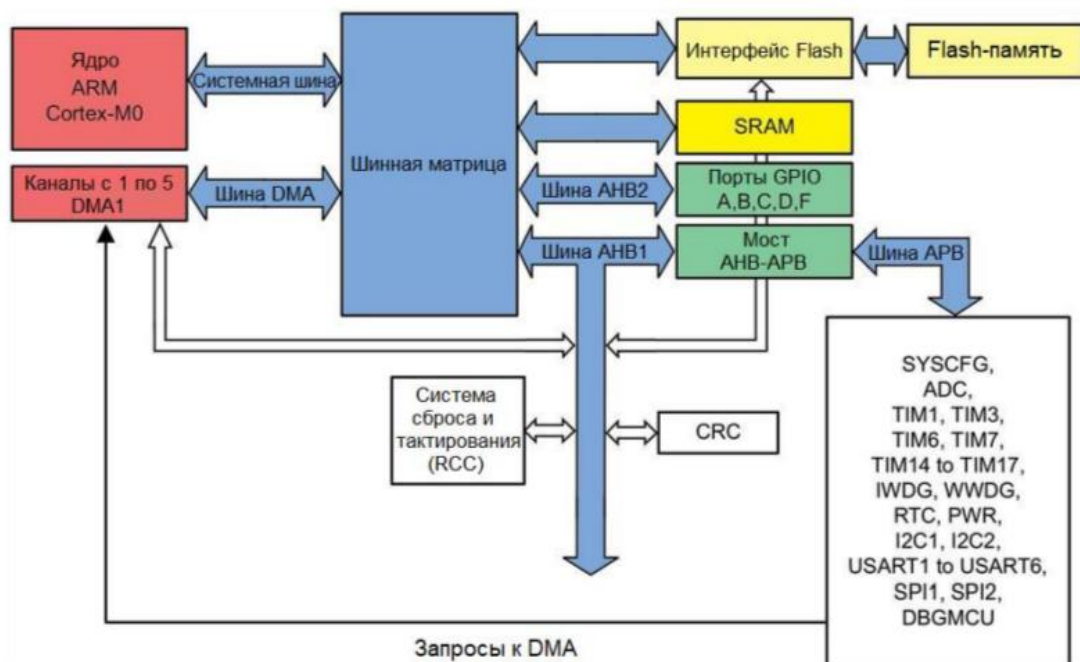


Рисунок 2 Архитектура шин микроконтроллера STM32F030

Аббревиатуры АНВ, АНВ1, АНВ2, АРВ и т. д. в мире STM32 всегда сбивают с толку. Они представляют собой две вещи одновременно:

1. Это аппаратные компоненты, используемые для подключения различных модулей внутри микроконтроллера, чтобы они могли обмениваться данными. Они могут тактироваться разными источниками тактовых сигналов с разными скоростями (то есть частотами). Это означает, что чтение через медленные шины может создать «заторы» в вашем приложении.

2. Они являются частью более общей спецификации Продвинутой архитектуры шин микроконтроллеров ARM (Advanced Microcontroller Bus Architecture, AMBA), которая определяет, как различные функциональные модули взаимодействуют друг с другом внутри микроконтроллера. AMBA является открытым стандартом, и она реализована в различных выпусках (и разновидностях) во всех процессорах ARM Cortex (включая Cortex-A и Cortex-R).

Мы остановились на еще одном моменте на рисунке 2: стрелка запросы к DMA, которая идет от модуля периферийных устройств (белый прямоугольник) к контроллеру DMA1. Что они делают? Контроллер NVIC уведомляет ядро Cortex-M об асинхронных запросах прерываний (interrupt requests, IRQs), поступающих от периферийных устройств. Когда периферийное устройство готово что-то сделать (например, UART готов к приему данных или переполнен таймер), оно уведомляет соответствующую ему линию запроса прерывания IRQ. Ядро выполняет за заданное количество тактовых циклов соответствующую ISR, которая содержит код, необходимый для обработки IRQ. Не забывайте, что периферийные устройства являются ведомыми устройствами: они не могут получить доступ к шине независимо. Чтобы начать транзакцию, всегда необходимо ведущее устройство. Однако поскольку периферийные устройства являются ведомыми устройствами, если мы используем контроллер DMA для передачи данных из периферийных устройств в память, у нас должен быть способ уведомить его о том, что периферийные устройства готовы к обмену данными. По этой причине доступно выделенное количество линий запросов от периферийных устройств к контроллеру DMA.

### **Контроллер DMA**

В каждом микроконтроллере STM32 контроллер DMA представляет собой аппаратный модуль, который:

- имеет два ведущих порта, называемых, соответственно, периферийным портом и портом памяти, которые подключены к продвинутой высокопроизводительной шине (АНВ), один может подключаться к ведомому

периферийному устройству, а другой – к контроллеру памяти (SRAM, Flash-память, FSMC и так далее); в некоторых контроллерах DMA периферийный порт также может взаимодействовать с контроллером памяти, что позволяет передавать данные из памяти в память (memory-to-memory);

- имеет один ведомый порт, подключенный к шине АНВ и используемый для программирования контроллера DMA от другого ведущего устройства, то есть ЦПУ;

- имеет несколько независимых и программируемых каналов (источников запросов), каждый из которых подключается к используемой периферийной линии запроса (UART\_TX, TIM\_UP и так далее – количество и тип запросов для канала устанавливаются во время проектирования микроконтроллера);

- позволяет назначать разные приоритеты каналам для того, чтобы разрешать конфликты доступа к памяти, отдавая более высокий приоритет более быстрым и важным периферийным устройствам;

- позволяет передавать данные в обоих направлениях, то есть из памяти в периферию (memory-to-peripheral) и из периферии в память (peripheral-to-memory).

Каждый микроконтроллер STM32 предоставляет различное количество контроллеров DMA и каналов в соответствии со своим семейством и видом поставки (sale type). В таблице 1 приведено их точное количество для микроконтроллеров STM32, оснащающих все платы Nucleo.

Таблица 1 Количество контроллеров DMA/каналов, доступных в каждой плате Nucleo

Nucleo P/N	Available DMAs	# Channels (DMA1 - DMA2)	MEM2MEM DMA
NUCLEO-F446RE	2	8 - 8	DMA2
NUCLEO-F411RE	2	8 - 8	DMA2
NUCLEO-F410RB	2	8 - 8	DMA2
NUCLEO-F401RE	2	8 - 8	DMA2
NUCLEO-F334R8	1	7	DMA1
NUCLEO-F303RE	2	7 - 5	DMA1 + DMA2
NUCLEO-F302R8	1	7	DMA1
NUCLEO-F103RB	2	7 - 5	DMA1 + DMA2
NUCLEO-F091RC	2	5 - 7	DMA1 + DMA2
NUCLEO-F072RB	2	5 - 7	DMA1 + DMA2
NUCLEO-F070RB	1	5	DMA1
NUCLEO-F030R8	1	5	DMA1
NUCLEO-L476RG	2	7 - 7	DMA1 + DMA2
NUCLEO-L152RE	2	7 - 5	DMA1 + DMA2
NUCLEO-L073RZ	1	7	DMA1
NUCLEO-L053R8	1	7	DMA1

Данные характеристики являются общими для всех микроконтроллеров STM32. Однако семейства STM32F2/F4/F7 предоставляют более совершенный контроллер DMA в сочетании с многоуровневой шинной матрицей, которая позволяет ускорять и распараллеливать передачи через DMA. По этой причине мы собираемся рассматривать их отдельно.

### Реализация DMA в микроконтроллерах F0/F1/F3/L1

На рисунке 3 показано представление DMA в микроконтроллерах серий F0/F1/F3/L1. Здесь для простоты показана только одна линия запроса, но каждый DMA реализует линию запроса для каждого канала. Каждая линия запроса имеет переменное количество источников запросов периферии, подключенной к ней. Во время разработки чипа канал привязывается к неизменному набору периферийных устройств. Однако на одном канале одновременно может быть активно только одно периферийное устройство. Например, в таблице 2 показано, как каналы связаны с периферийными устройствами в микроконтроллере STM32F030.

Каждая линия запроса также может быть сработана «программно». Эта способность используется для передач типа память-в-память. Каждый канал имеет конфигурируемый приоритет, позволяющий управлять доступом к шине АНВ. Внутренний арбитр управляет запросами, поступающими от каналов, в соответствии с конфигурируемым пользователем приоритетом. Если две линии запроса активируют



запрос и их каналы имеют одинаковый приоритет, то канал с меньшим номером выигрывает конфликт.

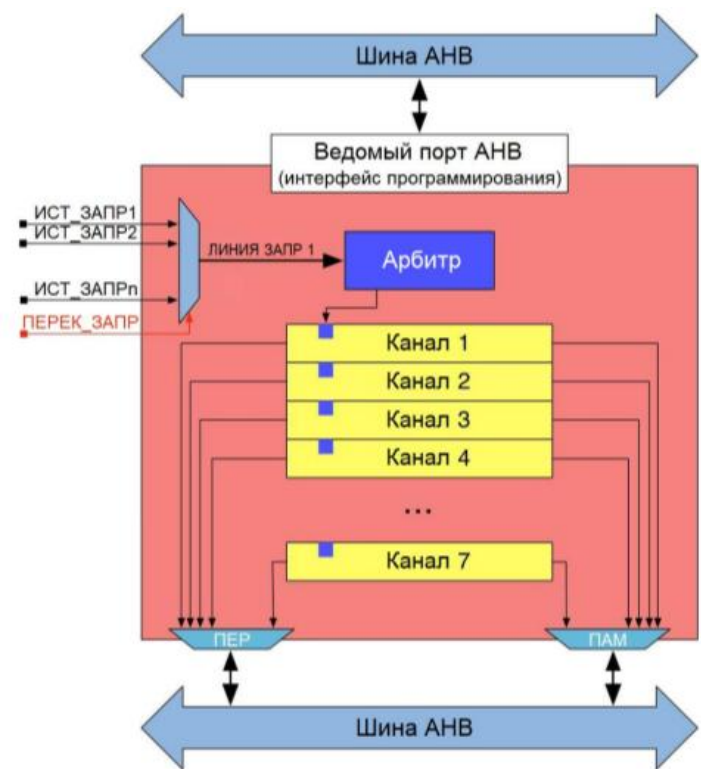


Рисунок 3 Представление структуры DMA (другие линии запроса опущены)

В зависимости от используемого вида поставки доступен один или два контроллера DMA, всего 12 независимых каналов (5 для DMA1 и 7 для DMA2). Например, как показано в таблице 2, STM32F030 предоставляет только DMA1 с 5 каналами.

Таблица 2 Как каналы связаны с периферийными устройствами в микроконтроллере STM32F030

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5
ADC	ADC	ADC	-	-	-
SPI	-	SPI1_RX	SPI1_TX	SPI2_RX	SPI2_TX
USART	-	USART1_TX USART3_TX	USART1_RX USART3_RX	USART1_TX USART2_TX	USART1_RX USART2_RX
I2C	-	I2C1_TX	I2C1_RX	I2C2_TX	I2C2_RX
TIM1	-	TIM1_CH1	TIM1_CH2	TIM1_CH4 TIM1_TRIG	TIM1_CH3 TIM1_UP

				TIM1_COM	
TIM3	-	TIM3_CH3	TIM3_CH4 TIM3_UP	TIM3_CH1 TIM3_TRIG	-
TIM6	-	-	TIM6_UP	-	-
TIM7	-	-	-	TIM7_UP	-
TIM15	-	-	-	-	TIM15_CH1 TIM15_UP TIM15_TRIG TIM15_COM
TIM16	-	-	TIM16_CH1 TIM16_UP	TIM16_CH1 TIM16_UP	-
TIM17	TIM17_CH1 TIM17_UP	TIM17_CH1 TIM17_UP	-	-	-

На рисунке 2 мы уже видели архитектуру шин STM32F030. Для полноты картины на рисунке 4 показана архитектура шин более производительного микроконтроллера с той же реализацией DMA (например, STM32F1). Как видите, два семейства имеют совершенно разную организацию внутренних шин.

Большинство микроконтроллеров STM32 имеют одинаковую компьютерную архитектуру, за исключением STM32F0 и STM32L0, спроектированных на базе ядер CortexM0/0+. По сути, они являются единственными ядрами Cortex-M, основанными на фонНеймановской архитектуре, по сравнению с другими ядрами Cortex-M, основанными на Гарвардской архитектуре. Принципиальное различие между этими двумя архитектурами заключается в том, что ядра Cortex-M0/0+ получают доступ к Flash-памяти, SRAM и периферийным устройствам по одной общей шине, в то время как другие ядра CortexM имеют две отдельные линии шин для доступа к Flash-памяти (одна для выборки команд, называемая шиной команд (instruction bus), или просто I-Bus или даже I-Code, и одна для доступа к константам, называемая шиной данных (data bus), или просто D-Bus или даже D-Code) и одну выделенную линию запроса для доступа к SRAM и периферийным устройствам (также называемую системной шиной (system bus) или просто S-Bus). Какие преимущества это дает нашим приложениям?

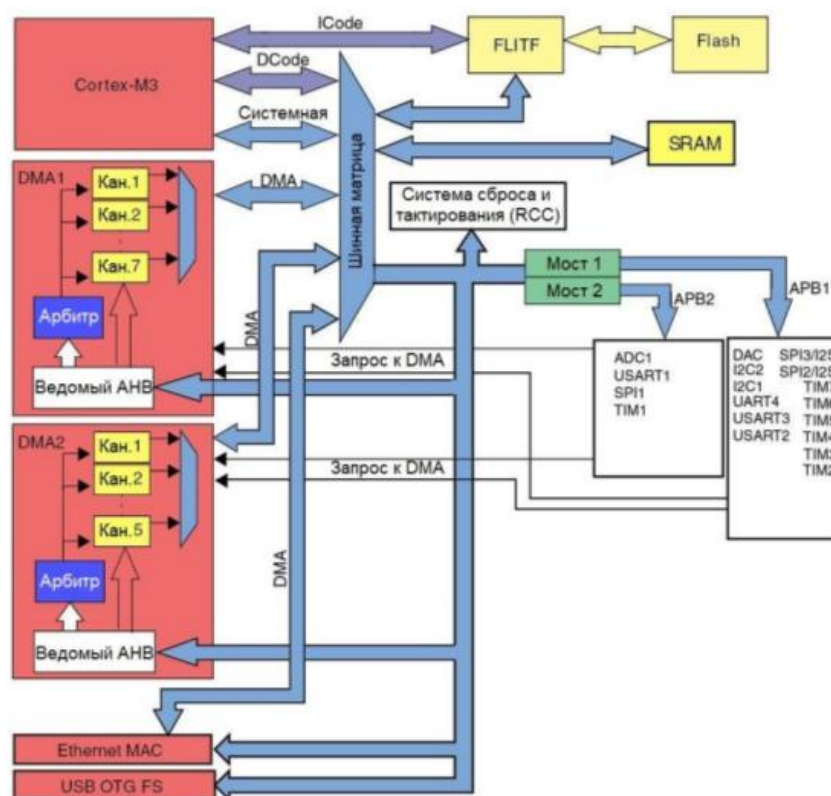


Рисунок 4 Архитектура шин в микроконтроллере STM32F1 от линейки Connectivity Line

В ядрах Cortex-M0/0+ контроллер DMA и ядро Cortex соперничают за доступ к памяти и периферийным устройствам через шинную матрицу. Предположим, что ЦПУ выполняет математические операции над данными, содержащимися в его внутренних регистрах (R0-R14). Если контроллер DMA передает данные в SRAM, шинная матрица разрешает доступ от ядра Cortex-M0/0+ к Flash-памяти для загрузки следующей инструкции для выполнения. Поэтому данное ядро «стопорится» в ожидании своей очереди (подробнее об этом чуть позже). В других ядрах Cortex-M процессор может получать доступ к Flash-памяти независимо, что повышает общую производительность. Это принципиальное отличие, оправдывающее цену микроконтроллеров STM32F0: они не только могут иметь меньше SRAM и Flash-памяти и работать на более низких частотах, но и имеют более простую и менее производительную архитектуру.

Однако важно отметить, что шинная матрица реализует алгоритмы планирования (scheduling policies), избегающие слишком долгого «застопоривания» определенного ведущего устройства (ЦПУ и DMA в микроконтроллерах линеек Value Line, или ЦПУ, DMA, Ethernet и USB в микроконтроллерах линеек Connectivity Line). Каждая передача через DMA состоит из четырех фаз: выборка и фаза арбитража, фаза вычисления адреса, фаза доступа к шине и фаза окончательного подтверждения

(которая используется, чтобы сигнализировать о том, что передача была завершена). Каждая фаза занимает один тактовый цикл, за исключением фазы доступа к шине, которая может длиться большее количество тактовых циклов. Однако ее максимальная продолжительность фиксирована, и шинная матрица гарантирует, что в конце фазы подтверждения будет назначено другое ведущее устройство для доступа к шине. Как мы увидим в следующем параграфе, семейства STM32F2/F4/F7 допускают более продвинутый параллелизм при доступе к ведомым устройствам. Однако подробности этих аспектов выходят за рамки данной книги. Настоятельно рекомендуется взглянуть на AN403110 от ST, чтобы лучше понять их.

Наконец, DMA также может выполнять передачи данных типа периферия-в-периферию (*peripheral-to-peripheral*) при определенных условиях, как мы увидим далее.

### **Реализация DMA в микроконтроллерах F2/F4/F7**

В микроконтроллерах STM32F2/F4/F7 реализован более совершенный контроллер DMA, как показано на рисунке 5. Он предлагает более высокую степень гибкости по сравнению с контроллером DMA, рассмотренным в других микроконтроллерах STM32. Каждый контроллер DMA реализует 8 разных потоков. Каждый поток предназначен для управления запросами на доступ к памяти с одного или нескольких периферийных устройств. Каждый поток может иметь до 8 каналов (запросов) в общей сложности (но имейте в виду, что только один канал/запрос может быть активным одновременно в потоке), и они имеют арбитра для обработки приоритета между запросами к DMA. Кроме того, каждый поток может по выбору предоставлять (является вариантом конфигурации) 32-разрядный буфер памяти логики «первым пришёл/первым ушёл» (*first-in/first-out, FIFO*) глубиной в четыре слова. FIFO используется для временного хранения данных, поступающих из источника, до его передачи в пункт назначения. Каждый поток также может быть запущен «программно». Данная возможность используется для выполнения передач типа память-в-память, но она ограничена только контроллером DMA2, как обозначено в таблице 1.

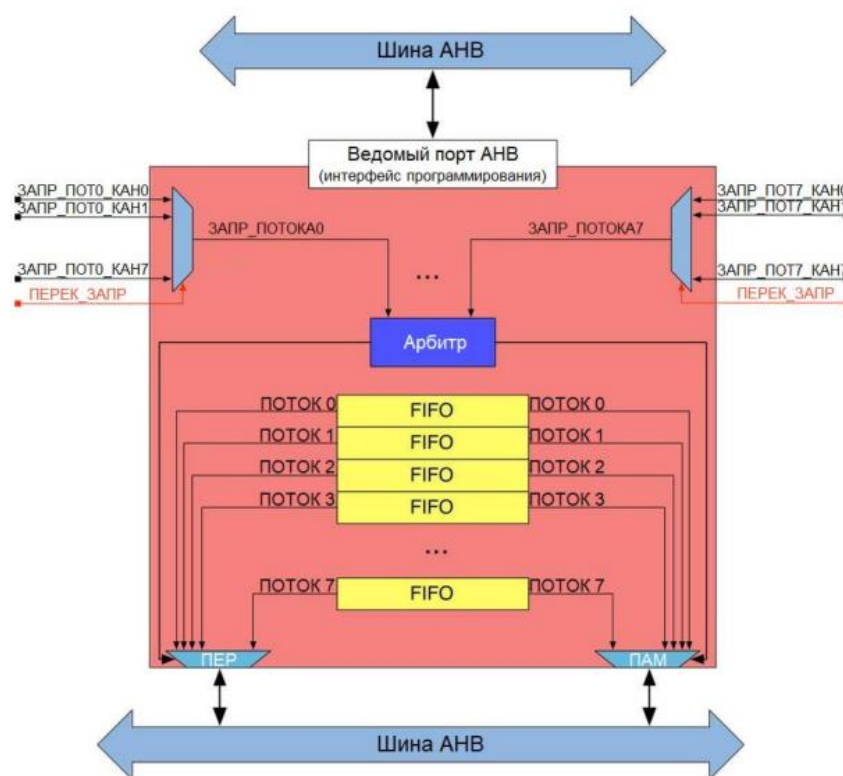


Рисунок 5 Архитектура DMA в микроконтроллере STM32F2/F4/F7

Каждый микроконтроллер STM32F2/F4/F7 имеет два контроллера DMA, что в сумме дает 16 независимых потоков. Как и в других микроконтроллерах STM32, канал связан с фиксированным набором периферийных устройств на этапе проектирования микросхемы. Таблица 3 показывает отображение запросов потоков/каналов DMA1 в микроконтроллере STM32F401RE. Микроконтроллеры STM32F2/F4/F7 включают в себя архитектуру с несколькими ведущими и несколькими ведомыми шинами, состоящую из:

1. 8 ведущих шин:

- шина ядра Cortex I-bus;
- шина ядра Cortex D-bus;
- шина ядра Cortex S-bus;
- шина DMA1 к памяти;
- шина DMA2 к памяти;
- шина DMA2 к периферии;
- шина DMA к устройству Ethernet (если доступна);
- шина DMA к устройству USB high-speed (если доступна).

2. 8 ведомых шин:

- шина внутренней Flash-памяти I-Code;

- шина внутренней Flash-памяти D-Code;
- шина основного внутреннего SRAM1;
- шина вспомогательного внутреннего SRAM2 (если доступна);
- шина вспомогательного внутреннего SRAM3 (если доступна);
- шина АНВ1 периферии, включая мосты АНВ-АРВ и шину АРВ периферии;
- шина АНВ2 периферии;
- шина АНВ3 периферийного устройства FMC (если доступна).

Таблица 3 Отображение запросов потоков/каналов к DMA1 в микроконтроллере STM32F401RE

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX		SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX		SPI3_TX
Channel 1	I2C1_RX	I2C3_RX				I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1		I2S3_EXT_RX	TIM4_CH2	I2S2_EXT_TX	I2S2_EXT_TX	TIM4_UP	TIM4_CH3
Channel 3	I2S3_EXT_RX	TIM2_UP TIM2_CH3	I2C3_RX	I2S2_EXT_RX	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4						USART2_RX	USART2_TX	
Channel 5			TIM3_CH4 TIM3_UP		TIM3_CH1 TIM3_TRIGGER	TIM3_CH2		TIM3_CH3
Channel 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIGGER	TIM5_CH1	TIM5_CH4 TIM5_TRIGGER	TIM5_CH2	I2C3_TX	TIM5_UP	
Channel 7			I2C2_RX	I2C2_RX				I2C2_TX

Ведущие и ведомые устройства соединены между собой через многоуровневую шинную матрицу, обеспечивающую одновременный доступ от отдельных ведущих устройств, а также эффективную работу, даже когда несколько высокоскоростных периферийных устройств работают одновременно. Эта архитектура показана на рисунке 611 в случае линеек STM32F405/415 и STM32F407/417.

Многоуровневая шинная матрица позволяет разным ведущим устройствам одновременно выполнять передачу данных, если они обращаются к разным ведомым

модулям (но для данного DMA только один поток одновременно может иметь доступ к шине). Помимо Гарвардской архитектуры Cortex-M и двух АНВ-портов DMA, эта структура повышает параллелизм передачи данных, тем самым способствуя сокращению времени выполнения и оптимизации эффективности DMA и энергопотребления.

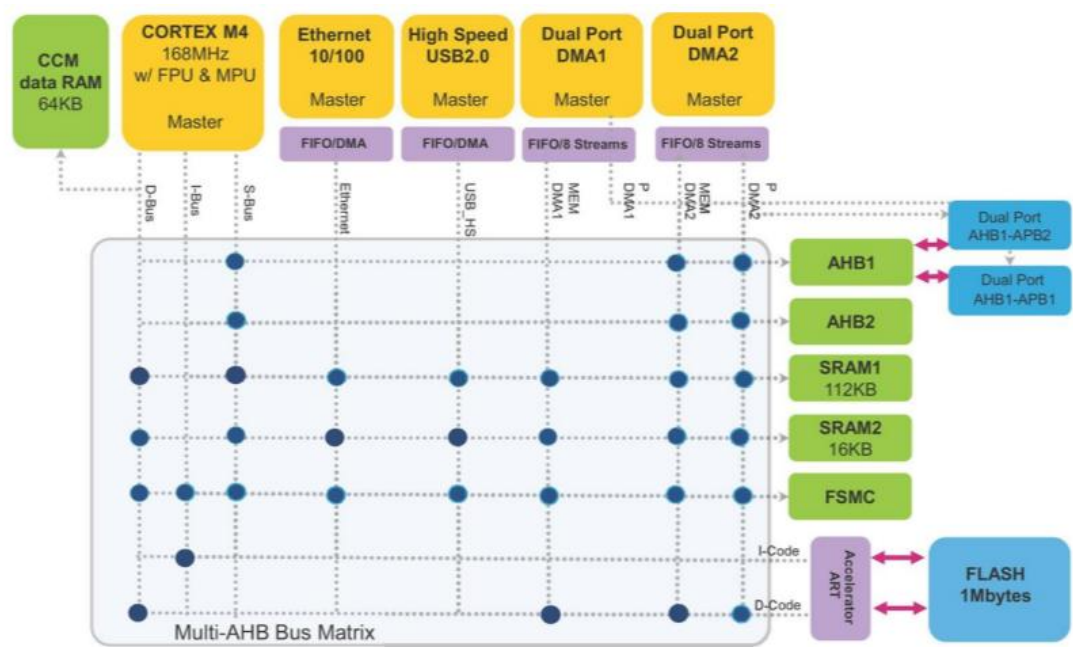


Рисунок 6 Многоуровневая шинная матрица в микроконтроллере STM32F405

### Реализация DMA в микроконтроллерах L0/L4

Реализация DMA в микроконтроллерах STM32L0/L4 имеет гибридный подход между реализацией DMA, заложенной в микроконтроллерах F0/F1/F3/L1 и F2/F4/F7. Фактически, она обеспечивает многопоточковый/многоканальный подход, но без поддержки внутренних буферов FIFO для каждого потока.

ST приняла другую номенклатуру (терминологию) для обозначения потоков и каналов в этих контроллерах DMA. Здесь потоки называются каналами, а каналы называются запросами (вероятно, эта номенклатурная схема более четкая, чем у потока/канала, используемого в микроконтроллерах F2/F4/F7). В таблице 4 показано отображение каналов/запросов в микроконтроллере STM32L053. Данная номенклатура влияет и на HAL, как мы увидим далее.

Таблица 4 Отображение каналов/запросов к DMA в микроконтроллере STM32L053

Request	Peripherals	Channel	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
---------	-------------	---------	-----------	-----------	-----------	-----------	-----------	-----------

number		1						
0	ADC	ADC	ADC	-	-	-	-	-
1	SPI1	-	SPI1_RX	SPI1_TX	-	-		
2	SPI2	-	-	-	SPI2_RX	SPI2_TX	SPI2_RX	SPI2_TX
3	USART1	-	USART1_TX	USART1_RX	USART1_TX	USART1_RX		
4	USART2	-	-	-	USART2_TX	USART2_RX	USART2_R X	USART2_T X
5	LPUART1	-	LPUART1_TX	LPUART1_RX	-	-	LPUART1_ RX	LPUART1_ TX
6	I2C1	-	I2C1_TX	I2C1_RX	-	-	I2C1_TX	I2C1_RX
7	I2C2	-	-	-	I2C2_TX	I2C2_RX	-	-
8	TIM2	TIM2_CH 3	TIM2_UP	TIM2_CH2	TIM2_CH4	TIM2_CH1	-	TIM2_CH2 TIM2_CH4
9	TIM6_UP/ DAC_chann el1	-	TIM6_UP/DAC _channel1	-	-	-	-	-
10	TIM3	-	TIM3_CH3	TIM3_CH4 TIM3_UP	-	TIM3_CH1	TIM3_TRI G	-
11	AES	AES_IN	AES_OUT	AES_OUT	-	AES_IN	-	-
12	USART4	-	USART4_RX	USART4_TX	-	-	USART4_R X	USART4_T X
13	USART5	-	USART5_RX	USART5_TX	-	-	USART5_R X	USART5_T X
14	I2C3	-	I2C3_TX	I2C3_RX	I2C3_TX	I2C3_RX	-	-
15	TIM7_UO/ DAC_chann el2	-	-	-	TIM7_UO/DAC _channel2	-	-	-



## Лекция на тему «Синхронные интерфейсы МК»

### Рассматриваемые вопросы:

1. Разрешение Выхода синхронизации.
2. Обзор схемы тактирования STM32.
3. Распределение тактового сигнала.

### Теоретический материал:

Почти каждая цифровая схема нуждается в способе синхронизации своих внутренних элементов или синхронизации с другими схемами. Тактовый сигнал – это генерируемый некоторым устройством периодический сигнал, являющийся самым распространенным видом сердечного ритма в цифровой электронике.

Однако один и тот же тактовый сигнал не может использоваться для питания всех компонентов и периферийных устройств, предоставляемых такими современными микроконтроллерами, как STM32. Кроме того, энергопотребление является критическим аспектом, напрямую связанным с тактовой частотой используемого периферийного устройства. Возможность выборочного отключения или уменьшения тактовой частоты некоторых компонентов микроконтроллера позволяет оптимизировать общее энергопотребление устройства. Все это требует, чтобы тактирование было организовано в иерархическую структуру, предоставляя разработчику возможность выбирать разные частоты и источники тактового сигнала.

### Распределение тактового сигнала

Тактовый сигнал (clock) – это генерируемый некоторым устройством прямоугольный сигнал, обычно с коэффициентом заполнения 50%, как показано на рисунке 1.

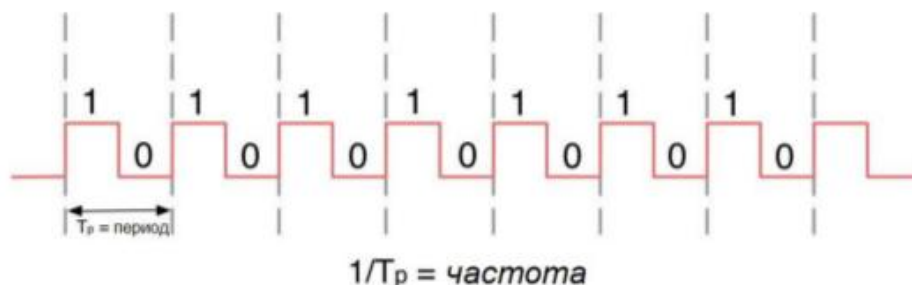


Рисунок 1 Типовой тактовый сигнал с коэффициентом заполнения 50%

Тактовый сигнал колеблется между уровнями напряжения  $V_L$  и  $V_H$ , которые для микроконтроллеров STM32 составляют часть напряжения питания VDD. Самым главным параметром тактового сигнала является частота, которая указывает, сколько раз напряжение переходит от уровня напряжения  $V_L$  к уровню  $V_H$  в секунду. Частота выражается в герцах.

Большинство микроконтроллеров STM32 могут тактироваться двумя отдельными источниками тактового сигнала: внутренним RC-генератором (называемым внутренним высокочастотным (High Speed Internal, HSI)) или внешним выделенным кварцевым генератором (называемым внешним высокочастотным (High Speed External, HSE)). Существует несколько причин предпочесть внешний кварцевый генератор внутреннему RCгенератору:

1. Внешний кварцевый генератор обеспечивает более высокую точность по сравнению с внутренней RC-цепью, погрешность которой оценивается с точностью до 1%, особенно когда рабочие температуры печатной платы далеки от температуры окружающей среды 25°C.

2. Некоторые периферийные устройства, особенно высокоскоростные, могут тактироваться только внешним выделенным кварцевым генератором, работающим на заданной частоте.

Вместе с высокочастотным генератором можно использовать другой источник тактового сигнала, подключаемый к низкочастотному генератору, который, в свою очередь, может тактироваться внешним кварцевым генератором (называемым внешним низкочастотным (Low Speed External, LSE)) или внутренним отдельным RC-генератором (называемым внутренним низкочастотным (Low Speed Internal, LSI)). Низкочастотный генератор используется для управления часами реального времени (Real Time Clock, RTC) и независимым сторожевым таймером (Independent Watchdog, IWDG).

Частота высокочастотного генератора не устанавливает рабочую частоту ни ядра CortexM, ни других периферийных устройств. Составная сеть распределения, также называемая схемой тактирования (clock tree), отвечает за распространение тактового сигнала в микроконтроллере STM32. Используя несколько программируемых блоков фазовой автоподстройки частоты (ФАПЧ, англ. Phase-Locked Loops, PLL) и предделителей, можно при необходимости увеличить/уменьшить частоту источника (смотри рисунок 2), в зависимости от характеристик, которые мы хотим достичь, максимальной скорости используемых периферийного устройства или шины и общего энергопотребления системы.

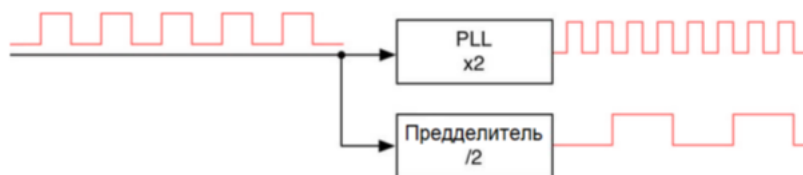


Рисунок 2 Как частота источника тактового сигнала увеличивается/уменьшается за счет использования блоков PLL и предделителей

### Обзор схемы тактирования STM32

Схема тактирования микроконтроллера STM32 может иметь достаточно сложную структуру. Даже в «простейших» микроконтроллерах STM32F0 внутренняя сеть тактирования может иметь до четырех каскадов PLL/предделителей, а мультиплексор системного тактового сигнала, англ. System Clock Multiplexer (также известный как переключатель системного тактового сигнала (System Clock Switch, SW)) может питаться несколькими замещающимися источниками.

Более того, подробное объяснение схемы тактирования каждого семейства STM32 является сложной задачей, которая также требует, чтобы мы сосредоточили наше внимание на конкретном номере устройства по каталогу (P/N). На самом деле на структуру схемы тактирования влияют главным образом следующие ключевые аспекты:

1. Основное семейство микроконтроллеров STM32. Например, все микроконтроллеры STM32F0 предоставляют только одну периферийную шину (APB1), которая может тактироваться на той же максимальной частоте ядра Cortex-M. Другие микроконтроллеры STM32 обычно предоставляют две периферийные шины, и только одна из них (APB2) может достигать максимальной тактовой частоты ЦПУ. И напротив, ни одна из периферийных шин, доступных в микроконтроллере STM32F7, не может достичь максимальной частоты ядра. В таблице 1 приведена максимальная тактовая частота для шин АНВ, APB1 и APB2 (с соответствующими тактовыми частотами таймеров) микроконтроллеров, оснащающих все платы Nucleo: вы можете отметить, что для некоторых микроконтроллеров STM32 можно достичь максимальной тактовой частоты только используя внешний HSE-генератор.

2. Тип и количество периферийных устройств, предоставляемых микроконтроллером. Сложность схемы тактирования увеличивается с увеличением количества доступных периферийных устройств. Кроме того, некоторые периферийные устройства требуют специальных источников тактового сигнала и частот, влияющих на число каскадов PLL.

3. Вид поставки и корпус микроконтроллера, определяющие действующий тип и количество предоставляемых периферийных устройств.

Таблица 1 Максимальные тактовые частоты шин АHB, APB1 и APB2 микроконтроллеров, оснащающих все платы Nucleo

Nucleo P/N	High-speed oscillator	AHB bus speed	APB1 peripheral clocks	APB1 timer clocks	APB2 peripheral clocks	APB2 timer clocks
NUCLEO-F446RE	HSE/HSI	180MHz	45MHz	90MHz	90MHz	180MHz
NUCLEO-F411RE	HSE/HSI	100MHz	50MHz	100MHz	100MHz	100MHz
NUCLEO-F410RB						
NUCLEO-F401RE	HSE/HSI	84MHz	42MHz	84MHz	84MHz	84MHz
NUCLEO-F334R8	HSE	72MHz	36MHz	72MHz	72MHz	72MHz
	HSI	64MHz	32MHz	64MHz	64MHz	64MHz
NUCLEO-F303RE	HSE/HSI	72MHz	36MHz	72MHz	72MHz	72MHz
NUCLEO-F302R8	HSE	72MHz	36MHz	72MHz	72MHz	72MHz
	HSI	64MHz	32MHz	64MHz	64MHz	64MHz
NUCLEO-F103RB	HSE	72MHz	36MHz	72MHz	72MHz	72MHz
	HSI	64MHz	32MHz	64MHz	64MHz	64MHz
NUCLEO-F091RC	HSE/HSI	48MHz	48MHz	48MHz	-	-
NUCLEO-F072RB						
NUCLEO-F070RB						
NUCLEO-F030R8						
NUCLEO-L476RG	HSE/HSI	80MHz	80MHz	80MHz	80MHz	80MHz
NUCLEO-L152RE	HSE/HSI	32MHz	32MHz	32MHz	32MHz	32MHz
NUCLEO-L073RZ	HSE/HSI	32MHz	32MHz	32MHz	32MHz	32MHz
NUCLEO-L053R8						

Дадим краткий обзор схемы тактирования STM32. Благодаря CubeMX можно абстрагироваться от конкретной реализации схемы тактирования, кроме случаев, когда необходимо иметь дело с особыми конфигурациями PLL, оптимальными по производительности и энергопотреблению.

На рисунке 3 показана схема тактирования одного из самых простых микроконтроллеров STM32: STM32F030R8. Она взята из соответствующего справочного руководства, предоставляемого ST. Красным цветом обозначен наиболее значимый путь: путь от HSI-генератора к ядру Cortex-M0, шине АHB и DMA. Представим наиболее важные элементы данного пути.

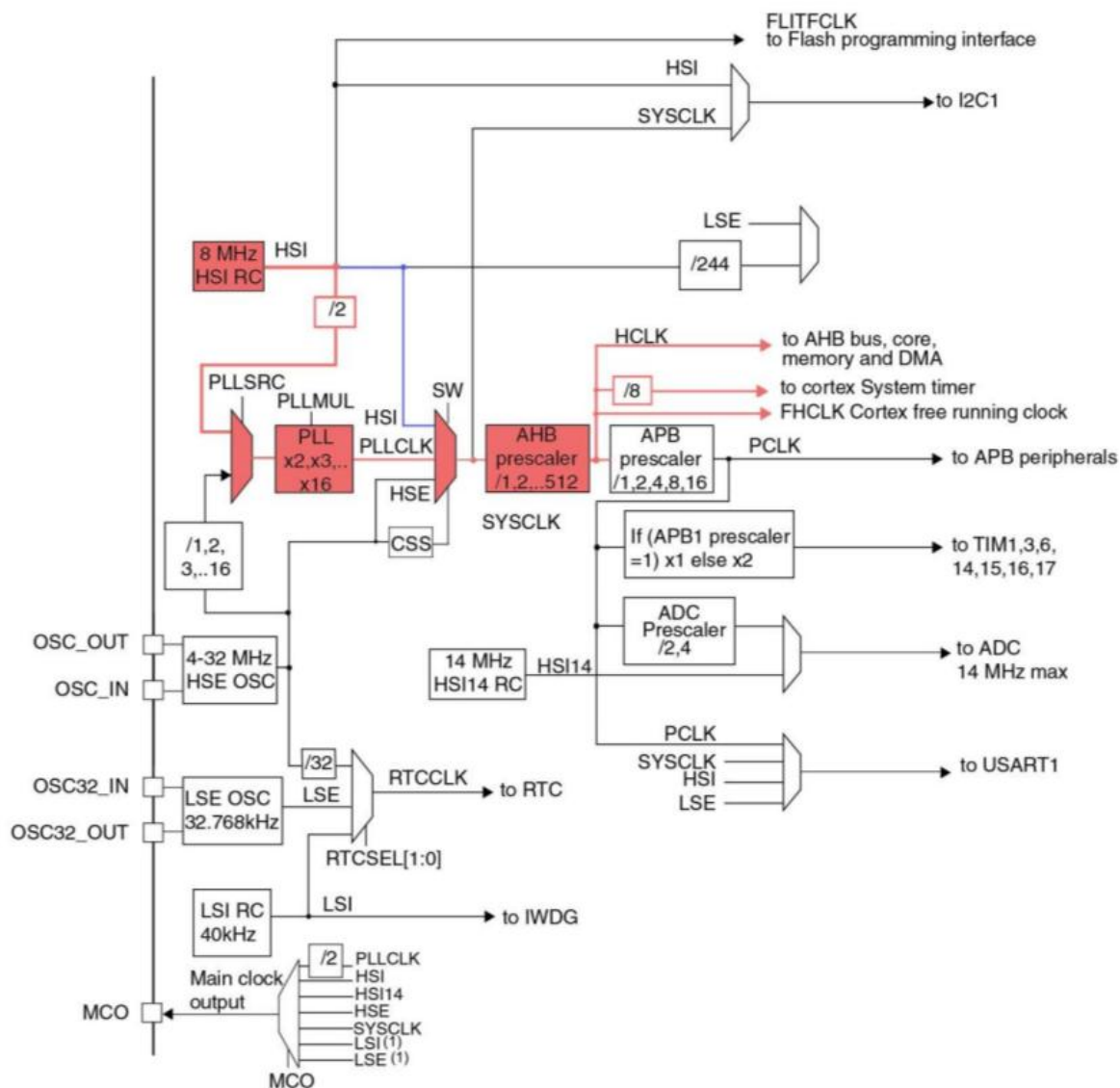


Рисунок 3 Схема тактирования микроконтроллера STM32F030R8

Путь начинается от внутреннего генератора на 8 МГц. Как уже было сказано, это RC-генератор, откалиброванный на заводе ST с точностью до 1% при температуре окружающей среды в 25°C. Затем тактовый сигнал HSI-генератора может быть направлен к переключателю системного тактового сигнала (SW) в том виде, в каком он на выходе генератора (путь выделен синим цветом на рисунке 3), или его можно направить к блоку умножителя PLL после его деления на 2 промежуточным делителем. Блок основного PLL может умножать тактовый сигнал в 4 МГц на значение вплоть до 12 для получения максимальной системной тактовой частоты (System Clock Frequency, SYSCLK), равной 48 МГц. Источник тактового сигнала SYSCLK может использоваться для питания периферийного устройства I2C1 (в качестве альтернативы HSI-генератору) и другого промежуточного делителя – делителя шины AHB, который может использоваться для понижения

высокочастотного тактового сигнала (High (speed) Clock, HCLK), в свою очередь, подводимого к шине АНВ, ядру и системному таймеру.

Конфигурация схемы тактирования выполняется специальным периферийным устройством, называемым Системой сброса и тактирования (Reset and Clock Control, RCC), и этот процесс в основном состоит из трех этапов:

1. Выбирается источник высокочастотного генератора (HSI или HSE) и конфигурируется должным образом, если используется HSE.

2. Если мы хотим подать сигнал SYSCLK с частотой, превышающей частоту, обеспечиваемую высокочастотным генератором, нам необходимо сконфигурировать блок основного PLL (который обеспечивает сигнал PLLCLK). В противном случае мы можем пропустить этот шаг.

3. Конфигурируется переключатель системного тактового сигнала (SW) выбором правильного источника тактового сигнала (HSI, HSE или PLLCLK). Затем выбираются правильные настройки предделителей шин АНВ, APB1 и APB2 (если доступна) для достижения требуемой частоты высокочастотного сигнала (HCLK – сигнал, который подается на ядро, DMA и шину АНВ) и частот продвинутых периферийных шин APB1 и APB2 (если есть).

Знание допустимых значений для PLL и предделителей может быть кошмаром, особенно для более сложных микроконтроллеров STM32. Только некоторые комбинации действительны для используемого микроконтроллера STM32, и их неправильная конфигурация может потенциально повредить микроконтроллер или, по крайней мере, привести к отказам (неправильная конфигурация тактирования может привести к ненормальному поведению, странным и непредсказуемым сбросам и тому подобное). К счастью для нас, инженеры STM32 предоставили отличный инструмент для упрощения конфигурации тактирования: CubeMX.

### **Разрешение Выхода синхронизации**

В зависимости от используемого корпуса ИС микроконтроллеры STM32 позволяют направлять тактовый сигнал на один или два выходных I/O, называемых выходами синхронизации (Master Clock Output, MCO). Это выполняется с помощью функции:

```
void HAL_RCC_MCOConfig(uint32_t RCC_MCOx, uint32_t RCC_MCOSource,
uint32_t RCC_MCODiv);
```

Например, чтобы направить PLLCLK к выводу MCO1 в микроконтроллере STM32F401RE (который соответствует выводу PA8), мы должны вызвать вышеупомянутую функцию следующим образом:

```
        HAL_RCC_MCOConfig(RCC_MCO1,          RCC_MCO1SOURCE_PLLCLK,  
RCC_MCODIV_1);
```

## **Лекция на тему «Режимы потребления МК»**

### **Рассматриваемые вопросы:**

1. Спящие режимы в микроконтроллерах на базе Cortex-M.
2. «Спящий режим по выходу».
3. Переход в/выход из спящих режимов.
4. Как микроконтроллеры Cortex-M управляют рабочим и спящим режимами.
5. Управление питанием в микроконтроллерах на базе Cortex-M.
6. Управление питанием.

### **Теоретический материал:**

#### **Управление питанием**

Энергоэффективность является одной из основных тем в микроэлектронной промышленности. Хорошо спроектированное устройство, с точки зрения энергоэффективности, не только потребляет меньше энергии, но и также позволяет упростить и минимизировать его схему питания, уменьшая габаритные размеры печатной платы, спецификацию компонентов и рассеиваемую энергию.

Часто мы думаем, что управление питанием электронной платы завязано лишь на схеме ее питания. В последние два десятилетия преобразование энергии стало темой горячего обсуждения. В результате исследований и разработок, проведенных производителями интегральных схем, было создано множество интегральных устройств, способных повысить общую эффективность энергопотребления во многих сферах применения, начиная от решений с пониженным энергопотреблением и заканчивая высоконагруженными преобразователями, способными выдавать тысячи ампер.

Современные микроконтроллеры предоставляют разработчикам множество инструментов для минимизации энергопотребления. Ядра Cortex-M не являются исключением: они предоставляют «абстрактную» модель управления питанием, которая может быть переработана производителями интегральных схем для создания собственной схемы питания. Это в точности относится к микроконтроллерам STM32: несмотря на то, что управление питанием присуще всем сериям STM32, оно достигает очень сложной реализации в семействах STM32L, которые предоставляют разработчикам масштабируемую модель питания для точной настройки необходимого энергопотребления. Все это позволяет создавать электронные устройства, способные



работать в течение многих лет даже при питании от батарейки пуговичного типоразмера.

### **Управление питанием в микроконтроллерах на базе Cortex-M**

Прежде чем мы изучим возможности программного выбора режима питания микроконтроллера, предоставляемые микроконтроллерами на базе Cortex-M, неплохо бы сделать некоторые соображения по поводу источников энергопотребления в цифровом устройстве.

Прежде всего, сложность самого устройства влияет на потребление энергии. Чем больше периферийных устройств и возможностей у нашей платы, тем больше требуется энергии для ее питания. Более того, некоторые периферийные устройства являются энергоемкими. Например, TFT-дисплеи потребляют значительно больше энергии по сравнению с другими компонентами на электронной плате. В конце концов, проектирование устройств с пониженным энергопотреблением требует тщательного выбора всех компонентов для разрабатываемого устройства. Например, в приложениях, где часы реального времени (RTC) поддерживаются активными при всех условиях, включая спящий режим, режим выключенного состояния и режим VBAT, потребление тока LSE-генератором становится более критичным на фоне общего потребления энергии устройством.

Сосредоточив наше внимание исключительно на микроконтроллере, первым аспектом, влияющим на потребление энергии, является его рабочая частота: чем быстрее работает процессор, тем больше он потребляет энергии. И это закон, высеченный на камне, который должны знать все разработчики микропрограммы: даже если используемый нами микроконтроллер способен работать на частоте до 200 МГц, и если у нас нет потребности во всей этой скорости, то мы можем сэкономить достаточно много энергии, просто уменьшив тактовую частоту. И это одна из основных причин, по которой микроконтроллеры STM32 имеют сложную схему распределения тактирования.

Еще одним следствием данного аспекта является то, что чем больше активно периферийных устройств, тем больше энергии потребляет микроконтроллер. Это означает, что правильно разработанная микропрограмма всегда немедленно отключает периферийное устройство, которое становится ненужным. Например, если нам нужна память EEPROM на шине I<sup>2</sup>C только во время процесса начальной загрузки (поскольку она хранит некоторые параметры конфигурации, которые мы загружаем в ОЗУ на время жизненного цикла микропрограммы), то мы должны отключить периферийное устройство I<sup>2</sup>C после ее завершения. По этой причине микроконтроллеры STM32

предоставляют возможность выборочного отключения каждого периферийного устройства, запрещая тактирование его источника тактового сигнала, вызывая макрос `__HAL_RCC_CLK_DISABLE()`, где – это конкретное периферийное устройство (например, `__HAL_RCC_DMA1_CLK_DISABLE()` позволяет запретить тактирование DMA1, в то время как `__HAL_RCC_DMA1_CLK_ENABLE()` разрешает его).

Говоря о микроконтроллерах, лучше всего говорить об их энергоэффективности, а не только об их энергопотреблении. В то время как энергопотребление устройства говорит нам только о том, сколько мА или мкА потребляет устройство, энергоэффективность измеряет, «сколько работы» оно может выполнить с ограниченным запасом энергии, например, в форме DMIPS/мВт или CoreMark/мВт. Таким образом, мы можем обнаружить, что для микроконтроллера STM32L4 наилучший энергетический компромисс достигается при его работе в рабочем режиме с пониженным энергопотреблением (Low-Power RUN, LPRUN).

Наконец, сама конструкция микроконтроллера и его периферийных устройств влияет на общее энергопотребление. По этой причине микроконтроллеры STM32L специально разработаны для обеспечения лучшего в своем классе энергопотребления при одновременном обеспечении наилучших характеристик в соответствии с конкретным подсемейством. Например, некоторые интерфейсы передачи данных в микроконтроллере STM32L4 (LPUART является одним из них) позволяют обмениваться данными в режиме DMA, когда микроконтроллер находится в режиме останова STOP2.

### **Как микроконтроллеры Cortex-M управляют рабочим и спящим режимами**

Когда микроконтроллер на базе Cortex-M перезагружается, его режим питания устанавливается в рабочий. В данном режиме необходимое энергопотребление определяется всей конструкцией микроконтроллера, но в большей степени рабочей частотой и количеством активных периферийных устройств. Здесь важно отметить, что Flash-память и память SRAM также являются «периферийными устройствами», внешними по отношению к ядру Cortex-M. Более того, внедрение продвинутых технологий предварительной выборки Flash-памяти, таких как ускоритель ART™ Accelerator, также влияет на общее энергопотребление.

В этом режиме разработчик может изменить способ, которым микроконтроллер потребляет энергию, регулируя его тактовую частоту и отключая ненужные периферийные устройства. Это может показаться очевидным, но важно отметить, что это лучшая оптимизация энергопотребления, которую мы можем сделать

во многих реальных ситуациях. Как мы увидим далее в этой главе, микроконтроллеры STM32L структурируют рабочий режим в несколько подрежимов, предлагая больший контроль над энергопотреблением и гарантируя при этом большинство функциональных возможностей и наилучшую производительность ЦПУ.

Если мы знаем, что нам не нужно ничего обрабатывать в течение определенного периода времени, то ядра Cortex-M позволяют нам переводить их в спящий режим, не выполняя циклы активного ожидания (busy-waits). В данном режиме ядро останавливается и его можно пробудить только «внешними событиями», исходящими от контроллера EXTI (например, кнопкой, подключенной к GPIO). Опять же, микроконтроллеры STM32L расширяют этот режим, предлагая до восьми различных подрежимов, как мы увидим далее.

Важно подчеркнуть, что ядро Cortex-M переходит в спящий режим «на добровольной основе»: две различные инструкции ARM, которые мы увидим позже, приостанавливают ЦПУ, оставляя некоторые его линии событий активными. При срабатывании этих линий ЦПУ возобновляет выполнение в течение некоторого времени пробуждения, которое зависит от действующего уровня сна и типа ядра Cortex-M (M0, M3 и так далее).

Задержка при пробуждении может быть выражена в тактовых циклах ЦПУ для «облегченных» спящих режимов и в мкс для режимов глубокого сна (deep sleep modes). Это означает, что чем глубже спящий режим, тем дольше время пробуждения. Разработчики должны решить, какой спящий режим следует использовать для их конкретных приложений: потребляемая энергия или время, затрачиваемое на переход и выход из состояния глубокого пониженного энергопотребления, которое может перевесить любой потенциальный выигрыш в экономии энергии. В носимых устройствах энергоэффективность является наиболее предпочтительным фактором, в то время как в некоторых приложениях управления производственными процессами задержка при пробуждении может быть действительно критичной.

Существуют также разные подходы к проектированию систем с пониженным энергопотреблением. В настоящее время многие встраиваемые системы управляются посредством прерываний. Это означает, что система остается в спящем режиме при отсутствии запросов на обработку. Когда приходит запрос на прерывание, процессор пробуждается, обслуживает его и возвращается в спящий режим, когда работа завершена. В другом случае, если запрос на обработку данных является периодическим и имеет постоянную продолжительность, и, если задержка обработки данных не является проблемой, вы можете запустить систему на самой низкой возможной

тактовой частоте, чтобы уменьшить энергопотребление. Не существует четкого ответа на вопрос, какой подход лучше, поскольку выбор будет зависеть от требований приложения к обработке данных, используемого микроконтроллера и других факторов, таких как тип источника питания.

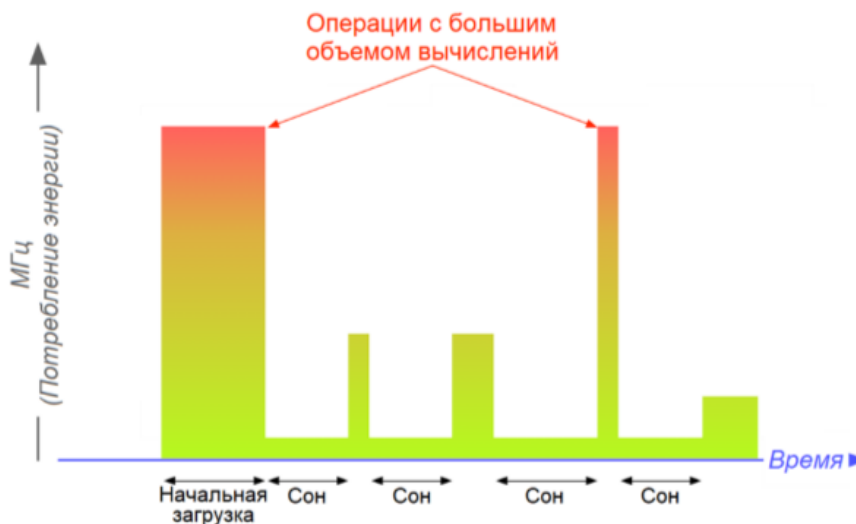


Рисунок 1: Как микропрограмма потенциально может управлять тактовой частотой и режимами питания во время своего выполнения

На рисунке 1 показана возможная стратегия минимизации энергопотребления. В процессе начальной загрузки микроконтроллера он работает на максимальной тактовой частоте, что позволяет быстро завершить все действия по инициализации. Когда все периферийные устройства сконфигурированы, тактовая частота снижается, и микроконтроллер переходит в спящий режим. В этот период микроконтроллер пробуждается прерываниями, которые могут обрабатываться при более низких тактовых частотах процессора. Когда требуется выполнение операций с большим объемом вычислений, тактовая частота может быть увеличена до максимума, а затем снова уменьшена после завершения работы.

Итак, когда же перейти в спящий режим? Как было сказано выше, мы должны выбрать подходящее время для перевода микроконтроллера в один из возможных спящих режимов. Если мы знаем, что микроконтроллер ожидает асинхронные события, за которыми следуют прерывания, то лучше перейти в спящий режим вместо выполнения циклов активного ожидания. Давайте рассмотрим классическое приложение мигания светодиодом.

```
... while(1)
{
```

```
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);  
    HAL_Delay(500);  
}
```

Этот с виду безобидный код оказывает существенное влияние на энергопотребление нашего устройства. Несмотря на то, что нам особо не нужно много чего делать в течение этих 500 мс, мы тратим много энергии на проверку значения глобального счетчика тиков таймера SysTick, чтобы посмотреть, прошло ли это время задержки. Вместо этого мы можем перестроить этот код, чтобы большую часть времени оставаться в спящем режиме, и мы можем сконфигурировать таймер, который будет пробуждать микроконтроллер каждые 100 мс.

### **Переход в/выход из спящих режимов**

Микроконтроллеры на базе Cortex-M предлагают две инструкции для их перевода в спящий режим: WFI и WFE. Инструкция «Ожидание прерывания» (Wait For Interrupt, WFI) также называется безусловной инструкцией перехода в спящий режим. Когда ЦПУ выполняет данную инструкцию, оно немедленно останавливает выполнение ядра. Процессор возобновит работу только по запросу прерывания, в зависимости от приоритета прерывания и действующего уровня сна, или в случае событий отладки. Если прерывание оказалось отложено (pending), пока микроконтроллер выполнял инструкцию WFI, он переходит в спящий режим и сразу же выходит из него.

«Ожидание события» (Wait For Event, WFE) является другой инструкцией, позволяющей перевести микроконтроллер в спящий режим. Она отличается от WFI тем, что проверяет состояние специального регистра событий, прежде чем остановить ядро: если этот регистр установлен, WFE сбрасывает его и не приостанавливает ЦПУ, продолжающее выполнение программы (это позволяет нам управлять отложенным событием, если это необходимо). В противном случае она приостанавливает микроконтроллер, пока данный регистр событий снова не будет установлен.

Но в чем именно разница между событием и прерыванием? События являются источником путаницы в мире STM32 (а также в мире Cortex-M в целом). Прежде чем мы выясним, что такое события, нам нужно лучше объяснить роль контроллера EXTI в микроконтроллере STM32.

Расширенный контроллер прерываний и событий (Extended Interrupts and Events Controller, EXTI) – это аппаратный компонент, встроенный в микроконтроллер, который управляет внешними и внутренними асинхронными прерываниями/событиями и генерирует запрос событий для ЦПУ/контроллера NVIC и запрос на пробуждение для

контроллера питания (смотри рисунок 2). Контроллер EXTI позволяет управлять несколькими линиями событий, которые могут пробудить микроконтроллер из некоторых спящих режимов (не все события могут пробудить микроконтроллер). Линии могут быть конфигурируемыми или прямыми и, следовательно, встроенными в микроконтроллер:

1. Линии являются конфигурируемыми: активный фронт может быть выбран независимо, а флаг состояния указывает источник прерывания. Конфигурируемые линии используются внешними прерываниями от вводов/выводов и несколькими периферийными устройствами.

2. Линии являются прямыми и аппаратными: они используются некоторыми периферийными устройствами для генерации пробуждения от события останова или прерывания. Флаг состояния предоставляется самим периферийным устройством. Например, RTC может использоваться для генерации события для пробуждения микроконтроллера.

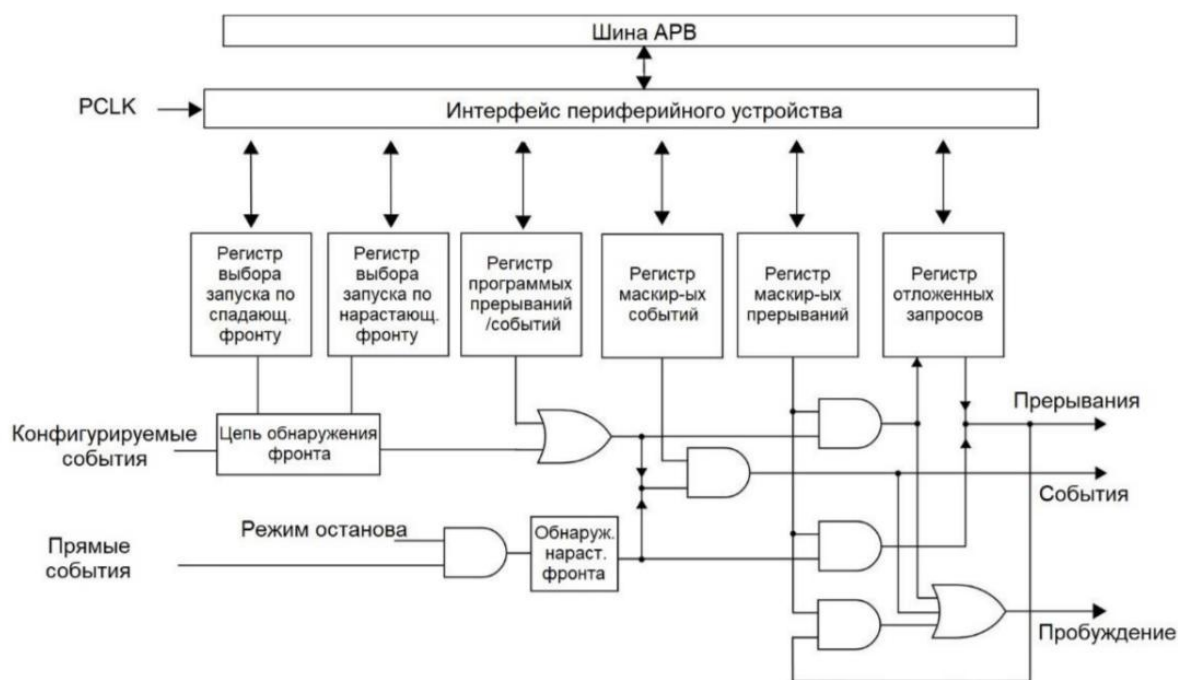


Рисунок 2 Как события могут быть использованы для пробуждения ядра

Другим важным аспектом, который необходимо прояснить в отношении контроллеров EXTI и NVIC, является то, что каждая линия может маскироваться независимо для генерации прерываний или событий. Например, GPIO можно сконфигурировать для работы в режиме `GPIO_MODE_EVT_*`, который отличается от режима `GPIO_MODE_IT_*`: в первом случае, когда срабатывает I/O, он не будет генерировать запрос IRQ, но установит флаг события. Это приведет к пробуждению

микроконтроллера, если он перешел в режим пониженного энергопотребления при помощи инструкции WFE.

Таким образом, инструкция WFE проверяет, нет ли отложенных событий, и по этой причине она также называется инструкцией условного перехода в спящий режим. Этот регистр событий может быть установлен:

- в случае перехода в и выхода из исключения (exception entrance and exit);
- когда включена функция SEV-On-Pend, регистр событий может быть установлен при изменении отложенного состояния прерывания с 0 на 1;
- если периферийное устройство устанавливает свою выделенную линию событий (это зависит от периферийного устройства);
- в случае выполнения инструкции SEV (Send Event – «Генерация события»);
- в случае события отладки (например, запрос на приостановку выполнения программы).

В ядрах Cortex-M3/4/7 мы можем временно маскировать выполнение этих прерываний, имеющих приоритет ниже значения, установленного в регистре BASEPRI. Однако данные прерывания все еще разрешены и помечаются как отложенные при их срабатывании. Мы можем сконфигурировать микроконтроллер для установки регистра событий в случае отложенных прерываний, установив бит SCR->SEVONPEND. Как следует из названия, этот регистр приведет к «установке регистра событий, если прерывания отложены». Это означает, что, если процессор был переведен в спящий режим по инструкции WFE, ЦПУ сразу же пробуждается, и мы можем в конечном итоге обрабатывать отложенные прерывания. Вместо этого WFI никогда не пробудит ядро. Cube HAL предоставляет две удобные функции, HAL\_PWR\_EnableSEVOnPend() и HAL\_PWR\_DisableSEVOnPend(), для выполнения этой настройки.

Если вместо этого прерывания маскируются установкой регистра PRIMASK, отложенное прерывание может пробудить процессор, независимо от используемой инструкции перехода в спящий режим (WFI или WFE): эта характеристика позволяет некоторым частям микроконтроллера отключаться программно путем запрета их тактирования, и программное обеспечение может обратно разрешить его после пробуждения перед выполнением ISR.

Итак, подведем итог. WFI и WFE ведут себя одинаково:

- пробуждают по запросам прерываний/исключений, которые разрешены и имеют более высокий приоритет, чем текущий уровень;
- микроконтроллер может быть пробужден событиями отладки;

- могут использоваться для перехода как в спящий режим, так и в режим глубокого сна.

Вместо этого WFI и WFE отличаются по следующим причинам:

- выполнение WFE не переводит микроконтроллер в спящий режим, если установлен внутренний регистр событий, в то время как выполнение WFI всегда приводит к спящему режиму;
- новое отложенное состояние запрещенного или маскированного прерывания может пробудить процессор от WFE, если установлен регистр SEVONPEND;
- выполнив WFE, микроконтроллер может быть пробужден внешним событием;
- выполнив WFI, микроконтроллер может быть пробужден разрешенным прерыванием, когда установлен регистр PRIMASK.

#### **«Спящий режим по выходу»**

Функция «Спящий режим по выходу» (Sleep-On-Exit) полезна для приложений, построенных на прерываниях, где все операции (кроме этапа инициализации) выполняются внутри обработчиков прерываний. Это программируемая функция, которую можно включить или отключить, установив бит регистра SCB->SCR. Когда она включена, ядро Cortex-M автоматически переходит в спящий режим (так же, как и при выполнении инструкции WFI) при выходе из обработчика исключения/прерывания. Функция Sleep-OnExit должна быть включена в конце этапа инициализации. В противном случае, если событие прерывания произойдет на этапе инициализации, когда функция Sleep-On-Exit уже включена, процессор перейдет в спящий режим, несмотря на то что этап инициализации еще не завершен.

CubeHAL предоставляет две удобные процедуры для включения/отключения этого режима: HAL\_PWR\_EnableSleepOnExit() и HAL\_PWR\_DisableSleepOnExit().

#### **Спящие режимы в микроконтроллерах на базе Cortex-M**

Микроконтроллеры на базе Cortex-M архитектурно поддерживают два спящих режима: нормальный спящий режим и глубокий сон. Микроконтроллеры STM32F называют их Спящим (sleep) режимом и режимом Остановка (stop) и добавляют третий, еще более глубокий режим, называемый режимом Ожидания (standby). Серия STM32L дополнительно расширяет эти два «основных» режима работы на несколько подрежимов.

И спящий режим, и режим глубокого сна достигаются с помощью инструкций WFI и WFE, которые мы рассмотрели ранее. Единственное отличие состоит в том, что



режим глубокого сна достигается установкой бита SLEEPDEEP в 1 в регистре PWR->SCR. Однако нам не нужно разбираться в этих подробностях, поскольку CubeHAL спроектирован для абстрагирования от них.

Обычно микроконтроллеры STM32 спроектированы таким образом, что в спящем режиме отключается только тактирование ЦПУ, в то время как на другие тактовые генераторы или источники аналогового тактового сигнала он не оказывает влияния (это означает, что все включенные периферийные устройства остаются активными). Вместо этого в режиме останова тактирование всех периферийных устройств домена питания 1,8 В (или 1,2 В для новейших микроконтроллеров STM32), отключается, а тактирование домена питания VDD остается включенным, за исключением HSI-генератора и HSE-генератора, которые отключаются. В режиме ожидания и домен питания 1,8 В, и домен питания VDD отключены.

## СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Калабеков Б.А. Цифровые устройства и микропроцессорные системы. Москва: Горячая линия - Телеком, 2000
2. Кузин А.В. Микропроцессорная техника. Москва.: Издательский центр»Академия»,2004
3. Кузовкин, В. А. Электротехника и электроника : учебник для среднего профессионального образования / В. А. Кузовкин, В. В. Филатов. — Москва : Издательство Юрайт, 2025. — 416 с. — (Профессиональное образование). — ISBN 978-5-534-20474-2. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: <https://urait.ru/bcode/561194>
4. Лаврентьев Б.Ф. Схемотехника электронных средств. Москва.: Издательский центр»Академия»,2010
5. Нефедов С. В. Микропроцессорные системы: учебное издание / Нефедов С. В., Иванов В. Н. - Москва : Академия, 2023. - 336 с.
6. Огородников, И. Н. Микропроцессорная техника: введение в Cortex-M3 : учебное пособие для вузов / И. Н. Огородников. — Москва : Издательство Юрайт, 2024. — 116 с. — (Высшее образование). — ISBN 978-5-534-08420-7. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: <https://urait.ru/bcode/538954>
7. Сажнев, А. М. Микропроцессорные системы: цифровые устройства и микропроцессоры : учебник для среднего профессионального образования / А. М. Сажнев. — 3-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2025. — 148 с. — (Профессиональное образование). — ISBN 978-5-534-18601-7. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: <https://urait.ru/bcode/566725>
8. Технические средства автоматизации и управления : учебник для вузов / под общей редакцией О. С. Колосова. — Москва : Издательство Юрайт, 2025. — 331 с. — (Высшее образование). — ISBN 978-5-534-19350-3. — Текст : электронный // Образовательная платформа Юрайт [сайт]. с. 309 — URL: <https://urait.ru/bcode/560599>
9. Федорова Г.Н. Разработка модулей программного обеспечения для компьютерных систем: учебное издание / Федорова Г.Н. Москва : Академия, 2024. — 384с.