Микроконтроллеры: обзор и практика применения. Часть 2

Валерий Жижин

В данной статье рассматриваются типовые первоначальные конфигурации наиболее популярных семейств STM32F1xx и STM32F3xx. Раскрываются общее содержание и особенности использования системных библиотек и библиотек периферийных устройств. Приводятся примеры реализации проектов с использованием различных периферийных модулей, входящих в состав микроконтроллеров. Приводится также пример инсталляции и настройки среды разработки IAR Embedded Workbench for ARM 7.70.

Введение

Данная статья является продолжением темы, начатой в статье «Микроконтроллеры: обзор и практика применения. Часть 1», опубликованной в журнале «Современная электроника» № 8, 2025 г.

В предыдущей статье рассматривались существующие типы архитектур современных микроконтроллеров и области их применения. Акцент был сделан на архитектуре 32-разрядных устройств на основе ARM-ядер Cortex-M как наиболее продвинутых на сегодняшний день.

В данной статье рассматривается типовая первоначальная конфигурация наиболее популярных семейств STM32F1xx и STM32F3xx. Раскрываются общее содержание и особенности использования системных библиотек и библиотек периферийных устройств. Приводятся примеры реализации проектов с использованием различных периферийных модулей, входящих в состав микроконтроллеров. Коды проектов написаны на языке Си.

Приводится также пример инсталляции и настройки среды разработки IAR Embedded Workbench for ARM 7.70.

Настройка среды разработки

Для создания проектов с применением микроконтроллеров необходимо первоначально установить среду разработки программного обеспечения.

Существует ряд различных сред, поддерживающих проектирование ПО для микроконтроллеров ряда STM32. Их обзор был приведён в первой части статьи.

Автор предпочитает использование среды **IAR Embedded Workbench or ARM,** поскольку она отличается интуитивно понятным интерфейсом при инсталляции и в работе, а также высокой скоростью компиляции и качеством сгенерированного кода.

Среда IAR Embedded Workbench проприетарна и требует лицензирования, но только в случае, если код превышает 32K. В этот программный продукт входит компилятор Си и C++. После установки IAR потребуется зарегистрировать его.

С сайта компании необходимо скачать архивы STM32F10x standard peripherals library и STM32F30x standard peripherals library для работы с МК ряда STM32F10x и STM32F30x соответственно.

В них содержатся:

- CMSIS – библиотека, определяющая работу ядра M3 или M4 соответственно;
- SPL – стандартная библиотека периферийных устройств, удобная для их конфигурации. Библиотека SPL написана на языке Си и организована в виде структур для каждого типа периферийного устройства;
- Examples – очень полезное приложение, содержащее программный код для типовой, наиболее распространённой периферии.

Библиотека CMSIS предоставляет последовательные и простые интерфейсы для работы с ядром, его периферией и операционной системой реального времени RTOS.

Главным компонентом CMSIS является компонент CMSIS-CORE. Он предоставляет стандартизированный интерфейс для ядер Cortex-M0, Cortex-M3, Cortex-M4, SC000, SC300. Важнейшими файлами CMSIS-CORE являются следующие.

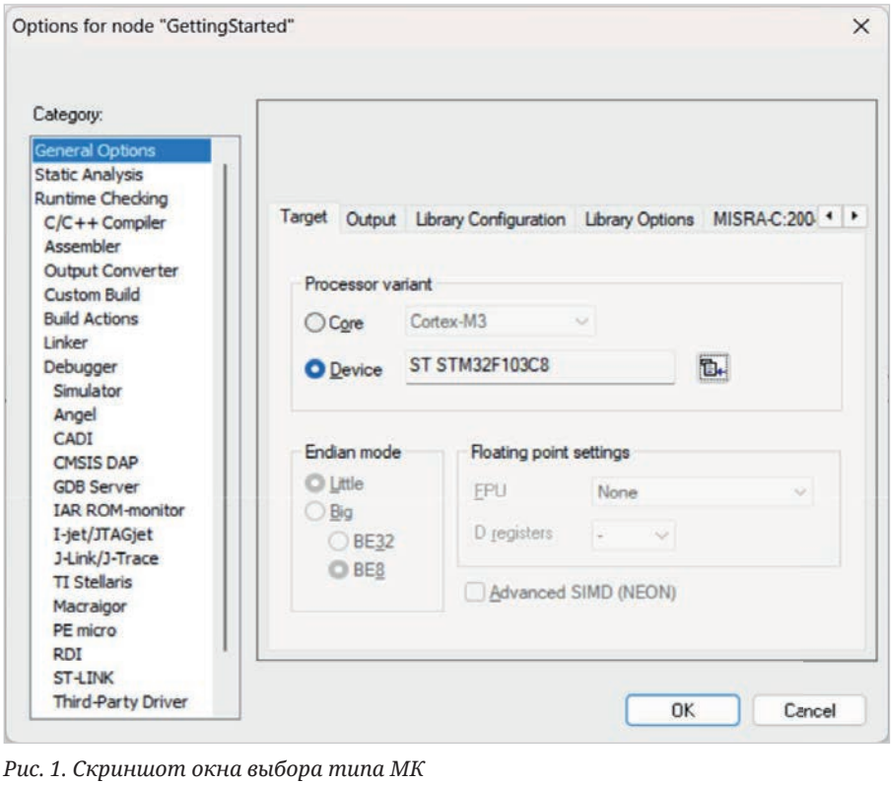


Рис. 1. Скриншот окна выбора типа МК

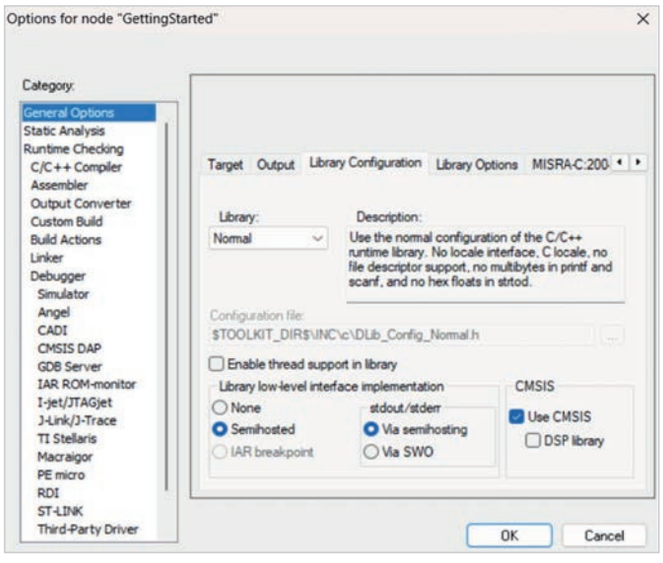


Рис. 2. Скриншот использования CMSIS

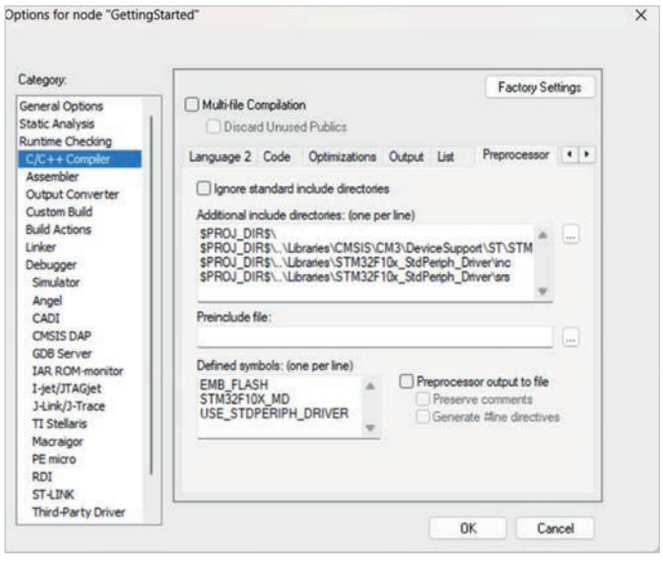


Рис. 3. Скриншот пути к папке с проектом и библиотекам

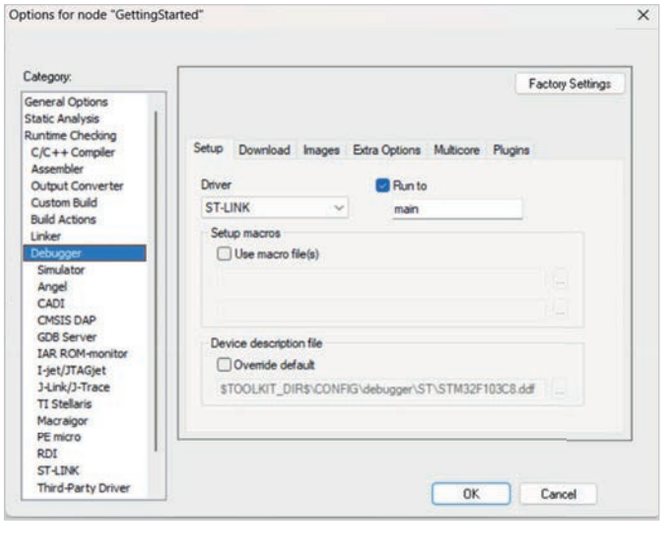


Рис. 4. Скриншот выбора отладчика

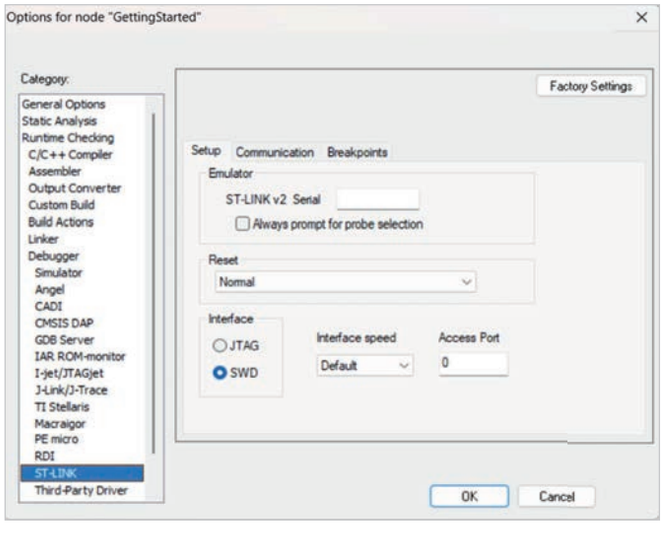


Рис. 5. Скриншот настройки ST-Link

- Файл startup_<device>.s, где вместо <device> подставляется идентификатор фирмы-производителя, например, stm32f1xx. Этот файл содержит обработчик (handler), который выполняется после сброса МК и вызывает функцию SystemInit(), векторы исключений и таблицу векторов прерываний.
- Файл <device>_conf.h. В нём хранится конфигурация периферийных устройств в закомментированном виде. Применяемые в создаваемом проекте периферийные модули должны быть раскомментированы.
- Файлы system_<device>.c и system_<device>.h содержат минимальные наборы функций для конфигурации системы тактирования:
  - void SystemCoreClockUpdate(void) – обновление переменной SystemCoreClock. Функция должна быть вызвана каждый раз, когда тактовая частота меняется во время работы;
  - void SystemInit(void) – инициализирует систему тактирования МК, в частности, синтезатор PLL. Содержит переменную, хранящую тактовую частоту.
- Заголовочный файл микроконтроллера <device>.h. Он содержит директивы препроцессора, перечисления для работы с периферией и прерываниями.

Для работы ядра микроконтроллера потребуется файл core_<cpu>.h (для stm32f1xx с ядром Cortex-M3 это файл **core_cm3.h**, для stm32f3xx с ядром Cortex-M4 это **core_cm4.h**). Они предоставляют доступ к регистрам ядра и периферии.

Для хранения файлов библиотек CMSIS, SPL, Examples нужно в основной папке, где будут находиться проекты с использованием микроконтроллеров ряда STM32, создать папку Libraries.

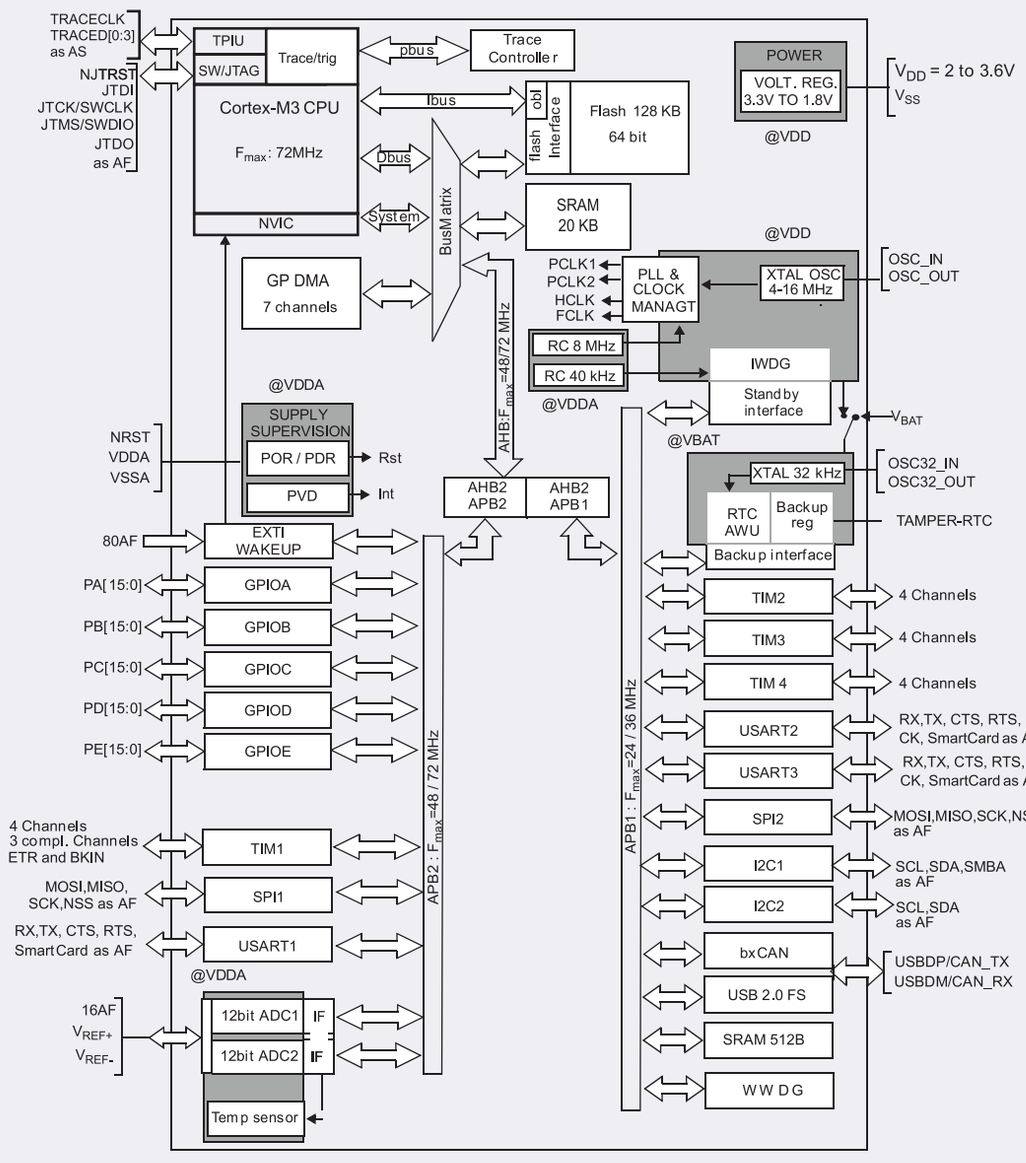
Запустив IAR, нужно создать рабочее пространство: File > New > Workspace. Далее создаём новый проект: Project > Creat > New > Project.

Программу можно писать на ассемблере, Си или C++. В приведённых ниже примерах будем использовать язык программирования Си.

Созданный проект необходимо настроить, для этого нужно открыть меню Project > Options. Во вкладке General Options выбираем целевой микроконтроллер, с каким будем работать, например, STM32F103C8T6 или STM32F303VT6 (рис. 1).

Далее во вкладке Library Coniguration нужно поставить галочку Use CMSIS (рис. 2).

Затем нужно добавить путь, по которому будут храниться файлы библиотеки Libraries с расширениями *.c и



1. T_A = –40°C to +105°C (junction temperature up to 125°C).
2. AF = alternate function on I/O port pin.

Рис. 6. Блок-схема микроконтроллера STM32F103

*.h. Для этого нужно во вкладке C/C++ Compiler > Preprocessor добавить &PROJDIR& (переменная, содержащая путь к папке с проектом), и далее согласно показанному на скриншоте (рис. 3).

Следующим шагом прописываем путь Project > Add Files > Libraries и добавляем в проект файлы из библиотеки SPL с расширением *.c для используемых в текущем проекте периферийных устройств. Они появятся в окне Workspace.

В разделе Debugger нужно выбрать в качестве средств отладки ST-Link. Проставить галочки Use flash loader и Verify во вкладке Download (рис. 4).

Во вкладке ST-Link нужно выбрать интерфейс SWD (рис. 5). Среда разработки готова к использованию.

Структура программного кода

Структура программы приложения для микроконтроллера должна обязательно включать в себя следующие элементы:

- директивы препроцессора;
- объявление структур для всех периферийных устройств;
- объявление глобальных переменных;
- прототипы используемых функций;
- функции конфигурации для используемых устройств;
- функцию main().

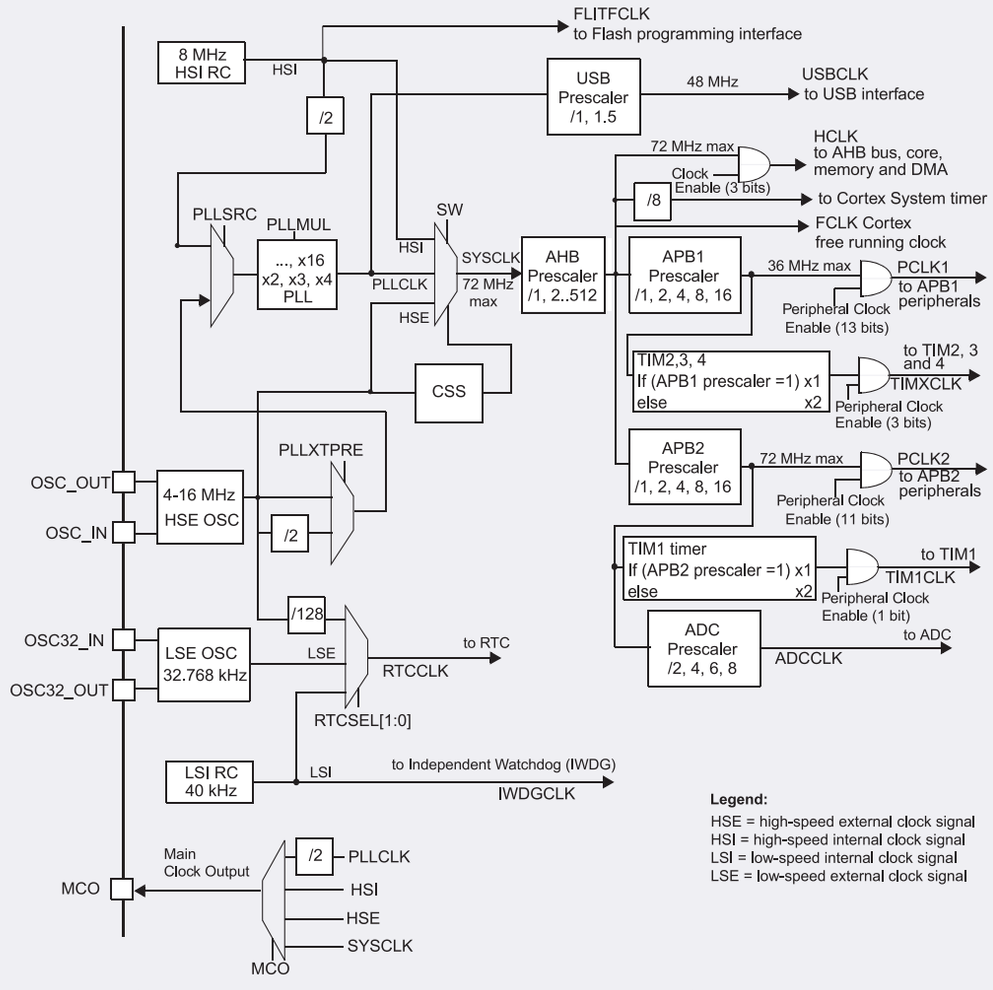
Примеры настройки и работы с периферийными устройствами

Блок-схема микроконтроллера на примере ряда STM32F103 показана на рис. 6 [1].

Она отражает состав микроконтроллера: ЦПУ, контроллер DMA, периферийные устройства, внутренние шины передачи данных и тактирования.

Прежде всего нужно разобраться с тактированием CPU и периферийных устройств.

В качестве источника тактовой частоты SYSCLK микроконтроллеров STM32F могут быть использованы (Reference Manual [2]):



1. When the HSI is used as a PLL clock input, the maximum system clock frequency that can be achieved is 64 MHz.
2. For the availability of the USB function both HSE and PLL must be enabled, with USBCLK running at 48 MHz.
3. To have an ADC conversion time of 1 µs, APB2 must be at 14 MHz, 28 MHz, or 56 MHz.

Рис. 7. Система тактирования микроконтроллера

- внутренний RC-генератор HSI с частотой 8 МГц;
- генератор HSE с внешним кварцевым резонатором или источником тактовых импульсов на жёсткой логике;
- PLL-умножитель частоты HSI или HSE на основе генератора с ФАПЧ.

Система тактирования микроконтроллера показана на рис. 7.

Программно сконфигурировать систему тактирования ЦПУ микроконтроллера можно двумя способами: в функции main() вызвать функцию **void SystemInit(void)** или написать свою функцию, например, **void Clk_Init(void)**, и также вызвать её в функции main().

В первом варианте частота тактирования будет составлять по умолчанию 72 МГц. Во втором будет определяться параметрами созданной функции.

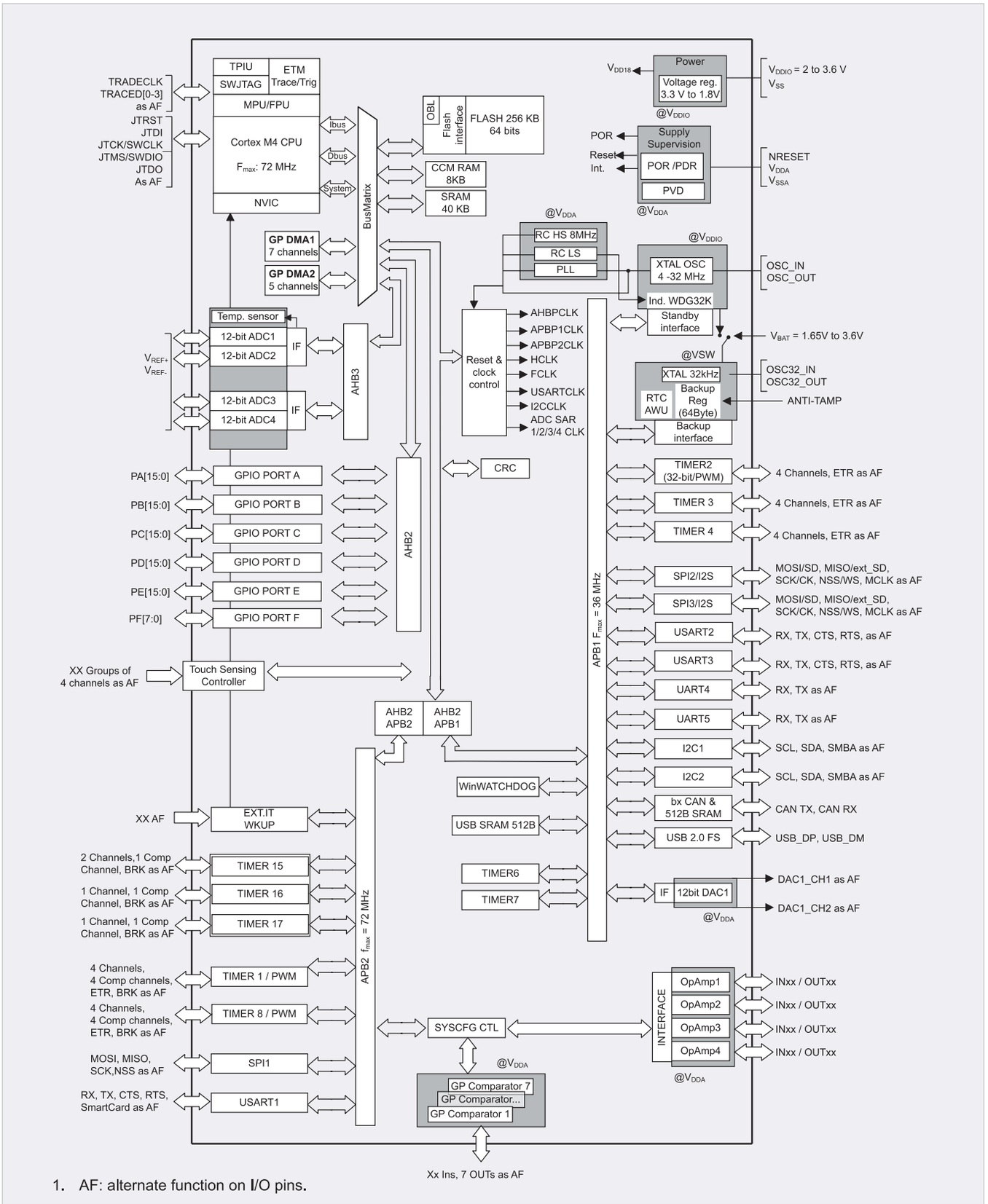
Предпочтительнее использовать второй способ, позволяющий более гибко управлять частотой тактирования в процессе выполнения программы, особенно в приложениях с пониженным энергопотреблением, когда не требуется высокая частота 72 МГц.

Пример функции тактового генератора на 36 МГц.

```
void Clk_Init(void)
{
 // Включение генератора HSI.
 RCC_HSICmd(ENABLE);
 // Ожидание установления внутреннего генератора HSI.
 while(RCC_GetFlagStatus(RCC_
 FLAG_HSIRDY) == RESET);
 RCC_SYSCLKConfig(RCC_
 SYSCLKSource_HSI);
 // Включение генератора HSE с внешним кварцевым резонатором.
 RCC_HSEConfig(RCC_HSE_ON);
 // Ожидание установления HSE.
 while(RCC_GetFlagStatus(RCC_
 FLAG_HSERDY) == RESET);
 // Инициализация PLL.
 PLLConfig(RCC_PLLSource_HSE_
 Div2,RCC_PLLMul_9);
 // 36MHz.
 RCC_PLLCmd(ENABLE);
 while(RCC_GetFlagStatus(RCC_
 FLAG_PLLRDY) == RESET);
 // Конфигурация внутренних шин тактирования.
 RCC_SYSCLKConfig(RCC_
 SYSCLKSource_PLLCLK);
 RCC_HCLKConfig(RCC_SYSCLK_
 Div1);//AHB
 RCC_PCLK1Config(RCC_HCLK_Div1);
 RCC_PCLK2Config(RCC_HCLK_Div1);
}
```

Работа с портами ввода-вывода GPIO

Средствами библиотеки STM32 Peripheral Library (SPL) можно легко управлять периферией контроллера без прямого обращения к регистрам.



1. AF: alternate function on I/O pins.

Рис. 8. Блок-схема микроконтроллера STM32F30x

Пример настройки порта ввода-вывода (GPIO).

Сначала на порт подаётся тактирование: **RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE).**

Периферийные устройства тактируются либо от шины APB2, либо от APB1. От какой именно, можно узнать по названию константы. В нашем случае это **RCC_APB2Periph_GPIOC**, поэтому шина – APB2. Константа расположена в файле stm32f10x_rcc.h.

Вот, что находится в stm32f10x_rcc.h.

```
#define RCC_APB2Periph_AFIO ((uint32_t)0x00000001)
#define RCC_APB2Periph_GPIOA ((uint32_t)0x00000004)
#define RCC_APB2Periph_GPIOB ((uint32_t)0x00000008)
#define RCC_APB2Periph_GPIOC ((uint32_t)0x00000010)
```

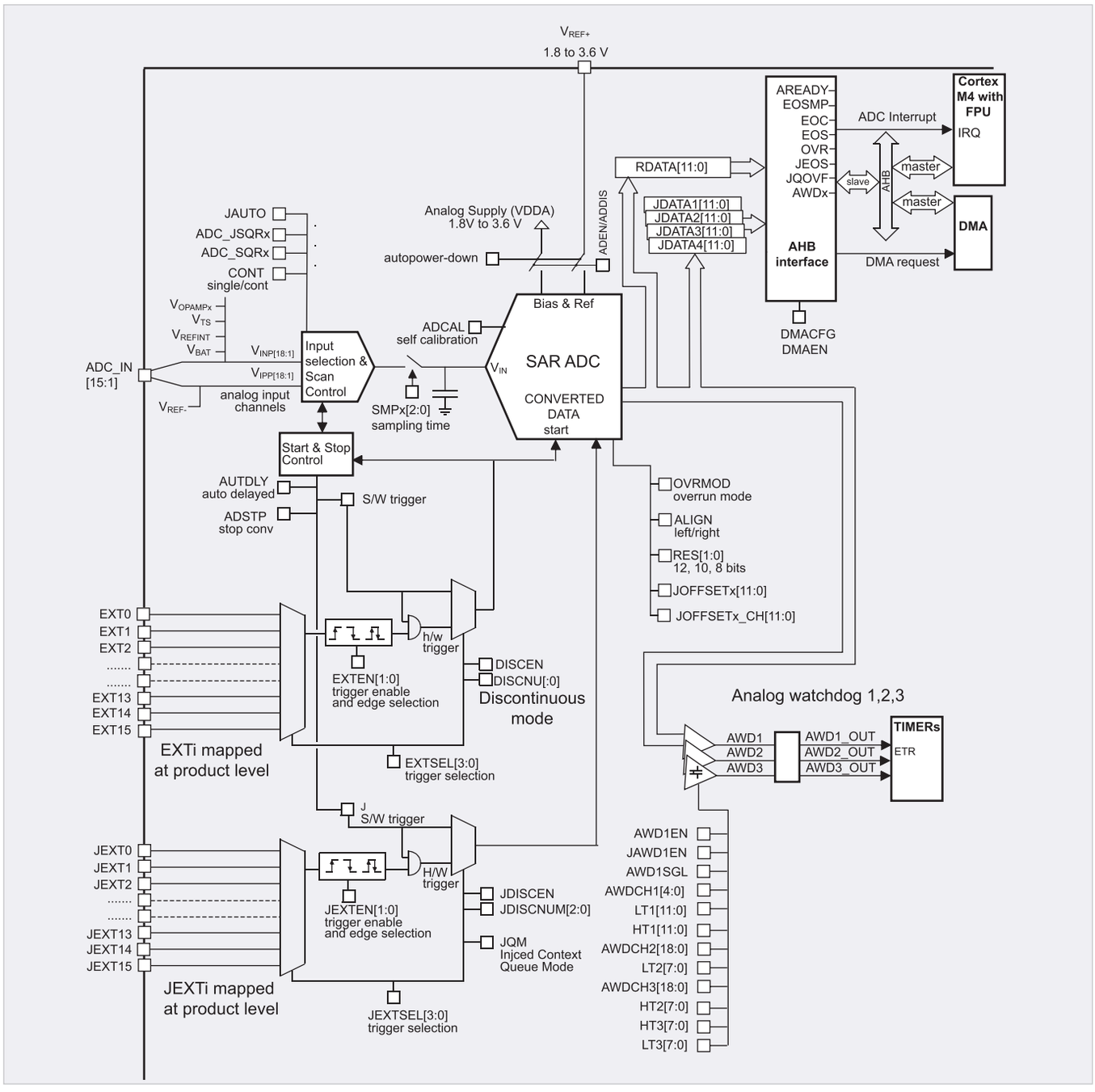
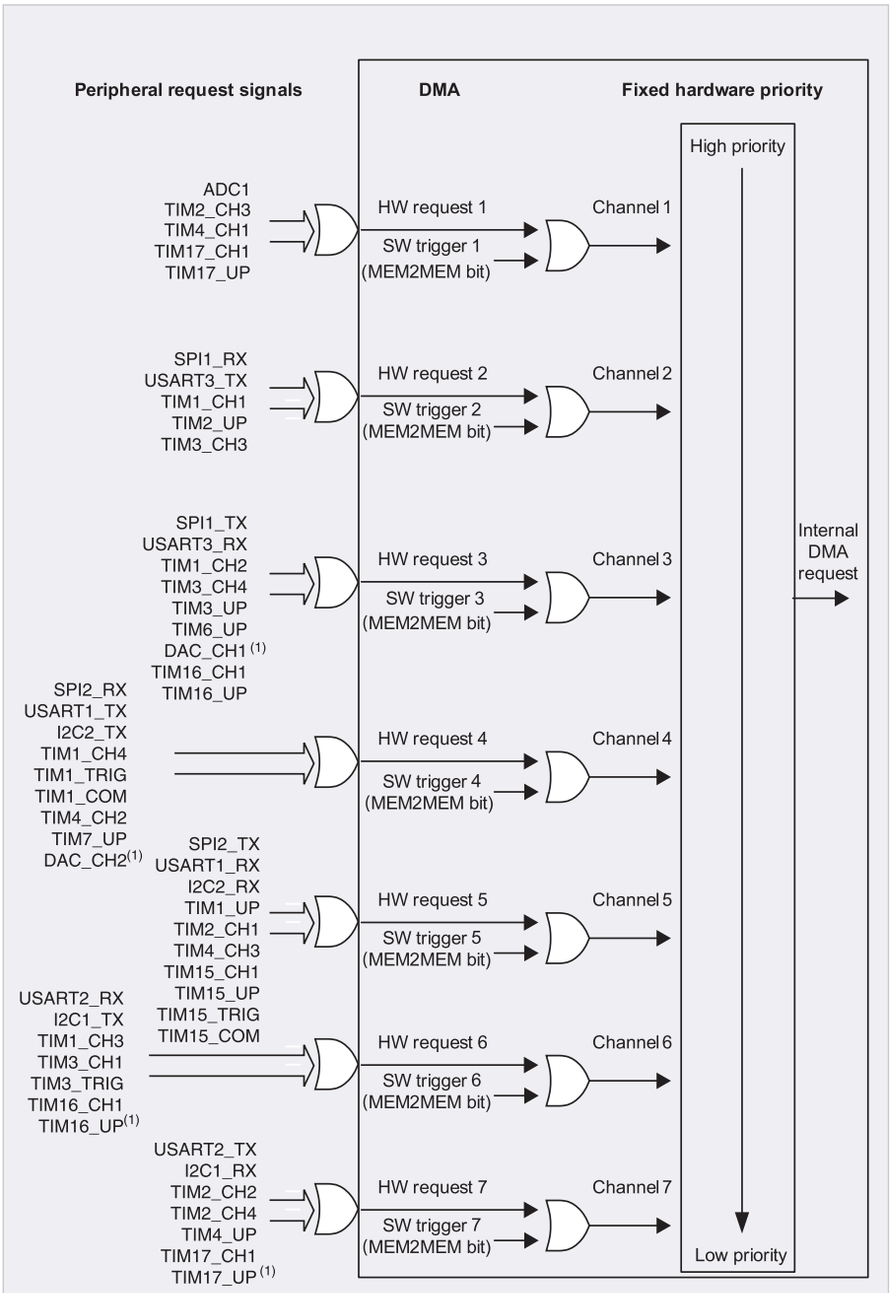


Рис. 9. Структурная схема модуля АЦП

#define RCC_APB2Periph_GPIOD ((uint32_t)0x00000020)
#define RCC_APB2Periph_GPIOE ((uint32_t)0x00000040)
#define RCC_APB2Periph_GPIOF ((uint32_t)0x00000080)
#define RCC_APB2Periph_GPIOG ((uint32_t)0x00000100)
#define RCC_APB2Periph_ADC1 ((uint32_t)0x00000200)
#define RCC_APB2Periph_ADC2 ((uint32_t)0x00000400)
#define RCC_APB2Periph_TIM1 ((uint32_t)0x00000800)
#define RCC_APB2Periph_SPI1 ((uint32_t)0x00001000)
#define RCC_APB2Periph_TIM8 ((uint32_t)0x00002000)
#define RCC_APB2Periph_USART1 ((uint32_t)0x00004000)
#define RCC_APB2Periph_ADC3 ((uint32_t)0x00008000)
#define RCC_APB2Periph_TIM15 ((uint32_t)0x00010000)
#define RCC_APB2Periph_TIM16 ((uint32_t)0x00020000)
#define RCC_APB2Periph_TIM17 ((uint32_t)0x00040000)

Здесь можно найти все необходимые константы для запуска тактирования любого периферийного устройства. Например, если мы хотим воспользоваться USART2, мы воспользуемся константой RCC_APB1Periph_USART2, а поскольку из её названия видно, что USART2 тактируется от шины APB1, включение тактирования мы произведём следующим образом: RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE). Одним вызовом функции можно включить тактирование сразу нескольких устройств, задав в параметре функции несколько констант через оператор побитовое ИЛИ: RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Perip | RCC_APB2Periph_ADC1, ENABLE).

Побитовое ИЛИ «соберёт» все единички из перечисленных констант, а сами константы являются маской,



1. DMA requests are mapped on this DMA channel only if the corresponding remapping bit is set in the SYSCFG_CFGR1 register. For more details, please refer to *Section 10.2: SYSCFG registers on page 162.*

Рис. 10. Структура DMA

предназначенной для такого использования.

Далее нужно создать функцию конфигурации используемого порта. Делается это заполнением полей соответствующей структуры, хранящейся в файле stm32f10x_gpio.c.

Пример функции:

```
void GPIO_Configuration(void)
{
 // Объявление структуры (InitStructure), которая содержит все параметры для настройки периферийного устройства в виде переменных – членов структуры.
 GPIO_InitTypeDef GPIO_InitStructure;
 RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOC, ENABLE);
 GPIO_DeInit(GPIOB);
 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_12 | GPIO_Pin_13;
 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
 // Двоичный выход.
 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
 // Ограничение быстродействия до 50 МГц.
 GPIO_Init(GPIOB, &GPIO_InitStructure);
 GPIO_DeInit(GPIOC);
 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
 // Двоичный выход.
 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
 // Ограничение быстродействия до 50 МГц.
 // Вызываем функцию инициализации, куда передаём указатель на сформированную структуру:
 GPIO_Init(GPIOC, &GPIO_InitStructure);
}
```

Все возможные значения для параметра GPIO_Mode содержатся в функции stm32f10x_gpio.h:

- GPIO_Mode_AIN – аналоговый вход;
- GPIO_Mode_IPD – вход с подтяжкой к земле (англ. Pull-down);
- GPIO_Mode_IN_FLOATING – вход без подтяжки (англ. Float);
- GPIO_Mode_IPU – вход с подтяжкой к питанию (англ. Pull-up);
- GPIO_Mode_Out_OD – выход с открытым стоком (англ. Open Drain);
- GPIO_Mode_Out_PP – выход двумя состояниями (англ. Push-Pull);
- GPIO_Mode_AF_OD – выход с открытым стоком для альтернативных функций (англ. Alternate Function). Используется в случаях, когда выводом должна управлять периферия, прикреплённая к данному разряду порта (например, вывод Tx USART и т.п.);
- GPIO_Mode_AF_PP – то же самое, но с двумя состояниями.

Для большинства устройств также требуется вызов команды «включение». Пример для включения USART1 и ADC: USART_Cmd(USART1, ENABLE), ADC_Cmd(ADC1, ENABLE).

Конфигурация портов GPIO завершена. При выполнении кода в функции main() должна быть обязательно вызвана функция **GPIO_Configuration();**

Работа с прерываниями

Вначале надо настроить и проинициализировать контроллер прерываний (NVIC – Nested Vectored Interrupt Controller). В архитектуре Cortex M3 каждому прерыванию можно выставить свой приоритет для случаев, когда возникает несколько прерываний одновременно. Поэтому NVIC представляет несколько вариантов формирования приоритетных групп [2]. При-

мер выбора варианта приоритетных групп состоит всего из одной команды: NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0). Эта команда выполняется один раз.

Далее для каждого прерывания надо произвести настройку и инициализацию с помощью структуры. Структура должна быть описана в соответствующей функции.

Вот, например, настройка прерывания для порта PC0.

```
void EXTI0_Config(void)
{
 // Включение тактирования порта.
 RCC_AHBPeriphClockCmd(RCC_
 AHBPeriph_GPIOC, ENABLE);
 // Настройка порта.
 GPIO_InitStructure.GPIO_Mode =
 GPIO_Mode_IN;
 GPIO_InitStructure.GPIO_PuPd =
 GPIO_PuPd_NOPULL;
 GPIO_InitStructure.GPIO_Pin =
 GPIO_Pin_0;
 GPIO_Init(GPIOC, &GPIO_
 InitStructure);
 RCC_APB2PeriphClockCmd(RCC_
 APB2Periph_SYSCFG, ENABLE);
 // Подключение линии прерывания
 к PC0.
 SYSCFG_EXTILineConfig(EXTI_
 PortSourceGPIOC, EXTI_
 PinSource0);
 // Конфигурация линии
 прерывания.
 EXTI_InitStructure.EXTI_Line =
 EXTI_Line0;
 EXTI_InitStructure.EXTI_Mode =
 EXTI_Mode_Interrupt;
 EXTI_InitStructure.EXTI_Trigger =
 EXTI_Trigger_Rising_Falling;
 EXTI_InitStructure.EXTI_LineCmd =
 ENABLE;
 EXTI_Init(&EXTI_
 InitStructure);кнл 0
 // Настройка контроллера
 прерываний.
 NVIC_InitStructure.NVIC_
 IRQChannel = EXTI0_IRQn;
 NVIC_InitStructure.NVIC_
 IRQChannelPreemptionPriority =
 0x02;
 NVIC_InitStructure.NVIC_
 IRQChannelSubPriority = 0x02;
 NVIC_InitStructure.NVIC_
 IRQChannelCmd = ENABLE;
 NVIC_Init(&NVIC_InitStructure);
}
```

В параметре NVIC_IRQChannel указываем, какое именно прерывание инициализируется. Константа **EXTI0_IRQn** обозначает канал, отвечающий за прерывания, связанные с GPIO (это канал «0»). Найдя её определение в файле stm32f10x.h, можно также увидеть ещё множество констант (ADC1_IRQn, TIM1_TRG_COM_TIM17_IRQn и др.), обозначающих прерывания от других периферийных устройств. Следующими двумя строками указывается приоритет прерываний (максимальные значения этих двух параметров определяются выбранной приоритетной группой). Последняя строка включает использование прерывания. Структура настроена, инициализируем её: NVIC_Init(&NVIC_InitStructure).

Внешние прерывания от портов GPIO могут быть настроены по переднему фронту **EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;**, заднему фронту **EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;**, одновременно по переднему и заднему фронтам **EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising_Falling;**.

При возникновении события прерывания код разработчика ПО выполняется в функции обработки прерывания.

Обработчиком прерывания является функция, название которой совпадает с названием соответствующего вектора прерывания в Startup-файле. Startup-файлы, входящие в состав STM32 Peripheral Library, написаны на ассемблере.

Вот фрагмент файла startup_stm32f10x_md_vl.s:

```
DCD SPI1_IRQHandler;
DCD SPI2_IRQHandler;
DCD USART1_IRQHandler;
DCD USART2_IRQHandler;
DCD EXTI0_IRQHandler;
```

Пример:

```
void EXTI0_IRQHandler(void)
{
 if (!(GPIOC->IDR & GPIO_Pin_0))
 // Если задний фронт.
 {
 TIM_Cmd(TIM3, ENABLE);
 // Запуск таймера.
 }
 if(GPIOC->IDR & GPIO_Pin_0)
 // Если передний фронт.
 {
 TIM_Cmd(TIM3, DISABLE);
 // Остановка таймера.
 }
 // Сброс флага прерывания.
 EXTI_ClearITPendingBit(EXTI_
 Line0);
}
```

В начало программы нужно добавить объявление структур:

```
EXTI_InitTypeDef EXTI_InitStructure;
NVIC_InitTypeDef NVIC_InitStructure;
```

В функции main() должна быть вызвана функция **__enable_irq()**, а включение прерываний производится функцией, которая находится в заголовочном файле периферийного устройства. Внешние прерывания включаются функцией **NVIC_EnableIRQ(EXTI0_IRQn);** (для 0 индекса порта).

Следующие примеры конфигурации периферийных устройств будут рассмотрены с использованием микроконтроллера ряда STM32F30x.

Блок-схема микроконтроллера STM32F30x показана на рис. 8. Взято из источника [3].

В отличие от STM32F10x, этот тип микроконтроллера имеет более развитую периферию: наличие встроенных аналоговых компараторов и операционных усилителей, что позволяет с минимальным количеством внешних компонентов создавать системы аналого-цифровой обработки сигналов в приложениях, не требующих высокой точности.

Использование таймеров

Таймер – наиболее часто используемое периферийное устройство в микроконтроллерах. Ему посвящено огромное количество примеров в Интернете, поэтому здесь будет приведён пример конкретной программы.

Возьмём один из базовых таймеров микроконтроллера STM32F3, например, таймер 2. Произведём его настройку и сгенерируем прерывания через равные промежутки времени.

Из библиотеки Standard Peripheral Library подключим несколько файлов, в которых реализовано взаимодействие с регистрами таймеров, и объявим соответствующую структуру:

```
#include "stm32f30x_gpio.h"
#include "stm32f30x_rcc.h"
#include "stm32f30x_tim.h"
#include "stm32f30x.h"
/***********************************/
TIM_TimeBase InitTypeDef timer;
/***********************************/
```

Настроим одну из ножек микроконтроллера на работу в режиме выхода. Это нужно, чтобы управлять светодиодом.

Минимальная инициализация таймера выглядит следующим образом.

```
/*************************************/
void Conig_Timer2(void)
{
 // Тактирование.
 RCC_AHBPeriphClockCmd(RCC_
 AHBPeriph_GPIOE, ENABLE);
 RCC_APB1PeriphClockCmd(RCC_
 APB1Periph_TIM2, ENABLE);
 // На этом выводе у нас синий свето-
 диод (STM32F3Discovery).
 Gpio.GPIO_Mode = GPIO_Mode_OUT;
 gpio.GPIO_Pin = GPIO_Pin_8;
 gpio.GPIO_Otype = GPIO_Otype_PP;
 gpio.GPIO_Speed = GPIO_
 Speed_50MHz;
 GPIO_Init(GPIOE, &gpio);
 // Настройка таймера TIM2.
 TIM_TimeBaseStructInit(&timer);
 timer.TIM_Prescaler = 7200;
 timer.TIM_Period = 20000;
 TIM_TimeBaseInit(TIM2, &timer);
 }
/*************************************/
```

В настройках присутствуют значения 7200 и 20 000. Таймер тактируется частотой 72 МГц. Prescaler, он же предделитель, нужен для деления тактовой частоты. Таким образом, получаем 72 МГц / 7200 = ***10 КГц***. Значит, один такт таймера соответствует (1/10 000) секунд, что равняется 100 микросекундам. Период таймера – это величина, досчитав до которой, программа перейдёт на обработчик прерывания по переполнению таймера. В нашем случае таймер дойдёт до 20 000, что соответствует (100×20 000) мкс или 2 секундам. То есть светодиод (который мы зажигаем и гасим в обработчике прерывания) будет мигать с периодом 4 секунды (2 секунды горит, 2 секунды не горит).

В функции main() вызываем функцию инициализации, а также включаем прерывания и таймер. Цикл while(1) пуст.

```
 main()
{
 __enable_irq();
 Conig_Timer2 ();
 TIM_ITConfig(TIM2, TIM_IT_Update,
 ENABLE);
 TIM_Cmd(TIM2, ENABLE);
 NVIC_EnableIRQ(TIM2_IRQn);
 while(1)
 {
 }
}
/*************************************/
```

Теперь нужно добавить код для обработчика прерываний:

```
/*************************************/
void TIM2_IRQHandler()
{
 TIM_ClearITPendingBit(TIM2, TIM_
 IT_Update);
 if (GPIO_ReadInputDataBit(GPIOE,
 GPIO_Pin_8) == 1)
 {
        GPIO_ResetBits(GPIOE,
 GPIO_Pin_8);
 }
 else
 {
        GPIO_SetBits(GPIOE, GPIO_
 Pin_8);
 }
}
/*************************************/
```

Конфигурация и работа с АЦП

Основные характеристики АЦП в STM32F3xx [4].

- АЦП является 12-битным.
- Имеются регулярные (имеют общий буфер) и инжектированные (индивидуальный буфер для каждого канала) каналы.
- Отдельный канал для встроенного температурного датчика и датчика напряжения питания микроконтроллера.
- Быстрое время преобразования – 0,2 мкс, причём это время не зависит от тактовой частоты шины AHB, на которой висят АЦП.
- Возможна генерация прерывания по окончании преобразования с инжектированного канала.
- Возможно прерывание от Analog Watchdog. Это нужно для того, чтобы следить, что измеренное напряжение не выходит за определённые значения. Причём может сканироваться как конкретный канал, так и группа каналов. В регистры ADC_HTR и ADC_LTR заносятся значения верхнего и нижнего порога соответственно, и в случае, если проверяемое напряжение выходит за эти пределы, генерируется прерывание.
- Возможно одиночное преобразование и преобразование в непрерывном режиме.
- Присутствует самокалибровка.
- Доступен запуск преобразования от внешнего события.
- Может работать в связке с DMA.
- Как и в других микроконтроллерах, возможно выравнивание результата по правому или по левому краям.

Структурная схема модуля АЦП представлена на рис. 9 [4].

Как и в предыдущем примере, будет использоваться библиотека STL и, соответственно, файлы:

stm32f30x_adc.h
stm32f30x_adc.c
stm32f30x_dma.h
stm32f30x_dma.c

В данной библиотеке всё реализовано точно так же, как и для любой другой периферии: в .c-файлах функции для настройки и работы с периферией, в .h – объявления специальных структур и переменных.

Настроим модуль **ADC1** так, чтобы он производил непрерывные преобразования одно за другим и результат при помощи **DMA** записывался в специально созданную переменную. Для реализации этого нужно выяснить, какой из каналов DMA требуется использовать.

Как видно из схемы на рис. 10, это первый канал DMA.

Для примера задействуем третий канал АЦП. Из даташита [3] и руководства пользователя узнаём, что третий канал ADC1 – это вывод PA2 нашего микроконтроллера.

Ниже пример кода. Необходимые подключаемые файлы:

```
/*************************************/
#include "stm32f30x_adc.h"
#include "stm32f30x_dma.h"
#include "stm32f30x_gpio.h"
#include "stm32f30x_rcc.h"
#include "stm32f30x_misc.h"
#include "stm32f30x.h"
/*************************************/
```

Объявление структур:

```
/*************************************/
GPIO_InitTypeDef GPIO_
InitStructure;
NVIC_InitTypeDef NVIC_
InitStructure;
ADC_InitTypeDef ADC_
InitStructure;
DMA_InitTypeDef DMA_
InitStructure;
ADC_CommonInitTypeDef ADC_
CommonInitStructure;
uint32_t ADC_Result;
/*************************************/
```

Инициализация всей периферии, которую будем использовать:

```
/*************************************/
void initialization(void)
{
 // Включаем тактирование DMA1,
 ADC12 и GPIOA.
 RCC_AHBPeriphClockCmd(RCC_
 AHBPeriph_DMA1, ENABLE);
```

```
RCC_AHBPeriphClockCmd(RCC_
AHBPeriph_ADC12, ENABLE);
RCC_AHBPeriphClockCmd(RCC_
AHBPeriph_GPIOA, ENABLE);
// Настраиваем пин на работу в
режиме аналогового входа.
GPIO_InitStructure.GPIO_Pin =
GPIO_Pin_2;
GPIO_InitStructure.GPIO_Mode =
GPIO_Mode_AN;
GPIO_Init(GPIOA, &GPIO_
InitStructure);
// Настройки DMA.
DMA_DeInit(DMA1_Channel1);
// Данные будем брать из регистра
данных ADC1.
DMA_InitStructure.DMA_
PeripheralBaseAddr =
(uint32_t)&(ADC1->DR);
// Переправлять данные будем в
переменную ADC_Result.
DMA_InitStructure.DMA_
MemoryBaseAddr = (uint32_t)&ADC_
Result;
// Передача данных из периферии в
память.
DMA_InitStructure.DMA_DIR =
DMA_DIR_PeripheralSRC;
// Размер буфера.
DMA_InitStructure.DMA_
BufferSize = 1;
// Адрес источника данных не
инкрементируем – он всегда один и
тот же.
DMA_InitStructure.DMA_
PeripheralInc = DMA_PeripheralInc_
Disable;
// Аналогично и с памятью.
 DMA_InitStructure.DMA_
MemoryInc = DMA_MemoryInc_
Disable;
// Настройки размера данных.
DMA_InitStructure.DMA_
PeripheralDataSize = DMA_
PeripheralDataSize_HalfWord;
DMA_InitStructure.DMA_
MemoryDataSize = DMA_
MemoryDataSize_HalfWord;
DMA_InitStructure.DMA_Mode =
DMA_Mode_Circular;
DMA_InitStructure.DMA_Priority =
DMA_Priority_High;
DMA_InitStructure.DMA_M2M =
DMA_M2M_Disable;
DMA_Init(DMA1_Channel1, &DMA_
InitStructure);
// Включаем первый канал DMA1.
DMA_Cmd(DMA1_Channel1,
ENABLE);
// Настраиваем тактирование АЦП.
RCC_ADCCLKConfig(RCC_
ADC12PLLCLK_Div2);
ADC_StructInit(&ADC_
InitStructure);
// Калибровка АЦП.
ADC_VoltageRegulatorCmd(ADC1,
ENABLE);
ADC_SelectCalibrationMode(ADC1,
ADC_CalibrationMode_Single);
ADC_StartCalibration(ADC1);
// Настраиваем непрерывные преоб-
разования.
ADC_CommonInitStructure.ADC_
Mode = ADC_Mode_Independent;
ADC_CommonInitStructure.ADC_
Clock = ADC_Clock_AsynClkMode;
ADC_CommonInitStructure.
ADC_DMAAccessMode = ADC_
DMAAccessMode_Disabled;
ADC_CommonInitStructure.ADC_
DMAMode = ADC_DMAMode_
OneShot;
ADC_CommonInitStructure.ADC_
TwoSamplingDelay = 0;
ADC_CommonInit(ADC1, &ADC_
CommonInitStructure);
// Включаем работу ДМА
через АЦП.
ADC_DMACmd(ADC1, ENABLE);
ADC_DMAConfig(ADC1, ADC_
DMAMode_Circular);
while(ADC_
GetCalibrationStatus(ADC1) !=
RESET );
// Продолжается настройка АЦП.
ADC_InitStructure.ADC_
ContinuousConvMode = ADC_
ContinuousConvMode_Enable;
ADC_InitStructure.ADC_Resolution =
ADC_Resolution_12b;
ADC_InitStructure.ADC_
ExternalTrigConvEvent = ADC_
ExternalTrigConvEvent_0;
ADC_InitStructure.ADC_
ExternalTrigEventEdge = ADC_
ExternalTrigEventEdge_None;
ADC_InitStructure.ADC_DataAlign =
ADC_DataAlign_Right;
 ADC_InitStructure.ADC_
OverrunMode = ADC_OverrunMode_
Disable;
ADC_InitStructure.ADC_
AutoInjMode = ADC_AutoInjec_
Disable;
ADC_InitStructure.ADC_
NbrOfRegChannel = 1;
ADC_Init(ADC1, &ADC_
InitStructure);
// Включаем третий канал первого
модуля АЦП.
ADC_RegularChannelConfig(ADC1, 3,
1, ADC_SampleTime_7Cycles5);
// Включаем АЦП.
ADC_Cmd(ADC1, ENABLE);
 while(!ADC_GetFlagStatus(ADC1,
 ADC_FLAG_RDY));
 ADC_StartConversion(ADC1);
 ADC_Result = ADC_
 GetConversionValue(ADC1);
}
/*************************************/
```

Осталось написать функцию main():

```
/*************************************/
int main(void)
{
 initialization();
 while(1)
 {
 }
}
/*************************************/
```

В теле главной функции вызываем созданную функцию инициализации, а в цикле while(1) пусто – то есть процессор свободен, всю работу взял на себя модуль DMA. Если прошить этот код в микроконтроллер, то в отладчике можно увидеть, что значение переменной ADC_Result меняется в соответствии с уровнем сигнала на выводе PA2.

Выводы

В приведённой статье были рассмотрены основные практические приёмы работы с микроконтроллерами ряда STM32F10x и STM32F30x.

К сожалению, объём статьи не позволяет сделать полный обзор алгоритмов работы со всей периферией, входящей в состав данных типов МК. Более подробную информацию по модели программирования и интерфейсам STM32F можно получить в источнике [5].

В последующей статье будет сделан обзор телекоммуникационных возможностей STM32F с иллюстрацией программного кода.

Литература

1. Datasheet STM32F103xC STM32F103xD STM32F103x. P. 12–13, 16, 17.
2. RM0008 Reference manual STM32F101xxx … STM32F107xxx. P. 90–98, 159–174.
3. Datasheet STM32F303xB STM32F303xC. P. 12, 19, 22–24.
4. RM0316 Reference manual STM32F303xB/C, STM32F303x6/8, STM32F328x8 and STM32F358xC advanced ARM®-based 32-bit MCUs. P. 174–188, 216–232, 413–440.
5. *Мартин М.* Инсайдерское руководство по STM32. URL: https://istarik.ru/file/STM32.pdf.