Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра комплексной и информационной безопасности
электронно-вычислительных систем (КИБЭВС)

М.В. Пилданов, В.С. Аврамчук

Аппаратные средства телекоммуникационных систем

Методические указания к лабораторным и практическим работам
для студентов направлений подготовки
10.05.02 - Информационная безопасность телекоммуникационных систем
(Управление безопасностью телекоммуникационных систем и сетей)

Томск 2026

УДК 004.421
ББК 32.973.26-018
П 32

Пилданов М.В. АППАРАТНЫЕ СРЕДСТВА ТЕЛЕКОММУНИКАЦИОННЫХ СИСТЕМ: Методические указания к лабораторным и практическим работам для студентов направлений подготовки 10.05.02 - Информационная безопасность телекоммуникационных систем (Управление безопасностью телекоммуникационных систем и сетей) / М. В. Пилданов, В. С. Аврамчук. — Томск: ТУСУР, 2026. ─ 205 с.

Настоящее учебно-методическое пособие содержит описания лабораторных и практических работ по дисциплине «Аппаратные средства телекоммуникационных систем» для студентов направлений подготовки «10.05.02 - Информационная безопасность телекоммуникационных систем (Управление безопасностью телекоммуникационных систем и сетей)».

Одобрено на заседании кафедры КИБЭВС протокол №10 от 02.06.2026 года

УДК 004.421
ББК 32.973.26-018

Оглавление

ВВЕДЕНИЕ

Целью освоения дисциплины «Аппаратные средства телекоммуникационных систем» является приобретение студентами знаний по основам построения, принципам функционирования, разновидностях, способах реализации, областях применения, направлении развития и, как следствие, возможностей использования на практике аппаратных средств телекоммуникационных систем.

В результате освоения дисциплины обучающийся должен:

– знать элементную базу вычислительной техники (ВТ);

– изучить принципы построения и функционирования комбинационных схем и цифровых автоматов;

– изучить основные особенности архитектуры и структуры различных классов процессоров (микропроцессоров);

– изучить принципы работы микропроцессорных систем;

– овладеть аппаратно-программными средствами ВТ, применяемыми во встроенных системах;

– сформировать способность участвовать в разработке компонентов телекоммуникационных систем;

– сформировать способность применять положения теорий цифровой обработки сигналов, информации и кодирования, электрической связи для решения профессиональных задач;

– изучить основные протоколы связи, используемые в телекоммуникационных системах.

1 Практическая работа №1. Введение. Знакомство с отладочной платой и средой программирования.

1.1 Цели практики

Познакомиться с устройством и принципом работы с отладочной платой Nucleo-F103RB на базе микроконтроллера STM32F103RBT6. Рассмотреть средства разработки и отладки программ для отладочной платы Nucleo-F103RB.

1.2 Теоретическая часть

Отладочная плата.

В рамках данного практического курса будет использоваться отладочная плата Nucleo-F103RB (рис. 1.1) на базе микроконтроллера STM32F103RBT6 с архитектурой ARM Cortex-M3.

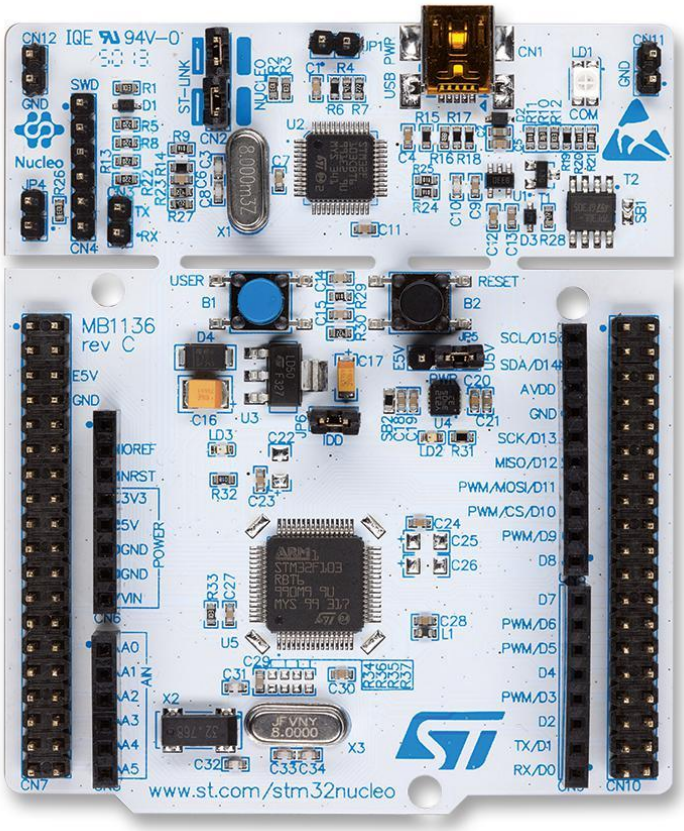


Рисунок 1.1 – Отладочная плата nucleo-F103RB

На рис. 1.2 представлены основные элементы отладочный платы nucleo-F103RB.

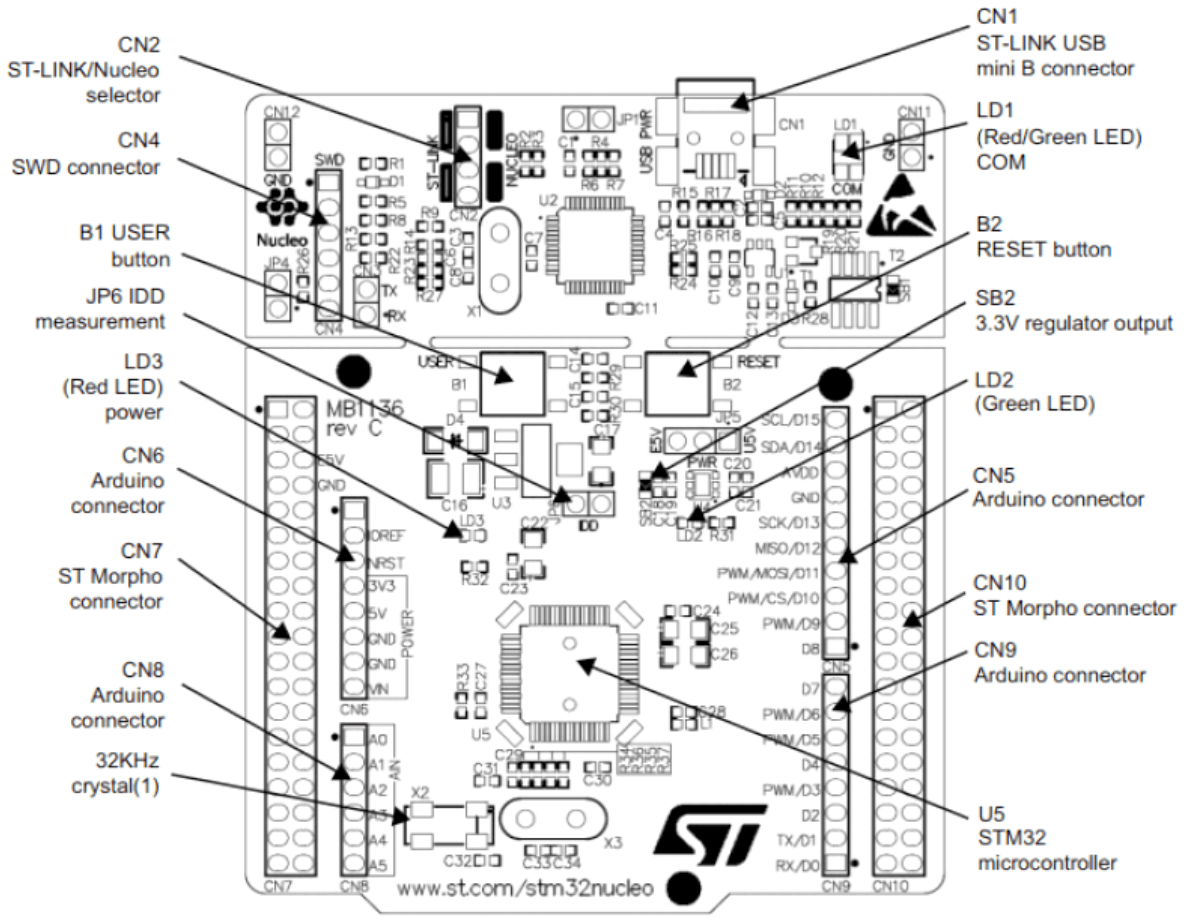


Рисунок 1.2 – Основные элементы nucleo-F103RB

Одним из наиболее важных элементов платы является микроконтроллер **STM32F103RBT6**. В классификации микроконтроллеров STM32 по назначению (рис. 1.3) принадлежит к классу «Mainstream».

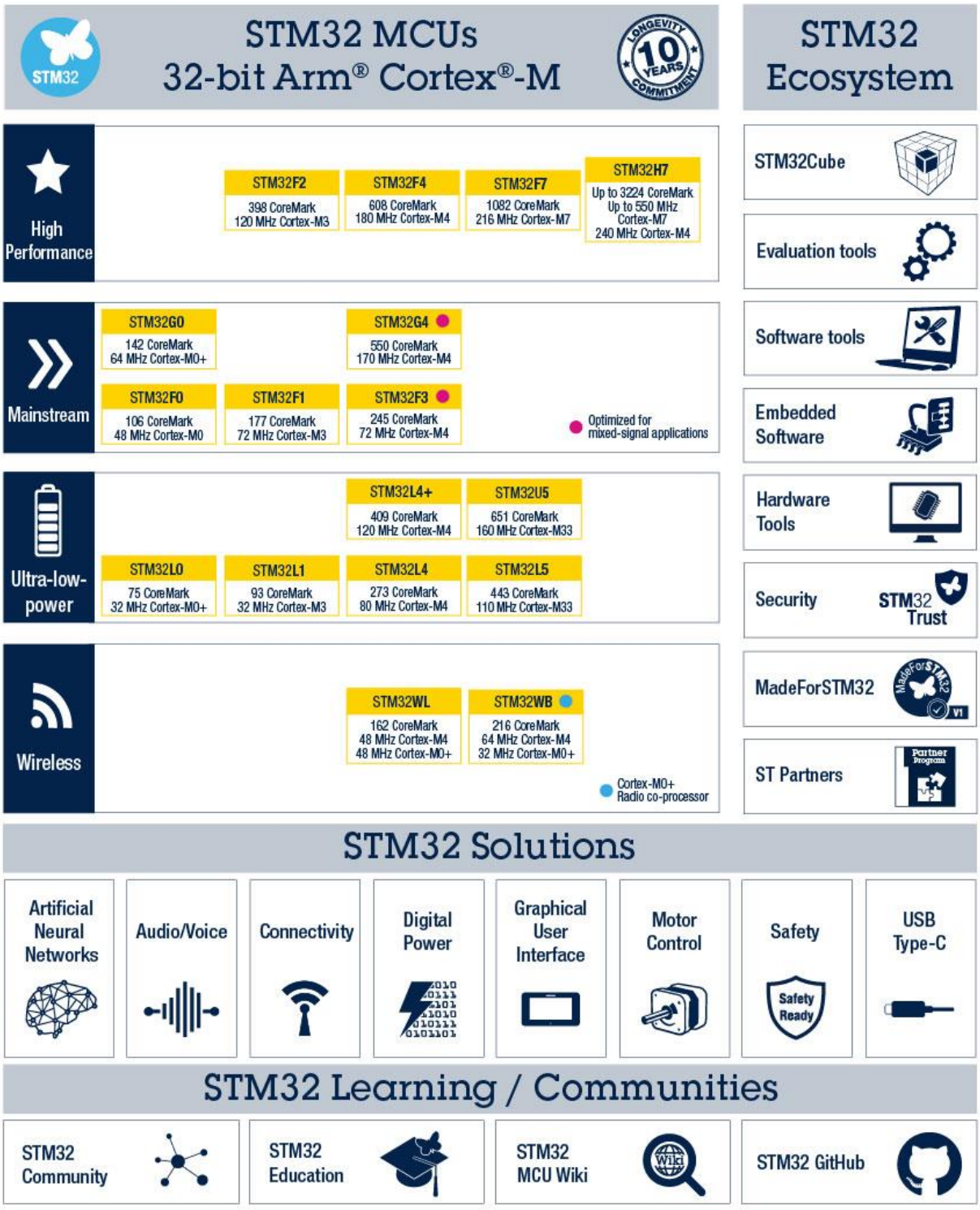


Рисунок 1.3 – Классификация STM32 по применению

На рисунке 1.4 представлена классификация микроконтроллеров по размеру памяти. **STM32F103RBT6** имеет 128 Кб памяти Flash (для хранения исполняемых файлов) и 20 Кб ОЗУ (SRAM) для хранения команд и данных.

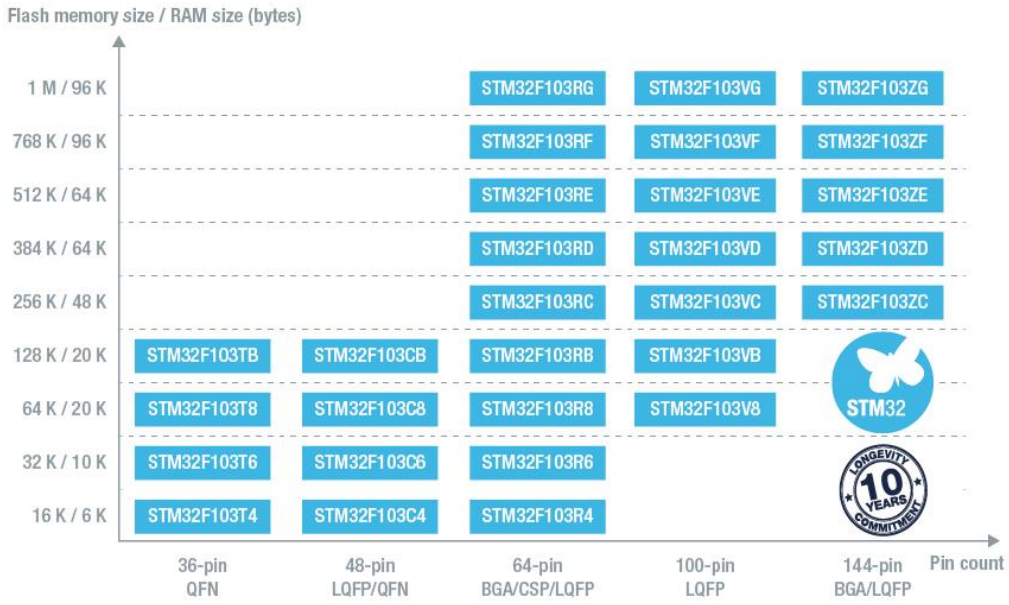


Рисунок 1.4 – Классификация STM32 по размеру памяти

В виду того, что сам микроконтроллер является элементом отладочной платы, которая реализует удобный («безпаечный») способ подключения периферийных устройств к микроконтроллеру, необходимо иметь представление о расположении контактов (pins, «пинов», рис. 1.5) и их соответствии выводам микроконтроллера (рис. 1.6).

Семейство отладочных плат Nucleo обеспечивает доступный и гибкий путь для проверки новых идей и изготовления прототипов устройств с любыми STM32 микроконтроллерами указанной линейки, выбирая различные комбинации производительности, потребляемой мощности и других характеристик. Поддержка соединителей Arduino (для подключения Arduino-совместимых устройств и переноса проектов на данную архитектуру) и ST Morpho (разъемы расширения для полного доступа ко всем выводам I/O STM32) (рис. 1.5) позволяет просто и быстро расширить функциональность открытой платформы STM32 Nucleo, используя широкий спектр специализированных плат расширения. Платы семейства не требуют отдельного инструментария для программирования и отладки, так как на STM32 Nucleo уже интегрирован ST-LINK/V2-1. Отладочные платы поставляются с комплектом программных библиотек HAL совместно с различными пакетами

примеров, а также с возможностью прямого доступа к онлайн ресурсам комплекса mbed. имеются специальные выводы.

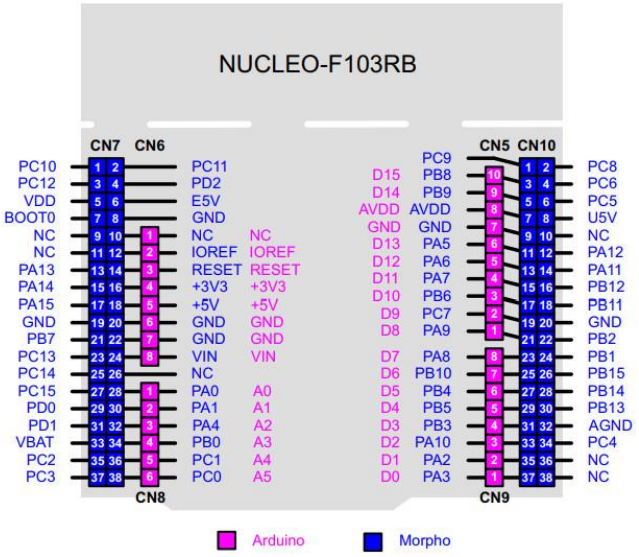


Рисунок 1.5 – Расположение контактов отладочной платы

Устройство микроконтроллера STM32F103RBT6. Расположения контактов микроконтроллера представлено на рисунке 1.5.

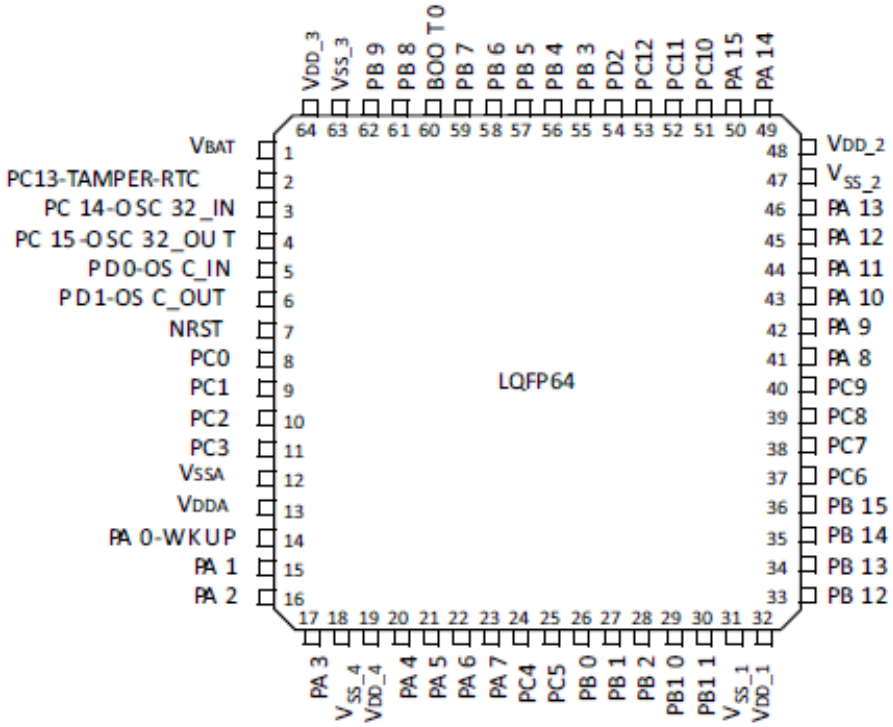


Рисунок 1.5 – Расположение контактов микроконтроллера

Структурная схема микроконтроллера STM32F103RBT6 представлена на рис. 1.6.

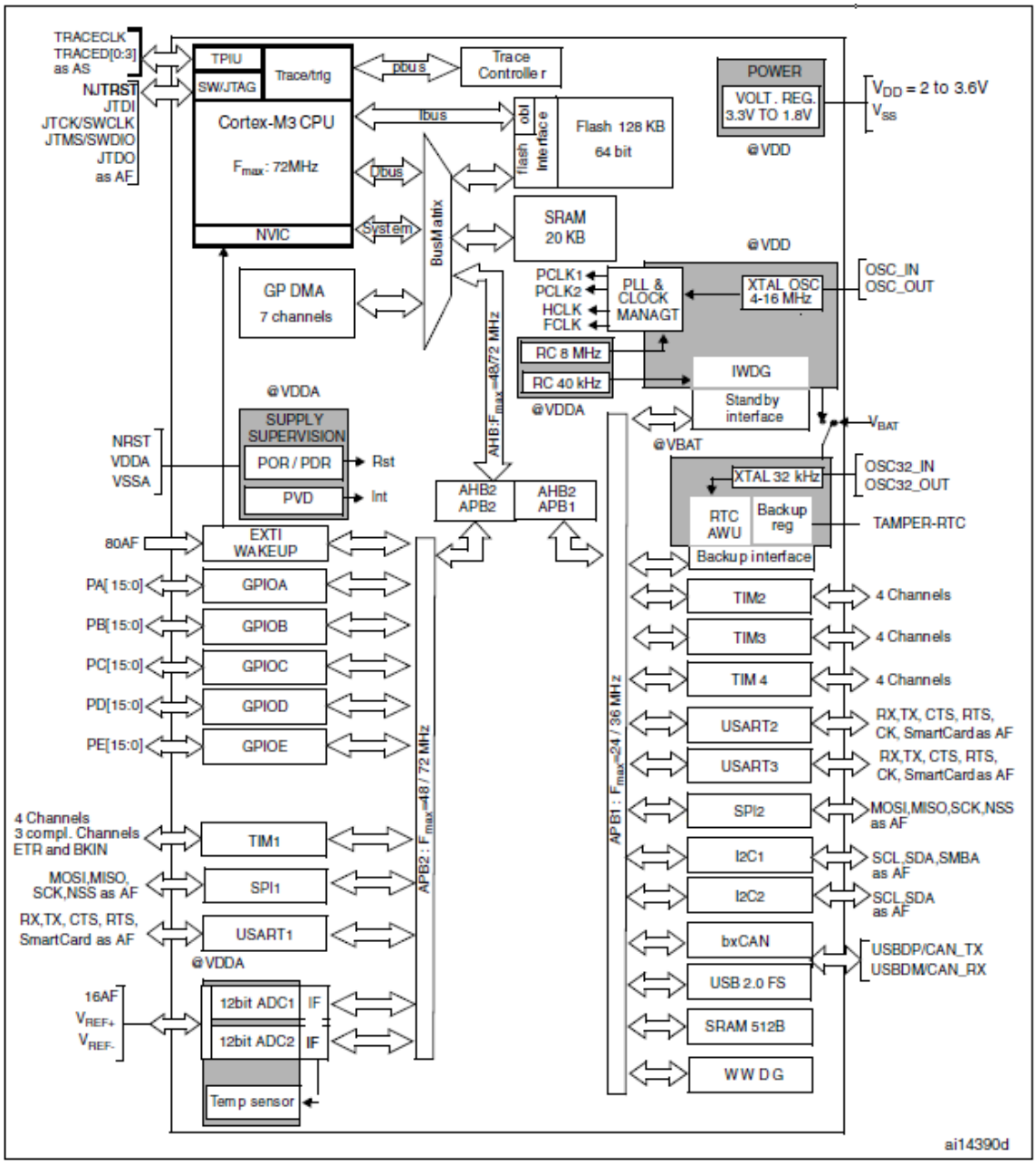


Рисунок 1.6 – Структурная схема микроконтроллера STM32F103RBT6

Для взаимодействия с периферийными устройствами в структурной схеме можно выделить 16-разрядные порты ввода-вывода общего назначения GPIOA -GPIOE (General Purpose Input Output), порты SPI1 и SPI2, I2C1 и I2C2, USART1 ¬ USART3, а также 12-разрядный аналогового-цифровые преобразователи ADC1 и ADC2. В таблице 1.1 представлены характеристики микроконтроллера STM32F103RBT6 их обозначения в структурной схеме.

Таблица 1.1 – Характеристики микроконтроллера STM32F103RBT6

Характеристика	Значение	Обозначение в схеме
Ядро	ARM® 32-bit Cortex®-M3	Cortex-M3 CPU
Макс. частота ядра	72 МГц	F_{max}: 72MHz
Частоты внешних генераторов:		
Высокоскоростной	4 16 МГц	XTAL_OSC 4-16MHz
Низкоскоростной	32,768 кГц	XTAL 32 kHz
Частоты внутренних генераторов:		
Высокоскоростной	8 МГц	RC 8 MHz
Низкоскоростной	40 кГц	RC 40 kHz
Память Flash	128 Кбайт (предвыборка 64 бита)	Flash 128 KB 64 bit
Память SRAM	20 Кбайт	SRAM 20 KB
	512 байт	SRAM 512 B
Напряжение питания	2 3.6 В	POWER
АЦП	2 x 12 бит, до 16 каналов	12 bit ADC1 и ADC2
Таймеры:	7	
16 битный таймер	4	TIM1 TIM4
Сторожевой таймер	2	IWDG, WWDG
Системный таймер	1	SysTick
Интерфейсы:		
I2C	2	I2C1, I2C2
SPI	2	SPI1, SPI2
USART	3	USART1 USART2
CAN	1	bxCAN
USB	1	USB 2.0 FS

Flash-память.

Доступ к данным и инструкциям flash-памяти выполняются через шину AHB. Блок предварительной выборки используется для предвыборки инструкций по шине IBus или данных по шине DBus.

Буфер предварительной выборки (2 x 64-битных блока) разрешается после сброса микроконтроллера. Благодаря буферу предварительной выборки возможно более быстрое выполнение команд, поскольку МК выбирает одно слово за раз, а следующее слово можно получить уже из буфера.

На рисунке 1.7 представлена структура Flash-памяти микроконтроллера. Основная (пользовательская) память разбита на 128 страниц по 1 Кб. Также память имеет системную область (Information block) размером 2016 байт и специальные регистры настроек flash-памяти.

Block	Name	Base addresses	Size (bytes)
Main memory	Page 0	0x0800 0000 - 0x0800 03FF	1 K
	Page 1	0x0800 0400 - 0x0800 07FF	1 K
	Page 2	0x0800 0800 - 0x0800 0BFF	1 K
	Page 3	0x0800 0C00 - 0x0800 0FFF	1 K
	Page 4	0x0800 1000 - 0x0800 13FF	1 K
	. . .	. . .	. . .
	Page 127	0x0801 FC00 - 0x0801 FFFF	1 K
Information block	System memory	0x1FFF F000 - 0x1FFF F7FF	2 K
	Option bytes	0x1FFF F800 - 0x1FFF F80F	16
Flash memory interface registers	FLASH_ACR	0x4002 2000 - 0x4002 2003	4
	FLASH_KEYR	0x4002 2004 - 0x4002 2007	4
	FLASH_OPTKEYR	0x4002 2008 - 0x4002 200B	4
	FLASH_SR	0x4002 200C - 0x4002 200F	4
	FLASH_CR	0x4002 2010 - 0x4002 2013	4
	FLASH_AR	0x4002 2014 - 0x4002 2017	4
	Reserved	0x4002 2018 - 0x4002 201B	4
	FLASH_OBR	0x4002 201C - 0x4002 201F	4
	FLASH_WRPR	0x4002 2020 - 0x4002 2023	4

Рисунок 1.7 – Структура Flash-памяти STM32F103RBT6

Система тактовых генераторов.

Важнейшей системой микроконтроллера является его система тактовых генераторов. В виду того, что разные периферийные устройства микроконтроллера работают на разных частотах, а также не всегда используются в определенных задачах, необходимо иметь возможность гибкой настройки

тактовых частот. На рисунке 1.8 представлена система тактовых генераторов STM32F103RBT6. Основными источниками тактовых сигналов являются внешние (HSE – High Speed External, LSE – Low Speed External) и внутренние (HSI – High Speed Internal, LSI – Low Speed Internal) генераторы сигналов, которые задают частоты, согласно таблице 1.1. Все остальные частоты для различных устройств получаются путем умножения (блок PLLMUL, PLL — система фазовой автоподстройки частоты) или деления (AHB Prescaler) базовых частот на определённый коэффициент.

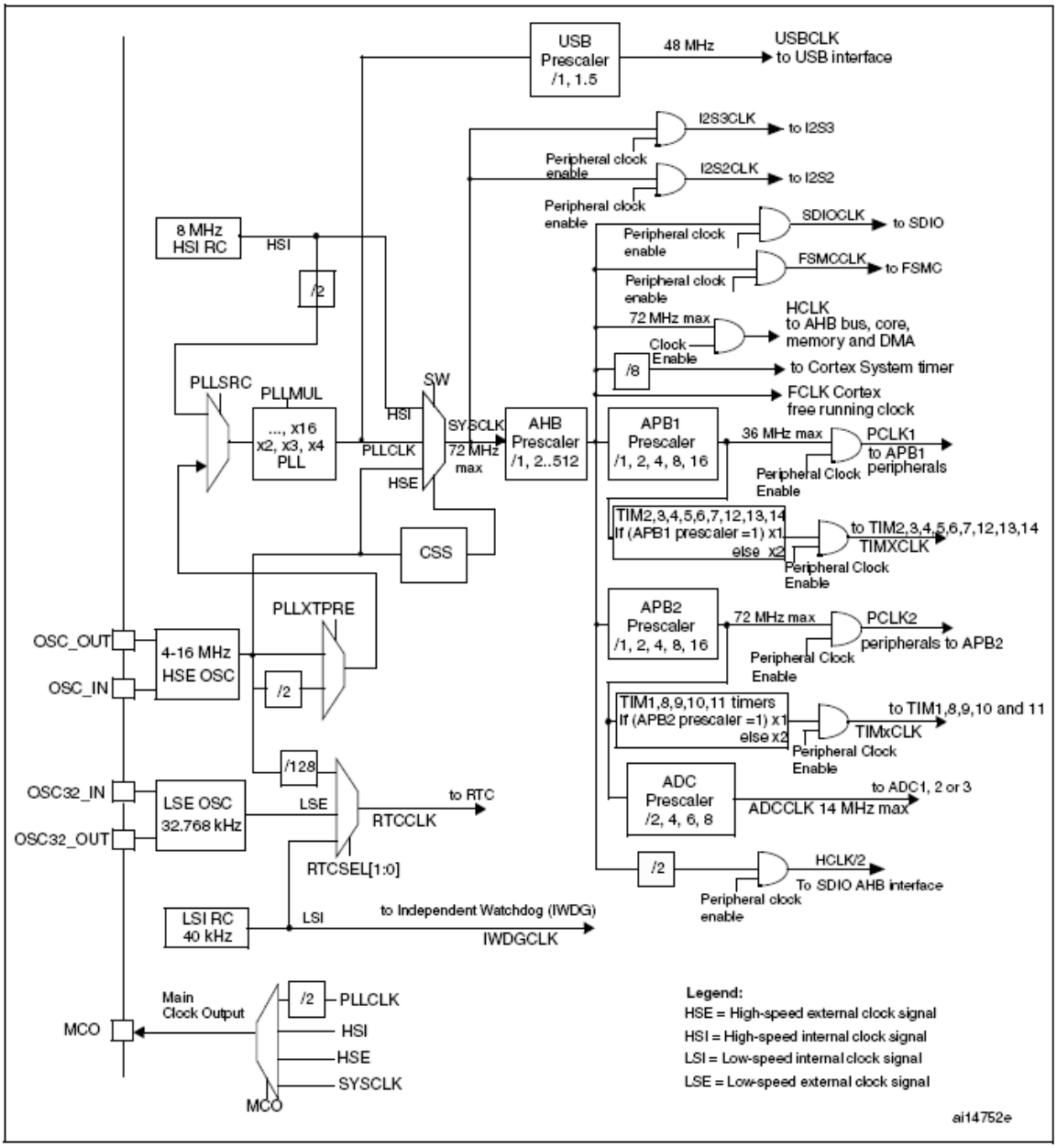


Рисунок 1.8 – Система тактовых генераторов STM32F103RBT6

Работа с частотами осуществляется через настройки значений в специальных регистрах RCC_CR (clock control register, рис. 1.9) и RCC_CFGR (clock configuration register, рис. 1.10).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved						PLL RDY	PLLON	Reserved				CSS ON	HSE BYP	HSE RDY	HSE ON
						r	rw					rw	rw	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]					Res.	HSI RDY	HSION
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

Рисунок 1.9 – Регистр RCC_CR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					MCO[2:0]			Res.	USB PRE	PLLMUL[3:0]				PLL XTPRE	PLL SRC
					rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADCPRE[1:0]		PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

Рисунок 1.10 – Регистр RCC_CFGR

High Speed Internal (HSI).

Встроенный в микропроцессор генератор HSI вырабатывает тактовую частоту 8 МГц. Генератор автоматически запускается при появлении питания Vcc и при выходе в нормальный режим работы выставляет флаг HSIRDY в регистре RCC_CR. Первоначально процессорное ядро запускается на тактовой частоте HSI. К преимуществам относится быстрое время начала генерации тактовой частоты после подачи питания и отсутствие необходимости в использовании дополнительных электронных компонентов для работы микроконтроллера. Недостаток – низкая стабильность частоты генерируемого сигнала, а при умножении на PLL погрешность тоже умножается.

High Speed External (HSE).

Внешний генератор HSE по умолчанию выключен и его включение/выключение управляется битом HSEON регистра RCC_CR. После включения HSE и его выхода в рабочий режим устанавливается бит HSERDY кроме этого, может быть сгенерировано прерывание. Также как и сигнал с генератора HSI, сигнал HSE может быть подан напрямую в качестве систем-

ного тактового сигнала либо поступать в блок умножения. Но в отличие от HSI в блок умножения он может поступать напрямую, либо пройдя через делитель на 2 (рис. 1.8).

В качестве внешнего генератора могут выступать:

– внешний тактовый сигнал, не превышающий 25МГц, поданный на ножку OSC_IN в то время, как нога OSC_OUT находится в высокоимпендансном состоянии;

– внешний кварцевый резонатор, подключенный на ножки OSC_IN и OSC_OUT. Внешний кварцевый резонатор должен быть в диапазоне от 4 до 16МГц и при его использовании достигается очень высокая стабильность частоты работы генератора.

Low Speed Internal (LSI) и Low Speed External (LSE).

Для работы с такими устройствами, как RTC (Real Time Clock – часы реального времени) и IWDG (Independent Watchdog – независимый сторожевой таймер) используются низкоскоростные тактовые сигналы LSE (Low Speed External – внешний низкоскоростной, для работы которого нужен внешний «часовой» кварц на 32.768кГц и LSI (Low Speed Internal – внутренний низкоскоростной, который генерирует 40кГц).

Phase lock loop (PLL).

Внутренний умножитель частоты (PLLMUL) может умножать частоту вошедшего тактового сигнала с одного из трех источников, на выбор:

– HSI/2 (половина частоты от HIS);

– HSE (частота HSE);

– HSE/2 (половина частоты от HSE).

Множитель варьируется от 2-х до 16-ти. По умолчанию умножитель выключен и его включение/выключение управляется битом PLLON регистра RCC_CR. После включения PLL и его выхода в рабочий режим устанавливается бит PLLRDY кроме этого, может быть сгенерировано прерывание. Работа умножителя конфигурируется через регистр RCC_CFGR. ***Работать с***

режимами работы умножителя необходимо только при выключенном PLL, т.е. при наличии нуля в поле PLLON регистра RCC_CR.

Для того, чтобы определить, какой именно источник сигнала использовать перед умножителем, в регистре RCC_CFGR есть бит PLLSRC, который задает источник либо HSE (значение 1) либо HSI/2 (значение 0). Чтобы выбрать источник HSE/2, необходимо установить бит в поле PLLXTPRE. Коэффициент умножения задаётся в диапазоне от 2 до 16 битами PLLMUL[3:0] (0000 – коэффициент 1, 1111 ¬¬– коэффициент 16).

После блока PLLMUL стоит мультиплексор, с помощью которого можно выбрать источник для системной тактовой частоты (SYSCLK). Для выбора есть специальное поле SW регистра RCC_CFGR. (значения 00 ¬– HSI, 01 – HSE, 10 ¬¬– PLLCLK, 11 – не разрешено).

Примечание: *из схемы на рис. 1.8 можно заметить, что шина USB тактируется только с частотой 48 МГц. У шины USB есть собственный делитель (USB Prescaler), с коэффициентами деления 1 и 1.5. Это означает, что при использовании шины USB сигнал PLLCLK должен иметь частоту либо 48 МГц (при USB Prescaler = 1) либо 72 МГц (при USB Prescaler = 1,5).*

Advanced High-performance Bus (AHB) Prescaler.

Системная тактовая частота SYSCLK попадает в делитель шины AHB, который делит ее на делитель от 1 до 512, и все остальные блоки получают на вход уже эту деленную частоту (некоторые блоки имеют еще свои дополнительные делители). По схеме на рис. 1.8 видно, какой блок имеет делитель, а какой нет, какие коэффициенты деления у делителей, и какая периферия в итоге какую частоту получает.

Например, блоки SDIO и FSMC работают на частоте шины AHB. Шины APB1 и APB2 имеют собственные делители с коэффициентами от 1 до 16. Блок ADC имеет делитель от 2 до 8 и т.д.

Так же на схеме рис. 1.8 подписаны максимальные допустимые частоты для блоков. Все делители, имеющие управляемые коэффициенты, могут

быть настроены, что в целом позволяет довольно гибко варьировать частоты отдельных блоков в различных пределах.

Шины для GPIO, I2C, ADC, TIM

На схеме рис. 1.8 не показано, к какой шине подключены порты ввода-вывода общего назначения и некоторые другие интерфейсы. В таблице 1.2 приведена принадлежность шин из рис. 1.8 к определенным интерфейсам или устройствам.

Таблица 1.2 – Соответствие шины и интерфейса

Интерфейс/устройство	**Шина**
GPIO	APB2
I2C	APB1
TIM	APB1/ APB2
ADC	APB2
DMA	AHB

Примечание: *Каждый блок имеет свой вход тактовых сигналов и бит управления этим входом.* ***По умолчанию практически все блоки отключены от тактовых сигналов.*** *Для работы с устройствами необходимо не забывать устанавливать соответствующие биты управляющих регистров для включения тактирования.*

Разнообразие делителей, умножителей и источников синхронизации необходимо для гибкой и точной настройки микроконтроллера под задачу, которую ему предстоит решать в том или ином проекте. Если не нужны какие-то блоки, то их можно отключить от источников синхронизации, тем самым уменьшив энергопотребление. Также, когда в проекте не требуется высокая производительность, можно сконфигурировать микроконтроллер на работу на меньшей частоте, что опять же понизит его энергопотребление.

Система защиты.

Иногда источники тактирования могут не функционировать, в связи с чем при выборе их управляющими регистрами элементы микроконтроллера могут не работать, т.е. устройство будет ожидать тактирования, а его не бу-

дет. Для решения подобных ситуаций, есть система защиты, по правилам которой можно переключиться на другой источник тактирования только в том случае, когда точно известно, что с ним все в порядке. Реализована такая система на флагах, т.е., например, если включить PLL, нужно дождаться пока появится флаг готовности PLL и только после этого переключаться на источник сигнала.

Существует также специальный блок, который на схеме рис. 7 обозначен как CSS. Из схемы видно, что на вход ему подается тактовая частота с HSE а выход из этого блока заведен на блок выбора источника системного тактового сигнала. Назначение этого блока — следить за тактовыми сигналами, поступающими с HSE и если с ними произойдет нештатная ситуация, например, они пропадут (кварц может «отвалиться»), то этот блок может вмешаться в ситуацию и тогда происходит следующее:

– автоматически отключается HSE;

– посылается событие на останов работы расширенных таймеров TIM1 и TIM8;

– генерируется прерывание CSSI, которое внутри процессора заведено на немаскируемое прерывание NMI, т.е. пропустить его невозможно;

– если в качестве источника системной частоты использовался HSE напрямую либо через PLL, то источник системной частоты автоматом переключается на HSI;

– если PLL был активен и входом ему служил сигнал с HSE, то PLL отключается.

Таким образом, микроконтроллер сохранит свою работоспособность, пусть и на пониженной и менее стабильной частоте, и проинформирует выполняющуюся программу об этом.

По умолчания данная защита выключена. Для ее включения необходимо установить в единицу бит CSSON регистра RCC_CR.

1.3 Создание проекта в программно-аппаратной платформе mbed

Mbed – программно-аппаратная платформа и операционная система для устройств на базе МК ARM Cortex-M. Платформа работает онлайн через браузер и содержит редактор кода, компилятор, набор библиотек с примерами кода под разные периферийные устройства.

Отличительной особенностью mbed является простота программирования для новичков. Данная платформа по своему принципу работы чем-то схожа с arduino IDE, где используются си-подобные скетчи.

Создадим простой проект в mbed. Для этого необходимо перейти на сайт https://os.mbed.com/mbed-os/ и выбрать Compiler (рис. 10, доступно для зарегистрированных пользователей).

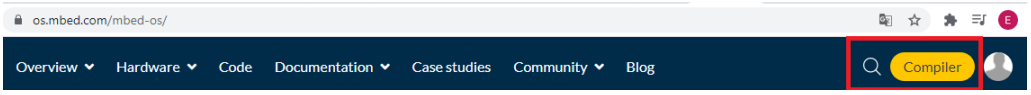


Рисунок 1.11 – Компилятор mbed

Затем появится окно проектов (рис. 1.12). Здесь можно увидеть все предыдущие начатые проекты, которые сохраняются автоматически и привязаны к профилю пользователя.

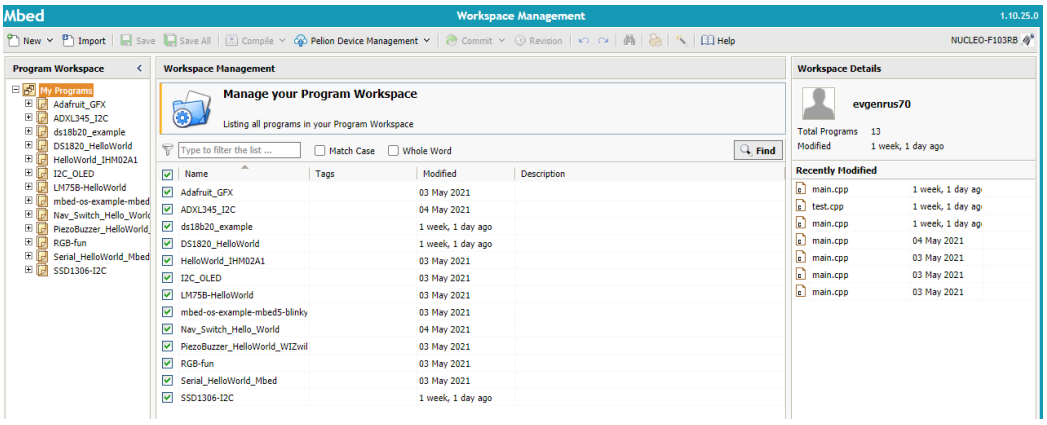


Рис. 1.12 – Окно проектов

Создадим новый проект (New). Выберем целевую платформу. В нашем случае отладочную плату nucleo-F103RB (рис. 1.13). Также выберем шаблонный проект, на основе которого можно в дальнейшем разрабатывать свои решения.

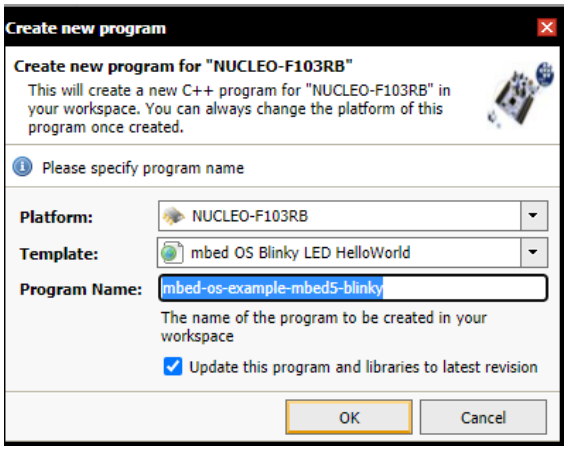


Рисунок 1.13 – Создание проекта

После создания проекта в дереве можно увидеть его структуру (рис. 1.14). В файле main.cpp находится основная программа, реализующая мигание светодиодом каждые 500 мс.

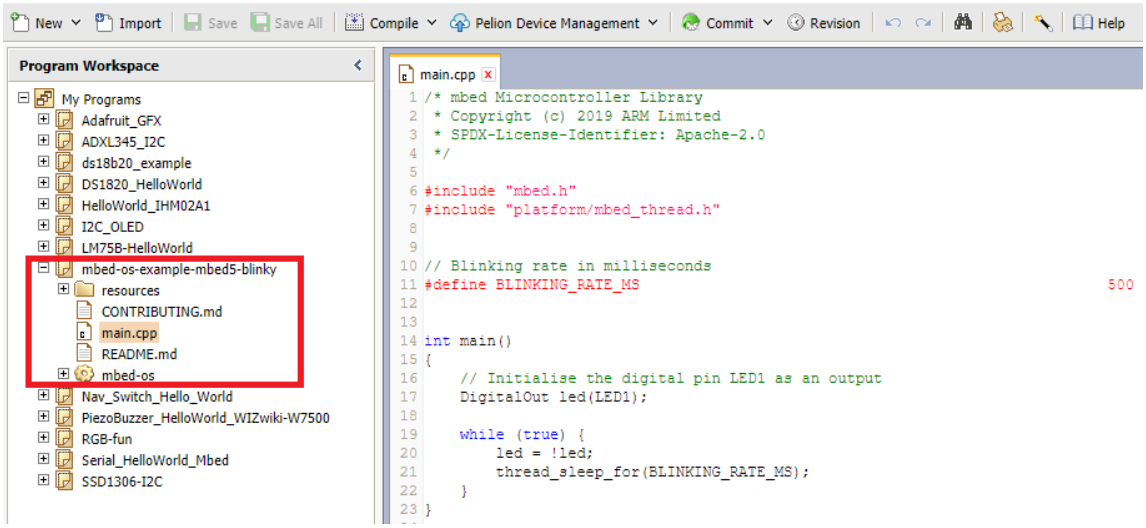


Рисунок 1.14 – Созданный проект

Скомпилируем проект, нажав кнопку Compile. Процесс компиляции представлен на рис. 1.15.

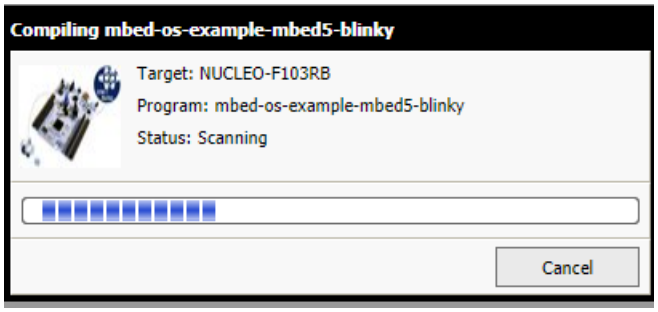


Рисунок 1.15 – Компиляция проекта

При успешной компиляции в окне Compile Output появится сообщение Success! После компиляции будет автоматически скачан исполняемый файл в формате .bin. Этот файл необходимо отправить во flash-память устройства простым копированием или через функцию проводника «Отправить» (рис. 1.16).

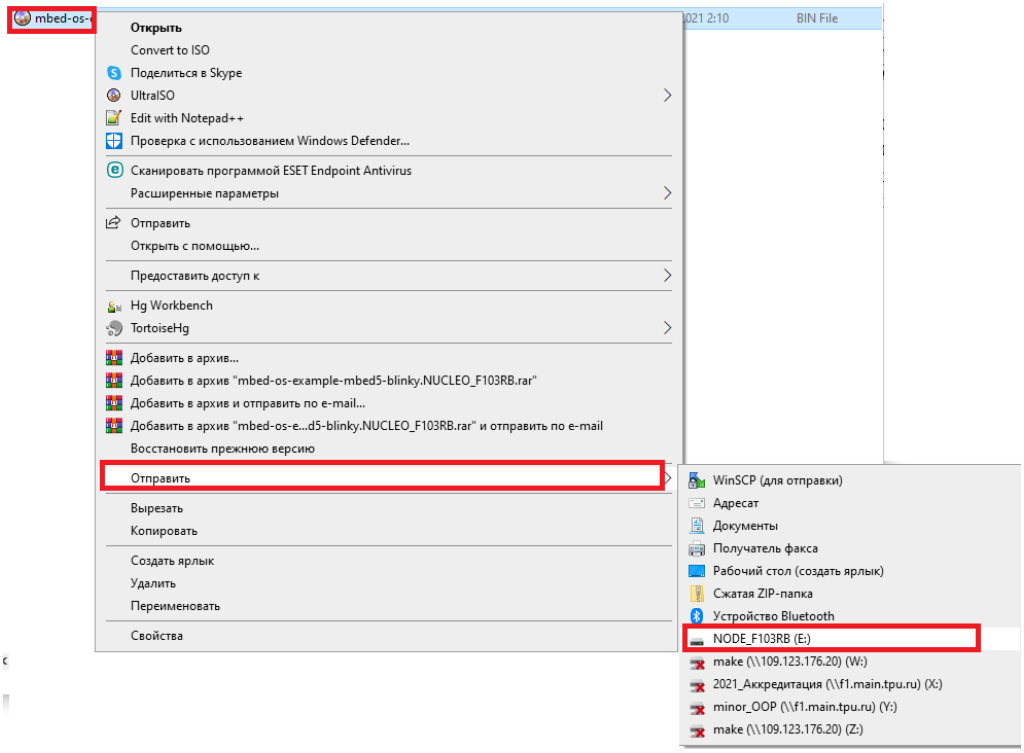


Рисунок 1.16 – Прошивка отладочной платы

1.4 Создание проекта в среде keil 5

Для разработки и отладки сложных и более комплексных проектов можно использовать среду разработку keil 5. В отличие от mbed, создание проекта в keil требует более детальной настройки, а процесс разработки и отладки имеет более высокий порог вхождения.

Рассмотрим процесс создания проекта в keil 5.

Выберем Project->New mVision Project и зададим директорию для проекта (рис. 1.17). Затем выберем устройство (рис. 1.18).

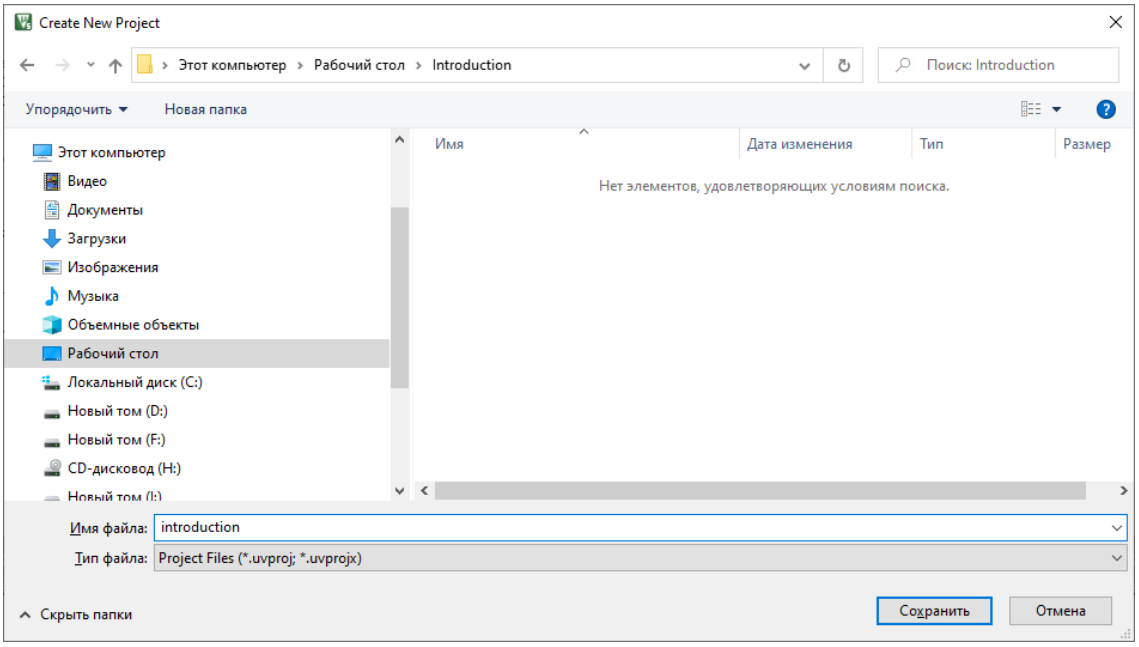


Рисунок 1.17 – Выбор директории проекта

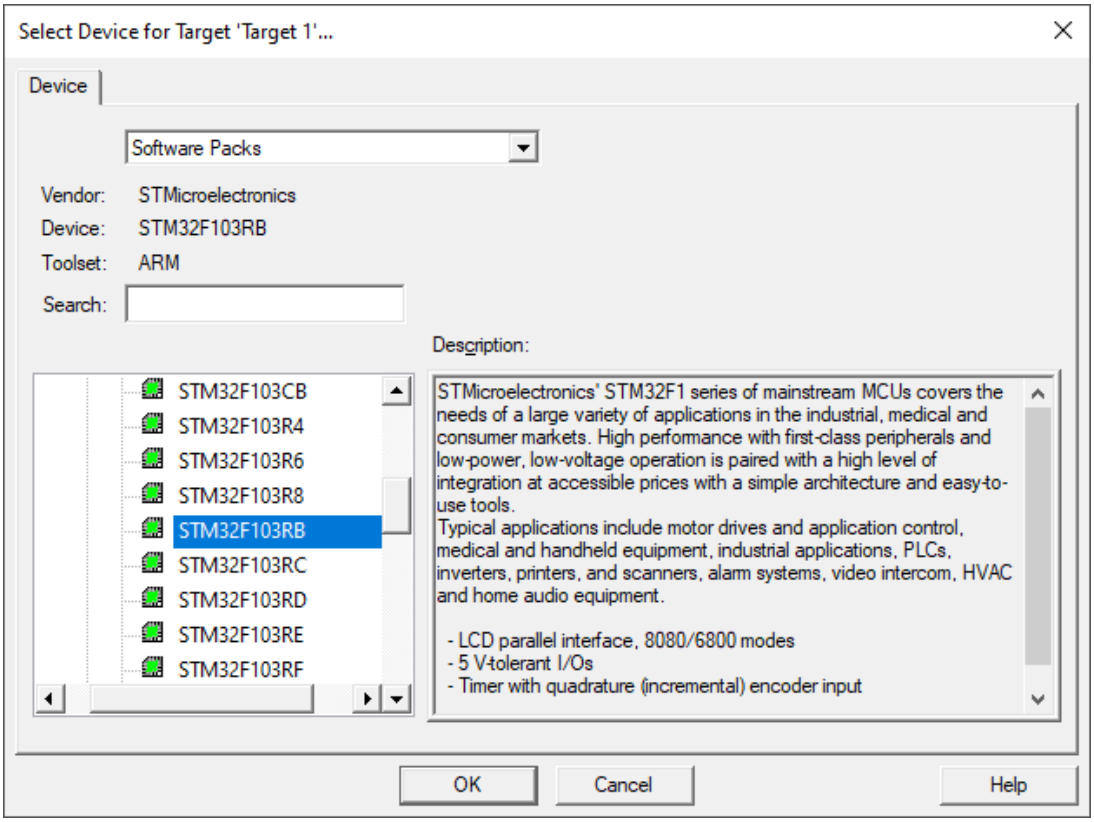


Рисунок 1.18 – Выбор устройства

В настройка среды выполнения выберем пункты, согласно рисунку 1.19.

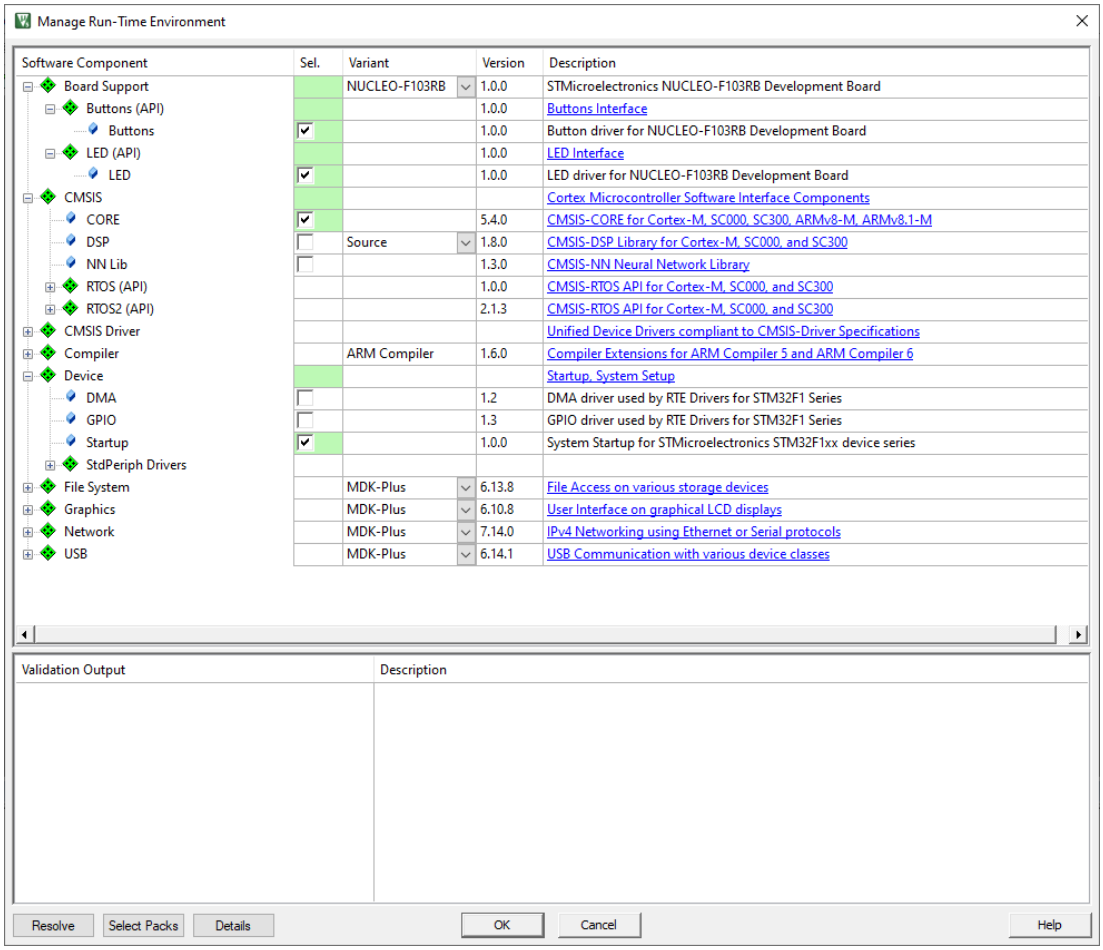


Рисунок 1.19 – Настройка среды выполнения

Board Support позволяет использовать готовые библиотеки для работы с периферией конкретной отладочной платы. Выберем кнопки и светодиоды.

CMSIS (Common Microcontroller Software Interface Standard) – это независимый от производителя уровень аппаратной абстракции для серии ядер Cortex-M, а также интерфейс отладчика (debugger). CMSIS предоставляет последовательные и простые интерфейсы для ядра, его периферии и операционных систем реального времени.

CMSIS-CORE – API для ядра Cortex-M и периферии. Эта часть предоставляет стандартизированный интерфейс для Cortex-M0, Cortex-M3, Cortex-M4, SC000, и SC300. Включает в себя также дополнительные SIMD-инструкции для Cortex-M4.

Device->Startup – включить файл настроек инициализации для микроконтроллеров STM32F1xx.

После применения настроек и создания проекта появится дерево проекта (рис. 1.20). Для написания программ необходимо добавить файлы исходного кода (рис. 1.21, 1.22).

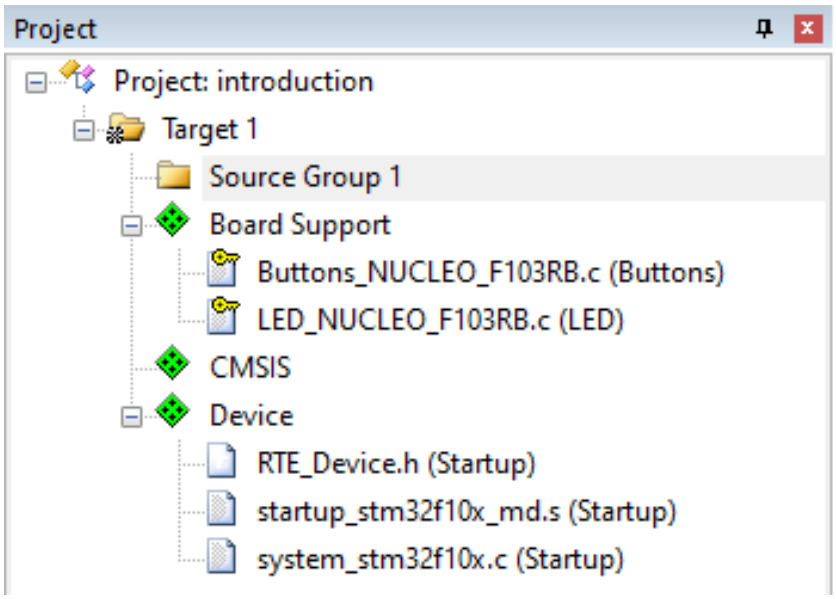


Рисунок 1.20 – Структура проекта

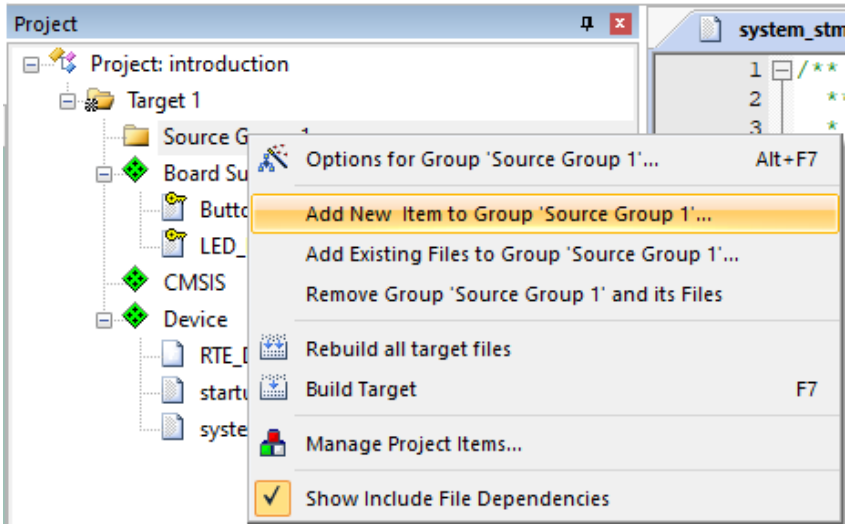


Рисунок 1.21 – Добавление файлов исходного кода

Добавим файл main.c в проект и напишем небольшую программу для мигания светодиодом с заданной задержкой на отладочной плате (рис. 1.22).

Для того, чтобы настроить свою частоту, используя настроечные регистры, необходимо добавить функцию настройки SystemCoreClockConfigure. Реализация функции представлена на рис. 1.23.

main.c | stm32f10x.h | startup_stm32f10x_md.s | system_stm32f10x.c | core_cm3.h | LED_NUCLEO_F103RB.c

```c
#include "stm32f10x.h"
#include <stdio.h>
#include "Board_LED.h"                  // ::Board Support:LED

volatile uint32_t msTicks;                                  // counts 1ms timeTicks
/*----------------------------------------------------------------------------
 * SysTick_Handler:
 *----------------------------------------------------------------------------*/
void SysTick_Handler(void) {
  msTicks++;
}

/*----------------------------------------------------------------------------
 * Delay: delays a number of Systicks
 *----------------------------------------------------------------------------*/
void Delay (uint32_t dlyTicks) {
  uint32_t curTicks;

  curTicks = msTicks;
  while ((msTicks - curTicks) < dlyTicks) { __NOP(); }
}


/*----------------------------------------------------------------------------
 * main: blink LED and check button state
 *----------------------------------------------------------------------------*/
int main (void) {
  int32_t max_num = LED_GetCount();
  int32_t num = 0;

  LED_Initialize();

  SysTick_Config(SystemCoreClock / 1000);                   // SysTick 1 msec interrupts

  for (;;) {
    LED_On(num);                                            // Turn specified LED on
    Delay(500);                                             // Wait 500ms
    LED_Off(num);                                           // Turn specified LED off
    Delay(500);                                             // Wait 500ms

    num++;                                                  // Change LED number
    if (num >= max_num) {
      num = 0;                                              // Restart with first LED
    }
  }
}
```

Рисунок 1.22 – Программа мигания светодиодом в keil

```c
/*----------------------------------------------------------------------------
 * SystemCoreClockConfigure: configure SystemCoreClock using HSI
                             (HSE is not populated on Nucleo board)
 *----------------------------------------------------------------------------*/
void SystemCoreClockConfigure(void) {

  RCC->CR |= ((uint32_t)RCC_CR_HSION);                     // Enable HSI
  while ((RCC->CR & RCC_CR_HSIRDY) == 0);                  // Wait for HSI Ready

  RCC->CFGR = RCC_CFGR_SW_HSI;                             // HSI is system clock
  while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_HSI);  // Wait for HSI used as system clock

  FLASH->ACR  = FLASH_ACR_PRFTBE;                          // Enable Prefetch Buffer
  FLASH->ACR |= FLASH_ACR_LATENCY;                         // Flash 1 wait state

  RCC->CFGR |= RCC_CFGR_HPRE_DIV1;                         // HCLK = SYSCLK
  RCC->CFGR |= RCC_CFGR_PPRE1_DIV2;                        // APB1 = HCLK/2
  RCC->CFGR |= RCC_CFGR_PPRE2_DIV1;                        // APB2 = HCLK

  RCC->CR &= ~RCC_CR_PLLON;                                // Disable PLL

  //  PLL configuration:  = HSI/2 * 12 = 48 MHz
  RCC->CFGR &= ~(RCC_CFGR_PLLSRC | RCC_CFGR_PLLXTPRE | RCC_CFGR_PLLMULL);
  RCC->CFGR |=  (RCC_CFGR_PLLSRC_HSI_Div2 | RCC_CFGR_PLLMULL12);

  RCC->CR |= RCC_CR_PLLON;                                 // Enable PLL
  while((RCC->CR & RCC_CR_PLLRDY) == 0) __NOP();           // Wait till PLL is ready

  RCC->CFGR &= ~RCC_CFGR_SW;                               // Select PLL as system clock source
  RCC->CFGR |=  RCC_CFGR_SW_PLL;
  while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL);  // Wait till PLL is system clock src
}
```

Рисунок 1.23 – Функция настройки частоты

Вызов данной функции необходимо добавить в функцию main (рис. 1.24). Также необходимо вызвать функцию SystemCoreClockUpdate для того, чтобы обновить параметр SystemCoreClock. Этот параметр будет являться частотой, на основе которой можно организовывать таймер для реализации задержки.

```
int main (void) {
  int32_t max_num = LED_GetCount();
  int32_t num = 0;

  SystemCoreClockConfigure();                          // configure HSI as System Clock
  SystemCoreClockUpdate();

  LED_Initialize();

  SysTick_Config(SystemCoreClock / 1000);              // SysTick 1 msec interrupts

  for (;;) {
    LED_On(num);                                       // Turn specified LED on
    Delay(500);                                        // Wait 500ms
    LED_Off(num);                                      // Turn specified LED off
    Delay(500);                                        // Wait 500ms

    num++;                                             // Change LED number
    if (num >= max_num) {
      num = 0;                                         // Restart with first LED
    }
  }
}
```

Рисунок 1.24 – Вызов функции настройки частоты и обновление SystemCoreClock

Для прошивки и отладки программы необходимо в настройках проектах в разделе Debug выбрать ST-Link Debugger (рис. 1.25). ST-Link – это внутрисхемный программатор-отладчик, позволяющий прошивать исполняемые файлы в микроконтроллер и осуществлять внутрисхемную отладку (debug) кода.

В Nucleo-F103RB встроен программатор-отладчик ST-Link/v2-1, который расположен в верхней части платы (рис. 1.26). Написанную программу необходимо скомпилировать (F7) и загрузить в микроконтроллер (F8) (рис. 1.26, кнопка LOAD).

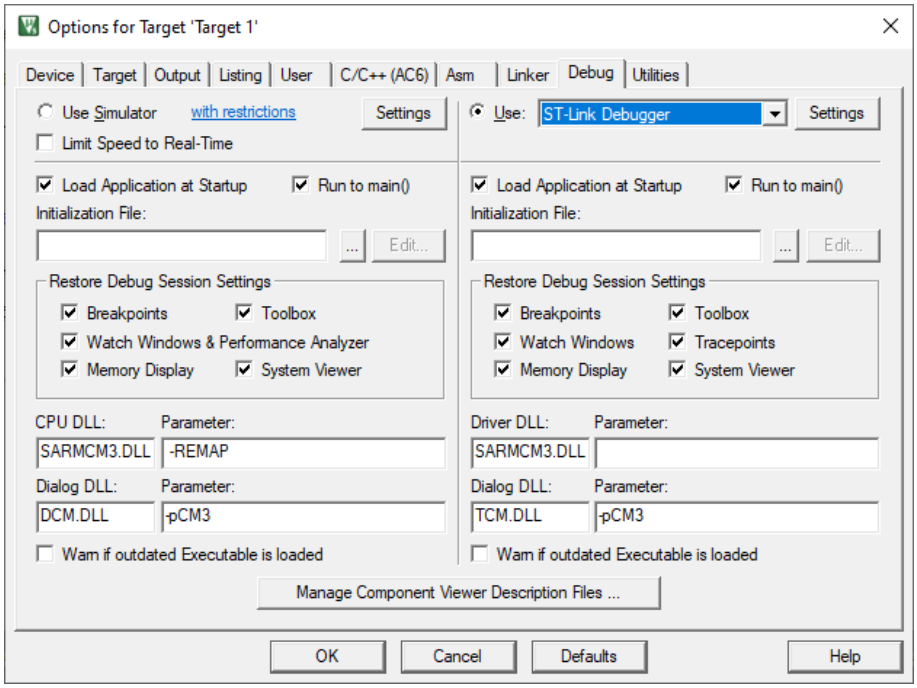


Рисунок 1.25 – Выбор отладчика

Рисунок 1.26 – Прошивка микроконтроллера

Для пошаговой отладки программы можно перейти в Start/Stop Debug Session (Ctrl + F5, Рис. 1.27). Нажимая клавишу F10 можно пройти программу по шагам (без захода в тело функций). Если необходимо зайти в тело функции, можно нажать F11.

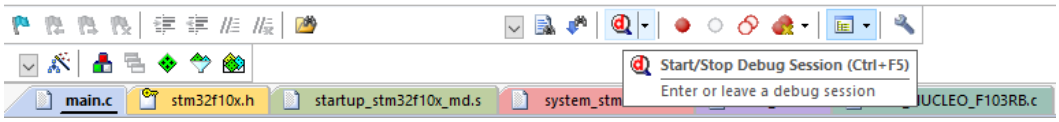


Рисунок 1.27 – Режим отладки

При отладке можно посмотреть значения переменных и содержимое регистров микроконтроллера (рис. 1.28-1.29), что помогает при поиске ошибок и изучении микроконтроллера.

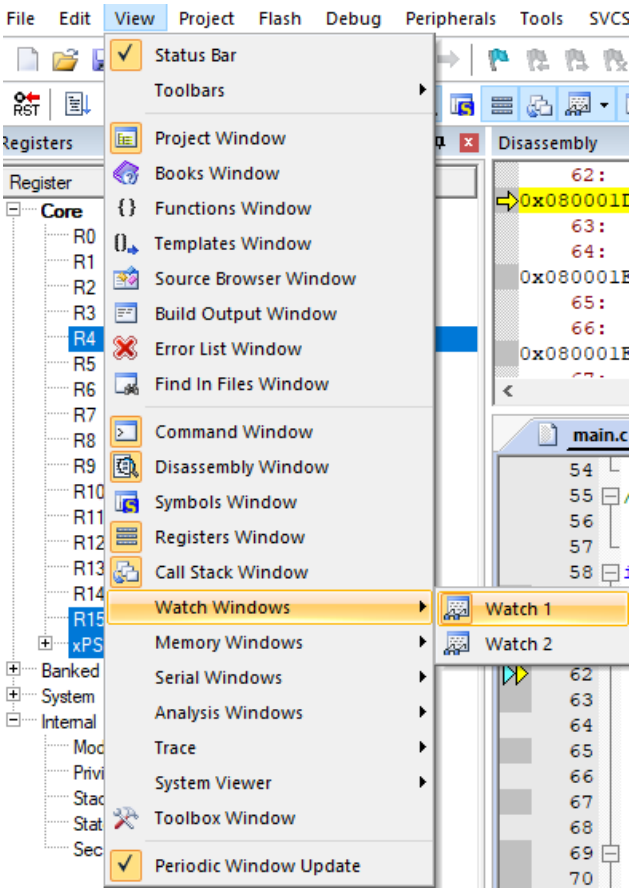


Рисунок 1.28 – Пошаговая отладка программы

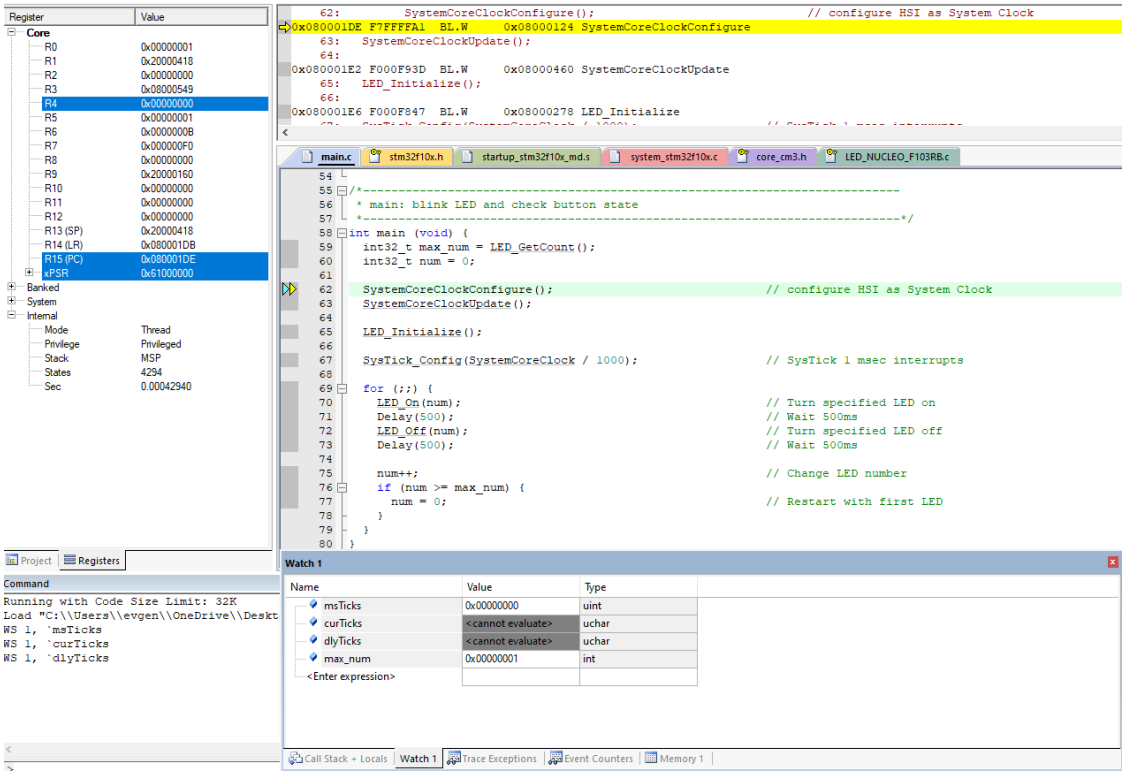


Рисунок 1.29 – Пошаговая отладка программы

1.5 ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Mbed OS.

1.1. Зарегистрироваться на https://os.mbed.com/

1.2. Создать простой проект в mbed для мигания светодиодом, используя шаблонный проект.

1.3. Разобраться в исходном коде программы. Выяснить, что делает каждая строчка программы.

1.4. Скомпилировать программу и загрузить прошивку в микроконтроллер (при наличии отладочной платы).

2. Keil.

2.1. Создать проект в Keil 5 (https://www2.keil.com/mdk5) на основе CMSIS для мигания светодиодом. Исходный код программы можно использовать из примера.

2.2. Разобраться в исходном коде программы. Выяснить, как работает настройка тактовой частоты (SystemCoreClockConfigure), инициализация светодиодов (LED_Initialize) и функция задержки (Delay). При разборе функций использовать reference manual микроконтроллера.

2.3. Скомпилировать программу и загрузить прошивку в микроконтроллер (при наличии отладочной платы).

2.4. Изменить частоту мигания светодиодов (на произвольное значение) путем настройки тактовой частоты (SystemCoreClockConfigure).

2.5. Запустить режим отладки (Start/Stop Debug Session). Вывести в окно отладки Watch 1 значение переменной SystemCoreClock после настройки частоты.

3. Отчётность.

3.1. Для слушателей со своими отладочными платами представить в виде отчёта исходный код и результаты работы программы (снимки платы).

3.2. Для слушателей без отладочных плат представить описание функций настройки тактовой частоты (SystemCoreClockConfigure), инициализации светодиодов (LED_Initialize) и функции задержки (Delay).

2 Практическая работа №2. Работа с интерфейсом I2C в OS Mbed.

2.1 Цель практики

Получить практические навыки работы с интерфейсом I2C на основе OS Mbed. Создать и настроить проект в среде mbed для работы с датчиком температуры. Реализовать запись команд, чтение температуры и запись граничных значений в (из) датчик (а) температуры по интерфейсу I2C.

2.2 Датчик температуры LM75B

В рамках данной работы будет использоваться датчик температуры LM75B, поддерживающий интерфейс I2C. Также, можно использовать любой другой датчик температуры с интерфейсом I2C, однако пример программы и описание работы будет представлен именно для этого датчика.

Полная версия документации к датчику LM75B доступна в прикрепленных файлах раздела «Практика 2. Работа с I2C (OS Mbed)» https://udo.tusur.ru/mod/assign/view.php?id=14674. Структурная схема датчика представлена на рисунке 2.1.

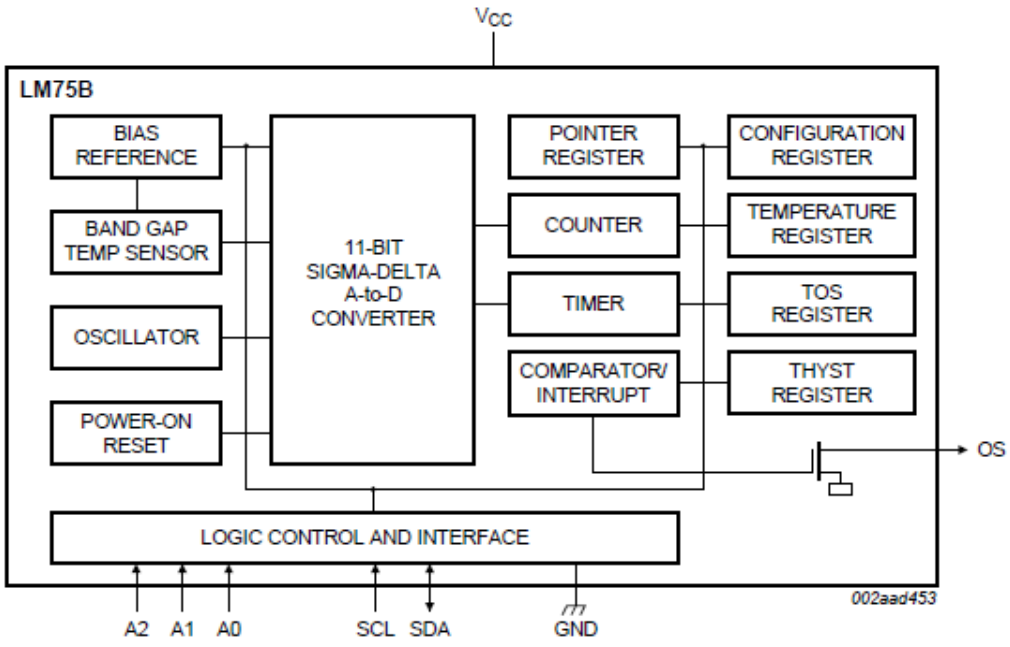


Рисунок 2.1 – Структурная схема датчика LM75B

В состав датчика входит непосредственно измеритель температуры (Band Gap Temp Sensor), 11-разрядный АЦП (11-Bit Sigma-Delta-A-to-D Converter), регистр-указатель для хранения адреса считываемого или записываемого регистров (Pointer Register), регистр настроек (Configuration Register), регистр температуры (Temperature Register), регистр TOS верхнего предела генерации сигнала OS (Overtemperature Shutdown), регистр THYST нижнего предела для отключения сигнала OS (гистерезиса).

Датчик имеет адресные входы A0¬A2, вход тактирования I2C SCL, вход/выход данных I2C SDA, выход сигнала OS для реализации термостата.

Для чтения температуры или записи настроек необходимо отправить адрес датчика, который согласно документации, имеет значение 1001000b (0x48).

Для чтения одного из 4-х регистров датчика или записи в какой-либо регистр (в регистр температуры записывать нельзя) необходимо отправить адрес регистра, согласно таблице 1.1.

Таблица 1.1 – Регистры датчика LM75B

Имя регистра	Адрес регистра	Режим работы	Значение по умолчанию	Описание
Conf	0x01	Чтение/Запись	00	Хранит настройки датчика и задает режим работы
Temp	0x00	Чтение	н/а	Хранит измеренную температуру
Tos	0x03	Чтение/Запись	5000h	Хранит верхний предел для генерации сигнала OS
Thyst	0x02	Чтение/Запись	4B00h	Хранит нижний предел для отключения сигнала OS

Структура регистра температуры представлена на рис. 2. Регистр хранит преобразованное АЦП 11-битное значение температуры, при этом млад-

шие 5 бит не используются. MSByte – старший байт, LSByte – младший байт. Шаг преобразования (точность температуры) составляет 0,125.

MSByte								LSByte							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	X	X	X	X	X

Рисунок 2.2 – Структура регистра температуры

Значения температуры хранятся в дополнительном коде. Конечное значение температуры вычисляется следующим образом:

– Если бит D10 равен 0 (положительное значение), то T = +(Temp data >> 5) * 0.125, где Temp data – значение в регистре.

– Если бит D10 равен 1 (отрицательное значение), то T = – (инверсия(Temp data >> 5) + 1) * 0.125.

Примеры значений температуры в соответствии со значениями в регистре представлены в таблице 2.

Таблица 1.2 – Примеры значений температуры

11-битное значение в доп. коде	**Значение в hex**	**Значение в десятичной системе**	**Конечное значение**
011 1111 1000	3F8	1016	+ 127.000
011 1111 0111	3F7	1015	+ 126.875
011 1111 0001	3F1	1009	+ 126.125
011 1110 1000	3E8	1000	+ 125.000
000 1100 1000	0C8	200	+ 25.000
000 0000 0001	001	1	+ 0.125
000 0000 0000	000	0	0.000
111 1111 1111	7FF	-1	– 0.125
111 0011 1000	738	-200	– 25.000
110 0100 1001	649	-439	– 54.875
110 0100 1000	648	-440	– 55.000

Структура регистров Tos и Thyst представлена на рис. 3. Значения температуры хранятся в старших 9-ти битах регистра. Точность температуры в этом случае составит 0,5.

MSByte								LSByte							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
D8	D7	D6	D5	D4	D3	D2	D1	D0	X	X	X	X	X	X	X

Рисунок 2.3 – Регистр Tos и Thyst

Значения температуры хранятся в дополнительном коде. Конечное значение температуры вычисляется следующим образом:

– Если бит D8 равен 0 (положительное значение), то T = +(Temp data >> 7) * 0. 5, где Temp data – значение в регистре.

– Если бит D8 равен 1 (отрицательное значение), то T = –(инверсия(Temp data >> 7) + 1) * 0.5.

Работа с датчиком по интерфейсу I2C. При программной реализации чтения температуры или записи настроек датчика необходимо соблюдать последовательность команд. На рисунках 2.4-2.7 представлены временные диаграммы, отражающие процедуры записи настроек, чтения и записи температуры.

Для записи настроек в датчик, ведущему (микроконтроллеру) необходимо передать адрес устройства (device address) с командой Write (W), а после подтверждения от ведомого (сигнала ACK), передать адрес регистра настроек (pointer byte) и значение для записи в этот регистр (configuration data byte) (рис. 2.4).

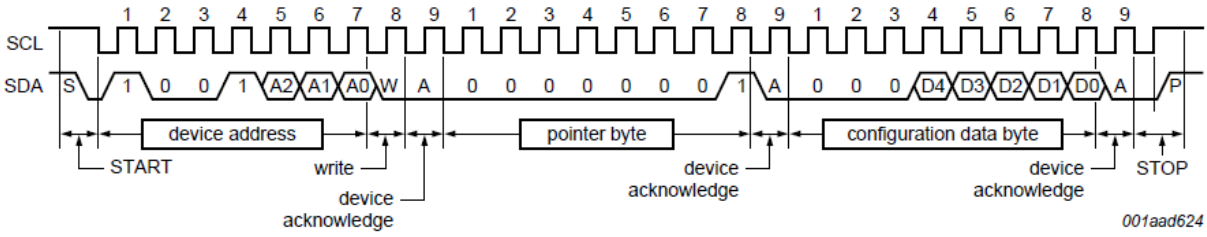


Рисунок 2.4 – Запись настроек в датчик

Для чтения регистра настроек необходимо передать адрес устройства и адрес регистра настроек. После сигнала RS (Restart) снова передать адрес устройства, но уже с командой чтения (R). Ведомое устройство при совпадении адреса передаст ведущему значения регистра настроек (рис. 2.5). Ведущее устройство после приема значения отправляет сигнал NACK, чтобы ведомое не повторяло передачу этого значения.

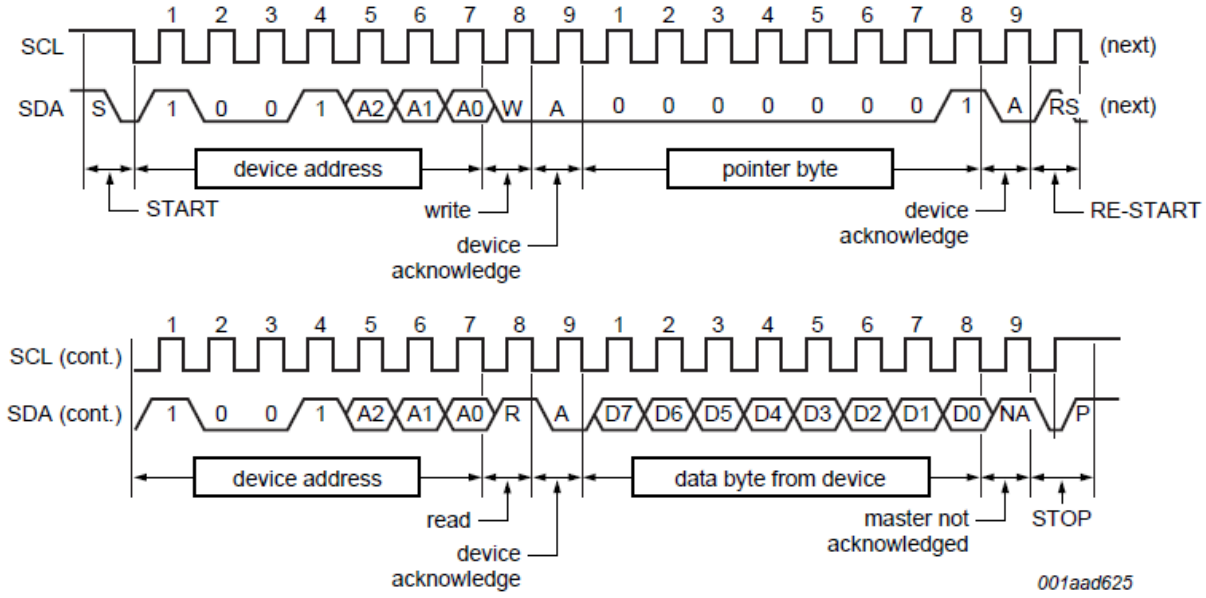


Рисунок 2.5 – Чтение регистра настроек

Для записи значений Tos и Thyst в датчик необходимо сначала передать адреса устройства, адрес регистра Tos или Thyst, а затем два байта значения температуры согласно формату, показанному на рисунке 2.3. Временная диаграмма этого процесса представлена на рис. 2.6.

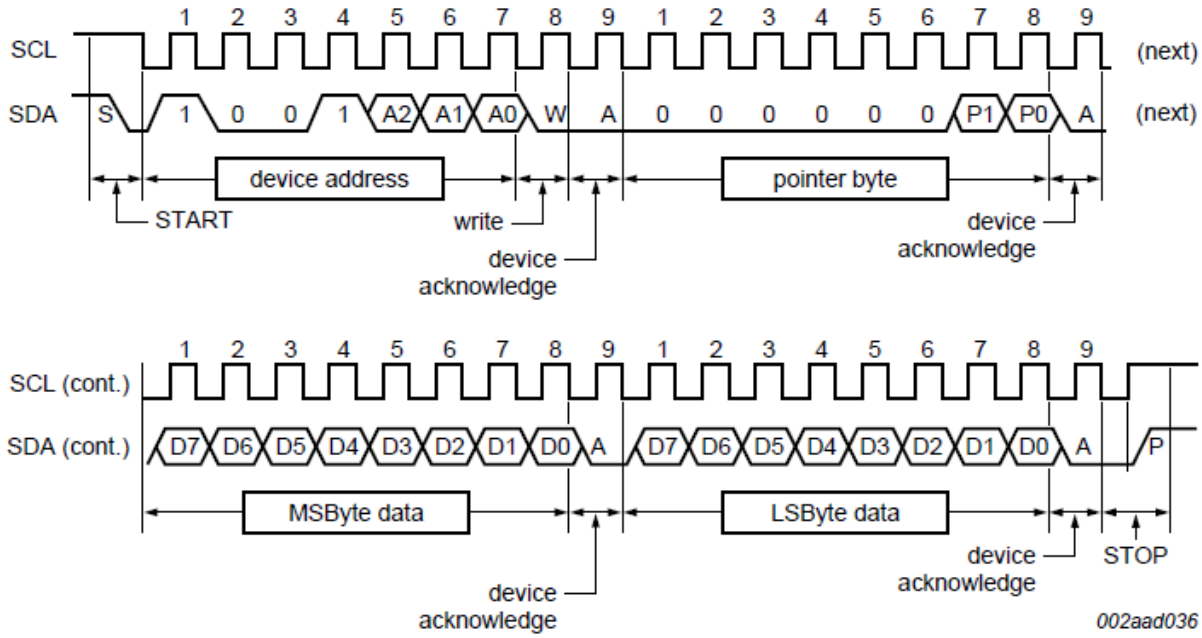


Рисунок 2.6 – Запись значений Tos и Thyst

Для чтения значения температуры необходимо выполнить процедуру, аналогичную для чтения регистра настроек (рис. 2.5), только читать необходимо 2 байта, вместо одного (рис. 2.7).

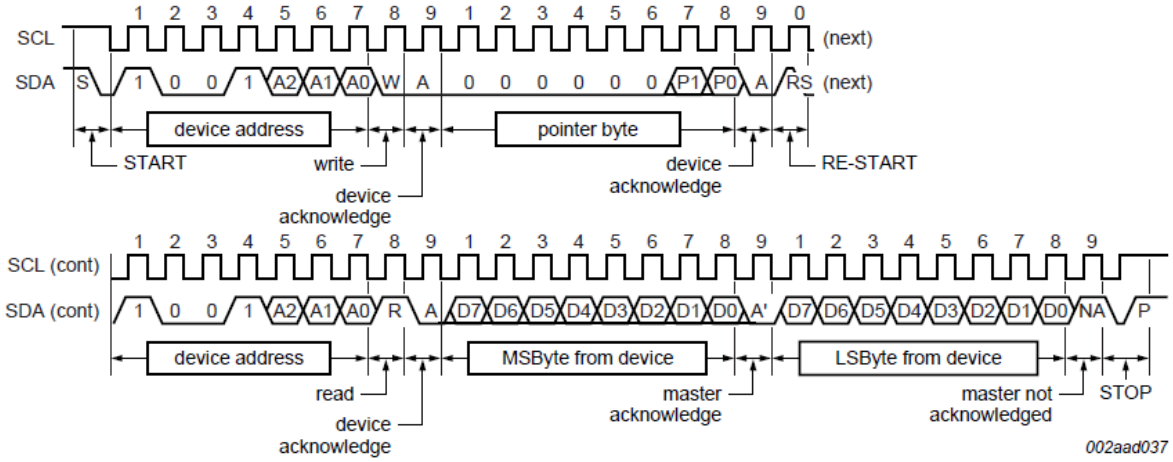


Рисунок 2.7 – Запись регистра температуры

2.3 Реализация в среде Mbed

Среда mbed предоставляет библиотеку на языке C++ с классом для работы с устройствами по интерфейсу I2C. Для работы с функциями (методами) I2C необходимо создать экземпляр класса I2C, указав в качестве пара-

метров названия контактов, которые на отладочной плате соответствуют контактам SCL и SDA интерфейса I2C:

- **I2C i2c (SDA, SCL);**

Для реализации процедуры чтения и записи значений, класс предоставляет методы read и write:

- **int read (int address, char* data, int length, bool repeated);**

int address – адрес устройства, смещенный на 1 разряд влево;

char* data – массив, в который читаются данные;

int length – количество читаемых байт данных;

bool repeated –рестарт, при значении true – не отправлять сигнал stop;

- **int write (int address, const char* data, int length, bool repeated);**

int address – адрес устройства, смещенный на 1 разряд влево;

const char* data – массив для передачи данных;

int length – количество передаваемых байт данных;

bool repeated –рестарт, при значении true – не отправлять сигнал stop;

2.4 ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Используя пример программы, доступный в электронном курсе (Практика 2. Работа с I2C OS Mbed), либо разработав свою программу в среде **Mbed**, реализовать мигание светодиодом при превышении температуры 29 градусов.

Отчётность. Для слушателей со своими отладочными платами представить в виде отчёта исходный код и результаты работы программы (снимки платы).

Для слушателей без отладочных плат представить описание всех функций записи и чтения значений в (из) датчик(а) температуры из примера программы.

3 Практическая работа №3. Работа с интерфейсом I2C с применением STM32CubeMX и HAL

3.1 Цель практики

Изучить принципы программной реализации процедуры взаимодействия с датчиком по протоколу I2C с применением среды CubeMX и библиотеки HAL.

3.2 Создание проекта в STM32CubeMX

STM32CubeMX – среда для генерации кода для микроконтроллеров STM32.

Основная идея программного обеспечения (ПО) STM32CubeMX заключается в общем инструменте для настройки и создании кода инициализации для микроконтроллеров STM32:

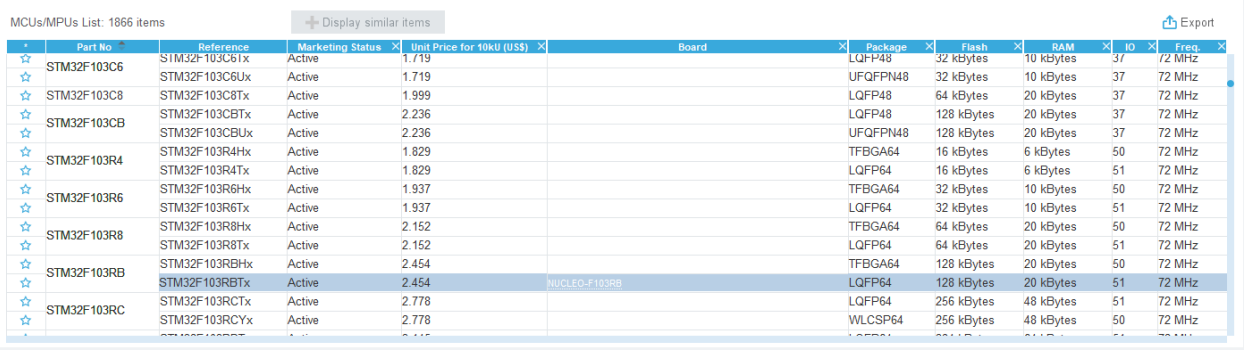
– назначение выводов с автоматическим разрешением конфликтов;

– построение дерева тактирования с динамической проверкой конфигурации;

– инициализация периферии с проверкой параметров на валидность;

– инициализация питания с оценкой результирующего потребления.

Создадим проект в **STM32CubeMX** для работы с интерфейсом I2C. При создании проекта выберем целевое устройство (рис. 3.1).



Part No	Reference	Marketing Status	Unit Price for 10kU (US$)	Board	Package	Flash	RAM	IO	Freq.
STM32F103C6	STM32F103C6Tx	Active	1.719		LQFP48	32 kBytes	10 kBytes	37	72 MHz
	STM32F103C6Ux	Active	1.719		UFQFPN48	32 kBytes	10 kBytes	37	72 MHz
STM32F103C8	STM32F103C8Tx	Active	1.999		LQFP48	64 kBytes	20 kBytes	37	72 MHz
STM32F103CB	STM32F103CBTx	Active	2.236		LQFP48	128 kBytes	20 kBytes	37	72 MHz
	STM32F103CBUx	Active	2.236		UFQFPN48	128 kBytes	20 kBytes	37	72 MHz
STM32F103R4	STM32F103R4Hx	Active	1.829		TFBGA64	16 kBytes	6 kBytes	50	72 MHz
	STM32F103R4Tx	Active	1.829		LQFP64	16 kBytes	6 kBytes	51	72 MHz
STM32F103R6	STM32F103R6Hx	Active	1.937		TFBGA64	32 kBytes	10 kBytes	50	72 MHz
	STM32F103R6Tx	Active	1.937		LQFP64	32 kBytes	10 kBytes	51	72 MHz
STM32F103R8	STM32F103R8Hx	Active	2.152		TFBGA64	64 kBytes	20 kBytes	50	72 MHz
	STM32F103R8Tx	Active	2.152		LQFP64	64 kBytes	20 kBytes	51	72 MHz
STM32F103RB	STM32F103RBHx	Active	2.454		TFBGA64	128 kBytes	20 kBytes	50	72 MHz
	STM32F103RBTx	Active	2.454	NUCLEO-F103RB	LQFP64	128 kBytes	20 kBytes	51	72 MHz
STM32F103RC	STM32F103RCTx	Active	2.778		LQFP64	256 kBytes	48 kBytes	51	72 MHz
	STM32F103RCYx	Active	2.778		WLCSP64	256 kBytes	48 kBytes	50	72 MHz

Рисунок 3.1 – Выбор устройства

После выбора устройства появится рисунок микроконтроллера с именованными контактами (рис. 3.2).

Рисунок 3.2 – Микроконтроллер STM32F103RB в CubeMX

Назначим сигналы SCL и SDA на контакты PB8 и PB9 в соответствии с отладочной платой nucleo-F103RB (рис. 3.3-3.4).

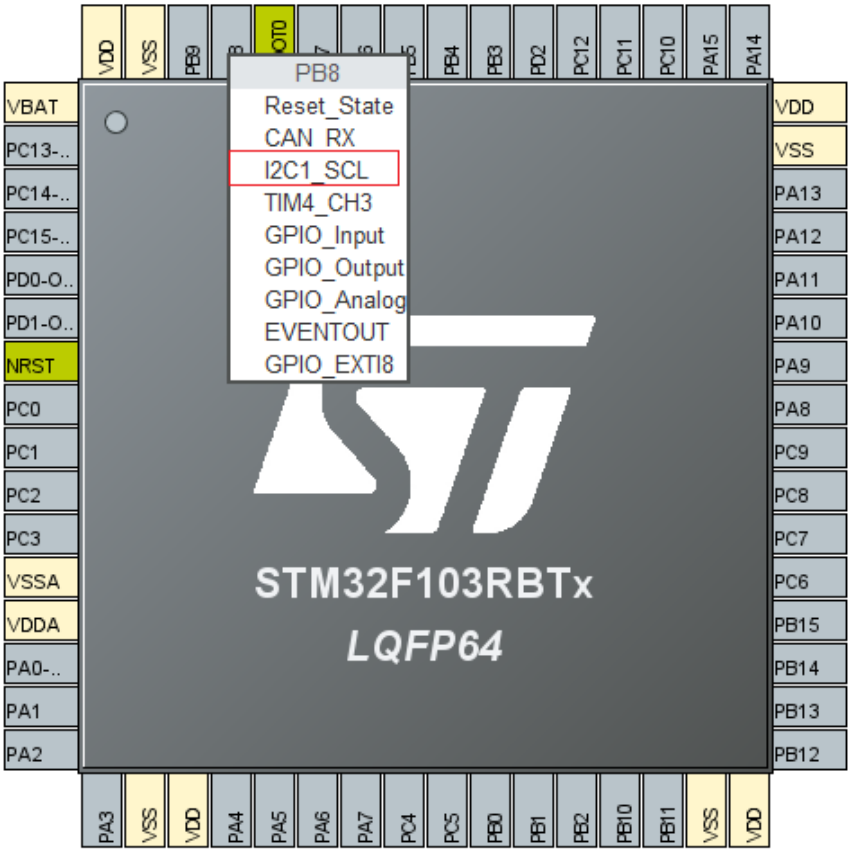


Рисунок 3.3 – Назначение сигнала SCL на PB8

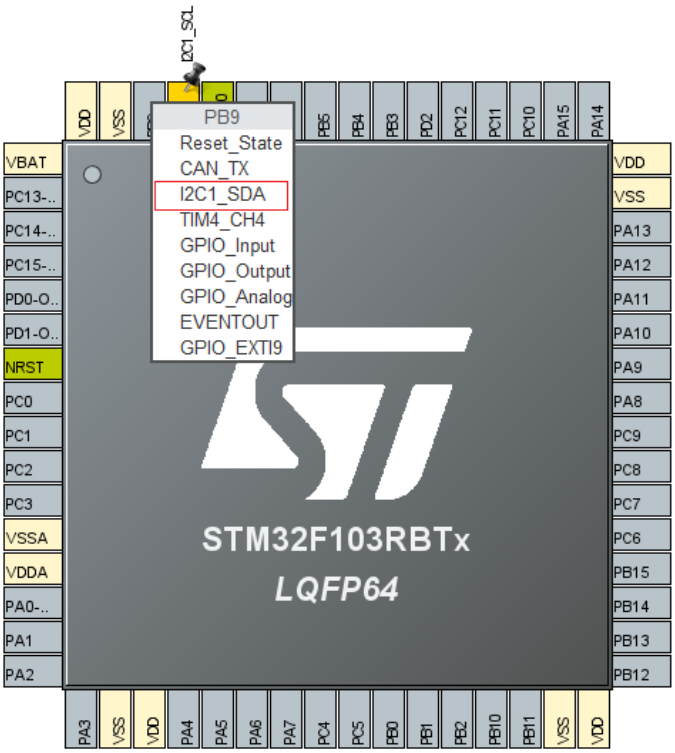


Рисунок 3.4 – Назначение сигнала SDA на PB9

В разделе Connectivity выберем I2C1, зададим режим I2C и увидим, что на контакты PB8 и PB9 назначены сигналы SCL и SDA.

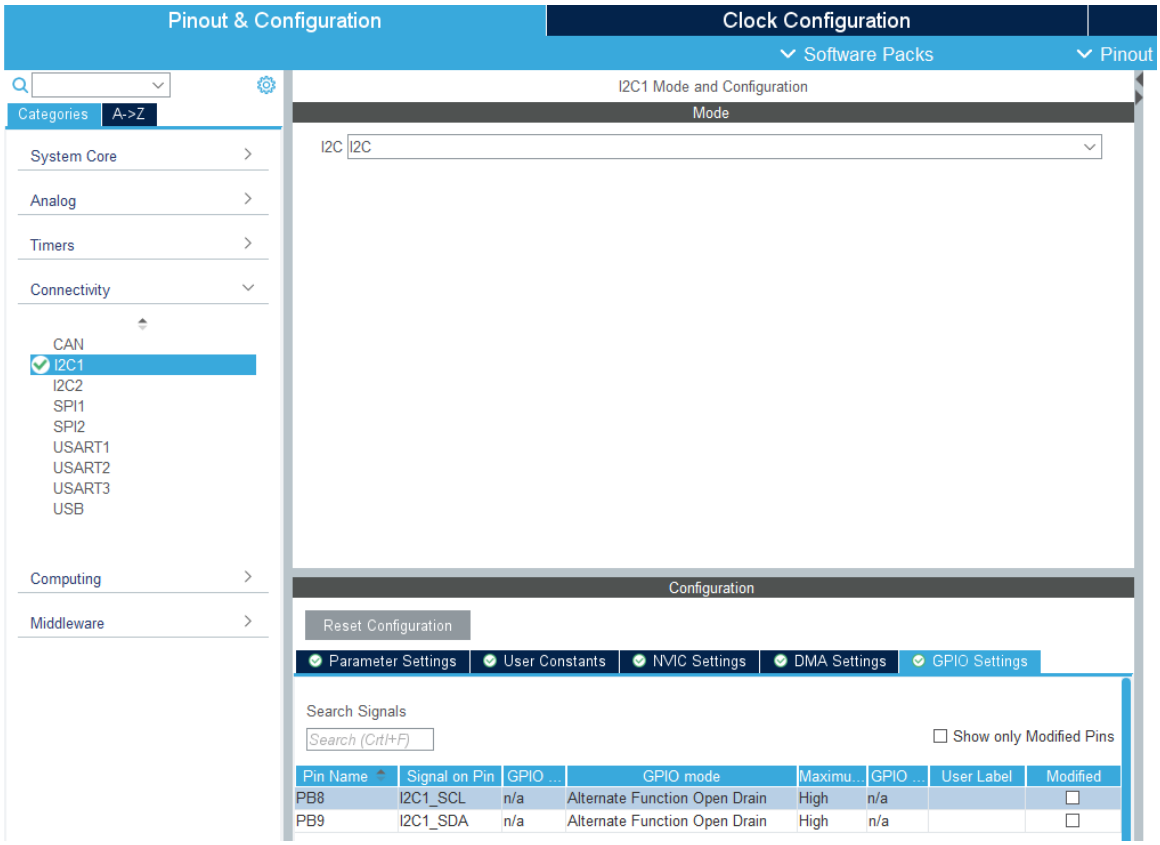


Рисунок 3.5 – Выбор интерфейса I2C

В разделе Clock Configuration оставим частоты тактирования по умолчанию 8 МГц (рис. 3.6).

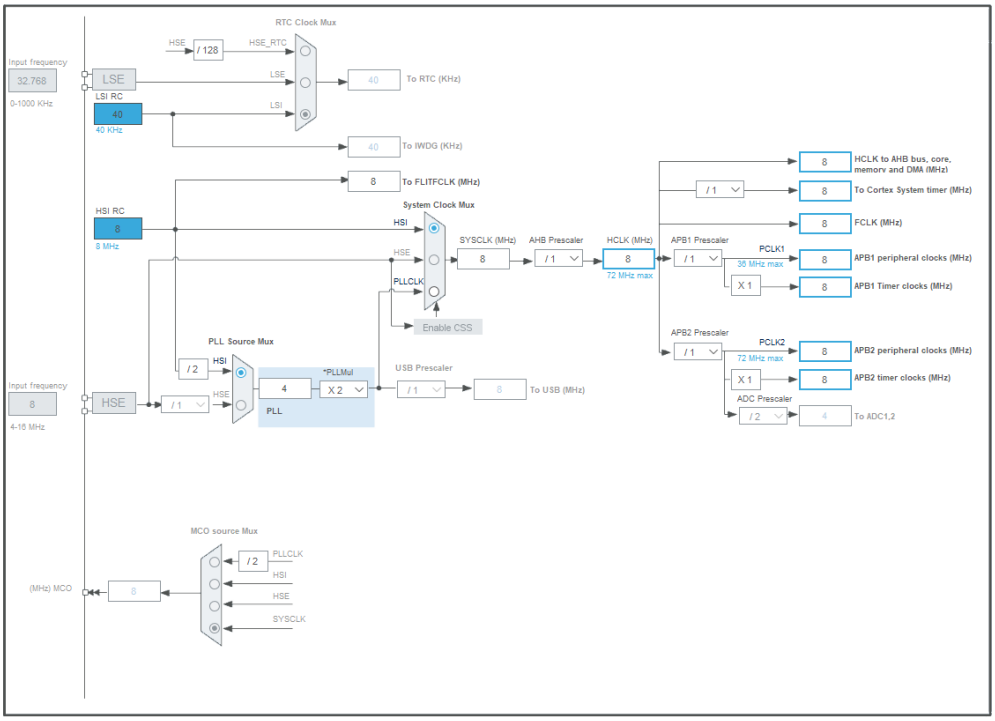


Рисунок 3.6 – Настройка тактирования

Настроим среду разработки, для которой будет генерироваться код (рис. 3.7). Выберем MDK-ARM (Keil 5).

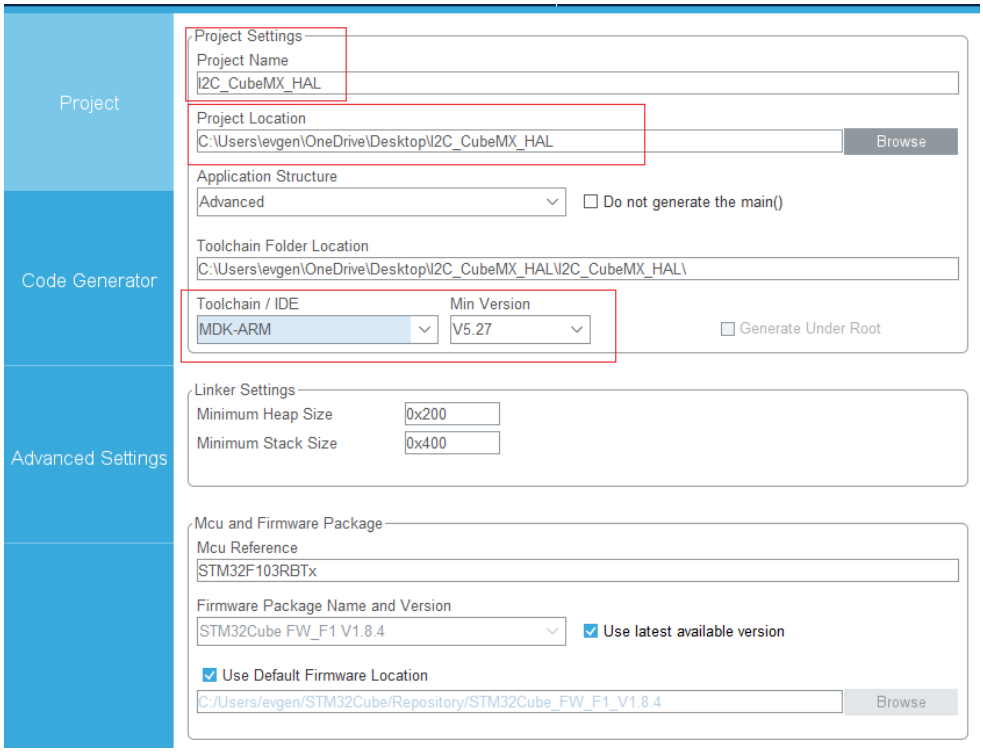


Рисунок 3.7 – Настройки проекта для генерации кода

После завершения настройки нажимаем Generate Code. После генерации кода, будет предложено открыть проект или открыть каталог с проектом. Откроем проект (рис. 3.8).

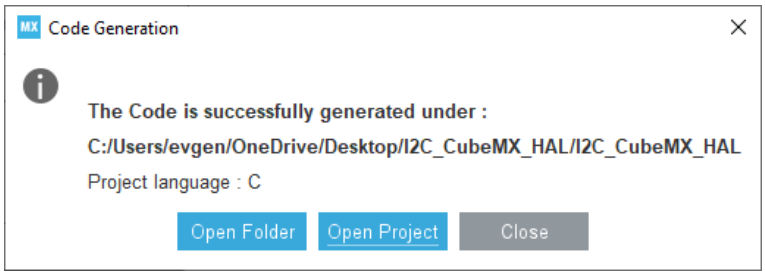


Рисунок 3.8 – Открытие проекта

После генерации проекта перейдем в файл main.c (Application/User/Core). В этом файле необходимо писать основной код программы (рис. 3.9).

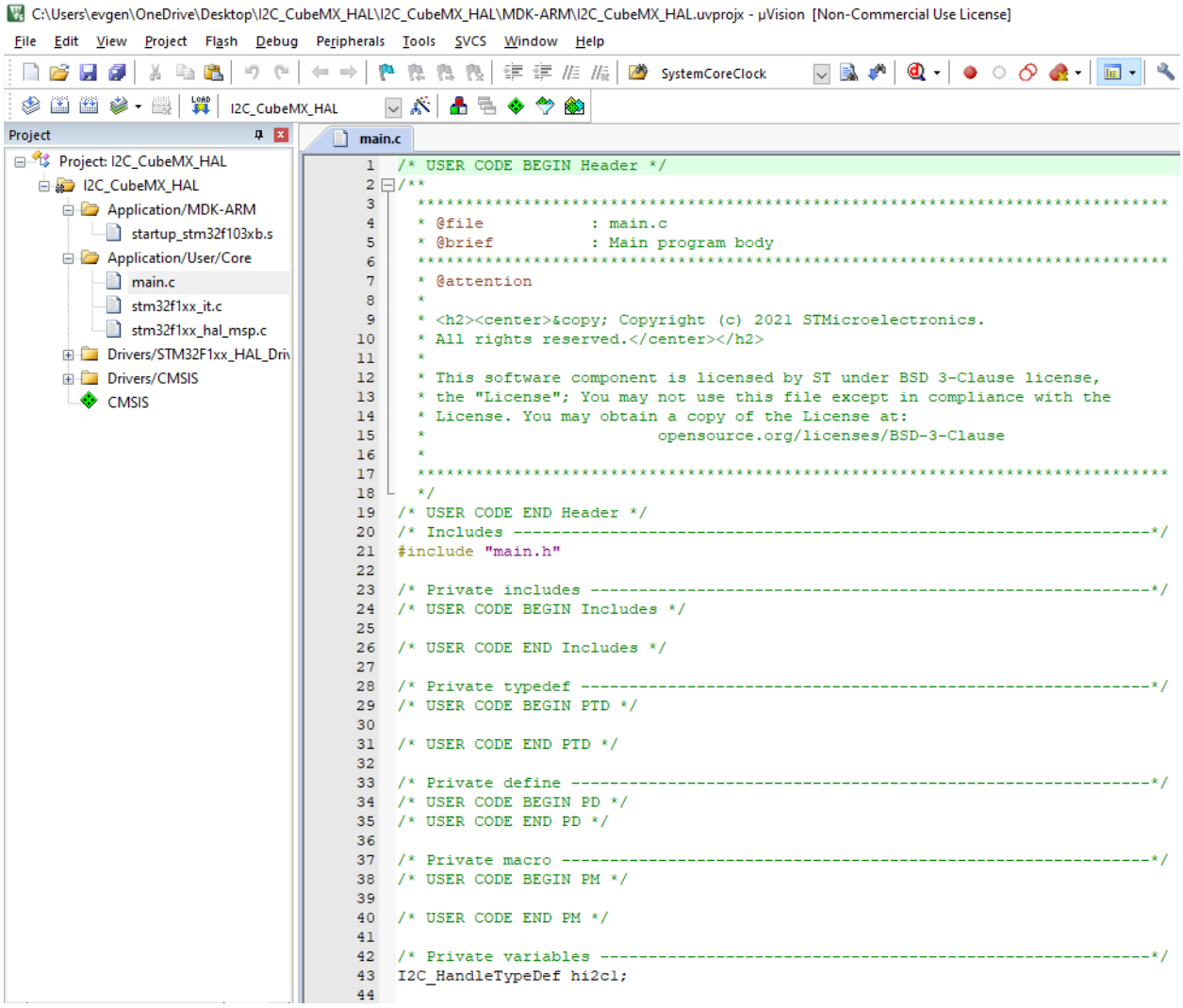


Рисунок 3.9 – Созданный проект в keil

После создания проекта, его необходимо скомпилировать (F7).

По умолчанию код, сгенерированный CubeMX, отключает интерфейс SWD (Serial Wire Debug). Чтобы его включить, в файле stm32f1xx_hal_msp.c необходимо строку __HAL_AFIO_REMAP_SWJ_DISABLE() заменить на __HAL_AFIO_REMAP_SWJ_ENABLE () (рис. 3.10). Если оставить интерфейс отключенным, то могут возникнуть проблемы при попытке прошить программу.

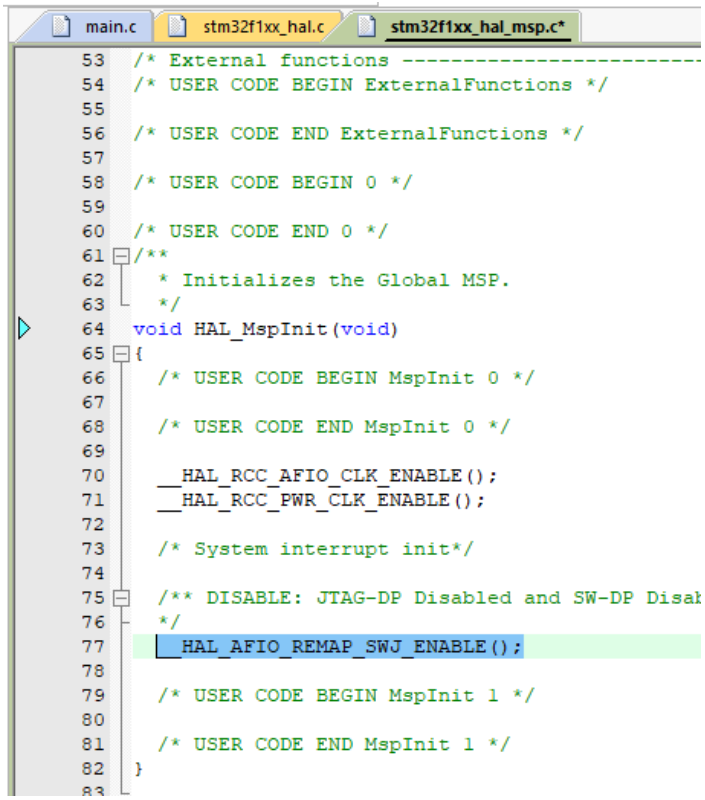


Рисунок 3.10 – Включение SWD

Как видно из функции MX_I2C1_Init, CubeMX автоматически сгенерировал настройки для интерфейса и задал стандартную скорость 100 кГц (рис. 3.11).

```
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}
```

Рисунок 3.11 – Функция инициализации I2C

Пропишем некоторые переменные и константы (адрес датчика, адреса регистров датчика, переменные для передачи значений, рис. 3.12) и добавим 2 функции в основной цикл программы HAL_I2C_Master_Transmit и HAL_I2C_Master_Receive (рис. 3.13).

```
#define    I2C_ADDRESS        0x90 // I2C device address
#define    LM75B_Conf         0x01 // configuration reg address
#define    LM75B_Temp         0x00 // temperature reg address
#define    LM75B_Tos          0x03 // Tos reg address
#define    LM75B_Thyst        0x02 // Thyst reg address
#define    I2C_TIMEOUT          10 // I2C timeout

uint8_t regData [2]; // temp reg data
uint8_t regAddress;  // reg address
```

Рисунок 3.12 – Адреса датчика и регистров

```
while (1)
{
  /* USER CODE END WHILE */
    HAL_I2C_Master_Transmit(&hi2c1, I2C_ADDRESS, &regAddress, 1,  I2C_TIMEOUT); // send temp reg address
    HAL_I2C_Master_Receive(&hi2c1,  I2C_ADDRESS , regData, 2,  I2C_TIMEOUT); // read temperature from reg by regAddress

  /* USER CODE BEGIN 3 */
}
```

Рисунок 3.13 – Функции чтения и записи данных

HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t Timeout):

– I2C_HandleTypeDef *hi2c – структура, хранящая настройки интерфейса I2C;

– uint16_t DevAddress адрес датчика;

– uint8_t *pData – передаваемые данные;

– uint16_t Size – количество передаваемых байт;

– uint32_t Timeout –время тайм-аута в случае возникновения ошибок

– HAL_I2C_Master_Receive(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t Timeout)

– I2C_HandleTypeDef *hi2c – структура, хранящая настройки интерфейса I2C;

– uint16_t DevAddress адрес датчика;

– uint8_t *pData – принимаемые данные;

– uint16_t Size – количество байт для чтения;

– uint32_t Timeout –время тайм-аута в случае возникновения ошибок

3.3 ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Используя пример программы, доступный в электронном курсе (Практика 3. Работа с I2C (CubeMX & HAL)), либо разработав свою программу с использованием CubeMX и HAL:

1. Реализовать запись значений Tos и Thyst в датчик температуры. Предусмотреть проверку записи значений путем их чтения из соответствующих регистров.
2. Реализовать мигание светодиодом при превышении температуры 29 градусов. Задержка задается произвольно.

Отчётность.

Для слушателей со своими отладочными платами представить в виде отчёта исходный код и результаты работы программы (снимки платы).

Для слушателей без отладочных плат представить описание функций MX_I2C1_Init, HAL_I2C_Master_Transmit, HAL_I2C_Master_Receive.

4 Практическая работа №4. Работа с интерфейсом I2C с применением CMSIS

4.1 Цель практики

Получить практические навыки работы с интерфейсом I2C на основе CMSIS (Common Microcontroller Software Interface Standard). Реализовать настройку регистров, частоты тактирования I2C. Написать программу чтения температуры с датчика и записи данных в датчик.

4.2 Теоретические сведения

Структурная схема интерфейса I2C в микроконтроллере STM32F1.

На рисунке 4.1. представлена структурная схема интерфейса I2C в микроконтроллере STM32.

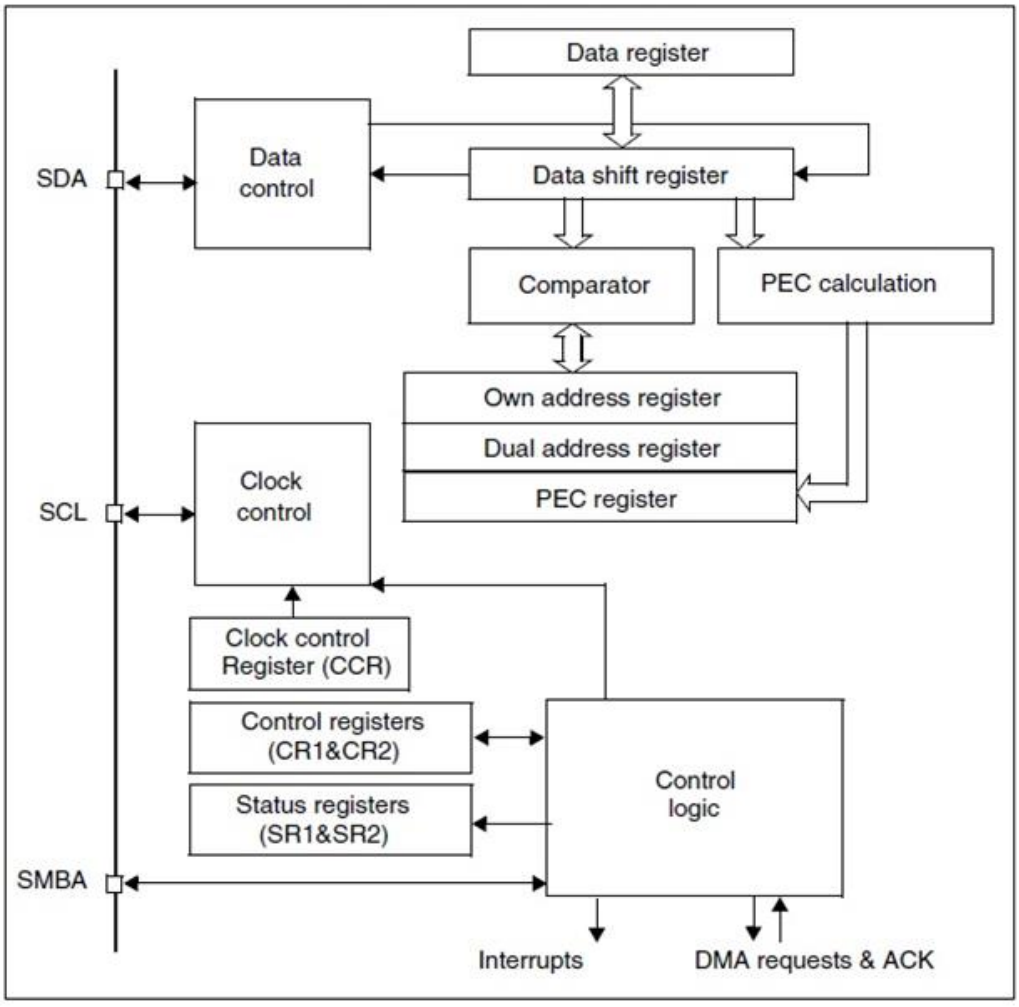
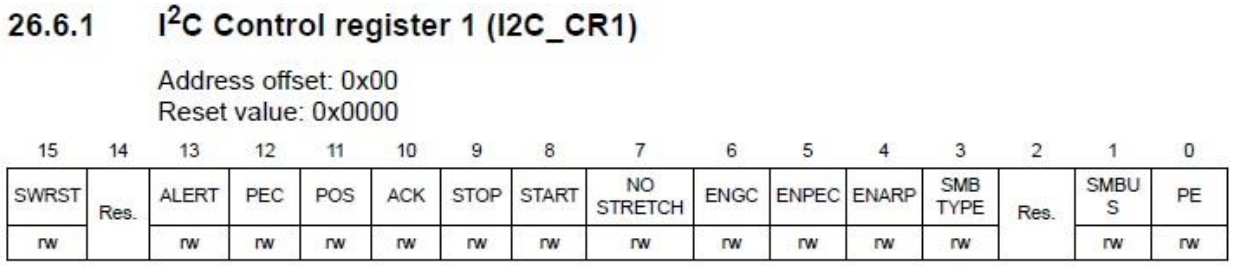


Рисунок 4.1 – Структурная схема интерфейса I2C в STM32F1

Аппаратная реализация I2C включает *регистр данных, адресный регистр, регистр двойного адреса*, *два управляющих регистра — CR1 и CR2*, *регистры статуса — SR1 и SR2*, *регистр, задающий частоту передачи данных или скорость (CCR)*, *регистр PEC (Packet error checking) – регистр отслеживания ошибок*, представляет собой битовое поле, состоящее из 8 бит регистра *SR2* (биты с 8 по 15), в которое записывается контрольная сумма кадра.

Рассмотрим подробнее регистр I2C_CR1 (рис. 4.2).



26.6.1 I²C Control register 1 (I2C_CR1)

Address offset: 0x00
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWRST	Res.	ALERT	PEC	POS	ACK	STOP	START	NO STRETCH	ENGC	ENPEC	ENARP	SMB TYPE	Res.	SMBUS	PE
rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw

Рисунок 4.2 – Регистр I2C_CR1

SWRST (Software reset): бит программного сброса. В случае наличия в нём логической единицы биты регистров шины сбрасываются в состояние по умолчанию. Данный бит может быть использован в случае возникновения ошибок.

ALERT (SMBus alert): бит разрешения генерации сигнала ALERT при работе шины в режиме SMBus. В случае единицы генерация разрешена.

PEC (Packet error checking): бит обнаружения ошибок. Данный бит устанавливается и сбрасывается программно, но также может быть сброшен аппаратно, когда передаётся PEC, или с помощью условия START или STOP, или когда бит PE равен 0:

- 0 – PEC передаётся;
- 1 – PEC не передаётся.

POS (Acknowledge/PEC Position for data reception): бит положения ACK/PEC в режиме MASTER в двухбайтовой конфигурации. Данный бит устанавливается или сбрасывается программно, а также может быть сброшен аппаратно, когда бит PE равен 0.

- 0 – бит ACK управляет ACK (NACK) текущего байта, получаемого в регистре сдвига. Бит PEC указывает, что текущий байт в регистре сдвига является PEC;
- 1 – бит ACK управляет ACK (NACK) следующего байта, который будет принят в регистре сдвига. Бит PEC указывает, что следующий байт в регистре сдвига является PEC.

ACK (Acknowledge enable): бит разрешения отправки ACK (NACK) после приёма байта данных или адреса. Устанавливается или сбрасывается программно, а также может быть сброшен аппаратно, когда бит PE равен 0.

- 0 – принятый байт не завершается условием ACK (NACK);
- 1 – принятый байт завершается условием ACK (NACK).

STOP (Stop generation): бит генерации условия STOP.

В режиме MASTER:

- 0 – условие STOP не генерируется;
- 1 – генерация STOP после передачи текущего байта или после отправки текущего условия START.

В режиме SLAVE:

- 0 – условие STOP не генерируется;
- 1 – линии SDA и SCL опустятся в 0 после передачи текущего байта.

START (Start generation): бит генерации условия START.

В режиме MASTER:

- 0 – условие START не генерируется;
- 1 – генерация условия START.

В режиме SLAVE:

- 0 – условие START не генерируется;
- 1 – Генерация условия START в случае, если шина I2C свободна.

NOSTRETCH (Clock stretching disable): данный бит используется для отключения увеличения времени в режиме SLAVE, когда установлен флаг

ADDR или BTF, до тех пор, пока он не будет сброшен программным обеспечением:

- 0 – передача от MASTER останавливается. Данная возможность используется в том случае, когда SLAVE требуется какое-то время на обработку данных. Ножка SCL прижимается к земле и MASTER будет ждать и не будет ничего посылать до тех пор, пока линия не будет отпущена;
- 1 – отложенный приём отключен.

ENGC (General call enable): бит работы с широковещательным запросом (адрес 0x00):

- 0 – обработка широковещательных запросов отключена. На обращение по адресу устройства 0x00 генерируется условие NACK.
- 1 – обработка широковещательных запросов включена. На обращение по адресу устройства 0x00 генерируется условие ACK.

ENPEC (PEC enable): бит включения аппаратного подсчёта CRC:

- 0 – аппаратный подсчёт CRC отключен.
- 1 – аппаратный подсчёт CRC включен.

ENARP (ARP enable, Address Resolution Protocol): бит включения ARP в режиме SMBus:

- 0 – ARP отключен.
- 1 – ARP включен.

SMBTYPE (SMBus type): бит типа устройства в режиме SMBus:

- 0 – Device.
- 1 – Host.

SMBUS (SMBus mode, System Management Bus mode): режим работы шины:

- 0 – I2C.
- 1 – SMBus.

PE (Peripheral enable): бит включения шины:

- 0 – периферия I2C отключена

- 1 – периферия включена.

Рассмотрим регистр I2C_CR2 (рис. 4.3).

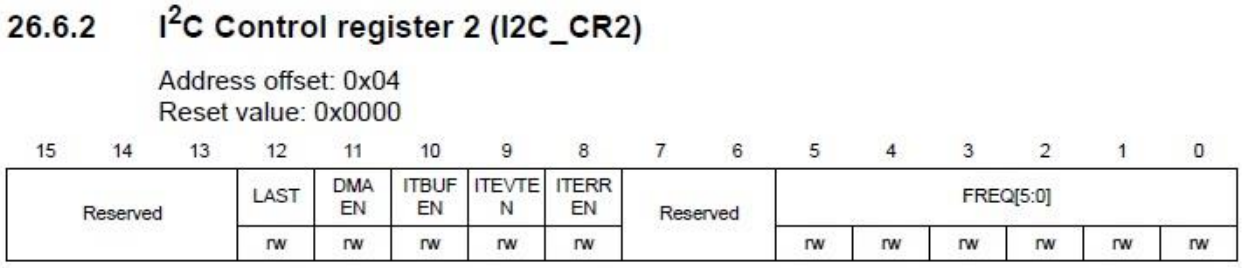


Рисунок 4.3 – Регистр I2C_CR2

LAST (DMA last transfer): бит разрешения генерации периферией DMA сигнала окончания передачи (EOF):

- 0 – генерация окончания передачи DMA запрещена.
- 1 – генерация окончания передачи DMA разрешена.

DMAEN (DMA requests enable): бит разрешения запросов к периферии DMA:

- 0 – запосы к DMA отключены.
- 1 – запосы к DMA включены, когда любой из флагов TxE или RxNE равен 1.

ITBUFEN (Buffer interrupt enable):

- 0 – Установка флагов TxE или RxNE не генерирует прерывания.
- 1 – Установка флагов TxE или RxNE генерирует прерывание по данному событию (независимо от состояния бита DMAEN).

ITEVTEN (Event interrupt enable): бит разрешения прерываний по событию:

- 0 – прерывания по событию запрещены.
- 1 – прерывания по событию разрешены.

ITERREN (Error interrupt enable): бит разрешения прерываний при возникновении ошибок:

- 0 – прерывания при возникновении ошибок запрещены.

- 1 – прерывания при возникновении ошибок разрешены.

FREQ[5:0] (Peripheral clock frequency): битовое поле для задания частоты тактирования APB (Advanced Peripheral Bus). В данное поле заносится число от 2 до 50 (МГц):

- 0 – не разрешено;
- 1 – не разрешено;
- 2 – 2 МГц;
- …….;
- 50 – 50 Мгц.

Рассмотрим регистр I2C_OAR1 (рис. 4.4).

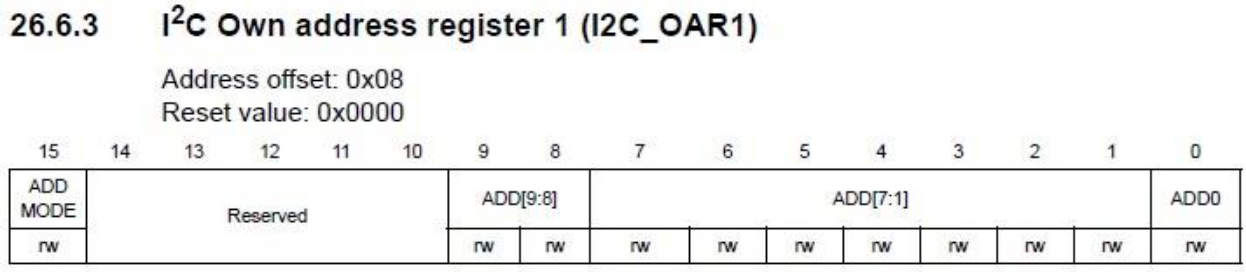


Рисунок 4.4 – Регистр I2C_OAR1

ADDMODE (Addressing mode): режим адресации в режиме SLAVE:

- 0 – 7-разрядный режим адресации (на 10-разрядный адрес не откликается).
- 1 – 10-разрядный режим адресации (на 10-разрядный адрес откликается).

ADD[9:8] (Interface address): 9 и 8 биты адреса устройства в режиме 10-разрядной адресации.

ADD[7:1] (Interface address): 7:1 биты адреса устройства.

ADD0 (Interface address): 0 бит адреса устройства в режиме 10-битной адресации.

Далее рассмотрим регистр I2C_OAR2 (рис. 4.5).

26.6.4 I²C Own address register 2 (I2C_OAR2)

Address offset: 0x0C
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								ADD2[7:1]							ENDUAL
								rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 4.5 – Регистр I2C_ OAR2

ADD2[7:1] (Interface address): биты 7:1 альтернативного адреса в режиме двойной адресации.

ENDUAL (Dual addressing mode enable): бит включения режима двойной адресации:

- 0 – режим двойной адресации выключен
- 1 – режим двойной адресации включен. Устройство откликается на альтернативный адрес.

Также рассмотрим регистр I2C_DR (рис. 4.6).

26.6.5 I²C Data register (I2C_DR)

Address offset: 0x10
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								DR[7:0]							
								rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 4.6 – Регистр I2C_ DR

В случае приёма в данный регистр записывается принятый байт. В случае передачи пишется байт, который нужно передать.

Рассмотрим регистр I2C_SR1 (рис. 4.7).

26.6.6 I²C Status register 1 (I2C_SR1)

Address offset: 0x14
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res.	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res.	STOPF	ADD10	BTF	ADDR	SB
rc_w0	rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	r	r		r	r	r	r	r

Рисунок 4.7 – Регистр I2C_ SR1

SMBALERT (SMBus alert): флаг события ALERT в режиме SMBus.

TIMEOUT (Timeout or Tlow error): флаг события при продолжительном времени нахождения ножки SCL прижатой к земле. В случае MASTER – 10 милисекунд, SLAVE – 25 милисекунд:

- 0 – таймаут не истёк;
- 1 – таймаут истёк.

PECERR (PEC Error in reception): флаг ошибки PEC при приёме:

- 0 – нет ошибки;
- 1 – обнаружена ошибка PEC.

OVR (Overrun/Underrun): флаг переполнения данных:

- 0 – переполнения не было;
- 1 – произошло событие переполнения данных.

AF (Acknowledge failure): отсутствие подтверждения ACK:

- 0 – нормальное подтверждение (ACK);
- 1 – отсутствие подтверждения (NACK). Для сброса нужно записать 0.

ARLO (Arbitration lost): бит потери арбитража в режиме MASTER

- 0 – потери арбитража не обнаружено;
- 1 – потеря арбитража. Для сброса нужно записать 0.

BERR (Bus error): флаг ошибки шины в плане установки условия START или STOP на шине в ненужный момент:

- 0 – ошибки не было;
- 1 – обнаружена ошибка.

TxE (Data register empty): флаг очистки регистра данных в режиме передачи. Устанавливается, когда из регистра данных DR данные переместятся в сдвиговый регистр:

- 0 – регистр данных не пустой;
- 1 – регистр данных пуст.

RxNE (Data register not empty): флаг очистки регистра данных в режиме приёма. Устанавливается, когда данные появились в регистре:

- 0 – регистр данных пуст;
- 1 – регистр данных не пустой.

STOPF (Stop detection): флаг появления условия STOP на шине в режиме SLAVE. Для сброса нужно прочитать регистр SR1 и произвести запись в CR1:

- 0 – обнаружено условие STOP;
- 1 – условие STOP не обнаружено.

ADD10 (10-bit header sent): флаг отправки 10-битного адреса в режиме MASTER:

- 0 – событие отправки 10-битного адреса не обнаружено;
- 1 – произошла отправка первого байта 10-битного адреса.

BTF (Byte transfer finished): флаг окончания приёма/передачи байта. Работает только при NOSTRETCH=0

- 0 – событие окончания приёма/передачи байта не происходило
- 1 – произошло событие окончания приёма/передачи байта.

ADDR (Address sent): Устанавливается после передачи адреса в режиме MASTER, а в режиме SLAVE – при совпадении пришедшего от MASTER адреса с собственным. Для сброса надо сначала прочитать регистр SR1, а потом и SR2.

SB (Start bit): Флаг возникновения сигнала START в режиме MASTER:

- 0 – условие START не обнаружено;
- 1 – обнаружено условие START.

Рассмотрим регистр I2C_SR2 (рис. 4.8).

26.6.7 I²C Status register 2 (I2C_SR2)

Address offset: 0x18
Reset value: 0x0000

Note: Reading I2C_SR2 after reading I2C_SR1 clears the ADDR flag, even if the ADDR flag was set after reading I2C_SR1. Consequently, I2C_SR2 must be read only when ADDR is found set in I2C_SR1 or when the STOPF bit is cleared.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PEC[7:0]								DUALF	SMB HOST	SMBDE FAULT	GEN CALL	Res.	TRA	BUSY	MSL
r	r	r	r	r	r	r	r	r	r	r	r		r	r	r

Рисунок 4.8 – Регистр I2C_ SR2

PEC[7:0] (Packet error checking register): битовое поле, в которое записывается контрольная сумма кадра при ENPEC=1.

DUALF (Dual flag): флаг обнаружения альтернативного адреса на шине в режиме SLAVE:

- 0 – альтернативный адрес не обнаружен;
- 1 – произошло совпадение запрошенного адреса с собственным альтернативным.

SMBHOST (SMBus host header): флаг приёма заголовка SMBus Host в режиме Device:

- 0 – заголовок не принимался;
- 1 – принят заголовок.

SMBDEFAULT (SMBus device default address): бит приёма адреса по умолчанию в режиме Device для SMBus-устройства:

- 0 – адрес по умолчанию не принимался;
- 1 – принят адрес по умолчанию.

GENCALL (General call address): флаг приёма широковещательного адреса в режиме SLAVE:

- 0 – широковещательный адрес не принимался;
- 1 – принят широковещательный адрес.

TRA (Transmitter/receiver): режим работы (приёмник/передатчик)

- 0 – данные приняты;

- 1 – данные отправлены.

BUSY (Bus busy): флаг занятости шины:

- 0 – шина свободна
- 1 – шина занята.

MSL (Master/slave): режим работы (MASTER/SLAVE)

- 0 – шина работает в режиме SLAVE;
- 1 – шина работает в режиме MASTER, включается аппаратно, когда модуль переводится в режим MASTER при помощи установки бита SB, сбрасывается тоже аппаратно.

Далее рассмотрим регистр I2C_CCR (Clock Control Register).

26.6.8 I²C Clock control register (I2C_CCR)

Address offset: 0x1C
Reset value: 0x0000

Note: *f_{PCLK1} must be at least 2 MHz to achieve Sm mode I²C frequencies. It must be at least 4 MHz to achieve Fm mode I²C frequencies. It must be a multiple of 10MHz to reach the 400 kHz maximum I²C Fm mode clock.*

The CCR register must be configured only when the I2C is disabled (PE = 0).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F/S	DUTY	Reserved		CCR[11:0]											
rw	rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 4.9 – Регистр I2C_ CCR

F/S (I2C master mode selection): бит установки скоростного режима

- 0 – режим Standard;
- 1 – режим Fast.

DUTY (Fm mode duty cycle): Данный бит устанавливает скважность шины в режиме Fast:

- 0 – отношение времени низкого состояния к времени высокого состояния 2;
- 1 – отношение времени низкого состояния к времени высокого состояния 16/9.

CCR[11:0] (Clock control register in Fm/Sm mode): битовое поле частоты работы шины в режиме MASTER для режимов Standard и Fast.

Время высокого и низкого состояния ножки SCL рассчитывается следующим образом:

В режиме Standard:

Thigh = Tlow = CCR * TPCLK1

В режиме Fast

Если DUTY=0:

Thigh = CCR * TPCLK1

Tlow = 2 * CCR * TPCLK1

Если DUTY=1:

Thigh = 9 * CCR * TPCLK1

Tlow = 16 * CCR * TPCLK1.

Рассмотрим последний регистр I2C_TRISE (рис. 4.10).

TRISE[5:0] (Maximum rise time in Fm/Sm mode): значение для расчёта времени нарастания фронта. Данное значение надо рассчитать по формуле: **T = TrMAX / TPCLK1 + 1** (TrMAX (SM) = 1000 нс, TrMAX (FM) – 300 нс).

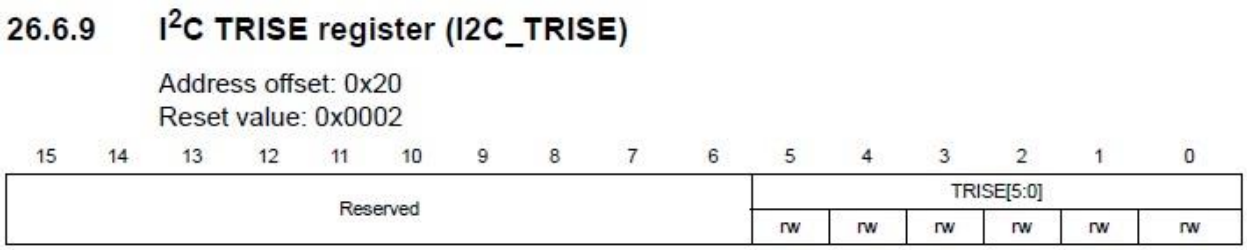


Рисунок 4.10 – Регистр I2C_ TRISE

4.3 ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. На основе примера программы, представленного в разделе Практика 4. Работа с I2C (CMSIS), настроить частоту передачи интерфейса I2C на 50кГц, 25 кГц, 10 кГц, двумя способами: используя регистры управления частотой (RCC_CR, RCC_CFGR) и регистр I2C_CCR. Проверить работоспособность чтения температуры с датчика.

2. Добавить в программу запись значений Tos и Thyst в регистры датчика температуры. Предусмотреть проверку записи значений путем их чтения из соответствующих регистров.

3. Реализовать мигание светодиодом при превышении температуры 29 градусов. Задержка задается произвольно.

Отчётность.

Для слушателей со своими отладочными платами представить в виде отчёта исходный код и результаты работы программы (снимки платы).

Для слушателей без отладочных плат представить описание функций SystemCoreClockConfigure, I2C_Init, I2C_WriteData, I2C_ReadData.

5 Практическая работа №5. Работа с цифровым датчиком по интерфейсу 1-Wire

5.1 Цель работы

Получить практические навыки работы с интерфейсом 1-Wire на основе CMSIS (Common Microcontroller Software Interface Standard). Реализовать функции записи команд и чтения данных по интерфейсу 1-Wire с датчика температуры DS18B20.

Постановка задачи:

1. Разработать библиотеку для работы с датчиком DS18B20. Для работы с датчиком также использовать ранее разработанные библиотеки в предыдущих лабораторных.

2. Вывести в отладчике Keil температуру с 2-х датчиков, а также их значения TH и TL.

3. Изменить разрядность АЦП одного из датчиков с 12 на 9 бит, а также его значения TH и TL на произвольные путем обновления регистра настроек.

4. Реализовать команду Copy Scratchpad для одного из датчиков DS18B20. Проверить правильность работы команды Copy Scratchpad после сброса питания (записанные значения TH и TL и настройки должны сохраниться в энергонезависимой EEPROM).

5. Реализовать мигание светодиодом при превышении температуры 29 градусов любого из датчиков. Задержка задается системным или периферийным таймером.

Содержание отчета

1) Цель работы;

2) Постановка задачи;

3) Листинг программ(ы);

4) Результаты работы программы;

5) Выводы.

5.2 Методические указания

Полная версия документации к датчику DS18B20 доступна в файлах лабораторной работы.

DS18B20 – цифровой датчик температуры, поддерживающий 9–12-битное преобразование температуры. Структурная схема датчика показана на рисунке 5.1.

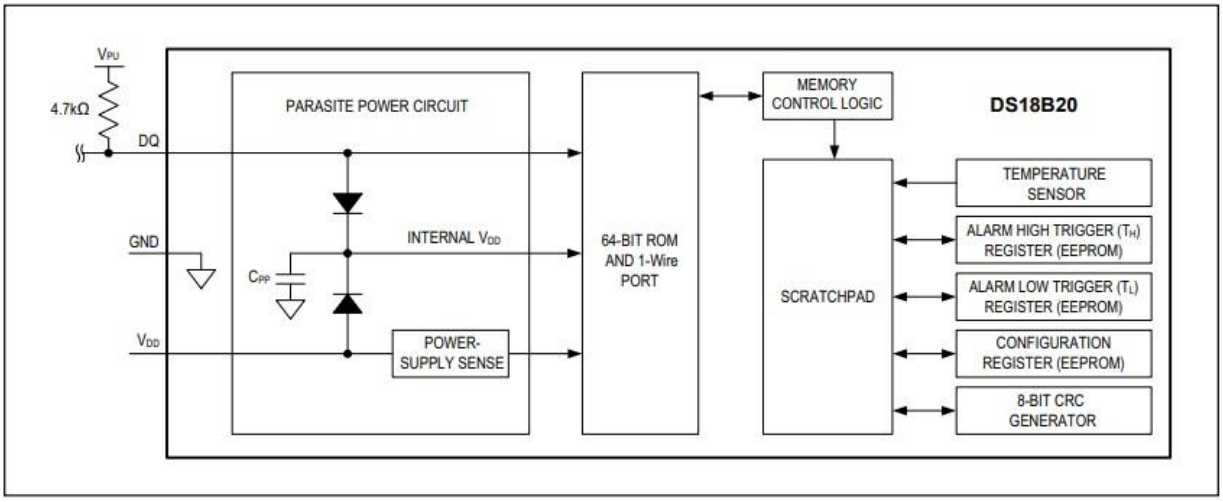


Рисунок 5.1 – Структурная схема DS18B20

В состав датчика входит схема питания (Parasite Power Circuit), 64-битная память ROM для хранения кода (адреса) устройства, схема управления (Memory Control Logic), память для хранения данных (Scratchpad): температуры, настроек, граничных значений температуры, контрольной суммы CRC8; измеритель температуры (Temperature Sensor), модуль расчета CRC8 (8-BIT CRC Generator), регистры (EEPROM) для сохранения граничных значений температуры (Alarm High Trigger и Alarm Low Trigger), регистр для сохранения настроек (Configuration Register).

Память для хранения данных (Scratchpad).

Для хранения температуры и настроек используется память размером 9 байт, структура которой представлена на рисунке 5.2.

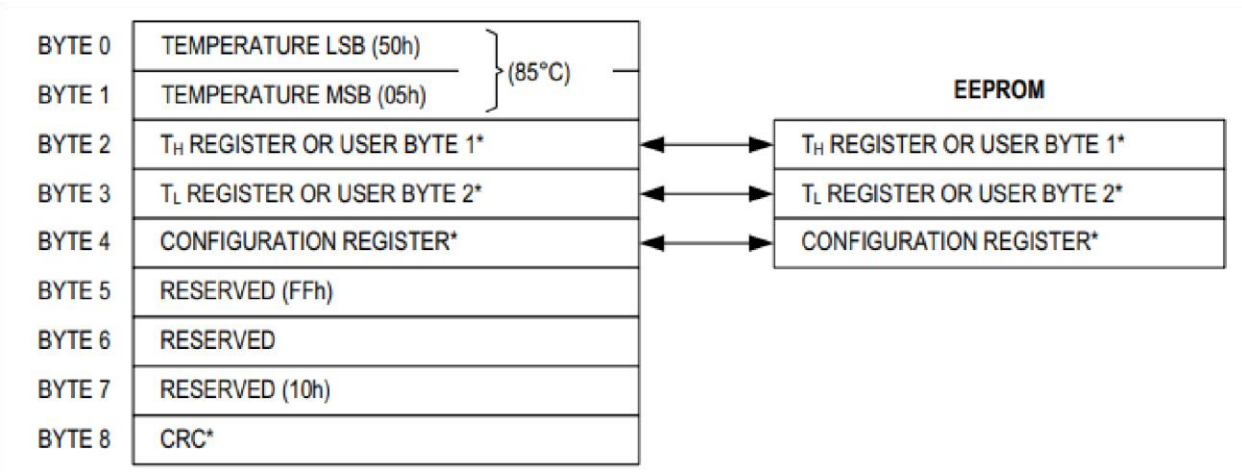


Рисунок 5.2 – Память для хранения температуры и настроек (Scratchpad)

Рассмотрим структуру основных регистров.

Температура хранится в младших двух байтах памяти данных. Формат регистра температуры представлена на рисунке 5.3. Старшие 5 бит отводятся под знак, остальные 12 бит для кодирования температуры.

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
LS BYTE	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
MS BYTE	S	S	S	S	S	2^6	2^5	2^4

S = SIGN

Рисунок 5.3 – Формат регистра температуры

Примеры значений температуры представлены на рисунке 5.4.

TEMPERATURE (°C)	DIGITAL OUTPUT (BINARY)	DIGITAL OUTPUT (HEX)
+125	0000 0111 1101 0000	07D0h
+85*	0000 0101 0101 0000	0550h
+25.0625	0000 0001 1001 0001	0191h
+10.125	0000 0000 1010 0010	00A2h
+0.5	0000 0000 0000 1000	0008h
0	0000 0000 0000 0000	0000h
-0.5	1111 1111 1111 1000	FFF8h
-10.125	1111 1111 0101 1110	FF5Eh
-25.0625	1111 1110 0110 1111	FE6Fh
-55	1111 1100 1001 0000	FC90h

Рисунок 5.4 – Примеры значений температуры

Формат регистров верхней и нижней границ TH и TL показан на рисунке 5.5. Старший бит отводится под знак, остальные 7 бит под кодирование температуры. При сравнении температуры с граничными значениями TH и TL используются биты с 4 по 11 регистра температуры, т.к. регистры TH и TL хранят только 8 бит. При превышении верхней границы TH или понижения температуры ниже границы TL датчик генерирует сигнал выхода за установленные границы (alarm flag).

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
S	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Рисунок 5.5 – Формат регистров TH и TL

Формат регистра настроек представлен на рисунке 5.6.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0	R1	R0	1	1	1	1	1

Рисунок 5.6 – Формат регистра настроек

Для настроек используются биты 5 и 6, которыми задается разрешающая способность АЦП (от 9 до 12 бит). На рисунке 5.7 приведены значения разрядности АЦП и времени преобразования.

R1	R0	RESOLUTION (BITS)	MAX CONVERSION TIME	
0	0	9	93.75ms	(t_{CONV}/8)
0	1	10	187.5ms	(t_{CONV}/4)
1	0	11	375ms	(t_{CONV}/2)
1	1	12	750ms	(t_{CONV})

Рисунок 5.7 – Значения разрядности АЦП и времени преобразования

Для контроля целостности данных используется контрольная сумма CRC8. CRC8 рассчитывается для байтов 0–7 памяти данных на основе образующего полинома X8+ X5+ X4+1. Схема расчёта CRC8 представлена на рис. 8. Данные бит за битом поступают на схему сложения по модулю два, в которой логические элементы расставлены согласно степеням образующего

полинома. После обработки всех байт данных в регистре CRC будет храниться финальная контрольная сумма CRC8 (рис. 5.8).

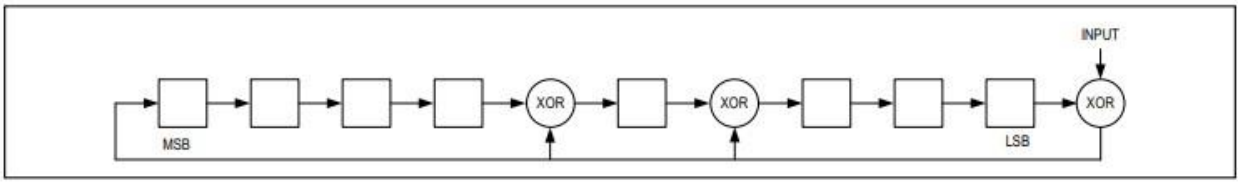


Рисунок 5.8 – Схема расчёта CRC8

Для проверки целостности данных необходимо вычислить CRC8 для принятых 8 байт память данных и сравнить полученную CRC8 с принятой CRC8 вместе с сообщением.

Также можно расчитать CRC8 для всех 9 байт данных, вместе с полученной CRC. Если результат будет равен 0, то сообщение принято без искажений.

Память ROM для хранения кода устройства.

Для хранения кода устройства используется память ROM, структура которой представлена на рисунке 5.9. В младшем байте хранится 8-битный код семейства в виде значения 28h. Затем от младших к старшим битам идет 48-битный серийный номер, а в старшем байте хранится 8-битная CRC, вычисленная для младших 56 бит.

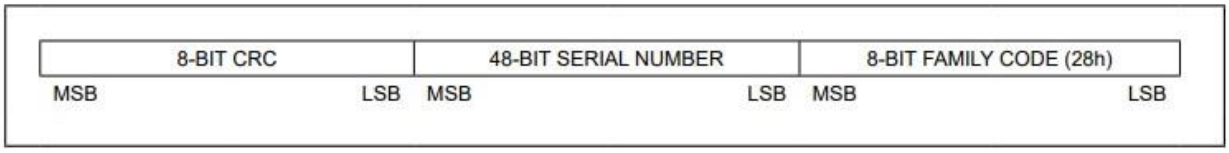


Рисунок 5.9 – Формат кода устройства

Идентификация устройств в сети. Алгоритм поиска адресов.

ROM-код используется для идентификации устройства в сети. Если устройство единственное в сети, то существует команда (0x33), позволяющая считать ROM-код устройства. Также, при наличии только одного устройства, можно использовать широковещательную команду (0xCC) для чтения данных с устройства, что позволяет не использовать ROM-код устройства.

Если устройств в сети несколько, то необходимо либо предварительно узнать код каждого устройства (например, включая устройства по одному и читая их коды в память), либо использовать команду поиска кодов (0xF0) и специальный алгоритм считывания кодов устройств, основанный на том, что доминантным является **низкий** уровень сигнала на шине. Таким образом, если какие-то устройства будут выставлять на шине низкий уровень, а какие-то — высокий, то «победят» те, кто выставлял низкий уровень, и, соответственно, на шине установится низкий уровень.

Идея работы алгоритма поиска устройств заключается в следующем.

После того, как мастер отправляет в сеть команду 0xF0, все Slave-устройства начинают обмениваться с мастером информацией по следующему алгоритму:

1. Slave передаёт мастеру младший бит ROM;
2. Slave передаёт мастеру инверсную копию младшего бита ROM и ждёт ответ от мастера;
3. Если мастер присылает в ответ тот же самый бит, который Slave посылал ему в пункте 1, то Slave повторяет шаги 1,2 для следующего бита ROM, в противном случае Slave прекращает обмен данными с мастером.

Со стороны мастера, при считывании битов от slave-устройств, возможны следующие ситуации:

1. Мастер считывает с шины 01. Это означает, что все устройства, участвующие в обмене данными, послали ему биты 01 (текущий бит ROM = 0, его инверсная копия = 1);
2. Мастер считывает с шины 10. Это означает, что все устройства, участвующие в обмене данными, послали ему биты 10 (аналогично первому пункту, только наоборот);
3. Мастер считывает с шины 00. Это означает, что какие-то устройства послали ему 01, а какие-то 10 (в обоих случаях победили те, кто посылал нули). Это означает, что перед мастером находится узел двоичного дерева, то

есть на этой ветви существуют такие ROM-адреса, у которых следующий бит равен нулю и такие, у которых следующий бит равен единице;

4. Есть ещё вариант, когда мастер считывает с шины 11. Казалось бы, такая ситуация является невероятной, однако она тоже возможна, например, если устройства, с которыми мастер решил продолжить обмен, отключились.

Таким образом, ответ от мастера является выбором направления обхода дерева. Отправляя в шину в качестве ответа ноль или единицу, мастер как раз и решает с какой группой устройств он продолжает работу и по какой ветке он дальше пойдёт. На рисунке 5.10 представлено дерево поиска ROM-кодов устройств для 4-х датчиков. Идея поиска адресов заключается в обходе дерева и побитном считывании адреса каждого датчика в память устройствамастера. Пока все ведомые устройства (датчики) отправляют свои одинаковые первый и второй бит (инверсию первого), т.е. передают одинаковую часть адреса, мастер считывает адрес по одной ветке (байт 0x28). После того, как ведомые устройства пришлют разные биты, например, два устройства пришлют «0» и два пришлют «1», осуществляется ветвление, и мастер продолжает обмен с устройствами, приславшими «0» (т.к. в 1-Wire «0» является доминирующими), запоминая номер бита, в котором произошло ветвление. Если после ветвления, оставшиеся устройства присылают разные биты, то ветвление осуществляется снова, запоминается номер бита ветвления, и обмен продолжается с устройством, приславшим «0». После чего, мастер считывает бит за битом адрес оставшегося ведомого устройства.

Согласно рисунку 5.10 это будет устройство с адресом DE00000BC13FA028.

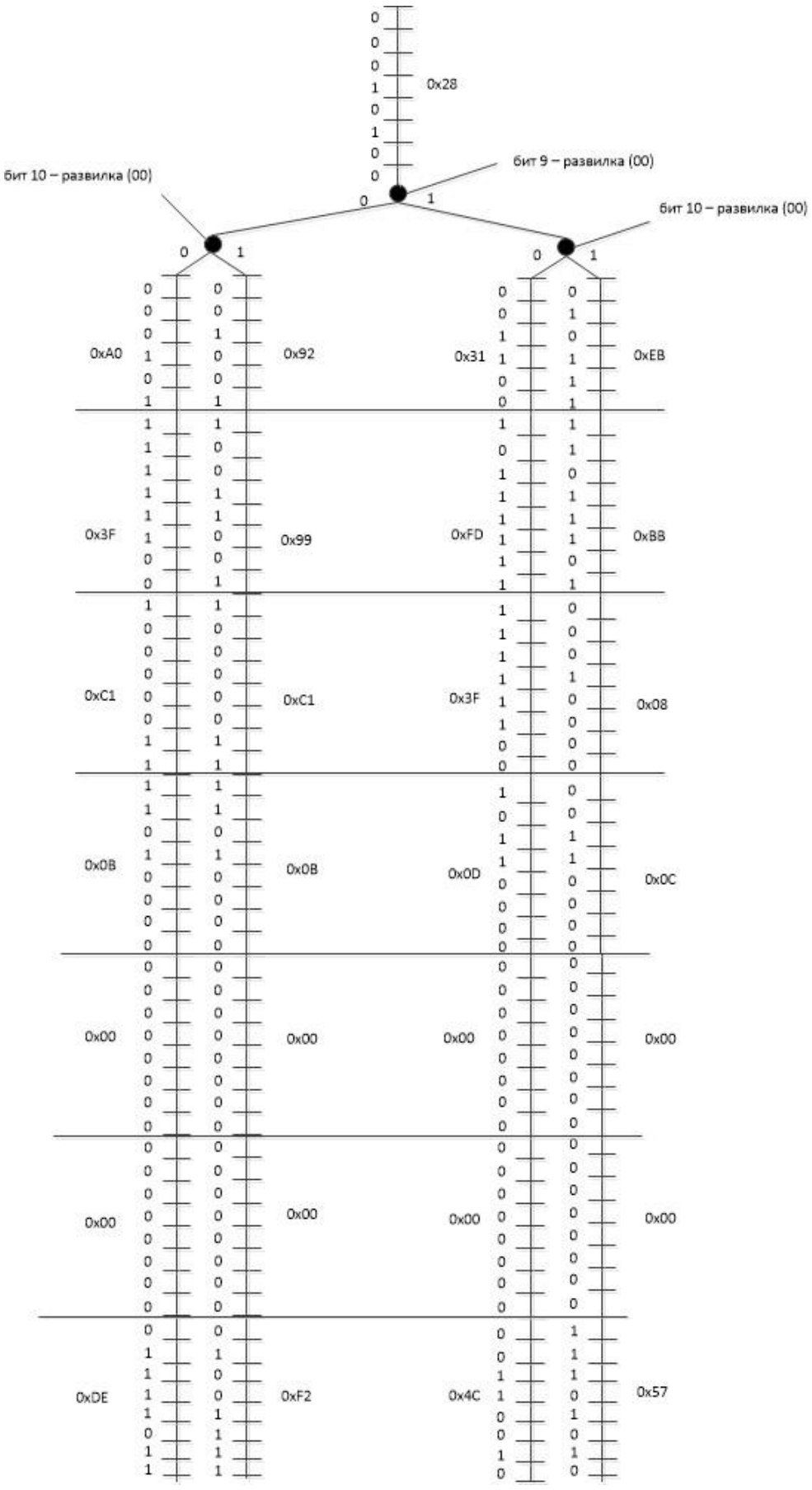


Рисунок 5.10 – Дерево поиска ROM-кодов устройств

После получения первого адреса, обхода дерева начинается заново с самого младшего бита. Получив общую часть адреса и осуществив переход с сторону «0» на 9-м бите, осуществляется переход в сторону «1» на 10-м бите, т.к. этот номер бита был запомнен как последнее ветвление. Считав адрес второго устройства, мастер получит F200000BC1999228, переместит точку ветвления на 9-й бит, после чего продолжит считывание с нулевого бита. При третьем считывании адреса, мастер на 9-бите осуществит переход в сторону «1», а затем на 10-м перейдет в сторону «0» и считает адрес третьего устройства 4C00000D3FFD3128, запомнив 10й-бит, как место ветвления. При считывании четвертого адреса, мастер, дойдя до первого ветвления (9-й бит), переходит в сторону «1», на втором ветвлении (10-й бит) также переходи в сторону «1» и считывает побитно адрес четвертого устройства. Считанные адреса можно сохранять в массив (ROM_code) специальной структуры, созданной для каждого датчика (рис. 5.11).

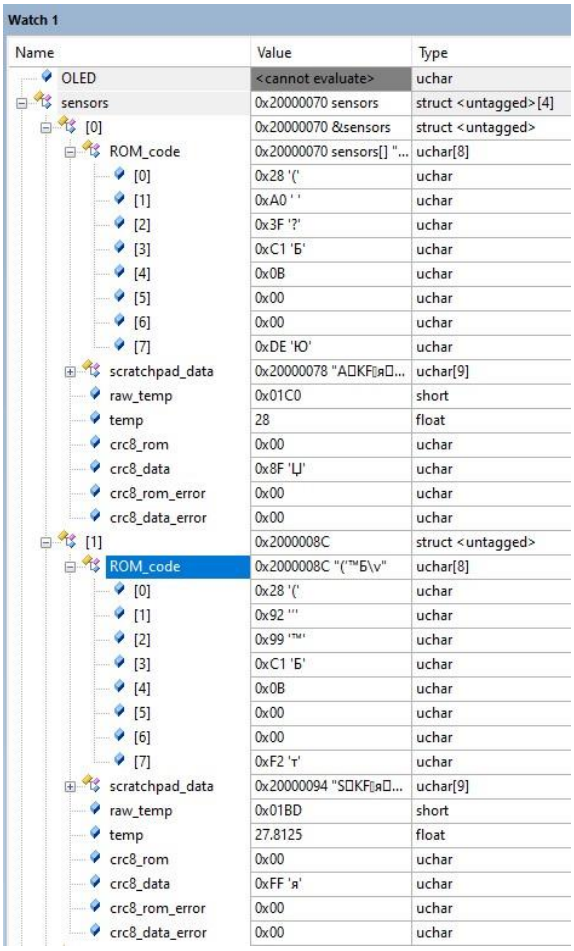


Рисунок 5.11 – Считанные адреса двух датчиков

Описание команд датчика.

В таблице 5.1 приведено описание команды для работы с датчиком по интерфейсу 1-Wire.

Таблица 5.1 – Описание команд

Команда	Код команды (hex)	Описание
Search ROM	F0	Поиск ROM-кодов подключенных устройств. Данная команда должна работать в связке со специальным алгоритмом поиска ROM-кодов.
Read ROM	33	Чтение ROM-кода, если только одно устройство присутствует в сети. При наличии нескольких устройств использовать не рекомендуется, т.к. будет возникать коллизия.
Match ROM	55	Адресация устройства. За этой командой должен следовать ROM-код устройства для установления связи с ним.
Skip ROM	CC	Широковещательная команда всем устройствам. После данной команды можно запустить преобразование температуры для всех датчиков одновременно. Если в сети есть только одной устройство, то после данной команды можно считать данные (Scratchpad) с устройства.
Alarm Search	EC	Команда, аналогичная Search ROM, только для устройств, в которых установлен Alarm flag.
Convert T	44	Команда преобразования температуры. После этой команды можно считать температуру командой Read Scratchpad (BE)
Write Scratchpad	4E	Команда записи данных в scratchpad. После нее можно последовательно записать данные в 3 регистра: TH, TL, Configuration
Read Scratchpad	BE	Команда чтения данных из scratchpad. Читает все 9 байт памяти Scratchpad вместе с контрольной суммой CRC8

Продолжение таблицы 5.1

Команда	**Код команды (hex)**	**Описание**
Copy Scratchpad	48	Команда копирования данных из Scratchpad в ПЗУ EEPROM для сохранения значений TH, TL и настроек после сброса питания
Recall E2	B8	Команда копирования данных из EEPROM в Scratchpad. Главное устройство может контролировать процесс загрузки в Scratchpad из ПЗУ EEPROM, считывая состояние шины после этой команды. Если на шине логический «0» - это значит идет операция перезагрузки, если логическая «1» - операция выполнена.
Read Power Supply	B4	Команда чтения способа питания датчиков. Если после подачи команды на шине присутствует логический «0» - это значит, что DS18B20 использует паразитное питание. Иначе DS18B20 использует внешнее питание.

Полное описание команд приведено в документации к датчику.

Схема подключения датчиков

На рисунке 5.12 представлена общая схема подключения датчиков DS18B20.

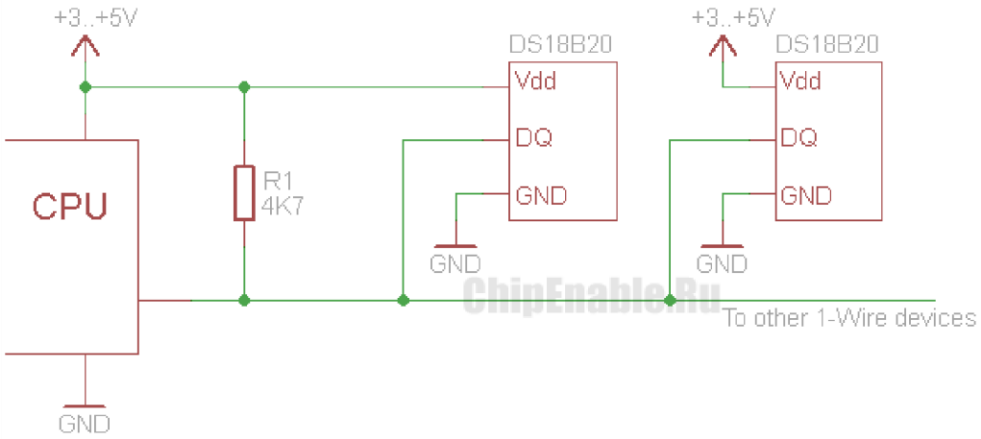


Рисунок 5.12 – Схема подключения датчиков DS18B20

Как видно из схемы рисунка 5.12, линия данных DQ подтянута к питанию через резистор 4.7 кОм. Входы Vdd и GND подключены к питанию и земле соответственно. При подключении к отладочным платам nucleo можно руководствоваться схемой на рисунке 5.13.

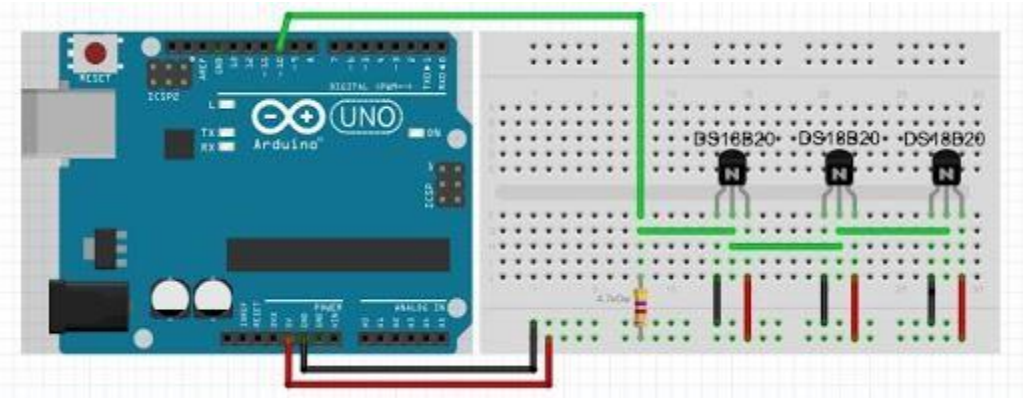


Рисунок 5.13 – Схема подключения датчиков DS18B20 (arduino, nucleo)

6 Практическая работа №6. Работа с интерфейсом 1-Wire с применением CMSIS. Часть 2. Вывод информации на OLED-дисплей

6.1 Цель практики

Получить практические навыки работы с интерфейсом 1-Wire на основе CMSIS (Common Microcontroller Software Interface Standard). Реализовать процедуру поиска адресов датчиков и вывод информации с нескольких датчиков на OLED-дисплей.

6.2 Методические указания

Память ROM для хранения кода устройства.

Для хранения кода устройства используется память ROM, структура которой представлена на рисунке 6.1. В младшем байте хранится 8-битный код семейства в виде значения 28h. Затем от младших к старшим битам идет 48-битный серийный номер, а в старшем байте хранится 8-битная CRC, вычисленная для младших 56 бит.

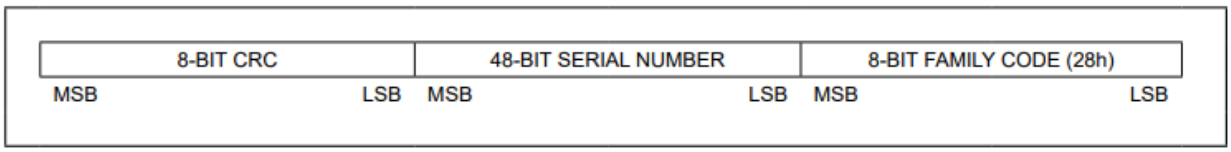


Рисунок 6.1 – Формат кода устройства

Идентификация устройств в сети. Алгоритм поиска адресов.

ROM-код используется для идентификации устройства в сети.

Если устройство единственное в сети, то существует команда (0x33), позволяющая считать ROM-код устройства. Также, при наличии только одного устройства, можно использовать широковещательную команду (0xCC) для чтения данных с устройства, что позволяет не использовать ROM-код устройства.

Если устройств в сети несколько, то необходимо либо предварительно узнать код каждого устройства (например, включая устройства по одному и читая их коды в память), либо использовать команду

поиска кодов (0xF0) и специальный алгоритм считывания кодов устройств, основанный на том, что доминантным является **низкий** уровень сигнала на шине. Таким образом, если какие-то устройства будут выставлять на шине низкий уровень, а какие-то — высокий, то «победят» те, кто выставлял низкий уровень, и, соответственно, на шине установится низкий уровень.

Идея работы алгоритма поиска устройств заключается в следующем.

После того, как мастер отправляет в сеть команду 0xF0, все Slave-устройства начинают обмениваться с мастером информацией по следующему алгоритму:

1. Slave передаёт мастеру младший бит ROM.
2. Slave передаёт мастеру инверсную копию младшего бита ROM и ждёт ответ от мастера.
3. Если мастер присылает в ответ тот же самый бит, который Slave посылал ему в пункте 1, то Slave повторяет шаги 1,2 для следующего бита ROM, в противном случае Slave прекращает обмен данными с мастером.

Со стороны мастера, при считывании битов от slave-устройств, возможны следующие ситуации:

1. Мастер считывает с шины 01. Это означает, что все устройства, участвующие в обмене данными, послали ему биты 01 (текущий бит ROM = 0, его инверсная копия = 1).
2. Мастер считывает с шины 10. Это означает, что все устройства, участвующие в обмене данными, послали ему биты 10 (аналогично первому пункту, только наоборот).
3. Мастер считывает с шины 00. Это означает, что какие-то устройства послали ему 01, а какие-то 10 (в обоих случаях победили те, кто посылал нули). Это означает, что перед мастером находится узел двоичного дерева, то есть на этой ветви существуют такие ROM-адреса, у которых следующий бит равен нулю и такие, у которых следующий бит равен единице.

4. Есть ещё вариант, когда мастер считывает с шины 11. Казалось бы, такая ситуация является невероятной, однако она тоже возможна, например, если устройства, с которыми мастер решил продолжить обмен, отключились.

Таким образом, ответ от мастера является выбором направления обхода дерева. Отправляя в шину в качестве ответа ноль или единицу, мастер как раз и решает с какой группой устройств он продолжает работу и по какой ветке он дальше пойдёт. На рисунке 6.2 представлено дерево поиска ROM-кодов устройств для 4-х датчиков. Идея поиска адресов заключается в обходе дерева и побитном считывании адреса каждого датчика в память устройства-мастера. Пока все ведомые устройства (датчики) отправляют свои одинаковые первый и второй бит (инверсию первого), т.е. передают одинаковую часть адреса, мастер считывает адрес по одной ветке (байт 0x28). После того, как ведомые устройства пришлют разные биты, например, два устройства пришлют «0» и два пришлют «1», осуществляется ветвление, и мастер продолжает обмен с устройствами, приславшими «0» (т.к. в 1-Wire «0» является доминирующими), запоминая номер бита, в котором произошло ветвление. Если после ветвления, оставшиеся устройства присылают разные биты, то ветвление осуществляется снова, запоминается номер бита ветвления, и обмен продолжается с устройством, приславшим «0». После чего, мастер считывает бит за битом адрес оставшегося ведомого устройства. Согласно рисунку 6.2 это будет устройство с адресом DE00000BC13FA028.

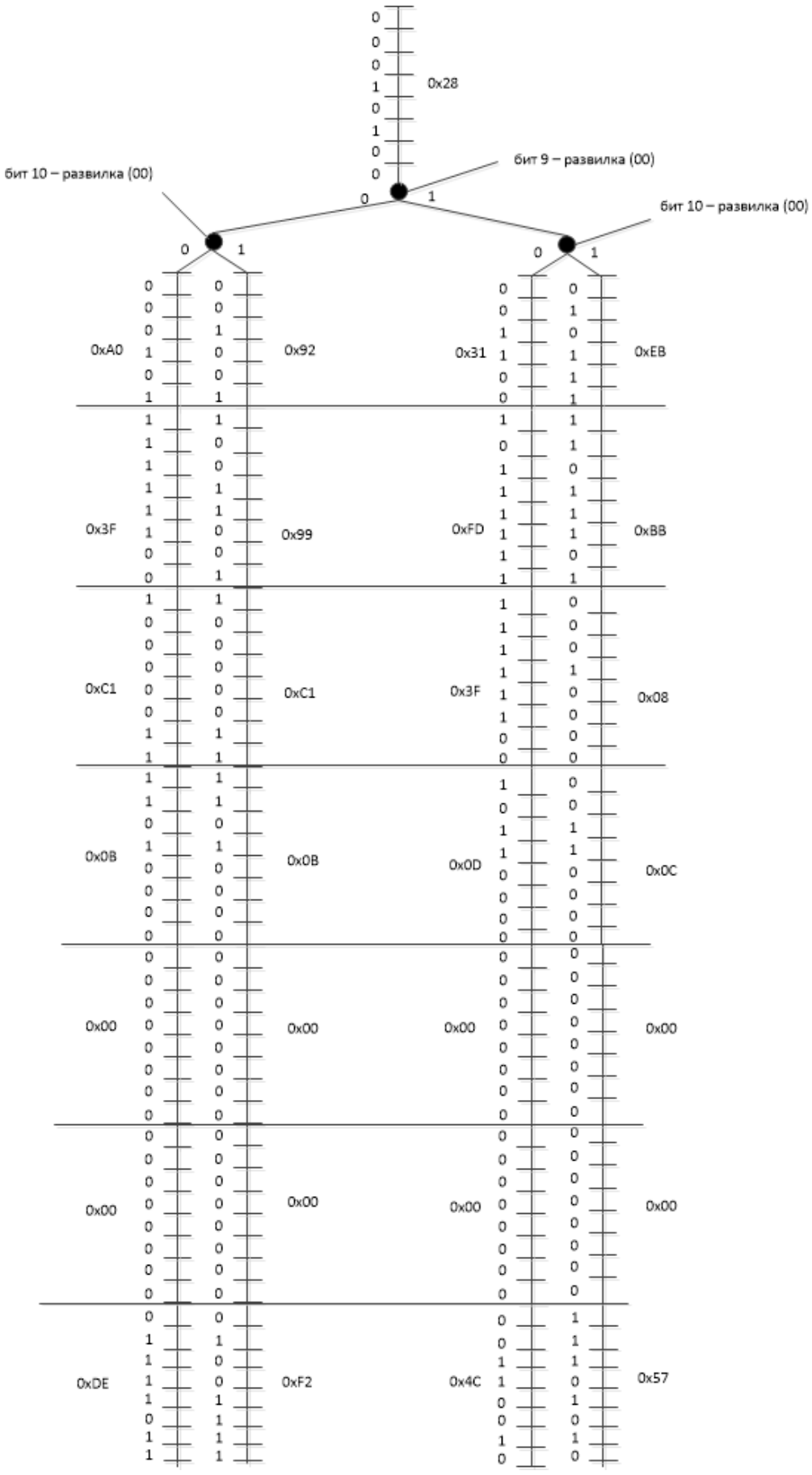


Рисунок 6.2 – Дерево поиска ROM-кодов устройств

После получения первого адреса, обхода дерева начинается заново с самого младшего бита. Получив общую часть адреса и осуществив переход с сторону «0» на 9-м бите, осуществляется переход в сторону «1» на 10-м бите, т.к. этот номер бита был запомнен как последнее ветвление. Считав адрес второго устройства, мастер получит F200000BC1999228, , переместит точку ветвления на 9-й бит, после чего продолжит считывание с нулевого бита. При третьем считывании адреса, мастер на 9-бите осуществит переход в сторону «1», а затем на 10-м перейдет в сторону «0» и считает адрес третьего устройства 4C00000D3FFD3128, запомнив 10й-бит, как место ветвления. При считывании четвертого адреса, мастер, дойдя до первого ветвления (9-й бит), переходит в сторону «1», на втором ветвлении (10-й бит) также переходи в сторону «1» и считывает побитно адрес четвертого устройства. Считанные адреса можно сохранять в массив (ROM_code) специальной структуры, созданной для каждого датчика (рис. 6.3).

Watch 1

Name	Value	Type
OLED	<cannot evaluate>	uchar
sensors	0x20000070 sensors	struct <untagged>[4]
[0]	0x20000070 &sensors	struct <untagged>
ROM_code	0x20000070 sensors[] "...	uchar[8]
[0]	0x28 '('	uchar
[1]	0xA0 ' '	uchar
[2]	0x3F '?'	uchar
[3]	0xC1 'Б'	uchar
[4]	0x0B	uchar
[5]	0x00	uchar
[6]	0x00	uchar
[7]	0xDE 'Ю'	uchar
scratchpad_data	0x20000078 "A□KF□я□...	uchar[9]
raw_temp	0x01C0	short
temp	28	float
crc8_rom	0x00	uchar
crc8_data	0x8F 'Џ'	uchar
crc8_rom_error	0x00	uchar
crc8_data_error	0x00	uchar
[1]	0x2000008C	struct <untagged>
ROM_code	0x2000008C "(™Б\v"	uchar[8]
[0]	0x28 '('	uchar
[1]	0x92 '’'	uchar
[2]	0x99 '™'	uchar
[3]	0xC1 'Б'	uchar
[4]	0x0B	uchar
[5]	0x00	uchar
[6]	0x00	uchar
[7]	0xF2 'т'	uchar
scratchpad_data	0x20000094 "S□KF□я□...	uchar[9]
raw_temp	0x01BD	short
temp	27.8125	float
crc8_rom	0x00	uchar
crc8_data	0xFF 'я'	uchar
crc8_rom_error	0x00	uchar
crc8_data_error	0x00	uchar

Рисунок 6.3 – Считанные адреса двух датчиков

Полная программная реализация алгоритма поиска адресов представлена в примере программы к текущей практике.

Работа с OLED-дисплеем.

Для вывода информации с датчиков можно использовать OLED-дисплей SSD1306 128x64 пикселя, входящий в состав платы расширения Accessory Shield (рис. 6.4). Также можно использовать любой другой дисплей, на который есть возможность вывести информацию температуру с 2-4 датчиков.

Рисунок 6.4 – OLED-дисплей в составе Accessory Shield

Распиновка дисплея в составе платы расширения представлена на рис. 6.5. В таблице 6.1 приведено описание основных, наиболее важных контактов.

Таблица 6.1 – Описание контактов дислея

<table>
<tr><th>Номер</th><th>Обозначение</th><th>Описание</th></tr>
<tr><td>10, 11, 12</td><td>BS0, BS1, BS2</td><td>Выбор протокола обмена
<table>
<tr><td></td><td>BS0</td><td>BS1</td><td>BS2</td></tr>
<tr><td>I2C</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>SPI (3 провода)</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>SPI (4 провода)</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>8 бит 68XX параллельный</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>8 бит 80XX параллельный</td><td>0</td><td>1</td><td>1</td></tr>
</table></td></tr>
<tr><td>14</td><td>RES</td><td>Сигнал сброса. При низком уровне происходит инициализация дисплея. В процессе выполнения операций необходимо удерживать уровень лог. 1</td></tr>
<tr><td>15</td><td>D/C</td><td>В режиме I2C задает выбор адерса ведомого устройства 3C (0) или 3D (1)</td></tr>
<tr><td>13</td><td>CS</td><td>Выбор микросхемы. Необходимо удерживать лог. 0 для взаимодействия с микроконтроллером</td></tr>
<tr><td>18</td><td>SCL</td><td>Сигнал тактирования I2C</td></tr>
<tr><td>19</td><td>SDA</td><td>Сигнал данных I2C</td></tr>
</table>

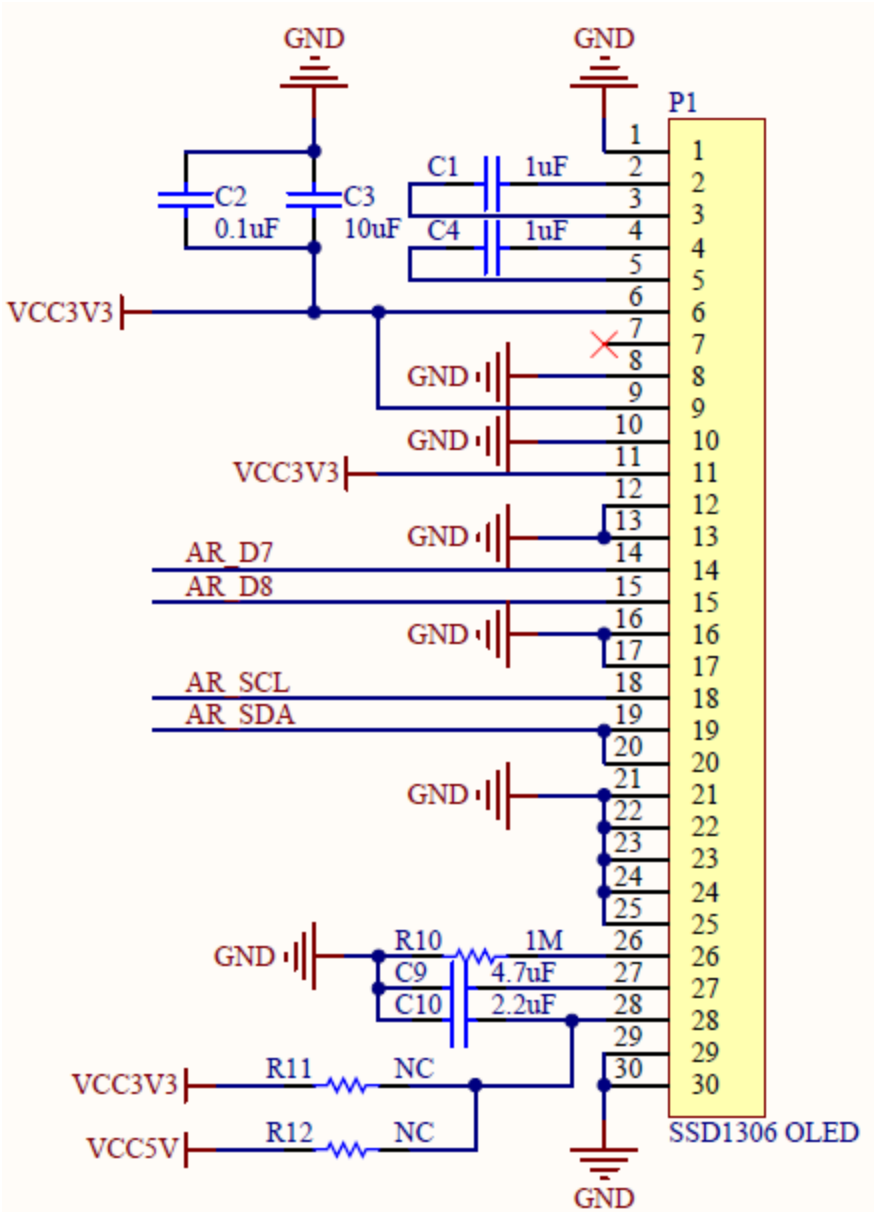


Рисунок 6.5 – Распиновка дисплея

Команды дисплея.

Для работы с дисплеем необходимо провести его инициализацию путем подачи на вход RES логической единицы, затем после задержки 1 мс логического нуля и после задержки 10 мс снова логической единицы. После этого можно отправлять последовательность команд для включения и настройки дисплея. После инициализации и настройки можно выводить информацию, передавая соответствующие команды и данные в память дисплея. Перед отправкой команд необходимо установить режим передачи команд, отправив в дисплей значение 00h. Для установки режима передачи

данных в память дисплея необходимо отправить код 40h. Основные команды для работы с дисплеем приведены в таблице 6.2. Полная система команда описана в документации к дисплею.

Таблица 6.2 – Основные команды дисплея

Команда (hex)	**Функция**	**Описание**
AE/AF	Включение/выключение дисплея	AE – выключить дисплей; AF – включить дисплей.
0x20 A[1:0]	Установка режима адресации	A[1:0] = 00b, горизонтальный режим адресации A[1:0] = 01b, вертикальный режим адресации A[1:0] = 10b, страничный режим адресации A[1:0] = 11b, недопустимо
A0/A1	Переназначение столбцов	A0: адрес столбца 0 соответствует SEG0; A1: адрес столбца 127 соответствует SEG0; (см. раздел организация памяти)
C0/C8	Переназначение строк	C0: нормальный режим, адреса строк соответствуют порядку COM0 – COM7; C8: переназначение, адреса строк соответствуют порядку COM7 – COM0; (см. раздел организация памяти)
8D,14,AF	Включение дисплея	Последовательность команд для включения дисплея
21 A[6:0] B[6:0]	Установка начального и конечного адреса столбцов	A[6:0] : адрес первого столбца B[6:0]: адрес последнего столбца
22 A[2:0] B[3:0]	Установка начального и конечного адреса строк	A[6:0] : адрес первой строки B[6:0]: адрес последней строки

Организация памяти дисплея

В OLED дисплее SSD1306 используется страничная организация памяти, представленная на рисунке 6.6. В каждую страницу можно записать до 128 байт данных, а всего в память дисплея можно записать до 1024 байт (8 страниц по 128 байт).

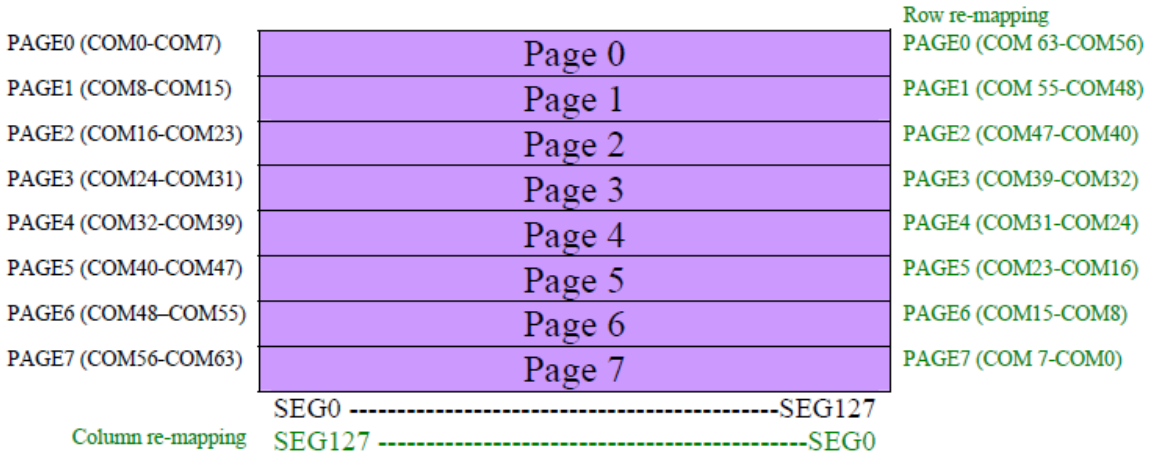


Рисунок 6.6 – Организация памяти дисплея

Запись данных в страницы осуществляется по столбцам, начиная с младшего бита (рис. 6.7).

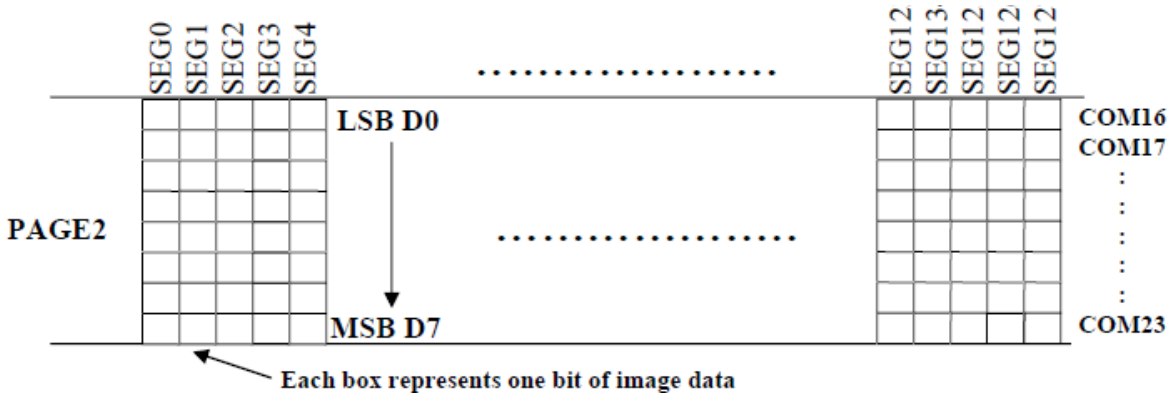


Рисунок 6.7 – Запись данных в страницу

Таким образом, для отображения символов на дисплее необходимо создать их битовый образ в памяти дисплея. В таблице 6.3 представлен пример образа цифры 0 в памяти дисплея. Единицы означают закрашенные пиксели, нули – незакрашенные.

Таблица 6.3 – Пример записи цифры 0

0	**1**	**1**	**1**	0	0
1	0	0	0	**1**	0
1	0	0	**1**	**1**	0
1	0	**1**	0	**1**	0
1	**1**	0	0	**1**	0
1	0	0	0	**1**	0
0	**1**	**1**	**1**	0	0
0	0	0	0	0	0

Таблица 6.4 – Цифра 0 в 3-й и 4-й страницах

бит	0	1	2	3	4	5	
16	0	0	0	0	0	0	
17	0	0	0	0	0	0	
18	0	0	0	0	0	0	
19	0	**1**	**1**	**1**	0	0	Page3
20	**1**	0	0	0	**1**	0	
21	**1**	0	0	**1**	**1**	0	
22	**1**	0	**1**	0	**1**	0	
23	**1**	**1**	0	0	**1**	0	
24	**1**	0	0	0	**1**	0	
25	0	**1**	**1**	**1**	0	0	
26	0	0	0	0	0	0	
27	0	0	0	0	0	0	Page 4
28	0	0	0	0	0	0	
29	0	0	0	0	0	0	
30	0	0	0	0	0	0	
31	0	0	0	0	0	0	

Символы могут располагаться в нескольких страницах (табл. 4). В таком случае необходимо реализовывать алгоритм записи, позволяющий разделить битовое представление символа на части и записать их в разные страницы.

Ниже приведен пример образов символов на языке Си для передачи их в память дисплея.

```
uint8_t Sym_0[6] = {0x3E, 0x51, 0x49, 0x45, 0x3E, 0x00};                    // 0
uint8_t Sym_1[4] = {0x00, 0x42, 0x7F, 0x40};                                // 1
uint8_t Sym_2[6] = {0x42, 0x61, 0x51, 0x49, 0x46, 0x00};                    // 2
uint8_t Sym_3[6] = {0x21, 0x41, 0x45, 0x4B, 0x31, 0x00};                    // 3
uint8_t Sym_4[6] = {0x18, 0x14, 0x12, 0x7F, 0x10, 0x00};                    // 4
uint8_t Sym_5[6] = {0x27, 0x45, 0x45, 0x45, 0x39, 0x00};                    // 5
uint8_t Sym_6[6] = {0x3C, 0x4A, 0x49, 0x49, 0x30, 0x00};                    // 6
uint8_t Sym_7[6] = {0x01, 0x79, 0x09, 0x05, 0x03, 0x00};                    // 7
uint8_t Sym_8[6] = {0x36, 0x49, 0x49, 0x49, 0x36, 0x00};                    // 8
uint8_t Sym_9[6] = {0x06, 0x49, 0x49, 0x29, 0x1E, 0x00};                    // 9
uint8_t Sym_A[6] = {0x7E, 0x09, 0x09, 0x09, 0x7E, 0x00};                    // A
uint8_t Sym_B[6] = {0x7F, 0x49, 0x49, 0x49, 0x36, 0x00};                    // B
uint8_t Sym_C[6] = {0x3E, 0x41, 0x41, 0x41, 0x22, 0x00};                    // C
uint8_t Sym_D[6] = {0x7F, 0x41, 0x41, 0x22, 0x1C, 0x00};                    // D
uint8_t Sym_E[6] = {0x7F, 0x49, 0x49, 0x49, 0x41, 0x00};                    // E
uint8_t Sym_F[6] = {0x7F, 0x09, 0x09, 0x09, 0x01, 0x00};                    // F
uint8_t Sym_x[6] = {0x44, 0x28, 0x10, 0x28, 0x44, 0x00};                    // x
uint8_t Sym_Minus[6] = {0x08, 0x08, 0x08, 0x08, 0x08, 0x00};                // -
uint8_t Sym_Plus[6] = {0x08, 0x08, 0x3E, 0x08, 0x08, 0x00};                 // +
uint8_t Sym_Point[2] = {0x80,0x00};                                         // .
```

Пример отображения температуры с датчиков представлен на рисунке 6.8.

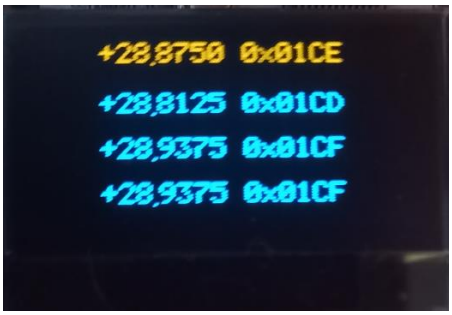


Рисунок 6.8 – Вывод температуры на дисплей

6.3 ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Изменить разрядность АЦП с 12 на 9 бит путем обновления регистра настроек. Реализовать команду Copy Scratchpad для датчика DS18B20. Проверить правильность работы команды Copy Scratchpad после сброса питания (записанные значения T_H и T_L и настройки должны сохраниться в энергонезависимой EEPROM).

2. Вывести на дисплей значения T_H и T_L каждого датчика.

3. На основе примера программы, представленного в разделе Практика 6. Работа с 1-Wire (CMSIS). Часть 2, реализовать вывод своей фамилии на OLED-дисплей.

4. Реализовать вывод на дисплей информации о превышении температуры хотя бы одного из датчиков 29 градусов. Формат вывода: «temp 1 is more then 29 C»

Отчётность

Для слушателей со своими отладочными платами представить в виде отчёта исходный код и результаты работы программы (снимки платы).

Для слушателей без отладочных плат представить описание функций ds18b20_Init, ds18b20_ConvertTemp, ds18b20_ReadStratchpad, ds18b20_ReadROM, Compute_CRC8, ds18b20_ReadBit, ds18b20_WriteBit.

7 Практическая работа №7. Работа с интерфейсом SPI. Логирование данных в памяти EEPROM.

7.1 Цель практики

Получить практические навыки работы с интерфейсом SPI на основе CMSIS (Common Microcontroller Software Interface Standard). Реализовать логирование данных с датчиков DS18B20 в памяти EEPROM, подключенной по интерфейсу SPI.

7.2 Методические указания

Память EEPROM LC25640.

Полная версия документации к памяти LC25640 доступна в прикрепленных файлах раздела «Практика 7. Работа с SPI (CMSIS)» https://udo.tusur.ru/mod/assign/view.php?id=14769.

В данном занятии будет использоваться память EEPROM (Electrically Erasable Programmable Read-Only Memory) LC25640 объемом 8 Кб (64 Кбит) с интерфейсом SPI. Условно-графическое обозначение микросхемы памяти изображено на рисунок 7.1.

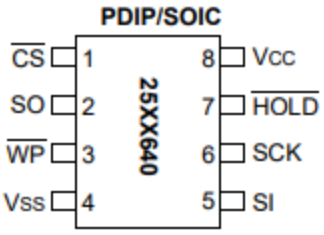


Рисунок 7.1 – Микросхема памяти EEPROM LC25640

Схема подключения памяти к отладочной плате Nucleo-F103RB представлена на рисунке 7.2. Описание контактов приведено в таблице 7.1. Входы $\overline{WP}$ и $\overline{HOLD}$ необходимо подключить к питанию для формирования постоянного уровня логической единицы и отключения аппаратной защиты от записи ($\overline{WP}$) и удержания транзакции ($\overline{HOLD}$).

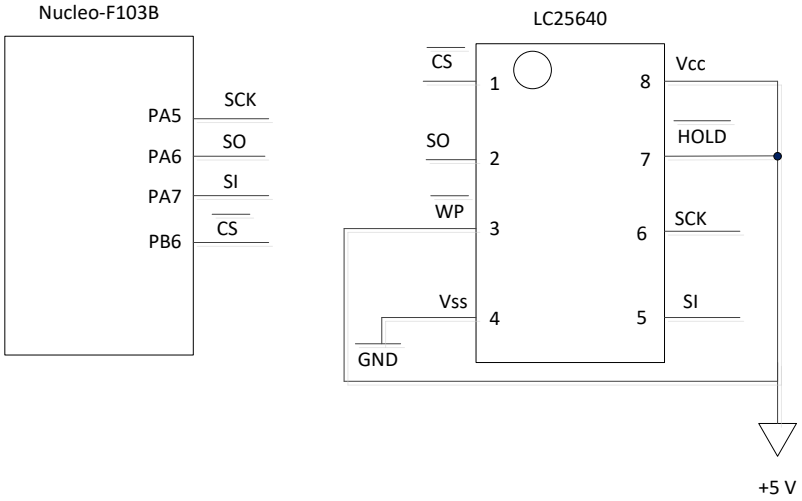


Рисунок 7.2 – Подключение микросхемы памяти к отладочной плате

Таблица 1. Описание контактов LC25640

Имя	**Номер**	**Описание**
$\overline{CS}$	1	Выбор микросхемы (chip select)
SO	2	Выход ведомого (slave output)
$\overline{WP}$	3	Защита от записи (write protect)
Vss	4	Нулевой потенциал (GND)
SI	5	Вход ведомого (slave input)
SCK	6	Тактирование (serial clock)
$\overline{HOLD}$	7	Удержание транзакции (holding input)
Vcc	8	Питание (supply voltage)

Регистр состояний LC25640.

Регистр состояний (Status регистр) хранит текущее состояние и настройки памяти. Структура регистра приведена на рисунке 7.3.

7	6	5	4	3	2	1	0
WPEN	X	X	X	BP1	BP0	WEL	WIP

Рисунок 7.3 – Структура регистра состояний

WIP (Write-in-Process) – значение в этом поле показывает занятость микросхемы выполнением операции записи. 1 – выполняется запись, 0 – запись завершена.

WEL (Write Enable Latch) – бит программного разрешения записи в память и регистр состояний. 1 – запись разрешена, 0 – запись запрещена. Значение бита обновляется командами WREN и WRDI.

BP0 и BP1 (Block protection). Микросхема LC25640 позволяет защитить определенные области памяти от записи путём изменения двух бит регистра состояний BP1 и BP0. В таблице 7.2 показаны области памяти, защищаемые от записи, в зависимости от значений BP0 и BP1.

Таблица 7.2 – Области памяти для защиты от записи

BP1	**BP0**	**Область памяти**
0	0	Нет защиты
0	1	Старшая четверть (адреса 1800h-1FFFh)
1	0	Старшая половина (адреса 1000h-1FFFh)
1	1	Вся память (адреса 0000h-1FFFh)

WPEN (Write-Protect Enable) – бит аппаратного разрешения записи в память. Работает в связке со входом $\overline{WP}$ и задает аппаратное разрешение записи в память и регистр состояний согласно таблице 7.3.

Таблица 7.3 – Режимы защиты памяти от записи

WPEN	**$\overline{WP}$**	**WEL**	**Защищенные области**	**Незащищенные области**	**Регистр состояний**
X	X	0	Защищены	Защищены	Защищен
0	X	1	Защищены	Доступны	Доступен
1	0	1	Защищены	Доступны	Защищен
X	1	1	Защищены	Доступны	Доступен

Система команд LC25640.

Для работы с памятью LC25640 по интерфейсу SPI необходимо использовать специальные команды чтения и записи (таблица 7.4).

Таблица 7.4 – Система команд LC25640

Название	Код команды	Описание
READ	0000 0011 (03h)	Чтение массива данных с выбранного адреса
WRITE	0000 0010 (02h)	Запись массива данных с выбранного адреса
WREN	0000 0110 (06h)	Разрешение записи данных
WRDI	0000 0100 (04h)	Запрет записи данных
RDSR	0000 0101 (05h)	Чтения регистра состояния
WRSR	0000 0001 (01h)	Запись регистра состояния

Работа с командами чтения и записи отражена во временных диаграммах, представленных на рисунках 7.4–7.8. Для чтения данных необходимо через интерфейс SPI отправить команду READ (03h) и адрес памяти, с которого нужно прочитать данные (рис. 7.4). Для записи данных необходимо отправить команду WRITE (02h), адрес память для записи и от 1 до 32 байт данных (рис. 7.5-7.6). Для чтения регистра состояния (Status Register) необходимо отправить команду RDSR (05h) и в ответ будет получено значение регистра (рис. 7.7). Для изменения регистра состояния необходимо отправить команду WRDR (01h) и значение для записи в регистр (рис. 7.8).

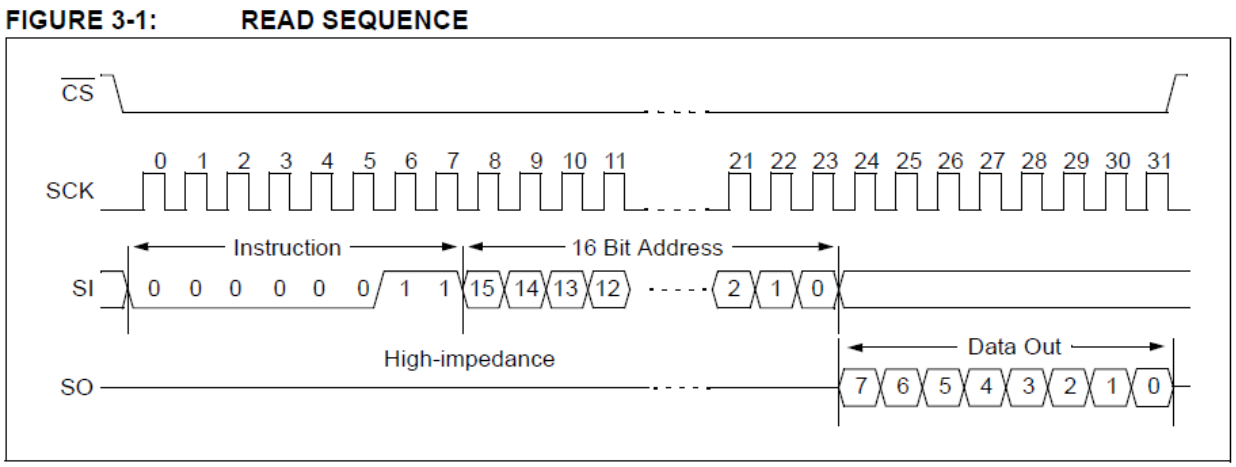


Рисунок 7.4 – Процедура чтения данных

FIGURE 3-2: BYTE WRITE SEQUENCE

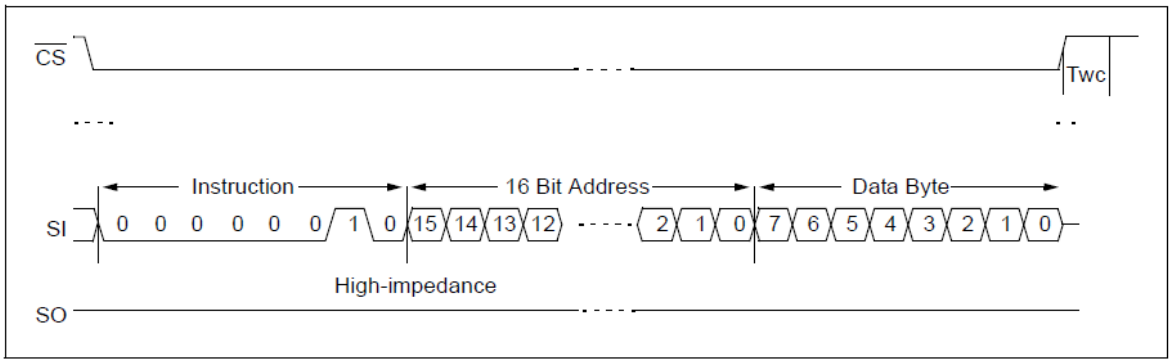


Рисунок 7.5 – Процедура записи байта данных

FIGURE 3-3: PAGE WRITE SEQUENCE

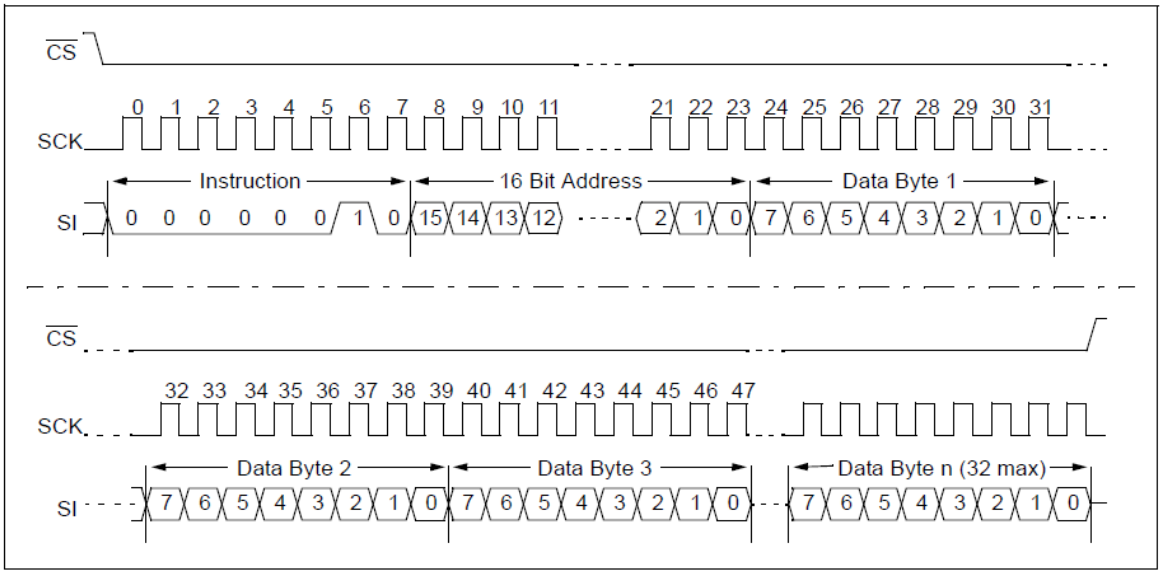


Рисунок 7.6 – Процедура записи массива данных (до 32 байт)

FIGURE 3-6: READ STATUS REGISTER TIMING SEQUENCE

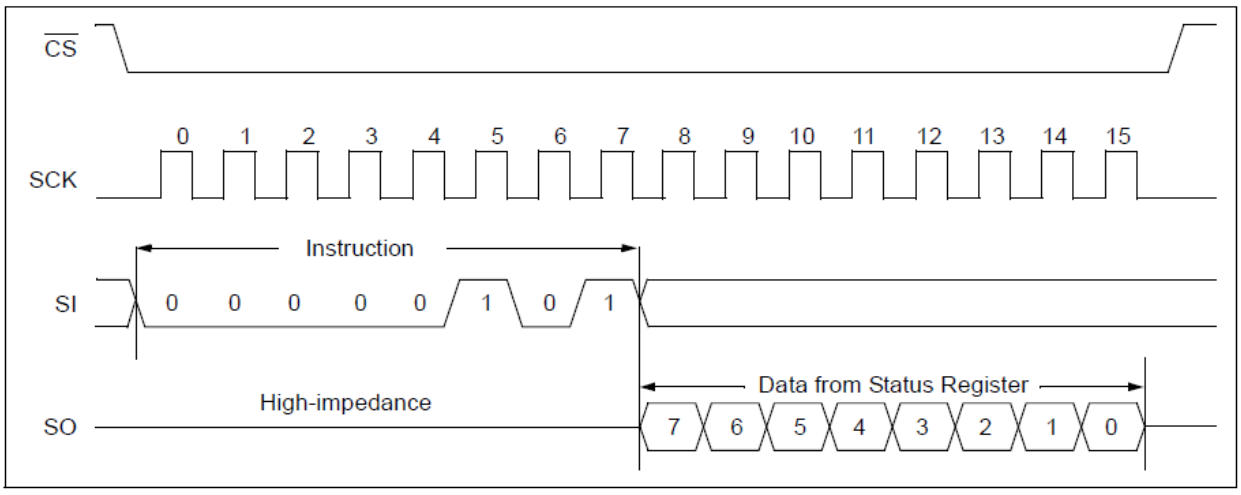


Рисунок 7.7 – Процедура чтения регистра состояния

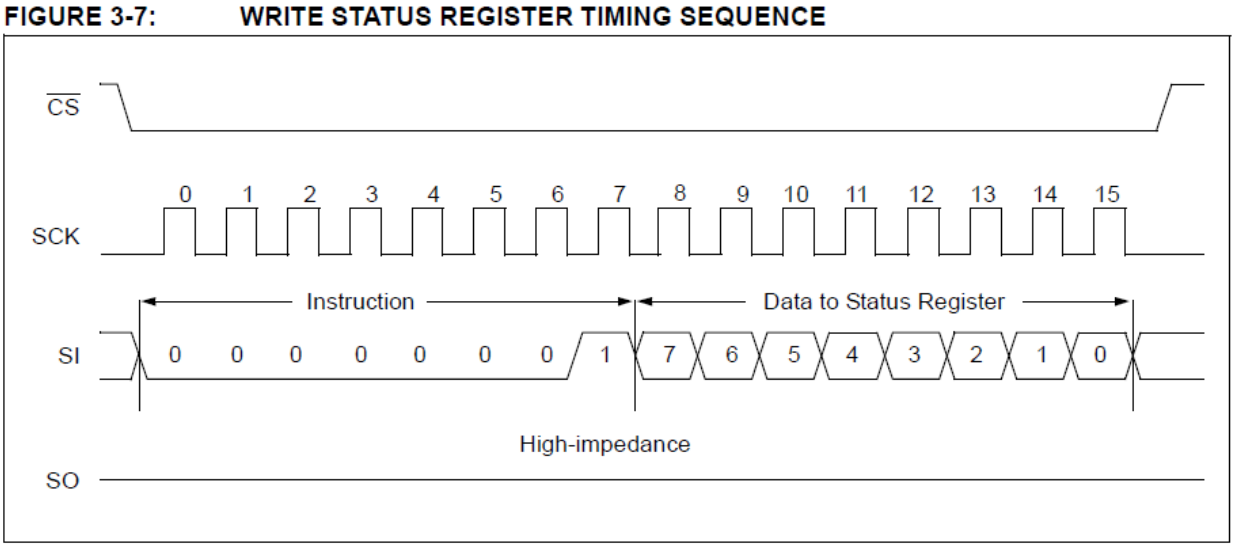


Рисунок 7.8 – Процедура записи в регистр состояния

Интерфейс SPI.

На рисунке 7.9 представлена структурная схема интерфейса SPI в микроконтроллере STM32. Аппаратная реализация SPI включает *буфер приема данных (Rx buffer), буфер отправки (Tx buffer), образующие регистр данных (DR), два управляющих регистра — CR1 и CR2, регистр статуса — SR, блок управления частотой передачи данных или скоростью (baud rate generator), блок управления работой интерфейса в режиме master-slave (master control logic).*

Рассмотрим этапы передачи данных по интерфейсу:

1. Для отправки значение записывается в **SPI_DR**, одновременно происходит установка флага **TXE=0** —что показывает, что **SPI_DR** содержит значение для отправки.

2. Если это первое отправляемое значение (до операции флаг **BSY = 0**), то значение записывается одновременно и в **SPI_DR** и в регистр сдвига (**Shift_Reg**), с которого первый бит (в зависимости от настроек **MSB/LSB**) выставляется по линии **MOSI.**

3. В следующем **SCK** такте, после записи значения в **SPI_DR** и в сдвиговый регистр (**Shift_Reg**), устанавливается флаг **TXE=1** — что означает что в регистр **SPI_DR** можно записать следующее значение для отправки.

При этом прежнее значение содержится в **Shift_Reg** и еще выгружается на линию **MOSI** (т.е. в момент установки флага **TXE = 1** происходит отправка лишь второго бита первоначального значения (из 8 или 16 бит значения)).

4. Если в **SPI_DR** больше данные не записываются — то с флагом **TXE =1** ничего и не происходит, интерфейс ждет загрузки следующего байта.

5. Данные из **Shift_Reg** выгружаются по такту **SCK** на линию **MOSI**. При передаче последнего бита данных, проверяется есть ли новое значение в **SPI_DR** для отправки (в этом случае флаг **TXE = 0**), если нет (это флаг **TXE = 1**), то устанавливается флаг **BSY = 0** и передача прекращается.

Figure 237. SPI block diagram

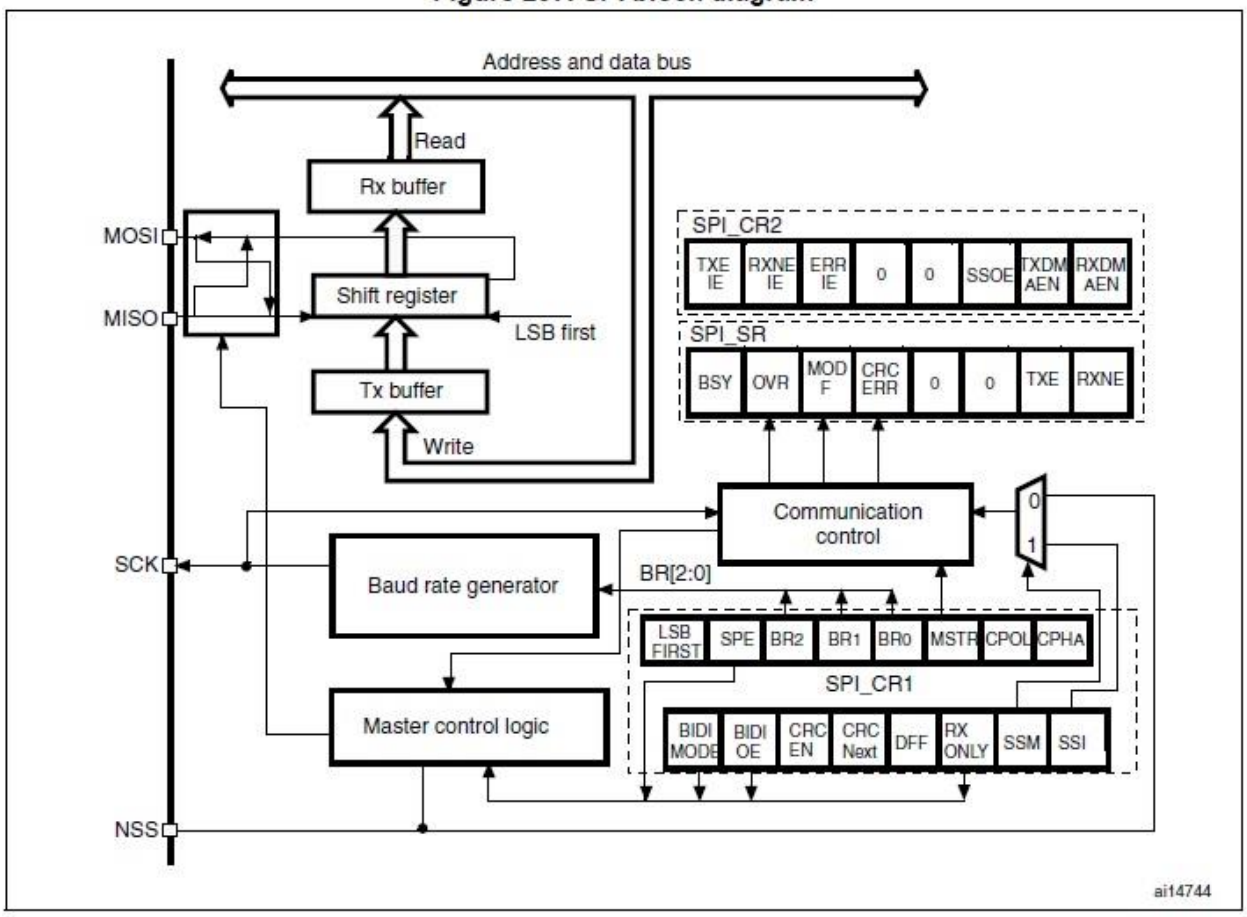


Рисунок 7.9 – Структурная схема интерфейса SPI в STM32F1

Временные диаграммы работы интерфейса представлены на рисунках 7.10 и 7.11.

Рисунок 7.10 – Временная диаграмма работы SPI. Передача байта

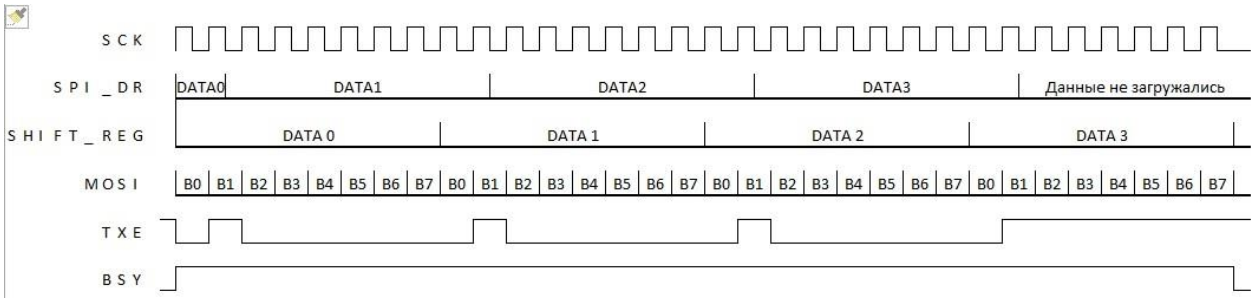


Рисунок 7.11 – Временная диаграмма работы SPI. Непрерывная передача нескольких байт

Описание регистров SPI в STM32 представлено на рисунках 7.12-7.15.

25.5.1 SPI control register 1 (SPI_CR1) (not used in I²S mode)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDI OE	CRC EN	CRC NEXT	DFF	RX ONLY	SSM	SSI	*LSB FIRST*	SPE	BR [2:0]			MSTR	CPOL	CPHA
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 7.12 – Регистр SPI_CR1

BIDIMODE (**Bidirectional data mode enable**): бит двунаправленного режима обмена.

0 – обмен осуществляется по двум направлениям раздельно по разным линиям.

1 – обмен осуществляется по двум направлениям по одной линии.

BIDIOE (**Output enable in bidirectional mode**): бит настройки однопроводного режима обмена данными.

0 – данные только передаются.

1 – данные только принимаются.

CRCEN (**Hardware CRC calculation enable**): бит включения аппаратного подсчёта контрольной суммы.

0 – аппаратный подсчёт CRC отключен.

1 – аппаратный подсчёт CRC включен.

CRCNEXT (**CRC transfer next**): бит включения передачи контрольной суммы в следующем кадре.

0 – передачи контрольной суммы в следующем кадре нет.

1 – следующий кадр – передача CRC.

DFF (**Data frame format**): Формат кадра.

0 – передача данных в 8-битном формате.

1 – передача данных в 16-битном формате.

RXONLY (**Receive only**): Бит управления направлением передачи данных в двухпроводном режиме.

0 – разрешена и передача и приём данных (полный дуплекс).

1 – разрешен только приём данных.

SSM (**Software slave management**): бит управления ножкой выбора микросхемы в ведомом режиме (NSS).

0 – Состояние ножки контролируется в аппаратном режиме.

1 – Состояние ножки не контролируется, контроль идёт по состоянию бита SSI.

SSI (**Internal slave select**): состояние данного бита заменяет состояние ножки NSS. Актуален только при установленном бите SSM. Значение этого бита подается на вывод NSS, а значение на выводе NSS игнорируется.

LSBFIRST (**Frame format**): бит очерёдности битов в кадре.

0 – обмен данными производится старшим битом вперёд.

1 – обмен производится старшим битом вперёд.

SPE (**SPI enable**): бит включения модуля.

0 – модуль SPI отключен.

1 – модуль SPI включен.

BR[2:0] (**Baud rate control**): битовое поле настройки скорости работы шины. В зависимости от комбинации значений битов данного поля устанавливается определённый делитель частоты работы шины, к которой подключен модуль (SPI1- APB2, SPI2 – APB1)

000: $f_{PCLK}/2$

001: $f_{PCLK}/4$

010: $f_{PCLK}/8$

011: $f_{PCLK}/16$

100: $f_{PCLK}/32$

101: $f_{PCLK}/64$

110: $f_{PCLK}/128$

111: $f_{PCLK}/256$

MSTR (**Master selection**): Настройка типа устройства.

0 – SLAVE.

1 – MASTER.

CPOL (**Clock polarity**): Настройка полярности шины синхронизации.

0 – в режиме ожидания уровень низкий.

1 – в режиме ожидания уровень высокий.

CPHA (**Clock phase**): Настройка фазы считывания значения бит.

0 – выборка данных по переднему фронту сигнала CLK.

1 – выборка данных по заднему фронту (по спаду) сигнала CLK.

25.5.2 SPI control register 2 (SPI_CR2)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TXEIE	RXNEIE	ERRIE	Res.	Res.	*SSOE*	TXDMAEN	RXDMAEN
								rw	rw	rw			rw	rw	rw

Рисунок 7.13 – Регистр SPI_CR2

TXEIE (**Tx buffer empty interrupt enable**): бит включения прерываний пустоты буфера передачи.

0 – прерывания TXE запрещены.

1 – прерывания TXE разрешены.

RXNEIE (**RX buffer not empty interrupt enable**): бит включения прерываний полноты (когда буфер не пуст) буфера приёма.

0 – прерывания RXNEIE запрещены.

1 – прерывания RXNEIE разрешены.

ERRIE (**Error interrupt enable**)): бит включения прерываний при возникновении ошибок.

0 – прерывания при возникновении ошибок запрещены.

1 – прерывания при возникновении ошибок разрешены.

SSOE (**SS output enable**): бит управления ножкой NSS в режиме MASTER.

0 – управление ножкой в режиме MASTER отключено.

1 – вывод NSS используется в качестве выхода и модуль SPI управляет данной ножкой сам.

TXDMAEN (**Tx buffer DMA enable**): флаг разрешения формирования запроса к DMA при установке бита TXE.

0 – формирование запроса запрещено.

1 – формирование запроса разрешено.

RXDMAEN (**Rx buffer DMA enable**): флаг разрешения формирования запроса к DMA при установке бита RXNE.

0 – формирование запроса запрещено.

1 – формирование запроса разрешено.

25.5.3 SPI status register (SPI_SR)

Address offset: 0x08

Reset value: 0x0002

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								BSY	OVR	MODF	CRC ERR	UDR	CHSIDE	TXE	RXNE
								r	r	r	rc_w0	r	r	r	r

Рисунок 7.14 – Регистр SPI_SR

BSY (**Busy flag**): флаг занятости шины.

0 – шина свободна.

1 – шина занята или буфер TX не пуст.

OVR (**Overrun flag**): флаг переполнения буфера.

0 – переполнение буфера не обнаружено.

1 – обнаружено переполнение буфера или новые данные в буфере пытаются перезаписать старые, которые ещё не были прочитаны.

MODF (**Mode fault**): Флаг обнаружения на шине NSS низкого уровня в режиме MASTER. Устанавливается аппаратно, сбрасывается только путём определённой последовательности.

0 – обнаружения низкого уровня не зафиксировано.

1 – обнаружен низкий уровень на шине NSS в режиме MASTER.

CRCERR (**CRC error flag**): флаг ошибки контрольной суммы.

0 – ошибка контрольной суммы не обнаружена.

1 – контрольная сумма отличается от рассчитанной.

UDR (**Underrun flag**): флаг недостаточного заполнения буфера. Актуален только для режима I2S. Также должен быть установлен бит ERRIE в регистре CR2. Устанавливается, когда программа ещё не загрузила значение в регистр DR. Очищается чтением регистра SR.

0 – неполного заполнения буфера не обнаружено.

1 – буфер заполнен не полностью.

CHSIDE (**Channel side**): флаг канала. Не актуален в режиме SPI.

0 – левый канал должен быть передан или получен.

1 – правый канал должен быть передан или получен.

TXE (**Transmit buffer empty**): флаг пустоты буфера передачи.

0 – буфер не пуст, то есть данные из регистра DR при передаче ещё не переместились в сдвиговый регистр.

1 – буфер пуст, то есть данные из регистра DR при передаче переместились в сдвиговый регистр.

RXNE (**Receive buffer not empty**): буфер заполнения буфера приёма.

0 – буфер ещё не заполнен, то есть при приёме данные из сдвигового регистра ещё не поступили в регистр DR.

1 – буфер заполнен, то есть при приёме данные из сдвигового регистра успешно и полностью поступили в регистр DR.

25.5.4 SPI data register (SPI_DR)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 7.15 – Регистр SPI_DR

DR[15:0] – данные для приема и отправки

7.3 ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. На основе примера программы, представленного в разделе Практика 7. Работа с SPI (CMSIS), реализовать логирование последних 100 измерений с датчика(ов) температуры в микросхеме памяти.

2. Отключить и включить питание микросхемы. Задать в программе произвольное значение температуры, прочитать данные из памяти и вывести в массив только те значения, которые превышают заданное.

Отчётность.

Для слушателей со своими отладочными платами представить в виде отчёта исходный код и результаты работы программы (снимки платы).

Для слушателей без отладочных плат представить описание функций SPI1_Init, SPI_Transmit, SPI_TransmitReceive, WriteEnableOrDisable, WriteData, ReadData.

8 Лабораторная работа №1. Работа с таймерами и портами ввода-вывода STM32

8.1 Цели работы

Познакомиться с принципом работы периферийных таймеров микроконтроллера STM32F103RBT6. Освоить принцип программной настройки портов ввода-вывода на языке Си для подачи на них сигналов с заданной задержкой.

Постановка задачи:

1. Разработать библиотеку на языке Си (с минимально необходимым функционалом) для настройки системной тактовой частоты (SYSCLK) микроконтроллера. Настроить частоту согласно варианту задания. Если точно настроить частоту не получается, можно взять меньшее наиболее близкое значение. Для реализации задержек разработать библиотеку с функцией Delay на основе системного таймера.

2. Разработать отдельные библиотеки на языке Си для работы с портами ввода-вывода, таймером (TIM) и светодиодом (с минимально необходимым функционалом). Реализовать формирование периодического прямоугольного сигнала на выходе («пине») порта A. Номер «пина» порта A и частота выходного сигнала (в кГц) выбирается согласно варианту (табл. 1.1). Задание п.2 выполнить в симуляторе Keil.

3. Настроить номер пина порта A для подачи сигнала на PA5 (к этому пину подключен светодиод LD2 отладочной платы). Настроить среду keil для загрузки прошивки в отладочную плату согласно методическим указаниям. Загрузить реализованную в п.1 и п.2 прошивку в отладочную плату микроконтроллера STM32F103RB. Оценить корректность работы прошивки. При необходимости внести изменения в программу.

Таблица 8.1 – Исходные данные

№ вар.	Частота SYSCLK, МГц	Номер пина/частота выходного сигнала, кГц	Источник тактирования	Номер таймера
1	8	13/1	HSI	4
2	8	12/2	HSE	3
3	12	11/3	HSI	4
4	16	10/4	HSE	3
5	20	9/5	HSI	4
6	24	8/6	HSE	3
7	28	7/7	HSI	4
8	32	6/8	HSE	3
9	36	4/9	HSI	4
10	40	3/10	HSE	3
11	44	2/11	HSI	4
12	48	1/12	HSE	3

8.2 Методические указания

Отладочная плата.

В рамках данного курса будет использоваться отладочная плата Nucleo-F103RB (рис. 8.1) на базе микроконтроллера STM32F103RBT6 с архитектурой ARM Cortex-M3.

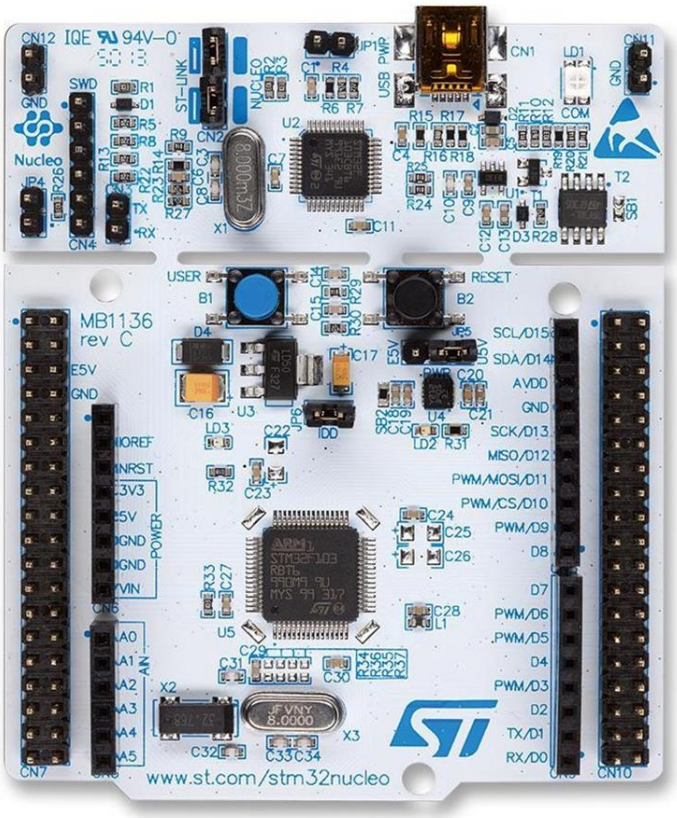


Рисунок 8.1 – Отладочная плата nucleo-F103RB

На рисунке 8.2 представлены основные элементы отладочный платы Nucleo-F103RB.

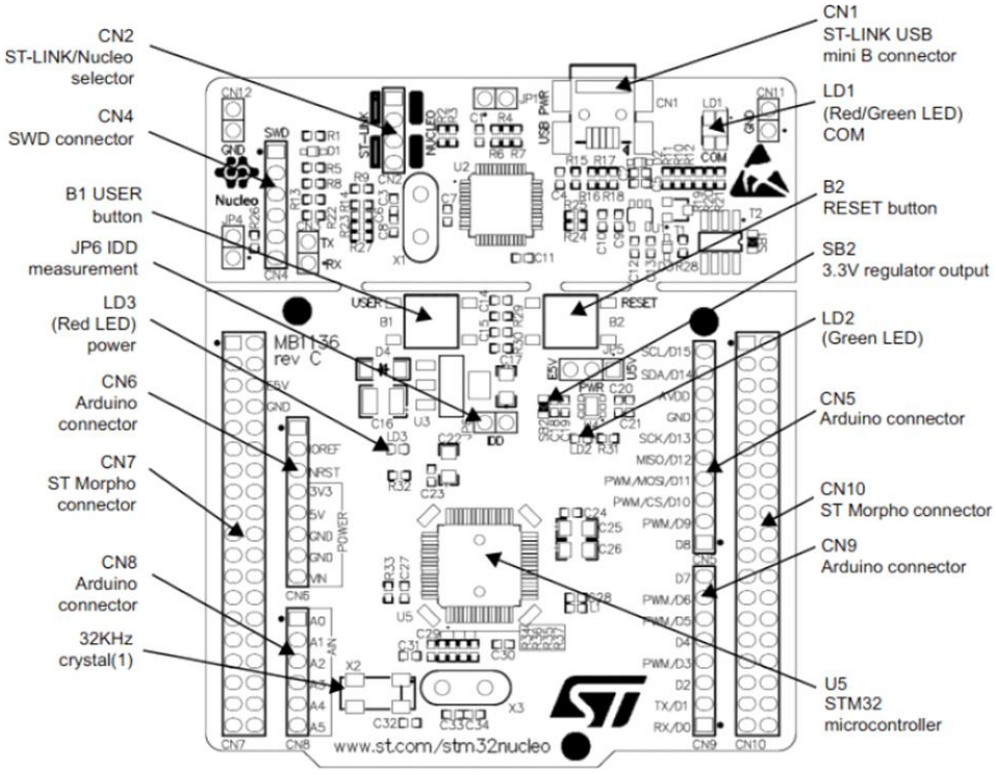


Рисунок 8.2 – Основные элементы Nucleo-F103RB

В виду того, что сам микроконтроллер является элементом отладочной платы, которая реализует удобный («безпаечный») способ подключения периферийных устройств к микроконтроллеру, необходимо иметь представление о расположении контактов (pins, «пинов», рис. 8.3) и их соответствии выводам микроконтроллера (рис. 8.4).

Семейство отладочных плат Nucleo обеспечивает доступный и гибкий путь для проверки новых идей и изготовления прототипов устройств с любыми STM32 микроконтроллерами указанной линейки, выбирая различные комбинации производительности, потребляемой мощности и других характеристик. Поддержка соединителей Arduino (для подключения Arduino-совместимых устройств и переноса проектов на данную архитектуру) и ST Morpho (разъемы расширения для полного доступа ко всем выводам I/O STM32) (рис. 8.3) позволяет просто и быстро расширить функциональность открытой платформы STM32 Nucleo, используя широкий спектр специализированных плат расширения. Платы семейства не требуют отдельного инструментария для программирования и отладки, так как на STM32 Nucleo уже интегрирован ST-LINK/V2-1. Отладочные платы поставляются с комплектом программных библиотек HAL совместно с различными пакетами примеров, а также с возможностью прямого доступа к онлайн ресурсам комплекса mbed.

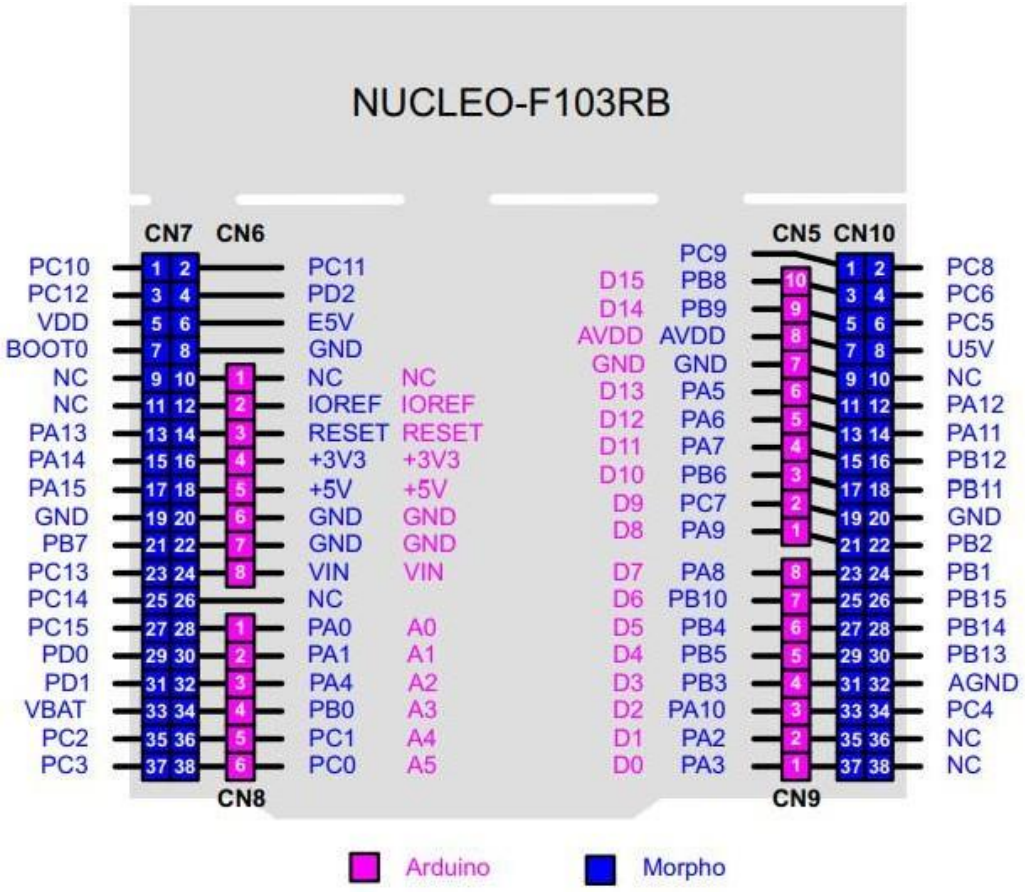


Рисунок 8.3 – Расположение контактов отладочной платы

Расположения контактов микроконтроллера представлено на рис. 8.4.

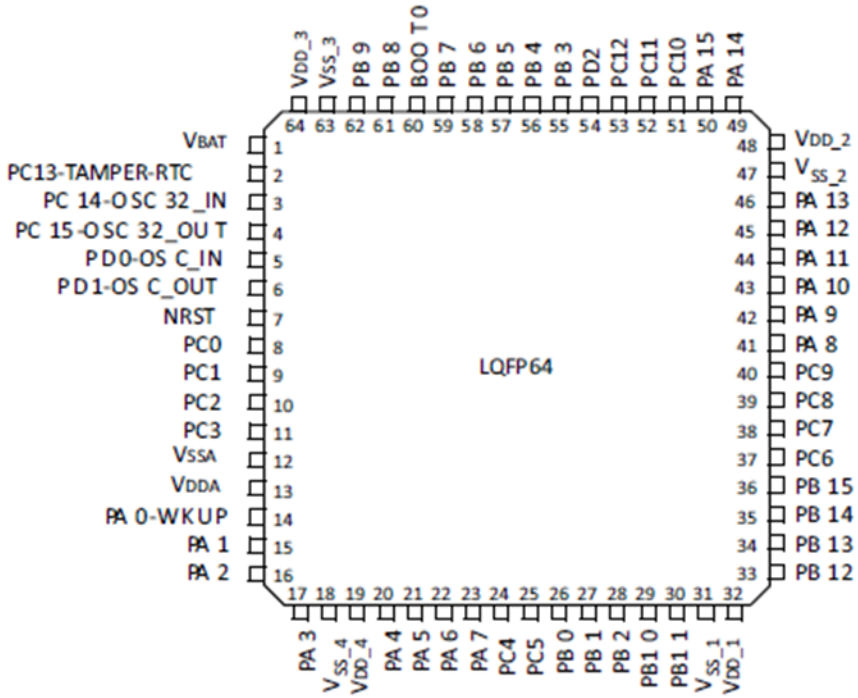


Рисунок 8.4 – Расположение контактов микроконтроллера

Для взаимодействия с периферийными устройствами в структурной схеме (рис. 8.5) можно выделить 16-разрядные порты ввода-вывода общего назначения GPIOA -GPIOE (General Purpose Input Output), порты SPI1 и SPI2, I2C1 и I2C2, USART1 USART3, а также 12-разрядный аналогового-цифровые преобразователи ADC1 и ADC2. В таблице 8.2 представлены характеристики микроконтроллера STM32F103RBT6 их обозначения в структурной схеме.

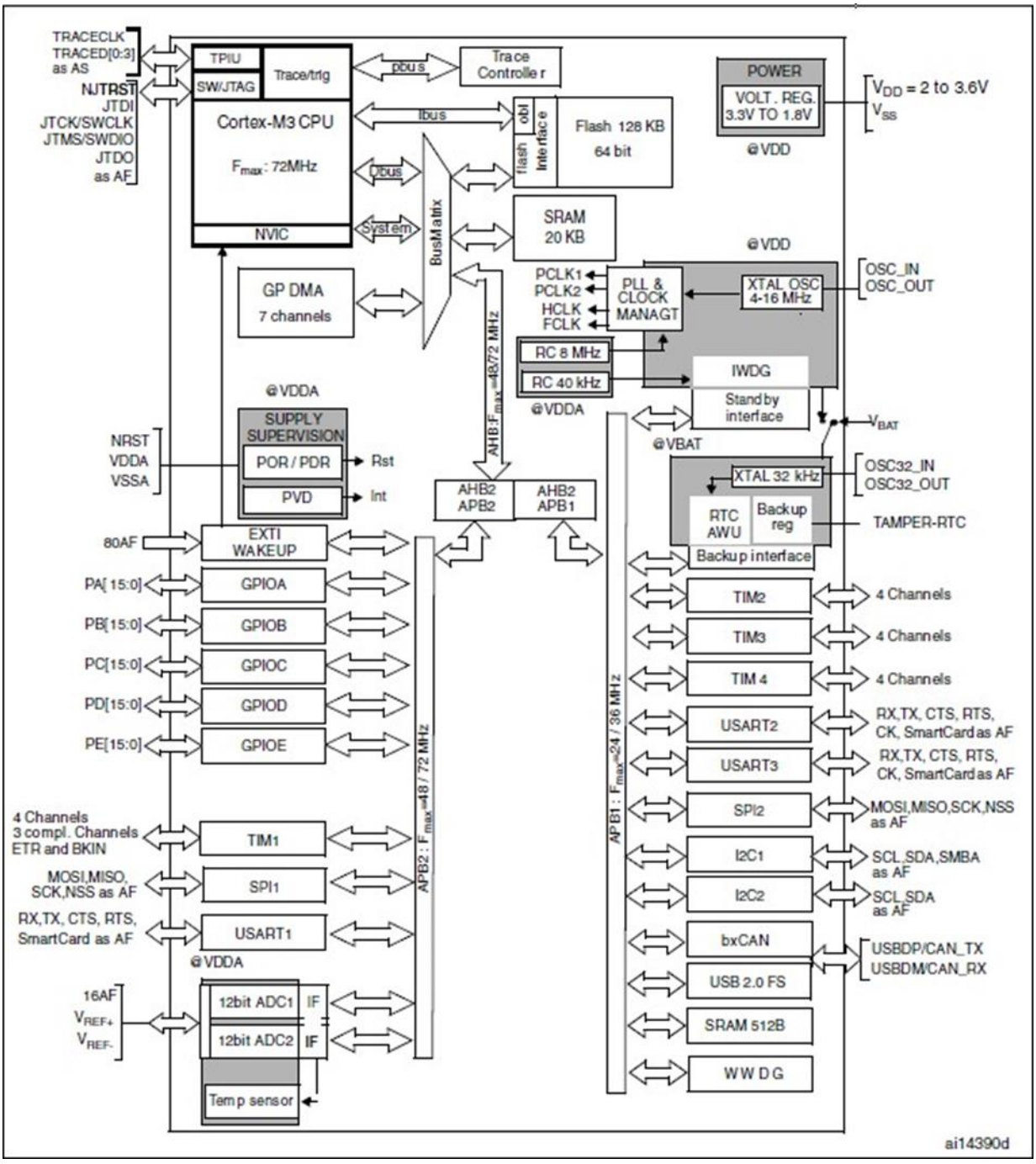


Рисунок 8.5 – Структурная схема микроконтроллера

Таблица 8.2 –Характеристики микроконтроллера STM32F103RBT6

Характеристика	**Значение**	**Обозначение в схеме**
Ядро	ARM® 32-bit Cortex®-M3	Cortex-M3 CPU
Макс. частота ядра	72 МГц	Fmax: 72MHz
Частоты внешних генераторов:		
Высокоскоростной	4 16 МГц	XTAL_OSC 4-16MHz
Низкоскоростной	32,768 кГц	XTAL 32 kHz
Частоты внутренних генераторов:		
Высокоскоростной	8 МГц	RC 8 MHz
Низкоскоростной	40 кГц	RC 40 kHz
Память Flash	128 Кбайт (предвыборка 64 бита)	Flash 128 KB 64 bit
Память SRAM	20 Кбайт	SRAM 20 KB
	512 байт	SRAM 512 B
Напряжение питания	2 3.6 В	POWER
АЦП	2 x 12 бит, до 16 каналов	12 bit ADC1 и ADC2
Таймеры:	7	
16 битный таймер	4	TIM1 TIM4
Сторожевой таймер	2	IWDG, WWDG
Системный таймер	1	SysTick
Интерфейсы:		
I2C	2	I2C1, I2C2
SPI	2	SPI1, SPI2
USART	3	USART1 USART2
CAN	1	bxCAN
USB	1	USB 2.0 FS

В таблице 8.3 приведена принадлежность шин к определенным интерфейсам или устройствам.

Таблица 8.3. Соответствие шины и интерфейса

Интерфейс/устройство	**Шина**
GPIO	APB2
I2C	APB1
TIM	APB1/ APB2
ADC	APB2
DMA	AHB

Примечание: *Каждый блок имеет свой вход тактовых сигналов и бит управления этим входом.* **По умолчанию практически все блоки отключены от тактовых сигналов.** *Для работы с устройствами необходимо не забывать устанавливать соответствующие биты управляющих регистров для включения тактирования.*

Разнообразие делителей, умножителей и источников синхронизации необходимо для гибкой и точной настройки микроконтроллера под задачу, которую ему предстоит решать в том или ином проекте. Если не нужны какие-то блоки, то их можно отключить от источников синхронизации, тем самым уменьшив энергопотребление. Также, когда в проекте не требуется высокая производительность, можно сконфигурировать микроконтроллер на работу на меньшей частоте, что опять же понизит его энергопотребление.

Система тактовых генераторов.

Важнейшей системой микроконтроллера является его система тактовых генераторов. В виду того, что разные периферийные устройства микроконтроллера работают на разных частотах, а также не всегда используются в определенных задачах, необходимо иметь возможность гибкой настройки тактовых частот. На рисунке 8.6 представлена система тактовых генераторов STM32F103RBT6. Основными источниками тактовых сигналов являются внешние (HSE – High Speed External, LSE – Low Speed External) и внутренние (HSI – High Speed Internal, LSI – Low Speed Internal) генераторы сигналов. Все остальные частоты для различных устройств получаются путем умноже-

ния (блок PLLMUL, PLL — система фазовой автоподстройки частоты) или деления (AHB Prescaler) базовых частот на определённый коэффициент.

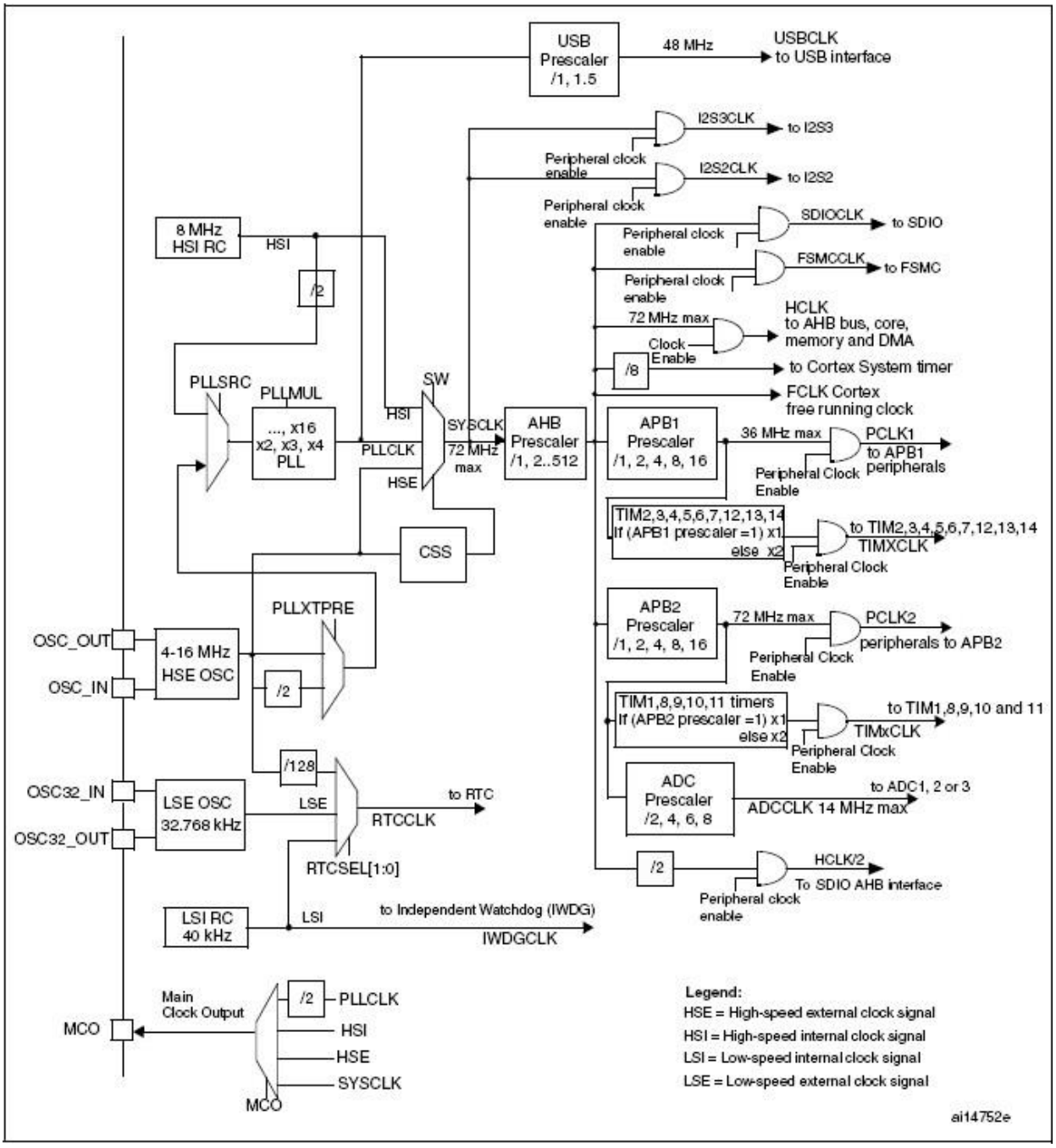


Рисунок 8.6 – Система тактовых генераторов STM32F103RBT6

Работа с частотами осуществляется через настройки значений в специальных регистрах RCC_CR (clock control register) и RCC_CFGR (clock configuration register, рис 8.7 и 8.8).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved						PLL RDY	PLLON	Reserved				CSS ON	HSE BYP	HSE RDY	HSE ON
						r	rw					rw	rw	r	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]					Res.	HSI RDY	HSION
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

Рисунок 8.7 – Регистр RCC_CR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					MCO[2:0]			Res.	USB PRE	PLLMUL[3:0]				PLL XTPRE	PLL SRC
					rw	rw	rw		rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADCPRE[1:0]		PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

Рисунок 8.8 – Регистр RCC_CFGR

High Speed Internal (HSI).

Встроенный в микропроцессор генератор HSI вырабатывает тактовую частоту 8 МГц. Генератор автоматически запускается при появлении питания Vcc и при выходе в нормальный режим работы выставляет флаг HSIRDY в регистре RCC_CR. Первоначально процессорное ядро запускается на тактовой частоте HSI. К преимуществам относится быстрое время начала генерации тактовой частоты после подачи питания и отсутствие необходимости в использовании дополнительных электронных компонентов для работы микроконтроллера. Недостаток – низкая стабильность частоты генерируемого сигнала, а при умножении на PLL погрешность тоже умножается.

High Speed External (HSE).

Внешний генератор HSE по умолчанию выключен и его включение/выключение управляется битом HSEON регистра RCC_CR. После включения HSE и его выхода в рабочий режим устанавливается бит HSERDY кроме этого, может быть сгенерировано прерывание. Также как и сигнал с генератора HSI, сигнал HSE может быть подан напрямую в качестве системного тактового сигнала либо поступать в блок умножения. Но в отличие от HSI в блок умножения он может поступать напрямую, либо пройдя через делитель на 2 (рис. 8.9).

В качестве внешнего генератора могут выступать:

– внешний тактовый сигнал, не превышающий 25МГц, поданный на ножку OSC_IN в то время, как нога OSC_OUT находится в высокоимпендансном состоянии;

– внешний кварцевый резонатор, подключенный на ножки OSC_IN и OSC_OUT. Внешний кварцевый резонатор должен быть в диапазоне от 4 до 16МГц и при его использовании достигается очень высокая стабильность частоты работы генератора.

Low Speed Internal (LSI) и Low Speed External (LSE).

Для работы с такими устройствами, как RTC (Real Time Clock – часы реального времени) и IWDG (Independent Watchdog – независимый сторожевой таймер) используются низкоскоростные тактовые сигналы LSE (Low Speed External – внешний низкоскоростной, для работы которого нужен внешний «часовой» кварц на 32.768кГц и LSI (Low Speed Internal – внутренний низкоскоростной, который генерирует 40кГц).

Phase lock loop (PLL).

Внутренний умножитель частоты (PLLMUL) может умножать частоту вошедшего тактового сигнала с одного из трех источников, на выбор:

– HSI/2 (половина частоты от HIS);

– HSE (частота HSE);

– HSE/2 (половина частоты от HSE).

Множитель варьируется от 2-х до 16-ти. По умолчанию умножитель выключен и его включение/выключение управляется битом PLLON регистра RCC_CR. После включения PLL и его выхода в рабочий режим устанавливается бит PLLRDY кроме этого, может быть сгенерировано прерывание. Работа умножителя конфигурируется через регистр RCC_CFGR. ***Работать с режимами работы умножителя необходимо только при выключенном PLL, т.е. при наличии нуля в поле PLLON регистра RCC_CR.***

Для того, чтобы определить, какой именно источник сигнала использовать перед умножителем, в регистре RCC_CFGR есть бит PLLSRC, который задает источник либо HSE (значение 1), либо HSI/2 (значение 0). Чтобы выбрать источник HSE/2, необходимо установить бит в поле PLLXTPRE. Коэффициент умножения задаётся в диапазоне от 2 до 16 битами PLLMUL[3:0] (0000 – коэффициент 2, 1111 – коэффициент 16).

После блока PLLMUL стоит мультиплексор, с помощью которого можно выбрать источник для системной тактовой частоты (SYSCLK). Для выбора есть специальное поле SW регистра RCC_CFGR. (значения 00 – HSI, 01 – HSE, 10 – PLLCLK, 11 – не разрешено).

Примечание: *из схемы на рисунке 8.6 можно заметить, что шина USB тактируется только с частотой 48 МГц. У шины USB есть собственный делитель (USB Prescaler), с коэффициентами деления 1 и 1.5. Это означает, что при использовании шины USB сигнал PLLCLK должен иметь частоту либо 48 МГц (при USB Prescaler = 1) либо 72 МГц (при USB Prescaler = 1,5).*

Advanced High-performance Bus (AHB) Prescaler.

Системная тактовая частота SYSCLK попадает в делитель шины AHB, который делит ее на делитель от 1 до 512, и все остальные блоки получают на вход уже эту деленную частоту (некоторые блоки имеют еще свои дополнительные делители). По схеме на рисунке 8.6 видно, какой блок имеет делитель, а какой нет, какие коэффициенты деления у делителей, и какая периферия в итоге какую частоту получает.

Например, блоки SDIO и FSMC работают на частоте шины AHB. Шины APB1 и APB2 имеют собственные делители с коэффициентами от 1 до 16. Блок ADC имеет делитель от 2 до 8 и т.д.

Так же на схеме рисунке 8.6 подписаны максимальные допустимые частоты для блоков. Все делители, имеющие управляемые коэффициенты, могут быть настроены, что в целом позволяет довольно гибко варьировать частоты отдельных блоков в различных пределах.

Таймеры общего и специального назначения.

Любой микроконтроллер должен содержать несколько встроенных таймеров-счетчиков (ТС). У STM32 имеются системный таймер

SysTick и четыре блока таймеров. Системный таймер SysTick является частью микропроцессорного ядра Cortex-M3. Таймер 1 – расширенный таймер, предназначенный для управления электродвигателем.

Остальные таймеры являются таймерами общего назначения. Все таймеры выполнены по общей архитектуре, а расширенный таймер отличается лишь добавлением специальных аппаратных блоков.

Системный таймер SysTick. Микропроцессорное ядро Cortex-M3 содержит 24-битный вычитающий счетчик с функциями автоматической перезагрузки и генерации прерывания, который называется таймером SysTick. Он предназначен для использования в качестве стандартного таймера во всех Cortex-микроконтроллерах. Таймер SysTick может использоваться для формирования шкалы времени в ОСРВ или для генерации периодических прерываний для обработки запланированных задач. У таймера SysTick имеется три регистра. С помощью регистра управления и статуса таймера SysTick, который расположен в области системных ресурсов процессора Cortex-M3, пользователь может выбрать источник синхронизации таймера. Если установить бит CLKSOURCE, то таймер SysTick будет работать на тактовой частоте микропроцессора (CPU). Если же его сбросить, таймер будет работать на частоте, равной 1/8 тактовой частоты CPU. Для задания периодичности счета необходимо инициализировать регистр текущего значения и регистр перезагружаемого значения. В регистре управления и статуса имеются биты ENABLE, позволяющий активизировать работу таймера, и TICKINT, управляющий активностью линии прерывания таймера.

Таймеры общего назначения. Все блоки таймеров (рисунок 8.9) выполнены на основе 16-битного перезагружаемого счетчика, который синхронизируется с выхода 16-битного предделителя (PSC). Перезагружаемое зна-

чение хранится в отдельном регистре (ARR). Счет может быть прямой, обратный или двунаправленный (сначала прямой до определенного значения, а затем обратный). Вход синхронизации счетчика можно связать с одним из восьми различных источников. В их число входят: специальный сигнал синхронизации, производный от сигнала главной системной синхронизации; выходной сигнал синхронизации одного из других таймеров или внешний сигнал синхронизации, связанный с выводами захвата/сравнения. Каждый из четырех таймеров микроконтроллера STM32 содержит 16-битный счетчик, 16-битный предделитель частоты и 4- канальный блок захвата/сравнения. Их можно синхронизировать системной синхронизацией, внешними сигналами или другими таймерами.

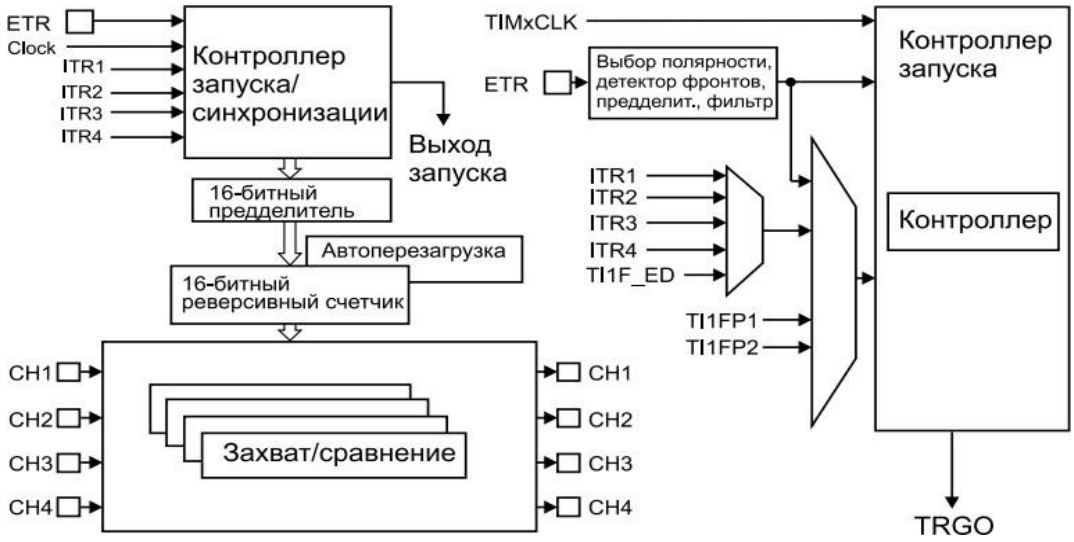


Рисунок 8.9 – Организация блока таймеров общего назначения

Помимо составляющего основу таймера счетчика, в каждый блок таймера также входит четырехканальный блок захвата/сравнения. Данный блок выполняет как стандартные функции захвата и сравнения, так и ряд специальных функций. Каждый из таймеров может генерировать прерывания и поддерживает прямой доступ к памяти (ПДП).

Блок захвата/сравнения.

Каждый канал захвата/сравнения (блок входного захвата, англ. Input capture, IC; блок выходного сравнения, англ. Output compare, OC) управляется через один регистр (рис. 8.10). Данный регистр имеет несколько функций,

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

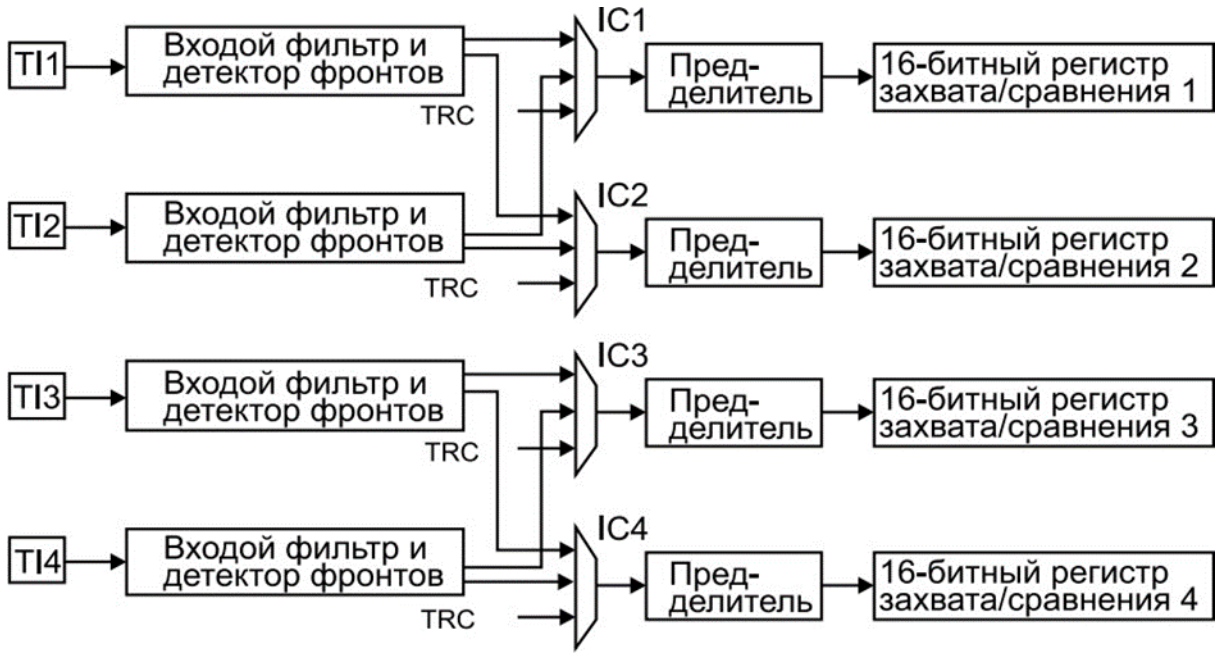


Рисунок 8.10 – Организация блока захвата/сравнения

Базовый блок захвата имеет 4 канала, подключенных к конфигурируемым детекторам фронтов. При обнаружении нарастающего или падающего фронта текущее значение счетчика записывается в 16-битный регистр захвата/сравнения. Когда возникает событие захвата, счетчик таймера может быть сброшен или приостановлен. Кроме того, одновременно с этим может быть запущено прерывание или ПДП-передача. Каждый из четырех блоков захвата имеет входной фильтр и детектор фронтов. При возникновении события захвата может быть запущено прерывание или ПДП- передача. Каждый таймер микроконтроллеров STM32 имеет 4 канала схем сравнения. В базовом режиме сравнения генерируется событие захвата, если обнаруживается совпадение состояния счетчика с 16-битным значением, хранящимся в регистре за-

хвата/сравнения. Данное событие может использоваться для изменения состояния, связанного с каналом захвата/сравнения вывода, генерации сброса таймера, прерывания или ПДП- передачи.

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

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

Схема выходного сравнения. Каждый таймер микроконтроллеров STM32 имеет 4 канала схем сравнения. В базовом режиме сравнения генерируется событие захвата, если обнаруживается совпадение состояния счетчика с 16-битным значением, хранящимся в регистре захвата/сравнения.

Данное событие может использоваться для изменения состояния, связанного с каналом захвата/сравнения вывода, генерации сброса таймера, прерывания или ПДП-передачи. Помимо базового режима сравнения, каждый таймер поддерживает специальный режим генерации ШИМ-сигналов. В этом режиме период ШИМ задается с помощью регистра автоматической перезагрузки таймера. Значение заполнения импульсов задается через регистр захвата/сравнения канала. Таким образом, каждый таймер может генерировать до четырех независимых ШИМ-сигналов. Позже мы увидим, что таймеры могут работать и синхронизированно, позволяя генерировать до 16 синхронизированных ШИМ-сигналов.

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

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

тренного отключения. Источником данного сигнала может быть специальный внешний вывод экстренного отключения или система защиты синхронизации, которая контролирует работу внешнего высокочастотного генератора. После активизации функция экстренного отключения работает полностью автономно и гарантирует перевод ШИМ-выходов в безопасное состояние в случае отказа системы синхронизации микроконтроллера STM32 или в случае повреждения внешней схемы.

Детальная схема устройства таймеров STM32 Cortex-M3 из reference manual представлена на рисунке 8.11.

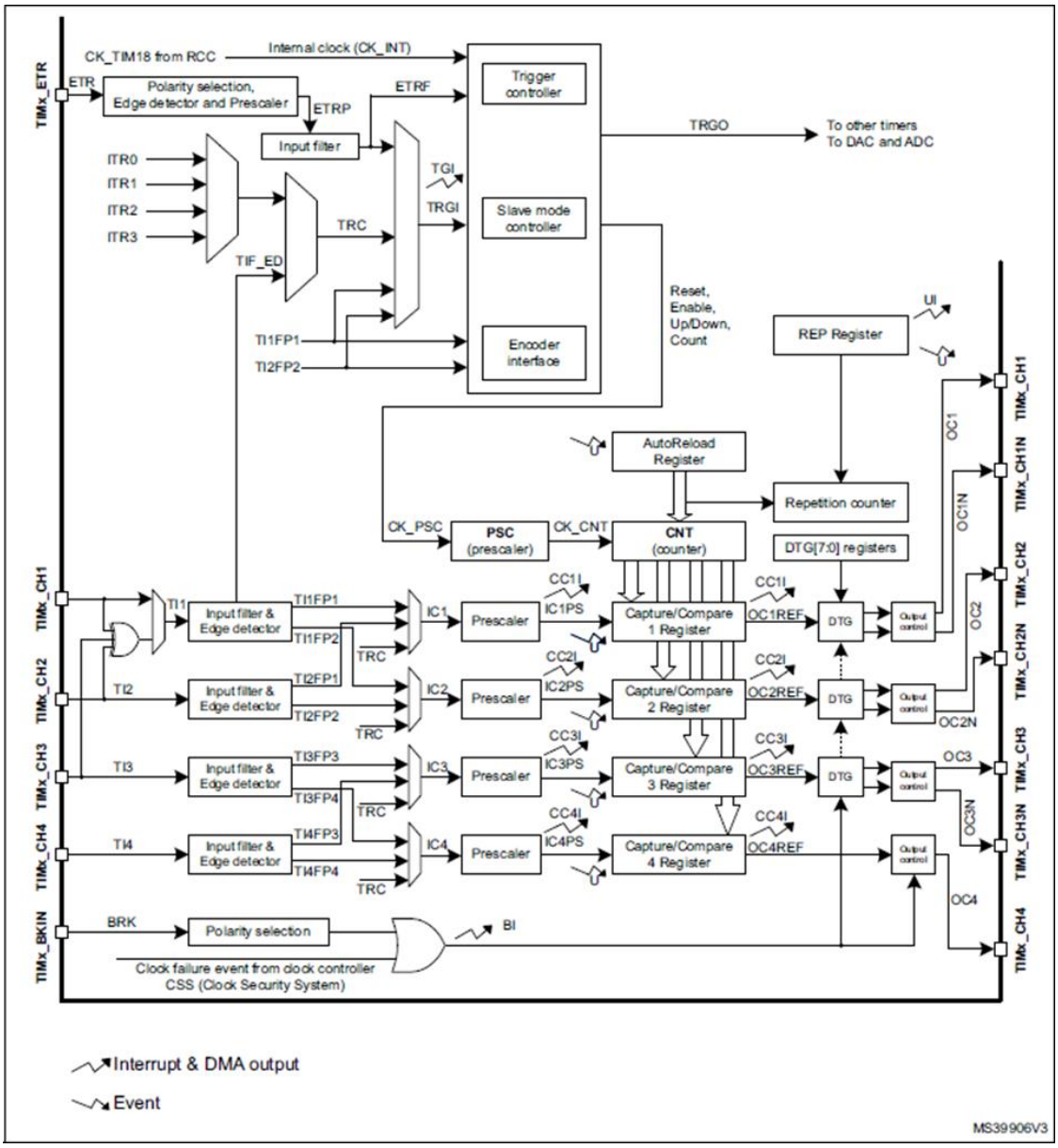


Рисунок 8.11 – Структура таймеров STM32 Cortex-M3

Регистры таймеров общего назначения.

Регистр управления 1 (рис. 8.12). Задает режим работы таймера.

15.4.1 **TIMx control register 1 (TIMx_CR1)**

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 8.12 – Регистр управления 1

Биты 9:8: **CKD (Clock division)** – настройки делителя.

00: tDTS = tCK_INT.

01: tDTS = 2 * tCK_INT.

10: tDTS = 4 * tCK_INT.

11: Зарезервировано.

CK_INT – частота тактирования таймера, соответственно, tCK_INT – время (или период) одного такта.

Бит 7: **ARPE (Auto-reload preload enable)** – бит включения предзагрузки регистра

0 – у регистра TIMx_ARR буферизация отключена,

1 – у регистра TIMx_ARR буферизация включена.

Биты **6:5: CMS (Center-aligned mode selection)** – выбор режима с выравниванием по краю или одного из режимов с выравниванием по центру.

00: режим с выравниванием по краю; счёт ведётся вверх или вниз (режим суммирующего или вычитающего счётчика) в зависимости от значения бита направления DIR.

01: режим 1 с выравниванием по центру; счёт ведётся вверх и вниз поочерёдно; но при счёте вниз происходит установка флага прерывания сравнения для каналов, которые сконфигурированы как выходы (CCxS=00 в регистре TIMx_CCMRx).

10: режим 2 с выравниванием по центру; счёт ведётся вверх и вниз поочерёдно; но при счёте вверх происходит установка флага прерывания срав-

нения для каналов, которые сконфигурированы как выходы (CCxS=00 в регистре TIMx_CCMRx).

11: режим 3 с выравниванием по центру; счёт ведётся вверх и вниз поочерёдно; установка флага прерывания сравнения для каналов, которые сконфигурированы как выходы (CCxS=00 в регистре TIMx_CCMRx) происходит как при счёте вверх, так и при счёте вниз.

Бит **4: DIR (Direction)** – бит направления счёта

0: счёт вверх

1: счёт вниз

Бит **3: OPM (One-pulse mode)** – бит режима одиночного импульса

0: при возникновении события обновления счётчик не останавливается

1: при возникновении события обновления счётчик останавливается

Бит **2: URS (Update request source)** – бит источника запроса на обновление.

0: к генерации прерывания обновления или запроса DMA (если разрешено) привело любое из следующих событий:

– переполнение/антипереполнение счётчика;

– установка бита UG;

– обновление, генерируемое с помощью контроллера подчинённого режима.

1: только переполнение (или обнуление при обратном счёте) привело к генерации UEV Бит 1**: UDIS (Update disable)** – данный бит устанавливается и очищается программно для разрешения/запрета генерации события обновления (UEV).

0: разрешена генерация UEV; случаи, в которых генерируется UEV, определяются значением бита URS.

1: генерация UEV отключена, теневые регистры сохраняют свои значения неизменными (ARR, PSC, CCRx), но счётчик и предделитель сбрасываются при установке бит а UG или при получении сигнала аппаратного сброса от контроллера подчинённого режима.

Бит 0: CEN (Counter enable) – бит включения счётчика.

0: счётчик выключен.

1: счётчик включен.

Регистр управления 2 (рис. 8.13). Задает режим работы таймера.

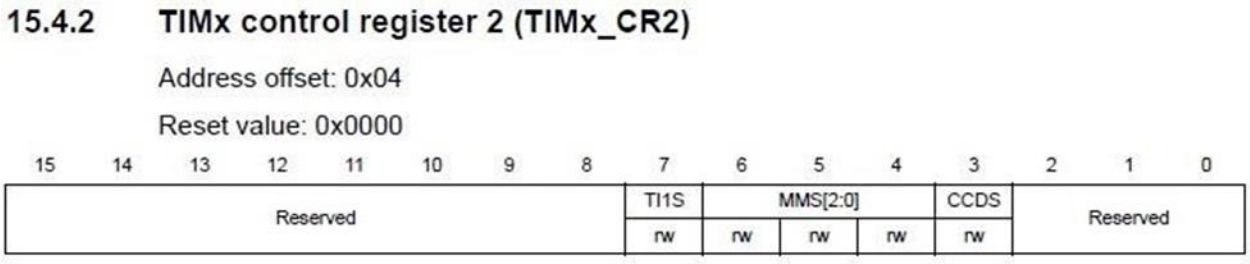


Рисунок 8.13 – Регистр управления 2

Бит 7: **TI1S (TI1 selection)** – бит выбора входа TI1.

0: к входу TI1 подключается вывод TIMy_CH1 микроконтроллера.

1: на вход TI1 подаётся XOR (исключающее ИЛИ) от сигналов с выводов TIMx_CH1, CH2 и CH3 (TI1=CH1 xor CH2 xor CH3).

Биты **6:4: MMS2:MMS0 (Master mode selection)** – выбор мастер-режима (при совместном использовании нескольких таймеров) Возможны следующие варианты:

000: Reset – сброс, в качестве сигнала TRGO используется бит UG регистра TIMx_EGR. Если сброс выполняется входным триггерным сигналом (контроллер подчинённого режима сконфигурирован в режиме перезагрузки), то сигнал TRGO задерживается относительно действительного момента сброса;

001: Enable – включение, в качестве сигнала TRGO используется сигнал запуска счёта CNT_EN. Режим может использоваться, например, для одновременного запуска нескольких таймеров или для управления окном, на протяжении которого включён подчинённый таймер. Сигнал CNT_EN является результатом логической операции OR (ИЛИ) между управляющим битом CEN и входным триггерным сигналом при работе таймера в стробирующем режиме (gated mode). Если сигнал CNT_EN управляется входным триг-

герным сигналом, тогда TRGO формируется с задержкой, за исключением случая, когда выбран режим master/slave (смотрите описание бита MSM регистра TIMx_SMCR);

010: Update – событие обновления формирует сигнал TRGO. Это, например, позволяет использовать таймер в качестве предделителя для подчинённого таймера (таким образом могут быть сформированы весьма продолжительные интервалы времени);

011: Compare Pulse – импульс TRGO формируется тогда, когда устанавливается флаг CC1IF (даже если он уже содержал значение 1), это происходит при входной фиксации или при срабатывании схемы сравнения;

100: Compare – сигнал OC1REF используется в качестве TRGO 101: Compare – сигнал OC2REF используется в качестве TRGO 110: Compare – сигнал OC3REF используется в качестве TRGO 111: Compare – сигнал OC4REF используется в качестве TRGO.

Бит 3: **CCDS (Capture/compare DMA selection)** – бит определяет, в каких случаях формируется запрос DMA:

0: CCx запрос DMA формируется, когда происходит событие CCx;

1: CCx запрос DMA формируется, когда происходит событие обновления.

Регистр состояния (рис. 8.14).

15.4.5 TIMx status register (TIMx_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Reserved		TIF	Res	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0			rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

Рисунок 8.14 – Регистр состояния

Биты **12:9: CC4OF:CC1OF (Capture/Compare 4…1 overcapture flag)** – биты переполнения флага входной фиксации (при работе канала в режиме входа), устанавливаются аппаратно в случае, если происходит фиксация значения счётчика в данном канале (фиксируемое значение помещается в ре-

гистр TIMx_CCRy , y=1 ..4) и при этом флаг фиксации CCyIF у этого канала уже установлен. Сброс бита выполняется программно путём записи в него значения 0.

0: переполнения флага не происходило.

1: обнаружено переполнение флага.

Бит **6: TIF (Trigger interrupt flag)** – данный флаг устанавливается аппаратно триггерным событием (trigger event), которое возникает под действием активного фронта сигнала на входе TRGI при включенном контроллере подчинённого режима во всех режимах, кроме стробирующего. Сбрасывается программно.

0: событие не происходило.

1: обнаружено триггерное событие.

Биты **4:1**: CC4IF (Capture/Compare 4…1 interrupt flag) – назначение данных битов зависит от режима работы канала. Сбрасывается программно.

При работе в режиме сравнения, когда канал используется для формирования выходного сигнала.

0: совпадение не происходило.

1: установка бита происходит при обнаружении равенства содержимого счётчика и содержимого регистра TIMx_CCRy в данном канале; в случае, если TIMx_CCRy превышает значение в регистре перезагрузки TIMx_ARR (а значит, счётчик никогда не достигнет значения в регистре сравнения), то бит устанавливается в 1 при переполнении счётчика (в режимах счёта вверх и вверх/вниз) или обнуления (в режиме счёта вниз).

Бит **0: UIF (Update interrupt flag)** – данный флаг устанавливается при возникновении события обновления. Устанавливается аппаратно, сбрасывается программно.

0: событие обновления не происходило.

1: обнаружено событие обновления.

Регистр автоматической перезагрузки (рис. 8.15).

15.4.12 TIMx auto-reload register (TIMx_ARR)

Address offset: 0x2C

Reset value: 0x 0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 8.15 – Регистр автоматической перезагрузки

В 16 битах данного регистра содержится число для автоматической перезагрузки счётчика. Сюда мы заносим число, до которого будет считать счётчик. Как только досчитает, перезагрузится и начнёт счёт с нуля.

Регистр предделителя (рис. 8.16).

15.4.11 TIMx prescaler (TIMx_PSC)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 8.16 – Регистр предделителя

Биты данного числа содержат значение коэффициента деления предделителя.

Регистр генерации событий (рис. 8.17).

15.4.6 TIMx event generation register (TIMx_EGR)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved									TG	Res.	CC4G	CC3G	CC2G	CC1G	UG
									w		w	w	w	w	w

Рисунок 8.17 – Регистр генерации событий

Назначение их следующее:

Бит **6: TG (Trigger generation)** – бит генерации триггерных событий, устанавливается программно.

0: событие не генерируется

1: генерация триггерного события, устанавливается флаг TIF в регистре TIMx_SR; возможна генерация прерывания и запроса DMA (если разрешено).

Бит **4:1: CC4G, CC3G, CC2G, CC1G (Capture/compare 4-1 generation)** – биты генерации событий фиксации / сравнения. Эти биты устанавливается программно для генерации события фиксации/сравнения (Capture/Compare) для соответствующего канала таймера (1, 2, 3, 4), а сброс бита происходит аппаратно.

0: событие не генерируется

1: генерация события Capture/Compare в соответствующем канале таймера (1, 2, 3, 4), действие бита зависит от режима работы канала

Бит **0: UG (Update generation)** – бит генерации события обновления. Данный бит устанавливается программно, сбрасывается аппаратно.

0: событие не генерируется

1: генерация события обновления. происходит инициализация счётчика и обновляется содержимое буферизируемых регистров, а также сбрасывается счётчик предделителя (коэффициент деления не изменяется). В режиме с выравниванием по центру, счётчик таймера сбрасывается в 0 при DIR=0 (во время счёта вверх) и инициализируется значением из регистра TIMx_ARR при DIR=1 (счёт вниз).

Регистр управления подчинённым режимом таймера (рис. 8. 18).

15.4.3 **TIMx slave mode control register (TIMx_SMCR)**

Address offset: 0x08

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ETP	ECE	ETPS[1:0]		ETF[3:0]				MSM	TS[2:0]			Res.	SMS[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw

Рисунок 8.18 – Регистр управления подчиненным режимом таймера

Бит **15: ETP** (**External trigger polarity**) – бит выбора сигнала, используемого как триггерный (ETR или его инверсия – not ETR).

0: ETR не инвертируется, активным является высокий уровень или нарастающий фронт.

1: ETR инвертируется, активным является низкий уровень или спад сигнала.

Бит **14: ECE (External clock enable)** – бит включения второго режима внешнего тактового сигнала.

0: второй режим внешнего тактового сигнала отключен.

1: второй режим внешнего тактового сигнала включён; счётчик тактируется каждым активным фронтом сигнала ETRF.

Биты **13:12: ETPS (External trigger prescaler)** – биты настройки предделителя внешнего триггерного сигнала.

00..11: коэффициент деления предделителя, который, соответственно, равен 1, 2, 4, 8.

Биты **11:8: ETF3:ETF0 (External trigger filter)** – битовое поле настройки частоты сэмплирования сигнала ETRP и длины цифрового фильтра, применяемого к сигналу ETRP. Цифровой фильтр выполнен в виде счётчика событий и для подтверждения перехода сигнала требуется N последовательных событий считывания нового значения:

0000: фильтр не используется, частота сэмплирования равна f DTS;

0001: fSAMPLING=fCK_INT, N=2;

0010: fSAMPLING=fCK_INT, N=4;

0011: fSAMPLING=fCK_INT, N=8;

0100: fSAMPLING=fDTS/2, N=6;

0101: fSAMPLING=fDTS/2, N=8;

0110: fSAMPLING=fDTS/4, N=6;

0111: fSAMPLING=fDTS/4, N=8;

1000: fSAMPLING=fDTS/8, N=6;

1001: fSAMPLING=fDTS/8, N=8;

1010: fSAMPLING=fDTS/16, N=5;

1011: fSAMPLING=fDTS/16, N=6;

1100: fSAMPLING=fDTS/16, N=8;

1101: fSAMPLING=fDTS/32, N=5;

1110: fSAMPLING=fDTS/32, N=6;

1111: fSAMPLING=fDTS/32, N=8.

Бит **7: MSM (Master/Slave mode)** – бит включения режима Master/Slave

0: режим отключен.

1: режим включён. Вводится задержка между сигналом запуска на входе TRGI и его воздействием на таймер для достижения точной синхронизации между данным таймером и его подчинёнными таймерами (которые управляются сигналом TRGO, см. описание битового поля MMS в регистре TIMx_CR2). Этот режим может быть полезен, если требуется синхронизация нескольких таймеров одним внешним событием.

Биты **6:4: TS (Trigger selection)** – поле выбора входного триггерного сигнала для синхронизации счётчика.

000: Internal Trigger 0 (ITR0) – внутренний триггерный сигнал 0 **001**: Internal Trigger 1 (ITR1) – внутренний триггерный сигнал 1 **010**: Internal Trigger 2 (ITR2) – внутренний триггерный сигнал 2 **011**: Internal Trigger 3 (ITR3) – внутренний триггерный сигнал 3 **100**: TI1 Edge Detector (TI1F_ED) – детектор фронта сигнала TI1.

101: Filtered Timer Input 1 (TI1FP1) – сигнал TI1 (после цифрового фильтра) **110**: Filtered Timer Input 2 (TI2FP2) – сигнал TI2 (после цифрового фильтра) **111**: External Trigger input (ETRF) – внешний триггерный вход ETRF.

Биты **2:0: SMS** (**Slave mode selection**) – поле выбора типа подчинённого режима.

000: подчинённый режим отключён; в этом случае при CEN=1 на предделитель поступает внутренний тактовый сигнал.

001: режим 1 энкодера – счёт вверх/вниз по фронту на TI2FP1; направление счёта определяется уровнем сигнала TI1FP2.

010: режим 2 энкодера – счёт вверх/вниз по фронту на TI1FP2; направление счёта определяется уровнем сигнала TI2FP1.

011: режим 3 энкодера – счёт вверх/вниз по фронту на любом из двух входов TI1FP1, TI2FP2; направление счёта при прохождении импульса на одном входе определяется уровнем сигнала в этот момент на другом входе.

100: Reset Mode, режим сброса – нарастающий фронт сигнала на выбранном триггерном входе (TRGI) сбрасывает счётчик и обновляет регистры.

101: Gated Mode, стробирующий режим – таймер выполняет счёт при высоком уровне сигнала на триггерном входе (TRGI); счётчик останавливается (но не сбрасывается) при переходе сигнала на запускающем входе к низкому уровню; таким образом, внешний сигнал управляет как запуском, так и остановкой счётчика.

110: Trigger Mode, триггерный режим – счётчик запускается (но не сбрасывается) нарастающим фронтом сигнала на триггерном входе TRGI; в отличие от стробирующего режима (Gated Mode), контролируется только запуск счётчика.

111: External Clock Mode 1, режим 1 внешнего тактового сигнала – счётчик тактируется нарастающими фронтами на выбранном триггерном входе (TRGI).

Регистр разрешения прерываний (рис. 8.19).

15.4.4 TIMx DMA/Interrupt enable register (TIMx_DIER)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

Рисунок 8.19 – Регистр разрешения прерываний

Бит **14: TDE (Trigger DMA request enable)** – бит запрета/разрешения запросов DMA в ответ на триггерный сигнал.

0: генерация запроса запрещена.

1: генерация запроса разрешена.

Бит **12: CC4DE (Capture/Compare 4 DMA request enable)** – бит разрешения запрос DMA в случае выполнения входной фиксации или срабатывания схемы сравнения в канале 4.

0: Запрос CC4 DMA запрещен.

1: Запрос CC4 DMA разрешен.

Бит **11: CC4DE (Capture/Compare 3 DMA request enable)** – бит разрешения запрос DMA в случае выполнения входной фиксации или срабатывания схемы сравнения в канале 3.

0: Запрос CC3 DMA запрещен.

1: Запрос CC3 DMA разрешен.

Бит **10: CC4DE (Capture/Compare 2 DMA request enable)** – бит разрешения запрос DMA в случае выполнения входной фиксации или срабатывания схемы сравнения в канале 2.

0: Запрос CC2 DMA запрещен.

1: Запрос CC2 DMA разрешен.

Бит **9: CC4DE (Capture/Compare 1 DMA request enable)** – бит разрешения запрос DMA в случае выполнения входной фиксации или срабатывания схемы сравнения в канале 1.

0: Запрос CC1 DMA запрещен.

1: Запрос CC1 DMA разрешен.

Бит **8: UDE (Update DMA request enable)** – бит разрешения запроса DMA в случае события обновления.

0: Запрос обновления DMA запрещен.

1: Запрос обновления DMA разрешен.

Бит **6: TIE (Trigger interrupt enable)** – бит разрешения прерывания в ответ на триггерный сигнал

0: Прерывание запрещено

1: Прерывание разрешено.

Бит **4: CC4IE (Capture/Compare 4 interrupt enable)** – бит разрешения прерывания в случае выполнения входной фиксации или срабатывания схемы сравнения в канале 4.

0: Прерывание запрещено.

1: Прерывание разрешено.

Бит **3 CC3IE (Capture/Compare 4 interrupt enable)** – бит разрешения прерывания в случае выполнения входной фиксации или срабатывания схемы сравнения в канале 3.

0: Прерывание запрещено

1: Прерывание разрешено.

Бит **2: CC2IE (Capture/Compare 4 interrupt enable)** – бит разрешения прерывания в случае выполнения входной фиксации или срабатывания схемы сравнения в канале 2.

0: Прерывание запрещено.

1: Прерывание разрешено.

Бит **1: CC1IE (Capture/Compare 4 interrupt enable)** – бит разрешения прерывания в случае выполнения входной фиксации или срабатывания схемы сравнения в канале 1

0: Прерывание запрещено

1: Прерывание разрешено.

Бит **0: UIE (Update interrupt enable)** – бит разрешения прерывания в случае, когда происходит событие обновления.

0: Прерывание запрещено

1: Прерывание разрешено.

Примеры программ для работы с таймерами и портами ввода - вывода представлены в файлах лабораторной работы.

Порты ввода/вывода.

Порт ввода-вывода – логическое объединение сигнальных линий, через которое принимаются и передаются данные.

GPIO – General Purpose Input Output.

Каждая линия порта, как правило, обозначается как Pnx или Pn.x, где

• n – обозначение порта;

• x – номер бита (линии) в порте.

• Например, PA5, P1.5.

Каждый порт ввода-вывода обслуживают как минимум 3 служебных регистра:

• регистр, содержащий данные (уровни сигналов) на всех линиях порта и используемый для записи сигналов в порт;

• регистр, содержащий состояния входов порта, доступен только для чтения, используется при чтении данных из порта;

• регистр направления линий порта: каждая линия порта может быть сконфигурирована как вход или как выход в зависимости от значения бита этого регистра.

Порты ввода/вывода в условно-графическом обозначении микроконтроллера показаны на рисунке 8.20.

Если какая-то линия порта ввода-вывода в схеме не используется, она должна быть определена как выход (соответствующий бит регистра направления должен соответствовать выходу), и ее выходное значение должно быть равно 0.

Большинство линий ввода-вывода могут быть сконфигурированы для выполнения альтернативных функций, обозначенных в назначении выводов микроконтроллера.

Для всех линий портов ввода-вывода, как правило, доступна программная конфигурация входных подтягивающих резисторов. **Подтягивающие резисторы** осуществляют доопределение потенциалов «брошенных»

входов напряжением высокого (Pull-up) или низкого (Pull-down) уровня (рис. 8.21).

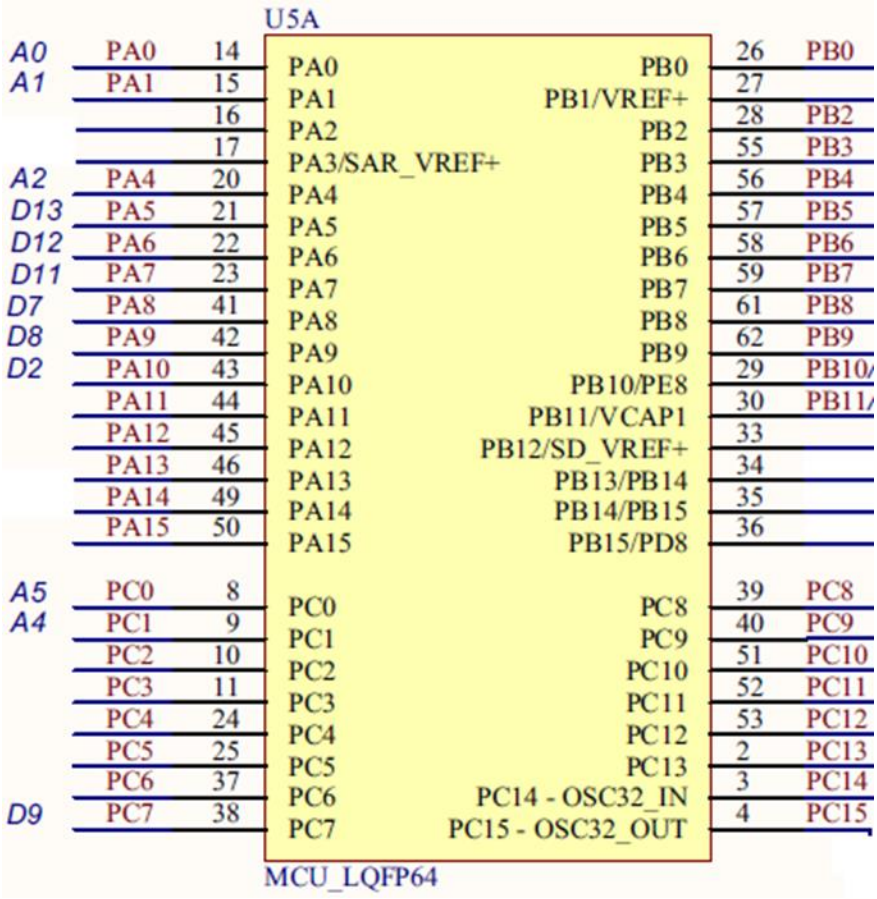


Рисунок 8.20 – Условно-графическое обозначение микроконтроллера

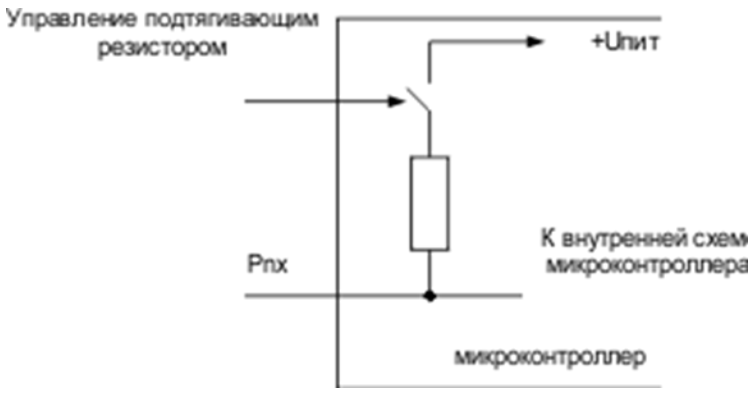


Рисунок 8.21 – Схема порта ввода/вывода

Режимы push-pull и open-drain

Настройка режима push-pull или open-drain зависит от схемы подключения устройства, с которым надо взаимодействовать. Выход PA5 микроконтроллера подключен к аноду светодиода (контакт 1, рис. 8.22).

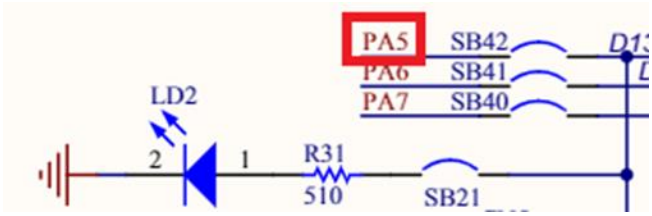


Рисунок 8.22 – Схема подключения светодиода LD2 в nucleo-F103RB

На рисунке 8.23 приведена схема устройства порта ввода-вывода. Красным выделена та самая часть, которая настраивается либо как push-pull, либо как open drain. Эта часть состоит из двух полевых транзисторов, подключенных к питанию Vdd (P-MOS) и земле Vss (N-MOS).

Figure 13. Basic structure of a standard I/O port bit

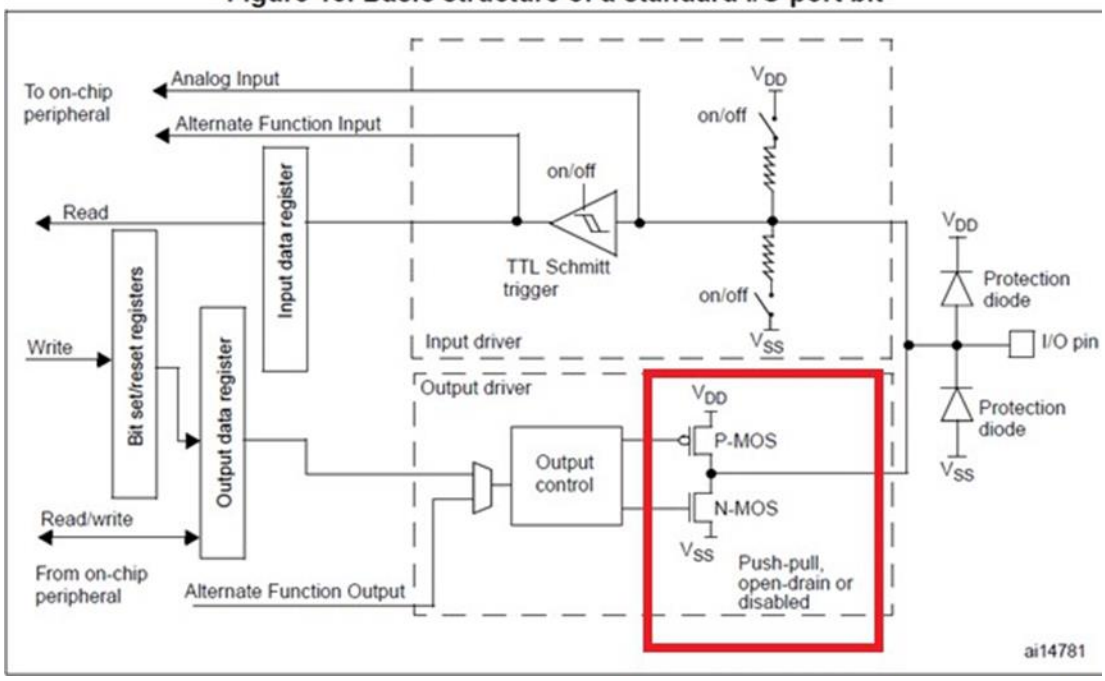


Рисунок 8.23 – Схема порта ввода/вывода из reference manual

На рисунке 8.24 показана отдельная схема в режиме push-pull для формирования высокого или низкого уровня сигнала на выходе порта. Справа приведен ее упрощенный вид в виде двух ключей. Верхний подключен к питанию Vdd, нижний к земле Vss (аналог GND).

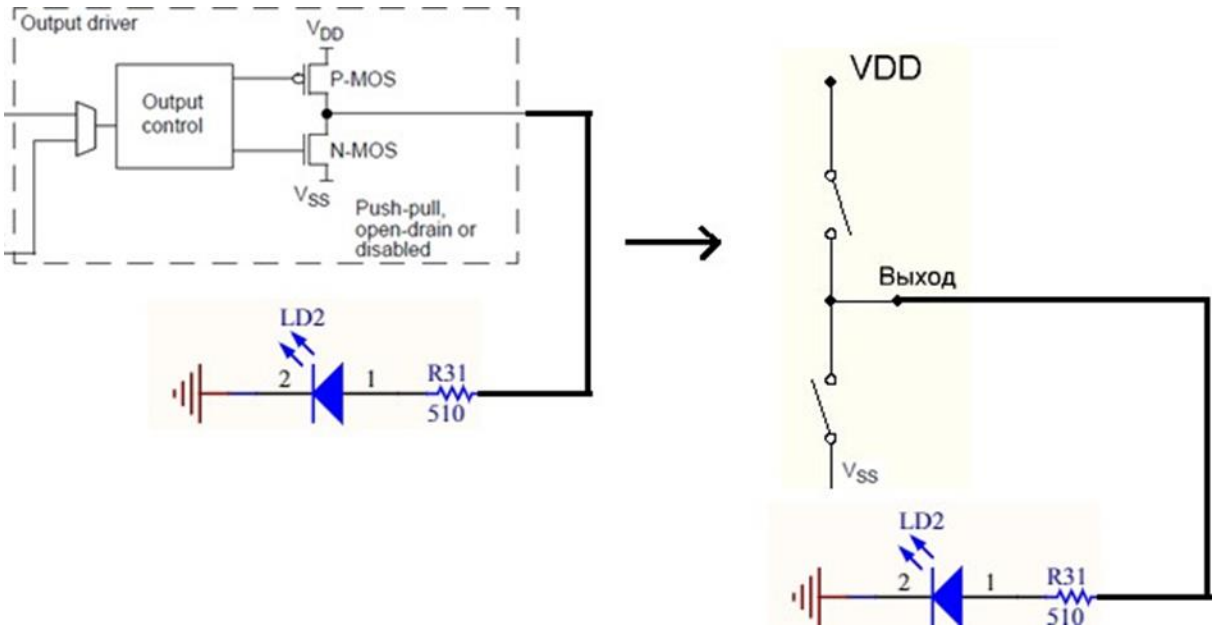


Рисунок 8.24 – Детальная схема push-pull

При подаче на выход порта логической единицы, верхний ключ замыкается, а нижний размыкается. Таким образом, от Vdd через замкнутый ключ и резистор R31 пойдет ток и зажжется светодиод LD2. При подаче логического нуля, верхний ключ закроется, нижний откроется, и шина порта будет притянута к земле Vss, т.е. ток не пойдет и светодиод не загорится.

В чем же отличие open-drain от push-pull и почему при open-drain светодиод на плате Nucleo-F103RB не горит?

При настройке режима open-drain, верхний транзистор становится закрытым, т.е. по сути блокирует доступ к источнику питания для протекания тока по шине (по сути, будто этой части совсем нет, есть только нижний транзистор или в эквивалентной схеме нижний ключ, подключенный к земле. рис. 8.25).

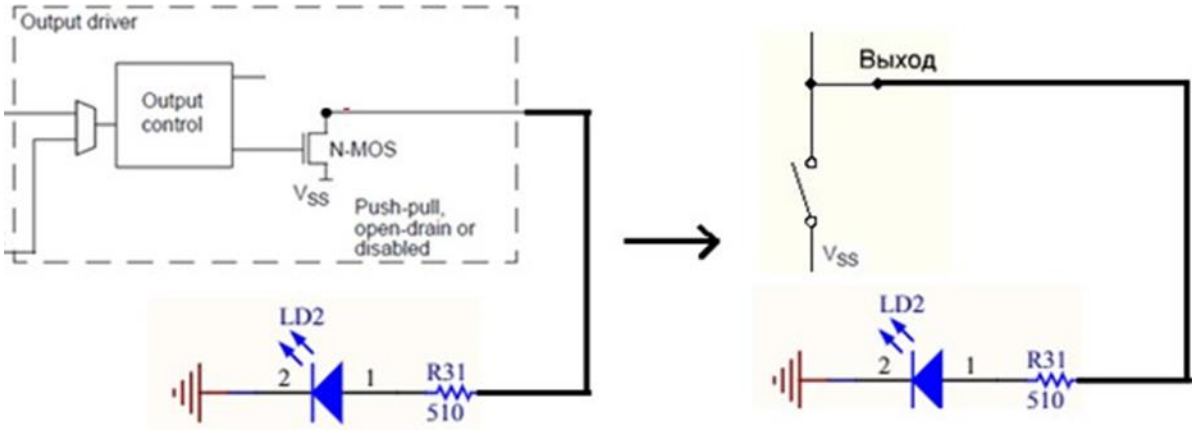


Рисунок 8.25 – Детальная схема open-drain

В таком случае, при нашей схеме подключения светодиода, току просто не откуда взяться, т.е. нет источника напряжения, от которого через наш резистор R31 пошел бы ток и зажег светодиод. Поэтому при подаче любого уровня 0 или 1, светодиод не горит, т.к. даже открыв транзистор N- MOS, шина будет притянута к земле, а светодиод тоже подключен к земле, т.е. источника напряжения нет.

Чтобы светодиод горел при open-drain, нужно его включить инверсно и его анод подключить к питанию (рис. 8.26).

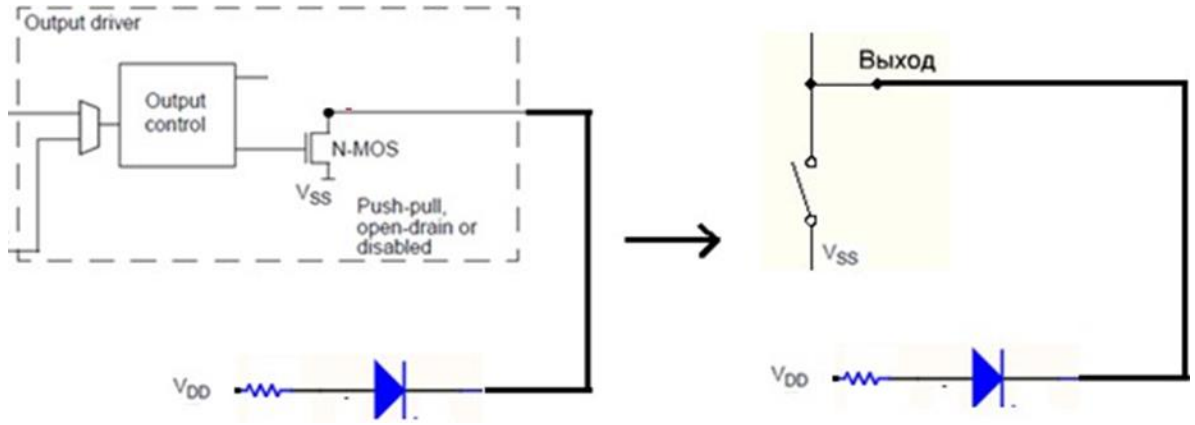


Рисунок 8.26 – Подключение светодиода для работы в режиме open-drain

Тогда, при открытии транзистора N-MOS, шина порта будет притянута к земле (Vss) и ток пойдет от Vdd через резистор и светодиод на землю Vss, таким образом, светодиод зажжется. Если транзистор N-MOS будет закрыт, то ток от Vdd не потечёт по цепи и светодиод гореть не будет.

По такой схеме (open-drain) мы как раз работаем с I2C (рис. 8.27) или 1-Wire, где шина данных притянута к питанию через резистор. Т.е. есть внешний источник питания для шины, поэтому порт только размыкает и замыкает ключ, подключенный к земле (транзистор N-MOS), чтобы ток от Vdd либо шел, либо не шел, формируя на шине, таким образом, логический нуль или единицу.

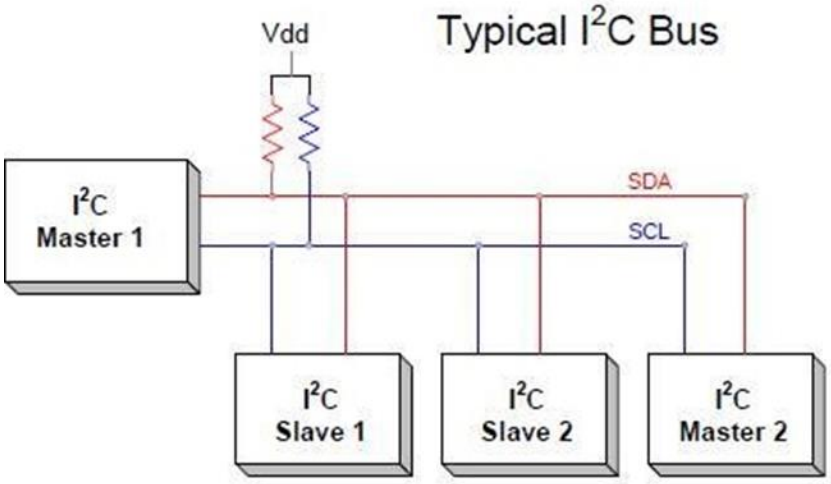


Рисунок 8.27 – Схема подключения интерфейса I2C

Настройка GPIO в STM32.

Для того, чтобы записывать данные в порт или считывать данные с порта необходимо разрешить его тактирование.

Разрешение тактирования портаосуществляется через регистр **RCC_APB2ENR**.

Например, чтобы разрешить тактирование портаA (Port A), необходимо установить логическую единицу в поле **IOPAEN**. Сделать это можно, записав в регистр **RCC_APB2ENR** (рис. 8.28) значение: 00000000000000000000000000000100 или 0x4 или 1 << 2.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										TIM11 EN	TIM10 EN	TIM9 EN	Reserved		
										rw	rw	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3 EN	USART 1EN	TIM8 EN	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	IOPG EN	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

Рисунок 8.28 – Регистр RCC_APB2ENR

Чтобы настроить режим работы пинов порта, необходимо записать определенные значения в регистр **GPIOA_CRL** для пинов 0-7 и **GPIO_CRH** для пинов 8-15 (рис. 8.29-8.30). Для настройки каждого пина используется 4 бита: CNFx[1:0] и MODEx[1:0].

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 8.29 – Регистр GPIOA_CRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 8.30 – Регистр GPIOA_CRH

Далее приведено описание полей регистров GPIO_CRL и GPIO_CRH (рис. 8.31).

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 8 .. 15)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table*.
In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved
In output mode (MODE[1:0] > 00):
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 8 .. 15)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table*.
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

Рисунок 8.31 – Описание полей регистров

Чтобы считать данные с порта x, необходимо обратиться к регистру **GPIOx_IDR** (рис. 8.32).

Биты IDR0 – IDR15 соответствуют входным пинам 0-15 порта x. x = GPIOA->IDR; y = (GPIOA->IDR & 0x20) >> 5.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Рисунок 8.32 – Регистр GPIO_IDR

Чтобы записать данные в порт x, необходимо записать значение в регистр **GPIOx_ODR** (рис. 8.33).

Биты ODR0 – ODR15 соответствуют выходным пинам 0-15 порта x. GPIOA->ODR = x; GPIOA->ODR |= 0x20; GPIOA->ODR&= ~0x20.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 8.33 – Регистр GPIO_ ODR

Другим способом записи значений н выход порта является использование регистра **GPIOx_BSRR** (рис. 8.34).

Запись **логической единицы** в поля BS0 – BS15 установит значение логической единицы на соответствующих выходах 0-15 порта x.

Запись **логической единицы** в поля BR0 – BR15 установит значение логического нуля на соответствующих выходах 0-15 порта x.

GPIOA->BSRR = 1 << 5 // PA5 = 1.

GPIOA->BSRR = 1 << 21 // PA5 = 0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Рисунок 8.34 – Регистр GPIO_BSRR

Принцип работы данного регистра напоминает работу RS – триггера (рис. 8.35). Только в данном случае, состояние R = 1, S = 1 не является запрещенным, а оговаривается документацией (BS имеет приоритет).

Bits 31:16 **BRy:** Port x Reset *bit y (y= 0 .. 15)*
These bits are write-only and can be accessed in Word mode only.
0: No action on the corresponding ODRx bit
1: Reset the corresponding ODRx bit
Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BSy:** Port x *Set bit y (y= 0 .. 15)*
These bits are write-only and can be accessed in Word mode only.
0: No action on the corresponding ODRx bit
1: Set the corresponding ODRx bit

Рисунок 8.35 – Описание полей регистра

8.3 Создание проекта в IDE Keil 5

Рассмотрим процесс создания проекта в keil 5.

Выберем Project->New mVision Project и зададим директорию для проекта (рис. 8.36). Затем выберем устройство (рис. 8.37).

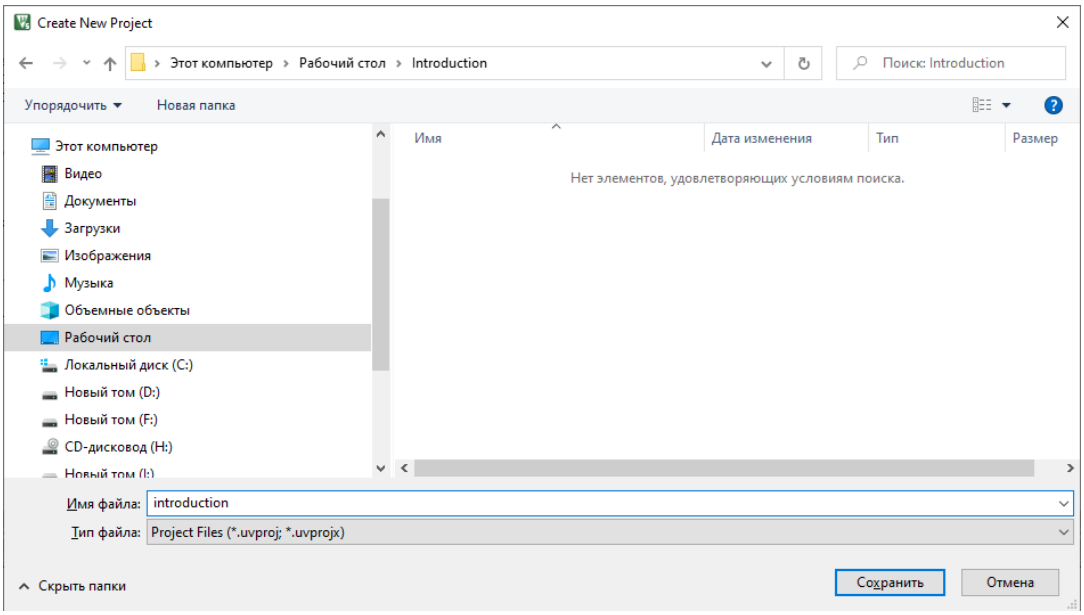


Рисунок 8.36 – Выборе директории проекта

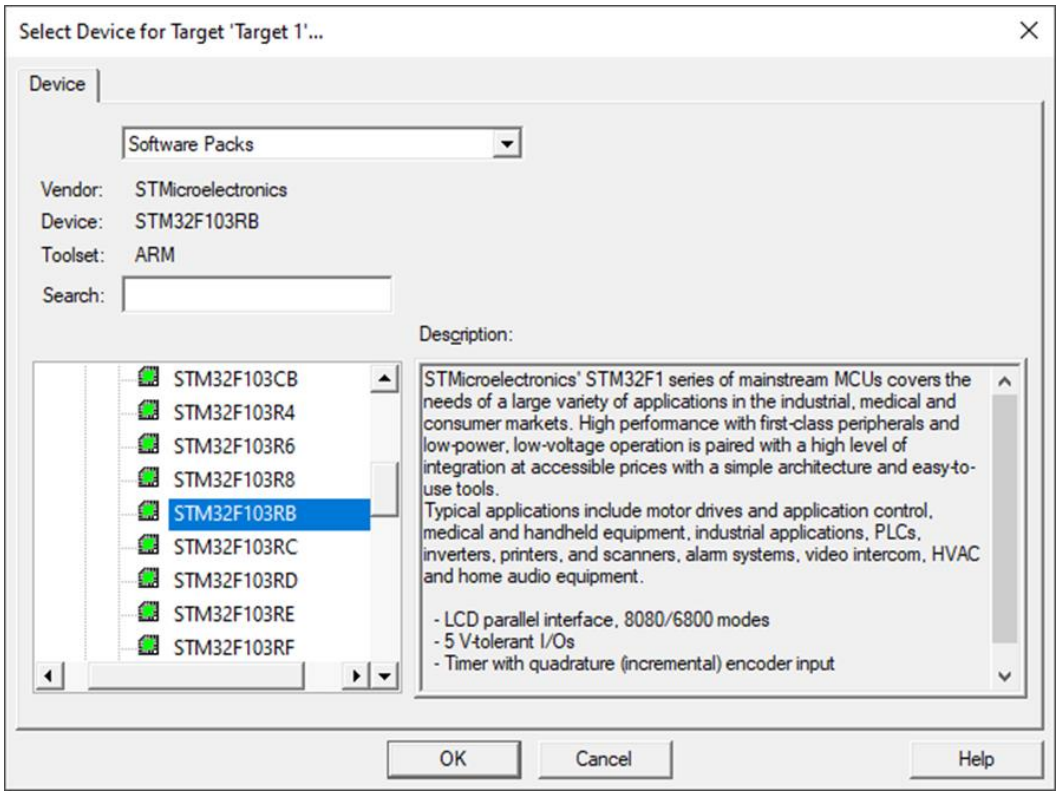


Рисунок 8.37 – Выбор устройства

В настройка среды выполнения выберем пункты, согласно рисунку 8.38.

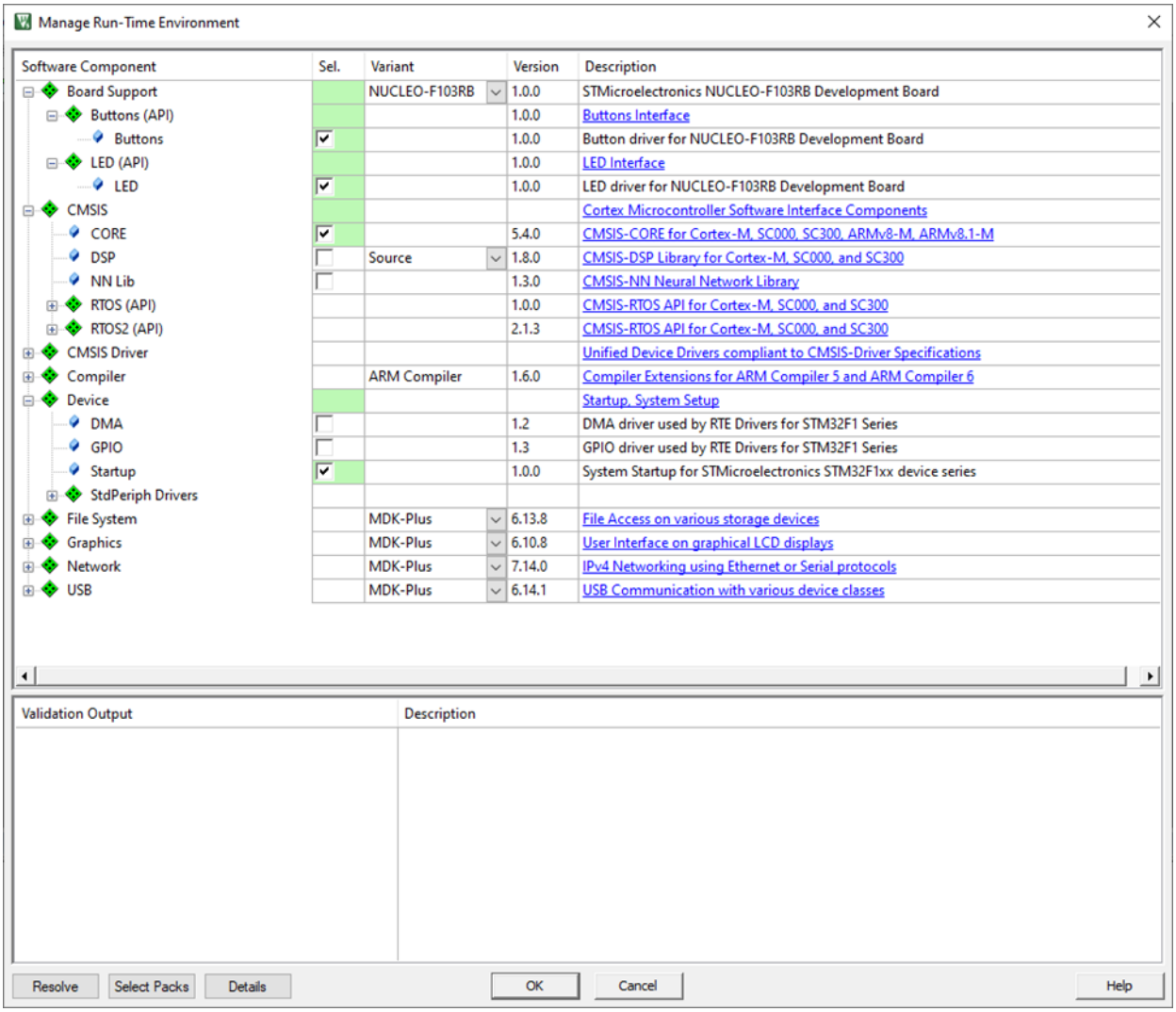


Рисунок 8.38 – Настройка среды выполнения

Board Support позволяет использовать готовые библиотеки для работы с периферией конкретной отладочной платы. Выберем кнопки и светодиоды.

CMSIS (Common Microcontroller Software Interface Standard) это независимый от производителя уровень аппаратной абстракции для серии ядер Cortex-M, а также интерфейс отладчика (debugger). CMSIS предоставляет последовательные и простые интерфейсы для ядра, его периферии и операционных систем реального времени.

CMSIS-CORE – API для ядра Cortex-M и периферии. Эта часть предоставляет стандартизированный интерфейс для Cort ex-M0, Cortex-M3, Cortex-

M4, SC000, и SC300. Включает в себя также дополнительные SIMD- инструкции для Cortex-M4.

Device->Startup – включить файл настроек инициализации для микроконтроллеров STM32F1xx.

После применения настроек и создания проекта появится дерево проекта (рис. 8.39). Для написания программ необходимо добавить файлы исходного кода (рис. 8.40-8.41).

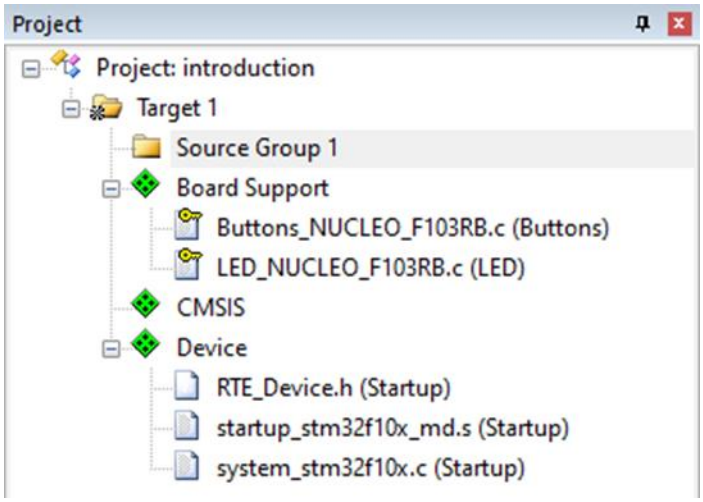


Рисунок 8.39 – Структура проекта

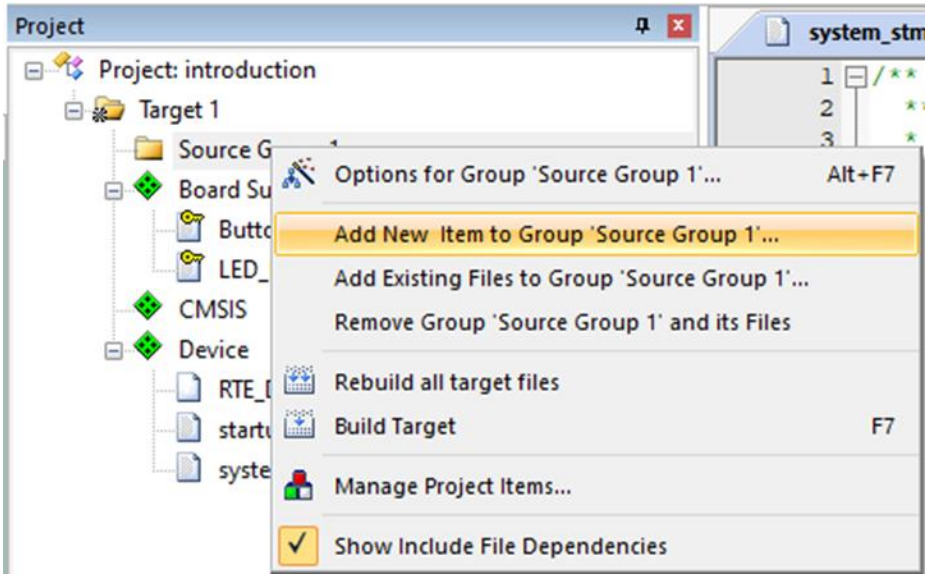


Рисунок 8.40 – Добавление файлов исходного кода

Добавим файл main.c в проект и напишем небольшую программу для мигания светодиодом с заданной задержкой на отладочной плате.

main.c | stm32f10x.h | startup_stm32f10x_md.s | system_stm32f10x.c | core_cm3.h | LED_NUCLEO_F103RB.c

```
#include "stm32f10x.h"
#include <stdio.h>
#include "Board_LED.h"                  // ::Board Support:LED

volatile uint32_t msTicks;                              // counts 1ms timeTicks
/*----------------------------------------------------------------------------
 * SysTick_Handler:
 *----------------------------------------------------------------------------*/
void SysTick_Handler(void) {
  msTicks++;
}

/*----------------------------------------------------------------------------
 * Delay: delays a number of Systicks
 *----------------------------------------------------------------------------*/
void Delay (uint32_t dlyTicks) {
  uint32_t curTicks;

  curTicks = msTicks;
  while ((msTicks - curTicks) < dlyTicks) { __NOP(); }
}


/*----------------------------------------------------------------------------
 * main: blink LED and check button state
 *----------------------------------------------------------------------------*/
int main (void) {
  int32_t max_num = LED_GetCount();
  int32_t num = 0;

  LED_Initialize();

  SysTick_Config(SystemCoreClock / 1000);               // SysTick 1 msec interrupts

  for (;;) {
    LED_On(num);                                        // Turn specified LED on
    Delay(500);                                         // Wait 500ms
    LED_Off(num);                                       // Turn specified LED off
    Delay(500);                                         // Wait 500ms

    num++;                                              // Change LED number
    if (num >= max_num) {
      num = 0;                                          // Restart with first LED
    }
  }
}
```

Рисунок 8.41 – Программа мигания светодиодом в keil

Для того, чтобы настроить свою частоту, используя настроечные регистры, необходимо добавить функцию настройки SystemCoreClockConfigure. Реализация функции представлена на рисунке 8.42.

```
/*----------------------------------------------------------------------------
  * SystemCoreClockConfigure: configure SystemCoreClock using HSI
                              (HSE is not populated on Nucleo board)
  *----------------------------------------------------------------------------*/
void SystemCoreClockConfigure(void) {

  RCC->CR |= ((uint32_t)RCC_CR_HSION);                     // Enable HSI
  while ((RCC->CR & RCC_CR_HSIRDY) == 0);                  // Wait for HSI Ready

  RCC->CFGR = RCC_CFGR_SW_HSI;                             // HSI is system clock
  while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_HSI);  // Wait for HSI used as system clock

  FLASH->ACR  = FLASH_ACR_PRFTBE;                          // Enable Prefetch Buffer
  FLASH->ACR |= FLASH_ACR_LATENCY;                         // Flash 1 wait state

  RCC->CFGR |= RCC_CFGR_HPRE_DIV1;                         // HCLK = SYSCLK
  RCC->CFGR |= RCC_CFGR_PPRE1_DIV2;                        // APB1 = HCLK/2
  RCC->CFGR |= RCC_CFGR_PPRE2_DIV1;                        // APB2 = HCLK

  RCC->CR &= ~RCC_CR_PLLON;                                // Disable PLL

  //  PLL configuration:  = HSI/2 * 12 = 48 MHz
  RCC->CFGR &= ~(RCC_CFGR_PLLSRC | RCC_CFGR_PLLXTPRE | RCC_CFGR_PLLMULL);
  RCC->CFGR |=  (RCC_CFGR_PLLSRC_HSI_Div2 | RCC_CFGR_PLLMULL12);

  RCC->CR |= RCC_CR_PLLON;                                 // Enable PLL
  while((RCC->CR & RCC_CR_PLLRDY) == 0) __NOP();           // Wait till PLL is ready

  RCC->CFGR &= ~RCC_CFGR_SW;                               // Select PLL as system clock source
  RCC->CFGR |=  RCC_CFGR_SW_PLL;
  while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL);  // Wait till PLL is system clock src
}
```

Рисунок 8.42 – Функция настройки частоты

Вызов данной функции необходимо добавить в функцию main (рис. 8.43). Также необходимо вызвать функцию SystemCoreClockUpdate для того, чтобы обновить параметр SystemCoreClock. Этот параметр будет являться частотой, на основе которой можно организовывать таймер для реализации задержки.

```
int main (void) {
  int32_t max_num = LED_GetCount();
  int32_t num = 0;

  SystemCoreClockConfigure();                              // configure HSI as System Clock
  SystemCoreClockUpdate();

  LED_Initialize();

  SysTick_Config(SystemCoreClock / 1000);                  // SysTick 1 msec interrupts

  for (;;) {
    LED_On(num);                                           // Turn specified LED on
    Delay(500);                                            // Wait 500ms
    LED_Off(num);                                          // Turn specified LED off
    Delay(500);                                            // Wait 500ms

    num++;                                                 // Change LED number
    if (num >= max_num) {
      num = 0;                                             // Restart with first LED
    }
  }
}
```

Рисунок 8.43 – Вызов функции настройки частоты и обновление SystemCoreClock

Для прошивки и отладки программы необходимо в настройках проектах в разделе Debug выбрать ST-Link Debugger (рис. 8.44). ST-Link – это внутрисхемный программатор-отладчик, позволяющий прошивать исполняемые файлы в микроконтроллер и осуществлять внутрисхемную отладку (debug) кода.

Написанную программу необходимо скомпилировать (F7) и загрузить в микроконтроллер (F8) (рис. 8.45, кнопка LOAD).

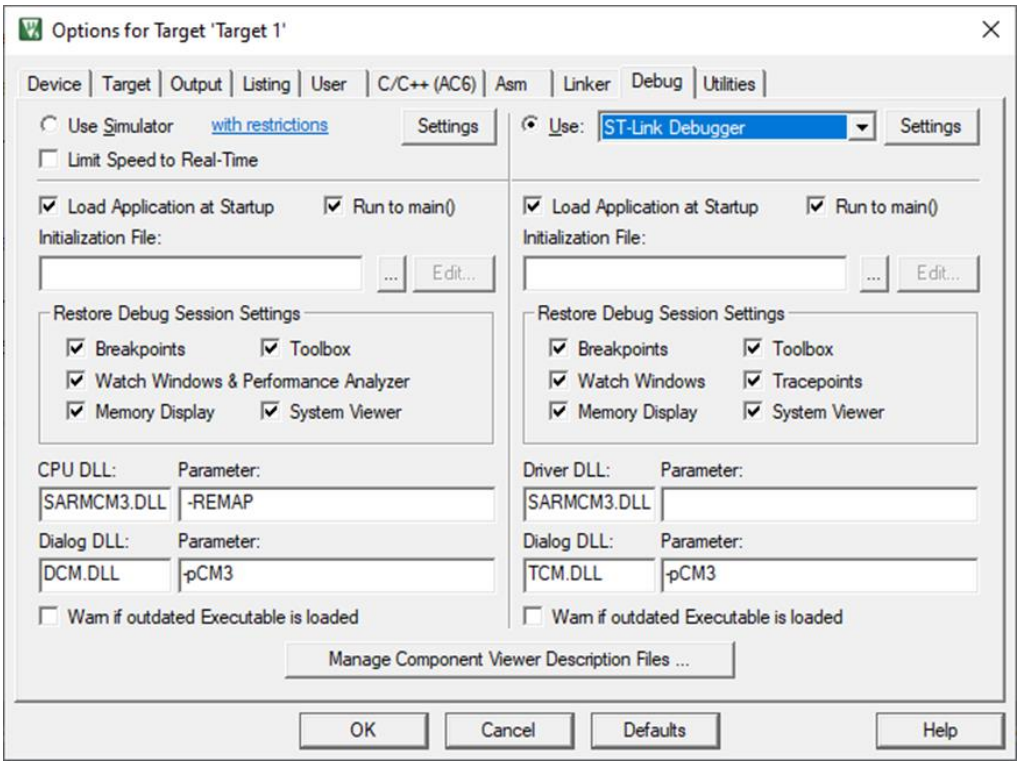


Рисунок 8.44 – Выбор отладчика

Рисунок 8.45 – Прошивка микроконтроллера

Для пошаговой отладки программы можно перейти в Start/Stop Debug Session (Ctrl + F5, Рис. 8.46). Нажимая клавишу F10 можно пройти программу по шагам (без захода в тело функций). Если необходимо зайти в тело функции, можно нажать F11.

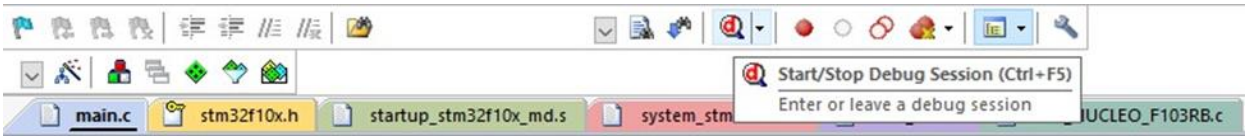


Рисунок 8.46 – Режим отладки

При отладке можно посмотреть значения переменных и содержимое регистров микроконтроллера (рис. 8.47-8.48), что помогает при поиске ошибок и изучении микроконтроллера.

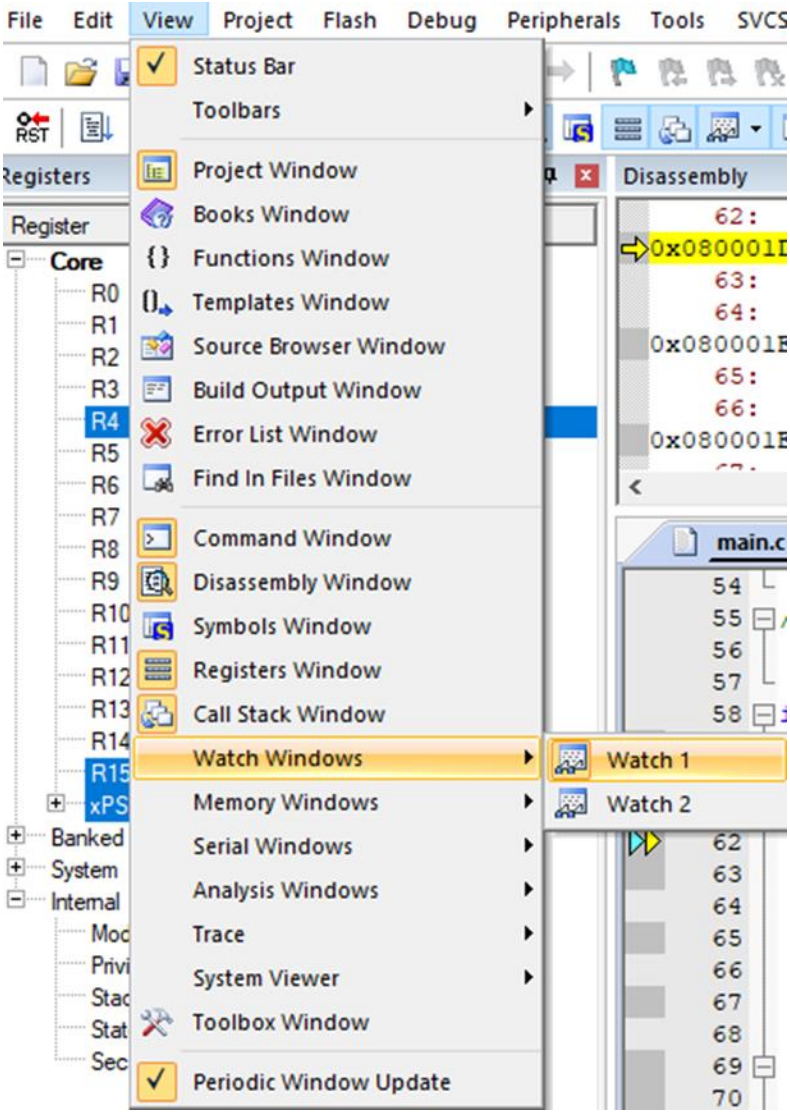


Рисунок 8.47 – Выбор окна для просмотра значений переменных

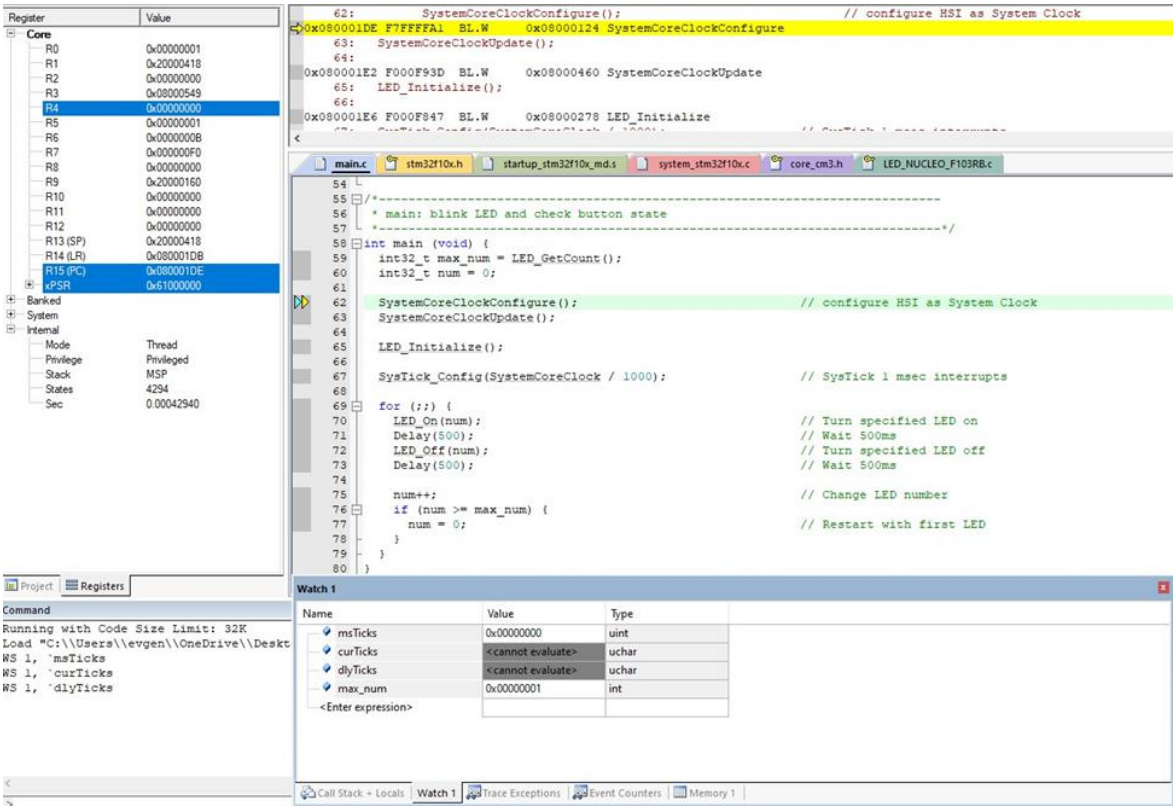


Рисунок 8.48 – Пошаговая отладка программы

Работа с портами в режиме симуляции.

Для работы корректной микроконтроллера в режиме симуляции необходимо в настройках во вкладке «Debug» прописать в поле Dialog DLL название библиотеки DARMSTM.DLL, а в поле Parameter значение ключа: -pSTM32F103RB (рис. 8.49).

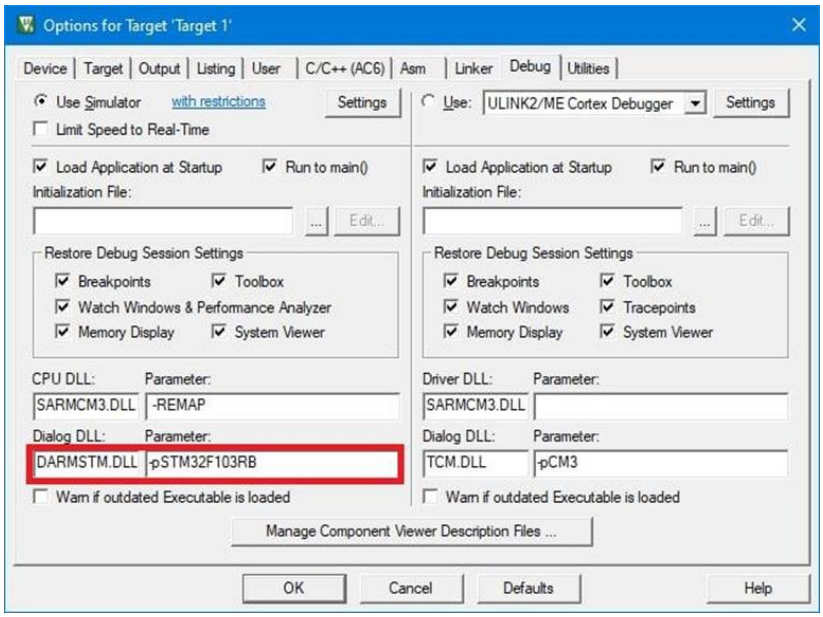


Рисунок 8.49 – Настройка параметров для симуляции микроконтроллера

Перед тем, как загружать программу в отладочную плату, можно посмотреть результаты работы в симуляторе keil. Для просмотра формируемого сигнала на выходе порта А, необходимо в режиме отладки выбрать логический анализатор (Logic Analyzer, рис. 8.50). После выбора анализатора, он должен появится в верхней части над исходным кодом (рис. 8.51).

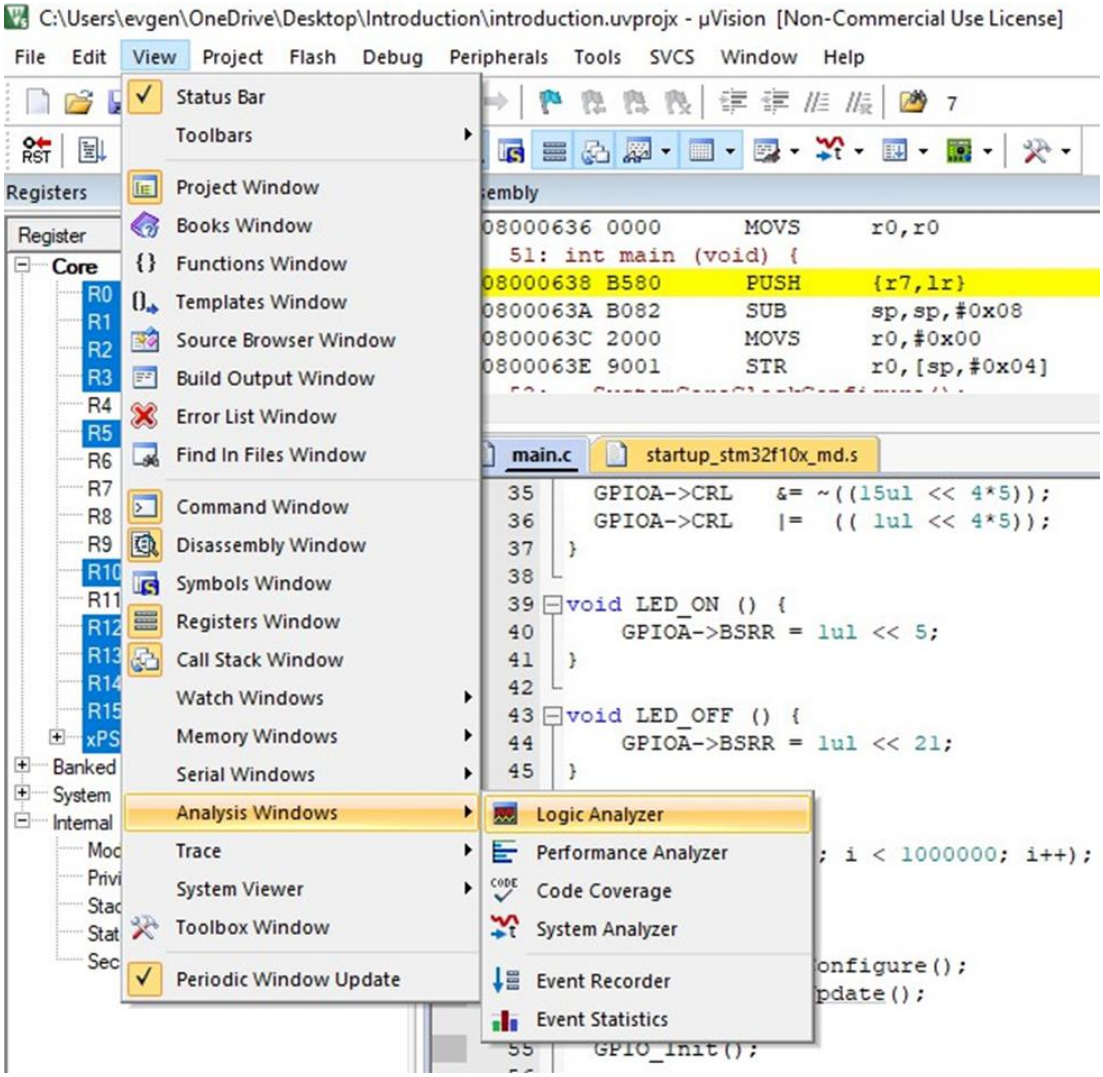


Рисунок 8.50 – Выбор логического анализатора

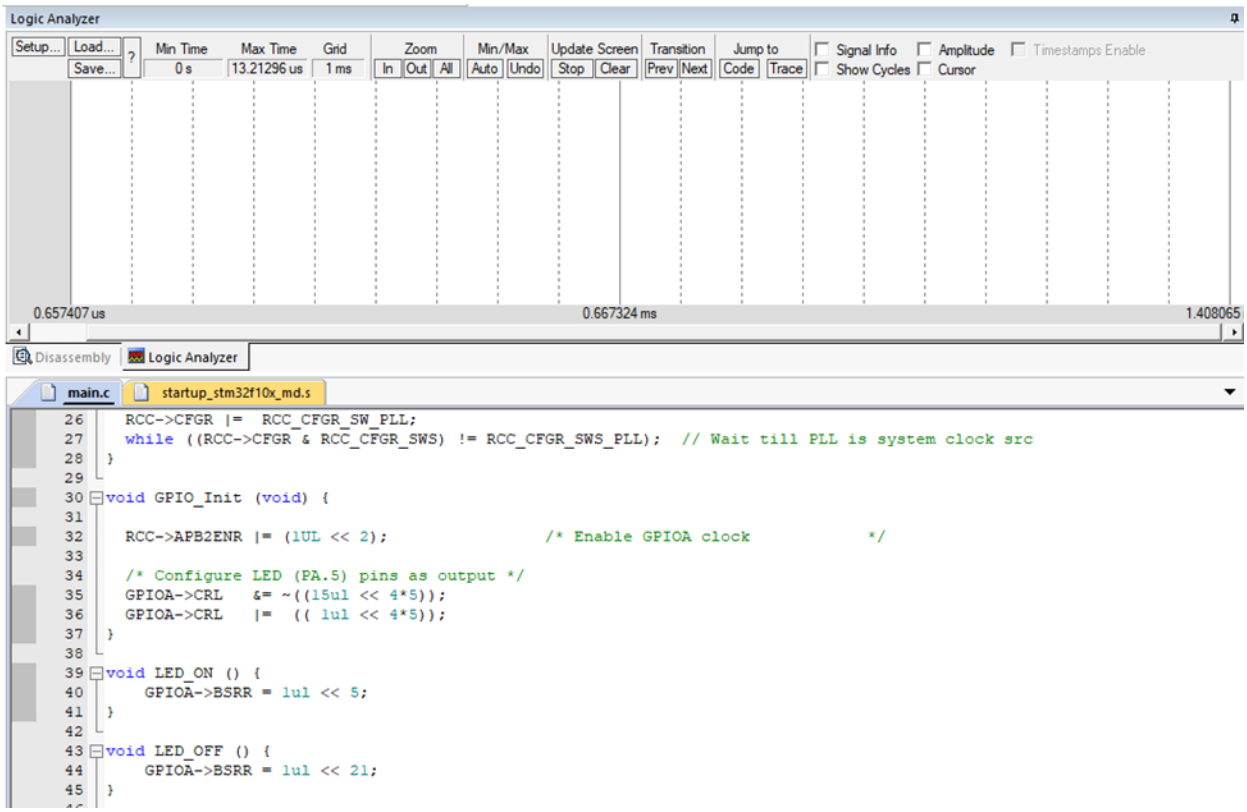


Рисунок 8.51 – Логический анализатор keil

Также значения на выходе/входе порта можно просматривать через Peripherals -> General Purpose-> GPIOA (рис. 8.52).

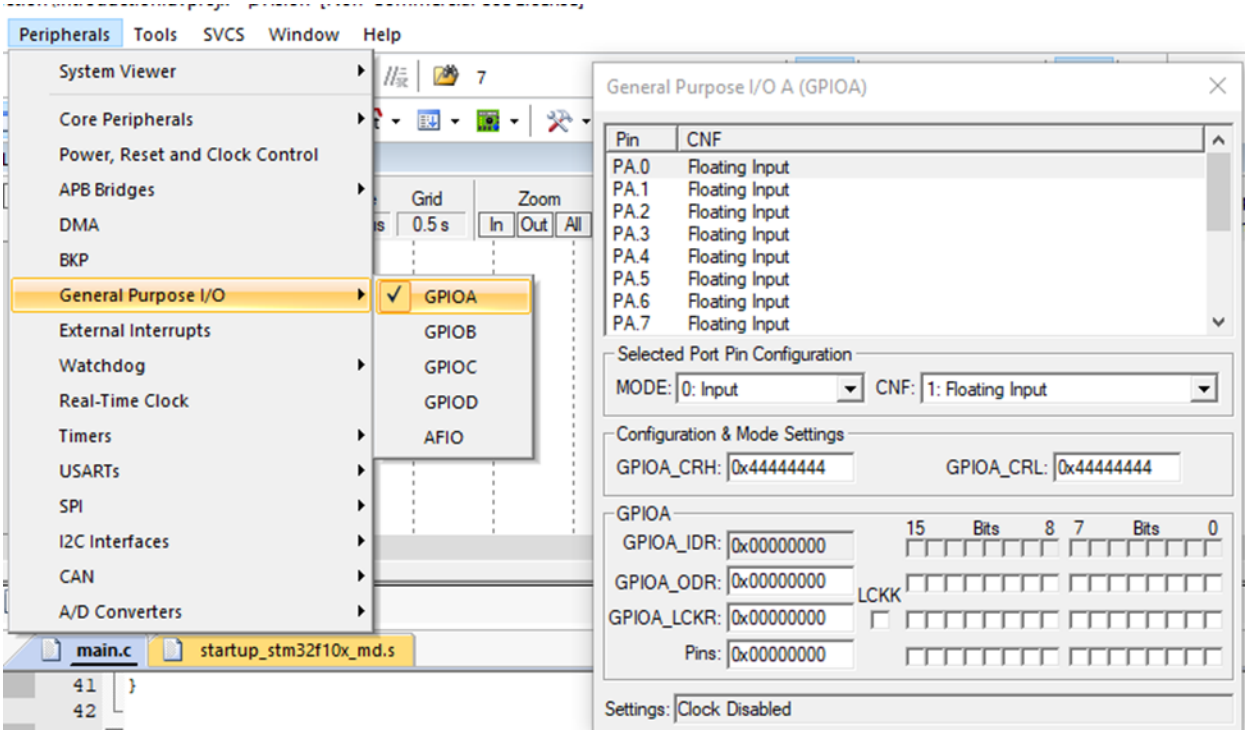


Рисунок 8.52 – Просмотр значений порта A

Для добавления источников сигналов в логический анализатор необходимо выбрать Setup и в появившемся окне добавить новых сигнал через опцию New Insert (рис. 8.53).

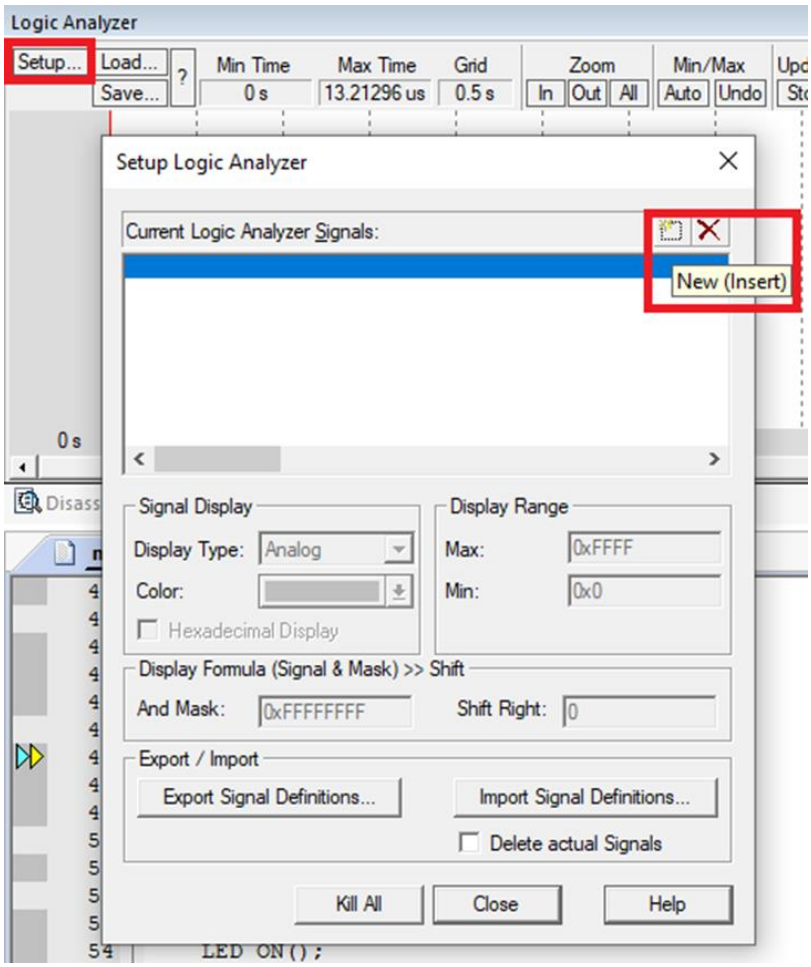


Рисунок 8.53 – Добавления источника сигнала в логический анализатор

Затем надо прописать с какого именно контакта нужно отобразить сигнал. Для этого пропишем выражение (PORTA & 0x20) >> 5, которое позволит отобразить значение с 5-го контакта порта A (рис. 8.54).

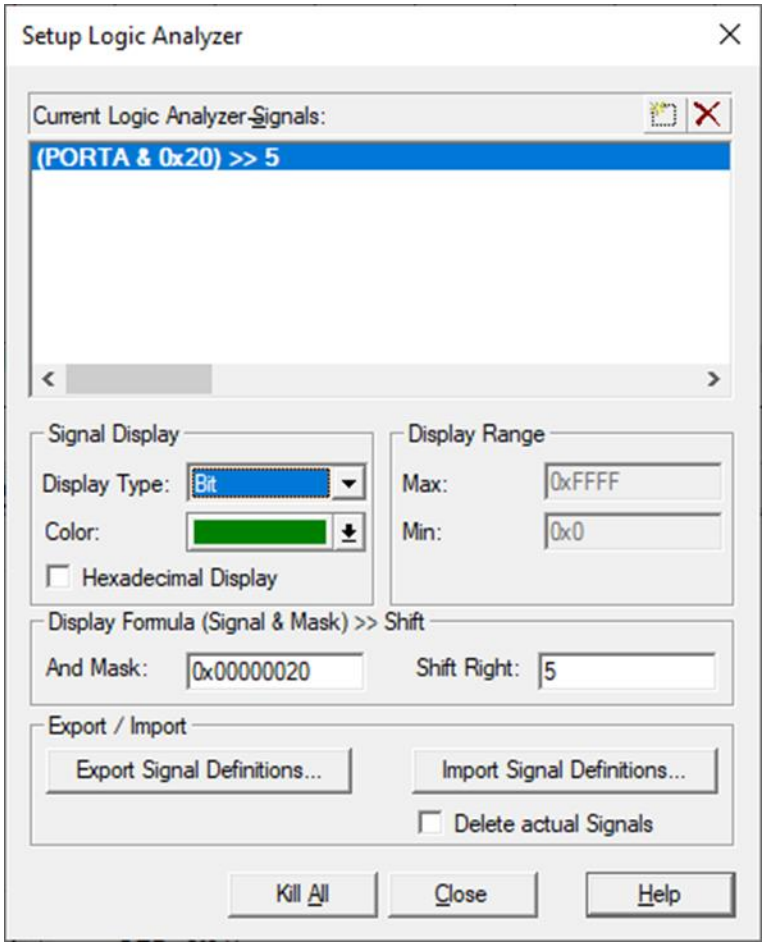


Рисунок 8.54 – Выбор конкретного контакта для анализатора

После запуска программы, можно увидеть сигнал на выходе PORTA.5 в логическом анализаторе и окне GPIO (рис. 8.55). Согласно программе по прерыванию таймера сигнал устанавливается в уровень логической единицы, что видно по логическому анализатору. По заданию необходимо реализовать изменение сигнала с заданной частотой. На рисунке 8.56 приведен пример сигнала с частотой 1 Гц (период = 1 сек). Период сигнала (и его частоту) можно определить, зная шаг сетки (в примере на рис. 6.8 Grid = 0.5 сек).

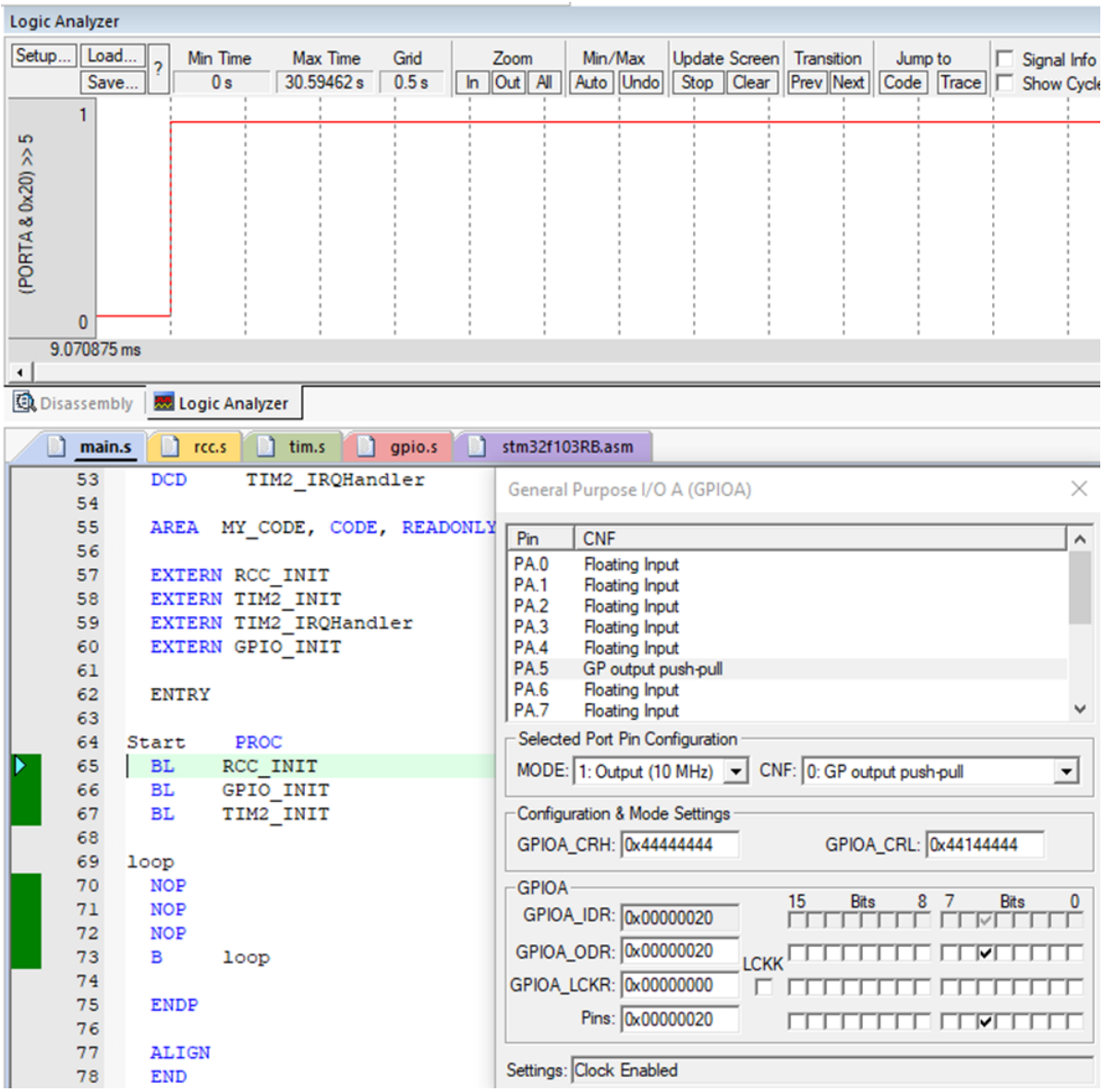


Рисунок 8.55 – Просмотр сигнала в логическом анализаторе и окне GPIO

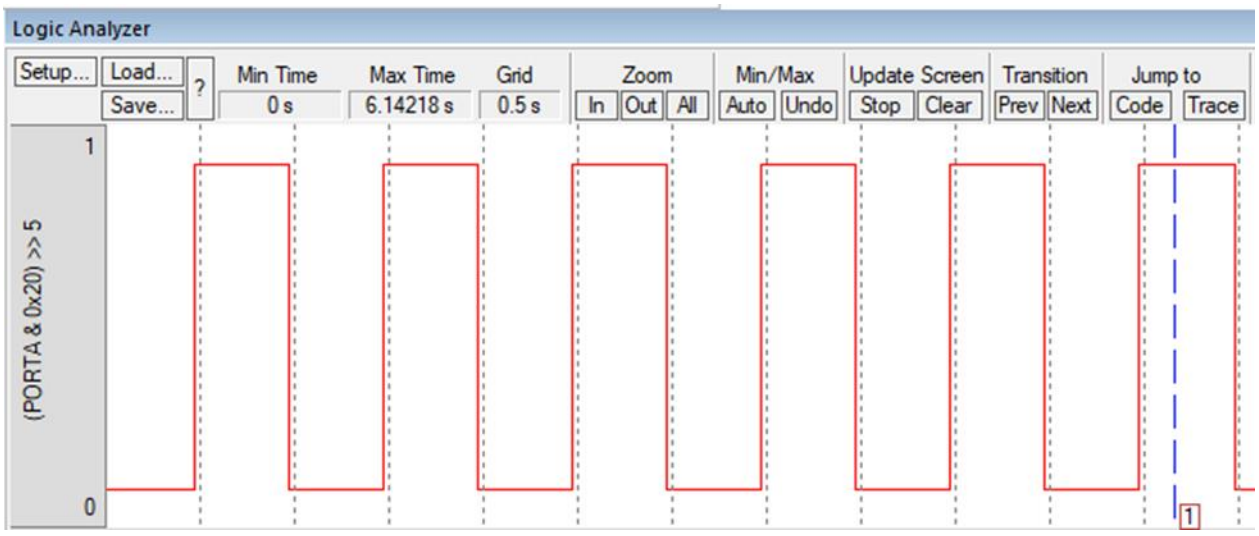


Рисунок 8.56 – Пример сигнала с частотой 1 Гц

Работа с отладочной платой.

После проверки программы в симуляторе ее можно загрузить в отладочную плату. Для этого в настройках проекта во вкладке Debug необходимо выбрать отладчик ST-Link Debugger (рис. 8.57).

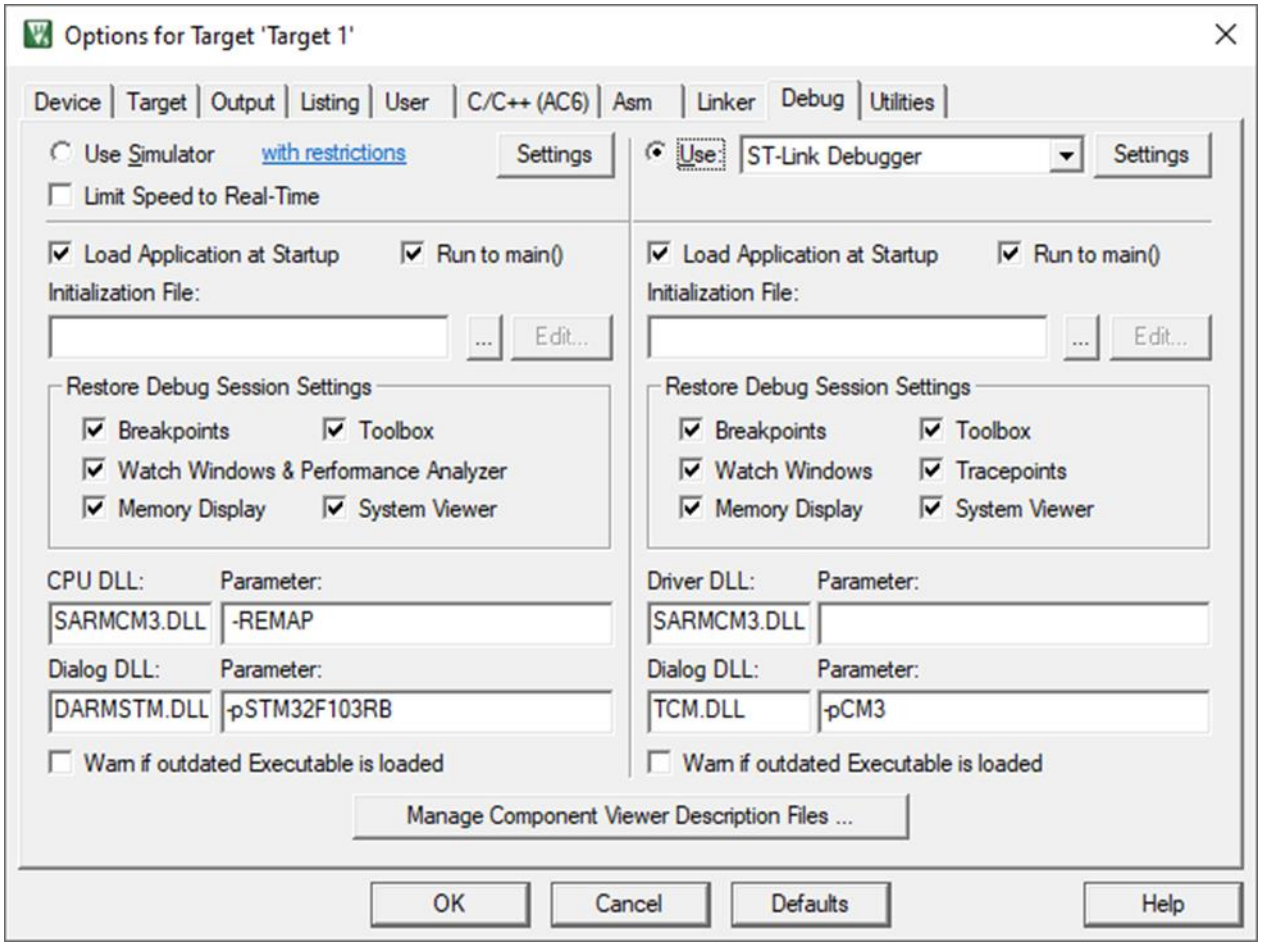


Рисунок 8.57 – Выбор отладчика ST-Link

Подключите отладочную плату к компьютеру по USB-порту. Убедитесь, что устройство обнаружено операционной системой в диспетчере устройств (рис. 8.58). При нажатии Start/Stop Debug Session программа загрузится в отладочную плату. Выполнить программу в режиме отладки можно нажав F5. При успешном выполнении примера программы должен загореться светодиод LD2 (рис. 8.59).

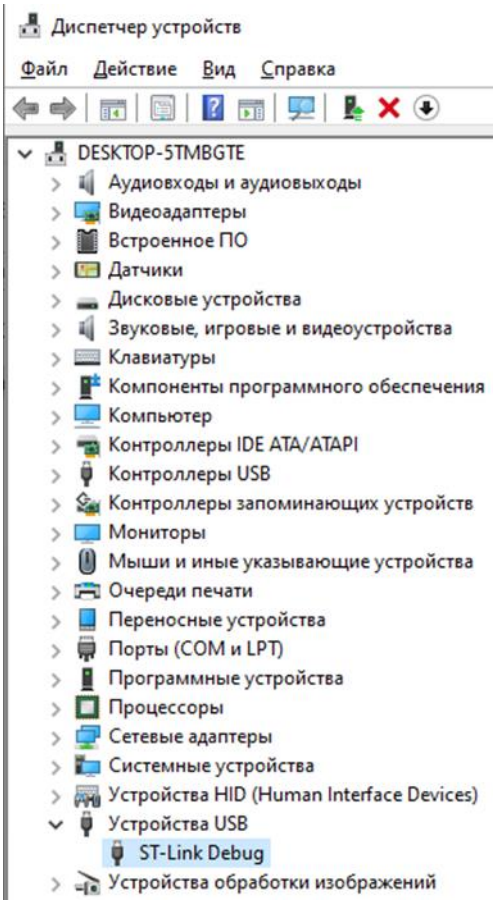


Рисунок 8.58 – ST-Link в диспетчере устройств

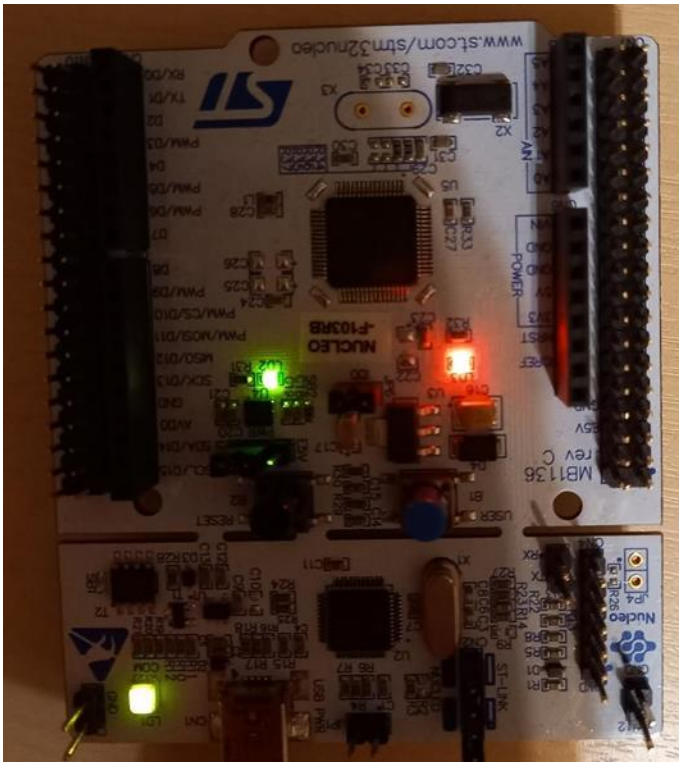


Рисунок 8.59 – Выполнение программы на Nucleo-F103RB

9 Лабораторная работа №2. Получение практических навыков работы с интерфейсом UART на основе CMSIS (Common Microcontroller Software Interface Standard)

9.1 Цели работы

Реализовать настройку регистров, скорости передачи данных, формата кадра данных. Написать программу приемника/передачтика данных по интерфейсу UART.

Задачи:

1. Разработать библиотеку для работы с интерфейсом UART STM32. С помощью специальной программы-терминала ввести и отправить с персонального компьютера (ПК) на микроконтроллер свою фамилию и имя. На микроконтроллере к полученной строке добавить номер группы (через пробел) и передать новую строку обратно на ПК.
2. Разработать программу на языке Си, которая принимает с ПК (используя специальную программу-терминал) по последовательному порту два целых положительных числа (разделенные пробелами), выполняет с ними операцию, согласно варианту задания (табл. 9.1) и отправляет результат обратно на ПК. Значения чисел могут быть любыми, размером не более 4 байт. Программа должна поддерживать многократный ввод пары чисел (предыдущие значения затираются) и выполнение операции. Скорость передачи и формат кадра задать согласно варианту задания (для всех вариантов 1 старт-бит, 8 бит данных, 1 стоп-бит).
3. Настроить прием/передачу данных по UART, используя прерывания интерфейса.
4. Написать отчет.

Таблица 9.1 – Варианты заданий

№	Операция	Скорость	Бит четности
1	Инкрементировать первое число и сложить со вторым, умноженным на 2	4800	Нет
2	Декрементировать первое число и вычесть из него инвертированное второе	9600	Четность
3	Сдвинуть первое число влево на 1 бит и умножить на второе, с обменянными тетрадами.	14400	Нечетность
4	Сдвинуть первое число вправо на 1 бит и разделить на второе инкрементированное	19200	Нет
5	Сделать инверсию первого числа и операцию «логическое И» со вторым, сложенным с константой 5.	28800	Четность
6	В обоих числах обменять местами тетрады и выполнить операцию «логическое ИЛИ»	38400	Нечетность
7	Умножить первое число на 2 и выполнить операцию «исключающее ИЛИ» со вторым числом, умноженным на 3	57600	Нет
8	В первом числе обменять местами тетрады и сложить с декрементированным вторым.	115200	Четность
9	Инвертировать оба числа и сложить	128000	Нечетность
10	В первом числе обменять местами тетрадры, во втором сделать инверсию и выполнить между ними операцию «исключающее ИЛИ»	256000	Нет

9.2 Методические указания

Последовательный порт (интерфейс).

Последовательный порт предназначен для организации связи по последовательному каналу между микроконтроллером и периферийными устройствами или другими МК/МП.

UART – Universal Asynchronous Receiver-Transmitter - универсальный асинхронный приемник/передатчик - наиболее распространенный последовательный интерфейс передачи данных. Через него к микроконтроллеру могут быть подключены самые разные устройства: от миниатюрного цифрового датчика, до компьютера. На базе UART могут создаваться локальные сети.

Интерфейс называется асинхронным. Это означает, что данные могут быть переданы в любой момент, без использования между приемником и передатчиком синхронизирующих импульсов. В одну сторону данные предаются с помощью одного сигнала. Сам сигнал содержит как данные, так и синхронизирующую информацию.

USART – Universal Synchronous/Asynchronous Receiver/Transmitter (универсальный синхронный /асинхронный приемник/передатчик).

На рисунке 9.1 показано подключение устройств по интерфейсу UART:

1. Rx – прием данных;
2. Tx – передача данных.

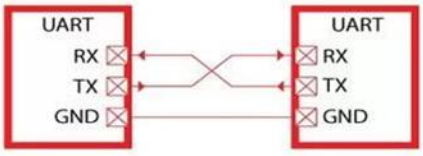


Рисунок 9.1 – Подключение устройств по UART

На рисунке 9.2 показано подключение устройств по интерфейсу USART:

1. Rx – прием данных;
2. Tx – передача данных;

3. XCK – синхронизация (тактирование).

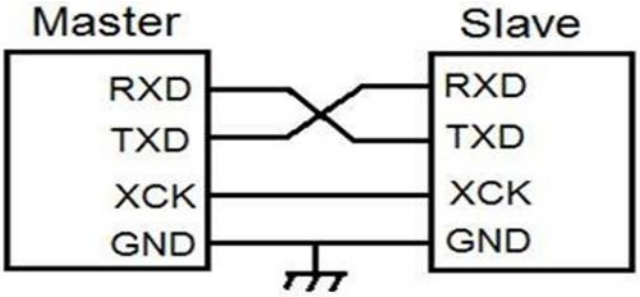


Рисунок 9.2 – Подключение устройств по USART

Асинхронность интерфейса подразумевает, что данные могут быть переданы в любой момент, без использования между приемником и передатчиком синхронизирующих импульсов.

Для того, чтобы приемник понял, что начинается передача данные сопровождаются специальными битами: **стартовым и стоповым** (рис. 9.3).

В режиме ожидания сигнал на входе приемника находится в высоком уровне. Первым передается **стартовый** бит. Он всегда имеет **низкий** уровень. По первому спаду сигнала в низкий уровень приемник начинает отсчитывать внутренние синхроимпульсы. Зная скорость передачи (период поступления входных битов), приемник считывает состояние следующих 8 битов. В конце передатчик формирует **стоп-бит**, всегда **высокого** уровня. Второй (необязательный) стоповый бит может быть настроен, как правило, на то, чтобы дать приемнику время подготовиться к следующему кадру, но на практике это используется редко.

Биты данных являются пользовательскими данными или информационными битами и идут сразу после стартового бита. Может быть от 5 до 9 бит данных, хотя чаще всего используется 7 или 8 битов. Эти биты данных обычно передаются в формате с первым младшим значащим битом.

Все биты передаются за одинаковые промежутки времени. Время передачи одного бита определяет скорость передачи интерфейса. Часто она указывается в бодах (бит в секунду при двоичном кодировании данных).

Кадр UART может также содержать дополнительный **бит четности**, который можно использовать для обнаружения ошибок. Этот бит вставляется между последним битом данных и стоповым битом. Значение бита четности зависит от типа используемого контроля четности (на четность или нечетность):

– при контроле на **четность** этот бит устанавливается таким образом, чтобы **общее количество единиц** в кадре было **четным**;

– при контроле на нечетность этот бит устанавливается таким образом, чтобы **общее количество единиц** в кадре было **нечетным**.

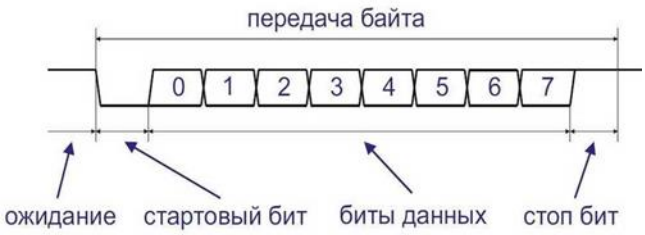


Рисунок 9.3 – Формат кадра UART

На рисунке 9.4 приведена структурная схема интерфейса U(S)ART в микроконтроллере STM32F103. Аппаратная реализация U(S)ART включает *регистр-приемник данных (RDR), регистр-передатчик данных (TDR), регистр сдвига приемника (Receive Shift Register), регистр сдвига передатчика (Transmit Shift Register), регистра статуса (SR), регистр скорости (BRR), управляющие регистры (CR1-CR3).*

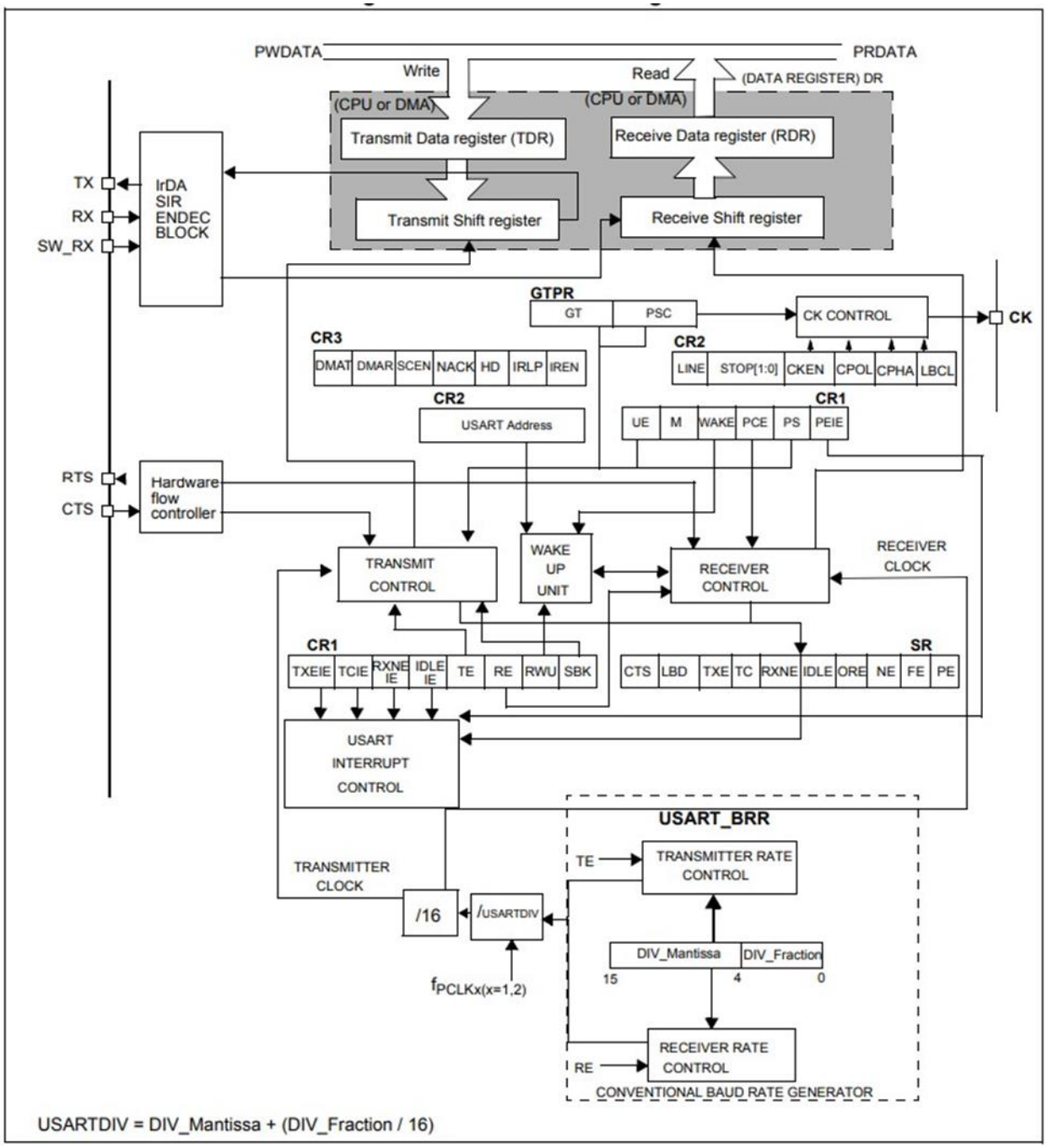


Рисунок 9.4 – Устройство U(S)ART в STM32F103

Передающая и приемная часть работают независимо друг от друга. Приемник и передатчик могут работать с разными устройствами, разными протоколами верхнего уровня и т.п. Скорость и формат данных должны быть одинаковыми.

При передаче данных, байт загружается в буферный регистр передатчика (TDR). Программно доступен только он. Если передача предыдущего байта закончена и регистр сдвига пуст, то байт перегружается в него, сдвига-

ется и побитно поступает на выход **TX**. Как только байт перегружен в регистр сдвига, буферный регистр освобождается и в него может быть загружено новый байт, который будет ждать окончания передачи и автоматически перегрузится в сдвиговый регистр.

Об окончании передачи данных регистром сдвига сообщается специальный флаг TC, об освобождении буферного регистра сообщает флаг TXE, по которому могут быть сформированы прерывания.

При приеме данных с входа **RX** данные поступают на сдвиговый регистр, сдвигаются и, после формирования полного байта, перегружаются в буферный регистр приема (RDR). Буферный регистр доступен программно. Из него считывается полученный байт. Если данные поступают сплошным потоком, без пауз, то после приема байта есть время, равное времени передачи байта, на то, чтобы считать его из буферного регистра. Иначе придет новый байт, а старый будет потерян.

Об окончании приема данных сообщает специальный флаг RXNE, по которому могут быть сформированы прерывания.

Аппаратная реализация I2C включает *регистр данных, адресный регистр, регистр двойного адреса, два управляющих регистра – CR1 и CR2, регистры статуса – SR1 и SR2, регистр, задающий частоту передачи данных или скорость (CCR), регистр PEC (Packet error checking) – регистр отслеживания ошибок*, представляет собой битовое поле, состоящее из 8 бит регистра *SR2* (биты с 8 по 15), в которое записывается контрольная сумма кадра.

9.3 Структура основных регистров U(S)ART

USART_SR – регистр состояния (рис. 9.5).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

Рисунок 9.5 – Регистр состояния

Бит 7 TXE. Буферный регистр передатчика пуст. Флаг становится равным 1 после того, как данное перегружается в регистр сдвига. Т.е. флаг сообщает о том, что буферный регистр пуст, может быть загружено новое данное. Флаг устанавливается аппаратно и сбрасывается после записи байта в буферный регистр USART_DR.

Бит 6 TC. Флаг устанавливается аппаратно и сообщает о том, что передача данного закончена, сдвиговый регистр пуст. Если предыдущий флаг (TXE) говорит о том, что можно в буферный регистр загружать новое данное. Физическая передача может продолжаться, на выходе TX возможно идет поток битов. То флаг TC сообщает, что все биты переданы. Сбрасывается флаг последовательным чтением регистров USART_SR и USART_DR.

Бит 5 RXNE. Буферный регистр приема не пуст. Флаг сообщает, что в буферном регистре приемника есть данное. Данное должно быть считано до прихода следующего, иначе оно будет потеряно. Сбрасывается флаг при чтении регистра USART_DR.

Бит 3 ORE. Ошибка переполнения. Флаг устанавливается в случае, если в приемный буферный регистр поступило новое данное, а предыдущее считано не было. Т.е. при потере данного.

Бит 2 NE. Флаг устанавливается при выделении шума во входном сигнале. Наличие шума определяется как слишком частое переключение входного сигнала. **Бит 1 FE**. Ошибка приема фрейма (кадра). Возникает, когда не был выделен стоп-бит. Т.е. после приема стартового бита и 8 битов данных на месте стопового бита UART считал сигнал низкого уровня.

Бит 0 PE. Ошибка паритета. Сигнализирует об ошибке при включенном контроле четности.

USART_DR - регистр данных (рис 9.6).

Рисунок 9.6 – Регистр данных

Используется для записи данных в буферный регистр передатчика и чтения данных из буферного регистра приемника. На самом деле это 2 регистра, для обращения к которым используется один адрес (рис. 9.7).

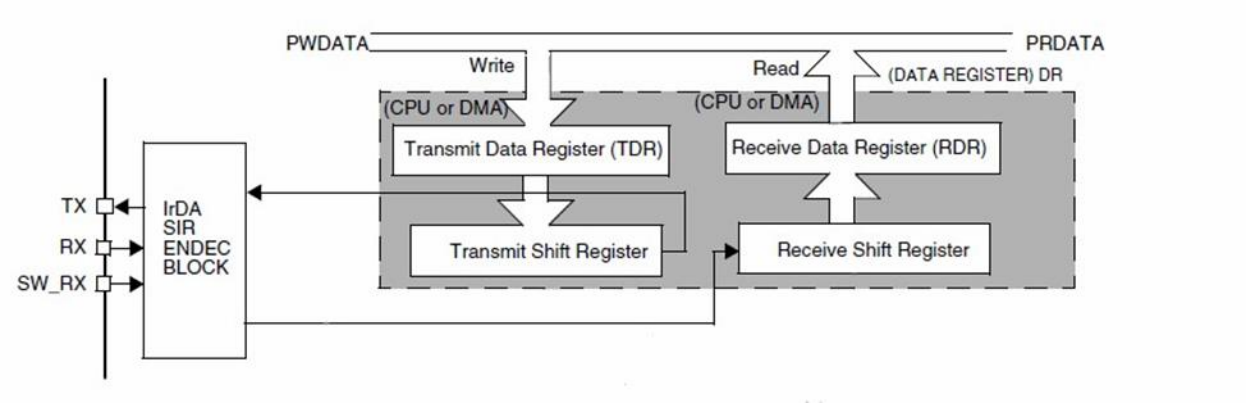


Рисунок 9.7 – Регистр данных

USART_BRR – регистр скорости (рис. 9.8 – 9.9).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 9.8 – Регистр скорости

Регистр содержит значение делителя частоты (USARTDIV), который определяет скорость передачи и приема данных.

Скорость вычисляется по формуле BAUD = Fck / (16 * USARTDIV), где

- BAUD – скорость, бод.
- Fck – входная частота тактирования UART:
- сигнал PCLK2 для UART1 (шина APB2);
- сигнал PCLK1 для UART2 и 3 (шина APB1).
- USARTDIV – значение регистра USART_BRR.

Десятичное значение USARTDIV = целая часть + (дробная часть / 16).

USARTDIV = Fck / (16 * BAUD)

- DIV_Mantissa = 78, DIV_Fraction = 5
- USARTDIV = 78 + (5 / 16) = 78,3125.
- BAUD = 72000000 / (16 * 78,3125) = 57462 бод.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Целая часть (12 разрядов)												Дробная часть (4 разр)			
11	10	9	8	7	6	5	4	3	2	1	0	-1	-2	-3	-4

Рисунок 9.9 – Регистр скорости

USART_CR1 - управляющий регистр 1 (рис. 9.10).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 9.10 – Управляющий регистр 1

Бит 13 **UE**. Разрешение работы UART. Отключение UART (UE=0) уменьшает ток потребления микроконтроллера.

Бит 12 **M**. Задает длину слова 8 бит (M=0) или 9 бит (M=1).

Бит 10 **PCE**. Разрешение контроля четности (PCE=1).

Бит 7 **TXEIE**. Разрешение прерывания по флагу TXE, т.е. когда буферный регистр передатчика пуст.

Бит 6 **TCIE**. Разрешение прерывания по флагу TC, т.е. когда пуст сдвиговый регистр передатчика.

Бит 5 **RXNEIE**. Разрешение прерывания по флагу RXNE, т.е. когда в буферном регистре приемника есть непрочитанное данное.

Бит 3 **TE**. Разрешение работы передатчика.

Бит2 **RE**. Разрешение работы приемника.

USART_CR2 – управляющий регистр 2 (рис. 9.11).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LINEN	STOP[1:0]		CLK EN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.	ADD[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Рисунок 9.11 – Управляющий регистр 2

Биты 13:12 **STOP**. Биты задают формат стопового признака слова

Примеры программ для работы с UART представлены в файлах лабораторной работы.

10 Лабораторная работа №3. Реализация логических функций

10.1 Цели работы

Использовать реализованную настройку приёмника/передатчика USART для ввода/вывода информации с использованием дополнительных функций битовых операций

Задачи:

1. Разработать функцию, изменяющий строку с четырёхбитным числом по основанию X на основание Y, исходя из вариантов в таблице 10.1. Ввод строки должен быть реализован через USART. Результат вывести на экран.

2. Разработать функцию, изменяющую строку с двумя четырёхбитными двоичными числами и выводящий результат побитовой операции, заданной по варианту в таблице 10.1. Ввод строки должен был реализован через USART. В отчёте привести соответствующую таблицу истинности.

Таблица 10.1 – Параметры вариантов

№	Основание X	Основание Y	Функция
1	2	4	И-НЕ
2	2	6	ИЛИ-НЕ
3	2	8	Исключающее ИЛИ
4	2	4	Эквиваленция
5	2	6	НЕ-И
6	10	8	НЕ-ИЛИ
7	10	4	Эквиваленция
8	10	6	Исключающее ИЛИ
9	10	8	ИЛИ-НЕ
10	10	4	И-НЕ
11	2	4	Исключающее ИЛИ
12	2	8	НЕ-ИЛИ
13	2	6	НЕ-И
14	2	6	ИЛИ-НЕ
15	2	4	Эквиваленция
16	2	8	Исключающее ИЛИ
17	2	8	И-НЕ
18	2	4	Исключающее ИЛИ
19	2	6	И-НЕ
20	10	8	Исключающее ИЛИ

Окончание таблицы 10.1

21	10	6	ИЛИ-НЕ
22	10	4	И-НЕ
23	10	8	Исключающее ИЛИ
24	10	6	И-НЕ
25	10	4	НЕ-И

Вариант выбирается исходя из номера внутри учебной группы. Для выполнения задач следует использовать разработанную в первой лабораторной библиотеку для работы с интерфейсом USART STM32.

10.2 Методические указания

Для разработки функций с логическими и арифметическими операциями следует узнать их приоритет (таблица 10.2).

Таблица 10.2 – Приоритет и описание операторов

Приоритет	**Оператор**	**Описание**
1	++	(Пост) Прединкремент
	--	(Пост) Преддекремент
	!	Логическое отрицание (NOT)
	~	Побитное отрицание (Bitwise NOT, дополнение до единицы)
2	*	Умножение
	/	Деление
	%	Взятие по модулю числа (Получение остатка от деления)
3	+	Сложение
	-	Вычитание
4	<<	Побитный сдвиг влево (Bitwise left shift)
	>>	Побитный сдвиг вправо (Bitwise right shift)
5	<	Меньше чем...
	<=	Меньше или равно...
	>	Больше чем...
	>=	Больше или равно...
6	==	Оператор определения равенства (Equal to)
	!=	Оператор определения неравенства (Not equal to)
7	&	Побитная операция И (Bitwise AND)
8	^	Побитная операция Исключающее ИЛИ (Bitwise XOR, exclusive or)

Продолжение таблицы 10.2

Приоритет	Оператор	Описание
9	|	Побитная операция ИЛИ (Bitwise OR, inclusive or)
10	&&	Логическая операция И (AND)
11	||	Логическая операция ИЛИ (OR)

Настройку системных часов можно выставить по умолчанию на HSE (рисунок 10.1) или оставить так, как они были в первой лабораторной. Аналогично следует поступить с инициализацией USART (рисунок 10.2), но в данном случае предлагается не включить обработчик прерываний в USART, а реализовывать алгоритм синхронно.

```
23  void SystemCoreClockConfigure(void) {
24      RCC->CR |= ((uint32_t)RCC_CR_HSEON);
25      while ((RCC->CR & RCC_CR_HSERDY) == 0);
26
27      RCC->CFGR = RCC_CFGR_SW_HSE;
28      while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_HSE);
29
30      RCC->CFGR = RCC_CFGR_HPRE_DIV1;
31      RCC->CFGR |= RCC_CFGR_PPRE1_DIV1;
32      RCC->CFGR |= RCC_CFGR_PPRE2_DIV1;
33
34      RCC->CR &= ~RCC_CR_PLLON;
35
36      //  PLL configuration:  = HSE * 9 = 72 MHz
37      RCC->CFGR &= ~(RCC_CFGR_PLLSRC | RCC_CFGR_PLLMULL);
38      RCC->CFGR |=  (RCC_CFGR_PLLSRC_HSE | RCC_CFGR_PLLMULL9);
39
40      RCC->CR |= RCC_CR_PLLON;
41      while((RCC->CR & RCC_CR_PLLRDY) == 0) __NOP();
42
43      RCC->CFGR &= ~RCC_CFGR_SW;
44      RCC->CFGR |=  RCC_CFGR_SW_PLL;
45      while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL);
46  }
```

Рисунок 10.1 – Настройка системных часов

```
48 void USART_Init () {
49   RCC->APB1ENR |= RCC_APB1ENR_USART2EN;
50   RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;
51
52   // CNF2 = 10, MODE2 = 01
53   GPIOA->CRL &= (~GPIO_CRL_CNF2_0);
54   GPIOA->CRL |= (GPIO_CRL_CNF2_1 | GPIO_CRL_MODE2);
55
56   // CNF3 = 10, MODE3 = 00, ODR3 = 1
57   GPIOA->CRL &= (~GPIO_CRL_CNF3_0);
58   GPIOA->CRL |= GPIO_CRL_CNF3_1;
59   GPIOA->CRL &= (~(GPIO_CRL_MODE3));
60   GPIOA->BSRR |= GPIO_ODR_ODR3;
61
62   //USARTDIV = SystemCoreClock / (16 * BAUD) = 72000000 / (16 * 9600) = 468,75
63
64   USART2->BRR = 7500;
65
66   USART2->CR1 |= USART_CR1_TE | USART_CR1_RE | USART_CR1_UE | USART_CR1_IDLEIE ;
67   USART2->CR2 = 0;
68   USART2->CR3 = 0;
69 }
```

Рисунок 10.2 – Инициализация USART

Далее будет представлен пример выполнения задач.

Преобразование массива символов text при знании её длины length можно провести в отдельной функции, считывая текст посимвольно слева направо и умножая с каждым новым символом на десять (рисунок 10.3). Таким образом, можно считать десятичное записанное число.

```
82 int str2int(unsigned char* text, int length) {
83   int i;
84   int num = 0;
85   for (i = 0; i < length; i++) {
86     num = 10*num + (text[i] - '0');
87   }
88   return num;
89 }
```

Рисунок 10.3 – Преобразование символьного массива в число

Стоит заметить, что преобразовать значение char = ‘5’ в int = 5 следует делать через операцию вычитания с символом ‘0’ (char – ‘0’). Обратное следует делать через сложение с символом ‘0’.

Для преобразования десятичного числа в двоичное будет использоваться отдельная функция, внутри которой будет только что реализованная str2int (рисунок 4). Полученное десятичное число следует брать остаток от деления на 2 для записи в строку и делить число на 2 (строки 94 и 95). Если потребу-

ется десятичное число преобразовать по другому основанию, то двойку следует заменить на требуемое число.

```
 91 void decimal2binary(unsigned char* input, int length, unsigned char* output) {
 92   int i;
 93   int decimal = str2int(input, length);
 94   for (i = 0; i < 4; i++) {
 95     if (decimal > 0) {
 96       output[3 - i] = (decimal % 2) + '0';
 97       decimal /= 2;
 98     }
 99     else {
100       output[3 - i] = '0';
101     }
102   }
103 }
```

Рисунок 10.4 – Преобразование десятичного числа в двоичное

Также если decimal уже дошло до нуля, то далее можно записывать лишь нули, которые ни на что не повлияют, так как находятся в старшей части числа.

При необходимости можно прописать и обратное преобразование – из двоичного в десятичное (рисунок 10.5). В первую очередь получается число, а после происходит разделение числа по цифрам при помощи остатка от деления. Чтобы провести преобразование из двоичного в другое основание, то следует изменить двойку в строке 109 на требуемое число.

Далее происходит преобразование числа в строку по каждой отдельной цифре при помощи деления на 10. Аналогично предыдущей функции у данной в старших разрядах можно выставить нули, но можно заменить и на пустой символ или пробел.

```
105 void binary2decimal(unsigned char* input, int length, unsigned char* output) {
106   int i;
107   int decimal = 0;
108   int change = 1;
109   int binary = str2int(input, length);
110   while (binary > 0) {
111     decimal += (binary % 10) * change;
112     change *= 2;
113     binary /= 10;
114   }
115   for (i = 0; i < 4; i++) {
116     if (decimal > 0) {
117       output[3 - i] = (decimal % 10) + '0';
118       decimal /= 10;
119     }
120     else {
121       output[3 - i] = ' ';
122     }
123   }
124 }
```

Рисунок 10.5 – Преобразование двоичного числа в десятичное

Далее остаётся лишь использовать два двоичных числа для логической операции – в данном случае логического И (рисунок 10.6). Функция настроена так, что в неё входят два символьных массива, которые внутри имеют двоичное четырёхбитное число. Каждый символ преобразовывается в цифру, а после применяется операция &. Так как результат тоже текстовый, следует выполнить преобразование в текст.

```
126 void operationAND(unsigned char* op_1, unsigned char* op_2, unsigned char* result) {
127   int i;
128   for (i = 0; i < 4; i++) {
129     result[i] = ( (op_1[i]-'0') & (op_2[i]-'0') ) + '0';
130   }
131 }
```

Рисунок 10.6 – Логическая операция И над двумя четырёхбитными операндами

Пример использования функций будет проходить внутри тела программы, где в начале следует объявить все необходимые элементы (рисунок 10.7). Далее следует выполнить все настройки – от системных часов до USART’a.

```
int main () {
  int i = 0;
  unsigned char inp_1[4] = {' ', ' ', ' ', ' '};
  unsigned char inp_2[4] = {' ', ' ', ' ', ' '};
  int inp_cnt_1 = 0;
  int inp_cnt_2 = 0;

  unsigned char out_1[4] = {' ', ' ', ' ', ' '};
  unsigned char out_2[4] = {' ', ' ', ' ', ' '};
  unsigned char out_3[4] = {' ', ' ', ' ', ' '};


  SystemCoreClockConfigure();
  SystemCoreClockUpdate();
  SysTick_Config(SystemCoreClock / 1000);   // SysTick 1 msec interrupts

  USART_Init();

  USART_Send('\n');     // Reset message
  USART_Send('\n');
  USART_Send('\r');
```

Рисунок 10.7 – Объявление переменных и настройка

Внутри бесконечного цикла в первую очередь будет идти два цикла приёма символов, которые будут заканчиваться по символу пробела (рисунок 10.8). После успешного приёма оба массива будут переданы на преобразование в бинарный вид, а их результат на логическую операцию.

```
  while (1) {

      USART_Receive(&symbol);
      while (symbol != ' ') {   // input first number before the space
        USART_Send(symbol);
        inp_1[inp_cnt_1++] = symbol;
        USART_Receive(&symbol);
      }
      USART_Send(symbol);

      USART_Receive(&symbol);
      while (symbol != ' ') {   // input second number before the space
        USART_Send(symbol);
        inp_2[inp_cnt_2++] = symbol;
        USART_Receive(&symbol);
      }

      decimal2binary(inp_1, inp_cnt_1, out_1);
      decimal2binary(inp_2, inp_cnt_2, out_2);
      operationAND(out_1, out_2, out_3);
```

Рисунок 10.8 – Приём символов и применение функций

Все три результата пойдут на вывод (рисунок 10.9), где через пробел будут выведены преобразованные два числа, а с новой строки результат операции.

```
178         USART_Send('\n');          // output
179         USART_Send('\r');
180         for (i = 0; i < 4; i++) {
181           USART_Send(out_1[i]);
182         }
183         USART_Send(' ');
184         for (i = 0; i < 4; i++) {
185           USART_Send(out_2[i]);
186         }
187         USART_Send('\n');
188         USART_Send('\r');
189         for (i = 0; i < 4; i++) {
190           USART_Send(out_3[i]);
191         }
192         USART_Send('\n');
193         USART_Send('\r');
194
195
196         inp_cnt_1 = 0;         // resetting variables
197         inp_cnt_2 = 0;
198         for (i = 0; i < 4; i++) {
199           out_1[i] = ' ';
200           out_2[i] = ' ';
201           out_3[i] = ' ';
202         }
203     }
```

Рисунок 10.9 – Отправка результата и сброс записанного

Стоит заметить, что в самом конце бесконечного цикла следует обновить значения только что использованных переменных.

Пример функционирования программы приведён на рисунке 10.10.

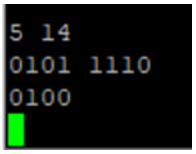


Рисунок 10.10 – Пример функционирования программы

Полный листинг программы примера приведен ниже:

```
#include "stm32f10x.h"

#include <stdio.h>


unsigned char symbol = '\0';
```

```
unsigned  char input[100];
int  input_cnt  = 0;

volatile  uint32_t  msTicks;                          // counts  1ms timeTicks
/*----------------------------------------------------------------------------
 * SysTick_Handler:
 *----------------------------------------------------------------------------*/
void SysTick_Handler(void)  {
  msTicks++;
}

void Delay  (uint32_t  dlyTicks)  {
  uint32_t  curTicks;

  curTicks  = msTicks;
  while  ((msTicks  - curTicks)  < dlyTicks)  { __NOP(); }
}

void SystemCoreClockConfigure(void)  {
                RCC->CR |= ((uint32_t)RCC_CR_HSEON);
                while  ((RCC->CR & RCC_CR_HSERDY)  == 0);

                RCC->CFGR = RCC_CFGR_SW_HSE;
                while  ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_HSE);

          RCC->CFGR = RCC_CFGR_HPRE_DIV1;                 // HCLK = SYSCLK
                RCC->CFGR |= RCC_CFGR_PPRE1_DIV1;           // APB1 = HCLK/2
                RCC->CFGR |= RCC_CFGR_PPRE2_DIV1;           // APB2 = HCLK/2
```

```
        RCC->CR &= ~RCC_CR_PLLON;

        //  PLL configuration:   = HSE * 9 = 72 MHz
        RCC->CFGR &= ~(RCC_CFGR_PLLSRC | RCC_CFGR_PLLMULL);
        RCC->CFGR |=  (RCC_CFGR_PLLSRC_HSE  | RCC_CFGR_PLLMULL9);

        RCC->CR |= RCC_CR_PLLON;
        while((RCC->CR  & RCC_CR_PLLRDY)  == 0) __NOP();

        RCC->CFGR &= ~RCC_CFGR_SW;
        RCC->CFGR |=  RCC_CFGR_SW_PLL;
        while  ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL);
}

void USART_Init  () {
    RCC->APB1ENR |= RCC_APB1ENR_USART2EN;
    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;

    // CNF2 = 10, MODE2 = 01
    GPIOA->CRL &= (~GPIO_CRL_CNF2_0);
    GPIOA->CRL |= (GPIO_CRL_CNF2_1 | GPIO_CRL_MODE2);

    // CNF3 = 10, MODE3 = 00, ODR3 = 1
    GPIOA->CRL &= (~GPIO_CRL_CNF3_0);
    GPIOA->CRL |= GPIO_CRL_CNF3_1;
    GPIOA->CRL &= (~(GPIO_CRL_MODE3));
    GPIOA->BSRR |= GPIO_ODR_ODR3;
```

```
	//USARTDIV  = SystemCoreClock  / (16 * BAUD)  = 72000000 / (16 * 9600) = 468,75


	USART2->BRR  = 7500;


	USART2->CR1 |= USART_CR1_TE  | USART_CR1_RE  | USART_CR1_UE  |
USART_CR1_IDLEIE  ;
	USART2->CR2  = 0;
	USART2->CR3  = 0;
}


void USART_Send  (unsigned  char symbol)  {
		while  ((USART2->SR  & USART_SR_TXE)  == 0) {}
		USART2->DR  = symbol;
}


void USART_Receive  (unsigned  char *data) {
		 while  ((USART2->SR  & USART_SR_RXNE)  == 0) {}
		*data = (unsigned  char)USART2->DR;
}



int str2int(unsigned  char* text,  int length)  {
	int i;
	int num  = 0;
	for (i = 0; i < length;  i++) {
		num  = 10*num  + (text[i]  - '0');
	}
```

```
        return num;
}

void decimal2binary(unsigned char* input, int length, unsigned char* output) {
        int i;
        int decimal = str2int(input, length);
        for (i = 0; i < 4; i++) {
                if (decimal > 0) {
                        output[3 - i] = (decimal % 2) + '0';
                        decimal /= 2;
                }
                else {
                        output[3 - i] = '0';
                }
        }
}

void binary2decimal(unsigned char* input, int length, unsigned char* output) {
        int i;
        int decimal = 0;
        int change = 1;
        int binary = str2int(input, length);
        while (binary > 0) {
                decimal += (binary % 10) * change;
                change *= 2;
                binary /= 10;
        }
        for (i = 0; i < 4; i++) {
```

```
            if (decimal > 0) {
                output[3 - i] = (decimal % 10) + '0';
                decimal /= 10;
            }
            else {
                output[3 - i] = ' ';
            }
        }
}

void operationAND(unsigned char* op_1, unsigned char* op_2, unsigned char* result) {
        int i;
        for (i = 0; i < 4; i++) {
            result[i] = ( (op_1[i]-'0') & (op_2[i]-'0') ) + '0';
        }
}


int main () {
        int i = 0;
        unsigned char inp_1[4] = {' ', ' ', ' ', ' '};
        unsigned char inp_2[4] = {' ', ' ', ' ', ' '};
        int inp_cnt_1 = 0;
        int inp_cnt_2 = 0;

        unsigned char out_1[4] = {' ', ' ', ' ', ' '};
        unsigned char out_2[4] = {' ', ' ', ' ', ' '};
        unsigned char out_3[4] = {' ', ' ', ' ', ' '};
```

```
SystemCoreClockConfigure();
SystemCoreClockUpdate();
SysTick_Config(SystemCoreClock / 1000);          // SysTick 1 msec interrupts

USART_Init();

USART_Send('\n');                    // Reset message
USART_Send('\n');
USART_Send('\r');

while (1) {

        USART_Receive(&symbol);
        while (symbol != ' ') {          // input first number before the space
            USART_Send(symbol);
            inp_1[inp_cnt_1++] = symbol;
            USART_Receive(&symbol);
        }
        USART_Send(symbol);

        USART_Receive(&symbol);
        while (symbol != ' ') {          // input second number before the space
            USART_Send(symbol);
            inp_2[inp_cnt_2++] = symbol;
            USART_Receive(&symbol);
        }
```

```
decimal2binary(inp_1, inp_cnt_1, out_1);
decimal2binary(inp_2, inp_cnt_2, out_2);
operationAND(out_1, out_2, out_3);

USART_Send('\n');                                   // output
USART_Send('\r');
for (i = 0; i < 4; i++) {
        USART_Send(out_1[i]);
}
USART_Send(' ');
for (i = 0; i < 4; i++) {
        USART_Send(out_2[i]);
}
USART_Send('\n');
USART_Send('\r');
for (i = 0; i < 4; i++) {
        USART_Send(out_3[i]);
}
USART_Send('\n');
USART_Send('\r');


inp_cnt_1 = 0;                          // resetting variables
inp_cnt_2 = 0;
for (i = 0; i < 4; i++) {
        out_1[i] = '';
        out_2[i] = '';
        out_3[i] = '';
```

```
                    }
            }


            return 0;
    }
```

11 Лабораторная работа №4. Работа с модулем АЦП

11.1 Цели работы

Ознакомление с работой модуля аналого-цифрового преобразователя (АЦП) микроконтроллера STM32F103RBT6 и его программной настройкой.

Задачи:

1. Изучить методические указания к работе.
2. Разработать функцию настройки тактирования микроконтроллера. Настроить частоту согласно варианту задания. Для реализации задержек разработать библиотеку с функцией Delay на основе системного таймера.
3. Разработать функцию инициализации и настройки АЦП в соответствии с вариантом.
4. Задать вход для чтения аналогового сигнала. Реализовать функцию чтения сигнала и вывода по UART в com-порт компьютера.
5. Подключить к отладочной плате потенциометр и проверить работу написанной программы.

Номер варианта индивидуального задания соответствует номеру ФИО студента в журнале учебной группы.

Для выполнения задач следует использовать разработанную в первой лабораторной библиотеку для работы с интерфейсом USART STM32.

Варианты индивидуальных заданий представлены в таблице 11.1.

Таблица 1 – Варианты заданий

№	Частота SYSCLK, МГц	АЦП	Частота АЦП, МГц	Вход для чтения
1	8	ADC1	4	PA0 (канал 0)
2	8	ADC2	2	PA1 (канал 1)
3	12	ADC1	6	PA4 (канал 4)
4	16	ADC2	4	PA6 (канал 6)
5	20	ADC1	10	PA7 (канал 7)
6	24	ADC2	3	PB0 (канал 8)
7	28	ADC1	7	PC0 (канал 10)
8	32	ADC2	8	PC1 (канал 11)
9	36	ADC1	6	PA0 (канал 0)
10	40	ADC2	10	PA1 (канал 1)
11	44	ADC1	11	PA4 (канал 4)

Окончание таблицы 1

12	48	ADC2	8	PA6 (канал 6)
13	8	ADC1	4	PA0 (канал 0)
14	8	ADC2	2	PA1 (канал 1)
15	12	ADC1	6	PA4 (канал 4)
16	16	ADC2	4	PA6 (канал 6)
17	20	ADC1	4	PA7 (канал 7)
18	24	ADC2	2	PB0 (канал 8)
19	28	ADC1	6	PC0 (канал 10)
20	32	ADC2	4	PC1 (канал 11)
21	36	ADC1	10	PA0 (канал 0)
22	40	ADC2	3	PA1 (канал 1)
23	44	ADC1	7	PA4 (канал 4)
24	50	ADC2	12	PA6 (канал 11)
25	52	ADC1	14	PA0 (канал 4)

11.2 Методические указания

Аналого-цифровые преобразователи (АЦП, англ. Analog-to-digital converter - ADC) являются устройствами, которые принимают входные аналоговые сигналы и генерируют соответствующие им цифровые сигналы, пригодные для обработки микропроцессорами и другими цифровыми устройствами (рис 11.1).

В STM32 используются АЦП последовательного приближения. В основе работы этого класса преобразователей лежит принцип дихотомии, т.е последовательного сравнения измеряемой величины с 1/2, 1/4, 1/8 и т.д. от возможного максимального значения ее. Это позволяет для N-разрядного АЦП выполнить весь процесс преобразования за N последовательных шагов (итераций) вместо 2N-1 при использовании последовательного счета и получить существенный выигрыш в быстродействии.

Возможности и особенности модулей ADC контроллеров stm32:

- разрешение 12 бит;
- скорость оцифровки до 1 мкс;
- до 18 физических каналов (16 внешних и 2 внутренних);
- программируемое время сэмплирования для каждого канала;

– две группы — обычная (до 16 каналов) и инжектированная (до 4-х каналов);

– предельный компаратор;

– прерывания по различным событиям;

– режим одиночных или непрерывных преобразований;

– режим сканирования заданного списка каналов;

– самокалибровка;

– автоматическое выравнивание результатов к старшему или младшему биту регистра;

– двойной режим (спаренная работа двух модулей ADC);

– использование DMA;

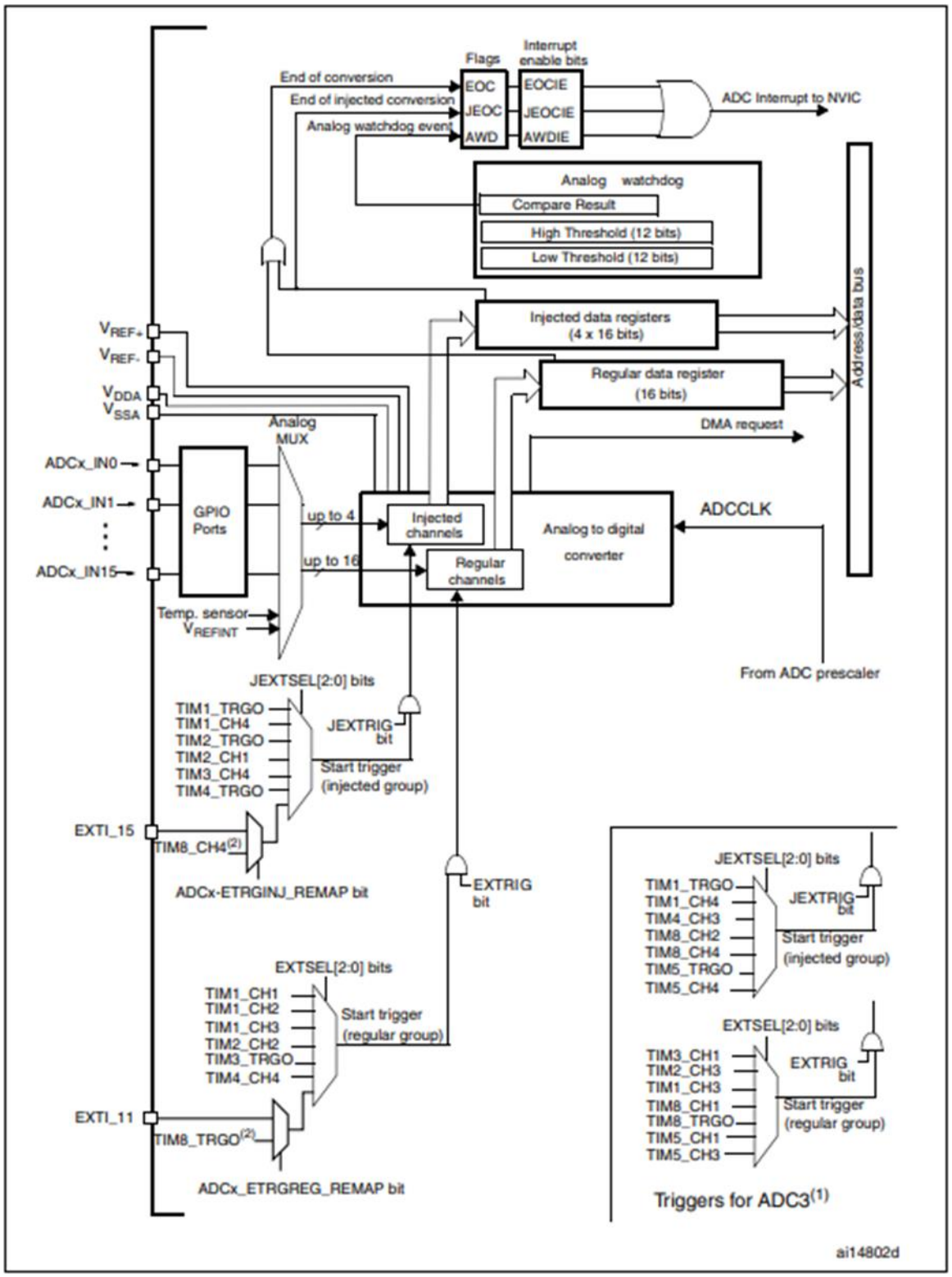


Рисунок 11.1 – Структурная схема модуля АЦП

На рисунке 11.1 продемонстрирована структурная схема модуля АЦП. Важнейшими элементами модуля АЦП являются: ADCx_IN0 – ADCx_IN15 (каналы АЦП), Analog MUX, Analog to digital converter (сам аналого-

цифровой преобразователь), Regular data register (регистр данных на выходе АЦП).

Модуль АЦП поддерживает три прерывания: End to conversion (EOCIE), End to injected conversion (JEOCIE), Analog watchdog event (AWDIE).

АЦП STM32 поддерживает 2 режима преобразований:

Режим регулярных преобразований.

В этом режиме задаётся группа от 1 до 16 каналов, обработка которых производится последовательно, а результат помещается в один-единственный 12-разрядный регистр, подразумевается, что данные будут сохраняться с помощью DMA.

Режим инжектированных преобразований.

В последовательность инжектированных преобразований может входить до 4 каналов, они имеют собственные регистры для сохранения результата.

Настройка АЦП.

В функции инициализации АЦП сначала нужно настроить тактирование. Тактирование настраивается через регистр RCC_CFGR по битам ADCPRE[1:0] (рисунок 11.2).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				MCO[3:0]				Res.	OTGFS PRE	PLLMUL[3:0]				PLL XTPRE	PLL SRC
				rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC PRE[1:0]		PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

Рисунок 11.2 – Регистр RCC_CFGR

В таблице 11.2 приведена принадлежность шин к определенным интерфейсам или устройствам. По ней видно, что ADC находится на шине APB2.

Таблица 2 – Соответствие шины и интерфейса

Интерфейс/устройство	**Шина**
GPIO	APB2
I2C	APB1
TIM	APB1/ APB2
ADC	APB2
DMA	AHB

Между синхроимпульсами шины и таковым входом модуля АЦП установлен программно-управляемый делитель ADC Prescaler (рисунок 11.3).

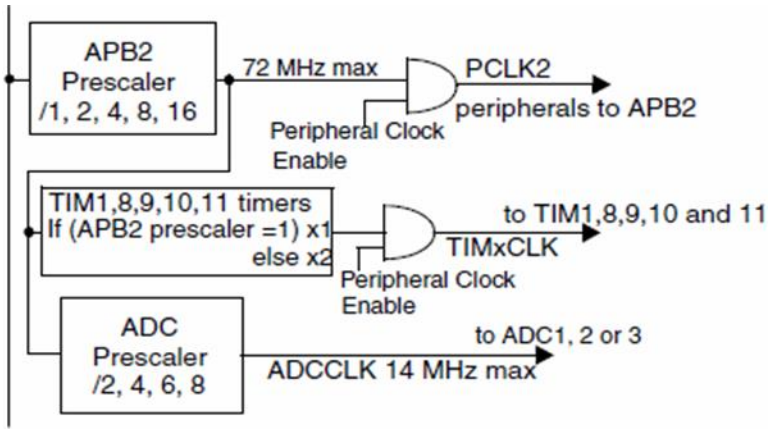


Рисунок 11.3 – ADC Prescaler

Он позволяет делить частоту тактирования шины APB2 на 2, 4, 6 и 8. Делитель общий для обоих модулей АЦП. Максимальная допустимая частота тактирования АЦП 14 мГц. Значение делителя должно быть таким, чтобы частота тактирования АЦП не превысила 14 мГц.

Биты 15:14 ADCPRE [1:0]:

– 00: / 2;
– 01: / 4;
– 10: / 6;
– 11: / 8.

Для выбора АЦП и указание нужного порта нужно использовать регистр RCC_APBENR (рисунок 11.4). Для включения тактирования АЦП важны биты ADC2 и ADC1. Для выбора порта важны биты IOP[X]EN. В данной работе предполагается использовать только порт B или A.

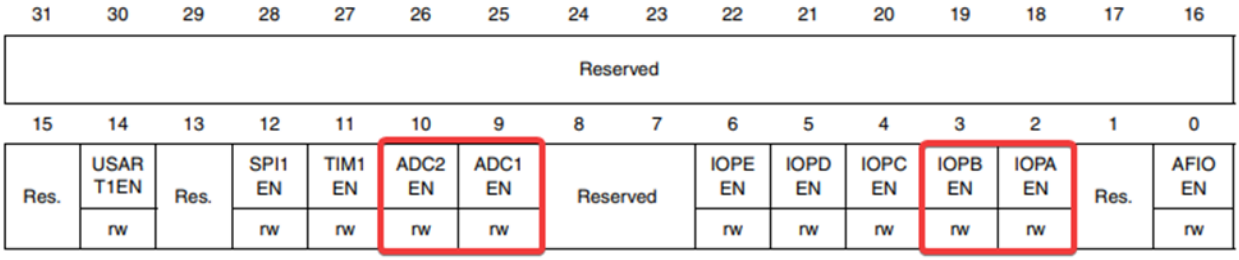


Рисунок 11.4 – Регистр RCC_APBENR

Ниже приведена часть кода, включающая тактирование АЦП и устанавливающая предделитель на 8:

RCC->CFGR |= 0b11<<14; //Выставляем предделитель на 8(11) - (64/8=8, частота не должна быть >14МГц)

RCC->APB2ENR |= (1<<2) | (1<<9);//Включаем тактирование порта A(2), и выбираем АЦП1 (9)

Следующим предполагается настроить выборку цифрового сигнала. На входе АЦП последовательного приближения всегда установлен аппаратный узел – устройство выборки и хранения аналогового сигнала. Устройство представляет собой конденсатор на входе АЦП, с аналоговым ключом в цепи входного сигнала. Ключ замыкается, и конденсатор заряжается от аналогового входа. Это процесс выборки. Затем ключ размыкается, и напряжение остается на конденсаторе неизменным, а значит и стабильным на входе АЦП. Это процесс хранения, в течение которого происходит аналогово-цифровое преобразование.

Для обеспечения точности измерения АЦП время выборки должно быть не менее определенного значения, зависящего от выходного сопротивления источника сигнала. Разработчики STM32 рассчитали эту зависимость для максимально-допустимой частоты тактирования 14 мГц.

STM32 позволяет устанавливать время выборки для каждого канала отдельно. Используются регистры ADC_SMPR1 и ADC_SMPR2. Для каждого канала выделены поля по 3 бита. На рисунке 11.5 показан регистр ADC_SMPR2. В SMP[x], x значится как номер канала.

Ниже представлены настройки выборки сигнала на первом канале (PA_0):

ADC1->SMPR2 |= ADC_SMPR2_SMP0_0 | ADC_SMPR2_SMP0_1 ; // время выборки 28,5 циклов

ADC1->SMPR2 &= ~ADC_SMPR2_SMP0_2 ;

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SMP9[2:0]			SMP8[2:0]			SMP7[2:0]			SMP6[2:0]			SMP5[2:1]	
Res.		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP 5_0	SMP4[2:0]			SMP3[2:0]			SMP2[2:0]			SMP1[2:0]			SMP0[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30 Reserved, must be kept at reset value.

Bits 29:0 **SMPx[2:0]:** Channel x Sample time selection
These bits are written by software to select the sample time individually for each channel.
During sample cycles channel selection bits must remain unchanged.
000: 1.5 cycles
001: 7.5 cycles
010: 13.5 cycles
011: 28.5 cycles
100: 41.5 cycles
101: 55.5 cycles
110: 71.5 cycles
111: 239.5 cycles

Рисунок 11.5 – Регистр ADC_SMPR2

Дальнейшая настройка также важна. Выбор каналов измерения происходит аналогично для всех режимов. Для регулярных каналов есть 3 регистра, которые определяют последовательность преобразований в одном цикле ADC_SQR1, ADC_SQR2 и ADC_SQR3 (рисунки 11.6-11.8). Поле L [3:0] регистра ADC_SQR1 определяет количество регулярных каналов, которые надо опросить. В одном цикле может быть опрошено до 16ти регулярных каналов, и они могут повторяться.

Для каждого номера в последовательности опроса отведены по 5 бит SQ, которые определяют номер канала. В поле SQ1 надо установить номер канала, который будет опрашиваться первым, в поле SQ2 – вторым и т.д.

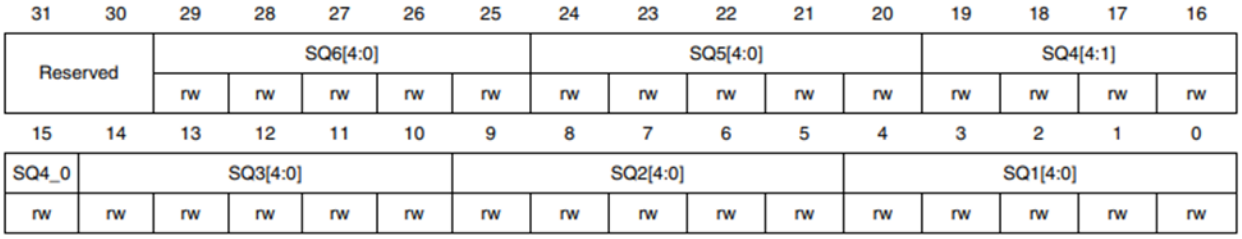


Рисунок 11.6 – Регистр ADC_SQR3

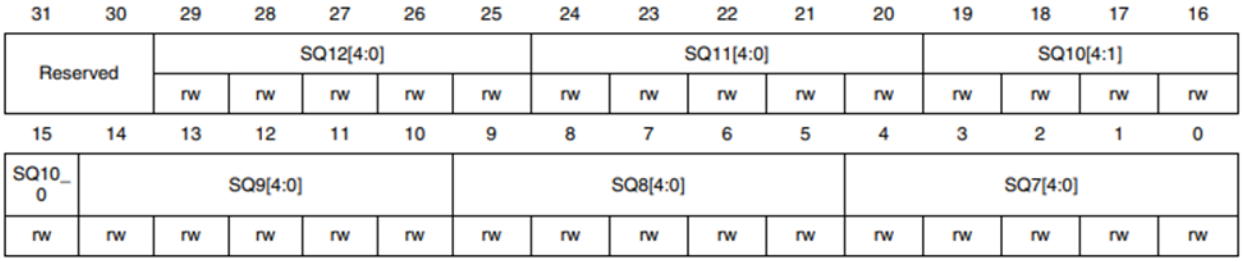


Рисунок 11.7 – Регистр ADC_SQR2

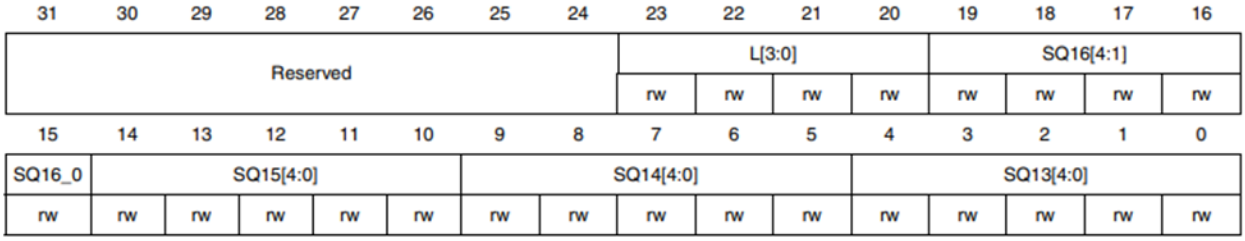


Рисунок 11.8 – Регистр ADC_SQR1

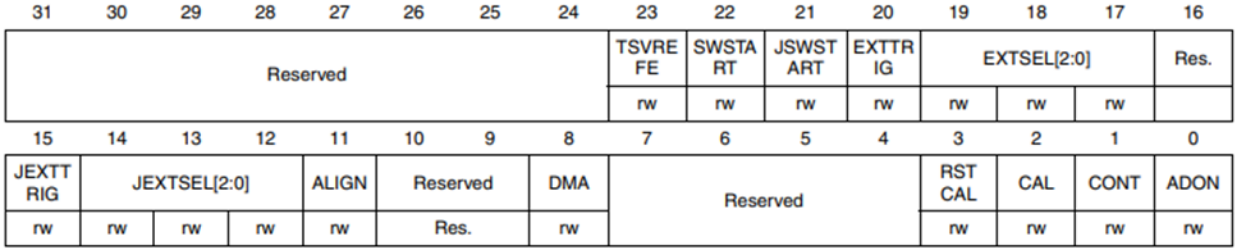
Ниже приведен код настройки последовательности для одного преобразования и первого канала:

```
// выбор каналов
ADC1->SQR1 =0; // 1 регулярный канал
ADC1->SQR3 =(1<<0); // 1 преобразование - канал 1
```

Дальше настроить значения в регистре ADC_CR2 (рисунок 11.9) программный запуск АЦП:

```
ADC1->CR2 |= 0b111<<17;  //Выбор программного запуска АЦП
ADC1->CR2 |= 1<<20; //Разрешение внешнего запуска преобразования для регулярных каналов.
```

ADC1->CR2 |= 1; //Включить АЦП



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								TSVREFE	SWSTART	JSWSTART	EXTTRIG	EXTSEL[2:0]			Res.
								rw	rw	rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JEXTTRIG	JEXTSEL[2:0]			ALIGN	Reserved		DMA	Reserved				RSTCAL	CAL	CONT	ADON
rw	rw	rw	rw	rw	Res.		rw					rw	rw	rw	rw

Рисунок 11.9 – Регистр ADC_CR

Более подробная информация о регистрах ADC находится здесь: [ADC. Описание регистров (rotr.info)](rotr.info).

Есть три метода чтения данных из АЦП:

1 – Метод опроса.

Это самый простой способ в коде для выполнения аналого-цифрового преобразования с использованием АЦП на аналоговом входном канале. Однако это не во всех случаях эффективный способ, поскольку считается, что это блокирующий способ использования АЦП. Таким образом, мы запускаем аналого-цифровое преобразование и ждем, пока АЦП завершит преобразование, чтобы центральный процессор мог возобновить обработку основного кода.

2 – Метод прерывания

Метод прерывания является эффективным способом выполнения преобразования АЦП неблокирующим образом, поэтому центральный процессор может возобновить выполнение основной процедуры кода до тех пор, пока АЦП не завершит преобразование и не подаст сигнал прерывания, чтобы центральный процессор мог переключиться в контекст ISR и сохранить результаты преобразования для дальнейшей обработки.

Однако, когда идет дело с несколькими каналами в циклическом режиме, будут периодические прерывания от АЦП, которые слишком велики для обработки процессором. Это приведет к возникновению дрожания, задержке прерывания и всевозможным проблемам с синхронизацией в системе.

3 – Метод DMA

И, наконец, метод DMA является наиболее эффективным способом преобразования нескольких каналов АЦП с очень высокой скоростью и по-прежнему передает результаты в память без вмешательства центрального процессора, что является очень крутым и экономящим время методом.

В этой работе используется простейший метод опроса. Ниже представлен код опроса АЦП:

```
while(1)
    {
    ADC1->CR2 |= 1<<22; // Запускаем АЦП
    while(!(ADC1->SR & 0b10)){} // Ожидание АЦП
    res = ADC1->DR; // Забираем результат
    V = ((float)res / 4096) * 3.3; //Преобразуем отсчеты АЦП в Вольты
    //Тут должна быть ваша отправка значений res и V по UART
    Delay_ms(1000);
  }
```

На рисунке 11.10 показан результат вывода значений АЦП.

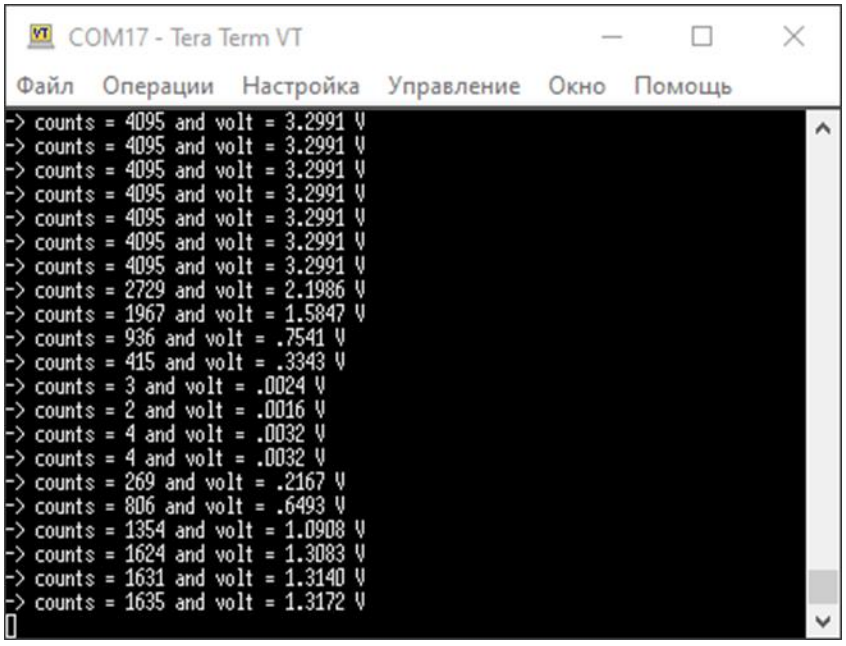


Рисунок 11.10 – Получение значений АЦП по UART

На рисунке 11.11 продемонстрировано расположение контактов Nucleo-F103RB.

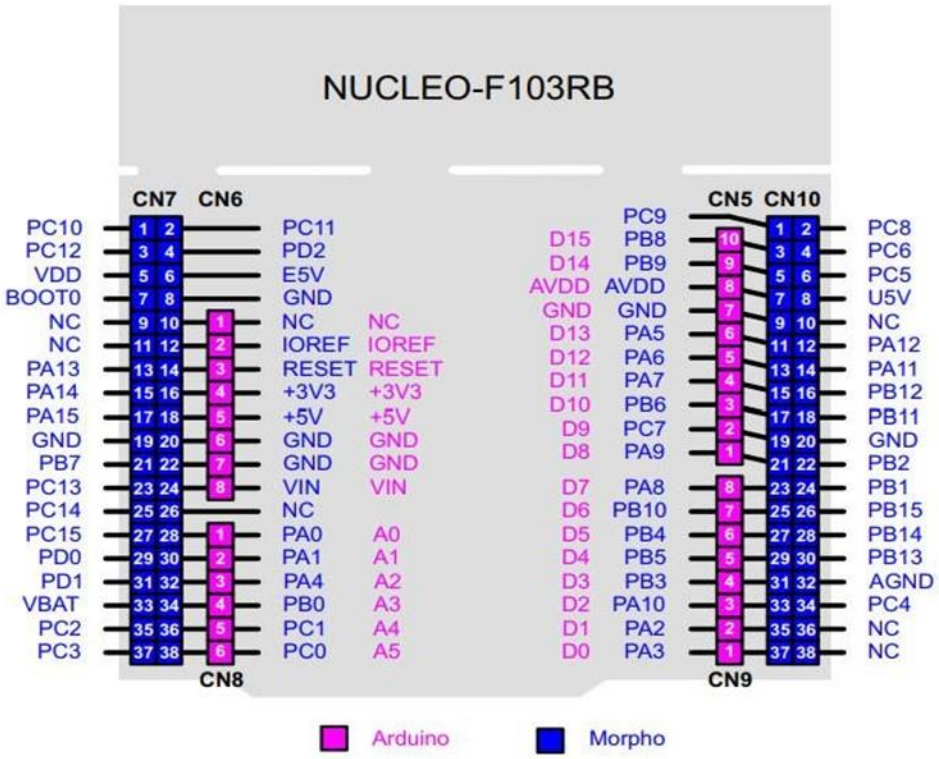


Рисунок 11.11 – Расположение контактов отладочной платы

Пожалуй, самым простым аналоговым датчиком является переменный резистор (потенциометр или реостат) (рисунок 11.12).

Рисунок 11.12 – Реостат

Переменный резистор фактически представляет собой делитель напряжения (рисунок 11.13).

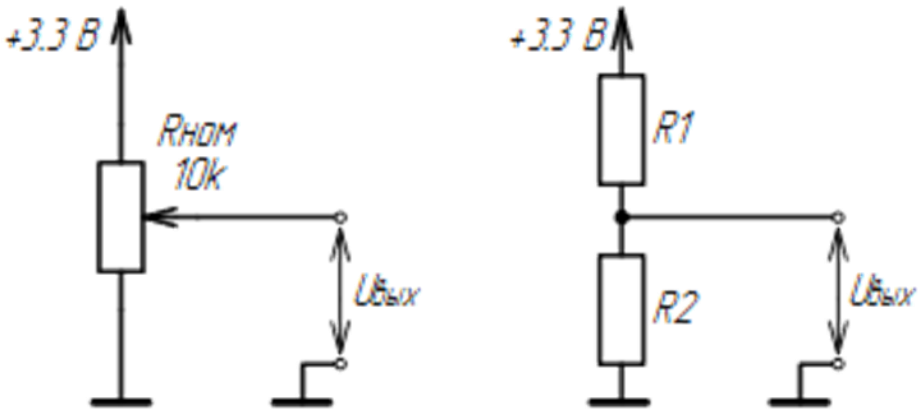


Рисунок 11.13 – Схема реостата и делителя напряжения.

В данной работе предлагается в качестве источника аналогового сигнала использовать реостат, подключенный к плате расширения Accessory Shield Xbee (рисунок 11.14).

Рисунок 11.14 – Accessory Shield Xbee

Accessory Shield Xbee оснащен встроенным регулируемым потенциометром 10k с максимальным регулируемым выходным напряжением 3,3 В.

Все контакты данной платы соответствуют с расположением контактов по Arduino Morpho (рисунок 11.15).

Для подключения потенциометра нужно плату запитать от 3,3В и подключится к контакту A0 (всего три провода vcc, gnd, a0).

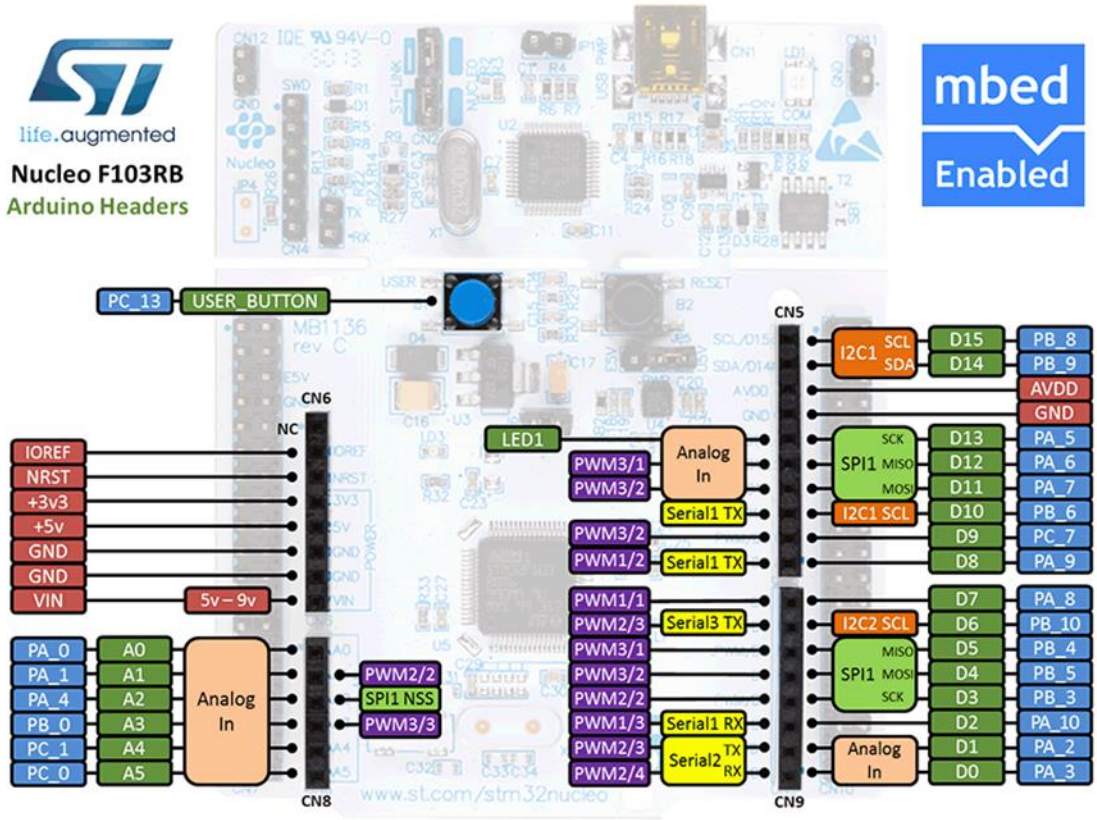


Рисунок 11.15 – Расположение контактов Accessory Shield Xbee

На рисунке 11.16 показан пример подключения к плате расширения.

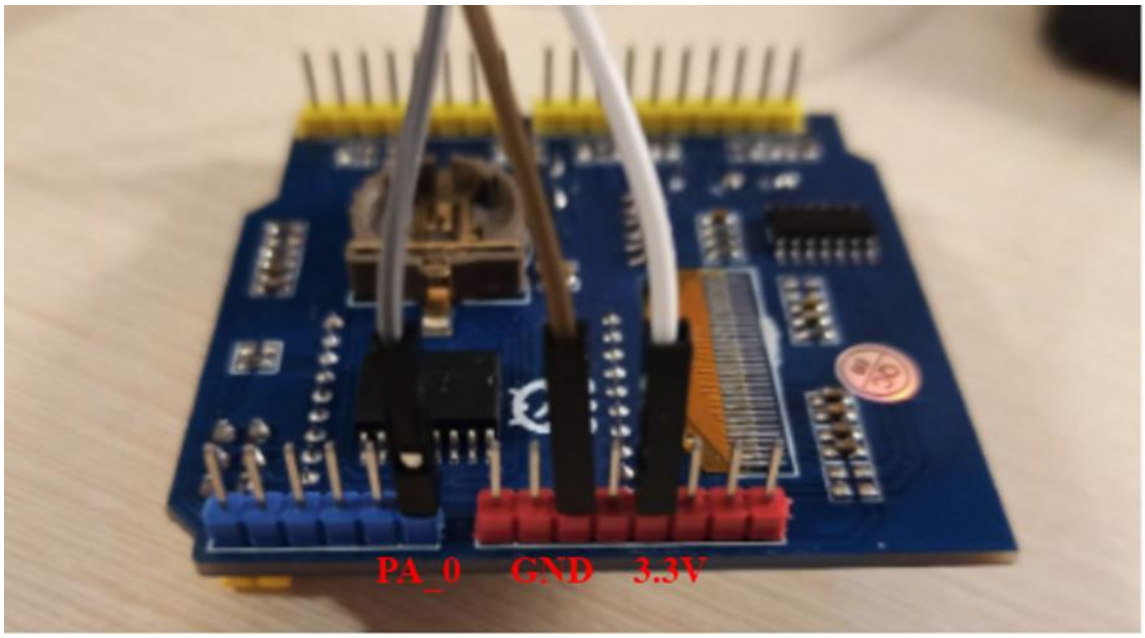


Рисунок 11.16 – Подключение Accessory Shield Xbee

Листинг функции инициализации АЦП представлен ниже:

```
void ADC_init(void){
    RCC->CFGR |= 3<<14; //Выставляем предделитель на 8(0b11=3) - (64/8=8, частота не должна быть >14МГц)
    RCC->APB2ENR |= (1<<2) | (1<<9);//Включаем тактирование порта A(2), АЦП1 (9)
    GPIOA->CRL &= ~ (GPIO_CRL_MODE1 | GPIO_CRL_CNF1); // PA1-аналоговый вход
    ADC1->SMPR2 |= ADC_SMPR2_SMP0_0 | ADC_SMPR2_SMP0_1 ; // время выборки 28,5 циклов
    ADC1->SMPR2 &= ~ADC_SMPR2_SMP0_2;
    ADC1->SQR1 = 1; // 1 регулярный канал
    ADC1->SQR3 |= (1<<0); // 1е преобразование - канал 1 (PA1) (тут нужно установит канал)
    ADC1->CR2 |= 7<<17;  //Выбор программного запуска АЦП (0b111 = 7)
    ADC1->CR2 |= 1<<20;          //Разрешение внешнего запуска преобразования для регулярных каналов.
    ADC1->CR2 |= 1;              //Включить АЦП
    //ADC1->CR2 |= 1<<2;      //Запуск калибровки
    //while  (!(ADC1->CR2 & (1<<2))){} // Ожидание окончания калибровки
}
```

Пример отправки данных с АЦП по UART представлен ниже:

```
int main () {

    SystemCoreClockConfigure();
    SystemCoreClockUpdate();
    SysTick_Config(SystemCoreClock / 1000);          // SysTick 1 msec interrupts
    ADC_init();
    USART_Init();
```

```
 while(1)
 {
 ADC1->CR2 |= 1<<22;
                while(!(ADC1->SR  & 2)){}
                res = ADC1->DR;
                V = ((float)res  / 4096) * 3.324;

                for(i=1000;  i>=1; i/=10)
                {
                        while  (!(USART1->SR & 128)) {}
                        USART1->DR = (char)((char)(res/i)%10)+48;
                }

                while  (!(USART1->SR & 128)) {}
                USART1->DR = '\r';

                for(j=0;j<1000000;j++){}

  }
}
```

12 Лабораторная работа №5. Работа с устройством «зуммер»

12.1 Цели работы

Ознакомление с работой устройства «зуммер» на плате расширения Accessory Shield Xbee и его программной настройкой.

Задачи:

1. Изучить методические указания к работе.

2. Выполнить ход работы, проведя настройку частоты SYSCLK и частоты АЦП в соответствии с номером варианта.

3. В качестве индивидуального задания разработать функцию, выполняющую вывод последовательности звуковых сигналов. В качестве последовательности используется фамилия выполняющего(-их) лабораторную работу студента(-ов), переведенная в код Морзе.

4. Подготовить отчет со скриншотами, подтверждающими выполнения хода работы и индивидуального задания (табл. 12.1).

Таблица 12.1 - Варианты заданий

№	Частота SYSCLK, МГц	Частота ADC, МГц
демонстрационный	8	4
1	12	3
2	16	4
3	20	5
4	24	3
5	28	7
6	32	4
7	36	6
8	40	10
9	44	11
10	48	6

12.2 Методические указания

Существует много способов связи между электронным устройством и человеком, но наиболее распространенным и популярным вариантом аудиосвязи является звуковой сигнал. Поэтому «зуммеры», способные преобразовывать электрические сигналы в акустические, играют важную роль в современных электрических и электронных устройствах. Мы часто встречаем “зуммеры” в таких устройствах как будильники, дверные звонки или другая бытовая техника. Однако многие не знают о различных типах «зуммеров» и о том, как они работают.

На рисунке 12.1 представлен внешний вид «зуммера» и его распиновка.

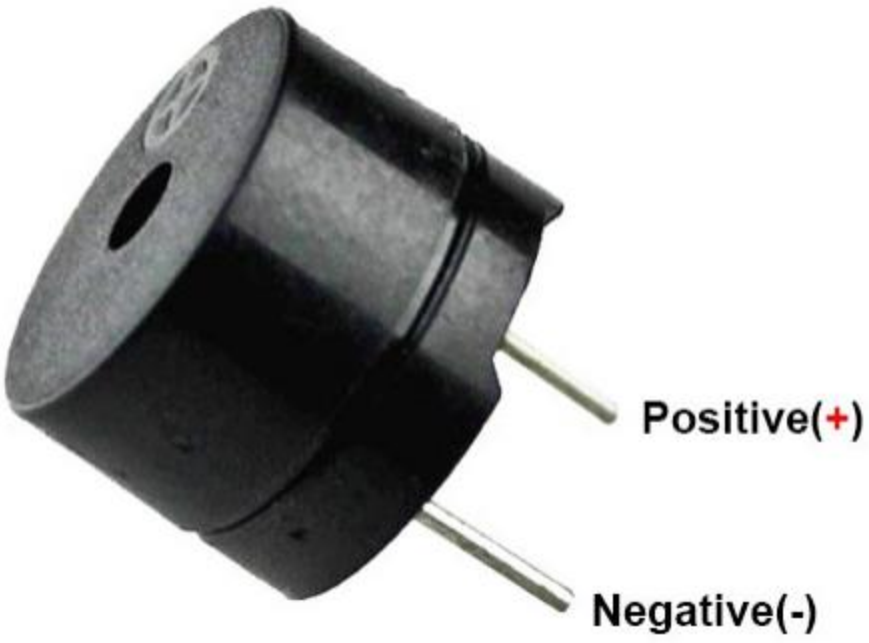


Рисунок 12.1 – Внешний вид «зуммера» и его распиновка

Основная идея создания звука заключается в создании вибрации, поскольку, как видно на голосовых связках человека, наличие звука определяется наличием вибраций. Внутри «зуммера» находится катушка провода, подключенная к контактам «зуммера». Кроме того, катушку с проводом окружает круглый магнит. Тонкий гибкий ферромагнитный металлический диск с прикрепленным к верху небольшим металлическим грузиком расположен над круглым магнитом и проволочной катушкой. Когда на проволоч-

ную катушку подаются импульсы тока, магнитная индуктивность заставляет металлический груз и металлический диск вибрировать вверх и вниз. Вибрация металлического диска порождает звуковые волны.

На рисунке 12.2 представлен принцип работы «зуммера».

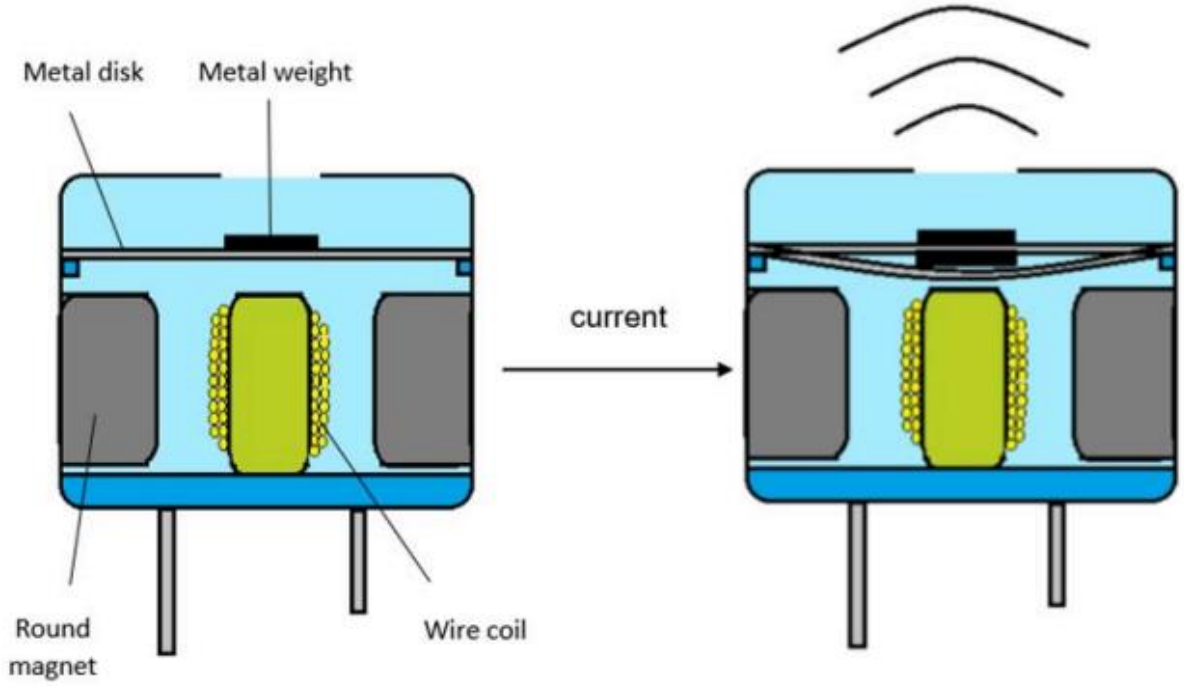


Рисунок 12.2 – Принцип работы «зуммера»

Для выполнения хода работы, необходимо обладать навыками, полученными в четвертой лабораторной работе при работе с АЦП и его настройке. Работа с “зуммером” в рамках лабораторной работы будет выполняться с помощью платы расширения Accessory Shield Xbee (рисунок 12.3).

Рисунок 12.3 – Accessory Shield Xbee

Для работы с АЦП необходимо наличие подключенных библиотек: через меню «Manage Run-Time Environment» (Рисунок 12.4) подключите библиотеки указанные ниже (Рисунок 12.5).

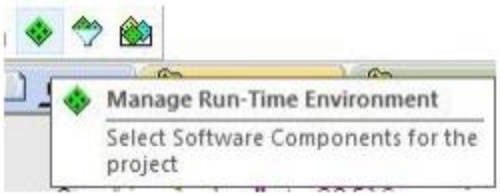


Рисунок 12.4 – Меню «Manage Run-Time Enviroment»

Device				Startup, System Setup
DMA	☐		1.3	DMA driver used by RTE Drivers for STM32F1 Series
GPIO	☐		1.3	GPIO driver used by RTE Drivers for STM32F1 Series
Startup	☑		1.0.0	System Startup for STMicroelectronics STM32F1xx device series
StdPeriph Drivers				
ADC	☑		3.6.0	Analog-to-digital converter (ADC) driver for STM32F10x
BKP	☐		3.6.0	Backup registers (BKP) driver for STM32F10x
CAN	☐		3.6.0	Controller area network (CAN) driver for STM32F1xx
CEC	☐		3.6.0	Consumer electronics control controller (CEC) driver for STM32F1xx
CRC	☐		3.6.0	CRC calculation unit (CRC) driver for STM32F1xx
DAC	☐		3.6.0	Digital-to-analog converter (DAC) driver for STM32F1xx
DBGMCU	☐		3.6.0	MCU debug component (DBGMCU) driver for STM32F1xx
DMA	☐		3.6.0	DMA controller (DMA) driver for STM32F1xx
EXTI	☐		3.6.0	External interrupt/event controller (EXTI) driver for STM32F1xx
FSMC	☐		3.6.0	Flexible Static Memory Controller (FSMC) driver for STM32F11x
Flash	☐		3.6.0	Embedded Flash memory driver for STM32F1xx
Framework	☑		3.6.0	Standard Peripherals Drivers Framework
GPIO	☑		3.6.0	General-purpose I/O (GPIO) driver for STM32F1xx
I2C	☐		3.6.0	Inter-integrated circuit (I2C) interface driver for STM32F1xx
IWDG	☐		3.6.0	Independent watchdog (IWDG) driver for STM32F1xx
PWR	☐		3.6.0	Power controller (PWR) driver for STM32F1xx
RCC	☑		3.6.0	Reset and clock control (RCC) driver for STM32F1xx
RTC	☐		3.6.0	Real-time clock (RTC) driver for STM32F1xx
SDIO	☐		3.6.0	Secure digital (SDIO) interface driver for STM32F1xx
SPI	☐		3.6.0	Serial peripheral interface (SPI) driver for STM32F1xx
TIM	☐		3.6.0	Timers (TIM) driver for STM32F1xx
USART	☐		3.6.0	Universal synchronous asynchronous receiver transmitter (USART) driver for STM32F1xx
WWDG	☐		3.6.0	Window watchdog (WWDG) driver for STM32F1xx

Рисунок 12.5 – Подключение библиотек

Настройте частоту SYSCLK и частоту ADC через функцию SystemCoreClockConfigure в соответствии с вариантом задания.

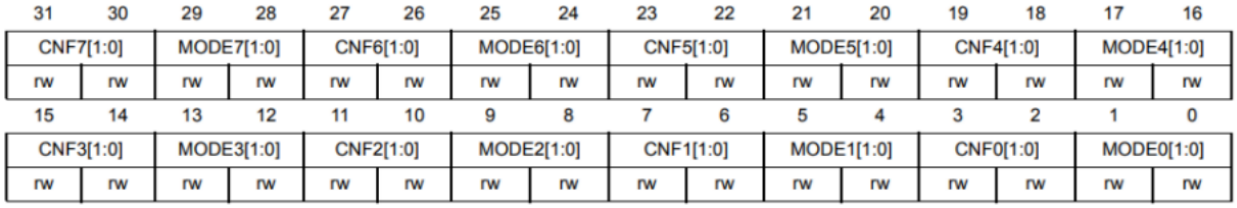
Необходимо настроить тактирование порта с помощью регистра RCC_APB2ENR (рис. 12.6). Настройка производится записью в регистр значения в соответствующую позицию.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										TIM11 EN	TIM10 EN	TIM9 EN	Reserved		
										rw	rw	rw			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3 EN	USART 1EN	TIM8 EN	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	IOPG EN	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

Рисунок 12.6 – Регистр APB2ENR

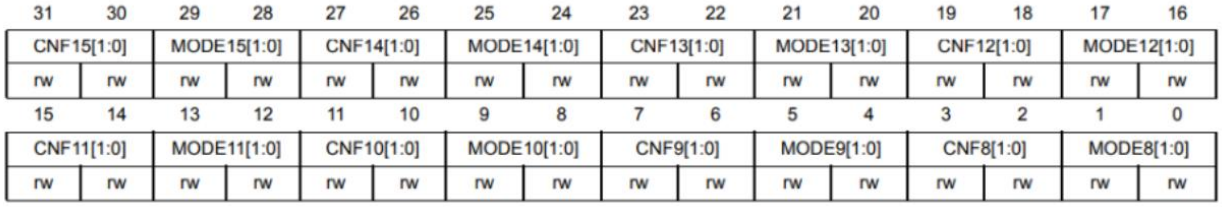
В нашем случае, чтобы разрешить тактирование, необходимо установить единицу в битовое поле 2. Чтобы настроить режим работы пинов порта, необходимо записать определенные значения в регистр GPIOA_CRL (рис.

12.7) для пинов 0-7 и GPIO_CRH для пинов 8-15. Для настройки каждого пина используется 4 бита: CNFx[1:0] и MODEx[1:0].



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 12.7 – Регистр GPIOA_CRL



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 12.8 – Регистр GPIOA_CRH

Настройте функцию, инициализирующую GPIO в соответствии с рисунком 12.9 и добавьте ее вызов в main.

```
void GPIO_init (void) {

  RCC->APB2ENR |= (1UL << 2); /* Enable GPIOA clock


  GPIOA->CRL   &= ~((15ul << 28));
  GPIOA->CRL   |=  (( 1ul << 28));
}
```

Рисунок 12.9 - Инициализация GPIO

В отчете отметьте какой режим устанавливается при такой инициализации (рис. 12.10).

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 8 .. 15)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table*.
In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved
In output mode (MODE[1:0] > 00):
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 8 .. 15)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table*.
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

Рисунок 12.10 – Режимы работы

Для подачи питания на «зуммер» добавьте функцию BEEPING. В отчете опишите принцип ее действия, а также какие режимы устанавливаются в регистрах BSRR и ODR.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Рисунок 12.11 – Регистр GPIO_BSRR

Код функции SystemCoreClockConfigure() представлен ниже:

```
void SystemCoreClockConfigure(void) {
        RCC->CR |= ((uint32_t)RCC_CR_HSION);
        while ((RCC->CR & RCC_CR_HSIRDY) == 0);
        RCC->CFGR = RCC_CFGR_SW_HSI;
        while ((RCC->CFGR & RCC_CFGR_SWS) !=
RCC_CFGR_SWS_HSI) {};
        RCC->CFGR = RCC_CFGR_HPRE_DIV1;
```

```
        // HCLK = SYSCLK
        RCC->CFGR |= RCC_CFGR_PPRE1_DIV1;
        // APB1 = HCLK/1
        RCC->CFGR |= RCC_CFGR_PPRE2_DIV1;
        // APB2 = HCLK/1
        RCC->CFGR |= RCC_CFGR_ADCPRE_DIV2;
        RCC->CR &= ~RCC_CR_PLLON;
}
```

Код функции BEEPING() представлен ниже:

```
void BEEPING () {
    GPIOA->BSRR = (GPIOA->ODR & (1ul << 7) ) ? 1ul << 23 :
1ul << 7;
}
```

Настройте функцию инициализации АЦП и добавьте ее вызов в main. Код функции ADC1_Init() представлен ниже:

```
void ADC1_Init()
{
        // Initialization  struct
        ADC_InitTypeDef  ADC_InitStruct;
        GPIO_InitTypeDef GPIO_InitStruct;
        // Step 1: Initialize  ADC1
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1,
ENABLE);
        ADC_InitStruct.ADC_ContinuousConvMode = DISABLE;
        ADC_InitStruct.ADC_DataAlign  = ADC_DataAlign_Right;
        ADC_InitStruct.ADC_ExternalTrigConv  = DISABLE;
        ADC_InitStruct.ADC_ExternalTrigConv  =
        ADC_ExternalTrigConv_None;
        ADC_InitStruct.ADC_Mode = ADC_Mode_Independent;
        ADC_InitStruct.ADC_NbrOfChannel = 1;
```

```
        ADC_InitStruct.ADC_ScanConvMode = DISABLE;
        ADC_Init(ADC1, &ADC_InitStruct);
        ADC_Cmd(ADC1, ENABLE);
        // Select input channel for ADC1
        // ADC1 channel 0 (PA0)
        ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1,
        ADC_SampleTime_55Cycles5);
        // Step 2: Initialize GPIOA (PA0)
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,
ENABLE);
        GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0;
        GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AIN;
        GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

Настройте функцию, используемую для получения значения с АЦП. В main инициализируйте переменную типа uint16_t, которая будет принимать значение с АЦП. Код функции ADC1_Read() представлен ниже:

```
uint16_t ADC1_Read()
{
        // Start ADC conversion
        ADC_SoftwareStartConvCmd(ADC1, ENABLE);
        // Wait until ADC conversion finished
        while (!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC));
        return ADC_GetConversionValue(ADC1);
}
```

Добавьте функции для реализации задержки. Код для реализации задержки представлен ниже:

```
volatile uint32_t msTicks;

void SysTick_Handler(void) {
```

```
  msTicks++;
}
void Delay (uint32_t dlyTicks) {
  uint32_t curTicks;
  curTicks = msTicks;
  while ((msTicks - curTicks) < dlyTicks) { __NOP(); }
}
```

В функцию main добавьте бесконечный цикл, где переменная будет принимать значение, полученное с АЦП, после чего настройте бесконечный цикл так, чтобы в зависимости от принятого с АЦП значения с помощью задержек будет изменяться частота подаваемого звукового сигнала. Пример оформления бесконечного цикла:

```
while(1)
{
adcValue = ADC1_Read();
BEEPING();
Delay(adcValue/40);
BEEPING();
Delay(adcValue/40);
}
```