Методическое указание по разработке программ под микроконтроллер STM32G474RE

Содержание

1. Цели и задачи практического курса. Организация кода. Основные архитектурные особенности серии контроллеров STM32. Основные модули микроконтроллера STM32G474RE.

В рамках следующего практического курса будут рассмотрены подходы проектирования программ под контроллеры STM32G474RE и основные периферийные модули, необходимые для управления двигателем постоянного тока. Также по окончании курса реализуется замкнутый контур управления ДПТ по скорости. Курс делится на 4 лабораторных занятия, в рамках которых рассматриваются следующие темы:

- инструменты разработки;
- настройка тактирования;
- настройка ножек контроллера, режимы ввода, вывода, альтернативы;
- настройка прерывания, обработка прерываний;
- работа с таймером;
- работа с АЦП;
- работа с интерфейсами связи USART, SPI;
- реализация разомкнутой и замкнутой системы управления двигателем.

Успешным завершением курса является рабочая реализация управления по замкнутому контуру. Реализация программы должна быть осуществлена на языке С стандарта 99 с использованием спец. регистров.

Микроконтроллер STM32 является одним из самых широко распространенных контроллеров на текущий день. Его распространению способствует низкая стоимость, широкие возможности, хорошая поддержка со стороны производителя, высокая надежность и простота в разработке. Основная особенность микроконтроллеров от ST заключается в архитектуре. Она позволяет разработчику:

- оперировать с 32-разрядными числами;
- иметь большие ресурсы;
- выполнять операции на встроенных модулях, снижая нагрузку ядра;
- гибко настраивать весь контроллер под необходимые параметры.

Основные элементы STM32G474RE (рисунок 1), которые будут рассмотрены в этом курсе:

- ядро ARM CORTEX-M4 — реализует обработку программы. Ядро включает в себя такие элементы, как процессор, производящий вычисления и обработку команд, NVIC (контроллер вложенных векторов прерываний), MPU (модуль защиты памяти), FPU (модуль вычисления чисел с плавающей запятой), JTAG/SW (интерфейсы загрузки/отладки программы), шины данных, инструкций и адресов (D-bus, I-bus, S-bus соответственно);
- AHB1/2 — высокоскоростная шина периферийных узлов;

- APB1/2 — шина периферийных узлов;
- RCC — контроллер перезапуска и тактирования;
- GPIO — периферийный модуль для настройки пинов контроллера;
- TIMER — периферийный модуль для настройки внутреннего счетчика, ШИМа и захвата/сравнения.
- ADC — аналого-цифровой преобразователь.
- USART, SPI — интерфейсы коммуникации.

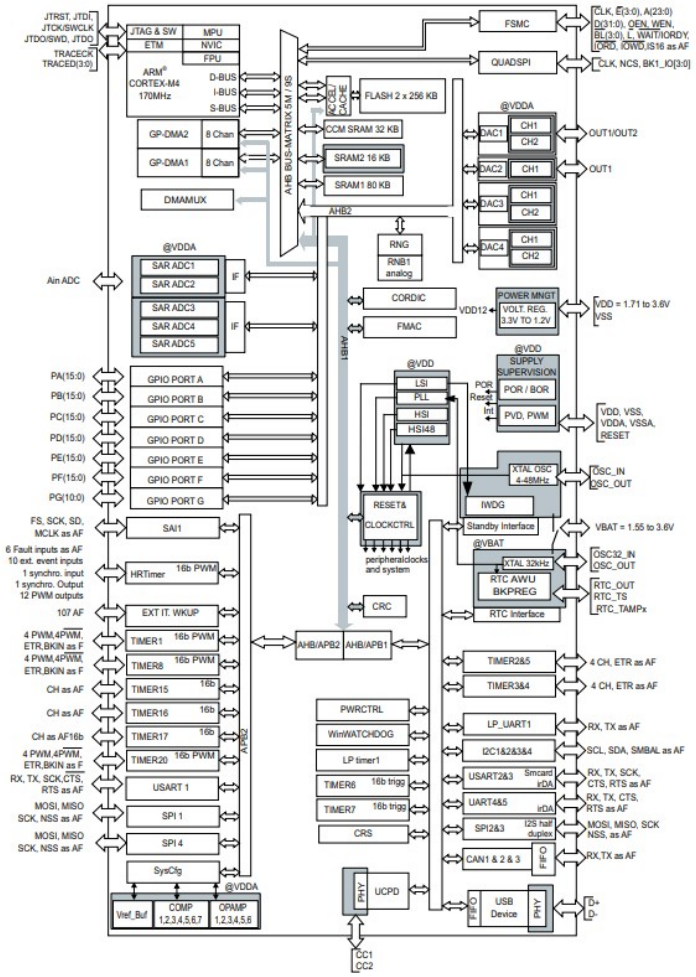


Рисунок 1. Блок-схема STM32G474RE

Организация кода будет построена следующим образом. Проект уже содержит все необходимые предустановки зависимостей. Также проект делится на модули (файлы с расширением .h, где описываются функции модуля и подключаются другие модули, и файлы .c, где реализуются эти функции). Точкой входа в программу является функция main в одноименном файле расширения .c (рисунок 2). В ней описывается основная логика работы программы, которая должна обязательно содержать бесконечный цикл, называемый основным циклом. Все функции, вызванные до этого цикла, как правило, производят инициализацию различных модулей, после чего в основном цикле реализуется основная логика работы приложения. Далее для гибкости и удобства дальнейшей работы файл main.c содержит заголовочный файл расширения .h, который описывает интерфейс основной

программы (рисунок 3). Каждый следующий модуль, разрабатываемый по ходу курса, должен иметь интерфейс в виде файла с расширением .h и реализацию интерфейса в виде файла с расширением .c и подключаться к основной программе только как библиотеки в виде заголовочных файлов в main.h. Основная библиотека, которая будет использоваться далее в каждом модуле, CMSIS , находится в файле stm32g4xx.h. Она подключается во всех заголовочных файлах.

```c
#include "main.h"

int main(void)
{
    RCC_Init();
    GPIO_Init();
    EXTI_Init();

    while(1)
    {

    }
}

void EnableLED(void)
{
    SET_BIT(GPIOA->BSRR, GPIO_BSRR_BS5);
}

void DisableLED(void)
{
    SET_BIT(GPIOA->BSRR, GPIO_BSRR_BR5);
}

void Delay(uint16_t tick_delay)
{
    for(uint32_t i = 0; i < tick_delay * 1400; ++i)
        __NOP();
}


```

Рисунок 2. Пример файла main.c

```c
#ifndef _MAIN_H_
#define _MAIN_H_
#include "stm32g4xx.h"
#include "rcc.h"
#include "gpio.h"
#include "exti.h"


void Delay(uint16_t tick_delay);
void EnableLED(void);
void DisableLED(void);

#endif /*  MAIN_H  */
```

Рисунок 3. Пример файла main.h

2. Настройка тактирования в микроконтроллере STM32G474RE. Модуль RCC.

При работе с любым цифровым устройством начальным этапом всегда служит настройка частоты его работы. В контроллерах STM32 за это отвечает модуль RCC (reset clock controller) — контроллер тактирования и перезапуска (рисунок 4). Источниками несущей частоты устройства могут быть как внутренний, так и внешний элементы. Микроконтроллер STM32 возможно тактировать с частотами как в мегагерцовых диапазонах, так и в килогерцовых. Настройка допускается достаточно гибкая, и работу периферии можно запускать на разных частотах (рисунок 4). Перечислим основные элементы в контроллере RCC:

- LSI (low speed internal) — низкочастотный внутренний источник тактирования (32 кГЦ);
- LSE (low speed external) — низкочастотный внешний источник тактирования (до 32 кГЦ);
- HSI (high speed internal) — высокочастотный внутренний источник тактирования (16 МГЦ);
- HSE (high speed external) — высокочастотный внешний источник тактирования (4-48 МГЦ);
- PLL (phase-locked loop) — фазовая автоподстройка частоты, система ФАПЧ используется умножения и преобразования частоты;
- PRESC (prescaler) — предделитель, делит несущую частоту на число, кратное 2м. Находится на каждом переходе с шины на шину тактирования и с шины тактирования на тактирования отдельного модуля;
- SYSCLK (system clock) — тактирование ядра. Определяет скорость выполнения инструкций программы.

Настройка RCC, как и любого другого периферийного модуля, производится путем записи нужных чисел в специальные регистры (области памяти, хранящие 32х-разрядные значения и отвечающие за программное взаимодействие с периферийным модулем).

Для настройки частоты необходимо осуществить следующий набор действий:

1. включить источник тактирования;
2. выбрать опорный источник;
3. подключить необходимые модули в цепь тактирования;
4. настроить предделители.

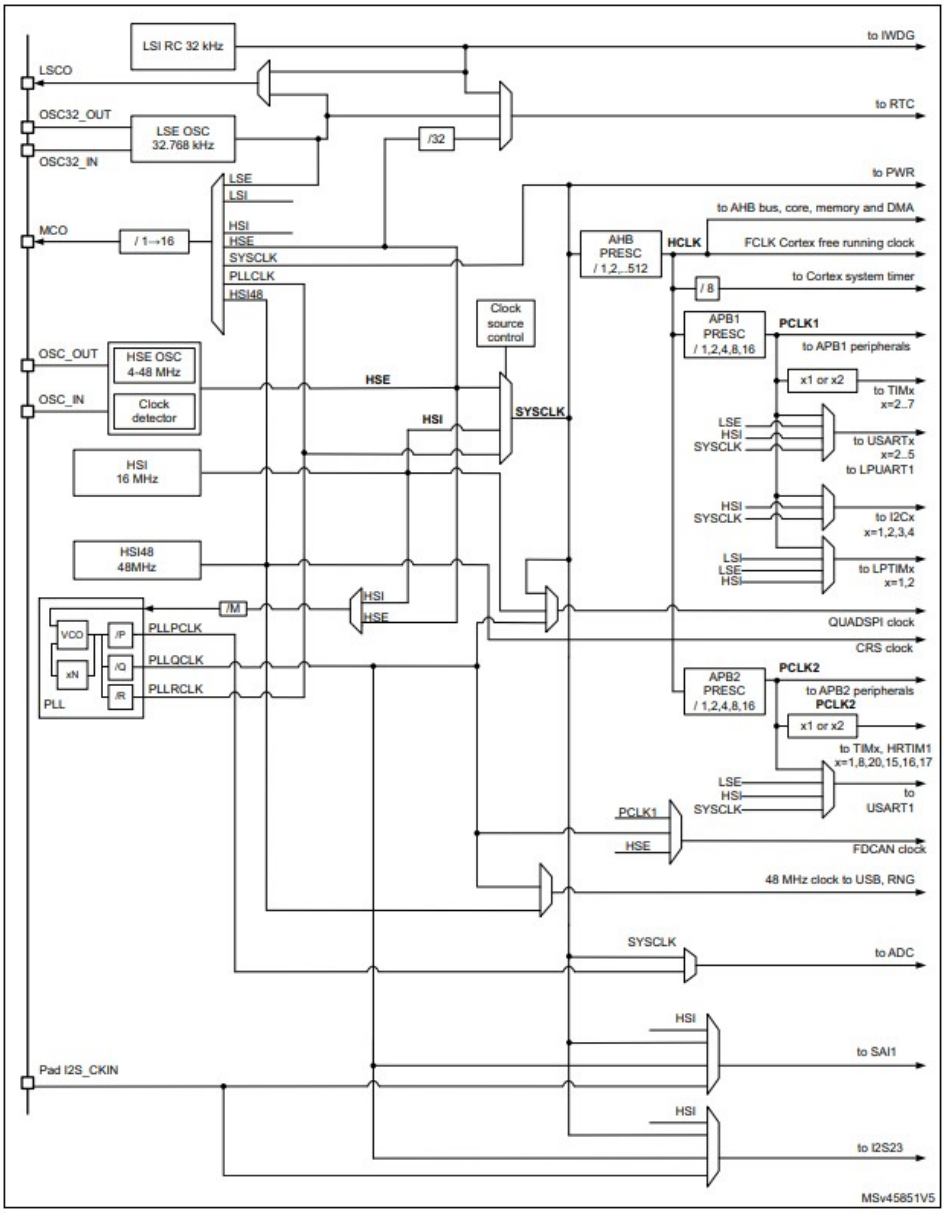


Рисунок 4. Дерево тактирования.

Задание 1. Включение тактирования GPIO порта A.

Далее будет рассмотрена задача включения тактирования периферийного модуля GPIOA с использованием внутреннего источника HSI и знакомство с принципом работы с периферийными регистрами. Эти регистры представляют собой простые 32-битные ячейки в памяти. В них может располагаться как значение, например, входного сигнала, так и значение конфигурации модуля, где каждый бит отвечает за ту или иную функцию. Значение бита может быть либо 0, либо 1. Необходимые регистры из приложения 1:

- RCC_CR — регистр управления, отвечает за включение источников тактирования (рисунок 5);
- RCC_CFGR — регистр конфигурации, отвечает за настройку тактирования, в частности, выбирает опорный источник для ядра (рисунок 6);

- RCC_ AHB2ENR — регистр тактирования шины AHB2, включает периферию на шине AHB2(рисунок 7);

7.4.1 Clock control register (RCC_CR)

Address offset: 0x00

Reset value: 0x0000 0063

HSEBYP is not affected by reset.

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	PLL RDY	PLLON	Res.	Res.	Res.	Res.	CSSON	HSEBYP	HSERDY	HSEON
						r	rw					rs	rw	r	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	HSI RDY	HSI KERON	HSION	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
					r	rw	rw								

Рисунок 5. Регистр управления модуля RCC

7.4.3 Clock configuration register (RCC_CFGR)

Address offset: 0x08

Reset value: 0x0000 0005

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during clock source switch.

From 0 to 15 wait states inserted if the access occurs when the APB or AHB prescalers values update is on going.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	MCOPRE[2:0]			MCOSEL[3:0]				Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
	rw	rw	rw	rw	rw	rw	rw								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	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	r	r	rw	rw

Bits 1:0 **SW[1:0]**: System clock switch
Set and cleared by software to select system clock source (SYSCLK).
Configured by hardware to force HSI16 oscillator selection when exiting stop and standby modes or in case of failure of the HSE oscillator.
00: Reserved, must be kept at reset value
01: HSI16 selected as system clock
10: HSE selected as system clock
11: PLL selected as system clock

Рисунок 6. Регистр конфигурации модуля RCC

7.4.15 AHB2 peripheral clock enable register (RCC_AHB2ENR)

Address offset: 0x4C

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access

Note: When the peripheral clock is not active, the peripheral registers read or write access is not supported.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	RNG EN	Res.	AES EN	Res.	Res.	Res.	Res.	DAC4 EN	DAC3 EN	DAC2 EN	DAC1 EN
					rw		rw					rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	ADC345 EN	ADC12 EN	Res.	Res.	Res.	Res.	Res.	Res.	GPIOG EN	GPIOF EN	GPIOE EN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
	rw	rw							rw	rw	rw	rw	rw	rw	rw

Рисунок 7. Регистр тактирования шины AHB2 модуля RCC

Порядок действий

1. Включить внутренний источник тактирования - выставить значение бита HSION (включения HSI) в 1.
2. Дождаться включения выбранного источника.
3. Очистить биты.
4. Сконфигурировать тактирование ядра от внутреннего источника — выставить 2 бита в поле SW соответственно значению HSI, то есть 01.
5. Дождаться нужного значения в выборе источника для системного тактирования.
6. Включить тактирование порта A GPIO — выставить бит GPIOA в 1.

Получится следующий исходный код (рисунок 8). Здесь RCC — указатель на структуру с модуля периферии RCC из документации (приложение 1), поля в котором полностью соответствуют названиям регистров. RCC_CR_HSION — предкомпиляционный макрос, соответствующий 32-разрядному числу из нулей с 1 в 8-ом бите. Остальные макросы повторяют логику. Операция «|=» — краткая форма записи логического сложения RCC→CR и RCC_CR_HSION. Функция «READ_BIT» — чтение бита, соответствующего позиции бита макроса.

```
#include "rcc.h"

void RCC_Init(void)
{
    RCC->CR |= RCC_CR_HSION;
    while(!READ_BIT(RCC->CR, RCC_CR_HSIRDY));
    RCC->CFGR |= RCC_CFGR_SW_HSI;
    while(READ_BIT(RCC->CFGR, RCC_CFGR_SW_HSI));
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;

}
```

Рисунок 8. Исходный код для включения тактирования на GPIOA.

Задание на самостоятельную работу

Включить тактирование для того же модуля с использованием внешнего тактирующего элемента HSE.

3. Базовое управление состоянием пинов микроконтроллера. Модуль GPIO.

В прошлом раздел был включен модуль GPIO (рисунок 9), который выполняет основную функцию в контроллере как базовый интерфейс взаимодействия с внешним миром. GPIO (general propose input output) — периферийный модуль, отвечающий за настройку пинов контроллера. Он может быть настроен на 3 различных режима:

- режим вывода сигнала (push-pull, open-drain, analog);
- режим входного сигнала (pull-up, pull-down, analog);
- альтернативный режим (в качестве пина интерфейса).

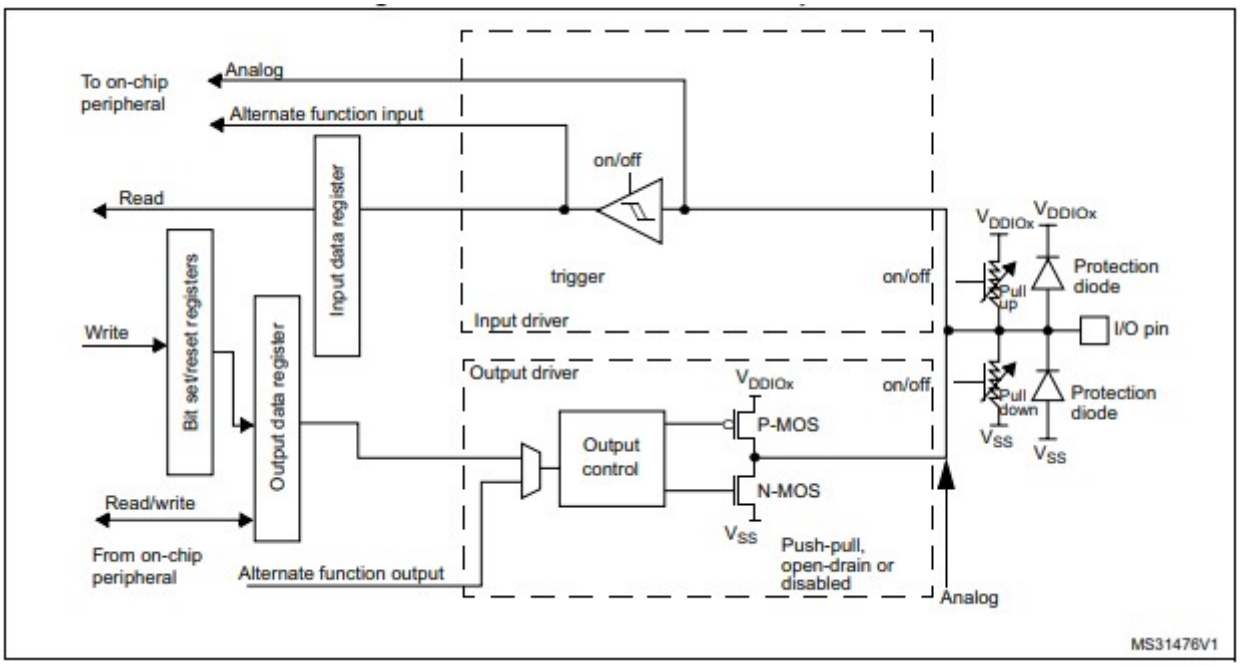


Рисунок 9. Структура GPIO

Порядок действий при настройке GPIO:

1. выбрать порт GPIO и включить на нем тактирование;
2. задать нужный режим;
3. задать скорость работы;
4. задать подтяжку;
5. задать исходное состояние;
6. *для режима альтернативной функции записать нужное значение конкретной функции;

Задание 2. Включение встроенного светодиода.

Следующая задача будет заключаться во включении встроенного светодиода на отладочной плате (рисунок 10).

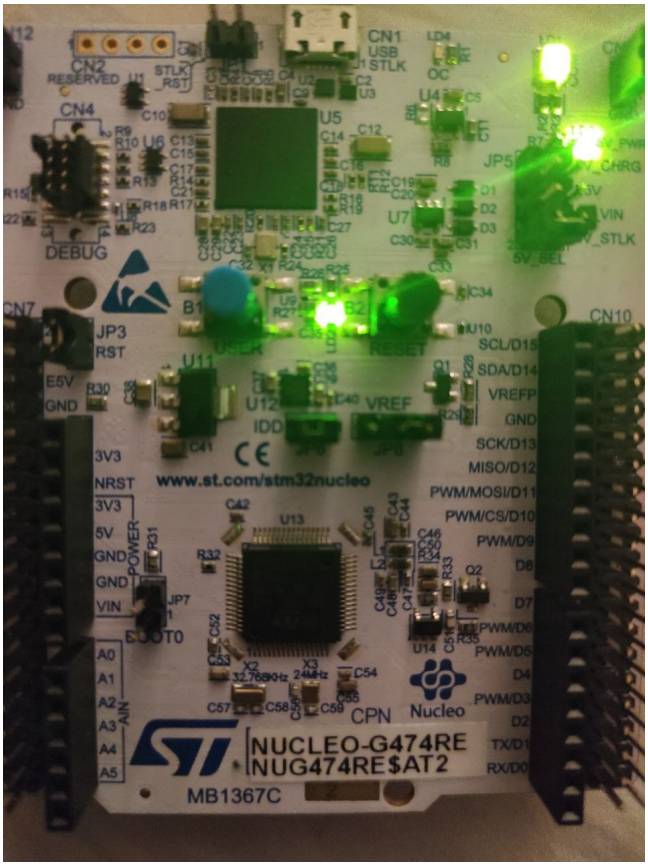


Рисунок 10. Включенный светодиод на отладочной плате.

Согласно документации (приложение 3), нужный светодиод обозначается маркировкой LD2. Это можно увидеть на блок-схеме отладочной платы (рисунок 11). Обозначение «User» подразумевает, что этот светодиод используется в пользовательских целях, то есть в целях разработчика.

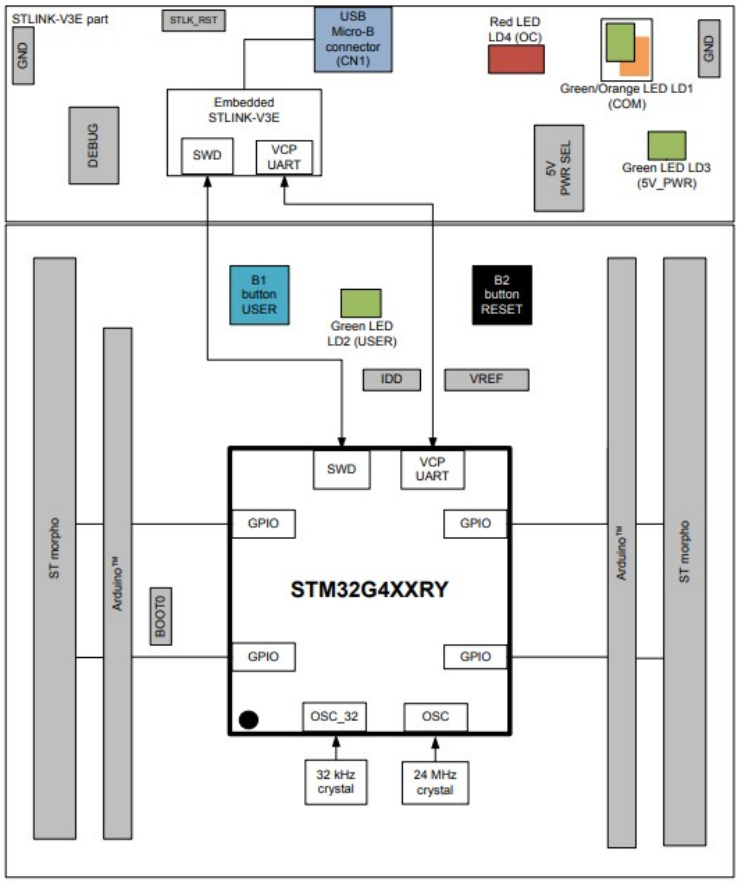


Рисунок 11. Блок-схема отладочной платы STM32G474RE

На принципиальной схеме (рисунок 12) можно увидеть, что это светодиод подключен к питанию и через N-канальный транзистор к пину PA5.

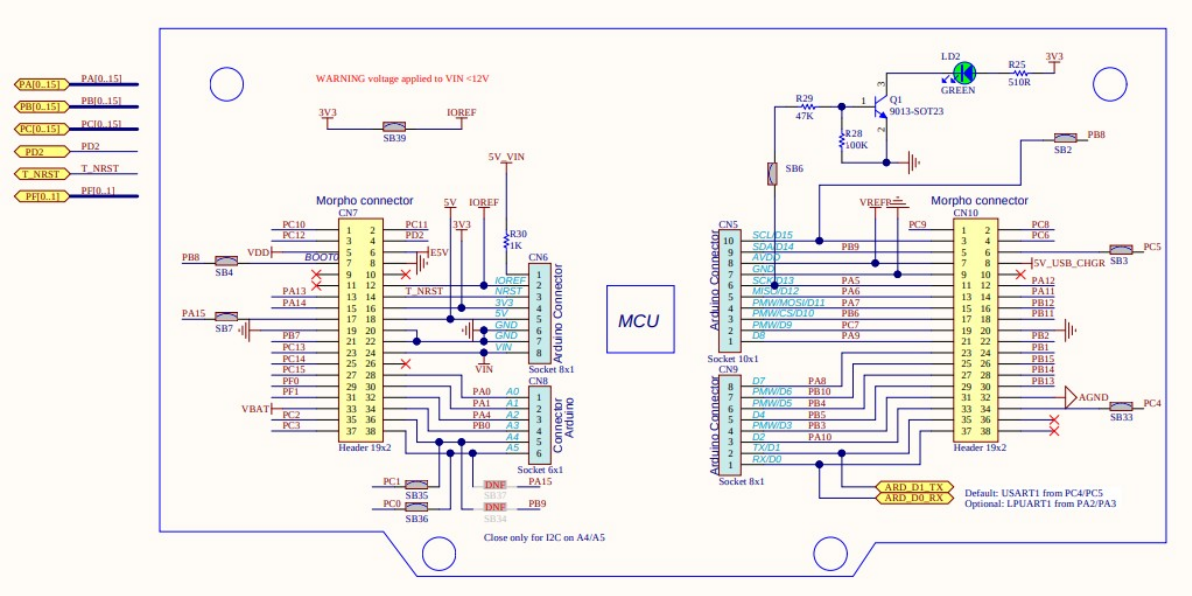
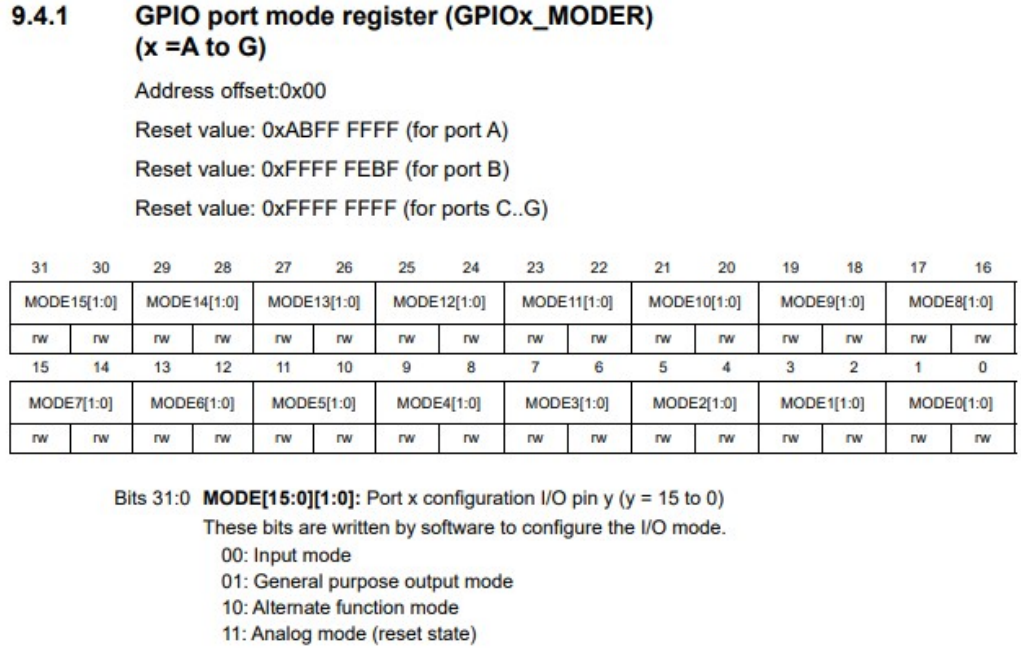


Рисунок 12. Принципиальная схема отладочной платы STM32G474RE.

Это означает, что для того, чтобы зажечь светодиод, необходимо настроить вывод PA5 в режим вывода и выставить уровень логической единицы на этом пине. Каждый порт содержит по 16 пинов, нумеруемых от 0 до 15. Необходимые регистры из приложения 1:

- GPIOx_MODER — регистр режима, отвечает за выбор режима для соответствующего номера пина на выбранном порту путем выставления 2x-битного значения (рисунок 13);
- GPIOx_BSSR — регистр логического уровня, отвечает за настройку выходного логического уровня пина. Регистр делится на 2 части, первая из которых отвечает за значение «BR» (bit reset) выставления логического 0 на пине, а вторая за BS (bit set) — выставления логической единицы. Выставлять одновременно значения BS и BR для одного и того же пина недопустимо (рисунок 14);
- GPIOx_OSPEEDR — регистр настройки скорости обновления сигнала. Зависит от тактирования порта (рисунок 15);
- GPIOx_PUPDR — регистр внутренней подтяжки пина, отвечает за организацию внутренней подтяжки пина к земле или к питанию (рисунок 16);
- GPIOx_IDR — регистр входных данных, содержит входной бит для каждого номера пина (рисунок 17).



9.4.1 GPIO port mode register (GPIOx_MODER) (x =A to G)

Address offset:0x00

Reset value: 0xABFF FFFF (for port A)

Reset value: 0xFFFF FEBF (for port B)

Reset value: 0xFFFF FFFF (for ports C..G)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[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
MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **MODE[15:0][1:0]:** Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O mode.
00: Input mode
01: General purpose output mode
10: Alternate function mode
11: Analog mode (reset state)

Рисунок 13. Регистр MODER модуля GPIO.

9.4.7 GPIO port bit set/reset register (GPIOx_BSRR) (x = A to G)

Address offset: 0x18

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

Bits 31:16 **BR[15:0]:** Port x reset I/O pin y (y = 15 to 0)
These bits are write-only. A read to these bits returns the value 0x0000.
0: No action on the corresponding ODx bit
1: Resets the corresponding ODx bit
Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BS[15:0]:** Port x set I/O pin y (y = 15 to 0)
These bits are write-only. A read to these bits returns the value 0x0000.
0: No action on the corresponding ODx bit
1: Sets the corresponding ODx bit

Рисунок 14. Регистр BSSR модуля GPIO.

9.4.3 GPIO port output speed register (GPIOx_OSPEEDR) (x = A to G)

Address offset: 0x08

Reset value: 0x0C00 0000 (for port A)

Reset value: 0x0000 0000 (for the other ports)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEED15 [1:0]		OSPEED14 [1:0]		OSPEED13 [1:0]		OSPEED12 [1:0]		OSPEED11 [1:0]		OSPEED10 [1:0]		OSPEED9 [1:0]		OSPEED8 [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
OSPEED7 [1:0]		OSPEED6 [1:0]		OSPEED5 [1:0]		OSPEED4 [1:0]		OSPEED3 [1:0]		OSPEED2 [1:0]		OSPEED1 [1:0]		OSPEED0 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **OSPEED[15:0][1:0]**: Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O output speed.
00: Low speed
01: Medium speed
10: High speed
11: Very high speed
Note: Refer to the device datasheet for the frequency specifications and the power supply and load conditions for each speed..

Рисунок 15. Регистр OSPEEDR модуля GPIO.

9.4.4 GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A to G)

Address offset: 0x0C

Reset value: 0x6400 0000 (for port A)

Reset value: 0x0000 0100 (for port B)

Reset value: 0x0000 0000 (for other ports)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPD15[1:0]		PUPD14[1:0]		PUPD13[1:0]		PUPD12[1:0]		PUPD11[1:0]		PUPD10[1:0]		PUPD9[1:0]		PUPD8[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
PUPD7[1:0]		PUPD6[1:0]		PUPD5[1:0]		PUPD4[1:0]		PUPD3[1:0]		PUPD2[1:0]		PUPD1[1:0]		PUPD0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **PUPD[15:0][1:0]:** Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O pull-up or pull-down
00: No pull-up, pull-down
01: Pull-up
10: Pull-down
11: Reserved

Рисунок 16. Регистр PUPDR модуля GPIO

9.4.5 GPIO port input data register (GPIOx_IDR) (x = A to G)

Address offset: 0x10

Reset value: 0x0000 XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ID[15:0]:** Port x input data I/O pin y (y = 15 to 0)
These bits are read-only. They contain the input value of the corresponding I/O port.

Рисунок 17. Регистр IDR модуля GPIO

Порядок действий

1. Выставить выходной режим для пина PA5 в регистре MODER.
2. Настроить скорость работы пина в регистре OSPEEDR в режим HIGH SPEED.
3. Очистить биты подтяжки для режима выхода.
4. Выставить логическую единицу на пине PA5 с помощью регистра BSSR — записать значение BS5 в регистр BSSR.

Получился следующий исходный код (рисунок 18). Теперь можно наблюдать, что светодиод загорелся. Функции CLEAR_BIT и SET_BIT соответственно очищают и выставляют значения макросов в регистре.

```c
#include "gpio.h"

// PA5 - светодиод
void GPIO_Init(void)
{
    /*---------- Настройка пина для светодиода ----------*/
    // Включаем тактирование порта A
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
    // Настраиваем ножку PA5 как выход, для управления светодиодом
    CLEAR_BIT(GPIOA->MODER, GPIO_MODER_MODE5);
    SET_BIT(GPIOA->MODER, GPIO_MODER_MODE5_0);
    // Настраиваем скорость (HIGH SPEED)
    SET_BIT(GPIOA->OSPEEDR, GPIO_OSPEEDER_OSPEEDR5_1);
    // Настраиваем подтяжку
    CLEAR_BIT(GPIOA->PUPDR, GPIO_PUPDR_PUPD5);
    // Выставляем логическую единицу на ножке PA5
    SET_BIT(GPIOA->BSRR, GPIO_BSRR_BS5);
}
```

Рисунок 18. Исходный код включения светодиода на пине PA5.

Задание на самостоятельную работу

Теперь задача усложняется. Необходимо задать внешнее условие переключения светодиода из включенного состояния в выключенное. Для этого необходимо использовать кнопку пользовательского назначения B1(рисунок 11), расположенную на плате и подключенную к питанию и ножке PC13(рисунок 19).

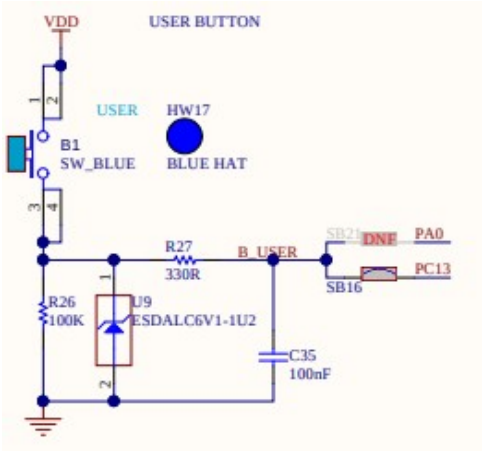


Рисунок 19. Принципиальная схема подключения кнопки B1.

Для считывания состояния этой кнопки достаточно:

1. Включить тактирование на соответствующем пину порту в регистре модуля RCC AHB2ENR;
2. Настроить режим входа на выводе с помощью регистра MODER;
3. Считать состояние вывода PC13 из регистра IDR.

Значения остальных регистров для PC13 необходимо оставить без изменений. **Также необходимо реализовать логику переключения состояния светодиода в зависимости от нажатий на кнопку в основном цикле программы.**

4. Управление состоянием пинов микроконтроллера через прерывания. Модуль EXTI.

В прошлом параграфе был рассмотрен пример базовой работы с интерфейсом GPIO. Легко было заметить эффект дребезга кнопки — процесс, при котором переключение состояния светодиод был не мгновенным (рисунок 20). На практике чаще избегают исполнения программы в основном цикле там, где это возможно. На помощь в этом деле приходит возможность асинхронной аппаратной обработки событий. За это отвечает периферийный модуль EXTI (extended interrupts and events controller) — периферийный модуль прерываний и событий. Этот модуль позволяет предварительно создать обработчик прерывания (функцию), который вызывается только в момент возникновения самого события.

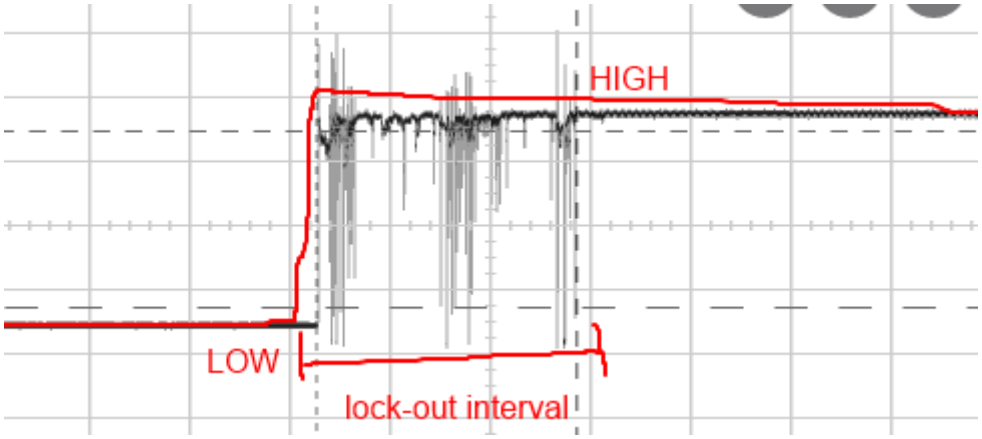


Рисунок 20. Дребезг кнопки.

Для применения на практике этой возможности, необходимо сделать следующее:

1. завести событие на внутреннем или внешнем модуле;
2. *для внешнего прерывания создать прерывание по фронту;
3. сбросить исходное состояние события;
4. включить прерывание;
5. разрешить каналу вложенных векторов NVIC обрабатывать это прерывание;
6. реализовать обработчик события в соответствии с разрешенным каналом, сбрасывая состояние события.

Задание 3. Реализация управления состоянием светодиода с помощью внешних прерываний

Требуется реализовать задачу из прошлого раздела через прерывания. Для этого прежде всего необходимо выбрать нужный канал. В соответствии с картой каналов прерываний из приложения 1 (рисунок 20), выбирается канал EXTI13, поскольку для кнопки используется пин PC13. В самом модуле EXTI также выбирается канал прерываний 13 (рисунок 21). В итоге потребуются регистры:

- SYSCFG_EXTICR4 — регистр внешних прерываний. Разрешает внешнее прерывание, находится на периферийном модуле SYSCFG (рисунок 22).
- EXTI_IMR1 — регистр маски прерываний. Снимает маску с канала прерывания (рисунок 23).
- EXTI_RTSR — регистр настройки фронта прерывания. Настраивает выставление события по восходящему фронту на пине (рисунок 24).

  *Для настройки события по нисходящему фронту существует регистр EXTI_FTSR.
- EXTI_PR1 — регистр событий. Содержит флаги событий. Если событие произошло на канале N, то в регистре PR1 бит PIF(N) содержит 1, в противном случае — 0 (рисунок 25).

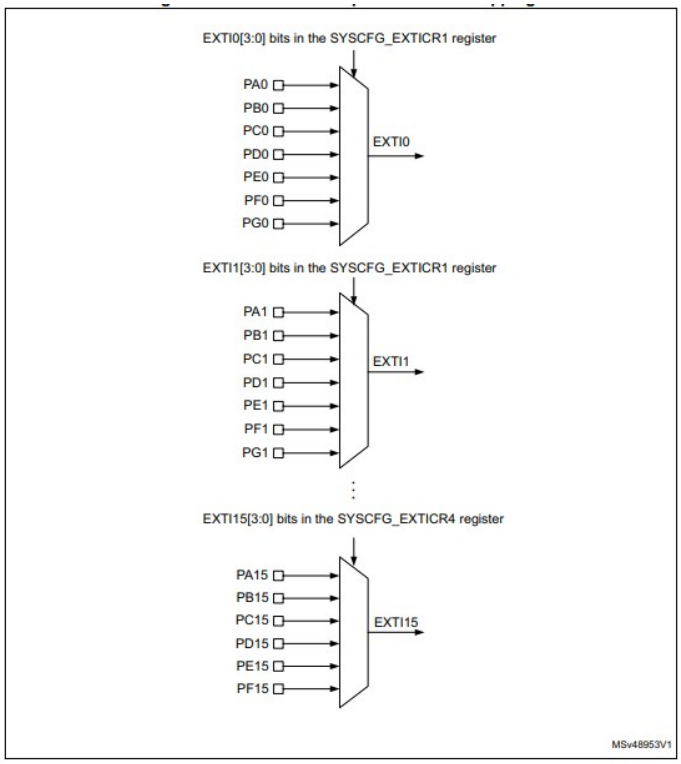


Рисунок 20. Карта каналов прерываний

Table 98. EXTI lines connections

EXTI line	Line source	Line type
0-15	GPIO	configurable
16	PVD	configurable
17	RTC alarm event	configurable
18	USB Device FS wakeup event	direct

Рисунок 21. Карта линий модуля EXTI.

10.2.6 SYSCFG external interrupt configuration register 4 (SYSCFG_EXTICR4)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI15[3:0]				EXTI14[3:0]				EXTI13[3:0]				EXTI12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:12 **EXTI15[3:0]**: EXTI 15 configuration bits

These bits are written by software to select the source input for the EXTI15 external interrupt.

0000: PA[15] pin
0001: PB[15] pin
0010: PC[15] pin
0011: PD[15] pin
0100: PE[15] pin
0101: PF[15] pin

Bits 11:8 **EXTI14[3:0]**: EXTI 14 configuration bits

These bits are written by software to select the source input for the EXTI14 external interrupt.

0000: PA[14] pin
0001: PB[14] pin
0010: PC[14] pin
0011: PD[14] pin
0100: PE[14] pin
0101: PF[14] pin

Bits 7:4 **EXTI13[3:0]**: EXTI 13 configuration bits

These bits are written by software to select the source input for the EXTI13 external interrupt.

0000: PA[13] pin
0001: PB[13] pin
0010: PC[13] pin
0011: PD[13] pin
0100: PE[13] pin
0101: PF[13] pin

Рисунок 22. Регистр EXTICR4 модуля SYSCFG

15.5.1 Interrupt mask register 1 (EXTI_IMR1)

Address offset: 0x00

Reset value: Direct lines are set to '1', others lines are set to '0'. See *Table 98: EXTI lines connections*.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IM31	IM30	IM29	IM28	IM27	IM26	IM25	IM24	IM23	IM22	IM21	IM20	IM19	IM18	IM17	IM16
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
IM15	IM14	IM13	IM12	IM11	IM10	IM9	IM8	IM7	IM6	IM5	IM4	IM3	IM2	IM1	IM0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **IMx:** Interrupt Mask on line x (x = 31 to 0)

0: Interrupt request from Line x is masked
1: Interrupt request from Line x is not masked

Note: The reset value for the direct lines is set to '1' in order to enable the interrupt by default.

Рисунок 23. Регистр IMR1 модуля EXTI

15.5.3 Rising trigger selection register 1 (EXTI_RTSR1)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RT31	RT30	RT29	Res.	Res.	Res.	Res.	Res.	Res.	RT22	RT21	RT20	RT19	Res.	RT17	RT16
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
RT15	RT14	RT13	RT12	RT11	RT10	RT9	RT8	RT7	RT6	RT5	RT4	RT3	RT2	RT1	RT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:29 **RTx:** Rising trigger event configuration bit of line x (x = 31 to 29)
0: Rising trigger disabled (for Event and Interrupt) for input line
1: Rising trigger enabled (for Event and Interrupt) for input line

Bits 28:23 Reserved, must be kept at reset value.

Bits 22:19 **RTx:** Rising trigger event configuration bit of line x (x = 22 to 19)
0: Rising trigger disabled (for Event and Interrupt) for input line
1: Rising trigger enabled (for Event and Interrupt) for input line

Bit 18 Reserved, must be kept at reset value.

Bits 17:0 **RTx:** Rising trigger event configuration bit of line x (x = 17 to 0)
0: Rising trigger disabled (for Event and Interrupt) for input line
1: Rising trigger enabled (for Event and Interrupt) for input line

Рисунок 24. Регистр RTSR1 модуля EXTI

15.5.6 Pending register 1 (EXTI_PR1)

Address offset: 0x14

Reset value: undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PIF31	PIF30	PIF29	Res.	Res.	Res.	Res.	Res.	Res.	PIF22	PIF21	PIF20	PIF19	Res.	PIF17	PIF16
rc_w1	rc_w1	rc_w1							rc_w1	rc_w1	rc_w1	rc_w1		rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PIF15	PIF14	PIF13	PIF12	PIF11	PIF10	PIF9	PIF8	PIF7	PIF6	PIF5	PIF4	PIF3	PIF2	PIF1	PIF0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:29 **PIFx:** Pending interrupt flag on line x (x = 31 to 29)
0: No trigger request occurred
1: Selected trigger request occurred
This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a '1' to the bit.

Bits 28:23 Reserved, must be kept at reset value.

Bits 22:19 **PIFx:** Pending interrupt flag on line x (x = 22 to 19)
0: No trigger request occurred
1: Selected trigger request occurred
This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a '1' to the bit.

Bit 18 Reserved, must be kept at reset value.

Bits 17:0 **PIFx:** Pending interrupt flag on line x (x = 17 to 0)
0: No trigger request occurred
1: Selected trigger request occurred
This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a '1' to the bit.

Рисунок 25. Регистр PR1 модуля EXTI

Порядок действий

1. Включить тактирование SYSCFGR.
2. Выставить биты в EXTI13 для использования порта PC как источника внешних прерываний.
3. Включить прерывания на канале 13 в EXTI путем выставления битов в IM13 (т.к. каналы прерывания от 0 до 15 полностью повторяют номера пинов).

4. Настроить возникновение события по восходящему фронту на канале 13 в регистре RTSR1.
5. Разрешить прерывания в контроллере вложенных векторов прерывания через вызов системной функции NVIC_EnableIRQ (EXTI15_10_IRQn). EXTI15_10 — акроним названия вектора прерывания для линий 10 - 15 из приложения 1(рисунок 26).
6. Сбросить флаг прерывания в регистре PR1 путем записи в бит PIF13 1.

Table 97. STM32G4 Series vector table

Position	Priority	Type of priority	Acronym	Description	Address
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000 00E0

Рисунок 26. Таблица векторов прерываний.

Получится следующий исходный код (рисунок 27).

```c
#include "exti.h"

void EXTI_Init(void)
{
    // Включаем такирование SysGonfig
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
    // Настраиваем пин для использования как EXTI
    SYSCFG->EXTICR[3] |= SYSCFG_EXTICR4_EXTI13_PC;
    // Включаем прерывание
    //EXTI->IMR1 |= EXTI_IMR1_IM13;
    SET_BIT(EXTI->IMR1, EXTI_IMR1_IM13);
    //Настраиваем фронт, по которому будет срабатывать прерывание
    //EXTI->RTSR1 |= EXTI_RTSR1_RT13;
    SET_BIT(EXTI->RTSR1, EXTI_RTSR1_RT13);
    // Разрешаем прерывание
    NVIC_EnableIRQ(EXTI15_10_IRQn);
    // Очищаем событие
    SET_BIT(EXTI->PR1, EXTI_PR1_PIF13);
}
```

Рисунок 27. Исходный код настройки внешних прерываний по нажатию кнопки на пине PC13.

Задание на самостоятельную работу.

Реализовать обработчик прерывания — функцию

`void EXTI15_10_IRQHandler(void)`, в которую необходимо перенести логику работы программы из прошлого раздела. В коде не должно быть циклов

*Замечание. Первой командой в обработчике прерываний должна быть очистка события в регистре PR1 на канале 13.

5. Работа с таймером в базовом режиме. Модуль TIM.

Базовый модуль TIMER (рисунок 27) — еще одна из важнейших аппаратных возможностей микроконтроллеров. Представляет из себя реализацию счетчика на внешнем устройстве и может работать независимо от основной программы. Этим часто пользуются для реализации задержек или каких-либо иных функций подсчета. Получается очень удобное решение, позволяющее выполнять ожидание какого-либо процесса вне основной программы. В это же время ресурсы контроллера свободны, и программа может выполнять другие полезные задачи. Также этот модуль позволяет совместно с модулем прерываний настроить асинхронную работу с какими-либо процессами и реализовать всю логику обработки этих процессов исключительно на обработчиках прерываний.

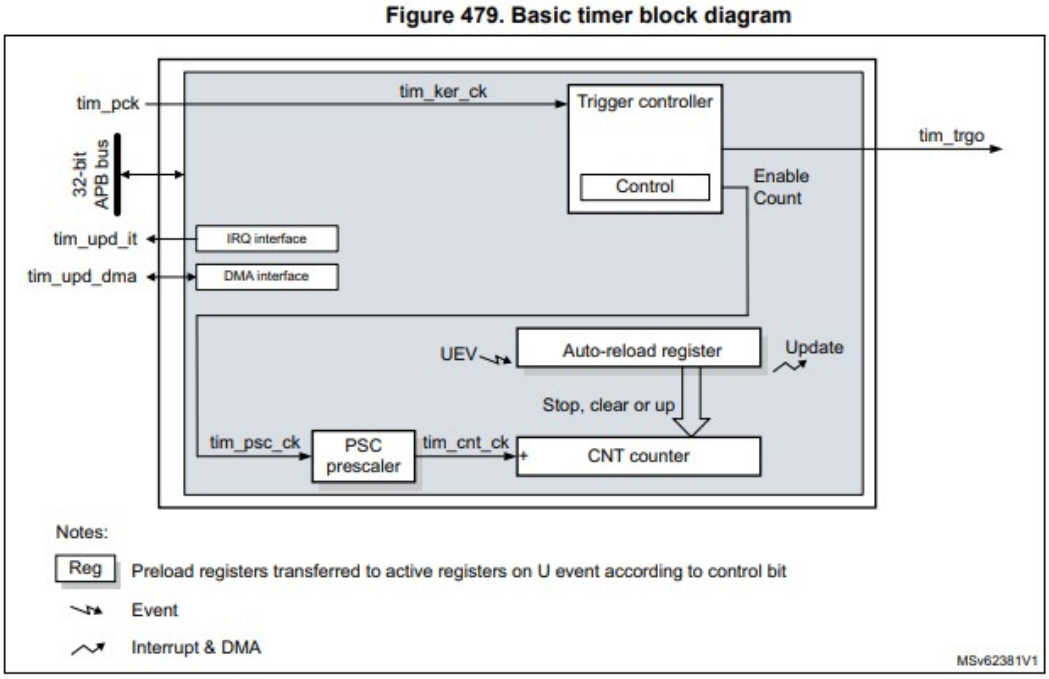


Рисунок 27. Блок-схема базового таймера

Для включения таймера необходим сделать следующее:

1. Включить тактирование таймера
2. Настроить предделитель частоты
3. Выставить период таймера

Задание 4. Настройка мигания светодиода с помощью таймера.

В предыдущем разделе было рассмотрено изменение состояния светодиода по внешнему событию — нажатию кнопки. Теперь в рамках изучения таймера можно заставить светодиод мигать, то есть периодически изменять свое состояние. Это можно было сделать и в основном цикле программы, однако STM32G474RE — это однопроцессорный контроллер, и занимать ресурсы на прокрутку программы «вхолостую», отсчитывая время, нецелесообразно, поскольку для таких задач и существует таймер. Для этой задачи понадобятся следующие регистры:

- TIMx_CR1 — регистр управления. Задает основные настройки таймера, производит его включение (рисунок 28).
- TIMx_PSC — регистр предделителя. Задает предделитель тактовой частоты таймера, что позволяет гибко настраивать периода отсчета (рисунок 29).
- TIMx_ARR — регистр перезапуска. Задает период таймера (рисунок 30).
- TIMx_SR — регистр состояния. Содержит состояние таймера (рисунок 31).

31.4.1 TIMx control register 1 (TIMx_CR1)(x = 6 to 7)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	DITHEN	UIFREMAP	Res.	Res.	Res.	ARPE	Res.	Res.	Res.	OPM	URS	UDIS	CEN
			rw	rw				rw				rw	rw	rw	rw

Рисунок 28. Регистр CR1 модуля TIM.

31.4.7 TIMx prescaler (TIMx_PSC)(x = 6 to 7)

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

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency $f_{tim\_cnt\_ck}$ is equal to $f_{tim\_psc\_ck}$ / (PSC[15:0] + 1).
PSC contains the value to be loaded into the active prescaler register at each update event.
(including when the counter is cleared through UG bit of TIMx_EGR register.

Рисунок 29. Регистр PSC модуля TIM.

31.4.8 TIMx auto-reload register (TIMx_ARR)(x = 6 to 7)

Address offset: 0x2C

Reset value: 0x0000 FFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ARR[19:16]			
												rw	rw	rw	rw
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

Bits 31:20 Reserved, must be kept at reset value.

Bits 19:0 **ARR[19:0]**: Auto-reload value

ARR is the value to be loaded into the actual auto-reload register.
Refer to *Section 31.3.4: Time-base unit on page 1451* for more details about ARR update and behavior.
The counter is blocked while the auto-reload value is null.
Non-dithering mode (DITHEN = 0)
The register holds the auto-reload value in ARR[15:0]. The ARR[19:16] bits are reserved.
Dithering mode (DITHEN = 1)
The register holds the integer part in ARR[19:4]. The ARR[3:0] bitfield contains the dithered part.

Рисунок 30. Регистр ARR модуля TIM.

31.4.4 TIMx status register (TIMx_SR)(x = 6 to 7)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	UIF
															rc_w0

Bits 15:1 Reserved, must be kept at reset value.

Bit 0 **UIF**: Update interrupt flag
This bit is set by hardware on an update event. It is cleared by software.
0: No update occurred.
1: Update interrupt pending. This bit is set by hardware when the registers are updated:
– On counter overflow if UDIS = 0 in the TIMx_CR1 register.
– When CNT is reinitialized by software using the UG bit in the TIMx_EGR register, if URS = 0 and UDIS = 0 in the TIMx_CR1 register.

Рисунок 31. Регистр CR модуля TIM.

Порядок действий

1. Включить тактирование на TIM6 на шине APB1ENR1.
2. Выставить значение предделителя в регистре PSC.
3. Выставить период таймера в регистре ARR.
4. Включить таймер выставлением бита в CEN в регистре CR1.

Полученный код (рисунок 32).

```c
#include "tim.h"

void Delay_TIM(uint32_t delay)
{
    RCC->APB1ENR1 |= RCC_APB1ENR1_TIM6EN;
    TIM6->PSC = 20;
    TIM6->ARR = delay;
    TIM6->CR1 |= TIM_CR1_CEN;
}
```

```c
int main(void)
{
    RCC_Init();
    GPIO_Init();
    Delay_TIM(1000);
    while(1)
    {

        if (!READ_BIT(TIM6->SR, TIM_SR_UIF))
        {
                /* code */

        }
    }
}
```

Рисунок 32. Исходный код реализации задержки с помощью аппаратного таймера.

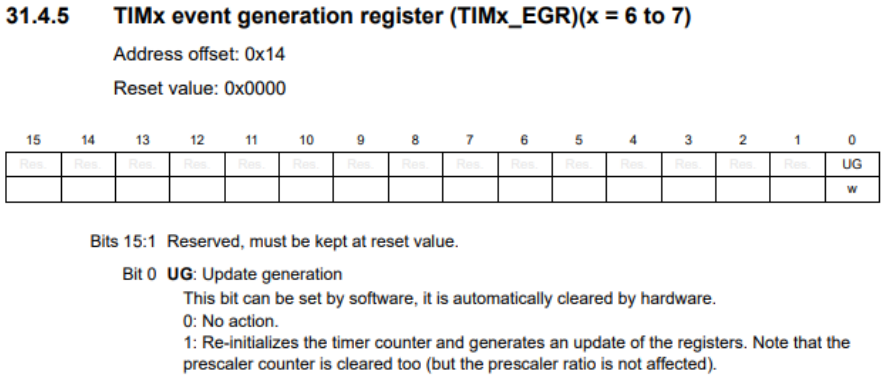
Задание на самостоятельную работу.

Реализовать мигание светодиода с использованием аппаратного таймера.

*Задание 5. Настройка мигания светодиода с помощью таймера с использованием прерывания.

В предыдущей реализации обработка осуществлялась в основном цикле, и, хотя процесс подсчета был вынесен на аппаратный модуль, событие обрабатывалось синхронно. В целях экономии ресурсов контроллера такую обработку можно осуществлять полностью асинхронно. Поэтому в следующем задании та же логика будет реализована с использованием возможностей прерывания самого модуля TIM. Условием возникновения прерывания будет достижение таймером периода. Для этого дополнительно понадобятся следующие регистры:

- TIMx_EGR — регистр создания события. Разрешает создание события (рисунок33).
- TIMx_DIER — регистр разрешения прерывания (рисунок 34).



31.4.5 TIMx event generation register (TIMx_EGR)(x = 6 to 7)

Address offset: 0x14

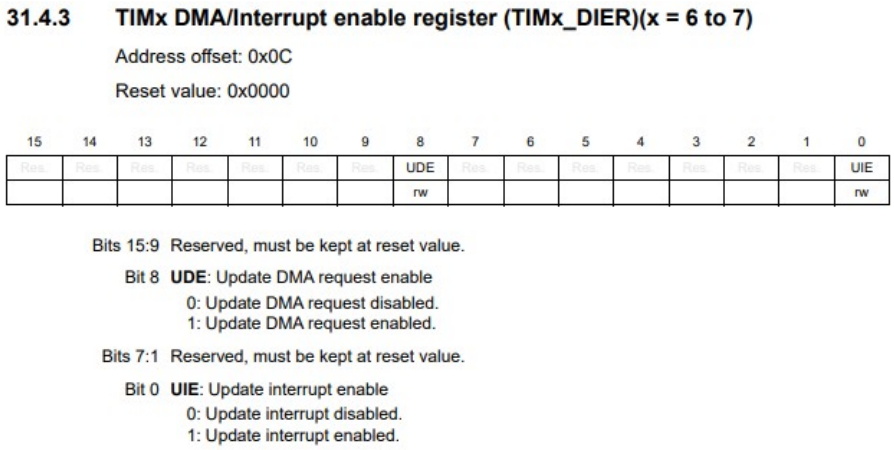
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	UG
															w

Bits 15:1 Reserved, must be kept at reset value.

Bit 0 **UG**: Update generation
This bit can be set by software, it is automatically cleared by hardware.
0: No action.
1: Re-initializes the timer counter and generates an update of the registers. Note that the prescaler counter is cleared too (but the prescaler ratio is not affected).

Рисунок 33. Регистр EGR модуля TIM.



31.4.3 TIMx DMA/Interrupt enable register (TIMx_DIER)(x = 6 to 7)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	UDE	Res.	Res.	Res.	Res.	Res.	Res.	Res.	UIE
							rw								rw

Bits 15:9 Reserved, must be kept at reset value.

Bit 8 **UDE**: Update DMA request enable
0: Update DMA request disabled.
1: Update DMA request enabled.

Bits 7:1 Reserved, must be kept at reset value.

Bit 0 **UIE**: Update interrupt enable
0: Update interrupt disabled.
1: Update interrupt enabled.

Рисунок 34. Регистр DIER модуля TIM.

Порядок действий.

1. Разрешить создание события в регистре EGR.
2. Разрешить прерывания в регистре DIER.

3. Разрешить прерывания в контроллере вложенных векторов прерывания через вызов системной функции NVIC_EnableIRQ (TIM6_IRQn). TIM6 — акроним названия вектора прерывания для таймера 6 из приложения 1.
4. Сбросить флаг прерывания в регистре SR путем записи в бит UIF 0.

Получается следующий исходный код (рисунок 35).

```
#include "tim.h"

void Delay_TIM(uint32_t delay)
{
    RCC->APB1ENR1 |= RCC_APB1ENR1_TIM6EN;
    TIM6->PSC = 20;
    TIM6->ARR = delay;
    TIM6->EGR |= TIM_EGR_UG;
    TIM6->DIER |= TIM_DIER_UIE;
    NVIC_EnableIRQ(TIM6_IRQn);
    TIM6->SR = 0;
    TIM6->CR1 |= TIM_CR1_CEN;
}
```

Рисунок 35. Исходный код настройки таймера с прерыванием.

Задание на самостоятельную работу.

Реализовать обработчик прерывания — функцию `void TIM6_IRQHandler (void)`.

*Замечание. Первой командой в обработчике прерываний должна быть очистка события в регистре SR модуля TIM6.

6. Продвинутая работа с таймерами. Режим ШИМ и Захвата/Сравнения.

В этом разделе будут рассмотрены продвинутые возможности таймера. Основополагающими режимами таймера являются ШИМ и захват/сравнение. Две основные возможности, реализуемые модулями захвата/сравнения (рисунок 36):

- По событиям на внешних выводах запоминать в регистрах захвата/сравнения значение счётчика («захват», в этом случае внешние выводы используются в качестве входов)
- Управлять внешними выводами в зависимости от результатов сравнения значений счётчика и регистров захвата/сравнения («сравнение», в этом случае внешние выводы используются в качестве выходов)

Также модули захвата/сравнения позволяют реализовывать на практике измерение параметров сигналов (период/длительность импульса), генерацию сигналов с заданными параметрами, широтно-импульсную модуляцию и т.д.

В свою очередь ШИМ — наиболее частый режим использования модуля захвата/сравнения. Его задача состоит в генерации прямоугольных импульсов с настраиваемой частотой и настраиваемым периодом состояния сигнала. В связке с усилителем мощности этот режим может фактически позволить управлять напряжением на выходе платы управления двигателем.

Далее будут рассмотрены режим захвата и режим ШИМ.

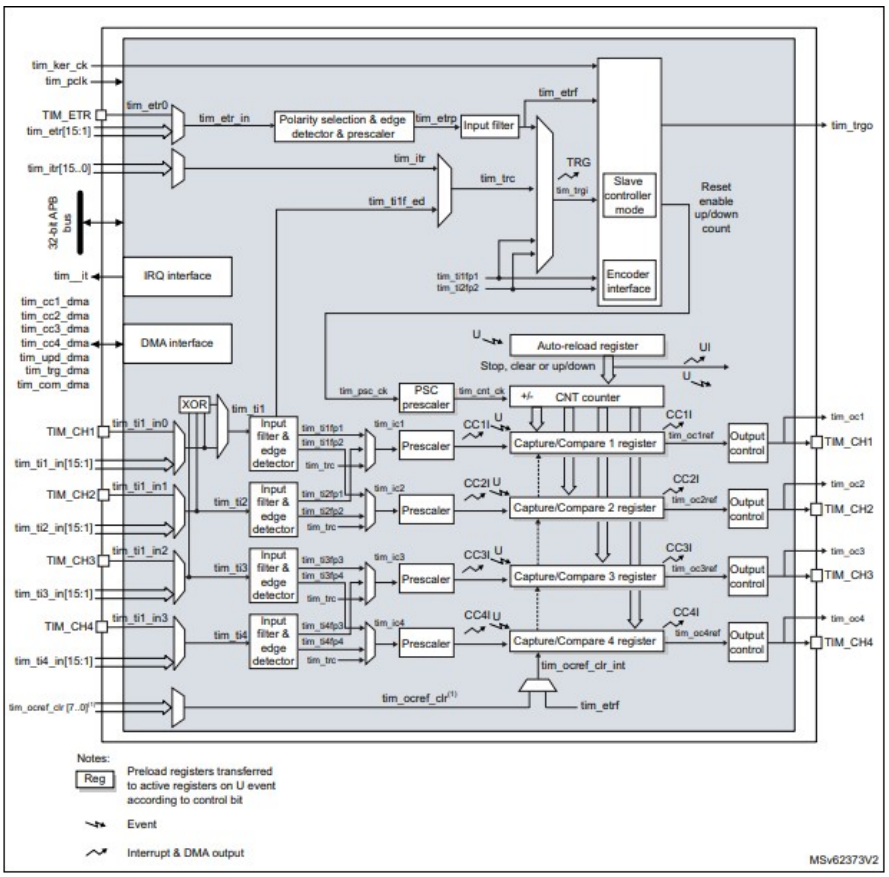


Рисунок 36. Блок схема таймера общего назначения

Задание 6. Настроить ШИМ с частотой 200 ГЦ.

Работа с модулем захвата/сравнения в таймере общего назначения строится вокруг регистра CCR (capture/compare register) (рисунок 36). В разных режимах этот регистр имеет разное назначение. В случае захвата регистр CCR служит регистром хранения «захваченного» значения. В случае ШИМ регистр CCR хранит заранее записанное значение, которое сравнивается с числом отсчитанных импульсов, и в случае равенства — переключает состояние выходного сигнала. Фактически в режиме ШИМ регистр CCR определяет период импульса. Другие регистры из модуля таймера общего назначения, которые понадобятся для настройки:

- TIMx_CR1 — регистр управления. Задает основные настройки таймера, производит его включение (рисунок 37).
- TIMx_PSC — регистр предделителя. Задает предделитель тактовой частоты таймера. Определяет частоту сигнала на входе/выходе (рисунок 29). Формула для вычисления частоты: **ftim_psc_ck / (PSC[15:0] + 1)**. Аналогичен базовому таймеру.
- TIMx_ARR — регистр перезапуска. Задает период таймера (рисунок 30). Аналогичен базовому таймеру.
- TIMx_SR — регистр состояния. Содержит состояние таймера (рисунок 38).

- TIMx_CCMR1 — регистр режима захвата/сравнения. Настраивает режим на первых двух каналах. Следующие два канала настраиваются через TIMx_CCMR2. Для режимов входа и выхода содержит разные настройки (рисунок 39 и рисунок 40).
- TIMx_CCER — регистр разрешения режима захвата/сравнения на каналах. Включает режим захвата/сравнения на различных каналах (рисунок 41).
- TIMx_CCRn — регистр захвата/сравнения канала номера n. Представляет собой 16-битное число, в режиме захвата — хранит «захваченное» значение с входа канала n. В режиме сравнения — меняет состояние выхода при достижении счетчиком значения в этом регистре (рисунок 42).

29.5.1 TIMx control register 1 (TIMx_CR1)(x = 2 to 5)

Address offset: 0x000

Reset value: 0x0000

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

Рисунок 37. Регистр CR2 модуля TIM general purpose.

29.5.5 TIMx status register (TIMx_SR)(x = 2 to 5)

Address offset: 0x010

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TERRF	IERRF	DIRF	IDXF	Res.	Res.	Res.	Res.
								rc_w0	rc_w0	rc_w0	rc_w0				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	CC4OF	CC3OF	CC2OF	CC1OF	Res.	Res.	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 38. Регистр SR модуля TIM general purpose.

29.5.8 TIMx capture/compare mode register 1 [alternate] (TIMx_CCMR1)(x = 2 to 5)

Address offset: 0x018

Reset value: 0x0000 0000

The same register can be used for output compare mode (this section) or for input capture mode (previous section). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function for input capture and for output compare modes. It is possible to combine both modes independently (e.g. channel 1 in input capture mode and channel 2 in output compare mode).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC2M[3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC1M[3]
							rw								rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 39. Регистр CCMR1 для выходных режимов модуля TIM general purpose.

29.5.7 TIMx capture/compare mode register 1 [alternate] (TIMx_CCMR1)(x = 2 to 5)

Address offset: 0x018

Reset value: 0x0000 0000

The same register can be used for input capture mode (this section) or for output compare mode (next section). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function for input capture and for output compare modes. It is possible to combine both modes independently (e.g. channel 1 in input capture mode and channel 2 in output compare mode).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]		IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 40. Регистр CCMR1 для входных режимов модуля TIM general purpose.

29.5.11 TIMx capture/compare enable register (TIMx_CCER)(x = 2 to 5)

Address offset: 0x020

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC4NP	Res.	CC4P	CC4E	CC3NP	Res.	CC3P	CC3E	CC2NP	Res.	CC2P	CC2E	CC1NP	Res.	CC1P	CC1E
rw		rw	rw	rw		rw	rw	rw		rw	rw	rw		rw	rw

Bit 15 **CC4NP**: *Capture/Compare 4 output Polarity.*
Refer to CC1NP description

Bit 14 Reserved, must be kept at reset value.

Bit 13 **CC4P**: *Capture/Compare 4 output Polarity.*
Refer to CC1P description

Bit 12 **CC4E**: *Capture/Compare 4 output enable.*
refer to CC1E description

Bit 11 **CC3NP**: *Capture/Compare 3 output Polarity.*
Refer to CC1NP description

Bit 10 Reserved, must be kept at reset value.

Рисунок 41. Регистр CCER модуля TIM general purpose.

29.5.17 TIMx capture/compare register 1 (TIMx_CCR1)(x = 3, 4)

Address offset: 0x034

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CCR1[19:16]			
												rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Риуинок 42. Регистр CCRn модуля TIM general purpose.

Предварительный этап

Помимо регистров таймера также потребуется использовать модуль GPIO. До этого настройка выходов контроллера производилась для использования их контроллером как вход или выход. Сейчас же управление выходом необходимо «передать» аппаратной периферии в виде таймера. Это можно сделать путем настройки вывода в режим альтернативной функции.

Согласно документации (приложение 2, таблица 13) вывод PA5 можно использовать в качестве канала ввода/вывода номер 1, модуля TIM2 (рисунок 43). Для этого достаточно настроить на этом выводе режим альтернативной функции 1 (AF1). Режим альтернативной функции выставляется с помощью 4-разрядного числа, значение которого соответствует выбранному номеру альтернативной функции, то есть 0000 → AF0; 0001 → AF1; …; 1111 → AF15. Из-за того, что альтернативных функций предусмотрено достаточно много для одного вывода (16 режимов), и для их задания требуются 4-битные значения, настройка альтернативных функций для порта GPIO (A,B,C,D..) с 16 выводами не поместилась в один 32-разрядный регистр. Поэтому таких регистров существует 2 — AFLR (рисунок 44) и AFHR (рисунок 45). Настройка вывода на порту производится с 0 по 7 номеров в первом регистре и с 8 по 15 во втором. В результате получается следующий исходный код (рисунок 46).

Port		AF0	AF1	AF2	AF3	
		I2C4/ SYS_AF	LPTIM1/ TIM2/5/ 15/16/17	I2C1/3/ TIM1/2/3/4/5/8/ 20/15/ COMP1	QUADSPI1/ I2C3/4/SAI1/USB/HRTIM1/ TIM8/20/15/ COMP3	I2 4/
Port A	PA0	-	TIM2_CH1	TIM5_CH1	-	
	PA1	RTC_ REFIN	TIM2_CH2	TIM5_CH2	-	
	PA2	-	TIM2_CH3	TIM5_CH3	-	
	PA3	-	TIM2_CH4	TIM5_CH4	SAI1_CK1	
	PA4	-	-	TIM3_CH2	-	
	PA5	-	TIM2_CH1	TIM2_ETR	-	
	PA6	-	TIM16_CH1	TIM3_CH1	-	
	PA7	-	TIM17_CH1	TIM3_CH2	-	
	PA8	MCO	-	I2C3_SCL	-	

Рисунок 43. Часть таблицы из приложения 2 с альтернативным функциями для PA5.

9.4.9 GPIO alternate function low register (GPIOx_AFRL) (x = A to G)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFSEL7[3:0]				AFSEL6[3:0]				AFSEL5[3:0]				AFSEL4[3:0]			
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
AFSEL3[3:0]				AFSEL2[3:0]				AFSEL1[3:0]				AFSEL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 44. Регистр AFLR модуля GPIO.

9.4.10 GPIO alternate function high register (GPIOx_AFRH) (x = A to G)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFSEL15[3:0]				AFSEL14[3:0]				AFSEL13[3:0]				AFSEL12[3:0]			
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
AFSEL11[3:0]				AFSEL10[3:0]				AFSEL9[3:0]				AFSEL8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 45. Регистр AFHR модуля GPIO.

```
void GPIO_Init(void)
{
    /*---------- Настройка пина ШИМ ----------*/
    // AF1 -> TIM2_CH1
    // Включаем тактирование порта A
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
    // Настраиваем ножку PA5 как выход, для управления светодиодом
    CLEAR_BIT(GPIOA->MODER, GPIO_MODER_MODE5);
    SET_BIT(GPIOA->MODER, GPIO_MODER_MODE5_1);
    // Настраиваем скорость (HIGH SPEED)
    SET_BIT(GPIOA->OSPEEDR, GPIO_OSPEEDER_OSPEEDR5);
    //Настраиваем подтяжку (без подтяжки)
    CLEAR_BIT(GPIOA->PUPDR, GPIO_PUPDR_PUPD5);
    //Выбираем альтернативную функцию для использования TIM2_CH1
    GPIOA->AFR[0] |= GPIO_AFRL_AFSEL5_0;
```

Рисунок 46. Исходный код настройки PA5 в режим альтернативной функции.

Порядок действий.

1. Включить тактирование на выводе PA5.
2. Настроить выход PA5 в режим альтернативной функции AF1 (передать управление выходом аппаратному периферийному модулю).
3. Включить тактирование таймера TIM2 на шине APB1.
4. Выставить предделитель 239 в регистре PSC. Расчет частоты тактирования таймера:

   ftim_psc_ck = 24МГц (HSE);

   ftim_clock = ftim_psc_ck / (PSC[15:0] + 1) = 24 МГц / 240 = 100 кГц — частота тактирования таймера.
5. Выставить частоту работы таймера относительно тактовой частоты, записав значение 500 в регистр ARR.

   tim_reload = 500;

   ftim = ftim_clock / tim_reload = 100 кГц / 500 = 200 Гц.
6. Настроить режим вывода канала 1 в качестве выхода, записав значение 00 в поле CC1S регистра CCMR1.
7. Выставить в регистре CCR1 скважность ШИМ в 50%. Согласно режиму PWM1 выход канала активен до тех пор, пока значение счетчика меньше значения CCR1. Для этого

в регистр CCR1 необходимо записать значение, равное 250, т.е. половину от величины периода таймера.

8. Настроить выходной режим на канале 1 в PWM mode 1 путем выставления в поле OC1M битов 0110.

Получается следующий исходный код (рисунок 47). После запуска можно заметить, что светодиод светится в 2 раза тусклее, чем при подаче прямого сигнала.

```
void TIM2_Init_PWM(void)
{
    RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN;
    // Настраиваем предделитель
    TIM2->PSC = 239;
    // Настраиваем регистр периода
    TIM2->ARR = 500;
    // Настраиваем 1 канал как выход (Ouptut mode)
    CLEAR_BIT(TIM2->CCMR1, TIM_CCMR1_CC1S);
    // Настраиваем режим в PWM mode 1
    TIM2->CCMR1 |= TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2;
    // Настройка переключения состояния сигнала
    TIM2->CCR1 = 250;
    // Включаем 1 канал таймера
    SET_BIT(TIM2->CCER, TIM_CCER_CC1E);
}
// /*
```

Рисунок 47. Исходный код настройки ШИМ на канале 1 таймера TIM2.

Задание на самостоятельную работу 1.

Настроить мигание светодиодом с частотой 0,5 Гц аппаратными средствами с помощью режима ШИМ. В течение 2 секунд светодиод горит, в течение еще 2 секунд — не горит.

Задание на самостоятельную работу 2.

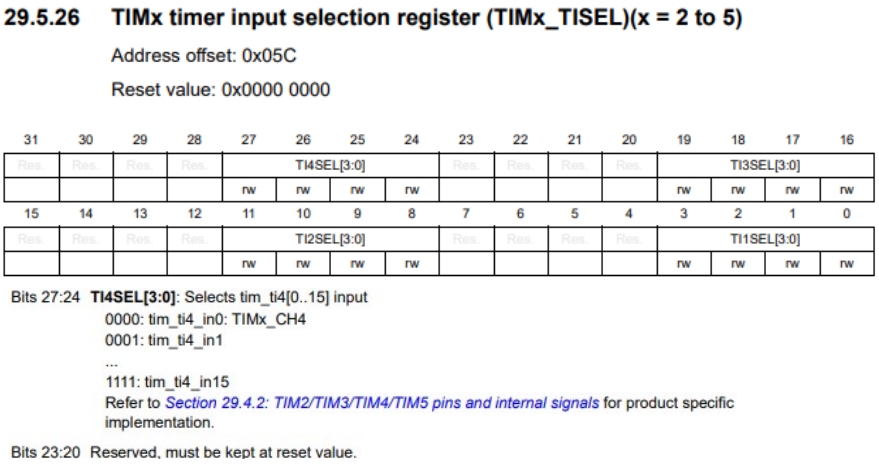
В предыдущем задании было рассмотрен принцип модуляции — получение квазинепрерывного сигнала на выводе контроллера. Яркость светодиода зависит от напряжения на нем, которое в свою очередь зависит от коэффициента заполнения ШИМ на выводе. Для дальнейшего задания необходимо программно изменять скважность импульсов от 0% до 100%.

Требуется настроить «плавное» мигание светодиодом с произвольной частотой аппаратными средствами с помощью режима ШИМ. Частота таймера должна быть более 200 Гц во избежание мерцания светодиода.

Задание 7. Настроить захват — вычислить частоту ШИМ.

В данном задании будет рассмотрен захват значения из внешней среды. Ранее регистр CCR использовался для сравнения значения счетчика. Теперь же CCR будет использоваться для захвата значения по импульсу счетчика. Кроме регистров, рассмотренных выше,

дополнительно понадобится регистр TISEL (рисунок 48). Этот регистр определяет источник для захвата значения. Источником могут быть до 16 различных сигналов, как внутренних, так и внешних (рисунок 36). В качестве источника выбирается вывод канала CH2, что соответствует значению tim_ti2_in0.



29.5.26 TIMx timer input selection register (TIMx_TISEL)(x = 2 to 5)

Address offset: 0x05C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	TI4SEL[3:0]				Res.	Res.	Res.	Res.	TI3SEL[3:0]			
				rw	rw	rw	rw					rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	TI2SEL[3:0]				Res.	Res.	Res.	Res.	TI1SEL[3:0]			
				rw	rw	rw	rw					rw	rw	rw	rw

Bits 27:24 **TI4SEL[3:0]**: Selects tim_ti4[0..15] input
0000: tim_ti4_in0: TIMx_CH4
0001: tim_ti4_in1
...
1111: tim_ti4_in15
Refer to *Section 29.4.2: TIM2/TIM3/TIM4/TIM5 pins and internal signals* for product specific implementation.

Bits 23:20 Reserved, must be kept at reset value.

Рисунок 48. Регистр TISEL модуля TIM.

Порядок действий.

9. Включить тактирование на выводе PA1.
10. Настроить выход PA1 в режим альтернативной функции AF1 (передать управление выходом аппаратному периферийному модулю).
11. Включить тактирование таймера TIM2 на шине APB1.
12. Выставить предделитель 239 в регистре PSC. Расчет частоты тактирования таймера.
13. Выставить источник входного сигнала с вывода CH1 (tim_ti2_in0) в регистре TISEL.
14. Настроить режим вывода канала 1 в качестве входа, записью значения 01 в поле CC2S регистра CCMR1.
15. Настроить полярность, по которой будет происходить захват. При нисходящем фронте захватывать значение. Выставить бит CC2P в регистре CCER.
16. Включить канал номер 2 таймера выставлением бита в поле CC2E в регистре CCER.
17. Включить прерывание по захвату значения в регистре DIER выставлением бита CC2IE.
18. Разрешить прерывание в контроллере вложенных прерываний через вызов системной функции NVIC_EnableIRQ (TIM2_IRQn).

Полученный исходный код (рисунок 49). Здесь по событию захвата значения будет вызвано прерывание. В результате обработки этого прерывания можно далее составить программу для вычисления периода ШИМ. Необходимо соединить проводом каналы таймера TIM2 №1 и № 2, PA1 и PA5.

```
void TIM2_Init_InputCapture(void)
{
    RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN;
    // Настраиваем предделитель
    TIM2->PSC = 239;
    // Настравиаем работу 2-го канала как вход
    TIM2->CCMR1 |= TIM_CCMR1_CC2S_0;
    // Настраиваем источник входного сигнала для 2-го канала (источник с пина)
    //TIM2->TISEL &= ~TIM_TISEL_TI1SEL_0;
    TIM2->TISEL |= TIM_TISEL_TI1SEL_0;
    // Настраиваем полярность входящего сигнала (захватываем по по спаду)
    //TIM2->CCER &= ~(TIM_CCER_CC2P | TIM_CCER_CC2NP);
    TIM2->CCER |= TIM_CCER_CC2P;
    // Включаем 2 канал таймера
    TIM2->CCER |= TIM_CCER_CC2E;
    // Включаем прерване по захвату
    TIM2->DIER |= TIM_DIER_CC2IE;
    NVIC_EnableIRQ(TIM2_IRQn);

}
```

Рисунок 49. Исходный код настройки захвата на PA1.

Задание на самостоятельную работу 1.

Реализовать обработчик прерывания таймера TIM2,

`void TIM2_IRQHandler (void)` (рисунок 50), в котором необходимо вычислить период запущенного ШИМ на таймере TIM2, канале CH1.

```
void TIM2_IRQHandler(void)
{
    CLEAR_BIT(TIM2->SR, TIM_SR_CC2IF);
    __NOP();
}

```

Рисунок 50. Каркаса кода обработчика прерывания таймера TIM2.

Задание на самостоятельную работу 2.

Вычислить период ШИМ на выходе контроллера у соседней группы учащихся, соединив проводом отладочные платы вашей и соседней групп.

*Замечание. Возможно, понадобится повысить точность таймера за счет изменения предделителя в меньшую сторону.

7. Работа с аналоговым сигналом. Модуль ADC.

АЦП — аналогово-цифровой преобразователь (рисунок 51). Наряду с таймером также является одним из важнейших периферийных модулей микроконтроллера. Ранее рассматривался таймер в режиме модуляции сигнала, который при достаточно высоких частотах создавал квазинепрервыный сигнал из дискретного. Модуль АЦП решает ровно противоположную задачу, заключающуюся в преобразовании непрерывного сигнала в дискретный.

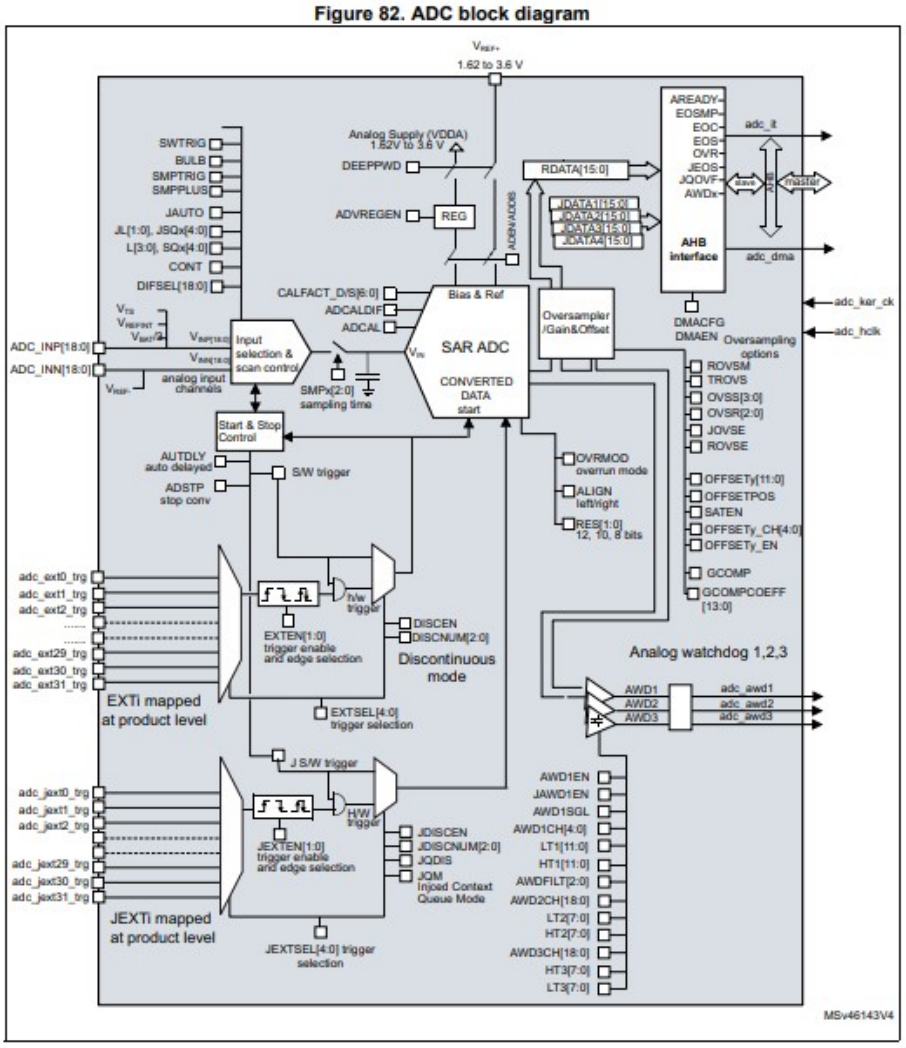


Рисунок. 51. Структурная схема АЦП в STM32G474RE

Настройка АЦП в STM32G474RE крайне гибкая. Основные особенности:

- Разрядность 12 бит
- Самокалибровка
- Выравнивание результата по выбранному краю
- Время преобразования — 12.5 цикла, время захвата — программируемое.

Помимо этих особенностей в STM32G474RE также существуют 2 канала преобразований:

Режим регулярных преобразований.

В этом режиме задаётся группа от 1 до 16 каналов, обработка которых производится последовательно, а результат помещается в один 12-разрядный регистр. Нельзя выбрать только один канал, необходимо задать именно группу каналов, а в ней выбрать их количество. Преобразования могут быть циклическими и периодическими, запускаться по таймеру или внешнему прерыванию.

Режим инжектированных преобразований.

В последовательность инжектированных преобразований может входить до 4 каналов, они имеют собственные регистры для сохранения результата. Сразу после запуска инжектированных преобразований, приостанавливается оцифровка регулярных, затем по окончании инжектированных, возобновляется оцифровка регулярных. В регистры компенсации можно записать число, которое будет автоматически вычитаться из результата преобразования, если в результате вычитания получается отрицательное число, регистр инжектированного преобразования дополняется знаком.

Дальнейшая работа будет организована на регулярных каналах.

Задание 8. Снять сигнал с напряжения питания.

В данном задании будет разобрана настройка АЦП на контроллере STM32G474RE. АЦП, как и любой другой модуль, работает на определенной частоте. Однако этот модуль имеет более универсальные настройки тактовой частоты, в связи с чем действия по настройке немного отличны от тех, что были ранее при работе с периферией (рисунок 52). Также следует обратить внимание, что каналы АЦП уже заранее соответствуют определенным выводам контроллера (приложение 2, table 12).

Как рассматривалось ранее (рисунок 9), вывод может работать как цифровой ввод/вывод, как альтернативная функция или как аналоговый вход. Все выводы после сброса настраиваются по умолчанию в аналоговом режиме, если нет других прописанных настроек.

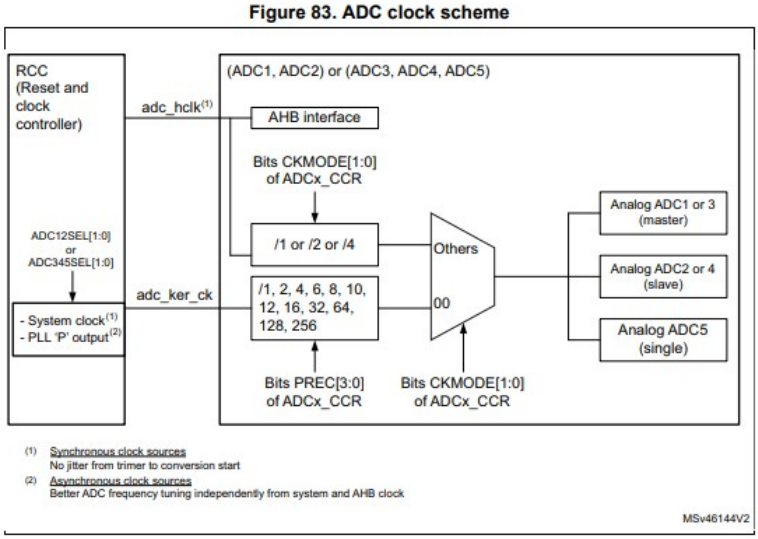


Рисунок. 52. Схема тактирования АЦП в STM32G474RE

В этом задании будет использоваться вывод PA0. У этого вывода есть аналоговый режим ADC12 и он соответствует каналу ADC12_IN1.Регистры, необходимые для настройки АЦП:

- ADC_CR — регистр управления АЦП. Задает свойства АЦП, производит его включение (рисунок 53).
- ADC_CFGR — регистр конфигурации АЦП. Определяет настройки АЦП. Очень важной настройкой здесь является отключение режима Overrun. При включенном режиме может произойти остановка записи в регистр данных обработанных каналов. Такое поведение будет продолжаться, пока регистр данных не освободится, и флаг overrun не будет обнулен программно (рисунок 54).
- ADC_ISR — регистр прерывания и статуса. Содержит флаги событий АЦП (рисунок 55).
- ADC_SMPR1 — регистр настройки времени выборки. Устанавливает время выборки для каждого из набора каналов (рисунок 56).
- ADC_SQR1 — регистр настройки последовательности преобразований в регулярном канале (рисунок 57).
- ADCx_CCR — общий регистр управления АЦП. Настраивает взаимодействия между АЦП и внешними сигналами (рисунок 58). В частности, задает предделитель тактовой частоты CKMODE (рисунок 52).
- RCC_CCIPR — регистр настройки независимого тактирования периферии. Определяет независимый источник тактирования периферии (рисунок 59).
- ADC_DR — регистр данных, куда сохраняется преобразованное значение.

21.6.3 ADC control register (ADC_CR)

Address offset: 0x08

Reset value: 0x2000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADCAL	ADCALDIF	DEEPPWD	ADVREGEN	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
rs	rw	rw	rw												

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	JADSTP	ADSTP	JADSTART	ADSTART	ADDIS	ADEN
										rs	rs	rs	rs	rs	rs

Рисунок 53. Регистр ADC_CR модуля ADC.

21.6.4 ADC configuration register (ADC_CFGR)

Address offset: 0x0C

Reset value: 0x8000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
JQDIS	AWD1CH[4:0]					JAUTO	JAWD1EN	AWD1EN	AWD1SGL	JQM	JDISCEN	DISCNUM[2:0]			DISCEN
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
ALIGN	AUTDLY	CONT	OVRMOD	EXTEN[1:0]		EXTSEL4	EXTSEL3	EXTSEL2	EXTSEL1	EXTSEL0	RES[1:0]		Res.	DMACFG	DMAEN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw

Рисунок 54. Регистр ADC_CFGR модуля ADC.

21.6.1 ADC interrupt and status register (ADC_ISR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	JQOVF	AWD3	AWD2	AWD1	JEOS	JEOC	OVR	EOS	EOC	EOSMP	ADRDY
					rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Рисунок 55. Регистр ADC_ISR модуля ADC.

21.6.6 ADC sample time register 1 (ADC_SMPR1)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SMPPLUS	Res.	SMP9[2:0]			SMP8[2:0]			SMP7[2:0]			SMP6[2:0]			SMP5[2:1]	
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
SMP5[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

Рисунок 56. Регистр ADC_SMPR1 модуля ADC.

21.6.11 ADC regular sequence register 1 (ADC_SQR1)

Address offset: 0x30

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	SQ4[4:0]					Res.	SQ3[4:0]					Res.	SQ2[4]
			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
SQ2[3:0]				Res.	SQ1[4:0]					Res.	Res.	L[3:0]			
rw	rw	rw	rw		rw	rw	rw	rw	rw			rw	rw	rw	rw

Bits 31:29 Reserved, must be kept at reset value.

Bits 28:24 **SQ4[4:0]:** 4th conversion in regular sequence

These bits are written by software with the channel number (0 to 18) assigned as the 4th in the regular conversion sequence.

Note: The software is allowed to write these bits only when ADSTART=0 (which ensures that no regular conversion is ongoing).

Bit 23 Reserved, must be kept at reset value.

Рисунок 57. Регистр ADC_SQR1 модуля ADC.

21.7.2 ADCx common control register (ADCx_CCR) (x=12 or 345)

Address offset: 0x08 (this offset address is relative to the master ADC base address + 0x300)

Reset value: 0x0000 0000

One interface controls ADC1 and ADC2, while the other interface controls ADC3, ADC4 and ADC5.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	VBATSEL	VSENSESEL	VREFEN	PRESC[3:0]				CKMODE[1:0]	
							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
MDMA[1:0]		DMACFG	Res.	DELAY[3:0]				Res.	Res.	Res.	DUAL[4:0]				
rw	rw	rw		rw	rw	rw	rw				rw	rw	rw	rw	rw

Рисунок 58. Регистр ADC_CCR модуля ADC.

7.4.26 Peripherals independent clock configuration register (RCC_CCIPR)

Address: 0x88

Reset value: 0x0000 0000

Access: no wait states, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADC345SEL[1:0]		ADC12SEL[1:0]		CLK48SEL[1:0]		FDCANSEL[1:0]		I2S23SEL[1:0]		SAI1SEL[1:0]		LPTIM1SEL[1:0		I2C3SEL[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
I2C2SEL[1:0]		I2C1SEL[1:0]		LPUART1SEL[1:0]		UART5SEL[1:0]		UART4SEL[1:0]		USART3SEL[1:0]		USART2SEL[1:0]		USART1SEL[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 59. Регистр RCC_CCIPR модуля RCC.

Порядок действий.

1. Включить тактирование АЦП ADC12EN на шине AHB2ENR.
2. Включить независимый источник тактирования в качестве системного тактирования в регистре RCC_CCIPR для ADC12.
3. Выбрать источник тактирования ADC_HCLK путем выставления в регистре ADC12_CCR нулевого значения.
4. Выключить режим Deep_power mode в регистре ADC_CR.
5. Включить внутренний регулятор ADVREGEN в регистре ADC_CR.
6. Выждать примерно 50 тактов микроконтроллера после включения регулятора.
7. Настроить калибровку для несимметричного канала.
8. Включить калибровку АЦП и дождаться её завершения.
9. Включить АЦП и дождаться стабилизации.
10. Очистить флаг готовности АЦП записью значения бита 1 в ADC_ISR_ADRDY.
11. Настроить длину преобразований на одно преобразование в регистре ADC_SQR1.
12. Настроить преобразование только первого канала ADC_SQR1.
13. Настроить преобразование в непостоянном режиме в регистре ADC_CFGR.

14. Выключить режим overrun в регистре ADC_CFGR.
15. Установить время выборки в 24.5 цикла АЦП в регистре ADC_SMRP.
16. Настроить таймер TIM4 на генерацию ШИМ с частотой 10 Гц.
17. Настроить запуск преобразования по фронту внешнего сигнала в регистре ADC_CFGR.
18. Выбрать источником сигнала запуска преобразования таймер TIM4 в регистре ADC_CFGR.

В результате должны получится 3 функции: инициализация АЦП (рисунок 60), инициализация таймера TIM4 (рисунок 61) и запуск преобразования (рисунок 62). Также приведен код использования АЦП в основном цикле (рисунок 63).

```
void ADC_Init(void)
{
    // Включаем тактирование АЦП
    RCC->AHB2ENR |= RCC_AHB2ENR_ADC12EN;
        RCC->CCIPR |= RCC_CCIPR_ADC12SEL_1;
    // Выбираем источник тактирования (adc_hclk)
    ADC12_COMMON->CCR |= ADC_CCR_CKMODE_0;
    // Выбираем тип входа (single ended)
    /* --- Калибровка АЦП перед работой --- */
    // Отключаем режим Deep-power-mode
    CLEAR_BIT(ADC1->CR, ADC_CR_DEEPPWD | ADC_CR_BITS_PROPERTY_RS);
    // Включаем внутренний регулятор
    SET_BIT(ADC1->CR, ADC_CR_ADVREGEN);
    // Ждем пока стабилизируется регултор
              __IO uint32_t wait_loop_index = ((2UL) * ((24000000UL / (100000UL * 2UL)) + 1UL));
    while (wait_loop_index != 0UL)
    {
      wait_loop_index--;
    }
    while(!READ_BIT(ADC1->CR, ADC_CR_ADVREGEN));

    // Указываем что калибровка для каналов типа single-ednded
    CLEAR_BIT(ADC1->CR, ADC_CR_ADCALDIF);
    // Запускаем калибровку АЦП
    SET_BIT(ADC1->CR, ADC_CR_ADCAL);
    // Ждем окончания калибровки (бит ADCAL автоматически сбрасывается)
    while(READ_BIT(ADC1->CR, ADC_CR_ADCAL));
    /* ------------------------------------ */
    // Включаем АЦП
    SET_BIT(ADC1->CR, ADC_CR_ADEN);
    // Ждем когда АЦП стабилизируется
    while(!READ_BIT(ADC1->CR, ADC_ISR_ADRDY));
    // Очищаем флаг
    SET_BIT(ADC1->ISR, ADC_ISR_ADRDY);
    // Настраиваем последовательность проебразования каналов
    // Сначала настраиваем длину преобразований,
    // т.е. количество преобразований в последовательности
    // ADC_SQR1_L[3:0] = 0 -> 1 преобразование
    CLEAR_BIT(ADC1->SQR1, ADC_SQR1_L);
    // Преобразовываем только 1-ый канал
    SET_BIT(ADC1->SQR1, ADC_SQR1_SQ1_0);
    // Преобразование в режиме discontinous
    CLEAR_BIT(ADC1->CFGR, ADC_CFGR_CONT);
    // Отключения режима overrun
    CLEAR_BIT(ADC1->CFGR, ADC_CFGR_OVRMOD);
    // Устанавливаем время выборки (sampling time)  24.5 ADC clocks
    ADC1->SMPR1 |= ADC_SMPR1_SMP1_0 | ADC_SMPR1_SMP1_1;
    // Запуск преобразования по фронту внешнему сигнала
    ADC1->CFGR |= ADC_CFGR_EXTEN_0;
    // Выбораем источник сигнала (TIM4_CC4)
    ADC1->CFGR |= ADC_CFGR_EXTSEL_0 | ADC_CFGR_EXTSEL_2;
}
```

Рисунок 60. Исходный код инициализации АЦП.

```
void TIM4_Init(void)
{
    // Включаем тактирование таймера
    RCC->APB1ENR1 |= RCC_APB1ENR1_TIM4EN;
    // Настраиваем предделитель
    TIM4->PSC = 2399;
    // Настраиваем регистр периода (10 Hz)
    TIM4->ARR = 50000;
    // Настравиаем работу 4-го канала как выход для генерации ШИМ
    CLEAR_BIT(TIM4->CCMR2, TIM_CCMR2_CC4S);
    // Настраиваем режим в PWM mode 1
    TIM4->CCMR2 |= TIM_CCMR2_OC4M_1 | TIM_CCMR2_OC4M_2;
    // Включаем сигнал
    TIM4->CCER |= TIM_CCER_CC4E;

}
```

Рисунок 61. Исходный код инициализации таймера TIM4

```
void ADC_Start(void)
{
    ADC1->CR |= ADC_CR_ADSTART;
}
```

Рисунок 62. Исходный код запуска преобразования АЦП.

```
    while(1)
    {
        while(!READ_BIT(ADC1->ISR, ADC_ISR_EOC));
        adc_val = ADC1->DR;
        if(READ_BIT(ADC1->ISR, ADC_ISR_EOC))
        {
            adc_val = ADC1->DR;
            printf("ADC value:%d\n", adc_val);
            ADC_Start();
        }

    }
}
```

Рисунок 63. Исходный код основного цикла с использованием АЦП в режиме discontinuous.

Задание на самостоятельную работу.

Как видно из разобранного примера, такой подход является не совсем оптимальным с точки зрения ресурсов контроллера. Это можно улучшить путем использования режима continuous для АЦП, чтобы не включать постоянно АЦП заново. Также ресурсы ядра можно не тратить на ожидание флага окончания преобразования, а настроить на АЦП прерывания и снимать значение в обработчике.

Настроить режим continuous у АЦП и завести прерывание на событие завершения преобразования. Вместо напряжения питания входным сигналом теперь является ранее созданный ШИМ на таймере TIM2.

8. Продвинутый способ работы с информацией. Модуль DMA.

Модуль DMA (direct memory access) представляет аппаратную возможность передачи информации из одного места в другое без участия ядра и программного кода (рисунок 64). Ранее уже рассматривались различные аппаратные возможности для асинхронной работы с помощью прерываний, но при этом не говорилось, как периферии могут общаться друг с другом без участия ядра. Сейчас же будет рассмотрена эта возможность.

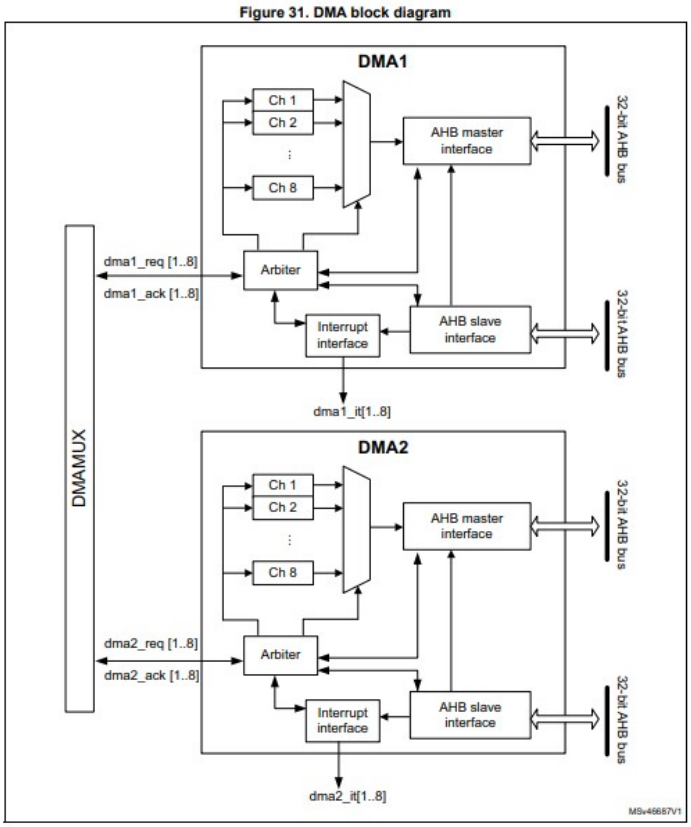


Рисунок 64. Блок-схема DMA.

Наряду с DMA также существует DMAMUX (dma multiplexer), который способствует перенаправлению запросов к DMA (рисунок 65). Он является умной шиной, связывающий все модули DMA между собой.

Figure 32. DMAMUX block diagram

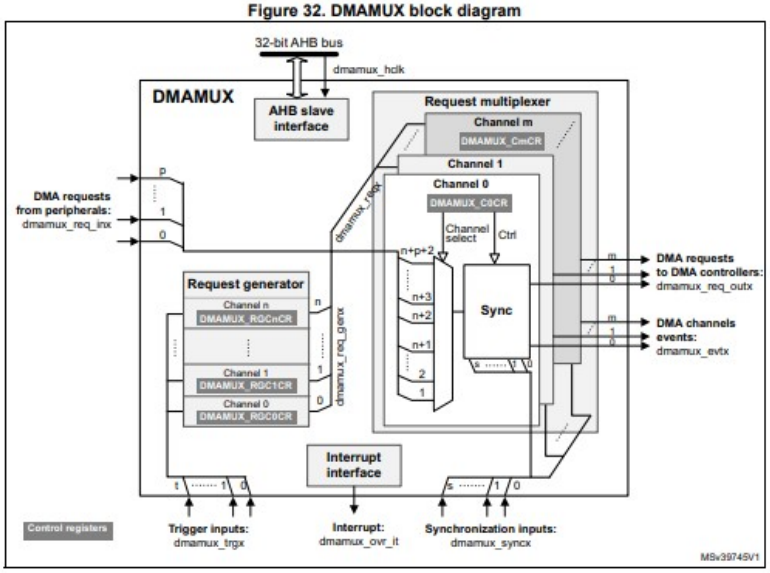


Рисунок 65. Блок-схема DMAMUX.

Задание 9. Настроить DMA для модуля АЦП.

В данном задании рассматривается задача автоматической записи через DMA значений из АЦП в переменную до тех пор, пока их не станет 10. Для этого понадобятся следующие регистры:

- DMAMUX_CxCR — регистр управления и настройки модуля DMAMUX (рисунок 66).
- DMA_ISR — регистр статуса прерывания (рисунок 67).
- DMA_IFCR — регистр очистки флага прерывания (рисунок 68).
- DMA_CCRx — регистр настройки канала (рисунок 69).
- DMA_CNDTRx — регистр настройки количества данных для отправки канала (рисунок 70).
- DMA_CPARx — регистр задания адреса периферии (рисунок 71).
- DMA_CMARx — регистр задания адреса памяти (рисунок 72).

13.6.1 DMAMUX request line multiplexer channel x configuration register (DMAMUX_CxCR)

Address offset: 0x000 + 0x04 * x (x = 0 to 15)

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	SYNC_ID[4:0]					NBREQ[4:0]					SPOL[1:0]		SE
			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
Res.	Res.	Res.	Res.	Res.	Res.	EGE	SOIE	Res.	DMAREQ_ID[6:0]						
						rw	rw		rw	rw	rw	rw	rw	rw	rw

Рисунок 66. Регистр DMAMUX_CxCR модуля DMAMUX.

12.6.1 DMA interrupt status register (DMA_ISR)

Address offset: 0x00

Reset value: 0x0000 0000

Every status bit is cleared by hardware when the software sets the corresponding clear bit or the corresponding global clear bit CGIFx, in the DMA_IFCR register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TEIF8	HTIF8	TCIF8	GIF8	TEIF7	HTIF7	TCIF7	GIF7	TEIF6	HTIF6	TCIF6	GIF6	TEIF5	HTIF5	TCIF5	GIF5
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TEIF4	HTIF4	TCIF4	GIF4	TEIF3	HTIF3	TCIF3	GIF3	TEIF2	HTIF2	TCIF2	GIF2	TEIF1	HTIF1	TCIF1	GIF1
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Рисунок 67. Регистр DMA_ISR модуля DMA.

12.6.2 DMA interrupt flag clear register (DMA_IFCR)

Address offset: 0x04

Reset value: 0x0000 0000

Setting the global clear bit CGIFx of the channel x in this DMA_IFCR register, causes the DMA hardware to clear the corresponding GIFx bit and any individual flag among TEIFx, HTIFx, TCIFx, in the DMA_ISR register.

Setting any individual clear bit among CTEIFx, CHTIFx, CTCIFx in this DMA_IFCR register, causes the DMA hardware to clear the corresponding individual flag and the global flag GIFx in the DMA_ISR register, provided that none of the two other individual flags is set.

Writing 0 into any flag clear bit has no effect.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CTEIF8	CHTIF8	CTCIF8	CGIF8	CTEIF7	CHTIF7	CTCIF7	CGIF7	CTEIF6	CHTIF6	CTCIF6	CGIF6	CTEIF5	CHTIF5	CTCIF5	CGIF5
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
CTEIF4	CHTIF4	CTCIF4	CGIF4	CTEIF3	CHTIF3	CTCIF3	CGIF3	CTEIF2	CHTIF2	CTCIF2	CGIF2	CTEIF1	CHTIF1	CTCIF1	CGIF1
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Рисунок 68. Регистр DMA_IFCR модуля DMA.

12.6.3 DMA channel x configuration register (DMA_CCRx)

Address offset: 0x08 + 0x14 * (x - 1), (x = 1 to 8)

Reset value: 0x0000 0000

The register fields/bits MEM2MEM, PL[1:0], MSIZE[1:0], PSIZE[1:0], MINC, PINC, and DIR are read-only when EN = 1.

The states of MEM2MEM and CIRC bits must not be both high at the same time.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	MEM2MEM	PL[1:0]		MSIZE[1:0]		PSIZE[1:0]		MINC	PINC	CIRC	DIR	TEIE	HTIE	TCIE	EN
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 69. Регистр DMA_CCRx модуля DMA.

12.6.4 DMA channel x number of data to transfer register (DMA_CNDTRx)

Address offset: 0x0C + 0x14 * (x - 1), (x = 1 to 8)

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NDT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 70. Регистр DMA_CNDTRx модуля DMA.

12.6.5 DMA channel x peripheral address register (DMA_CPARx)

Address offset: 0x10 + 0x14 * (x - 1), (x = 1 to 8)

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PA[31:16]															
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
PA[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 71. Регистр DMA_CPARx модуля DMA.

12.6.6 DMA channel x memory address register (DMA_CMARx)

Address offset: 0x14 + 0x14 * (x - 1), (x = 1 to 8)

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MA[31:16]															
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
MA[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 72. Регистр DMA_CPARx модуля DMA.

Порядок действий.

1. Включить тактирование DMA1 и DMAMUX1.
2. Согласно таблице 91 приложения 1 записать в регистр DMAMUX_CxCR id dma-запроса от модуля ADC1 5(101) в 7ми битное поле DMAREQ_ID.
3. Разрешить DMA в модуле АЦП в регистре ADC_CFGR.
4. Указать, откуда забирать данные, в регистре DMA_CPAR.
5. Указать, куда их перемещать, в регистре DMA_CMAR.
6. Указать количество необходимых перемещений в регистре DMA_CNDTR.
7. Настроить круговой режим в регистре DMA_CCRx в поле CIRC.
8. Включить инкремент адреса памяти в регистре DMA_CCRx в поле MINC.
9. Выставить размер данных 16 бит в регистре DMA_CCRx в поле PSIZE и MSIZE.
10. Выставить средний приоритет в регистре DMA_CCRx в поле PL.

11. Включить прерывания по окончанию перемещения в регистре DMA_CCRx в поле TCIE.
12. Включить DMA в регистре DMA_CCR.

Полученный исходный код выглядит следующим образом (рисунки 73 и 74).

```
void ADC_DMA_Init(void)
{
    // Включаем тактирование DMA
    RCC->AHB1ENR |= RCC_AHB1ENR_DMAMUX1EN | RCC_AHB1ENR_DMA1EN;
    // Настраиваем обработку сигналов в DMAMUX
    // Сигнал от ADC1 отправляем в 1-ый канал DMA1 (input id -> 5)
    DMAMUX1_Channel0->CR |= DMAMUX_CxCR_DMAREQ_ID_0 | DMAMUX_CxCR_DMAREQ_ID_2;
    // Разрешаем использование DMA в модуле АЦП
    ADC1->CFGR |= ADC_CFGR_DMACFG | ADC_CFGR_DMAEN;
    // Настраиваем работу канала DMA1 (выбираем 1 канал)
    // Задаем адрес переменной (памяти) куда записываем данные
    DMA1_Channel1->CPAR = (uint32_t)&(ADC1->DR);
    // Задаем адрес периферии, откуда забираем данные
    DMA1_Channel1->CMAR = (uint32_t)(adc_values_avr);
    // Задаем количество трансферов, которые необходимо произвести
    DMA1_Channel1->CNDTR = ADC_SAMPLES_NUM;
    // Настраиваем канал
    DMA1_Channel1->CCR = DMA_CCR_CIRC | DMA_CCR_MINC | DMA_CCR_PSIZE_0 | DMA_CCR_MSIZE_0
          | DMA_CCR_PL_0 | DMA_CCR_TCIE;
    // Разрешаем прерывания от 1-го канала DMA1
    NVIC_EnableIRQ(DMA1_Channel1_IRQn);

    // Включаем 1-ый канал DMA1
    DMA1_Channel1->CCR |= DMA_CCR_EN;

}
```

Рисунок 73. Исходный код реализации настройки DMA для ADC1.

```
#ifndef _ADC_H_
#define _ADC_H_
#include "stm32g4xx.h"

#define ADC_SAMPLES_NUM 10

extern volatile uint16_t adc_values_avr[ADC_SAMPLES_NUM];
extern volatile float adc_average;


void ADC_Init(void);
void ADC_Start(void);
void ADC_DMA_Init(void);
#endif /* _ADC_H_ */
```

Рисунок 74. Исходный код заголовочного файла настройки DMA для ADC1.

Задание на самостоятельную работу.

Реализовать обработчик прерывания для расчета средней значения ШИМ с таймера TIM2 `void DMA1_Channel1_IRQHandler (void).`

*Замечание. Сброс флага прерывания происходит путем записи бита в поле CTCIF1 в регистре DMA_ IFCR.

9. Интерфейсы связи с внешним миром. Модуль USART.

USART (universal synchronous/asynchronous receiver transmitter) — один из наиболее распространенных интерфейсов передачи данных (рисунок 75). Представляет из себя периферийный модуль, с помощью которого два различных устройства при одинаковой скорости передачи/приема могут «общаться» друг с другом. Эта скорость именуется Baud Rate и измеряется в количестве бит в секунду. Также этот интерфейс не разделяет связанные им устройства на приемник и передатчик: каждое из устройств может быть как приемником, так и передатчиком одновременно. Такая схема взаимодействия называется дуплексным режимом передачи (по двум каналам). Каналы в USART называются Tx (transmitt) - передающий, и Rx (recieve) - принимающий. USART может работать в двух режимах — в синхронном и асинхронном. В первом случае передача данных осуществляется в обе стороны одновременно, при наличии фронта дополнительной линии тактирования, соединяющей два устройства. Во втором случае передача ведется полностью независимо от передачи данных в другом канале, линия тактирования между устройствами в этом случае отсутствует, и интерфейс называют просто UART.

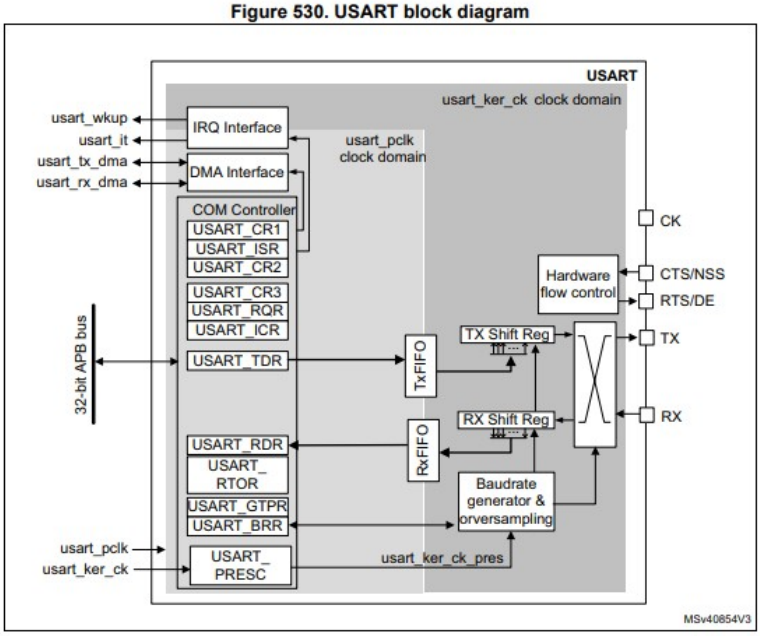


Рисунок 75. Блок-схема модуля USART.

Передача данных происходит последовательно по пакетам или кадрам (рисунок 76). Пакет содержит в себе стартовый бит, слово и стоп бит. Размер слова задается программно.

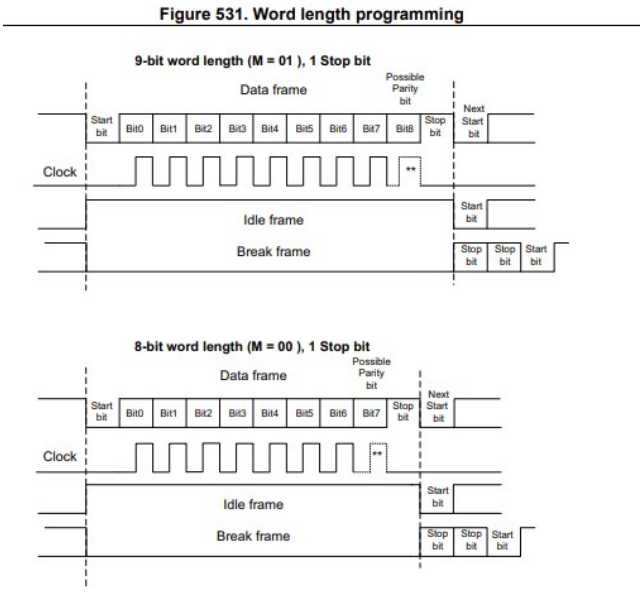


Рисунок 76. Структура пакета передачи модуля USART.

По схожему с USART принципу работы работают также интерфейсы RS232, RS485 и USB. Контроллер STM32G474RE позволяет преобразовать модуль USART в интерфейсы RS232 и RS485 за счет добавления дополнительных линий. Для преобразования USART в USB необходим внешний преобразователь интерфейсов.

Задание 10. Настроить UART на отправку и прием данных.

В данном задании необходимо настроить интерфейс UART и передать данные. Основной настройкой для UART является скорость работы Baud Rate. Необходимые регистры на настройки UART:

- USART_CR1 — регистр управления 1 модуля USART. Задает основные настройки интерфейса (рисунок 77).
- USART_BRR — регистр настройки скорости модуля USART (рисунок 78). Расчет скорости ведется следующим образом:

  USARTDIV — значение в регистре USART_BRR;

  usart_ker_ckpres — частота работы USART;

  baud_rate = usart_ker_ckpres/USARTDIV — итоговая скорость б/с.

  Как правило, значение выбирают из стандартных значений скорости передачи (таблица 1).
- USART_PRESC — регистр предделителя модуля USART (рисунок 79).
- USART_RDR — регистр получаемых данных модуля USART (рисунок 80).
- USART_TDR — регистр отравляемых данных модуля USART (рисунок 81).

- USART_ISR — регистр прерывания и статуса модуля USART (рисунок 82).

Скорость передачи, бод	Время передачи одного бита, мкс	Время передачи байта, мкс
4800	208	2083
9600	104	1042
19200	52	521
38400	26	260
57600	17	174
115200	8,7	87

Таблица 1. Стандартные скорости передачи.

37.8.2 USART control register 1 [alternate] (USART_CR1)

Address offset: 0x00

Reset value: 0x0000 0000

The same register can be used in FIFO mode enabled (previous section) and FIFO mode disabled (this section).

FIFO mode disabled

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	FIFO EN	M1	EOBIE	RTOIE	DEAT[4:0]					DEDT[4:0]				
		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
OVER8	CMIE	MME	M0	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	UESM	UE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 77. Регистр управления 1 модуля USART.

37.8.5 USART baud rate register (USART_BRR)

This register can only be written when the USART is disabled (UE = 0). It may be automatically updated by hardware in auto baud rate detection mode.

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BRR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **BRR[15:0]**: USART baud rate

BRR[15:4]

BRR[15:4] = USARTDIV[15:4]

BRR[3:0]

When OVER8 = 0, BRR[3:0] = USARTDIV[3:0].

When OVER8 = 1:

BRR[2:0] = USARTDIV[3:0] shifted 1 bit to the right.

BRR[3] must be kept cleared.

Рисунок 80. Регистр настройки скорости передачи модуля USART.

37.8.14 USART prescaler register (USART_PRESC)

This register can only be written when the USART is disabled (UE = 0).

Address offset: 0x2C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PRESCALER[3:0]			
												rw	rw	rw	rw

Рисунок 81. Регистр настройки предделителя тактовой частоты модуля USART.

37.8.12 USART receive data register (USART_RDR)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	RDR[8:0]								
							r	r	r	r	r	r	r	r	r

Рисунок 82. Регистр получаемых данных модуля USART.

37.8.13 USART transmit data register (USART_TDR)

Address offset: 0x28

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	TDR[8:0]								
							rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 83. Регистр отправляемых данных модуля USART.

37.8.10 USART interrupt and status register [alternate] (USART_ISR)

Address offset: 0x1C

Reset value: 0x0000 00C0

The same register can be used in FIFO mode enabled (previous section) and FIFO mode disabled (this section).

FIFO mode disabled

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	TCBGT	Res.	Res.	RE ACK	TE ACK	WUF	RWU	SBKF	CMF	BUSY
						r			r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRF	ABRE	UDR	EOBF	RTOF	CTS	CTSIF	LBDF	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Рисунок 84. Регистр прерывания и статуса модуля USART.

Порядок действий

1. Включить тактирование USART1.
2. Настроить выводы USART1(PA10,PA9) в режим альтернативной функции согласно datasheet(приложение 2, таблица 13).
3. Настроить скорость передачи 115200 с помощью регистра USART_BRR.

4. Включить прием и передачу по UART в регистре USART_CR1.
5. Включить прерывание по приему в регистре USART1_CR1.
6. Разрешить прерывания в контроллере вложенных прерываний.
7. Включить UART в регистре USART_CR1.

В результате получается следующий исходный код(рисунок 85).

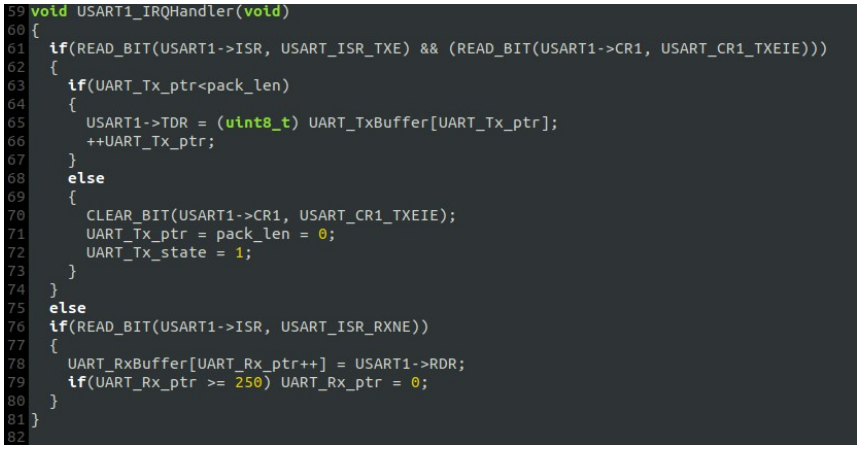
```
void UART_Init(void)
{
    // Включаем тактирование USART1
    RCC->APB2ENR |= RCC_APB2ENR_USART1EN;
    // Настраиваем скорость передачи (115200)
    // 24000000/115200 --> 208 = 0xD0
    USART1->BRR = 0xD0;
    // Настраиваем прием и передачу по UART
    USART1->CR1 |= USART_CR1_RE | USART_CR1_TE;
    // Включаем прерывание по приему
    USART1->CR1 |= USART_CR1_RXNEIE;
    // Разрешаем прерывание от USART1 в NVIC
    NVIC_EnableIRQ(USART1_IRQn);
    // Включаем USART1
    USART1->CR1 |= USART_CR1_UE;
}
```

Рисунок 85. Исходный код настройки UART.

К сожалению, обработчик прерывания UART реагирует на оба события — прием и отправку данных. Поэтому логика обработки должна включать проверку источника прерывания. Для этого алгоритм обработчика будет построен по принципу конечного автомата, где UART будет принимать состояния отправки, приема. События приема и отправки смогут замещать друг друга, если они разные, однако одно и то же событие — нет. Поэтому логика обработчика выглядит следующим образом:

1. Проверяется источник прерывания — прием или отправка.
2. В случае события отправки:
    1. Проверяется закончилась ли отправка пакета.
    2. В случае наличия неотправленных данных помещается следующие 8 бит для отправки и смещается указатель на следующий кадр.
    3. В противном случае — очищается событие отправки, очищается пакет отправки, выставляется флаг возможности приема следующего пакета.
3. В случае события приема:
    1. Считывается принятый кадр.
    2. В случае принятия пакета, смещается указатель на начало пакета.

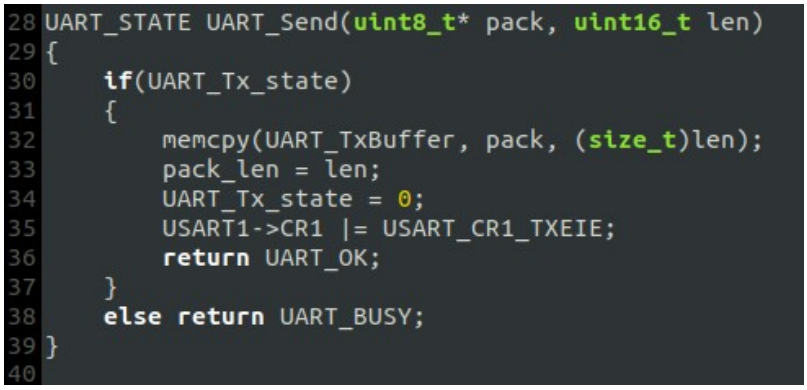
В результате получается следующий исходный код обработки прерывания UART(рисунок 86) и исходный код отправки данных по UART(рисунок 87).



```
void USART1_IRQHandler(void)
{
  if(READ_BIT(USART1->ISR, USART_ISR_TXE) && (READ_BIT(USART1->CR1, USART_CR1_TXEIE)))
  {
    if(UART_Tx_ptr<pack_len)
    {
      USART1->TDR = (uint8_t) UART_TxBuffer[UART_Tx_ptr];
      ++UART_Tx_ptr;
    }
    else
    {
      CLEAR_BIT(USART1->CR1, USART_CR1_TXEIE);
      UART_Tx_ptr = pack_len = 0;
      UART_Tx_state = 1;
    }
  }
  else
  if(READ_BIT(USART1->ISR, USART_ISR_RXNE))
  {
    UART_RxBuffer[UART_Rx_ptr++] = USART1->RDR;
    if(UART_Rx_ptr >= 250) UART_Rx_ptr = 0;
  }
}

```

Рисунок 86. Исходный код обработчика прерывания UART.



```
UART_STATE UART_Send(uint8_t* pack, uint16_t len)
{
    if(UART_Tx_state)
    {
        memcpy(UART_TxBuffer, pack, (size_t)len);
        pack_len = len;
        UART_Tx_state = 0;
        USART1->CR1 |= USART_CR1_TXEIE;
        return UART_OK;
    }
    else return UART_BUSY;
}

```

Рисунок 87. Исходный код отправки данных по UART.

Задание на самостоятельную работу.

Передать данные с АЦП на COM-порт компьютера для построения графиков зависимости значения от времени.

10. Интерфейсы связи с внешним миром. Модуль SPI.

Интерфейс SPI — ещё один крайне распространенный интерфейс взаимодействия(рисунок 88). Этот интерфейс работает в синхронном режиме от общей линии тактирования SCK, которую задает ведущее устройство. По роли во взаимодействии устройства делят на 2 категории — ведущее и ведомое(рисунок 89). Ведущее устройство выбирает среди ведомых то устройство, с котором хочет установить общение; задает частоту передачи между этими устройствами. Передача аналогично USART может осуществляться в дуплексном режиме: для этого в SPI существуют 2 линии MOSI(master out slave in) и MISO(master in slave out). Первая линия служит линией передачи от ведущего устройства ведомому, вторая — наоборот. В отличие от USART линии MOSI,MISO,SCK одного устройства должны полностью совпасть с одноименными линиями другого устройства.

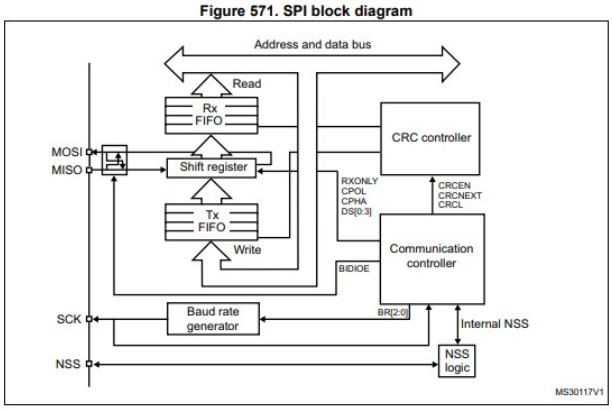
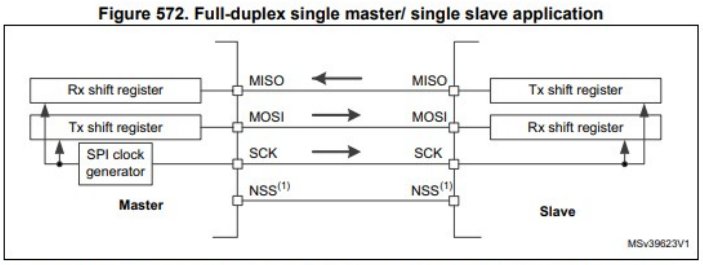


Рисунок 88. Блок-схема SPI.



1. The NSS pins can be used to provide a hardware control flow between master and slave. Optionally, the pins can be left unused by the peripheral. Then the flow has to be handled internally for both master and slave. For more details see *Section 39.5.5: Slave select (NSS) pin management*.

Рисунок 89.Схема взаимодействия между устройствами по интерфейсу SPI.

Задание 11. Настроить SPI в режиме Master.

В этом задании будет разобрана инициализация SPI в режиме Master. Необходимые для этого регистры:

- SPIx_CR1 — регистр управления 1 модуля SPI(рисунок 90).
- SPIx_CR2 — регистр управления 2 модуля SPI(рисунок 91).
- SPIx_SR — регистр состояния модуля SPI(рисунок 92).

- SPIx_DR — регистр данных модуля SPI. Сюда помещаются принимаемые или оправляемые данные(рисунок 93).

39.9.1 SPI control register 1 (SPIx_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDIOE	CRC EN	CRCN EXT	CRCL	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 90. Регистр управления 1 модуля SPI

39.9.2 SPI control register 2 (SPIx_CR2)

Address offset: 0x04

Reset value: 0x0700

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LDMA_TX	LDMA_RX	FRXTH	DS[3:0]				TXEIE	RXNEIE	ERRIE	FRF	NSSP	SSOE	TXDMAEN	RXDMAEN
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 91. Регистр управления 2 модуля SPI.

39.9.3 SPI status register (SPIx_SR)

Address offset: 0x08

Reset value: 0x0002

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	FTLVL[1:0]		FRLVL[1:0]		FRE	BSY	OVR	MODF	CRCERR	UDR	CHSIDE	TXE	RXNE
			r	r	r	r	r	r	r	r	rc_w0	r	r	r	r

Рисунок 92. Регистр состояния модуля SPI.

39.9.4 SPI data register (SPIx_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

Bits 15:0 **DR[15:0]:** Data register
Data received or to be transmitted
The data register serves as an interface between the Rx and Tx FIFOs. When the data register is read, RxFIFO is accessed while the write to data register accesses TxFIFO (See *Section 39.5.9: Data transmission and reception procedures*).

Note: Data is always right-aligned. Unused bits are ignored when writing to the register, and read as zero when the register is read. The Rx threshold setting must always correspond with the read access currently used.

Рисунок 93. Регистр данных модуля SPI.

Порядок действий

- Включить тактирование SPI1.
- Настроить выводы PB3,PB4,PB5 в режим альтернативной функции SPI1_SCK, SPI1_MISO, SPI1_MOSI соответственно и вывод PB2 в режим выхода.
- В регистре SPI_CR1 выставить скорость передачи fpclk/4 в полях BR, настроить полярность и синхронизации в режимы 1 в полях CPHA и CPOL, выставить NSS в режим программного управления с постоянным включением, выставив биты 1 в полях

SSM и SSI, настроить в режим работы Master в поле MSTR, включить SPI, выставив бит 1 в поле SPE.

- В регистре SPI_CR2 выставить длину данных в передаче 16 бит в поле DS, включить прерывания по приему в поле RXNEIE.
- Разрешить прерывания для SPI1 в контроллере вложенных прерываний.

Получается следующий исходный код(рисунок 94).

```
void SPI_Init(void)
{
    // Включаем тактирование SPI1
    RCC->APB2ENR |= RCC_APB2ENR_SPI1EN;

    SPI1->CR1 |= SPI_CR1_BR_0                   // Настраиваем скорость передачи (fpclk/4) -> 6
MBit/s
                | SPI_CR1_CPHA | SPI_CR1_CPOL   // Настраиваем полярность синхронизации
                | SPI_CR1_SSM | SPI_CR1_SSI     // Настраиваем работу пина NSS
                | SPI_CR1_MSTR                  // Насраиваем SPI1 на работу в режиме Master
                | SPI_CR1_SPE;                  // Включаем SPI1

    SPI1->CR2 |= SPI_CR2_DS         // Длина данных в передаче 16 бит
                // | SPI_CR2_TXEIE     // Включаем прерывание по отправке данных
                | SPI_CR2_RXNEIE;   // Включаем прерывание по приему данных
    // Разрешаем прерывания от SPI1 в NVIC
    NVIC_EnableIRQ(SPI1_IRQn);
}
```

Рисунок 94. Исходный код настройки SPI в режиме Master.

Задание на самостоятельную работу.

Реализовать обработчик прерываний `void SPI1_IRQHandler(void)` и функцию отправки значения `SPI_STATE SPI_Send(uint8_t* pack, uint16_t len)` аналогично разобранному UART.

Считать данные с датчика.

11. Реализация замкнутого контура по напряжению для двигателя ДПТ42.

Финальная лабораторная работа является завершением пройденного курса по изучению разработке под микроконтроллеры. В других частях курса были рассмотрены прочие аспекты проектирования систем управления. Предполагается, что каждый обучающийся получил индивидуальное задание на синтез корректирующего устройства системы управления. С помощью изученного инструментария на лабораторных работах и теоретических основ каждому обучающему предлагается реализовать индивидуальное задание с помощью ресурсов этого курса.

Задание на самостоятельную работу

Реализовать корректирующее цифровое устройство в виде разностного уравнения для замкнутого контура управления по скорости согласно индивидуальному заданию.

Источники и литература

1. Приложение 1 — Reference manual stm32g4(https://www.st.com/resource/en/reference_manual/rm0440-stm32g4-series-advanced-armbased-32bit-mcus-stmicroelectronics.pdf).

2. Приложение 2 — Datasheet stm32g474xB,stm32g474xC,stm32g474xE(https://www.st.com/resource/en/datasheet/stm32g474re.pdf)

3. Приложение 3 — User manual stm32g4(https://www.st.com/resource/en/user_manual/um2505-stm32g4-nucleo64-boards-mb1367-stmicroelectronics.pdf)