Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Локтионова Оксана Геннадьевна
Должность: проректор по учебной работе
Дата подписания: 24.04.2024 16:01:09
Уникальный программный ключ:
0b817ca911e6668abb13a5d426d39e5f1c11eabbf73e943df4a4851fda56d089

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра вычислительной техники

УТВЕРЖДАЮ
Проректор по учебной работе
О.Г. Локтионова
«__30__» __09__ 2022 г.

Аппаратно-программное обеспечение инфраструктуры систем искусственного интеллекта

Методические указания к практическим работам
для студентов направления подготовки
09.04.01 очной формы обучения

Курск 2022

УДК 001.89

Составители: В. И. Конченков, В. Н. Скакунов, А.В. Киселев, Е.А. Кулешова

Рецензент
Кандидат технических наук, доцент *Т.Н. Конаныхина*

Аппаратно-программное обеспечение инфраструктуры систем искусственного интеллекта: методические указания к практическим работам для студентов направления подготовки 09.04.01 очной формы обучения / Юго-Зап. гос. ун-т; сост.; В. И. Конченков, В. Н. Скакунов, А.В. Киселев, Е.А. Кулешова. – Курск, 2022. – 44 с. – Библиогр.: с. 44.

Методические указания содержат описание стенда на основе платы EasyMX Pro for STM32, среды разработки проекта и программи рования портов ввода-вывода общего назначения.

Предназначены для студентов направления подготовки 09.04.01 очной формы обучения.

Методические указания соответствуют рабочей программе дисциплины «Аппаратно-программное обеспечение инфраструктуры систем искусственного интеллекта».

Текст печатается в авторской редакции

Подписано в печать . Формат 60∗84 1/16.
Усл. печ. л. 2,85. Уч.-изд. л. 2,58. Тираж 50 экз. Заказ . Бесплатно.
Юго-Западный государственный университет.
305040 Курск, ул. 50 лет Октября, 94.

Практическая работа №1

ИЗУЧЕНИЕ СИСТЕМЫ ТАКТИРОВАНИЯ И РАБОТА С ПОРТАМИ ВВОДА-ВЫВОДА ОБЩЕГО НАЗНАЧЕНИЯ МИКРОКОНТРОЛЛЕРОВ STM32 СЕРИИ F1xxx

Цель: приобретение навыков работы с отладочным стендом EasyMX Pro for STM32 и средой разработки проектов IDE Keil uVision, изучение особенностей работы микроконтроллеров семейства STM32F1xxx при программировании обмена данными через порты ввода-вывода общего назначения, тактировании микроконтроллера.

Общие сведения об отладочном стенде EasyMX Pro for STM32

Практические работы выполняются с использованием отладочного стенда EasyMX Pro for STM32. Внешний вид стенда показан на рис. 1. Основой стенда служит микроконтроллер (МК) STM32F107VCT6 семейства Connectivity line, базирующийся на ядре Cortex-M3. Микроконтроллер STM32F107VC имеет следующие характеристики: 32-разрядная архитектура, максимальная тактовая частота 72 МГц, объем flash-памяти 256 кБт, объем RAM-памяти 64 кБт, 80 линий ввода-вывода общего назначения. Контроллер содержит множество интегрированных периферийных устройств, в том числе, семь 16-разрядных таймеров, 12-разрядный АЦП (16 каналов), два 12-разрядных ЦАП, 5 модулей USART, 3 модуля SPI, 2 модуля I2C, 2 модуля CAN, Ethernet-контроллер, поддержка USB 2.0 (OTG, Host, Device), модуль часов реального времени (RTC), SWD и JTAG интерфейсы. Микросхема выполнена в 100-выводном корпусе LQFP для поверхностного монтажа, напряжение питания (2...3,6) В.

Рис. 1 – Внешний вид стенда EasyMX Pro for STM32.

Отладочная плата EasyMXPro v7 for STM32 предназначена для изучения возможностей и особенностей программирования 32-разрядных ARM микроконтроллеров серии STM32 компании STMicroelectronics. На плате установлен 104 контактный разъем для подключения сменных модулей на основе микроконтроллеров семейств STM32F1xxx и STM32F4xxx с архитектурой Cortex M3 и Cortex M4.

Универсальность платы заключается еще и в том, что она поставляется со встроенным программатором/отладчиком mikroProg, выполненным на базе распространенного отладчика ST-LINK/V2 и поддерживает более 180 типов микроконтроллеров.

Плата содержит широкий набор периферии, включающий стерео MP3 кодек, разъемы для подключения наушников и микрофона, последовательную flash-память и память типа I2C EEPROM, потенциометр для задания входного напряжения на АЦП, слот для подключения MicroSD карты, навигационный джойстик, разъемы для подключения датчиков температуры и сенсорного TFT дисплея, разъемы стандартных интерфейсов – USB, CAN, Ethernet, USB-UART преобразователи. На плате установлен кварцевый резонатор частотой 25 МГц.

Все линии портов ввода-вывода общего назначения микроконтроллера выведены на штыревые разъемы, расположенные по краям платы. Для питания МК используется напряжение 3.3В. Питание внешних устройств возможно от встроенного стабилизатора напряжения 5В.

В комплектацию платы входит TFT-дисплей с разрешением 320x240 пикселей, управляемый контроллером ILI9341.

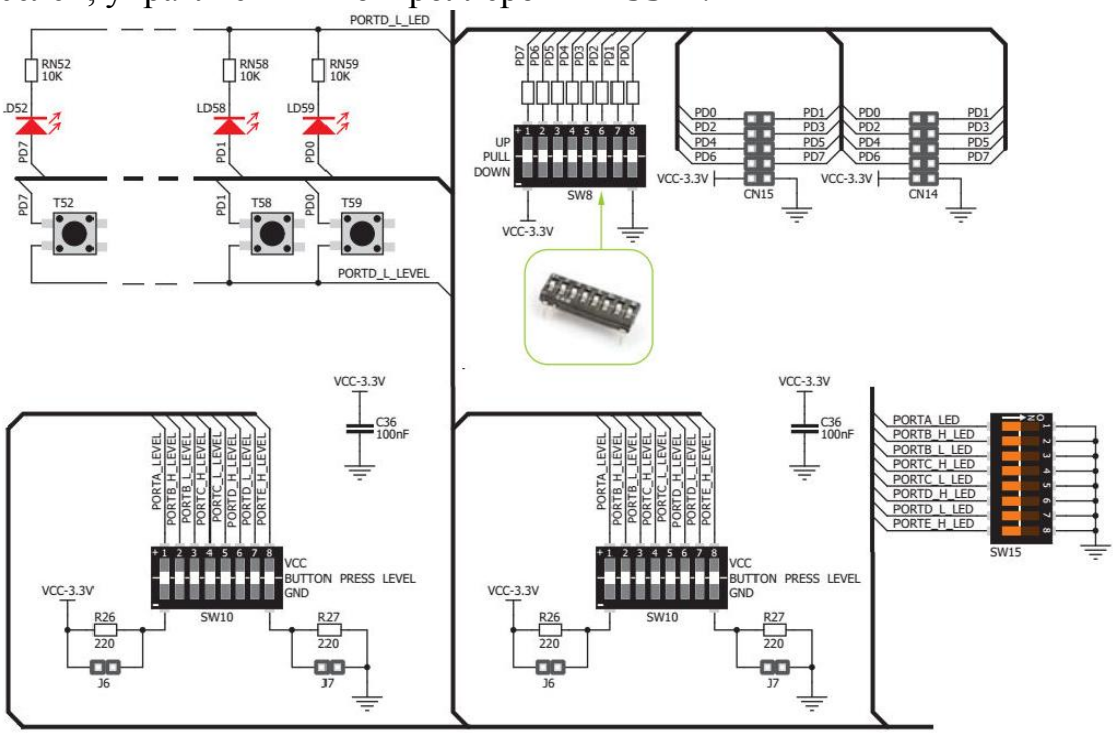

Рис. 2 – Подключение светодиодов, разъемов и кнопок к линиям портов ввода-выводаобщего назначения (на примере порта D)

Порты ввода-вывода общего назначения микроконтроллеров STM32 имеют 16 разрядов. На отладочном стенде линии портов сгруппированы побайтно (PD0-PD7 – PORTD/L, PD7-PD16- PORTD/H) и выведены на штыревые разъемы SN15, SN14 (рис. 2). Следует отметить, что не все линии выведены: во-первых, это связано с ограниченным числом выводов на корпусе самого микроконтроллера, приведенных в описании [1], во-вторых, с особенностями разводки платы. Коммутационные и индикаторные элементы используются и для микроконтроллера серии STM32F1, и для микроконтроллера серии STM32F4. Каждую из линий портов ввода-вывода можно через резистор подтянуть либо к напряжению питания (верхнее положение соответствующего движка переключателя SW8) или к земле (нижнее положение движка переключателя). Среднее положение движка соответствует подключению без подтяжки (рис. 3).

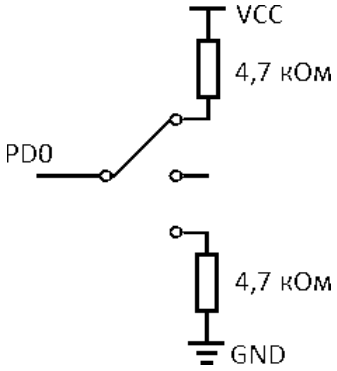

Рис. 3 – Схема подключения индивидуального подтягивающего резисторак линии порта ввода-вывода (одна из линий переключателя SW8).

Аноды светодиодов подключены непосредственно к линиям портов ввода-вывода, а катоды сгруппированы побайтно и через один из переключателей группы SW15 подключаются к «земле». Таким образом, чтобы иметь возможность выводить значения на светодиоды PORTD/L, необходимо перевести соответствующий движок (PORTD_L_LED) группы переключателей SW15 в положение ON. Светодиод загорается, если уровень напряжения на соответствующей линии равен логической единице. Ток через светодиод составляет около 0.2 мА.

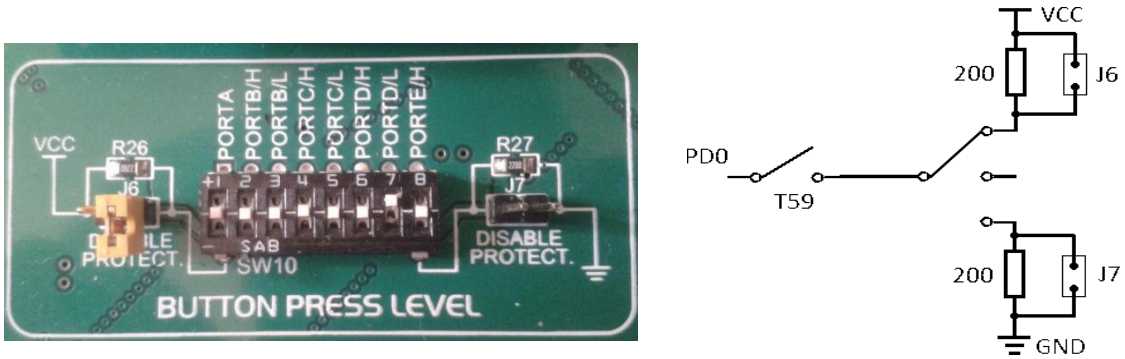

Рис. 4 – Подключение кнопки к линии ввода-вывода.

К линиям портов ввода-вывода подключены кнопки. На рис. 4 показана схема установки уровня на линии ввода-вывода при нажатии кнопки, которая задается переключателем из группы SW10 и джамперами J6 и J7. Наиболее употребителен случай при снятом джампере J6, установленном джампере J7 и установке в нижнее положение движка PORTD/L на SW10 (при нажатой кнопке на линии логический ноль, при отпущенной – логическая единица, рис. 5).

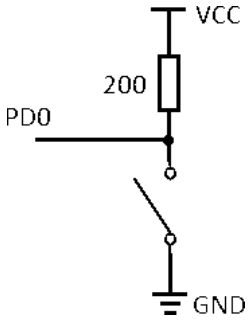

Рис. 5 – Типовое подключение кнопки к линии ввода-вывода микроконтроллера

На отладочном стенде установлен программатор и отладчик mikroProg, основанный на программаторе ST-LINK V2, который поддерживает более 180 устройств с ядром ARM Cortex-M3 и Cortex-M4 (рис. 6). Пять джамперов используются для выбора функций соответствующих линий портов: обычные цифровые разряды портов ввода-вывода (GPIO) или сигнальные линии программатора. Если драйверы на компьютере, к которому подключен стенд, корректно установлены, то в режиме JTAG-отладки будет мигать светодиод LED LINK.

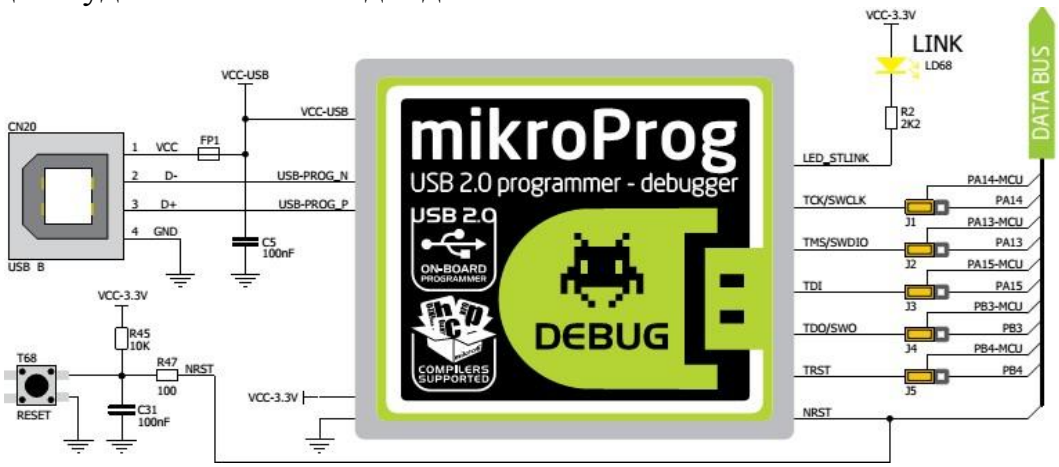

Рис. 6 – Программатор microProg

Система тактирования и сброса микроконтроллеров STM32

Для управления системным тактовым сигналом (SYSCLK) могут использоваться три различных источника:

- HSI oscillator clock (сигнал от внутреннего высокочастотного генератора);
- HSE oscillator clock (сигнал от внешнего высокочастотного резонатора);
- PLL clock (сигнал от блока ФАПЧ – фазовая подстройка частоты).

Кроме того, имеется внутренний низкочастотный RC генератор на 40 кГц (LSI RC), который управляет независимым сторожевым таймером (WatchDog) и, по желанию, часами реального времени RTC, которые используются для автоматического пробуждения из режима Stop/Standby. Дополнительно можно использовать внешний низкочастотный резонатор (LSE crystal) на 32.768 кГц, который, по желанию, управляет тактовым сигналом (RTCCLK) для RTC. Каждый источник тактового сигнала может быть независимо включен или выключен, если он не используется, чтобы оптимизировать потребляемую мощность.

Ядро микроконтроллера тактируется сигналом SYSCLK. От ядра отходит шина AHB, имеющая свою тактовую частоту (ей можно установить некий предделитель относительно SYSCLK). Эта шина подобна шине между процессором и северным мостом компьютера — точно так же она служит для связи ARM ядра и процессора периферии, а также к ней подключается память и контроллер прямого доступа к памяти (DMA).

Несколько делителей частоты позволяют конфигурировать частоту для шины AHB, высокочастотной шины APB (APB2) и низкочастотной шины APB (APB1). Максимальная частота шин AHB и APB2 для микроконтроллеров серии STM32F1 составляет 72 МГц, шины APB1 - 36 МГц. По умолчанию после запуска микроконтроллера вся периферия на шинах APB1 и APB2 отключена в целях экономии энергии.

Тактовый сигнал HSI генерируется с помощью внутреннего RC генератора на 8 МГц, и может использоваться непосредственно в качестве системного тактового сигнала, или, после деления на 2, использоваться в качестве входа для PLL. Этот источник тактирования имеет меньшее время запуска, чем HSE генератор на резонаторе, однако, даже после калибровки, его частота менее точна и стабильна, чем у генератора на внешнем кварцевом или керамическом резонаторе.

Высокочастотный внешний тактовый сигнал (HSE) может быть сгенерирован от двух возможных источников: с помощью внешнего кварцевого/керамического резонатора и с помощью внешнего сигнала. Резонатор и нагрузочные конденсаторы должны быть помещены так близко к выводам генератора, насколько это возможно, чтобы минимизировать искажения и время стабилизации при запуске. Значения нагрузочных конденсато-

ров должны быть откорректированы в соответствии с выбранным резонатором. На отладочном стенде установлен кварцевый резонатор с частотой 25 МГц. В режиме внешнего сигнала (External Clock) должен быть предоставлен внешний источник тактового сигнала с частотой до 50 МГц.

Настройка источника тактовой частоты, частоты системных шин AHB, APB1, APB2, равно как и других периферийных устройств, входящих в состав микроконтроллера, можно осуществлять различными способами. Во-первых, можно непосредственно вписать в соответствующие конфигурационные регистры блока RCC нужные значения и дождаться установки флагов готовности. Этот подход наиболее эффективен, однако труден на начальном этапе изучения микроконтроллеров STM32. Во-вторых, существуют различные графические конфигураторы (наиболее значимый – STM32CubeMX, продвигаемый компанией-производителем), позволяющий получить код инициализации микроконтроллера, выставив нужные флажки на специальной форме. Такой подход удобен и прост, однако при первоначальном изучении программирования и использования микроконтроллеров не дает ясного понимания внутренних процессов, происходящих в разрабатываемой микропроцессорной системе, поэтому будем пользоваться таким конфигуратором как вспомогательным средством. В практических работах используется библиотека SPL (StandardPeripherical Library) [2], распространяемая компанией-производителем, ко- торая обеспечивает разумный уровень абстракции и в то же время позво- ляет вести программирование на уровне установки отдельных битов в конфигурационных регистрах.

Зададимся значением тактовой частоты 64 МГц. На рис. 7 показаны настройки тактирования основных шин микроконтроллера (выставлены в системе STM32CubeMX), реализованные в функции initRCC().

```
void initRCC(void)
{
	RCC_DeInit();
	RCC_HSICmd(DISABLE);
	RCC_HSEConfig(RCC_HSE_ON); // разрешаем тактирование от внешнего генератора
	ErrorStatus HSEStartUpStatus = RCC_WaitForHSEStartUp();
	if (HSEStartUpStatus == SUCCESS){ // если внешний генератор запустился
		RCC_PREDIV2Config(RCC_PREDIV2_Div5); // предделитель 2 - делим HSE на 5
		RCC_PLL2Config(RCC_PLL2Mul_8); // умножаем частоту источника на 8
		RCC_PLL2Cmd(ENABLE);
		// в качестве источника PREDDIV1 сигнал PLL2MUL, предделитель 5
		RCC_PREDIV1Config(RCC_PREDIV1_Source_PLL2, RCC_PREDIV1_Div5);
		// в качестве источника PLL сигнал PREDDIV1, умножение на 8
		RCC_PLLConfig(RCC_PLLSource_PREDIV1, RCC_PLLMul_8);
		RCC_PLLCmd(ENABLE);
		RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK); // источник SYSCLOCK – PLL
		RCC_HCLKConfig(RCC_SYSCLK_Div1); // делитель для HCLK (HCLK = SYSCLK)
		RCC_PCLK2Config(RCC_HCLK_Div1); // делитель для PCLK2 (PCLK2 = HCLK)
		RCC_PCLK1Config(RCC_HCLK_Div4); // делитель для  PCLK1 (PCLK1 = HCLK/4)
		// ожидаем, пока HSE не установится в качестве источника SYSCLOCK
		while (RCC_GetSYSCLKSource() != 0x08) {}
	}
	else //Если HSE не смог запуститься, тактирование настроено некорректно
	{
		while (1) {} //Здесь следует поместить код обработчика этой ошибки
```

```
        }
}
```

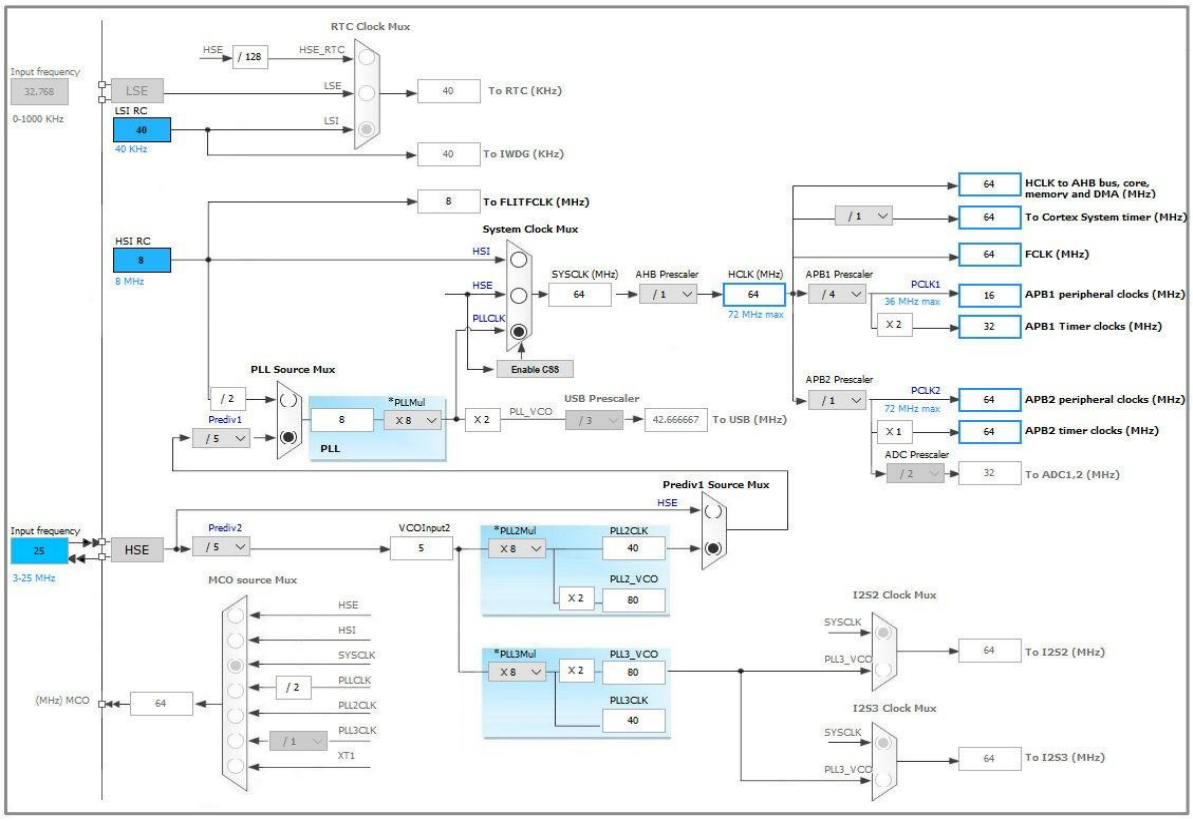

Рис. 7 – Определение частоты тактирования посредством конфигуратора STM32CubeMX

Порты ввода-вывода общего назначения контроллеров STM32

Вся периферия микроконтроллеров STM32 настраивается по стандартной процедуре:

1) включение тактирования соответствующего контроллера — буквально, подача на него тактового сигнала от шины APB1 или APB2;

2) настройки, специфичные для конкретной периферии – необходимо установить нужные биты в управляющие регистры;

3) выбор источников прерываний - каждый периферийный блок может генерировать прерывания по разным событиям;

4) назначение обработчика прерываний;

5) запуск бесконечного цикла – бесконечный цикл необходим, чтобы контроллер циклически проверял значения входных данных, реагировал на прерывания и т.д.

Самым простым и наиболее часто используемым периферийным устройством являются порты ввода-вывода общего назначения. На рис. 8 приведена упрощенная схема одной из линий портов ввода-вывода общего назначения [3].

На входе стоят защитные диоды, недающие снизить потенциал вывода

МК ниже потенциала земли или поднять его выше напряжения питания. Рассмотрим схемы входной (Input Driver) и выходной (Output Driver) цепей (рис. 8) и их режимы работы.

Состояние входа. Сразу после защитных диодов установлены управляемые подтягивающие резисторы, поэтому входной сигнал можно подтянуть к земле или к напряжению питания. Сигнал напрямую идёт в линию «Analog» на вход АЦП или компаратора, если аналоговые блоки подключены к этому выводу. Для работы с цифровыми сигналами установлен триггер Шмитта, служащий для борьбы с дребезгом контактов, и его выход попадает в регистр-защёлку входных данных IDR (input data register), из которого состояние ножки можно считать в программе. Для обеспечения работы встроенной периферии выход триггера Шмитта под именем «Alternate function input» соединен с линией GPIO, выполняющей альтернативные функции. В качестве этой периферии может выступать UART/USART, SPI, USB и т.д. Важно понимать, что все эти связи с периферией постоянно существуют, но программно активизирована и может работать только одна из них.

Состояние выходной цепи. Цифровые данные, записанные в порт выхода, лежат в регистре ODR (output data register). Он доступен как на запись, так и на чтение. Читая из ODR, вы не читаете состояние ножки как входа! Вы читаете то, что сами в него записали. К блоку Output control через мультиплексор подключены выходы не только линии выходного регистра (стандартный выходной сигнал), но и сигналы от других периферийных устройств МК (вывод выполняет альтернативные функции «Alternate function output»).

Режим работы выхода с точки зрения схемотехники настраивается в блоке Output control - можно сделать push-pull выход (линия жёстко притягивается к земле или питанию) или выход с открытым стоком. После драйвера в линию входит аналоговый выход (например, от ЦАП), далее располагаются описанные выше подтягивающие резисторы и защитные диоды. Драйвер цифрового выхода имеет также контроль крутизны, или скорости нарастания напряжения. Можно установить максимальную крутизну, и получить возможность переключать значения на выходной линии с частотой 50 МГц — но так мы получим и сильные электромагнитные помехи. Можно установить минимальную крутизну, с максимальной частотой 2 МГц, при этом значительно уменьшатся радиопомехи.

Атомарные операции. На рис. 8 кроме регистра ODR указан регистр Bit set/reset register (регистр BSR), позволяющий устанавливать и сбрасывать отдельные биты, соответствующие выводам порта ввода-вывода за один такт. Вообще говоря, запись в регистр ODR также производится атомарно. Однако, например, для установки 3 бита порта D нужно выполнить следующую команду:

```
GPIOD->ODR |= (1<<3);
```

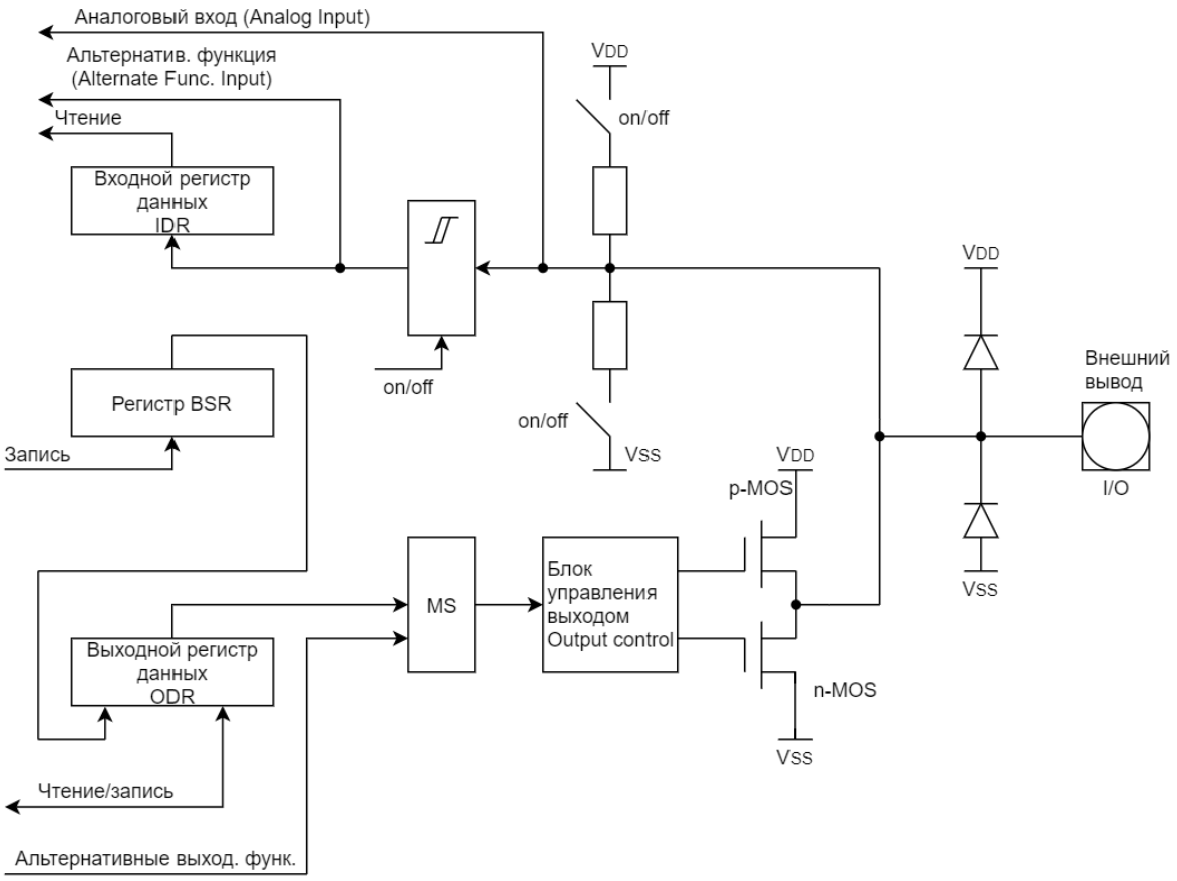

Рисунок 8 – Устройство линии портов ввода-вывода общего назначения

В этой команде нужно вначале считать данные из порта, выполнить побитовую операцию ИЛИ, затем записать данные в порт. Если во время этих действий возникнет прерывание, в порте могут оказаться неправильные значения. Если, к тому же, обработчик прерывания работает с этим портом, может возникнуть трудноуловимая ошибка. Поэтому в микроконтроллерах STM32 введены дополнительные регистры BSRR и BRR, служащие для установки и сброса отдельных битов. Каждый такт шины APB2 читаются регистры BSRR и BRR, и сразу же их содержимое записываются в регистр ODR, а сами эти регистры очищаются. Таким образом, если нужно установить 3 и 5 биты в порте, нужно записать BSRR слово 10100.

Блокирование конфигурации. При желании, можно заблокировать конфигурацию любого ***вывода*** от дальнейших изменений - любая попытка записи в регистр конфигурации окончится неуспехом. Это подойдёт для ответственных применений, где случайное переключение, к примеру, выхода из режима open drain в push-pull, приведет к короткому замыканию. Для включения блокирования предназначен регистр LCKR, только он снабжён защитой от случайной непреднамеренной записи — чтобы изменения вступили в силу, нужно подать специальную последовательность в бит LCKK.

Управляющие регистры

Всё управление контроллером GPIO сосредоточено в 32-битных регистрах GPIOx_RRR, где x — номер порта, а RRR — название регистра.

1) Конфигурационные регистры GPIOx_CRL и GPIOx_CRH

Формат регистра GPIOx_CRL

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

Формат регистра GPIOx_CRH

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

Младший конфигурационный регистр GPIOx_CRL настраивает ножки с номерами 0..7, старший конфигурационный регистр GPIOx_CRH настраивает ножки с номерами 8..15. У каждой ножки два параметра, MODE и CNF. Параметр MODE отвечает за режим вход/выход и скорость нарастания сигнала:
00 - вход (режим по умолчанию);
01 - выход со скоростью 10 МГц;
10 - выход со скоростью 2 МГц;
11 - выход со скоростью 50 МГц.
Параметр CNF отвечает за конфигурацию пина. В режиме **входа** (MODE=00):
00 — аналоговый режим;
01 — «плавающий вход» - не используются внутренние подтягивающие резисторы (по умолчанию);
10 — вход с подтяжкой к земле или питанию (в зависимости от состояния соответствующей линии регистра ODR: 0 – Input pull-down, 1- Input pull-up);
11 — зарезервирован.
В режиме **выхода** (MODE=01, 10 или 11):
00 — выход GPIO Push-pull;
01 — выход GPIO Open drain;
10 — выход альтернативной функции Push-pull;
11 — выход альтернативной функции Open drain;

2) Регистр выходных данных GPIOx_ODR

Формат регистра GPIOx_ODR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Резерв															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0

Каждый бит ODRy содержит в себе состояние соответствующей ножки ввода-вывода. Можно записывать данные, и они появятся на выходе порта, можно читать данные, но будет считано не реальное значение логического уровня, предыдущее записанное значение.

3) Регистр входных данных GPIOx_IDR

Формат регистра GPIOx_IDR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Резерв															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0

Каждый бит регистра IDR содержит в себе состояние соответствующей ножки ввода-вывода. Регистр доступен только для чтения.

4) Регистр атомарной установки/сброса битов выходных данных GPIOx_BSRR

Формат регистра GPIOx_BSRR

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

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

Старшие 16 бит служат для сброса соответствующих пинов в 0 (для этого в соответствующий бит нужно установить логическую единицу). Младшие 16 бит служат для установки битов в 1 (запись логической единицы устанавливает соответствующий бит в единицу). Если установлены и BRx, и BSx, бит BSx имеет приоритет. Регистр доступен только для записи — он сбрасывается в ноль на каждом такте APB2.

5) Регистр атомарного сброса битов выходных данных GPIOx_BRR

Формат регистра GPIOx_BRR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Резерв															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0

Младшие 16 бит служат для сброса соответствующих пинов (для сброса нужно в соответствующий бит установить логическую единицу). Регистр доступен только для записи, он сбрасывается в ноль на каждом такте APB2.

6) Регистр блокирования конфигурации GPIOx_LCKR

Формат регистра GPIOx_LCKR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Резерв															LCKK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0

Каждый бит LCKy блокирует соответствующие биты MODE/CNF регистров CRL/CRH от изменения, так что конфигурацию пина невозможно будет изменить вплоть до перезагрузки. Для активации блокирования необходимо записать блокирующую последовательность в бит LCKK: записываем 1, записываем 0, записываем 1, затем читаем 0, читаем 1. Чтение бита LCKK сообщает текущий статус блокировки: 0 — блокировки нет, 1 — есть.

Приведем кратко сведения о состоянии частей контроллера линии порта ввода вывода при работе в разных режимах в таблице 1.

Таблица 1 – Работа драйвера линии ввода-вывода общего назначения в различных режимах

Режим входа	1) отключается драйвер выхода; 2) входной триггер Шмитта включен; 3) резисторы подтяжек включаются по вашим настройкам, одно из трёх состояний — «вход, подтянутый к земле» (IPD), «вход, подтянутый к питанию» (IPU), «плавающий вход» (IN_FLOATING)
Режим выхода	1) драйвер выхода включен, и действует так: - в режиме «Push-Pull» работает как полумост, включая верхний транзистор в случае «1» и нижний в случае «0», - в режиме «Open drain» включает нижний транзистор в случае «0», а в случае «1» оставляет линию неподключенной (т.е. втретьем состоянии). 2) входной триггер Шмитта включен; 3) отключаются резисторы подтяжек; 4) выходной сигнал семплируется каждый такт шины APB2 и записывается в регистр IDR, и чтение этого регистра сообщает состояние ножки в режиме Open drain. 5) чтение регистра ODR сообщает последнее записанное состоя- ние в режиме Push-Pull.

Таблица 1

Режим альтернативной функции (не-GPIO-периферия)	1) выходной драйвер — в режиме Push-Pull (к примеру, так работает ножка TX модуля USART) или Open drain, в зависимости от требований контроллера; 2) выходной драйвер управляется сигналами периферии, а не регистром ODR; 3) входной триггер Шмитта включен; 4) резисторы подтяжки отключены; 5) выходной сигнал семплируется каждый такт шины APB2 и записывается в регистр IDR, и чтение этого регистра сообщает состояние ножки в режиме Open drain; 6) чтение регистра ODR сообщает последнее записанное состояние в режиме Push-Pull.
Аналоговый режим	1) выходной драйвер выключен; 2) триггер Шмитта полностью отключается, чтобы не влиять на напряжение на входе; 3) резисторы подтяжки отключены; 4) в регистре IDR — постоянно 0; 5) вся внутренняя аналоговая периферия имеет высокий входной импеданс, поэтому и сама ножка по отношению к остальной схеме будет иметь высокий входной импеданс.

Создание проекта в IDE Keil uVision

В главном меню выбираем Project ->Create uVision Project… В открывшемся диалоговом окне выбираем папку, где будут храниться файлы проекта. После этого отрывается окно выбора микроконтроллера (рис. 9). Выбираем STM32F107VC (такой контроллер установлен на отладочном стенде Mikroelektronika EasyMX PRO v7 for STM32).

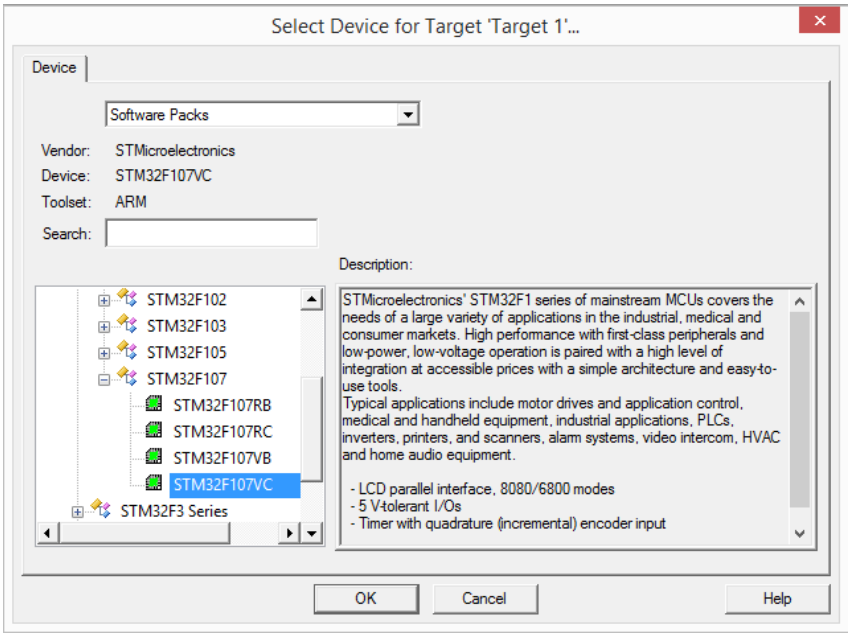

Рис. 9 – Окно выбора целевого микроконтроллера.

Далее необходимо выбрать библиотеки. В разделе CMSIS выбираем CORE, в разделе Device выбираем пункт Startup, в подразделе StdPeriph Drivers выбираем пункты Framework, GPIO, RCC. Если какой-то пункт требуется, но не выбран, внизу окна будет соответствующее сообщение.

После нажатия кнопки ОК открывается окно редактора исходного кода (рис. 10).

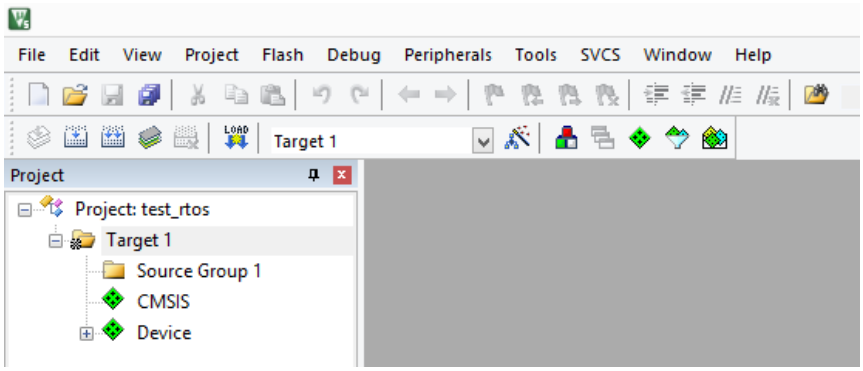

Рис. 10 – Фрагмент главного окна программы Keil uVision v5

В Keil широко применяется использование групп файлов. В частности, одной из единиц является Target (цель). Использование цели позволяет в одном проекте собирать разные прошивки, содержащие некоторые общие файлы исходного кода. Фактически, разные target'ы могут содержать разные настройки проекта (параметры компиляции, тип микроконтроллера, конфигурационные файлы и т.д.). Файловые группы существуют только в рамках проекта Keil и не обязаны совпадать с папками операционной системы.

Напишем простейшую программу – мигание светодиодами. В этой программе будем пользоваться функциями библиотеки SPL. Далее рассмотрим пример включения/выключения светодиодов с использованием непосредственного конфигурирования регистров. Выбрав в дереве проекта Source Group 1, в контекстном меню выбираем пункт «Add New Item to Group 'Source group 1'». В открывшемся окне выбираем тип файла C File(.c), вводим название main, в пути к файлу дописываем \src\main\, чтобы создаваемый файл исходного кода хранился в отдельной папке. В папке верхнего уровня \src в дальнейшем будем хранить файлы исходного кода, разбитые по группам (например, FreeRTOS, Drivers и т.д.)

Напишем код, который управляет миганием двух светодиодов. Задержка формируется пока при помощи циклов.

```
#include <stm32f10x_gpio.h>
#include <stm32f10x_rcc.h>
int main()
{
	// подаем тактирование на порт D
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
	// настраиваем линии 0 и 1 порта D на выход
	GPIO_InitTypeDef gpioInit; // создаем структуру для конфигурации порта
	gpioInit.GPIO_Mode = GPIO_Mode_Out_PP; // выход push-pull
	gpioInit.GPIO_Speed = GPIO_Speed_50MHz; // настройка крутизны выходных импульсов
```

```
        gpioInit.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
        GPIO_Init(GPIOD, &gpioInit);
        while (1)
        {
            GPIO_SetBits(GPIOD, GPIO_Pin_0);
            GPIO_ResetBits(GPIOD, GPIO_Pin_1);
            for (int i = 0; i < 1000000; i++);
            GPIO_ResetBits(GPIOD, GPIO_Pin_0);
            GPIO_SetBits(GPIOD, GPIO_Pin_1);
            for (int i = 0; i < 1000000; i++);
        }
        return 0;
}
```

Перед компиляцией в настройках проекта (рис. 11) в разделе C/C++ необходимо поставить флажок напротив пункта C99 Mode, а в разделе Debug установить в качестве отладчика ST-Link Debugger (рис. 12). В настройках отладчика во вкладке Flash Download следует отметить флажок напротив пункта Reset and run (рис. 13).

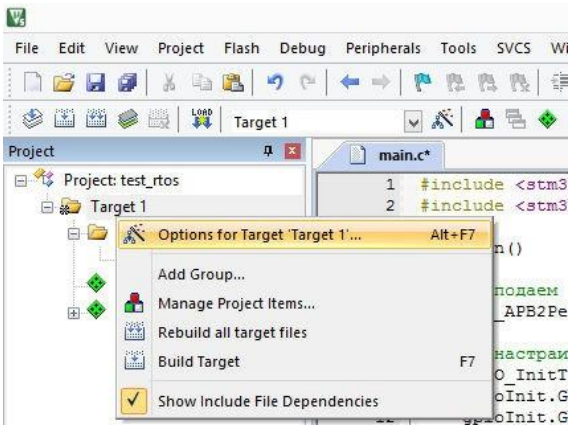

Рис. 11 – Вызов окна настройки проекта из контекстного меню

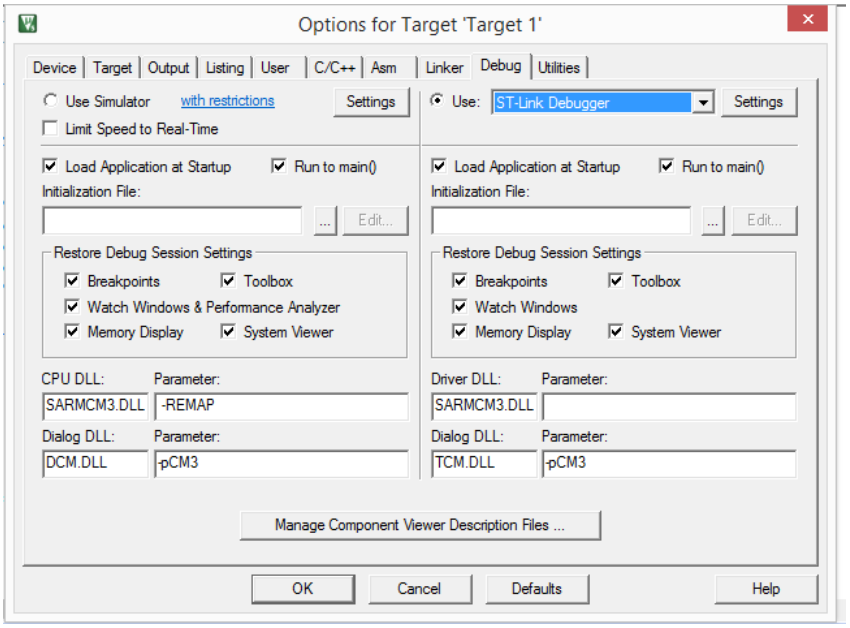

Рис. 12 – Окно выбора отладчика

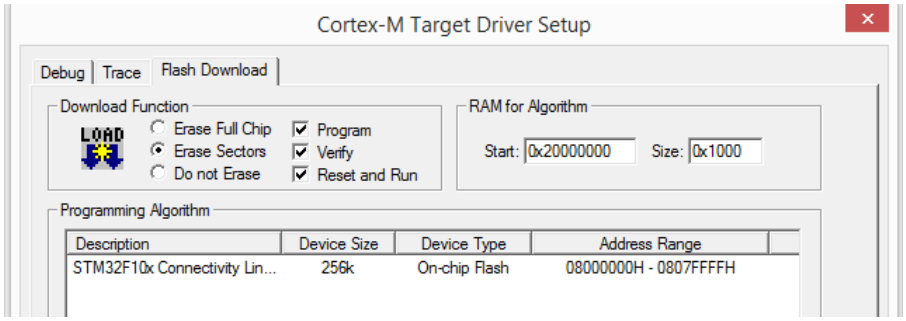

Рис. 13 – Окно настройки отладчика

После изменений в настройках проекта необходимо нажать кнопку «Сохранить все».

Собираем проект (Project ->Build target), прошиваем в плату (Flash -> Download). Видим, что светодиоды, подключенные к линиям PD0 и PD1, мигают по очереди. Основной минус программы – задержка формируется за счет выполнения ядром процессора пустых операций, что нерационально.

Поясним смысл программы, представленной в примере. Для работы с портами ввода-вывода общего назначения необходимо подключить библиотеку stm32f10x_gpio.c (инструкцией #include "stm32f10x_gpio.h"). Также необходимо подключить блок тактирования и сброса инструкцией #include "stm32f10x_rcc.h". Модуль GPIO соединен с шиной APB2. Следующая инструкция включает тактирование модуля GPIOD:

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
```

Конфигурирование порта осуществляется через соответствующую структуру – это общий подход, используемый в SPL. После ввода параметров вызывается функция-конфигуратор GPIO_Init(). Открыв исходный код этой функции, можно убедиться, что в ней настраиваются описанные выше регистры. Для просмотра определений нужно выделить идентификатор и в контекстном меню выбрать “Go To Definition Of ‘<идентификатор>’.

Примеры

1) В предыдущем примере конфигурирование и работа с портами ввода-вывода осуществлялась при помощи функций стандартной библиотеки. Напишем программу, так же мигающую двумя светодиодами, но без использования SPL.

Для настройки тактирования порта D нужно в регистре разрешения тактирования шины APB2 (RCC_APB2ENR) установить бит IOPDN. Далее конфигурируем линии GPIOD_Pin_0 и GPIOD_Pin_1 на выход push-pull, частоту переключения этих линий устанавливаем равной 50 МГц.

```
#include "stm32f10x.h"
int main(void)
{
// разрешаем тактирование порта D
RCC->APB2ENR |= RCC_APB2ENR_IOPDEN;
// сбрасываем биты CNF0 = 00, CNF1 = 00 (линии GPIOD_Pin_0 и GPIOD_Pin_1
// работают в режиме выхода GPIO Push-pull)
GPIOD->CRL &= !(GPIO_CRL_CNF1_1 | GPIO_CRL_CNF1_0 | GPIO_CRL_CNF0_1 |
GPIO_CRL_CNF0_0);
// устанавливаем биты MODE1 = 11, MODE2 = 11 (линии линии GPIOD_Pin_0 и GPIOD_Pin_1
// переключаются с частотой 50 МГц)
GPIOD->CRL |= GPIO_CRL_MODE1_1 | GPIO_CRL_MODE1_0 | GPIO_CRL_MODE0_1 |
GPIO_CRL_MODE0_0;
	while (1) {
		GPIOD->BSRR |= GPIO_BSRR_BS1; // устанавливаем значение GPIOD_Pin_1
		GPIOD->BRR |= GPIO_BRR_BR0; // сбрасываем значение GPIOD_Pin_0
		for (int i = 0; i < 1000000; i++);
		GPIOD->BSRR |= GPIO_BSRR_BS0; // устанавливаем значение GPIOD_Pin_0
		GPIOD->BRR |= GPIO_BRR_BR1; // сбрасываем значение GPIOD_Pin_1
		for (int i = 0; i < 1000000; i++);
	}
}
```

2) Рассмотрим, каким образом можно считывать значение с порта. Пусть при первом нажатии кнопки загорается светодиод, подключенный к линии PD0, при втором – подключенный к линии PD1. Настройку уровня нажатия кнопки нужно произвести согласно рис. 4, 5. Приведем ниже код, написанный с использованием SPL. Для считывания данных с линии PD2 используется функция GPIO_ReadInputDataBit(), для записи данных в линии PD1 и PD0 используется функция GPIO_WriteBit(). Самостоятельно изучите эти и другие функции работы с портами ввода-вывода общего назначения по справке по библиотеке SPL. Разберитесь с исходным кодом функции GPIO_WriteBit(). Чем отличается реализация функции GPIO_WriteBit() от функции GPIO_SetBits()?

```
#include <stm32f10x_gpio.h>
#include <stm32f10x_rcc.h>
int main()
{
	// подаем тактирование на порт D
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
	// настраиваем линии 0 и 1 порта D на выход
	GPIO_InitTypeDef gpioInit;
	gpioInit.GPIO_Mode = GPIO_Mode_Out_PP;
	gpioInit.GPIO_Speed = GPIO_Speed_50MHz;
	gpioInit.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
	GPIO_Init(GPIOD, &gpioInit);

	gpioInit.GPIO_Mode = GPIO_Mode_IPU;
	gpioInit.GPIO_Pin = GPIO_Pin_2;
	GPIO_Init(GPIOD, &gpioInit);
	uint16_t state = 0;
	while (1)
	{
		if (!GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_2))
		{
			if (state)
			{
				GPIO_WriteBit(GPIOD, GPIO_Pin_1, Bit_SET);
				GPIO_WriteBit(GPIOD, GPIO_Pin_0, Bit_RESET);
```

```
                        state = 0;
                }
                else
                {
                        GPIO_WriteBit(GPIOD, GPIO_Pin_1, Bit_RESET);
                        GPIO_WriteBit(GPIOD, GPIO_Pin_0, Bit_SET);
                        state = 1;
                }
        }
        for (int i = 0; i < 1000000; i++);
    }
    return 0;
}
```

3) Реализуем предыдущую задачу без использования библиотеки SPL.

```
#include "stm32f10x.h"
int main(void)
{
// 0. разрешаем тактирование порта D
RCC->APB2ENR |= RCC_APB2ENR_IOPDEN;
// 1. GPIOD_Pin_0 и GPIOD_Pin_1 - на выход, Push-pull, 50 МГц
// сбрасываем биты CNF0 = 00, CNF1 = 00 (линии GPIOD_Pin_0 и GPIOD_Pin_1
// работают в режиме выхода GPIO Push-pull)
GPIOD->CRL &= ~(GPIO_CRL_CNF1_1 | GPIO_CRL_CNF1_0 | GPIO_CRL_CNF0_1 |
GPIO_CRL_CNF0_0);
// устанавливаем биты MODE1 = 11, MODE2 = 11 (линии линии GPIOD_Pin_0 и GPIOD_Pin_1
// работают на выход, переключаются с частотой 50 МГц
GPIOD->CRL |= GPIO_CRL_MODE1_1 | GPIO_CRL_MODE1_0 | GPIO_CRL_MODE0_1 |
GPIO_CRL_MODE0_0;

// 2. GPIOD_Pin_2 - на вход, pull-up (подтяжка через внутренний резистор к питанию)
// устанавливаем биты MODE2 = 00 (линия GPIOD_Pin_2 - на вход)
GPIOD->CRL &= ~(GPIO_CRL_MODE2);
// устанавливаем биты CNF2 = 10 (линия GPIOD_Pin_2 - вход с подтяжкой)
GPIOD->CRL &= ~(GPIO_CRL_CNF2);
GPIOD->CRL |= GPIO_CRL_CNF2_1;
// устанавливаем бит 2 регистра ODR в 1 - подтяжка к питанию
GPIOD->ODR |= GPIO_ODR_ODR2;

uint16_t state = 0;
while (1) {
        if (!(GPIOD->IDR & GPIO_Pin_2))
        {
                if (state)
                {
                        GPIOD->BSRR |= GPIO_BSRR_BS1;//устанавливаем бит 1 порта D
                        GPIOD->BRR |= GPIO_BRR_BR0;//сбрасываем бит 0 порта D
                        state = 0;
                }
                else
                {
                        GPIOD->BSRR |= GPIO_BSRR_BS0;//устанавливаем бит 0 порта D
                        GPIOD->BRR |= GPIO_BRR_BR1;//сбрасываем бит 0 порта D
                        state = 1;
                        }
                }
                for (int i = 0; i < 1000000; i++) {};//задержка
        }
}
```

4) Приведем примеры конфигурирования линий портов ввода-вывода общего назначения для различных случаев

```
//Полагаем что выводы после сброса в режиме плавающего входа

//разрешаем тактирование порта A
RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;

//вход с подтяжкой к +
GPIOA->CRL &= ~GPIO_CRL_CNF0;
GPIOA->CRL |= GPIO_CRL_CNF0_1;
GPIOA->ODR |= GPIO_ODR_ODR0;

//вход с подтяжкой к -
GPIOA->CRL &= ~GPIO_CRL_CNF1;
GPIOA->CRL |= GPIO_CRL_CNF1_1;
GPIOA->ODR &= ~GPIO_ODR_ODR1;

//аналоговый режим
GPIOA->CRL &= ~GPIO_CRL_CNF2;

//выход с открытым стоком 2MHz
GPIOA->CRL &= ~GPIO_CRL_CNF3;
GPIOA->CRL |= GPIO_CRL_CNF3_0;
GPIOA->CRL |= GPIO_CRL_MODE3_1;

//двухтактный выход 10MHz
GPIOA->CRL &= ~GPIO_CRL_CNF4;
GPIOA->CRL |= GPIO_CRL_MODE4_0;

//альтернативная ф-ция, двухтактный выход, 50 MHz
GPIOA->CRL &= ~GPIO_CRL_CNF5;
GPIOA->CRL |= GPIO_CRL_CNF5_1;
GPIOA->CRL |= GPIO_CRL_MODE5;

//альтернативная ф-ция, выход с открытым стоком, 50 MHz
GPIOA->CRL |= GPIO_CRL_CNF6;
GPIOA->CRL |= GPIO_CRL_MODE6;
```

Задания

1) Выясните, какие линии портов ввода-вывода микроконтроллера подключены к светодиодам. Для этого поочередно выведите логические единицы на порты. ВНИМАНИЕ! Нельзя одновременно включать больше 16 светодиодов (больше одного порта)! Результаты занесите в таблицу 2 (в таблице уже отмечено, что все линии порта D выведены).

2) Напишите программу, которая по первому нажатию кнопки включает один светодиод, по второму – два светодиода, по третьему – четыре светодиода, по четвертому – мигает несколькими светодиодами, по пятому нажатию выключает все светодиоды.

3) Напишите программу, выводящую на набор светодиодов в 8-битном формате номер нажатия кнопки.

4) Напишите программу, реализующую бегущую единицу.

5) Напишите программу, реализующую бегущий ноль.

6) Напишите программу, включающую по первому нажатию светодиоды, подключенные к нулевым линиям портов, затем – подключенные к первым линиям и т.д.

Таблица 2 – Подключение светодиодов к линиям портов ввода-вывода

	PORTA	PORTB	PORTC	PORTD	PORTE
0				+	
1				+	
2				+	
3				+	
4				+	
5				+	
6				+	
7				+	
8				+	
9				+	
10				+	
11				+	
12				+	
13				+	
14				+	
15				+	

7) Реализуйте функциональность кнопок «больше», «меньше», «ввод». По нажатию кнопки «больше» («меньше») к переменной должна прибавляться (из переменной должна вычитаться) единица, при этом значение переменной должно выводиться на один из портов. По нажатию кнопки «ввод» значение переменной должно переписываться в другую переменную, которая отображается на другом порте. Реализуйте при помощи таких кнопок сложение двух чисел. Продумайте обработку переполнения.
8) Реализуйте функциональность кнопок «вправо», «влево», «вверх», «вниз» [4].

Контрольные вопросы

1) Перечислите регистры, отвечающие за конфигурирование портов ввода-вывода общего назначения микроконтроллеров STM32 серии F1.

2) Объясните устройство и принцип работы контроллера линий портов ввода-вывода общего назначения микроконтроллеров STM32.

3) Объясните правила подключения кнопок и нагрузки к линиям портов ввода-вывода общего назначения (на вход – In floating, In Pull-Up, In Pull-Down, на выход – Out Push-Pull,.Out Open-Drain).

4) Перечислите источники тактирования микроконтроллеров STM32, их основные особенности и области применения.

5) Каким образом осуществляется процедура инициализации тактирования микроконтроллера?

Практическая работа №2

ИССЛЕДОВАНИЕ РЕЖИМОВ РАБОТЫ ТАЙМЕРОВ ОБЩЕГО НАЗНАЧЕНИЯ

Цель работы: изучение структуры, возможностей и режимов работы каналов таймеров общего назначения микроконтроллера STM32F107VCT6, получение практических навыков программирования таймеров в интегрированной среде разработки Keil uVision.

Введение

Микроконтроллеры семейства STM32 содержат несколько типов таймеров с разным набором возможностей, а их число, достаточно большое, зависит от модели контроллера. Помимо универсальных таймеров, отличающихся разнообразием функций с примерно равными возможности, в контроллере имеются специальные сторожевые таймеры (WDG) и 24-разрядный системный счетчик/таймер (SysTick). По функциям, реализуемым в универсальных таймерах, условно их разделяют на 3 группы: базовые (basic timers), общего назначения (general-purpose timers) и продвинутые (advanced-control timers) таймеры. Система наименования таймеров, начинающаяся с префикса TIM, позволяет классифицировать их по номерам. В различных моделях контроллеров таймеры с одинаковыми номерами обычно совместимы по функциям.

Таймеры общего назначения, в свою очередь, разделены на несколько групп. Например, в составе серии STM32F100 выделены три группы таймеров общего назначения: TIM2-TIM5, TIM12-TIM14 и TIM15-TIM17. По функциям к ним близки базовые таймеры – TIM6, TIM7.

Эти группы таймеров незначительно отличаются по функциональным возможностям. Во многих приложениях часто применяются таймеры TIM2-TIM5, поскольку имеют большой набор программируемых режимов и отличаются многофункциональностью.

Таймер TIM1 (Advanced-control timer) с расширенными функциями предназначен для управления электроприводами переменного тока. Этот таймер, дополненный рядом аппаратных узлов, содержит три канала, имеющих по два противофазных выхода. В целом он представляет собой 6-канальный ШИМ-блок. Кроме того, здесь помимо интерфейса инкрементального энкодера предусмотрено подключение датчиков Холла, также выполняющих функции энкодера (звена обратной связи) электропривода.

Структура таймеров общего назначения

Таймеры общего назначения вместе с обычными операциями деления частоты, формирования одиночных сигналов или временных задержек реализуют три стандартных для современных таймеров режима работы – захват/сравнение/ШИМ (Capture/Campare/PWM – Pulse Width Modulation) и на их основе могут выполнять ряд дополнительных функций. Кроме того, возможности таймера расширяются за счет усложнения входных и выходных каскадов, включающих мультиплексоры, детекторы фронтов, цифровые фильтры, компараторы, предделители и схемы управления, комплементарные (противофазные) выходы.

Таймер, как типовое периферийное устройство, включает базовые элементы: программно управляемый предделитель тактовой частоты, 16-разрядный двоичный реверсивный счетчик и аппаратный блок, реализующий операции захвата/сравнения/ШИМ. Существенно увеличить операционные возможности таймера позволяет многоканальная структура, в которой базовые элементы дополняются блоками каналов. Как правило, таймеры содержат от 4 до 7 каналов. Блоки каналов таймера построены по одинаковым схемам, имеют собственные регистры сравнения/захвата и программно конфигурируемые входные и выходные каскады с достаточно сложной структурой. Каждый канал соединен с основным внешним выводом CHx, который в зависимости от режима работы настраивается на ввод (режим захвата) и часто обозначается как ICx, или на вывод (режимы сравнения и ШИМ) – OCx, где x – номер канала. Каналы независимо установливаются в один из основных режимов работы таймера – захват/ сравнение/ШИМ.

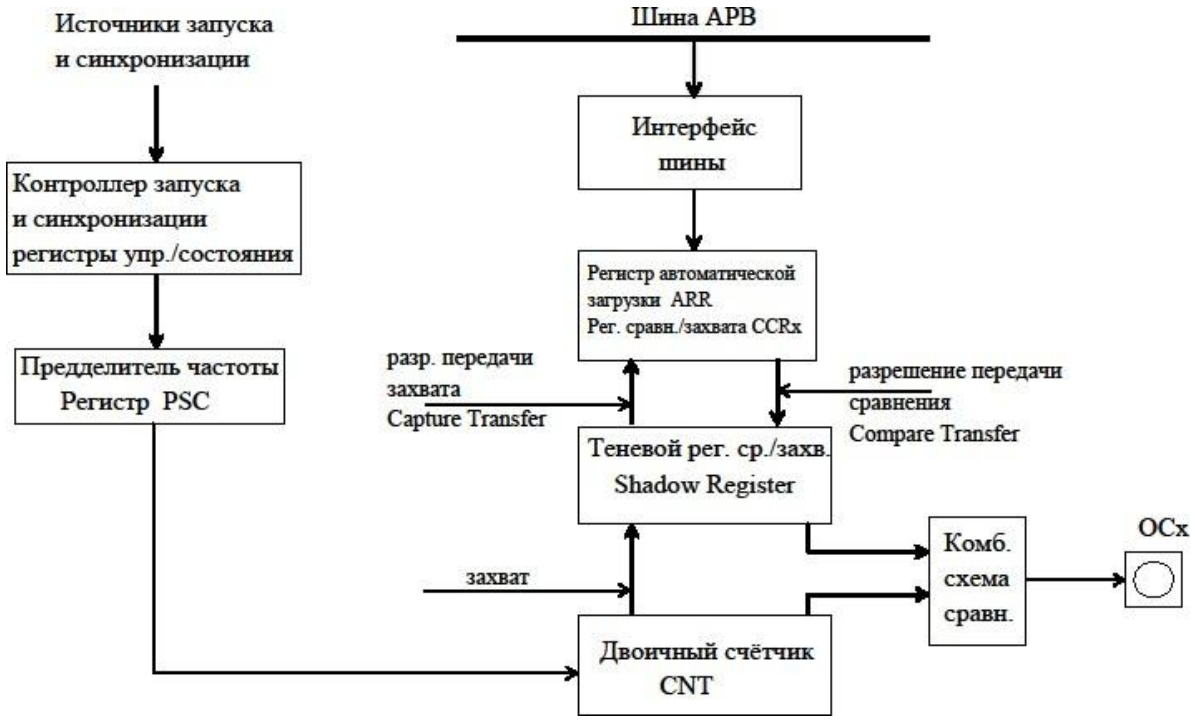

Рис. 1 - Упрощенная структурная схема канала таймера

Упрощенная структура таймера с одним каналом показана на рис. 1. Таймер построен на основе 16-разрядного двоичного счетчика CNT с 16-разрядным делителем частоты PSC и автоматически перезагружаемым регистром ARR (auto-reload-register). Счетчик таймера может работать в режиме суммирования, вычитания или реверсивного счета, досчитывая до заданного значения, а затем до нулевого.

Ко входу таймера через предварительный делитель частоты PSC подключается сигнал от одного из нескольких источников. В их число входят: специальный сигнал, генерируемый основной системой синхронизации, выходной сигнал от других таймеров, внешние сигналы тактирования или управления, подаваемые на выводы захвата (ICx). Коэффициент деления задается программно в 16-разрядном двоичном счетчике блока PSC и изменяется в широком диапазоне:1- 65536.

При программировании канала в режиме делителя частоты в регистр автозагрузки ARR загружается коэффициент деления (счета), а в режиме сравнения и ШИМ в него заносится двоичный код числа, определяющий длительность цикла повторения. Перед запуском счетчика в очередном цикле счета регистр ARR перезагружается в теневой регистр.

Блок захвата / сравнения состоит из программно доступного регистра предварительной загрузки CCRx (capture compare register), ассоциированного с ним теневого регистра, 16-разрядной схемы сравнения и программно реконфигурируемых входных и выходных каскадов. Для регистра предварительной загрузки всегда доступны операции записи и чтения. Во всех режимах захвата/сравнения/ШИМ, формирования одиночных сигналов и других участвует программно недоступный теневой регистр shadow register (фактически буфер данных для регистра захвата/сравнения). В режиме захвата данные из него копируются в регистр CCRx, а в режиме сравнения содержимое регистра CCRx загружается в теневой регистр.

В режимах сравнения и ШИМ в процессе счета текущее значение двоичного счетчика и содержимое теневого регистра на каждом такте работы сравнивается в 16-разрядной комбинационной схеме компаратора. Сигнал с компаратора поступает на вход контроллера режима работы схемы сравнения, формирующего сигнал совпадения OCxref, который в дальнейшем используется для изменения состояния выхода OCx таймера.

Режимы работы таймера общего назначения

Режим деления частоты

Суммирующий счетчик. В режиме деления частоты суммирующий (считающий вверх) таймер работает от 0 до предустановленного значения (содержимого регистра автоперезагрузки ARRx). При совпадении кодов генерируется событие переполнения счетчика (событие - Overflow), а затем происходит перезапуск с нулевого значения (счет с 0). Циклы выполняют-

ся непрерывно по мере заполнения счетчика. В тоже время, с помощью счетчика циклов предусмотрен и режим N кратного повторения.

Счётчик повторений RCRx. Этот счётчик имеется у нескольких таймеров (TIM15, TIM16 и TIM17) и выполняет следующую функцию: генерировать событие обновления (UEV) (прерывание или запрос DMA) не на каждое переполнение счётчика, а на каждые N переполнений. Счётчик может принимать значения от 0 до 255. Когда счётчик повторений настроен, таймер его копирует в теневой регистр и при каждом переполнении уменьшает значение на 1. Когда значение достигает нуля, генерируется событие обновления, таймер снова копирует счётчик повторений и т.д. В противном случае событие обновления генерируется при каждом переполнении счетчика.

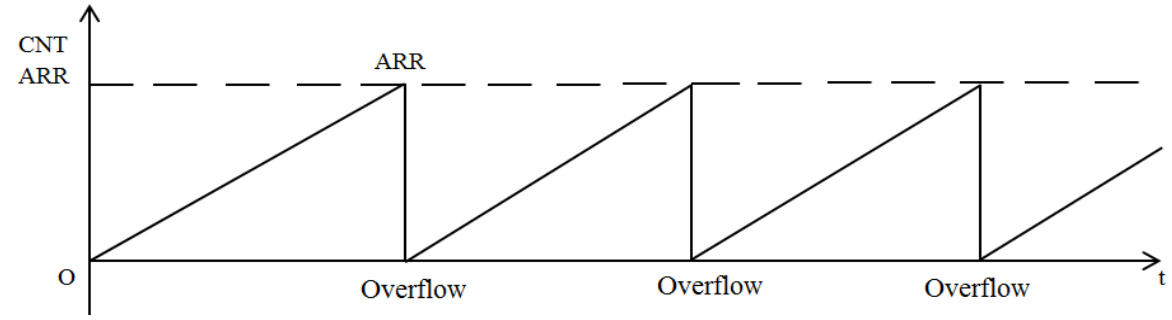

Рис. 2 - Временные диаграммы режима деления частоты в суммирующем счетчикес выравниванием по краю

Вычитающий счетчик. В регистр предзагрузки ARRx записывается значение счётчика, с которого таймер начинает работу, уменьшая содержимое по каждому импульсу на входе. При обнулении счётчика генерируется прерывание «ОбновлениеТаймера» (событие - Underflow), в счётчик записывается значение из ARRx и цикл повторяется.

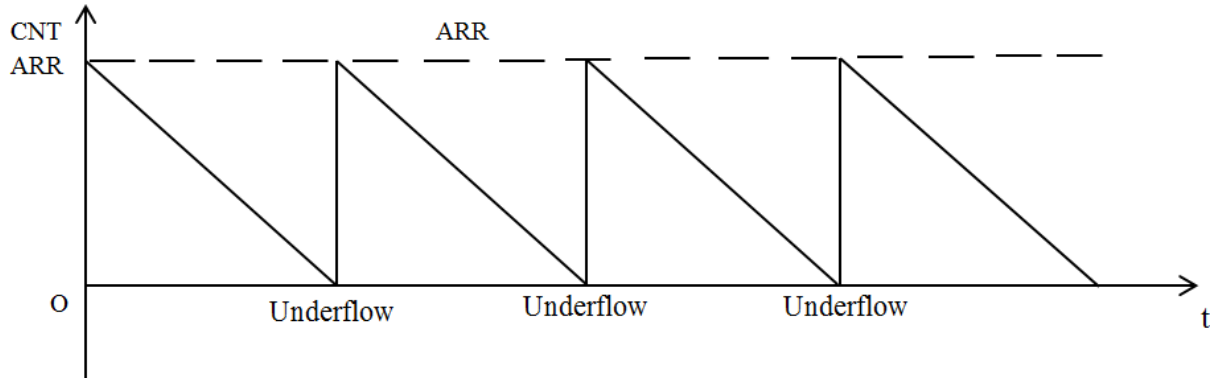

Рис. 3 - Временные диаграммы режима деления частоты в вычитающем счетчике свыравниванием по краю

В режиме реверсивного счета с использованием регистра ARRx выровненный по центру таймер считает сначала вверх, а затем вниз.

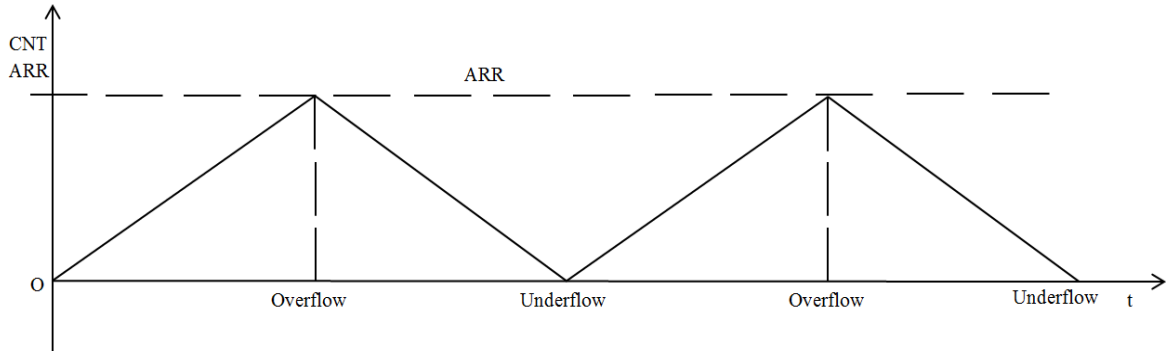

Рис. 4 - Временные диаграммы режима деления частоты с выравниванием по центру

Режим сравнения

Это базовый режим работы таймера при выполнении операций захват/сравнение/ШИМ. При инициализации канала в регистр настройки (предустановки) канала CCRx загружается код, который после запуска счетчика, считающего вверх, на каждом такте схемой компаратора сравнивает его текущее состояние с содержимым регистра CCRx. Совпадение значений кодов в счетчике и регистре CCRx вызывает запрос прерывания или ПДП, а также может изменять состояние выхода канала OCx по правилам, заданным в поле OCxM[2:0] регистра управления CCMR.

Временные диаграммы режима сравнения приведены на рис. 5.

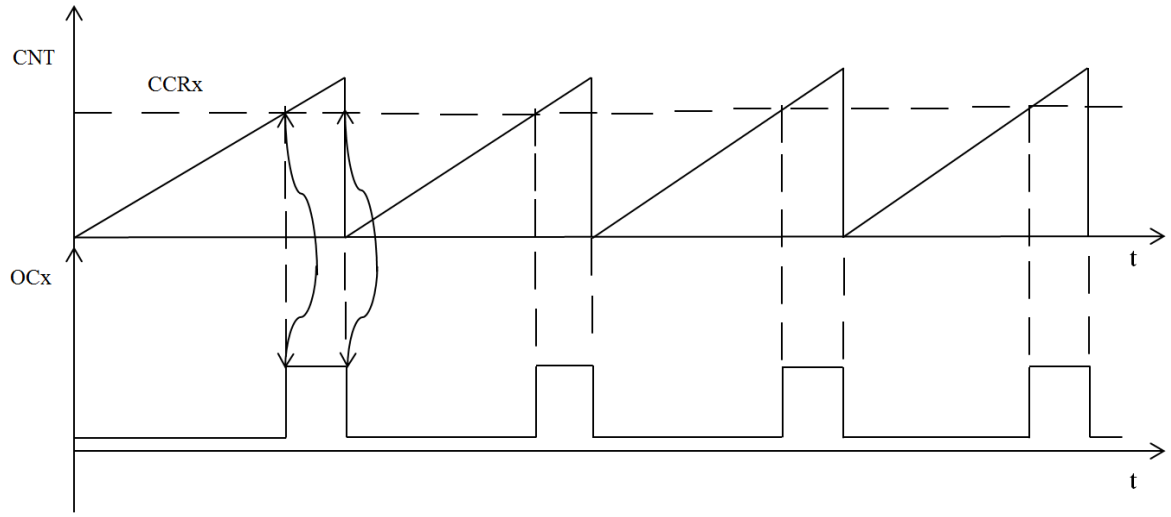

Рис. 5 - Временные диаграммы режима сравнения с выравниванием по краю

Режим захвата сигнала

В этом режиме под функцией захвата понимается пересылка текущего состояния счётчика в регистр захвата по внешнему событию. Для реализации режима детектор фронта входного каскада выбирает полярность фронта внешнего сигнала на выводе ICx – нарастающий (передний) или спадающий (задний). При этом одним фронтом сигнала на выводе ICx счетчик запускается, а другим фронтом, противоположным к запускающему, разрешается перезапись его текущего состояния в регистр

сравнения/захвата _CCRx. После завершения цикла в регистре захвата CCRx оказывается число импульсов, пропорциональное длительности временного интервала, ограниченного этими фронтами входного сигнала. При известном значении периода тактовых импульсов и сохраненном числе импульсов легко вычислить измеряемый интервал времени. При этом можно настроить генерацию прерывания и запроса DMA на приход очередного импульса, и, если в это время предыдущее значение CCRx не было считано, то будет сгенерировано так называемое прерывание over-capture, т.е. сигнал о том, что предыдущее значение потерялось.

Режим ШИМ

Принцип работы в режиме ШИМ заключается в следующем. В регистр предзагрузки ARRx загружается число, определяющее длительность периода поторения импульсов, а в регистр CCRx – число, определяющее длительность активной части импульсов (полупериода), так называемый коэффициент заполнения. На временной диаграмме (рис. 6) показана работа счетчика в режиме ШИМ. Счетчик работает непрерывно, досчитывая в каждом цикле до значения, установленного в регистре ARRx (событие «Переполнения»), после чего сбрасывается и начинает новый цикл счета. Как и в режиме сравнения непрерывно на каждом такте в компараторе происходит сравнение кодов в счетчике и теневом регистре CCRx.

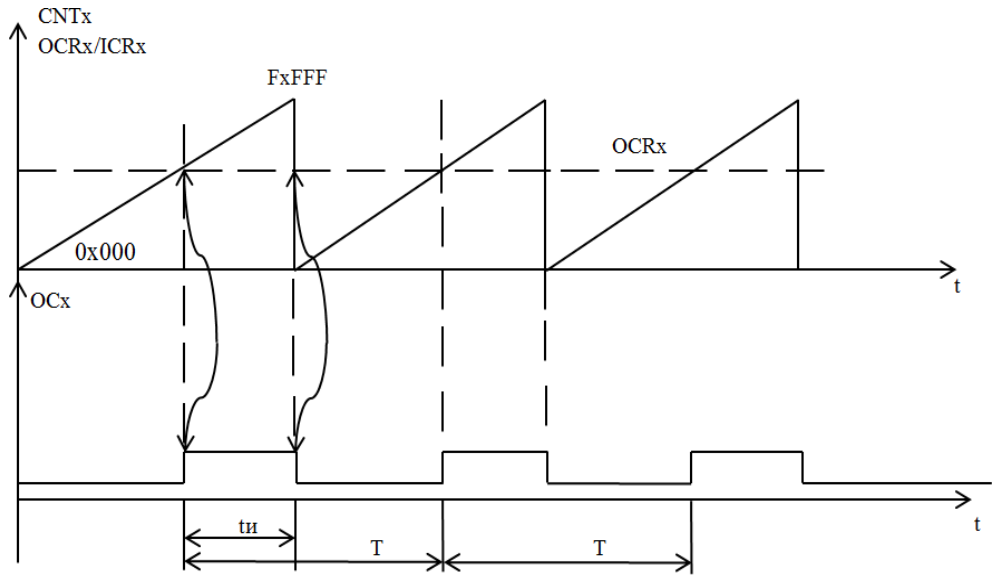

Рис. 6 - Временные диаграммы режима ШИМ
(выравнивание по краю)

Как и в режиме деления частоты, реализованы 2 способа выравнивания ШИМ-сигнала – выравнивание по краю (edge-aligned) и выравнивание по центру (center-aligned). На рис. 6 временные диаграммы работы таймера в режиме выравнивания по краю. В этом случае, состояние выходного

сигнала канала таймера (внешний вывод OCx) изменяется на противоположное по двум событиям – совпадение кодов в регистрах CCRx и CNT и переполнение счетчика (при достижении им значения, предустановленного в регистре ARRx). Одновременно генерируется сигнал запроса прерывания. При этом длительность положительного полупериода $t_и$ оказывается пропорциональной коэффициенту заполнения (код в регистре CCR_x), а период повторения T будет постоянным, заданным в регистре ARRx.

На рис. 7 приведена временная диаграмма режима ШИМ с выравниванием по центру.

Генерирование сигналов с широтно-импульсной модуляцией реализуется в двух вариантах (режимах): ШИМ1 и ШИМ2. Режимы отличаются полярностью формируемых сигналов – положительная и отрицательная.

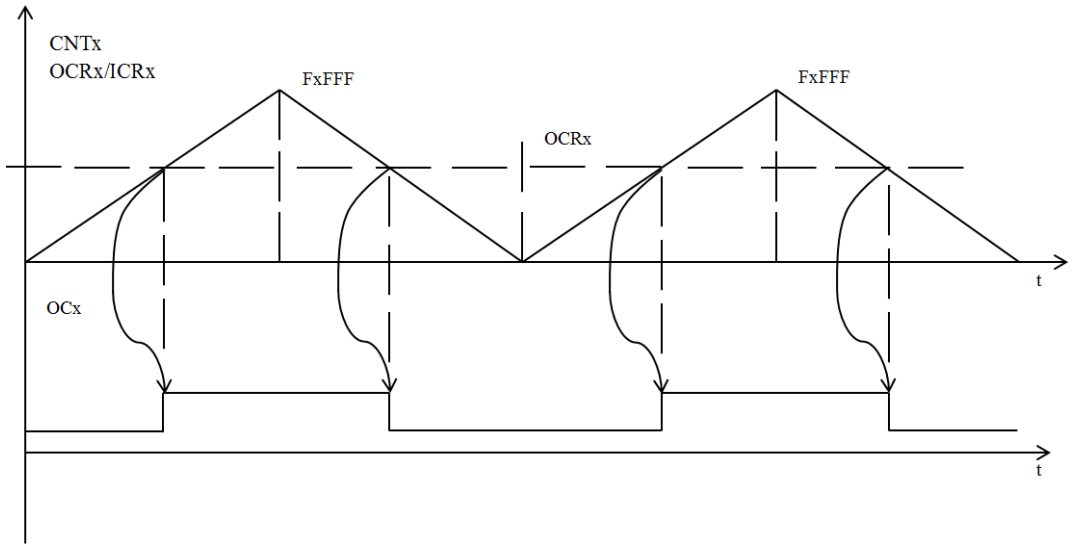

Рис. 7 - Временные диаграммы режима ШИМ (выравнивание по центру)

Регистры управления таймером

Программирование режимов работы таймера и настройка отдельных параметров для каждого канала производится индивидуально с помощью набора регистров управления и состояния. Во таймерах общего назначения микроконтроллеров STM32 серии F105/107 реализовано 15 типов регистров управления/состояния.

Адреса регистров управления отражены в адресном пространстве УВВ и расположены в нем в приведенном ниже порядке.

1) Регистры управления

Регистр управления 1 TIMx_CR1 (control register 1, address offset: 0x00)

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

Регистр управления 2 TIMx_CR2 (control register 2, address offset: 0x04)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TI1S	MMS[2:0]			CCDS	Reserved		
								rw	rw	rw	rw	rw			

Регистры управления реализуют множество функций, среди которых можно выделить следующие:

- каждый из каналов таймера может работать с прерываниями и/или поддерживать передачу данных по каналу ПДП: поля IC1F[2:0] – IC4F[2:0] и OC1M[2:0] – OC4M[2:0] соответственно,
- выбор режима сравнения или захвата: если поле CCxS=00, то режим сравнения, иначе – режим захвата (x=1…4, номер канала),
- установка режима ШИМ – ШИМ1 или ШИМ2,
- программное маскирование и запуск канала,
- правила установки состояния выхода OCx канала – поле OCxM[2:0] (табл. 1),
- выравнивание по краю или по центру,
- конфигурирование выходного каскада: полярность сигнала, комплементарный выход.

Приведем назначение некоторых конфигурационных битов.

ARPE – использование буферного регистра для регистра автоматической перезагрузки TIMx_ARR:

0 – содержимое TIMx_ARR будет обновлено сразу же после записи нового значения.

1 – счетчик досчитает до предыдущего значения и, только потом, после генерации события, содержимое TIMx_ARR обновится.

DIR – направление счета

0 – увеличение

1 – уменьшение

UDIS – разрешает генерацию события при переполнении/обнулении счетчика (а также в некоторых других случаях). По умолчанию бит сброшен. Это состояние разрешает генерацию событий. Установка бита в “1” запретит генерацию событий.

CEN – запуск счета производится записью “1” в этот разряд.

2) Регистр управления в режиме подчиненного TIMx_SMCR (slave mode control register, address offset: 0x08)

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

3) Регистр разрешения ПДП/прерываний TIMx_DIER (DMA/interrupt enable register, address offset: 0x0C)

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

В регистре DIER размещены разряды для разрешения/запрещения прерывания и запроса DMA от каждого канала. Разряды CC1IE….CC4IE разрешают/запрещают прерывания от соответствующих каналов, разряды CC1DE…CC4DE – запросы DMA.

4) Регистр состояния TIMx_SR (status register, address offset: 0x10)

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

В регистре статуса TIMx_SR потребуется отслеживать состояние бита **UIF** (флаг произошедшего события или генерации прерывания), который устанавливается при переполнении или обнулении счетного регистра таймера.

5) Регистр генерации событий TIMx_EGR (event generation register, address offset: 0x14)

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

6) Регистры режима захват/сравнение

Регистр режима захват/сравнение 1 TIMx CCMR1 (capture/compare mode register 1, address offset: 0x18)

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]	
IC2F[3:0]				IC2PSC[1:0]				IC1F[3:0]				IC1PSC[1:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Формат верхнего регистра для режима сравнения и ШИМ, нижнего – режима захвата.

Регистр режима захват/сравнение 2 TIMx CCMR2 (capture/compare mode register 2, address offset: 0x1C)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]		OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]	
IC4F[3:0]				IC4PSC[1:0]				IC3F[3:0]				IC3PSC[1:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Формат верхнего регистра для режима сравнения и ШИМ, нижнего – режима захвата.

Назначение отдельных разрядов регистров управления:

OCxCE – управление выводом OCx по входному сигналу ETRF, если 0, то не зависит, при 1 выход сбрасывается;

OCxPE – отключение/включение регистра предзагрузки CCRx, 0 – отключен и регистр может быть загружен в любое время, иначе при наступлении события обновления UEV;

OCxFE – используется в режимах ШИМ1 и ШИМ2 для ускорения воздействия триггера входного каскада на сигнал в выходной цепи, 0 – связи между этими элементами нет.

В табл. 1 показаны правила установки состояния выхода OCx.

Табл. 1 – Правила установки состояния выхода OCx

OCxM[2:0]	Управление формой выходного сигнала OCx для канала с номером x
000	совпадение регистра сравнения (CCRx) и счетного регистра (CNT) не влияет на сигнал OC1ref
001	при CCR1=CNT сигнал OC1ref=1
010	при CCR1=CNT сигнал OC1ref=0
011	при CCR1=CNT сигнал OC1ref меняется на противоположный (toggle)
100	OC1ref принудительно сбрасывается в ноль, независимо от результата сравнения
101	OC1ref принудительно устанавливается в единицу, независимо от результата сравнения
110	режим ШИМа №1. Если счетный регистр работает на сложение: при CNT<CCR1 сигнал OC1ref=1, иначе OC1ref=0. Если счетный регистр работает на вычитание: при CNT>CCR1 сигнал OC1ref=0, иначе OC1ref=1.
111	режим ШИМа №2. Если счетный регистр работает на сложение: при CNT<CCR1 сигнал OC1ref=0, иначе OC1ref=1. Если счетный регистр работает на вычитание: при CNT>CCR1 сигнал OC1ref=1, иначе OC1ref=0.

7) Регистр разрешения захвата/сравнения TIMx_CCER (capture/compare enable register, address offset: 0x20)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		CC4P	CC4E	Reserved		CC3P	CC3E	Reserved		CC2P	CC2E	Reserved		CC1P	CC1E
		rw	rw			rw	rw			rw	rw			rw	rw

8) Счетный регистр TIMx_CNT (counter, address offset: 0x24)

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

9) Предделитель TIMx_PSC (prescaler, address offset: 0x28)

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

10) Регистр автоматической перезагрузки TIMx_ARR (auto reload register, address offset: 0x2C)

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

12) Регистры захвата/сравнения

Регистр захвата-сравнения TIMx_CCR1 (capture/compare register 1, address offset: 0x34)

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

Регистр захвата-сравнения TIMx_CCR2 (capture/compare register 2, address offset: 0x38)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR2[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Регистр захвата-сравнения TIMx_CCR3 (capture/compare register 3, address offset: 0x3C)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR3[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Регистр захвата-сравнения TIMx_CCR4 (capture/compare register 4, address offset: 0x40)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR4[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

14) Регистр управления DMA TIMx_DCR (DMA control register, address offset: 0x48)

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

15) Регистр адреса DMA для полной передачи TIMx_DMAR (DMA address for full transfer, address offset: 0x4C)

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

Примеры использования таймеров-счетчиков

1) Выполнение подпрограммы через равные промежутки времени

Приведем код, в котором по прерыванию от таймера-счетчика TIM2 меняется состояние светодиодов на порте D (на рассматриваемом стенде все линии этого порта выведены на светодиоды). Функция initRCC() реализует настройку тактирования микроконтроллера. В рассматриваемой ситуации таймеры-счетчики, подключенные к шине APB1, тактируются сигналом частотой 32 МГц. Следовательно, если необходимо, чтобы светодиоды мигали один раз в секунду, установить значение предделителя 32000, а значение периода – 1000 (1000 мс светодиоды включены, 1000 мс - выключены). Конфигурирование таймера-счетчика осуществляется в функции init_timer(). Код, ответственный за смену состояния светодиодов, расположен в функции-обработчике прерывания. Важно: после входа в прерывание необходимо снять бит, сигнализирующий о том, что прерывание возникло, чтобы обеспечить реакцию микроконтроллера на другие прерывания TIM2_IRQHandler(). Управление прерываниями от периферийных устройств микроконтроллера осуществляется с помощью контрол-

лера приоритетных векторных прерываний (Nested Vectored Interrupt Controller, NVIC). При работе с контроллерами STM32 следует различать interrupt (прерывание) и event (аппаратное событие). При возникновении прерывания программа берет вектор-адрес (некоторая константа) соответствующего прерывания и помещает его в программный счетчик (при этом все регистры общего назначения помещаются в стек), после чего сразу происходит переход на обработчик прерывания. Event – аппаратное событие (опустошение буфера UART, достижение счетным регистром таймера определенного значения и др.). Событие может вызвать прерывание, но так бывает не всегда – событие прерывание вызывать не обязано. Номера и названия прерываний (в зависимости от типа контроллера) приведены в файле stm32f10x.h в перечислении IRQn.

Листинг 1

```
#include <stm32f10x.h>
#include <stm32f10x_gpio.h>
#include <stm32f10x_rcc.h>
#include <stm32f10x_tim.h>
#include <misc.h>
uint8_t state;
void initRCC(void)
{
	RCC_DeInit();
	RCC_HSICmd(DISABLE);
	RCC_HSEConfig(RCC_HSE_ON); // разрешаем тактирование от внешнего генератора
	ErrorStatus HSEStartUpStatus = RCC_WaitForHSEStartUp();
	if (HSEStartUpStatus == SUCCESS){ // если внешний генератор запустился
		RCC_PREDIV2Config(RCC_PREDIV2_Div5); // предделитель 2 - делим HSE на 5
		RCC_PLL2Config(RCC_PLL2Mul_8); // умножаем частоту источника на 8
		RCC_PLL2Cmd(ENABLE);
		// в качестве источника PREDDIV1 сигнал PLL2MUL, предделитель 5
		RCC_PREDIV1Config(RCC_PREDIV1_Source_PLL2, RCC_PREDIV1_Div5);
		// в качестве источника PLL сигнал PREDDIV1, умножение на 8
		RCC_PLLConfig(RCC_PLLSource_PREDIV1, RCC_PLLMul_8);
		RCC_PLLCmd(ENABLE);
		RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK); // источник SYSCLOCK – PLL
		RCC_HCLKConfig(RCC_SYSCLK_Div1); // делитель для HCLK (HCLK = SYSCLK)
		RCC_PCLK2Config(RCC_HCLK_Div1); // делитель для PCLK2 (PCLK2 = HCLK)
		RCC_PCLK1Config(RCC_HCLK_Div4); // делитель для  PCLK1 (PCLK1 = HCLK/4)
		// ожидаем, пока HSE не установится в качестве источника SYSCLOCK
		while (RCC_GetSYSCLKSource() != 0x08) {}
	}
	else //Если HSE не смог запуститься, тактирование настроено некорректно
	{
		while (1) {} //Здесь следует поместить код обработчика этой ошибки
	}
}

void init_leds(void) // Инициализируем все линии порта D на выход
{
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
	GPIO_InitTypeDef gpio;
	GPIO_StructInit(&gpio);
	gpio.GPIO_Mode = GPIO_Mode_Out_PP;
	gpio.GPIO_Pin = GPIO_Pin_All;
	GPIO_Init(GPIOD, &gpio);
}
```

```
/* SysClock = 64 МГц, HCLK = 64 МГц, PCLK1 = 32 МГц. Сигнал PCLK1 подается на вход
таймеров, подключенных к шине APB1*/
void init_timer(void)
{
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
//Инициализируем базовый таймер общего назначения: делитель 32000, период 1000 мс.
	TIM_TimeBaseInitTypeDef base_timer;	TIM_TimeBaseStructInit(&base_timer);
	base_timer.TIM_Prescaler = 32000 - 1;
	base_timer.TIM_Period = 1000;
	TIM_TimeBaseInit(TIM2, &base_timer);
	TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE); // Разрешаем прерывание по обновлению
	TIM_Cmd(TIM2, ENABLE); // Включаем таймер
	NVIC_EnableIRQ(TIM2_IRQn); // Разрешаем обработку прерывания по переполнению
}

int main(void)
{
	initRCC();
	init_leds();
	init_timer();
	state = 0;
	while (1) {};
}

void TIM2_IRQHandler()// обработчик прерывания от таймера TIM2
{
	/* Очищаем бит обрабатываемого прерывания */
	TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
	if (state){
		GPIOD->ODR = 0xAAAA; state = 0;
	}
	else {
		GPIOD->ODR = 0x5555; state = 1;
	}
}
```

Для сравнения приведем реализацию функций init_timer() и обработчика прерывания TIM2_IRQHandler(), где конфигурирование таймера-счетчика осуществляется через соответствующие регистры.

Листинг 2

```
void init_timer()
{
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
	TIM2->PSC = 32000 - 1; // настройка делителя частоты
	TIM2->ARR = 1000;// настройка периода счетчика
	TIM2->DIER |= TIM_DIER_UIE; // разрешение прерывания по обновлению
	TIM2->CR1 |= TIM_CR1_CEN; //разрешаем работу таймера TIM2
	NVIC_EnableIRQ(TIM2_IRQn);
}

void TIM2_IRQHandler()// обработчик прерывания от таймера TIM2
{
	TIM2->SR &= ~TIM_SR_UIF; // сбрасываем флаг прерывания по обновлению
	if (state){
		GPIOD->ODR = 0xAAAA; state = 0;
	}
	else{
		GPIOD->ODR = 0x5555; state = 1;
	}
}
```

Упражнение 1. Создайте проект в среде Keil uVision на основе кода, представленного в листинге 1. Скомпилируйте и загрузите файл прошивки в стенд EasyMX Pro for STM32. Переключателями группы SW15 подключите наборы светодиодов к порту D. При помощи секундомера убедитесь в правильности формируемых таймером-счетчиком временных интервалов. Варьируя параметры предделителя, периода таймера счетчика, а также параметры блока тактирования и сброса (Reset and Clock Control, RCC), сформируйте временные интервалы различной длительности.

Указание

1) Создайте проект в конфигураторе STM32CubeMX, указав название микроконтроллера STM32F107VCTx. На вкладке Pinout в разделе RCC в качестве источника высокочастотного тактового сигнала (High Speed Clock, HSE) установите значение Crystal/Ceramic Resonator (на отладочном стенде установлен резонатор на частоту 25 МГц).

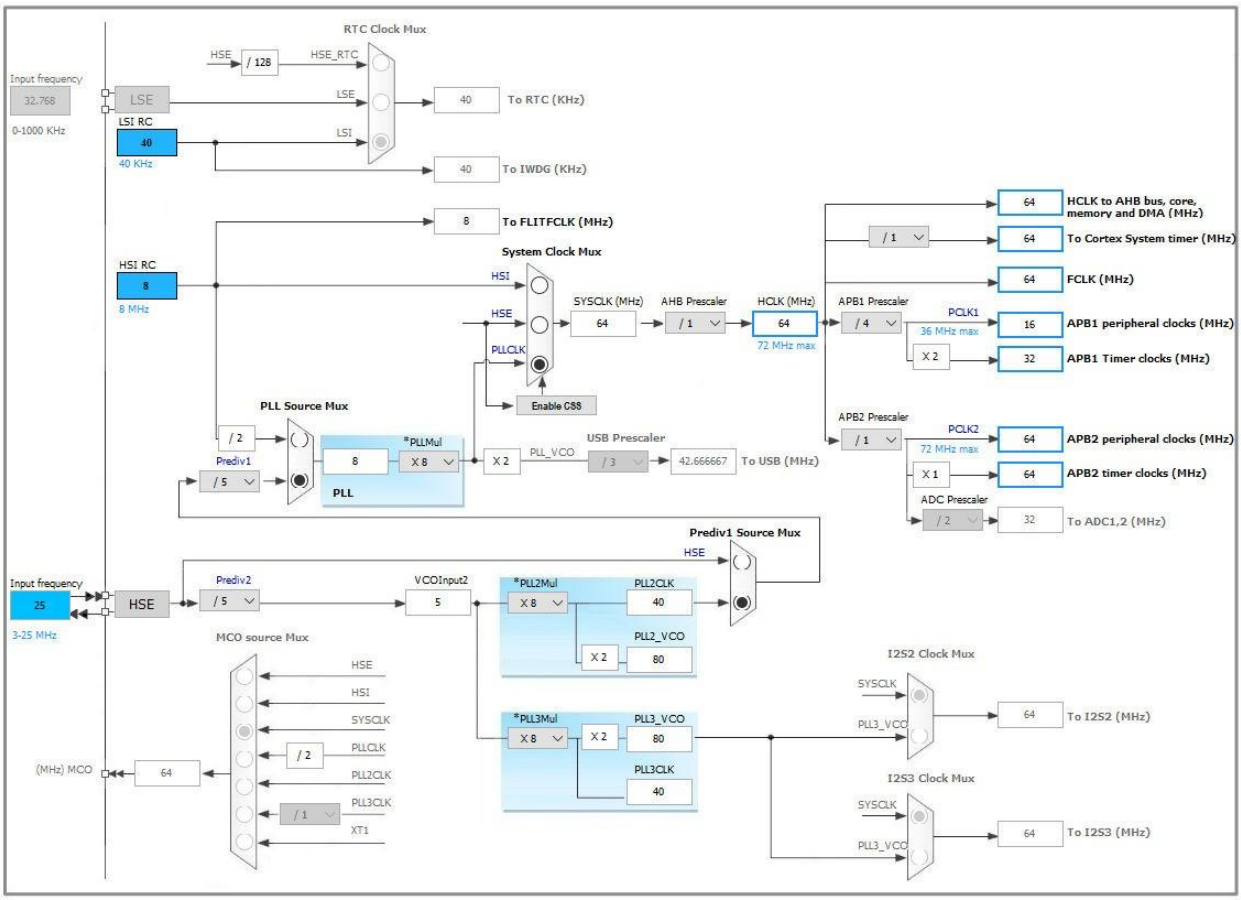

Рис. 8 – Конфигурирование блока RCC при помощи среды STM32CubeMX

2) На вкладке Clock Configuration (рис. 8) выполните следующие настройки:

- в окошке Input Frequency укажите значение 25 в соответствии с частотой присутствующего на стенде кварцевого резонатора;
- в окошке HCLK установите значение 64;
- значение APB1 Prescaler установите равным 4.

После этих установок выведется окно с предложением изменить источник тактового сигнала, поскольку введенные значения частот не могут быть обеспечены с использованием текущих значений делителей и множителей. При нажатии кнопки OK будет проведен автоматический подбор необходимых коэффициентов.

Сопоставьте код функции initRCC() и значения коэффициентов и переключателей на диаграмме, полученной на вкладке Clock Configuration. Попробуйте поменять настройки, выясните, как это отразится на частоте мигания светодиодов.

Упражнение 2. Замените код функции init_timer() и обработчика прерываний TIM2_IRQHandler() на код, приведенный на листинге 2. Убедитесь в правильности формирования временных задержек. В документации найдите регистры, используемые в этих функциях, выясните значения устанавливаемых битов, попробуйте заменить макроопределение (например, TIM_DIER_UIE, TIM_CR1_CEN) на битовые маски.

2) Формирование временной задержки. Функция delay_ms()

Не всегда возможно составить программу, чтобы действия, выполняющиеся через определенные промежутки времени, были запрограммированы в обработчике прерывания. Типичный пример – инициализация встраиваемого дисплея или GSM-модема: необходимо посылать команды друг за другом через регламентированные промежутки времени, причем такая процедура выполняется один раз при запуске устройства. Выходом из ситуации является использование функции delay_ms(), которая в течение заданного промежутка времени выполняет пустой цикл. Для реализации этой функции будем использовать таймер-счетчик TIM3. Инициализация производится аналогично приведенной выше инициализации таймера-счетчика TIM2, значение делителя нужно установить таким, чтобы тактовый сигнал, подаваемый на таймер, имел частоту 1 кГц, а значение периода можно назначить любым – это число будет передаваться в функцию delay_ms() в качестве параметра. Объявляем глобальную переменную

```
volatile uint8_t f_timer_end;
```

Модификатор volatile указывает компилятору, что при оптимизации кода данную переменную заменять чем-то и упрощать код с ее использованием нельзя.

Обработчик прерывания по событию «Обновление» приведем к следующему виду:

```
void TIM3_IRQHandler(void)
{
	extern volatile uint8_t f_timer_end;
	TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
	f_timer_end = 1;
	TIM_Cmd(TIM3, DISABLE); // запрещаем работу таймера TIM3
	TIM_ITConfig(TIM3, TIM_IT_Update, DISABLE); // запрещаем прерыв. по обновлению
}
```

То есть при возникновении прерывания значение переменной f_timer_end устанавливается равным 1, после чего работа таймера TIM3 запрещается и

запрещается прерывание по обновлению этого таймера. Модификатор extern указывает, что используется глобальная переменная.

Сама функция delay_ms() выглядит следующим образом:

```
void delay_ms(uint16_t n_msec)
{
	f_timer_end = 0;
	// частота счета 1 кГц
	TIM3->ARR = n_msec; // устанавливаем, сколько миллисекунд необходимо ждать
	TIM3->EGR |= TIM_EGR_UG; // устанавливаем бит Update Generation,
	                         // при этом таймер-счетчик переинициализируется
	TIM3->SR &= ~TIM_SR_UIF; // сбрасываем флаг прерывания по обновлению
	TIM3->DIER |= TIM_DIER_UIE; // разрешение прерывания по обновлению
	TIM3->CR1 |= TIM_CR1_CEN; // разрешаем работу таймера TIM3
	while (f_timer_end == 0);
}
```

Устанавливаем значение переменной f_timer_end равным нулю, затем настраиваем значение модуля счета (регистр ARR), далее настраиваем событие Update, разрешаем прерывание от таймера-счетчика TIM3, после чего в цикле ждем, когда установится флаг f_timer_end.

Из представленной реализации видно, что функция delay_ms неоптимальна, поскольку сводится к многократному выполнению пустой операции вплоть до установки некоторого флага. С другой стороны, использование таймера позволяет с высокой степенью точности формировать временные задержки. Следует отметить, что при использовании многопоточности в программе, например, при внедрении в проект операционной системы реального времени (ОСРВ) FreeRTOS, для формирования задержек следует пользоваться временной базой этой системы. В ОСРВ FreeRTOS функции задержки также реализуются через один из таймеров-счетчиков микроконтроллера.

Упражнение 3. По заданию преподавателя реализуйте с использованием функции delay_ms() смену состояний набора светодиодов.

Упражнение 4. Вынесите реализацию функции delay_ms() в отдельный файл.

Упражнение 5. Реализуйте функцию delay_us(), выполняющую задержку на заданное число микросекунд.

3) Формирование широтно-модулированного сигнала

Рассмотрим теперь формирование сигнала широтно-импульсной модуляции (ШИМ, в англоязычных текстах - PWM, pulse-width modulation) при помощи таймера. С помощью дискретного сигнала ("включено/выключено") можно управлять аналоговой величиной, что делает ШИМ очень популярной при использовании цифровых схем управления. Этот режим используется, например, при регулировании скорости вращения вала двигателя постоянного тока, когда на двигатель подается полное напряжение питания, но регулируется время, в течение которого оно подается.

Рассмотрим пример формирования ШИМ сигнала при помощи таймера счетчика TIM5 (листинг 3). На рис. 9 показаны осциллограммы сформированного широтно-модулированного сигнала.

Листинг 3

```
#define DUTY_CYCLE 1800

void init_timer_pwm(void)
{
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM5, ENABLE);
	TIM_TimeBaseInitTypeDef base_timer;
	TIM_TimeBaseStructInit(&base_timer);
	base_timer.TIM_Prescaler = 320 - 1; // частота составит 10^5 Гц
	base_timer.TIM_Period = 2000; // значение периода = 20 мс
	TIM_TimeBaseInit(TIM5, &base_timer);

	// Линия PA3 (TIM5_CH4) - в режиме альтернативной функции (выход таймера)
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO, ENABLE);
	GPIO_InitTypeDef gpio;
	GPIO_StructInit(&gpio);
	gpio.GPIO_Mode = GPIO_Mode_AF_PP;
	gpio.GPIO_Pin = GPIO_Pin_3;
	GPIO_Init(GPIOA, &gpio);

/* Конфигурируем канал ШИМ */
	TIM_OCInitTypeDef timer_oc; // структура, хранящая настройки канала ШИМ
	TIM_OCStructInit(&timer_oc);
	timer_oc.TIM_Pulse = DUTY_CYCLE; // значение, загружаемое в регистр сравнения

//Определяем режим таймера:
// 1 - прямой (DUTY_CYCLE определяет длительность уровня лог. 1 на выходе),
// 2 - инвертированный(DUTY_CYCLE определяет длительность уровня лог. 0 на выходе)
	timer_oc.TIM_OCMode = TIM_OCMode_PWM1;

// Определяем состояние выхода сравнения таймера.
/* Указав значение TIM_OutputState_Enable, мы подключаем выход сравнения таймера, состояние которого изменяется при совпадении значений счетного регистра и регистра сравнения, в котором хранится значение DUTY_CYCLE) к ножке микроконтроллера. Эта инструкция сводится к тому, что в регистре CCER (capture/compare enable register) устанавливается в 1 бит CC4E (capture/compare 4 output enable) */
	timer_oc.TIM_OutputState = TIM_OutputState_Enable;
	timer_oc.TIM_OCPolarity = TIM_OCPolarity_High;
	TIM_OC4Init(TIM5, &timer_oc); //Инициализируем TIM5 CHANNEL4
	TIM_Cmd(TIM5, ENABLE);
}
```

Упражнение 6. Воспользовавшись кодом, представленным в листинге 3, реализуйте формирование широтно-модулированного сигнала с разной частотой, скважностью, полярностью сигнала.

Упражнение 7. Используя стенд, состоящий из микросхемы Н-моста , двигателя постоянного тока и блока питания для двигателя, реализуйте управление скоростью вращения вала двигателя при помощи ШИМ.

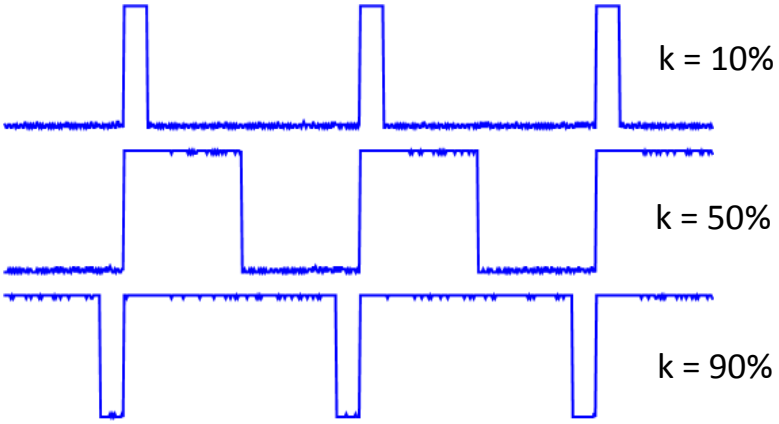

Рис. 9 – Осциллограммы сигналов ШИМ для разных значений коэффициента заполнения k

4) Определение частоты цифрового сигнала в режиме захвата входного сигнала

Для микроконтроллеров серии STM32F107 по умолчанию частота системной шины устанавливается равной 72 МГц. Для работы в режиме захвата входа установим частоту равной 24 МГц. В файле system_stm32f10x.c необходимо раскомментировать строку, соответствующую частоте 24 МГц и закомментировать строку, соответствующую частоте 72 МГц.

```
#if defined (STM32F10X_LD_VL) || (defined STM32F10X_MD_VL) || (defined
STM32F10X_HD_VL)
/* #define SYSCLK_FREQ_HSE HSE_VALUE */
#define SYSCLK_FREQ_24MHz 24000000
#else
/* #define SYSCLK_FREQ_HSE HSE_VALUE */
/* #define SYSCLK_FREQ_24MHz 24000000 */
/* #define SYSCLK_FREQ_36MHz 36000000 */
/* #define SYSCLK_FREQ_48MHz 48000000 */
/* #define SYSCLK_FREQ_56MHz 56000000 */
#define SYSCLK_FREQ_72MHz 72000000
#endif
```

Для демонстрации режима захвата входа будем использовать таймер TIM3. По справочнику определяем, что вход первого канала этого таймера подключен к линии PC6, однако для его активации необходимо произвести переназначение выводов микроконтроллера (remap). Поскольку будут использоваться альтернативные функции линий портов ввода-вывода, необходимо подать тактирование на соответствующие узлы:

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOD |
RCC_APB2Periph_AFIO, ENABLE);
```

Затем создаем и инициализируем структуру для конфигурирования портов ввода-вывода.

```
GPIO_InitTypeDef gpio_cfg;
GPIO_StructInit(&gpio_cfg);
```

Для настройки входа таймера необходимо выполнить следующий код

```
GPIO_PinRemapConfig(GPIO_FullRemap_TIM3, ENABLE);
gpio_cfg.GPIO_Mode = GPIO_Mode_IN_FLOATING;
gpio_cfg.GPIO_Pin = GPIO_Pin_6;
GPIO_Init(GPIOC, &gpio_cfg);
```

Сигнал, период которого будем определять, будем подавать с линии PC13 (потребуется соединительный провод). Сконфигурируем эту линию на выход:

```
gpio_cfg.GPIO_Mode = GPIO_Mode_Out_PP;
gpio_cfg.GPIO_Pin = GPIO_Pin_13;
GPIO_Init(GPIOC, &gpio_cfg);
```

Вычисленное значение периода будем выводить на порт D. Для этого все линии порта D необходимо сконфигурировать на выход:

```
gpio_cfg.GPIO_Mode = GPIO_Mode_Out_PP;
gpio_cfg.GPIO_Pin = GPIO_Pin_All;
GPIO_Init(GPIOD, &gpio_cfg);
```

Функция конфигурирования таймера приведена ниже

```
void init_timer(void)
{
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
	// Настраиваем предделитель так, чтобы таймер считал миллисекунды
	TIM_TimeBaseInitTypeDef timer_base;
	TIM_TimeBaseStructInit(&timer_base);
	timer_base.TIM_Prescaler = 24000 - 1;
	TIM_TimeBaseInit(TIM3, &timer_base);

	/* Настраиваем захват сигнала:
	- канал: 1
	- счёт: по нарастанию
	- источник: напрямую со входа
	- делитель: отключен
	- фильтр: отключен */
	TIM_ICInitTypeDef timer_ic1;
	timer_ic1.TIM_Channel = TIM_Channel_1;
	timer_ic1.TIM_ICPolarity = TIM_ICPolarity_Rising;
	timer_ic1.TIM_ICSelection = TIM_ICSelection_DirectTI;
	timer_ic1.TIM_ICPrescaler = TIM_ICPSC_DIV1;
	timer_ic1.TIM_ICFilter = 0;
	TIM_ICInit(TIM3, &timer_ic1);

	TIM_ITConfig(TIM3, TIM_IT_CC1, ENABLE); // Разрешаем прерывание по захвату

	TIM_Cmd(TIM3, ENABLE);
	NVIC_EnableIRQ(TIM3_IRQn);
}
```

Здесь, во-первых, настраивается предделитель таймера, во-вторых, параметры режима захвата. Выбирается номер канала таймера-счетчика, событие, при котором регистрируется новый импульс (по нарастанию, по убыванию или и по нарастанию, и по убыванию напряжения на входе). Далее настраиваем тип подключения источника входного сигнала, предделитель (можно выбрать значения 1, 2, 4, 8), фильтр (числовое значение от 1 до 15 определяет один из вариантов выборки (частоту и количество измерений), по которой определяется, совершился ли переход из одного со-

стояния в другое, 0 — фильтр отключен). В конце разрешаем таймеру генерировать прерывания по событию «захват», разрешаем работу таймера и разрешаем прерывания от таймера.

В обработчике прерывания от таймера-счетчика TIM3, во-первых, проверяем флаг источника, и, если этим флагом оказывается TIM_IT_CC1 (прерывание от источника TIM Capture Compare 1 Interrupt source), выполняем подсчет длительности периода входного сигнала. Для этого вначале сбрасываем флаг прерывания, затем сохраняем старое значение регистра захвата и сравнения TIM_CCR1 в переменной capture1, получаем новое значение регистра захвата и записываем в переменную capture2. В случае если предыдущее значение регистра TIM_CCR1 не было считано, а пришел новый запрос на прерывание по захвату, необходимо обработать событие over-capture. Код обработчика прерывания от таймера-счетчика TIM3 приведен ниже.

```
void TIM3_IRQHandler(void)
{
	if (TIM_GetITStatus(TIM3, TIM_IT_CC1) != RESET)
	{
		TIM_ClearITPendingBit(TIM3, TIM_IT_CC1);
		/* Запоминаем предыдущее измерение и считываем текущее */
		capture1 = capture2;
		capture2 = TIM_GetCapture1(TIM3);
		if (!capture_is_first)
			capture_is_ready = 1;
		capture_is_first = 0;
		/* Здесь обрабатываем событие over-capture */
		if (TIM_GetFlagStatus(TIM3, TIM_FLAG_CC1OF) != RESET)
		{
			TIM_ClearFlag(TIM3, TIM_FLAG_CC1OF);
			// ...
		}
	}
}
```

Для переключения линии PC13 через равные промежутки времени будем использовать сигнал SysTick. Устанавливаем тактирование системного таймера от шины AHB (в рассматриваемой ситуации частота этой шина равна 24 МГц). Для этого необходимо установить бит CLKSOURCE регистра STK_CTRL (устанавливается при помощи маски SysTick_CLKSource_HCLK). Далее нужно разрешить работу системного таймера, установив бит ENABLE регистра STK_CTRL (устанавливается при помощи маски SysTick_CTRL_ENABLE) и настроить генерацию запроса на обработку исключения SysTick каждый раз, когда счетный регистр системного таймера достигает значения нуля — для этого нужно установить бит TICKINT регистра STK_CTRL, воспользовавшись маской SysTick_CTRL_TICKINT. Таймер SysTick представляет собой реверсивный счетчик, начальное значение его счётного регистра копируется из регистра STK_LOAD. Таким образом, конфигурирование системного счетчика выглядит следующим образом:

```
SysTick->CTRL |= SysTick_CLKSource_HCLK | SysTick_CTRL_ENABLE | SysTick_CTRL_TICKINT;
SysTick->LOAD = 24000 - 1;
```

Согласно этим настройкам, при тактовой частоте 24 МГц исключение SysTick будет возникать с частотой 1 кГц. В обработчике события выпол- няется наращивание переменной systick_ms:

```
void SysTick_Handler(void)
{
	++systick_ms;
}
```

Ниже приведен основной цикл программы. Каждые 1000 миллисекунд меняем уровень на выводе PC13 на противоположный, так что период им- пульсов будет равен 2000 мс. Если флаг capture_is_ready установлен, за- прещаем прерывания от таймера, вычисляем время между соседними фронтами сигнала и выводим на набор светодиодов, подключенных к пор- ту D, после чего вновь разрешаем прерывания.

```
while (1)
{
	static uint32_t toggle_ms = 0;

	if (uint16_time_diff(systick_ms, toggle_ms) >= 1000)
	{
		toggle_ms = systick_ms;
		GPIO_Write(GPIOC, GPIO_ReadOutputData(GPIOC) ^ GPIO_Pin_13);
	}

	if (capture_is_ready)
	{
		NVIC_DisableIRQ(TIM3_IRQ
		n);capture_is_ready = 0;
		const uint16_t period = uint16_time_diff(capture2, capture1);
		// ...
		GPIOD->ODR = period;
		NVIC_EnableIRQ(TIM3_IRQn
		);
	}
}
```

Ниже приведена функция, вычисляющая разность во времени с учётом возможного переполнения таймера

```
uint16_t uint16_time_diff(uint16_t now, uint16_t before)
{
	return (now >= before) ? (now - before) : (UINT16_MAX - before + now + 1);
}
```

Упражнение 8. Используя фрагменты программного кода, приведен- ного выше, сформируйте прямоугольные импульсы на выходе PC13 и из- мерьте их частоту, используя режим захвата таймера.

Список использованных источников

1. Конченков В.И., Скакунов В.Н. Семейство микроконтроллеров STM32. Про- граммирование и применение: учеб. пособие – ВолгГТУ. – Волгоград, 2015. – 77 с.

2. Джозеф Ю. Ядро Cortex-M3. Полное руководство/пер. с англ. А.В. Евстифеева. – М.: Додэка-XXI, 2012. – 553 с.: ил. – (Мировая электроника).

3. Техническая документация на микроконтроллеры серий STM32F105x, STM32F107x [Электронный ресурс]. - Режим доступа : http://www.st.com/resource/en/datasheet/cd00220364.pdf. – Дата обращения 01.12.2019 г.

4. Стандартная библиотека периферии [Электронный ресурс]. – Режим доступа : http://www.st.com/content/st_com/en/products/embedded-software/mcus-embedded-softwa- re/ stm32-embedded-software/stm32-standard-peripheral-libraries/stsw-stm32054.html. – Да-та обращения 01.12.2019 г.

5. Reference manual STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM-based 32-bit MCUs [Электронный ресурс]. – Режим доступа : http://www.st.com/content/ccc/resource/technical/ document/reference_manual/59/b9/ba/7f/11/af/43/d5/CD00171190.pdf/files/CD00171190.pdf/jcr:content/translations/en.C D00171190.pdf/. - Дата обращения 01.12.2019 г.

6. Торгаев С.Н. Практическое руководство по программированию STM-микро- контроллеров : учеб. пособие / С.Н. Торгаев, М.В. Тригуб, И.С. Мусоров, Д.С. Черти- хина ; Томский политехнический университет. – Томск : Изд-во Томского политехни- ческого университета, 2015. – 111 с.