Министерство образования и науки
Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра конструирования узлов и деталей радиоэлектронной аппаратуры
(КУДР)

А.А. Бомбизов, Е.И. Тренкаль

CoIDE. ТАЙМЕРЫ

Методические указания к выполнению
лабораторной и самостоятельной работы
по дисциплине «Микропроцессорные устройства»

Томск 2017

1 Введение

В настоящий момент в рамках курса «микропроцессорные устройства» "освоены такие блоки, как порты ввода-вывода (GPIO), контроллер приоритетных векторных прерываний (NVIC) и контроллер внешних прерываний (EXTI). Это необходимый минимум, позволяющий организовать взаимодействие микроконтроллера с внешней средой. Для понимания логики работы микропроцессорного ядра и устройства микроконтроллера все работы выполнялись с использованием языка ассемблер, который малопригоден для разработки сложного и объемного программного обеспечения.

Целью настоящей работы является освоение принципов построения программы для микроконтроллера на языке Си с использованием среды разработки программного обеспечения CoIDE. Помимо этого, в рамках данной работы необходимо освоить периферийный блок – таймер.

2 Краткая теория

Существует различные средства разработки программного обеспечения для микроконтроллеров на ядре cortex. Среди них можно выделить:

ARMmbed – облачная среда разработки программного обеспечения;

Keil uVision – эта IDE (Integrated Development Environment, интегрированная среда разработки) является одной из самых мощных и массовых не только для ARM, но и для C51 и ряда других процессорных ядер. Помимо этого, в среде имеется виртуальный симулятор и ряд других средств. Недостатками является то, что данное средство доступно только под операционную систему Windows и к тому же является платным – от 1300 евро за один год использования и только для ядер cortex-m. Более серьезный вариант – 8000–9000 евро. В общем, копейки для Билла Гейтса.

CooCox CoIDE – бесплатный инструмент для ARM Cortex.

2.1 Создание проекта в CoIDE

При первом запуске среды разработки CooCox CoIDE появится окно, изображенное на рисунке 1.

Здесь на выбор предлагается 5 действий:

- *Browse in Repository* – это обширная библиотека, которая содержит большое количество исходных кодов для работы с периферией, драйвера и примеров работы.

- *Create a New Project* – создание нового проекта, предварительно выбирая тип микроконтроллера или тип отладочной платы.

- *Open a Project* – открыть существующий проект.

- *User Guide* – инструкция по работе с программой.

- *Forum* – форум на сайте разработчика, где вы можете оставить свой вопрос или поделиться своими идеями или наблюдениями.

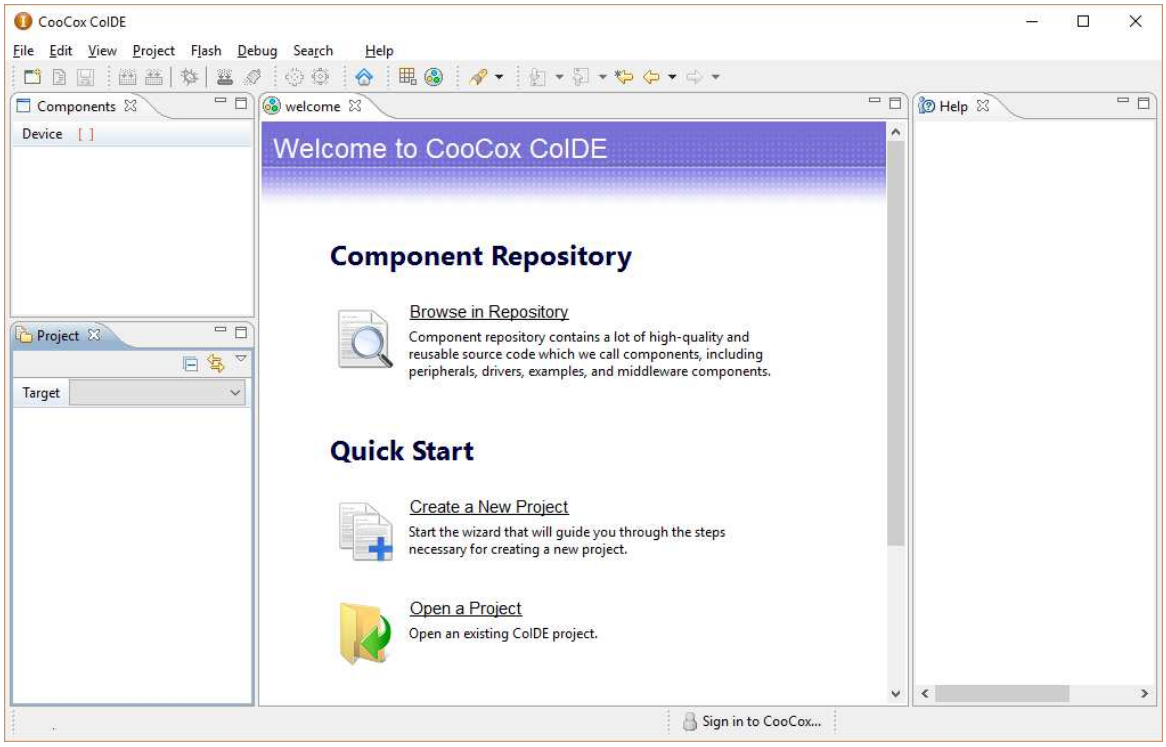

Рисунок 1 – Стартовое окно

Для создания нового проекта необходимо нажать *Create a New Project*, после чего будет отображено окно выбора местоположения и названия нового проекта (рисунок 2).

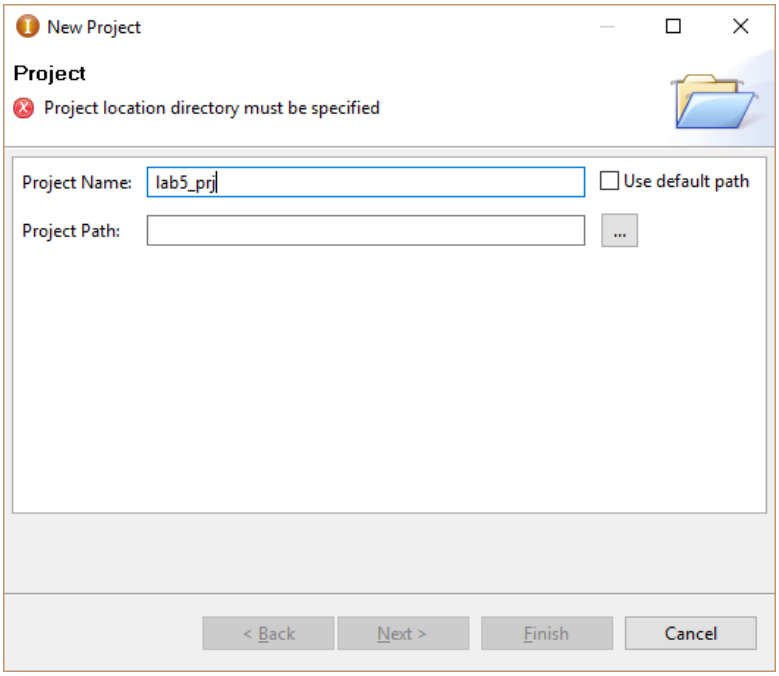

Рисунок 2 – Окно выбора местоположения и названия нового проекта

В данном окне для определения местоположения хранения нового проекта необходимо снять галочку с пункта Use default path и указать путь в поле Project path.

После нажатия на кнопку *Next* будет предложено выбрать основу создания проекта: на базе отдельного чипа (*Chip*) или отладочной платы (*Board*)(рисунок 3). В составе используемой среды программирования отсутствует заготовка для отладочной платы STM32F429I-DISC1, поэтому в данной работе за основу будет взят чип.

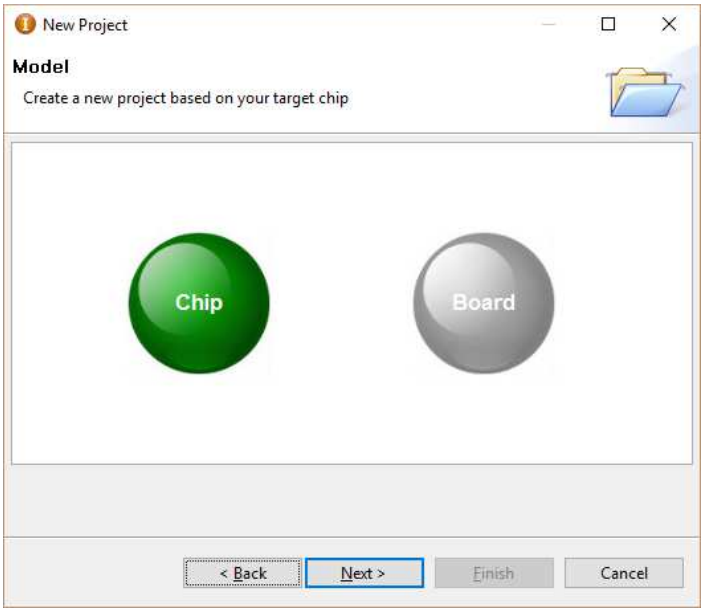

Рисунок 3 – Выбор основы проекта

В следующем окне (рисунок 4) предлагается выбрать используемый микроконтроллер (название используемого микроконтроллера можно увидеть на обороте отладочной платы).

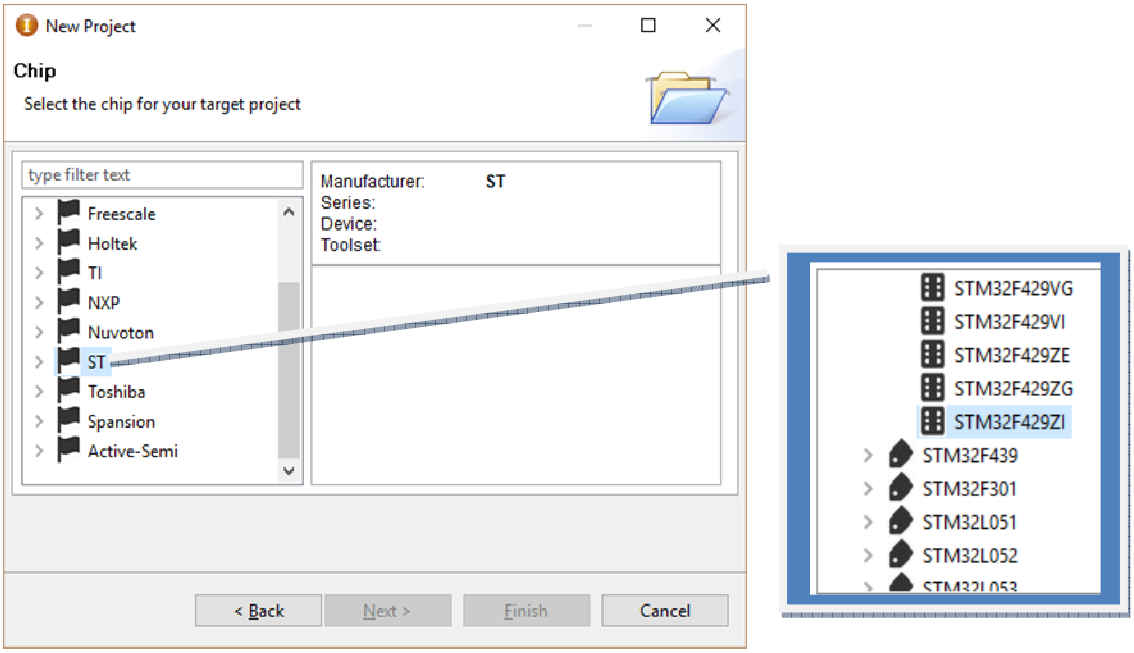

Рисунок 4 – Выбор используемого процессора

Затем необходимо подключить специальною библиотеку Cube Library, в которой размещено полное описание микроконтроллера STM32F429 (рисунок 6).

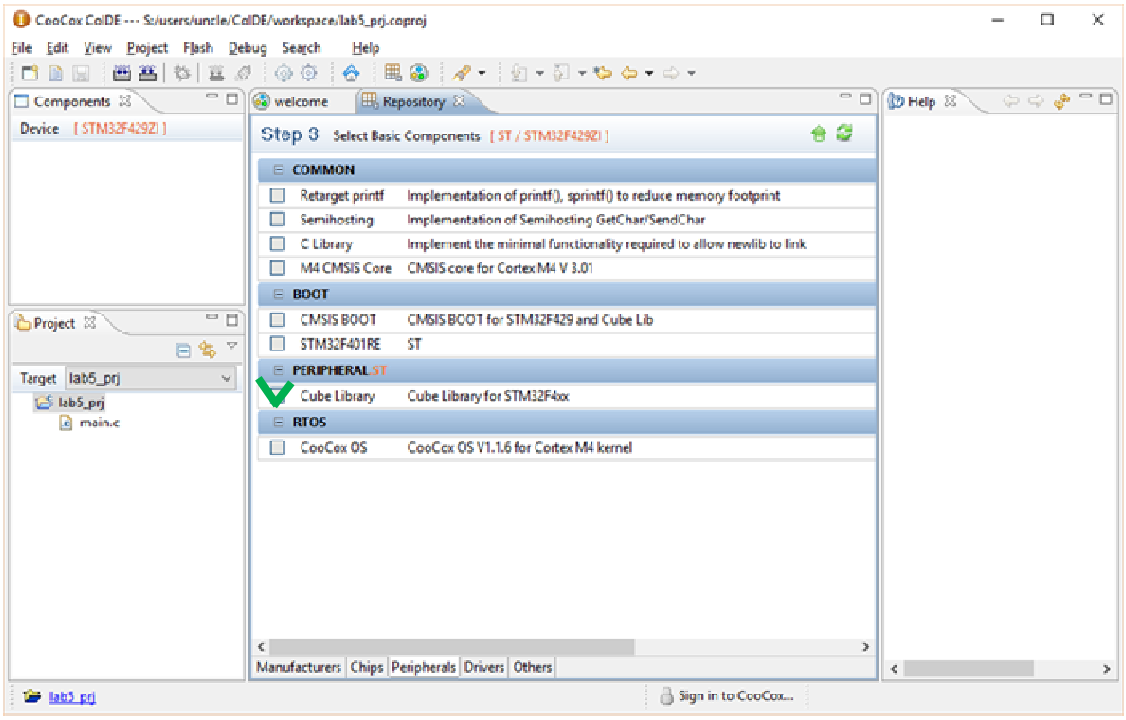

Рисунок 5 – Подключение Cube Library

В результате проделанных действий в созданный проект будут добавлены все необходимые файлы, необходимые для реализации проекта.

2.2 Специфика кода на языке Си

Первое, на что стоит обратить внимание, это файл startup_stm32f429xx.S

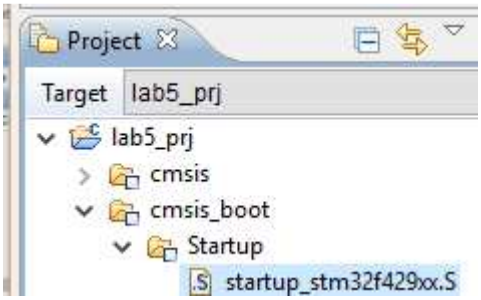

Рисунок 6 – startup_stm32f429xx.S

В нем содержится код запуска микроконтроллера на языке ассемблер, который имеет некоторые отличия от того, что было реализовано в рамках работы №4. Важно отметить, что в этом файле уже полностью определена таблица прерываний с указанием меток на обработчики (рисунок 7). Поэтому в программе, написанной на языке Си при описании обработчика прерывания необходимо использовать название из таблицы следующим образом:

```
void EXTI0_IRQHandler()
{
    //код обработчика прерывания
}
```

где название функции обработчика *EXTI0_IRQHandler* взято из таблицы прерываний (рисунок 7).

```
148 g_pfnVectors:
149   .word  _eram
150   .word  Reset_Handler
151
152   .word  NMI_Handler
153   .word  HardFault_Handler
154   .word  MemManage_Handler
155   .word  BusFault_Handler
156   .word  UsageFault_Handler
157   .word  0
158   .word  0
159   .word  0
160   .word  0
161   .word  SVC_Handler
162   .word  DebugMon_Handler
163   .word  0
164   .word  PendSV_Handler
165   .word  SysTick_Handler
166
167   /* External Interrupts */
168   .word      WWDG_IRQHandler                 /* Window WatchDog              */
169   .word      PVD_IRQHandler                  /* PVD through EXTI Line detection */
170   .word      TAMP_STAMP_IRQHandler           /* Tamper and TimeStamps through the EXTI line */
171   .word      RTC_WKUP_IRQHandler             /* RTC Wakeup through the EXTI line */
172   .word      FLASH_IRQHandler                /* FLASH                        */
173   .word      RCC_IRQHandler                  /* RCC                          */
174   .word      EXTI0_IRQHandler                /* EXTI Line0                   */
```

Рисунок 7 – Таблица прерываний

Анализируя файл запуска *startup_stm32f429xx.S*, нужно отметить, что внутри обработчика *Reset_Handler* (в предыдущих работах использовалась для этого исключения метка *Start*) происходит переход на метку *main*.

```
120 /* Call the application's entry point.*/
121   bl  main
122   bx  lr
```

Рисунок 8 – Переход на метку *main*

Далее в панели *Project* необходимо выбрать файл *main.c*, после чего в рабочей области среды разработки будет отображен начальный код с функцией *main* (Рисунок 9), переход на которую был выполнен в *startup_stm32f429xx.S*. Именно здесь начинается программа на языке Си. Нужно отметить, что в функции уже создан пока еще пустой бесконечный цикл.

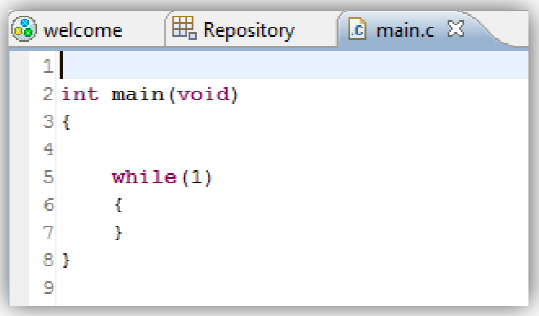

Рисунок 9 – Окно редактирования кода

После добавления *Cube Library* в проекте появились множество файлов с описанием необходимых функций и констант для организации доступа к средствам микроконтроллера. Среди них есть *STM32F429xx.h* с описанием всех базовых адресов и смещений периферийных регистров (подключение в основной код программы осуществляется путем записи в начале файла *main.c* строчки *#include "stm32f429xx.h").* Специфика языка Си позволяет организовать гораздо более простой и удобный способ доступа к периферийным блокам, чем ассемблер.

В языке Си существует тип данных – структура, которая описывается следующим образом:

```
typedef struct
{
  __IO uint32_t MODER;    /*!< GPIO port mode register,               Address offset: 0x00      */
  __IO uint32_t OTYPER;   /*!< GPIO port output type register,        Address offset: 0x04      */
  __IO uint32_t OSPEEDR;  /*!< GPIO port output speed register,       Address offset: 0x08      */
  __IO uint32_t PUPDR;    /*!< GPIO port pull-up/pull-down register,  Address offset: 0x0C      */
  __IO uint32_t IDR;      /*!< GPIO port input data register,         Address offset: 0x10      */
  __IO uint32_t ODR;      /*!< GPIO port output data register,        Address offset: 0x14      */
  __IO uint16_t BSRRL;    /*!< GPIO port bit set/reset low register,  Address offset: 0x18      */
  __IO uint16_t BSRRH;    /*!< GPIO port bit set/reset high register, Address offset: 0x1A      */
  __IO uint32_t LCKR;     /*!< GPIO port configuration lock register, Address offset: 0x1C      */
  __IO uint32_t AFR[2];   /*!< GPIO alternate function registers,     Address offset: 0x20-0x24 */
} GPIO_TypeDef;
```

где *typedef struct* – это начало записи структуры;

GPIO_TypeDef – имя структуры;

{всё что между скобками} – поля данных структуры.

В языке Си существует понятие указателя на ячейку памяти. В программе указатель может быть объявлен и проинициализирован следующим образом:

```
GPIO_TypeDef * GPIOG= 0x40021800;
```

Где *GPIOG* – переменная-указатель, с адресом 0x40021800.

Поля в структуре *GPIO_TypeDef* имеют тип данных *uint32_t*, то есть данные размером 32 бита или 4 байта. Поле *MODER* является первым в структуре и поэтому размещено в памяти по адресу 0x40021800. Каждое следующее поле смещено относительно предыдущего на размер типа данных, то есть на четыре байта. Поэтому структура, записанная с использование языка Си, автоматически реализует смещения всех регистров. Для сравнения на ассемблере записывались константы для каждого регистра.

Все блоки микроконтроллера описаны в виде указателей на соответствующие им структуры. Имя каждого указателя повторяет наименование блока.

Доступ к регистру на языке Си осуществляется следующим образом

RCC->

после чего будет отображено окно с возможностью выбора необходимого регистра (рисунок 10). Вызов подсказки выполняется нажатием комбинации клавиш CTRL+пробел.

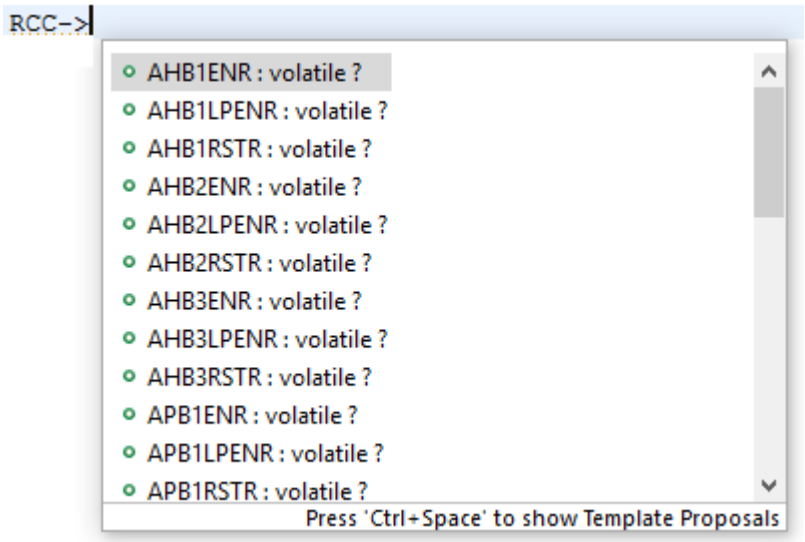

Рисунок 10 – Доступ к регистру

где *RCC* – наименование периферийного устройства в микроконтроллере;

«->» –операция разыменования указателя.

Команда

REG|=1<<X;

позволяет выставить в единицу бит *X* в регистре *REG* («|» – это операция ИЛИ).

Команда

REG&=~(1<<X);

сбрасывает бит *X* в регистре REG в ноль(«&» – это операция И).

Пример использования операции:

*RCC->AHB1ENR|=1<<6;*Устанавливает в единицу шестой бит в регистре *RCC->AHB1ENR*.

В *STM32F429xx.h* описаны все необходимые маски в виде констант (макросы) типа: *#define RCC_AHB1ENR_GPIOGEN ((uint32_t)0x00000040)*, содержащие значения целевого бита в регистре. К примеру, *1<<6*=0x00000040 GPIOGEN – соответствует названию бита в [1].

Поэтому операция включения тактирование порта *G* может быть записана следующим образом:

RCC->AHB1ENR|=RCC_AHB1ENR_GPIOGEN;

Выглядит поизящней, чем запись на ассемблере:

.equ *RCC_AHB1ENR, 0x40023800+0x30*

.equ *RCC_AHB1ENR_GPIOGEN, 1<<6*

…….…

LDR R0,= RCC_AHB1ENR

LDR R1,[R0]
LDR R2, =RCC_AHB1ENR_GPIOGEN
ORR R1,R2
STR R1,[R0]

2.3 Таймеры

В любом микроконтроллере после получения базовых знаний о портах ввода-вывода приходит закономерное желание выдерживать (отсчитывать, измерять) определенные интервалы времени. Для этого используются таймеры.

В STM32F429 существует 14 таймеров, которые подходят под определенную классификацию:

Basic Timers – это самые простые таймеры, имеющие возможность только отсчитывать промежуток времени и генерировать прерывание по его истечении. В микроконтроллерах STM32 это, обычно, таймеры TIM6/TIM7.

General-Purpose Timers – это таймеры общего назначения (универсальные таймеры). В STM32F429 это TIM2, TIM5 (32-битный) и TIM3, TIM4 (16-битный), обеспечивающие прямой, обратный и двунаправленный счёт, а также TIM9–TIM14 (16-битные), обеспечивающие только прямой счёт и имеющие меньшую функциональность.

Advanced-Control Timer – расширенный таймер (таймер улучшенного управления), помимо всех функций, доступных для General-Purpose Timers, позволяет также управлять электродвигателем. В STM32F429 это TIM1 и TIM8 (16-битный).

Особенностью таймеров общего назначения является также то, что на каждый таймер приходится блок из четырёх независимых каналов захвата/сравнения. Этот блок (т.е. каждый из четырёх каналов) позволяет также выполнять генерацию ШИМ (широтно-импульсная модуляция) и генерировать одиночный импульс на выходе порта ввода-вывода.

Таймеры позволяют генерировать прерывание при возникновении следующих событий:

- переполнение/обнуление или инициализация счётчика;
- при возникновении таких событий, как пуск, остановка, инициализация или счёт счётчика от внешнего/внутреннего сигнала;
- при захвате по входу;
- при сравнении.

Также имеется возможность подключить инкрементный энкодер и датчик Холла.

За работу таймеров отвечают следующие регистры.

Базовые регистры таймера-счётчика (*x* = 2 или 3, в зависимости от таймера):

- счётный регистр (*TIMx_CNT*);
- регистр предделителя (*TIMx_PSC*);
- регистр автоперезагрузки (*TIMx_ARR*).

Есть ещё множество дополнительных регистров, отвечающих за настройку и проверку статуса работы таймера и его остальных блоков:

- TIMx control register 1 (*TIMx_CR1*) – 1-й регистр управления.
- TIMx control register 2 (*TIMx_CR2*) – 2-й регистр управления.
- TIMx slave mode control register (*TIMx_SMCR*) – регистр управления подчинённым режимом (выбор режима Master/Slave, режима (источника) тактирования, настройка параметров запуска, выбор режима энкодера).
- TIMx DMA/Interrupt enable register (*TIMx_DIER*) – регистр разрешения прерываний/DMA.
- TIMx status register (*TIMx_SR*) – регистр статуса таймера TIMx. Здесь устанавливаются различные флаги произошедших событий.
- TIMx event generation register (*TIMx_EGR*) – регистр генерации событий.
- TIMx capture/compare mode register 1 (*TIMx_CCMR1*) – 1-й регистр выбора и настройки режимов захват/сравнение (каналы CH1 и CH2).
- TIMx capture/compare mode register 2 (*TIMx_CCMR2*) – 2-й регистр выбора и настройки режимов захват/сравнение (каналы CH3 и CH4).
- TIMx capture/compare enable register (*TIMx_CCER*) – регистр разрешения захвата/сравнения (настройка полярности и разрешение выхода захвата/сравнения).
- TIMx capture/compare register1 (*TIMx_CCR1*) – 1-й регистр захвата/сравнения (сохраняет значение счётного регистра в режиме захвата или же в него записывается нужное значение в режиме сравнения).
- TIMx capture/compare register 2 (*TIMx_CCR2*) – 2-й регистр захвата/сравнения.
- TIMx capture/compare register 3 (*TIMx_CCR3*) – 3-й регистр захвата/сравнения.
- TIMx capture/compare register 4 (*TIMx_CCR4*) – 4-й регистр захвата/сравнения.
- TIMx DMA control register (*TIMx_DCR*) – регистр управления DMA.
- TIMx DMA address for full transfer (*TIMx_DMAR*) – адрес полной пересылки DMA.

Тактирование таймера может осуществляться как от внешнего источника, так и от внутреннего. В данной работе будет рассматриваться

только внутренний источник тактирования. Тактовый сигнал поступает на блок делителя частоты (см. рисунок 11), где может быть замедлен 16-битным делителем *TIMx_PSC*. Допустим, тактовый генератор генерирует сигналы с частотой 8 МГц. Разработчику понадобилась частота изменения счетчика 250 Гц. Выставив в регистре *TIMx_PSC* значение 32000, будет достигнута требуемая частота. Затем преобразованный сигнал поступает на счетчик *TIMx_CNT*, который увеличивает (или изменяет в режиме down) своё значение с каждым пришедшим преобразованным тактовым сигналом.

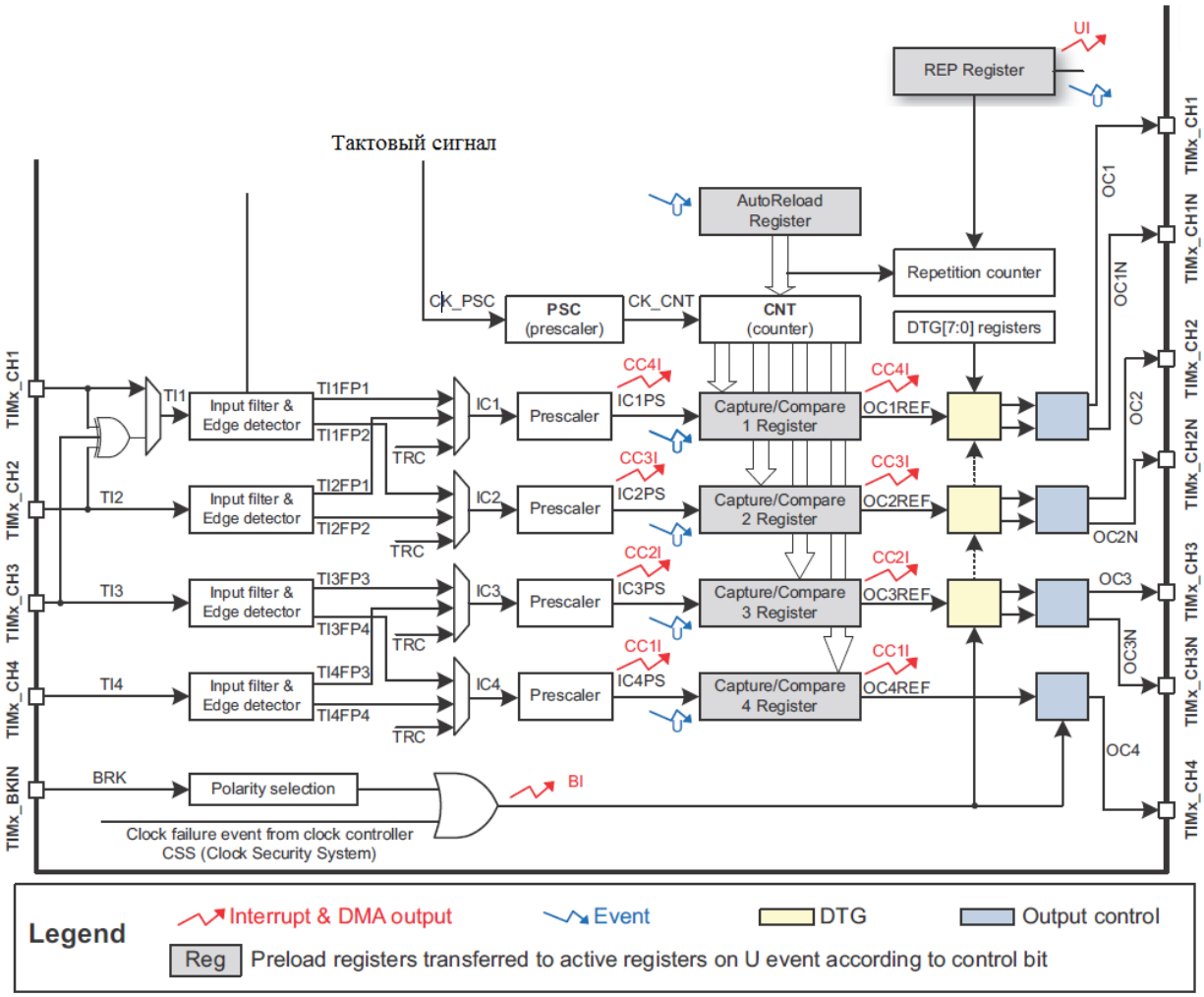

Рисунок 11 – Структура таймера [1]

Счетчик *TIMx_CNT* будет увеличиваться до тех пор, пока не достигнет регистра перезагрузки *TIMx_ARR*. После чего он обнулится (обновится). Это вызовет прерывание *Update interrupt* (*UI*) (см. рисунок 11). Исходя из всего этого, период срабатывания таймера может быть посчитан с использованием следующей формулы:

$$T = \frac{TIMx\_ARR}{\text{Частота Генератора}/(TIMx\_PSC + 1)}$$

В определенных конфигурациях таймера (кроме базовых) значение в счетчике можно сравнивать с регистром сравнения *TIMx_CCRx*, где *x* – номер

канала от 1 до 4. Когда значение счетчика совпадет с регистром сравнения, будет сгенерировано прерывание Capture/Compare x interrupt (*CCxI*). При возникновении этого прерывания значение счетчика не будет сброшено.

Таким образом, при помощи одного таймера можно организовать генерацию прямоугольного сигнала, где период будет задаваться значением в регистре *TIMx_ARR*, а длительность значением в регистре *TIMx_CCRx*. Это не что иное, как широтно-импульсная модуляция. Каждое изменение сигнала должно происходить в обработчике соответствующего прерывания. Например, переключение вывода порта в логическую единицу выполняется в обработчике прерывания *CCxI*, а в логический ноль в обработчике прерывания *UI*. Временная диаграмма работы таймера с генерацией ШИМ-сигнала изображена на рисунке 12.

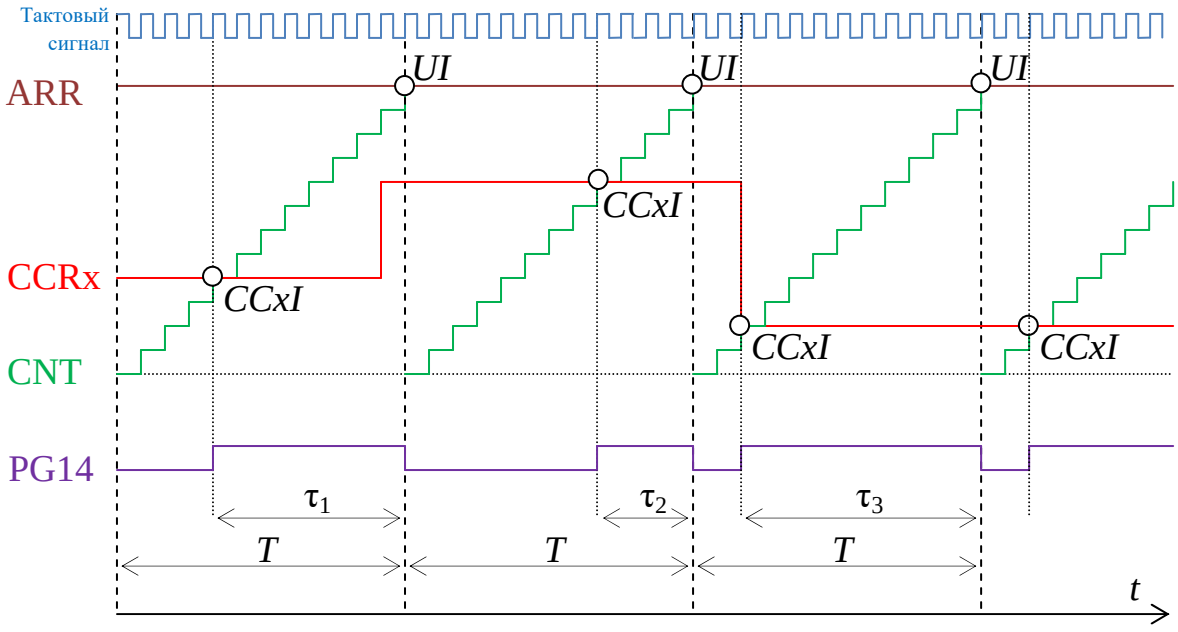

Рисунок 12 – Временная диаграмма работы таймера

3 Порядок выполнения работы

В ходе данной работы будут освоены следующие разделы: 1) работа с регистрами периферийных устройств с использованием языка Си; 2) работа с контроллером внешних прерываний *EXTI*; 3) освоение работы с базовым таймером (**Basic Timer**); 4) освоение работы с таймером общего назначения (**General-Purpose Timers**) на примере генерации ШИМ-сигнала.

Работа с регистрами периферийных устройств с использованием языка Си.

3.1 Изучите предложенный в п. 2 теоретический материал.

3.2 Создайте новый проект как описано в теоретическом материале.

3.3 В функции *main()* сконфигурируйте порты ввода-вывода таким образом, чтобы оба светодиода на отладочной плате зажглись.

3.4 Скомпилируйте проект путём выбора в меню *Project* пункта *Build*.

3.5 Запрограммируйте микроконтроллер путем выбора в меню *Flash* пункта *Program download*.

Работа с контроллером внешних прерываний *EXTI*. По результатам этого этапа красный светодиод должен переключаться по нажатию кнопки.

3.6 Настройте контроллер *EXTI*, чтобы генерировалось прерывание по нажатию кнопки.

3.7 Разрешите прерывание в контроллере приоритетных прерываний NVIC.

3.8 Добавьте в программу функцию – обработчик прерывания (смотрим теоретическую часть).

3.9 Внутри функции обработчика выполните переключение состояния вывода порта, к которому подключен светодиод.

3.10 Внутри функции обработчика выполните выход из режима прерывания.

3.11 Скомпилируйте программу, запрограммируйте микроконтроллер и проверьте работоспособность программы.

Освоение работы с базовым таймером (**Basic Timer**). По результатам данного этапа зеленый светодиод должен мигать с частотой 1 Гц.

3.12 Включите тактирование таймера 6.

3.13 С учетом частоты тактового генератора 16 МГц настройте период срабатывания прерывания *Update interrupt* (*UI*).

3.14 Разрешите тактирование таймера путем выставления бита *CEN* в регистре *TIM6_CR1* (см. описание регистра *TIM6_CR1*).

3.15 Разрешите прерывание таймера путём настройки регистра TIM6_DIER и контроллера NVIC.

3.16 Добавьте в программу функцию – обработчик прерывания таймера 6.

3.17 Внутри функции обработки прерывания выполните переключение зеленого светодиода и выполните выход из режима прерывания (см. регистр *TIM6_SR*).

3.18 Скомпилируйте программу, запрограммируйте микроконтроллер и проверьте работоспособность программы.

Освоение работы с таймером общего назначения (**General-Purpose Timers**) на примере генерации ШИМ-сигнала. В результате данного этапа должна быть реализована программа, которая при запуске генерирует слабое свечение красного светодиода. По нажатию кнопки яркость свечения должна возрастать вдвое.

3.19 Настройте таймер 2 по аналогии с таймером 6.

3.20 Установите период срабатывания равным степени двойки (256, 512, 1024 и т.д.).

3.21 Установите значение регистра *TIM2_CCR1* в единицу.

3.22 Помимо включенного прерывания *UI* включите в таймере 2 прерывание *CC1I*.

3.23 В обработчике прерывания таймера 2 выполните определение типа сработанного прерывания путем проверки битов в регистре *TIM2_SR*.

3.24 Если сработало прерывание *CC1I*, зажгите светодиод и выйдете из режима прерывания *CC1I*.

3.25 Если сработало прерывание *UI*, потушите светодиод и выйдете из режима прерывания *UI*.

3.26 Скомпилируйте программу, запрограммируйте микроконтроллер и проверьте работоспособность программы.

На текущей стадии разработки должен мигать зеленый светодиод с частотой 1 Гц, а красный слабо светиться. Если красный светодиод мигает, то необходимо добиться непрерывного свечения путём уменьшения делителя частоты тактирования таймера.

3.27 В прерывании, возникающем от нажатия кнопки, выполните удвоение регистра сравнения *TIM2_CCR1*. Если значение регистра сравнения превысило *TIM2_ARR*, то необходимо его вернуть в начальное состояние.

3.28 Скомпилируйте программу, запрограммируйте микроконтроллер и проверьте работоспособность программы.

3.29 Оформите отчет, содержащий титульный лист, введение, ход выполнения работы, ответы на контрольные вопросы и выводы.

3.30 Защитите отчет у преподавателя.

4 Контрольные вопросы

4.1 Опишите принцип работы таймера.

4.2 Сколько различных прерываний может сгенерировать таймер?

4.3 Какой регистр таймера отвечает за выход из режима прерывания?

4.4 На чем основан принцип изменения скважности ШИМ-сигнала?

4.5 Как изменить период срабатывания таймера?

Список литературы

1. RM0090. Reference manual. STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced ARM®-based 32-bit MCUs.– URL: www.st.com/resource/en/reference_manual/DM00031020.pdf (дата обращения: 10.01.2017).

2. PM0214. Programming manual. STM32F3, STM32F4 and STM32L4 Series. Cortex®-M4 programming manual.– URL:

www.st.com/resource/en/programming_manual/DM00046982.pdf (дата обращения: 10.01.2017).

3. MB1075. STM32F429I-DISCO schematics.– URL: http://www.st.com/resource/en/schematic_pack/stm32f429i-disco_sch.zip.

4. GNU ARM Assembler Quick Reference.– URL://http://www.ic.unicamp.br/~celio/mc404-2014/docs/gnu-arm-directives.pdf.