Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет радиотехники и электроники

Кафедра информационных радиотехнологий

Т. Н. Дворникова

ВСТРАИВАЕМЫЕ СИСТЕМЫ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для специальностей 1-39 01 02 «Радиоэлектронные системы», 1-39 01 04 «Радиоэлектронная защита информации»

Минск БГУИР 2023

УДК 004.383.3-022.53(076.5)
ББК 32.971.32-04я73
Д24

Р е ц е н з е н т ы:

кафедра инфокоммуникационных технологий
учреждения образования «Белорусская государственная академия связи»
(протокол №3 от 12.10.2022);

профессор кафедры бизнес-анализа и математического моделирования
учреждения образования федерации профсоюзов Беларуси
«Международный университет «МИТСО»
кандидат физико-математических наук,
доцент Г. Е. Хурсевич

Дворникова, Т. Н.
Д24 Встраиваемые системы. Лабораторный практикум : учеб.-метод. пособие / Т. Н. Дворникова. – Минск : БГУИР, 2023. – 160 с. : ил.
ISBN 978-985-543-703-2.

Состоит из 14 лабораторных работ и предназначено для изучения встраиваемых систем и получения практических навыков измерения их основных параметров и характеристик, а также приобретения практических навыков физического макетирования и исследования схем встраиваемых систем на лабораторных стендах. Содержит теоретические сведения по проектированию и применению наиболее распространенных узлов встраиваемых систем.

УДК 004.383.3-022.53(076.5)
ББК 32.971.32-04я73

ISBN 978-985-543-703-2

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Встраиваемые системы – специализированная микропроцессорная система управления, контроля и мониторинга, концепция разработки которой заключается в том, что такая система будет работать, будучи встроенной непосредственно в устройство, которым она управляет.

Встраиваемые системы имеют широкое применение в разработке телекоммуникационного оборудования, банковских автоматов, станков с числовым программным управлением, средствами управления технологическими процессами и их регулированием. Сегодня встраиваемые системы управляют широким кругом устройств. Вариация использования начинается от портативных устройств, например, бытовая техника и часы, до сложнейших стационарных систем, использующихся на заводах и различных производствах.

Встраиваемые системы создаются на базе микроконтроллеров, микропроцессоров, а также программируемых логических интегральных схем. В настоящее время на рынке представлено большое количество самых разнообразных микроконтроллеров.

Лабораторный практикум содержит комплекс работ по созданию проектов для встраиваемых систем на основе микроконтроллеров компании STMicroelectronics STM32F4 с использованием отладочной платы STM32F4 Discovery, WT32-S1 фирмы Wireless-tag: микроконтроллер ESP-32 и FPGA компании Altera.

Кроме того, в приложениях приводится информация относительно используемой интегрированной среды разработки для микроконтроллеров и ПЛИС, а также примеры программных кодов.

Для изучения встраиваемых систем и получения практических навыков физического макетирования в ходе лабораторных работ предлагается выполнить моделирование схем с использованием пакета программного обеспечения для проектирования автоматизированных электронных схем, моделирования схем Proteus Professional 8.

Для выполнения лабораторных работ предпочтительно знание основ теории цифровой схемотехники, разработки программного обеспечения и алгоритмизации, а также знание языков программирования C, C++, Blockly (UIFlow), Arduino и Micropython.

I. ЛАБОРАТОРНЫЕ РАБОТЫ (STM32)

Лабораторная работа №1

СОЗДАНИЕ ПРОЕКТА В СРЕДЕ РАЗРАБОТКИ. ИСПОЛЬЗОВАНИЕ ПОРТОВ ВВОДА/ВЫВОДА

Цель работы: ознакомиться с созданием проектов для платы, рассмотреть их структуру; изучить принципы работы с портами ввода/вывода и организации их взаимодействия.

Оборудование и программное обеспечение: плата STM32F4 Discovery, среда разработки CooCox CoIDE 1.7, инструменты построения проекта GNU Toolchain for ARM Embedded Processors, библиотека CMSIS при необходимости самостоятельного подключения файлов к проекту.

Теоретический материал

Контроллер STM32F407VG содержит пять 16-разрядных портов ввода/вывода общего назначения, которые обозначены как GPIOx, где x может иметь значения A, B, C, D, E [13]. Каждый порт GPIO имеет четыре 32-битных регистра конфигурации (GPIOx,_MODER, GPIOx_TYPER, GPIOx_SPEEDR, GPIOx_ORD), два 32-битных регистра данных (GPIOx_ODR, GPIOx_IDR) и два 32-битных регистра выбора дополнительных функций (GPIOx_AFRH и GPIOx_AFRL).

Светодиоды, предназначенные для программирования, на плате подключены к порту D (рисунок 1.1).

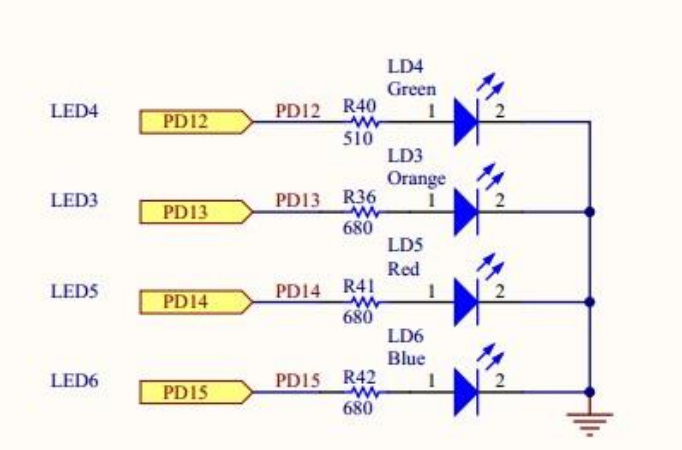

Рисунок 1.1 – Схема подключения светодиодов к порту на плате

Остальные четыре светодиода выполняют служебные функции индикации и в программировании для определенных действий не используются.

Создание проекта и его структура

Для выполнения лабораторных работ рекомендуется установить среду разработки CooCox CoIDE 1.7. Создание проекта выполняется с помощью мастера, который открывается при выполнении команды меню Project/New Project. В мастере следует пошагово указать имя проекта и его расположение, производителя и МК, для которого предназначена программа. Далее работа с проектом осуществляется с помощью окна Repository, которое позволяет добавить необходимые библиотеки управления периферийными частями МК, а также окна Project. Окно Repository также представляет собой мастер, содержащий несколько вкладок, основной из которых является вкладка Peripherals (рисунок 1.2).

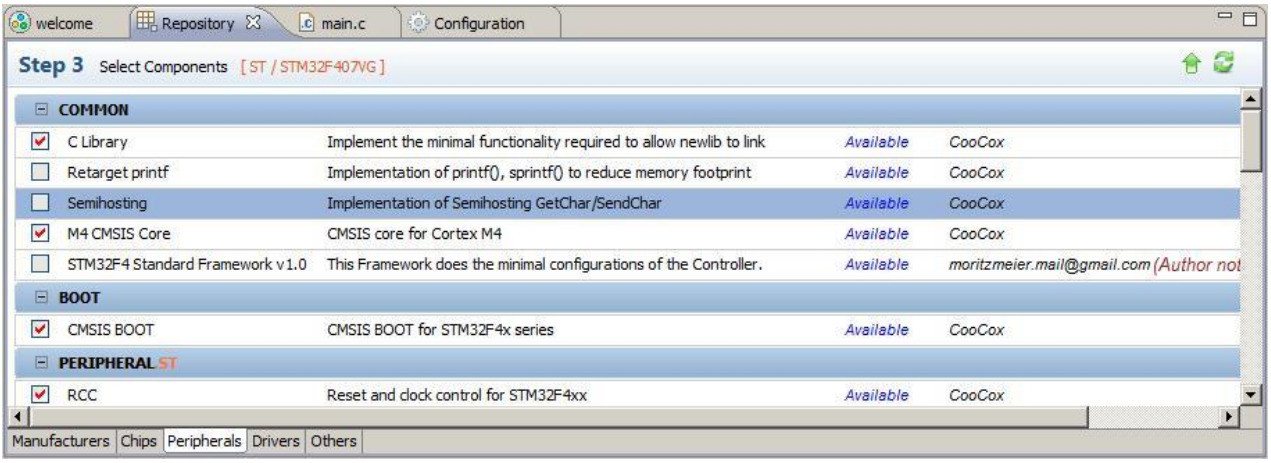

Рисунок 1.2 – Окно Repository

Проект программы для платы STM32F4 Discovery имеет схожую структуру для всех средств разработки [6]. Обычно он включает в себя два внутренних каталога: один – для файлов из библиотеки CMSIS, а другой – для файлов, предназначенных для работы с периферией. Для примера представлены типичные структуры проектов в средах разработки CoIDE и Keil (рисунок 1.3).

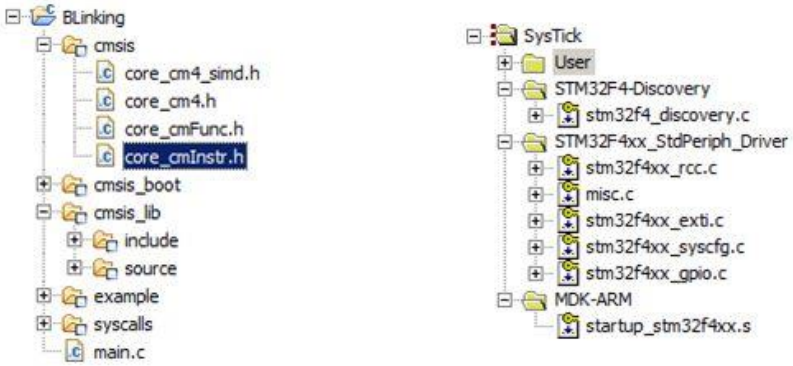

Рисунок 1.3 – Структуры проектов в разных средах разработки

Пример программы

Рассмотрим пример программы, которая демонстрирует работу с портами ввода/вывода. Она позволяет переключать состояния светодиода при нажатии пользовательской кнопки, которая находится на плате.

Листинг кода:

```
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"

void Delay(uint32_t nCount)
{

      while(nCount--)

      {
      }
}

int main(void)
{
      GPIO_InitTypeDef gpioConf;
      // инициализация входа, подключенного к кнопке
      RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
      gpioConf.GPIO_Pin = GPIO_Pin_0;
      gpioConf.GPIO_Mode = GPIO_Mode_IN;
      GPIO_Init(GPIOA, &gpioConf);

      RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
      // инициализация входа, подключенного к светодиоду
      gpioConf.GPIO_Pin = GPIO_Pin_13;
      gpioConf.GPIO_Mode = GPIO_Mode_OUT;
      gpioConf.GPIO_Speed = GPIO_Speed_100MHz;
      gpioConf.GPIO_OType = GPIO_OType_PP;
      gpioConf.GPIO_PuPd = GPIO_PuPd_NOPULL;

      GPIO_Init(GPIOD, &gpioConf);

      while(1) {

            if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0) == 0) {
                  if (GPIO_ReadOutputDataBit(GPIOD, GPIO_Pin_13))
                        GPIO_ResetBits(GPIOD, GPIO_Pin_13);
                  else
                        GPIO_SetBits(GPIOD, GPIO_Pin_13);
                  Delay(5000);
                  while(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0)==0);
                  Delay(5000);
```

```
                }
            }
        }
```

Ход работы

1. На основе кода примера приложения создать свой проект в среде разработки и проверить его работоспособность.
2. Ознакомиться с работой используемых функций, просмотрев исходный код, а также комментарии в исходных файлах библиотеки и в режиме отладки.
3. Создать новый проект в среде разработки для выполнения индивидуального задания, полученного от преподавателя.
4. Реализовать необходимую функциональность.
5. Запрограммировать плату и продемонстрировать работу программы.

Индивидуальное задание

В соответствии с вариантом из таблицы 1.1 реализовать схему включения/выключения светодиодов, расположенных на плате.

Таблица 1.1 – Варианты включения светодиодов

Вариант	Первый светодиод	Второй светодиод	Третий светодиод	Четвертый светодиод
1	1	2	3	4
2	2	1	3	4
3	2	3	1	4
4	2	3	4	1
5	2	4	3	1
6	4	2	1	3
7	3	4	2	1
8	4	3	1	2
9	4	1	3	2
10	1	4	2	3

Номера включения светодиодов присваиваются по их номерам в составе порта D (можно найти в документации).

Лабораторная работа №2

ПРЕРЫВАНИЯ И ИХ ИСПОЛЬЗОВАНИЕ. ИСПОЛЬЗОВАНИЕ ТАЙМЕРОВ

Цель работы: ознакомиться с понятием «прерывания» и возможностями, которые они предоставляют; научиться использовать таймеры для выполнения действий в определенные временные интервалы.

Оборудование и программное обеспечение: плата STM32F4 Discovery, среда разработки CooCox CoIDE 1.7, инструменты построения проекта GNU Toolchain for ARM Embedded Processors, библиотека CMSIS при необходимости самостоятельного подключения файлов к проекту.

Теоретический материал

Прерывания – механизм, который позволяет аппаратному обеспечению сообщать о наступлении важных событий в своей работе [14]. В момент, когда происходит прерывание, процессор переключается с выполнения основной программы на выполнение соответствующего обработчика прерываний. Как только выполнение обработчика завершено, продолжается выполнение основной программы с места, в котором она была прервана.

Для использования прерываний необходимо вначале настроить регистр, который называется Nested Vector Interrupt Controller (NVIC), а также встроенный контроллер вектора прерываний. Данный регистр является стандартной частью архитектуры ARM и встречается на всех процессорах, независимо от производителя. NVIC разработан таким образом, что задержка прерывания минимальна. NVIC поддерживает встроенные прерывания с 16 уровнями приоритета [7].

Микроконтроллер STM32F407VG содержит 14 таймеров [13]. В общем виде схема управления подсчетом импульсов может быть представлена следующим образом (рисунок 1.4).

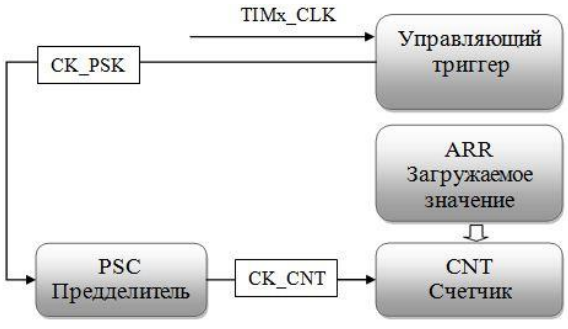

Рисунок 1.4 – Схема управления подсчетом импульсов

Производитель разделяет все таймеры на три типа:

1) с расширенными возможностями;
2) общего назначения;
3) базовые.

Каждый таймер может иметь до четырех линий захвата/сравнения (именно они используются в режиме генерации ШИМ).

Пример программы

Данная программа демонстрирует работу с прерываниями и с таймерами. В ней реализовано переключение светодиода, который подключен к порту ввода/вывода, через определенные интервалы времени.

Листинг кода:

```
#include «stm32f4xx.h»
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_tim.h"
#include "misc.h"

void INTTIM_Config(void);
void GPIO_Config(void);

int main(void)
{
	GPIO_Config();
	INTTIM_Config();

	while(1)
	{
	}
}

void TIM2_IRQHandler(void) {

	if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {

		TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
		GPIOD->ODR ^= GPIO_Pin_13;

	}

}

void GPIO_Config(void) {

	GPIO_InitTypeDef gpio_struct;
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
	gpio_struct.GPIO_Pin = GPIO_Pin_13;
```

```
	gpio_struct.GPIO_Mode = GPIO_Mode_OUT;
	gpio_struct.GPIO_OType = GPIO_OType_PP;
	gpio_struct.GPIO_PuPd = GPIO_PuPd_NOPULL;
	gpio_struct.GPIO_Speed = GPIO_Speed_100MHz;
	GPIO_Init(GPIOD, &gpio_struct);

}

void INTTIM_Config(void)

{

	NVIC_InitTypeDef nvic_struct;
	nvic_struct.NVIC_IRQChannel = TIM2_IRQn;
	nvic_struct.NVIC_IRQChannelPreemptionPriority = 0;
	nvic_struct.NVIC_IRQChannelSubPriority = 1;
	nvic_struct.NVIC_IRQChannelCmd = ENABLE;

	NVIC_Init(&nvic_struct);

	TIM_TimeBaseInitTypeDef tim_struct;
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
	tim_struct.TIM_Period = 10000 - 1;
	tim_struct.TIM_Prescaler = 168 - 1;
	tim_struct.TIM_ClockDivision = 0;
	tim_struct.TIM_CounterMode = TIM_CounterMode_Up;
	TIM_TimeBaseInit(TIM2, &tim_struct);
	TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
	TIM_Cmd(TIM2, ENABLE);
}
```

Обращает на себя внимание включение дополнительного заголовочного файла – misc.h. Именно в нем описаны функции, перечисления, структуры для работы с прерываниями.

Ход работы

1. На основе кода примера приложения создать свой проект в среде разработки и проверить его работоспособность.
2. Ознакомиться с работой используемых функций, просмотрев исходный код, а также комментарии в исходных файлах библиотеки и в режиме отладки.
3. Создать новый проект в среде разработки для выполнения индивидуального задания, полученного от преподавателя.
4. Реализовать необходимую функциональность.
5. Запрограммировать плату и продемонстрировать работу программы.

Индивидуальные задания

1. С помощью одного из таймеров общего назначения сгенерировать задержку длительностью 1 с (на основе данных о рабочей частоте). Задержку генерировать на основе прерываний таймера. Продемонстрировать расчеты, подтверждающие правильность задания задержки.
2. Реализовать с помощью прерывания по переполнению таймера временную задержку с поочередным включением светодиодов на плате по кругу.

Лабораторная работа №3

ГЕНЕРАЦИЯ СИГНАЛОВ ШИМ

Цель работы: ознакомиться с возможностями генерации ШИМ с помощью демонстрационной платы; научиться генерировать сигнал с заданными параметрами и использовать его для управления внешними устройствами.

Оборудование и программное обеспечение: плата STM32F4 Discovery, среда разработки CooCox CoIDE 1.7, инструменты построения проекта GNU Toolchain for ARM Embedded Processors, библиотека CMSIS при необходимости самостоятельного подключения файлов к проекту.

Теоретический материал

Широтно-импульсная модуляция (ШИМ) – способ управления средним значением напряжения на нагрузке путем изменения скважности импульсов, управляемых ключом. В основном микроконтроллеры позволяют генерировать цифровую ШИМ различной частоты.

Поскольку вывод сигнала ШИМ не является основной функцией пинов, которые подключены к светодиодам, то необходимо выполнить их соответствующую конфигурацию. Для этого следует задать выполнение пинами альтернативных функций. Генерация ШИМ в них связана с использованием дополнительных режимов таймера.

Перед подключением устройства известны такие параметры ШИМ, как частота и коэффициент заполнения. Для их расчета в контроллерах STM32 необходимо определить значение предделителя и автоматически загружаемое значение в регистре ARR (Auto-Reload Register). Расчет значения, которое следует записать в предделитель, выполняется следующим образом:

$$\mathrm{PSC} = \frac{\mathrm{TIMxCLK}}{\mathrm{TIMxCNT}} - 1, \qquad (1.1)$$

где PSC – значение предделителя; TIMxCLK – входная частота работы таймера; TIMxCNT – частота счетчика.

Для получения необходимой выходной частоты следует записать значение в регистр ARR, которое получается из следующего соотношения:

$$\mathrm{ARR\_VAL} = \frac{\mathrm{TIMxCNT}}{\mathrm{TIM\_out\_freq}} - 1, \qquad (1.2)$$

где ARR_VAL – значение для записи в регистр ARR; TIMxCNT – частота счетчика; TIMx_out_freq – нужная выходная частота ШИМ.

Последним этапом является задание нужного коэффициента заполнения, что обеспечит нужное значение напряжения на выходе. Данная настройка производится с помощью регистра захвата/сравнения (capture/compare register, CCRx) исходя из следующего соотношения:

$$D = \frac{CCRx\_VAL}{ARR\_VAL} \cdot 100\ \%, \qquad (1.3)$$

где D – коэффициент заполнения; CCRx_VAL – значение в регистре CCRx (x – номер регистра для конкретной линии); ARR_VAL – значение для записи в регистр ARR.

Особенностью данных микроконтроллеров является то, что в предделитель и другие регистры можно записать любое значение, которое можно описать с помощью определенного количества разрядов.

Выдача сигнала ШИМ на выход не является основным режимом работы выводов порта, а относится к дополнительным (альтернативным) режимам. Поэтому предварительно нужно задать нужный режим в настройках порта.

Для подключения альтернативных функций к портам ввода/вывода необходимо:

1. Подключить пин к альтернативной функции соответствующего периферийного устройства с помощью функции GPIO_PinAFConfig.
2. Сконфигурировать пин в режим выполнения альтернативной функции GPIO_Mode_AF.
3. Выполнить остальные настройки.
4. Вызвать GPIO_Init для применения указанных настроек.

Пример программы

Для ознакомления с возможностью генерации сигнала ШИМ и использования его для управления яркостью светодиодов представлена следующая программа (в листинге представлено содержание файла с главной функцией).

Листинг кода:

```
#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "misc.h"
```

```
#include "stm32f4xx_tim.h"
#include "init.h"

int main(void)
{
	init();

	int brightness = 0;
	int i;

	while(1)
	{
		brightness++;

		TIM4->CCR3 = 333 - (brightness + 0) % 333;
		TIM4->CCR4 = 333 - (brightness + 166 / 2) % 333;
		TIM4->CCR1 = 333 - (brightness + 333 / 2) % 333;
		TIM4->CCR2 = 333 - (brightness + 499 / 2) % 333;

		for(i=0;i < 10000; ++i);
			for(i=0;i < 10000; ++i);

				for(i=0;i < 10000; ++i);

	}

}
```

В основной функции происходит изменение значений регистра захвата/сравнения с определенной задержкой.

Функции с выполнением настроек перенесены в отдельный файл init.c. Соответственно, объявления функций находятся в файле init.h. В следующем листинге представлено содержимое файла init.c.

Листинг кода:

```
#include "init.h"
#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "misc.h"
#include "stm32f4xx_tim.h"

void init() {
	GPIOinit();
	TimerInit();

}

void TimerInit() {
```

```
	TIM_TimeBaseInitTypeDef time_init;
	TIM_OCInitTypeDef oc_init;
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

	uint16_t PrescalerValue = (uint16_t)((SystemCoreClock / 2) /
21000000);

	time_init.TIM_Period = 665;
	time_init.TIM_Prescaler = PrescalerValue;
	time_init.TIM_ClockDivision = 0;
	time_init.TIM_CounterMode = TIM_CounterMode_Up;

	TIM_TimeBaseInit(TIM4, &time_init);

	oc_init.TIM_OCMode = TIM_OCMode_PWM1;
	oc_init.TIM_OutputState = TIM_OutputState_Enable;
	oc_init.TIM_Pulse = 0;
	oc_init.TIM_OCPolarity = TIM_OCPolarity_High;

	TIM_OC1Init(TIM4, &oc_init);
	TIM_OC1PreloadConfig(TIM4, TIM_OCPreload_Enable);

	oc_init.TIM_OutputState = TIM_OutputState_Enable;
	oc_init.TIM_Pulse = 0;

	TIM_OC2Init(TIM4, &oc_init);
	TIM_OC2PreloadConfig(TIM4, TIM_OCPreload_Enable);

	oc_init.TIM_OutputState = TIM_OutputState_Enable;
	oc_init.TIM_Pulse = 0;

	TIM_OC3Init(TIM4, &oc_init);
	TIM_OC3PreloadConfig(TIM4, TIM_OCPreload_Enable);

	oc_init.TIM_OutputState = TIM_OutputState_Enable;
	oc_init.TIM_Pulse = 0;

	TIM_OC4Init(TIM4, &oc_init);
	TIM_OC4PreloadConfig(TIM4, TIM_OCPreload_Enable);

	TIM_ARRPreloadConfig(TIM4, ENABLE);
	TIM_Cmd(TIM4, ENABLE);
}

void GPIOinit() {
	GPIO_InitTypeDef gpio_init;
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

	gpio_init.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 |
	GPIO_Pin_15;
	gpio_init.GPIO_Mode = GPIO_Mode_AF;
```

```
    gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
    gpio_init.GPIO_OType = GPIO_OType_PP;
    gpio_init.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOD, &gpio_init);

    GPIO_PinAFConfig(GPIOD, GPIO_PinSource12, GPIO_AF_TIM4);
    GPIO_PinAFConfig(GPIOD, GPIO_PinSource13, GPIO_AF_TIM4);
    GPIO_PinAFConfig(GPIOD, GPIO_PinSource14, GPIO_AF_TIM4);
    GPIO_PinAFConfig(GPIOD, GPIO_PinSource15, GPIO_AF_TIM4);
}
```

Как видно из примера, для инициализации таймера необходимо сразу две структуры: одна – для инициализации непосредственно таймера, а другая – для задания режима сравнения выхода. Приведенная программа позволяет последовательно уменьшать яркость свечения пользовательских светодиодов на плате.

Ход работы

1. На основе кода примера приложения создать свой проект в среде разработки и проверить его работоспособность.
2. Ознакомиться с работой используемых функций, просмотрев исходный код, а также комментарии в исходных файлах библиотеки и в режиме отладки.
3. Создать новый проект в среде разработки для выполнения индивидуального задания, полученного от преподавателя.
4. Реализовать необходимую функциональность.
5. Запрограммировать плату и продемонстрировать работу программы.

Лабораторная работа №4

ИСПОЛЬЗОВАНИЕ АЦП

Цель работы: исследовать возможности использования АЦП.

Оборудование и программное обеспечение: перемычки для подключения источника напряжения (батарейки) ко входам АЦП.

Теоретический материал

Микроконтроллер STM32F407VG включает в себя три АЦП разрядностью 12 бит [15]. Каждый преобразователь способен принимать сигнал из 16 внешних каналов.

Кроме того, в состав контроллера (не платы) входит датчик температуры. Диапазон входных напряжений составляет 1,8…3,6 В. Датчик температуры подключен к входному каналу ADS_IN16, который используется для того, чтобы преобразовать выходное напряжение сенсора в цифровое значение.

Внутренний датчик температуры предназначен для отслеживания изменения температуры, а не для ее измерения, поскольку смещение показателей датчика может меняться в ходе изменений параметров процесса. Поэтому если необходимо точное измерение абсолютных значений температуры, то для этого лучше использовать внешний датчик [10].

Пример программы

В данном примере приведена программа, позволяющая измерять напряжение, которое подается на вход АЦП. Просмотр значений напряжения производится в режиме отладки (рисунок 1.5).

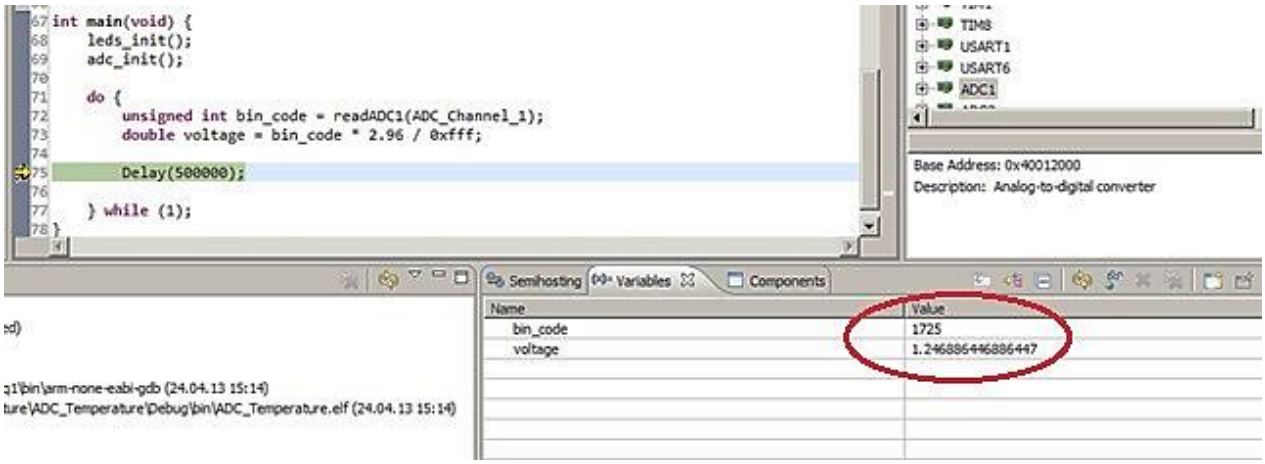

Рисунок 1.5 – Просмотр значения напряжения в режиме отладки

Для того чтобы правильно определить поданное напряжение, необходимо провести дополнительные измерения. Для этого с помощью мультиметра или вольтметра определяем напряжение, которое соответствует 3 В, подключив их к соответствующим выводам на плате. Мультиметр показал, что в данном случае напряжение равно 2,96 В. Поэтому записываем данный коэффициент непосредственно в код программы.

Расчет напряжения производится по формуле

$$U_{res} = \frac{U_{ref} \cdot BIN_{res}}{BIN_{max}}, \quad (1.4)$$

где U_{res} – значение поданного напряжения; U_{ref} – напряжение, относительно которого производится сравнение; BIN_{max} – двоичный код максимально возможного значения, которое может храниться в регистре данных АЦП, зависит от его разрядности (в нашем случае разрядность АЦП – 12, поэтому максимальному напряжению соответствует число 0xFFF – число, у которого в младших 12 разрядах все единицы); BIN_{res} – значение двоичного кода из регистра данных АЦП, которое записано после проведения измерений.

Соответственно, исходя из формулы (1.4), при подаче напряжения в 0 В мы получим минимальное значение напряжения. Сделаем это, соединив выводы земли GND и первого вывода порта A перемычкой. В результате значения в регистре данных АЦП изменятся следующим образом (рисунок 1.6).

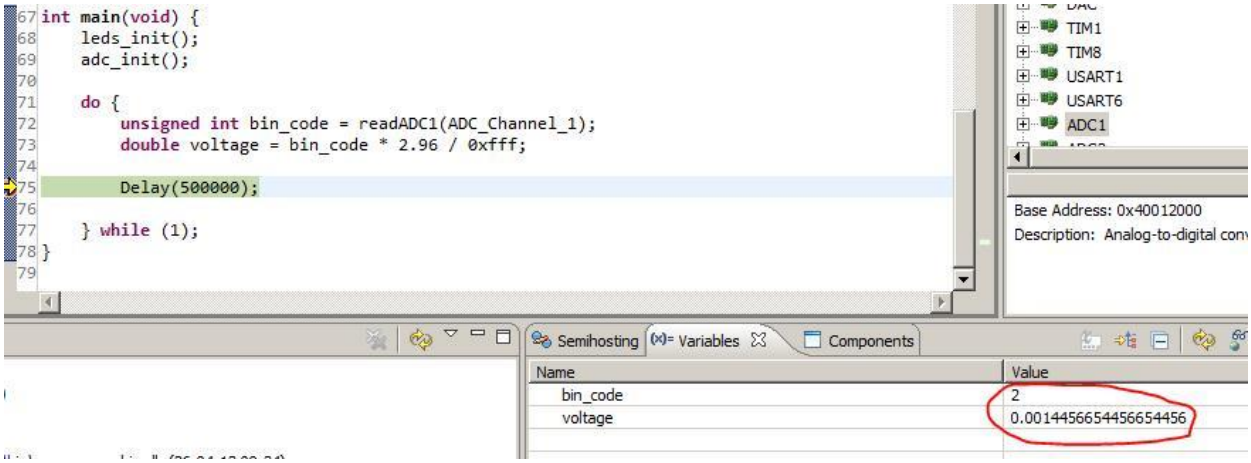

Рисунок 1.6 – Просмотр значения напряжения в режиме отладки при подключении земли ко входу

Как видим, значение достаточно близко к нулю, что говорит о правильности работы программы.

Для проведения проверки данной программы подключение источника питания выполнено следующим образом (рисунок 1.7).

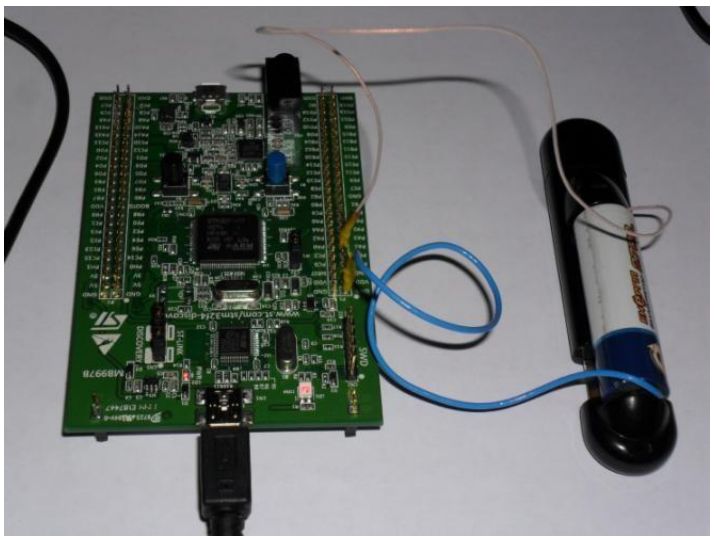

Рисунок 1.7 – Подключение источника питания
для проведения измерений АЦП

Рассмотрим более детально программу, которая выполняет необходимые действия. Содержимое основного файла main.c представлено ниже.

Листинг кода:

```
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_adc.h>

void leds_init() {
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

	GPIO_StructInit(&gpio);
	gpio.GPIO_Mode = GPIO_Mode_AN;
	gpio.GPIO_Pin = GPIO_Pin_1;
	GPIO_Init(GPIOA, &gpio);
}

void adc_init() {
	ADC_InitTypeDef ADC_InitStructure;
	ADC_CommonInitTypeDef adc_init;
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
	ADC_DeInit();

	ADC_StructInit(&ADC_InitStructure);
	adc_init.ADC_Mode = ADC_Mode_Independent;
	adc_init.ADC_Prescaler = ADC_Prescaler_Div2;

	ADC_InitStructure.ADC_ScanConvMode = DISABLE;
	ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
	ADC_InitStructure.ADC_ExternalTrigConv =
	ADC_ExternalTrigConvEdge_None;
```

```
	ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
	ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;

	ADC_CommonInit(&adc_init);
	ADC_Init(ADC1, &ADC_InitStructure);
	ADC_Cmd(ADC1, ENABLE);
}
u16 readADC1(u8 channel) {

	ADC_RegularChannelConfig(ADC1, channel, 1,
	ADC_SampleTime_3Cycles);
	ADC_SoftwareStartConv(ADC1);
	while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
	return ADC_GetConversionValue(ADC1);
}

void Delay(unsigned int Val) {

	for (; Val != 0; Val--);
}

int main(void) {

	leds_init();
	adc_init();

	do {
		unsigned int bin_code = readADC1(ADC_Channel_1);
		double voltage = bin_code * 2.96 / 0xfff;

		Delay(500000);
	} while (1);

}
```

В ней есть несколько моментов, на которые нужно обратить особенное внимание. Во-первых, при инициализации порта мы задаем значение, которое нужно для его работы в аналоговом режиме, с помощью GPIO_Mode_AN. Во-вторых, производим инициализацию и считывание значений из АЦП. Мы настраиваем АЦП таким образом, что преобразование производится программно, и выполняем его с помощью вызова функции readADC1.

Ход работы

1. На основе кода примера приложения создать свой проект в среде разработки и проверить его работоспособность.

2. Ознакомиться с работой используемых функций, просмотрев исходный код, а также комментарии в исходных файлах библиотеки и в режиме отладки.
3. Создать новый проект в среде разработки для выполнения индивидуального задания, полученного от преподавателя.
4. Реализовать необходимую функциональность.
5. Запрограммировать плату и продемонстрировать работу программы.

Индивидуальные задания

1. Разработать программу, которая при подаче напряжения, равного половине его максимального значения, выключает светодиод, а при подаче большего напряжения – включает светодиод. Проверить корректность работы программы с помощью мультиметра.
2. Продемонстрировать прием данных с нескольких каналов АЦП.
3. Подключить к АЦП для измерения встроенный датчик изменения температуры. При изменении в бóльшую сторону включать красный светодиод, при уменьшении – синий.
4. Подключить к АЦП встроенный источник напряжения. При изменении в бóльшую сторону включать красный светодиод, при уменьшении – синий.

Лабораторная работа №5

ИСПОЛЬЗОВАНИЕ USART

Цель работы: научиться использовать универсальный синхронный/асинхронный приемопередатчик; организовать передачу данных другим устройствам.

Оборудование и программное обеспечение: плата STM32F4 Discovery, среда разработки CooCox CoIDE 1.7.

Теоретический материал

Микроконтроллер на демонстрационной плате содержит в общей сложности шесть приемопередатчиков. При этом четыре из них являются синхронно-асинхронными (USART1, USART2, USART3, USART6) и два только асинхронными (USART4, USART5) [8].

Рассмотрим процесс настройки USART для приема данных. Для этого необходимо выполнить следующие действия:

1. Включить тактирование USART и порта, выходы которого используются для работы USART.
2. Выполнить настройку соответствующих выходов, указав при этом режим работы выполнения альтернативной функции.
3. Подключить выполнение альтернативной функции к указанным выходам.
4. Задать настройки работы USART (частота, размер передачи, количество стоп-бит, проверка на парность).
5. Включить прерывание по приему данных для USART.
6. Выключить USART.

Пример программы

В данном примере рассмотрим прием данных с помощью USART1. Для передачи и приема используются выходы порта A под номерами 9 (Transmit) и 10 (Receive). Для каждого приемопередатчика выделены свои пары выводов для приема и передачи, которые можно найти в документации к микроконтроллеру. Прежде всего включаем тактирование нужных устройств.

Листинг кода:

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
```

Далее проводим инициализацию выводов порта и назначаем выполнение альтернативной функции:

Листинг кода:

```
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_10;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_Init(GPIOA, &GPIO_InitStruct);
```

Далее выбираем альтернативную функцию, выполняемую выводами порта. Делаем это с помощью функции GPIO_PinAFConfig().

Листинг кода:

```
GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_USART1);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_USART1);
```

Следует обратить внимание, что настройке подлежат только определенные выводы, другие выводы, например, PA2 и PA3, использовать в качестве выводов USART не удастся.

Далее необходимо инициализировать USART. Код инициализации очень похож на инициализацию порта.

Листинг кода:

```
USART_InitStruct.USART_BaudRate = baudrate;
USART_InitStruct.USART_WordLength = USART_WordLength_8b;
USART_InitStruct.USART_StopBits = USART_StopBits_1;
USART_InitStruct.USART_Parity = USART_Parity_No;
USART_InitStruct.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
USART_InitStruct.USART_Mode = USART_Mode_Rx;
USART_Init(USART1, &USART_InitStruct);
```

На последнем этапе инициализации необходимо настроить прерывание и включить USART.

Листинг кода:

```
USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
```

```
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
USART_Cmd(USART1, ENABLE);
```

После того как мы включили прерывание, необходимо определить обработчик этого прерывания. Обработчик прерывания имеет имя USART1_IRQHandler и он не возвращает и не принимает никаких данных. Кроме того, USART поддерживает несколько прерываний и для всех прерываний используется один обработчик, поэтому необходимо проверять флаг (RXNE) в регистре статуса, чтобы организовать обработку разных прерываний в разных блоках кода обработчика [10]. После проверки состояния флага есть возможность сбросить флаг RXNE. Далее считываем данные из регистра данных и выводим их на дисплей.

Листинг кода:

```
void USART1_IRQHandler(void){
     if( USART_GetITStatus(USART1, USART_IT_RXNE) ){
          USART_ClearITPendingBit(USART1, USART_IT_RXNE);
          static uint8_t cnt = 0;
          uint8_t t = USART_ReceiveData(USART1);
          write_char(t);
     }
}
```

Рассмотрим прием данных с другого устройства (использована плата STM8S Value Line, однако возможно использовать любое другое устройство с UART или USART) и отображение принятой информации на символьном LCD-дисплее [4]. Более того, для проверки работы USART даже не нужно использовать другое устройство, т. к. можно просто замкнуть между собой выходы передатчика и приемника [9]. Важным моментом является то, что необходимо задать одинаковые параметры сообщения как на устройстве-передатчике, так и на приемнике.

Результат представлен на рисунке 1.8.

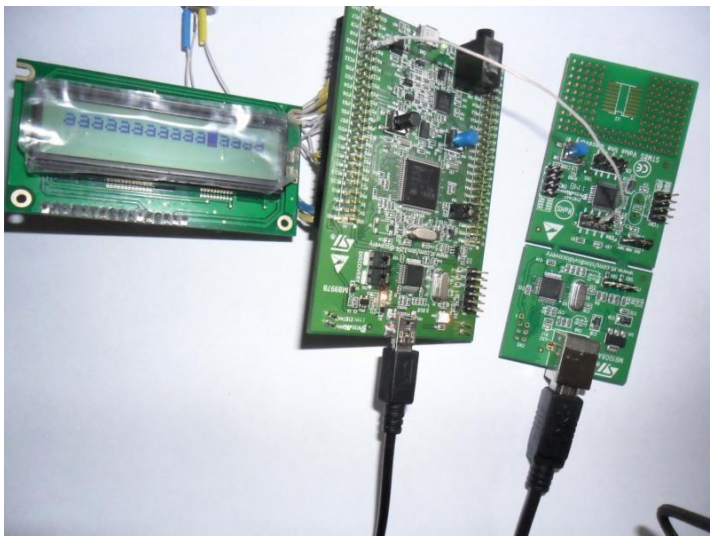

Рисунок 1.8 – Результат работы программы

Ход работы

1. На основе кода примера приложения создать свой проект в среде разработки и проверить его работоспособность.
2. Ознакомиться с работой используемых функций, просмотрев исходный код, а также комментарии в исходных файлах библиотеки в режиме отладки.
3. Создать новый проект в среде разработки для выполнения индивидуального задания, полученного от преподавателя.
4. Реализовать необходимую функциональность.
5. Запрограммировать плату и продемонстрировать работу программы.

Индивидуальные задания

1. Задействовать два модуля USART на плате и передать данные из одного в другой в асинхронном режиме. Для демонстрации приема данных изменять состояние одного из пользовательских светодиодов на противоположный.
2. Продемонстрировать прием/передачу в синхронном режиме. Для демонстрации приема данных изменять состояние одного из пользовательских светодиодов на противоположный.

Лабораторная работа №6

РАБОТА С SPI

Цель работы: научиться использовать интерфейс SPI для организации передачи данных между устройствами; исследовать режим ведущего и ведомого.

Оборудование и программное обеспечение: плата STM32F4 Discovery, среда разработки CooCox CoIDE 1.7, Keil 4.xx, инструменты построения проекта GNU Toolchain for ARM Embedded Processors, библиотека CMSIS, набор перемычек (3 шт.).

Теоретический материал

Serial Peripheral Interface (SPI) – популярный интерфейс для последовательного обмена данными между микросхемами. Шина SPI организована по принципу «ведущий – подчиненный». В качестве ведущего шины обычно выступает микроконтроллер, но им также может быть программируемая логика, DSP-контроллер или специализированная ИС. Подключенные к ведущей шине внешние устройства являются подчиненными шины. В их роли выступают различного рода микросхемы, в том числе запоминающие устройства (EEPROM, Flash-память, SPAM), часы реального времени (RTC), АЦП/ЦАП, цифровые потенциометры, специализированные контроллеры и др.

Главным составным блоком интерфейса SPI является обычный сдвиговый регистр, сигналы синхронизации и ввода/вывода битового потока которого и образуют интерфейсные сигналы. Таким образом, протокол SPI правильнее назвать не протоколом передачи данных, а протоколом обмена данными между двумя сдвиговыми регистрами, каждый из которых одновременно выполняет и функцию приемника, и функцию передатчика. Непременным условием передачи данных по шине SPI является генерация сигнала синхронизации шины. Этот сигнал имеет право генерировать только ведущий шины и от этого сигнала полностью зависит работа подчиненного шины.

Пример наиболее простого подключения по шине SPI показан на рисунке 1.9. Одноименные выводы соединяются между собой [5]. Присутствуют два канала передачи данных (MOSI, MISO), один канал для подачи тактовых импульсов (SCLK), а также линия включения ведомого устройства (SS). Ведомый становится активным при подаче низкого уровня по линии SS.

Рисунок 1.9 – Схема простого подключения с использованием SPI

SPI – чрезвычайно простой и распространенный последовательный интерфейс передачи данных, который основывается на сдвиговых регистрах [5]. Его преимуществом по сравнению с USART является возможность подключения нескольких ведомых устройств. Однако по сравнению с USART он поддерживает только синхронную передачу. Наличие нескольких модулей SPI в составе микроконтроллера позволяет обойтись без подключения дополнительных устройств с таким интерфейсом, а использовать несколько устройств на плате, соединив их входы между собой перемычками. Этот вариант является оптимальным для учебных целей, поскольку требует минимум дополнительного оборудования для начала работы.

Листинг кода:

```
#include <stm32f4xx.h>
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_spi.h>
#include <misc.h>

#define SPI_PINS GPIO_Pin_5 | \
                 GPIO_Pin_6 | \
                 GPIO_Pin_7

void init(void);
void delay(int count);
void SPI1_IRQHandler(void) {
      int res;
      if (SPI_I2S_GetITStatus(SPI1, SPI_I2S_IT_RXNE) != RESET) {
            SPI_I2S_ClearITPendingBit(SPI1, SPI_I2S_IT_RXNE);
            res = SPI_I2S_ReceiveData(SPI1);
      }
}

int main(void)
{
```

```
	init();
	int sendData = 0;
	while(1)
	{
		delay(100);
		SPI_I2S_SendData(SPI2, sendData);
		while (SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_TXE) ==
		RESET);
		sendData++;
		if (sendData == 0xff) sendData = 0;
	}
}

void init(void) {
	GPIO_InitTypeDef gpio_init;
	SPI_InitTypeDef spi_init;
	NVIC_InitTypeDef nvic_init;

	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA |
			RCC_AHB1Periph_GPIOB |
			RCC_AHB1Periph_GPIOC, ENABLE);

	/* Configure slave pins */
	gpio_init.GPIO_Mode = GPIO_Mode_AF;
	gpio_init.GPIO_Pin = SPI_PINS;
	gpio_init.GPIO_Speed = GPIO_Speed_50MHz;
	gpio_init.GPIO_OType = GPIO_OType_PP;
	gpio_init.GPIO_PuPd = GPIO_PuPd_DOWN;
	GPIO_Init(GPIOA, &gpio_init);

	GPIO_PinAFConfig(GPIOA, GPIO_PinSource5, GPIO_AF_SPI1);
	GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_SPI1);
	GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_SPI1);

	gpio_init.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3;
	GPIO_Init(GPIOC, &gpio_init);

	GPIO_PinAFConfig(GPIOC, GPIO_PinSource2, GPIO_AF_SPI2);
	GPIO_PinAFConfig(GPIOC, GPIO_PinSource3, GPIO_AF_SPI2);

	gpio_init.GPIO_Pin = GPIO_Pin_10;
	GPIO_Init(GPIOB, &gpio_init);

	GPIO_PinAFConfig(GPIOB, GPIO_PinSource10, GPIO_AF_SPI2);
```

```
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
	SPI_I2S_DeInit(SPI1);
	spi_init.SPI_Mode = SPI_Mode_Slave;
	spi_init.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
	spi_init.SPI_DataSize = SPI_DataSize_8b;
	spi_init.SPI_CPOL = SPI_CPOL_Low;
	spi_init.SPI_CPOL = SPI_CPHA_1Edge;
	spi_init.SPI_FirstBit = SPI_FirstBit_MSB;
	spi_init.SPI_NSS = SPI_NSS_Soft;
	spi_init.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2;
	SPI_Init(SPI1, &spi_init);
	SPI_I2S_ITConfig(SPI1, SPI_I2S_IT_RXNE, ENABLE);

	RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI2, ENABLE);
	SPI_StructInit(&spi_init);
	SPI_I2S_DeInit(SPI2);
	spi_init.SPI_Mode = SPI_Mode_Master;
	spi_init.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
	spi_init.SPI_DataSize = SPI_DataSize_8b;
	spi_init.SPI_CPOL = SPI_CPOL_Low;
	spi_init.SPI_CPOL = SPI_CPHA_1Edge;
	spi_init.SPI_FirstBit = SPI_FirstBit_MSB;
	spi_init.SPI_NSS = SPI_NSS_Soft;
	SPI_Init(SPI2, &spi_init);

	nvic_init.NVIC_IRQChannel = SPI1_IRQn;
	nvic_init.NVIC_IRQChannelCmd = ENABLE;
	nvic_init.NVIC_IRQChannelPreemptionPriority = 0;
	nvic_init.NVIC_IRQChannelSubPriority = 0;
	NVIC_Init(&nvic_init);

	SPI_Cmd(SPI1, ENABLE);
	SPI_Cmd(SPI2, ENABLE);
}

void delay(int count) {
	while(--count);
}
```

Ход работы

1. На основе кода примера приложения создать свой проект в среде разработки и проверить его работоспособность.
2. Ознакомиться с работой используемых функций, просмотрев исходный код, а также комментарии в исходных файлах библиотеки в режиме отладки.

3. Создать новый проект в среде разработки для выполнения индивидуального задания, полученного от преподавателя.
4. Реализовать необходимую функциональность.
5. Запрограммировать плату и продемонстрировать работу программы.

Индивидуальные задания

1. Настроить на отладочной плате один из модулей в режим приема данных и принять от другого устройства с отображением на семисегментном индикаторе.
2. Продемонстрировать работу в режиме ведущего с подключением нескольких устройств: первое устройство – другой модуль SPI на плате, а второе – любое другое устройство, которое может выводить полученные данные на подключенный семисегментный индикатор.
3. Организовать передачу данных нескольким ведомым устройствам через определенный интервал времени. Подключить два устройства к одному модулю SPI на отладочной плате и передавать данные попеременно с интервалом приблизительно в одну секунду с отображением принятых данных.
4. Осуществить передачу данных другому модулю SPI на отладочной плате (значения от 0x00 до 0x0F) с отображением двоичного значения на светодиодах на плате.
5. Организовать одновременную передачу данных нескольким ведомым устройствам, которые отображают полученное значение на семисегментном индикаторе и в двоичном коде с использованием светодиодов.

Лабораторная работа №7

РАБОТА С DMA

Цель работы: исследовать возможности использования DMA, основные характеристики DMA, взаимодействие с другими устройствами в составе микроконтроллера.

Оборудование и программное обеспечение: плата STM32F4 Discovery, мультиметр, среда разработки CooCox CoIDE 1.7, Keil 4.xx, инструменты построения проекта GNU Toolchain for ARM Embedded Processors, библиотека CMSIS.

Теоретический материал

Direct Memory Access (DMA, «прямой доступ к памяти») – механизм, используемый в контроллерах ARM для перемещения данных между памятью и периферией без участия процессора. Ключевым моментом является то, что при использовании DMA на перемещение данных не используются ресурсы процессора, что может быть особенно критичным при создании приложений, работающих с большим количеством данных и активно использующих периферию. Работа DMA обеспечивается отдельным контроллером, который выполняет определенные действия по команде процессора.

Рассматриваемый контроллер имеет в своем распоряжении сразу два контроллера DMA, которые обеспечивают в общей сложности 16 потоков (по 8 потоков на каждый контроллер), каждый из которых предназначен для управления запросами к памяти от одного или нескольких устройств. Каждый поток может обеспечивать до восьми каналов (запросов). Каждый контроллер DMA имеет устройство разрешения конфликтов для обработки запросов в соответствии с их приоритетом.

Контроллер DMA позволяет производить запись данных в трех направлениях:

- от периферии в память;
- из памяти к периферии;
- из памяти в память.

Передача ведется либо в режиме непосредственной передачи, либо в режиме очереди. Поддерживается различный размер данных для передачи, причем размер данных для приемника и источника может быть неодинаковым. В таком случае DMA определяет данную ситуацию и выполняет необходимые действия для оптимизации передачи, однако эта возможность поддерживается только в режиме очереди.

Кроме того, очень полезной может оказаться функция циклической передачи последней единицы данных источника.

Программист может задавать приоритеты для запроса каждого потока или приоритет будет определяться на основе значений по умолчанию.

Пример программы

Рассмотрим пример программы, которая использует DMA для передачи данных из памяти в ЦАП. Изменение уровня сигнала на выходе ЦАП происходит циклически с частотой изменения в 1 с. Изменения происходят на основе значения, которое считывается из памяти по сигналу переполнения таймера. В результате этого ЦАП отправляет запрос к DMA и получает данные, которые записываются в регистр данных, что приводит к изменению уровня сигнала. Схема взаимодействия выглядит следующим образом (рисунок 1.10).

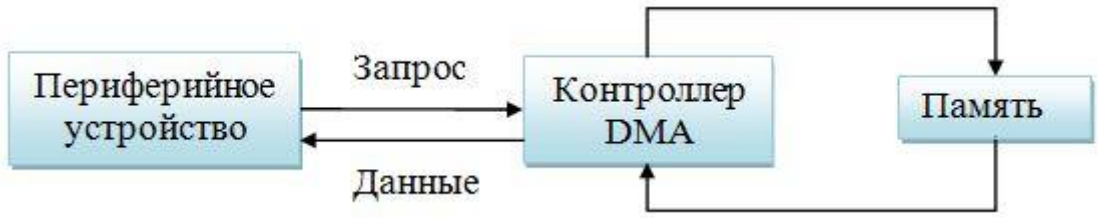

Рисунок 1.10 – Схема использования DMA

В самой программе следует обратить внимание на большое количество параметров, необходимых для настройки DMA, по сравнению с другими устройствами. С этим связана и сложность использования DMA, поскольку необходимо учитывать большое количество настроек. Тем не менее многие из них достаточно просты (направление передачи, значения адресов), поэтому изучение DMA можно сопоставить по сложности с другими устройствами, в чем очень помогает библиотека Standard Peripheral Library.

Листинг кода:

```
#include <stm32f4xx.h>
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_dma.h>
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_tim.h>
#include <stm32f4xx_dac.h>


    uint8_t levels[] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66,
0x77};
```

```
void init(void);
void init_gpio(void);
void init_timer(void);
void init_dac(void);
void init_dma(void);

int main(void)
{
	init();
	while(1)
	{
	}
}

void init(void) {
	init_gpio();
	init_timer();
	init_dac();
	init_dma();
}

void init_gpio(void) {
	GPIO_InitTypeDef gpio_init;
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
	gpio_init.GPIO_Mode = GPIO_Mode_AN;
	gpio_init.GPIO_Pin = GPIO_Pin_4;
	gpio_init.GPIO_OType = GPIO_OType_PP;
	gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
	gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
	GPIO_Init(GPIOA, &gpio_init);
}

void init_timer(void) {
	TIM_TimeBaseInitTypeDef tim_init;
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6, ENABLE);
	tim_init.TIM_CounterMode = TIM_CounterMode_Up;
	tim_init.TIM_Period = 16000 - 1;
	tim_init.TIM_Prescaler = 1000 - 1;
	TIM_TimeBaseInit(TIM6, &tim_init);

	TIM_SelectOutputTrigger(TIM6, TIM_TRGOSource_Update);

	TIM_Cmd(TIM6, ENABLE);
}

void init_dac(void) {
```

```
    DAC_InitTypeDef dac_init;
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
    DAC_StructInit(&dac_init);
    dac_init.DAC_Trigger = DAC_Trigger_T6_TRGO;
    dac_init.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
    dac_init.DAC_WaveGeneration = DAC_WaveGeneration_None;
    DAC_Init(DAC_Channel_1, &dac_init);

}

void init_dma(void) {

    DMA_InitTypeDef dma_init;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1, ENABLE);
    DMA_DeInit(DMA1_Stream5);
    dma_init.DMA_Channel = DMA_Channel_7;
    dma_init.DMA_PeripheralBaseAddr = (uint32_t)(DAC_BASE + 0x10);
    dma_init.DMA_Memory0BaseAddr = (uint32_t)&levels;
    dma_init.DMA_DIR = DMA_DIR_MemoryToPeripheral;
    dma_init.DMA_BufferSize = 8;
    dma_init.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    dma_init.DMA_MemoryInc = DMA_MemoryInc_Enable;
    dma_init.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
    dma_init.DMA_MemoryDataSize = DMA_PeripheralDataSize_Byte;
    dma_init.DMA_Mode = DMA_Mode_Circular;
    dma_init.DMA_Priority = DMA_Priority_High;
    dma_init.DMA_FIFOMode = DMA_FIFOMode_Disable;
    dma_init.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
    dma_init.DMA_MemoryBurst = DMA_MemoryBurst_Single;
    dma_init.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
    DMA_Init(DMA1_Stream5, &dma_init);
    DMA_Cmd(DMA1_Stream5, ENABLE);

    DAC_Cmd(DAC_Channel_1, ENABLE);
    DAC_DMACmd(DAC_Channel_1, ENABLE);
}
```

После компиляции программы-примера и прошивки отладочной платы следует наблюдать за напряжением на выходе PA4 с помощью мультиметра. Прибор должен показать ступенчатое изменение напряжения от 0 В до определенного значения, после чего цикл повторяется. Шаг увеличения одинаков для всего цикла, что можно увидеть по массиву значений для записи в регистр данных.

Ход работы

1. Проверить работу программы-примера, скомпилировав ее в одной из сред разработки и выполнив прошивку.
2. Ознакомиться с документацией по DMA для микроконтроллера на отладочной плате.
3. Попробовать изменить определенные параметры в программе-примере.
4. Организовать взаимодействие DMA и другого периферийного устройства в соответствии с индивидуальным заданием.

Индивидуальные задания

1. Реализовать асинхронную передачу данных через USART, считывая данные через определенные промежутки времени посредством DMA.
2. Организовать прием данных через USART с записью в память через DMA. Количество передаваемых данных известно заранее и равно размеру массива.
3. Продемонстрировать передачу данных через SPI из памяти в циклическом режиме.
4. Продемонстрировать прием данных через SPI с записью в память. Режим записи – циклический.
5. Реализовать запись в память значений, полученных с помощью работы АЦП через определенные промежутки времени. После записи первых 50 значений начинается новый цикл записи.

Лабораторная работа №8

ГЕНЕРАЦИЯ СИГНАЛОВ И УПРАВЛЕНИЕ ИХ ХАРАКТЕРИСТИКАМИ

Цель работы: изучить возможности генерации сигналов и способы управления их характеристиками с использованием отладочной платы и отображением информации.

Оборудование и программное обеспечение: отладочная плата STM32F4 Discovery, символьный LCD-дисплей, три тактовые кнопки и резисторы для их подключения, среда разработки для STM32 (CoIDE, Keil или др.), библиотеки CMSIS и SPL.

Теоретический материал

В собранном виде с запущенной программой установка имеет вид, показанный на рисунке 1.11.

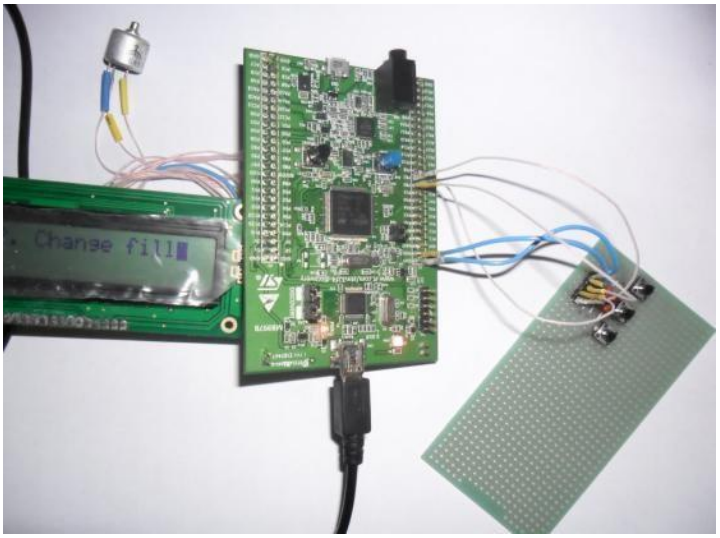

Рисунок 1.11 – Внешний вид установки

В подключении участвуют три порта ввода/вывода (к выводу порта A, на котором генерируется сигнал, подключается измерительное оборудование, которое на рисунке 1.11 не показано). Кнопки подключаются к выводам 0…2 порта B. Выходной сигнал можно наблюдать на выходе 0 порта A.

Схема подключения компонентов показана на рисунке 1.12.

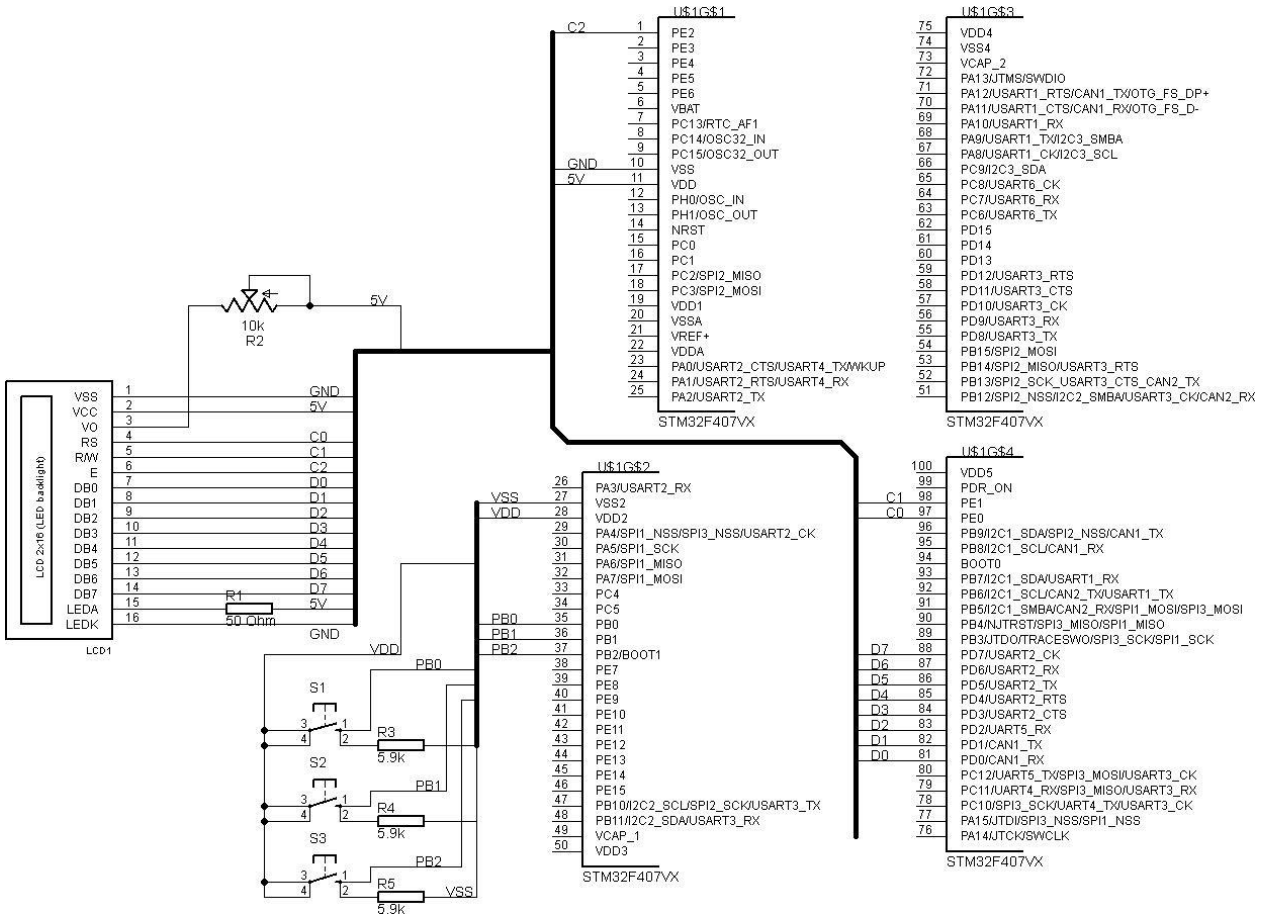

Рисунок 1.12 – Схема подключения компонентов

Описание подключения дисплея к плате можно найти в приложении, однако следует отметить необходимость для этого дополнительных резисторов, один из которых обладает переменным сопротивлением для регулирования яркости отображаемый символов.

Для управления вариантами меню и изменения характеристик используются подключенные к выходам 0…2 порта B тактовые кнопки. В данном примере использованы 4-контактные тактовые кнопки TS-06V (рисунок 1.13), однако можно использовать и другие кнопки.

Рисунок 1.13 – Внешний вид тактовой кнопки

Перед тем как выполнить подключение, следует изучить с помощью мультиметра в режиме измерения сопротивления, какие именно контакты замыкаются при нажатии, а какие остаются соединенными всегда. Кроме того, для подтяжки выходов к земле используются резисторы номиналом 5,9 кОм.

Для размещения описанного выше соединения используется макетная плата для пайки [11]. Все выводы на плате (три – для кнопок и два – для питания и земли) для удобства подключены к штыревому разъему. Подключение с отладочной платой выполняется с помощью перемычек. Возможно, более удобным вариантом для начинающих будет использование беспаечной макетной платы.

Листинг кода:

```
#include "init.h"
#include "hd44780lib.h"
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_exti.h>
#include <stm32f4xx_syscfg.h>
#include <stm32f4xx_tim.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <misc.h>

// инициализация портов, к которым подключен дисплей
void init_gpio_lcd(void) {
	GPIO_InitTypeDef gpio_init;
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD |
	RCC_AHB1Periph_GPIOE, ENABLE);
	gpio_init.GPIO_Pin = DATA_PINS;
	gpio_init.GPIO_Mode = GPIO_Mode_OUT;
	gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
	gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
	GPIO_Init(DATA_PORT, &gpio_init);
	gpio_init.GPIO_Pin = CONTROL_PINS;
	gpio_init.GPIO_Mode = GPIO_Mode_OUT;
	gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
	gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
	GPIO_Init(CONTROL_PORT, &gpio_init);
}

// инициализация порта, к которому подключены кнопки
void init_gpio_buttons(void) {
	GPIO_InitTypeDef gpio_init;
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
	gpio_init.GPIO_Mode = GPIO_Mode_IN;
	gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
	gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
	gpio_init.GPIO_Pin = BUTTON_PINS;
	GPIO_Init(GPIOB, &gpio_init);
}
```

```
// configures buttons inputs on board as interrupt source
void configure_buttons(void) {
	EXTI_InitTypeDef exti_init;
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
	SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource0);
	SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource1);
	SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource2);
	exti_init.EXTI_LineCmd = ENABLE;
	exti_init.EXTI_Line = EXTI_Line0 | EXTI_Line1 | EXTI_Line2;
	exti_init.EXTI_Mode = EXTI_Mode_Interrupt;
	exti_init.EXTI_Trigger = EXTI_Trigger_Rising;
	EXTI_Init(&exti_init);
}

// функция разрешения указанного прерывания
void enable_interrupt(IRQn_Type irq) {
	NVIC_InitTypeDef nvic_init;
	nvic_init.NVIC_IRQChannel = irq;
	nvic_init.NVIC_IRQChannelCmd = ENABLE;
	nvic_init.NVIC_IRQChannelSubPriority = 1;
	nvic_init.NVIC_IRQChannelPreemptionPriority = 1;
	NVIC_Init(&nvic_init);
}

// разрешаем прерывания по нажатии кнопки
void configure_interrupts(void) {
	enable_interrupt(EXTI0_IRQn);
	enable_interrupt(EXTI1_IRQn);
	enable_interrupt(EXTI2_IRQn);
}

void configure_delay_timer(void) {
	TIM_TimeBaseInitTypeDef tim_init;
	tim_init.TIM_CounterMode = TIM_CounterMode_Up;
	tim_init.TIM_Prescaler = 1600 - 1;//настраиваем предделитель
	таймера для задержек
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM7, ENABLE);
	TIM_TimeBaseInit(DELAY_TIMER, &tim_init);
}

// настройка таймера для генерации ШИМ
void configure_pwm_timer(void) {
	GPIO_InitTypeDef gpio_init;
	TIM_TimeBaseInitTypeDef tim_init;
	TIM_OCInitTypeDef oc_init;
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
	gpio_init.GPIO_Mode = GPIO_Mode_AF;
	gpio_init.GPIO_Pin = GPIO_Pin_0;
	gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
	gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
	GPIO_Init(GPIOA, &gpio_init);
```

```
	GPIO_PinAFConfig(GPIOD, GPIO_PinSource0, GPIO_AF_TIM2);

	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
	tim_init.TIM_Prescaler = 10 - 1;
	tim_init.TIM_Period = frequency_value - 1;
	tim_init.TIM_CounterMode = TIM_CounterMode_Up;
	tim_init.TIM_ClockDivision = 0;
	tim_init.TIM_RepetitionCounter = 0;
	TIM_TimeBaseInit(PWM_TIMER, &tim_init);

	oc_init.TIM_OCMode = TIM_OCMode_PWM1;
	oc_init.TIM_Pulse = frequency_value / 2;
	oc_init.TIM_OutputState = TIM_OutputState_Enable;
	TIM_OC1Init(PWM_TIMER, &oc_init);

	TIM_Cmd(PWM_TIMER, ENABLE);
}

// заполняем меню
void fill_menu(void) {
	main_option_index = 0;
	main_options[0] = create_menu_item("1.Change frequency",
	set_active_freq_menu);
	main_options[1] = create_menu_item("2. Change fill",
	set_active_fill_menu);

	main_menu.items[0] = create_menu_item("", down_menu);
	main_menu.items[1] = main_options[main_option_index];
	main_menu.items[2] = create_menu_item("", up_menu);

	freq_menu.items[0] = create_menu_item("", decrement_freq);
	freq_menu.items[1] = create_menu_item("",
	set_active_main_menu);
	freq_menu.items[2] = create_menu_item("", increment_freq);

	filling_menu.items[0] = create_menu_item("", increment_fill);
	filling_menu.items[1] = create_menu_item("",
	set_active_main_menu);
	filling_menu.items[2] = create_menu_item("", decrement_fill);
	set_active_menu(&main_menu);
}

void set_active_menu(struct menu * menu) {
	active_menu = menu;
}

void set_active_freq_menu(void) {
	active_menu = &freq_menu;
	show_freq_on_screen();
```

```
}

void set_active_fill_menu(void) {
      active_menu = &filling_menu;
      show_fill_on_screen();
}

void set_active_main_menu(void) {
      active_menu = &main_menu;
      show_text_on_screen(&main_menu.items[1]);
}

// реализация функции задержки
void delay(int times) {
      DELAY_TIMER->CNT = 0;
      DELAY_TIMER->ARR = times;
      TIM_Cmd(DELAY_TIMER, ENABLE); // запуск таймера
      while(TIM_GetFlagStatus(DELAY_TIMER, TIM_FLAG_Update) ==
      RESET); // ожидание
      TIM_ClearFlag(DELAY_TIMER, TIM_FLAG_Update);
      TIM_Cmd(DELAY_TIMER, DISABLE); // остановка таймера
}

struct menu_item create_menu_item(char * text, action act) {
      struct menu_item item;
      strcpy(item.menu_text, text);
      item.action = act;
      return item;
}

void show_text_on_screen(struct menu_item * item) {
      lcd_clear();
      write_string(item->menu_text);
}

// увеличение значения переменной для изменения частоты
void increment_freq(void) {
      if (frequency_value == MAX_FREQ_VALUE) return;
      frequency_value++;
      TIM_SetAutoreload(PWM_TIMER, frequency_value - 1);
      TIM_SetCompare1(PWM_TIMER, frequency_value * fill_value);
      show_freq_on_screen();
}

// уменьшение значения переменной для изменения частоты
void decrement_freq(void) {
      if (frequency_value == MIN_FREQ_VALUE) return;
      frequency_value--;
      TIM_SetAutoreload(PWM_TIMER, frequency_value - 1);
      TIM_SetCompare1(PWM_TIMER, frequency_value * fill_value);
      show_freq_on_screen();
```

```
}

void increment_fill(void) {
	if (MAX_FILL_VALUE - fill_value < 0.01) return;
	fill_value += 0.01;
	TIM_SetCompare1(PWM_TIMER, frequency_value * fill_value);
	show_fill_on_screen();
}

void decrement_fill(void) {
	if (fill_value - MIN_FILL_VALUE < 0.01) return;
	fill_value -= 0.01;
	TIM_SetCompare1(PWM_TIMER, frequency_value * fill_value);
	show_fill_on_screen();
}

void change_menu_item(struct menu_item * item, char * text, action
act) {
	strcpy(item->menu_text, text);
	item->action = act;
}

void chamge_menu_item(struct menu_item * cur, struct menu_item *
next) {
	cur = next;
}

void up_menu(void) {
	if (main_option_index == 1) {
	} else {
		main_option_index++;
		main_menu.items[1] = main_options[main_option_index];
		show_text_on_screen(&main_menu.items[1]);
	}
}

void down_menu(void) {
	if (main_option_index == 0) {
	} else {
		main_option_index--;
		main_menu.items[1] = main_options[main_option_index];
		show_text_on_screen(&main_menu.items[1]);
	}
}

void show_freq_on_screen(void) {
	char text[20];
	double freq = (double)16000000 / (frequency_value * 10) /
	1000;
	gcvt(freq, 4, text);
	strcat(text, " kHz");
```

```
	lcd_clear();
	write_string(text);
}

void show_fill_on_screen(void) {
	char text[20];
	gcvt(fill_value, 4, text);
	strcat(text, " %");
	lcd_clear();
	write_string(text);
}

// инициализация всех компонентов
void init_all(void) {
	frequency_value = 533;
	fill_value = 0.5;
	init_gpio_buttons();
	init_gpio_lcd();
	configure_buttons();
	configure_interrupts();
	configure_delay_timer();
	configure_pwm_timer();
	fill_menu();
	lcd_init();
	show_text_on_screen(&main_menu.items[1]);
}
```

В заголовочном файле находятся объявления функций, а также используемых переменных и констант.

Листинг кода:

```
#ifndef INIT_H
#define INIT_H

#include <stm32f4xx.h>
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_tim.h>

#define BUTTON_PINS GPIO_Pin_0 | \
                    GPIO_Pin_1 | \
                    GPIO_Pin_2

#define DELAY_TIMER TIM7
#define PWM_TIMER TIM2
#define MAX_FREQ_VALUE 533
#define MIN_FREQ_VALUE 400
#define MAX_FILL_VALUE 0.75
#define MIN_FILL_VALUE 0.25
```

```
typedef void (*action)(void);

// структура для представления пункта меню
struct menu_item {
      action action; // действие
      char menu_text[20]; // текст меню
};

struct menu {
      struct menu_item items[3]; // пункты меню
};

int frequency_value; // переменная для задания частоты, имеет
значения 400...533
double fill_value; // переменная для задания коэффициента
заполнения
struct menu * active_menu; // текущее меню
struct menu_item main_options[3]; // пункты главного меню
struct menu main_menu;
struct menu freq_menu;
struct menu filling_menu;
int main_option_index; // индекс текущего пункта в главном меню

void init_gpio_lcd(void);
void init_gpio_buttons(void);


void configure_buttons(void);
void configure_interrupts(void);
void configure_delay_timer(void);
void configure_pwm_timer(void);


void init_all(void);

struct menu_item create_menu_item(char * text, action act);
void show_text_on_screen(struct menu_item * item);
void show_freq_on_screen(void);
void show_fill_on_screen(void);
void fill_menu(void);
void set_active_menu(struct menu * menu);
void set_active_freq_menu(void);
void set_active_fill_menu(void);
void set_active_main_menu(void);
void empty_action(void);
void change_menu_item(struct menu_item * item, char * text, action
act);
void chamge_menu_item(struct menu_item * cur, struct menu_item *
next);
void up_menu(void);
void down_menu(void);
```

```
void increment_freq(void);
void decrement_freq(void);
void increment_fill(void);
void decrement_fill(void);

#endif // INIT_H
```

В файле с основной функцией вызывается функция инициализации всех используемых устройств и описаны обработчики прерываний для нажатий кнопок.

Листинг кода:

```
#include <stm32f4xx_exti.h>
#include <string.h>
#include "hd44780lib.h"
#include "init.h"

void EXTI0_IRQHandler(void) {
      EXTI_ClearITPendingBit(EXTI_Line0);
      if (active_menu != NULL)
            delay(100);
            active_menu->items[0].action();
}

void EXTI1_IRQHandler(void) {
      EXTI_ClearITPendingBit(EXTI_Line1);
      if (active_menu != NULL)
            delay(100);
            active_menu->items[1].action();
}

void EXTI2_IRQHandler(void) {
      EXTI_ClearITPendingBit(EXTI_Line2);
      if (active_menu != NULL)
            delay(100);
            active_menu->items[2].action();
}

int main(void)
{
      init_all();
      while(1)
      {
      }
}
```

Ход работы

В качестве примера рассмотрим программу, которая позволяет на выходе получить сигнал прямоугольной формы в заданном диапазоне частот и с заданным коэффициентом заполнения. Данная программа является своего рода закреплением пройденного до этого материала. Также ее можно рассматривать как продолжение работы с таймерами в режиме ШИМ. Соответственно пользователю необходимо предоставить возможность для регулирования указанных характеристик. Для примера взяты следующие значения:

- выходная частота – 3…4 кГц;
- коэффициент заполнения ±50 % от начального значения, которым считаем значение в 0,5.

Для генерации сигналов прямоугольной формы как нельзя лучше подходит включение таймера в режиме ШИМ для одного из выходных каналов. Необходимо лишь рассчитать значения для настройки самого таймера при указанной тактовой частоте устройства (они будут разными для 16 МГц и 168 МГц) и включить тактирование и нужный режим работы для остальных устройств в составе контроллера.

В соответствии с указанными значениями проведем расчет для настройки таймера (рассматривается случай с внутренней частотой 6 МГц). Поскольку более детально этот процесс описан в лабораторной работе №2, то далее приведены лишь результаты вычислений. Для того чтобы получить на выходе частоты в диапазоне 3…4 кГц, устанавливаем значение предделителя равным 10, тогда диапазон изменения значений в регистре автозагрузки будет меняться от 533 до 400. Именно эти значения используются как граничные.

Коэффициент заполнения не требует специальных расчетов, поэтому его значение хранится в программе в виде действительного числа, которое пользователь может изменять, и указанные изменения записываются в регистр сравнения для выбранного выходного канала.

В качестве решения для предоставления пользователю возможности регулирования параметров выходного сигнала предлагается также подключить символьный LCD-дисплей и три кнопки для выполнения регулировки. Функциональное назначение кнопок может меняться в соответствии с действиями пользователя. В то же время на экране следует отображать информацию о текущих изменениях (смена пунктов меню, значений характеристик), которые выполняет пользователь.

Перейдем к ознакомлению с кодом проекта. Функциональная часть объявлена и реализована в файлах init.h и init.c. Далее приводится листинг файла init.c.

Индивидуальные задания

1. Максимальная частота работы контроллера на плате составляет 168 МГц. Используя инструкцию по заданию частот, выполнить расчет и запустить рассмотренный пример с соблюдением значений параметров частоты и коэффициента заполнения.
2. По аналогии с приведенным примером записать программу, которая позволяет изменять частоту прямоугольного сигнала в диапазоне от 1 до 100 Гц и регулировать коэффициент заполнения во всем диапазоне. Работу продемонстрировать на частоте 1 Гц, изменяя значение коэффициента.

II. ЛАБОРАТОРНАЯ РАБОТА (ПЛИС)

Лабораторная работа №9

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ВЗАИМОДЕЙСТВИЯ ПРОГРАММИРУЕМОЙ ЛОГИКИ И МИКРОПРОЦЕССОРНОЙ СИСТЕМЫ ЧЕРЕЗ ИНТЕРФЕЙС JTAG

Цель работы: ознакомиться с методами программирования микропроцессорной системы; изучить интерфейс JTAG; разработать программное обеспечение взаимодействия программируемой логики и микропроцессорной системы через интерфейс JTAG.

Оборудование и программное обеспечение: ПЛИС Altera; среда разработки Quartus II; учебный макет CIC-560.

Теоретический материал

Среда разработки Quartus II

Для построения встраиваемых систем используются аппаратные и программные средства. В процессе создания этапы проектирования и разработки неотделимы друг от друга. Решение конечной задачи достигается путем нахождения компромисса в выборе проектной платформы и среды разработки.

Проектные платформы, использующиеся для создания встраиваемых систем, определяют процесс проектирования и разработки. Широко используемыми в производстве платформами можно назвать промышленные персональные компьютеры, микроконтроллеры, сигнальные процессоры, программно-аппаратные комплексы и контроллеры с программируемой логикой, программируемые логические интегральные схемы и сверхбольшие интегральные схемы программируемой логики. Программируемые логические интегральные схемы (ПЛИС, от англ. field-programmable gate array, FPGA) или сверхбольшие интегральные схемы программируемой логики (СБИСПЛ, от англ. programmable logic device, PLD) позволяют существенно модифицировать процесс разработки программно-аппаратных систем с использованием всего одной микросхемы и одних средств разработки.

Одним из важнейших критериев при создании встраиваемых систем является выбор среды программирования.

Современные элементы требуют использования высокотехнологичных средств программирования. Компания Altera предоставляет возможность

использования автоматизированной системы проектирования цифровых устройств Quartus II [17]. Рассмотрим подробнее выбранную среду разработки.

Программный пакет Quartus II является мультиплатформенной средой для проектирования цифровых систем на программируемых логических интегральных схемах при помощи ряда языков программирования.

Обобщенная схема проектирования в данной среде приведена на рисунке 2.1.

Рисунок 2.1 – Обобщенная схема проектирования в среде разработки Quartus II

Система автоматизированного проектирования включает:

- множество средств для разработки цифровых устройств (редактор схем, графический редактор, редактор текста и др.);
- компилятор для переноса готовой схемы на логику целевого устройства;
- анализаторы временных характеристик;
- программатор, позволяющий перемещать конфигурацию схемы из проекта в сверхбольшую интегрированную схему программируемой логики.

Языки описания аппаратуры Verilog HDL и VHDL

Для программирования ПЛИС используется язык описания аппаратуры.

Языки программирования Verilog HDL (от англ. hardware description language, HDL) и VHDL (от англ. very high speed integrated circuits hardware description language) относятся к языкам описания аппаратуры, которые применяются для проектирования цифровых схем на уровнях микросхем, регистровых передач, вентильных матрицах.Verilog HDL одинаково пригоден для восприятия как человеком, так и машиной. Имеет возможность использования на этапах разработки, синтеза, верификации, тестирования системы, отправки информации о проекте, модификации, конфигурации. Язык аппаратуры имеет возможность выполнять процесс параллельно, что является его отличительной чертой от того же языка программирования C. Verilog HDL и VHDL можно отнести к языкам высокого уровня описания аппаратуры.

Рассмотрим основные достоинства, объединяющие оба языка:

1. VHDL и Verilog HDL являются признанными стандартами описания цифровых устройств, которые применяются и в радиоэлектронной промышленности, и в военной сфере. Стандарт позволяет облегчить передачу документации между разработчиками и пользователями, а также системами автоматизированного проектирования.
2. Несомненным достоинством вышеуказанных языков является универсальность. VHDL и Verilog HDL применяются для описания как самих цифровых схем, так и алгоритмов работы и тестирования.
3. Благодаря данным языкам достигнут компромисс между требованиями к документации языка, комфортному восприятию человеком и функциональному средству для ввода, обработки и разработки схем.

VHDL разработан в 1983 году. Основной целью его создания было формальное описание (спецификация) логических схем для различных этапов разработки, начиная с модуля микросхемы до сложнейших вычислительных систем.

Выделим основные части языка описания аппаратуры VHDL:

- алгоритмическая, которая основана на языках Ada и Pascal, придающая языку VHDL свойства, необходимые для языков программирования;
- проблемно-ориентированная, благодаря которой VHDL является языком описания аппаратуры;
- объектно-ориентированная, интенсивно развивающаяся в настоящее время.

В отличие от VHDL, Verilog HDL был разработан позже компанией Gateway Design Automation как внутренний язык симуляции. Для общественного использования он был открыт в 1989 году. Однако стандарт языка был введен в 1995 году и именно этот год принято считать датой появления Verilog HDL.

Большая универсальность VHDL объясняется его основным предназначением для моделирования, позже выделено и синтезируемое подмножество. Это позволяет автоматически синтезировать разработанную модель алгоритмической системы. Похожая разработка есть и для его конкурента – языка Verilog HDL.

Существенным недостатком языка Verilog HDL является поддержка только самых элементарных типов данных: целые (32 бит), действительные (с плавающей запятой), «время» и «событие». Соответственно широкий набор типов данных и возможность создания дополнительных является большим преимуществом VHDL. В Verilog HDL сигналы могут быть только регистровыми и цепными.

Возможность применения языка VHDL в моделировании различных физических систем с поддержкой физических размерностей обусловлена синтаксисом, позволяющим описывать схемы в широкой вариативности стилей: структурное, потоковое и поведенческое описания. Verilog HDL тоже имеет возможность поддержки описания систем, но специализированного интерфейса для реализации обычных языков программирования у него нет.

Таким образом, несмотря на схожесть названий языков программирования Verilog HDL и VHDL они имеют достаточно существенные различия.

Рассмотрим построение проекта на языке VHDL с комментариями.

```
library ieee; -- библиотека ieee

use ieee.std_logic_1164.all; -- использование пакета типов данных
standard logic
use ieee.std_logic_unsigned.all; -- использование пакета для
счетчиков
use ieee.std_logic_arith.all; -- использование пакета для
арифметических операций

entity NAME is -- декларация имени объекта проекта
port (x1, x2: in bit; -- декларация входных портов
         y: out bit); -- декларация выходных портов
end NAME;
```

```
architecture functional of NAME is -- описание архитектуры
begin
  y <= x1 and x2 -- описание функции объекта
end functional;

signal x3: std_logic_vector (3 downto 0); -- декларация сигнала
```

Создание библиотечных модулей

Среда разработки Quartus II предоставляет возможность создания библиотечных модулей с последующим включением их в проект. Рассмотрим процесс реализации модуля на примере создания T-trigger:

1. Необходимо создать новый проект, затем создать новый файл и выбрать настройки для создания графического файла, установить необходимые настройки. Собрать схему, представленную на рисунке 2.2.
2. Сохранить собранную схему как символ (блок) в папку с проектом. Для этого необходимо выбрать File/Create/update/Create Symbol file for Current File (рисунки 2.3 и 2.4).
3. Для проверки использования созданного блока создать новый проект, в папку текущего проекта скопировать файлы с расширениями .bdf, .bsf из папки с проектом T-trigger.
4. Добавить в проект T-trigger. Для этого щелкнуть по значку с тремя точками и выбрать в каталоге файл с символом T-trigger. Добавить проект в рабочую область, как показано ранее (рисунок 2.5).

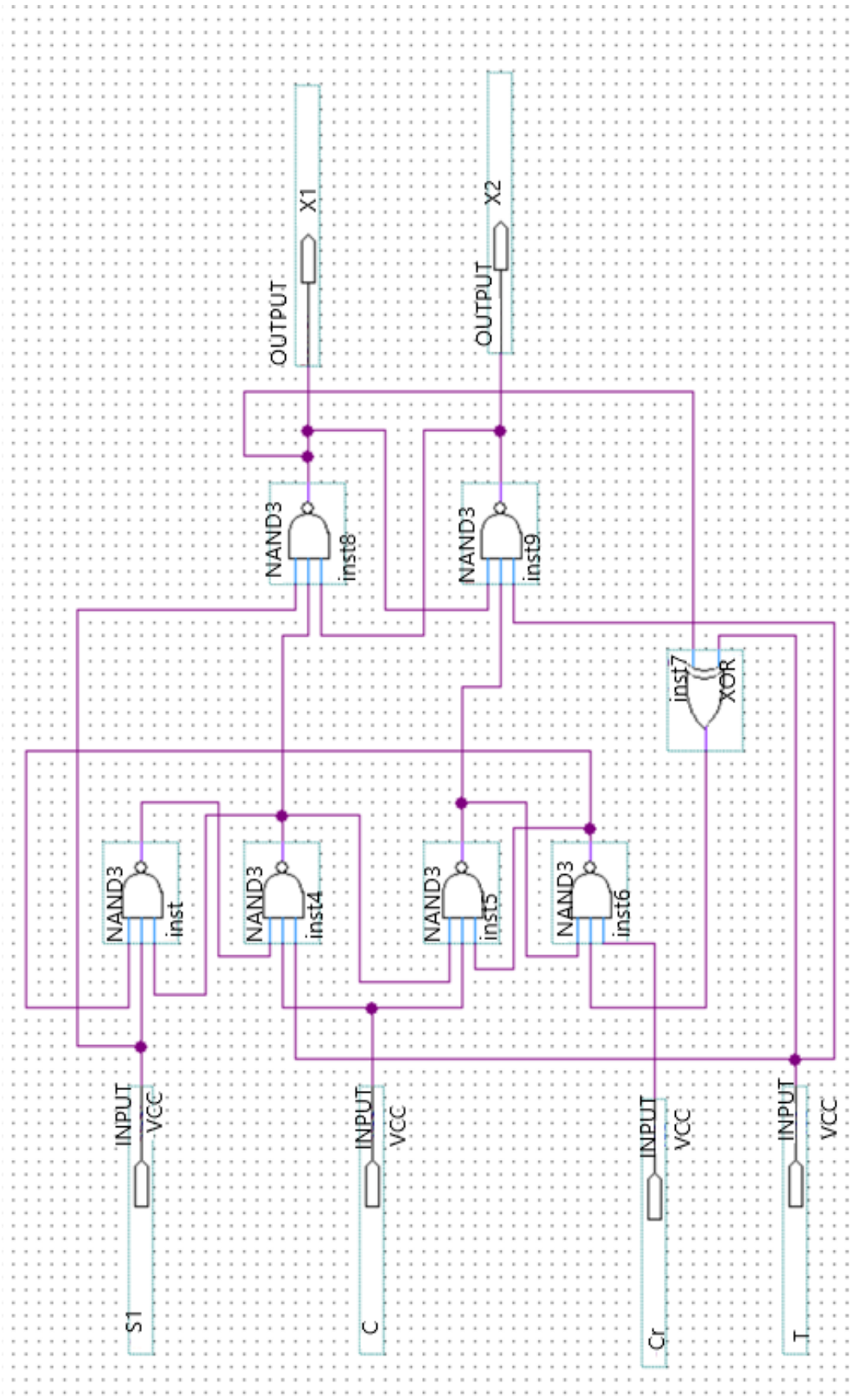

Рисунок 2.2 – Схема T-trigger

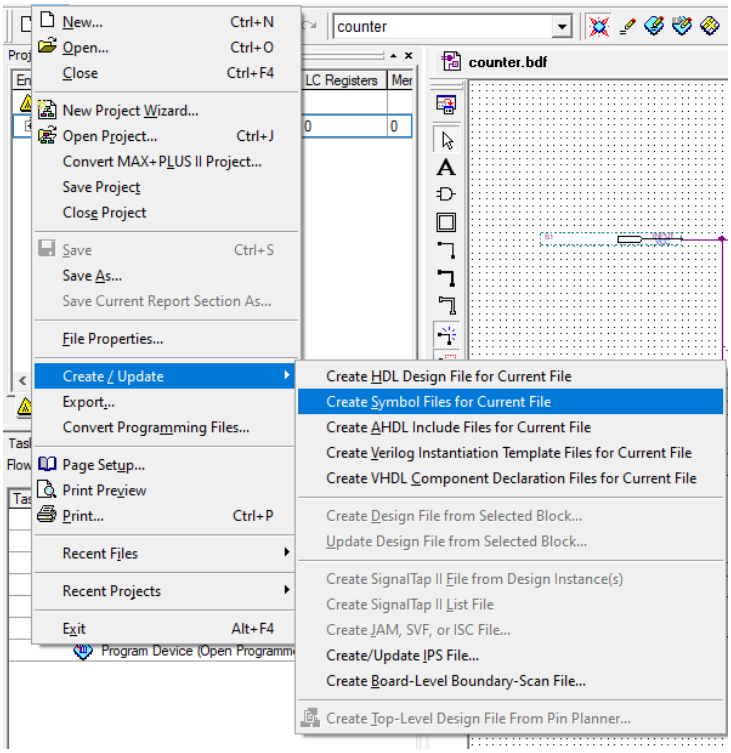

Рисунок 2.3 – Создание символа T-trigger

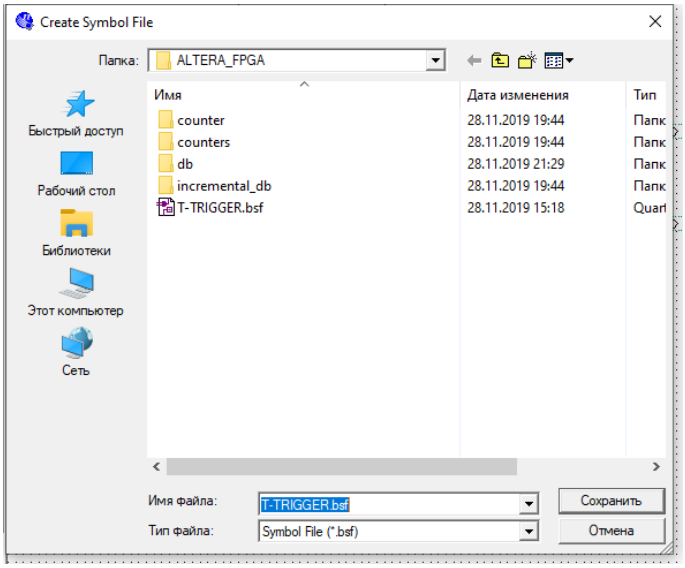

Рисунок 2.4 – Сохранение полученного символа

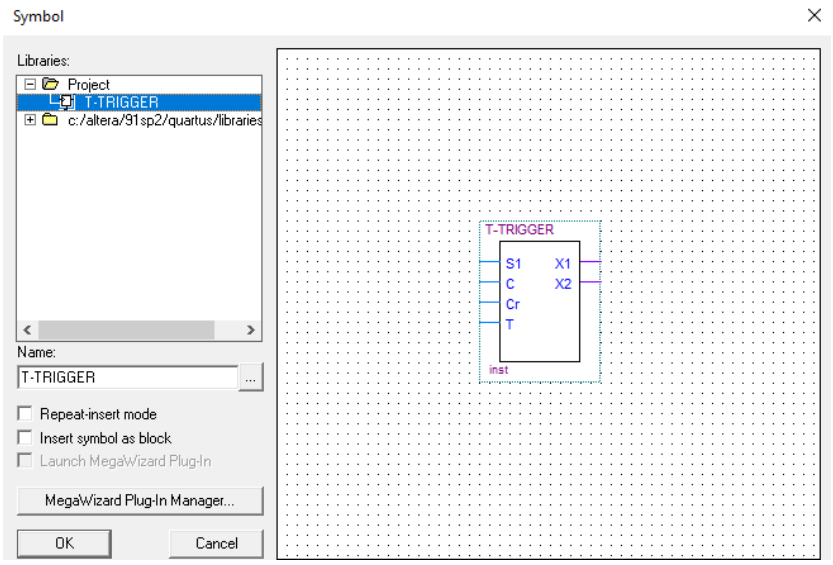

Рисунок 2.5 – Добавление в рабочую область символа T-trigger

Создание библиотечных модулей позволяет упросить разработку более сложных схем.

Описание лабораторной модели

Учебная система CIC-560 основана на новейших цифровых технологиях. Макет предоставляет широкий спектр экспериментов по разработке и применению как базовых, так и сложных логических схем.

Учебный комплекс включает: аналого-цифровые и цифроаналоговые преобразователи; клавиатуру; жидкокристаллический дисплей; интерфейс SCI; светодиоды; 8-разрядный семисегментный светодиодный дисплей; схемы управления шагового двигателя, а также двигателя постоянного тока. Данная комплектация позволяет использовать систему в учебных курсах по электронике, электротехнике, информатике, связи, автоматизации. Также учебный комплекс можно использовать для разработки микросхем и программного обеспечения, разработки и проверки базовых и сложных цифровых схем, изучения цифровой обработки сигналов и центрального процессорного устройства на микросхемах программируемой пользователем вентильной матрице с большим количеством элементов и выводов.

Внешний программатор предназначен для записи (программирования) данных в постоянное запоминающее устройство, а также процесса считывания информации. Также программатор может обладать рядом дополнительных функций: удаление, защита от считывания, защита от перепрограммирования и др. Существует два критерия, по которым определяют, поддерживает ли устройство микросхему:

– обеспечение работы с микросхемой во всех доступных режимах, которые были заложены разработчиками микросхемы;

– алгоритмы реализуются исключительно в рамках спецификации микросхемы.

Для реализации заложенных функций программатор должен иметь:

– колодку, обеспечивающую электрический контакт с выводами микросхемы;

– интерфейс, обеспечивающий ввод/вывод записи и считывание данных;

– программно-аппаратные драйверы, производящие формирование и считывание логических уровней и сложных тактовых сигналов.

USB Blaster Rev.c компании Altera предназначен для работы с программируемыми логическими интегральными схемами типов CPLD/FPGA. Внешний вид программатора представлен на рисунке 2.6.

Рисунок 2.6 – Загрузочный кабель для программирования ПЛИС типа CPLD и FPGA компании Altera USB Blaster Rev.c

Данный программатор имеет возможность поддержки следующих интерфейсов программирования: JTAG, Active Serial и Passive Serial.

JTAG – название рабочей группы по разработке стандарта IEEE 1149. Позднее это сокращение стало прочно ассоциироваться с разработанным этой группой специализированным аппаратным интерфейсом на базе стандарта IEEE 1149.1. Официальное название стандарта – Standard Test Access Port and Boundary-Scan Architecture. Интерфейс предназначен для подключения сложных цифровых микросхем или устройств уровня печатной платы к стандартной аппаратуре тестирования и отладки.

На данный момент интерфейс стал промышленным стандартом. Практически все сколько-нибудь сложные цифровые микросхемы оснащаются этим интерфейсом:

– для выходного контроля микросхем при производстве;

– тестирования собранных печатных плат;

– прошивки микросхем с памятью;

– отладочных работ при проектировании аппаратуры и программного обеспечения.

Метод программирования последовательных конфигурационных постоянный запоминающих устройств, который осуществляется через интерфейс JTAG, упрощает процесс отладки. В процессе разработки и отладки появляется необходимость в смене конфигурации FPGA, для этого применяют загрузку по интерфейсу JTAG с помощью загрузочных кабелей. При завершении процесса разработки появляется необходимость в обеспечении энергонезависимости. Для конфигурации FPGA в автономных устройствах существует два способа загрузки. Первый состоит в загрузке из внешнего параллельного постоянного запоминающего устройства или микропроцессора. Второй заключается в загрузке из последовательного конфигурационного постоянного запоминающего устройства, включающего два режима: Passive Serial и Active Serial. Используемый в данной работе режим Passive Serial в процессе отладки позволяет эмулироваться любым загрузочным кабелем Altera.

Ход работы

Для выполнения лабораторной работы необходимо изучить теоретический материал, позволяющий понимать принципы работы с программируемыми логическими интегральными схемами. Затем выполнить моделирование заданной схемы, компилировать проект, получить временные диаграммы и симуляцию разработанной схемы.

Создание проекта и графический интерфейс пользователя:

1. Загрузить программу Quartus II и выбрать во вкладке File пункт New Project Wizard (рисунок 2.7).

2. В появившемся окне New Project: Introduction нажать кнопку Next для продолжения создания проекта (рисунок 2.8).

3. Выбрать расположение проекта и имя, последняя строчка заполняется автоматически, далее нажать кнопку Next для продолжения (рисунок 2.9).

4. К текущему проекту есть возможность добавить файлы схем в форматах *.VHD, *.TDF, *.GDF. В нашем случае просто нажимаем кнопку Next для продолжения (рисунок 2.10).

5. В окне Family and Device Settings выбрать Cyclone в графе Family, также задать необходимые параметры в графах Package, Pin out и Speed grade как PQFP, 240 и 8 соответственно. В окне Available devices выбрать EP1C12Q240C8 (рисунок 2.11).

6. Нажать кнопку Finish для завершения создания проекта.

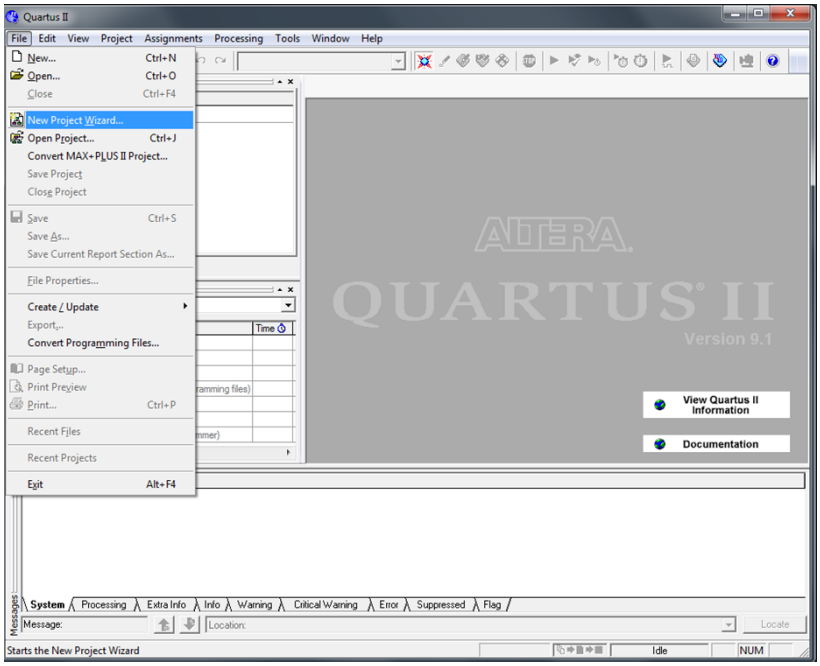

Рисунок 2.7 – Создание нового проекта с помощью мастера создания проекта Wizard

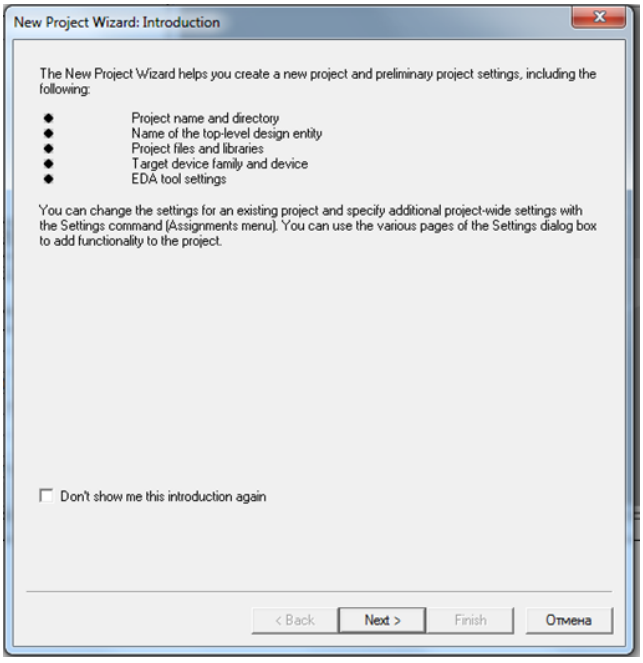

Рисунок 2.8 – Окно введения

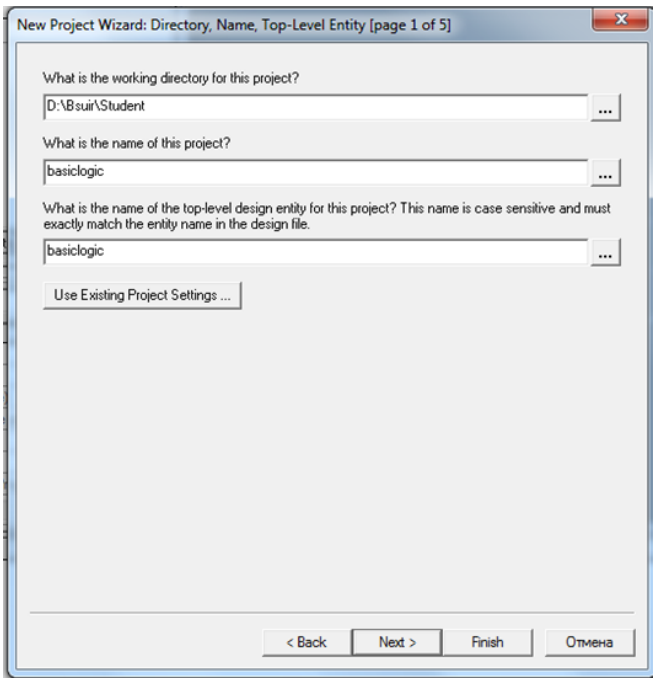

Рисунок 2.9 – Окно выбора расположения проекта, имени и объекта верхнего уровня

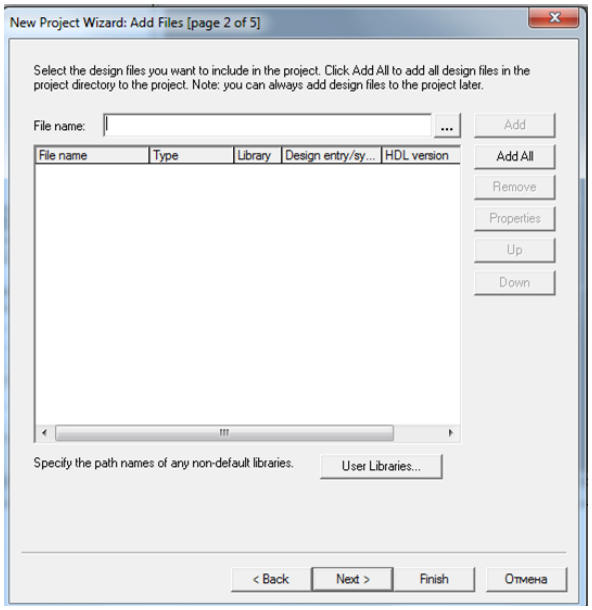

Рисунок 2.10 – Добавление файла в проект

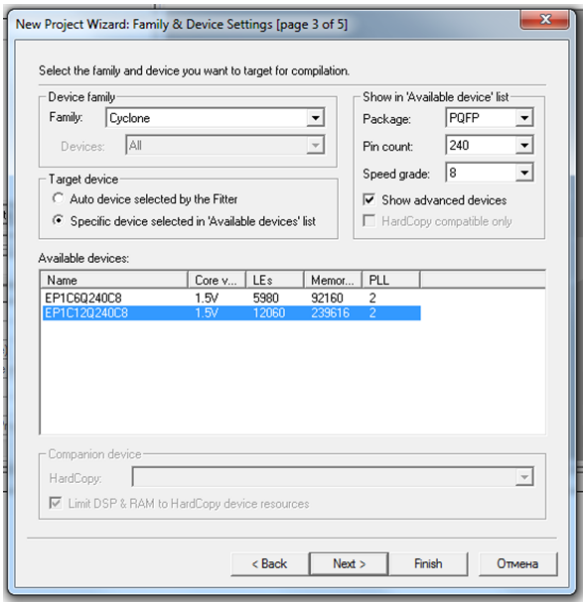

Рисунок 2.11 – Выбор устройства EP1C12Q240C8

Установка параметров проекта:

1. Выбрать во вкладке меню Assignments пункт Settings (рисунок 2.12).
2. В появившемся окне выбрать вкладку Device и нажать Device and Pin Options для продолжения установки (рисунок 2.13).

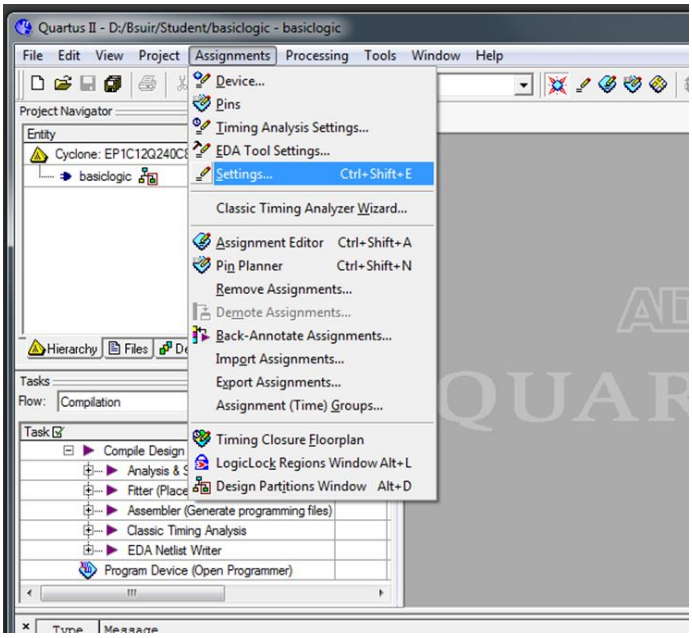

Рисунок 2.12 – Установка опций устройства и контактов

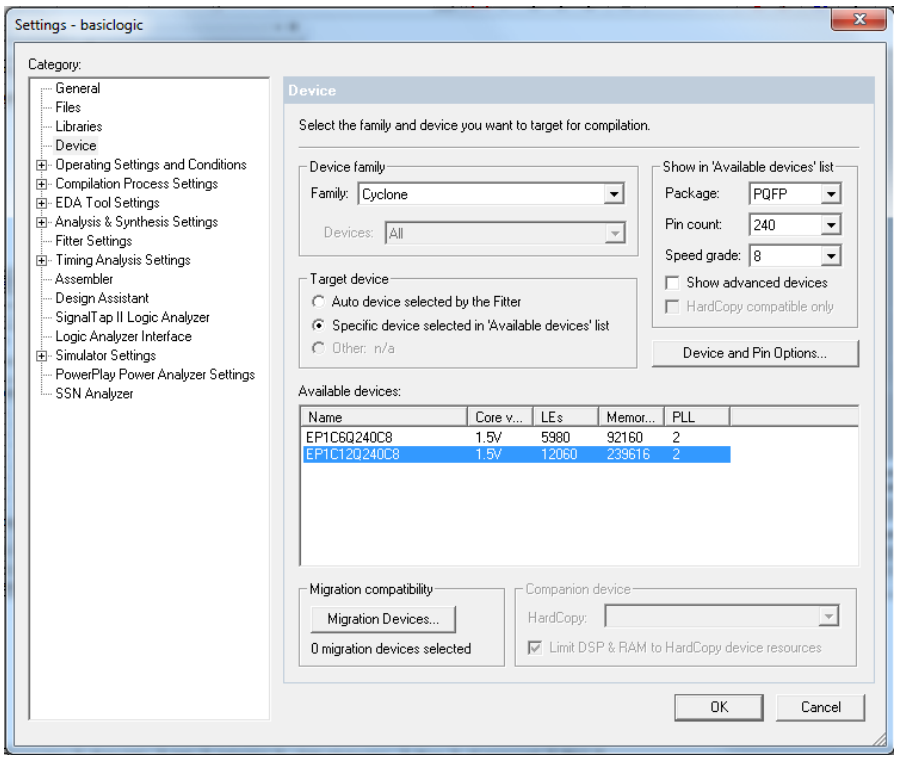

Рисунок 2.13 – Продолжение установки опций устройства и контактов

3. В появившемся окне Device and Pin Options выбрать вкладку Unused Pins. При разработке проекта неиспользуемые контакты устанавливаются как выходы, подключенные к земле (выбрать All output driving ground). Это необходимо для предотвращения паразитных излучений, а также уменьшения потребляемой мощности (рисунок 2.14).

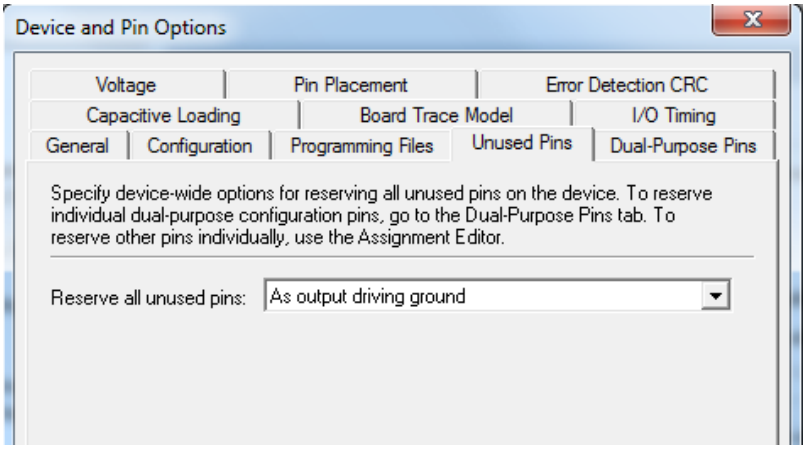

Рисунок 2.14 – Установка неиспользуемых контактов

4. Выбрать вкладку Configuration для установки конфигурации. В поле Configuration Scheme выбрать Passive Serial (can use Configuration Device). В Configuration Device выбрать EPC2 (рисунок 2.15).

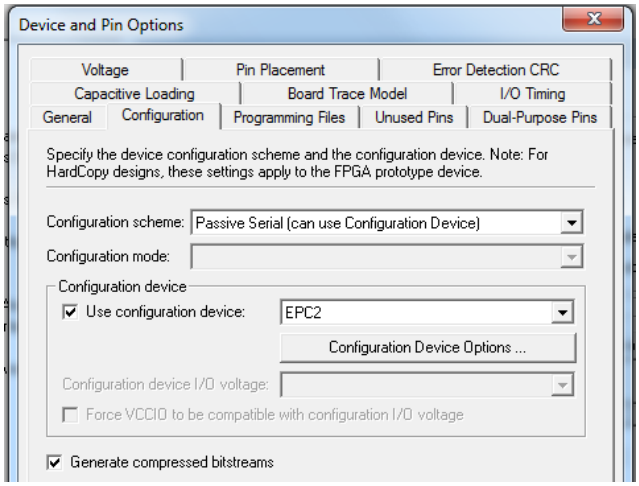

Рисунок 2.15 – Конфигурация устройства

5. Перейдя на вкладку Voltage, установить стандартные уровни напряжения ввода/вывода и выбрать 3.3VLVCMOS. Для завершения нажать OK (рисунок 2.16).

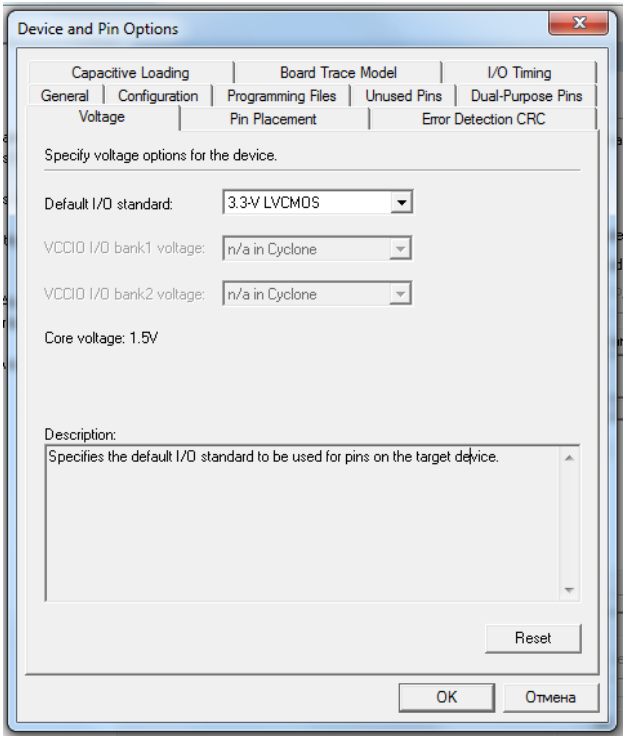

Рисунок 2.16 – Установки контактов ввода/вывода
в состояние 3.3VLVCMOS по умолчанию

6. Перейдя на ветку Compilation Process Settings, установить параметры, как показано на рисунках 2.17 и 2.18. Установки позволят выполнять повторную компиляцию для ускорения работы устройства.

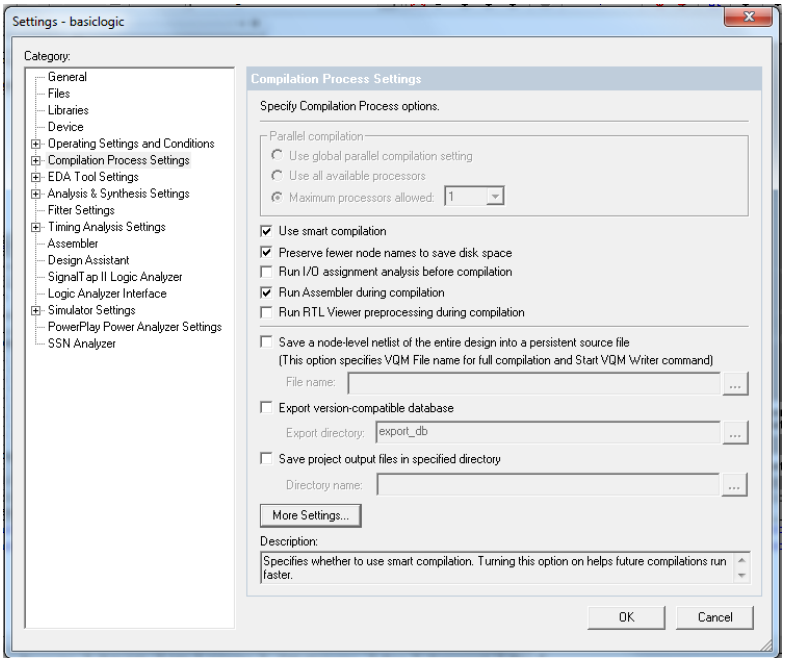

Рисунок 2.17 – Основные установки процесса компиляции

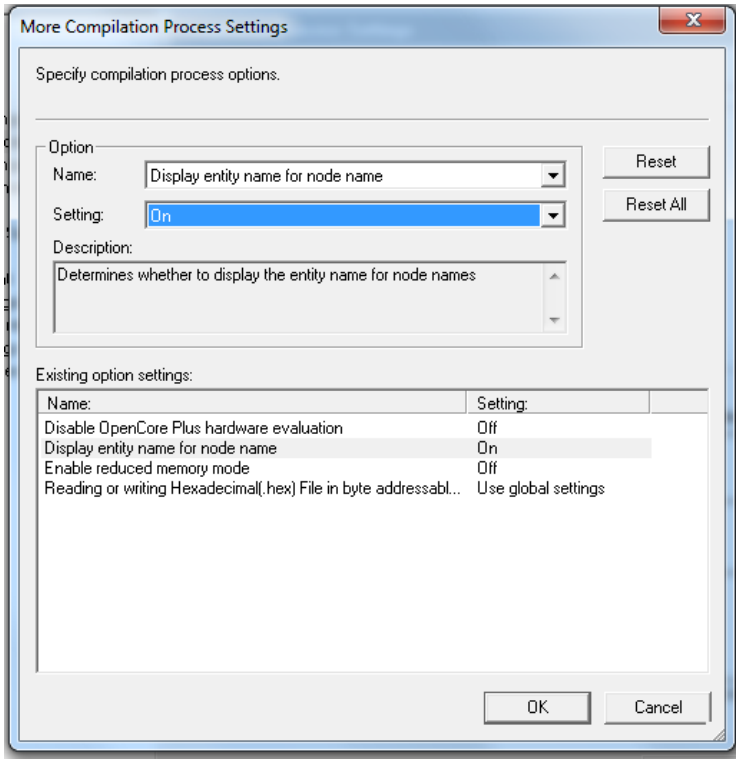

Рисунок 2.18 – Дополнительные установки процесса компиляции

7. После завершения нажать кнопку OK для выхода из режима установок.

Создание графического файла схемы:

1. В меню выбрать вкладку File, нажать кнопку New для создания нового файла (рисунок 2.19).

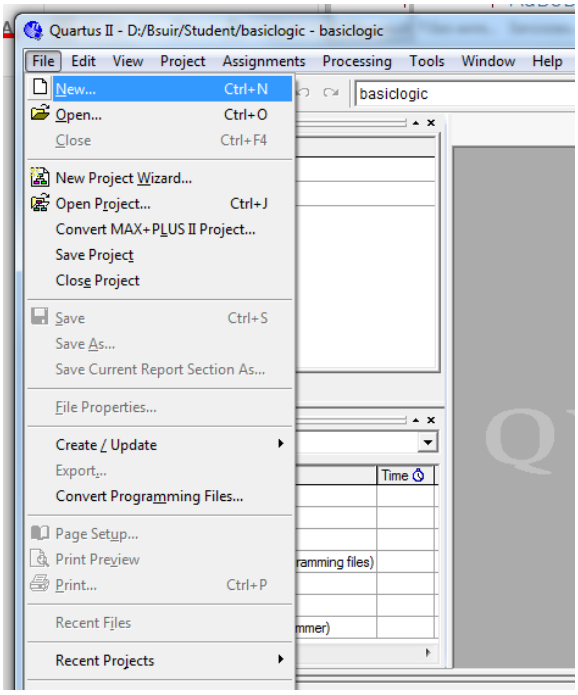

Рисунок 2.19 – Создание нового файла схемы

2. Выбрать Block Diagram/Schematic File и нажать OK (рисунок 2.20).

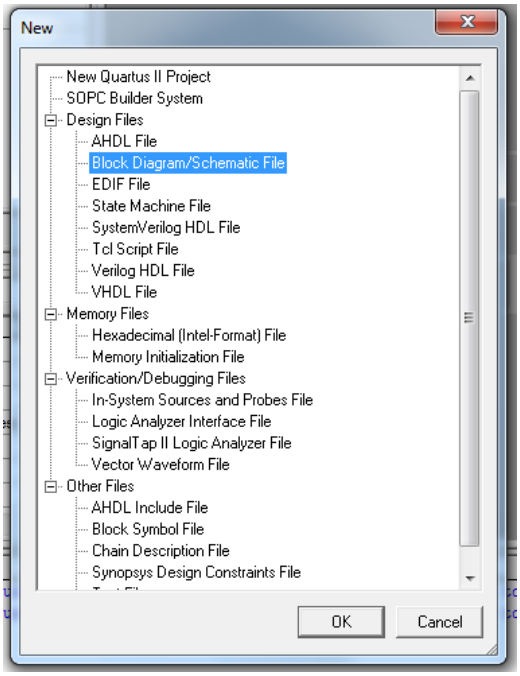

Рисунок 2.20 – Выбор файла Block Diagram/Schematic File

3. Далее в рабочей области появится лист для графического изображения схемы с именем Block1.bdf (рисунок 2.21).

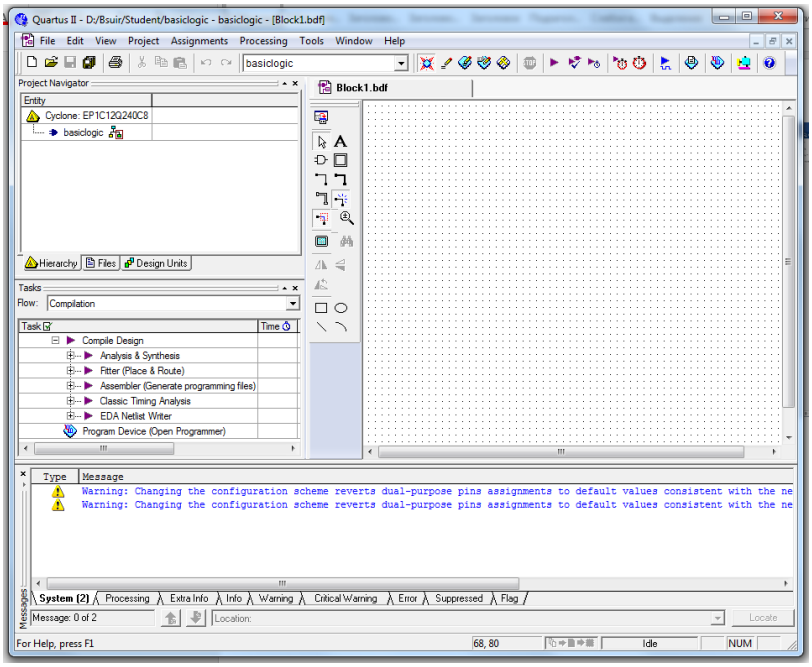

Рисунок 2.21 – Лист построения схемы в Quartus II

4. Двойное нажатие левой кнопкой мыши по рабочей области вызывает окно Symbol. Из дерева библиотеки выбрать логический элемент and2 и перенести его на страницу схемы (рисунок 2.22).

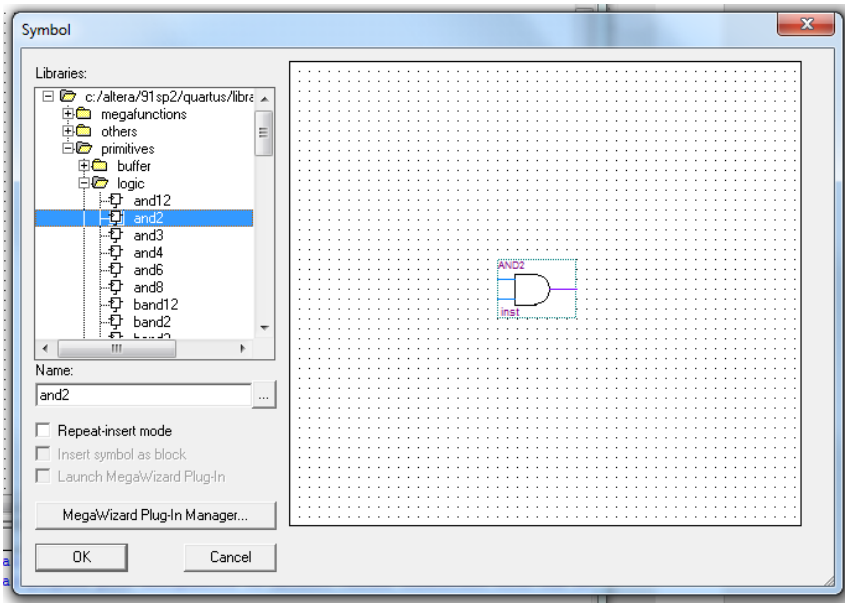

Рисунок 2.22 – Добавление логического элемента И

5. Далее в разрабатываемую схему необходимо добавить два входных контакта и один выходной (рисунки 2.23 и 2.24).

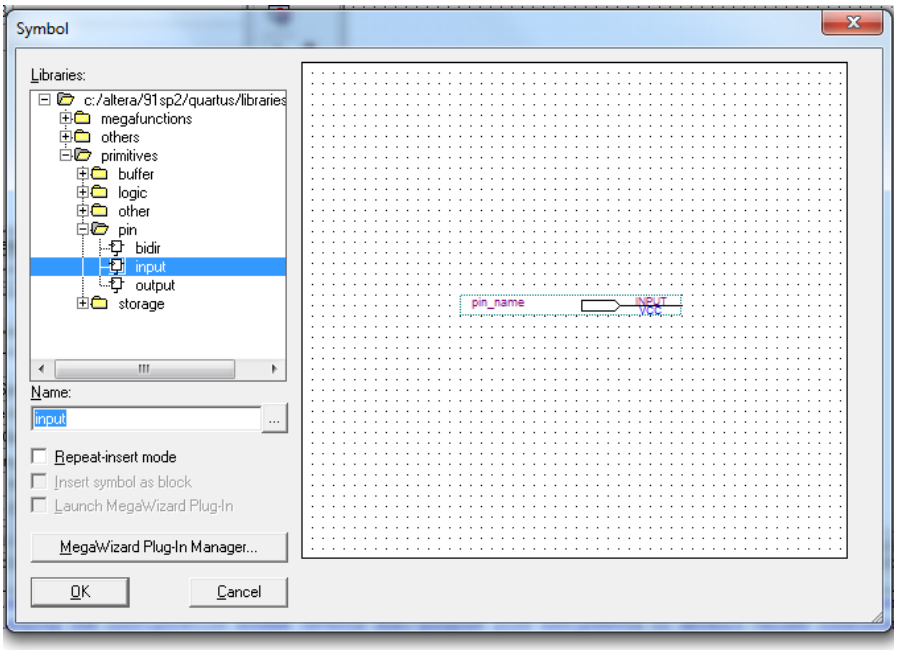

Рисунок 2.23 – Добавление входных контактов

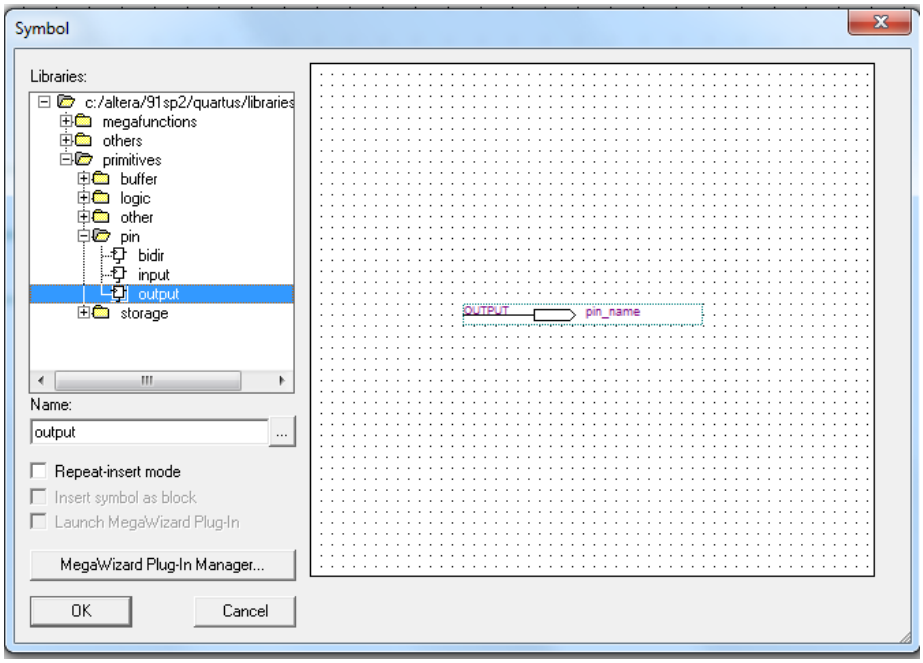

Рисунок 2.24 – Добавление выходного контакта

6. Соединить проводками входы/выходы логического элемента И с контактами (рисунок 2.25). Далее контакты переименовать уникальными названиями (рисунок 2.26).

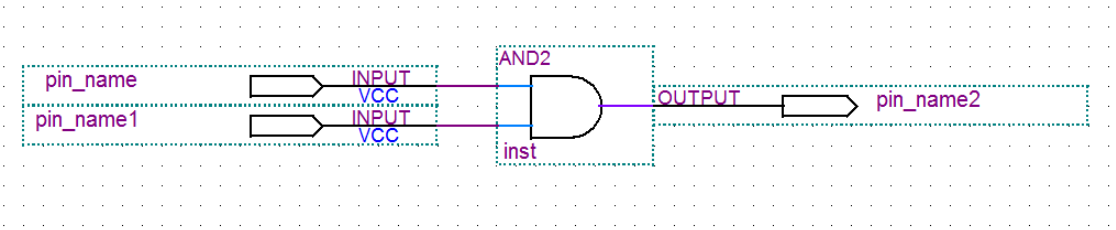

Рисунок 2.25 – Логический элемент И с двумя входными контактами и одним выходным

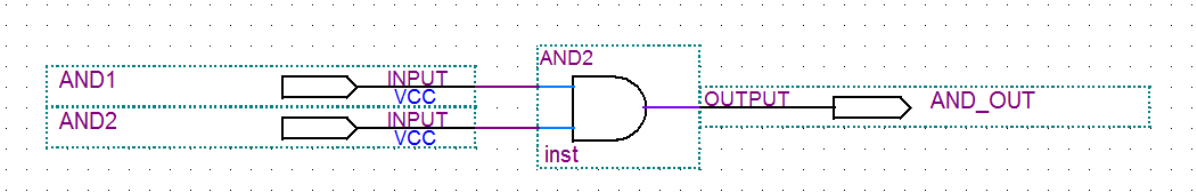

Рисунок 2.26 – Обозначение контактов уникальными именами

7. Для сохранения графического изображения схемы в меню File необходимо выбрать Save as, присвоив имя basiclogic.dbg (рисунки 2.27 и 2.28). Должен быть выбран блок Add file to current project.

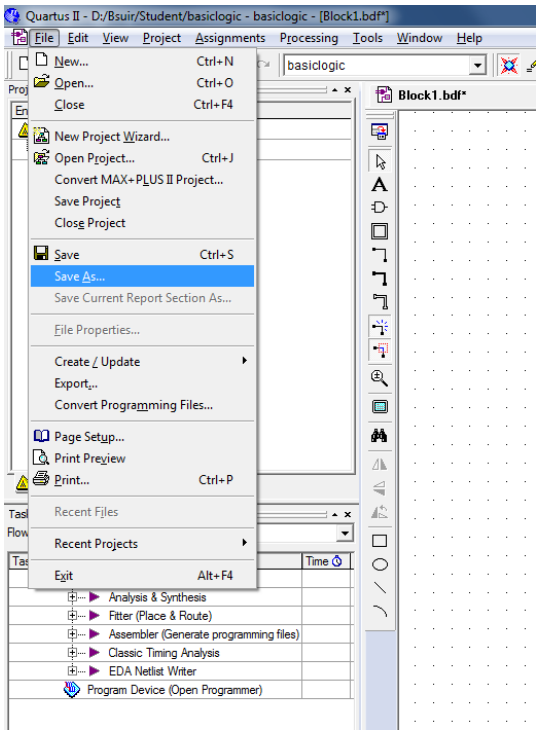

Рисунок 2.27 – Сохранение файла схемы

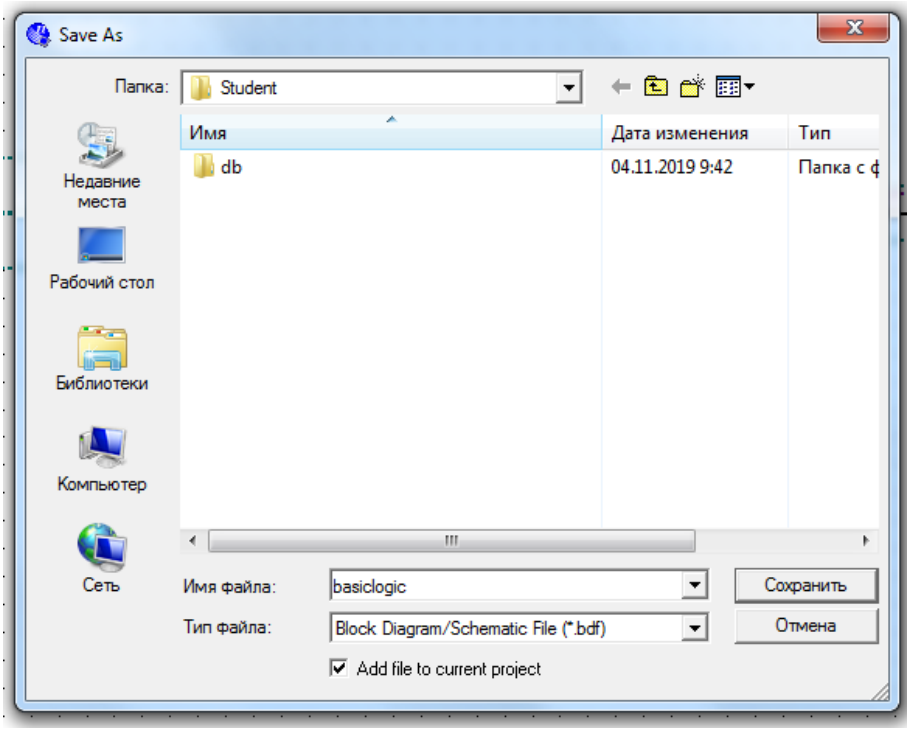

Рисунок 2.28 – Сохранение и добавление файла схемы в текущий проект

Компиляция проекта:

1. Для запуска процесса компиляции готовой схемы в меню выбрать Processing/Start Compilation (рисунок 2.29).

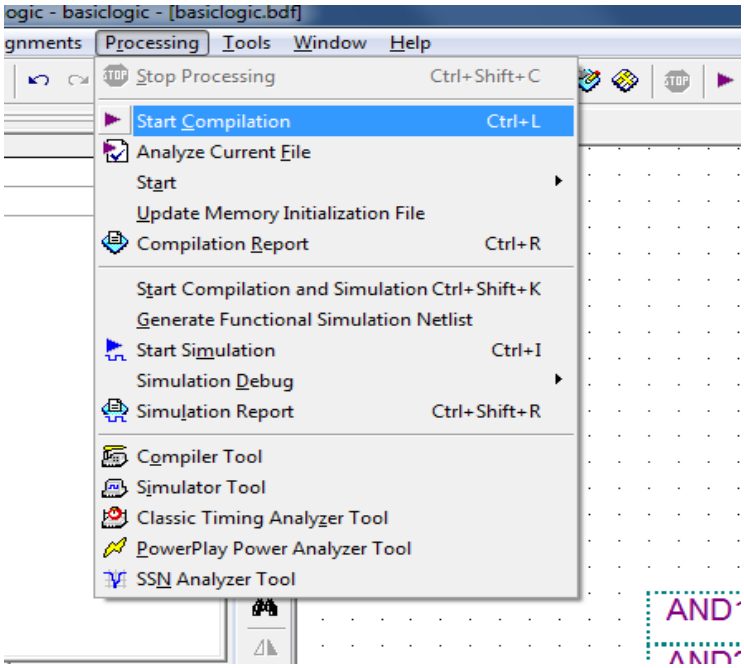

Рисунок 2.29 – Запуск процесса компиляции

При появлении ошибок необходимо их устранить, сохранить изменения и запустить начало компиляции повторно. Если компиляция завершится успешно, появится информация о компиляции проекта (рисунок 2.30).

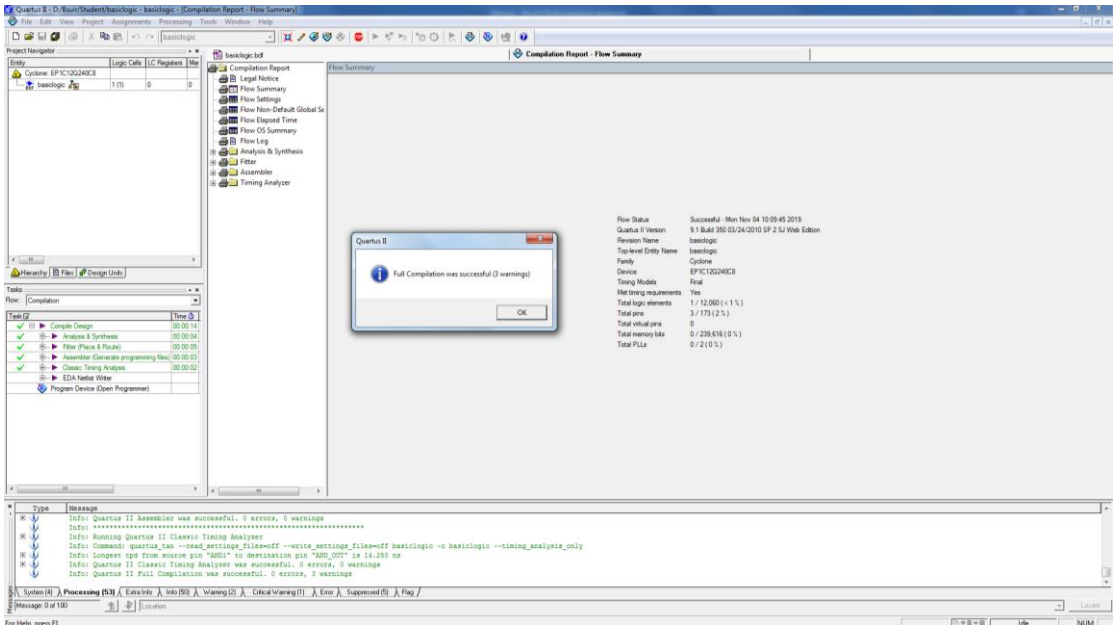

Рисунок 2.30 – Отчет о выполнении компиляции проекта

Назначение контактов устройства

Для проверки работоспособности собранной схемы на макете необходимо присвоить контакты на входы и выход схемы:

1. В меню выбрать вкладку Assignments/Pins (рисунок 2.31).

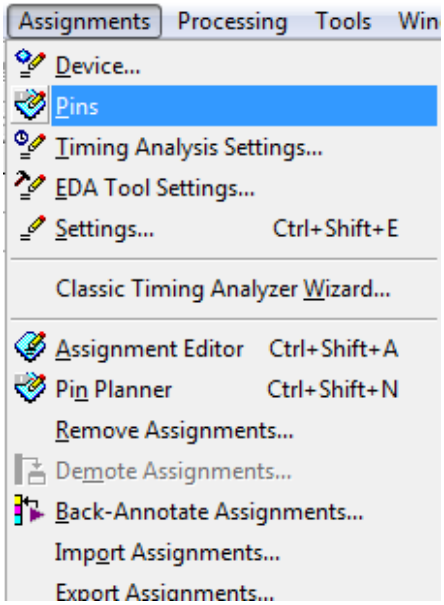

Рисунок 2.31 – Присвоение контактов устройства

2. В появившемся окне в графе Location задать контакты (рисунок 2.32).

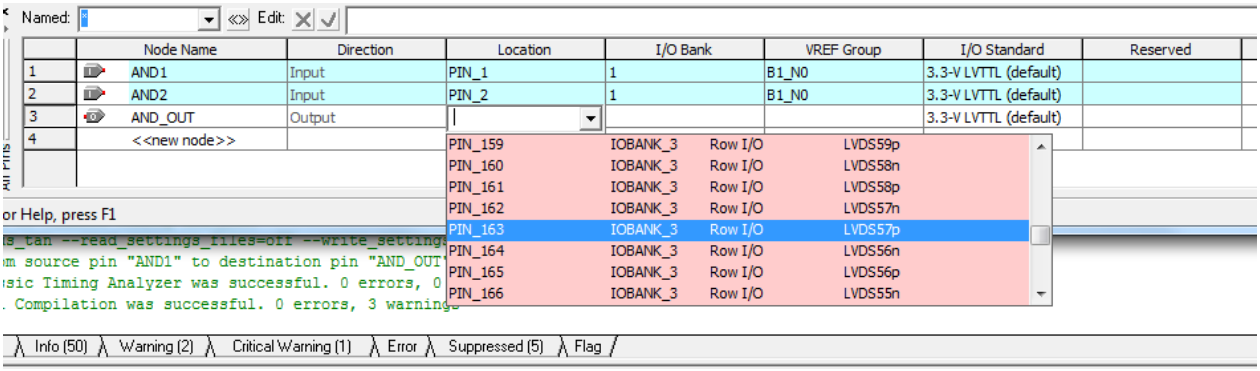

Рисунок 2.32 – Задание контактов устройства

Создание проекта с использованием языка VHDL:

1. Создать проект и файл (рисунок 2.33).

2. На рабочем листе воспроизвести код вычитающего счетчика (рисунок 2.34).

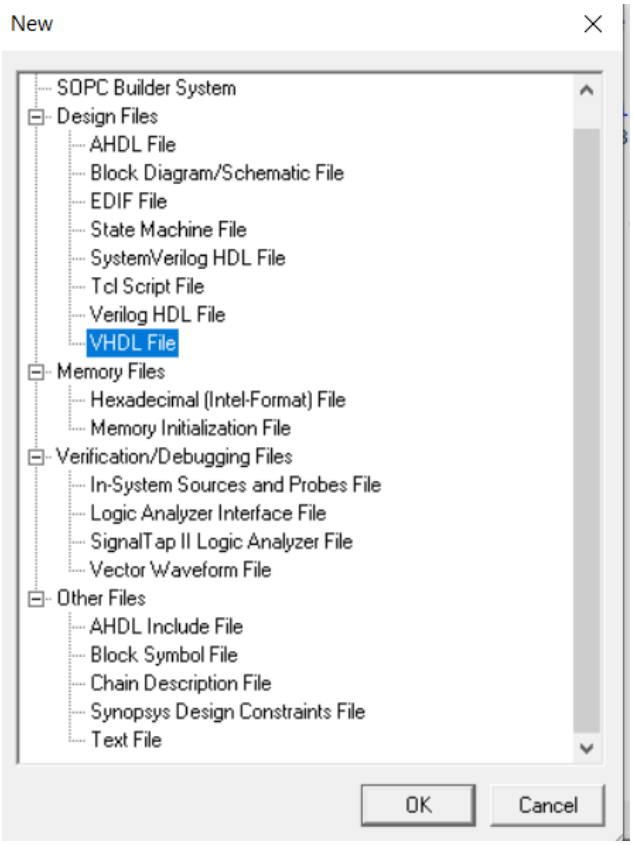

Рисунок 2.33 – Выбор файла Quartus II

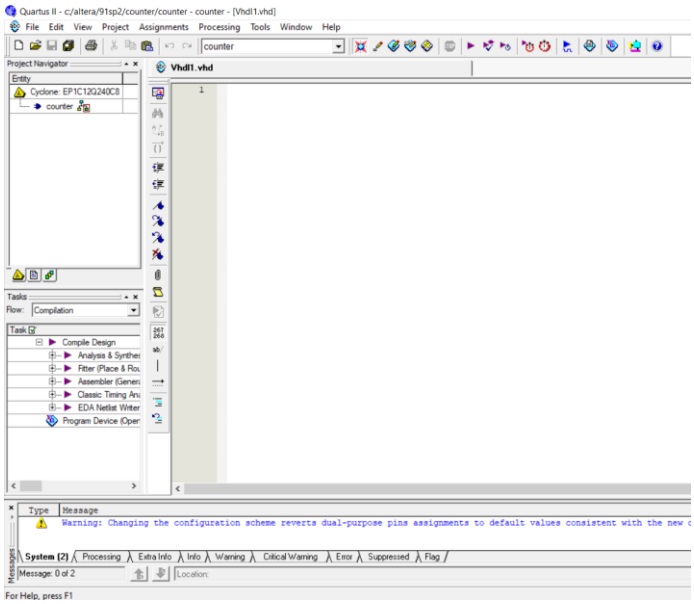

Рисунок 2.34 – Рабочее поле для программирования счетчика

Процесс создания проекта на языке *VHDL* представлен ниже.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity COUNTER is
port (clk, en, rst: in std_logic;
 count: out std_logic_vector (3 downto 0) := "0000");
end COUNTER;
architecture behav of COUNTER is
signal cnt: std_logic_vector (3 downto 0);
begin
  process (clk, en, cnt, rst)
  begin
   if (rst = '0') then
   cnt <= (others => '0');
   elsif (clk'event and clk = '1') then
   if (en = '1') then
    cnt <= cnt - 1;
  end if;
  end if;
  end process;
count <= cnt;
         end behav;
```

3. Для сохранения программы вычитающего счетчика файла в меню выбрать File/Save as и имя counter (рисунок 2.35).

4. Для создания блока counter выбрать вкладку counter.vhd, в верхней части окна выбрать File/Create/Update/Create symbol files for current files.

В целях проверки и дальнейшего использования необходимо создать новый файл и двойным нажатием по рабочей странице вызвать окно Symbol.

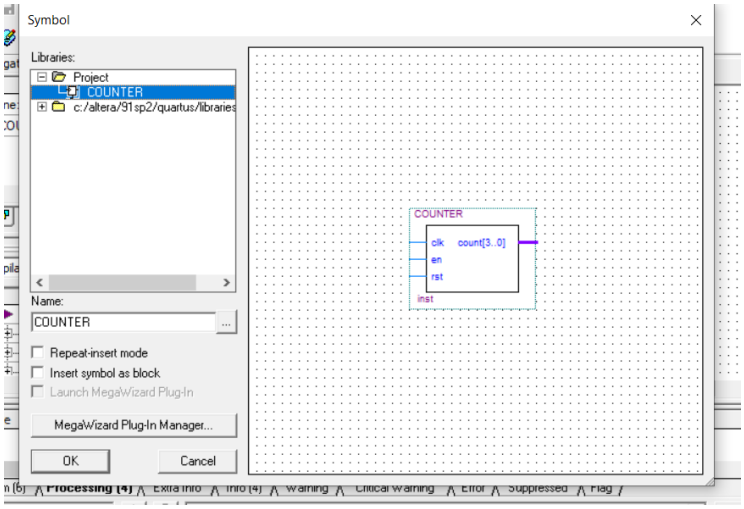

Рисунок 2.35 – Символ схемы вычитающего счетчика

Необходимо убедиться, что новый символ был добавлен в персональную библиотеку.

Создание проекта с использованием языка Verilog

Создадим генератор М-последовательности полинома $x^4 + x + 1$ с начальной последовательностью «0110» (для этих импульсов зададим тактовую частоту следования $f_t = 1$ кГц, получим временные диаграммы (графики), характеризующие работу фильтра):

1. Создать проект и файл (рисунок 2.36).

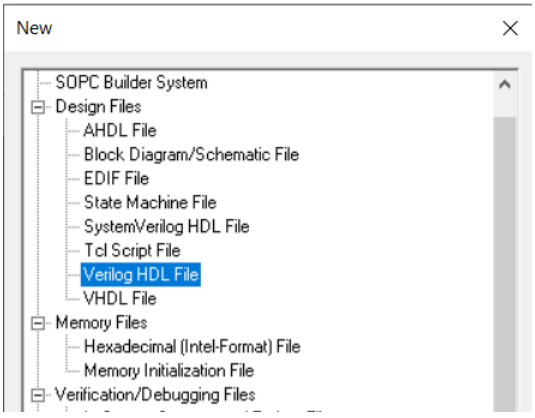

Рисунок 2.36 – Выбор файла Verilog HDL

2. На рабочем листе воспроизвести код генератора М-последовательности и провести компиляцию проекта (рисунок 2.37).

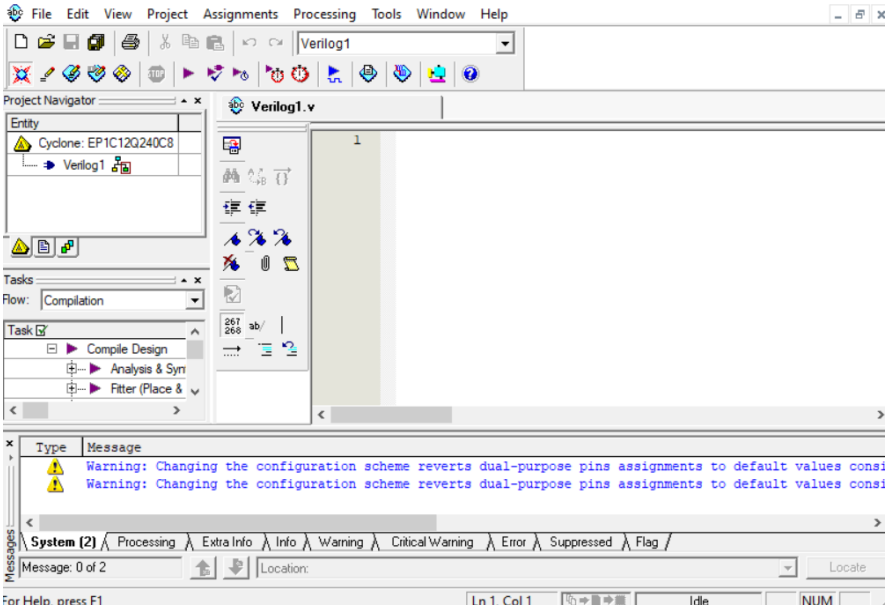

Рисунок 2.37 – Рабочее поле для программирования

Построение генератора М-последовательности на языке Verilog:

```
module m_sequence_generator (
 input clk, -- порт тактирования
 output reg [4:1] out, -- 4-битный регистр
 input rst -- порт сброса
 );
  wire feedback;
  assign feedback = out[4] ^ out[1]; -- обратная связь
  always @(posedge clk, posedge rst)
 begin
 if (rst)
 out = 4'b0110; -- начальная комбинация
 else
 out = {out[3:1], feedback};
 end
 endmodule
```

Проверка работы генератора М-последовательности на языке Verilog:

```
module m_seguence_generator_tb;
 reg clk_tb;
 reg rst_tb;
 wire [4:1] out_tb;
```

```
 initial
begin
clk_tb = 0;
rst_tb = 1;
#5 rst_tb = 0;
end
always
begin
#10;
clk_tb = ~ clk_tb;
end
m_sequence_generator
generator(.out(out_tb),.clk(clk_tb),.rst(rst_tb));
endmodule
```

3. Компиляция и моделирование временных диаграмм выполняется аналогичным образом.

4. Результат моделирования приведен на рисунке 2.38.

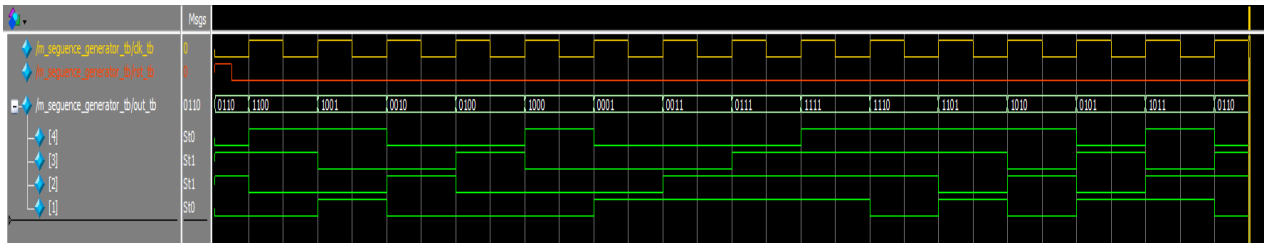

Рисунок 2.38 – Временная диаграмма работы собранного генератора М-последовательности

Синтез схемы вычитающего счетчика с помощью графического файла

Создать новый проект, выбрать Block Diagram/Schematic File. Собрать схему, представленную на рисунке 2.39.

Для этого необходимо выбрать нужные элементы по названиям (vcc, input, output, TFF). В окне Symbol в строке Name ввести название элемента и добавить его на рабочее поле (рисунок 2.40).

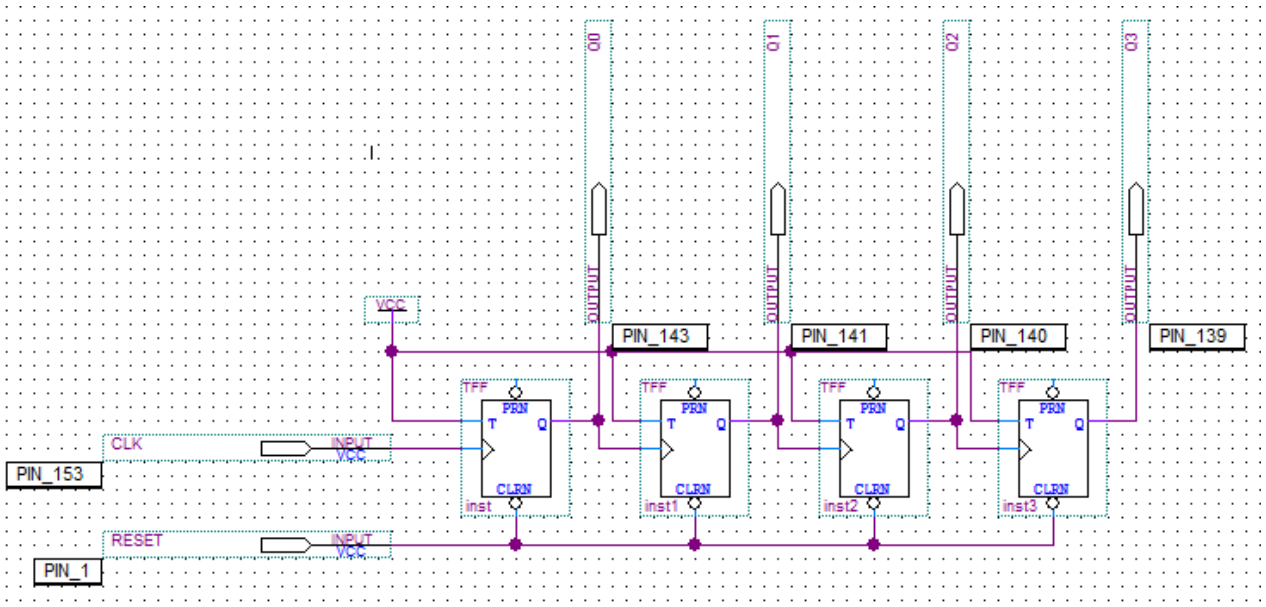

Рисунок 2.39 – Схема вычитающего счетчика

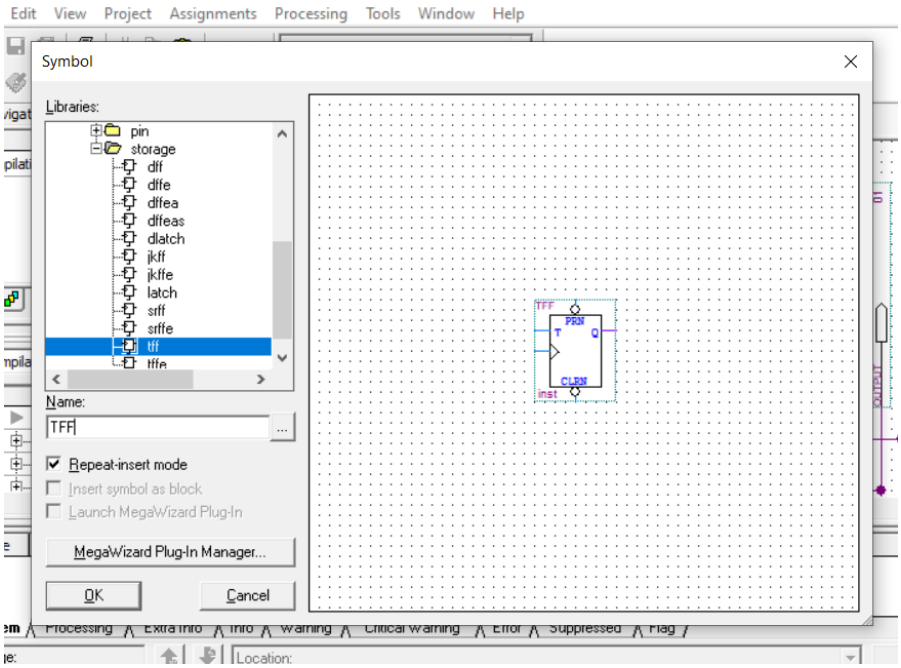

Рисунок 2.40 – Добавление элемента на рабочее поле

Моделирование работы временных диаграмм:

1. Для моделирования временных диаграмм в проекте выбрать Processing/Simulator Tool (рисунок 2.41).

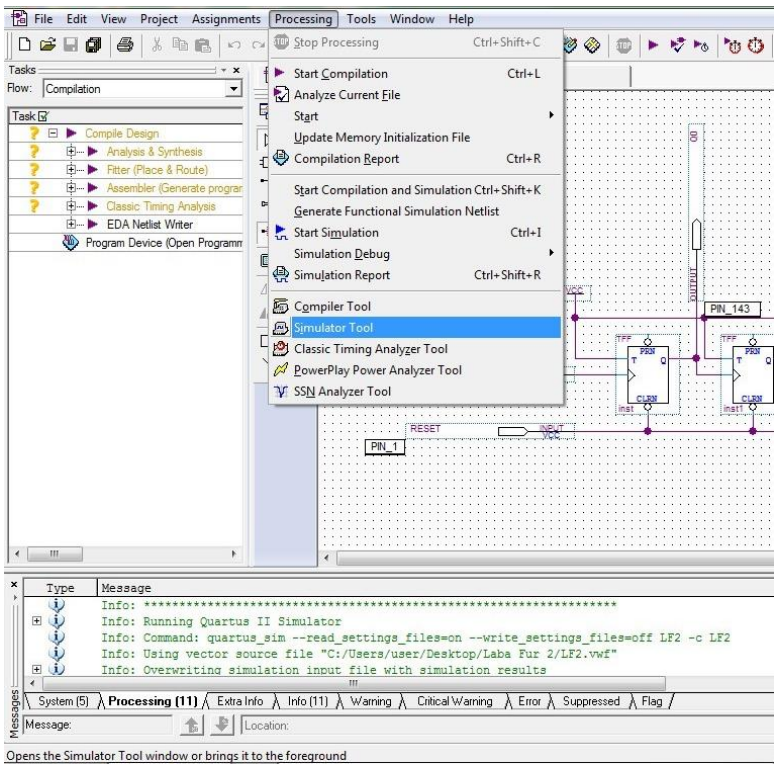

Рисунок 2.41 – Выбор симулятора в меню

2. В меню проверьте установку Overwrite simulator input file with simulator result (перезаписать входной файл моделирования с результатами моделирования), нажать кнопку Open для редактирования формы входного колебания моделирования (рисунок 2.42).

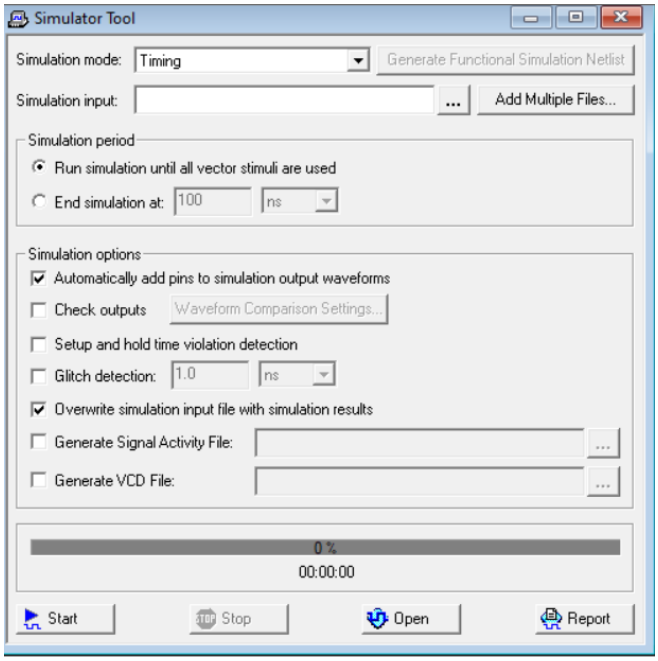

Рисунок 2.42 – Редактирование формы входного колебания для моделирования

3. В графе Edit выбрать строчку Insert, затем Insert Node or Bus (установка шины узла) (рисунок 2.43).

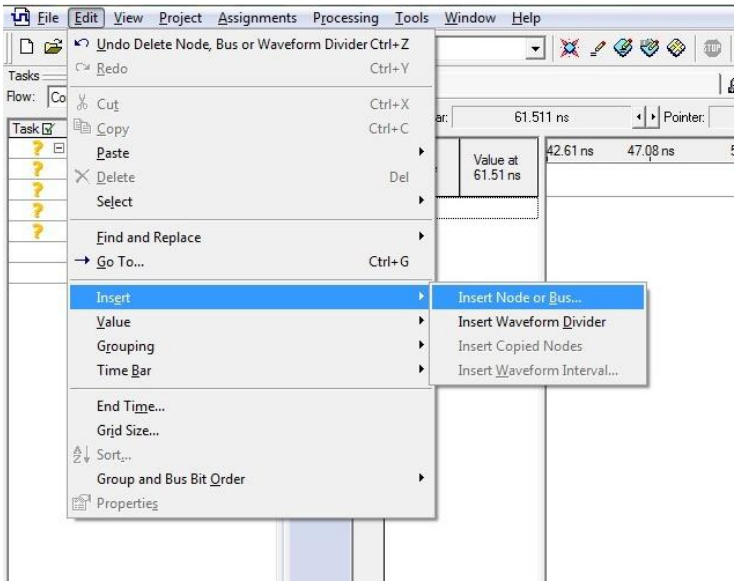

Рисунок 2.43 – Установка узла или шины для моделирования

4. Нажать кнопку Node Finder (поиск узла) для обнаружения всех сигнальных узлов или шин, которые необходимы при выполнении моделирования (рисунок 2.44).

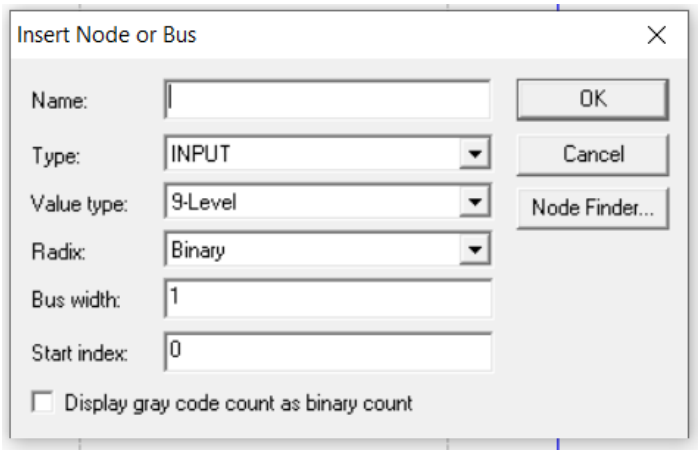

Рисунок 2.44 – Окно вставки узла или шины

5. В появившемся окне в категории Filter выбрать Pins: all и нажать кнопку List (рисунок 2.45).

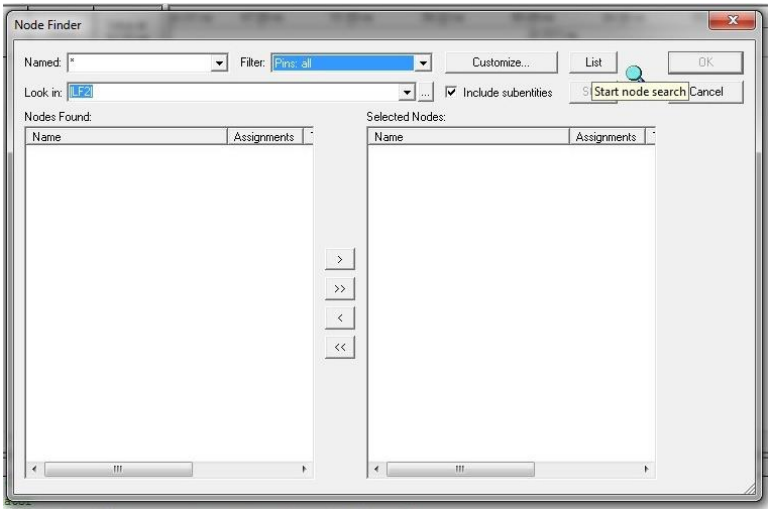

Рисунок 2.45 – Окно поиска узла

6. Все узлы, используемые в проекте, будут перечислены в поле Nodes Found. Нажать кнопку (») для копирования всех найденных узлов в поле Selected Nodes и нажимаем кнопку OK для завершения. Далее в окне Insert Node or Bus также нажать OK (рисунок 2.46).

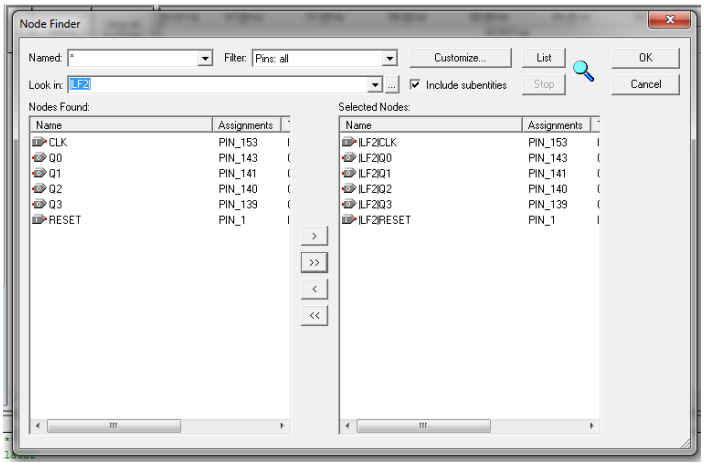

Рисунок 2.46 – Копирование найденных узлов

7. Выбранные контакты установлены в левой части окна моделирования. Выбрать Edit/End time в меню для установки полного времени моделирования (рисунок 2.47).

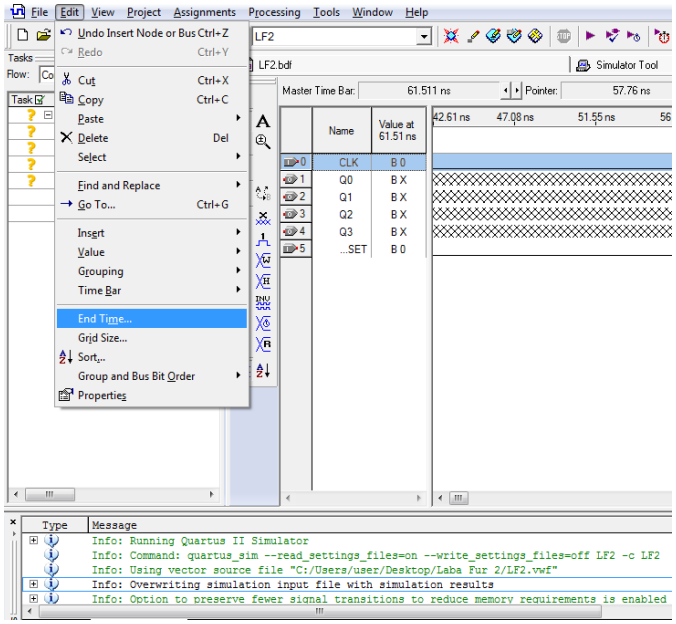

Рисунок 2.47 – Выбор времени окончания для установки временного моделирования

8. Установить время окончания моделирования равным 1 ms во вкладке Time и нажать OK после завершения операций (рисунок 2.48).

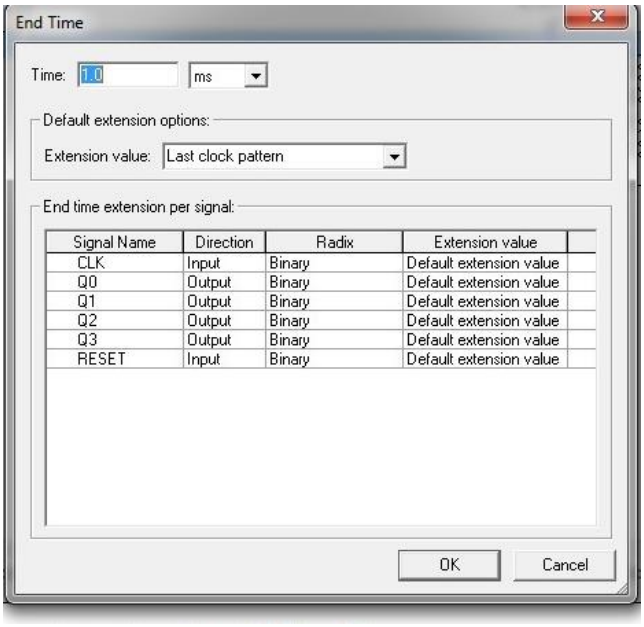

Рисунок 2.48 – Окно установки времени окончания моделирования

9. Выбрать Edit/Grid Size в меню для установки временного периода 100 ns (рисунки 2.49 и 2.50).

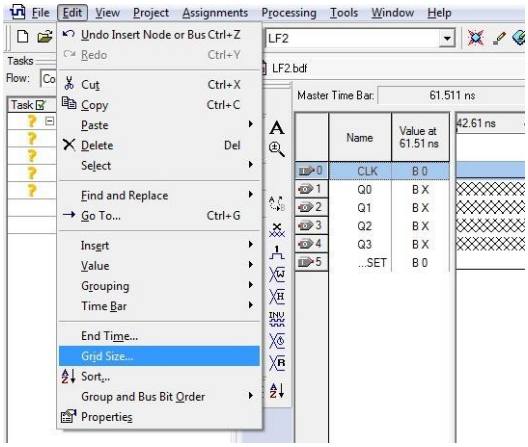

Рисунок 2.49 – Установка размера сетки

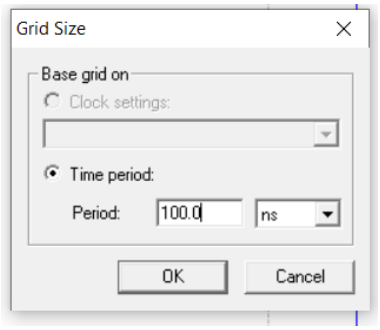

Рисунок 2.50 – Окно установки размера сетки

10. Установить форму входных колебаний на входных контактах (clk, reset) для моделирования. В левой стороне экрана расположен список значков, позволяющих редактировать форму колебаний (рисунок 2.51).

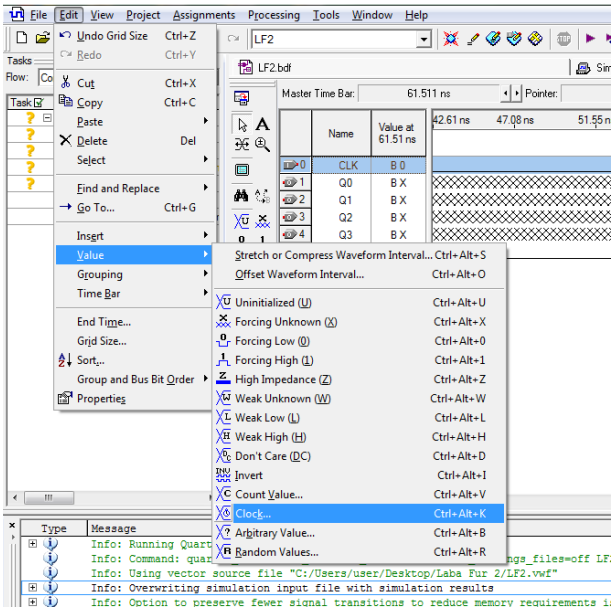

Рисунок 2.51 – Установка формы входных колебаний

11. Для редактирования формы сигнала CLK необходимо выделить clk и затем установить желаемую частоту. Установить временной период на входе clk, равный 100 ns (рисунки 2.52 и 2.53).

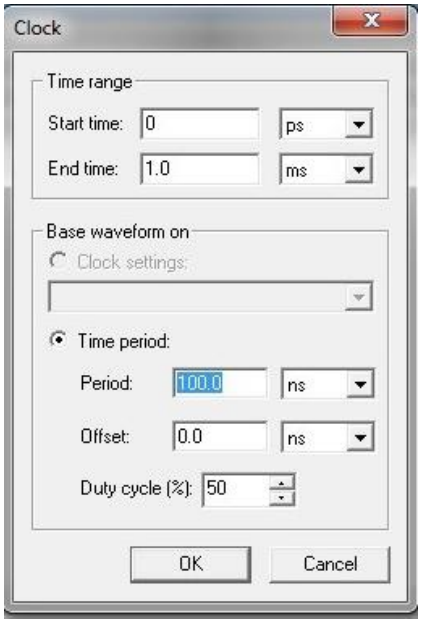

Рисунок 2.52 – Окно установки частоты на входе clk

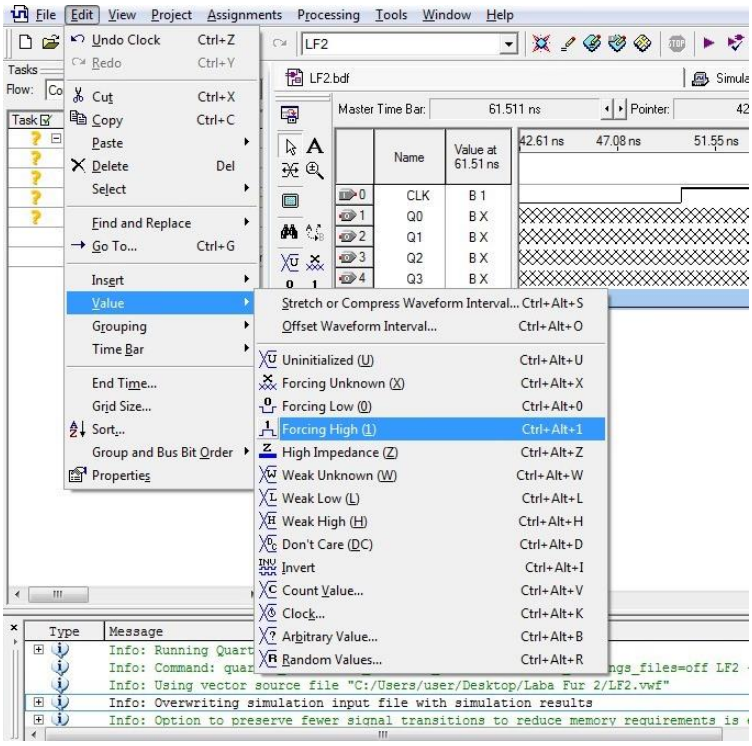

Рисунок 2.53 – Установка формы входных колебаний

12. Сохранить форму колебаний перед началом процесса моделирования (рисунок 2.54).

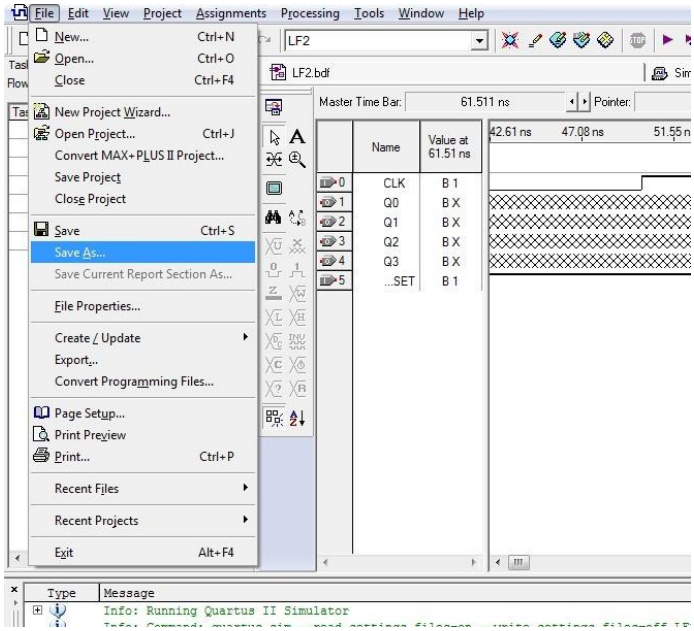

Рисунок 2.54 – Сохранение формы колебаний

13. Вернуться в окно Simulation Tool. Перейти к сохраненному файлу моделирования. Нажать кнопку Start для начала моделирования (рисунки 2.55 и 2.56).

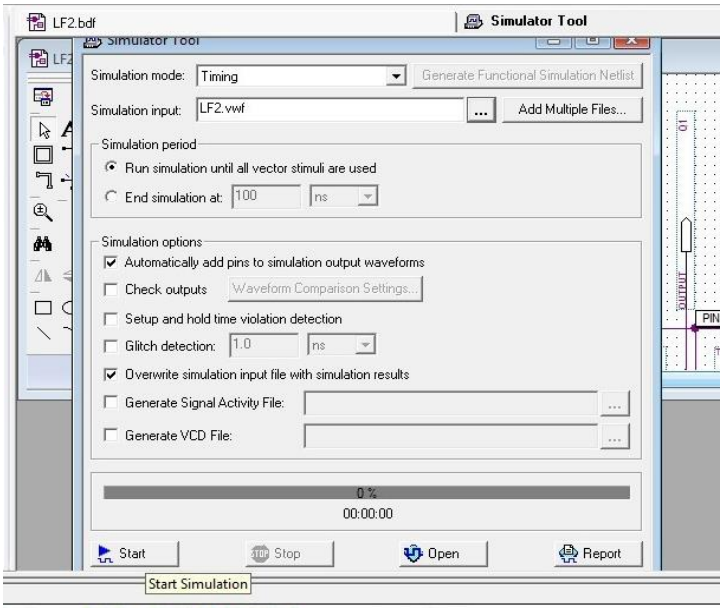

Рисунок 2.55 – Запуск моделирования

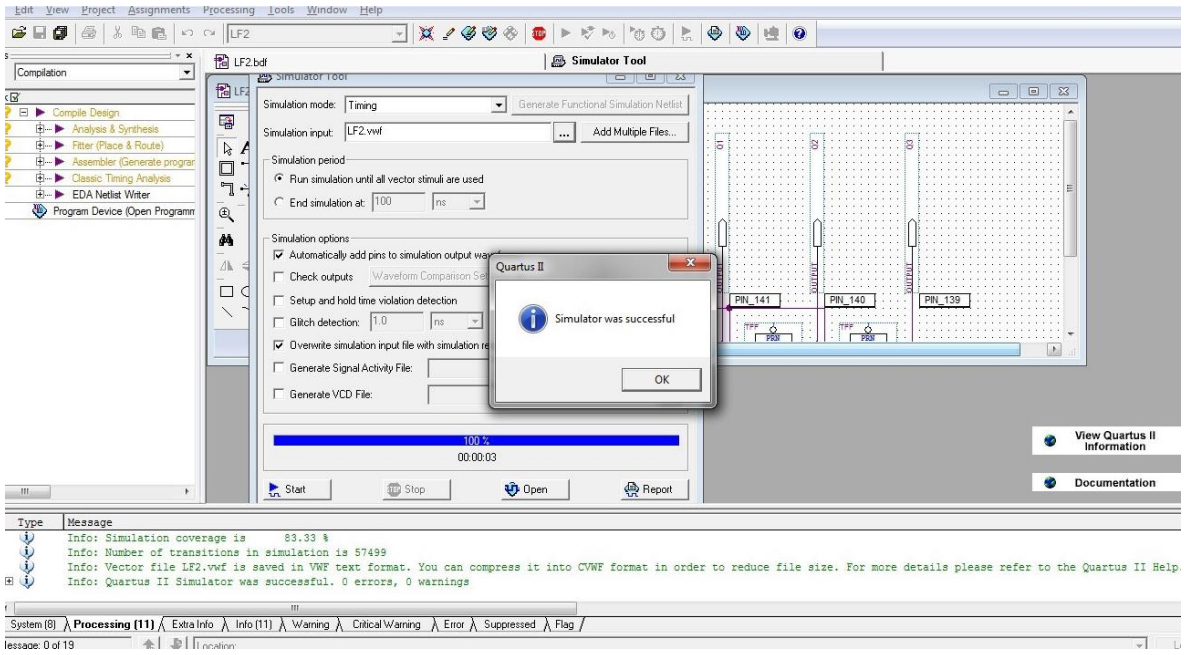

Рисунок 2.56 – Результат моделирования

14. Выбрать Processing/Start Simulation или нажать кнопку (обведена на рисунке 2.57) для начала моделирования.

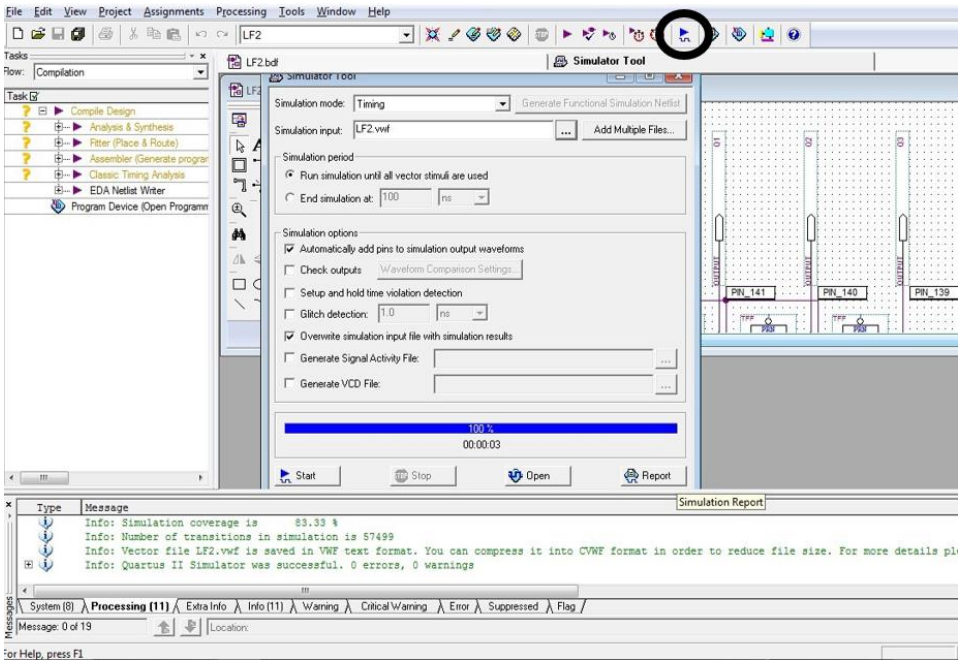

Рисунок 2.57 – Начало моделирования временных диаграмм

15. После завершения процесса моделирования для проверки результата моделирования нажать кнопку Report (обведена на рисунке 2.58).

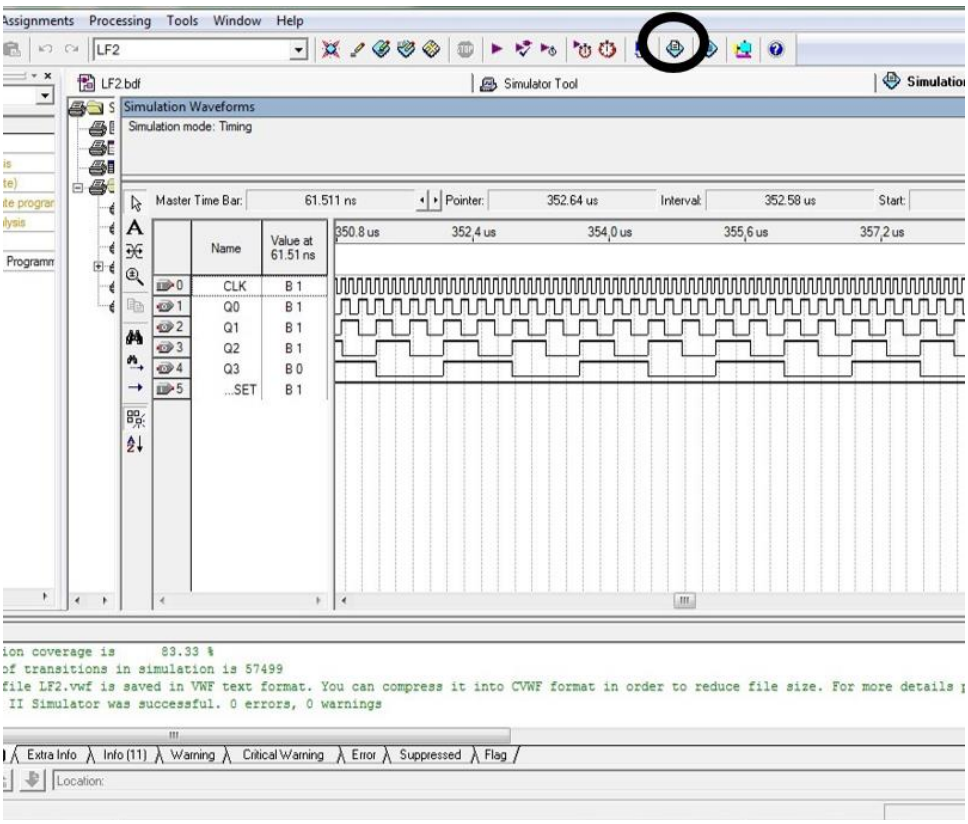

Рисунок 2.58 – Проверка результатов моделирования

16. После успешного выполнения вышеуказанных пунктов должен получиться результат, приведенный на рисунке 2.59.

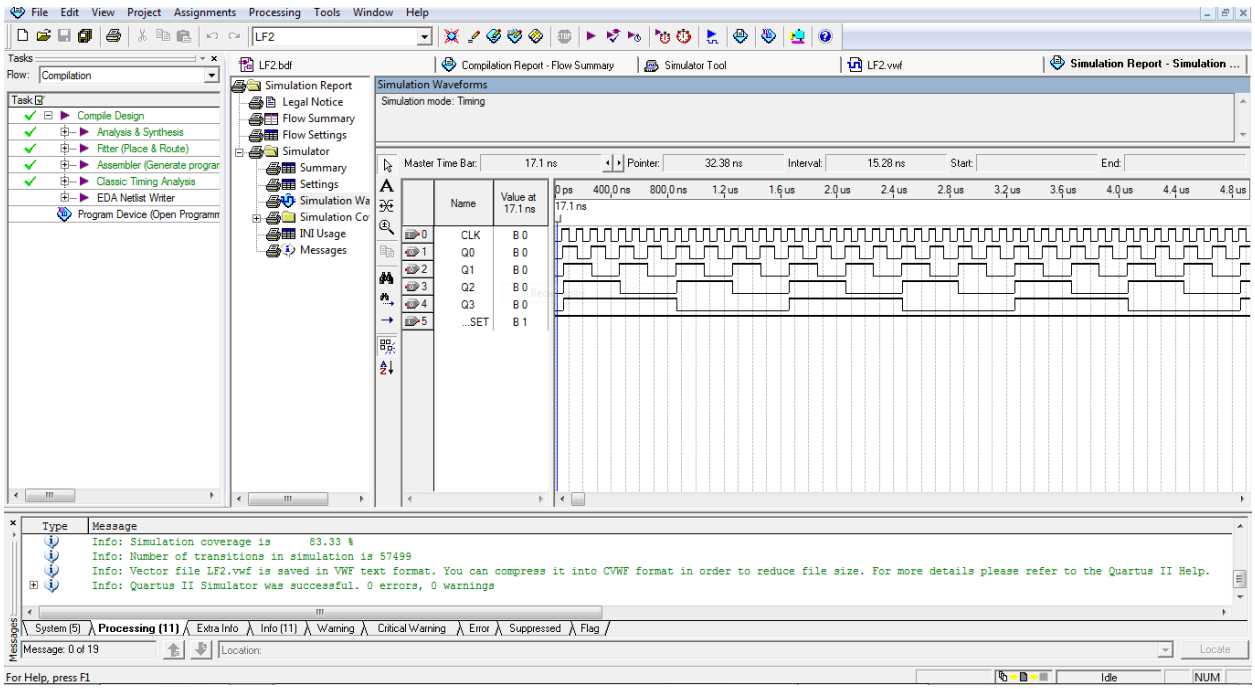

Рисунок 2.59 – Результат моделирования временных диаграмм

Симуляция разработанной схемы на макете:

1. Для проверки работы схемы на учебном макете CIC-560 необходимо присвоить контакты. В графе Location это можно сделать двумя способами: дважды щелкнуть левой кнопкой мыши и вести нужный номер с макета; щелкнуть один раз по строке и выбрать необходимый номер (рисунок 2.60).

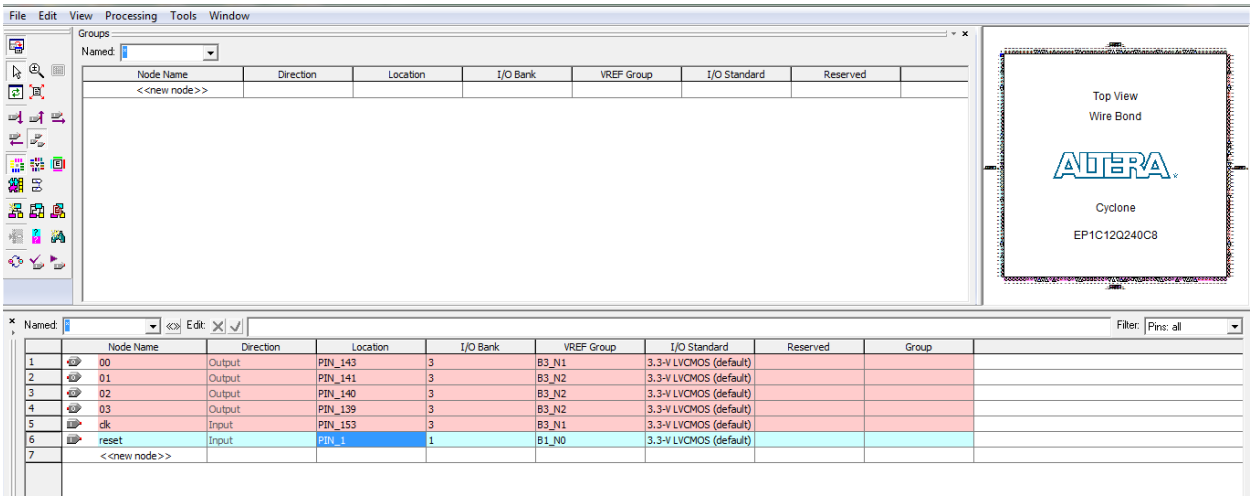

Рисунок 2.60 – Присвоение контактов

2. Далее в меню выбрать Tools, затем Programmer (рисунок 2.61).

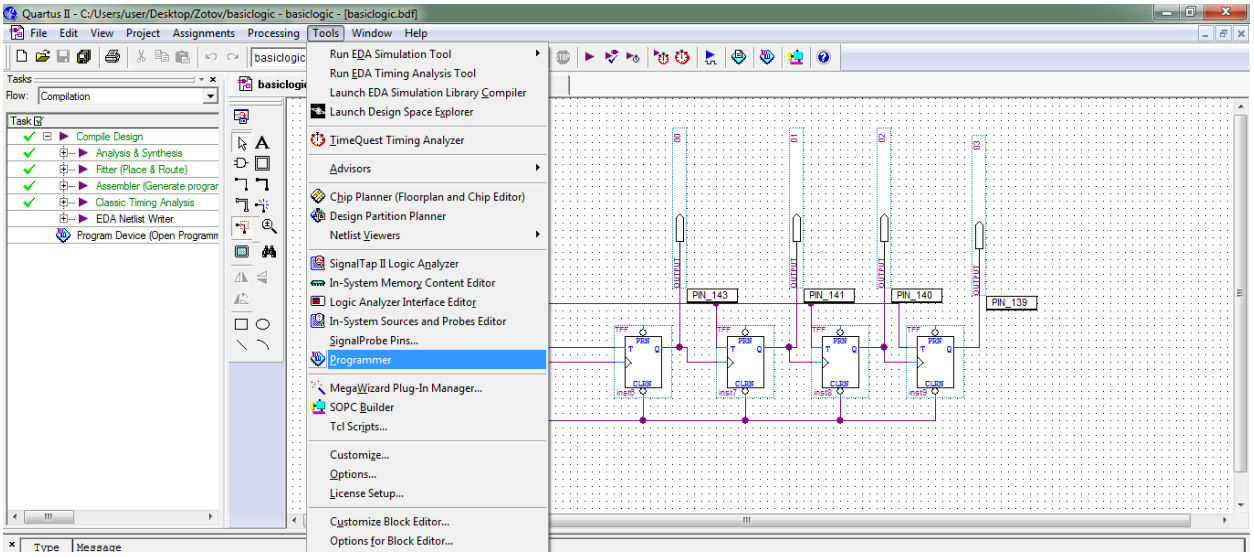

Рисунок 2.61 – Прошивка отладочного макета

3. В появившемся окне выбрать нужный макет и после того, как строчка засветилась синим, нажать Start (рисунок 2.62).

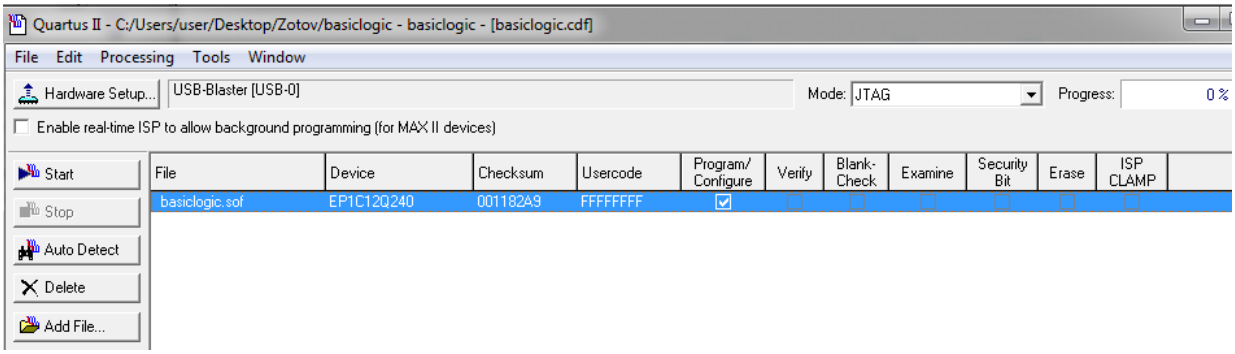

Рисунок 2.62 – Прошивка отладочного макета

Схема вшита в макет и готова к проверке (рисунок 2.63).

Включить учебный макет, на генераторе подать импульс (поставить в положение 1 или 2). В модуле кнопок первый переключатель поставить в положение ВКЛ. На модуле диодов показана работа вычитающего счетчика, вшитого в макет.

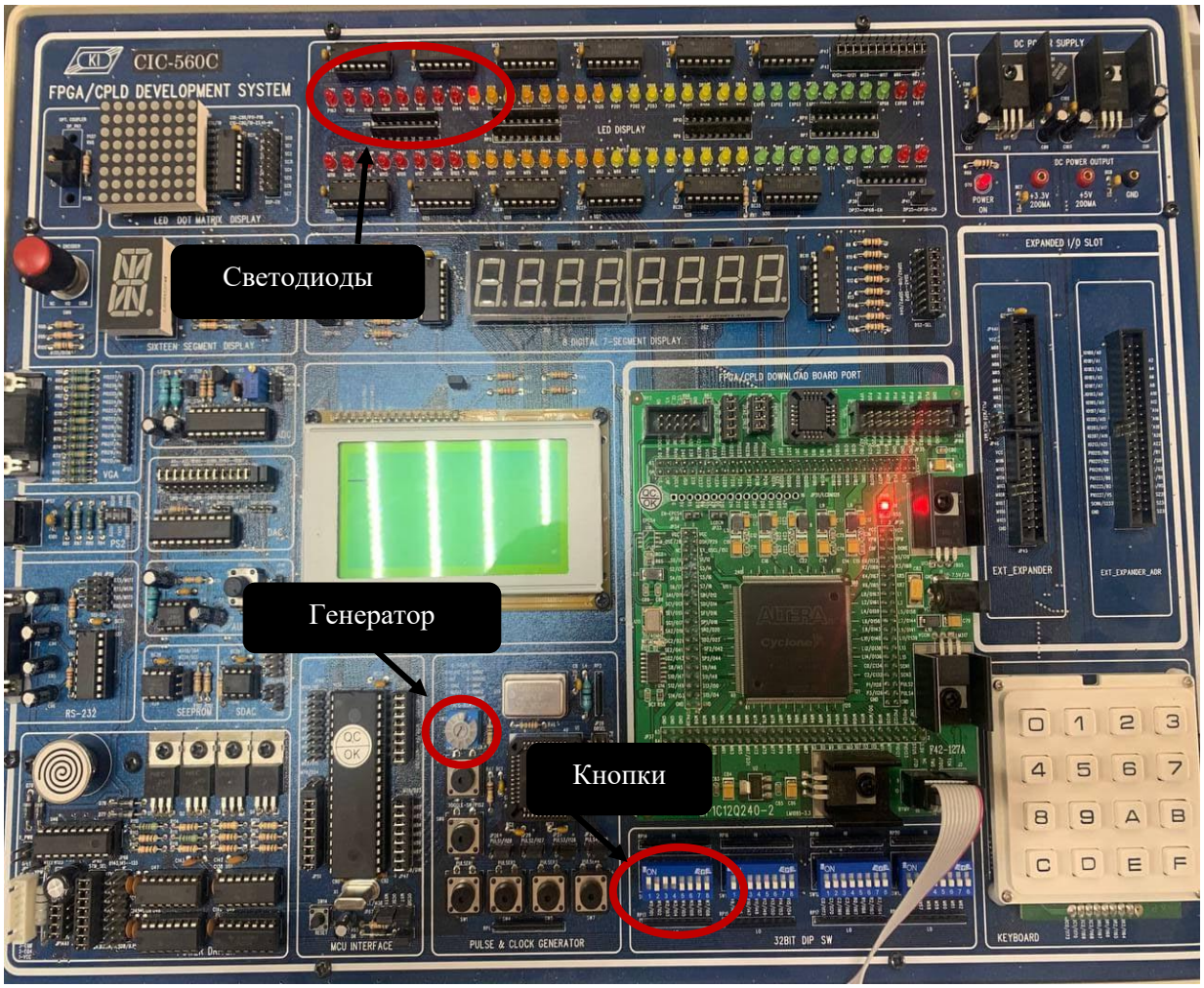

Рисунок 2.63 – Учебный макет CIC-560

Индивидуальные задания

1. Выполнить синтез заданного преподавателем устройства (таблица 2.1).

Таблица 2.1 – Варианты заданий

Вариант	Устройство
1	Суммирующий счетчик с заданным модулем счета
2	Делитель частоты
3	Генератор ПСП

2. На языке VHDL написать код устройства.
3. Выполнить моделирование работы временных диаграмм разработанного устройства.
4. Запрограммировать ПЛИС. Результаты выполнения лабораторной работы оформить в виде отчета.

III. ЛАБОРАТОРНЫЕ РАБОТЫ (ESP32)

Лабораторная работа №10

УСТАНОВКА ПРОШИВКИ MICROPYTHON ДЛЯ ESP32

Цель работы: ознакомиться и прошить плату ESP32.

Оборудование и программное обеспечение: язык MicroPython, микроконтроллер ESP32, интегрированная среда разработки Thony.

Теоретический материал

ESP32 – серия недорогих микроконтроллеров и микропроцессоров с низким энергопотреблением. Представляют собой систему на кристалле с интегрированным Wi-Fi и Bluetooth-контроллерами. В серии ESP32 используется микроконтроллерное ядро Tensilica Xtensa LX6 в вариантах с двумя и одним ядром. В систему интегрирован радиочастотный тракт: симметрирующий трансформатор, встроенные антенные коммутаторы, радиочастотные компоненты, малошумящий усилитель, усилитель мощности, фильтры и модули управления питанием (рисунок 3.1). ESP32 создан и разработан китайской компанией Espressif Systems, расположенной в Шанхае, а производится компанией TSMC по техпроцессу 40 нм. Серия является преемником микроконтроллеров ESP8266.

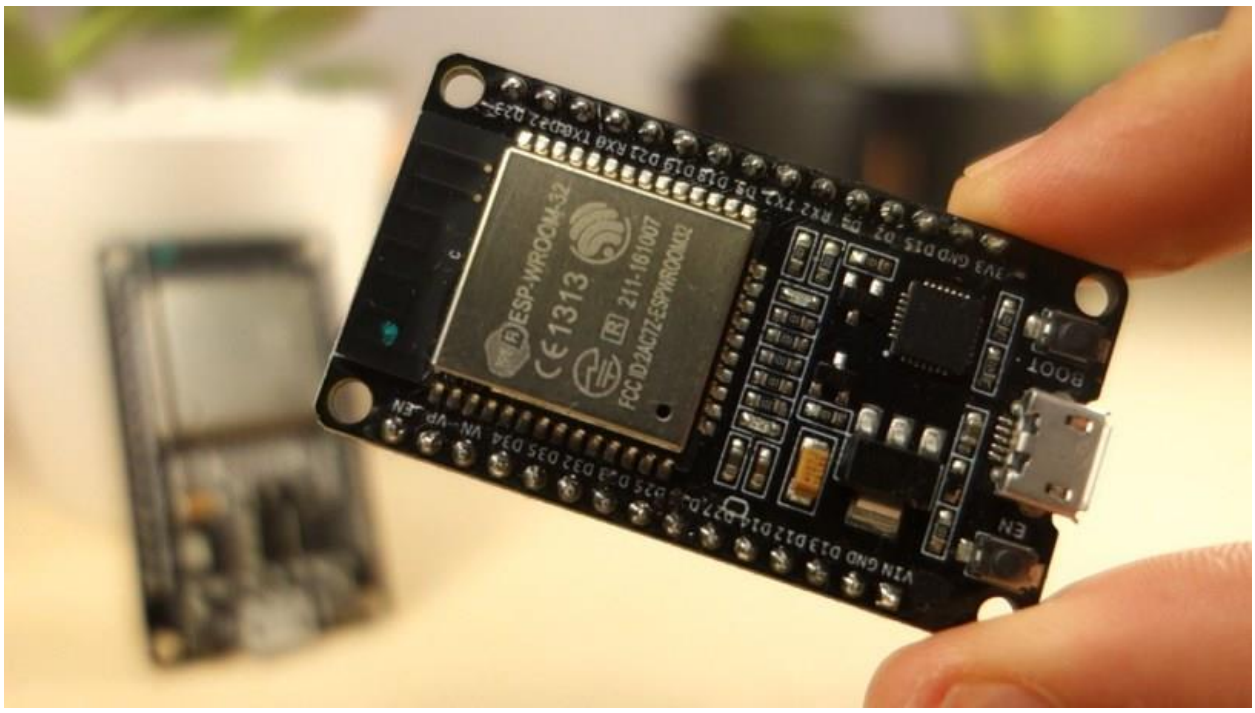

Рисунок 3.1 – Внешний вид тактовой ESP32

К периферийным устройствам ESP32 относятся:

– 18 каналов аналого-цифрового преобразователя (АЦП);
– 3 интерфейса SPI23;
– интерфейс UART2;
– интерфейс I2C16;
– выходные каналы ШИМ;
– 2 цифроаналоговых преобразователя (ЦАП);
– 2 интерфейса I2S10;
– емкостные считывающие GPIO.

Функции АЦП и ЦАП назначены строго определенным контактам. Тем не менее можно самостоятельно решить, какие из них будут отведены под интерфейсы UART, I2C, SPI, PWM и т. д. – они определяются в коде прошивки. Это возможно благодаря функции мультиплексирования чипа ESP32.

Существуют контакты, назначенные по умолчанию, как показано на рисунке 3.2 (это пример платы ESP32 с 36 контактами, расположение контактов может меняться в зависимости от производителя).

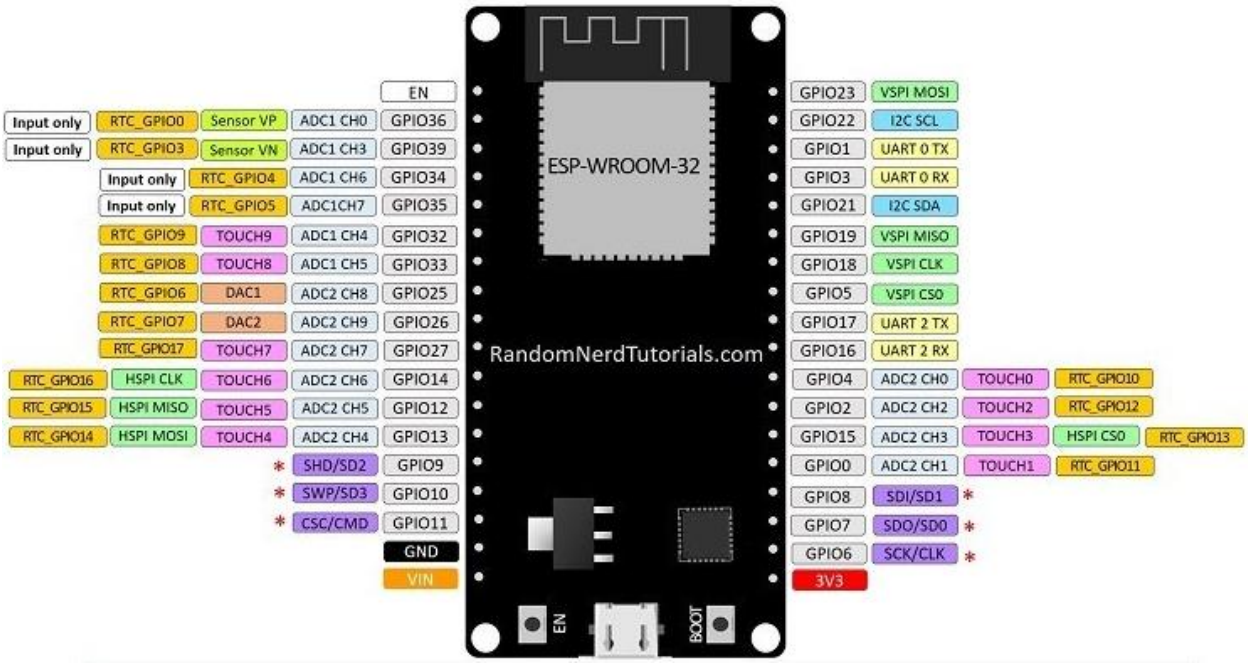

Рисунок 3.2 – Контакты ESP32

Кроме того, есть контакты с определенными функциями, которые делают их подходящими или неподходящими для конкретного проекта. В таблице 3.1 показано, какие выводы лучше всего использовать в качестве входов, а с какими следует соблюдать осторожность.

Контакты с ОК подходят для использования, без ОК также подходят для использования, однако следует обращать на них большее внимание, поскольку их поведение может отличаться от ожидаемого, в основном это происходит при загрузке.

Таблица 3.1 – Назначение выводов

GPIO	**Вход**	**Выход**	**Примечания**	**GPIO**	**Вход**	**Выход**	**Примечания**
0	pulled up	ОК	Выводит сигнал ШИМ при загрузке	**11**	x	x	Подключение к встроенному последовательному периферийному интерфейсу
1	TX pin	ОК	Отладочный вывод при загрузке	**12**	ОК	ОК	Загрузка прервется при значении «1»
2	ОК	ОК	Соединение с встроенным диодом	**13**	ОК	ОК	–
3	ОК	RX pin	«1» на загрузке	**14**	ОК	ОК	Выводит сигнал ШИМ при загрузке
4	ОК	ОК	–	**15**	ОК	ОК	Выводит сигнал ШИМ при загрузке
5	ОК	ОК	Выводит сигнал ШИМ при загрузке	**16**	ОК	ОК	–
6	x	x	Подключение к встроенному последовательному периферийному интерфейсу	**17**	ОК	ОК	–
7	x	x	Подключение к встроенному последовательному периферийному интерфейсу	**18**	ОК	ОК	–
8	x	x	Подключение к встроенному последовательному периферийному интерфейсу	**21**	ОК	ОК	–
9	x	x	Подключение к встроенному последовательному периферийному интерфейсу	**24…33**	ОК	ОК	–
10	x	x	Подключение к встроенному последовательному периферийному интерфейсу	**34…39**	ОК	–	Только ввод

GPIO с 34 по 39 являются только *входными* GPI. Эти контакты не имеют внутренних подтягивающих или понижающих резисторов.

SPI-flash встроена в ESP-WROOM-32. От GPIO 6 до GPIO 11 представлены в некоторых платах ESP32. Однако эти контакты подключены к встроенной Flash-памяти SPI на микросхеме ESP-WROOM-32. Использовать их рекомендуется для других целей. Назначение пинов:

- GPIO 6 (SCK/CLK);
- GPIO 7 (SDO/SD0);
- GPIO 8 (SDI/SD1);
- GPIO 9 (SHD/SD2);
- GPIO 10 (SWP/SD3);
- GPIO 11 (CSC/CMD).

ESP32 имеет 10 внутренних емкостных сенсорных датчиков. Они могут отслеживать все, что содержит электрический заряд, например, они могут обнаруживать изменения, возникающие при касании пальцами GPIO. Эти контакты могут быть легко встроены в датчики касания и заменять механические кнопки. Емкостные сенсорные контакты также могут быть использованы для выхода ESP32 из спящего режима.

Внутренние сенсорные датчики подключены к следующим GPIO:

- T0 (GPIO 4);
- T1 (GPIO 0);
- T2 (GPIO 2);
- T3 (GPIO 15);
- T4 (GPIO 13);
- T5 (GPIO 12);
- T6 (GPIO 14);
- T7 (GPIO 27);
- T8 (GPIO 33);
- T9 (GPIO 32).

ESP32 имеет входные каналы АЦП 18×12 бит. Это GPIO, которые можно использовать в качестве АЦП:

- ADC1_CH0 (GPIO 36);
- ADC1_CH1 (GPIO 37);
- ADC1_CH2 (GPIO 38);
- ADC1_CH3 (GPIO 39);
- ADC1_CH4 (GPIO 32);
- ADC1_CH5 (GPIO 33);

– ADC1_CH6 (GPIO 34);
– ADC1_CH7 (GPIO 35);
– ADC2_12 GPO (0);
– ADC2_CH2 (GPIO 2);
– ADC2_CH3 (GPIO 15);
– ADC2_CH4 (GPIO 13);
– ADC2_CH5 (GPIO 12);
– ADC2_CH6 (GPIO 14);
– ADC2_CH7 (GPIO 27);
– ADC2_CH8 (GPIO 25);
– ADC2_CH9 (GPIO 26).

Примечание – Контакты ADC2 нельзя применять при использовании Wi-Fi. Поэтому, если используется Wi-Fi и возникают проблемы с получением значения от GPIO ADC2, следует рассмотреть возможность использования GPIO ADC1, что должно решить данную проблему.

Входные каналы АЦП имеют разрешение 12 бит. Это означает, что можно получить аналоговые показания в диапазоне от 0 до 4095, где 0 соответствует 0 В, а 4095 – 3,3 В. Также есть возможность установить разрешение каналов в коде и диапазон АЦП.

При использовании выводов АЦП также следует помнить, что выводы АЦП ESP32 работают нелинейно.

Ход работы

Установка прошивки MicroPython для ESP32:

1. Перед первым подключением платы к ПК необходимо скачать и установить драйверы для работы с ESP32 (CP210xVCPInstaller_x64).
2. Найти подходящую прошивку под вашу плату на Github или на официальном сайт MicroPython (https://micropython.org/download/esp32).
3. Установить Python3:

   Для Windows:

   – проверить версию и наличие Python, для чего открыть консоль (Win+R, набрать cmd и нажать Enter) и ввести python-v или python-version;
   – если версия Python ниже 3 или он совсем не установлен, то скачать последнюю версию Python 3.x для Windows;
   – установить Python, запустив exe-файл.

Для Linux:

- проверить версию Python, которая уже стоит в вашей системе, введя в консоли python3-v или python3-version;
- при необходимости установить последнюю версию Python, введя в консоли sudo apt-get install python3.

4. После установки, ввести в командной строке pip install esptool и pip install adafruit-ampy.
5. Подключить плату к компьютеру.
6. Определить порт, в который подключена плата (COM1, 2, 3…).
7. Стереть старую прошивку: esptool --chip esp32 --port COM [НОМЕР_ПОРТА] erase_flash.
8. Создать новую прошивку: esptool --chip esp32 --port COM [НОМЕР_ПОРТА] --baud 460800 write_flash -z 0x1000 esp32.bin;
9. Установить Putty:

   Для Windows:

   - скачать последнюю версию, подходящую под вашу версию Windows;
   - запустить установщик программы и следовать инструкциям на экране.

   Для Linux: открыть консоль и ввести: sudo apt-get install putty.
10. Зайти в Putty.
11. Указать тип подключения (Serial).
12. Указать номер порта, к которому подключена плата (COM [НОМЕР_ПОРТА]).
13. Указать скорость работы (115200).
14. Указать имя в графе Saved Sessions и нажать Save.
15. Нажать Open.

Примечание – Как IDE рекомендуется использовать Thonny IDE.

Лабораторная работа №11

РАБОТА С ВХОДАМИ/ВЫХОДАМИ ESP32

Цель работы: создать электронную схему с помощью ESP32, используя светодиоды и резисторы.

Оборудование и программное обеспечение: язык MicroPython, микроконтроллер ESP32, интегрированная среда разработки Thony.

Теоретический материал

Порты ввода/вывода (GPIO)

Для связи с внешними устройствами у микроконтроллера существуют порты ввода/вывода. Они могут работать в режиме универсальных портов ввода/вывода или выполнять альтернативные функции.

В системе ESP32 есть 34 физических вывода портов. Каждый из них может быть использован как универсальный вывод или подключен к внутреннему периферийному устройству.

Упрощенная схема коммутации сигналов портов ввода/вывода представлена на рисунке 3.3.

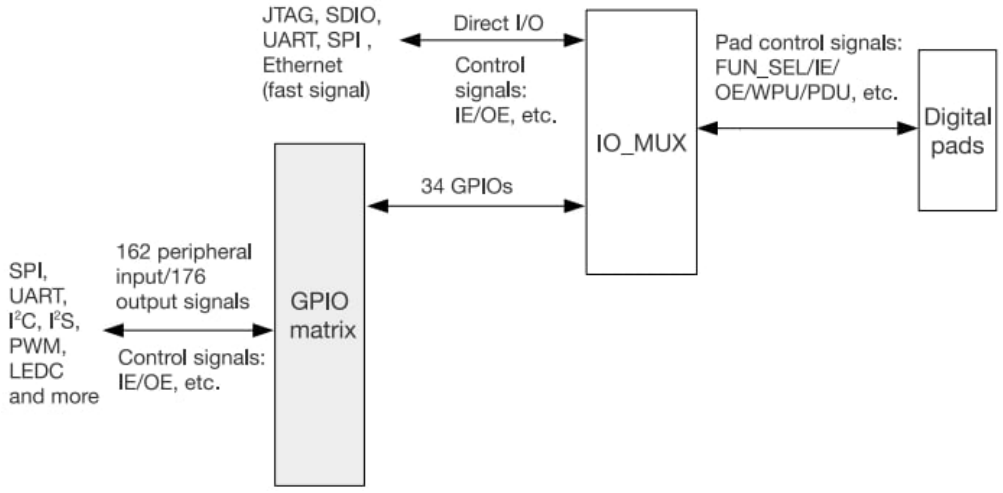

Рисунок 3.3 – Упрощенная схема коммутации сигналов портов ввода/вывода

Мультиплексор IO_MUX выбирает для каждого разряда порта режим универсального ввода/вывода или альтернативную функцию.

Матрица мультиплексоров GPIO matrix коммутирует сигналы периферийных устройств на выводы портов.

Отметим, что для каждого вывода можно:

– выбрать режим универсального ввода/вывода;

– задать режим альтернативной функции;
– задать конкретную альтернативную функцию, т. е. выбрать сигнал периферийного устройства.

Каждый дискретный вывод может быть установлен в режим:

– входа;
– без подтягивающего резистора;
– с подтягивающим резистором на питание;
– с подтягивающим резистором на землю;
– активного выхода;
– выхода с открытым стоком.

Режимы и возможности, в принципе, такие же, как у портов STM32. И там, и здесь разработчики реализовали все возможные варианты. Но у ESP32 практически неограниченный выбор коммутации сигналов альтернативных функций на выводы.

Электрические параметры портов ввода/вывода

К выводам микроконтроллера подключаются другие электронные компоненты. Надо понимать, что можно подключать непосредственно к выводам, а что требует дополнительных согласующих элементов. Поэтому важно знать электрические характеристики входных и выходных сигналов микроконтроллера (таблица 3.2).

Таблица 3.2 – Входные/выходные сигналы микроконтроллера

Обозначение	Параметр	Мин. значение	Типовое значение	Макс. значение	Единица измерения
V_{IH}	Входное напряжение высокого уровня	$0{,}75 \cdot V_{DD}$	–	$V_{DD} + 0{,}3$	В
V_{IL}	Входное напряжение низкого уровня	–0,3	–	$0{,}25 \cdot V_{DD}$	В
I_{IH}, I_{IL}	Входной ток	–	–	50	мА
V_{OH}	Выходное напряжение высокого уровня	$0{,}8 \cdot V_{DD}$	–	–	В
V_{OL}	Выходное напряжение низкого уровня	–	–	$0{,}1 \cdot V_{DD}$	В
I_{IO}	Вытекающий ток вывода	–	40	–	мА
I_{OL}	Втекающий ток вывода	–	28	–	мА
R_{PU}, R_{PD}	Сопротивление подтягивающего резистора	–	45	–	кОм

Из этого следует:

1. Дискретные входы воспринимают за низкий уровень сигналы с напряжением не выше 0,825 В, а за высокий уровень – не ниже 2,475 В.
2. Напряжение на дискретном выходе составляет не менее 2,64 В при высоком состоянии сигнала и не более 0,33 В при низком.
3. Максимально допустимый выходной вытекающий ток выводов – 40 мА, а втекающий – 28 мА.

В таблице 3.2 преведены значения параметров при напряжении питания микроконтроллера, равном 3,3 В.

Пример программы

Определить подходящие контакты можно с помощью рисунка 3.2.

Для реализации некоторой схемы нам понадобятся как минимум ESP32, два резистора и два светодиода. Как вариант соединения предложим следующую схему (рисунок 3.4).

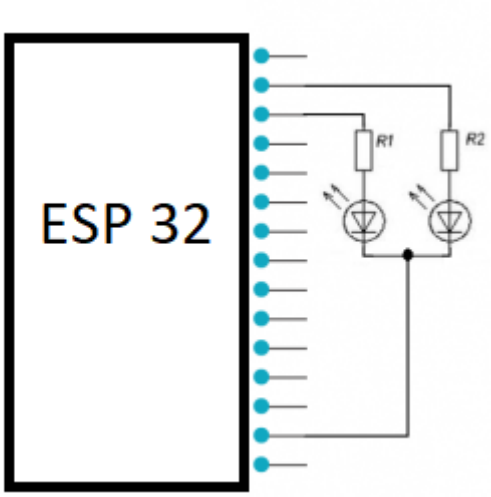

Рисунок 3.4 – Вариант соединения

Ход работы

Листинг кода:

```
import time
time.sleep(1)                # ждать секунду
time.sleep_ms(200)    # ждать 200 миллисекунд
time.sleep_us(1000)   # ждать 1000 микросекунд
from machine import Pin
p0 = Pin(0, Pin.OUT) # по сути помещаем GPIO0 в переменную и
назначаем его на вывод
```

```
p2 = Pin(2, Pin.IN)     # то же самое, но на ввод
p0.on()             # подаем питание на контакт, который положили в переменную
p0.off()                # убираем питание с контакта
p0.value(1)             # 1 – включен, 0 – выключен
print(p2.value())           # получить значение, которое сейчас на контакте
p4   =   Pin(4,   Pin.IN,   Pin.PULL_UP)   #   включить   внутренний подтягивающий резистор
p5 = Pin(5, Pin.OUT, value=1) # при создании переменной сразу ее включаем
```

Индивидуальное задание

Заставить диоды моргать с временным промежутком, заданным преподавателем, используя библиотеки machine, time и новые знания о работе контактов в ESP32.

Лабораторная работа №12

ТАЙМЕРЫ

Цель работы: узнать об альтернативном способе реализации времени на контроллере ESP32.

Оборудование и программное обеспечение: язык MicroPython, микроконтроллер ESP32, интегрированная среда разработки Thony.

Теоретический материал

В системе ESP32 существует четыре таймера общего назначения. В MicroPython они имеют id от 0 до 3 и вызываются через библиотеку machine следующим образом:

```
from machine import Timer

tim0 = Timer(0)
tim0.init(period=5000, mode=Timer.ONE_SHOT, callback=lambda t:print(0))

tim1 = Timer(1)
tim1.init(period=2000, mode=Timer.PERIODIC, callback=lambda t:print(1))
```

Структура таймеров проста, типична для подобных устройств и похожа на схему таймеров STM32 в режиме счетчиков (рисунок 3.5).

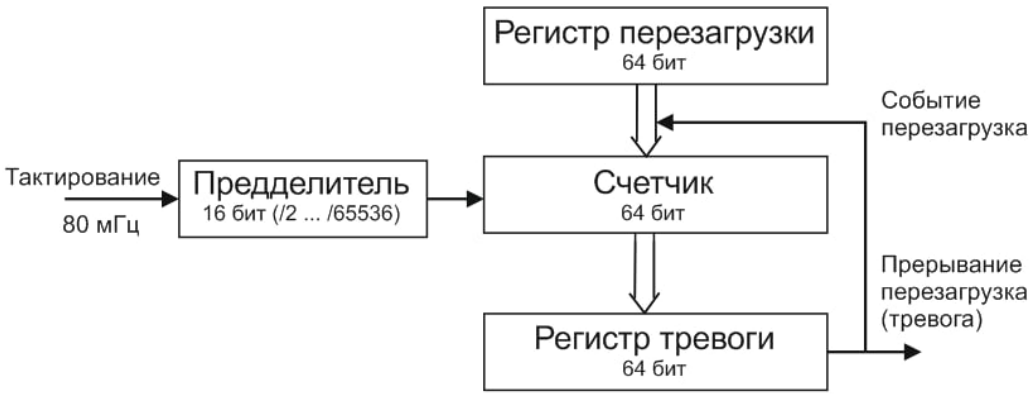

Рисунок 3.5 – Структура таймеров

Можно выделить главные моменты функционирования таймеров:

1. В стандартной конфигурации таймеры тактируются частотой 80 мГц. На входе каждого счетчика установлен 16-разрядный предделитель,

который снижает частоту тактирования от 2 до 65 536 раз. Коэффициент деления устанавливается программно.

2. При значении предделителя, равном 1 или 2, входная частота делится на 2. При значении 0 коэффициент деления равен 65 536. Остальные значения предделителя соответствуют коэффициенту деления входной частоты.
3. Основной счетчик – 64-разрядный. Можно не заботиться о его переполнении. Например, при частоте тактирования 1 мГц (период 1 мкс) счетчик переполнится почти через 600 000 лет. Он может считать в прямом или обратном направлении, быть остановленным. Его значение может быть считано и установлено программно.
4. Счетчик может быть загружен значением из регистра перезагрузки по аппаратному событию перезагрузки. В терминологии ESP32 это называется событием тревоги. Оно происходит при совпадении значения счетчика со значением, заданным программно в регистре тревоги.
5. Событие перезагрузки вырабатывается не только при строгом равенстве значений счетчика и регистра тревоги, а также при превышении значения счетчика заданного порога для прямого счета и при уменьшении значения счетчика ниже порога для обратного счета. Это позволяет не пропустить событие перезагрузки в случае опоздания установки регистра тревоги по отношению к состоянию счетчика.

Каждый модуль таймеров содержит сторожевой таймер и связанные с ним регистры.

Каждый модуль таймеров может генерировать три типа прерываний:

1) прерывание по истечении времени ожидания сторожевого таймера;
2) прерывание перезагрузки (тревоги) таймера 0 (TIMER_0);
3) прерывание перезагрузки (тревоги) таймера 1 (TIMER_1).

Какие регистры используются для работы с таймерами и их форматы знать необязательно.

Индивидуальное задание

Собрать электронную схему по рисунку 3.8 из лабораторной работы №13, используя таймеры.

Лабораторная работа №13

ШИРОТНО-ИМПУЛЬСНАЯ МОДУЛЯЦИЯ

Цель работы: изучить широтно-импульсную модуляцию и применить ее на практике.

Оборудование и программное обеспечение: язык MicroPython, микроконтроллер ESP32, интегрированная среда разработки Thony.

Теоретический материал

Широтно-импульсная модуляция (ШИМ) – это метод, который изменяет ширину импульса, сохраняя постоянную частоту сигнала. Техника ШИМ в основном используется для управления яркостью светодиодов, скоростью вращения двигателя постоянного тока, управления серводвигателем или в других случаях, когда необходимо генерировать аналоговый сигнал с использованием цифрового источника.

Прежде чем объяснить генерацию ШИМ на ESP32, следует обсудить некоторые термины, связанные с ШИМ. Существует несколько основных сокращений и определений. T_{on} (время включения) – это продолжительность времени, когда сигнал высокий. T_{off} (время выключения) – это продолжительность времени, когда сигнал низкий. Период – это сумма времени включения и выключения сигнала ШИМ. Рабочий цикл (скважность) – это процент времени, когда сигнал был высоким в течение периода сигнала ШИМ (определяется как $T_{on}/T_{полн} \cdot 100$).

Например, если импульс с общим периодом 10 мс остается высоким в течение 5 мс, тогда скважность будет равна $5/10 \cdot 100 = 50$ % (рисунок 3.6).

Рисунок 3.6 – Пример

Есть еще понятие частоты ШИМ. Частота сигнала ШИМ определяет, как быстро ШИМ завершает один период. Один период – полное включение и выключение сигнала ШИМ, как показано на рисунке 3.6.

Ход работы

Реализуем ШИМ, при которой светодиод будет плавно увеличивать и уменьшать свою яркость.

Порядок работы:

1. Подключить ШИМ к ножке, к которой подключен светодиод.
2. Установить начальную частоту.
3. Реализовать цикл, меняющий эту частоту.

Необходимо помнить, что максимальное заполнение ШИМ соответствует 1023, минимальное – 0.

Пример программы

Чтобы реализовать ШИМ, будем использовать следующие библиотеки:

```
from machine import Pin, PWM
pwm0 = PWM(Pin(0))    # создать ШИМ объекта из контакта
pwm0.freq()           # получить текущую частоту (по дефолту 5000)
pwm0.freq(1000)       # установить частоту от 1 Гц до 40 МГц
pwm0.duty()           # получить текущий рабочий цикл от 0 до 1023 (default
512, 50 %)
pwm0.duty(256)        # установить рабочий цикл от 0 до 1023 (25 %)
pwm0.deinit()         # выключить ШИМ для данного контакта
pwm2 = PWM(Pin(2), freq=20000, duty=512) # создать и конфигурировать ШИМ
pwm0.init()           # включить ШИМ для данного пина
```

Для подключения ШИМ необходимо указать, к какой ножке с помощью функции PWM подключаем ШИМ. Также необходимо помнить, что не все контакты его поддерживают.

Индивидуальное задание

Собрать электронную схему и реализовать цикл, изменяющий частоту с помощью ШИМ, в пределах частот, предложенных преподавателем для светодиода (рисунок 3.7).

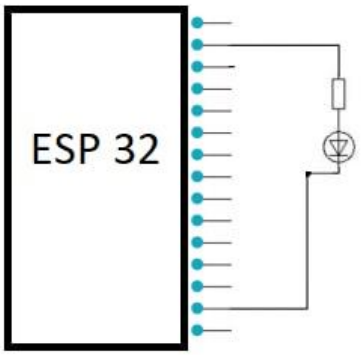

Рисунок 3.7 – Схема индивидуального задания

Лабораторная работа №14

БЕСПРОВОДНОЙ WI-FI-МОДУЛЬ ESP32

Цель работы: ознакомиться с Wi-Fi-модулем ESP32 и научиться пользоваться им в практических задачах.

Оборудование и программное обеспечение: язык MicroPython, микроконтроллер ESP32, интегрированная среда разработки Thony.

Теоретический материал

Wi-Fi-модули ESP8266 за время своего существования стали поистине народными и получили широкое распространение в любительской разработке устройств интернета вещей. Но жизнь не стоит на месте, и компания-разработчик Espressif выпустила новый микроконтроллер – ESP32, который получил значительный прирост в производительности по сравнению с ESP8266 [16]. Вычислительная мощность возросла в четыре раза. У ESP32 есть два ядра, каждое из которых работает на частоте 160 МГц (ESP8266 имеет одно ядро, работающее на частоте 80 МГц). Контроллер имеет 520 Кбайт оперативной памяти и 448 Кбайт Flash-памяти, поддерживает не только Wi-Fi (802.11n с максимальной скоростью 150 Мбит в секунду), но и Bluetooth 4.2 BR/EDR и Low Energy.

Основным недостатком плат ESP8266 было очень малое количество контактов, в ESP32 этот недостаток устранен, выводов гораздо больше, и они многофункциональные. Блок ввода/вывода имеет специальный мультиплексор, который позволяет назначать различные функции на один вывод микроконтроллера. Значительно увеличено количество аналоговых входов (18 АЦП (12-бит) и 2 ЦАП (8-бит)), поддержка PWM на всех контактах, 10 портов в режиме сенсорных кнопок. ESP32 имеет три UART, два I2C, четыре SPI, два I2S. Также имеется инфракрасный контроллер (прием – передача), шина CAN 2.0. Еще есть датчик температуры и датчик Холла. Для шифрования при передаче данных по Wi-Fi в ESP32 имеются криптографические модули AES и SHA (рисунок 3.8).

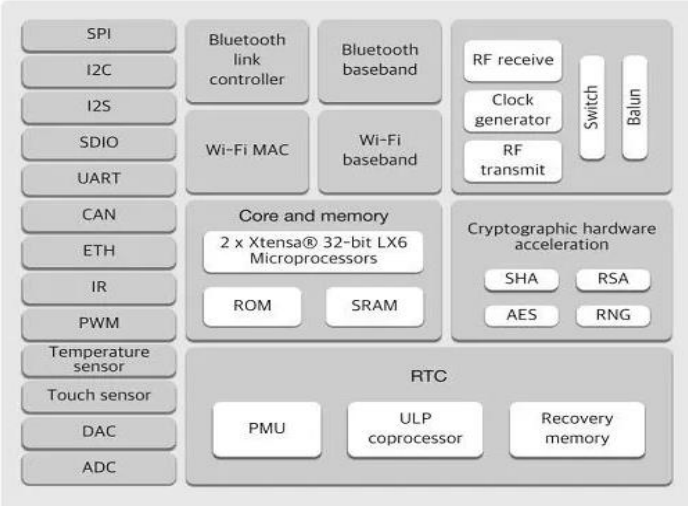

Рисунок 3.8 – Блок-схема периферии ESP32

Для удобной работы с микроконтроллером ESP32 был выпущен модуль WROOM32 (рисунок 3.9), и появилось множество отладочных плат на нем, например FireBeetle от DFRobot (рисунок 3.10).

Рисунок 3.9 – Модуль ESP-WROOM32

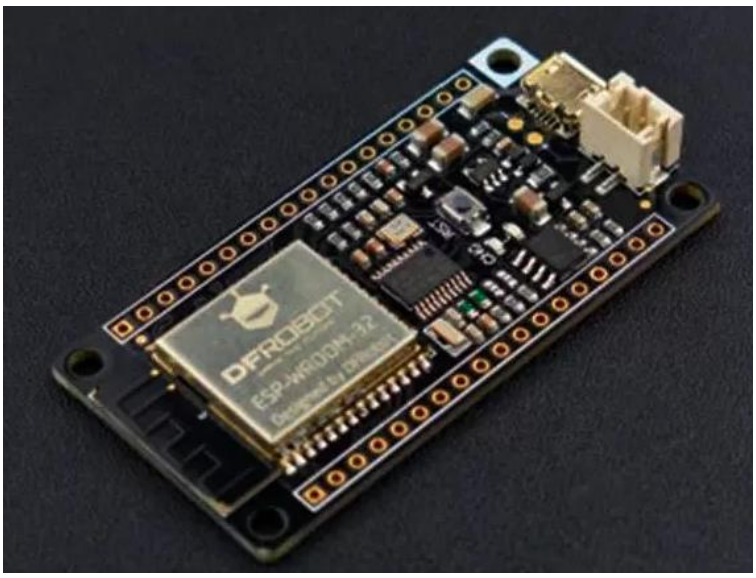

Рисунок 3.10 – Отладочная плата FireBeetle от DFRobot на ESP-WROOM32

Немаловажный вопрос в свете использования модулей для устройств интернета вещей – энергопотребление. Питание модуля: максимальный ток потребления в режиме передачи Wi-Fi или Bluetooth составляет 160–260 мА, без включенных Wi-Fi или Bluetooth – 20 мА, в спящем режиме – 10 мкА.

Ход работы

Для выполнения лабораторной работы необходимо использовать библиотеку network (import network).

Небольшая справка по использованию:

```
wlan = network.WLAN(network.STA_IF) # создание интерфейса станции
wlan.active(True)       # активация интерфейса
wlan.scan()             # сканирование в поиске точки доступа
wlan.isconnected()      # проверка соединения с точкой доступа
wlan.connect('essid', 'password') # соединение с точкой доступа
wlan.config('mac')      # получение MAC-адрес интерфейса
wlan.ifconfig()         # получение IP/netmask/gw/DNS-адреса интерфейса
ap = network.WLAN(network.AP_IF) # создание точки доступа
ap.config(essid='ESP-AP') # установка имени точки доступа
ap.config(max_clients=10) # максимальное количество клиентов
ap.active(True)         # активация интерфейса
```

Примечание – Станция = клиент, точка доступа = роутер (рисунок 3.11).

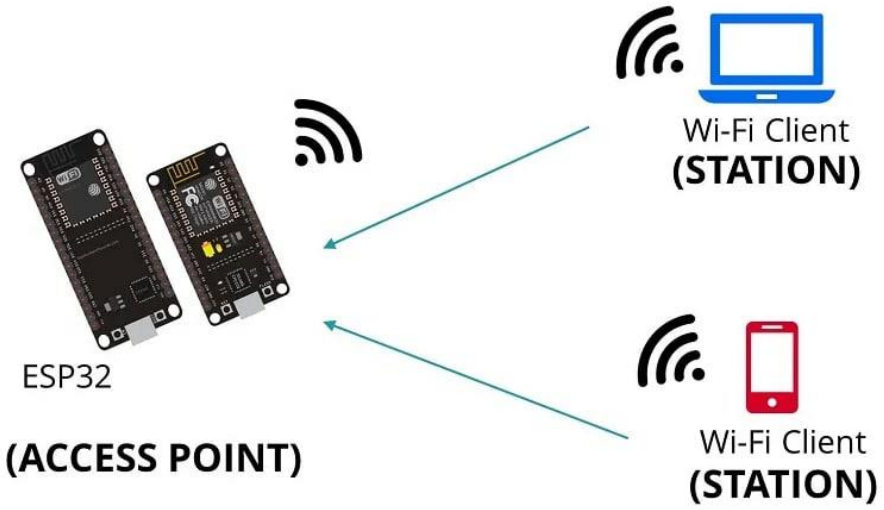

Рисунок 3.11 – Взаимодействие ESP32 с другими устройствами по Wi-Fi

Пример кода (соединение с некоторой сетью):

```
def do_connect():
    import network
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        print('connecting to network...')
        wlan.connect('essid', 'password')
        while not wlan.isconnected():
            pass
    print('network config:', wlan.ifconfig())
```

Полноценная точка доступа выглядит следующим образом:

```
try:
  import usocket as socket
except:
  import socket
import network
import esp
esp.osdebug(None)
import gc
gc.collect()
ssid = 'BSUIR-AP'
password = 'atiugadai'
ap = network.WLAN(network.AP_IF)
ap.active(True)
ap.config(essid=ssid, password=password)
```

```
print('Connection successful')
print(ap.ifconfig())
def web_page():
  html = """<html><head><meta name="viewport" content="width=device-
width, initial-scale=1"></head>
  <body><h1>Group 841301</h1><h2>Wifi-AccessPoint</h2><p><button
onclick="location.href=’https://bsuir.by’;"><table
border="0"><tr><td>ON</td></tr></table></button><button><table
border="0"><tr><td>OFF</td></tr></table></button></p> </body></html>"""
  return html
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 80))
s.listen(5)
while True:
  conn, addr = s.accept()
  print('Got a connection from %s' % str(addr))
  request = conn.recv(1024)
  print('Content = %s' % str(request))
  response = web_page()
  conn.send(response)
  conn.close()
```

Индивидуальное задание

Создать программу, которая посредством подключения к точке доступа будет управлять светодиодом.

ПРИЛОЖЕНИЕ А

ОБЩЕЕ ОПИСАНИЕ АРХИТЕКТУРЫ ARM И 32-РАЗРЯДНЫХ МИКРОКОНТРОЛЛЕРОВ STM

32-разрядная архитектура ARM

Процессоры ARM являются ключевым компонентом для большого количества успешных 32-битных встраиваемых систем. Процессоры ARM широко используются в мобильных телефонах, планшетах и других портативных устройствах. ARM основаны на RISC-архитектуре, что позволяет уменьшить потребление энергии процессором и, таким образом, делает их идеальным выбором для встраиваемых систем.

Хотя ARM основаны на RISC-архитектуре, они не полностью повторяют принципы построения таких систем. Для того чтобы сделать ARM более приспособленным к использованию во встраиваемых системах, пришлось пойти на следующие отклонения от принципов RISC:

1. Переменное количество циклов выполнения для простых инструкций. Простые инструкции ARM могут потребовать на выполнение более одного цикла. Например, выполнение инструкции Load и Save зависит от количества регистров, которые им переданы.
2. Возможность соединять команды сдвига и вращения с командами обработки информации.
3. Условное выполнение – инструкция выполняется только в том случае, если соблюдено конкретное условие. Это увеличивает производительность и позволяет избавиться от операторов ветвления.
4. Улучшенные инструкции – процессоры ARM поддерживают улучшенные DSP-инструкции для операций с цифровыми сигналами.

Программист может рассматривать ядро ARM как набор функциональных блоков (ALU, MMU и др.), соединенных шиной данных. Данные поступают в процессор через шину данных. Декодер инструкций обрабатывает инструкции перед их выполнением. ARM могут работать только с данными, которые записаны в регистрах, поэтому перед выполнением инструкций в регистры записываются дынные для их выполнения. ALU считывает данные из регистров, выполняет необходимые операции и записывает результат обратно в регистр, откуда его можно записать во внешнюю память.

Процессоры ARM содержат до 18 регистров: 16 регистров данных и 2 регистра процессов. Все регистры содержат 32 бита и именуются от R0 до R15.

Регистры R13, R14, R15 используются для выполнения определенных специфических задач:

– R13 используется в качестве указателя стека;
– R14 используется как связывающий регистр;
– R15 играет роль счетчика.

В зависимости от контекста эти регистры могут использоваться как регистры общего назначения. Также имеется два программных регистра, которые называются CPSR (Current Program Status Register) и SPSR (Saved Program Status Register), которые используются для сохранения состояния процессора и программы.

Одними из последних процессоров для встраиваемых систем являются процессоры, основанные на архитектуре ARM Cortex-M4 [1]. Эти процессоры предназначены для использования в цифровой обработке сигналов (Digital Signal Processing, DSP). Микроконтроллер STM32F407VG, установленный на рассматриваемой плате, в качестве основы использует именно решение ARM Cortex-M4. В общем виде микроконтроллеры, основанные на базе ARM Cortex-M4, имеют следующие внутренние модули (рисунок А.1).

Рисунок А.1 – Встроенные модули ARM Cortex-M4

На рисунке А.1 видно, что рассматриваемый образец обладает богатой периферией и может использоваться для решения многих практических задач различной направленности.

Семейство микроконтроллеров STM32

Семейство микроконтроллеров STM32 построено с использованием 32-разрядного ядра Cortex различных версий (в микроконтроллере, установленном на плате, используется ядро Cortex-M4). Некоторые основные характеристики ядра микроконтроллеров STM32 представлены в таблице А.1.

Таблица А.1 – Основные характеристики ядра микроконтроллеров STM32

Характеристика	Значение
Ширина слов для данных, разряд	32
Архитектура	Гарвард
Конвейер	3-ступенчатый
Набор инструкций	RISC
Организация памяти программ, разряд	32
Буфер предвыборки, разряд	2×64
Средний размер инструкций, байт	2
Тип прерываний	Векторизированные
Задержка реагирования на прерывания	12 циклов
Режимы управления энергопотреблением	Сон, сон по входу, глубокий сон
Отладочный интерфейс	ST-LINK, JTAG

Микроконтроллеры данного типа построены на гарвардской архитектуре и имеют 3-ступенчатый конвейер, который минимизирует время выполнения команд. Они разработаны для построения систем с максимальной энергоэффективностью и имеют несколько режимов управления энергопотреблением. В них используются внутренние интерфейсы памяти шириной больше, чем средняя длина инструкции. Это минимизирует число доступов к шине памяти, а следовательно, и потребление электроэнергии, связанное с операциями по шине и чтением энергозависимой памяти. Технология непрерывной обработки прерываний с исключением внутренних операций над стеком (tail chaining) сокращает время реакции на прерывания и исключает лишние операции со стеком.

На рисунке А.2 представлено упрощенное представление цифрового периферийного устройства.

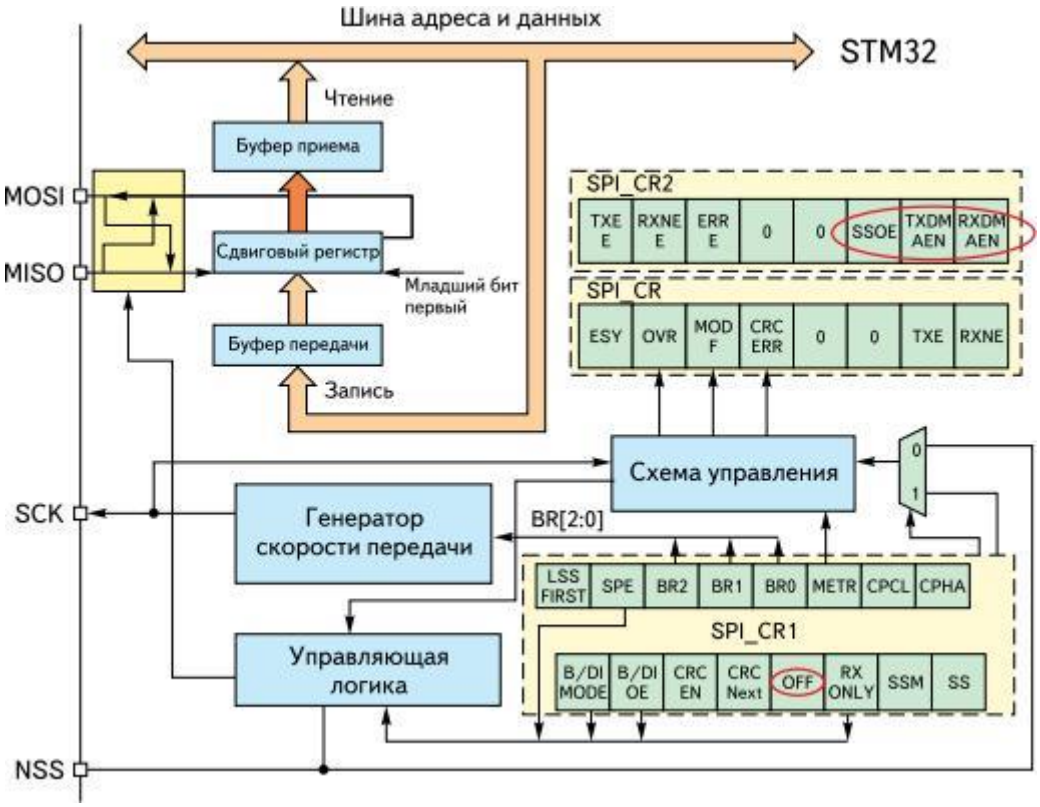

Рисунок А.2 – Представление цифрового периферийного устройства

Периферийный узел может быть разделен на два главных блока. Первый блок – это ядро, которое содержит конечные автоматы, счетчики и любой вид комбинаторной или последовательной логики. Оно предназначено для выполнения задач, не требующих участия процессора, таких как простые задачи передачи данных, управления аналоговыми входами или выполнения функций, привязанных к синхросигналам. Ядро периферийного узла связывается с внешним миром через порты ввода/вывода МК. Внешние соединения могут состоять из нескольких сигналов или сложных шин. Второй блок – настройка и управление периферией, которые осуществляются приложением через регистры, соединенные с внутренней шиной, разделяемой с другими ресурсами МК.

Краткое описание платы STM32F4 Discovery

Плата STM32F4 Discovery (рисунок А.3) предназначена для ознакомления с возможностями 32-битного МК на основе ARM-архитектуры, а также для реализации собственных устройств и приложений с использованием аппаратного обеспечения платы [2].

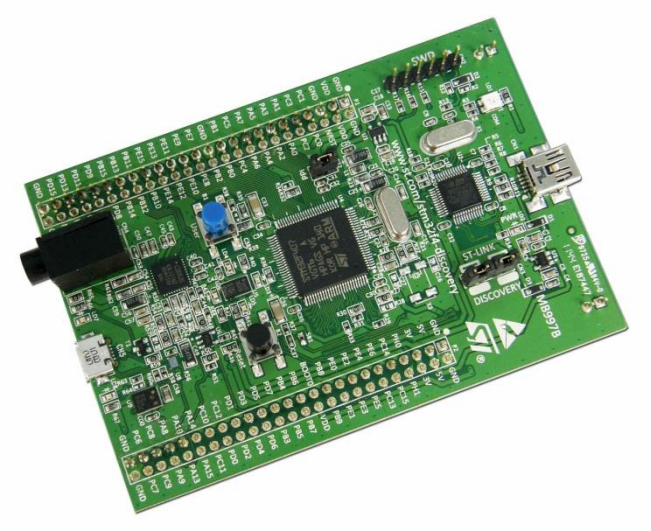

Рисунок А.3 – Внешний вид платы STM32F4 Discovery

Плата STM32F4 Discovery оснащена:

- микроконтроллером STM32F407VGT6 с ядром Cortex-M4F тактовой частотой 168 МГц, 1 Мбайт Flash-памяти, 192 Кбайт RAM в корпусе LQFP100;
- отладчиком ST-Link/V2 для отладки и программирования МК;
- питанием платы через USB или от внешнего источника питания (5 В);
- датчиком движения ST MEMS LIS302DL и выходами цифрового акселерометра по трем осям;
- датчиком звука ST MEMS MP45DT02;
- звуковым ЦАП CS43L22;
- восемью светодиодами: LD1 (красный/зеленый) для USB-подключения; LD2 (красный) для питания 3.3 В; четыре пользовательские светодиода: LD3 (оранжевый), LD4 (зеленый), LD5 (красный), LD6 (синий); два светодиода для USB On-The-Go – LD7 (зеленый) и LD8 (красный);
- двумя кнопками (для программирования пользователем и перезапуска).

Таким образом, отладочная плата оснащена большим количеством периферии, что позволяет сразу же реализовывать на ней примеры различной сложности.

ПРИЛОЖЕНИЕ Б

НАЧАЛО РАБОТЫ С ОТЛАДОЧНОЙ ПЛАТОЙ STM32F4 DISCOVERY

Подключение платы и установка среды разработки

Несмотря на внешнюю сложность платы, ее устройство хорошо продумано, поэтому даже абсолютным новичкам не составит труда освоить азы работы с STM32F4 Discovery.

Рассмотрим программное обеспечение, необходимое для подключения. Для подключения платы к ПК потребуется кабель USB типа A и mini-USB типа B (рисунок Б.1). В комплекте поставки такой шнур отсутствует.

Рисунок Б.1 – Кабель для подключения платы к компьютеру

При подключении следует быть внимательным: на плате есть два разъема USB, один из которых предназначен для подключения, а второй – для реализации работы с USB.

Предполагается, что на компьютере установлена операционная система семейства Windows (XP, Vista, Windows 7 и др.). Существует возможность подключения к ПК с Linux, но это потребует намного больше усилий по установке программного обеспечения.

Для начала работы следует установить среду разработки. Рассмотрим детали установки и использования среды разработки Keil uVision.

Скачать программу можно с сайта производителя https://www.keil.com/download/product. Выбрав для загрузки пункт MDK-ARM последней версии, попадаем на страницу, на которой необходимо ввести свои персональные данные (рисунок Б.2).

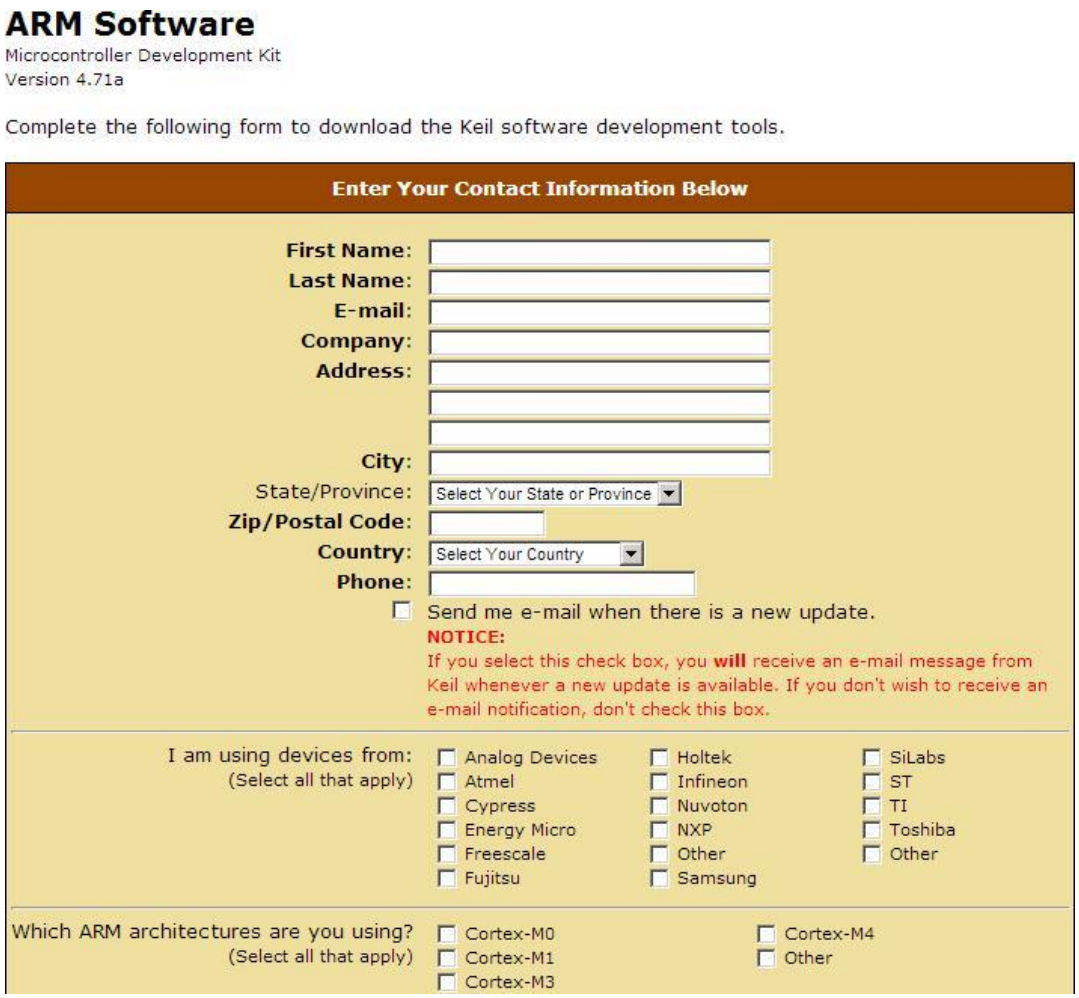

Рисунок Б.2 – Форма регистрации для загрузки Keil

После заполнения формы получаем возможность скачать программный продукт.

Установка потребует достаточно много свободного пространства (около 2,5 Гбайт), поэтому следует заранее подготовить необходимые ресурсы. После запуска установки откроется следующее окно (рисунок Б.3).

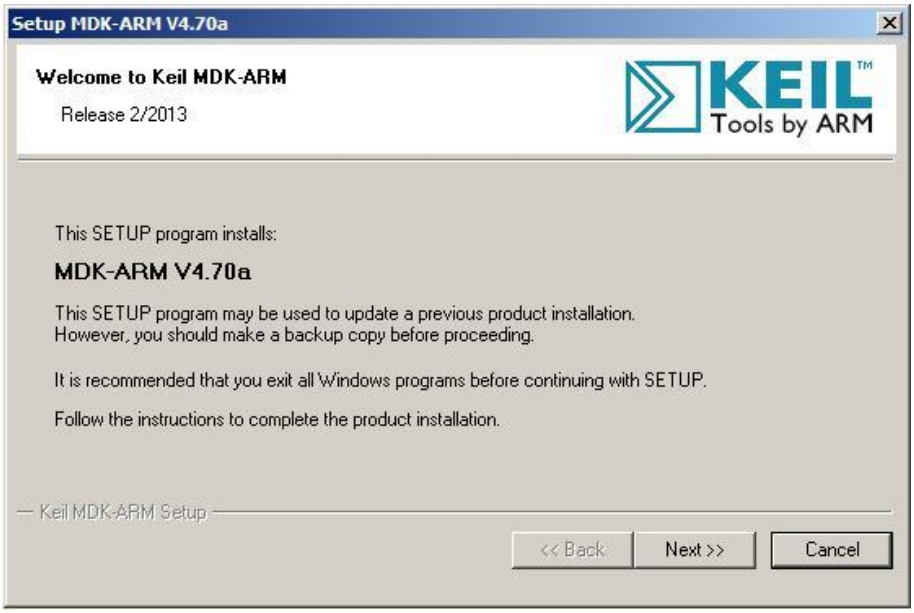

Рисунок Б.3 – Окно установщика Keil

Далее необходимо принять условия лицензионного соглашения, выбрать директорию для инсталляции (рисунок Б.4), а также указать информацию о пользователе.

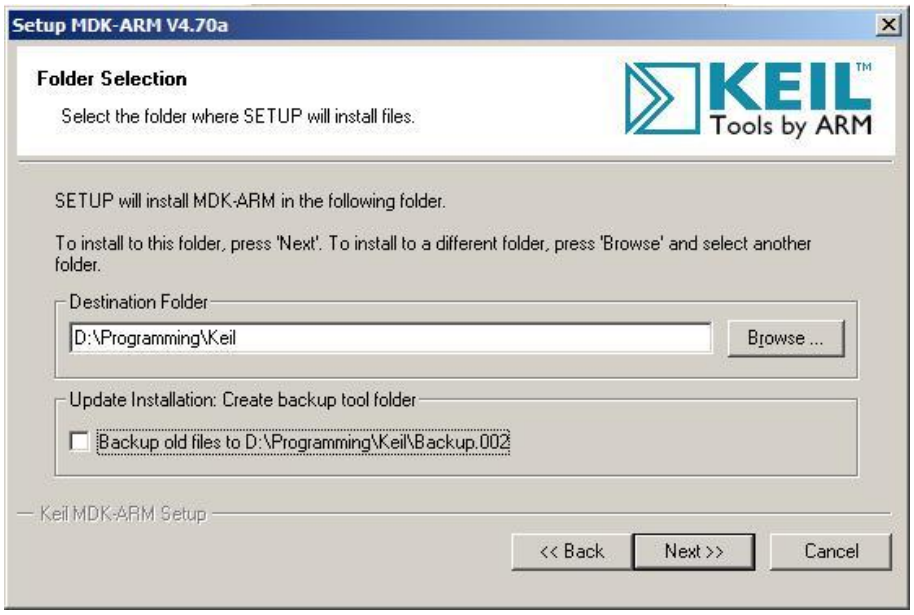

Рисунок Б.4 – Выбор директории для инсталляции

После этого начнется непосредственно процесс установки.

Кроме самой среды разработки для написания программ необходимы также дополнительные библиотеки Cortex Microcontroller Software Interface Standard (CMSIS) и Standard Peripheral Library (SPL). Лучше всего загрузить данные библиотеки с сайта производителя st.com.

Библиотека SPL служит для управления всеми основными устройствами, входящими в состав микроконтроллера: USART, SPI, DMA, ADC, DAC и др. Основным достоинством библиотеки является то, что она делает более понятным управление микроконтроллером для разработчика, в том числе и начинающего, который еще не знает всех тонкостей настроек. Кроме того, вместе с самой библиотекой распространяются и демонстрационные проекты для работы с основными устройствами, на которые можно опираться при написании собственных проектов.

После открытия окно среды выглядит следующим образом (рисунок Б.5).

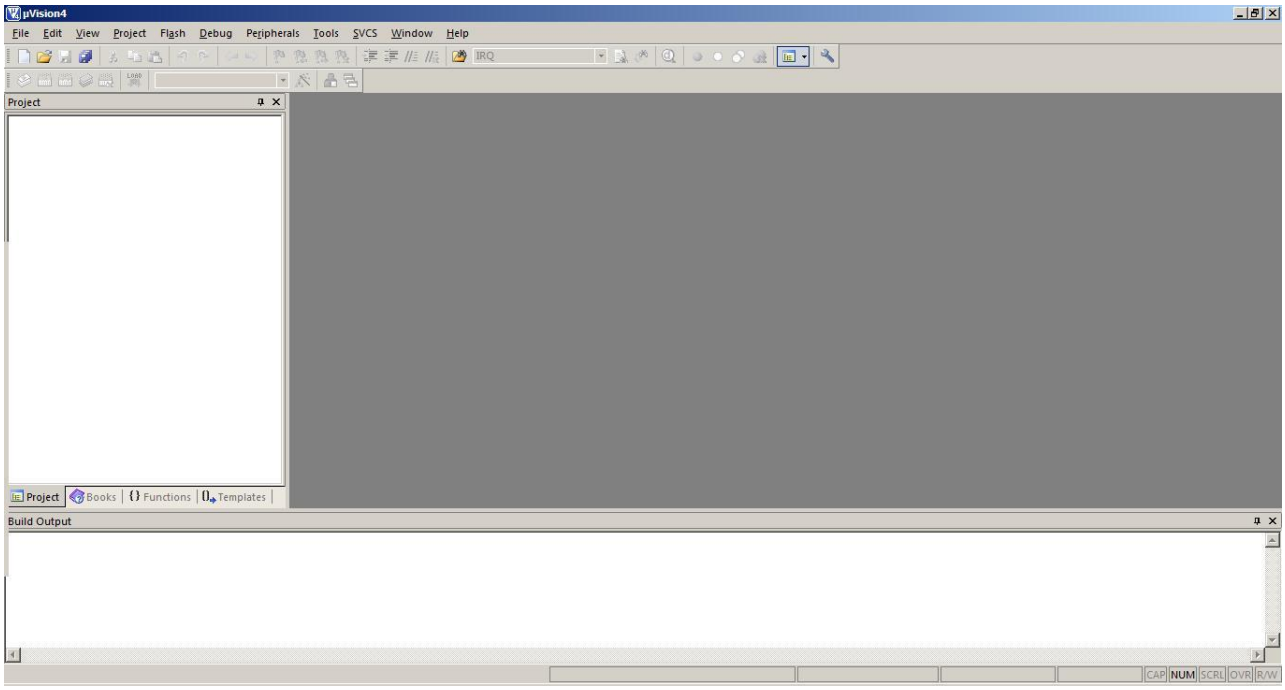

Рисунок Б.5 – Окно среды разработки

Теперь можно создать проект для устройства. Для этого выполним команду меню Project/New uVision Project. В результате появится диалоговое окно для выбора папки расположения проекта. Для каждого проекта желательно создавать отдельную папку, поскольку проект может разрастаться и содержать большое количество файлов, поэтому держать два проекта в одной папке будет неудобно. После выбора папки следует выбрать микроконтроллер, для которого планируется написание программы. Выберем производителя STMicroelectronics и конкретную модель контроллера STM32F407VG (рисунок Б.6).

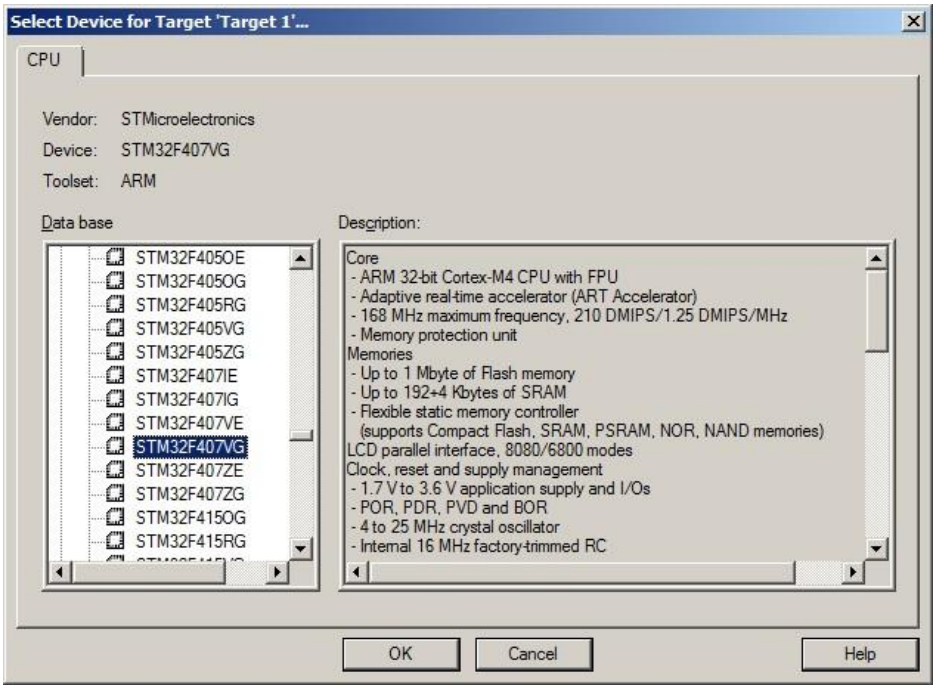

Рисунок Б.6 – Выбор производителя STMicroelectronics и модели микроконтроллера STM32F407VG

Далее откроется диалоговое окно с предложением добавить к проекту файл запуска startup_stm32f4xx.s, которое нужно подтвердить. Среда разработки создаст проект, а также группу в нем для исходных файлов, в которой и будет расположен упомянутый выше файл. Группы файлов служат для удобной организации представления файлов в проекте. В данном примере будем использовать четыре группы файлов: Startup, User, CMSIS и SPL. Для начала переименуем созданную по умолчанию группу и назовем ее Startup и добавим еще три группы с помощью контекстного меню проекта в панели Project, выполнив команду Add Group.

Теперь добавим необходимые файлы в проект. Перед добавлением файлов желательно скопировать их в директорию проекта. Из библиотеки CMSIS понадобятся следующие файлы, которые находятся в директории, где распакован архив с библиотекой:

- stm32f4xx.h;
- system_stm32f4xx.h;
- system_stm32f4xx.c;
- stm32f4xx_conf.h;
- core_cm4.h.

В группу SPL нужно добавить файлы из библиотеки SPL для работы с устройствами. Необходимо добавлять пару исходного и заголовочного файлов. Например, для работы с портами ввода/вывода следует добавить файлы

stm32f4xx_gpio.c и stm32f4xx_gpio.h. Обязательно следует подключить файлы для управления устройством сброса и тактирования, а именно stm32f4xx_rcc.c и stm32f4xx_rcc.h. Они содержат функции для включения тактирования периферийных устройств.

После этого остается добавить еще один файл, в котором содержится функция main. Вначале создадим и сохраним его с помощью команд меню File, а затем добавим его в группу Users. В результате получим проект со следующей структурой (рисунок Б.7).

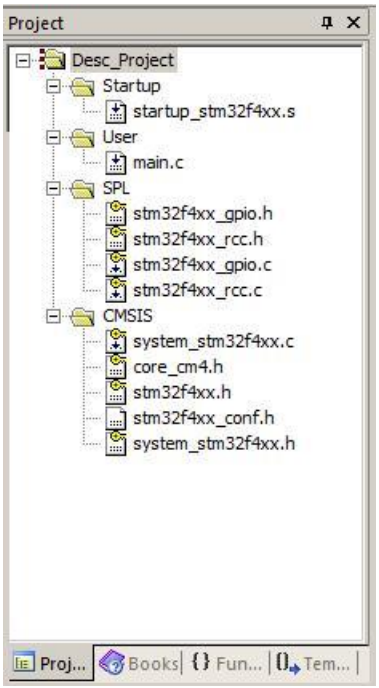

Рисунок Б.7 – Созданный проект

Осталось сделать еще некоторые настройки, которые нужны для компиляции кода и выполнения отладки. Настройки производятся в окне настроек, которое открывается через команду Project/Options for target… или кнопкой на панели инструментов, или используя комбинацию клавиш Alt + F7. В открывшемся окне на вкладке C/C++ зададим определения USE_STDPERIPH_DRIVER, STM32F4XX в текстовом поле Define. Здесь же можно задать пути к директориям с заголовочными файлами.

Настройки отладки по умолчанию позволяют проводить отладку в режиме симуляции, однако, если устройство подключено к компьютеру, то лучше производить отладку на самом устройстве. Для этого необходимо на вкладке Debug для пункта Use указать ST-Link Debugger. После этого следует нажать кнопку Settings. На вкладке Debug значение Port установить как SW.

Затем перейти к вкладке Trace Download и добавить в список Programming Algorithm устройство STM32F4xx (рисунок Б.8).

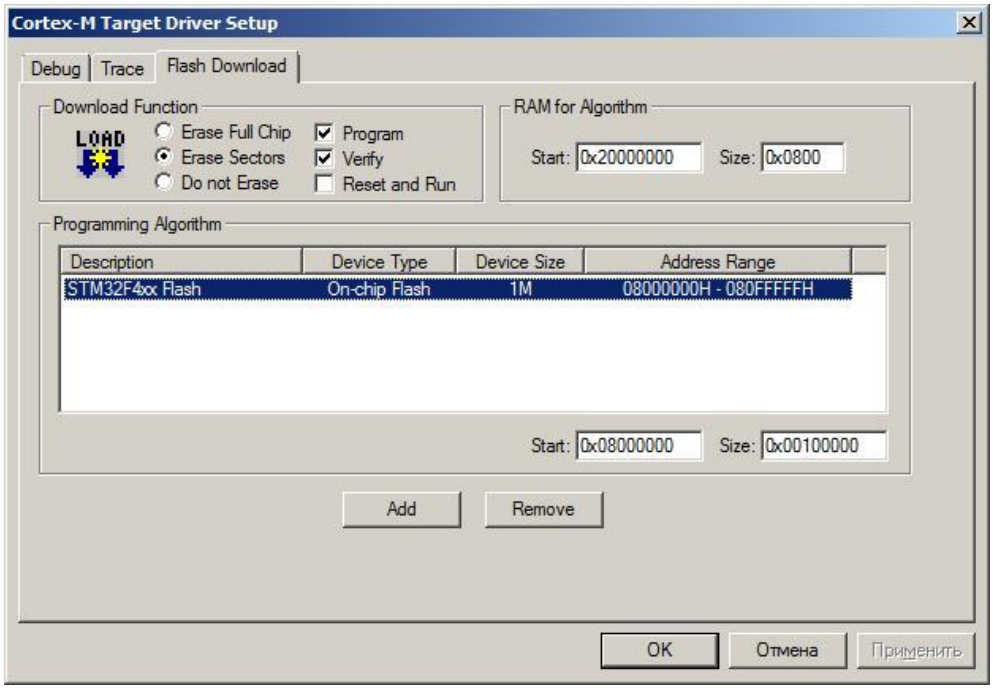

Рисунок Б.8 – Добавление в список Programming Algorithm устройства STM32F4xx

На вкладке Utilities выбрать вариант Use Target Driver for Flash Programming и также установить значение ST-Link Debugger. Проверить, что в окне после нажатия кнопки Settings значения совпадают с заданными для отладки.

После проведения данных действий можно приступать к написанию программы и проведению отладки на демонстрационной плате. Однако перед этим следует подключить плату к ПК.

Для работы с платой необходимо, чтобы прошла установка драйверов. Обычно при установке средств разработки, например Keil, происходит также установка соответствующего программного обеспечения (если этого не произошло, то следует найти в директории установки необходимые программы и установить их самостоятельно), поэтому рекомендуется производить указанные ниже действия уже после установки одной из сред разработки. Если же вариант с установкой среды разработки не подходит, то следует загрузить и установить программу STM32 ST-Link Utility, которая также рассматривается в приложении.

Процесс установки драйверов при подключении устройства рассмотрим на примере операционной системы Windows 7.

При подключении устройства сразу же начинается установка драйверов для нового устройства. В области системного трея появляется уведомление следующего вида (рисунок Б.9).

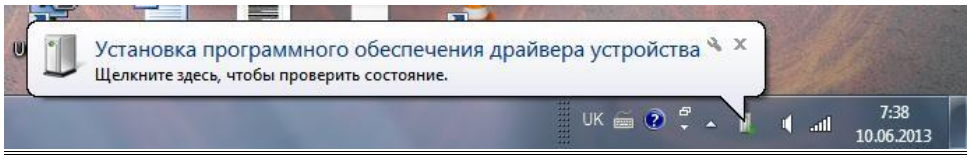

Рисунок Б.9 – Уведомление при подключении устройства

Нажав на уведомление левой кнопкой мыши, откроем окно, которое отображает динамику установки драйверов (рисунок Б.10).

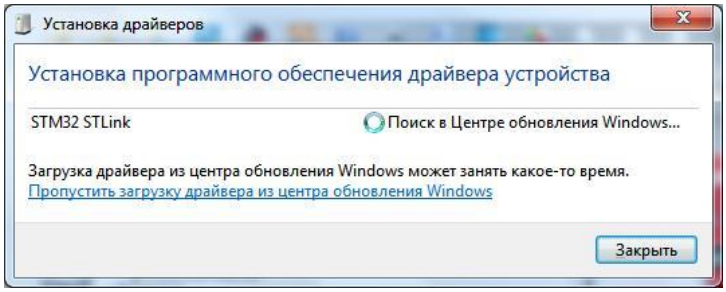

Рисунок Б.10 – Динамика установки драйвера

В результате успешной установки наблюдаем сообщение следующего вида, которое говорит о том, что имя подключенного устройства – STMicroelectronics STLink Dongle (рисунок Б.11).

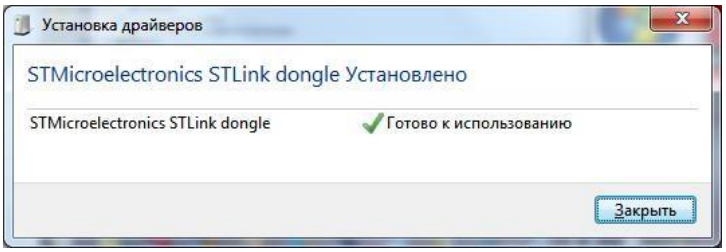

Рисунок Б.11 – Сообщение при успешной установке

В Диспетчере устройств (Пуск/Компьютер/Свойства/Диспетчер устройств) можем найти подключенное устройство (рисунок Б.12).

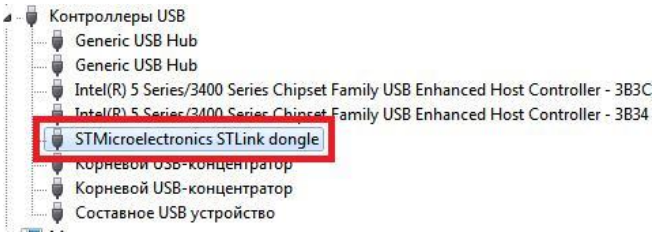

Рисунок Б.12 – Поиск подключенного устройства в Диспетчере устройств

Теперь устройство готово для взаимодействия с программным обеспечением, установленным на ПК.

Дополнительное программное обеспечение

Рассмотрим графические средства настройки микроконтроллеров STM32. Производитель микроконтроллеров STMicroelectronics предлагает программу с возможностью графической настройки микроконтроллеров серии STM32 – MicroXplorer [12]. Она позволяет получить исходный код инициализации периферийных устройств и соответствующих выводов и отображает результат настройки (занятые выводы, режим работы и др.). Тем не менее с ее помощью нельзя настроить работу с прерываниями для конкретного устройства, провести инициализацию устройств, входящих в состав контроллера, поэтому данный инструмент больше подходит для наглядности при разработке, а не как полноценная альтернатива написания кода инициализации.

Программа представляет собой плагин к среде разработки Eclips. Соответственно, предварительно необходимо установить Eclipse, а затем добавить к ней плагин.

Eclipse можно скачать с сайта https://www.eclipse.org. При этом загружается архив, который необходимо распаковать и создать нужные ярлыки для запуска. Для использования плагина подходят последние версии программного продукта.

Сам плагин доступен для загрузки на сайте производителя https://www.st.com/en/development-tools/stsw-stm32095.html. В результате также загружается архив.

Установка расширения производится из среды разработки. Для этого выбираем команду меню Help/Install New Software. В открывшемся окне нажимаем кнопку Add и в диалоговом окне, нажав кнопку Archive, указываем путь к расширению. Называем полученное расположение и переходим к процессу установки. Отмечаем пункт MicroXplorer, который появился в списке ниже. Далее соглашаемся с условиями лицензионного соглашения и во время установки подтверждаем запросы, которые появляются от диалоговых окон.

После окончания процесса среда предложит перезагрузить среду разработки для того, чтобы применить изменения и загрузить расширение.

После перезагрузки в главном меню можно увидеть новый пункт – MicroXplorer (рисунок Б.13).

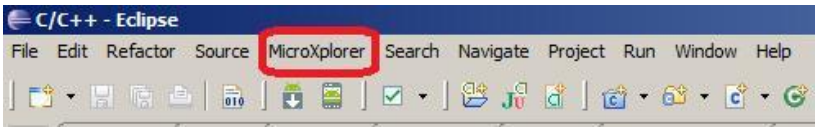

Рисунок Б.13 – Новый пункт меню

С помощью него можно на выбор открыть расширение в отдельном представлении Eclipse или установить функциональность MicroXplorer, доступной во всем расположении (respective) Eclipse.

Рассмотрим работу в режиме отдельного представления. Выполним команду MicroXplorer/Open MicroXplorer As New View. В результате среди прочих откроется еще одно окно, которое можно развернуть на всю ширину рабочей области, дважды щелкнув по заголовку. В результате окно приложения выглядит следующим образом (рисунок Б.14).

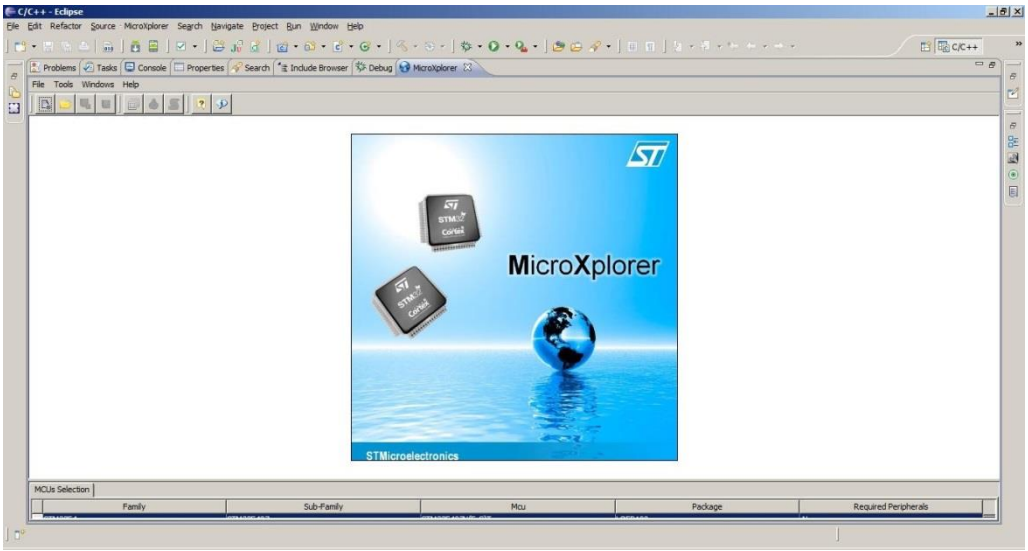

Рисунок Б.14 – Окно приложения в Eclipse

Для начала необходимо задать, какое именно устройство используется, выполнив команду Tools/MCUs Selector. После этого откроется диалоговое окно (рисунок Б.15), в котором в списке указываем интересующий микроконтроллер и нажимаем кнопку OK.

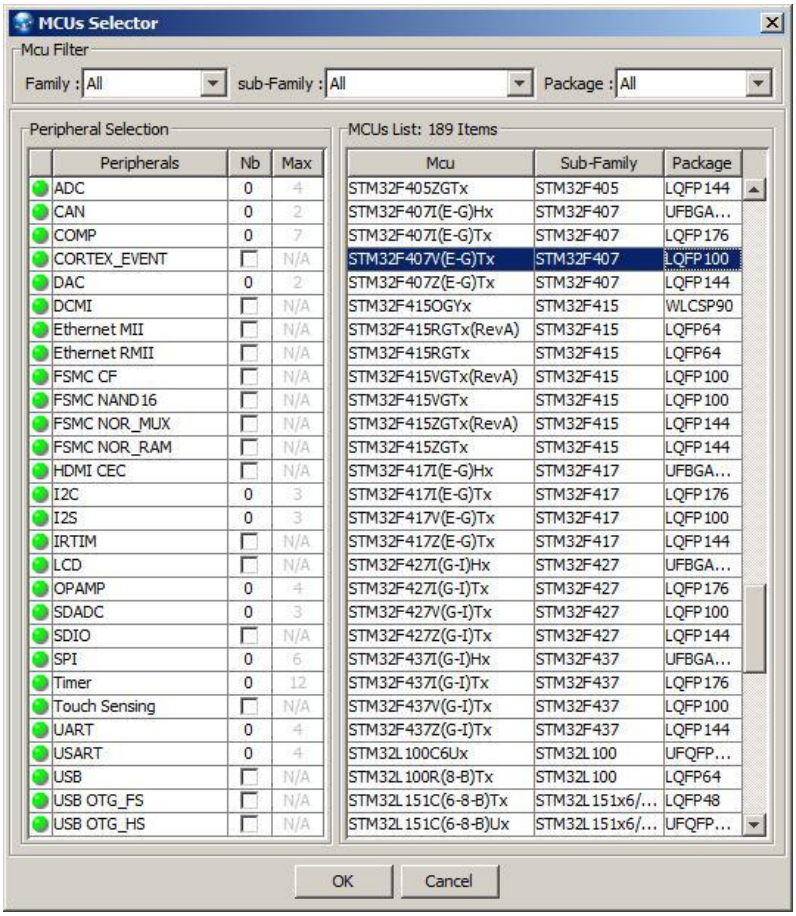

Рисунок Б.15 – Окно выбора микроконтроллера в Eclipse

После этого в рабочей области появится панель с устройствами, входящими в состав контроллера, а также его условное изображение, на котором обозначены все основные выводы (рисунок Б.16). Сама рабочая область представлена тремя вкладками: Pinout, Configuration, Power Consumption Calculator. По мере того как происходит настройка и включение соответствующих устройств, выводы, которые будут задействованы, выделяются зеленым цветом. В то же время некоторые из устройств в панели помечаются желтым или красным цветом. Это происходит из-за того, что устройства используют одинаковые выводы в своей работе, поэтому может происходить частичный (использование устройства все еще возможно) либо полный конфликт (задействовать устройство не удастся), о чем и сообщают соответствующие цвета. Таким образом, можно выявить и избежать подобных конфликтов уже на стадии проектирования, что значительно упрощает дальнейшую разработку.

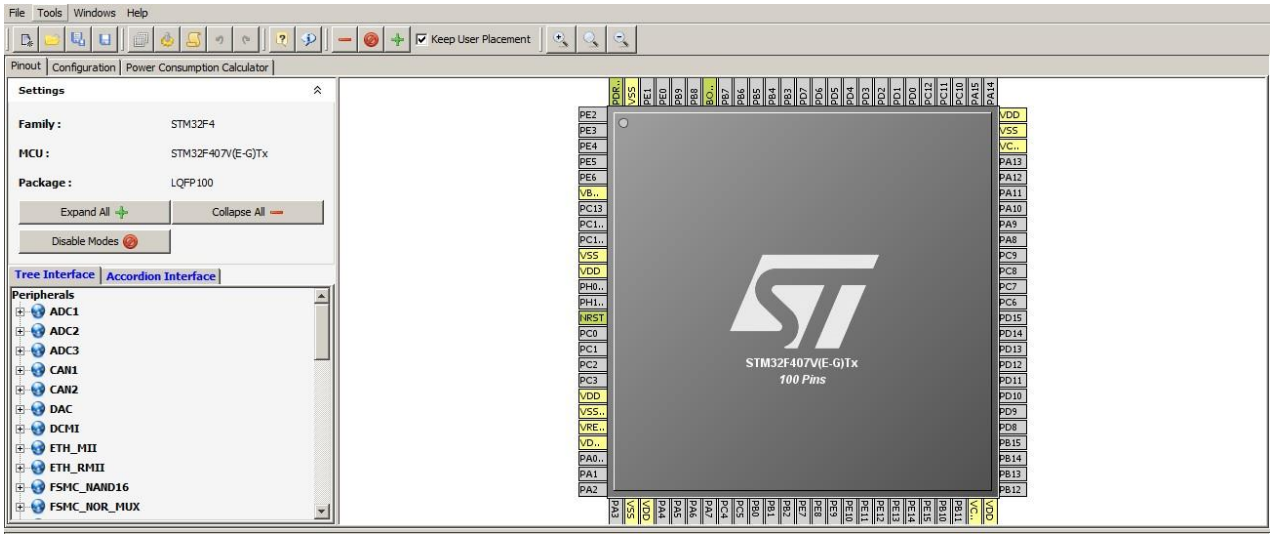

Рисунок Б.16 – Условное графическое обозначение контроллера

Рассмотрим пример настройки первого АЦП (ADC1) в визуальном редакторе. Предположим, что необходимо принимать сигнал с нулевого входа. Эту настройку задаем, развернув узел ADC1 и установив галочку в поле выбора IN0 (рисунок Б.17). Вывод PA0 теперь должен быть выделен зеленым, а возле него появится подпись ADC1_IN0. Кроме того, переместившись в панели устройств к другим АЦП (ADC2 и ADC3), можно увидеть, что они имеют желтый цвет. Это связано с тем, что нулевой вход всех АЦП одинаков, поэтому одновременно несколькими устройствами он использоваться не может. Тем не менее остальные АЦП все еще могут использоваться, но проводить измерения они будут не с нулевого входа.

Рисунок Б.17 – Установка входа АЦП

После этого перейдем на вкладку Configuration (рисунок Б.18). Данная вкладка предназначена для настроек портов ввода/вывода в соответствии с выбранной периферией контроллера.

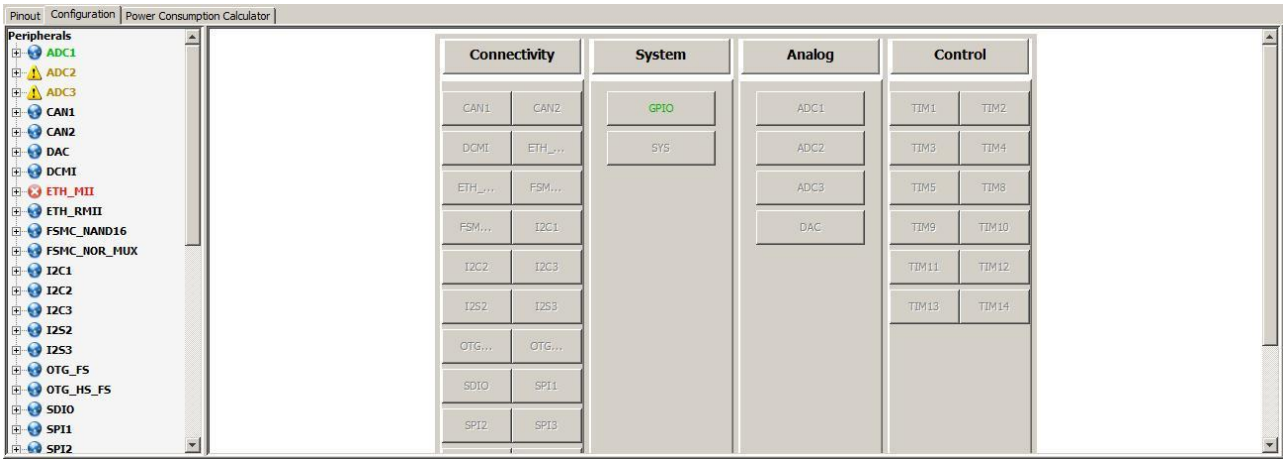

Рисунок Б.18 – Вкладка Configuration

После нажатия кнопки GPIO откроется окно Pin Configuration (рисунок Б.19). Именно в нем и производится настройка работы выводов.

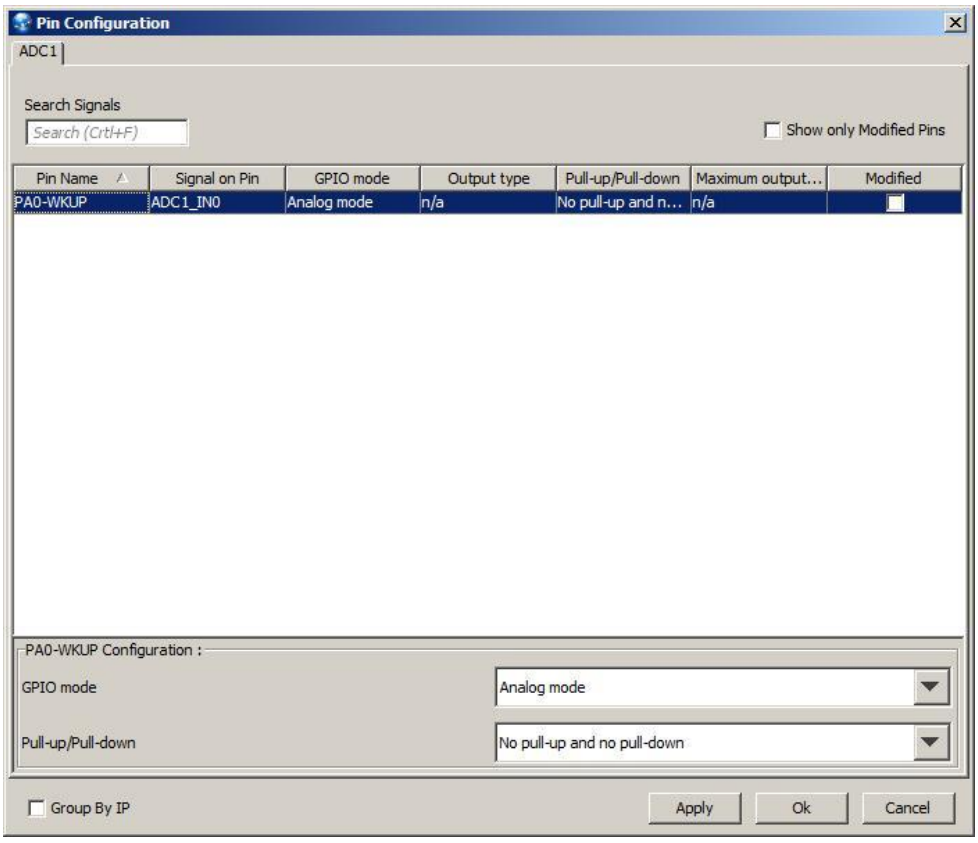

Рисунок Б.19 – Окно настройки выводов

В нашем случае сгенерированная настройка не требует изменений, однако при использовании других устройств параметры по умолчанию следует изменять для их корректной работы. При этом на каждое выбранное устройство появляется своя вкладка в окне, что позволяет сразу же произвести полную настройку.

Завершающим этапом работы с дополнением является генерация программного кода инициализации. Она производится командой меню Tools/Generate Code… . В результате создаются папки с файлами исходных кодов и заголовочных файлов. Как говорилось ранее, код, генерируемый программой, нужно дорабатывать, т. к. многие важные настройки не генерируются программой. Лучше всего скопировать код, который выполняет инициализацию портов в свою программу, а остальное настраивать самостоятельно.

Среда разработки CoIDE

В качестве альтернативы для начинающих опишем также среду разработки CoIDE, которая намного проще в использовании по сравнению с другими программными продуктами.

Для разработки программного обеспечения под 32-разрядные процессоры ARM (в том числе и для рассматриваемой платы) рекомендуются следующие программные продукты:

- Atollic TrueSTUDIO;
- IAR Embedded Workbench;
- Keil uVision с инструментами MDK-ARM;
- Altium TASKING VX-Toolset.

Все перечисленные среды являются коммерческими. Бесплатное их использование возможно лишь с ограничениями по размеру бинарного кода программы или по срокам использования.

Наиболее известные бесплатные альтернативы – CooCox IDE и использование Eclipse вместе с плагином для разработки 32-разрядных процессоров. Второй вариант требует множество дополнительных настроек и достаточно сложен для тех, кто только начинает разработку под МК.

Среда разработки CooCox CoIDE 1.7 – бесплатный инструмент, который ориентирован на разработку для 32-разрядных процессоров ARM разных производителей: Atmel, Energy Micro, Freescale, Holtek, NXP, Nuvoton, TI. Она построена на основе Eclipse, однако не имеет таких же возможностей для расширения. Несмотря на это данная среда требует минимальных настроек для начала программирования и отладки, поэтому представляется идеальным вариантом для начала работы с платой STM32F4 Discovery.

Для начала разработки, кроме самой CoIDE, потребуется установка средств для построения проекта GNU Toolchain for ARM Embedded Processors [3].

Разработка с использованием CoIDE связана с концепцией репозитория: все доступные компоненты и библиотеки устанавливаются с помощью мастера,

исходя из того, для какого именно процессора создан проект. Для загрузки библиотек необходимо подключение к сети Интернет.

Средства настройки частоты работы микроконтроллера

По умолчанию рабочая частота микроконтроллера на отладочной плате составляет 16 МГц. Данную частоту обеспечивает внутренний высокочастотный генератор тактовых импульсов (High Speed Internal Oscillator, HSI). Однако максимальная рабочая частота составляет 168 МГц, что значительно больше значения по умолчанию. Для того чтобы настроить работу устройства на нужной частоте, производитель предлагает воспользоваться специально разработанным программным обеспечением – Clock Configuration Tool. Оно представляет собой xls-файл с макросами, поэтому для его использования следует задать разрешение выполнения макросов в Excel. Интерфейс программы (рисунок Б.20) организован на одном листе с использованием полей ввода и выбора значений и наглядно отображает схему тактирования, которая используется в устройстве.

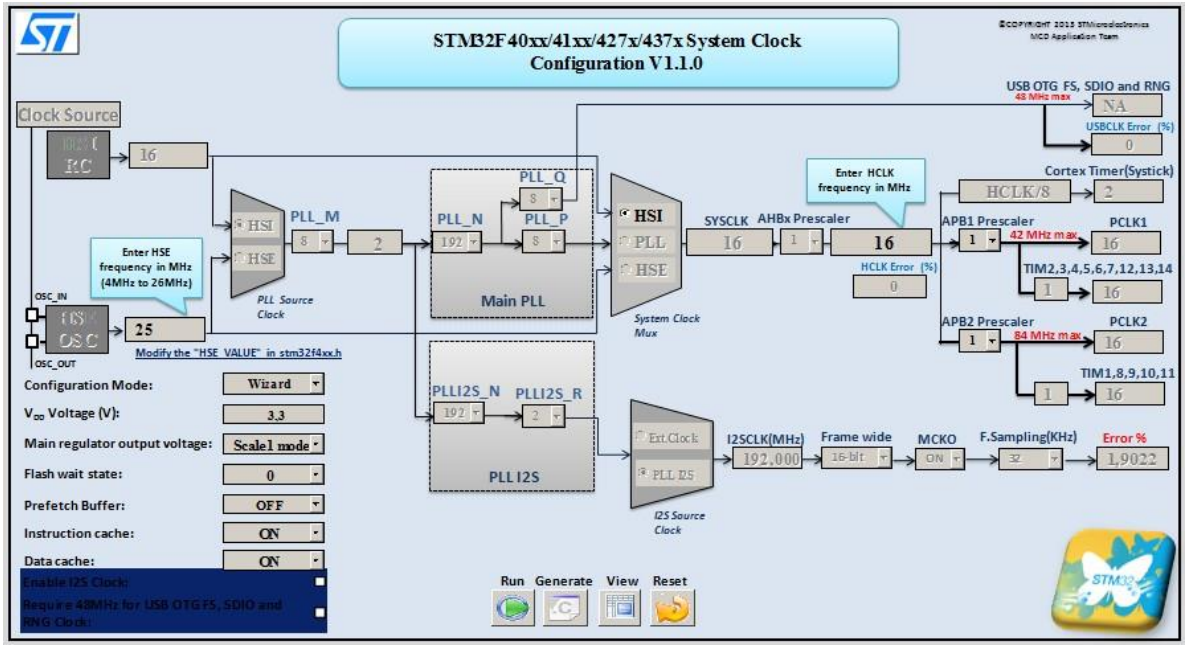

Рисунок Б.20 – Интерфейс программы для настройки системы тактирования

В нижней части схемы находятся кнопки, с помощью которых запускаются макросы, выполняющие основную работу. При нажатии кнопки Generate в папке с файлом генерируется новый файл конфигурации с именем system_stm32f4xx.c. Его можно подключить к проекту в среде разработки, заменив существующий файл. Для того чтобы применить настройки, следует отследить, вызывается ли функция SystemInit(), поскольку без ее вызова

устройство будет работать на частоте по умолчанию. В случае отсутствия вызова функции ее следует вызвать самостоятельно, например, в основной функции main. После этого устройство будет работать на той частоте, которая задана, поэтому, возможно, следует изменить значения параметров работы устройств для поддержания корректной работы в дальнейшем.

Программные средства для прошивки платы

Несмотря на то что среды разработки интегрируют в себе функциональность по прошивке микроконтроллеров, полезно будет узнать, что существуют и отдельные программы, которые предназначены специально для работы с памятью устройств (программирование, очистка, проверка). Кроме того, при работе с режимами пониженного энергопотребления именно эти программы позволяют выполнить программирование независимо от текущего режима работы, с чем могут возникнуть проблемы при выполнении данных действий из среды разработки.

Рассмотрим два программных продукта, которые реализуют функциональность для выполнения прошивки без использования среды разработки – STM32 ST-LINK Utility и ST Visual Programmer.

Программа STM32 ST-LINK Utility предназначена для работы с 32-разрядными контроллерами через интерфейс ST-LINK (в том числе и с его второй версией). Ее интерфейс организован максимально просто и понятно (рисунок Б.21). Основная рабочая область организована в виде двух вкладок, в которых отображается память устройства, а также представлен двоичный файл для записи. Наиболее важные команды сосредоточены в меню Target. Если в момент открытия программы устройство не было подключено к ПК, то выполнить подключение следует командой Target/Connect. После этого в панели сообщений должно появиться сообщение об успешном подключении и информация о подключенном устройстве. Выполнить программирование устройства можно командой Target/Program… . При этом следует заранее открыть файл с программой. Стирание памяти программ устройства производится командой Target/Erase Chip.

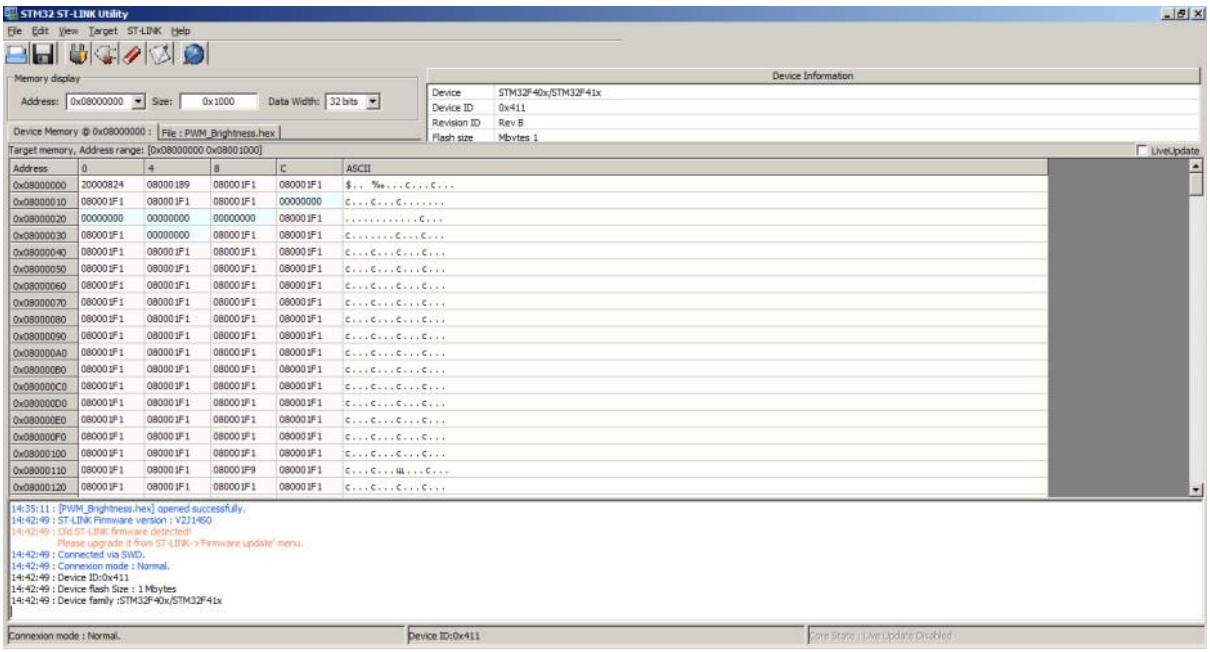

Рисунок Б.21 – Окно программы STM32 ST-LINK Utility

Программа ST Visual Programmer (STVP) является универсальным средством для прошивки микроконтроллеров производства STMicroelectronics. Она может работать с устройствами семейств STM32, STM8, STM7, а также с большим количеством интерфейсов, что является преимуществом. В остальном же STVP и STM32 ST-LINK Utility очень похожи между собой как по интерфейсу (рисунок Б.22), так и по функциональности.

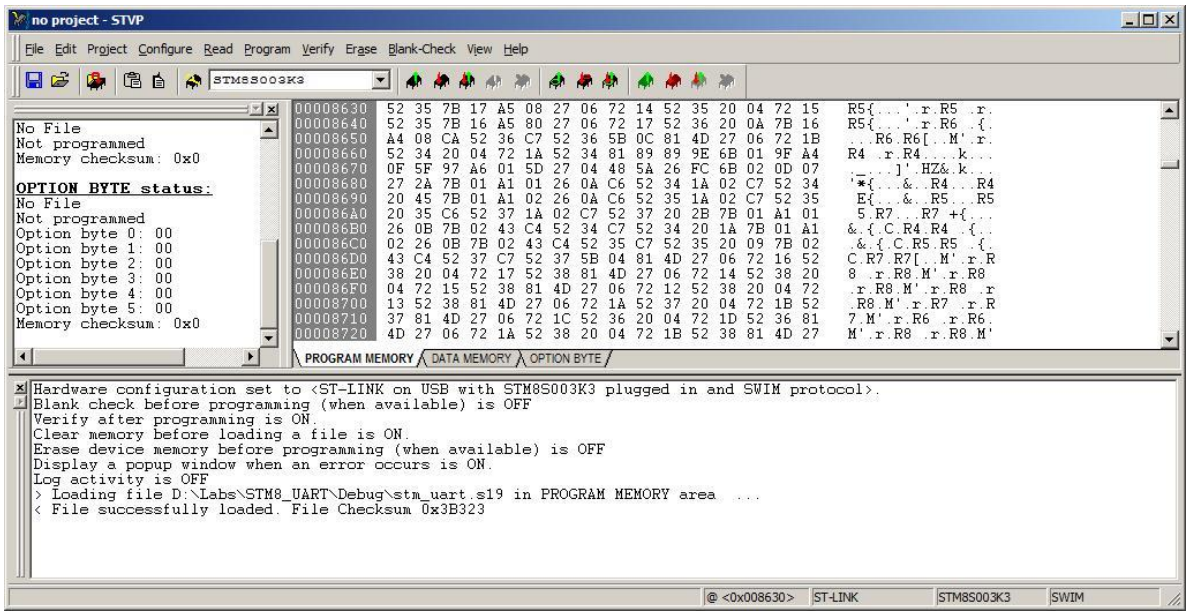

Рисунок Б.22 – Окно программы STVP

Использование STVP во многом аналогично использованию STM32 ST-LINK Utility, кроме того, что необходимо самостоятельно задать настройки

подключения. Эти действия выполняются в диалоговом окне (рисунок Б.23), которое открывается командой Configure/Configure ST Visual Programmer. В нем указывается аппаратное обеспечение, которое используется для подключения устройства.

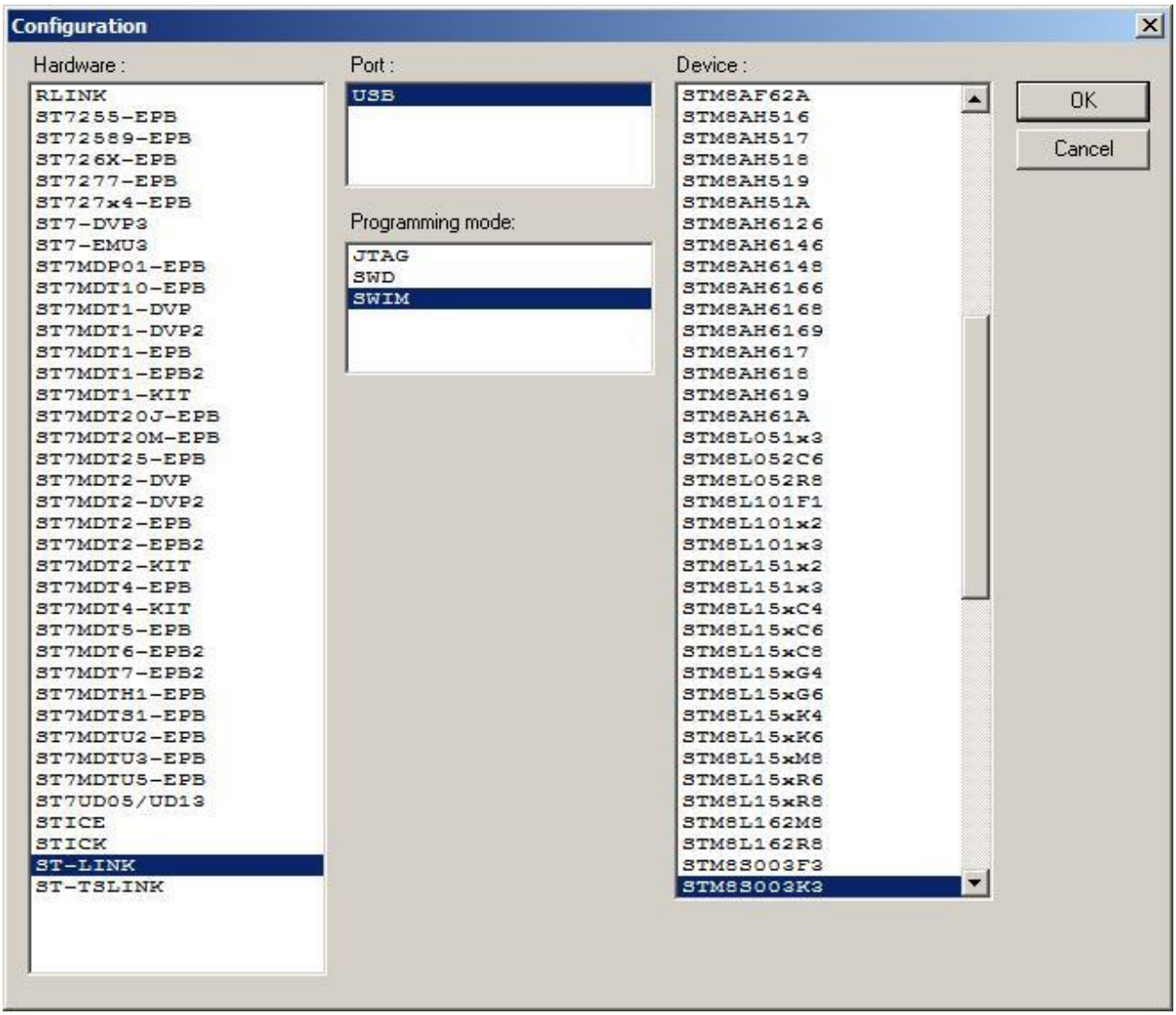

Рисунок Б.23 – Окно настройки программирования устройства

Работа с отладочной платой STM32F4 Discovery с использованием описанных инструментов происходит по схожему сценарию, поэтому предпочтение в их использовании зависит от пользователя. Автоматическое подключение при запуске облегчит работу новичкам, для тех, кто работает с различными устройствами, можно рекомендовать STVP.

Порядок работы с STM32CubeMX

STM32CubeMX представляет собой графический инструмент, который позволяет легко конфигурировать микроконтроллеры и микропроцессоры производства STM32, а также генерировать соответствующий С-код инициализации для ядра ARM Cortex-M или ARM Cortex-A.

Программное обеспечение CubeMX имеет следующие особенности:

– интуитивный выбор микроконтроллеров и микропроцессоров STM32;
– богатый и простой в использовании графический пользовательский интерфейс;
– генерируемый С-код имеет совместимость со многими популярными средами разработки (IAR Embedded Workbench, MDK-ARM и STM32CubeIDE);
– возможность создания дополнительных пакетов расширений с помощью STM32PackCreator и последующего их внедрения в проект;
– программное обеспечение распространяется на всех операционных системах (Linux, Windows, MacOS).

Рассмотрим порядок использования STM32CubeMX и ее интерфейс.

Запустим программу STM32CubeMX. После ее открытия нас приветствует начальное окно программы (рисунок Б.24).

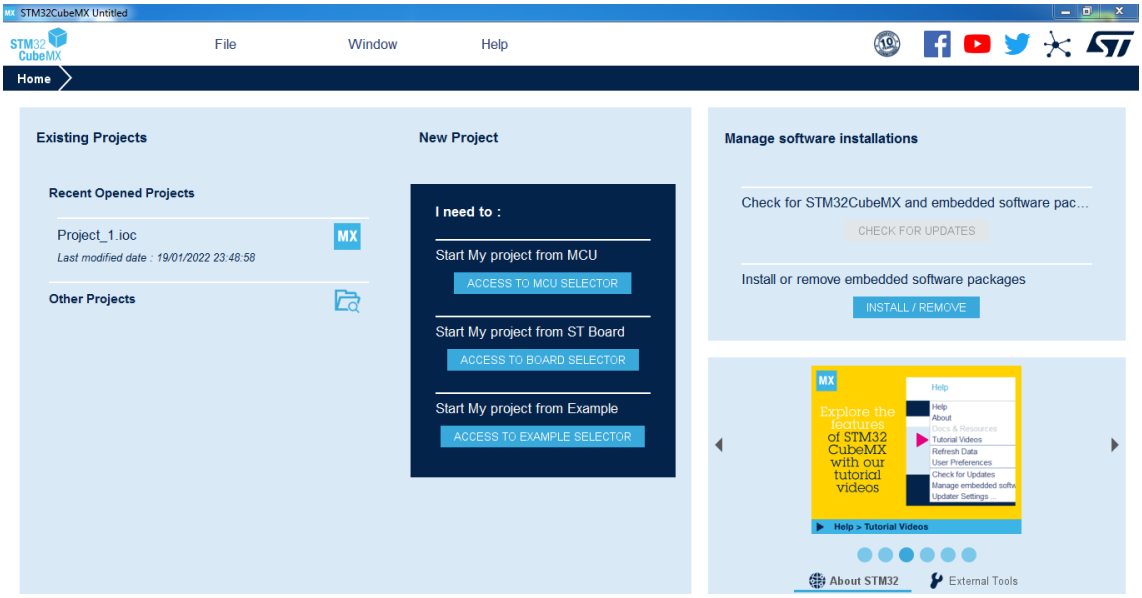

Рисунок Б.24 – Начальное окно STM32CubeMX

Для создания проекта нажмем File/New Project либо Ctrl + N (рисунок Б.25).

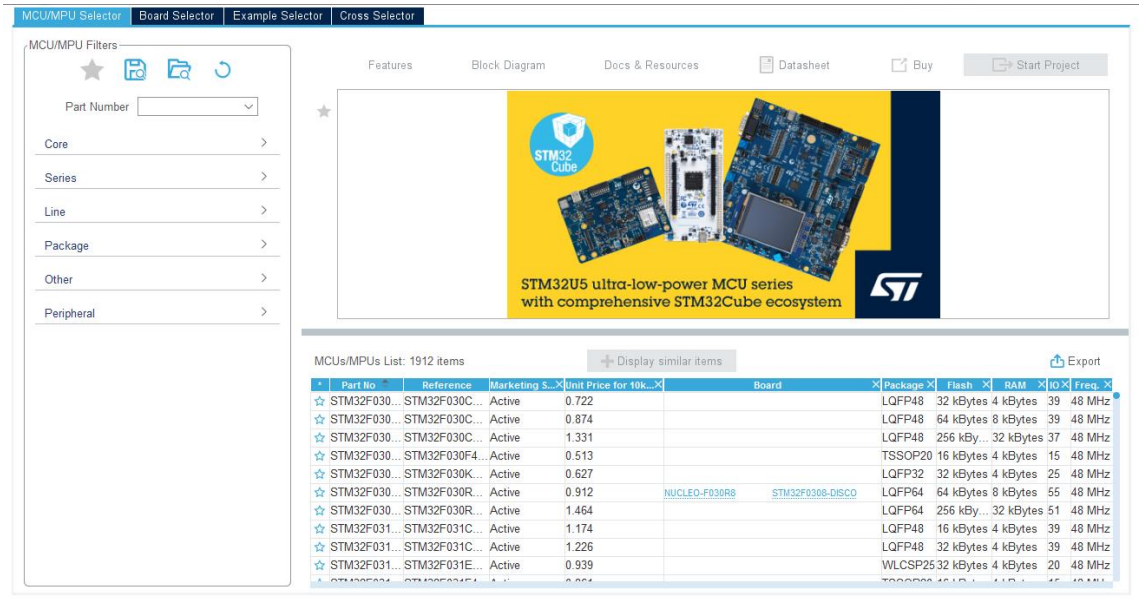

Рисунок Б.25 – Окно выбора микроконтроллера в STM32CubeMX

Из предложенного списка выберем микроконтроллер STM32F407VETx и нажмем Start Project.

В результате создания проекта появится следующее окно (рисунок Б.26).

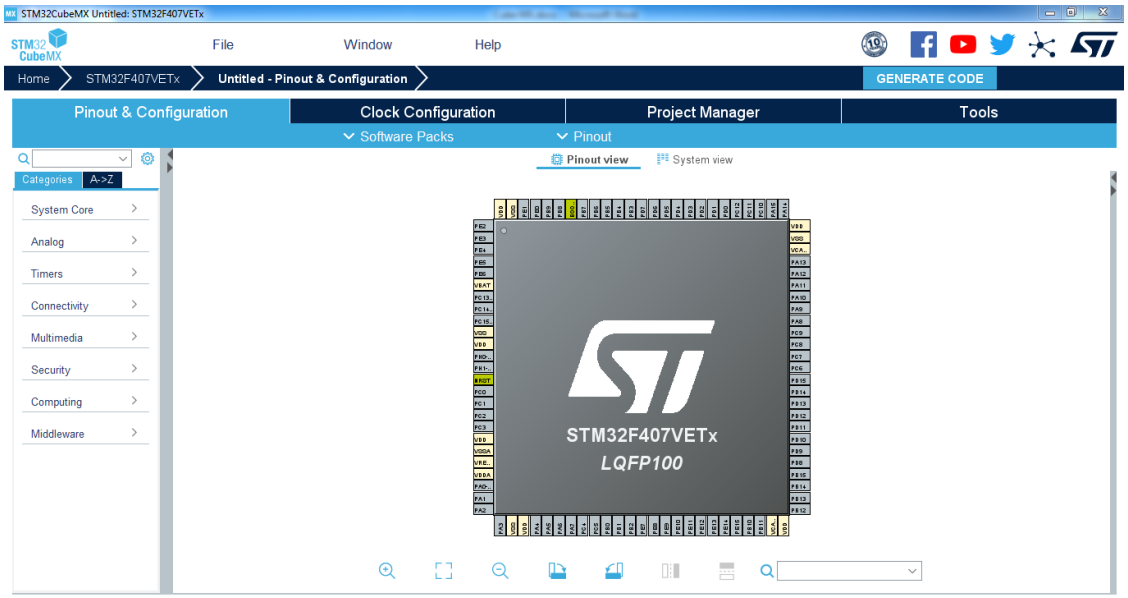

Рисунок Б.26 – Вкладка Pinout & Configuration

Зададим назначение выводов микроконтроллера. Для этого двойным щелчком правой кнопкой мыши выберем требуемый вывод (pin) и из выпадающего списка назначение вывода.

Перейдем на вкладку Clock Configuration. Окно программы примет следующий вид (рисунок Б.27).

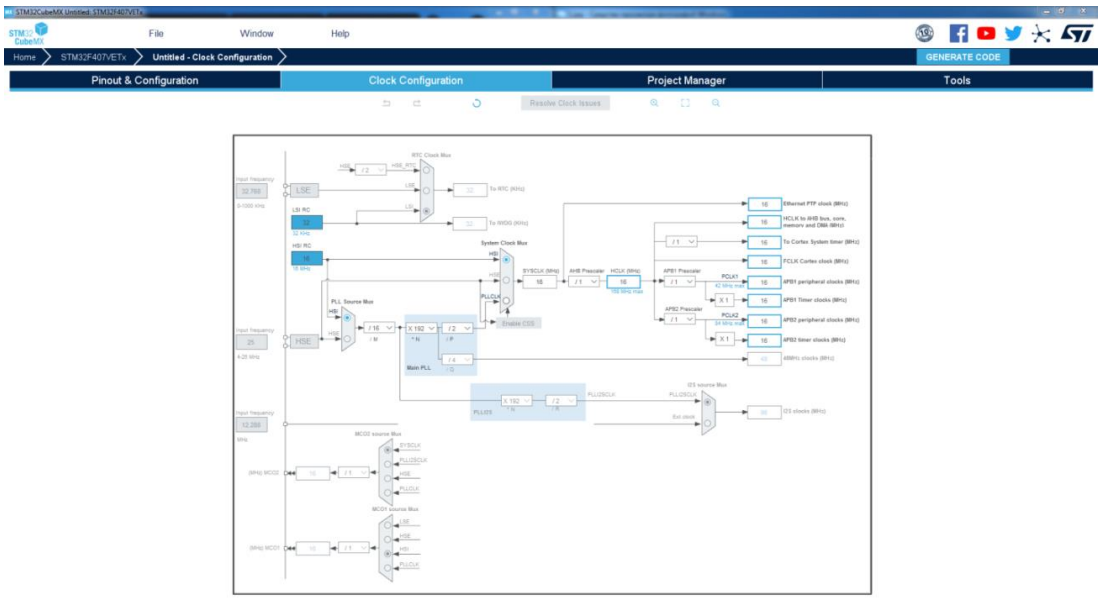

Рисунок Б.27 – Вкладка Clock Configuration

При необходимости здесь можно более детально произвести настройку выбранного микроконтроллера.

Перейдем на вкладку Project Manager. Окно программы примет следующий вид (рисунок Б.28).

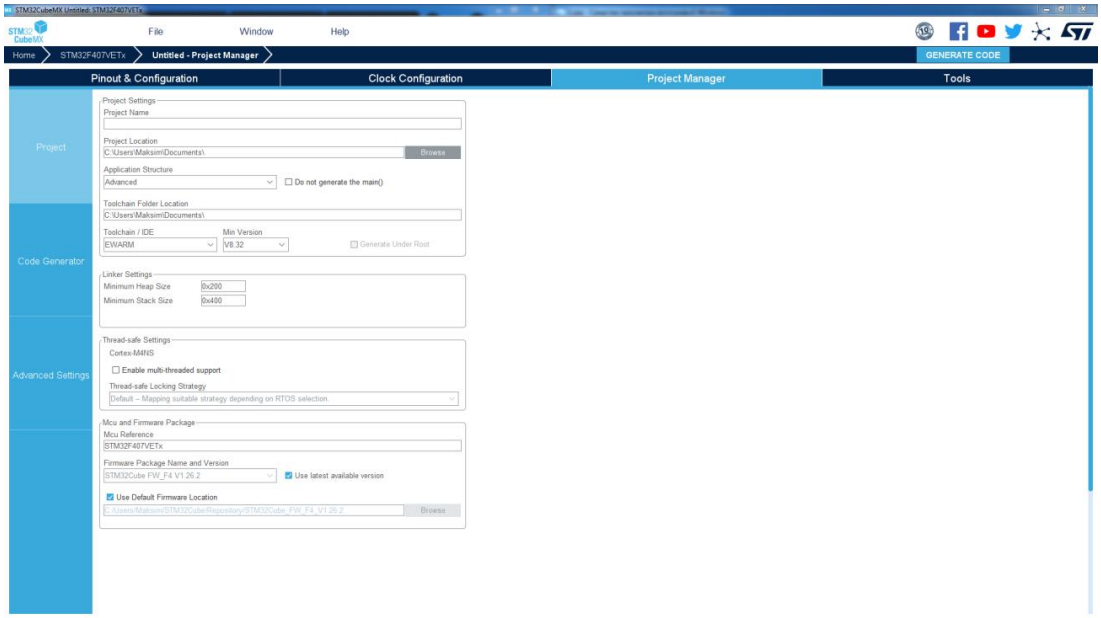

Рисунок Б.28 – Вкладка Project Manager

В поле Project Name укажем название проекта, в поле Project Location – расположение проекта на компьютере, в поле Toolchain/IDE выберем интегрированную среду разработки (в нашем случае всегда будем выбирать STM32CubeIDE).

Перейдем на вкладку Tools. Окно программы примет следующий вид (рисунок Б.29).

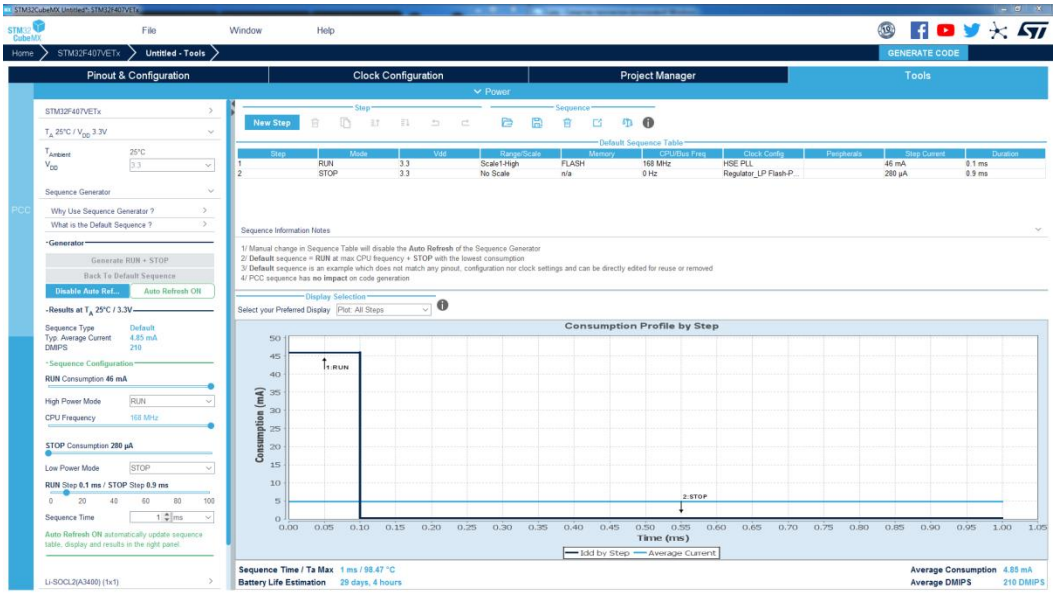

Рисунок Б.29 – Вкладка Tools

На данной вкладке в левой части окна (рисунок Б.30), если необходимо, можно произвести установку таких параметров, как температура, энергопотребление или рабочая частота для выбранного микроконтроллера.

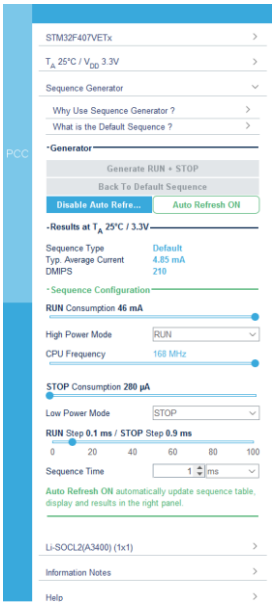

Рисунок Б.30 – Установка дополнительных параметров микроконтроллера

На данном этапе процесс конфигурирования микроконтроллера окончен.

Нажмем кнопку GENERATE CODE.

После генерирования кода в папке с проектом будут созданы файлы, отражающие конфигурацию выбранного микроконтроллера.

Для дальнейшей работы с данными файлами и прошивки микроконтроллера необходимо использовать интегрированную среду разработки. В нашем случае, как было упомянуто выше, в качестве среды разработки будем использовать STM32CubeIDE.

Выполним установку STM32CubeIDE (скачать установщик можно бесплатно с официального сайта разработчика st.com) и запустим программу. После ее открытия нас приветствует начальное окно программы, имеющее следующий вид (рисунок Б.31).

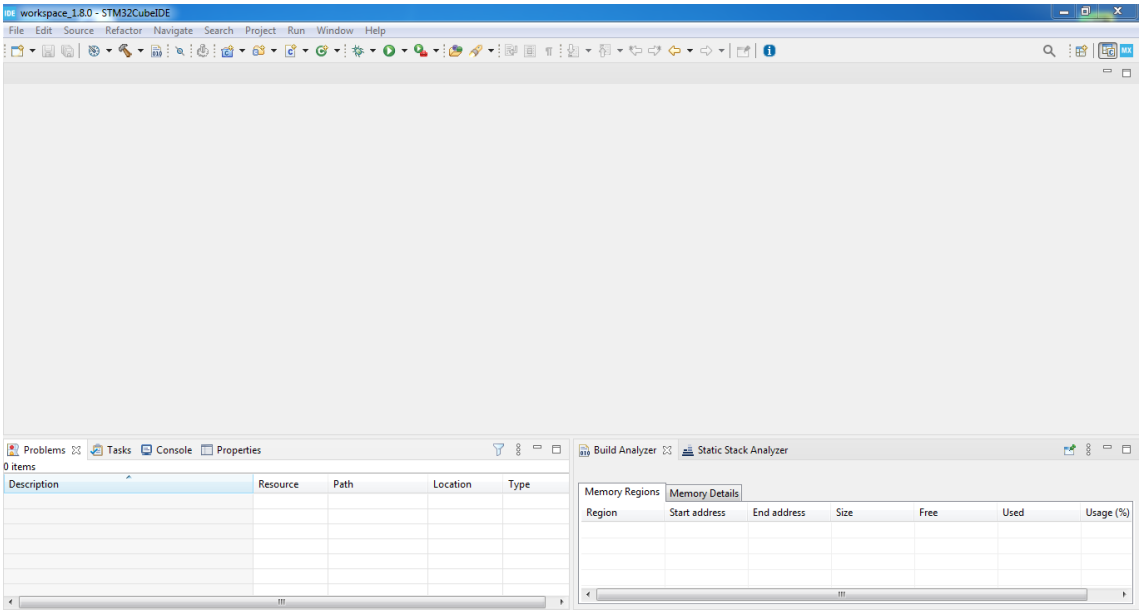

Рисунок Б.31 – Начальное окно STM32CubeIDE

Для открытия проекта, созданного в STM32CubeMX (т. е. для прошивки сконфигурированного микроконтроллера), необходимо нажать File/Open Projects from File System… .

В результате появится окно следующего вида (рисунок Б.32).

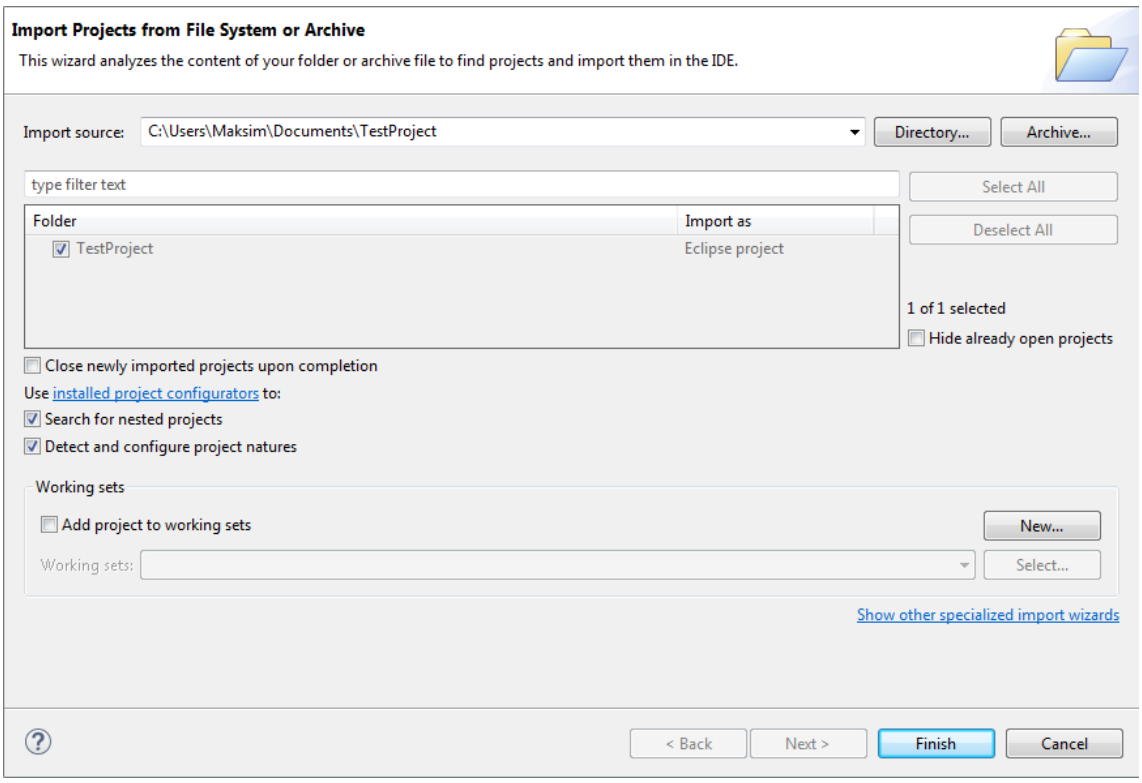

Рисунок Б.32 – Окно открытия проекта

В окне открытия проекта нажмем кнопку Directory… и выберем папку с проектом (проект, который мы создавали в STM32CubeMX). После этого нажмем Finish… . Проект будет открыт.

В правом верхнем углу необходимо нажать кнопку C/C++ (если до этого она не была нажата) (рисунок Б.33).

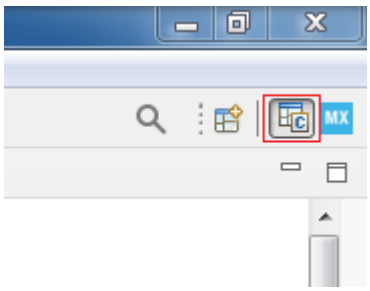

Рисунок Б.33 – Кнопка C/C++

Для открытия исполнительного файла (рисунок Б.34), непосредственно в котором будем писать код, необходимо во вкладке Projects Explorer (если данной вкладки нет, следует нажать Window/Show view/Projects Explorer) открыть файл main.c, который находится по следующему пути: название проекта/Core/Src/main.c.

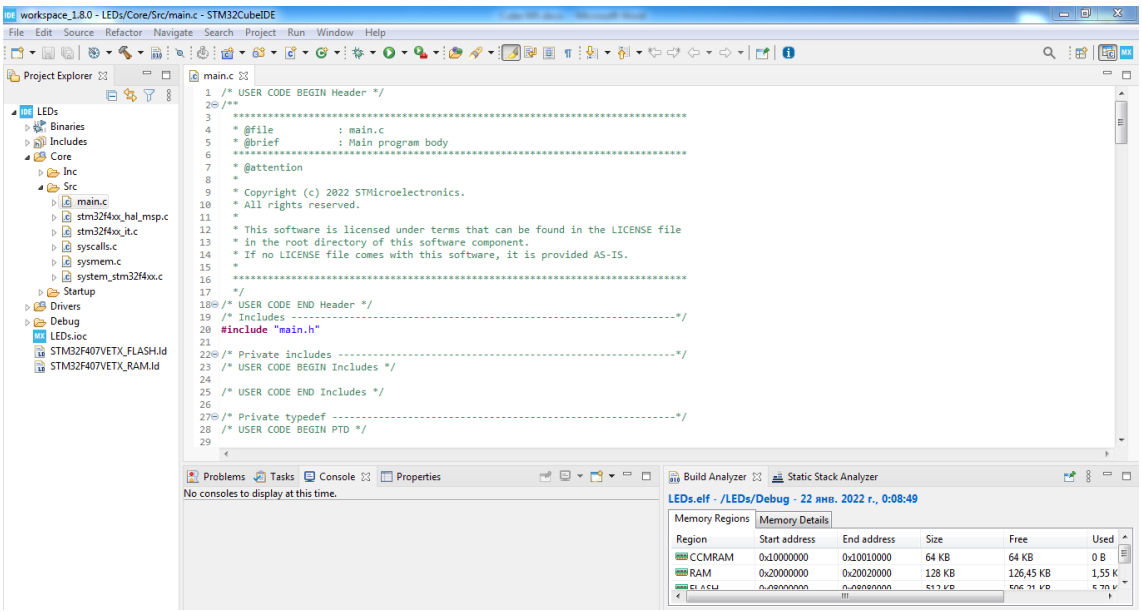

Рисунок Б.34 – Исполнительный файл

После открытия исполнительного файла в настройках проекта разрешим конвертацию проекта в файл с расширением .hex для дальнейшего использования в программе Proteus. Для этого во вкладке Projects Explorer нажмем ПКМ по названию проекта и выберем Properties (или Alt + Enter) (рисунок Б.35).

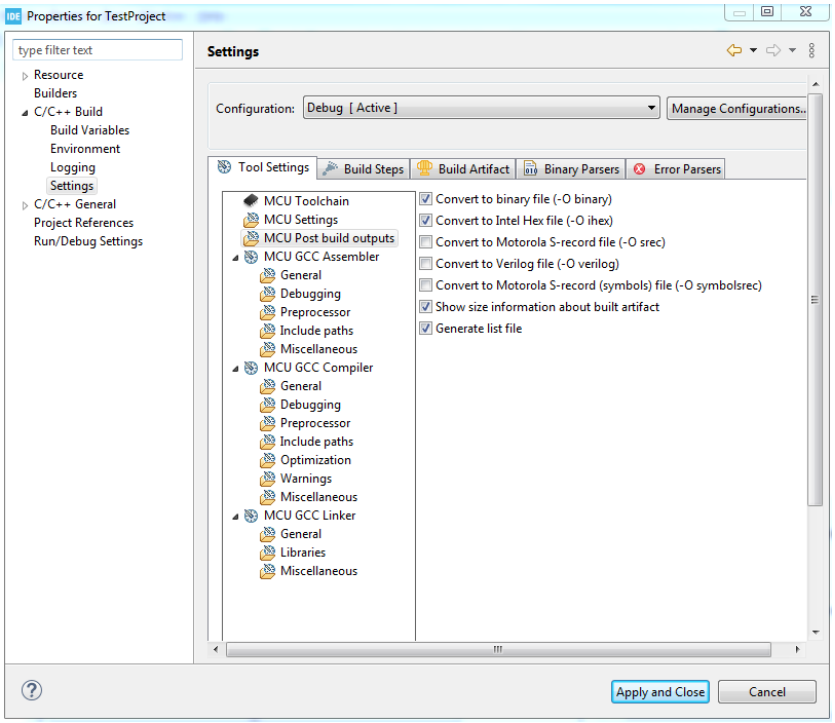

Рисунок Б.35 – Настройки проекта

В появившемся окне выберем C/C++ Build/Settings. В кладке Tool Settings/MCU Post build outputs устанавливаем Convert to Intel Hex file (-O ihex).

Далее запишем код программы в исполнительный файл (main.c). После нажмем Project/Build Project. Если ошибок нет, то в папке с проектом появится файл с расширением .hex, который будет использоваться в Proteus.

Порядок работы с программой Proteus будет рассмотрен далее.

На данном этапе был рассмотрен способ конфигурации и программирования микроконтроллера на основе двух программ: STM32CubeMX и STM32CubeIDE. В первой программе производилось конфигурирование выбранного микроконтроллера, во второй – его программирование.

Порядок работы с STM32CubeIDE

Как было описано выше, процесс конфигурирования и программирования микроконтроллера производился на основе использования двух программ: STM32CubeMX и STM32CubeIDE. Однако использование обеих программ не является обязательным, и произвести операции конфигурирования и программирования микроконтроллера можно на основе использования только одной программы – STM32CubeIDE.

STM32CubeIDE – это универсальный инструмент разработки, который является частью программной экосистемы STM32Cube. Данная IDE представляет собой передовую платформу разработки C/C++ с функциями настройки периферийных устройств, генерации, компиляции и отладки кода под микроконтроллеры и микропроцессоры производства STM32. STM32CubeIDE по сравнению с STM32CubeMX объединяет в себе функции настройки микроконтроллеров, их конфигурирования и программирования, предлагая для этого более универсальные инструменты, которые позволяют сократить время на разработку.

Интегрированная среда разработки CubeIDE имеет следующие особенности:

- интегрированное ПО STM32CubeMX;
- IDE основана на Eclipse, поддерживает плагин ECLIPSE, GNU C/C ++ в цепочке инструментов ARM и отладчик GDB;
- содержит дополнительные расширенные функции отладки, в том числе:
  а) отладку ядра ЦП и периферийных регистров;
  б) просмотр переменных в реальном времени;
  в) системный анализ и отслеживание процессов в реальном времени;

– поддерживает отладки ST-LINK и J-LINK;

– поддерживает операционные системы Windows, Linux и MacOS.

Рассмотрим порядок использования программы STM32CubeIDE на примере выполнения первой лабораторной работы.

Откроем STM32CubeIDE и создадим проект (File/New/STM32 Project) (рисунок Б.36).

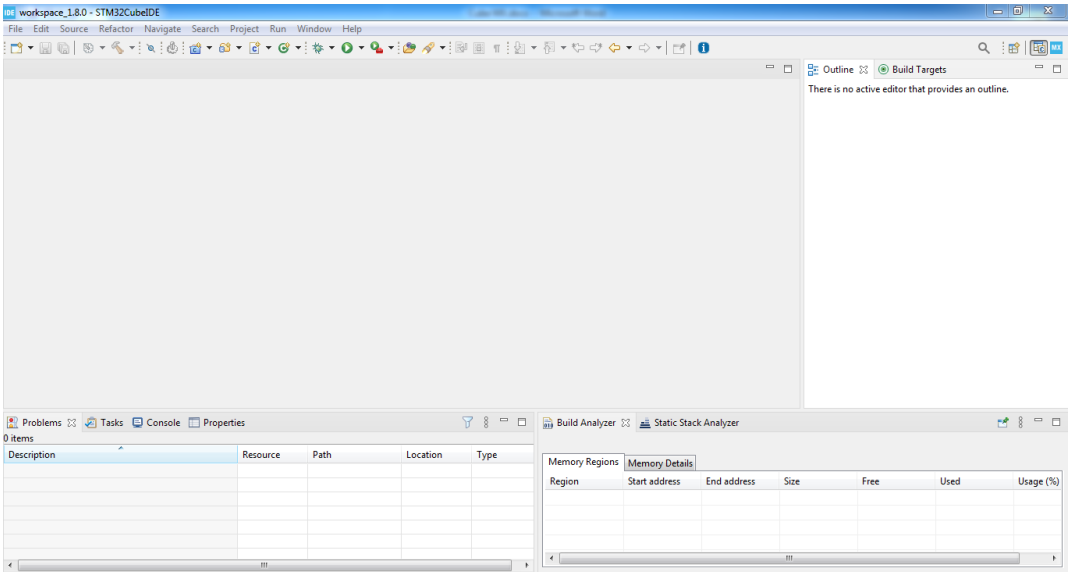

Рисунок Б.36 – Начальное окно STM32CubeIDE

В ходе создания проекта выберем плату STM32F401VETx из списка и нажмем Next. Назовем проект LEDs и нажмем Finish (рисунки Б.37–Б.39).

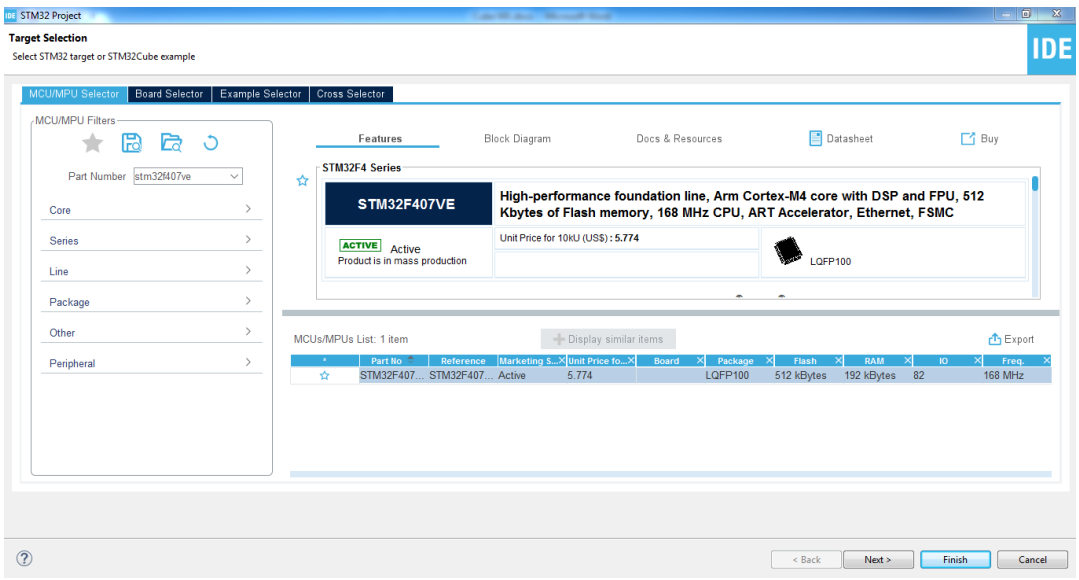

Рисунок Б.37 – Выбор микроконтроллера в STM32CubeIDE

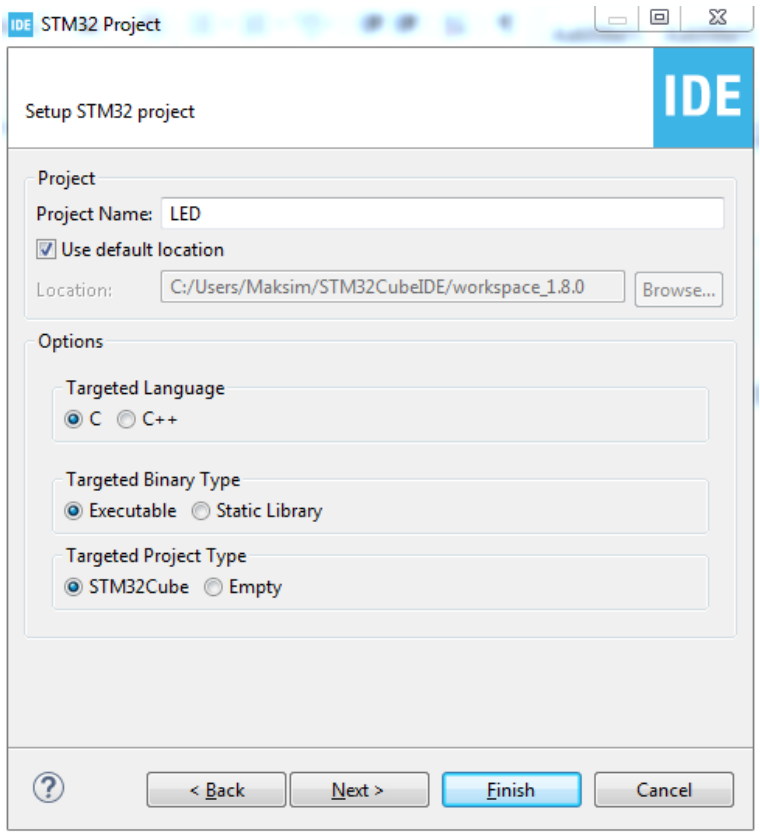

Рисунок Б.38 – Выбор параметров проекта в STM32CubeIDE

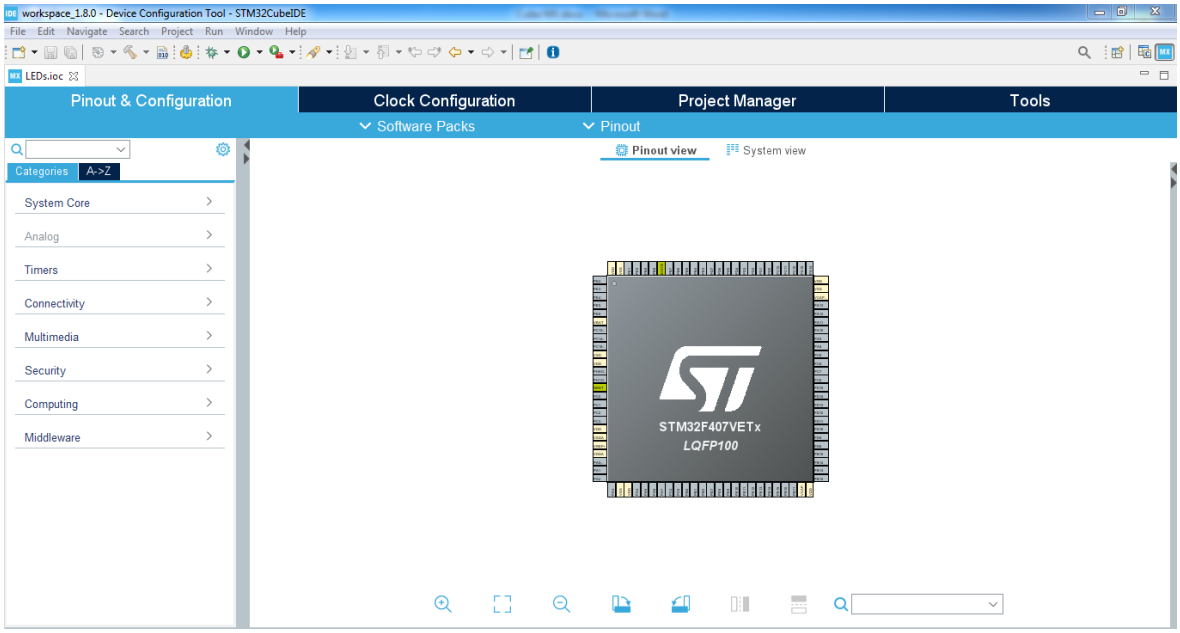

Рисунок Б.39 – Параметры микроконтроллера

После создания проекта укажем входы и выходы, которые будем использовать (четыре выхода светодиода). Также установим использование внешнего кварца (Crystal/Ceramic Resonator) и включим работу Serial Wire (рисунок Б.40).

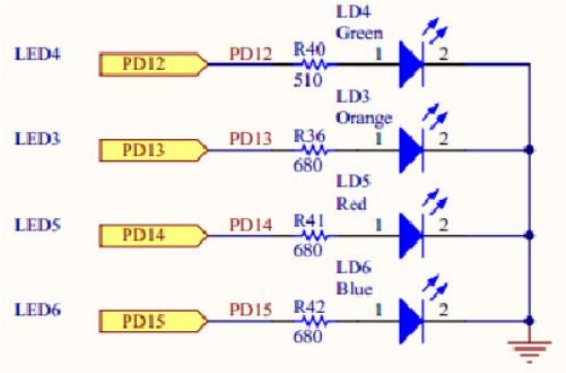

Рисунок Б.40 – Схема подключения светодиодов

Для того чтобы задать выход, нажмем на контакт PD15 и выберем GPIO_Output. Аналогичные действия выполним для контактов PD12, PD13 и PD14. Далее на вкладке A→Z нажмем RCC (рисунок Б.41).

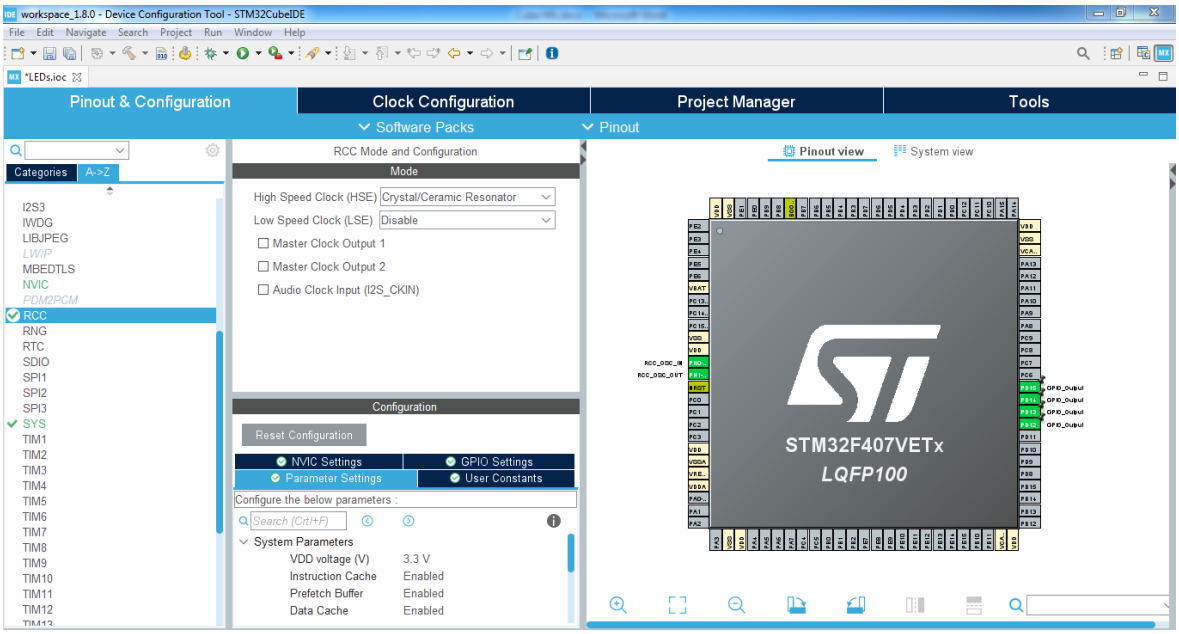

Рисунок Б.41 – Подключение выходов диодов

В появившемся окне установим High Speed Clock (HSE) на Crystal/Ceramic Resonator. На вкладке A→Z выберем SYS. В появившемся окне установим Debug на Serial Wire (рисунок Б.42).

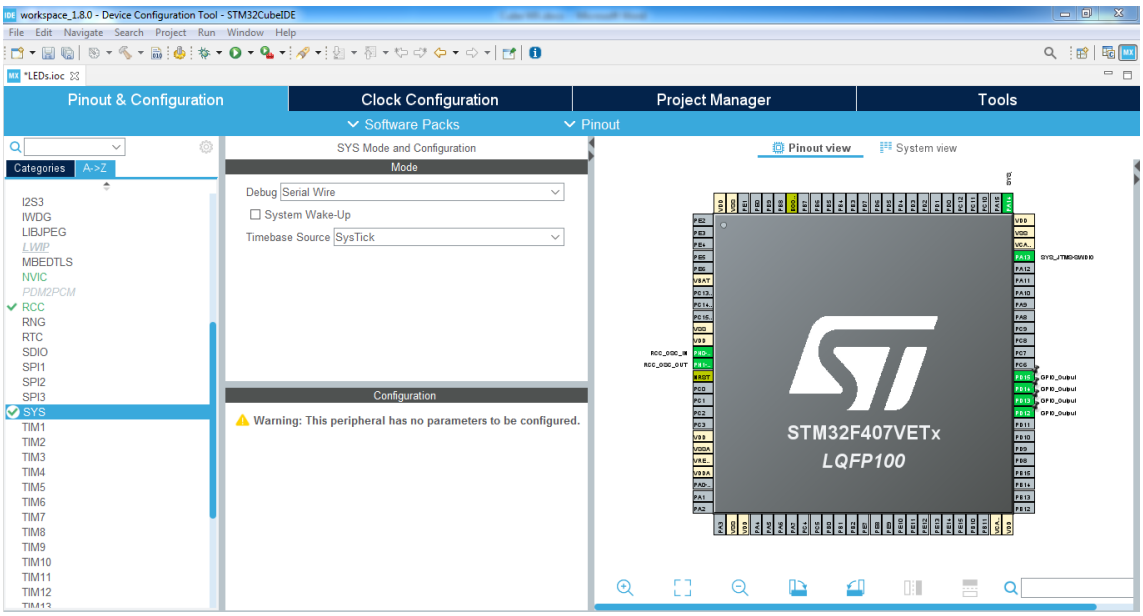

Рисунок Б.42 – Подключение кварцевого резонатора

Далее создадим исполнительный файл, в который будем писать код (Project/Generate code). В появившемся окне нажмем yes (см. рисунок Б.34).

После создания исполнительного файла в настройках проекта разрешим конвертацию проекта в файл с расширением .hex для дальнейшего использования в программе Proteus. Для это в Project Explorer нажмем ПКМ по названию проекта и выберем Properties (или Alt + Enter) (см. рисунок Б.35).

В появившемся окне выберем C/C++ Build/Settings. Во кладке Tool Settings/MCU Post build outputs установим Convert to Intel Hex file (-O ihex).

Далее запишем код программы в исполнительный файл (main.c). После нажмем Project/Build Project. Если ошибок нет, то в папке с проектом появится файл с расширением .hex, который будет использоваться в Proteus (рисунок Б.43).

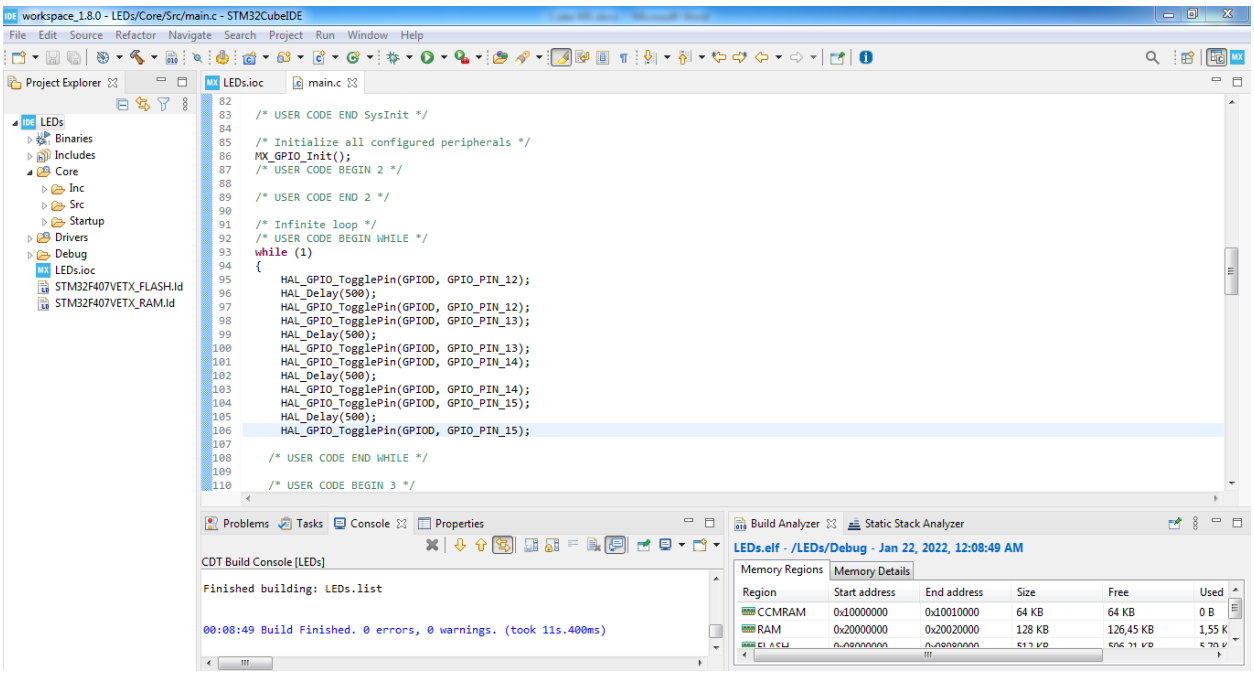

Рисунок Б.43 – Код программы

Порядок работы с Proteus 8

Proteus представляет собой пакет программного обеспечения для автоматизированного проектирования (САПР) электронных схем на основе моделей электронных компонентов.

Отличительной чертой пакета Proteus является возможность моделирования работы программируемых устройств: микроконтроллеров, микропроцессоров, DSP и т. д. Среда Proteus имеет огромную библиотеку электронных компонентов, а недостающие можно сделать самостоятельно. Каждый компонент в библиотеке содержит справочные данные. Предусмотрена поддержка SPICE-моделей, которые часто предоставляются производителями электронных компонентов.

Пакет Proteus состоит из двух подпрограмм: ISIS – программа синтеза и моделирования непосредственно электронных схем и ARES – программа разработки печатных плат. Также в состав Proteus восьмой версии входит среда разработки VSM Studio, позволяющая быстро написать программу для микроконтроллера, используемого в проекте, и скомпилировать ее. Примечательной особенностью является то, что в ARES можно увидеть 3D-модель печатной платы, что позволяет разработчику оценить свое устройство еще на стадии разработки.

Пакет программного обеспечения Proteus имеет следующие особенности:

– моделирование схем с использованием разнообразных виртуальных компонентов;

– разработка печатной платы, включая 3D-визуализацию ее сборки;
– поддержка нескольких семейств микроконтроллеров от разных производителей (Microchip, Philips, Atmel, NXP, Parallax);
– выполнение всех этапов разработки электронного устройства на основе микроконтроллера в единой среде;
– возможность написания, отладки и тестирования микропрограммного обеспечения еще до физического изготовления опытного образца системы;
– существенное преимущество в скорости разработки и проектирования.

Продемонстрируем порядок работы в программе Proteus на основе созданного проекта в STM32CubeIDE, т. е. произведем моделирование схемы (см. рисунок Б.40) на основе сконфигурированного микроконтроллера STM32f407VETx.

Откроем Proteus и создадим проект (File/New Project) (рисунки Б.44 и Б.45).

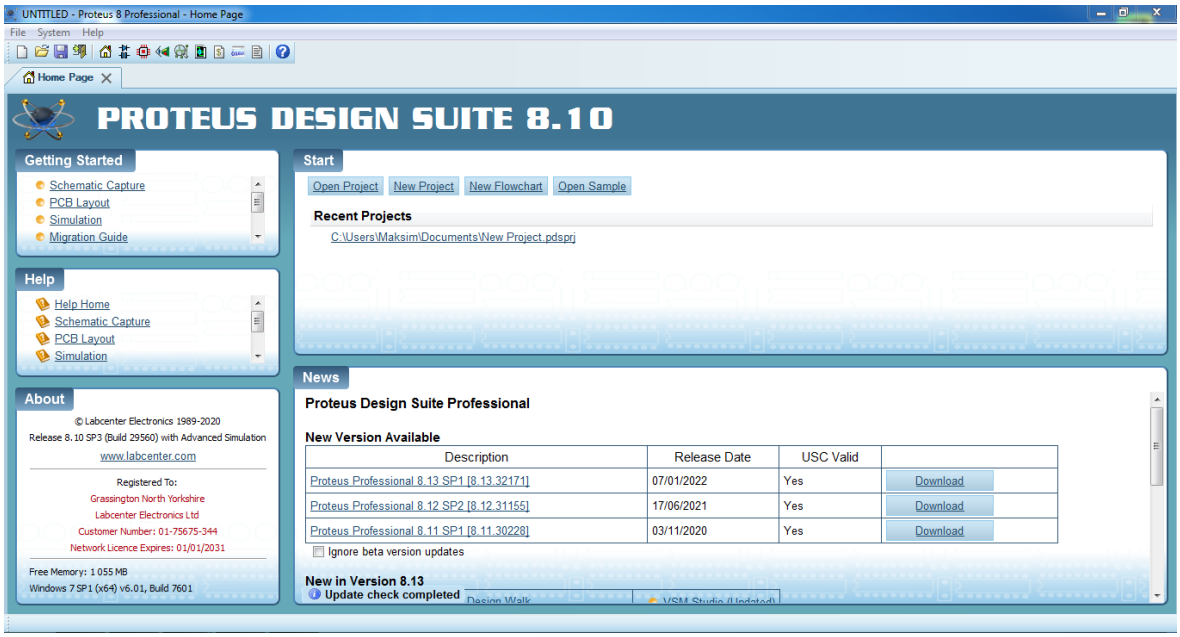

Рисунок Б.44 – Начальное окно Proteus

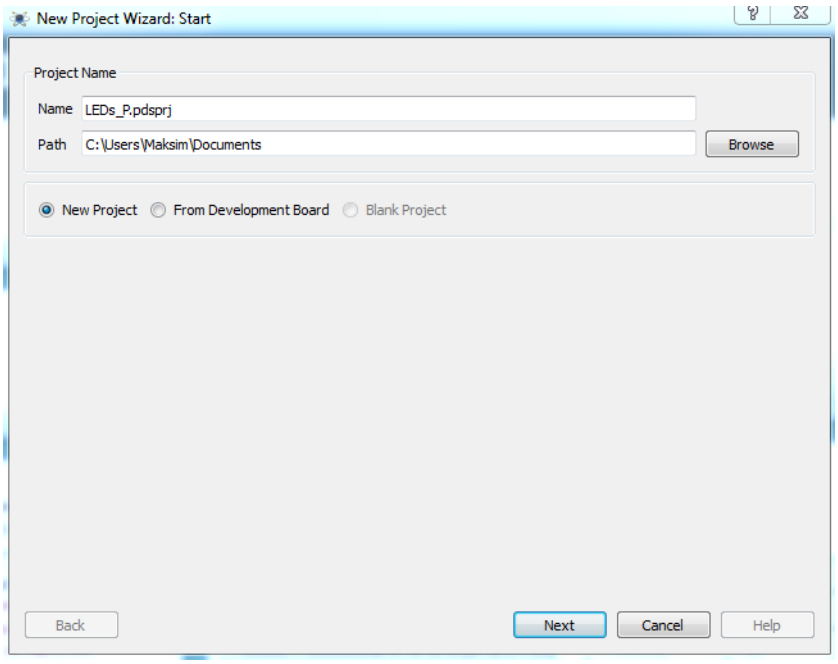

Рисунок Б.45 – Выбор параметров проекта в Proteus

В следующем окне выберем Create a schematic from the selected template и в качестве Design Templates выберем DEFAULT. Нажмем Next дважды (рисунок Б.46).

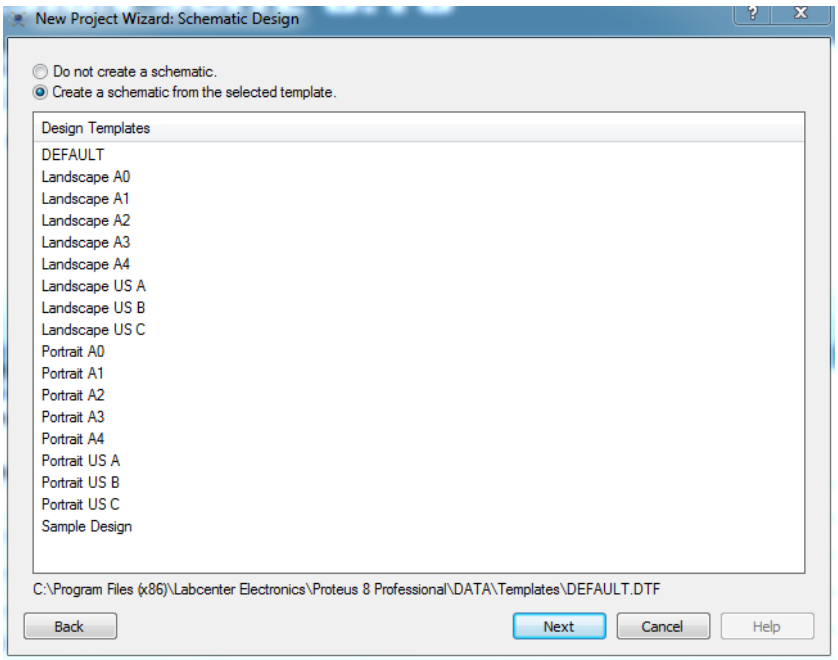

Рисунок Б.46 – Выбор шаблона проекта

В следующем окне выберем Create Firmware Project и плату (такую же плату, под которую делали проект в STM32CubeIDE). Проверим, установлены ли нужные библиотеки. Для этого нажмем Compilers… . Если необходимо,

установим недостающую библиотеку (GCC for ARM). Нажмем OK, Next и Finish. Проект будет создан (рисунки Б.47 и Б.48).

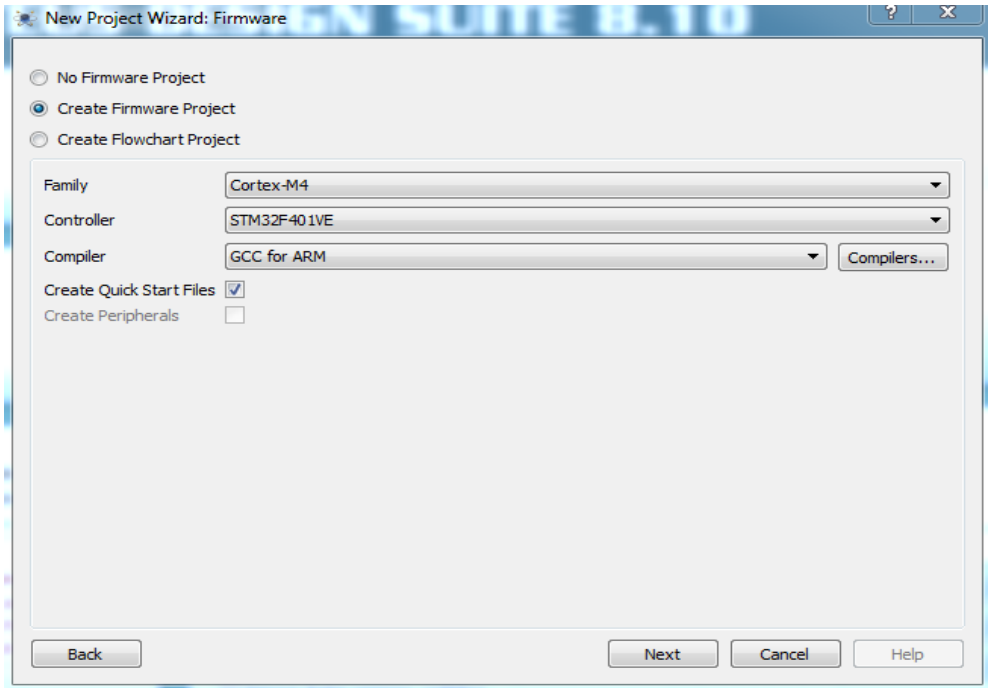

Рисунок Б.47 – Выбор микроконтроллера

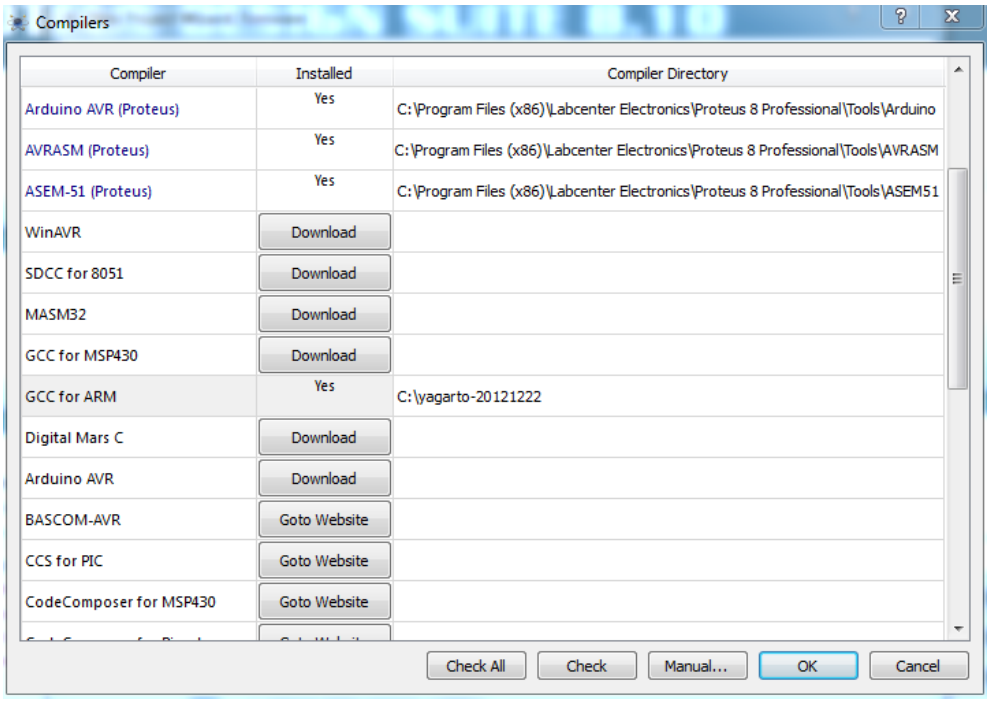

Рисунок Б.48 – Установка дополнительных библиотек

После создания проекта добавим к контроллеру необходимые элементы (питание, землю, диоды, резисторы и т. д.) и соединим их (рисунок Б.49).

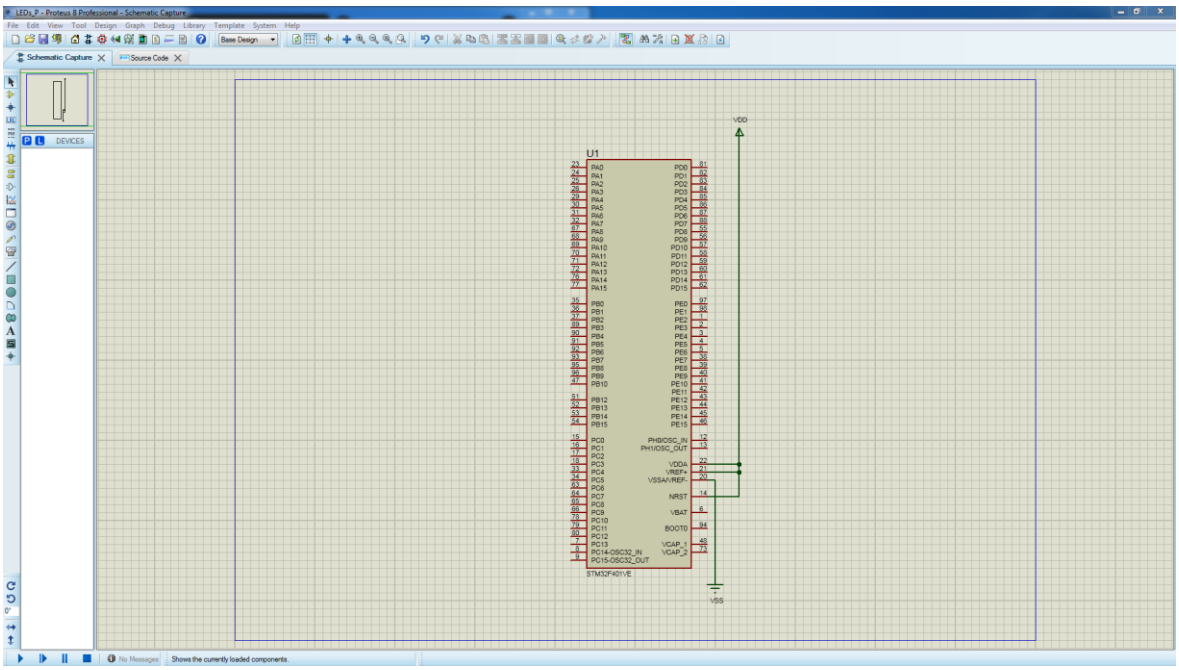

Рисунок Б.49 – Исходная схема

Для добавления элементов нажмем Library/Pick Parts (рисунок Б.50).

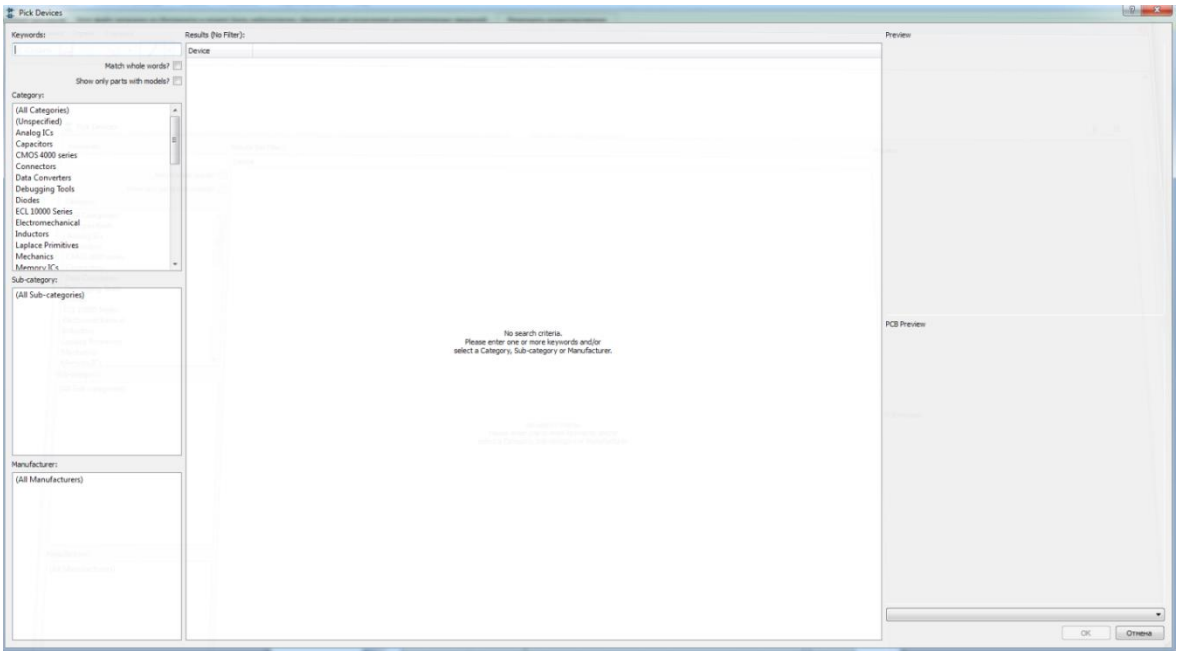

Рисунок Б.50 – Библиотека элементов

Найдем нужный элемент, выберем его и нажмем OK. Элемент добавится слева в список используемых элементов, где его можно будет выбрать и добавить к схеме (рисунок Б.51).

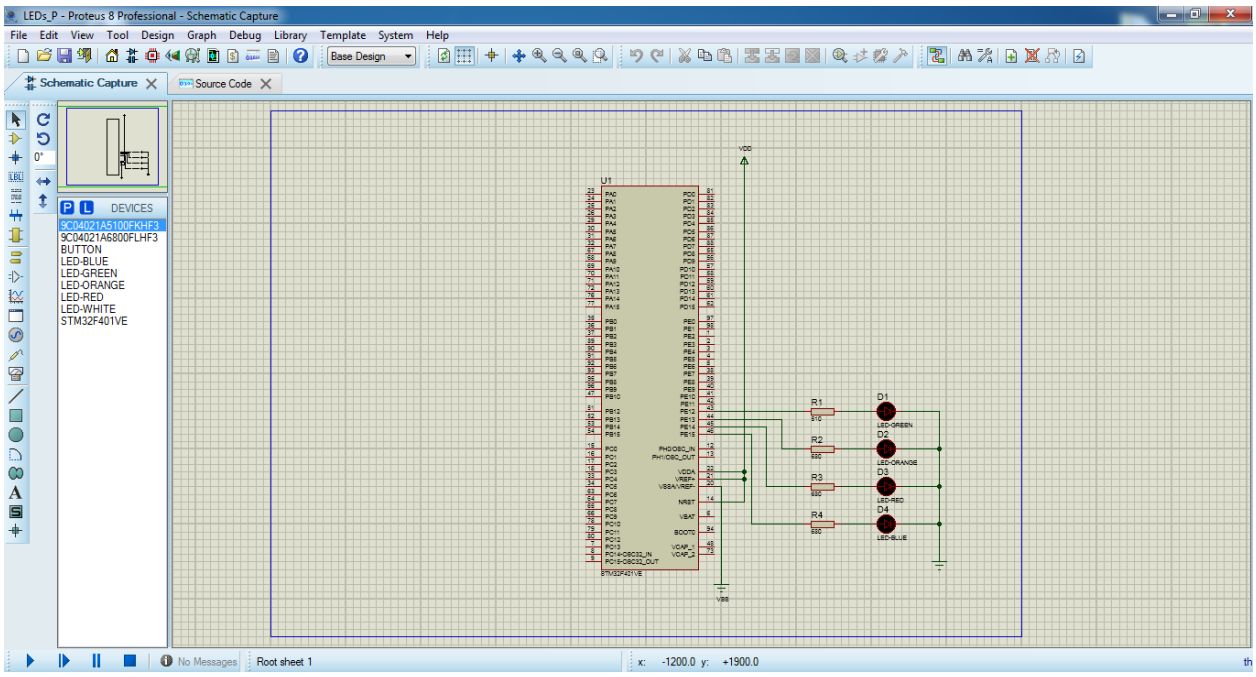

Рисунок Б.51 – Схема после добавления необходимых элементов

После того как собрали схему, зададим программный файл контроллеру (файл с расширением .hex из проекта в программе STM32CubeIDE). Для этого дважды нажмем на контроллер и укажем путь к файлу (рисунок Б.52).

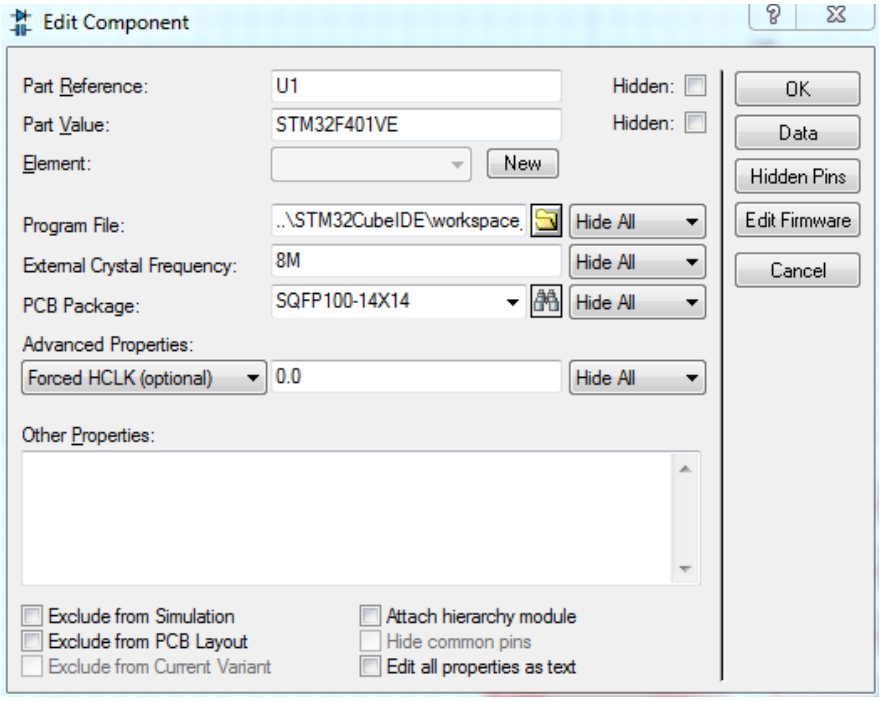

Рисунок Б.52 – Параметры контроллера

Далее запустим симуляцию и проверим работу контроллера. Для запуска симуляции необходимо нажать Debug/Run Simulation (или клавиша F12) (рисунки Б.53–Б.56).

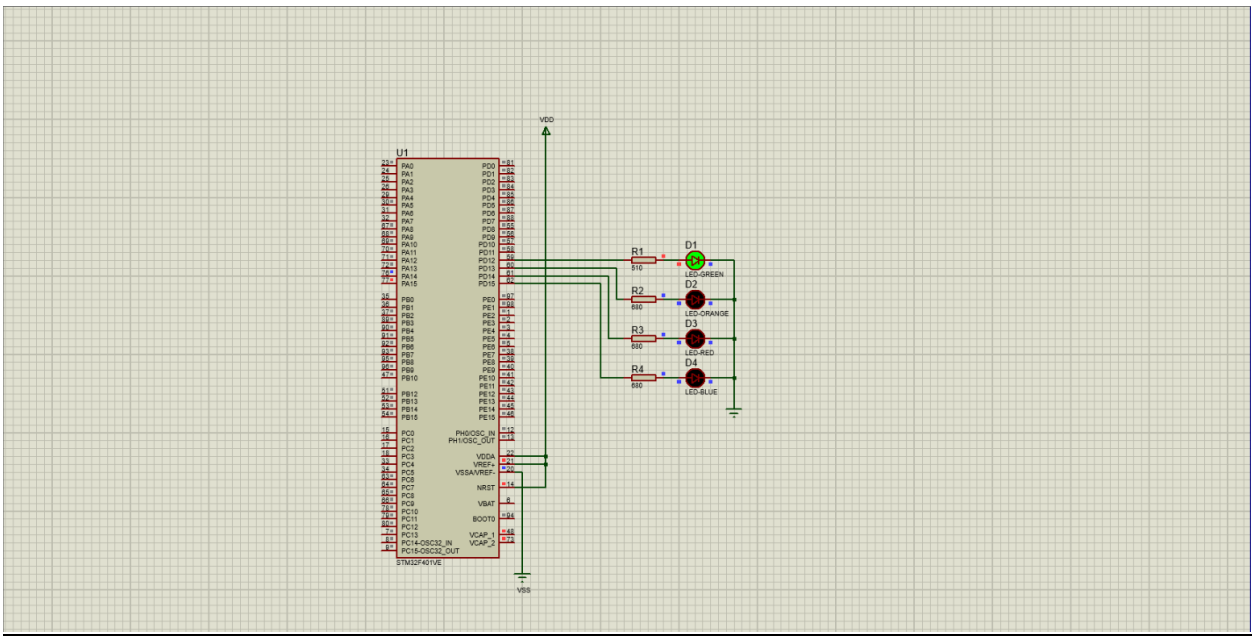

Рисунок Б.53 – Пример работы контроллера (1)

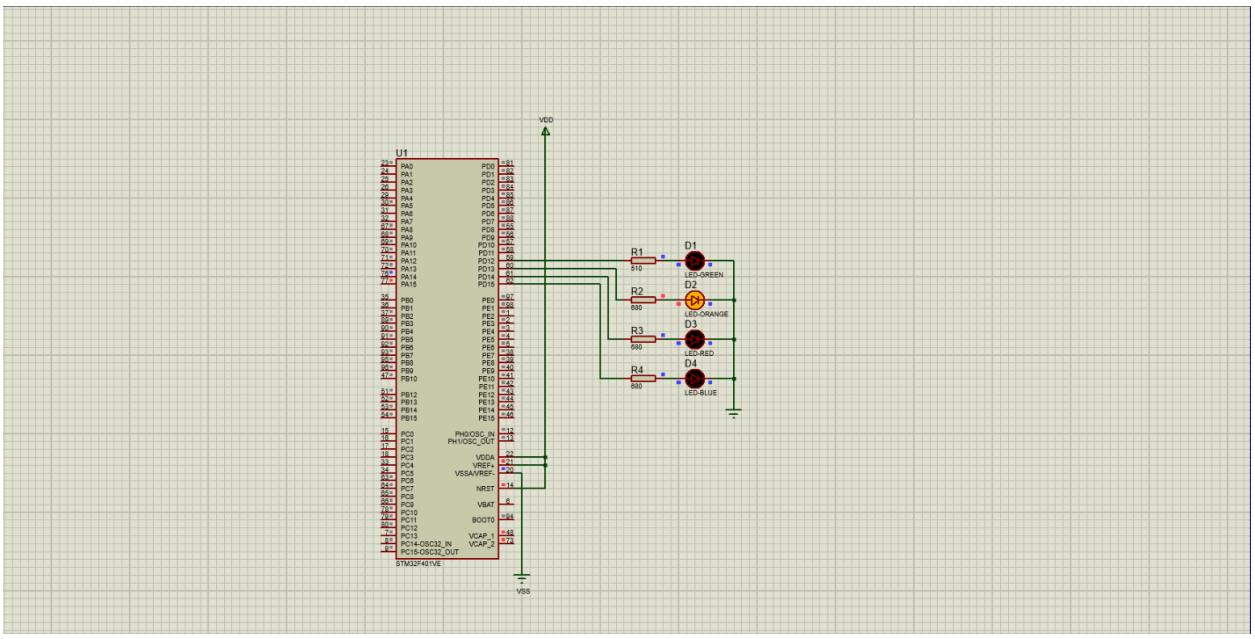

Рисунок Б.54 – Пример работы контроллера (2)

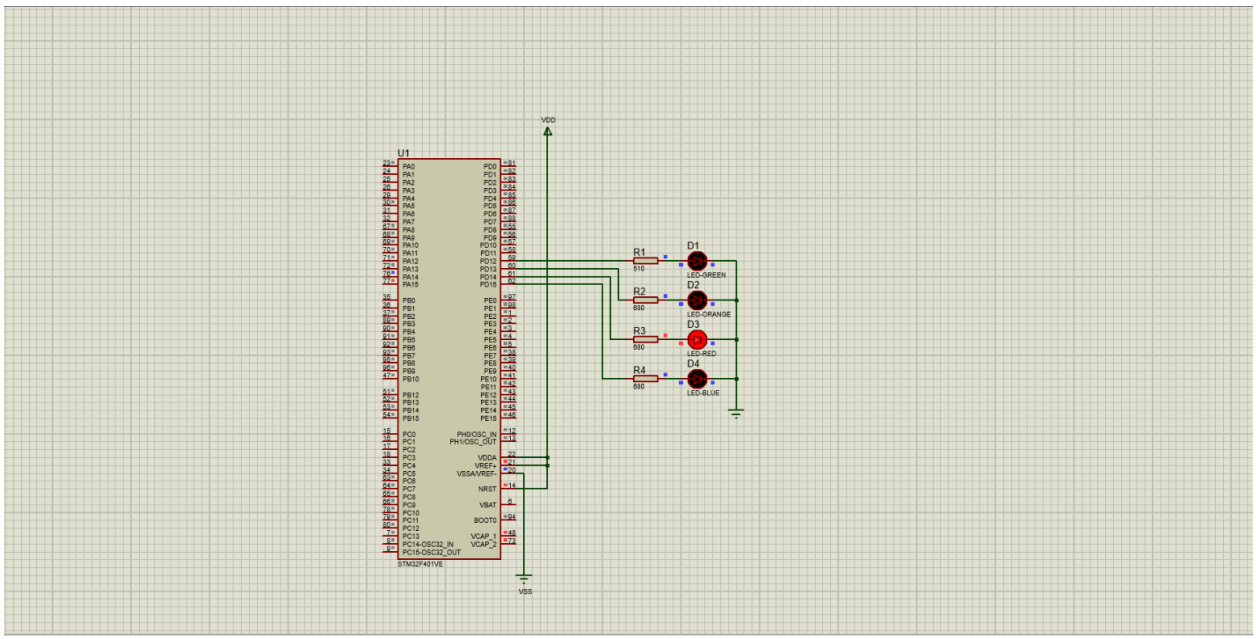

Рисунок Б.55 – Пример работы контроллера (3)

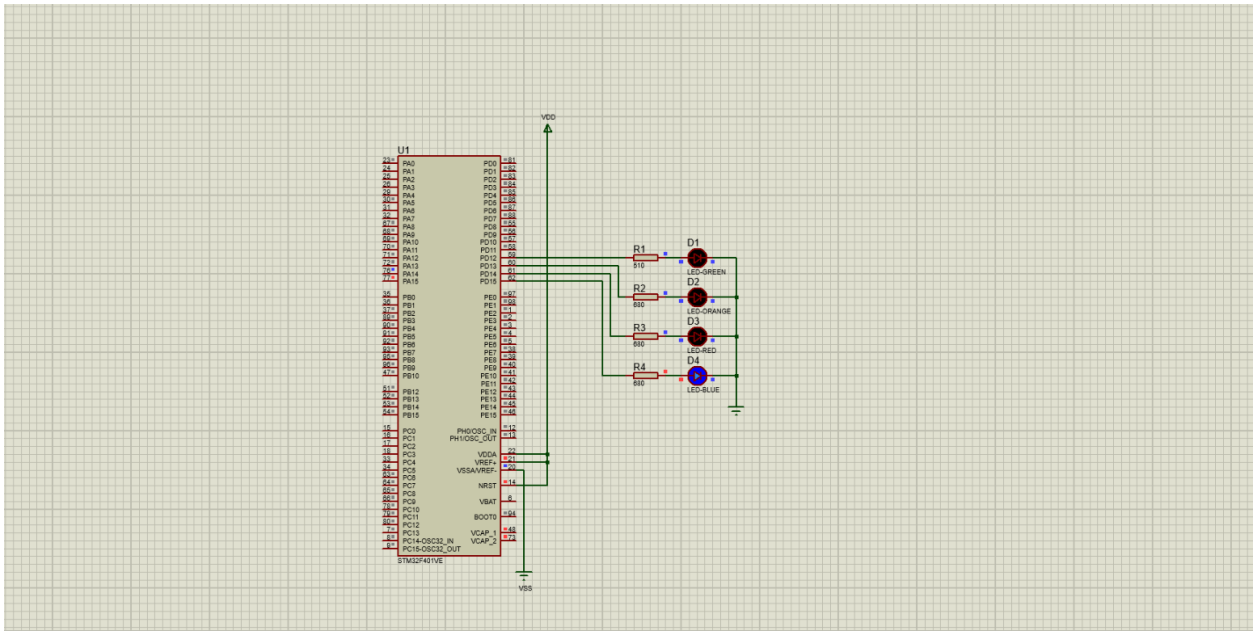

Рисунок Б.56 – Пример работы контроллера (4)

Таким образом, программное обеспечение Proteus позволило выполнить моделирование работы сконфигурированного микроконтроллера на основе созданной прошивки.

ПРИЛОЖЕНИЕ В

РАБОТА С ДИСПЛЕЕМ

Вместе с непосредственной обработкой сигналов важной частью работы схемы является отображение информации, результатов обработки для пользователя системы. В том случае, когда схема достаточно проста и направлена на отслеживание небольшого количесвта показателей, достаточно, например, использовать семисегментный индикатор или символьный LCD-дисплей. Такие дисплеи зачастую могут отображать больше информации, чем индикаторы, составленные из нескольких семисегментных, а также способны отображать намного больше различных символов.

Для ознакомления с работой таких дисплеев рассмотрим дисплей RC1602A (рисунок В.1), а также его использование под управлением STM32F4 Discovery. Данный дисплей отображает две строки по 16 символов в каждой и работает под управлением контроллера KS0066U. Он обеспечивает следующую функциональность по работе с дисплеем:

- выполнение команд управления (управление разрядностью, курсором и др.);
- выполнение записи и считывание памяти дисплея.

Рисунок В.1 – Внешний вид дисплея RC1602A

Использованная схема подключения дисплея представлена на рисунке В.2.

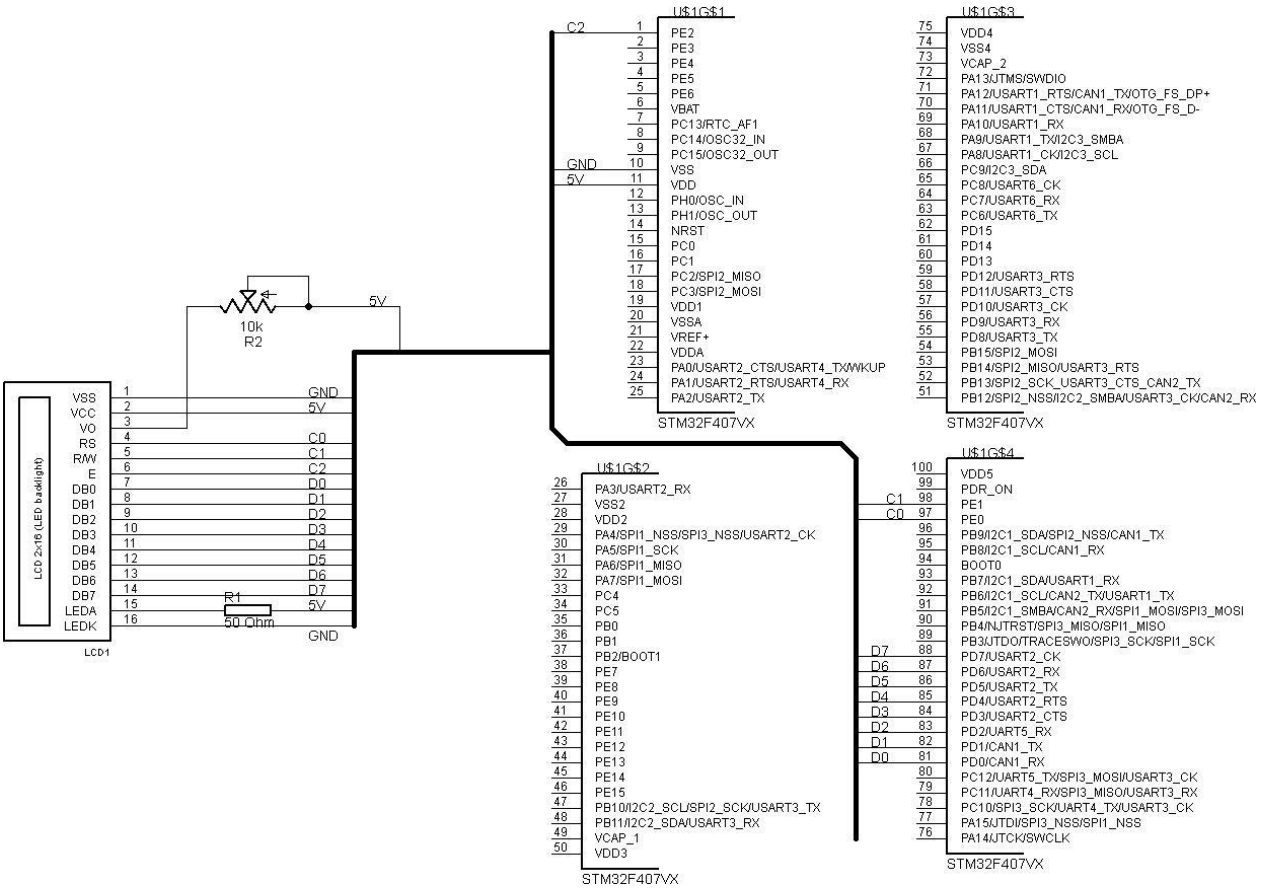

Рисунок В.2 – Схема подключения дисплея

Для управления дисплеем с помощью платы STM32F4 Discovery написана библиотека (реализована не вся функциональность, а лишь наиболее необходимое: выполнение команд и запись данных в память для отображения). Сама память разделена на две части: память отображения (DDRAM) и память, отведенная под символы пользователя (CGRAM). Информация, записанная в память для отображения, появляется на экране. Поскольку размер памяти больше размера экрана, то отображается лишь ее часть, а отображение остальной части можно обеспечить с помощью сдвига отображаемой на экране памяти. Более детальное описание возможностей дисплеев такого типа можно найти в документации к ним в Интернете.

Далее приведен код библиотеки: заголовочный файл и файл с реализацией.

Ниже представлен заголовочный файл библиотеки.

Листинг кода:

```
#ifndef KS006ULIB_H
#define KS006ULIB_H
```

```
#include <stm32f4xx_gpio.h>
#define DATA_PORT GPIOD
#define CONTROL_PORT GPIOE
#define DATA_PINS GPIO_Pin_0 | \
      GPIO_Pin_1 | \
      GPIO_Pin_2 | \
      GPIO_Pin_3 | \
      GPIO_Pin_4 | \
      GPIO_Pin_5 | \
      GPIO_Pin_6 | \
      GPIO_Pin_7
#define CONTROL_PINS GPIO_Pin_0 | \
      GPIO_Pin_1 | \
      GPIO_Pin_2
#define RS_PIN GPIO_Pin_0
#define RW_PIN GPIO_Pin_1
#define E_PIN GPIO_Pin_2

void lcd_init();
void lcd_command(unsigned char command_data);
void write_data(unsigned char data);
void write_char(unsigned char data);
void write_string(char * string);
void write_string_at(char * string, unsigned char address);
void write_at(unsigned char symbol, unsigned char address);
void set_rs();
void unset_rs();
void set_rw();
void unset_rw();
void set_e();
void unset_e();

#define lcd_clear() lcd_command(0x01)
#define lcd_set_address(address) lcd_command(0b10000000 & address)

#endif
```

Далее представлен файл реализации библиотеки.

Листинг кода:

```
#include "ks0066ulib.h"

extern void delay(int times);

void lcd_init() {
      delay(20);
      lcd_command(0b00110000);
      lcd_command(0b00110000);
```

```
	lcd_command(0b00110000);
	lcd_command(0b00111000);
	lcd_command(0b00001111);
	lcd_command(0b00000001);
	lcd_command(0b00000110);
}

void lcd_command(unsigned char command_data) {
	unset_rs(); unset_rw();
	write_data(command_data);
	set_e();
	delay(40);
	unset_e();
}

void write_char(unsigned char data) {
	set_rs(); unset_rw();
	write_data(data);
	set_e();
	delay(40);
	unset_e();
}

void write_string(char * string) {
	uint32_t i;
	for(i = 0; string[i] != '\0'; ++i)
		write_char(string[i]);
}

void write_string_at(char * string, unsigned char address) {
	lcd_set_address(address);
	write_string(string);
}

void write_data(unsigned char data) {
	GPIO_Write(DATA_PORT, data);
}

void write_at(unsigned char symbol, unsigned char address) {
	lcd_set_address(address);
	write_char(symbol);
}

void set_rs() {
	GPIO_SetBits(CONTROL_PORT, RS_PIN);
}
```

```
void unset_rs(){
     GPIO_ResetBits(CONTROL_PORT, RS_PIN);
}

void set_rw(){
     GPIO_SetBits(CONTROL_PORT, RW_PIN);
}

void unset_rw(){
     GPIO_ResetBits(CONTROL_PORT, RW_PIN);
}

void set_e(){
     GPIO_SetBits(CONTROL_PORT, E_PIN);
}

void unset_e(){
     GPIO_ResetBits(CONTROL_PORT, E_PIN);
}
```

Остановимся на особенностях использования данной библиотеки. Предполагается, что она будет использоваться как часть проекта в среде разработки. Соответственно следует настроить для проекта пути поиска заголовочных файлов, чтобы они включали путь к stm32f4xx_gpio.h, или отредактировать файл соответствующим образом. Кроме того, необходимо задать значения CONTROL_PORT и DATA_PORT и номера контактов для вашего варианта настройки. Также вам необходимо определить функцию delay(int), которая используется для генерации задержек между сигналами от платы к дисплею. При тестировании библиотеки использовался вариант функции, которая в цикле декрементирует значение переменной. Кроме того, необходимо самостоятельно настроить соответствующие порты для вывода перед использованием, после чего вызвать функцию lcd_init().

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бугаев, В. И. Лабораторный практикум для изучения микроконтроллеров архитектуры ARM Cortex M4 на базе отладочного модуля STM32F4 Discovery / В. И. Бугаев, М. П. Мусиенко, Я. М. Крайнык. – Москва – Николаев : МФТИ – ЧГУ, 2013. – 71 с.

2. STM32F4DISCOVERY. Отладочный комплект на базе STM32F407VGT6 ARM CortexM4-F [Электронный ресурс]. – 2022. – Режим доступа : http://www.chipdip.ru/product/stm32f4discovery.

3. GNU Tools for ARM Embedded Processors [Электронный ресурс]. – 2022. – Режим доступа : https://launchpad.net/gcc-arm-embedded/+download.

4. Бородулин, А. STM8 и STM32 – объединенное пространство 8- и 32-разрядных микроконтроллеров / А. Бородулин // Компоненты и технологии №10. – 2009. – с. 55–59.

5. STM32F4 GPIO tutorial [Электронный ресурс]. – 2022. – Режим доступа : http://eliaselectronics.com/stm32f4-tutorials/stm32f4-gpio-tutorial.

6. STM32F4: PWM [Электронный ресурс]. – 2022. – Режим доступа : http://amarkham.com/?p=37.

7. STM32F4: INTERRUPT TIMER [Электронный ресурс]. – 2022. – Режим доступа : http://amarkham.com/p=29.

8. Программирование STM32F4. USART. Пример программы. [Электронный ресурс]. – 2022. – Режим доступа : http://microtechnics.ru/ programmirovanie-stm32f4-usart-primer-programmy.

9. Микроконтроллеры AVR. UART. Использование прерываний. [Электронный ресурс]. – 2022. – Режим доступа : http://microtechnics.ru/ mikrokontrollery-avr-uart-ispolzovanie-preryvanij.

10. STM32 ADC Примеры использования. Шаг 1 [Электронный ресурс]. – 2022. – Режим доступа : http://mycontroller.ru/stm32-adc-primeryi-ispolzovaniya-shag-1.

11. Подключаем HD44780 дисплей к STM32 [Электронный ресурс]. – 2022. – Режим доступа : http://easystm32.ru/indication/22-hd44780-and-stm32.

12. MicroXplorer Eclipse plugin, graphical tool to configure STM32 microcontrollers. [Электронный ресурс]. – 2022. – Режим доступа : http://www.st.com/web/en/catalog/tools/PF257931.

13. Clock configuration tool for STM32F40x/41x microcontrollers [Электронный ресурс]. – 2022. – Режим доступа : http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF257927.

14. Торгаев, С. Н. Практическое руководство по программированию STM-микроконтроллеров : учеб. пособие / С. Н. Торгаев, М. В. Тригуб, И. С. Мусоров. – Томск : Изд-во ТПУ, 2015. – 111 с.

15. Reference manual. STM32F40xxx, STM32F41xxx, STM32F42xxx, STM32F43xxx advanced ARM-based 32-bit MCUs.

16. Espressif Systems ESP32-DevKitM-1. Программирование ESP IDF ESP32-DevKitM-1. Руководство пользователя по программированию ESP32 IDF.

17. Руководство по установке и лицензированию ПО Altera. Микросхемы ПЛИС серии MAX, семейство MAX V, семейство MAX II, семейство MAX3000A, семейства MAX7000B.

Св. план 2023, поз. 7

Учебное издание

Дворникова Татьяна Николаевна

ВСТРАИВАЕМЫЕ СИСТЕМЫ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *Е. С. Юрец*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *А. А. Лущикова*

Подписано в печать 22.08.2023. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 9,42. Уч.-изд. л. 10,1. Тираж 50 экз. Заказ 239.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
Ул. П. Бровки, 6, 220013, г. Минск