Тульский государственный университет
Институт высокоточных систем им. В.П.Грязева
Кафедра "Приборы управления"

АО «ПКК Миландр»

Р.В. Алалуев
В.М. Глаголев
А.Я. Матвеев
Л.Л. Владимиров

Основы программирования 32-разрядных микроконтроллеров 1986ВЕ91Т компании «Миландр»

Руководство для выполнения лабораторных работ в среде Keil uVision

Москва
2015

Алалуев Р. В. **Основы программирования 32-разрядных микроконтроллеров 1986ВЕ91Т компании «Миландр»**: руководство к выполнению лабораторных работ / Р. В. Алалуев, В.М. Глаголев, А.А. Мосур, Л. Л. Владимиров. – М., 2017. – 128 с.: ил.

Пособие содержит руководство к выполнению лабораторных работ по программированию микроконтроллеров на основе отладочной платы для 32-разрядного микроконтроллера 1986ВЕ91Т, разработанного и производимого компанией АО «ПКК Миландр» (г. Москва, Зеленоград).

Рассмотрены следующие темы: установка и настройка среды Keil uVision; работа портов ввода-вывода; работа таймера; цифро-аналоговый преобразователь; аналого-цифровой преобразователь; схемы тактирования и изменение тактовой частоты; модуль UART; модуль CAN; модуль USB и др.

Пособие может быть использовано для изучения архитектуры и методов программирования 32-разрядных микроконтроллеров. Предназначено для студентов бакалавриата и магистратуры, изучающих программирование и применение микроконтроллеров.

Содержание

Лабораторная работа № 1. Установка и настройка Keil uVision. Подготовка первого проекта

Цель работы:

Подготовка рабочей среды для выполнения лабораторных работ по данному курсу. Знакомство со средой программирования Keil uVision. Создание простейшего пустого проекта и конфигурирование среды разработки.

Приборы и материалы:

1. ПК, совместимый со средой программирования Keil uVision и имеющий USB порт.
2. Папка с необходимыми файлами (рис. 1.1).

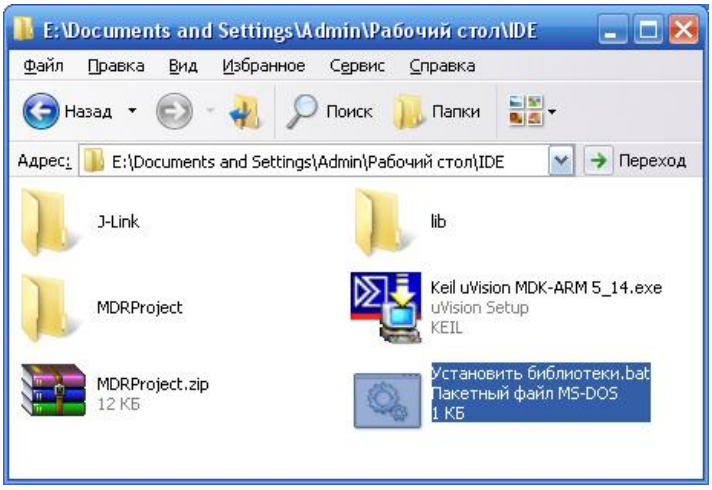


Рисунок 1.1 – Материалы лабораторного комплекса

Порядок работы:

Описание Keil uVision.

Keil uVision представляет собой IDE (Integrated Development Environment) – интегрированную среду разработки, включающую набор утилит для выполнения полного комплекса мероприятий по написанию программного обеспечения микроконтроллеров.

Среди основных программных средств Keil uVision можно отметить:

1) Базу данных микроконтроллеров, содержащую подробную информацию обо всех поддерживаемых устройствах;
2) Менеджер проектов, служащий для объединения отдельных текстов программных модулей и файлов группы, обрабатываемые по единым правилам;
3) Встроенный редактор кода;
4) Средства автоматической компиляции, ассемблирования и компоновки проекта, предназначенные для создания исполняемого модуля программы;
5) Отладчик-симулятор, отлаживающий работу скомпилированный программы на

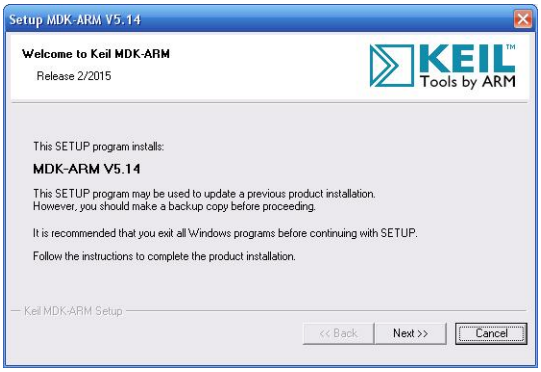
виртуальной модели микропроцессора.
6) Дополнительные утилиты
Для загрузки программ, разработанных и скомпилированных в **Keil uVision** применяется внутрисхемный J-Tag программатор-отладчик J-Link.

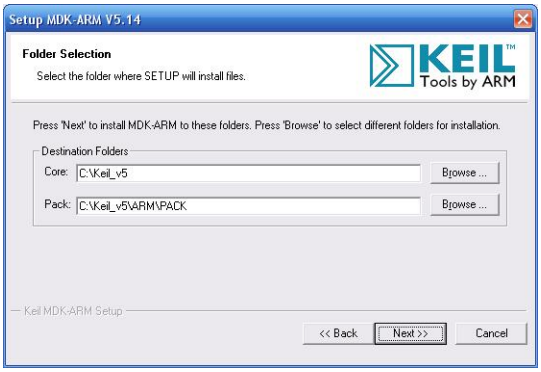
Установка Keil uVision.

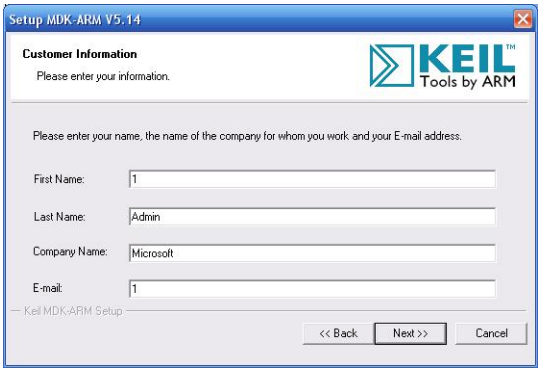
1. Запустить установку среды (файл **Keil uVision MDK-ARM 5_14.exe** в материалах, данных преподавателем) – рисунок 1.2.

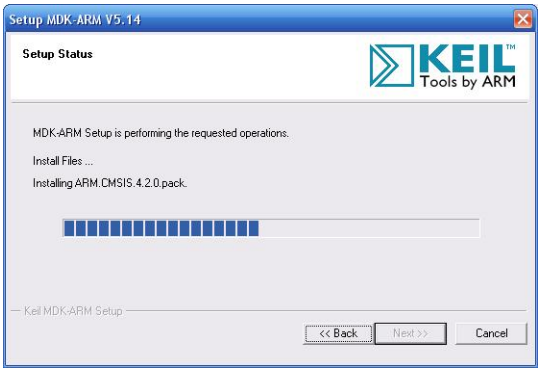
Рисунок 1.2 – Запуск установки Keil uVision.

2. Пройти стандартную процедуру установки программного обеспечения, согласившись с условиями лицензионного соглашения и проверив, что программа **устанавливается на диск C** – рисунок 1.2. Это необходимо для верной работы исполняемого файла «Установить библиотеки.bat».









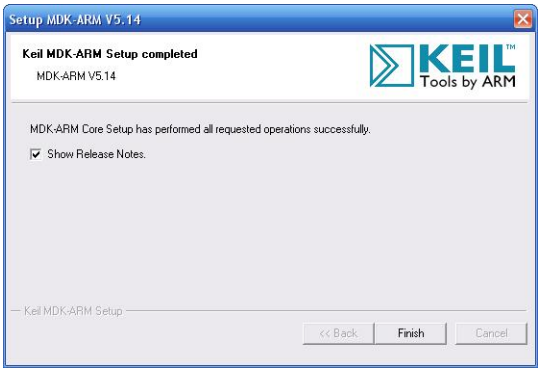


Рисунок 1.3 – Группа скриншотов процесса установки Keil uVision

3. Установить библиотеки, необходимые для того чтобы IDE смогла работать с МК компании Миландр. Для этого запустить исполняемый файл «Установить библиотеки.bat» – Рис 1.4.

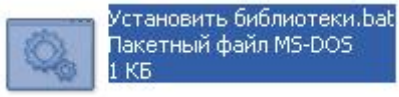


Рисунок 1.4 – Запуск установки библиотек.

Аналогично можно скопировать файл MDR32F9x.FLM из папки lib в папку с установленной средой (по умолчанию C:\Keil_v5\ARM\Flash), а после этого запустить файл Milandr.MDR1986BExx.1.3.0.pack из папки lib. Процесс установки показан на рис 1.5.

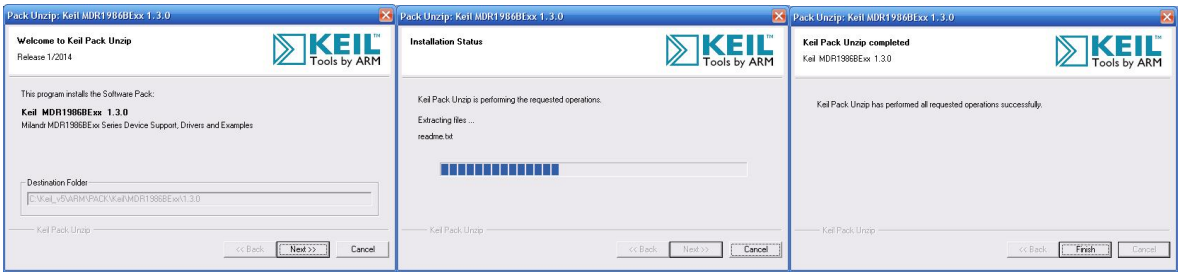


Рисунок 1.5 – Группа скриншотов процесса установки библиотек.

4. Запустить заранее созданный проект, чтобы убедиться в правильности установки среды разработки. Для этого:

4.1 Разархивировать «**MDRProject.zip**», открыть появившуюся папку «MDRProject» и запустить файл «MDRProject.uvprojx» – Рис 1.6.

Рисунок 1.6 – Запуск первого проекта.

4.2 В появившемся окне Keil uVision нажать на кнопку «**Build**» или воспользоваться горячей клавишей «F7» – Рисунок 1.7.

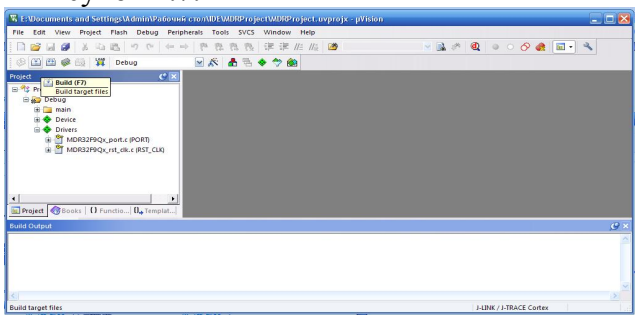


Рисунок 1.7 – Построение первого проекта.

4.3 В окне «Build Output» найти строку ошибок и предупреждений и убедиться в их отсутствии – Рис 1.8.

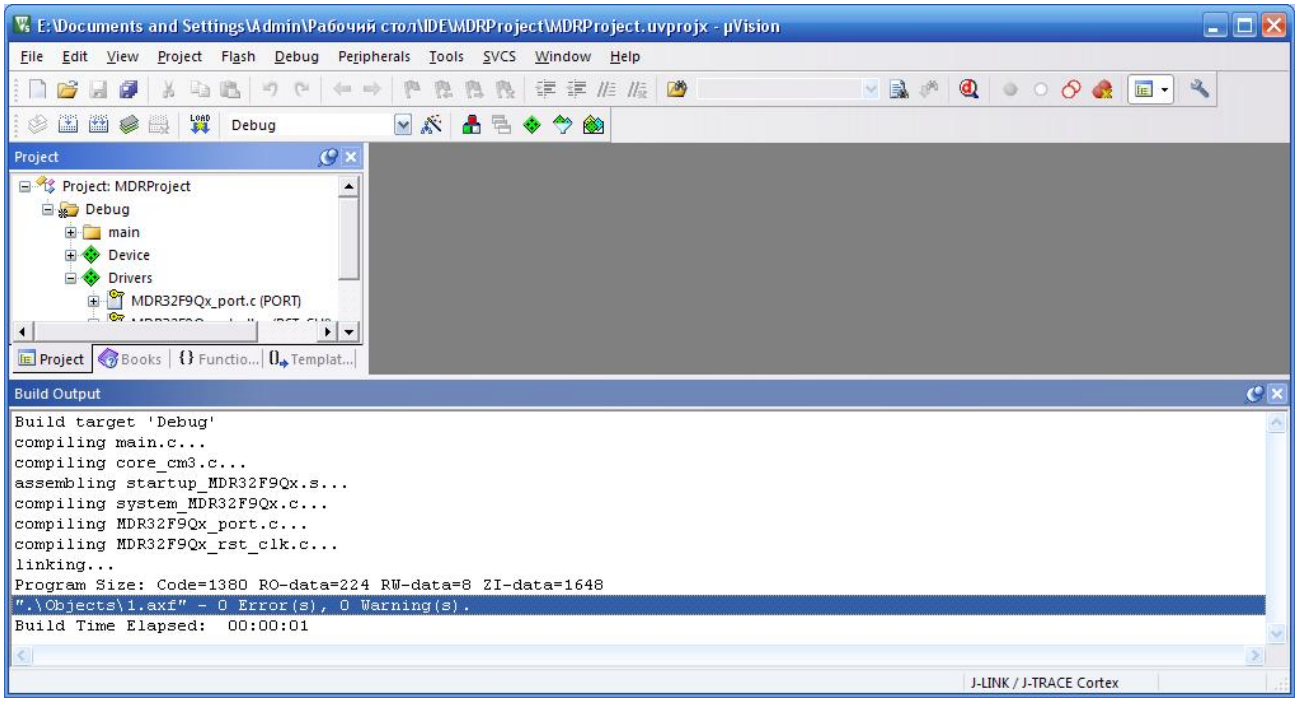


Рисунок 1.8 – Отсутствие ошибок при построении проекта.

5. Установить драйвер программатора J-Link, запустив файл «**InstDrivers.exe**», который находится в папке «USBDriver», которая в свою очередь вложена в папку «J-Link» – Рисунок 1.9. Данная программа не имеет графического вывода, если ничего не произошло после запуска, то драйверы установлены.

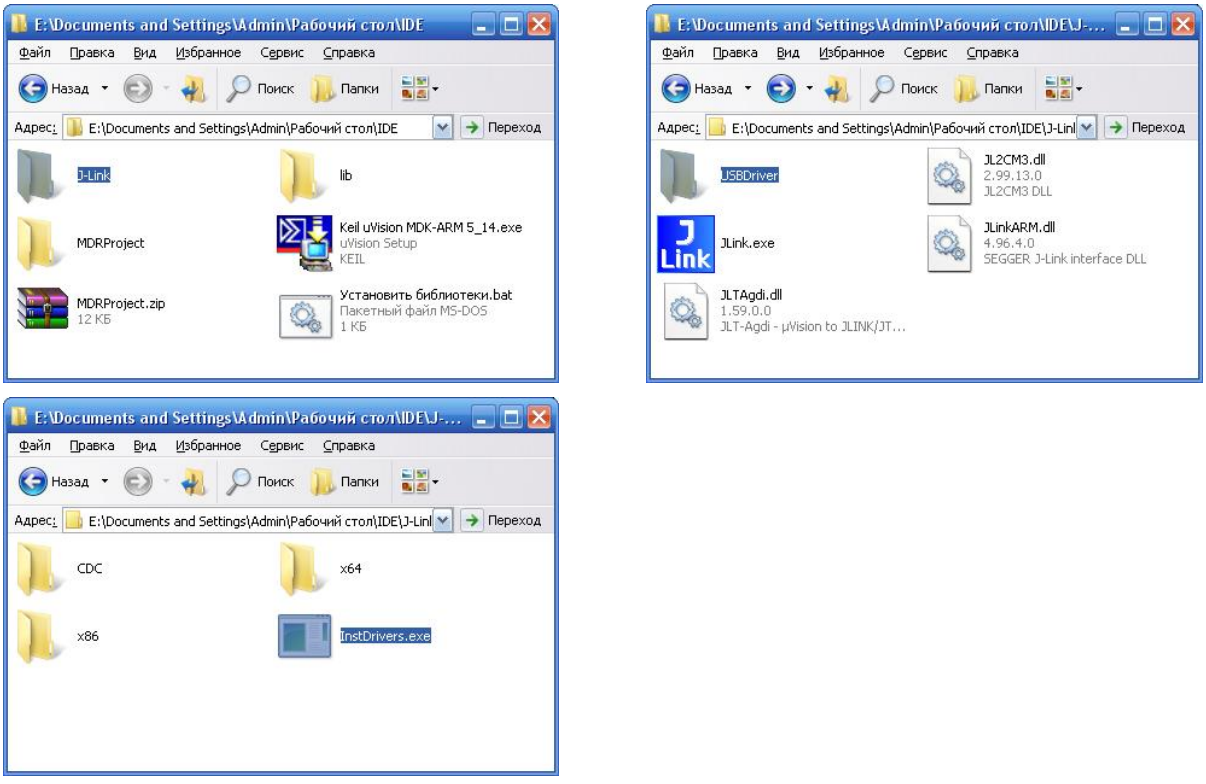


Рисунок 1.9 – Группа скриншотов процесса установки драйвера для программатора J-Link.

После выполнения указанных действий, рабочая среда должна быть готова для выполнения дальнейших лабораторных работ.

6. Подключить к компьютеру программатор J-Link или TP-Link.

Рисунок 1.10 Программатор J-Link (TP-Link)

7. Создание нового проекта.

Создать новый проект, нажав **Project** > **New µVision Project**…

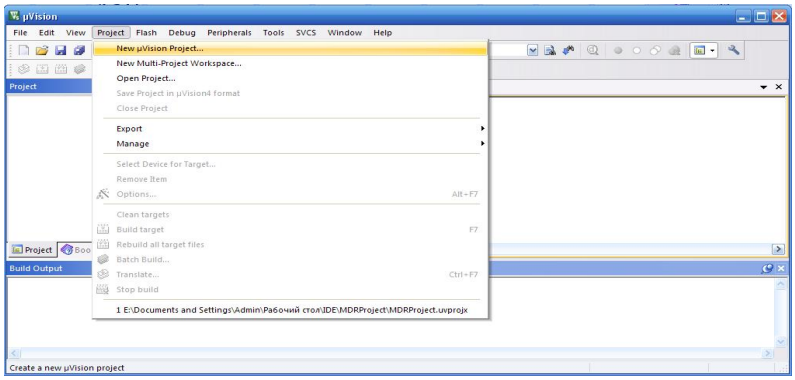


Рисунок 1.11 Создание нового проекта

Указать путь, куда сохранится новый проект (Желательно для проекта создать отдельную папку), задать уникальное имя и нажать кнопку **Сохранить**.

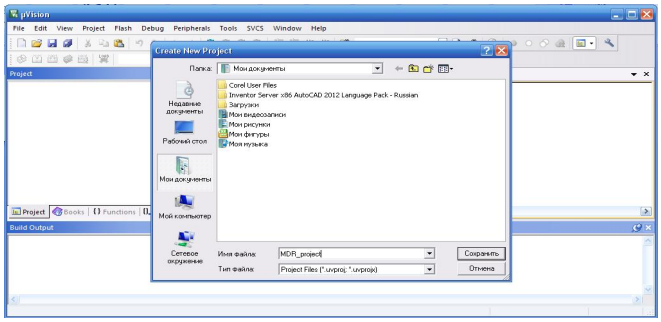


Рисунок 1.12 Процесс сохранения проекта

После именования нового проекта возникает окно выбора процессора, в котором необходимо раскрыть иерархическое дерево доступных процессоров и последовательно выбрать **Milandr > Milandr > Cortex-M3 > MDR1986BE91**

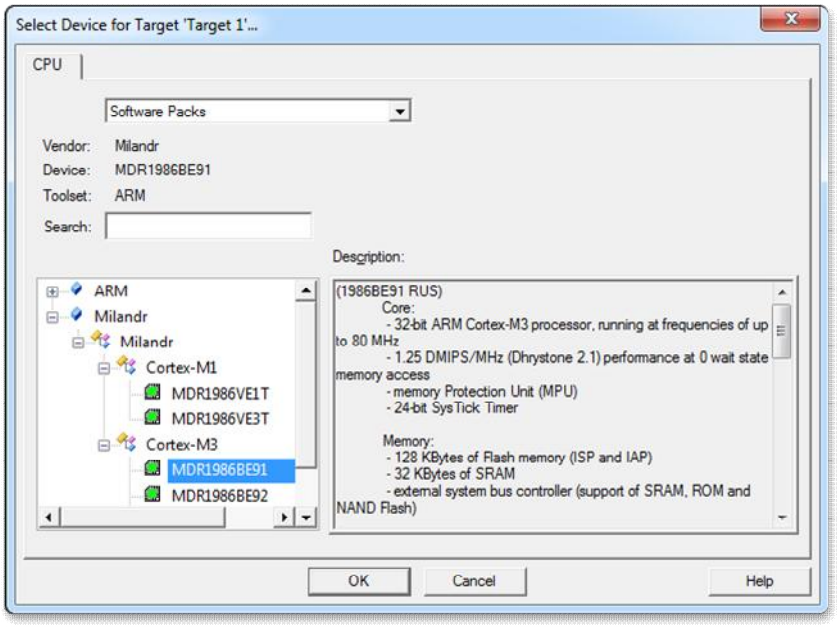


Рисунок 1.13 Выбор марки микропроцессора

После подтверждения выбора Milandr MDR1986BE91 возникнет окно выбора библиотек, в котором необходимо выбрать следующие компоненты **Device > Startup_MDR1986BE9x**, затем **Drivers** > **PORT** и **Drivers > RST_CLK**.

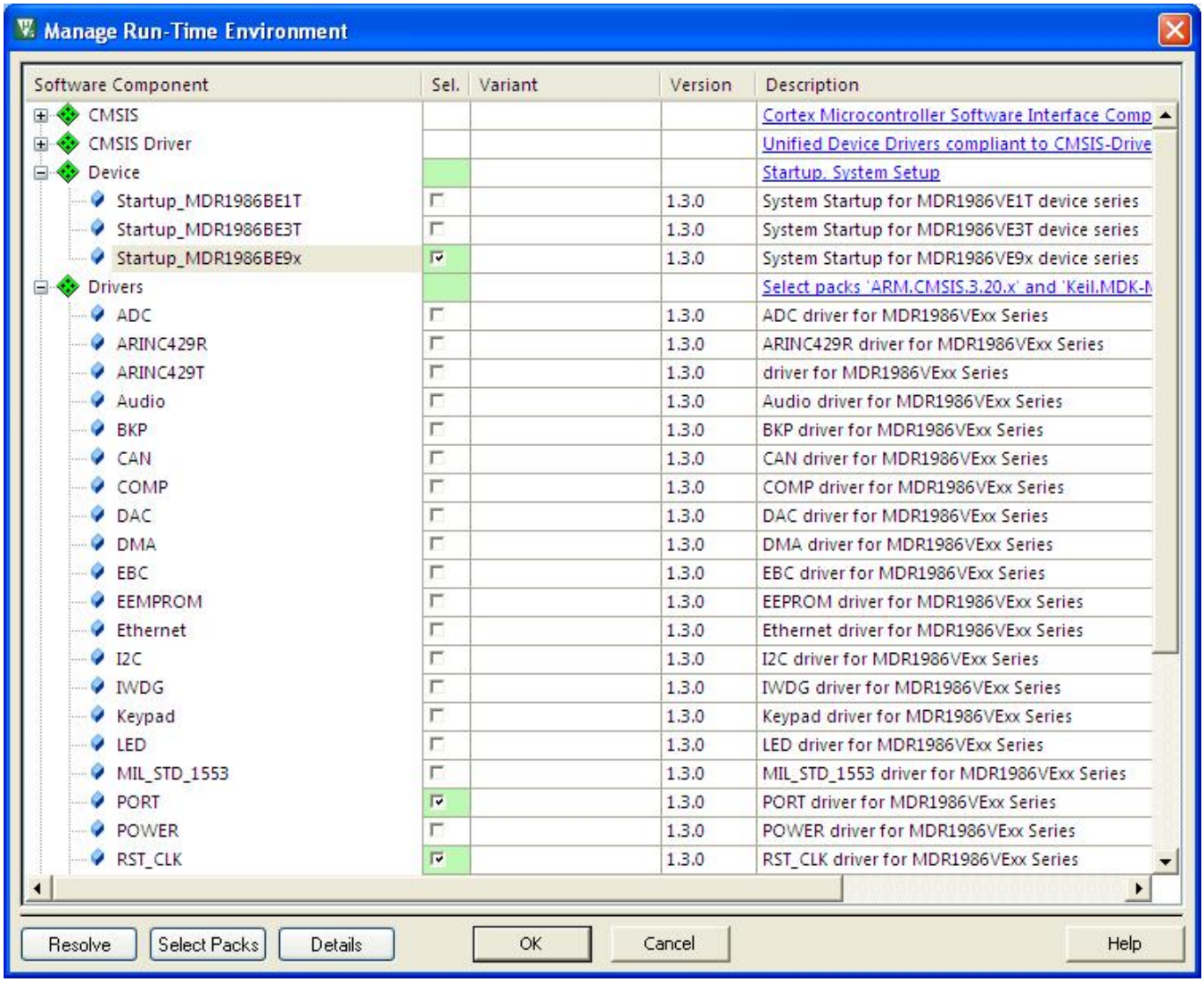


Рисунок 1.14 Настройка библиотек

8. Настройка параметров проекта

После подтверждения выбора библиотек нажать **ОК** и перейти к настройке параметров проекта, нажав **Project > Options for Target** ‘**Target1’…** или нажав сочетание клавиш **Alt+F7**.

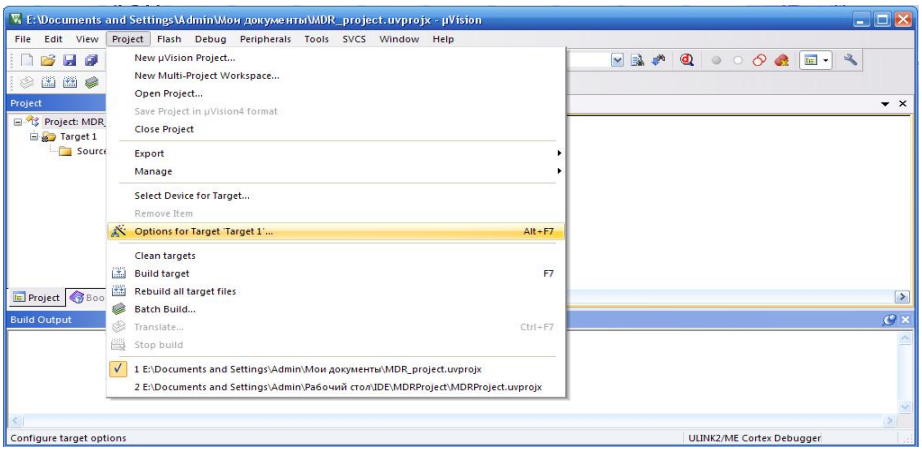


Рисунок 1.15 Выбор пункта меню настройка параметров проекта

На вкладке **Target** установить значение **Xtal (MHz)** равным **8.0**, перейти на вкладку **Debug**.

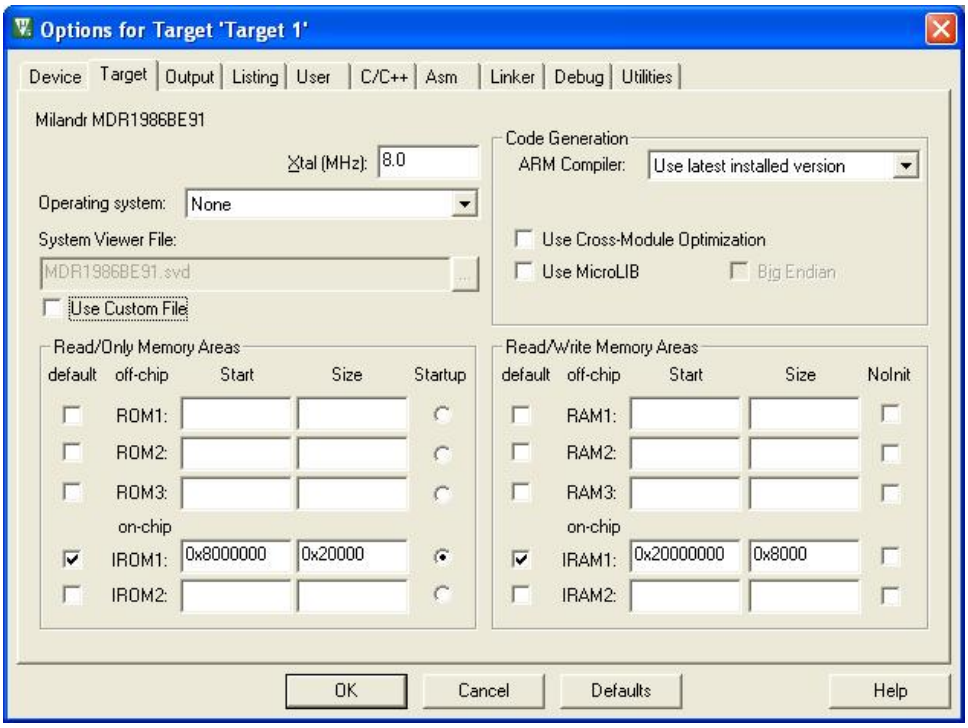


Рисунок 1.16 Вкладка **Target**

На вкладке **Debug** выбрать из выпадающего списка **J-LINK / J-TRACE Cortex.** Затем в окна “**CPU DLL:”** ввести значения “**SARMCM3.DLL**”, а в окна “**Parameter**”, находящиеся при окнах “CPU DLL” и “Driver DLL” ввести значения “**-MPU**”.

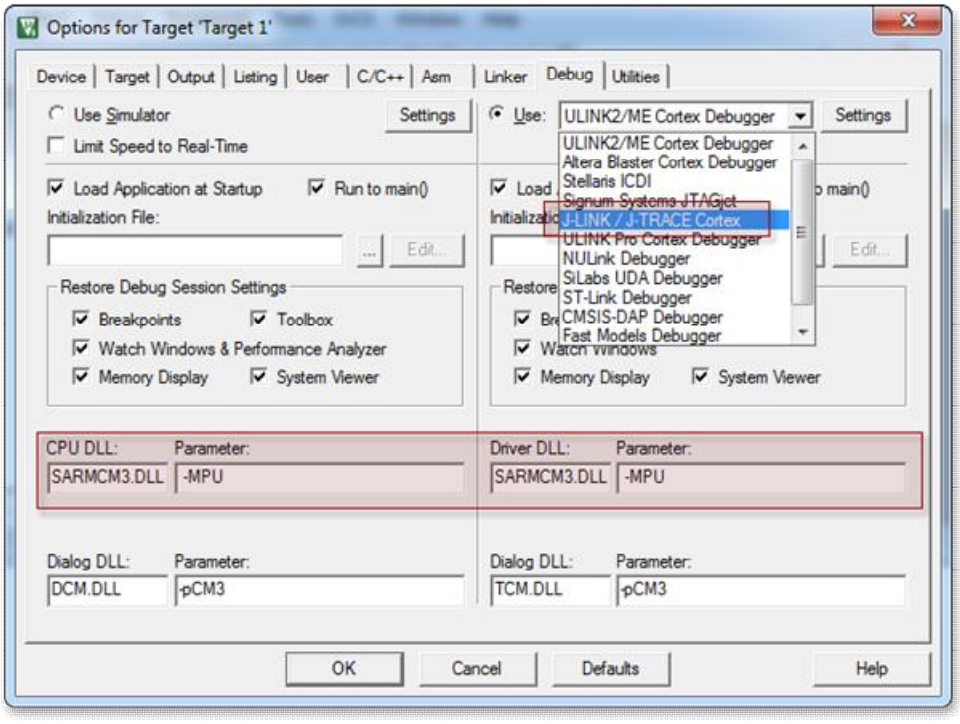


Рисунок 1.17 Вкладка **Debug**

В правом верхнем углу вкладки **Debug** нажать кнопку **Settings** и установить в поле “**Max Clock**” значение “**1MHz**”,в поле “**Port**” значение “**SW**” перейти на вкладку **Flash**

Download.

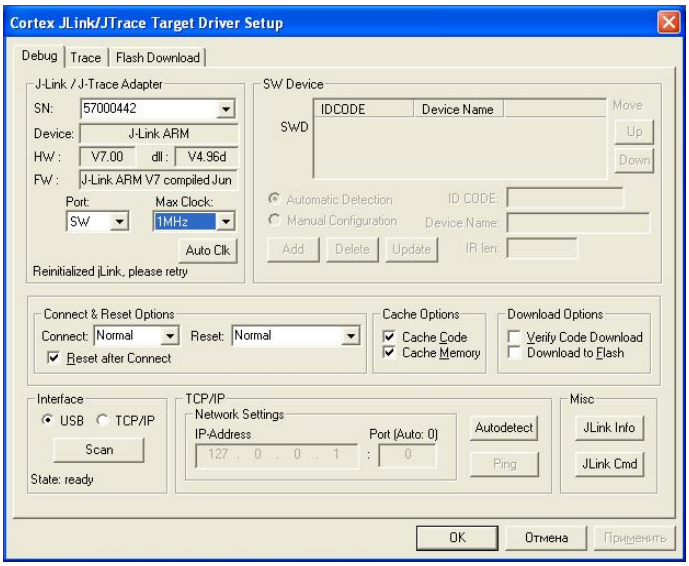


Рисунок 1.18 Окно после нажатия кнопки **Settings**

На вкладке **Flash Download** установить точку напротив “**Erase Full Chip”**, установить галочки напротив строк “**Program”, “Verify”, “Reset and Run”.** В зоне “**RAM for Algorithm**” найти поле “**Size**” и установить значение “**0x0800”**. Далее нажать кнопку “**Add**”, выбрать строку “**MDR32F9x**”, нажать **Add**, затем нажать **ОК** трижды.

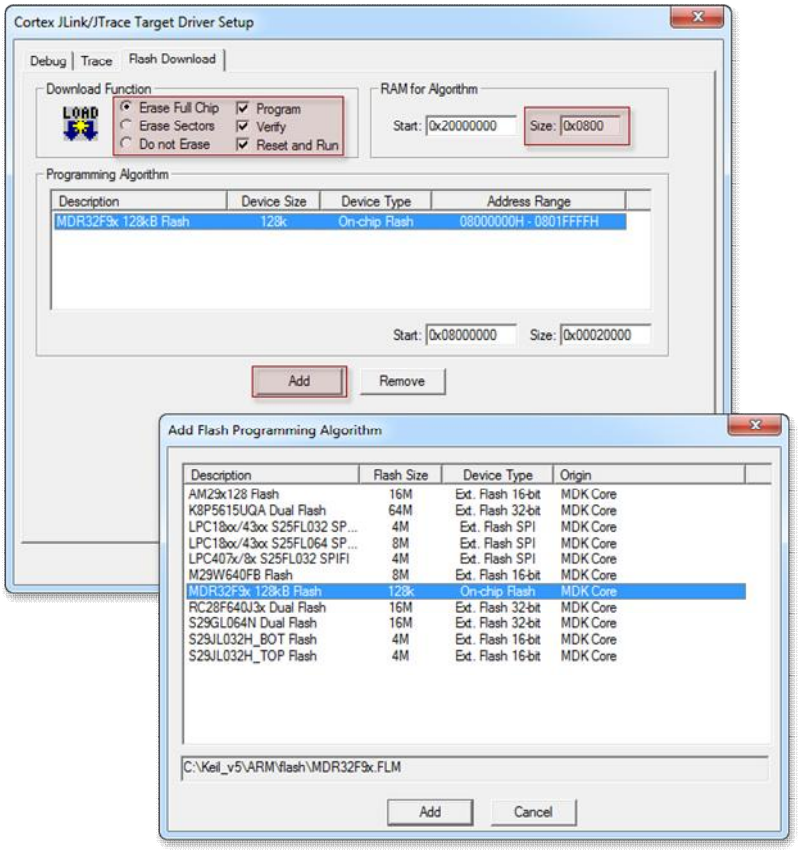


Рисунок 1.19 Вкладка **Flash Download** и окно **Programming Algorithm**

Создайте файл main.c, для этого на вкладке **Project** в списке **Source Group 1** с помощью

правой кнопки мыши выберите меню **Add New Item to Group 'Source Group1'….** В появившемся диалоговом окне введите тип файла C, название main. Рис. 1.20. Сохраните файл в папку с проектом.

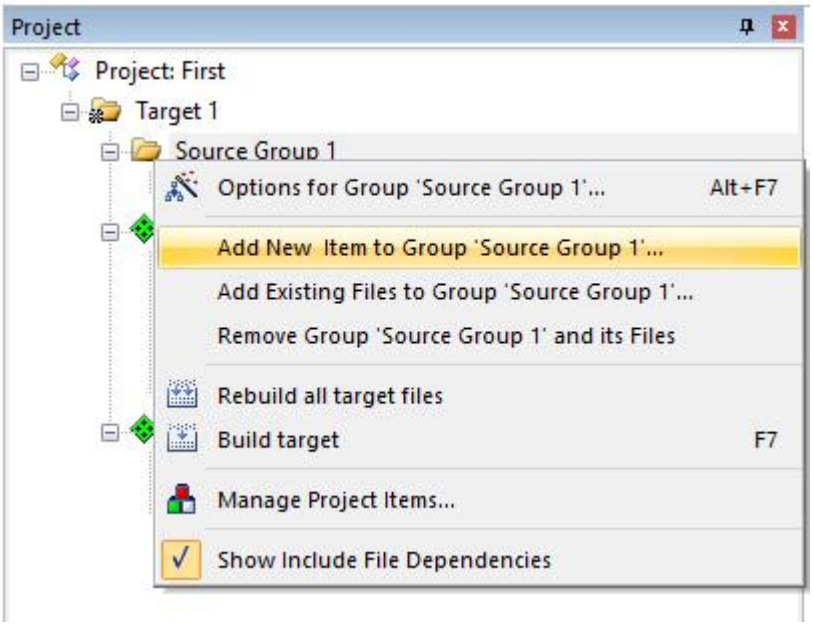


Рисунок 1.20 Добавление первого файла в проект.

Таким образом система готова к программированию. Проверить правильность настроек удастся в ходе выполнения лабораторной работы 2.

Лабораторная работа № 2. Изучение работы портов ввода-вывода

Цель работы:

Изучение основ программирования для микроконтроллеров (МК) ARM на примере программы мигания светодиодами. Изучение работы портов микроконтроллера 1986VE91T.

Приборы и материалы:

1. Отладочная плата MDR1986VE91T Rev 4
2. Программатор J-Link ARM
3. Блок питания 5В, 1.4А
4. ПК с установленной средой программирования Keil uVision

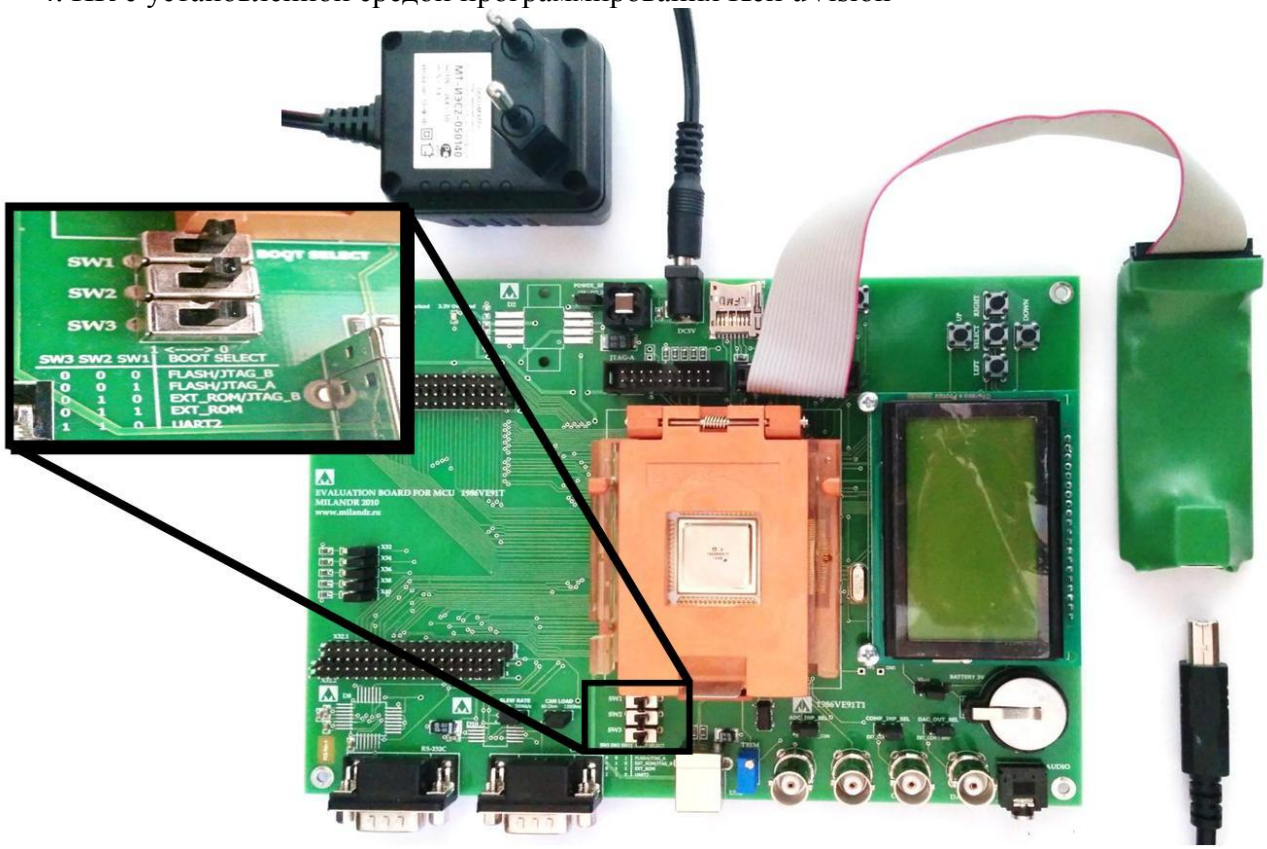


Рисунок 2.1 – Подключение платы 1986VE91T (крупно выделены переключатели выбора режима загрузки)

Порядок работы:

1. Подключить программатор к порту JTAG-B платы (рис. 3.1).
2. Установить переключатели SW1, SW2 и SW3 в положение 0 (рис. 3.1).
3. Подключить блок питания к плате (рис. 1).
4. Подключить программатор J-Link к USB порту компьютера и дождаться окончания установки драйверов.
5. Открыть проект MDRProject в среде программирования Keil uVision. Если проект отсутствует или не открывается, необходимо создать его по рекомендациям к лабораторной работе 2.
6. Скомпилировать проект, нажав кнопку «*Build*». При успешной компиляции, в окне **Build Output** появится надпись *«0 Error(s), 0 Warnings»* (рисунок 3.2).

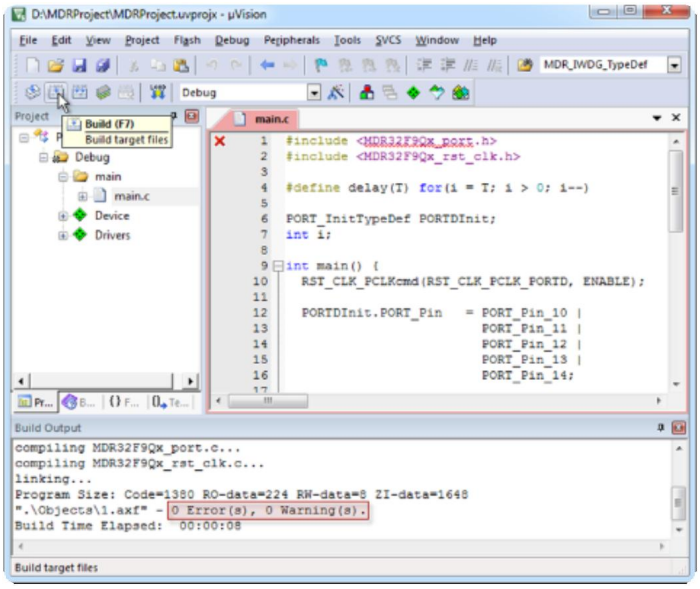


Рисунок 2.2 – Компиляция проекта

7. Подать питание на плату, вставив блок питания в сеть 220В.
8. Загрузить микропрограмму на микроконтроллер с помощью кнопки *Download* (рисунок 3.3).
9. При первой отладке, драйвер программатора J-Link выдаст уведомление о том, что устройство “MDR1986BE91” ему не известно и предложит выбрать устройство вручную. Чтобы проигнорировать уведомление, нажмите кнопку «No» (рисунок 3.4). Успешная загрузка микропрограммы обозначается строчкой «*Verify OK*» в окне **Build Output**.

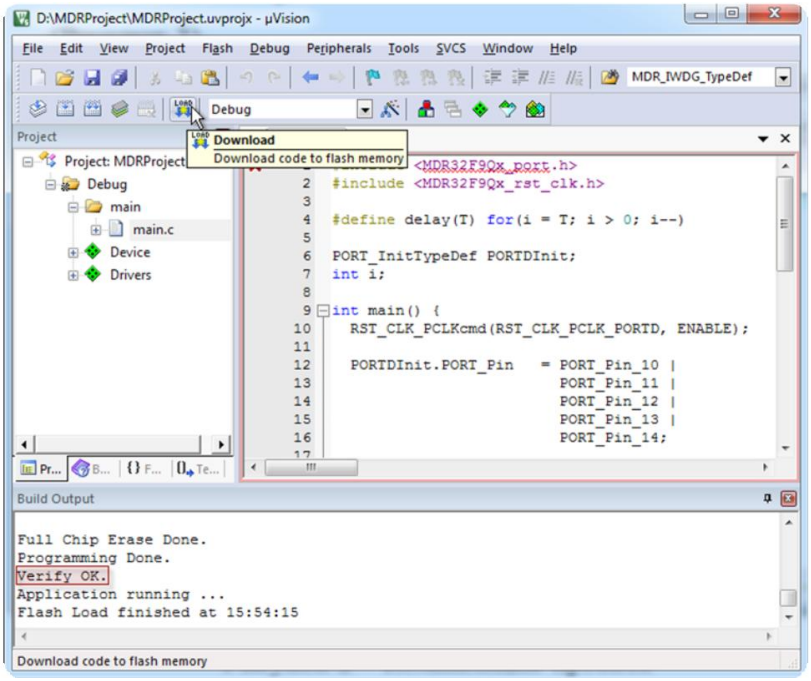


Рисунок 2.3 - Загрузка микропрограммы в устройство

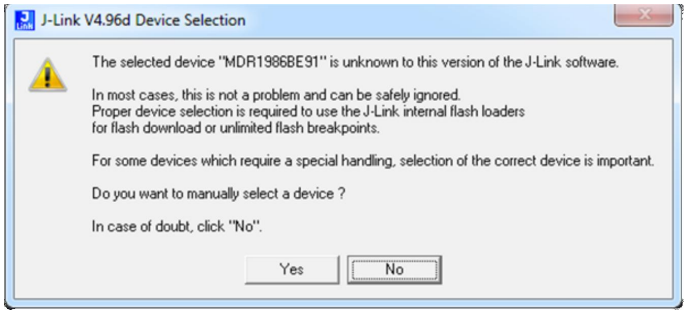


Рисунок 2.4 – Уведомление о неизвестном устройстве

10. Если программатору не удалось загрузить микропрограмму, **попробуйте перевести переключатель SW2 (рисунок 1) в положение 1**, активировав режим загрузки с внешнего носителя данных (EXT_ROM/JTAG_B) и еще раз нажать кнопку *Download*. При успешной загрузке микропрограммы, необходимо вернуть SW2 в положение 0 и перезагрузить микроконтроллер нажатием кнопки RESET. Микропрограмма должна начать исполняться и мигать светодиодами VD7 – VD11.
11. Выполнить индивидуальное задание преподавателя.

Сведения для выполнения

Микроконтроллер – это программно-аппаратный комплекс, позволяющий решать определенный круг узкоспециальных задач без применения компьютера. По сути, МК – это полноценный компьютер с ПЗУ, ОЗУ, интерфейсами ввода-вывода информации и, разумеется, процессором. Однако, в отличие от компьютера, МК обычно не содержит операционной системы и программируется пользователем на исполнение одного необходимого алгоритма. Обычно, этот алгоритм производит управление одним или несколькими устройствами с совершенной произвольной целью. Это может быть сбор данных с нескольких датчиков, поддержание определенного режима чего-либо на основе собранных данных, микроконтроллер может организовывать интерфейс между оконечной аппаратной частью устройства (например, рулевыми машинками самолета) и высокоуровневыми устройствами, например, смартфоном. Область применения МК ограничена только идеями разработчика.

В данном методическом комплексе будет рассмотрена задача использования 32-разрядного микроконтроллера на базе процессорного ядра ARM Cortex-M3 (обычно используется в смартфонах) компании Миландр – **1986BE91**.

Программирование для микроконтроллеров обычно производится на языке Си, однако, при создании микропрограмм для МК архитектуры ARM, повсеместно используются библиотеки, позволяющие отвязать код от конкретного МК и значительно улучшить его переносимость с одного типа микроконтроллера на другой. Библиотеки дают возможность управлять функциями МК с помощью инвариантного программного кода на разных устройствах. Каждая функция МК содержится в своей библиотеке, и они подключаются по мере необходимости.

Для управления подключенными к проекту библиотеками необходимо воспользоваться кнопкой «*Manage Run-Time Environment*» (рисунок 2.5). Минимальный набор необходимых библиотек состоит из основной системной библиотеки **Device** / **Startup_MDR1986E9x** и библиотеки управления тактованием **Device** / **RST_CLK**. Однако, используя только функции этих двух библиотек, можно лишь включить МК, а чтобы что-либо сделать с его помощью, понадобится еще хотя бы одна библиотека, для работы со светодиодами нам потребуется библиотека **Device** / **PORT** для управления портами ввода/вывода. Таким образом, видеокадр окна с подключенными библиотеками начального проекта **MDRProject** представлен на рисунке 2.5.

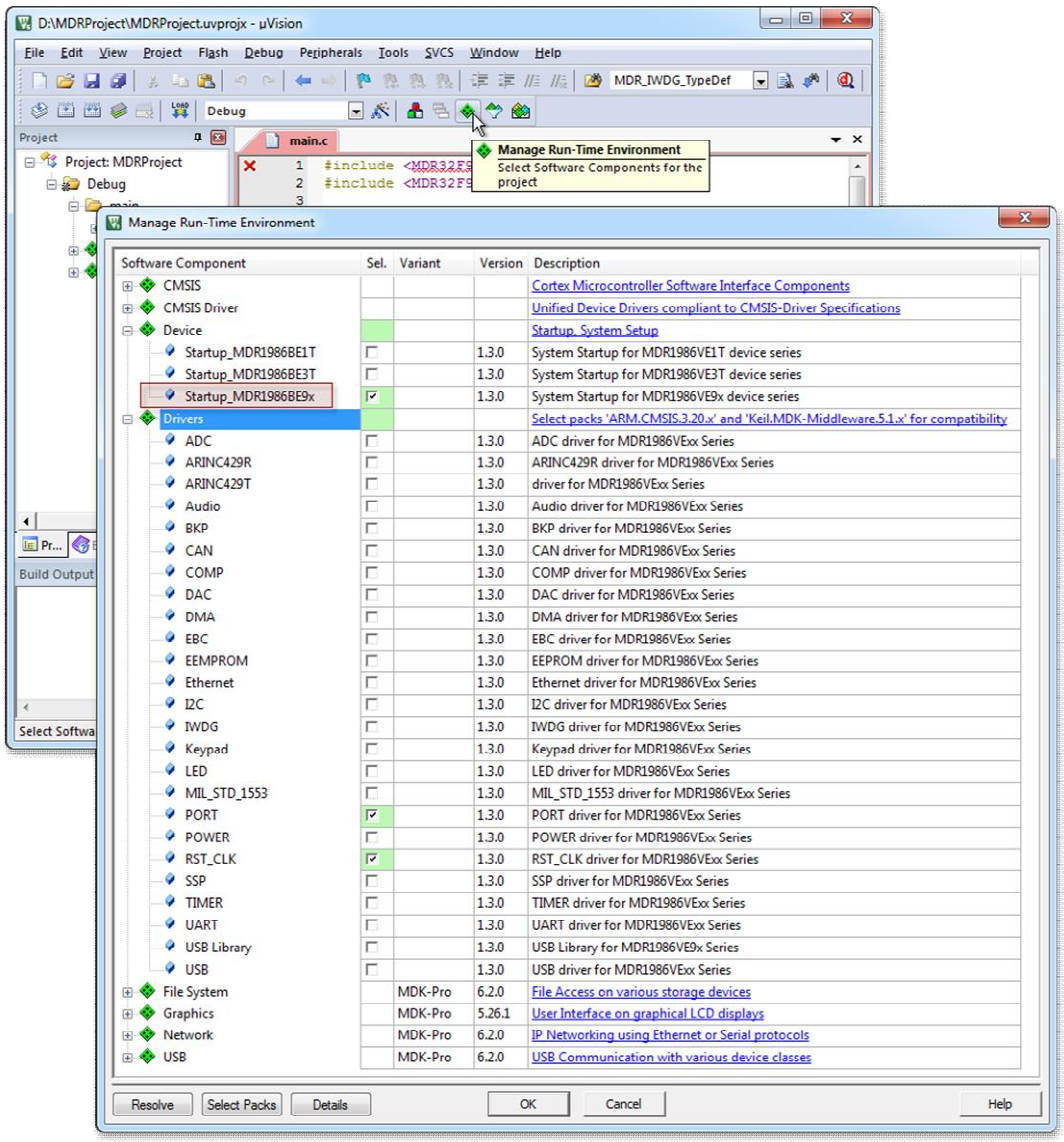


Рисунок 2.5 – Менеджер библиотек проекта

Рассмотрим программный код, позволяющий мигать светодиодами.

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>

// Подключение заголовочных файлов тех библиотек,
// которые непосредственно используются в данном файле исходного кода


#define delay(T) for(i = T; i > 0; i--) // Определение функции задержки (см. примечение 1)

PORT_InitTypeDef PORTDInit;        // Объявление структуры, с помощью которой
// будет происходить инициализация порта (см. примечение 2)

int i; // Глобальная переменная счетчика, которая используется в функции delay()

int main() { // Главная функция, с которой начинается работа программы
 RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTD, ENABLE); // Включение тактования порта D (устройства)

 PORTDInit.PORT_Pin =       PORT_Pin_10 | // (см. примечание 2)
            PORT_Pin_11 |
            PORT_Pin_12 |
            PORT_Pin_13 | // Объявляем номера ножек порта, которые
            PORT_Pin_14; // настраиваются данной структурой
```

```
 PORTDInit.PORT_OE  = PORT_OE_OUT;   // Конфигурация группы выводов как выход
 PORTDInit.PORT_FUNC = PORT_FUNC_PORT; // Работа а режиме порта ввода-вывода
 PORTDInit.PORT_MODE  = PORT_MODE_DIGITAL;     // Цифровой режим
 PORTDInit.PORT_SPEED = PORT_SPEED_SLOW;       // Низкая частота тактования порта

 PORT_Init(MDR_PORTD, &PORTDInit);     //Инициализация порта D объявленной структурой

   while(1){    //Главный цикл (см. примечение 3)
     PORT_SetBits(MDR_PORTD, PORT_Pin_10);      // Установка единицы на 10 пине в порту D
     delay(0xFFFF);               // Задержка (см. примечение 1)
     PORT_ResetBits(MDR_PORTD, PORT_Pin_10);    // Установка нуля на 10 пине в порту D
     PORT_SetBits(MDR_PORTD, PORT_Pin_11);
     delay(0xFFFF);
     PORT_ResetBits(MDR_PORTD, PORT_Pin_11);
     PORT_SetBits(MDR_PORTD, PORT_Pin_12);
     delay(0xFFFF);
     PORT_ResetBits(MDR_PORTD, PORT_Pin_12);
     PORT_SetBits(MDR_PORTD, PORT_Pin_13);
     delay(0xFFFF);
     PORT_ResetBits(MDR_PORTD, PORT_Pin_13);
     PORT_SetBits(MDR_PORTD, PORT_Pin_14);
     delay(0xFFFF);
     PORT_ResetBits(MDR_PORTD, PORT_Pin_14);
   }
}
```

Примечание 1

Поскольку частота работы процессора очень высокая (8 млн. тактов в секунду), глаз не сможет различить мигание светодиода, если написать строчки, включающие и выключающие светодиод подряд. Для формирования задержки в простейшем варианте используют пустой цикл с огромным числом итераций. Недостаток данного способа в том, что пока процессор занят отсчетом итераций, он не может делать что-то еще. Именно такой вид первой программы для МК является классическим. Как и программа, которая выводит на экран надпись “Hello World”, мигание светодиодом с задержкой через цикл не имеет практического применения, но позволяет быстро создать хоть что-то работающее.

Примечание 2

При программировании микроконтроллеров ARM с использованием библиотек повсеместно применяется такой элемент языка Си, как структура. Структура похожа на массив, но каждый элемент структуры имеет имя вместо номера, такое имя называют полем, количество полей строго определено. Например, если принять автомобиль как структуру, то его полями могут быть: размер колеса, количество пассажиров, грузоподъемность, мощность двигателя. Синтаксис обращения к элементам структуры:

```
[имя структуры].[имя элемента]
```

В нашем случае структурой является порт микроконтроллера D. Порт микроконтроллера представляет собой набор выводов микроконтроллера (порт отвечает за 16 Выводов), каждый вывод можно настроить на выполнение той или иной функции, более подробно см.[1]. Краткое описание структуры приведено ниже.

```
typedef struct
{
  uint16_t PORT_Pin;          /*!< Определяет какие ножки порта будут сконфигурированы
                              например значение поля в двоичном коде 0b0000 0001 0000 0010
                              конфигурирует 8 и 1 бит. Для задания значений можно
                              воспользоваться стандартными битовыми масками PORT_Pin_x,
                              которые записываются через побитовое ИЛИ: PORT_Pin_1|PORT_Pin_8.  */
  PORT_OE_TypeDef PORT_OE;    /*!< Определяет режим работы порта: ввод или вывод.  Может
                              принимать значения PORT_OE_IN - ввод (0) или PORT_OE_OUT - вывод (1)*/
  PORT_PULL_UP_TypeDef PORT_PULL_UP;
                              /*!< Определяет включение верхнего подтягивающего резистора
                               PORT_PULL_UP_OFF - выключен (0) или  PORT_PULL_UP_ON - включен (1)  */
  PORT_PULL_DOWN_TypeDef PORT_PULL_DOWN;
                              /*!< Определяет включение нижнего подтягивающего резистора
                               PORT_PULL_DOWN_OFF - выключен(0) или PORT_PULL_DOWN_ON - включен(1) */
  PORT_PD_SHM_TypeDef PORT_PD_SHM;
                              /*!< Определяет включение или выключение триггера Шмидта
```

```
                              PORT_PD_SHM_OFF – выключен(0) или PORT_PD_SHM_ON – включен(1)
                              при выключенном триггере гистерезис составляет 200мВ, а при включенном
                              400мВ*/
  PORT_PD_TypeDef PORT_PD;    /*!< Определяет режим работы ножки. Управляемый драйвер
                              PORT_PD_DRIVER (0) или PORT_PD_OPEN (1) или открытый сток*/
  PORT_GFEN_TypeDef PORT_GFEN;
                              /*!< Определяет режим работы входного фильтра ножки.
                              Выключен PORT_GFEN_OFF (0) или включен PORT_GFEN_ON (1)*/
  PORT_FUNC_TypeDef PORT_FUNC;
                              /*!< Определяет режим работы вывода порта:
                              - Порт PORT_FUNC_PORT (0);
                              - Основная функция PORT_FUNC_MAIN (1);
                              - Альтернативная функция  PORT_FUNC_ALTER (2);
                              - Переопределенная функция  PORT_FUNC_OVERRID (3);*/
  PORT_SPEED_TypeDef PORT_SPEED;
                              /*!< Определяет скорость работы порта:
                              зарезервировано (передатчик отключен)       PORT_OUTPUT_OFF    (0);
                              медленный фронт (порядка 100 нс)            PORT_SPEED_SLOW    (1);
                              быстрый фронт (порядка 20 нс)               PORT_SPEED_FAST    (2);
                              максимально быстрый фронт (порядка 10 нс)   PORT_SPEED_MAXFAST (3);*/
  PORT_MODE_TypeDef PORT_MODE;
                              /*!< Определяет режим работы контроллера:
                              0 – аналоговый                PORT_MODE_ANALOG      = 0x0,
                              1 – цифровой                  PORT_MODE_DIGITAL     = 0x1*/
}PORT_InitTypeDef;
```

Примечание 3

Порядок инициализации устройств МК следующий:

1. Определение структуры соответствующего типа
2. Включение тактирования устройства
3. Заполнение элементов структуры требуемыми значениями
4. Инициализация устройства с параметрами определенными в структуре

Примечание 4

Очередная особенность программирования для МК заключается в том, что программа никогда не заканчивается. Представьте, что вы сделали фонарик с переключением мощности по кнопке и он перестает работать после того, как вы один раз переключили все режимы. Правильный вариант состоит в том, чтобы после последнего режима переходить обратно на первый. Таким образом, необходимо делать программу, в которой при любых внешних воздействиях будет выполняться некоторый код. Из этих соображений, в программе для МК существует такое понятие, как главный цикл. Это бесконечный цикл внутри функции **main()**. Все инструкции внутри главного цикла бесконечно повторяются, пока подается питание на МК. Все, что до главного цикла, выполняется лишь однажды, при включении. Главный цикл **while(1).**

Задания:

Задание 1 (по вариантам)

1. Изменить программу таким образом, чтобы светодиоды «бегали» в обоих направлениях.
2. Изменить программу таким образом, чтобы светодиоды «бежали» из центра в края.
3. Изменить программу таким образом, чтобы светодиоды «бежали» из краев в центр.
4. Изменить программу таким образом, чтобы светодиоды «бегали» с задержкой в один светодиод (в каждый момент времени должны гореть два светодиода).
5. Изменить программу таким образом, чтобы чётные светодиоды переключались в 2 раза быстрее нечётных.

Задание 2 (общее)

Организовать считывание кнопок джойстика на отладочной плате и реагировать на их состояние соответствующими светодиодами, пользуясь следующей информацией:

- Кнопки джойстика располагаются в порту *C*, номера пинов с 10 по 14.

- Константа поля **PORT_OE** структуры **PORT_InitTypeDef** для конфигурации группы портов как вход имеет вид **PORT_OE_IN**.
- Кнопки на плате при нажатии соединяются с цепью GND (землей).
- Сигнатура функции для чтения состояния пина имеет вид:**uint8_t PORT_ReadInputDataBit(MDR_PORT_TypeDef* PORTx, uint32_t PORT_Pin)**

Лабораторная работа № 3. Изучение работы таймера

Цель работы:

Изучение основных особенностей работы с таймерами с использованием прерывания при программировании для микроконтроллеров (МК) ARM.

Приборы и материалы:

1. Отладочная плата MDR1986VE91T Rev 4
2. Программатор J-Link ARM
3. Блок питания 5В, 1.4А
4. ПК с установленной средой программирования Keil uVision

Порядок работы:

1. Собрать аппаратную часть по рекомендациям лабораторной работы № 1
2. Открыть проект MDRProject в среде программирования Keil uVision.
3. Подключить к проекту библиотеку TIMER, необходимую для работы с таймерами.

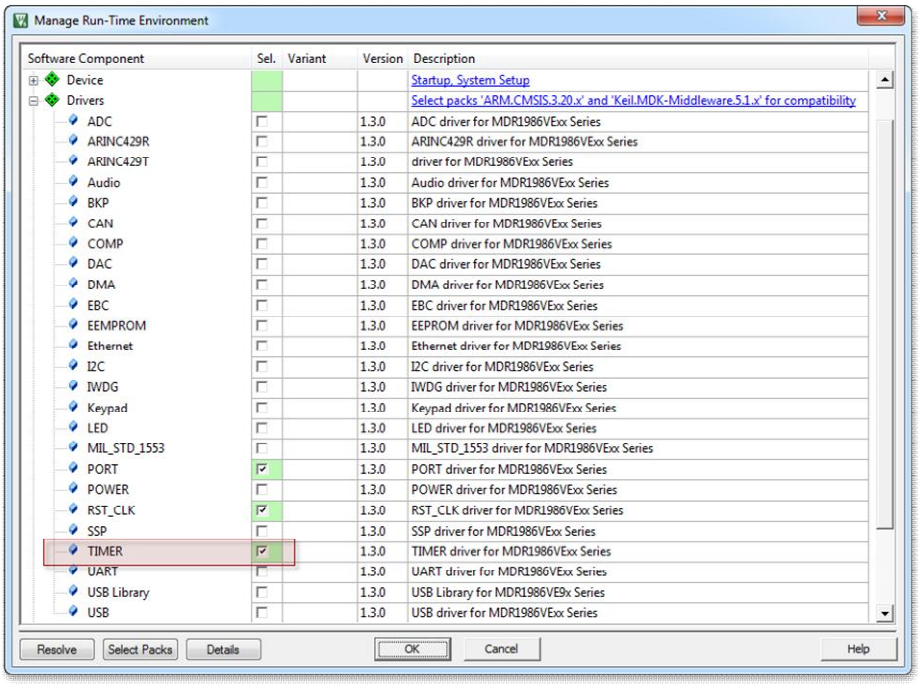


Рисунок 3.1 – Библиотеки проекта для работы с таймерами

4. Добавить в заголовок основного файла исходного кода *main.c* ссылку на заголовочный файл библиотеки TIMER:

```
#include <MDR32F9Qx_timer.h>
```

5. Вынести содержимое главного цикла в отдельную функцию, которая при каждом вызове

переключает светодиод на следующий:

```
uint8_t cur_i;
void NextLED(){
    switch(cur_i++ % 5) {
            case 0:
                PORT_ResetBits(MDR_PORTD, PORT_Pin_14);
                PORT_SetBits(MDR_PORTD, PORT_Pin_10);
                break;
            case 1:
                PORT_ResetBits(MDR_PORTD, PORT_Pin_10);
                PORT_SetBits(MDR_PORTD, PORT_Pin_11);
                break;
            case 2:
                PORT_ResetBits(MDR_PORTD, PORT_Pin_11);
                PORT_SetBits(MDR_PORTD, PORT_Pin_12);
                break;
            case 3:
                PORT_ResetBits(MDR_PORTD, PORT_Pin_12);
                PORT_SetBits(MDR_PORTD, PORT_Pin_13);
                break;
            case 4:
                PORT_ResetBits(MDR_PORTD, PORT_Pin_13);
                PORT_SetBits(MDR_PORTD, PORT_Pin_14);
                break;
    }
}
```

6. Инициализировать переменную **`cur_i`** нулём в функции **`main()`** до главного цикла, чтобы при первом вызове директиве **`cur_i++`**, было к чему прибавлять единицу.
7. Создать функцию **`TimerInit()`**, в которой производится инициализация аппаратного таймера **`TIMER1`** и соответствующего прерывания по переполнению, и вызвать ее в функции **`main()`** до главного цикла **`while(1)`**:

```
//Объявление инициализационной структуры
TIMER_CntInitTypeDef TIM1Init;

void TimerInit(){
        //Включение тактирования таймера
        RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER1, ENABLE);

        //Установка первого делителя тактовой частоты таймера
        TIMER_BRGInit(MDR_TIMER1, TIMER_HCLKdiv1);

        //Загрузка значений по-умолчанию в структуру TIM1Init
        TIMER_CntStructInit(&TIM1Init);
        TIM1Init.TIMER_Prescaler  = 8000; // Второй делитель частоты
        TIM1Init.TIMER_Period     = 1000; // Период до обновлния или основание счета
        TIMER_CntInit(MDR_TIMER1, &TIM1Init);

        //Настройка прерывания
        NVIC_EnableIRQ(Timer1_IRQn);      //Включение прерываний от TIMER1
        NVIC_SetPriority(Timer1_IRQn, 0); //Установка приоритета прерываний 0-15

        //Включение прерывания при равенстве нулю значения TIMER1
        TIMER_ITConfig(MDR_TIMER1, TIMER_STATUS_CNT_ZERO, ENABLE);

        //Запуск таймера
        TIMER_Cmd(MDR_TIMER1, ENABLE);
}
```

8. Написать обработчик прерывания TIMER1, в котором вызвать функцию **`NextLED()`**.

```
void Timer1_IRQHandler() {
        //Проверка что причина прерывания – обновление таймера
        if(TIMER_GetITStatus(MDR_TIMER1, TIMER_STATUS_CNT_ZERO)){
            NextLED();

            //Очистка флага прерывания (это необходимо делать в конце)
            TIMER_ClearITPendingBit(MDR_TIMER1, TIMER_STATUS_CNT_ZERO);
        }
}
```

9. Загрузить и отладить программу по рекомендациям в лабораторной работе № 2. Полный текст программы можно найти в файле Timer_Int.c
10. Выполнить индивидуальное задание.

Сведения для выполнения

Рисунок 2 иллюстрирует работу таймера в микроконтроллерах. После запуска, таймер начинает прибавлять к начальному значению счетчика фиксированное число через фиксированный промежуток времени, который устанавливается двумя делителями тактовой частоты МК. При достижении таймером значения периода, таймер «Обновляется» и сбрасывается в 0. При этом срабатывает соответствующее прерывание (если оно настроено). Можно настроить прерывание, возникающее при данном событии. Таким образом, в коде выше, общий делитель частоты равен 8000, что при делении на тактовую частоту МК 8 МГц, дает тактовую частоту таймера 1000 раз в секунду. Период установлен в 1000, что позволяет генерировать прерывания каждую секунду.

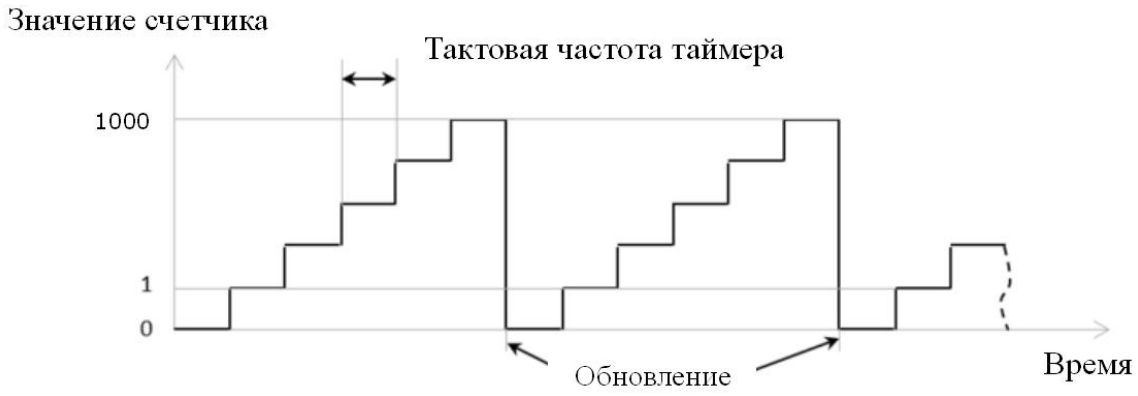


Рисунок 3.2 – Иллюстрация работы таймера

Преимущество такого способа отсчета временных интервалов в том, что главный цикл при этом пуст и МК в промежутке между прерываниями может выполнять любой другой код. Обычно программа для МК организована таким образом, чтобы прерывания меняли переменную состояния, а главный цикл ее проверял и выполнял различные действия в зависимости от текущего состояния.

Таймеры МК 1986ВЕ9Т выполнены на основе 16-битного перезагружаемого счетчика, который синхронизируется с выхода 16-битного предделителя. Перезагружаемое значение хранится в отдельном регистре. Счет может быть прямой, обратный или двунаправленный (сначала прямой до определенного значения, а затем обратный).

Каждый из трех таймеров микроконтроллера содержит 16-битный счетчик, 16-битный предделитель частоты и 4-канальный блок захвата/сравнения. Их можно синхронизировать системной синхронизацией, внешними сигналами или другими таймерами.

Помимо составляющего основу таймера счетчика, в каждый блок таймера также входит четырехканальный блок захвата/сравнения. Данный блок выполняет функции захвата и сравнения, и ряд специальных функций. Таймеры имеют 4 канала схем захвата и ШИМ с функциями формирования «мертвой зоны» и аппаратной блокировки. Каждый из таймеров может генерировать прерывания и запросы прямого доступа к памяти.

Инициализация таймера и системы прерываний состоит из нескольких этапов:

1. Включение тактирования таймера.
2. Установка первого делителя тактовой частоты таймера
3. Заполнение элементов структуры требуемыми значениями
4. Инициализация устройства с параметрами определенными в структуре
5. Настройка и включение прерываний
6. Разрешение работы таймера

Рассмотрим структуру для инициализации таймера подробнее:

```
typedef struct {

        uint16_t TIMER_IniCounter;    /*!< Определяет начальное значение счетчика, может быть задан в
                                           диапазоне 0x0000 and 0xFFFF. */
        uint16_t TIMER_Prescaler;     /*!< определяет второй делитель тактовой частоты таймера. Может
                                      быть задан в диапазоне 0x0000 and 0xFFFF. Расчет тактовой
                                      частоты таймера можно вести по формуле
                                      CLK = TIMER_CLK/(TIMER_Prescaler + 1) */
        uint16_t TIMER_Period;        /*!< Определяет основание счета счетчика которое зашлется в
                                      регистр Auto-Reload Register (ARR) при следующем обновлении.
                                      Может  быть задан в диапазоне 0x0000 and 0xFFFF.*/

        uint16_t TIMER_CounterMode;   /*!< Определяет режим работы таймера:
счет тактовой частоты в одном направлении   TIMER_CntMode_ClkFixedDir
счет тактовой частоты с автоматическим реверсированием TIMER_CntMode_ClkChangeDir
счет событий в одном направлении TIMER_CntMode_EvtFixedDir
счет событий с автоматическим реверсированием TIMER_CntMode_EvtChangeDir */

        uint16_t TIMER_CounterDirection; /*!< Определяет направление счета:
суммирующий счетчик TIMER_CntDir_Up
вычитающий счетчик TIMER_CntDir_Dn  */

        uint16_t TIMER_EventSource;   /*!< Определяет источник событий (тактирования) для таймера.
не определен (тактируется от тактовой частоты)      TIMER_EvSrc_None
Таймер 1 (переключается по обновлению таймера 1)    TIMER_EvSrc_TM1
Таймер 2 (переключается по обновлению таймера 2)    TIMER_EvSrc_TM2
Таймер 3 (переключается по обновлению таймера 3)    TIMER_EvSrc_TM3
Событие в канале 1                                  TIMER_EvSrc_CH1
Событие в канале 2                                  TIMER_EvSrc_CH2
Событие в канале 3                                  TIMER_EvSrc_CH3
Событие в канале 4                                  TIMER_EvSrc_CH4
Событие на входе ETR                                TIMER_EvSrc_ETR       */


        uint16_t TIMER_FilterSampling;  /*!< Определяет частоту сэмплирования входных данных(FDTS).
 Частота сэмплирования равна частоте таймера        TIMER_FDTS_TIMER_CLK_div_1
 Частота сэмплирования равна частоте таймера/2      TIMER_FDTS_TIMER_CLK_div_2
 Частота сэмплирования равна частоте таймера/3      TIMER_FDTS_TIMER_CLK_div_3
 Частота сэмплирования равна частоте таймера/4      TIMER_FDTS_TIMER_CLK_div_4  */

        uint16_t TIMER_ARR_UpdateMode; /*!< Разрешение мгновенного обновления ARR
ARR будет перезаписан в момент записи в ARR         TIMER_ARR_Update_Immediately
ARR будет перезаписан при завершении счета CNT      TIMER_ARR_Update_On_CNT_Overflow             */

        uint16_t TIMER_ETR_FilterConf;/*!< Определяет конфигурацию фильтра на входе ETR. Более
подробно можно посмотреть в файле MDR32F9Qx_timer.h определение @ref IMER_FilterConfiguration */

        uint16_t TIMER_ETR_Prescaler; /*!< Определяет предделитель тактовой частоты на входе ETR
Более подробно можно посмотреть в файле MDR32F9Qx_timer.h определение @ref TIMER_ETR_Prescaler*/

        uint16_t TIMER_ETR_Polarity;  /*!< Определяет полярность ETR сигнала.
                                      определение в @ref TIMER_ETR_Polarity */

        uint16_t TIMER_BRK_Polarity;  /*!< Определяет полярность BRK сигнала.
                                      определение в @ref TIMER_BRK_Polarity */
} TIMER_CntInitTypeDef;
```

Прерывания или IRQ - это исключения, вызываемые периферийными устройствами или программными запросами. Все прерывания асинхронны по отношению к выполняемым инструкциям, то есть микропроцессор завершает текущую операцию и автоматически вызывает соответствующую процедуру обработки прерывания.

Задания:

Задание 1

Изменить интервал переключения светодиодов на 500мс.

Задание 2

Изменить процедуру обработки прерывания таким образом, чтобы интервалы переключение светодиодов для четных и нечетных светодиодов были различными.

Приложение: полный текст программы

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_timer.h>

PORT_InitTypeDef PORTDInit;
void PortsInit(){// Процедура инициализации светодиодов
  PORT_StructInit(&PORTDInit); //Load defaults
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTD, ENABLE);
  PORTDInit.PORT_Pin   = PORT_Pin_10 | PORT_Pin_11 | PORT_Pin_12 | PORT_Pin_13 | PORT_Pin_14;
  PORTDInit.PORT_OE    = PORT_OE_OUT;
  PORTDInit.PORT_MODE  = PORT_MODE_DIGITAL;
  PORTDInit.PORT_SPEED = PORT_SPEED_SLOW;
  PORT_Init(MDR_PORTD, &PORTDInit);
}

//Процедура переключения светодиодов
uint8_t cur_i;
void NextLED(){
       switch(cur_i++ % 5) {
              case 0:
                     PORT_ResetBits(MDR_PORTD, PORT_Pin_14);
                     PORT_SetBits(MDR_PORTD, PORT_Pin_10);
                     break;
              case 1:
                     PORT_ResetBits(MDR_PORTD, PORT_Pin_10);
                     PORT_SetBits(MDR_PORTD, PORT_Pin_11);
                     break;
              case 2:
                     PORT_ResetBits(MDR_PORTD, PORT_Pin_11);
                     PORT_SetBits(MDR_PORTD, PORT_Pin_12);
                     break;
              case 3:
                     PORT_ResetBits(MDR_PORTD, PORT_Pin_12);
                     PORT_SetBits(MDR_PORTD, PORT_Pin_13);
                     break;
              case 4:
                     PORT_ResetBits(MDR_PORTD, PORT_Pin_13);
                     PORT_SetBits(MDR_PORTD, PORT_Pin_14);
                     break;
       }
}
TIMER_CntInitTypeDef TIM1Init;
void TimerInit(){//Процедура инициализации таймера
  RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER1, ENABLE);     // Включение тактирования
  TIMER_BRGInit(MDR_TIMER1, TIMER_HCLKdiv1); // Настройка делителя тактовой частоты
  TIMER_CntStructInit(&TIM1Init);    //Заполнение структуры значениями по умолчанию
  //Задание предделителя тактовой частоты
  TIM1Init.TIMER_Prescaler        = 8000;
  //Задание модуля счета
  TIM1Init.TIMER_Period           = 1000;
  // Процедура инициализации с помощью структуры
  TIMER_CntInit(MDR_TIMER1, &TIM1Init);

  //Включение прерываний
  NVIC_EnableIRQ(Timer1_IRQn);
  //Установка приоритета прерываний
  NVIC_SetPriority(Timer1_IRQn, 0);
  TIMER_ITConfig(MDR_TIMER1, TIMER_STATUS_CNT_ZERO, ENABLE);

  TIMER_Cmd(MDR_TIMER1, ENABLE);
}
//Процедура обработки прерывания вызванного таймером
void Timer1_IRQHandler() {
  if(TIMER_GetITStatus(MDR_TIMER1, TIMER_STATUS_CNT_ZERO)){
    NextLED();
    // Очистка флага прерывания в таймере (предотвращает повторный вызов того же прерывания)
    TIMER_ClearITPendingBit(MDR_TIMER1, TIMER_STATUS_CNT_ZERO);
  }
}
int main() {
  PortsInit();
  TimerInit();
  cur_i = 0;
  while(1){
  // Silence is priceless
  }
}
```

Лабораторная работа № 4. Изучение цифро-аналогового преобразователя

Цель работы:

Изучение основных особенностей работы с цифро-аналоговым преобразователем (ЦАП) при программировании для микроконтроллеров (МК) ARM. Разработка программы для генерации синусоиды.

Приборы и материалы:

1. Отладочная плата MDR1986VE91T Rev 4.
2. Программатор J-Link ARM.
3. Блок питания 5В, 1.4А.
4. ПК с установленной средой программирования Keil uVision.
5. Осциллограф.
6. Коаксиальный кабель для соединения двух BNC выходов.

Порядок работы:

1. Собрать аппаратную часть по рекомендациям лабораторной работы № 2
2. Открыть проект MDRProject в среде программирования Keil uVision.
3. Подключить в менеджере ***Manage run-time environment*** к проекту библиотеки DAC, PORT, RST_CLK, необходимые для работы с ЦАП.
4. Стереть имеющийся в файле *main.c* исходный код и добавить в начало заголовочные файлы, необходимые в данном проекте:

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_dac.h>
#include <math.h>
```

5. Добавить функцию инициализации порта для ЦАП

```
PORT_InitTypeDef PORTEInit; //Объявление структуры
void DACPortInit(){
    PORT_StructInit(&PORTEInit); //Загрузка умолчаний
        RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTE, ENABLE); //Тактование
        /* Настройка DAC1 (стр.11 спецификации) */
        PORTEInit.PORT_Pin  = PORT_Pin_9; //Пин 9
        PORTEInit.PORT_OE   = PORT_OE_OUT; // Порт E
        PORTEInit.PORT_MODE = PORT_MODE_ANALOG; //Режим Аналоговый
  PORT_Init(MDR_PORTE, &PORTEInit);    //Настройка порта
}
```

6. Добавить функцию инициализации ЦАП

```
void DACInit(){
    RST_CLK_PCLKcmd(RST_CLK_PCLK_DAC, ENABLE); //Тактование
    DAC1_Init(DAC1_AVCC);   //Настройка DAC1 на работу с AVCC
    DAC1_Cmd(ENABLE);       //Активация DAC1
}
```

Первая строчка функции **DACInit()** подает тактовый сигнал на периферийное устройство ЦАП. Функция **DAC1_Init()** позволяет настроить источник опорного напряжения для ЦАП1. Последняя строчка разрешает работу ЦАП1. Если предполагается использование внешнего источника, необходимо передать в функцию значение **DACn_REF**, где . В противном случае, для использования напряжения питания МК в качестве опорного, необходимо передать в функцию значение **DACn_AVCC**.

7. Добавить в файл исходного кода функцию **main()**.

```
float a;
int main() {
    RST_CLK_PCLKcmd(RST_CLK_PCLK_RST_CLK, ENABLE);
    DACPortInit();
    DACInit();

    while(1){
          for (a=0; a<360; a+=5)
               DAC1_SetData((sinf(a*PI/180)*SCALE + SCALE)/2);
    }
}
```

В приведенном выше фрагменте кода выполняется вызов функции **DAC1_SetData()** с аргументами, которые вычисляются по функции синуса от равномерно увеличивающегося аргумента. Так как синус находится в пределах от -1 до 1, его необходимо масштабировать в отрезок от 0 до максимального значения. Так как ЦАП 12-битный, максимальное значение равно 0xFFF (12 двоичных единиц). Таким образом, для перевода значений аргумента функции **DAC1_SetData()** в вольты, необходимо использовать формулу , где V_{cc} является напряжением питания МК (3,3В). С учётом сказанного, необходимо добавить определения констант после подключения библиотек.

```
#define PI 3.14159265
#define MAX 2 /*Volt*/
#define SCALE (0x7FF * MAX) / 3.3
```

8. Скомпилировать исходный код и загрузить его в МК
9. Переключить перемычку **DAC_OUT_SEL** в положение **EXT_CON** (рисунок 5.1).

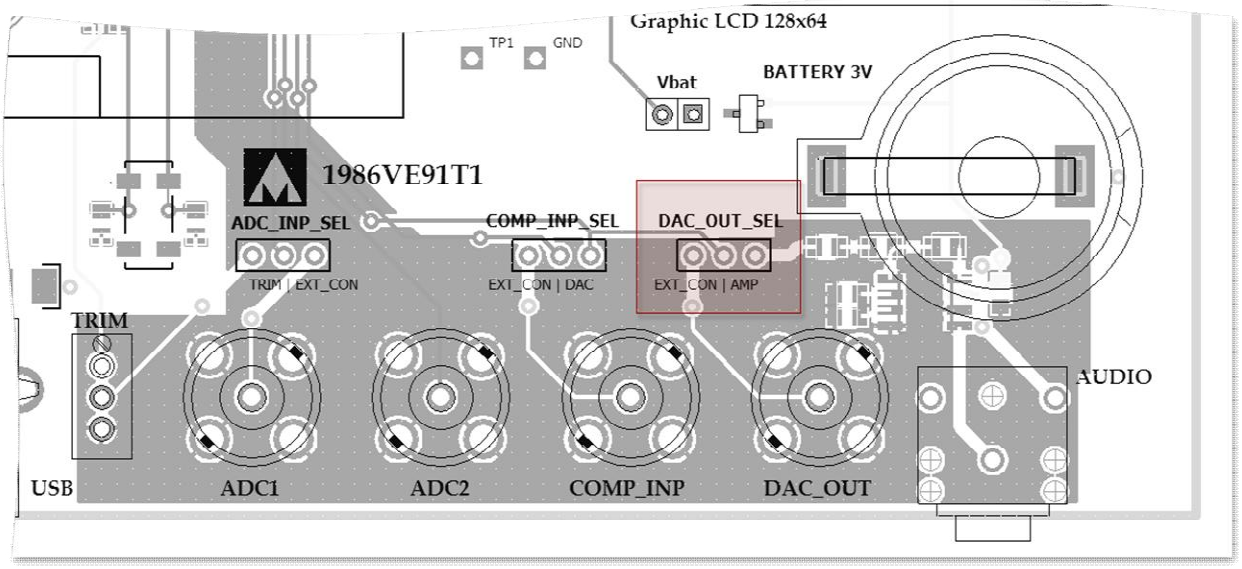


Рисунок 4.1 - Перемычка DAC_OUT_SEL

10. Соединить BNC вывод **DAC_OUT** со входом осциллографа.

11. Настроить вертикальный и горизонтальный масштаб осциллографа до получения синусоид, аналогичных изображенным на рисунке 4.2.

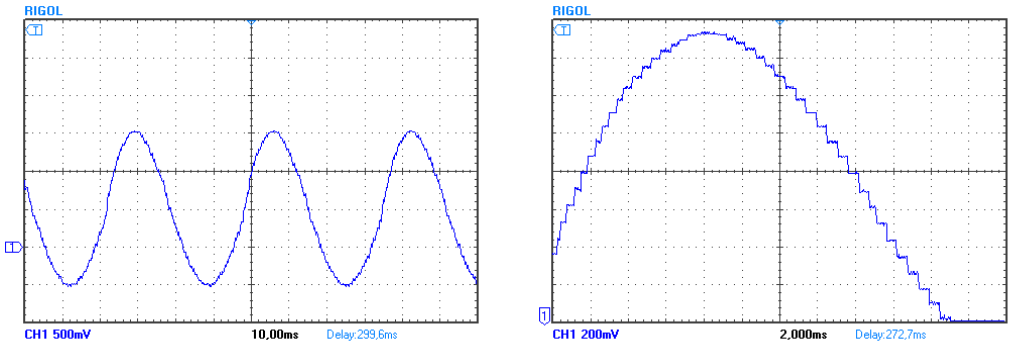


Рисунок 4.2 – Осциллограммы работы алгоритма

Обратите внимание на ступенчатую структуру синусоиды. Это происходит из-за того, что шаг изменения аргумента синуса 5 градусов, что слишком велико для достижения гладкости синусоиды.

Задания:

Задание 1

Вывести на осциллограф симметричную пилообразную функцию (рисунок 4.3).

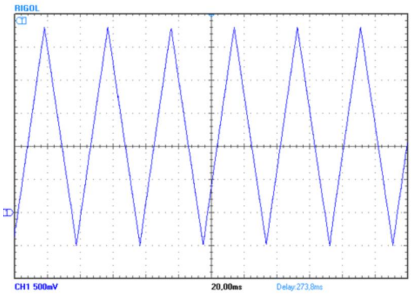


Рисунок 4.3 – Симметричная пилообразная функция

Задание 2

Вывести на осциллограф параболу в диапазоне . В промежутках между периодичным выводами парабол, поддерживать значение напряжения на уровне 3В. Пример результирующей осциллограммы представлен на рисунке 4.4.

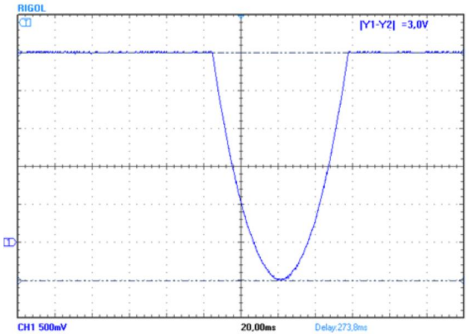


Рисунок 4.4 – Парабола

Приложение:

Полный текст программы

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_dac.h>
#include <math.h>

#define PI 3.14159265
#define MAX 3 /*Volt*/
#define SCALE (0xFFF * MAX) / 3.3

PORT_InitTypeDef PORTEInit;
void DACPortInit(){
        PORT_StructInit(&PORTEInit); //Load defaults
              RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTE, ENABLE);
              PORTEInit.PORT_Pin  = PORT_Pin_9;
              PORTEInit.PORT_OE   = PORT_OE_OUT;
              PORTEInit.PORT_MODE = PORT_MODE_ANALOG;
  PORT_Init(MDR_PORTE, &PORTEInit);
}

void DACInit(){
        RST_CLK_PCLKcmd(RST_CLK_PCLK_DAC, ENABLE);
        DAC1_Init(DAC1_AVCC);
        DAC1_Cmd(ENABLE);
}

float a;
int i; //4 task 2
int main() {
        RST_CLK_PCLKcmd(RST_CLK_PCLK_RST_CLK, ENABLE);
        DACPortInit();
        DACInit();

        while(1){
               //Jumper to EXT_CON!!!

              //* //Base part
              for (a=0; a<360; a+=5)
                  DAC1_SetData((sinf(a*PI/180)*SCALE + SCALE)/2);
              //*/

              /* //Task 1
              for (a=0; a<0xFFF; a+=5) DAC1_SetData(a);
              for (a=0xFFF-1; a>1; a-=5) DAC1_SetData(a);
              //*/

              /* //Task 2
              for (a=-0x7FF; a<0; a+=5)
                  DAC1_SetData(a*a/(0x7FF*0x7FF)*SCALE);
              for (a=0; a<0x7FF; a+=5) DAC1_SetData(a*a/(0x7FF*0x7FF)*SCALE);
              DAC1_SetData(SCALE);
              for (i=0; i<0xFFFF; i++);
              //*/
        }
}
```

Лабораторная работа № 5. Изучение аналого-цифрового преобразователя

Цель работы:

Изучение основных особенностей работы с аналого-цифровым преобразователем (АЦП) и работа со встроенным отладчиком при программировании для микроконтроллеров (МК) ARM.

Приборы и материалы:

1. Отладочная плата MDR1986VE91T Rev 4
2. Программатор J-Link ARM
3. Блок питания 5В, 1.4А
4. ПК с установленной средой программирования Keil uVision
5. Цифровой вольтметр

Порядок работы:

1. Собрать аппаратную часть по рекомендациям лабораторной работы № 2
2. Открыть проект MDRProject в среде программирования Keil uVision.
3. Подключить в менеджере Manage Run-Time Environment к проекту библиотеки ADC, PORT, RST_CLK необходимые для работы с АЦП.
4. Очистить содержимое файла *main.c* и добавить ссылки на подключенные библиотеки:

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_adc.h>
#include <stdbool.h>
```

5. Добавить в файл исходного кода функцию **ADCInit()**, в которой произвести инициализацию седьмого канала АЦП1 (на отладочной плате к этому каналу подключено переменное сопротивление).

```
ADC_InitTypeDef ADC;    //Общая инициализацинная структура подсистемы АЦП
ADCx_InitTypeDef ADC1;  //Инициализацинная структура для АЦП1

void ADCInit(){
       //Подача тактования на процессор и АЦП
       RST_CLK_PCLKcmd(RST_CLK_PCLK_RST_CLK | RST_CLK_PCLK_ADC, ENABLE);

       ADC_StructInit(&ADC);  //Заполнение структуры умолч. значениями
       ADC_Init(&ADC);        //Инициализация

       ADCx_StructInit(&ADC1);
       ADC1.ADC_ChannelNumber = ADC_CH_ADC7; //Выбор седьмого канала
       ADC1_Init(&ADC1);

       //Инициализация прерываний АЦП
       NVIC_EnableIRQ(ADC_IRQn);
       NVIC_SetPriority(ADC_IRQn, 0);

       //Включение прерываний по окончанию преобразования
       ADC1_ITConfig(ADCx_IT_END_OF_CONVERSION, ENABLE);

       //Включение АЦП1
       ADC1_Cmd(ENABLE);
}
```

6. Добавить в файл исходного кода обработчик прерывания по окончанию аналого-цифрового преобразования.

```
bool conInProgress;      //Флаг «в процессе преобразования»
unsigned int rawResult;  //Необработанный результат
unsigned char channel;   //Номер канала
float result;            //Результат в вольтах

void ADC_IRQHandler() {  //Обработчик прерываний АЦП
    //Проверка что причина прерывания соответствует концу преобразования
    if(ADC_GetITStatus(ADC1_IT_END_OF_CONVERSION)){

       rawResult = ADC1_GetResult(); //Получение результата
       channel = (rawResult & 0x1F0000) >> 16; //Сохранение номера канала
       rawResult &= 0xFFF;    //Удаление номера канала из результата
```

```
        //Преобразование результата в вольты
        result = (float)rawResult / (float)SCALE;

        conInProgress = false; //Очистка флага «в процессе преобр-я»
        NVIC_ClearPendingIRQ(ADC_IRQn); //Очистка флага прерывания
    }
}
```

Как видно из кода, результат содержит в себе не только преобразованное значение напряжения на канале, но и номер этого канала, который нужно убрать из результата. На странице 323 спецификации имеется таблица 301 (рисунок 5.1), иллюстрирующая это.

MDR_ADC->ADCx_RESULT

Таблица 301 – Регистр ADCx_RESULT

Номер	31...21	20…16	15…12	11…0
Доступ	U	RO	U	RO
Сброс	0	0	0	0
	-	**CHANNEL [4:0]**	-	**RESULT [11:0]**

Рисунок 5.1 – Структура регистра с результатом преобразования

С помощью встроенного в Windows калькулятора в режиме программиста выясняем, что битовая маска, обнуляющая все биты, кроме блока с 20 по 16 включительно, имеет вид, отображенный в шестнадцатеричном формате на рисунке 5.2 (0x1F0000).

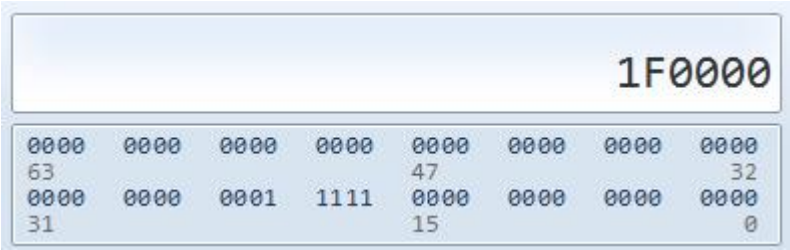


Рисунок 5.2 – Битовая маска для вычисления номера канала

Конъюнктивно применяя данную битовую маску к результату и смещая его на 16 разрядов вправо (для избавления от 16 значимых нулей в конце), выделяем номер канала, который теперь можно сохранить в отдельную переменную. Для выделения результата преобразования применим битовую маску 0xFFF (12 единиц в двоичном представлении). Далее необходимо преобразовать полученный результат из цифрового кода, пропорционального напряжению, в вольты. Для этого результат необходимо поделить на коэффициент SCALE, значение которого предстоит определить экспериментальным путём.

7. Последним штрихом является написание функции **main()**, с которой начнётся исполнение программы.

```
int i;      //Счетчик для задержки циклом
int main() {
    ADCInit();
    while(1){
       for(i = 0xFFFF; i > 0; i--);  //Задержка циклом (плохой вариант)
       if (!conInProgress){          //Не выполняется ли преобразование?
           ADC1_Start();             //Начать преобразование!
           conInProgress = true;     //Преобразование выполняется
       }
    }
}
```

Таким образом, в главном цикле через определенный промежуток времени запускается аналого-цифровое преобразование на седьмом канале первого АЦП. По окончании преобразования, вызывается соответствующий обработчик прерывания, в котором результат преобразования заносится в переменную result. Чтобы данный код заработал, не хватает константы SCALE, и чтобы найти такое ее значение, с помощью которого можно получить результат в вольтах, зададим ее равной единице.

```
#define SCALE 1
```

8. Подать питание на плату и загрузить программу в микроконтроллер по рекомендациям лабораторной работы №2
9. Поставить перемычку ADC_INP_SEL в положение TRIM для подключения переменного сопротивления в качестве источника сигнала для АЦП (рисунок 5.3).

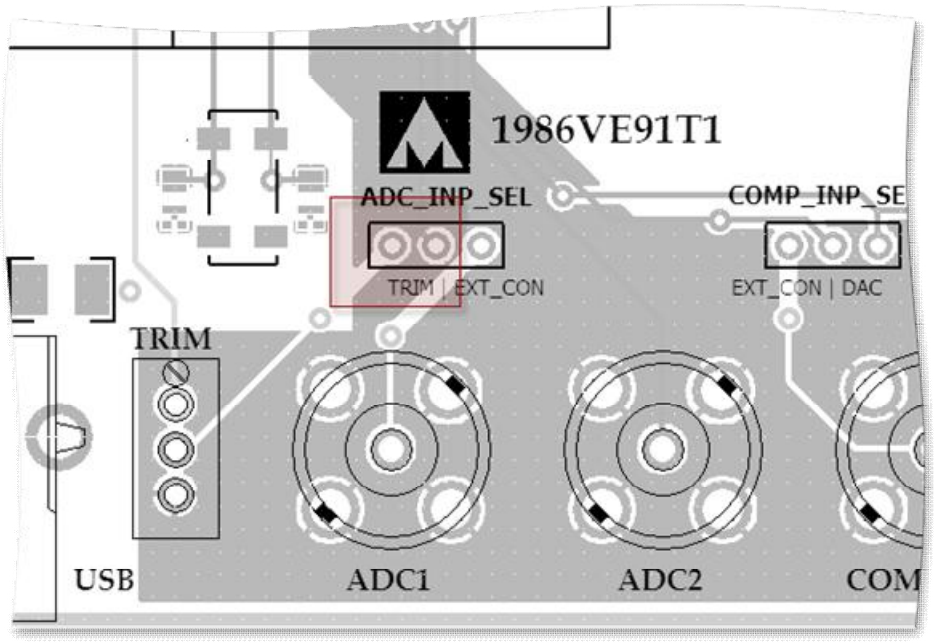


Рисунок 5.3 – Блок аналогового входа

10. Подключить плюсовой вход вольтметра к среднему контакту перемычки (можно замкнуть перемычку в положение TRIM с помощью щупа вольтметра)
11. Подключить минусовой контакт вольтметра к любому общему проводу отладочной платы (удобно использовать любое крепёжное отверстие по краям платы)

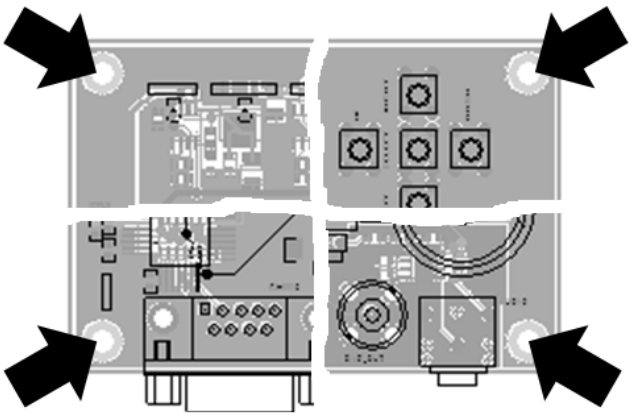


Рисунок 5.4 – Крепежные отверстия

12. Запустить сеанс отладки в программе Keil uVision (Ctrl+F5 или кнопка на панели инструментов)
13. Добавить переменную **result** в **Watch 1** (правый клик по переменной в коде | *Add 'result' to… | Watch 1*) и запустить код на исполнение (F5 или кнопка на панели инструментов)
14. Вращать вал переменного сопротивления (**TRIM** на рисунке 6.4), пока напряжение на АЦП не станет равным 2 вольта.
15. Считать из окна **Watch 1** отладчика значение переменной **result** при известном напряжении на АЦП (рисунок 5.5).
16. Поделить полученное цифровое значение на напряжение, которому оно соответствует.
18. Остановить отладку.
17. Полученный коэффициент вписать в код в качестве константы SCALE.
19. Загрузить код с новым коэффициентом SCALE в МК.
20. Запустить сеанс отладки и убедиться, что переменная result отображает значение, совпадающее с показаниями вольтметра.

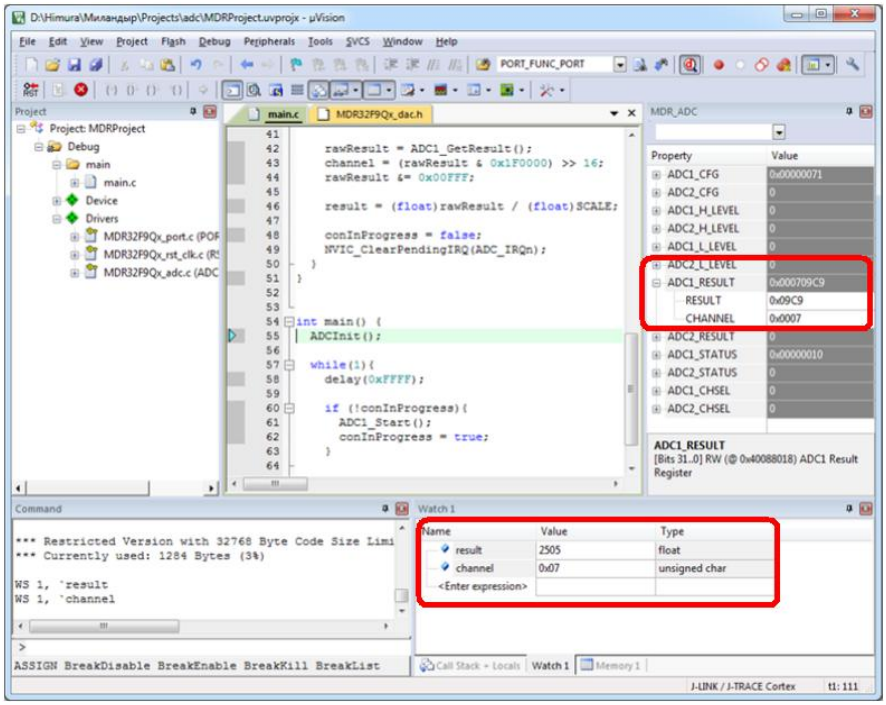


Рисунок 5.5 – Окно Keil uVision в процессе отладки

В окне среды Keil uVision на рисунке 5.5 также открыто окно регистров **MDR_ADC**. Данное окно можно открыть в меню *System Viewer Windows* ( ) на панели инструментов. При анализе содержимого регистра **ADC1_RESULT**, можно подтвердить правильность его описания (рисунок 6.2) и даже понять его структуру при отсутствии описания. Отладка является мощнейшим инструментом разработки и помогает увидеть любые ошибки времени выполнения легко и наглядно.

Задание:

Используя знания, полученные в лабораторной работе № 5, создать цифровой преобразователь сигнала. При входном уровне сигнала (от переменного сопротивления) от 0 до 1 вольта, он должен выдавать сигнал от 2 до 3 вольт аналогичной формы.

Приложение:

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_adc.h>
#include <stdbool.h>

#define delay(T) for(i = T; i > 0; i--)
int i;

#define SCALE 1252.5

ADC_InitTypeDef ADC;
ADCx_InitTypeDef ADC1;

void ADCInit(){
	RST_CLK_PCLKcmd(RST_CLK_PCLK_RST_CLK | RST_CLK_PCLK_ADC, ENABLE);
	ADC_StructInit(&ADC);
	ADC_Init(&ADC);
	ADCx_StructInit(&ADC1);
	ADC1.ADC_ChannelNumber = ADC_CH_ADC7; //Switch to TRIM!!!
	ADC1_Init(&ADC1);
	//Int
	NVIC_EnableIRQ(ADC_IRQn);
	NVIC_SetPriority(ADC_IRQn, 0);
	ADC1_ITConfig(ADCx_IT_END_OF_CONVERSION, ENABLE);
	ADC1_Cmd(ENABLE);
}

bool conInProgress;
unsigned int rawResult;
unsigned char channel;
float result;

void ADC_IRQHandler() {
	if(ADC_GetITStatus(ADC1_IT_END_OF_CONVERSION)){
		rawResult = ADC1_GetResult();
		channel = (rawResult & 0x1F0000) >> 16;
		rawResult &= 0x00FFF;
		result = (float)rawResult / (float)SCALE;
		conInProgress = false;
		NVIC_ClearPendingIRQ(ADC_IRQn);
	}
}
int main() {
	ADCInit();
	while(1){
		delay(0xFFFF);
		if (!conInProgress){
			ADC1_Start();
			conInProgress = true;
		}
	}
}
```

Лабораторная работа №6. Изучение схемы тактировая. Изменение тактовой частоты.

Цель работы:

Изучение модуля RST_CLK микроконтроллера 1986BE91T, и способов его программирования в среде Keil uVision5, написание программы для изучения тактирования от различных источников, а также изменения частоты тактирования.

Приборы и материалы:

1. Отладочная плата MDR1986VE91T Rev 4
2. Программатор J-Link ARM
3. Блок питания 5В, 1.4А
4. ПК с установленной средой программирования Keil uVision
5. Осциллограф

Порядок работы:

1. Собрать аппаратную часть по рекомендациям лабораторной работы № 2
2. Открыть проект MDRProject в среде программирования Keil uVision.
3. Подключить к проекту библиотеку RST_CLK в менеджере Manage Run-Time Environment, необходимую для работы схемы тактирования.
4. Выполнить работы согласно Лабораторной работе №3 (про таймеры) загрузить программу в микроконтроллер.
5. Наблюдать переключение светодиодов стенда с частотой около 1 секунды. Подключить осциллограф к контакту 29 разъёма X32.1 (Нулевой провод согласно рис.5.5). Получившаяся осциллограмма показана рис. 6.1.

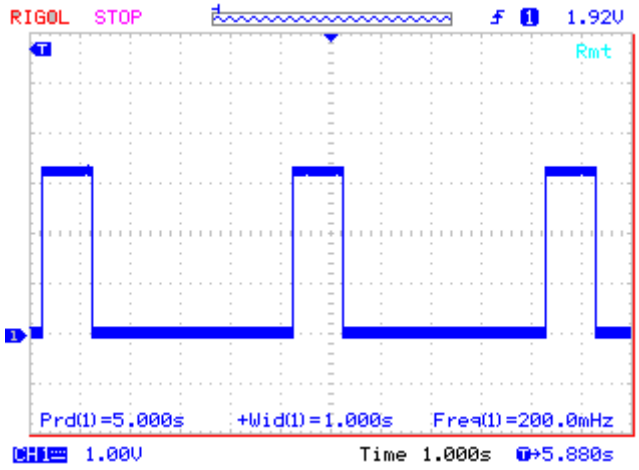


Рисунок 6.1 Осциллограмма напряжения на светодиоде.

Как видим, длительность импульса составляет 1 с.

6. Добавить перед while(1) в функции main() следующий код

```
/* Включение HSE осциллятора (внешнего кварцевого резонатора)*/
  RST_CLK_HSEconfig(RST_CLK_HSE_ON);
  if (RST_CLK_HSEstatus() == SUCCESS)  /* Если HSE осциллятор включился и прошел текст*/
  {
  // Выбор HSE осциллятора в качестве источника тактовых импульсов для CPU_PLL
  // и установка умножителя тактовой частоты CPU_PLL равного 7
  // Частота внешнего кварца равна 8 МГц Максимальная частота процессора 80 МГц ,
  // RST_CLK_CPU_PLLconfig ( Источник тактирования PLL, Коэффициент умножения=9+1 );
  // Коэффициент умножения=0 соответствует умножение на 1.

  RST_CLK_CPU_PLLconfig ( RST_CLK_CPU_PLLsrcHSEdiv1, 9 );
```

```
/* Включение схемы PLL*/
RST_CLK_CPU_PLLcmd(ENABLE);
if (RST_CLK_CPU_PLLstatus() == SUCCESS)        //Если включение CPU_PLL прошло успешно
  {
     /* Установка CPU_C3_prescaler = 2 */
     RST_CLK_CPUclkPrescaler(RST_CLK_CPUclkDIV2);
     /* Установка CPU_C2_SEL от CPU_PLL выхода вместо CPU_C1 такта*/
     RST_CLK_CPU_PLLuse(ENABLE);
     /* Выбор CPU_C3 такта на мультиплексоре тактовых импульсов микропроцессора (CPU clock
     MUX) */
     RST_CLK_CPUclkSelection(RST_CLK_CPUclkCPU_C3);
  }
    else         /* блок CPU_PLL не включился*/
  {while(1);}
}
  else           /* кварцевый резонатор HSE не включился */
{while(1);}
```

7. Перекомпилировать и загрузить приложение на микроконтроллер.

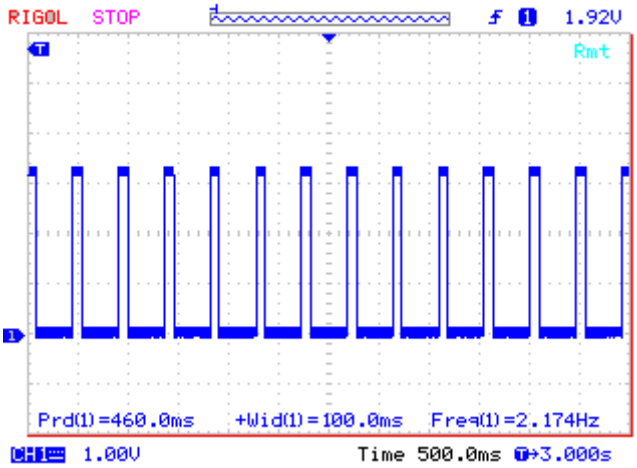


Рисунок 6.2. Осциллограмма напряжения после изменения тактовой частоты

Как видим на осциллографе, частота импульсов увеличилась в 10 раз, визуально светодиоды стенда переключаются с гораздо большей частотой, то есть микропроцессор работает на частоте 8*10=80МГц.

8. Выполнить индивидуальное задание преподавателя.

Сведения для выполнения

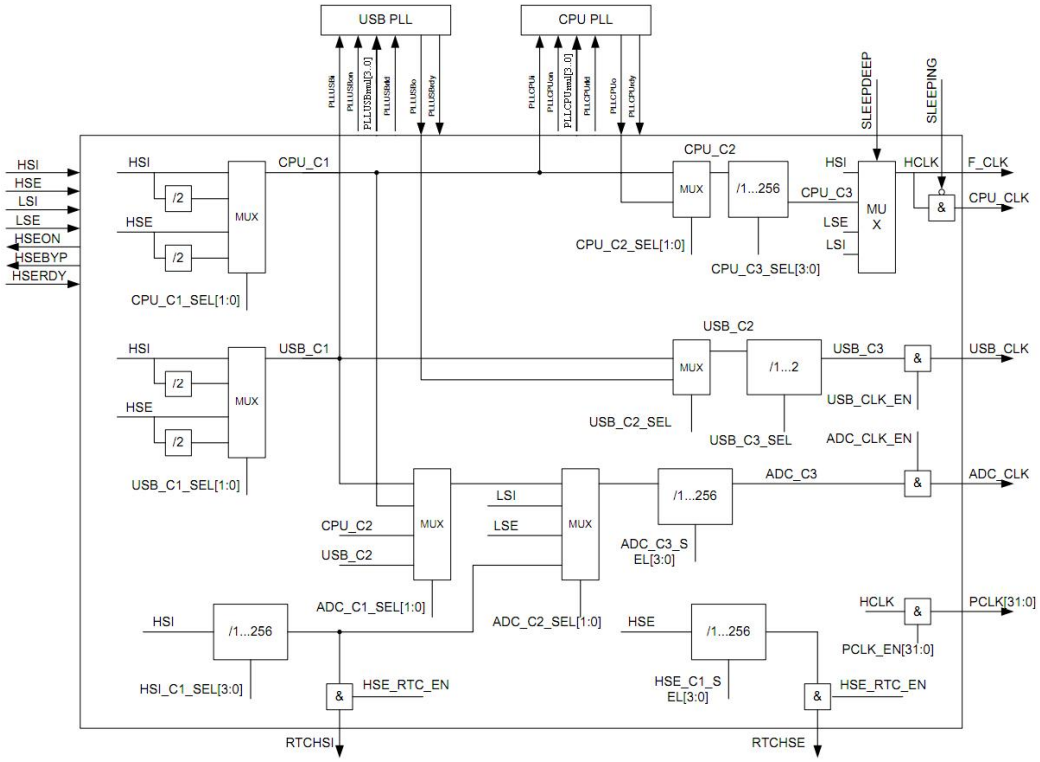


Рис 6.3 Структурная схема тактирования микроконтроллера

Рассмотрим основные элементы системы тактирования:

1 Встроенный RC генератор HSI

Генератор HSI вырабатывает тактовую частоту 8 МГц. Генератор автоматически запускается при появлении питания UCC и при выходе в нормальный режим работы вырабатывает сигнал HSIRDY в регистре батарейного домена BKP_REG_0F. Первоначально процессорное ядро запускается на тактовой частоте HSI. При дальнейшей работе генератор HSI может быть отключен при помощи сигнала HSION в регистре BKP_REG_0F. Также генератор может быть подстроен при помощи сигнала HSITRIM в регистре BKP_REG_0F.

2 Встроенный RC генератор LSI

Генератор LSI вырабатывает тактовую частоту 40 кГц. Генератор автоматически запускается при появлении питания UCC и при выходе в нормальный режим работы вырабатывает сигнал LSIRDY в регистре BKP_REG_0F. Первоначально тактовая частота генератора LSI используется для формирования дополнительной задержки tpor. При дальнейшей работе генератор LSI может быть отключен при помощи сигнала LSION в регистре BKP_REG_0F.

3 Внешний генератор HSE

Генератор HSE предназначен для выработки тактовой частоты 2.16 МГц с помощью внешнего резонатора. Генератор запускается при появлении питания UCC и сигнала разрешения HSEON в регистре HS_CONTROL. При выходе в нормальный режим работы вырабатывает сигнал HSERDY в регистре CLOCK_STATUS. Также этот генератор может работать в режиме HSEBYP, когда входная тактовая частота с входа OSC_IN проходит напрямую на выход HSE. Выход OSC_OUT находится в этом режиме в третьем состоянии.

4 Внешний генератор LSE

Генератор LSE предназначен для выработки тактовой частоты 32 КГц с помощью внешнего резонатора. Генератор запускается при появлении питания BDUCC и сигнала разрешения LSEON в регистре BKP_REG_0F. При выходе в нормальный режим работы вырабатывает сигнал LSERDY в регистре BKP_REG_0F. Также осциллятор может работать в режиме LSEBYP, когда входная тактовая частота с входа OSC_IN32 проходит напрямую на выход LSE. Выход OSC_OUT32 находится в этом режиме третьем состоянии. Так как генератор LSE питается от напряжения питания BDUCC и его регистр управления BKP_REG_0F расположен в батарейном домене, то генератор может продолжать работать при пропадании основного питания UCC. Генератор LSE используется для работы часов реального времени.

5 Встроенный блок умножения системной тактовой частоты

Блок умножения позволяет провести умножение входной тактовой частоты на коэффициент от 2 до 16, задаваемый на входе PLLCPUMUL[3:0] в регистре PLL_CONTROL. Входная частота блока умножителя должна быть в диапазоне 2…16 МГц, выходная - до 100 МГц. При выходе блока умножителя тактовой частоты в расчетный режим вырабатывается сигнал PLLCPURDY в регистре CLOCK_STATUS. Блок включается с помощью сигнала PLLCPUON в регистре PLL_CONTROL. Выходная частота используется как основная частота процессора и периферии.

6 Встроенный блок умножения USB тактовой частоты

Блок умножения позволяет провести умножение входной тактовой частоты на коэффициент от 2 до 16, задаваемый на входе PLLUSBMUL[3:0] в регистре PLL_CONTROL. Входная частота блока умножителя должна быть в диапазоне 2…16 МГц, выходная должна составлять 48 МГц. При выходе блока умножителя тактовой частоты в расчетный режим вырабатывается сигнал PLLUSBRDY в регистре CLOCK_STATUS. Блок включается с помощью сигнала PLLUSBON в регистре PLL_CONTROL. Выходная частота используется как основная частота протокольной части USB интерфейса. Управление тактовыми частотами ведется через периферийный блок RST_CLK. При включении питания микроконтроллер запускается на частоте HSI генератора. Выдача тактовых сигналов синхронизации для всех периферийных блоков, кроме RST_CLK, отключена. Для начала работы с нужным периферийным блоком необходимо включить его тактовую частоту в регистре PER_CLOCK. Некоторые контроллеры интерфейсов (UART, CAN, USB, Таймеры) могут работать на частотах, отличных от частоты процессорного ядра, поэтому в соответствующих регистрах (UART_CLOCK, CAN_CLOCK, USB_CLOCK, TIM_CLOCK) могут быть заданы их скорости работы. Для изменения тактовой частоты ядра можно перейти на другой генератор и/или воспользоваться блоком умножения тактовой частоты. Для корректной смены тактовой частоты сначала должны быть сформированы необходимые тактовые частоты и затем осуществлено переключение на них на соответствующих мультиплексорах, управляемых регистрами CPU_CLOCK и USB_CLOCK.

7 Делители частоты и мультиплексоры

Эти элементы позволяют коммутировать внутренние цепи схемы тактирования. Коммутация позволяет создать гибкую конфигурацию тактирования всех устройств микроконтроллера.

Для включения тактирования необходимо:

- разрешить работу тактового генератора;
- проверить правильность включения осциллятора или генератора, и прошел ли он тест;
- выбрать его в качестве источника тактовых импульсов схемы PLL или микроконтроллера в зависимости от той конфигурации, которая Вам необходима.

Для работы со схемой тактирования предусмотрены процедуры, объявленные в файле MDR32F9Qx_rst_clk.h., они включают в себя процедуры инициализации настройки и проверки работоспособности устройств схемы тактирования.

Задание:

Задание 1.
Измените частоту работы микропроцессора на 4Мгц, 8МГц, или другую по заданию преподавателя.

Задание 2.
Напишите программу, работающую на частоте генератора LSI или LSE по заданию преподавателя.

Приложение:
Текст программы

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_timer.h>

PORT_InitTypeDef PORTDInit;
//Инициализация портов ввода вывода
void PortsInit(){
  PORT_StructInit(&PORTDInit); //Load defaults
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTD, ENABLE);
  PORTDInit.PORT_Pin   = PORT_Pin_10 | PORT_Pin_11 | PORT_Pin_12 | PORT_Pin_13 | PORT_Pin_14;
  PORTDInit.PORT_OE    = PORT_OE_OUT;
  PORTDInit.PORT_MODE  = PORT_MODE_DIGITAL;
  PORTDInit.PORT_SPEED = PORT_SPEED_SLOW;
  PORT_Init(MDR_PORTD, &PORTDInit);
}
uint8_t cur_i;
//Процедура мигания светодиодами
void NextLED(){
  switch(cur_i++ % 5) {
    case 0:
       PORT_ResetBits(MDR_PORTD, PORT_Pin_14);
       PORT_SetBits(MDR_PORTD, PORT_Pin_10);
       break;
    case 1:
       PORT_ResetBits(MDR_PORTD, PORT_Pin_10);
       PORT_SetBits(MDR_PORTD, PORT_Pin_11);
       break;
    case 2:
       PORT_ResetBits(MDR_PORTD, PORT_Pin_11);
       PORT_SetBits(MDR_PORTD, PORT_Pin_12);
       break;
    case 3:
       PORT_ResetBits(MDR_PORTD, PORT_Pin_12);
       PORT_SetBits(MDR_PORTD, PORT_Pin_13);
       break;
    case 4:
       PORT_ResetBits(MDR_PORTD, PORT_Pin_13);
       PORT_SetBits(MDR_PORTD, PORT_Pin_14);
       break;
  }
}

TIMER_CntInitTypeDef TIM1Init;

// Процедура инициализации таймера
// при частоте процессора 8 МГц обеспечивает вызов процедуры обработки прерывания 1 раз в
// секунду

void TimerInit(){
  RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER1, ENABLE);
  TIMER_BRGInit(MDR_TIMER1, TIMER_HCLKdiv1);
  TIMER_CntStructInit(&TIM1Init);    //Load defaults
  TIM1Init.TIMER_Prescaler        = 8000;
  TIM1Init.TIMER_Period           = 1000;
  TIMER_CntInit(MDR_TIMER1, &TIM1Init);

// Инициализация прерываний таймера
  NVIC_EnableIRQ(Timer1_IRQn);
  NVIC_SetPriority(Timer1_IRQn, 0);
```

```
  TIMER_ITConfig(MDR_TIMER1, TIMER_STATUS_CNT_ZERO, ENABLE);

  TIMER_Cmd(MDR_TIMER1, ENABLE);
}

//Обработчик прерывания таймера
void Timer1_IRQHandler() {
  if(TIMER_GetITStatus(MDR_TIMER1, TIMER_STATUS_CNT_ZERO)){
    NextLED();
    TIMER_ClearITPendingBit(MDR_TIMER1, TIMER_STATUS_CNT_ZERO);
  }
}

int main() {
  PortsInit();
  TimerInit();
  cur_i = 0;

/* Включение HSE осциллятора (внешнего кварцевого резонатора)*/
  RST_CLK_HSEconfig(RST_CLK_HSE_ON);
  if (RST_CLK_HSEstatus() == SUCCESS)  /* Если HSE осциллятор включился и прошел текст*/
  {
  // Выбор HSE осциллятора в качестве источника тактовых импульсов для CPU_PLL
  // и установка умножителя тактовой частоты CPU_PLL равного 7
  // Частота внешнего кварца равна 8 МГц Максимальная частота процессора 80 МГц ,
  // RST_CLK_CPU_PLLconfig ( Источник тактирования PLL, Коэффициент умножения 10=9+1);

  RST_CLK_CPU_PLLconfig ( RST_CLK_CPU_PLLsrcHSEdiv1, 9 );
  /* Включение схемы PLL*/
  RST_CLK_CPU_PLLcmd(ENABLE);
  if (RST_CLK_CPU_PLLstatus() == SUCCESS)       //Если включение CPU_PLL прошло успешно
    {
       /* Установка CPU_C3_prescaler = 2 */
       RST_CLK_CPUclkPrescaler(RST_CLK_CPUclkDIV2);
       /* Установка CPU_C2_SEL от CPU_PLL выхода вместо CPU_C1 такта*/
       RST_CLK_CPU_PLLuse(ENABLE);
       /* Выбор CPU_C3 такта на мультиплексоре тактовых импульсов микропроцессора (CPU clock
       MUX) */
       RST_CLK_CPUclkSelection(RST_CLK_CPUclkCPU_C3);
    }
      else        /* блок CPU_PLL не включился*/
    {while(1);}
  }
    else          /* кварцевый резонатор HSE не включился */
  {while(1);}
while(1){
               // Основной пустой цикл
       }
}
```

Лабораторная работа №7. Изучение модуля UART

Цель работы:

Изучение модуля UART микроконтроллера 1986BE91T и способов его программирования в среде Keil uVision5, написание простейших программ, изучение .

Приборы и материалы:

1. Отладочная плата MDR1986VE91T Rev 4
2. Программатор J-Link ARM
3. Блок питания 5В, 1.4А
4. ПК с установленной средой программирования Keil uVision
5. Цифровой осциллограф
6. Нуль модемный кабель для подключения по интерфейсу RS232C
7. Свободная программа Putty.exe.

Порядок работы:

1. Собрать лабораторную установку согласно рис. 7.1

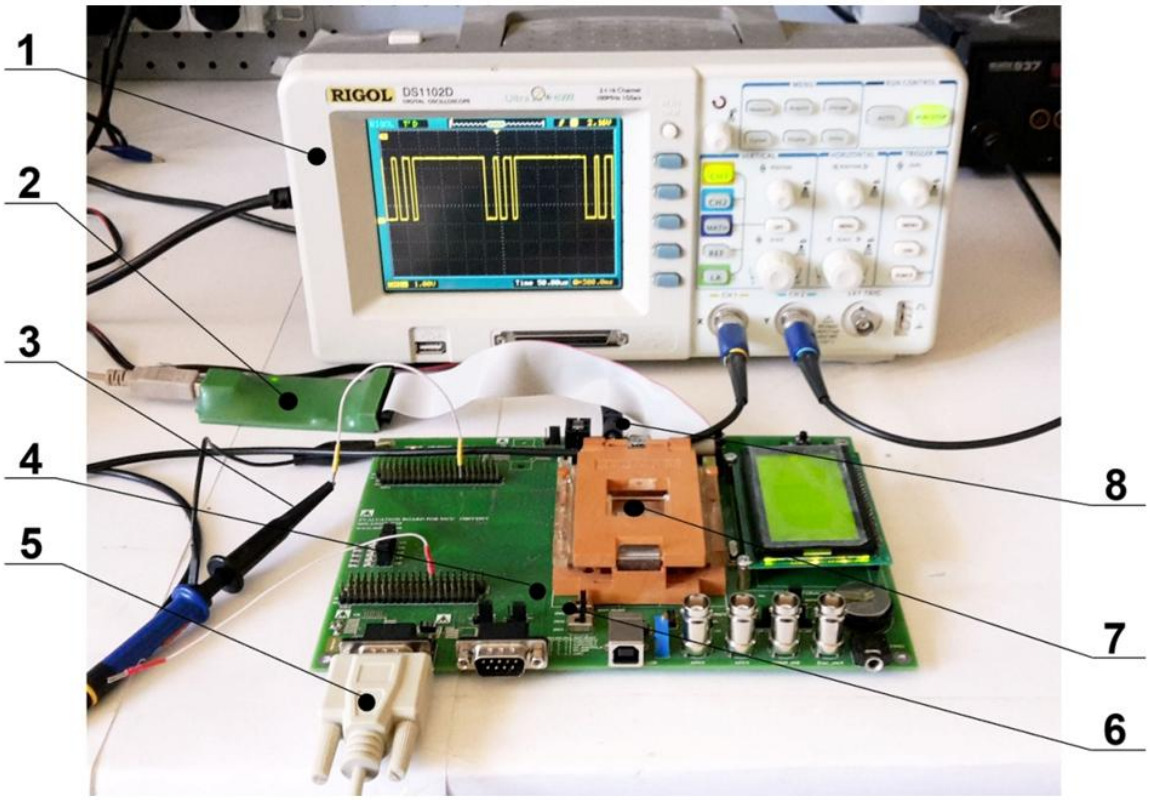


Рисунок 7.1 Внешний вид лабораторной установки: 1- осциллограф, 2-программтатор TP-LINK (J-LINK), 3-щуп осциллографа, 4-демонстрационная плата на базе 1986BE91T, 5-нуль модемный кабель для подключения по интерфейсу RS-232C, 6-микропереключатели для выбора режима загрузки, 7-микроконтроллер 1986BE91T в контактирующем устройстве, 8-разъем питания.

2. Нульмодемный кабель подключить к порту RS 232C ЭВМ (9 контактный разъем на задней панели).
3. Установить программу PuTTY [5]
http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

4. Подключить программатор к разъему JTAG_B
5. Создать проект в среде Keil uVision5 или использовать MDRproject
6. Подключить библиотеки PORT, RST_CLK, UART в менеджере Manage Run-Time Environment.
7. Заменить текст main.c на текст программы (см. Приложение). Данная программа принимает один байт данных по RS232 и тот же, принятый байт данных, выдает обратно на компьютер.
8. Скомпилировать и загрузить проект в микроконтроллер
9. Запустить программу Putty, установить настройки: *Serial line* – COM№ порт к которому подключен микроконтроллер (Возможные варианты № можно узнать в диспетчере устройств Windows); *Speed – 115200*; *Connection type - Serial* как показано на рис. 7.2 и нажать Open

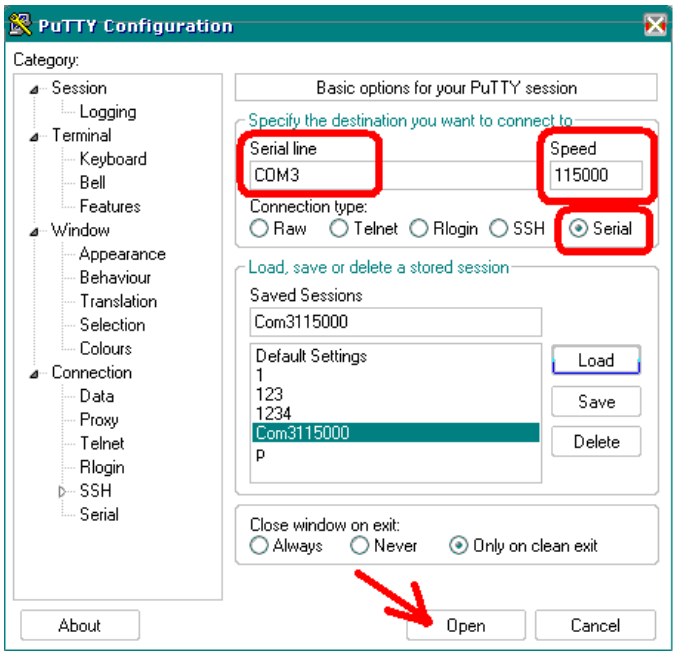


Рисунок 7.2. Окно программы Putty

10. В появившемся окне рис 7.3 ввести произвольный текст.

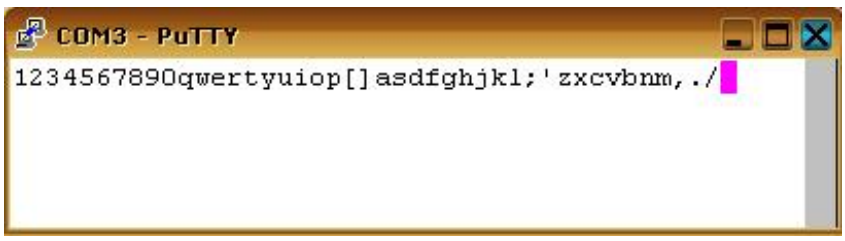


Рисунок 7.3. Окно программы Putty

Как видим, те данные, которые вводятся с клавиатуры, отображаются в окне Putty, что говорит о двунаправленной передаче данных между контроллером и ЭВМ.
11. Подключить осциллограф к ножкам 13 и 14 разъема X32.2 зажать любую клавишу на клавиатуре компьютера и добиться на экране осциллографа картинки содержащей процесс передачи байт как показано на рис 7.4.

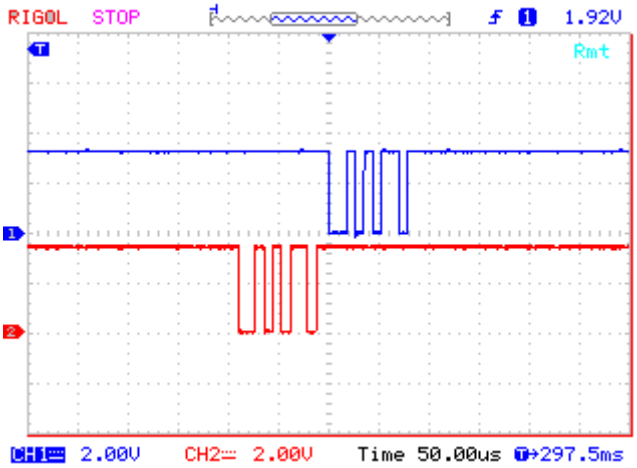


Рисунок 7.4 Прием байта в микроконтроллер (внизу), обратная передача данных на ЭВМ (вверху).

12. Выполнить индивидуальное задание преподавателя.

Сведения для выполнения:

Модуль универсального асинхронного приемопередатчика (UART – Universal Synchronous Asynchronous Receiver Transmitter) представляет собой периферийное устройство микроконтроллера. В состав контроллера включен кодек (ENDEC – ENcoder/DEcoder) последовательного интерфейса инфракрасной (ИК) передачи данных в соответствии с протоколом SIR (SIR – Serial Infra Red) ассоциации Infrared Data Association (IrDA).

Основные характеристики модуля UART

Может быть запрограммирован для использования как в качестве универсального асинхронного приемопередатчика, так и для инфракрасного обмена данными (SIR). Содержит независимые буферы приема (16x12) и передачи (16x8) типа FIFO (First In First Out – первый вошел, первый вышел), что позволяет снизить интенсивность прерываний центрального процессора. Программное отключение FIFO позволяет ограничить размер буфера одним байтом.

Программное управление скоростью обмена. Обеспечивается возможность деления тактовой частоты опорного генератора в диапазоне (1x16 – 65535x16). Допускается использование нецелых коэффициентов деления частоты, что позволяет использовать любой опорный генератор с частотой более 3.6864 МГц максимальная скорость обмена:

- в режиме UART – до 921600 бит/с;
- в режиме IrDA – до 460800 бит/с;
- в режиме IrDA с пониженным энергопотреблением – до 115200 бит/с.

Поддержка стандартных элементов асинхронного протокола связи – стартового и стопового бит, а так же бита контроля четности, которые добавляются перед передачей и удаляются после приема.

Независимое маскирование прерываний от буфера FIFO передатчика, буфера FIFO приемника, по таймауту приемника, по изменению линий состояния модема, а также в случае обнаружения ошибки.

- Поддержка прямого доступа к памяти.
- Обнаружение ложных стартовых бит.
- Формирование и обнаружения сигнала разрыва линии.
- Поддержка функция управления модемом (линии CTS, DCD, DSR, RTS, DTR и RI).

Возможность организации аппаратного управления потоком данных.

Полностью программируемый асинхронный последовательный интерфейс с характеристиками:

- данные длиной 5, 6, 7 или 8 бит;
- формирование и контроль четности (проверочный бит выставляется по четности, нечетности, имеет фиксированное значение, либо не передается);
- формирование 1 или 2 стоповых бит.

Кодек ИУ обмена данными IrDA SIR обеспечивает:

- программный выбор обмена данными по линиям асинхронного приемопередатчика либо кодека ИК связи IrDA SIR;
- поддержку функционирования с информационной скоростью до 115200 бит/с в режиме полудуплекса;
- поддержку длительности бит для нормального режима (3/16) и для режима пониженного энергопотребления (1.41 – 2.23 мкс).

Наличие идентификационного регистра, однозначно идентифицирующего модуль, что позволяет операционной системе выполнять автоматическую конфигурацию.

Работа в режиме UART

Для инициализации UART в библиотеке предусмотрена структура UART_InitTypeDef;

```
typedef  struct
{
  uint32_t  UART_BaudRate;            //Задается частота передачи данных
  uint16_t  UART_WordLength;        //Задается число передаваемых бит данных
  uint16_t  UART_StopBits;              //Задается число стоповых битов
  uint16_t  UART_Parity;                 //Задается контрольна четность или на нечетность
  uint16_t  UART_FIFOMode;          //Задается режим работы FiFO буфера (включен или выключен)
  uint16_t  UART_HardwareFlowControl; //Разрешение аппаратного контроля за линиями интерфейса RS232C
}UART_InitTypeDef;
```

В программе, приведенной в приложении, частота передачи 115200, длина 8 бит, 1 стоп бит, без контроля четности, выключенный Fifo буфер, аппаратный контроль линий интерфейса.

Прием данных по интерфейсу и передача данных осуществляются с использованием системы прерываний. Процедура обработки прерываний

```
void UART2_IRQHandler(void)
```

Эта процедура вызывается при приеме данных по интерфейсу RS232C, а также по окончании передачи данных по интерфейсу. Перед вызовом процедуры необходимо разрешить процедуру обработки прерываний.

```
UART_ITConfig (MDR_UART2, UART_IT_RX, ENABLE);
```

Данная процедура разрешает вызов процедуры обработки прерывания UART2 по приему одного байта данных.

Задание:

Задание 1
Измените частоту передаваемых данных по UART,

Задание 2
Напишите программу, которая в зависимости от принятого байта данных зажигает один из светодиодов, например, если это цифра, то загорается 1 светодиод, если русская маленькая буква, то второй светодиод, русская большая буква – третий светодиод, английская маленькая – четвертый, английская большая – пятый.

Приложение:

Текст программы
Файл *main.c* .

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_uart.h>

PORT_InitTypeDef PortInit;    // определение переменной для инициализации портов ввода вывода
UART_InitTypeDef UART_InitStructure; // определение переменной для инициализации UART

uint32_t uart2_IT_TX_flag = RESET;  // Флаг устанавливается после передачи одного байта
uint32_t uart2_IT_RX_flag = RESET;  // Флаг устанавливается после приема одного байта

void UART2_IRQHandler(void)
{
if (UART_GetITStatusMasked(MDR_UART2, UART_IT_RX) == SET)
        //проверка установки флага прерывания по окончании приема данных
  {
    UART_ClearITPendingBit(MDR_UART2, UART_IT_RX);//очистка флага прерывания
    uart2_IT_RX_flag = SET;   //установка флага передача данных завершена
  }
if (UART_GetITStatusMasked(MDR_UART2, UART_IT_TX) == SET)
        //проверка установки флага прерывания по окончании передачи данных
  {
    UART_ClearITPendingBit(MDR_UART2, UART_IT_TX);  //очистка флага прерывания
    uart2_IT_TX_flag = SET;   //установка флага передача данных завершена
  }
}

int main (void) {

    static uint8_t ReciveByte=0x00; //данные для приема
    // Включение HSE осциллятора (внешнего кварцевого резонатора) для обеспечения стабильной
    // частоты UART
    RST_CLK_HSEconfig(RST_CLK_HSE_ON);
    if (RST_CLK_HSEstatus() == SUCCESS){
        // Если HSE осциллятор включился и прошел текст
        // Выбор HSE осциллятора в качестве источника тактовых импульсов для CPU_PLL
        // и установка умножителя тактовой частоты CPU_PLL равного 8 = 7+1
        RST_CLK_CPU_PLLconfig(RST_CLK_CPU_PLLsrcHSEdiv1, 7);
        // Включение схемы PLL
        RST_CLK_CPU_PLLcmd(ENABLE);
        if (RST_CLK_HSEstatus() == SUCCESS){ //Если включение CPU_PLL прошло успешно
            RST_CLK_CPUclkPrescaler(RST_CLK_CPUclkDIV2); // Установка CPU_C3_prescaler = 2
            RST_CLK_CPU_PLLuse(ENABLE);
            // Установка CPU_C2_SEL от CPU_PLL выхода вместо CPU_C1 такта
            /* Выбор CPU_C3 такта на мультиплексоре тактовых импульсов микропроцессора (CPU
            clock MUX) */
            RST_CLK_CPUclkSelection(RST_CLK_CPUclkCPU_C3);
        }
        else while(1);// блок CPU_PLL не включился
    }
    else while(1); // кварцевый резонатор HSE не включился
    RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTF, ENABLE);  //Разрешение тактирования порта F
    // заполнение полей переменной  PortInit для обеспечения работы UART
    PortInit.PORT_PULL_UP = PORT_PULL_UP_OFF;
```

```
    PortInit.PORT_PULL_DOWN = PORT_PULL_DOWN_OFF;
    PortInit.PORT_PD_SHM = PORT_PD_SHM_OFF;
    PortInit.PORT_PD = PORT_PD_DRIVER;
    PortInit.PORT_GFEN = PORT_GFEN_OFF;
    PortInit.PORT_FUNC =     PORT_FUNC_OVERRID;
    PortInit.PORT_SPEED = PORT_SPEED_MAXFAST;
    PortInit.PORT_MODE = PORT_MODE_DIGITAL;

    // Конфигурация 1 ножки порта PORTF как выхода (UART2_TX)
    PortInit.PORT_OE = PORT_OE_OUT;
    PortInit.PORT_Pin = PORT_Pin_1;
    PORT_Init(MDR_PORTF, &PortInit);

    // Конфигурация 0 ножки порта PORTF как входа (UART2_RX)
    PortInit.PORT_OE = PORT_OE_IN;
    PortInit.PORT_Pin = PORT_Pin_0;
    PORT_Init(MDR_PORTF, &PortInit);

    //Разрешение тактирования UART2
    RST_CLK_PCLKcmd(RST_CLK_PCLK_UART2, ENABLE);
    // Инициализация делителя тактовой частоты для UART2
    UART_BRGInit(MDR_UART2, UART_HCLKdiv1);
    // Разрешение прерывания для UART2
    NVIC_EnableIRQ(UART2_IRQn);
    // Заполнение полей для переменной UART_InitStructure
    UART_InitStructure.UART_BaudRate   = 115200;    //тактовая частота передачи данных
    UART_InitStructure.UART_WordLength = UART_WordLength8b; //длина символов 8 бит
    UART_InitStructure.UART_StopBits   = UART_StopBits1;    //1 стоп бит
    UART_InitStructure.UART_Parity     = UART_Parity_No;    // нет контроля четности
    UART_InitStructure.UART_FIFOMode   = UART_FIFO_OFF;     // выключение FIFO буфера
    /* Аппаратный контроль за передачей и приемом */
    UART_InitStructure.UART_HardwareFlowControl = UART_HardwareFlowControl_RXE | \
UART_HardwareFlowControl_TXE;

    UART_Init (MDR_UART2, &UART_InitStructure);   //Инициализация UART2
    UART_ITConfig (MDR_UART2, UART_IT_RX, ENABLE);//Разрешение прерывания по приему
    UART_ITConfig (MDR_UART2, UART_IT_TX, ENABLE);//Разрешение прерывания по окончани передачи

    UART_Cmd(MDR_UART2, ENABLE); //Разрешение работы UART2

    while (1) {
        while (uart2_IT_RX_flag != SET); //ждем пока не не установиться флаг по приему байта
        uart2_IT_RX_flag = RESET;                    //очищаем флаг приема
        ReciveByte = UART_ReceiveData (MDR_UART2);  //считываем принятый байт
        UART_SendData (MDR_UART2, ReciveByte);      //отправляем принятый байт обратно
        while (uart2_IT_TX_flag != SET);            //ждем пока байт уйдет
        uart2_IT_TX_flag = RESET;                   //очищаем флаг передачи
    }
}
```

Лабораторная работа №8. Изучение модуля CAN

Цель работы:

Научиться использовать CAN интерфейс передачи данных в микроконтроллере 1986BE91T.

Приборы и материалы:

1. Отладочная плата MDR1986VE91T Rev 4
2. Программатор J-Link ARM
3. Блок питания 5В, 1.4А
4. ПК с установленной средой программирования Keil uVision
5. Кабель для подключения по интерфейсу CAN
6. Программа BUSMASTER 2.4.0
7. Преобразователь USB_CAN типа VS_COM [3] на базе микросхемы SJA1000 [4].

Порядок работы:

1. Собрать лабораторную установку согласно рис. 8.1, подключить программатор и преобразователь USB_CAN к компьютеру.

Рисунок 8.1 Лабораторная установка: 1- преобразователь CAN-USB типа VS-COM; 2- программатор TP-link (J-link); 3-штеккер блока питания 5В; 4-отладочная плата; 5- кабель для подключения по интерфейсу CAN.

2. Запустить приложение BUSMASTER, Выполнить настройку приложения **CAN>Driver Selection>VS_com CAN-API.**

3. В появившемся окне Рис. 8.2 выполнить поиск VSCAN **Search for Devices on COM-Ports**

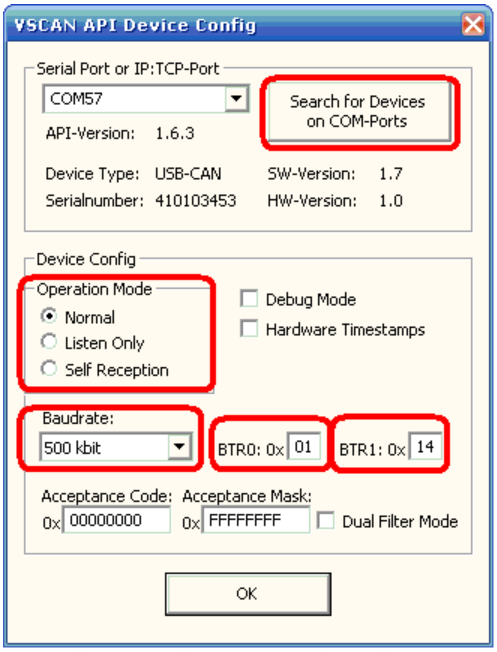


Рисунок 8.2. Настройки программы BUSMASTER

4. Выполнить настройки устройства BAUDRATE=500 кбит/с, BTR0=0x01, BTR1=0x14. Согласно описанию SJA1000 [4], данные настройки обеспечивают длительность Sync Segment - 1TQ, Propagation Segment 0TQ, Phase Segment 1 – 5TQ, Phase Segment 2 – 2TQ.

TQ=1/(8*500) с.

5. Выполнить **CAN>Transmit>Configure** для задания передаваемых из компьютера сообщений. Окно представлено на рис. 9.3. В поле **Trigger Cyclic on Event** задайте **Time Delay** =500 мс. И нажмите **ADD.**

6. В правой части окна рис 9.3 добавьте два сообщения с идентификаторами 0x1, 0x2 и произвольными данными. Тип сообщения **Std** (Стандартное). Передаваемые сообщения отображаются в TX Message List.

7. Теперь подготовим микроконтроллер для приема передачи данных. Данный проект удобно делать на базе проекта составленного в лабораторной работе № 4. Изучение работы таймера.

Скопируйте проект в отдельную папку. Откройте его. Выполните **Proect>manage>Run time....** В появившемся окне добавьте **Drivers>CAN.**

8. Полный код программы с комментариями представлен в приложении. Откомпилируйте и загрузите этот код в микроконтроллер.

9. Выполнить в программе BUSMASTER **CAN>Transmit >Enable** для включения передачи данных по интерфейсу CAN. Все 5 светодиодов стенда должны мигать с частотой порядка 1 Гц. Это говорит о передаче и приеме пакетов.

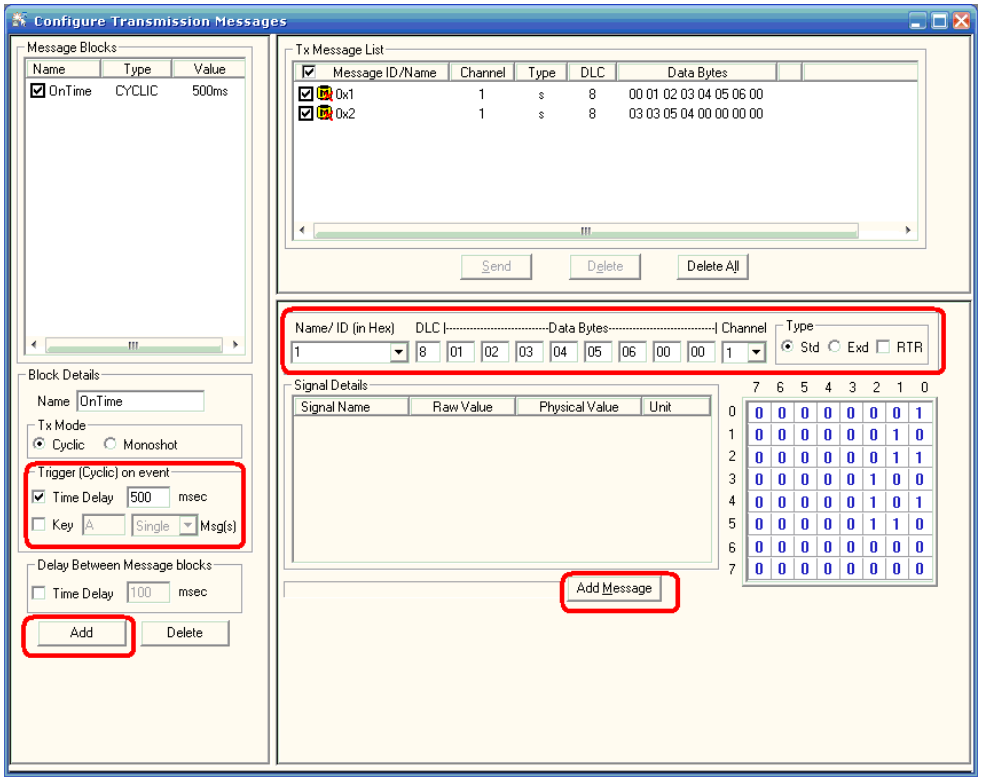


Рисунок 8.3. Настройки передаваемых сообщений программы BUSMASTER

10. Процесс передачи данных можно контролировать на контактах разъема представленных в таблице 8.1:

Таблица 8.1

Обозначение разъема, номер контакта	Имя контакта разъема	Функциональное назначение	Порта микроконтрол-лера, номер бита	Номер ножки микроконтрол-лера
X32.2 - 9	PD9	CAN_TX,	PortC, 8	83
X32.2 - 10	PD15	CAN_RX	PortC, 9	82

Осциллограмма представлена на рис 8.4

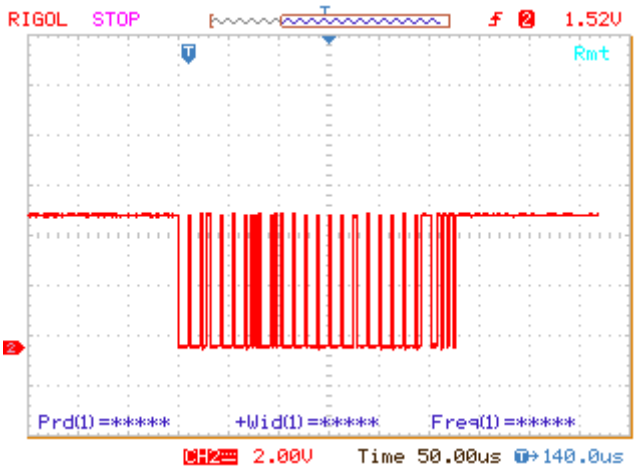
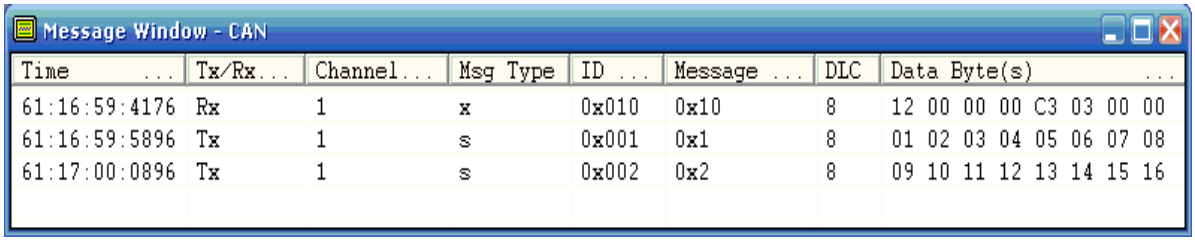


Рисунок 8.4 Осциллограмма передачи по CAN интерфейсу (X32.2 - 9).

В программе BUSMASTER окно сообщений должно показывать три сообщения, одно на прием и два на передачу



Time ...	Tx/Rx...	Channel...	Msg Type	ID ...	Message ...	DLC	Data Byte(s) ...
61:16:59:4176	Rx	1	x	0x010	0x10	8	12 00 00 00 C3 03 00 00
61:16:59:5896	Tx	1	s	0x001	0x1	8	01 02 03 04 05 06 07 08
61:17:00:0896	Tx	1	s	0x002	0x2	8	09 10 11 12 13 14 15 16

Рисунок 8.5 Окно сообщений.

11. Выполнить индивидуальное задание.

Сведения для выполнения

Все узлы шины CAN должны работать на одной скорости. Протокол CAN использует кодирование без возврата в ноль (NRZ). Также при передаче не передаются тактовые сигналы. Таким образом, приемники должны засинхронизоваться с тактовым сигналом передатчика.

Поскольку все узлы имеют свои индивидуальные тактовые генераторы, все приемники имеют специальный блок синхронизации DPLL. Максимальная скорость передачи CAN 1 Мбит/сек. Время битового интервала Nominal Bit Time определяется как:

TBIT = 1/Скорость передачи.

Блок DPLL разбивает битовый интервал на интервалы Time Quanta (TQ). Битовый интервал состоит из 4 частей:
Synchronization Segment (Sync_Seg), Propagation Time Segment (PSEG), Phase Buffer Segment 1 (SEG1), Phase Buffer Segment 2 (SEG2) рис. 9.6. По определению Nominal Bit Time программируется длительностью от 8 до 25 TQ. В этом случае

Nominal Bit Time = TQ * (Sync_Seg + PSEG + SEG1 + SEG2)

Время TQ фиксировано и определяется периодом генератора и программируемым предделителем BRP со значением от 1 до 65536:

TQ (µs) = ((BRP+1))/CLK (MHz) или **TQ (µs) = ((BRP+1)) * Tclk (µs)**

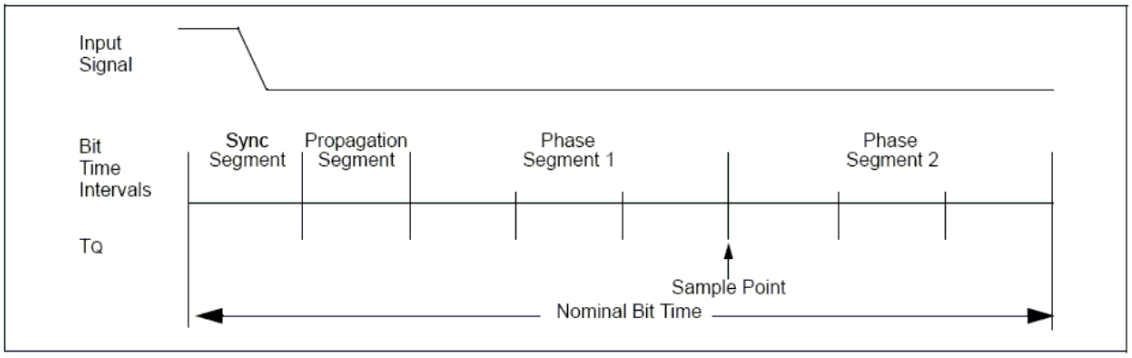


Рисунок 8.6. Передача одного бита информации по CAN

Synchronization Segment Эта часть битового интервала, в которой должно происходить переключение сигнала. Длительность этого интервала 1 TQ. Если переключение происходит в этой области, то приемник засинхронизирован с передатчиком.

Propagation Time Segment Эта часть предназначена, чтобы компенсировать физические задержки времени распространения сигнала в шине и внутренние задержки в узлах. Длительность этого интервала может быть запрограммирована от 1 до 8 TQ.

Phase Buffer Segments 1 и 2 Эти интервалы предназначены для более точной установки точки семплирования, которая располагается между ними. Длительности этих интервалов могут быть запрограммированы между 1 и 8 TQ.

Более подробно о CAN см. техническое описание микроконтроллера [1]

Порядок работы контроллера CAN (см. приложение).

1. Инициализация выводов микроконтроллера PortC8 PortC9;
2. Разрешить тактирования CAN контроллера;
3. Инициализация делителя тактовой частоты CAN;
4. Инициализация CAN с помощью структуры sCAN;
5. Инициализация прерываний CAN;
6. Отправка и прием сообщений;

Задания:

Задание 1. Изменить частоту передачи, или временные характеристики битового интервала по CAN интерфейсу на заданную преподавателем

Задание 2. Изменить тип передаваемых сообщений по заданию преподавателя

Задание 3. Обеспечить передачу сообщений из микроконтроллера по прерываниям (по окончанию передачи данных).

Приложение:

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_timer.h>
#include <MDR32F9Qx_can.h>

//Определения для светодиодов
#define LED1            PORT_Pin_10
#define LED2            PORT_Pin_11
#define LED3            PORT_Pin_12
#define LED4            PORT_Pin_13
#define LED5            PORT_Pin_14
#define MDR_PORTLED     MDR_PORTD

//Определение номеров приемного и передающего буфера и др. переменных
__IO uint32_t rx_buf = 0;
__IO uint32_t tx_buf = 1;
uint32_t j = 0;
PORT_InitTypeDef PORT_InitStructure;
PORT_InitTypeDef PORTDInit;

// Инициализация светодиодов
void PortsInit(){
  PORT_StructInit(&PORTDInit); //Load defaults
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTD, ENABLE);
  PORTDInit.PORT_Pin   = PORT_Pin_10 | PORT_Pin_11 | PORT_Pin_12 | PORT_Pin_13 | PORT_Pin_14;
  PORTDInit.PORT_OE    = PORT_OE_OUT;
  PORTDInit.PORT_MODE  = PORT_MODE_DIGITAL;
  PORTDInit.PORT_SPEED = PORT_SPEED_SLOW;
  PORT_Init(MDR_PORTD, &PORTDInit);
}

// Изменение состояния светодиода
void TogglePIN(MDR_PORT_TypeDef* PORTx, uint32_t PORT_Pin)
{
  if (PORT_ReadInputDataBit(PORTx, PORT_Pin))
  {PORT_ResetBits(PORTx, PORT_Pin);}
  else
  {PORT_SetBits(PORTx, PORT_Pin);}
}

TIMER_CntInitTypeDef TIM1Init;
```

```
// Инициализация таймера
void TimerInit(){
  RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER1, ENABLE);
  TIMER_BRGInit(MDR_TIMER1, TIMER_HCLKdiv1);
  TIMER_CntStructInit(&TIM1Init);    //Load defaults
  TIM1Init.TIMER_Prescaler        = 8000;
  TIM1Init.TIMER_Period           = 1000;
  TIMER_CntInit(MDR_TIMER1, &TIM1Init);
  NVIC_EnableIRQ(Timer1_IRQn);
  NVIC_SetPriority(Timer1_IRQn, 0);
  TIMER_ITConfig(MDR_TIMER1, TIMER_STATUS_CNT_ZERO, ENABLE);
  TIMER_Cmd(MDR_TIMER1, ENABLE);
}
// Моргагание 1 светодиодом 1 раз в секунду
void ToggleLed(uint32_t LED_Num)
{
  if (PORT_ReadInputDataBit(MDR_PORTLED, LED_Num))
  {PORT_ResetBits(MDR_PORTLED, LED_Num);}
  else
  {PORT_SetBits(MDR_PORTLED, LED_Num);}
}
// Включение светодиода
void LEDOn(uint32_t LED_Num){PORT_SetBits(MDR_PORTLED, LED_Num);}
// Выключение светодиода
void LEDOff(uint32_t LED_Num){PORT_ResetBits(MDR_PORTLED, LED_Num);}
// Задержка
volatile void Delay_Tick(volatile uint32_t Tick){while (Tick--);}

int main() {
  CAN_InitTypeDef  sCAN;      // переменная для инициализации CAN
  CAN_TxMsgTypeDef TxMsg;     // переменная для передаваемого сообщения

  PortsInit();//Инициализация портов ввода вывода
  TimerInit();//Инициализация таймера

  /* Включение HSE осциллятора (внешнего кварцевого резонатора)*/
  RST_CLK_HSEconfig(RST_CLK_HSE_ON);

  if (RST_CLK_HSEstatus() == SUCCESS){  /* Если HSE осциллятор включился и прошел текст*/
    // Выбор HSE осциллятора в качестве источника тактовых импульсов для CPU_PLL
    // и установка умножителя тактовой частоты CPU_PLL равного 7
    // Частота внешнего кварца равна 8 МГц Максимальная частота процессора 80 МГц ,
    // RST_CLK_CPU_PLLconfig ( Источник тактирования PLL, Коэффициент умножения =N+1);
    // Коэффициент умножения=0 соответствует умножение на 1 и т.д..
    RST_CLK_CPU_PLLconfig ( RST_CLK_CPU_PLLsrcHSEdiv1, 0 );
    /* Включение схемы PLL*/
    RST_CLK_CPU_PLLcmd(ENABLE);
    if (RST_CLK_CPU_PLLstatus() == SUCCESS) {//Если включение CPU_PLL прошло успешно
      /* Установка CPU_C3_prescaler = 1 */
      RST_CLK_CPUclkPrescaler(RST_CLK_CPUclkDIV1);
      /* Установка CPU_C2_SEL от CPU_PLL выхода вместо CPU_C1 такта*/
      RST_CLK_CPU_PLLuse(ENABLE);
      /* Выбор CPU_C3 такта на мультиплексоре тактовых импульсов микропроцессора (CPU clock MUX) */
      RST_CLK_CPUclkSelection(RST_CLK_CPUclkCPU_C3);
    }
    else        /* блок CPU_PLL не включился*/
    {while(1);}
  }
  else          /* кварцевый резонатор HSE не включился */
  {while(1);}
  // В результате этих действий микропроцессор работает на частоте 8МГц

  //разрешение тактирования блока RST_CLK
  RST_CLK_PCLKcmd(RST_CLK_PCLK_RST_CLK ,ENABLE);

  //разрешение тактирования порта PORTC
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);

  PORT_DeInit(MDR_PORTC);
  //Инициализация ножек CAN1 порта
  PORT_InitStructure.PORT_Pin   = PORT_Pin_8 ;
  PORT_InitStructure.PORT_OE    = PORT_OE_OUT;
  PORT_InitStructure.PORT_FUNC  = PORT_FUNC_MAIN;
  PORT_InitStructure.PORT_MODE  = PORT_MODE_DIGITAL;
  PORT_InitStructure.PORT_SPEED = PORT_SPEED_FAST;

  PORT_Init(MDR_PORTC, &PORT_InitStructure);

  PORT_InitStructure.PORT_Pin   = PORT_Pin_9;
```

```
  PORT_InitStructure.PORT_OE    = PORT_OE_IN;
  PORT_InitStructure.PORT_FUNC  = PORT_FUNC_MAIN;
  PORT_InitStructure.PORT_MODE  = PORT_MODE_DIGITAL;
  PORT_InitStructure.PORT_SPEED = PORT_SPEED_FAST;

  PORT_Init(MDR_PORTC, &PORT_InitStructure);
  // Инициализация тактирования CAN1
  RST_CLK_PCLKcmd(RST_CLK_PCLK_CAN1,ENABLE);
  // Инициализация делителя тактовой частоты CAN1
  CAN_BRGInit(MDR_CAN1,CAN_HCLKdiv1);
  // Инициализация CAN1 (обнуление)
  CAN_DeInit(MDR_CAN1);
  // Заполнение полей структуры
  CAN_StructInit (&sCAN);
  sCAN.CAN_ROP  = DISABLE;//прием собственных пакетов запрещен
  sCAN.CAN_SAP  = DISABLE;//подтверждение приема собственных пакетов запрещено
  sCAN.CAN_STM  = DISABLE;//режим самотестирования запрещен SELF TEST MODE
  sCAN.CAN_ROM  = DISABLE;//режим только чтение запрещен ENABLE READONLY MODE
  sCAN.CAN_PSEG = CAN_PSEG_Mul_2TQ; //Длительность PSEG
  sCAN.CAN_SEG1 = CAN_SEG1_Mul_3TQ; //Длительность SEG1
  sCAN.CAN_SEG2 = CAN_SEG2_Mul_2TQ; //Длительность SEG2
  sCAN.CAN_SJW  = CAN_SJW_Mul_4TQ;  //Длительность сегмента подстройки частоты передачи
  sCAN.CAN_SB   = CAN_SB_1_SAMPLE;  //Одна выборка на бит
  sCAN.CAN_BRP  = 1;  //предделитель частоты процессора для CAN Частота CAN = PCLK/(BRP + 1)
  // Инициализация CAN1
  CAN_Init (MDR_CAN1,&sCAN);
  // Разрешение работы CAN1
  CAN_Cmd(MDR_CAN1, ENABLE);
  /* Разрешение прерывания CAN1*/
  NVIC_EnableIRQ(CAN1_IRQn);

  /* Разрешение CAN1 GLB_INT  (глобальных прерываний CAN), RX_INT (прерываний по приему CAN)*/
  CAN_ITConfig( MDR_CAN1, CAN_IT_GLBINTEN | CAN_IT_RXINTEN, ENABLE);

  /* Разрешение прерывания при приеме данных в буфер*/
  CAN_RxITConfig( MDR_CAN1 ,(1<<rx_buf), ENABLE);

  // Инициализация приемного буфера
  CAN_Receive(MDR_CAN1, rx_buf, ENABLE);

  while(1){
    //Подготовка сообщения для передачи
    TxMsg.IDE     = CAN_ID_EXT;
    TxMsg.DLC     = 0x08;
    TxMsg.PRIOR_0 = DISABLE;
    TxMsg.ID      = 0x10;
    TxMsg.Data[1] = j;
    TxMsg.Data[0] = 0x12;
    j++;
    //Инициализация передачи данных
    CAN_Transmit(MDR_CAN1, tx_buf, &TxMsg);
    // Задержка
    Delay_Tick(1000000);
    // Моргание светодиодом
    ToggleLed(LED3);
  }
}
void CAN1_IRQHandler(void){
  CAN_RxMsgTypeDef RxMessage;
  // Получить сообщение
  CAN_GetRawReceivedData(MDR_CAN1, rx_buf, &RxMessage);
  // Если это сообщение с идентификатором 1 то моргать LED2
  if((CAN_EXTID_TO_STDID(RxMessage.Rx_Header.ID)==0x1))
  {ToggleLed(LED2); };
  // Если это сообщение с идентификатором 2 то моргать LED4
  if((CAN_EXTID_TO_STDID(RxMessage.Rx_Header.ID)==0x2))
  {ToggleLed(LED4); };
  // Очистка бита приема передачи сообщения
  CAN_ITClearRxTxPendingBit(MDR_CAN1, rx_buf, CAN_STATUS_RX_READY);
  ToggleLed(LED5);
}

// Прерывание от таймера (моргагание 1 светодиодом 1 раз в секунду)
void Timer1_IRQHandler() {
  if(TIMER_GetITStatus(MDR_TIMER1, TIMER_STATUS_CNT_ZERO)){
    TogglePIN(MDR_PORTD, PORT_Pin_10);
    TIMER_ClearITPendingBit(MDR_TIMER1, TIMER_STATUS_CNT_ZERO);
  }
}
```

Лабораторная работа №9. Создание USB – CDC устройства

Цель работы:

Создание простейшего USB CDC устройства типа виртуальный COM порт на базе 1986BE91T.

Приборы и материалы:

1. Отладочная плата MDR1986VE91T Rev 4.
2. Программатор J-Link ARM.
3. Блок питания 5В, 1.4А.
4. ПК с установленной средой программирования Keil uVision.
5. Осциллограф.
6. Генератор импульсов.
7. Кабель USB для подключения к компьютеру.

Порядок работы:

1. Собрать лабораторную установку согласно рис. 9.1, подключить программатор и USB кабель, включить питание лабораторной установки.

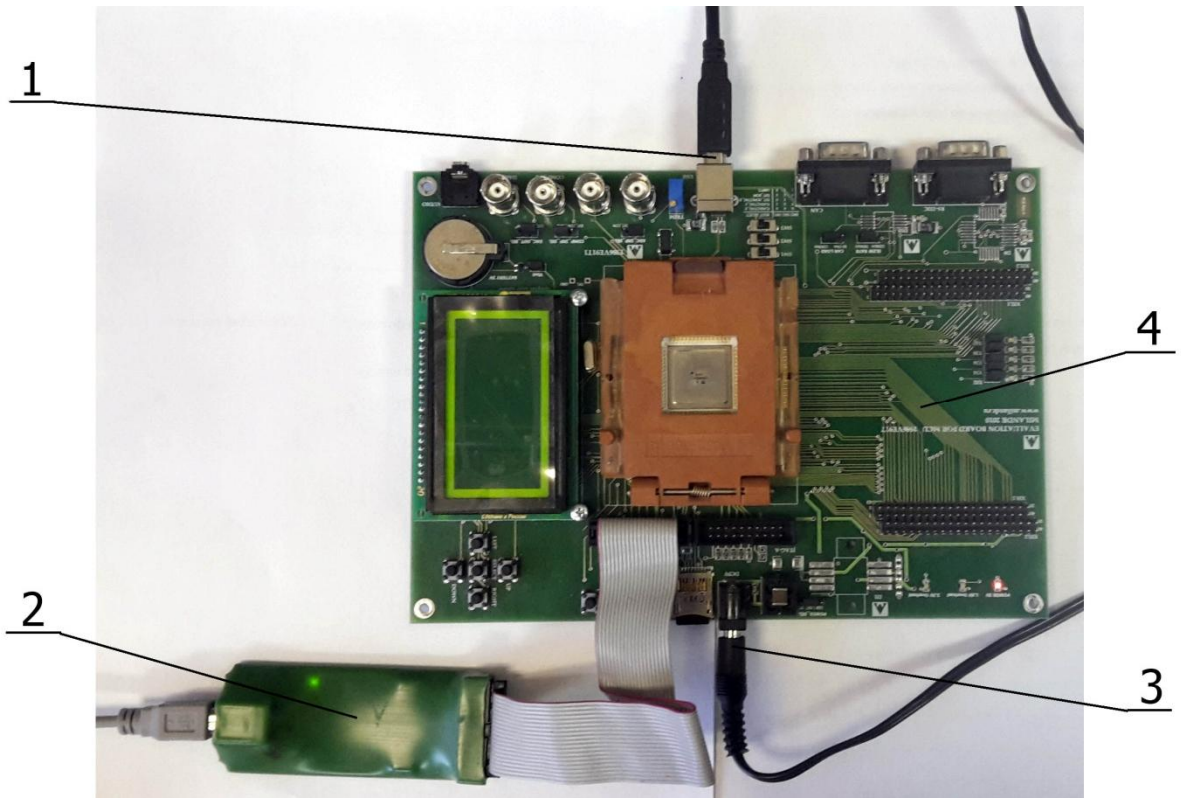


Рисунок 9.1 Лабораторная установка: 1- USB кабель; 2- программатор TP-link (J-link); 3- штеккер блока питания 5В; 4-отладочная плата.

2. Подключить USB кабель (1) к компьютеру.
3. Установить программу PuTTY [5] расположенную по адресу
http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
4. Подключить программатор к разъему JTAG_B
5. Запустить Keil uVision5 и создать проект в среде Keil uVision5 или использовать MDRproject
6. Подключить библиотеки Startup_MDR1986DT9x, PORT, RST_CLK, USB,

USB_librarary в менеджере библиотек.
7. Заменить текст main.c на текст программы (см. Приложение). Данная программа принимает один байт данных по USB и тот же, принятый байт данных, выдает обратно на компьютер.
8. Скомпилировать и загрузить проект в микроконтроллер
9. Далее необходимо установить драйвер виртуального COM порта usbser.sys. Для windows.xp просто устанавливаем MDRVComPort.inf (Находится в папке с примерами оригинального компакт диска или на сайте фирмы Миландр). Для 64 разрядных операционных необходимо разрешить установку неподписанных драйверов по инструкции приведенной в описании операционной системы, скачать драйвер mchpusb.inf фирмы микрочип и заменить в нем список поддерживаемых устройств с соответствии с нижеприведенным текстом:
[DeviceList]
%DESCRIPTION%=DriverInstall, USB\VID_04D8&PID_000A,
USB\VID_0483&PID_F125
[DeviceList.NTamd64]
%DESCRIPTION%=DriverInstall, USB\VID_04D8&PID_000A,
USB\VID_0483&PID_F125
Данным текстом мы добавляем наше устройство в список поддерживаемых устройств. Windows определит производителя COM порта как Microchip. Более подробно описание inf файла можно прочитать например тут [7]
Далее необходимо установить mchpusb.inf на ЭВМ с windows 8. В результате вышеприведенных действий в диспетчере устройств появится наш виртуальный COM порт Рис 9.2.

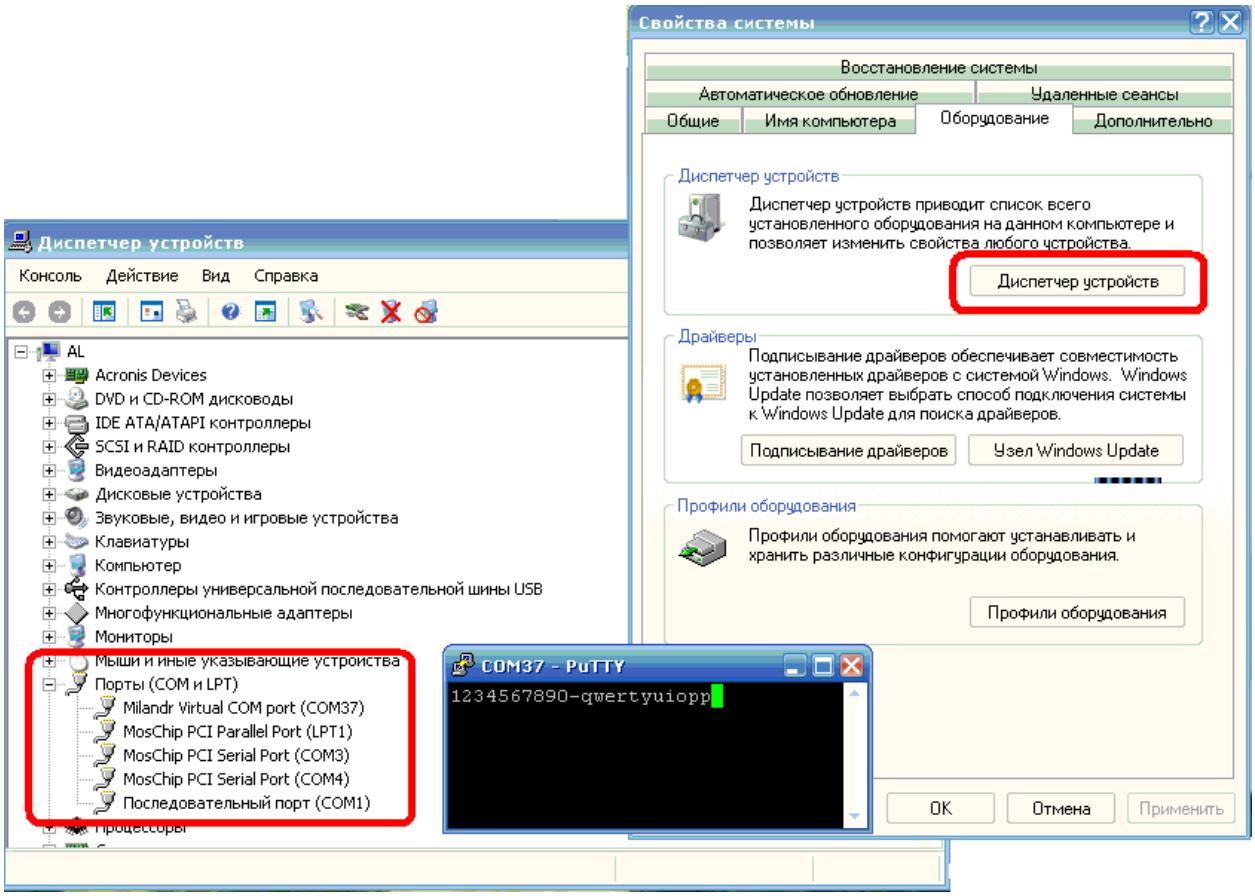


Рисунок 9.2 Окно программы Putty и диспетчера устройств.

9. Запустить программу Putty, установить настройки: *Serial line* – COM№ порт к которому подключен микроконтроллер (Возможные варианты № можно узнать в диспетчере устройств Windows. В нашем случае это COM37); *Speed – 115200*; *Connection type - Serial* и нажать Open.
10. В появившемся окне рисунок 9.2 ввести произвольный текст. Как видим, те данные, которые вводятся с клавиатуры, отображаются в окне Putty, что говорит о двунаправленной передаче данных между контроллером и ЭВМ по шине USB.
11. Выполнить индивидуальное задание.

Сведения для выполнения

Основы построения устройств USB можно изучить, воспользовавшись документом USB in a NutShell - путеводитель по стандарту USB. Этот документ доступен на русском и английском языках [6].

Устройства USB принадлежат к одному из классов. В частности, спецификация USB описывает класс коммуникационных устройств (Communication Device Class – CDC), который определяет множество различных устройств: модемы, терминалы, телефоны, Ethernet адаптеры, хабы, ADSL модемы, последовательный порт. Последовательный порт (COM порт) не применяется в ПК нового поколения, его заменяет интерфейс USB. Для соответствия этому изменению приложения на базе COM порта должны переходить на USB. Приложение-устройство класса CDC может быть подключено к ПК и являтся виртуальным последовательным портом. Для обеспечения работы последовательного порта в ЭВМ необходим драйвер. В операционной системе Microsoft Windows стандартный драйвер последовательного порта называется usbser.sys, и он является частью набора драйверов. Однако, в отличие от других стандартных драйверов наподобие Mass Storage Devices (MSD), драйвер usbser.sys не загружается автоматически, когда устройство CDC подключено в первый раз. Для установки драйвера необходимо воспользоваться .inf файлом и установить его вручную. Для операционной системы Windows XP это файл MDRVComPort.inf, Для 64 разрядных операционных систем придется модифицировать драйвер mchpusb.inf фирмы Microchip (см. Порядок работы).

Для того чтобы компьютер мог сам определить тип устройства подключенного по USB, и выбрать для этого устройства необходимую программу драйвер все устройства USB имеют иерархию дескрипторов, которые описывают информацию для хоста – такую, как тип устройства, изготовитель, какую версию USB поддерживает устройство, какими способами устройство может быть сконфигурировано, количество конечных точек и их типы и т. д.

Наиболее общие дескрипторы USB следующие: дескрипторы устройства, дескрипторы конфигурации, дескрипторы интерфейса, дескрипторы конечной точки, строковые дескрипторы.

Рассмотрим дескриптор USB для библиотек Миландр расположенные в файле USB_Library\MDR32F9Qx_usb_CDC.c

Дескриптор устройства

```
static uint8_t Usb_CDC_Device_Descriptor[0x12] =
{
    0x12,            /* bLength                  */
    0x01,            /* bDescriptorType (Device) */
    0x10, 0x01,      /* bcdUSB                   */
    0x02,            /* bDeviceClass (CDC)       */
    0x00,            /* bDeviceSubClass          */
    0x00,            /* bDeviceProtocol          */
    MAX_PACKET_SIZE, /* bMaxPacketSize0   MAX_PACKET_SIZE=32    */
    0x83, 0x04,      /* idVendor                 */
    0x25, 0xF1,      /* idProduct                */
    0x00, 0x00,      /* bcdDevice                */
    0x00,            /* iManufacturer            */
```

```
    0x00,           /* iProduct                */
    0x00,           /* iSerialNumber           */
    0x01            /* bNumConfigurations      */
};
```

Этот дескриптор стройства длинной 18 байт (0x12) . Описание полей дескриптора приведено в таблице.

№	Поле	Число байт	Тип данных	Описание
0	bLength	1	Число	Размер дескриптора в байтах (для дескриптора устройства размер 12 байт)
1	bDescriptorType	1	Константа	Тип дескриптора - Device Descriptor (0x01)
2	bcdUSB	2	BCD	Номер спецификации USB, с которой совместимо устройство. USB2.0
4	bDeviceClass	1	Class (класс)	Код класса (назначается организацией USB Org) Если равно 0, то каждый интерфейс указывает свой собственный код класса. Если равен 0xFF, то код класса определяется вендором. Иначе поле содержит код стандартного класса. Для USB CDC класс 1
5	bDeviceSubClass	1	SubClass (подкласс)	Код подкласса (назначается организацией USB Org)
6	bDeviceProtocol	1	Протокол	Код протокола (назначается организацией USB Org)
7	bMaxPacketSize	1	Число	Максимальный размер пакета для конечной точки 0. Допустимые размеры 8, 16, 32, 64
8	idVendor	2	ID	Vendor ID, VID (назначается организацией USB Org)
10	idProduct	2	ID	Product ID, PID (назначается организацией - производителем)
12	bcdDevice	2	BCD	Device Release Number (номер версии устройства)
14	iManufacturer	1	Индекс	Индекс строки, описывающей производителя
15	iProduct	1	Индекс	Индекс строки, описывающей продукт
16	iSerialNumber	1	Индекс	Индекс строки, содержащей серийный номер
17	bNumConfigurations	1	Целое (Integer)	Количество возможных конфигураций

Более подробно можно узнать в спецификации USB, а также в документе USB in a NutShell - путеводитель по стандарту USB [6] . Помимо дескриптора устройства в файле USB_Library\MDR32F9Qx_usb_CDC.c расположен дескриптор конфигурации, а также все другие необходимые дескрипторы.

Программа расположенная в приложении позволяет переслать обратно на компьютер принятый байт данных.

Задания:

Задание 1. Измените программу таким образом, чтобы при приеме определенных символов загорались светодиоды стенда (символы и номера светодиодов задает преподаватель).
Задание 2. Измените программу таким образом, чтобы при нажатии кнопок на стенде в ЭВМ отсылались соответствующие оповещения по типу «Нажата кнопка «ВВЕРХ».
Задание 3. Попробуйте изменить длину передаваемого пакета данных на 2 байта и попробуйте переслать по USB данные АЦП в формате 16 разрядного целого со знаком.

Приложение:

```
#include "MDR32Fx.h"
#include "MDR32F9Qx_usb_handlers.h"
#include "MDR32F9Qx_rst_clk.h"

#define BUFFER_LENGTH                        100

USB_Clock_TypeDef USB_Clock_InitStruct;
USB_DeviceBUSParam_TypeDef USB_DeviceBUSParam;
// Буфер для USB
static uint8_t Buffer[BUFFER_LENGTH];

#ifdef USB_CDC_LINE_CODING_SUPPORTED
static USB_CDC_LineCoding_TypeDef LineCoding;
#endif /* USB_CDC_LINE_CODING_SUPPORTED */

static void Setup_CPU_Clock(void);
static void Setup_USB(void);
static void VCom_Configuration(void);

int main(void)
{
  // Конфигурация параметров виртуального COM порта
  VCom_Configuration();
  /* CDC инициализация */
  // инициализация Буфера длинной 1 байт
  USB_CDC_Init(Buffer, 1, SET);
  // Установка тактовой частоты микропроцессора
  Setup_CPU_Clock();
  Setup_USB();
  /* Основной цикл оставляем пустым */
  while (1)  {}
}

/* Установка тактовой частоты процессора */
void Setup_CPU_Clock(void)
{

  // Включение тактирования от внешнего кварцевого резонатора
  RST_CLK_HSEconfig(RST_CLK_HSE_ON);
  if (RST_CLK_HSEstatus() != SUCCESS)
  {
     while (1){}//Если внешний источник тактовой частоты не запустился
                                //то переходим в бесконечный цикл
  }

  // Определение параметров схемы PLL, коэффициент деления внешнего источника тактовой частоты 1,
 //коэффициент умножения 10 (CPU_C1_SEL = HSE)
  RST_CLK_CPU_PLLconfig(RST_CLK_CPU_PLLsrcHSEdiv1, RST_CLK_CPU_PLLmul10);
  // Разрешение работы PLL
  RST_CLK_CPU_PLLcmd(ENABLE);
  if (RST_CLK_CPU_PLLstatus() != SUCCESS)
  {
    while  (1){}//Если внешний источник тактовой частоты не запустился
                //то переходим в бесконечный цикл
  }
  //Выбираем источник тактовой частоты процессора
  //см. Спецификация микроконтроллеров серии 1986BE9x, K1986BE9x,
  //K1986BE92QI, K1986BE92QC, K1986BE91H4 стр. 123 раздел Сигналы тактовой частоты
  /* CPU_C3_SEL = CPU_C2_SEL */
  RST_CLK_CPUclkPrescaler(RST_CLK_CPUclkDIV1);
  /* CPU_C2_SEL = PLL */
```

```
  RST_CLK_CPU_PLLuse(ENABLE);
  /* HCLK_SEL = CPU_C3_SEL */
  RST_CLK_CPUclkSelection(RST_CLK_CPUclkCPU_C3);
}

// Включение и настройка USB
void Setup_USB(void)
{
  /* Включение тактирования схемы USB*/
  RST_CLK_PCLKcmd(RST_CLK_PCLK_USB, ENABLE);
       // Выбор источника тактирования USB
  USB_Clock_InitStruct.USB_USBC1_Source = USB_C1HSEdiv2;
       // Выбор коэффициента умножения схемы PLL для USB
  USB_Clock_InitStruct.USB_PLLUSBMUL    = USB_PLLUSBMUL12;
       //Выбор режима USB FULL Speed
  USB_DeviceBUSParam.MODE  = USB_SC_SCFSP_Full;
       //Выбор скорости передачи данных 12мБайт в сек.
  USB_DeviceBUSParam.SPEED = USB_SC_SCFSR_12Mb;
       // Подтягивание линии DP к питанию.
  USB_DeviceBUSParam.PULL  = USB_HSCR_DP_PULLUP_Set;
       // Инициализация USB  с заданными параметрами
  USB_DeviceInit(&USB_Clock_InitStruct, &USB_DeviceBUSParam);
  /* Разрешение всех видов прерываний от USB*/
  USB_SetSIM(USB_SIS_Msk);
  // Включение питания USB и разрешение передачи и приема данных
  USB_DevicePowerOn();
  /* Включение прерываний USB */
#ifdef USB_INT_HANDLE_REQUIRED
  NVIC_EnableIRQ(USB_IRQn);
#endif /* USB_INT_HANDLE_REQUIRED */

  USB_DEVICE_HANDLE_RESET;
}

/* Задание конфигурации последовательной линии связи которую может прочитать хост*/
static void VCom_Configuration(void)
{
#ifdef USB_CDC_LINE_CODING_SUPPORTED
  LineCoding.dwDTERate = 115200;
  LineCoding.bCharFormat = 0;
  LineCoding.bParityType = 0;
  LineCoding.bDataBits = 8;
#endif /* USB_CDC_LINE_CODING_SUPPORTED */
}

/*USB_CDC_HANDLE_DATA_RECEIVE implementation - data echoing*/
// Данная процедура автоматически вызывается при приеме данных по USB.
USB_Result USB_CDC_RecieveData(uint8_t* Buffer, uint32_t Length)
{
  USB_Result result;

  /* Передача одного байта назад на устройство */
  result =     USB_CDC_SendData(Buffer, Length);
       return USB_SUCCESS;
}

#ifdef USB_CDC_LINE_CODING_SUPPORTED
//Эти два запроса отправляются хостом, чтобы изменить или прочитать конфигурацию последовательной
линии связи, которая включает в себя:
//• Baudrate (скорость передачи данных)
//• Number of stop bits (количество стоп-битов)
//• Parity check (наличие бита контроля четности)
//• Number of data bits (количество бит данных)
//Когда программа терминала (ПО хоста наподобие HyperTerminal) меняет установки COM-порта, то
//отправляется запрос SetLineCoding с новыми параметрами. Хост может также запросить текущие
//настройки запросом GetLineCoding, и не менять их, если настройки корректны.


/* USB_CDC_HANDLE_GET_LINE_CODING implementation example */
USB_Result USB_CDC_GetLineCoding(uint16_t wINDEX, USB_CDC_LineCoding_TypeDef* DATA)
{
  assert_param(DATA);
  if (wINDEX != 0)
  {
    /* Invalid interface */
    return USB_ERR_INV_REQ;
  }

  /* Just store received settings */
```

```
  *DATA = LineCoding;
  return USB_SUCCESS;
}

/* USB_CDC_HANDLE_SET_LINE_CODING implementation example */
USB_Result USB_CDC_SetLineCoding(uint16_t wINDEX, const USB_CDC_LineCoding_TypeDef* DATA)
{
  assert_param(DATA);
  if (wINDEX != 0)
  {
    /* Invalid interface */
    return USB_ERR_INV_REQ;
  }
  /* Just send back settings stored earlier */
  LineCoding = *DATA;
  return USB_SUCCESS;
}
#endif /* USB_CDC_LINE_CODING_SUPPORTED */
```

Лабораторная работа №10. Изучение работы таймера в режиме захвата

Цель работы:

Изучение схемы подключения и порядка настройки МК, для реализации режима захвата в таймерах.

Приборы и материалы:

1. Отладочная плата MDR1986VE91T Rev 4.
2. Программатор J-Link ARM.
3. Блок питания 5В, 1.4А.
4. ПК с установленной средой программирования Keil uVision.
5. Генератор тактовых импульсов.
6. Осциллограф.

Ход работы:

1. Собрать лабораторную установку по рисунку 10.1

2. Подключить универсальный генератор сигналов к разъему отладочной платы X33.2 – контакт номер 6, а так же подключить параллельно контакту осциллограф. Второй канал осциллографа подключить к любому светодиоду контакты разъема X32, X34, X36, X38, X40 при этом перемычка не убирается.

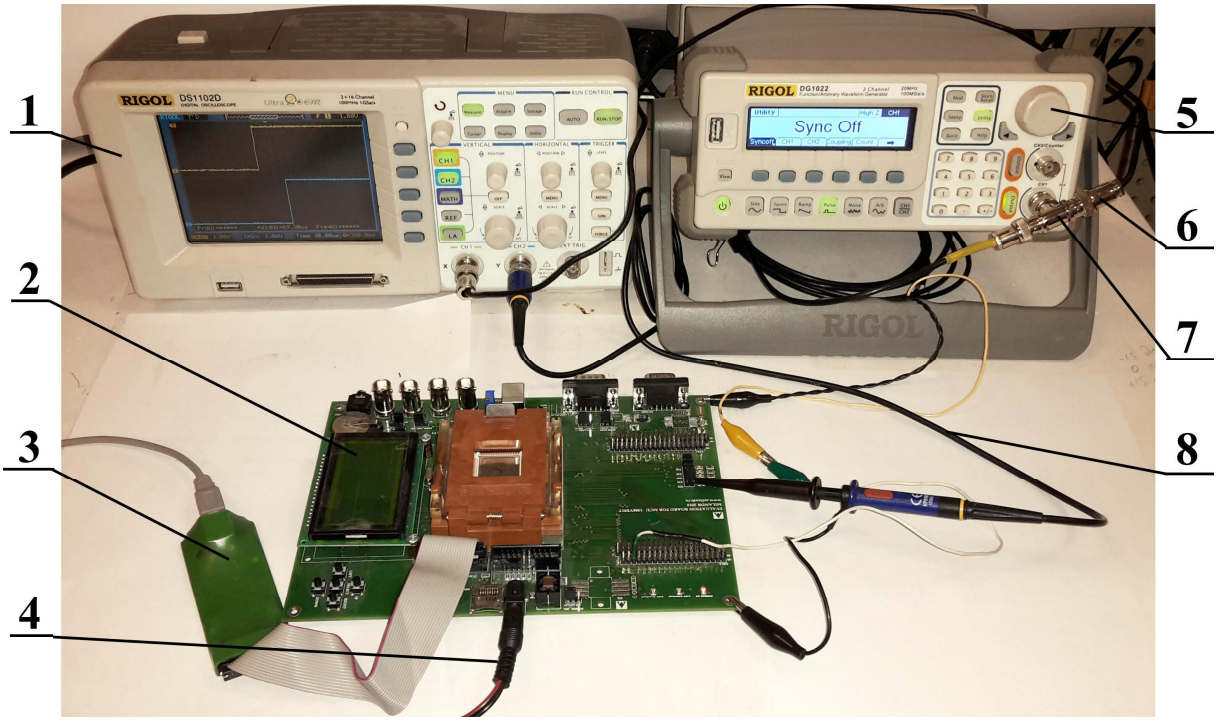


Рисунок 10.1 Общий вид лабораторной установки: 1 – осциллограф; 2 – отладочная плата; 3 - программатор J-Link ARM; 4 – шнур питания; 5 – универсальный генератор сигналов; 6 – коаксиальный кабель для соединения двух BNC выходов; 7 – разветвитель BNC; 8 – осциллографический пробник.

2. Включить осциллограф в режиме двух каналов.

3. Установить генератор тактовых сигналов по настройкам выданным преподавателем, например, период импульсов 50мс, длительность импульса 1-5мс, амплитуда импульсов 3В. Внешний вид сигналов представлен на рисунке 10.2

4. Включить отладочную плату.

5. Создайть новый проект, подключить в менеджере ***Manage run-time environment*** к проекту библиотеки Startup_MDR1986BE9x, PORT, TIMER , RST_CLK необходимые для работы.

6. Скопировать в main.c код представленный в приложении. Данный код реализует захват положительного и отрицательного фронтов входного тактового сигнала, подаваемого с генератора, вызов процедуры обработки прерывания в которой реализуется формирование выходных тактовых импульсов аналогичных входным. Затем скомпилировать проект, произвести программирование МК.

7. Запустить генератор тактовых сигналов в заданном режиме и зафиксировать показания на осциллографе (Рисунок 10.2).

8. Сделать выводы о времени реакции на прерывание от таймера.

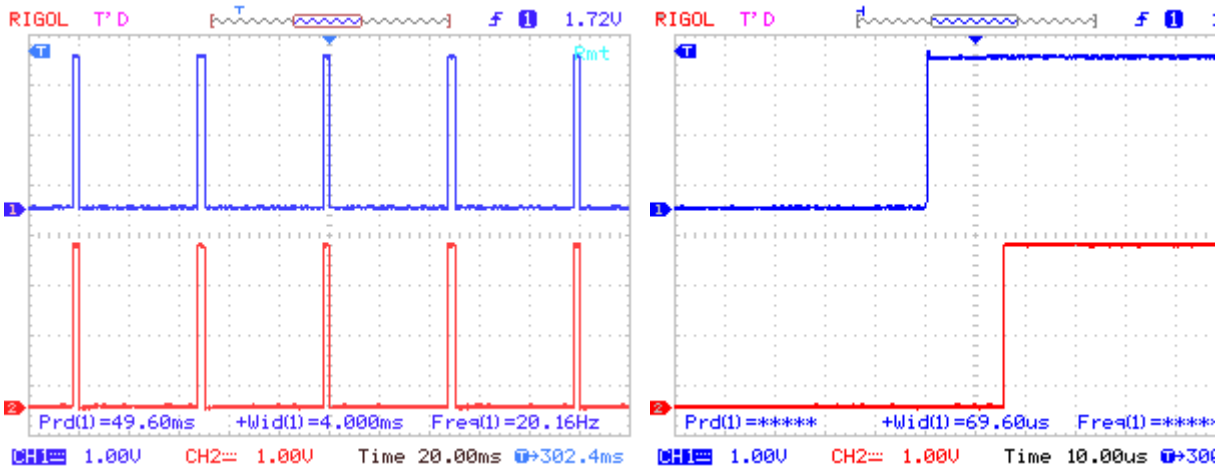


Рисунок 10.2 Результаты работы программы (справа фронт импульса показан укрупненно. Можно видеть задержку прерывания).

9. Выполнить индивидуальное задание

Сведения для выполнения:

Все таймеры выполнены на основе 16-битного перезагружаемого счетчика. Счет может быть прямой, обратный или двунаправленный. Каждый из трех таймеров микроконтроллера содержит 16-битный счетчик, 16-битный предделитель частоты и 4-канальный блок захвата/сравнения. Рассмотрим блок захвата сравнения.

Структурная схема блока захвата представлена на рис. 10.3.

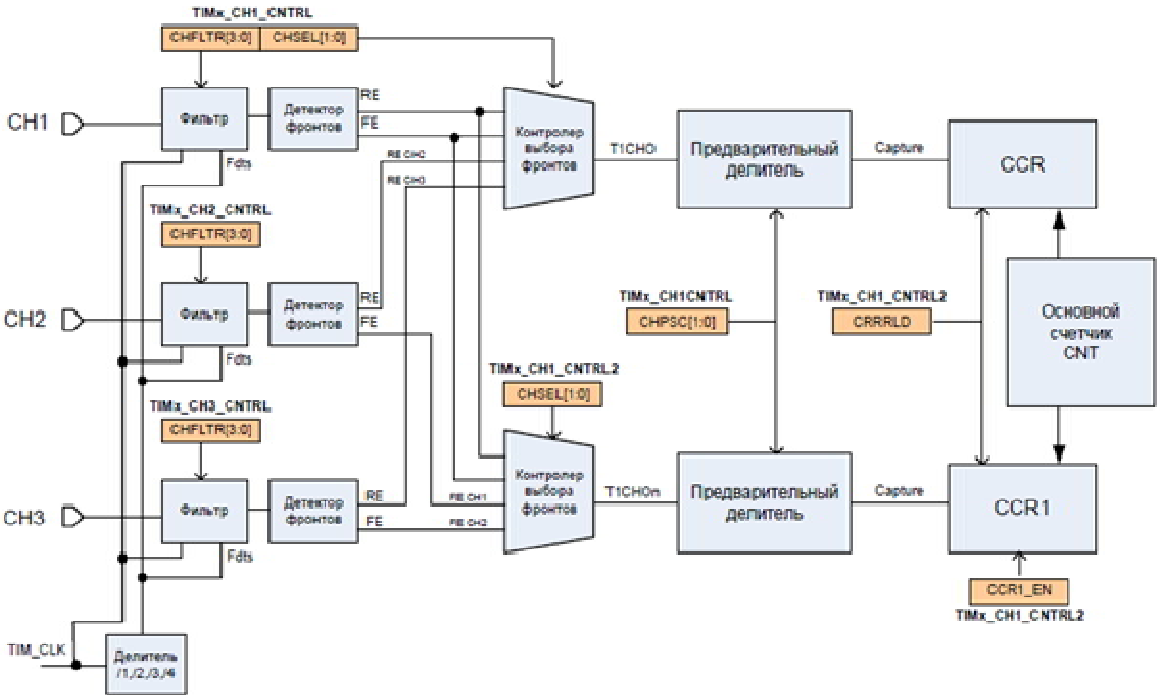


Рисунок 10.3 – Структурная схема блока захвата таймера.

Для включения режима захвата для определенного канала необходимо:

- Инициализировать порты микроконтроллера CHxi на прием событий как переопределенную или альтернативную функцию.

- Включить тактирование таймера, настроить предделитель и модуль счета.

- Настроить выбранный канал таймера с помощью структуры ChnInitTypeDef.

- Разрешить прерывания таймера по захвату фронта и среза.

- Написать процедуру обработки прерывания таймера в которой описать необходимые команды (**Примечание:** внешние прерывания в микроконтроллере реализованы только по уровню, и прерывания таймеров можно использовать для детектирования фронтов сигналов, то есть реализовать внешние прерывания по положительному или отрицательному фронту).

Для регистрации событий по линиям CHi используется схема регистрации событий. Входной сигнал фиксируется в Таймере с частотой Fdts, или TIM_CLK. Также вход может быть настроен на прием импульсов заданной длины за счет конфигурирования блока «фильтр». На выходе блока фильтр вырабатывает сигнал положительного перепада и отрицательного перепада. На блоке контроллера выбора фронтов производится выбор используемого для захвата сигнала между положительным фронтом канала, отрицательным фронтом канала и положительными и отрицательными фронтами сигналов от других каналов.

После блока контроллера выбора фронтов сигнал подается на предварительный делитель, который может быть использован для фиксации каждого события, каждого второго, каждого четвертого и каждого восьмого события. Выход предварительного делителя является сигналом Capture для регистра CCR, и Capture1 для регистра CCR1, при этом в регистры CCR и CCR1 записывается текущее значение основного счетчика CNT.

Рассмотрим структуру инициализации канала:

```
typedef struct
{
uint16_t TIMER_CH_Number;
/* Определяет Номер канала ТАЙМЕРА, который будет сконфигурирован. Этот параметр может быть значением номера канала ТАЙМЕРА. Он может принимать значения: TIMER_CHANNEL1, TIMER_CHANNEL2, TIMER_CHANNEL3, TIMER_CHANNEL4 */

uint16_t TIMER_CH_Mode;
/*Определяет режим канала Таймера. Возможные значения:
TIMER_CH_MODE_CAPTURE        - Определяет режим канала - захват.
TIMER_CH_MODE_PWM            - Определяет режим канала ШИМ */

uint16_t TIMER_CH_ETR_Ena;
/*Разрешение работы входа ETR. */

uint16_t TIMER_CH_ETR_Reset;
/* Включает или отключает сброс таймера по сигналу ETR.
TIMER_CH_ETR_RESET_Disable   - Отключает сброс ETR
TIMER_CH_ETR_RESET_Enable    - Включает сброс ETR */

uint16_t TIMER_CH_BRK_Reset;
/* Включает или отключает сброс таймера по сигналу BRK.
TIMER_CH_BRK_RESET_Disable   - Отключает сброс BRK
TIMER_CH_BRK_RESET_Enable    - Включает сброс  BRK */

uint16_t TIMER_CH_REF_Format;
/* Определяет порядок выработки сигнала REF. См. Таблицу 276 [1]*/

uint16_t TIMER_CH_Prescaler;
/* Определяет конфигурацию Делителя частоты Канала ТАЙМЕРА.
TIMER_CH_Prescaler_None      - Нет предделителя входной частоты.
TIMER_CH_Prescaler_div_2     - Выбирает входной делитель частоты 2
TIMER_CH_Prescaler_div_4     - Выбирает входной делитель частоты 4
TIMER_CH_Prescaler_div_8     - Выбирает входной делитель частоты 8
*/

uint16_t TIMER_CH_EventSource;
/* Определяет источник события для канала.
TIMER_CH_EvSrc_PE      - Выбирает положительный фронт из текущего канала таймера.
TIMER_CH_EvSrc_NE      - Выбирает отрицательный фронт из текущего канала таймера.
```

```
TIMER_CH_EvSrc_PE_OC1  - Выбирает положительный фронт из другого канала таймера (Вариант 1)
TIMER_CH_EvSrc_PE_OC2  - Выбирает положительный фронт из другого канала таймера (Вариант 2).
*/

uint16_t TIMER_CH_FilterConf;
/* Определяет конфигурацию фильтра канала таймера. См. Таблицу 276 [2] и файл MDR32F9Qx_timer_h */

uint16_t TIMER_CH_CCR_UpdateMode;
/* Определяет режим обновления CCR и CCR1.
TIMER_CH_CCR_Update_Immediately             - CCR, CCR1 обновляются в любой момент времени.
TIMER_CH_CCR_Update_On_CNT_eq_0             - CCR, CCR1 обновляются, при условии (CNT == 0).
*/

uint16_t TIMER_CH_CCR1_Ena;
/* Включает или отключает регистр CCR1. */

uint16_t TIMER_CH_CCR1_EventSource;
/* Определяет источник события для регистра CCR1.
TIMER_CH_CCR1EvSrc_PE     - Выбирает положительный фронт из текущего канала таймера.
TIMER_CH_CCR1EvSrc_NE     - Выбирает отрицательный фронт из текущего канала таймера.
TIMER_CH_CCR1EvSrc_NE_OC1 -Выбирает отрицательный фронт из другого канала таймера (вариант 1).
TIMER_CH_CCR1EvSrc_NE_OC2 -Выбирает отрицательный фронт из другого канала таймера (вариант 2).
*/
}TIMER_ChnInitTypeDef;
```

Далее передаём созданную и описанную структуру в канал таймера при помощи процедуры с сигнатурой - **`TIMER_ChnInit(MDR_TIMER1, &CH1Init);`**

Задания:

1. Написать программу для подсчёта периода импульса. Значение периода импульса просмотреть в отладчике.
2. Написать программу для подсчета длительности импульса. Значение длительности импульса просмотреть в отладчике.

Приложение:

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_timer.h>

void PortInit(void){
  PORT_InitTypeDef PORTCInit;
  PORT_InitTypeDef PORTDInit;
  //Включение тактирования портов C и D
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTD, ENABLE);
  // Инициализация порта C4 как переопределенная функция
  // таймер 1, канал 1, захват/ШИМ
  PORTCInit.PORT_Pin          =  PORT_Pin_4; //X33.2-6 контакт
  PORTCInit.PORT_OE           = PORT_OE_IN;
  PORTCInit.PORT_PULL_UP      = PORT_PULL_UP_OFF;
  PORTCInit.PORT_PULL_DOWN    = PORT_PULL_DOWN_ON ;
  PORTCInit.PORT_PD_SHM       = PORT_PD_SHM_ON ;
  PORTCInit.PORT_PD           = PORT_PD_DRIVER;
  PORTCInit.PORT_FUNC         = PORT_FUNC_OVERRID ;
  PORTCInit.PORT_SPEED        = PORT_SPEED_MAXFAST;
  PORTCInit.PORT_MODE         = PORT_MODE_DIGITAL;
  PORT_Init(MDR_PORTC, &PORTCInit);
  // Инициализация 5 светодиодов стенда.
  PORTDInit.PORT_Pin          = PORT_Pin_10|PORT_Pin_11|PORT_Pin_12|PORT_Pin_13|PORT_Pin_14;

  PORTDInit.PORT_OE           = PORT_OE_OUT;
  PORTDInit.PORT_FUNC         = PORT_FUNC_PORT;
  PORTDInit.PORT_MODE         = PORT_MODE_DIGITAL;
  PORTDInit.PORT_SPEED        = PORT_SPEED_SLOW;

  PORT_Init(MDR_PORTD, &PORTDInit);
}
```

```
void TimerInit(){

  TIMER_CntInitTypeDef TIM1Init;
  TIMER_ChnInitTypeDef CH1Init;
  //Задание начельных значений регистров для таймера.
  TIMER_DeInit(MDR_TIMER1);
  //Включение тактирования таймера
  RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER1, ENABLE);
  // Включение делителя тектовой частоты
  TIMER_BRGInit(MDR_TIMER1, TIMER_HCLKdiv1);
  TIMER_CntStructInit(&TIM1Init);
  // Задаем параметры счета
  TIM1Init.TIMER_Prescaler           = 1;
  TIM1Init.TIMER_Period              = 5000;
  // Инициализация таймера
  TIMER_CntInit (MDR_TIMER1, &TIM1Init);
  // Настройка первого канала таймера на регистрацию событий
  // Загрузка параметров по умолчанию
  TIMER_ChnStructInit(&CH1Init);
  // Выбираем первый канал
  CH1Init.TIMER_CH_Number           = TIMER_CHANNEL1;
  // Выбираем режим захвата
  CH1Init.TIMER_CH_Mode             = TIMER_CH_MODE_CAPTURE;
  // Выбираем положительный фронт импульса для события захвата в регистр CCR
  CH1Init.TIMER_CH_EventSource      = TIMER_CH_EvSrc_PE;
  // Выбираем отрицательный фронт импульса для события захвата в регистр CCR1
  CH1Init.TIMER_CH_CCR1_EventSource = TIMER_CH_CCR1EvSrc_NE;
  // Немедленное обновление регистров CCR и CCR1
  CH1Init.TIMER_CH_CCR_UpdateMode   = TIMER_CH_CCR_Update_Immediately;
  // Инициализация таймера
  TIMER_ChnInit(MDR_TIMER1, &CH1Init);
  // Очистка флагов событий
  TIMER_ClearFlag(MDR_TIMER1, 0);
  // Разрешение прерываний по захвату фронта и среза
  TIMER_ITConfig(MDR_TIMER1, TIMER_STATUS_CCR_CAP_CH1 |
                      TIMER_STATUS_CCR_CAP1_CH1, ENABLE);
  // Включение прерывания от таймера
  NVIC_EnableIRQ(Timer1_IRQn);
  // Установка приоритета прерывания
  NVIC_SetPriority(Timer1_IRQn, 0);
  // Разрешение работы таймера
  TIMER_Cmd(MDR_TIMER1, ENABLE);
}

//Процедура обработки прерывания для таймера
void Timer1_IRQHandler(){
  // Если пришел фронт импульса то включаем все светодиоды
  if(TIMER_GetITStatus(MDR_TIMER1, TIMER_STATUS_CCR_CAP_CH1)){
    PORT_SetBits(MDR_PORTD, PORT_Pin_10);
    PORT_SetBits(MDR_PORTD, PORT_Pin_11);
    PORT_SetBits(MDR_PORTD, PORT_Pin_12);
    PORT_SetBits(MDR_PORTD, PORT_Pin_13);
    PORT_SetBits(MDR_PORTD, PORT_Pin_14);
    // Очистка флага прерывания
    TIMER_ClearITPendingBit(MDR_TIMER1, TIMER_STATUS_CCR_CAP_CH1);
  }
// Если пришел срез импульса то выключаем все светодиоды
  if(TIMER_GetITStatus(MDR_TIMER1, TIMER_STATUS_CCR_CAP1_CH1)){
    PORT_ResetBits(MDR_PORTD, PORT_Pin_10);
    PORT_ResetBits(MDR_PORTD, PORT_Pin_11);
    PORT_ResetBits(MDR_PORTD, PORT_Pin_12);
    PORT_ResetBits(MDR_PORTD, PORT_Pin_13);
    PORT_ResetBits(MDR_PORTD, PORT_Pin_14);
    // Очистка флага прерывания
    TIMER_ClearITPendingBit(MDR_TIMER1, TIMER_STATUS_CCR_CAP1_CH1);
  }
}
int main(){
  PortInit();
  TimerInit();
  while(1){    }
}
```

Лабораторная работа №11. Изучение работы таймера в режиме ШИМ

Цель работы:

Изучение основных особенностей построения схемы подключения и настройки МК, для реализации режима широтно-импульсной модуляции (ШИМ) с помощью таймеров. На примере 6 канального ШИМ для трехфазного моста.

Приборы и материалы:

1. Отладочная плата MDR1986VE91T Rev 4
2. Программатор J-Link ARM
3. Блок питания 5В, 1.4А
4. ПК с установленной средой программирования Keil uVision
5. Осциллограф

Ход работы:

1. Собрать лабораторную установку согласно рисунку 11.1.

Рисунок 11.1 Общий вид лабораторной установки: 1 – осциллограф; 2 – отладочная плата; 3 - программатор J-Link ARM; 4 – шнур питания.

2. Включить отладочную плату.

3. Создайть новый проект, подключить в менеджере ***Manage run-time environment*** к проекту библиотеки, PORT, TIMER, RST_CLK, необходимые для работы.

4. Скопировать в main.c код представленный в приложении. Данный код формирует на трех каналах таймера 6 выходных сигналов с разной скважностью и полярностью. Затем скомпилировать проект, произвести программирование МК.

5. Включить осциллограф в режиме двух каналов. Снять осциллограммы напряжений при подключении к контактам приведенным в таблице 11.1.

Таблица 11.1. Контакты отладочной платы для наблюдения ШИМ сигнала.

Обозначение разъема,	Имя контакта разъема	Функциональное назначение	Порта микроконтрол-	Номер ножки микроконтрол-

номер контакта			лера, номер бита	лера
X33.1, 35	PA1	CH1	PortA, 1	129
X33.1, 36	PA2	CH1N	PortA, 2	128
X33.1, 31	PA3	CH2	PortA, 3	127
X33.1, 32	PA4	CH2N	PortA, 4	126
X33.1, 29	PA5	CH3	PortA, 5	125
X33.1, 28	PA8	CH3N	PortA, 8	122

6. На осциллографе наблюдать картину генерации импульсов создаваемые ШИМ МК. На рисунках 11.2 – 11.4 представлены картины генерации. Частота генерации составила 1.623 кГц что соответствует частоте работы МК
F=1.623*(0xFFF)=6.65 МГц. МК в данном случае работает от встроенной некалиброванной RC цепочки (см. рис31[2]).

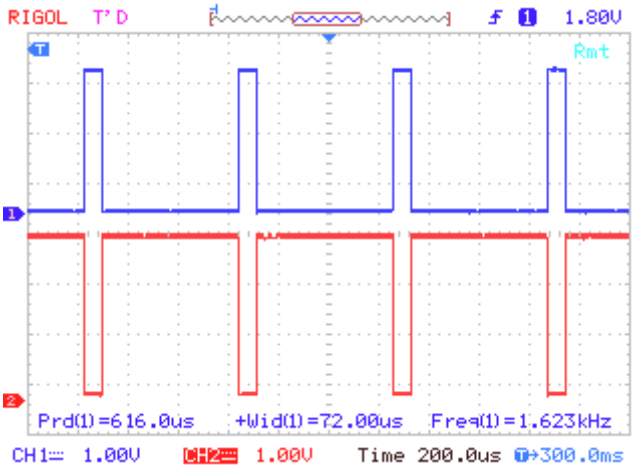


Рисунок 11.2 ШИМ канала 1: сверху CH1, снизу CH1N.

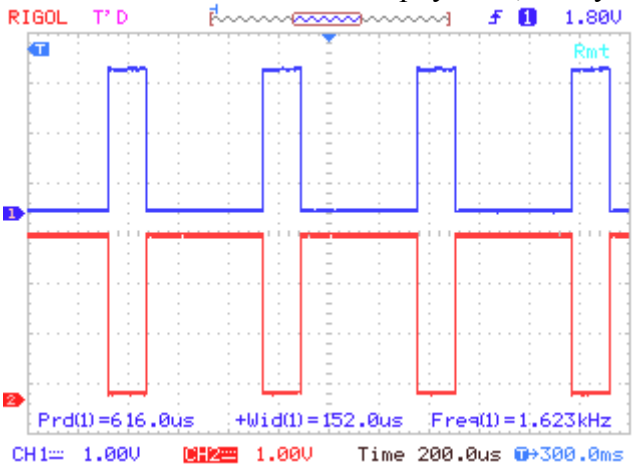


Рисунок 11.3 ШИМ канала 2: сверху CH2, снизу CH2N.

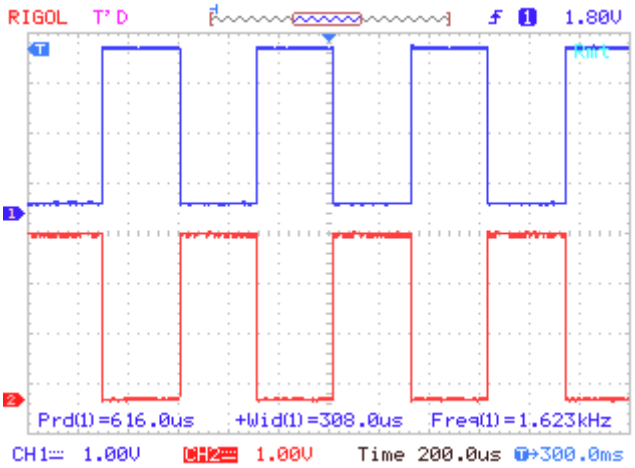


Рисунок 11.4 ШИМ канала 3: сверху CH3, снизу CH3N.

Задания

1. Составить программу генератора ШИМ с линейно изменяющейся от 0 до 100% скважностью для одного на выбор канала, за время 10 сек.

2. Составить программу генератора ШИМ с изменяющейся по синусоидальному закону скважностью для одного на выбор канала, период синусоиды 1 сек.

Сведения для выполнения

Широтно-импульсная модуляция (ШИМ, англ. pulse-width modulation (PWM)) — процесс управления мощностью, подводимой к нагрузке, путём изменения скважности импульсов, при постоянной частоте. ШИМ используется в импульсных источниках питания электронных устройств, вентильных схемах управления различными видами электродвигателей, в последовательных цифроаналоговых преобразователях и т.п.

Структурная схема блока формирования ШИМ для МК 1986ВЕ91Т представлена на рис. 11.5 Он состоит из: схемы сравнения **CCR**, схемы формирования выходного сигнала **REF**, формирователя специальной паузы между переключениями транзисторов верхнего и нижнего плеч в вентильной схеме управления электродвигателем - Dead time generator (**DTG**) (так называемый генератор мертвого времени), двух выходных каналов (**OUTPUT)** прямого и инверсного. Они предназначаютя для управлениями транзисторами верхнего и нижнего плеч.

Для включения режима ШИМ необходимо провести следующие действия:

- Инициализировать выводы микроконтроллера на выполнения функции выходных каналов таймера **OUTPUT**. (Порт А, ножки 1,2,3,4,5,8)

- Включить тактирование таймера, установить модуль счета и предделитель с помощью структуры **TIMER_CntInitTypeDef**.

- Включить и настроить три канала таймера на выполнение функции ШИМ генератора с помощью структуры **TIMER_ChnInitTypeDef**.

- Настроить выходные каналы **(OUTPUT)** таймера на выдачу прямого и инверсного канала сигнала **REF** с помощью структуры **TIMER_ChnOutInitTypeDef**.

- С помощью процедуры **TIMER_SetChnCompare (MDR_TIMER1, TIMER_CHANNEL1, CCR1_Val)** установить длительность импульса ШИМ сигнала.

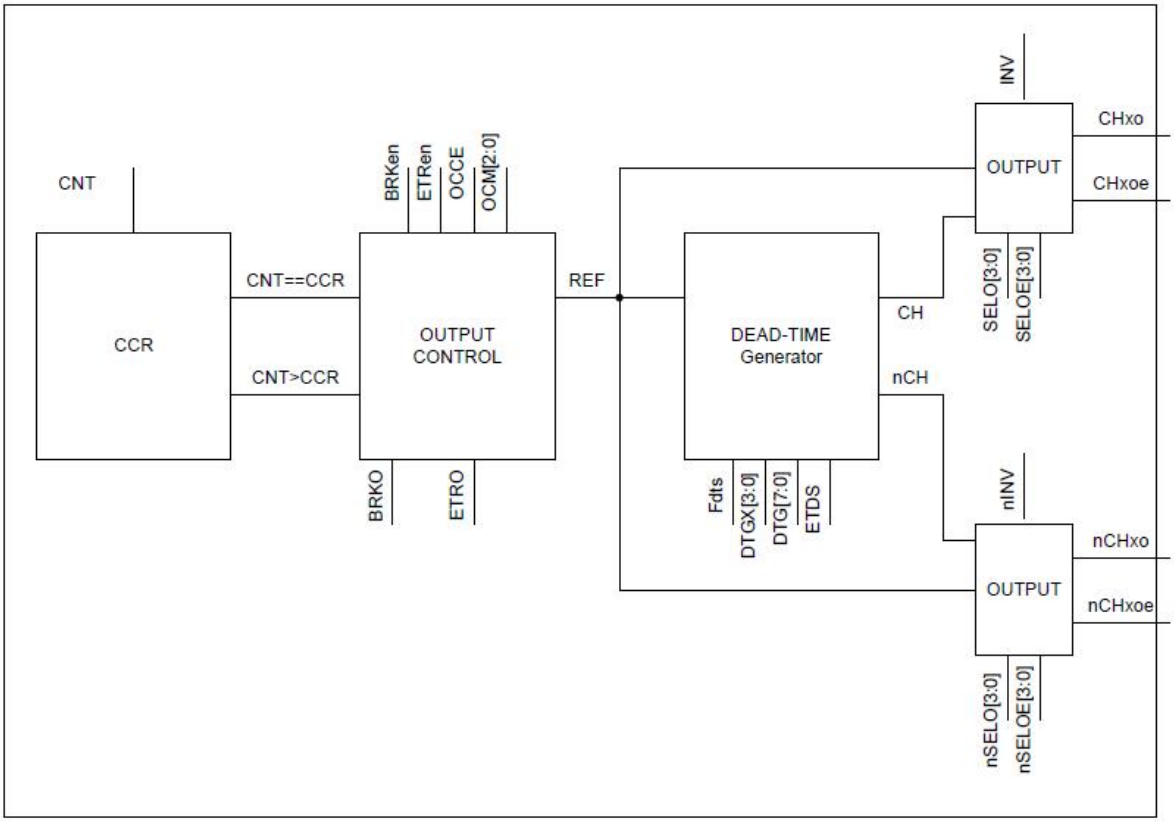


Рисунок 11.5 - Структурная схема блока формирования ШИМ

Рассмотрим структуру для инициализации выходов таймера в режиме генерации ШИМ.

```
typedef struct {
uint16_t TIMER_CH_Number;
/*Определяет Номер канала ТАЙМЕРА, который будет сконфигурирован*/

uint16_t TIMER_CH_DirOut_Polarity;
/*Определяет полярность выхода CHx.
TIMER_CHOPolarity_NonInverted  - неинвертированный выход
TIMER_CHOPolarity_Inverted     - инвертированный выход*/

uint16_t TIMER_CH_DirOut_Source;
/*Определяет источник сигнала для выхода CHx.
TIMER_CH_OutSrc_Only_0    - всегда на выход выдается 0, канал на выход не работает;
TIMER_CH_OutSrc_Only_1    - всегда на выход выдается 1, канал всегда работает на выход;
TIMER_CH_OutSrc_REF       - на  выход выдается сигнал REF;
TIMER_CH_OutSrc_DTG       - на выход выдается сигнал с DTG */


uint16_t TIMER_CH_DirOut_Mode;
/*Определяет, один из четырех режимов работы прямого канала при его работе на вывод.
TIMER_CH_OutMode_Input  - всегда на CHyOE выдается 0, канал на выход не работает;
TIMER_CH_OutMode_Output   - всегда на CHyOE выдается 1, канал всегда работает на выход;
TIMER_CH_OutMode_REF_as_OE - на CHyOE выдается сигнал REF, при REF = 0 третье состояние, при REF = 1
выход;
TIMER_CH_OutMode_DTG_as_OE - на CHyOE выдается сигнал с DTG, при CHyOE = 0 третье состояние, при
CHyOE = 1 */

uint16_t TIMER_CH_NegOut_Polarity;
/*Определяет полярность выхода CHxN*/

uint16_t TIMER_CH_NegOut_Source;
/*Определяет источник сигнала для выхода CHxN*/

uint16_t TIMER_CH_NegOut_Mode;
```

```
/*Определяет, один из четырех режимов работы инверсного канала при его работе на вывод.*/

uint16_t TIMER_CH_DTG_MainPrescaler;
/*Определяет основной делитель частоты DTG. см таблицу 282 [2]*/

uint16_t TIMER_CH_DTG_AuxPrescaler;
/*Определяет вспомогательный делитель частоты DTG. см таблицу 282 [2]*/

uint16_t TIMER_CH_DTG_ClockSource;
/*Определяет источник тактовых импульсов для DTG. см таблицу 282 [2]*/

}TIMER_ChnOutInitTypeDef;
```

Приложение:

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_timer.h>

// Переменные для инициализации таймера
TIMER_CntInitTypeDef sTIM_CntInit;
TIMER_ChnInitTypeDef sTIM_ChnInit;
TIMER_ChnOutInitTypeDef sTIM_ChnOutInit;
PORT_InitTypeDef PORT_InitStructure;

// Значения длительности импульса по каналам 1,2,3
uint16_t CCR1_Val = 0x1FF;
uint16_t CCR2_Val = 0x3FF;
uint16_t CCR3_Val = 0x7FF;

int main(){
//Включение тактирования порта A
RST_CLK_PCLKcmd((RST_CLK_PCLK_PORTA), ENABLE);
/* Сброс установок порта A*/
PORT_DeInit(MDR_PORTA);
/* Конфигурация выводов таймера TIMER1: CH1, CH1N, CH2, CH2N, CH3, CH3N  */
/* Настройка порта A разряды 1, 2, 3, 4, 5 ,8 */
// На демонстрационной плате в разъеме X33:1 контакты 35, 36, 31, 32, 29, 28 соответственно.
PORT_InitStructure.PORT_Pin  =(PORT_Pin_1  |  PORT_Pin_2  |  PORT_Pin_3  |  PORT_Pin_4  |  PORT_Pin_5|
PORT_Pin_8);
PORT_InitStructure.PORT_OE    = PORT_OE_OUT;
PORT_InitStructure.PORT_FUNC  = PORT_FUNC_ALTER;
PORT_InitStructure.PORT_MODE  = PORT_MODE_DIGITAL;
PORT_InitStructure.PORT_SPEED = PORT_SPEED_FAST;
PORT_Init(MDR_PORTA, &PORT_InitStructure);

// Включение тактирования таймера 1
RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER1,ENABLE);
/* Сброс установок таймера 1 */
TIMER_DeInit(MDR_TIMER1);
// Заполнение структуры значениями по умолчанию
TIMER_CntStructInit(&sTIM_CntInit);
// Предделитель частоты таймера равен 1 то есть частота таймера равна частоте микроконтоллера.
sTIM_CntInit.TIMER_Prescaler               = 0x0;
// Модуль счета = 0xFFF
// Период импульсов ШИМ = 0xFFF / частота счета таймера .
sTIM_CntInit.TIMER_Period                  = 0xFFF;
// Инициализация таймера 1
TIMER_CntInit (MDR_TIMER1,&sTIM_CntInit);
/* Инициализация каналов таймера 1:  CH1,CH1N,CH2,CH2N,CH3,CH3N */
TIMER_ChnStructInit(&sTIM_ChnInit);
// Режим работы канала - генерация ШИМ
sTIM_ChnInit.TIMER_CH_Mode                = TIMER_CH_MODE_PWM;
// Формат выработки сигнала REF номер 6:
// при счете вверх – REF = 1 если  CNT<CCR, иначе REF = 0;
// значение регистра CCR определяет длительность импульса ШИМ
sTIM_ChnInit.TIMER_CH_REF_Format          = TIMER_CH_REF_Format6;

// Инициализируем канал 1
sTIM_ChnInit.TIMER_CH_Number              = TIMER_CHANNEL1;
TIMER_ChnInit(MDR_TIMER1, &sTIM_ChnInit);

// Инициализируем канал 2
sTIM_ChnInit.TIMER_CH_Number              = TIMER_CHANNEL2;
TIMER_ChnInit(MDR_TIMER1, &sTIM_ChnInit);

// Инициализируем канал 3
```

```
sTIM_ChnInit.TIMER_CH_Number              = TIMER_CHANNEL3;
TIMER_ChnInit(MDR_TIMER1, &sTIM_ChnInit);

// Устанавливаем длительность импульсов по каждому каналу
TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL1, CCR1_Val);
TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL2, CCR2_Val);
TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL3, CCR3_Val);

/* Инициализация прямого и инверсного выходов (стр289 описания!!!)
для каждого из каналов таймера 1: CH1,CH1N,CH2,CH2N,CH3,CH3N */

// Заполнение элементов структуры значениями по умолчанию
TIMER_ChnOutStructInit(&sTIM_ChnOutInit);
// Выбор источника сигнала для прямого выхода  CHxN - сигнал REF
sTIM_ChnOutInit.TIMER_CH_DirOut_Source      = TIMER_CH_OutSrc_REF;
// Настройна прямого выхода микроконтроллера CHxN на вывод данных
sTIM_ChnOutInit.TIMER_CH_DirOut_Mode        = TIMER_CH_OutMode_Output;
// Выбор источника сигнала для инверсного выхода  CHxN - сигнал REF
sTIM_ChnOutInit.TIMER_CH_NegOut_Source      = TIMER_CH_OutSrc_REF;
// Настройна инверсного выхода микроконтроллера CHxN на вывод данных
sTIM_ChnOutInit.TIMER_CH_NegOut_Mode          = TIMER_CH_OutMode_Output;
// Настраиваем выходы канала 1
sTIM_ChnOutInit.TIMER_CH_Number                  = TIMER_CHANNEL1;
TIMER_ChnOutInit(MDR_TIMER1, &sTIM_ChnOutInit);
// Настраиваем выходы канала 2
sTIM_ChnOutInit.TIMER_CH_Number                  = TIMER_CHANNEL2;
TIMER_ChnOutInit(MDR_TIMER1, &sTIM_ChnOutInit);
// Настраиваем выходы канала 3
sTIM_ChnOutInit.TIMER_CH_Number                  = TIMER_CHANNEL3;
TIMER_ChnOutInit(MDR_TIMER1, &sTIM_ChnOutInit);

/* Включаем делитель тактовой частоты таймера 1*/
TIMER_BRGInit(MDR_TIMER1,TIMER_HCLKdiv1);

/* После всех настроек разрешаем работу таймера 1 */
TIMER_Cmd(MDR_TIMER1,ENABLE);

while(1){}

}
```

Лабораторная работа №12. Изучение внешних прерываний (EXT_INT)

Цель работы:

Научиться использовать использовать внешние прерывания в микроконтроллере КР1986ВЕ1Т, на примере создания преобразователя длительность импульса – напряжение (диапазон измерения длительности 0-50мс, диапазон выходного напряжения 0-3.3В).

Приборы и материалы:

1. Отладочная плата MDR1986VE91T Rev 4.
2. Программатор J-Link ARM.
3. Блок питания 5В, 1.4А.
4. ПК с установленной средой программирования Keil uVision.
5. Генератор тактовых импульсов.
6. Осциллограф.
7. Коаксиальный кабель для соединения двух BNC выходов- 2шт., разветвитель BNC, Соединительные провода.

Ход работы:

1. Собрать лабораторную установку согласно рис.12.2.

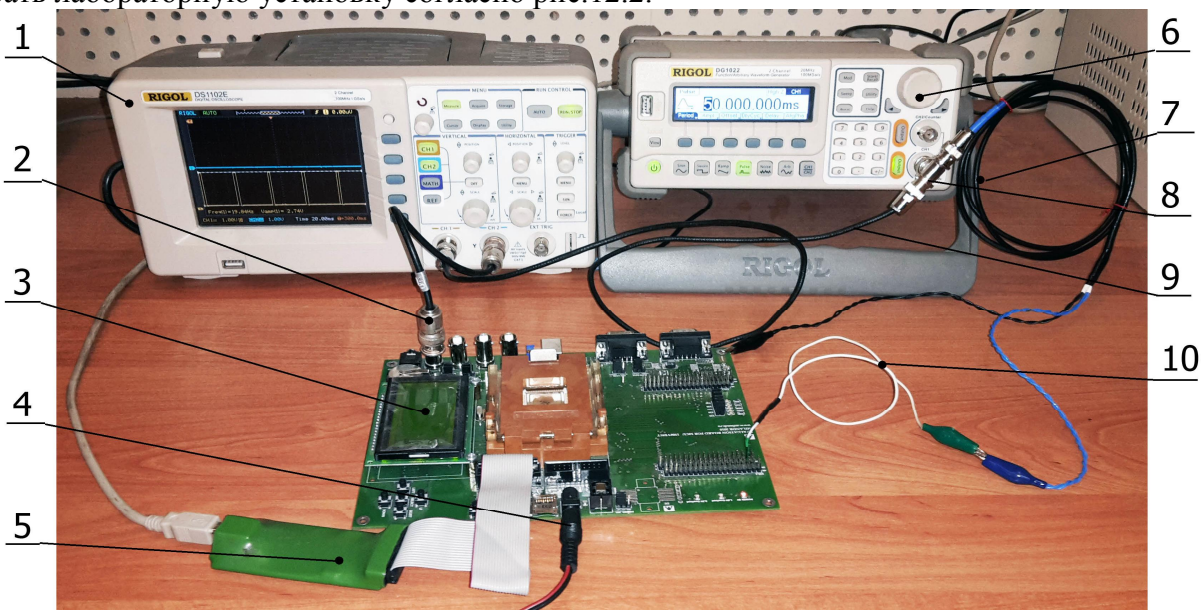


Рисунок 12.1 Общий вид лабораторной установки: 1 – осциллограф; 2,9 – коаксиальный кабель для соединения двух BNC выходов; 3 – отладочная плата; 4 – шнур питания; 5 - программатор J-Link ARM; 6 – универсальный генератор сигналов; 8 – разветвитель BNC; 9 – осциллографический пробник; 10 – соединительный провод.

2. Произвести коммутацию выходного сигнала генератора тактовых импульсов с отладоной платой разъем X33.1 контакт 38 , а так же с 1 каналом осциллографа. Второй канал осциллографа подключить к выходу ЦАП стенда **DAC_OUT**. Общий вид всей установки представлен на рис.12.1.

3. Включить осциллограф в режиме двух каналов, настроить масштабы по напряжению 2В/дел, по времени 10мс/дел.

4. Установить выходной сигнал генератора тактовых сигналов по настройкам выдаваемыми преподавателем, например, длительность импульса 1-5мс период следования импульсов не более 50мс, амплитуда импульсов 3.0В. Внешний вид сигнала представлен на рисунке 12.2

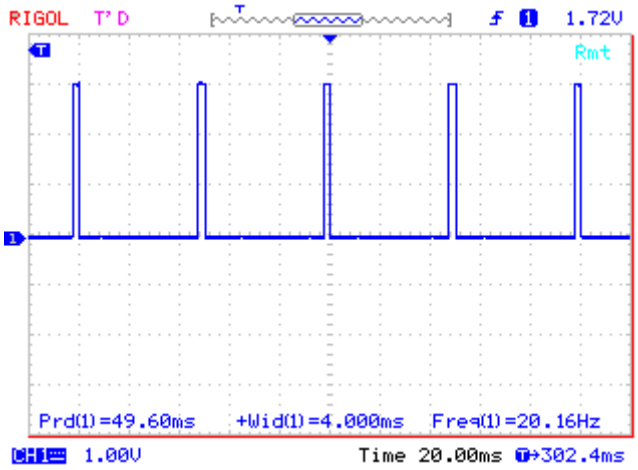


Рисунок 12.2 Осциллограмма передаваемого сигнала

5. Включить отладочную плату.

6. Создайть новый проект, подключить в менеджере ***Manage run-time environment*** к проекту библиотеки Startup_MDR1986BE9x, DAC, PORT, TIMER, RST_CLK, необходимые для работы с ЦАП.

7. Скопировать в main.c код представленный в приложении. Затем скомпилировать проект, произвести программирование МК.

8. Запустить генератор тактовых сигналов в заданном режиме и зафиксировать показания на осциллографе.

9. Проверить, что изменение периода следования импульсов вызывает изменение напряжения на выходе DAC_OUT стенда.

Задания

1) Настроить прием внешних прерываний с другим номером по заданию преподавателя (EXT_INT2, EXT_INT3, EXT_INT4)

2) Настроить одновременный прием 2 внешних прерываний с разными приоритетами или с одинаковыми приоритетами, определить тип приоритетности прерываний (циклический приоритет или вложенный приоритет прерываний).

Сведения для выполнения:

Процедуры обработки прерываний или прерывания (IRQ) вызываются микропроцессором по запросу периферийных устройств. Например, при приеме данных по последовательному порту необходимо их считывание и обработка. Контроллер последовательного порта посылает микропроцессору запрос прерывания в котором сообщает микропроцессору, что данные приняты и их необходимо обработать. Микропроцессор вызывает соответствующую процедуру обработки прерывания. Все прерывания асинхронны по отношению к выполняемым инструкциям.

В данной лабораторной работе рассматривается применение внешних прерываний. Процессор способен обрабатывать прерывания, сформированные по уровню сигнала.

Прерывание такого типа считается активным до тех пор, пока периферийное устройство не снимет активный уровень сигнала запроса. Как правило, это происходит после

соответствующего обращения процедуры обработки прерывания к периферийному устройству.

После того, как процессор передал управление на обработчик, он автоматически снимает признак ожидания обслуживания прерывания (см. раздел “Аппаратное и программное управление прерываниями”). Если прерывание формируется по уровню сигнала, а сигнал запроса не снят до возврата из обработчика, процессор вновь переведет прерывание в состояние ожидания обслуживания, что, в свою очередь, приведет к повторному вызову его обработчика. Таким образом, периферийное устройство может поддерживать сигнал запроса прерывания в активном состоянии до тех пор, пока не перестанет нуждаться в обслуживаний вызываемых периферийными устройствами.

Рассмотрим формирование программного кода для настройки внешних прерываний Микроконтроллер КР1986ВЕ91Т имеет 4 линии внешних прерываний. Срабатывание внешних прерываний происходит по уровню (срабатывание по фронту или спаду не предусмотрено). Для включения внешних прерываний достаточно запрограммировать порт на выполнение альтернативной или переопределенной функции. Более подробно в разделе порты ввода-вывода MDR_PORTx [2].

Вот пример настройки для данной лабораторной работы

```
//Порт PA0 работающий для приёма внешнего прерывания
PORT_InitTypeDef PORTAInit;

void TimePortInit(){
  PORT_StructInit(&PORTAInit); //Загрузка по умолчанию
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTA, ENABLE);
  //Порт PA0 альтернативная функция EXT_INT1. На плате контакт 38 – разъема X33.1
  PORTAInit.PORT_Pin  = PORT_Pin_0;
  PORTAInit.PORT_OE   = PORT_OE_IN;
  PORTAInit.PORT_MODE = PORT_MODE_DIGITAL;
  PORTAInit.PORT_FUNC = PORT_FUNC_ALTER;
  PORT_Init(MDR_PORTA, &PORTAInit);
}
```

Далее необходимо выдать разрешение на прерывание и определить её приоритет.

```
NVIC_EnableIRQ(EXT_INT1_IRQn);//Разрешение на внешнее прерывание
NVIC_SetPriority(EXT_INT1_IRQn, 0);//Приоритет прерывания(0-max)
```

И последнее это определить процедуру обработки прерывания, которая определяет, что будет делать МК, в результате внешнего прерывания. Сигнатура фукции имеет вид: void EXT_INT1_IRQHandler(void){ }.

Приложение

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_timer.h>
#include <MDR32F9Qx_dac.h>
#include <math.h>

//Инициализация порта работающего для ЦАП
PORT_InitTypeDef PORTEInit;
void DACPortInit(){
  PORT_StructInit(&PORTEInit); //Load defaults
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTE, ENABLE);
  PORTEInit.PORT_Pin  = PORT_Pin_9;
  PORTEInit.PORT_OE   = PORT_OE_OUT;
  PORTEInit.PORT_MODE = PORT_MODE_ANALOG;
  PORT_Init(MDR_PORTE, &PORTEInit);
}

//Инициализация ЦАП
void DACInit(){
  RST_CLK_PCLKcmd(RST_CLK_PCLK_DAC, ENABLE);
  RST_CLK_PCLKcmd(RST_CLK_PCLK_RST_CLK, ENABLE);
  DAC1_Init(DAC1_AVCC);
  DAC1_Cmd(ENABLE);
```

```
}

//Порт работающий для приёма события
PORT_InitTypeDef PORTAInit;
void TimePortInit(){
  PORT_StructInit(&PORTAInit); //Load defaults
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTA, ENABLE);
  PORTAInit.PORT_Pin  = PORT_Pin_0;
  PORTAInit.PORT_OE   = PORT_OE_IN;
  PORTAInit.PORT_MODE = PORT_MODE_DIGITAL;
  PORTAInit.PORT_FUNC = PORT_FUNC_ALTER;
// Порт PA0 альтернативное значение EXT_INT1 (Внешнее прерывание). На плате 38 контакт - X33.1
// Внешнее прерывание срабатывает по уровню (лог 1 на входе)
  PORT_Init(MDR_PORTA, &PORTAInit);
}

// Инициализация таймера
// Максимальная длительность импульса 50мс. разрядность ЦАП 12
// Период счета таймера= 50/2^12=0,01220703125мс
// Предделитель таймера при тактовой частоте 8МГц
// (0,01220703125/1000)*8000000=97.6 (примем 98)

TIMER_CntInitTypeDef TIM1Init;
uint16_t time = 0;
void TimerInit(){
  RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER1, ENABLE);
  TIMER_BRGInit(MDR_TIMER1, TIMER_HCLKdiv1);
  TIMER_CntStructInit(&TIM1Init);    //Загрузка структуры параметрами по умолчанию
  TIM1Init.TIMER_Prescaler      = 98;
  TIM1Init.TIMER_Period         = 65535;
  TIMER_CntInit(MDR_TIMER1, &TIM1Init);
  TIMER_Cmd(MDR_TIMER1, ENABLE);
}

// Разрешение внешних прерываний
void EXTinit(){
  NVIC_EnableIRQ(EXT_INT1_IRQn);
  NVIC_SetPriority(EXT_INT1_IRQn, 0);
}

uint16_t timeTec;
uint16_t DAC_value;
// Обработчик внешних прерываний
void EXT_INT1_IRQHandler(void){
timeTec = TIMER_GetCounter(MDR_TIMER1);
DAC_value = timeTec - time;   // Вычисление цифрового кода для выдачи на ЦАП
if(DAC_value>4095){DAC_value=4095;}
time = timeTec;
DAC1_SetData(DAC_value);      //Вывод данных на ЦАП
                              //Цикл ожидание сброса тактового бита
while(PORT_ReadInputDataBit(MDR_PORTA, PORT_Pin_0)==1){};
}

int main(){
       TimePortInit();        // Инициализация порта для внешнего прерывания
       DACPortInit();         // Инициализация порта ввода вывода для ЦАП
       DACInit();             // Инициализация ЦАП
       TimerInit();           // Инициализация таймера
       EXTinit();             // Разрешение внешних прерываний
       while(1){}
}
```

Лабораторная работа №13. Изучение аналогового компаратора

Цель работы:

Изучить структуру и принцип работы аналогового компаратора микроконтроллера 1986ВЕ91Т.

Приборы и материалы:

1. Отладочная плата MDR1986VE91T Rev 4.
2. Программатор J-Link ARM.
3. Блок питания 5В, 1.4А.
4. ПК с установленной средой программирования Keil uVision.
5. Генератор синусоидального напряжения.
6. Осциллограф.
7. Коаксиальный кабель для соединения двух BNC выходов- 2шт., разветвитель BNC, Соединительные провода.

Ход работы:

1. Собрать лабораторную установку согласно рис. 13.1

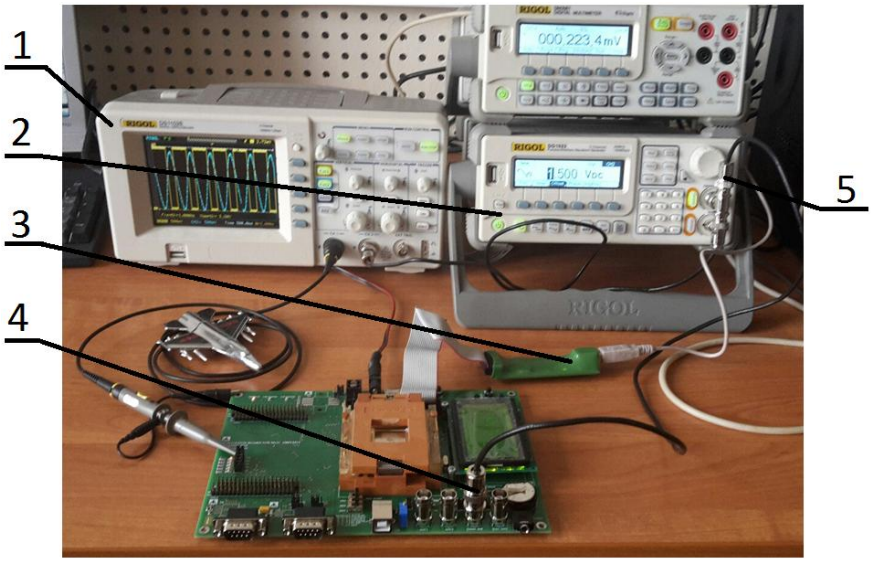


Рисунок 13.1 Общий вид лабораторной установки: 1 – осциллограф; 2 – универсальный генератор сигналов; 3 - программатор J-Link ARM; 4 – отладочная плата; 5 – коаксиальный кабель для соединения двух BNC выходов, разветвитель BNC.

2. Создать проект в среде Keil uVision5 или использовать MDRproject
3. Подключить библиотеки PORT, RST_CLK, COMP в менеджере Manage Run-Time Environment.
4. Заменить текст main.c на текст программы (см. Приложение).
Данная программа сравнивает сигнал с входа IN3 с внутренней шкалой напряжения микроконтроллера. Значение внутренней шкалы установлено 22/32 или 2,27В
5. Скомпилировать и загрузить проект в микроконтроллер.
6. Подключить генератор с помощью коаксиального кабеля ко входу платы COMP_INP и одновременно к осциллографу.
7. Ко второму входу осциллографа подключить сигнал с выхода компаратора порт D10 микроконтроллера контакты X32, или X32.1:24
8. На генераторе выставить синусоидальный сигнал со следующими параметрами: размах 3В; смещение – 1,5В; частота 10Гц.
9. После установки параметров включить работу соответствующего выхода генератора.

10. На осциллографе наблюдаем картину аналогичную рис 13.2.

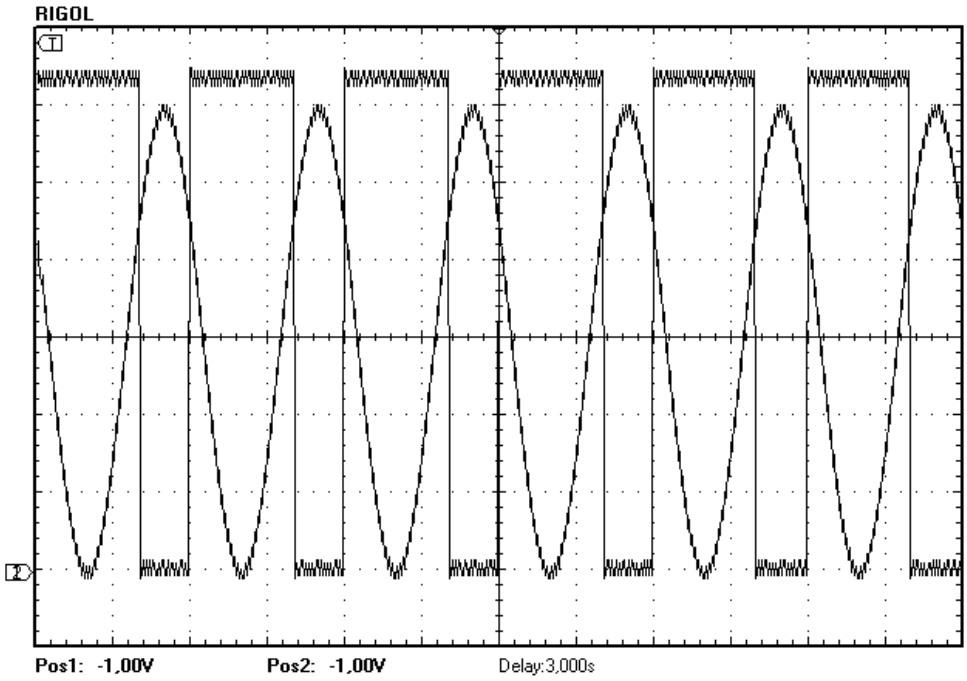


Рисунок 13.2 - Результат работы программы. Входная синусоида и результат ее преобразования в виде прямоугольного импульса.

12. Выполнить индивидуальное задание преподавателя.

Задания

1. Измените напряжение внутренней шкалы по заданию преподавателя.
2. Сравните сигналы с двух каналов генератора IN1, IN2, IN3.
3. Составьте программу с использованием прерывания от компаратора по подсчету импульсов аналогового напряжения.

Сведения для выполнения:

Один из самых простых модулей микроконтроллера- это аналоговый компаратор. Он сравнивает между собой два напряжения и запоминает результат сравнения в регистре. Структура приведена на рис.13.3.

Обычно аналоговый компаратор используется как однобитный АЦП. Например, компаратором можно отслеживать уровень заряда батареи или момент перехода переменного напряжения через ноль. Еще он может быть задействован для измерения длительности аналоговых сигналов.

В микроконтроллере реализована схема компаратора, обеспечивающая следующие режимы работы:

- сравнение двух сигналов с трех различных выводов микросхемы;
- сравнение сигнала с трех различных выводов с внутренней шкалой напряжений;
- сравнение сигнала с вывода IN1 с внутренним источником опорного напряжения;
- формирование внутренней шкалы напряжений от питания микроконтроллера и от внешних выводов.

Для включения компаратора необходимо установить бит ON в 1, Это осуществляется с помощью процедуры `COMP_Cmd(ENABLE)`. Выводы порта E должны быть сконфигурированы как аналоговые и должны быть отключены какие-либо внутренние

подтяжки.

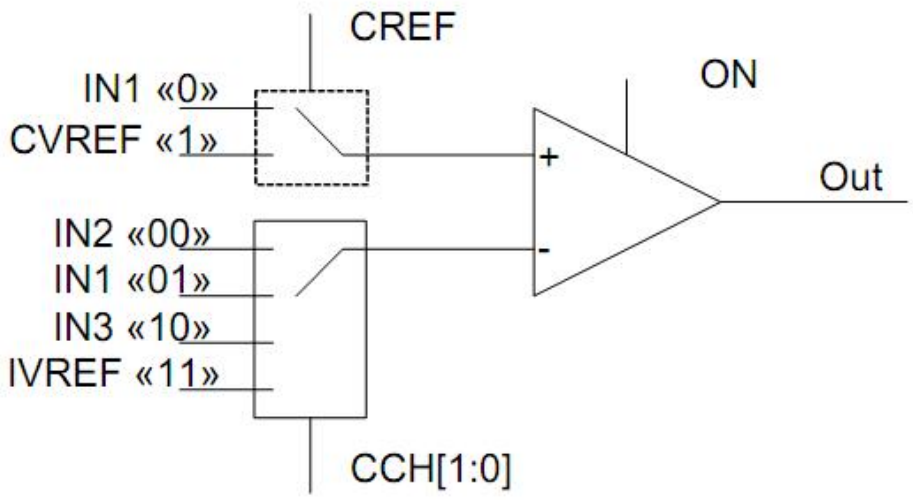


Рисунок 13.3 - Структура мультиплексирования входов компаратора (IVREF – выход внутреннего источника опорного напряжения 1.2 В).

Сравнение внешних сигналов

Компаратор позволяет проводить сравнение двух сигналов, поступающих с трех выводов микросхемы. На вход «+» компаратора может быть подан сигнал IN1 (бит CREF=0), на вход «–» могут быть поданы сигналы IN1 (CCH=01), IN2 (CCH=00) и IN3 (CCH=10), при этом, если уровень на входе «+» будет больше уровня на входе «–», то выход Out установится в 1.

Сравнение сигнала с внутренним источником опорного напряжения

Компаратор позволяет проводить сравнение сигнала, поступающего с вывода IN1 микросхемы, с внутренним источником опорного напряжения IVREF. Для этого на вход «+» компаратора должен быть подан сигнал IN1 (бит CREF=0), на вход «–» должен быть подан сигнал IVREF (CCH=11), при этом, если уровень на входе «+» будет больше уровня на входе «–», то выход Out установится в 1. Кофигурация выводов осуществляется с помощью процедуры COMP_Init(&COMP_InitStructure).

Сравнение внешних сигналов с внутренней шкалой напряжений

Компаратор позволяет проводить сравнение внешних сигналов, поступающих с трех выводов микросхемы, со шкалой напряжений, формируемых внутри микросхемы. На вход «+» компаратора должен быть подан сигнал CVREF (бит CREF=1), на вход «–» могут быть поданы сигналы IN1 (CCH=01), IN2 (CCH=00) и IN3 (CCH=10), при этом, если уровень на входе «+» будет больше уровня на входе «–», то выход Out установится в 1. Настройка компаратора для работы с внутренней шкалой напряжений осуществляептся с помощью процедуры COMP_CVRefInit(&COMP_CVRefInitStructure).

Внутренняя шкала напряжений формируется на резистивном делителе (Рисунок 13.4), который включается битом CVREN=1.

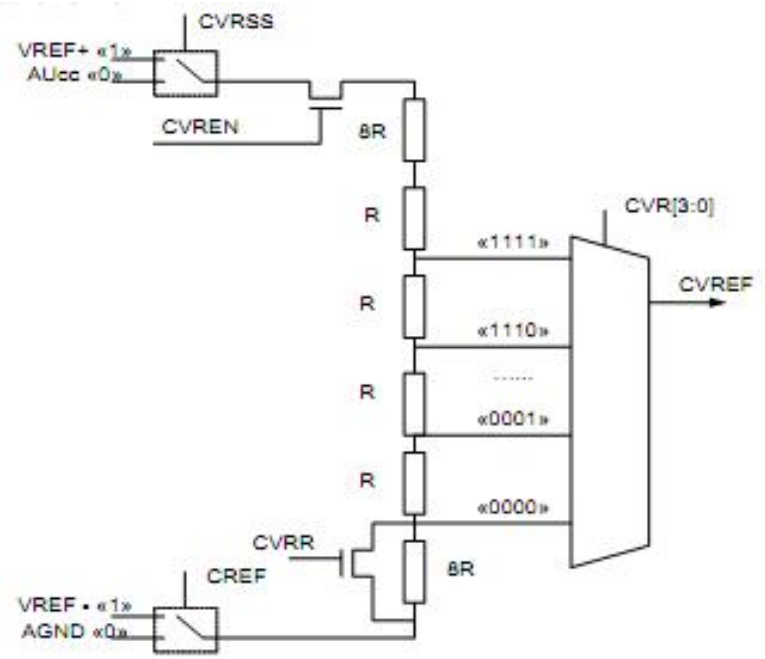


Рисунок 13.4 - Структура блока формирования CVREF

При этом в качестве опорного напряжения делителя может выступать питание микросхемы AUcc (CVRSS = 0), либо напряжение на выводе COMP_VREF+ (CVRSS = 1). Нижнее опорное напряжение компаратора задается на выводе COMP_VREF-. Напряжение на выводе CVREF формируется на основании комбинации бит CVRR и CVR и приведены в Таблица 13.1, как справочные данные. Реальные значения в конкретном кристалле могут отличаться за счет технологического разброса параметров.

Результат работы компаратора сигнал Out может быть проинвертирован с помощью бита INV и выдан на вывод микросхемы OUT_COMP. Также результат сравнения доступен внутри микроконтроллера. Комбинационный сигнал с компаратора отображается в бите Rslt_As (при чтении может быть считан как 1, но при этом не выработать прерывания). Зафиксированный в триггере по тактовой частоте HCLK сигнал сравнения отображается в бите Rslt_Sy. Флаг Rst_lch фиксирует событие появления положительного сигнала сравнения и устанавливается в 1 до тех пор, пока не будет считан регистр COMP_RESULT_LATCH. С помощью функции COMP_GetResultLatch().

Таблица 13.1 – Формирование внутренней шкалы напряжений CVREF

CVRR	CVR[3:0]	Отношение резисторов	Напряжение CREF при U_{CC}=3.3 В, В	Входной импеданс VREF+, Ом	Примечание
0	0000	8/32	0.83	12K	
	0001	9/32	0.93	13K	
	0010	10/32	1.03	13.8K	
	0011	11/32	1.13	14.4K	
	0100	12/32	1.24	15K	
	0101	13/32	1.34	15.4K	
	0110	14/32	1.44	15.8K	
	0111	15/32	1.55	15.9K	
	1000	16/32	1.65	16K	
	1001	17/32	1.75	15.9K	
	1010	18/32	1.86	15.8K	
	1011	19/32	1.96	15.4K	
	1100	20/32	2.06	15K	
	1101	21/32	2.17	14.4K	
	1110	22/32	2.27	13.8K	
	1111	23/32	2.37	12.9K	
1	0000	0/24	0.00	0.5K	
	0001	1/24	0.14	1.9K	
	0010	2/24	0.28	3.7K	
	0011	3/24	0.41	5.3K	
	0100	4/24	0.55	6.7K	
	0101	5/24	0.69	7.9K	
	0110	6/24	0.83	9K	
	0111	7/24	0.96	9.9K	
	1000	8/24	1.10	10.7K	
	1001	9/24	1.24	11.3K	
	1010	10/24	1.38	11.7K	
	1011	11/24	1.51	11.9K	
	1100	12/24	1.65	12K	
	1101	13/24	1.79	11.9K	
	1110	14/24	1.93	11.7K	
	1111	15/24	2.06	11.3K	

Процедуры для работы с компаратором находятся в файле библиотеки MDR32F9Qx_comp.c, например:

void COMP_CVRefSourceConfig(uint32_t Source) – Инициализация источников опорного напряжения для внутренней шкалы.

void COMP_CVRefScaleConfig(uint32_t Scale) – Инициализация врутренней шкалы

void COMP_CVRefCmd(FunctionalState NewState) – Включение внутренней шкалы

void COMP_CVRefInit(const COMP_CVRefInitTypeDef* COMP_CVRefInitStruct) – Полная инициализация внутренней шкалы.

Приложение

```
#include "MDR32F9Qx_port.h"
#include "MDR32F9Qx_rst_clk.h"
#include "MDR32F9Qx_comp.h"

PORT_InitTypeDef PORT_InitStructure;
COMP_InitTypeDef COMP_InitStructure;
COMP_CVRefInitTypeDef COMP_CVRefInitStructure;
```

```
int main(void){
  //Включение тактирования порта E
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTE,ENABLE);

  //Заполнение структуры для инициализации порта
  PORT_InitStructure.PORT_PULL_UP = PORT_PULL_UP_OFF;
  PORT_InitStructure.PORT_PULL_DOWN = PORT_PULL_DOWN_OFF;
  PORT_InitStructure.PORT_MODE = PORT_MODE_ANALOG;

  //Конфигурируем ножку 8 порта E как IN3 компаратора
  PORT_InitStructure.PORT_OE = PORT_OE_IN;
  PORT_InitStructure.PORT_Pin = PORT_Pin_8;
  PORT_Init(MDR_PORTE, &PORT_InitStructure);

  //Включение тактирования порта D
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTD,ENABLE);
  PORT_StructInit(&PORT_InitStructure);

  //Заполнение структуры для инициализации порта
  PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL;

  //Конфигурируем ножку 10 порта D как выход
  PORT_InitStructure.PORT_OE = PORT_OE_OUT;
  PORT_InitStructure.PORT_Pin = PORT_Pin_10;
  PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
  PORT_Init(MDR_PORTD, &PORT_InitStructure);

  //Включаем тактирование компаратора
  RST_CLK_PCLKcmd(RST_CLK_PCLK_COMP,ENABLE);

  //Инициализация структуры значениями по умолчанию
  COMP_StructInit(&COMP_InitStructure);

  //Заполнение структуры инициализации компаратора
  COMP_InitStructure.COMP_PlusInputSource = COMP_PlusInput_CVREF;
  COMP_InitStructure.COMP_MinusInputSource = COMP_MinusInput_IN3;

  //Инициализация аналогового компаратора
  COMP_Init(&COMP_InitStructure);

  //Инициализация структуры опорного напряжения значениями по умолчанию
  COMP_CVRefStructInit(&COMP_CVRefInitStructure);

  //Заполнение элементов структуры для инициализации источника опорного напряжения
  COMP_CVRefInitStructure.COMP_CVRefSource = COMP_CVREF_SOURCE_AVdd;
  COMP_CVRefInitStructure.COMP_CVRefRange = COMP_CVREF_RANGE_Up;
  COMP_CVRefInitStructure.COMP_CVRefScale = COMP_CVREF_SCALE_22_div_32 ;

  //Инициализация источника опрного напряжения
  COMP_CVRefInit(&COMP_CVRefInitStructure);

  //Разрешение работы аналогового компаратора
  COMP_Cmd(ENABLE);

  //Проверка готовности аналогового компаратора
  while(COMP_GetCfgFlagStatus(COMP_CFG_FLAG_READY) != SET);

  //Разрешение работы источника опорного напряжения
  COMP_CVRefCmd(ENABLE);
  while (1)
  { //Проверяем флаг компаратора и устанавливаем ножку 10 порта D по результату сравнения.
    if (COMP_GetFlagStatus(COMP_STATUS_FLAG_AS)){
      PORT_SetBits(MDR_PORTD, PORT_Pin_10);
    }
    else {
      PORT_ResetBits(MDR_PORTD, PORT_Pin_10);
    }
  }
}
```

Лабораторная работа №14. Изучение часов реального времени

Цель работы: Изучение применения КР1986ВЕ91Т в качестве таймера с использованием встроенной схемы часов реального времени.

Приборы и материалы:

1. Отладочная плата MDR1986VE91T Rev 4.
2. Программатор J-Link ARM.
3. Блок питания 5В, 1.4А.
4. ПК с установленной средой программирования Keil uVision.
5. Осциллограф.

Ход работы:

1. Подключите программатор к отладочной плате MDR1986VE91T Rev 4.
2. Создать проект в среде Keil uVision5 или использовать MDRproject
3. Подключить библиотеки PORT, RST_CLK, BKP в менеджере Manage Run-Time Environment.
4. Заменить текст main.c на текст программы (см. Приложение). Эта программа включает часы реального времени с периодом срабатывания около 1 секунды, а также запускает таймер на 1 минуту
5. Скомпилировать и загрузить проект в микроконтроллер.
6. Результат работы представлен на рисунке. На Рис. 14.1 видны импульсы 1 раз в секунду появляющиеся по сигналам часов реального времени, а также виден сигнал таймера ALARM через 60 секунд

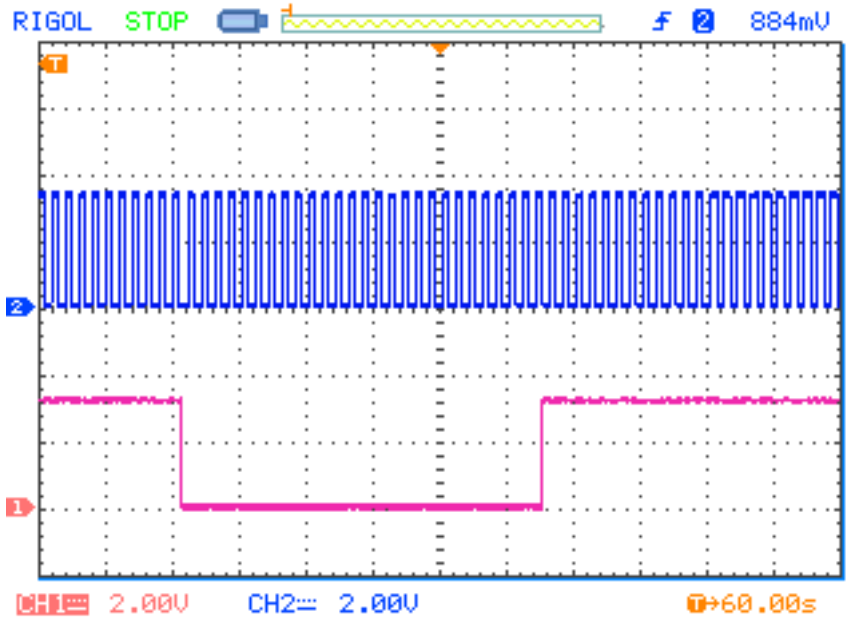


Рис 14.1. Результат работы программы CH1-сигнал ALARM (флаг BKP_RTC_FLAG_ALRF), CH2-сигнал секунд (флаг BKP_RTC_FLAG_SECF).

Задания

1. Измените источник тактирования на HSI, LSI.
2. Измените время срабатывания таймера на заданное преподавателем.

Сведения для выполнения:

Рассмотрим батарейный домен и часы реального времени MDR_BKP Блок батарейного домена предназначен для обеспечения функций часов реального времени и сохранения некоторого набора пользовательских данных при отключении основного источника питания. При снижении питания U_{CC} в блоке SW происходит автоматическое переключение питания BDU_{CC} с U_{CC} на BU_{CC}. Если на BU_{CC} имеется отдельный источник питания (батарейка), то батарейный домен остается включенным и может выполнять свои функции.

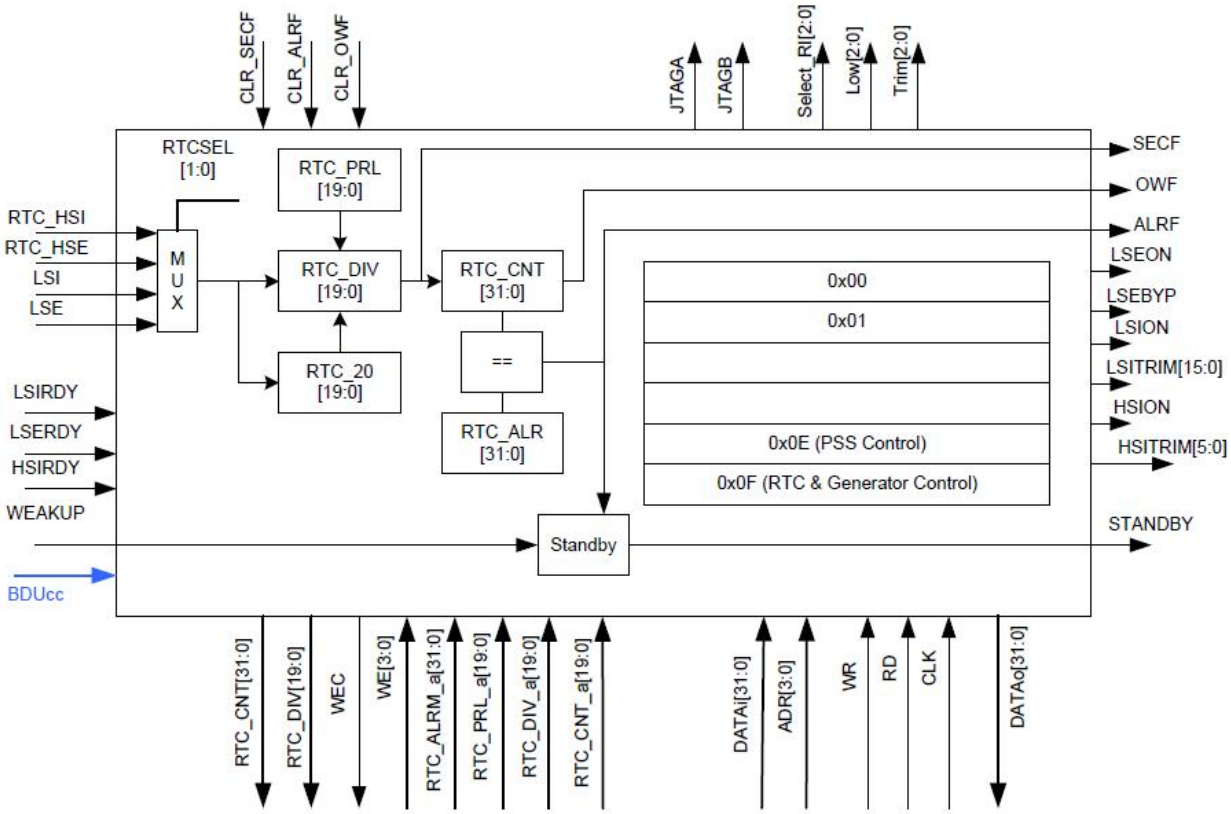


Рисунок 14.2. Структурная блок-схема батарейного домена
и часов реального времени

Часы реального времени

Часы реального времени позволяют организовать механизм отсчета времени в кристалле, в том числе при отключении основного источника питания. Включение часов реального времени осуществляется битом RTCEN. В качестве источника тактовой частоты часов реального времени может выступать генератор LSI, или осциллятор LSE, или HSE, или HSI с дополнительным делителем до 256 (HSE и HSI формируются в блоке управления тактовыми частотами и могут быть выбраны только при наличии питания DU_{CC}, LSI может быть выбран при наличии питания U_{CC}, LSE может быть выбран при наличии U_{CC} или BU_{CC}). Выбор между источниками осуществляется битами RTCSEL. При возможном отключении основного источника питания U_{CC} в качестве источника тактовой частоты должен использоваться осциллятор LSE, так как он также имеет питание BDU_{CC}. Биты управления осциллятором LSE расположены в батарейном домене и таким образом при отключении основного питания они не сбрасываются.

Для калибровки тактовой частоты используются биты CAL[6:0]. Значение CAL определяет, какое число тактов из 2^{20} будет замаскировано. Таким образом, с помощью бит CAL[6:0] производится замедление хода часов. Изменение значения бит CAL может быть осуществлено в ходе работы часов реального времени.

Регистр RTC_DIV выступает в роли 20-битного предварительного делителя входной тактовой частоты, таким образом, чтобы на его выходе была тактовая частота в 1 Гц. Для задания коэффициента деления регистра RTC_DIV используется регистр RTC_PRL.

Регистр RTC_CNT предназначен для отсчета времени в секундах и работает на выходной частоте делителя RTC_DIV. Регистр RTC_ALR предназначен для задания времени, при совпадении с которым вырабатывается флаг прерывания и пробуждения процессора.

Таким образом, бит STANDBY, отключающий внутренний регулятор напряжения, автоматически сбрасывается при совпадении RTC_CNT и RTC_ALR.

Бит STANDBY также может быть сброшен с помощью вывода WAKEUP.

Регистры аварийного сохранения

Батарейный домен имеет 16 встроенных 32-х разрядных регистров аварийного сохранения. 16-й и 15-й регистры служат для хранения бит управления батарейным доменом, оставшиеся 14 регистров могут быть использованы разработчиком программы.

Приложение

```
#include "MDR32F9Qx_port.h"
#include "MDR32F9Qx_rst_clk.h"
#include "MDR32F9Qx_bkp.h"
//Выбираем тактирование от внешнего источника HSE
#define RTC_HSE_CLK
//Начальное значение счетчика
#define COUNT_VALUE          0
//Значение основного счетчика для выдачи сигнала ALRF
#define ALARM_VALUE          60
//Основание счета для предварительного делителя при использовании
//источников тактовых импульсов HSE HSI
#define PRESC_VALUE_HS 1000000
//Основание счета для предварительного делителя при использовании
//источников тактовых импульсов LSI
#define PRESC_VALUE_LS 32000
//Переменная для инициализации портов
PORT_InitTypeDef PORT_InitStructure;
//Процедура обработки прерывание схемы
void BACKUP_IRQHandler(void){
  //Проверка флага SECF и морганием светодиодом раз в секунду
  if (BKP_RTC_GetFlagStatus(BKP_RTC_FLAG_SECF)==SET){
    if (PORT_ReadInputDataBit(MDR_PORTD,PORT_Pin_10)==0){PORT_SetBits(MDR_PORTD,PORT_Pin_10);}
    else {PORT_ResetBits(MDR_PORTD,PORT_Pin_10);}
  }
  //Проверка флага ALRF и включение светодиода при совпадении основного счетчика и ALARM_VALUE
  if (BKP_RTC_GetFlagStatus(BKP_RTC_FLAG_ALRF)==SET){
    PORT_SetBits(MDR_PORTD,PORT_Pin_11);
  }
  BKP_RTC_ITConfig(BKP_RTC_CS_SECF | BKP_RTC_CS_ALRF, ENABLE);     //Очистка флагов
}
int main(void){
//Инициализация светодиодов
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTD, ENABLE);
  PORT_StructInit(&PORT_InitStructure);
  PORT_InitStructure.PORT_Pin = (PORT_Pin_All & ~(PORT_Pin_10 | PORT_Pin_11));
  PORT_Init(MDR_PORTD, &PORT_InitStructure);
  PORT_InitStructure.PORT_Pin   = (PORT_Pin_10 | PORT_Pin_11);
  PORT_InitStructure.PORT_OE    = PORT_OE_OUT;
  PORT_InitStructure.PORT_FUNC  = PORT_FUNC_PORT;
  PORT_InitStructure.PORT_MODE  = PORT_MODE_DIGITAL;
  PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
  PORT_Init(MDR_PORTD, &PORT_InitStructure);
//Включение тактирования MDR_BKP
  RST_CLK_PCLKcmd(RST_CLK_PCLK_BKP,ENABLE);
  BKP_RTC_Reset(ENABLE);
  BKP_RTC_Reset(DISABLE);
//Выполняется в случае выбора внутреннего источника тактовых импульсов
#ifdef RTC_HSI_CLK
```

```
  RST_CLK_HSIadjust(25);
  RST_CLK_RTC_HSIclkEnable(ENABLE);
  RST_CLK_HSIclkPrescaler(RTCHS_PRESC);
  BKP_RTCclkSource(BKP_RTC_HSIclk);
#endif
//Выполняется в случае выбора высокочастотного внешнего источника тактовых импульсов
#ifdef RTC_HSE_CLK
  RST_CLK_HSEconfig(RST_CLK_HSE_ON);
  while (RST_CLK_HSEstatus()!=SUCCESS);
  RST_CLK_RTC_HSEclkEnable(ENABLE);
//Установка делителя частоты HSE равного 8
  RST_CLK_HSEclkPrescaler(RST_CLK_HSEclkDIV8);
  BKP_RTCclkSource(BKP_RTC_HSEclk);
#endif
//Выполняется в случае если выбран внутренний источник LSI
#ifdef RTC_LSI_CLK
  RST_CLK_LSIadjust(12);
  BKP_RTCclkSource(BKP_RTC_LSIclk);
  while (RST_CLK_LSIstatus()!=SUCCESS);
#endif
//Разрешаем прерывание схемы BKP в случае установки флагов ALRF, SECF и OWF
  BKP_RTC_ITConfig(BKP_RTC_IT_ALRF | BKP_RTC_IT_SECF | BKP_RTC_IT_OWF,ENABLE);
  BKP_RTC_WaitForUpdate();//Ждем завершение записи в регистр BKP
  BKP_RTC_SetCounter(COUNT_VALUE); //Обнуляем основной счетчик
  BKP_RTC_WaitForUpdate();//Ждем пока обновится
//В зависимости от выбранного источника тактирования устанавливаем основание счета
//для предварительного делителя часов реального времени
#ifdef RTC_HSI_CLK
  BKP_RTC_SetPrescaler(PRESC_VALUE_HS);
#endif
#ifdef RTC_HSE_CLK
  BKP_RTC_SetPrescaler(PRESC_VALUE_HS);
#endif
#ifdef RTC_LSI_CLK
  BKP_RTC_SetPrescaler(PRESC_VALUE_LS);
#endif
  BKP_RTC_WaitForUpdate();
//Устанавливаем срабатывание сигнала ALRF через ALARM_VALUE=60 импульсов основного счетчика
  BKP_RTC_SetAlarm(ALARM_VALUE);
  BKP_RTC_WaitForUpdate();
  BKP_RTC_Enable(ENABLE);  //Разрешаем работу BKR
//Обнуляем флаги ALRF и SECF путем записи в них единицы
  BKP_RTC_ITConfig(BKP_RTC_CS_SECF | BKP_RTC_CS_ALRF, ENABLE);
  NVIC_EnableIRQ(BACKUP_IRQn); //Разрешаем прерывание от схемы BKP

  while(1){}; //В бесконечном цикле ждем прерывание
}//main();
```

Лабораторная работа №15: Сторожевой таймер IWDG микроконтроллера 1986BE91T

Цель работы: Изучение работы сторожевого таймера в микроконтроллере 1986BE91T, создание проекта с использованием сторожевого таймера.

Приборы и материалы:

1. Отладочная плата MDR1986VE91T Rev 4.
2. Программатор J-Link ARM.
3. Блок питания 5В, 1.4А.
4. ПК с установленной средой программирования Keil uVision.
5. Осциллограф.

Ход работы:

1. Подключите программатор к отладочной плате MDR1986VE91T Rev 4.
2. Создать проект в среде Keil uVision5 или использовать MDRproject
3. Подключить библиотеки PORT, RST_CLK, IWDG в менеджере Manage Run-Time Environment.

4. Заменить текст main.c на текст программы (см. Приложение). 5. Скомпилировать и загрузить проект в микроконтроллер.
6. Результат работы представлен на рисунке 15.1. На Рис (канал CH1 осциллогафа) видны импульсы 1 раз в секунду говорящие о том что программа выполняет бесконечный цикл while(1), а также виден (канал CH2 осциллографа) импульс который генерируется при перезагрузке микроконтроллера через 13 сек.

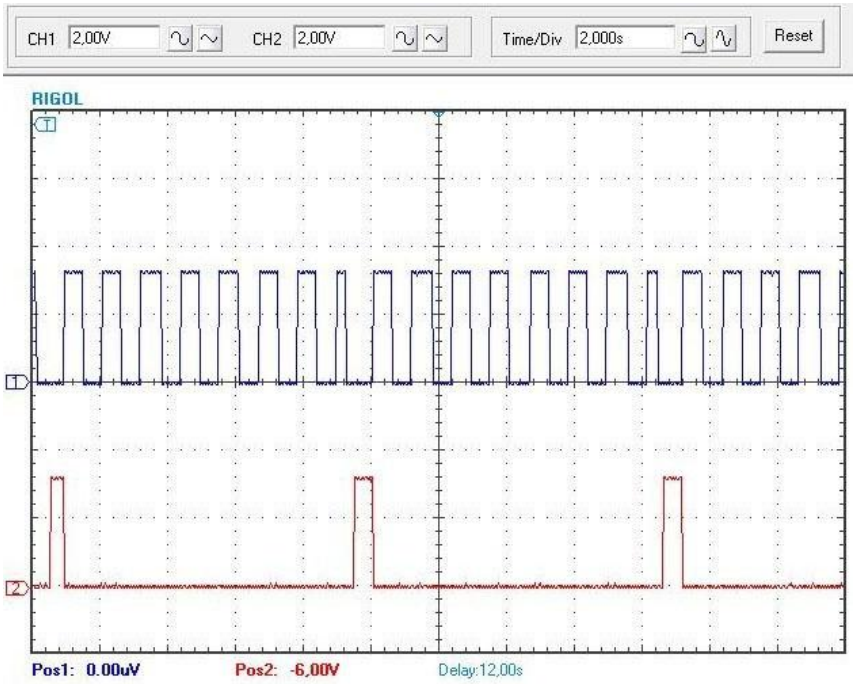


Рисунок 15.1 Результат работы программы – периодическая перезагрузка МК: CH1(Порт D10) – переключение светодиода внутри бесконечного цикла while(1); CH2 (Порт D11)– переключение светодиода при перезагрузке (инициализации МК)

На рис.2 приведена осциллограмма работы сторожевого таймера IWDG. На осциллографе имеются 2 канала подключения, которые видно на осциллограмме. Первый канал подключен к выводу 10 порта D, второй канал к выводу 11 соответственно.График первого канала показывает работу процессора до срабатывания сторожевого таймера IWDG. График второго канала показывает срабатывание сторожевого таймера IWDG и перезагрузку микроконтроллера.

Задания:

1. Измените период срабатывания таймера на время заданное преподавателем.
2. Создайте программу проверки человека оператора которая сбрасывает таймер по нажатию кнопки. То есть если человек перестанет периодически нажимать кнопку то сработает перезагрузка микроконтроллера.

Сведения для выполнения:

Сторожевые таймеры могут быть подходящим решением для встраиваемых систем, которые не могут постоянно находиться под присмотром человека.

Большинство встраиваемых систем должны полагаться на свои силы. Во-первых, если программное обеспечение зависло, не всегда существует возможность дождаться того, кто бы его перезапустил. Во-вторых, некоторые из устройств, например такие, как космические зонды, попросту не достижимы для людей-операторов. И, в-третьих, скорость, с которой оператор может перезагрузить систему, может быть слишком низкой, чтобы удовлетворить временным требованиям по приведению изделия в рабочее состояние.

Сторожевой таймер (*контрольный таймер*, англ. *Watchdog timer* — букв. «сторожевая собака») — аппаратно реализованная схема контроля над зависанием системы. Представляет собой таймер, который периодически сбрасывается контролируемой системой. Если сброса не произошло в течение некоторого интервала времени, происходит принудительная перезагрузка системы. В некоторых случаях сторожевой таймер может посылать системе сигнал на перезагрузку («мягкая» перезагрузка), в других же — перезагрузка происходит аппаратно (замыканием сигнального провода RST или подобного ему).[9]

Особенностью функционирования МК-систем управления является то, что они практически всегда работают в автономном режиме (без участия или контроля со стороны человека- оператора). Причем МК-системы зачастую формируют коды управляющих воздействий для исполнительных механизмов и ошибки, а тем более неконтролируемые действия могут стоить очень дорого, поэтому они должны быть исключены независимо от вариантов развития событий.

Например, вполне реальными представляются следующие ситуации: МК настроен на обработку некоторой совокупности данных, при этом возможно возникновение их сочетания, которое не предусмотрено программистом, что может привести к зацикливанию программы. Или на практике достаточно распространенным является случай сбоя программного кода при перепадах напряжения, что также может привести к непредвиденным ситуациям. Наиболее разумным выходом в данных и им подобных ситуациях является перезапуск программного кода (перезапуск контроллера), который должен быть выполнен без участия человека.

В принципе эту работу может выполнить один из стандартных таймеров счетчиков МК, однако здесь есть свои особенности. Во-первых стандартные Т/С необходимы для решения задач управления, а кроме того, это устройство должно тактироваться отдельным генератором, для того, чтобы иметь возможность управлять выходом из различных режимов энергосбережения.

Микроконтроллер 1986ВЕ91Т имеет независимый сторожевой таймер (IWDG) основаный на 12-битном уменьшающем свое значение счетчике и 8-битном предварительном делителе частоты. Его частота равна 40 кГц, задается независимым внутренним RC-генератором, и поскольку этот генератор работает независимо от главного генератора, таймер IWDG может работать в режиме остановки (Stop) и резервном режиме (Standby). Он может использоваться как таймер сброса устройства в случае возникновения проблем или как свободно запускаемый таймер для создания временных задержек. Он может настраиваться как аппаратно, так и программно через устанавливаемые байты. Счетчик может быть остановлен в режиме отладки.

Таймер IWDG тактируется специализированным низкочастотным сигналом LSI, благодаря чему он продолжает работу, даже если пропадает системный тактовый сигнал. Таймер осуществляет обратный отсчёт от заданного значения до нуля. Сброс процессора таймер производит при достижении счётчиком таймера нуля [8].

Этот таймер лучше всего подходит для тех приложений, которым необходимо, чтобы сторожевой таймер был запущен как полностью независимый процесс, вне основного приложения, но у которого были бы не слишком высокие требования к временным параметрам.

Таймер IWDG запускается путём записи значения OxCCCC в ключевой регистр IWDG_KR с помощью процедуры `IWDG_Enable();`. Счётчик таймера начинает обратный отсчёт от значения, заданного в регистре перезагрузки IWDG_RLR. По умолчанию этот регистр имеет значение OxFFF. Когда счётчик досчитает до нуля, формируется сигнал сброса для процессора от таймера IWDG.

Каждый раз, когда в ключевой регистр IWDG_KR записывается значение OxAAAA, данные из регистра IWDG_RLR перегружаются в счётчик, что возобновляет отсчёт и предотвращает сброс процессора от IWDG (процедура `IWDG_ReloadCounter();`).

Если с помощью соответствующих битов конфигурации разрешена функция «аппаратный сторожевой таймер», то сторожевой таймер автоматически запускается после подачи питания.

Регистр предварительного делителя IWDG_PR позволяет программно задать коэффициент деления тактовой частоты для сторожевого таймера IWDG. Настройка происходит с помощью процедуры `IWDG_SetPrescaler(IWDG_Prescaler_128)`. В нашем случае коэффициент деления равен 128.

По умолчанию регистры IWDGPR и IWDGRLR защищены от записи. Прежде чем изменить их значения, необходимо записать код 0x5555 (процедурой `IWDG_WriteAccessEnable();`) в регистр IWDG_KR. Запись в ключевой регистр любого другого значения снова заблокирует запись в регистры IWDGPR и IWDGRLR.
Время срабатывания таймера вычисляется при помощи формулы: T=IWDG_PR* IWDG_RLR/40000 с. В нашем примере составляет около 13 с. [8].

В связи с нестабильностью внутреннего RC генератора микроконтроллера, указанные в таблице значения времени могут незначительно отличаться от реальных.

Приложение:

```
#include "MDR32F9Qx_port.h"
#include "MDR32F9Qx_rst_clk.h"
#include "MDR32F9Qx_iwdg.h"
//Переменная для инициализации порта
PORT_InitTypeDef PORT_InitStructure;
//Определение функции задержки
void Delay(__IO uint32_t nCount);
//Включение светодиода
void LEDOn(uint32_t LED_Num){PORT_SetBits(MDR_PORTD, LED_Num);}
//Выключение светодиода
void LEDOff(uint32_t LED_Num){PORT_ResetBits(MDR_PORTD, LED_Num);}
//Единичное моргание светодиода с задержкой del
void BlinkLED1(uint32_t num, uint32_t del){
  uint32_t cnt;
  for ( cnt = 0; cnt < num; cnt++){
    LEDOn(PORT_Pin_10);
    Delay(del);
    LEDOff(PORT_Pin_10);
    Delay(del);
  }
}
int main(void){
  //Включение внутреннего генератора LSI
  RST_CLK_LSIcmd(ENABLE);
  while (!(RST_CLK_LSIstatus()==SUCCESS)){;};
  //Включение тактирования порта D
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTD, ENABLE);
  //Сброс настроек порта D для уменьшения энергопотребления
  PORT_StructInit(&PORT_InitStructure);
  PORT_InitStructure.PORT_Pin = (PORT_Pin_All & ~(PORT_Pin_10|PORT_Pin_11));
  PORT_Init(MDR_PORTD, &PORT_InitStructure);
  //Настройка порта D для работы со светодиодами pin 10, pin 11
  PORT_InitStructure.PORT_Pin   = PORT_Pin_10|PORT_Pin_11;
  PORT_InitStructure.PORT_OE    = PORT_OE_OUT;
  PORT_InitStructure.PORT_FUNC  = PORT_FUNC_PORT;
  PORT_InitStructure.PORT_MODE  = PORT_MODE_DIGITAL;
  PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
  PORT_Init(MDR_PORTD, &PORT_InitStructure);
  Delay(100000);
  // При включении моргаем светодиодом 11
  LEDOn(PORT_Pin_11);
  // Включение тактирования сторожевого таймера IWDG
  RST_CLK_PCLKcmd(RST_CLK_PCLK_IWDG,ENABLE);
  Delay(300000);
  //Разрешение работы сторожевого таймера IWDG
  IWDG_Enable();
  //Разрешение записи во внутренние регистры сторожевого таймера IWDG_PR,IWDG_RLR
  IWDG_WriteAccessEnable();
  //Настройка предделителя частоты IWDG
  IWDG_SetPrescaler(IWDG_Prescaler_128);
  //Ожидание срабатывания схемы IWDG
```

```
  while (IWDG_GetFlagStatus(IWDG_FLAG_PVU) !=SET)  {  }
  //Установка времени перезагрузки IWDG
  IWDG_SetReload(0xFFF);
  IWDG_WriteAccessDisable();
  LEDOff(PORT_Pin_11);
  //Перезагружаем сторожевой таймер IWDG
  IWDG_ReloadCounter();
  while (1){
    // в основном цикле моргаем светодиодом 1
    BlinkLED1(1,300000);
  }
}
//Определение функции задержки
void Delay(__IO uint32_t nCount)
{
  for (; nCount != 0; nCount--);
}
```

Лабораторная работа №16: Сторожевой таймер WWDG микроконтроллера 1986BE91T

Цель работы:

Изучение работы оконного сторожевого таймера WWDG типа.

Приборы и материалы:

1. Отладочная плана MDR1986VE91T Rev 4
2. Программатор J-Link ARM
3. Блок питания 5В, 1.4А
4. ПК с установленной средой программирования Keil u Vision
5. Цифровой вольтметр

Порядок работы:

1. Собрать лабораторную установку как на рисунке 2.1
2. Создать новый проект.
3. Подключить библиотеки RSTCLK, PORT в менеджере Manage Run-Time Environment.
4. Поскольку в менеджере библиотек нет WWDG. Её необходимо подключить в окне «Project window» правой клавишей мыши как показано на рисунке 15.1 выбрав: Source Group1>>Add Existing Files to Group «Source Group 1».

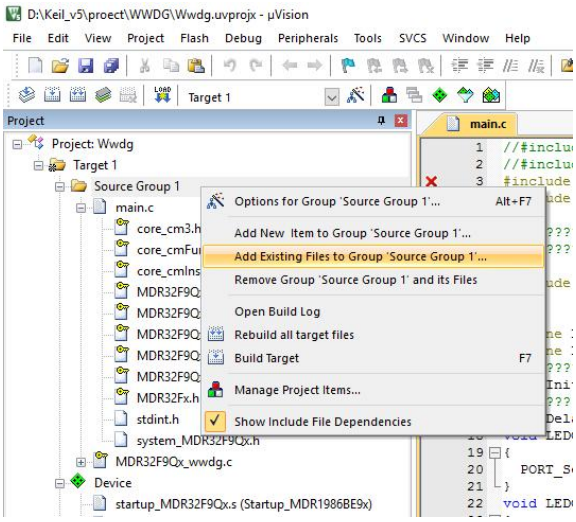


Рисунок 15.1 Подключение библиотек для работы с WWDG.

Необходимо добавить два файла, стандартные пути к ним следующие:

c:\Keil_v5\ARM\PACK\Keil\MDR1986BExx\1.4\Libraries\MDR32F9Qx_StdPeriph_Driver\src\MDR32F9Qx_wwdg.c

c:\Keil_v5\ARM\PACK\Keil\MDR1986BExx\1.4\Libraries\MDR32F9Qx_StdPeriph_Driver\inc\MDR32F9Qx_wwdg.h

5. Из приложения добавить программный код в проект

6. Индикация работы WWDG осуществляют светодиоды LED1 и LED2, располагающиеся около перемычек X32, X34 на плате (см. Рисунок 15.2).

LED2 моргает при перезагрузке или включении микроконтроллера

LED1 моргает в основном цикле while (1)

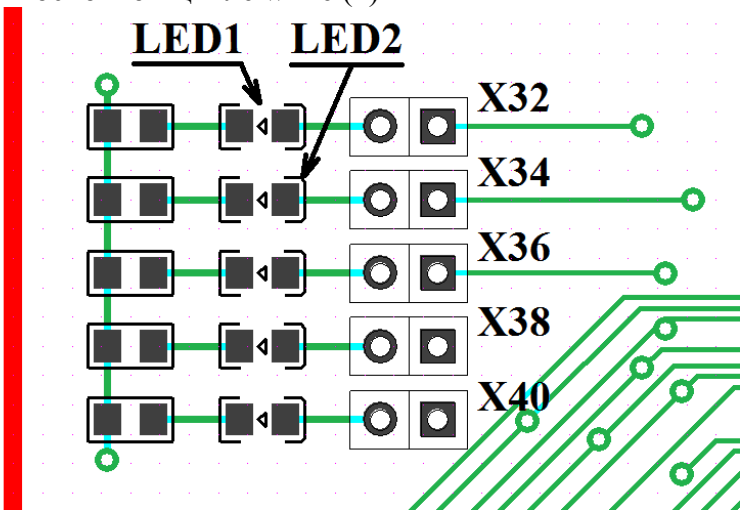


Рисунок 15.2 Расположение диодов на плате

7. Если сторожевой таймер не срабатывает, то моргает только LED1 в основном цикле, если сторожевой таймер срабатывает, то микроконтроллер постоянно перезагружается и от перезагрузки моргает LED2.

Сведения для выполнения.

Начальные сведения по сторожевым таймерам можно почитать в лабораторной работе 14. Оконный сторожевой таймер позволяет произвести сброс микропроцессора если произошёл сбой при выполнении программы. При нормальном выполнении программы оконный сторожевой таймер WWDG должен периодически обновляться. Обновление должно происходить в определённом временном промежутке то есть не раньше чем N тактов после предыдущего обновления и не позже чем через M тактов после предыдущего обновления, то есть в окне обновления (рис 15.3). В программе расположенной в Приложении начальное значение N не устанавливается. Фактически оно равно нулю.

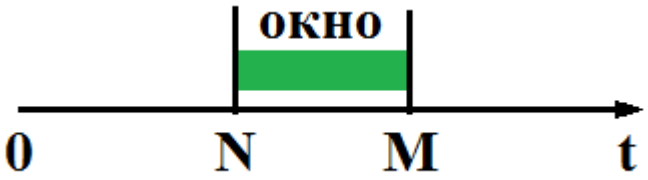


Рисунок 15.3 Временное окно обновления WWDG микроконтроллера 1986BE91T

Это позволяет контролировать работу микропроцессора, то есть задать минимальное и максимальное время выполнения алгоритма и в случае его нарушения выполнять перезагрузку микроконтроллера.

Оконный сторожевой таймер микроконтроллера 1986BE91T аналогичен сторожевому таймеру STM32F103. Рассмотрим реализацию оконного сторожевого таймера.

Таймер тактируется сигналом, полученным делением сигнала тактовой частоты PCLK[12] .Функциональная схема оконного сторожевого таймера WWDG приведена на рисунке 15.4.

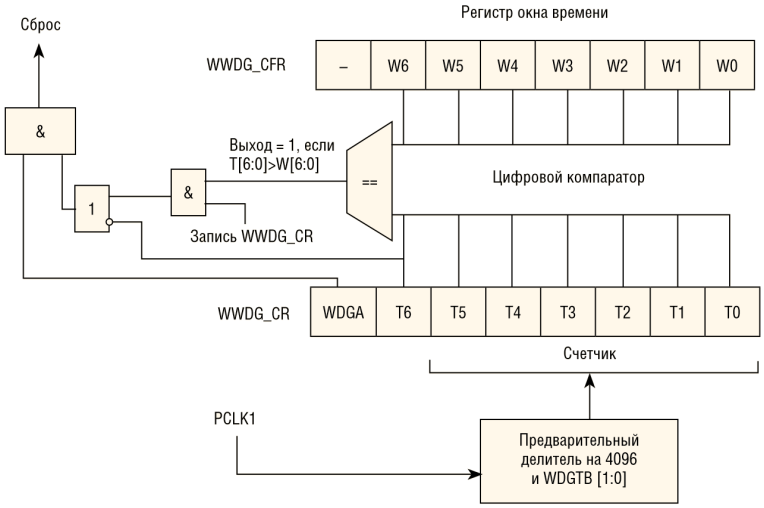


Рисунок 15.4 Функциональная схема оконного сторожевого таймера

Фактически оконный сторожевой таймер является расширенной версией традиционного встраиваемого сторожевого таймера. Оконный сторожевой таймер представляет собой 7-битный вычитающий счётчик, тактируемый сигналом PCLK[12], поделённым на 4096 с помощью аппаратного 12-битного предварительного делителя и программно управляемого делителя от 1 до 8. Оконный сторожевой таймер активируется путём установки бита WDGA в регистре WWDG_CR. С этого момента его нельзя отключить, за исключением общего сброса. После активизации сторожевой таймер начинает счёт в обратном направлении и генерирует сброс при изменении состояния счётчика с 0x40 на 0x3F, то есть при обнулении шестого разряда таймера, условно обозначаемого как T6. Если программа перегружает счётчик, когда его значение больше, чем значение в регистре окна времени, это также инициирует сброс. Чтобы предотвратить сброс процессора, прикладная программа, во время

нормальной операции, должна регулярно производить запись в регистр WWDG_CR. Эта операция должна производиться только тогда, когда значение счётчика меньше, чем значение в регистре окна времени. Значение, которое можно записать в регистр WWDG_CR, должно быть в пределах 0x7F…0x40. Счётчик таймера WWDG является автономным. Он всегда производит обратный отсчёт. Даже тогда, когда сторожевой таймер отключён. При включении сторожевого таймера необходимо установить бит T6 счётчика, чтобы предотвратить немедленный сброс процессора. Биты T[5:0] содержат число, которое представляет собой временную задержку до сброса от WWDG. Регистр конфигурации WWDG_CFR содержит верхний предел окна времени (по умолчанию 7F). Чтобы предотвратить сброс, обратный счётчик должен быть перезагружен тогда, когда его значение меньше, чем значение регистра окна времени, но больше 0x3F. На рисунке 15.5 представлена временная диаграмма, демонстрирующая оконный процесс работы сторожевого таймера.

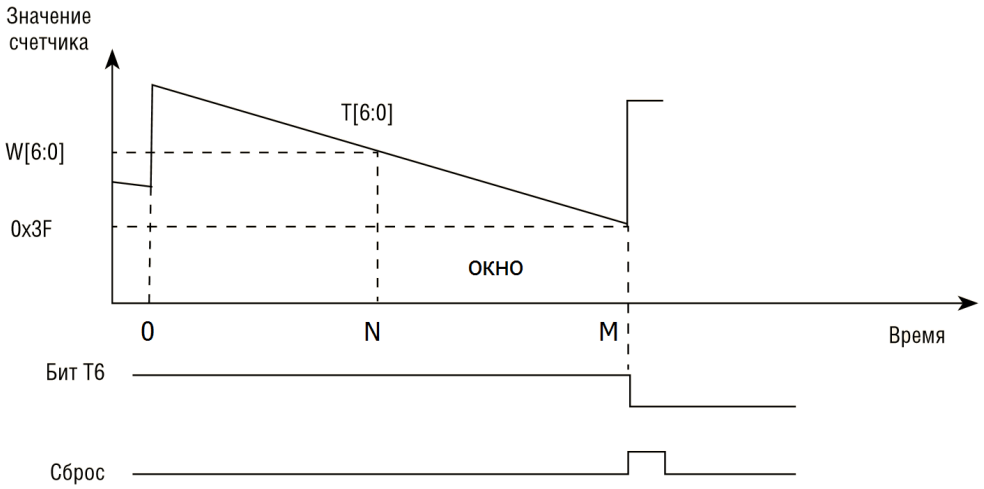


Рисунок 15.5 Временная диаграмма работы оконного сторожевого таймера

Другой способ перезагрузить счётчик состоит в том, чтобы использовать прерывание от раннего пробуждения. Это прерывание разрешается установкой бита EWI в регистре WWDG_CFR. Когда обратный счётчик достигает значения 0x40, генерируется прерывание от раннего пробуждения, и можно использовать соответствующий обработчик, чтобы перезагрузить счётчик и предотвратить сброс от WWDG. Запрос этого прерывания очищается записью нуля в разряд EWIF регистра WWDG_SR. [10]

Рассмотрим некоторые процедуры для работы с оконным сторожевым таймером

1. Разрешение работы и начальная загрузка счетчика (установка регистра CR - биты T и бита WDGA)

```
void WWDG_Enable(uint32_t Counter)
```

2. Периодическая перезагрузка счетчика (просто установка регистра CR - биты T)

```
void WWDG_SetCounter(uint32_t Counter)
```

3. Установка минимального времени окна (N) (установка регистра CFR - биты W)

```
void WWDG_SetWindowValue(uint32_t WindowValue)
```

Задание:

1. В вызове процедуры BlinkLED1 необходимо изменить второй параметр и наблюдать работу микропроцессора при постоянной перезагрузке (по срабатыванию WWDG) и в случае нормальной работы в цикле while(1).
2. Изменить пример (см. Приложение) таким образом, чтобы начальная граница окна N была не равна нулю (рис 15.1). Экспериментально определить временную границу.

Приложение:

```
//  Подключение заголовочных файлов тех библиотек,
// которые непосредственно используются в данном файле исходного кода
#include "MDR32F9Qx_port.h"
#include "MDR32F9Qx_rst_clk.h"
#include "MDR32F9Qx_wwdg.h"
//  Определение ножек к которым подключены светодиоды
#define LED1            PORT_Pin_10
#define LED2            PORT_Pin_11
// Объявление структуры, с помощью которой
// будет происходить инициализация порта
PORT_InitTypeDef PORT_InitStructure;
void Delay(__IO uint32_t nCount); // Определение функции задержки
void LEDOn(uint32_t LED_Num) //  функция включения светодиода
{
  PORT_SetBits(MDR_PORTD, LED_Num);
}
void LEDOff(uint32_t LED_Num) // функция выключения светодиода
{
  PORT_ResetBits(MDR_PORTD, LED_Num);
}
void BlinkLED1(uint32_t num, uint32_t del) // функция моргания светодиодом 1
{
uint32_t cnt;
  for ( cnt = 0; cnt < num; cnt++)
  {
    LEDOn(LED1);
    Delay(del);
    LEDOff(LED1);
    Delay(del);
  }
}
void BlinkLED2(uint32_t num, uint32_t del) // функция моргания светодиодом 2

{
uint32_t cnt;
  for ( cnt = 0; cnt < num; cnt++)
  {
    LEDOn(LED2);
    Delay(del);
    LEDOff(LED2);
    Delay(del);
  }
}

int main(void) // начало программы
{
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTD, ENABLE);  // включение тактирования порта D

  PORT_StructInit(&PORT_InitStructure); // начало инициализации структуры портов
  PORT_InitStructure.PORT_Pin = (PORT_Pin_All & ~(PORT_Pin_10|PORT_Pin_11));
  PORT_Init(MDR_PORTD, &PORT_InitStructure);// инициализация порта D
  PORT_InitStructure.PORT_Pin   = PORT_Pin_10|PORT_Pin_11; // инициализация ножек
  PORT_InitStructure.PORT_OE    = PORT_OE_OUT;
  PORT_InitStructure.PORT_FUNC  = PORT_FUNC_PORT;
  PORT_InitStructure.PORT_MODE  = PORT_MODE_DIGITAL;
  PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
  PORT_Init(MDR_PORTD, &PORT_InitStructure);

  BlinkLED2(1,30000); // моргание светодиодом 2

  RST_CLK_PCLKcmd(RST_CLK_PCLK_WWDG,ENABLE); // включение тактирования WWDG таймера
  // установка предварительного делителя частоты WWDG таймера
  // WWDG counter clock = (PCLK1/4096)/WWDG_Prescaler_8;  (8000000/4096)/8
  WWDG_SetPrescaler(WWDG_Prescaler_8);
  // запись в WWDG начального значения счета и разрешение счета
  WWDG_Enable(0x7F);
  while (1) // Начало основного цикла
  {
   if (WWDG_GetFlagStatus() == SET) // если WWDG прерывания запрещены флаг установлен
    {
      WWDG_ClearFlag();// Очистка флага прерывания
```

```
      BlinkLED1(1,1240);  // моргание светодиодом в основном цикле + задержка
      // увеличение второго параметра приводит к увеличению
      // времени цикла и срабатыванию таймера => постоянная перезагрузка
      // уменьшение второго параметра приводит к уменьшению
      // времени цикла и таймер не успевает срабатывать => микроконтроллер не перезагружается
      WWDG_SetCounter(0x7F); // загружаем WWDG таймер заново
    }
  }
}
// процедура задержки
void Delay(__IO uint32_t nCount)
{
  for (; nCount != 0; nCount--);
}
```

Лабораторная работа №17: Управление асинхронным электродвигателем переменного тока по принципу постоянства отношения V/f

Цель работы: Научится использовать микроконтроллер 1986ВЕ1Т в качестве управляющего микроконтроллера двигателя переменного тока. В качестве алгоритма управления выбрана простая синусоидальная ШИМ модуляция по принципу V/f.

Приборы и материалы:

1. Отладочная плата MDR1986VE91T Rev 4.
2. Программатор J-Link ARM.
3. Блок питания 5В, 1.4А.
4. ПК с установленной средой программирования Keil uVision.
5. Асинхронный электродвигатель.
6. Макетная плата с силовым инвертором.
7. Электродвигатель трехфазный асинхронный электродвигатель мощностью до 5.Ватт напряжением 12 Вольт. Можно использовать и синхронный электродвигатель шпинделя CD ROM без нагрузки. Электродвигатель необходим для визуализации вращения.
8. Осциллограф с логическим анализатором.

Сведения для выполнения

Принцип постоянства отношения V/f применяется для управления асинхронными электродвигателями [12. 13]. Он может использоваться при отсутствии требований к динамическим характеристикам электропривода, таким как время разгона торможения, а также нет необходимости в обеспечении максимального момента на всем диапазоне регулировки частоты вращения. Это позволяет использовать синусоидальную установившуюся модель асинхронного электродвигателя, в которой величина магнитного потока статора пропорциональна отношению амплитуды и частоты напряжения статорной обмотки. Если данное отношение поддерживать на постоянном уровне, то постоянство будет сохранять и магнитный поток статора и, таким образом, вращающий момент будет зависеть только от скольжения. На рис.17.1. представлен линейный график зависимости напряжения от частоты. Более подробно можно почитать в [12,13].

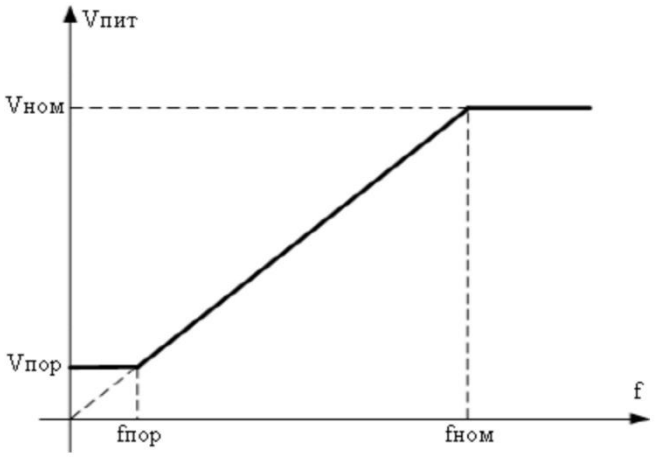


Рисунок 17.1 График зависимости напряжения на обмотке от частоты.

Наклон, прямой на графике определяется по номинальным значениям напряжения питания и частоты питающей сети, которые приводятся в паспорте на электродвигатель, а пороговая частота выбирается по проценту (например, 5%) от номинальной частоты. Грубо говоря, скалярный принцип управления "V/f" заключается в подаче на обмотки электродвигателя 3-фазного синусоидального тока, амплитуда которого пропорциональна частоте, за исключением частот ниже порогового значения и выше номинального, как показано на рисунке 1. Для реализации указанного принципа применяется синусоидальная ШИМ модуляция. Три сдвинутые по фазе на 120 градусов широтно-импульсно модулированные синусоидой последовательности подаются на обмотки электродвигателя UVW. Пример такой последовательности для одной обмотки приведен на рис.17.2

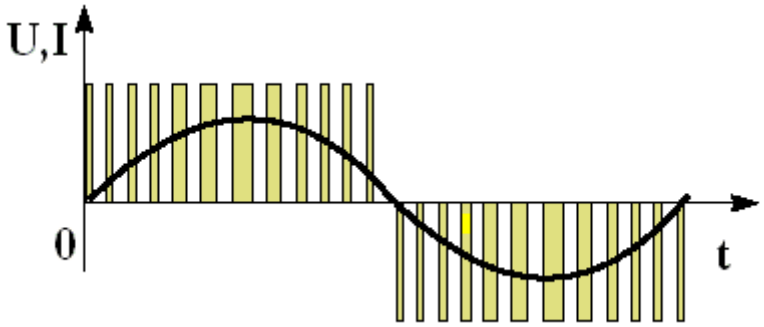


Рисунок 17.2. ШИМ последовательность, формируемая на обмотке и синусоидальный ток обмотки (жирная линия).

Для формирования последовательностей применяется трехфазный инвертор рис. 17.3. Внешний вид ШИМ модулированного сигнала представлен на рис 17.4. Укрупненно ШИМ сигнал представлен на рис 17.5. Как видим напряжение на обмотках симметрично. Это так называемая центрированная ШИМ.

Описание лабораторной установки

Рассмотрим назначение и работу элементов схемы. Подключение макетной платы с силовым инвертором к демонстрационно-отладочной плате 1986BE91T осуществляется по схеме представленной в таблице 17.1:

Таблица17.1 Подключение макетной платы и силовым инвертором к демонстрационно отладочной плате

№	Порт микро-контроллера, цепь	Разъем и контакт платы 1986BE91_EvBrd	Функция порта	Цепь макетной платы (рис17.3)	Комментарий
1	PA1	X33:1.35	TIMER1, CH1	HU	Управление верхним

					ключем фазы U
2	PA2	X33:1.36	TIMER1, CH1N	LU	Управление нижним ключем фазы U
3	PA3	X33:1.31	TIMER1, CH2	HV	-//- фазы V
4	PA4	X33:1.32	TIMER1, CH2N	LV	-//- фазы V
5	PA5	X33:1.29	TIMER1, CH3	HW	-//- фазы W
6	PA8	X33:1.28	TIMER1, CH3N	LW	-//- фазы W
7	GND	X33:1.1		0V	Общий провод

Схема электрическая трехфазного инвертора рис.17.3 состоит из трех одинаковых каналов, формирующих ШИМ сигнал на трех обмотках электродвигателя UVW. Каждый канал имеет в своем составе полумостовой драйвер IR2101 с помощью которого осуществляется управления МОП ключами VT1-VT6 марки (IRF). Конденсатор C1 сглаживает пульсации питающего напряжения. Конденсаторы C2-C4 (bootstrap) и диоды VD1 – VD3 обеспечивают потенциал на базе транзисторов верхних плеч мостов выше уровня питания +12В. При ШИМ модуляции поочередно открываются транзисторы VT1 и VT2. В фазе если открыт транзистор VT2 то конденсатор C2 заряжается через диод VD1. Если транзистор VT2 закрыт то заряд, накопленный в конденсаторе C2 будет расходоваться на открывания транзистора VT1. Это позволяет транзисторам верхних плеч открываться полностью. Ёмкость конденсаторов зависит от частоты ШИМ (чем больше частота, тем меньше ёмкость, более подробно о подборе ёмкости можно узнать из специализированной литературы). Шунтовый резистор R7 сопротивлением 0.5 Ом вырабатывает напряжение пропорциональное току через инвертор. С помощью него мы в дальнейшем будем контролировать потребляемый ток.

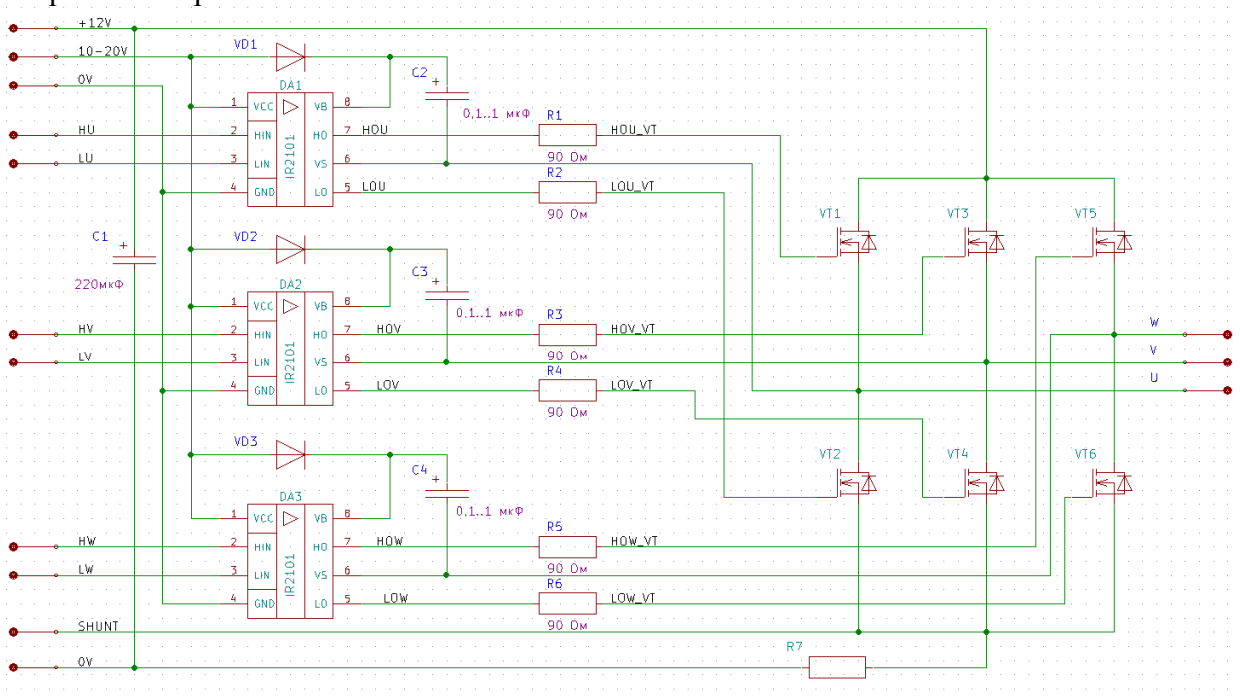


Рисунок 17.3 Электрическая принципиальная схема макетной платы с трехфазным инвертором.

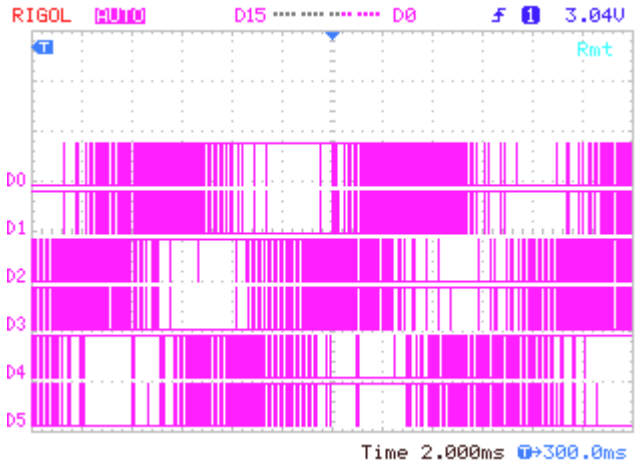


Рисунок 17.4 Осциллограмма управляющих сигналов трехфазной ШИМ. (D0-логический сигнал управления транзистором VT1; D1-VT2; D2-VT3; D3-VT4; D4-VT5; D5-VT6;)

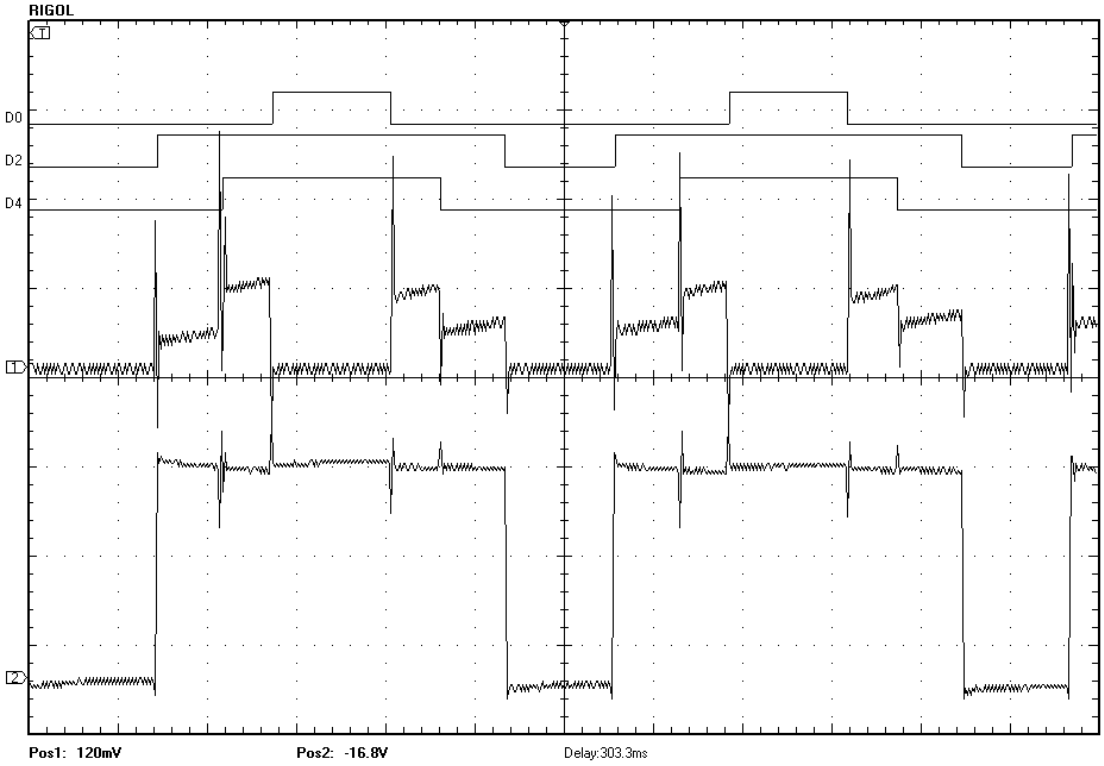


Рисунок 17.5 Осциллограмма трехфазной ШИМ (D0, D2, D4 – Управляющие напряжения ключей обмоток U,V,W соответственно; CH1- Напряжение в цепи SHUNT рис.__(Масштаб-1В/дел) ; CH2- Напряжение на обмотке V Масштаб-5В/дел). Масштаб по времени 10мкс/дел.

Описание программы

Полный текст программы приведен в приложении. Программа с помощью таймеров формирует три синусоидальные ШИМ последовательности, сдвинутые на 120 электрических градусов. Частота модулирующей синусоиды и амплитуда модуляции изменяются с помощью кнопок стенда.

Для настройки микроконтроллера выполняются следующие процедуры:

Включение внешнего тактового генератора на 8 МГц и установка коэффициента умножения частоты равным 10. Это обеспечивает частоту микроконтроллера 80 МГц

```
CLK_init() ;// Инициализация тактирования
```

Инициализация кнопок стенда см таблицу 17.2 Инициализация портов на ввод цифровых данных.

```
buttoninit();// Инициализация кнопок
```

Инициализация светодиодов стенда см. таблицу 17.2. Инициализация портов на вывод цифровых данных.

```
svetodiodinit();// Инициализация светодиодов
```

Таблица 17.2 Подключение кнопок и светодиодов к 1986BE91T

№	Порт микро-контроллера, цепь	Контакт разъема 1986BE91_EvBrd	Функция порта	Цепь макетной платы (рис17.3)	Комментарий
1	PC10	X33:2.30	Вход	SEL	Кнопка Старт/Стоп
2	PC11	X33:2.29	Вход	UP	Кнопка Увеличение скорости вращения электродвигателя
3	PC12	X33:2.23	Вход	DOWN	Кнопка Уменьшение скорости вращения электродвигателя
4	PC13	X33:2.26	Вход	LEFT	
5	PC14	X33:2.28	Вход	RIGHT	
6	PD10	X32:1.29	Выход	VD7	Светодиод
7	PD11	X32:1.30	Выход	VD8	Светодиод
8	PD12	X32:1.31	Выход	VD9	Светодиод
9	PD13	X32:1.32	Выход	VD10	Светодиод
10	PD14	X32:1.34	Выход	VD11	Светодиод

Процедура инициализации таймера имеет сигнатуру

```
timerPWMinit();// Инициализация таймера для генерации ШИМ
```

Рассмотрим данную процедуру по порядку:

1 – инициализируются порты (PA1-PA5,PA8) в качестве выводов ШИМ таймера 1;

2 – с помощью структуры `sTIM_CntInit` инициализируются таймер в режиме двунаправленного счета

3 – с помощью структуры `sTIM_ChnInit` инициализируются три канала таймера в режиме генерации ШИМ. Обновление регистра CCR происходит при равенстве 0 счётчика таймера, что соответствует по времени центру осциллограммы трёхфазной ШИМ Рис.__. Этим обеспечивается одновременное обновление всех каналов таймера и отсутствие неполных периодов ШИМ.

4 - с помощью структуры `sTIM_ChnOutInit` инициализируется выходные схемы таймеров. При этом имеется возможность включить схему DTG (DEAD TIME GENERATOR) эта схема позволяет ввести задержку между включением верхнего ключа и выключением нижнего ключа. Включение схемы позволяет избежать ситуации, когда и верхний и нижний ключ открыты одновременно или открылся нижний ключ пока не закрылся верхний. Осциллограмма переключения ключей с использованием DTG представлена на рисунке 17.6 , а без использования ключей на рисунке 17.5 . Время задержки определяется временем включения и выключения транзисторов . Это время можно определить ориентировочно по техническим характеристикам транзистора . Параметры Turn On Delay Time и Turn Off Delay Time в таблице 17.3.

Таблица 17.3. Пример таблицы с характеристиками транзисторов

Electrical Characteristics @ T_J = 25°C (unless otherwise specified)

	Parameter	Min.	Typ.	Max.	Units	Conditions
$t_{d(on)}$	Turn-On Delay Time	—	3.9	—	ns	V_{DD} = 15V
t_r	Rise Time	—	4.0	—		I_D = 0.91A
$t_{d(off)}$	Turn-Off Delay Time	—	9.0	—		R_G = 6.2Ω
t_f	Fall Time	—	1.7	—		R_D = 16Ω, See Fig. 10 ③

В коде определяется время задержки между включением и выключением верхнего и нижнего плеч транзисторов моста с помощью основного и вспомогательного делителя частоты DTG.

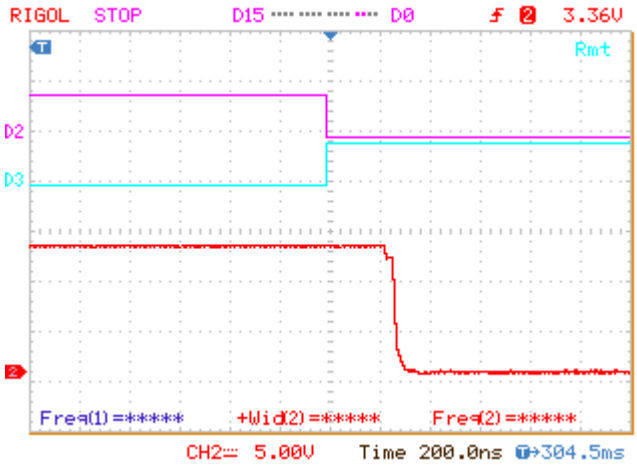


Рисунок 17.6 Осциллограмма переключения управляющих сигналов транзисторов.(D2- Сигнал управления верхним ключом, D3- Сигнал управления нижним ключом, CH2 – линейное напряжение электродвигателя) без использования схемы DTG.

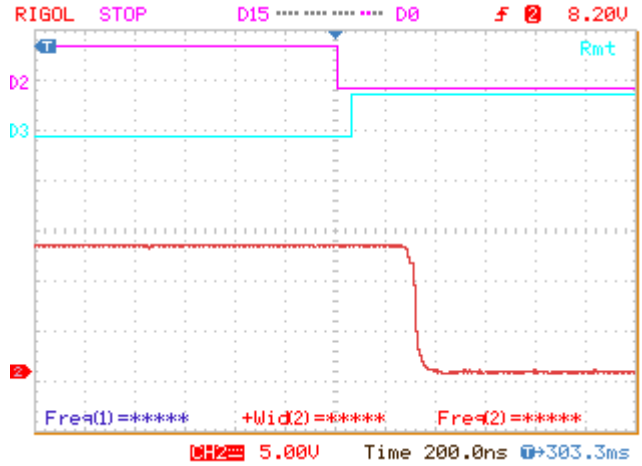


Рисунок 17.7 Осциллограмма переключения управляющих сигналов транзисторов с использованием DEAD TIME длительностью 4 такта при частоте 80 МГц.(D2- Сигнал управления верхним ключом, D3- Сигнал управления нижним ключом, CH2 – линейное напряжение электродвигателя) с использованием схемы DTG.

Таймер 2 отвечает за периодическую смену скважности ШИМ. Данная смена скважности производится в прерывании таймера которое вызывается с фиксированной частотой 2400 Гц. Сигнатура Процедуры выглядит так:

```
Timer2Init();// Инициализация таймера для смены скважности ШИМ.
```

В процедуре происходит последовательно инициализация таймера и инициализация прерывания таймера по пересечению нуля. В процедуре обработки прерывания:

```
void Timer2_IRQHandler()
```

Происходит смена скважности ШИМ. Для этого переменные Theta1, Theta2, Theta3 представляющие собой фазы, сдвинутые на 120 электрических градусов, получают приращение Delta. При этом, 360 электрических градусов фазы напряжений обмоток соответствует 480 условных единиц фазы. Таким образом, сдвиг на 120 электрических градусов обеспечивается приращением равным 160. Далее с помощью процедуры

```
duty_cycle(Theta, Vm, MAX_PWM).
```

Данная процедура рассчитывает число в соответствии с таблицей синусов величину для записи в регистр сравнения канала таймера. Это число обеспечивает нужный ШИМ. Параметрами процедуры являются: фаза (Theta) в диапазона о 0 до 479, амплитула (Vm) в диапазоне от 0 до max_pwm/2, максимально возможный размах ШИМ MAX_PWM, определяемый модулем счета счетчика. Для ускорения расчетов применяется таблица значений синуса `tab_sin[121]`. Расчетная формула имеет вид

```
duty_cycle=Vm*Sin(Theta)+MAXPWM/2.
```

В основном цикле программы `while(1)` происходит проверка нажатия кнопок стенда. При нажатии кнопки SEL мотор останавливается, при нажатии кнопки UP увеличивается частота вращения ротора, при нажатии кнопки DOWN уменьшается частота вращения ротора. Параметры MIN_DUTY_CYCLE, MAX_DUTY_CYCLE, MAX_DUTY_CYCLE_ELECTRIC_FREC соответствуют Vпор, Vном, Fном на рисунке 17.1

На базе этих параметров и требуемой электрической частоты вращения Frec рассчитывается приращение фазы на 1 шаг Delta и амплитуда синусоиды Amp_duty_cycle.

Ход работы

1. Собрать лабораторную установку. Подключить макетную плату с силовым инвертором к Отладочной плате MDR1986VE91T Rev 4. в соответствии с таблицей 17.1

2. Включить отладочную плату.

3. Создать новый проект, подключить в менеджере *Manage run-time environment* к проекту библиотеки Startup_MDR1986BE9x, PORT, TIMER, RST_CLK, необходимые для работы.

4. Добавьте в папку проекта файлы *joystick.h* и *joystick.c* из «DEMO» примера, представленного на компакт диске Миландр для этой платы. Данный файл предназначен для обработки нажатия кнопок на стенде.

5. Добавьте *joystick.c* в проект. Для этого на вкладке *Project Window* щелкните правой кнопкой мыши на *SourceGroup1* , в появившемся меню выберите *Add Existing Files to Group.....* , далее в окне выбора выберите файл *joystick.c* см. рис.17.8

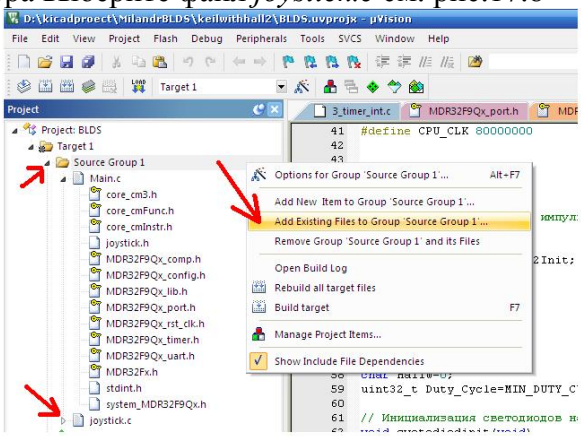


Рисунок 17.8 Добавление файла в проект.

6. Скопировать в main.c код представленный в приложении. Затем скомпилировать проект.

7. Включить питание макетной платы и произвести программирование МК.

8. Проверить прохождение логических сигналов от микропроцесора до баз транзисторов путем подключения логического анализатора к контактам макетной платы, D0-HU (***X***33:1.35)**,** D1-LU (***X***33:1.36)**,** первого канала осциллографа к контакту CH1-HOU_VT макетной платы, второго канала осциллографа к контакту CH2-LOU_VT макетной платы. После этого на отладочной плате нажать кнопку SEL. Наблюдать на экране осциллографа ШИМ сигнал. Добиться на экране осциллографа картинки представленной на рисунке 17.9. На данной картинке видна задержка сигнала в драйвере электродвигателя IR2101(от момента подачи логического импульса до начала импульса на базе транзистора проходит немногим более 200 нс.).

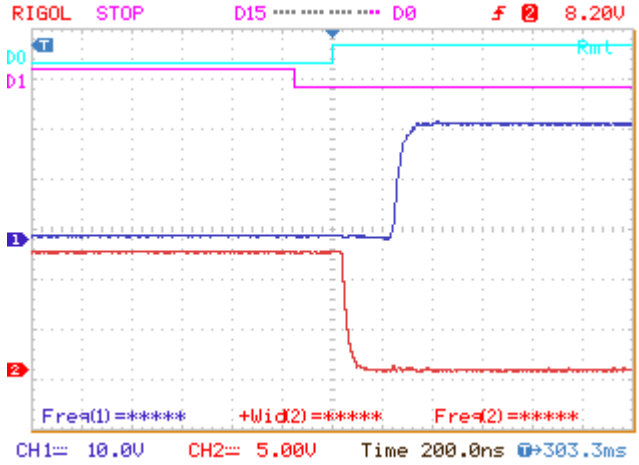


Рисунок 17.9. Логические сигналы управления ключами и сигналы на базах транзистора после драйвера.

9. Проверить наличие сигналов трехфазной ШИМ. Для этого подключить логический анализатор осциллографа к контактам макетной платы (D0- логический сигнал управления транзистором VT1 цепь HU; D1-VT2 цепь LU; D2-VT3 цепь H цепь HV; D3-VT4 цепь LV; D4-VT5 цепь HW; D5-VT6 цепь LW;). С помощью кнопок UP и DOWN отрегулируйте амплитуду ШИМ. Внешний вид осциллограммы сигналов трехфазной ШИМ представлен на рисунке 17.4 Для включение выключения генерации трехфазной ШИМ последовательности используйте кнопку SEL.

10. Убедиться в наличии трехфазной центрированной ШИМ последовательности. Для этого подключите каналы логического анализатора D0, D2, D4 – Управляющие напряжения ключей обмоток U,V,W соответственно. Два канала осциллографа подключить по следующей схеме CH1- Напряжение в цепи SHUNT рис.17.5 (Масштаб-1В/дел); CH2- Напряжение на обмотке V Масштаб-5В/дел). Внешний вид осциллограммы центрированной ШИМ представлен на рис. 17.5.

11. Провести исследование работы схемы при наличии задержки включения транзисторов (Наличие сигнала DEAD TIME). Для этого в процедуре `timerPWMinit()` раскомментируйте строчки

```
      sTIM_ChnOutInit.TIMER_CH_DirOut_Source= TIMER_CH_OutSrc_DTG,
sTIM_ChnOutInit.TIMER_CH_NegOut_Source      = TIMER_CH_OutSrc_DTG.
```

Далее необходимо откомпилировать и провести программирование микроконтроллера. Это позволяет подключить схему DTG к выходному сигналу. В программе длительность сигнала DTG определяется основным делителем частоты =1 и предварительным делителем частоты 3+1=4. То есть между выключением нижнего и включением верхнего плеч будет задержка 4 такта тактовой частоты таймера. Зафиксируйте осциллограмму ШИМ сигнала как на рис 17.7

12. Провести исследование работы схемы без использования DEAD-TIME. Для этого Для этого в процедуре `timerPWMinit()` раскомментируйте строчки

```
      sTIM_ChnOutInit.TIMER_CH_NegOut_Source       = TIMER_CH_OutSrc_REF,
      sTIM_ChnOutInit.TIMER_CH_DirOut_Source       = TIMER_CH_OutSrc_REF.
```

Далее необходимо откомпилировать и провести программирование микроконтроллера.

Это включает режим генерации ШИМ сигнала без DTG. То есть включение верхнего ключа моста и выключение нижнего происходит одновременно, как показано на рис17.6. Зафиксируйте осциллограмму этого ШИМ сигнала

13. Выключить ШИМ сигнал нажатием кнопки SEL. Подключите к клеммам макетной платы электродвигатель. Включите вращение нажатием кнопки SEL. С помощью кнопок UP и DOWN можно регулировать скорость вращения.

Задания

1. Изменить направление вращения.
2. Изменить частоту ШИМ на заданную преподавателем.
3. Реализовать датчик тока электродвигателя путем обработки сигнала с помощью встроенного АЦП микроконтроллера (указание: необходимо применить встроеную RC цепочку).

Приложение

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_timer.h>
#include "joystick.h"

// Переменные для инициализации таймера и портов
TIMER_CntInitTypeDef sTIM_CntInit;
TIMER_ChnInitTypeDef sTIM_ChnInit;
TIMER_ChnOutInitTypeDef sTIM_ChnOutInit;
PORT_InitTypeDef PORT_InitStructure;

// Максимальная частота вращения привода CD ROM 24x 6000 об/мин или 100 Гц примем 120Гц (чтобы число
делилось на 2, 3, 6)
// Число пар полюсов привода CD ROM 6
// Максимальная частота вращения, Гц
#define MAX_FREC 120
// Число пар полюсов
#define POLUS    6
// Примем, Частоту модификации ШИМ = 2400.
#define FREC_MOD_PWM 2400
// Максимальная электрическая частота вращения, Гц
#define MAX_ELECTRIC_FREC MAX_FREC*POLUS
// Шаг изменения частоты ()
#define STEP_ELECTRIC_FREC 1
// Максимальный период ШИМ ()
// Частота ШИМ может быть определена по формуле
// Частота ШИМ  FPWM = FCNT/(2*MAX_PWM)
// Коэффициент  2 отражает тот факт, что счет ведется в две стороны на одном периоде ШИМ
// Для нашего примера FPWM = 80000000/(2*2048)=19540 Гц;
#define MAX_PWM 0x7FF
// Максимальная скважность ШИМ
#define MAX_DUTY_CYCLE (MAX_PWM/2)
// Электрическая частота вращения на которой достигается максимальная скважность импульсов =
номанальная частота вращения * число пар полюсов
#define MAX_DUTY_CYCLE_ELECTRIC_FREC 350
// Минимальная скважность ШИМ на минимальной частоте вращения
#define MIN_DUTY_CYCLE MAX_PWM/20
// Число точек разбиения синусоиды на 90 градусов
#define MAX_THETA     120
// Число точек разбиения синусоиды на 180 градусов
#define MAX_THETAx2   240
// Число точек разбиения синусоиды на 270 градусов
#define MAX_THETAx3   360
// Число точек разбиения синусоиды на 360 градусов
#define MAX_THETAx4   480

// Таблица значений синуса от 0 до 90 градусов в диапазоне 0-127 (120 точек)
const unsigned char tab_sin[121]=
{
0,2,3,5,7,8,10,12,13,15,
```

```
17,18,20,22,23,25,26,28,30,31,
33,34,36,38,39,41,42,44,46,47,
49,50,52,53,55,56,58,59,61,62,
63,65,66,68,69,71,72,73,75,76,
77,79,80,81,82,84,85,86,87,89,
90,91,92,93,94,95,97,98,99,100,
101,102,103,104,105,106,107,107,108,109,
110,111,112,112,113,114,115,115,116,117,
117,118,119,119,120,120,121,121,122,122,
123,123,123,124,124,125,125,125,125,126,
126,126,126,126,127,127,127,127,127,127,
127
};

//Процедура вычисления ШИМ
uint16_t duty_cycle(uint16_t theta, uint16_t Vm, uint16_t max_pwm)
{
  signed char   sinus ;
  float       Vmxsin ;

   if (theta <= MAX_THETA)                    // x in [0;pi/2]
    sinus = tab_sin[theta];                   // sin(x)
   else{
      if (theta <= MAX_THETAx2)               // x in [pi/2;pi]
       sinus = tab_sin[MAX_THETAx2-theta];    // sin(pi-x)
      else{
        if (theta<=MAX_THETAx3)               // x in [pi;3*pi/2]
         sinus = - tab_sin[theta-MAX_THETAx2]; // sin(x-pi)
        else                                  // x in [3*pi/2;2*pi]
         sinus = - tab_sin[MAX_THETAx4-theta]; // sin(2*pi-x)
      }
    }
   Vmxsin =(float) sinus;
   Vmxsin = (Vmxsin * Vm)/128 ; // максимальное значение синуса 127
   return ((uint16_t) (max_pwm/2 + Vmxsin));
}

// Значения длительности импульса по каналам 1,2,3
uint16_t CCR1_Val = 0x1FF;
uint16_t CCR2_Val = 0x3FF;
uint16_t CCR3_Val = 0x4FF;
uint32_t Button_Val;
uint16_t Frec = 0;
uint16_t Amp_duty_cycle = 0;
uint16_t Duty_Cycle_U = 0;
uint16_t Duty_Cycle_V = 0;
uint16_t Duty_Cycle_W = 0;
float Theta1 = 0;
float Theta2 = 160;
float Theta3 = 320;
float Delta =0;

// Инициализация светодиодов на плате для индикации режима работы
void svetodiodinit(void)
{
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTD, ENABLE);
  PORT_InitStructure.PORT_Pin = PORT_Pin_10|PORT_Pin_11|PORT_Pin_12|PORT_Pin_13|PORT_Pin_14;
  PORT_InitStructure.PORT_OE    = PORT_OE_OUT;
  PORT_InitStructure.PORT_FUNC  = PORT_FUNC_PORT;
  PORT_InitStructure.PORT_MODE  = PORT_MODE_DIGITAL;
  PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
  PORT_Init(MDR_PORTD, &PORT_InitStructure);
}
// Инициализация кнопок на плате
void buttoninit(void)
{
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);
  PORT_InitStructure.PORT_Pin   = PORT_Pin_10|PORT_Pin_11|PORT_Pin_12|PORT_Pin_13|PORT_Pin_14;
  PORT_InitStructure.PORT_OE    = PORT_OE_IN;
  PORT_InitStructure.PORT_FUNC  = PORT_FUNC_PORT;
  PORT_InitStructure.PORT_MODE  = PORT_MODE_DIGITAL;
  PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
  PORT_Init(MDR_PORTC, &PORT_InitStructure);
}
TIMER_CntInitTypeDef TIM2Init;

void Timer2Init(void){
       RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER2, ENABLE);
       TIMER_BRGInit(MDR_TIMER2, TIMER_HCLKdiv1);
```

```
        TIMER_CntStructInit(&TIM2Init);      //Load defaults
        // Предделитель, значение 3 соответствует делению частоты процессора на 4
        TIM2Init.TIMER_Prescaler        = 3;
        // Обеспечиваем частоту смены ШИМ в прерывании 2400Гц = ((80000000/4)/8334)
        TIM2Init.TIMER_Period           = 8333;
       TIMER_CntInit(MDR_TIMER2, &TIM2Init);
        //Инициализация прерываний
        NVIC_EnableIRQ(Timer2_IRQn);
        NVIC_SetPriority(Timer2_IRQn, 0);
        TIMER_ITConfig(MDR_TIMER2, TIMER_STATUS_CNT_ZERO, ENABLE);
        TIMER_Cmd(MDR_TIMER2, ENABLE);
};

void Timer2_IRQHandler() {
  if(TIMER_GetITStatus(MDR_TIMER2, TIMER_STATUS_CNT_ZERO)){
    TIMER_ClearITPendingBit(MDR_TIMER2, TIMER_STATUS_CNT_ZERO);
    PORT_WriteBit(MDR_PORTD, PORT_Pin_13, Bit_SET );
    // Прибавляем значение фазы ШИМ на шаг.
    Theta1=Theta1+Delta;
    Theta2=Theta1+160;
    Theta3=Theta2+160;
    if (Theta1>=480){
       Theta1 -= 480;PORT_WriteBit(MDR_PORTD, PORT_Pin_10, Bit_SET);
    }
    if (Theta2>=480) Theta2 -= 480;
    if (Theta3>=480) Theta3 -= 480;
              // Расчет и установка требуемых новых параметров ШИМ.
    TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL1,
                        duty_cycle((uint16_t)Theta1, Amp_duty_cycle, MAX_PWM));
    TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL2,
                        duty_cycle((uint16_t)Theta2, Amp_duty_cycle, MAX_PWM));
    TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL3,
                        duty_cycle((uint16_t)Theta3, Amp_duty_cycle, MAX_PWM));
    PORT_WriteBit(MDR_PORTD, PORT_Pin_13|PORT_Pin_10,  Bit_RESET);
  }
};

void CLK_init(void) { //Установка частоты тактирования 80МГц
  /* Включение HSE осциллятора (внешнего кварцевого резонатора)*/
  RST_CLK_HSEconfig(RST_CLK_HSE_ON);
  if (RST_CLK_HSEstatus() == SUCCESS){  /* Если HSE осциллятор включился и прошел текст*/
    // Выбор HSE осциллятора в качестве источника тактовых импульсов для CPU_PLL
    // и установка умножителя тактовой частоты CPU_PLL равного 7
    // Частота внешнего кварца равна 8 МГц Максимальная частота процессора 80 МГц ,
    // RST_CLK_CPU_PLLconfig ( Источник тактирования PLL, Коэффициент умножения 9+1);
                      // Коэффициент умножения=0 соответствует умножение на 1.
    RST_CLK_CPU_PLLconfig ( RST_CLK_CPU_PLLsrcHSEdiv1, 9 );
    /* Включение схемы PLL*/
    RST_CLK_CPU_PLLcmd(ENABLE);
    if (RST_CLK_CPU_PLLstatus() == SUCCESS){//Если включение CPU_PLL прошло успешно
      /* Установка CPU_C3_prescaler = 1 */
      RST_CLK_CPUclkPrescaler(RST_CLK_CPUclkDIV1);
      /* Установка CPU_C2_SEL от CPU_PLL выхода вместо CPU_C1 такта*/
      RST_CLK_CPU_PLLuse(ENABLE);
      /*ВыборCPU_C3 такта на мультиплексоре тактовых импульсов микропроцессора (CPU clock MUX) */
      RST_CLK_CPUclkSelection(RST_CLK_CPUclkCPU_C3);
    }
    else        /* блок CPU_PLL не включился*/
      {while(1);}
  }
  else          /* кварцевый резонатор HSE не включился */
    {while(1);};
}

void timerPWMinit(void)
{
  //Включение тактирования порта A
  RST_CLK_PCLKcmd((RST_CLK_PCLK_PORTA), ENABLE);
  /* Сброс установок порта A*/
  PORT_DeInit(MDR_PORTA);
  /* Конфигурация выводов таймера TIMER1: CH1, CH1N, CH2, CH2N, CH3, CH3N  */
  /* Настройка порта A разряды 1, 2, 3, 4, 5 ,8 */
  // На демонстрационной плате в разъеме X33:1 контакты 35, 36, 31, 32, 29, 28 соответственно.
  PORT_InitStructure.PORT_Pin =(PORT_Pin_1 | PORT_Pin_2 | PORT_Pin_3 |
                                PORT_Pin_4 | PORT_Pin_5|   PORT_Pin_8);
  PORT_InitStructure.PORT_OE    = PORT_OE_OUT;
  PORT_InitStructure.PORT_FUNC  = PORT_FUNC_ALTER;
  PORT_InitStructure.PORT_MODE  = PORT_MODE_DIGITAL;
  PORT_InitStructure.PORT_SPEED = PORT_SPEED_FAST;
```

```
  PORT_Init(MDR_PORTA, &PORT_InitStructure);

  // Включение тактирования таймера 1
  RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER1,ENABLE);
  /* Сброс установок таймера 1 */
  TIMER_DeInit(MDR_TIMER1);
  // Заполнение структуры значениями по умолчанию
  TIMER_CntStructInit(&sTIM_CntInit);
  // Предделитель частоты таймера равен 1 то есть частота счета таймера равна частоте
  // микроконтоллера.
  sTIM_CntInit.TIMER_Prescaler               = 0x0;
  // Модуль счета = 0x7FF
  // Период импульсов ШИМ = 2 * 0x7FF / частота счета таймера.
  sTIM_CntInit.TIMER_Period                  = MAX_PWM;
  // Счет в две стороны
  sTIM_CntInit.TIMER_CounterMode =   TIMER_CntMode_ClkChangeDir  ;
  // Инициализация таймера 1
  TIMER_CntInit (MDR_TIMER1,&sTIM_CntInit);
  /* Инициализация каналов таймера 1:  CH1,CH1N,CH2,CH2N,CH3,CH3N */
  TIMER_ChnStructInit(&sTIM_ChnInit);
  // Режим работы канала - генерация ШИМ
  sTIM_ChnInit.TIMER_CH_Mode                 = TIMER_CH_MODE_PWM;
  // Формат выработки сигнала REF номер 6:
  // при счете вверх - REF = 1 если  CNT<CCR, иначе REF = 0;
  // значение регистра CCR определяет длительность импульса ШИМ
  sTIM_ChnInit.TIMER_CH_REF_Format           = TIMER_CH_REF_Format6;
  // обновление регистра CCR в момент равенства нулю счетчика
  sTIM_ChnInit.TIMER_CH_CCR_UpdateMode =  TIMER_CH_CCR_Update_On_CNT_eq_0;

  // Инициализируем канал 1
  sTIM_ChnInit.TIMER_CH_Number               = TIMER_CHANNEL1;
  TIMER_ChnInit(MDR_TIMER1, &sTIM_ChnInit);

  // Инициализируем канал 2
  sTIM_ChnInit.TIMER_CH_Number               = TIMER_CHANNEL2;
  TIMER_ChnInit(MDR_TIMER1, &sTIM_ChnInit);

  // Инициализируем канал 3
  sTIM_ChnInit.TIMER_CH_Number               = TIMER_CHANNEL3;
  TIMER_ChnInit(MDR_TIMER1, &sTIM_ChnInit);

  /* Инициализация прямого и инверсного выходов (стр289 описания!!!)
   для каждого из каналов таймера 1: CH1,CH1N,CH2,CH2N,CH3,CH3N */
  // Заполнение элементов структуры значениями по умолчанию
  TIMER_ChnOutStructInit(&sTIM_ChnOutInit);
  // Выбор источника сигнала для прямого выхода  CHxN - сигнал REF
  sTIM_ChnOutInit.TIMER_CH_DirOut_Source     = TIMER_CH_OutSrc_DTG;
  // Если нет необходиости в DTG то раскомментируйте следующую строчку
  // sTIM_ChnOutInit.TIMER_CH_NegOut_Source = TIMER_CH_OutSrc_REF
  // Настройна прямого выхода микроконтроллера CHxN на вывод данных
  sTIM_ChnOutInit.TIMER_CH_DirOut_Mode              = TIMER_CH_OutMode_Output;
  // Выбор источника сигнала для инверсного выхода  CHxN - сигнал REF
  sTIM_ChnOutInit.TIMER_CH_NegOut_Source     = TIMER_CH_OutSrc_DTG  ;
  // Если нет необходиости в DTG то раскомментируйте следующую строчку
  // sTIM_ChnOutInit.TIMER_CH_DirOut_Source = TIMER_CH_OutSrc_REF
  // Настройна инверсного выхода микроконтроллера CHxN на вывод данных
  sTIM_ChnOutInit.TIMER_CH_NegOut_Mode = TIMER_CH_OutMode_Output;
  // Основной делитель для схемы DTG
  sTIM_ChnOutInit.TIMER_CH_DTG_MainPrescaler = 1;
  // Вспомогательный делитель для схемы DTG
  sTIM_ChnOutInit.TIMER_CH_DTG_AuxPrescaler = 3;
  // Источник тактирования схемы DTG
  sTIM_ChnOutInit.TIMER_CH_DTG_ClockSource = TIMER_CH_DTG_ClkSrc_TIMER_CLK;
  // Настраиваем выходы канала 1
  sTIM_ChnOutInit.TIMER_CH_Number                    = TIMER_CHANNEL1;
  TIMER_ChnOutInit(MDR_TIMER1, &sTIM_ChnOutInit);
  // Настраиваем выходы канала 2
  sTIM_ChnOutInit.TIMER_CH_Number                    = TIMER_CHANNEL2;
  TIMER_ChnOutInit(MDR_TIMER1, &sTIM_ChnOutInit);
  // Настраиваем выходы канала 3
  sTIM_ChnOutInit.TIMER_CH_Number                    = TIMER_CHANNEL3;
  TIMER_ChnOutInit(MDR_TIMER1, &sTIM_ChnOutInit);
  /* Включаем делитель тактовой частоты таймера 1*/
  TIMER_BRGInit(MDR_TIMER1,TIMER_HCLKdiv1);
  /* После всех настроек разрешаем работу таймера 1 */
  TIMER_Cmd(MDR_TIMER1,ENABLE);
};

int main(){
```

```
  CLK_init() ;// Инициализация тактирования
  buttoninit();// Инициализация кнопок
  svetodiodinit();// Инициализация светодиодов
  timerPWMinit();// Инициализация таймера для генерации ШИМ
  Timer2Init();// Инициализация таймера для смены скважности ШИМ

while(1)
{
  // Считываем данные с кнопок.
  PORT_WriteBit(MDR_PORTD, PORT_Pin_12, Bit_SET);
  Button_Val=PORT_ReadInputData(MDR_PORTC);
  // Проверяем какая кнопка нажата
  if KEY_PRESSED(SEL){
    PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_SET);
    //Ждем пока кнопку не отпустим.
    WAIT_UNTIL_KEY_RELEASED(SEL);
    Delta=0;
    Amp_duty_cycle    =0;
    Frec=0;
    PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_RESET);
  }
if KEY_PRESSED(UP){
  PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_SET);
  WAIT_UNTIL_KEY_RELEASED(UP);
  Frec++;
  if (Frec>MAX_ELECTRIC_FREC) Frec=MAX_ELECTRIC_FREC;
  // Определяем приращение фазы на каждом периоде изменения ШИМ
  Delta= ( (float) MAX_THETAx4 /(FREC_MOD_PWM/Frec));
  // Определяем амплитуду изменения ШИМ сигнала. про принципу постоянства V/f
  Amp_duty_cycle      =(uint16_t)((float)(( (float) (MAX_DUTY_CYCLE-MIN_DUTY_CYCLE)/
                      MAX_DUTY_CYCLE_ELECTRIC_FREC)*Frec)+MIN_DUTY_CYCLE);
  if (Amp_duty_cycle>MAX_DUTY_CYCLE){Amp_duty_cycle=MAX_DUTY_CYCLE;};
  PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_RESET);
}
if KEY_PRESSED(DOWN){
  PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_SET);
  WAIT_UNTIL_KEY_RELEASED(DOWN)
  if (Frec>1) {
    Frec--;
    // Определяем приращение фазы на каждом периоде изменения ШИМ
    Delta= ( (float) MAX_THETAx4 /(FREC_MOD_PWM/Frec));
    // Определяем амплитуду изменения ШИМ сигнала. про принципу постоянства V/f
    Amp_duty_cycle    = (uint16_t)((float)(((float) (MAX_DUTY_CYCLE-MIN_DUTY_CYCLE)/
                      MAX_DUTY_CYCLE_ELECTRIC_FREC)*Frec)+MIN_DUTY_CYCLE);
                       // Ограничитель амплитуды ШИМ
    if (Amp_duty_cycle>MAX_DUTY_CYCLE){Amp_duty_cycle=MAX_DUTY_CYCLE;};
  }
  else{
    Delta=0;
    Frec=0;
    Amp_duty_cycle    = 0;
  };
  PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_RESET);
};
duty_cycle((uint16_t)Theta1, Amp_duty_cycle, MAX_PWM);
};

}
```

Лабораторная работа №18 Управление трехфазным бесколлекторным электродвигателем постоянного тока без использования датчиков положения ротора

Цель работы: Научится использовать микроконтроллер 1986BE1T в качестве управляющего микроконтроллера электродвигателя постоянного тока без использования датчиков. В качестве алгоритма управления применяется алгоритм с использованием ЭДС свободной фазы.

Приборы и материалы:

1. Отладочная плата *MDR1986VE91T Rev 4*.

2. Программатор *J-Link ARM*.

3. Блок питания 5В, 1.4А.

4. ПК с установленной средой программирования *Keil uVision*.

5. Макетная плата с силовым инвертором и схемой измерения сигналов обратной ЭДС электродвигателя.

6. Бесколлекторный электродвигатель постоянного тока от шпинделя CD ROM без нагрузки. Электродвигатель необходим для визуализации вращения.

7. Осциллограф с логическим анализатором.

Сведения для выполнения

В коллекторных двигателях постоянного тока коммутация обмоток в нужный момент времени осуществляется с помощью коллекторного узла(якоря). В бесколлекторных двигателях вместо коллекторного узла применяется электронная схема - инвертор. Управление бесколлекторными двигателями без датчиков используется в тех случаях, когда пусковой момент существенно не изменяется и когда отсутствует необходимость в точном позиционировании ротора, например, в вентиляторах.

На каждом шаге коммутации, обмотка одной фазы подключается к положительному напряжению питания, другая - к отрицательному, а третья – неподключенная используется для измерения обратной ЭДС. Обратная ЭДС неподключенной фазы в результате пересекает ноль при пересечении среднего значения положительного и отрицательного напряжений. Пересечение ноля возникает всегда в центре между двумя шагами. Таким образом, пересечение нуля должно вызывать коммутацию обмоток. Это поясняет рис.18.1

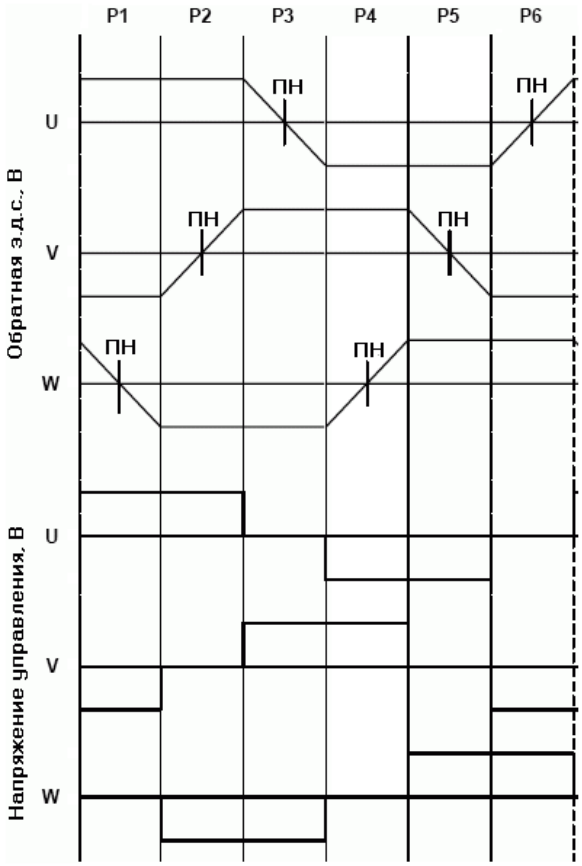


Рисунок 18.1 Детектирование пересечения нуля (средней точки).

Для определения сигналов обратной ЭДС используется метод виртуальной средней точки, с которой сравнивается напряжение на свободной обмотке. Для сравнения используются аналоговые компараторы микроконтроллера. Компаратор вызывает

прерывание в момент перехода напряжения через виртуальную среднюю точку. Для того чтобы прерывания начали срабатывать устойчиво необходим предварительный разгон ротора до некоторой скорости с помощью программного управления. После предварительного разгона включается режим работы по обратной ЭДС. Также на рис 18.1 представлены графики подаваемых напряжений на обмотки UVW. Как видим на рисунке всего есть 6 фаз коммутации P1-P6. Этим Фазам соответствуют 6 состояний ключей инвертора – базовых векторов. Помимо этих 6 состояний к базовым векторам относят состояния: все транзисторы закрыты – P0, все верхние ключи открыты - P7, все нижние ключи открыты P8. Полностью базовые вектора представлены в таблице.

Таблица 18.1 Таблица базовых векторов для управления бесколлекторным двигателем.

Базовый вектор	Верхний ключ фазы U	Нижний ключ фазы U	Верхний ключ фазы V	Нижний ключ фазы V	Верхний ключ фазы W	Нижний ключ фазы W
P0	закрыт	закрыт	закрыт	закрыт	закрыт	закрыт
P1	ШИМ	закрыт	закрыт	открыт	закрыт	закрыт
P2	ШИМ	закрыт	закрыт	закрыт	закрыт	открыт
P3	закрыт	закрыт	ШИМ	закрыт	закрыт	открыт
P4	закрыт	открыт	ШИМ	закрыт	закрыт	закрыт
P5	закрыт	открыт	закрыт	закрыт	ШИМ	закрыт
P6	закрыт	закрыт	закрыт	открыт	ШИМ	закрыт
P7	открыт	закрыт	открыт	закрыт	открыт	закрыт
P8	закрыт	открыт	закрыт	открыт	закрыт	открыт

Более подробно о режиме управления без использования датчиков (по обратной ЭДС) можно почитать [11,14].

Описание лабораторной установки

Силовая часть схемы аналогична рис.17.3. Для обеспечения измерения обратной ЭДС неподключенной обмотки электродвигателя применяется схема, представленная на рисунке 18.2. Напряжения обмоток подаются на входы UVW. Далее после фильтрации с помощью RC цепочек (например R14-C6) формируется 4 сигнала: REFERENCE – виртуальная средняя точка; Comp_U, Comp_V, Comp_W – выходы соответствующих подаваемые на компаратор. Фильтрация напряжения необходима для того чтобы в сигнале подаваемом на компараторы не было влияния ШИМ сигнала вырабатываемого инвертором.

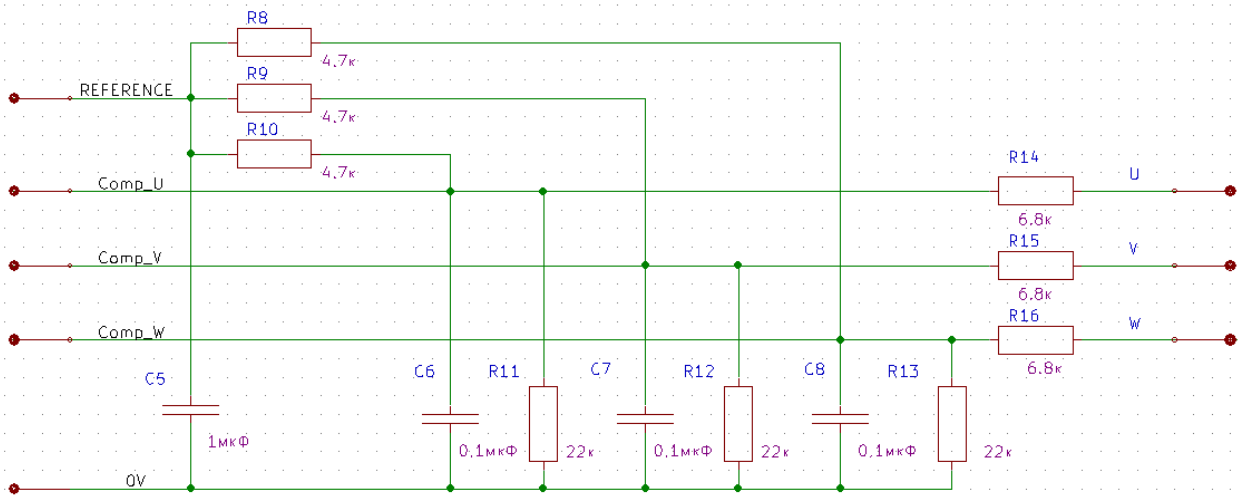


Рисунок 18.2. Электрическая принципиальная схема измерения сигналов обратной ЭДС электродвигателя.

Подключение данной схемы к микроконтроллеру отладочной платы 1986BE91T осуществляется согласно следующей таблице18.2. Сигналы COMP REF+ ,COMP REF- подключаются к выходы REFERENCE. Этим обеспечивается подача сигнала REFERENCE

на микроконтроллер и независимость этого напряжения от настроек компаратора.

Таблица.18.2 Подключение схемы измерения сигналов обратной ЭДС к отладочной плате 1986BE91T.

№	Порт микро-контроллера, цепь	Контакт разъема 1986BE91_EvBrd	Функция порта	Цепь макетной платы (рис18.2)	Коммен-тарий
1	PE2	X32:1.23	COMP IN1	Comp_U	Фаза U
2	PE3	X32:1.24	COMP IN2	Comp_V	Фаза V
3	PE8	COMP_INP	COMP IN3	Comp_W	Фаза W
4	PE4	X32:2.24	COMP REF+	REFERENCE	Виртуальная средняя точка
5	PE5	X32:2.23	COMP REF-	REFERENCE	

Работа схемы происходит в два этапа на первом этапе происходит предварительный разгон ротора при этом можно по осциллографу контролировать уровень обратной ЭДС и четкость срабатывания компараторов. Как только уровень обратной ЭДС превысит уровень помех в 3-5 раз схему переводят в режим разгона по обратной ЭДС. Для перехода в режим разгона необходимо один раз нажать кнопку SEL. Для перехода в режим работы по обратной ЭДС и отключения его нужно одновременно нажать две любые кнопки.

Описание программы

Полный текст программы приведен в приложении. В программе используются:

Таймер 1, для генерации ШИМ

Таймер 2, для изменения базовых векторов в режиме разгона

Компаратор, для обеспечения работы по наведенной ЭДС.

Кнопки отладочной платы осуществляют управление стендом.

Для настройки микроконтроллера выполняются следующие процедуры:

Включение внешнего тактового генератора на 8 МГц и установка коэффициента умножения частоты равным 10. Это обеспечивает частоту микроконтроллера 80 МГц

```
CLK_init() ;// Инициализация тактирования
```

Инициализация кнопок стенда см таблицу 17.2. Инициализация портов на ввод цифровых данных.

```
buttoninit();// Инициализация кнопок
```

Инициализация светодиодов стенда см. таблицу 17.2. Инициализация портов на вывод цифровых данных.

```
svetodiodinit();// Инициализация светодиодов
```

Инициализация компаратора

```
COMPInit();
```

При инициализации компаратора последовательно выполняется следующие команды:

1 - включается тактирование компаратора и порта E,

2 - порт E настраивается на выполнение функций компаратора,

3 - компаратор инициализируется структурой в которой на плюсовой вход подается напряжение REFERENCE, а на минусовой вход подается напряжение CompU.

4 - настраивается источник опорного напряжения для работы с внешними входами COMP REF+, COMP REF-. В принципе делитель опорного напряжения можно устанавливать с любым коэффициентом деления. Поскольку на входы COMP REF+, COMP REF- подается одинаковое напряжение REFERENCE от схемы измерения сигналов обратной ЭДС.

Инициализация таймера для генерации ШИМ

5 – Инициализируется прерывание по срабатыванию компаратора с сигнатурой

`void COMPARATOR_IRQHandler();` В этом прерывании осуществляется переключение обмоток в соответствии с таблицей18.1.

Инициализация таймера для генерации ШИМ

```
timerPWMinit();// Инициализация таймера для генерации ШИМ
```

Рассмотрим данную процедуру по порядку:

1 – инициализируются порты (PA1-PA5,PA8) в качестве выводов ШИМ таймера 1;

2 – с помощью структуры `sTIM_CntInit` инициализируются таймер в режиме двунаправленного счета

3 – с помощью структуры `sTIM_ChnInit` инициализируются три канала таймера в режиме генерации ШИМ. Обновление регистра CCR происходит при равенстве 0 счетчика таймера, что соответствует по времени центру осциллограммы трехфазной ШИМ Рис.__. Этим обеспечивается одновременное обновление всех каналов таймера и отсутствие неполных периодов ШИМ.

4 - с помощью структуры `sTIM_ChnOutInit` инициализируется выходные схемы таймеров. В начальный момент времени все транзисторы закрываются (TIMER_CH_OutSrc_Only_0).

Инициализация таймера для режима разгона.

```
Timer2Init();
```

В результате инициализации таймер работает на частоте 10000Гц. Основание счета устанавливается переменной TIMER2_MODUL по формуле:

TIMER2_MODUL = ((CPU_CLK /(TIMER2_PRESCALER+1))/(START_FREC*POLUS*6)),

где CPU_CLK – тактовая частота процессора; TIMER2_PRESCALER – предварительный делитель частоты таймера; START_ELECTRIC_FREC – начальная электрическая частота вращения.

В основном цикле программы `while(1)` происходит проверка нажатия кнопок стенда. В зависимости от того какая кнопка нажата выполняются различные действия. Рассмотрим назначение кнопок в программе:

SEL – Включение/выключение режима разгона электродвигателя (Переменная Run)

UP – увеличение скорости вращения электродвигателя. Процедура `Timer2change(Period);` изменяет период смены базовых векторов ШИМ.

DOWN – уменьшение скорости вращения электродвигателя.

LEFT – Увеличение скважности ШИМ. Увеличивает переменную `Duty_Cycle`. Это приводит к увеличению скорости электродвигателя при работе по датчикам. Изменение скважности осуществляется функциями типа:

```
TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL1, Duty_Cycle).
```

RIGHT – Уменьшение скважности ШИМ. Действует по принципу LEFT.

MULTIPLE – одновременное нажатие двух любых кнопок приводит к включению выключению режима работы по обратной ЭДС (Переменная RunRun)

Процедура `void Set_Base_Vector(char base_vector)` устанавливает базовый вектор в соответствии с таблицей коммутации (Таблица18.1).

Порядок работы

1. Собрать лабораторную установку. Подключить макетную плату с силовым инвертором к Отладочной плате MDR1986VE91T Rev 4. в соответствии с таблицей 18.2

2. Включить отладочную плату.

3. Создать новый проект, подключить в менеджере *Manage run-time environment* к проекту библиотеки Startup_MDR1986BE9x, PORT, TIMER, RST_CLK, COMP необходимые для работы.

4. Добавьте в папку проекта файлы *joystick.h* и *joystick.c* из «DEMO» примера, представленного на компакт диске Миландр для этой платы. Данный файл предназначен для обработки нажатия кнопок на стенде.

5. Добавьте *joystick.c* в проект. Для этого на вкладке *Project Window* щелкните правой кнопкой мыши на *SourceGroup1* , в появившемся меню выберите *Add Existing Files to*

Group..... , далее в окне выбора выберите файл *joystick.c* см. рис.17.8

6. Скопировать в main.c код представленный в приложении. Затем скомпилировать проект.

7. Включить питание отладочной платы и произвести программирование МК.

8. Подключить осциллограф к каналу CompU и REFERENCE схемы измерения сигналов обратной ЭДС.

9. Подключить логический анализатор осциллографа к контактам макетной платы (D0-логический сигнал управления транзистором VT1 цепь HU; D1-VT2 цепь LU; D2-VT3 цепь HV; D3-VT4 цепь LV; D4-VT5 цепь HW; D5-VT6 цепь LW; D6 – показывает сигнал прерывания подключается PD13; D7 – показывает сигнал прерывания подключается к PD11 - X32:1.30, D8 – показывает сигнал инверсии PD13 - X32:1.32).

10. Включить питание макетной платы

11. Нажать кнопку SEL и запустить вращение электродвигателя в режиме разгона.

12. С помощью кнопок UP, DOWN, LEFT, RIGHT разгоняем электродвигатель. При этом по осциллографу контролируем обратную ЭДС. Пример осциллограммы представлен на рисунке 18.3.

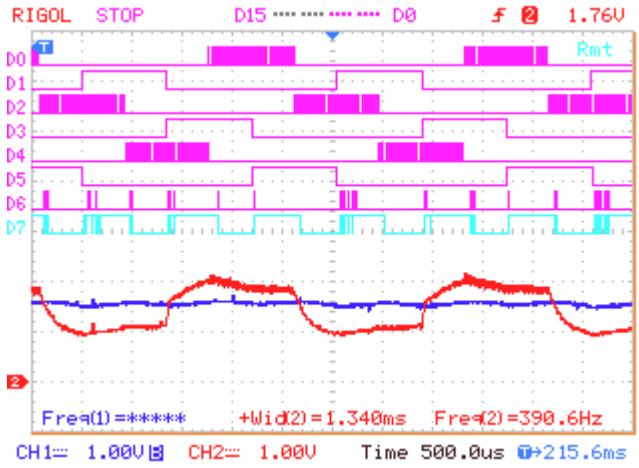


Рисунок 18.3. Осциллограмма полученная в режиме разгона двигателя (Run=True) Схема подключения см. пункт 9.

13. Как только переключение компаратора будут происходить четко, как на рис.18.3 нажмите одновременно две любые кнопке (кроме SEL). Тем самым включится режим работы по обратной ЭДС. (RunRun=TRUE). Далее электродвигатель разгонится и картинка на осциллографе примет вид Рис 18.4.

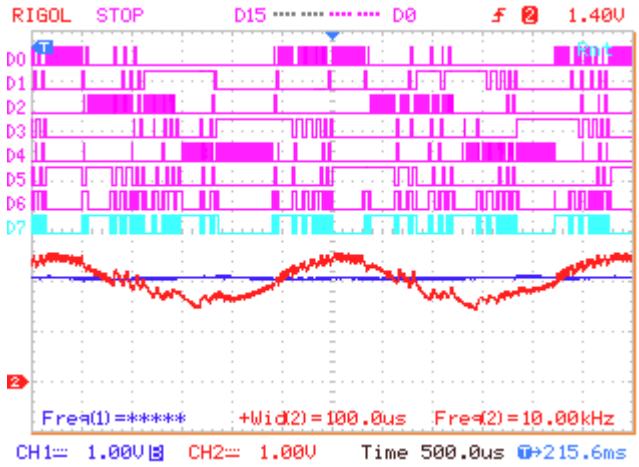


Рисунок 18.4 Осциллограмма полученная в режиме работы с использованием сигналов

обратной ЭДС (RunRun=True) Схема подключения см. пункт 9.

14. Фиксируем полученные осциллограммы для отчета.

Задания

1. Изменить направление вращения электродвигателя.
2. Изменить период ШИМ.
3. Оптимизировать функцию `void Set_Base_Vector(char base_vector)` для обеспечения вращения только в одну сторону.

Приложение

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_timer.h>
#include <MDR32F9Qx_uart.h>
#include <MDR32F9Qx_comp.h>
#include "joystick.h"

// Максимальная частота вращения привода CD ROM 24x 6000 об/мин или 100 Гц примем 120Гц (чтобы
// число делилось на 2, 3, 6)
// Число пар полюсов привода CD ROM 6
// Максимальная частота вращения, Гц
#define MAX_FREC 120
// Начальная частота вращения, Гц
#define START_FREC 5
// Число пар полюсов
#define POLUS    6
// Максимальная электрическая частота вращения, Гц
#define MAX_ELECTRIC_FREC (MAX_FREC*POLUS)
// Начальная электрическая частота вращения, Гц
#define START_ELECTRIC_FREC (START_FREC*POLUS)
// Шаг изменения частоты ()
#define STEP_ELECTRIC_FREC 1
// Максимальный период ШИМ ()
// Частота ШИМ может быть определена по формуле
// Частота ШИМ  FPWM = FCNT/(2*MAX_PWM)
// Коэффициент  2 отражает тот факт, что счет ведется в две стороны на одном периоде ШИМ
// Для нашего примера FPWM = 80000000/(2*4096)=19540 Гц;
#define MAX_PWM 0x554
// Максимальная скважность ШИМ
#define MAX_DUTY_CYCLE (MAX_PWM)
// Минимальная скважность ШИМ на минимальной частоте вращения
#define MIN_DUTY_CYCLE (MAX_PWM/5)
// Предделитель Таймера 2 (1999+1)
#define TIMER2_PRESCALER 1999
// Частота процессора и таймера Гц
#define CPU_CLK 80000000
// Модуль счета  Таймера 2
#define TIMER2_MODUL ((CPU_CLK /(TIMER2_PRESCALER+1))/(START_ELECTRIC_FREC*6) )

// Переменные для инициализации таймера
TIMER_CntInitTypeDef sTIM_CntInit;
TIMER_ChnInitTypeDef sTIM_ChnInit;
TIMER_ChnOutInitTypeDef sTIM_ChnOutInit;
TIMER_CntInitTypeDef TIM2Init;
// Переменные для инициализации порта и компаратора
PORT_InitTypeDef PORT_InitStructure;
COMP_InitTypeDef COMP_InitStructure;
COMP_CVRefInitTypeDef COMP_CVRefInitStructure;
// Переменная для хранения результата работы компаратора
uint32_t CompRezAs = 0;
// Значения длительности импульса по каналам 1,2,3
uint32_t Button_Val;
char Step;
char Run = 0 ;//Включение и выключение разгона (разгон по таймеру)
char RunRun = 0;//Включение и выключение разгона (разгон по компараторам)
char Inversion = 0;// Инверсия компаратора
uint32_t Duty_Cycle = MIN_DUTY_CYCLE;
uint32_t Period     = TIMER2_MODUL;
// Инициализация светодиодов на плате для индикации режима работы
void svetodiodinit(void)
{
```

```
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTD, ENABLE);
  PORT_InitStructure.PORT_Pin=PORT_Pin_10|PORT_Pin_11|PORT_Pin_12|PORT_Pin_13|PORT_Pin_14;
  PORT_InitStructure.PORT_OE    = PORT_OE_OUT;
  PORT_InitStructure.PORT_FUNC  = PORT_FUNC_PORT;
  PORT_InitStructure.PORT_MODE  = PORT_MODE_DIGITAL;
  PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
  PORT_Init(MDR_PORTD, &PORT_InitStructure);
}
// Инициализация кнопок на плате
void buttoninit(void)
{
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);
  PORT_InitStructure.PORT_Pin=PORT_Pin_10|PORT_Pin_11|PORT_Pin_12|PORT_Pin_13|PORT_Pin_14;
  PORT_InitStructure.PORT_OE    = PORT_OE_IN;
  PORT_InitStructure.PORT_FUNC  = PORT_FUNC_PORT;
  PORT_InitStructure.PORT_MODE  = PORT_MODE_DIGITAL;
  PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
  PORT_Init(MDR_PORTC, &PORT_InitStructure);
}
// Процедура инициализации таймера
void Timer2Init(void){
  RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER2, ENABLE);
  TIMER_BRGInit(MDR_TIMER2, TIMER_HCLKdiv1);
  TIMER_CntStructInit(&TIM2Init);    //Load defaults
  //Предделитель,значение 7999 соответствует делению частоты процессора на 8000
  // => частота таймера 10000Гц
  TIM2Init.TIMER_Prescaler        = TIMER2_PRESCALER;
  // Период ШИМ TIMER2_MODUL = (80000000/(7999+1))/(5*6*6)
  TIM2Init.TIMER_Period           = TIMER2_MODUL;
  TIMER_CntInit(MDR_TIMER2, &TIM2Init);
  //Разрешение прерывания по таймеру
  NVIC_EnableIRQ(Timer2_IRQn);
  NVIC_SetPriority(Timer2_IRQn, 0);
  TIMER_ITConfig(MDR_TIMER2, TIMER_STATUS_CNT_ZERO, ENABLE);
  TIMER_Cmd(MDR_TIMER2, ENABLE);
};
// Процедура переинициализации таймера при смене периода вращения.
void Timer2change(uint32_t Per){
  RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER2, ENABLE);
  TIMER_BRGInit(MDR_TIMER2, TIMER_HCLKdiv1);
  TIMER_CntStructInit(&TIM2Init);    //Load defaults
  // Предделитель, значение 7999 соответствует делению частоты процессора на 8000
  // => частота таймера 10000Гц
  TIM2Init.TIMER_Prescaler        = TIMER2_PRESCALER;
  // Период ШИМ 1667 = (80000000/(7999+1))/6
  TIM2Init.TIMER_Period           = Per;
  TIMER_CntInit(MDR_TIMER2, &TIM2Init);
  //Разрешение прерывания от таймера
  NVIC_EnableIRQ(Timer2_IRQn);
  NVIC_SetPriority(Timer2_IRQn, 0);
  TIMER_ITConfig(MDR_TIMER2, TIMER_STATUS_CNT_ZERO, ENABLE);
  TIMER_Cmd(MDR_TIMER2, ENABLE);
};
// Процедура установки нового базового вектора
void Set_Base_Vector(char base_vector){
  switch (base_vector){
    case(0)://Все транзисторы выключены
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_Only_0 );
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_0 );
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_0);
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_0);
      break;
    case(1)://Включаем ШИМ на обмотке U
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_REF);
      //Выключаем нижний ключ
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_0 );
      //Выключаем верхний ключ на обмотке V
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_Only_0);
      //Включаем нижний ключ на обмотке V
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_1);
      //Выключаем верхний ключ на обмотке W
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_Only_0);
      //Выключаем нижний ключ на обмотке W
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_0);
      break;
    case(2):
      //Включаем ШИМ на обмотке U
```

```
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_REF);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_0 );
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_0);
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_1);
      break;
    case(3):
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_0 );
      // Включаем ШИМ на обмотке V
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_REF);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_0);
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_1);
      break;
    case(4):
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_1 );
      //Включаем ШИМ на обмотке V
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_REF);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_0);
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_0);
      break;
    case(5):
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_1 );
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_0);
      // Включаем ШИМ на обмотке W
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_REF);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_0);
      break;
    case(6):
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_0 );
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_1);
      // Включаем ШИМ на обмотке W
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_REF);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_0);
      break;
    case(7): // Включаем все верхние ключи
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_Only_1);
      //Выключаем все нижние ключи
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_0 );
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_Only_1);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_0);
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_Only_1);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_0);
      break;
    case(8):
      //Выключаем все верхние ключи
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_Only_0);
      //Включаем все нижние ключи
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_1 );
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_1);
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_1);
      break;
  }
;};

// Прерывание по таймеру (выполняется в режиме разгона)
void Timer2_IRQHandler() {
  if(TIMER_GetITStatus(MDR_TIMER2, TIMER_STATUS_CNT_ZERO)){
    TIMER_ClearITPendingBit(MDR_TIMER2, TIMER_STATUS_CNT_ZERO);
    // Если включен режим разгона
              if (Run) {
                     Step--;
                     if (Step<1)     {Step=6;};
                     //Установим новый базовый вектор
                     Set_Base_Vector(Step);
              }
              else {
                     Step=8;
              }
```

```
    PORT_WriteBit(MDR_PORTD, PORT_Pin_10,  !PORT_ReadInputDataBit(MDR_PORTD, PORT_Pin_10));
    TIMER_ClearITPendingBit(MDR_TIMER2, TIMER_STATUS_CNT_ZERO);
  };
};
//Прерывание по компаратору
void COMPARATOR_IRQHandler() {
  PORT_WriteBit(MDR_PORTD, PORT_Pin_13,  Bit_SET);
  // Считывем значение компаратора
  CompRezAs=COMP_GetResultLatch();
  // На каждом шаге изменяем
  if (Inversion){
    COMP_OutInversionConfig(COMP_OUT_INV_Disable);
  }
  else{
    COMP_OutInversionConfig(COMP_OUT_INV_Enable);
  }
  PORT_WriteBit(MDR_PORTD, PORT_Pin_11,  Inversion);


  Inversion= !Inversion;
  // Если включен режим разгона по обратной ЭДС
  if (RunRun){
    Step++;
    if (Step>6){Step=1;};
    // На шагах 1,3,5 проверяем наличие логического нуля на выходе компаратора
    if ((Step==1)||(Step==3)||(Step==5)){
      if (!Inversion){Set_Base_Vector(Step);}
      else{//Если не так, то возвращаемся к предыдущему шагу
        Step--;
        if (Step==0){Step=6;};
      };
    }
    // На шагах 2,4,6 проверяем наличие логической единицы на выходе компаратора
    if ((Step==2)||(Step==4)||(Step==6)){
      if (Inversion){Set_Base_Vector(Step);}
      else{//Если не так, то возвращаемся к предыдущему шагу
        Step--;
        if (Step==0){Step=6;};
      };
    }
  };
  PORT_WriteBit(MDR_PORTD, PORT_Pin_13,  Bit_RESET);
};
//Установка частоты тактирования 80МГц
void CLK_init(void){/* Включение HSE осциллятора (внешнего кварцевого резонатора)*/
  RST_CLK_HSEconfig(RST_CLK_HSE_ON);
  if (RST_CLK_HSEstatus() == SUCCESS){  /* Если HSE осциллятор включился и прошел текст*/
    // Выбор HSE осциллятора в качестве источника тактовых импульсов для CPU_PLL
    // и установка умножителя тактовой частоты CPU_PLL равного 7
    // Частота внешнего кварца равна 8 МГц Максимальная частота процессора 80 МГц ,
    // RST_CLK_CPU_PLLconfig ( Источник тактирования PLL, Коэффициент умножения 9+1);
    // Коэффициент умножения=0 соответствует умножение на 1.
    RST_CLK_CPU_PLLconfig ( RST_CLK_CPU_PLLsrcHSEdiv1, 9 );
    /* Включение схемы PLL*/
    RST_CLK_CPU_PLLcmd(ENABLE);
    if (RST_CLK_CPU_PLLstatus() == SUCCESS) { //Если включение CPU_PLL прошло успешно
      /* Установка CPU_C3_prescaler = 1 */
      RST_CLK_CPUclkPrescaler(RST_CLK_CPUclkDIV1);
      /* Установка CPU_C2_SEL от CPU_PLL выхода вместо CPU_C1 такта*/
      RST_CLK_CPU_PLLuse(ENABLE);
      /*Выбор CPU_C3такта на мультиплексоре тактовых импульсов микропроцессора(CPU clock MUX)*/
      RST_CLK_CPUclkSelection(RST_CLK_CPUclkCPU_C3);
    }
    else /* блок CPU_PLL не включился*/ {while(1);}
  }
  else /* кварцевый резонатор HSE не включился */ {while(1);};
}

void timerPWMinit(void)
{
  //Включение тактирования порта A
  RST_CLK_PCLKcmd((RST_CLK_PCLK_PORTA), ENABLE);
  /* Сброс установок порта A*/
  PORT_DeInit(MDR_PORTA);
  /* Конфигурация выводов таймера TIMER1: CH1, CH1N, CH2, CH2N, CH3, CH3N  */
  /* Настройка порта A разряды 1, 2, 3, 4, 5 ,8 */
  // На демонстрационной плате в разъеме X33:1 контакты 35, 36, 31, 32, 29, 28 соответственно.
PORT_InitStructure.PORT_Pin=(PORT_Pin_1|PORT_Pin_2|PORT_Pin_3|PORT_Pin_4|PORT_Pin_5|PORT_Pin_8);
  PORT_InitStructure.PORT_OE    = PORT_OE_OUT;
```

```
  PORT_InitStructure.PORT_FUNC  = PORT_FUNC_ALTER;
  PORT_InitStructure.PORT_MODE  = PORT_MODE_DIGITAL;
  PORT_InitStructure.PORT_SPEED = PORT_SPEED_FAST;
  PORT_Init(MDR_PORTA, &PORT_InitStructure);
  // Включение тактирования таймера 1
  RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER1,ENABLE);
  /* Сброс установок таймера 1 */
  TIMER_DeInit(MDR_TIMER1);
  // Заполнение структуры значениями по умолчанию
  TIMER_CntStructInit(&sTIM_CntInit);
  //Предделитель частоты таймера равен 1 то есть частота таймера равна частоте микроконтоллера.
  sTIM_CntInit.TIMER_Prescaler               = 0x0;
  // Модуль счета = MAX_PWM
  // Период импульсов ШИМ = 2 * MAX_PWM  / частота счета таймера .
  // Для нашего примера частота ШИМ FPWM = 80000000/(2*(0x554))=29325 Гц;
  sTIM_CntInit.TIMER_Period                  = MAX_PWM ;
  // Счет в две стороны
  sTIM_CntInit.TIMER_CounterMode =   TIMER_CntMode_ClkChangeDir  ;
  // Инициализация таймера 1
  TIMER_CntInit (MDR_TIMER1,&sTIM_CntInit);
  /* Инициализация каналов таймера 1:  CH1,CH1N,CH2,CH2N,CH3,CH3N */
  TIMER_ChnStructInit(&sTIM_ChnInit);
  // Режим работы канала - генерация ШИМ
  sTIM_ChnInit.TIMER_CH_Mode                 = TIMER_CH_MODE_PWM;
  // Формат выработки сигнала REF номер 6:
  // при счете вверх – REF = 1 если  CNT<CCR, иначе REF = 0;
  // значение регистра CCR определяет длительность импульса ШИМ
  sTIM_ChnInit.TIMER_CH_REF_Format           = TIMER_CH_REF_Format6;
  // обновление регистра CCR в момент равенства нулю счетчика
  sTIM_ChnInit.TIMER_CH_CCR_UpdateMode =  TIMER_CH_CCR_Update_On_CNT_eq_0;
  // Инициализируем канал 1
  sTIM_ChnInit.TIMER_CH_Number               = TIMER_CHANNEL1;
  TIMER_ChnInit(MDR_TIMER1, &sTIM_ChnInit);
  // Инициализируем канал 2
  sTIM_ChnInit.TIMER_CH_Number               = TIMER_CHANNEL2;
  TIMER_ChnInit(MDR_TIMER1, &sTIM_ChnInit);
  // Инициализируем канал 3
  sTIM_ChnInit.TIMER_CH_Number               = TIMER_CHANNEL3;
  TIMER_ChnInit(MDR_TIMER1, &sTIM_ChnInit);
  /* Инициализация прямого и инверсного выходов (стр289 описания!!!)
  для каждого из каналов таймера 1: CH1,CH1N,CH2,CH2N,CH3,CH3N */
  // Заполнение элементов структуры значениями по умолчанию
  TIMER_ChnOutStructInit(&sTIM_ChnOutInit);
  // Выключаем верхний ключ
  sTIM_ChnOutInit.TIMER_CH_DirOut_Source    = TIMER_CH_OutSrc_Only_0   ;
  // Если нет необходиости в DTG то раскомментируйте следующую строчку
  // sTIM_ChnOutInit.TIMER_CH_NegOut_Source = TIMER_CH_OutSrc_REF
  // Настройна прямого выхода микроконтроллера CHxN на вывод данных
  sTIM_ChnOutInit.TIMER_CH_DirOut_Mode             = TIMER_CH_OutMode_Output;
  //Выключаем нижний ключ
  sTIM_ChnOutInit.TIMER_CH_NegOut_Source    = TIMER_CH_OutSrc_Only_0   ;
  // Если нет необходиости в DTG то раскомментируйте следующую строчку
  // sTIM_ChnOutInit.TIMER_CH_NegOut_Source = TIMER_CH_OutSrc_REF
  // Настройна инверсного выхода микроконтроллера CHxN на вывод данных
  sTIM_ChnOutInit.TIMER_CH_NegOut_Mode = TIMER_CH_OutMode_Output;
  // Настраиваем выходы канала 1
  sTIM_ChnOutInit.TIMER_CH_Number                   = TIMER_CHANNEL1;
  TIMER_ChnOutInit(MDR_TIMER1, &sTIM_ChnOutInit);
  // Настраиваем выходы канала 2
  sTIM_ChnOutInit.TIMER_CH_Number                   = TIMER_CHANNEL2;
  TIMER_ChnOutInit(MDR_TIMER1, &sTIM_ChnOutInit);
  // Настраиваем выходы канала 3
  sTIM_ChnOutInit.TIMER_CH_Number                   = TIMER_CHANNEL3;
  TIMER_ChnOutInit(MDR_TIMER1, &sTIM_ChnOutInit);
  /* Включаем делитель тактовой частоты таймера 1*/
  TIMER_BRGInit(MDR_TIMER1,TIMER_HCLKdiv1);
  /* После всех настроек разрешаем работу таймера 1 */
  TIMER_Cmd(MDR_TIMER1,ENABLE);
};

//Включение компаратора
void COMPInit(void){
  /* Включаем тактирование порта PORTE */
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTE,ENABLE);
  PORT_StructInit( &PORT_InitStructure);
  PORT_InitStructure.PORT_PULL_UP = PORT_PULL_UP_OFF;
  PORT_InitStructure.PORT_PULL_DOWN = PORT_PULL_DOWN_OFF;
  PORT_InitStructure.PORT_MODE = PORT_MODE_ANALOG;
  PORT_InitStructure.PORT_SPEED      = PORT_SPEED_SLOW ;
```

```
  PORT_InitStructure.PORT_OE = PORT_OE_IN;
  PORT_InitStructure.PORT_Pin = PORT_Pin_2|PORT_Pin_3|PORT_Pin_4|PORT_Pin_5|PORT_Pin_8;
  // Настраиваем ножки порта E как входы компаратора
  PORT_Init(MDR_PORTE, &PORT_InitStructure);
  /* Включаем тактирование компаратора*/
  RST_CLK_PCLKcmd(RST_CLK_PCLK_COMP,ENABLE);
  /* Заполяем структурц для инициализации компаратора значениями по умолчанию*/
  COMP_StructInit(&COMP_InitStructure);
  /* Настраиваем входы компаратора */
  COMP_InitStructure.COMP_PlusInputSource = COMP_PlusInput_CVREF;
  COMP_InitStructure.COMP_MinusInputSource = COMP_MinusInput_IN1;
  /* Инициализация компаратора*/
  COMP_Init(&COMP_InitStructure);
  /*Заполняем структуру для инициализации источника опорного напряженияCOMP_CVRefInitStructure*/
  COMP_CVRefStructInit(&COMP_CVRefInitStructure);

  /*Выбирем верхний диапазон работы компаратора и делитель 8/32 (В принципе может быть любой)
       поскольку на входы ref+ ref- подается  одинаковое напряжение */
  COMP_CVRefInitStructure.COMP_CVRefSource = COMP_CVREF_SOURCE_VRef;
  COMP_CVRefInitStructure.COMP_CVRefRange = COMP_CVREF_RANGE_Up;
  COMP_CVRefInitStructure.COMP_CVRefScale = COMP_CVREF_SCALE_8_div_32;
  /* Инициализация источника опорного напряжения */
  COMP_CVRefInit(&COMP_CVRefInitStructure);
  /* Разрешение работы компароатора COMP  */
  COMP_Cmd(ENABLE);
  /* Check READY flag */
  while(COMP_GetCfgFlagStatus(COMP_CFG_FLAG_READY) != SET);
    NVIC_EnableIRQ(COMPARATOR_IRQn);
    NVIC_SetPriority(COMPARATOR_IRQn, 2);
    COMP_ITConfig(ENABLE);
    /* Enables COMP CVRef */
    COMP_CVRefCmd(ENABLE);
};

int main(){
  CLK_init() ;
  buttoninit();
  svetodiodinit();
  COMPInit();
  timerPWMinit();
  Timer2Init();
  Step=0;
  while(1){
// Считываем данные с кнопок.
    PORT_WriteBit(MDR_PORTD, PORT_Pin_12, Bit_SET);
    Button_Val=PORT_ReadInputData(MDR_PORTC);
// Проверяем какая кнопка нажата
// Если нажата кнопка SEL то включаем выключаем двигатель
    if KEY_PRESSED(SEL){
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_SET);
      WAIT_UNTIL_KEY_RELEASED(SEL)
      if (Run) {Run=0;Step=8;Set_Base_Vector(Step);}
      else {Run=1;Step=0;};
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Run);
    }
// Если нажата кнопка UP то увеличиваем скорость двигателя (уменьшаем модуль таймера2)
// при этом увеличивается частота коммутации обмоток
    if KEY_PRESSED(UP){
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_SET);
      WAIT_UNTIL_KEY_RELEASED(UP)
      Period=Period-2;
      if (Period < 5)Period = 5;
      Timer2change(Period);
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_RESET);
    }
// Если нажата кнопка DOWN то уменьшаем  скорость двигателя (увеличиваем модуль таймера2)
// при этом уменьшается  частота коммутации обмоток
    if KEY_PRESSED(DOWN){
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_SET);
      WAIT_UNTIL_KEY_RELEASED(DOWN)
      Period=Period+6;
      if (Period > 200)Period = 200;
      Timer2change(Period);
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_RESET);
    }
// Если нажата кнопка RIGHT то уменьшаем скважность ШИМ при этом уменьшается  потребляемый ток
// в режиме разгона и скорость двигателя в режиме работы без датчиков
    if KEY_PRESSED(RIGHT){
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_SET);
```

```
      WAIT_UNTIL_KEY_RELEASED(RIGHT)
      if (Duty_Cycle>0)Duty_Cycle=Duty_Cycle-10;
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_RESET);
    }
// Если нажата кнопка LEFT то увеличивается скважность ШИМ при этом увеличивается потребляемый
// ток в  режиме разгона и скорость двигателя в режиме работы без датчиков
    if KEY_PRESSED(LEFT){
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_SET);
      WAIT_UNTIL_KEY_RELEASED(LEFT)
      if (Duty_Cycle<MAX_DUTY_CYCLE) Duty_Cycle=Duty_Cycle+10;
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_RESET);
    }
// Если нажата кнопка MULTIPLE то происходит переключение из режима разгона
// в режим работы без датчиков и наоборот

    if KEY_PRESSED(MULTIPLE ){
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_SET);
      WAIT_UNTIL_KEY_RELEASED(MULTIPLE )
      RunRun = !RunRun;
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_RESET);
    }
    TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL1, Duty_Cycle);
    TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL2, Duty_Cycle);
    TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL3, Duty_Cycle);

  }; // while(1);
}//main();
```

Лабораторная работа №19 Управление трехфазным бесколлекторным электродвигателем постоянного тока с использованием датчиков Холла

Цель работы: Изучение варианта использования микроконтроллера КР1986ВЕ91Т для управления бесколлекторным электродвигателем постоянного тока с использованием датчиков Холла.

Приборы и материалы:

1. Отладочная плата MDR1986VE91T Rev 4.
2. Программатор J-Link ARM.
3. Блок питания 5В, 1.4А.
4. ПК с установленной средой программирования Keil uVision.
5. Макетная плата с силовым инвертором и схемой измерения сигналов вырабатываемых датчиками Холла установленными на электродвигателе.
6. Бесколлекторный электродвигатель постоянного тока от шпинделя CD ROM без нагрузки. Электродвигатель необходим для визуализации вращения.
7. Осциллограф с логическим анализатором.

Сведения для выполнения

В электродвигателях постоянного тока коллекторный узел управляет коммутацией обмоток. В нужный момент времени подключается нужная обмотка. В БКЭПТ коммутацией обмоток управляет электроника. Для определения момента коммутации электроника может использовать датчики положения. Датчики положения наиболее часто используются в приложениях, где пусковой момент существенно варьируется или где требуется его высокое значение.

Чтобы вызвать вращение ротора необходимо пропустить ток через обмотки статора в определенной последовательности, задавая вращение в одном направлении, например, по часовой стрелке. Изменение последовательности коммутации приводит к реверсированию

двигателя (вращение в противоположном направлении). Трехфазные бесколлекторные электродвигатели постоянного тока характеризуются шестью состояниями коммутации. Когда все шесть состояний в последовательности коммутации выполнены, то для продолжения вращения последовательность повторяется. Последовательность определяет полное электрическое вращение. У двигателей с несколькими полюсами электрическое вращение не соответствует механическому вращению. Четырехполюсные бесколлекторные электродвигатели постоянного тока использует четыре электрических цикла вращения для выполнения одного механического вращения. Осциллограмма представленная на рисунке 19.1 показывает работу бесколлекторного электродвигателя постоянного тока. На ней обозначены 6 состояний коммутации обмоток (базовых векторов) цифрами P1-P6. Эти состояния соответствуют таблице 19.1 базовых векторов[15].

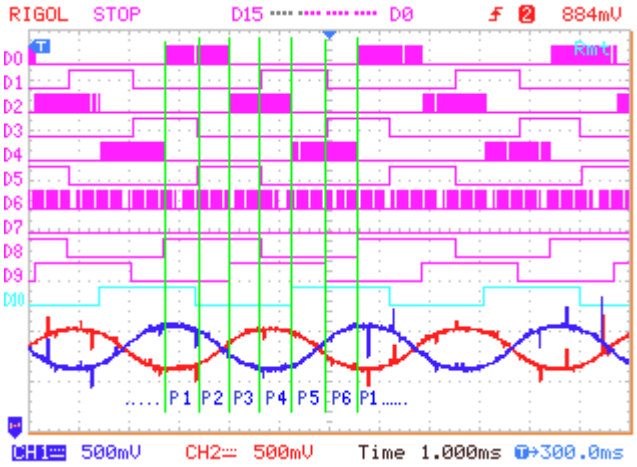


Рисунок 19.1 Осциллограмма управляющих сигналов электродвигателя, схема подключения соответствует Рис 17.3, Рис 19.4, Таблице 17.1, Таблице19.2: (D0- логический сигнал управления транзистором VT1 цепь HU; D1-VT2 цепь LU; D2-VT3 цепь HV; D3-VT4 цепь LV; D4-VT5 цепь HW; D5-VT6 цепь LW; D6 – показывает сигнал прерывания таймера; D7 –не подключен , D8, D9, D10 – Выходные сигналы датчиков Холла UVW подключаются соответственно к выводам PC2 (X33.2:12), PC4 (X33.2:6), PC6 (X33.2:8) микроконтроллера (контактам отладочной платы). .

Таблица 19.1. Состояние ключей инвертора в зависимости от положения, вычисленного по показаниям датчиков Холла.

Состояние комму-тации	Датчики Холла			Верхний ключ фазы U	Нижний ключ фазы U	Верхний ключ фазы V	Нижний ключ фазы V	Верхний ключ фазы W	Нижний ключ фазы W
	U	V	W						
P1	1	0	1	ШИМ	закрыт	закрыт	открыт	закрыт	закрыт
P2	1	0	0	ШИМ	закрыт	закрыт	закрыт	закрыт	открыт
P3	1	1	0	закрыт	закрыт	ШИМ	закрыт	закрыт	открыт
P4	0	1	0	закрыт	открыт	ШИМ	закрыт	закрыт	закрыт
P5	0	1	1	закрыт	открыт	закрыт	закрыт	ШИМ	закрыт
P6	0	0	1	закрыт	закрыт	закрыт	открыт	ШИМ	закрыт

Описание лабораторной установки

Силовая часть схемы аналогична Рис.17.3 Для измерения положения ротора применяется схема с тремя датчиками Холла Рис19.4. Датчики Холла располагаются на электродвигателе под углом 120 электрических градусов. Подключение схемы изменения

сигналов с датчиков Холла к отладочной плате 1986BE91T осуществляется согласно Таблицы 19.2.

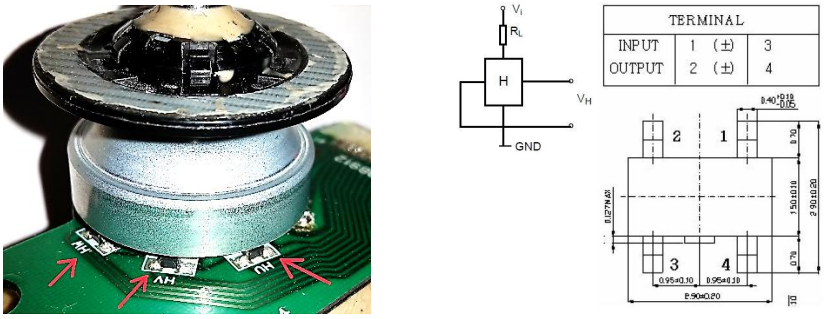


Рисунок 19.2 Слева - 3 датчика Холла на электродвигателе. Справа - схема включения и габаритный чертеж датчика Холла.

Питание датчиков Холла осуществляется с помощью стабилизатора напряжения DA9. Сигнал с датчиков Холла обрабатывается компараторами. Для обработки сигнала датчиков Холла применяется компаратор с гистерезисом выходные сигналы датчиков Холла подключаются к цепям HU2, HU4, HV2, HV4, HW2, HW4.

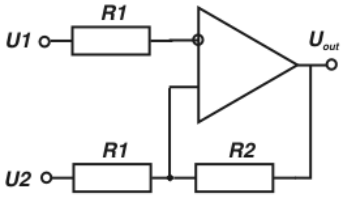


Рисунок 19.3 Схема компаратора с гистерезисом.

Гистерезис определяется по формуле.

$$\Delta U = (R1/R2)(U_{out}max - U_{out}min)$$

Таблица 19.2 Подключение схемы обработки сигнала датчика Холла к микроконтроллеру.

№	Порт микро-контроллера, цепь	Контакт разъема 1986BE91_EvBrd	Функция порта	Цепь макетной платы (рис___)	Коммен-тарий
1	PC2	X33:2.12	Цифровой вход	HallU	Фаза U
2	PC4	X33:2.6	Цифровой вход	HallV	Фаза V
3	PC6	X33:2.8	Цифровой вход	HallW	Фаза W

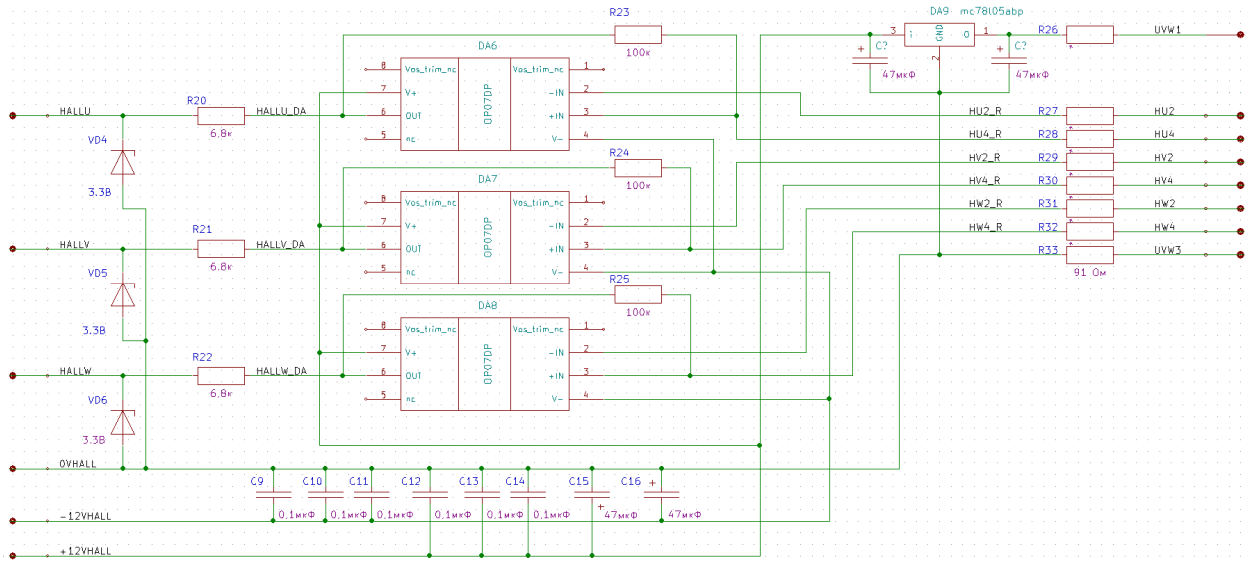


Рисунок 19.4 Схема обработки сигналов вырабатываемых датчиками Холла

Описание программы

Полный текст программы приведен в приложении. В программе используются:

Таймер 1, для генерации ШИМ

Таймер 3, периодически генерирует прерывания по которым осуществляется проверка состояния датчиков. И в зависимости от состояния датчиков выбирается соответствующий базовый вектор.

Кнопки отладочной платы осуществляют управление стендом.

Для настройки микроконтроллера выполняются следующие процедуры:

Включение внешнего тактового генератора на 8 МГц и установка коэффициента умножения частоты равным 10. Это обеспечивает частоту микроконтроллера 80 МГц

```
CLK_init() ;// Инициализация тактирования
```

Инициализация кнопок стенда см таблицу 17.2. Инициализация портов на ввод цифровых данных.

```
buttoninit();// Инициализация кнопок
```

Инициализация светодиодов стенда см. таблицу 17.2. Инициализация портов на вывод цифровых данных.

```
svetodiodinit();// Инициализация светодиодов
```

Инициализация таймера для генерации ШИМ

```
timerPWMinit();// Инициализация таймера для генерации ШИМ
```

Рассмотрим данную процедуру по порядку:

1 – инициализируются порты (PA1-PA5,PA8) в качестве выводов ШИМ таймера 1;

2 – с помощью структуры `sTIM_CntInit` инициализируются таймер в режиме двунаправленного счета

3 – с помощью структуры `sTIM_ChnInit` инициализируются три канала таймера в режиме генерации ШИМ. Обновление регистра CCR происходит при равенстве 0 счетчика таймера, что соответствует по времени центру осциллограммы трехфазной ШИМ Рис.__. Этим обеспечивается одновременное обновление всех каналов таймера и отсутствие неполных периодов ШИМ.

4 - с помощью структуры `sTIM_ChnOutInit` инициализируется выходные схемы таймеров. В начальный момент времени все транзисторы закрываются (TIMER_CH_OutSrc_Only_0).

Инициализация порта С для приема сигналов с датчиков Холла

```
void PortTimer3init(void)
```

Инициализация таймера для периодической проверки состояния датчиков Холла и смены базовых векторов.

```
Timer3Init();
```

Таймер 3 инициализируется на частоте 40МГц (Предделитель =1 , следовательно частота равна 80000000/2) Модуль счета равен 2222. Также инициализируется прерывание таймера по обнулению счетчика. Данное прерывание срабатывает с частотой 18кГц.

Обработчик прерывания таймера 3.

```
void Timer3_IRQHandler()
```

В прерывании таймера 3 считывается состояние датчиков Холла. Для исключения ложных срабатываний считывание происходит до тех пор пока два последовательных считывания не выдадут один результат.

В основном цикле программы `while(1)` происходит проверка нажатия кнопок стенда. В зависимости от того какая кнопка нажата выполняются различные действия. Рассмотрим назначение кнопок в программе:

SEL – включение/выключение режима вращения электродвигателя

UP – при нажатии кнопки двигатель проворачивается на 1 шаг коммутации

DOWN – при нажатии кнопки двигатель проворачивается на 1 шаг коммутации в обратном направлении

LEFT – Увеличение скважности ШИМ. Увеличивает переменную `Duty_Cycle`. Это приводит к увеличению скорости электродвигателя при работе по датчикам. Изменение скважности осуществляется функциями типа:

```
TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL1, Duty_Cycle).
```

RIGHT – Уменьшение скважности ШИМ. Действует по принципу LEFT.

Процедура `void Set_Base_Vector(char base_vector)` устанавливает базовый вектор в соответствии с таблицей коммутации (Таблица 19.1).

Порядок работы

1. Собрать лабораторную установку. Подключить макетную плату с силовым инвертором к Отладочной плате MDR1986VE91T Rev 4. в соответствии с таблицей 17.21 и 19.2

2. Включить отладочную плату.

3. Создать новый проект, подключить в менеджере *Manage run-time environment* к проекту библиотеки Startup_MDR1986BE9x, PORT, TIMER, RST_CLK необходимые для работы.

4. Добавьте в папку проекта файлы *joystick.h* и *joystick.c* из «DEMO» примера, представленного на компакт диске Миландр для этой платы. Данный файл предназначен для обработки нажатия кнопок на стенде.

5. Добавьте *joystick.c* в проект. Для этого на вкладке *Project Window* щелкните правой кнопкой мыши на *SourceGroup1* , в появившемся меню выберите *Add Existing Files to Group.....* , далее в окне выбора выберите файл *joystick.c* см. рис.17.8

6. Скопировать в main.c код представленный в приложении. Затем скомпилировать проект.

7. Включить питание отладочной платы и произвести программирование МК.

8. Подключить осциллограф CH1- к контакту HV2, CH2 –к контакту HV4 датчика Холла

9. подключить логический анализатор осциллографа к контактам макетной платы (D0-логический сигнал управления транзистором VT1 цепь HU; D1-VT2 цепь LU; D2-VT3 цепь HV; D3-VT4 цепь LV; D4-VT5 цепь HW; D5-VT6 цепь LW; D6 – показывает сигнал

прерывания подключается PD13; D8 – выходной сигнал датчика Холла U подключается к контакту отладочной платы PC2 (X33.2:12), D9 - датчика V - PC4 (X33.2:6), D10 - датчика W PC6 (X33.2:8)

10. Включить питание макетной платы

11. Нажать кнопки UP и DOWN и фиксировать поворот ротора электродвигателя по и против часовой стрелки.

12. Нажать кнопку SEL и запустить вращение электродвигателя.

13. С помощью кнопок LEFT и RIGHT возможно регулировать скорость вращения.

14. Зафиксировать на осциллографе картинку работы электродвигателя аналогичную рис 19.1 .

Задания

1. Измените направление вращения на противоположное
2. Измените частоту ШИМ

Приложение

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_timer.h>
#include "joystick.h"

// Максимальная частота вращения привода CD ROM 24x 6000 об/мин или 100 Гц примем 120Гц (чтобы число делилось на 2, 3, 6)
// Число пар полюсов привода CD ROM 6
// Максимальная частота вращения, Гц
#define MAX_FREC 120
// Начальная частота вращения, Гц
#define START_FREC 5
// Число пар полюсов
#define POLUS    6
// Максимальная электрическая частота вращения, Гц
#define MAX_ELECTRIC_FREC (MAX_FREC*POLUS)
// Начальная электрическая частота вращения, Гц
#define START_ELECTRIC_FREC (START_FREC*POLUS)
// Шаг изменения частоты ()
#define STEP_ELECTRIC_FREC 1
// Максимальный период ШИМ ()
// Частота ШИМ может быть определена по формуле
// Частота ШИМ  FPWM = FCNT/(2*MAX_PWM)
// Коэффициент  2 отражает тот факт, что счет ведется в две стороны на одном периоде ШИМ
// Для нашего примера FPWM = 80000000/(2*4096)=19540 Гц;
#define MAX_PWM 0x554
// Максимальная скважность ШИМ
#define MAX_DUTY_CYCLE (MAX_PWM)
// Минимальная скважность ШИМ на минимальной частоте вращения
#define MIN_DUTY_CYCLE (MAX_PWM/5)
// Предделитель Таймера 2 (1999+1)
#define CPU_CLK 80000000
// Номера битов порта к которым подключены датчики
#define BHallU  2
#define BHallV  4
#define BHallW  6
// Значения кнопок
uint32_t Button_Val;
// Переменные для инициализации таймера
TIMER_CntInitTypeDef sTIM_CntInit;
TIMER_ChnInitTypeDef sTIM_ChnInit;
TIMER_ChnOutInitTypeDef sTIM_ChnOutInit;
PORT_InitTypeDef PORT_InitStructure;
TIMER_CntInitTypeDef TIM2Init;
char Step;
char Run =0 ;
char Vector=0;
//Величина ШИМ
uint32_t Duty_Cycle=MIN_DUTY_CYCLE;
// Инициализация светодиодов на плате для индикации режима работы
void svetodiodinit(void){
```

```
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTD, ENABLE);
  PORT_InitStructure.PORT_Pin=PORT_Pin_10|PORT_Pin_11|PORT_Pin_12|PORT_Pin_13|PORT_Pin_14;
  PORT_InitStructure.PORT_OE    = PORT_OE_OUT;
  PORT_InitStructure.PORT_FUNC  = PORT_FUNC_PORT;
  PORT_InitStructure.PORT_MODE  = PORT_MODE_DIGITAL;
  PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
  PORT_Init(MDR_PORTD, &PORT_InitStructure);
}
// Инициализация кнопок на плате
void buttoninit(void){
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);
  PORT_InitStructure.PORT_Pin=PORT_Pin_10|PORT_Pin_11|PORT_Pin_12|PORT_Pin_13|PORT_Pin_14;
  PORT_InitStructure.PORT_OE    = PORT_OE_IN;
  PORT_InitStructure.PORT_FUNC  = PORT_FUNC_PORT;
  PORT_InitStructure.PORT_MODE  = PORT_MODE_DIGITAL;
  PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
  PORT_Init(MDR_PORTC, &PORT_InitStructure);
}
void Set_Base_Vector(char base_vector){
  switch (base_vector){
    case(0)://Все транзисторы выключены
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_Only_0 );
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_0 );
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_0);
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_0);
      break;
    case(1):// Включаем ШИМ на обмотке U
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_REF);
      //Выключаем нижний ключ
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_0 );
      // Выключаем верхний ключ на обмотке U
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_Only_0);
      //Включаем нижний ключ
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_1);
      //Выключаем верхний ключ
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_Only_0);
      //Выключаем нижний ключ
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_0);
      break;
    case(2):
      // Включаем ШИМ на обмотке U
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_REF);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_0 );
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_0);
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_1);
      break;
    case(3):
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_0 );
      // Включаем ШИМ на обмотке V
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_REF);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_0);
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_1);
      break;
    case(4):
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_1 );
      // Включаем ШИМ на обмотке V
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_REF);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_0);
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_0);
      break;
    case(5):
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_1 );
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_0);
      // Включаем ШИМ на обмотке W
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_REF);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_0);
      break;
    case(6):
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_Only_0);
```

```
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_0 );
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_1);
      // Включаем ШИМ на обмотке W
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_REF);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_0);
      break;
    case(7):
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_Only_1);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_0 );
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_Only_1);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_0);
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_Only_1);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_0);
      break;
    case(8):
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL1,TIMER_CH_OutSrc_Only_1 );
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL2,TIMER_CH_OutSrc_Only_1);
      TIMER_ChnOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3, TIMER_CH_OutSrc_Only_0);
      TIMER_ChnNOutSourceConfig(MDR_TIMER1, TIMER_CHANNEL3,TIMER_CH_OutSrc_Only_1);
      break;
  };
};

void CLK_init(void){
  //Установка частоты тактирования 80МГц
  /* Включение HSE осциллятора (внешнего кварцевого резонатора)*/
  RST_CLK_HSEconfig(RST_CLK_HSE_ON);
  if (RST_CLK_HSEstatus() == SUCCESS){  /* Если HSE осциллятор включился и прошел текст*/
    // Выбор HSE осциллятора в качестве источника тактовых импульсов для CPU_PLL
    // и установка умножителя тактовой частоты CPU_PLL равного 7
    // Частота внешнего кварца равна 8 МГц Максимальная частота процессора 80 МГц ,
    // RST_CLK_CPU_PLLconfig ( Источник тактирования PLL, Коэффициент умножения 9+1);
    // Коэффициент умножения=0 соответствует умножение на 1.
    RST_CLK_CPU_PLLconfig ( RST_CLK_CPU_PLLsrcHSEdiv1, 9 );
    /* Включение схемы PLL*/
    RST_CLK_CPU_PLLcmd(ENABLE);
    if (RST_CLK_CPU_PLLstatus() == SUCCESS){ //Если включение CPU_PLL прошло успешно
      /* Установка CPU_C3_prescaler = 1 */
      RST_CLK_CPUclkPrescaler(RST_CLK_CPUclkDIV1);
      /* Установка CPU_C2_SEL от CPU_PLL выхода вместо CPU_C1 такта*/
      RST_CLK_CPU_PLLuse(ENABLE);
      /*Выбор CPU_C3 такта на мультиплексоре тактовых импульсов микропроцессора(CPU clock MUX)*/
        RST_CLK_CPUclkSelection(RST_CLK_CPUclkCPU_C3);
    }
    else/* блок CPU_PLL не включился*/   {  while(1);  }
  }
  else/* кварцевый резонатор HSE не включился */   {  while(1);  };
}

void timerPWMinit(void){
  //Включение тактирования порта A
  RST_CLK_PCLKcmd((RST_CLK_PCLK_PORTA), ENABLE);
  /* Сброс установок порта A*/
  PORT_DeInit(MDR_PORTA);
  /* Конфигурация выводов таймера TIMER1: CH1, CH1N, CH2, CH2N, CH3, CH3N  */
  /* Настройка порта A разряды 1, 2, 3, 4, 5 ,8 */
  // На демонстрационной плате в разъеме X33:1 контакты 35, 36, 31, 32, 29, 28 соответственно.
PORT_InitStructure.PORT_Pin =(PORT_Pin_1|PORT_Pin_2|PORT_Pin_3|PORT_Pin_4|PORT_Pin_5|PORT_Pin_8);
  PORT_InitStructure.PORT_OE    = PORT_OE_OUT;
  PORT_InitStructure.PORT_FUNC  = PORT_FUNC_ALTER;
  PORT_InitStructure.PORT_MODE  = PORT_MODE_DIGITAL;
  PORT_InitStructure.PORT_SPEED = PORT_SPEED_FAST;
  PORT_Init(MDR_PORTA, &PORT_InitStructure);

  // Включение тактирования таймера 1
  RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER1,ENABLE);
  /* Сброс установок таймера 1 */
  TIMER_DeInit(MDR_TIMER1);
  // Заполнение структуры значениями по умолчанию
  TIMER_CntStructInit(&sTIM_CntInit);
  // Предделитель частоты таймера равен 1 то есть частота таймера равна частоте микроконтоллера.
  sTIM_CntInit.TIMER_Prescaler               = 0x0;
  // Модуль счета = MAX_PWM
  // Период импульсов ШИМ = 2 * MAX_PWM  / частота счета таймера .
  sTIM_CntInit.TIMER_Period                  = MAX_PWM ;
  // Счет в две стороны
```

```
  sTIM_CntInit.TIMER_CounterMode =   TIMER_CntMode_ClkChangeDir  ;
  // Инициализация таймера 1
  TIMER_CntInit (MDR_TIMER1,&sTIM_CntInit);
  /* Инициализация каналов таймера 1:  CH1,CH1N,CH2,CH2N,CH3,CH3N */
  TIMER_ChnStructInit(&sTIM_ChnInit);
  // Режим работы канала - генерация ШИМ
  sTIM_ChnInit.TIMER_CH_Mode                = TIMER_CH_MODE_PWM;
  // Формат выработки сигнала REF номер 6:
  // при счете вверх - REF = 1 если  CNT<CCR, иначе REF = 0;
  // значение регистра CCR определяет длительность импульса ШИМ
  sTIM_ChnInit.TIMER_CH_REF_Format          = TIMER_CH_REF_Format6;
  // обновление регистра CCR в момент равенства нулю счетчика
  sTIM_ChnInit.TIMER_CH_CCR_UpdateMode =  TIMER_CH_CCR_Update_On_CNT_eq_0;
  // Инициализируем канал 1
  sTIM_ChnInit.TIMER_CH_Number              = TIMER_CHANNEL1;
  TIMER_ChnInit(MDR_TIMER1, &sTIM_ChnInit);
  // Инициализируем канал 2
  sTIM_ChnInit.TIMER_CH_Number              = TIMER_CHANNEL2;
  TIMER_ChnInit(MDR_TIMER1, &sTIM_ChnInit);
  // Инициализируем канал 3
  sTIM_ChnInit.TIMER_CH_Number              = TIMER_CHANNEL3;
  TIMER_ChnInit(MDR_TIMER1, &sTIM_ChnInit);

  /* Инициализация прямого и инверсного выходов (стр289 описания!!!)
  для каждого из каналов таймера 1: CH1,CH1N,CH2,CH2N,CH3,CH3N */
  // Заполнение элементов структуры значениями по умолчанию
  TIMER_ChnOutStructInit(&sTIM_ChnOutInit);
  // Выключаем верхний ключ
  sTIM_ChnOutInit.TIMER_CH_DirOut_Source    = TIMER_CH_OutSrc_Only_0   ;
  // Если нет необходиости в DTG то раскомментируйте следующую строчку
  // sTIM_ChnOutInit.TIMER_CH_NegOut_Source = TIMER_CH_OutSrc_REF
  // Настройна прямого выхода микроконтроллера CHxN на вывод данных
  sTIM_ChnOutInit.TIMER_CH_DirOut_Mode             = TIMER_CH_OutMode_Output;
  //Выключаем нижний ключ
  sTIM_ChnOutInit.TIMER_CH_NegOut_Source    = TIMER_CH_OutSrc_Only_0   ;
  // Если нет необходиости в DTG то раскомментируйте следующую строчку
  // sTIM_ChnOutInit.TIMER_CH_NegOut_Source = TIMER_CH_OutSrc_REF
  // Настройна инверсного выхода микроконтроллера CHxN на вывод данных
  sTIM_ChnOutInit.TIMER_CH_NegOut_Mode = TIMER_CH_OutMode_Output;

  // Настраиваем выходы канала 1
  sTIM_ChnOutInit.TIMER_CH_Number                    = TIMER_CHANNEL1;
  TIMER_ChnOutInit(MDR_TIMER1, &sTIM_ChnOutInit);
  // Настраиваем выходы канала 2
  sTIM_ChnOutInit.TIMER_CH_Number                    = TIMER_CHANNEL2;
  TIMER_ChnOutInit(MDR_TIMER1, &sTIM_ChnOutInit);
  // Настраиваем выходы канала 3
  sTIM_ChnOutInit.TIMER_CH_Number                    = TIMER_CHANNEL3;
  TIMER_ChnOutInit(MDR_TIMER1, &sTIM_ChnOutInit);
  /* Включаем делитель тактовой частоты таймера 1*/
  TIMER_BRGInit(MDR_TIMER1,TIMER_HCLKdiv1);
  /* После всех настроек разрешаем работу таймера 1 */
  TIMER_Cmd(MDR_TIMER1,ENABLE);
};

void timer3init(void){
  TIMER_CntInitTypeDef TIM3Init;
  //Задание начельных значений регистров для таймера.
  TIMER_DeInit(MDR_TIMER3);
  //Включение тактирования таймера
  RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER3, ENABLE);
  // Включение делителя тектовой частоты
  TIMER_BRGInit(MDR_TIMER3, TIMER_HCLKdiv1);
  TIMER_CntStructInit(&TIM3Init);
  // Задаем параметры счета
  TIM3Init.TIMER_Prescaler           = 1;
  TIM3Init.TIMER_Period              = 2222;
  // Инициализация таймера
  TIMER_CntInit (MDR_TIMER3, &TIM3Init);
  // Очистим флаги событий
  TIMER_ClearFlag(MDR_TIMER3, 0);
  // Разрешение прерываний по захвату фронта и среза
  TIMER_ITConfig(MDR_TIMER3, TIMER_STATUS_CNT_ZERO, ENABLE);
  // Включение прерывания от таймера
  NVIC_EnableIRQ(Timer3_IRQn);
  // Установка приоритета прерывания
  NVIC_SetPriority(Timer3_IRQn, 2);
  // Разрешение работы таймера
  TIMER_Cmd(MDR_TIMER3, ENABLE);
```

```
};


//Процедура обработки прерывания для таймера
void Timer3_IRQHandler(){
  char data, data1;
  data=1;
  // Визуализация прерывания
  PORT_WriteBit(MDR_PORTD, PORT_Pin_13,  Bit_SET);
  if(TIMER_GetITStatus(MDR_TIMER3, TIMER_STATUS_CNT_ZERO)){
    //Считываем данные 2 раза для уменьшения шумов
    data=(char)  (PORT_ReadInputData(MDR_PORTC)&((1<<BHallU)|(1<<BHallV)|(1<<BHallW)));
    data1=(char)      (PORT_ReadInputData(MDR_PORTC)&((1<<BHallU)|(1<<BHallV)|(1<<BHallW)));
    // Цикл пока не будет повтора данных.
    while (!(data==data1)){
      data1=data;
      data=(char)(PORT_ReadInputData(MDR_PORTC)&((1<<BHallU)|(1<<BHallV)|(1<<BHallW)));
    };
    //В зависимости от состояния датчиков холла устанавливаем вектора
    switch (data){
      case ((1<<BHallU)|(1<<BHallW)) :Vector=1;
      break;
      case ((1<<BHallU)) :Vector=2;
      break;
      case ((1<<BHallU)|(1<<BHallV)) :Vector=3;
      break;
      case ((1<<BHallV)) :Vector=4;
      break;
      case ((1<<BHallV)|(1<<BHallW)) :Vector=5;
      break;
      case ((1<<BHallW)) :Vector=6;
      break;
    };
    if (Run)   Set_Base_Vector(Vector);
    // Очистка флага прерывания
    TIMER_ClearITPendingBit(MDR_TIMER3, TIMER_STATUS_CNT_ZERO);
  };
  PORT_WriteBit(MDR_PORTD, PORT_Pin_13,  Bit_RESET);
};

void PortTimer3init(void){
  //Включение тактирования порта C
  RST_CLK_PCLKcmd((RST_CLK_PCLK_PORTC), ENABLE);
  /* Настройка порта C разряды 2, 4, 6 */
  // На демонстрационной плате в разъеме X33:2 контакты 12, 6, 8 соответственно.
  PORT_InitStructure.PORT_Pin = (PORT_Pin_2 | PORT_Pin_4 | PORT_Pin_6);
  PORT_InitStructure.PORT_OE    = PORT_OE_IN;
  PORT_InitStructure.PORT_FUNC  = PORT_FUNC_PORT ;
  PORT_InitStructure.PORT_MODE  = PORT_MODE_DIGITAL;
  PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW ;
  PORT_Init(MDR_PORTC, &PORT_InitStructure);
};

int main(){
  CLK_init() ;
  buttoninit();
  svetodiodinit();
  PortTimer3init();
  timer3init();
  timerPWMinit();
  Step=0;
  while(1){
    // Считываем данные с кнопок.
    PORT_WriteBit(MDR_PORTD, PORT_Pin_12, Bit_SET);
    Button_Val=PORT_ReadInputData(MDR_PORTC);
    // Проверяем какая кнопка нажата
    //Включение/выключение вращения
    if KEY_PRESSED(SEL){
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_SET);
      WAIT_UNTIL_KEY_RELEASED(SEL)
      if (Run) {
        Run=0;
        Step=8;
      }
      else{
        Run=1;
        TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL1, Duty_Cycle);
        TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL2, Duty_Cycle);
        TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL3, Duty_Cycle);
```

```
        Step=0;
      };
      Set_Base_Vector(Step);
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Run);
    }
    // Вращение в ручном режиме 1 нажатие 1 такт вращения
    if KEY_PRESSED(UP){
      WAIT_UNTIL_KEY_RELEASED(UP)
      Vector ++;
      if (Vector>6) {Vector=1;};
      Set_Base_Vector(Vector);
    }
    // Вращение в ручном режиме 1 нажатие 1 такт вращения
    if KEY_PRESSED(DOWN){
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_SET);
      WAIT_UNTIL_KEY_RELEASED(DOWN)
      if (Vector>0) {Vector --;} else {Vector=6;};
      Set_Base_Vector(Vector);
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_RESET);
    }
    // Уменьшение скорости вращения
    if KEY_PRESSED(RIGHT){
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_SET);
      WAIT_UNTIL_KEY_RELEASED(RIGHT)
      if (Run){
        if (Duty_Cycle>0)Duty_Cycle=Duty_Cycle-10;
        TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL1, Duty_Cycle);
        TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL2, Duty_Cycle);
        TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL3, Duty_Cycle);
      };
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_RESET);
    };
    // Увеличение скорости вращения
    if KEY_PRESSED(LEFT){
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_SET);
      WAIT_UNTIL_KEY_RELEASED(LEFT)
      if (Run){
        if (Duty_Cycle<MAX_DUTY_CYCLE) Duty_Cycle=Duty_Cycle+10;
        TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL1, Duty_Cycle);
        TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL2, Duty_Cycle);
        TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL3, Duty_Cycle);
      };
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_RESET);
    };
    if KEY_PRESSED(MULTIPLE ){
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_SET);
      WAIT_UNTIL_KEY_RELEASED(MULTIPLE )
      PORT_WriteBit(MDR_PORTD, PORT_Pin_14, Bit_RESET);
    }
  };//while
};//main
```

Литература

1. ТСКЯ.431296.001РЭ ТСКЯ.431296.001РЭ Микросхемы интегральные 1986ВЕ91Т, 1986ВЕ92У, 1986ВЕ93У, 1986ВЕ94Т Руководство по эксплуатации
2. Спецификация микроконтроллеров серии 1986ВЕ9х, К1986ВЕ9х, К1986ВЕ92QI, К1986ВЕ92QC, К1986ВЕ91Н4//
3. VS_COM User manual Edition: November 2012 //www.visionsystems.de
4. SJA1000 Stand-alone CAN controller, DATA SHEET. http://www.semiconductors.philips.com
5. Сайт для скачивания программы Putty
http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
6. USB in a NutShell - путеводитель по стандарту USB. Автор microsin
http://microsin.ru/content/view/1107/44/
7. ARM7: техника использования виртуального последовательного порта USB CDC
Автор microsin http://microsin.net/programming/arm-working-with-usb/iar-usb-cdc.html
8. Источник из интернета Личный кабинет: http://www.lichnycabinet.ru/v1/nezavisimyj-storozhevoj-tajmer
9. Источник из интернета: Википедия. Сторожевой таймер.
https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%BE%D1%80%D0%BE%D0%B6%D0%B5%D0%B2%D0%BE%D0%B9_%D1%82%D0%B0%D0%B9%D0%BC%D0%B5%D1%80
10. Олег Вальпа. Современные 32-разрядные ARM-микроконтроллеры серии STM32: сторожевые таймеры //Современная электроника, № 5 2015, С30-34.
11. Калачев Ю. Н. Векторное регулирование (заметки практика, Москва , компания «Эфо» 2013 г.71 с.
12. www.atmel.com. AVR494: AC Induction Motor Control Using the constant V/f Principle and a Natural PWM Algorithm.
13. www.gaw.ru. AVR494: Управление асинхронным электродвигателем переменного тока по принципу постоянства V/f и обычного ШИМ-управления
14. http://www.gaw.ru/html.cgi/txt/app/micros/avr/AVR444.htm. AVR444: Управление трехфазным бесколлекторным электродвигателем постоянного тока без датчиков
15. http://www.gaw.ru/html.cgi/txt/app/micros/avr/AVR443.htm: AVR443: Управление трехфазным бесколлекторным электродвигателем постоянного тока с использованием датчиков

Мы рады любому сотрудничеству и стремимся к тому, чтобы работа на нашем оборудовании проходила для Вас максимально плодотворно.

Более подробно узнать о деятельности компании «Миландр» можно на сайте milandr.ru. Задать вопросы, оставить отзывы или сделать предложение можно на нашем форуме forum.milandr.ru.

По вопросам организации и проведения курсов на основе элементной базы АО «ПКК Миландр»:

Кузнецов Михаил Викторович,
Руководитель направления по внедрению и развитию образовательных программ,
Москва, Зеленоград, Георгиевский проспект, дом 5,
тел.: +7 (495) 981-54-33, доб. 111,
e-mail: kuznetsov.m@milandr.ru