Алалуев Р. В. **Основы программирования 32-разрядных микроконтроллеров 1986ВЕ91Т компании «Миландр»**: руководство к выполнению лабораторных работ / Р. В. Алалуев, В.М. Глаголев, А. Я. Матвеев, Л. Л. Владимиров. – М., 2015. – 53 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.

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

Оглавление

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).

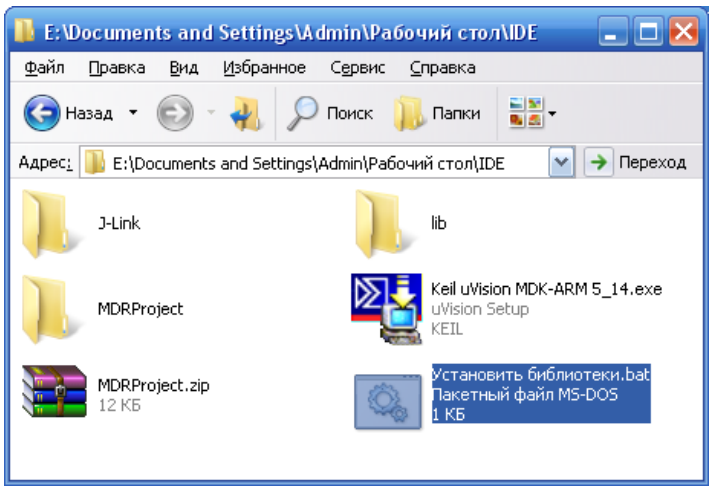


Рисунок 1.1 – Материалы лабораторного комплекса

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».

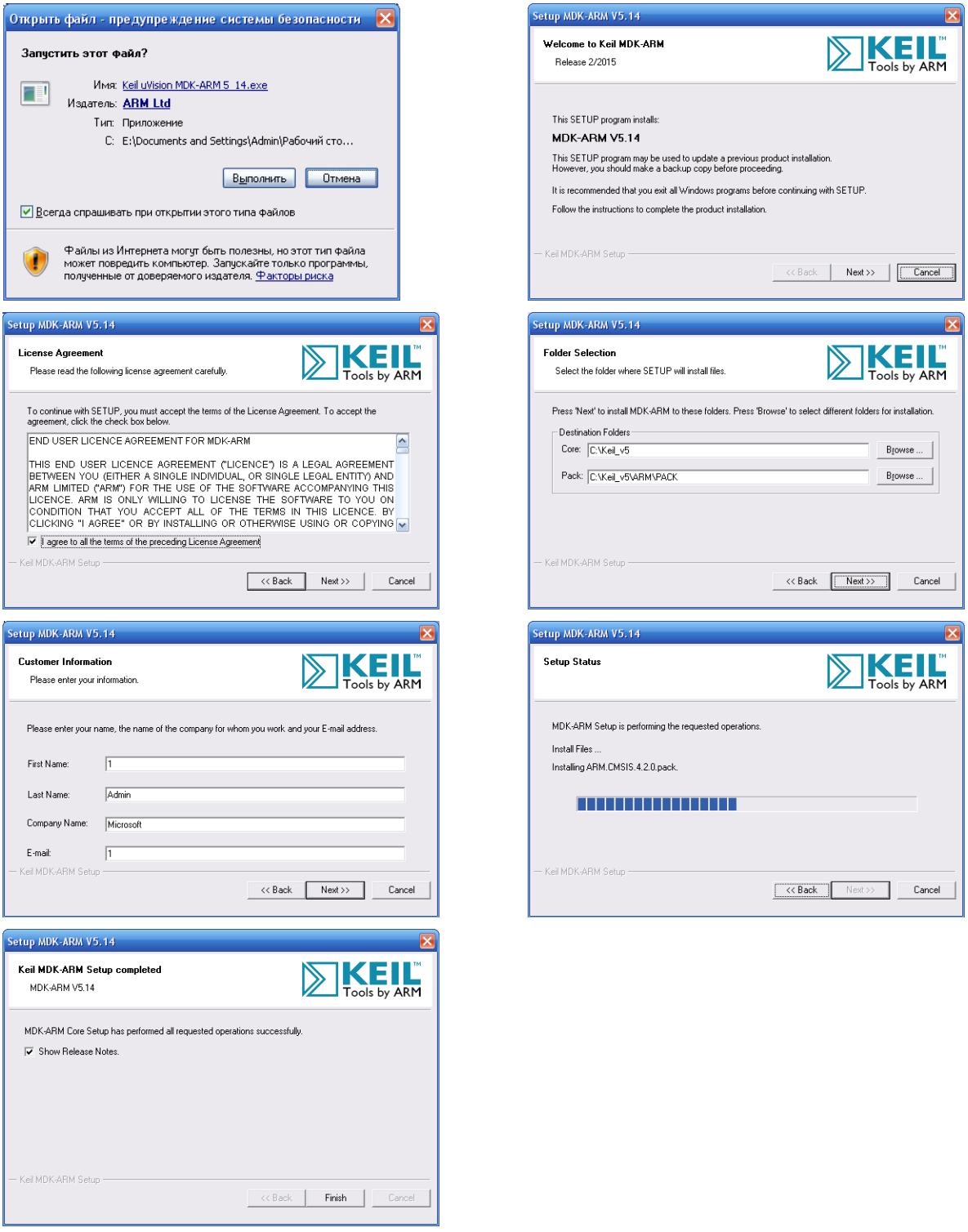


Рисунок 1.3 – Группа скриншотов процесса установки Keil uVision

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.

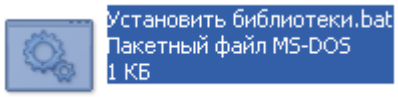


Рисунок 1.4 – Запуск установки библиотек.

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.

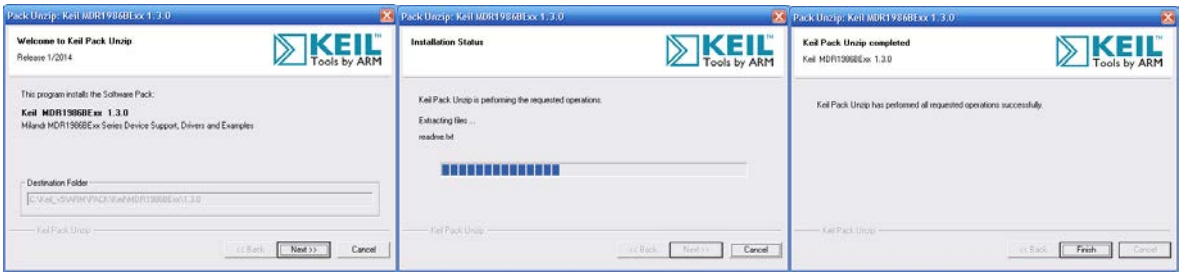


Рисунок 1.5 – Группа скриншотов процесса установки библиотек.

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.

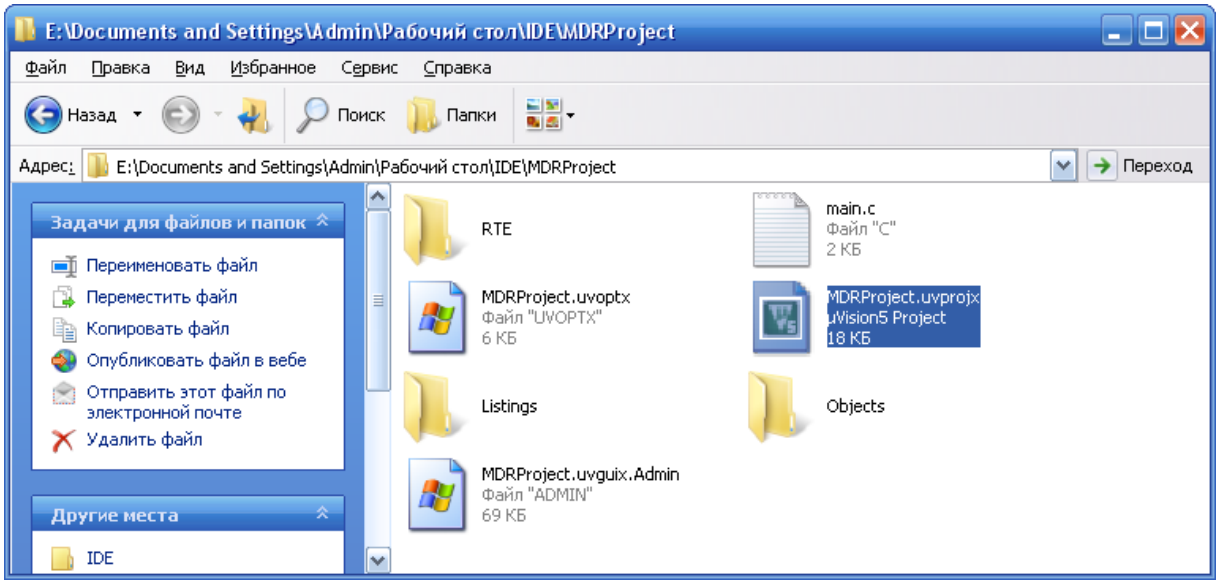


Рисунок 1.6 – Запуск первого проекта.

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

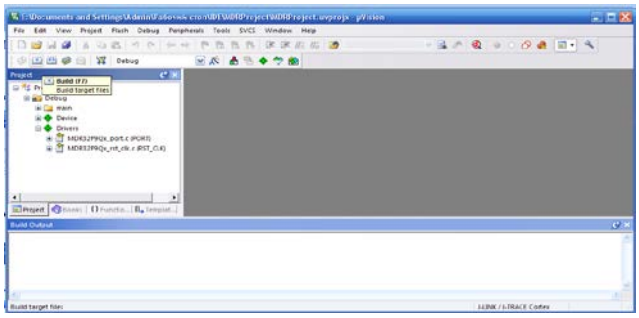


Рисунок 1.7 – Построение первого проекта.

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

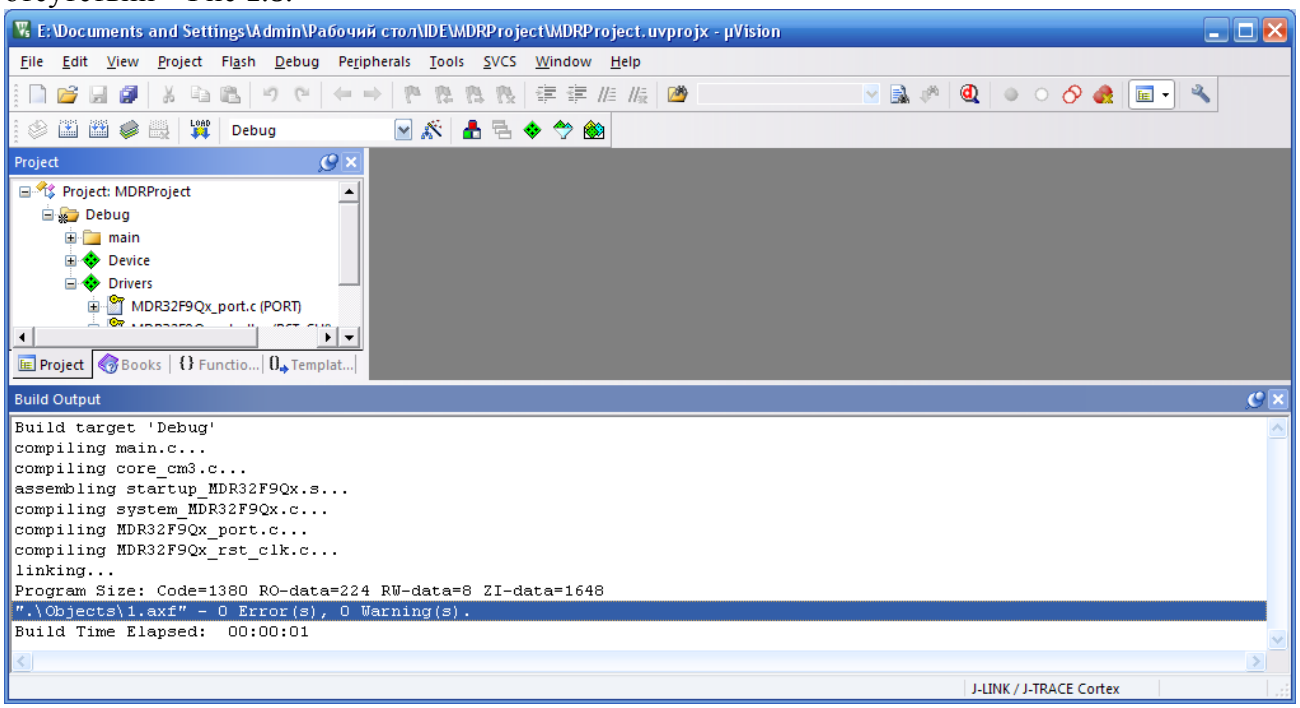


Рисунок 1.8 – Отсутствие ошибок при построении проекта.

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.

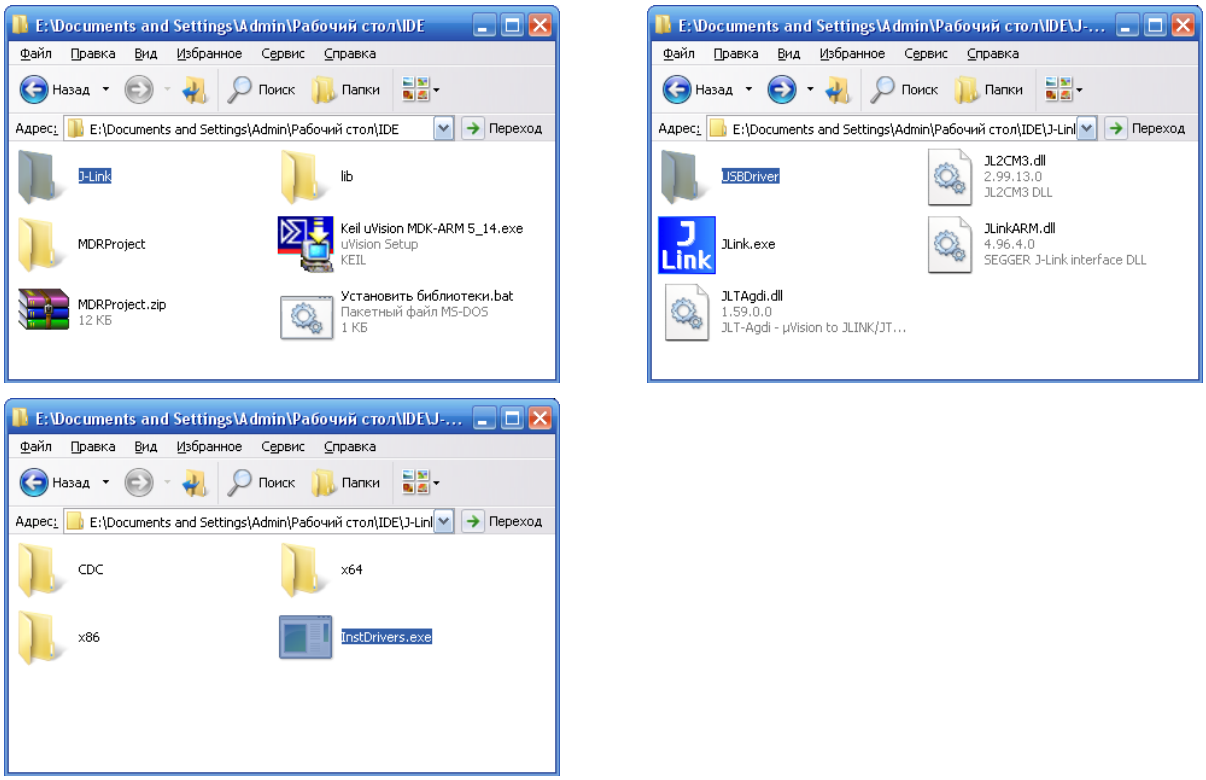


Рисунок 1.9 – Группа скриншотов процесса установки драйвера для программатора J-Link.

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**…

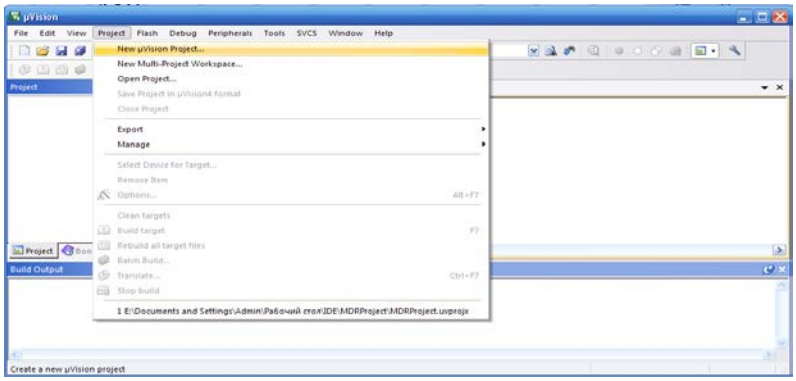


Рисунок 1.11 Создание нового проекта

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

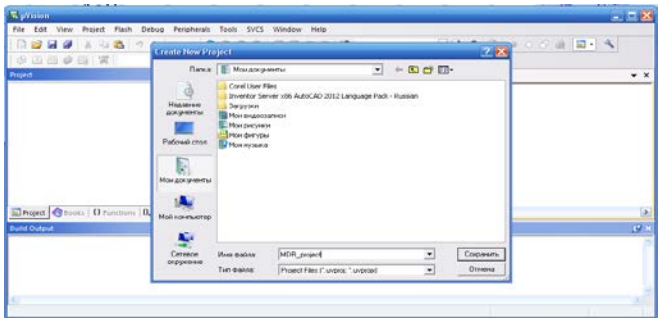


Рисунок 1.12 Процесс сохранения проекта

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

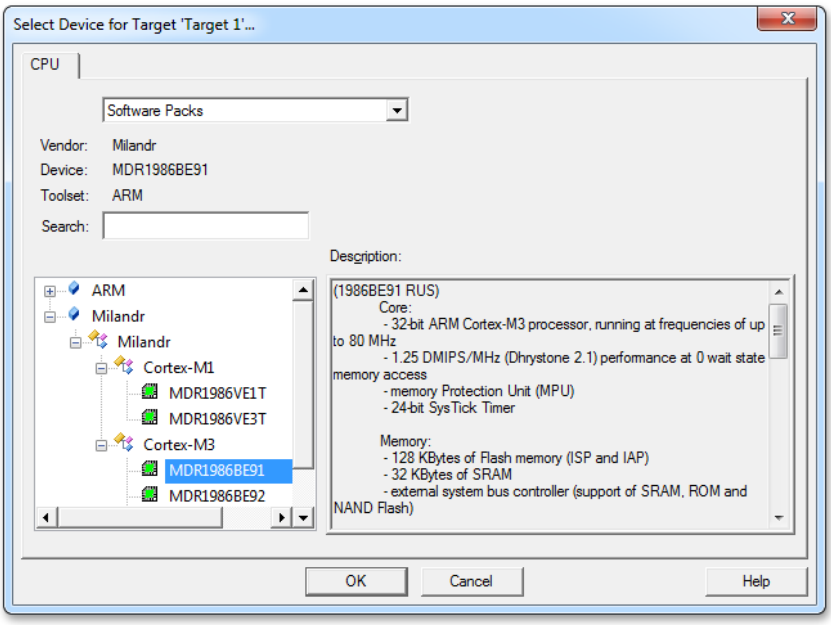


Рисунок 1.13 Выбор марки микропроцессора

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**.

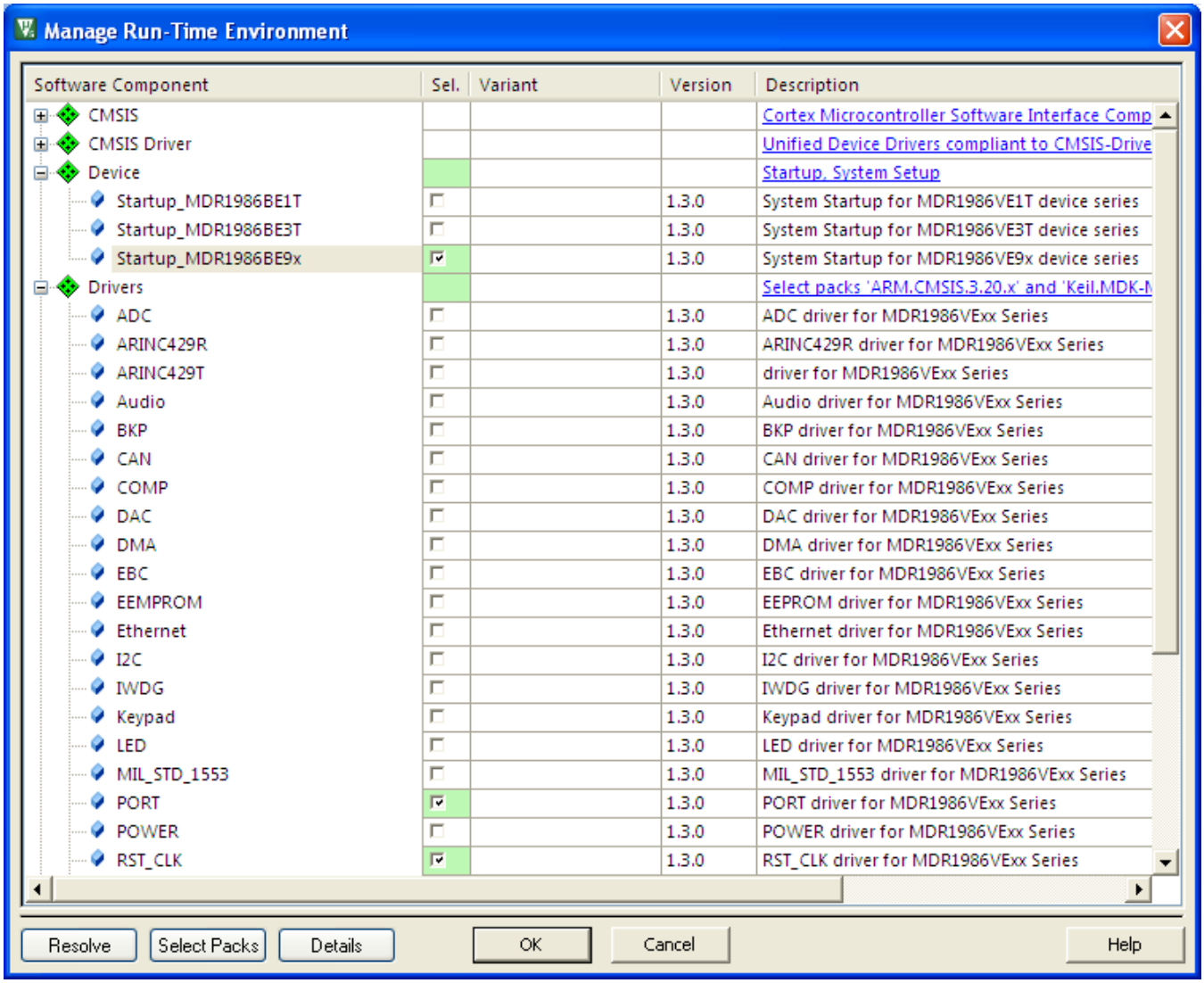


Рисунок 1.14 Настройка библиотек

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**.

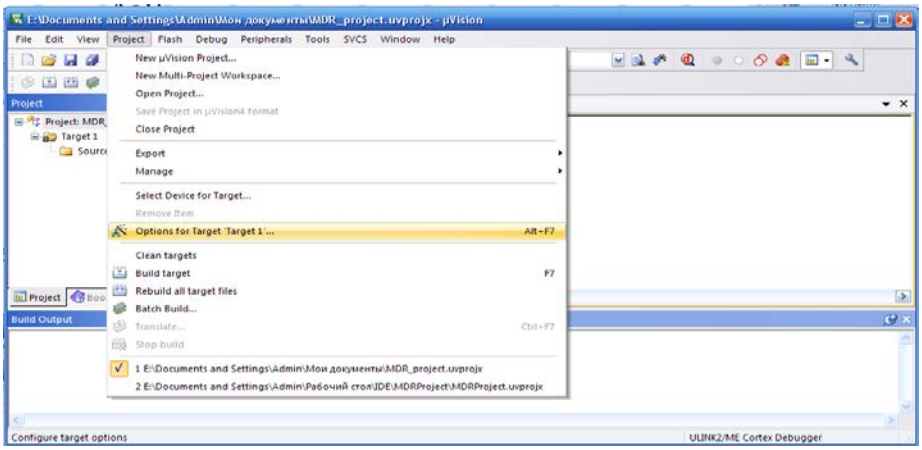


Рисунок 1.15 Выбор пункта меню настройка параметров проекта

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

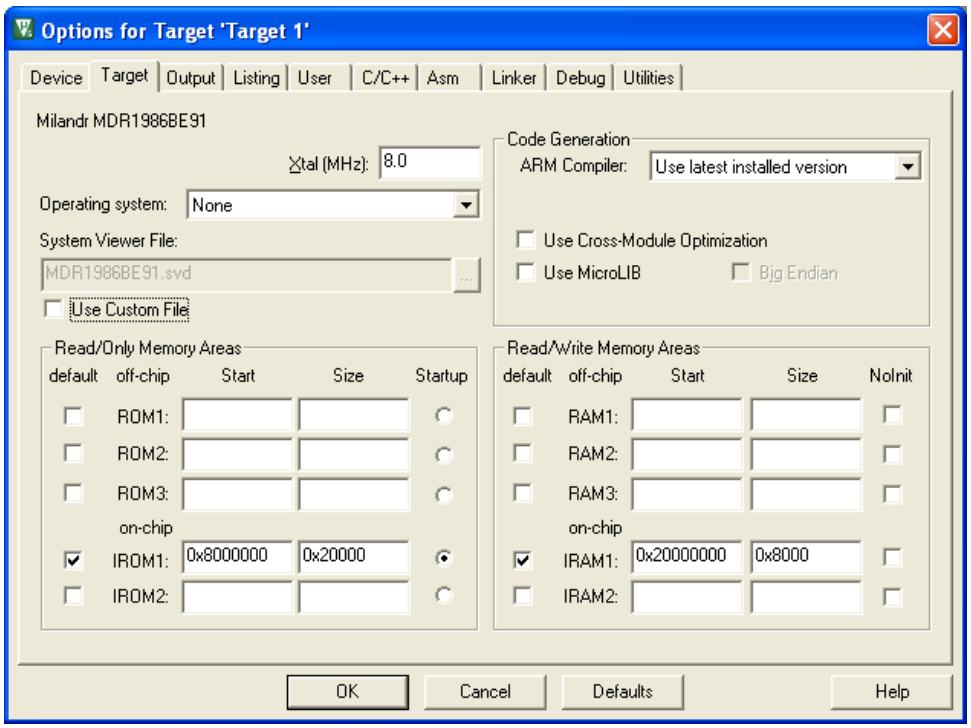


Рисунок 1.16 Вкладка **Target**

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” ввести значения “**-MPU**”.

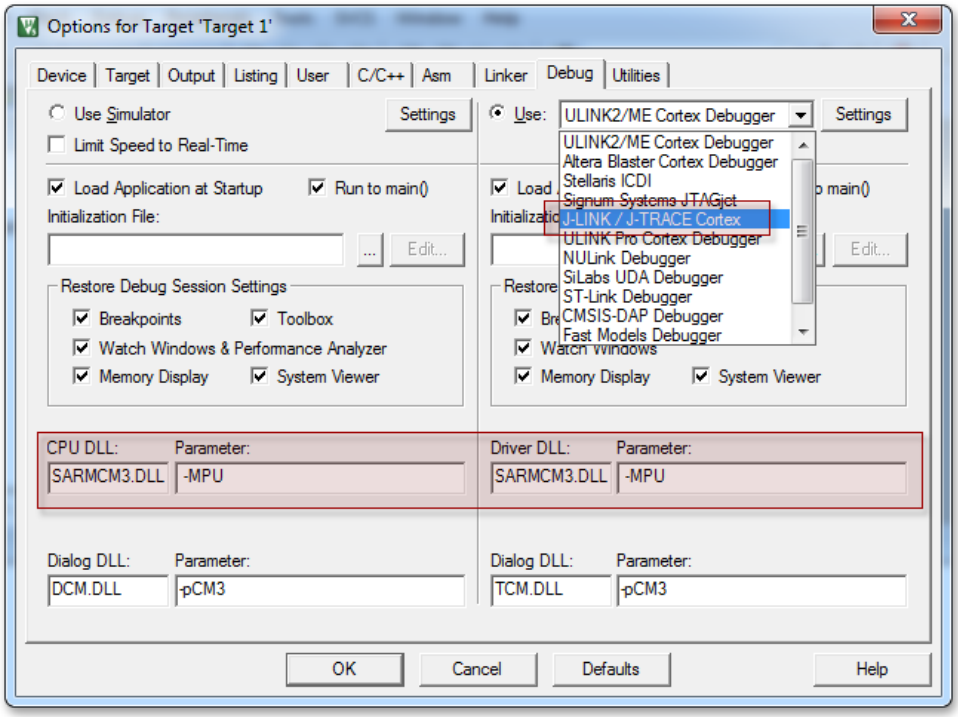


Рисунок 1.17 Вкладка **Debug**

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**.

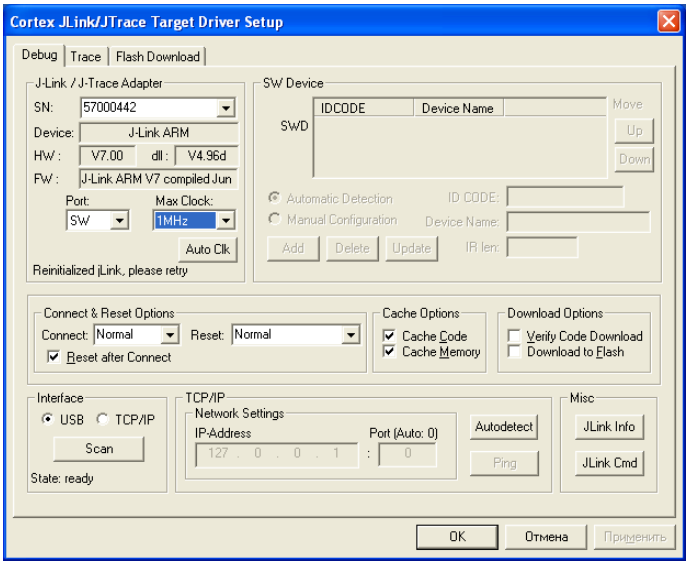


Рисунок 1.18 Окно после нажатия кнопки **Settings**

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.

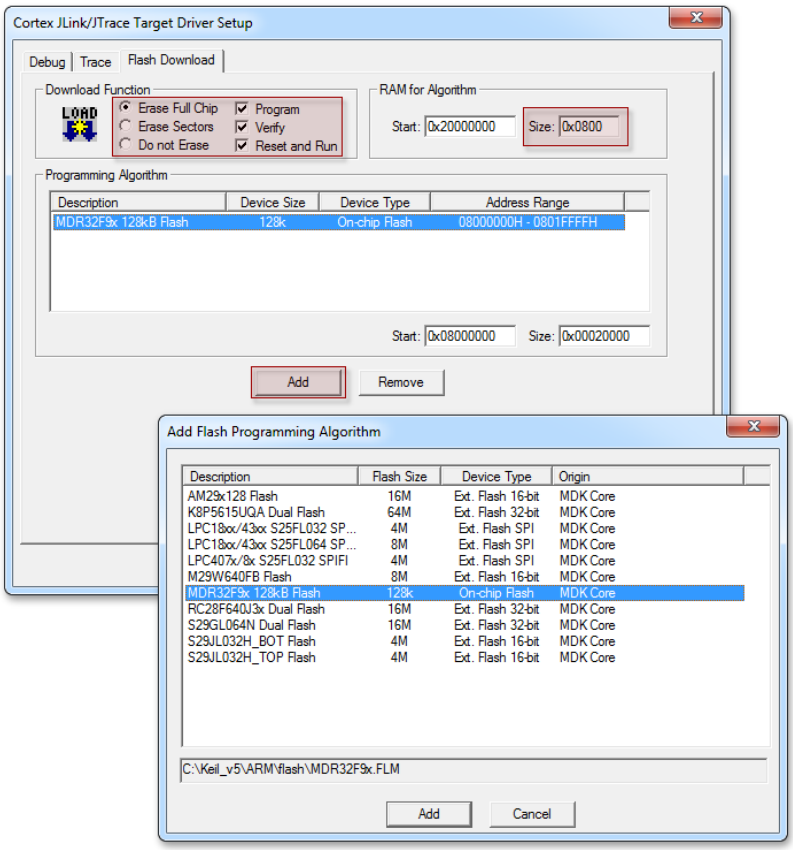


Рисунок 1.19 Вкладка **Flash Download** и окно **Programming Algorithm**

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

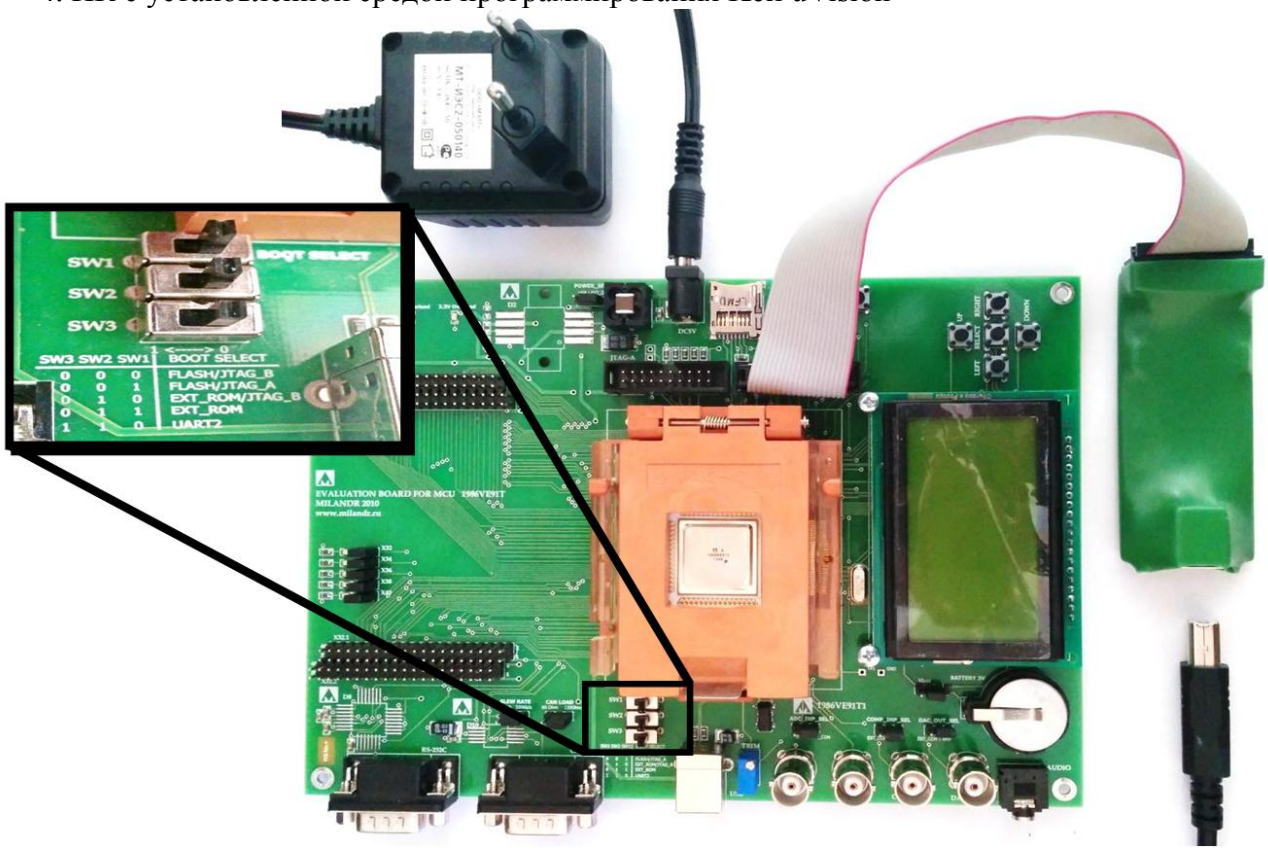


Рисунок 2.1 – Подключение платы 1986VE91T (крупно выделены переключатели выбора режима загрузки)

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).

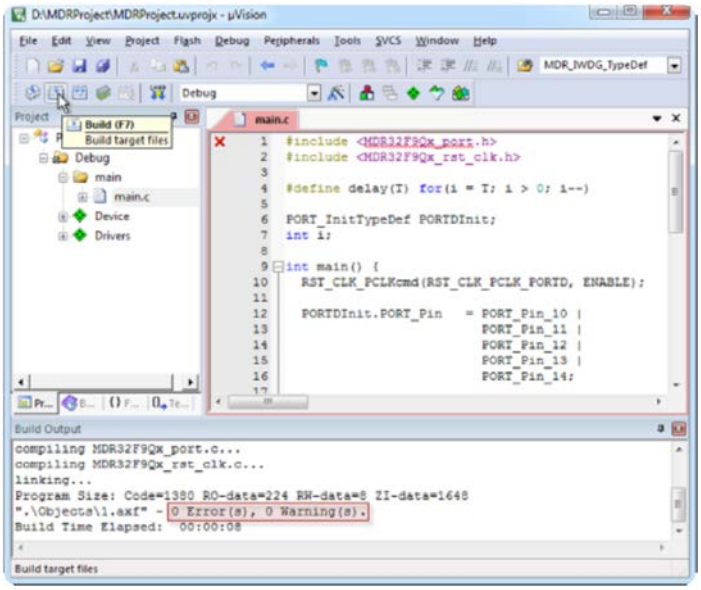


Рисунок 2.2 – Компиляция проекта

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**.

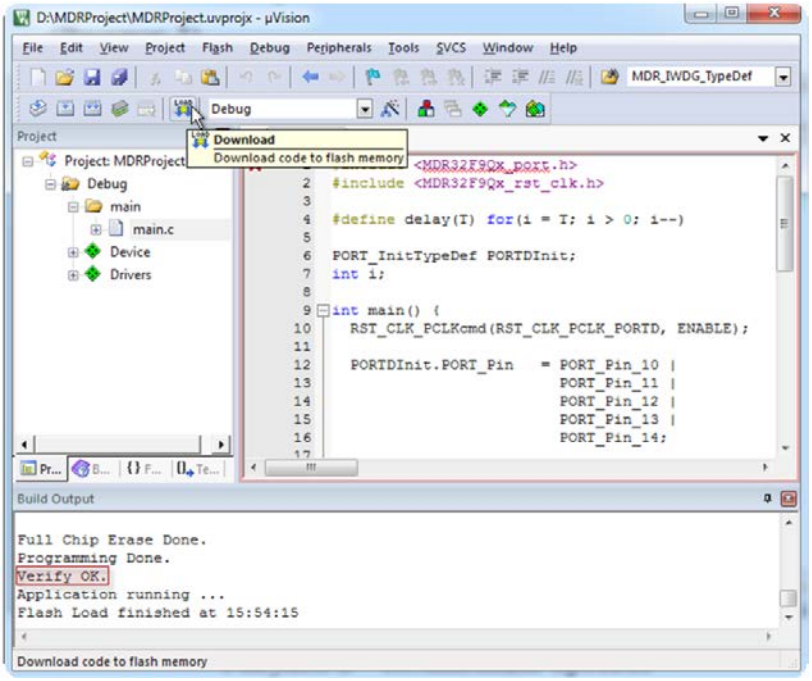


Рисунок 2.3 - Загрузка микропрограммы в устройство

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 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.

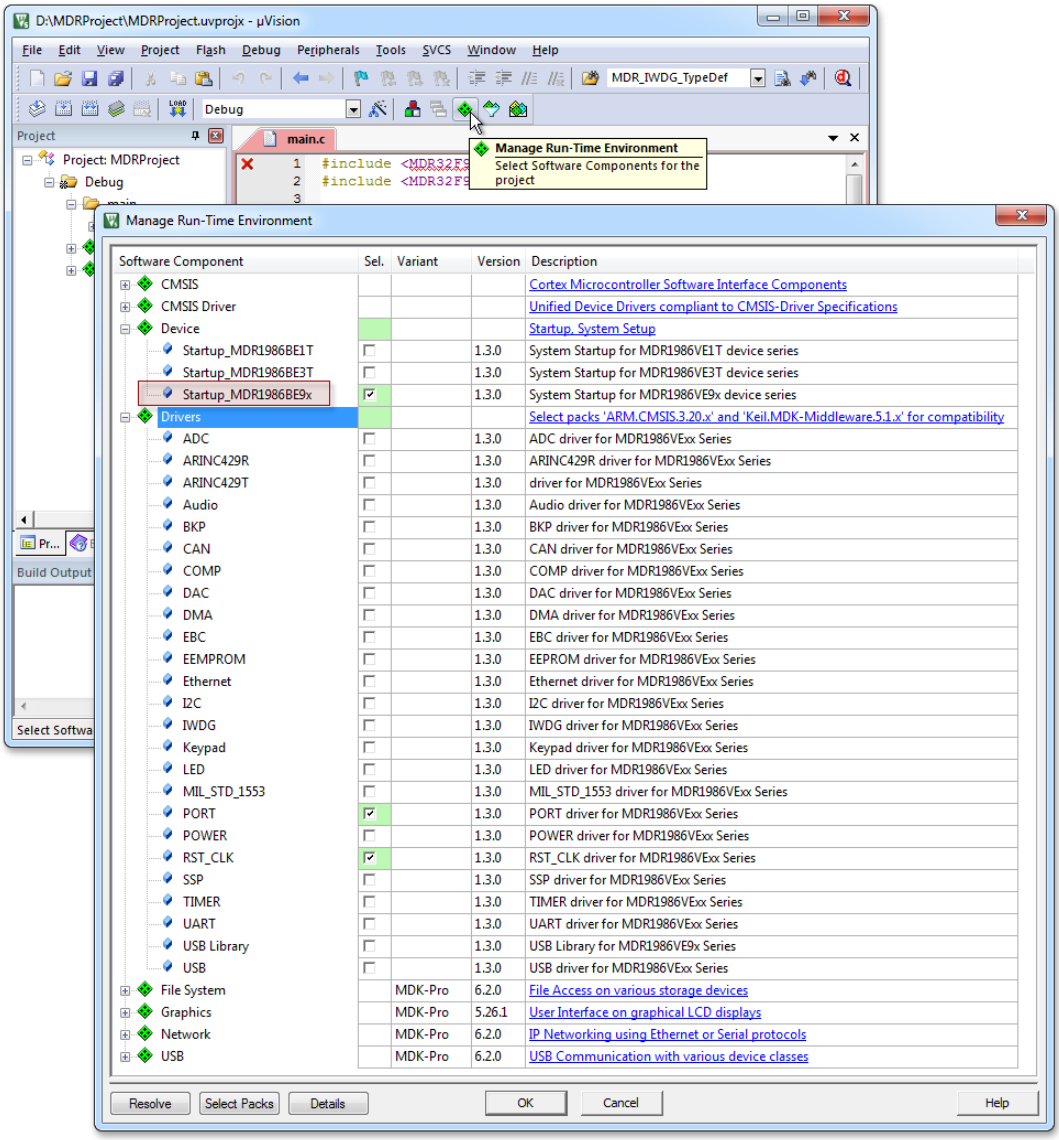


Рисунок 2.5 – Менеджер библиотек проекта

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.

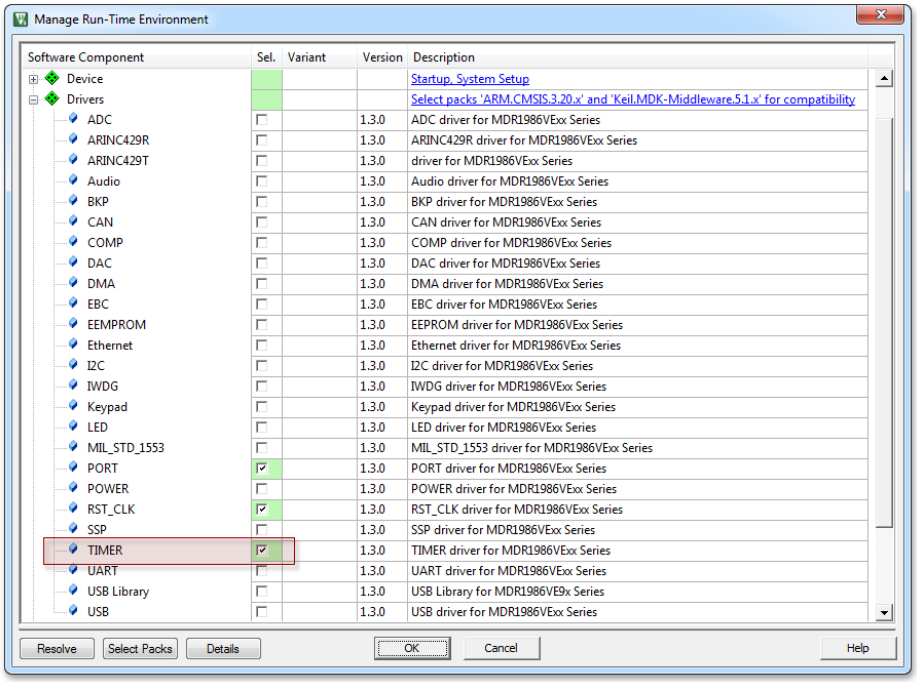


Рисунок 3.1 – Библиотеки проекта для работы с таймерами

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.

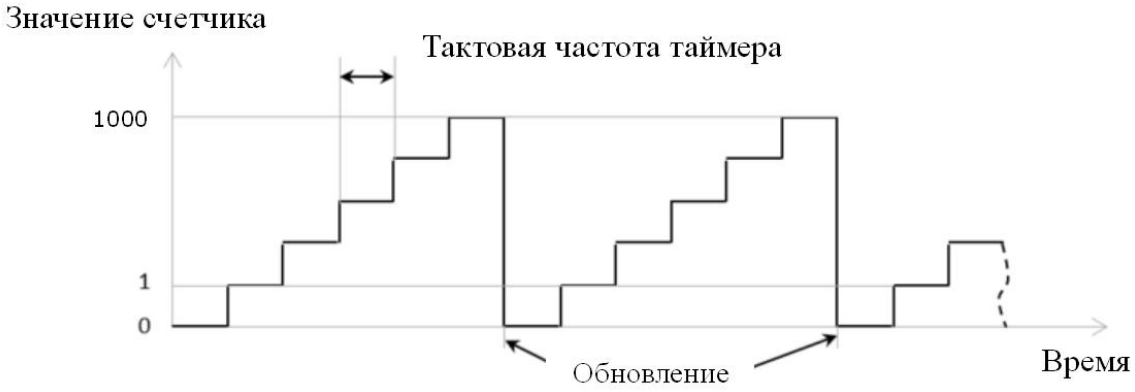


Рисунок 3.2 – Иллюстрация работы таймера

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 как стандартные функции захвата и сравнения, так и ряд специальных функций. Таймеры с 4 каналами 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.

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).

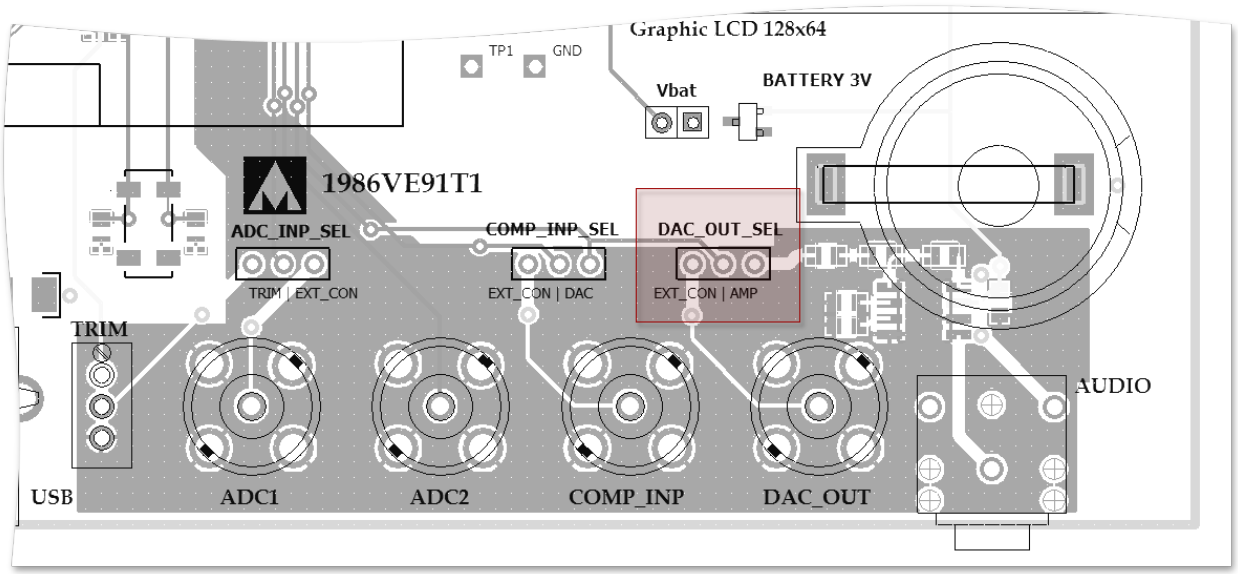


Рисунок 4.1 - Перемычка DAC_OUT_SEL

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.

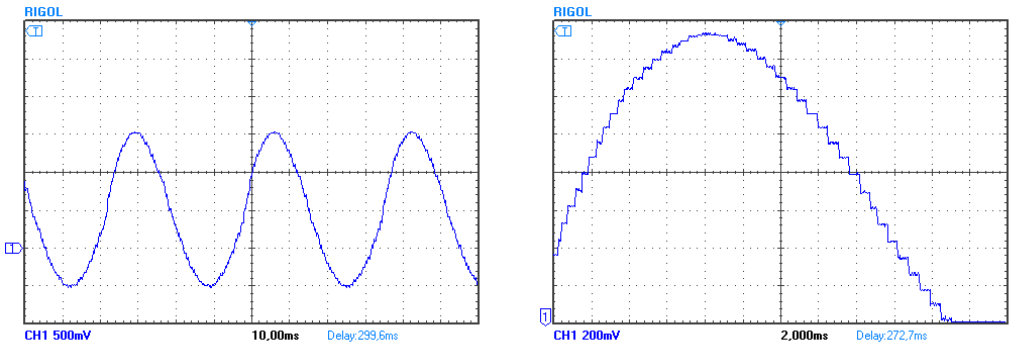


Рисунок 4.2 – Осциллограммы работы алгоритма

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).

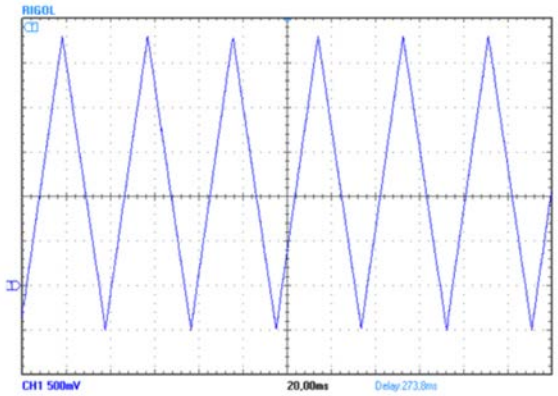


Рисунок 4.3 – Симметричная пилообразная функция

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.

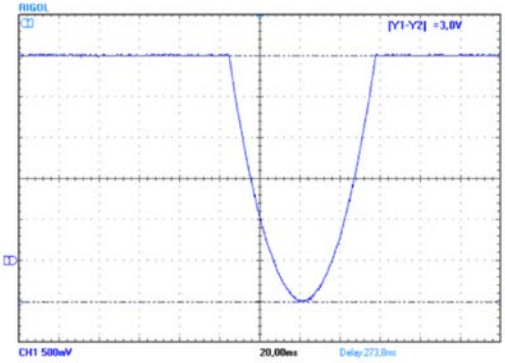


Рисунок 4.4 – Парабола

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).

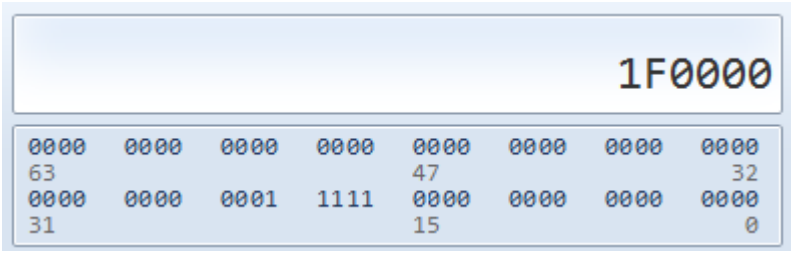


Рисунок 5.2 – Битовая маска для вычисления номера канала

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).

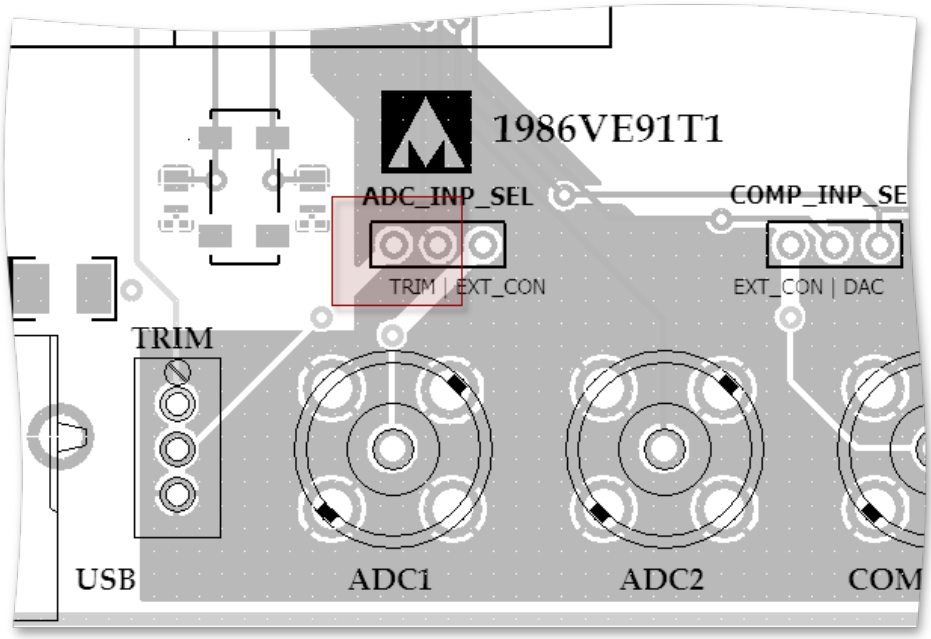


Рисунок 5.3 – Блок аналогового входа

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)

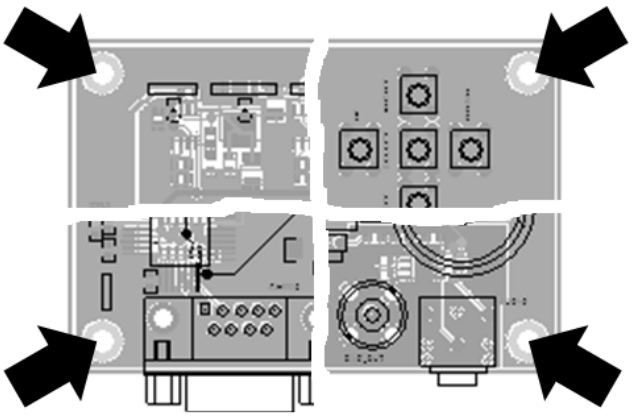


Рисунок 5.4 – Крепежные отверстия

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.

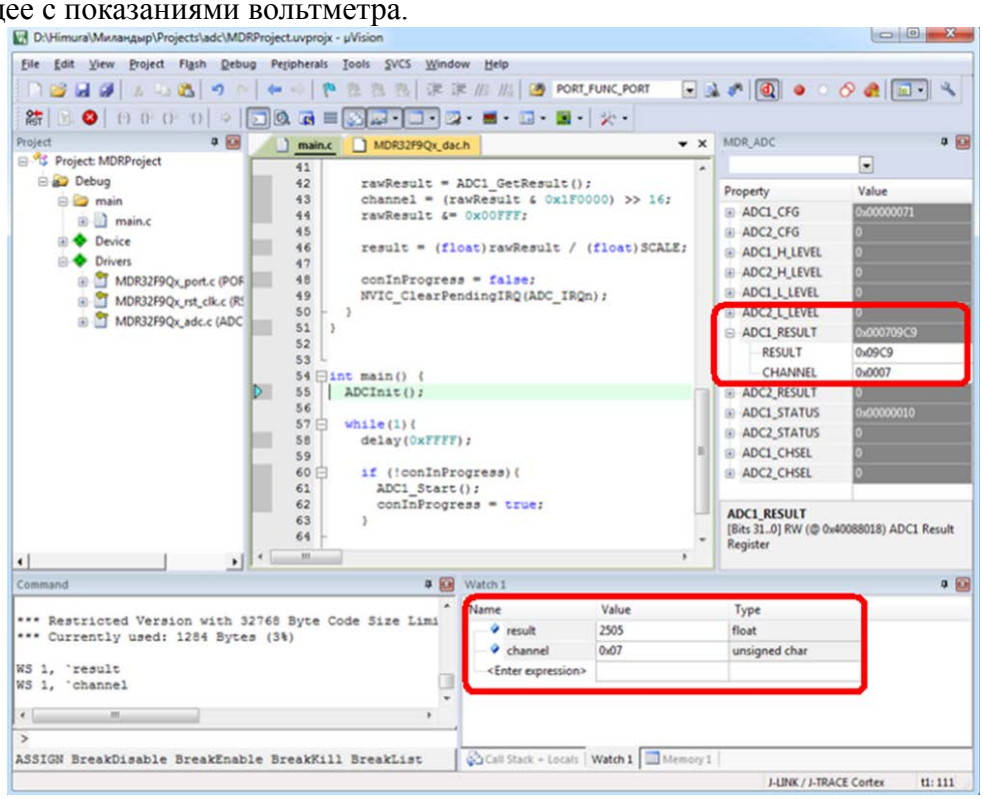


Рисунок 5.5 – Окно Keil uVision в процессе отладки

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, 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.

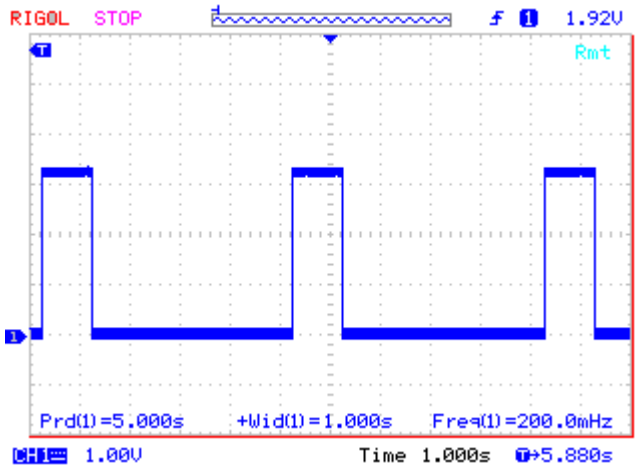


Рисунок 6.1 осциллограмма напряжения на светодиоде.

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.

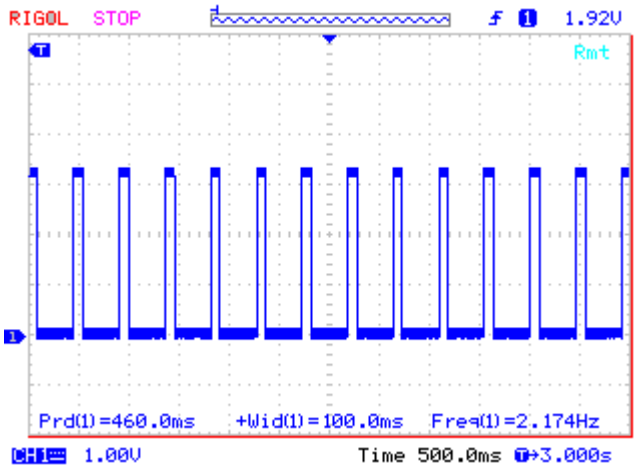


Рисунок 6.2. Осциллограмма напряжения после изменения тактовой частоты

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

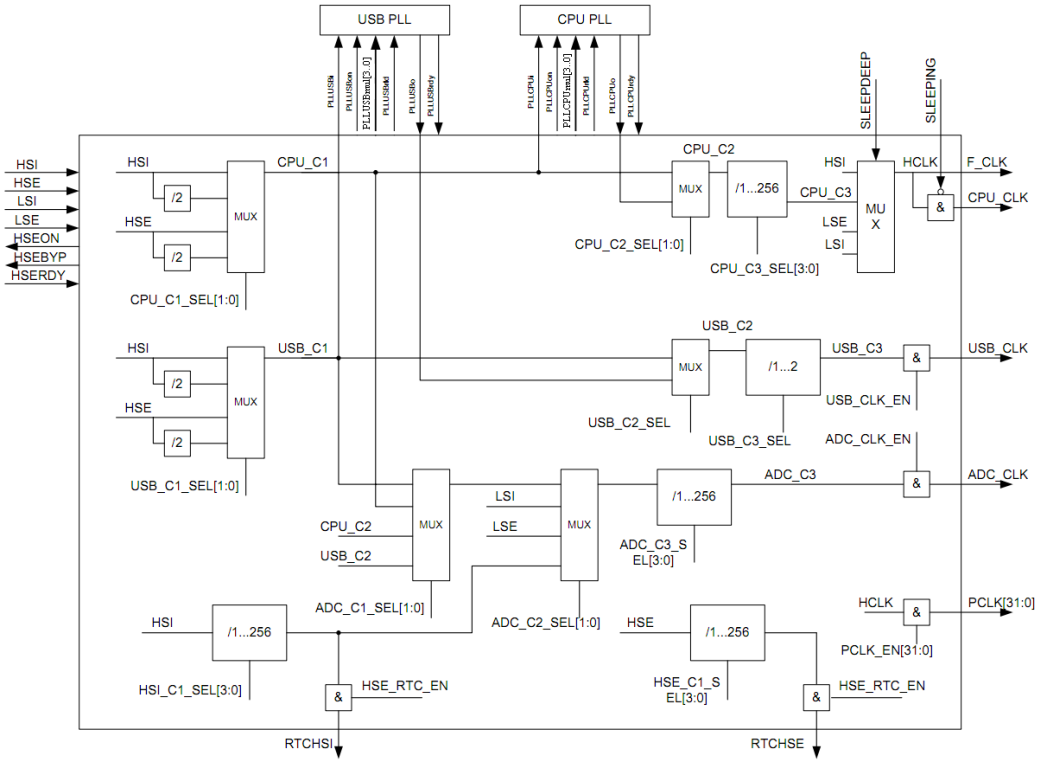


Рис 6.3 Структурная схема тактирования микроконтроллера

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 9);

  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

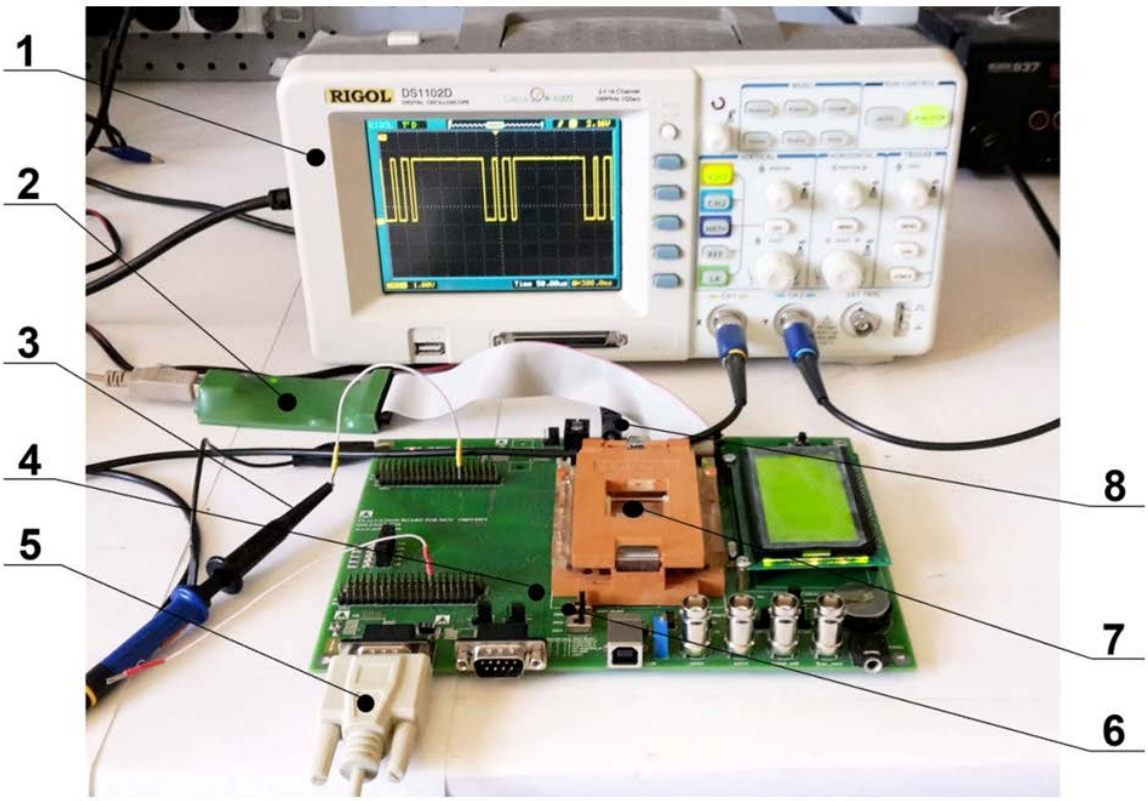


Рисунок 7.1 Внешний вид лабораторной установки: 1- осциллограф, 2-программтатор TP-LINK (J-LINK), 3-щуп осциллографа, 4-демонстрационная плата на базе 1986BE91T, 5-нуль модемный кабель для подключения по интерфейсу RS-232C, 6-микропереключатели для выбора режима загрузки, 7-микроконтроллер 1986BE91T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

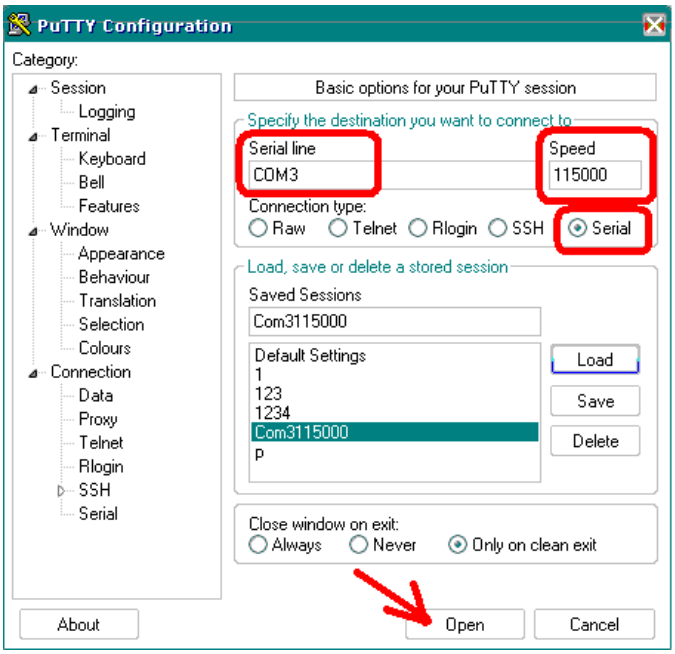


Рисунок 7.2. Окно программы Putty

10. В появившемся окне рис 7.3 ввести произвольный текст.

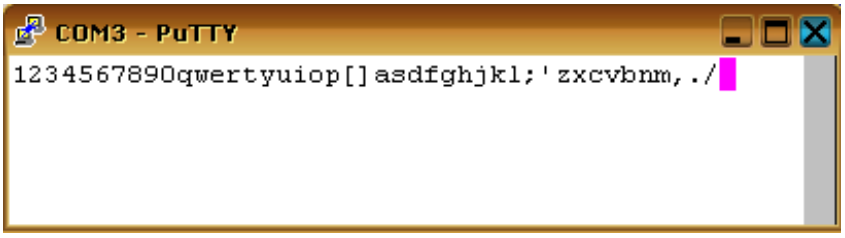


Рисунок 7.3. Окно программы Putty

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.

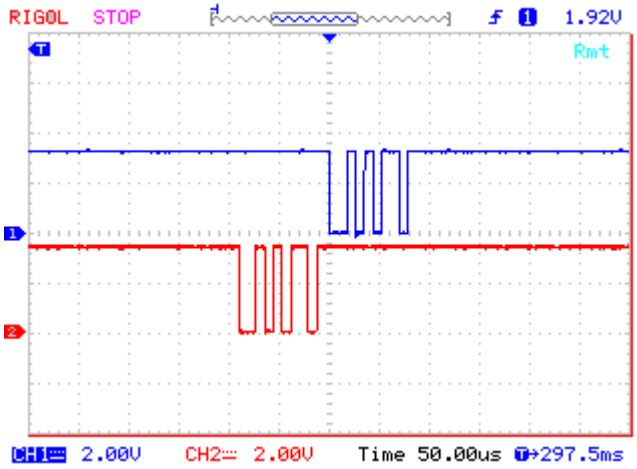


Рисунок 7.4 Прием байта в микроконтроллер (внизу), обратная передача данных на ЭВМ (вверху).

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 7
        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]

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**

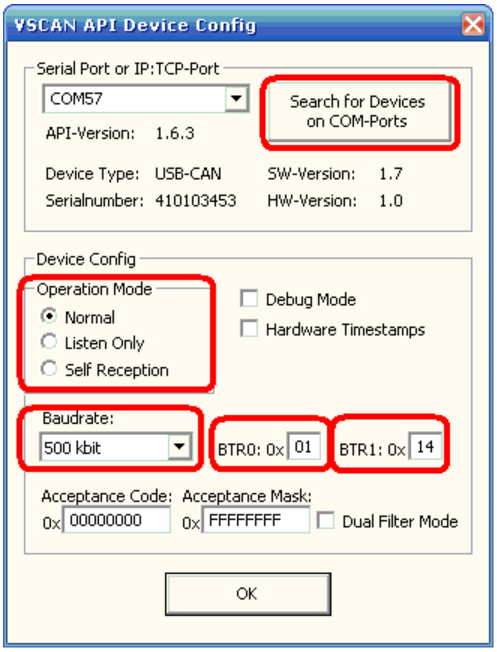


Рисунок 8.2. Настройки программы BUSMASTER

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.

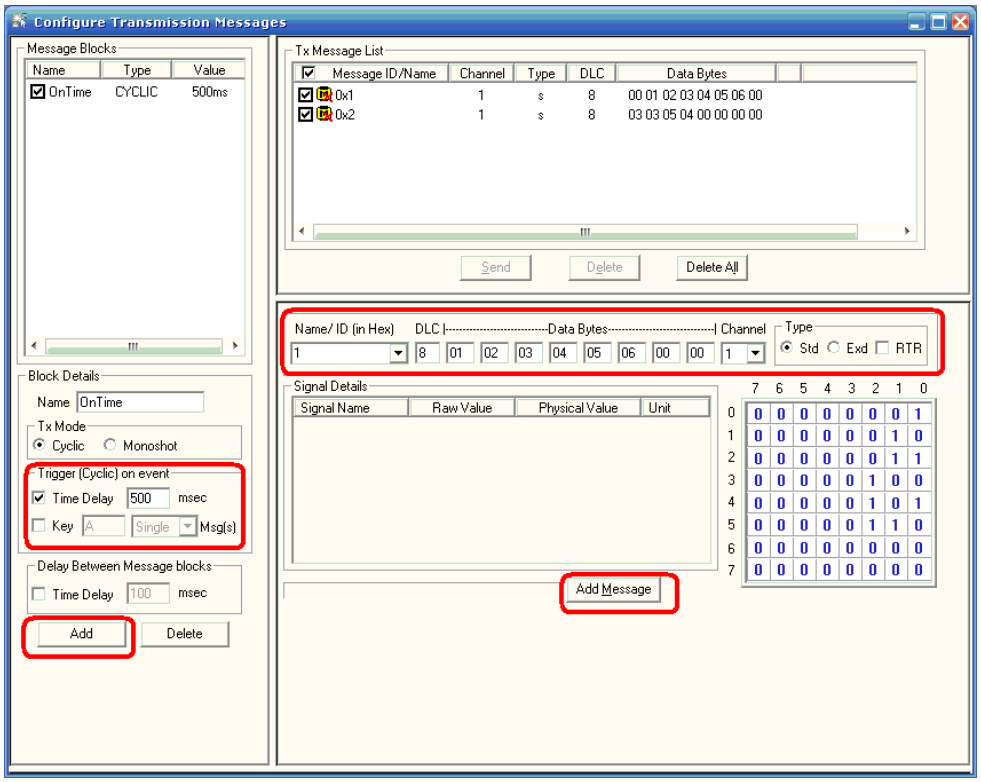


Рисунок 8.3. Настройки передаваемых сообщений программы BUSMASTER

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

Таблица 8.1

Номер контакта разъема	Имя контакта разъема	Функциональное назначение	Номер порта микроконтрол-лера	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

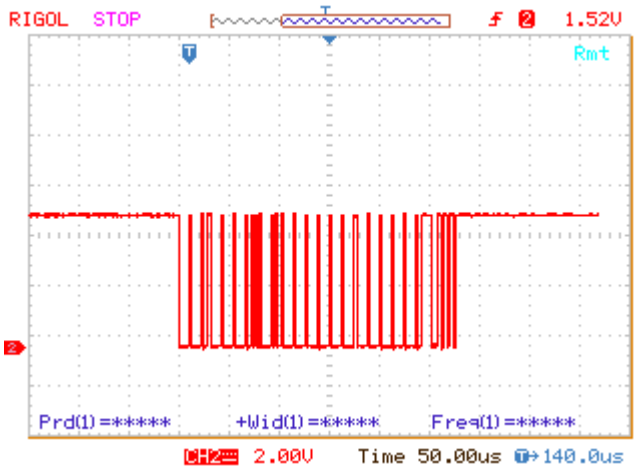
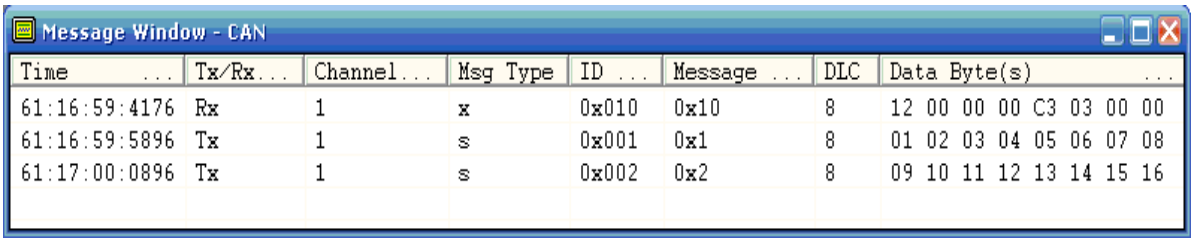


Рисунок 8.4 Осциллограмма передачи по CAN интерфейсу (X32.2 - 9).

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)**

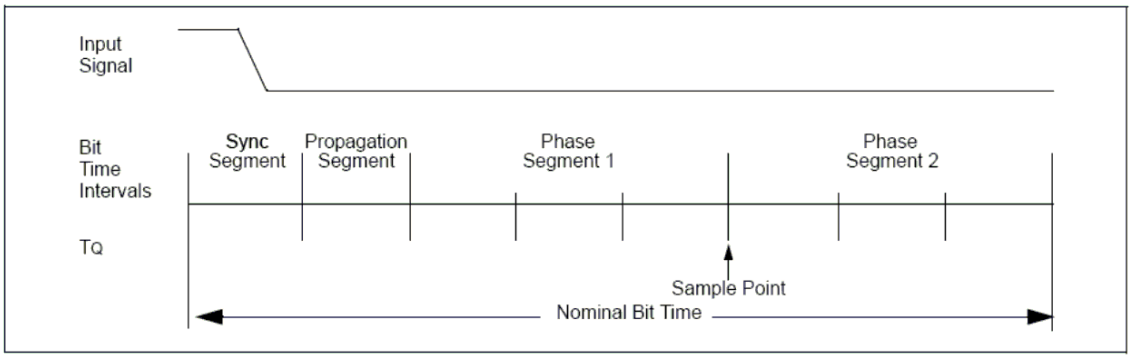


Рисунок 8.6. Передача одного бита информации по CAN

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);
  }
}
```

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