Минобрнауки России

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Оренбургский государственный университет»**

Кафедра промышленной электроники и информационно-измерительной техники

**Методические указания**

для обучающихся по освоению дисциплины

*« Б1.Д.В.6 Отладочные средства микропроцессорных систем»*

Уровень высшего образования

БАКАЛАВРИАТ

Направление подготовки

*11.03.04 Электроника и наноэлектроника*

(код и наименование направления подготовки)

*Промышленная электроника*

(наименование направленности (профиля) образовательной программы)

Квалификация

*Бакалавр*

Форма обучения

*Очная*

Год набора 2023

Составитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Хлуденев

Методические указания рассмотрены и одобрены на заседании кафедры промышленной электроники и информационно-измерительной техники

Заведующий кафедрой ПЭиИИТ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.В. Худорожков

Методические указания являются приложением к рабочей программе дисциплины «Отладочные средства микропроцессорных систем», зарегистрированной в ЦИТ под учетным номером \_\_\_\_\_\_\_\_\_.

**1 Общие положения**

Дисциплина относится к обязательным дисциплинам (модулям) вариативной части блока Д «Дисциплины (модули)». Изучается в 8 семестре.

В результате изучения дисциплины студент должен:

**Знать:** архитектуру и принципы функционирования микроконтроллеров.

**Уметь:** программировать цифровые электронные устройства на платформе микроконтроллеров.

**Владеть:** средствами и навыками разработки и отладки программного обеспечения для микроконтроллеров.

Достижение планируемых результатов обучения обеспечивается добросовестным отношением обучающегося к изучению теоретического материала дисциплины с использованием материала лекций и рекомендованной литературы, ознакомлением с рекомендованными периодическими изданиями, своевременным выполнением лабораторных работ и заданий на практических занятиях, а также выполнением и защитой курсового проекта.

**2 Общие методические рекомендации**

Контроль достижения планируемых результатов обучения обеспечивается использованием оценочных средств, представленных в «Фонде оценочных средств …» (приложение к рабочей программе дисциплины), в учебном процессе во время занятий по расписанию, а также во время промежуточной аттестации.

**2.1 Теоретический материал, изучаемый при освоении дисциплины**

**Раздел 1 Средства разработки и автономной отладки программ**

Типовой процесс разработки программного обеспечения МПУ. Языки программирования МПУ. Основные средства языка Ассемблера, технология программирования. Программирование на языках высокого уровня. Основные средства языка ANSI C. Кросс системы разработки программного обеспечения. Редакторы текста, трансляторы, редакторы связей, библиотекари. Основные режимы трансляции программ.

Основные источники ошибок программ. Основные методы отладки программ. Методы и приемы тестирования программных средств. Симуляторы-отладчики. Программно-логические модели МП и МК, периферийных модулей МПУ. Стимулы.

**Раздел 2 Средства комплексной отладки**

Комплексирование аппаратных и программных средств МПУ. Цели и задачи комплексной отладки. Загрузчики и программаторы. Внутрисхемное программирование. Отладочные мониторы. Эмуляторы ПЗУ. Схемные эмуляторы и симуляторы. Встроенные средства комплексной отладки микроконтроллеров. Интерфейсы программирования и отладки.

**Вопросы, изучаемые самостоятельно с использованием рекомендуемой литературы:**

**Раздел 1 Средства разработки и автономной отладки программ**

Директивы языка Ассемблера PIC16. Основные средства языка ANSI C. Основные средства среды MPLAB IDE, Keil uVision.

**Раздел 2 Средства комплексной отладки**

Схемные эмуляторы MPLAB ICDx, ST-Link/v2. Встроенные средства отладки PIC16, ARM32 Cortex M3.

**Контроль усвоения** изученного теоретического материала осуществляется методом опроса на лабораторных работах, практических занятиях, а также методом **тестирования**. Объем (тематику) проверяемого при тестировании материала и дату проведения тестирования определяет ведущий преподаватель.

**Рекомендуемая литература**:

1. Яценков, В. С. Микроконтроллеры MicroCHIP [Текст] : практическое руководство / В. С. Яценков.- 2-е изд., испр. и доп. - Москва : Горячая линия-Телеком, 2008. - 280 с. : ил. - (Современная электроника). - На обл.: Схемы, примеры программ, описания, ресурсы.
2. Хлуденев, А. В. Средства разработки и отладки программ для микроконтроллеров [Электронный ресурс] : учебное пособие для обучающихся по образовательным программам высшего образования по направлениям подготовки 11.03.03 Конструирование и технология электронных средств и 11.03.04 Электроника и наноэлектроника / А. В. Хлуденев; М-во науки и высш. образования Рос. Федерации, Федер. гос. бюджет. образоват. учреждение высш. образования "Оренбург. гос. ун-т". - Оренбург : ОГУ. - 2019. - 106 с. – Режим доступа: <http://artlib.osu.ru/web/books/metod_all/92671_20190322.pdf>
3. Тавернье, К. РIС-микроконтроллеры [Текст] : практика применения: пер. с фр. / К. Тавернье. - М. : ДМК Пресс, 2002. - 272 c. - (Справочник) - ISBN 5-94074-115-0. - ISBN 2-10-002866-9.
4. Предко, М. Руководство по микроконтроллерам [Комплект] : в 2 т. / М. Предко. - М. : Постмаркет, 2001 - ISBN 5-901095-07-3. Т. 1. - 2001. - 416 с.: ил. Т. 2. - 2001. - 488 с.: ил. + 1 электрон. опт. диск (CD-ROM).

INTERNET - ISBN 5-93517-203-8.

**2.2 Практикум**

С целью привития умений использовать средства отладки микропроцессорных систем предусмотрено выполнение студентами лабораторных работ и заданий на практических занятиях. Тематика лабораторных работ и практических занятий приведена в рабочей программе дисциплины.

Отчет о выполненной лабораторной работе каждый обучающийся защищает индивидуально. Во время защиты обучающийся должен быть готовым пояснить методику проведения исследований, показать умение анализировать результаты, полученные в ходе проведения исследований, быть готовым ответить на вопросы преподавателя по теме проводимых исследований.

Практические задания 1 - 4 студенты выполняют под руководством преподавателя с использованием методических рекомендаций

Хлуденев, А. В. Разработка и кодирование алгоритмов для PIC-микро [Электронный ре-сурс] : методические указания для обучающихся по образовательной программе высшего образо-вания по направлению подготовки 11.03.04 Электроника и наноэлектроника / А. В. Хлуденев; М-во образования и науки Рос. Федерации, Федер. гос. бюджет. образоват. учреждение высш. обра-зования "Оренбург. гос. ун-т", Каф. пром. электроники и информ.-измер. техники. - Оренбург : ОГУ. - 2018. - 28 с.

Методические рекомендации для практических занятий 5 – 7 приведены в приложении А.

Исходные тексты разработанных программ являются исходными данными для выполнения соответствующих лабораторных работ.

Лабораторные работы 1 – 4 студенты выполняют под руководством преподавателя с использованием методических рекомендаций

Хлуденев, А. В. Отладочные средства микропроцессорных систем [Электронный ресурс] : методические указания для обучающихся по образовательной программе высшего образования по направлению подготовки 11.03.04 Электроника и наноэлектроника / А. В. Хлуденев; М-во образования и науки Рос. Федерации, Федер. гос. бюджет. образоват. учреждение высш. образования "Оренбург. гос. ун-т", Каф. пром. электроники и информ.-измер. техники. - Оренбург : ОГУ. - 2018. - 23 с.

Методические рекомендации для лабораторных занятий 5 – 7 приведены в приложении Б.

Каждая лабораторная работа предусматривает выполнение серии экспериментов, в ходе которых выполняется тестирование программы. По результатам тестирования принимаются решения о корректности программного кода или о необходимости локализации и исправления ошибок. Лабораторные работы выполняются на учебно-отладочных стендах. Перед выполнением каждой лабораторной работы необходимо четко представлять цель и план тестирования программы. По каждой лабораторной работе необходимо формировать отчет, содержание которого определяется методическими указаниями. Отчет оформляется в соответствии с требованиями СТО 02069024.101–2015.

Во время защиты отчета по лабораторной работе необходимо пояснить методику тестирования, показать умение анализировать полученные результаты, быть готовым ответить на контрольные вопросы по теме работы.

**2.3 Курсовое проектирование**

Целью выполнения курсового проекта в восьмом семестре является закрепление практических навыков самостоятельного решения инженерных задач в области программной реализации алгоритмов управления и обработки информации, развитие творческих способностей и овладение инструментальными средствами разработки микропроцессорных устройств.

Темой проекта является разработка микропроцессорного устройства, реализующего функции обработки информации и управления (по вариантам). Предполагается, что устройство выполнено на платформе микроконтроллера семейства PIC16 или Cortex M3, и в качестве его прототипа используется учебно-отладочный стенд. Требуется разработать и отладить устройство на платформе микроконтроллера. Задание должно быть выполнено с использованием инструментальных средств разработки. Правильность выполнения задания контролируется выполнением тестирования и отладки программы. В ходе тестирования устанавливается соответствие реализуемых функций требованиям задания, локализуются и исправляются допущенные ошибки и неточности.

Над курсовым проектом студенты работают самостоятельно – в свободное от занятий время. Возникающие в процессе работы вопросы студенты могут решать в часы консультаций с руководителем проекта. При выполнении курсового проекта руководствоваться методическими указаниями:

Хлуденев, А. В. Разработка и отладка устройств на микроконтроллерах [Текст] : методические указания / А. В. Хлуденев; М-во образования и науки Рос. Федерации, Федер. агентство по образованию, Гос. образоват. учреждение высш. проф. образования "Оренбург. гос. ун-т", Каф. пром. электроники и информ.-измер. техники. - Оренбург : ГОУ ОГУ, 2008. - 47 с.

Срок сдачи преподавателю законченного курсового проекта на проверку – за 1 – 3 дня до защиты. Защита курсового проекта – 18 учебная неделя (согласно расписанию).

**Приложение А**

**Методические рекомендации к практическим занятиям**

**5 Работа с портами ввода-вывода K1986ВЕ9х**

Нажатие с последующим отпусканием кнопки RC1 должно вызывать модификацию содержимого ячейки Cnt. Содержимое ячейки Cnt должно отображаться на светоизлучающих диодах, подключенных к линиям порта PB3 - PB0. Варианты заданий приведены в таблице 1.1.

Таблица 1.1

|  |  |  |
| --- | --- | --- |
| Вариант | Кнопка | Задание |
| 1 | Sel | Формирователь последовательности чисел Фибоначчи 0, 1,…,233 |
| 2 | Down | Вычитающий двоичный счетчик по модулю 16 |
| 3 | Up | Суммирующий восьмиразрядный счетчик в унитарном коде |
| 4 | Down | Вычитающий восьмиразрядный счетчик в унитарном коде |
| 5 | Left | Суммирующий восьмиразрядный счетчик в коде Джонсона |
| 6 | Right | Вычитающий восьмиразрядный счетчик в коде Джонсона |

На отладочной плате LDM-K1986ВЕ92QI-H размещены кнопки клавишного джойстика «UP», «DOWN», «LEFT», «RIGHT» и «SEL», подключенные к выводам микроконтроллера. Схема подключения клавиш джойстика к линиям портов приведена на рисунке 1.1. Один из контактов каждой кнопки подсоединен к «земле», а второй к линии порта и подтягивающему резистору. Таким образом, в отжатом состоянии кнопки на линии порта действует высокий потенциал, а в нажатом состоянии – низкий.

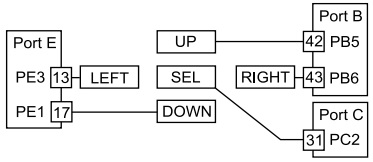


Рисунок 1.1

К четырем линиям порта PB3 – PB0 подключены светоизлучающие диоды. В работе рассматривается задача модификации значения переменной cell и вывода ее четырех младших разрядов на линии порта PB3 – PB0. Вид модификации определяется активной кнопкой:

«UP» - инкремент;

«DOWN» - декремент;

«LEFT» - сдвиг влево;

«RIGHT» - сдвиг вправо;

«SEL» - поразрядное НЕ.

Процессор имеет 24-х разрядный системный таймер SysTick, выполняющий обратный счет от загруженного в него значения до нуля; перезагрузка значения в регистр LOAD происходит по следующему фронту синхросигнала, затем счёт продолжается. Когда процессор остановлен для отладки, таймер не декрементируется.

Необходимо выбрать источник тактового сигнала, разрешить прерывания и включить системный таймер. Прерывания будут нужны для создания функции задержки. В регистр LOAD необходимо загрузить значение задержки. Настроим таймер на прерывание раз в одну миллисекунду. Таймер настроим на тактирование от HCLK – это частота тактирования ядра микроконтроллера. Она равна 8 МГц или 8000000 тактов в секунду. Тогда количество тактов в одной миллисекунде 8000000 / 1000. Так как один такт используется для загрузки таймера, то это значение необходимо уменьшить на единицу.

#define CLKSOURCE (1<<2) //Источник синхросигнала: 0 - LSI, 1 - HCLK

#define TCKINT (1<<1) //Разрешает запрос на прерывание

#define ENABLE (1<<0) //Разрешает работу таймера

void Init\_SysTick (void) //Прерывание раз в миллисекунду.

{

SysTick->LOAD |= (8000000/1000)-1;

SysTick->CTRL |= CLKSOURCE|TCKINT|ENABLE;

}

В микроконтроллерах с архитектурой ARM32 используется контроллер вложенных векторных прерываний NVIC (Nested Vector Interrupt Controller). NVIC представляет собой специальную подсистему, управляющую аппаратными прерываниями. NVIC позволяет задать каждому прерыванию определенный приоритет, в соответствии с которым будет определяться порядок обработки запросов на прерывания, поступивших одновременно. Сначала обрабатывается запрос на прерывание с большим приоритетом. Остальные запросы ставятся в очередь типа FIFO (первый зашел – первый вышел). Создавая обработчик прерывания, нужно стараться сделать его как можно короче и проще, чтобы излишне не загружать процессор.

При появлении запроса прерывания от системного таймера, управление передается на вектор прерывания SysTick\_Handler. Для обработки прерывания используется функция SysTick\_Handler, выполняющая обратный отсчет миллисекунд:

volatile uint32\_t Delay\_dec = 0;

void SysTick\_Handler (void)

{

if (Delay\_dec) Delay\_dec--;

}

Функция задержки, использующая данное прерывание, принимает длительность задержки (в миллисекундах) Delay\_ms\_Data, копирует ее в переменную Delay\_dec. После того, как задержка выдержана, функция завершает свою работу.

void Delay\_ms (uint32\_t Delay\_ms\_Data)

{

Delay\_dec = Delay\_ms\_Data;

while (Delay\_dec) {};

}

**Конфигурирование линий ввода-вывода**

Для того чтобы начать использовать линии ввода-вывода, нужно предварительно сконфигурировать их соответствующим образом. На самом низком уровне работа с портами ввода-вывода осуществляется с помощью специальных регистров микроконтроллера. Эти регистры доступны как ячейки памяти, расположенные по определенным адресам. Для облегчения труда программиста многие разработчики микроконтроллеров предоставляют специальные библиотеки функций для работы с периферийными устройствами. Для микроконтроллера K1986ВЕ92QI, подходит библиотека «MDR32F9Qx Standard Peripherals Library. В этой библиотеке есть функции для работы со всеми устройствами в составе микроконтроллера. При этом программисту не надо напрямую обращаться к регистрам, достаточно вызвать подходящую функцию.

Для работы с портами ввода-вывода к проекту должен быть подключен библиотечный модуль MDR32F9Qx\_port.c. В модуле MDR32F9Qx\_port.c есть все необходимое для работы с портами. Для удобства настройки портов ввода-вывода рекомендуется использовать специальную структуру типа PORT\_InitTypeDef. Тип PORT\_InitTypeDef описан в заголовке MDR32F9Qx\_port.h:

typedef struct

{ uint16\_t PORT\_Pin;

PORT\_OE\_TypeDef PORT\_OE;

PORT\_PULL\_UP\_TypeDef PORT\_PULL\_UP; PORT\_PULL\_DOWN\_TypeDef PORT\_PULL\_DOWN; PORT\_PD\_SHM\_TypeDef PORT\_PD\_SHM;

PORT\_PD\_TypeDef PORT\_PD;

PORT\_GFEN\_TypeDef PORT\_GFEN;

PORT\_FUNC\_TypeDef PORT\_FUNC;

PORT\_SPEED\_TypeDef PORT\_SPEED;

PORT\_MODE\_TypeDef PORT\_MODE;

} PORT\_InitTypeDef;

В поле PORT\_Pin типа uint16\_t указывают, какие линии подлежат конфигурированию. Каждый бит в этом поле отвечает за отдельную линию ввода-вывода. Линии нумеруются, начиная с нуля. Так, если указать единицу в 4-м разряде, т.е. занести в поле значение 0x0010, это будет означать, что мы конфигурируем линию 4. Можно указать единицы сразу в нескольких разрядах, например, во 2-м, 3-м и 5-м. Тогда все настройки будут, соответственно, относится к линиям 2, 3 и 5. Для этого в поле PORT\_Pin занесем значение 0x002С.

Для работы необходима функция цифрового входа или выхода. Для задания направления используется поле PORT\_OE типа PORT\_OE\_TypeDef. Если требуется сделать вывод цифровым входом, то:

PortInitStructure.PORT\_OE = PORT\_OE\_IN;

а если цифровым выходом, то:

PortInitStructure.PORT\_OE = PORT\_OE\_OUT;

В поле PORT\_SPEED типа PORT\_SPEED\_TypeDef указывают скорость работы линии вывода. Возможны следующие значения:

− PORT\_OUTPUT\_OFF – выход выключен;

− PORT\_SPEED\_SLOW – низкая скорость (фронт порядка 100 нс, частота до 5 МГц);

− PORT\_SPEED\_FAST – высокая скорость (фронт порядка 20 нс, частота до 25 МГц);

− PORT\_SPEED\_MAXFAST – предельно высокая скорость (фронт порядка 10 нс, частота до 50 МГц).

Подпрограмма-функция InitPortLed для начальной инициализации линий

PB3 - PB0 на вывод

#define LEDS\_MASK (0xF)

#define LED0\_OFS (0)

void InitPortLed(void) {

MDR\_PORTB->FUNC &= ~((0xFF << (LED0\_OFS << 1))); // Port

MDR\_PORTB->ANALOG |= LEDS\_MASK; // Digital

MDR\_PORTB->PWR |= (0xAA << (LED0\_OFS << 1)); // Slow

MDR\_PORTB->RXTX &= ~LEDS\_MASK;

MDR\_PORTB->OE |= LEDS\_MASK;

}

Подпрограмма-функция InitPortJoystick для начальной инициализации линий трех портов, которые используются для считывания информации с кнопок джойстика:

void InitPortJoystick(void) {

MDR\_PORTB->FUNC &= ~((0xF << (5 << 1))); // UP - PB5, RIGNHT - PB6

MDR\_PORTB->ANALOG |= 0x3 << 5; // Digital

MDR\_PORTC->FUNC &= ~((0x3 << (2 << 1))); // SEL - PC2

MDR\_PORTC->ANALOG |= 0x1 << 2; // Digital

MDR\_PORTE->FUNC &= ~((0x33 << (1 << 1))); //DOWN - PE1, LEFT - PE3

// (0b110011)

MDR\_PORTE->ANALOG |= 0x5 << 1; // Digital

}

Подпрограмма-функция GetKey возвращает унитарный код нажатой кнопки:

int GetKey(void)

{

int data;

data = 0;

if(~(MDR\_PORTC->RXTX) & (1<<2)) data |= 1; // SEL PC2

if(~(MDR\_PORTB->RXTX) & (1<<6)) data |= (1<<1); // RIGHT PB6

if(~(MDR\_PORTE->RXTX) & (1<<3)) data |= (1<<2); // LEFT PE3

if(~(MDR\_PORTB->RXTX) & (1<<5)) data |= (1<<3); // UP PB5

if(~(MDR\_PORTE->RXTX) & (1<<1)) data |= (1<<4); // DOWN PE1

return (data);

}

Основной модуль программы использует глобальные переменные

int key, keyh, keyh2, difkey;

char cell;

и имеет структуру, включающую операторы и вызов функций инициализации и бесконечный цикл. В бесконечном цикле переменной key возвращается унитарный код активной кнопки, по комбинации значений активного разряда 1, 0, 0 в текущем и двух предыдущих циклах, определяется и выполняется действие над переменной cell. Затем ее младшая тетрада выводится на линии порта PB3 – PB0. Задержка длительностью 30 мс позволяет программно подавить дребезг контактов кнопок. Исходный текст основного модуля:

int main()

{

Init\_SysTick(); //Инициализируем системный таймер.

RST\_CLK\_PCLKcmd(RST\_CLK\_PCLK\_PORTC, ENABLE);

//включить тактирование порта С

RST\_CLK\_PCLKcmd(RST\_CLK\_PCLK\_PORTB, ENABLE);

//включить тактирование порта B

RST\_CLK\_PCLKcmd(RST\_CLK\_PCLK\_PORTE, ENABLE);

//включить тактирование порта E

InitPortJoystick();

InitPortLed();

while(1)

{ //бесконечный цикл

key = GetKey();

difkey = key & ~(keyh | keyh2);

switch (difkey) {

case 1: cell = ~cell; break; // инвертировать

case 2: cell>>=1; break; // сдвиг вправо

case 4: cell<<=1; break; // сдвиг влево

case 8: cell++; break; // инкремент

case 16: cell--; break; // декремент

}

MDR\_PORTB->RXTX = (MDR\_PORTB->RXTX & ~LEDS\_MASK) | (cell & LEDS\_MASK);

keyh2 = keyh;

keyh = key;

Delay\_ms (30);

}

}

**Контрольные вопросы**

Поясните структуру программы на языке Си.

Чем различаются локальные и глобальные переменные?

Поясните назначение директив препроцессора.

Перечислите основные типы данных.

Чем отличаются главная функция программы и пользовательские функции.

Поясните организацию системы тактирования Cortex-M3.

Как может тактироваться микроконтроллер семейства 1986ВЕ9х?

Поясните организацию и работу системного таймера в 1986ВЕ9x?

Поясните работу разработанной программы.

Поясните, каким образом реализуется задержка.

Что понимают под механизмом прерываний?

Как используются прерывания для формирования задержки?

Что такое линия ввода-вывода?

Сколько линий ввода-вывода доступно у микроконтроллера K1986ВЕ92QI?

Зачем задают скорость работы цифрового выхода?

В каких режимах может работать вывод микроконтроллера K1986ВЕ92QI?

Как задать состояние цифрового выхода?

Как узнать, в каком состоянии находится цифровой выход?

Как прочитать состояние цифрового входа?

Какие меры следует предусмотреть для защиты от ложных срабатываний кнопки?

**6 Вывод на графический дисплей**

Необходимо разработать и кодировать на языке C алгоритм программы вывода символьной информации.

Нажатие с последующим отпусканием кнопки должно вызывать модификацию содержимого ячейки Cnt. Содержимое ячейки Cnt должно отображаться на LCD в десятичном коде для вариантов 1 – 3 и в шестнадцатеричном коде для вариантов 4 - 6. Варианты заданий приведены в таблице 1.1.

Вывод информации на графический дисплей (LCD) в отладочных платах LDM-K1986ВЕ92QI-H осуществляется при помощи жидко-кристаллического (ЖК) дисплея MT-12864J-2YLG-3V0 (производство компании «МЭЛТ», г. Москва). Контроллер управления графического дисплея К145ВГ10 (производство ОАО «Ангстрем», г. Зеленоград) аналогичен контроллеру KS0108 фирмы SAMSUNG. Полная техническая информация на модуль графического дисплея MT-12864J содержится в спецификации.

Графический модуль дисплея содержит два независимых контроллера, поскольку видимая область (отображаемое поле точек) графического дисплея (128×64 точек) состоит из двух областей (64×64 точки). Каждая область дисплея работает под управлением своего контроллера. Эти контроллеры при программировании обозначают терминами «Кристалл 1» и «Кристалл 2» (Chip 1 и Chip 2). Для хранения данных, выводимых на ЖК-дисплей, модуль содержит ОЗУ размером 1024 байта: 64×64×2 бит (по 64×64 бит на каждый кристалл). Для выбора нужного кристалла используются выводы E1, E2. ОЗУ каждого кристалла разбито на 8 страниц (Page 0 – 7) размером по 64×8 бит каждая. Каждой светящейся точке на ЖК-дисплее соответствует логическая "1" в ячейке ОЗУ модуля. Для начальной установки модуля необходимо подать сигнал RES (логический "0") длительностью не менее 1 мкс. После деактивации сигнала RES (переключение в логическую "1" с временем фронта не более 200 нс) необходимо выдержать паузу не менее 10 мкс. При программировании каждая страница представлена в виде матрицы 64×8 бита. Запись информации в модуль осуществляется по страницам (64 байта данных). Для записи байта данных по произвольному адресу необходимо предварительно установить страницу ОЗУ и установить адрес внутри страницы ОЗУ. Это осуществляется командами Set Page и Set Address соответственно. После этого можно записать байт данных. Запись каждого байта должна сопровождаться подачей строба (переключение в логическую "1" с временем фронта не более 200 нс) на вход E. Модуль поддерживает непрерывную последовательность операций записи: после записи одного байта счетчик адреса автоматически увеличивается на 1 и модуль готов к новой операции записи по следующему адресу без предварительной установки страницы ОЗУ и адреса. Счетчик адреса считает только внутри одной страницы. При достижении адреса 63 следующим значением счетчика будет 0 и т. д. Между любыми двумя передачами данных или команд необходимо выдержать паузу не менее 8 мкс или ожидать сброса флага BUSY в регистре состояния того кристалла, к которому будет обращение. Возможна скорость потока данных до 100–130 тыс. байтов / с. Графический модуль может также работать в режиме чтения ОЗУ. Линия R/W служит для переключения режима чтение/запись. Линия A0 служит для переключения между режимами «Команды» (A0 = 0) или «Данные» (A0 = 1). Рутинные процедуры для работы с графическим модулем дисплея собраны в библиотеку, которая представлена в виде отдельного файла lcd\_mlt.lib. Для организации взаимодействия с графическим ЖК-модулем данную библиотеку следует подключить к проекту посредством команд Project/Manage/Components, Environment, Books.../Project items/Files/Add Files. Прототипы всех библиотечных функций собраны в отдельный файл lcd\_mlt.h, который необходимо включить в программу посредством оператора #include. Рассмотрим несколько программ, которые используют библиотечные функции для работы с графическим модулем ЖК-дисплея. Программа 4. Вывод графической информации на ЖК-дисплей организован в виде программы (рис. 3.12). Библиотека lcd\_mlt.lib была подключена к проекту с помощью команд [Project/Manage...].

**Контрольные вопросы**

Поясните подключение к микроконтроллеру жидкокристаллическим индикатором MT-12864J.

Поясните состав библиотеки MLT\_LCD.

Поясните назначение и работу функций U\_MLT\_Ini, U\_MLT\_Pin\_Cfg, U\_MLT\_LCD\_Init.

Поясните назначение и работу функций U\_MLT\_Disp\_On, U\_MLT\_Disp\_Off.

Поясните назначение и работу функций U\_MLT\_Set\_Page, U\_MLT\_Set\_Address, U\_MLT\_Clear\_Page.

Поясните назначение и работу функций U\_MLT\_Put\_Char, U\_MLT\_Put\_String, sprint.

Поясните структуру сформированной программы.

**7 Аналоговый ввод в K1986BE92QI**

**7.1 Описание задания**

7.1.1 В микроконтроллере реализовано два 12-разрядных АЦП. С помощью АЦП можно оцифровать сигнал от 16 внешних аналоговых выводов порта D и от двух внутренних каналов, на которые выводятся датчик температуры и источник опорного напряжения. Скорость выборки составляет до 512 тысяч преобразований в секунду для каждого АЦП. В качестве опорного напряжения преобразования могут выступать:

– питание АЦП с выводов AUCC и AGND;

– внешние сигналы с выводов ADC0\_REF+ и ADC\_REF-.

Контроллер АЦП позволяет:

- оцифровать сигнал с одного из 16 внешних каналов;

- оцифровать значение встроенного датчика температуры;

- оцифровать значение встроенного источника опорного напряжения;

- выполнять автоматический опрос заданных каналов;

- формировать запрос прерывания при выходе оцифрованного значения за заданные пределы;

- запускать два АЦП синхронно для увеличения скорости выборки.

Для осуществления преобразования требуется не менее 28 тактов синхронизации CLK. В качестве синхросигнала может выступать CPU\_CLK процессора, либо ADC\_CLK, формируемый в блоке «Сигналы тактовой частоты». Выбор синхросигнала осуществляется с помощью бита Cfg\_REG\_CLKS. Сигнал CPU\_CLK формируется путем деления частоты опорного генератора на коэффициент Cfg\_REG\_DIVCLK[3:0]. Максимальная частота CLK не может превышать 14 МГц.

Внутри микроконтроллеров семейства 1986ВЕ9х есть встроенный датчик температуры, который позволяет измерять температуру корпуса микроконтроллера. Аналоговый сигнал датчика подается на канал 31 АЦП (ADC\_CH\_TEMP\_SENSOR).

АЦП начинает преобразование по команде процессорного ядра, по завершении результаты преобразования поступают в 32-битный регистр ADC1\_RESULT для ADC1 или ADC2\_RESULT для ADC2, откуда их затем можно считать. По завершении преобразование АЦП устанавливает флаг конец преобразования FLG\_REG\_EOCIF, который доступен для чтения в регистрах состояния АЦП – ADC1\_STATUS и ADC2\_STATUS. Проверяя значение этого флага, можно определить момент завершения преобразования. Кроме того, АЦП может формировать запрос прерывания по окончании преобразования. При работе с каждым из двух АЦП в микроконтроллерах семейства 1986ВЕ9х можно использовать следующие основные режимы:

− режим одиночного преобразования по одному каналу с опросом бита окончания преобразования;

− режим одиночного преобразования по одному каналу с прерыванием по окончанию преобразования;

− режим многократного преобразования по одному каналу с использованием прямого доступа к памяти;

− режим многократного преобразования c автоматическим переключением нескольких каналов и использованием прямого доступа к памяти.

В работе используется второй режим. Процесс работы с АЦП при использовании прерывания по окончанию преобразования выглядит следующим образом:

1) запуск преобразования;

2) ожидание сигнала из обработчика прерываний о завершении преобразования;

3) считывание результатов преобразования;

4) обработка результатов преобразования;

5) пауза;

6) возврат к пункту 1.

В работе решается задача, включающая выполнение функций:

- модификация и вывод на ЖКИ значения переменной cell;

- аналого-цифровое преобразование сигнала датчика температуры;

- отображение значения температуры на ЖКИ.

Программа состоит из основного модуля main.c, модуля работы с ЖКИ mlt\_lcd.c и модуля работы с АЦП adc.c. В модуле adc сохранены функция инициализации АЦП U\_ADC\_Init() и обработчик прерывания ADC\_IRQHandler().

В функции U\_ADC\_Init() определены структуры для инициализации АЦП

ADC\_InitTypeDef ADC\_InitStructure;

ADCx\_InitTypeDef ADCx\_InitStructure;

При настройке общей конфигурации АЦП1 разрешим работу температурного датчика:

ADC\_InitStructure.ADC\_TempSensor = ADC\_TEMP\_SENSOR\_Enable; ADC\_InitStructure.ADC\_TempSensorAmplifier = ADC\_TEMP\_SENSOR\_AMPLIFIER\_Enable; ADC\_InitStructure.ADC\_TempSensorConversion = ADC\_TEMP\_SENSOR\_CONVERSION\_Enable;

При выборе канала АЦП указываем канал для температурного датчика:

ADCx\_InitStructure.ADC\_ChannelNumber = ADC\_CH\_TEMP\_SENSOR;

Функция инициализации АЦП U\_ADC\_Init() разрешает прерывания по окончанию аналого-цифрового преобразования:

ADC1\_ITConfig (ADC1\_IT\_END\_OF\_CONVERSION, ENABLE);

Разрешает работу АЦП1:

ADC1\_Cmd (ENABLE);

Также задает приоритет аппаратного прерывания от АЦП:

NVIC\_SetPriority (ADC\_IRQn, 1);

и разрешает аппаратные прерывания от АЦП:

NVIC\_EnableIRQ (ADC\_IRQn);

Функция U\_ADC\_Init() вызывается из основного модуля программы после инициализации портов.

Зададим период аналого-цифрового преобразования около 1 с. Основной цикл программы содержит вызов функции задержки 30 мс. Используем счетчик циклов cntr. Запуск АЦП функцией ADC1\_Start() будет происходить, если предыдущее преобразование завершено (флаг conInProgress сброшен) и после предыдущего запуска выполнено 33 цикла.

После запуска преобразования функцией ADC1\_Start(), процессорное ядро сбрасывает счетчик циклов cntr и устанавливает флаг выполнения преобразования conInProgress. Сброс флага conInProgress выполняет обработчик аппаратного прерывания от АЦП ADC\_IRQHandler().

В обработчике проверяется, что прерывание возникло из-за окончания преобразования ADC1\_IT\_END\_OF\_CONVERSION. При выполнении условия считывается результат преобразования

DU = ADC1\_GetResult () & 0x0000FFFF;

При обработке результатов требуется преобразовать значение DU, полученное с АЦП, в значение температуры в градусах Цельсия. Для этого нужны калибровочные данные, которые являются результатами аналого-цифрового преобразования для двух разных температур, например T1 = 25 ºС и T2 = 85 ºС. Калибровку датчика температуры производят в климатической камере. Температуру можно определить по формуле:

T = (DU − D1) ∙ (T2 – T1) (D2 − D1) + T1, (3.1)

− D1 – значение кода АЦП при температуре T1;

− D2 – значение кода АЦП при температуре T2.

Приблизительные калибровочные данные приведены в заголовочном файле adc.h:

#define ADC\_TS\_T1 25.0F // Температура в 1-й точке

#define ADC\_TS\_D1 0x6A0 // Значение кода АЦП в 1-й точке

#define ADC\_TS\_T2 60.0F // Температура во 2-й точке

#define ADC\_TS\_D2 0x7A0 // Значение кода АЦП во 2-й точке

Значения переменных T и DU выводятся на ЖКИ. Сбрасывается флаг conInProgress и прерывание от АЦП выводится из режима ожидания

NVIC\_ClearPendingIRQ(ADC\_IRQn);

**7.2 Содержание отчета**

Отчет должен содержать:

- описание решения задачи и схема алгоритма;

- текст программы с комментариями.

**8 Аналоговый ввод через канал прямого доступа к памяти**

Необходимо разработать и кодировать на языке C алгоритм программы аналогового ввода. Нажатие с последующим отпусканием кнопки должно вызывать инкремент переменной Cnt. Значение трех младших разрядов Cnt должно выводиться на линии порта PB3 - PB0. Программа должна выполнять запуск модуля аналого-цифрового преобразователя (АЦП), получать результаты преобразования, обрабатывать и выводить на LCD. Варианты заданий приведены в таблице 4.1.

Таблица 4.1

|  |  |
| --- | --- |
| Вариант | Задание |
| 1 | Информация из канала AN0 в шестнадцатеричной форме (0 .. 3FF) |
| 2 | Информация из канала AN0 в шестнадцатеричной форме (0 .. FF) |
| 3 | Информация из канала AN0 в десятичной форме (0 .. 1023) |
| 4 | Информация из канала AN0 в десятичной форме (0 .. 255) |
| 5 | Информация из канала AN0 в десятичной форме (0.000 .. 4.095) |
| 6 | Информация из канала AN0 в десятичной форме (-128 .. 127) |

Микроконтроллеры семейства 1986ВЕ9x позволяют работать с АЦП с использованием прямого доступа к памяти. Прямой доступ к памяти (англ. DMA – Direct Memory Access) представляет собой механизм, позволяющий периферийным устройствам напрямую, без использования центрального процессора, обращаться к оперативной памяти или памяти программ. Использование DMA позволяет существенно разгрузить центральный процессор, не отвлекая его на взаимодействие с периферийными устройствами. Кроме того, скорость работы периферийных устройств также существенно возрастает. Однако настройка DMA требует определенных знаний и является одним из наиболее сложных аспектов в работе с микроконтроллерами. В рамках работы мы применим DMA для получения блока однотипных измерений с помощью АЦП. Как было видно ранее, результаты измерений напряжения требуют усреднения, чтобы получились стабильные показания. Для этого надо выполнить некоторое количество измерений и найти среднее арифметическое значение. Конечно, это можно сделать и без DMA, последовательно выполнив требуемое количество измерений, но гораздо эффективней это делается с использованием прямого доступа к памяти. На рисунке 3.4 схематично показан процесс работы с АЦП без использования DMA, а на рисунке 3.5 – с использованием DMA. В первом случае процессор вынужден самостоятельно обращаться и к АЦП, и к ОЗУ. Во втором же случае взаимодействие между АЦП и ОЗУ ведется посредством DMA, а процессор в этом не участвует. Рассмотрим процесс использования DMA.

Модуль adc.c.

Измерение температуры корпуса микроконтроллера с использованием DMA производится в задаче U\_ADC\_Task\_Function():

Процесс преобразования запускается путем вызова функции DMA\_Cmd(), разрешающей работу DMA совместно с АЦП.

АЦП последовательно выполнит количество преобразований, определяемое константой U\_ADC\_BUFFER\_SIZE, описанной в заголовке adc.h (в нашем случае 256 преобразований). Результаты будут помещены в массив ADC\_Buffer. Заметим, что процессор совершенно не будет отвлекаться на эту работу, а будет заниматься выполнением других задач. Как только последнее из 256 преобразований будет выполнено, DMA сгенерирует аппаратное прерывание, и процессор переключится на выполнение функции-обработчика аппаратных прерываний от канала DMA (DMA\_Channel\_ADC1). Первым делом при этом необходимо подготовить к работе новый цикл аналого-цифровых преобразований и запретить дальнейшую работу канала DMA c АЦП. Если этого не сделать, процессор будет бесконечно заходить в эту функцию-обработчик прерывания и, очевидно, зависнет.

После этого задача U\_ADC\_Task\_Function продолжит свое выполнение, усреднив результат преобразований:

for (i = 0, D = 0; i < U\_ADC\_BUFFER\_SIZE; i++)

D += ADC\_Buffer[i];

D /= U\_ADC\_BUFFER\_SIZE

Далее усредненный результат преобразуется к температуре уже рассмотренным ранее способом:

T = ((int32\_t) D - (int32\_t) ADC\_TS\_D1) \* (ADC\_TS\_T2 - ADC\_TS\_T1) / (ADC\_TS\_D2 - ADC\_TS\_D1) + ADC\_TS\_T1;

3.10.2. Настройка прямого доступа к памяти для работы с АЦП

Теперь рассмотрим процесс настройки DMA для совместной работы с АЦП. Совокупность настроек DMA в целом располагаем в структуре DMA\_InitStructure: DMA\_CtrlDataInitTypeDef DMA\_InitStructure;

Настройки канала DMA располагаем в структуре DMA\_Channel\_InitStructure: DMA\_ChannelInitTypeDef DMA\_Channel\_InitStructure;

Заметим, что обе структуры являются глобальными, поскольку с ними придется работать и за пределами модуля generator.c, а именно – в модуле обработчиков аппаратных прерываний MDR32F9Qx\_it.c. В модуле adc.c настройки DMA сосредоточены в функции DMA\_Config():

static void DMA\_Config(void)

{

// Разрешить тактирование

DMA RST\_CLK\_PCLKcmd (RST\_CLK\_PCLK\_DMA | RST\_CLK\_PCLK\_SSP1 | RST\_CLK\_PCLK\_SSP2, ENABLE);

// Запретить все прерывания, в том числе от SSP1 и SSP2

NVIC->ICPR[0] = 0xFFFFFFFF; NVIC->ICER[0] = 0xFFFFFFFF;

// Сбросить все настройки DMA

DMA\_DeInit();

DMA\_StructInit (&DMA\_Channel\_InitStructure); DMA\_InitStructure.DMA\_SourceBaseAddr = (uint32\_t)(&(MDR\_ADC->ADC1\_RESULT));

DMA\_InitStructure.DMA\_DestBaseAddr = (uint32\_t) &ADC\_Buffer; DMA\_InitStructure.DMA\_CycleSize = U\_ADC\_BUFFER\_SIZE; DMA\_InitStructure.DMA\_SourceIncSize = DMA\_SourceIncNo; DMA\_InitStructure.DMA\_DestIncSize = DMA\_DestIncHalfword; DMA\_InitStructure.DMA\_MemoryDataSize = DMA\_MemoryDataSize\_HalfWord; DMA\_InitStructure.DMA\_NumContinuous = DMA\_Transfers\_1; DMA\_InitStructure.DMA\_SourceProtCtrl = DMA\_SourcePrivileged; DMA\_InitStructure.DMA\_DestProtCtrl = DMA\_DestPrivileged; DMA\_InitStructure.DMA\_Mode = DMA\_Mode\_Basic;

// Задать структуру канала DMA\_

Channel\_InitStructure.DMA\_PriCtrlData = &DMA\_InitStructure; DMA\_Channel\_InitStructure.DMA\_Priority = DMA\_Priority\_Default; DMA\_Channel\_InitStructure.DMA\_UseBurst = DMA\_BurstClear; DMA\_Channel\_InitStructure.DMA\_SelectDataStructure = DMA\_CTRL\_DATA\_PRIMARY;

// Инициализировать канал DMA

DMA\_Init (DMA\_Channel\_ADC1, &DMA\_Channel\_InitStructure);

MDR\_DMA->CHNL\_REQ\_MASK\_CLR = 1 << DMA\_Channel\_ADC1; MDR\_DMA->CHNL\_USEBURST\_CLR = 1 << DMA\_Channel\_ADC1;

// Разрешить работу DMA с каналом АЦП1 DMA\_

Cmd (DMA\_Channel\_ADC1, ENABLE);

// Задать приоритет аппаратного прерывания от DMA

NVIC\_SetPriority (DMA\_IRQn, 1);

}

Вначале нужно разрешить тактирование DMA и обязательно интерфейсов SSP1 и SSP2:

RST\_CLK\_PCLKcmd (RST\_CLK\_PCLK\_DMA | RST\_CLK\_PCLK\_SSP1 | RST\_CLK\_PCLK\_SSP2, ENABLE);

По логике вещей, последовательные интерфейсы SSP1 и SSP2 не имеют непосредственного отношения к DMA, хотя и могут работать с ним. Но в микроконтроллеры семейства 1986ВЕ9х вкралась аппаратная ошибка, в результате которой невозможно корректно работать с DMA, если SSP1 и SSP2 не тактируются. Поэтому возьмем за правило включать тактирование интерфейсов SSP1 и SSP2, когда DMA задействован в проекте. Если этого не сделать, программа попросту зависнет после возникновения прерывания от DMA. По той же причине на время настройки DMA нужно отключить все аппаратные прерывания:

NVIC->ICPR[0] = 0xFFFFFFFF; NVIC->ICER[0] = 0xFFFFFFFF;

Деинициализируем, т.е. сбросим настройки DMA:

DMA\_DeInit();

Инициализируем структуру для настройки DMA, передав в качестве параметра адрес структуры для настройки требуемого канала DMA:

DMA\_StructInit (&DMA\_Channel\_InitStructure);

Далее заполняем структуру для настройки DMA. Зададим адрес источника данных, в качестве которого будет использоваться регистр результатов преобразования данных АЦП1 – ADC1\_RESULT. Оттуда будут браться данные с помощью DMA.

DMA\_InitStructure.DMA\_SourceBaseAddr = (uint32\_t)(&(MDR\_ADC->ADC1\_RESULT));

Укажем адрес приемника данных. В качестве него используется буфер (массив) результатов измерений ADC\_Buffer. В его элементы будут записываться результаты аналого-цифрового преобразования: DMA\_InitStructure.DMA\_DestBaseAddr = (uint32\_t) &ADC\_Buffer;

Зададим размер буфера результатов измерений, т.е. определим, сколько раз мы произведем измерения:

DMA\_InitStructure.DMA\_CycleSize = U\_ADC\_BUFFER\_SIZE;

Запретим автоматическое увеличение адреса источника, поскольку читаем данные всегда из одного и того же регистра ADC1\_RESULT: DMA\_InitStructure.DMA\_SourceIncSize = DMA\_SourceIncNo;

Разрешаем автоматическое увеличение адреса приемника на полслова (т.е. на 2), поскольку записываем данные в разные элементы буфера результатов измерений ADC\_Buffer:

DMA\_InitStructure.DMA\_DestIncSize = DMA\_DestIncHalfword;

Указываем, по сколько байт будем передавать за один раз. В нашем случае по два байта или, по-другому, по полслова: DMA\_InitStructure.DMA\_MemoryDataSize = DMA\_MemoryDataSize\_HalfWord;

Казалось бы, зачем нужна эта настройка? Ведь мы только что указали, что адрес приемника будет автоматически увеличиваться на те же два байта. Однако это разные вещи. Элементы массива могут иметь больший размер, чем количество данных, передаваемых за один раз. Например, мы можем использовать для хранения результатов измерений массив с элементами по 4 байта, но передавать за один раз лишь два байта. При этом у нас будут следующие настройки:

DMA\_InitStructure.DMA\_DestIncSize = DMA\_DestIncWord; DMA\_InitStructure.DMA\_MemoryDataSize = DMA\_MemoryDataSize\_HalfWord; Таким образом, адрес автоматически увеличивается на 4 байта, а передаем за раз лишь два байта. Далее укажем, сколько раз будем повторять полные циклы передачи данных:

DMA\_InitStructure.DMA\_NumContinuous = DMA\_Transfers\_1;

В нашем случае требуется выполнить цикл лишь один раз. Это означает, что DMA один раз выполнит передачу U\_ADC\_BUFFER\_SIZE элементов данных, а затем остановится. Можно выбирать значения: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 и 1024. Режим бесконечного повтора передачи, к сожалению, не предусмотрен. Включим режимы защиты источника и приемника: DMA\_InitStructure.DMA\_SourceProtCtrl = DMA\_SourcePrivileged; DMA\_InitStructure.DMA\_DestProtCtrl = DMA\_DestPrivileged;

Смысл этих двух настроек не будем сейчас рассматривать. Также зададим так называемый базовый режим работы DMA:

DMA\_InitStructure.DMA\_Mode = DMA\_Mode\_Basic;

Помимо базового, есть еще целый ряд других режимов работы DMA. Но объяснение их смысла займет много времени и отвлечет нас от основной цели данной работы. Тем, кто заинтересовался этим вопросом, можно порекомендовать обратиться к фирменной документации на микроконтроллеры семейства 1986ВЕ9х.

Теперь заполним структуру для настройки канала DMA. К первичному блоку настроек канала привяжем ранее заполненную структуру настройки DMA и выберем ее в качестве используемой в данный момент:

DMA\_Channel\_InitStructure.DMA\_PriCtrlData = &DMA\_InitStructure; DMA\_Channel\_InitStructure.DMA\_SelectDataStructure = DMA\_CTRL\_DATA\_PRIMARY;

Вообще-то есть два блока настроек канала DMA: первичный (DMA\_PriCtrlData) и альтернативный (DMA\_AltCtrlData). Альтернативный требуется лишь при режиме пинг-понг, поэтому нами сейчас не рассматривается. Приоритет канала DMA установим по умолчанию: DMA\_Channel\_InitStructure.DMA\_Priority = DMA\_Priority\_Default;

Снимем запрет на одиночные запросы к DMA: DMA\_Channel\_InitStructure.DMA\_UseBurst = DMA\_BurstClear;

Полученной структурой инициализируем канал DMA для работы с таймером АЦП1:

DMA\_Init (DMA\_Channel\_ADC1, &DMA\_Channel\_InitStructure);

Снимем запрет на запросы к DMA со стороны АЦП1. Это рекомендуется для нормальной работы DMA (что-то вроде заплатки для устранения аппаратных ошибок в микроконтроллерах 1986ВЕ9x):

MDR\_DMA->CHNL\_REQ\_MASK\_CLR = 1 << DMA\_Channel\_ADC1; MDR\_DMA->CHNL\_USEBURST\_CLR = 1 << DMA\_Channel\_ADC1;

Разрешим работу DMA с каналом для АЦП1:

DMA\_Cmd (DMA\_Channel\_ADC1, ENABLE);

Зададим приоритет 1 для аппаратного прерывания от DMA:

NVIC\_SetPriority (DMA\_IRQn, 1);

В функции ADC\_Config() производится настройка АЦП, выбирается режим циклического (многократного) преобразования:

ADCx\_InitStructure.ADC\_SamplingMode = ADC\_SAMPLING\_MODE\_CICLIC\_CONV;

Вместо прерываний от ADC разрешаются прерывания от DMA: NVIC\_EnableIRQ (DMA\_IRQn);

**Контрольные вопросы**

К какому типу принадлежит АЦП, встроенный в микроконтроллер, как он работает.

Какая максимальная частота дискретизации АЦП?

Объясните работу АЦП в режиме обмена данными без участия процессора.

Что такое DMA? Для чего его используют?

Каким образом выполняется инициализация модуля АЦП?

Каким образом выполняется запуск АЦП?

Как программа определяет момент окончания аналого-цифрового преобразования?

К какому каналу АЦП подключен датчик температуры, встроенный в микроконтроллер?

Каким образом можно привести показания АЦП от температурного датчика к значению температуры в градусах Цельсия?

**9 Обмен через последовательные интерфейсы**

**9.1 Описание задания**

9.1.1 Универсальный асинхронный приёмопередатчик (*UART*) — узел вычислительных устройств, предназначенный для организации связи с другими цифровыми устройствами. Преобразует передаваемые данные в последовательный вид так, чтобы было возможно передать их по цифровой линии другому аналогичному устройству. Метод преобразования хорошо стандартизован и широко применяется в компьютерной технике. Представляет собой логическую схему, с одной стороны подключённую к шине вычислительного устройства, а с другой имеющую два или более выводов для внешнего соединения.

При построении кода программы необходимо задать конфигурация 1 вывода порта PORTF как выхода (UART2\_TX)

PortInit.PORT\_OE = PORT\_OE\_OUT;

PortInit.PORT\_Pin = PORT\_Pin\_1;

PORT\_Init(MDR\_PORTD, &PortInit);

и конфигурацию 0 вывода порта PORTF как входа (UART2\_RX)

PortInit.PORT\_OE = PORT\_OE\_IN;

PortInit.PORT\_Pin = PORT\_Pin\_0;

PORT\_Init(MDR\_PORTD, &PortInit);

Разрешить тактирования порта F

RST\_CLK\_PCLKcmd(RST\_CLK\_PCLK\_PORTD, ENABLE);

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 прер. по приему

UART\_ITConfig (MDR\_UART2, UART\_IT\_TX, ENABLE); //Разрешение прерывания по окончани передачи

UART\_Cmd(MDR\_UART2, ENABLE); //Разрешение работы UART2

Основной цикл программы

while (1) {

while (uart2\_IT\_RX\_flag != SET);

//ждем пока не установится 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

}

}

**9.2 Содержание отчета**

Отчет должен содержать:

- описание решения задачи и схема алгоритма;

- текст программы с комментариями.

**10 Использование часов реального времени**

**10.1 Описание задания**

10.1.1 Блок батарейного домена (зона батарейного питания) предназначен для обеспечения функций часов реального времени (Real time clock, RTC) и сохранения некоторого набора пользовательских данных при отключении основного источника питания.

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

Для калибровки тактовой частоты используются биты CAL[6:0]. Значение CAL определяет, какое число тактов из 220 будет замаскировано. Таким образом, с помощью бит CAL[6:0] производится замедление хода часов. Изменение значения бит CAL может быть осуществлено в ходе работы часов реального времени. Регистр RTC\_DIV выступает в роли 20-битного предварительного делителя входной тактовой частоты таким образом, чтобы на его выходе была тактовая частота в 1 Гц. Для задания коэффициента деления регистра RTC\_DIV используется регистр RTC\_PRL. Регистр RTC\_CNT предназначен для отсчета времени в секундах и работает на выходной частоте делителя RTC\_DIV. Регистр RTC\_ALR предназначен для задания времени, при совпадении с которым вырабатывается флаг прерывания и пробуждения процессора. Таким образом, бит STANDBY, отключающий внутренний регулятор напряжения, автоматически сбрасывается при совпадении RTC\_CNT и RTC\_ALR. Бит STANDBY также может быть сброшен с помощью вывода WAKEUP.

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

Для работы с часами реального времени используют пять регистров RTC\_CNT, RTC\_DIV, RTC\_PRL, RTC\_ALRM, RTC\_CS. Особенностью этих регистров является сравнительно медленная запись. Обращение к этим регистрам рекомендуется проводить по технологии «чтение–модификация–запись». В рамках этой технологии предусмотрен флаг готовности к записи (WEC=RTC\_CS [6]). Его необходимо проверять перед каждым актом записи («чтение–модификация–запись») в регистр блока RTC.

MDR\_BKP, базовый адрес 0x400D\_8000 – контроллер батарейного домена и часов реального времени;

MDR\_BKP->REG\_00, смещение 0x00 – BKP-регистр 00 [31:0];

MDR\_BKP->REG\_01, смещение 0x04 – BKP-регистр 01 [31:0];

MDR\_BKP->REG\_02, смещение 0x08 – BKP-регистр 02 [31:0];

. . .

MDR\_BKP->REG\_0D, смещение 0x34 – BKP-регистр 0D [31:0];

MDR\_BKP->REG\_0E, смещение 0x38 –BKP-регистр 0E и биты управления батарейным доменом;

MDR\_BKP->REG\_0F, смещение 0x3C –BKP-регистр 0F и биты управления блоками RTC, LSE, LSI и HSI;

MDR\_BKP->RTC\_CNT, смещение 0x40, [31:0] (RTC\_CNT) – значение основного счетчика часов реального времени;

MDR\_BKP->RTC\_DIV, смещение 0x44,

[31:20] – не используется;

[19:0] (RTC\_DIV) – значение счетчика предварительного делителя часов реального времени;

MDR\_BKP->RTC\_PRL, смещение 0x48,

[19:0] (RTC\_PRL) – значение основания для счета счетчика предварительного делителя часов реального времени;

Программа, демонстрирующая работу часов реального времени в батарейном домене микроконтроллера. Программа инициализирует систему, состоящую из часов реального времени (RTC), расположенных в батарейном домене микроконтроллера и графического дисплея. На экране дисплея программа отображает статическую информацию о тактовой частоте микроконтроллера. После запуска часов программа каждую секунду отображает на экране дисплея динамическую информацию о текущем времени в формате «часы, минуты, секунды». Благодаря тому, что часы расположены в батарейном домене, работоспособность часов сохраняется после выключения основного питания контроллера. При повторном включении контроллера не происходит повторной инициализации часов, вместо этого программа просто выводит на экран дисплея динамическую информацию о текущем времени.

Текст программы приведен в файле mainRTC.c

**10.2 Содержание отчета**

Отчет должен содержать:

- описание решения задачи и схема алгоритма;

- текст программы с комментариями.

**Приложение Б**

**Методические рекомендации для выполнения лабораторных работ**

**Работа в среде разработки Keil uVision**

Создание нового проекта

Создать новый проект командой меню Project – New uVision Project… (рисунок 2) и выбираем папку для нового проекта (в названии пути к папке не должно быть русских символов). Рекомендуется для каждой лабораторной работы создавать новую папку, расположенную в C:\Keil\ARM\Labs\.

Затем будет предложено выбрать модель целевого МК, для которого будет написана программа. Выбираем микроконтроллер MDR1986BE92 (Milandr – Milandr – Cortex-M3- MDR1986BE92).

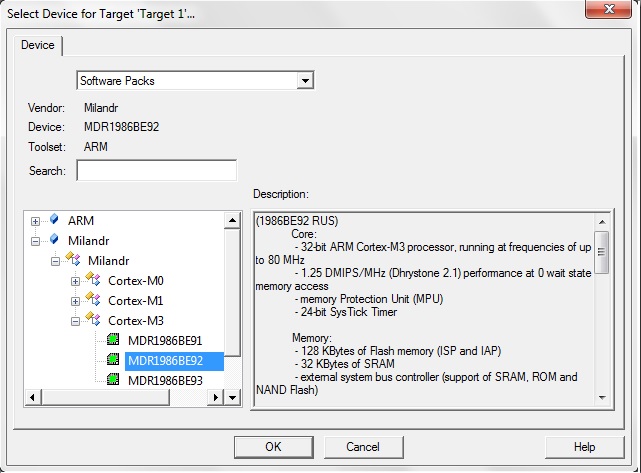


Рисунок 3 – Выбор модели МК

После выбора целевого микроконтроллера необходимо определить периферию и окружение компиляции. Для каждой лабораторной работы потребуется свой набор. Для первого проекта выберем (рисунок 4) Device\Startup\_MDR1986BE9x (поддержка ядра микроконтроллера серии 1986BE9x), Drivers\PORT (порты ввода/вывода), Drivers\RST\_CLK (сигналы тактовой частоты).

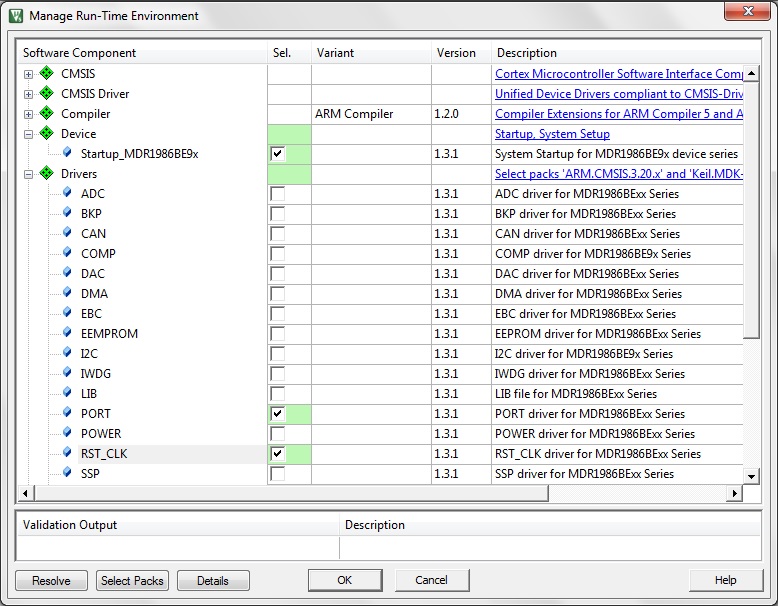


Рисунок 4 –Окно выбора окружения компиляции

Затем в дереве проекта щелкнуть правой кнопкой мыши на «Source group 1» и выбрать «add new item to Group 'Source group 1'». В появившемся окне выбрать файл расширения «\*.h», задать ему имя «MDR32F9Qx\_board.h» и сохранить в папку «config» в той же директории, что и проект.

Далее необходимо указать компилятору путь до файла «MDR32F9Qx\_board.h». Для этого в дереве проекта щелкнуть правой кнопкой мыши на «Target 1», выбрать «Options for Target 1» и перейти на вкладку «C/C++».В поле «Include path» добавить строку «./config».

Далее необходимо подключить демонстрационно-отладочную плату 1986EvBrd к компьютеру. Для этого нужно подключить JTAG-отладчик при помощи шлейфа к разъему «JTAG-B» отладочной платы. Выбрать режим загрузки «Flash/JTAG\_B», установкой значений«0» и «0» на переключателях «SW1» и «SW2» соответственно. Подключить отладчик к компьютеру при помощи кабеля USB, удостовериться, что операционная система правильно обнаружила и установила драйверы устройства. Завершающий шаг - настройка JTAG-отладчика. Для его настройки необходимо в дереве проекта щелкнуть правой кнопкой мыши на «Target 1 - Options for Target 1» и перейти на вкладку «Debug» (рисунок 5).

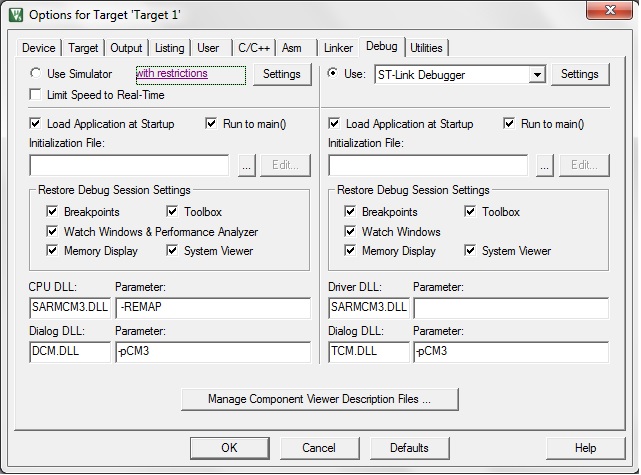


Рисунок 5 – Окно настроек Target 1, вкладка Debug

В выпадающем списке выбрать «ST-Link Debugger». Затем нажать кнопку «Settings». В выпадающем меню «Port» необходимо выбрать режим «SW» (рисунок 8), и задать скорость передачи данных («Max Clock») 24 не более 2 MHz. Удостовериться, что в поле «SW Device» присутствует устройство ARMCoreSightSW-DP.

Рисунок 8 – Окно настроек отладчика, вкладка Debug

Далее необходимо задать порядок загрузки программы в память микроконтроллера. Для этого перейти на вкладку «Flash Download» (рисунок 6), нажать кнопку «add» и из списка выбрать «1986BE IAP 128kB Flash».Затем отметить позиции: «Erase Full Chip», «Program», «Verify», «Reset and Run» на вкладке «Flash Download».

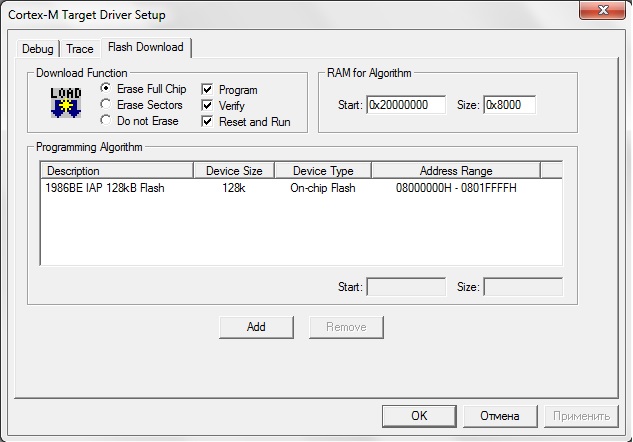


Рисунок 6 –Окно настроек отладчика, вкладка Flash Download

Теперь проект настроен и готов к работе: можно вводить текст программы, отлаживать ее и компилировать. Для этого необходимо добавить в проект новый файл main.c (в дереве проекта щелкнуть правой кнопкой мыши на «Source group 1 - add new item to Group `Source group 1''и выбрать «CFile (.c)»). В появившемся окне можно писать код программы. Например, любой код из представленных в разделе 4. Далее нужно скомпилировать код программы. Для этого необходимо выбрать «Project-Build target», либо нажать на соответствующую кнопку на панели инструментов или «горячей» клавишей F7.

Скомпилированная без ошибок программа может быть загружена в память микроконтроллера. Для этого нужно выбрать «Flash-Download» или нажать на соответствующую кнопку на панели инструментов.

Для отладки программы в среде программирования Keil uVision могут быть использованы средства JTAG-отладчика. Для этого необходимо нажать на кнопку «start/stop debug». Режим отладки предоставляет возможность ставить точки остановки выполнения программы на микроконтроллере.

После нажатия кнопки «start/stop debug» окно программы Keil uVision изменяется. При этом программа на микроконтроллере не выполняется до тех пор, пока пользователь не нажмет соответствующую кнопку.

Для управления ходом выполнения программы предусмотрен соответствующий блок кнопок:

– кнопка Run запускает выполнение программы на микроконтроллере;

– кнопка Stop останавливает выполнение программы;

– кнопка Step выполняет переход к следующей строке по ходу выполнения программы;

– кнопка Step over выполняет переход к следующей строке по ходу выполнения программы, не заходя в текущую функцию;

– кнопка Step out выполняет переход к точке выхода из текущей функции.

В любой момент времени текущая строка подсвечивается при помощи символа стрелки в окне отображения кода программы. Над окном кода программы отображается окно дизассемблирования, в котором показывается ход выполнения программы в кодах ассемблера. Слева от окна кода программы отображается окно управления регистрами микроконтроллера, позволяющее просматривать и редактировать все регистры микроконтроллера. Под окном управления регистрами располагается командное окно, позволяющее выполнять произвольные команды в любой момент времени. Для выхода из режима отладки необходимо нажать на кнопку «start/stop debug».

**5 Отладка программы обмена с портами ввода – вывода**

**5.1 Порядок выполнения**

5.1 Подготовьте план тестирования программы.

5.2 Создайте новый проект. Включите в него драйверы PORT, RST CLK, исходные модули, сформированные при выполнении практического задания 5.

5.3 Выполните настройку компилятора и отладчика системы Keil uVision.

5.4 Подключите отладочный стенд LDM-K1986BE92QI к инструментальному компьютеру через внутрисхемный отладчик ST Link v.2.

5.5 Выполните компиляцию программы и загрузку программы.

5.6 Выполните тестирование программы в соответствии с подготовленным планом.

5.7 По результатам тестирования сделайте выводы о корректности выполнения программы. При обнаружении признаков ошибки необходимо ее локализовать и исправить. Повторно выполнить п.п. 5.5 – 5.6.

**5.2 Контрольные вопросы**

Перечислите основные инструментальные средства и режимы работы отладчика системы Keil uVision.

Перечислите основные режимы отладки программ в среде Keil uVision?

Поясните по схеме алгоритма работу программы.

Перечислите условия контролепригодности.

Приведите примеры использованных средств, обеспечивающих выполнение условий контролепригодности.

Как адаптировать среду Keil для работы с определённой моделью микроконтроллера?

Как выбираются основные опции проекта?

Как создать новый проект в среде Keil?

Каковы назначение и возможности библиотеки CMSIS?

В чем заключаются этапы компиляции, компоновки программы.

Поясните, как построен план тестирования программы.

Какими средствами можно выполнить тестирование программы?

Каким образом результаты тестирования подтверждают корректность работы программы?

**6 Отладка программы вывод на графический дисплей**

**6.1 Порядок выполнения**

6.1 Подготовьте план тестирования программы.

6.2 Создайте новый проект. Включите в него драйверы PORT, RST CLK, исходные модули, сформированные при выполнении практического задания 6, исходные модули библиотеки MLT\_LCD.

6.3 Выполните настройку компилятора и отладчика системы Keil uVision.

6.4 Подключите отладочный стенд LDM-K1986BE92QI к инструментальному компьютеру через внутрисхемный отладчик ST Link v.2.

6.5 Выполните компиляцию программы и загрузку программы.

6.6 Выполните тестирование программы в соответствии с подготовленным планом.

6.7 По результатам тестирования сделайте выводы о корректности выполнения программы. При обнаружении признаков ошибки необходимо ее локализовать и исправить. Повторно выполнить п.п. 6.5 – 6.6.

**6.2 Контрольные вопросы**

Поясните по схеме алгоритма работу программы символьного вывода.

Поясните, как построен план тестирования программы.

Поясните, каким образом сформирован проект в системе Keil MDK-ARM.

Поясните, каким образом выполнена загрузка кода в память программ.

Каким образом результаты тестирования подтверждают корректность работы программы?

**7 Отладка программы аналогового ввода K1986BE92QI**

**7.1 Порядок выполнения**

7.1 Подготовьте план тестирования программы.

7.2 Создайте новый проект. Включите в него драйверы PORT, RST CLK, ADC, DMA, исходные модули, сформированные при выполнении практического задания 8, исходные модули библиотеки MLT\_LCD.

7.3 Выполните настройку компилятора и отладчика системы Keil uVision.

7.4 Подключите отладочный стенд LDM-K1986BE92QI к инструментальному компьютеру через внутрисхемный отладчик ST Link v.2.

7.5 Выполните компиляцию программы и загрузку программы.

7.6 Выполните тестирование программы в соответствии с подготовленным планом.

7.7 По результатам тестирования сделайте выводы о корректности выполнения программы. При обнаружении признаков ошибки необходимо ее локализовать и исправить. Повторно выполнить п.п. 7.5 – 7.6.

**7.2 Контрольные вопросы**

Поясните по схеме алгоритма работу программы аналогового ввода.

Поясните, как построен план тестирования программы.

Поясните, каким образом сформирован проект в системе Keil MDK-ARM.

Поясните, каким образом выполнена загрузка кода в память программ.

Каким образом результаты тестирования подтверждают корректность работы программы?

**8 Отладка программы, использующей канал прямого доступа к памяти**

**8.1 Порядок выполнения**

8.1 Подготовьте план тестирования программы.

8.2 Создайте новый проект. Включите в него драйверы PORT, RST CLK, ADC, DMA, исходные модули, сформированные при выполнении практического задания 8, исходные модули библиотеки MLT\_LCD.

8.3 Выполните настройку компилятора и отладчика системы Keil uVision.

8.4 Подключите отладочный стенд LDM-K1986BE92QI к инструментальному компьютеру через внутрисхемный отладчик ST Link v.2.

8.5 Выполните компиляцию программы и загрузку программы.

8.6 Выполните тестирование программы в соответствии с подготовленным планом.

8.7 По результатам тестирования сделайте выводы о корректности выполнения программы. При обнаружении признаков ошибки необходимо ее локализовать и исправить. Повторно выполнить п.п. 8.5 – 8.6.

**8.2 Контрольные вопросы**

Поясните по схеме алгоритма работу программы аналогового ввода.

Поясните, как построен план тестирования программы.

Поясните, каким образом сформирован проект в системе Keil MDK-ARM.

Поясните, каким образом выполнена загрузка кода в память программ.

Каким образом результаты тестирования подтверждают корректность работы программы?

**9 Отладка программы обмена через последовательные интерфейсы**

**9.1 Порядок выполнения**

9.1.1 Подготовьте план тестирования программы.

9.1.2 Создайте новый проект. Включите в него драйверы PORT, RST CLK, UART, исходные модули, сформированные при выполнении практического задания 15, исходные модули библиотеки MLT\_LCD.

9.1.3 Выполните настройку компилятора и отладчика системы Keil uVision.

9.1.4 Подключите отладочный стенд LDM-K1986BE92QI к инструментальному компьютеру через внутрисхемный отладчик ST Link v.2.

9.1.5 Выполните компиляцию программы и загрузку программы.

9.1.6 Выполните тестирование программы в соответствии с подготовленным планом.

Для работы с модулем UART необходимо подключить нуль-модемный кабель к порту RS-232 отладочной платы и к порту RS-232 персонального компьютера.

Для того что бы принимать и отправлять данные через порт RS-232 (COM порт) необходимо использовать стороннее программное обеспечение, такое как терминальные программы для интерфейса RS-232. Рекомендуется использовать свободную программу Terminal.

**9.2 Контрольные вопросы**

Каким образом линии UART можно подключить к выводам МК?

Каким образом организуется тактирование UART?

Перечислите параметры UART, которые требуется настроить.

Каким образом выполняется настройка параметров UART?

Каким образом выполняется включение модуля UART?

Каким образом программа определяет событие - получение байта?

Каким образом программа инициализирует передачу байта?

Каким образом результаты тестирования подтверждают корректность работы программы?

**10 Отладка программы, использующей часы реального времени**

**10.1 Порядок выполнения**

10.1.1 Подготовьте план тестирования программы.

10.1.2 Создайте новый проект. Включите в него драйверы PORT, RST CLK, RTC, исходные модули, сформированные при выполнении практического задания 16, исходные модули библиотеки MLT\_LCD.

10.1.3 Выполните настройку компилятора и отладчика системы Keil uVision.

10.1.4 Подключите отладочный стенд LDM-K1986BE92QI к инструментальному компьютеру через внутрисхемный отладчик ST Link v.2.

10.1.5 Выполните компиляцию программы и загрузку программы.

10.1.6 Выполните тестирование программы в соответствии с подготовленным планом.

**10.2 Контрольные вопросы**

Поясните организацию и работу модуля RTC.

Каким образом выполнена инициализация модуля RTC?

Поясните по схеме алгоритма работу программы.

Поясните, как построен план тестирования программы.

Поясните, каким образом сформирован проект в системе Keil MDK-ARM.

Поясните, каким образом выполнена загрузка кода в память программ.

Каким образом результаты тестирования подтверждают корректность работы программы?