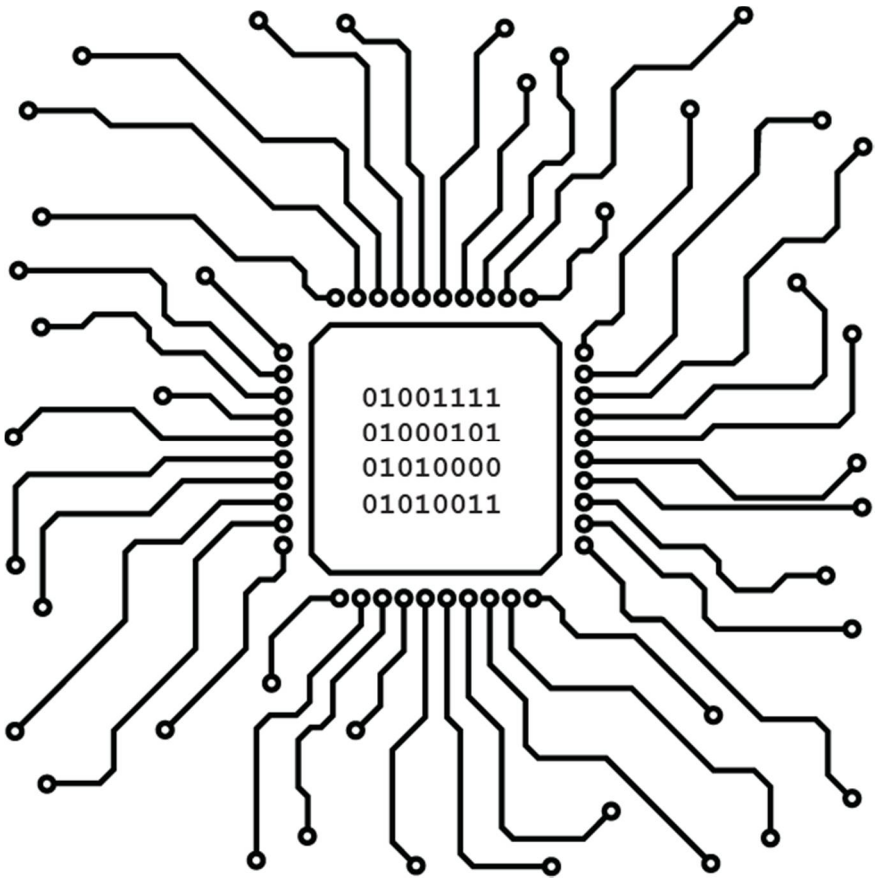
УНИВЕРСИТЕТ ИТМО

А.С. Васильев, О.Ю. Лашманов,

А.В. Пантюшин

ОСНОВЫ ПРОГРАММИРОВАНИЯ МИКРОКОНТРОЛЛЕРОВ



Санкт-Петербург

2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

А.С. Васильев, О.Ю. Лашманов
А.В. Пантюшин

ОСНОВЫ ПРОГРАММИРОВАНИЯ МИКРОКОНТРОЛЛЕРОВ

Учебно-методическое пособие

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург

2016

Васильев А.С., Лашманов О.Ю., Пантюшин А.В. Основы программирования микроконтроллеров. – СПб: Университет ИТМО, 2016. – 95с.

В учебно-методическом пособии рассмотрены основные понятия и концепции программирования микроконтроллеров на примере микроконтроллеров серии 1986ВЕ9х производства ЗАО «ПКК МИЛАНДР» в среде разработки Keil uVision. Первый раздел пособия содержит техническое описание устройства микроконтроллера, теоретические сведения основ программирования на языке С в среде Keil uVision, второй раздел включает варианты индивидуальных заданий для выполнения студентами лабораторных работ по курсу «Основы программирования микроконтроллеров». Задания рассчитаны на освоение и понимания концепции и базового функционала микроконтроллеров серии 1986ВЕ9х.

Учебно-методическое пособие предназначено для студентов по направлению подготовки 12.04.02 «Оптотехника», 27.04.05 «Инноватика» и специальности 12.05.01 «Электронные и оптико-электронные приборы и системы специального назначения»

Рекомендовано к печати ученым советом факультета лазерной и световой инженерии, протокол № 9от 13.09.2016

Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

Введение

Микроконтроллеры представляют собой мощный вычислительный инструмент и прекрасную основу для создания современных высокопроизводительных и экономичных встраиваемых систем многоцелевого назначения. В одной микросхеме микроконтроллер включает в себя микропроцессор, память программ (обычно на основе ПЗУ), память данных (обычно на основе ОЗУ), устройство ввода/вывода, генератор тактовых сигналов, аппаратную поддержку интерфейсов I2C, SPI и многое другое.

Однокристальные микроконтроллеры находят широкое применение в самых разнообразных сферах: от измерительных приборов, фотоаппаратов и видеокамер, принтеров, сканеров и копировальных аппаратов до изделий электронных развлечений и всевозможной домашней техники.

В рамках обучения принципам работы и архитектуры микроконтроллеров студентам будет предложено познакомиться с микроконтроллерами серии 1986ВЕ9х, основанных на ядре ARM архитектуре, производства одного из лидера разработки отечественной микроэлектронной элементной базы ЗАО «ПКК МИЛАНДР».

На заре возникновения микропроцессоров разработка программного обеспечения происходила исключительно на том или ином языке ассемблера, ориентированном на конкретное устройство. По сути, такие языки представляли собой символьные мнемоники соответствующих машинных кодов, а перевод мнемоники в машинный код выполнялся транслятором. При этом главный недостаток ассемблерных языков заключался в том, что каждый из них был привязан к конкретному типу устройств и логике его работы. Кроме того, ассемблер сложен в освоении, что требует достаточно больших усилий для его изучения, но главное, если впоследствии потребуется перейти на использование микроконтроллеров других производителей, то они (усилия) окажутся потраченными впустую.

Студентам, обучающимся по курсу «Основы программирования микроконтроллеров», будет предложено разрабатывать программное обеспечение микроконтроллера на языке С. Язык С, являясь языком высокого уровня, лишен недостатков ассемблера и может использоваться для программирования любого микропроцессора, для которого есть компилятор с языка С. В языке С все низкоуровневые операции, выполняемые компьютерами, представлены в виде абстрактных конструкций, позволяющих разработчикам сосредоточиться на программировании одной лишь логики, не заботясь о машинном коде. Изучив язык С, можно легко переходить от одного семейства микроконтроллеров к другому, тратя гораздо меньше времени на разработку.

Студентам в рамках данного курса будет предложено выполнять разработку программного обеспечения микроконтроллеров в интегрированной среде программирования Keil uVision фирмы Keil Elektronik. Данная среда предоставляет пользователю набор средств для написания и отладки кода программ для микроконтроллеров на основе ядра ARM7, ARM9, Cortex M3 и других. В бесплатный дистрибутив входят следующие средства:

- интегрированная среда разработки;
- C/C++ компилятор;
- макроассемблер и линковщик;
- дебаггер uVision;
- дополнительные утилиты.

1 Архитектура микроконтроллеров серии 1986ВЕ9х

Микроконтроллеры серии 1986ВЕ9х, К1986ВЕ9х и К1986ВЕ92QI, К1986ВЕ92QC (далее 1986ВЕ9х), построенные на базе высокопроизводительного процессорного RISC ядра ARM Cortex-M3, содержат встроенную 128 Кбайт Flash-память программ и 32 Кбайта ОЗУ. Микроконтроллеры работают на тактовой частоте до 80 МГц. Периферия микроконтроллера включает контроллер USB интерфейса со встроенным аналоговым приемопередатчиком со скоростями передачи 12 Мбит/с (Full Speed) и 1.5 Мбит/с (Low Speed), стандартные интерфейсы UART, SPI и I2C, контроллер внешней системной шины, что позволяет работать с внешними микросхемами статического ОЗУ и ПЗУ, NAND Flash-памятью и другими внешними устройствами. Микроконтроллеры содержат три 16-разрядных таймера с 4 каналами схем захвата и ШИМ с функциями формирования «мертвой зоны» и аппаратной блокировки, а также системный 24-х разрядный таймер и два сторожевых таймера. Кроме того, в состав микроконтроллеров входят: два 12-разрядных высокоскоростных (до 0,5М выборок в сек) АЦП с возможностью оцифровки информации от 16 внешних каналов и от встроенных датчиков температуры и опорного напряжения; два 12-разрядных ЦАП; встроенный компаратор с тремя входами и внутренней шкалой напряжений.

Встроенные RC генераторы HSI (8 МГц) и LSI (40 кГц) и внешние генераторы HSE (2…16 МГц) и LSE (32 кГц) и две схемы умножения тактовой частоты PLL для ядра и USB интерфейса позволяют гибко настраивать скорость работы микроконтроллеров. Архитектура системы памяти за счет матрицы системных шин позволяет минимизировать возможные конфликты при работе системы и повысить общую производительность. Контроллер DMA позволяет ускорить обмен информацией между ОЗУ и периферией без участи процессорного ядра. Встроенный регулятор предназначенный для формирования питания внутренний цифровой части формирует напряжения 1,8В и не требует дополнительных внешних элементов. Таким образом, для работы микроконтроллера достаточно одного внешнего напряжения питания в диапазоне от 2,2 до 3,6В. Так же в микроконтроллерах реализован батарейный домен, работающий от внешней батареи при отсутствии основного питания. В батарейном домене могут быть сохранены специальные флаги, а также работают часы реального времени. Встроенные детекторы напряжения питания могут отслеживать уровень внешнего основного питания, уровень напряжения питания на батареи. Аппаратные схемы сброса при просадке питания позволяют исключить сбойную работу микросхемы при выходе уровня напряжения питания за допустимые приделы.

Основные технические характеристики микроконтроллеров серии 1986ВЕ9х приведены в таблице 1.

Таблица1 – Основные характеристики микроконтроллеров серии 1986ВЕ9х

	1986ВЕ91Т **1986ВЕ94Т**	**К1986ВЕ91Н4**	**1986ВЕ92У** **К1986ВЕ92QI** **К1986ВЕ92QC**	**1986ВЕ93У**
Корпус	132 вывода	бескорпусная	64 вывода	48 выводов
Ядро	ARM Cortex-M3			
ПЗУ	128 Кбайт Flash			
ОЗУ	32 Кбайт			
Питание	2,2…3,6 В			
Частота	80 МГц			
USER IO	96	96	43	30
USB	Device и Host FS (до 12 Мбит/с) встроенный PHY			
UART	2	2	2	2
CAN	2	2	2	2
SPI	2	2	2	1
I2C	1	1	1	-
2 х 12-разрядных АЦП	16 каналов	16 каналов	8 каналов	4 канала
ЦАП 12 разрядов	2	2	1	1
Компаратор	3 входа	3 входа	2 входа	2 входа
Внешняя шина	32 разряда	32 разряда	8 разряда	-

На рисунке1 показана структурная блок-схема микроконтроллера серии 1986ВЕ9х.

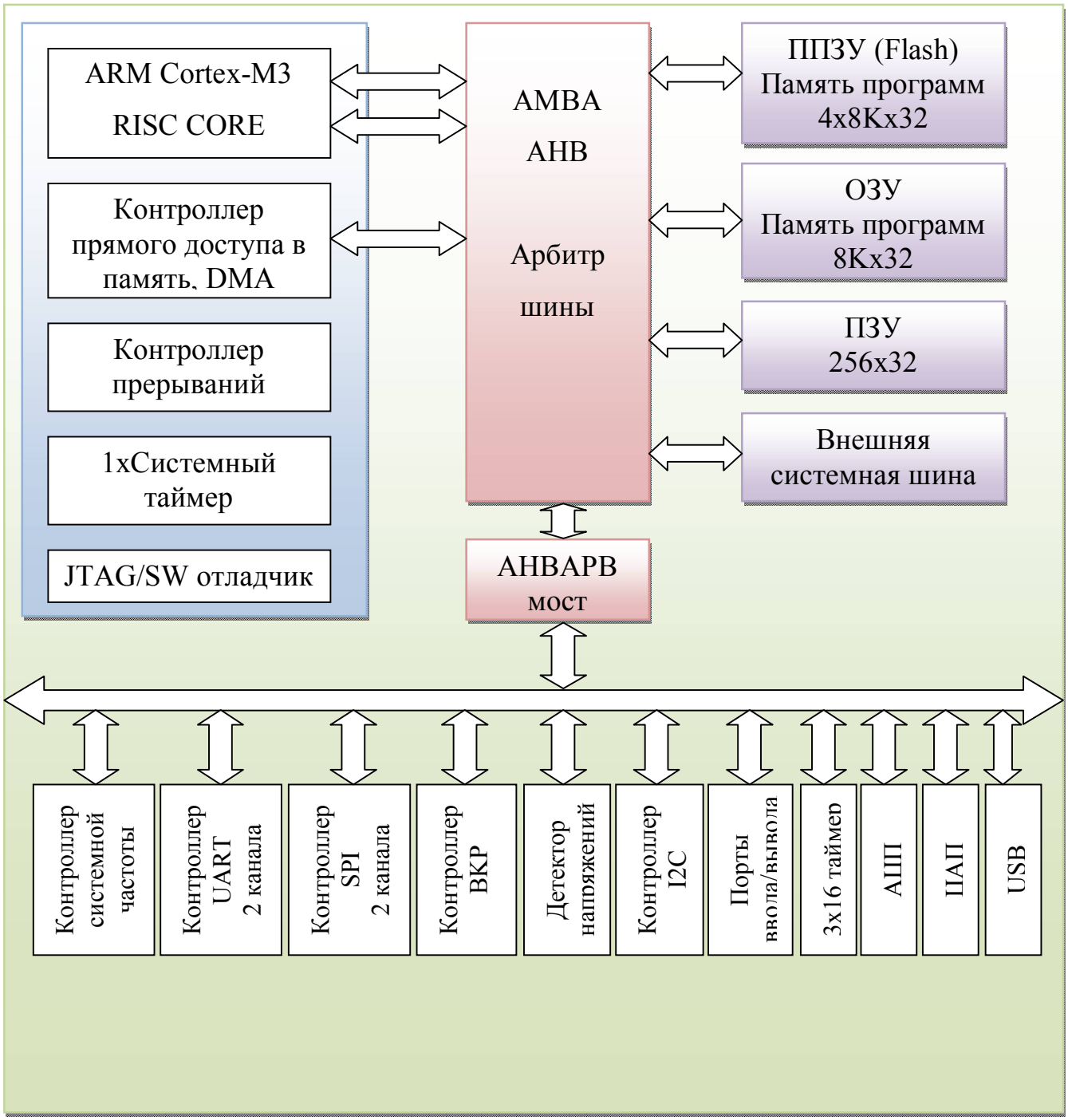


Рисунок 1 – Структурная блок-схема микроконтроллера серии 1986ВЕ9х.

Память микроконтроллера реализована с единым адресным пространством, которая имеет максимальный объем 4 Гбайта. В данное адресное пространство отображаются различные модули памяти и периферии.

Адрес	**Размер**	**Блок**		**Примечание**
Память программ				
0x0000_0000	1 Кбайт	BOOT ROM		Загрузочная программа
0x0800_0000	128 Кбайт	EEPROM		Область Flash-памяти программ с пользовательской программой
0x1000_0000	256 Мбайт	EXTERNAL BUS		Область доступа к внешней системной шине
Память данных				
0x2000_0000	32Кбайт	SYSTEM RAM		Область внутреннего ОЗУ
0x2200_0000	16 Мбайт	SYSTEM RAM Bit Band Region		Область битового доступа внутреннего ОЗУ
0x3000_0000	256 Мбайт	EXTERNAL BUS		Область доступа к внешней системной шине
Периферия				
0x4000_0000	1536 байт	0	CAN1	Регистры контроллера интерфейса CAN1
0x4000_8000	1536 байт	1	CAN2	Регистры контроллера интерфейса CAN2
0x4001_0000	904 байт	2	USB	Регистры контроллера интерфейса USB
0x4001_8000	20 байт	3	EEPROM_CNTRL	Регистры контроллера Flash-памяти программ
0x4002_0000	48 байт	4	RST_CLK	Регистры контроллера сигналов тактовой частоты
0x4002_8000	80 байт	5	DMA	Регистры контроллера прямого доступа в память
0x4003_0000	72 байт	6	UART1	Регистры контроллера интерфейса UART1
0x4003_8000	72 байт	7	UART2	Регистры контроллера интерфейса UART2
0x4004_0000	36 байт	8	SPI1	Регистры контроллера интерфейса SSP1
0x4005_0000	28 байт	10	I2C1	Регистры контроллера интерфейса I2C1
0x4005_8000	4 байт	11	POWER	Регистры детектора напряжения питания
0x4006_0000	12 байт	12	WWDT	Регистры контроллера сторожевого таймера WWDT
0x4006_8000	16 байт	13	IWDT	Регистры контроллера сторожевого таймера IWDT
0x4007_0000	128 байт	14	TIMER1	Регистры управления

				Таймер 1
0x4007_8000	128 байт	15	TIMER2	Регистры управления Таймер 2
0x4008_0000	128 байт	16	TIMER3	Регистры управления Таймер 3
0x4008_8000	48 байт	17	ADC	Регистры управления АЦП
0x4009_0000	12 байт	18	DAC	Регистры управления ЦАП
0x4009_8000	12 байт	19	COMP	Регистры управления Компаратора
0x400A_0000	36 байт	20	SPI2	Регистры контроллера интерфейса SSP1
0x400A_8000	32 байт	21	PORTA	Регистры управления порта A
0x400B_0000	32 байт	22	PORTB	Регистры управления порта B
0x400B_8000	32 байт	23	PORTC	Регистры управления порта C
0x400C_0000	32 байт	24	PORTD	Регистры управления порта D
0x400C_8000	32 байт	25	PORTE	Регистры управления порта E
0x400D_8000	84 байт	27	BKP	Регистры доступа и управления батарейным доменом
0x400E_8000	32 байт	29	PORTF	Регистры управления порта F
0x400F_0000	88 байт	30	EXT_BUS_CNTRL	Область доступа к внешней системной шине
0x4200_0000	16 Мбайт	PERIPHERAL Bit Band Region		Область битового доступа к регистрам периферии
0x5000_0000	256 Мбайт	EXTERNAL BUS		Область доступа к внешней системной шине
Внешняя системная шина				
0x6000_0000	1 Гбайт	EXTERNAL BUS		Область доступа к внешней системной шине
0xA000_0000	1 Гбайт	EXTERNAL BUS		Область доступа к внешней системной шине
SYSTEM REGION				
0xE000_0000	256 Мбайт			Системные регистры процессора ARM Cortex-M3

После включения питания и снятия внутренних и внешних сигналов сброса микроконтроллер начинает выполнять программу из загрузочной

области ПЗУ BOOT ROM. В загрузочной программе микроконтроллер определяет, в каком из режимов он будет функционировать, и переходит в этот режим.

1.1 Описание основных регистров микроконтроллера

1.1.1 Порты ввода/вывод

Микроконтроллер имеет 6 портов ввода/вывода. Порты 16-разрядные, и их выводы мультиплексируются между различными функциональными блоками, управление для каждого вывода отдельное. Для того, чтобы выводы порта перешли под управление того или иного периферийного блока, необходимо задать для нужных выводов выполняемую функцию и настройки.

Для работы с портами ввода/вывода используются библиотека MDR32F9Qx_port.h, которая описывает следующие регистры:

- `MDR_PORTA`
- `MDR_PORTB`
- `MDR_PORTC`
- `MDR_PORTD`
- `MDR_PORTE`
- `MDR_PORTF`

Для инициализации используется структура типа PORT_InitTypeDef с полями:

- `PORT_OE` – направление передачи данных
- `PORT_FUNC` – режим работы вывода порта
- `PORT_MODE` – режим работы контроллера
- `PORT_SPEED` – скорость фронта вывода
- `PORT_Pin` – выбор выводов для инициализации

Функциональное назначение портов приведено в таблице 120 и 121 стр.184 спецификации микроконтроллеров серии 1986BE9x.

1.1.2 Контроллер АЦП.

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 один из 16 внешних каналов;

– оцифровать значение встроенного датчика температуры;
– оцифровать значение встроенного источника опорного напряжения;
– осуществить автоматический опрос заданных каналов;
– выработать прерывание 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 частота процессора CPU_CLK, либо частота ADC_CLK, формируемая в блоке «Сигналы тактовой частоты».

Для работы с портами ввода/вывода используются библиотека `MDR32F9Qx_adc.h`, которая описывает регистры АЦП с помощью задания структур `ADC_InitTypeDef` для настройки преобразователя и `ADCx_InitTypeDef` для настройки канала преобразователя.

Структура `ADC_InitTypeDef` имеет следующие поля:
– `ADC_SynchronousMode` – выбор режима работы двух преобразователей;
– `ADC_StartDelay` – определяет задержку начала преобразований от старта системы [0:15];
– `ADC_TempSensor` – включение/выключение температурного датчика;
– `ADC_TempSensorAmplifier` – включение/выключение усилителя температурного датчика;
– `ADC_TempSensorConversion` – включение/выключение преобразования показаний от температурного датчика;
– `ADC_IntVRefConversion` – включение/выключение преобразования показаний опорного напряжения;
– `ADC_IntVRefTrimming` – определяет интервал считывания значений опорного напряжения;

Структура `ADCx_InitTypeDef` имеет следующие поля:
– `ADC_ClockSource` – указывает источник тактирующего сигнала;
– `ADC_SamplingMode` – задает режим считывания показаний;
– `ADC_ChannelSwitching` – включение/выключение возможности переключения каналов АЦП;
– `ADC_ChannelNumber` – номер канала;
– `ADC_Channels` – маска номеров каналов;
– `ADC_LevelControl` – включение/выключение слежения за уровнем АЦП;
– `ADC_LowLevel` – значение нижнего уровня АЦП;
– `ADC_HighLevel` – значение верхнего уровня АЦП;
– `ADC_VRefSource` – определяет источник питания АЦП;

– ADC_IntVRefSource – определяет тип напряжения источника питания АЦП;
– ADC_Prescaler – задает параметры предусилителя;
– ADC_DelayGo – задержка начала преобразований в последовательном режиме;

Подробное описание регистров блока контроллера АЦП приведено в таблице 292 стр.318 спецификации микроконтроллеров серии 1986BE9x.

1.1.3 Контроллер ЦАП.

В микроконтроллере реализовано два ЦАП. Для включения ЦАП необходимо чтобы бит Cfg_ON_DACx был установлен в 1, используемые выводы ЦАП порта E были сконфигурированы как аналоговые и были отключены какие-либо внутренние подтяжки. Оба ЦАП могут работать независимо или совместно. При независимой работе ЦАП (бит Cfg_SYNC_A=0) после записи данных в регистр данных DACx_DATA на выходе DACx_OUT формируется уровень напряжения, соответствующий записанному значению. При синхронной работе (бит Cfg_SYNC_A=1) данные обоих ЦАП могут быть обновлены одной записью в один из регистров DACx_DATA. ЦАП может работать от внутренней опоры Cfg_M_REFx=0, тогда ЦАП формирует выходной сигнал в диапазоне от 0 до напряжения питания AUCC. В режиме работы с внешней опорой Cfg_M_REFx=1 ЦАП формирует выходное напряжение в диапазоне от 0 до значения DACx_REF.

Для работы с аналоговыми портами ввода/вывода используется библиотека MDR32F9Qx_dac.h. Для работы ЦАП достаточно сконфигурировать соответствующий вывод на работу в аналоговом режиме и использовать функцию DAC1_Init или DAC2_Init для указания источника опорного напряжения.

Описание регистров блока контроллера АЦП приведено в таблице 307 стр.326 спецификации микроконтроллеров серии 1986BE9x.

1.1.4 Контроллер интерфейсаI2C

I2C является двухпроводным, двунаправленным последовательным каналом связи с простым и эффективным методом обмена данными между устройствами. Интерфейс применяется, когда надо организовать обмен на коротком расстоянии между несколькими устройствами. Стандарт интерфейса I2C является многомастерным с обнаружением коллизий и арбитражем, исключающим потерю данных при обмене, когда два или более мастера пытаются осуществить передачу одновременно.

Интерфейс работает на 3 скоростях:
– нормальная: 100 Kbps (DIV=150 при HCLK=80МГц);

– быстрая: 400 Kbps (DIV=25 при HCLK=80МГц);
– очень быстрая: 1 Mbps (DIV=1 при HCLK=80МГц).

I2C системы используют последовательную линию данных SDA и линию тактового сигнала SCL. Все устройства, подсоединенные к этим двум линиям, должны работать в режиме открытого стока, обеспечивая тем самым создание на линии «проводного И» за счет внешних резисторов подтяжки обоих линий к питанию.

Передача данных между мастером и ведомым осуществляется по линии SDA и синхронизируется по линии SCL. После завершения передачи информации осуществляется передача в обратную сторону одного бита подтверждения. Каждый принимаемый бит фиксируется принимающей стороной при высоком уровне SCL и может изменяться передатчиком при низком уровне. Изменение линии SDA при высоком уровне SCL является командным состоянием (см. «Сигнал START» и «Сигнал STOP»).

Для работы с шиной I2C используется библиотека `MDR32F9Qx_i2c.h`, которая описывает тип данных, `MDR_I2C_TypeDef`. Тип данных `MDR_I2C_TypeDef` имеет следующие поля:

– `PRL` – младшая часть делителя частоты;
– `PRH` – старшая часть делителя частоты;
– `CTR`– управление контроллером I2C;
– `RXD` – прочитанные данные;
– `STA` – статус;
– `TXD` – данные для записи;
– `CMD` – команда в шину данных.

Описание работы протокола I2Cприведено на стр.334 спецификации микроконтроллеров серии 1986BE9x.

Описание регистров блока контроллера I2Cприведено в таблице 322 стр.336спецификации микроконтроллеров серии 1986BE9x.

1.1.5 Контроллер интерфейса синхронной последовательной связи

Модуль порта синхронной последовательной связи (SSP – Synchronous Serial Port) выполняет функции интерфейса последовательной синхронной связи в режиме ведущего и ведомого устройства и обеспечивает обмен данными с подключенным ведомым или ведущим периферийным устройством в соответствии с одним из протоколов:

– интерфейс SPI фирмы Motorola;
– интерфейс SSI фирмы Texas Instruments;
– интерфейс Microwire фирмы National Semiconductor.

Как в ведущем, так и в ведомом режиме работы модуль SSP обеспечивает:

– преобразование данных, размещенных во внутреннем буфере FIFO передатчика (восемь 16-разрядных ячеек данных) из параллельного в последовательный формат;
– преобразование данных из последовательного в параллельный формат и их запись в аналогичный буфер FIFO приемника (восемь 16-разрядный ячеек данных).

Модуль формирует сигналы прерываний по следующим событиям:
– необходимость обслуживания буферов FIFO приемника и передатчика;
– переполнение буфера FIFO приемника;
– наличие данных в буфере FIFO приемника по истечении времени таймаута.

Модуль порта синхронной последовательной связи обладает следующими характеристиками:
– может функционировать как в ведущем, так и в ведомом режиме;
– программное управление скоростью обмена;
– состоит из независимых буферов приема и передачи (8 ячеек по 16 бит) с организацией доступа типа FIFO (First In First Out – первый вошел, первый вышел);
– программный выбор одного из интерфейсов обмена: SPI, Microwire, SSI;
– программируемая длительность информационного кадра от 4 до 16 бит;
– независимое маскирование прерываний от буфера FIFO передатчика, буфера FIFO приемника, а также по переполнению буфера приемника;
– доступна возможность тестирования по шлейфу, соединяющему вход с выходом;
– поддержка прямого доступа к памяти (DMA).

Для работы с контроллером SSP используется библиотека `MDR32F9Qx_ssp.h`, которая описывает структуру `MDR_SSP_TypeDef`. Структура `MDR_SSP_TypeDef` имеет следующие поля:
– `CR0` – управления блоками модуля SSP;
– `CR1` – управления блоками модуля SSP;
– `DR` – чтения принятых и запись передаваемых данных;
– `SR` – содержит информацию о состоянии буферов FIFO приемника и передатчика и занятости модуля SSP;
– `CPSR` – делитель тактовой частоты;
– `IMSC` – установка маски прерывания;
– `RIS` – текущее состояние прерываний;
– `MIS` – маскированое состояние прерываний;
– `ICR` – регистр сброса прерываний;

– DMACR – регистр управления прямым доступом к памяти.

Описание модуля порта синхронной последовательной связи представлено на стр.342спецификации микроконтроллеров серии 1986BE9x.

1.1.6 Контроллер универсального асинхронного приемопередатчика

Модуль универсального асинхронного приемопередатчика (UART – Universal Synchronous Asynchronous Receiver Transmitter) представляет собой периферийное устройство микроконтроллера.

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.

Имеется возможность программного управления скоростью обмена. Обеспечивается возможность деления тактовой частоты опорного генератора в диапазоне (1x16 – 65535x16).

Следующие ключевые параметры могут быть заданы программно:

- скорость передачи данных – целая и дробная часть числа;
- количество бит данных;
- количество стоповых бит;
- режим контроля четности;
- разрешение или запрет использования буферов FIFO (глубина очереди данных – 32 элемента или один элемент, соответственно);
- порог срабатывания прерывания по заполнению буферов FIFO (1/8, 1/4, 1/2, 3/4 и 7/8);
- частота внутреннего тактового генератора (номинальное значение - 1.8432 МГц) может быть задана в диапазоне 1.42 – 2.12 МГц для обеспечения возможности формирования бит данных с укороченной длительностью в режиме пониженного энергопотребления;
- режим аппаратного управления потоком данных.

Устройство выполняет следующие функции:

- преобразование данных, полученных от периферийного устройства, из последовательной в параллельную форму;
- преобразование данных, передаваемых на периферийное устройство, из параллельной и последовательную форму.

Процессор читает и записывает данные, а также управляющую информацию и информацию о состоянии модуля. Прием и передача данных

буферизуются с помощью внутренней памяти FIFO, позволяющей сохранить до 16 байтов независимо для режимов приема и передачи.

Для работы с контроллером асинхронного приемо-передатчика используется библиотека MDR32F9Qx_uart.h, которая описывает структуру UART_InitTypeDef.

Структура UART_InitTypeDef имеет следующие поля:

- UART_BaudRate – скорость передачи данных;
- UART_WordLength – длина слова в пакете;
- UART_StopBits – количество стоп битов;
- UART_Parity – контроль четности;
- UART_FIFOMode – определяет режим работы FIFO;
- UART_HardwareFlowControl – включает/выключает аппаратный контроль потока.

При работе с UART контроллером применяются функции записи данных UART_SendDataи чтенияUART_ReceiveData.

Описание модуля универсального асинхронного приемопередатчика представлено на стр.373 спецификации микроконтроллеров серии 1986BE9x.

1.1.7 Таймеры общего назначения

Все блоки таймеров 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 имеют 4 канала схем захвата и ШИМ с функциями формирования «мертвой зоны» и аппаратной блокировки. Каждый из таймеров может генерировать прерывания и запросы DMA.

Для работы с таймерами используется структура TIMER_CntInitTypeDef с полями:

- TIMER_Prescaler – значение величины предделителя
- TIMER_Period – период таймера
- TIMER_CounterMode – режим счета
- TIMER_CounterDirection – направление счета

– TIMER_EventSource – источник событий для таймера
– TIMER_FilterSampling – указывает фильтр событий
– TIMER_ARR_UpdateMode – режим сброса счетчика
– TIMER_ETR_FilterConf – задает параметры выхода ETR
– TIMER_ETR_Prescaler – задает параметры предделителя фильтра выхода ETR
– TIMER_ETR_Polarity – задает полярностьвыхода ETR
– TIMER_BRK_Polarity – задает полярностьвыхода BRK

Структура TIMER_ChnInitTypeDefсполями:

– TIMER_CH_Mode – задает режим работы таймера
– TIMER_CH_REF_Format – формат выработки сигнала REFв режима ШИМ
– TIMER_CH_Number – номер канала таймера

Структура TIMER_ChnOutInitTypeDef сполями:

– TIMER_CH_DirOut_Polarity – полярность выхода CHx таймера
– TIMER_CH_DirOut_Source – задает сигнал на выходе CHx таймера
– TIMER_CH_DirOut_Mode – задает сигнал на выходе CHx таймера
– TIMER_CH_NegOut_Polarity – полярность инверсного выхода CHx таймера
– TIMER_CH_NegOut_Source – задает сигнал на инверсном выходе CHx таймера
– TIMER_CH_NegOut_Mode – задает сигнал на инверсном выходе CHx таймера
– TIMER_CH_Number – номер канала

Описание работы таймеров общего назначения представлено на стр.274спецификации микроконтроллеров серии 1986BE9x.

1.1.8 Контроллер прерываний

Векторный контроллер прерываний с возможностью вложения (NVIC – Nested Vectored Interrupt Controller) обеспечивает:

– программное задание уровня приоритета в диапазоне от 0 до 15 независимо каждому прерыванию. Более высокое значение соответствует меньшему приоритету, таким образом, уровень 0 отвечает наивысшему приоритету прерывания;
– срабатывание сигнала прерывания по импульсу и по уровню;
– динамическое изменение приоритета прерываний;
– разделение исключений по группам с одинаковым приоритетом и по подгруппам внутри одной группы;
– передача управления из одного обработчика исключения на другой без восстановления контекста.

Процессор автоматически сохраняет в стеке свое состояние (контекст) по входу в обработчик прерывания и восстанавливает его по завершению обработчика без необходимости непосредственного программирования этих операций. Это обеспечивает обработку исключительных ситуаций с малой задержкой.

Поскольку прерывание может возникнуть при выполнении любой произвольной команды фона, её адрес запоминается в так называемом программном стеке. После чего выполнение предается на часть программы, специально написанную разработчиком для реакции на событие, вызвавшее данное прерывание. Эта небольшая часть программы называется обработчиком прерывания. Когда обработчик будет выполнен до конца, программа, воспользовавшись адресом, сохранённым в программном стеке, вернётся в то место, откуда была вызвана для обработки данного прерывания.

Для использования вектора прерывания необходимо:

- Разрешить использование прерываний в программе;
- Разрешить вызов интересующего нас прерывания специальным битом в соответствующем регистре;
- Создать условия для возникновения прерывания, например, если это переполнение таймера, то запустить его;
- Разместить в программе обработчик прерывания, оформив его в соответствии с требованиями компилятора.

Для разрешения использования прерываний в программе используется функция `void NVIC_EnableIRQ(IRQn_t IRQn)`, в которую передается имя прерывания.

Для разрешения вызовов прерываний у каждого модуля описан свой регистр, например для таймера TIMER_ITConfig (MDR_TIMER1, TIMER_STATUS_CNT_ARR, ENABLE);

Описание прерываний происходит в файле, формат которого предоставляет производитель микроконтроллера MDR32F9Qx_it.c и соответствующем ему заголовочном файле MDR32F9Qx_it.h.

2 Компилятор и программная среда разработки

В настоящее время программа для микроконтроллера пишется на одном из языков программирования в виде текстового файла. Это означает, что для написания программы можно воспользоваться любым текстовым редактором. Специальная программа-транслятор преобразует исходный текст программы в машинные коды, понятные микропроцессору, этот процесс называется компиляцией. В результате компиляции будет создан так называемый hex-файл, предназначенный для загрузки в память микроконтроллера. Если открыть hex-файл обычным блокнотом, то он будет состоять из строк шестнадцатеричных цифр, поэтому файл и называют "hex" – от английского слова "hexadecimal" – "шестнадцатеричная система счисления".

Для облегчения процесса разработки программы часто используются интегрированная среда программирования, в состав которой включается определенный набор программных средств: редактор исходного текста, трансляторы с выбранного языка программирования, редакторы связей, загрузчики, отладчики и так далее. При разработке программы, помимо файла с самим текстом программы, среда разработки использует множество вспомогательных файлов. Для того, чтобы все эти файлы были должным образом взаимосвязаны, создается программный проект.

Для разработки программного обеспечения для микроконтроллеров серии 1986BE9x может быть использована одна из следующих сред разработки:

- Keil uVision - компилятор C/C++, ассемблер; отладчик; трассировка; внутрисхемное программирование; USB JTAG адаптер ULINK2; бесплатная версия с ограничением размера кода программы в 32 Кбайт.
- CodeMaster-ARM Фитон, компилятор C, ассемблер; отладчик; трассировка; внутрисхемное программирование; USB JTAG адаптер JEM-ARM-V2; бесплатная версия с ограничением размера кода программы в 8 Кбайт
- IAR Embedded Workbench - компилятор C/C++, ассемблер; отладчик; трассировка; внутрисхемное программирование; USB JTAG адаптер J-LINK.

В рамках данного курса студентам будет предложено ознакомиться со средой программирования Keil uVisionv5.

2.1 Установка и первый запуск интегрированной среды разработки Keil uVision

Среду программирования Keil uVision можно найти на сайте разработчика по адресу https://www.keil.com/download/product/. Здесь необходимо скачать и установить MDK-ARM (требуется регистрация).

Для поддержки микроконтроллеров серии 1986BE9x необходимо скачать и установить последнюю версию библиотеки стандартной периферии от ПКК Миландр «Milandr.MDR1986BExx.1.4.0.pack».

Также требуется установить драйверы JTAG-отладчика Segger Jlink.

После установки всех компонентов среды программирования Keil uVision можно приступать к созданию первого проекта. Для этого необходимо создать новый проект Project – New uVision Project… (рисунок 2) и выбираем директорию для нового проекта (в названии пути к директории не должно быть русских символов).

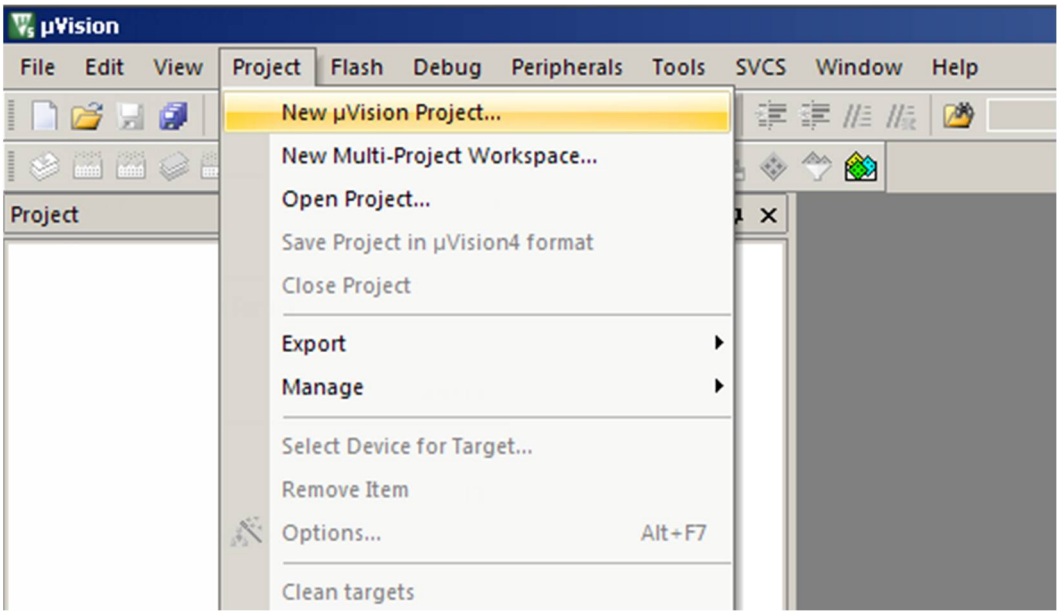


Рисунок 2 –Создание нового проекта.

Затем будет предложено выбрать устройство для которого будет написана программа. Выбираем микроконтроллер MDR1986BE93(Milandr – Milandr – Cortex-M3- MDR1986BE93).

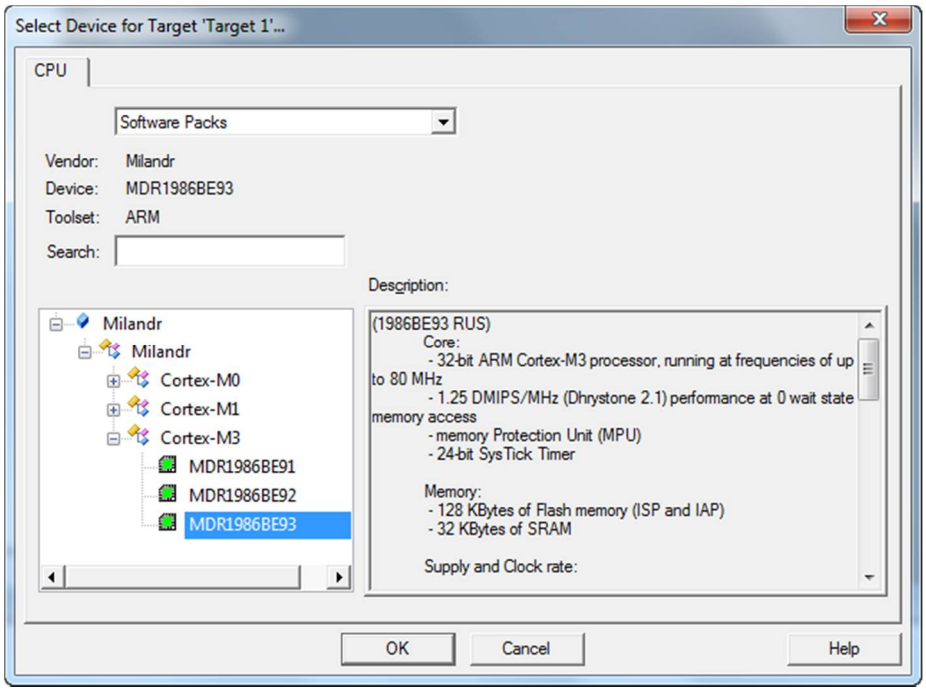


Рисунок 3 –Создание нового проекта.

После выбора целевого микроконтроллера необходимо определить периферию и окружение компиляции. 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).

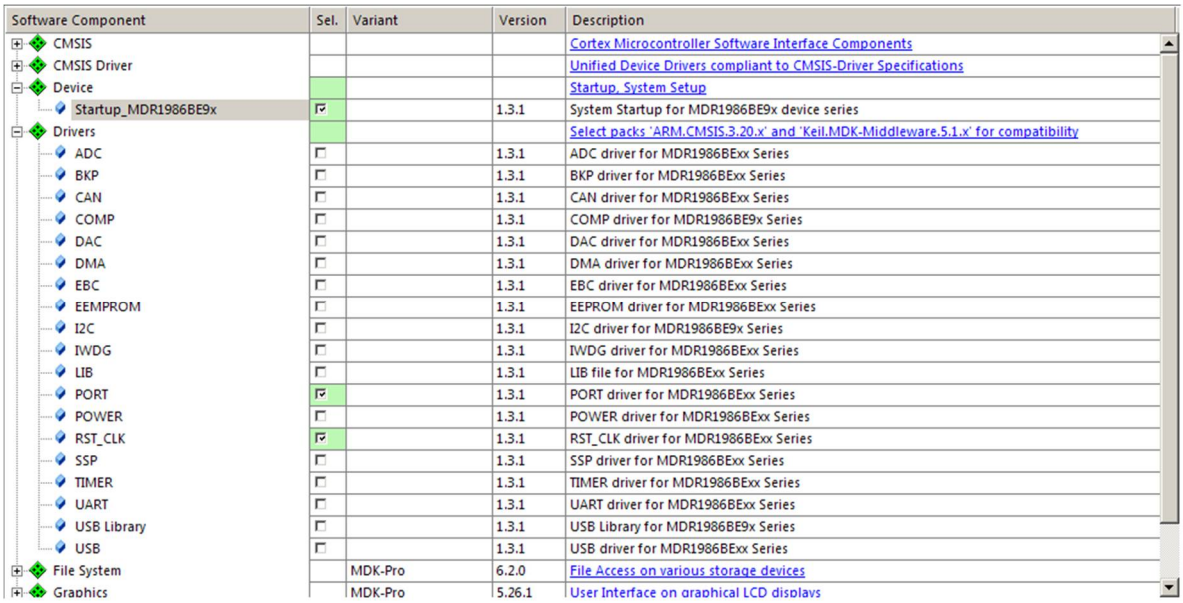


Рисунок 4 –Окно выбора окружения компиляции

Затем в дереве проекта щелкнуть правой кнопкой мыши на «Source group 1» и выбрать «add new item to Group 'Source group 1'» (рис. 5). 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.

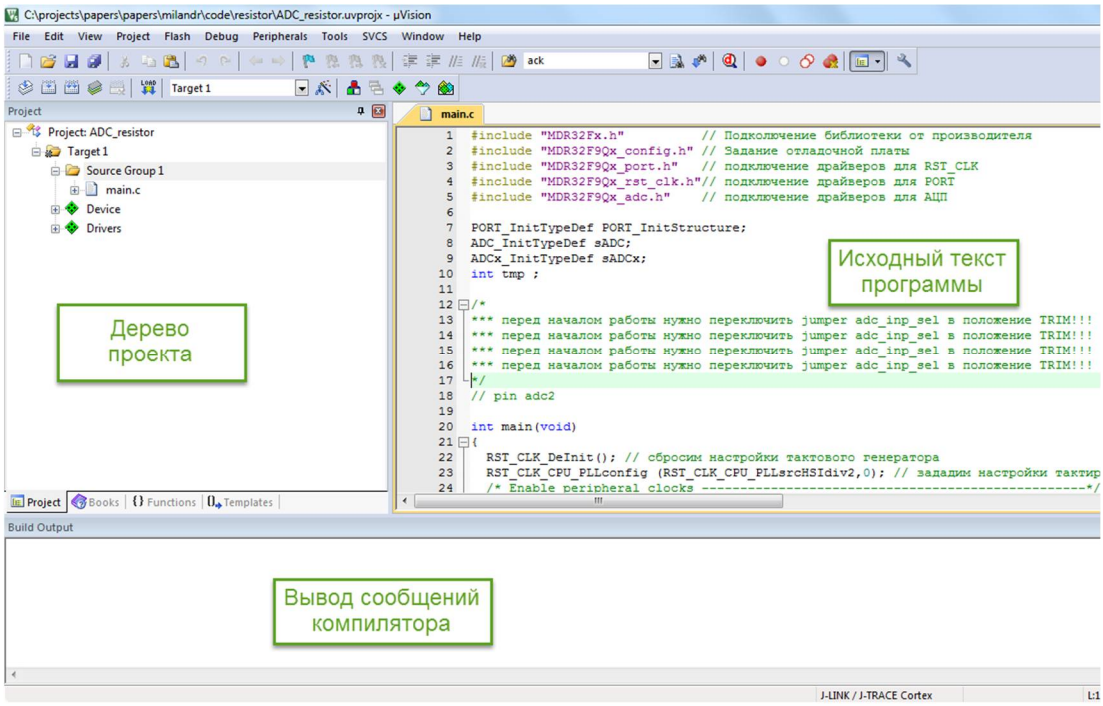


Рисунок 5 –Главное окно программы Keil uVision5

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» (рисунок 6).

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 7).

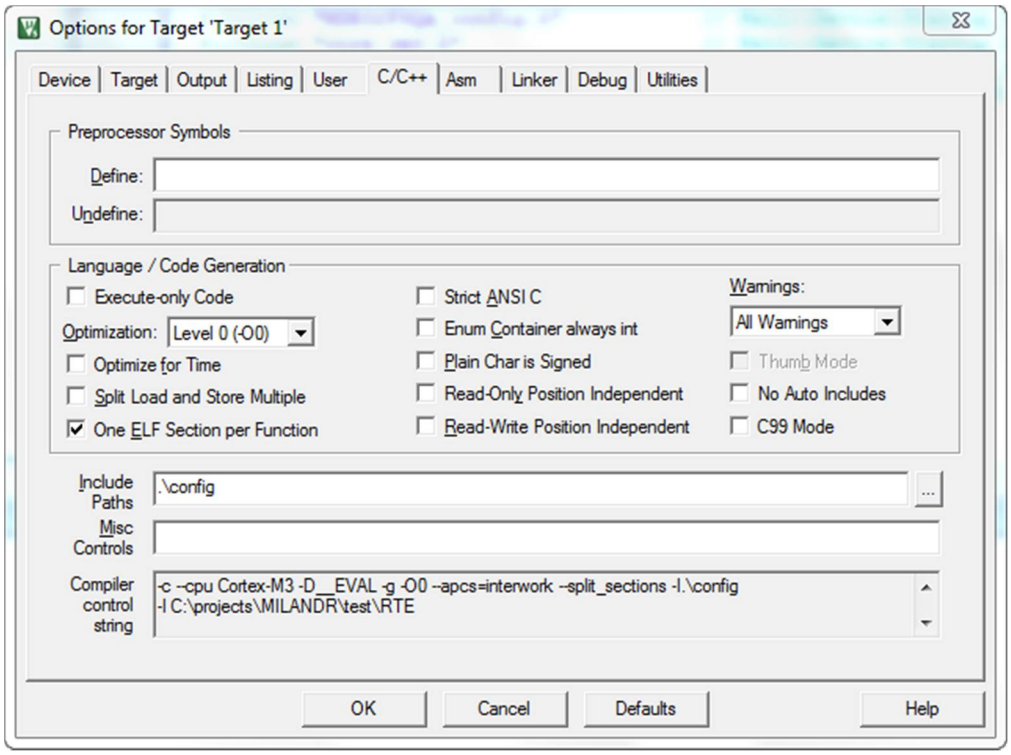


Рисунок 6 – Окно настроек Target 1, вкладка C/C++

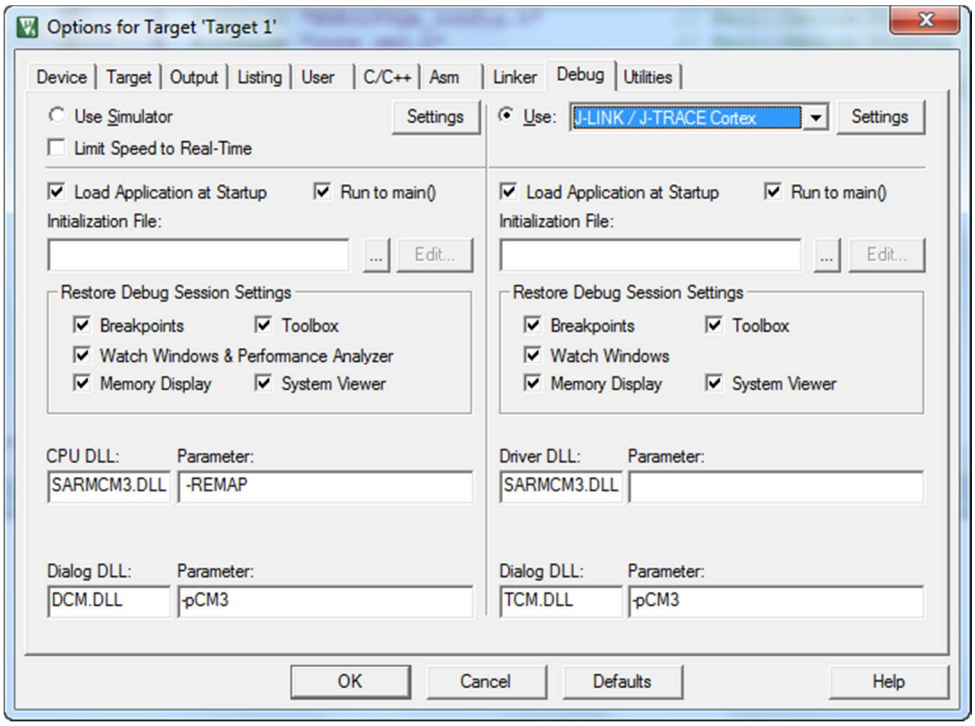


Рисунок 7 – Окно настроек Target 1, вкладка Debug

В выпадающем списке выбрать «J-LINK/J-TRACE Cortex». Затем нажать кнопку «Settings». В выпадающем меню «Port» необходимо выбрать режим «SW» (рисунок 8), и задать скорость передачи данных («Max Clock»)

не более 2 MHz. Удостовериться, что в поле «SW Device» присутствует устройство ARMCoreSightSW-DP.

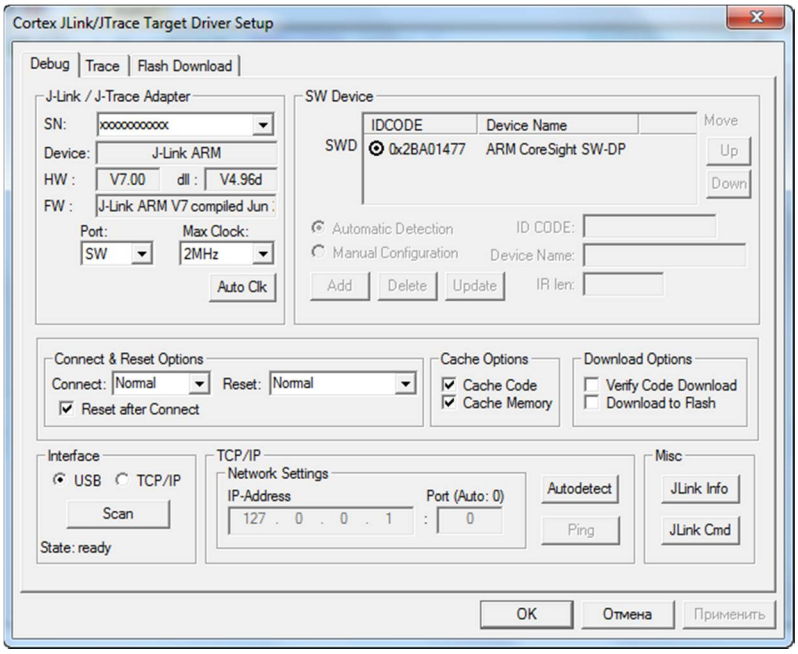


Рисунок 8 – Окно настроек отладчика, вкладка Debug

Далее необходимо задать порядок загрузки программы в память микроконтроллера. Для этого перейти на вкладку «Flash Download» (рисунок 9), 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».

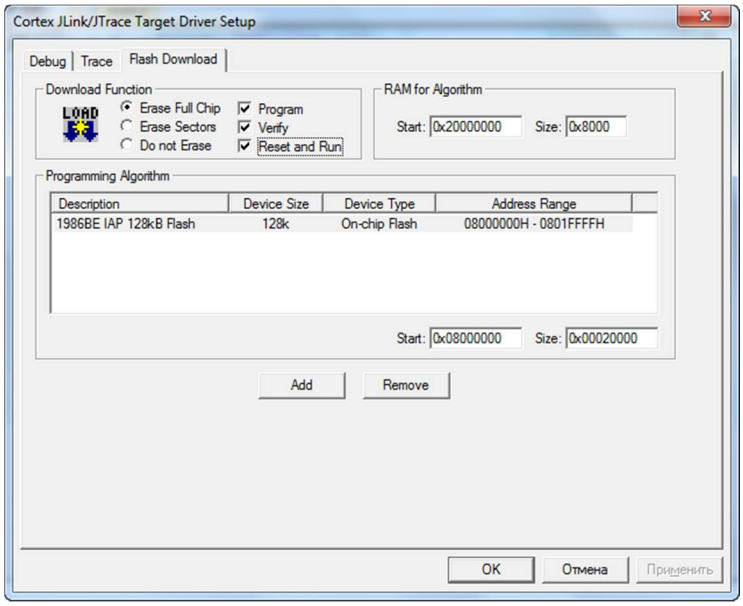


Рисунок 9 –Окно настроек отладчика, вкладка Flash Download

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 (рисунок 10).

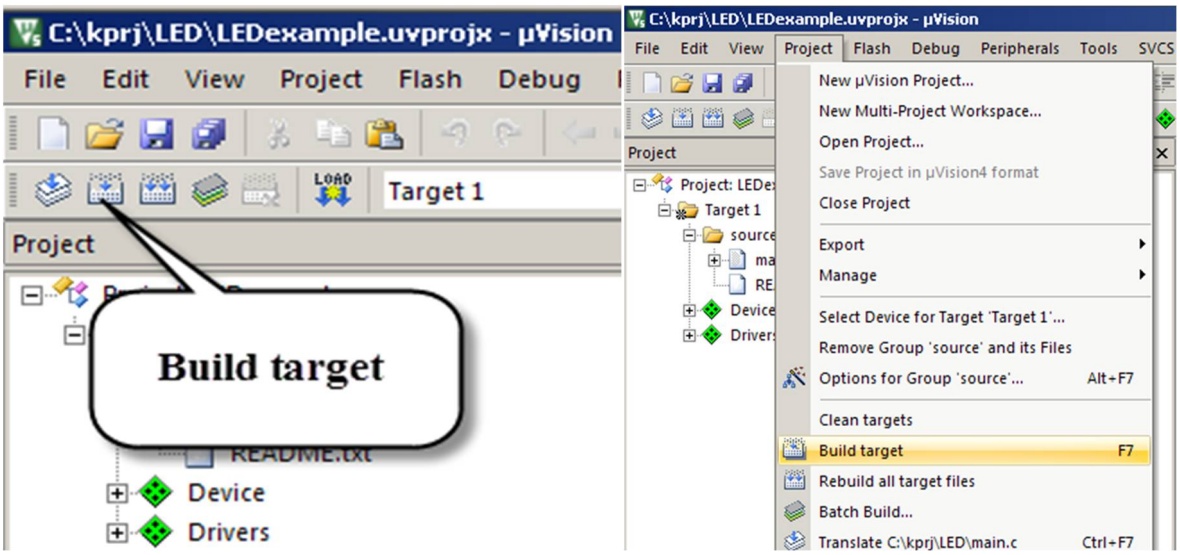


Рисунок 10 –Компиляция кода программы

Скомпилированная без ошибок программа может быть загружена в память микроконтроллера. Для этого нужно выбрать «Flash-Download» или нажать на соответствующую кнопку на панели инструментов (рисунок 11).

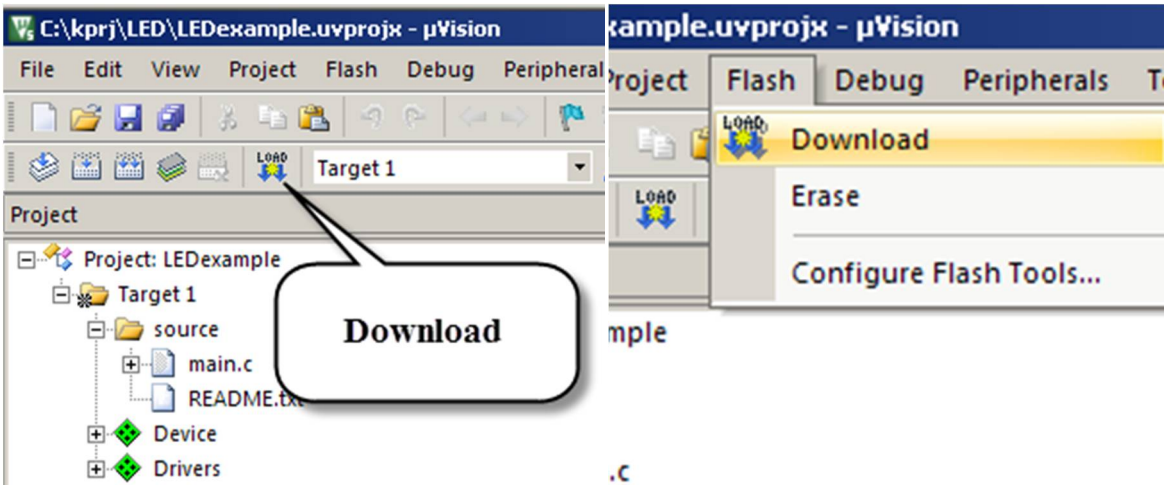


Рисунок 11 –Загрузка скомпилированного кода программы в память микроконтроллера

Для отладки программы в среде программирования Keil uVision могут быть использованы средства JTAG-отладчика. Для этого необходимо нажать на кнопку «start/stop debug» (рисунок 12). Режим отладки предоставляет возможность ставить точки остановки выполнения программы на микроконтроллере.

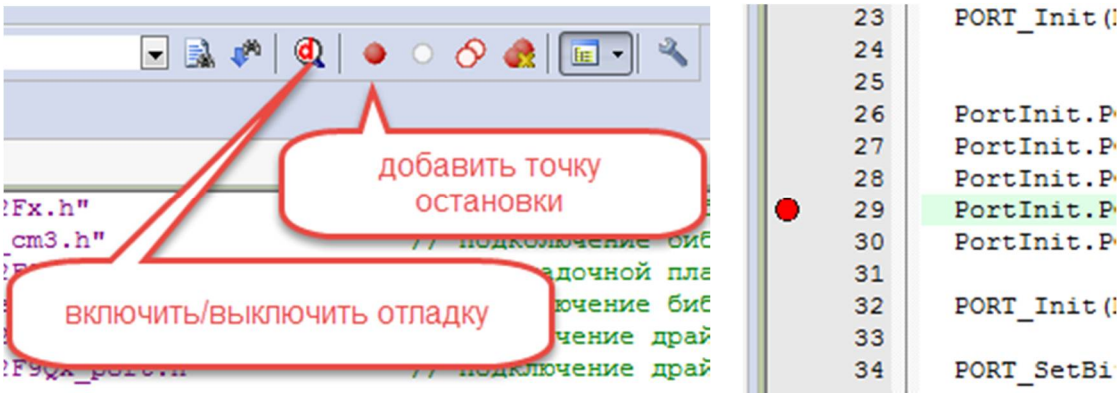


Рисунок 12 –Использование режима отладки

После нажатия кнопки «start/stop debug» окно программы Keil uVision изменяется и принимает вид, показанный на рисунке13. При этом программа на микроконтроллере не выполняется до тех пор, пока пользователь не нажмет соответствующую кнопку.

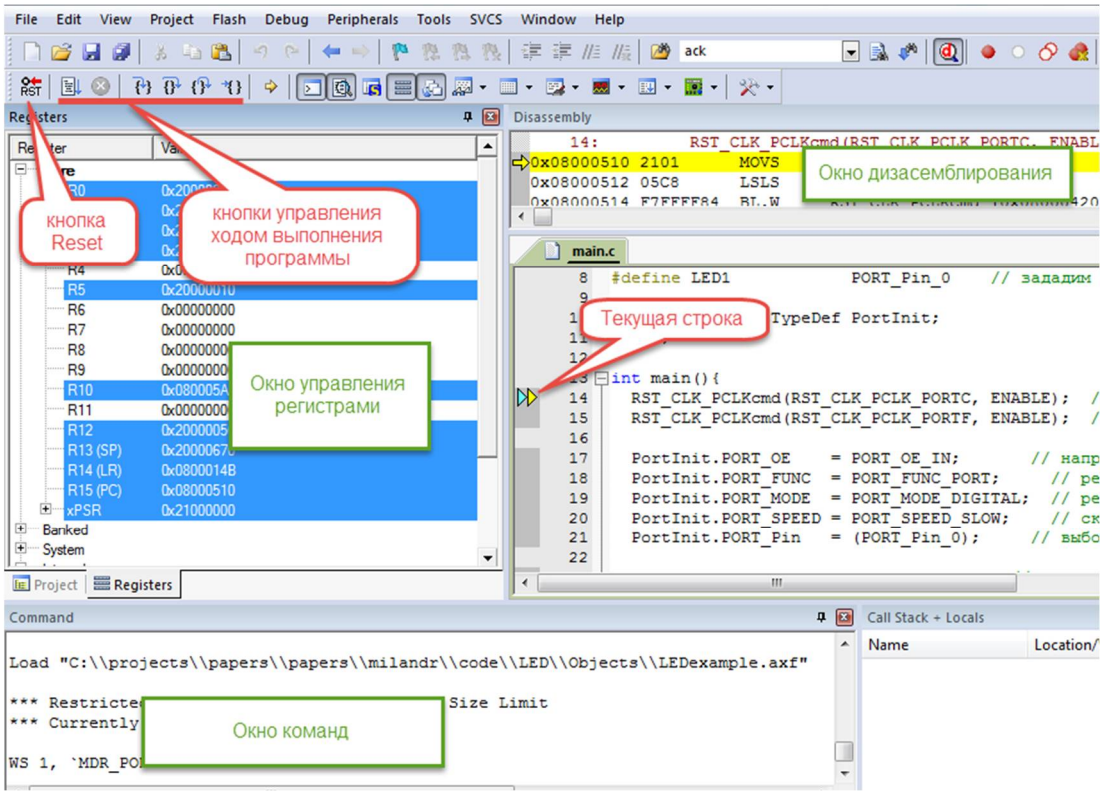


Рисунок 13 –Главное окно в режиме отладки

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».

3 Основы программирование на языке С

3.1 Вводные понятия

Язык программирования С – высокоуровневый абстрактный язык программирования. Первая версия языка С была разработана в середине 60-ых годов для разработки операционной системы Unix в лаборатории Bell. Один из самых первых разработчиков этого языка, Кен Томсон, решил, что требуется язык для создания более сложных языков программирования. Он создал такой язык и назвал его «B». В процессе развития своего творения Томсон постоянно боролся с ограничением ресурсов памяти, что теперь очень похоже на встраиваемые системы. Деннис Ричи решил расширить язык «B» свойством генерировать малый по объему код, который сможет соперничать с кодом, написанным на ассемблере. В 1973 году важнейшие свойства этого нового языка «C» были получены.

Возрастающая популярность С заложена в его переносимости. Компиляторы С были созданы для многих, отчего его популярность еще больше выросла. Наиболее бурно С стал использоваться в 80 годах, когда стал основным языком для создания программ персональных компьютеров. Основной идеей языка С является работа с функциями, аргументы которых передаются как параметры.

Прежде всего, рассмотрим такие вводные понятия как комментарии, ключевые слова, идентификаторы, литералы, операторы и знаки пунктуации.

Комментарий – поясняющий текст, который при компиляции не учитывается. Комментарии бывают многострочными (начинаются с символов `/*` и заканчиваются символами `*/`) и однострочными (начинаются с символов `//`). Например:

```
/*Многострочный комментарий часто размещают в начале файла, где
он содержит имя автора и описание программы*/
#include "MDR32Fx.h" //Подключаем заголовочный файл MDR32Fx.h
```

Идентификатор – это последовательность букв, цифр и символов подчеркивания «_», которая не должна начинаться с цифры, используемая для именования различных программных элементов, – переменных, констант, функций, типов и т.д. Регистр букв имеет значение.

Ключевое слово - это зарезервированное слово четко определенного назначения. Ключевые слова не могут использоваться в качестве идентификаторов:

```
asm, auto; bit, bool, break; case, char, const, continue default,
defined, do, double; else, enum, explicit, extern; false, float,
for; goto; if, inline, int; long; register, return; short, signed,
sizeof, static, struct, switch; true, typedef; union, unsigned;
void; while.
```

Литерал – постоянное значение некоторого типа, используемое в выражениях. Примеры числовых литералов:

```
10 — число 10 в десятичной форме;
0xA — число 10 в шестнадцатеричной форме (префикс 0x);
0b1010 — число 10 в двоичной форме (префикс 0b);
012 — число 10 в восьмеричной форме (префикс 0);
10.5 — число с плавающей точкой;
105e-1 — число 10,5 в экспоненциальной форме;
10U — беззнаковая константа (суффикс U);
10L — знаковая константа (суффикс L).
```

Символьные литералы заключаются в одинарные кавычки, например, 'B', '*'. Для обозначения непечатаемых и специальных символов в литералах используются так называемые escape-последовательности:

```
'\a'— звуковой сигнал;
'\b' — клавиша <Backspace>;
'\f' — прогон листа;
'\n'— символ перевода строки
'\r'— возврат каретки;
'\t'— горизонтальная табуляция;
'\v' — вертикальная табуляция;
'\0'— нулевой символ;
'\\'— обратная косая;
'\''— апостроф.
```

Кроме того, любой символ можно представить с помощью литерала по его ASCII-коду, например, литерал '\t' равнозначен '\x09'.

Строковые литералы ограничиваются двойными кавычками, а в памяти хранятся как последовательности символов, заканчивающиеся нулевым символом '\0'. Специальные символы внутри строки должны предваряться обратной косой ("\"). Примеры строковых литералов:

```
"" — пустая строка: один символ '\0';
"B" — два символа: 'B' и '\0';
"A\tB\n" — пять символов: 'A', табуляция, 'B', перевод строки,
'\0'.
```

Оператор – это символ, указывающий компилятору, какие действия выполнить над операндами. Некоторые символы могут трактоваться по-разному в зависимости от контекста. Например, знак «-» может использоваться для изменения знака числа или в качестве оператора вычитания. Операторы, соединяющие операнды, представляют собой выражения. Выражения могут быть заключены в круглые скобки и отделяются друг от друга символом точки с запятой («;»).

Приоритет (по убыванию) выполнения операторов в выражениях языка С указан в таблице 2.

Таблица2 – Приоритет выполнения операторов в языке С

Уровень	Операторы	Категории	Описание
1	()		Круглые скобки
	[]		Элемент массива
	.	Доступ к данным	Обращение к элементу структуры, например PortInit.PORT_OE – обращение к регистру направления передачи данных
	->		Обращение к элементу структуры, определенной указателем, например pStruct ->x – элемент x структуры на которую указывает pStruct
	++,-- (постфиксы)	Арифметические	Операторы автоинкремента и автодекремента после того, как выражение, в котором задействованы соответствующие операнды, будет вычислено. Примеры: a = b++; равнозначно a = b; b = b+1; a = b --; равнозначно a = Ь; b = b-1;
2	++,-- (префиксы)		Операторы автоинкремента и автодекремента перед тем, как выражение, в котором задействованы соответствующие операнды, будет вычислено. Примеры: a = ++b; равнозначно b = b+1;a = b a = —b; равнозначно b = b-1;a = b;
	!	Логические	Логической (унарное) отрицание
	~	Поразрядные	Поразрядное отрицание
	&	Доступ к памяти	Адрес
3	+, -(унарные)	Арифметические	Изменение знака операнда
	*(унарный)	Доступ к данным	Разыменование указателя
4	*(бинарные)	Арифметические	Умножение
	/		Деление
	%		Остаток от деления
5	+, -(бинарные)		Сложение и вычитание
6	<<	Поразрядные	Поразрядный сдвиг влево
	>>		Поразрядный сдвиг вправо
7	<, >, <=, >=	Сравнения	Меньше, больше и т.д.
8	==, !=		Равно, не равно
9	&	Поразрядные	Поразрядное «И»
10	^		Поразрядное «Исключающее ИЛИ»
11	|		Поразрядное «ИЛИ»
12	&&	Логические	Логическое «И»
13	||		Логическое «ИЛИ»
14	=	Присваивание	Возможны также сочетания оператора присваивания с арифметическими и поразрядными операторами, например: a += b; равнозначно a = a + b; a*= b+c; равнозначно a =a* (b + c) ;

Когда в выражении используется оператор инкремента или декремента, то важно обращать внимание на то, где стоят два знака + или –: перед переменной или после переменной:

```
a=4, b=7;
a = b++; /* Эта строчка на С означает: взять значение переменной
b присвоить его переменной a затем добавить 1 к переменной b и
сохранить результат в b. Теперь a = 7; b = 8  */

a=4, b=7;
a = ++b; /* Эта строчка на С означает: взять значение переменной
b затем добавить к нему 1 и сохранить результат в b и этот же
результат присвоить переменной a. Теперь a = 8, b = 8.*/
```

3.2 Структура программы на языке С

Структура программы написанной на языке С имеет следующий вид:

```
//Директивы препроцессора
#include "MDR32Fx.h"
#define BLINK_DELAY 20000
…
//объявления глобальных типов, переменных, констант
uint32_t cnt;
float x, y;
…
//Пользовательские функции
Function1
{
  //Объявления локальных типов, переменных и констант
      …
  //Операторы
      …
}
…
Function N
{
  //Объявления локальных типов, переменных и констант
      …
  //Операторы
      …
}

//Главная функция программы
int main (void)
{
  //Объявления локальных типов, переменных и констант
      …
  //Операторы
      …
}
```

Программы обычно начинаются с директив процессора (начинаются с символа «#»), которые не являются конструкциями языка С и обрабатываются до фактической компиляции программы. Их смысл – подстановка некоторого кода в программу. Так, к примеру, очень часто

используется директива `#include`, которая включает в файл с исходным кодом программы текст внешнего заголовочного файла (с расширением `.h`). Заголовочные файлы содержат определения глобальных типов, переменных констант и функций.

Далее в программе объявляются глобальные переменные, типы, константы. Обращаться к таким переменным можно из любого места программы.

Затем описываются функции, входящие в программу. **Функция** представляет собой «контейнер», в котором выполняется некоторый фрагмент программного кода. Использование функций упрощает написание и отладку программ, поскольку в них удобно размещать повторяющиеся группы операторов. Любая программа на языке С содержит главную функцию под названием `main()`. Эта функция выполняется первой при запуске программы.

Пользовательские функции определяются после директив препроцессора и глобальных объявлений типов, переменных и констант в файле с исходным кодом или в заголовочном файле.

При этом используется следующий синтаксис:

```
Тип_возвращаемого_значения  Имя_функции (Список_параметров)
{
//Тело функции
}
```

В качестве имени функции можно использовать любое слово, отражающее выполняемое функцией действие, при этом имя не должно начинаться с цифры.

Тело функции представляет собой набор команд и операторов, расположенных между открывающейся и соответствующей закрывающейся фигурными скобками.

В функцию могут передаваться параметры (список параметров), один или несколько – это идентификаторы, которые могут использоваться внутри функции. Вместо них подставляются соответствующие значения, указанные при вызове функции (если в функцию передается более одного параметра, то они отделяются друг от друга запятыми, как при объявлении, так и при вызове).

Значения, переданные в функцию, фактически не изменяются, а просто копируются в параметры, которые в этом смысле выполняют роль локальных переменных. При этом следует следить за тем, чтобы тип передаваемых значений соответствовал типу параметров, объявленных в заголовке функции.

Возвращать функция может только один параметр. В качестве возвращаемого значения может использоваться ключевое слово `void`. Это означает, что функция не возвращает никакого значения. Если функция

предназначена для возврата значения некоторого типа, то для этого в ее теле используют ключевое слово `return`, после которого (через пробел) указывают возвращаемое значение. При этом все операторы после слова returnигнорируются, и происходит возврат в вызывающую функцию. Например:

```
int power3(int n)
{
     return n*n*n;
}
void main()
{
     int x;
     x = power3(2); //x = 8
}
```

Слово `return` может также использоваться без указания возвращаемого выражения. В этом случае оно просто означает выход из функции.

3.3 Типы данных, переменные, константы, структуры

Тип данных определяет диапазон допустимых значений и соответственно - пространство, отводимое в памяти данных, для переменных, констант и результатов, возвращаемых функциями. К стандартным типам данных языка С относятся: `signed char; unsigned char; int; unsigned int; long; unsigned long; float; long long; unsigned long long`.

Таблица3 – Стандартные типы данных языка С

Typedef	Длина (биты | байты)	Диапазон значений
bit	1	0,1
char	8 | 1	-128...127
signed char		
unsigned char		0…255
int	16 | 2	-32768...32767
signed int		
short int		
signed short int		
unsigned int		0…65535
unsigned short int		
long int	32 | 4	-2 147 483 648... 2 147 483 647
signed long int		
unsigned long int		0...4 294 967 295
float	32 | 4	-3.4E38 .. 3.4E38

double	64 | 8	1.7E-308...1.7E+308
long double	80 | 10	3.4E-4932...3.4E+4932

Переменная – это именованная величина определенного типа, которая может изменяться в ходе выполнения программы. Для объявления переменных (т.е., выделения для них памяти) в программе на С используется следующая конструкция:

```
тип_переменной  идентификатор1,идентификатор2, . . .;
```

например:

```
int i;//Объявление целочисленной переменной i
char c1,c2; //Объявление символьных переменных c1 и c2
```

Для доступа к переменной в программе используется соответствующий идентификатор (обязательно после объявления переменной). Значения, присваиваемые переменной, должны соответствовать ей по типу (или правилам приведения типов, рассматриваемым ниже).
Примеры:

```
i = 2;//Ошибка! Переменная i еще не объявлена
int i;//Объявление целочисленной переменной i
float f; //Объявление вещественной переменной f
i = 2; //Переменнойiприсвоенозначение2
f= 3.3 //Переменной f присвоено значение 3,3
f=  i; //Переменной f присвоено значение переменной i
//( в данном случае будет выполнено автоматическое приведение
типов, т.е. f = 2.0)
```

По области видимости переменные могут быть глобальными и локальными. К глобальным переменным имеют доступ все функции программы. Такие переменные объявляются в программе перед объявлением всех функций. К локальным переменным имеет доступ только та функция, в которой они объявлены.

Имена переменных обладают определенной областью видимости, которая подразумевает, что компилятор использует переменные в соответствии с тем, где они находятся. Имена переменных, объявленных внутри функции, имеют область видимости, ограниченную конкретной функцией. Например, в нескольких функциях можно объявить переменную `int i`, которая в каждой функции не будет иметь никакой связи с аналогичными переменными в других функциях. Точно так же и переменная, объявленная внутри блока (ограниченного фигурными скобками {…}), остается локальной по отношению к этому блоку.

Глобальные переменные имеют область видимости, которая начинается от места их объявления и продолжается до конца программного файла.

Константа – это именованная величина определенного типа, которая, в отличие от переменной, не может изменяться в ходе выполнения программы, а имеет конкретное значение, определенное в момент объявления. Для объявления констант в программе на С используется следующая конструкция:

```
const тип_константы идентификатор  = значение;
```

Например:

```
Const int i=10;//Объявление целочисленной константы i.
```

Величина, объявленная как константа, будет размещена компилятором в памяти RAM, то есть в той же области памяти, где хранятся переменные.
Для доступа к константе используется ее идентификатор.
Примеры:

```
c ='A'; //Ошибка! Константа c еще не объявлена
int i; //Объявление целочисленной переменной i
const c ='A';//Объявление константы c
i = 2; //Переменной i присвоено значение 2
c = i; //Ошибка! Попытка присвоить значение константе
i = c; //Переменной i присвоено значение константы c
/* В данном случае будет выполнено автоматическое приведение типов,
т.е. i= 65 (ASCII-код символа 'A')*/
```

Приведение типов – это принудительное преобразование значения одного типа к другому, совместимому с исходным. Это важно при выполнении арифметических операций, когда полученные значения могут выходить за допустимые пределы.

Приведение типов бывает явным и неявным. Неявное приведение типов используется в операторах присваивания, когда компилятор сам выполняет необходимые преобразования без участия программиста.

Для явного приведения типа некоторой переменной перед ней следует указать в круглых скобках имя нового типа, например:

```
int X;
intY = 200;
char C = 30;
X = (int) C * 10 + Y; //Переменная C приведена к типу int
```

Если бы в этом примере не было выполнено явное приведение типов, то компилятор предположил бы, что выражение `C * 10`- это восьмиразрядное умножение (разрядности типа char) и вместо корректного

значения 300 (0x12C) в стек было бы помещено урезанное значение 44 (0x2C). Таким образом, в результате вычисления выражения `C * 10 +Y` переменной X было бы присвоено значение 640, а не корректное 3200. В результате приведения типа переменная C распознается компилятором как 16-тиразрядная, и описанной выше ошибки не возникает.

Оператор `sizeof` применяется для вычисления размера области памяти (в байтах), отводимой под некоторую переменную, результат выражения или тип. Например:

```
int a;
float f;
a = sizeof (int); //a = 2
a = sizeof (f); //a = 4
f = 3.3;
a = sizeof(f + a); //a = 4, поскольку тип результата – float
```

Структура – это особый тип данных, состоящий из нескольких разнотипных переменных (полей). В общем случае объявление структуры имеет следующий вид:

```
struct  имя_структуры
{
тип  поле_1;
…
тип  поле_N;
};
```

Как и любой другой тип, структуру можно в дальнейшем использовать для объявления переменных, например:

```
struct  MyStructure   //Объявление структуры MyStructure
{
int Fieldl;
char Field2;
float Field3;
};

struct MyStructure YourStruct, OurStruct; //Объявление переменных
YourStruct и OurStruct типа MyStructure
```

Допускается инициализация полей непосредственно при объявлении переменных-структур с помощью перечня значений в фигурных скобках, например:

```
struct DATE
{
int Day;
int Month;
```

```
int Year;
}

structDATEMyBirthday = {7, 8, 1974};
```

Для доступа к полям структуры в программе используют запись вида имя_структуры.поле. То есть, в представленном выше примере структуры DATE для инициализации полей можно было воспользоваться следующими операторами:

```
MyBirthday.Day = 7;
MyBirthday.Month = 8;
MyBirthday.Year = 1974;
```

Структуры, в свою очередь, могут быть полями других структур, например:

```
structME
{
char MyName[30];//Строка длиной 30 символов – имя
struct DATEMyBirthday; //Структура, хранящая день рождения
}

struct ME MyData;
MyData.MyName = "John Smith";
MyData.MyBirthday.Day = 7;
MyData.MyBirthday.Month = 8;
MyData.MyBirthday.Year = 1974;
```

Структуры могут выступать в качестве параметров функций, а также возвращаемого результата. Ниже представлен пример функции, возвращающей структуру типа DATE:

```
struct DATEJanuaryl(int CurYear)
{
struct DATEJan0l;
Jan0l. Day = 1;
Jan0l.Month = 1;
Jan0l.Year = CurYear;
return JanOl;
}
…

struct DATE BeginOfTheYear;
BeginOfTheYear = Januaryl(2006);
```

3.4 Указатели и адреса переменных

Указатель – это переменная, содержащая адрес некоторого элемента данных (переменной, константы, функции). В языке С указатели тесно связаны с обработкой массивов и строк.

Для объявления переменной как указателя используется оператор «*»:

Для присвоения адреса некоторой переменной указателю используется оператор «&», например:

```
char*p; // объявляем указатель (способен содержать адрес хранения
любого символа);
char c= 'A';// объявляем символьную переменную c;
p = &c; // теперь p содержит адрес хранения символа c.
```

Для того чтобы извлечь значение переменной, на которую указывает указатель, используется оператор разыменования «*».

```
char *p;
char b;
char c = 'A';
p = &c;
b= *p; // теперь b равняется не адресу хранения переменной c, а
значению переменной c, то есть 'A'.
```

Аналогичным образом, этот оператор можно использовать и для присвоения некоторого значения переменной, на которую указывает указатель:

```
char *p;
char b, char c = 'A';
p = &b; // теперь p содержит адрес хранения ещё не
инициализированной переменной b;
*p = b; // инициализируем переменную b значением 'A'.
```

Применительно к программированию микроконтроллеров, указатели можно использовать, к примеру, для записи данных в порт ввода/вывода. Предположим, регистр данных порта расположен в памяти по адресу 0x16. В этом случае, для записи в него значения 0xFF можно воспользоваться следующим фрагментом программного кода:

```
unsigned char *p; //объявление указателя на символ
p = 0x16; // присваиваем указателю адрес порта
*p = 0xFF; // теперь на выводах порта будут все единицы
```

С помощью указателей, используемых в качестве параметров функций, можно организовать возврат более одного значения. Рассмотрим следующий пример:

```
int SumAndDiv(int *a, int *b) /* Объявляем функцию, возвращающую
сумму от целочисленного деления без остатка и остатка */
{
     int bufA, bufB; // Объявление целых чисел – bufA и bufB;
     bufA = *a; // bufA = значению, на которое указывает
указатель a;
     bufB = *b; // bufB = значению, на которое указывает указатель
b;
     *a = bufA / bufB; //Указатель a ссылается на результат
целочисленного деления без остатка
     *b = bufA % bufB; //Указатель b ссылается на остаток от
целочисленного деления
     return bufA + bufB; //Функция возвращает сумму
}
…
int vl, v2, sum;
vl   =   10;
v2   =   3;
sum  =   SumAndDiv(&vl,&v2);//sum=4; vl=3; v2=1
```

В функции SumAndDiv вначале сохраняются в буферных переменных значения, на которые указывают указатели a и b. Затем в переменную, на которую указывает указатель a записывается результат деления переданных в функцию значений без остатка, а в переменную, на которую указывается указатель b – остаток от такого деления. Поскольку с помощью указателей мы напрямую обращались к ячейкам памяти, а не к переменным, то после выхода из функции содержимое этих ячеек остается неизменным. При вызове функции SumAndDiv в нее передаются адреса переменных vl и v2, которым в теле функции соответствуют указатели a и b.

3.5 Массивы и строки

Массив – это тип данных, который используется для представления последовательности однотипных значений. Массивы объявляются подобно обычным переменным, с указанием в квадратных скобках размерности (количества элементов в массиве):

```
int digits[10]; //Массив из десяти элементов типа int
char str[10]; //Массив из десяти символов (строка)
```

Доступ к элементам массива реализуется с помощью индекса (порядкового номера элемента, начиная с 0):

```
digits[0] = 0;
digits[1]=1;
str[0] = 'A';
str[l]='B';
```

Зачастую гораздо удобнее инициализировать массив непосредственно при его объявлении, например:

```
int digits [10] = {0, 1, 2, 3, 4, 5, 6, 7 8, 9};
char str[10] = {'T', 'h', 'e', ' ', 'v', 'i', 'n', 'e', 'd'};
int n;
char c;
n = digits[2]; //n =2
c = str[l]; //c = 'h'
```

При разработке программ следует учитывать, что компилятор не всегда может отследить выход индекса за пределы массива. Другими словами, если бы мы в представленном выше примере использовали оператор:

```
c = str[l2];
```

то на этапе компиляции ошибки не возникнет, но в ходе выполнения программы переменной `c` будет присвоено значение "13-го", несуществующего элемента массива, то есть извлечено содержимое области памяти, адрес которой на 12 элементов смещен относительно начального адреса массива.

Язык С допускает использование многомерных массивов, то есть массивов, элементами которых являются массивы. Примеры объявления таких массивов:

```
int a2[10][2]; //Двухмерный массив 10x2
int a3[3][2][5]; //Трехмерный массив 3x2x5
```

Фактически, все элементы многомерного массива хранятся в памяти последовательно, поэтому представленные ниже примеры инициализации аналогичны:

```
int a[2][3] = { {1, 2, 3},
{4, 5,  6} };
```

и

```
int a[2] [3] = { 1, 2, 3, 4, 5, 6 };
```

Для доступа к элементам многомерного массива используются индексы по каждой из размерностей или операции разыменования указателя. Например, чтобы извлечь в некоторую переменную `n` цифру 4 из представленного выше массива a, можно воспользоваться одним из двух вариантов:

```
n = a[1][0]; //Извлекаем элемент из "строки" 1 "столбца" 0 (первый),
то есть – цифру 4
```

```
n = *(a+3); //a - указатель на первый элемент массива,
следовательно a + 3 - указатель на 4-й элемент.
```

Строка в С – это массив типа `char`. Выше был рассмотрен пример объявления такого массива, соответствующего строке "`The line`". Это объявление можно было бы выполнить и с помощью строкового литерала:

```
char str[10] = "The line";
```

Однако в таком случае следует помнить, что компилятор неявно завершает строковые литералы символом '\0', и, таким образом, реальная длина строки больше на 1. Это следует учитывать, чтобы при инициализации массива не выйти за его пределы.

При инициализации строк размерность массива можно явно не указывать:

```
charstr[] = "The line";
```

В этом случае компилятор определит размерность самостоятельно. (Это правило применимо и к инициализации любых других массивов.)

3.6 Операторы ветвления

Операторы ветвления используются для выполнения того или иного блока кода в зависимости от некоторого условия. К таким операторам в языке С относятся `if-else` и `switch-case`.

Компилятор может интерпретировать ноль, как " ложно ", и не ноль, как "истину". Логические операции могут объединять несколько проверяемых условий. Например:

```
if((выражение1)&&((выражение2)||(выражение3)))
{
/* Код программы здесь будет выполняться если:
Выражение1 "Истина" (значит не ноль), и хотя бы одно из выражений
2 и 3 тоже "Истина" (значит не ноль).*/
};
```

3.6.1 Оператор if-else

В простейшем случае оператор if-else имеет следующую структуру:

```
if (условное_выражение)
{
  блок_кода_1;
}
else
{
```

```
  блок__кода_2;
}
```

Если условное выражение истинно, то выполняется блок кода 1, в противном случае – блок кода 2. При этом в качестве блока кода 2 допускается использовать последовательность операторов `else-if`:

```
if (выражение1)
{
  блок_кода_1;
}
else
{
if (выражение2)
    {
      блок_кода_2;
    }
else
    {
if (выражение 3)
      {
        блок_кода_3;
      }
else
      {
        блок_кода_4;
      }
}
}
```

В данном случае каждое выражение будет вычисляться поочередно до тех пор, пока не будет найдено выражение, давшее истинный результат. Если же результаты вычислений всех выражений окажутся ложными, то будет выполнен блок кода 4.

В тех случаях, когда при ложных результатах всех условных выражений не требуется выполнять никаких действий, можно опустить завершающий блок кода вместе с последней ветвью `else`.

Вместо оператора `if-else` можно использовать условные выражения. Так, конструкцию вида:

```
if (условие) блок_кода_1; else блок_кода_2;
```

можно заменить следующим условным выражением:

```
условие ? блок_кода_1 : блок_кода 2;
```

Например:

```
(a == 1) ?b= a*3 : b = 0;//Если a=1, то b = a*3, иначе b = 0
```

3.6.2 Оператор switch-case

В оператора `if-else` можно использовать только выражения, которые сводятся к значению `TRUE` или `FALSE`. В тех случаях, когда необходимо применять выражения, дающие произвольный числовой результат, удобнее воспользоваться оператором `switch-case`. Этот оператор позволяет с помощью некоторой переменной выбирать одно из выражений, соответствующее заданной константе. Его синтаксис:

```
switch(выражение)
{
caseконстанта-выражение1 :блок_кода
caseконстанта-выражение2 :блок_кода
caseконстанта-выражение3 : блок кода
default:блок_кода
}
```

Например код:

```
If (a == 1) b = a*3; else b = 0;
//Если a=1, то b = a*3, иначе b = 0
```

можно заменить кодом:

```
switch(a)
{
case1 :b = a * 3;break;
case2 : b=a+ 10;break;
case3 :b= 0;break;
default;
}
```

Оператор `break` приводит к немедленному выходу из блока `switch`. Если по каким-то причинам необходимо продолжить проверку соответствия выражения выбора тем константам, которые указаны в ветвях `case`, следует убрать операторы `break`.

Ко всему прочему, ветви `case` можно каскадировать. Такой прием особенно удобен в тех случаях, когда нужно, например, проверить введенный символ, не заботясь о том, представляет ли он прописную букву или строчную, или когда нужно получить одну и ту же реакцию на несколько разных чисел. Рассмотрим это на примере фрагмента некоторой абстрактной программы:

```
switch (input)
{
case 'a' : case 'A': DoA(); break;
case 'b': case 'B': DoB(); break;
case '0': case '1': case '2' : case '3' : Do0123(); break;
```

```
case '4': case '5' : case '6' : case '7' : D04567(); break;
default : DoDefault(); break;
}
```

3.7 Циклические конструкции

Циклические конструкции применяют для повторения некоторого блока кода на основании условия цикла. В языке С используются циклические конструкции `while`, `for` и `do-while`.

3.7.1 Конструкция while

Цикл `while` имеет следующий синтаксис:

```
while (условное_выражение)
{
  // Выполнение тела цикла, если выражение истинно
}
```

Другими словами, циклы `while` имеет смысл использовать в тех случаях, когда соответствующий оператор или блок операторов необходимо выполнять до тех пор, пока условное выражение истинно. Пример формирования строки, состоящей из нечетных цифр:

```
int c, i;
const char str[] = "0123456789";
char OddNums[5]; //Строка для хранения нечетных цифр
c = 0; //Счетчик циклов

i = 0;//Индекс массива OddNums
while (c < 10) //До тех пор, пока c меньше 10
{
   //Если остаток от деления c на 2 = 1, то записываем в i-ю позицию
массива OddNums c-й элемент строки str, после чего значение i
автоматически инкрементируется
if ((c % 2) == 1) OddNums[i++] = str[c];
  c++;//c = c + 1 .
}
```

3.7.2 Конструкция for

Цикл `for` имеет следующий синтаксис:

```
for (выражение1; выражение2; выражение3)
{
//  Выполнение  тела цикла
}
```

Выражение 1 выполняется только один раз при входе в цикл, и обычно представляет собой оператор присваивания некоторого начального

значения счетчику цикла. Выражение2 – это условное выражение, определяющее момент выхода из цикла (цикл выполняется до тех пор, пока оно равно TRUE или 1).

Выражение 3 – еще один оператор присваивания, в котором обычно изменяется счетчик цикла или некоторая переменная, влияющая на выполнение условия в выражении 2. Выражения могут быть представлены любыми операторами, включая пустые (то есть, вместо выражения можно поставить только символ точки с запятой).

Циклы while и for в большинстве случаев взаимозаменяемы. Так, представленный выше пример для цикла while, можно переписать в следующем виде:

```
int c, i;
const char str[] = "0123456789";
char OddNums[5]; //Строка для хранения нечетных цифр
i = 0; //Индекс массива OddNums
for (c =0; c < 10; c++)//До тех пор, пока с меньше 10
if ((c % 2) ==  1) OddNums[i++] = str[c];
```

В одних ситуациях удобнее применять циклы for, в других - while. Достоинством циклов for является более наглядная инициализация и организация изменения счетчика цикла. С другой стороны, циклы while более гибкие и обеспечивают больше возможностей для реализации нестандартных программных решений при организации повторяющихся вычислений.

3.7.3 Конструкция do-while

Кроме циклов while и for, в которых вначале выполняется проверка истинности условия цикла, и только потом управление передается блоку операторов цикла, в языке С имеется также конструкция do-while. Она отличается от первых двух тем, что в ней вначале выполняется блок операторов, и только потом проверяется выполнение условия. Другими словами, цикл do-while всегда выполняется как минимум один раз, вне зависимости от условия цикла. Цикл do-while имеет следующий синтаксис:

```
do
{
// Выполнение блока операторов цикла
}
while (условное_выражение);
```

3.7.4 Организация бесконечных циклов

Для организации бесконечного цикла в качестве условного выражения в конструкции `while` или `do-while` можно просто указать значение `TRUE` или 1:

```
while(1) блок_операторов;
```

В случае циклов for это будет выглядеть следующим образом:

```
for (;;) блок_операторов;
```

3.7.5 Операторы break и continue

Если в теле любого цикла встречается оператор `break`, управление тут же передается на оператор, следующий за оператором цикла, вне зависимости от истинности или неистинности условного выражения. При этом во вложенных циклах выход осуществляется не на самый верхний уровень вложенности, а лишь на один уровень вверх.

При выполнении оператор `continue` все находящиеся после него операторы блока пропускаются, а управление передается в начало цикла для следующей итерации. На практике операторы `continue` используются гораздо реже, чем операторы `break`, однако в сложных циклах, требующих принятия решений на основании многих факторов, использование операторов `continue` может быть весьма удобным.

3.8 Директивы препроцессора

Директивы препроцессора, по сути, не являются составной частью языка С, а используются для подстановки кода в текст программы. Препроцессор – это особая программа, которая выполняет предварительную обработку данных до начала компиляции. Рассмотрим стандартные директивы препроцессора.

3.8.1 Директива #include

Директива `#include` используется фактически во всех программах для включения в текст программы заголовочных файлов (с определениями), имеющий расширение .h, или файлов .c (не содержащих функцию `main()`).Эта директива может использоваться в двух формах:

```
#include <имя_файла>
```

или

```
#include "имя_файла"
```

Если имя файла заключено между знаками < и >, то он считается частью стандартной библиотеки, поставляемой вместе с компилятором. Если же имя файла заключено между знаками " и ", то считается, что он расположен в той же папке, что и файл с исходным кодом.

Для написания программ для микроконтроллеров серии 1986BE9x потребуются следующие заголовочные файлы:

```
#include "MDR32Fx.h"
#include "core_cm3.h"
#include "MDR32F9Qx_config.h"
#include "system_MDR32F9Qx.h"
```

3.8.2 Директива #define

Директива `#define` указывает препроцессору на необходимость выполнить подстановку, в тексте программы определенной последовательности символов другой последовательностью. Формат директивы:

```
#define заменяемая_последовательность фрагмент_подстановки
```

Например:

```
#define MyName "JohnSmith"
#define Condition (a>b)
#define Operationa = b
…
char str[ ] = MyName; //Равнозначно char str[ ] = "JohnSmith";
inta, b;
if ConditionOperatrion; //Равнозначноif (a>b) a = b;
```

В директиве `#define` могут использоваться параметры, благодаря чему она становится очень мощным и гибким инструментом, позволяющим заменять один простой текстовый элемент сложным выражением. Такие подстановки называют макросами.

Например, выражение, в котором выбирается большее из двух значений, можно представить в виде следующего макроса.

```
#define larger(x,y) ((x) > (y) ?(x) : (y))
```

Если определен такой макрос, то код, использующий его, может иметь следующий вид:

```
int a = 9;
int b = 7;
intc= 0;
c= larger(a, b);
```

Напоминает вызов функции, однако это не функция – компилятор получит от препроцессора последнюю строку в следующем виде:

```
c = ((a) > (b) ? (a) : (b));
```

Основное отличие макроса от функции заключается в том, что код макроподстановки вставляется препроцессором в программный код везде, где встречается заданный текстовый элемент, код же функции определяется только в одном месте, а в тех местах, где указано ее имя, осуществляется вызов этого кода.

Есть и еще одно отличие, особенно важное при программировании микроконтроллеров, – при использовании макросов, в отличие от функций, ничего не помещается в стек, что позволяет сэкономить оперативную память.

Кроме того, макросы преобразуются в обычный код, который выполняется быстрее, чем код функции, на вызов которого и возврат управления в вызывающую функцию уходит дополнительное машинное время. Наконец, при использовании макросов не требуется формального объявления типов данных.

3.9 Управление регистрами микроконтроллера

Управление работой микроконтроллера в большинстве случаев сводится к следующему простому набору действий с его регистрами:

- Запись в регистр необходимого значения;
- Чтение значения из регистра;
- Установка в единицу нужных разрядов регистра;
- Сброс разрядов регистра в ноль;
- Проверка состояния разряда регистра;
- Изменение логического состояния разряда регистра на противоположное.

Во всех указанных действиях принимает участие оператор присваивания языка С, записываемый в виде знака равенства. Принцип действия оператора сводится к записи в регистр или переменную расположенную слева от него, значения записанного справа.

```
A = 16; //A присвоили значение 16
A = B;//A присвоили значение переменной B
A = B+10;//A присвоили значение переменной B+10
```

Язык С не имеет в своём составе команд непосредственного сброса или установки разрядов переменной, однако присутствуют побитовые логические операции «И» и «ИЛИ», которые успешно используются для этих целей.

Оператор побитовой логической операции «ИЛИ» записывается в виде вертикальной черты – «|» и может выполняться между двумя переменными, а так же между переменной и константой.

Таким образом для любого бита логическое «ИЛИ» с «1» даст в результате «1», независимо от состояния этого бита, а логическая операция «ИЛИ» с «0» оставит в результате состояние исходного бита без изменения. Это свойство позволяет использовать логическую операцию «ИЛИ» для установки n-ого разряда в регистре.

Для этого необходимо вычислить константу с единичным n-ым битом по формуле 2^n которая называется битовой маской и выполнить логическое «ИЛИ» между ней и регистром, например для установки бита №7 в регистре SREG:

```
SREG = SREG | 128;
```

Такую работу с регистром принято называть «чтение - модификация – запись», в отличие от простого присваивания она сохраняет состояние остальных битов без изменения.

Приведённый программный код, устанавливая седьмой бит в регистре SREG, оставляет остальные биты регистра без изменений.

Единственный недостаток такой записи – это константа 128. По ней нелегко угадать установленный седьмой бит, поэтому чаще маску для n-ого бита записывают в следующем виде:

```
SREG = SREG | (1<<7);
```

`(1<<N)` - это выражение на языке С, которое означает что число один, сдвинуто на n разрядов влево. Это и есть маска с установленным n-ым битом. Тогда предыдущий пример может быть записан в виде:

```
SREG |= (1<<7);
```

Данный код означает взять содержимое справа от знака равенства, выполнить между ним и регистром слева операцию, стоящую перед знаком равенства и записать результат в регистр или переменную слева.

Ещё одна логическая операция языка С - побитовое «И», записывается в виде символа «&». Как известно, операция логического «И», применительно к двум битам даёт единицу тогда и только тогда, когда оба исходных бита имеют значение один, это позволяет применять её для сброса разрядов в регистрах. При этом используется битовая маска, в которой все разряды единичные, кроме бита на позиции сбрасываемого. Её легко получить из маски с установленным n-ым битом, применив к ней операцию побитного инвертирования «~», которая изменяет состояние бита на противоположное. Например, `(1<<3)` – в двоичном виде представляет значение `00001000b`. После инвертирования `~(1<<3)` получим значение

11110111b. Таким образом, сброс седьмого бита в регистре SREG будет выглядеть следующим образом:

```
SREG = SREG & (~(1<<7));
//или
SREG &= ~(1<<7);
```

4 Примеры программ для микроконтроллеров серии 1986BE9x

Представленные примеры предназначены для запуска на демонстрационно-отладочной плате 1986EvBrd производства ЗАО «ПКК МИЛАНДР» без использования платы расширения.

4.1 Мигание светодиодом с помощью кнопки

К выводу 0 порта C подключена кнопка («Select»), к порту F – светодиоды (VD2 и VD3). При старте загораются все светодиоды на порту F. Пока кнопка не нажата, светодиод на выводе 0 порта F (VD2) выключен, при нажатии на кнопку светодиод загорается.

```
#include "MDR32Fx.h"
#include "core_cm3.h"
#include "MDR32F9Qx_config.h"
#include "system_MDR32F9Qx.h"
#include "MDR32F9Qx_rst_clk.h"
#include "MDR32F9Qx_port.h"

#define LED1   PORT_Pin_0 //определить нулевой вывод как LED1

static PORT_InitTypeDef PortInit; //объявление структуры PortInit

int main(){
RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);//включить
тактирование порта C
RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTF, ENABLE); );//включить
тактирование порта F

//Инициализация порта C на вход
PortInit.PORT_OE = PORT_OE_IN; // направление передачи данных =
вход
PortInit.PORT_FUNC = PORT_FUNC_PORT; // режим работы вывода порта
= Порт
PortInit.PORT_MODE = PORT_MODE_DIGITAL; // режим работы выводе =
цифровой
PortInit.PORT_SPEED = PORT_SPEED_SLOW; // скорость фронта вывода
= медленный
PortInit.PORT_Pin   = (PORT_Pin_0); // выбор вывода 0 для
инициализации
PORT_Init(MDR_PORTC, &PortInit); //инициализация порта C
заданными параметрами

//Инициализация порта F на выход
PortInit.PORT_OE = PORT_OE_OUT; // направление передачи данных =
Выход
PortInit.PORT_FUNC  = PORT_FUNC_PORT; // режим работы вывода порта
= Порт
PortInit.PORT_MODE  = PORT_MODE_DIGITAL; // режим работы вывода
```

```
= Цифровой
PortInit.PORT_SPEED = PORT_SPEED_SLOW; // скорость фронта вывода
= медленный
PortInit.PORT_Pin   = (PORT_Pin_All);// выбор всех выводов для
инициализации
PORT_Init(MDR_PORTF, &PortInit); //инициализация порта F
заданными параметрами

PORT_SetBits(MDR_PORTF, PORT_Pin_All); // включить все светодиоды
при старте

while(1){ //бесконечный цикл
     if (PORT_ReadInputDataBit(MDR_PORTC,PORT_Pin_0) == 0)
//если кнопка не нажата…
          {
               PORT_SetBits(MDR_PORTF, LED1); // включить
светодиод на выводе 0 порта F
          }
          else //иначе
          {
               PORT_ResetBits(MDR_PORTF, LED1); // выключить
светодиод на выводе 0 порта F
          }
     }
}
```

4.2 Светодиоды, мигающие раз в секунду

К выводам 0 и 1 порта F подключены светодиоды(VD2 и VD3), которые должны переключаться раз в секунду. Это можно сделать с помощью прерываний при переполнении системного таймера SysTick.

```
#include "MDR32Fx.h"
#include "core_cm3.h"
#include "MDR32F9Qx_config.h"
#include "system_MDR32F9Qx.h"
#include "MDR32F9Qx_rst_clk.h"
#include "MDR32F9Qx_port.h"

static PORT_InitTypeDef PortInit;//объявление структуры PortInit
volatile uint32_tdelay_dec = 0;// объявление переменной delay_dec

//Обработчик прерывания системного таймера
void SysTick_Handler (void)
{
     if (delay_dec !=0) delay_dec--;//вычитать из delay_dec, пока
не станет равен 0
}
```

```
//функцияв ременной задержки
void delay_ms (uint32_t delay_ms)
{
     delay_dec = delay_ms;//присвоить delay_dec значение delay_ms
     while (delay_dec) {}; // выполнять функцию пока delay_dec не
станет равным 0
}

int main(){

     RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTF, ENABLE); );//включить
тактирование порта F

//Инициализация порта F на выход
     PortInit.PORT_OE = PORT_OE_OUT; // направление передачи
данных = Выход
     PortInit.PORT_FUNC  = PORT_FUNC_PORT; // режим работы вывода
порта = Порт
     PortInit.PORT_MODE = PORT_MODE_DIGITAL; // режим работы
вывода= цифровой
     PortInit.PORT_SPEED = PORT_SPEED_SLOW; // скорость фронта
вывода = медленная
     PortInit.PORT_Pin = (PORT_Pin_All);// выбор вывода для
инициализации

     PORT_Init(MDR_PORTF, &PortInit); //инициализация порта
Fзаданными параметрами

//Инициализация системного таймера
     SysTick->LOAD |= (8000000/1000)-1; //значение задержки
прерывания при тактовой частоте 8 МГц = 1мс
     SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Pos; //источник
тактирования HCLK
     SysTick->CTRL |= SysTick_CTRL_COUNTFLAG_Pos;// при
досчитывании до нуля таймер генерирует прерывание
     SysTick->CTRL |= ~SysTick_CTRL_ENABLE_Pos;//включить работу
таймера

     while(1){
               delay_ms(1000); //задержка 1 с
               PORT_SetBits(MDR_PORTF, PORT_Pin_All); //
включить светодиоды
               delay_ms(1000);//задержка 1 с
               PORT_ResetBits(MDR_PORTF, PORT_Pin_All); //
выключить светодиоды
     }
}
```

4.3 Передача символа на ПК каждые 5 секунд через интерфейс RS-232

Вывод 5 порта B назначен в качестве линии передачи TXD приемопередатчика UART1, а вывод 6 того же порта – в качестве линии приема RXD. Для преобразования уровней последовательных сигналов, используемых микроконтроллером к уровням используемых интерфейсом RS-232 на демонстрационно-отладочной плате 1986EvBrd используется микросхема 5559ИН4.

При старте микроконтроллер начинает отсылать по последовательному интерфейсу на вход приемопередатчика UART1 цифровое значение переменной `i`каждые 5 секунды. Значение переменной `i` после каждой отправки увеличивается на 1.

```
#include "MDR32F9Qx_config.h"
#include "MDR32Fx.h"
#include "MDR32F9Qx_uart.h"
#include "MDR32F9Qx_port.h"
#include "MDR32F9Qx_rst_clk.h"

static PORT_InitTypeDef PortInit;//объявление структуры PortInit
static UART_InitTypeDef UART_InitStructure;//объявление
структуры UART_InitStructure

volatile uint32_tdelay_dec = 0;// объявление переменной delay_dec

//Обработчик прерывания системного таймера(см. 4.2)
void SysTick_Handler (void)
{
     if (delay_dec !=0) delay_dec--;
}


//функция временной задержки(см. 4.2)
void delay_ms (uint32_tdelay_ms)
{
     delay_dec = delay_ms;
     while (delay_dec) {};
}

int main (void)
{
uint8_t i = 0;//объявление переменной счетчика, хранящей
передаваемое по UARTзначение

//Инициализация системного таймера для функции задержки (см. 4.2)
SysTick->LOAD |= (8000000/1000)-1;
SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Pos;
```

```
SysTick->CTRL |= SysTick_CTRL_COUNTFLAG_Pos;
SysTick->CTRL |= ~SysTick_CTRL_ENABLE_Pos;

RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTB,ENABLE); );//включить
тактирование порта B

//Инициализация порта B для функции UART
PortInit.PORT_PULL_UP = PORT_PULL_UP_OFF;//подтяжка в питание
выключена
PortInit.PORT_PULL_DOWN = PORT_PULL_DOWN_OFF;//подтяжка в ноль
выключена
PortInit.PORT_PD_SHM = PORT_PD_SHM_OFF;//режим триггера Шмитта
выключен
PortInit.PORT_PD = PORT_PD_DRIVER;//режим управляемого драйвера
PortInit.PORT_GFEN = PORT_GFEN_OFF;//входной фильтр выключен
PortInit.PORT_FUNC = PORT_FUNC_ALTER;//альтернативная функция
порта
PortInit.PORT_SPEED = PORT_SPEED_MAXFAST;//скорость фронта
вывода = быстрая
PortInit.PORT_MODE = PORT_MODE_DIGITAL; //режим работы вывода =
цифровой

//Инициализация вывода 5 как UART_TX
PortInit.PORT_Pin = PORT_Pin_5;
PORT_Init(MDR_PORTB, &PortInit);

//Инициализация вывода 6 как UART_RX
PortInit.PORT_OE = PORT_OE_IN;
PortInit.PORT_Pin = PORT_Pin_6;
PORT_Init(MDR_PORTB, &PortInit);

RST_CLK_CPU_PLLconfig
(RST_CLK_CPU_PLLsrcHSIdiv2,0);//конфигурация источника
тактирования HSI делением частоты на 2 и без умножения
RST_CLK_PCLKcmd(RST_CLK_PCLK_UART1, ENABLE); );//включить
тактирование UART1
UART_BRGInit(MDR_UART1, UART_HCLKdiv1);//делитель тактовой
частоты UART = 1

//КонфигурацияUART
UART_InitStructure.UART_BaudRate = 115000; //скорость передачи
данных 15000 бод/сек
UART_InitStructure.UART_WordLength = UART_WordLength8b;//длина
слова в посылке = 8бит
UART_InitStructure.UART_StopBits = UART_StopBits1;//один
стоповый бит
UART_InitStructure.UART_Parity = UART_Parity_No;//без проверки
четности
UART_InitStructure.UART_FIFOMode = UART_FIFO_ON;//включить
работу буфера FIFO приемника и передачи
UART_InitStructure.UART_HardwareFlowControl =
UART_HardwareFlowControl_RXE |
```

```
UART_HardwareFlowControl_TXE;//Разрешить прием, разрешить
передачу данных

//ИнициализацияUART1 с заданными параметрами
UART_Init (MDR_UART1,&UART_InitStructure);

UART_Cmd(MDR_UART1,ENABLE); //включить сконфигурированный UART

while (1) //бесконечный цикл
  {
     UART_SendData (MDR_UART1, i++); //Послать значение счетчика
по UART1, и прибавить к нему 1
     delay_ms(5000);//подождать 5 секунд
  }

}
```

4.4 Включение и выключение светодиода с помощью ПК по интерфейсу RS-232

Вывод 5 порта B назначен в качестве линии передачи TXD приемопередатчика UART1, а вывод 6 того же порта – в качестве линии приема RXD. К выводам 0 и 1 порта Fподключены светодиоды VD2 и VD3.

При получении определенной команды от ПК по интерфейсу RS-232 происходит включение или выключение светодиодов. Код команды и ее описание представлены в таблице 4.

Прием команды микроконтроллером от ПК по интерфейсу RS-232 реализован на основе сигнала прерывания .

Таблица4 – Описание команд управления светодиодами

Код команды	Описание
0x0A	Включить светодиод VD2
0x1A	Выключить светодиод VD2
0x0B	Включить светодиод VD3
0x1B	Выключить светодиод VD3
0xAA	Включить светодиоды VD2 и VD3
0x11	Выключить светодиоды VD2 и VD3

```
#include "MDR32F9Qx_config.h"
#include "MDR32Fx.h"
#include "MDR32F9Qx_uart.h"
#include "MDR32F9Qx_port.h"
#include "MDR32F9Qx_rst_clk.h"
#include "core_cm3.h"
#include "system_MDR32F9Qx.h
```

```
static PORT_InitTypeDef PortInit;//объявление структуры PortInit
static UART_InitTypeDef UART_InitStructure;//объявление
структуры UART_InitStructure

uint8_t Recieve_buff[256]; //объявление массива Recieve_buff,
выполняющего роль буфера для приема
uint8_t Recieve_Wr = 0, Recieve_Rd = 0, Recieve_Cnt =
0;//Recieve_Wr – счетчик при записи элемента в массив, Recieve_Rd
– счетчик при чтении элемента из массива, Recieve_Cnt – общий
счетчик обработанных элементов массива.

// Обработчик прерывания UART
void UART1_IRQHandler (void)
{
     if(UART_GetITStatus(MDR_UART1, UART_IT_RX == 1 ))//если
сигнал прерывания от приема нового пакета
     {
          UART_ClearITPendingBit(MDR_UART1, UART_IT_RX);
//сбросить флаг прерывания
          Recieve_buff[Recieve_Wr] =
UART_ReceiveData(MDR_UART1); //записать полученные данные в
массив Recieve_buff
          Recieve_Wr++; //увеличитьRecieve_Wr
          Recieve_Cnt++; //увеличить Recieve_Cnt
     }
}
//Функция инициализации светодиодов (см.4.1, 4.2)
void init_leds(void)
{
     RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTF, ENABLE);
     PortInit.PORT_Pin   = PORT_Pin_0|PORT_Pin_1;
     PortInit.PORT_OE    = PORT_OE_OUT;
     PortInit.PORT_FUNC  = PORT_FUNC_PORT;
     PortInit.PORT_MODE  = PORT_MODE_DIGITAL;
     PortInit.PORT_SPEED = PORT_SPEED_FAST;
     PORT_Init(MDR_PORTF, &PortInit);
}
//Функция инициализации UART (см. 4.3)
void init_UART(void)
{
     RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTB,ENABLE);
     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_ALTER;
     PortInit.PORT_SPEED = PORT_SPEED_MAXFAST;
     PortInit.PORT_MODE = PORT_MODE_DIGITAL;
```

```
     PortInit.PORT_Pin = PORT_Pin_5;
     PORT_Init(MDR_PORTB, &PortInit);
     PortInit.PORT_Pin = PORT_Pin_6;
     PORT_Init(MDR_PORTB, &PortInit);
     RST_CLK_CPU_PLLconfig (RST_CLK_CPU_PLLsrcHSIdiv2,0);
     RST_CLK_PCLKcmd(RST_CLK_PCLK_UART1, ENABLE);
     UART_BRGInit(MDR_UART1, UART_HCLKdiv1);
     UART_InitStructure.UART_BaudRate  = 115000;
     UART_InitStructure.UART_WordLength  = UART_WordLength8b;
     UART_InitStructure.UART_StopBits = UART_StopBits1;
     UART_InitStructure.UART_Parity = UART_Parity_No;
     UART_InitStructure.UART_FIFOMode = UART_FIFO_OFF;
     UART_InitStructure.UART_HardwareFlowControl =
UART_HardwareFlowControl_RXE | UART_HardwareFlowControl_TXE;

     UART_Init (MDR_UART1,&UART_InitStructure);
     UART_Cmd(MDR_UART1,ENABLE);
     NVIC_EnableIRQ(UART1_IRQn);//включить прерывания UART
     UART_ITConfig(MDR_UART1,UART_IT_RX, ENABLE);//включить
прерывания по приходу нового пакета вUART
}

int main (void)
{
     init_leds(); //инициализация светодиодов
     init_UART();// инициализацияUART

while (1) //бесконечный цикл
{
if(Recieve_Cnt> 0) //если общий счетчик больше 0, т.е. в массиве
Recieve_buff есть необработанные данные, выполнить обработку в
машине состояний switch-case в соответствии с таблицей 4.
{
     switch(Recieve_buff[Recieve_Rd]){
     case 0x0A : PORT_SetBits(MDR_PORTF, PORT_Pin_0); break;
     case 0x1A : PORT_ResetBits(MDR_PORTF, PORT_Pin_0); break;
     case 0x0B : PORT_SetBits(MDR_PORTF, PORT_Pin_1); break;
     case 0x1B : PORT_ResetBits(MDR_PORTF, PORT_Pin_1); break;
     case 0xAA : PORT_SetBits(MDR_PORTF, PORT_Pin_0|PORT_Pin_1);
break;
     case 0x11 : PORT_ResetBits(MDR_PORTF,
PORT_Pin_0|PORT_Pin_1); break;
     default : break;
     }

     Recieve_Rd++; //увеличить Recieve_Rd
     Recieve_Cnt--;// уменьшить Recieve_Cnt
  }
}
}
```

4.5 Управление яркостью свечения светодиода с помощью программной реализации широтно-импульсной модуляции

Широтно-Импульсная Модуляция (ШИМ) – управление средней мощностью нагрузки с помощью серии высокочастотных импульсов. Регулируется усреднённая мощность изменением длительности импульсов и пауз между ними. Чем длиннее импульсы и короче паузы между ними, тем средняя мощность в нагрузке выше. ШИМ – простой случай цифро-аналогового преобразования – плавного управления напряжением или мощностью цифровыми схемами, имеющими только два состояния.

ШИМ представляет собой импульсный сигнал постоянной частоты и переменной скважности (отношение длительности импульса к периоду его следования). С помощью задания скважности можно менять среднее напряжение на выходе ШИМ. Таким способом, меняя выходную мощность, можно управлять яркостью светодиода.

К выводу 0 порта F подключен светодиод VD2. К выводу 0 порта C подключена кнопка SELECT. С помощью метода широтно-импульсной модуляции будем управлять яркостью светодиода. Кнопка SELECT задает уровень скважности. ШИМ генерируется на основе 8 разрядного счетчика заданного переменной `PWM_Counter`.

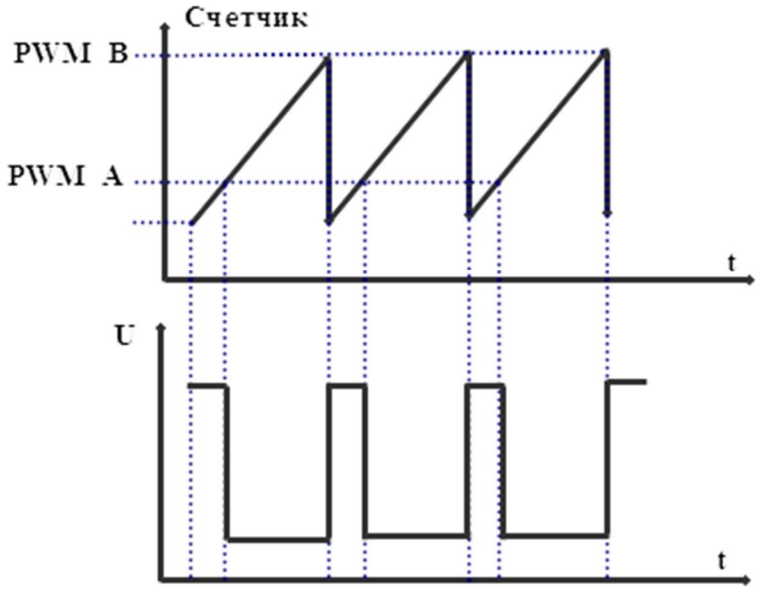


Рисунок 14 – Диаграмма генерации широтно-импульсной модуляции

```
#include "MDR32F9Qx_config.h"
#include "MDR32Fx.h"
#include "MDR32F9Qx_rst_clk.h"
#include "MDR32F9Qx_port.h"
```

```
#defineLED1         PORT_Pin_0 //ОпределитьPORT_Pin_0какLED1

static PORT_InitTypeDefPortInit;

//Функция инициализации порта F на выход (см. 4.1, 4.2)
void init_leds()
{
     RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTF, ENABLE);
     PortInit.PORT_Pin   = (LED1);
     PortInit.PORT_OE    = PORT_OE_OUT;
     PortInit.PORT_FUNC  = PORT_FUNC_PORT;
     PortInit.PORT_MODE  = PORT_MODE_DIGITAL;
     PortInit.PORT_SPEED = PORT_SPEED_FAST;

     PORT_Init(MDR_PORTF, &PortInit);
}

//Функция инициализация порта C на вход (см. 4.1)
void init_button()
{
     RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);
     PortInit.PORT_Pin   = (Button_select);
     PortInit.PORT_OE    = PORT_OE_IN;
     PortInit.PORT_FUNC  = PORT_FUNC_PORT;
     PortInit.PORT_MODE  = PORT_MODE_DIGITAL;
     PortInit.PORT_SPEED = PORT_SPEED_FAST;

     PORT_Init(MDR_PORTC, &PortInit);
}

uint8_t PWM_Counter = 0; //объявление счетчика PWM_Counter
uint8_t PWM_A = 0; //объявление переменной PWM_A (задает
длительность импульса)
uint8_t PWM_B = 40; //объявление переменной PWM_B (задает период
импульса)

static uint8_t btn_old_state = 0; //переменная, хранящая
предыдущее состояние кнопки
static uint8_t btn_state; //переменная, хранящая текущее
состояние кнопки

int main(){
     init_leds(); //инициализация светодиода
     init_button();////инициализация кнопки

     while(1){ //бесконечный цикл

//Увеличение переменной PWM_A только в момент нажатия кнопки
    btn_state = PORT_ReadInputDataBit(MDR_PORTC, Button_select);
//запомнить текущее состояние кнопки (нажата или нет)
          if(btn_old_state == 0 && btn_state == 1)//если кнопка
была не нажата, а теперь нажата
```

```
          {
          PWM_A++;м // увеличитьPWM_A
          if(PWM_A >=PWM_B) // если досчитали до значенияPWM_B
               {
                    PWM_A = 0; //сбросить в ноль
               }
          }
          btn_old_state = btn_state; // запомнить предыдущее
состояние кнопки

//Генерация ШИМ
          if(PWM_Counter>= PWM_B) //если счетчик досчитал до
значения PWM_B
          {
                    PWM_Counter = 0; //сбросить счетчик
                    PORT_SetBits(MDR_PORTF, LED1);//включить
светодиод
          }    else if (PWM_Counter == PWM_A){ //если счетчик
равен PWM_A
                    PWM_Counter++; //увеличить счетчик
                    PORT_ResetBits(MDR_PORTF, LED1);
//выключить светодиод
          }else { //иначе
                    PWM_Counter++; //увеличить счетчик
          }
     }
}
```

4.6 Аппаратная реализация широтно-импульсной модуляции

Программная реализация ШИМ приведенная в примере 4.5 используется в крайних случаях, когда на микроконтроллере отсутствует или уже занят вывод специального назначения для таймера. В общем случае рекомендуется реализовывать аппаратный ШИМ микроконтроллера на основе прерываний от счетчика таймера. Таймер микроконтроллера работает независимо от центрального процессора и в таком случае не требует дополнительных вычислительных ресурсов.

На выводах 1, 2, 3, 4, 5 порта А располагается каналы 1, 2 и 3 таймера1, которые могут быть сконфигурированы в режиме ШИМ. Зафиксировать сигнал ШИМ с разной скважностью можно на ножках 7, 8, 9, 10,12 разъема X25 с помощью цифрового осциллографа.

```
#include "MDR32F9Qx_config.h"
#include "MDR32Fx.h"
#include "MDR32F9Qx_timer.h"
#include "MDR32F9Qx_rst_clk.h"
#include "MDR32F9Qx_port.h"

/*Объявление структур данных*/
TIMER_CntInitTypeDef sTIM_CntInit;
```

```
TIMER_ChnInitTypeDef sTIM_ChnInit;
TIMER_ChnOutInitTypeDef sTIM_ChnOutInit;
PORT_InitTypeDef PORT_InitStructure;

uint16_t CCR1_Val = 500;
uint16_t CCR2_Val = 1000;
uint16_t CCR3_Val = 2000;

int main(void)
{
/* Включим тактирование для устройств периферии*/
RST_CLK_PCLKcmd((RST_CLK_PCLK_RST_CLK),ENABLE);
RST_CLK_PCLKcmd((RST_CLK_PCLK_TIMER1), ENABLE);
RST_CLK_PCLKcmd((RST_CLK_PCLK_PORTA), ENABLE);

/* Конфигурация выводов 1, 2, 3, 4, 5 порта A */
PORT_InitStructure.PORT_Pin   = (PORT_Pin_1 | PORT_Pin_2 |
PORT_Pin_3 | PORT_Pin_4 | PORT_Pin_5);
PORT_InitStructure.PORT_OE    = PORT_OE_OUT;
PORT_InitStructure.PORT_FUNC  = PORT_FUNC_ALTER;
PORT_InitStructure.PORT_MODE  = PORT_MODE_DIGITAL;
PORT_InitStructure.PORT_SPEED = PORT_SPEED_FAST;
PORT_Init(MDR_PORTA, &PORT_InitStructure);

/*Конфигурация таймера
Генерируем 5 ШИМ сигналов с разной скважностью:
TIM1CLK = 8 MHz, Prescaler = 0, TIM1 counter clock = 8 MHz
TIM1 частота = TIM1CLK/(TIM1_Period + 1) = 1.95 KHz
TIM1 канал 1 & канал 1N скважность = TIM1->CCR1 / (TIM1_Period +
1) = 12.5%
TIM1 канал 2 & канал 2N скважность = TIM1->CCR2 / (TIM1_Period +
1) = 25%
TIM1 канал 3 скважность = TIM1->CCR3 / (TIM1_Period + 1) = 50%
-- */

/* Настройка счетчика TIMER1

sTIM_CntInit.TIMER_Prescaler = 0x0; // предусилитель
sTIM_CntInit.TIMER_Period = 4000;// период таймера
sTIM_CntInit.TIMER_CounterMode  = TIMER_CntMode_ClkFixedDir; //
режим счета
sTIM_CntInit.TIMER_CounterDirection  = TIMER_CntDir_Up;//
направление счета
sTIM_CntInit.TIMER_EventSource = TIMER_EvSrc_None; / источник
событий для таймера
sTIM_CntInit.TIMER_FilterSampling = TIMER_FDTS_TIMER_CLK_div_1;
// указывает фильтр событий
sTIM_CntInit.TIMER_ARR_UpdateMode =
TIMER_ARR_Update_Immediately; // режим сброса счетчика
sTIM_CntInit.TIMER_ETR_FilterConf =
TIMER_Filter_1FF_at_TIMER_CLK;// задает параметры выхода ETR
sTIM_CntInit.TIMER_ETR_Prescaler = TIMER_ETR_Prescaler_None;
```

```
//задает предусилитель выхода ETR
sTIM_CntInit.TIMER_ETR_Polarity = TIMER_ETRPolarity_NonInverted;
// задает полярность выхода ETR
sTIM_CntInit.TIMER_BRK_Polarity = TIMER_BRKPolarity_NonInverted;
// задает полярность выхода BRK

TIMER_CntInit (MDR_TIMER1,&sTIM_CntInit);

// инициализация каналов  1,1N,2,2N,3
sTIM_ChnInit.TIMER_CH_Mode = TIMER_CH_MODE_PWM; // режим работы
канала
sTIM_ChnInit.TIMER_CH_REF_Format = TIMER_CH_REF_Format6; //
формат выработки сигнала REF в режиме ШИМ



sTIM_ChnInit.TIMER_CH_Number = TIMER_CHANNEL1; // номер канала
TIMER_ChnInit(MDR_TIMER1, &sTIM_ChnInit);

sTIM_ChnInit.TIMER_CH_Number = TIMER_CHANNEL2;
TIMER_ChnInit(MDR_TIMER1, &sTIM_ChnInit);

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

TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL1, CCR1_Val);//
задать значение CCR0 для канала 1
TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL2, CCR2_Val);//
задать значение CCR0 для канала 2
TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL3, CCR3_Val);//
задать значение CCR0 для канала 3

//Настройки выходных сигналов каналов 1,1N,2,2N,3 Output

sTIM_ChnOutInit.TIMER_CH_DirOut_Polarity =
TIMER_CHOPolarity_NonInverted; // полярность прямого выхода
sTIM_ChnOutInit.TIMER_CH_DirOut_Source = TIMER_CH_OutSrc_REF;//
источник опорного напряжения прямого выхода
sTIM_ChnOutInit.TIMER_CH_DirOut_Mode =
TIMER_CH_OutMode_Output;// режим работы прямого выхода = выход
sTIM_ChnOutInit.TIMER_CH_NegOut_Polarity =
TIMER_CHOPolarity_NonInverted;// полярность инверсного  выхода
sTIM_ChnOutInit.TIMER_CH_NegOut_Source = TIMER_CH_OutSrc_REF;//
источник опорного напряжения инверсного  выхода
sTIM_ChnOutInit.TIMER_CH_NegOut_Mode = TIMER_CH_OutMode_Output;
// режим работы инверсного выхода = выход

sTIM_ChnOutInit.TIMER_CH_Number = TIMER_CHANNEL1; // номер канала
TIMER_ChnOutInit(MDR_TIMER1, &sTIM_ChnOutInit);

sTIM_ChnOutInit.TIMER_CH_Number = TIMER_CHANNEL2;
TIMER_ChnOutInit(MDR_TIMER1, &sTIM_ChnOutInit);
```

```
sTIM_ChnOutInit.TIMER_CH_Number = TIMER_CHANNEL3;
TIMER_ChnOutInit(MDR_TIMER1, &sTIM_ChnOutInit);

TIMER_BRGInit(MDR_TIMER1,TIMER_HCLKdiv1); // Включить
тактирование TIMER1
TIMER_Cmd(MDR_TIMER1,ENABLE); //включить TIMER1

while(1)
{
}
}
```

4.7 Вывод значения оцифрованного сигнала, управляемого переменным резистором, на ПК с помощью интерфейса RS-232

К выводу 2 порта D с помощью перемычки X4-X5 «ADC_INP_SEL» может быть подключен разъем BNC X2 «ADC» или многооборотный переменный резистор R1 на 10 кОм.

С помощью встроенного высокоскоростного 12-разрядного аналогово-цифрового преобразователя будем оцифровывать сигнал изменяемый переменным резистором R1 и передавать оцифрованные значения по интерфейсу RS-232 на ПК. Средствами среды математического моделирования MATLAB будем принимать значения оцифрованного сигнала и строить график его изменения.

```
#include "MDR32Fx.h
#include "MDR32F9Qx_config.h"
#include "MDR32F9Qx_port.h
#include "MDR32F9Qx_rst_clk.h"
#include "MDR32F9Qx_adc.h"
#include "MDR32F9Qx_uart.h"
#include <string.h>

PORT_InitTypeDef PortInit;//объявление структуры PortInit
ADC_InitTypeDef sADC;//объявление структурыs ADC
ADCx_InitTypeDef sADCx;//объявление структурыs ADCx
UART_InitTypeDef UART_InitStructure;//объявление структуры
InitStructure

uint16_ttmp; //переменная хранения текущего значения АЦП
charbuf[2]; //массив-буфер для передачи двух байтов

//Функция инициализации UART (см. 4.3)
void init_UART(void)
{
RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTB,ENABLE);

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_ALTER;
PortInit.PORT_SPEED = PORT_SPEED_MAXFAST;
PortInit.PORT_MODE = PORT_MODE_DIGITAL;

PortInit.PORT_Pin = PORT_Pin_5;
PORT_Init(MDR_PORTB, &PortInit);

PortInit.PORT_Pin = PORT_Pin_6;
PORT_Init(MDR_PORTB, &PortInit);

RST_CLK_CPU_PLLconfig (RST_CLK_CPU_PLLsrcHSIdiv2,0);

RST_CLK_PCLKcmd(RST_CLK_PCLK_UART1, ENABLE);

UART_BRGInit(MDR_UART1, UART_HCLKdiv1);

UART_InitStructure.UART_BaudRate = 57600;
UART_InitStructure.UART_WordLength = UART_WordLength8b;
UART_InitStructure.UART_StopBits = UART_StopBits1;
UART_InitStructure.UART_Parity = UART_Parity_No;
UART_InitStructure.UART_FIFOMode = UART_FIFO_ON;
UART_InitStructure.UART_HardwareFlowControl =
UART_HardwareFlowControl_RXE | UART_HardwareFlowControl_TXE;

UART_Init (MDR_UART1,&UART_InitStructure);

UART_Cmd(MDR_UART1,ENABLE);
}


//функция временной задержки (см. 4.2)
volatileuint32_tdelay_dec = 0;

void SysTick_Handler (void)
{
     if (delay_dec !=0) delay_dec--;
}

void delay_ms (uint32_t delay_ms)
{
     delay_dec = delay_ms;
     while (delay_dec) {};
}

int main(void)
{
//Инициализация системного таймера для функции задержки (см. 4.2)
SysTick->LOAD |= (8000000/1000)-1;
SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Pos;
```

```
SysTick->CTRL |= SysTick_CTRL_COUNTFLAG_Pos;
SysTick->CTRL |= ~SysTick_CTRL_ENABLE_Pos;

RST_CLK_PCLKcmd(RST_CLK_PCLK_ADC, ENABLE);//включение
тактирования АЦП
RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTD,ENABLE);//включение
тактирования порта D

PortInit.PORT_Pin = PORT_Pin_2; // номер вывода
PortInit.PORT_OE = PORT_OE_IN; // направление работы вывода-вход
PortInit.PORT_MODE = PORT_MODE_ANALOG; // режим работы-аналоговый

PORT_Init(MDR_PORTD, &PortInit); //инициализировать порт D с
заданными параметрами

/* настройкаАЦП */
sADC.ADC_SynchronousMode = ADC_SyncMode_Independent; // режим
работы АЦП-независимый
sADC.ADC_StartDelay  = 0; // задержка включения АЦП от начала
работы = 0
sADC.ADC_IntVRefTrimming = 1; //подстройка опорного напряжения

ADC_Init (&sADC); //инициализировать АЦП с заданными параметрами

/* настройка каналаADC1 АЦП */
ADCx_StructInit (&sADCx);
sADCx.ADC_ClockSource = ADC_CLOCK_SOURCE_CPU; // источник
тактирования
sADCx.ADC_SamplingMode = ADC_SAMPLING_MODE_CICLIC_CONV; // режим
считывания
sADCx.ADC_ChannelSwitching = ADC_CH_SWITCHING_Disable; //
выключить переключение каналов
sADCx.ADC_ChannelNumber = ADC_CH_ADC2;//номер канала
sADCx.ADC_Channels = 0;  // маска каналов нужна только, если
предусмотрено переключение каналов
sADCx.ADC_LevelControl = ADC_LEVEL_CONTROL_Disable; // отключить
контроль за уровнем сигнала
sADCx.ADC_LowLevel = 0; // задать нижний уровень сигнала
sADCx.ADC_HighLevel = 0; // задать верхний уровень сигнала
sADCx.ADC_VRefSource = ADC_VREF_SOURCE_INTERNAL; // источник
опорного напряжения
sADCx.ADC_IntVRefSource = ADC_INT_VREF_SOURCE_INEXACT; // тип
опорного напряжения
sADCx.ADC_Prescaler = ADC_CLK_div_512; // делитель частоты
sADCx.ADC_DelayGo = 7; // задержка начала измерений

ADC1_Init (&sADCx);

ADC1_Cmd (ENABLE); //включить канал ADC1

init_UART();// инициализация UART
```

```
while(1){
     tmp = MDR_ADC->ADC1_RESULT& 0x0FFF; //считать текущее
значение АЦП
     memcpy(buf, &tmp, sizeof(uint16_t));//записать значения в
буфер buf

     while(UART_GetFlagStatus (MDR_UART1, UART_FLAG_BUSY)){}
//дождаться, когда UART будет готов к передачи
     UART_SendData(MDR_UART1, buf[1]);// передать 1-ый байт
значения АЦП

     while(UART_GetFlagStatus (MDR_UART1, UART_FLAG_BUSY)){}
//дождаться, когда UART будет готов к передачи
     UART_SendData(MDR_UART1, buf[0]); ]);// передать 1-ый байт
значения АЦП

     delay_ms(500); // подождать 500 мс
}
}
```

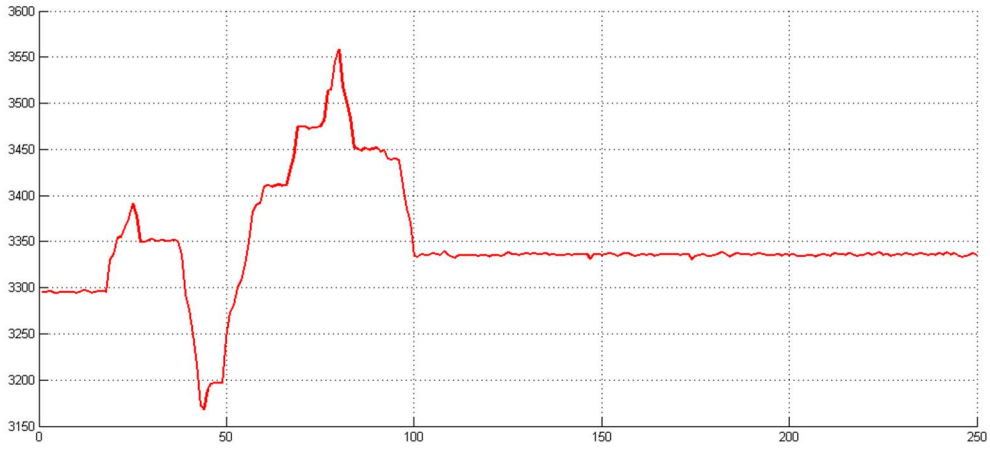


Рисунок 15 – График построения оцифрованных значений в среде MATLAB

Прием данных и построение графика (рисунок 15) в MATLAB реализуется с помощью кода:

```
s = serial('COM2','BaudRate',57600);
fopen(s);
cnt = 250; %число отсчетов
as = 0;
%настройка отображения графика
scrsz = get(0,'ScreenSize');
figure('Position',[scrsz(3)/4 scrsz(4)/4 scrsz(3)/1.5
scrsz(4)/2]);
xlim([0 cnt]);
gridon;
```

```
for i=1:cnt;
    A = fread(s,2,'uchar');
    as(i)=(A(2)+bitshift(A(1),8));

hold on
    plot(as, '-','Color',[1 0 0],'LineWidth',2)
    drawnow;

end;
fclose(s);
```

5 Лабораторный отладочный макет

В состав лабораторного отладочного макета входят две платы - демонстрационно-отладочная плата 1986EvBrd производства ЗАО «ПКК МИЛАНДР» и плата расширения.

Демонстрационно-отладочная плата 1986EvBrd (рисунок 16) предназначена для:

- демонстрации функционирования и оценки производительности микроконтроллера1986ВЕ93У и его основных периферийных модулей;
- демонстрации функционирования интерфейсных микросхем CAN и COM (RS-232) интерфейсов;
- отладки собственных проектов с применением установленных на плате блоков;
- программирования памяти программ микроконтроллеров 1986ВЕ93У.

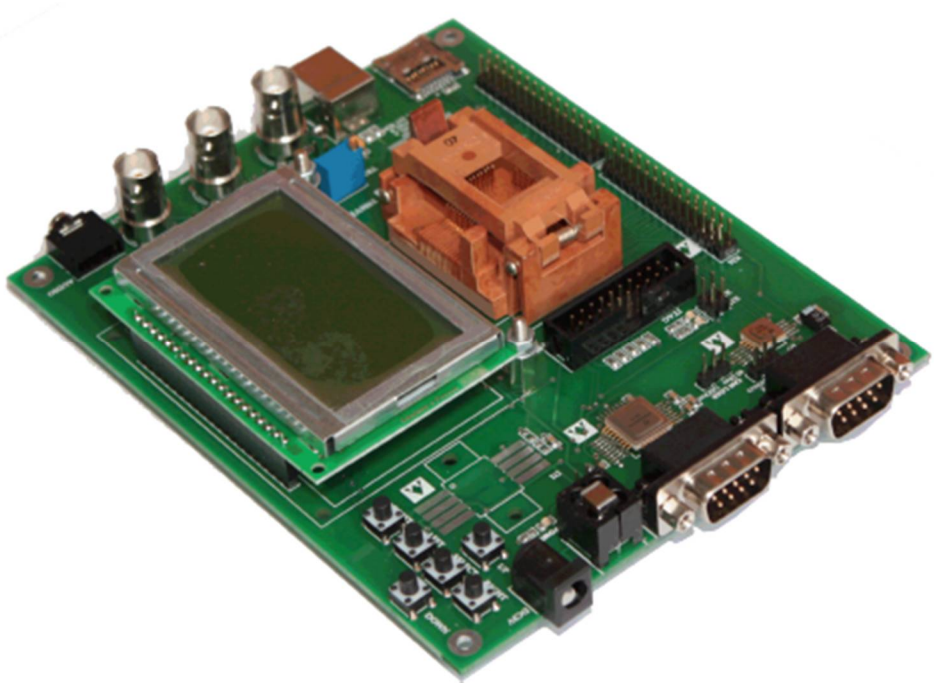


Рисунок 16 – Демонстрационно-отладочная плата 1986EvBrd

Для демонстрации функционирования, 1986EvBrd может подключаться к персональному компьютеру по CAN или COM (RS-232) интерфейсу;

Для программирования памяти программ микроконтроллеров 1986ВЕ93У применяется внешний внутрисхемный программатор ULINK2 (для среды программирования Keil) или JEM-ARM-V2(для среды программирования Phyton).

Питание платы 1986EvBrd осуществляется от адаптера постоянного тока напряжением+5 вольт или от шины USB.

Установленные компоненты на плату показаны на рисунке 17, их описание в таблице 5.

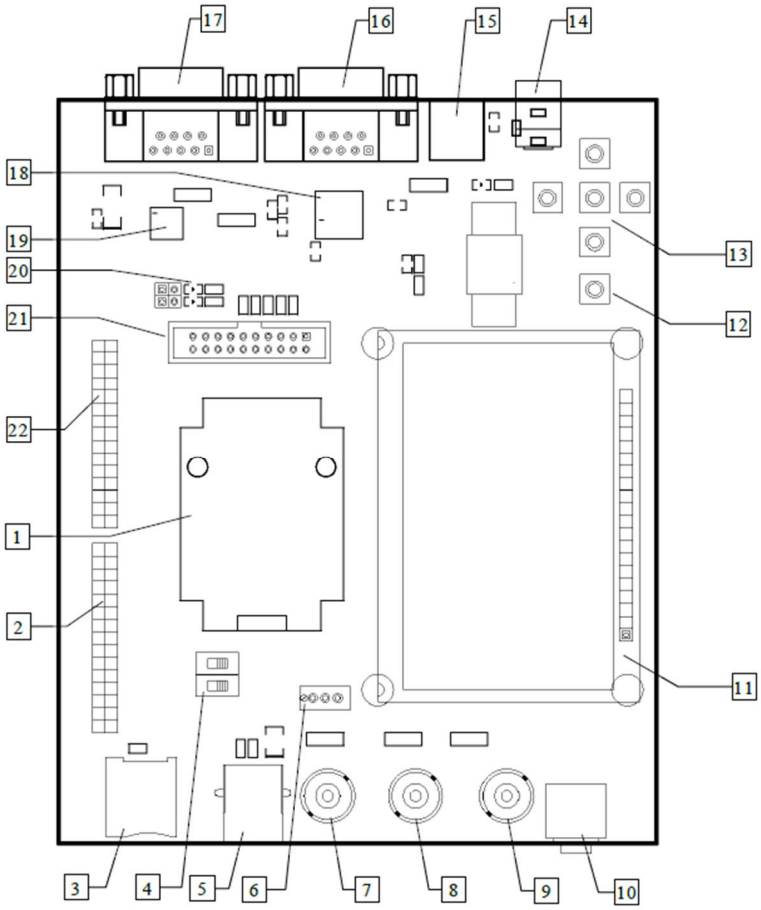


Рисунок 17 – Компоненты платы 1986EvBrd

Таблица5 – Описание компонентов платы 1986EvBrd

Поз. № на рисунке 17	Описание компонентов платы 1986EvBrd
1.	Контактирующее устройство для микроконтроллера 1986ВЕ93У. Микроконтроллер должен быть установлен в спутник-держатель.
2.	Разъем X25 портов A,E,F микроконтроллера.
3.	Разъем карты памяти microSD.
4.	Переключатели выбора режима загрузки.
5.	Разъем USB-B.
6.	Подстроечный резистор на 2-м канале АЦП
7.	Разъем BNC внешнего сигнала на 2-м канале АЦП
8.	Разъем BNC внешнего сигнала на 1-м входе компаратора

9.	Разъем BNC выхода ЦАП1
10.	Разъем Audio 3.5мм выхода ЦАП1 через звуковой усилитель.
11.	ЖК индикатор 128x64MT-12864A
12.	Кнопка RESET.
13.	Кнопки UP, DOWN, LEFT, RIGHT, SELECT
14.	Разъем питания 5В.
15.	Фильтр питания.
16.	Разъем RS-232.
17.	Разъем CAN.
18.	Приемо-передатчик RS-232 5559ИН4.
19.	Приемо-передатчик CAN 5559ИН14.
20.	Набор светодиодов на порте F.
21.	Разъем отладки JTAG-A.
22.	Разъем X24 портов B,C,D микроконтроллера

Назначение установленных на плате конфигурационных перемычек:

- POWER_SEL – выбор источника питания для платы между разъемом USB и внешним источником питания.
- SLEW RATE – выбор скорости передачи данных интерфейса CAN.
- CAN LOAD – выбор нагрузки линии CAN.
- ADC_INP_SEL – выбор источника сигнала для 2-го канала АЦП между подстроечным резистором «TRIM» и BNC разъемом «ADC».
- COMP_INP_SEL – выбор источника сигнала на 1-м входе компаратора между BNC разъемом «COMP_INP» и выходом ЦАП1.
- DAC_OUT_SEL – выбор назначения сигнала с выхода ЦАП1 между BNC разъемом «DAC_OUT» и звуковым усилителем.

Назначение установленных на плате переключателей и клавиш:

- SW1, SW2 – переключатели выбора режима работы. Конфигурация режимов работы представлена в таблице 6.
- UP, DOWN, LEFT, RIGHT, SELECT – программируемые пользователем клавиши.
- RESET – сигнал аппаратного сброса МК.

Таблица6 – Режимы работы платы 1986EvBrd

SW2	SW1	Режим работы
0	0	Режим микроконтроллера, код исполняется из Flash памяти начиная с адреса 0x0800_0000.
0	1	Режим микроконтроллера, код исполняется из Flash памяти начиная с адреса 0x0800_0000, отладка через разъем JTAG_A.
1	0	Режим микропроцессора, код исполняется из внешней памяти начиная с адреса 0x1000_0000.
1	1	Режим микропроцессора, код исполняется из внешней памяти начиная с адреса 0x1000_0000, отладка через разъем JTAG_B.

Плата расширения (рисунок 18) предназначена для увеличения технических возможностей демонстрационно-отладочной платы 1986EvBrd и подключается к разъемам X24, X25 отладочной платы. Электрическая схема платы расширения представлена в Приложении 1.

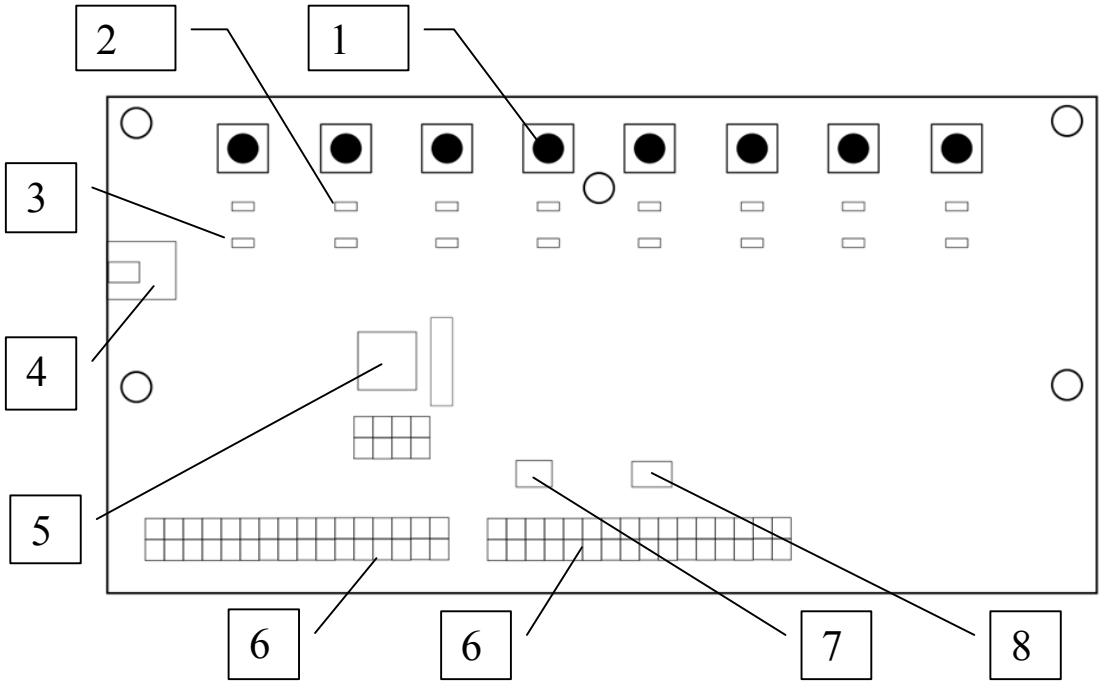


Рисунок 18 – Плата расширения

В состав платы расширения входят:

Поз. № на рисунке 18	Описание компонентов платы расширения
1.	Блок из 8 кнопок KEY;
2.	Блок из 8 светодиодов LED_k, которые хранят текущее состояние соответствующей кнопки KEY;
3.	Блок из 8 светодиодов LED;
4.	Разъем mini-USB для переключения режима лабораторной работы (2 или 4 лабораторные работы). Для переключения используется программное обеспечение «Lab_changer».
5.	Микроконтроллер STM32, предназначенный для генерации аналоговых сигналов (лабораторная работа №4, №6), взаимодействия по протоколам SPI, I2C (лабораторная работа №5) и управлением режимом работы платы расширения;
6.	Разъем для подключения платы расширения к демонстрационно-отладочная плата 1986EvBrd
7.	EEPROM MICROWIRE – постоянное запоминающее устройство на основе микросхемы M93C86xx, реализует обмен данными по протоколу

	MICROWIRE (лабораторная работа №5);
8.	EEPROM I2C – постоянное запоминающее устройство на основе микросхемы M24C01/02-W, реализует обмен данными по протоколу I2C (лабораторная работа №5);

Меры безопасности при работе с лабораторным отладочным макетом

Отладочный макет предназначен для использования в качестве средства разработки аппаратуры и программного обеспечения в условиях учебной/промышленной лаборатории. Для облегчения использования компоненты платы и соединительные проводники открыты для пользователя и окружающей среды.

При работе с отладочным макетом может произойти разряд статического электричества, что может повлечь повреждение компонентов платы. Поэтому, устройство должно иметь постоянную защиту от электростатического разряда. В дополнение к изложенным выше, необходимо соблюдать следующие рекомендации:

- Пока питание платы включено, не изменяйте подключение электронных компонентов на разъёмах зоны макетирования.
- Пока питание платы включено, не касайтесь открытых проводников и электронных компонентов.
- Будьте осторожны при манипуляциях с переключателями, кнопками, ручками управления при включённом питании платы.
- Перед переноской устройства или начала работы с ним уравняйте потенциалы Вашего тела и платы касанием снятия заряда статического электричества.

Лабораторная работа №1: Управление светодиодом

Цель работы: ознакомиться с лабораторным комплексом, изучить принципы построения и организацию программ микроконтроллеров серии 1986BE9x в программной среде разработки Keil uVision, на практике реализовать взаимодействие с портами ввода/вывода.

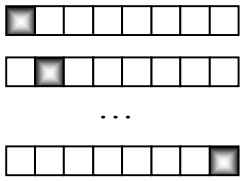
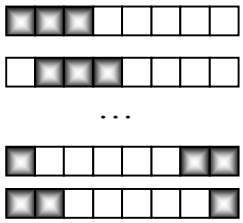
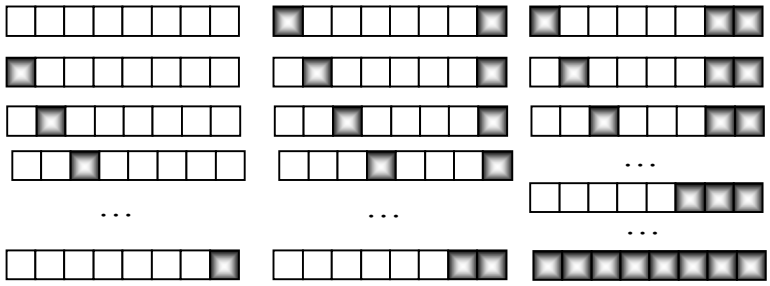
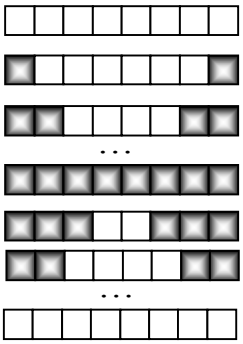
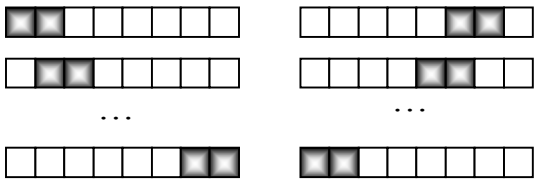
Задание

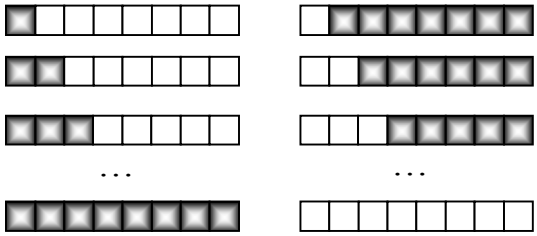
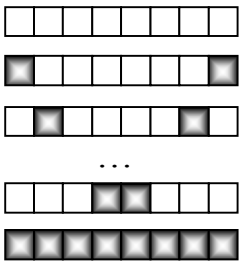
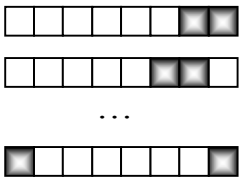
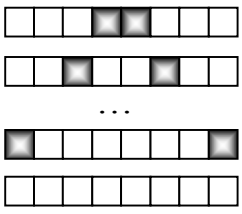
Реализовать устройство мигания светодиодами LED1-LED8 в соответствии с вариантом индивидуального задания.

Содержание отчета

1. Титульный лист, содержащий название работы, номер варианта индивидуального задания, фамилия, имя, отчество номер группы и логин студента.
2. Цель работы
3. Краткие теоретические сведения.
4. Электрическая принципиальная схема устройства.
5. Код программы
6. Выводы.

Варианты индивидуального задания

№	Задание	Исходные данные
1.	«Бегущий огонек» для 1 светодиода слева направо	
2.	Бегущий огонек» для 3-х светодиодов слева направо	
3.	«Накопление» слева направо	
4.	Последовательное включение и выключение всех светодиодов к центру	
5.	«Бегущий огонек» для 2-х светодиодов слева направо и обратно	

6.	Последовательное зажигание и выключение светодиодов	
7.	«Накопление» справа налево (выключение светодиодов)	
8.	«Сталкивающиеся огоньки»	
9.	«Бегущий огонек» 2 светодиода справа налево	
10.	«Разбегающиеся огоньки»	

Лабораторная работа №2: Логические выражения

Цель работы: изучение логических операций и операций ветвления микроконтроллера серии 1986ВЕ9х, операций ввода/вывода данных с помощью кнопок и светодиодов.

Задание

Реализовать устройство определения попадания точки в заштрихованную область. Область ограничена линиями контура. Описать математически систему неравенств, обеспечивающую попадание произвольной точки в заштрихованную область.

Ввод координат точек по оси X осуществить с помощью кнопок KEY1-KEY4, по оси Y – кнопок KEY5-KEY8. Проверка попадания точки в область должна производиться по нажатию кнопки SELECT. Вывод результата реализовать на светодиоды LED1-LED8. При попадании точки в заштрихованную область светодиоды LED1-LED8 должны моргнуть один раз, если точка не попадает в область – моргнуть два раза.

Комбинация выставленных состояний кнопок соответствует 3 разрядному целому числу со знаком в бинарном виде. Старший разряд отвечает за знак (если кнопка включена – то число с минусом). Пример сформированных чисел показан на рисунке 20.

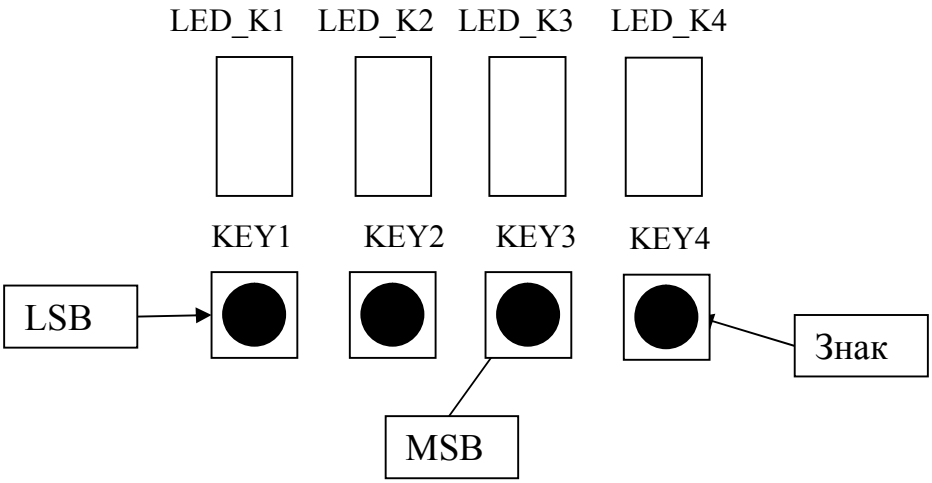


Рисунок 19 – Структура формирования числа

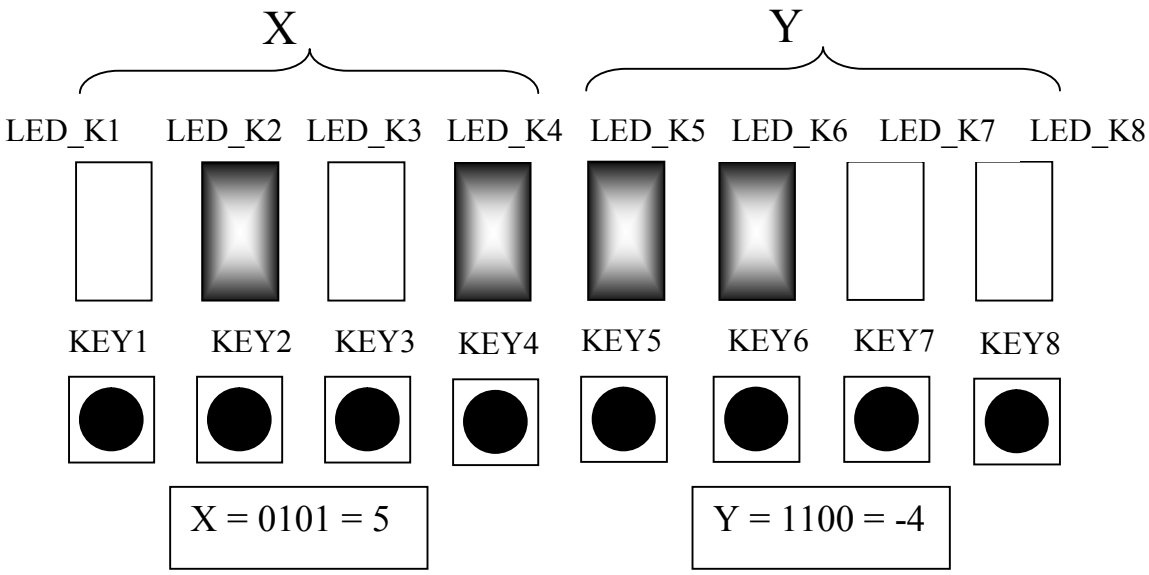
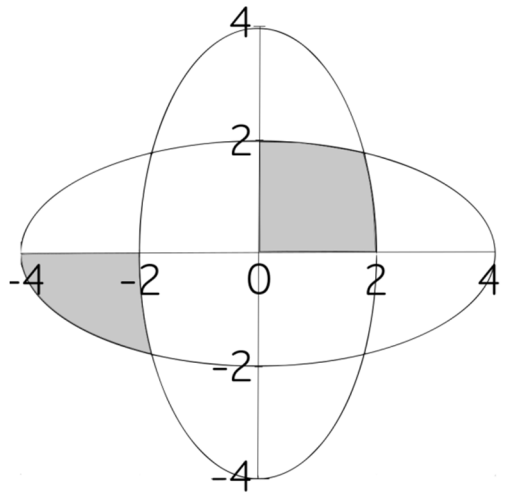
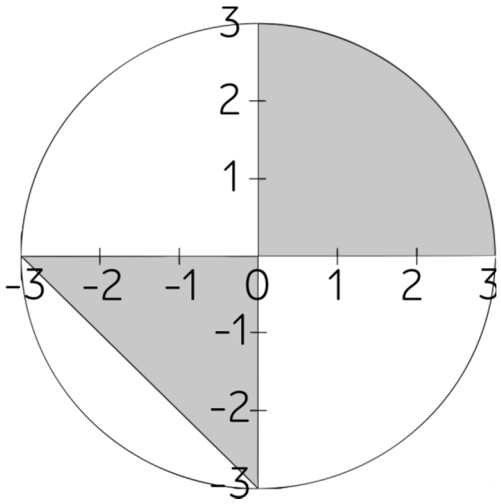
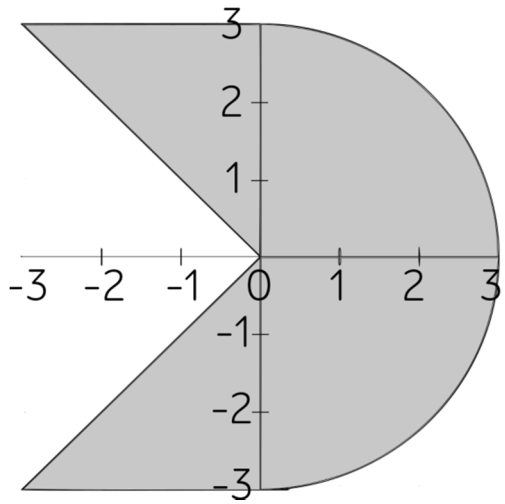
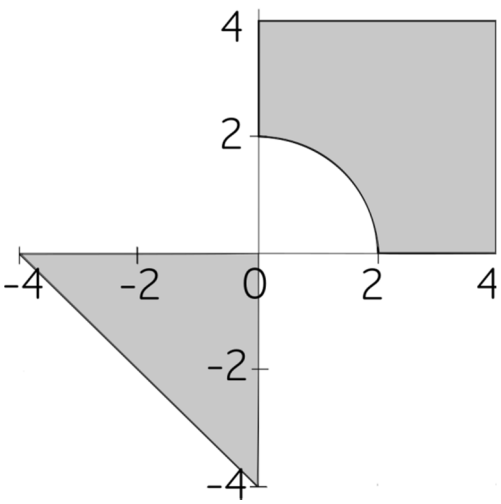
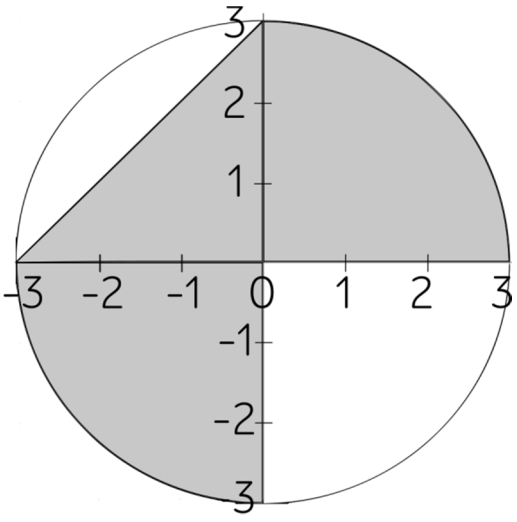
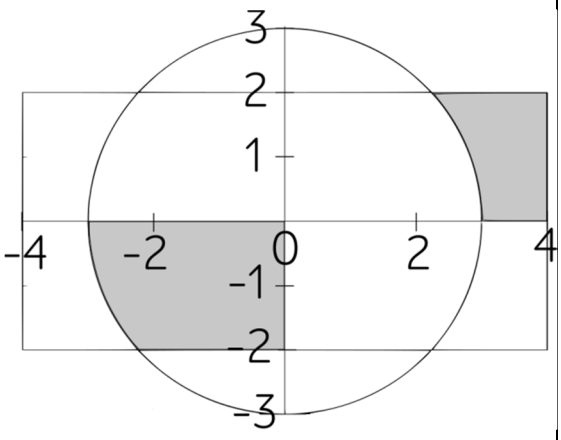


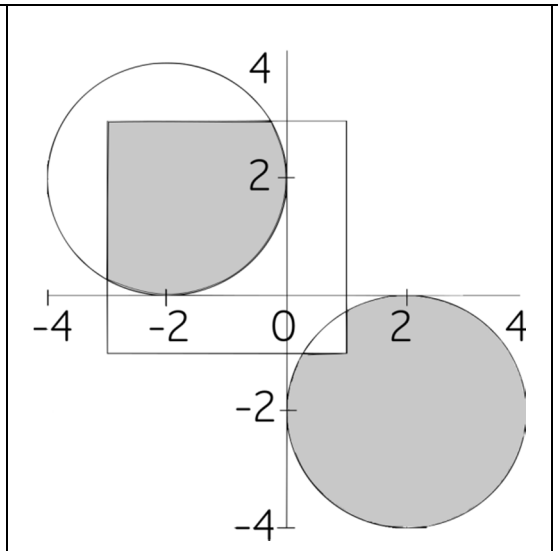
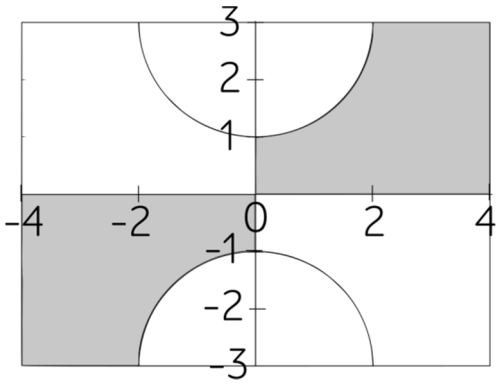
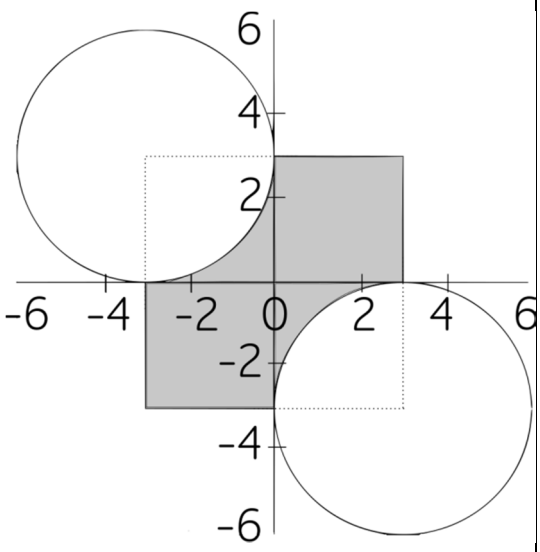
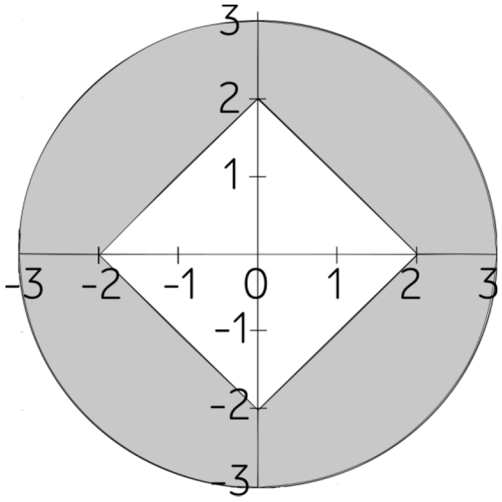
Рисунок 20 – Пример формирования координат по Хи Y

Содержание отчета

1. Титульный лист, содержащий название работы, номер варианта индивидуального задания, фамилия, имя, отчество номер группы и логин студента.
2. Цель работы
3. Краткие теоретические сведения.
4. Электрическая принципиальная схема устройства.
5. Код программы
6. Выводы.

Варианты индивидуального задания

№	Задание	№	Задание
1.		2.	
3.		4.	
5.		6.	

7.		8.	
9.		10.	

Лабораторная работа №3: Реализация протокола обмена данных по интерфейсу RS-232

Цель работы: освоить приемы организации взаимодействия микроконтроллера с персональным компьютером по интерфейсу RS-232.

Задание

Реализовать устройство, выводящее на светодиоды LED1-LED8 число в бинарном виде, переданное по интерфейсу RS-232 с персонального компьютера по протоколу в соответствии с вариантом индивидуального задания через программу-терминал (например «`Terminal v.1.9b by Br@y`»).

Содержание отчета

1. Титульный лист, содержащий название работы, номер варианта индивидуального задания, фамилия, имя, отчество номер группы и логин студента.
2. Цель работы
3. Краткие теоретические сведения.
4. Электрическая принципиальная схема устройства.
5. Код программы
6. Выводы.

Варианты индивидуального задания

№	Протокол	Скорость передачи данных
1.	0x10 0x11 0x12 0x02 Data 0x7E CRC	9600
2.	0x72 0x30 0x64 0xBC Data 0xBC 0x10 CRC	19200
3.	0x02 0x03 0x04 0xA0 Data 0x7F CRC	14400
4.	0xAB 0xCD 0xEF 0xFF 0x14 0x02 Data CRC	57600
5.	0x1B 0x51 0x0A 0x19 0x78 Data 0x2C CRC	115200
6.	0xE2 0xE1 0x22 0x02 0x7C Data 0x45 CRC	28800
7.	0x10 0x20 0x30 Data 0xBC 0x10 CRC	115200
8.	0x0A 0x11 0x0A 0x15 Data 0x85 CRC	28800
9.	0xAC 0xAD 0xAE 0x32 0x44 0x54 Data 0x44 CRC	9600
10.	0xFF 0xFE 0xFF 0x880x15 Data 0x14 CRC	57600

Поле Data – 1 байт, содержащее передаваемое с персонального компьютера число в диапазоне от 0 до 255.

Поле CRC – 1 байт, контрольная сумма пакета и равна дополнению суммы *всех* байт пакета без учета переноса до значения 0x100. Например, для пакета 0xCC 0xF2 контрольная сумма будет равна:

$$CRC = 0\times100 - \left(\left(0\times CC + 0\times F2\right)\,\&\,0\times FF\right) = 0\times100 - 0\times BE = 0\times42$$

Лабораторная работа №4: Оцифровка и фильтрация аналогового сигнала

Цель работы: изучить принципы аналогово-цифрового преобразования сигналов, познакомиться с базовыми методами обработки цифрового входного сигнала.

Задание

Реализовать устройство оцифровки сигнала, преобразования цифровым фильтром в соответствии с вариантом индивидуального задания и передачу на персональный компьютер.

Устройство должно оцифровывать входной аналоговый сигнал, подаваемого с платы расширения на ножку PD3 и выводить значения по интерфейсу RS-232. Средствами среды математического моделирования MATLAB производить чтение данных и выводить на экран в виде графика.

При включении кнопки KEY1 должна производиться фильтрация сигнала и передача по интерфейсу RS-232 уже отфильтрованного сигнала.

Внимание! *Для выполнения лабораторной работы №4 необходимо переключить режим работы платы расширения с помощью программного обеспечения «Lab_changer».*

Содержание отчета

1. Титульный лист, содержащий название работы, номер варианта индивидуального задания, фамилия, имя, отчество номер группы и логин студента.
2. Цель работы
3. Краткие теоретические сведения.
4. Электрическая принципиальная схема устройства.
5. Код программы
6. Выводы.

Варианты индивидуальных заданий

№	Задание
1.	Медианный фильтр с ядром 3
2.	Пороговый фильтр по уровню $0,7U_{\max}$
3.	Медианный фильтр с ядром 7
4.	Пороговый фильтр по уровню $0,4U_{\max}$
5.	Медианный фильтр с ядром 5

6.	Усредняющий фильтр с ядром 5
7.	Усредняющий фильтр с ядром 7
8.	Пороговый фильтр по уровню $0,9U_{\max}$
9.	Усредняющий фильтр с ядром 3
10.	Пороговый фильтр по уровню $0,5U_{\max}$

Здесь $U_{\max}$ - максимальная амплитуда входного аналогового сигнала [В]

Лабораторная работа №5: Обмен данными по интерфейсу I2C/SPI/Microwire

Цель работы: изучить работу распространенных интерфейсов I2C, SPI, Microwire для обмена информации между микросхемами внутри устройств.

Задание

Реализовать устройство чтения/записи данных во внешнее постоянное запоминающее устройство типа EEPROM по интерфейсу в соответствии с вариантом индивидуального задания

Запись и чтение данных в запоминающее устройство осуществлять по интерфейсу RS-232.

Содержание отчета

1. Титульный лист, содержащий название работы, номер варианта индивидуального задания, фамилия, имя, отчество номер группы и логин студента.
2. Цель работы
3. Краткие теоретические сведения.
4. Электрическая принципиальная схема устройства.
5. Код программы
6. Выводы.

Варианты индивидуальных заданий

№	Устройство ПЗУ типа EEPROM	Протокол
1.	STM32L151	SPI
2.	M24C01/02-W	I2C

3.	M93C86xx	Microwire
4.	M24C01/02-W	I2C
5.	STM32L151	SPI
6.	M24C01/02-W	I2C
7.	M93C86xx	Microwire
8.	STM32L151	SPI
9.	M93C86xx	Microwire
10.	M24C01/02-W	I2C

Описание протоколов приведено в техническом описании соответствующих устройств ПЗУ и хранится в директории «…/Microntroller_programming/datasheets».

Лабораторная работа №6: Генерация аналогового сигнала

Цель работы: изучить принципы цифро-аналогового преобразования сигналов.

Задание

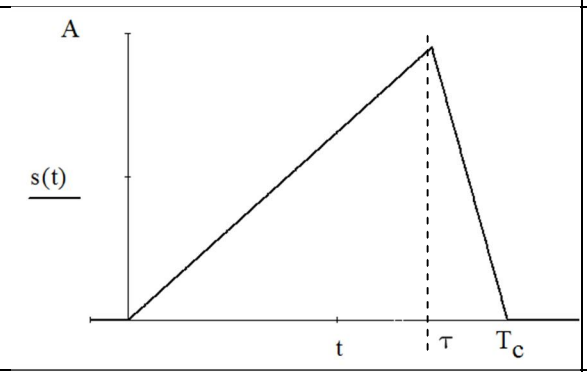
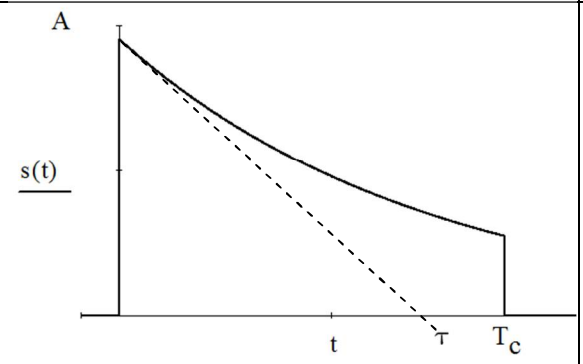
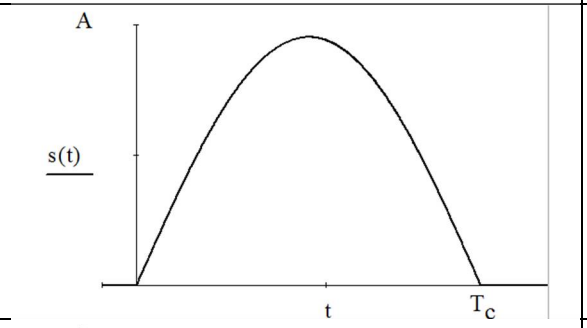
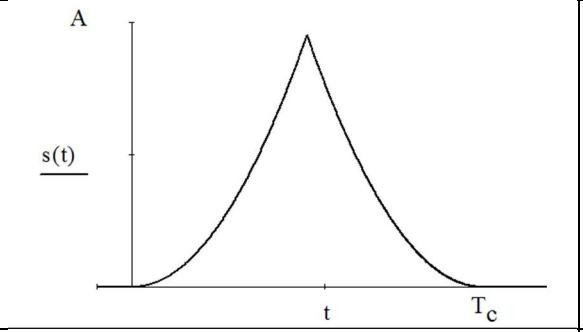
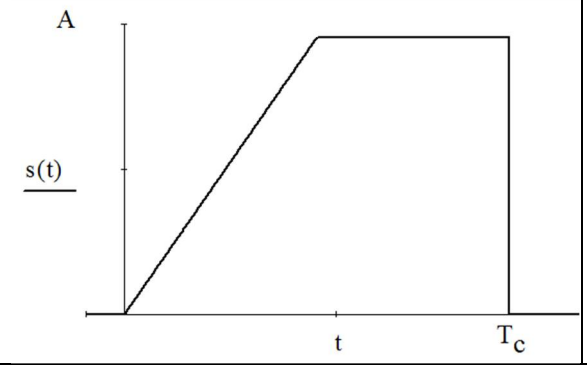
Реализовать устройство генерации заданного в соответствии с вариантом индивидуального задания аналогового сигнала с использованием встроенного в микроконтроллер цифро-аналогового преобразователя (ЦАП).

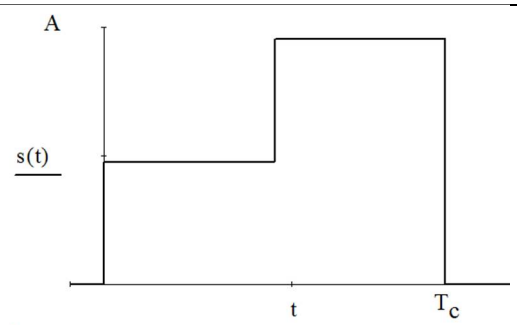
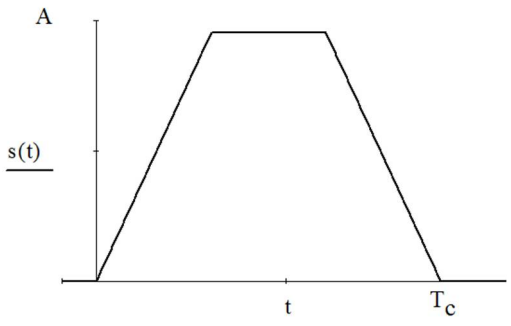
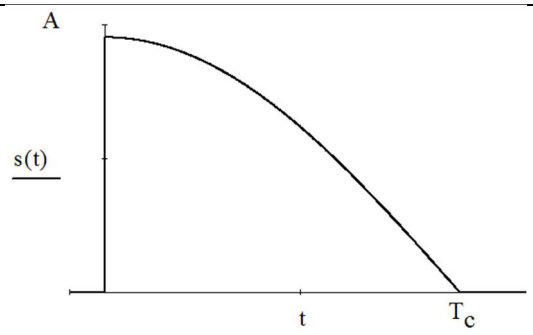
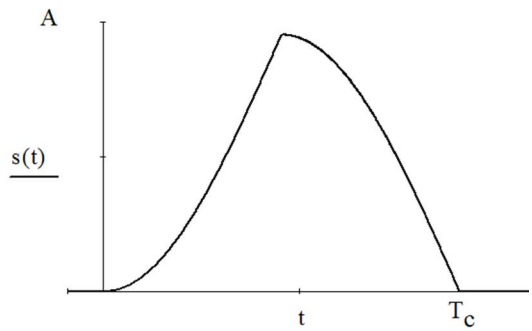
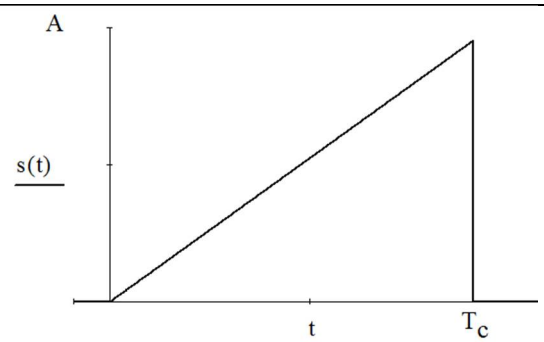
Устройство должно генерировать аналоговый сигнал на разъеме X14 «DAC_OUT» демонстрационно-отладочной платы плата 1986EvBrd.

Содержание отчета

1. Титульный лист, содержащий название работы, номер варианта индивидуального задания, фамилия, имя, отчество номер группы и логин студента.
2. Цель работы
3. Краткие теоретические сведения.
4. Электрическая принципиальная схема устройства.
5. Код программы
6. Выводы.

Варианты индивидуальных заданий

№	Вид сигнала	Функция
1.		$s(t)=\begin{cases}\dfrac{A}{\tau}t \text{ при } t\in[0,\tau] \\ A\dfrac{t-T_c}{\tau-T_c} \text{ при } t\in[\tau,T_c] \\ 0 \text{ при других } t\end{cases}$
2.		$s(t)=\begin{cases}A\cdot e^{-\frac{t}{\tau}} \text{ при } t\in[0,T_c] \\ 0 \text{ при других } t\end{cases}$
3.		$s(t)=\begin{cases}A\cdot\sin\left(\dfrac{\pi}{T_c}t\right) \text{ при } t\in[0,T_c] \\ 0 \text{ при других } t\end{cases}$
4.		$s(t)=\begin{cases}\dfrac{4A}{T_c^2}t^2 \text{ при } t\in\left[0,\dfrac{T_c}{2}\right] \\ \dfrac{4A}{T_c^2}(t-T_c)^2 \text{ при } t\in\left[\dfrac{T_c}{2},T_c\right] \\ 0 \text{ при других } t\end{cases}$
5.		$s(t)=\begin{cases}\dfrac{2A}{T_c}t \text{ при } t\in\left[0,\dfrac{T_c}{2}\right] \\ A \text{ при } t\in\left[\dfrac{T_c}{2},T_c\right] \\ 0 \text{ при других } t\end{cases}$

6.		$s(t)=\begin{cases}\frac{A}{2} \text{ при } t\in\left[0,\frac{T_c}{2}\right]\\ A \text{ при } t\in\left[\frac{T_c}{2},T_c\right]\\ 0 \text{ при других } t\end{cases}$
7.		$s(t)=\begin{cases}\frac{3A}{T_c} \text{ при } t\in\left[0,\frac{T_c}{3}\right]\\ A \text{ при } t\in\left[\frac{T_c}{3},\frac{2T_c}{3}\right]\\ 3A\left(1-\frac{t}{T_c}\right) \text{ при } t\in\left[\frac{2T_c}{3},T_c\right]\\ 0 \text{ при других } t\end{cases}$
8.		$s(t)=\begin{cases}A\cdot\cos\left(\frac{\pi}{2T_c}t\right) \text{ при } t\in[0,T_c]\\ 0 \text{ при других } t\end{cases}$
9.		$s(t)=\begin{cases}A\left[1-\cos\left(\frac{\pi}{T_c}t\right)\right] \text{ при } t\in\left[0,\frac{T_c}{2}\right]\\ A\cos\left[\pi\left(\frac{t}{T_c}-\frac{1}{2}\right)\right] \text{ при } t\in\left[\frac{T_c}{2},T_c\right]\\ 0 \text{ при других } t\end{cases}$
10.		$s(t)=\begin{cases}\frac{2A}{T_c}t \text{ при } t\in\left[0,\frac{T_c}{2}\right]\\ 0 \text{ при других } t\end{cases}$

Значение параметров амплитуды A и длительности T_c и τ подбирается студентом самостоятельно.

Лабораторная работа №7: Вывод графической информации на ЖК-дисплей

Цель работы: научиться использовать устройство ЖК дисплея совместно с микроконтроллером для вывода символьной и графической информации.

Задание

Реализовать устройство вывода 3-х информационных сообщений на ЖК дисплей.

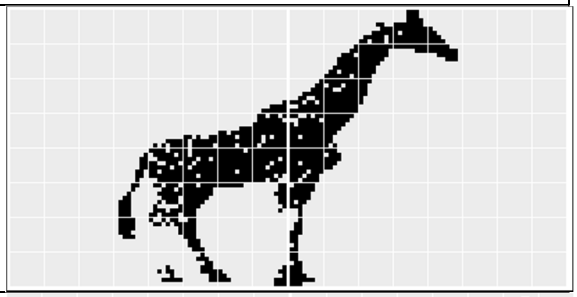
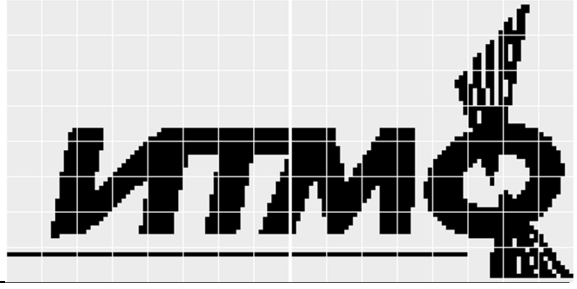
Выбор информационного сообщение осуществляется с помощью кнопок KEY1, KEY2, KEY3.

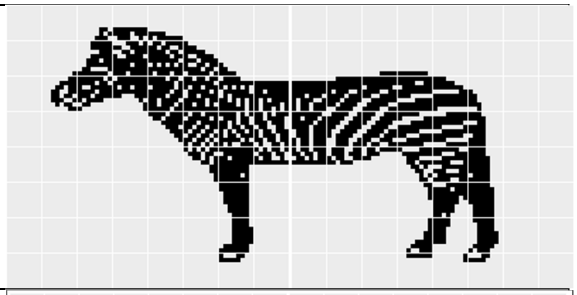
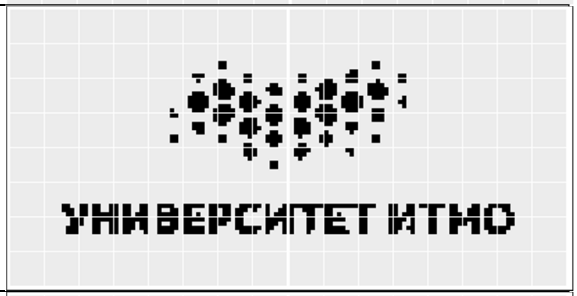
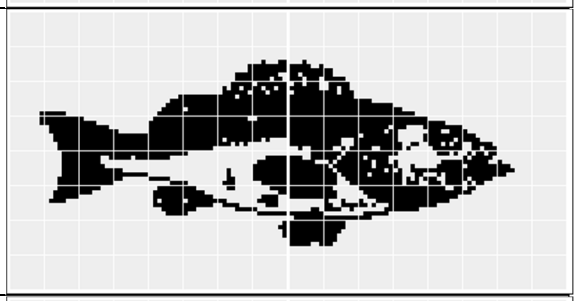
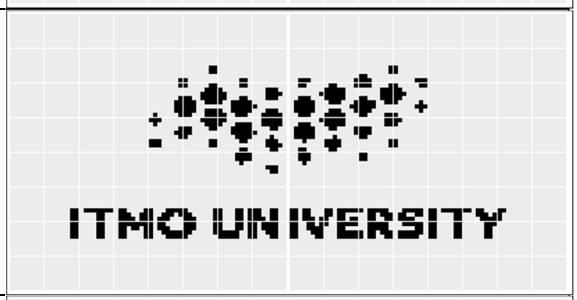
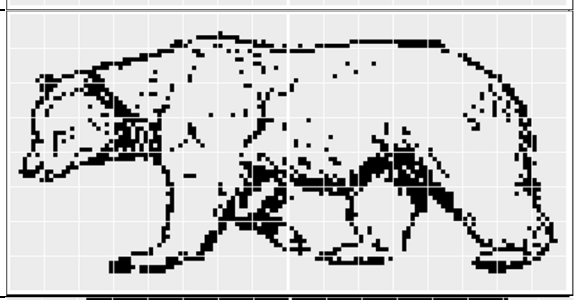
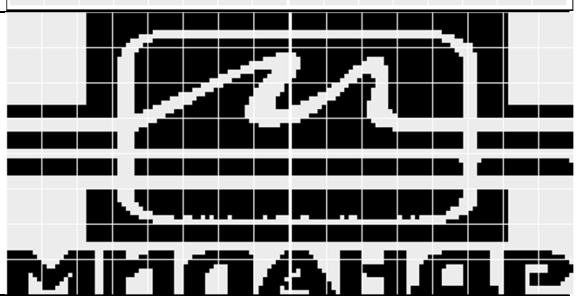
Информационное сообщение №3 подготовить с помощью графического редактора KS0108.

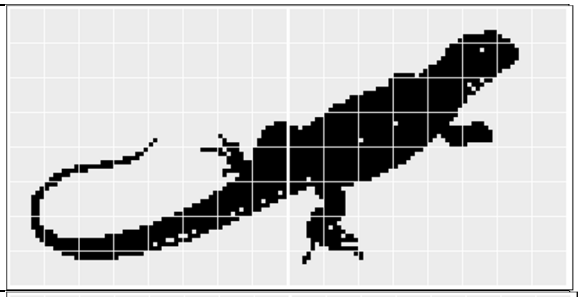
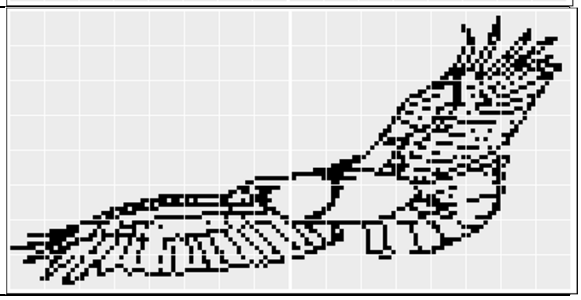
Содержание отчета

1. Титульный лист, содержащий название работы, номер варианта индивидуального задания, фамилия, имя, отчество номер группы и логин студента.
2. Цель работы
3. Краткие теоретические сведения.
4. Электрическая принципиальная схема устройства.
5. Код программы
6. Выводы.

Вариант индивидуального задания

№	Информационное сообщение №1	Информационное сообщение №2	Информационное сообщение №3
1.	Фамилия студента	«Жирафчик любит морковку»	
2.		«МИЛАНДР»	

3.		«Группа №» XXXX	
4.		«ИТМО. ОЭПиС»	
5.		«ITMO University»	
6.		«Персиковые человечки»	
7.		«Мишка косолапый»	
8.		«OEPS! ITMO!»	

9.		«Кафедра ОЭПиС»	
10.		«Университет ИТМО»	

Исходные изображения хранятся в директории ««…/Microntroller_programming/Images»

Список литературы

1. Шпак Ю.А. Программирование на языке С для AVR и PIC микроконтроллеров.- К.: «МК-Пресс», М: Издательский дом «Додэка-XXI» 2007 -400 с.

2. Г.С. Воробьева Микроконтроллеры семейства AVR Лабораторный практикум– Томск, 2009

3. Доронин, И.С. Микроконтроллеры AVR : сб. лаб. работ / И.С. Доронин, К.Н. Окишев. – Хабаровск : Изд-во ДВГУПС, 2013. – 122 с

Содержание

1 Архитектура микроконтроллеров серии 1986ВЕ9х 5
2 Компилятор и программная среда разработки 19
3 Основы программирование на языке С..... 28
4 Примеры программ для микроконтроллеров серии 1986ВЕ9х 51
5 Лабораторный отладочный макет 69
Меры безопасности при работе с лабораторным отладочным макетом 74
Лабораторная работа №1: Управление светодиодом 74
Лабораторная работа №2: Логические выражения 77
Лабораторная работа №3: Реализация протокола обмена данных по интерфейсу RS-232 80
Лабораторная работа №4: Оцифровка и фильтрация аналогового сигнала ... 82
Лабораторная работа №5: Обмен данными по интерфейсу I2C/SPI/Microwire 83
Лабораторная работа №6: Генерация аналогового сигнала..... 84
Лабораторная работа №7: Вывод графической информации на ЖК-дисплей 87
Список литературы 90

Миссия университета – генерация передовых знаний, внедрение инновационных разработок и подготовка элитных кадров, способных действовать в условиях быстро меняющегося мира и обеспечивать опережающее развитие науки, технологий и других областей для содействия решению актуальных задач.

КАФЕДРА ОПТИКО-ЭЛЕКТРОННЫХ ПРИБОРОВ И СИСТЕМ И ЕЕ НАУЧНО-ПЕДАГОГИЧЕСКАЯ ШКОЛА

Кафедра создавалась в 1937-38 годах и существовала под следующими названиями:

с 1938 по 1958 год - кафедра военных оптических приборов;

с 1958 по 1967 год - кафедра специальных оптических приборов;

с 1967 по 1992 год - кафедра оптико-электронных приборов;

с 1992 года - кафедра оптико-электронных приборов и систем.

Кафедру возглавляли:

с 1938 по 1942 год - профессор К.Е. Солодилов;

с 1942 по 1945 год профессор А.Н. Захарьевский (по совместительству);

с 1945 по 1946 год - профессор М.А. Резунов;

с 1947 по 1972 год - профессор С.Т. Цуккерман;

с 1972 по 1992 год - заслуженный деятель науки и техники РСФСР, профессор Л.Ф. Порфирьев;

с 1992 по 2007 год - заслуженный деятель науки РФ, профессор Э.Д. Панков.

с 2007 года по настоящее время - почетный работник высшего профессионального образования, профессор В.В. Коротаев.

1938 по 1970 кафедра входила в состав оптического факультета.

В 1970 году кафедра вошла в состав факультета оптико электронного приборостроения, который в 1976 году был переименован в инженерно-физический факультет.

В 1998 г кафедра вошла в состав факультета оптико-информационных систем и технологий.

В 2015 году кафедра вошла в состав факультета лазерной и световой инженерии

Кафедра оптико-электронных приборов и систем (ОЭПиС) осуществляет подготовку профессионалов в области создания оптико-электронных и видеоинформационных приборов и систем, а также в области разработки их программного обеспечения.

Результаты научных исследований кафедры докладываются на ведущих мировых научных форумах, публикуются в виде научных статей и монографий.

Приборы, разработанные на кафедре, поставляются на предприятия России и на предприятия других стран.

Этот уникальный опыт передается нашим студентам.

На кафедре работают 6 докторов наук, профессоров, однако большую часть коллектива составляют молодые люди в возрасте от 18 до 35 лет, в том числе 11 молодых кандидатов наук.

Коллектив кафедры Оптико-электронных приборов и систем (ОЭПиС) – сформировавшаяся научная и научно-педагогическая школа, существующая с 1938 года.

За эти годы были подготовлены более тысячи специалистов, более ста докторов и кандидатов наук.

С 2007 г. заведующим кафедрой является почетный работник высшего профессионального образования Российской Федерации, профессор В.В. Коротаев.

Только в период с 2007 по 2015 год на кафедре были защищены 28 диссертаций на соискание ученой степени кандидата технических наук.

В 2012 году научно-педагогическая школа кафедры ОЭПиС «Оптико-электронное приборостроение» была внесена в реестр ведущих научных и научно-педагогических школ Санкт-Петербурга.

Подробная информация о кафедре ОЭПиС имеется на сайте кафедры: http://oep.ifmo.ru/

Васильев Александр Сергеевич
Лашманов Олег Юрьевич
Пантюшин Антон Валерьевич

Основы программирования микроконтроллеров

Учебно-методическое пособие

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49