Министерство сельского хозяйства Российской Федерации
Департамент научно-технологической политики и образования
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Волгоградский государственный аграрный университет»

Кафедра «Электрооборудование и электрохозяйство предприятий
АПК»

А.П. Евдокимов
Л.Л. Владимиров

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРА К1986ВЕ92QI КОМПАНИИ «МИЛАНДР»

Лабораторный практикум
по дисциплине «Электроника и микропроцессорная техника»
для студентов, обучающихся по направлению подготовки
13.03.02 Электроэнергетика и электротехника,
профили: «Электроснабжение» и «Релейная защита и автоматизация
электроэнергетических систем»
(все формы обучения)

Волгоград
Волгоградский ГАУ
2018

УДК 621.38
ББК 32.85
Е-15

Рецензенты:
доктор технических наук, заведующий кафедрой «Системы автоматизированного проектирования и поискового конструирования» ФГБОУ ВО «Волгоградский государственный технический университет» *М.В. Щербаков;* доктор технических наук, профессор кафедры «Электрооборудование и электрохозяйство предприятий АПК» ФГБОУ ВО «Волгоградский государственный аграрный университет» *В.Г. Рябцев*

Евдокимов, Алексей Петрович
Е-15 Программирование микроконтроллера К1986ВЕ92QI компании «Миландр»: лабораторный практикум по дисциплине «Электроника и микропроцессорная техника» для студентов, обучающихся по направлению подготовки 13.03.02 Электроэнергетика и электротехника, профили «Электроснабжение» и «Релейная защита и автоматизация электроэнергетических систем» (все формы обучения) / А.П. Евдокимов, Л.Л. Владимиров. – Волгоград: ФГБОУ ВО Волгоградский ГАУ, 2018. – 76 с.

Лабораторный практикум содержит описание 9 лабораторных работ по изучению новейшего 32-разрядного отечественного микроконтроллера К1986ВЕ92QI, выпускаемого АО «ПКК Миландр» (Москва, Зеленоград). Каждая лабораторная работа содержит краткие теоретические сведения, необходимые студенту при подготовке к выполнению лабораторной работы. Общая продолжительность лабораторных работ – 18 академических часов.

Издание подготовлено в рамках Договора о сотрудничестве между компанией «Миландр» и Волгоградским ГАУ. Проведение лабораторных работ осуществляется на отладочных платах, переданных компанией «Миландр» Университету на безвозмездной основе.

УДК 621.38
ББК 32.85

ВВЕДЕНИЕ

Микроконтроллер – микросхема, предназначенная для управления электронными устройствами. По сути, это однокристальная микро-ЭВМ, способная выполнять относительно несложные задачи. Микроконтроллеры относят к так называемым встраиваемым системам, поскольку они размещаются непосредственно в технологическом оборудовании и предназначены для управления им.

Микроконтроллеры очень широко применяются в электроэнергетике. Это микропроцессорные устройства сигнализации, включая контроллер мнемощита; микропроцессорные устройства измерения электрических величин, в том числе, счетчики электрической энергии; микроконтроллеры, диагностирующие состояния электрооборудования: электрических машин и аппаратов, высоковольтных вводов, устройств определения места повреждения линии; устройства противоаварийной автоматики.

Долгое время микроконтроллеры были восьмиразрядными. Программы для восьмиразрядных микроконтроллеров писались на языке ассемблера, они получались компактными и как нельзя лучше подходили для целей управления в реальном масштабе времени. Программы, созданные на языке Си или других языках высокого уровня, занимали больший объем памяти, которая у восьмиразрядных микроконтроллеров была невелика.

По мере увеличения сложности решаемых задач увеличивалась разрядность, а также объем памяти и быстродействие микроконтроллеров. В наши дни всё большую популярность среди инженеров приобретают 32-разрядные микроконтроллеры, обладающие относительно большим объемом памяти, высоким быстродействием и разветвленной периферией.

Написание программы на языке ассемблера объемом более нескольких килобайт требует больших усилий программиста, хотя программа в итоге и получается компактной, быстродействующей. Если память программ микроконтроллера составляет сотни килобайт, то с целью сокращения трудозатрат и времени программирования разумно перейти на язык высокого уровня, такого, например, как Си, а создание машинного кода переложить на компилятор. Более того, на языке Си написано большое количество программ, которые по тематикам объединяются в библиотеки: библиотека для настройки таймера, библиотека для работы с жидкокристаллическим дисплеем и т.п. Изучение библиотек, конечно, потребует дополнительного времени, но работающую программу в итоге мы получим быстрее. Этот путь мы и выберем.

ЛАБОРАТОРНАЯ РАБОТА 1.

АРХИТЕКТУРА МИКРОКОНТРОЛЛЕРА

ЦЕЛЬ РАБОТЫ

Изучить архитектуру микроконтроллера К1986ВЕ92QI и устройство лабораторного стенда.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Микроконтроллер К1986ВЕ92QI российской компании «Миландр» (АО «ПКК Миландр») имеет следующие характеристики.

Ядро:

– ARM 32-битное RISC-ядро Cortex™-M3 ревизии 2.0, тактовая частота до 80 МГц;

– блок аппаратной защиты памяти MPU;

– умножение за один цикл, аппаратная реализация деления.

Память:

– встроенная энергонезависимая Flash-память программ размером 128 Кбайт;

– встроенное ОЗУ размером 32 Кбайт;

– контроллер внешней шины с поддержкой микросхем памяти СОЗУ, ПЗУ, NAND Flash.

Питание и тактовая частота:

– внешнее питание 2,2 ÷ 3,6 В;

– встроенный регулируемый стабилизатор напряжения на 1,8 В для питания ядра;

– встроенные схемы контроля питания;

– встроенный домен с батарейным питанием;

– встроенные подстраиваемые RC-генераторы 8 МГц и 40 кГц;

– внешние кварцевые резонаторы на 2 ÷ 16 МГц и 32 кГц;

– встроенный умножитель тактовой частоты PLL для ядра;

– встроенный умножитель тактовой частоты PLL для USB.

Режим пониженного энергопотребления:

– режимы Sleep, Deep Sleep и Standby;

– батарейный домен с часами реального времени и регистрами аварийного сохранения.

Аналоговые модули:

– два 12-разрядных АЦП (до 8 каналов);

– температурный датчик;

– двухканальный 12-разрядный ЦАП;

– встроенный компаратор.

Периферия:

– контроллер DMA с функциями передачи периферия – память, память – память;

– два контроллера CAN интерфейса;

– контроллер USB интерфейса с функциями работы Device и Host;

– контроллеры интерфейсов UART, SPI, I2C;

– три 16-разрядных таймер-счетчика с функциями ШИМ и регистрации событий;

– 43 пользовательских линий ввода-вывода.

Отладочные интерфейсы: SWD и JTAG.

Возможно, пока вам понятны не все термины, но со временем многое прояснится.

Структурная схема (архитектура) микроконтроллера К1986ВЕ92QI показана на рисунке 1.1.

Микроконтроллер К1986ВЕ92QI построен на базе высокопроизводительного процессорного RISC-ядра ARM Cortex-M3. Для связи с процессорным ядром имеется три шины:

– I-Code – шина выборки инструкций (команд);

– D-Code – шина выборки данных, расположенных в коде программы;

– S-Bus – шина выборки данных, расположенных в области оперативного запоминающего устройства (RAM).

В микроконтроллере реализован контроллер прямого доступа в память (DMA), который осуществляет выборку через шину DMA Bus.

Память программ (EEPROM) предназначена для хранения основной рабочей программы. Память EEPROM допускает многократную перезапись, ее объем составляет 128 Кбайт. Информация в памяти программ сохраняется и после отключения питания.

Оперативное запоминающее устройство (RAM) служит для хранения входных, выходных и промежуточных данных, обрабатываемых процессором. Объем RAM – 32 Кбайт.

Постоянное запоминающее устройство, или ROM, предназначено для хранения программы запуска микроконтроллера. В ходе выполнения этой программы определяется режим запуска основной программы или переход в режим программирования микроконтроллера.

EXTERNAL BUS (Внешняя шина) предназначена для хранения кода программ во внешних микросхемах памяти, подсоединенных к внешней системной шине.

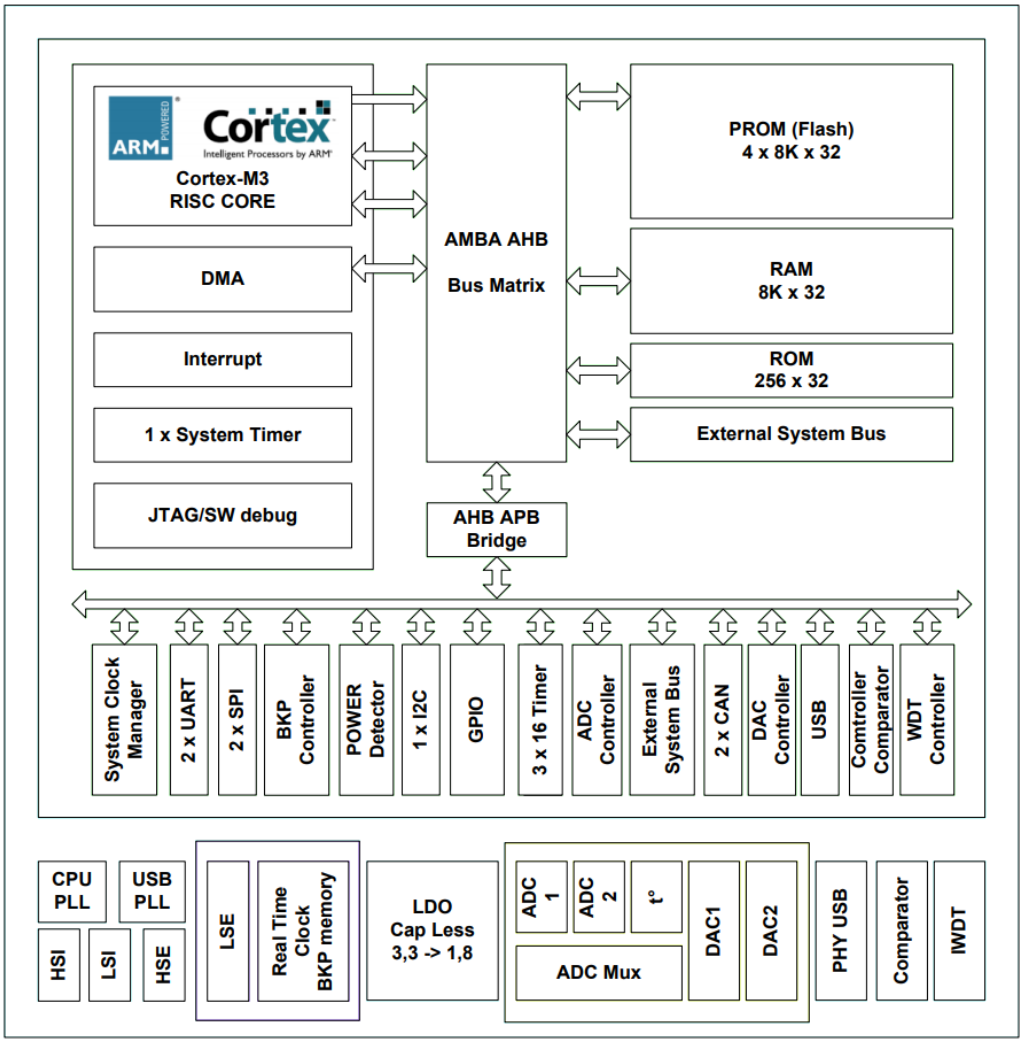

Рисунок 1.1 – Архитектура микроконтроллера К1986ВЕ92QI

Блок BUS MATRIX предназначен для переключения системных шин I-Code, D-Code, S-Bus и DMA Bus между различными областями памяти.

Interrupt – контроллер прерываний.

Системный таймер System Timer – 24-разрядный счетчик, используется операционными системами реального времени для подсчета тактов или как обычный счетчик.

JTAG/SW debug – блок, позволяющий вести отладку программы по интерфейсам JTAG или SW.

Связь шин ядра и периферии осуществляется через блок AHB/APB Bridge.

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 си-

стемной шины, что позволяет работать с внешними микросхемами статического ОЗУ и ПЗУ, NAND Flash-памятью и другими внешними устройствами.

Микроконтроллер содержит, помимо 24-разрядного системного таймера, три 16-разрядных таймера с 4 каналами схем захвата и широтно-импульсной модуляции (ШИМ), а также два сторожевых таймера.

Кроме того, как уже отмечалось, в состав микроконтроллера входят:

– два 12-разрядных высокоскоростных 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); внешние генераторы: HSE (2…16 МГц) и LSE (32 кГц); две схемы умножения тактовой частоты PLL: для ядра и USB-интерфейса.

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 участия процессорного ядра.

Встроенный регулятор, предназначенный для формирования питания внутренней цифровой части, формирует напряжение 1,8 В и не требует дополнительных внешних элементов. Таким образом, для работы микроконтроллера достаточно одного внешнего напряжения питания в диапазоне от 2,2 до 3,6 В.

Также в микроконтроллерах реализован батарейный домен, работающий от внешней батареи, который предназначен для обеспечения функций часов реального времени и сохранения некоторого объёма данных при отсутствии основного питания. Встроенные детекторы напряжения питания могут отслеживать уровень напряжения питания на батарее. Аппаратные схемы сброса при снижении напряжения питания за допустимые пределы автоматически перезапускают микроконтроллер.

Расположение выводов микроконтроллера показано на рисунке 1.2. Для обмена данными с внешними устройствами микроконтроллер имеет 6 портов ввода/вывода MDR_PORTx. Порт представляет собой 16-разрядный регистр, при этом один и тот же вывод (линия) может использоваться различными функциональными блоками.

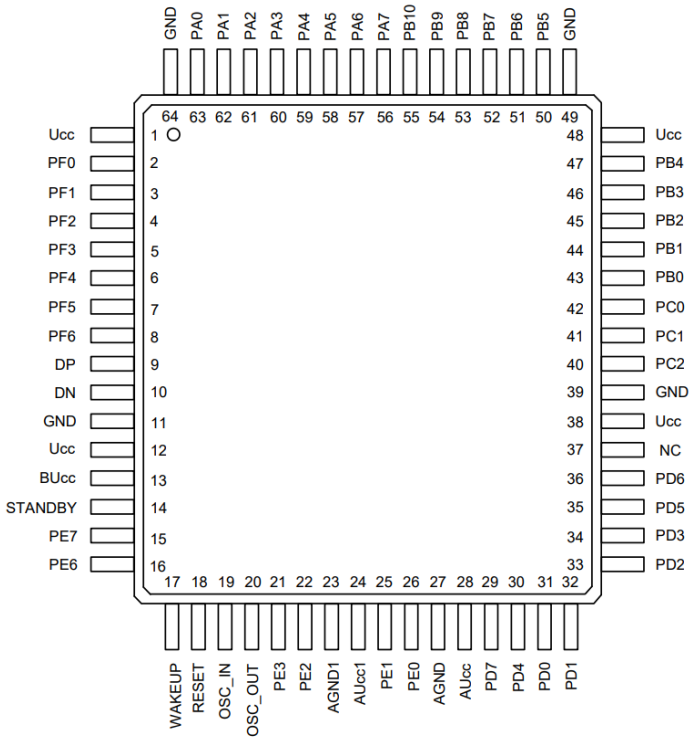

Рисунок 1.2 – Расположение выводов микроконтроллера К1986ВЕ92QI

Назначение линий портов микроконтроллера приведено в таблице 1.1. Для того, чтобы линии порта перешли под управление того или иного периферийного блока, необходимо задать для выбранных линий выполняемую функцию и настройки. Обратим внимание на то, что микроконтроллер имеет только 64 вывода, поэтому не каждая линия порта может быть связана с выводом микросхемы.

Таблица 1.1 – Функции линий портов микроконтроллера К1986ВЕ92QI

Линия	Вывод	Цифровая функция			Аналоговая функция
		Основная	Альтернат.	Переопред.	
1	2	3	4	5	6
Порт А					
PA0	63	DATA0	EXT_INT1	–	–
PA1	62	DATA1	TMR1_CH1	TMR2_CH1	–
PA2	61	DATA2	TMR1_CH1N	TMR2_CH1N	–
PA3	60	DATA3	TMR1_CH2	TMR2_CH2	–
PA4	59	DATA4	TMR1_CH2N	TMR2_CH2N	–
PA5	58	DATA5	TMR1_CH3	TMR2_CH3	–
PA6	57	DATA6	CAN1_TX	UART1_RXD	–
PA7	56	DATA7	CAN1_RX	UART1_TXD	–

Окончание таблицы 1.1

1	2	3	4	5	6
Порт B					
PB0	43	DATA16	TMR3_CH1	UART1_TXD	–
PB1	44	DATA17	TMR3_CH1N	UART2_RXD	–
PB2	45	DATA18	TMR3_CH2	CAN1_TX	–
PB3	46	DATA19	TMR3_CH2N	CAN1_RX	–
PB4	47	DATA20	TMR3_BLK	TMR3_ETR	–
PB5	50	DATA21	UART1_TXD	TMR3_CH3	–
PB6	51	DATA22	UART1_RXD	MR3_CH3N	–
PB7	52	DATA23	nSIROUT1	TMR3_CH4	–
PB8	53	DATA24	COMP_OUT	TMR3_CH4N	–
PB9	54	DATA25	nSIRIN1	EXT_INT4	–
PB10	55	DATA26	EXT_INT2	nSIROUT1	–
Порт C					
PC0	42	–	SCL1	SSP2_FSS	–
PC1	41	OE	SDA1	SSP2_CLK	–
PC2	40	WE	TMR3_CH1	SSP2_RXD	–
Порт D					
PD0	31	TMR1_CH1N	UART2_RXD	TMR3_CH1	ADC0_REF+
PD1	32	TMR1_CH1	UART2_TXD	TMR3_CH1N	ADC1_REF-
PD2	33	BUSY1	SSP2_RXD	TMR3_CH2	ADC2
PD3	34	–	SSP2_FSS	TMR3_CH2N	ADC3
PD4	30	TMR1_ETR	nSIROUT2	TMR3_BLK	ADC4
PD5	35	CLE	SSP2_CLK	TMR2_ETR	ADC5
PD6	36	ALE	SSP2_TXD	TMR2_BLK	ADC6
PD7	29	TMR1_BLK	nSIRIN2	UART1_RXD	ADC7
Порт E					
PE0	26	ADDR16	TMR2_CH1	CAN1_RX	DAC2_OUT
PE1	25	ADDR17	TMR2_ CH1N	CAN1_TX	DAC2_REF
PE2	22	ADDR18	TMR2_CH3	TMR3_CH1	COMP_IN1
PE3	21	ADDR19	TMR2_CH3N	TMR3_CH1N	COMP_IN2
PE6	16	ADDR22	CAN2_RX	TMR3_CH3	OSC_IN32
PE7	15	ADDR23	CAN2_TX	TMR3_CH3N	OSC_OUT32
Порт F					
PF0	2	ADDR0	SSP1_TXD	UART2_RXD	–
PF1	3	ADDR1	SSP1_CLK	UART2_TXD	–
PF2	4	ADDR2	SSP1_FSS	CAN2_RX	–
PF3	5	ADDR3	SSP1_RXD	CAN2_TX	–
PF4	6	ADDR4	–	–	–
PF5	7	ADDR5	–	–	–
PF6	8	ADDR6	TMR1_CH1	–	–

Рассмотрим в качестве примера Порт А. Только 8 линий порта из 16 имеют физическую связь с выводами микросхемы: PA0, PA1…PA7. Если весь порт настроить на выполнение основной функции, то с позиции внешнего устройства он будет представлять собой восьмиразрядный регистр, в который по восьми параллельным линиям можно записывать восьмиразрядный двоичный код либо считывать его. Впрочем, настраивать можно каждую линию в отдельности. Например, линия PA0 порта может быть настроена на выполнение альтернативной функции EXT_INT1, что позволит осуществлять прерывание основной программы микроконтроллера от внешнего устройства, а вот работа с аналоговыми сигналами в данном порту не предусмотрена.

Большинство линий портов помимо основных и альтернативных функций могут быть настроены и на выполнение так называемых «переопределенных» функций – это просто третий вариант настройки линии порта на выполнение цифровой функции. С аналоговыми сигналами могут работать только порты D и E. Также имеются линии, имеющие только одну функцию – это PF4 и PF5 порта F.

Остальные выводы микроконтроллера используются следующим образом:

1, 12, 38, 48 – основное питание 2,2…3,6 В;

28 – питание аналого-цифрового преобразователя, цифро-аналогового преобразователя и компаратора 2,4…3,6 В;

24 – питание схем умножения тактовой частоты PLL 2,2…3,6 В;

13 – питание батарейного домена 1.8…3,6 В (используется при отсутствии основного питания);

11, 39, 49, 64, 23, 27 – общий;

18 – сигнал внешнего сброса;

17 – сигнал внешнего выхода из режима Standby (режима пониженного энергопотребления);

14 – флаг режима Standby;

19 – вход генератора HSE;

20 – выход генератора HSE;

9 – шина USB D+;

10 – шина USB D–;

37 – не используется.

Лабораторный стенд для микроконтроллера К1986ВЕ92QI

Внешний вид отладочной платы с элементами управления и коммутации показан на рисунке 1.3.

Рассмотрим элементы отладочной платы, которые прежде всего потребуются нам в работе.

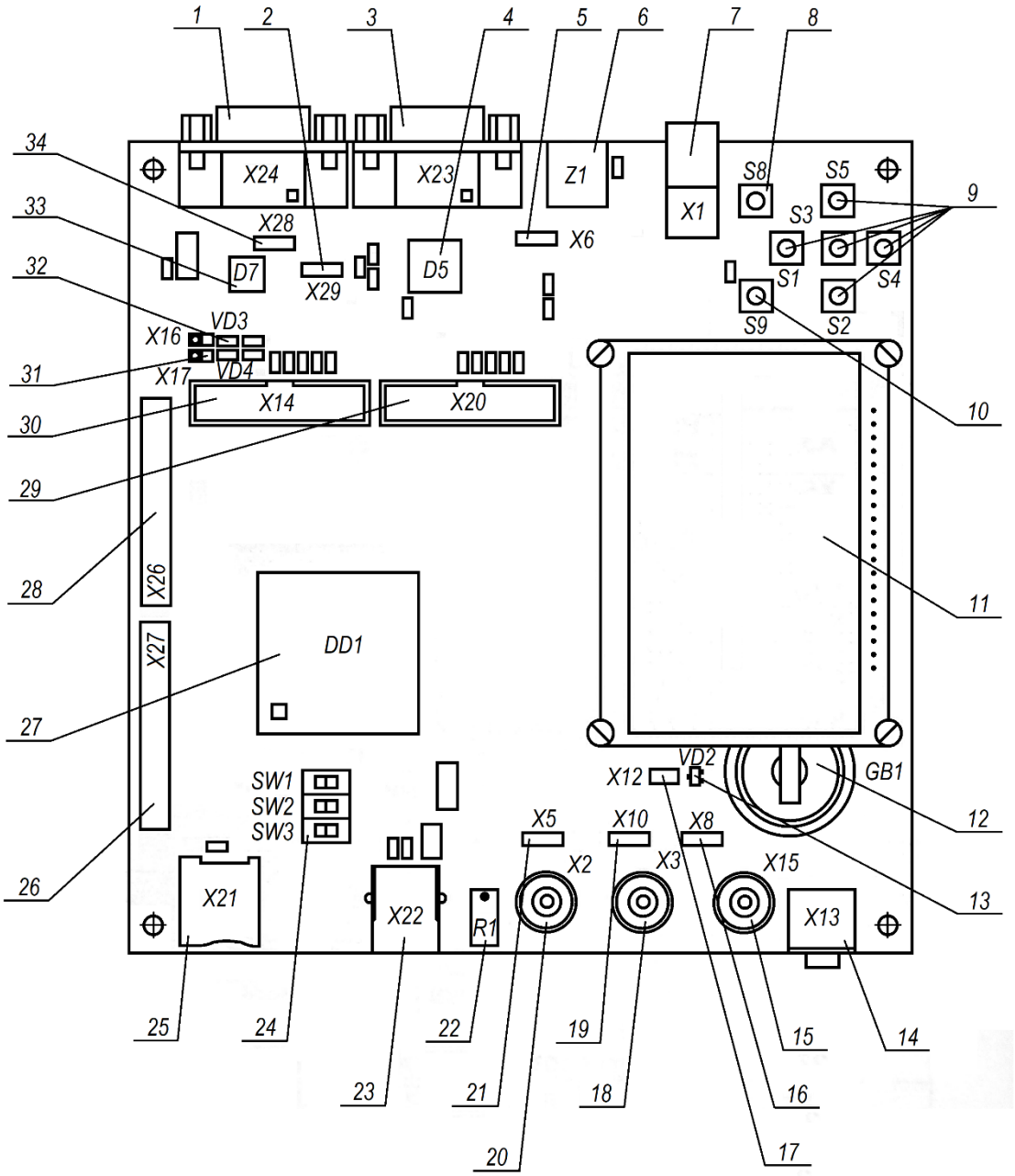

Рисунок 1.3 – Расположение элементов управления
и коммутации отладочной платы

Кнопки S1 «UP», S2 «LEFT», S3 «SELECT», S4 «DOWN», S5 «RIGHT» могут быть нами запрограммированы. Кнопка S1 подключена к линии PB5 порта B, кнопка S2 – к линии PE3 порта E, кнопка S3 – к линии PC2 порта C, кнопка S4 – к линии PE1 порта E, кнопка S5 – к линии PB6 порта B.

Кнопка S8 «RESET» предназначена для аппаратного сброса.

Кнопка S9 «WAKEUP» служит для выхода микроконтроллера из режима пониженного энергопотребления STANDBY.

Светодиоды VD3 и VD4 (поз. 32 на рисунке 1.3) подключены через ограничивающие ток резисторы к линиям PC0 и PC1 порта C и могут служить для простейшей индикации.

Жидкокристаллический модуль (поз. 11 на рисунке 1.3) MT–12864J v.1 отечественной компании «МЭЛТ» содержит собственный контроллер управления и жидкокристаллическую панель. Модуль содержит оперативное запоминающее устройство (ОЗУ) для хранения данных, выводимых на экран, размером 64x64x2 бит. Каждой светящейся точке на экране соответствует логическая «1» в ячейке ОЗУ модуля. Таких точек на экране 8192.

Отдельно остановимся на разъемах X26 и X27 (поз. 28 и 26 на рисунке 1.3). На самой плате они обозначены, соответственно, как XP13 и XP14, а их выводы имеют непосредственную связь с линиями портов микроконтроллера. Подключение портов микроконтроллера к разъемам X26 и X27 приведено в таблице 1.2.

Таблица 1.2 – Связь линий портов микроконтроллера с разъемами X26 и X27

Номер контакта разъема	Линия порта /питание	
	Разъем X26	Разъем X27
1	2	3
1, 2	GND	GND
3, 4	+3,3 В	+3,3 В
5	PD0	PA6
6	PD1	PA7
7	PD2	PA4
8	PD3	PA5
9	PD4	PA2
10	PD5	PA3
11	PD6	PA0
12	–	PA1
13	PB0	–
14	PB1	–
15	PB2	PE1
16	PB3	PE3
17	PB4	–
18	PB5	–
19	PB6	PF0
20	PB7	PF1

Окончание таблицы 1.2

1	2	3
21	PB8	PF2
22	PB9	PF3
23	PB10	PF4
24	PC0	PF5
25	PC1	PF6
26	PC2	–
27, 28	+5В	+5В
29, 30	GND	GND

Отметим, что многие линии портов заняты, и к ним нельзя подключать внешние устройства. Например, PD0…PD4 заняты JTAG-интерфейсом (JTAG-B), кроме того, как мы помним, ряд линий заняты кнопками S1…S5. А еще – передача данных, АЦП, ЦАП… Словом, свободных линий совсем немного.

Описание элементов отладочной платы сведено в таблицу 1.3.

Таблица 1.3 – Элементы отладочной платы

Обозначение	Описание	Поз.
1	2	3
DD1	Контактное устройство для микроконтроллера	27
D5	Приемопередатчик RS-232	4
D7	Приемопередатчик CAN	33
GB1	Батарейный отсек	12
R1	Подстроечный резистор канала 7 АЦП	22
SW1SW3	Переключатели	24
S1-S5	Кнопки UP, LEFT, SELECT, DOWN, RIGHT	9
S8	Кнопка RESET	8
S9	Кнопка WAKEUP	10
VD2	Транзистор для подключения батарейного отсека	13
VD3, VD4	Набор светодиодов для порта С	32
X1	Разъем питания 5В	7
X2	Разъем BNC внешнего сигнала канала 7 АЦП	20
X3	Разъем BNC внешнего сигнала на 1-м входе компаратора	18
X5	Разъем для установки конфигурационных перемычек	21
X6		5
X8		16
X10		19

Окончание таблицы 1.3

1	2	3
X12	Разъем для установки конфигурационных перемычек	17
X13	Разъем Audio 3,5 мм выхода ЦАП1 через звуковой усилитель	14
X14	Разъем отладки JTAG-A	30
X15	Разъем BNC выхода ЦАП-1	15
X16, X17	Разъемы для установки конфигурационных перемычек	31
X20	Разъем отладки JTAG-B	29
X21	Разъем карты памяти micro-SD	25
X22	Разъем USB-B	23
X23	Разъем интерфейса RS-232	3
X24	Разъем интерфейса CAN	1
X26	Разъем портов B, C, D микроконтроллера	28
X27	Разъем портов A, E, F микроконтроллера	26
X28	Разъем для установки конфигурационных перемычек	34
X29		2
Z1	Фильтр питания	6
–	Жидкокристаллический модуль	11

До начала работы с отладочной платой необходимо установить конфигурационные перемычки: разъем X5 – в положение «EXT_CON», разъем X6 – в положение «EXT_DC», разъем X8 – в положение «EXT_CON», разъем X10 – в положение «EXT_CON», разъем X12 – в положение «Vbat», разъем X16 – перемкнуть 1 и 2 контакты, разъем X17 – перемкнуть 1 и 2 контакты, разъем X28 – в положение «125kb/s», разъем X29 – в положение «120Ohm».

Поскольку для связи отладочной платы с программатором мы будем использовать разъем X20, то переключатели SW1, SW2 и SW3 должны быть установлены в положение «0».

Плата программатора, поставляемая компанией «Миландр» вместе с отладочной платой, имеет разъем USB для подключения к компьютеру и шлейф с разъемом для подключения к отладочной плате.

Источник электропитания преобразует переменное напряжение 176…264 В частотой 50 Гц в постоянное напряжение 5 В при токе до 2А.

ПРАКТИЧЕСКАЯ ЧАСТЬ

1. Проверьте правильность расстановки конфигурационных перемычек и переключателей на отладочной плате.

2. Аккуратно подключите шлейф программатора к разъему X20 (поз. 29 на рисунке 1.3), совместив выступающую часть разъема, соединенного с шлейфом, с прорезью разъема, закрепленного на отладочной плате.

3. Подсоедините кабель USB к программатору.

4. Включите компьютер, дождитесь загрузки операционной системы и подключите программатор с помощью кабеля к разъему USB компьютера.

5. Подключите источник питания, поставляемый с отладочной платой, к сети, затем соедините выход 5В источника питания с разъемом X1 (поз. 7 на рисунке 1.3).

6. Наблюдайте за работой микроконтроллера на отладочной плате.

Обратим внимание на то, что сейчас отладочная плата подготовлена к режиму отладки, и если отключить кабель USB от компьютера, работа микроконтроллера прекратится. Разумеется, микроконтроллер может работать автономно, но для этого программатор от платы должен быть отключен.

Внимание! Все коммутации на отладочной плате можно делать только при отключенном источнике питания.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как называется изучаемый вами микроконтроллер?
2. Перечислите основные характеристики микроконтроллера.
3. Как называется и что собой представляет порт ввода /вывода микроконтроллера?
4. Какие виды функций могут быть заданы для линий порта?
5. Опишите органы управления и коммутации отладочной платы.

ЛАБОРАТОРНАЯ РАБОТА 2.

СРЕДА ПРОГРАММИРОВАНИЯ МИКРОКОНТРОЛЛЕРА

ЦЕЛЬ РАБОТЫ

Уметь создавать проекты в среде разработки Keil µVision.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Интегрированная среда разработки Keil µVision будет использоваться для написания программ для микроконтроллера К1986ВЕ92QI на языке Си.

Работу среды программирования начнем изучать в процессе создания первого проекта.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Подключите отладочную плату к компьютеру, подайте питание на плату и запустите среду разработки Keil µVision, используя ярлык на рабочем столе.

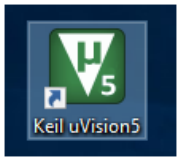

Рисунок 2.1 – Ярлык для запуска Keil µVision

В открывшемся меню (рисунок 2.2) выберите вкладку *Project*, а затем, в открывшемся окне, – *New µVision Project*.

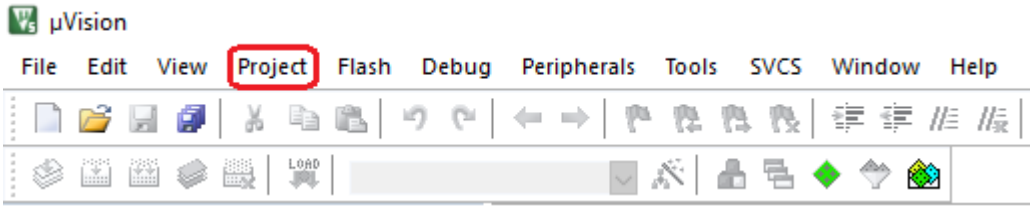

Рисунок 2.2 – Фрагмент меню и панели инструментов среды разработки Keil µVision

Пропишите путь к проекту и дайте ему имя. В нашем случае: *D:\MILANDR_PRACTICUM\1_PROBA\BEGINNING* (рисунок 2.3). Не забудьте сохранить проект.

Далее, в открывшемся окне выберите микроконтроллер (рисунок 2.4):

Milandr → MDR1986 → Cortex-M3 → MDR1986BE92 → OK.

Теперь подключите некоторые библиотеки: *Startup* – всегда, остальные – по потребности. Для этого в открывшемся окне *Manage Run-Time Environment* (рисунок 2.5) установите следующие флажки:

Device → Startup; Drivers → PORT; Drivers → RST_CLK.

Затем нажмите кнопку *OK*.

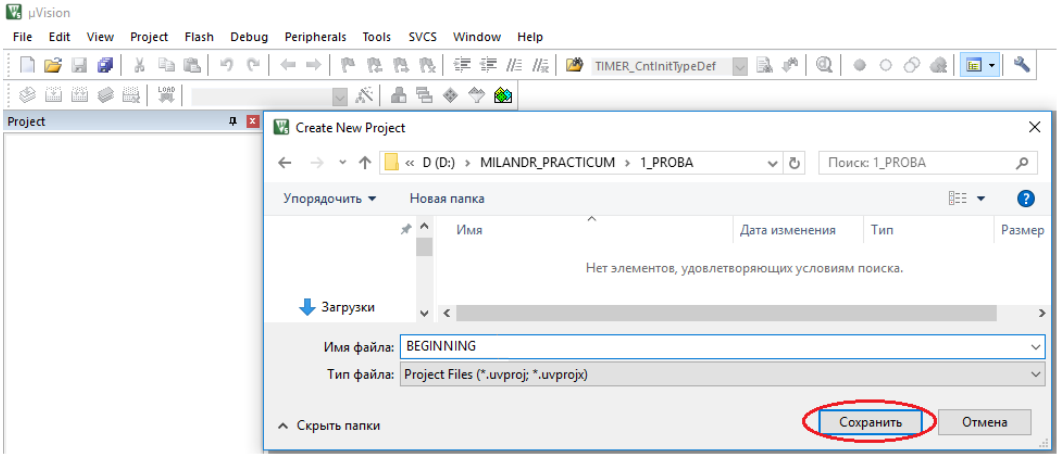

Рисунок 2.3 – Создание нового проекта: путь и имя проекта

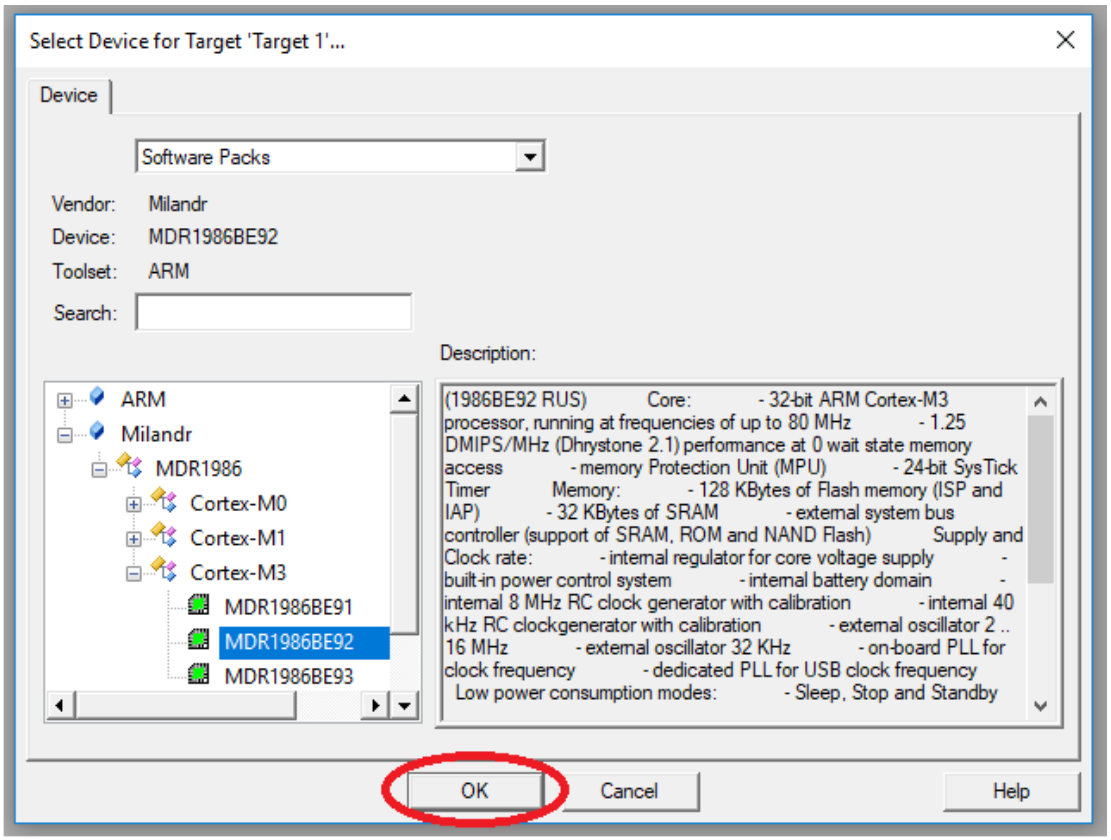

Рисунок 2.4 – Создание нового проекта: выбор микроконтроллера

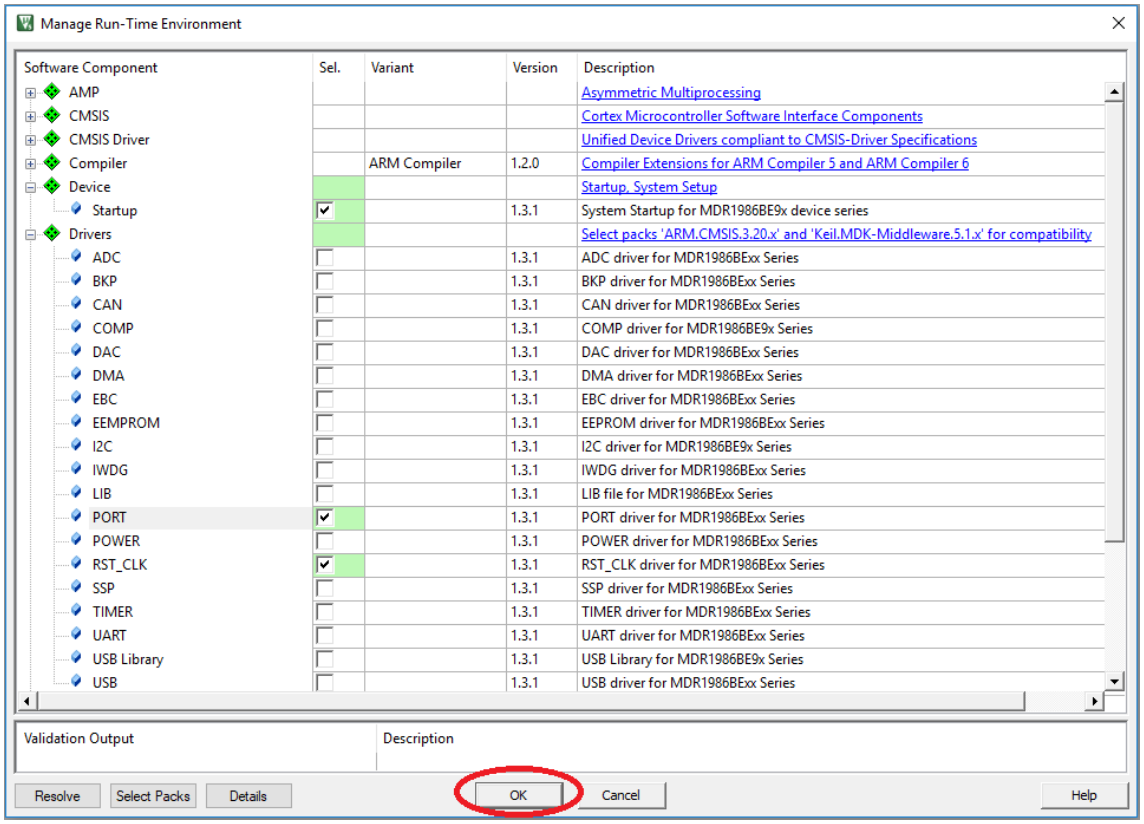

Рисунок 2.5 – Создание нового проекта: подключение библиотек

Если в дальнейшем потребуется открыть окно подключения библиотек *Manage Run-Time Environment*, достаточно нажать на значок в виде зеленого ромба на панели инструментов (рисунок 2.6).

В результате в окне Project увидим следующую структуру проекта.

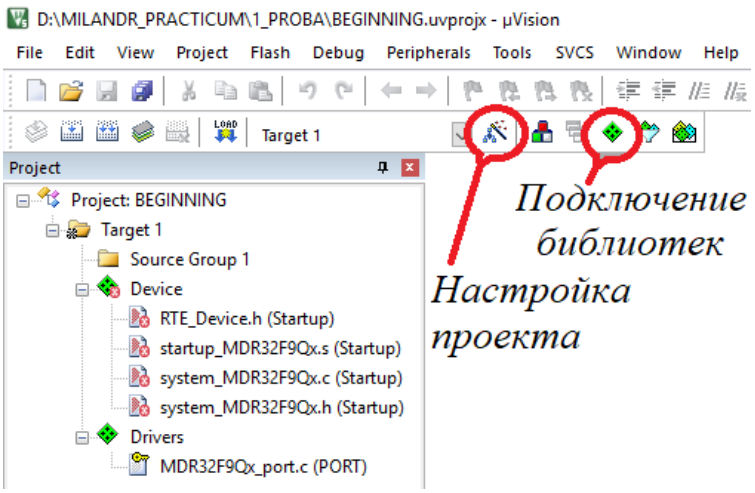

Рисунок 2.6 – Создание нового проекта: структура проекта

Приступим к настройке проекта. Для этого необходимо сделать клик по кнопке настройки проекта (рисунок 2.6). В открывшемся меню выбрать вкладку *Target* и установить частоту базового генератора микроконтроллера 8.0 МГц (рисунок 2.7).

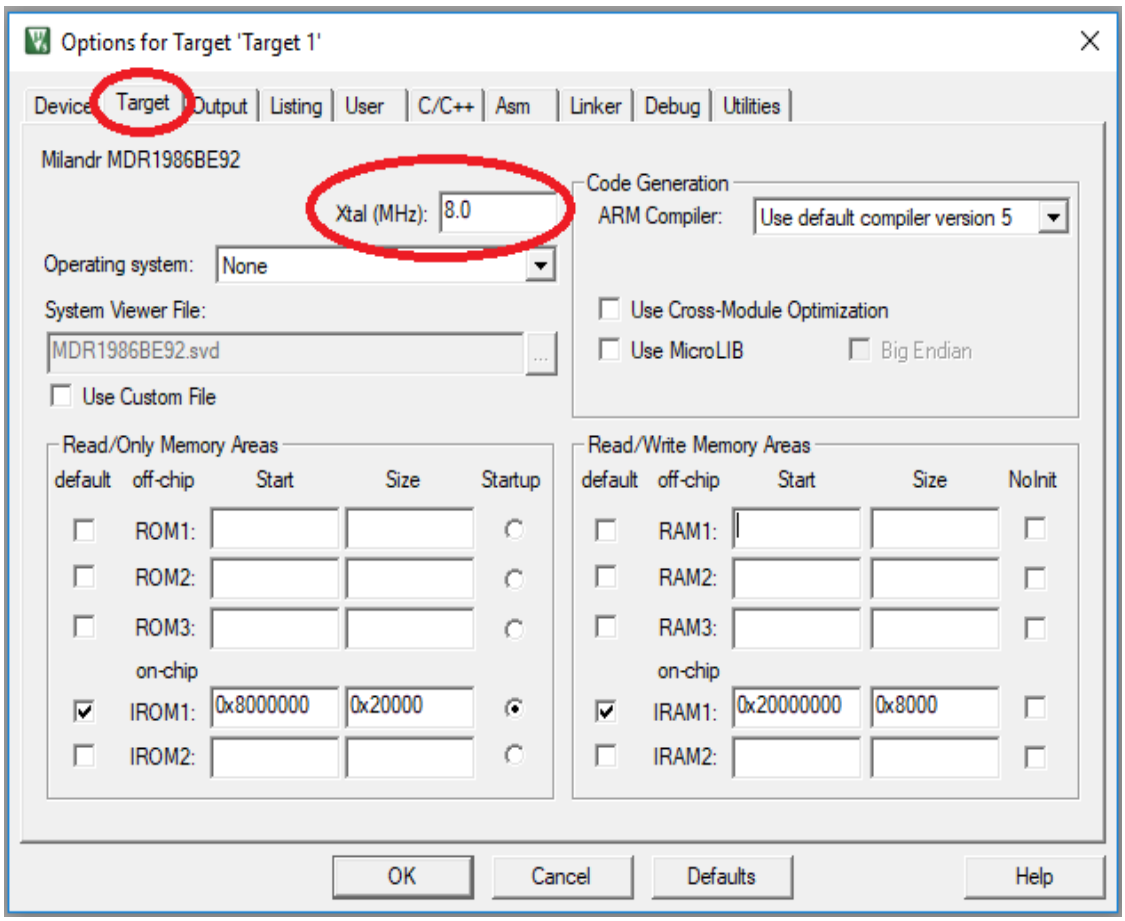

Рисунок 2.7 – Создание нового проекта: установка частоты базового генератора микроконтроллера

Выберите вкладку *Output* и поставьте галочку напротив *Create HEX File* (создать файл в шестнадцатеричных кодах), как это показано на рисунке 2.8.

Помимо библиотек, которые предлагает среда Keil µVision, нам могут потребоваться и иные библиотеки. На вкладке *C/C++* необходимо прописать путь к дополнительным библиотекам. Например, библиотека, позволяющая настраивать жидкокристаллический индикатор, будет располагаться в папке *Generic*. Путь к этой папке следует указать в поле *Include Paths* (рисунок 2.9), для чего нажать кнопку справа и выбрать директорию, где располагается библиотечный файл.

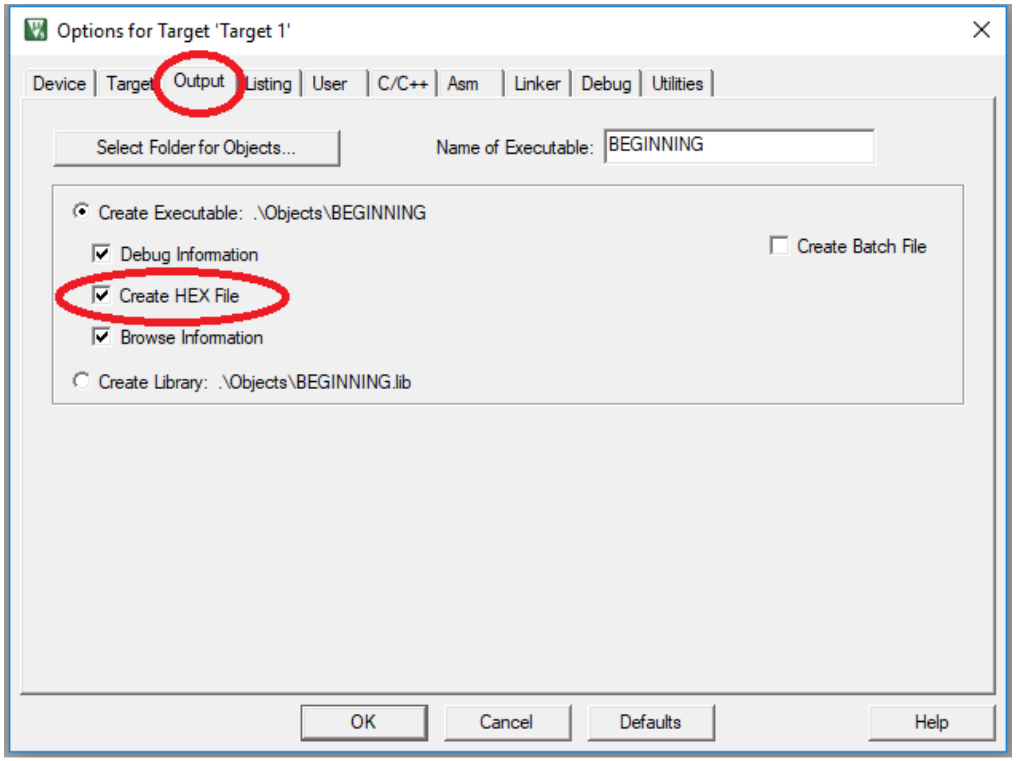

Рисунок 2.8 – Создание нового проекта:
требование создания HEX-файла

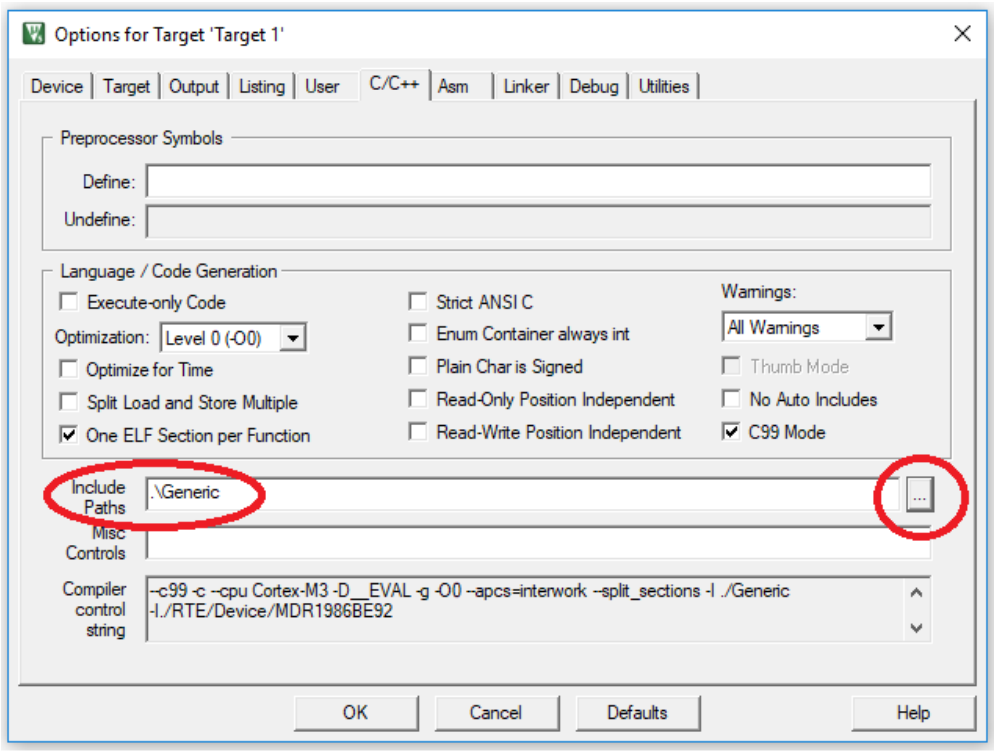

Рисунок 2.9 – Создание нового проекта: указание пути
к библиотечным файлам

На вкладке *Debug* (отладка) установить тип используемого программатора (*J-LINK/J-TRACE Cortex*) и нажать кнопку *Settings* (установки) в последовательности, показанной на рисунке 2.10.

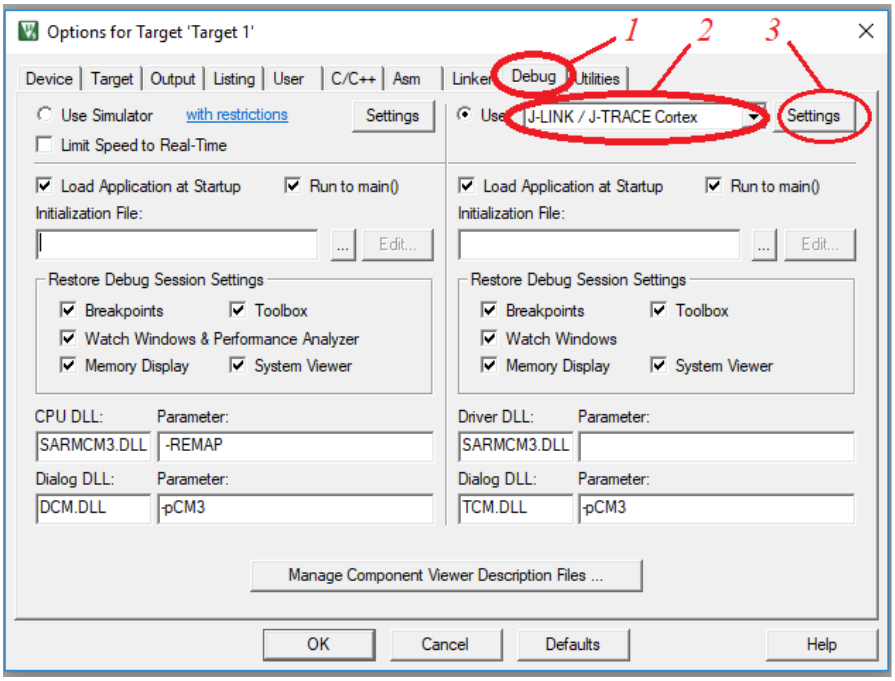

Рисунок 2.10 – Создание нового проекта: установка средств отладки

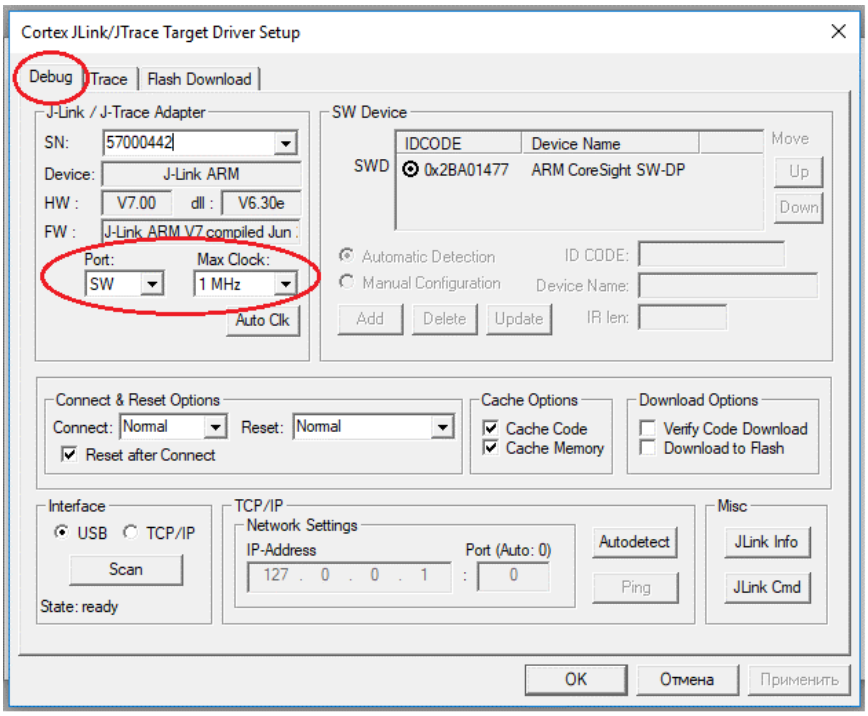

Рисунок 2.11 – Создание нового проекта: настройка программатора

На рисунке 2.11 показаны установки, которые нужно сделать на вкладке *Settings* для нормальной работы программатора. И наконец, на вкладке *Flash Download* (загрузка памяти) отметим действия, которые нужно совершить после загрузки программы во флэш-память микроконтроллера: очистка всего кристалла, запись программы в память, проверка правильности записи, автоматический запуск исполнения программы микроконтроллером (рисунок 2.12).

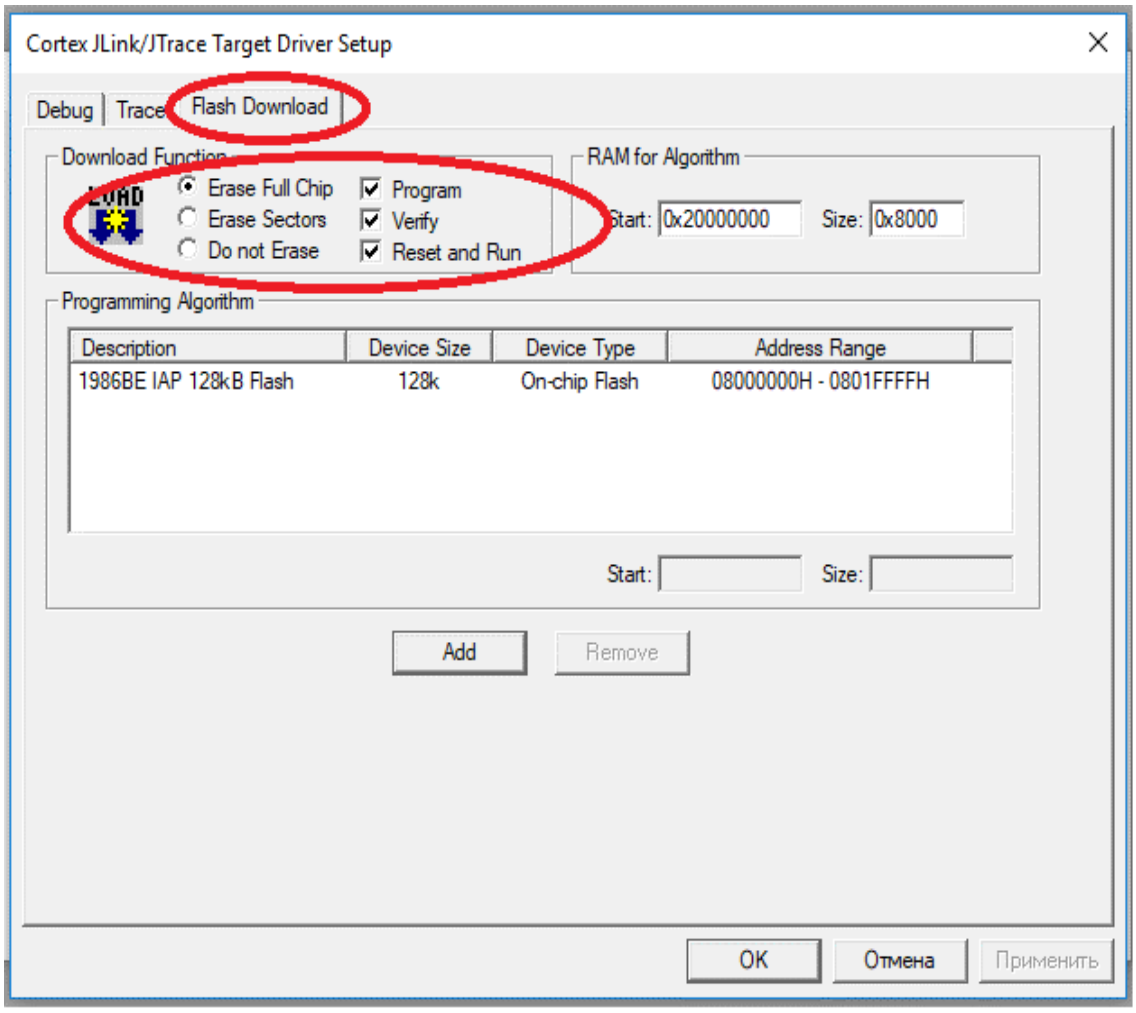

Рисунок 2.12 – Создание нового проекта. Установки программатора при записи программы во флэш-память программ

Для написания программы на языке Си создадим файл *main.c*, для чего на вкладке *Project* в списке *Source Group 1* с помощью правой кнопки мыши выберем меню *Add New Item to Group 'Source Group1'...* (рисунок 2.13). В появившемся окне (рисунок 2.14) выберем *C File (.c)*, запишем имя *main* и нажмем кнопку *Add* (добавить).

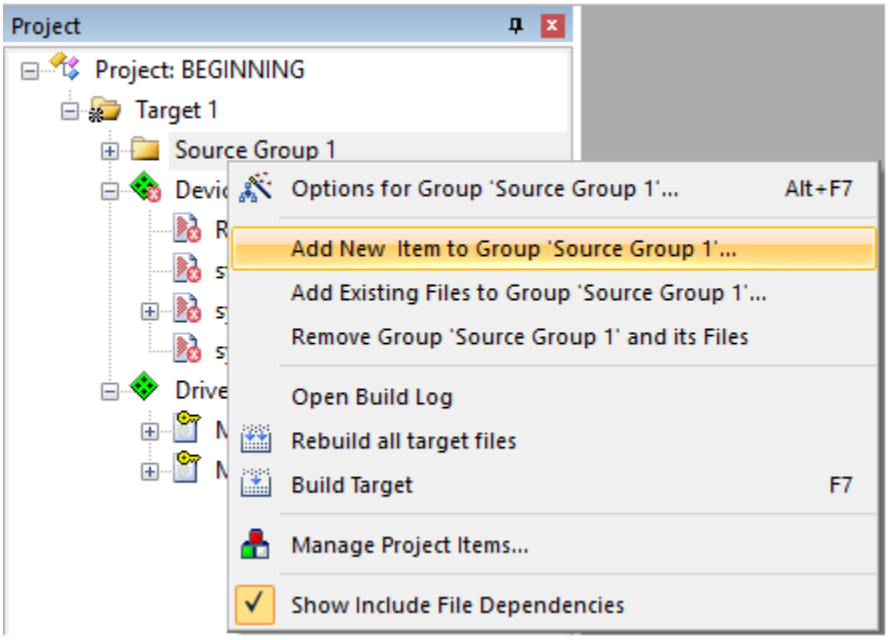

Рисунок 2.13 – Создание файла для написания программы на языке Си

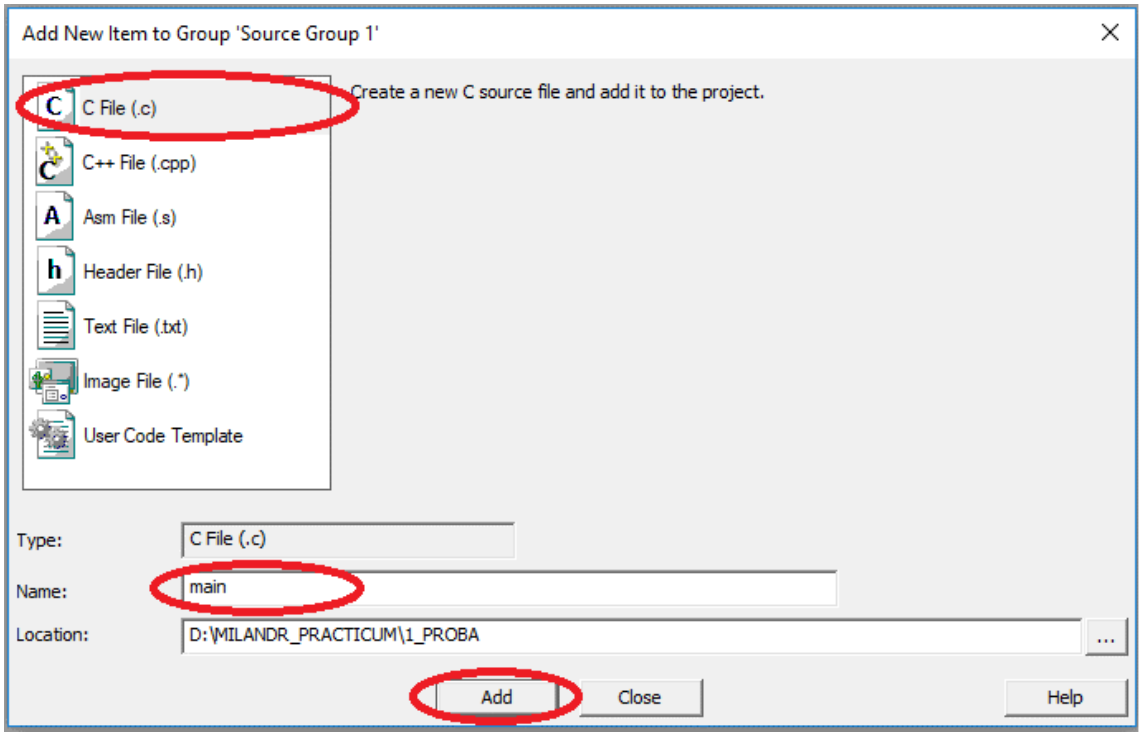

Рисунок 2.14 – Присвоение имени файлу

В созданном файле и будем писать программу. Но для начала скопируем ее текст из настоящего лабораторного практикума и вставим в окно редактирования программ.

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#define DELAY(T) for (i = T; i > 0; i--)
int i;
int main()
{
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);
  PORT_InitTypeDef Nastroyka;
  Nastroyka.PORT_Pin   = PORT_Pin_0;
  Nastroyka.PORT_OE    = PORT_OE_OUT;
  Nastroyka.PORT_FUNC  = PORT_FUNC_PORT;
  Nastroyka.PORT_MODE  = PORT_MODE_DIGITAL;
  Nastroyka.PORT_SPEED = PORT_SPEED_SLOW;
  PORT_Init (MDR_PORTC, &Nastroyka);
  while (1)
       {
    PORT_SetBits(MDR_PORTC, PORT_Pin_0);
    DELAY(100000);
         PORT_ResetBits(MDR_PORTC, PORT_Pin_0);
    DELAY(100000);
  }
}
```

Обратите внимание, что после последней фигурной скобки следует поставить одну пустую строку.

Теперь нажмем кнопку *Rebuild* (перестроить) на панели инструментов (рисунок 2.15).

Рисунок 2.15 – Присвоение имени файлу

При этом в окне *Build Output*, расположенном в нижней части экрана, появится текст:

```
*** Using Compiler 'V5.06 update 5 (build 528)',
folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Rebuild target 'Target 1'
assembling startup_MDR32F9Qx.s
compiling system_MDR32F9Qx.c
compiling main.c
compiling MDR32F9Qx_port.c
compiling MDR32F9Qx_rst_clk.c
```

```
linking...
Program Size: Code=1252 RO-data=224 RW-data=8 ZI-data=1632
FromELF: creating hex file...
".\Objects\BEGINNING.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:02
```

Главное, что мы должны увидеть в этом тексте: 0 Error(s), 0 Warning(s), то есть: ошибок нет, предостережений нет.

Далее нажимаем кнопку *LOAD* (загрузить во флэш-память программ), – см. рисунок 2.15.

Здесь отчет в окне *Build Output* будет подлиннее. Пока для нас главное:

```
Full Chip Erase Done.
Programming Done.
Verify OK.
Application running
```

То есть всё, что необходимо, исполнено; приложение работает.

И в самом деле, светодиод VD3, расположенный на плате, стал мигать.

Мы уже говорили о том, что микроконтроллер может работать и автономно, без участия компьютера. Чтобы убедиться в этом, необходимо сначала отключить питание от отладочной платы, затем отсоединить разъем шлейфа JTAG-B от отладочной платы, а затем вновь подключить к плате источник питания. Но пока делать этого не будем.

Сделаем первый шаг в программировании: заменим строки в программе `DELAY(100000)` на `DELAY(10000)`, после чего вновь последовательно нажмем кнопки *Rebuild* и *LOAD*. Частота миганий светодиода изменилась. Вот так! Не сложно?

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Каково назначение интегрированной среды разработки Keil µVision?

2. Как создать проект в среде разработки Keil µVision?

3. Как изменить частоту миганий светодиодов в приведенном примере программы?

ЛАБОРАТОРНАЯ РАБОТА 3.

УПРАВЛЕНИЕ ПОРТАМИ МИКРОКОНТРОЛЛЕРА

ЦЕЛЬ РАБОТЫ

Уметь настраивать порты микроконтроллера. Овладеть навыками записи программ в микроконтроллер.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Порт микроконтроллера представляет собой параллельный регистр, через который микроконтроллер обменивается информацией с внешними устройствами. У микроконтроллера К1986ВЕ92QI имеется 6 портов ввода/вывода: A, B, C, D, E, F. Порты имеют соответствующие имена: MDR_PORTA, MDR_PORTB, MDR_PORTC, MDR_PORTD, MDR_PORTE, MDR_PORTF.

Каждый порт является 16-разрядным, а его выводы (линии) могут использоваться различными функциональными блоками. Для того чтобы выводы порта перешли под управление того или иного периферийного блока, необходимо задать для нужных выводов выполняемую функцию и настройки. Поскольку микроконтроллер К1986ВЕ92QI размещен в корпусе, имеющем 64 вывода, то не все разряды портов связаны с этими выводами (таблица 3.1). Всего задействовано 43 линии для подключения внешних устройств: порт A имеет 8 линий, порт B – 11, порт C – 3, порт D – 8, порт E – 6, порт F – 7.

Для настройки портов микроконтроллеров используют регистры специального назначения, и в зависимости от информации, помещаемой в них, порт будет работать в том или ином режиме. Такой подход предполагает скрупулезное изучение устройства микроконтроллера, что по мере роста сложности последних становится проблематичным. Разработчик предлагает более простой путь: работать не с регистрами микроконтроллера, а с заранее подготовленными функциями, которые им сведены в библиотеку.

Для настройки порта воспользуемся библиотекой *MDR32F9Qx.h* и пошагово настроим порт.

Порт не может работать без подключения к тактовому генератору, и это надо сделать в самом начале настройки порта. Подключение осуществляется посредством специальной функции из другого библиотечного модуля – *MDR32F9Qx_rst_clk.c*:

```
RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);
```

Отнеситесь к этой записи как к шаблону, в котором вы можете менять только две вещи: выбрать порт (например, `PORTC` или `PORTA`) и разрешить тактирование порта (`ENABLE`). Если потребуется запретить тактирование, вместо слова `ENABLE` в функции следует записать `DISABLE`.

Таблица 3.1 – Линии порта, связанные с выводами микросхемы

Порт	Линия порта	Номер вывода микросхемы	Порт	Линия порта	Номер вывода микросхемы
MDR_PORTA	PA0	63	MDR_PORTD	PD0	31
	PA1	62		PD1	32
	PA2	61		PD2	33
	PA3	60		PD3	34
	PA4	59		PD4	30
	PA5	58		PD5	35
	PA6	57		PD6	36
	PA7	56		PD7	29
MDR_PORTB	PB0	43	MDR_PORTE	PE0	26
	PB1	44		PE1	25
	PB2	45		PE2	22
	PB3	46		PE3	21
	PB4	47		–	–
	PB5	50		–	–
	PB6	51		PE6	16
	PB7	52		PE7	15
	PB8	53	MDR_PORTF	PF0	2
	PB9	52		PF1	3
	PB10	54		PF2	4
MDR_PORTC	PC0	42		PF3	5
	PC1	41		PF4	6
	PC2	40		PF5	7
				PF6	8

Если требуется подключить несколько портов, то используется логическая функция ИЛИ:

```
RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTA | RST_CLK_PCLK_PORTC |
                RST_CLK_PCLK_PORTD | RST_CLK_PCLK_PORTF,
                ENABLE);
```

Знак « | » обозначает логическую функцию ИЛИ, он находится на верхнем регистре той же клавиши, что и слеш « / », но при английской раскладке клавиатуры.

Для инициализации порта потребуется *структура* следующего типа: `PORT_InitTypeDef`. После указания типа структуры необходимо придумать и указать имя, например, `Nastroyka` или, скажем, `PortInit`. Тогда описание структуры с именем будет выглядеть так:

```
PORT_InitTypeDef Nastroyka;
```

Теперь необходимо описать *поля*, входящие в структуру. Это и будет настройкой порта.

Сначала выберем линию порта, которую нужно настроить. Допустим, это линия 0. Тогда нужно сделать следующую запись:

```
Nastroyka.PORT_Pin = PORT_Pin_0;
```

То есть сначала имя структуры, затем – точка, следом – имя настраиваемого *поля*, знак равенства, значение *поля*.

Если требуется настроить сразу несколько линий порта, скажем, первую, третью и пятую, то запись будет такой:

```
Nastroyka.PORT_Pin = PORT_Pin_1 | PORT_Pin_3 | PORT_Pin_5;
```

Если требуется настройка *всех* (по-английски – *all*) линий порта, то запись потребуется такая:

```
Nastroyka.PORT_Pin = PORT_Pin_All;
```

Мы уже знаем, что линиями порта могут воспользоваться различные периферийные блоки, и об этом тоже нужно сообщить микроконтроллеру. Пока мы хотим использовать порт как порт, т.е. как параллельный регистр для хранения и выдачи информации. Описывается это так:

```
Nastroyka.PORT_FUNC = PORT_FUNC_PORT;
```

Информация, с которой имеет дело порт, может быть аналоговой (`ANALOG`) или цифровой (`DIGITAL`). Пусть – цифровой, тогда:

```
Nastroyka.PORT_MODE = PORT_MODE_DIGITAL;
```

У порта может быть два режима: передача информации (`OUT`) и ее прием (`IN`):

```
Nastroyka.PORT_OE = PORT_OE_OUT;
```

Значит, если к порту микроконтроллера нужно подключить индикаторный светодиод, то – `OUT`, а если кнопку для подачи логических нулей и единиц в микроконтроллер, то – `IN`.

Переход из логического нуля в логическую единицу и обратно в линии порта происходит не мгновенно. Длительность фронта и среза установим самой большой, т.е. скорость переключения будет самой низкой, медленной (`SLOW`):

```
Nastroyka.PORT_SPEED = PORT_SPEED_SLOW;
```

И, наконец, определимся с портом, к которому имеют отношение все указанные настройки, созданные нами с помощью структуры `Nastroyka`. Пусть это будет порт C, который называется `MDR_PORTC`:

```
PORT_Init(MDR_PORTC, &Nastroyka);
```

Всё, настройка завершена.

Теперь научимся работать с разрядами (битами) порта. Чтобы записать логическую единицу в определенном разряде порта (установить бит) необходимо указать порт и линию порта (вывод, «пин»). Пусть это будет порт C, линия 0:

```
PORT_SetBits(MDR_PORTC, PORT_Pin_0);
```

Если требуется записать там же логический ноль (сбросить бит):

```
PORT_ResetBits(MDR_PORTC, PORT_Pin_0);
```

Чтобы прочитать информацию, по линии 0 порта C, используется следующая функция:

```
PORT_ReadInputDataBit(MDR_PORTC, PORT_Pin_0);
```

Мы можем и сами создавать подобные программные элементы, используя *директивы*.

Например, `#define` – это директива, которая определяет идентификатор и последовательность символов, которой будет замещаться данный идентификатор при его обнаружении в тексте программы. То есть идентификатор – это *макрос*, который заменяет последовательность, часто очень большую, символов.

Структура данной директивы имеет следующий вид:

```
#define ИДЕНТИФИКАТОР последовательность_символов
```

Например,

```
#define NOMER 25
```

Таким образом, если компилятор обнаружит в тексте программы слово NOMER, он заменит его на 25. Количество символов для замены не ограничено.

Подобным образом можно определять функции, значения аргументов которых будут определяться в ходе выполнения программы. Например,

```
#define DELAY(T) for (i = T; i > 0; i--)
```

Теперь, если в программе встретится выражение DELAY(55), будут выполняться следующие действия. Переменной i будет присвоено значение 55, затем оно будет уменьшено на единицу, потом – вновь на единицу, и так до тех пор, пока i будет оставаться положительным. Как только i станет равным нулю, осуществится переход к следующей строке программы. На выполнение описанных действий будет затрачено время, поэтому описанный прием иногда применяют, чтобы создать временную задержку. К сожалению, при этом тратится не только время: мы занимаем процессор рутинной процедурой, и он уже не может заниматься ничем другим, что, конечно же, неэффективно. В дальнейшем мы найдем иные способы задавать интервалы времени, а время – важнейший фактор в управлении технологическими процессами.

Итак, вернемся к программе, которую в лабораторной работе №2 мы бездумно залили в микроконтроллер и наблюдали мигание светодиода. Ниже еще раз приведен текст программы, но теперь уже с краткими комментариями, смысл которых мы только что прояснили.

```
// Мигание светодиода
// Подключение заголовочных файлов необходимых библиотек
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
// Определение функции задержки
#define DELAY(T) for (i = T; i > 0; i--)
// Глобальная переменная i счетчика для макроса DELAY()
int i;

// Объявление главной функции,
// с которой начинается работа программы
int main()
{
```

```
  // Включение тактирования порта C
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);
  // Указание типа структуры и имени структуры
  PORT_InitTypeDef Nastroyka;
  // Объявление номера линии порта, которая
  // настраивается данной структурой
  Nastroyka.PORT_Pin = PORT_Pin_0;
  // Конфигурация линии порта как выхода
  Nastroyka.PORT_OE = PORT_OE_OUT;
  // Работа в режиме порта ввода-вывода
  Nastroyka.PORT_FUNC = PORT_FUNC_PORT;
  // Цифровой режим
  Nastroyka.PORT_MODE = PORT_MODE_DIGITAL;
  // Низкая скорость переключения (пологий фронт)
  Nastroyka.PORT_SPEED = PORT_SPEED_SLOW;
  // Инициализация порта C объявленной структурой
  PORT_Init(MDR_PORTC, & Nastroyka);

  // Главный цикл
  while(1)
  {
    // Установка единицы на линии PC0
    PORT_SetBits (MDR_PORTC, PORT_Pin_0);
    DELAY(100000);  // Задержка
    // Установка нуля на PC0
    PORT_ResetBits (MDR_PORTC, PORT_Pin_0);
    DELAY(100000);  // Задержка
  }
}
```

ПРАКТИЧЕСКАЯ ЧАСТЬ

1. Подключите отладочную плату к компьютеру.

2. Подайте питание на плату.

3. Запустите среду программирования Keil µVision.

4. Занесите программу в микроконтроллер с помощью программатора. Удостоверьтесь в правильности работы устройства.

5. По заданию преподавателя измените длительность задержки и вновь проделайте пункт 3.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что представляет собой порт микроконтроллера и каково его назначение?

2. Сколько портов в микроконтроллере К1986ВЕ92QI и чем они отличаются друг от друга?

3. Как настраиваются режимы работы портов?

4. Поясните логику работы программы, приведенной в данной лабораторной работе.

ЛАБОРАТОРНАЯ РАБОТА 4.

ПРЕРЫВАНИЕ ОТ КНОПКИ

ЦЕЛЬ РАБОТЫ

Уметь организовать прерывание при возникновении внешнего события.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Прерывание – это изменение естественного хода выполнения программы под воздействием внешней причины (*события прерывания*). Таким событием в микроконтроллере может быть, в частности, нажатие кнопки. При возникновении события прерывания в определенном регистре специального назначения один из битов устанавливается в логическую единицу (появляется так называемый *флаг*), а микроконтроллер переходит к исполнению команды, записанной по фиксированному адресу, который называется *вектором прерывания*. При этом начинается выполнение *отдельной* программы, по завершении которой осуществляется очистка флага, а затем – возврат к основной программе. Под отдельной программой понимается совокупность команд, которую должен выполнить микроконтроллер в результате внешнего события (в данном случае – нажатия кнопки).

Опишем процедуру прерывания, назвав ее, например, `INTERRUPT`.

Сначала сбросим флаг внешнего прерывания, чтобы прерывание не было заблокировано:

```
NVIC_ClearPendingIRQ(EXT_INT2_IRQn);
```

Теперь разрешим прерывания «в принципе» (глобальное разрешение):

```
__enable_irq();
```

а затем уже разрешим конкретный вид прерываний, в нашем случае – внешнее прерывание (`EXT_INT2`):

```
NVIC_EnableIRQ(EXT_INT2_IRQn);
```

Главное, ради чего осуществляется прерывание – это обработка прерываний, т.е. выполнение определенной программы, ради которой и было организовано прерывание. Обработка прерывания осуществля-

ется процедурой `void EXT_INT2_IRQHandler(void)`. В рамках процедуры сначала выполняется набор необходимых команд, а завершается процесс сбросом флага внешнего прерывания:

```
NVIC_ClearPendingIRQ(EXT_INT2_IRQn);
```

Рассмотрим программу, суть которой сводится к следующему. При подаче питания на отладочную плату светодиод VD3 будет мигать с низкой частотой, при нажатии на кнопку, предварительно подключенную к плате, частота мигания возрастет, при следующем нажатии – еще увеличится, затем – опять, а вот при четвертом нажатии на кнопку всё вернется в исходное состояние, и светодиод будет мигать медленно.

```
// Изменение частоты миганий светодиода VD3
// на отладочной плате по прерыванию от внешней кнопки
// Подключение заголовочных файлов необходимых библиотек
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
// Объявление переменных
int j;
int i;
int T;
// Определение первой функции задержки
#define DELAY1(D) for (j = D; j > 0; j--)
// Определение второй функции задержки
#define DELAY2(T) for (i = T; i > 0; i--)
// Процедура инициализации портов
void PortsInit()
{
  // Включение тактирования портов B и C
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTB |
                  RST_CLK_PCLK_PORTC,
                  ENABLE);

  // Указание типа структуры и имени структуры
  PORT_InitTypeDef Nastroyka_C;
  // Работа в режиме порта ввода-вывода
  Nastroyka_C.PORT_FUNC = PORT_FUNC_PORT
  // Цифровой режим
  Nastroyka_C.PORT_MODE = PORT_MODE_DIGITAL;
  // Низкая скорость переключения (пологий фронт)
  Nastroyka_C.PORT_SPEED = PORT_SPEED_SLOW;
  // Конфигурация линии порта как выхода
  Nastroyka_C.PORT_OE = PORT_OE_OUT;
```

```
  // Объявление номера линии порта, которая
  // настраивается данной структурой
  Nastroyka_C.PORT_Pin = PORT_Pin_0;
  // Инициализация порта C объявленной структурой
  PORT_Init(MDR_PORTC, &Nastroyka_C);

  // Указание типа структуры и имени структуры
  PORT_InitTypeDef Nastroyka_B;
  // Работа в альтернативном режиме порта EXT_INT2
  Nastroyka_B.PORT_FUNC = PORT_FUNC_ALTER;
  // Цифровой режим
  Nastroyka_B.PORT_MODE = PORT_MODE_DIGITAL;
  // Низкая скорость переключения (пологий фронт)
  Nastroyka_B.PORT_SPEED = PORT_SPEED_SLOW;
  // Конфигурация линии порта как входа
  Nastroyka_B.PORT_OE = PORT_OE_IN;
  // Объявление номера линии порта, которая
  // настраивается данной структурой
  Nastroyka_B.PORT_Pin = PORT_Pin_10;
  //Инициализация порта C объявленной структурой
  PORT_Init(MDR_PORTB, &Nastroyka_B);
}

// Процедура инициализации внешнего прерывания
void INTERRUPT()
{
  // Очистка флага внешнего прерывания EXT_INT2
  NVIC_ClearPendingIRQ(EXT_INT2_IRQn);
  // Глобальное разрешение прерываний
  __enable_irq();
  // Разрешения внешнего прерывания EXT_INT2
  NVIC_EnableIRQ(EXT_INT2_IRQn);
}

// Процедура обработки внешнего прерывания EXT_INT2
void EXT_INT2_IRQHandler(void)
{
  // Уменьшение времени задержки,
  // т.е. увеличение частоты мигания светодиода
  T = T - 50000;
  // Если задержка равна нулю,
  // то возвращение в исходное состояние
  if (T == 0) T = 200000;
  // Задержка, в течение которой прекращается дребезг контактов
  DELAY1(100000);

  // Если сигнал внешнего прерывания завершен,
```

```
  // очистить флаг внешнего прерывания EXT_INT2
  if (PORT_ReadInputDataBit(MDR_PORTB, PORT_Pin_10) == 0)
  {
          NVIC_ClearPendingIRQ(EXT_INT2_IRQn);
        }
}

// Объявление главной функции
int main()
{
  // Вызов процедуры инициализации портов
  PortsInit();
  // Вызов процедуры инициализации внешнего прерывания
  INTERRUPT();
  // Задание начального значения второй функции задержки,
  // определяющей время свечения светодиода и паузы
  T = 200000;
  // Главный цикл
  while (1)
  {
    // Установка единицы на линии PC0 порта C
    PORT_SetBits(MDR_PORTC, PORT_Pin_0);
         DELAY2(T);  // Задержка
    // Сброс единицы на линии PC0 порта C
    PORT_ResetBits(MDR_PORTC, PORT_Pin_0);
    // Сброс единицы на 0 линии порта C
    DELAY2(T);  // Задержка
  }
}
```

Итак, главная программа начинается с вызова процедур настройки портов и инициализации внешнего прерывания. Затем переменной `T` задается значение равное 200000, и начинается главный цикл. Устанавливается логическая единица на линии 0 порта C; загорается светодиод VD3; затем задержка, определяемая переменной `T`; сброс линии 0 порта C (логический ноль на линии); вновь задержка, и – бесконечный цикл.

При внешнем прерывании (нажатие кнопки) значение переменной `T` уменьшается на 50000. Если при этом переменная `T` станет равной нулю, то она вновь принимает исходное значение, если нет – то произойдет небольшая задержка, за время которой успеют прекратиться импульсы, которые всегда возникают при замыкании/размыкании механических контактов, и осуществится опрос линии 10 порта B с целью определить, отпущена ли кнопка:

```
if (PORT_ReadInputDataBit(MDR_PORTB, PORT_Pin_10) == 0)
```

Если кнопка отпущена (на линии 10 порта B логический ноль), то осуществится выход из прерывания со сбросом флага, если нет – то продолжается опрос линии 10 порта B. При этом в главном цикле значение `T` будет другим, изменится значение функции `DELAY2(T)`, а значит, и время свечения и паузы светодиода.

ПРАКТИЧЕСКАЯ ЧАСТЬ

1. Подключите дополнительную кнопку SB к разъему XP13 отладочной платы в соответствии с рисунком 4.1

2. Подключите отладочную плату к компьютеру.

3. Подайте питание на плату.

4. Запустите среду программирования Keil µVision.

5. Занесите программу в микроконтроллер с помощью программатора. Нажимая на кнопку SB, наблюдайте за изменением частоты мигания светодиода VD3. Удостоверьтесь в правильности работы устройства.

6. По заданию преподавателя измените начальную частоту мигания светодиода, количество ступеней при переключении частоты.

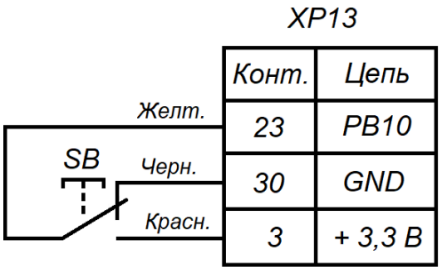

Рисунок 4.1 – Подключение кнопки к отладочному стенду

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что называют прерыванием в микропроцессорной технике? Под воздействием каких причин оно происходит?

2. Что такое «вектор прерывания»?

3. Как осуществляется обработка прерывания?

4. Поясните логику работы программы, приведенной в данной лабораторной работе.

ЛАБОРАТОРНАЯ РАБОТА 5.

ПРЕРЫВАНИЕ ОТ ТАЙМЕРА

ЦЕЛЬ РАБОТЫ

Изучить характеристики таймера микроконтроллера. Уметь использовать таймер для задания интервалов времени и организации прерываний.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

В состав микроконтроллера К1986ВЕ92QI входят 3 таймера общего назначения (на рисунке 1.1 обозначены как «3 x 16 Timer» – найдите их). Таймер представляет собой 16-разрядный счетчик, у которого есть три режима счета: прямой, обратный и двунаправленный (сначала прямой счет до определенного значения, а затем обратный). Для изменения частоты счета у каждого таймера имеется 16-разрядный предделитель с переменным коэффициентом деления, который также представляет собой счетчик. На основе таймеров построены четырехканальные блоки, обеспечивающие режимы «Захват», «Сравнение» и «ШИМ».

Режим «Захват» позволяет по внешнему сигналу останавливать счет, и может быть использован, например, когда необходимо измерить интервал времени между двумя событиями.

В режиме «Сравнение» микроконтроллер выдает сигнал, когда состояние таймера совпадает с заранее записанным числом, т.е. работает как таймер в обыденном понимании, например, как таймер в мобильном телефоне.

Режим «ШИМ» – это режим широтно-импульсной модуляции: микроконтроллер выдает прямоугольные импульсы неизменной частоты, неизменной амплитуды, но переменной длительности. При этом, понятно, средний за период уровень сигнала будет также переменным. Данный режим широко используется, когда необходимо регулировать яркость свечения лампы накаливания, частоту вращения двигателя постоянного тока и т.п.

О режиме «ШИМ» поговорим в дальнейшем, а пока займемся настройкой таймера в качестве счетчика.

Прежде всего, таймер должен быть подключен к тактовому генератору, подобно тому, как мы это делали при настройке порта:

```
RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER1, ENABLE);
```

Для дальнейшей настройки таймера потребуется несколько структур. Чтобы оценить возможности настройки таймера, приведем их в полном объеме.

Начнем со структуры **TIMER_CntInitTypeDef**. Структура имеет следующие поля:

– 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 – фильтр событий;

– TIMER_ARR_UpdateMode – режим сброса счетчика;

– TIMER_ETR_FilterConf – параметры выхода ETR;

– TIMER_ETR_Prescaler – параметры предделителя фильтра выхода;

– ETR TIMER_ETR_Polarity – полярность выхода;

– ETR TIMER_BRK_Polarity – полярность выхода BRK.

Структура **TIMER_ChnInitTypeDef** имеет следующие поля:

– TIMER_CH_Mode – задает режим работы таймера;

– TIMER_CH_REF_Format – формат выработки сигнала REF в режиме ШИМ;

– TIMER_CH_Number – номер канала таймера.

И, наконец, поля структуры **TIMER_ChnOutInitTypeDef**:

– TIMER_CH_DirOut_Polarity – полярность выхода CHx;

– TIMER_CH_DirOut_Source – сигнал на выходе CHx;

– TIMER_CH_DirOut_Mode – сигнал на выходе CHx;

– TIMER_CH_NegOut_Polarity – полярность инверсного выхода CHx;

– TIMER_CH_NegOut_Source – сигнал на инверсном выходе CHx;

– TIMER_CH_NegOut_Mode – сигнал на инверсном выходе CHx;

– TIMER_CH_Number – номер канала.

Строго говоря, все эти поля надо прописать, однако некоторые из них являются типичными и встречаются очень часто, поэтому могут быть установлены «по умолчанию». Для этого в программе необходимо записать следующую строку:

```
TIMER_CntStructInit(&TIM1Init);
```

При этом произойдет заполнение полей с помощью структуры, описанной в библиотеке в файле *MDR32F9Qx_timer.c*:

```
void TIMER_CntStructInit(TIMER_CntInitTypeDef
*TIMER_CntInitStruct)
{
  TIMER_CntInitStruct->TIMER_IniCounter       = 0;
  TIMER_CntInitStruct->TIMER_Prescaler        = 0;
  TIMER_CntInitStruct->TIMER_Period           = 0;
  TIMER_CntInitStruct->TIMER_CounterMode      =
TIMER_CntMode_ClkFixedDir;
  TIMER_CntInitStruct->TIMER_CounterDirection = TIMER_CntDir_Up;
  TIMER_CntInitStruct->TIMER_EventSource      =
TIMER_EvSrc_None;
  TIMER_CntInitStruct->TIMER_FilterSampling   =
TIMER_FDTS_TIMER_CLK_div_1;
  TIMER_CntInitStruct->TIMER_ARR_UpdateMode   =
TIMER_ARR_Update_Immediately;
  TIMER_CntInitStruct->TIMER_ETR_FilterConf   =
TIMER_Filter_1FF_at_TIMER_CLK;
  TIMER_CntInitStruct->TIMER_ETR_Prescaler    =
TIMER_ETR_Prescaler_None;
  TIMER_CntInitStruct->TIMER_ETR_Polarity     =
TIMER_ETRPolarity_NonInverted;
  TIMER_CntInitStruct->TIMER_BRK_Polarity     =
TIMER_BRKPolarity_NonInverted;
}
```

После настройки «по умолчанию» отдельные позиции можно скорректировать. Что же остается настроить? Совсем немного.

Настройка *делителя* тактовой частоты осуществляется с помощью параметра `TIMER_HCLK`, который участвует в выражении:

```
TIMER_BRGInit(MDR_TIMER1, TIMER_HCLKdiv1);
```

Данный параметр может принимать следующие значения:

– `TIMER_HCLKdiv1`: нет деления входной частоты;

– `TIMER_HCLKdiv2`: делитель входной частоты на 2 (т.е. замедлит переключение таймера в 2 раза);

– `TIMER_HCLKdiv4`: делитель входной частоты на 4;

– `TIMER_HCLKdiv8`: делитель входной частоты на 8;

– `TIMER_HCLKdiv16`: делитель входной частоты на 16;

– `TIMER_HCLKdiv32`: делитель входной частоты на 32;

– `TIMER_HCLKdiv64`: делитель входной частоты на 64;

– TIMER_HCLKdiv128: делитель входной частоты на 128.

Зададим коэффициент деления *предделителя* тактовой частоты:

```
TIM1Init.TIMER_Prescaler = 8000;
```

Это еще один способ изменить частоту переключения таймера: чем больше число – тем медленнее работает таймер. Значение 16-разрядного *предделителя* могут быть от 1 до 65535 (т.е. $2^{16} – 1$). Например, тактовая частота составляет 8 МГц, делитель тактовой частоты не применялся, коэффициент деления предделителя 8000. Тогда таймер будет считать с частотой $8 \cdot 10^6$ Гц / $8 \cdot 10^3 = 10^3$ Гц, или периодом 1 мс.

Зададим период срабатывания таймера:

```
TIM1Init.TIMER_Period = 500;
```

В этом случае таймер досчитает до 500, выдаст сигнал и обнулится, а затем снова будет считать до 500. Значение периода можно задавать числами от 1 до 65535. Таким образом, в рассматриваемом примере период срабатывания таймера составляет 1 мс · 500 = 500 мс, то есть таймер будет обнуляться 2 раза в секунду (частота – 2 Гц).

Завершается настройка указанием названия таймера, к которому применена настройка, аналогично тому, как это делалось для порта:

```
TIMER_CntInit(MDR_TIMER1, &TIM1Init);
```

Теперь поговорим о сигнале, который выдает таймер. Его можно использовать для *прерывания* основной программы и выполнения каких-либо действий всякий раз, когда таймер завершает свой счет – например, включение и выключение светодиода. Работа по отдельной программе после возникновения события прерывания, как вы помните, называется обработкой прерывания. Преимущество такого подхода в сравнении с формированием временных интервалов с помощью циклического программного счета заключается в том, что формирование интервалов времени происходит аппаратно, а микроконтроллер при этом может выполнять основную программу.

Для разрешения прерываний от таймера 1 необходимо записать:

```
NVIC_EnableIRQ(TIMER1_IRQn);
```

Здесь дается лишь общее разрешение на прерывание от таймера TIMER1, но пока не ясно, при каком состоянии таймера оно произойдет. Установим возникновение прерывания при равенстве нулю значения TIMER1:

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

Прерывание в микроконтроллере может быть вызвано различными источниками, поэтому необходимо задать их приоритет, т.е. указать микроконтроллеру, какое прерывание необходимо обработать в первую очередь, во вторую и т.д. Чем меньше число, обозначающее приоритет, тем выше уровень приоритета, тем раньше будет обработано прерывание. Вот так устанавливается приоритет, равный нулю:

```
NVIC_SetPriority(TIMER1_IRQn, 0);
```

Теперь можно запустить таймер:

```
TIMER_Cmd(MDR_TIMER1, ENABLE);
```

Итак, таймер начал считать с нужной нам частотой до заданного нами значения, и вот он обнулился и должен приступить к обработке прерывания. Делается это с помощью процедуры:

```
void Timer1_IRQHandler()
```

Допустим, если состояние таймера стало равным нулю, то следует выполнить процедуру `LED()`. Опишем данную ситуацию так:

```
if (TIMER_GetITStatus(MDR_TIMER1, TIMER_STATUS_CNT_ZERO))
  LED();
```

Прерывание всегда сопровождается появлением флага, или логической единицы, в определенном регистре. Флаг не только сигнализирует о том, что прерывание возникло, но и не позволяет до завершения обработки прерывания приступать к обработке следующего прерывания. Флаг должен быть сброшен программно:

```
TIMER_ClearITPendingBit(MDR_TIMER1, TIMER_STATUS_CNT_ZERO);
```

Рассмотрим программу, которая заставляет поочередно зажигать светодиоды VD3 и VD4, расположенные на отладочной плате:

```
// Подключение заголовочных файлов тех библиотек,
// которые непосредственно используются в данной программе
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_timer.h>

// Процедура инициализация порта
void PortsInit()
{
  // Указание типа структуры и имени структуры
```

```
  PORT_InitTypeDef Nastroyka;
  // Включение тактирования порта C
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);
  // Заполнение структуры значениями по умолчанию
  PORT_StructInit(&Nastroyka);

  // Объявление номера линии порта,
  // которая настраивается данной структурой
  Nastroyka.PORT_Pin = PORT_Pin_0;
  // Конфигурация группы линий как выход
  Nastroyka.PORT_OE = PORT_OE_OUT;
  // Работа в режиме порта ввода-вывода
  Nastroyka.PORT_FUNC = PORT_FUNC_PORT;
  // Цифровой режим
  Nastroyka.PORT_MODE = PORT_MODE_DIGITAL;
  // Низкая скорость переключения (пологий фронт)
  Nastroyka.PORT_SPEED = PORT_SPEED_SLOW;
  // Инициализация порта C объявленной структурой
  PORT_Init(MDR_PORTC, &Nastroyka);
}

// Процедура включения/выключения светодиода
void LED()
{
  // Объявление переменной i
  static uint8_t i = 0;
  // Инкрементируя i, находить остаток от деления i на 2
  switch (i++ % 2)
  {
  // В случае если остаток равен нулю, сбросить бит по линии 0
  case 0:
    PORT_ResetBits(MDR_PORTC, PORT_Pin_0);
    // В противном случае перейти к следующей команде
    break;
  // В случае если остаток равен единице, установить бит по ли-
нии 0
  case 1:
    PORT_SetBits(MDR_PORTC, PORT_Pin_0);
    // В противном случае перейти к следующей команде
    break;
  }
}

// Процедура инициализации таймера
void TimerInit()
{
  // Указание типа структуры и имени структуры
  TIMER_CntInitTypeDef TIM1Init;
```

```
  // Включение тактирования
  RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER1, ENABLE);
  // Заполнение структуры значениями по умолчанию
  TIMER_CntStructInit(&TIM1Init);
  // Настройка делителя тактовой частоты
  TIMER_BRGInit (MDR_TIMER1, TIMER_HCLKdiv1);
  // Задание предделителя тактовой частоты
  TIM1Init.TIMER_Prescaler = 8000;
  // Задание периода срабатывания таймера
  TIM1Init.TIMER_Period = 500;
  // Инициализация порта таймера объявленной структурой
  TIMER_CntInit (MDR_TIMER1, &TIM1Init);
  // Включение прерываний
  NVIC_EnableIRQ (TIMER1_IRQn);
  // Установка приоритета прерываний
  NVIC_SetPriority (TIMER1_IRQn, 0);
  // Включение прерывания при равенстве нулю значения TIMER1
  TIMER_ITConfig(MDR_TIMER1, TIMER_STATUS_CNT_ZERO, ENABLE);
  // Запуск таймера
  TIMER_Cmd(MDR_TIMER1, ENABLE);
}

// Процедура обработки прерывания, вызванного таймером
void Timer1_IRQHandler()
{
  // Если таймер сброшен в ноль, вызвать процедуру LED()
  // и сбросить флаг прерывания
  if (TIMER_GetITStatus(MDR_TIMER1, TIMER_STATUS_CNT_ZERO))
  {
    LED();
    TIMER_ClearITPendingBit(MDR_TIMER1, TIMER_STATUS_CNT_ZERO);
  }
}

// Объявление главной функции
int main()
{
  // Вызов процедуры инициализации порта
  PortsInit();
  // Вызов процедуры инициализации таймера
  TimerInit();
  // Пустой цикл
  while (1)
  {
  }
}
```

Если с настройкой портов и таймеров всё более или менее ясно, то о программировании на языке Си разговор только начинается. В лабораторном практикуме мы не можем позволить себе систематическое изложение основ программирования, поэтому пойдем от частного к общему.

Итак, что означает фрагмент

```
while (1)
{
      }
```

в конце программы?

Полностью конструкция, содержащая оператор `while`, выглядит так:

```
while (условие)
{
  Блок операций
}
```

То есть пока выполняется *условие*, выполняется *блок операций*.

А вот если вместо условия записать константу, как в нашем случае, то блок операций будет выполнятся всегда, ведь константа не изменится. Но у нас и операций-то нет – значит, мы создали «пустой» цикл. При этом микроконтроллер работает (задающий генератор выдает импульсы, ведет свой подсчет таймер...), но блок операций не исполняется ввиду его отсутствия.

Главная функция нашей программы имеет вид:

```
int main()
```

Здесь `int` (от integer – целый) указывает тип данных – целочисленный. Иногда в скобках после главной функции пишут void (пустой), чем подчеркивают, что тип данных, с которым будет иметь дело функция, – произвольный.

Внутри главной функции – названия двух процедур:

```
PortsInit();
TimerInit();
```

Первая инициализирует порт, вторая – таймер. Описание этих процедур должно быть выполнено до обращения к главной функции.

Процедура `LED()` также должна быть описана до обращения к ней. Внутри процедуры объявляется переменная `i`:

```
static uint8_t i;
```

То есть переменная `i` – беззнаковая (u – unsigned), целая (int – integer), 8-битная. Спецификатор `static` позволяет переменной сохранять свое значение между вызовами процедуры, а не инициализировать ее повторно при каждом вызове.

Далее познакомимся с оператором `switch`.

```
switch (переменная)
{
  case константа 1:
    Блок операторов 1
    break;
  case константа 2:
    Блок операторов 2
    break;
  ...
  default:
    Блок операторов 3
    break;
}
```

Оператор `switch` сравнивает значение переменной с несколькими константами. В случае (`case`), если переменная равна константе 1, выполняется блок операторов 1, после этого выполняется оператор `break`, программа выходит из блока оператора `switch` и выполняет следующий после блока `switch` оператор. Если переменная равна константе 2, то будет выполнен блок операторов 2, затем – выход из блока `switch` по оператору `break` и так далее. Если же переменная не равна ни одной из прописанных констант (`default`), то будет выполнен блок операторов 3, а затем `break` – выход. В нашем случае значение переменной `i` увеличивается на единицу, результат делится на 2, а затем определяется остаток от деления. Ясно, что остаток будет принимать значения то 0, то 1. При этом, исходя из текста программы, в первом случае на линии порта будет выставлен логический 0, а во втором – логическая единица.

ПРАКТИЧЕСКАЯ ЧАСТЬ

1. Подключите отладочную плату к компьютеру.
2. Подайте питание на плату.
3. Запустите среду программирования Keil µVision.

4. Занесите программу в микроконтроллер с помощью программатора. Удостоверьтесь в правильности работы устройства.

5. По заданию преподавателя измените частоту мигания светодиода, соотношение длительности свечения и паузы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что представляет собой таймер общего назначения микроконтроллера К1986ВЕ92QI?

2. Для чего нужен делитель тактовой частоты? Как настроить его работу?

3. Для чего нужен предделитель? Как настроить его работу?

4. Как организовать прерывание от таймера?

5. Поясните логику работы программы, приведенной в данной лабораторной работе.

ЛАБОРАТОРНАЯ РАБОТА 6.

РЕЖИМ ШИРОТНО-ИМПУЛЬСНОЙ МОДУЛЯЦИИ

ЦЕЛЬ РАБОТЫ

Уметь настраивать таймер в режиме широтно-импульсной модуляции.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Как вы помните из предыдущей работы, в режиме ШИМ микроконтроллер вырабатывает последовательность прямоугольных импульсов, у которых амплитуда и период являются постоянными, а ширина импульса изменяется в соответствии с программой (рисунок 6.1).

При этом чем больше длительность импульса, тем выше среднее за период значение сигнала. Средний уровень сигнала может быть выделен с помощью фильтра нижних частот либо период T выбирается таким, что в силу инерционности объекта воздействия последний реагирует не на каждый импульс, а на средний уровень сигнала. Например, при частоте импульсов, подаваемых на светодиод, выше 40-50 Гц глаз человека перестает замечать мерцания и реагирует на средний уровень яркости свечения светодиода.

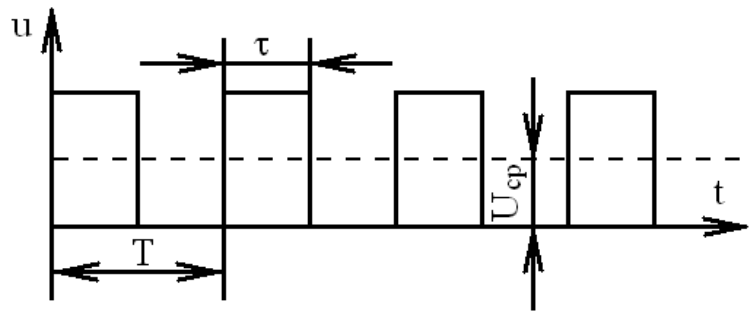

Рисунок 6.1 – Широтно-импульсная модуляция:
T = const – период сигнала; τ = var – длительность импульса;
$U_{ср}$ = var – среднее значение сигнала за период

Напомним, что принцип широтно-импульсной модуляции часто применяют в автоматизации при необходимости цифрового управления такими величинами, как скорость вращения двигателя постоянного тока, яркость свечения лампы, температура электронагревателя и т.п.

Рассмотрим программу, при выполнении которой микроконтроллер вырабатывает последовательность прямоугольных импульсов, у которых ширина сначала плавно возрастает до максимума, а затем резко спадает до минимума. Ряд блоков программы был изучен ранее и пояснений не требует, тем не менее внимательно прочитайте каждый комментарий.

```
// Подключение заголовочных файлов тех библиотек,
// которые непосредственно используются в данной программе
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_timer.h>
// Определение макроса задержки
#define DELAY(T) for (i = T; i > 0; i--)
// Глобальная переменная счетчика,
// которая используется в макросе DELAY()
int i;
// Глобальная переменная длительности импульса по каналу 1
uint16_t CCR1_Val;

// Процедура инициализация порта
void PortInit()
{
        // Создание структуры для инициализации порта
  PORT_InitTypeDef Nastroyka;
  // Включение тактирования порта A
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTA, ENABLE);
  // Сброс установок порта A
  PORT_DeInit(MDR_PORTA);
  // Настройка порта A, линия 1
  Nastroyka.PORT_Pin = PORT_Pin_1;
  // Работа линии на передачу
  Nastroyka.PORT_OE = PORT_OE_OUT;
  // Альтернативная функция: TMR1_CH1
  Nastroyka.PORT_FUNC = PORT_FUNC_ALTER;
  // Цифровой режим
  Nastroyka.PORT_MODE = PORT_MODE_DIGITAL;
  // Крутой фронт
  Nastroyka.PORT_SPEED = PORT_SPEED_FAST;
  // Инициализация порта A объявленной структурой
  PORT_Init(MDR_PORTA, &Nastroyka);
}

// Процедура инициализация таймера
void TimerInit()
{
```

```
  // Включение тактирования таймера 1
  RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER1, ENABLE);

  // Сброс установок таймера 1
  TIMER_DeInit(MDR_TIMER1);

  // Создание структуры конфигурации таймера
  TIMER_CntInitTypeDef sTIM_CntInit;
  // Заполнение структуры значениями по умолчанию
  TIMER_CntStructInit(&sTIM_CntInit);
  // Предделитель частоты таймера равен 1
  sTIM_CntInit.TIMER_Prescaler = 0x0;
  // Установка максимально возможного периода импульсов ШИМ
  sTIM_CntInit.TIMER_Period = 0xFFF;
  // Инициализация таймера 1 объявленной структурой
  TIMER_CntInit (MDR_TIMER1, &sTIM_CntInit);
  // Инициализация канала CH1 таймера 1 объявленной структурой
  TIMER_ChnStructInit (&sTIM_ChnInit);

  // Создание структуры конфигурации канала
  TIMER_ChnInitTypeDef sTIM_ChnInit;
  // Режим работы канала – генерация ШИМ
  sTIM_ChnInit.TIMER_CH_Mode = TIMER_CH_MODE_PWM;
  // Для получения сигнала ШИМ следует установить
  // формат выработки сигнала REF номер 6
  sTIM_ChnInit.TIMER_CH_REF_Format = TIMER_CH_REF_Format6;
  // Выбор канала 1
  sTIM_ChnInit.TIMER_CH_Number = TIMER_CHANNEL1;
  // Инициализация канала 1 объявленной структурой
  TIMER_ChnInit(MDR_TIMER1, &sTIM_ChnInit);
  // Установка длительности импульсов по первому каналу
  TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL1, CCR1_Val);

  // Создание структуры конфигурации выходов канала
  TIMER_ChnOutInitTypeDef sTIM_ChnOutInit;
  // Заполнение элементов структуры значениями по умолчанию
  TIMER_ChnOutStructInit(&sTIM_ChnOutInit);
  // Выбор источника сигнала для прямого выхода CHx - сигнал REF
  sTIM_ChnOutInit.TIMER_CH_DirOut_Source = TIMER_CH_OutSrc_REF;
  // Настройка прямого выхода канала таймера CHx на вывод данных
  sTIM_ChnOutInit.TIMER_CH_DirOut_Mode =
TIMER_CH_OutMode_Output;
  // Настройка выходов канала 1
  sTIM_ChnOutInit.TIMER_CH_Number = TIMER_CHANNEL1;
  TIMER_ChnOutInit (MDR_TIMER1, &sTIM_ChnOutInit);

  // Включение делителя тактовой частоты таймера 1
```

```
  TIMER_BRGInit(MDR_TIMER1, TIMER_HCLKdiv1);
  // Разрешение работы таймера 1
  TIMER_Cmd(MDR_TIMER1, ENABLE); }
// Объявление главной функции
int main()
{
  // Установка начальной длительности импульса ШИМ
  CCR1_Val = 0x001;
  // Вызов процедуры инициализации порта
  PortInit();
  // Вызов процедуры инициализации таймера
  TimerInit();
  // Главный цикл
  while (1)
  {
    // Инкремент переменной CCR1_Val (длительность импульса)
    CCR1_Val++;
    DELAY(5000);  // Задержка
    // Установка новой длительности импульса
    TIMER_SetChnCompare(MDR_TIMER1, TIMER_CHANNEL1, CCR1_Val);
    if (CCR1_Val == 0xFFF)
     {
            CCR1_Val = 0x001;
           }
       }
}
```

Итак, главная функция `main()` начинается с вызова процедуры инициализации порта. Таймер TIMER1 содержит 4 канала ШИМ. Как следует из таблицы 1.1 первый канал таймера TMR1_CH1 является цифровой альтернативной функцией и закреплен за линией PA1 (вывод микросхемы – 62). В процедуре инициализации порта это прописано так:

```
Nastroyka.PORT_FUNC = PORT_FUNC_ALTER
```

Переменная `CCR1_Val` отвечает за длительность импульса, до вызова всех процедур ей присвоено значение 0x001. Когда начинается основной цикл, оператором `CCR1_Val++` увеличивается на 1 значение переменной `CCR1_Val`, которая отвечает за длительность импульса. С помощью оператора `if` проверяем достигнуто ли предельное значение ширины импульса, равное периоду следования импульсов 0xFFF. Если достигнуто, то переменной `CCR1_Val` присваивается значение 0x001, импульс становится минимальной ширины, и процесс начинается снова.

ПРАКТИЧЕСКАЯ ЧАСТЬ

1. Переведите мультиметр в режим измерения постоянного напряжения. С помощью соединительных проводников подключите черный вывод мультиметра к выводу 30 разъема X27 отладочной платы (рисунок 1.3), а красный вывод мультиметра – к выводу 12 того же разъема платы.

2. Подключите отладочную плату к компьютеру.

3. Подайте питание на плату.

4. Запустите среду программирования Keil μVision.

5. Занесите программу в микроконтроллер с помощью программатора. Наблюдая за изменением показаний мультиметра, удостоверьтесь в правильности работы устройства.

6. Отключите блок питания от отладочной платы и USB-кабель – от компьютера.

7. Включите и настройте осциллограф. Подключите кабель осциллографа к отладочной плате: земля – к выводу 30 разъема X27 (рисунок 1.3), потенциальный вывод – к выводу 12 того же разъема платы.

8. Измерьте амплитуду, период следования, минимальную и максимальную длительность импульсов. Рассчитайте частоту как величину обратную периоду.

9. Напишите и испытайте программу, по которой средний уровень сигнала будет плавно уменьшаться, а затем резко возрастать.

10. Напишите и испытайте программу, по которой средний уровень сигнала сначала будет плавно возрастать, а затем плавно уменьшаться.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что называют широтно-импульсной модуляцией и где она применяется?

2. Какой блок микроконтроллера может работать в режиме ШИМ?

3. Какие выводы микроконтроллера могут передавать сигналы ШИМ?

4. Как установить период ШИМ?

5. Как установить длительность импульса в режиме ШИМ?

6. Поясните логику работы программы, приведенной в данной лабораторной работе.

ЛАБОРАТОРНАЯ РАБОТА 7.

ИНДИКАЦИЯ

ЦЕЛЬ РАБОТЫ

Изучить принцип действия жидкокристаллического (ЖК) модуля МТ-12864J. Научится выводить информацию на экран модуля.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Жидкокристаллический модуль МТ–12864J (поз. 11 на рисунке 1.3) состоит из двух встроенных контроллеров управления и ЖК-панели (экрана). Каждой светящейся точке на экране соответствует логическая единица в ячейке оперативного запоминающего устройства (ОЗУ) модуля.

	Контроллер 1					Контроллер 2				
	0	1	2	···	63	0	1	2	···	63
0										
1										
2										
3		Страница 0					Страница 0			
4										
5										
6										
7										
0										
1										
2										
3		Страница 1					Страница 1			
4										
5										
6										
7										
		···					···			
0										
1										
2										
3		Страница 7					Страница 7			
4										
5										
6										
7										

Рисунок 7.1 – Структура экрана ЖК-модуля МТ-12864J

Экран, как и ОЗУ, имеет структуру, показанную на рисунке 7.1. Экран делится на две половины: левую, которой управляет контроллер 1, и правую, которой управляет контроллер 2. Каждая половина экрана состоит из 8 страниц: страница 0, страница 1, ..., страница 7. Каждая страница содержит восемь строк (номера от 0 до 7) и 64 колонки (номера от 0 до 63). Таким образом на экране расположено 64 х 2 = 128 колонок и 8 х 8 = 64 строки – всего 128 х 64 = 8192 элемента (пикселя) экрана. Такое количество элементов позволяет выводить на экран не только знаковую информацию, но и примитивные рисунки.

Для работы с ЖК-модулем компанией «Миландр» разработана библиотечная программа *lcd.c*, основные функции которой мы и изучим.

Начнем с классического приветствия программистов «Привет, мир!»:

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

int main(void)
{
  LCD_Init();
  LCD_PutString("Привет, мир!", 3);
  while (1)
  {
  }
}
```

Сначала подключаются файлы, необходимые для работы портов, задающего генератора и ЖК-модуля. Затем вызывается функция `LCD_Init()`, позволяющая работать с экраном, и функция `LCD_PutString("Привет, мир!", 3)`, позволяющая вывести целую строчку символов. Последняя функция имеет следующее описание:

```
void LCD_PutString(char* string, uint8_t y);
```

из которого нам важно понять лишь то, что вместо указателя на массив со строкой `char* string` следует *в кавычках* записать желаемый текст, вместо беззнаковой переменной `y` – номер страницы экрана от 0 до 7, поскольку один символ по вертикали занимает 8 пикселей, то есть высота символа равна высоте страницы. Изменяя значение `y`, можно смещать текст на экране по вертикали. К слову сказать, по го-

ризонтали один символ занимает также 8 пикселей, то есть при 128 колонках на экране в одном ряду может разместится 128 : 8 = 16 символов, или знакомест.

Полезными могут оказаться и другие функции.

```
void LCD_ClearString(uint8_t y);
```

Функция позволяет очистить страницу экрана от текста, т.е. удалить строчку текста с экрана. Здесь y – номер страницы от 0 до 7.

```
// Пример использования
LCD_ClearString(3);

void LCD_ScrollString(char* string, uint8_t y, uint8_t direc-
tion);
```

Функция позволяет осуществить перемещение строчки текста («прокрутку») на одно знакоместо влево – LCD_SCROLL_LEFT и вправо – LCD_SCROLL_RIGHT.

```
// Примеры использования
LCD_ScrollString("Привет, мир!", 3, LCD_SCROLL_LEFT);
LCD_ScrollString("Привет, мир!", 3, LCD_SCROLL_RIGHT);
```

Скорость перемещения по экрану можно регулировать, включая задержки после исполнения функции.

```
void LCD_PutSymbol(char symbol, uint8_t x, uint8_t y);
```

Функция служит для вывода на экран одиночного символа. Здесь char – символьный тип данных, x – номер знакоместа по горизонтали (от 0 до 15), y – номер знакоместа по вертикали, или, что тоже самое, номер страницы экрана, или номер строчки текста (от 0 до 7).

```
// Примеры использования
LCD_PutSymbol('2', 0, 0);   // Символ 2 отображается
                            // в левом верхнем углу
LCD_PutSymbol('2', 15, 0);  // Символ 2 отображается
                            // в правом верхнем углу
LCD_PutSymbol('2', 0, 7);   // Символ 2 отображается
                            // в левом нижнем углу
LCD_PutSymbol('2', 15, 7);  // Символ 2 отображается
                            // в правом нижнем углу
```

Символ ' находится на нижнем регистре клавиши «Э» при английской раскладке клавиатуры.

Символ можно также записать с помощью таблицы кодировки ASCII, в которой, например, цифре 2 присвоено шестнадцатеричное значение 0x32. В этом случае символ 2 можно записать в левом нижнем углу так:

```
LCD_PutSymbol(0x32, 0, 7);
```

Обратите внимание: никаких кавычек нет.

Рассмотрим еще один способ вывода текста на экран.

Целочисленной переменной *a* присвоим значение, например, 5:

```
int32_t a = 5;
```

Зарезервируем память для строки. 16 знакомест + 1 элемент для указателя конца строки – всего 17 элементов:

```
char stroka[17];
```

Отформатируем строку. Массив – `stroka`, размер массива – 17, вывести результат в формате десятичного знакового числа – `"%d"`, вывести значение переменной `a`:

```
snprintf (stroka, 17, "%d", a);
```

Вывести строку на четвертой странице экрана слева:

```
LCD_PutString(stroka, 4);
```

Заметим, что таким образом могут быть выведены значения не только одной переменной, но и целых выражений. Например, если предварительно задать значения переменных `a = 1`, `b = 2`, `c = 3`, то выражение

```
snprintf(stroka, 17, "%d + %d + %d = %d", a, b, c, a + b + c);
LCD_PutString(stroka, 4);
```

выведет на экран:

```
1 + 2 + 3 = 6
```

Кроме десятичного знакового формата `%d` нам может потребоваться число с плавающей запятой (точкой). Такой формат указывается так: `%f`, при этом после децимальной точки будет 6 знаков. Если количество знаков нужно уменьшить, например, до трех, то следует записать `%.3f`:

```
snprintf (stroka, 17, "%.3f ", a);
```

Разумеется, предварительно необходимо задать тип переменной a и ее значение, например:

```
float a = 5;
```

При этом полный текст программы имеет следующий вид.

```
// Вывод числа с плавающей точкой на экран
// Подключение файла инициализации портов
#include <MDR32F9Qx_port.h>
//Подключение файла инициализации тактового генератора
#include <MDR32F9Qx_rst_clk.h>
// Подключение файла инициализации ЖК-модуля
#include "lcd.h"
char stroka[17];  // Установка размера массива для строки
float a = 5;      // a – число с плавающей запятой (точкой)

// Объявление главной функции
int main()
{
  // Вызов функции инициализации ЖК-модуля
  LCD_Init();
  // Форматирование символа на экране
  snprintf(stroka, 17, "%.3f", a);
  // Расположение символа на 4-й строчке текста
  // (или 4-й странице экрана)
  LCD_PutString(stroka, 4);
  while (1)  // Пустой цикл
  {
  }
}
```

ПРАКТИЧЕСКАЯ ЧАСТЬ

1. Подключите отладочную плату к компьютеру.
2. Подайте питание на плату.
3. Запустите среду программирования Keil µVision.
4. С помощью программатора занесите в микроконтроллер программу вывода числа с плавающей точкой на экран. Наблюдайте результат на экране ЖК-модуля. Измените число значащих цифр после запятой числа на экране.
5. Выведите на экран текст «Студент» на странице 6 экрана.
6. Заставьте перемещаться текст «Студент» по экрану ЖК-модуля. Измените скорость перемещения.

7. Выведите символ «9» в левой части, в правой части, в центре страницы 5 экрана.

8. Используя ASCII-код выведите символ «8» в левой части, в правой части, в центре страницы 7 экрана.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что представляет собой жидкокристаллический модуль MT–12864J?
2. Как вывести строку текста на экран ЖК-модуля?
3. Как удалить строку текста с экрана ЖК-модуля?
4. Как осуществить перемещение строчки текста по экрану ЖК-модуля?
5. Как вывести одиночный символ на экран ЖК-модуля?
6. Как вывести число с плавающей запятой на экран ЖК-модуля?

ЛАБОРАТОРНАЯ РАБОТА 8.

НАСТРОЙКА МОДУЛЯ АЦП

ЦЕЛЬ РАБОТЫ

Изучить работу модуля АЦП и научиться его настраивать.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Аналоговым называют сигнал, который может принимать любое значение в заданном диапазоне. Аналого-цифровой преобразователь (АЦП) – устройство, преобразующее входной аналоговый сигнал в цифровой код (цифровой сигнал).

Для того чтобы преобразовать аналоговый сигнал в цифровой, необходимо выполнить три операции: дискретизацию, квантование и кодирование.

Дискретизация – представление непрерывного аналогового сигнала последовательностью его значений (отсчетов). Эти отсчеты берутся в моменты времени t, отделенные друг от друга интервалом Δt, который называется интервалом дискретизации (рисунок 8.1). Величину, обратную интервалу дискретизации, называют частотой дискретизации.

Чем меньше интервал дискретизации и, соответственно, выше частота дискретизации, тем меньше различия между исходным сигналом и его дискретным отражением. Восстановление аналогового сигнала из дискретного осуществляется с помощью фильтра нижних частот. Согласно теореме В.А. Котельникова, восстановление будет точным только в том случае, если частота дискретизации $f_{\text{Д}}$ больше чем в 2 раза превышает верхнее значение частоты $f_{\text{В}}$ в спектре аналогового сигнала: $f_{\text{Д}} > 2 \cdot f_{\text{В}}$.

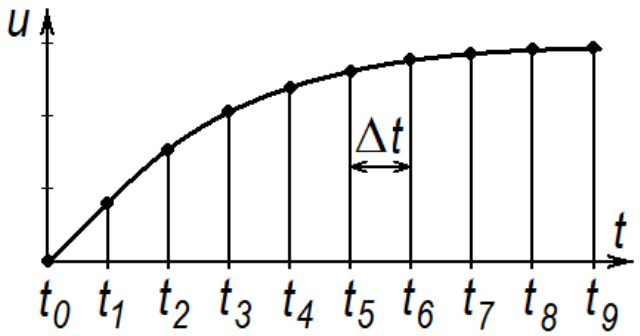

Рисунок 8.1 – Аналого-цифровое преобразование: дискретизация

Квантование представляет собой замену величины отсчета сигнала при дискретизации ближайшим значением из разрешенного набора фиксированных значений сигнала, которые называют уровнем квантования (рисунок 8.2). Таким образом, квантованный сигнал, в отличие от исходных дискретных отсчетов, может принимать только конечное число значений, в данном случае – 0, 1, 2, 3.

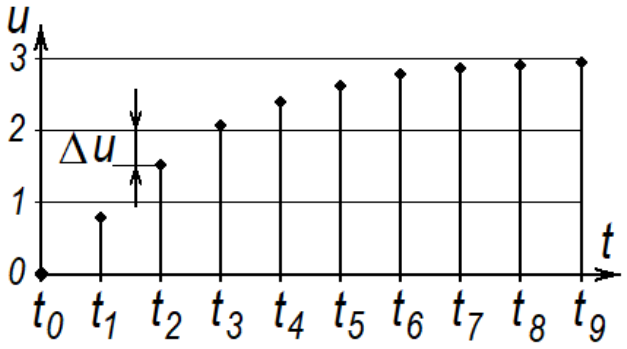

Рисунок 8.2 – Аналого-цифровое преобразование: квантование

При квантовании происходит искажение сигнала, которое невозможно устранить фильтрацией. Чтобы уменьшить искажения от квантования (так называемый «шум квантования»), необходимо увеличивать количество уровней квантования, тогда разность между дискретными отсчетами и фиксированными уровнями квантования Δu будет меньше.

Поскольку квантованный сигнал может принимать конечное число значений, его можно представить в каждый момент отсчета числом, равным порядковому номеру уровня квантования, т.е. закодировать его. Для кодирования сигналов широко применяют двоичный код. Если двоичный код имеет n разрядов, то с его помощью можно описать $N = 2^n$ уровней квантования. Например, для уровней квантования, показанных на рисунке 8.2, двоичные коды будут выглядеть следующим образом: 00; 01; 10; 11.

В состав микроконтроллера входят два 12-разрядных АЦП, (найдите блоки ADC на структурной схеме, рисунок 1.1). Каждый АЦП имеет 8 каналов, пронумерованных от 0 до 7: ADC0, ..., ADC7. Выводы микроконтроллера, которые используются для подключения источников аналогового сигнала, приведены в таблице 8.1.

Обратим внимание, что канал ADC7 не выведен на разъем X26 (поз. 28 на рисунке 1.3), и может быть использован двумя способами. Если перемычка X5 (поз. 21 рисунка 1.3) находится в положении «EXT_CON», то к каналу ADC7 подключается коаксиальный разъем

BNC (поз. 20 рисунка 1.3), на который с помощью дополнительного кабеля можно подать внешний аналоговый сигнал от 0 до 3,3 В. Если перемычка X5 находится в положении «TRIM», то к каналу ADC7 подключается многооборотный подстроечный резистор R1 (поз. 22 рисунка 1.3), с помощью которого на седьмой канал АЦП от внутреннего источника отладочной платы можно подать постоянное напряжение от 0 до 3,3 В. Мы будем работать именно с подстроечным резистором.

Таблица 8.1 – Распределение входов АЦП по линиям портов и выводам микроконтроллера К1986ВЕ92QI

Линия регистра	Вывод микроконтр.	Вывод разъема X26	Аналоговая функция
PD0	31	PD0	ADC0_REF+
PD1	32	PD1	ADC1_REF-
PD2	33	PD2	ADC2
PD3	34	PD3	ADC3
PD4	30	PD4	ADC4
PD5	35	PD5	ADC5
PD6	36	PD6	ADC6
PD7	29	–	ADC7

Для настройки АЦП используется библиотека *MDR32F9Qx_adc.h*, которая описывает работу АЦП с помощью задания структур `ADC_InitTypeDef` и `ADCx_InitTypeDef`.

Структура `ADC_InitTypeDef` имеет следующие поля:

– ADC_SynchronousMode – выбор режима работы двух преобразователей;

– `ADC_StartDelay` – определяет задержку начала преобразований от старта системы (0...15 тактов);

– `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 внутреннего стабилизированного источника 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.

Однако мы не станем прописывать каждое поле, а большинство установок сделаем по умолчанию, используя функции ADC_StructInit и ADCx_StructInit. Что конкретно при этом будет включено или выключено, можно найти в библиотечном файле *MDR32F9Qx_adc.c*.

Ниже приведена программа, которая позволяет оцифровывать постоянное напряжение, подаваемое на седьмой канал первого АЦП, и выводить результат на экран ЖК-модуля.

```
// Подключение заголовочных файлов необходимых библиотек
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_adc.h>
#include "lcd.h"

// Определение функции задержки
#define DELAY(T) for (i = T; i > 0; i--)
// Определение калибровочной константы
#define KALIBR 1247
// Объявление переменных
int i;
float U;
uint32_t RESULT;
char stroka[17];  // Размер массива для строки
```

```
// Процедура общей настройки АЦП
void ADCInit()
{
  // Включение тактирования АЦП
  RST_CLK_PCLKcmd(RST_CLK_PCLK_ADC, ENABLE);
  // Объявление структур для общей настройки АЦП
  ADC_InitTypeDef ADC;
  // Загрузка значений по умолчанию в структуру ADC
  ADC_StructInit(&ADC);
  // Инициализация АЦП объявленной структурой
  ADC_Init(&ADC);
}

// Процедура настройки АЦП1
void ADC1Init()
{
  // Объявление структур для общей настройки АЦП1
  ADCx_InitTypeDef ADC1;
  // Загрузка значений по умолчанию в структуру ADC1
  ADCx_StructInit(&ADC1);
  // Установка номера канала АЦП,
  // подключенного к резистору R1 платы
  ADC1.ADC_ChannelNumber = ADC_CH_ADC7;
  // Инициализация первого АЦП объявленной структурой
  ADC1_Init(&ADC1);
  // Включение первого АЦП
  ADC1_Cmd(ENABLE);
}

// Объявление главной функции
int main ()
{
  ADCInit();   // Вызов функции общей настройки АЦП
  ADC1Init();  // Вызов функции индивидуальной настройки АЦП1
  LCD_Init();  // Вызов функции инициализации ЖК-модуля
  // Основной цикл
  while (1)
  {
    ADC1_Start();  // Начало преобразования
         // Ожидание флага завершения преобразования
         while (ADC1_GetFlagStatus(ADC1_FLAG_END_OF_CONVERSION)
== 0);
    // Чтение результата преобразования
    RESULT = ADC1_GetResult() & 0x00000FFF;
    // Калибровка результата преобразований
    U = (float)RESULT / KALIBR;
    // Вывод результата на экран
```

```
    snprintf(stroka, 17, "U = %.2fB", U);
    LCD_PutString(stroka, 4);
    // Задержка изображения на экране
    DELAY(0xFFFF);
  }
}
```

В дополнительных разъяснениях, пожалуй, нуждается только способ записи результата. Результат преобразования автоматически помещается в 32-разрядный регистр `ADC1_RESULT`. Сам результат преобразования записывается в разряды (биты) с 0 по 11, разряды с 12 по 15 и с 21 по 31 не используются, а вот в разрядах с 16 по 20 записывается номер канала, по которому получен результат. Но сейчас нам номер канала не нужен, поэтому чтобы 32-разрядный регистр хранил только результат преобразования, содержимое регистра необходимо логически умножить на двоичное число $00000000000000000000111111111111_2$, или в шестнадцатеричном коде – 0x00000FFF, что и было сделано операцией

```
RESULT = ADC1_GetResult() & 0x00000FFF;
```

Отметим, что такая операция называется «маскирование» или «наложение маски».

Чтобы перевести полученный числовой результат в вольты, разделим его на калибровочную константу, которая подбирается экспериментально путем сопоставления показаний вольтметра, которым следует измерить напряжение на входе АЦП, с отображением результата на экране ЖК-модуля:

```
U = (float)RESULT / KALIBR;
```

ПРАКТИЧЕСКАЯ ЧАСТЬ

1. Убедитесь, что перемычка X5 (поз. 21 на рисунке 1.3) стоит в положении TRIM. В этом случае ко входу АЦП будет подключен подстроечный резистор R1 (поз. 22 на рисунке 1.3).

2. Подключите отладочную плату к компьютеру.

3. Подайте питание на плату.

4. Запустите среду программирования Keil µVision.

5. Занесите программу в микроконтроллер с помощью программатора. Наблюдайте результат на экране ЖК-модуля. Измените число значащих цифр после запятой числа на экране.

6. Предельно аккуратно вставьте наконечник тонкой отвертки в шлиц винтового стержня построечного резистора R1 и сделайте несколько оборотов по движению или против движения часовой стрелки. Наблюдайте за изменениями значений оцифрованного напряжения на экране ЖК-модуля.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какой сигнал называют аналоговым?
2. Какой сигнал называют цифровым?
3. Что такое дискретизация сигнала?
4. Что такое квантование сигнала?
5. Что понимают под кодированием при аналого-цифровом преобразовании?
6. Сколько каналов у встроенного аналого-цифрового преобразователя микроконтроллера К1986ВЕ92QI?
7. Сколько разрядов содержит код на выходе АЦП?
8. Поясните логику работы программы, приведенной в данной лабораторной работе.

ЛАБОРАТОРНАЯ РАБОТА 9.

МОДУЛЬ УНИВЕРСАЛЬНОГО АСИНХРОННОГО ПРИЕМОПЕРЕДАТЧИКА

ЦЕЛЬ РАБОТЫ

Изучить работу модуля универсального асинхронного приемопередатчика UART и научиться его настраивать.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

В данной работе мы осуществим вывод информации из микроконтроллера на монитор персонального компьютера (ПК). Одним из возможных вариантов такой передачи является схема, показанная на рисунке 9.1.

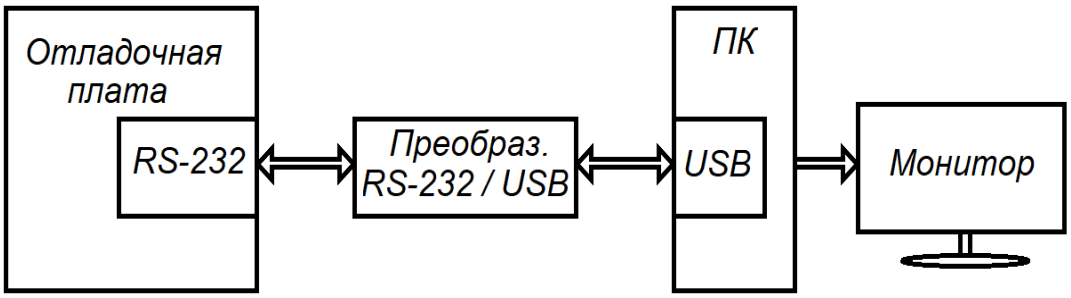

Рисунок 9.1 – Схема соединений отладочной платы с персональным компьютером

Расположение разъема для интерфейса RS-232 (COM-порт) отладочной платы показана на рисунке 1.3, поз. 3. Если персональный компьютер имеет аналогичный разъем, то отладочная плата соединяется с ПК напрямую кабелем RS-232 9F-9F если у компьютера имеется только разъем для интерфейса USB, потребуется преобразователь интерфейсов, который выполнен также в виде кабеля. В последнем случае после установки программного обеспечения у ПК появится виртуальный COM-порт, с которым можно работать также, как с COM-портом, существующим физически.

Для обмена информацией между компьютером и внешним терминалом существуют различные программы – мы воспользуемся свободно распространяемой программой *Terminal v1.9b*, интерфейс которой весьма прост и интуитивно ясен.

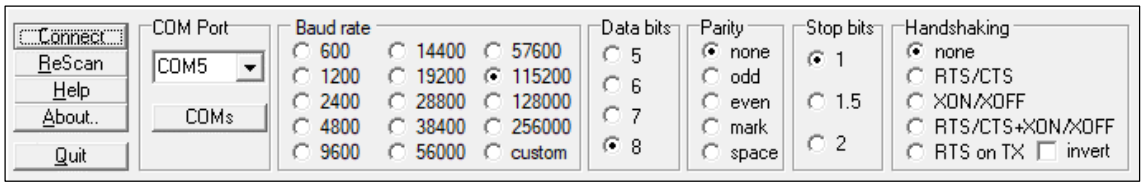

Рисунок 9.2 – Настройка интерфейса программы *Terminal v1.9b*

После подключения кабелей и запуска программы следует настроить режим обмена:

– скорость обмена *Baud rate*, допустим, 115200 бод;

– количество бит в одном пакете данных *Data bits* – 8;

– проверка данных на четность *Parity* – *none* (отсутствует);

– количество стоповых битов *Stop bits* – 1;

– управление потоком данных *Handshaking* – *none* (отсутствует).

Далее следует нажать кнопку *Connect* (соединить) и выбрать из выпадающего меню COM-порт, по которому будет осуществляться обмен.

Поле, в котором будет располагаться получаемая компьютером информация, также подлежит настройке.

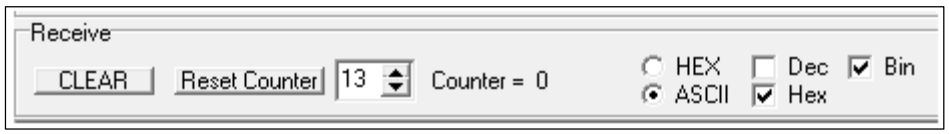

Рисунок 9.3 – Настройка поля приема информации

Для того чтобы передаваемые символы (буквы, цифры, пробелы и т.п.) отображались привычным для нас образом, выберем кодировку *ASCII*; для контроля приходящих кодов, которые будут располагаться в правой части экрана, можем выбрать *Dec* (десятеричный), *Hex* (шестнадцатеричный), *Bin* (двоичный) либо не выбирать ничего. Счетчик *Counter* подсчитывает, сколько раз тот или иной символ был получен компьютером. Код подсчитываемого символа, например, 13 (перевод строки), выставляется вручную. Сбрасывается счетчик подсчета символов кнопкой *Reset Counter*. Очистить поле приема информации можно кнопкой *CLEAR*.

Теперь поговорим о микроконтроллере. Модуль универсального асинхронного приемопередатчика UART (Universal Asynchronous Receiver-Transmitter) представляет собой периферийное устройство микроконтроллера (найдите его на рисунке 1.1). Модуль передает информацию побитно, для чего преобразует данные, передаваемые на периферийное устройство, из параллельной в последовательную фор-

му. Получает информацию также побитно, для чего преобразует данные, полученные от периферийного устройства, из последовательной в параллельную форму.

Микроконтроллер содержит два однотипных блока: UART1 и UART2. Для передачи информации от каждого блока необходимо три проводника: по одному осуществляется передача данных (transmit), по второму – прием (receive); третий провод – земля (ground). В таблице 9.1 приведено распределение функций UART по выводам микроконтроллера. Обратим внимание на то, что функции UART относятся к альтернативным либо переопределенным функциям порта и нуждаются в настройке.

Таблица 9.1 – Распределение функций UART по линиям портов и выводам микроконтроллера К1986ВЕ92QI

Линия порта	Вывод микроконтр.	Цифровая функция	
		Альтернативная	Переопределенная
PA6	57	–	UART1_RXD
PA7	56	–	UART1_TXD
PB0	43	–	UART1_TXD
PB1	44	–	UART2_RXD
PB5	50	UART1_TXD	–
PB6	51	UART1_RXD	–
PD0	31	UART2_RXD	–
PD1	32	UART2_TXD	–
PD7	29	–	UART1_RXD
PF0	2	–	UART2_RXD
PF1	3	–	UART2_TXD

Для обмена данными с компьютером выберем блок UART2 и настроим линии PF0 и PF1 порта F соответственно на прием (переопределенная цифровая функция UART2_RXD) и на передачу (переопределенная цифровая функция UART2_TXD).

Наша первая программа будет совсем простой: передача по одному байту от микроконтроллера на ПК, тем не менее прием данных от ПК микроконтроллером также будет описан.

Для работы с блоком UART будет использована 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 буфера FIFO: осуществлять передачу по нескольку байт или побайтно;

– UART_HardwareFlowControl – включает/выключает аппаратный контроль потока.

При работе с UART-контроллером применяются функции записи данных UART_SendData и чтения UART_ReceiveData.

Рассмотрим программу, которая периодически посылает на ПК текст «UART», переводит строку и делает «возврат каретки», т.е. ставит курсор в начало строки.

```
// Подключение необходимых библиотечных файлов
#include <MDR32F9Qx_uart.h>
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
// Определение функции задержки
#define DELAY(T) for (i = T; i > 0; i--)
int i; // Глобальная переменная счетчика в макроса DELAY()

// Процедура инициализации порта
void PortsInit()
{
  // Включение тактирования порта F
  RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTF, ENABLE);
  // Объявление структуры для инициализации порта
  PORT_InitTypeDef Nastroyka;
  // Инициализация порта B для функции UART
  // Настройка порта по умолчанию
  PORT_StructInit(&Nastroyka);
  // Переопределение функции порта
  Nastroyka.PORT_FUNC = PORT_FUNC_OVERRID;
  // Установка короткого фронта
  Nastroyka.PORT_SPEED = PORT_SPEED_MAXFAST;
  // Цифровой режим работы вывода
  Nastroyka.PORT_MODE = PORT_MODE_DIGITAL;
  // Инициализация вывода PF1 как UART_TX (передача)
  Nastroyka.PORT_Pin = PORT_Pin_1;
  Nastroyka.PORT_OE  = PORT_OE_OUT;
  PORT_Init(MDR_PORTF, &Nastroyka);
  // Инициализация вывода PF0 как UART_RX (прием)
  Nastroyka.PORT_Pin = PORT_Pin_0;
  Nastroyka.PORT_OE  = PORT_OE_IN;
  PORT_Init(MDR_PORTF, &Nastroyka);
}
// Процедура инициализации контроллера UART
void UARTInit()
```

```
{
  // Включение тактирования UART2
  RST_CLK_PCLKcmd(RST_CLK_PCLK_UART2, ENABLE);
  // Объявление структуры для инициализации контроллера UART
  UART_InitTypeDef UART_InitStructure;
  // Делитель тактовой частоты UART = 1
  UART_BRGInit(MDR_UART2, UART_HCLKdiv1);
  // Конфигурация UART
  // Скорость передачи данных – 115200 бод
  UART_InitStructure.UART_BaudRate = 115200;
  // Количество бит в посылке – 8
  UART_InitStructure.UART_WordLength = UART_WordLength8b;
  // Один стоп-бит
  UART_InitStructure.UART_StopBits = UART_StopBits1;
  // Без проверки четности
  UART_InitStructure.UART_Parity = UART_Parity_No;
  // Выключить работу буфера FIFO приемника и передатчика,
  // т.е. передача осуществляется по одному байту
  UART_InitStructure.UART_FIFOMode = UART_FIFO_OFF;
  // Разрешить прием и передачу данных
  UART_InitStructure.UART_HardwareFlowControl =
    UART_HardwareFlowControl_RXE
  | UART_HardwareFlowControl_TXE;
  // Инициализация UART2 с заданными параметрами
  UART_Init(MDR_UART2, &UART_InitStructure);
  // Включить сконфигурированный UART
  UART_Cmd(MDR_UART2, ENABLE);
}

// Объявление главной функции
int main(void)
{
  PortsInit();  // Вызов функции инициализации порта
  UARTInit();   // Вызов функции инициализации UART
  // Бесконечный цикл
  while (1)
  {
    UART_SendData(MDR_UART2, 'U');  // Передать символ U
    DELAY(50000);                   // Задержка
    UART_SendData(MDR_UART2, 'A');  // Передать символ A
    DELAY(50000);                   // Задержка
    UART_SendData(MDR_UART2, 'R');  // Передать символ R
    DELAY(50000);                   // Задержка
    UART_SendData(MDR_UART2, 'T');  // Передать символ T
    DELAY(50000);                    // Задержка
    UART_SendData(MDR_UART2, '\n');  // Перевод строки
    DELAY(50000);                    // Задержка
```

```
    UART_SendData(MDR_UART2, '\r');  // Возврат каретки
    DELAY(500000);                   // Задержка
        }
}
```

ПРАКТИЧЕСКАЯ ЧАСТЬ

1. Подключите отладочную плату к компьютеру так, как это делалось в предыдущих лабораторных работах.

2. Соедините COM-порт отладочной платы с компьютером так, как это показано на рисунке 9.1

3. Подайте питание на плату.

4. Запустите среду программирования Keil µVision.

5. Запустите и настройте программу Terminal v1.9b.

6. С помощью программатора занесите в микроконтроллер программу передачи информации на монитор компьютера. Наблюдайте результат на экране.

7. Внесите изменение в передаваемое сообщение.

8. Измените скорость передачи сообщения.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как организовать обмен данными между микроконтроллером и персональным компьютером? Какие интерфейсы при этом используются?

2. Как настроить интерфейс программы *Terminal v1.9b* для обмена данными между микроконтроллером и персональным компьютером?

3. Как настроить порты микроконтроллера для приема и передачи информации с помощью UART?

4. Как изменить скорость обмена данными между микроконтроллером и персональным компьютером?

5. Поясните логику работы программы, приведенной в данной лабораторной работе.

ЗАКЛЮЧЕНИЕ

Постоянно и стремительно развиваясь, микроконтроллеры предоставляют разработчику систем управления всё бóльшие возможности, но при этом и освоение микроконтроллеров становится сложнее. За отведенное на изучение дисциплины время мы сумели сделать лишь первые робкие шаги, но, уверен, что и они вызывают определенные затруднения.

В последнее время издано немало методических пособий, помогающих освоить микроконтроллеры, названия некоторых из них приведены в библиографическом списке.

Работа в среде проектирования Keil μVision, в том числе вопросы отладки программ, которые совсем не затрагиваются в настоящем практикуме, прекрасно изложены в работах [1], [2] и [4].

Сведения о языке Си, в объеме, достаточном для программирования микроконтроллеров, приведены в учебно-методических пособиях [3] и [4].

В работе [5] более подробно, чем обычно, описывается структура микроконтроллера, хотя, конечно, наиболее полное описание приведено в фирменной спецификации [9].

Познакомиться с применением микроконтроллера в системах автоматического регулирования можно в учебном пособии [6].

Учебное пособие [7] поможет глубже понять механизмы прерываний в микроконтроллере.

Расширить свои знания об аналого-цифровых преобразователях можно, прочитав работы [1], [3], [6], [7], [8].

Принцип действия универсального асинхронного приемопередатчика описан в работах [3], [4], 5], [8].

Однако надо понимать, что сведения, изложенные в указанных пособиях, невозможно зазубрить, они будут полезны лишь при условии активного освоения микроконтроллера путем написания собственных программ в среде программирования и проверки их на отладочной плате.

Дальнейшее, более углубленное, изучение микропроцессорной техники предполагается в рамках магистерской программы подготовки. Будут рассмотрены такие вопросы, как цифро-аналоговое преобразование, передача данных с использованием интерфейсов I^2C, SPI, USB, прямой доступ в память – словом, всё то, что имеется на отладочной плате, но не изучено из-за недостатка времени.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

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

2. Благодаров А.В. Программирование микроконтроллеров на основе отечественных микросхем семейства 1986ВЕ9х разработки и производства компании «Миландр» / А.В. Благодаров, Л.Л. Владимиров. – М., 2016. – 242 с.: ил.

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

4. Пуговкин А.В. Методическое пособие по программированию микроконтроллеров: Учебно-методическое пособие / А.В. Пуговкин, И.А. Куан, Н.К. Ахметов, А.В. Бойченко. – Томск, 2016. – 69 с.

5. Строганова С.М. Методические указания к выполнению лабораторных работ по микроконтроллерам семейства 1986ВЕ9Х компании «Миландр» для студентов специальности 27.03.04 «Управление в технических системах» / С.М. Строганова, Н.Н. Теодорович. – Королев, 2016. – 85 с.

6. Лабораторный практикум по микроконтроллерам семейства Cortex-M: Методическое пособие по проведению лабораторных работ на отладочных платах фирмы «Миландр» / сост.: В.Г. Рубанов, А.С. Кижук, Д.А. Бушуев, Е.Б. Карик, Е.П. Добринский. – Белгород: Изд-во БГТУ, 2016. – 61 с.

7. Огородников И.Н. Микропроцессорная техника: введение в Cortex-M3: учеб. пособие / И.Н. Огородников. – Екатеринбург: Изд-во Урал. ун-та, 2015. – 116 с.

8. Торгаев С.Н. Программирование микроконтроллеров с ядром Cortex-M3 в задачах диагностики и контроля: учебное пособие / С.Н Торгаев., И.С. Мусоров, А.А. Солдатов, П.В. Сорокин. – Томск: STT, 2017. – 104 с.

Режим доступа к источникам [1] – [8]: https://edu.milandr.ru/library/

9. Микросхема 32-разрядной однокристальной микро-ЭВМ с памятью Flash-типа 1986ВЕ9ху, К1986ВЕ9ху, К1986ВЕ9хуК, К1986ВЕ92QI, К1986ВЕ92QC, 1986ВЕ91Н4, К1986ВЕ91Н4, 1986ВЕ94Н4, К1986ВЕ94Н4 / Спецификация. – Режим доступа: https://ic.milandr.ru/products/mikrokontrollery_i_protsessory/32_razryadnye_mikrokontrollery/1986ve9kh_yadro_arm_cortex_m3/k1986ve92qi/#docs_tab

СОДЕРЖАНИЕ

ДЛЯ ЗАМЕТОК

ДЛЯ ЗАМЕТОК

УЧЕБНОЕ ИЗДАНИЕ

Евдокимов Алексей Петрович
Владимиров Леонид Леонидович

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРА К1986ВЕ92QI КОМПАНИИ «МИЛАНДР»

Лабораторный практикум
по дисциплине «Электроника и микропроцессорная техника»
для студентов, обучающихся по направлению подготовки
13.03.02 Электроэнергетика и электротехника,
профили «Электроснабжение» и «Релейная защита и автоматизация электроэнергетических систем»
(все формы обучения)

В авторской редакции

Компьютерная верстка *Дергачевой Е.С.*

Подписано в печать 10.10. 2018. Формат 60x84$^{1/16}$.
Усл. печ. л. 4,42. Тираж 100. Заказ.
ИПК ФГБОУ ВПО Волгоградский ГАУ «Нива».
400002, Волгоград, пр. Университетский, 26.