МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра вычислительной техники

УТВЕРЖДАЮ
Проректор по учебной работе
О.Г. Локтионова
«30» 09 2022 г.

Аппаратно-программное обеспечение инфраструктуры систем искусственного интеллекта

Методические указания к лабораторным работам
для студентов направления подготовки
09.04.01 очной формы обучения

Курск 2022

УДК 001.89

Составители: С. И. Кирносенко, В. И. Конченков, В. Н. Скакунов, А.В. Киселев

Рецензент
Кандидат технических наук, доцент *Т.Н. Конаныхина*

Аппаратно-программное обеспечение инфраструктуры систем искусственного интеллекта: методические указания к лабораторным работам для студентов направления подготовки 09.04.01 очной формы обучения / Юго-Зап. гос. ун-т; сост.; С. И. Кирносенко, В. И. Конченков, В. Н. Скакунов, А.В. Киселев. – Курск, 2022. - 67 с.: - ил. 23, табл. 7.– Библиогр.: с. 67.

Методические указания содержат описание учебного комплекса лабораторных работ, построенного на основе отладочной платы EasyAVR6. Рассмотрены возможности интегрированной среды разработки Atmel Studio.

Предназначены для студентов направления подготовки 09.04.01 очной формы обучения.

Методические указания соответствуют рабочей программе дисциплины «Аппаратно-программное обеспечение инфраструктуры систем искусственного интеллекта».

Текст печатается в авторской редакции

Подписано в печать . Формат 60∗84 1/16.
Усл. печ. л. 2,85. Уч.-изд. л. 2,58. Тираж 50 экз. Заказ . Бесплатно.
Юго-Западный государственный университет.
305040 Курск, ул. 50 лет Октября, 94.

1. ЛАБОРАТОРНЫЕ РАБОТЫ

1.1. ЛАБОРАТОРНАЯ РАБОТА № 1

Основы программирования микроконтроллеров семейства ATmega

Цель работы: изучение особенностей архитектуры, системы команд и программирования микроконтроллеров семейства ATmega, освоение приемов написания и отладки программ на ассемблере в интегрированной среде разработки Atmel Studio.

1. Архитектура микроконтроллеров семейства ATmega

Микроконтроллеры AVR удачно совмещает преимущества RISC-архитектуры с достижениями фирмы Atmel в области создания Flash-памяти, что сделало их весьма популярным на мировом рынке 8-разрядных микроконтроллеров (МК) [1,4,5].

Семейство ATmega объединяет большое количество микроконтроллеров различной степени сложности и функциональных возможностей. Так, например, микроконтроллер ATmega16 имеет следующие аппаратные особенности:

- 8-разрядное арифметико-логическое устройство (АЛУ);
- 32 регистра общего назначения;
- внутренняя Flash-память программ объемом 16 Кбайт;
- внутренняя EEPROM-память данных объемом 512 байт;
- внутреннее ОЗУ данных (SRAM) объемом 1 Кбайт;
- 4 параллельных порта, предоставляющих 32 линии ввода/вывода;
- 3 программируемых таймера/счетчика с функцией ШИМ;
- 10-разрядный 8-канальный АЦП и аналоговый компаратор;
- последовательные интерфейсы: USART, SPI, I2C;

- пиковая производительность 16MIPS при максимальной частоте 16 МГц.

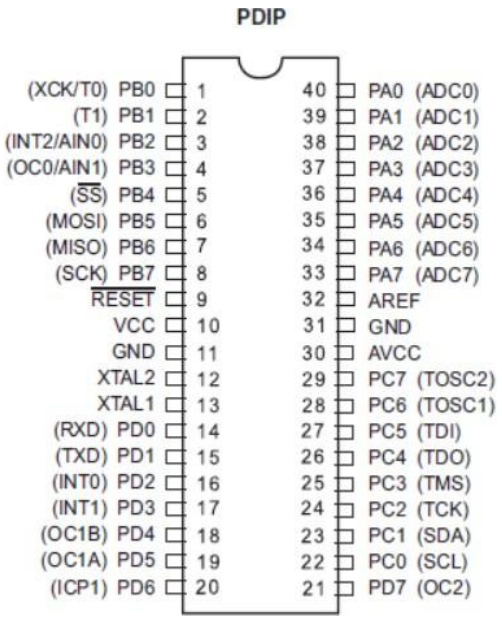


Рисунок 1.1. Наименование линий микроконтроллера ATmega16

На Рисунок 1.1 приведено назначение выводов микроконтроллера ATmega16, выполненного в DIP-корпусе. В скобках указаны альтернативные функции выводов, если они имеются.

Структурная схема микроконтроллера ATmega16 представлена на рис. 2.1. Обмен данными в параллельном коде, как и во всех МК, выполняется через стандартные цифровые порты ввода-вывода. В МК ATMega16 порт A (PA0-PA7), порт B (PB0-PB7), порт C (PC0-PC7) и порт D (PD0-PD7) представляют собой 8-разрядные двунаправленные порты ввода/вывода со встроенными нагрузочными резисторами. Выходные буферы обеспечивают ток 20 мА и способны прямо управлять светодиодным индикатором. При использовании выводов портов в качестве входов и установке внешнего сигнала в низкое состояние, ток будет вытекать только при подключенных встроенных нагрузочных резисторах. Порт A при наличии внешней памяти данных используется для организации мультиплексируемой шины адреса/данных, а также может предоставлять аналоговые входы для АЦП.

Порты B, C и D используются также при реализации специальных функций. Линия RESET отводится входному сигналу сброса, для выполнения которого необходимо удерживать низкий уровень на входе не менее 50 нс. Линии XTAL1 и XTAL2 – выводы встроенного усилителя генератора тактовой частоты, к которым подключается внешний кварцевый резонатор. Линия AVCC – напряжение питания аналого-цифрового преобразователя, которая подсоединяется к внешнему питанию через низкочастотный фильтр. Линия AREF — вход опорного напряжения для АЦП в диапазоне между GND и AVCC. Линии VCC и GND — напряжение питания и земля.

Для повышения производительности используется Гарвардская архитектура и, следовательно, память программ и память данных разделены, а доступ к ним осуществляется по различным шинам. Инструкции из памяти программ исполняются с одноуровневой конвейеризацией. Пока одна инструкция непосредственно исполняется, следующая загружается из памяти. Это позволяет выполнять очередную инструкцию на каждом такте микроконтроллера. В качестве памяти программ используется внутренняя Flash-память. Она организована в виде матрицы 16-разрядных ячеек и может загружаться с помощью внешнего программатора или через последовательный порт SPI. Регистры общего назначения включают 32 8-разрядных регистра со временем доступа в пределах одного такта, что обеспечивает выполнение многих операций 8-разрядного арифметико-логического устройства (ALU) за один такт. При выполнении типичной двухадресной команды в ALU оба операнда загружаются из регистров общего назначения, а после выполнения операции результат записывается также в регистр общего назначения. Шесть последних из 32-х регистров общего назначения могут быть использованы как три 16-разрядных адресных указателя X, Y, и Z для адресации памяти данных.

В состав микроконтроллера включен достаточно большой набор периферийных блоков, обеспечивающих эффективную работу с внешними устройствами. К их числу относят параллельные и последовательные порты,

таймеры/счетчики, подсистему прерывания, устройства аналогового интерфейса.

Таймеры/счетчики выполняют общесистемные функции по формированию временных интервалов для системных прерываний, генераторов периодических и одиночных сигналов, функции сторожевого таймера, а также дополнительные функции: сравнения, захвата, широтно-импульсной модуляции (ШИМ), достаточно подробно рассмотренные в разделе 1.2.2 настоящего пособия. Микроконтроллер ATmega16 содержит два 8-разрядных таймера/счетчика (T/C0, T/C2) и один 16-разрядный таймер/счетчик (T/C1). В таймерах T/C0, T/C2 реализованы функции счетчика и одноканального ШИМ-модулятора, а в 16-разрядном таймере T/C1 дополнительно введены режимы сравнения/захвата и 4-канального ШИМ-модулятора. Кроме того, в контроллере имеется программируемый сторожевой таймер (WDT) со встроенным генератором тактовых импульсов.

В микроконтроллерах семейства Mega последовательные порты общего назначения USART дополняются встроенными специальными последовательными интерфейсами – SPI, I2C, JTAG. Для порта USART могут быть установлены синхронный и асинхронный режимы передачи данных. К числу основных характеристик относятся широкий диапазон скоростей, фильтрация помех, возможность обнаружения трех типов ошибок: по переполнению буфера приемника, по нарушению формата данных, по ошибке на паритет и формирование запросов прерывания от трех событий: по завершению передачи или приема, по пустому буферу передатчика.

Интерфейсы SPI и I2C обеспечивают высокоскоростной дуплексный синхронный обмен данными при организации связи МК с различными измерительными датчиками и интегральными схемами с аналогичными встроенными интерфейсами. Возможности интерфейсов рассмотрены в разделе 1.2.2. Кроме того, интерфейс SPI используется при внутрисхемной загрузке внутренней Flash-памяти контроллера, а интерфейс I2C широко применяется для построения распределенных систем управления, в

частности, для организации каналов связи в информационно-управляющих и многопроцессорных системах. В AVR-контроллерах семейства Mega встроен интерфейс TWI (Two Wire Service Interface), который является полным аналогом базовой версии I2C.

Интерфейс JTAG (Joint Test Action Group) первоначально разрабатывался для решения проблем тестирования как промышленный стандарт подключения сложных цифровых микросхем или устройств к аппаратуре тестирования и отладки. В интегрированных коммуникационных модулях микроконтроллеров интерфейс JTAG применяется как способ связи между несколькими устройствами системы, имеющими аналогичный интерфейс, и для внутрисхемного программирования микроконтроллеров.

Микроконтроллер ATmega16 имеет многоуровневую приоритетную систему прерываний от 20 внешних и внутренних источников прерываний. Таблица векторов прерываний занимает младшие адреса в памяти программ. Приоритет прерывания определяется положением вектора прерывания в таблице. Вектор содержит адрес команды безусловного перехода к подпрограмме обработки прерывания. Каждый запрос прерываний может быть программно разрешен или запрещен.

Аналоговый интерфейс микроконтроллера представляют аналого-цифровой преобразователь (АЦП) и схема компаратора. Модуль 10-разрядного АЦП последовательного приближения имеет 8-канальный аналоговый мультиплексор. Быстродействие АЦП – до 15 тыс. оп./с. Предусмотрено два режима работы: непрерывного преобразования через заданные интервалы времени с циклическим опросом каналов и одиночного запуска из программы пользователя. Основная функция компаратора – сравнение аналоговых напряжений на двух внешних выводах МК. Результат сравнения в виде двоичной переменной может быть считан из программы и, кроме того, вызвать прерывание. По прерыванию захватывается состояние таймера Т/С1, что позволяет реализовать операцию по измерению длительности аналоговых сигналов.

2. Организация памяти микроконтроллеров семейства ATmega

Микроконтроллеры AVR имеют раздельные пространства адресов памяти программ и данных (Гарвардская архитектура). Организация памяти показана на рис 2.2.

Особенности архитектуры микроконтроллера отражаются на организации памяти.

- В качестве памяти программ используется внутренняя Flash-память. Она построена в виде матрицы 16-разрядных ячеек и может загружаться программатором или через последовательный порт SPI.
- Все инструкции имеют длину в 16 или 32 разряда. Память программ и 16-разрядная шина команд, поддерживающие формат инструкций, вместе с одноуровневым конвейером позволяют выполнить большинство инструкций за один такт синхрогенератора (50 нс при частоте FOSC=20 МГц).

Рисунок 1.2. Организация памяти микроконтроллеров семейства ATmega

- Память данных имеет 8-разрядную организацию. Младшие 32 адреса пространства занимают регистры общего назначения (РОН), далее

следуют 64 адреса стандартных регистров ввода-вывода и в старших моделях (ATmega128, ATmega64) 160 адресов дополнительных регистров ввода-вывода, а затем внутреннее ОЗУ данных объемом до 4096 ячеек. Возможно применение внешнего ОЗУ данных объемом до 60 Кбайт.

- Внутренняя энергонезависимая память типа EEPROM объемом до 4 Кбайт (в старших моделях) размещена в изолированном от общей памяти адресном пространстве. Обращение к памяти осуществляется через специальные регистры, в которых задаются адрес, данные и тип операции – чтение/запись.

7 1	Addr	
R31	$1F	Старший байт регистра Z
R30	$1E	Младший байт регистра Z
R29	$1D	Старший байт регистра Y
R28	$1C	Младший байт регистра Y
R27	$1B	Старший байт регистра X
R26	$1A	Младший байт регистра X
…		
R17	$11	
R16	$10	
R15	$0F	
R14	$0E	
R13	$0D	
…		
R2	$02	
R1	$01	
R0	$00	

Рисунок 1.3. Регистры общего назначения микроконтроллеров семейства

ATmega Как видно из Рисунок 1.2, все 32 регистра общего назначения включены в единое адресное пространство ОЗУ данных и занимают младшие адреса. Файл регистров общего назначения прямо связан с АЛУ (рис.1.2), каждый из регистров способен работать как аккумулятор.

Большинство команд выполняются за один такт, при этом из регистров

файла могут быть выбраны два операнда, выполнена операция и результат возвращен в регистровый файл. Старшие шесть регистров файла могут объединяться в три 16- разрядных регистра X,Y,Z и выполнять функции адресных указателей (Рисунок 1.3).

Следующие 64 адреса за регистрами общего назначения занимают стандартное пространство служебных регистров ввода-вывода (реализовано во всех моделях). В этой области сгруппированы регистры управления и состояния (статуса) внутренних программируемых периферийных устройств. При использовании команд IN и OUT адреса этих регистров размещены в отдельном диапазоне: $00 – $3F. Но в то же время к регистрам ввода-вывода можно обращаться и как к ячейкам внутреннего ОЗУ с адресами единого адресного пространства (Рисунок 1.2). При этом к непосредственному адресу ввода-вывода прибавляется смещение - $20. Адрес регистра как ячейки ОЗУ приводится далее в круглых скобках. Регистры ввода-вывода с $00 ($20) по $1F ($3F) имеют программно доступные биты. Обращение к ним осуществляется командами SBI и CBI, а проверка состояния - командами SBIS и SBIC.

В старших моделях микроконтроллеров AVR (ATmega128, ATmega64) часть регистров ввода-вывода размещена в расширенном адресном пространстве по адресам с $60 по $FF. Обращение к этим регистрам производится как к ячейкам ОЗУ данных командами LD/LDS/LDD, ST/STS/STD.

В Приложении 1 приведен список регистров ввода-вывода. В пространстве регистров ввода-вывода находятся регистры управления процессором микроконтроллера: регистр состояния, указатель стека, регистр выбора страницы, регистр управления коэффициентом деления частот, выполняющие общесистемные функции.

Регистр состояния SREG

	7	6	5	4	3	2	1	0	
$3F($5F)	I	T	H	S	V	N	Z	C	SREG
Исходное значение	0	0	0	0	0	0	0	0	

Bit7 - Разрешение всех прерываний. Для разрешения прерываний этот бит должен быть установлен. Разрешение конкретного прерывания выполняется регистрами маски прерывания EIMSK и TIMSK. Если этот бит сброшен, то ни одно из прерываний не обрабатывается. Бит аппаратно очищается после возникновения прерывания и устанавливается, разрешая последующие прерывания, командой RETI.

Bit 6 - Бит сохранения копии. Команды копирования бита BLD и BST используют этот бит как источник и приемник при операциях с битами. Командой BST бит регистра общего назначения копируется в бит T, командой BLD бит T копируется в бит регистра общего назначения.

Bit 5 - Флаг полупереноса. Флаг полупереноса указывает на перенос между тетрадами при выполнении ряда арифметических операций.

Bit 4 - Бит знака. Бит S имеет значение результата операции “Исключающее ИЛИ” (N xor V) над флагами отрицательного значения (N) и дополнения до двух флага переполнения (V) .

Bit 3 - Флаг переполнения. Этот бит поддерживает арифметику дополнения до двух.

Bit 2 - Флаг отрицательного значения. Этот флаг указывает на отрицательный результат ряда арифметических и логических операций.

Bit 1 - Флаг нулевого значения. Этот флаг указывает на нулевой результат ряда арифметических и логических операций.

Bit 0 - Флаг переноса. Этот флаг указывает на перенос при арифметических и логических операциях.

Более подробная информация о флагах регистра состояния микроконтроллера приведена в описании системы команд.

Указатель стека — SP

	7	6	5	4	3	2	1	0	
$3E($5E)	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
$3D($5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
Исходное	0	0	0	0	0	0	0	0	
значение	0	0	0	0	0	0	0	0	

Микроконтроллеры AVR оснащены 16-разрядным указателем стека, размещенном в двух регистрах ввода-вывода. Поскольку микроконтроллеры ATmega128 поддерживают объем ОЗУ до 64 Кбайт, то используются все 16 разрядов указателя стека.

Указатель стека указывает на область в ОЗУ данных, в которой размещается стек подпрограмм и обработчиков прерываний. Начальный адрес указателя должен задаваться программно в процессе инициализаци. Стек растет в область младших адресов. Начальное значение адреса в указателе стека должно быть значительно больше $60 для размещения необходимого объема данных. Указатель стека декрементируется на единицу при каждом занесении командой PUSH данных в стек и на две единицы при занесении в стек адреса при вызове подпрограммы или процедуры прерывания.

Указатель стека инкрементируется на единицу при извлечении данных из стека командой POP и на две единицы при извлечении адреса из стека при возврате из подпрограммы (RET) или возврате из процедуры прерывания (RETI).

Регистр выбора страницы ОЗУ – RAMPZ

	7	6	5	4	3	2	1	0	
$3B($5B)	-	-	-	-	-	-	-	RAMPZ0	RAMPZ
Исходное значение	0	0	0	0	0	0	0	0	

Регистр RAMPZ используется обычно для определения страницы ОЗУ данных, к которой возможно обращение посредством указателя Z. Поскольку микроконтроллеры ATmega128 не поддерживают ОЗУ объемом более 64 К, то этот регистр используется только для выбора страницы в памяти программ при использовании команды ELPM. Имеются следующие варианты использования единственного значащего бита RAMPZO:

RAMPZO = 0. Команде ELPM доступна память программ с адресами от $0000 до S7FFF (младшие 64 Кбайт);

RAMPZO = 1. Команде ELPM доступна память программ с адресами от $8000 до $FFFF (старшие 64 Кбайт).

Отметим, что на команду LPM установки регистра RAMPZ не воздействуют.

Микроконтроллер ATmega64 не содержит регистра RAMPZ и не имеет команды ELPM. Команда LPM способна перекрыть все пространства памяти программ микроконтроллера ATmega64.

Регистр управления делением частоты кварцевого генератора – XDIV

	7	6	5	4	3	2	1	0	
$3C($5C)	XDIVEN	XDIV6	XDIV5	XDIV4	XDIV3	XDIV2	XDIV1	XDIV0	XDIV
Исходное значение	0	0	0	0	0	0	0	0	

Bit (6 - 0) - Биты выбора коэффициента деления. Эти биты определяют коэффициент деления тактовой частоты при установленном бите XDIVEN. Если десятичное значение этих семи битов обозначить через d, то результирующая тактовая частота CPU будет вычисляться по формуле: FCLK=XTAL/(129-d)

Состояния этих битов можно изменить только при сброшенном бите XDIVEN. При установленном бите XDIVEN, записанное ранее в биты XDIV6

XDIV0 значение будет определять коэффициент деления. При сбросе бита XDIVEN записанные в биты XDIV6 – XDIV0 значения игнорируются. Тактовая частота, уменьшенная делителем, поступает и на MCU и на периферийные устройства с тем же коэффициентом деления.

Система команд и ассемблер микроконтроллеров семейства ATmega

Подробное описание команд и ассемблера приведено в прилагаемых материалах AVR Instruction Set.

3. Среда разработки Atmel Studio

Atmel Studio – это интегрированная среда разработки для микроконтроллеров семейства ATmega, бесплатная и доступная для загрузки после регистрации на официальном сайте (http://www.atmel.com/). Она поддерживает разработку программ, как на Ассемблере так и на языке С, а также включает отладчик, редактор исходного кода, ассемблер и прочие утилиты.

Для начала работы следует запустить исполняемый файл atmelstudio.exe. После чего откроется пустое главное окно программы. Далее необходимо создать новый проект через меню «File/New/Project...» (Рисунок 1.4).

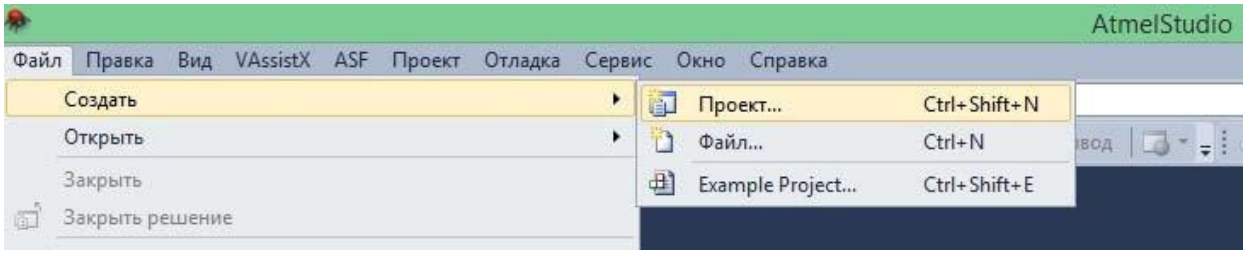


Рисунок 1.4. Создание проекта в среде Atmel Studio.

В открывшемся диалоговом окне выбираем тип проекта AVR Assembler Project, вводим название проекта, выбираем расположение (Рисунок 1.5). Наличие в имени или пути символов кириллицы или пробелов может

привести к проблемам при работе с проектом. В следующем окне выбираем контроллер - семейство megaAVR, 8bit, ATMega16 (Рисунок 1.6). Обратите внимание, что после наведения курсора на название контроллера справа отображается информация о доступных средства отладки и программирования для этого контроллера, а также ссылка на инструкцию (datasheet) для этого контроллера. После нажатия кнопки «OK» в диалоге будет создан новый проект, в главном окне будет отображена полностью работоспособная рабочая область (Рисунок 1.7). Центральную область главного окна занимает редактор кода, в котором и следует набирать исходный код программы.

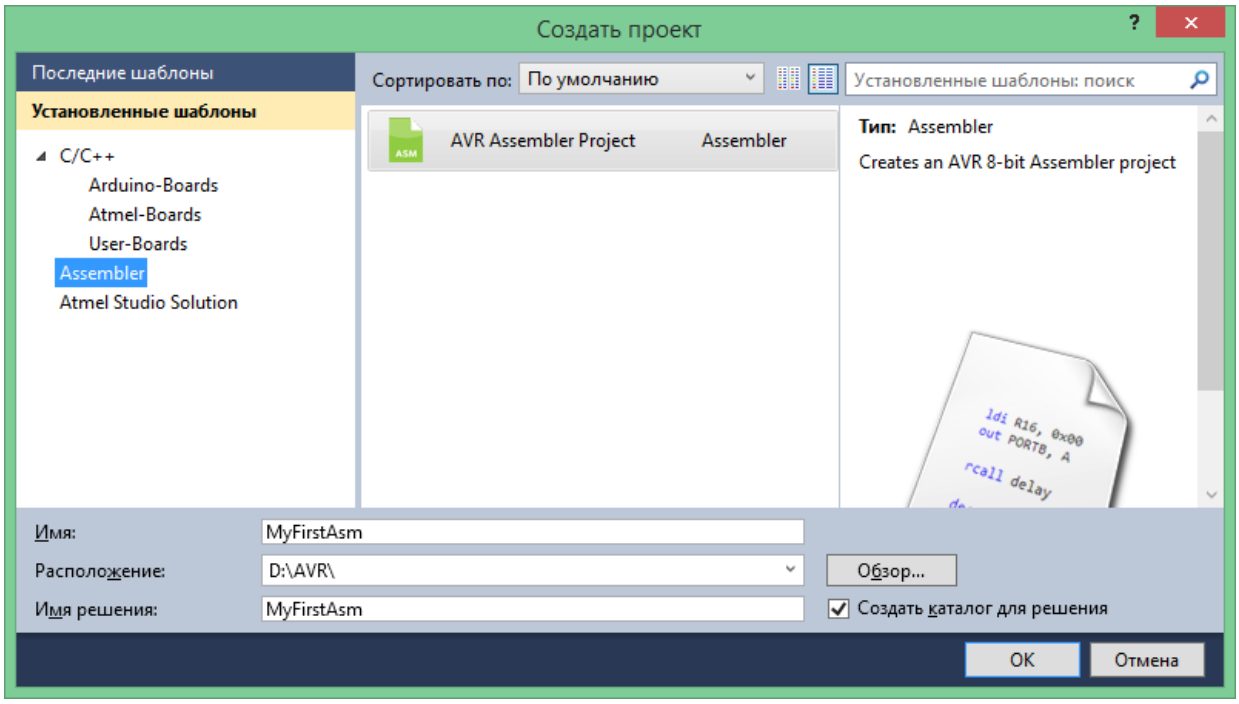


Рисунок 1.5. Выбор типа проекта.

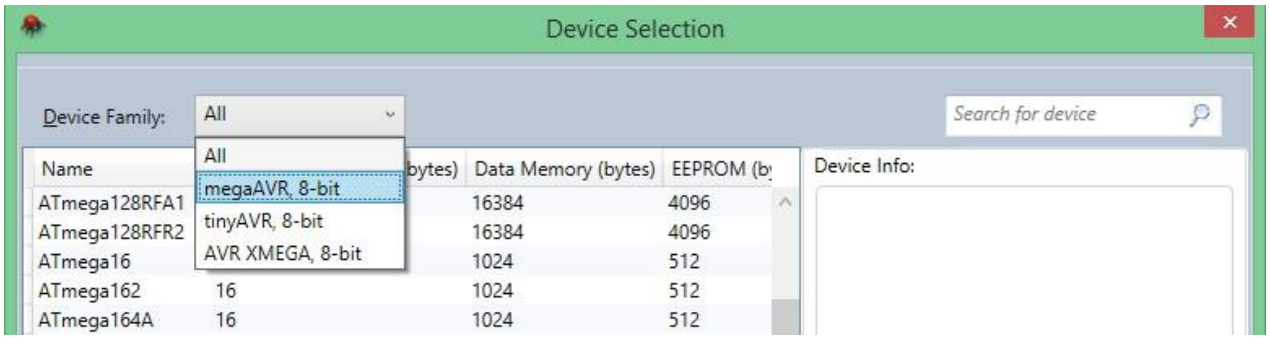


Рисунок 1.6. Выбор типа микроконтроллера.

Как только исходный код набран, можно выполнить компиляцию посредством команды меню «Build/Build Solution» (F7). В случае неудачной компиляции в нижней области главного окна на закладке «Error List» будут перечислены обнаруженные в набранном коде ошибки. Если компиляция прошла успешно можно переходить к отладке программы посредством команды меню «Debug/Continue» (F5). Подобно другим средам Atmel Studio во время отладки предоставляет команды для выполнения очередной инструкции «Debug/Step over» (F10) и «Debug/Step into» (F11), прогона программы до конца или до следующей точки останова «Debug/Continue» (F5). Точки останова, в свою очередь, могут быть установлены в любой момент, если кликнуть по полю левее строки кода в редакторе.

Во время отладки в окне «Processor» отображается состояние регистров. Также доступны для просмотра память данных и программ.

При запуске отладки в первый раз, скорее всего, появится информационное окно с требованием выбрать отладчик (Рисунок 1.8). В открывшемся окне в разделе Tool выбираем Simulator (Рисунок 1.9) и сохраняем этот конфигурационный файл.

4. Общее задание

Рассмотреть приведенный в листинге 1 пример программы для сложения массивов 8-разрядных чисел. Выполнить ее отладку, убедиться в правильности ее работы, отметить к каким изменениям приводит каждая команда и как эти изменения отражаются на состоянии регистров и памяти контроллера.

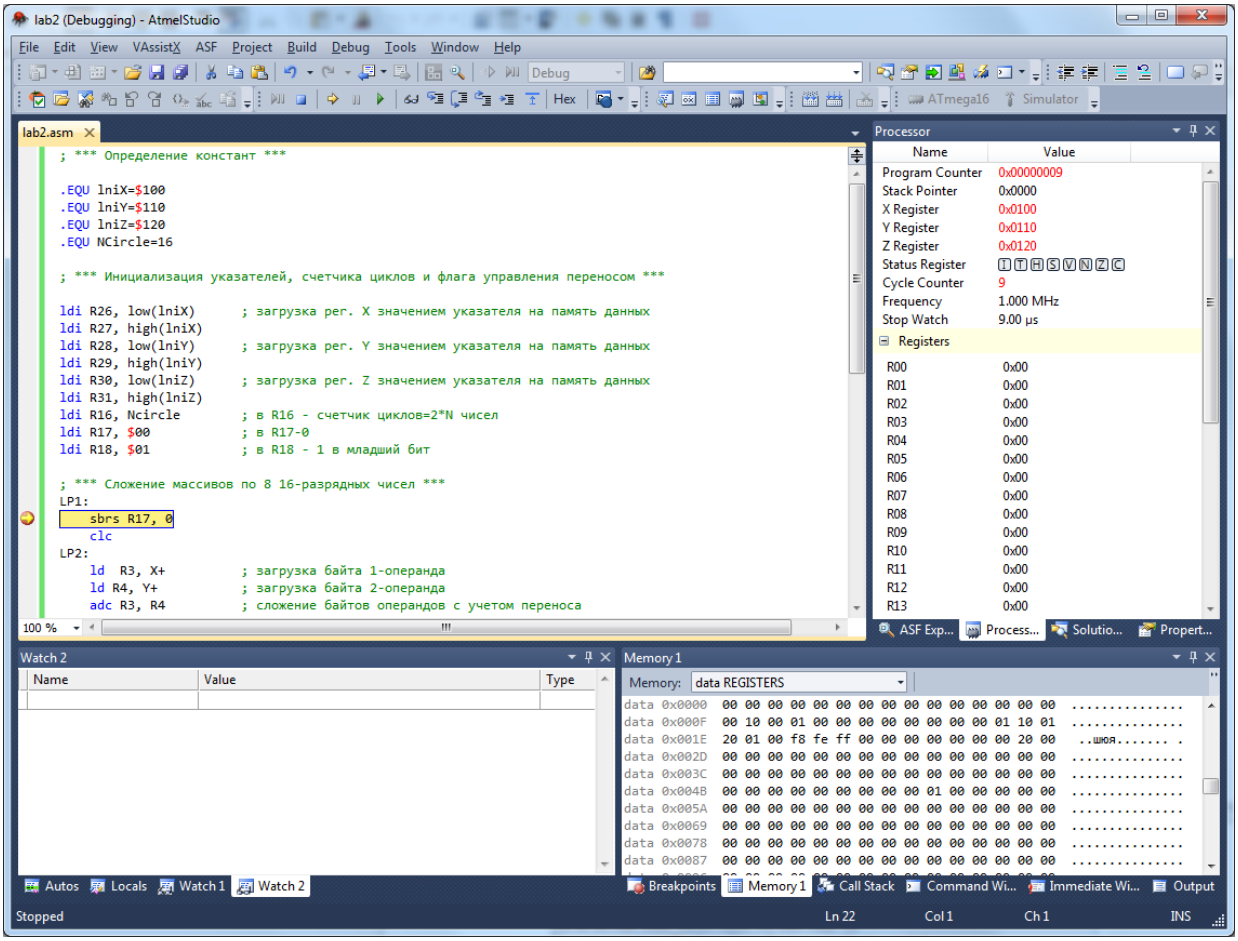


Рисунок 1.7. Основное окно программы.

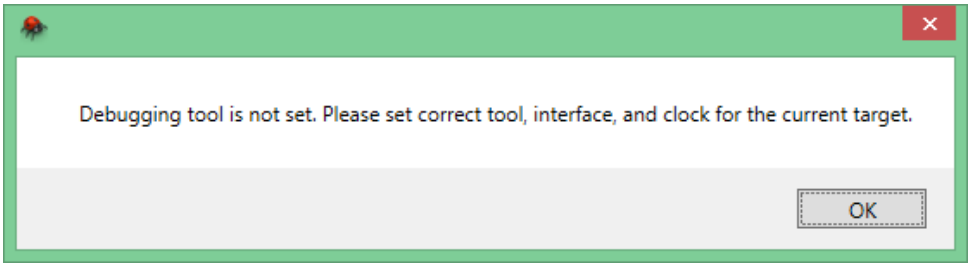


Рисунок 1.8. Информационное окно с требованием выбора отладчика

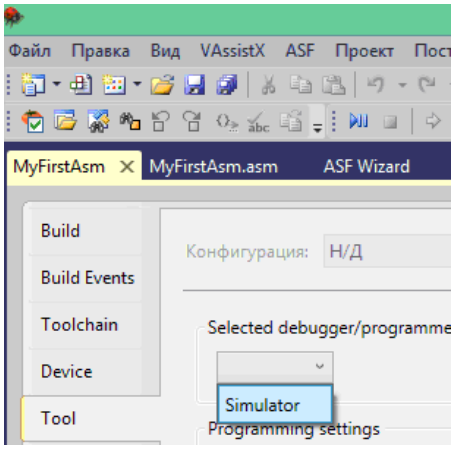


Рисунок 1.9. Выбор отладчика

Листинг 1. Программа поэлементного сложения двух массивов 8-битных целых чисел

```
.include "m16def.inc"      ;ATmega16
;
; *** Определение констант ***
.EQU lniRes=$140
.EQU N=8
; заносим исходные значения элементов массива в память
       ldi R16, N
       mov R1, R16                ; счетчик итераций по массиву

       ldi XL, low(tabA*2) ; помещаем в регистр-указатель адрес массива tabA
       ldi XH, high(tabA*2); память программ хранит 16-битные слова, поэтому
                           ; адрес домножаем на 2

       ldi YL, low(tabB*2) ; помещаем в регистр-указатель адрес массива tabB
       ldi YH, high(tabB*2); память программ хранит 16-битные слова, поэтому
                           ; адрес домножаем на 2

       ldi R19, 1          ; храним единицу
       ldi R22, 0          ; храним номер текущего элемента массива

LOADX:                     ; загружаем в регистр R17 очередное значение из tabA
       mov ZL, XL          ; копируем указатель X в указатель Z
       mov ZH, XH
       lpm R17, Z          ; помещаем в R17 значение по указателю Z
       adc XL, R19         ; наращиваем указатель XL на единицу
       brcs L1             ; если возникло переполнение XL, наращиваем XH
       jmp LOADY                ; переходим к загрузке элемента из массива tabB
L1:
       inc XH
LOADY:                     ; загружаем в регистр R18 очередное значение из tabB
       mov ZL, YL          ; копируем указатель Y в указатель Z
       mov ZH, YH
       lpm R18, Z          ; помещаем в R18 значение по указателю Z
       adc YL, R19         ; наращиваем указатель YL на единицу
       brcs L2             ; если возникло переполнение YL, наращиваем YH
       jmp SUM             ; переходим к суммированию соответствующих элементов массива
L2:
       inc YH
SUM:                       ; суммирование соответствующих элементов массива
       mov R20, R17
       add R20, R18

TO_MEM:                    ; вычисляем адрес, куда будем записывать очередной результат
       ldi ZL, low(lniRes*2) ; помещаем в регистр-указатель адрес массива, в котором
       ldi ZH, high(lniRes*2); будет храниться результат
       adc ZL, R22         ; наращиваем указатель ZL  на номер текущего элемента массива
       brcs L3
       jmp NEXT
L3:
       inc ZH
NEXT:
       st Z, R20           ; помещаем сумму в память
       inc R22
       dec R1
       brne LOADX
MAINLOOP:
       jmp MAINLOOP        ; бесконечный массив
       ; задаем два массива 8-битных чисел
tabA: .db 80, 90, 100, 110, 115, 120, 130, 130
tabB: .db 50, 60, 70, 80, 90, 100, 110, 120
```

5. Индивидуальные задания

1. Выполнить попарное сложение 16-разрядных чисел со знаком, содержащихся в двух массивах объемом по 16 чисел с начальными адресами $xx, $yy, результаты сохранить в массиве с начальным адресом $zz. В двух произвольных регистрах в виде маски сохранить при сложении каких пар чисел возникло переполнение. Общее количество переполнений также сохранить в произвольном регистре.

2. Выполнить попарное вычитание 16-разрядных чисел со знаком, содержащихся в двух массивах объемом по 10 чисел с начальными адресами $xx, $yy, результаты сохранить в массиве с начальным адресом $zz. При переполнении результат округлять до максимального положительного или отрицательного значения. Общее количество округлений сохранить в произвольном регистре.

3. Выбрать числа с минимальным и максимальным значением из массива, содержащего 20 16-разрядных чисел со знаком, которые размещены в памяти, начиная с адреса $xx. В произвольных регистрах сохранить сами числа, а также их порядковые номера в исходном массиве данных.

4. Выполнить расчет среднего значения для массива из 16 16-разрядных чисел со знаком.

5. Выполнить сравнение 16-разрядных чисел, содержащихся в массивеобъемом из 20 чисел с начальным адресом $xx с эталоном, хранящимся в произвольных регистрах. Числа, несовпадающие с эталоном, разместить в массиве с начальным адресом $yy. Общее число несовпадающих чисел сохранить в произвольном регистре.

Приложение 1

Таблица П1

Регистры ввода-вывода микроконтроллеров семейства ATmega

Адрес I/O (адрес SRAM)	Обозначение	Функция
$9D	UCSR1C	Управляющий регистр C USART1 (USART Control/Status Register C)
$9C	UDR1	Регистр данных USART1 (USART Data Register)
$9B	UCSR1A	Управляющий регистр A USART1 (USART Control/Status Register A)
$9A	UCSR1B	Управляющий регистр B USART1 (USART Control/Status Register B)
$99	UBRR1L	Регистр управления скоростью USART1, мл. байт (USART Baud Rate Register)
$98	UBRR1H	Регистр управления скоростью USART1, ст. байт (USART Baud Rate Register)
$95	UCSR0C	Регистр управления USARTO (USART Control/Status Register)
$90	UBRR0H	Регистр управления скоростью USART0. ст. байт(USART Baud Rate Register)
$8C	TCCR3C	Управляющий регистр C таймера/счетчика 3 (Timer/Counter3 Control Register C)
$8B	TCCR3A	Управляющий регистр A таймера/счетчика 3 (Timer/Counter3 Control Register A)
$8A	TCCR3B	Управляющий регистр B таймера/счетчика 3 (Timer/Counter3 Control Register B)
$89	TCNT3H	Старший байт таймера/счетчика 3 (Timer/Counter3 High Byte)
$88	TCNT3L	Младший байт таймера/счетчика 3 (Timer/Counter3 Low Byte)
$87	OCR3AH	Старший байт регистра A сравнения выхода таймера/счетчика 3 (Timer/Counter3 Output Compare Register A High Byte)
$86	OCR3AL	Младший байт регистра A сравнения выхода таймера/счетчика 3 (Timer/Counter3 Output Compare Register A Low Byte)
$85	OCR3BH	Старший байт регистра B сравнения выхода таймера/счетчика 3 (Timer/Counter3 Output Compare Register B High Byte)
$84	OCR3BL	Младший байт регистра B сравнения выхода таймера/счетчика 3 (Timer/Countcr3 Output Compare Register B Low Byte)
$83	OCR3CH	Старший байт регистра C сравнения выхода таймера/счетчика 3 (Timer/Counter3 Output Compare Register C High Byte)
$82	OCR3CL	Младший байт регистра C сравнения выхода таймера/счетчика 3 (Timer/Counter3 Output Compare Register C Low Byte)
$81	ICR3H	Старший байт регистра захвата таймера/счетчика 3 (Timer/Counter3 Input Capture Register High Byte)
$80	ICR3L	Младший байт регистра захвата таймера/счетчика 3 (Timer/Counter3 Input Capture Register Low Byte)
$7D	ETIMSK	Дополнительный регистр масок прерываний по таймерам/счетчикам (Timer Interrupt MaSK register)
$7C	ETIFR	Дополнительный регистр флагов прерываний по таймерам/счетчикам (Timer/Counter Interrupt Flag Register)

Продолжение таблицы П1

Адрес I/O (адрес SRAM)	Обозначение	Функция
$7A	TCCR1C	Управляющий регистр C таймера/счетчика 1 (Timer /Counter 1 Control Register C)
$79	OCR1CH	Старший байт регистра C сравнения выхода таймера/счетчика 1 (Timer/Counter 1 Output Compare Register C High Byte)
$78	OCR1CL	Младший байт регистра C сравнения выхода таймера/счетчика 1 (Timer/Counter 1 Output Compare Register C Low Byte)
$74	TWCR	Регистр управления двухпроводным интерфейсом TWI (Two-Wire Serial Interface Control Register)
$73	TWDR	Регистр данных TWI (Two-Wire Serial Interface Data Register)
$72	TWAR	Регистр адреса TWI (Two-Wire Serial Interface Address Register)
$71	TWSR	Регистр состояния TWI (Two-Wire Serial Interface Status Register)
$70	TWBR	Регистр управления скоростью TWI (Two-Wire Serial Interface Bit Rate Register)
$6F	OSCCAL	Регистр калибровки встроенного RC генератора (Oscilator Calibration Register)
$6D	XMCRA	Регистр управления A доступом к внешней памяти данных (eXternal Memory Control Register A)
$6C	XMCRB	Регистр управления B доступом к внешней памяти данных (eXternal Memory Control Register B)
$6A	EICRA	Регистр управления внешними прерываниями A (External Interrupt Control Register A)
$68	SPMCSR	Регистр управления самопрограммированием Flash памяти программ (Self Program Memory Control and Status Register)
$65	PORTG	Регистр данных порта G (Data Register, Port G)
$64	DDRG	Регистр направления данных порта G (Data Direction Register, Port G)
$63	PING	Регистр чтения входов порта G (Input Pins, Port G)
$62	PORTF	Регистр данных порта F (Data Register, Port F)
$61	DDRF	Регистр направления данных порта F (Data Direction Register, Port F)
$3F($5F)	SREG	Регистр статуса (Status REGister)
$3E($5F)	SPH	Верхний байт указателя стека (Stack Pointer High)
$3D($5D)	SPL	Нижний байт указателя стека (Stack Pointer Low)
$3C($5C)	XDIV	Регистр управления делением тактовой частоты (XTAL Divide Control Register)
$3B($5B)	RAMPZ	Регистр выбора страницы Z RAM (RAM Page Z Select Register)
$3A($5A)	EICRB	Регистр управления внешними прерываниями B (External Interrupt Control Register B)
$39($59)	EIMSK	Регистр масок внешних прерываний (External Interrupt MaSK register)
$38($58)	EIFR	Регистр флагов внешних прерываний (External Interrupt Flag Register)
$37($57)	TIMSK	Регистр масок прерываний по таймерам/счетчикам (Timer/ Interrupt MaSK register)
$36($56)	TIFR	Регистр флагов прерывания по таймерам/счетчикам (Timer/Counter Interrupt Flag Register)

Продолжение таблицы П1

Адрес I/O (адрес SRAM)	Обозначение	Функция
$35($55)	MCUCR	Регистр управления MCU (MCU General Control Register)
$34($54)	MCUSR	Регистр статуса MCU (MCU Status Register)
$33($53)	TCCR0	Регистр управления таймером/счетчиком 0 (Timer/Counter0 Control Register)
$32($52)	TCNT0	Таймер/счетчик 0 (Timer/Counter0 (8bit))
$31($51)	OCR0	Регистр сравнения выхода таймера/счетчика 0 (Timer/Counter0 Output Compare Register)
$30($50)	ASSR	Регистр статуса асинхронного режима (Asynchronous Mode Status Register)
$2F($4F)	TCCR1A	Управляющий регистр A таймера/счетчика 1 (Timer/Counter 1 Control Register A)
$2E($4E)	TCCR1B	Управляющий регистр B таймера/счетчика 1 (Timer/Counter 1 Control Register B)
$2D($4D)	TCNT1H	Старший байт таймера/счетчика 1 (Timer/Counter 1 High Byte)
$2C(44C)	TCNT1L	Младший байт таймера/счетчика 1 (Timer/Counter 1 Low Byte)
$2B($4B)	OCR1AH	Старший байт регистра A сравнения выхода таймера/счетчика 1 (Timer/Counter 1 Output Compare Register A High Byte)
$2A($4A)	OCR1AL	Младший байт регистра A сравнения выхода таймера/счетчика 1 (Timer/Counter 1 Output Compare Register A Low Byte)
$29($49)	OCR1BH	Старший байт регистра B сравнения выхода таймера/счетчика 1 (Timer/Counter 1 Output Compare Register B High Byte)
$28($48)	OCR1BL	Младший байт регистра B сравнения выхода таймера/счетчика 1 (Timer/Counter 1 Output Compare Register B Low Byte)
$27($47)	ICR1H	Старший байт регистра захвата таймера/счетчика 1 (Timer/Counter1 Input Capture Register High Byte)
$26($46)	ICR1L	Младший байт регистра захвата таймера/счетчика 1 (Timer/Counter1 Input Capture Register Low Byte)
$25($45)	TCCR2	Регистр управления таймером/счетчиком 2 (Timer/Counter2 Control Register)
$24($44)	TCNT2	Таймер/счетчик 2 (Timer/Counter2 (8bit))
$23($43)	OCR2	Регистр сравнения выхода таймера/счетчика 2 (Timer/Counter2 Output Compare Register)
$22($42)	OCDR	Регистр управления отладочным режимом (On-Chip Debug Register)
$21($41)	WDTCR	Регистр управления сторожевым таймером (Watchdog Timer Control Register)
$20($40)	SFIOR	Регистр управления специальными функциями ввода-вывода (Special Function IO Register)
$1F($3F)	EEARH	Старший байт регистра адреса EEPROM (EEPROM Address Register High)
$1E($3E)	EEARL	Младший байт регистра адреса EEPROM (EEPROM Address Register Low)
$1D($3D)	EEDR	Регистр данных EEPROM (EEPROM Data Register)
$1C($3C)	EECR	Регистр управления EEPROM (EEPROM Control Register)
$1B($3B)	PORTA	Регистр данных порта A (Data Register, Port A)

Окончание таблицы П1

Адрес I/O (адрес SRAM)	Обозначение	Функция
$1A($3A)	DDRA	Регистр направления данных порта A (Data Direction Register, Port A)
$19($39)	PINA	Регистр чтения входов порта A (Input Pins, Port A)
$18($38)	PORTB	Регистр данных порта B (Data Register, Port B)
$17($37)	DDRB	Регистр направления данных порта B (Data Direction Register, Port B)
$16($36)	PINB	Регистр чтения входов порта B (Input Pins, Port B)
$15($35)	PORTC	Регистр данных порта C (Data Register, Port C)
$12($32)	PORTD	Регистр данных порта D (Data Register, Port D)
$11($31)	DDRD	Регистр направления данных порта D (Data Direction Register, Port D)
$10($30)	PIND	Регистр чтения входов порта D (Input Pins, Port D)
$0F($2F)	SPDR	Регистр данных SPI I/O (SPI I/O Data Register)
$0E($2E)	SPSR	Регистр статуса SPI (SPI Status Register)
$0D($2D)	SPCR	Регистр управления SPI (SPI Control Register)
$0C($2C)	UDR0	Регистр данных USART0 (USART Data Register)
$0B($2B)	UCSR0A	Управляющий регистр A USART0 (USART Control/Status Register A)
$0A($2A)	UCSR0B	Управляющий регистр B USART0 (USART Control/Status Register B)
$09($29)	UBRR0L	Регистр управления скоростью USART0, мл. байт (USART Baud Rate Register)
$08($28)	ACSR	Регистр статуса и управления аналогового компаратора (Analog Comparator Control and Status Register)
$07($27)	ADMUX	Регистр выбора мультиплексора ADC (ADC Multiplexer Select Register)
$06($26)	ADCSR	Регистр статуса и управления ADC (ADC Control and Status Register)
$05($25)	ADCH	Старший байт регистра данных ADC (ADC Data Register High)
$04($24)	ADCL	Младший байт регистра данных ADC (ADC Data Register Low)
$03($23)	PORTE	Регистр данных порта E (Data Register, Port E)
$02($22)	DDRE	Регистр направления данных порта E (Data Direction Register, Port E)
$01($21)	PINE	Регистр чтения входов порта E (Input Pins, Port E)
$00($20)	PINF	Регистр чтения входов порта F (Input Pins, Port F)

Приложение 2

Таблица П2

Команды пересылки данных

Мнемоника	Операнды	Описание	Операция	Флаги	Кол-во циклов
ELPM	Rd,Z 0≤d≤31	Расширенная загрузка из памяти программ в регистр	Rd ← (Z+RAMPZ)	Нет	3
ELPM	Rd,Z+ 0≤d≤31	Расширенная загрузка из памяти программ в регистр с постинкрементом	Rd ← (Z+RAMPZ) Z ← Z+1	Нет	3
MOV	Rd,Rr 0≤d≤31 0≤r≤31	Копировать регистр	Rd ← Rr	Нет	1
LDI	Rd,K 16≤d≤31 0≤k≤255	Загрузить непосредственное значение	Rd ← K	Нет	1
LDS	Rd,k 0≤d≤31 0≤k≤65535	Загрузить из ОЗУ	Rd ← (k)	Нет	2
LD	Rd,X 0≤d≤31	Загрузить косвенно	Rd ← (X)	Нет	2
LD	Rd,X+ 0≤d≤31	Загрузить косвенно с постинкрементом	Rd ← (X) X ← X+1	Нет	2
LD	Rd,-X 0≤d≤31	Загрузить косвенно с преддекрементом	X ← X-1 Rd ← (X)	Нет	2
LD	Rd,Y 0≤d≤31	Загрузить косвенно	Rd ← (Y)	Нет	2
LD	Rd,Y+ 0≤d≤31	Загрузить косвенно с постинкрементом	Rd ← (Y) Y ← Y+1	Нет	2
LD	Rd,-Y 0≤d≤31	Загрузить косвенно с преддекрементом	Y ← Y-1 Rd ← (Y)	Нет	2
LDD	Rd,Y+q 0≤d≤31 0≤q≤63	Загрузить косвенно со смещением	Rd ← (Y+q)	Нет	2
LD	Rd,Z 0≤d≤31	Загрузить косвенно	Rd ← (Z)	Нет	2
LD	Rd,Z+ 0≤d≤31	Загрузить косвенно с постинкрементом	Rd ←(Z) Z ← Z+1	Нет	2
LD	Rd,-Z 0≤d≤31	Загрузить косвенно с преддекрементом	Z ← Z-1 Rd ← (Z)	Нет	2
LDD	Rd,Z+q 0≤d≤31 0≤q≤31	Загрузить косвенно со смещением	Rd ← (Z+q)	Нет	2
ST	X,Rr 0≤r≤31	Записать косвенно	(X) ← Rr	Нет	2

Окончание таблицы П2

Мнемоника	Операнды	Описание	Операция	Флаги	Кол-во циклов
ST	X+,Rr 0≤r≤31	Записать косвенно с постинкрементом	(X) ← Rr X ← X+1	Нет	2
ST	-X,Rr 0≤r≤31	Записать косвенно с преддекрементом	X ← X-1 (X) ← Rr	Нет	2
ST	Y,Rr 0≤r≤31	Записать косвенно	(Y) ← Rr	Нет	2
ST	Y+,Rr 0≤r≤31	Записать косвенно с постинкрементом	(Y) ← Rr Y ← Y+1	Нет	2
ST	-Y,Rr 0≤r≤31	Записать косвенно с преддекрементом	Y ← Y-1 (Y) ← Rr	Нет	2
STD	Y+q,Rr 0≤r≤31 0≤q≤63	Записать косвенно со смещением	(Y+q) ← Rr	Нет	2
ST	Z,Rr 0≤r≤31	Записать косвенно	(Z) ← Rr	Нет	2
ST	Z+,Rr 0≤r≤31	Записать косвенно с постинкрементом	(Z) ← Rr Z ← Z+1	Нет	2
ST	-Z,Rr 0≤r≤31	Записать косвенно с преддекрементом	Z ← Z-1 (Z) ← Rr	Нет	2
STD	Z+q,Rr 0≤r≤31 0≤q≤63	Записать косвенно со смещением	(Z+q) ← Rr	Нет	2
LPM		Загрузить байт из памяти программ	R0 ← (Z)	Нет	3
LPM	Rd,Z 0≤d≤31	Загрузить байт из памяти программ	Rd ← (Z)	Нет	3
LPM	Rd,Z+ 0≤d≤31	Загрузить байт из памяти программ	Rd ← (Z) Z ← Z +1	Нет	3
IN	Rd,P 0≤d≤31 0≤P≤63	Загрузить байт из порта I/O в регистр	Rd ← P	Нет	1
OUT	P,Rr 0≤r≤31 0≤P≤63	Записать данные из регистра в порт I/O	P ← Rr	Нет	1
PUSH	Rr 0≤r≤31	Сохранить регистр в стеке	STACK ← Rr	Нет	2
POP	Rd 0≤d≤31	Выгрузить регистр из стека	Rd ← STACK	Нет	2

Таблица П3

Команды переходов

Мнемоника	Операнды	Описание	Операция	Флаги	Кол-во циклов
RJMP	k -2K≤k≤2K	Перейти относительно	PC ← PC+k+1	Нет	2
IJMP		Перейти косвенно	PC ← Z	Нет	2
JMP	k 0≤k≤4M	Перейти	PC ← k	Нет	3
RCALL	K -2K≤k≤2K	Вызвать подпрограмму относительно	PC ← PC+k+1	Нет	3
ICALL		Вызвать подпрограмму косвенно	PC ← Z	Нет	3
CALL	K 0≤k≤64K	Выполнить длинный вызов подпрограммы	PC ← k	Нет	4
RET		Вернуться из подпрограммы	PC ← STACK	Нет	4
RETI		Вернуться из прерывания	PC ← STACK	1	4
CPSE	Rd,Rr 0≤d≤31 0≤r≤31	Сравнить и пропустить, если равно	if Rd=Rr then PC ← PC+2(or 3)	Нет	1/2/3
SBRC	Rr,b 0≤r≤31 0≤b≤7	Пропустить, если бит в регистре очищен	if Rr(b)=0 then PC ← PC+2(or 3)	Нет	1/2/3
SBRS	Rr,b 0≤r≤31 0≤b≤7	Пропустить, если бит в регистре установлен	if Rr(b)=1 then PC ← PC+2(or 3)	Нет	1/2/3
SBIC	P,b 0≤P≤31 0≤b≤7	Пропустить, если бит в регистре I/O очищен	if I/O(b)=0 then PC ← PC+2(or 3)	Нет	1/2/3
SBIS	P,b 0≤P≤31 0≤b≤7	Пропустить, если бит в регистре I/O установлен	if I/O(b)=1 then PC ← PC+2(or 3)	Нет	1/2/3
BRBS	s,k 0≤s≤7 -64≤k≤+63	Перейти, если бит в регистре статуса установлен	if SREG(s)=1 then PC ← PC+k+1	Нет	1/2
BRBC	s,k 0≤s≤7 -64≤k≤+63	Перейти, если бит в регистре статуса очищен	if SREG(s)=0 then PC ← PC+k+1	Нет	1/2
BREQ	k -64≤k≤+63	Перейти, если равно	if Rd=Rr(Z=1) then PC ← PC+k+1	Нет	1/2
BRNE	k -64≤k≤+63	Перейти, если не равно	if Rd≠Rr(Z=0) then PC ← PC+k+1	Нет	
BRCS	k -64≤k≤+63	Перейти, если флаг переноса установлен	if C=1 then PC ← PC+k+1	Нет	1/2

Окончание таблицы П3

Мнемоника	Операнды	Описание	Операция	Флаги	Кол-во циклов
BRCC	k -64≤k≤+63	Перейти, если флаг переноса очищен	if C=0 then PC ← PC+k+1	Нет	1/2
BRSH	k -64≤k≤+63	Перейти, если равно или больше (без знака)	if Rd≥ Rr(C=0) then PC ← PC+k+1	Нет	1/2
BRLO	k -64≤k≤+63	Перейти, если меньше (без знака)	if Rd< Rr(C=1) then PC ← PC+k+1	Нет	1/2
BRMI	k -64≤k≤+63	Перейти, если минус	if N=1 then PC ← PC+k+1	Нет	1/2
BRPL	k -64≤k≤+63	Перейти, если плюс	if N=0 then PC ← PC+k+1	Нет	1/2
BRGE	k -64≤k≤+63	Перейти, если больше или равно (с учетом знака)	if Rd≥ Rr(N⊕ V=0) then PC ← PC+k+1	Нет	1/2
BRLT	k -64≤k≤+63	Перейти, если меньше чем (со знаком)	if Rd< Rr(N⊕ V=1) then PC ← PC+k+1	Нет	1/2
BRHS	k -64≤k≤+63	Перейти, если флаг полупереноса установлен	if H=1 then PC ← PC+k+1	Нет	1/2
BRHC	k -64≤k≤+63	Перейти, если флаг полупереноса очищен	if H=0 then PC ← PC+k+1	Нет	1/2
BRTS	k -64≤k≤+63	Перейти, если флаг T установлен	if T=1 then PC ← PC+k+1	Нет	1/2
BRTC	k -64≤k≤+63	Перейти, если флаг T очищен	if T=0 then PC ← PC+k+1	Нет	1/2
BRVS	k -64≤k≤+63	Перейти, если флаг переполнения установлен	if V=1 then PC ← PC+k+1	Нет	1/2
BRVC	k -64≤k≤+63	Перейти, если флаг переполнения очищен	if V=0 then PC ← PC+k+1	Нет	1/2
BRIE	k -64≤k≤+63	Перейти, если глобальное прерывание разрешено	if I=1 then PC ← PC+k+1	Нет	1/2
BRID	k -64≤k≤+63	Перейти, если глобальное прерывание запрещено	if I=0 then PC ← PC+k+1	Нет	1/2

Таблица П4

Арифметико-логические команды

Мнемоника	Операнды	Описание	Операция	Флаги	Кол-во циклов
ADD	Rd,Rr 0≤d≤31 0≤r≤31	Сложить без переноса	Rd ← Rd + Rr	Z, C, N, V, H	1
ADC	Rd,Rr 0≤d≤31 0≤r≤31	Сложить с переносом	Rd ← Rd + Rr + C	Z, C, N, V, H	1
ADIW	Rd,K d ∈ {24,26,28,30} 0≤K≤63	Сложить непосредственное значение со словом	Rdh:Rdl ← Rdh:Rdl + K	Z, C, N, V	2
SUB	Rd,Rr 0≤d≤31 0≤r≤31	Вычесть без заема	Rd ← Rd - Rr	Z, C, N, V, H	1
SUBI	Rd,K 16≤d≤31 0≤K≤255	Вычесть непосредственное значение	Rd ← Rd – K	Z, C, N, V, H	1
SBC	Rd,Rr 0≤d≤31 0≤r≤31	Вычесть с заемом	Rd ← Rd – Rr – C	Z, C, N, V, H	1
SBCI	Rd,K 16≤d≤31 0≤K≤255	Вычесть непосредственное значение с заемом	Rd ← K – Rr – C	Z, C, N, V, H	1
SBIW	Rd,K d ∈ {24,26,28,30} 0≤K≤63	Вычесть непосредственное значение из слова	Rdh:Rdl ← Rdh:Rdl – K	Z, C, N, V	2
AND	Rd,Rr 0≤d≤31 0≤r≤31	Выполнить логическое AND	Rd ← Rd * Rr	Z, N, V	1
ANDI	Rd,K 16≤d≤31 0≤K≤255	Выполнить логическое AND	Rd ← Rd * K	Z, N, V	1
OR	Rd,Rr 0≤d≤31 0≤r≤31	Выполнить логическое OR	Rd ← Rd v Rr	Z, N, V	1
ORI	Rd,K 16≤d≤31 0≤K≤255	Выполнить логическое OR с непосредственным значением	Rd ← Rd v K	Z, N, V	1
EOR	Rd,Rr 0≤d≤31 0≤r≤31	Выполнить исключающее OR	Rd ← Rd ⊕ Rr	Z, N, V	1
COM	Rd 0≤d≤31	Выполнить дополнение до единицы	Rd ← $FF – Rd	Z, C, N, V	1

Окончание таблицы П4

Мнемоника	Операнды	Описание	Операция	Флаги	Кол-во циклов
NEG	Rd 0≤d≤31	Выполнить дополнение до двух	Rd ← $00 – Rd	Z, C, N, V, H	1
SBR	Rd,K 16≤d≤31 0≤K≤255	Установить биты в регистре	Rd ← Rd v K	Z, N, V	1
CBR	Rd,K 16≤d≤31 0≤K≤255	Очистить биты в регистре	Rd ← Rd * ($FF-K)	Z, N, V	1
INC	Rd 0≤d≤31	Инкрементировать	Rd ← Rd + 1	Z, N, V	1
DEC	Rd 0≤d≤31	Декрементировать	Rd ← Rd – 1	Z, N, V	1
TST	Rd 0≤d≤31	Проверить на ноль или минус	Rd ← Rd * Rd	Z, N, V	1
CLR	Rd 0≤d≤31	Очистить регистр	Rd ← Rd ⊕ Rd	Z, N, V	1
SER	Rd 0≤d≤31	Установить все биты регистра	Rd ← $FF	Нет	1
CP	Rd,Rr 0≤d≤31 0≤r≤31	Сравнить	Rd – Rr	Z, C, N, V, H	1
CPC	Rd,Rr 0≤d≤31 0≤r≤31	Сравнить с учетом переноса	Rd – Rr – C	Z, C, N, V, H	1
CPI	Rd,K 16≤d≤31 0≤K≤255 0≤b≤7	Сравнить с константой	Rd – K	Z, C, N, V, H	1

Таблица П5

Команды сдвигов и операций с битами

Мнемоника	Операнды	Описание	Операция	Флаги	Кол-во циклов
LSL	Rd 0≤d≤31	Логически сдвинуть влево	Rd(n+1)← Rd(n) Rd(0) ← 0, C ← Rd (7)	Z,C,N, V, H	1
LSR	Rd 0≤d≤31	Логически сдвинуть вправо	Rd(n)← Rd(n+1) Rd(7) ← 0, C ← Rd (0)	Z,C,N, V	1
ROL	Rd 0≤d≤31	Сдвинуть влево через перенос	Rd(0)← C Rd(n+1)← Rd(n) C ← Rd(7)	Z,C,N, V, H	1
ROR	Rd 0≤d≤31	Сдвинуть вправо через перенос	Rd(7) → C Rd(n) ← Rd(n+1) C ← Rd(0)	Z,C,N, V	1
ASR	Rd 0≤d≤31	Арифметически сдвинуть вправо	Rd(n) ← Rd(n+1) C ← Rd(0) n=0-6	Z,C,N, V	1
SWAP	Rd 0≤d≤31	Поменять тетрады местами	Rd(3..0) <-> Rd(7..4)	нет	1
BSET	s 0≤s≤7	Установить флаг	SREG(s) ← 1	SREG(s)	1
BCLR	s 0≤s≤7	Очистить флаг	SREG(s) ← 0	SREG(s)	1
SBI	P, b 0≤P≤31 0≤b≤7	Установить бит в регистре I/O	I/O (P,b) ← 1	нет	2
CBI	P, b 0≤P≤31 0≤b≤7	Очистить бит в регистре I/O	I/O (P,b) ← 0	нет	2
BST	Rd, b 0≤d≤31 0≤b≤7	Переписать бит из регистра во флаг T	T ← Rd(b)	T	1
BLD	Rd, b 0≤d≤31 0≤b≤7	Загрузить флаг T в бит регистра	Rd(b) ← T	Нет	1
SEC		Установить флаг переноса	C ← 1	C	1
CLC		Очистить флаг переноса	C ← 0	C	1
SEN		Установить флаг отрицательного значения	N ← 1	N	1
CLN		Очистить флаг отрицательного значения	N ← 0	N	1
SEZ		Установить флаг нулевого значения	Z ← 1	Z	1

Окончание таблицы П5

Мнемоника	Операнды	Описание	Операция	Флаги	Кол-во циклов
CLZ		Очистить флаг нулевого значения	Z ← 0	Z	1
SEI		Установить флаг глобального прерывания	I ← 1	I	1
CLI		Очистить флаг глобального прерывания	I ← 0	I	1
SES		Установить флаг знака	S ← 1	S	1
CLS		Очистить флаг знака	S ← 0	S	1
SEV		Установить флаг переполнения	V ← 1	V	1
CLV		Очистить флаг переполнения	V ← 0	V	1
SET		Установить флаг T	T ← 1	T	1
CLT		Очистить флаг T	T ← 0	T	1
SEH		Установить флаг полупереноса	H ← 1	H	1
CLH		Очистить флаг полупереноса	H ← 0	H	1
NOP		Выполнить холостую команду		Нет	1
SLEEP		Установить режим SLEEP	См. описание команды	Нет	1
WDR		Сбросить сторожевой таймер	См. описание команды	Нет	1

ЛАБОРАТОРНАЯ РАБОТА № 2

Изучение отладочной платы EasyAVR6

Цель работы: изучение устройства и возможностей отладочной платы EasyAVR6, получение практических навыков программирования процедур опроса состояния внешних устройств и обмена данными с ними.

1. Описание отладочной платы EasyAVR6

Отладочная плата предназначена для исследования режимов работы программируемых устройств, встроенных в AVR-контроллеры семейства ATmega, разработки программ, обеспечивающих взаимодействие микроконтроллеров (МК) с системами отображения информации и исполнительными устройствами, организации каналов связи для передачи данных последовательным и параллельным кодами в разрабатываемой микропроцессорной системе. Программы создаются в интегрированной среде разработки AVR Studio и загружаются во внутреннюю память программ микроконтроллера через USB интерфейс персонального компьютера (ПК).

Установленный на отладочной плате набор панелей с холодным контактом (сокетов) для DIP-корпусов позволяет производить отладку программного обеспечения для большого числа микроконтроллеров семейств ATmega и ATtiny. На плате размещено 8 панелей, что позволяет программировать AVR-контроллеры с 40, 28, 20, 14, 8 выводами в DIP-корпусах, причем по 2 панели предусмотрены для корпусов DIP40, DIP20, DIP8. На обратной стороне платы указаны все типы программируемых микроконтроллеров, которые могут быть установлены в панели с равным числом выводов, имеющих для каждой из панелей определенное функциональное назначение. В лабораторных работах исследуются особенности работы встроенных программируемых устройств микроконтроллеров ATmega16/32x и ATmega 8, проектирования

микропроцессорных систем на базе контроллеров этого типа. Поэтому перед началом работы со стендом необходимо убедиться в наличии соответствующих микроконтроллеров в панели DIP40B или DIP28B.

Схема размещения элементов на отладочной плате EASY AVR6 показана на Рисунок 1.10.

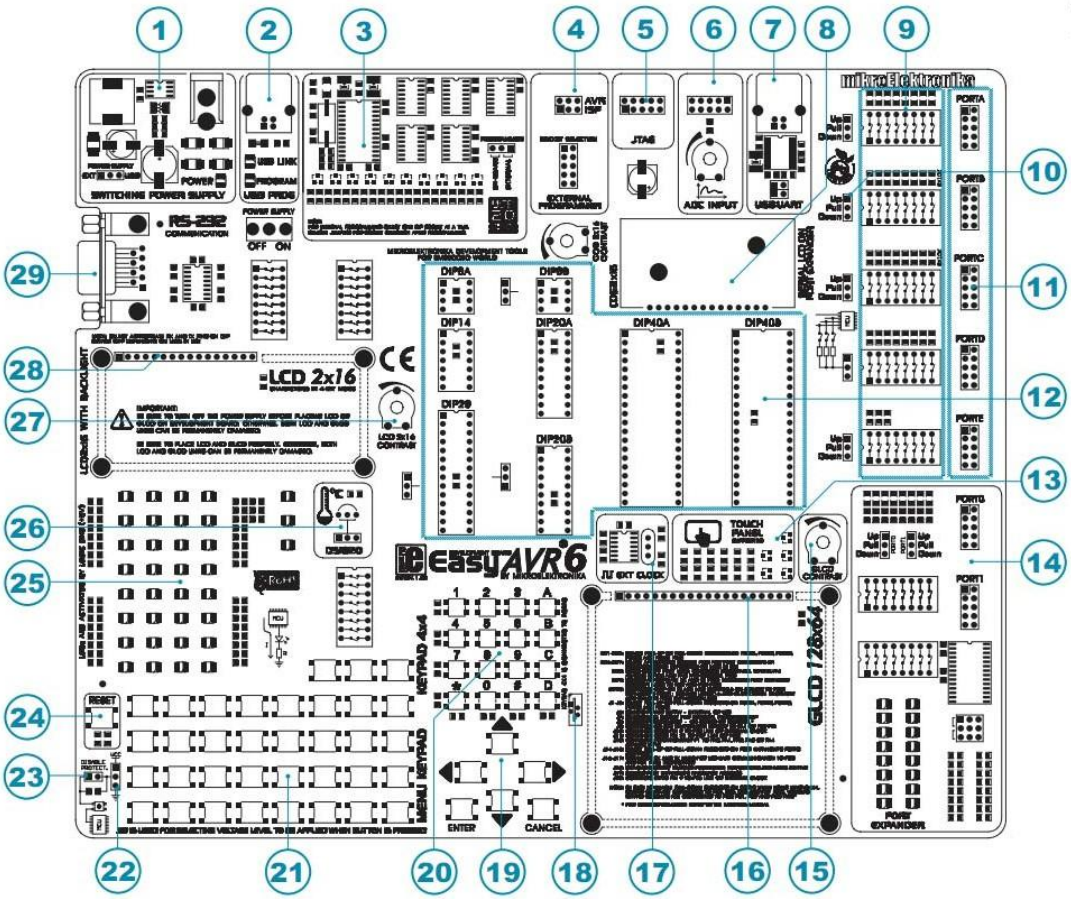


Рисунок 1.10. Схема размещения элементов на отладочной плате EasyAVR6

На Рисунок 1.10 цифрами обозначены наиболее важные коммутационные элементы платы, имеющее следующее назначение.

1. Схема блока питания с разъемом AC/DC для подключения внешнего источника напряжения переменного или постоянного тока.
2. Разъем для подключения встроенного USB-программатора.
3. Встроенный программатор.
4. Штыревой соединитель для дополнительного программатора.

5. Штыревой соединитель JTAG-интерфейса.

6. Входы АЦП.

7. Модуль моста USB-UART с разъемом USB B.

8. Встроенный жидкокристаллический индикатор LCD 2x16.

9. Переключатели SW1-SW5 (8-ми позиционные) для включения или отключения подтягивающих резисторов по отдельным линиям портов.

10. Переключатели J1-J5, соединяющие отдельные группы подтягивающих резисторов портов к выводам источника питания +5В или GND.

11. Штыревые разъемы входов/выходов портов микроконтроллеров.

12. Панели для установки AVR-контроллеров.

13. Контроллер сенсорного экрана.

14. Схема расширителя портов.

15. Потенциометр для регулировки яркости графического индикатора (GLCD).

16. Разъем для подключения GLCD.

17. Генератор тактовых импульсов.

18. Разъем для подключения внешней сенсорной панели.

19. Клавиатура функциональных кнопок «Меню».

20. Буквенно-цифровая клавиатура 4x4.

21. Кнопки для моделирования цифровых сигналов.

22. Переключатель уровня логического 0 или 1 при нажатии кнопок моделирования цифровых сигналов.

23. Перемычка для подключения/отключения защитного резистора к кнопкам моделирования цифровых сигналов.

24. Кнопка сброса.

25. Светодиоды - индикаторы состояния выводов МК.

26. Разъем для подключения датчика температуры DS1820.

27. Регулятор подсветки жидкокристаллического индикатора.

28. Соединитель для подключения внешнего двустрочного знакосинтезирующего жидкокристаллического индикатора.

29. Разъем DB-9 интерфейса RS-232.

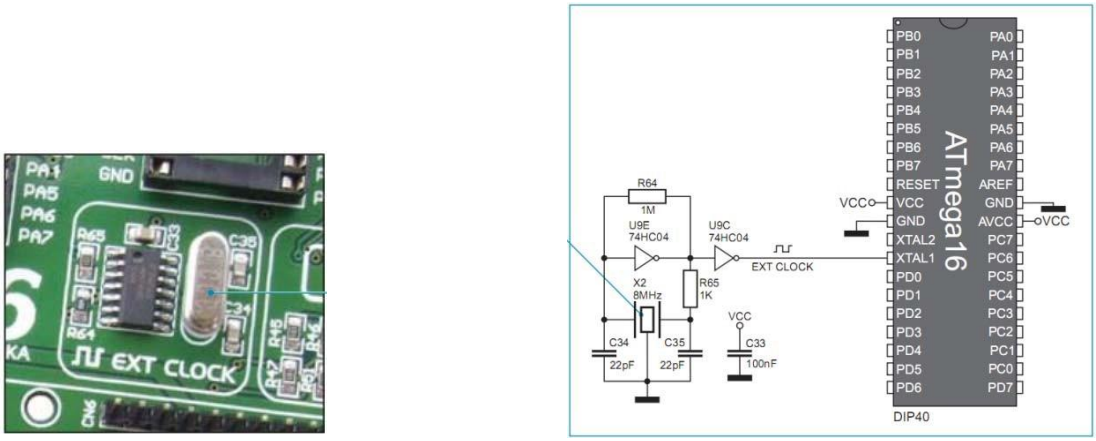
Назначение отдельных переключателей на плате. В AVR-контроллерах для формирования системных тактовых импульсов могут использоваться разные источники сигналов. Во всех контроллерах встроенный генератор может работать с внутренним колебательным контуром на RC-цепочке или внешним кварцевым резонатором, а в отдельных случаях - с внешним колебательным контуром на RC-цепи. Кроме того, тактовая частота может прямо подаваться от внешнего генератора по входу XTAL1 или CLK, в зависимости от типа микроконтроллера. На плате предусмотрена возможность выбора или внутреннего источника сигналов или внешнего генератора. Подключение внешнего генератора осуществляется с помощью переключателей J10, J20, расположенных рядом с сокетами DIP28 и DIP8. В табл. 6 показано, что этими переключателями соответствующий вывод МК может быть связан с выходом внешнего генератора или быть выведен как стандартный вход/выход линии порта к внешнему разъему на отладочной плате. Переключатель J11 устанавливает функцию вывода 21 микроконтроллера ATmega16.

Таблица 6

Назначение переключателей

Джампер	Позиция	Функция
J10	PB3	PB3 – стандартный вх/вых МК ATtiny
	CLK	На линию PB3 подается сигнал clock от внешнего генератора
J11	V_{cc}	Вывод AREF встроенного АЦП микроконтроллера ATmega16 соединяется с V_{cc}
	PC7	PC7 – стандартный вх/вых МК
J20	CLK	На линию PB6 подается сигнал clock от внешнего генератора
	PB6	PB6 – стандартный вх/вых МК ATmega8

На Рисунок 1.11 приведена схема внешнего генератора тактовых импульсов, построенная на логических элементах и кварцевом резонаторе. Кварцевый резонатор устанавливается на панели, как сменный элемент, и может выбираться в диапазоне, ограниченном максимальной частотой для программируемого микроконтроллера.



а) б)

Рисунок 1.11. а) внешний генератор, б) схема соединения генератора с МК

Встроенный USB 2.0 программатор. Программатор используется для прошивки ПЗУ микроконтроллеров. Встроенный программатор на отладочной плате EasyAVR6 может быть связан с персональным компьютером через USB-порт. В тоже время предусмотрена возможность применения внешнего программатора при установке переключателя J8 в положение external (Рисунок 1.12).

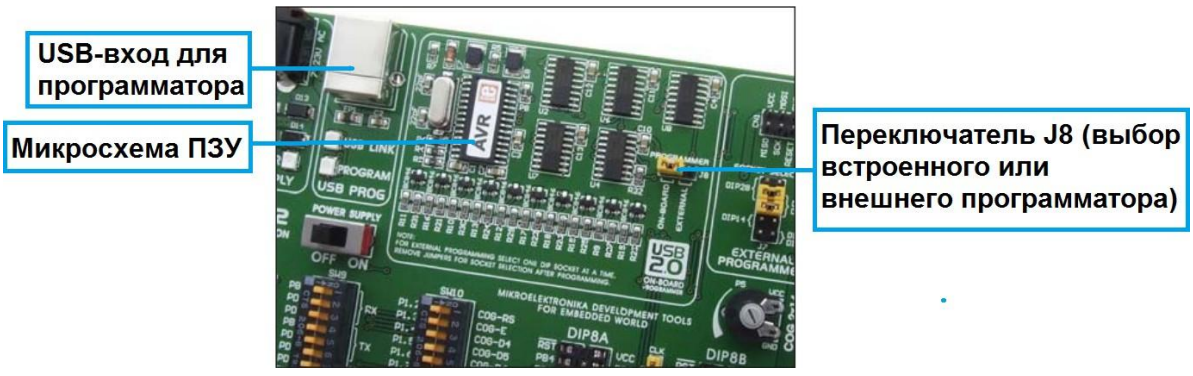


Рисунок 1.12. Внешний вид USB-порта отладочной платы

Блок питания. Для платы отладчика EasyAVR6 может быть выбрана одна из двух схем подключения источника питания.

- Питание +5В от ПК через USB-порт.
- Внешний источник питания, подключенный к разъему AC/DC.

При внешнем источнике питания стабилизатор напряжения – микросхема MC34063A со схемой выпрямителя используются для включения источника переменного тока (AC) в диапазоне от 7В до 23В. Предусмотрено и подключение источника постоянного тока (DC) в диапазоне от 9В до 32В.

Схема подключения источника питания показана на Рисунок 1.13.

Переключатель J6 выполняет функцию селектора при выборе источника питания. Когда источником питания является USB-порт, селектор должен быть установлен в положение USB, а при использовании внешнего источника питания – в положение EXT. В лабораторных работах будет использоваться питание от интерфейса USB персонального компьютера (переключатель J6 в положении USB).

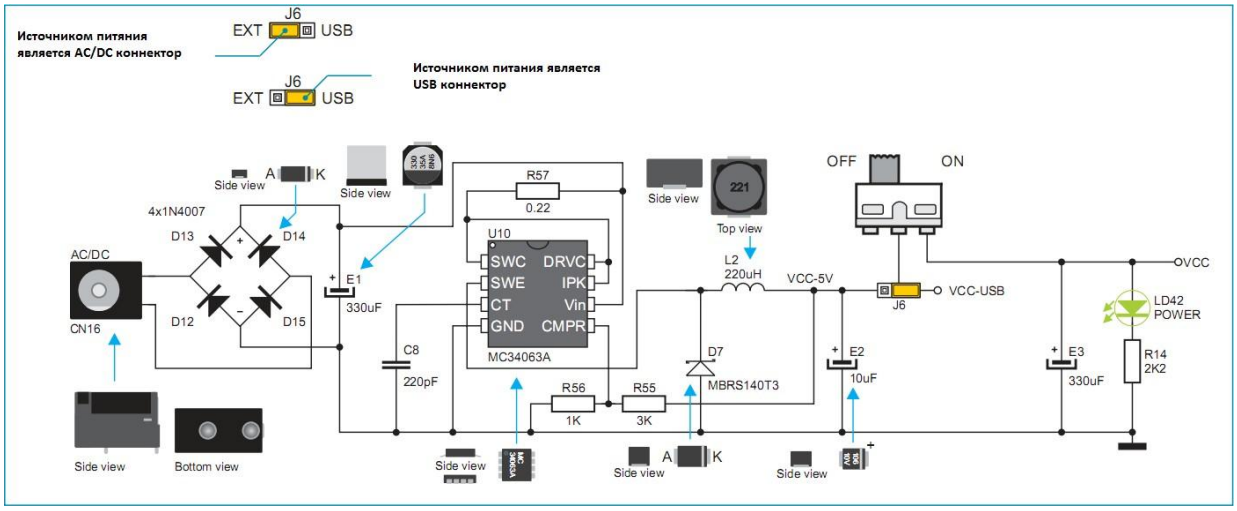


Рисунок 1.13. Схема подключения источника питания.

Использование светодиодов и кнопок стенда. Каждый вывод портов микроконтроллера подключен на стенде к одной из кнопок (T1 - T33) и к одному из светодиодов (LD1 – LD35) (Рисунок 1.14). Светодиоды с индивидуальными токоограничительными резисторами сгруппированы по портам и каждая из таких групп светодиодов может быть отключена от

микроконтроллера с помощью микропереключателя SW8: SW8.1 – PORTA и PORTE, SW8.2 – PORTB, SW8.3 – PORTC, SW8.4 – PORTD. Все светодиоды подключены катодом к общей линии питания (GND). Таким образом, светодиод будет гореть при подаче лог. 1 на соответствующий вывод микроконтроллера.

Все кнопки T1-T33, как простой двухполюсник, работающий на замыкание, имеют один общий вывод, потенциал на котором можно задать с помощью переключателя J13: в положении 1-2 — Vcc (+5В), в положении 3-4 — GND (0В). Таким образом, можно задать логический уровень на выводе микроконтроллера при нажатой кнопке. При использовании кнопок рекомендуется, чтобы переключатель J18 был разомкнут, т.к. при этом последовательно с общим проводом кнопок включен ограничивающий резистор (220 Ом), который защищает вывод микроконтроллера от короткого замыкания в выходной цепи при неправильной конфигурации порта. Например, вывод PA0 настроен на выход в лог. 1, а кнопка T1 подключена к общему проводу. Для задания необходимого логического уровня на выводе микроконтроллера при отжатой кнопке можно использовать внутренние подтягивающие резисторы AVR-контроллера. При этом подтяжка уровня выходного напряжения обеспечивается только к лог. 1. Вместе с тем имеется возможность использовать группы микропереключателей SW1-SW5, которые вместе с перемычками J1-J5 позволяют задать подтяжку индивидуально по каждому выводу, как к лог. 1, так и к лог. 0.

Описание клавиатуры. На плате размещены две клавиатуры: клавиатура ввода данных 4x4 и функциональная клавиатура «Меню» для задания определенных операций.

Набор клавиш 4x4 - это стандартная буквенно-цифровая клавиатура, соединенная с портом PC микроконтроллера. Уровни по умолчанию (при отжатых кнопках в ряду) могут быть установлены с помощью подтягивающих резисторов порта C - микропереключатели SW3 с перемычкой J3 (Рисунок 1.15).

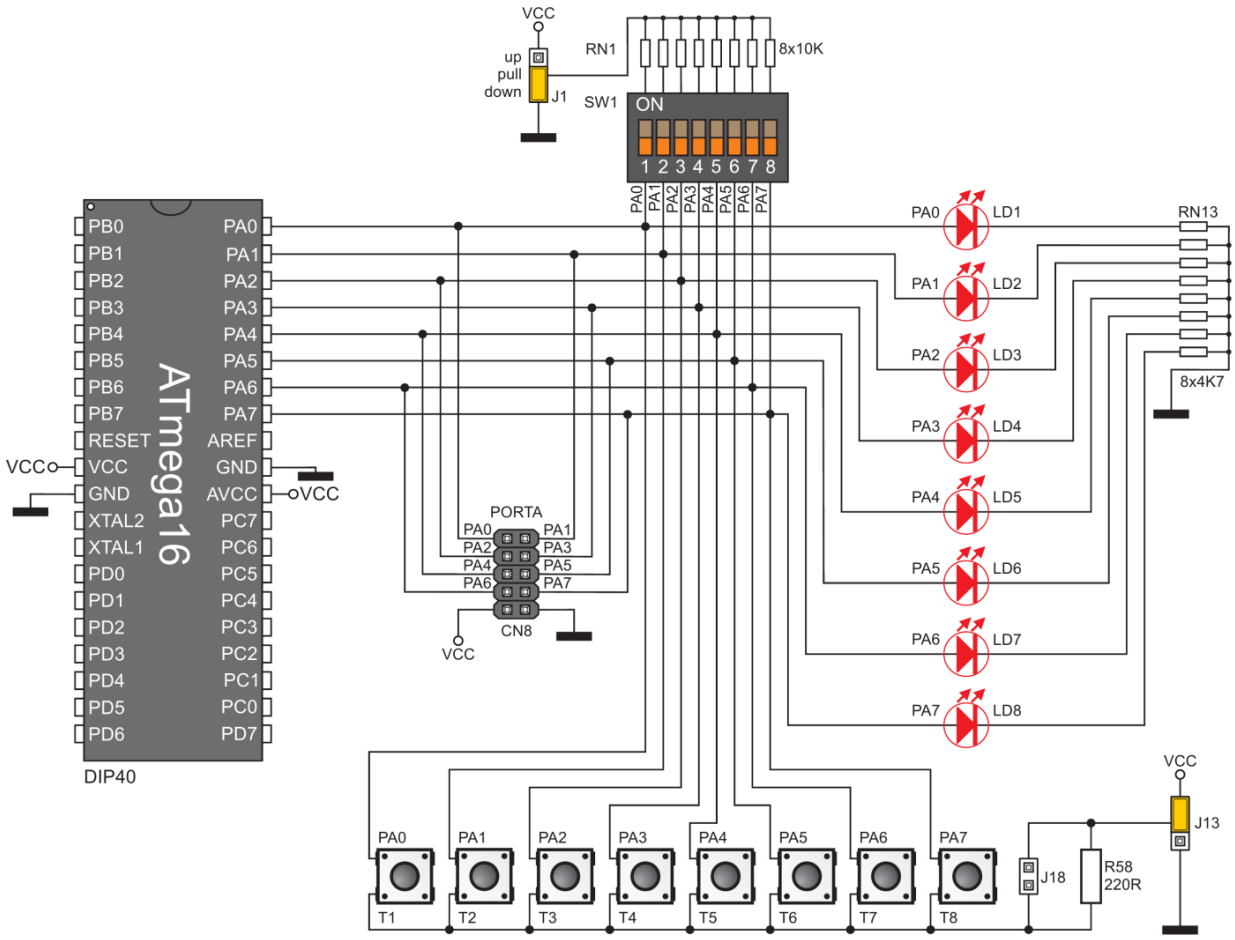


Рисунок 1.14. Схема включения светодиодов и элементов коммутации

Для контроля состояния кнопки в строке необходимо последовательно устанавливать низкий потенциал (логический 0) на выводы микроконтроллера PC4, PC5, PC6, PC7 (строки клавиатуры). В этом случае на одном из выводов PC0, PC1, PC2 или PC3 (столбцы клавиатуры), подключенном к нажатой кнопке, установится уровень логический 0. Для определения текущего состояния клавиатуры микроконтроллер должен периодически считывать состояния выводов PC0 - PC3. Эта процедура опроса клавиатуры определяет состояние нажатой кнопки. В реальных системах опрос клавиатуры производится периодически по сигналам таймера или осуществляется по запросу прерывания, формируемому клавиатурой. Основной проблемой при обслуживании клавиатуры является защита от дребезга контактов, который может привести к формированию ложной информации. Для защиты применяются как аппаратные, так и программные

средства. На плате EasyAVR6 аппаратных средств защиты от дребезга контактов не предусмотрено, поэтому должны применяться только программные методы, все варианты которых сводятся, в основном, к многократному чтению состояния кнопок клавиатуры.

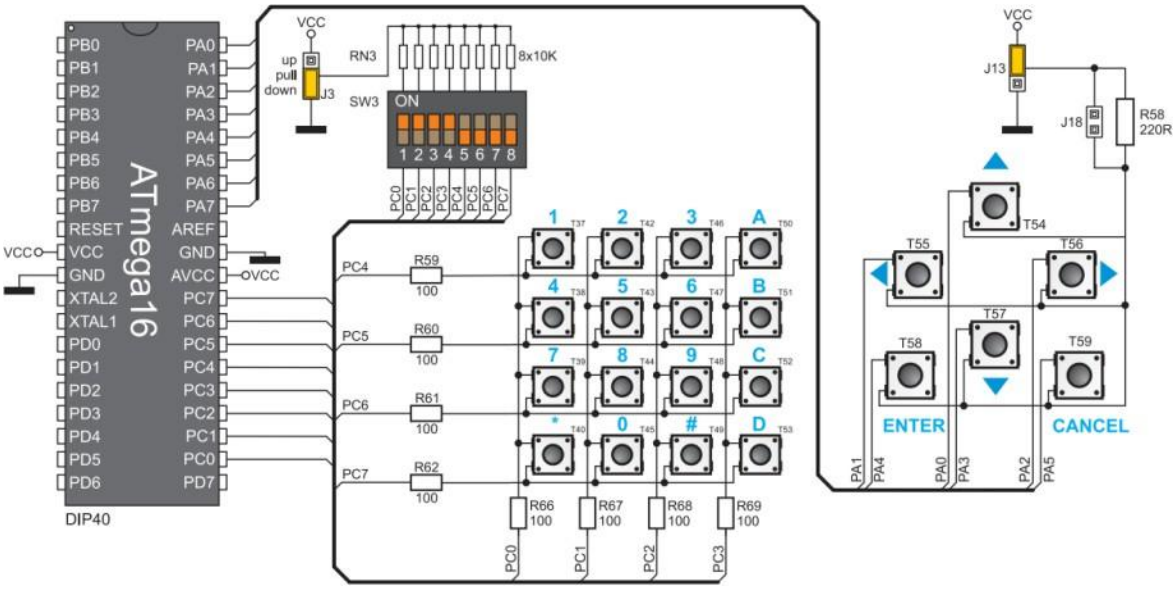


Рисунок 1.15. Схема подключения буквенно-цифровой и функциональнойклавиатур

Аналогичным способом организована функциональная клавиатура (клавиатура «Меню»). Кнопки T54-T59 (Рисунок 1.15) по линиям столбцов связаны со входами порта A (PA0-PA5), а потенциал линий строк (лог. 1 или лог. 0) задается принудительно с помощью переключателей J13 и J18 от источника питания.

2. Порты ввода/вывода

Микроконтроллеры ATmega16 имеют 4 параллельных порта ввода/вывода PA, PB, PC и PD. Все порты могут работать, как в режиме двунаправленного ввода/вывода, так и выполнять альтернативную функцию.

Рассмотрим работу порта в режиме 8-разрядного двунаправленного ввода/вывода. Программное управление портом осуществляется через три

расположенных в пространстве ввода/вывода (памяти данных) регистра: регистр данных - PORTx, регистр направления данных - DDRx и регистр входных данных — PINx. Регистр входных данных PINx обеспечивает только возможность чтения, регистры данных и направления данных обеспечивают возможность и чтения и записи. Регистр DDRx служит для настройки направления передачи данных – логический 0 в разрядах этого регистра означает, что соответствующие линии сконфигурированы на вход, логическая 1 – на выход. Значение, хранящееся в регистре PORTx, если порт сконфигурирован на выход, выставляется на выводах микроконтроллера, а если порт сконфигурирован на вход, это значение определяет, подключен ли внутренний подтягивающий резистор (логическая 1 в соответствующей линии – резистор подключен). Регистр PINx служит для чтения состояния порта.

Все выводы порта оснащены индивидуально подключаемыми встроенными нагрузочными резисторами. Выходные буферы порта обеспечивают ток до 20мА, что достаточно для прямого управления светодиодными индикаторами. Если выводы Px.0 – Px.7 используются в качестве входов и внешним сигналом удерживаются на низком уровне, то вытекающий ток обеспечивается подключением внутренних нагрузочных резисторов. После сброса выводы портов находятся в третьем состоянии.

Таблица 7

Воздействие битов DDRxn на работу выводов порта "x"

DDR	PORT	I/O	Нагрузочный резистор	Описание
0	0	Вход	Не подключен	Третье состояние
0	1	Вход	Подключен	При входном низком уровне Pxn обеспечивает вытекающий ток
1	0	Выход	Не подключен	Низкий уровень, двухтактный выход
1	1	Выход	Не подключен	Высокий уровень, двухтактный выход

3. Использование отладочной платы EasyAVR6

После того как программа отлажена в интегрированной среде разработки Atmel Studio можно приступать к работе с отладочной платой. Отладочная плата подключается к персональному компьютеру с помощью USB-кабеля. Кабелем соединяются USB-порт ПК с разъемом CN2 (на плате помечен как USB PROG). Для подачи питания необходимо выключатель POWER SUPPLY перевести в положение ON. После этого должны зажечься светодиоды POWER и USB LINK. Если светодиод USB LINK не горит, то необходимо установить драйвер для программатора на стенде.

Для переноса написанной программы на отладочную плату необходимо воспользоваться специальной утилитой AVRFLASH, поставляемой вместе с платой EasyAVR6 и позволяющей работать с его встроенным программатором. Средства Atmel Studio не позволяют работать со стендом напрямую. Прежде всего, необходимо получить Release сборку вашей программы (Рисунок 1.16). Для этого воспользуйтесь переключателем на панели инструментов и выполните сборку проекта.

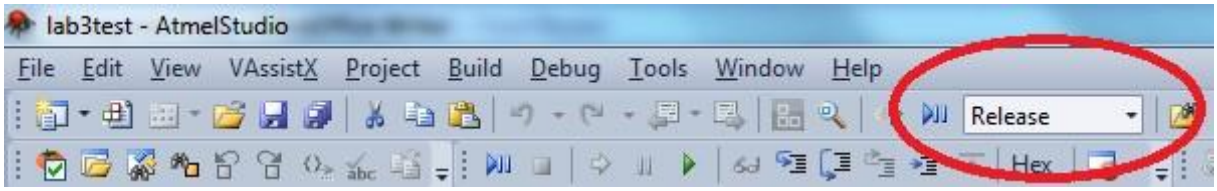


Рисунок 1.16. Выбор типа сборки «Release» в Atmel Studio

В результате в папке вашего проекта появится папка «Release», содержащая файл с расширением *.hex. Это готовый для прошивки файл, содержащий машинный код вашей программы. Далее необходимо запустить программу AVRFLASH. Через меню «File/Load hex» открыть полученный *.hex файл для вашей программы. Убедитесь что стенд подключен к компьютеру, его питание включено и все настройки прошивки соответствуют необходимым, как это показано на рисунок 1.17.

Будьте внимательны и осторожны! Неверно выполненная «прошивка» может привести к неработоспособности вашей программы и даже к порче установленного на стенде микроконтроллера. Как только вы убедились в том, что все верно, можно нажать кнопку «Write», что запустит процесс «прошивки». После его завершения вы сразу сможете наблюдать результат выполнения вашей программы на стенде. При внесении изменений в исходный код и повторной сборке необходимо также повторно загрузить полученный *.hex файл через меню «File/Load hex», а уже после этого выполнять прошивку.

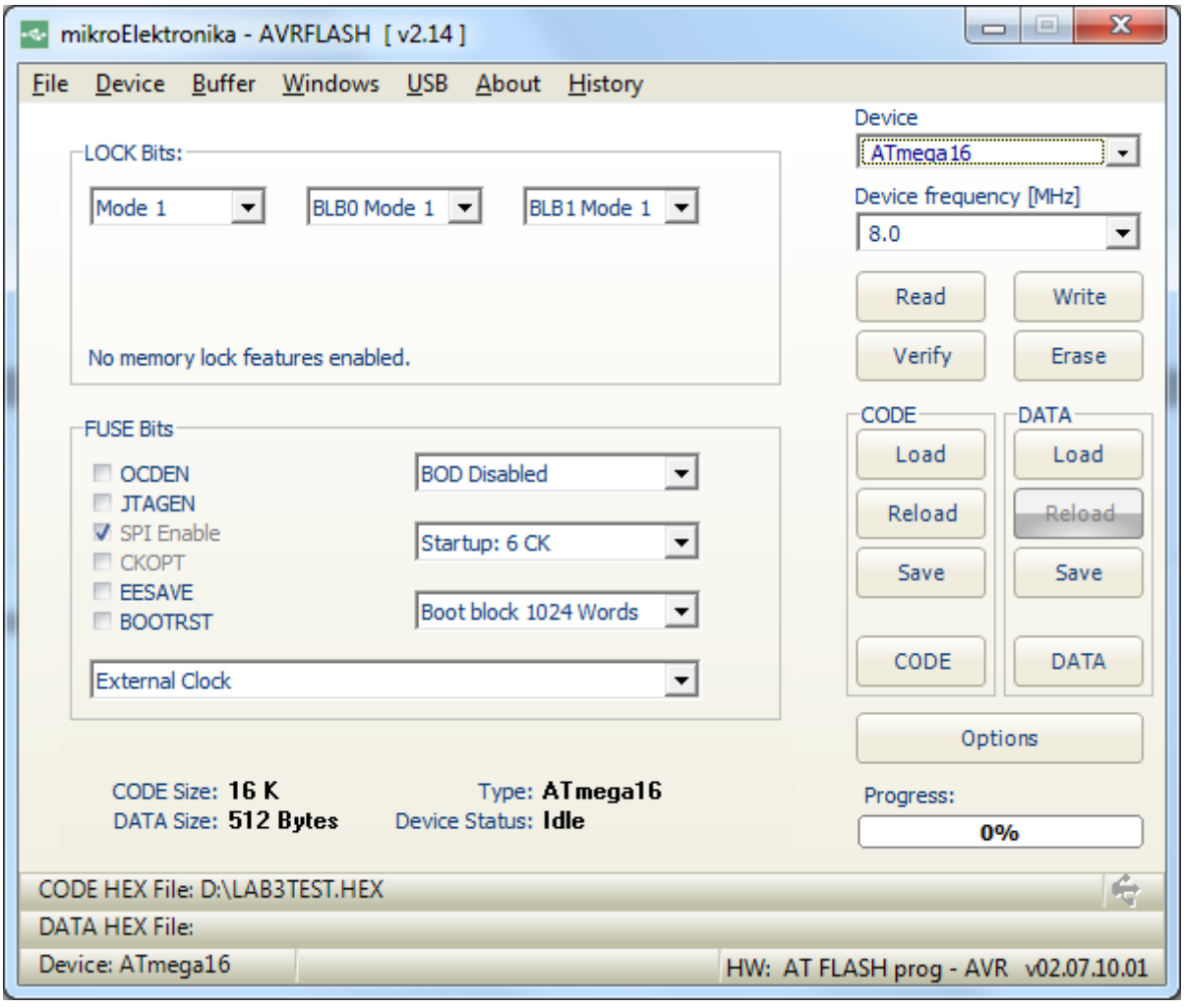


Рисунок 1.17. Настройки прошивки для ATmega16

4. Общее задание

Рассмотрите программу, исходный код которой приведен ниже. Выполните ее компиляцию и отладку в Atmel Studio, а затем прошивку на отладочную плату. Убедитесь в том, что она работает.

Листинг 2. Программа, определяющая нажатия кнопок на 16-кнопочной клавиатуре

```
.include "m16def.inc"      ;ATmega16

.Equ Stack=$03FF
.Equ KB_PORT=PORTC         ;Порт клавиатуры
.Equ KB_DDR=DDRC
.Equ KB_PIN=PINC
.Equ LED_PORT=PORTA        ;Порт для вывода номера нажатой клавиши
.Equ LED_DDR=DDRA

.Def Acc=R16
.Def Bcc=R17
.Def Ccc=R18
.Def KeyFlags=R19          ;Сдвиговый регист клавиатуры

;**************************** таблица прерываний ****************************
.ORG $0
       rjmp Start          ;Вектор прерывания события RESET
.ORG $50                   ;Адрес исполняемой части

;************************ основной цикл программы ************************
Start:
       ldi  R16,  high(Stack) ;Инициализация  стека
       out SPH, R16
       ldi R16, low(Stack)
       out SPL, R16
       ldi Acc, $FF
       out LED_DDR, Acc
       ldi Acc, $0F
       out KB_PORT, Acc
       clr Acc
       out KB_DDR, Acc
       ldi KeyFlags, 0

MainCycle:
       rcall AskKey        ;Опрос KB
       brtc MainCycle      ;Если флаг T очищен
       rcall ShowKeyNumber ;Вывод номера последней нажатой клавиши в порт
       rjmp MainCycle

AskKey:
       clr Bcc
       clt                 ;Очищаем флаг T
       sbi KB_DDR, 4       ;Ищем нажатую клавишу в 0-ом ряду
       rcall Check         ;Проверка на нажатие в 0-ом ряду
       cbi KB_DDR, 4       ;Восстанавливаем PORT
```

```
        brcc Dreb           ;Если было нажатие переводим на Dreb
        sbi KB_DDR, 5
        rcall Check
        cbi KB_DDR, 5
        brcc Dreb
        sbi KB_DDR, 6
        rcall Check
        cbi KB_DDR, 6
        brcc Dreb
        sbi KB_DDR, 7
        rcall Check
        cbi KB_DDR, 7

Dreb:                       ;Проверка на дребезг
        lsl KeyFlags        ;Сдвигаем сдвиговый регистр клавиатуры
        bld KeyFlags, 0     ;Записываем в 0-ой бит значение флага T
        cpi KeyFlags, $FF   ;Если точно было нажатие
        breq Check02        ;то на выход
        clt                 ;Если не было нажатия то очищаем флаг T и на выход
        ret

Check:
        in Acc, KB_PIN      ;Читаем состояние выводов порта клавиатуры
        ldi Ccc, 5          ;Устанавливаем счетчик цикла
Check01:
        dec Ccc             ;Уменьшаем счетчик цикла
        breq Check02        ;Если прошли 4 колонки, то на выход (нажатия нет)
        inc Bcc             ;Номер кнопки
        lsr Acc             ;Поиск нажатой клавиши
        brcs Check01        ;Если флаг C=0 было нажатие и мы
        set                 ;устанавливаем флаг T
Check02:
        ret

ShowKeyNumber:
        out LED_PORT, Bcc   ;вывод номера нажатой кнопки на светодиоды
        ret
```

Вышеприведенная программа определяет факт нажатия одной из клавиш 16-кнопочной клавиатуры, организованной в виде матрицы 4x4, заносит номер нажатой клавиши в регистр R17, которому в начале программы присвоен псевдоним Bcc. Номер нажатой клавиши в двоичной кодировке отображается на восьми светодиодных индикаторах через порт A. Подразумевается, что в каждый момент времени может быть нажата только одна клавиша. В программе предусмотрена защита от дребезга контактов. Факт нажатия клавиши фиксируется установкой флага T.

В начале программы с помощью директив EQU и DEF регистрам общего назначения и регистрам портов присваиваются псевдонимы, удобные для использования в контексте данной программы.

Директива ORG $0 размещает по нулевому адресу памяти программ команду rjmp Start, которая выполняет безусловный переход на основной код программы, инициализирующий микроконтроллер (инициализация стека, портов). Этот код начинается с адреса $50 (определяется директивой ORG $50).

В основном цикле программы осуществляется только вызов процедур. Это делает программу более понятной и обеспечивает ее структурность. В данном случае вызываются две процедуры: процедура опроса клавиатуры и процедура вывода кода нажатой клавиши на светодиоды через порт A. Последняя вызывается только тогда, когда имело место нажатие клавиши. Признаком нажатия служит флаг T – флаг общего назначения в регистре SREG.

Процедура AskKey использует так называемый «метод скользящего нуля» для поиска нажатой клавиши. Работает он следующим образом. Имеется клавиатура размерностью 4x4. Каждая колонка «подтянута» резистором к шине питания достаточно большим сопротивлением. Таким образом, на линиях колонок присутствует уровень логической единицы. Линии колонок присоединены к микроконтроллеру. Режим работы этих выводов микроконтроллера – вход. Линии рядов также присоединены к порту микроконтроллера и выводы порта сконфигурированы как входные. Нажатие какой-либо из клавиш замыкает определенную линию колонок и линию рядов. Таким образом, логические уровни замкнутых линий будут равны. При условии, что все выводы клавиатурного порта микроконтроллера работают на вход (режим-вход), замкнутые линии будут иметь уровень логической единицы (так как колонки подтянуты резистором к шине питания, а после нажатия клавиши к линии колонки подключается линия соответствующего ряда). Теперь, если на линию ряда подать уровень

логического нуля, то через резистор пойдет небольшой ток, а напряжение на замкнутых линиях будет соответствовать уровню логического нуля. Если логический ноль теперь поочередно подавать на каждую из линий рядов (т.е. как бы двигать ноль в порту), то нажатие легко отслеживается по наличию низкого уровня на линии колонок.

Неприятным атрибутом любых контактов, в том числе и кнопок клавиатуры, является наличие дребезга, т.е. быстрое и непредсказуемое изменение состояния: замкнуто – незамкнуто. Это происходит при нажатии, отпускании, соединении, разъединении и т.д. Существует множество методов борьбы с дребезгом, один из которых мы разберем подробно.

Будем опрашивать клавиатуру и каждый раз, если клавиша нажата и удерживается, будем вдвигать единичку в статус-байт клавиатуры (этот байт мы создаем сами, обозначим его KeyFlags). Если клавиша не нажата - вдвигаем ноль. Как только значение этого байта станет равным $FF - микроконтроллер фиксирует нажатие клавиши. При этом устанавливается флаг T и из основного цикла вызывается процедура ShowKeyNumber вывода номера нажатой клавиши на светодиоды.

5. Отладка взаимодействия с клавиатурой

Встроенные средства отладчика Atmel Studio (Рисунок 1.18) позволяют выполнить полноценную отладку рассмотренной программы без необходимости использовать отладочную плату. Все что для этого требуется симулировать поведение клавиатуры выставляя соответствующие биты регистра PIN порта PC. Действие кнопки можно симулировать установкой/снятием младших 4-х бит в регистре PINC, а включение/выключение светодиода, соответственно, определяется состоянием бит регистра PORTA.

6. Индивидуальные задания

1. Составить программу, которая анализирует номер нажатой клавиши и вычисляет соответствующий десятичный номер. При отображении выводить младший и старший разряды на две отдельные колонки светодиодов (кодировка клавиш задается преподавателем).

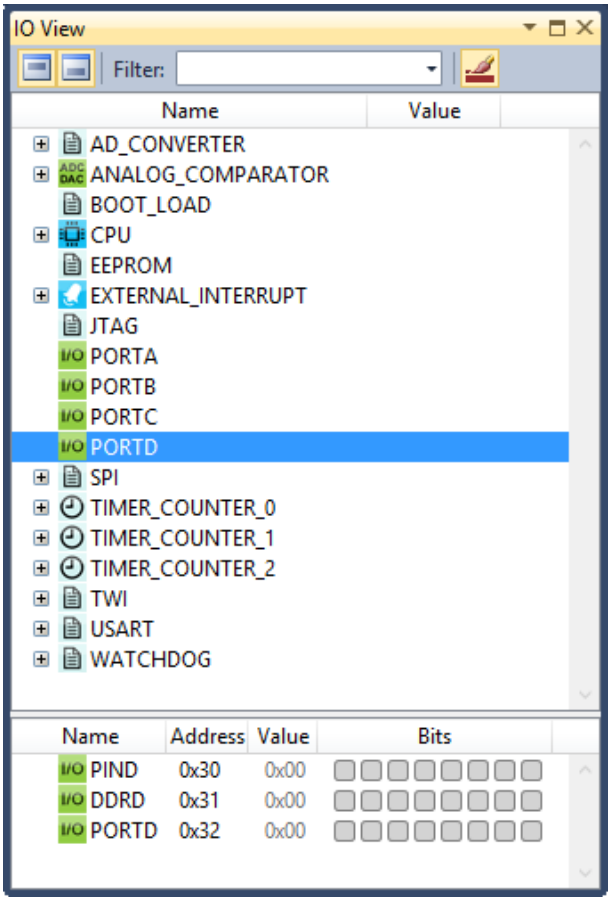


Рисунок 1.18. Окно отладки для портов ввода-вывода

2. Разработать и реализовать различные варианты подавления дребезга контактов клавиатуры.

3. Составить программу опроса клавиатуры, в которой при нажатии каждой клавиши загорается соответствующий ей светодиод, но только в том случае если нажато не более одной клавиши.

4. Составить программу опроса клавиатуры, в которой при нажатии любой одной из клавиш соответствующий ей светодиод зажигается илигаснет в зависимости от того какой из двух режимов работы программы

активен. Переключение активного режима осуществляется нажатием одной из клавиш.

5. Составить программу опроса клавиатуры, в которой каждой клавише соответствует отдельный 8-разрядный счетчик. Значение счетчика наращивается при каждом нажатии клавиши. Значение счетчика должно отображаться с использованием 8 светодиодных индикаторов сразу после нажатия.

6. Составить программу отображения на светодиодных индикаторах количества нажатых одновременно клавиш клавиатуры.

7. Составить программу, которая запоминает номера четырех последних нажатых клавиш и выводит на светодиоды номер клавиши, нажатой четыре клавиши назад.

ЛАБОРАТОРНАЯ РАБОТА № 3

Использование таймеров/счетчиков для формирования временных задержек

Цель работы: изучение особенностей режимов работы и программирования таймеров/счетчиков.

Введение

Очень часто при работе с различными внешними устройствами нужно точно отсчитывать интервалы времени. Простейшим способом является использование программной задержки, которые заставляют процессор выполнять «пустую» операцию, занимающую определенное число тактов. Так работают функции _delay_ms(), _delay_us(), описываемые в модуле <util/delay.h> (первая функция своим аргументом имеет задаваемое программистом количество миллисекунд, вторая – заданное количество микросекунд). Однако, этот способ малоэффективен с точки зрения использования процессорного времени, а кроме того, неточен – количество тактов, в течение которых будет выполняться программа задержки, зависит от модели контроллера и от особенностей компиляции программы с языка С. В этом случае длительность машинного такта зависит от частоты процессора, которая задается от внешнего источника тактового сигнала и может изменяться в широком диапазоне.

В состав AVR-контроллеров входит калиброванный внутренний RC-генератор (контроллер ATMega16 поддерживает частоты 1, 2, 4, 8 МГц). Тактовая частота процессора может задаваться от внешнего источника по трем вариантам: подключением RC-цепочки, кварцевого резонатора или генератора тактовых импульсов. Каждый из указанных способов имеет свои достоинства и недостатки: если внутренний генератор не требует дополнительных цепей, то внешний кварцевый резонатор обеспечивая большую точность, требует дополнительной «обвязки» и место на плате. Отладочный стенд EasyAVR6 имеет в своем составе кварцевый резонатор с

частотой 8 МГц. При прошивке контроллера при помощи программы AVRFLASH нужно просто указать «Использовать внешний генератор» и выбрать частоту контроллера равной 8МГц. При отладке функций задержки поэкспериментируйте с частотами контроллера и генератора, оцените точность формируемых задержек.

Формирование задержки по достижении счетным регистром заданного значения

Наиболее очевидным способом сформировать задержку является периодическая проверка счетного регистра: запускаем функцию, которая в цикле проверяет некий регистр; как только значение этого регистра превысит заданное, происходит выход из функции. Чтобы знать, с каким числом сравнивать значение счетного регистра, нужно знать время одного такта. Рассмотрим пример. Пусть необходимо получить задержку 0.2 с. Предположим, что частота контроллера равна 4000000 Гц, соответственно, время одного колебания $t_0 = 0.25 \cdot 10^{-6}$ с. Чтобы создать задержку в 0.2 с, нужно 800000 тактов. В состав микроконтроллера ATMega16 входит два 8-разрядных счетчика T0 и T2 и один 16-разрядный счетчик T1. Максимальный предделитель частоты и для восьмиразрядных, и для шестнадцатиразрядных счетчиков равен 1024, поэтому в данном случае такую задержку можно сформировать только при помощи счетчика T1. Установим предделитель равным 1024. Тогда время одного отсчета становится равным $t_0' = 256 \cdot 10^{-6}$ с, для обеспечения задержки в 0,2 с необходимо $0,2 / 256 \cdot 10^6 = 781.25$. Округляем это значение до 781 . Теперь осталось правильно отконфигурировать таймер T1.

По справочным данным в работе [1] в разделе «16-разрядные таймеры/счетчики» выписываем формат регистров, входящих в состав таймера/счетчика T1. Для конфигурационных регистров нужно кратко выписать назначение каждого бита. Для решения нашей задачи (написать

функцию временной задержки 0.2 с, используя 16-разрядный счетчик) нам нужен режим Normal (счетчик последовательно увеличивает значение счетного регистра и, дойдя до $2^{16} = 65536$, сбрасывается). Поскольку мы сами программно будем сравнивать значение счетного регистра с заданным числом, разряды, отвечающие за работу регистров сравнения, оставим неустановленными. Обратите внимание, какие управляющие разряды отвечают за источник тактового сигнала и коэффициент деления. Настройте их так, чтобы коэффициент деления был равен 1024. Разряды, касающиеся вывода тактового сигнала во внешние цепи, оставляем неустановленными.

Пример выполнения этого задания дан в листинге 3. Лучше, если вы вначале поработаете со справочником, а потом обратитесь к примеру.

Скомпилируйте и запустите получившуюся программу на микроконтроллере. Обязательно проконтролируйте формируемую задержку по секундомеру.

Листинг 3. Использование счетного регистра таймера для формирования задержки

```
#define F_CPU 4000000UL
#include <avr/io.h>
void wait1(void) /* функция задержки */
{
TCNT1=0; // обнуляем счетный регистр
	while(TCNT1<781){};// ждем, пока содержимое счетного регистра
} // не достигнет нужного значения
int main(void)
{
	unsigned char a;
	int j = 0;
	DDRB = 0xFF;
// конфигурируем таймер T1: выбираем нормальный режим и коэффициент деления 1024
	TCCR1A=0b00000000;
	 // режим работы блока сравнения. Нам эта часть пока не нужна, поэтому нули
	 // 7 - COM1A1 = 0
	 // 6 - COM1A0 = 0
	 // 5 - COM1B1 = 0
	 // 4 - COM1B0 = 0
	 // принудительное изменение состояния OC1A. Нам пока не нужно, нули
	 // 3 - FOC1A = 0
	 // 2 - FOC1B = 0
	 // режим работы таймера
	 // 1 - WGM11 = 0
	 // 0 - WGM10 = 0
	TCCR1B=0b00000101;
	 // 7 - ICNC1 = 0 - схема подавления помех блока захвата отключена
	 // 6 - ICES1 = 0 - выбран сигнал захвата по спаду
```

```
        // 5 - не используется, равен 0
        // режим работы таймера
        // 4 - WGM13 = 0
        // 3 - WGM12 = 0
// Все биты WGM13..WGM10 установлены в 0, то есть выбран нормальный режим работы таймера
        // 2 - CS12 = 1 - //Эти три бита определяют коэффициент деления частоты.
        // 1 - CS11 = 0   //Значение 0b101 означает деление частоты тактового
        // 0 - CS10 = 1   //сигнала на 1024
      while(1)
      {
            a = 0b10000000;
            PORTB = 0x00;
            j = 0;
            while (a != 0)
            {
                  j++;
                  PORTB = a;
                  a = a >> 1;
                  wait1();
            }

      }
      return 0;
}
```

Задания

1) Создайте функцию задержки на 30 с при помощи 8-разрядного таймера/счетчика.

2) С использованием 8-разрядного таймера/счетчика создайте функцию задержки, в которой входным параметром было бы количество миллисекунд.

3) Реализуйте управление по числу нажатий кнопки: 1 раз – задержка 0.1 с, 2 раза – задержка 0.5 с, 3 раза – задержка 1 с, 4 раза – задержка 0.1 с и т.д.

Формирование задержки с использованием прерывания по переполнению таймера-счетчика

В предыдущем примере мы сформировали достаточно точно временные задержки при помощи таймеров, однако не пользовались главным преимуществом таймеров – разгрузки вычислительного ядра за счет использования прерываний. Конфигурирование таймера при обработке прерывания по переполнению остается прежним. Необходимо установить общее разрешение прерываний (команда sei()), но прежде установить разрешение прерывания по переполнению выбранного счетчика в регистре TIMSK (как правильно это сделать, см. раздел «Прерывания от

таймеров/счетчиков» в работе [1]). Также нужно описать функцию обработки прерывания. В AVR GCC для этого нужно подключить заголовочный файл <avr/interrupt.h>, функция обработки прерывания должна называться ISR, а своим аргументом она будет иметь конкретное название прерывания (выписаны в комментариях в листинге 2). Обратите внимание, что вся функциональность, реализованная ранее в функции main, теперь реализуется в функции обработки прерывания. Внимательно разберитесь с листингом 2.

Задание

Создайте функцию задержки с использованием прерывания по переполнению таймера T0. Время задержки и частоту контроллера укажет преподаватель.

Листинг 4. Формирование временной задержки с использованием прерывания по переполнению счетчика

```
#define F_CPU 4000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
unsigned char a = 0b10000000; // выбираем начальное положение огонька PB7
ISR(TIMER1_OVF_vect) // прерывание по переполнению счетчика T1
{
	TCNT1=64755; // 2^16-781 - установили начальное значение	a
	a = a >> 1; // сдвинули 1 влево на 1 разряд
	if (a == 0) a = 0b10000000; // если достигли края, возвращаемся
	PORTB = rab; // выводим новое значение
}
// другие макроимена прерываний от таймеров для ATMega16

// TIMER0_COMP_vect --- Таймер-счетчик0 совпадение
// TIMER0_OVF_vect --- Таймер-счетчик0 переполнение
// TIMER1_CAPT_vect --- Таймер-счетчик1 захват
// TIMER1_COMPA_vect -- Таймер-счетчик1 совпадение A
// TIMER1_COMPB_vect -- Таймер-счетчик1 совпадение B
// TIMER1_OVF_vect -- Таймер-счетчик1 переполнение
// TIMER2_COMP_vect -- Таймер-счетчик2 совпадение
// TIMER2_OVF_vect -- Таймер-счетчик2 переполнение

int main(void)
{
	DDRB  =  0xFF;
	DDRD  =  0xFF;
	PORTB = 0x80;
// конфигурируем таймер T1: выбираем нормальный режим и коэффициент деления 1024
	TCCR1A=0b00000000;
	 // режим работы блока сравнения. Нам эта часть пока не нужна, поэтому нули
	 // 7 - COM1A1 = 0
	 // 6 - COM1A0 = 0
```

```
        // 5 - COM1B1 = 0
        // 4 - COM1B0 = 0
        // принудительное изменение состояния OC1A. Нам пока не нужно, нули
        // 3 - FOC1A = 0
        // 2 - FOC1B = 0
        // режим работы таймера
        // 1 - WGM11 = 0
        // 0 - WGM10 = 0
       TCCR1B=0b00000101;
        // 7 - ICNC1 = 0 - схема подавления помех блока захвата отключена
        // 6 - ICES1 = 0 - выбран сигнал захвата по спаду
        // 5 - не используется, равен 0
        // режим работы таймера
        // 4 - WGM13 = 0
        // 3 - WGM12 = 0
// Все биты WGM13..WGM10 установлены в 0, то есть выбран нормальный режим работы таймера
        // 2 - CS12 = 1
        // 1 - CS11 = 0
        // 0 - CS10 = 1
// Эти три бита определяют коэффициент деления частоты. Значение 101 означает
// деление частоты тактового сигнала на 1024
 TIMSK |= (1<<TOIE1);// установить разрешения прерывания по переполнению таймера 1
 // То есть нам нужно установить 1 в бит TOIE1
// Формат регистра TIMSK для контроллера ATMega16
// 7 - OCIE2 - флаг разрешения прерывания по событию "совпадение" счетчика T2
// 6 - TOIE2 - флаг разрешения прерывания по переполнению счетчика T2
// 5 - TICIEI1 - флаг разрешения прерывания по событию "захват" счетчика T1
// 4 - OCIE1A - флаг разрешения прерывания по событию "Совпадение A" счетчика T1
// 3 - OCIE1B - флаг разрешения прерывания по событию "Совпадение B" счетчика T1
// 2 - TOIE1 - флаг разрешения прерывания по переполнению счетчика T1
// 1 - OCIE0 - флаг разрешения прерывания по событию "совпадение" счетчика T0
// 0 - TOIE0 - флаг разрешения прерывания по переполнению счетчика T0
 sei(); // общее разрешение прерываний
        TCNT1=64755; // 2^16-781 - устанавливаем начальное значение счетного регистра
        while(1){};
        return 0;
}
```

Формирование задержки с использованием прерывания по совпадению таймера/счетчика

Этот способ отличается от предыдущего тем, что необходимо настроить счетчик не в режиме Normal, а в режиме CTC (листинг 3).

Задание

Создайте функцию задержки с использованием прерывания по совпадению таймера T0. Время задержки и частоту контроллера укажет преподаватель.

Листинг 5. Формирование временной задержки с использованием прерывания по совпадению счетчика

```
#define F_CPU 4000000UL
#include <avr/io.h>
```

```
#include <avr/interrupt.h>

unsigned char a = 0b10000000; // выбираем начальное положение огонька PB7

ISR(TIMER1_COMPA_vect) // прерывание по сопадению A счетчика T1
{
        a = a >> 1; // сдвинули 1 влево на 1 разряд
        if (a == 0) a = 0b10000000; // если достигли края, возвращаемся
        PORTB = a; // выводим новое значение
};

int main(void)
{
        DDRB = 0xFF;
        DDRD = 0xFF;
        PORTD = 0x80;//0b10000000
        PORTB = 0x80;//0b10000000
       // конфигурируем таймер T1: выбираем режим CTC и коэффициент деления 1024
       TCCR1A=0b00000000;
       TCCR1B=0b00001101;
       // WGM13 WGM12 WGM11 WGM10 = 0100 - выбран режим сброса таймера при совпадении
       // счетного регистра с регистром OCR1A
       // Младшие три бита определяют коэффициент деления частоты. Значение 101 означает
       // деление частоты тактового сигнала на 1024
        TIMSK |= (1<<OCIE1A);
       // установить разрешения прерывания по переполнению таймера 1
       // в регистр OCR1A записываем число 781 - таймер должен быть сброшен
       //при достижении счетным регистром TCNT1 этого значения
        OCR1AH = 0x03; // старший байт число 781
        OCR1AL = 0x0D; // младший байт числа 781
        sei(); // общее разрешение прерываний
        while(1){};
        return 0;
};
```

ЛАБОРАТОРНАЯ РАБОТА № 4

Управление шаговым двигателем

Цель работы: изучение конструктивных особенностей и способов управления шаговыми двигателями, составление и отладка программ управления шаговыми двигателями на физическом макете.

Введение

Шаговый двигатель – это электромеханическое устройство, которое преобразует электрические импульсы в дискретные механические перемещения [11]. Преимущества шаговых двигателей:

- угол поворота ротора определяется числом импульсов, которые поданы на двигатель;
- двигатель обеспечивает полный момент в режиме остановки (если на обмотки подано напряжение);
- прецизионное позиционирование и повторяемость, возможность позиционирования без обратной связи;
- возможность быстрого старта/остановки/реверсирования;
- высокая надежность, связанная с отсутствием щеток, срок службы шагового двигателя фактически определяется сроком службы подшипников;
- возможность получения очень низких скоростей вращения для нагрузки, присоединенной непосредственно к валу двигателя без промежуточного редуктора;
- в довольно большом диапазоне скорость пропорциональна частоте входных импульсов.

Недостатки шаговых двигателей:

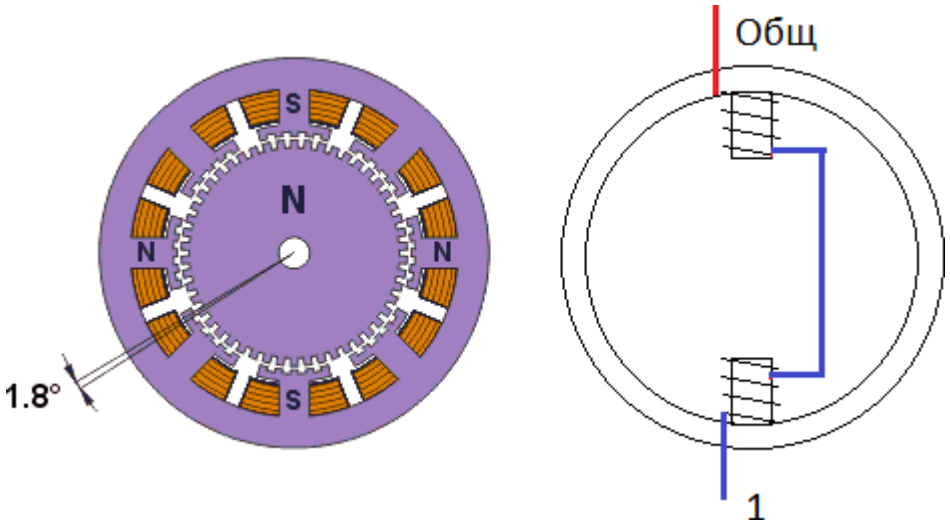
- шаговым двигателем присуще явление резонанса;
- потребление энергии не уменьшается даже без нагрузки;

- затруднена работа на высоких скоростях;
- невысокая удельная мощность;
- относительно сложная схема управления.

Устройство шаговых двигателей

В шаговом двигателе вращающий момент создается магнитными потоками статора и ротора, которые соответствующим образом ориентированы друг относительно друга. Статор изготовлен из материала с высокой магнитной проницаемостью и имеет несколько полюсов. Вращающий момент пропорционален величине магнитного поля, которая пропорциональна току в обмотке и количеству витков. Если хотя бы одна обмотка шагового двигателя запитана, ротор принимает определенное положение. Он будет находиться в этом положении до тех пор, пока внешний приложенный момент не превысит некоторого значения, называемого моментом удержания. После этого ротор повернется и будет стараться принять одно из следующих положений равновесия.

Статор шаговых двигателей имеет несколько обмоток, ротор имеет зубцы, расположенные в осевом направлении (Рисунок 1.19 а).



а) б)

Рисунок 1.19. Общее устройство шагового двигателя.

Ротор разделен на две части, между которыми расположен цилиндрический постоянный магнит. Таким образом, зубцы верхней

половинки ротора являются северными полюсами, а зубцы нижней половинки – южными. Число пар полюсов ротора равно количеству зубцов на одной из его половинок. Статор гибридного двигателя также имеет зубцы, обеспечивая большое количество эквивалентных полюсов, в отличие от основных полюсов, на которых расположены обмотки. На рисунке Рисунок 1.19 а показан двигатель, имеющий 8 основных полюсов. Зубцы ротора обеспечивают меньшее сопротивление магнитной цепи в определенных положениях ротора, что улучшает статический и динамический момент. Это обеспечивается соответствующим расположением зубцов, когда часть зубцов ротора находится строго напротив зубцов статора, а часть между ними. Хотя количество основных полюсов может быть различным, шаговые двигатели имеют, как правило, не более четырех обмоток. Обмотки диаметрально противоположно расположенных полюсов электрически могут быть соединены последовательно, как показано на Рисунок 1.19 б. На Рисунок 1.20 показано электрическое устройство униполярного шагового двигателя: имеются 4 обмотки, соединенные звездой. Подавая по очереди на каждую из обмоток электрические импульсы, мы создаем вращающееся магнитное поле, которое заставляет вращаться вал двигателя.

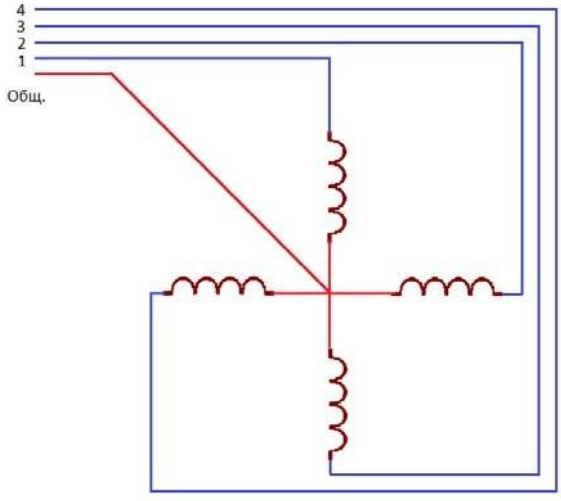


Рисунок 1.20. Соединение обмоток униполярного шагового двигателя

Биполярные и униполярные шаговые двигатели

В зависимости от конфигурации обмоток двигатели делятся на биполярные и униполярные. Биполярный двигатель имеет одну обмотку в

каждой фазе, которая для изменения направления магнитного поля должна переполюсовываться драйвером. Для такого типа двигателя требуется мостовой драйвер, или полумостовой с двухполярным питанием. Всего биполярный двигатель имеет две обмотки и, соответственно, четыре вывода (Рисунок 1.21).

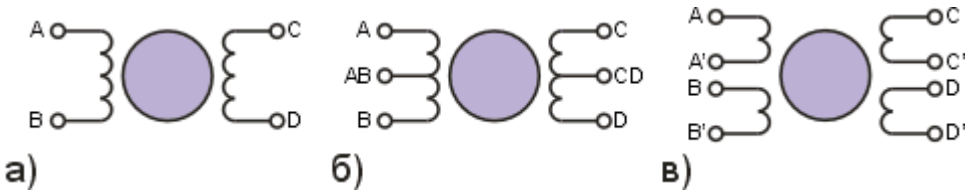


Рисунок 1.21. Биполярный двигатель (а), униполярный (б) ичетырехобмоточный (в).

Униполярный двигатель также имеет одну обмотку в каждой фазе, но от середины обмотки сделан отвод. Это позволяет изменять направление магнитного поля, создаваемого обмоткой, простым переключением половинок обмотки. При этом существенно упрощается схема драйвера. Драйвер должен иметь только 4 простых ключа (Рисунок 1.22).

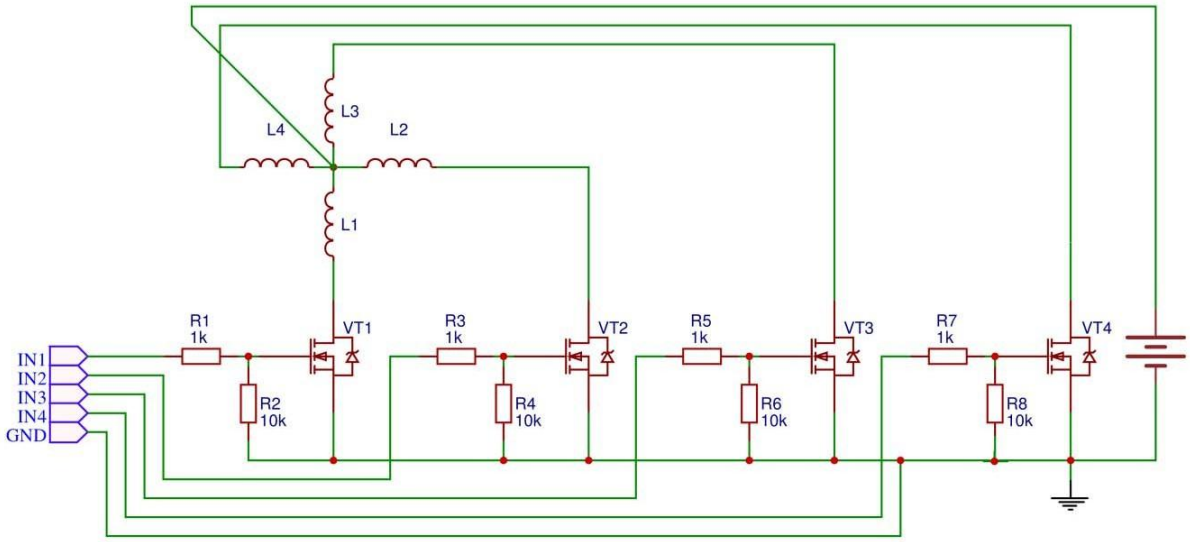


Рисунок 1.22. Схема драйвера униполярного двигателя на n-канальных MOSFET-

транзисторах

Таким образом, в униполярном двигателе используется другой способ изменения направления магнитного поля. Средние выводы обмоток могут

быть объединены внутри двигателя, поэтому такой двигатель может иметь 5 или 6 выводов (рисунок 3 б). Схема подключения обмоток, показанная на рисунке 2, соответствует соединению средних отводов обмоток AB и CD (рисунок 3б). Иногда униполярные двигатели имеют раздельные 4 обмотки (рисунок 3в). При соответствующем соединении обмоток такой двигатель можно использовать как униполярный или как биполярный. Униполярный двигатель с двумя обмотками и отводами тоже можно использовать в биполярном режиме, если отводы AB и CD оставить неподключенными. В любом случае ток обмоток следует выбирать так, чтобы не превысить максимальной рассеиваемой мощности.

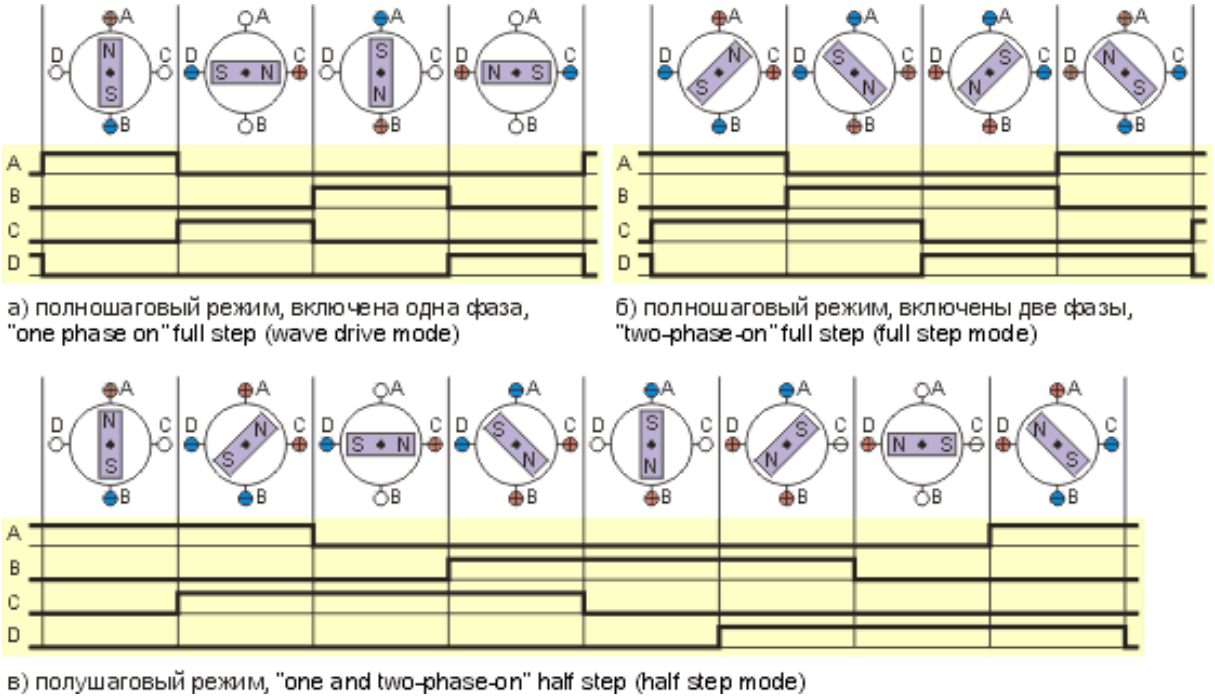
Если сравнивать между собой биполярный и униполярный двигатели, то биполярный при одних и тех же размерах обеспечивает больший момент. Момент, создаваемый шаговым двигателем, пропорционален величине магнитного поля, создаваемого обмотками статора. Путь для повышения магнитного поля – это увеличение тока или числа витков обмоток. Естественным ограничением при повышении тока обмоток является опасность насыщения железного сердечника. Однако на практике это ограничение действует редко. Гораздо более существенным является ограничение по нагреву двигателя вследствие омических потерь в обмотках. В униполярном двигателе в каждый момент времени используется лишь половина обмоток. Другая половина просто занимает место в окне сердечника, что вынуждает делать обмотки проводом меньшего диаметра. В то же время в биполярном двигателе всегда работают все обмотки, поэтому сечение отдельных обмоток вдвое больше, а омическое сопротивление – соответственно вдвое меньше. Это позволяет увеличить ток в $\sqrt{2}$ раз при тех же потерях, что дает выигрыш в моменте примерно 40%. Если же повышенного момента не требуется, униполярный двигатель позволяет уменьшить габариты или просто работать с меньшими потерями. Униполярные двигатели требуют значительно более простых схем управления обмотками. Однако в настоящее время существуют

специализированные микросхемы драйверов для биполярных двигателей, с использованием которых драйвер получается не сложнее, чем для униполярного двигателя.

Способы управления шаговыми двигателями

Существует несколько способов управления фазами шагового двигателя. Первый способ обеспечивается попеременной коммутации фаз, при этом они не перекрываются, в один момент времени включена только одна фаза (Рисунок 1.23 а). Этот способ называют ”one phase on” full step или wave drive mode. Точки равновесия ротора для каждого шага совпадают с

«естественными» точками равновесия ротора у незапитанного двигателя. Недостатком этого способа управления является то, что для биполярного двигателя в один и тот же момент времени используется 50% обмоток, а для униполярного – только 25%. Это означает, что в таком режиме не может быть получен полный момент.



а) полношаговый режим, включена одна фаза, "one phase on" full step (wave drive mode)

б) полношаговый режим, включены две фазы, "two-phase-on" full step (full step mode)

в) полушаговый режим, "one and two-phase-on" half step (half step mode)

Рисунок 1.23. Различные способы управления фазами шагового двигателя Второй способ - управление фазами с перекрытием: две фазы

включены в одно и то же время. Его называют ”two-phase-on” full step или

просто full step mode. При этом способе управления ротор фиксируется в промежуточных позициях между полюсами статора (рисунок 1.23 б) и обеспечивается примерно на 40% больший момент, чем в случае одной включенной фазы. Этот способ управления обеспечивает такой же угол шага, как и первый способ, но положение точек равновесия ротора смещено на полшага.

Третий способ является комбинацией первых двух и называется полушаговым режимом, ”one and two-phase-on” half step или просто half step mode, когда двигатель делает шаг в половину основного. Каждый второй шаг запитана лишь одна фаза, а в остальных случаях запитаны две (Рисунок 1.23 в). В результате угловое перемещение ротора составляет половину угла шага для первых двух способов управления. Кроме уменьшения размера шага этот способ управления позволяет частично избавиться от явления резонанса. Полушаговый режим обычно не позволяет получить полный момент, хотя наиболее совершенные драйверы реализуют модифицированный полушаговый режим, в котором двигатель обеспечивает практически полный момент, при этом рассеиваемая мощность не превышает номинальной.

Еще один способ управления называется микрошаговым режимом или micro stepping mode. При этом способе управления ток в фазах нужно менять небольшими шагами, обеспечивая таким образом дробление половинного шага на еще меньшие микрошаги. Когда одновременно включены две фазы, но их токи не равны, то положение равновесия ротора будет лежать не в середине шага, а в другом месте, определяемом соотношением токов фаз. Меняя это соотношение, можно обеспечить некоторое количество микрошагов внутри одного шага. Вместе с тем, для реализации микрошагового режима требуются значительно более сложные драйверы, позволяющие задавать ток в обмотках с необходимой дискретностью. Полушаговый режим является частным случаем микрошагового режима, но он не требует формирования ступенчатого тока питания катушек, поэтому часто реализуется.

Пример программы

В лабораторной работе используется униполярный шаговый двигатель MITSUMI M42SP-5, вал которого через редуктор, систему направляющих и тросиков перемещает каретку. Угол поворота вала этого двигателя составляет 7.5° . Сопротивление обмоток составляет порядка 50 Ом, питание осуществляется от источника постоянного напряжения 12 В. Рекомендованный режим – 200 импульсов в секунду.

Физический макет, на котором размещен шаговый двигатель, по линиям IN1-IN4 плоским шлейфом необходимо подключить к выводам порта D – PD0-PD3 на штыревом разъеме стенда EasyAVR6, в программе эти выводы необходимо сконфигурировать на выход. Выпишем ниже функцию, позволяющую реализовать режим ”one phase on” full step (полношаговый режим, задействующий одну фазу).

```
/* полношаговый режим с одной включенной фазой – движение вперед */
void step_forward(int num, int tact)
{
	int i;
	for(i=0; i<num; i++)
	{
		PORTB = 0b00001000;
		wait1(tact);
		PORTB = 0b00000100;
		wait1(tact);
		PORTB = 0b00000010;
		wait1(tact);
		PORTB = 0b00000001;
		wait1(tact);
	}
}
```

Как видно из кода, подаем на каждую из обмоток по очереди импульс, процесс повторяем num раз. Функция wait1() служит для формирования временной задержки. Желательно, чтобы принимаемым аргументом было время в миллисекундах. Можно использовать в качестве такой функции штатные функции _delay_ms(), _delay_us(), а можно реализовать собственную функцию на основе таймера-счетчика (по аналогии с используемой в лабораторной работе №3).

Задания

1) Реализуйте по временным диаграммам, представленным на рисунке 4, полношаговый режим с двумя включенными фазами (”two-phase-on” full step) и полушаговый режим (”one and two-phase-on” half step). Оформите алгоритмы в виде функций, принимающих в качестве аргументов количество шагов, время включения одной фазы в миллисекундах, направление вращения вала двигателя.

2) Проверьте реализованные в первом пункте функции. Для каждой функции измерьте расстояние, на которое переместится каретка за 200 циклов в каждом из режимов. Сравните пути и скорости, проходимые кареткой в разных режимах. Используя эти данные, реализуйте функцию или макрос, пересчитывающий заданное расстояние в миллиметрах в количество шагов. Запустив движение каретки в одном направлении, а затем – с теми же самыми параметрами (количество шагов и время включения одной фазы) в другом направлении, убедитесь, что каретка возвращается в исходное положение. Испытайте функцию на различных временах включения одной фазы, выявите граничное значение, при котором вал двигателя прекращает вращаться. Постройте график зависимости скорости перемещения каретки от времени включения одной фазы для полношагового режима.

3) Используя прерывание по совпадению таймера-счетчика для формирования временных задержек, реализуйте все три описанные выше режима работы шагового двигателя. Используя осциллограф с логическим анализатором, снимите временные диаграммы каждого из режимов работы (щупы логического анализатора нужно подключить вместо входов драйвера двигателя).

4) Выведите каретку на середину полозьев, заставьте ее циклически двигаться вперед-назад на 3.5 см. Пронаблюдайте не менее 10 колебаний. На сколько сместилась каретка от своего начального положения?

5) Напишите программу, которая заставляет каретку двигаться вперед, если нажата кнопка PD0, останавливаться, если нажата кнопка PD1, и двигаться назад, если нажата кнопка PD2.

6) Реализуйте обработку сигнала с концевых датчиков, построенных на основе оптопары. Если при движении каретки луч в оптроне оказался перекрыт шторкой, установленной на краю каретки, прекратите движение и вернитесь на несколько шагов назад.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Евстифеев, А.В. Микроконтроллеры семейства Mega. Руководство пользователя. М.: «Издательский дом Додэка – XXI», 2007. – 592 с.
2. Микропроцессорные системы: Учебное пособие для вузов / Е.К. Александров, Р.И. Грушвицкий, М.С. Куприянов и др.; Под общей ред. Д.В. Пузанкова. – СПб.: Политехника, 2002. – 935 с.
3. Аппаратные и программные средства встраиваемых систем: учеб. пособие / А.О. Ключев, Д.Р. Ковязина, Е.В. Петров, А.Е. Платунов. – СПб.: СПбГУ ИТМО, 2010. – 287 с.
4. Белов, А.В. Разработка устройств на микроконтроллерах AVR / А.В. Белов. – М.: Наука и техника, 2012. – 528 с.
5. Хартов, В.Я. Микроконтроллеры AVR. Практикум для начинающих: учеб. Пособие / В.Я. Хартов. – 2-е изд. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2012. – 280 с.
6. Конченков В.И., Скакунов В.Н. Семейство микроконтроллеров STM32. Программирование и применение: учеб. пособие – ВолгГТУ. – Волгоград, 2015. – 77 с.
7. Редькин, П.П. Микроконтроллеры ARM7 семейства LPC2000. Руководство пользователя. – М.: Издательский дом «Додэка-XXI», 2007. – 560 с.
8. ARM7TDMI (Rev.3) Technical Reference Manual - ARM© 1994-2001
9. Редькин, П.П. Микроконтроллеры Atmel архитектуры AVR32 семейства AT32UC3. Руководство пользователя. – М.: Техносфера, 2010. – 784 с.
10. Зубарев, А.А. Ассемблер для микроконтроллеров AVR: Учебное пособие. – Омск: Изд-во СибАДИ, 2007. – 112 с.
11. Шаговые двигатели: учеб. пособие/ А. В. Емельянов, А. Н. Шилин/ ВолгГТУ. - Волгоград, 2005. - 48 с.