МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Белорусский национальный технический университет

Кафедра «Робототехнические системы»

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ

Лабораторный практикум

Часть 1

Минск
БНТУ
2014

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Белорусский национальный технический университет

Кафедра «Робототехнические системы»

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ

Лабораторный практикум
для студентов специальностей
1-53 01 01 «Автоматизация технологических процессов
и производств», 1-53 01 06 «Промышленные роботы
и робототехнические комплексы»

В 2 частях

Часть 1

Минск
БНТУ
2014

УДК 004.31-181.48(075.8)
ББК 32.97я7
П78

Составители:
Ф. Л. Сиротин, А. М. Капустина,
Ю. А. Агейчик, Е. В. Голубчик

Рецензенты:
А. В. Василевский, И. А. Кашталян

П78 **Программирование** микроконтроллеров : лабораторный практикум для студентов специальностей 1-53 01 01 «Автоматизация технологических процессов и производств», 1-53 01 06 «Промышленные роботы и робототехнические комплексы» : в 2 ч. / сост.: Ф. Л. Сиротин [и др.]. – Минск : БНТУ, 2014– . – Ч. 1. – 2014. – 64 с.
ISBN 978-985-550-261-7 (Ч. 1).

Излагаются основные сведения о микроконтроллерах AVR семейства Mega, их программировании на языке СИ, описывается стенд для реализации различных программ управления электродвигателем на основе микроконтроллера Atmega 8, приводятся краткие сведения о пакете Proteus, позволяющем моделировать схемы в т.ч. с микроконтроллерами.

УДК 004.31-181.48(075.8)
ББК 32.97я7

ISBN 978-985-550-261-7 (Ч. 1)
ISBN 978-985-550-262-4

1. Введение в микроконтроллеры AVR семейство Mega

Современную микроэлектронику трудно представить без такой важной составляющей, как микроконтроллеры (в дальнейшем МК). Микроконтроллерные технологии очень эффективны. Одно и то же устройство, которое раньше собиралось на традиционных элементах, будучи собрано с применением микроконтроллеров, становится проще. Оно не требует регулировки и меньше по размерам. Кроме того, с применением микроконтроллеров появляются практически безграничные возможности по добавлению новых потребительских функций и возможностей к уже существующим устройствам. Достаточно просто поменять программу.

ATMega8 — 8-разрядный КМОП микроконтроллер, основанный на архитектуре Atmel AVR. Контроллер выполняет большинство инструкций за 1 такт, поэтому вычислительная мощность контроллера равна 1MIPS на 1 Мгц.

МК имеет RISC-архитектуру, но формат команды двухоперандный, за один такт может быть обращение только к двум регистрам. Контроллер содержит 32 регистра, которые могут равноправно использоваться в арифметических операциях.

Основные аппаратные характеристики МК:

* 8 Кбтфлеш-памяти команд;
* 512 байт электрически программируемой памяти;
* 1 Кбайт статической памяти;
* 23 линии ввода/вывода общего назначения;
* 32 РОНа;
* три многоцелевых таймер-счётчика с режимом сравнения;
* поддержка внутренних и внешних прерываний;
* универсальный асинхронный адаптер;
* байт-ориентированный двухпроводной последовательный интерфейс;
* 6/8 канальный АЦП с точностью 8 и 10 двоичных разрядов;
* сторожевой таймер;
* последовательный порт SPI;
* расширенные режимы управления энергопотреблением.

Ядро микроконтроллера — AVR.

Количество записей во флэш-память – 10 000 раз.

Рассмотрим, что же он из себя внешне представляет (рис. 1.1).

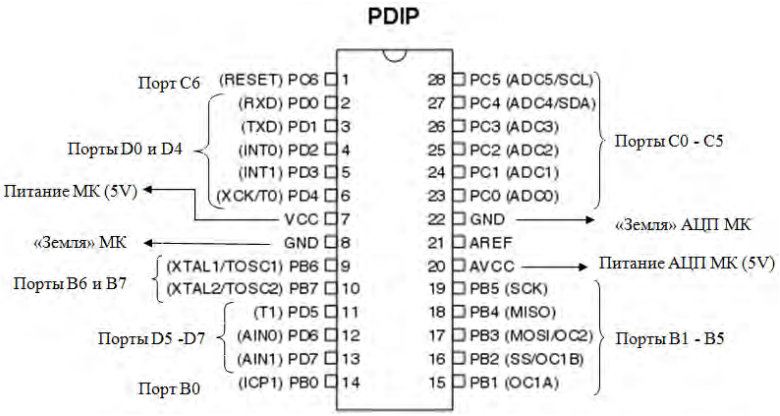

Рис. 1.1. МК Atmega8 в корпусе DIP

На рис. 1.1 приведен МК Atmega8 в DIP корпусе (существуют и другие корпуса) с обозначением ножек. Кратко изучим их назначение (подробно изучим с лабораторных работах):

- все то, что имеет обозначение PX – это порты где, X – номер порта. На все порты можно подавать сигналы или принимать, что и будем делать в лабораторных работах;
- питание МК в 5 вольт подается на ножки VCC и GND;
- программирование МК производится по ножкам: 1 (reset), 17 (MOSI), 18 (MISO), 19 (SCK)с помощью программатора.

Специально разработанные лабораторные работы построены таким образом, чтобы любой человек смог изучить язык программирования (в данном случае C++, но существует и другие языки) от практически нулевого уровня, до уровня, позволяющего писать программы средней сложности.

Каждая новая лабораторная работа начинается с постановки задачи. Затем вы можете увидеть процесс построения алгоритма и,

наконец, увидите, как создается управляющая программа для наших задач.

Все программные работы и примеры к ним приведены на языке СИ, который поддерживается средой программирования МК AVR – CodeVisionAVR.

2. Введение в язык C++ и CodeVisionAVR

CodeVisionAVR — это кросс-компилятор СИ, интегрированная среда разработки (IDE — Integrated Development Environment) и автоматический генератор программ (CodeWizardAVR), разработанные для семейства AVR-МК фирмы Atmel.

Программа является 32-битовым приложением, которое работает под операционными системами Windows 95, 98, NT 4, 2000 , XP, 7.

CodeVisionAVR обеспечивает выполнение почти всех элементов языка C++ (в дальнейшем просто «С» или «Си»), которые разрешены архитектурой AVR, с некоторыми добавленными характеристиками, которые реализуют преимущество специфики архитектуры AVR.

Программное обеспечение CodeVision может работать с такими программаторами серии Atmel, как с STK500/AVRISP/AVRProg , KandaSystems STK200+/300, Dontronics DT006, VogelElektronik VTEC-ISP, Futurlec JRAVR и платой разработчика MicroTronics ATCPU/Mega2000.

Кроме стандартных библиотек СИ, компилятор СИ CodeVisionAVR имеет библиотеки для:

- алфавитно-цифровых LCD-модулей;
- шины I2C от Philips;
- температурного датчика LM75 от National Semiconductor;
- часов реального времени PCF8563, PCF8583 от Philips и DS1302, DS1307 от Dallas Semiconductor;
- протокола 1 -Wire от Dallas Semiconductor;
- температурного датчика DS1820/DS18S20 от Dallas Semiconductor;
- термометра/термостата DS1621 от Dallas Semiconductor;
- EEPROM DS2430 и DS2433 от Dallas Semiconductor;
- SPI;

- управления питанием;
- задержек;
- преобразования кода Грея.

CodeVisionAVR также содержит автоматический генератор программ — CodeWizardAVR, который позволяет написать за несколько минут весь код, необходимый для выполнения следующих функций:

- установка доступа к внешней памяти;
- идентификация источника сброса чипа;
- инициализация порта ввода/вывода;
- инициализация внешних прерываний;
- инициализация таймеров/счётчиков;
- инициализация сторожевого таймера;
- инициализация UART и прерываний, управляющих буфером последовательной связи;
- инициализация аналогового компаратора;
- инициализация АЦП;
- инициализация интерфейса SPI;
- инициализация шины 12C, температурного датчика LM75, термометра/термостата DS1621 и часов реального времени PCF8563, PCF8583, DS1302, DS1307;
- инициализация шины 1-Wire и температурного датчика DS1820/DS18S20;
- инициализация LCD-модуля.

В лабораторных работах приводятся лишь начальные сведения о языке СИ и, где необходимо, поясняются специфические особенности реализации языка компилятором СИ CodeVisionAVR.

Препроцессор (макропроцессор) — это составная часть языка СИ, которая обрабатывает исходный текст программы до того, как он пройдет через компилятор. Препроцессор читает строки текста и выполняет действия, определяемые командными строками. Если первым символом в строке, отличным от пробела, является символ #, то такая строка рассматривается препроцессором как командная. Командные строки называются директивами препроцессора.

Директивы препроцессора позволяют:

- включать в программу текст из других файлов;
- передавать компилятору специальные директивы;
- определять макросы, которые облегчают программирование и улучшают удобочитаемость исходного кода;
- задавать условия компиляции для отладочных целей и/или для уменьшения размера получаемого кода.

Препроцессор компилятора CodeVisionAVR имеет несколько директив. В табл. 2.1 даётся их краткое описание.

Таблица 2.1

Директивы препроцессора компилятора CodeVisionAVR

Директива	Назначение
#include	Используется для включения в программу другого файла
#define	Используется для замены одних лексических единиц языка Си на другие, а также для генерации макросов
#undef	Используется для отмены действия директивы #define
#if	Используются для условной компиляции
#ifdef	
#ifndef	
#else	
#endif	
#line	Используется для изменения встроенных макросов _LINE_ и_FILE_
#error	Позволяет остановить компиляцию и отобразить сообщение об ошибке
#asm	Используются для включения в программу ассемблерного кода
#endasm	
#pragma…	Разрешает специальные директивы компилятора

В лабораторных работах будет использовать только директива include, которая позволяет в дальнейшем использовать функции и команды, не разрешенные в СИ.

Операнд — это константа, литерал, идентификатор, вызов функции, индексное выражение, выражение выбора элемента или более сложное выражение, сформированное комбинацией операндов, зна-

ков операций и круглых скобок. Иными словами, операнд – это математическая функция. Каждый операнд имеет тип (табл. 2.2).

Таблица 2.2

Операнды

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

В языке СИ переменные делятся на типы. Переменная каждого типа может принимать значения из одного определенного диапазона. Например:

- переменная типа char — это только целые числа;
- переменная типа float — вещественные числа (десятичная дробь) и т. д.

Использование переменных нескольких фиксированных типов — это отличительная особенность любого языка высокого уровня. Разные версии языка СИ поддерживают различное количество типов переменных. Версия СИ, используемая в CodeVisionAVR, поддерживает тринадцать типов переменных (табл. 2.3).

Таблица 2.3

Типы переменных языка СИ в CodeVisionAVR

Название	Количество бит	Значение
bit	1	0 или 1
char	8	-128 – 127
unsigned char	8	0 – 255
signed char	8	-128 – 127
int	16	-32768 – 32767
short int	16	-32768 – 32767
unsigned int	16	0 – 65535
signed int	16	-32768 – 32767
long int	32	-2147483648 – 2147483647
unsigned long int	32	0 – 4294967295
signed long int	32	-2147483648 – 2147483647
float	32	±1.175e-38 – ±3.402e38
double	32	±1.175e-38 – ±3.402e38

В языке СИ любая переменная, прежде чем будет использована, должна быть описана. При описании задается ее тип. В дальнейшем диапазон принимаемых значений должен строго соответствовать

выбранному типу переменной. Описание переменной и задание типа необходимы потому, что оттранслированная с языка СИ программа выделяет для хранения значений каждой переменной определенные ресурсы памяти.

Это могут быть ячейки ОЗУ, регистры общего назначения или даже ячейки EEPROM или Flash-памяти (памяти программ). В зависимости от заданного типа, выделяется различное количество ячеек для каждой конкретной переменной. Описывая переменную, мы сообщаем транслятору, сколько ячеек выделять и как затем интерпретировать их содержимое. Посмотрим, как выглядит строка описания переменной в программе. Она представляет собой запись следующего вида:

Тип Имя;

где «Тип» — это тип переменной, а «Имя» — ее имя.

Имя переменной выбирает программист. Допускается использование только латинских букв, цифр и символа подчеркивания. Начинаться имя должно с буквы или символа подчеркивания.

Кроме арифметических и логических выражений язык СИ использует функции. В отличие от математических функций, функции языка СИ не всегда имеют входные значения и даже не обязательно возвращают результат. Далее на конкретных примерах мы увидим, как и почему это происходит.

Вообще, роль функций в языке СИ огромная. Программа на языке СИ просто-напросто состоит из одной или нескольких функций. Каждая функция имеет свое имя и описание. По имени производится обращение к функции. Описание определяет выполняемые функцией действия и преобразования. Вот как выглядит описание функции в программе СИ:

```
тип Name (список параметров) {
тело функции }
```

Здесь Name — это имя функции. Имя для функции выбирается по тем же правилам, что и для переменной. При описании функции перед ее именем положено указать тип возвращаемого значения. Это необходимо транслятору, так как для возвращаемого значения он тоже резервирует ячейки.

Если перед именем функции вместо типа возвращаемого значения записать слово void, то это будет означать, что данная функция не возвращает никаких значений. В круглых скобках после имени функции записывается список передаваемых в нее параметров.

Функция может иметь любое количество параметров. Если параметров два и более, то они записываются через запятую. Перед именем каждого параметра также должен быть указан его тип. Если у функции нет параметров, то в скобках вместо списка параметров должно стоять слово void. В фигурных скобках размещается тело функции:

```
void main (void) {a=5;}
```

В этом случае операторы, составляющие тело функции, разделяет только точка с запятой. Вот пример такой записи:

```
тип Name (список параметров) {тело функции }
```

```
void main(void) {a=5;c=6;}
```

Любая программа на языке СИ должна обязательно содержать одну главную функцию. Главная функция должна иметь имя main. Выполнение программы всегда начинается с выполнения функции main. Функция main в данной версии языка СИ никогда не имеет параметров и никогда не возвращает никакого значения.

Тело функции, кроме команд, может содержать описание переменных. Все переменные должны быть описаны в самом начале функции, до первого оператора. Такие переменные могут быть использованы только в той функции, вначале которой они описаны. Вне этой функции данной переменной как бы не существует.

Если вы объявите переменную в одной функции, а примените ее в другой, то транслятор выдаст сообщение об ошибке. Это дает возможность объявлять внутри разных функций переменные с одинаковыми именами и использовать их независимо друг от друга.

Глобальная переменная объявляется не внутри функций, а в начале программы, еще до описания самой первой функции. Не спешите без необходимости делать переменную глобальной. Если

программа достаточно большая, то можно случайно присвоить двум разным переменным одно и то же имя, что приведет к ошибке. Такую ошибку очень трудно найти.

Начнем изучение СИ с описания нам пока неизвестных используемых там команд, которые пригодятся в при выполнении лабораторных работ.

include

Оператор присоединения внешних файлов. В 1 строке нашей будущей программы этот оператор присоединяет к основному тексту программы стандартный текст описаний для микроконтроллера Atmega8.

while

Оператор цикла. Форма написания команды whi1e очень похожа на форму описания функции. В общем случае команда while выглядит следующим образом:

while (условие){тело цикла}

Перевод английского слова whi1e — «пока». Эта команда организует цикл, многократно повторяя тело цикла до тех пор, пока выполняется «условие», то есть пока выражение в скобках является истинным. В языке СИ принято считать, что выражение истинно, если оно не равно нулю, и ложно, если равно.

Комментарии

В программе на языке СИ широко используются комментарии, которые не позволят вам запутаться при написании программы. Принято два способа написания комментариев.

Первый способ — использование специальных обозначений начала и конца комментария. Начало комментария помечается парой символов /*, а конец комментария символами */. Это выглядит следующим образом:

/* Комментарий */

Причем комментарий, выделенный таким образом, может занимать не одну, а несколько строк.

Второй способ написания комментария — двойная наклонная черта (//). В этом случае комментарий начинается сразу после двойной наклонной черты и заканчивается в конце текущей строки.

В дальнейшем наши программы будут начинаться с заголовка, в нашем случае это многострочный комментарий, который мастер поместил с информацией о том, что программа создана при помощи CodeWizardAVR. Также автоматически будут указаны такие параметры, как тип процессора, его тактовая частота частоту, модель памяти (mega — означает большая модель), размер используемой внешней памяти и размер стека. После комментариев будет начинаться вся основная часть программы, которую мы и будем учиться писать.

3. Моделирование схем с помощью программы Proteus

3.1. Описание

Proteus Professional представляет собой систему схемотехнического моделирования, базирующуюся на основе моделей электронных компонентов принятых в PSpice. Отличительной чертой пакета Proteus Professional является возможность моделирования работы программируемых устройств: микроконтроллеров, микропроцессоров, DSP и прочее. Дополнительно в пакет Proteus Professional входит система проектирования печатных плат. Proteus Professional может симулировать работу следующих микроконтроллеров: 8051, ARM7, AVR, Motorola, PIC, BasicStamp. Библиотека компонентов содержит справочные данные.

Поддерживает МК: PIC, 8051, AVR, HC11, ARM7/LPC2000 и другие распространенные процессоры. Более 6000 аналоговых и цифровых моделей устройств. Работает с большинством компилятором и ассемблерами.

PROTEUS VSM позволяет очень достоверно моделировать и отлаживать достаточно сложные устройства, в которых может содержаться несколько МК одновременно, и даже разные семейства в

одном устройстве. Нужно только ясно понимать, что моделирование электронной схемы не абсолютно точно повторяет работу реального устройства. Вместе с тем для отладки алгоритма работы МК, этого более чем достаточно. PROTEUS содержит огромную библиотеку электронных компонентов. Отсутствующие модели можно дополнить.

Если компонент не программируемый, то нужно на сайте производителя скачать его SPICE модель и добавить в подходящий корпус.

PROTEUS 7 состоит из двух основных модулей:

ISIS – графический редактор принципиальных схем служит для ввода разработанных проектов с последующей имитацией и передачей для разработки печатных плат в ARES. К тому же после отладки устройства можно сразу развести печатную плату в ARES, которая поддерживает авторазмещение и трассировку по уже существующей схеме.

ARES – графический редактор печатных плат со встроенным менеджером библиотек и автотрассировщиком ELECTRA, автоматической расстановкой компонентов на печатной плате.

PROTEUS имеет уникальные возможности.

USBCONN – этот инструмент позволяет подключиться к реальному USB порту компьютера.

COMPIM – этот компонент позволяет вашему виртуальному устройству подключиться к реальному COM-порту вашего ПК.

Примеры:

– вы можете подключить через "шнурок" к свободному COM-порту сотовый телефон и отлаживать устройство на МК, которое должно управлять им.

– вы можете подключить к COM-порту любое реальное устройство с которым ваш создаваемый прибор будет общаться в реальности!

PROTEUS VSM – великолепно работает с популярными компиляторами Си для МК:

– CodeVisionAVR (для МК AVR)
– IAR (для любых МК)
– ICC (дляМК AVR, msp430, ARM7, Motorola)

– WinAVR (для МК AVR)
– Keil (для МК архитектуры 8051 и ARM)
– HiTECH (для МК архитектуры 8051 и PIC от Microchip)

Для выполнения лабораторных работ необходимо смоделировать в Proteus лабораторный стенд. В данной работе была уже показана приблизительная модель стенда на рисунке 3.1. Теперь мы остановимся на этом месте, чтобы подробнее рассмотреть создание в Proteus схемы и её моделирования.

3.2. Работа с Proteus

Начнем с открытия программы, установку программы попросите у преподавателя. Нам нужно открыть программу с названием ISIS, путь к ней лежит через C:\Program Files\LabcenterElectronics\Proteus 7 Professional\BIN. После запуска ISIS.exe нам откроется рабочее окно, в котором мы собственно и будем создавать нашу схему (рис. 3.1).

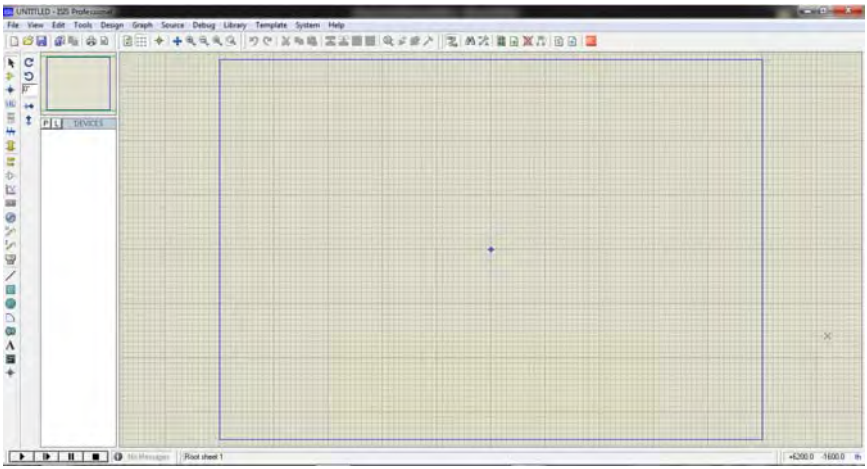

Рис. 3.1. Рабочее окно программы Proteus

Теперь нам необходимо попасть в библиотеку программы, в которой уже имеются все нужные элементы. В этой библиотеке собранны все возможные элементы, используемые в создании принципиальных схем. Чтобы вызвать меню библиотеки нужно найти кнопку слева Pick from Libraries (рис. 3.2).

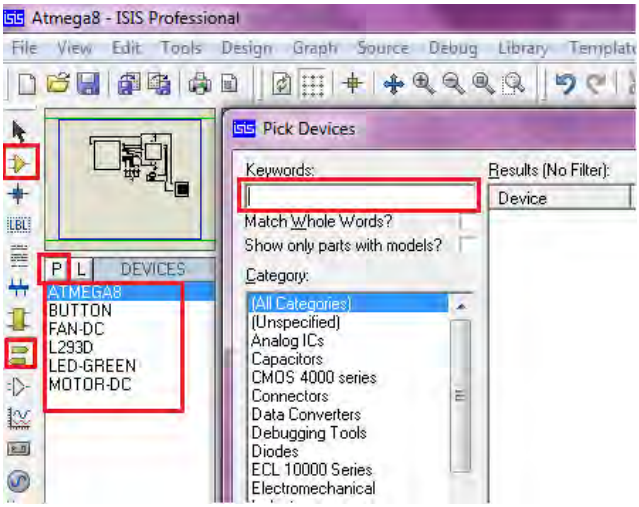

Рис. 3.2. Основные кнопки для создания схем:
1 – Pick from Libraries; 2 – Devices; 3 – Terminals Mode

В открывшемся окне есть строка Keywords, здесь мы вводим слово или часть слова и при нажатии Enter получаем все имеющиеся элементы в базе по введенному запросу. Так же можно найти необходимые элементы с помощью уже имеющихся тут категорий (Category). Справа мы можем увидеть схематический вид элемента, который будет применяется на схеме и натуральный вид со всеми размерами (этого вида может не быть). После выбора элемента кнопкой Ok или двойным щелчком, элемент условно помещается на конце вашего курсора-карандаша. Чтобы поместить элемент в нужную вам область, наведите туда курсор и кликните 1 раз. Так же программа сохраняет все выбранные элементы в данном проекте в специальном поле, с помощью которого можно миновав библиотеку

вставлять нужные нам элементы (рис. 3.2). Сверху также отображаются выбранные вами элементы. Для нашего стенда необходимы следующие:

1) ATMEGA 8. микроконтроллер
2) LED-GREEN. светодиоды для индикации с зеленым свечением
3) BUTTON. кнопка
4) MOTOR-DC. мотор с индикацией количества оборотов
5) L293D. драйвер

Так же нам нужно подключить питание и «землю» к схеме. Для этого найдем объект с названием Terminals Mode (рис. 3.2). В открытом поле находим Groundи Power. Устанавливаем их к выводам микросхем и подсоединяем «карандашом» соответствующие им контакты.

Все объекты в Proteus либо интеллектуально подсоединены на питание (если соответствующие выводы отсутствуют как, например, у МК ATMega8), либо нужно подсоединить (при наличии выводов у данного объекта). При этом достаточно просто соединить вывод Vcc электронного компонента с объектом Power (означает питание), а GNDc объектом Ground. Значения напряжения и тока для объекта выставляются автоматически при подключении выводов питания названных выше!

Далее все элементы на схеме соединяем проводами (в данном случае линиями). Для этого наводим на элемент курсор-карандаш, ищем места, специально отведенные для линий. В перекрестии курсора-карандаша появится маленький треугольник, зажимаем левую клавишу мыши и тянем линию к другому элементу, находим такой же место с маленьким треугольником и отпускаем клавишу. Аналогично соединяем все элементы.

Чтобы условно записать программу в виртуальный контроллер, в программе Proteus нужно дважды кликнуть по контроллеру. В открывшемся окне нужно указать путь к созданному вами hex-файлу в строке Program File, после нажать кнопку Ok (рис. 3.3).

Кроме того, в окне настройки виртуального МК есть поля, где следует установить параметры согласно рис. 3.3.

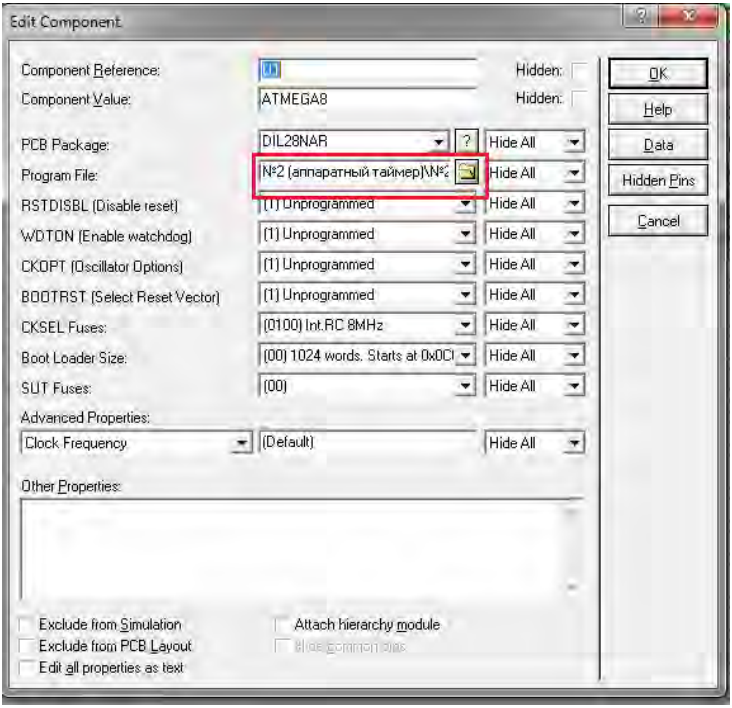

Рис. 3.3. Запись hex файла в МК

Управление смоделированной схемой производится с помощью кнопок [▶ ▮▶ ⏸ ■].

Если все сделано правильно, то при запуске модели вы увидите точно такую же работу МК, как и на реальном стенде.

Для корректного отображения скорости вращения двигателя в Proteus, необходимо выбрать соответствующие параметры используемого двигателя (рис. 3.4).

Выставляем все эти значения согласно рис. 3.4.

Для исследования изменяемых характеристик в Proteusможно использовать осциллограф (особенно понадобиться в следующих лабораторных работах). Она находится в Virtual Instruments Mode (рис. 3.5).

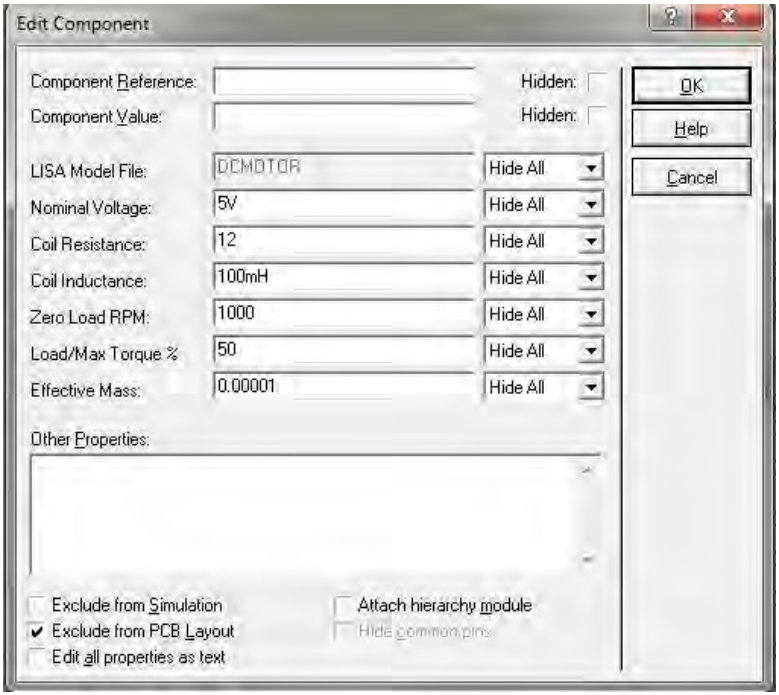

Рис. 3.4. Изменение параметров двигателя:
nominal voltage – номинальное напряжение; coil resistance – сопротивление катушки; coil inductance – катушки индуктивности;
Zero load – нулевая нагрузка; Torque – крутящий момент;
Effective mass – эффективная масса

Подключаем один из 4 каналов к нужной части схемы и получаем нужные нам характеристики. Обратите внимание на то, что осциллограф может выдавать преобразованные сигналы в зависимости от того, на какой вывод вы подключились (рисунок сигнала на картинке осциллографа в Proteuscоответствует получаемому сигналу). Вызов осциллографа происходит из вкладки Debug>Digital Oscilloscope в режиме симуляции.

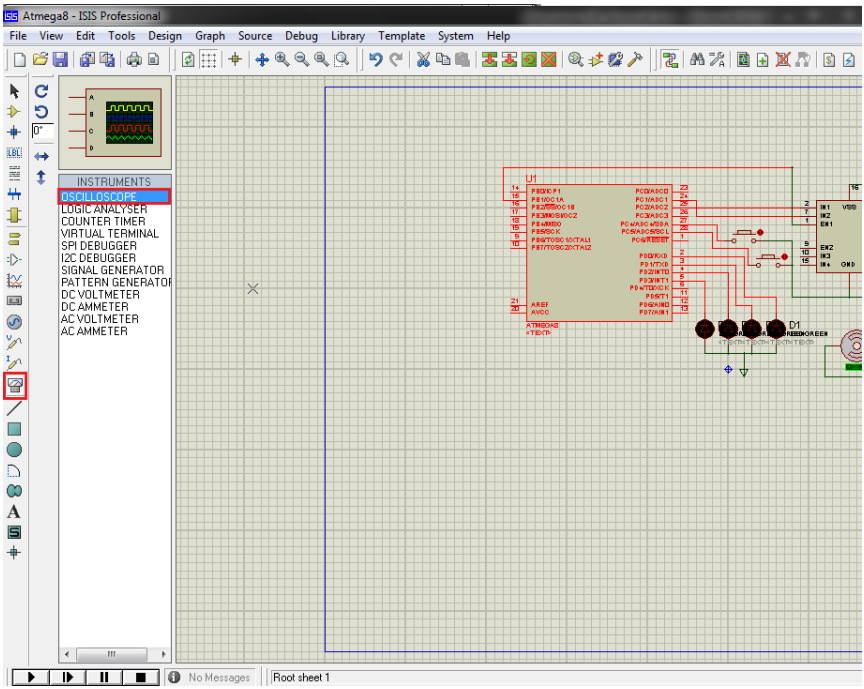

Рис. 3.5. Выбор осциллографа

4. Описание лабораторного стенда

Устройство МК ATMega8 мы изучили. Осталось разобраться в том, как правильно все подключить (рис. 4.1).

На рис. 4.1 изображено подключение драйвера к двигателю, сам двигатель, 2 кнопки и 4 светодиода, подключенные к микроконтроллеру. Светодиоды подключаются к МК через резисторы 220 Ом, из-за свойств пропускания тока диодами. Как все это работает, будет зависеть от того, как мы напишем программу. Осталось разобраться в назначении драйвера для двигателя L293D.

Для управления двигателями необходимо устройство, которое бы преобразовывало управляющие сигналы малой мощности в то-

ки, достаточные для управления моторами. Такое устройство называют драйвером двигателей.

L293D содержит сразу два драйвера для управления электродвигателями небольшой мощности (четыре независимых канала, объединенных в две пары). Имеет две пары входов для управляющих сигналов и две пары выходов для подключения электромоторов. Кроме того, у L293D есть два входа для включения каждого из драйверов. Эти входы используются для управления скоростью вращения электромоторов с помощью широтно-модулированного сигнала (ШИМ).

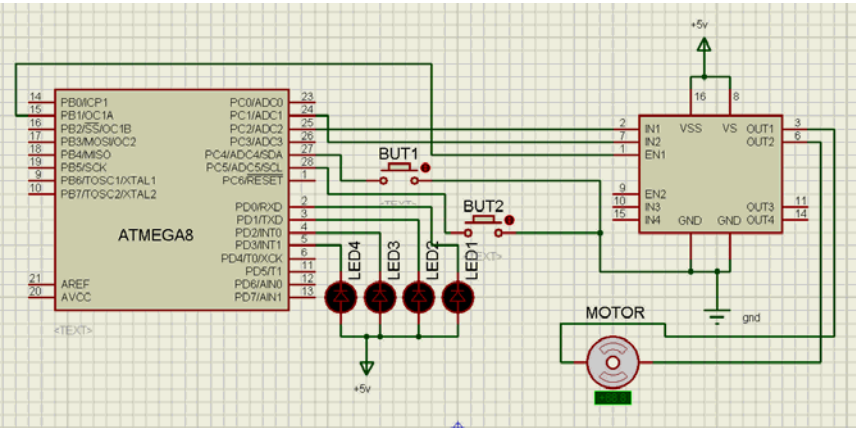

Рис. 4.1. Принципиальная схема лабораторного стенда

L293D обеспечивает разделение электропитания для микросхемы и для управляемых ею двигателей, что позволяет подключить электродвигатели с большим напряжением питания, чем у микросхемы. Разделение электропитания микросхем и электродвигателей может быть также необходимо для уменьшения помех, вызванных бросками напряжения, связанными с работой моторов.

Принцип работы каждого из драйверов, входящих в состав микросхемы, идентичен, поэтому рассмотрим принцип работы одного из них (рис. 4.2).

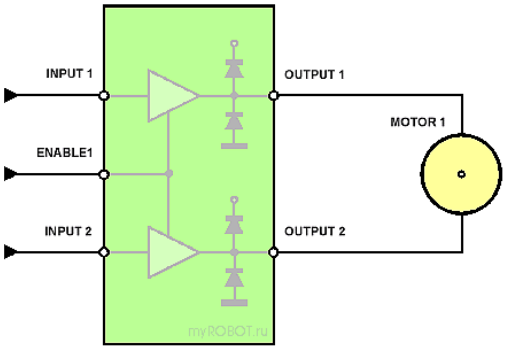

Рис. 4.2. Схема драйвера двигателей

К выходам OUTPUT1 и OUTPUT2 подключим электромотор MOTOR1.

На вход ENABLE1, включающий драйвер, подадим сигнал (соединим с положительным полюсом источника питания +5V). Если при этом на входы INPUT1 и INPUT2 не подаются сигналы, то мотор вращаться не будет.

Если вход INPUT1 соединить с положительным полюсом источника питания, а вход INPUT2 – с отрицательным, то мотор начнет вращаться.

Если соединить вход INPUT1 с отрицательным полюсом источника питания, а вход INPUT2 – с положительным. Мотор начнет вращаться в другую сторону.

Подадим сигналы одного уровня сразу на оба управляющих входа INPUT1 и INPUT2 (соединить оба входа с положительным полюсом источника питания или с отрицательным) – мотор вращаться не будет.

Если мы уберем сигнал с входа ENABLE1, то при любых вариантах наличия сигналов на входах INPUT1 и INPUT2 мотор вращаться не будет.

Представить лучше принцип работы драйвера двигателя можно, рассмотрев табл. 4.1.

Таблица 4.1

Логика работы драйвера

Enable	Input 1	Input 2	Output 1	Output 2
1	0	0	0	0
1	1	0	1	0
1	0	1	0	1
1	1	1	1	1

Теперь рассмотрим назначение выводов микросхемы L293D (рис. 4.3).

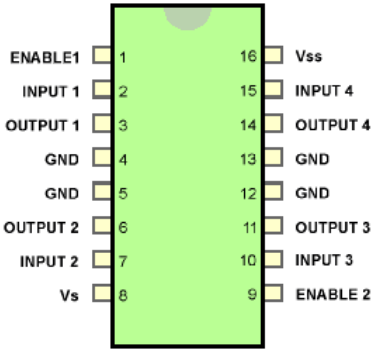

Рис. 4.3. Назначение выводов L293D

Входы ENABLE1 и ENABLE2 отвечают за включение каждого из драйверов, входящих в состав микросхемы.

Входы INPUT1 и INPUT2 управляют двигателем, подключенным к выходам OUTPUT1 и OUTPUT2.

Входы INPUT3 и INPUT4 управляют двигателем, подключенным к выходам OUTPUT3 и OUTPUT4.

Контакт Vs соединяют с положительным полюсом источника электропитания двигателей или просто с положительным полюсом

питания, если питание схемы и двигателей единое. Проще говоря, этот контакт отвечает за питание электродвигателей.

Контакт Vss соединяют с положительным полюсом источника питания (+5V). Этот контакт обеспечивает питание самой микросхемы.

Четыре контакта GND соединяют с "землей" (общим проводом или отрицательным полюсом источника питания). Кроме того, с помощью этих контактов обычно обеспечивают теплоотвод от микросхемы, поэтому их лучше всего распаивать на достаточно широкую контактную площадку.

Характеристики микросхемы L293D:

- напряжение питания двигателей (Vs) – 4,5...36V
- напряжение питания микросхемы (Vss) – 5V
- допустимый ток нагрузки – 600mA (на каждый канал)
- пиковый (максимальный) ток на выходе – 1,2A (на каждый канал)
- логический "0" входного напряжения до 1,5V
- логическая "1" входного напряжения – 2,3...7V
- скорость переключений до 5 kHz
- защита от перегрева.

На рис. 4.4 приведена схема простейшего LPT-программатора. Он состоит всего из 5 проводов, подключаемых к портам МК, поэтому он так и называется «Пять проводков».

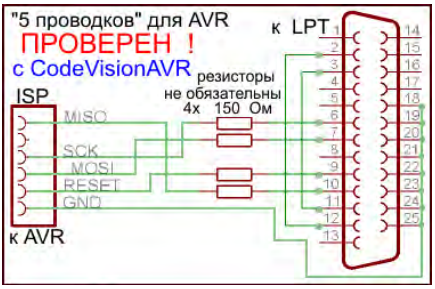

Рис. 4.4. Схема LPT-программатора

Резисторы в 150 Ом нужны для защиты LPT порта от тока, который выдает МК. Провода напрямую подключают к ножкам МК и можно программировать. Для надежности программирования ножки МК Reset и Vcc соединяют резистором в 10 кОм, Reset и GND– керамическим конденсатором на 0,1–0,015 мкФ. После сборки такого программатора при наличии LPT порта можно запрограммировать любой микроконтроллер AVR, что мы и научимся делать в лабораторных работах.

Недостатком LPT программаторов является их длина проводов – не больше 15 см, а если больше, то запись либо медленная, либо с ошибками. Единственный способ использовать длинные провода – это чередовать каждый сигнальный провод с землей или использовать качественный экранированный кабель.

Очень важно знать! При программировании МК, сам МК должен питаться от 5 Вольт.

Примечания к программатору

После того, как программа написана, осталось записать ее в МК. Для этого выполняем действия согласно рис. 4.5, 4.6 (если программатор LPT) или согласно рис. 4.7 (если программатор USB).

Программатор у нас серии STK 200+/300 (т.е. LPT). Если используется программатор USBAVR910, описание работы с ним смотрите ниже. Задержка между сигналами влияет на скорость записи и используется, если у программатора длинные провода. Используется порт LPT1. Настройку SCK Freq можно менять, она влияет на частоту строба, которая также может помочь, если у программатора длинные провода. Все остальные настройки выставить согласно рис. 4.5.

Часть рисунка подписана как «Не использовать» – это установка фьюзов и команда program all(запись всего, в том числе и фьюзов даже, если не стоит галочка Program Fuses). Фьюзы – особые настройки для изменения некоторой функциональности МК. Их перепрограммирование без особых знаний приведет к тому, что МК будет исправен, но вы ничего не сможете с ним сделать в силу, блокировки ножек отвечающих за последовательное программирование. Чтобы его восстановить, необходима дорогостоящая аппаратура (к примеру, параллельный программатор или реаниматор фьюзов)!!!!

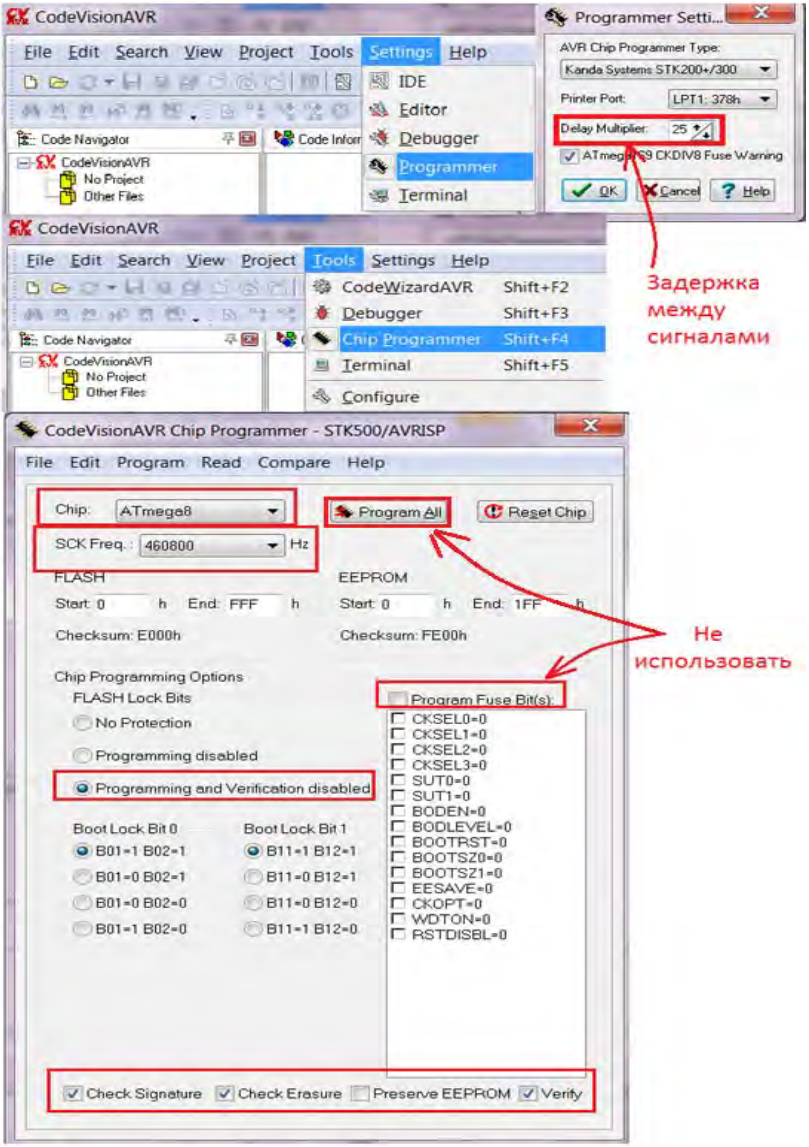

Рис. 4.5. Запись программы в МК

После того, как настроили программатор согласно рис. 4.5, можно приступить к программированию (рис. 4.6).

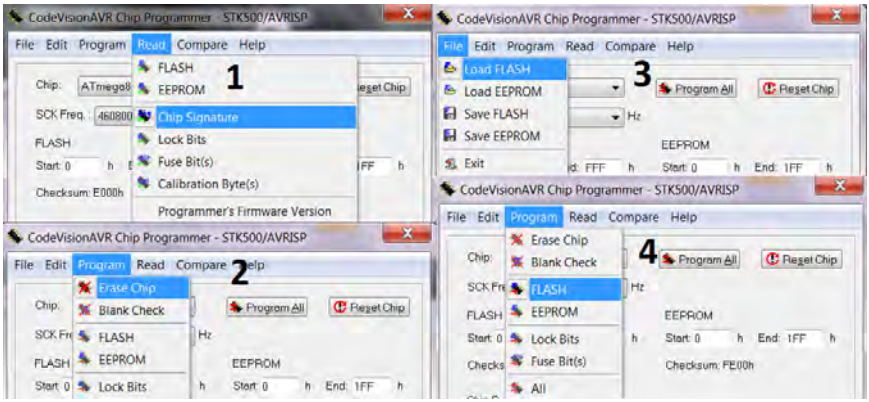

Рис. 4.6. Последовательность действий для записи программы в МК

Сначала проверяем сигнатуру чипа (действие 1). Если появилось окно с ошибкой, то стоит проверить подключение, в частности питание МК; попробуйте увеличить задержку в настройках программатора. Если ничего не выходит, то следует проверять всю схему целиком, а также работоспособность LPT порта у компьютера.

Если же получили окно, в котором программа увидела, что это ATmega8, то приступим к стиранию чипа (действие 2). Иногда появляется окно с ошибкой стирания – это может быть дефект из-за длины проводов программатора (следует опять же увеличить задержку) или сбой в самом МК (следует выбрать команду Blank-Check, которая исправит программный сбой).

После того, как стирание флэш-памяти чипа произошло успешно, можно записать в него новую программу. Для этого нужно загрузить ее в буфер обмена (действие 3), после команды Load FLASH в появившемся окне выбираем файл программы (имеет расширение hex в папке вашей программы) – встречали его при компиляции – это HEX-файл, который CodeVisionAVR скомпилировала для записи в МК.

Примечание: hex файл находится в папке Папка самой программы/Exe

Файл загрузили. Для записи выберем команду FLASH.Вы увидите окно прогресса, которой будет извещать о проценте записанной программы. После этого произойдет сравнение записанной программы, с программой в буфере обмена. Если все произошло успешно, и вы правильно написали программу, то увидите ее работу на стенде. Если же запись произошла с ошибкой, то повторите все заново, начиная со стирания.

МК рассчитан на 10 000 записей (может быть и меньше)!

Если программатор USB

Если в качестве программатора используется USBAVR910, то после установки его в USB ПК начнет искать драйвер для нового USB устройства. Следует отказаться от автоматической установки и указать путь к папке с драйвером согласно операционной системе (драйвер спросите у преподавателя). Если операционная система не предложила выбрать драйвер, а просто выдала сообщение, что таковой не установлен, то следует перейти в Диспетчер устройств, где вы увидите неопознанное устройство, которому необходимо обновить драйвер.

После установки у вас появиться новое устройство, которое имеет виртуальный COM-порт (рис. 4.7).

Далее нужно в CodeVisionAvr выбрать именно этот программатор – AtmelAVRProg (AVR910) – и указать номер порта.

CodeVision с USB программатором работает очень медленно (стирание происходит около минуты) из-за того, что он «не умеет» правильно пересылать пакеты данных по протоколу USB. Но существует другая программа, которая предназначена именно для этого программатора – AVRProg, – написанная самим производителем МК (ее тоже можно взять у преподавателя или скачать с сайтов). Это программа входит в пакет AVRStudio. Ее можно просто копировать на другие ПК без пакета AVRStudio.

AVRProg запуститься в случае, если она нашла программатор в списке устройств ПК. Если запуск не произошел, проверьте все контакты. Если она запустилась, то появиться окно, где человек, знающий английский язык, без труда поймет, как с ней обращаться (рис. 4.8).

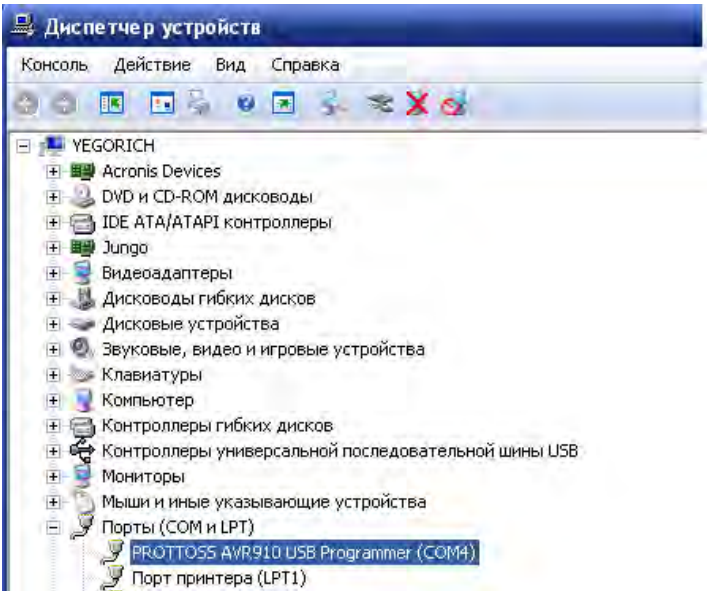

Рис. 4.7. Установка драйвера для программатора

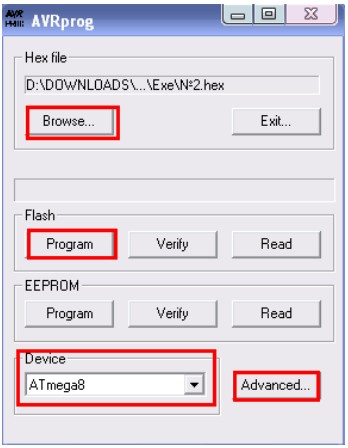

Рис. 4.8. Окно программы AVRprog

5. Управление электродвигателем

Лабораторная работа № 1

ВКЛЮЧЕНИЕ ДВИГАТЕЛЯ И СВЕТОДИОДА (ИНДИКАТОРА) НА ОПРЕДЕЛЕННЫЙ ПРОМЕЖУТОК ВРЕМЕНИ, ИСПОЛЬЗУЯ ПРОГРАММНЫЙ ТАЙМЕР. УСТРАНЕНИЕ ДРЕБЕЗГА КОНТАКТОВ

Цель работы:

Научиться использовать порты МК для включения двигателя на определенный промежуток времени, а также для включения светодиодов и реагирования на нажатие кнопки. На основе сведений приведенных ниже устранить дребезг контактов.

Общие сведения:

Рассмотрим структуру МК. Порт имеете три части (они же команды в языке C++):

- DDRx (регистр направления передачи данных) – определяет, является тот или иной вывод порта входам или выходом; если некоторый разряд регистра DDRx содержит логический 0, то соответствующий вывод порта сконфигурирован как вход, в противном случае – как выход;
- PORTx (регистр порта) – если вывод выполняет роль выхода, то в соответствующий разряд записывается значение, предназначенное для вывода; если вывод выполняет роль входа, то логический 0 в некотором разряде регистра PORTx соответствует высокоомный вход, а логическая 1 – вход, нагруженный подтягивающим сопротивлением;
- PINx (регистр выводов порта) – в отличие от регистров DDRx и PORTx доступен только для чтения и позволяет считать входные данные порта на внутреннюю шину мк.

В схеме используется кнопка, имеющая одну группу из двух нормально разомкнутых контактов. А если есть контакты, значит, есть и дребезг этих контактов. Теперь рассмотрим способ борьбы с дребезгом контактов программным путем.

Самый простой способ борьбы с дребезгом – введение в программу специальных задержек. Рассмотрим это подробнее. Начнем с исходно-

го состояния, когда контакты кнопки разомкнуты. Программа ожидает их замыкания. В момент замыкания возникает дребезг контактов.

Дребезг приводит к тому, что на соответствующем разряде порта PD вместо простого перехода с единицы в ноль мы получим серию импульсов. Для того, чтобы избавится от их паразитного влияния, программа должна сработать следующим образом. Обнаружив первый же нулевой уровень на входе, программа должна перейти в режим ожидания. В режиме ожидания программа приостанавливает все свои действия и просто отрабатывает задержку.

Время задержки должно быть выбрано таким образом, чтобы оно превышало время дребезга контактов. Такую же процедуру задержки нужно ввести в том месте программы, где она ожидает отпускания кнопки.

Ход работы:

Программирование контроллера AVR в среде CodeVisionAVR. После запуска программы разворачивается окно (рис. 5.1).

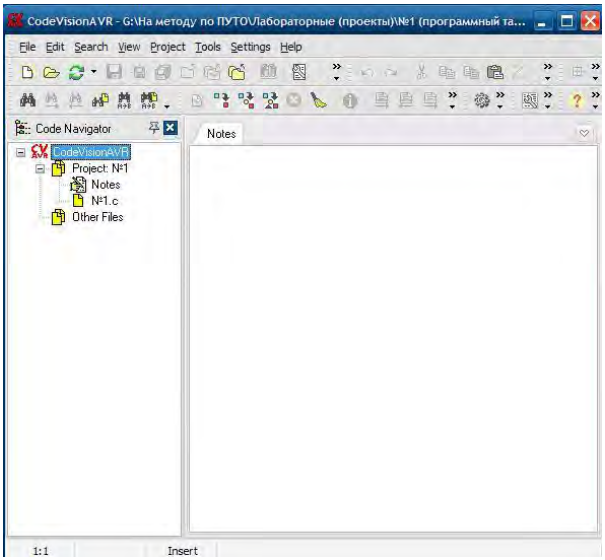

Рис. 5.1. Окно программы CodeVisionAVR

Далее открываем вкладку FILE, New (рис. 5.2).

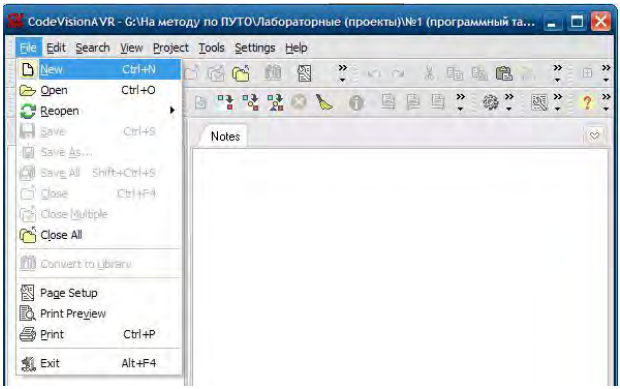

Рис. 5.2. Создание нового проекта

Выбираем Project (рис. 5.3).

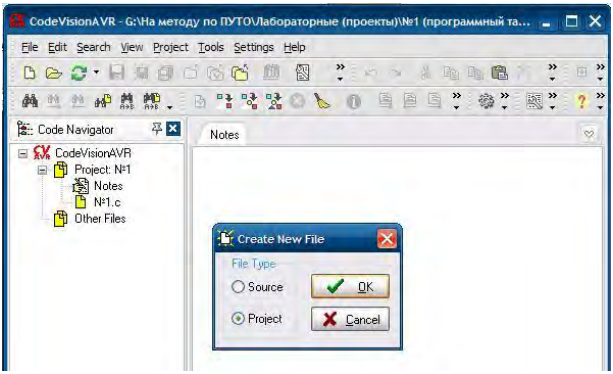

Рис. 5.3. Выбор проекта в CodeVisionAVR

Дальше спросят, будем ли использовать CodeWizard – соглашаемся (рис. 5.4).

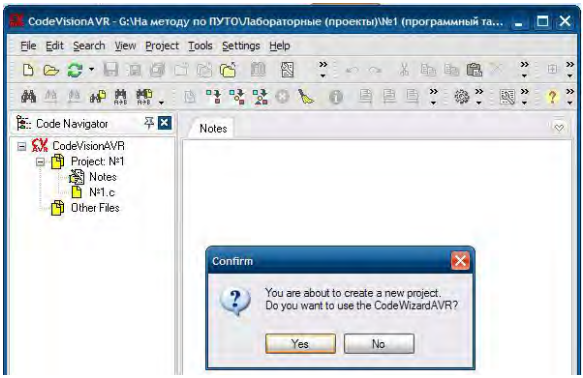

Рис. 5.4. Выбор мастера CodeWizardAVR

Выбираем в списке МК Chip:ATmega8, и ставим частоту на 8 MHz согласно рис. 5.5.

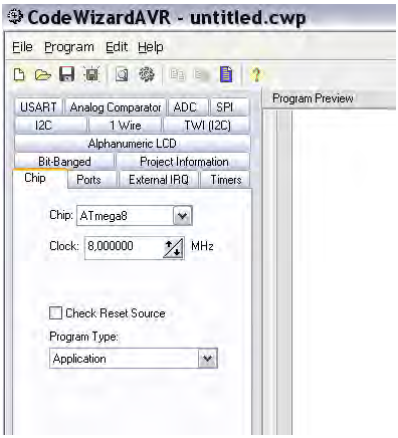

Рис. 5.5. Окно настройки параметров работы МК Atmega8

Следующим действием выбираем вкладки Ports:PortC, где настраиваем порты С на вход/выход согласно рис. 5.6.

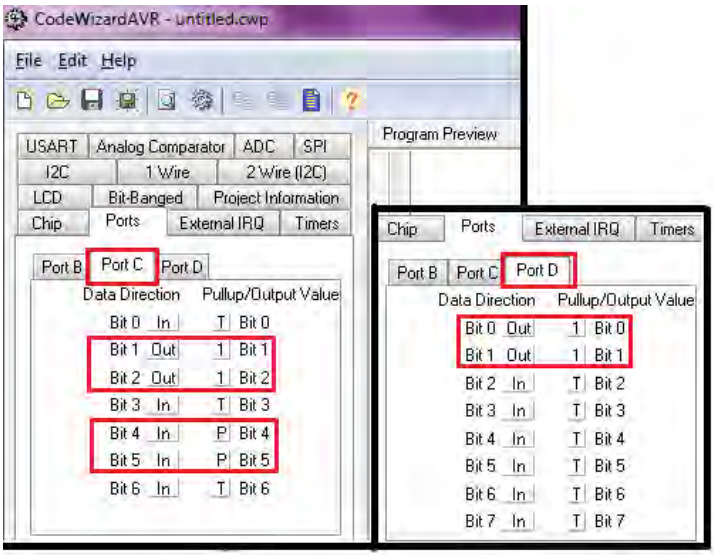

Рис. 5.6. Настройка портов входа/выхода

Это настройка требует пояснений. На выход в данной работе идет 2 порта С.1 и С.2 (соответственно в окне Wizardэто Bit 1 и Bit 2). Чтобы в начале программы выходы были выключены, необходимо выходное состояние поставить в логическую 1 (именно 1, а не 0 – так принято во всех МК в нашем мире). Точно также выставляем порты D.0 и D.1 для включения светодиода.

Порты С.4 и С.5 включаем на вход и включаем у них состояние P – это означает, что к этим портам подключили внутренний подтягивающий резистор для согласования уровней (так выставляется выход на кнопку). Соответственно состояние T – transitозначает отсутствие резистора на выходе.

После того, как все установлено, сохраняем заготовку проекта (рис. 5.7).

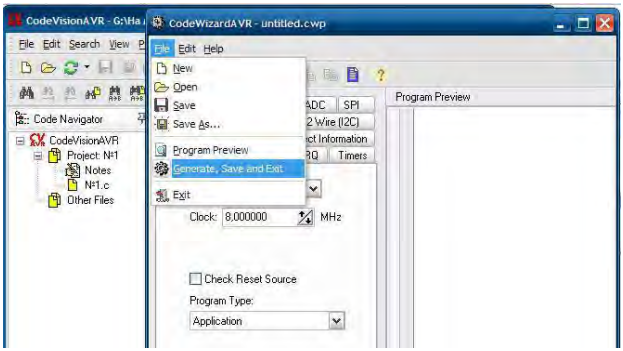

Рис. 5.7. Компиляция и сохранение проекта

Сохраняем файлы .c .prj .cwp под одним именем и желательно в отдельно созданную папку (рис. 5.8).

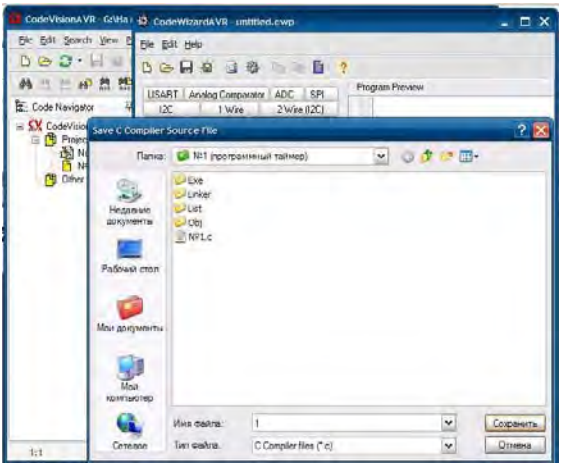

Рис. 5.8. Сохранение файлов программы

После сохранения открывается окно Си редактора (рис. 5.9).

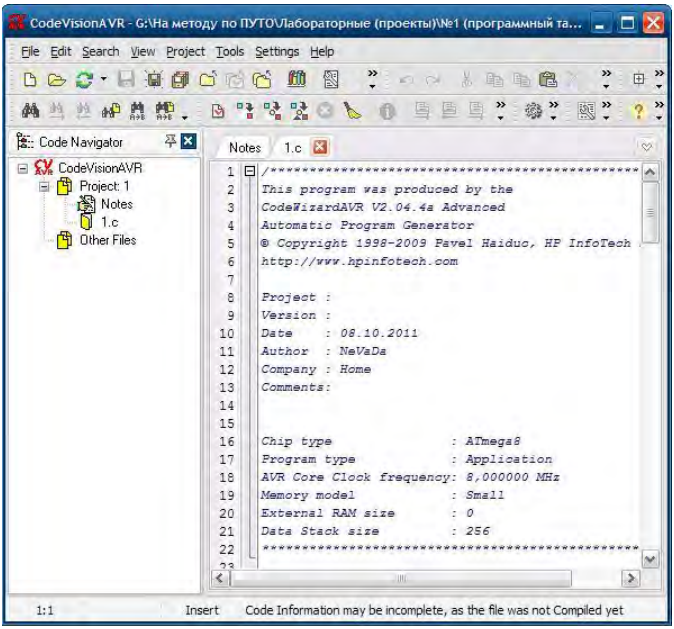

Рис. 5.9. Окно с листингом программы

Пишем здесь свою программу согласно заданию в цикле while(1). В конце лабораторной работы приведен весь листинг программы с пояснениями.

После того, как программа написана, необходимо ее проверить и сохранить в hex-файл (файл для записи в МК). Для этого компилируем исходный файл (Shift+F9) согласно рис. 5.10.

Нам важно, чтобы после компиляции появилось такое окно с надписями (без ошибок и предупреждений), как на рис. 5.11.

На рис. 5.11 в окне компиляции кроме сообщения об ошибках выводится информация о размере сгенерированного кода, проценте занятой памяти при записи на данный МК и прочее.

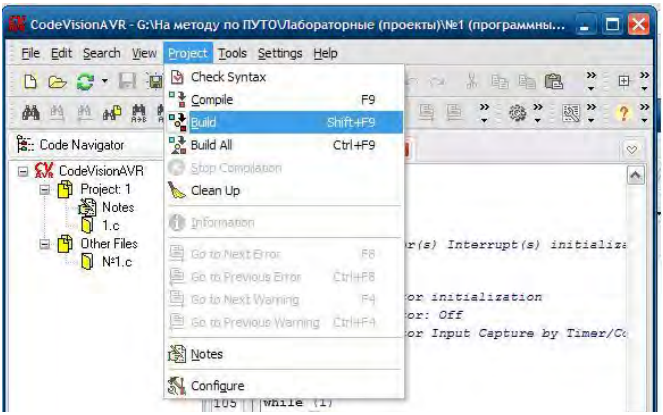

Рис. 5.10. Компиляция проекта

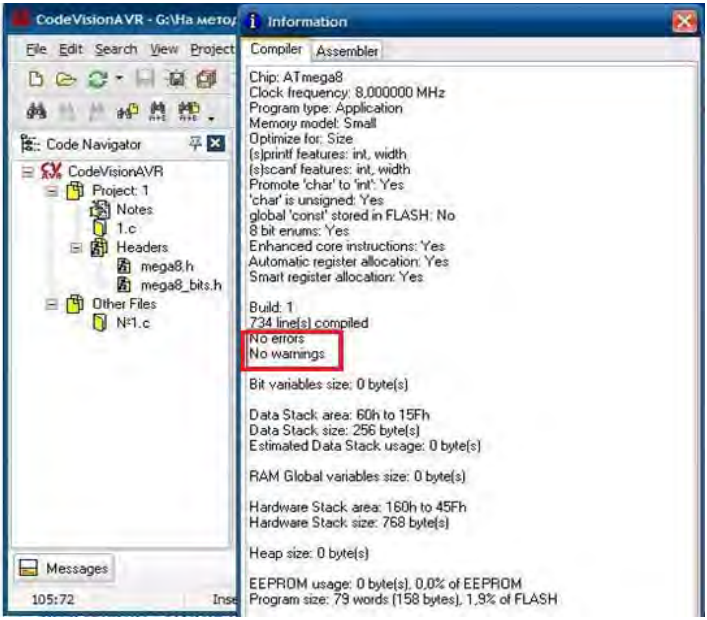

Рис. 5.11. Окно компиляции в CodeVisionAVR

Примечание: каждый раз, когда вы компилируете набранную программу в CodeVision, автоматически создается hex-файл, который размещен в папке Exe (она размещена в основной папке программы).

Листинг программы:

```
/*****************************************************
This program was produced by the
CodeWizardAVR V2.04.4a Advanced
Automatic Program Generator
© Copyright 1998-2009 PavelHaiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project :
Version :
Date    : 09.10.2011
Author  :NeVaDa
Company :
Comments:

Chip type           : ATmega8
Program type        : Application
AVR Core Clock frequency: 8,000000 MHz
Memory model        : Small
External RAM size   : 0
Data Stack size     : 256
*****************************************************/

#include<mega8.h>  //библиотека для МКAtmega8
(автоматически)
#include<delay.h>  // подключаем библиотеку задержки

// Declare your global variables here - приглашениеввестикодздесь

voidmain(void)  // начало главной части программы
```

```
{
// Declare your local variables here

// Input/Output Ports initialization - настройкапортоввввода/вывода
// Port B initialization   - настройкапортовB
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In
Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T
State0=T
PORTB=0x00;  //запись состояний портов типа "вкл/выкл" в 16-м
коде
DDRB=0x00;   //запись состояний типа "вход/выход" в 16-м коде

// Port C initialization  -настройкапортовC
// Func6=In Func5=In Func4=In Func3=In Func2=Out Func1=Out
Func0=In
// State6=T State5=P State4=P State3=T State2=1 State1=1 State0=T
PORTC=0x36;
DDRC=0x06;

// Port D initialization  -настройкапортов D
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=Out
Func1=Out Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=0 State1=0
State0=T
PORTD=0x03;  // аналогично для порта D
DDRD=0x03;

// Timer/Counter 0 initialization  -настрокатаймера 0
// Clock source: System Clock
// Clock value: Timer 0 Stopped
TCCR0=0x00;
TCNT0=0x00;

// Timer/Counter 1 initialization    - настройкатаймера 1
// Clock source: System Clock
```

```
    // Clock value: Timer1 Stopped
    // Mode: Normal top=FFFFh
    // OC1A output: Discon.
    // OC1B output: Discon.
    // Noise Canceler: Off
    // Input Capture on Falling Edge
    // Timer1 Overflow Interrupt: Off
    // Input Capture Interrupt: Off
    // Compare A Match Interrupt: Off
    // Compare B Match Interrupt: Off
    TCCR1A=0x00;
    TCCR1B=0x00;
    TCNT1H=0x00;
    TCNT1L=0x00;
    ICR1H=0x00;
    ICR1L=0x00;
    OCR1AH=0x00;
    OCR1AL=0x00;
    OCR1BH=0x00;
    OCR1BL=0x00;

    // Timer/Counter 2 initialization  -настройкатаймера  2
    // Clock source: System Clock
    // Clock value: Timer2 Stopped
    // Mode: Normal top=FFh
    // OC2 output: Disconnected
    ASSR=0x00;
    TCCR2=0x00;
    TCNT2=0x00;
    OCR2=0x00;

    // External Interrupt(s) initialization  -
настройкавнешнихпрерываний
    // INT0: Off
    // INT1: Off
    MCUCR=0x00;
```

```
// Timer(s)/Counter(s) Interrupt(s) initialization   - настройкамаски-прерываний
TIMSK=0x00;

// Analog Comparator initialization   - настройкааналоговогокомпаратора
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

PORTB.1=1;// при выключенном B.1 включается сигнал для работы двигателя
while (1)    //основной цикл программы, где и пишем код
      {
if (PINC.4==0){ //указываем действие при нажатии кнопки 1, подключенной к порту C.4
delay_ms(200); //задержка для устранения дребезга контактов
        PORTC.1=1;PORTC.2=0;  //перключение направления вращения двигателя
        PORTD.0=0; PORTD.1=1; // вкл. 1-й светодиод и выкл. 2-й
delay_ms(5000);PORTC.2=1;  PORTD.0=1;} // выключения светодиода и двигателя по истечении 5 секунд

if (PINC.5==0) { //аналогично для кнопки 2, подключенной к порту C.5
delay_ms(200);
        PORTC.2=1;PORTC.1=0;
        PORTD.0=1; PORTD.1=0;
delay_ms(7000);PORTC.1=1;  PORTD.1=1;} // выключения светодиода и двигателя по истечении 7 секунд
      };
}
```

Лабораторная работа № 2

ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО ТАЙМЕРА ДЛЯ ВКЛЮЧЕНИЯ ДВИГАТЕЛЯ НА ОПРЕДЕЛЕННЫЙ ПРОМЕЖУТОК ВРЕМЕНИ

Цель работы:

Изучить принцип работы программного таймера. Написать программу включения двигателя и светодиода, оповещающего о направлении вращения вала двигателя, на определенный промежуток времени.Выяснить преимущества программного таймера.

Общие сведения:

Таймер/Счетчик0 – это самый простой таймер или счетчик в семействе микроконтроллеров AT. У него нет дополнительных функций, он просто считает.

Почему называется «таймер/счетчик». На самом деле Таймер/счетчик – аппаратная часть МК, которая подсчитывает количество каких-либо импульсов. Таких импульсов может быть два вида:

1. Генератор МК;
2. Импульсы на ножке МК.

1. Генератор МК. МК работает с определенной частотой, которая определяется выбранным генератором (внутренним или внешним). Так вот, в этом режиме импульс для таймера подается от этого генератора с той же частотой. Именно этот режим работы таймера/счетчика и называется Таймером.

2. Импульсы на ножке МК. Значение таймера/счетчика изменяется в зависимости от состояния сигнала на определенной ножке. Т.е. считает количество импульсов на ножке. Этот режим называют Счетчиком.

Как же работает Таймер/Счетчик? В памяти МК есть часть отведенная для таймера, называется регистр TCNT0. Этот регистр, при запуске равен 0, при появлении импульса от генератора или ножки МК, значение регистра увеличивается на 1. Так как регистр TCNT0 восьмиразрядный, то его максимальное значение равно 0xFF. После

того как Таймер/Счетчик насчитал 256 импульсов (0xFF) он сбрасывается на 0 и начинает все заново.

Таймер/Счетчик0 при переполнении и сбросе регистра вызывает TCNT0 прерывание. С какой частотой будет выполняться прерывание не сложно посчитать. Для этого частоту работы МК надо разделить на 256 (максимальное значение таймера). К примеру, частота ATmega8 по умолчанию (заводские настройки) составляет 1 МГц, отсюда 1000000/256= 3906,25 раз в секунду будет вызываться прерывание. Это много! Для этого существуют предделители. Предделитель заставляет таймер реагировать не на каждый импульс, а через n раз.

В ATmega8 для Таймера/Счетчика0 можно устанавливать предделители 8, 64, 256, 1024. Таким образом таймер0 будет считать не все импульсы подряд, а только 8-ой, 64-ый, 256-ой и 1024-ый соответственно. Посчитаем на примере, 1000000/8/256= 488,28125 Герц. – это с предделителем 8. и 1000000/1024/256= 3,814697265625 в секунду с предделителем 1024.

Таким образом, зная частоту работы МК можно высчитывать частоту срабатывания прерываний. Такое прерывание называется «прерывание по переполнению таймера0».

Но что делать, если надо выставить конкретную частоту, которую нам не могут обеспечить предделители. Для этого существует «прерывание при совпадении с регистром (обычно A или B). В соответствующий регистр записываем любое число в зависимости от разрядности в 16-м коде, по достижении которого таймером, происходит выполнение какой-либо операции.

Одна трудность: предделитель нельзя использовать в режиме счетчика, только если таймер работает от генератора МК. Это плохо, но для подобных целей используются другие таймеры/счетчики.

У каждого МК есть определенное количество таймеров, например у Atmega8 существует только 3:

- Timer0 – регистров сравнения нету;
- Timer1 – два регистра сравнения (16-разрядных);
- Timer2 – один регистр сравнения (8-разрядный).

Ход работы:

Выберем МК и частоту его работы, как и прошлой работе. PORT C.1 и C.2ставим на выход (рис. 5.12). Кнопок в этой работе использовать не будем, ввиду некоторой сложности работы.

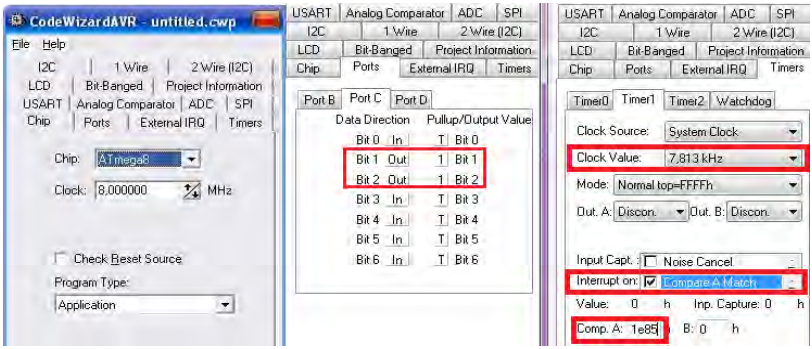

Рис. 5.12. Настройка портов и таймера МК при помощи Wizard

Кроме включения двигателя будем включать светодиоды, которые будут извещать о направлении вращения вала двигателя. Для этого порт D сконфигурируем так: BitD1- Out – 1; BitD3 - Out – 1;

В параметрах таймера выберем

Timer 1,

Clock Value 7,813 kHz, - частота счета

Interrupton: Compare A Match, - прерывание по совпадению с регистром А.

Comp. A = 1e85 (это число означает, что он будет тактироваться с частотой в 1 секунду, т.к мы выставили прерывание, когда таймер насчитает7813).

Генерируем код и сохраняем.

В листинге, кроме известных, появилась еще новая строка:

interrupt [TIM1_COMPA] void timer1_compa_isr(void) . Здесь надо вписать действие при прерывании таймера:

TCNT1H=0;

TCNT1L=0;

sec++;

Для корректной роботы таймера в эти регистры обязательно записать 0 (так таймер будет сбрасываться при достижении 7813 и считать заново). И еще в «тело» таймера добавили переменную, которую будем инкрементировать:sec++ (она и будет для нас задержкой в секундах).

Листинг программы (с пояснениями):

```
/*****************************************************
This program was produced by the
CodeWizardAVR V2.04.4a Advanced
Automatic Program Generator
© Copyright 1998-2009 PavelHaiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project :
Version :
Date    : 09.10.2011
Author  :NeVaDa
Company :
Comments:

Chip type               : ATmega8
Program type            : Application
AVR Core Clock frequency: 8,000000 MHz
Memory model            : Small
External RAM size       : 0
Data Stack size         : 256
*****************************************************/

#include <mega8.h>

unsigned int sec; // Объявляем переменную sec типом - беззнаковый целый

// Здесь указываем действие выполняемое при прерывания таймера
interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{
TCNT1H=0; // обязательно нужно выставить регистры в 0
TCNT1L=0; // их два, т.к. таймер 16-битный
sec++; // прибавляем к переменной sec 1.
}

// Declare your global variables here
```

```
    void main(void)
    {
    // Declare your local variables here

    // Input/Output Ports initialization
    // Port B initialization
    // Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In
Func1=In Func0=In
    // State7=T State6=T State5=T State4=T State3=T State2=T State1=T
State0=T
    PORTB=0x00;
    DDRB=0x00;

    // Port C initialization
    // Func6=In Func5=In Func4=In Func3=In Func2=Out Func1=Out
Func0=In
    // State6=T State5=P State4=P State3=T State2=1 State1=1 State0=T
    PORTC=0x36;
    DDRC=0x06;

    // Port D initialization
    // Func7=In Func6=In Func5=In Func4=In Func3=In Func2=Out
Func1=Out Func0=In
    // State7=T State6=T State5=T State4=T State3=T State2=1 State1=1
State0=T
    PORTD=0x06;
    DDRD=0x06;

    // Timer/Counter 0 initialization
    // Clock source: System Clock
    // Clock value: Timer 0 Stopped
    TCCR0=0x00;
    TCNT0=0x00;

    // Timer/Counter 1 initialization
    // Clock source: System Clock
    // Clock value: 7,813 kHz
    // Mode: Normal top=FFFFh
    // OC1A output: Discon.
```

```
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: On
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x05;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x1E; // это регистр сравнения A, при совпадении с которым таймер
OCR1AL=0x85; // выполняет описанное выше действие. В итоге получаем частоту 1 Гц.
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
MCUCR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x10;
// Analog Comparator initialization
```

```
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// Global enable interrupts
#asm("sei")

PORTB.1=1;// при выключенном B.1 включается сигнал для работы двигателя
while (1)
    {
if (sec==0){PORTC.1=0;PORTD.1=0;} // в начале времени вкл. порт C.1 - вращение вала двигателя + вкл. 1-го диода на 5 сек
if (sec==5){PORTC.1=1;PORTD.1=1;} // выключаем порт C.1 - остановка двигателя и выкл.1-го диода на 2 секунды
if (sec==7){PORTC.2=0;PORTD.3=0;} // вкл. порт C.2 - вращение вала двигателя в другую сторону + вкл.3-го диода на 7 сек
if (sec==14) {PORTC.2=1;PORTD.3=1; sec=0;} // выкл. порт C.2 - ост.двигателя + выкл.3-го диода, возврат в начало цикла
      };
}
```

Как видно из проделанной работы, для наших целей не очень удобно было использовать аппаратный таймер. Но именно аппаратный таймер позволяет разгрузить процессор, саму программу и позволяет более точно делать временные задержки. С использованием аппаратного таймера можно делать задержки на часы и даже на сутки и при этом получить погрешность от 1 сек (за час) до 18 секунд (за сутки). Эти погрешности связанны с тем, что в основе генератора МК положена работа RC-генератора, погрешность у которого ±3%. Чтобы сделать таймер/часы точнее нужно подключать к МК внешний кварцевый генератор.

Что же касается программного таймера, то точность у него очень мала (±20%). Поэтому его целесообразно использовать для кратковременного включения чего-либо.

Лабораторная работа № 3

ПРОГРАММНАЯ ШИРОТНО-ИМПУЛЬСНАЯ МОДУЛЯЦИЯ (ШИМ) ДЛЯ УПРАВЛЕНИЯ МОЩНОСТЬЮ ДВИГАТЕЛЯ

Цель работы:

Ознакомиться с принципом ШИМ. На основе полученных знаний написать программу, в которой при нажатии первой кнопки скорость вращения вала двигателя будет возрастать от min к max, а при нажатии второй кнопки скорость вращения вала двигателя будет снижаться. При этом диоды должны извещать о скорости вращения вала двигателя по закону:

– включен 1 диод – скорость 25% от max скорости вращения вала;
– включено 2 диода – скорость 50 %;
– включено 3 диода – скорость 75%;
– включено 4 диоды – скорость 100%.

Общие сведения:

Широтно-импульсная модуляция (в зарубежных источниках этот режим называется PWM – PulseWidthModulation) – способ задания аналогового сигнала цифровым методом, то есть из цифрового выхода, дающего только нули и единицы, позволяет получить плавно меняющиеся величины.

Представим себе мощный двигатель. Если включить его постоянно, то он раскрутится до максимального значения и так будет работать. Если выключить, то остановится через некоторое время, вследствие своей инерционности. А вот если двигатель включать на несколько секунд каждую минуту, то он раскрутится, но далеко не на полную скорость – большая инерция сгладит рывки от включающегося двигателя, а сопротивление от трения не даст ему крутиться бесконечно долго. Чем больше продолжительность включения двигателя в минуту, тем быстрей он будет крутиться.

При ШИМ подается на выход сигнал, состоящий из высоких и низких уровней (применимо к нашей аналогии – включаем и вы-

ключаем двигатель), то есть нулей и единицы. А затем это все пропускается через интегрирующую цепочку (в нашем случае драйвер L293D и масса якоря двигателя). В результате интегрирования на выходе будет величина напряжения, равная площади под импульсами.

Меняя скважность (отношение длительности периода к длительности импульса) можно плавно менять эту площадь, а значит и напряжение на выходе (рис. 5.13). Таким образом, если на выходе сплошные 1, то на выходе будет напряжение высокого уровня, в случае нашего двигателя, на выходе из моста L293 это 5 вольт, если нули, то ноль. А если 50% времени будет высокий уровень, а 50% низкий то 2,5 вольт. Интегрирующей цепочкой тут будет служить масса якоря двигателя, обладающего малой, но достаточной инерцией.

Рис. 5.13. Схема ШИМ сигнала

А если взять и подавать ШИМ сигнал не от нуля до максимума, а от минуса до плюса. Скажем от +12 до -12. А можно задавать переменный сигнал! Когда на входе ноль, то на выходе -12В, когда один, то +12В. Если скважность 50% то на выходе 0В. Если скважность менять по синусоидальному закону от максимума к минимуму, то получим переменное напряжение. А если взять три таких ШИМ генератора и подавать через них синусоиды сдвинутые на 120 градусов между собой, то получим самое обычное трехфазное напряжение, а значит сможем управлять бесколлекторными асин-

хронными и синхронными двигателями. На этом принципе построены все современные промышленные привода переменного тока типа Unidrive и OmronJxx.

В качестве сглаживающей интегрирующей цепи в ШИМ применена обычная RC цепочка (в нашем случае она включена в микросхему L293D) и масса якоря двигателя.

Существует аппаратная и программная ШИМ. Разница заключается в точности, сложности настройки, а также в том, что аппаратной ШИМ у МК ограниченное количество (обычно 1–4 канала), а программной ШИМ можно реализовывать столько, сколько ножек у МК.

В этой лабораторной работе научимся использовать программную ШИМ.

Для задания мощности двигателя выведем формулу:

$$P_{дв} = \frac{t_{импульса}}{t_{скважности}} \cdot 100\%,$$

где $P_{дв}$ – мощность двигателя от номинальной в процентах;

$t_{импульса}$ – длительность 1 импульса;

$t_{скважности}$ – длительность скважности.

Для того, чтобы задать программную ШИМ, достаточно включить нужный порт на время $= t_{импульса}$ и выключить на время $= t_{скважности}$, в зависимости от их соотношения и получим необходимую мощность двигателя.

Ход работы:

При помощи мастера CodeWizard (Shift+F2) выставить порты согласно табл. 5.1 или рис. 5.14.

Получить программу исходя из алгоритма: есть переменная, которая в зависимости от нажатий кнопок меняет свои значения, эти значения и будут определять мощность и включение светодиодов. В качестве таймера использовать команду delay, т.е. включать порт B.1, к примеру, на 1 мкс и выключать на 10 мкс для задания 10% от номинальной мощности. Написать программу согласно заданию!

Таблица 5.1

Настройка портов МК

Имя и номер порта	Вход/выход	Включен/выключен (в начале программы)
B.1	Выход	вкл.
C.1	Выход	выкл.
C.2	Вход	выкл.
C.4	Вход	вкл.
C.5	Выход	вкл.
D.0	Выход	выкл.
D.1	Выход	выкл.
D.2	Выход	выкл.
D.3	Выход	выкл.

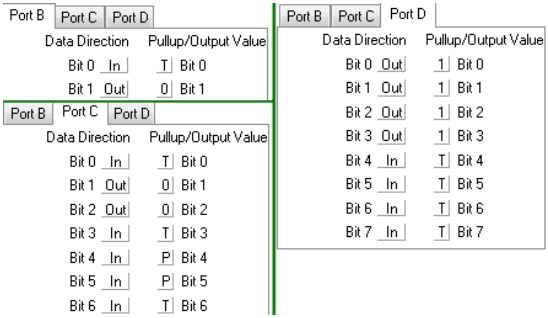

Рис. 5.14. Настройка портов МК

Листинг основной части программы:

```
#include<mega8.h>
#include<delay.h>
unsignedintvar; // описываем переменную типа безнаковый целый
……
……
PORTC.1=0; // прописываем направление вращения двигателя
```

```
var=1; // присваиваем начальное значение
while (1)
{
if (PINC.4==0 &var!=4) {delay_ms(200); var++;} // действиепринажатииикнопкиС.4
if (PINC.5==0 &var!=1) {delay_ms(200); var--;} // действиепринажатииикнопкиС.4
if (var==1) {PORTD=0x0E; PORTB.1=1; delay_ms(6);
                PORTB.1=0; delay_ms(24); }          // 25% отмахскорости
if (var==2) {PORTD=0x0C; PORTB.1=1; delay_ms(12);
                PORTB.1=0; delay_ms(24); }          // 50% отмахскорости
if (var==3) {PORTD=0x08; PORTB.1=1; delay_ms(18);
                PORTB.1=0; delay_ms(24); }          // 75% отмахскорости
if (var==4) {PORTB.1=1; PORTD=0x00;}    // max скорость
};}
```

При нажатии кнопки button1 увеличивается значение переменной var на 1, до значения 4 с задержкой в 200 мс (это ещё и для устранения дребезга контактов). При нажатии кнопки button2 уменьшается значение переменной var на 1, до значения 1 с задержкой в 200 мс. В итоге в зависимости от значения var порт B.1 включается и выключается на промежутки времени, соотношение которых и дает нам нужную мощность. Также значение var влияет на диоды включение, которых извещает нам о мощности.

Появилась одна тонкость: вместо того, чтобы при значении var=1 на включение светодиодов вместо записи

PORTD.0=0; PORTD.1=1; PORTD.2=1; PORTD.3=1; // включить 1 светодиод, подключенный к порту D.1

можно записать это в 16-м коде типа

PORTD=0x0E // в 2-м коде это 11110, где предпоследняя единица нужна из-за особенностей структуры МК, а остальные 1110 указывают как раз вкл/выкл портов по убыванию (в данном случае от D.3 до D.0соответственно).

Такой вариант записи сокращает программу, что может пригодится при написании программ большего размера, чем флэш-память МК. Для перевода величин использовался инженерный калькулятор в Windows, но есть ещё способ: выбрать вкл./выкл. портов в CodeWizard и затем скопировать нужные строки (рис. 5.15).

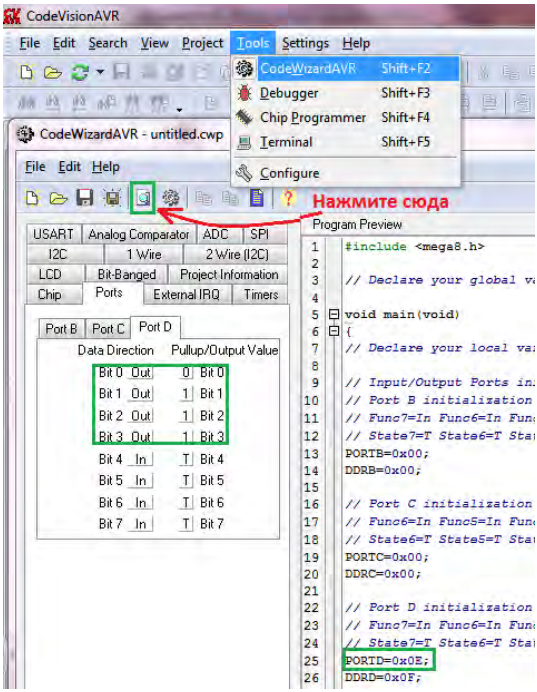

Рис. 5.15. Просмотр созданной программы в CodeWizard

Как видно из программы, скорость вращения вала двигателя будет переключаться мгновенно, а не плавно, как это можно сделать при помощи аппаратной ШИМ. Это является основным недостатком программной ШИМ.

Примечания к Proteus

Для наблюдения изменения сигнала воспользуйтесь осцилографом.

Лабораторная работа № 4

ИСПОЛЬЗОВАНИЕ АППАРАТНОЙ ШИМ ДЛЯ УПРАВЛЕНИЯ ДВИГАТЕЛЕМ

Цель работы:

Изучить принцип работы внутренних регистров МК и написать программу изменения мощности двигателя при помощи аппаратной ШИМ.

Общие сведения:

Аппаратную ШИМ проще всего сделать на ШИМ генераторе, который встроен в таймеры МК. В Atmega8 3 ШИМ канала (рис. 5.16):

- OC1A (16-битный);
- OC1B (16-битный);
- OC2 (8-битный).

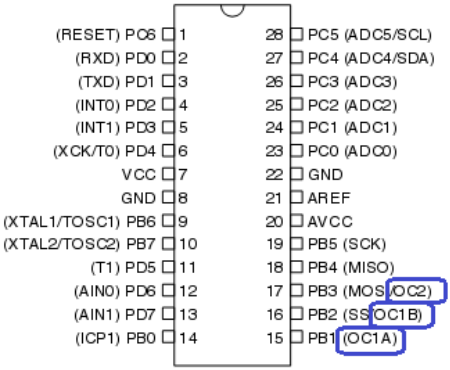

Рис. 5.16. ШИМ каналы МК Atmega8

У таймера есть особый регистр сравнения OCR**. Когда значение в счётном регистре таймера достигает значения находящегося в регистре сравнения, то могут возникнуть следующие аппаратные события:

- Прерывание по совпадению (уже использовали в аппаратном таймере)
- Изменение состояния внешнего выхода сравнения ОС**.

Предположим, что мы настроили наш ШИМ генератор так, что, когда значение в счетном регистре больше, чем в регистре сравнения, то на выходе у нас 1, а когда меньше, то 0. Таймер будет считать как ему и положено, от нуля до 256, с частотой, которую мы настроим битами предделителя таймера. После переполнения сбрасывается в 0 и продолжает заново. В итоге на выходе появляются импульсы. А если увеличить значение в регистре сравнения, то ширина импульсов станет уже.

У таймера может быть определенное количество регистров сравнения. Зависит от модели МК и типа таймера. Например, у Atmega8:

- Timer1 – два регистра сравнения (16-разрядных)
- Timer2 – один регистр сравнения (8-разрядный)

Самих режимов ШИМ существует несколько (рис. 5.17, 5.18):

1) Fast PWM (быстрый ШИМ)

В этом режиме счетчик считает от нуля до 255, после достижения переполнения сбрасывается в нуль и счет начинается снова. Когда значение в счетчике достигает значения регистра сравнения, то соответствующий ему вывод ОСххсбрасыватся в ноль. При обнулении счетчика этот вывод устанавливается в 1.

Частота получившегося ШИМ сигнала определяется просто: частота процесора 8МГц, таймер считает до 256 с тактовой частотой. Значит один период ШИМ будет равен 8 000 000/256 = 31250 Гц. Это максимальная скорость на внутреннем 8Мгц тактовом генераторе. Еще есть возможность повысить разрешение, сделав счет 8, 9, 10 разрядным (если разрядность таймера позволяет), но надо учитывать, что повышение разрядности, вместе с повышением дискретности выходного аналогового сигнала, резко снижает частоту ШИМ.

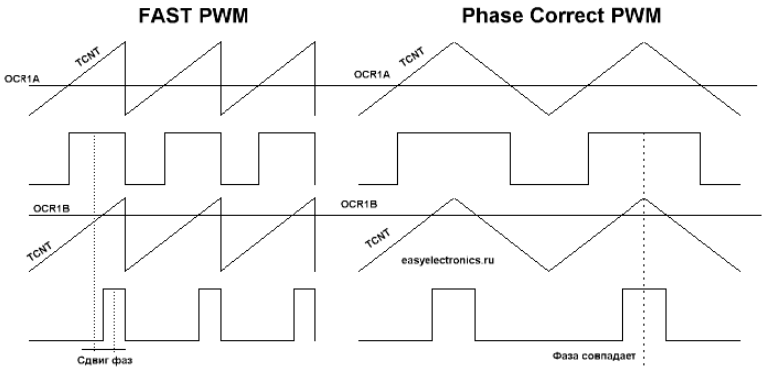

Рис. 5.17. Сравнение режимов Fast PWM
и Phase Correct PWM аппаратной ШИМ

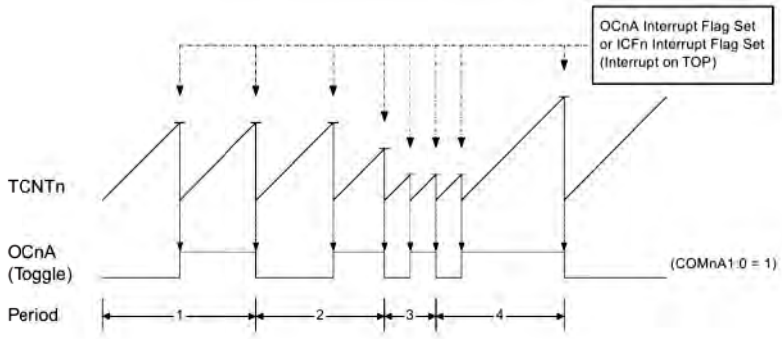

Рис. 5.18. Режим Clear timer on compare (TCNT) ШИМ МК Atmega8

2) PhaseCorrect PWM (ШИМ с точной фазой)

Работает также, но счетчик считает уже по-другому. Сначала от 0 до 255, потом от 255 до 0. Вывод OCxx при первом совпадении сбрасывается, при втором устанавливается.

Но частота ШИМ при этом падает вдвое, из-за большего периода. Основное его предназначение, делать многофазные ШИМ сигналы, например, трехфазную синусоиду. Чтобы при изменении

скважности не сбивался угол фазового сдвига между двумя ШИМ сигналами. Т.е. центры импульсов в разных каналах и на разной скважности будут совпадать. Чтобы не было кривых импульсов, то в регистр сравнения любое значение попадает через буферный регистр и заносится только тогда, когда значение в счетчике достигнет максимума. Т.е. к началу нового периода ШИМ импульса.

3) Clear Timer On Compare (Сброс при сравнении)

Это уже скорей ЧИМ – частотно-импульсномоделированный сигнал. Тут работает несколько иначе, чем при других режимах. Тут счетный таймер считает не от 0 до предела, а от 0 до регистра сравнения! А после чего сбрасывается.

В лабораторной работе вы научитесь использовать «быстрый ШИМ». Он относительно прост в настройке и удобен для того, чтобы снизить мощность двигателя.

Ход работы:

1) Создать проект при помощи CodeWizard. Выставить порты согласно табл. 3.1 или рис. 3.2 предыдущей лабораторной работы.

2) Во вкладке Timer1 настроить таймер на работу ШИМ. Для этого необходимо выставить режимы согласно рис. 5.19.

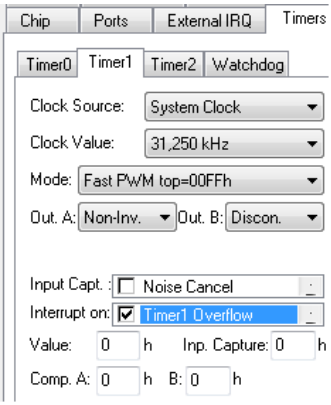

Рис. 5.19. Настройка таймера на режим Fast PWM аппаратной ШИМ

В итоге получаем таймер, который будет считать с частотой 31,25 кГц (как показала практика, чем меньше частота, тем лучше работает МК, также частота влияет на период импульса) и выдавать на выход А (т.е. порт В.1) не инвертирующий сигнал ШИМ (значение Non-inv). После того, как таймер досчитает до 255, он сбросится в ноль и будет считать заново (значение Interrupton:Timer1 Overflow).

3) Получить программу аналогичную программе лабораторной работы № 3, но уже с использованием аппаратного таймера.

Листинг основной части программы:

```
PORTC.1=0; // задаем направление вращения
while (1)
      {
  if((PINC.4==0)&(OCR1A!=255)) // увеличивать пока не достигло максиума
{  delay_ms(7); // задержка 7 мс (скорость наростания уровня сигнала)
      OCR1A++; //увеличиваем заполнение
  }
  if((PINC.5==0)&(OCR1A!=0)) // уменьшать пока не достигло миниума
  {
  delay_ms(7); // задержка 7 мс (скорость снижения уровня сигнала).
  OCR1A--; //уменьшаем заполнение
        };
if (OCR1A>=64) PORTD.0=0; else PORTD.0=1;  // вкл/выкл 1-й светодиод
if (OCR1A>=85) PORTD.1=0; else PORTD.1=1;  // вкл/выкл 2-йсветодиод
if (OCR1A>=128) PORTD.2=0; else PORTD.2=1; // вкл/выкл 3-йсветодиод
if (OCR1A>=254) PORTD.3=0; else PORTD.3=1; // вкл/выкл 4-йсветодиод
} }
```

При нажатии кнопки button1 ширина импульса ШИМ сигнала (OCR1A, поступающего на порт B.1) увеличивается со скоростью 1мкс за 7 мс до тех пор, пока не достигнет максимума (значения 255). При нажатии кнопки button2 возникает обратное действие до тех пор, пока ширина ШИМ сигнала не станет минимальной (значение 0), т.е. 7 мкс (это было установлено опытным путем).

Включение светодиодов делаем исходя из сравнения величины сигнала в регистре OCR1A с величинами, характеризиющими скорость вращения вала. Например, для 25% это будет величина равная

$$25\% \cdot 255 = 63{,}75 \approx 64.$$

Исходя из проделанных работ, можно выявить ещё одно преимущество аппаратной ШИМ – это ее плавность изменения, чего нельзя добиться программным путем.

Примечания к Proteus

Для наблюдения изменения сигнала воспользуйтесь осцилографом.

Список используемой литературы

1) Евстифеев, А. В. Микроконтроллеры AVR семейства Tiny и Mega фирмы “Atmel” / А. В. Евстифеев. – М. : Издательский дом «Додэка-XXI», 2004. – 560 с.

2) Мортон, Дж. Микроконтроллеры AVR. Вводный курс / Дж. Мортон ; пер. с англ. – М. : Издательский дом «Додэка-XXI», 2006. – 272 с. : ил. (Серия «Мировая электроника»).

3) Ревич, Ю. В. Практическое программирование микроконтроллеров AtmelAVR на языке ассемблера / Ю. В. Ревич. – СПб. : БХВ-Петербург, 2008. – 384 с. : (Аппаратные средства)

4) Программирование на языке С для AVR и PIC микроконтроллеров / сост. Ю. А. Шпак. – К. : «МК-Пресс», 2006. – 400 с., ил.

5) Лебедев, М. Б. CodeVisionAVR : пособие для начинающих / М. Б. Лебедев. – М. : «Додэка-XXI», 2008. – 592 с. : ил.

Оглавление

Для заметок

Учебное издание

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ

Лабораторный практикум
для студентов специальностей
1-53 01 01 «Автоматизация технологических процессов
и производств», 1-53 01 06 «Промышленные роботы
и робототехнические комплексы»

В 2 частях

Часть 1

Составители:
СИРОТИН Феликс Львович
КАПУСТИНА Анна Михайловна
АГЕЙЧИК Юрий Александрович
ГОЛУБЧИК Егор Васильевич

Технический редактор *О. В. Песенько*

Подписано в печать 12.02.2014. Формат $60\times84^{1}/_{16}$. Бумага офсетная. Ризография.
Усл. печ. л. 3,72. Уч.-изд. л. 2,91. Тираж 100. Заказ 643.
Издатель и полиграфическое исполнение: Белорусский национальный технический университет.
Свидетельство о государственной регистрации издателя, изготовителя, распространителя
печатных изданий № 1/173 от 12.02.2014. Пр. Независимости, 65. 220013, г. Минск.