

## Лабораторные работы №1, 2

Курс: «Микропроцессорная техника в мехатронике и робототехнике»

### Изучение программной среды Code Vision Avr для разработки программ на языке СИ для микроконтроллеров AVR

#### 1. Назначение и основные характеристики Code Vision AVR

Для создания программ на языке СИ целесообразно использовать программную среду CodeVisionAVR. Это среда специально предназначена для разработки программ на языке СИ для микроконтроллеров серии AVR.

Code Vision AVR разработана румынской фирмой «HP Infotech», специализирующейся на разработке программного обеспечения.

Инсталляционный пакет свободно распространяемой версии программы, рассчитанной на создание программ, результирующий код которых не превышает 4 Кбайта, можно найти в Интернете на официальном сайте фирмы «HP Infotech»: [http://www.hpinfotech.ro/cvavr\\_download.html](http://www.hpinfotech.ro/cvavr_download.html)

Code Vision AVR не имеет встроенных средств отладки программ и пользуется отладочными средствами системы AVR Studio, официальной среды разработки программ фирмы ATMEL, с которой хорошо интегрируется.

Система Code Vision AVR, работает не с программами, а с проектами.

Кроме работы с проектами, среда Code Vision AVR может проверять программу на наличие синтаксических ошибок, производить трансляцию программы и сохранение результатов трансляции в HEX-файле, а также производить расширенную трансляцию.

В процессе расширенной трансляции программы формируется не только HEX-файл, но и файл той же программы, переведенный на язык Ассемблер, а также специальный файл в COF-формате, предназначенный для передачи программы в AVR Studio для отладки. После создания и расширенной трансляции проекта его директория будет содержать файлы со следующими расширениями:

prj – файл проекта Code Vision AVR;

txt – файл комментариев. Это простой текстовый файл, который вы заполняете по своему усмотрению;

c – текст программы на языке СИ;

asm – текст программы на Ассемблере (сформирован Code Vision);

cof – формат для передачи программы в другие системы для отладки;

eep – содержимое EEPROM (формируется одновременно с HEX-файлом);

hex – результат трансляции программы;

inc – файл-дополнение к программе на Ассемблере с описанием всех зарезервированных ячеек и определением констант;

lst – листинг трансляции программы на Ассемблере;

map – распределение памяти микроконтроллера для всех переменных программы на СИ;

obj – объектный файл (промежуточный файл, используемый при трансляции);

rom – описание содержимого программной памяти (та же информация, что и в HEX-файле,

но в виде таблицы);

вес – еще одно дополнение к программе на Ассемблере, содержащее команды переопределения векторов прерываний;

swr – файл построителя проекта. Содержит все параметры, которые вы ввели в построитель (

Все перечисленные выше файлы имеют одинаковые имена, соответствующее имени проекта.

## 2. Интерфейс системы Code Vision AVR

Интерфейс программы Code Vision AVR показан на рис. 1. Центральное место занимает окно программ. В этом окне может быть открыто сразу несколько программных модулей. Для каждого модуля появляется отдельная вкладка. Кроме того, там же появляется вкладка «Notes» - окно комментариев к текущему проекту. Все вкладки окна программ, обладают свойствами текстового редактора. Поддерживаются функции выделения фрагментов, их перетаскивания, копирования, вставки, поиска, поиска и замены и т. д.

Остальные окна имеют вспомогательное значение. Для работы с программой их наличие не обязательно. Каждое из них может быть закрыто или переведено во всплывающий режим. Для этого в заголовке каждого окна есть соответствующие инструменты. Рассмотрим назначение вспомогательных окон.

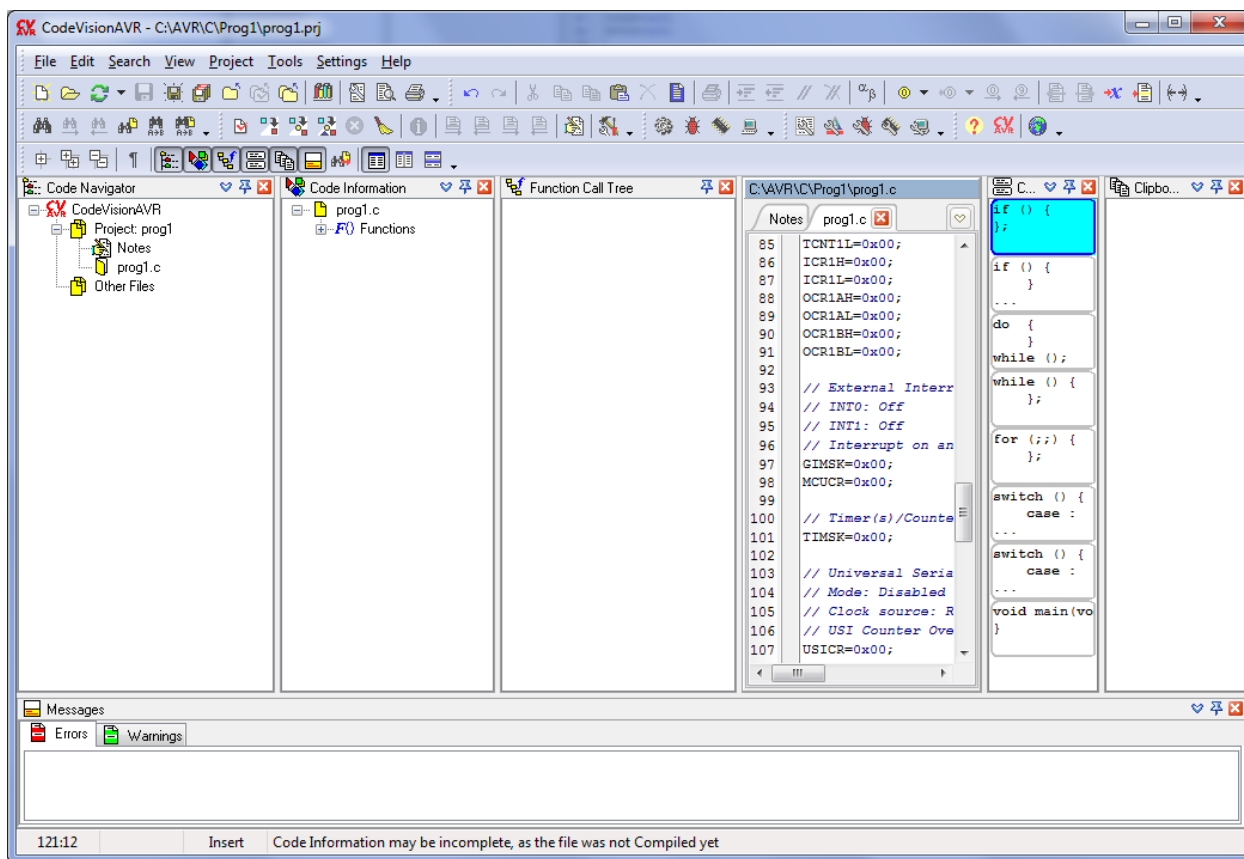


Рис. 1 Основное окно программы CodeVision

### **Окно «Code Navigator».**

В этом окне показывается структура текущего открытого проекта. Структура включает в себя список всех файлов, из которых состоит проект, а также список найденных ошибок и предупреждений, который появляется здесь после трансляции программы. В данном случае под файлами проекта понимаются только исходные файлы (тексты программ на языке СИ и файл описания).

### **Окно «Code Information».**

Содержит информацию о всех функциях программы и присоединенных библиотеках. Информация представлена в виде дерева. Присоединенные библиотеки вы найдете раскрыв ветвь «Includes», а дерево используемых функций, раскрыв «Functions». Содержимое всех этих ветвей появляется автоматически (формируется путем анализа текста программы). Такая структура очень удобна для навигации. Щелчок мышью по имени любой из функций в дереве проекта приведет к тому, что окно с текстом программы, содержащей эту функцию, переместится на передний план, и текстовый курсор установится в начало выбранной функции.

### **Окно «Function Call Tree».**

Показывает последовательность вложенных вызовов функций, которые интенсивно используют стек.

### **Окно «Code Templates» (Шаблоны Кода).**

Это окно – помощь программисту. Оно содержит шаблоны нескольких основных конструкций языка СИ. В частности, шаблоны операторов for, while, if и так далее. В любой момент программист может «перетащить» нужную структуру при помощи мыши в окно с основным текстом программы и заполнить полученный шаблон командами, и фрагмент программы готов. При желании вы можете пополнить набор шаблонов. Для того чтобы ввести новый элемент в список шаблонов, сначала наберите его текст в текстовом окне программы, выделите его и «перетяните» при помощи мыши в окно шаблонов. Записанный таким образом шаблон останется в окне шаблонов даже после загрузки нового проекта. С этого момента вы всегда можете использовать его в любой другой программе. Лишние шаблоны можно удалить.

Для этого достаточно щелкнуть по ненужному шаблону правой кнопкой мыши и в появившемся меню выбрать пункт «Delete».

### **Окно «Clipboard History» (История значений буфера обмена).**

Структура этого окна такая же, как структура окна «Code Templates». Но вместо шаблонов в окно в процессе редактирования помещается все, что попало в буфер обмена. Если вам снова понадобилось какое-либо из этих значений, вы в любой момент можете «перетянуть» его при помощи мышки в вашу программу.

### **Окно «Messages» (Ссообщения).**

Сюда выводятся все сообщения об ошибках и предупреждения в процессе трансляции. Вы можете быстро перейти к тому месту программы, где найдена ошибка. Для этого достаточно выполнить двойной щелчок мышкой по соответствующему сообщению об ошибке в окне.

## **3. Создание проекта без использования мастера**

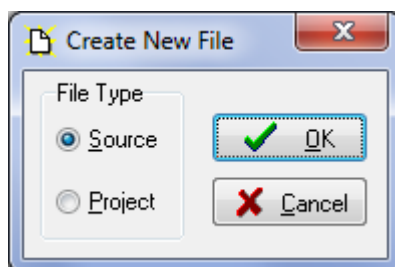
Программа Code Vision AVR так же, как и любая современная программа, работающая в среде

Windows, имеет строку меню и панель инструментов. При помощи меню можно управлять всеми функциями системы. Кнопки панели инструментов дублируют самые важные команды меню.

Рассмотрим случай создания проекта без использования мастера. Для того, чтобы начать процесс создания проекта, выберите пункт «New» меню «File» или нажмите соответствующую кнопку на панели инструментов. На экране появится окно «Create new file» (рис. 2). В окне предлагается выбрать вид создаваемого файла: исходный текст программы («Source») или файл проекта («Project»).


Сначала выберите пункт «Source» и нажмите кнопку «Ok». На экране появится текстовое окно. Внесите в это окно текст программы. Можно сначала внести лишь сокращенный вариант, если полного еще не существует. Теперь нужно записать набранный текст на жесткий диск. Для этого выберите пункт «File Save» меню «File» или нажмите соответствующую кнопку. Откроется диалог записи файла. Вы должны набрать имя файла, выбрать директорию для проекта и нажать кнопку «Сохранить».

Следующий этап – создание самого проекта. Снова выберите «New» меню «File» или нажмите соответствующую кнопку. Теперь вместо «Source» выберите «Project» (рис. 2). На вопрос, будете ли вы создавать новый проект при помощи построителя, ответьте «No». Откроется диалог записи файла. Вы должны выбрать имя для файла проекта и записать его в ту же директорию, что и файл с текстом программы. После нажатия кнопки «Сохранить» появляется окно проекта, в котором пока нет файлов с программой (рис. 3).



*Рис. 2 Создание проекта*

Для того, чтобы добавить созданный нами файл к нашему проекту, нажмите кнопку «Add». Откроется диалог, в котором вы должны найти и выбрать созданный ранее файл с программой. После нажатия кнопки «Открыть» файл присоединится к проекту. Теперь нажмите кнопку «Ok», и окно проекта закроется. С вновь созданным проектом вы можете работать так же, как и с проектами, созданными при помощи построителя.

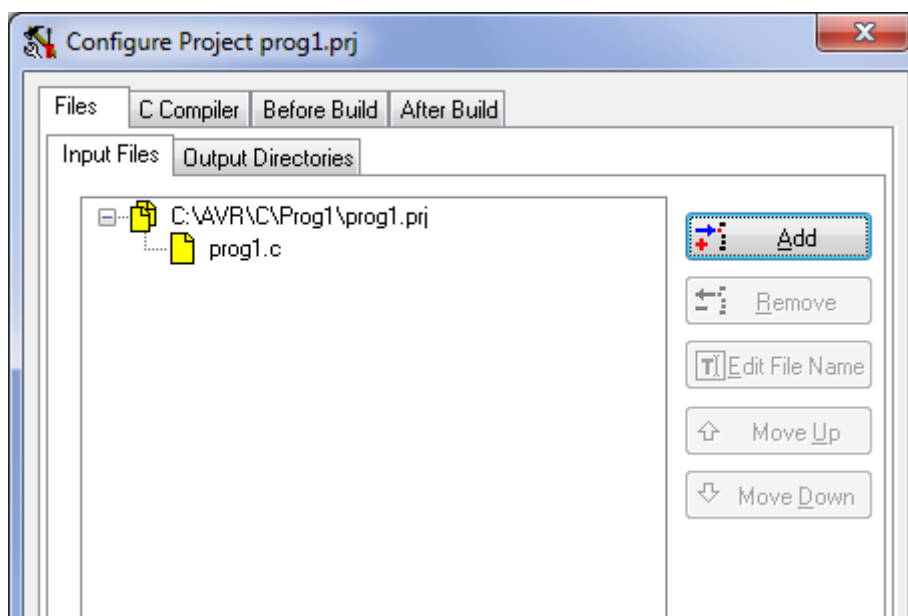
Перед тем, как приступить к трансляции проекта, необходимо настроить параметры компилятора. Снова открываем окно «Configure project». Для этого нажимаем кнопку . В окне проекта (рис. 3) открываем вкладку «C Compiler». На этой вкладке имеется множество параметров, определяющих стратегию компиляции. Установим только главные из них.

Сначала нужно выбрать тип микросхемы. Для этого служит выпадающее меню с заголовком «Chip:». Затем нужно выбрать частоту тактового генератора. Данные о частоте будут использованы транслятором при формировании процедур задержки. Выбор частоты производится при помощи другого выпадающего меню, озаглавленного «Clock:». На этом можно было бы и остановиться. Для остальных параметров можно оставить значения по умолчанию. Однако, при желании, вы можете

выбрать способ оптимизации. Для выбора способа оптимизации служит поле «Optimize for:» («Оптимизировать по:»). Предлагаются два способа:

- ♦ оптимизация по размеру («Size»);
- ♦ оптимизация по скорости («Speed»).

Оптимизация по размеру заставляет компилятор создавать результирующий код программы, минимальный по длине. Оптимизация по скорости позволяет создать более длинную, но зато более быстродействующую программу.



*Рис. 3 Окно конфигурации проекта*

После того, как все параметры установлены, запишите все эти изменения, нажав кнопку «Ok» в нижней части окна «Configure project». Окно проекта закроется. Теперь можно приступить к компиляции. Директивы режима компиляции приведены в табл. 1. Если программа достаточно большая, то перед компиляцией можно проверить ее на ошибки. Для этого служит директива «Проверка синтаксиса». При выборе директивы «Компиляция» проверка синтаксиса производится автоматически.

*Директивы работы с программой*

*Таблица 1*

Название	Пункт меню «Project»	Горячая клавиша	Описание
Проверка синтаксиса	Check Syntax	-	Проверить синтаксис программы в текущем окне
Компиляция	Compile	F9	Скомпилировать программу
Построить	Make	Shift+F9	Построить проект
Перестроить все	Make All Files	Ctrl+F9	Перестроить все файлы проекта
Конфигурация	Configure	-	Открыть окно конфигурации проекта

В процессе компиляции создается объектный файл в HEX-формате, а также файл, содержащий данные для EEPROM (файл с расширением eep). Директива «Построить проект» запускает процедуру полного построения проекта. Полное построение включает в себя проверку синтаксиса, компиляцию, создание файла программы на Ассемблере и файла в формате COF (для передачи в AVR Studio для отладки).

Учтите, что файл COF создается только в том случае, если выставлен соответствующий параметр в окне проекта (окно «Configure project», вкладка «C Compiler», параметр «File output format(s):»). По умолчанию значение этого параметра равно «COF ROM HEX EEP».

Кроме основных перечисленных выше директив, программа «Code Vision AVR» имеет несколько дополнительных (меню «Tools»). Некоторые из них перечислены в табл. 2. Директива «Запуск мастера» позволяет в любой момент в процессе редактирования программы запускать мастер-построитель программ. Это может понадобиться для редактирования созданной ранее заготовки программы.

Директива «Отладчик» предназначена для вызова внешней программы-отладчика. При первой попытке вызова этой директивы программа запрашивает путь к исполняемому файлу отладчика. Если указать путь к программе AVR Studio, то в дальнейшем директива «Отладчик» может использоваться для вызова этой программы.

*Дополнительные директивы Code Vision AVR*

*Таблица 2*

Название	Пункт меню «Tools»	Горячая клавиша	Описание
Запуск мастера	CodeWizardAVR	Shift+F2	Запуск мастера-построителя
Отладчик	Debugger	Shift+F3	Запуск внешней программы-отладчика
Программатор	Chip Programmer	Shift+F4	Запуск программатора микросхем

Особое значение имеет директива «Программатор». Дело в том, что программа Code Vision AVR имеет в своем составе систему управления программатором. Поддерживается несколько видов программаторов. Благодаря этому программа Code Vision AVR является полнофункциональной. То есть позволяет проводить полный цикл работ по разработке программ для микроконтроллеров.

Действительно, при помощи этой системы можно написать программу, оттранслировать ее и прошить в программную память микроконтроллера. Отсутствует лишь возможность отладки. Но она реализуется при помощи AVR Studio. Перед тем, как пользоваться программатором, необходимо настроить его параметры. Окно настройки параметров программатора открывается при выборе пункта «Programmer» из меню «Settings».

Прежде всего, нужно знать название платы программатора, которая подключена к вашему компьютеру. Кроме вида программатора, необходимо выбрать имя порта, к которому он подключен, а также некоторые специальные параметры.

#### **4. Отладка программы**

После того, как программа полностью оттранслирована, можно производить ее отладку. Для этого, не закрывая CodeVisionAVR, необходимо открыть программу AVR Studio. Затем в AVR Studio необходимо

выбрать пункт «Open File» в меню «File». Появится диалог открытия файла. В этом диалоге нужно изменить тип открываемого файла. В поле «Тип файла» нужно выбрать «Object Files (\*.hex; \*.d90; \*.a90; \*.r90; \*.obj; \*.cof; \*.dbg;)». Затем нужно найти на вашем диске директорию с проектом на CodeVisionAVR.

Когда вы откроете эту директорию, то вы увидите три файла с одинаковым именем. Чтобы узнать, какой из этих файлов имеет расширение «cof», нужно подвести к каждому файлу курсор мыши и посмотреть его описание во всплывающем желтом окне подсказки. Обычно это самый первый (верхний) из файлов. Выберите его и нажмите кнопку «Открыть». Сразу после этого откроется другой диалог. На этот раз диалог записи файла.

Программа предложит записать на диск файл проекта в формате AVR Studio. Имя этого файла уже будет дано по умолчанию. Просто нажмите кнопку «Сохранить», и файл проекта сохранится в той же директории, что и проект на СИ. После этого откроется знакомое нам окно выбора микросхемы и отладочной платформы. Выберите платформу «AVR Simulator», а в качестве микросхемы - ту, которую вы используете в своей программе. Нажмите кнопку «Finish».

После нажатия этой кнопки начнется процесс подготовки к режиму отладки. Как только он закончится, программа AVR Studio перейдет в отладочный режим. При этом в окне программ вы увидите текст программы на языке СИ, а содержимое остальных окон будет точно такое же, как и при отладке программ на Ассемблере.

В процессе отладки вы можете пользоваться всеми удобствами и преимуществами программы AVR Studio:

- ставить точки останова любого типа;
- просматривать и изменять содержимое всех регистров;
- запускать программу в пошаговом режиме и в режиме выполнения под управлением отладчика.

Если в процессе отладки обнаружится ошибка, исправлять ее нужно следующим образом:

- не закрывая AVR Studio, перейдите в окно CodeVisionAVR и измените текст программы;
- запишите изменения и перетранслируйте программу;
- вернитесь в AVR Studio, где вы увидите сообщение о том, что программа изменилась, и предложение учесть изменения;
- ответьте «Yes» и продолжайте отладку;
- по окончании отладки закройте программу AVR Studio.

## **5. Создание программы с использованием мастера-построителя**

Отличительной особенностью системы CodeVisionAVR является наличие мастера-построителя программы. Мастер-построитель облегчает работу программисту. Он избавляет от необходимости листать справочник и выискивать информацию о том, какой регистр за что отвечает и какие коды нужно в него записать. Результат работы мастера – это заготовка будущей программы, в которую включены все команды предварительной настройки, а также заготовки всех процедур языка СИ. Поэтому изучим работу мастера, создадим заготовку программы на языке СИ, дополним ее оригинальным кодом и получим, в

результате, готовую программу.

Проект CodeVisionAVR содержит дополнительную информацию, такую, как тип используемого процессора, тактовую частоту кварца и т. п. Эта информация необходима в процессе трансляции и отладки программ. За исключением текста программы, все остальные файлы проекта создаются и изменяются автоматически.

Задача программиста – написать текст программы, для которого в проекте отводится отдельный файл с расширением «с». Например, «proba.c». При помощи мастера мы будем создавать новый проект.

В качестве примера возьмем простейшую задачу, которая будет формулироваться следующим образом: «Разработать устройство управления одним светодиодным индикатором при помощи одной кнопки. При нажатии кнопки светодиод должен зажегаться, при отпускании – погаснуть. Кнопку подключить к порту PD.0, а светодиод к порту PB.1. Замыкание кнопки соответствует нулевому сигналу. Светодиод горит при нулевом сигнале».

Запустим CodeVisionAVR.

Для создания нового проекта выберем в меню «File» пункт «New». Откроется небольшой диалог, в котором мы должны выбрать тип создаваемого файла. Предлагается два варианта:

- «Source» (Исходный текст программы);
- «Project» (Проект). Выбираем Project и нажимаем «Ok».

Появляется окно с вопросом «You are about to create a new project. Do you want to use the CodeWizardAVR?». В переводе на русский это означает: «Вы создаете новый проект. Будете ли вы использовать построитель CodeWizardAVR?».

Выбираем «Yes», Выбираем нужный тип чипа (AT90, ATiny, ATmega, FPSLIC), после чего открывается окно построителя (рис. 4). Как видите, это окно имеет множество вкладок, каждая из которых содержит элементы выбора режимов.

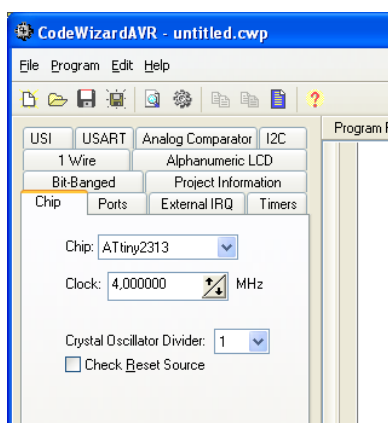


Рис. 4. Управляющая панель окна мастера CodeVisionAVR

Все эти управляющие элементы позволяют настроить параметры создаваемой заготовки программы. Сразу после запуска мастера все параметры принимают значения по умолчанию (все внутренние устройства выключены, а все порты ввода-вывода настроены на ввод, внутренние нагрузочные резисторы отключены). Это соответствует начальному состоянию микроконтроллера непосредственно после системного сброса.



Пройдемся немного по вкладкам мастера и выберем необходимые нам параметры. Те параметры, которые нам не нужны оставим без изменения со значениями по умолчанию.

Первая вкладка называется «Chip». На этой вкладке мы можем выбрать общие параметры проекта. Используя выпадающий список «Chip», выберем тип микроконтроллера. Для этого щелкаем мышью по окошку и в выпавшем списке выбираем ATtiny2313.

При помощи поля «Clock» выбираем частоту кварцевого резонатора. В нашем случае она должна быть равна 4 МГц. При помощи поля «Crystal Oscillator Divider» выбирается коэффициент деления тактового генератора. Коэффициент деления может изменяться от 1 до 256. Мы выберем его равным единице (без деления). То есть оставим значение по умолчанию.

Без изменений оставим флажок «Check Reset Source» (Проверка источника сигнала сброса). Включение данного флажка добавляет к создаваемой программе процедуру, связанную с определением источника сигнала системного сброса.

Покончив с общими настройками, перейдем к вкладке (Порты). Эта вкладка позволяет настроить все имеющиеся порты ввода-вывода. На вкладке «Ports» мы видим еще три вкладки поменьше (рис. 5). По одной вкладке для каждого порта. Как уже говорилось выше, порт PA в данной схеме мы не применяем. Поэтому сразу выбираем вкладку «Port B».

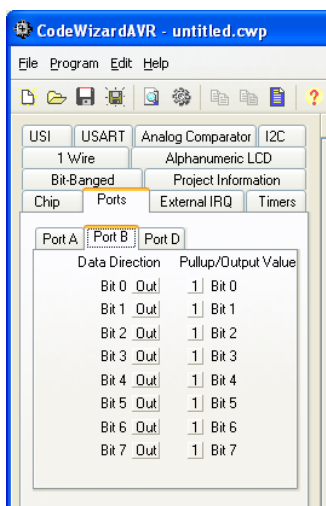


Рис. 5. Настройка порта PB

На вкладке мы видим два столбца с параметрами. Столбец «Data direction» (Направление передачи данных) позволяет настроить каждую линию порта либо на ввод, либо на вывод. По умолчанию каждый параметр имеет значение «In» (вход). Поменяем для каждого разряда это значение на «Out» (Выход).

Для того, чтобы поменять значение разряда, достаточно щелкнуть по полю с надписью «In» один раз мышью, и параметр сменится на «Out». Повторный щелчок заставит вернуться к «In». Каждое поле столбца «Data direction» определяет, какое значение будет присвоено соответствующему разряду регистра DDRB в нашей будущей программе.

Второй столбец на той же вкладке называется «Pullup/Output Value» (Включение нагрузки/Выходное значение). Этот столбец определяет, какое значение будет присвоено каждому из разрядов регистра PORTB. В нашем случае порт PB работает на вывод. Поэтому содержимое регистра PORTB определяет выходное значение всех разрядов порта. По умолчанию все они имеют значение

«0». Но по условиям нашей задачи они должны быть равны единице (при старте программы светодиод должен быть отключен). Поэтому изменим все нули на единицы. Для этого также достаточно простого щелчка мыши. В результате всех операций вкладка «Port B» будет выглядеть так, как это показано на рис. 5

Теперь перейдем к настройке последнего порта. Для этого выберем вкладку «Port D». На вкладке мы увидим такие же органы управления, как на вкладке «Port B» (см. рис. 6). По условиям задачи порт PD микроконтроллера должен работать на ввод. Поэтому состояние элементов первого столбца мы не меняем.

Однако нам нужно включить внутренние нагрузочные резисторы для каждого из входов. Для этого изменим значения элементов второго столбца. Так как порт PD работает в режиме ввода, элементы в столбце «Pullup/Output Value» принимают значение «Т» или «Р».

«Т» (Terminate) означает отсутствие внутренней нагрузки, а «Р» (Pull-up) означает: нагрузка включена. Включим нагрузку для каждого разряда порта PD, изменив при помощи мыши значение поля с «Т» на «Р». В результате элементы управления будут выглядеть так, как показано на рис. 6.

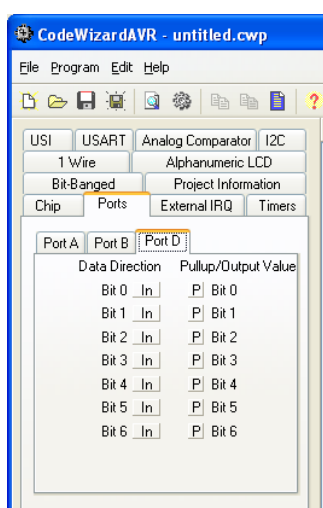
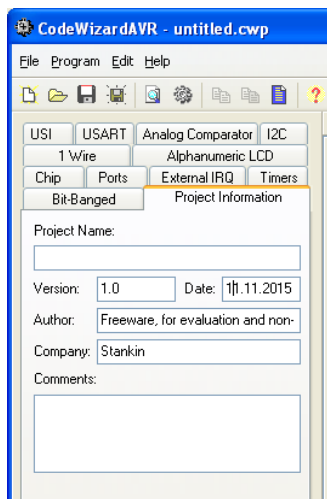


Рис.6 Настройка порта PD

Для данной простейшей программы настройки можно считать оконченными. Остальные системы микроконтроллера нам пока не нужны. Оставим их настройки без изменений.

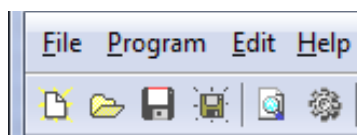
Воспользуемся еще одним полезным свойством мастера программ. Откроем вкладку «Project Information» (рис. 7).



*Рис. 7 Занесение информации без заголовка программы*

В поле «Project Name» вы можете занести название вашего проекта. Поле «Version» предназначено для номера версии. В поле «Date» помещают дату разработки программы. В полях «Author» и «Company» помещается, соответственно, имя автора и название компании. В поле «Comments:» можно поместить любые необходимые комментарии. Вся эта информация будет автоматически помещена в заголовок будущей программы.

После того, как все параметры выставлены, приступаем непосредственно к процессу генерации программы. Для этого выбираем в меню «File» мастера пункт «Generate, Save and Exit», или нажимаем кнопку с графическим изображением шестерни (рис. 8). Процесс генерации начнется с запроса имени файла будущей программы. Для этого откроется стандартный диалог сохранения файла, в котором вы сначала должны выбрать каталог, а затем указать имя файла программы. Что касается каталога, то нам нужно самостоятельно выбрать каталог, где будет размещен весь ваш проект.



*Рис. 8 Меню мастера генерации программы*

Рекомендуется для каждого проекта создавать свой отдельный каталог. Назовем каталог, например, именем «Prog1». Путь к каталогу может быть таким «C:\AVR\C\Prog1\». Если каталог еще не создан, то вы можете создать его прямо в диалоге создания файла. Файлу рекомендую присвоить то же самое имя: «Prog1».

Для завершения процесса создания файла нажмите кнопку «Сохранить». В результате на ваш жесткий диск запишется файл «Prog1.c», в который будет помещен созданный построителем текст заготовки вашей программы. Расширение «.c» набирать не нужно. Оно присваивается автоматически. Это стандартное расширение для программ на языке СИ.

Однако процесс генерации проекта на этом не заканчивается. Сразу после того, как файл

программы будет записан, откроется новый диалог, который запросит имя для файла проекта. Файл проекта предназначен для хранения параметров конкретного проекта.

Кроме типа используемой микросхемы и частоты задающего генератора, файл проекта используется для хранения вспомогательной информации, такой как наличие и расположение точек останова, закладок и т. д. В качестве имени файла проекта удобнее всего использовать то же самое имя, что и для текста программы. Файлу проекта автоматически присваивается расширение «.prj».

Когда файл проекта будет записан, диалог записи файла откроется в третий раз. Теперь нам предложат выбрать имя для файла данных построителя. В этот файл будут записаны текущие значения всех параметров, мастера. То есть значения всех управляющих элементов со всех его вкладок. И те, которые мы изменили в процессе настройки, и те, которые остались без изменений (по умолчанию).

Эти данные могут понадобиться, если потребуется заново пересоздать проект. Используя файл данных построителя, вы можете в любой момент восстановить значения всех его элементов, немного подкорректировать их и создать новый проект. Файлу данных построителя присвоим то же самое имя, что обоим предыдущим («Progl»). Новый файл получит расширение «.cwp» (Code Wizard Project). После того, как и этот файл будет записан, процесс генерации проекта завершается. На экране появляются два новых окна. В одном окне открывается содержимое файла «Progl.c». Второе окно – это файл комментариев. Сюда вы можете записать, а затем сохранить на диске любые замечания по вашей программе. В дальнейшем они всегда будут у вас перед глазами.

Макет программы состоит из настроек, инициализации и основного цикла программы. Основной цикл - это обязательный элемент любой программы для микроконтроллера. Поэтому мастер всегда создает заготовку этого цикла. То есть создает цикл, тело которого пока не содержит не одной команды.

В том месте, где программист должен расположить команды, составляющие тело этого цикла, мастер помещает специальное сообщение, приглашающее вставить туда код программы. Оно гласит: Place your code here (Пожалуйста, вставьте ваш код). Мы последовали этому приглашению и вставили требуемый код. В нашем случае он состоит всего из одной команды. Эта команда присваивает регистру порта PB (PORTB) значение входных уровней порта PB (PIND): PORTB=PIND.

Наша программа на языке СИ готова. Выполняясь многократно в бесконечном цикле, команда присвоения постоянно переносит содержимое порта PD в порт PB, реализуя тем самым наш алгоритм.

### **Работа программы, написанной на языке СИ**

Теперь разберемся, как программа работает. Программа на языке СИ сильно абстрагирована от системы команд микроконтроллера. Основные операторы языка СИ вовсе не привязаны к командам микроконтроллера. Для реализации всего одной команды на языке СИ на самом деле используется не одна, а несколько команд микроконтроллера. Иногда даже целая небольшая программа.

В результате облегчается труд программиста, так как он теперь работает с более крупными категориями. Ему не приходится вдаваться в мелкие подробности, и он может сосредоточиться на главном. Язык СИ так же, как и другие языки программирования, состоит из команд. Для записи каждой команды СИ использует свои операторы и псевдооператоры. Но форма написания команд в программе приближена к форме, принятой в математике.

В языке СИ для хранения различных данных используются переменные. Понятие «переменная» в языке СИ аналогично одноименному математическому понятию. Каждая переменная имеет свое имя, и ей можно присваивать различные значения. Используя переменные, можно строить различные выражения. Каждое выражения представляет собой одну или несколько переменных и числовых констант, связанных арифметическими и (или) логическими операциями. Например:

- $a*b$  – произведение переменных  $a$  и  $b$  (символ  $*$  означает умножение);
- $kl/2$  – переменная  $kl$ , деленная на два (символ  $\langle / \rangle$  означает деление);
- $massal + massa2$  – сумма двух переменных ( $massal$  и  $massa2$ );
- $tkon \ll 2$  – циклический сдвиг содержимого переменной  $tkon$  на 2 разряда влево;
- $dat \& mask$  – логическое умножение (операция «И») между двумя переменными ( $dat$  и  $mask$ ).

Приведенные примеры – это простейшие выражения, каждое из которых состоит всего из двух членов. Язык СИ допускает выражения практически любой сложности.

В языке СИ переменные делятся на типы. Переменная каждого типа может принимать значения из одного определенного диапазона (табл. 3). 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. 3).

В языке СИ любая переменная, прежде чем она 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 ее типа необходимы потому, что оттранслированная с языка СИ программа выделяет для хранения значений каждой переменной определенные ресурсы памяти.

*Типы данных языка СИ*

*Таблица 3*

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	$\pm 1.175e-38$ — $\pm 3.402e38$
double	32	$\pm 1.175e-38$ — $\pm 3.402e38$

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. Косвенно о количестве выделяемых ячеек можно судить по содержимому графы «Количество бит» табл. 3. 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. Принцип формирования имен в языке СИ не отличается от подобного принципа в языке Ассемблер. 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. Функция в языке СИ – это аналог соответствующего математического понятия. Функция получает одно или несколько значений в качестве параметров, производит над ними некие вычисления и возвращает результат.

Правда, в отличие от математических функций, функции языка СИ не всегда имеют входные значения и даже не обязательно возвращают результат.

Определение функции на языке СИ происходит по следующей схеме:

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 обязательно указывается 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.

Тело функции – это набор операторов на языке СИ, выполняющих некие действия. В конце каждого оператора ставится точка с запятой.

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, в начале 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.

Переменные, объявленные внутри функций, называются локальными. При написании программ иногда необходим другой порядок использования переменных. Иногда нужны переменные, которые могут работать сразу со всеми функциями. Такие переменные называются глобальными переменными.

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.

Обратимся к созданной программе и опишем используемые там команды и операторы.

*include*

*Оператор присоединения внешних файлов.* Данный оператор присоединяет к основному тексту программы стандартный текст описаний для микроконтроллера ATtiny2313.

*while*

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

while (условие) {тело цикла};

Перевод английского слова while — «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. Тело цикла — это ряд любых операторов языка СИ. Как и любая другая команда, вся конструкция while должна заканчиваться символом «точка с запятой».

В нашей программе в качестве условия в этом операторе используется просто число 1. Так как 1 — не ноль, то такое условие всегда истинно. Это значит, что цикл будет выполняться бесконечно. Тело цикла составляет единственная команда присваивания.

### **Комментарии**

В программе на языке СИ так же, как и в Ассемблере, широко используются комментарии. В языке СИ 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. В нашем примере шапка программы выполнена в виде комментария, который записан именно таким образом. Второй способ написания комментария — двойная наклонная черта (//).

В этом случае комментарий начинается сразу после двойной наклонной черты и заканчивается в конце текущей строки.

### **Описание программы**

Текст нашей программы, в основном, сформирован автоматически. Большую часть программы занимает функция `main`. Вся программа снабжена комментариями, которые также сформированы автоматически.

Начинается программа с заголовка. В начале заголовка мастер поместил информацию о том, что программа создана при помощи `CodeWizardAVR`. Далее в заголовок включен блок информации из вкладки «Project Information». Далее заголовок сообщает тип процессора, его тактовую частоту, модель памяти, размер используемой внешней памяти и размер стека.

После команды `include` мастер поместил сообщение для программиста. Сообщение предупреждает о том, что именно в этом месте программисту нужно поместить описание всех глобальных переменных. В данном случае глобальные переменные не нужны.

Теперь перейдем к функции `main`. Функция `main` содержит в себе набор команд, настройки системы и заготовку главного цикла программы. Под настройкой системы понимается запись требуемых значений во все управляющие регистры микроконтроллера.

Мастер-построитель программы присваивает значения всем без исключения служебным регистрам. И тем, значения которых должны отличаться от значений по умолчанию, и тем, значения которых не изменяются.

В последнем случае регистру присваивается то же самое значение, какое он и так имеет после системного сброса. Такие, на первый взгляд, избыточные действия имеют свой смысл. Они гарантируют правильную работу программы в том случае, если в результате ошибки в программе управление будет передано на ее начало.

Лишние команды при желании можно убрать. В нашем случае достаточно оставить лишь команды инициализации портов и команду инициализации компаратора.

Теперь посмотрим, как же происходит присвоение значений. Регистру `CLKPR` присваивается значение `0x80`. Для присвоения значения используется хорошо знакомый нам символ «`=`» (равно). В языке СИ такой символ называется оператором присвоения. Таким же самым образом присваиваются значения и всем остальным регистрам.

После инициализации всех регистров начинается основной цикл программы. Основной цикл – это обязательный элемент любой программы для микроконтроллера. Поэтому мастер всегда создает заготовку этого цикла. То есть создает цикл, тело которого пока не содержит не одной команды.

### **6. Порядок выполнения работы**

Во время выполнения работы необходимо выполнить следующее:

1. Изучить работу программной среды `CodeVisionAVR` согласно приведенному выше описанию.
2. С помощью мастера построителя создать программу управления одним светодиодом с помощью одной кнопки, проверить на отсутствие ошибок, провести трансляцию и продемонстрировать результат преподавателю.
3. С помощью программы `AVR Studio` провести отладку созданной программ. Результат продемонстрировать преподавателю.



4. Создать и отладить программу на языке СИ в программной среде Code Vision AVR согласно заданию. Результат показать преподавателю.
5. Составить отчет и защитить работу.

## 7. Задание

Вариант 1. Разработать устройство управления одним светодиодным индикатором при помощи одной кнопки. При каждом нажатии кнопки светодиод должен поочередно включаться и отключаться. При первом нажатии кнопки светодиод должен включиться, при следующем отключиться и т. д.

Вариант 2. Разработать схему управления светодиодом при помощи одной кнопки. При нажатии кнопки светодиод должен изменять свое состояние на противоположное (включен или выключен). При разработке программы принять меры для борьбы сдребезгом контактов.

Вариант 3. Создать устройство с одним светодиодом и одной управляющей кнопкой. Кнопка должна включать и выключать мигание светодиода. Пока кнопка отпущена, светодиод не должен светиться. Все время, пока кнопка нажата, светодиод должен мигать с частотой 5 Гц.

Вариант 4. Разработать автомат «Бегущие огни» для управления составной гирляндой из восьми отдельных гирлянд. Устройство должно обеспечивать «движение» огня в двух разных направлениях. Переключение направления «движения» должно осуществляться при помощи кнопки. Процедура задержки должна формироваться с помощью вложенных циклов.

Вариант 5. Разработать автомат «Бегущие огни» для управления составной гирляндой из восьми отдельных гирлянд. Устройство должно обеспечивать «движение» огня в двух разных направлениях. Переключение направления «движения» должно осуществляться при помощи кнопки. Процедура задержки  $\approx 200\text{мс}$  должна формироваться с помощью таймеров/счетчиков и не использовать прерывания.