ЛАБОРАТОРНАЯ РАБОТА №1.
Разработка микропроцессорного устройства управления светодиодными индикаторами

Цель. Изучение организации памяти и базовой архитектуры МК. Составление принципиальной схемы, разработка алгоритма работы устройства, конфигурирование портов ввода/вывода контроллера, написание программы для МК предложенного устройства.

Задача. Разработать устройство управления светодиодным индикатором при помощи кнопки.

ПРЕДВАРИТЕЛЬНЫЕ СВЕДЕНИЯ

Выбор микроконтроллера

По постановке задачи в разрабатываемом устройстве при нажатии кнопки светодиод должен зажечься, при отпускании – погаснуть. Для этого к микроконтроллеру нам нужно подключить светодиод и кнопку управления. То есть, нужен микроконтроллер, который имеет не менее двух портов. Данным условиям удовлетворяют микроконтроллеры ATtiny2313, ATMega8, ATmega16, ATmega32 и др.

ATmega8 сочетает в себе функциональность, компактность и сравнительно не высокую цену. Такие качества дали широчайшее распространение ATmega8 (рис.1) среди профессиональных и любительских конструкций. Микроконтроллер имеет широкий набор модулей, и может быть использован в большом количестве устройств.

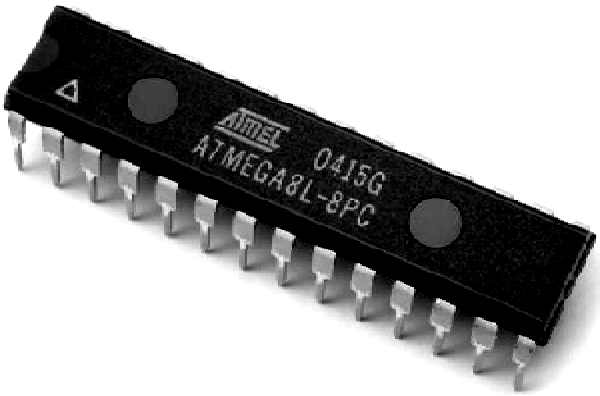

Рис.1. Микроконтроллер ATMega8 в корпусе PDIP

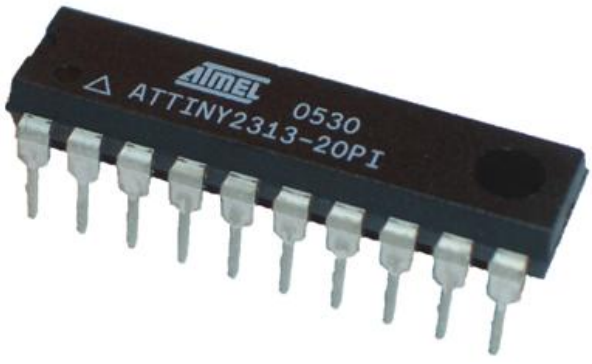

Рис.2. Микроконтроллер ATtiny2313 в корпусе PDIP

Микроконтроллер ATmega8 имеет два полноценных порта с разрядностью 8 бит в отличии от ATtiny2313 (рис.2), младшего собрата. Наличие в ATmega8 аналогово-цифрового преобразователя, дающего возможность измерять такие параметры как напряжение, ток, емкость что позволяет разработать полноценный мультиметр на базе этого микроконтроллера. Так же ATmega8 имеет порт UART для приема и передачи данных TTL уровня.

Порт для работы по протоколу TWI (возможность реализовать программный I^2C). По I^2C к ATmega8 можно подключить целый спектр устройств:

- внешнюю EEPROM память серии 24cXX,
- ЖКИ индикаторы и графические дисплеи,
- регуляторы громкости, сопротивления.

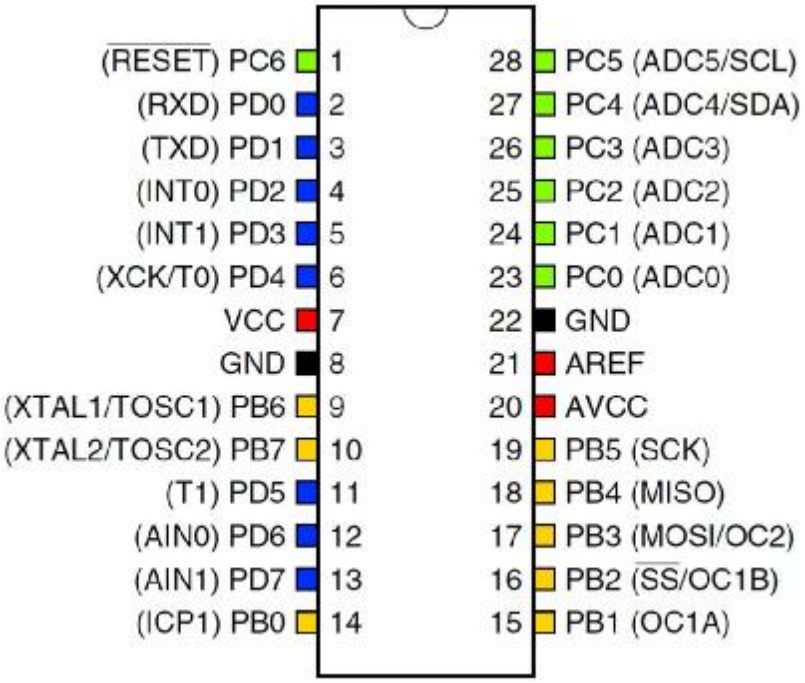

Рис.3. Цоколёвка ATMega8 в корпусе PDIP.

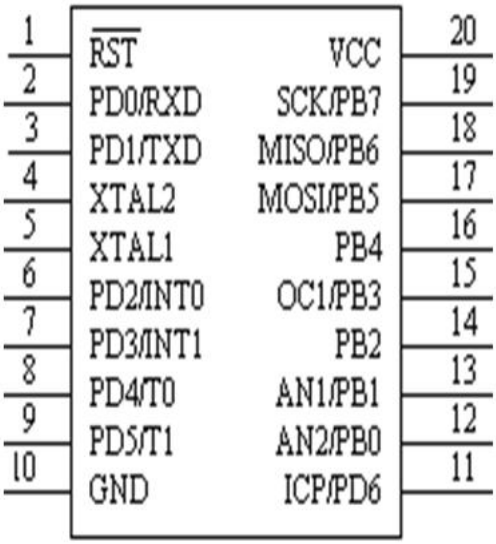

Рис.4. Цоколёвка ATtiny2313 в корпусе PDIP.

Рис.5. Микроконтроллер ATMega8 в корпусе TQFP

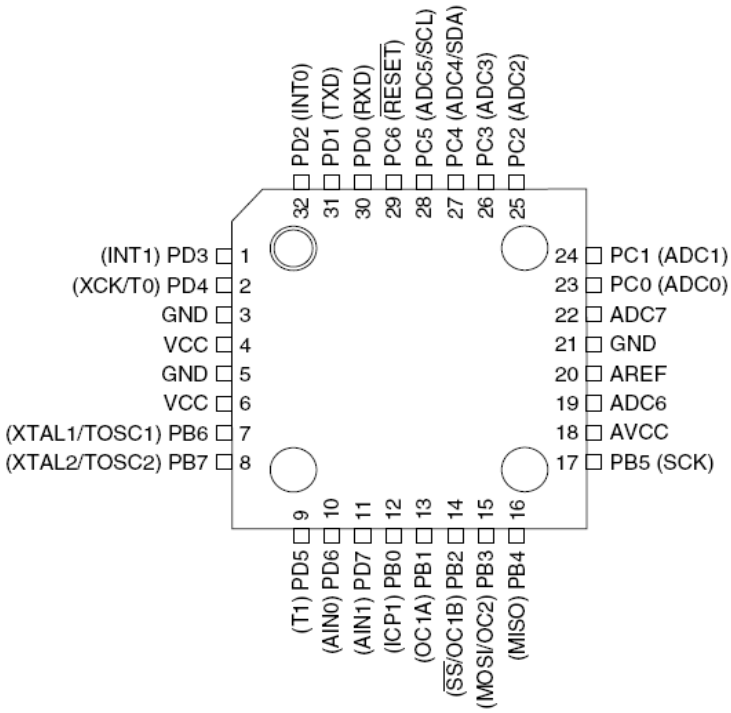

Рис.6. Цоколёвка ATMega8 в корпусе TQFP.

Таб. 1 Описание выводов микроконтроллера модели ATMega8

Обозначения	Номер вывода		Тип вывода	Описание
	DIP	TQFP		
Порт B. (8-разрядный двунаправленный порт ввода/вывода с внутренними подтягивающими резисторами)				
PB0 (ICP)	14	12	I/O	B0 (Вход захвата таймера/счетчика T1 (режим Capture))
PB1 (OC1A)	15	13	I/O	B1 (Выход A таймера/счетчика T1 (режим Compare, PWM))
PB2 ($\overline{SS}$ /OC1B)	16	14	I/O	B2 (Выбор Slave-устройства в канале SPI/ Выход B таймера/счетчика T1 (режим Compare, PWM))
PB3 (MOSI/OC2)	17	15	I/O	B3 (Выход (Master) или вход (Slave) данных канала SPI/ Выход таймера/счетчика T2 (режим Compare, PWM))
PB4 (MISO)	18	16	I/O	B4 (Вход (Master) или выход (Slave) данных канала SPI)
PB5 (SCK)	19	17	I/O	B5 (Выход (Master) или вход (Slave) тактового сигнала SPI)
PB6(XTAL1/TOSC1)	9	7	I/O	B6 (Вход тактового сигнала/Выход для подключения резонатара к таймеру/счетчику T2)
PB7(XTAL2/TOSC2)	10	8	I/O	B7 (Выход тактового генератора/ Вывод для подключения резонатора к таймеру/счетчику T2)
Порт C. (7-разрядный двунаправленный порт ввода/вывода с внутренними подтягивающими резисторами)				
PC0 (ADC0)	23	23	I/O	C0 (Вход АЦП)
PC1 (ADC1)	24	24	I/O	C1 (Вход АЦП)
PC2 (ADC2)	25	25	I/O	C2 (Вход АЦП)
PC3 (ADC3)	26	26	I/O	C3 (Вход АЦП)
PC4 (ADC4/SDA)	27	27	I/O	C4 (Вход АЦП/Линия данных модуля TWI)
PC5 (ADC5/SCL)	28	28	I/O	C5 (Вход АЦП/Тактовый сигнал модуля TWI)
PC6 ($\overline{RESET}$)	1	29	I/O	C6 (Вход сброса)
ADC6	-	19	I	Вход АЦП
ADC7	-	22	I	Вход АЦП
Порт D. (8-разрядный двунаправленный порт ввода/вывода с внутренними подтягивающими резисторами)				
PD0 (RXD)	2	30	I/O	D0 (Вход USART)
PD1 (TXD)	3	31	I/O	D1 (Выход USART)
PD2 (INT0)	4	32	I/O	D2 (Вход внешнего прерывания)
PD3 (INT1)	5	1	I/O	D3 (Вход внешнего прерывания)
PD4 (T0/XCK)	6	2	I/O	D4 (Вход внешнего тактового сигнала таймера/счетчика T0/Вход/ выход внешнего тактового сигнала USART)
PD5 (T1)	11	9	I/O	D5 (Вход внешнего тактового сигнала таймера/счетчика T1)
PD6 (AIN0)	12	10	I/O	D6 (Положительный вход компоратора)
PD7 (AIN1)	13	11	I/O	D7 (Отрицательный вход компоратора)

AREF	21	20	P	Вход опорного напряжения для АЦП
AGND	22	21	P	Аналоговый общий вывод
AV_{cc}	20	18	P	Вывод источника питания АЦП
GND	8	3,5	P	Общий вывод
V_{cc}	7	4,6	P	Вывод источника питания

Структурная схема типичной микропроцессорной системы

Основным действующим элементом современной микропроцессорной системы является микроконтроллер (рис. 1, 2, 5). Однако для того, чтобы понять основополагающие принципы работы, сначала все же необходимо остановиться на микропроцессоре. Сразу нужно сказать, что микропроцессор не работает сам по себе. Микропроцессор — это все-

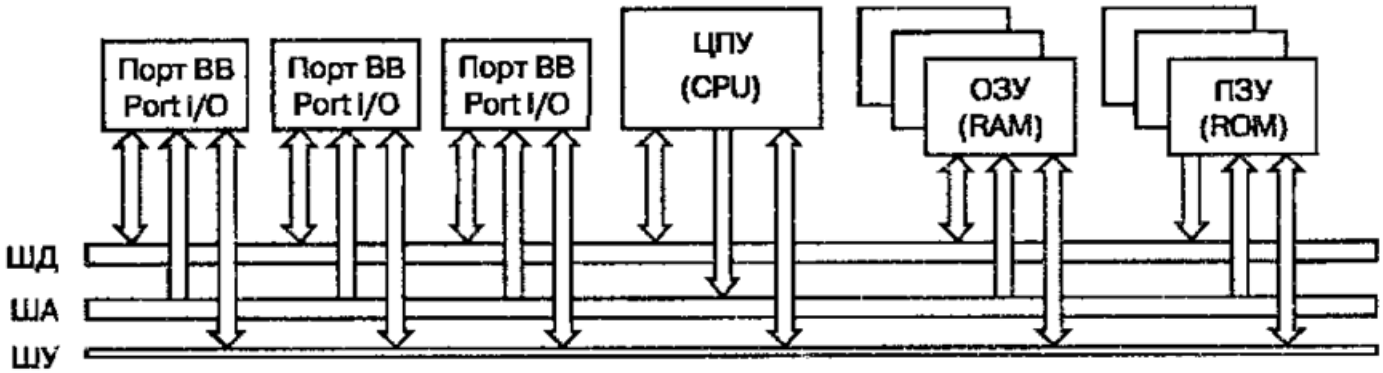

Рис.7. Структурная схема типовой микропроцессорной системы

го лишь часть той или иной микропроцессорной системы.

Кроме собственно микропроцессора, в состав микропроцессорной системы входят и другие, не менее важные элементы. На рис.7 приведена обобщенная структурная схема типичной микропроцессорной системы.

Каждое наименование как в русском, так и в английском варианте представляет собой определенное сокращение полного названия элемента:

- CPU (Central Processing Unit) — центральное процессорное устройство (ЦПУ).
- RAM (Random Access Memory) — устройство с произвольным доступом, или оперативное запоминающее устройство (ОЗУ).
- ROM (Read Only Memory) — память только для чтения, или постоянное запоминающее устройство (ПЗУ).
- Port I/O (Port Input/Output) — порт ввода—вывода.

Рассмотрим все эти элементы подробнее. ***Процессор*** — это самая главная часть, сердце всей системы. Процессор не всегда был микропроцессором. Были времена, когда процессор представлял собой одну или даже несколько электронных плат, собранных из радиоэлементов. Теперь же это достаточно интегрированная малогабаритная микросхема, содержащая в своем составе множество компонент. Процессор предназначен для того, что бы выполнять различные операции с числами. Последовательность этих операций называется ***программой***. Каждая операция кодируется в виде числа и записывается в память. Те числа, с которыми процессор выполняет свои операции, называются данными.

Данные также записаны в память. По сути дела, ***процессор*** — это цифровой автомат, способный выполнять определенный набор операций с числами. Но главной его особенностью является возможность запрограммировать любую последовательность его действий.

Все три части вычислительной системы (процессор, модуль памяти, порты ввода/вывода) связаны между собой шинами данных (рис.7). По этим шинам передаются цифровые сигналы от процессора к модулю памяти, от процессора — к портам ввода—вывода. А также и в обратном направлении: от портов ввода вывода и памяти к процессору.

Процессор может выполнять операции:

- арифметические действия (все простейшие операции, которые можно произвести над числом, он может считать число из любой ячейки памяти, складывать, вычитать, сравнивать, умножать и делить прочитанные числа.
- записывать обратно в память результат вычислений
- логические операции с числами (булевы функции).

Набор операций, которые процессор способен выполнять с участием портов ввода—-вывода, гораздо меньше, чем операций с ячейками памяти. В них также можно записывать и считывать информацию. Однако хранение чисел — это не главное назначение портов.

Порт ввода — это специальное электронное устройство, на которое извне поступают какие-либо электрические сигналы, предназначенные для управления микропроцессорным устройством. Например, сигналы, возникающие при нажатии клавиш на клавиатуре, сигналы, возникающие при срабатывании различных датчиков, и т. п. Процессор считывает их в виде чисел и обрабатывает полученные числа в соответствии с алгоритмом управления.

Порт вывода выполняет обратную функцию. В них процессор записывает различные числа, которые затем поступают на внешние устройства в виде электрических сигналов.

Эти сигналы используются для управления. Управлять можно любым устройством, которое допускает электрическое управление, это индикаторы, дисплеи, электромагнитные реле, электромоторы, электропневмоклапаны, электрические нагреватели и т. д.

Нужно только усилить управляющие сигналы до требуемой мощности. Кроме перечисленных выше команд в любой микропроцессор заложен набор специальных команд, специфических для задач управления процессом вычислений.

Виды памяти

Два вида памяти (ОЗУ и ПЗУ) предназначены для хранения информации (данных и программ). Оба вида памяти представляют собой набор ячеек, в каждой из которых может храниться одно двоичное число. Деление на постоянную и оперативную память достаточно условно. С точки зрения процессора, оба эти вида памяти практически идентичны. Однако все же между ними есть одно довольно существенное различие.

После того, как информация записана в ОЗУ, она хранится там лишь до тех пор, пока подано напряжение питания. Как только питание будет отключено, информация, записанная в ОЗУ, тут же теряется. Об этом мы уже говорили выше. Классический пример ячейки ОЗУ — это простейший регистр, построенный на D-триггерах.

В такой регистр можно записывать информацию и читать ее оттуда. Однако если отключить, а затем включить питание, то все триггеры, из которых состоят регистры ОЗУ, установятся в случайное состояние. Информация будет утеряна. Современные микросхемы памяти строятся на основе совсем других технологий. Но и по сей день не придумано достаточно быстродействующее устройство памяти, не теряющее информации при выключении питания.

Самая распространенная на сегодняшний день технология построения ОЗУ— это так называемая динамическая память. Хранение информации в микросхемах динамической памяти осуществляется при помощи динамически подзаряжаемых миниатюрных емкостей (конденсаторов), выполненных интегральным способом на кристалле кремния.

Каждый конденсатор хранит один бит информации. Если значение бита должно быть равно единице, то схема управления заряжает конденсатор. Если в ячейке должен быть логический ноль, то конденсатор разряжается. Заряженный конденсатор может хранить свой заряд, а, значит, и записанную в него информацию в течение всего нескольких миллисекунд. Дтя того, что бы информация не потерялась, используют регенерацию памяти.

Специальная схема периодически считывает содержимое каждой ячейки памяти и подзаряжает конденсаторы для тех битов, где записана единица. Дтя ускорения процесса регенерации все ячейки памяти каждой микросхемы разбиваются на строки. Считывание и обновление производится сразу для целой строки. Дтя нормальной работы динамического ОЗУ регенерации должна непрерывно работать в течение всего времени, пока включено питание. В современных ОЗУ схема регенерации встраивается внутрь самих микросхем.

Постоянное запоминающее устройство (ПЗУ) предназначено для долговременного хранения информации и не теряет записанную информацию даже после выключения питания. При изготовлении микросхем ПЗУ применяются совершенно другие технологии. На заре микропроцессорной техники микросхемы ПЗУ осуществляли хранение информации благодаря прожиганию внутренних микроперемычек на кристалле. Занесенная таким образом информация не могла быть изменена. Если информация устаревала, микросхему просто выбрасывали и заменяли на другую.

На смену однократно программируемым ПЗУ пришли ПЗУ с ультрафиолетовым стиранием. Такие микросхемы ПЗУ допускали многократное использование. Пережи-

гаемые перемычки получили возможность восстанавливаться. Перед повторным использованием микросхему нужно было «стереть». То есть восстановить перемычки. Для этого кристалл микросхем подвергался облучению световым потоком ультрафиолетового диапазона, для чего микросхемы снабжались специальным окошечком в верхней части корпуса.

Количество циклов записи-стирания для таких микросхем было ограничено. Микросхемы с ультрафиолетовым стиранием просуществовали достаточно долго. Они и сейчас работают во множестве микропроцессорных устройств, изготовленных на рубеже прошлого и нынешнего веков.

Современные же микросхемы ПЗУ строятся по так называемой флэш-технологии (Flash). Такие микросхемы также основаны на применении специальных пережигаемых перемычек с возможностью восстановления. Но стирание информации в данном случае происходит электрическим путем. Поэтому такие микросхемы еще называют ЭСПЗУ (электрически стираемые ПЗУ). Весь процесс стирания осуществляется внутри микросхемы. Для запуска процесса стирания достаточно подать определенную комбинацию сигналов на ее входы.

Будучи включенными в состав микропроцессорной системы, микросхемы ОЗУ и микросхемы ПЗУ работают как единая память программ и данных. Хотя процессор и работает с обоими видами памяти одинаково, но из ПЗУ он может только читать информацию. Запись информации в ПЗУ невозможна. Если микропроцессор все же попытается произвести запись, то ничего страшного не произойдет. Просто в ячейке останется то, что там было до попытки записи.

Порты ввода—вывода

Порты ввода—вывода — это специальные микросхемы, при помощи которых микропроцессорная система может общаться с внешним миром. Без портов теряется весь смысл микропроцессорной системы. Она не может работать сама по себе. Микропроцессор должен чем-то управлять, а иначе зачем он? Через порты ввода процессор получает внешние воздействия (управляющие сигналы). Например, сигналы от кнопок, датчиков. При помощи портов вывода процессор управляет внешними устройствами (реле, моторами, световыми индикаторами, дисплеями).

Процессор работает с портами ввода—вывода практически так же, как и с ячейками памяти. Работа с портами сводится к тому, что процессор просто читает число из порта ввода или записывает число в порт вывода. В качестве порта вывода чаще всего выступает обыкновенный параллельный регистр. Порт ввода еще проще. Это простая ключевая схема, которая по команде с центрального процессора подает внешние данные на его входы.

Процессор и цифровые шины

Главным управляющим элементом всей микропроцессорной системы является процессор. Именно он, за исключением нескольких особых случаев, управляет и памя-

тью, и портами ввода—вывода. Память и порты ввода—вывода являются пассивными устройствами и могут только отвечать на управляющие воздействия.

Для того, чтобы процессор мог управлять микропроцессорной системой, он соединен со всеми ее элементами при помощи цифровых шин. Как уже говорилось, шина — это набор параллельных проводников, по которым передается цифровой сигнал. Эти проводники называются линиями шины.

В каждый момент времени по шине передается одно двоичное число. По каждой линии передается один разряд этого числа. В любой микропроцессорной системе имеется, по крайней мере, три основных шины. Все они изображены на рис. 7, но даны только русскоязычные названия шин. Ниже приведена расшифровка этих названий и их англоязычный эквивалент:

- ШД — шина данных (DATA bus);
- ША — шина адреса (ADDR bus);
- ШУ — шина управления (CONTROL bus).

ВЫПОЛНЕНИЕ РАБОТЫ.

Разработка принципиальной электрической схемы

Разработаем принципиальную электрическую схему, способную выполнять описанную выше задачу. К микроконтроллеру нам нужно подключить светодиод и кнопку управления. Для подключения к микроконтроллеру AVR любых внешних устройств используются порты ввода—вывода. Причем каждый такой порт способен работать либо на ввод, либо на вывод.

Удобнее всего светодиод подключить к одному из портов, а кнопку — к другому. В этом случае управляющая программа должна будет настроить порт, к которому подключен светодиод, на вывод, а порт, к которому подключена кнопка, на ввод. Других специальных требований к микроконтроллеру не имеется. Поэтому выберем микроконтроллер.

Очевидно, что нам нужен микроконтроллер, который имеет не менее двух портов. Данным условиям удовлетворяют многие микроконтроллеры AVR. Остановим свой выбор на микросхеме ATtiny2313. Эта микросхема, хотя и относится к семейству «Ttiny», на самом деле занимает некое промежуточное место между семейством «Ttiny» и семейством «Mega». Она не так перегружена внутренней периферией и не столь сложна, как микросхемы семейства «Mega». Но и не настолько примитивна, как все остальные контроллеры семейства «Ttiny».

Эта микросхема содержит два основных и один дополнительный порт ввода—вывода, имеет не только восьмиразрядный, но и шестнадцатиразрядный таймер/ счетчик. Имеет оптимальные размеры (20-выводной корпус).

Итак, если не считать порта A, который включается только в особом режиме, который мы пока рассматривать не будем, микроконтроллер имеет два основных порта ввода—вывода (порт B и порт D). Договоримся, что для управления светодиодом мы будем использовать младший разряд порта B (линия PB.0), а для считывания информации с

кнопки управления используем младший разряд порта D (линия PD.0). Полная схема устройства, позволяющего решить поставленную выше задачу, приведена на рис. 8.

Для подключения кнопки S1 использована классическая схема. В исходном состоянии контакты кнопки разомкнуты. Через резистор Rl на вход PD.0 микроконтроллера подается «плюс» напряжения питания, что соответствует сигналу логической единицы.

При замыкании кнопки напряжение падает до нуля, что соответствует логическому нулю. Таким образом, считывая значение сигнала на соответствующем выводе порта, программа может определять момент нажатия кнопки. Несмотря на простоту данной схемы, микроконтроллер AVR позволяет ее упростить. А именно, исключим резистор Rl, заменив его внутренним нагрузочным резистором микроконтроллера. Как уже говорилось выше, микроконтроллеры серии AVR имеют встроенные нагрузочные резисторы для каждого разряда порта. Главное при написании программы—не забыть включить программным путем соответствующий резистор.

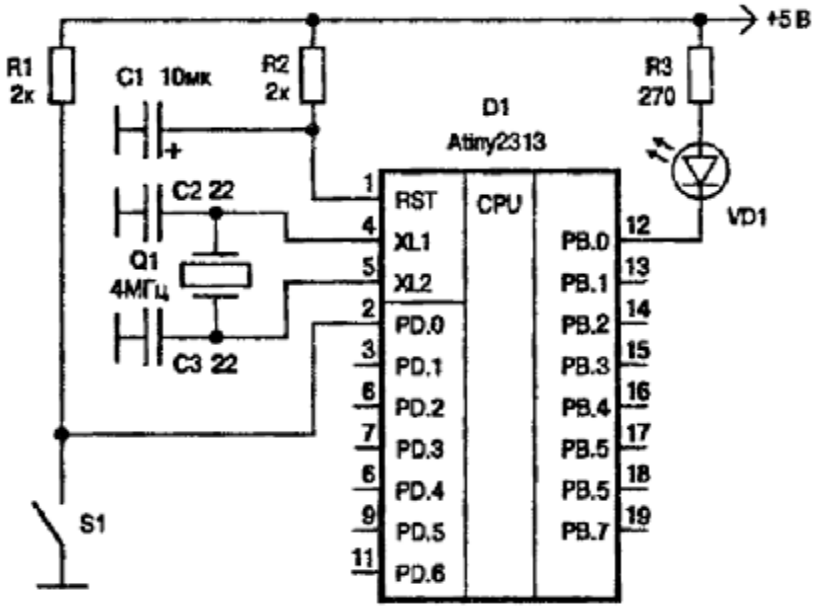

Рис.8. Принципиальная схема со светодиодом и кнопкой

Подключение светодиода также выполнено по классической схеме. Это непосредственное подключение к выходу порта. Каждый выход микроконтроллера рассчитан на непосредственное управление светодиодом среднего размера с током потребления до 20 мА. В цепь светодиода включен токоограничивающий резистор R3.

Для того, чтобы зажечь светодиод, микроконтроллер должен подать на вывод PB.0 сигнал логического нуля. В этом случае напряжение, приложенное к цепочке R3, VDl, окажется равным напряжению питания, что вызовет ток через светодиод, и он загорится. Если же на вывод PB.0 подать сигнал логической единицы, падение напряжения на светодиоде и резисторе окажется равным нулю, и светодиод погаснет.

Кроме цепи подкчлючения кнопки и цепи управления светодиодом, на схеме вы можете видеть еще несколько цепей. Это стандартные цепи, обеспечивающие нормальную работу микроконтроллера. Кварцевый резонатор Ql обеспечивает работу встроенного тактового генератора. Конденсаторы С2 и С3 — это цепи согласования кварцевого резонатора.

Элементы Cl, R2—это стандартная цепь начального сброса. Такая цепь обеспечивает сброс микроконтроллера в момент включения питания. Еще недавно подобная цепь была обязательным атрибутом любой микропроцессорной системы. Однако технология производства микроконтроллеров достигла такого уровня, что обе эти цепи (внешний кварц и цепь начального сброса) теперь можно исключить.

Большинство микроконтроллеров AVR, кроме тактового генератора с внешним кварцевым резонатором, содержат внутренний RC-генератор, не требующий никаких внешних цепей. Если вы не предъявляете высоких требований к точности и стабильности частоты задающего генератора, то микросхему можно перевести в режим внутреннего RC-генератора и отказаться как от внешнего кварца (Ql), так и от согласующих конденсаторов (С2 и С3).

Цепь начального сброса тоже можно исключить. Любой микроконтроллер AVR имеет внутреннюю систему сброса, которая в большинстве случаев прекрасно обеспечивает стабильный сброс при включении питания. Внешние цепи сброса применяются только при наличии особых требований к длительности импульса сброса. А это бывает лишь в тех случаях, когда микроконтроллер работает в условиях больших помех и нестабильного питания. Все описанные выше переключения производятся при помощи соответствующих fuse-переключателей.

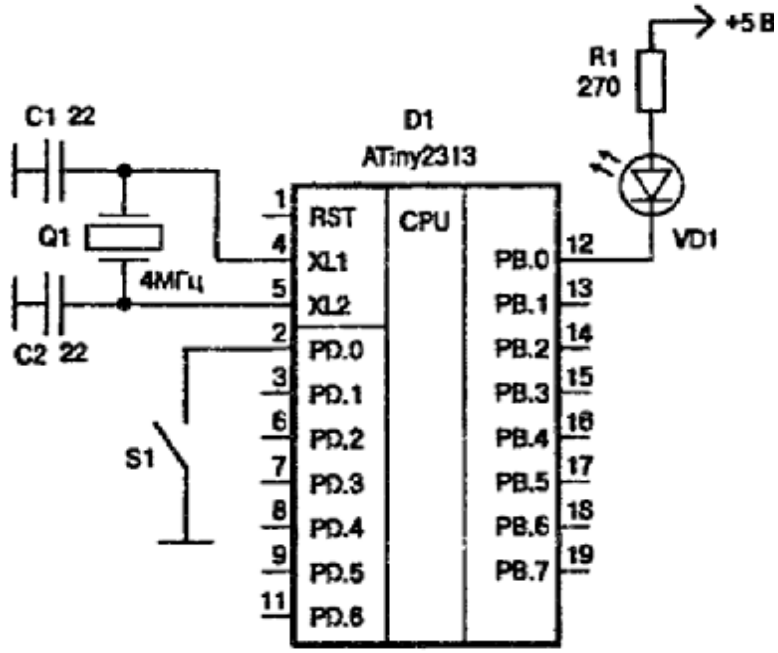

Рис. 9. Усовершенствованная схема

Три освободившихся вывода микроконтроллера могут быть использованы как дополнительный порт (порт А). Но в данном случае в этом нет необходимости.

Упростим схему, показанную на рис. 8, с учетом описанных выше возможностей.

От внешнего кварца пока отказываться не будем. Он нам пригодиться чуть позже, когда мы начнем формировать временные интервалы. Доработанная схема изображена на рис.9, а на рис.10 и 11 показаны готовая печатная плата и готовая плата в сборе.

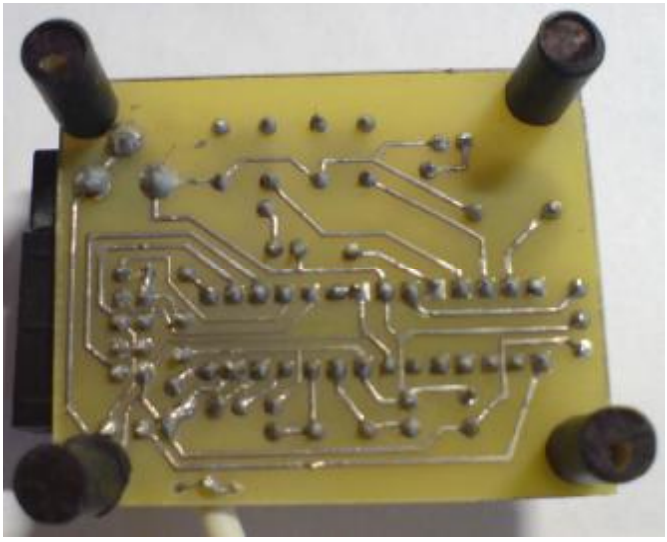

Рис. 10. Печатная плата

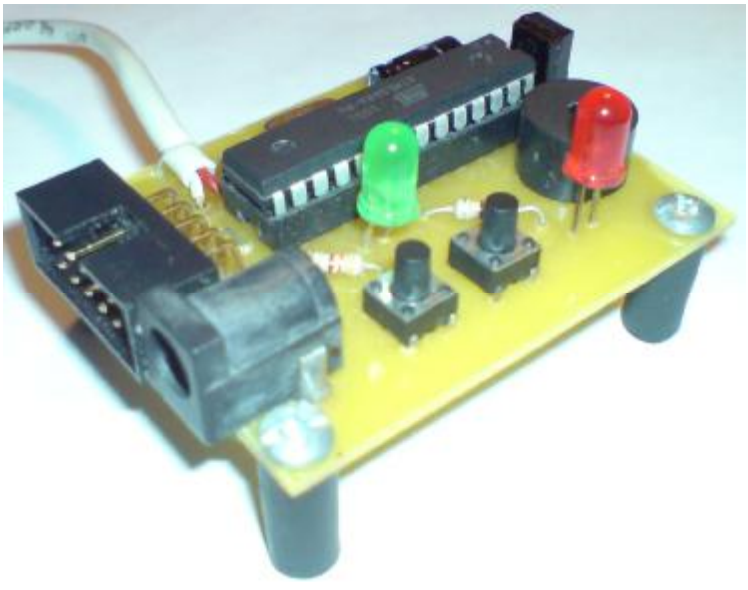

Рис. 11. Монтажная сторона платы

Разработка алгоритма

При наличие схемы приступим к разработке программы. Разработку любой программы будем начинать с разработки алгоритма.

Алгоритм — это последовательность действий, которую должен произвести микроконтроллер, чтобы достичь требуемого результата. Для простых задач алгоритм можно просто описать словами. Для более сложных задач алгоритм рисуется в графическом виде.

В нашем случае алгоритм таков: после операций начальной настройки портов микроконтроллер должен войти в непрерывный цикл, в процессе которого он должен опрашивать вход, подключенный к нашей кнопке, и в зависимости от ее состояния управлять светодиодом. Опишем действия подробнее.

Операции начальной настройки:

- установить начальное значение для вершины стека микроконтроллера (при программировании на Ассемблере);
- настроить порт B на вывод информации;
- подать на выход PB.0 сигнал логической единицы (потушить светодиод);
- сконфигурировать порт D на ввод;
- включить внутренние нагрузочные резисторы порта D.

Операции, составляющее тело цикла:

- прочитать состояние младшего разряда порта PD (PD.0);
- если значение этого разряда равно единице, выключить светодиод;
- если значение разряда PD.0 равно нулю, включить светодиод;
- перейти на начало цикла.

Использование системы виртуального моделирования

В наш век повальной компьютеризации появилось огромное количество программ симуляторов, заменяющих реальные радиодетали и приборы, виртуальными моделями. Симуляторы позволяют, без сборки реального устройства, отладить работу схемы, найти ошибки, полученные на стадии проектирования, снять необходимые характеристики и многое другое.

Одна из таких программ PROTEUS 7.5. Именно этим симулятором мы и будем пользоваться в дальнейшем. Симуляция радиоэлементов это не единственная способность программы. Proteus, созданная фирмой Labcenter Electronics на основе ядра SPICE3F5 университета Berkeley, является так называемой средой сквозного проектирования. Это означает создание устройства, начиная с его графического изображения (принципиальной схемы) и заканчивая изготовления печатной платы устройства, с возможностью контроля на каждом этапе производства.

В «сферу-влияний» PROTEUS входят как простейшие аналоговые устройства так и сложные системы созданные на микроконтроллерах. Доступна огромная библиотека моделей элементов, пополнять которую может сам пользователь, естественно для этого

нужно досконально знать работу элемента и уметь программировать. Достаточный набор инструментов и функций, среди которых вольтметр, амперметр, осциллограф, всевозможные генераторы, способность отлаживать программное обеспечение микроконтроллеров, делают PROTEUS хорошим помощником разработчику электронных устройств.

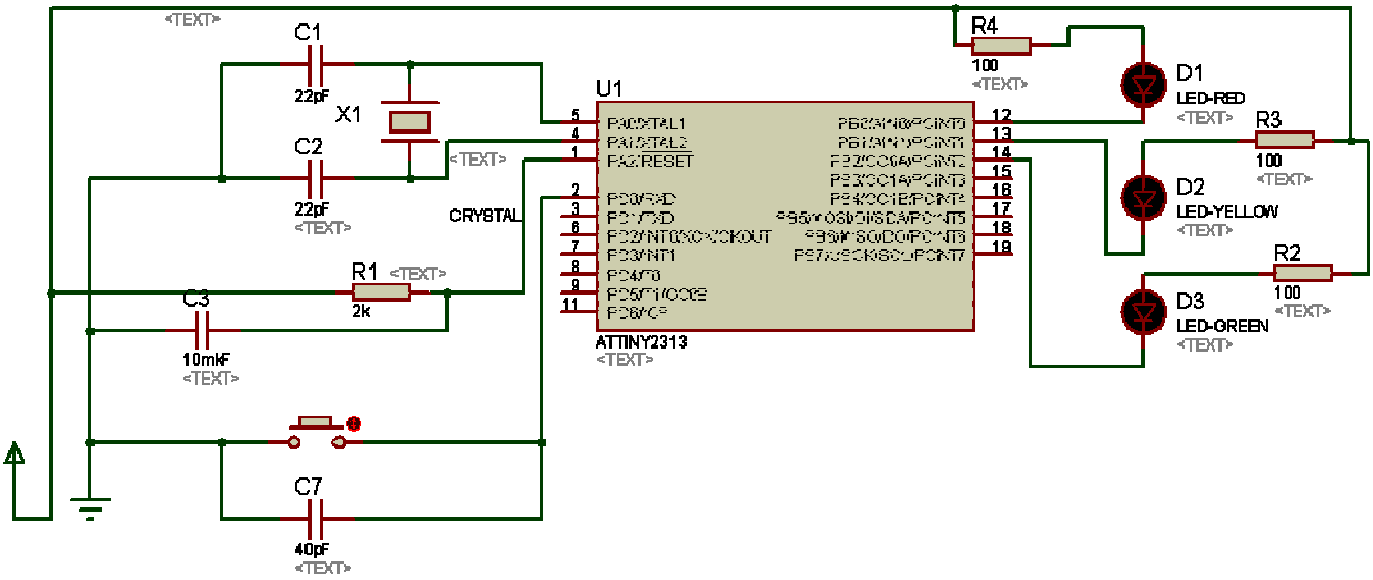

Рис. 12. Принципиальная схема со светодиодным светофором и кнопкой, выполненная в PROTEUS

Proteus VSM состоит из двух самостоятельных программ ISIS и ARES. ARES это трассировщик печатных плат с возможностью создания своих библиотек корпусов.

Основной программой является ISIS, в ней предусмотрена горячая связь с ARES для передачи проекта для разводки платы.

Разработка программы на языке C++

Для создания программ на языке C++ мы будем использовать программную среду CodeVisionAVR. Это среда специально предназначена для разработки программ на языке C++ для микроконтроллеров серии AVR. Среда CodeVisionAVR не имеет своего отладчика, но позволяет отлаживать программы, используя возможности системы AVR Studio.

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 C++. Поэтому давайте сначала научимся работать с мастером, создадим заготовку нашей программы на языке C++, а затем разберем подробнее все ее элементы. И уже после этого создадим из заготовки готовую программу.

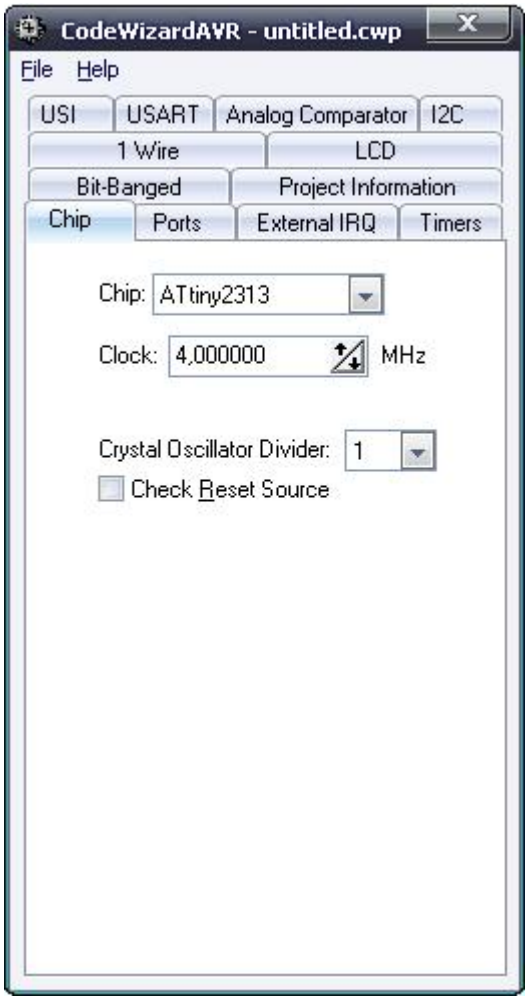

Рис. 13. Окно мастера CodeVisionAVR

Для того, чтобы понять работу мастера, нам необходимо прояснить несколько новых понятий. Программа CodeVisionAVR, как и любая современная среда программирования, работает не просто с текстом программы, а с так называемым Проектом.

Проект, кроме текста программы, содержит ряд вспомогательных файлов, содержащих 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—лишь написать текст программы, для которого в проекте отводится отдельный файл с расширением «c». Например, «proba.c». При помощи мастера мы будем создавать новый проект. В дальнейшем мы еще рассмотрим подробно процесс установки и работу с программной средой CodeVisionAVR. Сейчас же считаем, что она установлена и запущена.

Для создания нового проекта выберем в меню «File» пункт «New». Откроется небольшой диалог, в котором вы 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?». Мы договорились, что будем, поэтому выбираем «Yes», после чего открывается окно построителя (см. рис. 13). Как видите, это окно имеет множество вкладок, каждая из которых содержит элементы выбора режимов.

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 по вкладкам мастера и выберем необходимые нам параметры. Для тех же параметров, которые нам не нужны, оставим значения по умолчанию.

Первая вкладка называется «Chip». На этой вкладке мы можем выбрать общие параметры проекта. Используя выпадающий список «Chip», выберем тип микроконтрол-

лера. Для этого щелкаем мышью по окошку и в выпавшем списке выбираем ATtiny2313 или ATMega8.

При помощи поля «Clock» выбираем частоту кварцевого резонатора. В нашем случае она должна быть равна 4-12 МГц. При помощи поля «Crystal Oscillator Divider» выбирается ***коэффициент деления тактового генератора.*** Поясним этот параметр. Дело в том, что выбранный нами микроконтроллер имеет систему предварительного деления тактовых импульсов. Если частота тактового генератора нас не устраивает, мы можем поделить ее, и микроконтроллер будет работать на другой, более низкой частоте. 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). Не будем углубляться в его назначение. Достаточно будет понять, что включение данного флажка добавляет к создаваемой программе процедуру, связанную с определением источника сигнала системного сброса.

Закончив с общими настройками, перейдем к вкладке (Порты). Эта вкладка позволяет настроить все имеющиеся порты ввода—вывода.

Каждому порту в МК AVR соответствуют минимум три регистра:

DDRx - регистр направления работы - вход или выход. x - означает букву A, B, C, D, E... порта, по числу портов в конкретном МК.

PINx - регистр содержит значения физических (т.е. реальных, тех которые мы получим измерив вольтметром и преобразовав в 1 или 0) уровней сигнала на соответствующих ножках МК.

PORTx - регистр в который записываются желаемые значения "1" или "0" и которые необходимо получить на соответствующих ножках МК при назначении их выходом. Т.е. если соответствующий бит в регистре DDRx установлен (значит равен "1").

Если бит в регистре DDRx равен "0", а в такой же бит PORTx записана "1" то "ножка" МК будет "входом с подтяжкой" т.е. к ней подключен резистор примерно 40 кОм от питания МК.

На вкладке «Ports» мы видим еще три вкладки поменьше (см. рис.14, 15). По одной вкладке для каждого порта. Как уже говорилось выше, порт PA в данной схеме мы не применяем.

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 пор-

та. 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 показано на рис. 14.

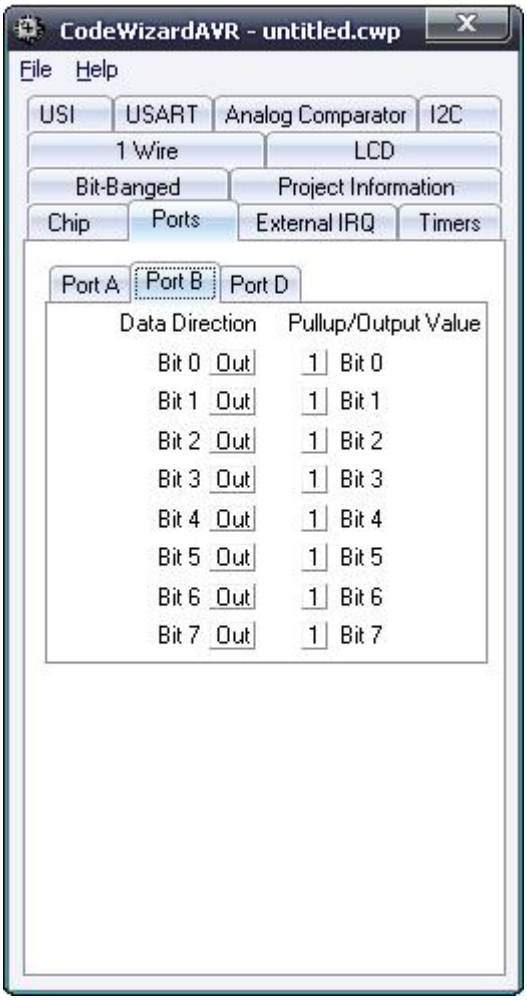

Рис. 14. Настройка порта PB

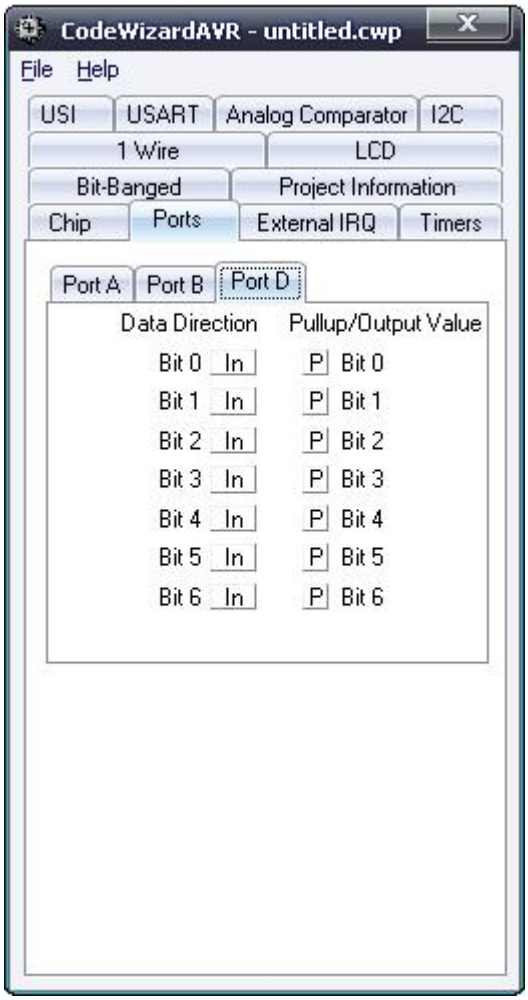

Рис. 15. Настройка порта PD

Перейдем к настройке порта D. Для этого выберем вкладку «Potr D». На вкладке мы увидим такие же органы управления, как на вкладке «Port B» (см. рис. 15). По условиям задачи порт PD микроконтроллера должен работать на ввод. Поэтому состояние элементов первого столбца мы не меняем.

Однако не забывайте, что 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 «T» или «P». ***«T» (Terminate)*** означает отсутствие внутренней нагрузки, а ***«P» (Pull-up)*** означает: нагрузка включена. Включим нагрузку для каждого разряда порта PD, изменив при помощи мыши значение поля с «T» на «P». В результате элементы управления будут выглядеть так, как показано на рис. 15.

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

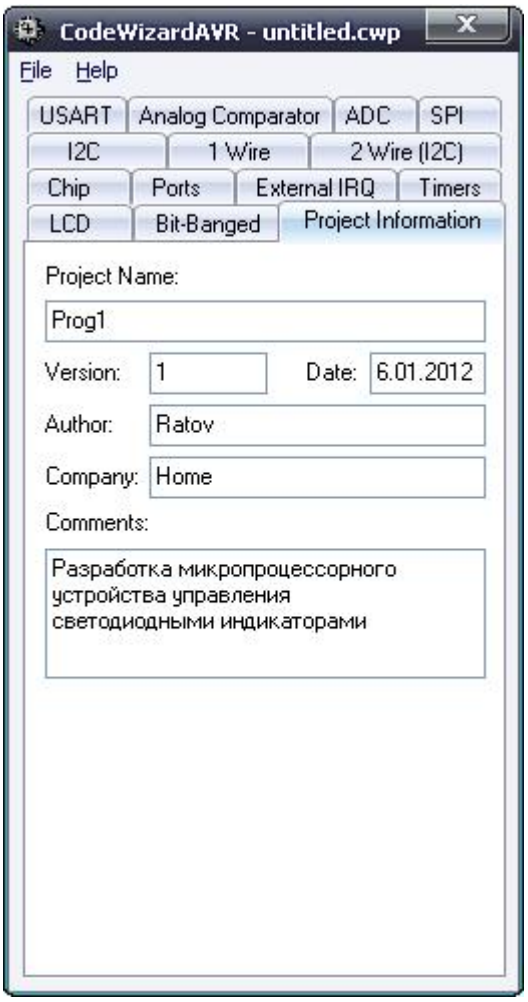

Рис. 16. Занесение информации для заголовка программы

Воспользуемся еще одним полезным свойством мастера программ. Откроем вкладку **«Project Information»** (см. рис. 16). 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» нашего мастера пункт «Generate, Save and Exit», как это показано на рис. 17. 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 вам нужно самостоятельно выбрать каталог, где будет размещен весь ваш проект.

Рекомендуется для каждого проекта создавать свой отдельный каталог. В нашем случае самое простое — назвать каталог именем «Prog1». А вернее, выбрать что-то вроде «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».

Однако вы можете выбрать имя по своему усмотрению. 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 C++.

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. Подробнее о закладках и точках останова мы узнаем, когда будем изучать работу отладчика**.** В качестве имени файла проекта удобнее всего использовать то же самое имя, что и для текста программы. То есть «Prog1». 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).

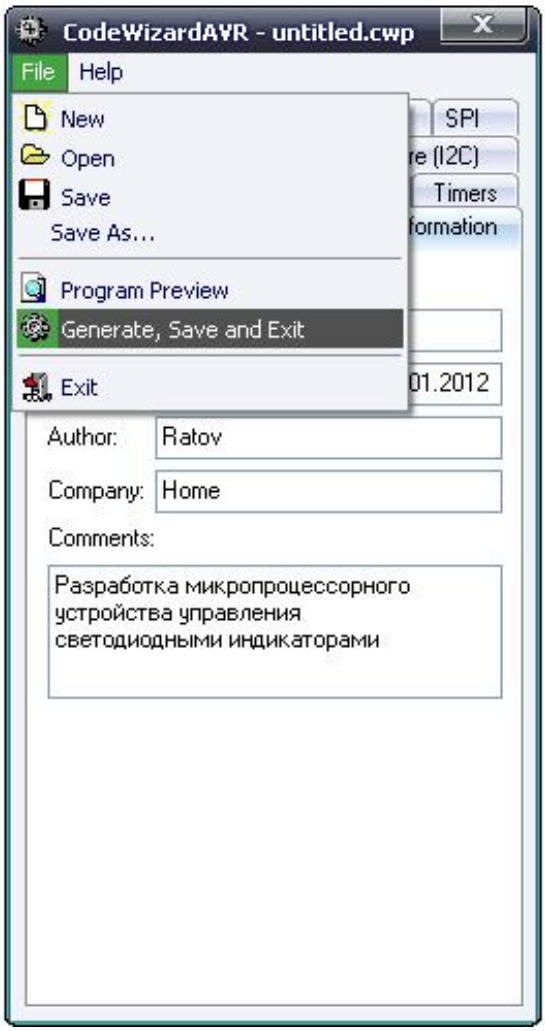

Рис. 17. Запуск процесса генерации програмы программы

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 («Prog1»). 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 «Prog1.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.

Посмотрим теперь, что же сформировал наш построитель. Текст программы, полученной описанным выше образом, дополненный русскоязычными комментариями, приведен в листинге 1.

Все русскоязычные комментарии дублируют соответствующие англоязычные комментарии, автоматически созданные в процессе генерации программы. Кроме новых комментариев, в программу добавлена только одна дополнительная строка (строка 32). Именно она превращает созданную построителем заготовку в законченную программу. Итак, мы получили программу на языке C++.

Анализ работы программы

Сначала рассмотрим небольшое введение в язык C++. Программа на языке C++, в отличие от Ассемблера, гораздо более абстрагирована от системы команд микроконтроллера. Основные операторы языка C++ вовсе не привязаны к командам микроконтроллера. Для реализации всего одной команды на языке C++ 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 С++ так же, как и другие языки программирования, состоит из команд. Для записи каждой команды С++ использует свои операторы и псевдооператоры. Но форма написания команд в программе приближена к форме, принятой в математике.

В языке С++ для хранения различных данных используются переменные. Понятие «переменная» в языке С++ 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;
kl/2 — переменная kl, деленная на два;
massal + massa2 — сумма двух переменных (massal и massa2);

tkon < < 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 их двух членов. Язык С++ допускает выражения практически любой сложности.

В языке С++ 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 С++ поддерживают различное количество типов переменных. Версия С++, используемая в CodeVisionAVR, поддерживает тринадцать типов переменных.

В языке С++ 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 С++ программа выделяет для хранения значений каждой переменной определенные ресурсы памяти.

Листинг 1

```
1  #include <tiny2313.h>
  // Declare your global variables here (определение ваших
                                        глобальных переменных)
2 void main(void){
   // Declare your local variables here
   // Crystal Oscillator division factor: 1 (коэффициент
                     деления частоты системного генератора: 1)
3      CLKPR=0x80;
4      CLKPR=0x00;


 // Input/Output Ports initialization (инициализация портов
ввода-вывода)
 // Port A initialization (инициализация порта A)
 // Func2=In Func1=In Func0=In
 // State2=T State1=T State0=T
5      PORTA=0x00;
6      DDRA=0x00;

 // Port B initialization (инициализация порта B)
 // Func7=Out Func6=Out Func5=Out Func4=Out
 // Func3=Out Func2=Out Func1=Out Func0=Out
 // State7=1 State6=1 State5=1 State4=1
 // State3=1 State2=1 State1=1 State0=1
7      PORTB=0xFF;
8      DDRB=0xFF;

// Port D initialization (инициализация порта D)
// Func6=In Func5=In Func4=In Func3=In
// Func2=In Func1=In Func0=In
// State6=P State5=P State4=P State3=P
// State2=P State1=P State0=P
9      PORTD=0x7F;
10     DDRD=0x00;

// Timer/Counter 0 initialization (инициализация
                                    таймера/счётчика 0)
// Clock source: System Clock (Источник сигнала: системный
                                    генератор)
// Clock value: Timer 0 Stopped(Значение частоты: Таймер 0
                                    остановлен)
 // Mode: Normal top=FFh (Режим Normal макс. значение FFH)
 // OC0A output: Disconnected   (выход 0C0A отключён)
```

```
   // OC0B output: Disconnected   (выход 0C0B отключён)
11      TCCR0A=0x00;
12      TCCR0B=0x00;
13      TCNT0=0x00;
14      OCR0A=0x00;
15      OCR0B=0x00;
  // Timer/Counter 1 initialization(инициализация
                                    таймера/счётчика 1)
 // Clock source: System Clock (Источник сигнала: системный
                                   генератор)
 // Clock value: Timer1 Stopped (Значение частоты: Таймер 1
                                   остановлен)
   // Mode: Normal top=FFFFh (Режим Normal макс.значение FFFFH)
   // OC1A output: Discon.   (выход 0C0A отключён)
   // OC1B output: Discon.   (выход 0C0B отключён)
   // 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
16      TCCR1A=0x00;
17      TCCR1B=0x00;
18      TCNT1H=0x00;
19      TCNT1L=0x00;
20      ICR1H=0x00;
21      ICR1L=0x00;
22      OCR1AH=0x00;
23      OCR1AL=0x00;
24      OCR1BH=0x00;
25      OCR1BL=0x00;

 // External Interrupt(s) initialization (Инициализация
                                          внешних прерываний)
   // INT0: Off    (Прерывание INT0 выключено)
   // INT1: Off    (Прерывание INT1 выключено)
   // Interrupt on any change on pins PCINT0-7: Off
26      GIMSK=0x00;
27      MCUCR=0x00;

 // Timer(s)/Counter(s) Interrupt(s) initialization
                  (Инициализация прерываний от таймера)
28      TIMSK=0x00;

   // Universal Serial Interface initialization(Инициализация
                универсального последовательного интерфейса)
```

```
   // Mode: Disabled   (Режим выключен)
   // Clock source: Register & Counter=no clk.
   // USI Counter Overflow Interrupt: Off
29      USICR=0x00;

    // Analog Comparator initialization (Инициализация
                                          аналогового компаратора)
   // Analog Comparator: Off  (Аналоговый компаратор выключен)
   // Analog Comparator Input Capture by Timer/Counter 1: Off
30      ACSR=0x80;

31      while (1) {
      // Place your code here   (Пожалуйста вставьте Ваш код)
32          PORTB=PIND;
        };
   }
```

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. Принцип формирования имен в языке C++ 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 C++ использует функции.

Функция в языке C++— 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 C++ 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 C++ огромная. Программа на языке C++ 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 C++:

```
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.

Тело функции—это набор операторов на языке C++, выполняющих некие действия. В конце каждого оператора ставится точка с занятой. Если функция небольшая, то ее можно записать в одну строку.

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

тип Name (список параметров) { тело функции }

Любая программа на языке C++ должна обязательно содержать одну главную функцию. Главная функция должна иметь имя **main**. Выполнение программы всегда начинается с выполнения функции main. В нашем случае (см. листинг 1) описание функции main начинается со строки 2 и заканчивается в конце программы. Функция main в данной версии языка C++ 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 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.

Рассмотрим неизвестные команды листинга 1.

`include`

Оператор присоединения внешних файлов. Данный оператор выполняет точно такую же роль, что и аналогичный оператор в языке Ассемблер. В строке 1 программы этот 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 C++ принято считать, что выражение истинно, если оно не равно нулю, и ложно, если равно.

Тело цикла — это ряд любых операторов языка C++. Как и любая другая команда, вся конструкция while должна заканчиваться символом тючка с запятой».

В программе на листинге 1 оператор `while` вы можете видеть в строке 31. В качестве условия в этом операторе используется просто число 1. Так как 1 — не ноль, то такое условие всегда истинно. Такой прием позволяет создавать бесконечные циклы. Это значит, что цикл, начинающийся в строке 31, будет выполняться бесконечно. Тело цикла составляет единственная команда (строка 32).

Как уже говорилось, текст программы листинга 1 в основном сформирован автоматически. В строке 1 находится команда include, присоединяющая файл описаний. После команды include мастер поместил сообщение для программиста. Сообщение предупреждает о том, что именно в этом месте программисту нужно поместить описание всех глобальных переменных (если, конечно, они вам понадобятся). В данном конкретном случае глобальные переменные нам не нужны. Поэтому мы добавлять их не будем.

Функция `main` содержит в себе набор команд, настройки системы (строки 3—30) и заготовку главного цикла программы (строка 31).

Настройка системы — это 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 (строки 7, 8 для порта PB и строки 9,10 для порта PD). А также команду инициализации компаратора (строка 30).

Рассмотрим, как происходит присвоение значений. В строке 3 регистру `CLKPR` присваивается значение `0x80`. Для присвоения значения используется оператор присвоения «=» (равно). Таким же самым образом присваиваются значения и всем остальным регистрам.

Что касается настройки портов PB и PD, то в строках 7, 8 регистрам PORTB и DDRB присваивается значение 0xFF. А в строках 9, 10 в регистр PORTD записывается 0x7F, а в регистр DDRD—ноль.

В регистр PORTD можно записать 0xFF (в двоичном варианте 0b11111111). Но в программе на C++ в тот же регистр мы записываем 0x7F (0b01111111). Эти два числа отличаются лишь значением старшего бита. Но для данного порта это несущественно, так как его старший разряд незадействован. Поэтому в каждой программе мы делаем так, как это удобнее.

После инициализации всех регистров начинается основной цикл программы (строка 31). 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 (строка 38). В нашем случае он состоит всего из одной команды. Эта команда присваивает регистру PORTB значение регистра PORTD. Выполняясь многократно в бесконечном цикле, команда присвоения постоянно переносит содержимое порта PD в порт PB, реализуя тем самым заданный алгоритм.