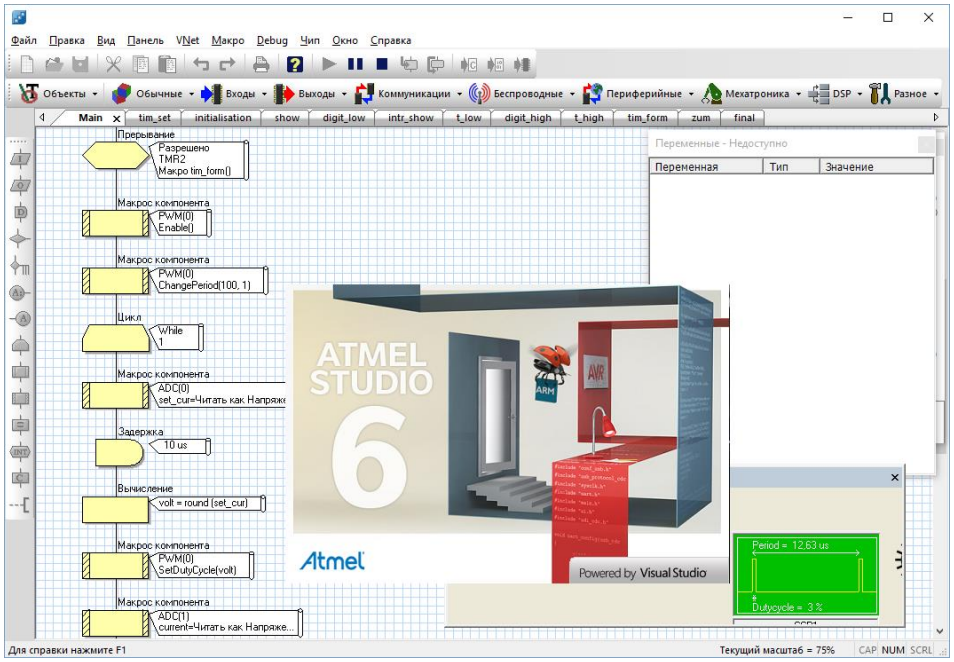
В.Н. Гололобов

Рассказ, навеянный собственной глупостью



Москва - 2015

Оглавление

Порой глупости мы делаем, основываясь на вполне разумных предположениях и добрых намерениях.

Случилось так, что я взялся за ремонт одного устройства, что, как правило, не делаю без жестокой необходимости. Не спутниковая система, но при отсутствии схемы и описания программы для «сердца» аппарата микроконтроллера ATMega 48, и это занятие, что называется, «на любителя». Но так уж сложились обстоятельства.

Сделав поспешные выводы на основании скудного описания работы устройства, я решил проверить микроконтроллер. Однако во всех устройствах микроконтроллеры защищают от считывания, а, когда я «снёс» защиту, то ожидаемо удалил и исходную программу.

Проверка показала, что микроконтроллер вполне исправен. Программу пришлось писать заново.

Глава 1. Когда схема устройства отсутствует

Начиная работу с аппаратом, схемы которого у вас нет, следует быть готовым к тому, что количество ошибок окажется значительным. Особенно это касается современных устройств с поверхностным монтажом. Обозначение компонентов так плохо читаются, что не всегда знаешь, с чем ты имеешь дело. Конечно, если это плановая производственная задача, то на ознакомление отводится время. В любительских условиях, когда хочется всё сделать «побыстрее», спешка вносит свой вклад в количество ошибок.

Данный аппарат был разработан на базе микроконтроллера. Это несколько упрощает задачу, поскольку от контроллера и следует «плясать».

К микроконтроллеру ATMega 48 подключены два двухразрядных семисегментных индикатора. Что подразумевает динамическое управление индикаторами, сегменты которых занимают один из портов микроконтроллера и ещё половину второго порта занимают общие выводы индикаторов.

Помимо индикаторов есть задатчик, потенциометр, с помощью которого устанавливается рабочий ток. Наличие задатчика подразумевает, что один из выводов микроконтроллера будет использоваться в качестве АЦП.

> *Вот эта часть схемы и ввела меня в заблуждение – я решил, что индикаторы должны показывать напряжение, которое считывается с задатчика.*

Реальная схема оказалась устроенной иначе. Индикаторы показывают реальный ток, протекающий в рабочей цепи через электроды. Ток, конечно, считывается как падение напряжения на резисторе с небольшим сопротивлением, включённом в рабочую цепь. А считывание ещё одного напряжения – это задача второго канала АЦП.

В кратком описании аппарата написано, что рабочий ток формируется с помощью цифро-аналогового преобразования. Для микроконтроллера разумно было предположить, и это подтвердилось, что преобразование осуществляется с помощью широтно-импульсной модуляции. То есть, один из выводов формирует сигнал PWM.

А для интеграции сигнала используется RC цепь, подкреплённая операционным усилителем. Трудно различимая надпись на одном из компонентов 358 дала повод считать его сдвоенным

операционным усилителем LM358, первая половина которого используется в качестве повторителя управляющего напряжения. Которое, в свою очередь, управляет транзистором IRF. Появление транзисторов такого плана существенно упростило многие схемные решения.

Интересным оказалось решение разработчиков по второй половине операционного усилителя, формирующего управляющее напряжение. Но об этом позже.

По первому впечатлению было не понятно назначение транзисторов (часть с маркировкой 3G, но есть и 1G), включённых в цепи общих электродов индикаторов. Их назначение выяснилось тогда, когда пришлось разбираться с клавиатурой из четырёх клавиш, подключённых к первым сегментам индикаторов.

Реле, включающее и выключающее рабочую цепь, оказалось без маркировки. А источник звукового сигнала, почему так сделано, не понятно, был выбран электромагнитный. И то, и другое подключалось через свои транзисторы, управляемые от выводов микроконтроллера.

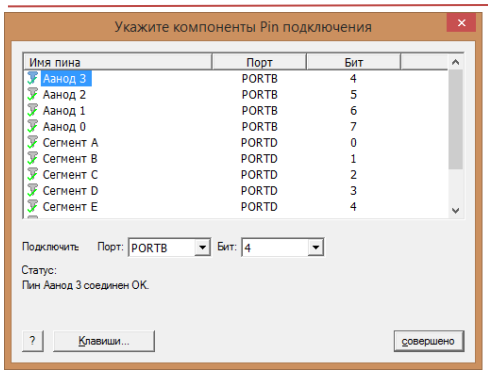
Столь существенное использование портов контроллера в какой-то мере объясняло, почему клавиатура, работающая на ввод информации, подключалась параллельно сегментам индикатора, для которых выводы контроллера должны работать на вывод.

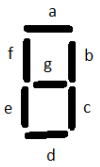
Глава 2. Проверка микроконтроллера (начало)

Чтобы не говорили, самый быстрый способ проверить микроконтроллер – это использовать графический язык программирования. Бытует мнение, что эти программы пригодны только для того, чтобы помигать светодиодом. Аппарат, с которым я столкнулся, далеко не светодиод, которым можно только помигать. Я знаю несколько программ с графическим языком программирования. Одна из них создана отечественным разработчиком С. Глушенко – это FLProg – и предназначена для работы с модулями Arduino, что не мешает, думаю, использовать её и создания программ для ряда микроконтроллеров AVR. И эту программу автор успешно использовал при написании программ для весьма сложных устройств. Но в моём случае удобнее было использовать другую программу, что подтвердилось позже.

Для поверки работы АЦП нужно выводить считываемые данные на индикатор. Поэтому начать проверку микроконтроллера следовало с организации работы индикаторов. Четыре семисегментных индикатора при обычном подключении потребовали бы более четырёх портов выводов от микроконтроллера. Поэтому их используют в режиме динамической индикации: сегменты подключают к одному порту, а общие выводы к четырём выводам другого порта; быстро переключая последовательно общие выводы, можно видеть все четыре индикатора, показывающие нужные цифры.

В программе Flowcode 5 для AVR выберем контроллер ATMega48. В разделе основного меню «Выходы» выберем 7seg4. На панели появится индикатор. В окне свойств, выделив индикатор, выберем раздел «Соединения», где подключим индикатор с общими анодами:

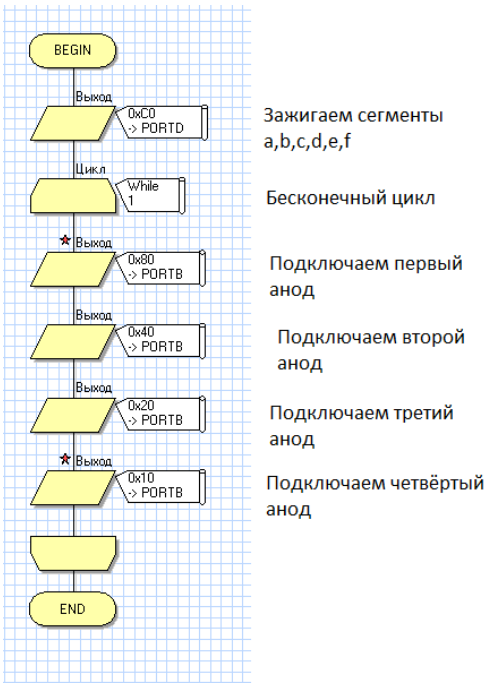




Расположение сегментов

Рис. 2.1. Подключение индикатора к МК

Чтобы зажечь все нули на индикаторе, в порт D, к которому подключены катоды, впишем шестнадцатеричное 0xC0. Затем в бесконечном цикле будем переключать аноды. Программа выглядит так:



Зажигаем сегменты a,b,c,d,e,f

Бесконечный цикл

Подключаем первый анод

Подключаем второй анод

Подключаем третий анод

Подключаем четвёртый анод

Рис. 2.2. Первая программа проверки МК

У программы Flowcode есть собственный отладчик, который мы будем использовать, но я предпочитаю отладку осуществлять в программе ISIS (Proteus), где позже можно будет добавить разные компоненты схемы.

Справедливости ради отмечу, что я использовал и отладчик Flowcode, и отладчик ISIS. В одних случаях удобнее использовать одну программу, в других лучше проверить работу всего устройства в Proteus. Я не поборник упорного использования только одной из программ, хотя понимаю, что длительная работа с одной программой даёт лучшее понимание всех её особенностей, даёт опыт работы, который трудно приобрести, если работать с программой только время от времени. Но в данном случае меня интересовал только конечный результат. А какими средствами он будет достигнут, это меня волновало в меньшей мере.

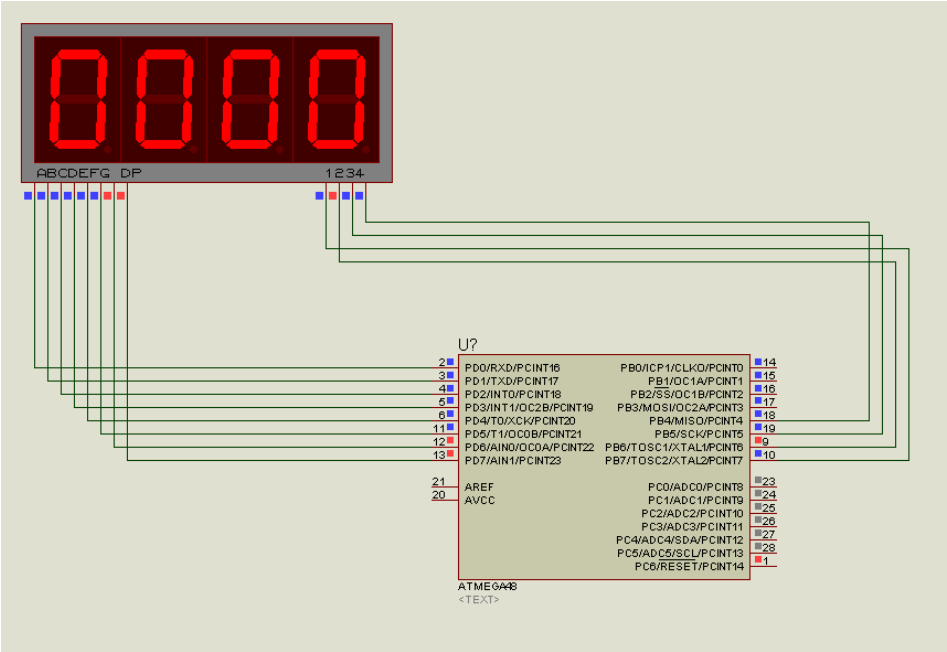


Рис. 2.3. Проверка первой программы в ISIS

Все индикаторы зажигаются и горят, но хотелось бы удостовериться, что они высвечивают правильную информацию. Для этого будем высвечивать на первом индикаторе единицу, на втором двойку и т.д. Чтобы понять, какое должно быть шестнадцатеричное число, высвечивающее нужную цифру, я использую калькулятор, который есть во всех операционных системах. В Windows следует выбрать вид «Программист». Сегмент «a» - это младший бит, а точка зажигается старшим битом. Для высвечивания единицы нужно зажечь (установить в ноль) биты первый и второй PD1 и PD2, оставив остальные в единице.

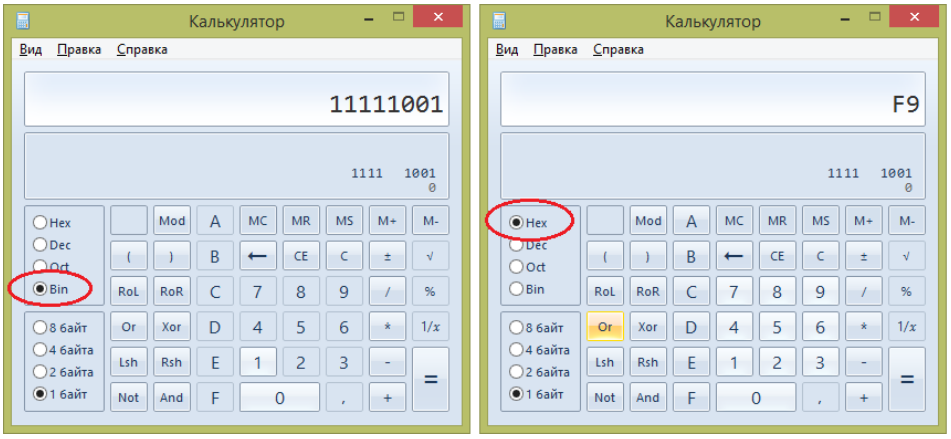
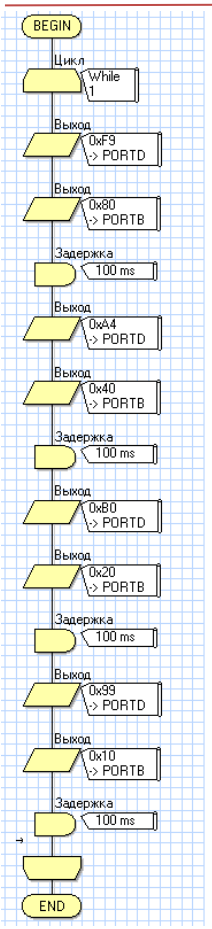


Рис. 2.4. Использование калькулятора для получения нужных цифр

Изменим программу, добавив паузы после считывания цифры:



Первая цифра

Вторая цифра и т.д.

Рис. 2.5. Изменённая программа для проверки каждой из цифр

Правда, при моделировании в программе Proteus цифры высвечиваются правильно, но по одной цифре, как, впрочем, и в отладчике Flowcode.

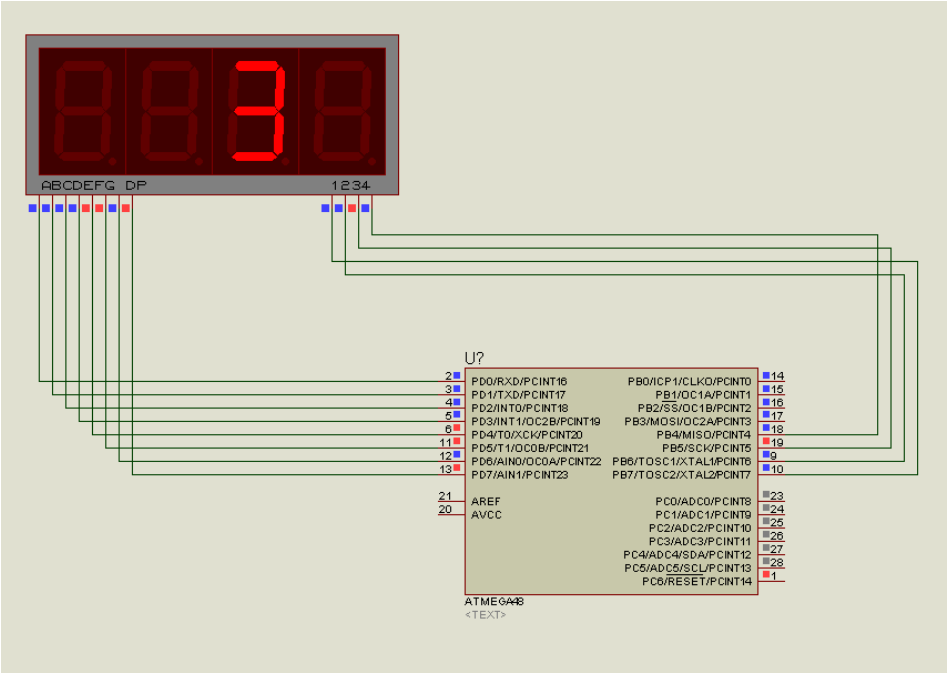


Рис. 2.6. Моделирование программы в ISIS

Мы пока использовали бесконечный цикл wile. В реальности, если мы так сделаем, то ничего другого программа сделать не сможет, застряв в этом бесконечном цикле. Чтобы получить изменения, но не использовать бесконечный цикл, мы используем прерывание по таймеру, которое будет переключать цифры, но не будет мешать работе основной программы.

В программе Flowcode в основной программе мы разрешаем прерывание по таймеру 0. Введя переменную *i* типа *byte*, присваиваем ей значение единица. А в бесконечном цикле основной программы будем обращаться к макросу (подпрограмме), который назван *show*. Что за макрос?

В этом месте мы меняем значения цифр на индикаторах, которые зависят от переменной *i*, принимающей значения от 1 до 4. Это можно сделать, используя конструкции *if…then.* Но мы используем более удобную конструкцию, которая называется *switch*.

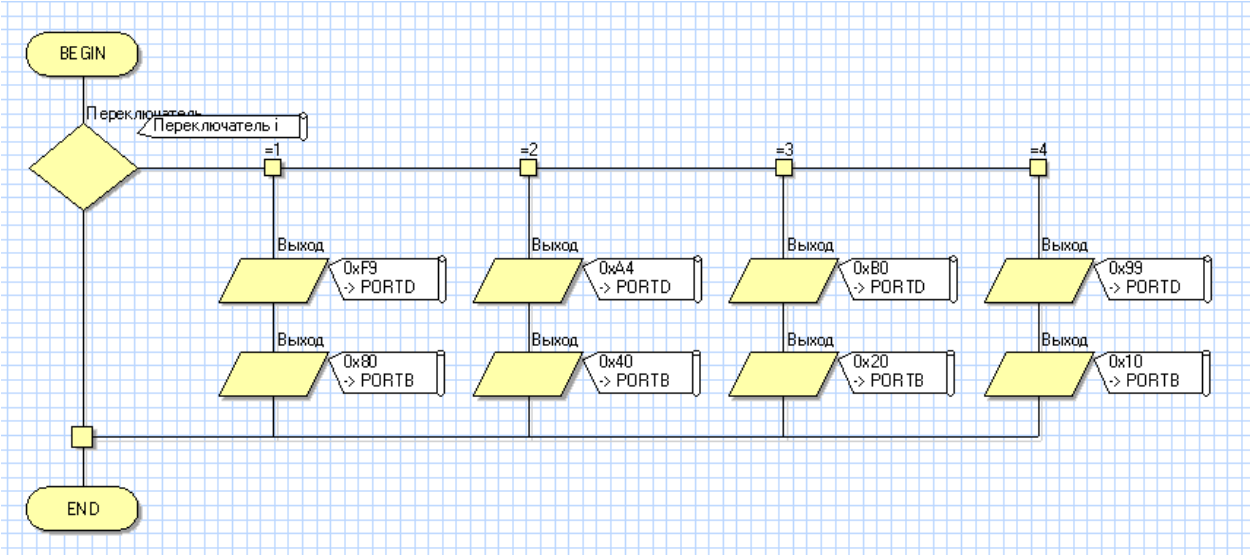


Рис. 2.7. Макрос, определяющий цифры четырёх индикаторов

Для обустройства переключателя в его диалоговом окне (после создания нового макроса и добавления *switch*) отмечаем четыре точки переключения:

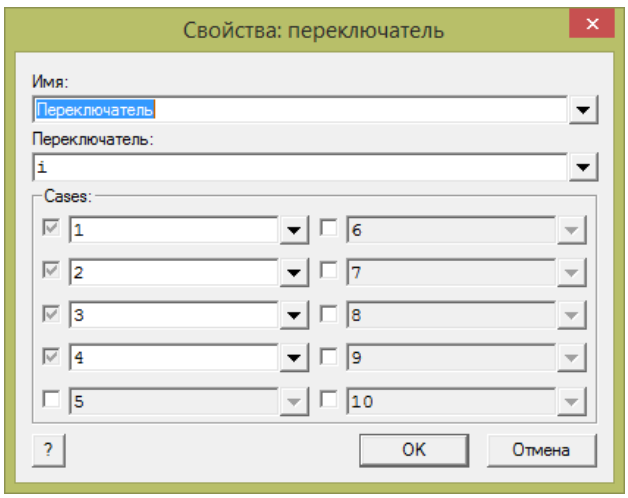


Рис. 2.8. Диалог переключателя switch

В окне «переключатель» указана переменная *i*, «заведующая» переключением. Ниже четыре значения этой переменной. А в ветви переключателя переносим те значения, которые раньше мы вычислили в предыдущей программе.

Прерывание должно обращаться к своему макросу, который мы назовём *intr_show*. Мы создаём новый макрос, который заполняем следующим образом:

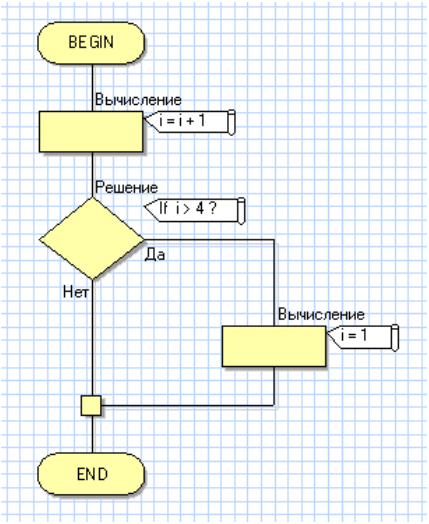


Рис. 2.9. Макрос, обслуживающий прерывание

При каждом прерывании переменная *i* увеличивается на единицу, пока переменная не станет больше 4.

А вот как выглядит основная программа:

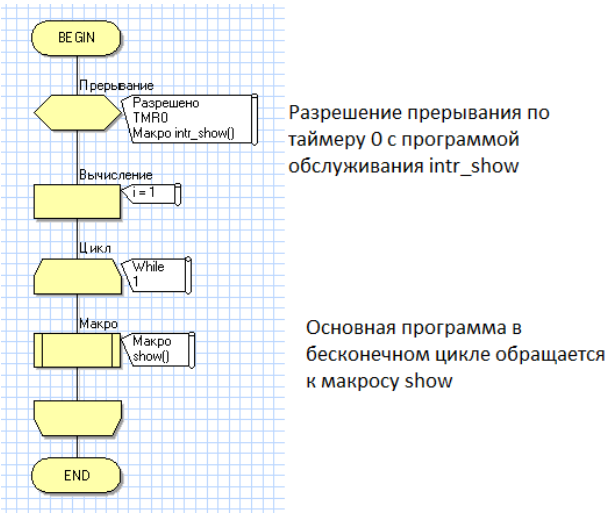


Рис. 2.10. Основная программа

В диалоговом окне прерывания можно выбрать, по какому таймеру или другому событию будет происходить прерывание, а для таймера выбрать частоту прерываний, используя делитель. И следует проверить, задана ли тактовая частота микроконтроллера, что можно сделать в разделе основного меню «Чип», где есть подраздел «Настройки проекта». Тактовая частота должна быть равна 8 МГц.

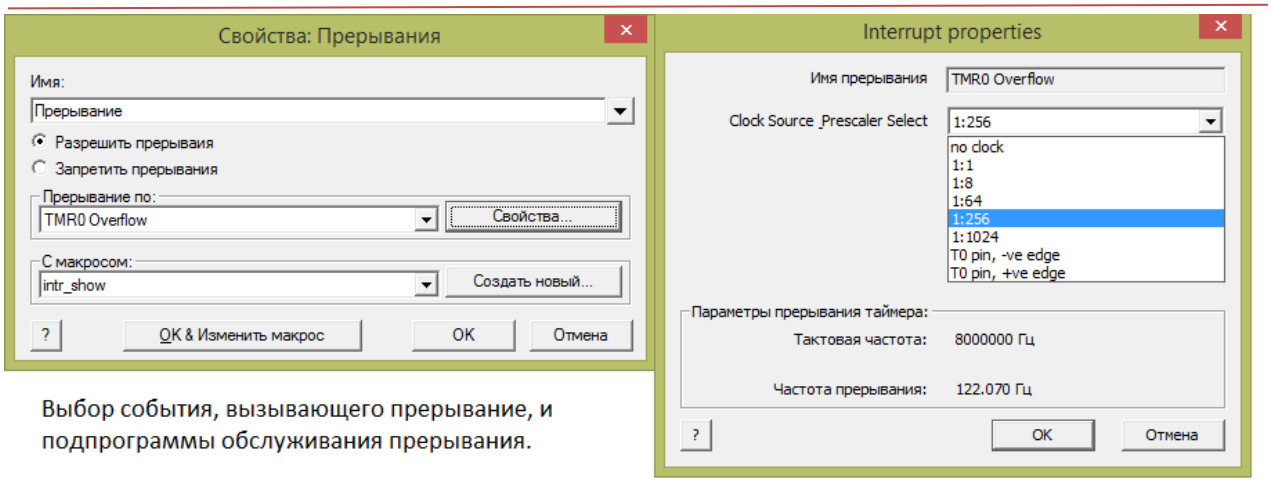


Рис. 2.11. Диалоговое окно настройки прерывания

Глава 3. Проверка АЦП

Мы подготовили индикаторы для отображения значений, получаемых от аналогово-цифрового преобразователя. Хотя для АЦП будет использоваться только два из четырёх индикаторов, в реальном аппарате их четыре.

В аппарате используются два канала АЦП: первый считывает напряжение с задатчика, формируя впоследствии требуемый ток в нагрузке, а второй считывает падение напряжения на резисторе, включённом в цепь нагрузки, чтобы на индикаторах отобразить реальный ток в нагрузке.

Первый канал считывает напряжение от 0 до 5 вольт. Второй канал считывает напряжение от 0 до 1,6 вольт. Для первого канала опорное напряжение, видимо, следует выбрать равным напряжению питания, а для второго канала в оригинальной схеме было организовано напряжение в 3 вольта. Поскольку после восстановления аппарата не требовалось восстановление его полной функциональности, подразумевающей использование нескольких диапазонов рабочих токов, а достаточно было бы одного, от 0 до 5 мА, в качестве второго опорного напряжения можно использовать внутренний источник Vref. При токе в 5 мА падение напряжения на измерительном сопротивлении не превышает 100 мВ, а внутренний источник даёт напряжение равное 1,1 В.

Итак, проверим работу первого канала. Для этого в программе Flowcode добавим АЦП. В разделе основного меню есть «Входы», где можно найти ADC. Этот встроенный модуль микроконтроллера появляется в виде ручки на панели:

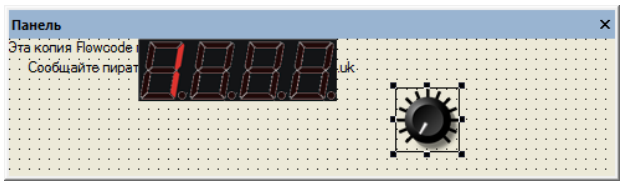


Рис. 3.1. Добавление на панель АЦП

После выделения АЦП (ручки на панели) в окне свойств можно выбрать расширенные свойства, где определяется ряд параметров АЦП.

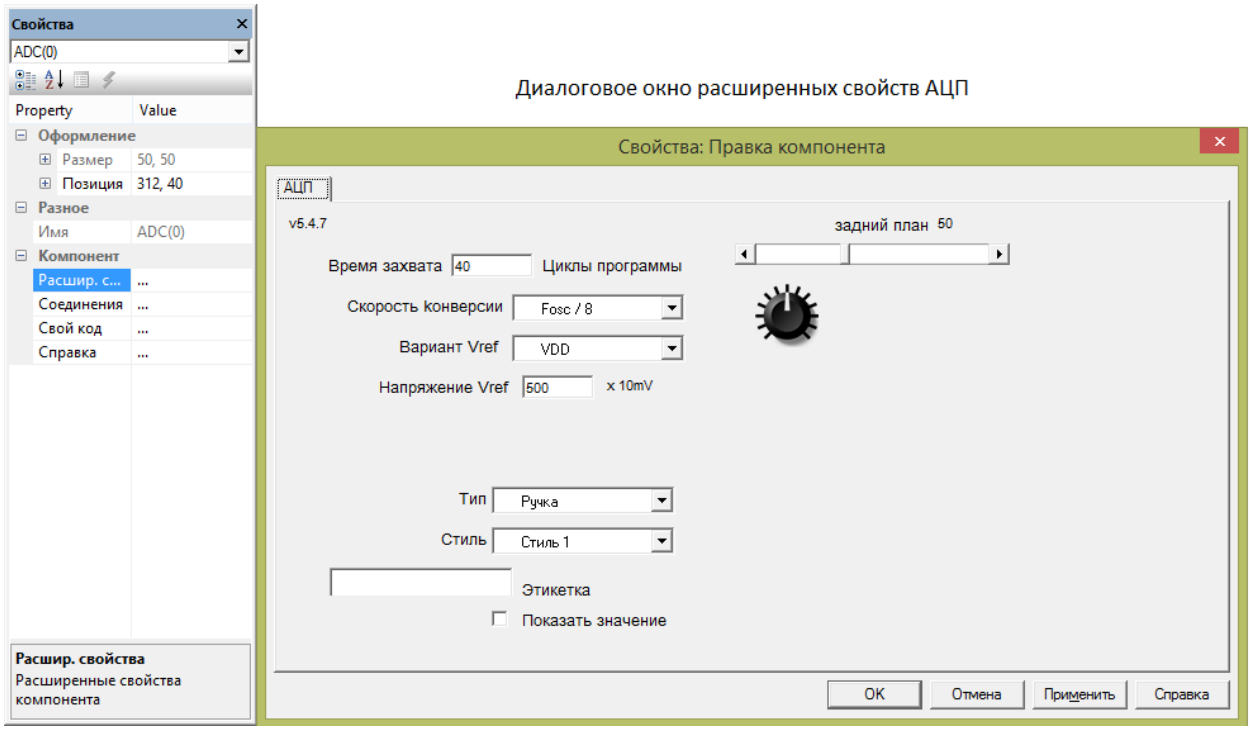


Рис. 3.2. Диалоговое окно расширенных свойств АЦП

Здесь нас интересует опорное напряжение. А в разделе свойств «Соединение» мы укажем канал, к которому подключен задатчик.

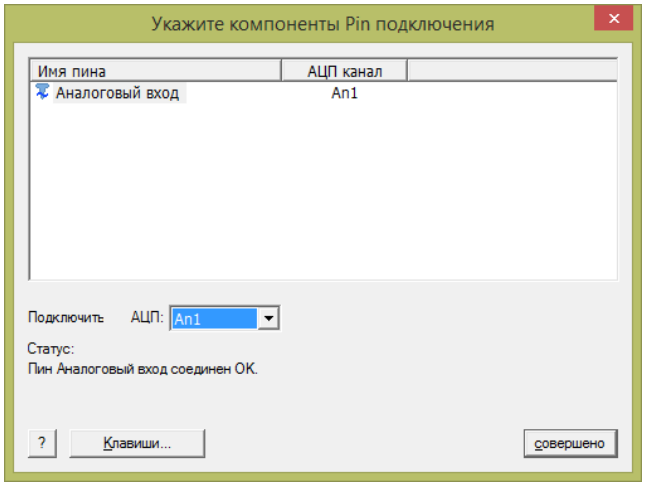


Рис. 3.3. Диалог соединения

АЦП позволяет реализовать разные режимы считывания значения напряжения. Выберем считывание значения как напряжения. Но прежде добавим в основную программу макрос АЦП:

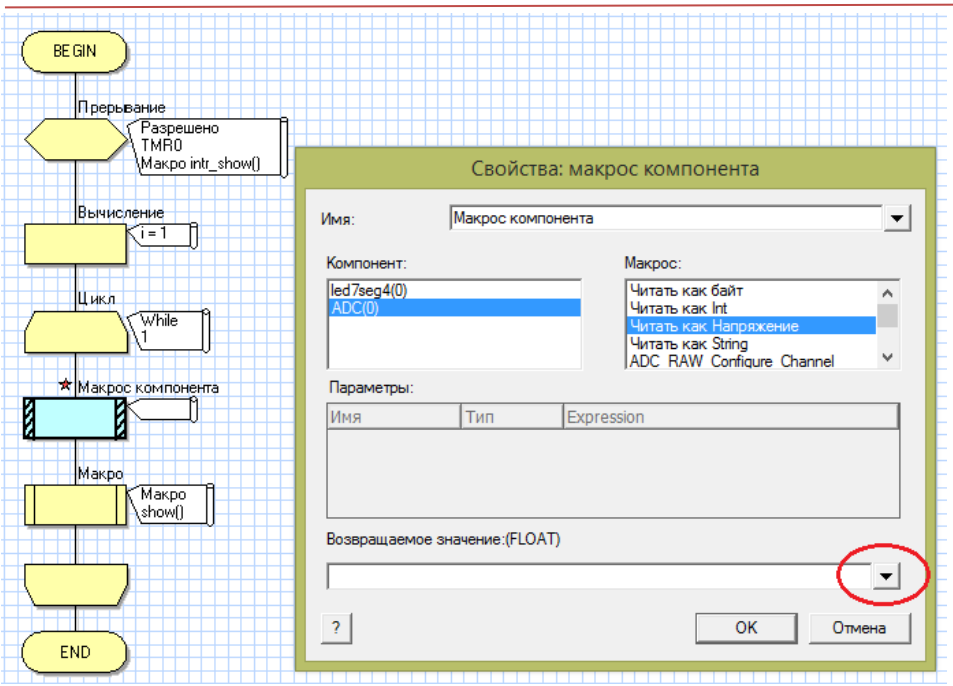


Рис. 3.4. Добавление макроса компонента и диалог его настройки

Как видно из диалога, нам понадобится переменная типа *float* (с плавающей точкой), которая будет значением, возвращаемым из процедуры. Назовём её *volt*, а для создания новой переменной нажмём отмеченную на рисунке кнопочку. Этим открывается диалог выбора переменной для оформляемой нами операции. Щелчком правой клавиши мышки по названию «Переменные» можно вызвать диалог создания новой переменной (Add new, добавить новую).

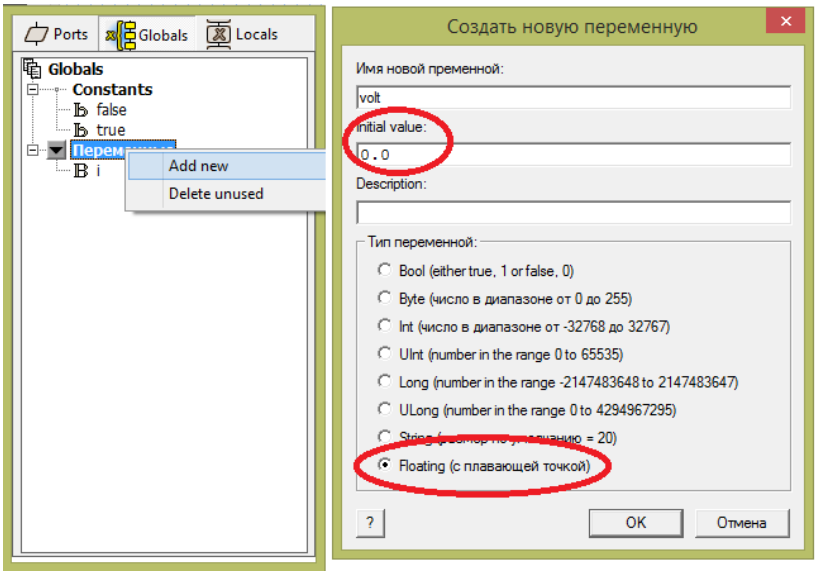


Рис. 3.5. Создание новой переменной в диалоге настройки АЦП

Кнопка **ОК** закрывает этот диалог, возвращая нас к выбору переменной, что можно завершить двойным щелчком левой клавиши мышки по вновь созданной переменной *volt*, которая и станет возвращаемым значением при считывании напряжения. Осталось закрыть диалоговое окно настройки макроса компонента, что позволяет сделать очередная кнопка **ОК**.

Мы настроили АЦП, но что мы получили в итоге? Проще всего проверить это в программе Flowcode при пошаговой отладке программы.

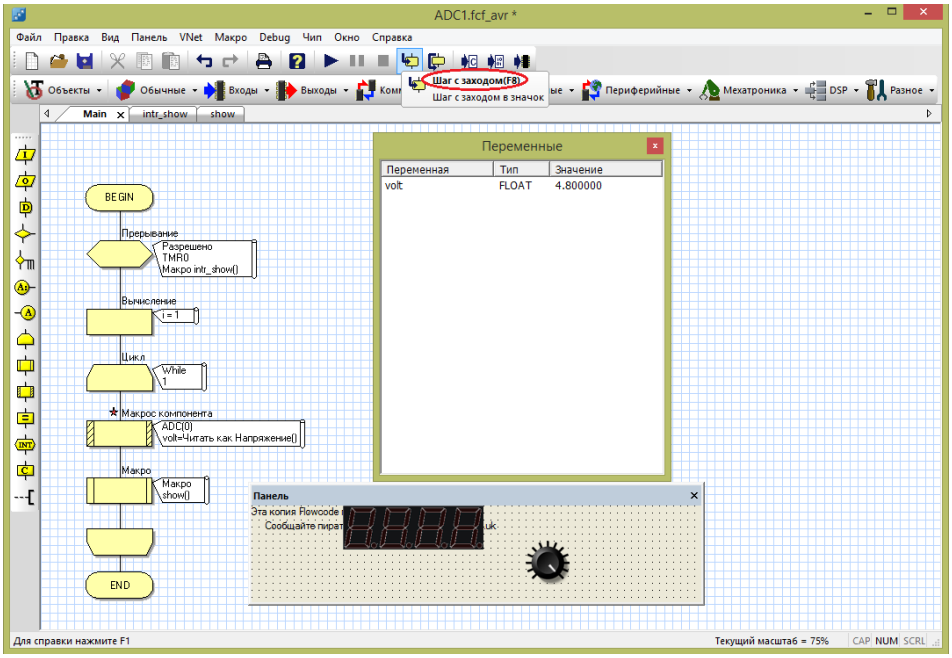


Рис. 3.6. Пошаговая отладка чтения напряжения

В окне «Переменные», которое появляется при начале отладки, достаточно щёлкнуть правой клавишей мышки, чтобы в новом диалоге двойным щелчком левой клавиши мышки по интересующей нас переменной добавить её для наблюдения. Продолжая шагами проходить программу, «повернув» мышкой ручку АЦП на панели, можно увидеть значение переменной *volt*.

Однако пока мы убедились только в том, что мы читаем напряжение на входе АЦП. Хотелось бы увидеть это значение и на индикаторах. Я не исключаю, что есть другие решения по отображению чисел на семисегментных индикаторах, но я их не знаю, поэтому использую то, что знаю.

Значение переменной *volt* я намерен разбить на две цифры, первую из которых назову *volt_high*, а вторую *volt_low*; для них создадим две новые переменные типа *byte*. Первую цифру *volt_high* я отправлю на первый из двух последних индикаторов, а вторую на второй. Для этого используем программный компонент Flowcode, который называется «Вычисление». Добавим его в основную программу, откроем окно свойств двойным щелчком левой клавиши по нему и запишем все операции:

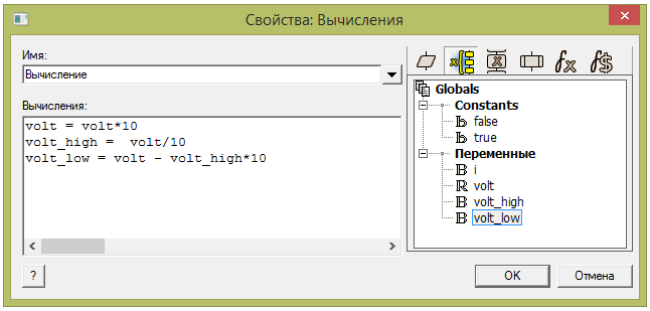


Рис. 3.7. Вычисление цифр, отображающих напряжение на индикаторах

Переменные можно вписывать в окно вычислений, а можно ввести, дважды щёлкнув по переменной левой клавишей мышки в окне отображения переменных. И обратите внимание, что все переменные, которые мы используем – это глобальные переменные. В некоторых случаях удобнее использовать локальные переменные, что даст экономию оперативной памяти микроконтроллера, в других случаях удобнее использовать глобальные переменные. Выбор следует предварительно продумать, чтобы не сделать ошибки.

Завершив описание вычислений, можно в пошаговом режиме проверить, всё ли будет правильно:

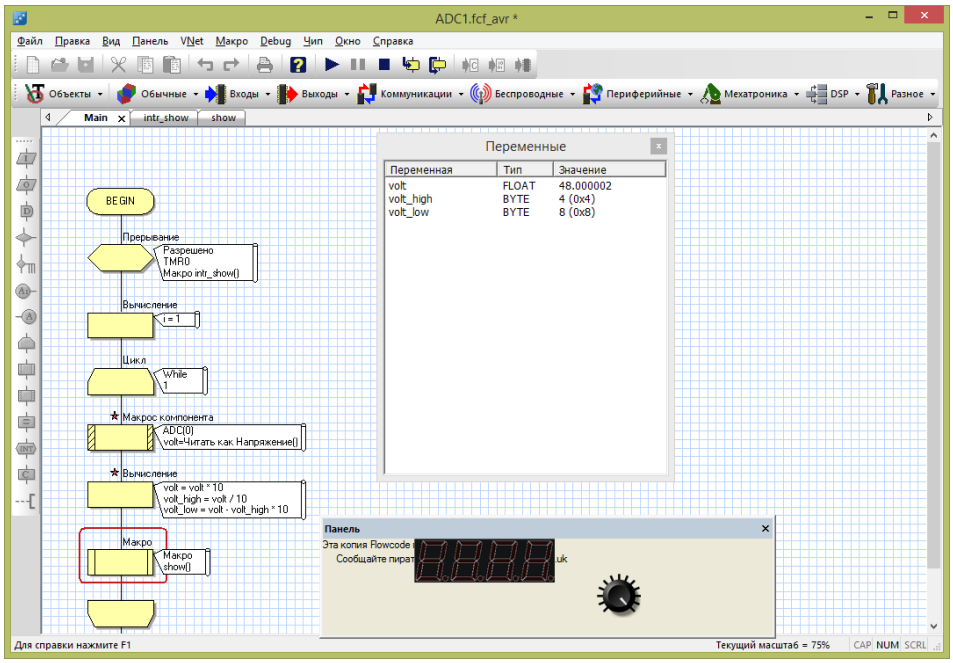


Рис. 3.8. Проверка вычислений в пошаговом режиме отладки

Осталось подправить макрос *show*, чтобы отображать новые переменные.

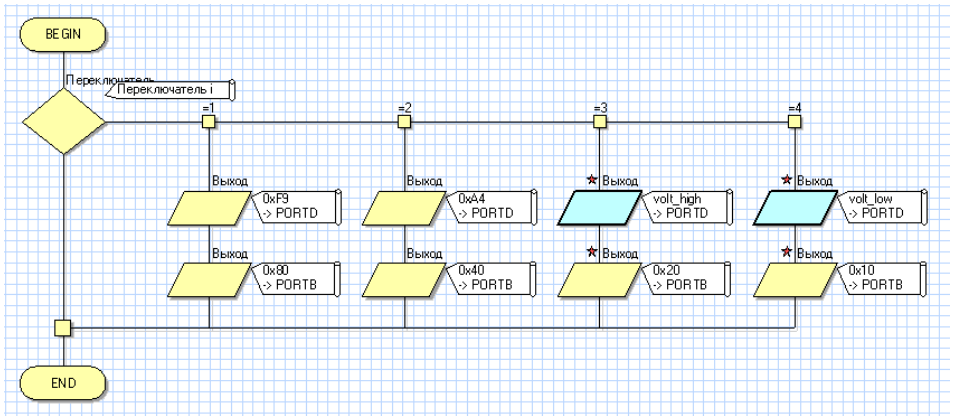


Рис. 3.9. Исправление макроса *show* для отображения напряжения

Проверить правильность работы программы можно в пошаговом режиме, но при этом следует учесть, что прерывание в этом режиме может работать не так, как хотелось бы. Чтобы этого избежать, перед прохождением макроса *show* можно менять значение переменной *i*, которая определяет, какой индикатор будет показан. Для этого перед входом в макрос show в окне переменных, где мы (надеюсь) добавили наблюдение за переменной *i*, щёлкнем правой

клавишей мышки по этой переменной, а в появившемся окне выберем изменение значения переменной, которое и осуществим в следующем окне.

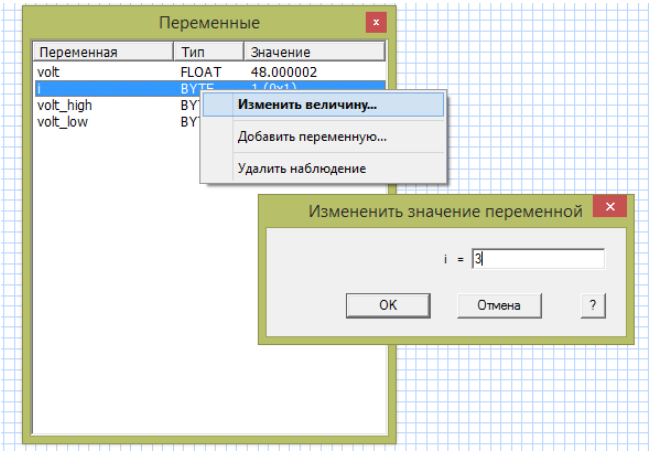


Рис. 3.10. Изменение значения переменной

Но мы забыли ещё одну операцию – преобразование цифры в значение для семисегментного индикатора. Выполним это.

Создадим ещё одну процедуру (макрос), в которой будем преобразовывать значения цифр соответственно требованиям подключения сегментов. Назовём макрос *dig_trans* и добавим переменную *k* типа *byte*, по значению которой будем переключать цифры. Используем, как и раньше, переключатель *switch* по переменной k. Нам потребуется десять (от 0 до 9) ветвей в переключателе. Обратите внимание, что вместо 10 следует использовать 0.

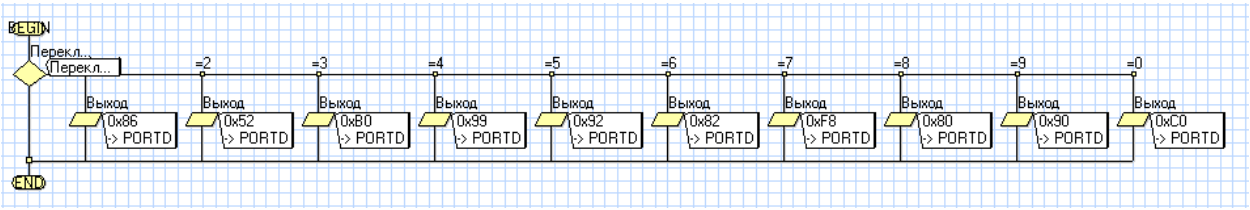


Рис. 3.11. Содержание макроса *dig_trans*

Подправим и макрос *show*:

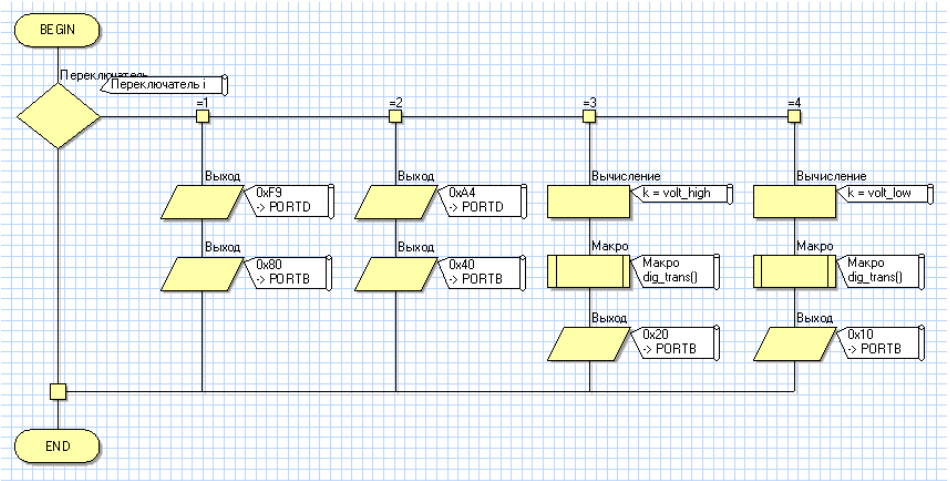


Рис. 3.12. Исправленный макрос *show*

Последнее, что можно исправить – добавить небольшую паузу (20 микросекунд) после считывания АЦП.

Если запустить программу на выполнение, то цифры меняются последовательно, но соответствуют тем, что показывают переменные, в чём можно убедиться, меняя положение ручки АЦП.

Глава 4. Как формируется ток в нагрузке

Как сказано выше, для формирования тока в нагрузке использована широтно-импульсная модуляция. Вывод, на который выводится сигнал, это PB1.

К уже созданной программе добавим программный элемент PWM, который находится в разделе основного меню «Мехатроника». В расширенных свойствах нового компонента настроим номер канала и период.

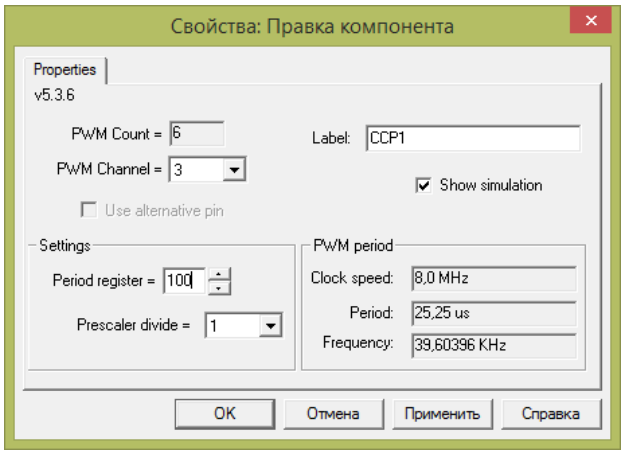


Рис. 4.1. Настройка расширенных свойств PWM

Добавим макрос этого компонента в основную программу. И настроим вначале период PWM.

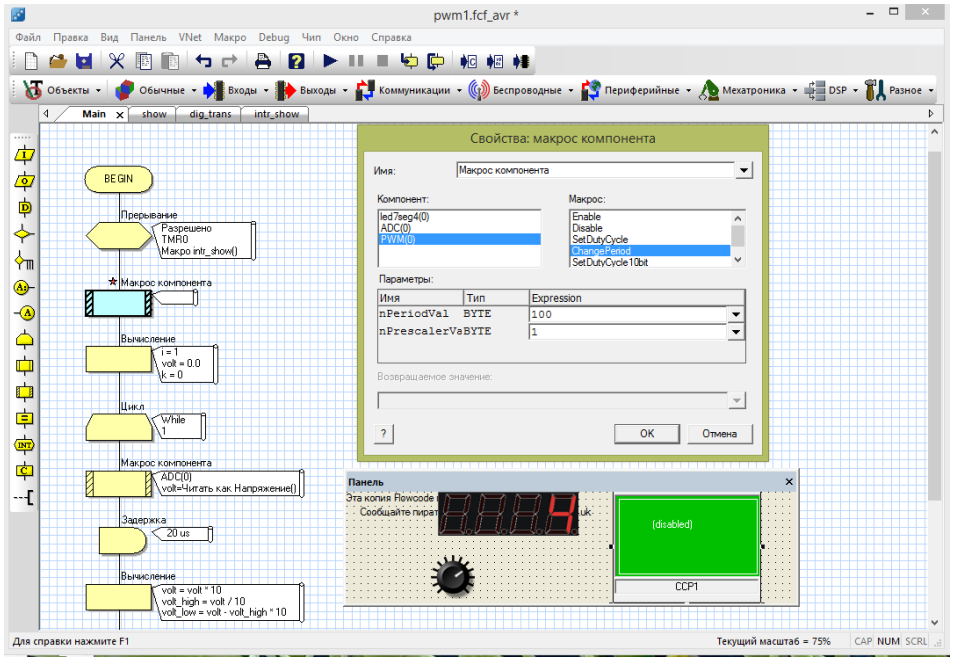


Рис. 4.2. Настройка периода PWM

Настроив период, озаботимся тем, чтобы настроить скважность импульсов, которая будет управляться переменной *volt*, считываемой с задатчика.

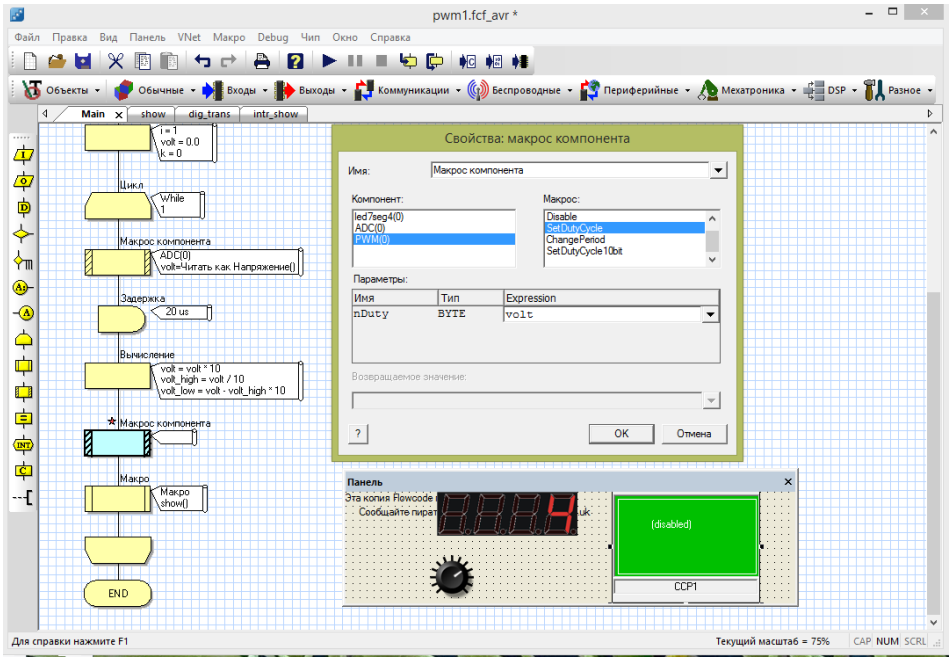


Рис. 4.3. Настройка скважности импульсов PWM

Проверим работу формирователя импульсов в программе Flowcode. Осталось разрешить работу формирователя импульсов... и запустить отладку.

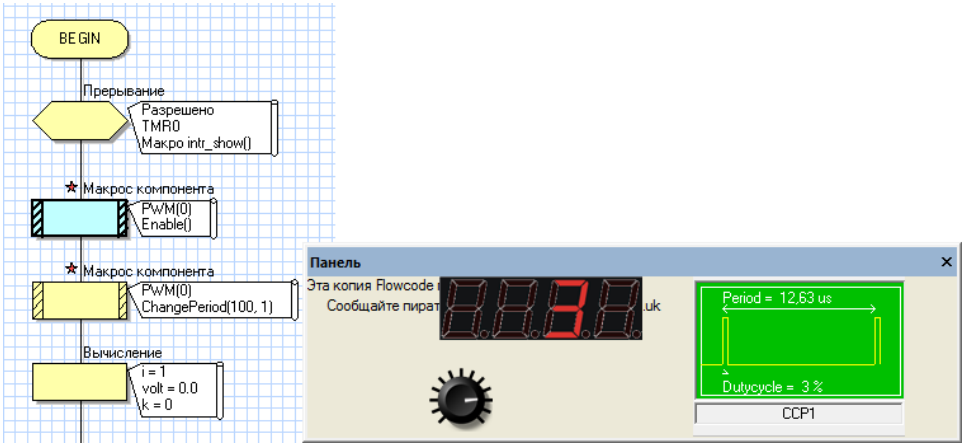


Рис. 4.4. Разрешение работы PWM и проверка работы

Меняя положения ручки АЦП, можно убедиться, что длительность импульса меняется. Можно выполнить ещё одну операцию, округлив значение, считываемое в переменную *volt*. Для этого добавляется ещё одна переменная типа *byte* и компонент «Вычисление»:

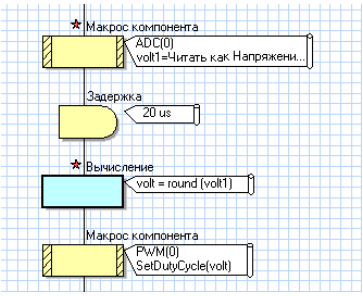


Рис. 4.5. Округление считываемого АЦП значения

Глава 5. Как управляется ток в нагрузке

Сигнал с вывода PWM должен попасть на интегрирующую RC цепь. Затем, что разработчики и сделали, напряжение с интегрирующей цепи подаётся на вход операционного усилителя, включённого повторителем напряжения.

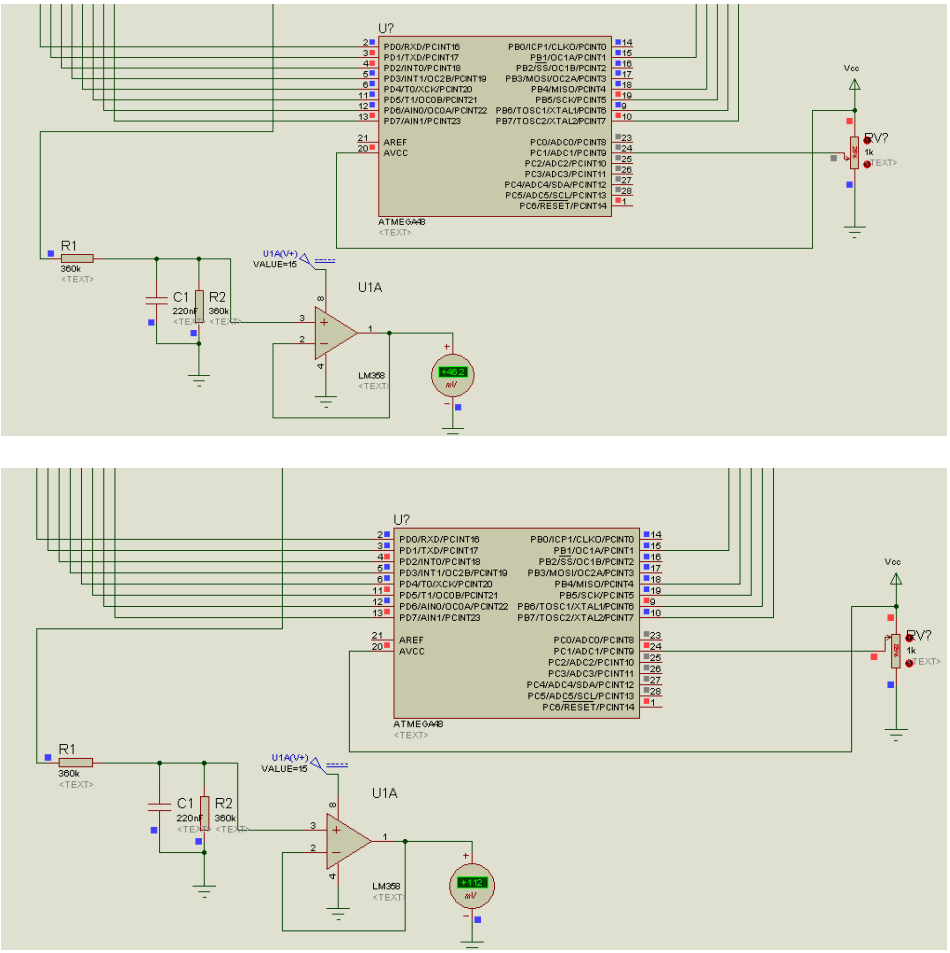


Рис. 5.1. Интегрирование сигнала PWM

А вот со второй половиной микросхемы сдвоенного операционного усилителя разработчики поступили, с моей точки зрения, очень остроумно. Напряжения в 100 мВ недостаточно для управления транзистором IRF610, который и является управляющим элементом. Можно, конечно, второй половиной микросхемы воспользоваться для усиления. Но оригинальная схема использует несколько диапазонов регулировки тока, что потребовало бы, вероятно, коррекции коэффициента

усиления. Мало того, напряжение управления меняется в достаточно широких пределах, тогда как напряжение на затворе управляющего транзистора должно меняться в более узком диапазоне.

Транзистор в качестве управляющего элемента в схеме использован как резистор, управляемый напряжением. Можно проверить, что при перемещении задатчика из одного крайнего положения в другое напряжение управления меняется в 4-5 раз. Но взгляните на график зависимости тока транзистора от напряжения на затворе, взятый из справочных данных.

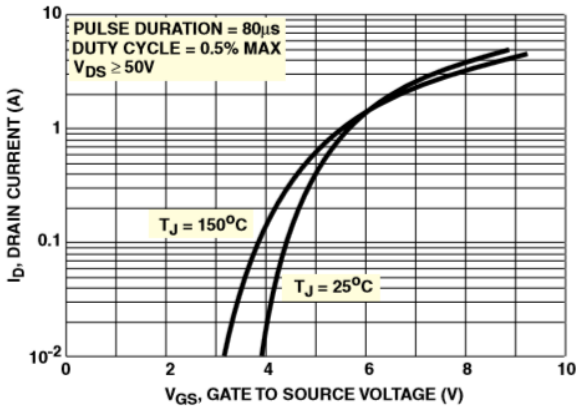


FIGURE 7. TRANSFER CHARACTERISTICS

Рис. 5.2. График зависимости тока транзистора от напряжения на затворе

При изменении тока от 10 мА до 100 мА напряжение на затворе меняется в 1,1 раз.

Готов признаться, что если бы мне предложили решить подобную задачу, я оказался бы в очень затруднительном положении, не зная, как её решить.

Но вот, как поступили разработчики этого устройства.

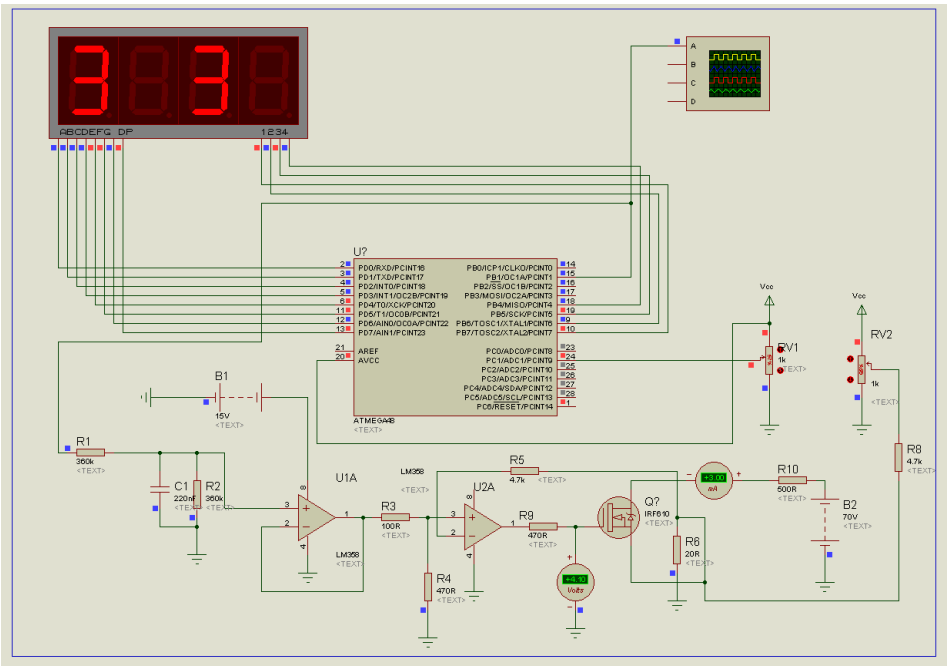


Рис. 5.3. Схема управления транзистором

Используя резистор R6, по которому протекает ток нагрузки, в качестве элемента отрицательной обратной связи они получили требуемое решение. Их ли это находка, нет ли, я не знаю, но думаю, что не стал бы писать этот рассказ, если бы не был восхищён столь простым и исчерпывающим проблему решением.

В программу потребуется внести ещё ряд дополнений. В частности:

1. Первые два индикатора отображают заданное таймеру время в минутах.
2. Время, заданное таймеру, устанавливается с помощью клавиш.
3. После начала процесса, когда таймер обнулится, нужно выключить ток в нагрузке и подать звуковой сигнал.

Моделируя программу в Flowcode и в Proteus, можно заметить разницу: работающая в первом случае программа, может не заработать в Proteus. Вдобавок, как об этом я упоминал раньше, общие выводы индикаторов подключены к выводам микроконтроллера через транзисторы. Можно ли это повторить при моделировании в Flowcode, я не знаю. И ещё – при моделировании обе программы показывают отображаемые значения на индикаторах не так, как должно быть в реальности. Поэтому время от времени я загружаю программу в микроконтроллер реального устройства и проверяю, что у меня получилось на данный момент.

Проверяя в очередной раз результаты работы, я обнаружил, что лимит программной памяти, отпущенный природой микроконтроллеру ATMega48, превышен. Естественно, программа не может быть загружена, о чём сообщает программа SinaProg, которую я использую с простым параллельным программатором, идентифицируемым как Stk200, состоящим из нескольких резисторов и проводков для внутрисхемного программирования. Программатор помог мне снять защиту вместе с исходной программой, но в данном случае помочь ничем не может.

Если вы сталкивались с подобной проблемой, то можете воспользоваться тем приёмом, который применил я. А именно: я загрузил с сайта разработчика программу Atmel Studio 6. Эта среда разработки использует тот же компилятор, что и программа Flowcode. Но не подключает всё из библиотеки, а только нужное. Достаточно после трансляции в программе Flowcode скопировать полученный Си-файл и вставить его в исходный файл AtmelStudio, как при трансляции файла в hex-файл можно увидеть, что:

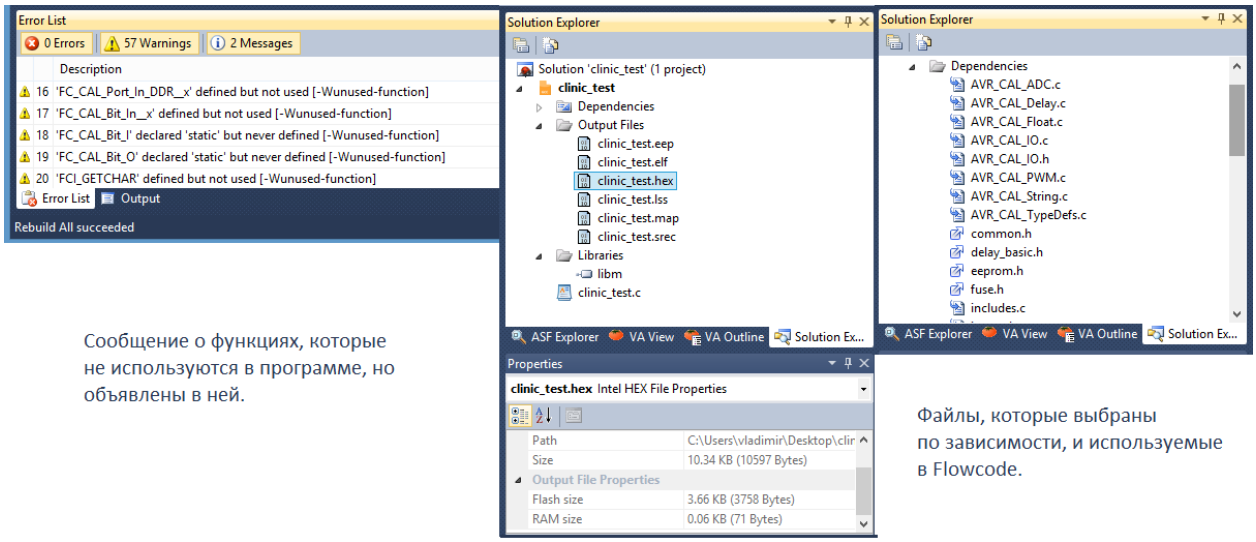


Рис. 5.4. Трансляция файла в Atmel Studio

Функции, которые программа не использует, Atmel Studio не транслирует, поэтому hex-файл получается более компактным. Удобство здесь в том, что можно продолжить работу над программой в Flowcode, а транслировать её в Atmel Studio, но ни текст программы на Си, ни переделки библиотечных файлов не требуется. Atmel Studio использует все файлы, заданные в программе Flowcode, но выбирает из них только то, что нужно для работы программы.

Глава 6. Клавиатура

Четыре клавиши, используемые для управления аппаратом, доставили мне более всего хлопот.

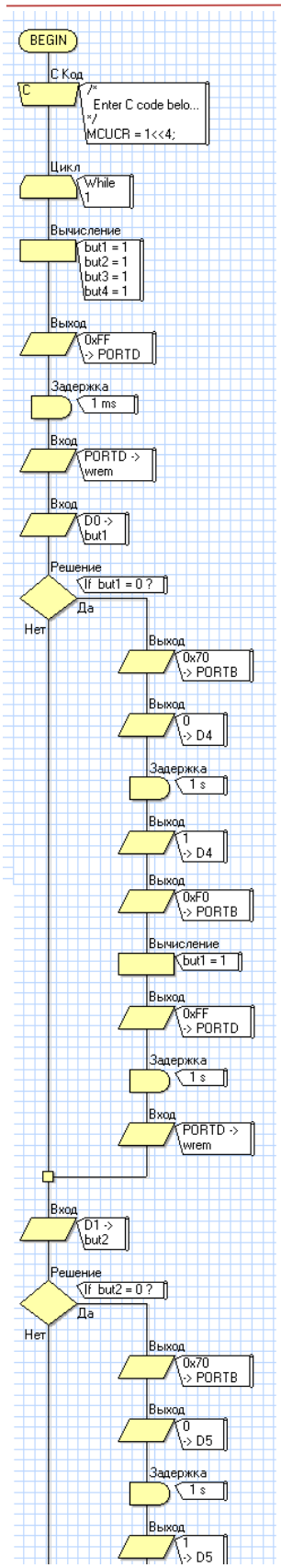
По сути, четыре кнопки, которые через резисторы при нажатии подключаются к «земле», самая распространённая конструкция. И даже не то, что кнопки подключены к младшим битам порта, которые обслуживают сегменты индикаторов, смутило меня – использовать одни и те же выводы на ввод и на вывод никто не запрещает. Смутило меня то, что кнопки подключаются к общему проводу через сопротивления в 5 к. Первоначально я неправильно прочитал номинал, решив, что это 510 Ом. И некоторое время не мог понять, почему кнопки не работают.

Из-за отсутствия схемы, из-за спешки я не удосужился проанализировать, почему сделано так, но стало понятно, зачем общие выводы индикаторов подключены через транзисторы. Выключая транзисторы, запретив использование подтягивающих резисторов, можно сделать входы «висящими в воздухе», что позволяет прочитать нажатые кнопки как ноль. Я не исключаю и того, что всё выше написанное не более, чем моё очередное заблуждение. Но с помощью мультиметра я пытался понять, когда кнопки дают достаточно низкое напряжение, чтобы обнулить входы, и освобождение входов было единственным, что помогло считывать клавиши. При этом даже то, что в какой-то момент я держал плату в руке, мешало работе клавиатуры.

Признаться, мне очень хотелось заменить резисторы, оставив сопротивление 200-300 Ом, чтобы проверить, будет ли клавиатура работать с подтягивающими резисторами. Но несколько раньше я уже сталкивался с ошибкой, которая могла вывести схему из строя, поэтому экспериментировать не стал.

Идея, как я её понял, была в том, чтобы выключить транзисторы, запретить использование подтягивающих резисторов и переопределить четыре младших бита на вход.

Проверяя работу микроконтроллера, я, конечно, старался использовать короткие программы, выполняющие только определённые функции, нужные для работы. Поэтому считывание клавиатуры первоначально выглядело небольшой программой, которую я загрузил в МК и проверил (здесь компьютер не помощник) с реальным устройством. Несколько попыток придать программе более «приличный» вид окончились неудачей. Отчего-то не захотела работать клавиатура при считывании всего порта с маской и переключением по результату считывания. Наверное, следовало выяснить, в чём проблема, но каждая проверка требовала разборки и сборки, что, согласитесь, немного утомительно. И я смирился с тем, что работало. Вот часть этой тестовой программы:



Запрет использования подтягивающих резисторов.

Все переменные для считывания кнопок в единицу.

Выключение порта.

Считывание порта в переменную wrem.

Опрос первой кнопки.

Если она обнулила вход...

Включение индикаторов.

Индикация срабатывания кнопки.

Выключение транзисторов общих выводов индикторов.

Выключение порта сегментов.

Считывание порта.

Опрос второй кнопки.

Если вторая кнопка нажата.

И т.д.

Рис. 6.1. Часть программы считывания кнопок

После решения со считыванием кнопок осталось подобрать частоту переключения звуковой сигнализации близкую к резонансной частоте излучателя, включить и выключить реле в рабочей

цепи и ещё некоторые мелочи, связанные с организацией работы таймера, добавить точку для двух индикаторов, использовать массив для записи цифр и т.п.

Заключение

Не всегда разумные предположения дают разумные результаты. Но и из своих ошибок можно постараться извлечь полезный опыт. Так, например, я никогда не сталкивался, хотя и слышал, с проблемой снятия битов защиты. Оказалось, что параллельный программатор, подключаемый к порту LPT и состоящий из нескольких резисторов и проводков, позволяет легко это сделать. То есть, если вы по ошибке заблокировали микросхему, то в режиме внутрисхемного программирования с помощью такого программатора можете восстановить работоспособность контроллера.

Многие специалисты считают, что программа Flowcode хороша только для создания ёлочных гирлянд. Я убедился, что с ней вполне можно работать и при создания программы к более сложным устройствам.

У каждой программы есть свои недостатки. Так Flowcode не слишком заботится о памяти реального микроконтроллера. Но эту проблему можно обойти с помощью Atmel Studio, которая, кстати, не позволяет создавать отладочный вариант программы, если памяти не хватает.

В рассказе я использовал отображение напряжения с задатчика, в реальном устройстве использован ещё один канал АЦП, считывающий падение напряжения с резистора в рабочей цепи, что и послужило причиной моей изначальной ошибки, но добавило уважения к разработчикам этого аппарата.

Не думаю, что мне удалось бы сделать всё быстрее, начни я работу с написания программы на языке Си, а тем более на ассемблере. Программа, возможно, получилась не самой красивой, но мне нужно было от неё только то, чтобы она работала. И она заработала.