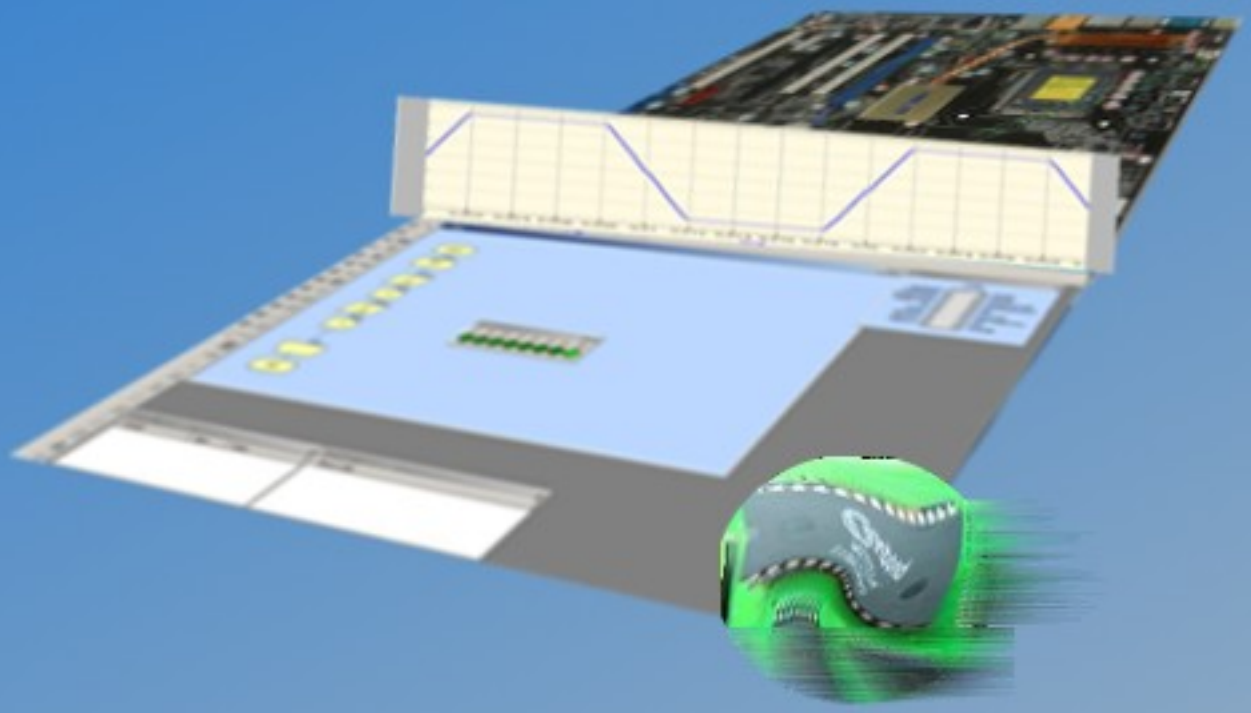
В.Н. Гололобов

Пинг-понг



Москва 2008

Оглавление

Я частенько заглядываю в свой старенький, слега потрескавшийся почтовый ящик, установленный на обочине моего сайта в то время, когда стройка этого сайта на Yandex'е только начиналась. Я не жду писем, но всегда обеспокоен тем, что могу не ответить кому-то на возникший вопрос по поводу моих рассказов, опубликованных на сайте, или не ответить на чью-то просьбу о помощи. Не думаю, что моя помощь эффективна, но письма, оставшиеся без ответа, не только разочаровывают или раздражают, они оскорбительны. Я рад пришедшим письмам — они позволяют ориентироваться в интересах читателей, в полноте изложения: многое для меня столь очевидно, что мне не приходит в голову детально описывать штатные процедуры, а для кого-то это может стать камнем преткновения; я предпочитаю дать контур предмета, указать пути к цели, чтобы читатель самостоятельно прошел весь путь, сделал собственные открытия, а не смотрел моими глазами, но не у всех хватает сил на самостоятельное путешествие. Полученные письма, порой, заставляют меня еще раз обращаться к уже написанному, но не в смысле исправлений — занятие почти бесполезное — а в плане написания нового рассказа о том же предмете, но в другом ракурсе. Если вам приходилось создавать какое-нибудь электронное устройство, вы знаете, что хочется быстрее спаять его, начать наладку, проверку, но не всегда получается так. Приходится возвращаться к схеме, выяснять, как она работает, вновь обращаться к макету и приборам. Все это напоминает игру в настольный теннис: пинг-понг, пинг-понг и мимо стола...

Микроконтроллер в программе KTechlab (Linux)

У меня в столе есть микросхема контроллера PIC16F628A. Не самый «крутой», не самый дорогой, этот контроллер позволяет мне проверить на макете правильность работы программ, о которых я пишу, и работу программатора, что, подчас, тоже оказывается маленькой, но проблемой. Я не фанат какого-либо контроллера или производителя, убежден, что при выборе микроконтроллера, как при выборе любой элементной базы, следует учитывать множество факторов, но те истории, что я сочиняю, я адресую начинающим, для которых выбор конкретной микросхемы — дело будущего, и, почти все, о чем идет речь, исключая, может быть, некоторые детали, не соотносятся ни с конкретным микроконтроллером, ни с конкретным производителем. Хотя, например, программа KTechlab ориентирована на PIC-контроллеры, как и Piklab, и MPLAB, о которых речь пойдет ниже.

Программа есть в дистрибутивах Fedora (а, значит, ASPLinux) и Ubuntu (Debian), есть и в ALTLinux, которым я недавно заменил один из дистрибутивов Fedora, думаю, есть в Mandriva. Кроме того, есть возможность использовать Linux-программы в Windows, как я использую Windows-программы в Linux. Некоторые «недоразумения» бывают, но, надеюсь, это не помешает моему рассказу.

Программа KTechlab, если ее сравнивать с FlowCode, может показаться менее удобной, когда речь идет об отладке программы для микроконтроллера. Но у нее есть и свои преимущества, о которых я постараюсь рассказать.

Будем предполагать, что KTechlab есть в списке программ, которые обнаруживаются либо на одном из дисков дистрибутива, либо в Интернете, когда запускаешь одну из утилит (в Linux их может быть несколько) «Установки и удаления программ». Если это так, то достаточно выбрать KTechlab из списка, пометить для установки и запустить установку, чтобы программа получилась готовой к применению. Правда, по зависимостям может потребоваться установка, скажем, gpsim — программы отладки, и gputils — компилятора ассемблера и ряда дополнений.

KTechlab предназначена не только для работы с микроконтроллерами. На вкладке компонентов программы можно увидеть достаточно представительную элементную базу, позволяющую познакомиться и с аналоговой, и с цифровой техникой, если при создании нового проекта выбрать проект Circuit (схема).

Не всякие, но простые готовые программы для микроконтроллера можно проверить в работе, например, с транзисторной «довеской», не выходя из KEechalab. В этом случае микроконтроллер используется наравне с другими элементами схемы, а для «загрузки» программы в контроллер используется отладочный .cod файл, который генерируется при компиляции исходного кода в hex-файл.

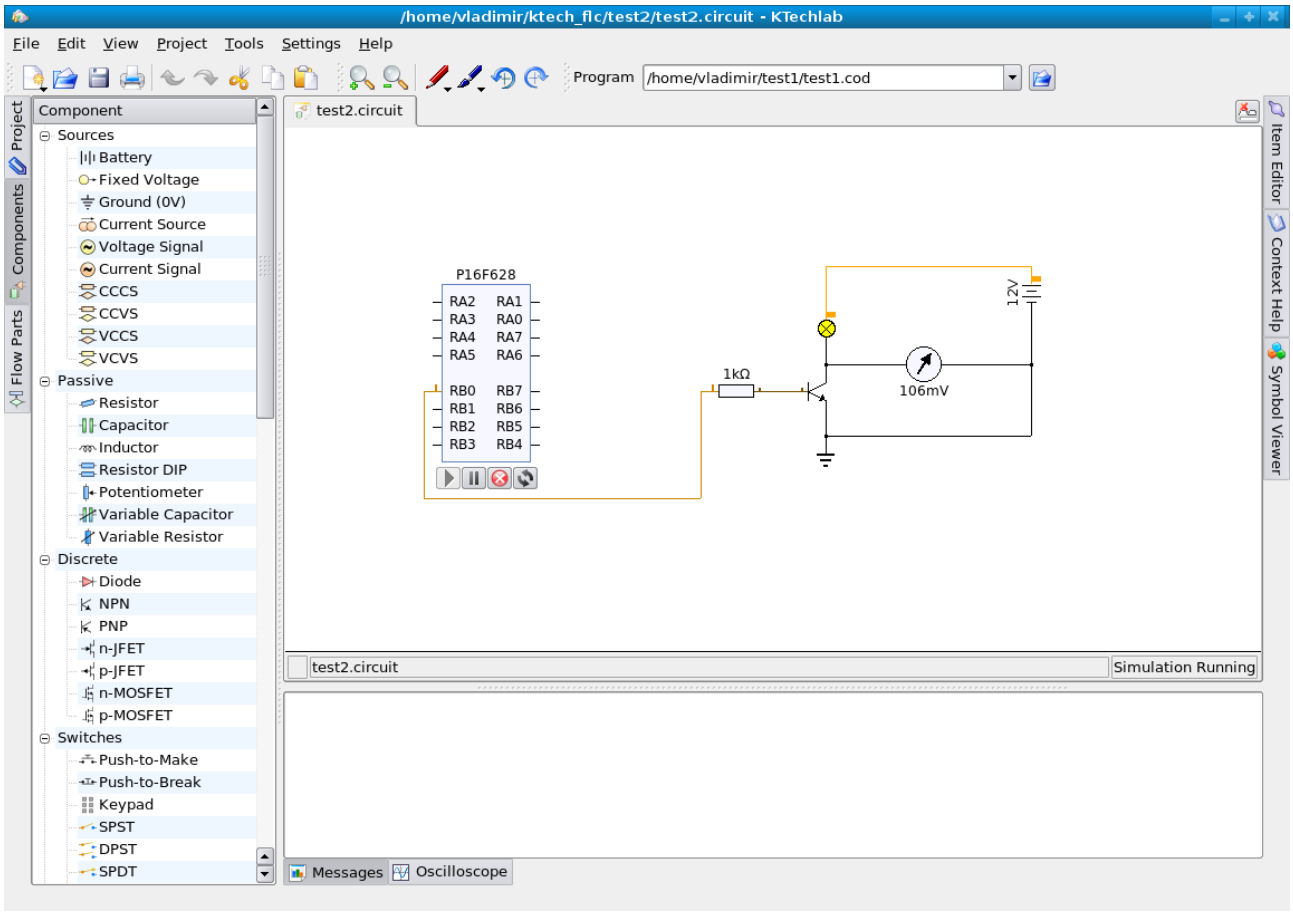


Рис. 1.1. Программа KTechlab

Я не нашел переключения интерфейса программы на русский язык, хотя очень большое количество программ в Linux давно русифицировано. Можно было бы это сделать, исходные коды программы есть, но я считаю полезным даже для самых начинающих радиолюбителей осваивать английский в объеме чтения документации, описаний, руководств и подсказок, как считаю полезным не всегда слишком детально расписывать все шаги — при желании этого можно добиться самостоятельно, и собственные находки запомнятся лучше, чем самое подробное описание в книге, но... попробую в этот раз останавливаться подробнее на каждом из шагов, которые планирую сделать.

Речь пойдет о работе программы с микроконтроллерами (конкретно я расскажу о PIC16F628A), поэтому после запуска программы обратимся к созданию нового файла: *File-New* в основном меню программы или первая кнопка слева с иконкой листа на инструментальной панели.

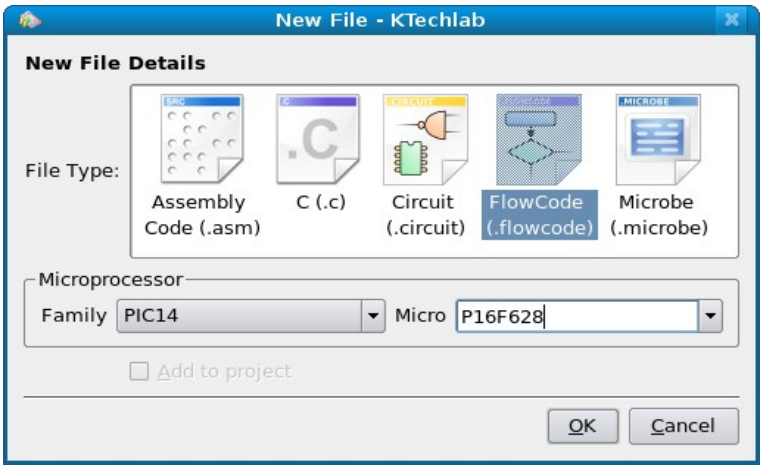


Рис. 1.2. Диалоговое окно создания нового файла в KTechlab

Задав устройство (окно Micro, кнопка со стрелкой ▼ справа от окошка) и выбрав FlowCode, с помощью кнопки OK мы окажемся в среде работы с выбранным ранее микроконтроллером.

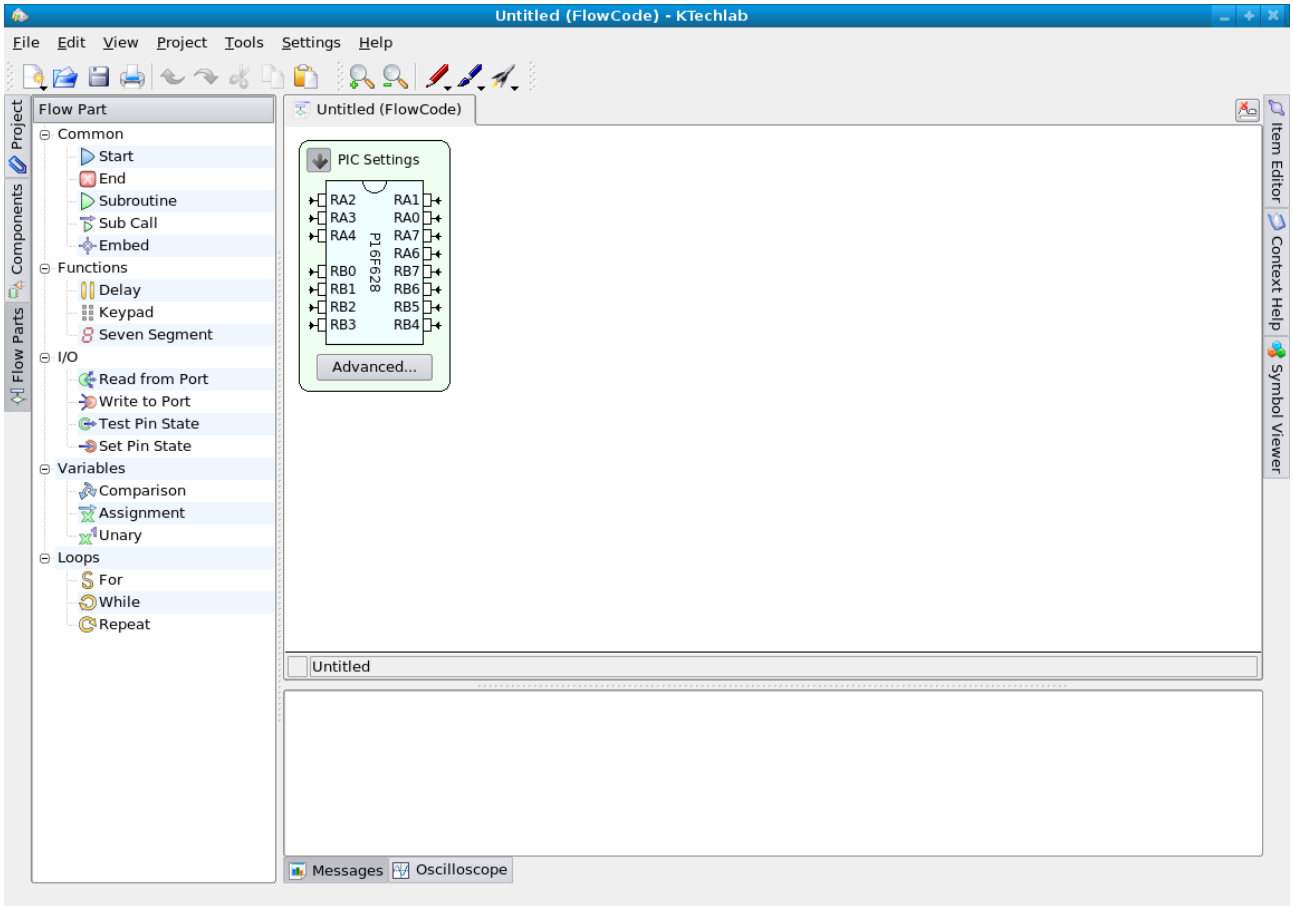


Рис. 1.3. Начало работы над программой для микроконтроллера

Как я говорил выше, одно из преимуществ программы KTechlab в том, что она позволяет работать не только с микроконтроллерами, но и со схемами обычных электронных устройств. Не в таком объеме, как Proteus, но для начинающих вполне достаточном. В отличие от нее программа FlowCode работает только с микроконтроллерами.

Создав новый файл, я намерен сохранить его, даже не начав с ним работать. Для этого я использую основное меню: *File-Save As...*

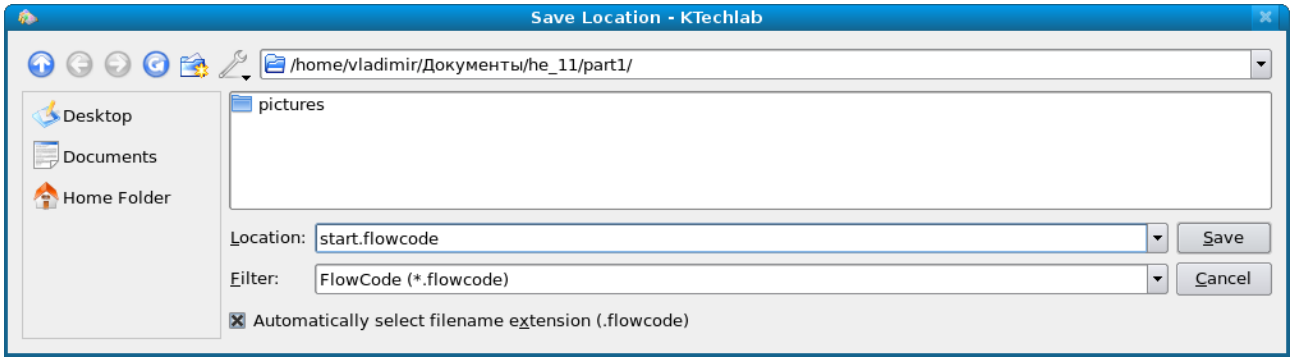


Рис. 1.4. Диалоговое окно сохранения файла

Теперь, возможно это и не обязательно, я хочу создать новый проект: *Project-New Project...*

В похожем окне диалога задаю то же имя проекта, что и имя созданного файла, затем добавляю ранее созданный файл в проект: *Project-Add Current File...*

Если теперь переключить вкладку слева (там три вкладки — FlowParts, Components и Project) на Project, то в составе проекта обнаружится ранее созданный файл. Но в данный момент нас больше должна интересовать вкладка FlowParts, куда мы и возвращаемся.

Многие языки программирования используют операторные скобки. В Pascal это пара Begin-End, здесь, насколько я понимаю, пара Start-End. Нажимая и удерживая левую клавишу мышки, я «цепляю» элемент Start и «тащу» его в рабочее поле программы. Аналогично поступаю и с элементом End. Между этими двумя элементами будет располагаться программа. Какая программа?

Если судить по моей электронной переписке, по разделам любительских форумов (да и профессиональных), круг интересов и область применения контроллеров весьма велики. Читать же описание создания сложной программы, особенно, если лично вам она не интересна, весьма скучное занятие. Поэтому, программа будет простой. Сделаем мультивибратор, частота которого будет управляться с помощью двух кнопок: больше, меньше. На первом этапе начальная частота будет 0.5 Гц. Такой выбор обусловлен тем, что результат удобно наблюдать при отладке программы, удобно наблюдать при макетировании без использования приборов — достаточно поставить на вывод контроллера светодиод (возможно, добавив «на всякий случай» резистор для ограничения тока). И, наконец, в устройство добавим семисегментный индикатор для отображения частоты.

Но даже такую простую программу я, такой я недалекий, разобью на несколько этапов. Первый этап работы над программой выглядит как ответы на некоторые вопросы:

- Какого вида сигнал должен получиться на выходе контроллера?
- Как этот сигнал можно реализовать в программе?

Сигнал на выходе мультивибратора, если к нему нет дополнительных требований, это меандр, то есть, периодически повторяющаяся равновременная последовательность высокого и низкого уровня. Таким образом, мне нужно уметь устанавливать на выходах контроллера высокий и низкий уровень. Для удержания этих уровней в программе можно добавить паузы. Если паузы будут секундные, то частота должна получится желаемой. И, кроме того, чередование этих двух уровней должно повторяться бесконечно.

Второй этап. К уже выбранным двум элементам я добавлю элемент *Set Pin State*. Напомню, что слева три вкладки, на одной из которых, Flow Parts, я и нахожу искомое. Цепляем программный элемент и перетаскиваем его мышкой в рабочее поле редактора.

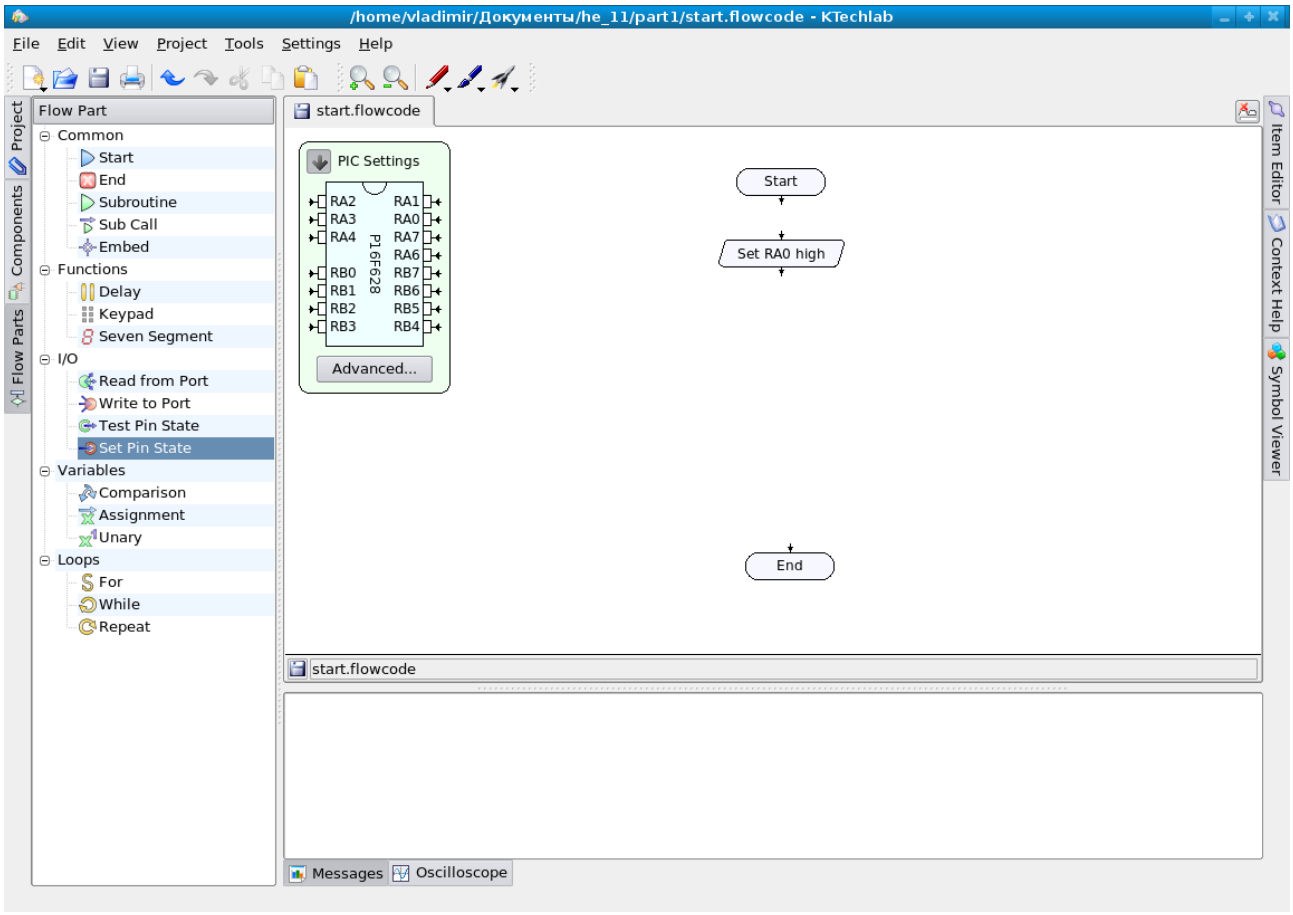


Рис. 1.5. Начало второго этапа работы над программой

Вывод, который используется по умолчанию — RA0. Я сейчас не собираюсь его менять, это 15й вывод микросхемы. Если щелкнуть мышкой по вновь установленному элементу, то на верхней панели появляется возможность изменить вывод и его состояние (стрелка ▼ справа).

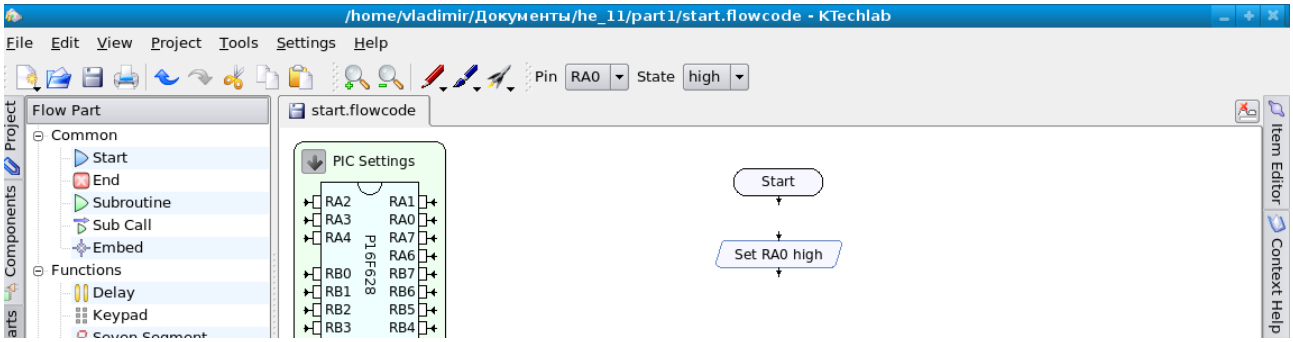


Рис. 1.6. Средство управления выводами

Pin — вывод, State — состояние. Сейчас нас устроит высокое состояние вывода. Добавим паузу (*Delay)* из набора программных элементов, оставив ее длительность «по умолчанию» равной секунде, и добавим еще раз пару элементов *Set Pin State* и *Delay*, но в *Pin State*

изменим состояние вывода с high на low.

Мы получили такой набор «полуфабрикатов»:

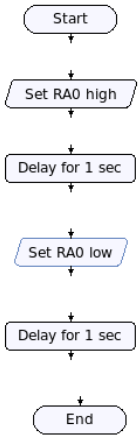


Рис. 1.7. Набор элементов для первого этапа работы

Для завершения этого этапа, напомню, нам нужно получить бесконечное повторение этой комбинации, добавим цикл: элемент *Loops-While*, внутрь которого перенесем все элементы, кроме операторных скобок *Start-End*. Мне приходится немного «удлинять» окно цикла, чтобы поместить все без тесноты, что можно сделать, щелкнув по окну While, зацепив его нижний угол и «протащив» окно до нужного размера.

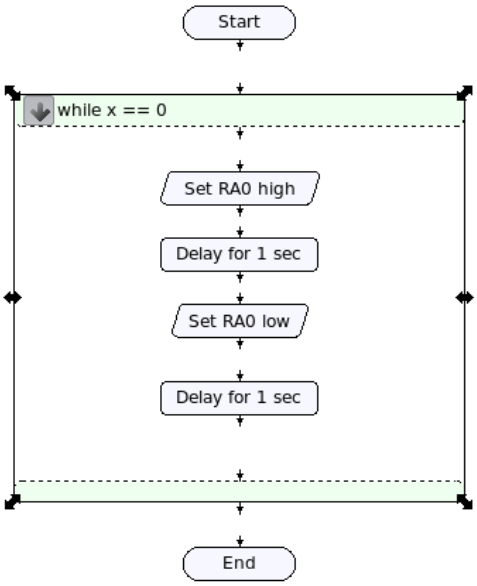


Рис. 1.8. Меандр в программе

Теперь можно соединить все элементы программы. Подводим маркер мышки к стрелочкам (они есть сверху и снизу у каждого элемента) элементов, нажимаем левую клавишу мышки и тянем линию к следующей стрелочке. Эту процедуру нужно выполнить обязательно. Даже тогда, когда при размещении стрелочки эти совпадают, они еще не соединены. Программа приобретает вид, я бы сказал, алгоритма.

Конечно, я не узнаю вид сигнала — прямоугольные импульсы — в этом программном фрагменте, но мне достаточно понятно, что я хочу получить в результате выполнения программы.

Я не знаю, как классифицируют специалисты такой способ создания программы. Для себя (для внутреннего пользования) я с удовольствием назову это графическим языком программирования.

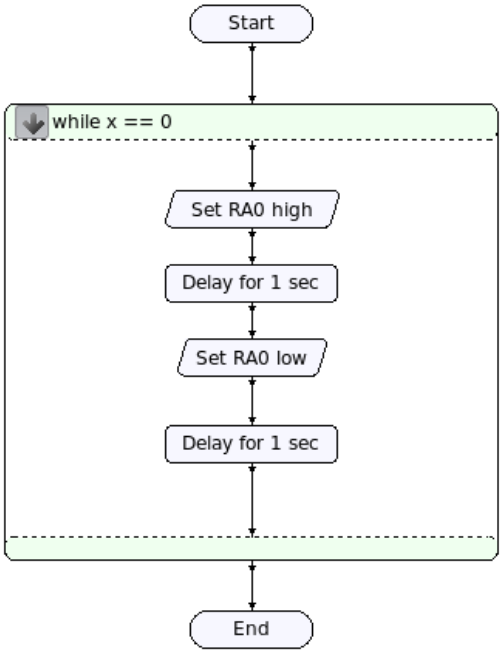


Рис. 1.9. Соединение элементов программы

Третий этап. Проверка и отладка.

Прежде, чем двинуться дальше, а я не знаю, в полной ли мере программа понимает мои намерения, я хочу сказать ей, что вывод RA0 я хочу использовать для выхода, а не для входа.

Выводы портов микросхемы контроллера могут использоваться для разных целей: цифрового ввода и вывода сигналов, аналогового ввода, входа и выхода сетевого приемо-передатчика, подключения внешнего кварца и т.д. Заглянув в справочный листок микроконтроллера, можно увидеть, что его выводы имеют несколько обозначений. Несомненно, должен существовать механизм определения назначения выводов. Как правило, это назначение определяется при программировании контроллера, например, при задании слова конфигурации или при выполнении определенных команд в программе.

То, что мне нужно, можно сделать, если щелкнуть по клавише с надписью **Advanced** на картинке с изображением микросхемы в верхнем левом углу рабочего поля программы. В появившемся диалоговом окне для PORTA в окошке Type (TRIS register):, где по умолчанию установлены биты 01111111, я последний бит изменю на 0.

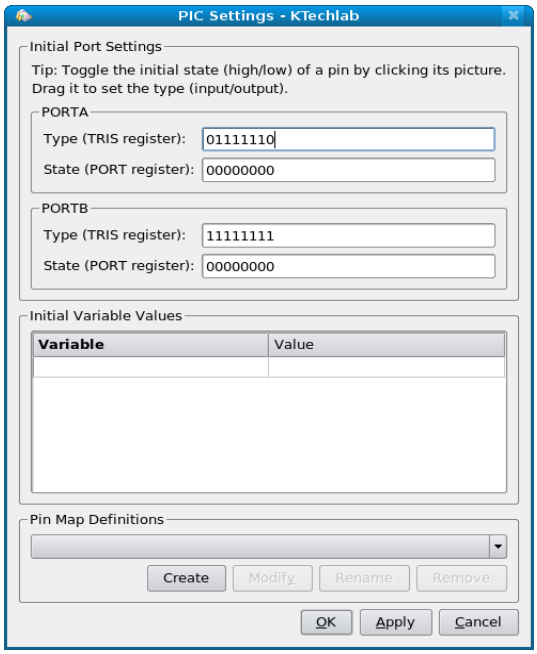


Рис. 1.10. Диалоговое окно назначения выводов порта

Закрыв окно нажатием на клавиши **Apply** и **ОК**, я могу проверить, если присмотрюсь к изображению микросхемы, что стрелочка возле RA0 сменила направление.

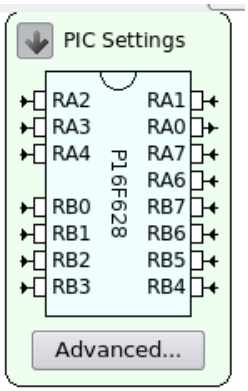


Рис. 1.11. Изменение направления ввода на рисунке микросхемы

Теперь она направлена вне микросхемы. Чтобы проверить программу, скажем, в отладчике, а затем на макетной плате, ее следует оттранслировать в загрузочный hex-файл. KTechlab для ряда программ имеет две возможности для проверки и отладки. Но прежде я хочу показать одну ошибку, которую можно совершить, работая со многими программами.

Эта ошибка в виде соответствующего сообщения появляется в данном случае при попытке трансляции программы, в других программах она может появиться при запуске отладчика. Но само сообщение может не указывать на причину, по которой программа не может дальше работать правильно, сообщение только указывает на наличие некоторых проблем. Так, как же выглядит сообщение об ошибке при трансляции программы в KTechlab?

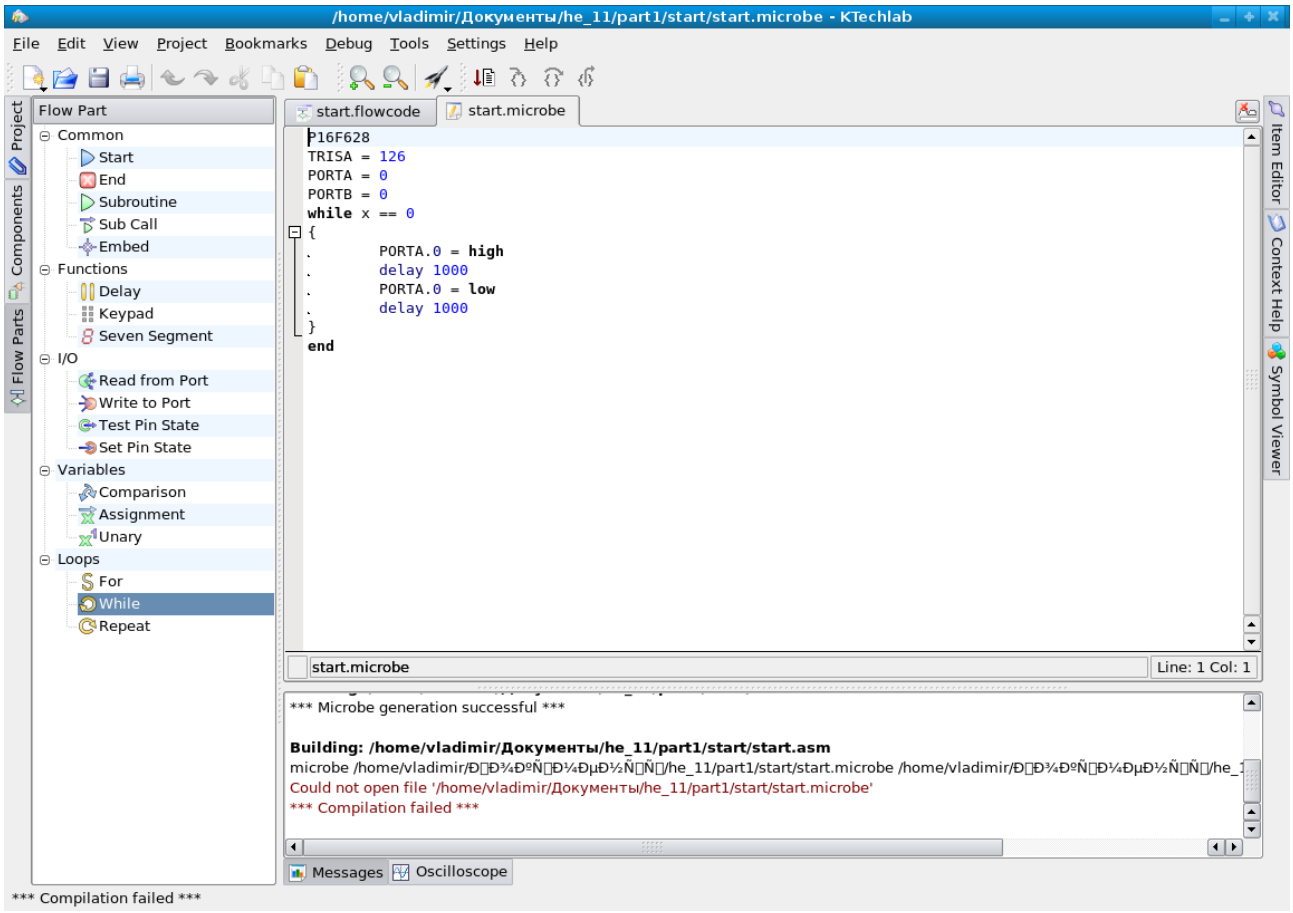


Рис. 1.12. Ошибка расположения рабочей папки

Сообщению: «Could not open file '/home/vladimir/Документы/he_11/part1/start/start.microbe (невозможно открыть файл и т.д.)», — я обязан тем, что расположил рабочую папку проекта в своей папке «Документы». Это штатная папка операционной системы, созданная без моего участия. В предыдущих версиях папка называлась «Documents», сейчас она стала называться по-русски, но программа не понимает кириллицы. И не может открыть файл.

Такое поведение характерно для многих программ, и не верьте им, когда они предлагают сохранить ваш проект в папке «Мои документы». Обычно, я сразу создаю папку в коневой директории и называю папку «work», названия проектов и файлов записываю латиницей. Некоторые программы имеют ограничения на длину пути к файлам проекта. Конечно, не все программы такие. Но иногда очень трудно, в отличие от приведенного примера, понять в чем проблема — лучше не рисковать. Если вы помните, было время, когда любое имя файла было ограничено несколькими символами, в ранних компиляторах было ограничение на длину имени переменных. Все эти детали выясняются при чтении документации, но иногда они ускользают от внимания, и, если в программе при трансляции возникают проблемы, то следует либо обратиться к описанию компилятора, либо не использовать длинных путей и длинных имен переменных.

Еще одна проблема возникла у меня после перемещения проекта в новую папку. Трансляция в microbe прошла, а дальше... отчего-то появились жалобы на отсутствие декларации переменной «x» в коде. Мне не захотелось заново начинать создание проекта, поэтому я просто добавил эту переменную в программу до цикла, где эта переменная используется.

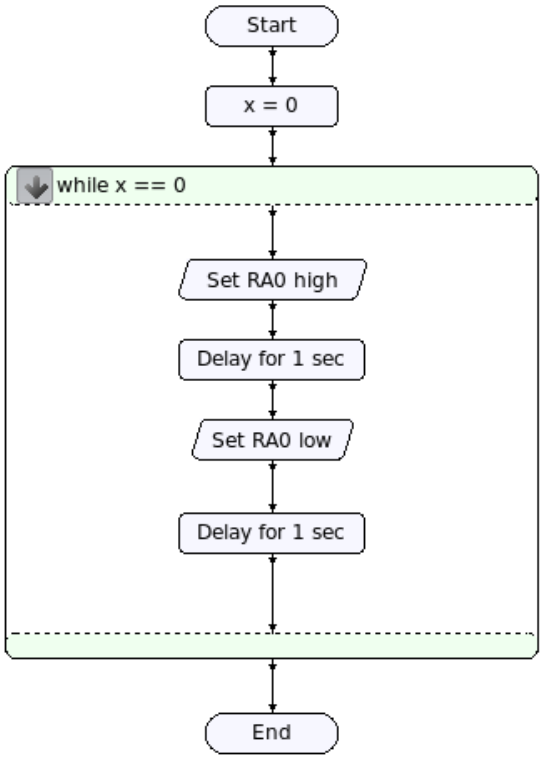


Рис. 1.13. Задание переменной x

Теперь можно вновь транслировать программу в microbe, ассемблер и hex-файл.

Трансляция программы запускается с помощью кнопки на основной инструментальной панели с иконкой ракеты. Появляющееся выпадающее меню позволяет выбрать результат трансляции.

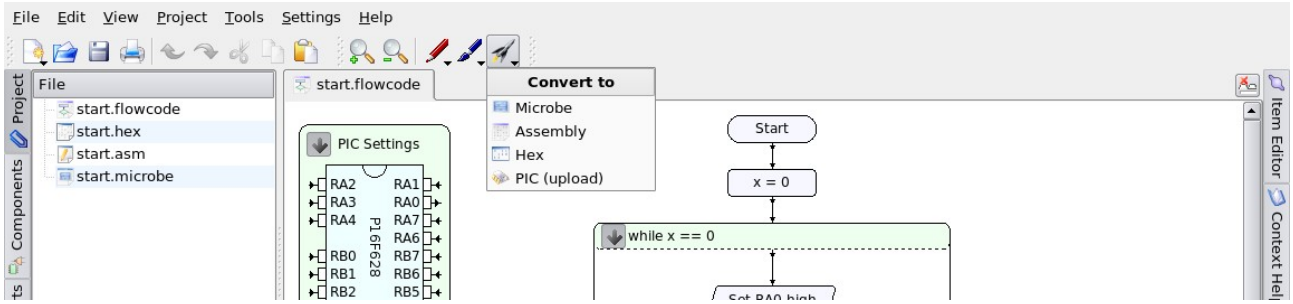


Рис. 1.14. Меню выбора трансляции программы

Я не люблю приводить назначение всех панелей и меню программы, для этого есть руководства. Но для многих начинающих работу с программой чтение документации на английском затруднительно. Поэтому об основных панелях интерфейса и диалогах программы я постараюсь ниже рассказать, а сейчас... Можно попробовать загрузить hex-файл в микросхему, перенести ее на макетную плату и проверить. Или, нет, пожалуй...

Начиная эту главу, я привел рисунок, где микроконтроллер используется в качестве элемента электрической схемы. Подобный подход можно реализовать сейчас для проверки работы программы.

Можно создать новый проект или использовать существующий, создав новый файл, но

при выборе типа файла, рисунок 1.2, указать Circuit — схема. На вкладке Components есть в разделе Integrated Circuits (интегрированные схемы) элемент PIC, который мы и перетащим в поле редактора схемы.

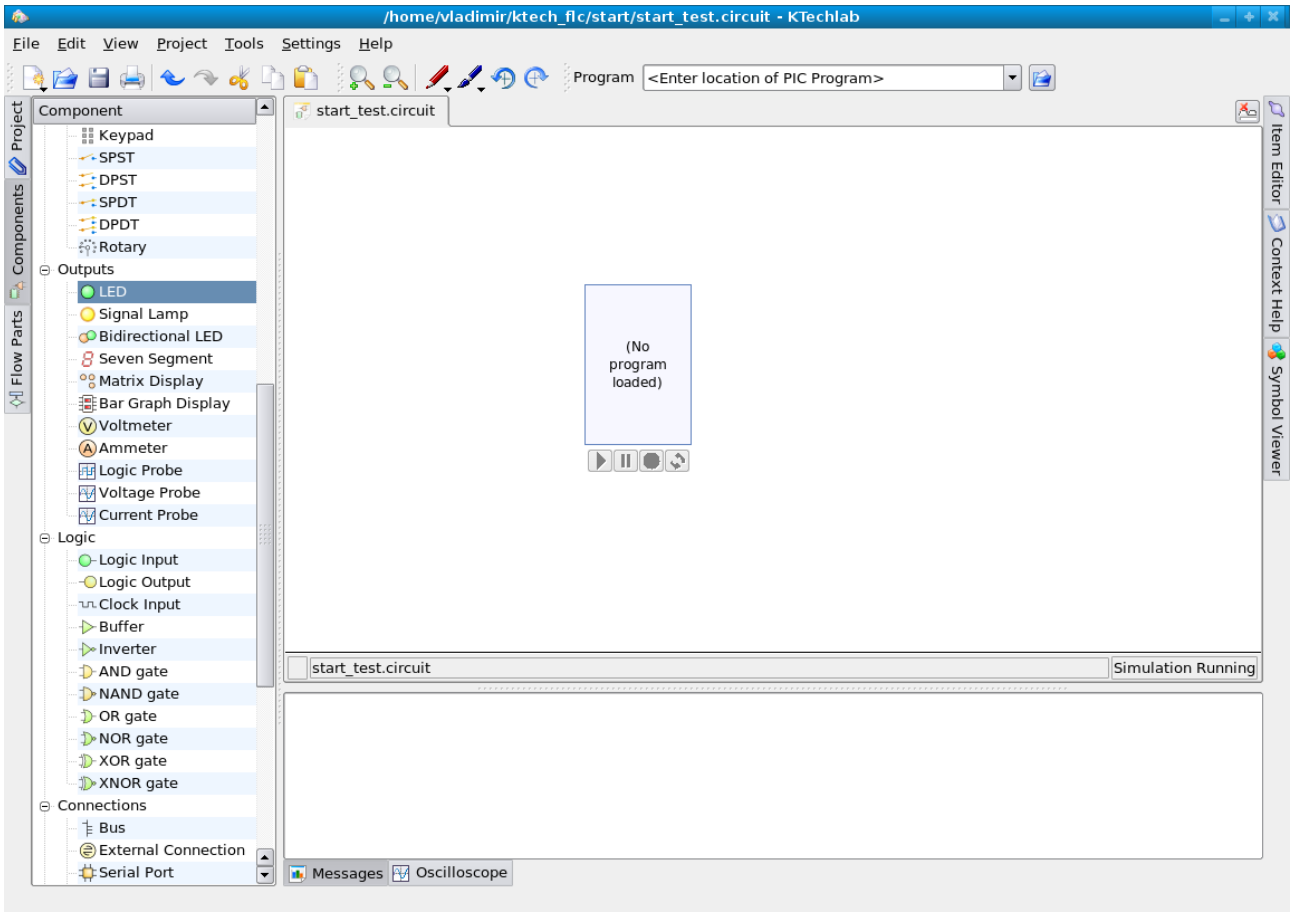


Рис. 1.15. Создание схемы для проверки программы

Двойной щелчок по этому компоненту создает на инструментальной панели окно с названием Program. Сейчас в окне написано: «<Enter location of PIC Program>» (введите место расположения PIC программы). Кнопка с иконкой папки правее открывает диалоговое окно выбора программы.

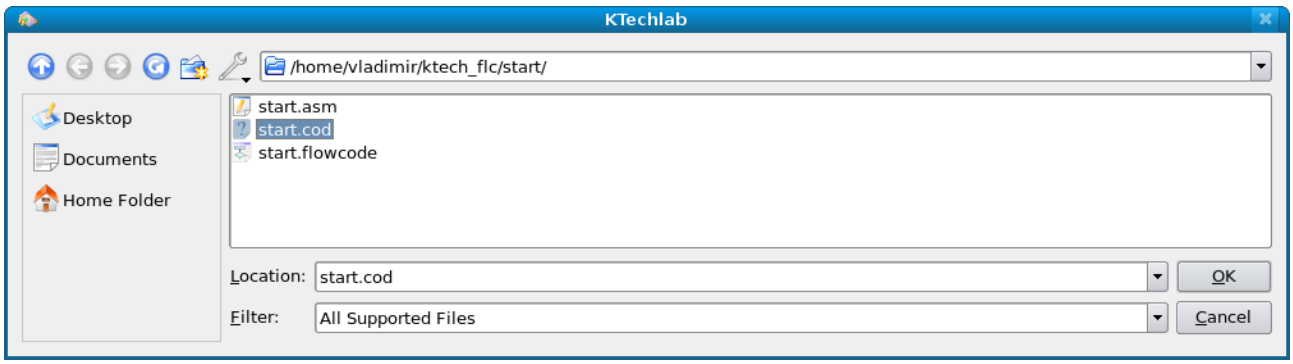


Рис. 1.16. Диалоговое окно выбора программы для контроллера

Как видно из рисунка, я выбираю созданный при трансляции программы отладочный файл с расширением .cod. Подтвердив выбор клавишей **ОК**, люблю эту клавишу, я получаю

обычный схемный рисунок контроллера, модель которого появляется над графическим изображением.

В окне компонентов, в длинном списке в рубрике Outputs, есть светодиод — LED. Его я и использую для проверки, присоединив к выводу RA0.

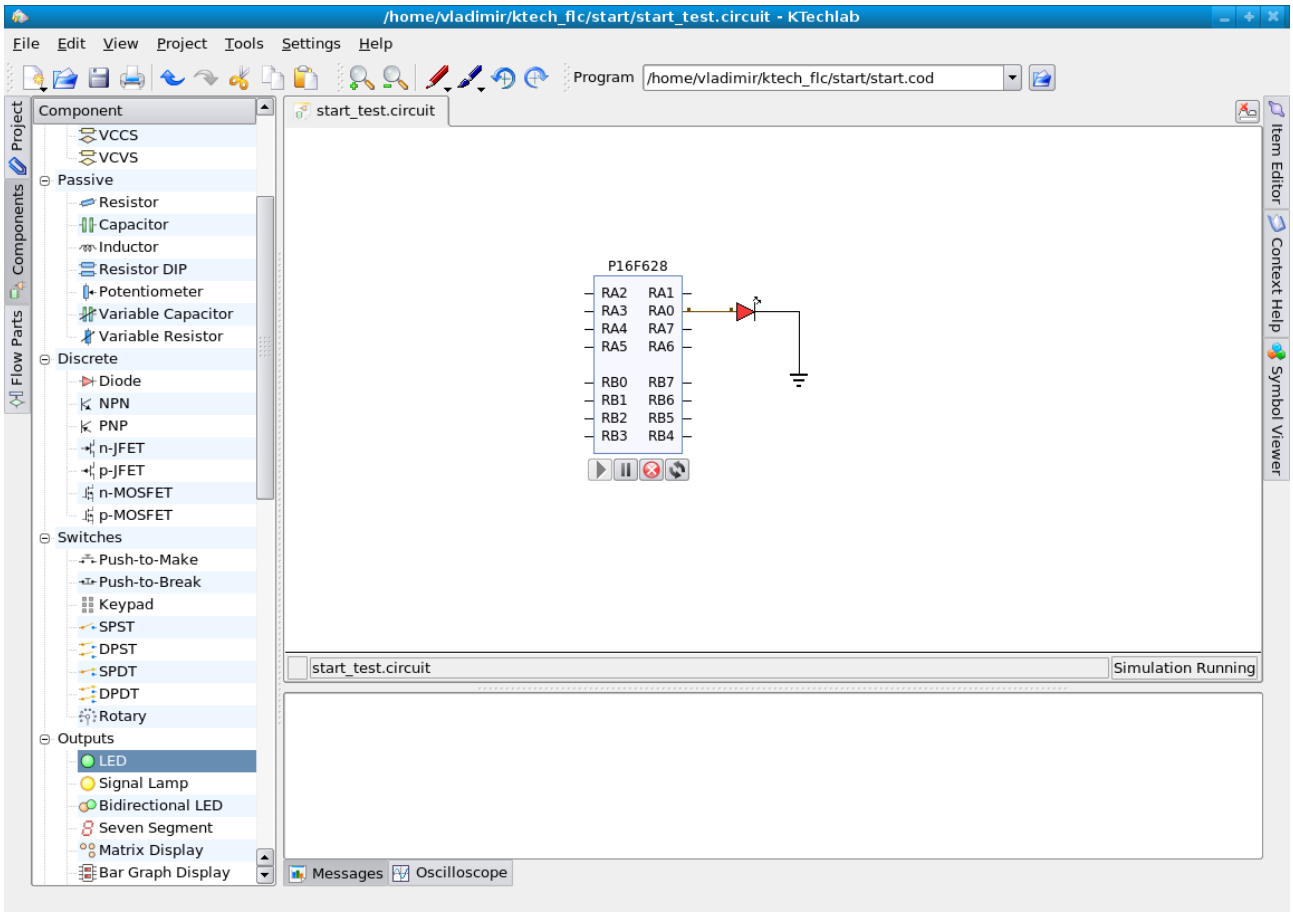


Рис. 1.17. Окончательная схема проверки программы

Для запуска симуляции достаточно нажать клавишу ► на инструментальной панельке управления под графическим изображением микроконтроллера.

KTechlab позволяет использовать, например, два микроконтроллера, в которые можно загрузить разные программы. Есть ли какие-то ограничения на сложность и размер программ? Ответ на этот вопрос следует искать в руководстве к KTechlab.

Думаю, удобно использовать схему для проверки, положим, работы микроконтроллера с семисегментным индикатором и кнопками управления, для работы с внешним АЦП или аналоговыми компонентами. Попробуйте!

Микроконтроллер в программе FlowCode (Windows)

Программа FlowCode существует в двух версиях — для контроллеров AVR и PIC. Удобно, что можно импортировать решения из одной версии в другую. Для начинающих удобна простота отладки, поскольку есть много внешних элементов устройств, обычно используемых вместе с микроконтроллерами.

Чтобы не возвращаться к вопросу о назначении элементов панелей и не ссылаться на то, что я ранее писал об этом, я приведу описания панелей.

Первая левая инструментальная панель — это панель команд.

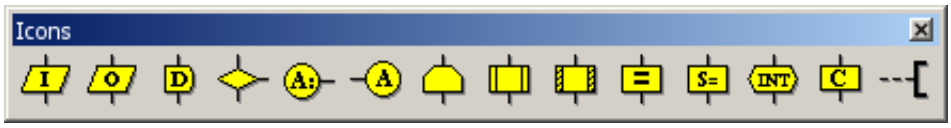


Рис. 2.1. Инструментальная панель команд

Перечень представленных команд (слева-направо на рисунке, сверху-вниз при запуске):

Input (ввод), *Output* (вывод), *Delay* (пауза), *Decision* (ветвление), *Connection Point* (две точки соединения), *Loop* (цикл), *Macro* (макрос), *Component Macro* (макрос компонента, добавленного в программу), *Calculation* (вычисление), *String Manipulation* (строковые операции), *Interrupt* (прерывание), *C Code* (блок кода на языке Си), *Comment* (комментарий).

Вторая инструментальная панель для добавочных внешних элементов.

Рис. 2.2. Инструментальная панель добавочных элементов

Компоненты (слева-направо):

LEDs (светодиоды), *Switches* (переключатели), *LCDDisplay* (жидкокристаллический дисплей), *ADC* (АЦП, если есть порт АЦП), *LED7Seg1* (семисегментный индикатор), LED7Seg4 (блок из 4х семисегментных индикаторов), *Buggy* (компонент игрушки), далее несколько стандартных интерфейсов *TCP_IP, Bluetooth, RS232, IrDA, AddDefines* (добавить определения), *LinMaster* (ведущий в локальной сети), *Custom* (заказной компонент), *EEPROM* (перепрограммируемая память), *Alarm* (охранное устройство), *Thermometer* (термометр), *PWM* (широтно-импульсный модулятор), *SPI* (последовательный внешний интерфейс), *WebServer* (web-сервер), *CAN* (сеть абонентского доступа), *KeyPad* (клавиатура), *LinSlave* (ведомый в локальной сети), *FormulaFlowCode* (компонент игры), *I2C* (шина связи между ИС).

Работа с элементами программы в FlowCode схожа с аналогичной в KTechlab. Если кого-то интересует более подробное описание (все-таки, чтобы не повторять полностью), то можно почитать help или те истории, относящиеся к программе, что есть на моем сайте.

Шарик пинг-понга перелетел на половину программы FlowCode, когда программа для микроконтроллера должна была генерировать меандр с частотой 0.5 Гц.

Такая программа в среде разработки FlowCode должна иметь следующий вид:

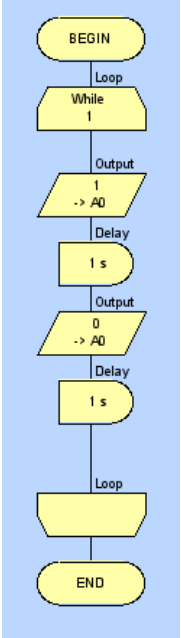


Рис. 2.3. Вид предыдущей программы микроконтроллера в FlowCode

Программа полностью построена из элементов Output (выход), Delay (пауза) и Loop (цикл). Цикл, в отличие от KTechlab, существует в единственном виде, но можно изменить его свойства, превратив в цикл for. Так же в свойствах Output задано управление единственным выводом RA0 (A0 на рисунке). Во всем остальном программы, практически, одинаковы. Но в отношении отладки с FlowCode проще. Достаточно добавить светодиоды с панели добавочных элементов, чтобы можно было запустить проверку программы.

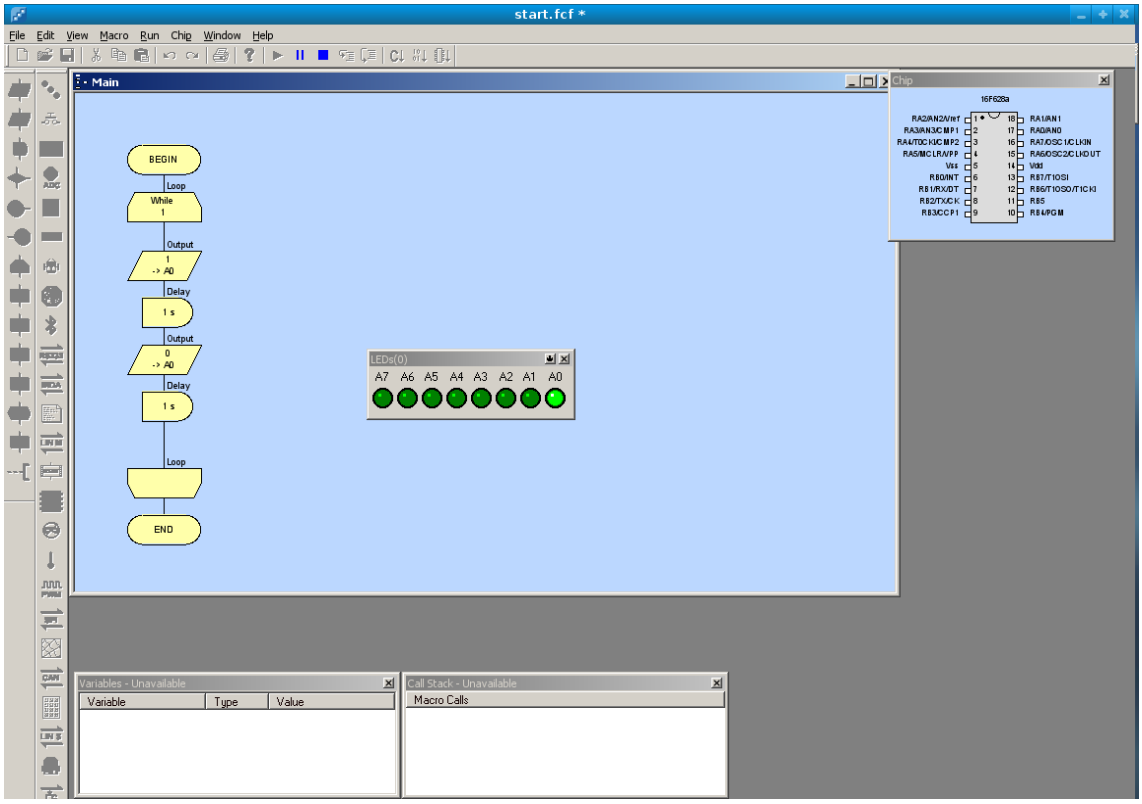


Рис. 2.4. Первая проверка начального этапа программы

Очень, конечно, хочется увидеть это все в «живом» виде. У меня даже есть макетная плата

с панелькой для микросхемы, а к некоторым выводам подключены светодиоды...

Хочется-перехочется.

Следующий этап работы выглядит в данный момент так:

- Ввести две кнопки управления частотой.
- Попробовать действие кнопок при удвоении частоты.

Ввести две кнопки для управления, в отношении программы, означает добавить два программных элемента ввода (Input). Труда это не составит, взял и перетащил с инструментальной панели элементов программы. Так, вперед!

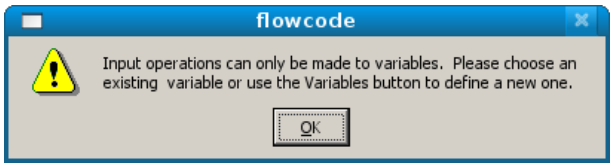


Рис. 2.5. Начало следующего этапа создания программы

Что такое я сделал, обидев программу FlowCode?

Сознаюсь, мне пришло в голову, что (я все-таки «железячник») вход на схемах обычно слева, а выход справа. К микросхеме контроллера это не имеет отношения, но привычнее будет, если входы будут привязаны к порту A, а выходы к порту B. Я переобозначил два элемента программы Output: без проблем, вызываем двойным щелчком по элементу программы диалоговое окно свойств, в котором меняем порт A на B.

Затем перетащил два программных элемента ввода, расположив их перед началом цикла (Loop), и попробовал задать свойства первого Input (тот же двойной щелчок), привязав его к единственному биту (выводу микросхемы) A0, второй будет A1. Чем, похоже, и обидел программу, которая сообщила, что операции ввода должны быть сделаны для переменных — по умолчанию стоит значение «0». Программа предлагает выбрать существующую переменную или создать новую, используя клавишу **Variables**.

Нажатие указанной клавиши открывает диалоговое окно создания переменных, в котором есть клавиша **Add New Variable...**, открывающая следующее окно, где собственно и создается новая переменная, для которой я приготовил имя «more» — больше. Тип переменной Byte меня вполне устроит.

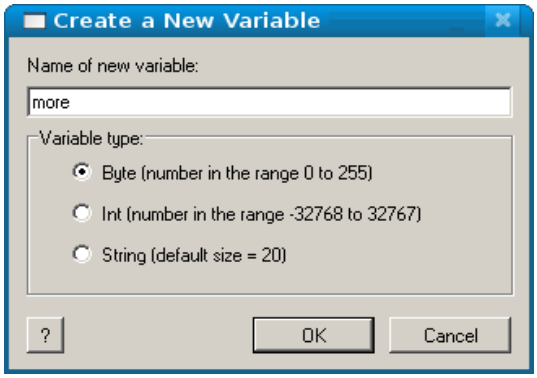


Рис. 2.6. Создание новой переменной в FlowCode

Как часто бывает в программах, это не единственный путь создания переменных. В основном меню *Edit-Variables...* можно открыть диалоговое окно переменных, где и создать,

точно так же, нужные переменные.

Клавиша **ОК**, и попадаем в предыдущий диалог.

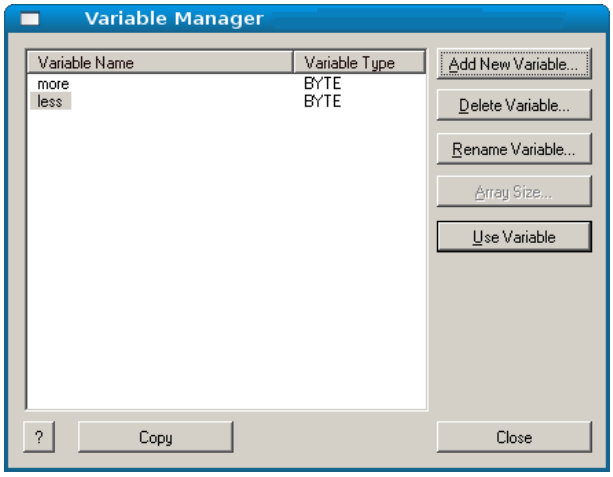


Рис. 2.7. Диалоговое окно работы с переменными

Клавиша **Use Variable**, и мы возвращаемся к началу путешествия, где, помирившись с программой, можем нажать **ОК**. Для второго элемента ввода, как видно на рисунке выше, я выбрал переменную less (меньше). И пора, думаю, показать, во что я превратил программу.

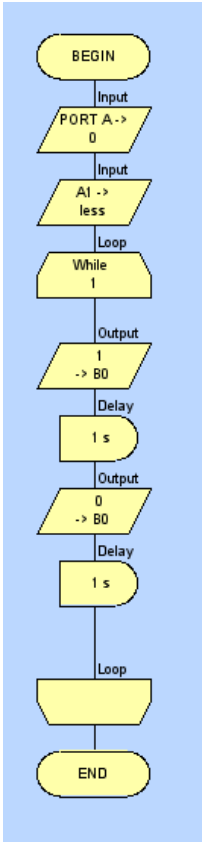


Рис. 2.8. Начало работы над программой на втором этапе

Почему я не добавил элементы ввода в цикл? А при перетаскивании этих элементов рука направилась именно внутрь цикла, поскольку опрашивать кнопки управления тоже нужно в бесконечном цикле, но пока я перетаскивал, у меня мелькнула мысль, что операции внутри

цикла, возможно, придется переписывать несколько раз для каждого измененного значения частоты. Так ли это, покажет время, но ввод я пока вынес за пределы цикла.

Чтобы не переписывать многократно одинаковые фрагменты программы, программисты давно придумали такие удобные средства, как подпрограмма, или процедура, а в данном случае программный элемент Macro.

Перетаскиваем его в программу ниже элементов Input, он вписывается в программу, как Call Macro (вызов макроса), а двойной щелчок по нему открывает диалог работы с макросами.

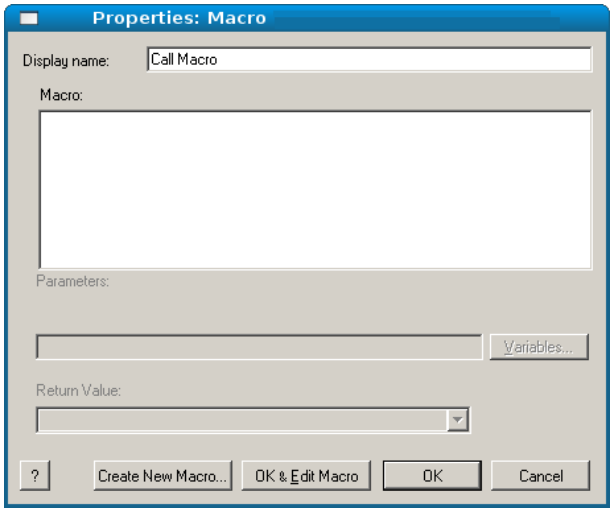


Рис. 2.9. Диалоговое окно работы с макросами

Клавиша **Create New Macro...** (создать новый макрос), как мне кажется, хорошее начало для создания нужной мне подпрограммы.

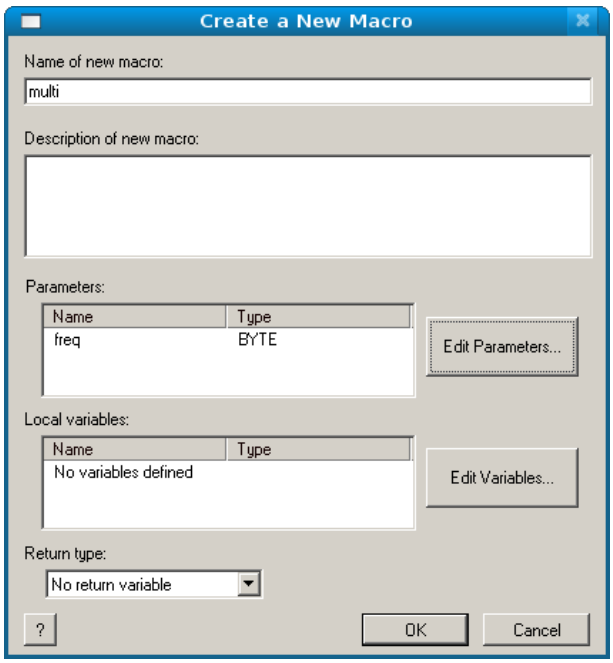


Рис. 2.10. Задание свойств подпрограммы

Я дал подпрограмме имя «multi» и, пока «на всякий случай», добавил параметр freq. Я

еще не знаю, смогу ли я воспользоваться этим параметром, как я это смогу сделать, но удалить лишнее я всегда смогу (если не забуду, как обычно бывает). Добавление параметра потребовало создания еще одной переменной freq_ch, но, все неприятности когда-нибудь заканчиваются. Я, наконец, получаю подпрограмму. Вернее, заготовку для нее.

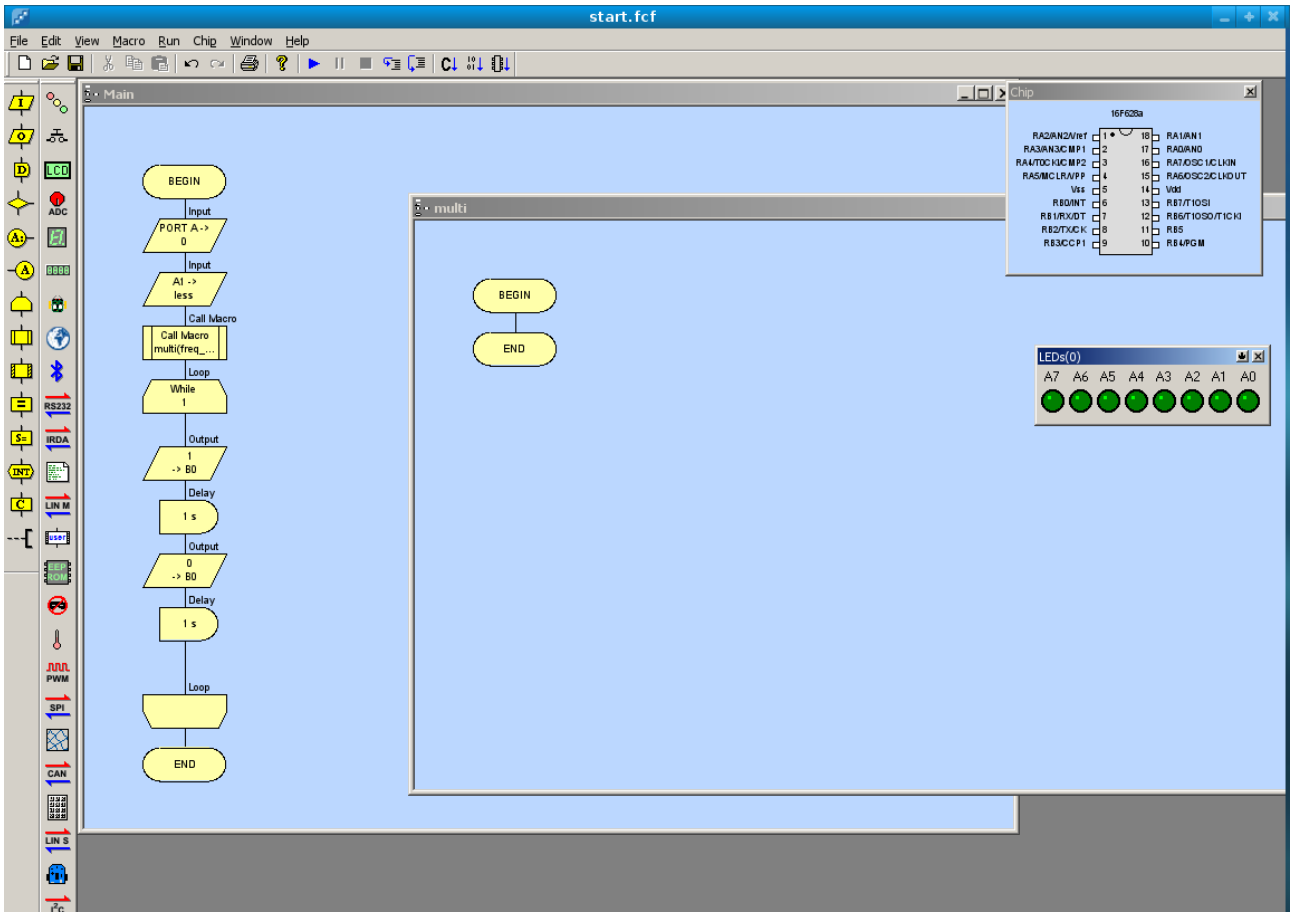


Рис. 2.11. Создание заготовки для подпрограммы

Теперь я перенесу весь цикл из основной программы в подпрограмму. Как во многих графических редакторах, для выделения достаточно, нажав левую клавишу мышки, обвести нужные элементы. Затем (перетащить не получилось) в основном меню *Edit-Cut* (вырезать), перемещаемся в рабочее поле подпрограммы и щелкаем мышкой по линии связи операторных скобок (появляется стрелочка), и в основном меню *Edit-Paste* (вставить).

Подпрограмма выглядит так, как выглядела программа в момент начала работы. Но мне этого мало. Я, я не забыл об этом, я хочу изменять частоту мультивибратора, а подпрограмма, в конечном счете, и есть мультивибратор. Изменить частоту я могу единственным образом, я, по крайней мере, пока не вижу другого способа, изменив длительность пауз. Поэтому в свойствах паузы (двойной щелчок по первому, затем второму элементу Delay подпрограммы) я меняю число «1» на переменную freq_ch, которую создал при создании подпрограммы. Посмотрим, что из этого получится. Пока я не превратил программу в «небоскреб», можно проверить, работает ли задумка?

Я добавлю в основную программу (из которой удален цикл) после двух элементов Input новый элемент, который называется Calculation (вычисление). В его свойствах, я надеюсь, я смогу задать нужное значение для начальной частоты мультивибратора. То есть, начальную длительность пауз.

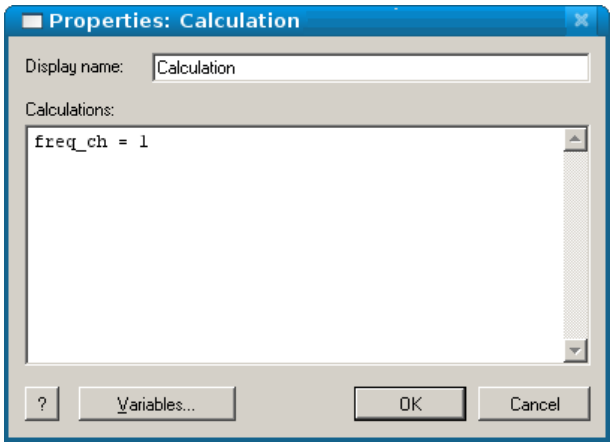


Рис. 2.12. Задание начальной частоты мультивибратора

Достаточно нажать на кнопку ► основной инструментальной панели программы, чтобы проверить работоспособность конструкции, если не забыть (что я и сделал) поменять в свойствах привязку светодиодов к порту B, а не A, как было раньше.

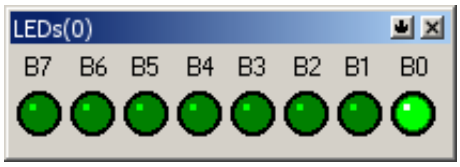


Рис. 2.13. Добавочный элемент светодиоды

Первая из кнопок основной панели LEDs — доступ к свойствам через выпадающее меню, где можно выбрать пункт *Component Connections*... и заменить привязку к порту.

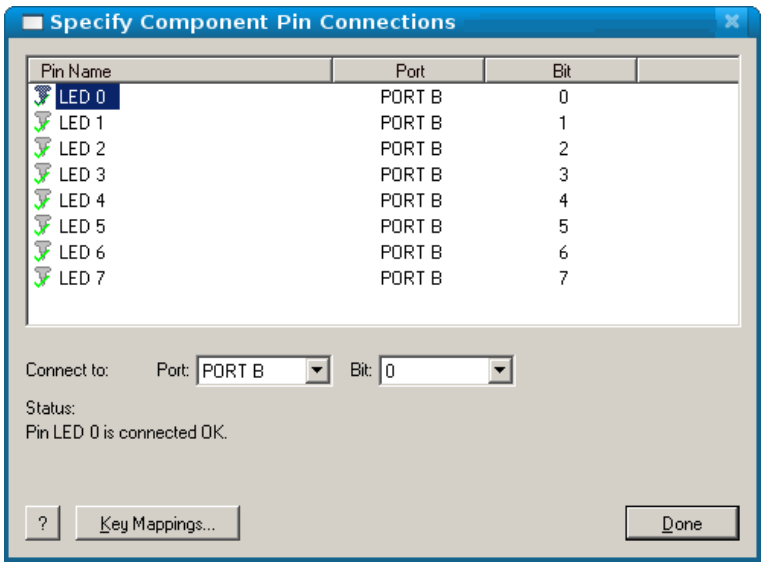


Рис. 2.14. Изменение порта «приписки» светодиодов

После сделанных изменений проверка программы запускается клавишей ►на основной инструментальной панели FlowCode.

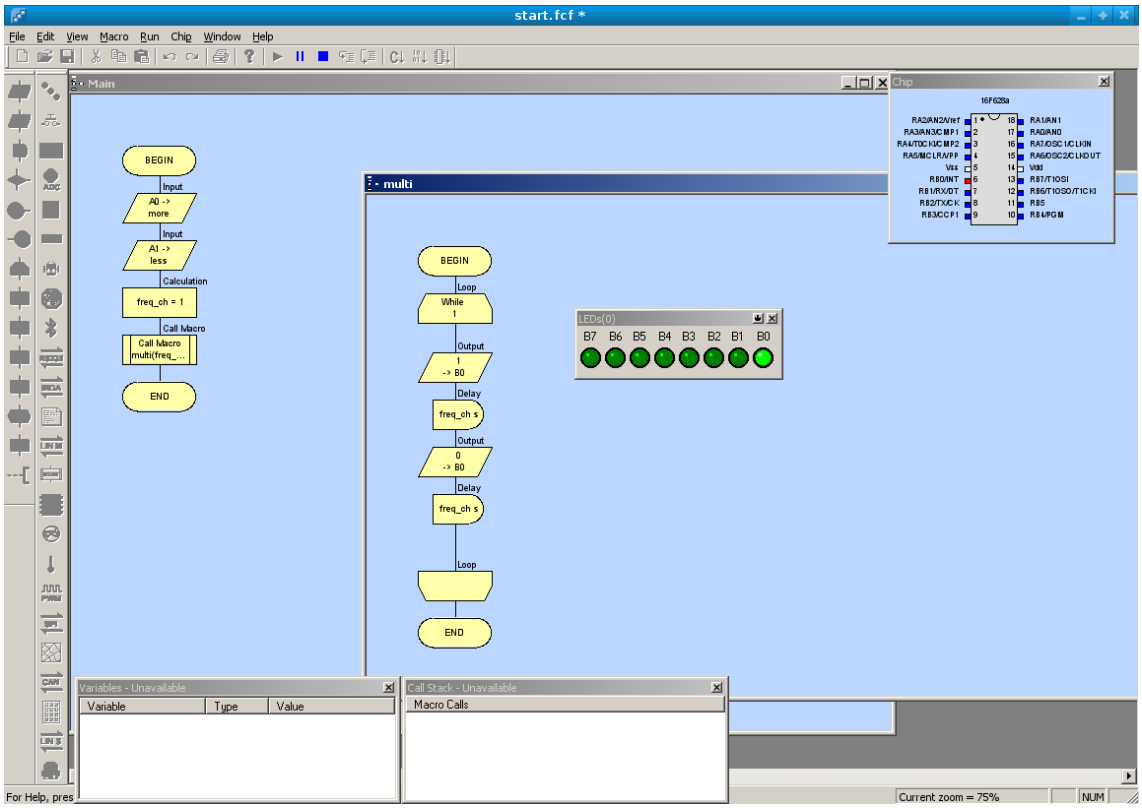


Рис. 2.15. Проверка работы подпрограммы

Я не вижу разницы в работе программы сравнительно с предыдущей проверкой. И еще, пока «болезнь не запущена», с программами это бывает, проведем еще одну диагностику.

Я намерен в дальнейшем менять значение частоты, меняя значение переменной freq_ch. При нажатии кнопки «больше» эта переменная должна делиться на 2. Проверим. Добавим после первого элемента Calculation второй такой же. А в нем запишем freq_ch = freq_ch/2. Запустим программу...

...чтобы убедиться, что-то идет не так, как должно. Программа сейчас выглядит (основная программа):

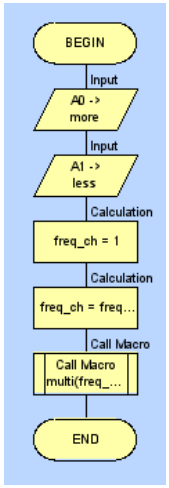


Рис. 2.16. Вид основной программы на данный момент

Светодиод вместо того, чтобы мигать быстрее, то ли мигает, то ли нет — непонятно. Но есть простой способ проверить, а я думаю, что проблема с переменной freq_ch, проверить, что с ней происходит. Для этого достаточно в подпрограмме включить точку останова на элементе, скажем, первого вывода в порт B. Для включения точки останова можно щелкнуть правой клавишей мышки по этому элементу и в выпадающем меню щелкнуть по пункту *Toggle Breakpoint*. Запуск программы выводит в окно наблюдения переменные и их значение.

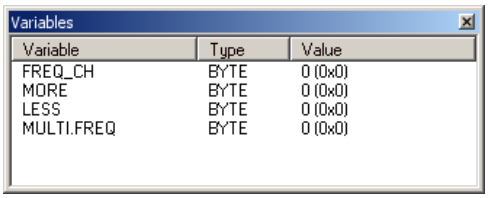


Рис. 2.17. Окно наблюдения при включении точки останова

Видно, что после деления переменная превращается в ноль. Первое, что приходит в голову, изменить тип переменной на int (целое). И задать значение не «1», а 1000. И поменять в подпрограмме в свойствах паузы секунды на миллисекунды.

Теперь после деления на два переменная становится равной 500, а светодиод исправно мигает. Своевременная проверка.

Итак. Мы создали подпрограмму работы мультивибратора. Мы добавили два элемента ввода (для двух кнопок управления). И, наконец, разобрались с механизмом изменения частоты будущего мультивибратора. Не пора ли перебросить шарик пинг-понга на другую половину стола?

Развитие программы в KTechlab (Linux)

Программа FlowCode позволила нам легко создать подпрограмму и осуществить (и проверить) механизм изменения частоты мультивибратора. Позволит ли программа KTechlab повторить эти достижения?

Попробуем.

Как выглядела наша программа, перед тем, как мы отправились на другую половину?

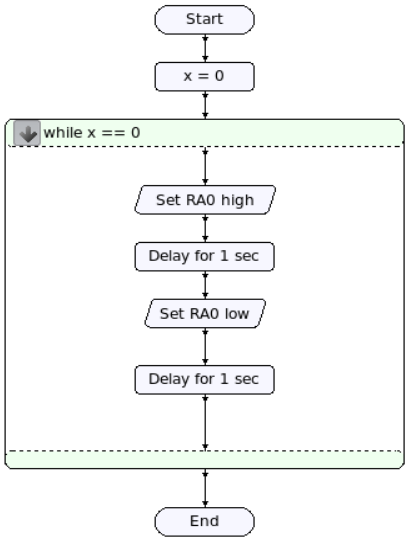


Рис. 3.1. Вид программы предыдущего этапа в KTechlab

«На всякий случай» я создаю новую папку под названием «start2», удаляю из проекта все файлы. Это легко сделать в менеджере проекта (слева, вкладка Project), щелкнув по файлу, включенному в проект, правой клавишей мышки и выбрав из выпадающего меню пункт *Remove from Project* (удалить из проекта):

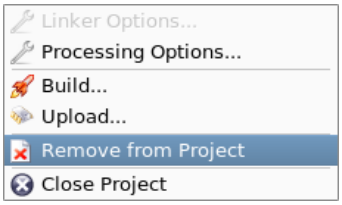


Рис. 3.2. Выпадающее меню управления проектом

После удаления файла, я закрываю его. И так до тех пор, пока не удалены все файлы, а в рабочем поле программы остается основной файл, который был назван ранее start.flowcode. Замечу, что при повторном запуске KTechlab, программа открывает последний проект, над которым велась работа. Так программа у меня настроена.

Теперь, удалив все лишнее, я создаю новый проект «start2», который размещаю в новой папке. Сохраняю открытый файл как start2.flowcode. Если теперь щелкнуть правой клавишей мышки в менеджере проекта на пустом месте, то в выпадающем меню можно выбрать *Add Current File...* (добавить текущий файл).

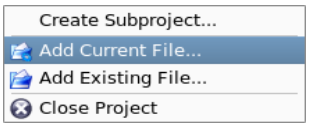


Рис. 3.3. Выпадающее меню добавления файла к проекту

Этим в проект будет добавлен исходный файл для следующего этапа работы. Пока мы только повторим все действия, которые проделали в программе FlowCode:

1. Создадим подпрограмму, куда перенесем работу мультивибратора.
2. Создадим два программных элемента ввода для двух кнопок.
3. Создадим переменные для организации изменения частоты.

Все необходимые программные элементы в KTechlab на вкладке Flow Parts левого окна.

Есть элемент Subroutine (подпрограмма), думаю, он-то нам и нужен. Перетащим его в рабочее окно программы. По умолчанию подпрограмма называется MySub. Если выделить элемент щелчком по нему (хотя бы по имени), то на основной панели программы появляется окно свойств, где можно изменить имя подпрограммы. В FlowCode мы назвали подпрограмму «multi». Сохраним это имя. Кстати, я удалил все связи вне цикла, что можно сделать проведя курсором мышки (при нажатой левой клавише) по связи и нажав клавишу **Delete** на клавиатуре. Теперь свободный «от пут» цикл можно перенести в подпрограмму. Но, вспоминая, что мы заменяли значения пауз именами переменных, прежде, чем переносить, я хочу проделать замену. Но одного желания, как выясняется, мало. Я не могу поступить с паузой в программе KTechlab, как поступил с ней в FlowCode. Придется что-то придумать. Например, я могу добавить цикл, который ничего не будет делать в течение нужного мне (но пока не известного) количества проходов по циклу. Этот прием позволит мне добиться желаемого, хотя работа цикла будет сильно зависеть от тактовой частоты контроллера, то есть, ее придется либо рассчитывать, либо подбирать. Однако я пока не вижу альтернативы.

Заменим элементы Delay (пауза) циклами While (пока). Конструкция программного блока «мультивибратора» получается такая.

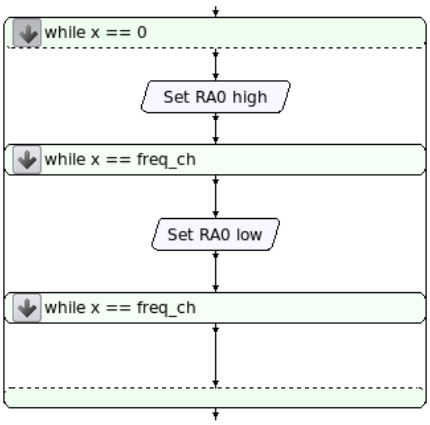


Рис. 3.4. Конструкция «мультивибратора»

В циклах While условие окончание цикла «по умолчанию» это число. Но его, похоже, можно заменить переменной. Осталось проверить, будет ли эта переменная локальной, или нет. Все три цикла While работают с переменной x, которая должна быть локальной. А

freq_ch? Я задал ее значение в основной программе равное 1000. Кстати, тоже следует проверить — попытка в цикле for задать такое количество проходов не увенчалась успехом. То есть, задать можно, но при трансляции на ассемблер возникает ошибка: «Слишком большое число». Количество проходов в цикле for, скорее всего, имеет тип Byte (байт).

Первой проверкой послужит трансляция программы в hex-файл...

И 1000 тоже слишком много. Смиримся, пока пусть будет 255. Теперь трансляция проходит полностью. При трансляции программа спрашивает имена сохраняемых файлов, открывая диалог сохранения. Если при сохранении ассемблерного файла не добавить расширение .asm, то программа делает вид, что что-то делает, но не делает, похоже, ничего. Да и я бы не стал, поскольку ассемблерный файл может храниться с несколькими расширениями.

Для отладки программы, созданной в KTechlab, удобно использовать программу gpsim. И, если я не ошибаюсь, в некоторых дистрибутивах Linux этот отладчик лучше запускать из терминала. Программа KTechlab при трансляции создает файл с расширением .cod — его-то и нужно открыть (*File-Open*) в программе gpsim.

Первый же отладочный прогон показывает мне, что условие while x == freq_ch следует заменить на x != freq_ch. Кроме того, внутрь цикла while следует добавить элемент Assignment (присваивание) в который записать x = x+1, иначе переменная не будет меняться. Пауза в итоге выглядит так:

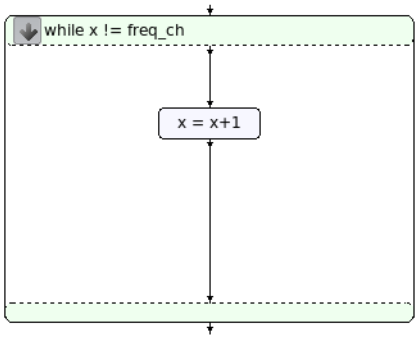


Рис. 3.5. Организация «паузы» на данном этапе

Отладка в gpsim выглядит иначе, чем в FlowCode.

Сразу скажу, что в основном меню следует в разделе *Windows* (окна) выбрать пункты *Source* (исходник) и *Breadboard* (макет). Первым выбором открывается окно наблюдения за исходным текстом программы, по которому будет перемещаться курсор отладки, а вторым действием мы открываем окно «макетной платы», где появится изображение микросхемы с отображением состояния выходов.

В окне наблюдения за исходным текстом можно задавать точки останова (щелчком клавиши мышки по адресу, слева), можно выбирать для наблюдения за их состоянием переменные, щелкнув по ним правой клавишей мышки после выделения и выбрав из выпадающего меню пункт *Add to watch* (добавить в окно наблюдения), или проделать еще ряд операций, характерных для отладки. Что можно, как это сделать — лучше почитать руководство к программе gpsim (на моем сайте есть перевод).

При отладке программа долго «крутится» в «паузе». Светодиод RA0 горит. Пора пришла проверить, передается ли значение переменной freq_ch в эту «паузу»?

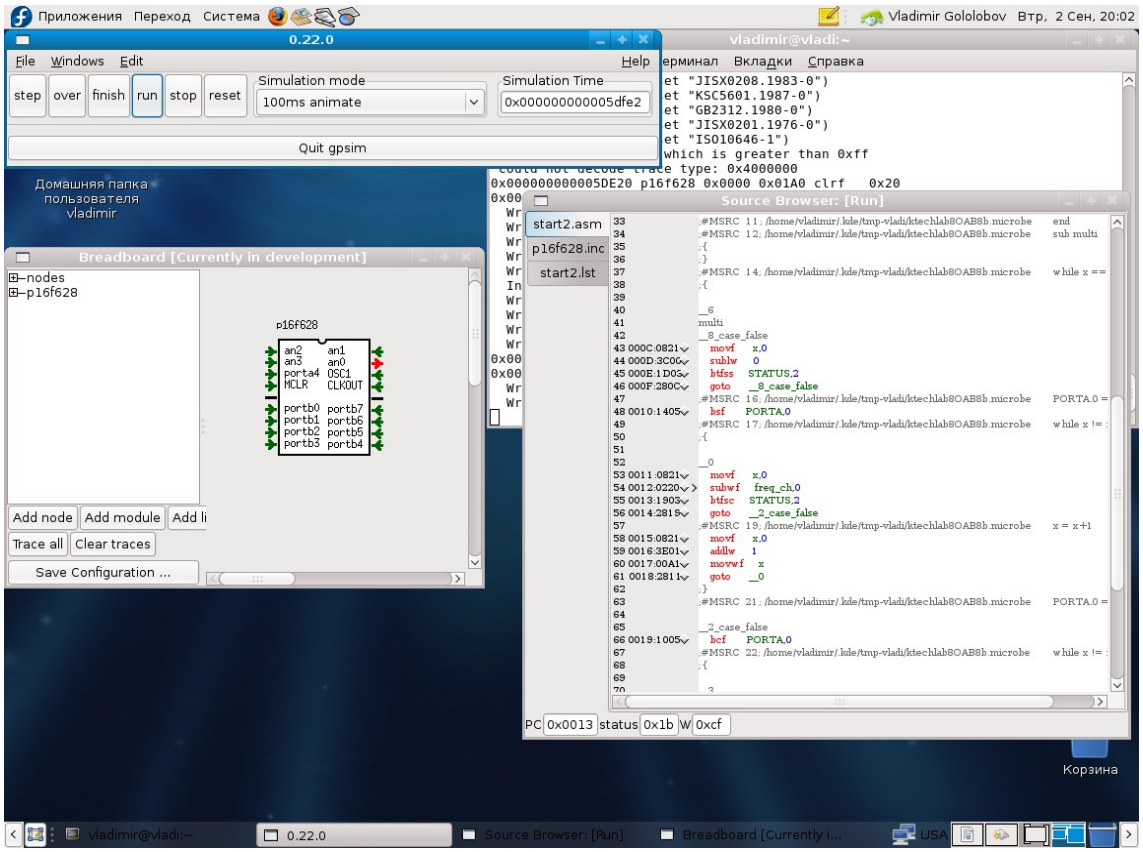


Рис. 3.6. Отладка программы в gpsim

Передается, но... надо еще раз «подправить» блок программы мультивибратора. Я, «на всякий случай» заменил переменные x: в первом случае на i, во втором на k. Так их удобнее наблюдать, да и хотелось бы, чтобы они не перепутывались. Но после первого прохода цикла эти переменные сохраняют значение, которое «набрали» за время работы цикла, и при втором проходе цикл завершается «досрочно». Правка свелась к обнулению этих индексных переменных перед входом в цикл.

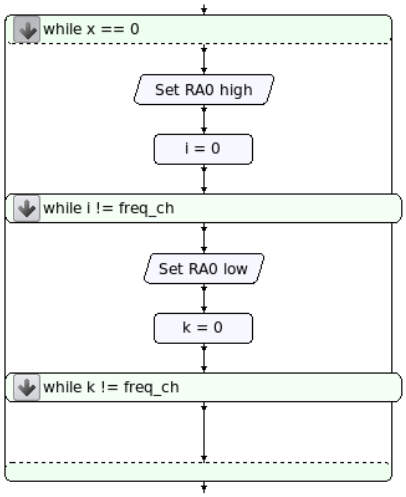


Рис. 3.7. Исправленная версия программного блока работы мультиметра

Теперь при отладке и первый проход, и второй проход циклов «паузы» проходят

правильно. И можно продолжить работу над проектом.

Продолжая подход к программе, обозначенный еще в FlowCode, я хочу заменить выходы, полученные «по умолчанию» как выводы порта A, выходами порта B, а порт A использовать для ввода. Это не сложно сделать, изменяя свойства соответствующих элементов.

Для ввода программа KTechlab в наборе программных элементов вкладки Flow Parts предлагает два элемента, связанных с вводом: Test Pin State (проверка состояния вывода) и Read from Port (прочитать из порта). Есть смысл попробовать их комбинацию, где последний элемент будет в первом случае передавать значение в переменную «more», во втором в переменную «less» (как и прежде).

Перед тем, как продолжить, хочу отметить, что, внося изменения в программу, полезно проверить, все ли соединения программных элементов были сделаны. Проще всего это сделать после первой трансляции в промежуточный код, названный в программе Microbe.

Выбирая из выпадающего меню пункт *Convert to: microbe*, я задаю имя файла и опции:

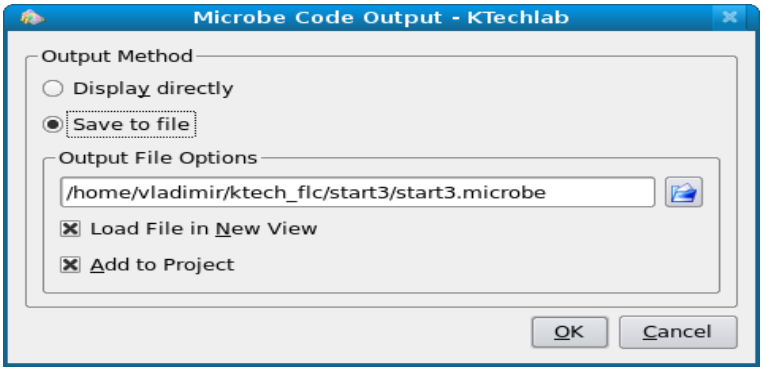


Рис. 3.8. Задание файла для трансляции программы

После трансляции файл сохраняется в заданном мною месте и открывается на вкладке рабочего поля редактора программы.

Конечно, полноценная проверка предполагает знакомство и с программированием, и с языками программирования. Но помогает большое сходство в графическом и текстовом представлениях программы.

Сравнение двух окон редактирования позволяет быстро найти первый графический элемент программы, а затем проверить наличие всех графических блоков. Если соединение не было сделано, то блок отсутствует в текстовом представлении.

К подобным ошибкам приводит неудачное размещение графических программных блоков, когда из стрелки (метки) соединения накладываются, создавая иллюзию соединения, тогда как соединение не было сделано. В спешке это упущение легко не заметить. Позже, когда элементы программы «свернуты» это сделать еще труднее.

Самое разумное, если не считать того, что спешка всегда вредна, проверять каждый шаг, связанный с соединением графических модулей, до того, как «сворачивать» окно модуля. Можно использовать для проверки вышеописанный метод или, просто, перемещать компоненты по рабочему полю редактора, проверяя, потянутся ли за ним линии соединения.

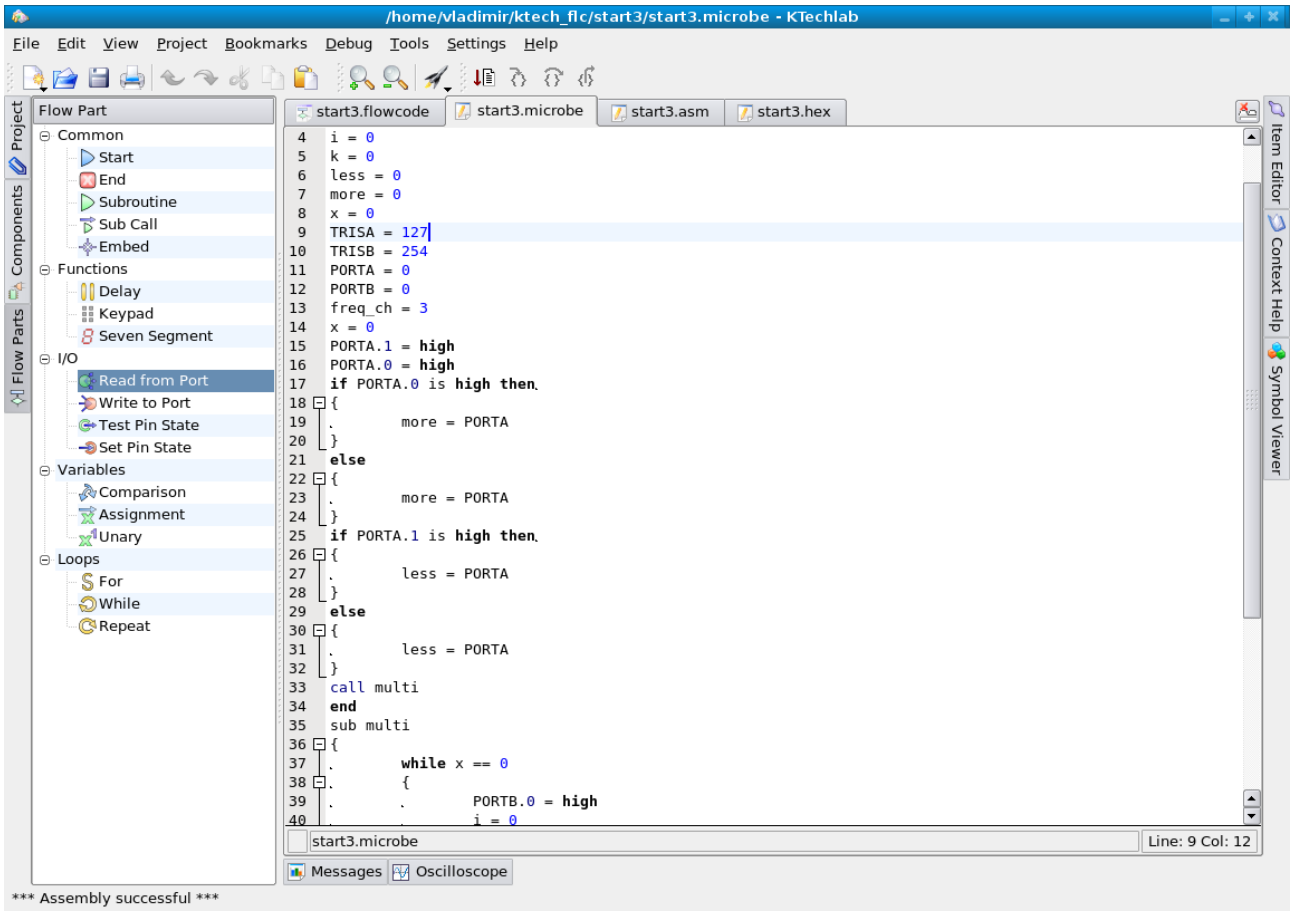


Рис. 3.9. Вид кода программы на вкладке файла .microbe

Вернемся к программе.

Меня интересует, что будет прочитано элементом Test Pin State?

Или, иначе, как это будет отображено элементом Read from Port? Пока я не знаю, как мне удобнее реализовать этот программный блок опроса кнопок управления. Какой программный элемент удобнее для моих целей? Позже, если я сейчас ошибусь в выборе, будет труднее внести исправления, а сейчас я даже не начал работу над этим этапом и смогу вернуться к началу, почти не потеряв ни времени, ни сил. Кроме того, полезно при освоении работы с контроллером параллельно осваивать среду программирования. Иногда, бросив взгляд на компоненты программы, ты воспринимаешь их не такими, какие они есть в действительности, а такими, какими ты хотел бы их видеть. Впоследствии предубеждение может стать причиной ошибок или неприязни к программе.

Один из примеров такой предвзятости — условие равенства в языке Си. Нам привычнее писать if x = 0, а не так, как следует if x ==0, и мы подсознательно, стоит ослабить внимание, пишем «как слышим». Притом, что подобная ошибка при чтении текста программы не бросается в глаза. А, когда она обнаруживается после проверки и отладки программы, досада на себя бывает больше, чем того заслуживает ошибка. Это же можно отнести к служебному слову ассемблерной программы END. Порой забываешь его добавить, и при трансляции можешь получить множество ошибок, которые невозможно отыскать в силу их отсутствия.

Проще всего провести проверку в отладочной программе. Вид программы с добавленными для отладки элементами установки выводов:

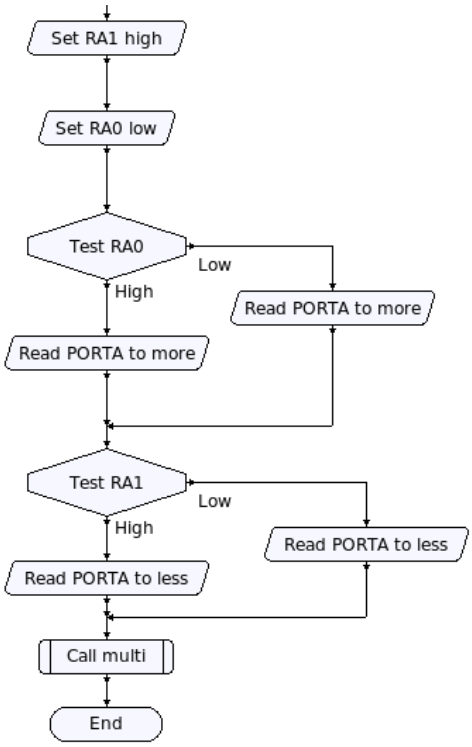


Рис. 3.10. Фрагмент программы с отладочными «добавками»

И еще, перед запуском отладочной программы, после трансляции программы на ассемблер и в hex-файл, полезно заглянуть на вкладку ассемблерного кода.

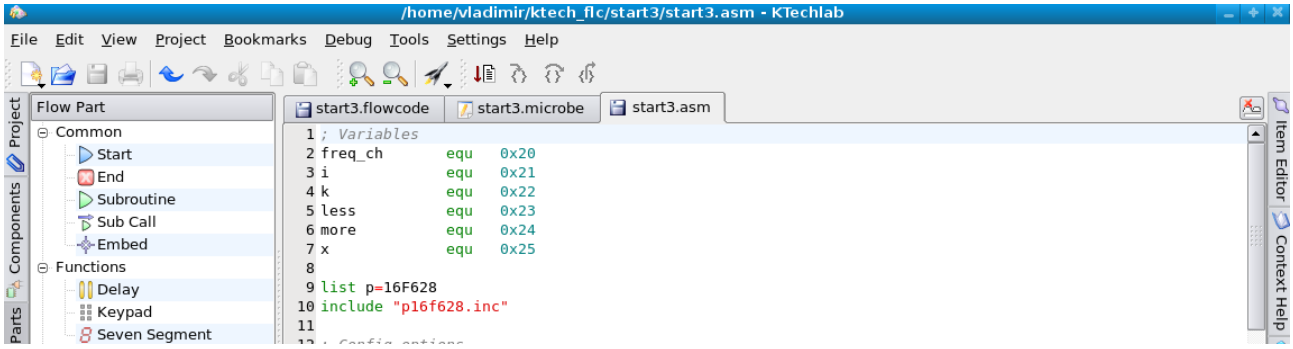


Рис. 3.11. Фрагмент кода программы на ассемблере

Запись `more equ 0x24` означает, что для переменной «more» зарезервирована ячейка памяти с шестнадцатеричным адресом 24. Вторая переменная «less» расположилась в предыдущей ячейке.

Я уже говорил, что в отладчик gpsim следует загрузить создаваемый при трансляции программы файл с расширением .cod. Открывается он обычным для любого файла образом. Но затем, когда файл открыт, полезно открыть окна наблюдения: в основном меню *Windows-Source* (*Ram, Breadboard*).

В данном случае я выбираю для Simulation Mode (режим симуляции) значение 100ms annimate (анимацию в 100 мс).

Запуск отладки клавишей **Run** приводит в движение все в открытых окнах — маркер перемещается по строкам текста на ассемблере, вывод микросхемы, назначенный на выход,

меняет свой цвет, соответственно тому, высокий или низкий уровень на выводе, значения в ячейках памяти меняются.

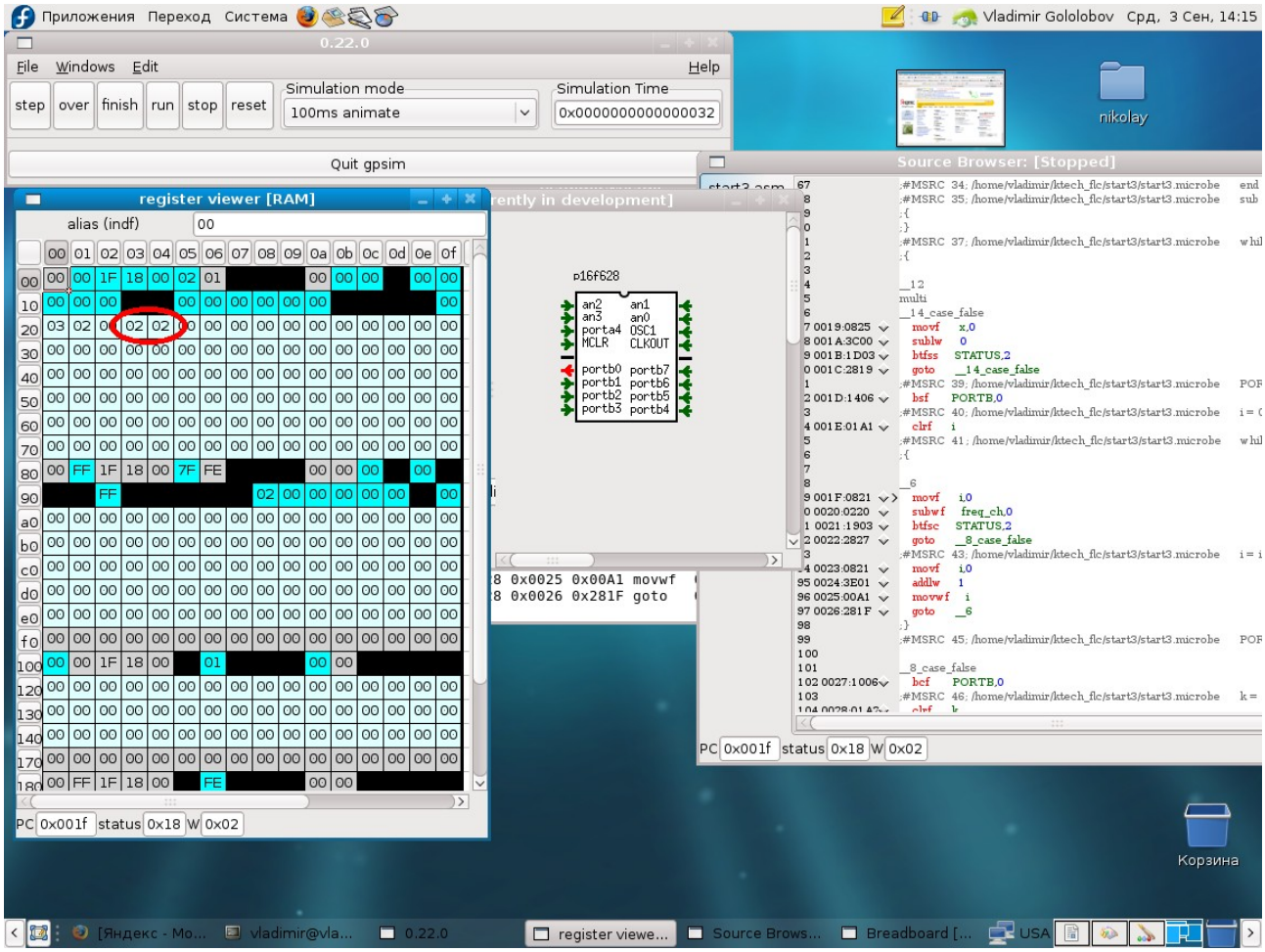


Рис. 3.12. Отладка программы в gpsim

Я выделил ячейки памяти, где «живут» переменные «more» и «less». Конечно, никаких чудесных явлений — бит RA0 в состоянии никого уровня, RA1высокого, число — 2.

Я столь подробно остановился в этом месте работы над программой, хотя результат был очевиден с самого начала, не только по причине своей недалекости, но и оттого, что полезно лишний раз напомнить — иногда отлаживание программы можно, и я считаю, нужно, начинать до создания полной программы. Проще сопоставить свое представление о том, что должна делать программа, с реальным поведением программы в маленьких фрагментах, чем обнаруживать это в виде ошибок в большой программе. И потом, в «доисторические» времена программирования отладочные средства не были столь развитыми, как сегодня, а для осмысления поведения программы в нее добавляли «маркеры» — вставки только для отладки. Сегодня отладочные средства и разнообразны, и удобны, но не стоит забывать об этих «маркерах», иной раз они оказываются очень полезны. Например, при записи в ячейки памяти EEPROM, запись может быть инверсной. Без проверки этого можно сделать ошибку, которую трудно будет выявить. Да и, наконец, мне очень хотелось лишний раз упомянуть программу gpsim. Что я и сделал.

А собирался... добавить в программу опрос клавиш для увеличения и уменьшения частоты генерации. Опрос, конечно, следует вести непрерывно, то есть, использовать бесконечный цикл. Или использовать прерывание. Прерываний в KTechlab я не нахожу, да и использовать

их предпочитаю только тогда, когда без них не обойтись.

Кнопки. Кнопки управления частотой. Они соединяются с выводами RA0 (больше) и RA1 (меньше). Выводы «подтянуты» резисторами к высокому уровню, а кнопки эти выводы «сажают» на общий провод, то есть, в низкий уровень.

Таким образом, низкое состояние вывода должно менять частоту (через переменную freq_ch), а высокое состояние означает, что кнопка не нажата, делать ничего не нужно, можно продолжать генерировать меандр прежней частоты.

Итак, цикл, в котором добавлен опрос клавиш.

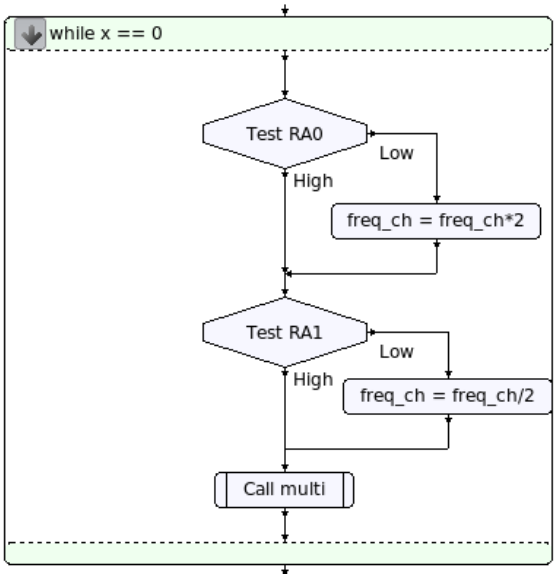


Рис. 3.13. Фрагмент программы опроса клавиш

При желании увеличить частоту мы меняем переменную freq_ch, умножая ее на 2, или делим на 2 при уменьшении частоты. Для этого использовались элементы программы Assignment. Вся программа (без подпрограммы) принимает вид.

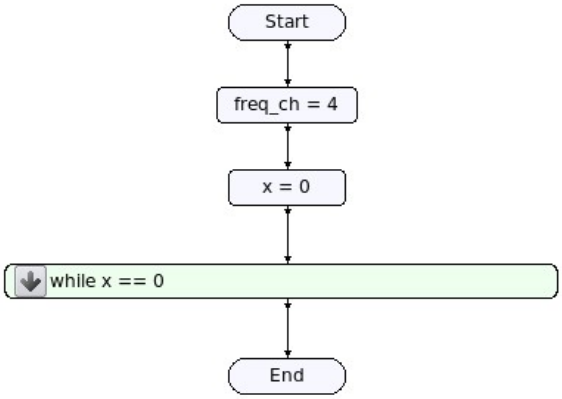


Рис. 3.14. Вид программы на данном этапе

И пора вернуться к FlowCode.

Развитие программы в FlowCode (Windows)

Первое, что следует сделать — перенести все «достижения» в ранее написанную часть программы. Вместо элемента Assignment программы KTechlab, вне всяких сомнений, следует использовать элемент Calculation программы FlowCode. Если я ничего не перепутал, то программа должна получиться следующей.

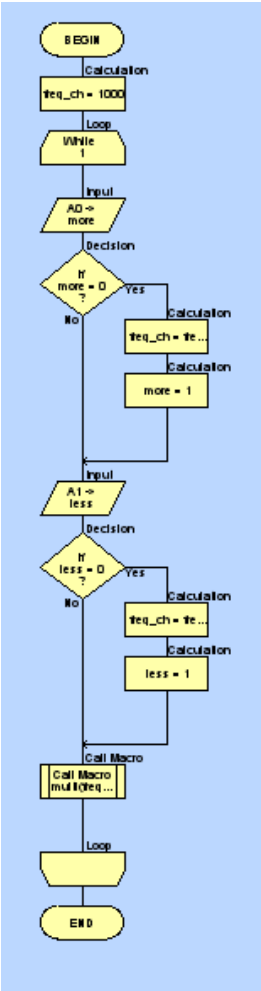


Рис. 4.1. Вид программы (на данном этапе) в среде программирования FlowCode

Определенное удобство программы FlowCode в том, что внешние элементы можно добавить для отладки с панели добавочных элементов, а отладку запустить непосредственно в среде разработки без трансляции файла. Поэтому, добавив элемент Switches, я запускаю отладку, чтобы убедиться...

...программа не реагирует на мои попытки изменить частоту генератора.

Используем прием, которым мы уже пользовались: ставим точку останова (правая клавиша мышки и *Toggle Breakpoint* — переключить точку останова, в выпадающем меню) на второе ветвление. Программа останавливается, но повторный запуск на продолжение выполнения программы ее уже не останавливает, несмотря на наличие точки останова. То есть, программа повторно не попадает в точку останова. И где же она?

Есть единственное место, где она может быть — подпрограмма. Подпрограмма — это бесконечный (в данный момент) цикл. Попав в него после «схода» с точки останова,

программа больше из него не возвращается. Хорошо, что проверил.

Я вижу два варианта, как исправить упущение. Первый самый простой. Добавим в программу переменную «lp» (loop, цикл). До входа в подпрограмму присвоим ей значение «1». Цикл подпрограммы изменим условием While lp = 1, а в конце подпрограммы присвоим этой переменной значение «0».

Теперь мы попадаем в подпрограмму и возвращаемся из нее, что можно проверить с помощью точек останова, создавая их внутри подпрограммы и вне ее.

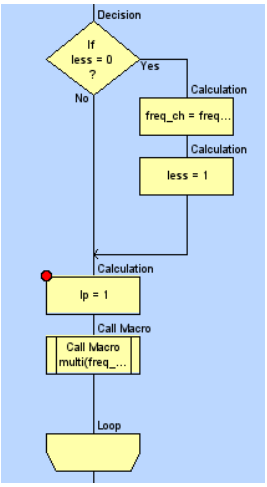


Рис. 4.2. Исправленный фрагмент программы

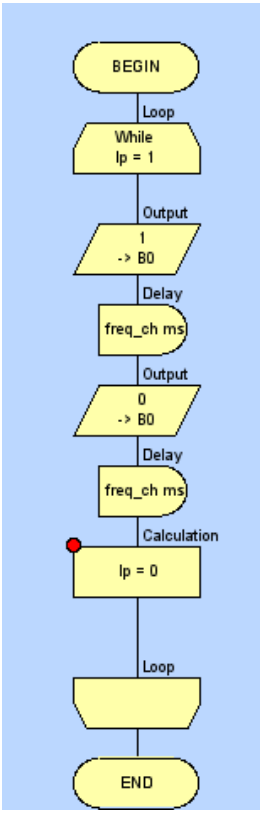


Рис. 4.3. Исправленная подпрограмма

Второй путь — внести опрос клавиатуры в подпрограмму. Правда, при этом смысл

подпрограммы полностью теряется. Но отладка программы выявляет еще одно: меняя переменные «more» и «less», я думаю об увеличении и уменьшении частоты, а меняю при этом длительность, то есть, период. Все с точностью «до наоборот». Вовремя заметил. Изменение (заменить операции на обратные) это пара щелчков мышкой.

Двигаемся дальше. Я проверяю клавиатуру условием if more = 0, поскольку задумывал «подтянуть» входы резисторами, а кнопкой «приземлять» их. Однако в программе FloeCode внешний элемент Switches в разомкнутом состоянии оставляет вход в состоянии «0», а при замыкании подает на вход «1». Срочно исправлять (и не забыть вернуть к первоначальному варианту перед прошивкой!).

Есть еще одно решение проблемы с ловушкой в бесконечном цикле — использовать прерывание. Хотелось бы его опробовать, но, пока я старательно вносил правку в «стройную последовательность» своих ошибок, на горизонте (мысленном) замаячила еще одна проблема.

Что за проблема? Проблема пауз, с одной стороны, и проблема времени опроса с другой. Клавиши изменения частоты (больше-меньше) — сейчас временные интервалы большие и удобные для отладки — приходится удерживать нажатыми подолгу. Причина в том, что программа должна «пройти через паузы». Пока она «отдыхает», она не обращает внимание на нажатые клавиши. Но, когда мы приведем длительности к реальным нуждам, скажем, частоты в диапазоне 1-10 кГц, то нажатие, естественное для нас, не заставит ли программу со скоростью заданной частоты увеличивать эту частоту? За время удержания клавиши, например, в одну секунду, программа сможет прочитать нажатие клавиши очень много раз.

Перво-наперво, есть ли такая проблема в данном, конкретном случае?

В данном случае, возможно, нет. Почему?

Немного странный ответ: «Потому, что мы используем механические кнопки, подверженные дребезгу контактов».

Этот дребезг предстоит нейтрализовать. А самый простой способ нейтрализации — добавить паузу после опроса состояния кнопки, если оно менялось, и еще раз, после паузы, проверить, изменилось ли оно? Конструкция программы, примерно, следующая.

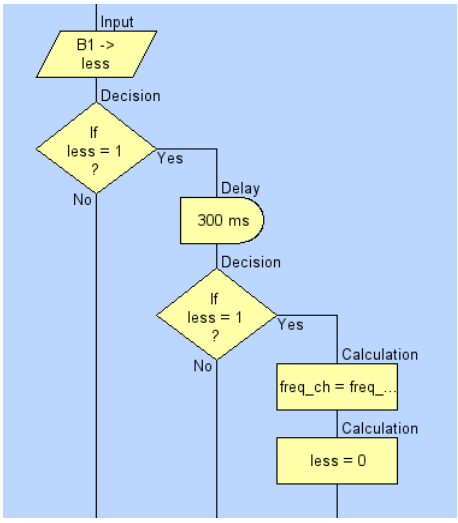


Рис. 4.4. «Антидребезговый» блок программы

Время «антидребезговой» паузы, следовательно, время необходимого удержания кнопки при смене частоты, может быть выбрано равным секунде или двум, когда выполнение других

блоков программы не требуется. И ничто нам не помешает добавить еще одну паузу в секунду или больше перед вызовом подпрограммы. Мы устраним дребезг контактов, его в любом случае нужно устранить, и не позволим программе в этом месте что-то делать, что могло бы помешать «правильно» переключать частоту. Можно использовать и другой прием, например, ждать размыкания кнопки (с «антидребезгом»). Ниже я использую программный элемент Connection Point (точка соединения) и светодиод на выводе A7 для индикации процесса изменения частоты. Пара Connection Point — это в более привычном облике пара, образуемая меткой и оператором Goto (переход к метке).

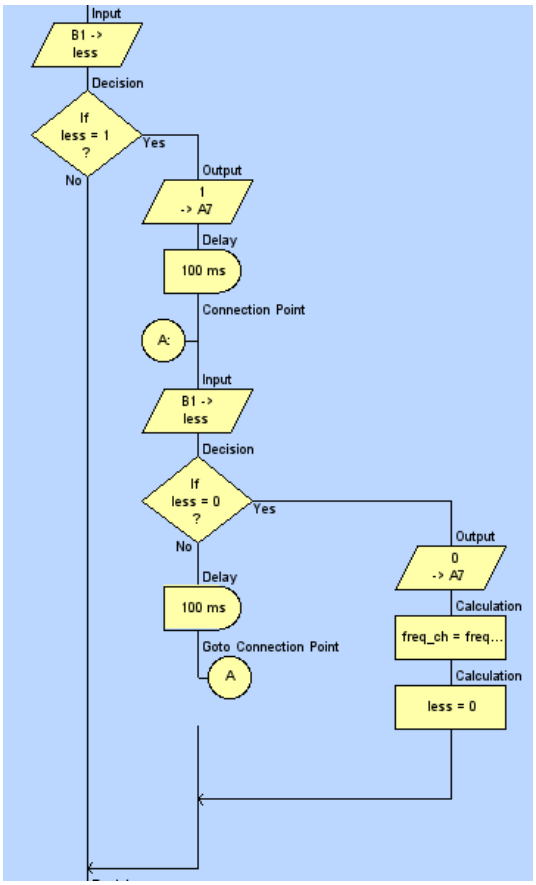


Рис. 4.5. Возможное решение по остановке программы для опроса кнопок

И, наконец, можно вернуться к еще одному варианту решения проблемы «попадания в бесконечный цикл», к варианту с прерыванием.

К программе вида, изображенного на рисунке 4.1, можно добавить прерывание. Для этого есть программный элемент Interrupt. Присваивание переменных «more» и «less» внесем в подпрограмму обработки прерывания.

Если открыть свойства элемента Interrupt (прерывание), то можно увидеть список доступных прерываний.

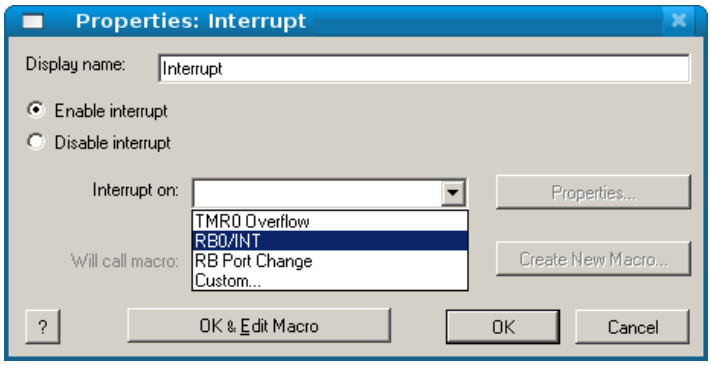


Рис. 4.6. Список возможных прерываний в FlowCode

Прерывания по таймеру меня сейчас не интересуют, а начать эксперимент я хочу с внешнего прерывания RB0/INT.

Программа приобретает вид, показанный ниже.

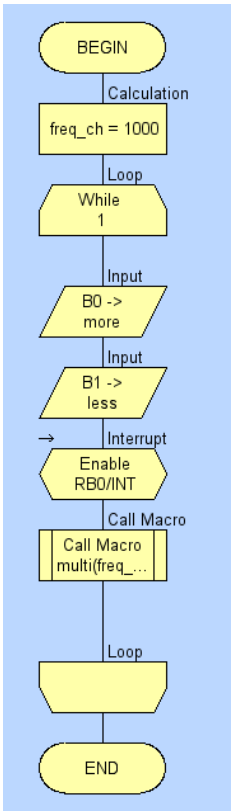


Рис. 4.7. Программа с использованием прерывания

Прерывание чем-то похоже на вызов подпрограммы. Видимо, наличием подпрограммы обработки прерывания.

Подпрограмма обработки прерывания с точкой останова, я хочу посмотреть, попадаю ли я в эту подпрограмму, когда нажимаю кнопку B0, ниже.

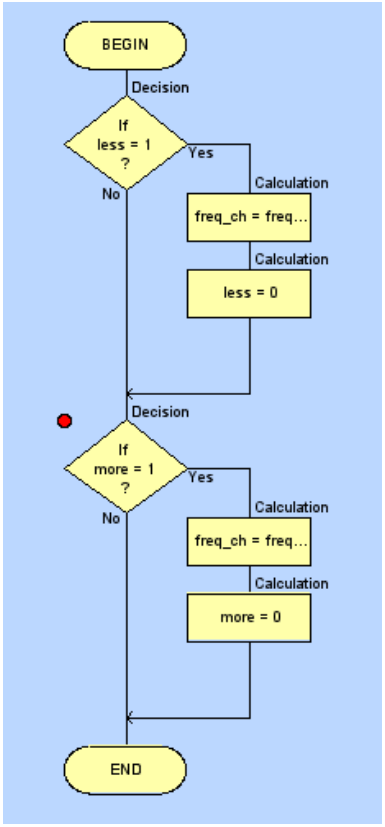


Рис. 4.8. Подпрограмма обработки прерывания

Запущенная программа, после нажатия на кнопку B0 останавливается, как и задумывалось, на точке останова.

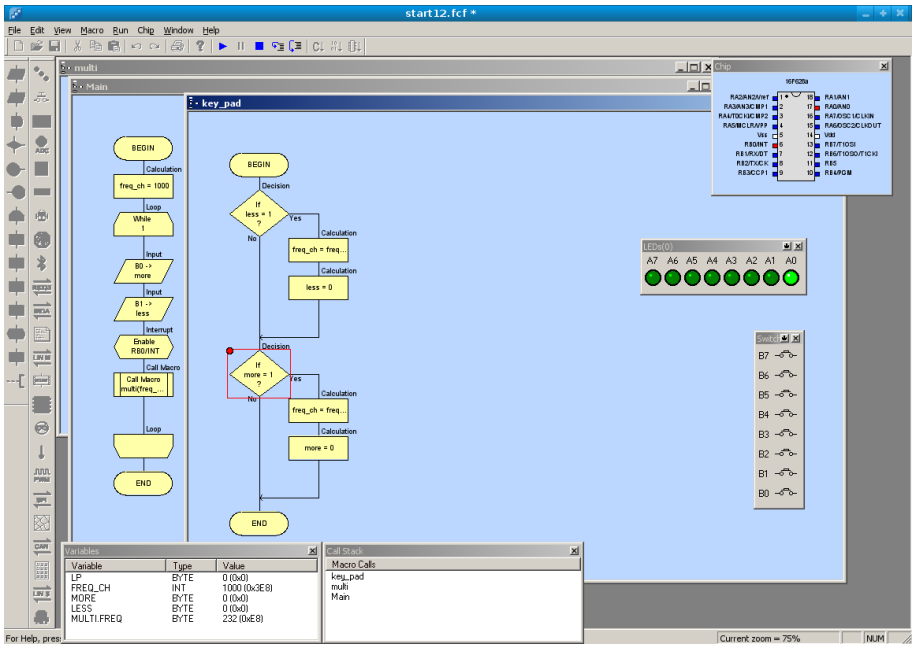


Рис. 4.9. Работа программы с прерыванием

Если бы это прерывание (RB0/INT) было единственным доступным механизмом

прерывания, то и тогда можно было бы его использовать: вместо двух кнопок будут три, две для изменения частоты, третья кнопка, чтобы изменения вступили в силу. Но...

Список штатных прерываний, предложенный программой FlowCode, продолжается прерыванием по RB Port Change (прерывание по изменению состояния порта B). В описании микроконтроллера PIC16F628A можно найти, что изменение состояния входов RB4-RB7 может инициировать прерывание, если оно разрешено. Выбор входов для соединения с кнопками управления частотой не носит принципиального характера, можно легко заменить вход RB0 на RB4, а RB1 на RB5. Не сложнее изменить свойства программного элемента Iterrupt.

Подпрограмма «работы мультивибратора» остается прежней. А в подпрограмму обработки прерываний добавлены опросы клавиш на входах B4 и B5.

При необходимости в работу можно внести описанные выше «антидребезговые» добавления. А в целом, можно сказать, что в разных ситуациях лучше использовать те приемы работы, которые дают должный эффект, лучше выполняют работу устройства и хорошо вам знакомы и понятны, когда вы достаточно ясно представляете себе, почему выбрали тот или иной вариант. Аналогичными соображениями вы руководствуетесь и при выборе типа и модели микроконтроллера.

Вид основной программы и подпрограммы обработки прерываний показан ниже.

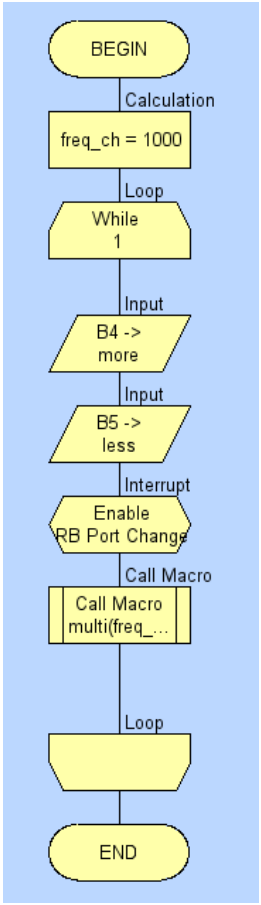


Рис. 4.10. Программа, использующая прерывания по изменению состояния порта B

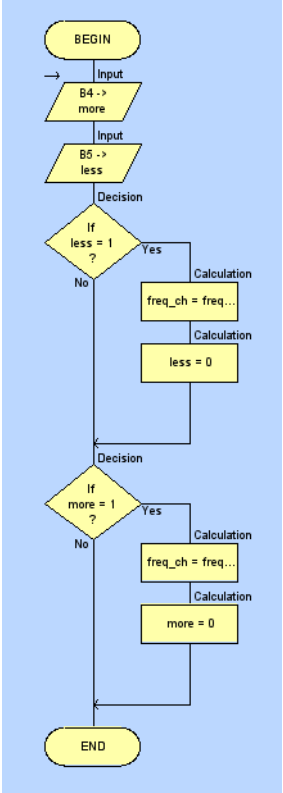


Рис. 4.11. Подпрограмма обработки прерываний

Следующий этап работы.

Кроме изменения частоты генератора мы намеревались отображать эти изменения на семисегментном индикаторе. Добавим его в программу, удалив из нее светодиоды.

Рис. 4.12. Семисегментный индикатор из набора дополнительных элементов

Как и остальные элементы, индикатор имеет свойства. Для доступа к свойствам подключения индикатора служит кнопка на титульной панели рядом с кнопкой закрывания. При ее нажатии из выпадающего меню следует выбрать пункт *Component Connections...*

По умолчанию анод индикатора подключен к выводу 3 порта А. В дальнейшем этот вывод следует сделать в программе выходом и установить в состояние высокого уровня.

Свойства подключения можно изменить, например, так.

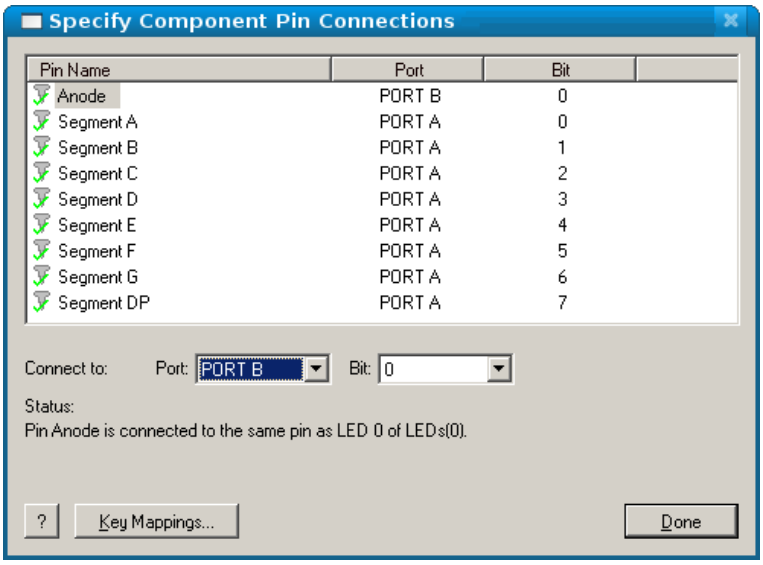


Рис. 4.13. Изменение свойств индикатора

Анод теперь подключен к B0. А с тем, чтобы переключатель Switches не мешал нам распорядиться выводами порта B на наше усмотрение, можно изменить его свойства. Для этого достаточно для нужных бит порта, выделяя их, в окошке Bit: выбрать Unconnected (не присоединен).

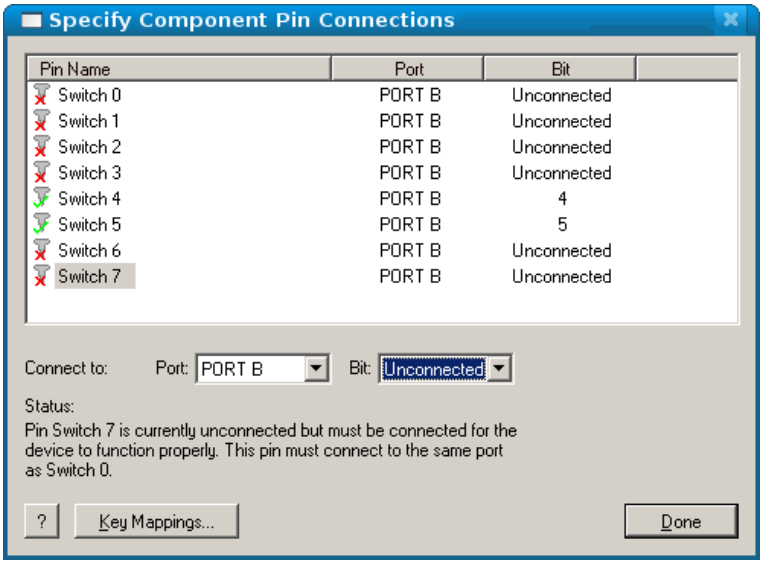


Рис. 4.14. Изменение свойств переключателя

Вывод порта A7 подключен к точке на индикаторе, что позволяет в блоке «работы мультивибратора» использовать этот бит для вывода отображения сигнала, заменив в подпрограмме прежнее значение A0.

После запуска программы все элементы индикатора зажигаются, а точка мигает с выбранной скоростью.

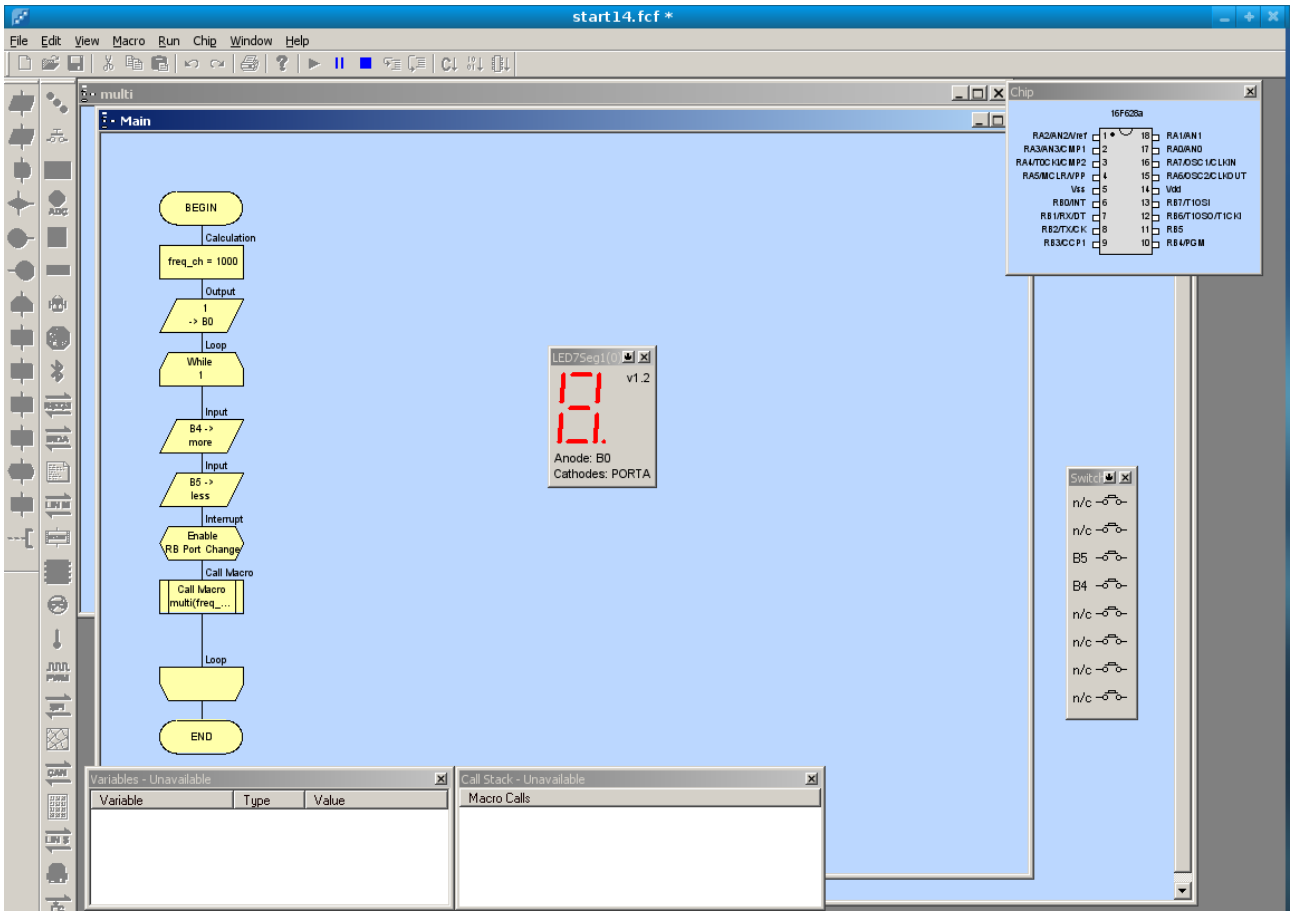


Рис. 4.15. Начало этапы работы с индикатором

Чтобы не усложнять вид программы, а ясно, что порядок работы с индикатором при отображении частот, скажем, в диапазоне 1-9 кГц, это не более, чем повторение одних и тех же программных фрагментов, я остановлюсь на основной частоте 2, а кнопки меньше и больше будут переключать ее на 1 и 4 (деление и умножение на 2, заложенные ранее).

Обслуживание индикатора можно оформить в виде подпрограммы. Назовем ее «indicator». И вначале погасим индикатор, «записав единицы» во все биты порта. Затем можно, используя ветвление, обслужить все задуманные значения частоты, зажигая нужные сегменты (установкой соответствующего бита в «0»).

На этом этапе работы, а программа уже несколько «подросла» количественно, можно проверить и исправить ошибки, если они есть, можно дописать такие фрагменты, как выход частоты из заданного диапазона, например, высвечивая «E» на индикаторе. Можно сравнить полученную конструкцию с той, что могла бы быть получена без использования микроконтроллера. Устройство достаточно простое, его можно реализовать, используя цифровой счетчик и тактовый генератор на вентилях. А можно удовлетвориться достигнутым и перейти к макетной плате и внешнему оформлению устройства.

Можно и продолжить разработку, введя еще один семисегментный индикатор для расширения диапазона частот.

Главное, вы начали работу с микроконтроллерами, создав первое устройство, и вам ясно, что создание второго и третьего — все в вашей власти. Нужно только продолжить работу.

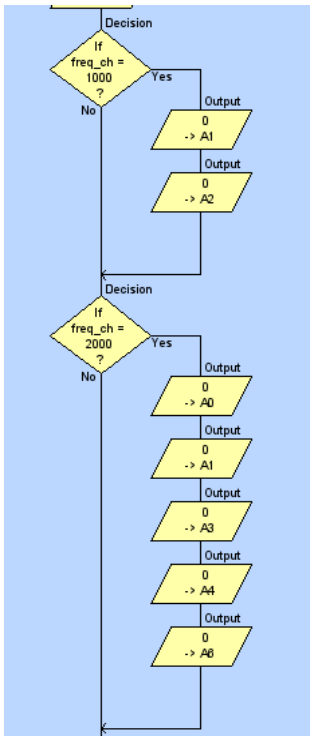


Рис. 4.16. Фрагмент подпрограммы обслуживания индикатора

Создание схемы устройства при заданных назначении, параметрах и прочих критериях выбора, как правило, дает несколько возможных решений. Программирование тоже. И иногда полезно при работе с микроконтроллером вспоминать, как могло бы это реализоваться в «железе». Приведу пример.

Мы говорили о дребезге контактов. Из схемотехники мне вспоминается решение, использующее кнопку с контактами на переключение и RS-триггер. При нажатии кнопки управляющее напряжение подается, скажем, на вход установки, а при отпускании на вход сброса. Сколько бы ни дребезжали контакты, формируется единственный переключающий импульс. Аналогичную схему можно применить и в программе для микроконтроллера.

Ниже показана программа, где входы A0 и A1 используются для установки и сброса переменной flag. Состояние флага отображают светодиоды B0 — флаг установлен, B1 — флаг сброшен.

Можно проверить, что после первого нажатия кнопки A0 флаг устанавливается, светодиод B0 зажигается, и повторные нажатия на эту кнопку (дребезг контактов) не меняют состояния. Такое решение по устранению дребезга контактов, вернее, влияния дребезга контактов на работу устройства, в некоторых случаях может оказаться предпочтительнее других. А использование аналогий, как прием, вполне оправдывает себя.

Начиная создавать программу для микроконтроллера, далеко не всегда удается «увидеть» всю ее целиком и осознать все детали. Пройдя по, казалось бы, верному пути достаточно далеко, иной раз приходится возвращаться назад. Чтобы легче было вернуться назад, полезно сохранять отдельные этапы работы в виде отдельных файлов программы. Если в названии файла отобразить название этапа, то его легче обнаружить в образующемся длинном списке файлов. Как я уже говорил, порой следует транслировать программу до конца, чтобы посмотреть ее вид на языке Си или ассемблера. Если в программе есть ошибка, программа может не транслироваться в hex-файл и сообщать об этом при трансляции — повод еще раз все проверить и найти причину этого.

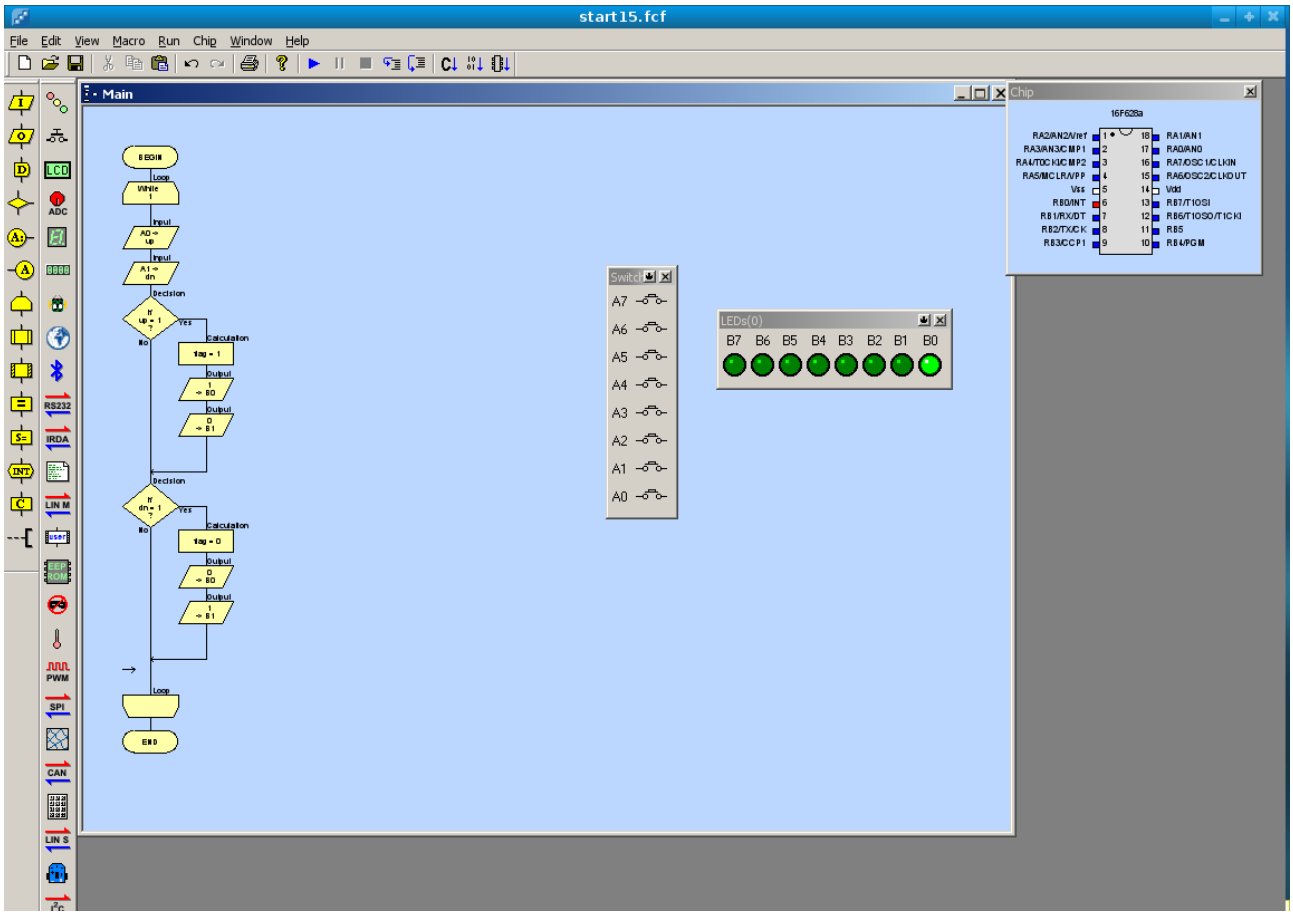


Рис. 4.17. Еще один вариант программы «антидребезга»

После первого шага, пусть создания простейшей программы с мигающим светодиодом, первого шага, доведенного до конца, то есть, до макетной платы с микроконтроллером и «фонариком», у вас обязательно появится много идей, как с пользой применить микроконтроллер. Реализуя эти идеи, вы больше узнаете о программе FlowCode, научитесь работать с ней. Но, возможно, легкое чувство неудовлетворенности не оставит вас. Знатоки, едва речь заходит о микроконтроллерах, в один голос утверждают, программировать нужно на ассемблере.

Я не отношусь к знатокам. Работает устройство, и хорошо. Как написана программа — какая разница?

Но, если вас это беспокоит, то я посоветовал бы, конечно, если вы не знакомы с языками программирования, использовать программу FlowCode, как удобный самоучитель по программированию. Сначала на языке Си. Язык более высокого уровня, а, следовательно, более простой, чем ассемблер. Да, и более универсальный. Итак.

FlowCode как самоучитель программирования на языке Си

Вернемся к самой простой программе — помигать светодиодом. Даже упростим задачу — зажечь светодиод.

Программа в графическом представлении выглядит так.

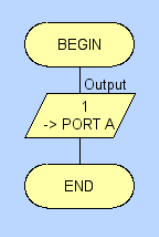


Рис. 5.1. Прототип программы для трансляции в код на языке Си

Следующее, что мы сделаем, выберем в основном меню раздел *Chip*, в котором выберем пункт *Compile to C...* (компилировать на Си). После успешной компиляции можно открыть файл на языке Си в блокноте с помощью пункта *View C...* того же раздела *Chip* основного меню. Значительная часть файла, все строки, начинающиеся с двойной косой черты //, это комментарии, то есть, пояснения для читающего файл, все комментарии игнорируются при дальнейшей трансляции кода программы. Хотя, подчас, комментарии с использованием кириллицы, вставленные в графическое представление программы, вызывают ошибку при компиляции.

Очень много строк, начинающихся со значка # и слова define — определить. Это служебное слово, за которым следует, что должно быть определено, чтобы при дальнейшей трансляции компилятором, используемым программой FlowCode, текста программы на ассемблер и дальше в загружаемый код, все было определено и обозначено. Что именно следует определять, зависит от ваших нужд, того как будет написана программа и конкретного компилятора.

Есть еще одно служебное слово #include — включить, предназначенное к добавлению в программу других файлов, чаще файлов заголовков, которые активно используются в языке Си. В данном случае, как это следует из комментария, включены функции, необходимые для работы с контроллером.

Сама программа (не умаляя значения всего остального) выглядит почти так же коротко, как ее графическое представление.

```
void main()
{
	//Инициализация
	cmcon = 0x07;

	//Output: 1 -> PORT A
	trisa = 0x00;
	porta = 1;
}
```

Для работы с программами на языках Си и ассемблер и PIC-контроллерами я считаю наилучшей средой разработки MPLAB. В Windows у меня версия 8.10, в Linux 7.11. Но в Linux какие-то проблемы с компилятором. MPLAB позволяет работать с разными компиляторами языка Си. Есть компилятор производства HI-TECH, его демо-версия доступна для свободного скачивания, но после обязательной регистрации на сайте. Если

попытаться воспользоваться чужой версией, то это может не получиться. Есть ограничения на оперативную память, возможно, еще какие-то, но для изучения программирования микроконтроллеров на языке Си компилятор вполне подходит.

После первой установки программы MPLAB и компилятора следует проверить, все ли определилось с местом установки компилятора. В основном меню в разделе *Project* есть пункт *Set Language Tool Locations...*, открывающий диалоговое окно задания расположения трансляторов с разных языков.

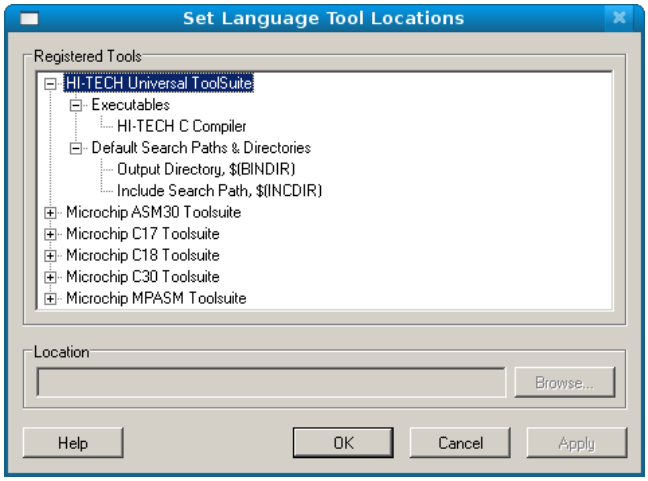


Рис. 5.2. Диалоговое окно расположения компилятора в MPLAB

Если с расположением есть неувязки, то клавиша **Browse...** поможет указать место (Location), где расположены нужные файлы. Как видно на рисунке, MPLAB работает со многими языковыми средствами.

Для работы с компилятором HI-TECH текст программы, показанный выше, следует слегка «подправить», переписав команды в верхнем регистре. Что следует сделать при использовании компиляторов других производителей, мы рассмотрим, когда (и если) будем говорить о других компиляторах. В любом случае, используя короткие фрагменты программ в FlowCode, можно получать их код на языке Си. И, если в FlowCode ясен смысл фрагмента, то полученный код на языке Си можно использовать для написания кода других программ на языке Си. В этом и есть смысл, который я вкладывал в название главы.

В качестве примера пользы от сочетания таких средств, как FlowCode и MPLAB, я хочу привести решение одного маленького неудобства, возникшего у моего знакомого. Ему понадобилось при работе с FlowCode сделать паузу на 30 секунд. Программа предоставляет штатное решение — программный элемент Delay. Но во время паузы работа программы останавливается, а нужно, чтобы она опрашивала клавиатуру. Самое простое — добавить в программу цикл опроса клавиатуры, который выполнялся бы столько раз, сколько нужно для паузы в 30 секунд. Но сколько раз следует выполнить цикл?

Посмотрим, как такое неудобство можно решить с помощью MPLAB. Создадим в FlowCode цикл, но с «мигающим светодиодом». Оттранслируем этот фрагмент на Си, «подправим» для компилятора HI-TECH и проверим его работу в MPLAB.

```
#include <pic16f62xa.h>

int i;

void main()
{
```

```
        //Loop: While 1
        while( 1 )
        {
                //Output: 1 -> PORT B
                TRISB = 0x00;
                RB0 = 1;

                //Delay
                for(i=0;i<10;i++);

                //Output: 0 -> PORT B
                TRISB = 0x00;
                RB0 = 0;

                //Delay
                for(i=0;i<10;i++);
        }
}
```

Переменную int i (целое) я добавил «от руки», как и цикл for.

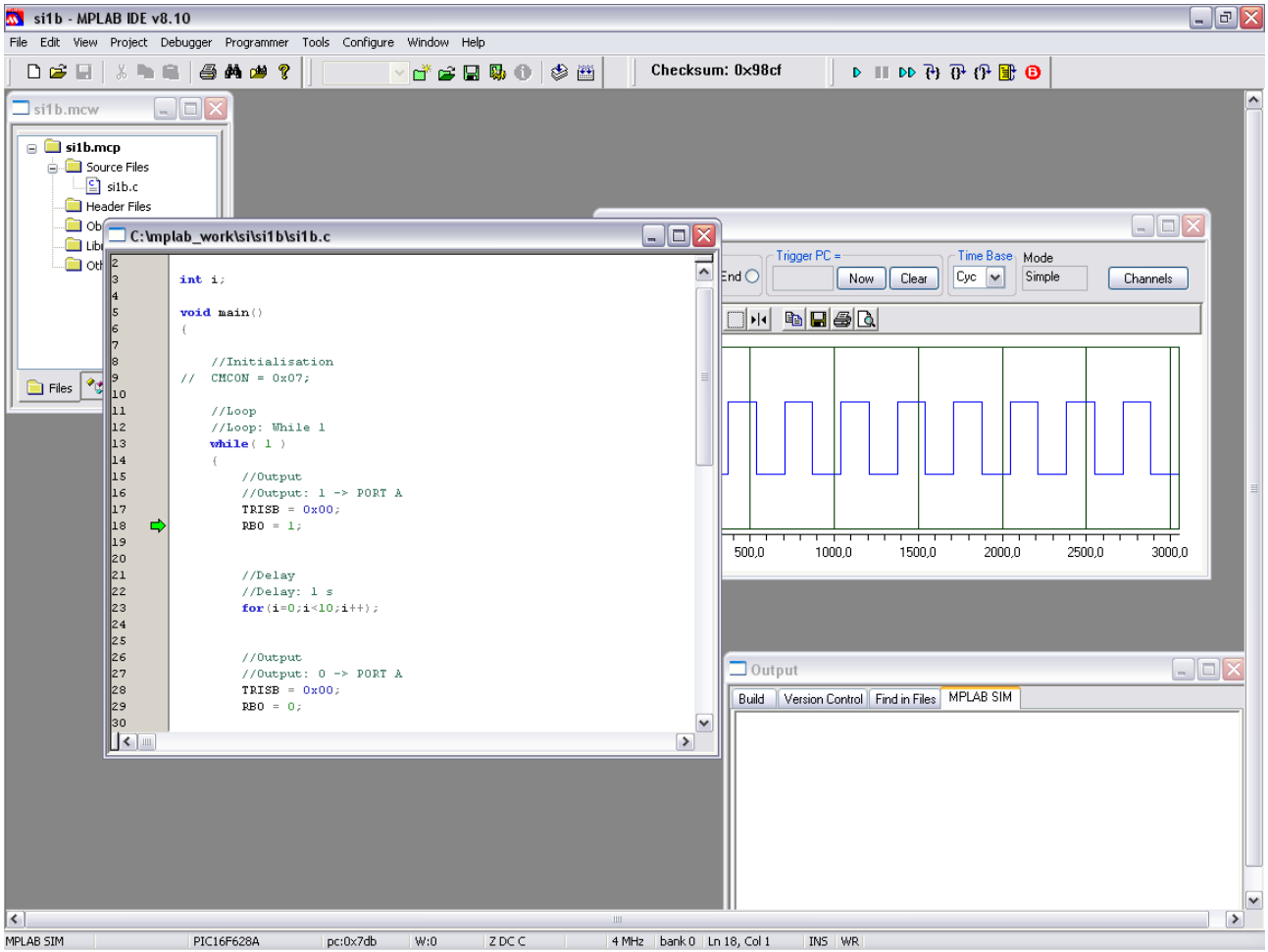


Рис. 5.3. Проверка работы программы в MPLAB

Проверка осуществлялась с помощью отладчика (в основном меню *Debugger-Animate*), а осциллограмму «показал» логический анализатор (основное меню *View-Simulator Logic Analyzer*). Для этого в окне логического анализатора следует нажать клавишу **Channels**, в диалоге выбрать (в данном случае RB0) предмет наблюдения, который и добавить для

наблюдения клавишей **Add**. После запуска отладчика осциллограмма начнет «заполняться». Если на инструментальной панели логического анализатора выбрать клавишу **Cursor** (клавиша с иконкой ▶▌◀), то появившийся курсор можно установить на импульс, длительность которого нас интересует. Красные прямые на рисунке ниже — это и есть искомый курсор. При наведении курсора мышки на эти прямые курсор мышки превращается в изображение руки с указательным пальцем, которым можно, удерживая левую клавишу мышки, перемещать метки измерительного курсора.

Длительность будет измерена в тактах (cyc), но, зная тактовую частоту, легко получить время (с учетом 10 повторов).

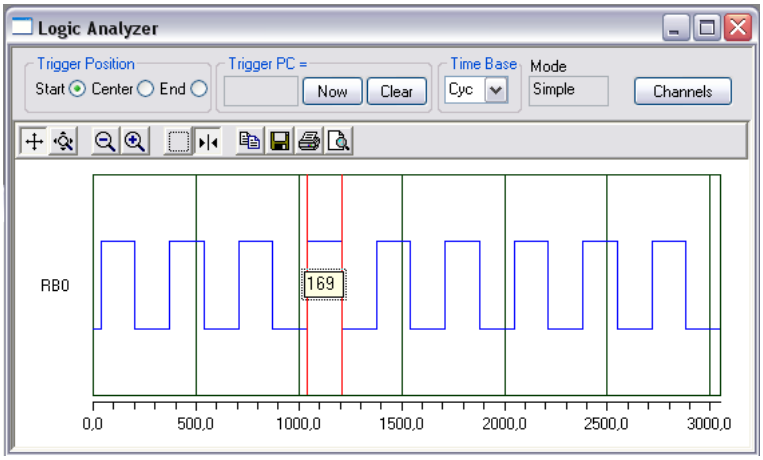


Рис. 5.4. Измерение «длительности» импульса

Признаться, первоначально я выбрал порт A для этого эксперимента и был озадачен — осциллограмма показывала «просечки» напряжения вместо импульсов. Затем я вспомнил, что по умолчанию выводы порта A не имеют «подтягивающих» резисторов. Смена порта (A на B) дала нужный результат.

Чтобы яснее понять, как образуется эта длительность, можно рассмотреть работу ассемблерного кода в MPLAB.

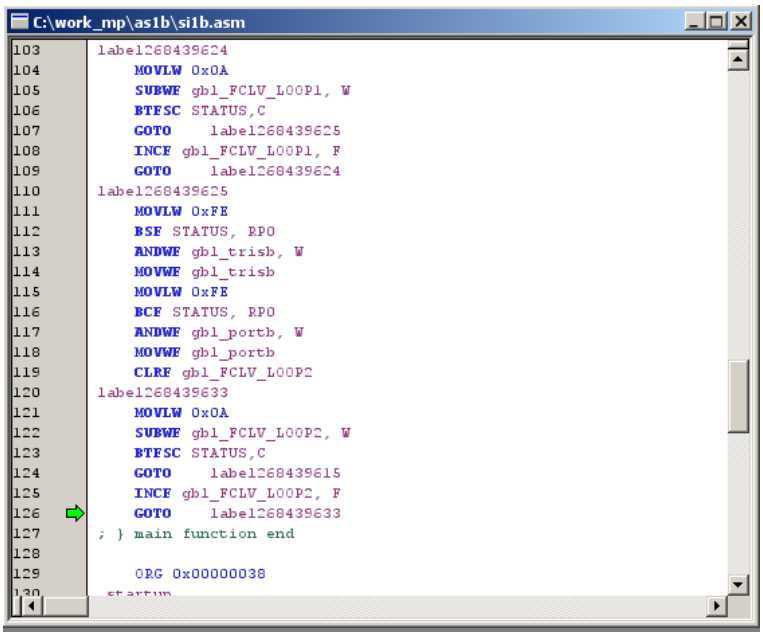


Рис. 5.5. Работа отладчика с кодом на ассемблере

Цикл for десять раз будет повторять набор команд, на которые указывает курсор анимации

отладки. А, зная, за сколько тактов выполняется каждая из команд, можно «вручную» определить общую «длительность» цикла.

Мне кажется, что, используя возможности программы FlowCode, программу MPLAB, хороший учебник по языку Си и руководство к выбранному вами компилятору, можно быстрее овладеть программированием на языке Си (и аналогично на ассемблере) в преломлении к разработке устройств с использованием микроконтроллеров.

KTechlab и язык высокого уровня

В среде разработки KTechlab я повторю предыдущую программу.

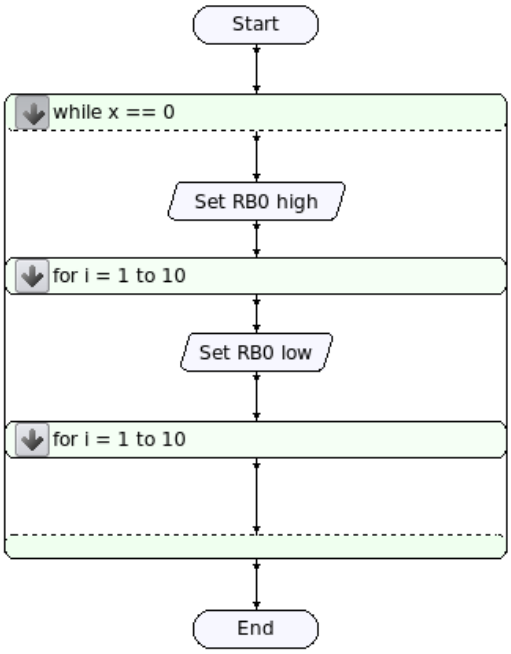


Рис. 6.1. Программа «прототип» в среде KTechlab

Программа KTechlab не предлагает трансляцию на язык Си. Но можно транслировать на язык Microbe. Как выглядит текст программы графического представления в этом случае:

```
P16F628
TRISA = 127
PORTA = 0
PORTB = 0
while x == 0
{
      PORTB.0 = high
      __label_forloop:
      for i = 1 to 10
      {
            goto __label_forloop
      }
      PORTB.0 = low
      __label_forloop__32:
      for i = 1 to 10
      {
            goto __label_forloop__32
      }
}
end
```

Как видно из текста, язык не в полной мере Си, поскольку это Microbe. Для отдельного фрагмента переделка кода не представляет особой трудности, если нужен код именно на языке Си. Но, насколько я знаю, при программировании микроконтроллеров используется не единственный язык высокого уровня Си, есть среды разработки, использующие Basic, Pascal. Чем, спрашивается, хуже язык Microbe, если он позволяет добиться результата не хуже, чем

Си.

Я хочу привести выдержки из руководства KTechlab Handbook, написанного Дэвидом Сэкстоном и Дениэлем Кларком:

> *Microbe компилирует программу, написанную на привычном языке для PIC, как вспомогательную программу для KTechlab. Синтаксис был разработан для согласования с FlowCode программами.*
>
> *Следующие правила относятся к именам переменных и меток:*
>
> - *Они могут содержать только буквенно-цифровые символы [a..z][A..Z][0..9] и подчеркивание "_".*
> - *Они зависимы от регистра (case-sensitive).*
> - *Они не могут начинаться с цифры.*
> - *Они не должны начинаться с "__" (двойное подчеркивание), поскольку это зарезервировано для использования компилятором.*
>
> *Фигурные скобки, {}, показывают начало и конец блока кода. Они могут появиться в любом месте до начала и после завершения блока кода.*
>
> *PIC id (идентификатор) должен быть вставлен вверху программы. Конец основной программы (main program) отмечается с помощью «end». Подпрограмма должна помещаться после «end».*
>
> *Подпрограмма может быть вызвана из любого места в коде. Синтаксис:*

```
sub SubName
{
// Code...
}
```

> *Подпрограмма вызывается с помощью «call SubName».*
>
> *Условный переход. Пример:*

```
if porta.0 is high then
{
      delay 200
}
else
{
      delay 300
}
```

> *Цикл repeat. Пример:*

```
repeat
{
      [statement block]
}
until [expression]
```

> И т.д.

Как и с программой FlowCode, в KTechlab можно переходить от простых графических программных блоков, к блокам на Microbe. Рассмотрим несколько примеров (заодно хочу посмотреть, как работает KTechlab в дистрибутиве ALTLinux 4.0, которым я недавно заменил Fedora 7).

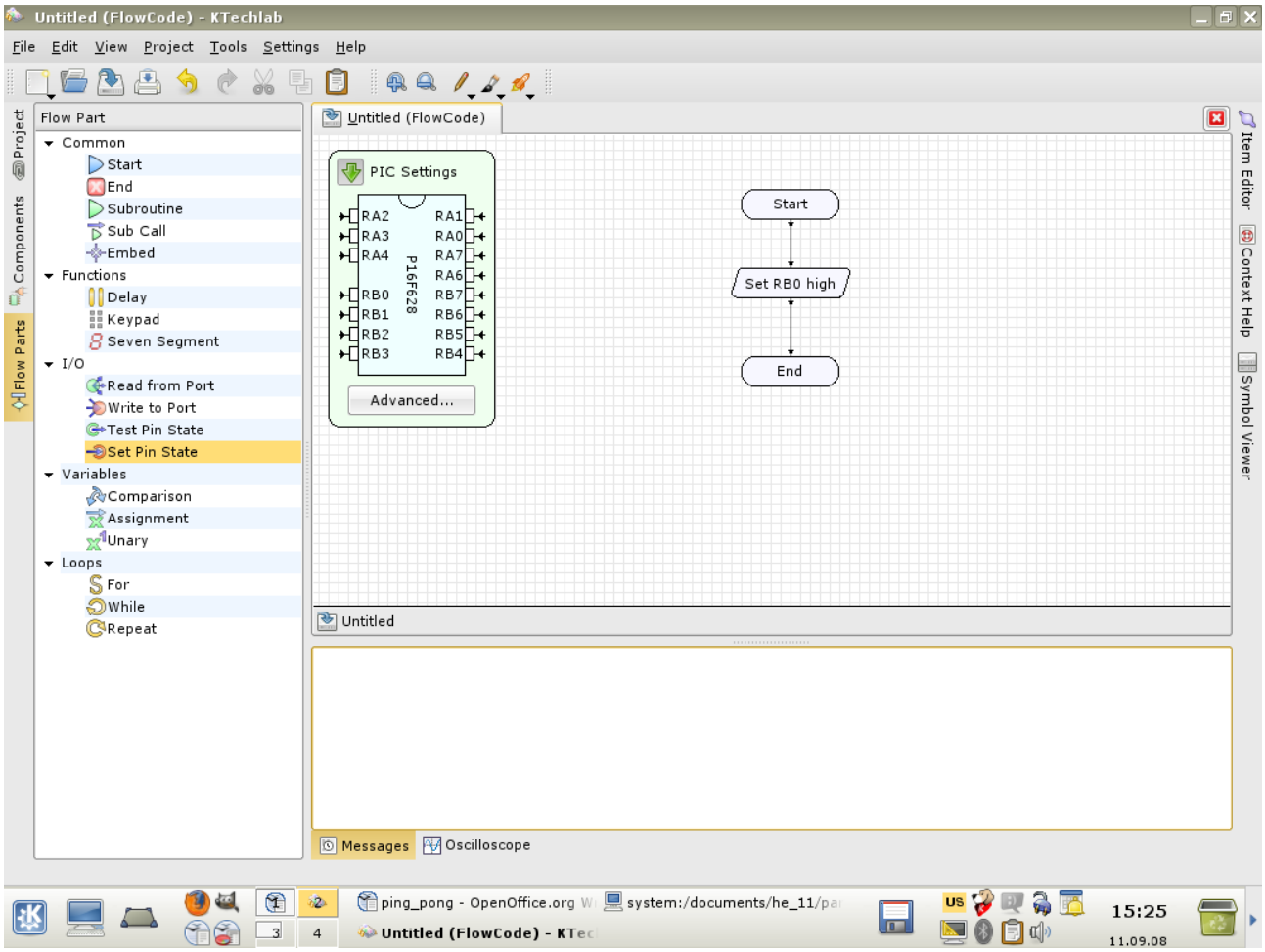


Рис. 6.2. Самая простая программа в KTechlab

После трансляции в microbe можно сохранить блок текста отдельно.

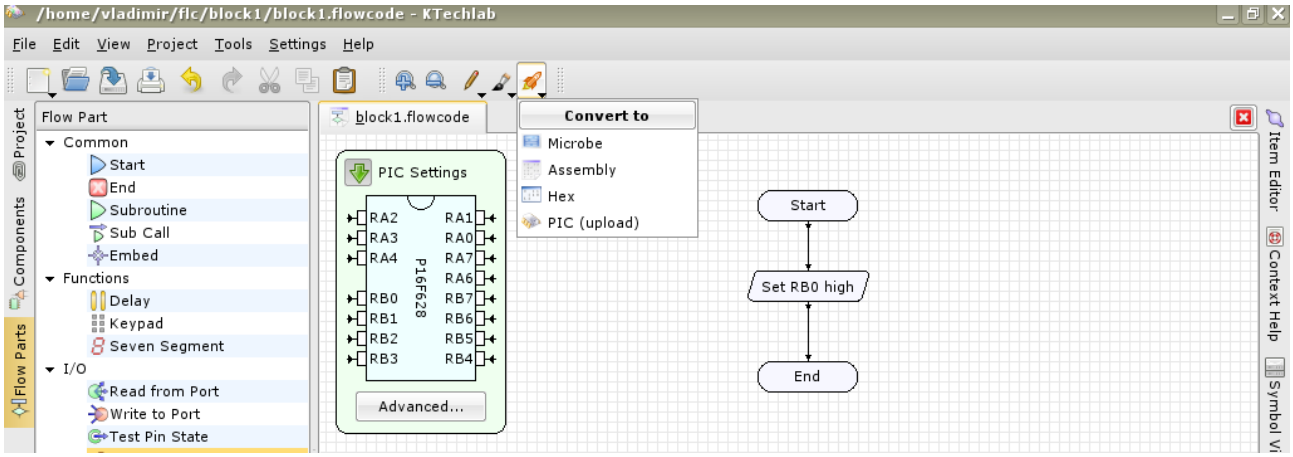


Рис. 6.3. Трансляция графического представления в языковую

Текст программы выглядит вполне понятно:

```
P16F628
TRISA = 127
TRISB = 254
PORTA = 0
```

```
PORTB = 0
PORTB.0 = high
end
```

Получим еще один простой блок.

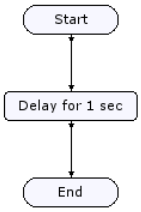


Рис. 6.4. Еще один простой графический блок программы

Который в microbe представлении имеет вид:

```
P16F628
TRISA = 127
PORTA = 0
PORTB = 0
delay 1000
end
```

Теперь можно объединить оба текстовых блока (удалив, конечно, лишнее):

```
P16F628
TRISA = 127
TRISB = 254
PORTA = 0
PORTB = 0
PORTB.0 = high
delay 1000
end
```

Можно и развить программу:

```
P16F628
TRISA = 127
TRISB = 254
PORTA = 0
PORTB = 0
PORTB.0 = high
delay 1000
PORTB.0 = low
delay 1000
end
```

Все эти переделки, как и с другими языками программирования, можно выполнить в текстовом редакторе. Программа зажжет светодиод на выходе RB0, даст ему секунду светиться, затем погасит его на секунду. Признаться, я все это проделал не выходя из текстового процессора, в котором сейчас работаю, но лучше для этой цели использовать простейший текстовый редактор, как gedit или kedit (мы находимся в Linux!).

Теперь создадим новый проект в KTechlab, создадим новый файл, а в диалоге выбора типа файла выберем...

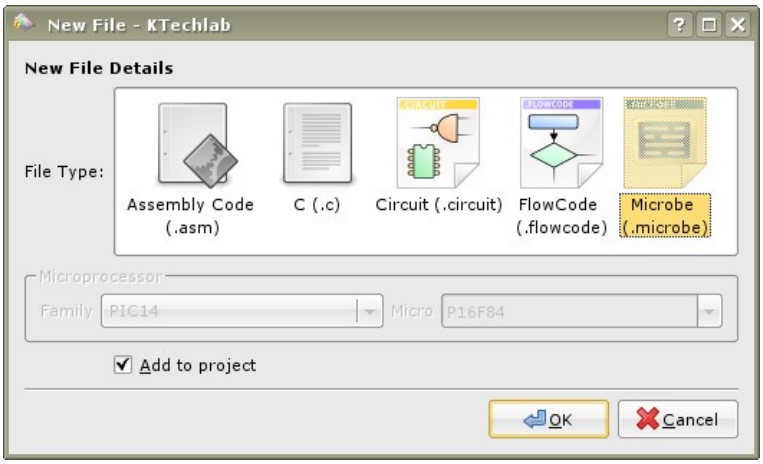


Рис. 6.5. Выбор типа файла Microbe

Хотя это и не совсем правильно, я перенесу прямым копированием из этого текста вышеприведенный фрагмент (формально полную программу) во вновь созданный файл в окне KTechlab, оттранслирую его на ассемблер и в hex-файл.

Или я немного забыл, как работать в KTechlab, или несколько отлична работа программы от варианта, установленного в Fedora 9, но для успешной трансляции я выбираю режим только отображения результата, а затем уже сохраняю полученный файл *Save As*...

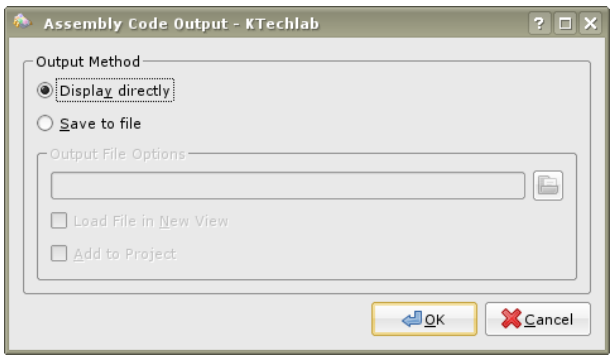


Рис. 6.6. Выбор режима трансляции файла

Без этого появляются проблемы с открытием файла. Но это детали. Полученный в результате hex-файл можно загрузить (непосредственно из KTechlab) в контроллер и проверить его работу на макетной плате. Кому-то, возможно, работать с программой в текстовом режиме удобнее. Преимущество тестового редактора в легком поиске всех переменных и подпрограмм, графическое представление лишено этого. Что ж, можно использовать microbe. Создав необходимый набор типовых блоков программы, можно писать достаточно сложные программы, а по мере работы, решая свои задачи, относящиеся к микроконтроллеру, язык microbe будет освоен.

Однако и приведение кода к синтаксису языка Си с последующей трансляцией на ассемблер возможно.

Создадим в KTechlab новый проект. Создадим новый файл, выбрав в качестве типа файла файл на языке Си.

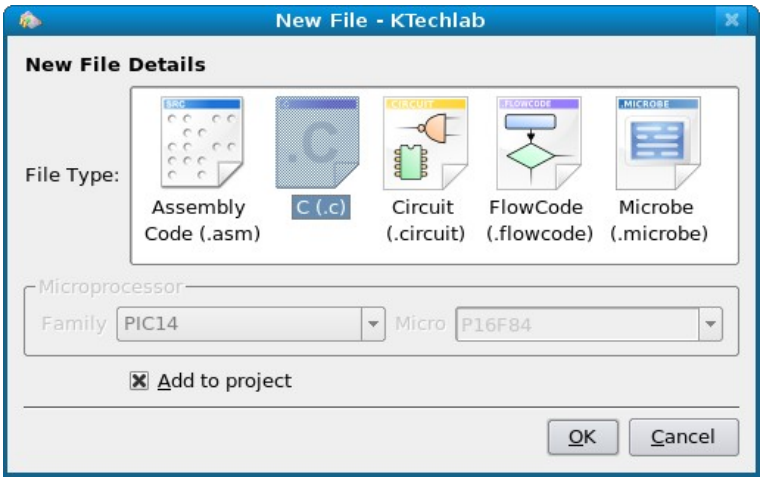


Рис. 6.7. Выбор типа файла для кодирования на Си

Перенесем в редактор текст, полученный на языке Microbe. Внесем исправления, очевидные на первый взгляд.

```
void main()
{
int i;
int x;

TRISB = 0x00;

while(x == 0)
{
      RB0 = 1;
      for (i=0;i=10;i++);
      RB0 = 0;
      for (i=0;i=10;i++);
}
}
```

Транслятор, который использует KTechlab — это свободно распространяемый компилятор SDCC. Трансляция программы проходит, когда она приобретает вид:

```
#include <pic16f628a.h>
void main()
{
int i;

TRISB = 0;
while(1)
{
      PORTB = 1;
      for (i=0;i<=10;i++);
      PORTB = 0;
      for (i=0;i<=10;i++);
}
}
```

Как в Windows MPLAB, так в Linux есть среда разработки программ на языке Си и ассемблере — Piklab. Если программа KTechlab позволяет работать на языках высокого уровня, позволяет перевести графический вид программы на язык высокого уровня, то работать на Си или ассемблере лучше в Piklab.

Интерфейс Piklab схож и с KTechlab, и с такой средой программирования на многих языках высокого уровня, как KDevelop. И, как обычно, создаем новый проект (основное меню, *Project-New Project...*).

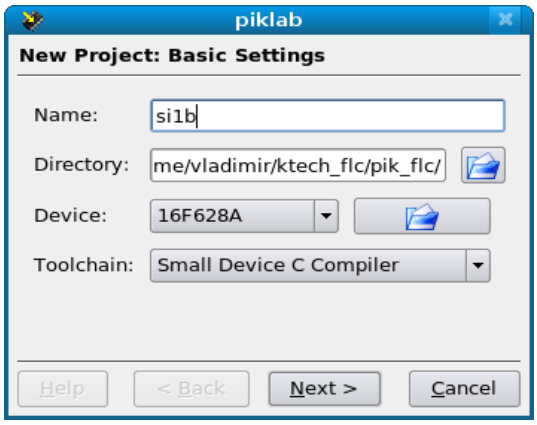


Рис. 6.8. Создание нового проекта в Piklab

Программа поддерживает работу со многими PIC-контроллерами (отображаемыми в окошке Device: при нажатии на кнопку с иконкой ▼) и длинным списком компиляторов, который можно обнаружить в окне Toolchain:, в данном случае, как и ранее SDCC.

Piklab при создании нового проекта в редакторе кода позволяет получить либо уже существующий файл, либо чистый лист, либо шаблон, который я выбрал.

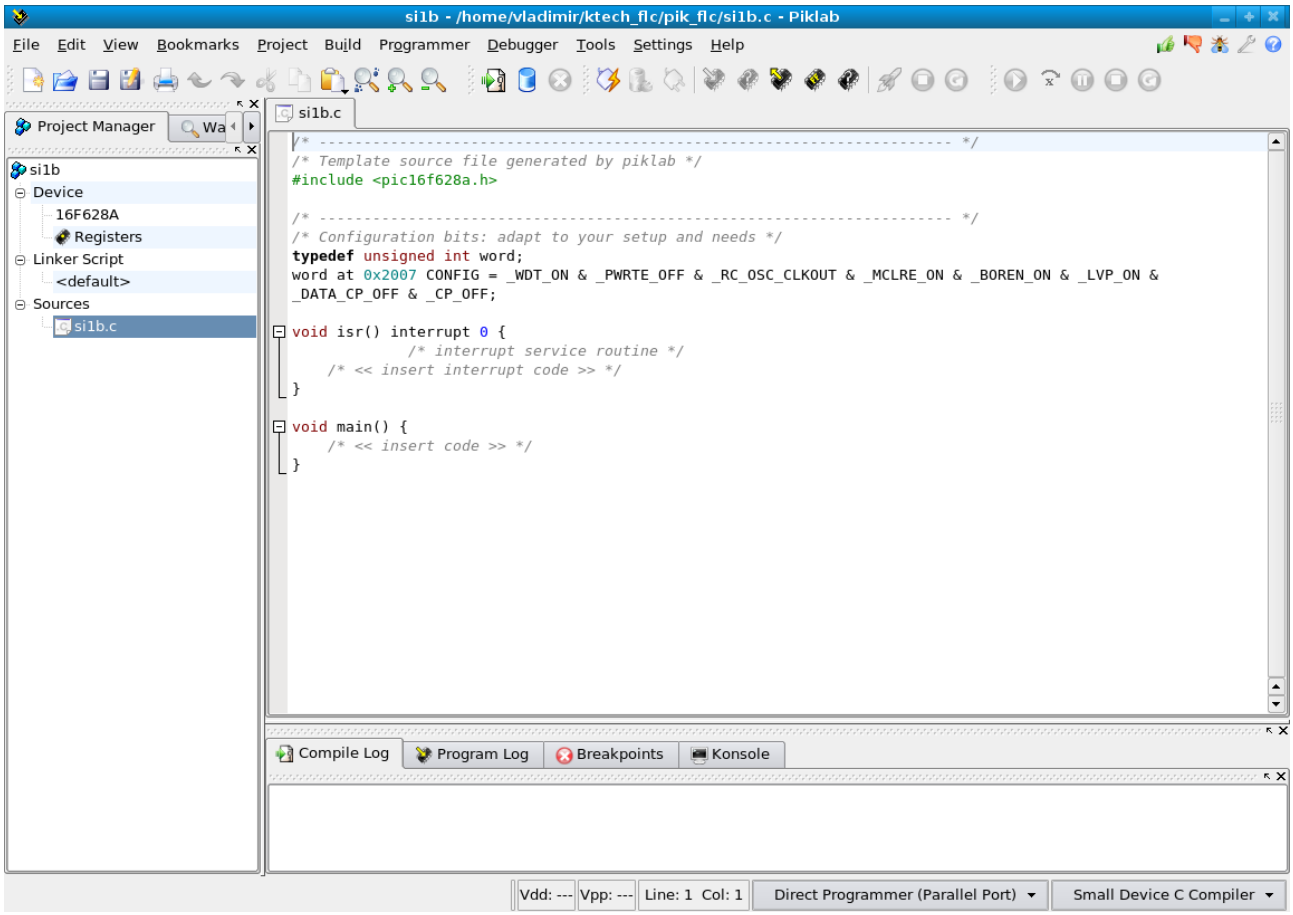


Рис. 6.9. Редактор кода вновь созданного проекта в Piklab

Содержательную часть (под грифом void main()) предыдущего фрагмента я переношу в редактор, непосредственно скопировав его с предыдущей страницы. В готовом шаблоне есть записанное слово конфигурации для контроллера. Но я предпочитаю другую конфигурацию. Чтобы изменить ее, достаточно воспользоваться пунктом *Config Generator... (генератор конфигурации)* раздела *Tools (инструменты)* основного меню.

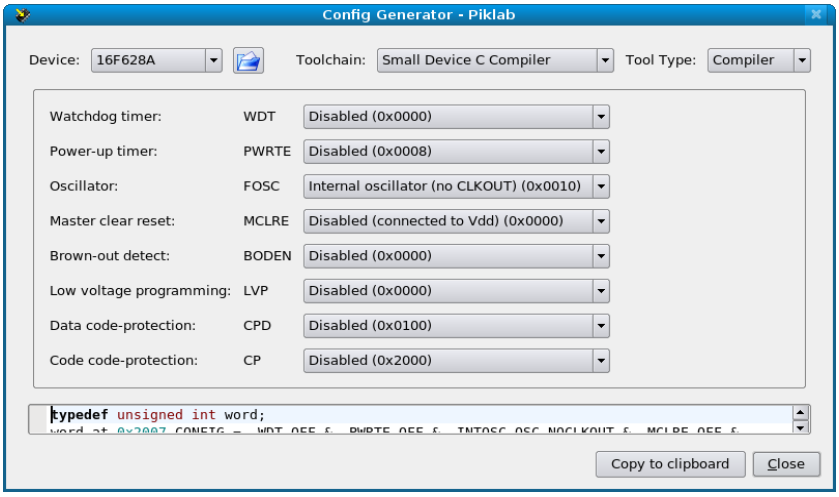


Рис. 6.10. Задание слова конфигурации микроконтроллера

После выбора всех бит конфигурации клавиша **Copy to clipboard** (копировать в буфер обмена) переносит нужный текст в буфер обмена, откуда, щелкнув правой клавишей мышки, можно выбрать пункт *Paste (вставить)* из выпадающего меню в окне редактора кода программы, чтобы добавить код. Предварительно можно выделить старый код в шаблоне, чтобы одним движением заменить его на нужный.

Попытка транслировать текст в ассемблер оказалась удачна, тогда как трансляция в hex-файл не получилась. Причину этого события удалось обнаружить, открыв раздел *Configure Toolchains... (Конфигурация инструментария)* в *Settings (Установки)* основного меню.

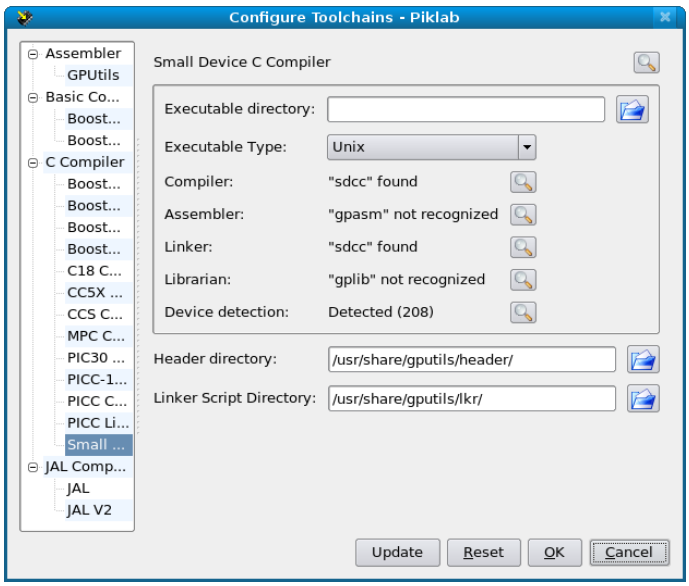


Рис. 6.11. Задание расположения трансляторов в Piklab

Если компилятор SDCC найден, то транслятор в ассемблерный код gpasm не распознается

(строка Assembler: «gpasm» not recognized). И не помогают попытки указать расположение файлов из набора gputils. Проверка работы Piklab в другом дистрибутиве, Ubuntu, показывает, что все в порядке. Все распознается. Правда, версия программы не столь нова. Но работает исправно.

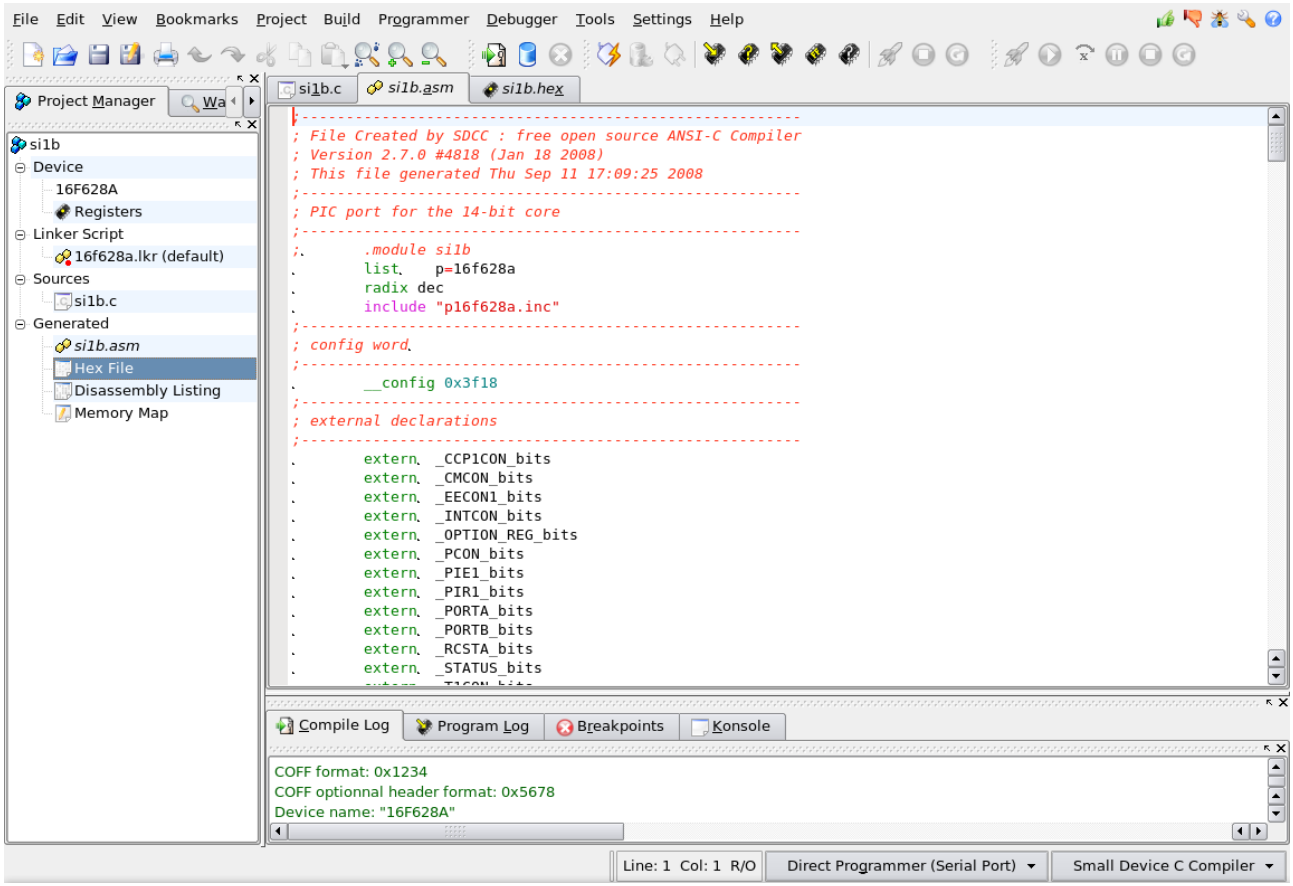


Рис. 6.12. Трансляция си-файла в программе Piklab

И так тоже бывает.

Несомненное достоинство сред разработки KTechlab и Piklab — использование полнофункционального и свободного компилятора Си SDCC (хотя есть версия и для Windows), возможность работать с простым в исполнении программатором JDM (схема для PIC16 ниже), не выходя из среды разработки, и, наконец, то, что эти программы распространяются свободно и бесплатно.

Очень часто можно слышать, что программы для Linux, в отличие от коммерческих, «сырые», малофункциональные и «безответственные». Мне приходилось работать с коммерческими и «ответственными» программами, в которых «глюки» не только имеют место, но спрятаны так глубоко, что для выявления ошибок, чтобы получить возможность передать их в службу поддержки, нужно проделать очень большой объем работы. Но это так, к слову.

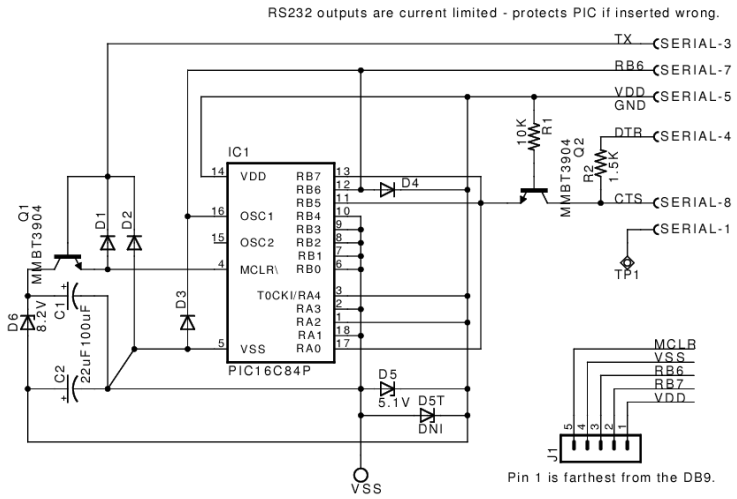


Рис. 6.13. Программатор JDM для совместной работы с Piklab и KTechlab

Собрать такой программатор не представляет труда даже для начинающего. Он получается небольшим и достаточно удобным в работе, не требует дефицитных деталей. Если стабилитрон D6 заменить аналогичным на напряжение 5.1 В и добавить два светодиода АЛ307А в прямом включении (или один светодиод и стабилитрон с соответствующим напряжением стабилизации), то процесс обращения к программатору можно контролировать визуально.

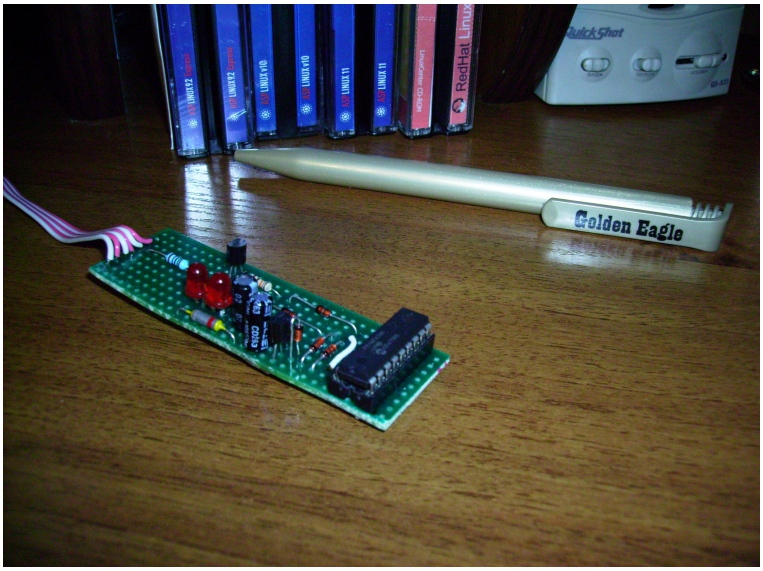


Рис. 6.14. Вид собранного программатора

Подключение и настройка программатора, например, в KTechlab потребует добавление программного модуля, но, единожды настроенный, программатор работает вполне убедительно.

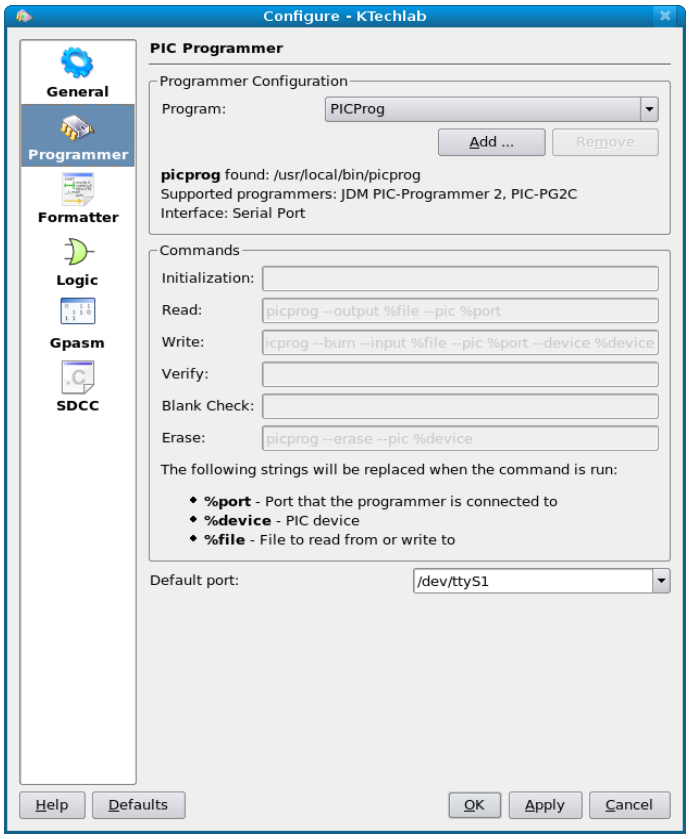


Рис. 6.15. Настройка программатора в KTechlab

Что дальше?

Ни одна профессия не позволяет достичь вершин с тем, чтобы с этих вершин, угнездившись в достигнутом, равнодушно взирать на окружающий мир. Так не бывает. Приходится каждый день, каждый час решать все новые и новые задачи, перетряхивать ворох документации и искать информацию, которая помогла бы реализовать новые идеи, «освежить» старые решения, пополнить запас знаний.

Радиолюбители, народ любознательный и энергичный, порой, могут дать «сто очков вперед» любому профессионалу в части любознательности. Сделав первые шаги в освоении микроконтроллеров, они обязательно захотят знать, а что дальше? У них уже есть идеи, как применить контроллер, но это будни. Это текущие планы. А, что еще?

Очень интересным в этом отношении мне представляется использование встроенного во многие микроконтроллеры модуля сетевой работы USART. Где можно применить эту функцию контроллера?

Положим, что контроллер выполняет некую программу, связанную с обработкой состояния датчика. В этом случае, как правило, датчик и контроллер могут быть разнесены на некоторое расстояние. Или, например, пульт управления и исполняющее устройство, они тоже могут находится в разных местах. И пусть пульт управления имеет всего две-три кнопки, а между пультом и управляющим модулем расстояние в несколько метров, для связи между ними удобно использовать встроенный приемо-передатчик USART. В качестве сетевого интерфейса я отдал бы предпочтение RS485: всего два провода, хорошая защита от внешнего электромагнитного воздействия, малые создаваемые помехи, допустимые большие расстояния, для одного модуля всего одна микросхема интерфейса RS485. Не самое сложное и не самое дорогое решение.

Посмотрим, как можно использовать USART для задачи соединения модуля (с одной кнопкой) управления с исполняющим модулем (один светодиод).

Программа для первого модуля может выглядеть (я использую FlowCode) следующим образом:

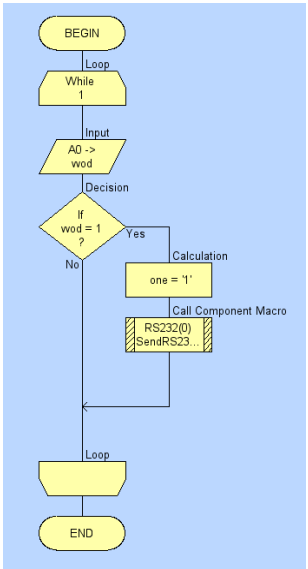


Рис. 7.1. Программа для модуля управления

Кроме уже привычных программных компонент, после добавления дополнительного

элемента RS232 (с инструментальной панели дополнительных элементов) я использую программный элемент *Component Macro*, появляющийся как вызов подпрограммы, обрабатывающей обращения к встроенному USART. Сама подпрограмма уже написана производителем FlowCode, остается только настройка.

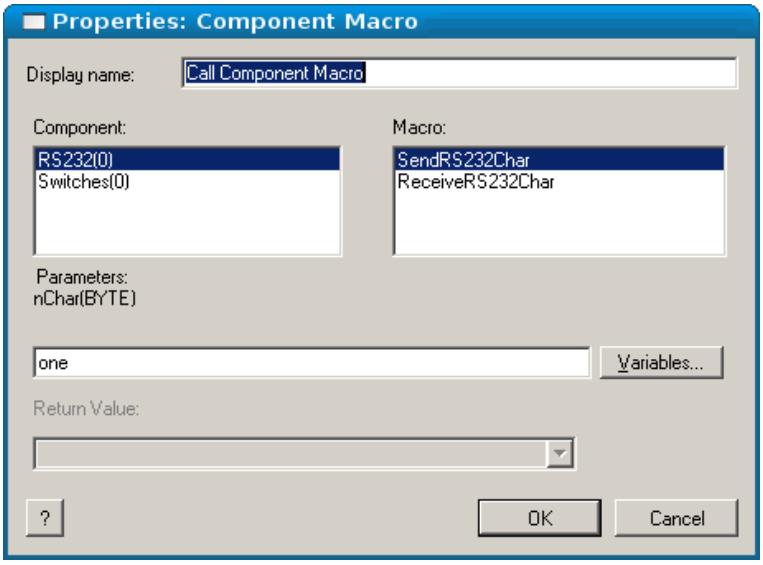


Рис. 7.2. Диалоговое окно настройки компонента RS232

Выбрав в окне Component: нужный мне дополнительный элемент RS232(0), выбрав в окне Macro: нужный мне вид работы SendRS232Char (отправка символа по RS232), я создаю новую переменную «one» (типа Byte), которой в ветке программы (до вызова подпрограммы отправки символа) с помощью программного компонента *Calculate* присваиваю значение '1'.

На этом написание программы для первого модуля я заканчиваю. Можно проверить работу программы.

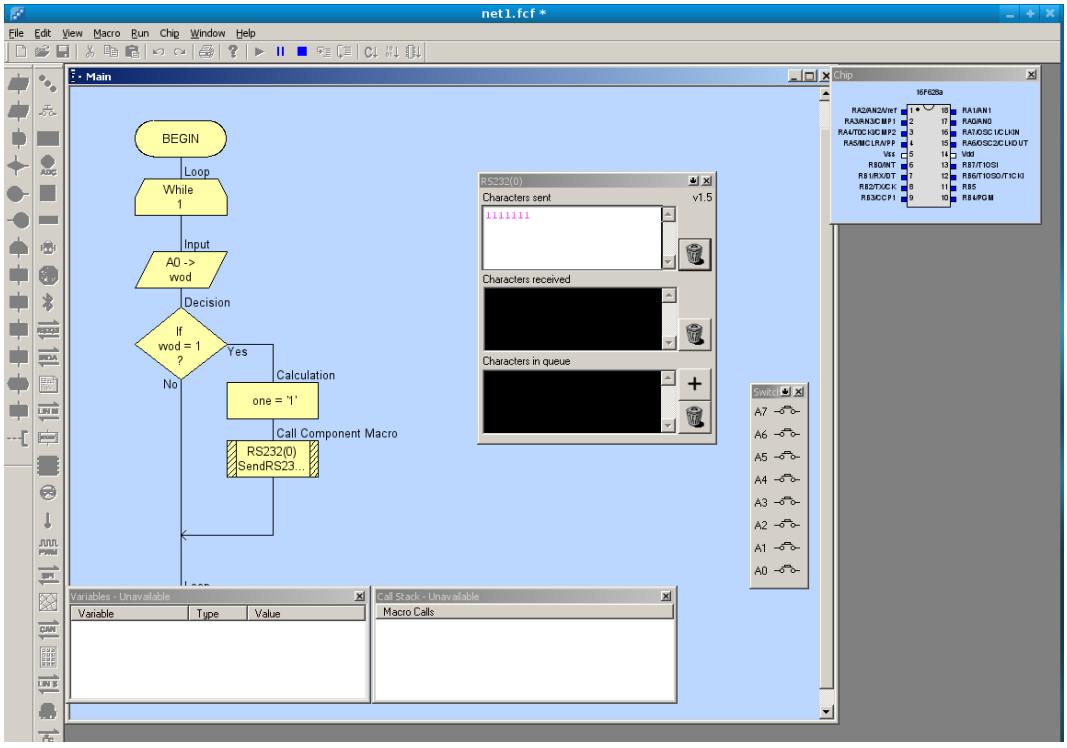


Рис. 7.3. Проверка программы первого модуля

Нажимая кнопку A0 на элементе Switches(0), я могу наблюдать, как в окне Characters sent

элемента RS232(0) появляются единицы (я не убирал «дребезг», и каждое нажатие может отправлять несколько единиц). Окно наблюдения за этим процессом в программе FlowCode черное, а символы зеленые, но они плохо видны в тексте, поэтому в графическом редакторе GIMP (перед тем, как вставить рисунок в текст) я инвертирую цвета.

После задания слова конфигурации (и выбора модели контроллера, напомню, основное меню *Chip*) я транслирую проект в hex-файл.

Теперь создадим еще одну программу. Она должна принимать символ '1' по USART и включать светодиод на RB0.

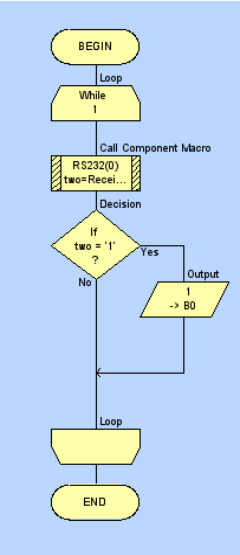


Рис. 7.4. Программа второго модуля, приемного

Здесь я тоже использую *Component Macro* для добавочного элемента RS232.

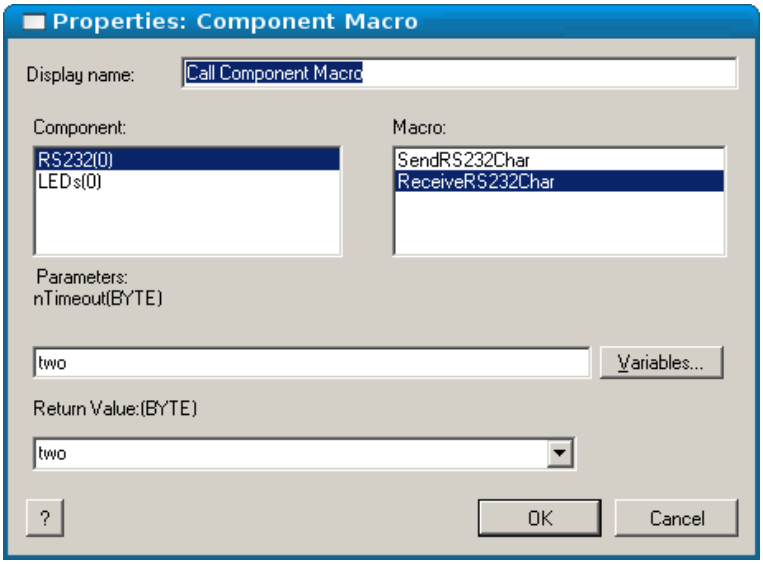


Рис. 7.5. Диалоговое окно настройки компонента RS232

Запустив программу, в компоненте RS232(0), в окне Characters in queue с помощью клавиши с иконкой + можно открыть окно ввода необходимого символа, как если бы мы этот символ отправили из первого модуля.

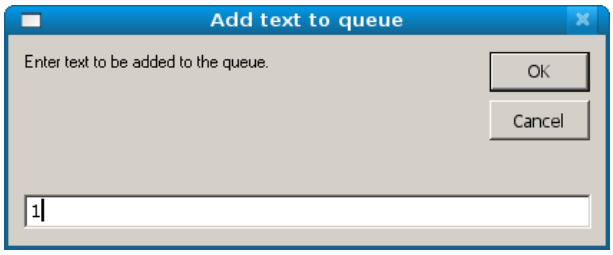


Рис. 7.6. Окно ввода символа, который получит второй модуль

Если теперь нажать клавишу **ОК**, то программа должна зажечь светодиод на выводе RB0.

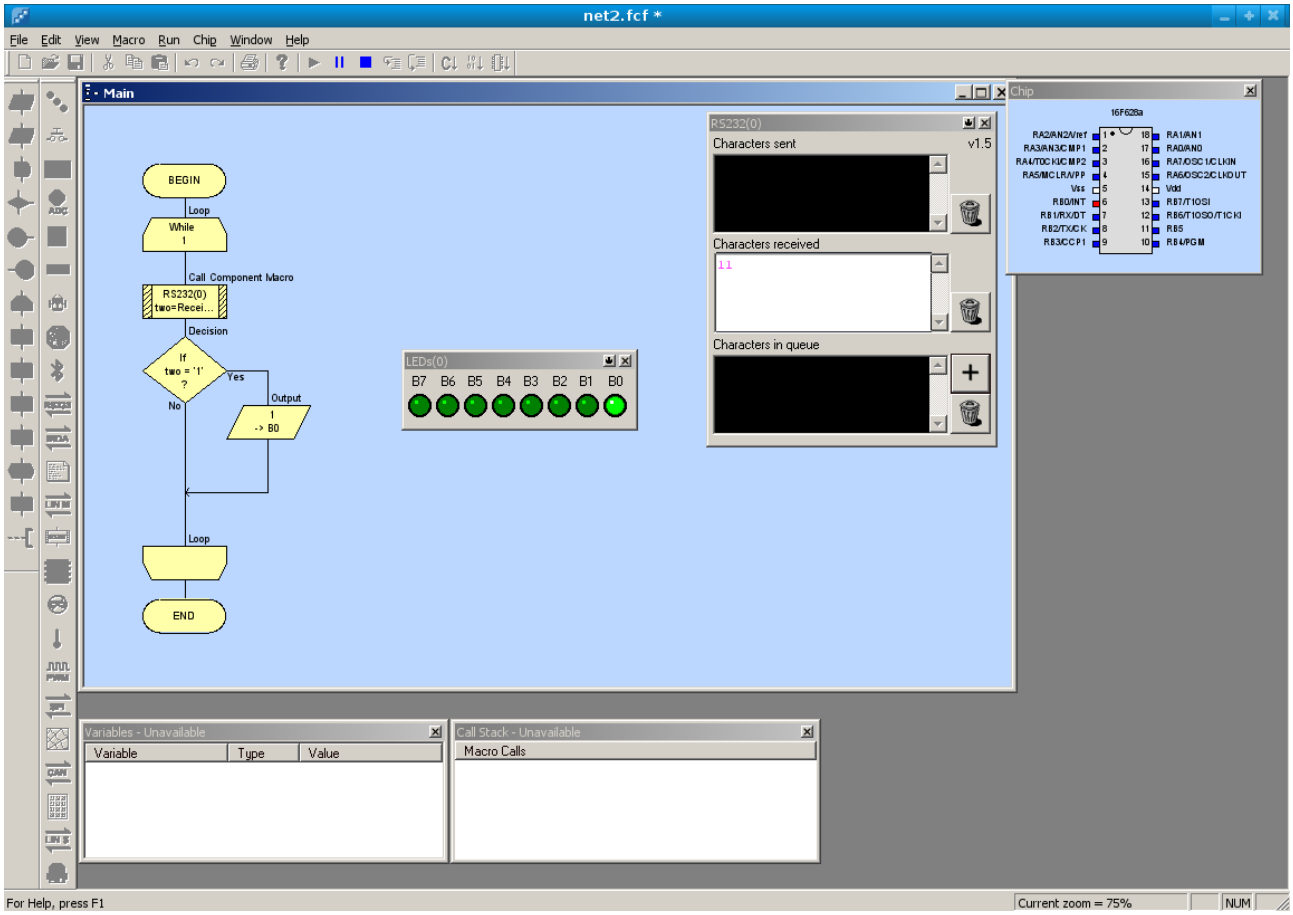


Рис. 7.7. Проверка второго модуля

Окошко Characters received я опять «подправил» в Gimp, чтобы можно было видеть полученный символ. Для трансляции в hex-файл, правда, приходится переходить в Windows (программа для этой ОС). Многие программы транслируются в Linux без проблем, но, похоже, не все. И, что дальше? Можно, конечно, «прошить» программы в контроллеры и проверить, что называется «живьем», но...

Есть такая программа (тоже для Windows), которая называется Proteus. Чем она мне сейчас интересна, так это тем, что в ней можно симулировать работу двух микроконтроллеров. Что я и хочу проделать.

Добавив в редакторе Proteus ISIS два микроконтроллера PIC16F628A и все необходимые внешние элементы, я настраиваю контроллеры:

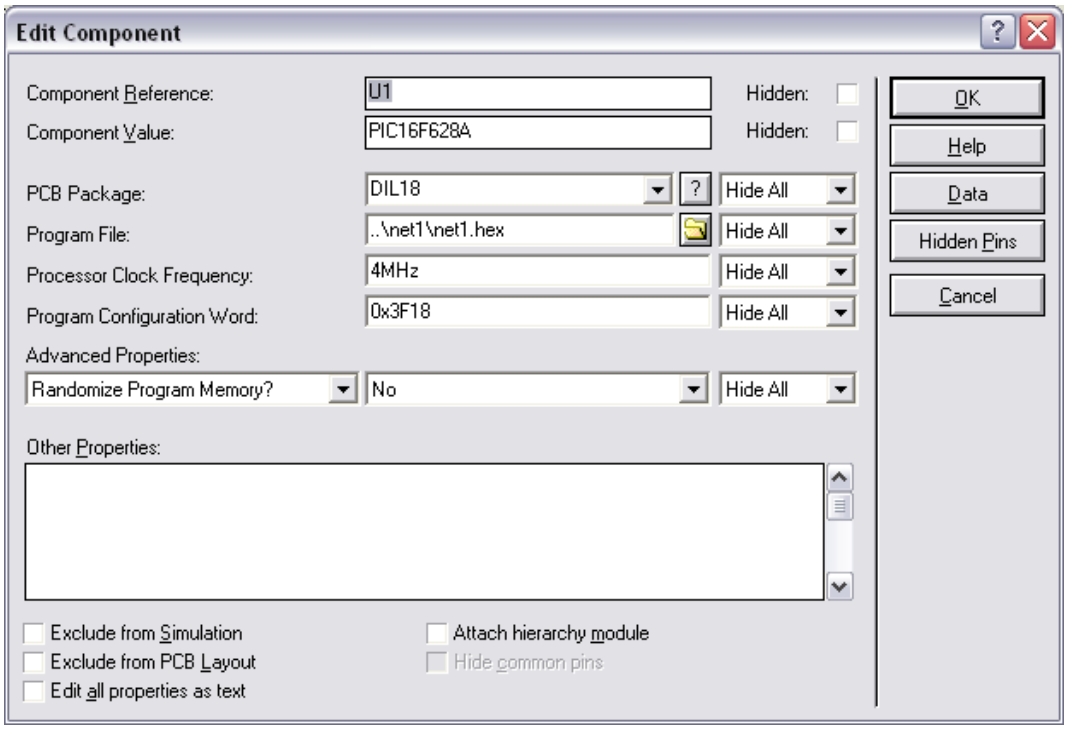


Рис. 7.8. Настройка первого контроллера

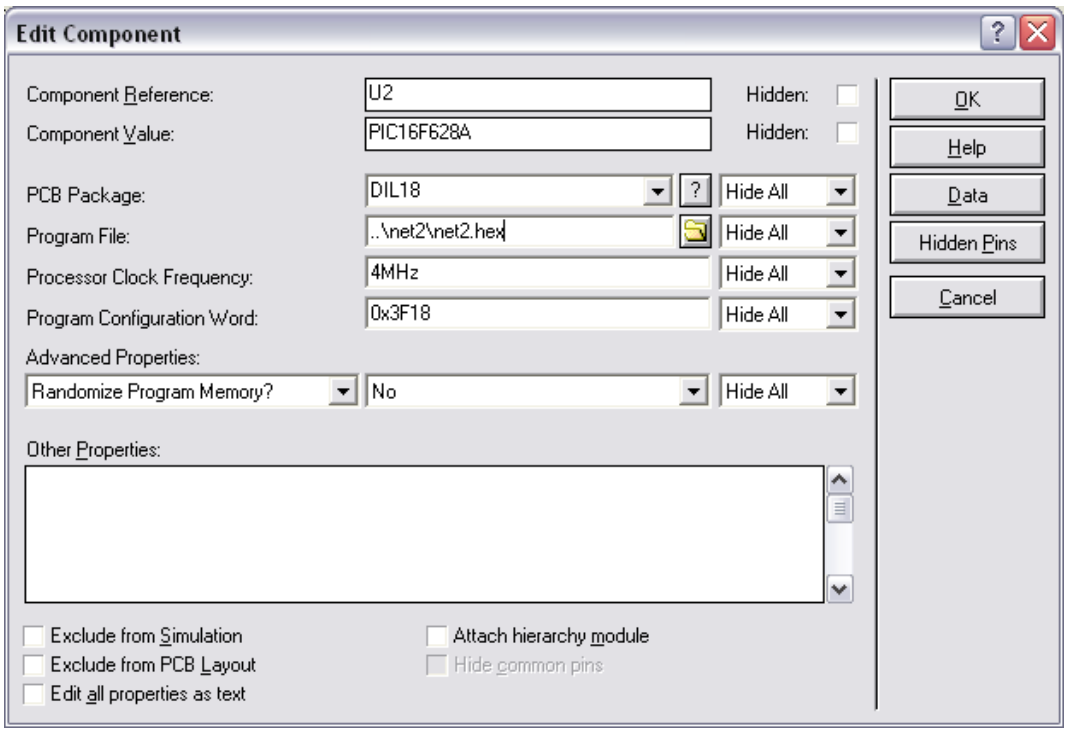


Рис. 7.9. Настройка второго контроллера

Теперь, нажимая кнопку в собранной схеме, я вижу, что все действительно должно работать.

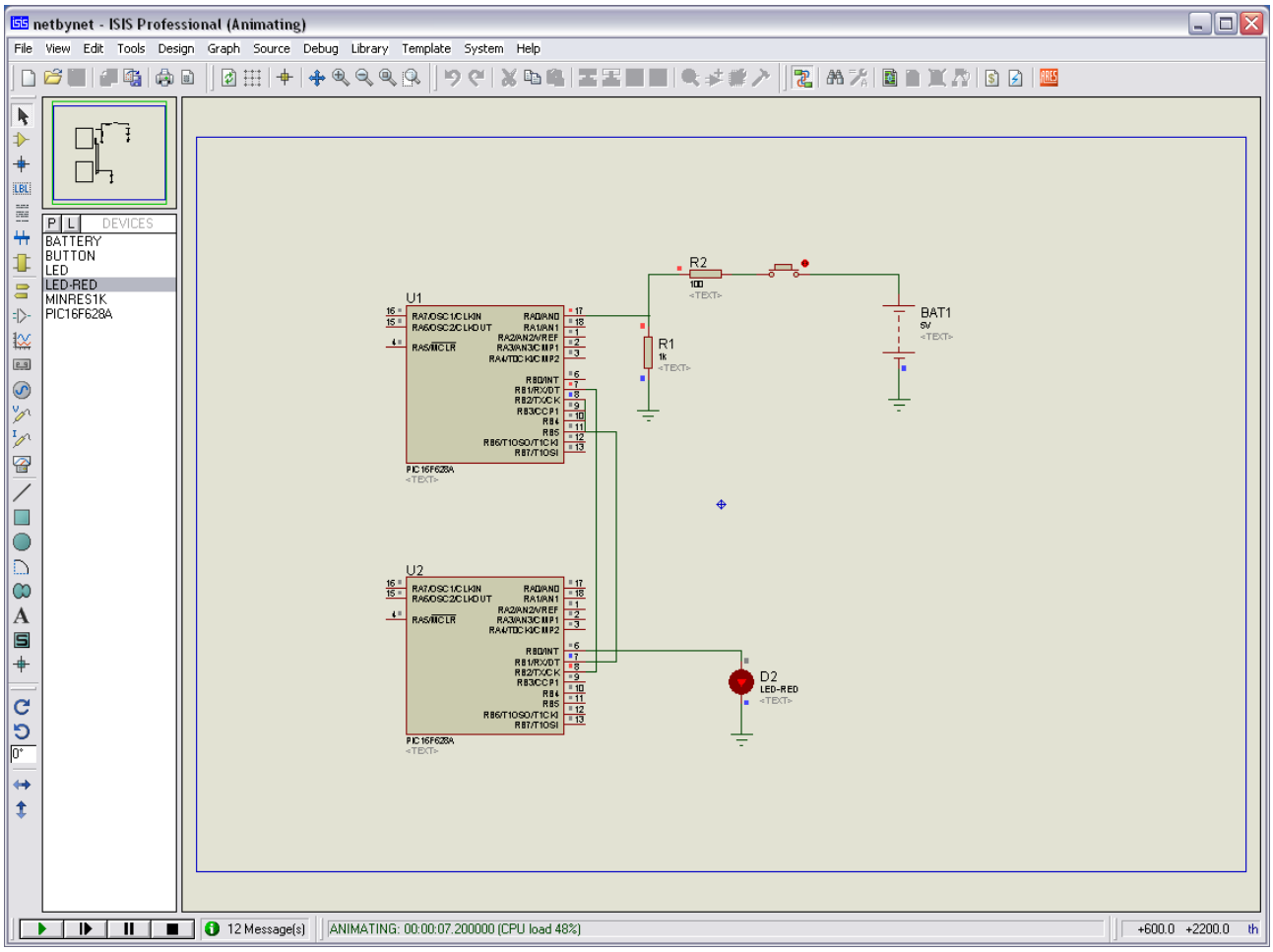


Рис. 7.10. Проверка работы двух контроллеров в Proteus

Так «беседуют» два микроконтроллера.

Не будем им мешать.