Отладка в gpsim

Прежде чем перейти к программе отладки, у меня есть желание несколько упростить запуск отладочной программы и необходимость написать подходящий для работы с отладчиком код программы для микроконтроллера PIC16F628A.

Особых трудностей в запуске gpsim из консоли нет, но при многократном повторении это вызывает некоторое раздражение. Создание кнопки запуска (или ярлычка программы) в Linux не вызывает особенных затруднений – достаточно щелкнуть правой клавишей мышки по рабочему столу и в выпадающем меню выбрать пункт «Создать кнопку запуска...». Как мне помнится, некогда была возможность и включить запуск программы в терминале. Сейчас такой возможности я не нахожу. Думаю, причина в том, что ранее я в основном работал в графической среде KDE, а не Gnome, как сейчас. Поскольку я использую часть программ, предназначенных для работы с KDE, базовые пакеты этой графической среды установлены на компьютере, и я могу перейти в KDE, завершив сеанс, и при входе в новый сеанс сменить текущую графическую среду на KDE. Что я и делаю.

В KDE, щелкнув правой клавишей мышки по рабочему столу, я вызываю выпадающее меню.

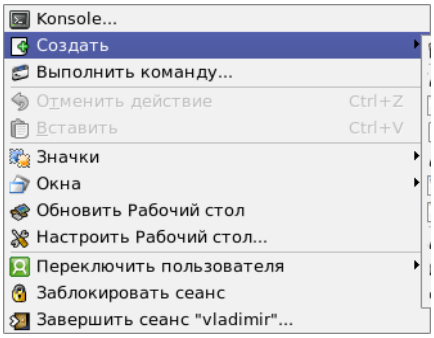


Рис. 2.24 Всплывающее меню в KDE

И при создании кнопки запуска приложения действительно нахожу необходимую мне опцию в разделе дополнительных параметров.

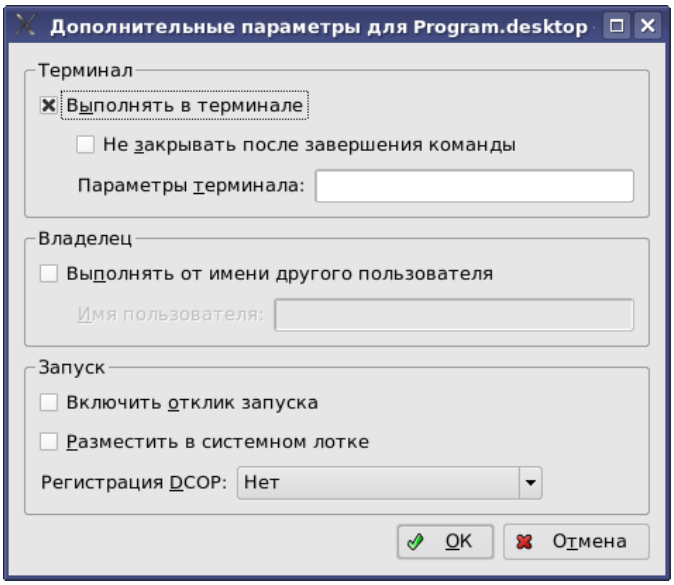


Рис. 2.25. Диалоговое окно создания кнопки запуска приложения в терминале

Теперь по возвращении в Gnome я имею кнопку запуска, которая делает все необходимое, то есть, запускает gpsim в терминале.

Для подходящего кода программы я воспользуюсь тем кодом, что некогда был написан в среде программирования MPLAB. Я скопирую текст программы, удалю все лишнее, и вставлю в новый проект, который создам в Piklab. На этой стадии работы я хочу, чтобы при получении команды – простой символ «1», отправляемый из графического интерфейса, написанного в среде Gambas – моя программа микроконтроллера зажигала светодиод, который будет на макетной плате присоединен к одному из выводов порта. А при получении второй команды, тоже в виде простого символа «0», микроконтроллер гасил этот светодиод. Проверочный код программы выглядит следующим образом:

Файл заголовка counter_start.h:

```
#define bitset(var,bitno) ((var) |= 1 << (bitno)) // Установка бит порта
#define bitclr(var,bitno) ((var) &= ~(1 << (bitno))) // Сброс бит порта
unsigned char getch(void); // Прием символа
void     init_comms(); // Инициализация коммуникации
```

Основной файл программы:

```
/* ------------------------------------------------------------------ */
```

```
/* Template source file generated by piklab */
#include <pic16f628a.h>

/* ------------------------------------------------------------------- */
/* Configuration bits: adapt to your setup and needs */
typedef unsigned int word;
word at 0x2007 CONFIG = _WDT_ON & _PWRTE_OFF & _RC_OSC_CLKOUT & _MCLRE_ON &
_BOREN_ON & _LVP_ON & _DATA_CP_OFF & _CP_OFF;

#include <stdio.h>
#include "counter_start.h"

unsigned char input;  // Для считывания приемного регистра
unsigned char COMMAND;// Символ команды

unsigned char getch() // Получение байта
{
        while(!RCIF)   // Устанавливается, когда регистр не пуст
                continue;
        return RCREG;
}

void cmd() // Получение и выполнение команды
{
        COMMAND = input;
        switch (COMMAND) {
                case '1': bitset (PORTA, 0);
                break;
                case '0': bitclr (PORTA, 0);
                break;
                }
}

void init_comms()      // Инициализация модуля
{
        PORTA = 0x0;   // Настройка портов A и B
        CMCON = 0x7;
        TRISA = 0x0;
        TRISB = 0xFE;
        RCSTA = 0x90;  // Настройка приемника
        TXSTA = 0x6;   // Настройка передатчика
        SPBRG = 0x68;  // Настройка режима приема-передачи
        bitclr (PORTB, 0);     // Выключаем драйвер RS485 на передачу
}

void main() {
        init_comms(); // Инициализация модуля

start:  CREN =1; // Начинаем работать
        input = getch();
        cmd();
        goto start;
}
```

Разобраться в коде программы, которую когда-то писал ты сам, не проще, чем в чужой программе. Читать описание долго, а комментарии достаточно короткие, чтобы были понятны пока работаешь над программой (да и то не всегда), но не дающие понимания через несколько дней после завершения работы над кодом программы. Словом, я постарался выбросить все, что пока не относится к делу (каким это представляется в данный момент), включая часть шаблона для организации прерываний. Что из этого получится, не знаю. Пока я только привел код к виду, когда трансляция проходит без ошибок. А хотелось бы мне, чтобы получая от COM-порта команду «1» на вход RB1 (первый вывод порта B), микроконтроллер зажигал светодиод на RA0 (нулевой вывод порта A), а по команде «0», гасил его.

После трансляции программа Piklab формирует нужный мне для отладки файл, поэтому я готов приступить к описанию работы с отладочной программой gpsim.

После запуска программы (удачно созданной кнопочкой запуска) мы получаем следующий вид программы:

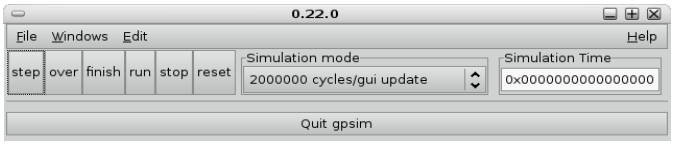


Рис. 2.26. Вид отладчика gpsim после первого запуска

Основное меню содержит три пункта, первый из которых File позволяет мне только открыть файл или выйти из программы. Я открываю файл counter_start.cod. И... ничего не происходит. Все-таки это не редактор текста. Но если обратиться к следующему пункту основного меню Windows, то открывается доступ к отображению различных рабочих окон отладчика.

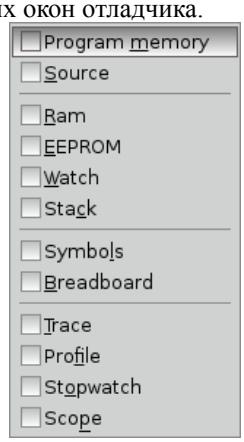


Рис. 2.27. Подменю раздела Windows основного меню отладчика

Для начала активизируем окно Source, что должно открыть исходный текст программы. И этот текст открывается. Отладочный текст имеет вид смешанного текста: исходного на языке Си, и оттранслированного на ассемблере. Комментарии видны, что позволит мне, надеюсь, сориентироваться в дремучих зарослях ошибок.

Если теперь запустить программу отладки с помощью клавиши Run на основной панели, то анимация быстро замирает. Только счетчик времени симуляции Simulation Time продолжает свой счет. Так и должно быть, поскольку вслед за инициализацией микроконтроллер переходит к ожиданию ввода команды через COM-порт компьютера.

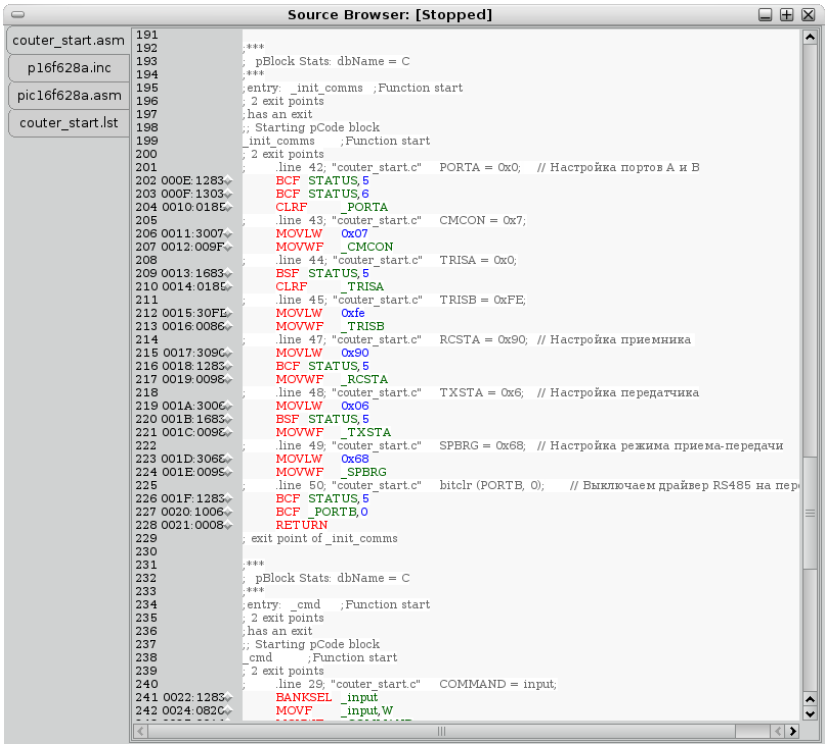


Рис. 2.28. Исходный текст программы, с которым предстоит работа

Мне хватает сообразительности (или я где-то это вычитал) открыть в подменю раздела Windows еще одно отладочное окно – Breadboard (макетная плата). На плате, как и положено, есть микроконтроллер. Есть и ряд клавиш, назначение которых мне пока не ясно, но мне хотелось бы поэкспериментировать с этими клавишами. Дело в

том, что работая с программой MPLAB, я использовал инструмент отладки, который назывался **usart.** Или эта возможность задавалась в настройках программы? Сейчас не помню, но хочу отыскать нечто похожее в gpsim. Правильное решение – чтение документации, и неправильное – щелканье по всем клавишам, в конце концов совместными усилиями приносят плоды. И здесь, видимо, уместно еще раз отметить, что при компиляции исходного кода в Ubuntu, я получил дополнительные файлы из исходного текста (или текстов) программы, а в Fedora Core потребовалось добавить пакет gpsim-devel. При первом открывании макетной платы она выглядит так:

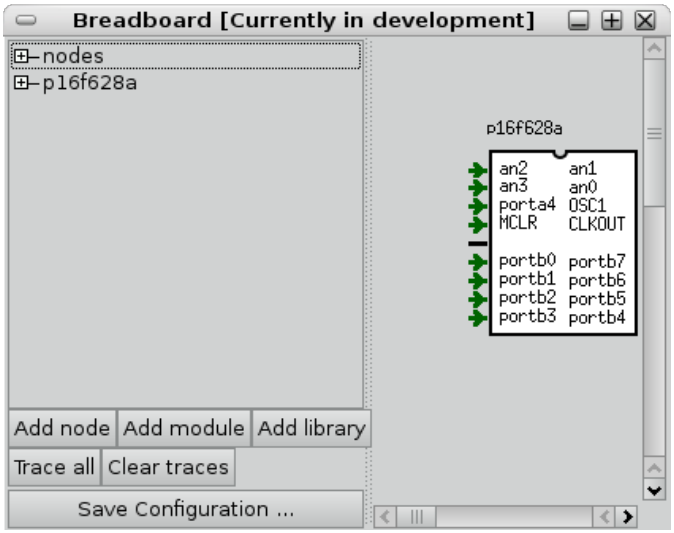


Рис. 2.29. Окно макетной платы отладчика gpsim

Теперь следует воспользоваться клавишей **Add library**, чтобы получить доступ к дополнительным модулям. И я не сразу понимаю, что именно следует ввести в окошке диалога, пока не повторяю то, что собственно и написано.

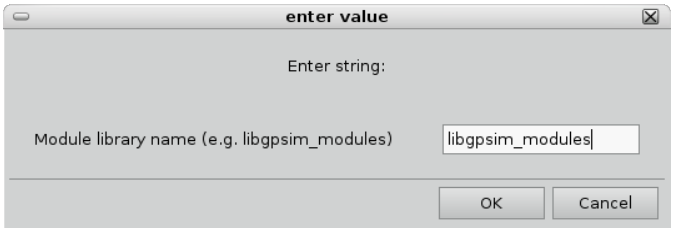


Рис. 2.30. Диалоговое окно добавления библиотеки

Теперь можно воспользоваться клавишей **Add module** для поиска модуля usart.

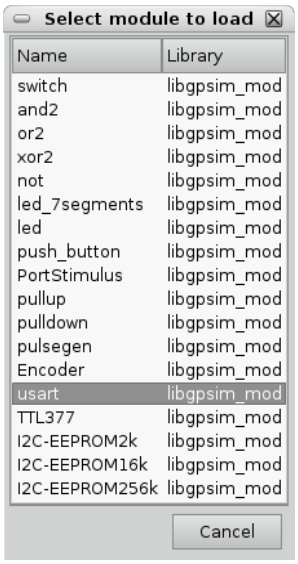


Рис. 2.31. Диалоговое окно добавления модулей

Двойной щелчок по этому модулю добавляет его на макетную плату и открывает еще одно (консольное) окно с надписью USART. Небольшой опыт работы с программами симуляции подсказывает мне, что новое приобретение в виде модуля usart необходимо подключить к микроконтроллеру. И что для этих целей можно воспользоваться узлами схемы (node). Для добавления узла на макетной плате есть клавиша **Add node**. Я добавлю два узла, которые назову rb1 и rb2. Диалоговое окно, появляющееся после нажатия клавиши добавления узла, похоже на окно диалога добавления библиотеки и требует введения имени узла. Можно проверить, что после нажатия на клавишу **ОК**, раздел узлов пополняется вновь созданными.

Теперь я хочу соединить узлы и выводы микроконтроллера. Для этого, открыв содержание микроконтроллера щелчком по квадратику со значком плюс слева от имени контроллера на макетной плате, я выбираю portb1, по которому тоже дважды щелкаю. Это действие вызывает появление новой клавиши в нижней части макетной платы с названием **Connect stimulus to node**. Щелчок по этой клавише открывает список всех созданных узлов, и я выбираю узел rb1, по которому дважды щелкаю с помощью клавиши мышки. После этого надпись выше клавиши, которая прежде гласила, что вывод не подключен, меняется на «Connected to node rb1». Те же операции я повторяю с выводом portb2 микроконтроллера, который подключаю к

узлу rb2.

После подключения контроллера я закрываю дерево его выводов щелчком по квадратику с его названием и значком минус внутри и открываю дерево выводов usart. Меня интересуют выводы usart.RXPIN и usart.TXPIN, которые я подключаю соответственно к узлам rb2 и rb1, точно так же, как сделал это с выводами контроллера (заметьте, что RXPIN оказывает соединен с portb2, а TXPIN с portb1).

В этом месте я споткнулся (а кто не спотыкается?). Открывающаяся при добавлении модуля usart консоль USART в моем представлении должна принимать ввод с клавиатуры и отображать то, что отправляет микроконтроллер. До отправки еще далеко, а ввести что-нибудь с клавиатуры я пытался. Если вернуться к коду моей проверочной программы и ее описанию, то можно заметить, что я отправляю символ «1» для зажигания светодиода на выводе RA0, и символ «0» для гашения светодиода. Это теперь так, а первоначально я собирался использовать символы «1» и «2». Почему я все переделал? Я сейчас долго и нудно постараюсь все объяснить. Например, так: НЕ ПОЛУЧИЛОСЬ.

Ввод в консоли команд не заработал. Тогда я решил использовать установку атрибутов usart. Чтобы это сделать достаточно на макетной плате в перечне элементов дважды щелкнуть по нужному элементу. Ниже открывается окно с перечнем атрибутов элементов. И теперь можно двойным щелчком клавиши мышки по нужному атрибуту вывести его в окошко пониже, где можно изменить значение атрибута.

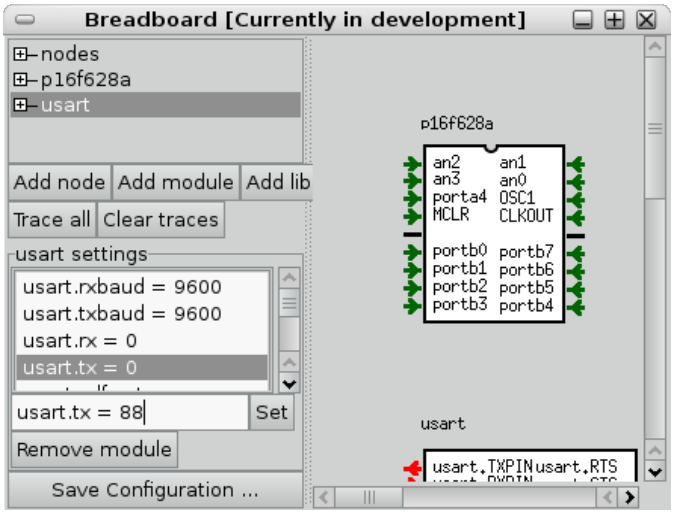


Рис. 2.32. Изменение значение атрибута элемента макетной платы

Я использовал атрибут usart.tx, которому задал значение «88». А наблюдать, как меняется вводимый символ можно с помощью окна наблюдения Watch Viewer. Для добавления в окно переменной, я наблюдаю за переменной COMMAND, которая получает значение вводимого символа, для ее добавления к наблюдению следует выделить эту переменную в тексте (окно Source), щелкнуть правой клавишей мышки по выделению, и выпадающем меню выбрать пункт Add to watch. Еще хорошо бы установить точку останова в том месте, где ей передается значение из ввода, просто щелкнуть в области значений строк левой клавишей мышки. Повторный щелчок снимает точку останова (breakpoint). Можно в тексте нажать правую клавишу мышки и из выпадающего меню выбрать Breakpoint here.

После запуска программы отладки, установив значение атрибута usart.tx, я стал вводить разные варианты для символа «1». И с двойными кавычками, и с одинарными, и без кавычек, и в виде шестнадцатеричного кода символа ASCII, и десятичного – все без положительных результатов, если не считать того, что вводимые десятичные значения порядка 100 меняли переменную COMMAND. Так я пришел к двум числам 88 и 89, которые давали значение символа «0» и символа «1», соответственно. Именно по этой причине я переделал код программы, что было совсем не сложно. Именно так я получил некоторую видимость работы программы, поскольку стал краснеть (на Breadboard) при вводе числа 89 нулевой вывод порта A, как и было задумано.

Я перебрал множество чисел, пытаясь выявить связь со значениями кодов ASCII, но понял, что это будет очень затяжное занятие и отказался от этого.

Затем, возвращаясь временами к USART консоли, и пытаясь ввести с клавиатуры символ через консоль, я случайно зацепил какую-то букву. Теперь программа отладки упорно стала «виснуть». Мне это не понравилось и я, что значит бессистемная работа, и я стал вновь пытаться ввести что-то полезное в окно консоли USART.

При этих попытках, то есть, активизировать окно и нажать на клавиатуре цифру 1 или 2, я не вижу результата, а нажав буквенную клавишу, я застреваю в функции getch(), выйти из которой мне никак не удается. Последнюю неприятность я пытаюсь исправить добавив в оператор переключения строку выхода, если ни один из случаев команды не совпадает. Теперь оператор выглядит так:

```
switch (COMMAND) {
        case '1': bitset (PORTA, 0);
        break;
        case '2': bitclr (PORTA, 0);
        break;
        default: break;
}
```

Мне кажется, что оснований где-либо застревать быть не должно. И действительно, это похоже, работает, поскольку при активизации окна консоли USART и вводе с клавиатуры любой буквы я останавливаюсь, но с помощью щелчков по клавишам **over** и **run** на основной панели отладчика мне удается оживить отладку

и переместиться к ожиданию очередного ввода.

Однако цифры не работают. В настоящий момент (да и в будущем) для меня не важно, как я обозначу команду. Мне ничто не мешает заменить символ «0» на «a», а символ «1» на «b» в тексте программы, откомпилировать ее заново и попробовать еще раз запустить отладку. Если консоль не работает с цифрами, то с буквами она работает.

Заменим цифры на буквы, и ... И опять ничего. Но! Но теперь я проверяю, какой код я получаю в переменной COMMAND, когда нажимаю букву «a» и когда нажимаю букву «b» (латинские). Код странный: a = 193, b = 198 (десятичные). Но именно эти значения я задаю в переключателе switch кода основной программы:

```
switch (COMMAND) {
       case 193: bitset (PORTA, 0);
       break;
       case 198: bitclr (PORTA, 0);
       break;
       default: break;
       }
}
```

Теперь отладка проходит, а когда я нажимаю клавишу «a» на клавиатуре вывод RA0, долженствующий изображать светодиод, краснеет.

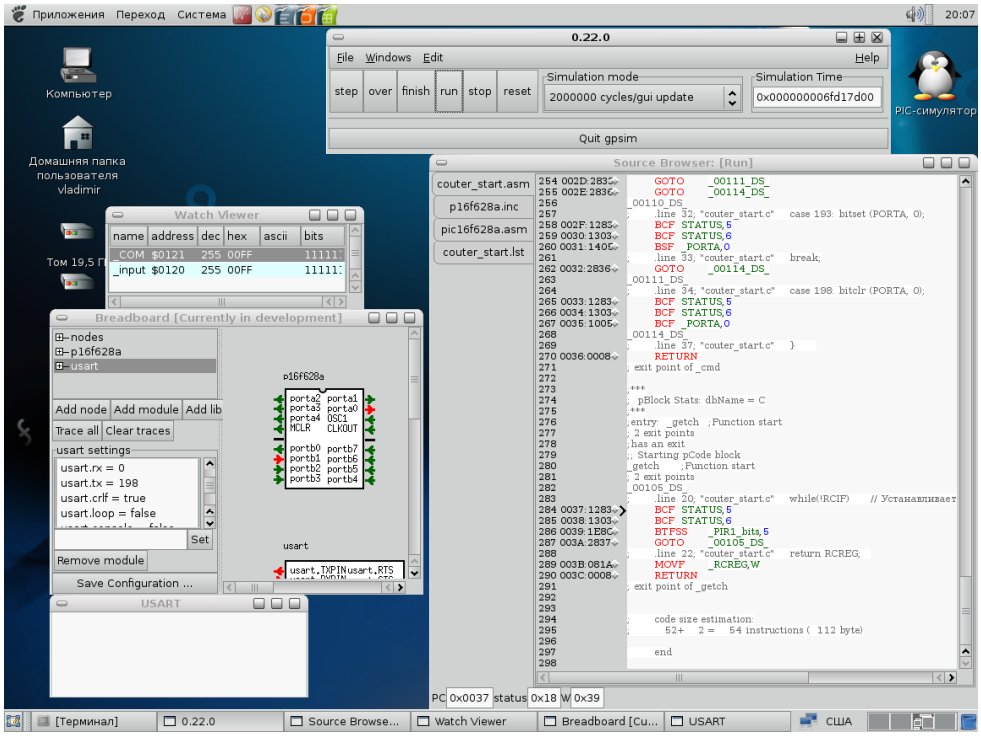


Рис. 2.33. Вид работающей отладочной программы gpsim

И все бы хорошо, когда бы не было плохо. Попытка запустить отладку вторично вновь сталкивает меня с проблемой зависания на функции getch(). Не вижу другого пути, как вернуться к чтению документации, тем более, что документация со времени моего первого знакомства с программой изменилась, а в пакете я нахожу папку с примерами. Приостановим-ка бессистемную работу, попробуем разобраться.

Глава 3. У камелька

Мне нравится словосочетание «у камелька». В нем есть что-то домашнее, уютное. Такое же ощущение должно появиться у вас при работе с Gambas, едва что-то начнет получаться. Но если вам не нравится название главы – зачеркните его, напишите такое, какое вам понравится. Любая книга, особенно техническая, даже если это роман или сказка «в технической прозе», как у меня, это не более, чем рабочий инструмент, мастерок каменщика или топор плотника. А инструмент должен быть удобным, это главное.

После опытов с «железом», что мне привычнее, я хотел бы вернуться к программной части рассказа. Более того, сделать эту часть только программной без милых моему сердцу воспоминаний о запахе канифоли и таинственном мерцании экрана осциллографа.

Я не программист с богатым опытом работы над проектами, умением писать коды сложных программ, не программист, которому есть что вспомнить о приключении уравнений или борьбе со злобными «багами», об удачах в строительстве классов и провалах в фундаменте интерфейсов. Поэтому я в очередной раз решил «сжульничать», воспользовавшись трудами опытных программистов, взяв за основу главы что-нибудь любопытное из нескольких книг, которые наметил купить.

Увы. Я не нашел их в ближайших книжных магазинах, не нашел все нужное мне в одном интернет-магазине, а затевать путешествие по всей Москве не захотелось. Да и есть ли в этом смысл, если моя задача рассказать не столько о том, каким увлекательным занятием является программирование, что так и есть, сколько показать, насколько легко можно пользоваться современными средами программирования для своих нужд.

Вспоминая, что выше я упоминал о том, как когда-то, давно-давным, мы с сыном писали программу для самообучения печати «вслепую», я подумал что этот пример будет не хуже любого другого. Любая задача становится интересной только тогда, когда становится вашей задачей. Если кто-то другой предлагает задачу, то ее решение для вас – это уже работа. И если я даже придумаю нечто невероятно интересное (для меня), оно может показаться скучным для остальных.

Я много лет печатаю русский текст «вслепую», но умение делать это с латинской раскладкой утратил вскорости после того, как обрел это умение. И все эти годы, если возникает нужда что-то напечатать, например, на английском, я поправляю очки, зависаю над клавиатурой и лихо долблю по клавиатуре двумя пальцами. Мне очень интересно, смогу ли я, пока дописываю эту книгу, если когда-нибудь допишу ее, за две-три недели восстановить умение печать не только кириллицей, но и латиницей?

Начало проекта «Машинистка»

Прежде, чем открыть новый проект в Gambas, мне, так уж я устроен, мне необходимо

как-то описать, а что, собственно, я намерен делать.

Я уже рассказывал о тех многочисленных возможностях, что существуют в Linux (практически в любом более или менее полном дистрибутиве) для такого рода работы. Но я хорошо знаю себя – если я начну работать с программой, помогающей мне составить план работы над проектом, я обязательно затею это в картинках, или с базой данных. Не то, что я не могу описать все словами, такой подход мне кажется даже более привлекательным, чем использование изображений, но я почти уверен, как бы хорошо я ни описал рисунок, вы, читая описание, увидите совсем другой, свой собственный. Великое благо в для искусства, эта особенность восприятия заставила изобрести формулы в математике и чертежи в технике. Программирование, хотя и родом из математики, все больше отдаляется от техники, сближаясь с искусством.

И все-таки, самым простым способом описать задачу остается лист бумаги и карандаш, или текстовый процессор и клавиатура. Чем я и воспользуюсь до появления сомнений в разночтении.

Что представляет собой процесс печати на пишущей машинке? Не будучи специалистом в этом вопросе, я воспользуюсь теми знаниями, что некогда получил при чтении книги «Осваиваем микрокомпьютер». Возможно, существует множество методик, но я использую ту, что использовал сам. Итак.

Основой этой методики является принцип обслуживания определенных клавиш клавиатуры пальцами ваших двух рук. За каждым пальцем закреплено несколько клавиш. В начальный момент восемь пальцев двух рук находятся на среднем ряду клавиатуры так, чтобы указательный палец левой руки был на латинской букве F, а указательный палец правой руки на букве J. Клавиатура имеет на клавишах выступы, которые позволяют легко на ощупь поставить руки в начальное положение.

Второй принцип метода требует, чтобы после нажатия на любую клавишу палец возвращался в исходную позицию. То есть печать выглядит так, что вы из начального положения печатаете любую букву соответствующим пальцем, а затем возвращаетесь к начальному положению.

И, наконец, это уже не принцип метода, а мой личный опыт. Не пытайтесь запомнить, где на клавиатуре находится та или иная буква. Даже если вы это делаете неосознанно, следите за этим, чтобы пресечь все попытки «заучить» клавиатуру. Пусть за вас это делают ваши пальцы – это их епархия, это их работа, это их «мозги» должны напрягаться. И это, похоже, важный момент. Когда я сам начинал учиться печатать, я понял, насколько это мешает мне быстро освоить печать.

Таким образом, теперь уже о программе, я предполагаю нарисовать клавиатуру, раскрасив разными цветами рабочее поле для каждого пальца. При печати эти раскрашенные клавиши будут менять цвет, когда нажата соответствующая клавиша.

Далее, над клавиатурой будет располагаться поле для вывода печатаемого текста, как лист бумаги, выступающий из пишущей машинки. После заполнения строки, это поле должно удлиняться, как если бы вы нажали перевод строки (если это так называется у пишущей машинки).

Последнее, что мне понадобится для программы, это поле, где будут выводиться

слова, которые следует напечатать. Дело в том, что методика предполагает поэтапное использование в начале обучения только основного ряда клавиш, на котором размещаются ваши руки в начальный момент, затем к этому ряду подключается использование ряда, расположенного над ним, затем под ним, и лишь позже вам будут предложены слова, использующие все три ряда клавиш. Пробел удобно нажимать большими пальцами рук, как удобнее в тот или иной момент. При необходимости печати больших букв можно использовать свободную руку для того, чтобы нажать клавишу **Shift**.

Пока я не готов составить полное описание программы, но мне ясно, с чего следует начать. А может, признаюсь, сказывается, что мне интереснее начать работать с программой, чем продумывать план работы. К плану, я уверен, придется вернуться, но сейчас я запущу Gamabs и создам новый графический проект.

Из предлагаемых при запуске вариантов я выбираю графический Qt проект. Имя проекту пусть будет Typist (машинистка). После двойного щелчка по FMain в менеджере проекта (левое окно) появляется основная форма программы. Я использую версию Gambas, работа над которой еще не завершена, а в стабильной версии форма, если я не ошибаюсь, появляется сразу. Но это не важно. Для того, чтобы нарисовать клавиши клавиатуры, я использую объекты TextLabel. В поле свойств Text (меню свойств объекта) я буду вводить буквы для клавиш, использую свойство Alignment для задания Center, чтобы буква располагалась в центре клавиши, и использую свойство Border для создания контура (я выбираю вариант Raised). Теперь я могу задать исходный цвет клавиши с помощью свойства Background, что важно для меня, поскольку мне нужно разметить «сферу» деятельности для каждого из восьми пальцев. Выбрав темно-зеленый цвет (на закладке Free), я вижу, что черные буквы на этом фоне, практически, не видны. По этой причине я меняю цвет Foreground для каждого из трех объектов TextLabel на белый. Несколько минут ушло у меня на первые шаги по построению проекта. Если его откомпилировать и запустить, то выглядеть это будет так:

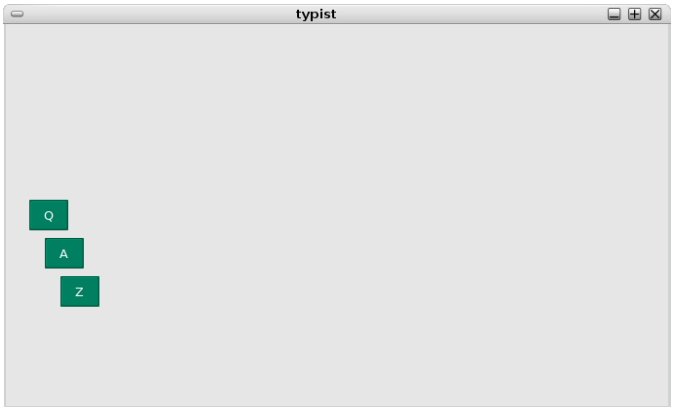


Рис. 3.1. Первые шаги в проекте «Машинистка»

Чтобы упростить работу я предполагаю скопировать это построение и повторить его еще девять раз. Должна получиться полная клавиатура без клавиши пробела. Затем я изменю исходные цвета и добавлю клавишу пробела.

Для копирования я, удерживая левую клавишу мышки, провожу курсором прямоугольник, захватывающий все три клавиши, после отпускания клавиши мышки объекты оказываются выделены. Насколько могу судить, операция совершенно одинаковая во многих приложениях и в разных операционных системах. Теперь, нажав правую клавишу мышки после размещения курсора на одном из выделенных объектов (и это довольно типично), из выпадающего меню я выбираю пункт Copy, а, щелкнув правой клавишей мышки на свободном месте формы, из выпадающего меню выбираю Paste. Мне остается подхватить мышкой полученные три клавиши и перенести их чуть правее уже готовых. Я пока не меняю ни букв внутри клавиш, ни их цвет. После того, как клавиатура будет готова, я изменю цвет и названия клавиш. Поскольку в работе принимает участие по четыре пальца каждой руки, я использую четыре цвета – темно-зеленый, синий, желтый и красный. Для второй половины клавиатуры эти цвета повторяются в обратном порядке. Цвет букв на желтом фоне приходится вернуть к черному, иначе их совсем не видно, и я разделяю клавиатуру для левой руки и правой больше, чем это имеет место в действительности, чтобы явно обозначить «сферы влияния». Пока у меня получается следующий вид работающей программы.

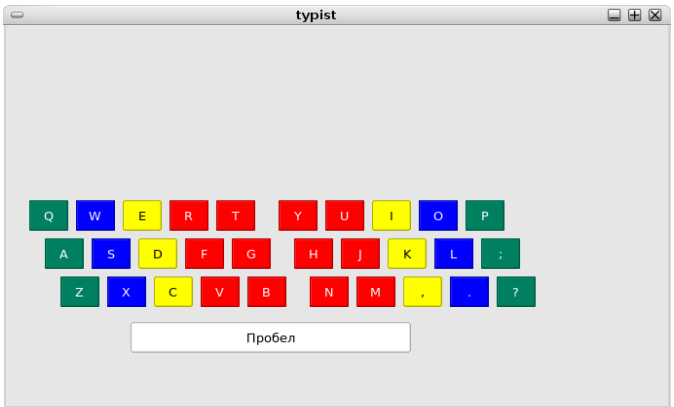


Рис. 3.2. Рисунок клавиатуры в проекте «Машинистка»

Как видно из рисунка, каждый палец обслуживает три клавиши, кроме указательных, у которых вдвое больше работы. Пробел, напомню, можно нажимать большим пальцем, каким в данный момент удобнее.

К рисунку я добавлю TextBox, объект изображающий лист бумаги, а чуть правее еще одну TextLabel, где должны появляться слова, которые следует напечатать. Рисунок приобретает почти окончательный вид, хотя прошло не более получаса с момента начала работы (с перекуром).

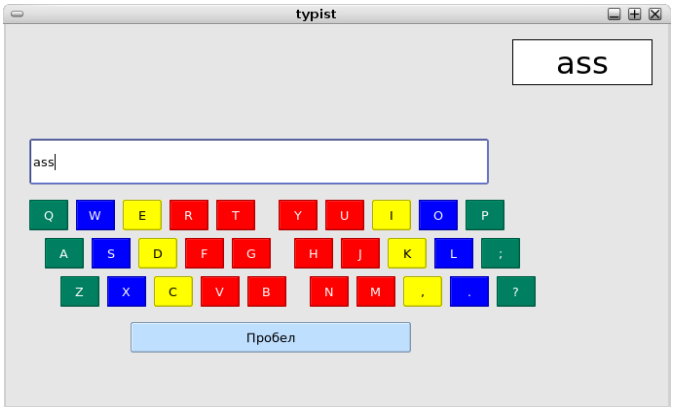


Рис. 3.3. Почти окончательный вид рисунка

Если программу запустить, то на листе бумаги уже можно начинать печатать.

Теперь мне хотелось бы, хотя это не обязательно, отображать нажатые клавиши изменением цвета. Задача для программиста, как мне кажется, совсем простая. Для программиста, но не для меня. Попробую подойти к решению, предполагая, что среди свойств объектов, размещенных мной на форме есть те, что помогут мне в осуществлении задуманного. С этой целью я нажму правой клавишей мышки на объекте TextBox, выберу раздел Event выпадающего меню и выберу пункт KeyPress. Теперь в коде программы появляется шаблон подпрограммы отвечающий этому событию. Поскольку я не знаю, прав ли я в своем предположении, я проведу опыт, добавив в шаблон простую строку кода. На рисунке клавиатуры клавиша «Q» изображена с помощью TextLabel. Если я ввожу это имя в редакторе кода, то появляется список всех этикеток, и мне достаточно выбрать TextLabel1 для моего эксперимента. После того, как я ставлю точку после выбранного объекта открывается меню выбора его свойств, где я могу выбрать свойства фона. А значение этого свойства я просто скопирую с соответствующего у клавиши пробела. Теперь код программы выглядит следующим образом:

```
PUBLIC SUB TextBox1_KeyPress()
       TextLabel1.Background = &HBFDFFF&
END
```

В итоге после запуска программы при нажатии на любую клавишу цвет клавиши

«Q» меняется. И остается таким. А это не совсем то, что мне нужно. Я хотел бы изменить эту часть кода примерно так:

```
PUBLIC SUB TextBox1_KeyPress()
        IF Key.Code = 113 THEN TextLabel1.Background = &HBFDFFF&
END
```

Значение 113 – это десятичное значение кода ASCII символа «q». Проверив этот вариант, я убеждаюсь, что он не работает. Обращение к руководству (help) еще больше огорчает меня. В руководстве явно не советуют использовать такой вариант работы с символами. Хорошо бы сейчас посоветоваться с опытным программистом, но такой возможности нет. Придется попытаться обойти проблему. И сделать это я хочу с помощью следующего кода:

```
PUBLIC SUB TextBox1_KeyPress()
        TextLabel32.Text = Str$(Key.Code)
END
```

При нажатии любой клавиши в окошко, которое я предназначил для появления слов-заданий, у меня оно имеет имя TextLabel32, в это окошко в виде строки будут отправляться коды символов нажатой клавиши, останется только проверить их. Первый эксперимент с выводом кода дает результат:

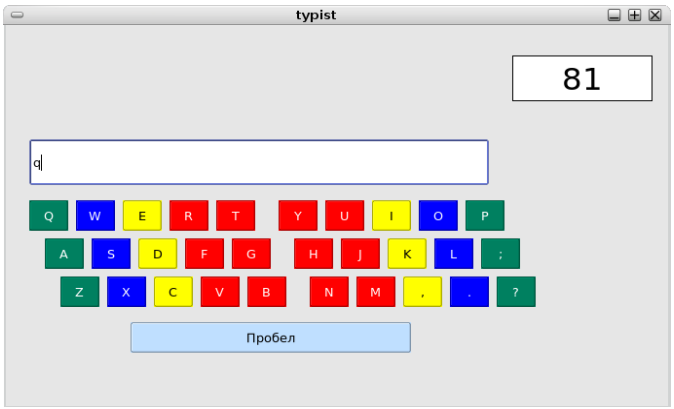


Рис. 3.4. Определение кода нажатой клавиши

Совет, насколько я понимаю, был правилен. Полученное число похоже на десятичное значение заглавной «Q», и я попробую проверить, что у меня получится, если я в программу добавлю событие – отпускание нажатой клавиши (с помощью раздела Event выпадающего меню и пункта KeyRelease при щелчке правой клавишей мышки на TextBox) с небольшой добавкой кода:

```
PUBLIC SUB TextBox1_KeyPress()
	IF Key.Code = 81 THEN TextLabel1.Background = &HBFDFFF&
END

PUBLIC SUB TextBox1_KeyRelease()
	IF Key.Code = 81 THEN TextLabel1.Background = &H008060&
END
```

Теперь, если это клавиша «q», она действительно меняет цвет при нажатии, возвращаясь к первоначальному при отпускании клавиши. Первоначальный цвет я в коде программы добавил, просто копируя его значение из меню свойств клавиши.

Я не думаю, что это хороший стиль программирования, но я не стилист, не программист, по причине чего использую то, что нашел. Я могу проверить коды всех нужных мне клавиш, дописать аналогичный текст для всех клавиш и проверить, все ли они правильно работают. Пока мне этого должно хватить. Тем более, что полученные коды совпадают с десятичными значениями заглавных букв кода ASCII, откуда их можно взять. Текст программы в этой части (а это пока все, что есть) приобретает вид:

```
' Gambas class file

PUBLIC SUB _new()
END

PUBLIC SUB Form_Open()
END

PUBLIC SUB TextBox1_KeyPress()
	IF Key.Code = 81 THEN TextLabel1.Background = &HBFDFFF& 'q
	IF Key.Code = 65 THEN TextLabel2.Background = &HBFDFFF& 'a
	IF Key.Code = 90 THEN TextLabel3.Background = &HBFDFFF& 'z
	IF Key.Code = 87 THEN TextLabel6.Background = &HBFDFFF& 'w
	IF Key.Code = 83 THEN TextLabel5.Background = &HBFDFFF& 's
	IF Key.Code = 88 THEN TextLabel4.Background = &HBFDFFF& 'x
	IF Key.Code = 69 THEN TextLabel9.Background = &HBFDFFF& 'e
	IF Key.Code = 68 THEN TextLabel8.Background = &HBFDFFF& 'd
	IF Key.Code = 67 THEN TextLabel7.Background = &HBFDFFF& 'c
	IF Key.Code = 82 THEN TextLabel12.Background = &HBFDFFF& 'r
	IF Key.Code = 70 THEN TextLabel11.Background = &HBFDFFF& 'f
	IF Key.Code = 86 THEN TextLabel10.Background = &HBFDFFF& 'v
	IF Key.Code = 84 THEN TextLabel15.Background = &HBFDFFF& 't
	IF Key.Code = 71 THEN TextLabel14.Background = &HBFDFFF& 'g
	IF Key.Code = 66 THEN TextLabel13.Background = &HBFDFFF& 'b
	IF Key.Code = 89 THEN TextLabel18.Background = &HBFDFFF& 'y
	IF Key.Code = 72 THEN TextLabel17.Background = &HBFDFFF& 'h
	IF Key.Code = 78 THEN TextLabel16.Background = &HBFDFFF& 'n
	IF Key.Code = 85 THEN TextLabel21.Background = &HBFDFFF& 'u
	IF Key.Code = 74 THEN TextLabel20.Background = &HBFDFFF& 'j
	IF Key.Code = 77 THEN TextLabel19.Background = &HBFDFFF& 'm
	IF Key.Code = 73 THEN TextLabel24.Background = &HBFDFFF& 'i
```

```
        IF Key.Code = 75 THEN TextLabel23.Background = &HBFDFFF& 'k
        IF Key.Code = 44 THEN TextLabel22.Background = &HBFDFFF& ',
        IF Key.Code = 79 THEN TextLabel27.Background = &HBFDFFF& 'o
        IF Key.Code = 76 THEN TextLabel26.Background = &HBFDFFF& 'l
        IF Key.Code = 46 THEN TextLabel25.Background = &HBFDFFF& '.
        IF Key.Code = 80 THEN TextLabel30.Background = &HBFDFFF& 'p
        IF Key.Code = 59 THEN TextLabel29.Background = &HBFDFFF& ';
        IF Key.Code = 47 THEN TextLabel28.Background = &HBFDFFF& '/
        IF Key.Code = 32 THEN TextLabel31.Background = &HFFFF00& 'Пробел
END

PUBLIC SUB TextBox1_KeyRelease()
        IF Key.Code = 81 THEN TextLabel1.Background = &H008060& 'q
        IF Key.Code = 65 THEN TextLabel2.Background = &H008060& 'a
        IF Key.Code = 90 THEN TextLabel3.Background = &H008060& 'z
        IF Key.Code = 87 THEN TextLabel6.Background = &H0000FF& 'w
        IF Key.Code = 83 THEN TextLabel5.Background = &H0000FF& 's
        IF Key.Code = 88 THEN TextLabel4.Background = &H0000FF& 'x
        IF Key.Code = 69 THEN TextLabel9.Background = &HFFFF00& 'e
        IF Key.Code = 68 THEN TextLabel8.Background = &HFFFF00& 'd
        IF Key.Code = 67 THEN TextLabel7.Background = &HFFFF00& 'c
        IF Key.Code = 82 THEN TextLabel12.Background = &HFF0000& 'r
        IF Key.Code = 70 THEN TextLabel11.Background = &HFF0000& 'f
        IF Key.Code = 86 THEN TextLabel10.Background = &HFF0000& 'v
        IF Key.Code = 84 THEN TextLabel15.Background = &HFF0000& 't
        IF Key.Code = 71 THEN TextLabel14.Background = &HFF0000& 'g
        IF Key.Code = 66 THEN TextLabel13.Background = &HFF0000& 'b
        IF Key.Code = 89 THEN TextLabel18.Background = &HFF0000& 'y
        IF Key.Code = 72 THEN TextLabel17.Background = &HFF0000& 'h
        IF Key.Code = 78 THEN TextLabel16.Background = &HFF0000& 'n
        IF Key.Code = 85 THEN TextLabel21.Background = &HFF0000& 'u
        IF Key.Code = 74 THEN TextLabel20.Background = &HFF0000& 'j
        IF Key.Code = 77 THEN TextLabel19.Background = &HFF0000& 'm
        IF Key.Code = 73 THEN TextLabel24.Background = &HFFFF00& 'i
        IF Key.Code = 75 THEN TextLabel23.Background = &HFFFF00& 'k
        IF Key.Code = 44 THEN TextLabel22.Background = &HFFFF00& ',
        IF Key.Code = 79 THEN TextLabel27.Background = &H0000FF& 'o
        IF Key.Code = 76 THEN TextLabel26.Background = &H0000FF& 'l
        IF Key.Code = 46 THEN TextLabel25.Background = &H0000FF& '.
        IF Key.Code = 80 THEN TextLabel30.Background = &H008060& 'p
        IF Key.Code = 59 THEN TextLabel29.Background = &H008060& ';
        IF Key.Code = 47 THEN TextLabel28.Background = &H008060& '/
        IF Key.Code = 32 THEN TextLabel31.Background = &HBFDFFF& 'Пробел
END
```

Добавляя часть кода я проверяю, работает ли она, поправляя «на ходу» ошибки, появляющиеся даже при копировании. Интересно было бы посмотреть, как эту часть программы могут кодировать программисты. Можно было бы поискать тексты в Интернете, но позже, поскольку есть «неполадки» поинтереснее. Но и с ними позже. Сейчас меня заботит другое – текст, который я ввожу, отображается строкой в TextBox. А мне хотелось бы, чтобы при нажатии клавиши **Enter** текст перемещался на следующую строку. Попробовав варианты с добавлением символа перевода строки

и возврата каретки, я прихожу к выводу, что есть простое решение – заменить TextBox на TextArea. Исправив в тексте кода имя объекта, я получаю все, что мне хотелось бы. При нажатии на клавиатуре ввода текст смещается вверх, а курсор оказывается на новой строке. Это оказывается гораздо проще, чем мои попытки проделать это в TextBox.

Далее мне хотелось бы решить следующую задачу. После нажатия на пробел слово-задание должно меняться. Но эти слова следует где-то разместить. Самый простой вариант – создать строковый массив.

После нескольких неудачных попыток мне удается разместить объявление массива (TestWords[]) в самом начале текста кода и инициализировать массив первыми тремя словами. А с помощью конструкции IF...THEN заставить как-то изменяться слова в поле TextLabel32 (где и положено появляться словам-заданиям). Для этого я добавляю к объявлению массива еще и индексную переменную «I».

```
' Gambas class file
PUBLIC TestWords AS String[] = ["ass", "add", "fad"]
PUBLIC I AS Integer = 0

PUBLIC SUB TextArea1_KeyRelease()
        ......................
        ......................
        IF Key.Code = 32 THEN
                TextLabel31.Background = &HFFFF00&
                TextLabel32.Text = TestWords[I]
                I = I + 1
        ENDIF 'Пробел
END
```

Развиваем успех

Но это, конечно, не все. Слов для работы на среднем ряду клавиатуры у меня 13. Хотелось бы, чтобы эти слова появлялись в окне задания непрерывно, то есть, после появления последнего слова вновь появлялось первое. На первом этапе обучения используются только слова, составленные из букв среднего ряда клавиатуры. А это значит, что существует второй этап обучения, когда слова должны содержать буквы среднего и верхнего ряда, затем третий этап, и четвертый этап. То есть, мне понадобится несколько массивов со словами-заданиями.

В данный момент, прав я или нет, я предполагаю добавить нужное количество текстовых массивов со словами заданиями (назову их Less1[]...Less4[]). Я сохраню текстовый массив TestWords[], как рабочий массив. А для переключения между заданиями использую основное меню, которое можно создать, если щелкнуть правой клавишей мышки в свободном месте формы и выбрать пункт Menu editor... в выпадающем меню. Открывается диалог создания меню, где можно, нажав клавишу Insert, ввести текст пунктов меню (Caption), и с помощью стрелок управления разместить вводимые пункты меню, как подменю основного меню.

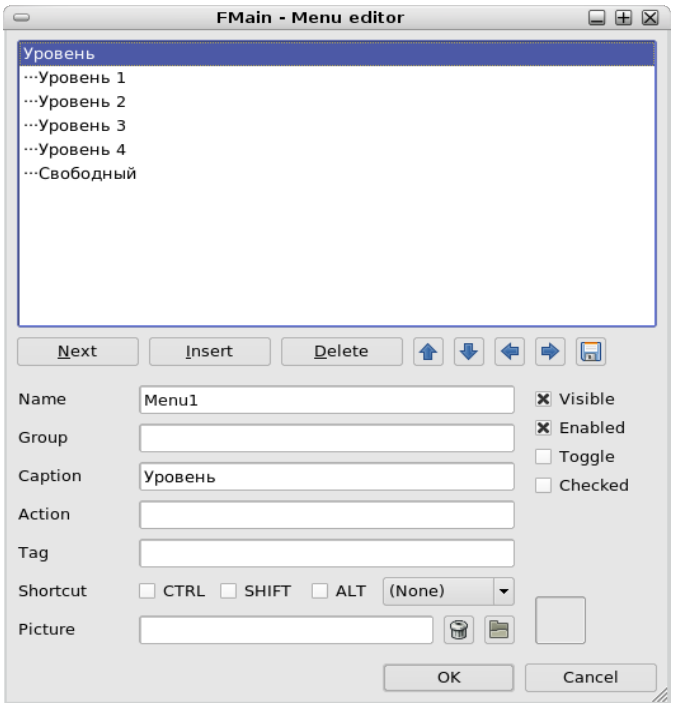


Рис. 3.5. Диалоговое окно создания основного меню в Gambas

Редактор позволяет сделать больше, но это сейчас для меня самое нужное. Поскольку, закончив создание этого меню, я после запуска программы могу открыть его на форме, выбрать с помощью курсора подменю «Уровень 1», и, щелкнув по нему, получить подпрограмму, где будет сделано присваивание содержимого первого текстового массива рабочему массиву:

```
PUBLIC SUB Menu2_Click()
        TestWords = Less1
        TextLabel32.Text = TestWords[I]
END
```

Здесь же я вывожу первое слово-задание в поле TextLabel32. Работа с программой должна выглядеть следующим образом: после запуска программы следует выбрать нужный уровень, а после появления первого слова задания можно приступать к работе. Если для первого уровня я списал слова-задания из книги, добавив два-три своих, то для второго и последующих уровней слова придется придумать. Начало работы с программой выглядит так:

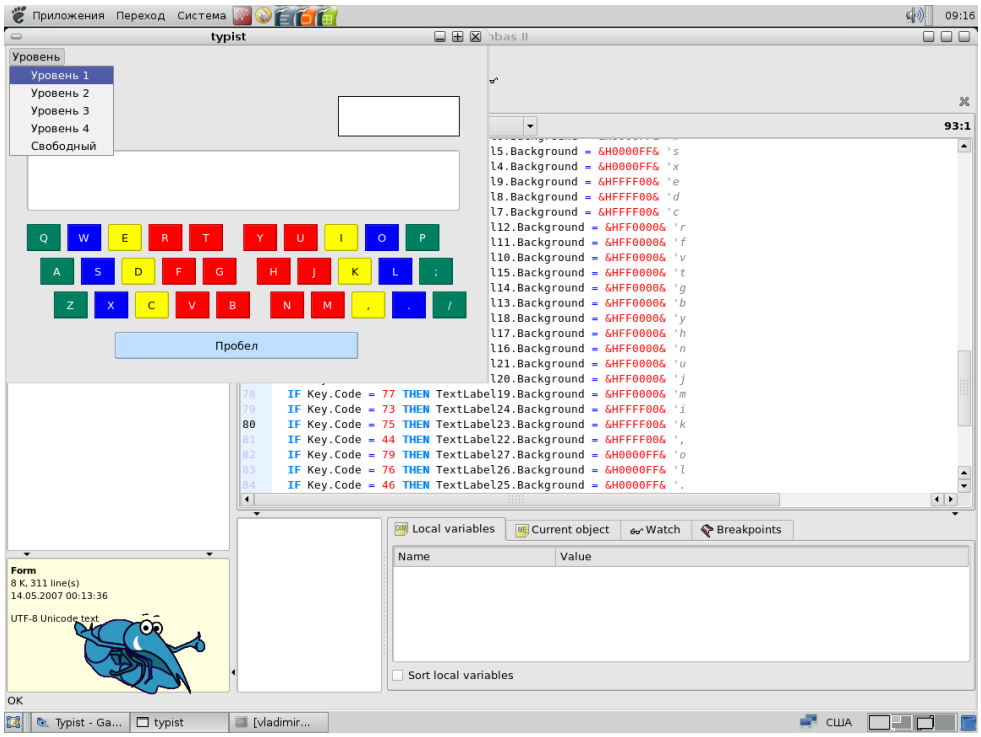


Рис. 3.6. Начало работы с программой

Я не знаю, есть ли какие-либо закономерности в словах-заданиях в той методике, которую пытаюсь реализовать, и нужно ли, чтобы все слова были осмысленными, или лучше, если они будут бессмысленными? Я попробую видоизменить то, что есть, пока моя фантазия не исчерпает моего терпения.

Слова первого уровня:

ass, add, fad, all, lass, falls, lads, dad, alas, ask, salad, hag, jak, jagh.

Из них создадим:

wass, qadd, fat, yell, kugg, talihg, lords, pad, yellow, krast, gatapul, jegos, tajer, troll.

Для третьего уровня:

azax, valan, lams, xaj, fan, sabac, malz, cagh, jana, balls, mama, xanas, vagaz, bags.

И для четвертого уровня я опять воспользуюсь книгой:

fox, box, soks, hnoks, ring, table, tan, whale, mate, rain, sent, tank, scary, plum.

После добавления кода, в основном путем копирования, программа приобретает вид:

```
'' Gambas class file

PUBLIC TestWords AS String[13]

PUBLIC Less1 AS String[] = ["ass", "add", "fad", "all", "lass", "falls",
"lads", "dad", "alas", "ask", "salad", "hag", "jak", "jagh"]

PUBLIC Less2 AS String[] = ["wass", "qadd", "fat", "yell", "kugg", "talihg",
"lords", "pad", "yellow", "krast", "gatapul", "jegos", "tajer", "troll"]

PUBLIC Less3 AS String[] = ["azax", "valan", "lams", "xaj", "fan", "sabac",
"malz", "cagh", "jana", "balls", "mama", "xanas", "vagaz", "bags"]

PUBLIC Less4 AS String[] = ["fox", "box", "soks", "hnoks", "ring", "table",
"tan", "whale", "mate", "rain", "sent", "tank", "scary", "plum"]

PUBLIC I AS Integer = 0

PUBLIC SUB _new()
END

PUBLIC SUB Form_Open()
END

PUBLIC SUB TextArea1_KeyPress()
        IF Key.Code = 81 THEN TextLabel1.Background = &HBFDFFF& 'q
        IF Key.Code = 65 THEN TextLabel2.Background = &HBFDFFF& 'a
        IF Key.Code = 90 THEN TextLabel3.Background = &HBFDFFF& 'z
        IF Key.Code = 87 THEN TextLabel6.Background = &HBFDFFF& 'w
        IF Key.Code = 83 THEN TextLabel5.Background = &HBFDFFF& 's
        IF Key.Code = 88 THEN TextLabel4.Background = &HBFDFFF& 'x
```

```
        IF Key.Code = 69 THEN TextLabel9.Background = &HBFDFFF& 'e
        IF Key.Code = 68 THEN TextLabel8.Background = &HBFDFFF& 'd
        IF Key.Code = 67 THEN TextLabel7.Background = &HBFDFFF& 'c
        IF Key.Code = 82 THEN TextLabel12.Background = &HBFDFFF& 'r
        IF Key.Code = 70 THEN TextLabel11.Background = &HBFDFFF& 'f
        IF Key.Code = 86 THEN TextLabel10.Background = &HBFDFFF& 'v
        IF Key.Code = 84 THEN TextLabel15.Background = &HBFDFFF& 't
        IF Key.Code = 71 THEN TextLabel14.Background = &HBFDFFF& 'g
        IF Key.Code = 66 THEN TextLabel13.Background = &HBFDFFF& 'b
        IF Key.Code = 89 THEN TextLabel18.Background = &HBFDFFF& 'y
        IF Key.Code = 72 THEN TextLabel17.Background = &HBFDFFF& 'h
        IF Key.Code = 78 THEN TextLabel16.Background = &HBFDFFF& 'n
        IF Key.Code = 85 THEN TextLabel21.Background = &HBFDFFF& 'u
        IF Key.Code = 74 THEN TextLabel20.Background = &HBFDFFF& 'j
        IF Key.Code = 77 THEN TextLabel19.Background = &HBFDFFF& 'm
        IF Key.Code = 73 THEN TextLabel24.Background = &HBFDFFF& 'i
        IF Key.Code = 75 THEN TextLabel23.Background = &HBFDFFF& 'k
        IF Key.Code = 44 THEN TextLabel22.Background = &HBFDFFF& ',
        IF Key.Code = 79 THEN TextLabel27.Background = &HBFDFFF& 'o
        IF Key.Code = 76 THEN TextLabel26.Background = &HBFDFFF& 'l
        IF Key.Code = 46 THEN TextLabel25.Background = &HBFDFFF& '.
        IF Key.Code = 80 THEN TextLabel30.Background = &HBFDFFF& 'p
        IF Key.Code = 59 THEN TextLabel29.Background = &HBFDFFF& ';
        IF Key.Code = 47 THEN TextLabel28.Background = &HBFDFFF& '/
        IF Key.Code = 32 AND IF I < 13 THEN
                I = I + 1
                TextLabel31.Background = &HFFFF00&
                TextLabel32.Text = TestWords[I]
        ELSE IF Key.Code = 32 AND IF I = 13 THEN
                I = 0
                TextLabel31.Background = &HFFFF00&
                TextLabel32.Text = TestWords[I]
        ENDIF 'Пробел
END

PUBLIC SUB TextArea1_KeyRelease()
        IF Key.Code = 81 THEN TextLabel1.Background = &H008060& 'q
        IF Key.Code = 65 THEN TextLabel2.Background = &H008060& 'a
        IF Key.Code = 90 THEN TextLabel3.Background = &H008060& 'z
        IF Key.Code = 87 THEN TextLabel6.Background = &H0000FF& 'w
        IF Key.Code = 83 THEN TextLabel5.Background = &H0000FF& 's
        IF Key.Code = 88 THEN TextLabel4.Background = &H0000FF& 'x
        IF Key.Code = 69 THEN TextLabel9.Background = &HFFFF00& 'e
        IF Key.Code = 68 THEN TextLabel8.Background = &HFFFF00& 'd
        IF Key.Code = 67 THEN TextLabel7.Background = &HFFFF00& 'c
        IF Key.Code = 82 THEN TextLabel12.Background = &HFF0000& 'r
        IF Key.Code = 70 THEN TextLabel11.Background = &HFF0000& 'f
        IF Key.Code = 86 THEN TextLabel10.Background = &HFF0000& 'v
        IF Key.Code = 84 THEN TextLabel15.Background = &HFF0000& 't
        IF Key.Code = 71 THEN TextLabel14.Background = &HFF0000& 'g
        IF Key.Code = 66 THEN TextLabel13.Background = &HFF0000& 'b
        IF Key.Code = 89 THEN TextLabel18.Background = &HFF0000& 'y
        IF Key.Code = 72 THEN TextLabel17.Background = &HFF0000& 'h
        IF Key.Code = 78 THEN TextLabel16.Background = &HFF0000& 'n
```

```
        IF Key.Code = 85 THEN TextLabel21.Background = &HFF0000& 'u
        IF Key.Code = 74 THEN TextLabel20.Background = &HFF0000& 'j
        IF Key.Code = 77 THEN TextLabel19.Background = &HFF0000& 'm
        IF Key.Code = 73 THEN TextLabel24.Background = &HFFFF00& 'i
        IF Key.Code = 75 THEN TextLabel23.Background = &HFFFF00& 'k
        IF Key.Code = 44 THEN TextLabel22.Background = &HFFFF00& ',
        IF Key.Code = 79 THEN TextLabel27.Background = &H0000FF& 'o
        IF Key.Code = 76 THEN TextLabel26.Background = &H0000FF& 'l
        IF Key.Code = 46 THEN TextLabel25.Background = &H0000FF& '.
        IF Key.Code = 80 THEN TextLabel30.Background = &H008060& 'p
        IF Key.Code = 59 THEN TextLabel29.Background = &H008060& ';
        IF Key.Code = 47 THEN TextLabel28.Background = &H008060& '/
        IF Key.Code = 32 THEN TextLabel31.Background = &HBFDFFF& 'Пробел
END

PUBLIC SUB Menu2_Click()
        TestWords = Less1
        TextLabel32.Text = TestWords[I]
END

PUBLIC SUB Menu3_Click()
        TestWords = Less2
        TextLabel32.Text = TestWords[I]
END

PUBLIC SUB Menu4_Click()
        TestWords = Less3
        TextLabel32.Text = TestWords[I]
END

PUBLIC SUB Menu5_Click()
        TestWords = Less4
        TextLabel32.Text = TestWords[I]
END
```

Программу можно запустить, и начать в ней работать.

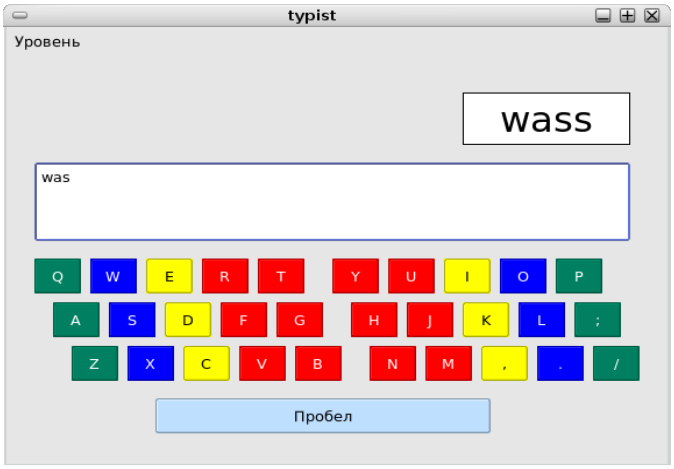


Рис. 3.7. Работающая завершенная программа проекта «Машинистка»

Последнее, что я сделаю, это создам исполняемый файл. Для этого в разделе основного меню Project я выберу пункт Make executable..., где можно указать папку для установки программы (я ее установлю в домашнюю папку) и можно выбрать опцию создания ярлычка на рабочем столе. Картинку к этому ярлычку (справа с именем typist) я добавил позже.

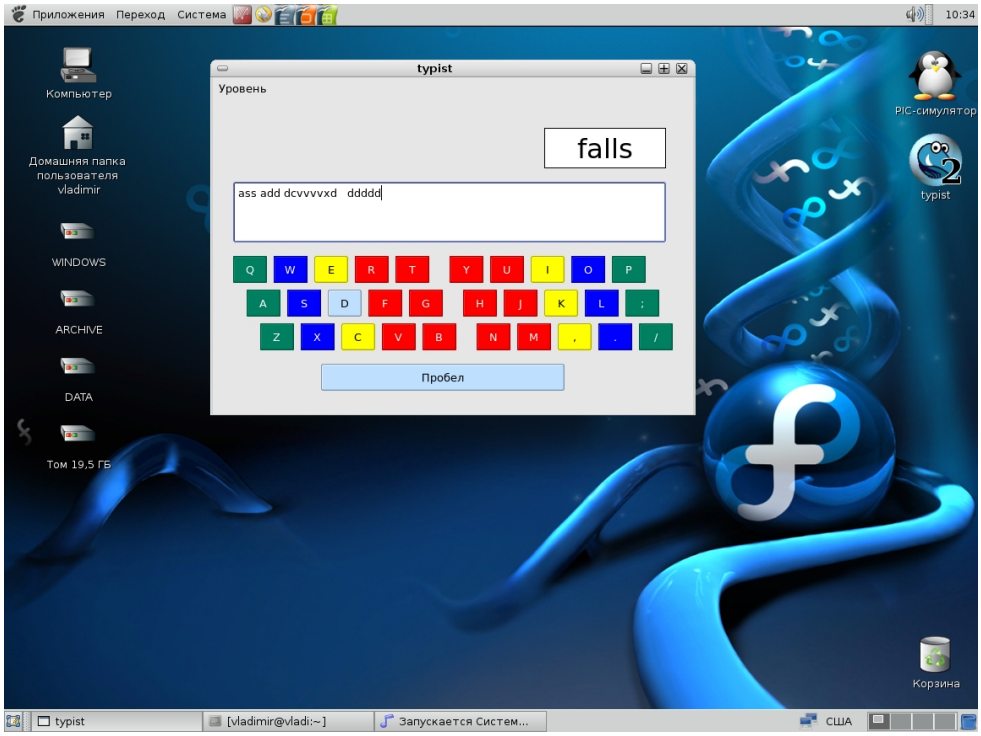


Рис. 3.8. Запуск программы как исполняемой

Она запускается и работает, как любая другая программа. Немного поразмыслив, я решил добавить в основное меню пункт «Выход», то есть, добавить код:

```
PUBLIC SUB Menu7_Click()
       FMain.Close
END
```

И на этом завершить работу.

Заметки и пометки «на память»

Для начала я хочу сказать, что изменение цвета при нажатии клавиши процедура не обязательная, возможно, это даже будет мешать. Если эту процедуру убрать, то код программы уместится на «четвертушке» листка бумаги. Для опытного программиста несколько минут работы. Для меня, положим, день. Много ли это, чтобы создать работающую полезную программу? Как вы думаете?

По мере создания кода, в основном копированием, я использовал только подсказку по языку, да некоторые эксперименты по выяснению того, что у меня происходит. Вместе с тем, система программирования Gambas оснащена отладчиком (которым я пока не умею пользоваться), как и положено любой системе программирования.

Проблемы с программой, насколько я понимаю, у меня остались. Например, хотя я и планировал создать программу, помогающую освоить печать вслепую для латинской раскладки клавиатуры, от программы было бы больше пользы, если бы дополнительно можно было выбрать кириллицу, чтобы научиться печать на родном языке. Но.

Но тогда мне точно пришлось бы отказаться от «мигающих» клавиш, поскольку я не получал кодов, когда проверял их. Причина, как мне кажется, в использовании UTF-8 и системных настроек. Это можно было бы исследовать дополнительно, но любые исследования влекут появление новых интересных проблем, исследование которых... и т.д. Когда-то надо остановиться, и чем раньше, тем лучше.

Текст слов-заданий можно было бы разместить в произвольном текстовом файле, чтобы считывать его при необходимости. Но этот файл был бы еще одним файлом, без которого программа не смогла бы работать, а для этого нужны веские основания. Я решил ограничиться строковыми массивами.

Если вы захотите повторить программу и научиться печатать вслепую, я приведу несколько советов из книги, откуда взята методика:

> *«Что нужно помнить?*
> *Представьте себе, что основные клавиши, как магниты притягивают ваши пальцы в основную позицию.*
> *Чтобы вспомнить, где находится нужная клавиша и каким пальцем ее нужно нажать, смотрите на схему клавиатуры (на экране), а не на саму клавиатуру».*

И добавлю от себя:

> *Не пытайтесь запомнить расположение клавиш на клавиатуре, это только повредит вам. То есть, не думайте о том, где нужная клавиша, но о том, каким пальцем до нее добраться. Думайте о пальцах, об остальном пусть думают они.*

Не спешите, пытаясь за день освоить печать. Каждодневные упражнения по 15-30 минут, переход на другой уровень, а через день возвращение на прежний – все это нормально. Повторяйте предыдущие уровни, переходите на следующие, пока не почувствуете, что можете печатать. Тогда начинайте печатать свободный текст (в программе), затем в любом редакторе. И не бросайте это после того как освоили печать. Должно пройти какое-то время, чтобы привычка закрепилась.

До завершения работы над книгой (если я надумаю ее завершать), я постараюсь, следуя своим собственным советам, освоить печать латиницей. Если и когда это произойдет, я обязательно расскажу вам, что из этого получилось.

Зачем я вообще затеял рассказ о программе для обучения печати вслепую? Мне хотелось показать, насколько это просто при работе в современной среде программирования. Порой появляются проблемы, решать которые удобнее, используя программирование. Если вы думаете, что с созданием программы вам не справиться, вы можете оказаться на пути, который займет больше времени и отнимет у вас больше сил, а, в конечном счете, может и не дать нужного решения.

Некогда, принимая решение, сделать ли устройство с использованием процессора или обойтись более дешевыми цифровыми микросхемами, я решил отказаться от процессора. Решение с цифровыми микросхемами, когда было найдено, казалось достаточно простым, дешевым и легким в реализации. Однако первые же испытания устройства потребовали коррекции. А первые испытания готового образца выявили необходимость и в смене некоторых принципиальных подходов. В итоге переделки, которые занимали бы минуты при использовании процессора, выливались в дни.

Я не программист, для меня привычнее лист бумаги и карандаш, паяльник и приборы. Мне, например, кажется более утомительным использование программы симуляции работы электрической схемы вместо того, чтобы на клочке бумаги набросать участок схемы, измерить или посмотреть осциллографом, и все понять. Перебирая такие клочки, я готов честно признаться, что я не прав.

И еще одно. Программирование само по себе очень увлекательно. Вот я упомянул, что не получил кодов клавиш при переключении раскладки клавиатуры на русскую. Уверен, что поиск причины этого окажется увлекательнейшим приключением. А смысл приключения не только в том удовольствии, которое получишь, когда доберешься до финиша, но и в том, что, возможно, будет получен ответ на вопрос, почему при симуляции работы микроконтроллера, о чем я говорил в предыдущей главе, у меня возникли некоторые проблемы с подачей команд. Может быть эти две проблемы не связаны, или связаны?

Я уже говорил, что тот способ, которым я заставляю «мигать» клавиши схемы клавиатуры на экране, не слишком хорош. Он не позволяет быстро переделать программу под обучение печати русских текстов. Поменяйте соответствующие строки на строки вида

```
IF Key.Code = 81 THEN TextLabel1.Background = &HBFDFFF& 'q
```

на

```
IF Key.Text = "й" THEN TextLabel1.Background = &HBFDFFF& 'й
```

и получите возможность работать с русским языком. Конечно, это следует поменять и в том месте кода, где обрабатывается отпускание клавиши, поменять слова-задания в массивах. Можно сделать программу двуязычной. Копирования при написании кода программы больше, но сама программа изменится мало.

Последнее, о чем следует сказать, относится к тем, кто всегда стремится к совершенству всех своих творений. В Linux есть профессионально созданная программа аналогичного назначения, которая называется KTouch. Она, видимо, использует более совершенную методику обучения, соответственно иначе построена. Если вам захочется внести подобные усовершенствования в свой проект, то можно ориентироваться на эту программу.

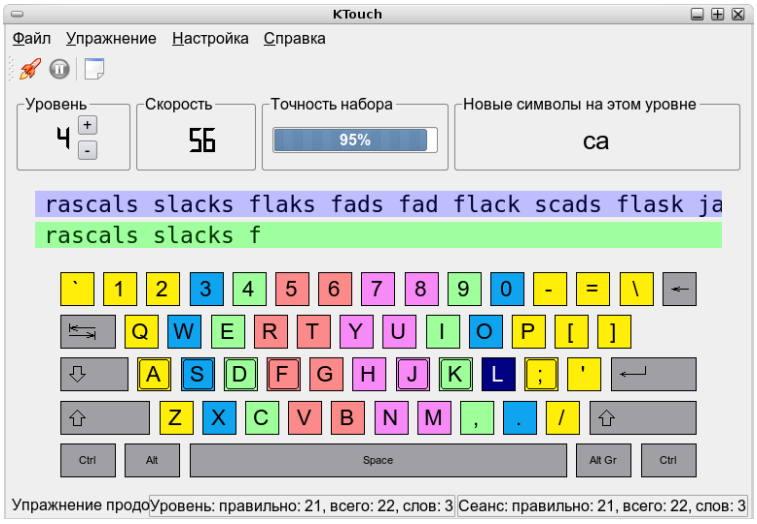


Рис. 3.9. Работа программы KTouch

Глава 4. Охота на кентавра

Засада в интерфейсе

Пришла пора соединить интерфейс, созданный в Gambas, с микроконтроллером, запрограммированным в Piklab. Прежде, чем осуществить, и, конечно, проверить стыковку «на орбите», следует подправить обе части будущего единого целого. С этого и начнем.

Напомню, как выглядит интерфейсная часть проекта.

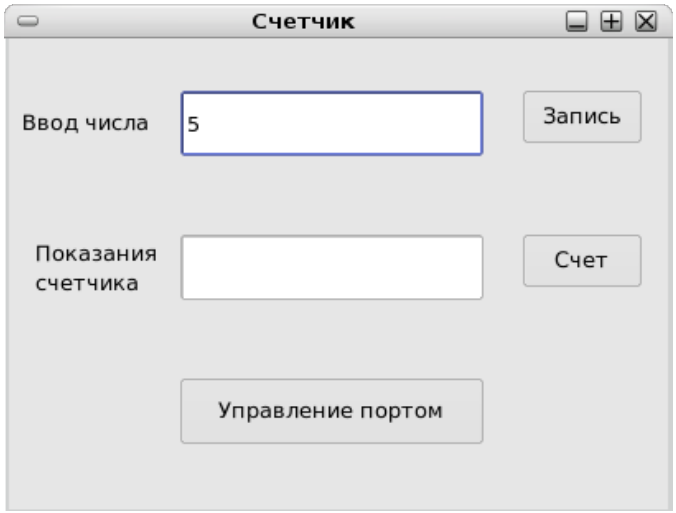


Рис. 1.4. Работа интерфейсной части проекта

Начинаем мы работу с нажатия на клавишу «Управление портом», чтобы открыть порт. Завершив работу повторно нажимаем эту клавишу для закрывания порта.

В окно ввода числа мы можем вписать число от 0 до 15. Не забыли, у нас 4 бита для записи в счетчик. После ввода числа нажимаем на клавишу **Запись**, программа должна отправить введенное число в COM-порт. Дальше об этом числе должен позаботиться микроконтроллер, то есть, принять число, перенести его в свой порт, соединенный с выводами счетчика, предназначенными для задания значения предустановки, а затем, после того как мы нажмем на клавишу **Запись**, интерфейс даст команду контроллеру на формирование сигнала предустановки. Контроллер, по окончании формирования импульса, должен прочитать состояние выходных бит счетчика и передать это в символьном виде программному интерфейсу. Интерфейс

после получения числа должен вывести его в окно «Показания счетчика».

Из приведенного выше описания пожеланий следует сделать некий план работы. Не хочется называть его алгоритмом. План.

- Ввод числа в окно ввода (обеспечено системой).
- Нажатие (click) клавиши **Запись** отправляет число в символьном виде в COM-порт, предваренное символом команды «a», по которой микроконтроллер начнет обработку числа, формирование импульса записи и отправку в COM-порт символьного значения выходов счетчика.
- Интерфейс ожидает появление ответа в COM-порте и выводит в окно «Показания счетчика» полученное число или сообщение об отсутствии ответа от контроллера после, например, 1 секунды ожидания.

Вот такой план. По мере его осуществления, я думаю, его придется подправить, но такова его участь. Запускаем среду программирования Gambas и начинаем с того места, где остановились в проекте «counter».

Как и следовало ожидать, первое, что я делаю, меняю план. Я подумал, что после запуска интерфейса, если когда-нибудь он будет готов, я могу забыть включить порт, и сразу попытаться ввести число и нажать на клавишу записи. Чтобы напомнить себе о необходимости открыть порт, я добавлю следующий код:

```
PUBLIC SUB Button1_Click()
	IF SerialPort1.Status THEN
		TextBox1.BackColor = Color.Green
	ELSE
		TextBox1.Text = "Вы забыли включить порт"
	ENDIF
END
```

Вы помните, что для получения шаблона подпрограммы обработки нажатия клавиши, достаточно двойного щелчка мышкой по клавише на форме. В данном случае клавиша **Запись** – это Button1. А фрагмент кода я вырезал из предыдущей заготовки проекта в том месте, где обрабатывалось нажатие на клавишу **Управление портом**. Проверим, как работает эта вставка.

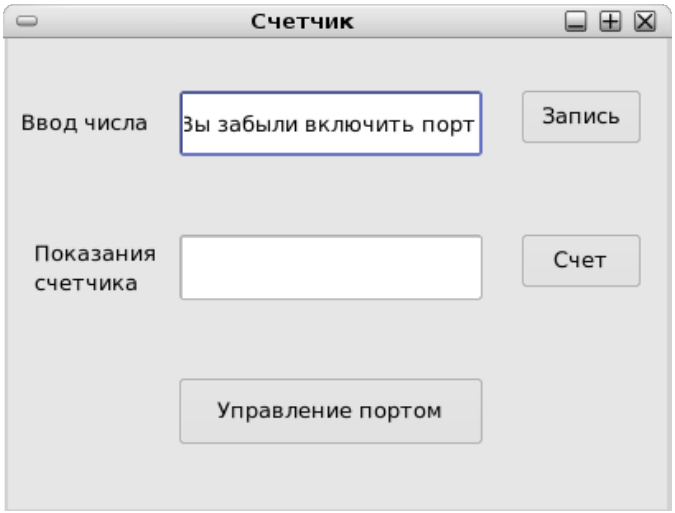


Рис. 4.2. Проверка изменений плана работы

Программный код в данный момент выглядит так:

```
' Gambas class file

PUBLIC SUB _new()
END

PUBLIC SUB Form_Open()
END


PUBLIC SUB ToggleButton1_Click() 'Клавиша "Управление портом"
	IF ToggleButton1.Value THEN
		SerialPort1.Open
		TextBox1.Text = SerialPort1.Status
	ELSE
		SerialPort1.Close
		TextBox1.Text = SerialPort1.Status
		TextBox1.BackColor = Color.Default
	ENDIF
END

PUBLIC SUB Button1_Click() ' Клавиша "Запись"
	IF SerialPort1.Status THEN
		TextBox1.BackColor = Color.Green
	ELSE
		TextBox1.Text = "Вы забыли включить порт"
	ENDIF
END
```

Теперь по плану. Нажатием клавиши **Запись** отправим в порт символ команды «a» и символ числа, которое введем. Поскольку число 15 имеет два символа, подумаем, как их переправить в контроллер. Для упрощения задачи я предполагаю использовать два символа при записи всех чисел. То есть, 1 будет выглядеть как 01. Если этот вариант вам не нравится, то можно сейчас (или позже) добавить блок кода для автоматического преобразования, или при программировании контроллера учесть эту особенность ввода чисел.

Как отправить символ в COM-порт (порт последовательного ввода-вывода)? Вот, что я нахожу в разделе help:

> *SerialPort (gb.net)*
> *This class was designed to allow to communicate using a serial interface (usually RS-232 serial port). This class inherits from Stream class, so you can use standard streams methods to send and receive data, and to close the port.*
>
> *Последовательный порт (раздел gb.net)*
> *Этот класс был разработан для коммуникаций с использованием последовательного интерфейса (обычно, последовательный порт RS232). Класс наследует от класса Stream (поток), так что вы можете использовать стандартные потоковые метода для отправки и получения данных, и для закрывания порта.*

Насколько я понимаю, мне нужно отправиться к справке по работе с потоком:

> *Stream (gb)*
> *This class is the parent class of every Gambas object that are stream.*
> *These objects can be used with all the input/output Gambas functions: PRINT, INPUT, LINE INPUT, CLOSE, and so on.*
>
> *Поток (раздел gb)*
> *Этот класс родительский для каждого Gambas объекта, являющегося потоком.*
> *Эти объекты могут быть использованы со всеми функциями ввода-вывода Gmabas: PRINT, INPUT, LINE INPUT, CLOSE и т.д.*

Таким образом, для отправки чего-либо в последовательный порт, я могу (или предполагаю, что могу) воспользоваться функцией PRINT. Хорошо. В help есть описание синтаксиса этой функции:

```
PRINT [#hStream ,] Expression [{;|;;| ,} Expression ...] [{ ; | ;; | , }]
```

Попробуем подставить вместо hStream наш порт SerialPort1, а что касается выражения (Expression), то я не знаю, что с ним делать. И после нескольких неудачных попыток останавливаюсь на таком варианте блока программы:

```
PUBLIC SUB Button1_Click() 'Клавиша "Запись"
	IF SerialPort1.Status THEN
		TextBox1.BackColor = Color.Green
		PRINT #SerialPort1 "a"
		PRINT #SerialPort1 TextBox1.Text
	ELSE
		TextBox1.Text = "Вы забыли включить порт"
	ENDIF
END
```

При нажатии на клавишу **Запись**, если последовательный порт открыт, то текстовое поле зеленеет (Color.Green), в порт отправляется символ «a», затем отправляется то, что введено в зеленеющее поле текстового ввода (TextBox1.Text). Если же я забыл открыть порт, то получаю в этом поле сообщение «Вы забыли включить порт» (надо было написать «открыть порт», но исправлять лень).

В таком виде при компиляции и после запуска программы ошибок не возникает. Не то, что в предыдущие попытки, когда я пытался понять, что мне делать с этими выражениями в функции печати. Я их превращал в значения, затем превращал в строки, пытался делать еще что-то не менее «умное», но ошибки возникали либо при компиляции, либо при работе.

Будем считать, что с этим блоком кода программы я справился.

До каких пор будем так считать? Ровно до тех, пока не станет ясно, что все это не так. Но при таком подходе хотелось бы ясности уже сейчас, пока кода мало, и легче разобраться с ошибками. Беда только в том, что я не знаю как проверить, действительно ли я отправляю что-то в порт, или это правильная с точки зрения языка конструкция, но не выполняющая своего предназначения.

Можно было бы подключить кабелем второй компьютер, на котором запустить программу чтения из последовательного порта (ее тоже быстренько написать?).

Я поступлю иначе, а вы решите для себя, как удобнее вам. Я использую то, что у меня осталось от предыдущих экспериментов, когда я макетировал модули для «умного дома». С макетной платой, на которой я размещал микроконтроллер, «общение» происходило именно по COM-порту через преобразователь интерфейса RS232 в RS485. Ниже я приведу схемы. А макетную плату я использую для разрешения сомнений.

Железное решение

Вот и вновь передо мною среда программирования микроконтроллеров Piklab, напомню, что я использую контроллер PIC16F628A и язык программирования Си.

Вот и вновь передо мною Piklab, где я подправлю программу, чтобы для начала получить команду «a», по которой зажгу светодиод, уже установленный на макетной плате. Вообще, это удобно – использовать светодиоды на выводах порта контроллера. Можно многое проверить в разных экспериментах. Проект в Piklab тоже называется «counter».

Я приведу только блок кода, в котором (по некоторому размышлению) изменил обработку полученных команд:

```
void cmd() // Получение и выполнение команды
{
        COMMAND = input;

        switch (COMMAND) {
                case 'a': bitset (PORTA, 0); // «a», включить светодиод
                break;
                case 'b': bitclr (PORTA, 0); // «b», выключить
                break;
                default: break;
        }
}
```

Перед использованием Gambas-интерфейса я приведу команды в соответствие.

Мне нужно только оттранслировать программу, выбрав в основном меню в разделе Build пункт Build Project. Программатор с установленной в панельку микросхемой контроллера у меня подключен давно. Остается подключить его к программе, раздел основного меню Programmer, а пункт Connect. И можно, проверив на странице Hex-кода, что слово конфигурации соответствует моим пожеланиям (2118, так оно выглядит в Piklab), можно записать программу в контроллер, что я и делаю. Для этого использую клавишу на основной инструментальной панели с изображением микросхемы, в которую направлена стрелка.

Программируется микросхема довольно быстро (программа маленькая), наученный горьким опытом, я проверяю правильность записи, щелкнув по клавише инструментальной панели с вопросительным знаком поверх микросхемы. Все в порядке. Остается самая для меня неприятная часть работы – найти, куда я убрал (запихнул) макетную плату и блок питания. Пообедаю, и найду (?).

Скоро сказка сказывается, да не скоро дело делается. Но, с другой стороны, «сколько веревочке ни виться, а пора приниматься за работу».

В программе интерфейса – я вновь в Gambas – я использую вторую клавишу **Счет** для выключения светодиода путем отправки символа команды выключения «b». В блоке кода щелчка по клавише **Запись** я закомментирую лишнюю команду, и новый

блок получается таким:

```
PUBLIC SUB Button2_Click() ' Клавиша "Запись"
       PRINT #SerialPort1 "b"
END
```

Увы. То, что так убедительно и красиво на бумаге, в жизни бывает не менее убедительно, но не так красиво. Не работает макет, а причин может быть много. Первым делом нужно проверить, отправляется ли что-то в порт, когда я нажимаю клавиши передачи символов управления.

В моем распоряжении есть, по меньшей мере, два способа это проверить – включить осциллограф, чтобы попытаться увидеть сигнал на выходе COM-порта, или попытаться мультиметром увидеть изменение сигнала на том же выводе порта или в линии RS485. Если с линией мне все ясно, она хорошо выделена у меня на макетной плате, то с выводом порта сложнее. Для начала следует найти схему включения конвертера RS232-RS485.

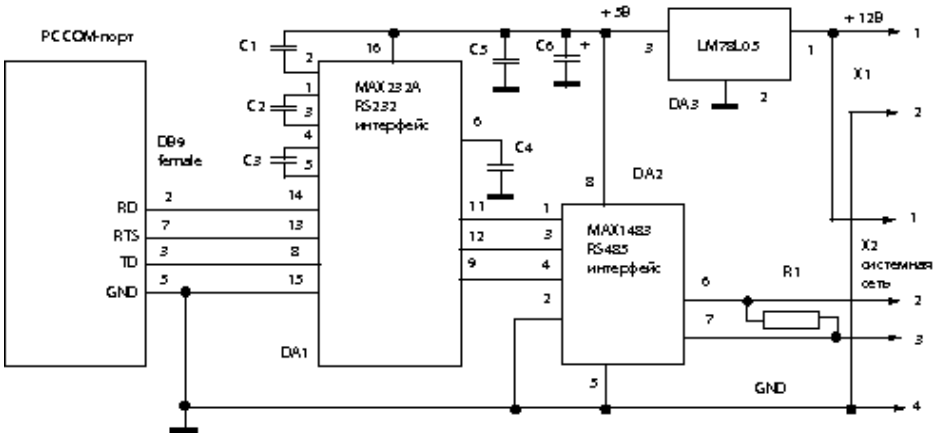


Рис. 4.3. Схема конвертера RS232-RS485

Проверка состояния вывода RTS (13 вывод микросхемы MAX232) с помощью мультиметра показывает, что при первом нажатии интерфейсной клавиши **Управление портом** напряжение на этом выводе меняется (на напряжение противоположного знака), а второе нажатие возвращает его к первоначальному виду. Похоже, обращение к порту есть. Но попытка отправить что-то функцией PRINT не дает видимых изменений на выводе 8 микросхемы MAX232. Хуже того, программа, как выяснилось, виснет, не позволяя закрыть порт повторным нажатием клавиши **Управление портом**. И что это? Кого ждем?

Видимо, ждем закрытия порта, поскольку, если подождать подольше, то порт

закрывается. Почему так долго?

И это не единственное «почему». Подключив осциллограф к выводу 8 микросхемы MAX232A, где я ожидал бы увидеть передаваемые с компьютера данные, я их не вижу. Работая над книгой «Умный дом», я в начальный момент в схожей ситуации использовал программу тестирования COM-порта, позже использовал собственную программу, написанную в среде Visual Basic. Но эти возможности касались работы в Windows, тогда как сейчас я использую Linux. Думаю, программы для работы с COM-портом есть и для платфомы Linux, но быстро найти хотя бы одну из них не получается. Тут я вспоминаю, что был еще вариант программы управления модулями «умного дома», написанный в среде KDevelop из дистрибутива Linux.

Я запускаю KDevelop, и чтобы не «изобретать велосипед», просто пытаюсь проделать все то, что описал в соответствующей главе собственной книги. Делаю при этом несколько поспешных ошибок, делаю вывод о том, что описание не мешало бы сделать более подробным (сам виноват, тогда казалось, что этим обидишь читателя), поскольку далеко не всегда понимаю, что следует сделать, отыскиваю в Интернете нужный для этих целей пакет, дополняющий классы необходимыми для работы с последовательным портом, и... застреваю на компиляции после попытки добавить эти классы. Вот так. Упростил себе задачу!

Мне казалось, что я быстро смогу повторить сделанную некогда программу, упростив ее до тестового варианта. Мне не много нужно – открыть и закрыть порт, да отправить в порт символ «a», правда в виде строки вида «aaaaaaaaaaaaaaaaaaaaaa». Одиночный символ наблюдать на экране осциллографа достаточно сложно, проще увидеть наличие «жизни» на выводах микросхемы при передаче нескольких символов. Но у меня до этого наблюдения дело не доходит. Я застреваю на компиляции файла из-за того, что добавленные классы работы с портом обращаются к библиотеке Qt, есть такая в Linux, и не могут с ней разобраться даже с помощью собственного описания, хотя твердо помню, что все это проделывал года два назад, все работало, была даже программа, прекрасно работавшая с той же макетной платой, что я использую сейчас. Если она была, ее надо найти.

Отыскать что-либо на моем компьютере по прошествии двух лет задача не для слабонервных. Каждый раз, сталкиваясь с этим, я даю себе слово навести порядок, но попытки навести порядок, боюсь, предвносят не меньше беспорядка, чем создается его в «плановом порядке». Все, после очередной переустановки операционной системы наведу полный порядок.

Нужная мне программа после получаса поисков находится на CD-диске. Ее компиляция не вызывает особых проблем. Остается выяснить, в чем разница?

Она только в версии добавленных мною файлов. Я скачал из Интернета последнюю версию, а компиляция проходит с более ранней. Самое простое, что я могу придумать – скопировать нужные мне файлы работы с последовательным портом из старого проекта в новый. Это заметно улучшает процесс «оживления» программы, приходится только закоментировать несколько строк, относящихся к

проверке использования пакета для Linux или Windows. Теперь у меня проходит компиляция и нового проекта. Остается поправить несколько строк кода, чтобы вернуться к попытке увидеть данные на выходе COM-порта.

Попытка быстро проверить работу новой программы с помощью некогда работавшей казалась мне удачной находкой, правильным и быстрым решением. Я ошибался. Но теперь мне хочется довести это «лирическое отступление» до логического завершения, тем более, что пока я не вижу другого пути. Мне нужно убедиться в том, что COM-порт работает, что конвертер и макетная плата работают, поскольку за прошедшие годы всякое могло случиться. Пора проверить все.

Итак. Схему конвертера я привел выше. Разъем X2 на этой схеме передает напряжение питания 12 В (выводы 1 и 4) и сигнал в линию RS485 (выводы 2 и 3).

На макетной плате расположена микросхема интерфейса RS485 и панелька для микроконтроллера PIC16F628A.

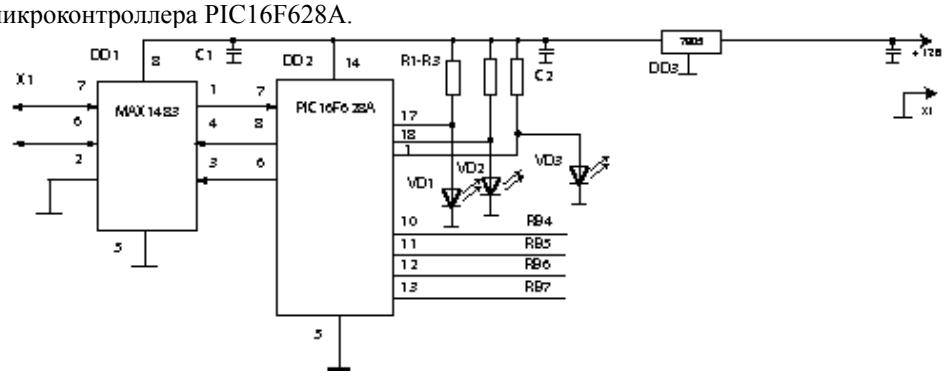


Рис. 4.4. Схема макетной платы

Используемые мною микросхемы MAX1483 интерфейса RS485 имеют штатное, практически, включение, совпадающее с включением аналогичной микросхемы MAX3082.

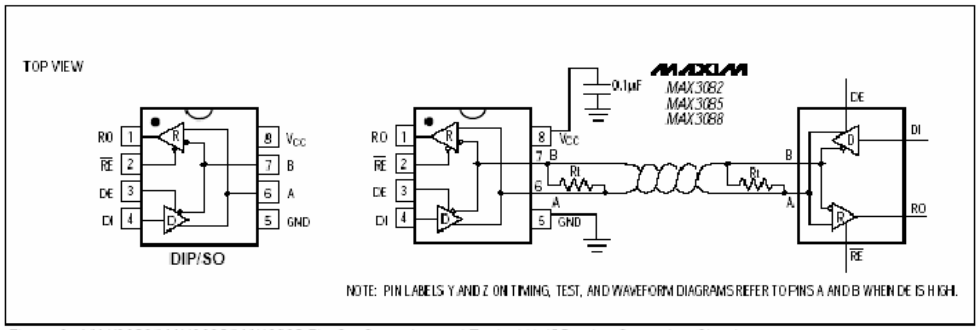


Figure 3. MAX3082/MAX3085/MAX3088 Pin Configuration and Typical Half-Duplex Operating Circuit

Рис. 4.5. Штатное включение микросхемы MAX1483

И я хочу проверить работу порта, наблюдая сигналы на его выходе, на выходе микросхемы MAX232A, в линии RS485, на выходе интерфейсной микросхемы макетной платы. Для этого я хочу из некогда работавшей программы проекта, написанного на C++ в среде KDevelop, сделать простую программу работы с COM-портом.

Лирическое отступление

Я не планировал обращаться к среде программирования KDevelop. Обычно не принято говорить с людьми о своих знакомых, которых твой собеседник не знает, и мне остается только познакомить (бегло и наспех) вас с моим «шапочным» знакомым. То что мое знакомство «шапочное» характеризует не с лучшей стороны меня, но никак не среду программирования KDevelop. Очень хорошая и удобная, с моей точки зрения, среда программирования на множестве языков.

Сразу после запуска программы, если вы не создали еще ни одного проекта, окно редактирования KDevelop пусто, хотя рабочее окно имеет помимо основного меню еще множество закладок. Выбрав в основном меню раздел Проект, с помощью подменю Новый проект... вы попадаете в диалог создания нового проекта. Если установить опцию (чуть ниже окна дерева выбора языка программирования слева) «Показать все шаблоны проекта», то вы можете подивиться тому обилию языков программирования, которыми можете воспользоваться для своих целей. Но меня интересует только быстрое повторение старого проекта, который был сделан как простейшее приложение KDE на C++. Этот вариант я и выбираю.

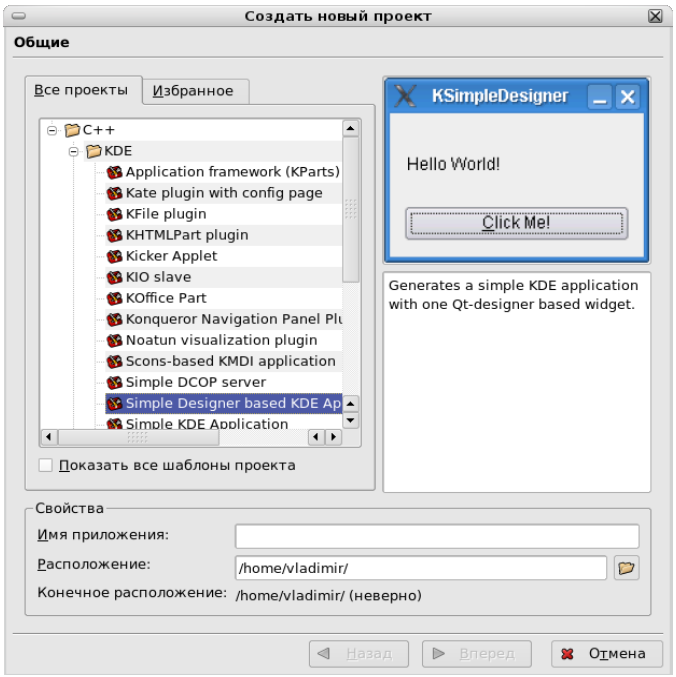


Рис. 4.6. Окно диалога выбора языкового шаблона в KDevelop

После ввода имени проекта (я назвал его counter_test) безучастная пока клавиша **Вперед** оживает, можно двигаться дальше. В этой версии необходимо ввести имя автора на следующей странице диалога, и можно дополнительно, если вы знаете как этим воспользоваться в дальнейшем, задать управление версиями и подправить шаблон файла заголовков проекта. После завершения всех подготовительных процедур и нажатия на клавишу **Готово** можно приступать к работе над проектом.

В качестве заготовок (шаблонов) у меня есть основной файл программы, файл заголовков, файл графического интерфейса, который можно исправить. Но прежде, чем править интерфейс, следует разобраться с добавлением нужного мне класса (или классов), требуемого для работы с последовательным портом. Для этого можно воспользоваться либо клавишей инструментального меню (крайней слева), подсказка которой предлагает «Генерировать новый класс», либо в разделе основного меню Проект выбрать пункт Новый класс. Эту операцию я проделываю трижды, для каждого из трех файлов пакета: Posix_QextSerialPort, QextSerialBase и QextSerialPort.

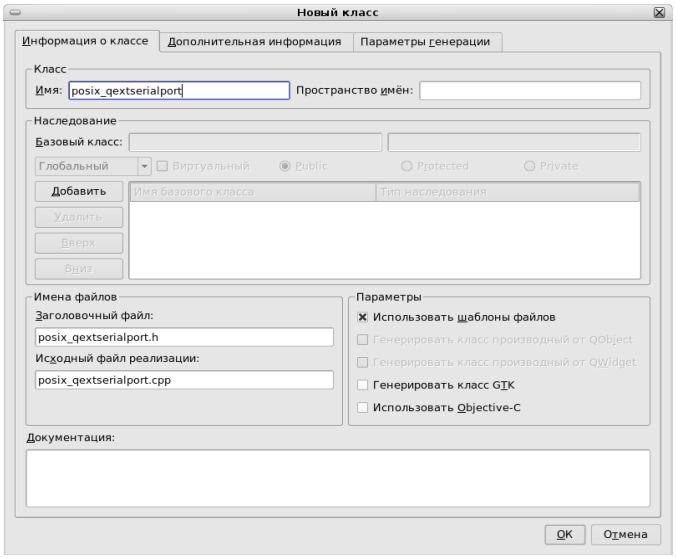


Рис. 4.7. Диалог создания нового класса в KDevelop

После нажатия на клавишу **ОК** появляется диалоговое окно, в котором можно выбрать, будет ли созданный класс добавлен к проекту. Я соглашаюсь добавить его к проекту. После трехкратного повторения этих операций мой проект пополняется тремя классами. Теперь я меняю созданные файлы (программы и заголовка) теми

файлами, что есть в пакете (старой версии). Просто перезаписываю их. Кроме этого в параметрах, по старой памяти, в разделе Поддержка C++ на странице Авто-дополнение кода, там где рядом с окном баз данных для авто-дополнения кода я нажимаю клавишу **Добавить**, чтобы добавить Qt.

Теперь в разделе Сборка нажимаем пункт Запустить automake и др., когда процесс успешно завершен, запускам в том же разделе пункт Запустить configure, а следом Собрать проект.

После первой попытки компиляции я подправляю файл qextserialport.h:

```
/*POSIX CODE*/
//#ifdef _TTY_POSIX_
#include "posix_qextserialport.h"
#define QextBaseType Posix_QextSerialPort

/*MS WINDOWS CODE*/
//#else
//#include "win_qextserialport.h"
//#define QextBaseType Win_QextSerialPort
//#endif
```

Правка свелась к удалению нескольких строк (я их закомментировал). Теперь компиляция проекта стала проходить без ошибок.

Далее я намерен переделать шаблон графического интерфейса. Мне нужны три кнопки: открыть порт – закрыть порт, запись и еще одна, которую я называл счет, но к счету она не будет иметь отношения. Прежде, чем мне удалось внести эти исправления, пришлось использовать подменю, которое появляется после щелчка правой клавиши мышки по форме, где я использовал попеременно пункты Разорвать расположение и Расположить по сетке, сохраняя сделанные изменения.

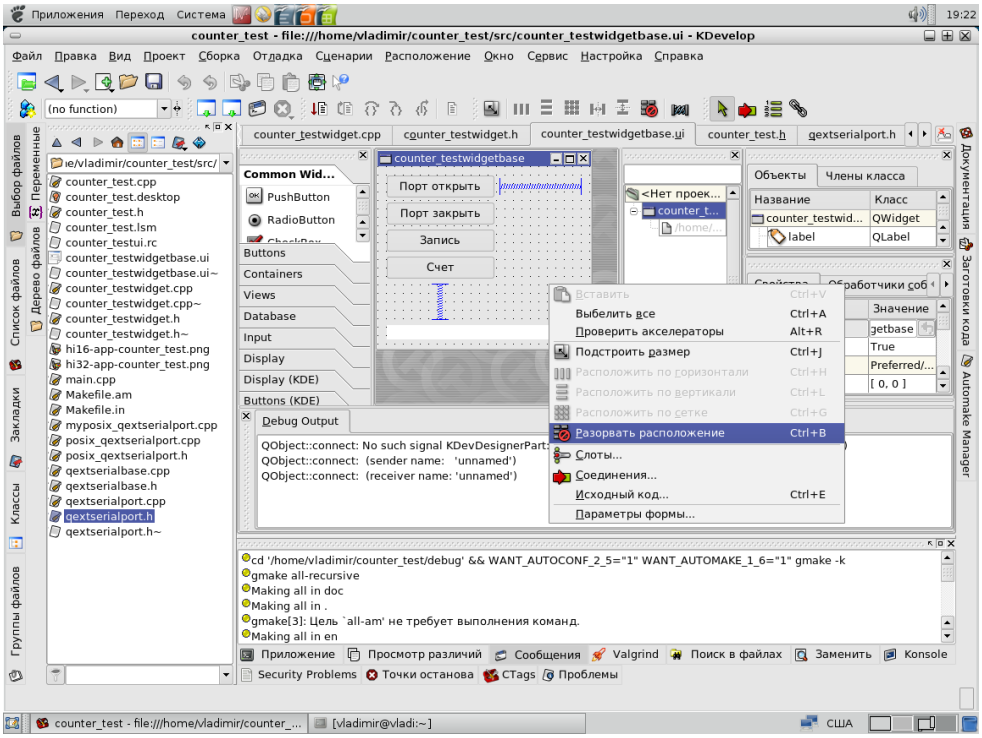


Рис. 4.8. Переделки графического интерфейса

После переделок интерфейса в разделе выпадающего меню Слоты... я добавляю слоты по аналогии с уже существующим, используя клавишу **Добавить функцию**, а в разделе Соединения... добавляю соединения с помощью клавиши **Новое**. Мне остается добавить в файл counter_testwidget.h:

```
public slots:
    /*$PUBLIC_SLOTS$*/
    virtual void button1_clicked();
    virtual void button2_clicked();
    virtual void button3_clicked();
    virtual void button4_clicked();
```

несколько строк (я повторяю имеющуюся запись virtual void button1_clicked(), где добавил только 1 после имени button). Аналогично я поступаю с файлом counter_testwidget.cpp:

```
void counter_testWidget::button1_clicked()
```

```
{
    if ( label->text().isEmpty() )
    {
        label->setText( "Hello World!" );
    }
    else
    {
        label->clear();
    }
}

void counter_testWidget::button2_clicked()
{
}

void counter_testWidget::button3_clicked()
{
}

void counter_testWidget::button4_clicked()
{
}
```

Где к развернутой записи о button1 (здесь я тоже добавил 1) я, скопировав, естественно, добавил пустые пока записи об остальных клавишах интерфейса. Именно в этот файл, опять путем копирования, я собираюсь перенести все необходимые мне события. В результате этих манипуляций файл counter_testwidget.cpp выглядит так:

```
#include <qlabel.h>

#include "counter_testwidget.h"
#include "posix_qextserialport.h"
#include "qextserialport.h"
#include "qextserialbase.h"

Posix_QextSerialPort comPort("/dev/ttyS0");

counter_testWidget::counter_testWidget(QWidget* parent, const char* name,
WFlags fl)
: counter_testWidgetBase(parent,name,fl)
{}

counter_testWidget::~counter_testWidget()
{}

/*$SPECIALIZATION$*/
void counter_testWidget::button1_clicked()
{
        comPort.setName("/dev/ttyS0");
        comPort.setDataBits(DATA_8);
        comPort.setStopBits (STOP_1);
```

```
        comPort.setBaudRate (BAUD2400);
        comPort.open(0);
        comPort.setRts (FALSE);
        if (comPort.isOpen())
{
        label->setText( "port open" );
}
}

void counter_testWidget::button2_clicked()
{
        comPort.close();
        if (!comPort.isOpen())
        label->setText( "port close" );
}

void counter_testWidget::button3_clicked()
{
        comPort.writeBlock("a", 1);
}

void counter_testWidget::button4_clicked()
{
        comPort.writeBlock("b", 1);
}


#include "counter_testwidget.moc"
```

Я не знаю, все ли правильно, но при сборке проекта претензий ко мне нет. К этому моменту я увидел две ошибки, допущенные при настройке порта в Gambas: скорость я задавал 9600, и бит RTS не сбрасывал. Есть надежда, что это и есть источник проблем. Запускаем программу... и...

«Но программа работала!» – говорю я.

«Не умеешь, не берись!» – отвечает внутренний голос.

Сей веселый диалог единственный результат эксперимента. Осциллограф не показывает наличие изменений на выводе передачи порта, микроконтроллер никак не реагирует на команды. А я точно помню, что программа работала, сейчас же она ведет себя точно так же, как ее Gambas-аналог. Даже «подвисание» при закрывании порта имеет место. Может быть, прав внутренний голос? Ну, нет! Внутренний голос может принадлежать моей лени, но никак не моему упрямству.

Вспоминая, как я некогда пытался устроить диалог между программой и микроконтроллером, я готов повторить еще одну проверку. В прошлый раз я отыскал в Интернете программу, которая позволяла проверить работу COM-порта. Эта программа проработала дней 15, а затем перестала работать, не захотела работала и после переустановки. Мне хватило 15и дней для продолжения работы, более того, позже я нашел в своем архиве еще одну программу, которая была получена с оборудованием и предназначалась для его настройки. Эту программу я хочу отыскать.

И отыскиваю, хотя с таким количеством ворчания, что его хватило бы на месяц работы. Программа предназначена для Windows, но охотно работает под эмулятором Wine, не требует установки, и окончательно ставит меня в тупик. У меня такое ощущение, что я что-то упустил между запуском программы в Gambas и в KDevelop. Мелькнули какие-то соображения, которые я тут же забыл. У программы есть окошко, в которое можно ввести символ управления, напомню, для зажигания светодиода я использую символ «a», для гашения – «b». Выключив клавишу RTS в программе, есть у нее такая возможность, я наблюдаю, как зажигается и гаснет светодиод на макетной плате под управлением микроконтроллера, подчиняющегося отправляемым мною символам. Даже осциллограф включать не пришлось, все работает. Обидно.

С другой стороны, это лучше, чем испорченный COM-порт. Хоть это утешает.

Я еще раз просматриваю настройки порта в программе, не нахожу отличий: 2400, 8 бит данных, 1 стоповый бит, нет проверки на четность. Все, как я прописываю в тестовой программе в KDevelop, и те же настройки, что я задавал в программе на Gambas.

Расстроенный и разочарованный, сердитый на себя и на весь мир, я запускаю программу на Gambas, чтобы наблюдать еще одно любопытное явление: после открывания порта, отправки команды «зажечь светодиод (запись)», если я закрываю порт, то светодиод зажигается. Я могу погасить его манипулируя клавишей «Управление портом» и клавишей гашения (счет).

Но время позднее. Я выключаю компьютер до завтрашнего дня, пусть он будет добрее ко мне!..

Будет. Но не этот завтрашний день. Опять программа на Gambas не работает даже с уговорами через открывание и закрывание порта, программа в KDevelop с ней солидарна. Появился еще один повод для размышлений, не пора ли прекратить все: Linux мало кого интересует, радиолюбители больше интересуются справочными данными и конкретными указаниями, какие настройки нужно сделать, чтобы схема заработала. А дилетантские рассуждения о программировании, кому это надо?

Похоже, пора. Попробую еще раз с самого начала, а там...

Первой я запускаю Windows-программу для работы с COM-портом, затем программу в Gambas.

К загадке, почему не работает ранее работавшая программа, опять добавляется загадка странного поведения недавно написанной программы, поскольку с помощью странного манипулирования открыванием и закрыванием порта я вновь могу зажигать и гасить светодиод на макетной плате.

Вот. Я вспомнил, что хотел проверить. При работе модема нужно то ли включать, то ли отключать контроль потока, программный или аппаратный, уж и не помню. Переместившись в программу, написанную в среде KDevelop, я нахожу, как отключить этот контроль:

```
comPort.setFlowControl(FLOW_OFF);
```

Добавленная к настройкам порта при его открывании эта строка устраняет все странности. Все работает. Думаю, в прошлый раз мне просто повезло. Видимо, при включении компьютера порт получал настройки по умолчанию, включающие этот параметр. Мне не пришлось его задавать специально.

Теперь я могу вернуться в Gambas, где в свойствах последовательного порта есть и этот параметр. Мне даже не приходится добавлять код в программу.

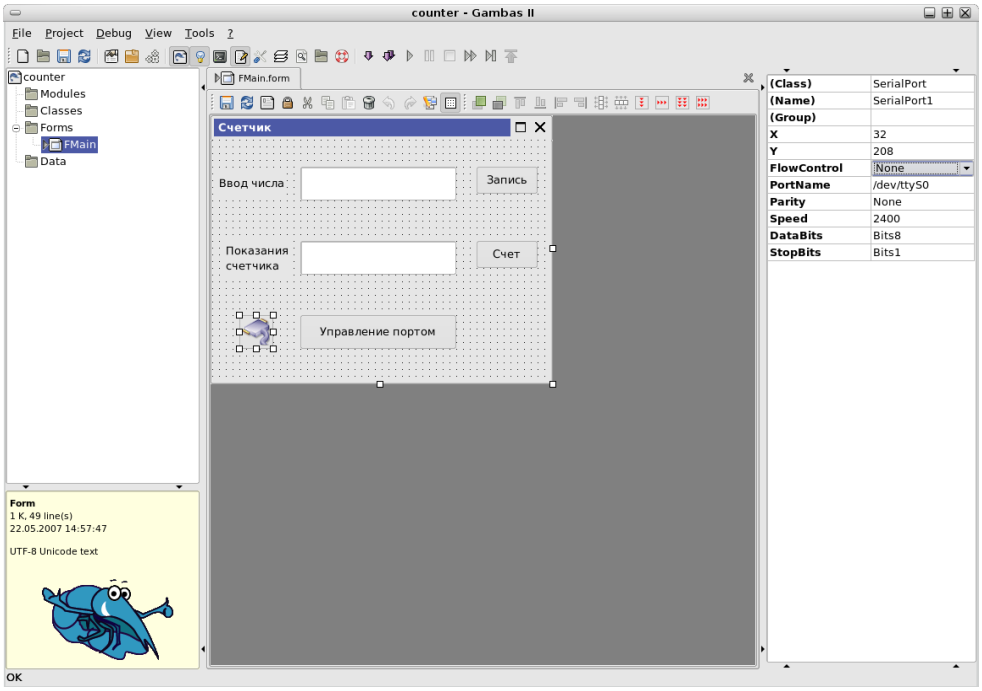


Рис. 4.9. Задание параметра контроля потока порта в Gambas

Включив порт клавишей **Управление портом**, щелкая клавишами Запись и **Счет,** я могу включать и выключать светодиод, как и планировалось изначально. Все-таки день был добрее. И вывод – не следует полагаться на удачу, следует познакомиться с работой оборудования, которое ты предполагаешь использовать, и разобраться с его параметрами. И задать все параметры без исключения, даже если в прошлый раз повезло, и кто-то сделал это за тебя. Надейся на себя. Так будет вернее.

Грустное завершение рассказа о счетчике

Пришла пора расстаться с проектом проверки счетчика. Я не собирался его выполнять полностью и проверять на макетной плате. Однако, чтобы не испытывать угрызений совести, я использую существующую макетную плату, на которой есть три светодиода (как на рисунке 4.4), подключенных к выводам порта A: RA0, RA1 и RA2. Выводы RA3, RA4 и RA5 имеют «подтягивающие» резисторы (их нет на рисунке, как и выводов). Под эти возможности готового макета я переделаю обе программы, интерфейса и микроконтроллера.

По команде «Запись» от графического интерфейса микроконтроллер должен переписать заданное число, три бита позволят мне использовать числа от 0 до 7, в три младших бита порта A. А по команде «Счет» прочитать состояние следующих трех выводов порта A, сделаем вид, что они соединены с выходами счетчика, и отправлять их состояние в символьном виде обратно. Таким образом микроконтроллер получает две команды, я оставлю их прежними, хотя легко можно заменить их любыми подходящими символами или набором символов. После первой команды контроллер получает число в символьном виде, которое должен обработать для отображения светодиодными индикаторами. Я могу соединять выводы RA0 и RA3, RA1 и RA4, и т.д., но могу и просто подключать входные выводы к общему проводу, как мне заблагорассудится.

Начать писать код программы можно с интерфейсной части проекта в Gamabs, а можно вначале запрограммировать микроконтроллер в программе Piklab. Пусть будет Piklab.

Начну с того, что подправлю файл заголовка:

```
#define bitset(var,bitno) ((var) |= 1 << (bitno)) // Установка бит порта
#define bitclr(var,bitno) ((var) &= ~(1 << (bitno))) // Сброс бит порта
unsigned char getch(); // Прием символа
void putch(unsigned char); // Передача символа
void     init_comms(); // Инициализация коммуникации
void cmd();
```

Правка касается пока только добавления функции передачи символа, с помощью которой можно будет отправлять с контроллера информацию о состоянии выводов порта A, которые якобы соединены с выходом счетчика.

В основном тексте программы сразу следует изменить строку в функции void init_comms(). Именно в этой функции задается конфигурация порта A, где прежде все выводы были направлены на выход. Теперь три из них следует перенаправить на вход.

```
TRISA = 0x38;
```

Не всегда я так поступаю, что неправильно, но на этот раз я проверю изменилось ли состояние выводов контроллера с помощью программы gpsim. После запуска симуляции я могу убедиться, что установка выводов порта A соответствует желаемому.

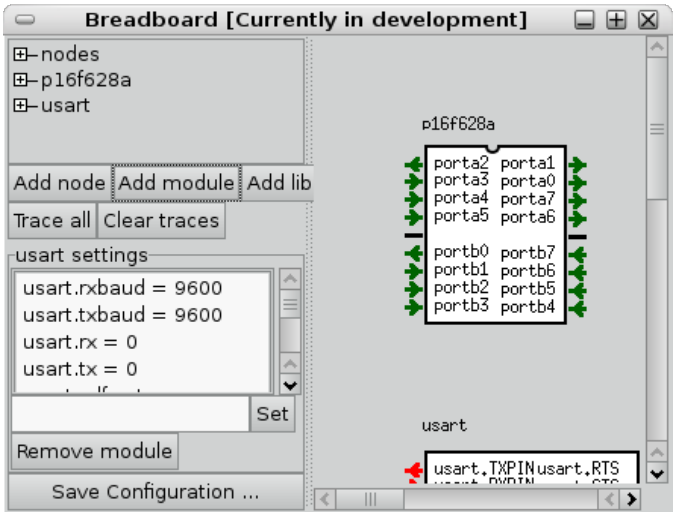


Рис. 4.10. Результат конфигурации выводов порта A в программе gpsim

На виртуальной макетной плате проекта, где стрелки выводов портов показывают, как они будут работать, видно, что выводы porta3 – porta5 должны работать на ввод.

В файл заголовка я добавлю еще одну переменную и две функции, отвечающие командам интерфейсной части проекта «Запись» и «Счет»:

```
unsigned char NUMBER;
void cmd_write ();
void cmd_count ();
```

А сами функции определю в основном файле следующим образом, но последовательно проверяя их работу:

```
void cmd_write ()
{
        NUMBER = input;

        switch (NUMBER) {
                case '0':
                bitclr (PORTA, 0);
```

```
                bitclr (PORTA, 1);
                bitclr (PORTA, 2);
                break;
                case '1':
                bitset (PORTA, 0);
                bitclr (PORTA, 1);
                bitclr (PORTA, 2);
                break;
                case '2':
                bitclr (PORTA, 0);
                bitset (PORTA, 1);
                bitclr (PORTA, 2);
                break;
                case '3':
                bitset (PORTA, 0);
                bitset (PORTA, 1);
                bitclr (PORTA, 2);
                break;
                case '4':
                bitclr (PORTA, 0);
                bitclr (PORTA, 1);
                bitset (PORTA, 2);
                break;
                case '5':
                bitset (PORTA, 0);
                bitclr (PORTA, 1);
                bitset (PORTA, 2);
                break;
                case '6':
                bitclr (PORTA, 0);
                bitset (PORTA, 1);
                bitset (PORTA, 2);
                break;
                case '7':
                bitset (PORTA, 0);
                bitset (PORTA, 1);
                bitset (PORTA, 2);
                break;
                default: break;
                }
}
```

Сразу внесем изменение в функцию приема команды:

```
void cmd() // Получение и выполнение команды
{
        COMMAND = input;
        switch (COMMAND) {
                case 'a':
                input = getch();
                cmd_write ();
                break;
        }
```

```
}
```

И проверим, работает ли эта часть программы. О пользе проверки программы после даже небольших изменений я выше говорил, а сейчас готов подтвердить это личным примером. Первая же трансляция заставляет меня искать ошибки – опечатка, которую я умудрился добавить в код программы. И возникает вторая проблема, как проверить правильность кода. Логично это сделать в программе симуляции gpsim, но отношения с gpsim в части использования USART у меня как-то не сложились.

На всякий случай я еще раз перепроверяю эти отношения, но увы мне! Единственное, что остается, использовать макетную плату для непосредственной проверки. Запрограммировав микроконтроллер в программе Piklab, водрузив контроллер в панельку макетной платы, я подключаю ее к COM-порту и запускаю интерфейс в Gambas. Основные команды, в которых я использую отправку символов «a» и «b», продолжают работать, но попытка ввести в окно ввода числа с тем, чтобы светодиоды отобразили результат этого нововведения, оканчивается неудачей. Скорее всего, мое видение программы, написанной для микроконтроллера, и его видение этой программы разнятся слишком сильно. Это же может касаться и интерфейсной части.

Попробуем разобраться. Вначале с интерфейсом в Gambas. Здесь у меня только одно сомнение, правильно ли я пытаюсь прочитать ввод в окне «Ввод числа» и передать его в COM-порт. Немного повозившись с разными вариантами проверки этого, я нахожу правильное решение – если команда с помощью отправки символа a:

```
PRINT #SerialPort1 "a"
```

отрабатывается исправно, то я могу, удалив эту строку из текста программы, использовать ввод не числа, а этого символа в окно ввода числа. Тогда следующая строка:

```
PRINT #SerialPort1 TextBox1.Text
```

отправит команду после нажатия на клавишу «Запись». Так ли это? Я проверяю, и убеждаюсь, что это так. Осталось восстановить текст интерфейсной части проекта и считать, что с этой частью в настоящий момент все в порядке.

Теперь стоит подумать, как проверить программу микроконтроллера? Здесь в первую очередь мне хотелось бы понять, попадаю ли я в функцию cmd_write (). Почти вся функция должна проверяться работой светодиодов, а то, что они не горят, может свидетельствовать как о том, что в функцию я не попадаю, так и о том, что я миную переключатель switch (NUMBER) в этой функции. Чтобы проверить, я добавлю в конец функции проверку:

```
void cmd_write () {

        NUMBER = input;
        switch (NUMBER) {
```

```
                case '0':
                bitclr (PORTA, 0);
                bitclr (PORTA, 1);
                bitclr (PORTA, 2);
                break;
                ..........................................
                default:
                bitset (PORTA, 2); // Проверка
                break;
                }
}
```

То есть, если я не могу отработать значение числа, но в функцию обработки попадаю, светодиод на выводе RA2 будет загораться. Программируем контроллер. Переносим на макетную плату. Убеждаемся, что светодиод горит. В функцию записи я попадаю, но переменная NUMBER не отвечает ни одному из перечисленных случаев. По прошлому опыту работы с функцией получения символа getch(), я достаточно много провозился с ней, я могу предположить, что не в полной мере понял ее работу. Ничто не мешает сделать еще одну попытку понять работу этой функции, но сперва попытаюсь перемещать ее с места на место, может быть, получится как-то обойти проблему.

В одном из вариантов, размышляя о собственном бессилии и пытаясь придумать нечто полезное, я перебираю все числа, которые следует ввести для передачи, отправляя их через COM-порт в микроконтроллер. Неожиданно для себя я обнаруживаю реакцию контроллера на эти числа. Они не отрабатываются должным образом, но контроллер реагирует на них. По этой причине мое внимание привлекает конструкция кода:

```
void cmd_write () {

        NUMBER = input;

        switch (NUMBER) {
                ..................
                case '1':
                bitset (PORTA, 0);
                bitclr (PORTA, 1);
                bitclr (PORTA, 2);
                break;
                ...................
                }
}
```

В моем понимании эта конструкция устанавливает вывод RA0 в единицу, а следующие два вывода порта в ноль. Но, насколько я помню, это может быть и не так. Программа может последовательно перебирать заданные команды, каждый раз при установке или сбросе бита очищая все остальные. То есть в показанном случае сброс вывода RA2 окажется последней командой, которая сбросит и RA1, и RA0. Были у

меня и прежде сомнения относительно использования функции побитовой установки и сброса, для проверки меняю эту конструкцию на другую:

```
void cmd_write () {

        NUMBER = getch();

        switch (NUMBER) {
                ..............................
                case '1':
                PORTA = 0x1;
                break;
                ..............................
                }
}
```

и это срабатывает. Теперь основная программа для микроконтроллера выглядит следующим образом:

```
/* ------------------------------------------------------------------------ */
/* Template source file generated by piklab */
#include <pic16f628a.h>

/* ------------------------------------------------------------------------ */
/* Configuration bits: adapt to your setup and needs */

typedef unsigned int word;
word at 0x2007 CONFIG = _WDT_OFF & _PWRTE_OFF & _INTOSC_OSC_NOCLKOUT &
_MCLRE_OFF & _BOREN_OFF & _LVP_OFF & _DATA_CP_OFF & _CP_OFF;

#include <stdio.h>
#include "counter_start.h"

unsigned char input;  // Для считывания приемного регистра
unsigned char COMMAND;// символ команды
unsigned char NUMBER;

unsigned char getch() // Получение байта
{
        while(!RCIF)   // Устанавливается, когда регистр не пуст
                continue;
        return RCREG;
}

void cmd() // Получение и выполнение команды
{
        COMMAND = input;

        switch (COMMAND) {
                case 'a':
                getch();
                cmd_write ();
```

```
                break;
                case 'b': bitset (PORTA, 1);
                break;
                default: break;
                }
}

void cmd_write () {

        NUMBER = getch();

        switch (NUMBER) {
                case '0':
                PORTA = 0x0;
                break;
                case '1':
                PORTA = 0x1;
                break;
                case '2':
                PORTA = 0x2;
                break;
                case '3':
                PORTA = 0x3;
                break;
                case '4':
                PORTA = 0x4;
                break;
                case '5':
                PORTA = 0x5;
                break;
                case '6':
                PORTA = 0x6;
                break;
                case '7':
                PORTA = 0x7;
                break;
                default:
                break;
                }
}

void cmd_count ()
{

}

void init_comms()      // Инициализация модуля
{
        PORTA = 0x0;   // Настройка портов A и B
        CMCON = 0x7;
        TRISA = 0x38;
        TRISB = 0xFE;

        RCSTA = 0x90;  // Настройка приемника
```

```
        TXSTA = 0x6;   // Настройка передатчика
        SPBRG = 0x68;  // Настройка режима приема-передачи
        bitclr (PORTB, 0);    // Выключаем драйвер RS485 на передачу
}

void main(void) {
        init_comms(); // Инициализация модуля
start:  CREN =1; // Начинаем работать
        //getch();
        input = getch();
        cmd();
        goto start;
}
```

Светодиоды зажигаются соответственно вводимому в окно «Ввод числа» числу.

Остается реализовать и отладить ту часть программы микроконтроллера, которая будет передавать состояние вводов от микроконтроллера к интерфейсу. А код программы интерфейса дополнить чтением из COM-порта того, что отправил микроконтроллер.

В интерфейсной части проекта в Gambas я добавляю одну строку:

```
PUBLIC SUB Button2_Click()
  PRINT #SerialPort1 "b"
  INPUT #SerialPort1 TextBox2.Text
END
```

До переделки кода программы микроконтроллера интерфейс выводит в окне приема то, что я отправляю в контроллер.

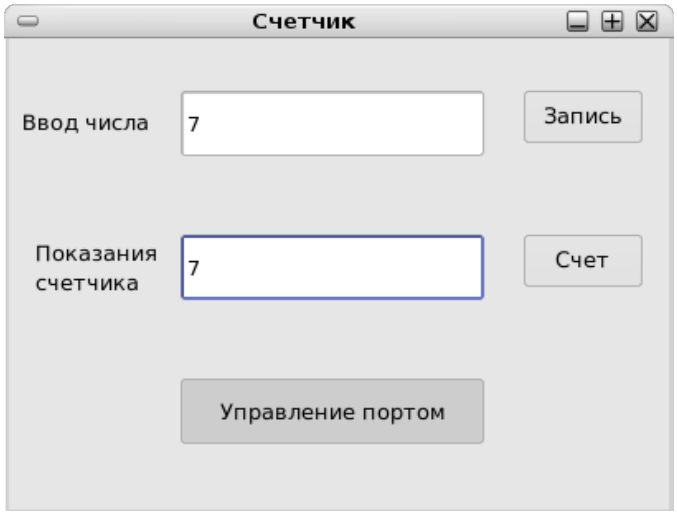


Рис. 4.11. Получение символов от COM-порта в интерфейсной части проекта

Посмотрим, что будет после переделки, с которой, как я подозреваю, тоже не все будет гладко.

В основной текст программы для микроконтроллера я добавляю функцию отправки символа:

```
void putch(unsigned char byte) // Отправка байта
{
        while(!TXIF)   // Устанавливается, когда регистр не пуст
                continue;
        TXREG = byte;
}
```

не забыв объявить ее в файле заголовка.

В функцию приема команд я добавляю отправку состояния порта, добавив переменную unsigned char COUNTER в начало программы:

```
void cmd() // Получение и выполнение команды
{
        COMMAND = input;

        switch (COMMAND) {
                case 'a':
                getch();
                cmd_write ();
                break;
                case 'b': cmd_count();
                break;
                default: break;
                }
}
```

И, конечно, заполняю функцию отправки значения вводов порта:

```
void cmd_count ()
{
        COUNTER = PORTA;
        bitclr (PORTA, 0); // Проверка
        putch(COUNTER);
}
```

Строку проверки, признаюсь, я добавил после первого неудачного опыта. Последовательность эксперимента такова – я вписываю число 7, нажимаю клавишу **Запись**, затем нажимаю клавишу **Счет**, ожидая получить нечто, соответствующее моим ожиданиям. Поскольку я записываю семерку, то все светодиоды должны загораться. Если я попадаю в функцию отправки микроконтроллером символа, то один из светодиодов должен погаснуть. Проверяю.

Гаснут все светодиоды. Что напоминает мне о двух вещах: не следует наступать на

грабли, о которых только что говорил (использование конструкции bitclr (PORTA, 0)), и второе – не мешало бы разрешить передачу!

Я вношу исправления, копируя необходимые части кода программы из прежнего проекта, и в итоге перестает работать все, даже то, что работало.

Приплыли!

Глава 5. Сказка о неудачливом радиолюбителе

Нет, нет, и нет. Не буду.

Это я уговариваю себя отказаться от полномасштабной проверки проекта «counter». То есть, впаять в макетную плату панельку для счетчика, запаять все микросхемы перехода с RS232 на RS485 и т.д. Иногда я скучаю по пайке. Тогда включаю паяльник и начинаю что-нибудь паять. Но сегодня, чтобы спаять что-нибудь нужное, мне надо поехать в «Чип и Дип» и купить панельки, микросхемы, да и резисторов не мешало бы прикупить, еще несколько микросхем операционных усилителей никак не соберусь купить, не забыть бы...

Нет, нет, и нет. Магазин возле Курского вокзала, в который мне удобно было добираться, закрылся, а ехать в другие края сегодня не хочется. Да и с проектом я как-то «завял» даже на уровне макетной проверки. Одно другому не мешает, можно спаять новую макетную плату, подготовив ее к окончательной проверке, затем вернуться к нерешенным проблемам. Но нет. Если не решить возникшие проблемы, если не разобраться в причине неудач, то покупка всего необходимого для новой макетной платы с последующей пайкой может оказаться пустой тратой денег.

Начнем все сначала.

Возвращение на круги своея

Я возвращаюсь в программу Piklab, где, пользуясь текстом программы, приведенным выше, когда у меня еще что-то работало, возвращаю программу микроконтроллера к ее раннему состоянию. Сейчас она выглядит так:

Файл заголовка counter_start.h

```
#define bitset(var,bitno) ((var) |= 1 << (bitno)) // Установка бит порта
#define bitclr(var,bitno) ((var) &= ~(1 << (bitno))) // Сброс бит порта
unsigned char getch(); // Прием символа
void init_comms(); // Инициализация коммуникации

void cmd();
void cmd_write ();
void cmd_count ();
```

Основной файл counter_start.c

```
/* ------------------------------------------------------------------- */
/* Template source file generated by piklab */
#include <pic16f628a.h>
```

```
/* ------------------------------------------------------------------- */
/* Configuration bits: adapt to your setup and needs */
typedef unsigned int word;
word at 0x2007 CONFIG = _WDT_OFF & _PWRTE_OFF & _INTOSC_OSC_NOCLKOUT &
_MCLRE_OFF & _BOREN_OFF & _LVP_OFF & _DATA_CP_OFF & _CP_OFF;

#include <stdio.h>
#include "counter_start.h"

unsigned char input;  // Для считывания приемного регистра
unsigned char COMMAND;// Символ команды
unsigned char NUMBER;         //  Значение для записи


unsigned char getch() // Получение байта
{
        while(!RCIF)   // Устанавливается, когда регистр не пуст
                continue;
        return RCREG;
}

void cmd() // Получение и выполнение команды
{
        COMMAND = input;

        switch (COMMAND) {
                case 'a':
                cmd_write ();
                break;
                case 'b':
                PORTA = 0x0;
                break;
                default:
                break;
                }
}

void cmd_write () {

        NUMBER = getch();

        switch (NUMBER) {
                case '0':
                PORTA = 0x0;   // Все светодиоды погашены
                break;
                case '1':
                PORTA = 0x1;   // Горит первый светодиод
                break;
                case '2':
                PORTA = 0x2;   // Горит второй светодиод
                break;
                case '3':      // Горят первый и второй светодиоды
                PORTA = 0x3;
                break;
```

```
                case '4':
                PORTA = 0x4;
                break;
                case '5':
                PORTA = 0x5;
                break;
                case '6':
                PORTA = 0x6;
                break;
                case '7':
                PORTA = 0x7;
                break;
                default:
                break;
                }
}

void cmd_count ()
{

}

void init_comms()      // Инициализация модуля
{
        PORTA = 0x0;   // Настройка портов A и B
        CMCON = 0x7;
        TRISA = 0x38;
        TRISB = 0xFE;
        RCSTA = 0x90;  // Настройка приемника
        TXSTA = 0x6;   // Настройка передатчика
        SPBRG = 0x68;  // Настройка режима приема-передачи
        PORTB = 0xFE;  // Выключаем драйвер RS485 на передачу
}

void main(void) {
        init_comms(); // Инициализация модуля

start:  CREN =1; // Начинаем работать
        input = getch();
        cmd();
        goto start;
}
```

Компилируем этот проект, загружаем результат в микроконтроллер. Опять я использую готовую программу для работы с COM-портом: отправляю символ «a», он проходит, но на макетной плате нет видимых изменений, как и должно быть, впрочем. Я отправляю символ «7», все три светодиода зажигаются. Я отправляю символ «b», все светодиоды гаснут. На всякий случай я повторяю эти операции заменив 7 на 1. Все работает правильно. Правильно работает и отправка строк вида «a7», «a1» или «a0».

Возвращаюсь к интерфейсной части на Gambas, пытаясь добавляя и перемещать разные строки, закрывать их символами комментария, но добиваюсь только одного –

проект превращается в мешанину строк, в которых я сам перестаю что-либо понимать.

Самое разумное в этот момент, насколько я понимаю, удалить все лишнее и упростить все до предела. В первую очередь я хочу удалить клавишу **Управление портом**. Все, что нужно сделать для управления работой порта, можно сделать при нажатии двух оставшихся клавиш. Зачем усложнять себе жизнь? В итоге программа приобретает вид:

```
' Gambas class file

PUBLIC SUB _new()
END

PUBLIC SUB Form_Open()
END

PUBLIC SUB Button1_Click() ' Клавиша "Запись"
	SerialPort1.Open
	SerialPort1.RTS = FALSE
	PRINT #SerialPort1 TextBox1.Text
	SerialPort1.Close
END

PUBLIC SUB Button2_Click() ' Клавиша "Счет"
	PRINT #SerialPort1 "b"
END
```

Нужную мне строку вида «a7» я вписываю в окошко перед нажатием клавиши «Запись», и теперь эта часть программы работает, зажигая на макетной плате соответствующее команде количество светодиодов. Можно вернуться к считыванию состояния тех выводов контроллера, которые (как будто) подключены к выходам счетчика.

Для этого вернемся в программу Piklab, где произведем должные изменения в коде. Но прежде, чем это сделать, проверим одну часть кода, которая отвечает за команды:

```
void cmd() // Получение и выполнение команды
{
	COMMAND = input;

	switch (COMMAND) {
		case 'a':
		cmd_write (); // Выполнить принятую команду
		break;
		case 'b':
		PORTA = 0x0; // Погасить светодиоды
		break;
		default:
		break;
```

```
                }
}
```

То есть, отправляя команду «a7», я зажигаю все светодиоды, а после команды «b», которую я отправляю клавишей **Счет** интерфейса, они должны погаснуть.

Небольшие изменения в коде программы интерфейса (клавишу управления портом я удалил):

```
PUBLIC SUB Button2_Click() ' Клавиша "Счет"
        SerialPort1.Open
        SerialPort1.RTS = FALSE
        PRINT #SerialPort1 "b"
        SerialPort1.Close
END
```

Эта часть программы работает – ввод команды и нажатие клавиши **Запись** зажигает все три светодиода, нажатие клавиши **Счет** их гасит.

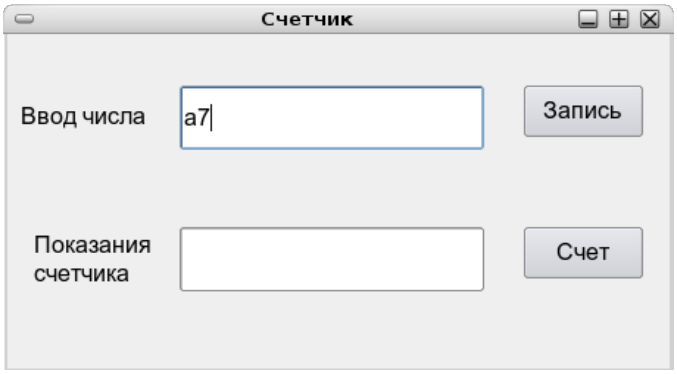


Рис. 5.1. Изменения в интерфейсной части проекта

Теперь можно попытать счастье с добавлением передачи состояния вводов.

Попытать можно, но будет ли оно, счастье? Пока ничего кроме «чудес в решете» я не наблюдаю. Во-первых, при подключении программы к одному COM-порту, а программатора ко второму, одновременно, программатор дает сбои. Хорошо, подключаемся по очереди, получается лучше. Но, стоит мне добавить в текст программы для микроконтроллера описание функции, которая передавала бы состояние порта B, как контроллер перестает работать. Код, с моей точки зрения, не содержит ничего особенного:

```
void cmd_count ()
{
        CREN =0; // Выключаем прием
```

```
        PORTB = 0xFF;  // Включаем передатчик
        TXEN = 1; // Включаем передачу
        COUNT = '7'; //PORTB;
        putch(COUNT);
        PORTB = 0xFE;  // Выключаем передатчик
        TXEN = 0; // Выключаем передачу
        CREN =1; // Включаем прием
}
```

И самое главное, я пока не использую эту функцию в программе. А если я удаляю ее, то контроллер, по крайней мере, правильно реагирует на команды. Если бы не видел это своими глазами (или руками, как угодно), то не поверил бы. В чем же фокус? Приходит в голову мысль о неудачных именах, выбранных мною для функций. Я исправляю эти имена, но это ничего не меняет. Все тот же «кошмарный сон».

Раздраженный неудачей, сердитый на себя, и на весь мир заодно, я возвращаю текст программы к прежнему виду, но злополучную функцию переношу в самый конец кода, ниже функции main (), что в нормальном состоянии не сделал бы. Однако теперь контроллер выполняет команды, которые я от него ожидаю. Еще один вариант – размещение функции передачи состояния порта cmd_count() прямо над функцией main(). Опять программирование контроллера, и вновь удачно. Контроллер выполняет команды управления светодиодами.

Если вы что-то в этом понимаете, то я рад за вас, лично я ничего не понимаю! Теперь пора бы заняться работой функции передачи состояния порта. На что и уходит достаточно много времени. Я уже не пытаюсь сразу передать от микроконтроллера состояние порта, я хочу получить ответ на команду в виде символа, например, «7». Выглядит эта часть кода программы микроконтроллера следующим образом:

```
void cmd() // Получение и выполнение команды
{
switch (input) {
        case 'a':
        PORTA = 0x0; // Погасить светодиоды
        break;
        case 'b':
        CREN = 0; // Выключаем прием
        PORTB = 0xFF;  // Включаем передатчик
        TXEN = 1; // Включаем передачу
        putch('7');
        PORTB = 0xFE;  // Выключаем передатчик
        TXEN = 0; // Выключаем передачу
        CREN = 1; // Включаем прием
        PORTA = 0x7; // Включаем все светодиоды
        break;
        default:
        break;
        }
}
```

Моя самоуверенность и ожидание быстрого успеха давно покинули меня, уступив место упрямству и раздражению, причина которого в том, что с этой программой микроконтроллера, в ее более сложном виде, я работал. И контроллер работал, и все работало. Но не в этот раз! Я даже не пытаюсь получить результат в интерфейсной части проекта, а использую программу для работы с COM-портом. Не самый удобный вариант, но ей я пока доверяю больше, чем своим творениям. Я ожидаю появления семерки в окне отображения вывода после отправки команды, привязанной к символу «b». Я упрощаю программу микроконтроллера, удалив все, что не относится к приему и выполнению в виде отсылки символа «7» этой команды.

Я еще раз просматриваю текст старой, некогда работавшей программы, которую, очень сильно упростив, я использовал. Замечаю, что в начальный момент в функции переключения при отправки ответа я использовал задержку перед выключением передатчика. В ходе экспериментов этот фрагмент я убрал. Восстанавливаю его.

```
switch (input) {
        ......
        PORTB = 0xFF;  // Включаем передатчик
        TXEN = 1; // Включаем передачу
        putch('7');
        for (i=0;i<1000;i++); // Пауза
        PORTB = 0xFE;  // Выключаем передатчик
        TXEN = 0; // Выключаем передачу
        ......
        }
}
```

Пытаюсь понять работу функции для работы с передатчиком USART, putch(byte). Возвращаюсь к чтению документации по контроллеру PIC16F628A в ее части работы с передатчиком. Пока читаю, все понятно. Стоит задуматься о прочитанном, ничего не понятно. Например, как работает флаг прерывания, напомню, как выглядит функция передачи символа,

```
void putch(unsigned char byte)
{
        while(!TXIF) // Отправка байта
                continue;
        TXREG = byte;
}
```

которую я не придумал, а честно «срисовал» когда-то; флаг прерывания TXIF, который устанавливается и снимается, даже если прерывание не используется, должен реагировать, по описанию, на появление отправляемого символа в регистре TXREG. Но я записываю туда что-то после опроса состояния флага. Добавляет к моему замешательству и то, что я, насколько это понимаю, «кручусь» в цикле while до тех пор, пока флаг сброшен, но он появляется только после переписи символа из

программно доступного мне регистра TXREG во внутренний регистр передатчика по фронту одного из импульсов, кажется стопового, предыдущего отправленного символа. Мне это все понятно в середине процесса, который, как некогда было модно говорить – «процесс пошел», существует, но мне не понятно, как он начинается, если отправляется первый символ.

Я совсем запутался. Я пытаюсь тупо переделать функцию putch(), удаляю и добавляю команды светодиодной индикации, чтобы понять, где застревает работа программы. И бросаю все.

У меня нет больше сил воевать с «ветряными мельницами». Хотя два полезных проблеска в этой мгле имели место: я понимаю, что скрытые от меня процессы, как перепись во внутренние регистры, никаким осциллографом не «отловишь», то есть, основная идея, использовать компьютер, правильна; и второе – при работе с программой COM-порта, как я вспоминаю, есть режим отображения в окне вывода символов ASCII, что я использовал, но можно отображать шестнадцатеричные значения, что я еще не пробовал. К моему удивлению, сменив режим отображения, после каждого цикла приема-передачи символа я вижу значение «00». Кстати и интерфейсная программа в моих попытках принять что-то от контроллера, постоянно донимала меня напоминаниями о «конце файла». Москит, который появляется при запуске Gambas, изображал ужас, затем крайнее отчаяние и показывал сообщение «Конец файла!». Я пытался этот конец файла куда-то пристраивать, но безнадежно, а теперь понимаю, что и пристраивать его не следовало, поскольку это «00» и есть, видимо, конец файла?

И еще одну полезную вещь я нахожу, когда в последний раз без надежды на успех возвращаюсь к интерфейсу на Gambas – в его многочисленных примерах есть пример работы с последовательным портом, который оказывается хорошей альтернативой программе работы с COM-портом для Windows. По этому случаю я решаю не бросать все, а еще немного помучиться.

Видимо, чтобы придать моим мучениям некоторую остроту, начинает давать сбои программатор. Он исправно работал все это время, а теперь дает ошибки записи в контроллер. Но у меня есть от этого средство. Не зря же я покупал программатор. Им пользоваться не так удобно, как самодельным, но у него автономное питание, есть некоторые дополнительные улучшения, на крайний случай есть программа, устойчиво работающая с ним. Меняю программатор, попутно проверяя очень меня удививший факт – появление ошибок при одновременной работе программы программатора и программы работы с COM-портом (на разных портах!). Ошибка остается. Но если выключить вторую программу, то программирование проходит успешно, при этом я пока использую программирование в программе Piklab, а не в отдельной программе. И на том «Спасибо!».

И спасибо моему упрямству. Я нахожу, как заставить микроконтроллер отправить нужную мне семерку. Оказалось, что достаточно увеличить паузу между вызовом функции отправки символа putch() и выключением передатчика.

```
switch (input) {
```

```
        ......
        PORTB = 0xFF;  // Включаем передатчик
        TXEN = 1; // Включаем передачу
        putch('7');
        for (i=0;i<5000;i++); // Пауза
        PORTB = 0xFE;  // Выключаем передатчик
        TXEN = 0; // Выключаем передачу
        ......
        }
}
```

Долгожданный ответ я вижу в окне вывода.

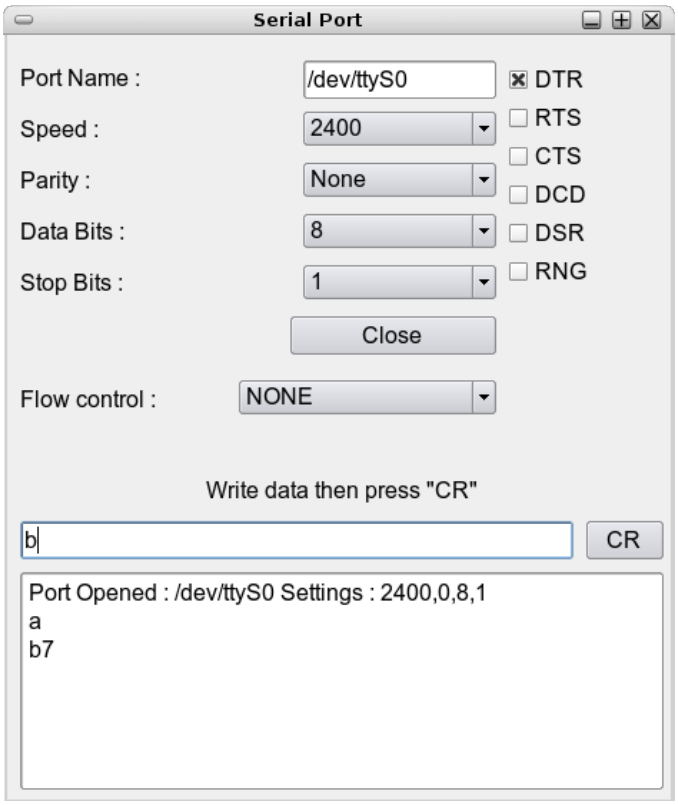


Рис. 5.2. Проверка с помощью примера из пакета системы Gambas

Долгожданный ответ скрыт в символах «b7».

Расширение кругов (на воде?)

Я не собирался доводить развитие проекта, о котором шла речь выше, даже до этой стадии. Все казалось достаточно очевидным, я не ждал каких-то подвохов, программа для микроконтроллера была взята из уже проверенного варианта, но теперь, когда все оказалось вопреки ожидаемому столь запутано, есть смысл продолжить работу над проектом до некоего логического завершения.

Начну с интерфейсной части проекта. Поскольку в терминале ответ от микроконтроллера виден, он должен быть виден и в соответствующем окне интерфейса. Часть кода, написанного мною в Gambas, которая отвечает за получение ответа от контроллера, выглядит так:

```
PUBLIC SUB Button2_Click() ' Клавиша "Счет"
       SerialPort1.Open
       SerialPort1.RTS = FALSE
       PRINT #SerialPort1, "b"
       INPUT #SerialPort1, TextBox2.Text, Eof(SerialPort1)
END
```

Этот код мне кажется правильным, но работать он не желает, что я могу заключить после ряда безуспешных проб. Проведя ряд экспериментов по добавлению настроек порта непосредственно в текст программы, по добавлению разных переменных, я наконец нахожу разумным (и почему не сразу?) заглянуть в текст программы примера работы с последовательным портом, который я использовал для предыдущей проверки, и в котором получил долгожданный ответ, как последовательность символов «b7».

По виду коды в своей сути совпадают, хотя программы разные, и коды, соответственно, тоже. Проходит несколько часов заполненных фантазиями и разочарованием, прежде чем я замечаю, что получение ответа от COM-порта выполнено в работающем коде иначе, чем у меня. Я исправляю это место, копируя с последующим исправлением нужную часть. Теперь работает и мой интерфейс.

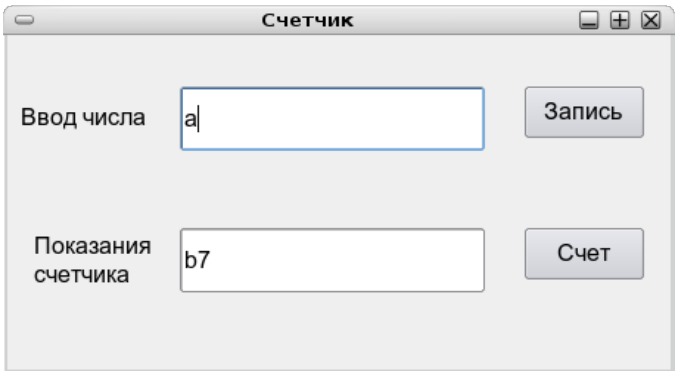


Рис. 5.3. Отклик контроллера в интерфейсной части программы

Часть кода, ответственная за это выглядит так:

```
PUBLIC SUB Button2_Click() ' Клавиша "Счет"
        SerialPort1.Open
        SerialPort1.RTS = FALSE
        PRINT #SerialPort1, "b"
END

PUBLIC SUB SerialPort1_Read()
        DIM answ AS String
        READ #SerialPort1, answ, Lof(SerialPort1)
        TextBox2.Text = TextBox2.Text & answ
        SerialPort1.Close
END
```

Здесь получение отклика от микроконтроллера интерфейсом вынесено как событие. Открывание и закрывание порта я поделил между двумя процедурами, что неправильно, но это легко поправить, хотя бы возвращаясь к первоначальному решению – добавить кнопку открывания и закрывания порта. Мне же следует решить задачу опроса порта ввода контроллера и передачу не просто символа, а этого состояния порта в виде символа. То есть, пора возвращаться к программе Piklab и программатору. Я сейчас не доверяю самодельному программатору и буду использовать купленный ExtraPic, не дорогой, но хорошо работающий (пока). И проверку я буду проводить, видимо, используя не интерфейс, а программу работы с COM-портом из примеров системы Gambas.

Не буду спешить. Для начала переделаю часть программы, отвечающую на команду «b»:

```
void cmd() // Получение и выполнение команды
{
        switch (input) {
                case 'a':
                PORTA = 0x0;
                break;
                case 'b':
        COUNT = PORTA & 0x38;
        if (COUNT == 0x38) PORTA = 0x7; // Индикация тестовая
                break;
                default:
                break;
        }
}
```

COUNT – это переменная типа unsigned char, добавленная мною в программу давно, но до сих пор не используемая. Теперь я хочу использовать ее для того, чтобы подтвердить, что я получаю при чтении в эту переменную именно состояния выводов порта A, которые буду использовать впоследствии.

Светодиоды загораются. Попробуем отправить эту переменную через COM-порт. Шестнадцатеричное число 38 соответствует символу «8» кода ASCII. В первом эксперименте это число я и отправлю.

```
switch (input) {
        ......
        PORTB = 0xFF;  // Включаем передатчик
        TXEN = 1; // Включаем передачу
        putch(0x38);
        for (i=0;i<5000;i++); // Пауза
        PORTB = 0xFE;  // Выключаем передатчик
        TXEN = 0; // Выключаем передачу
        ......
        }
}
```

Эта посылка доходит до адресата. Теперь пойдем дальше. Шаг я делаю совсем маленький – добавляю присвоение переменной состояния порта с маской:

```
COUNT = PORTA & 0x38;
```

Все. Нет реакции ни на команду «a», которая ничего не должна делать, только погасить все светодиоды (должна бы), ни на команду «b», которая должна прислать посылку и зажечь все светодиоды. Ни одна из команд не работает. И этого я вновь не понимаю. И не понимаю долго.

Чтобы отвлечься от грустных размышлений о том, как плохо не знать языков, я делаю попытку понять, что не так с самодельным программатором, который не так давно исправно работал. Программа Piklab, как мне кажется, стала вести себя несколько иначе после обновления. Проверить мне это легко, достаточно перезагрузить компьютер и войти в Ubuntu, где программа не обновлялась. И впрямь, когда я запускаю программу в Ubuntu, компилирую текст, а затем записываю его в контроллер, используя самодельный программатор, то ошибок не возникает. Мало того, запись идет гораздо быстрее, а в окне вывода мало сообщений, тогда как в Fedora 7 эти сообщения при записи кода мелькают непрерывно. Я не исключаю, впрочем, случайностей – когда что-то работает неустойчиво, несколько попыток не дают верного результата. А поскольку я перезагрузил компьютер, я возвращаюсь к проблеме программы, которая не захотела работать. Для продвижения вперед я удаляю всю программу, оставляя только строки вида:

```
COUNT = PORTA;
COUNT = COUNT << 2; // Сдвиг на два бита влево
COUNT = COUNT >> 5; // Сдвиг на пять бит вправо
```

Такими операциями сдвига я намереваюсь выделить вводы RA3-RA5 с целью их использования, когда, и если, мне удастся решить проблемы передачи значений в интерфейс. Используя симулятор gpsim, я отслеживаю значения порта и переменной

COUNT, и вижу, что операции побитового сдвига не выполняются. Закрадывается крамольная мысль, что дело-то может быть не в моих плохих знаниях, а в том, что используемый мной компилятор SDCC – та часть проекта, что относится к программированию PIC-контроллеров находится в стадии разработки – не всегда ведет себя корректно. Чтобы проверить это, я загружаю с сайта HI-TECH демо-версию их компилятора PICC. Слегка споткнувшись на опции «Позволять выполнение файла как программы» на вкладке «Права», я устанавливаю компилятор, перенастраиваю программатор на работу с этим компилятором, поправляю прежний текст (вместо #include <pic16f628a.h> следует записать #include <pic16f62xa.h>, а еще лучше этого не писать, а добавить #include <htc.h>), удалив строку настройки конфигурации по адресу 2007h. Код программы компилируется, а программа работает.

С элегантной записью слова конфигурации по адресу 2007h я пока не знаю что сделать, по причине чего использую «обходной маневр» – записываю программу в контроллер из Piklab, прочитываю ее, а затем поправляю слово конфигурации в окошке обозначенном адресом «2007», куда записываю 2118. Результирующую программу я записываю еще раз. Долго, но прием срабатывает. Кстати, и самодельный программатор опять работает без ошибок, что может быть случайным.

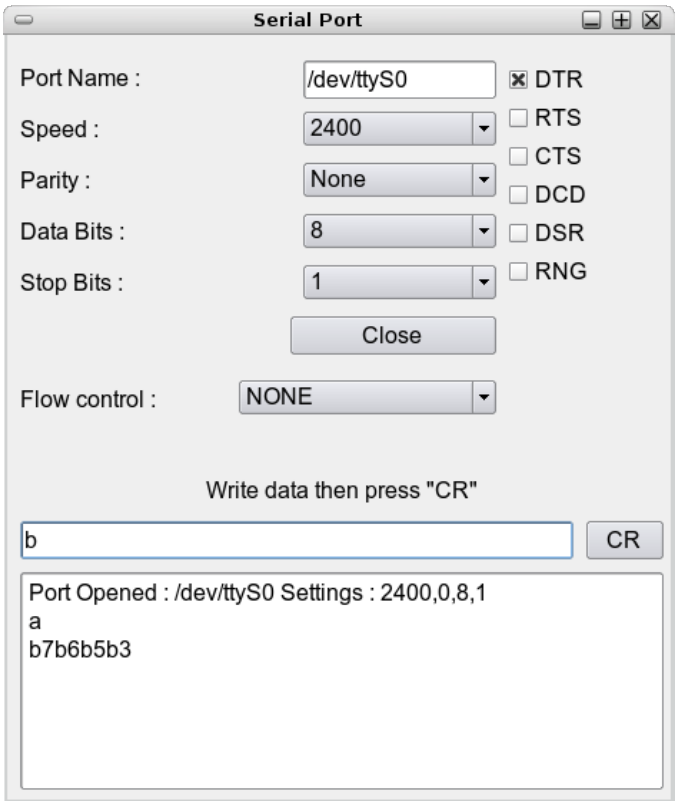


Рис. 5.4. Работающая программа после смены компилятора

Я даже могу себе позволить, замыкая выводы на землю, получить несколько значений – b6, b5 и b3. В этом есть своя прелесть, не так уж я неправ, но есть и недостаток, связанный с тем, что компилятор HI-TECH не генерирует файл .cod, который нужен gpsim для полноценной симуляции. Кроме того, компилятор в полной версии проработает 45 дней, потом его придется сменить на бесплатную lite версию, в которой нет поддержки PIC16F628A, в которой я работаю с микроконтроллером как PIC16F627A, но это меньшее из зол. Важнее было выявить причину столь странного поведения программы, когда операция присваивания переменной, пусть даже значения порта, полностью выводила программу из строя. Мне кажется, что к тому моменту, когда я закончу писать эту книгу (если закончу), компилятор SDCC будет лишен таких чудачеств.

Моя попытка удалить Piklab, установив его версию для Fedora Core 6, не приносит успеха – обе версии теперь работают одинаково, отлично от той, что есть в Ubuntu. Остается проверить, будет ли и как работать программа с интерфейсной частью. Еще

раз для верности запускаю программу Piklab, «кланяюсь» ей, как и положено трижды (программируя микроконтроллер), чтобы запустив интерфейсную часть, получить результат:

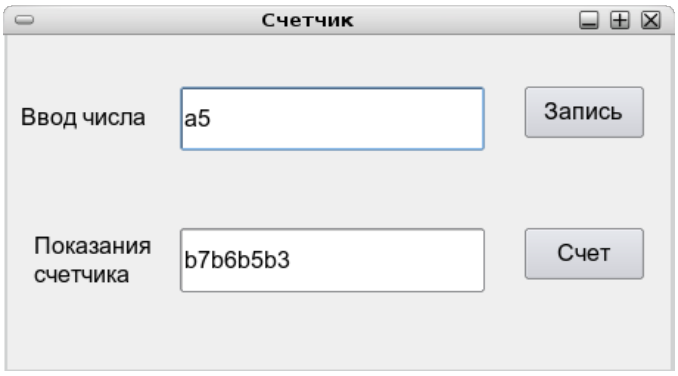


Рис. 5.5. Результат работы интерфейсной части по считыванию порта

Как и прежде я после первого получения кода «b7» последовательно замыкаю выводы RA3-RA5 на землю, чтобы получить другие значения состояния порта.

Подача управляющих сигналов к настоящему моменту мне кажется чисто технической задачей при написании кода. Чуть подольше, возможно, придется повозиться с организацией временных интервалов, но можно использовать встроенные таймеры. Считывание результата работы счетчика тоже не представляется трудным. Пока. Мне же не дает покоя то, как работает симулятор gpsim. Попробую потратить немного (если будет так) времени, чтобы разобраться с ним. Попутно хорошо бы понять, как, работая с компилятором HI-TECH, добиться симуляции схемы с помощью gpsim.

gpsim как зерцало грешника

Жаль, что не сложились отношения с компилятором SDCC, но что есть, то и есть, из этого исходить и приходится.

Самый простой вариант перехода к симуляции, но очень долгий, это из ассемблерной программы, даваемой компилятором, генерировать полный набор с помощью gpasm, который есть в программе Piklab. Но чистить и подгонять файл это долго. Здесь есть над чем задуматься.

Вторая проблема – проблема симуляции. Что-то не так получается с симуляцией. Этим, пожалуй, можно заняться сейчас. Программа gpsim приходит с несколькими примерами программ, из которых меня в первую очередь интересует программа usart_gui. Есть ассемблерный файл, есть файл .stc – файл задания стимулов. Насколько я понимаю без файла usart_gui.cod симуляция не должна работать. А для получения этого файла я создаю папку на рабочем столе, копирую папку examples из набора gpsim, в которой создаю папку с громким названием test. В эту папку копирую оба файла из папки примеров: usart_gui.asm и usart_gui.stc. Содержимое папки в данный момент выглядит так:

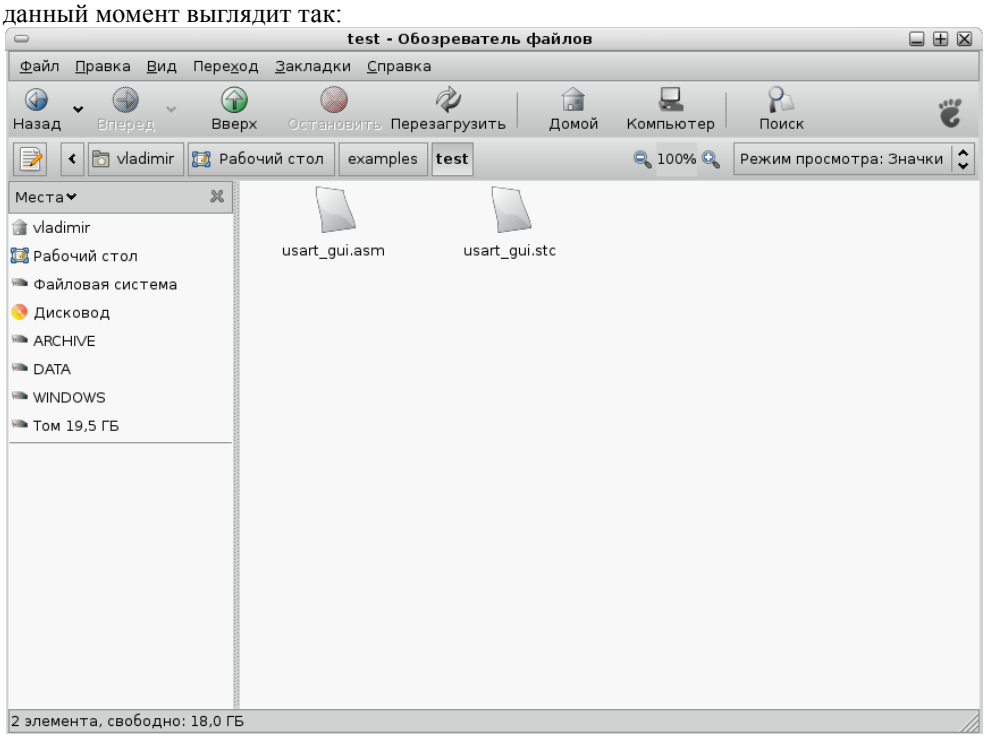


Рис. 5.6. Папка с исходными файлами из примеров gpsim

Теперь можно запустить программу Piklab, создать новый проект, который я назову также usart_gui. Для работы с ассемблерным файлом в разделе Toolchain следует выбрать GPUtils, затем выбрать опцию добавления в качестве исходного файла уже существующего, в качестве которого на следующем шаге создания проекта выбрать usart_gui.asm, что выполняется нажатием на клавишу **Add**, с последующим указанием места расположения файла:

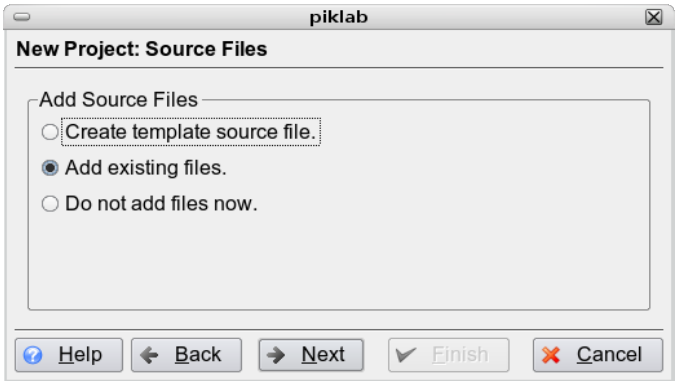


Рис. 5.7. Создание тестового проекта в Piklab

Файл готов к работе с ним, можно запустить компиляцию с помощью пункта Build Project раздела Build основного меню программы Piklab. Компиляция проходит быстро, а папка test пополняется рядом полезных файлов:

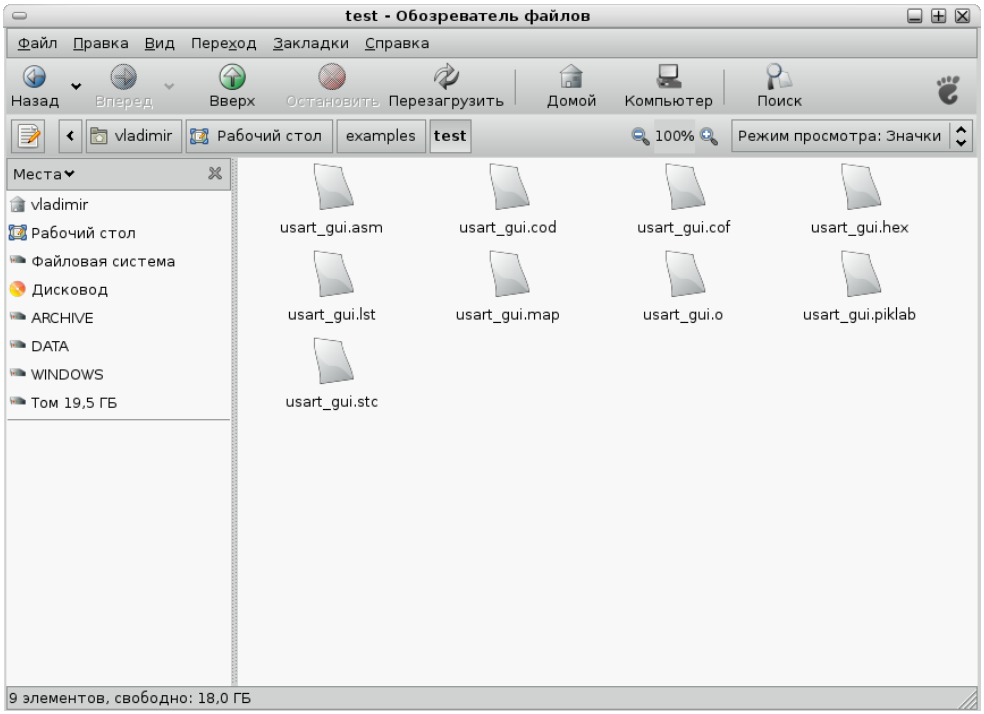


Рис. 5.8. Вид папки проекта после компиляции исходного текста

То, что я так подробно описываю эти рутинные операции, свидетельствует только о мере моей растерянности и озабоченности, вызванной не столько тем, что что-то не получается, редко когда бывает иначе, сколько тем, что я могу неправильно истолковать результаты своих опытов, приписав инструменту качества, которыми отличаются мои неумелые руки. Например, после получения всех необходимых файлов я загружаю в gpsim файл usart_gui.cod, полученный в результате компиляции, следом файл usart_gui.stc. И... и ничего хорошего. Оказывается достаточно загрузить файла usart_gui.stc, чтобы можно было запустить симуляцию. Все необходимое, включая настройку узлов, получается как по мановению волшебной палочки. Запуск симуляции с помощью клавиши Run сразу выводит сообщение в окно USART, а когда я ввожу символы с клавиатуры, я вижу их вывод в том же окне.

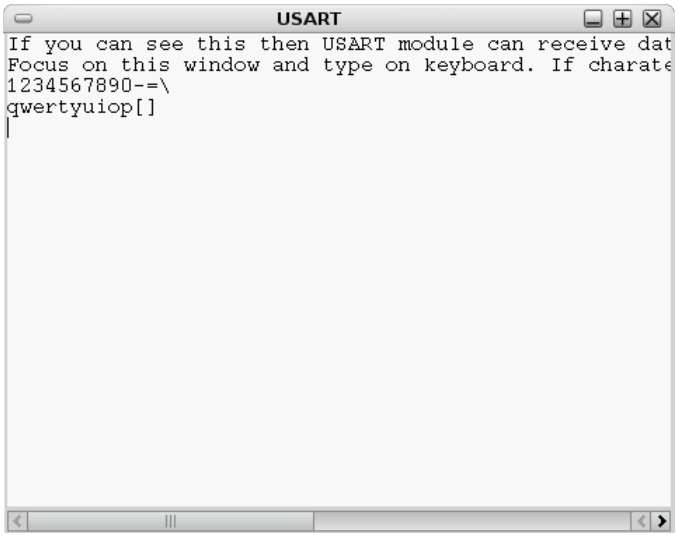


Рис. 5.9. Работа примера симуляции с модулем usart

У меня вывод символов в окно USART с клавиатуры, практически, не получался. И только одно утешает меня – если остановить симуляцию, нажать на клавишу reset программы gpsim, а затем вновь запустить симуляцию, я получаю тот же результат, что получал и раньше, то есть, опять ничего не работает. Но это не мешает мне несколько раз повторить удачную находку с загрузкой файлов .stc из предыдущих проектов. Чтобы получить этот файл, я использую клавишу Save Cofiguration... на макетной плате Breadboard. Клавиша срабатывает не сразу, но когда срабатывает, то предлагает сохранить файл с тем именем, который ты выбираешь. В руководстве к программе gpsim упоминалось, что стимулы лучше получать из файла, но как это сделать я не понял. Теперь есть возможность избежать бесконечных повторов с вводом библиотеки, с заданием и привязкой узлов, и т.д.

Однако радость преждевременна. Файл, который я получаю с помощью клавиши сохранения конфигурации, если не ввести новое имя, называется netlist.stc, и загружает только часть необходимого. Используя в качестве подсказки работающий файл из примера и используя метод проб и ошибок, я нахожу, что следует изменить строку в начале файла netlist.stc:

```
# load s mycod.cod
```

Убрав комментарий, нужно заменить шаблон mycod на имя реального файла, который у меня назван:

```
load s counter_start.cod
```

Кроме этого, двигаясь вниз по файлу (после некоторого потраченного времени), я поднимаю текст загрузки модуля usart над параметрами контроллера:

```
# Module libraries:

module library libgpsim_modules.so

# Modules:
module load usart U1

U1.rxbaud=9600
U1.txbaud=9600
U1.rx=0
U1.tx=0
U1.crlf=true
U1.loop=false
U1.console=false

U1.xpos=72,00000000000000
U1.ypos=276,0000000000000

p16f628a.CONFIG=$ff
p16f628a.WarnMode=true
p16f628a.SafeMode=true
p16f628a.UnknownMode=true
p16f628a.BreakOnReset=true
p16f628a.BreakOnInvalidRegisterRead=true
p16f628a.BreakOnInvalidRegisterWrite=true
p16f628a.frequency=20000000,00000000
p16f628a.xpos=72,00000000000000
p16f628a.ypos=72,00000000000000
```

После этой операции все начинает загружаться правильно, включая то, что оба узла, node rb1 и node rb2, которые я создал для соединения usart с контроллером, оба узла определяются и появляются правильным образом. Но чуть позже выясняется, что формат чисел, определяемый системой, не вполне согласуется с программным. Я имею ввиду запятую в десятичных числах. Ее следует поменять на точку. Например, в последней строке:

```
p16f628a.ypos=72.00000000000000
```

Несколько программ, приведенных в примерах, показывают как работает симулятор, и сколько интересных возможностей скрывают дополнительные модули.

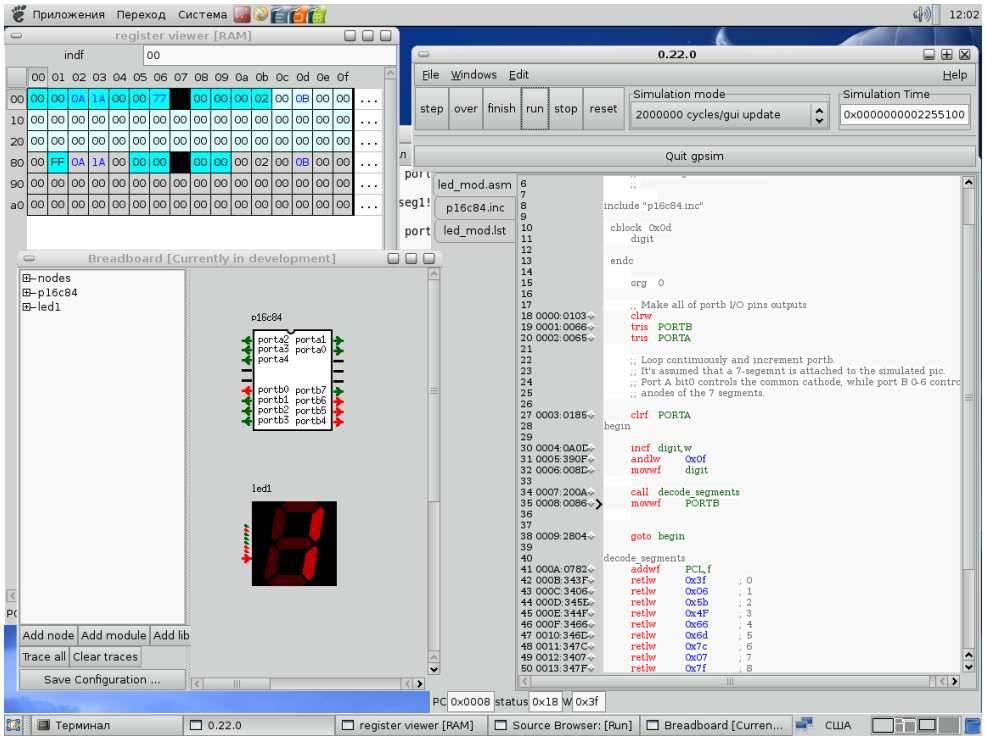


Рис. 5.10. Работа симулятора gpsim с семисегментным индикатором

Что можно сказать по предварительным результатам:

- Не возникает проблем при работе в связке Piklab-gpsim, если использовать ассемблер.
- Есть некоторые проблемы при использовании компилятора «C» SDCC, связанные, скорее всего, с самим компилятором.
- Компилятор HI-TECH, полная и свободная версия, не дает при компиляции нужного файла формата .cod.

Что касается ассемблера, с профессиональной точки зрения это наилучший вариант, позволяющий полностью контролировать код программы, но вариант наиболее трудоемкий и долгий. Написание программного кода на ассемблере – предмет отдельного, я бы сказал, увлечения. Это увлечение вполне доступно любому, а по мере накопления опыта и код станет более изящным, и количество готовых программных модулей увеличится, хотя бы за счет разбиения программ из примеров на отдельные работающие модули. Словом, есть над чем подумать, поскольку в этом

виде все работает, и работает хорошо.

С языком «С» при желании использовать компилятор SDCC, похоже, следует разобраться, почитав документацию к этому компилятору. Использование же компилятора HI-TECH позволяет обойти проблему отсутствующего формата следующим образом (мне не очень понятным). В документации к компилятору говорится о том, что есть опция –OUTPUT=cod, которая должна выводить требуемый файл. Для использования этой опции перед компиляцией проекта в Piklab следует войти в пункт Ptoject Options... раздела Project основного меню, и выбрать режим работы не автоматический, а пользовательский.

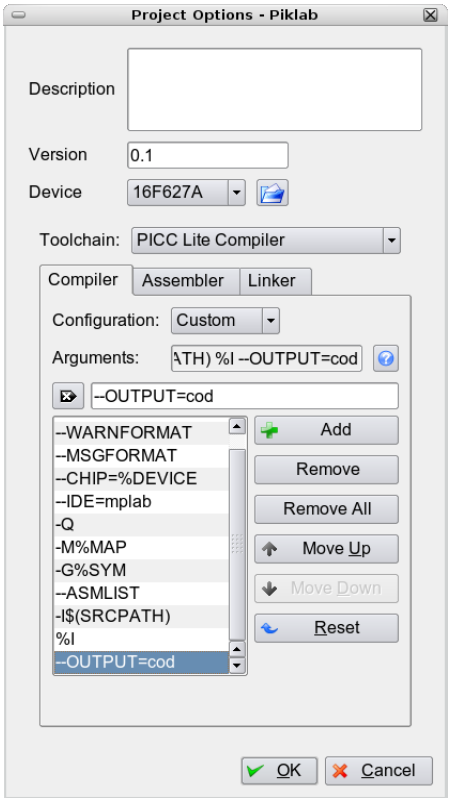


Рис. 5.11. Выбор пользовательского режима трансляции программы

Если в нижнем окне ничего не выделять, то в окне Arguments: можно ввести нужную опцию, а нажав на клавишу **Add**, добавить ее к пользовательским опциям. В этом месте я немного не понимаю происходящее. Если теперь компилировать проект, то нужного файла в проекте не обнаруживается. Но! Если убрать опцию -S, которая

имеет место «по умолчанию», то после трансляции проекта файл с расширением .cod появляется. Я пытался смещать эту опцию, добавлять ее на вкладке Assembler, лучшего результата, чем получилось, я не достиг. В итоге последовательность действий такова (если вы не найдете лучшего решения):

1. Установить пользовательские опции трансляции проекта.
2. Оттранслировать файл, используя Build-Compile File.
3. Отладить файл в отладчике gpsim.
4. Оттранслировать проект, используя автоматический режим.

При этом возникает еще одна особенность. Использование PICC Lite работает довольно устойчиво, а полная версия (демо-версия), иногда отладчиком воспринимается, но чаще после начала загрузки дает сбой и исчезает с сообщением, что все у меня плохо. При работающем варианте, что мне кажется удобнее, gpsim показывает код текста программы, написанный на «С».

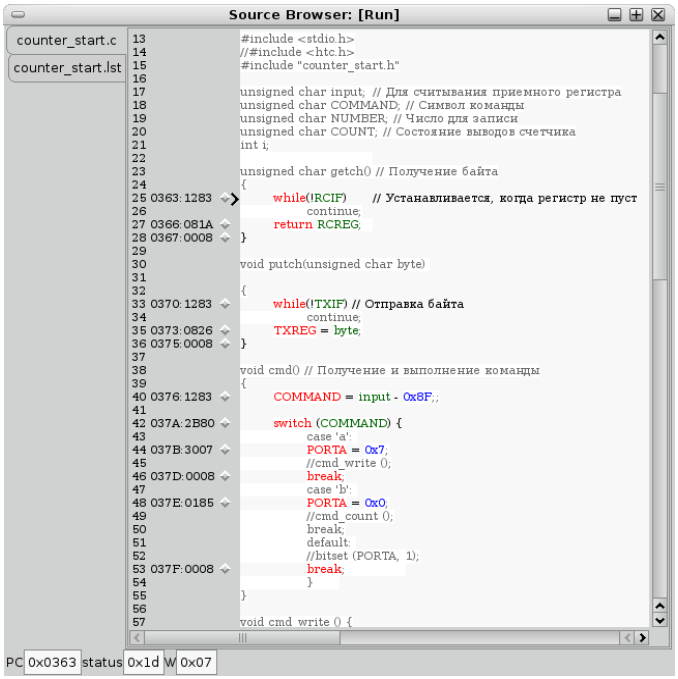


Рис. 5.12. Работа gpsim при симуляции cod-файла компилятора HI-TECH

После отладки программы можно выбрать автоматический режим компиляции в

опциях проекта, еще раз оттранслировать его, выбрав Build-Build Project, и загрузить полученный hex-файл в контроллер.

В настоящий момент для меня остается не решенной проблема ввода символов с клавиатуры. Если бы не корректная работа программы из набора примеров gpsim, я готов бы был отмахнуться от этой проблемы. Но пример работает корректно. Правда, после рестарта отображение в окне USART напоминает то, что получаю я, но это после рестарта.

Из предположений, которые я отнес бы к разумным, есть два. Первое касается локали, используемой системой, но оно слабое, поскольку пример из набора пакета gpsim работает правильно. Второе, более правдоподобное, как мне кажется, может быть связано с настройками контроллера и модуля usart. И то, и другое должно быть настроено правильно. Я взял настройки старой программы, что некогда работала, но все ли в этом правильно, ведь прежде я работал в Windows и с программой MPLAB. Чтобы проверить настройки, в какой-то момент я пытался сделать при симуляции разные установки частоты контроллера, но это не привело к успеху. Однако могли быть и другие причины.

Попробуем иначе проверить настройки. Для начала я изменю тестовую программу микроконтроллера. Я не буду принимать ввод с клавиатуры, но лишь отправлю на экран символ «7». Посмотрим, что обнаружится в регистре TXREG модуля usart, и что придет на экран USART?

Файл заголовка.

```
void putch(unsigned char); // Передача символа
void init_comms(); // Инициализация коммуникации
void cmd(); // Отправка символа «7»
```

Основной файл программы

```
/* ------------------------------------------------------------------------ */
/* Template source file generated by piklab */

#include <pic16f62xa.h>
#include <stdio.h>
#include "test1.h"
int i;

/*unsigned char getch() // Получение байта
{
        while(!RCIF)   // Устанавливается, когда регистр не пуст
                continue;
        return RCREG;
} */

void putch(unsigned char byte)  // Отправка байта

{
```

```
        while(!TXIF) // Устанавливается, когда регистр пуст
                continue;
        TXREG = byte;
}

void cmd() // Получение и выполнение команды
{
        CREN = 0; // Выключаем прием
        PORTB = 0xFF;  // Включаем передатчик
        TXEN = 1; // Включаем передачу
        putch('7'); // Отправка символа «7»
        for (i=0;i<5000;i++); // Пауза
        PORTB = 0xFA;  // Выключаем передатчик
        TXEN = 0; // Выключаем передачу
        CREN = 1; // Включаем прием
        PORTA = 0x7;   // Тестовая индикация
}

void init_comms()      // Инициализация модуля
{
        PORTA = 0x0;   // Настройка портов A и B
        CMCON = 0x7;
        TRISA = 0x38;
        TRISB = 0xFA;
        RCSTA = 0x90;  // Настройка приемника
        TXSTA = 0x6;   // Настройка передатчика
        SPBRG = 0x68;  // Настройка режима приема-передачи
        PORTB = 0xFE;  // Выключаем драйвер RS485 на передачу
}


void main(void) {
        init_comms(); // Инициализация модуля

start:  cmd();
        for (i=0;i<30000;i++); // Пауза
        PORTA = 0;     // Тестовая индикация
        for (i=0;i<30000;i++); // Пауза
        goto start;
}
```

В качестве компилятора я использую PICC Lite. В свойствах проекта настройки компилятора, как и выше, пользовательские – без опции -S, которую я удаляю с помощью клавиши **Remove**, и с добавленной опцией –OUTPUT=cod. Компиляцию выполним только для основного файла (Build-Compile File). Файл test1.stc для работы с симулятором имеет вид:

```
# This file was written by gpsim.
# You can use this file for example like this:
#     gpsim -s mycode.cod -c netlist.stc
# If you want to add commands, you can create another .stc file
# and load this file from it. Something like this:
```

```
# ----------- myproject.stc ---------------

load s test1.cod

#frequency 20000000
# load c netlist.stc
# ----------------------------------------
# You can then just load this new file:
#     gpsim -c myproject.stc
# and use netlist.stc whenever you save from the breadboard.
# Processor position:
# Module libraries:

module library libgpsim_modules.so

# Modules:

module load usart U1

U1.rxbaud=2400
U1.txbaud=2400
U1.rx=0
U1.tx=0
U1.crlf=true
U1.loop=false
U1.console=true
U1.xpos=72.00000000000000
U1.ypos=276.0000000000000

p16f628a.CONFIG=$ff
p16f628a.WarnMode=true
p16f628a.SafeMode=true
p16f628a.UnknownMode=true
p16f628a.BreakOnReset=true
p16f628a.BreakOnInvalidRegisterRead=true
p16f628a.BreakOnInvalidRegisterWrite=true
p16f628a.frequency=1000000.00000000
p16f628a.xpos=72.00000000000000
p16f628a.ypos=72.00000000000000

# Connections:

node rb1
attach rb1 portb1 U1.TXPIN

node rb2
attach rb2 portb2 U1.RXPIN

# End.
```

Запуск симуляции не отображает в окне USART ничего. И, хотя в файле test1.stc частоту контроллера я определил в 1МГц, мне приходит в голову заглянуть в его свойства на макетной плате симулятора. Не напрасно!

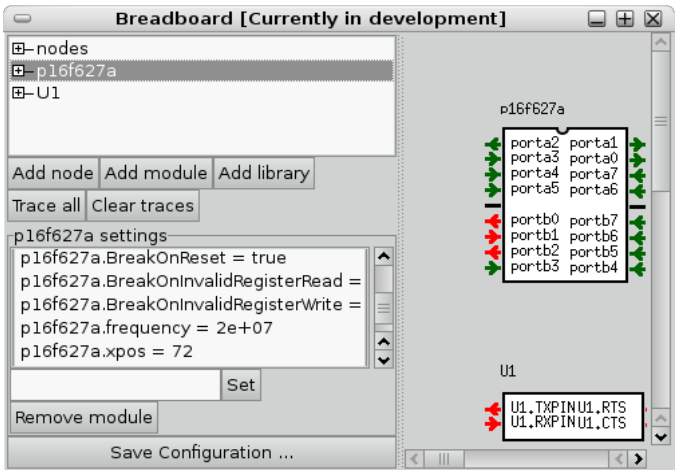


Рис. 5.13. Отображение настроек контроллера в gpsim

Вопреки моей уверенности в частоте контроллера она нисколько не изменилась против настроек «по умолчанию». Рассерженный этим обстоятельством, я удаляю весь блок настроек из stc-файла, пытаюсь «на ходу» поменять частоту, но без особого успеха, а затем, вспомнив, что в начале файла есть упоминание о частоте:

```
#frequency 20000000
```

Меняю эту строку, придав ей следующий вид:

```
frequency 1000000
```

Теперь, загрузив stc-файл в симулятор gpsim, я вижу, что частота приняла требуемое значение, что сказывается и на том, что я получаю в окне USART.

Рис. 5.14. Вид символа «7» в окне USART симулятора gpsim

Не победа, но шажок вперед. Сейчас вид окна вывода USART похож на то, что я получал, запуская пример работы с usart, но после рестарта программы.

При этом в регистре TXREG, доступном для записи из программы регистре, который можно посмотреть в окне register viewer [RAM] симулятора по шестнадцатеричному адресу 19h , отображается правильное значение 37h.

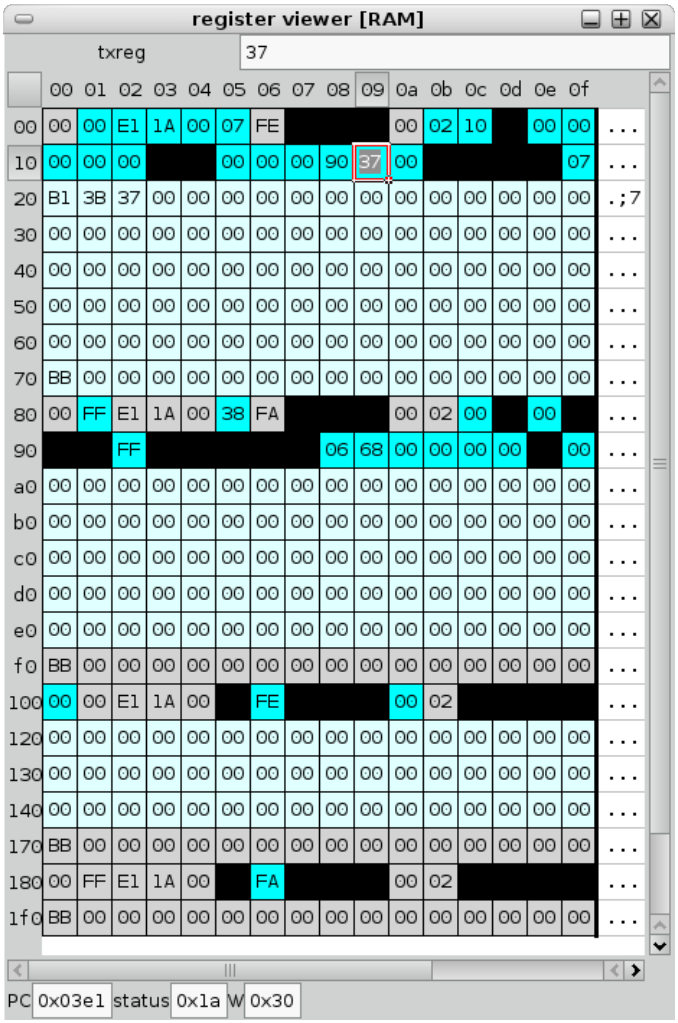


Рис. 5.15. Отображение регистров в симуляторе gpsim

Интересно, подумалось мне в этой связи, а почему я задаю частоту микроконтроллера в 1 МГц? Ответа я не знаю, тратить время на поиски старых записей не хочется, поэтому я исправляю в stc-файле частоту на 4МГц.

```
frequency 4000000
```

И столь долгожданное чудо свершилось. После запуска симулятора я, наконец, вижу в окне USART:

Рис. 5.16. Правильное отображение символа в окне USART симулятора

Еще не победа, но хороший шаг вперед. Пока удача на моей стороне, любопытно проверить, как оттранслирует эту программу компилятор SDCC. Удалим из папки проекта в Piklab все файлы кроме основных и stc-файла, последний не зависит от компилятора. И, сменив в свойствах проекта (Project Options...) компилятор PICC Lite на Small Device C Compiller (SDCC), а контроллер на PIC16F628A, слегка подправив текст, что также сводится к замене контроллера, я запускаю компиляцию, а следом симуляцию. Как и следовало ожидать, результат не отличается от предыдущего. И даже индикаторы на breadboard, вернее выводы порта, которые я предназначил к индикации в программе, весело меняют цвет. В этом случае SDCC компилятор работает прекрасно.

А как обстоят дела с отображением ввода с клавиатуры? Для этой цели у меня предназначена программа test2. Она просто получает символ с клавиатуры и отправляет его обратно. Попробую я этот тест сразу с компилятором SDCC.

Файл заголовка

```
unsigned char getch(); // Прием символа
void putch(unsigned char); // Передача символа
void init_comms(); // Инициализация коммуникации
void cmd();
```

Основной файл

```
/* ------------------------------------------------------------------------ */
/* Template source file generated by piklab */
```

```
#include <pic16f628.h>
#include <stdio.h>
#include "test2.h"

unsigned char input;
int i;

unsigned char getch() // Получение байта
{
        while(!RCIF)   // Устанавливается, когда регистр не пуст
                continue;
        return RCREG;
}

void putch(unsigned char byte)
{
        while(!TXIF) // Отправка байта
                continue;
        TXREG = byte;
}

void cmd() // Получение и выполнение команды
{
        CREN = 0; // Выключаем прием
        PORTB = 0xFF;  // Включаем передатчик
        TXEN = 1; // Включаем передачу
        putch(input);
        for (i=0;i<5000;i++); // Пауза
        PORTB = 0xFA;  // Выключаем передатчик
        TXEN = 0; // Выключаем передачу
        CREN = 1; // Включаем прием
}

void init_comms()      // Инициализация модуля
{
        PORTA = 0x0;   // Настройка портов A и B
        CMCON = 0x7;
        TRISA = 0x38;
        TRISB = 0xFA;
        RCSTA = 0x90;  // Настройка приемника
        TXSTA = 0x6;   // Настройка передатчика
        SPBRG = 0x68;  // Настройка режима приема-передачи
        PORTB = 0xFE;  // Выключаем драйвер RS485 на передачу
        PORTA = 0x7;   // Тестовая индикация
}

void main(void) {
        init_comms(); // Инициализация модуля

start:  CREN =1; // Начинаем работать
        input = getch();
        cmd();
        goto start;
}
```

Результат радует глаз.

Рис. 5.17. Правильное отображение ввода с клавиатуры в gpsim

Наконец-то! И в этом случае компилятор SDCC работает верно.

Остается проверить программу полностью. С компилятором SDCC та же проблема, что и раньше – не отрабатывается присвоение переменной значения состояния порта, а свободная версия HI-TECH вполне, с моей точки зрения, адекватно отрабатывает программу.

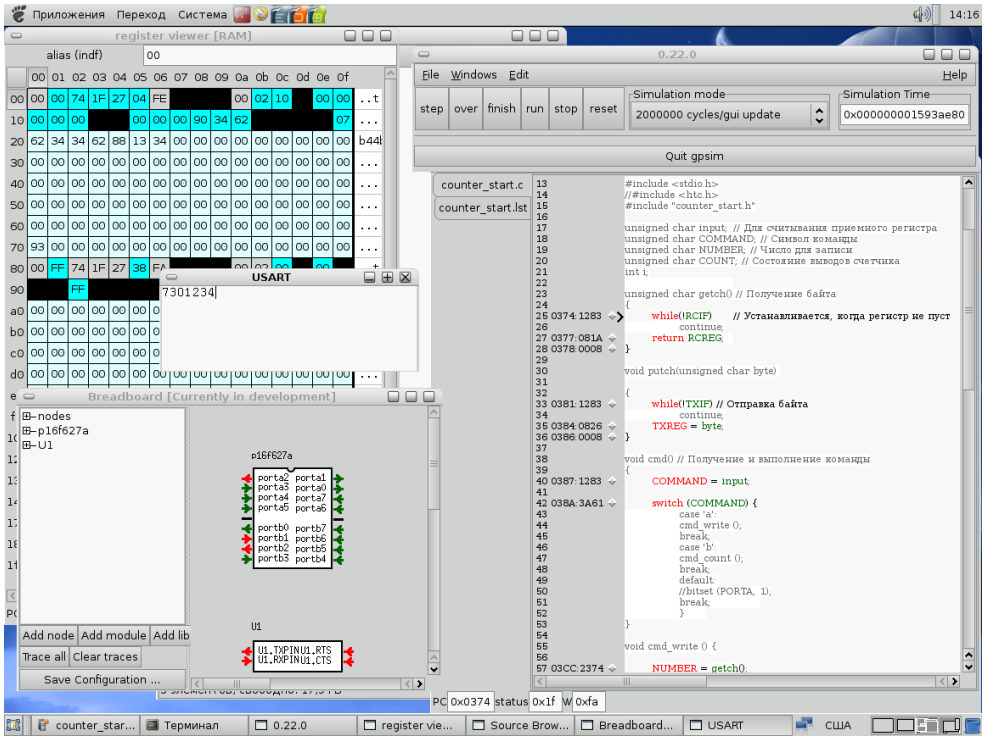


Рис. 5.18. Окончательная проверка программы в gpsim

Программа, которую я тестирую с помощью симулятора, имеет вид:

Файл заголовка

```
unsigned char getch(); // Прием символа
void putch(unsigned char); // Передача символа
void init_comms(); // Инициализация коммуникации
void cmd(); // Прием общих команд
void cmd_write (); // Прием команды «a»
void cmd_count (); // Прием команды «b»
```

Основной файл

```
/* ------------------------------------------------------------------------ */
/* Template source file generated by piklab */
#include <pic16f62xa.h>
#include <stdio.h>
#include "counter_start.h"

unsigned char input;  // Для считывания приемного регистра
unsigned char COMMAND;// Символ команды
unsigned char NUMBER; // Число для записи
unsigned char COUNT; // Состояние выводов счетчика
int i;

unsigned char getch() // Получение байта
{
        while(!RCIF)   // Устанавливается, когда регистр не пуст
                continue;
        return RCREG;
}

void putch(unsigned char byte)

{
        while(!TXIF) // Отправка байта
                continue;
        TXREG = byte;
}

void cmd() // Получение и выполнение команды
{
        COMMAND = input;

        switch (COMMAND) {
                case 'a':
                cmd_write ();
                break;
                case 'b':
                cmd_count ();
                break;
                default:
```

```
                break;
                }
}

void cmd_write () {

        NUMBER = getch();

        switch (NUMBER) {
                case '0':
                PORTA = 0x0;
                break;
                case '1':
                PORTA = 0x1;
                break;
                case '2':
                PORTA = 0x2;
                break;
                case '3':
                PORTA = 0x3;
                break;
                case '4':
                PORTA = 0x4;
                break;
                case '5':
                PORTA = 0x5;
                break;
                case '6':
                PORTA = 0x6;
                break;
                case '7':
                PORTA = 0x7;
                break;
                default:
                break;
                }
}

void init_comms()      // Инициализация модуля
{
        PORTA = 0x0;   // Настройка портов A и B
        CMCON = 0x7;
        TRISA = 0x38;
        TRISB = 0xFA;
        RCSTA = 0x90;  // Настройка приемника
        TXSTA = 0x6;   // Настройка передатчика
        SPBRG = 0x68;  // Настройка режима приема-передачи
        PORTB = 0xFE;  // Выключаем драйвер RS485 на передачу
        PORTA = 0x1;   // Тестовая индикация
}

void cmd_count ()
{
        COUNT = PORTA;
```

```
        CREN = 0; // Выключаем прием
        PORTB = 0xFF;  // Включаем передатчик
        TXEN = 1; // Включаем передачу
        COUNT = COUNT + 0x30;
        putch(COUNT);
        for (i=0;i<5000;i++); // Пауза
        PORTB = 0xFA;  // Выключаем передатчик
        TXEN = 0; // Выключаем передачу
        CREN = 1; // Включаем прием
}


void main(void) {
        init_comms(); // Инициализация модуля

start:  CREN =1; // Начинаем работать
        input = getch();
        cmd();
        goto start;
}
```

stc-файл, конечно, почти совпадает по содержанию с тем, что было приведено выше, исключая строку

```
load s counter_start.cod
```

где меняется имя cod-файла на относящееся к текущей программе. В остальном все, практически, сохраняется. Включая, что после сброса компьютера, клавишей **Reset** симулятора, его частота устанавливается «по умолчанию», что мешает правильно понимать ввод-вывод.

Вялая попытка оправдаться

При последней проверке я немного слукавил. Та программа контроллера, которую я использовал совместно с интерфейсом и макетной платой (физической) отличается от тестовой версии наличием двух строк и функция отправки состояния порта A, что выглядит так:

```
void cmd_count ()
{
        COUNT = PORTA;
        CREN = 0; // Выключаем прием
        PORTB = 0xFF;  // Включаем передатчик
        TXEN = 1; // Включаем передачу
        COUNT = COUNT << 2;
        COUNT = COUNT >> 5;
        COUNT = COUNT + 0x30;
        putch(COUNT);
        for (i=0;i<5000;i++); // Пауза
        PORTB = 0xFA;  // Выключаем передатчик
        TXEN = 0; // Выключаем передачу
        CREN = 1; // Включаем прием
}
```

На «живой» макетной плате выводы порта соединяются с подтягивающими резисторами. При симуляции в gpsim есть возможность добавить такую «подтяжку», но мне не хотелось морочить голову ни себе, ни вам. Если при симуляции использовать две удаленные строки, то порта A всегда будет обнулен, что я и получаю, а мне гораздо больше нравится использовать команду «a» с символом включения индикаторов – числом от 0 до 7, и получать их при чтении состояния порта. Слукавил.

Но самая большая тайна заключена в поведении компилятора SDCC, прекрасно работающего во многих случаях, а при чтении состояния порта

```
COUNT = PORTA;
```

не желающего работать. Это очень интересная загадка. К которой можно добавить тот факт, что собственно присвоение переменной состояния порта имеет место, как это показывает симуляция в gpsim. Интересно. Пока у меня только одно объяснение, функции чтения и записи в usart взяты мною из примеров для компилятора HI-TECH. Может быть здесь какой-то подводный камень?..

Нет. Не здесь.

Все гораздо проще. Я использовал версию компилятора 2.6. Появилась новая версия, 2.7.0. Замена компилятора, и нет места тайне. Теперь симуляция проходит точно так же, как и с компилятором HI-TECH. Остается проверить работу с «живой»

макетной платой и интерфейсом, да проверить всю связку в другом дистрибутиве Linux (Ubuntu).

Что ж, все работает и с другим дистрибутивом после замены версии SDCC. Работает и макет с интерфейсной частью проекта. Лучше не бывает.

Хотя нет, бывает. Меня не оставляло ощущение дискомфорта от того, что в Ubuntu программирование контроллера в Piklab идет быстрее, а в окне программирования гораздо меньше сообщений, чем я получаю в Fedora 7. И в этом случае все оказалось просто – я сам, как полагаю, зацепил настройки Piklab, установив на вкладке General вывод всех сообщений об ошибках. Стоило поменять эту опцию на другую, как показано на рисунке ниже, и все стало проходить быстро, как и в Ubuntu.

Сам виноват.

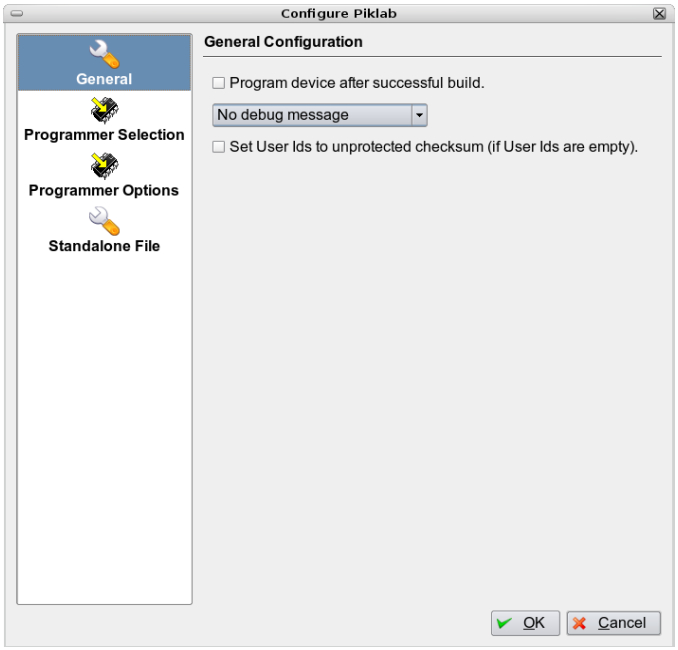


Рис. 5.19. Установка опции вывода сообщений в Piklab

И оргвыводы

Где можно с пользой применить такой подход? Я имею ввиду не борьбу «с ветряными мельницами», а использование микроконтроллера и интерфейса, написанного в удобной среде программирования подобной Gamabas. В первую очередь там, где для наладки или проверки требуется большое количество импульсных сигналов, следующих в некоторой сложной последовательности. Конечно, можно найти решение с обычным генератором импульсов, как тактовым, а затем формировать все остальное, используя схемы задержки, укорачивания и удлинения импульсов и т.д., но мне кажется, что веселый дует «компьютер и микроконтроллер» справятся с подобной задачей лучше, построение пойдет быстрее и обойдется дешевле, но, главное, реализация получится гораздо интереснее.

Имея некоторый опыт в работе со всеми необходимыми программами, можно было бы написать программу, которая называлась бы «Тестер для микросхем». Если на макетную плату рядом с панелькой для микроконтроллера поставить панельку для микросхем, если в микроконтроллер добавить несколько программ обслуживания этих микросхем, а в программу интерфейса добавить команды перехода к этим программам обслуживания микросхем, то что-то из всего этого полезное, видимо, и получится.

Еще больше пользы эти шаги по созданию проверочного стенда могут принести при проверке функциональных цифровых модулей, требующих достаточно большого количества управляющих сигналов. Хотя, если подумать, можно найти и другие применения этому дуэту – программный интерфейс и микроконтроллер. Жизнь подскажет.

Итак. Среда программирования Gambas позволяет достаточно легко и быстро создавать интерфейсную часть для общения по COM-порту, например, с микроконтроллером. А возможностей у микроконтроллера, великое множество.

Программа Piklab великолепно и без каких-либо проблем работает с программами, написанными на ассемблере. Но писать программы на ассемблере есть смысл тогда, когда есть интерес к ассемблеру или задача достаточно специфическая. В противном случае лучше воспользоваться либо компилятором PICC Lite производства HI-TECH, или купить его полную версию, либо, что удобнее, использовать компилятор SDCC.

Программа Piklab достаточно уверенно работает с простейшим самодельным программатором JDM Classic, но еще успешнее с покупным и недорогим EXTRA-PIC.

При правильной настройке симулятор gpsim, который работает в Linux, оказывается не менее удобным отладочным средством, чем встроенный симулятор MPLAB.

Чтобы это доказать себе, а только себе и можно что-то доказать, я опробую еще одно средство, которое есть в gpsim, для чего быстро (как и в прошлый раз) напишу код программы test3 для контроллера PIC16F628A, код даже без файла заголовков.

```
/* ------------------------------------------------------------------------ */
/* Template source file generated by piklab */
#include <pic16f628a.h>

/* ------------------------------------------------------------------------ */
/* Configuration bits: adapt to your setup and needs */

typedef unsigned int word;
word at 0x2007 CONFIG = _WDT_ON & _PWRTE_OFF & _RC_OSC_CLKOUT & _MCLRE_ON &
_BOREN_ON & _LVP_ON & _DATA_CP_OFF & _CP_OFF;

int i;

void init_comms()      // Инициализация модуля
{
        CMCON = 0x7; // Настройка портов A и B
        TRISA = 0x38;
        TRISB = 0xFA;
        RCSTA = 0x90;  // Настройка приемника
        TXSTA = 0x6;   // Настройка передатчика
        SPBRG = 0x68;  // Настройка режима приема-передачи
        PORTB = 0xFE;  // Выключаем драйвер RS485 на передачу
        PORTA = 0x0;
}

void main() {
    /* << insert code >> */

start:  init_comms();
        PORTA = 0x0;
        for (i=0;i<100;i++);
        PORTA = 0x1;
        for (i=0;i<100;i++);
        PORTA = 0x1;
        goto start;
}
```

Если код откомпилировать, запустить gpsim, в котором открыть cod-файл, а затем открыть осциллограф (Windows-Scope), и щелкнуть в его правой части, где появляется окошко для ввода, куда вписать, например, porta0, а после запуска симуляции нажать Enter на клавиатуре (при этом окошко исчезает), то можно наблюдать изменение сигнала на нулевом выводе порта A:

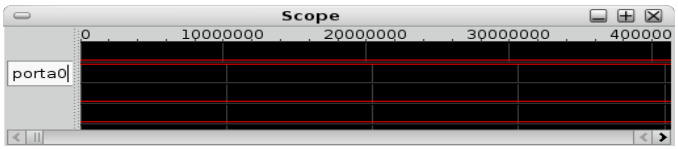


Рис. 5.20. Осциллограф в отладчике gpsim

По словам автора проекта gpsim этот инструмент нуждается в дополнительной доработке, и пока только отображает логическое состояние вводов-выводов портов, но оно уже есть. И, когда будет доработано, возможно будет не хуже того, что в MPLAB. В Linux многое меняется быстро.

Глава 6. Сказка о ловком программисте

Очень мне хотелось в этой главе позлословить о специалистах, пугающих ужасными последствиями перехода с Windows на Linux, о тех, кто кормится от Windows, повторяя, какие они белые и пушистые, что будет особенно заметно в новой версии. Но, написав половину главы, я увидел, как это скучно. И никакого отношения не имеет ни к электронике, ни к программированию, ни, особенно, к радиолюбительству.

Я вычеркнул все написанное, оставив только название главы себе в назидание в качестве «белого дракона на белом полотне».

За время написания этой книги я многократно обращался к книгам по программированию, написанным профессионалами. Листая их, находил много интересного и полезного, но для программистов. Это касается и работы с графикой, и расчетных задач, и всего того, что входит в круг интересов и забот программистов. Но какое это имеет отношение к электронике или радиолюбительству?

С другой стороны, мне знакомо чувство огорчения и неудовлетворенности, которое испытываешь, когда, найдя неисправность в полезном и, порой, весьма дорогостоящем электронном изделии, понимаешь, что следует заменить контроллер, но найти программу для загрузки в этот контроллер невозможно. Подчас, вся схема прибора или устройства – это микроконтроллер. В профессиональной практике ремонт подобной электроники сводится не к замене контроллера, а к замене платы на новую, даже если это единственная плата, и самое ценное на этой плате – программа, «зашитая» в контроллер.

Любитель, получая подобное неработающее электронное чудо, которое его владельцы, возможно, давно и безуспешно «прогуливали» по мастерским и техническим центрам, находится в более выгодном положении, чем профессионал. Есть что-то, что безнадежно испорчено, есть интерес вернуть это к жизни, есть время, чтобы осуществить это. Нужно не так много – умение программировать. Понятно, что повторить программу, заложенную в промышленное изделие едва ли удастся, но ничто не мешает создать другую. Даже, если изделие не будет выполнять все функции, заложенные производителем, но будет выполнять базовые, обычно этого бывает достаточно.

Если программисту-профессионалу важно изучить теоретические аспекты предмета, то как быстрее и проще научиться программировать любителю?

Программируя.

Предварительное рассмотрение проекта «Генератор»

Если я скажу, что весь смысл проекта в демонстрации удобства использования прерывания в некоторых случаях при работе с микроконтроллером, то вам не

захочется читать дальше. Поэтому я не скажу, в чем смысл проекта, а буду уверять, что ценность предлагаемого решения значительно больше вложенного в него труда, что практическая реализация проекта даст несомненные преимущества, и в определенных условиях это решение может оказаться наилучшим.

Как и с проверкой работы счетчика, проект рассчитан на связку компьютер-микроконтроллер. Самым очевидным и достаточно удобным было бы построение генератора импульсов с задаваемой частотой и скважностью импульсов. Такое решение, во-первых, легко реализовать, во-вторых, далеко не каждый любитель имеет в своем распоряжении генератор импульсов с регулируемой скважностью. Однако вы прекрасно справитесь с решением этой задачи и без моих подсказок. А я хочу рассказать о построении генератора пилообразного напряжения.

Схемных решений, как мне кажется, может быть, по меньшей мере, несколько. Я остановлюсь на самом очевидном – микроконтроллер в качестве задающего генератора для цифро-аналогового преобразователя (ЦАП), формирователя сигнала. Для предварительного макетирования и создания функционального прототипа устройства я с большим удовольствием воспользуюсь одной из программ EDA (САПР), о которых рассказывал в предыдущей книге «Наглядная электроника». Там достаточно много написано о пользе применения этих замечательных программ, особенно на предварительном этапе разработки и для начинающих, чтобы самому последовать своим советам. Схема, макет которой я соберу в программе KTechlab, выглядит так:

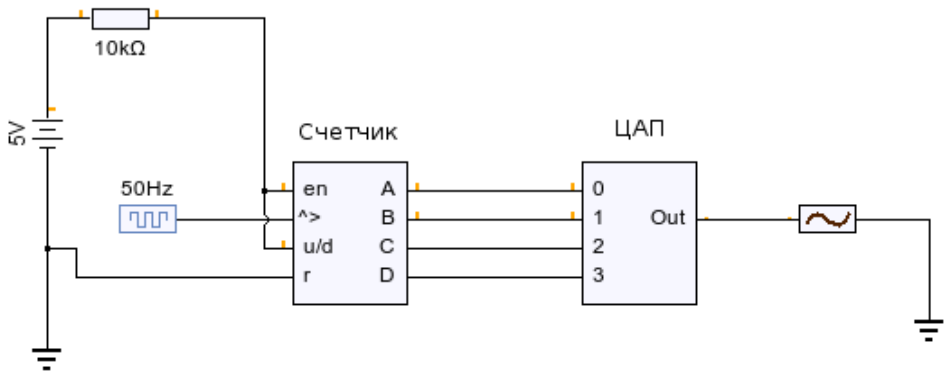


Рис. 6.1. Предварительная схема проекта

В окончательном виде она должна приобрести вид, при котором функции тактового генератора (50 Гц на рисунке) и счетчика возьмет на себя контроллер, а вторым базовым элементом станет цифро-аналоговый преобразователь. Работа этой

схемы в программе KTechlab выглядит следующим образом:

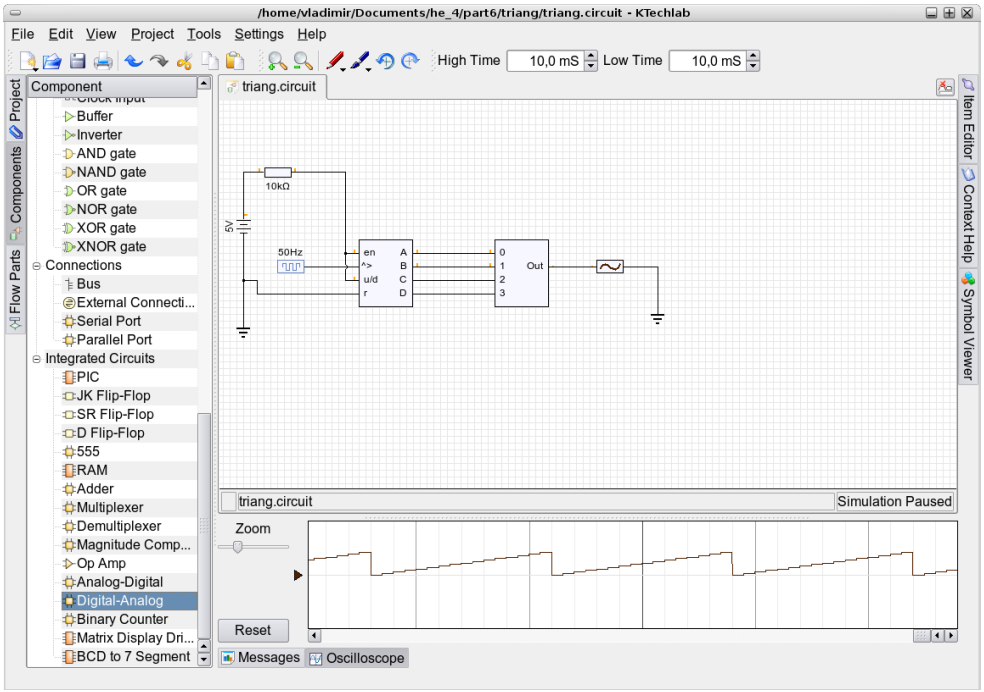


Рис. 6.2. Генератор пилообразного напряжения в программе KTechlab

Здесь представлен вид сигнала при «прямом» счете двоичного счетчика. Можете проверить, что присоединив вывод счетчика, обозначенный в программе, как «u/d» к общему проводу, вы получите «обратную пилу»:

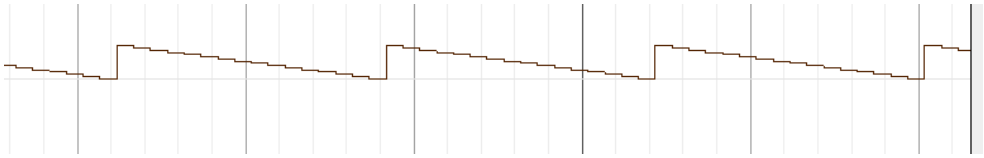


Рис. 6.3. «Обратное» пилообразное напряжение

Из этого можно сделать (поспешное?) предположение, что при смене направления счета в середине периода должно получиться «треугольное» напряжение на выходе

цифро-аналогового преобразователя.

Само предположение, возможно, и не поспешное, а вот реализация, которую я пытаюсь осуществить, явно поспешна: добавить четырех-входовый элемент И с инвертором на одном из входов, выход которого присоединить ко входу выбора направления счета.

Счетчик у меня «застревает» в середине отсчета, «качаясь» между «8» и «7». Попытаюсь, что-то изменить. Что-то изменить...

В итоге получается следующее:

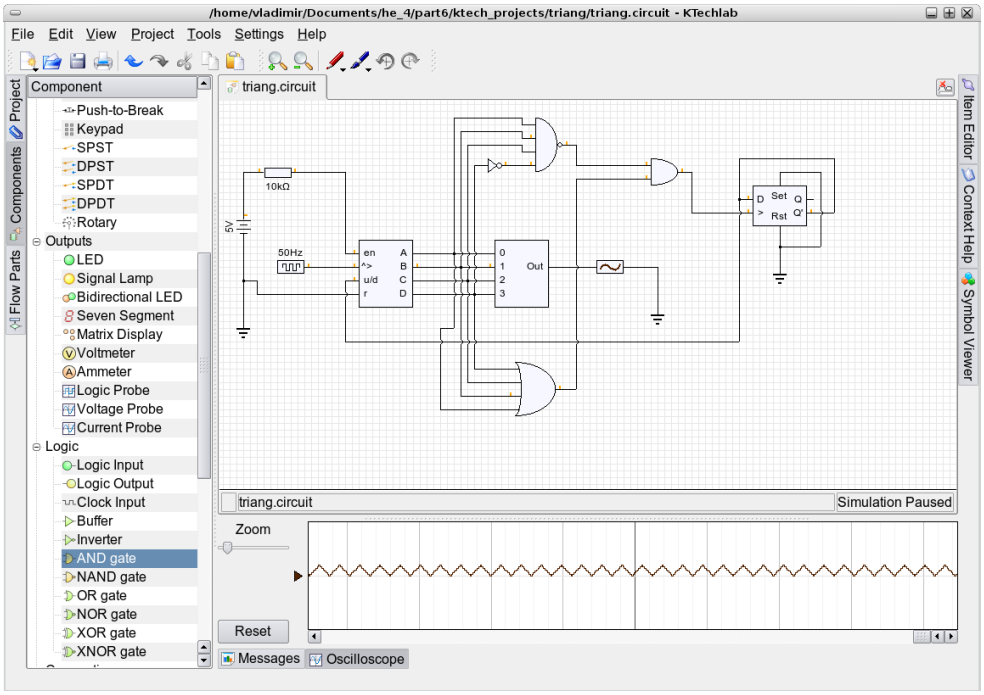


Рис. 6.4. Реализация треугольного напряжения на выходе генератора

Будем надеяться, что с идеей все в порядке, и можно подвести некоторые предварительные итоги.

- С помощью двоичного счетчика и цифро-аналогового преобразователя можно получить на выходе генератора «пилообразное» напряжение.
- Изменение направления счета меняет характер «пилообразного» напряжения.
- Если менять направление счета в середине периода, то можно получить

«треугольное» напряжение на выходе генератора.

- Реализация генератора с такими параметрами на базе цифровых микросхем возможна, но не столь проста, как хотелось бы.
- Дополнительные проблемы возникнут при разработке конструкции.

В последнем случае я имею ввиду необходимость добавления элементов перестройки частоты, необходимость создания шкал и т.п., если, напомню, создавать схему на элементах, подобных тем, что изображены на рисунке 6.1. Иное дело волшебная пара – микроконтроллер и программа на Gambas.

Продолжение работы над проектом «Генератор»

Так получилось, что на две недели пришлось оставить книгу. Если есть работа, которую можно «работать», ее нужно «работать».

Что до книги, то это время тоже не прошло зря, поскольку появилась возможность призадуматься. Вспомнить о тех сомнениях, что были в начале, и что появились в процессе работы. На сайтах радиолюбителей я встретил жалобы на то, что нынче, совсем не то, что раньше. Что не пойдешь в ближайший магазин купить нужные детали, в лучшем случае надо заказывать через Интернет, и ждать, когда все это получишь. Что компоненты стали настолько маленькими, что с ними трудно работать без специального оборудования, что без дорогостоящих приборов не подступишься к современным устройствам. Что молодежь, которая через радиолюбительство приходила в профессиональную деятельность, сегодня не хочет «горбиться» годы, изучая электронику и все с ней связанное, предпочитая зарабатывать деньги там, где и знать меньше нужно, да и работать проще, и, главное, денег много больше.

Если вы подумали, что все это я нашел на отечественных сайтах, то забудьте, речь идет о сетованиях американских любителей электроники (electronic hobbyists). Вот так.

Кроме того, я наткнулся на статью, где автор, много лет посвятивший популяризации Linux, принимавший участие в превращении этой операционной системы в то, чем она стала сегодня, задает себе тот же вопрос, который я задаю себе на протяжении всей книги: «А не напрасный ли это труд?».

Я не стремлюсь к обязательному изданию моих книг, довольствуясь тем, что размещаю их на сайте, который кто-то посещает и находит, надеюсь, нечто полезное для себя. Если хотя бы несколько человек из тех, что заходили на мой сайт за десять лет его существования, нашли для себя что-то полезное, если хотя бы несколько человек увлеклись электроникой, поверив мне, что это интереснейшее занятие, то книги написаны не зря.

Но есть одно, для меня существенное, но. Если число тех, кто увлекается электроникой или радиолюбительством, сокращается, если радиолюбители все больше увлекаются программированием, и уже не в состоянии, взяв паяльник в руки, спаять простую схему, то больше ли пользы или вреда я принесу? Не знаю.

Я уверен, что сегодня любителям следует учиться программированию, следует использовать компьютер в своей практике.

Когда-то любители сетовали на вытеснение электронных ламп из обихода. Им было комфортно, с чем я согласен, работать с лампами. Потом микросхемы вытеснили транзисторы. И работать с микросхемами не так удобно, как с транзисторами. Так и есть. Сегодня микросхемы уменьшаются в размерах и усложняются. Мне кажется, что разобраться с этими компонентами проще за компьютером. Но это перестает быть похоже на старое доброе радиолюбительство.

Словом, дописываю эту главу книги, как получится, и книгу на этом завершу. А

потом подумаю, следует ли начинать новую?

В плане завершения главы посмотрим, как устроить так, чтобы микроконтроллер менял частоту генератора по заданным в программе интерфейса значениям частоты или периода. Самый простой способ задания частоты в программе контроллера – использовать цикл при счете, где время счета определяется паузой, задаваемой программно. Что-нибудь в этом роде:

```
start: for (i=0; i<=255; i++) // Наращивание значения на выходе
       {       PORTA = i;
               for (k=0; k<period; k++); // Пауза
       }
       goto start;
```

Я не уверен, что можно передать значение period, что значение не должно быть константой. И, конечно, этот период лучше было бы организовать с помощью таймера или ассемблерной вставки, используя дополнительную программу PiKLoops. Второй вопрос, который я задаю себе: «Как остановить этот бесконечный цикл, чтобы изменить частоту?»

Последнее я попробую сделать с помощью прерывания при изменении состояния usart микроконтроллера. Мне не хочется включать осциллограф, устанавливать ЦАП и наблюдать реальный сигнал на выходе. Речь идет только об эксперименте, который определил бы возможность применения решения. Поэтому я воспользуюсь тем, что имею на макетной плате – тремя светодиодами, подключенными к выводам контроллера. После передачи по COM-порту чисел от 1 до 9 скорость их переключения должна измениться заметным образом, что и будет свидетельствовать о правильной работе схемы. Начальную же скорость переключения я постараюсь подобрать опытным путем удобной для наблюдения. Вот такой у меня план.

Итак, первое, что можно сделать, это подобрать подходящее для наблюдения время в цикле задержки, который определит период конечного сигнала. Программа будет выглядеть следующим образом:

```
int per;
int i;
int l;

void main() {
       PORTA = 0x0;   // Настройка портов A и B
       TRISA = 0x38;
       TRISB = 0xFA;
       per = 1000; // Цикл задержки для формирования периода
start:  for (i=0; i<8; i++) {
               PORTA = i; // Переключение трех выводов порта A.
               for (l=0; l <= per; l++); // Формирование периода сигнала.
       }
       goto start;
}
```

С помощью переменной per я постараюсь подобрать удобное для наблюдения время. Я использую компилятор SDCC, а при создании нового проекта воспользуюсь предлагаемым шаблоном, который позволяет задать нужную конфигурацию, записываемую по адресу 2007h. Напомню, что я использую внутренний тактовый генератор, а слово конфигурации в программе Piklab выглядит как 2118. В данный момент я не буду пытаться использовать симулятор, а сразу запишу полученный hex-код в микросхему. И, если не забыть настроить хотя бы порт A на вывод, как это сделал я, то изменение значения переменной per с 1000 на 10000 дает удобный для прямого наблюдения эффект. Попутно я разрешил свои сомнения по поводу условия работы цикла for, похоже, максимальное значение можно задать с помощью переменной. Формировать цикл задержки я намерен с помощью коэффициента, так что переменная будет выглядеть как per = 1000*k. Можно проверить это до выполнения следующего шага. Хотя это и выглядит нелепо, но займет немного времени.

Теперь мне хотелось бы проверить, будет ли работать идея при использовании прерывания. Шаблон имеет заготовку для кода подпрограммы прерывания. В подпрограмме я намереваюсь получить от программы-интерфейса значение коэффициента, назовем его «k», который позже определит переменную per. Меняя значение k от 1 до 9 я попытаюсь добиться такого же наблюдаемого эффекта.

В описании микросхемы PIC16F628A есть разделы, относящиеся к прерываниям. В первую очередь в описании приема данных по USART:

Рекомендованные действия при приеме данных в асинхронном режиме:

- Установить требуемую скорость передачи с помощью регистра SPBRG и бита BRGH.
- Выбрать асинхронный режим сбросом бита SYNC в ‘0’ и установкой бита SPEN в ‘1’.
- *Если необходимо, разрешить прерывания установкой бита RCIE (бит 5) в ‘1’ (PIE1=8Ch).*
- Если прием 9-битный, установить бит RX9 в ‘1’.
- Разрешить прием установкой бита CREN в ‘1’.
- Ожидать установку бита RCIF, или прерывание, если оно разрешено битом RCIE.
- Считать 9-й бит данных (если разрешен 9-разрядный прием) из регистра RCSTA и проверить возникновение ошибки.
- Считать 8 бит данных из регистра RCREG.
- При возникновении ошибки переполнения сбросить бит CREN в ‘0’.

Здесь упоминается регистр PIE1. Обратимся к описанию этого регистра:

Регистр PIE1 (адрес 8Ch), доступен для чтения и записи, содержит биты

разрешения периферийных прерываний.

Примечание. Для разрешения периферийных прерываний необходимо установить в '1' бит PEIE (INTCON<6>).

Биты регистра:

EEIE CMIE RCIE TXIE нет CCP1IE TMR2IE TMR1IE

бит 7 (EEIE – Разрешение прерывания по окончанию записи в EEPROM данных):

1 = прерывание разрешено
0 = прерывание запрещено

бит 6 (CMIE – Разрешение прерывания от компараторов):

1 = прерывание разрешено
0 = прерывание запрещено

бит 5 (RCIE – Разрешение прерывания от приемника USART):

1 = прерывание разрешено
0 = прерывание запрещено

бит 4 (TXIE – Разрешение прерывания от передатчика USART):

1 = прерывание разрешено
0 = прерывание запрещено

бит 3 (Не используется, читается как '0').
бит 2 (CCP1IE – Разрешение прерывания от модуля CCP1):

1 = прерывание разрешено
0 = прерывание запрещено

бит 1 (TMR2IE – Разрешение прерывания по переполнению TMR2):

1 = прерывание разрешено
0 = прерывание запрещено

бит 0 (TMR1IE – Разрешение прерывания по переполнению TMR1):

1 = прерывание разрешено
0 = прерывание запрещено

И еще один регистр, упомянутый выше.

Регистр INTCON (адрес 0Bh, 8Bh, 10Bh или 18Bh), доступен для чтения и записи, содержит биты разрешений и флагов некоторых источников прерываний.

Примечание. Флаги прерываний устанавливаются при возникновении условий прерываний вне зависимости от соответствующих битов разрешения и бита общего разрешения прерываний GIE (INTCON<7>).

Биты регистра:

GIE PEIE T0IE INTE RBIE T0IF INTF RBIF

бит 7 (GIE – Глобальное разрешение прерываний):

1 = разрешены все немаскированные прерывания
0 = все прерывания запрещены

бит 6 (PEIE – Разрешение прерываний от периферийных модулей):

1 = разрешены все немаскированные прерывания периферийных модулей
0 = прерывания от периферийных модулей запрещены

бит 5 (T0IE – Разрешение прерывания по переполнению TMR0):

1 = прерывание разрешено
0 = прерывание запрещено

бит 4 (INTE – Разрешение внешнего прерывания INT):

1 = прерывание разрешено
0 = прерывание запрещено

бит 3 (RBIE – Разрешение прерывания по изменению сигнала на входах RB7:RB4 PORTB):

1 = прерывание разрешено
0 = прерывание запрещено

бит 2 (T0IF – Флаг прерывания по переполнению TMR0):

1 = произошло переполнение TMR0 (сбрасывается программно)

0 = переполнения TMR0 не было

бит 1 (INTF – Флаг внешнего прерывания INT):

1 = выполнено условие внешнего прерывания на выводе RB0/INT (сбрасывается программно)
0 = внешнего прерывания не было

бит 0 (RBIF – Флаг прерывания по изменению уровня сигнала на входах RB4:RB7 PORTB):

1 = зафиксировано изменение уровня сигнала на одном из входов RB7:RB4 (сбрасывается программно)
0 = не было изменения уровня сигнала ни на одном из входов RB7:RB4

Я выделил то, что по моему мнению относится к задаче. Как мне кажется программа может выглядеть следующим образом:

```
/* ------------------------------------------------------------------------ */
/* Template source file generated by piklab */
#include <pic16f628a.h>

/* ------------------------------------------------------------------------ */
/* Configuration bits: adapt to your setup and needs */

typedef unsigned int word;

word at 0x2007 CONFIG = _WDT_OFF & _PWRTE_OFF & _INTOSC_OSC_NOCLKOUT &
_MCLRE_OFF & _BOREN_OFF & _LVP_OFF & _DATA_CP_OFF & _CP_OFF;

int i;
int k;
int l;
int per; // Переменная, определяющая с коэффициентом k период.

unsigned char getch() // Получение байта
{
        while(!RCIF)   // Устанавливается, когда регистр не пуст
                continue;
        return RCREG;
}

void isr() interrupt 0 {
/* interrupt service routine */
    /* << insert interrupt code >> */
        k = getch() - 0x30;
        return;
}
```

```
void main() {
    /* << insert code >> */
        PORTA = 0x0;   // Настройка портов A и B
        CMCON = 0x7;
        TRISA = 0x38;
        TRISB = 0xFA;
        RCSTA = 0x90;  // Настройка приемника
        TXSTA = 0x6;   // Настройка передатчика
        SPBRG = 0x68;  // Настройка режима приема-передачи
        PORTB = 0xFE;  // Выключаем драйвер RS485 на передачу
        PIE1 = 0x20;   // Разрешение прерывания от приемника USART.
        INTCON = 0xC0; // Разрешение перефирийных прерываний.
        TXEN = 0; // Выключаем передачу
        CREN = 1; // Включаем прием.
        k = 1;

start:  per = 1000*k;
        for (i=0; i<8; i++) {
                PORTA = i;
                for (l=0; l <= per; l++);
        }
        goto start;
}
```

Получив после трансляции cod-файл для запуска симуляции, я хочу проверить работу программы в gpsim. Выделив переменные k и per, я добавляю их в окно наблюдения. Первый ввод с клавиатуры в окне USART дает мне:

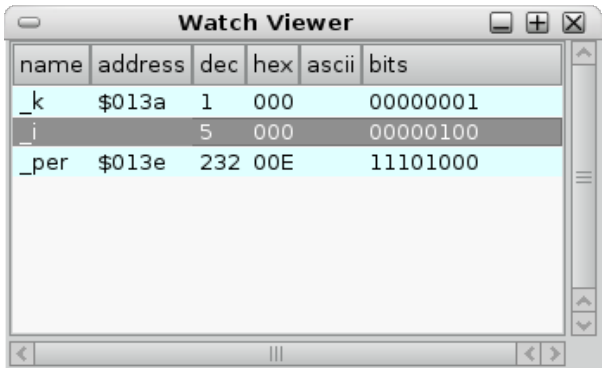


Рис. 6.5. Наблюдение переменных при задании коэффициента 1

На макетной плате выводы, к которым я присоединяю светодиоды, меняются с некоторой частотой, которая меняется, если я ввожу с клавиатуры коэффициент 9 при активизации окна USART, а переменные меняются:

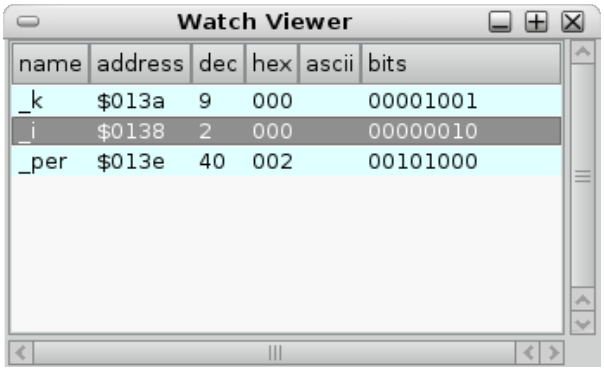


Рис. 6.6. Наблюдение переменных при вводе коэффициента 9

Все, казалось бы, хорошо. Но после программирования микросхемы и переноса ее на реальную макетную плату я не вижу заметных изменений в поведении светодиодов. Я пытаюсь работать с графическим интерфейсом. Затем пытаюсь проверить, запуская пример работы с последовательным портом. Результат одинаков. То есть, я его не вижу.

Пора определиться, где возникла проблема? В программе, написанной для микроконтроллера, или в программе графического интерфейса?

Самый простой способ отделить «зерна от плевел» – использовать готовую программу для работы с COM-портом. Что я и делаю. И она позволяет мне увидеть изменение в поведении светодиодов. Из чего следует, что проблемы возникают при передаче значения от моего графического интерфейса к контроллеру.

Попробуем разобраться с этим.

Для отправки значения в COM-порт я использовал функцию (или метод) PRINT. Есть другая возможность – WRITE. Заменив в программе интерфейса обработку нажатия клавиши «Запись», с помощью которой я отправляю нужное мне значение в последовательный порт, следующим образом:

```
PUBLIC SUB Button1_Click() ' Клавиша "Запись"
       WRITE #SerialPort1 TextBox1.Text
END
```

я получаю окончательный результат, который был нужен. Теперь контроллер послушно меняет скорость переключения светодиодов.

Я не придумал в назидательных целях последнюю проблему. Все вышло само-собой. Но получилась яркая иллюстрация того, что последнее слово остается за физической реализацией. Как бы ты ни был уверен, что все правильно, только собрав устройство и проверив его работу, можно определить (да и то не всегда), что работа завершена, а устройство сделано правильно и работает.

Завершение

У меня не было намерения описывать конкретную схему. Я собирался рассказать о среде программирования Gambas, несколько больше, чем сделал ранее, больше рассказать о системе программирования PIC-контроллеров Piklab и отладчике gpsim. Мне хотелось показать, что сегодня почти любая разработка так или иначе связана с программированием, и что любителям полезно и нужно учиться программировать, благо есть такие великолепные программные средства, как Gambas, Piklab, gpsim. Они полнофункциональны и свободны, доступны не только профессионалам, но и любому и каждому.

Вместе с тем, описанные выше подходы могут найти практическое применение, например, при создании смешанных генераторов, когда одновременно нужно иметь взаимосвязанные логические (или импульсные) и аналоговые сигналы. Аналогично пилообразному напряжению, можно получить синусоидальное, если заполнить массив числами, соответствующими синусоидальной функции, и использовать их для управления ЦАП. Аналогично можно получить ступенчатое приближение любого нужного сигнала. Или можно использовать генератор пилообразного напряжения для управления варикапом, можно найти, я полагаю, множество интересных применений даже для такой простой схемы.

Сомнения, появившиеся у меня, когда я только подумывал о написании этой книги, не развеялись, как я надеялся, в процессе работы, но только усилились. Кроме первоначальных сомнений по поводу моего непрофессионального рассказа о программировании, появились сомнения, которые касаются отношения любителей моего поколения к компьютерам и всего, что с ними связано. И любители электроники, и радиолюбители сетуют на то, что молодежь все больше замыкается в программировании, забывая о физической реализации своих проектов. И это справедливо. Я не так плохо умею пользоваться паяльником, но я пользовался макетом, который спаял, по моим подсчетам года два назад, для проверки всего, что потребовалось при написании этой книги, и предыдущей, и предыдущей. Мне не понадобилось что-то паять, если не считать простой схемы программатора, которая работала бы с Piklab. Справедливые сетования.

Linux. Я использую эту операционную систему несколько лет, не вижу какой-либо разницы с точки зрения пользователя между ней и Windows, разве, что она удобнее во многих отношениях. Когда я начинал пользоваться Linux, мне не хватало, например, англо-русского словаря, чтобы проверить, правильно ли я понимаю документацию, с которой работаю. Но очень быстро нашелся Stardict, который ничем не хуже Lingvo. Не хватало программ САПР (EDA), но со временем отыскались и они. Я говорил выше, когда рассказывал о том, как просто можно в Gambas создать полезную программу для того, чтобы научиться печатать вслепую, что постараюсь овладеть печатью вслепую, используя ее. Но есть и готовая программа, о которой я тоже упоминал. И все, что написано в этом послесловии латиницей, написано мной

вслепую, хотя, боюсь, я не уделял даже 10-15 минут каждодневным тренировкам – не случилось, не было настроения.

Мне нравится работать в Linux, я охотно пишу об этой операционной системе и программах для нее. Но пользователей Linux мало, видимо, благосостояние наших граждан так возросло, что им не нужны бесплатные программы, как бы ни были они хороши. Есть о чем задуматься.

В результате я не столько закончил работу над книгой, сколько поспешно свернул работу над ней. Это самый мой неудачный проект. Мне жаль. Мне есть о чем подумать. Чем я и собираюсь заняться.

Конспекты

Gambas дружелюбен к пишущим на VB, но используя Linux

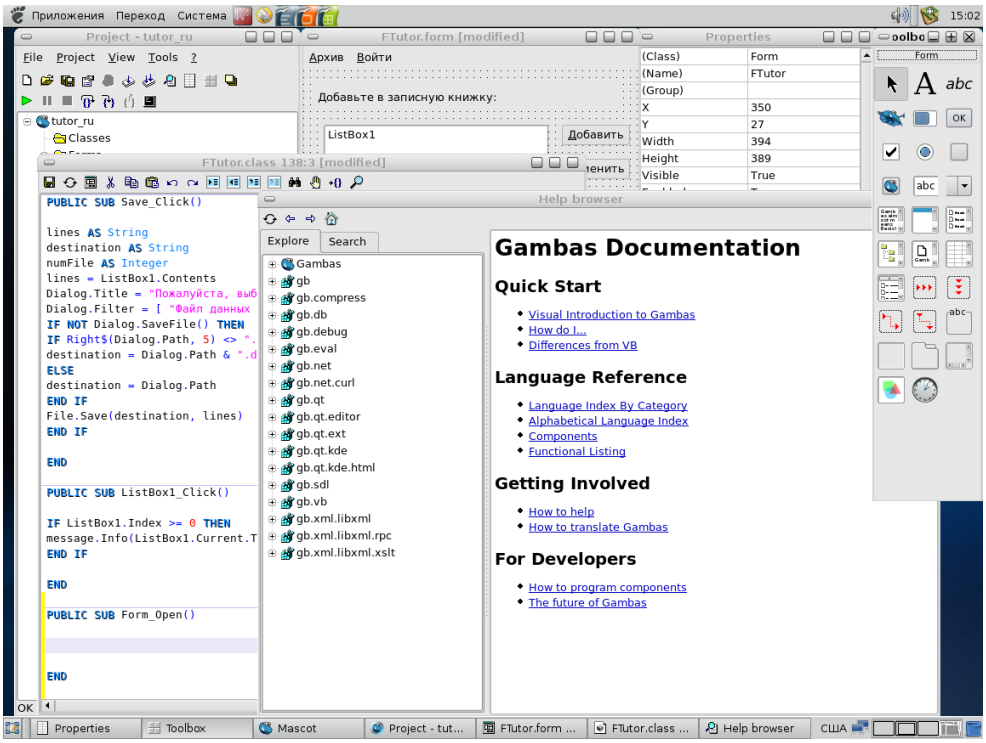
«Феноменальное количество ошибок и несообразностей, делающих Visual Basic столь очаровательным, подвигли меня начать этот проект». Вот как Benoit Minisini, 30-летний француз, живущий в пригороде Парижа, начинает описание своего проекта, названного Gambas.

Gambas - это замена VB для Linux. Не клон, поскольку Benoit'y никогда не нравилась реализация Microsoft: «Кажется, Microsoft осознает низкое качество ее языка, поскольку VB.Net не поддерживает обратной совместимости с предыдущими версиями Visual Basic. Я думаю, они выбросили исходный код интерпретатора Visual Basic, и что VB .Net - это только .Net компилятор реального времени, чей синтаксис похож на Visual Basic».

Так что, Gambas предназначен быть лучше Visual Basic.

Benoit взял от VB то, что посчитал действительно полезным: язык BASIC и легкий IDE. Он также взял несколько идей от Java, создав легкий и мощный язык. Насколько мощный? Полный IDE был создан в самом Gambas, а индикатор уровня его готовности даже не достигал 1.0.

В этой статье мы постараемся описать текущий статус Gambas, и как он соотносится с VB. Мы сравним снимки экрана и исходный код, так что вы можете увидеть полную картину.



множество IDE без языка BASIC.

Gambas имеет уникальную смесь возможностей, которые помогут тысячам разработчиков на VB перебраться с Windows на Linux. Следовательно, Gambas может принести больше приложений и пользователей в Linux.

И, наконец, разработка компонент Gambas означает для всякого наличие возможности написать GUI компонент, базируемый на GTK+ с тем же интерфейсом, что и QT, так что каждая программа Gambas может затем переключаться между QT GUI и GTK+ GUI.

Начала BASIC

Одна из первых программ, которые вы можете непременно найти в любом языке программирования, это достопочтенная «Hello World».

Hello VB World

1. Начинаем с пустой формы (form)
2. Размещаем клавишу команды - Command Button (Command1) и этикетку (Label1) на форме
3. Пишем код:

```
Sub Command1_Click()
        Label1.Caption = "Hello World!"
End Sub
```

Hello Gambas World

1. Начинаем с пустой формы (form)
2. Размещаем клавишу - Button (Button1) и этикетку (Label1) на форме
3. Пишем код:

```
SUB Button1_Click()
        Label1.Text = "Hello World!"
END
```

Если вы не хотите создавать программу, основанную на форме (form-based), вы можете начать проект без форм и написать код:

```
SUB main()
        PRINT "Hello World!"
END
```

Есть множество прелестных простых вещиц, которые вы можете сделать с Gambas - и еще нечто, что не может Visual Basic! (В VB есть метод Print, но он обращается только к отладчику и GDI объектам, подобным принтеру, форме или вместилищу картинки.)

Отличия от VB

Поскольку Gambas не предназначался быть клоном Microsoft Visual Basic, он все еще BASIC, и есть много сходства между этими двумя языками, и много связей между функциональными возможностями. Есть, возможно, больше сходства, чем различий, но вы можете просто скопировать свои VB проекты и рассчитывать на компиляцию под Gambas.

Из раздела DifferencesFromVB:

Не специфические для языка различия

- VB вставляет код класса для каждого объекта формы в тот же самый файл, где находится определение формы. Gambas держит их раздельно в .form и .class файлах.
- Расширения файлов:

VB	Gambas	Тип файла
.vbp	.project (только .project, один на директорию)	Файл определения проекта
.bas	.module	Модуль
.cls	.class	Файл класса
.frm	.form	Файл определения формы
.frx	(все, что хотите)	Двоичные файлы ресурсов

- Проекты Gambas определены, как директория с файлом .project в ней, и все файлы находятся в этой директории. VB может иметь множество файлов проекта в каждой директории, и может вытащить одинаковые исходные файлы из разных директорий в разных проектах, что имеет и полезную и вредную сторону.
- Измерения экрана в VB представлены в «twips», единицах 1/1440 дюйма; в Gambas они сделаны в фактических пикселях.
- Управление формой в программах Gambas частное (private) по определению. Вы можете изменить это, войдя в диалог свойств проекта (Project Properties) и установив флажок «Make Form Controls Public».

VB имеет это, Gambas нет

- Вы не можете редактировать код в режиме аварийной остановки (Break mode) в Gambas; вам нужно вначале закончить выполнение программы.
- Параметры функции и процедуры передаются *по значению* (*by value)*. Они не могут быть переданы *по ссылке* (*by reference)*, как в Visual Basic. Заметьте, что VB передает параметры *по ссылке*, если вы не используете его ключевое слово `ByVal`, так что будьте внимательны, когда вы пытаетесь портировать VB проект.
- В Gambas нет такой штуки, как глобальная переменная расширения проекта (project-wide global variable). В качестве обходного пути рассмотрите создание класса, называемого Global, и объявите ваши глобальные переменные, как статические общие переменные (static public) в этом классе, а затем ссылайтесь на них, как на глобальные имена переменных (Global.variablename) в своем проекте. Это остается плохой практикой программирования, но во всяком случае они будут идентифицированы, как глобальные переменные, когда бы вы ни использовали их ;)
- Пока вы не включили «Option Explicit» (явный выбор) в VB модуль, вы не нуждаетесь в объявлении переменных до их использования. Gambas ведет себя так, как если бы «Option Explicit» было всегда включено, что создает намного лучший код при совсем небольшой дополнительной работе.
- Нет прямого Gambas эквивалента для «Index» свойства управления VB формой. Вы можете легко создать массивы управления, но вы должны сделать это в коде. Нет в настоящее время (0.62) пути сделать это графически. Таким образом, когда вы копируете управление и вставляете его обратно в форму, оно приходит иначе, чем приглашение для вас к созданию массива управления, автоматически переименовывая скопированное управление с подходящим именем.
- Вы не можете в настоящее время создавать прозрачные этикетки (transparent labels) в Gambas; фон всегда непрозрачный.
- В Gambas событие перемещения мышки (MouseMove event) обнаруживается только, когда клавиша мышки отжата, что означает, что вы совсем не можете использовать ее для традиционного эффекта выделения мышкой (другого, а не стандартного средства, связанного с каждым управлением). Исключение составляет управление *DrawingArea*, которое имеет свойство трассировки (Tracking property), что позволяет получить событие движения мышки, даже если клавиша нажата.

Gambas имеет это, VB нет

- В отличие от VB, вам не нужно компилировать в GUI поддержку, если вы хотите написать приложение командной строки в Gambas. Только снимите «gb.qt» компонент в свойствах проекта (Project Properties) и будьте уверены,

что определили Sub Main.

- Gambas придерживается концепции *control groups*, которая позволяет вам поддерживать события из любого числа разных мест управления с единственной поддержкой подпрограммой. Это удаляет излишний код, и может быть использовано для выполнения разных вещей, которые делает индексное управление в VB, и еще некоторых, которые VB сделать не может.
- Тогда как VB делает невозможным запустить программу синхронно и получить ее вывод без изучения того, как сделать вызовы API (Shell просто запускает программу в фоновом режиме), Gambas позволяет вам сделать это, используя SHELL и EXEC, управление процессами при старте с использованием Process object, и даже читая из и записывая в них, позволяя вам легко добавить функциональности со вспомогательными приложениями. Это создает условия для невероятно легкого написания внешнего интерфейса для почти любой процедуры командной строки.
- Вы можете сделать все, что выше, так же успешно со специальными файлами и Unix устройствами, такими как последовательные или параллельные порты. Используйте /proc файловой системы для написания RAID монитора, например, или используйте именованные конвейеры (pipes) для получения множественных каналов информации от прикладной части программы на любом другом языке.
- Для получения окон нестандартной формы вам достаточно установить ME.Mask свойство текущего окна к картинке, которая имеет прозрачные области. VB требует вызова API и чуть-чуть работы.
- Вы можете создавать управление и меню динамически, только обрабатывая их инструкцией NEW.
- В Gambas вы можете создавать и отображать столько много копий формы, сколько захотите.
- Вы можете вставить Gambas форму в другую: когда вы обрабатываете первую, задайте вторую, как родительскую.
- И, конечно, Gambas - это Free Software, чье расширение окружения написано в нем самом, позволяя вам подгонять его к более серьезному использованию, чем тренировки в BASIC.

Усиливая мощь Linux

«Это Unix философия: Пишите программы, которые делают одну вещь, но делают ее хорошо. Пишите программы для совместной работы. Пишите программы для поддержки текстовых потоков, поскольку это универсальный интерфейс» -- Doug McIlroy.

Одно из основных преимуществ Gambas над VB в том, что Gambas построен в соответствии с философией Unix. Это значит, вы можете усилить мощь Linux и его

тысяч средств вместо написания кода.

Давайте, рассмотрим некоторые примеры.

- примеры управления другими программами с использованием конвейеров (pipes)
- cd burner? ()
- sample player? (mpg123?)
- загрузка файлов из Интернета? (то есть, lynx -source url)

Приумножая ваши текущие знания

Если вы всегда программировали на BASIC, вы почувствуете теплое ощущение от знакомого вам, когда начнете программировать в Gambas.

VB программисты почувствуют полный комфорт с самого начала.

Чтобы понять, что Gambas - это не VB, и VB программистам нужно быть готовыми к некоторым различиям. Но большинство изменений существуют по определенным причинам: чтобы сделать язык лучше. (Важно помнить, что Gambas позаимствовал некоторые хорошие идеи у Java и других языков программирования.)

С Gambas любой может начать программировать графическое приложение сразу, и это приведет больше программистов, и принесет больше приложений в GNU/Linux.

Конвертирование вашего существующего кода

Если вы VB программист, и не чувствуете особого счастья от последних изменений платформы (и сопутствующей цены), вы можете подумать о конвертировании ваших приложений в Gambas.

Разработчики Gambas уже сделали небольшой скрипт, в настоящий момент в версии 0.1, который конвертирует VB формы в Gambas.

Vb2Gb было написано на Perl по причине больших возможностей по обработке текста в этом языке, и вскоре будет реализовано в Gambas.

В VB описание формы и исходного кода смешаны в едином файле. Gambas разделяет это на два файла: описание формы и модуль класса.

Есть также несколько имен управления и свойств, которые мы можем конвертировать автоматически. Например, в VB мы имеем «CommandButtons», тогда как в Gambas (или QT?) они названы просто «Buttons»; свойство «Caption» названо «Text» в Gambas.

Резюмируя

Gambas - это самое близкое к VB в мире Linux. Проект не был изначально предназначен заменить миллионы строк VB кода, но предоставляет быстро прогрессирующий инструмент, базирующийся на языке BASIC. Тем не менее,

Gambas уже представляет собой инструмент для легкой миграции с собственных приложений VB.

Gambas в постоянном совершенствовании. Его текущая версия рассматривается как alpha программа, поскольку некоторые возможности еще определяются, но Gambas уже используется в таких сложных приложениях, как его собственный IDE. Нет смысла спешить с релизом 1.0, и есть цель - сделать хорошую вещь сразу.

Gambas имеет сетевой график. Некоторые из новых возможностей, которые ожидаются в следующих выпусках:

- Компонент Network
- SDL компонент
- Персистентная система объектов
- Проектировщик отчетов
- Perl'o-подобный компонент регулярных выражений

Согласно Daniel Campos, создателю компонента Network, он достаточно стабилен, чтобы именоваться 'beta' версией, и уже предоставляет некоторые интересные функциональные возможности:

- DNS/NIS клиент
- Socket клиент : для начала TCP или UNIX socket соединения.
- Socket Server : для обслуживания TCP или (UNIX, пока нет) sockets.
- Datagram Client/Server : UDP sockets
- SerialPort : класс для управления последовательными устройствами (RS-232, и т.д.)

TO-DO: завершить статью с необузданным оптимизмом

Разработка приложений в Gambas

Руководство и пример выполнены в Gambas

Резюме: Мы собираемся создать простую программу в Gambas. Мы научимся поддерживать события и некоторые типы и техники для работы с этим удивительным инструментом.

Моя благодарность: Daniel Oxley и Dave Sharples. Они просмотрели и поправили мой бедный английский перевод с испанского.

Загружаемый исходный код (комментарии, имена переменных и т.д.) сделан на испанском. Однако код, показанный на этих страницах, транслировался.

David Asorey Alvarez. February de 2005.

Введение

Gambas - это IDE (Integrated Development Environment - интегрированное окружение разработки), ориентированное на RAD (Rapid Applications Development - быстрая разработка приложений) подобное популярным патентованным программам Microsoft Visual Basic или Borland Delphi.

В Gambas возможно разрабатывать GUI программы (Graphical User Interface - графический интерфейс пользователя) очень быстро, поскольку IDE включает конструктор форм, редактор кода, обозреватель кода, интегрированную систему помощи и т.д.

Такого рода инструменты весьма характерны для мира Microsoft Windows, на платформе Linux есть только несколько, или они на ранних стадиях разработки. Мы находим несколько, подобно Kdevelop, Kylix или VDK Builder.

Есть добрая традиция и обыкновение в мире Linux/Unix использовать множество различных инструментов, каждый из которых специализируется на решении единственной конкретной задачи (например, компилятор, редактор, отладчик - каждый из них используется отдельно). По этой причине программные IDE относительно внове для пользователей Linux/Unix.

Есть множество программистов и разработчиков, кто пользуется такого рода интегрированными инструментами, поскольку они пришли с других платформ или

они чувствуют себя комфортнее с IDE.

По словам автора, Benoit Minisini: «*Gambas нацелен на то, чтобы дать вам возможность создавать мощные программы легко и быстро. Но чистота программирования остается на вашей совести...*». Язык программирования, используемый в Gambas - это версия «старого» BASIC. Возможно, что кто-нибудь будет удивлен этим выбором, поскольку BASIC кажется слишком простым и ограниченным. Должны напомнить, что одной из целей Gambas является содействие программированию не программистов.

Цель данного руководства - дать введение в Gambas IDE, и разработать простую программу с его помощью, но мы подразумеваем, что читатель в какой-то мере знаком с программированием, и что термины подобные функции, событию или переменной ему знакомы. Gambas имеет встроенную систему помощи, предоставляющую и вводную, и более детальную документацию.

Версия Gambas, использованная в данном руководстве - это 1.0−1. Домашняя страница Gambas - http://gambas.sourceforge.net

Загрузите исходный код примера программы: agenda.tar.gz

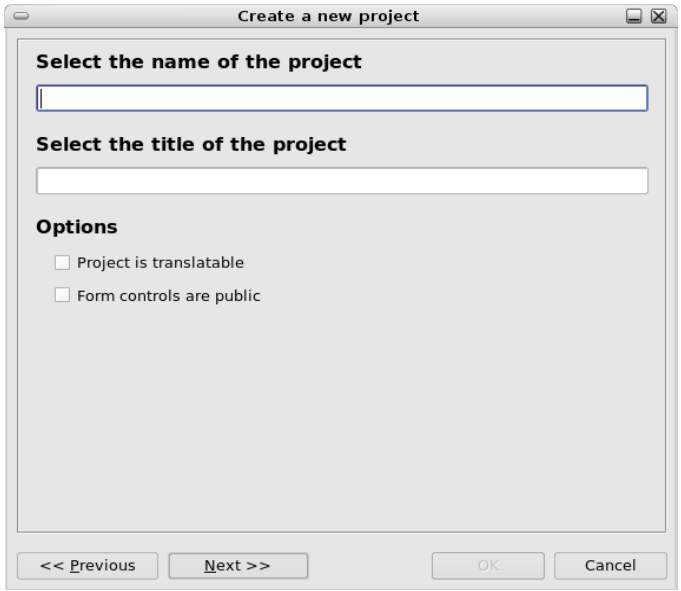
Первые шаги

В Gambas help есть документ озаглавленный «Наглядное введение в Gambas - Visual Introduction to Gambas», который рассказывает о программе, ее различных инструментах, окнах и меню. Мы не собираемся повторять эту информацию в данном руководстве.

Мы постараемся разработать законченную программу в Gambas с самого начала, и разберемся с требованиями и проблемами по мере их появления.

Наша программа будет приложением типа записной книжки или памятки. Мы сможем добавлять и удалять записи, модифицировать существующие заметки. Они будут сохраняться и читаться из файла.

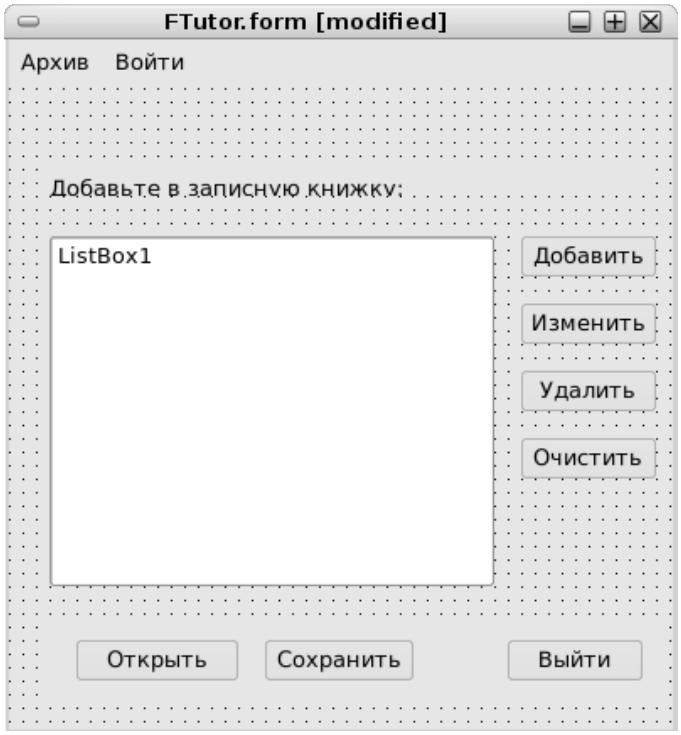
Как только Gambas откроется, мы выберем «New project - новый проект» из меню. Определим его, как graphical project, графический проект, в мастере проектов и мы затем должны обеспечить некоторую информацию, такую как имя проекта и заголовок проекта:



Есть две опции: «Project is translatable - проект транслируемый» и «Form controls are public - управление формами общее». Мы оставим эти опции не выбранными и нажмем клавишу **Next**.

Последняя задача в мастере проектов - выбрать директорию, где будет сохраняться проект. После чего откроется основное Gambas IDE. В главном окне проекта следует щелкнуть правой клавишей мышки на **Forms** и выбрать «New form - новая форма».

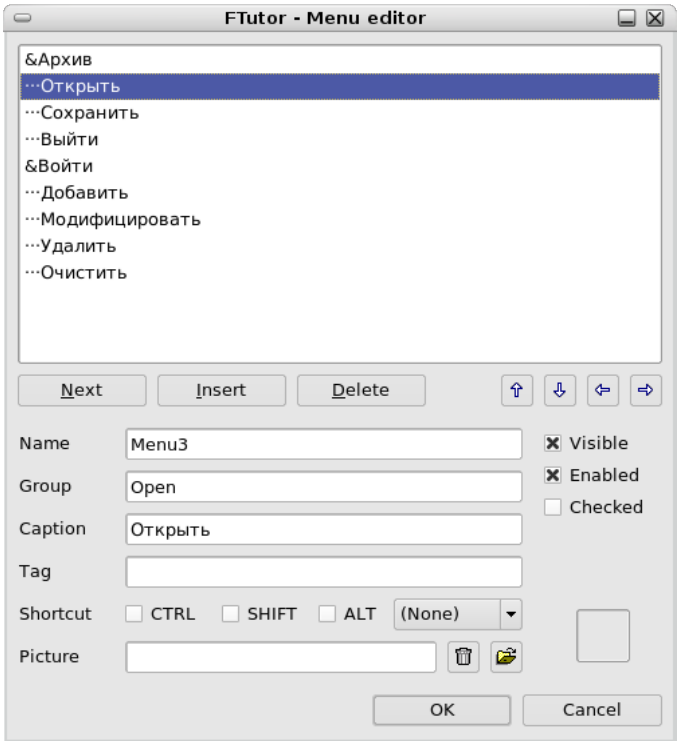
Мы собираемся разработать основную форму, которая содержит управление ListBox и несколько клавиш (open - открыть, save - сохранить, exit - выйти, add - добавить, modify - изменить, delete - удалить, ...). Нечто, похожее на это:



Можно видеть несколько общих средств управления: Label - надпись, ListBox - область списка и несколько клавиш. Мы добавляем каждое средство управления в форму, выбирая их на инструментальной панели (Toolbox) Gambas и «рисуя» контур управления на форме. Пожалуйста, отметьте, что клавиши **Open** - открыть, **Save** - сохранить и **Exit** - выйти расположены на **Panel** - панель, которая должна быть добавлена на форму первой и не непосредственно в основное тело формы.

Если мы хотим задать горячие клавиши, мы должны предпослать символ «амперсанд» (&) выбранной букве в тексте клавиши. Например, О&ткрыть, &Сохранить

Мы можем создать и отредактировать основное меню нашей программы, щелкнув правой клавишей мышки на свободном пространстве формы и выбрав в выпадающем меню «Menu editor - редактор меню»:



После создания пунктов меню вы можете заметить, что среди типичных свойств, как имя, надпись, и т.д., есть также свойство, названное «Group - группа», клавиши формы имеют это свойство Group, как вы можете увидеть, выбрав клавишу и просмотрев ее свойства в окне «Properties - свойства». Эта опция очень интересна, поскольку мы можем ассоциировать один и тот же код с разными средствами управления, имеющими одинаковые функции. Например, пункт меню «Open - открыть» и клавиша **Open** должны выполнить одну и ту же задачу: открыть файл на диске. Используя свойство Group, мы напишем код для этой задачи только один раз.

В нашей программе мы объединим клавишу и пункт меню Open в группу, названную Open, Save - сохранить меню и клавишу в группу, названную... Save, и т.д.

Теперь, если мы дважды щелкнем по клавише или одноименному пункту меню, редактор кода откроется и курсор окажется на заголовке подпрограммы, названной общим именем группы, выбранном для средств управления, ассоциированных с ней. Например: PUBLIC SUB Open_Click() , где Open - это свойство Group, названной Open, определенное ранее.

Поддержка событий

Программы с графическим пользовательским интерфейсом обычно «event driven - движимые событиями». Что означает, когда пользователь «заставляет что-либо произойти» в окне приложения, генерируется событие. Это событие может быть связано с функцией или подпрограммой, которые отвечают на действие пользователя.

Пример: если пользователь «щелкнул» по средству управления, генерируются некоторые события - MousePress, при нажатии клавиши мышки, MouseRelease, при отпускании, и, наконец, Click (щелчок), как глобальное действие. Если пользователь сделал двойной щелчок, тогда генерируется событие DblClick. Не все средства управления отвечают на события или генерируют события, нелепо говорить о событии Resize in a button (уменьшить размер клавиши), поскольку это событие обычно генерируется, когда мы изменяем размер окна (управление Form).

В Gambas мы будем редактировать код процедуры для события (1), подобный этому:

```
PUBLIC SUB Control_Event
```

Здесь Control - это имя средства управления, где генерируется событие, а Event тип события. Некоторые средства управления имеют предопределенные события. Наиболее полезное предопределенное событие для клавиши, например, это Click.

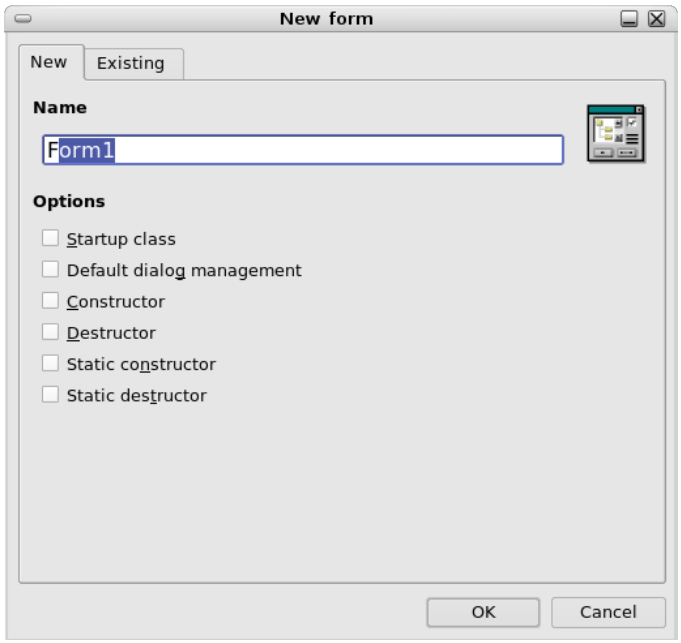
В Gambas при двойном щелчке по любому средству управления редактор кода открывается и курсор позиционируется на объявлении подпрограммы для предопределенного события. Есть исключение, если управление ассоциировано с группой действия (свойство Group определено), редактор кода показывает подпрограмму для группы действия, как упоминалось выше.

Конструирование форм

- Следует понимать при конструировании форм:
- Экран пользователя может отличаться от нашего: другое разрешение,

оконный менеджер и/или размер шрифта. Не пытайтесь предельно использовать все пространство, этикетки. клавиши и другие средства управления могут обрезаться или быть неразборчивы.

- Хорошая практика оставлять основное окно программы «растягиваемым». В Gambas обратите внимание на свойство формы Border - рамка. Не устанавливайте его в Fixed - фиксировано.
- При создании новой формы есть несколько интересных опций:



Опции, относящиеся к конструктору и деструктуру (constructor и destructor), полезны для инициализации и завершения задачи в окне.

Генерируются следующие определения:

```
' Gambas class

file PUBLIC SUB _new()

END

PUBLIC SUB _free()
```

```
END
PUBLIC SUB Form_Open()
END
```

Если мы выберем «Static constructor - статический конструктор» и/или «Static destructor - статический деструктор», определения станут:

```
' Gambas class file
STATIC PUBLIC SUB _init()
END
STATIC PUBLIC SUB _exit()
END
PUBLIC SUB _new()
END
PUBLIC SUB _free()
END
PUBLIC SUB Form_Open()
END
```

Используя эти процедуры, мы можем изменить процесс открывания закрывания окна. Мы можем инициализировать средства управления, переменные и т.д. В процедуре, объявленной как STATIC, процедура имеет доступ только к STATIC переменным.

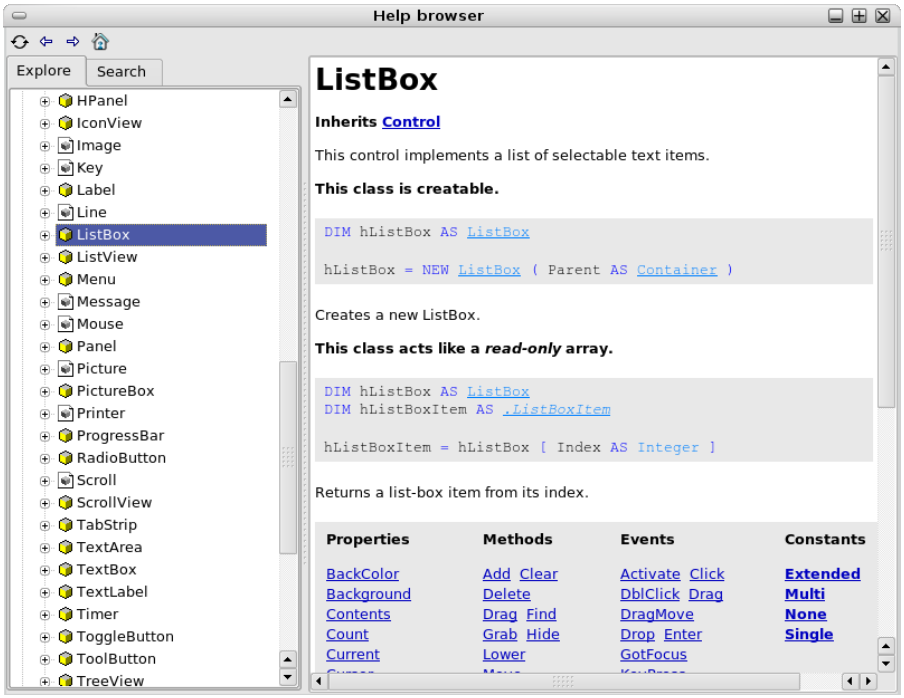
Прыжок в ...

Наша форма полностью разработана. Пришло время написать код для управления.

Первая задача - обеспечить некоторую функциональность управлению. Мы собираемся поработать с клавишами (и равнозначными пунктами меню) Add - добавить, Modify - изменить, Delete - удалить и Clean – очистить.

Действие Clean

Эта клавиша должна удалять все записи в ListBox. Для выполнения этой задачи мы поищем в системе помощи (help system) документацию, относящуюся к ListBox:



Эта документация в «дереве» под пунктом gb.qt, компонент Gambas, который включает все «видимые» средства управления (клавиши, этикетки, меню и т.д.). Прочитаем, что ListBox предоставляет метод Clean, который очищает все вводы. Это все, что мы хотели, и мы с удовольствием воспользуемся этим методом.

Двойной щелчок по клавише **Очистить** (или пункту меню «Очистить»), открывается редактор кода и курсор позиционируется на соответствующей процедуре. Мы запишем следующий код:

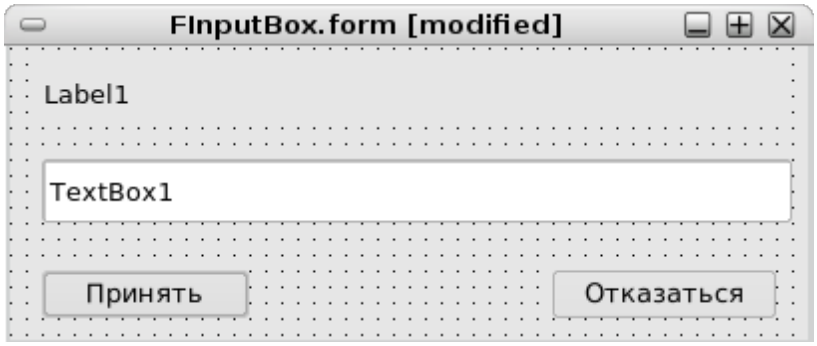
```
PUBLIC SUB Clean_Click()
ListBox1.Clean()
END
```

Очень легко :–)

Действие Add

Это действие более сложное. Пользователи будут добавлять ввод (строку текста) в ListBox, используя эту клавишу.

Gambas не предоставляет диалог типа «InputBox», так что мы собираемся создать нечто в этом роде. Мы создадим новую Form, но теперь мы зададим, что форма имеет конструктор. Почему? Потому, что в создаваемом образце мы хотим задать некоторые свойства формы, такие как заголовок, показ сообщений и определенное значение в поле текстового ввода. Вот предлагаемая конструкция:



Форма очень проста. Она имеет Label, текстовый ввод (TextBox) и две клавиши, **Accept** - принять и **Cancel** - отказаться. Разумный диалог предполагает наличие удобной для пользователя возможности отказаться с помощью клавиши **Escape** и принять с помощью клавиши **Enter**:

> Button управление имеет свойства, названные *Default* и *Cancel*. Мы установим Default в True для клавиши **Принять** и Cancel свойство в True для клавиши **Отказаться**.

Применение этих свойств приведет к тому, что когда пользователь нажмет клавишу <ENTER>, форма поведет себя также, как если бы пользователь щелкнул по клавише "Принять", а нажатие на <ESC> будет эквивалентно нажатию на клавишу **Отказаться**.

Следующая проблема, как вернуть написанный текст в диалоге в основное окно. Мы должны помнить, что в Gambas нет глобальных переменных, так что нам следует поискать другое решение. В «Tip of the day» №7, (пункт меню «? > Tips of the day - подсказки дня») есть предложение использовать переменную, объявленную как PUBLIC, так что эта переменная видима из любой точки или класса внутри

программы,

Создадим новый модуль (правый щелчок по Modules > New module) и мы назовем этот модуль MCommon, например. Вот реализация модуля:

```
' Gambas module file
PUBLIC my_text AS String
```

Очень просто и легко. Теперь мы имеем переменную, которая может быть доступна из любой точки внутри программы при использовании следующей нотации: MCommon.my_text

Теперь мы запишем код для нашего диалога InputBox:

```
' Gambas class file
PUBLIC SUB _new(title AS String, message AS String, OPTIONAL text
AS String)
	ME.Caption = title
	Label1.Caption = message
	' a String is evaluated to FALSE if it is empty:
	IF text THEN TextBox1.Text = text
END
PUBLIC SUB Button1_Click() ' This is the "Accept" button
	MCommon.my_text = TextBox1.Text
	ME.Close(0)
END
PUBLIC SUB Button2_Click() ' This is the "Cancel" button
	ME.Close(0)
END
```

Процедура _new - это конструктор. Используя его, мы можем устанавливать разные заголовки, этикетки и содержание текстовых вводов каждый раз, когда используется диалог. Более того, эти свойства устанавливаются в момент создания.

Клавиша **Принять** копирует текст в TextBox в переменную my_text, определенную в модуле MCommon и закрывает диалог. Клавиша **Отказаться** просто закрывает диалог.

Поскольку переменная MCommon.my_text в общем пользовании, мы должны помнить о необходимости clear - очистить каждый раз при ее использовании. Мы увидим это сейчас.

Процедура для клавиши **Добавить - Add** в основной форме снабжена следующим кодом. Он хорошо прокомментирован:

```
PUBLIC SUB Add_Click()
        ' Declarating our "Inputbox"
        f AS FInputBox
        ' We create the InputBox, setting the title, message
        ' and a default value: the system date and time
        ' and a small arrow
        f = NEW FInputBox("Write an entry", "Please, write here the entry to
be added:",
        CStr(Now) & " -> ")
        ' We show the InputBox
        f.ShowModal()
        ' If the user has written some text, it will
        ' be in the shared variable MCommon.my_text
        IF MCommon.my_text THEN ' An empty string is False
        ' The ListBox control has a method for adding entries: Add()
        ListBox1.Add(MCommon.my_text)
        ' We "empty" the shared variable
        MCommon.my_text = ""
        END IF
END
```

Действие Modify

Применяя эту клавишу пользователь может изменить ввод в ListBox. Если записей нет, клавиша ничего не делает, а, если пользователь не выбрал запись, он будет предупрежден об этом. Вот реализация для этого действия:

```
' "Modify" action
PUBLIC SUB Modify_Click()
```

```
        f AS FInputBox
        IF ListBox1.Count > 0 THEN ' If the ListBox is empty, its property
        ' Count is 0
        IF ListBox1.Index = -1 THEN
        ' The Index property returns the index for the selected entry.
        ' It there is not selected line, it returns -1.
        ' We warn the user.
        message.Info("You must select the line to modify.")
        ELSE
        ' The user has selected a valid entry.
        ' We show our InputBox with the text of the selected entry.
        ' The selected text is the property Text of the
        ' selected object ListBoxItem
        ' which is accesible through the property
        ' Selected of the ListBox
        f = NEW FInputBox("Modify entry", "Please, modify the selected
entry:", ListBox1.Current.Text)
        f.ShowModal()
        ' The dialog box FInputBox changes the shared variable
        ' MCommon.my_text
        ' If MCommon.my_text is not empty, we load it in the
        ' selected ListBoxItem
        IF MCommon.my_text THEN ListBox1.Current.Text = MCommon.my_text
        ' We "empty" the shared variable after its use
        MCommon.my_text = ""
        END IF
END
```

Действие Delete

Как мы видели выше, ListBox должен содержать хотя бы одну строку, а пользователь должен выделить одну строку.

Код очень похож на тот, что для действия Modify - изменить:

```
PUBLIC SUB Delete_Click()
	i AS Integer
	i = ListBox1.Index
	IF i >= 0 THEN
	ListBox1.Remove(i) ' The Remove method erases the selected line
	ELSE IF ListBox1.Count > 0 AND i = -1 THEN
	' We check that the ListBox is not empty and
	' that some entry is selected.
	message.Info("You must select the desired line to delete.")
	END IF
END
```

Можно видеть, что реализация для этих четырех действий общая для клавиш и их эквивалентов в меню.

Теперь мы реализуем действие, относящееся к управлению файлами (Open - открыть, Save - сохранить) и закрытию программы. Мы начнем с того, что попроще.

Действие Exit

Функция для этой клавиши (и ассоциированного пункта меню) - закрыть программу. Очень просто:

```
PUBLIC SUB Exit_Click()
	ME.Close(0) ' ME is a reference to the form itself
	FInputBox
END
```

Мы можем проявить больше дружелюбия к пользователю, добавив диалог наподобие: «Вы собираетесь выйти из программы. Вы уверены?», – и создав подходящее задание. Но, давайте, оставим эту реализацию в качестве домашнего задания для читателей.

Действие Open

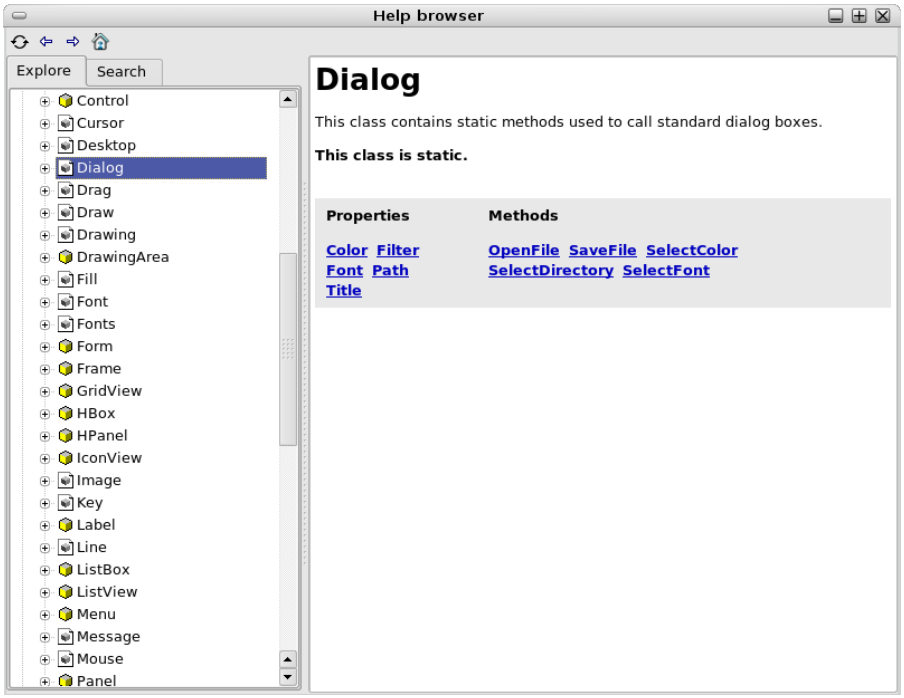
Что это действие должно сделать? Выяснить у пользователя путь к файлу, прочитать этот файл и загрузить данные в ListBox. Вот соответствующий код:

```
PUBLIC SUB Open_Click()
```

```
        DIM lin AS String
        DIM arr_strings AS String[]
        Dialog.Title = "Please select a file"
        Dialog.Filter = [ "Minder data (*.data)", "All files (*.*)" ]
        IF NOT Dialog.OpenFile() THEN
        arr_strings = Split(File.LOAD(Dialog.Path), "\n")
        ListBox1.Clean()
        FOR EACH lin IN arr_strings
        ListBox1.Add(lin)
        NEXT
        END IF
END
```

Эта часть кода представляет интересную особенность языка Gambas, «non instanceable - не запрашиваемые» или статические классы (2). Мы не можем создавать объекты этих классов, но мы можем прямо использовать их. В этом коде мы можем видеть два из этих классов в действии: класс File и класс Dialog.

Например, класс Dialog предоставляет доступ к типичному стандартному диалогу для выбора файлов, системных цветов и т.д. Это документировано в теме help, gb.qt



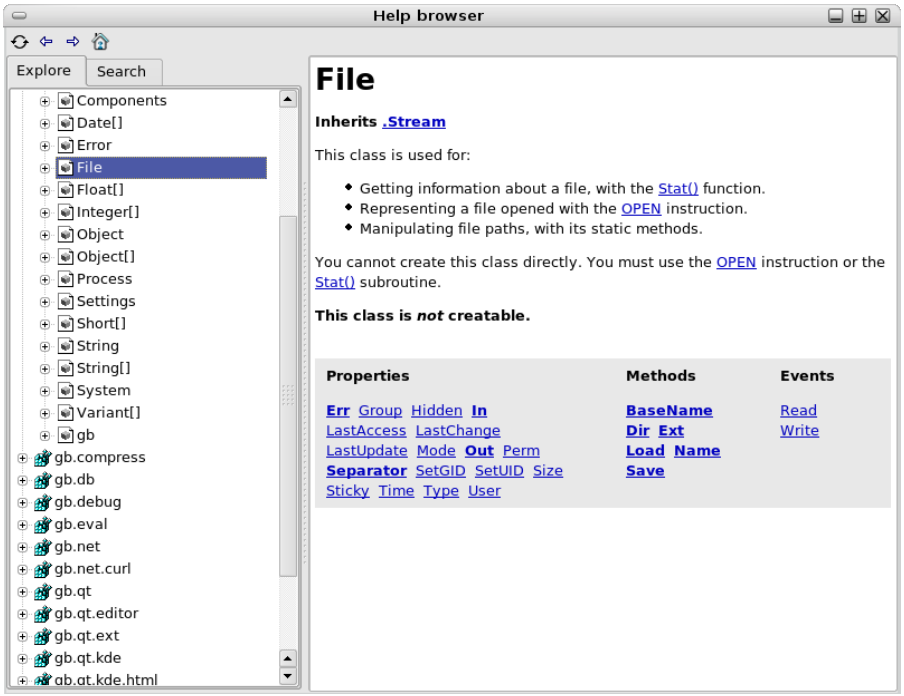
В нашей программе мы хотим выбрать файл и загрузить его содержимое в ListBox. Мы будем использовать класс Dialog следующим образом:

```
Dialog.Title = "Please select a file"
Dialog.Filter = [ "Minder data (*.data)", "All files (*.*)" ]
IF NOT Dialog.OpenFile() THEN
' etc ...
```

Мы задали заголовок диалога, предоставили фильтр для выбора типа файла по расширению (*.data) и, наконец, мы вызвали метод класса OpenFile(). Есть особенность в класее Dialog: если пользователь НЕ выбрал файл (то есть, пользователь нажал клавишу отказа (cancel) или клавишу ESC) возвращаемое значение в методе OpenFile() становится True. Когда же пользователь выбирает файл, мы можем получить доступ к полному пути через свойство Dialog.Path

Класс File (его документация под пунктом gb в системе помощи) предоставляет

несколько методов, позволяющих нам работать с файлами:



В документации к Gambas в разделе, озаглавленном «How do I ...» есть несколько примеров кода для чтения и записи файлов. Мы будем использовать метод Load(), чьими аргументами являются путь к файлу и возвращаемая String,которая содержит все данные в этом файле. Мы можем разделить строки возвращаемых данных, используя функцию Split(). Ее аргументы - это строки, которые следует разделить и разделитель (символ новой строки в данном случае \n) и возвращаемый Array of Strings. Из этих соображений мы должны объявить переменную arr_strings as String[]:

```
DIM arr_strings AS String[]
```

Когда мы получим все строки данных, содержащиеся в файле, мы должны очистить ListBox и добавить каждую строку, используя метод Add() ListBox.

Действие Save

Когда пользователь нажимает клавишу **Save - сохранить**, программа должна вывести содержимое ListBox в текстовый файл. Мы покажем диалог Save File, запрашивая у пользователя имя файла для сохранения данных. Вот код для этого действия:

```
PUBLIC SUB Save_Click()
        lines AS String
        destination AS String
        numFile AS Integer
        lines = ListBox1.Contents
        Dialog.Title = "Please, select a file"
        Dialog.Filter = [ "Minder data (*.data)" ]
        IF NOT Dialog.SaveFile() THEN
        IF Right$(Dialog.Path, 5) <> ".data" THEN
        destination = Dialog.Path & ".data"
        ELSE
        destination = Dialog.Path
        END IF
        File.Save(destination, lines)
        END IF
END
```

Мы хотим сохранить данные в файле с расширением .data, так что, если имя файла предоставленное пользователем не заканчивается с ".data", мы будем дописывать расширение. Мы используем метод Save(), который предоставляется классом File.

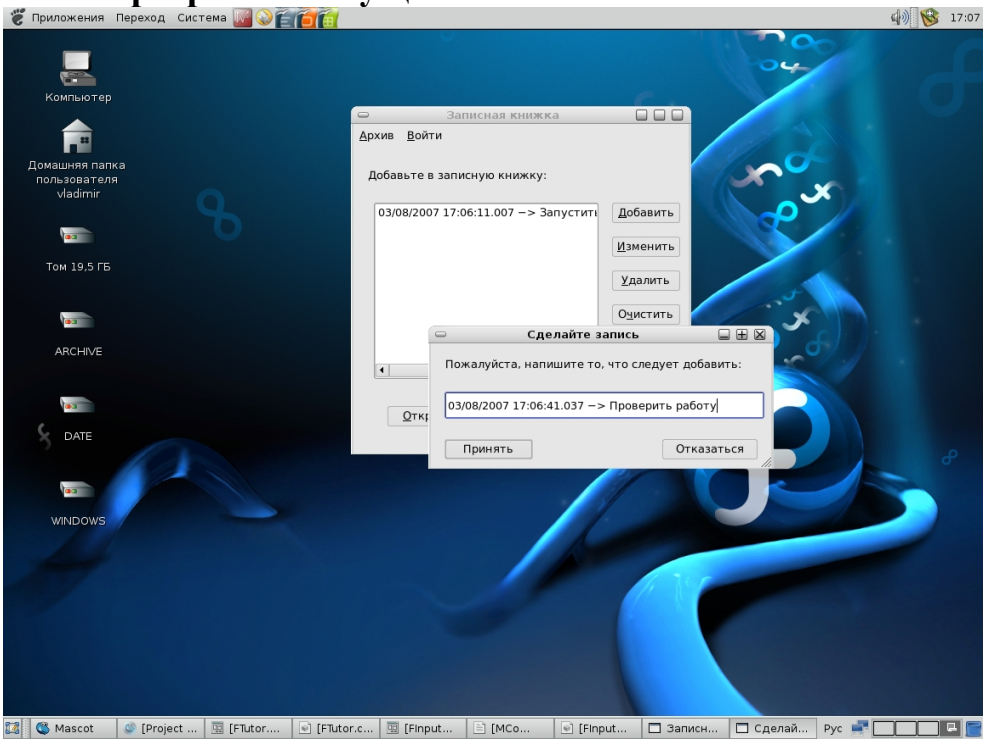
Аргументы этого метода - путь к файлу и текст, который мы хотим сохранить. Мы получаем доступ к содержимому ListBox, используя свойство Contents, которое возвращает String, с «new line» (\n) разделителем каждого ввода в ListBox.

Окончательная подгонка

Пользователю интересно увидеть весь текст записи, а listbox может оказаться не достаточно широким, чтобы отобразить его, если запись очень длинная. Мы реализуем эту возможность следующим образом: когда пользователь дважды щелкает клавишей мышки по вводу, мы покажем весь текст в окне диалога:

```
PUBLIC SUB ListBox1_DblClick()
        IF ListBox1.Index >= 0 THEN
        message.Info(ListBox1.Current.Text)
        END IF
END
```

Наша программа запущена



Развертывание завершения программы

OK. Наша программа готова. Мы можем проверить ее в любое время внутри IDE, нажав на клавишу F5.

Теперь мы хотим использовать программу, как обычное приложение, то есть, без IDE. Для выполнения этого есть опция основного меню: Project > Create executable. Этим создается «монолитный» исполняемый файл, включающий все ресурсы проекта. Этот файл не будет машинным кодом, но «байт-кодом», выполняемым интерпретатором Gambas – gbx. Это предполагает, что мы нуждаемся, чтобы этот интерпретатор был

установлен в системе для запуска программ, написанных в Gambas. (Это похоже на другие языки программирования. Например, нам нужно установить Java для выполнения программ, написанных на Java).

Все дистрибутивы, в которые включен Gambas, компоненты Gambas разделены и есть пакет «Gambas runtime», который включает интерпретатор, но не полный IDE.

Мы можем создать RPM или DEB архив нашей программы. Эти пакеты будут иметь интерпретатор Gambas (gambas−runtime программа) по зависимости. Есть мастер для помощи в генерации такого пакета. Он очень прост и интуитивно понятен. Вы можете найти его в меню Project > Create installation package ...

Окончание

Мы убедились, что это очень легко создавать простые, но функциональные приложения в Gambas. Этот инструмент предлагает некоторые средства управления и предопределенные классы. Есть также множество расширений или компонентов, доступных в порядке создания клиент/серверных приложений, доступа к базам данных, приложений мультимедиа и т.д.

IMHO, Я думаю, что Gambas очень многообещающий инструмент. К счастью, Gambas' mailing−list разработчиков очень активен, а ошибки устраняются очень быстро.

Спасибо, Benoit (et col.)! Хорошая работа!

Об этом документе и его авторе

Как говорилось выше, мы работали с версией 1.0−1 Gambas в этом руководстве (предварительно скомпилированный пакет для Debian Sid). Во время написания этого документа была выпущена версия 1.0.3, и весьма вероятно, что в момент прочтения этого, появится новая версия. Весьма благоразумно почитать журнал изменений, чтобы избежать возможных расхождений от версии к версии.

Все комментарии, предложения, коррективы и улучшения этого документа приветствуются.

Мой mail – forodejazz@gmail.com

Примечания:

- События нуждаются в поддержке процедуры или подпрограммы: функция, которая не возвращает какого-либо значения.

- Я не эксперт в технологии объектно-ориентированного программирования. Мои извинения, если я использовал некорректные термины ;−)

Список литературы

1. Мозговой М. Занимательное программирование. Из-во Питер, 2005
2. С.Н. Лукин. Понятно о Visual Basic.NET. Из-во Диалог-МИФИ, 2005
3. Тэтчелл Дж., Беннет Б., Фрейзер К., Смит Б.Р. Осваиваем микро-компьютер. Из-во Мир, 1989