Видеоигра на PIC18

Введение

Примечание: Данный проект является компиляцией двух предыдущих проектов:

- VGA-терминал на PIC18 [http://wiki.pic24.ru/doku.php/osa/articles/vga_terminal]
- "Квартет" [http://wiki.pic24.ru/doku.php/osa/ref/appendix/quartet]

(Оригинал статьи здесь [http://wiki.pic24.ru/doku.php/osa/articles/vga_game].)

Можно поставить под сомнение эффективность использования микроконтроллеров PIC для формирования сигналов управления монитором VGA, но нельзя отрицать того, что такое их применение возможно. Данный пример демонстрирует применение микроконтроллера PIC18F2550 в качестве самостоятельной игровой приставки. В 2004-м году была написана программа-терминал [http://wiki.pic24.ru/doku.php/osa/articles/vga_terminal], позволяющая использовать PIC18 для формирования цветного текстового изображения на мониторе VGA. Я немного расширил возможности этой программы, добавив анимацию, полифонический звук и кнопочное управление. В далеких 80-х, когда у меня была игровая приставка «Синклер» я был без ума от игрушки Boulder Dash фирмы First Star Software [http://www.firststarsoftware.com]. Сейчас я сделал попытку перенести ее на ПИК. Что из этого вышло, - выкладываю здесь с исходниками.

Исходные тексты программы: vga_game.rar

Программа написана на Си (HT-PICC18), за исключением обработчика прерывания, который целиком написан на ассемблере. Выполнением программы управляет ОСРВ OSA.

Характеристики программы:

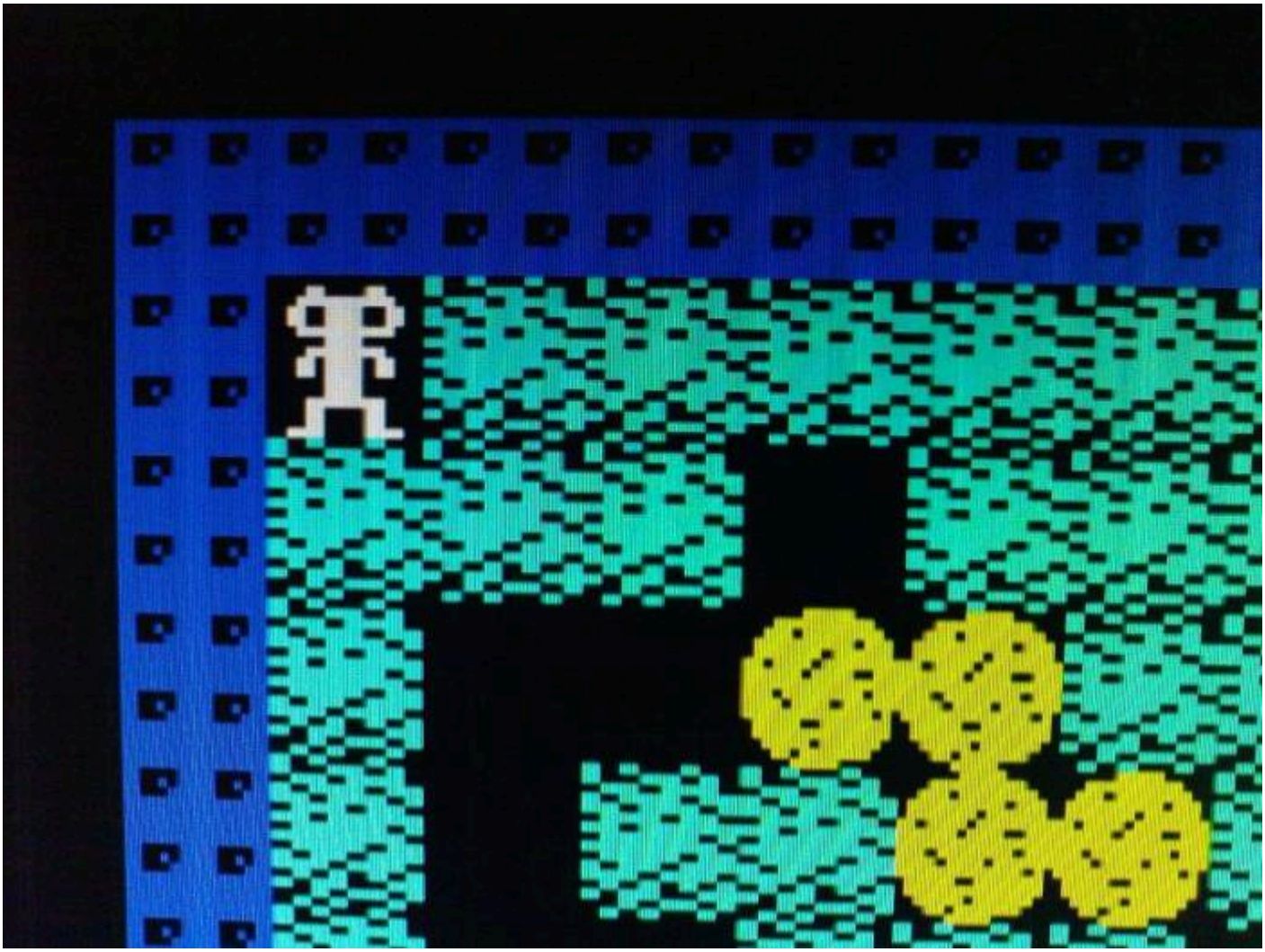
- Контроллер: **PIC18F2550**
- Таковая частота: **48 MHz (12 MIPS)**
- RTOS: **OSA**
- VGA: **256×200 пикселей, 15 цветов**
- Полифония: **5 голосов (4 - музыка, 1 - игровые эффекты)**
- Частота семплирования: **15 КГц**
- Игровое поле: **40×20 клеток**
- Видимая область игрового поля: **16×12 клеток**

Видео HQ (34 Mb)

Вот небольшое видео. Качество не очень хорошее, т.к. снималось на дешевую web-камеру (звук

www.youtube.com/v/PZjuyCuH7Ac

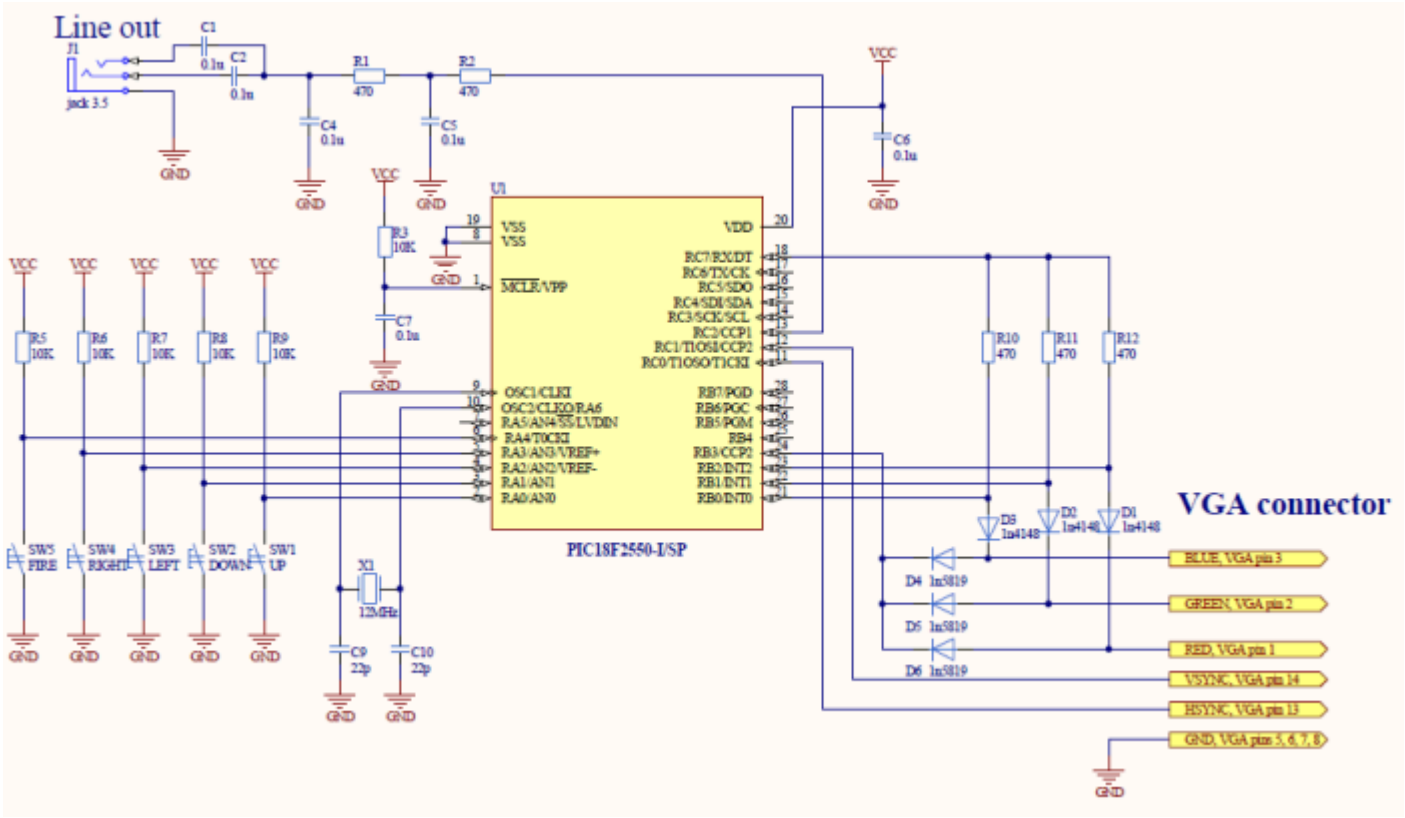
К сожалению, в лучшем качестве снять не удалось. На самом деле изображение гораздо четче:



Описание

Схема

Схема устройства довольно простая:

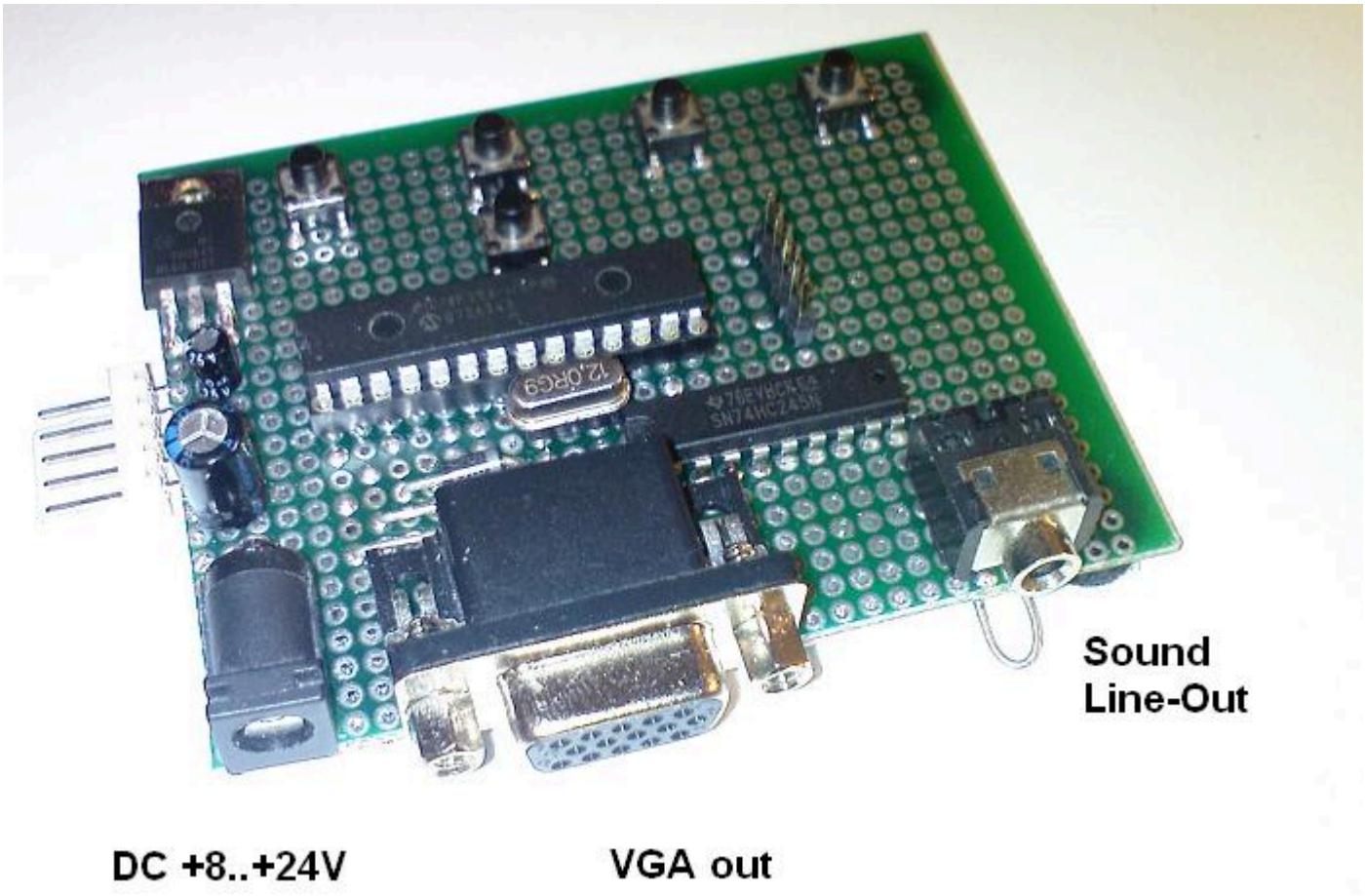


Логически схему можно разбить на 4 части:

1. Сердце устройства - микроконтроллер;
2. 3 резистора и 6 диодов для формирования цветного изображения;
3. фильтр низких частот второго порядка для вывода звука;
4. пять кнопок с резистивной подтяжкой к +5В.

Принцип формирования изображения описан здесь [http://wiki.pic24.ru/doku.php/osa/articles/vga_terminal#генерация_изображения].

А вот фото самого устройства:



Распределение ресурсов

Как видно из характеристик программы, контроллеру приходится успевать делать довольно много дел одновременно и четко синхронно: генерация синхроимпульсов, генерация RGB-сигналов, генерация звука - все эти задачи контроллер должен делать синхронно с точностью до такта, иначе картинка будет дрожать, цвета - прыгать влево-вправо, звук - дребезжать. Да и игровые функции хотелось бы обрабатывать в реальном времени: обработка кнопок, формирование данных для анимации, игровой процесс, воспроизведение музыки и игровых эффектов.

Было решено сделать так: часть задач, особо критичных ко времени, затолкать в прерывание, а все остальные задачи, допускающие отклонение в несколько сотен тактов контроллера, оформить в виде функций.

На обработчик прерывания возлагаются следующие задачи:

- формирование VGA-синхроимпульсов;
- формирование RGB-сигналов;
- синтезатор звуков;
- чтение кнопок и устранение дребезга.

Вне прерывания будет выполняться все остальное:

- формирование данных анимации;
- игровой алгоритм;
- формирование данных для синтезатора (музыка и игровые эффекты).

Учитывая, что обработчик прерывания функционально сильно перегружен, весь его код полностью написан на ассемблере. Это позволяет не только с точностью до такта синхронизировать задачу формирования VGA-сигналов, но и максимально оптимизировать код вручную так, как компилятору будет сделать не по силам. Например, при прорисовке игрового поля и текстовой строки программа загружена по-разному и имеет свободное время для обработки звука в разные моменты времени. Поэтому код обработки звука разбит на несколько маленьких кусочков, которые

втиснуты в «щели», образующиеся при формировании изображения. Например, на прорисовку каждой клеточки игрового поля уходит ровно 14 команд, а время прорисовки - 16 тактов, следовательно, есть 2 свободных такта. Прорисовка сделана так, что нечетные клеточки имеют эти два свободных такта в конце прорисовки, а четные - в начале, и т.к. они идут последовательно, то на стыках «нечетная-четная» (а таких стыков 8) появляются «щели» по 4 такта (в общей сложности 32 такта), куда и втискиваются части кода для формирования звука. При прорисовке текстовой строки эти «щели» образуются в других местах. Использование ассемблера позволило утрамбовать код так, чтобы программа успевала сделать все за выделенные ей 380 тактов (380 * 83.3 нс = 31.7 мкс - период горизонтального синхроимпульса)

Кроме того, нам нужно помнить, что для выполнения всех остальных задач остается совсем немного времени. Фактически во время прорисовки 400 строк видеоизображения у контроллера нет ни такта свободного времени для выполнения второстепенных задач, т.к. все остальное время (время синхроимпульса и левого и правого бордюра) тратится на управление синтезатором и подготовку данных для вывода на экран, а также на сохранение/восстановление контекста прерывания. Таким образом, из всего времени, за которое прорисовываются 525 строк одного кадра, есть всего 125, во время обработки которых контроллер более или менее разгружен (в это время он занят только синтезатором). Следовательно, нам нужен механизм, который позволил бы наиболее эффективно распределять оставшееся время между остальными задачами.

И здесь нам на помощь приходит RTOS. Дело в том, что, например, просчет одного игрового шага (траектории движения всяких «бабочек» и «огневушек», падение камней и алмазов и пр.) может длиться сравнительно долго. Из-за этого музыка, которая играется по фону, генерировалась бы со срывами (т.к. во время просчета игрового алгоритма программа иногда будет полностью уходить в прерывания на прорисовку 400 линий растра, что будет затягивать время просчета на долгие миллисекунды). RTOS же позволила прерывать длительные расчеты для того, чтобы сформировать указания синтезатору для воспроизведения очередных нот. Для этого нужно было сделать всего две вещи:

1. задаче, проигрывающей мелодию, установить более высокий приоритет;
2. из задачи расчета игрового шага периодически передавать управление ядру ОС.

Формирование изображения

Формирование изображения практически такое же, как и в проекте "Терминал" [http://wiki.pic24.ru/doku.php/osa/articles/vga_terminal]. Разница состоит в том, что на один пиксель выделяется две строки растра, а размер спрайта не 8×16 пикселей, а 16×16. Т.е. вся информация выводится в виде двухцветных матриц (один из цветов всегда черный). Эффект анимации создается за счет быстрой смены матриц. Пои прорисовке очередной строки из массива спрайтов, расположенного в ROM, программа выбирает байты, соответствующие текущему номеру строки. На каждый спрайт по два байта в одной строке.

Синтезатор

Синтезатор построен по тому же принципу, что и в программе "Квартет" [http://wiki.pic24.ru/doku.php/osa/ref/appendix/quartet]. Т.е. на каждый канал заведена переменная типа структуры, содержащей информацию о частоте, текущей фазе, громкости и пр. для данного канала. Исходя из данных содержащихся в этой структуре, выбирается байт из массива, в котором хранится оцифрованный период синусоиды (на самом деле это не совсем синусоида, а функция, вычисленная как (6*sin(x)+3*sin(2x)+sin(3x))/10). Сумма мгновенных значений амплитуд для всех каналов выводится через ШИМ. Одновременно пересчитывается громкость, затухание, для игрового канала - шум, сдвиг частоты, затухание шума. (Единственное отличие от синтезатора из «Квартета» - это добавление для канала игровых звуков эффекта шума.)

Синтезатор формирует мгновенные значения амплитуд для всех каналов в два прохода (т.е. за время прорисовки двух строк растра): на четных строках просчитываются амплитуды для каналов 0,1 и 2, а на нечетных - 3 и 4 (игровой). Поэтому и частота семплирования в два раза ниже частоты горизонтальных синхроимпульсов и равна 15.74 КГц.

Музыка

Музыка также организована по принципу, схожему с программой "Квартет" [http://wiki.pic24.ru/doku.php/osa/ref/appendix/quartet]. На каждый звуковой канал заведена своя «нотная тетрадь», откуда по очереди берутся и проигрываются ноты, выдерживаются паузы, выполняются повторы фрагментов. Но здесь, в отличие от «Квартета», на каждый звуковой канал не выделена своя задача, а все они обрабатываются в одной задаче - **Task_Music**.

Анимация

Т.к. игра содержит анимированные объекты: сам человечек, бабочки, огневушки, алмазы, домик, взрывы, - то требуется с определенным периодом обновлять данные, чтобы код прорисовки, расположенный в обработчике прерывания, знал, какой спрайт для данного объекта нужно выбрать. Эта задача проходится в цикле по всем восьмистам клеткам игрового поля и обновляет у всех анимированных объектов два младших бита, отвечающих за фазу анимации (т.е. на каждый анимированный объект предусмотрено 4 фазы движения). Т.к. данная задача может выполняться длительное время, то после обработки каждых сорока клеток задача возвращает управление планировщику.

Задача обработки анимации **Task_Animate** запускается после прорисовки каждого 8-го кадра (т.е. 7.5 раз в секунду).

Игра

Это была самая простая часть программы. Задача обработки игрового алгоритма **Task_Game** активируется после прорисовки каждого 16-го кадра (4 раза в секунду) и выполняет один шаг обновления игрового поля. Пробегаясь по всем его клеткам, задача ищет объекты, которые должны двигаться, вычисляет для них, в зависимости от траектории, следующее местоположение, а также следит за правилами игры: например, столкновение движущегося объекта (человечка или бабочки) с летящим камнем должно приводить к взрыву; взрыв уничтожает все в радиусе 2 клеток, кроме титановой стены; бабочка после смерти превращается в несколько алмазов и т.д.

Так как задача выполняется довольно длительное время, она иногда (после проверки каждых сорока клеток) передает управление планировщику, чтобы задача формирования музыки или обработки звуковых эффектов смогли получить управление. После проверки и обработки всех восьмисот клеток игрового поля **Task_Game** отправляет короткое сообщение задаче **Task_Sound** с кодом звука, который должен сопроводить данный шаг игры, например, звук взрыва, или падение камня или поедание алмаза. Если данный шаг не должен сопровождаться никакими звуками, то сообщение не отсылается.

Возможно, при написании алгоритма игры где-то есть несоответствие с оригинальными правилами (например, в порядке падения камней друг на друга, когда они валят с разных сторон), но внешне все получилось очень похоже. В моей версии игры нет еще четырех эффектов, присущих обычному Boulder Dash (я использую терминологию, которой пользовался в детстве): растущая биомасса (для убийства бабочек), не растущая биомасса (пропускающая камни и алмазы с задержкой), растущая кирпичная стена (заполняет пустые пространства влево и вправо), золотая стена (превращает камни в алмазы и наоборот). Если честно, то я просто поленился.

Управление:

- кнопками «влево», «вправо», «вверх» и «вниз» двигаем человечка;
- комбинация любой из этих кнопок с кнопкой «Fire» делает шаг в соответствующую сторону без перемещения человечка (например, съесть алмаз, находящийся справа, оставшись при этом стоять на месте);
- если так получилось, что человечек оказался завален камнями и ему не сделать шаг ни в одну сторону, то можно нажать все 4 кнопки направления сразу - это приведет к перезагрузке уровня;
- нажатием всех пяти кнопок сразу мы переходим на другой уровень.

Создание новых уровней

За вами остается возможность создавать свои игровые уровни. В файле game_data.c инициализируется массив **const char MAP[][][]**, в котором хранятся карты игровых полей.

Количество карт определяется константой **NUMBER_OF_MAPS** (в моей программе = 3, но можно увеличить и создать больше карт). Каждая карта представляет собой массив char'ов размерностью (MAP_SIZE_Y+1)x(MAP_SIZE_X+1). Первая строка содержит определения цветов для разных типов объекта для конкретной карты (см. константы CL_xxx), а также требуемое количество алмазов для прохождения уровня. Далее следуют MAP_SIZE_Y строк, в которых задается сама карта с описанием объектов:

(пробел) - пустота;
титановая не ломаемая стена;
= - кирпичная стена;
. - земля;
m - человечек;
h - домик;
x - бабочка;
f - огневушка;
o - камень;
+ - алмаз.

Бордюр карты должен состоять из титановой стены. По самому исходному файлу довольно просто разобраться, что там к чему и как что менять, чтобы создавать свои карты. При небольшом усилии и небольшой модификации программы можно сделать подгружаемые карты из внешней EEPROM.

Сборка проекта

Проект создан в интегрированной среде MPLAB IDE [http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002]. Для сборки использовался компилятор HT-PICC18 STD 9.51pl2.

Скачиваем **файлы операционной системы OSA [http://wiki.pic24.ru/doku.php/osa/ref/download/intro]**, распаковываем архив на диск C: (должна получиться папка C:\OSA).

Распаковываем файл **vga_game.rar** в папку C:\TEST\VGA. При этом внутри создастся папка VGA_GAME. В MPLAB IDE открываем проект.

Примечание. *При распаковке в другую папку, отличную от C:\TEST\VGA\VGA_GAME, нужно будет через меню Project\Build options…\Project в закладке Directories в списке include-путей заменить путь к файлам проекта на тот, куда Вы распаковали файлы из архива.*

Выполняем сборку нажатием **Ctrl+F10**.

Заключение

Создавая данный проект, я преследовал несколько целей:

- Во-первых, просто упражнялся в оптимизации кода;
- Во-вторых, всегда интересно, что можно выжать из контроллера;
- В-третьих, этот проект является очередным примером использования RTOS, причем в данном случае ее применение более чем оправдано, т.к. из-за огромных временных затрат на прорисовку изображения создается дефицит временных ресурсов контроллера, который ОС и позволяет использовать с максимальной эффективностью, исключая протормаживания и зависания.

Само устройство можно совершенствовать дальше. Например, применив внешнюю SRAM-память, можно расширить графические возможности (и количество цветов и избавиться от ограничения «1 цвет на спрайт»). Повесив внешний музыкальный контроллер (пускай просто второй ПИК), можно значительно расширить музыкальные возможности устройства (например, повысить частоту семплирования, увеличить разрядность ШИМ, увеличить количество звуковых каналов, добавить имитацию различных музыкальных инструментов и барабанов, сделать стерео и т.д.). Можно добавить внешнюю EEPROM для подгрузки карт, спрайтов, музыки и сохранения игр. Разгрузив контроллер, оставив на нем только правила игры и VGA-вывод (что, собственно, тоже можно

разнести по разным контроллерам), мы можем усложнять алгоритм игры, добавлять интересные видеоэффекты и т.д. и т.п.

Если у кого-то возникнет желание сделать для себя (или не для себя) интересную поделку, - не стесняйтесь пользоваться моими наработками и исходниками хоть в любительских целях, хоть в коммерческих.

Удачи!

Виктор Тимофеев, май, 2009

osa@pic24.ru [mailto:osa@pic24.ru]

Ссылки

- Видеоигра на PIC18 [http://wiki.pic24.ru/doku.php/osa/articles/vga_game] - оригинал статьи на www.pic24.ru [http://www.pic24.ru]

Проекты, заслуживающие внимания:

- http://avga.prometheus4.com/ [http://avga.prometheus4.com/] - несколько цветных видеоигр с выводом на VGA (ATmega).
- http://www.linusakesson.net/scene/craft/ [http://www.linusakesson.net/scene/craft/] - очень интересная демка с анимацией и полифонией (ATmega).
- http://www.rickard.gunee.com/projects/ [http://www.rickard.gunee.com/projects/] - цветные игры «Тетрис» и «Pong» с выводом на телевизор (Scenix)

И еще ссылки:

- http://z80speccy.narod.ru/ [http://z80speccy.narod.ru/] - отличный эмулятор ZX Spectrum
- http://andygame.narod.ru/zx/game/index.html [http://andygame.narod.ru/zx/game/index.html] - игрушки для него (в том числе и Boulder Dash)