Trò chơi điện tử trên PIC18

Giới thiệu

Người ta có thể đặt câu hỏi về tính hiệu quả của việc sử dụng vi điều khiển PIC để tạo ra tín hiệu điều khiển màn hình VGA, nhưng không thể phủ nhận rằng việc sử dụng chúng là hoàn toàn khả thi. Ví dụ này minh họa việc sử dụng vi điều khiển PIC18F2550 như một bảng điều khiển trò chơi độc lập. Năm 2004, một chương trình đầu cuối đã được viết cho phép bạn sử dụng PIC18 để tạo hình ảnh văn bản màu trên màn hình VGA. Tôi đã mở rộng một chút khả năng của chương trình này bằng cách thêm hoạt ảnh, âm thanh đa âm và điều khiển bằng nút nhấn. Quay trở lại những năm 80, khi tôi có máy chơi game Sinclair, tôi đã phát cuồng với món đồ chơi Boulder Dash của First Star Software [http://www.firststarsoftware.com] . Bây giờ tôi đã cố gắng chuyển nó sang PIK. Điều gì đến của nó, tôi đăng nó ở đây với mã nguồn.

Văn bản nguồn của chương trình: vga_game.rar

Chương trình được viết bằng C (HT-PICC18), ngoại trừ trình xử lý ngắt, được viết hoàn toàn bằng trình hợp dịch. Việc thực thi chương trình được điều khiển bởi OSA RTOS.

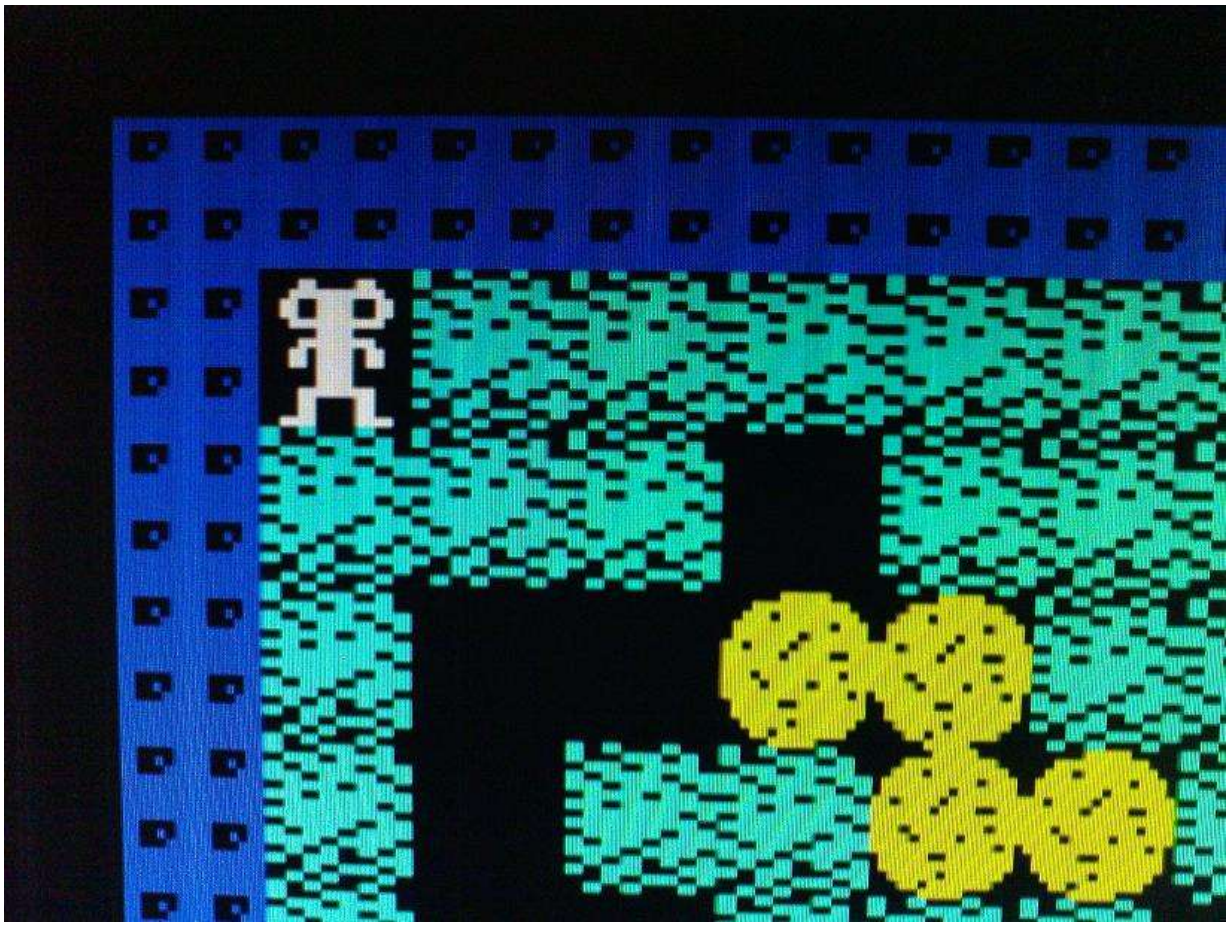
Tính năng chương trình:

Bộ điều khiển	PIC18F2550
Tần suất như vậy	48 MHz (12 MIPS)
RTOS	OSA
VGA	256x200 pixel, 15 màu
Phức điệu	5 giọng nói (4 - âm nhạc, 1 - hiệu ứng trò chơi). Tần số lấy mẫu 15 kHz
sân chơi	40x20 ô
Khu vực có thể nhìn thấy của sân chơi	16x12 ô

Video HQ (34 Mb)

Đây là một đoạn video ngắn. Chất lượng không tốt lắm, bởi vì được quay trên máy ảnh web rẻ tiền (âm thanh được ghi riêng qua đường truyền).

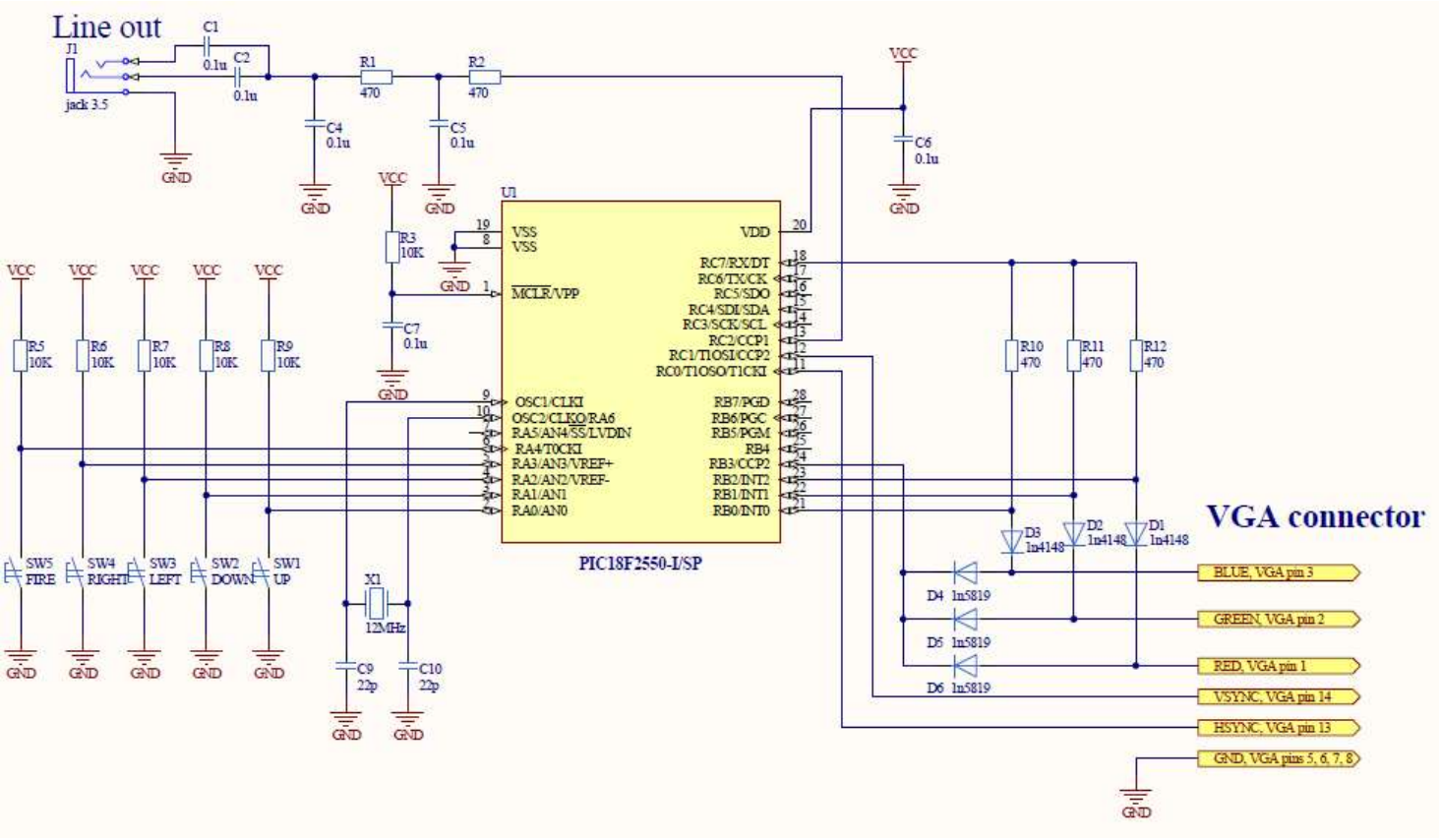
Thật không may, nó không thể chụp ở chất lượng tốt nhất. Trên thực tế, hình ảnh rõ ràng hơn nhiều:



Sự mô tả

Cơ chế

Sơ đồ thiết bị khá đơn giản:

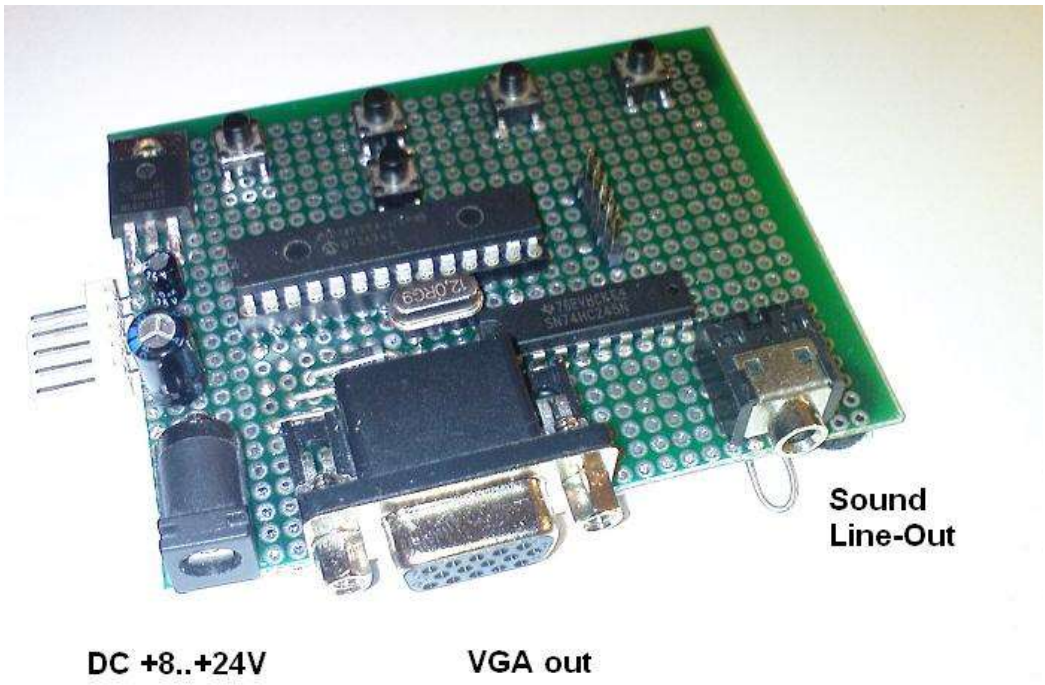


Về mặt logic, sơ đồ có thể được chia thành 4 phần:

1. Trái tim của thiết bị là bộ vi điều khiển;
2. 3 điện trở và 6 điốt để tạo ảnh màu;
3. bộ lọc thông thấp bậc hai cho đầu ra âm thanh;
4. năm nút có điện trở kéo lên đến + 5V.

Nguyên tắc của hình ảnh được mô tả ở đây .

Và đây là ảnh của chính thiết bị:



Phân bổ nguồn lực

Có thể thấy đặc điểm của chương trình là bộ điều khiển phải xoay sở làm khá nhiều việc cùng lúc và đồng bộ rõ ràng: tạo xung đồng bộ, tạo tín hiệu RGB, tạo âm thanh - bộ điều khiển phải thực hiện đồng bộ tất cả các công việc này. với độ chính xác của một nhịp, nếu không hình ảnh sẽ bị rung, màu sắc sẽ nhảy sang trái sang phải, âm thanh - lục lạc. Có, và tôi muốn xử lý các chức năng trò chơi trong thời gian thực: xử lý nút, tạo dữ liệu cho hoạt ảnh, trò chơi, phát lại nhạc và hiệu ứng trò chơi.

Nó đã được quyết định để làm điều này: đẩy một số tác vụ đặc biệt quan trọng về thời gian vào trạng thái gián đoạn và sắp xếp tất cả các tác vụ khác cho phép sai lệch vài trăm chu kỳ bộ điều khiển như các chức năng.

Trình xử lý ngắt có các nhiệm vụ sau:

- tạo xung đồng bộ VGA;
- hình thành các tín hiệu RGB;
- bộ tổng hợp âm thanh;
- đọc các nút và loại bỏ nói nhảm.

Ngoài thời gian gián đoạn, mọi thứ khác sẽ được thực hiện:

- tạo dữ liệu hoạt hình;
- thuật toán trò chơi;
- tạo dữ liệu cho bộ tổng hợp (âm nhạc và hiệu ứng trò chơi).

Xét rằng trình xử lý ngắt bị quá tải nặng về mặt chức năng, tất cả mã của nó hoàn toàn được viết bằng trình hợp dịch. Điều này không chỉ cho phép đồng bộ hóa nhiệm vụ tạo tín hiệu VGA với độ chính xác của một chu kỳ đồng hồ, mà còn để tối ưu hóa mã theo cách thủ công càng nhiều càng tốt theo cách mà trình biên dịch sẽ không thể làm được. Ví dụ: khi vẽ một sân chơi và một dòng văn bản, chương trình được tải khác nhau và có thời gian rảnh để xử lý âm thanh vào những thời điểm khác nhau. Do đó, mã xử lý âm thanh được chia thành nhiều mảnh nhỏ được ép vào các "khe" được hình thành trong quá trình hình ảnh. Ví dụ, cần chính xác 14 đội để vẽ mỗi ô của sân chơi và thời gian vẽ là 16 chu kỳ, do đó, có 2 chu kỳ miễn phí. Bản vẽ được thực hiện để các ô lẻ có hai chu kỳ tự do này ở cuối bản vẽ, và thậm chí cả những cái - lúc đầu, và kể từ khi chúng đi tuần tự, sau đó tại các điểm giao nhau "chẵn-lẻ" (và có 8 điểm giao nhau như vậy) có "khe" mỗi vòng 4 chu kỳ (tổng cộng 32 chu kỳ), nơi các phần của mã bị ép lại để tạo thành âm thanh. Khi vẽ một dòng văn bản, những "khe hở" này được hình thành ở những nơi khác. Việc sử dụng trình hợp dịch cho phép chúng tôi thu gọn mã để chương trình có thời gian thực hiện mọi thứ trong 380 chu kỳ được phân bổ cho nó (380 * 83,3 ns = 31,7 µs - chu kỳ của xung đồng bộ ngang)

Ngoài ra, chúng ta cần nhớ rằng chỉ còn rất ít thời gian để hoàn thành tất cả các nhiệm vụ khác. Trên thực tế, trong khi vẽ 400 dòng video, bộ điều khiển không có thời gian rảnh để thực hiện các tác vụ phụ, bởi vì tất cả thời gian còn lại (đồng hồ và thời gian biên trái và phải) được dành cho việc điều khiển bộ tổng hợp và chuẩn bị dữ liệu để hiển thị trên màn hình, cũng như lưu / khôi phục ngữ cảnh ngắt. Do đó, trong tổng số thời gian mà 525 dòng của một khung được vẽ, chỉ có 125 dòng, trong quá trình xử lý, bộ điều khiển ít nhiều không được tải (tại thời điểm này nó chỉ được chiếm bởi bộ tổng hợp). Do đó, chúng tôi cần một cơ chế cho phép chúng tôi phân phối hiệu quả nhất thời gian còn lại giữa các nhiệm vụ khác.

Và đây RTOS đến để giải cứu. Thực tế là, ví dụ, tính toán sai một bước của trò chơi (quỹ đạo chuyển động của tất cả các loại "bướm" và "pháo hoa", sự rơi của đá và kim cương, v.v.) có thể mất một thời gian tương đối dài. Do đó, âm nhạc được phát trong nền sẽ bị gián đoạn (bởi vì trong quá trình hiển thị thuật toán trò chơi, chương trình đôi khi sẽ hoàn toàn bị gián đoạn để vẽ 400 đường raster, điều này sẽ làm chậm thời gian hiển thị dài mili giây). Mặt khác, RTOS có thể làm gián đoạn các phép tính dài hạn để tạo hướng dẫn cho bộ tổng hợp chơi các nốt tiếp theo. Để làm điều này, bạn chỉ phải làm hai điều:

1. đặt mức độ ưu tiên cao hơn cho nhiệm vụ chơi giai điệu;
2. Từ nhiệm vụ tính toán bước trò chơi, định kỳ chuyển quyền điều khiển sang nhân hệ điều hành.

Hình ảnh

Hình ảnh thực tế giống như trong dự án "Terminal" . Sự khác biệt là hai đường raster được phân bổ trên mỗi pixel và kích thước sprite không phải là 8x16 pixel mà là 16x16. Những thứ kia. tất cả thông tin được hiển thị dưới dạng ma trận hai màu (một trong các màu luôn là màu đen). Hiệu ứng hoạt ảnh được tạo ra bằng cách thay đổi nhanh các ma trận. Khi vẽ dòng tiếp theo từ mảng sprite nằm trong ROM, chương trình sẽ chọn các byte tương ứng với số dòng hiện tại. Đối với mỗi sprite, hai byte trên mỗi dòng.

Bộ tổng hợp

Bộ tổng hợp được xây dựng trên nguyên tắc tương tự như trong chương trình Quartet . Những thứ kia. Mỗi kênh có một biến kiểu cấu trúc chứa thông tin về tần số, pha hiện tại, âm lượng, v.v. cho kênh này. Dựa trên dữ liệu có trong cấu trúc này, một byte được chọn từ mảng lưu trữ chu kỳ số hóa của hình sin (trên thực tế, đây không phải là hình sin hoàn toàn, mà là một hàm được tính bằng (6 * sin (x) + 3 * sin (2x) + sin (3x)) / 10). Tổng các giá trị biên độ tức thời cho tất cả các kênh được xuất qua PWM. Đồng thời, âm lượng, độ suy giảm được tính toán lại cho kênh trò chơi - nhiễu, dịch tần số, suy giảm nhiễu. (Sự khác biệt duy nhất so với bộ tổng hợp Quartet là việc bổ sung hiệu ứng tiếng ồn cho kênh âm thanh của trò chơi.)

Bộ tổng hợp tạo ra các giá trị biên độ tức thời cho tất cả các kênh trong hai lần chuyển (tức là trong khi vẽ hai đường raster): trên các đường chẵn, biên độ cho các kênh 0,1 và 2 được tính toán và trên các đường lẻ - 3 và 4 (trò chơi) . Do đó, tần số lấy mẫu thấp hơn hai lần so với tần số của xung đồng bộ ngang và bằng 15,74 kHz.

Âm nhạc

Phần âm nhạc cũng được tổ chức tương tự như chương trình Bộ tứ . Mỗi kênh âm thanh có một "sổ ghi chép" riêng, từ đó các ghi chú được thực hiện và phát lần lượt, các khoảng dừng được duy trì, các đoạn được lặp lại. Nhưng ở đây, trái ngược với "Bộ tứ", mỗi kênh âm thanh không có nhiệm vụ riêng và tất cả chúng đều được xử lý trong một nhiệm vụ - **Task_Music** .

Hoạt hình

Tại vì trò chơi có chứa các đối tượng hoạt hình: người đàn ông, con bướm, pháo sáng, kim cương, ngôi nhà, vụ nổ, sau đó bắt buộc phải cập nhật dữ liệu trong một khoảng thời gian nhất định để mã bản vẽ nằm trong trình xử lý ngắt biết sprite nào cho đối tượng này cần được chọn. Nhiệm vụ này lặp lại qua tất cả tám trăm ô của sân chơi và cập nhật hai bit ít quan trọng nhất cho tất cả các đối tượng hoạt hình, chịu trách nhiệm cho giai đoạn hoạt ảnh (tức là có 4 giai đoạn chuyển động cho mỗi đối tượng hoạt hình). Tại vì Vì tác vụ này có thể chạy trong một thời gian dài, sau khi xử lý mỗi bốn mươi ô, tác vụ sẽ trả lại quyền điều khiển cho bộ lập lịch.

Tác vụ xử lý hoạt ảnh **Task_Animate** chạy sau mỗi khung hình thứ 8 (tức là 7,5 lần mỗi giây) được vẽ.

Tro choi

Đây là phần dễ nhất của chương trình. Nhiệm vụ xử lý thuật toán trò chơi **Task_Game** được kích hoạt sau khi vẽ mỗi khung hình thứ 16 (4 lần mỗi giây) và thực hiện một bước cập nhật sân chơi. Chạy qua tất cả các ô của nó, nhiệm vụ tìm kiếm các đối tượng sẽ di chuyển, tính toán cho chúng, tùy thuộc vào quỹ đạo, vị trí tiếp theo và cũng tuân theo các quy tắc của trò chơi: ví dụ: va chạm của một đối tượng chuyển động (một người đàn ông hoặc một con bướm) với một hòn đá bay nên dẫn đến một vụ nổ; vụ nổ phá hủy mọi thứ trong bán kính 2 ô, ngoại trừ bức tường titan; một con bướm sau khi chết biến thành nhiều viên kim cương, v.v.

Vì tác vụ chạy trong một thời gian khá dài, nên đôi khi nó (sau khi kiểm tra mỗi bốn mươi ô) chuyển quyền kiểm soát cho bộ lập lịch để tác vụ tạo nhạc hoặc xử lý hiệu ứng âm thanh có thể kiểm soát. Sau khi kiểm tra và xử lý tất cả tám trăm ô của sân chơi, **Task_Game** sẽ gửi một tin nhắn ngắn đến nhiệm vụ **Task_Sound** kèm theo mã âm thanh sẽ đi kèm với bước này của trò chơi, chẳng hạn như âm thanh của một vụ nổ, đá rơi hoặc ăn. một viên kim cương. Nếu bước này không được kèm theo bất kỳ âm thanh nào, thì tin nhắn sẽ không được gửi.

Có lẽ, khi viết thuật toán trò chơi, có chỗ nào đó không giống với các quy tắc ban đầu (ví dụ, thứ tự các viên đá rơi vào nhau khi chúng rơi từ các phía khác nhau), nhưng bề ngoài thì mọi thứ lại rất giống nhau. Phiên bản trò chơi của tôi không có bốn hiệu ứng khác của Boulder Dash thông thường (tôi đang sử dụng thuật ngữ mà tôi đã sử dụng khi còn nhỏ): sinh khối tăng trưởng (để giết bướm), sinh khối không phát triển (cho phép đá và kim cương đi qua với sự chậm trễ), phát triển tường gạch (lấp đầy các khoảng trống bên trái và bên phải), bức tường vàng (biến đá thành kim cương và ngược lại). Thành thật mà nói, tôi chỉ lười biếng.

Điều khiển:

- sử dụng các nút "trái", "phải", "lên" và "xuống" để di chuyển người đàn ông nhỏ;
- sự kết hợp của bất kỳ nút nào trong số các nút này với nút "Fire" thực hiện một bước theo hướng tương ứng mà không di chuyển người đàn ông nhỏ bé (ví dụ: ăn viên kim cương ở bên phải, trong khi vẫn đứng yên);
- nếu đã xảy ra trường hợp người đàn ông nhỏ bé bị ném đá và anh ta không thể bước một bước theo một hướng, thì bạn có thể nhấn tất cả 4 nút hướng cùng một lúc - điều này sẽ dẫn đến việc khởi động lại cấp độ;
- bằng cách nhấn tất cả năm nút cùng một lúc, chúng ta sẽ chuyển sang một cấp độ khác.

Tạo cấp độ mới

Việc tạo cấp độ trò chơi của riêng bạn là tùy thuộc vào bạn. Tệp game_data.c khởi tạo mảng **const char MAP [] [] []** , mảng này lưu trữ bản đồ của các sân chơi. Số lượng bản đồ được xác định bởi **NUMBER_OF_MAPS** hằng số (trong chương trình của tôi = 3, nhưng bạn có thể tăng và tạo nhiều bản đồ hơn). Mỗi bản đồ là một mảng ký tự có kích thước (MAP_SIZE_Y + 1) x (MAP_SIZE_X + 1). Dòng đầu tiên chứa các định nghĩa màu sắc cho các loại đối tượng khác nhau cho một bản đồ cụ thể (xem hằng số CL_xxx), cũng như số lượng kim cương cần thiết để hoàn thành cấp độ. Tiếp theo là dòng MAP_SIZE_Y, trong đó bản đồ chính nó được thiết lập với mô tả về các đối tượng:

(không gian) - trống rỗng;
bức tường không thể phá vỡ bằng titan;
= - tường gạch;
. - Trái đất;
m - người đàn ông nhỏ bé;
h - ngôi nhà;
x - con bướm;
f - bình chữa cháy;
o - đá;
+ - kim cương.

Đường viền của bản đồ phải bao gồm một bức tường titan. Từ chính tệp nguồn, khá dễ dàng để tìm ra những gì và cách thay đổi những gì để tạo bản đồ của riêng bạn. Với một chút nỗ lực và một chút sửa đổi của chương trình, bạn có thể tạo bản đồ có thể tải được từ một EEPROM bên ngoài.

Xây dựng dự án

Dự án được tạo trong môi trường tích hợp MPLAB IDE [http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002] . Trình biên dịch HT-PICC18 STD 9.51pl2 được sử dụng để lắp ráp.

Tải xuống **các tệp hệ điều hành OSA** , giải nén kho lưu trữ vào ổ C: (thư mục C: \ OSA sẽ xuất hiện).

Giải nén tệp **vga_game.rar** vào thư mục C: \ TEST \ VGA. Thao tác này sẽ tạo một thư mục VGA_GAME bên trong. Mở dự án trong MPLAB IDE.

Ghi chú. *Khi giải nén vào một thư mục khác ngoài C: \ TEST \ VGA \ VGA_GAME, bạn sẽ cần sử dụng tùy chọn Project \ Build... \ Project trong tab Thư mục trong danh sách đường dẫn bao gồm để thay đổi đường dẫn đến tệp dự án đến một nơi bạn giải nén các tệp từ kho lưu trữ.*

Xây dựng bằng cách nhấn **Ctrl + F10** .

Sự kết luận

Tạo dự án này, tôi theo đuổi một số mục tiêu:

- Thứ nhất, tôi chỉ đang thực hành tối ưu hóa mã;
- Thứ hai, nó luôn thú vị những gì có thể được vắt ra khỏi bộ điều khiển;
- Thứ ba, dự án này là một ví dụ khác về việc sử dụng RTOS, và trong trường hợp này, việc sử dụng nó nhiều hơn là hợp lý, bởi vì. do thời gian dành cho việc vẽ hình ảnh quá lớn, tạo ra sự thiếu hụt tài nguyên bộ điều khiển tạm thời mà hệ điều hành cho phép sử dụng với hiệu quả tối đa, không bao gồm hiện tượng chậm và đơ máy.

Bản thân thiết bị có thể được cải thiện hơn nữa. Ví dụ: bằng cách sử dụng SRAM bên ngoài, bạn có thể mở rộng khả năng đồ họa (và số lượng màu cũng như loại bỏ giới hạn "1 màu cho mỗi sprite"). Bằng cách treo bộ điều khiển âm nhạc bên ngoài (ngay cả khi chỉ là PIC thứ hai), bạn có thể mở rộng đáng kể khả năng âm nhạc của thiết bị (ví dụ: tăng tốc độ lấy mẫu, tăng độ sâu bit PWM, tăng số lượng kênh âm thanh, thêm âm thanh bắt chước của các nhạc cụ và trống khác nhau, tạo âm thanh nổi, v.v.). Bạn có thể thêm một EEPROM bên ngoài để tải bản đồ, hình ảnh, nhạc và lưu trò chơi. Sau khi dỡ bỏ bộ điều khiển, chỉ để lại các quy tắc của trò chơi và đầu ra VGA trên đó (trên thực tế, cũng có thể được trải rộng trên các bộ điều khiển khác nhau), chúng tôi có thể phức tạp hóa thuật toán trò chơi, thêm các hiệu ứng video thú vị, v.v. vân vân.

Nếu ai đó có mong muốn tạo ra một tác phẩm thủ công thú vị cho chính họ (hoặc không cho chính họ), hãy thoải mái sử dụng các phát triển và nguồn của tôi, ngay cả cho mục đích nghiệp dư, thậm chí cho mục đích thương mại.

Chúc may mắn!

Victor Timofeev, tháng 5 năm 2009

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

Liên kết

Dưới đây là một số dự án đáng xem xét:

- http://bdash.gpio.dk/ [http://bdash.gpio.dk/] - gần với trò chơi gốc hơn trên ATMega
- http://avga.prometheus4.com/ [http://avga.prometheus4.com/] - một số trò chơi video màu với đầu ra VGA (ATmega).
- http://www.linusakesson.net/scene/craft/ [http://www.linusakesson.net/scene/craft/] - bản demo rất thú vị với hoạt ảnh và đa âm (ATmega).
- http://www.rickard.gunee.com/projects/ [http://www.rickard.gunee.com/projects/] - trò chơi màu "Tetris" và "Pong" với đầu ra TV (Scenix)
- http://www.belogic.com/uzebox/index.htm [http://www.belogic.com/uzebox/index.htm] - hệ thống video dựa trên ATmega644: 240x224 pixel, 256 màu. Bạn có thể nhúng nó vào chương trình của mình.

Và các liên kết khác:

- http://z80speccy.narod.ru/ [http://z80speccy.narod.ru/] - trình giả lập ZX Spectrum tuyệt vời
- http://andygame.narod.ru/zx/game/index.html [http://andygame.narod.ru/zx/game/index.html] - đồ chơi cho anh ta (bao gồm cả Boulder Dash)

(Nếu ai có bất kỳ liên kết thú vị nào khác về việc sử dụng hình ảnh VGA 8-bit, vui lòng gửi cho họ. Tôi sẽ thêm những liên kết đáng chú ý vào danh sách).

osa / posts / vga_game.txt Sửa đổi lần cuối: 26/11/2015 9:12 PM Bởi osa_chief