Văn bản đầu cuối VGA trên PIC18

Giới thiệu

Chúng tôi cung cấp cho bạn một chương trình dành cho PIC18, chương trình này thực hiện chức năng của một card màn hình đơn giản để hiển thị thông tin văn bản trên màn hình VGA. Tất nhiên, PIC18 không hoàn toàn phù hợp để giải quyết những vấn đề như vậy, tuy nhiên, ví dụ này cho thấy điều này không phải là không thể. Chương trình này được viết vào năm 2004 và được đưa vào đây để giải thích cách thức hoạt động của dự án Trò chơi điện tử PIC18 .

Bạn có thể tìm thấy mã nguồn của chương trình đầu cuối tại đây: terminal.rar

Vì vậy, bài viết mô tả một chương trình thực hiện các chức năng của một card màn hình với các đặc điểm sau:

Độ phân giải văn bản	30x30 ký tự
Độ phân giải đồ họa	240x480 pixel
Kích thước ma trận số hóa	8x16 pixel
Số lượng màu	mười lăm
Mô phỏng con trỏ	Có. Có thể thay đổi kích thước.
Bộ nhân vật	Chữ Latinh, chữ Kirin, bút danh
Giao diện điều khiển	SPI, lên đến 250 kbps

Mặc dù card màn hình có khả năng hiển thị 15 màu nhưng có một số hạn chế:

1. nền chỉ có thể là màu đen;
2. một ký tự chỉ được có một màu (không tính màu nền).

Đây là hình ảnh thử nghiệm được tạo ra bởi card màn hình (bản thân chương trình đã được viết vào năm 2004, nhưng tính năng màn hình demo chỉ được thêm vào cho mục đích trình diễn):

Điều khiển VGA

Một chút lý thuyết

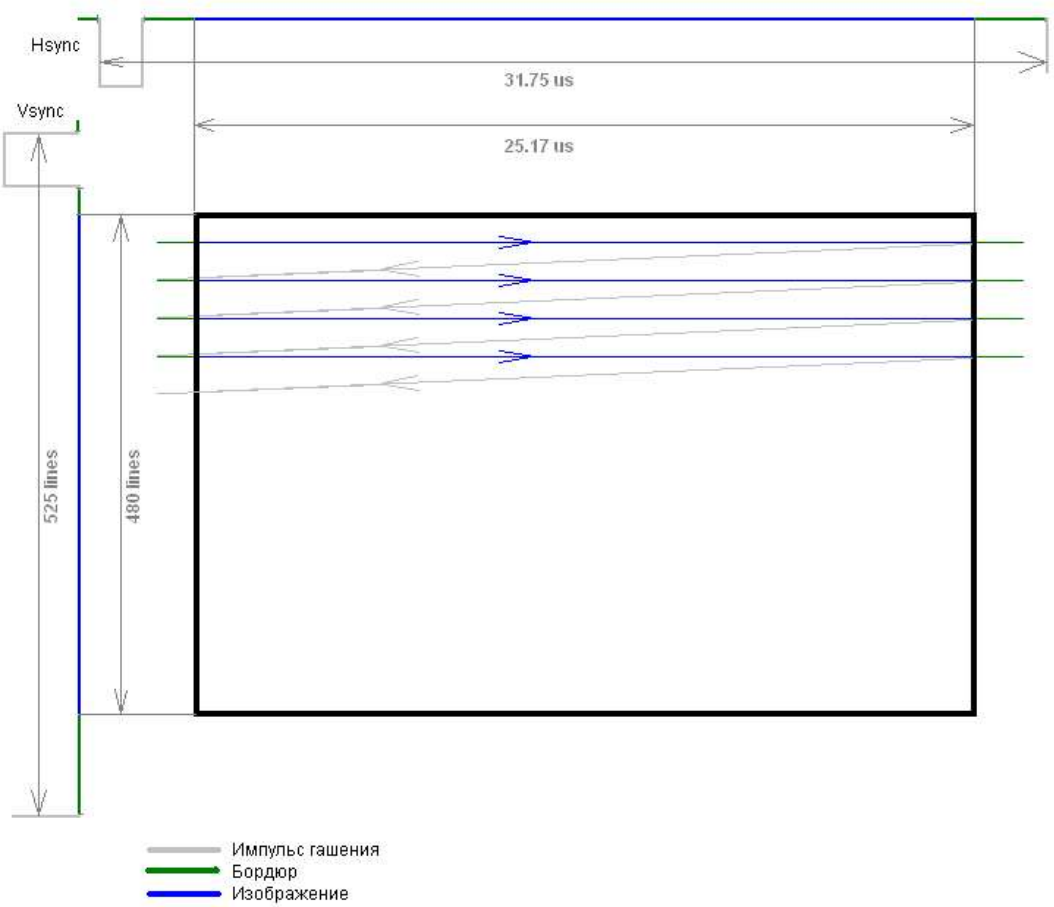
Màn hình VGA được điều khiển bởi 5 đường: 2 kỹ thuật số (đồng bộ ngang và dọc) và 3 tương tự (đỏ, xanh lá cây, xanh dương). Các vạch thời gian ở đó để cho màn hình biết khung hình đang bắt đầu từ đâu, độ phân giải và tốc độ khung hình là bao nhiêu. Các đầu vào tương tự được sử dụng để thiết lập màu sắc. Điện áp chúng ta áp dụng cho đầu vào tương tự càng cao, thì bóng của màu tương ứng mà chúng ta nhận được càng sáng. Bằng cách điều chỉnh các mức điện áp trên ba đường dây analog theo nhiều cách kết hợp khác nhau, có thể thu được các sắc thái màu khác nhau. Ví dụ: bằng cách áp dụng các mức giống nhau cho các đầu vào điều khiển màu đỏ và xanh lục (không có màu xanh lam), bạn có thể nhận được màu vàng.

Thông tin trên màn hình được vẽ từng dòng từ trên xuống dưới. Mỗi dòng được vẽ từ trái sang phải. Khi toàn bộ dòng được vẽ, chúng tôi áp dụng một xung đồng bộ ngang cho màn hình, xung này cho màn hình biết rằng phần thông tin tiếp theo sẽ được xuất ra từ dòng tiếp theo. Khi tất cả các đường được vẽ, chúng tôi tạo ra một xung đồng bộ dọc cho màn hình biết rằng khung đã kết thúc và thông tin sẽ theo đó để tạo thành khung tiếp theo.

Để mắt người không thể nhìn thấy hiện tượng nhấp nháy, tốc độ làm mới khung hình được chọn ít nhất là 60 Hz. Những thứ kia. 60 khung hình nên được vẽ mỗi giây. Mỗi khung ở chế độ VGA tiêu chuẩn bao gồm 525 dòng, trong đó 480 dòng là thông tin (chứa thông tin để hiển thị trên màn hình), và phần còn lại là các cạnh đầu và cuối của quá trình đồng bộ hóa, cũng như các đường viền dưới và trên và làm không mang thông tin màu sắc. Mỗi dòng cũng được chia hợp lý thành nhiều phần: các cạnh đầu và cuối của quá trình đồng bộ hóa, các đường viền bên trái và bên phải, thông tin video để hiển thị và xóa xung đồng bộ hóa.

Dễ dàng tính được rằng thời gian của một dòng = 1s / 60 khung hình / 525 dòng = 31,75 µs

Dưới đây là hình giải thích cách chuyển thông tin sang màn hình VGA:



Tạo hình ảnh

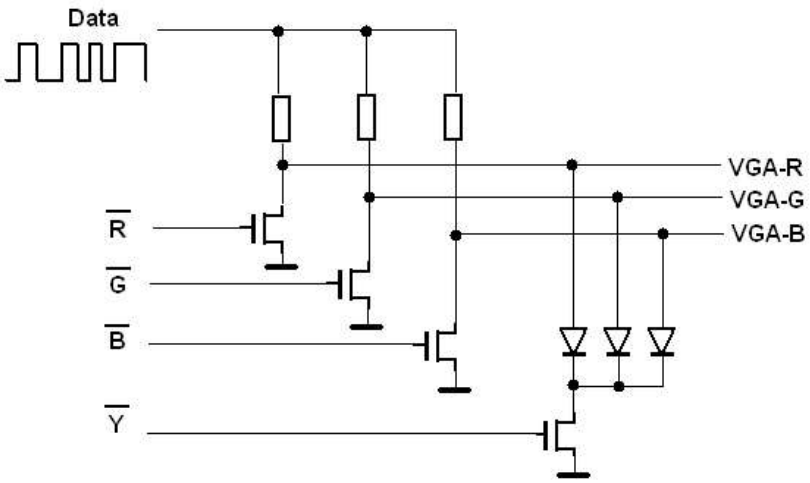
Thế hệ đồng hồ

Mọi thứ đều đơn giản ở đây. Tất cả những gì chúng ta cần là tạo ra các xung phân cực âm với thời gian 3,9 µs và chu kỳ không đổi là 31,75 µs. Đây sẽ là các xung đồng bộ ngang. Trong mỗi xung thứ 524 và thứ 525, chúng ta sẽ tạo thành một xung đồng bộ hóa theo chiều dọc âm. Rõ ràng, rất tiện lợi khi sử dụng bộ ngắt hẹn giờ cho mục đích này.

Có tính đến các yêu cầu về tính đồng bộ của bản thân các xung đồng bộ và các tín hiệu hình thành màu sắc, ở giai đoạn thiết kế, người ta đã quyết định đặt tất cả các mã trực tiếp tạo hình ảnh vào một ngắt. Thứ nhất, điều này sẽ cho phép bạn kết nối các tín hiệu RGB để đồng bộ hóa các xung một cách chặt chẽ, chính xác với đồng hồ và thứ hai, do đó, chúng tôi sẽ song song các quá trình tạo tín hiệu VGA với việc hình thành bộ nhớ video và hoạt động SPI.

Hình ảnh màu sắc

Chúng ta sẽ định hình các tín hiệu RGB như thế nào? Có một số tùy chọn, nhưng, với các chi tiết cụ thể của thông tin đầu ra, tức là văn bản màu, bao gồm các ký tự rộng 8 pixel, trong đó mỗi ký tự chỉ có thể được vẽ bằng một màu, nó đã được quyết định tổ chức đầu ra như sau (có một lý do khác, nhưng về nó sau một chút): với một đầu ra, chúng tôi tạo ra giá trị của pixel "sáng" / "tắt", chúng tôi điều chỉnh pixel với ba đầu ra thoát khí để có được màu sắc mong muốn và điều chỉnh độ sáng với một đầu ra. Về mặt sơ đồ, điều này có thể được biểu diễn như sau:



Khi đầu ra Dữ liệu ở "0", bất kể trạng thái của các đầu ra khác, tất cả các đường RGB sẽ ở "0", tương ứng với màu đen. Nhưng khi đầu ra Dữ liệu được đặt thành "1", thì trạng thái của các đường RGB sẽ được chọn bởi các tín hiệu bộ điều khiển ~ R, ~ G và ~ B. Nếu chân được đặt thành "0" và bóng bán dẫn tương ứng được đóng lại, thì "1" từ đầu ra Dữ liệu sẽ được chuyển đến điều khiển màu VGA. Nếu đầu ra được đặt thành "1" và bóng bán dẫn tương ứng đang mở, thì dòng điều khiển màu tương ứng sẽ là "0". Do đó, bằng cách tạo các số không và các số không ở đầu ra Dữ liệu, chúng ta có thể đặt màu cho mỗi điểm sáng bằng cách đặt các chân ~ R, ~ G, ~ B thành giá trị mong muốn. Ví dụ, để có được điểm màu xanh lam, chúng ta cần loại bỏ thành phần màu đỏ, để lại màu xanh lam và xanh lục. Những thứ kia.

Chúng ta còn một lối thoát nữa - ~ Y. Đây là điều khiển độ sáng. Khi nó được đặt thành "0", bóng bán dẫn sẽ đóng lại và các mức điện áp trên đường VGA-R, VGA-G và VGA-B sẽ được hình thành bởi một bộ chia điện trở, phía trên của nó là điện trở trong cổng của các bóng bán dẫn điều khiển màu, và phía dưới là điện trở đầu vào VGA = 75 Ohm. Nhưng khi đầu ra ~ Y được đặt thành "1", cả ba đường điều khiển màu được nhấn xuống đất qua các điốt, do đó giảm điện áp trên các đường dây hoạt động xuống 0,35V (sử dụng điốt 1N5819).

Do đó, chúng tôi nhận được 16 màu: 8 bình thường + 8 giảm độ sáng. (Trên thực tế, chúng tôi nhận được 15 màu, vì cách tiếp cận này tạo ra hai màu đen.)

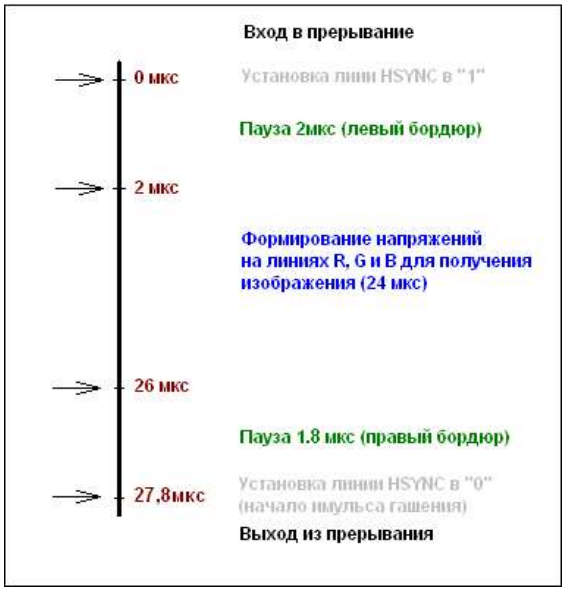
Trong thực tế, chúng ta có thể triển khai các bóng bán dẫn xả mở trên chính bộ điều khiển bằng cách đặt chốt đầu ra thành "0" và điều khiển thanh ghi hướng chân (TRIS). Do đó, chỉ có điốt và điện trở sẽ có mặt trong thiết bị cài đặt màu.

Sắp xếp theo thứ tự pixel

Như có thể thấy từ hình giải thích thứ tự đồng bộ hóa, phần nhìn thấy của đường kéo dài 25,17 µs. Điều này có nghĩa là, có một bộ điều khiển với hiệu suất 10 MIPS (tại thời điểm viết chương trình, đây là hiệu suất tối đa cho PIC18), tức là với thời gian chu kỳ chương trình = 100 ns, chúng ta có thể tạo một vùng có thể xem được với độ phân giải ngang tối đa là 251 pixel. Nhưng ở đây có thể lập luận rằng cần ít nhất một chu kỳ đồng hồ để tạo dữ liệu và một chu kỳ đồng hồ cho đầu ra, tức là đã 200 ns. Nhưng chúng tôi sẽ sử dụng khả năng của các thiết bị ngoại vi, cụ thể là, chúng tôi sẽ sử dụng mô-đun USART, mô-đun này có thể tạo cho chúng tôi một chuỗi gồm 8 số không và những số không có thời lượng 100 ns mỗi số (đối với điều này, bạn sẽ cần chạy USART đồng bộ ở tốc độ tối đa). Do đó, chúng ta chỉ cần hai chu kỳ để tạo thành 8 pixel . Và 6 chu kỳ khác vẫn còn cho tất cả các loại nhu cầu bổ sung: đọc bảng tạo ký tự và hình thành màu sắc.

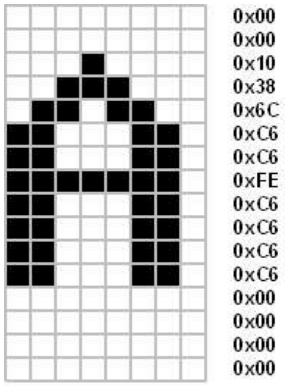
(Sử dụng USART là lý do thứ hai tại sao chúng tôi sử dụng một chân điều khiển duy nhất để tạo chuỗi pixel "bật" / "tắt").

Tại vì nó phải sử dụng phông chữ có chiều rộng 8 pixel, khi đó độ phân giải sẽ là bội số của 8, tức là tối đa có thể là 248 điểm (31 ký tự). Tuy nhiên, để đơn giản và tiện lợi, chúng ta sẽ lấy 30. Như vậy, độ phân giải ngang của hình ảnh video sẽ là 240 pixel. Để hiển thị hình ảnh, sau khi tạo xung đồng bộ, chúng ta cần tạm dừng (đường viền bên trái), sau đó vẽ đường hình ảnh hiện tại, sau đó lại tạm dừng (đường viền bên phải). Thứ tự xử lý ngắt sẽ như sau:

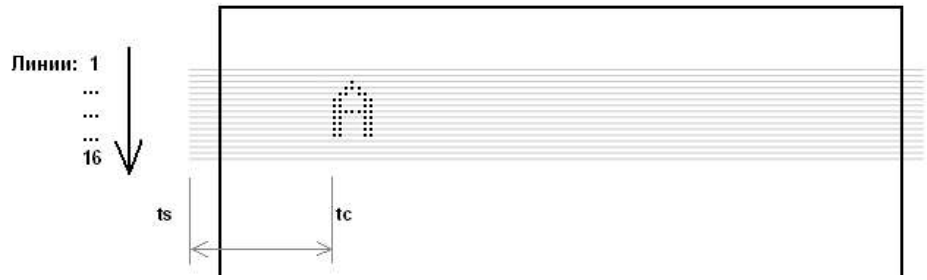


Với tổ chức này, chúng ta có 3,9 µs từ mỗi dòng (39 lệnh, trừ đi 5-6 cho đầu vào / đầu ra để ngắt và đặt lại bit ngắt) để xử lý dữ liệu từ SPI. Các lần tạm dừng ở đây được đánh dấu có điều kiện, tức là trong thực tế, thời gian này được sử dụng cho các tính toán liên quan (mô phỏng con trỏ, đếm dòng, tạo con trỏ).

Chương trình của chúng tôi sử dụng phông chữ rộng 8 pixel. Trình tạo ký tự được lưu trữ trong ROM nội bộ của bộ điều khiển và là một mảng trong đó 16 byte được phân bổ cho mô tả của mỗi ký tự (một byte cho mỗi dòng raster). Ví dụ, chữ 'A' được viết như thế này:



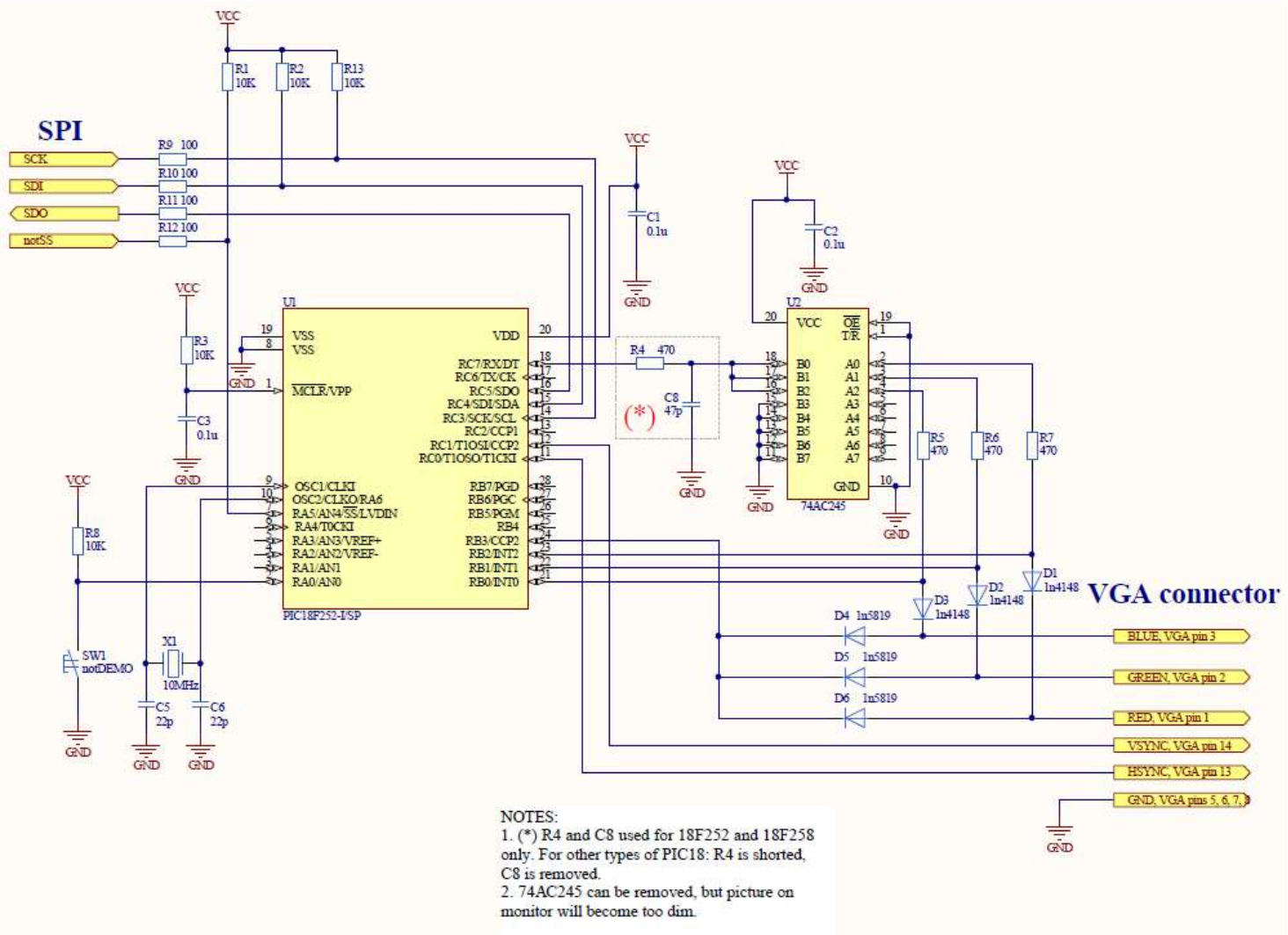
Và để xuất ra ký tự này, trong quá trình vẽ 16 dòng, trong cùng khoảng thời gian sau khi bắt đầu xung đồng bộ ngang, chúng tôi sẽ ghi vào thanh ghi TXREG tất cả các byte mô tả ký tự này: ở dòng đầu tiên và dòng thứ hai, byte 0x00, ở thứ ba - 0x10, ở thứ tư - 0x38, v.v. Sau đó, nó sẽ giống như thế này trên màn hình:



Ở đây ts là đầu dòng, tc là đầu ký tự.

Để xuất ra một dòng raster của mỗi ký tự, chúng ta có 8 chu kỳ. Trong thời gian này, chúng ta phải có thời gian để chọn byte hiện tại từ mảng tạo ký tự cho ký tự tiếp theo và chuẩn bị giá trị màu cho nó. Đúng 8 chu kỳ sau khi bắt đầu đầu ra của ký tự trước đó, chúng tôi xuất ra chu kỳ tiếp theo, sau 8 chu kỳ khác - chu kỳ tiếp theo, và như vậy tất cả 30 ký hiệu đều được xuất.

sơ đồ mạch



Hai từ về một số giải pháp mạch điện.

Trước hết, bạn cần chú ý đến chip 74AC245 tùy chọn, nhưng mong muốn. Chức năng chính của nó là dỡ tải đầu ra của RC7. Tại vì Vì các đầu vào đường màu VGA có điện trở thấp (75 ôm), nên dòng điện mà chúng truyền động là tương đối cao. Chân RC7 phải làm việc với tải có điện trở thấp trên ba dòng cùng một lúc: điều khiển màu đỏ, xanh lá cây và xanh lam. Về nguyên tắc, bạn không thể lắp cái vi mạch này mà tải cả cái vào RC7 mà hình trên màn hình sẽ hơi đơ.

Thứ hai, chúng ta hãy chú ý đến chuỗi RC ở đầu ra của RC7 (R4-C8). Khi gỡ lỗi chương trình, một sự khác biệt được tìm thấy trong hoạt động của bộ điều khiển PIC18F252 (sự khác biệt tương tự cũng được tìm thấy đối với PIC18F258, PIC18F452 và PIC18F458) với mô tả trong tài liệu: nó nói rằng xung USART hoạt động ở chế độ đồng bộ với tốc độ tối đa là đồng bộ với đồng hồ Q4. Tuy nhiên, trên thực tế hóa ra chúng được đồng bộ hóa với đồng hồ Q2. Để đặt màu cho từng ký tự, thanh ghi màu (TRISB) đã được cập nhật đồng thời với thời điểm bắt đầu truyền byte tiếp theo qua USART. Vấn đề là việc ghi vào thanh ghi TRISB trong các bộ điều khiển này được thực hiện chính xác trên đồng hồ Q4. Những thứ kia. ghi vào thanh ghi điều khiển màu không đồng bộ với thời điểm bắt đầu truyền byte tiếp theo qua USART trên Q4-Q2 = 50 ns (ở tần số xung nhịp 40 MHz). Trên màn hình, sự không đồng bộ này thể hiện dưới dạng "bò" màu của các ký tự liền kề chồng lên nhau. Để loại bỏ hiệu ứng này, một mạch RC được đưa vào mạch, làm trễ tín hiệu từ đầu ra RC7 đi 50 ns.

Ghi chú . *Sự khác biệt này chỉ được tìm thấy trên PIC18F252, PIC18F258, PIC18F452 và PIC18F458. Nếu bộ điều khiển của một sửa đổi mới hơn được sử dụng (ví dụ: 2520), thì chuỗi RC không cần phải được cài đặt.*

Tổ chức chương trình

Cấu trúc bộ nhớ video

Bộ nhớ video chứa các thông tin sau:

- mảng ký tự hiển thị trên màn hình (8 bit mỗi ký tự);
- mảng màu cho tất cả sự quen thuộc (4 bit cho mỗi ký tự).

Trường màn hình được chia thành 900 không gian ký tự (30 theo chiều dọc và 30 theo chiều ngang). Vì vậy, nó cần 900 byte để lưu trữ các ký tự và 450 byte để lưu trữ mảng màu. Tính đến các yêu cầu để giảm thiểu thời gian xử lý dữ liệu trong hai mảng này, chúng tôi đã quyết định tổ chức bộ nhớ video dưới dạng một mảng các khối 3 byte:

xy	XX	YY

ở đâu:

- x - 4 bit xác định màu của ký tự XX;
- y - 4 bit xác định màu của biểu tượng YY;
- XX - một ký tự ở vị trí ký tự chẵn;
- YY - một ký tự ở vị trí ký tự lẻ.

Các khối ba byte này được lưu trữ tuần tự trong bộ nhớ dưới dạng một mảng 450 khối. Tổ chức bộ nhớ này cho phép đánh địa chỉ hai mảng cùng một lúc (một mảng ký tự và một mảng màu) với cùng một con trỏ FSR.

Phân phối SFR

Để không lãng phí thời gian khan hiếm vào việc hình thành các thanh ghi FSR để chỉ ra biểu tượng hoạt động (để hiển thị hoặc để viết), cũng như một số thanh ghi đặc biệt khác, chương trình được tổ chức theo cách mà một số thanh ghi nhất định được gán các chức năng nhất định. trong toàn bộ quá trình thực hiện chương trình:

FSR1 luôn trỏ tới khối đang hoạt động trong mảng bộ nhớ video đang được ghi vào (thực tế là vị trí con trỏ hiện tại).
FSR2 Luôn trỏ đến ký tự đang hiển thị
TBLPTR Luôn trỏ đến mảng trình tạo ký tự (đến dòng hiện tại; xem trình tạo ký tự để biết chi tiết)
PCLATH Luôn chứa giá trị 0x4 - byte cao của địa chỉ chương trình con xử lý SPI (vì có một bảng nhảy, quyền truy cập vào được thực hiện bằng cách gán thanh ghi PCL)

Việc sửa lỗi đăng ký này sẽ tiết kiệm thời gian có thể dành cho:

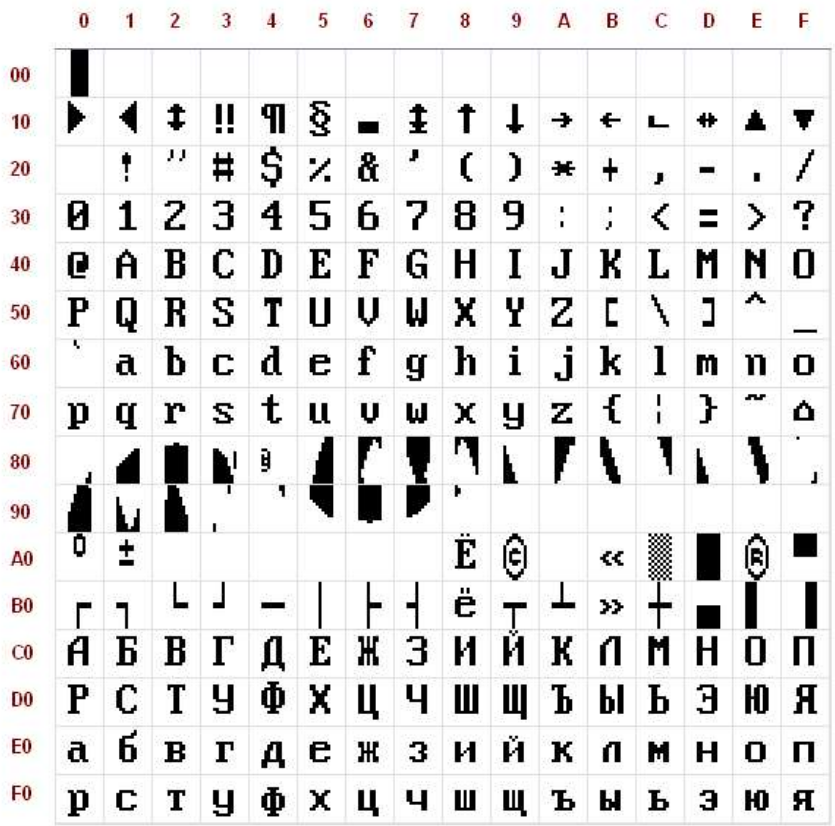
- thế hệ con trỏ (cả RAM và ROM);
- lưu / khôi phục SFR khi vào / ra đến / từ một ngắt.

trình tạo ký tự

Trình tạo ký tự được tổ chức dưới dạng một mảng byte để lưu trữ thông tin khoảng 256 ký tự, 16 byte cho mỗi ký tự. Chúng ta cần đặt những dữ liệu này theo một thứ tự nhất định để giảm thiểu thời gian lấy mẫu của byte cần thiết để hiển thị trên màn hình, có tính đến thực tế là trong một lần chuyển (để vẽ một dòng raster) từ tất cả 16 byte mô tả một ký tự cụ thể, đối với mỗi ký tự sẽ chỉ được chọn một byte và với cùng một chỉ mục. Tức là, khi xuất ra dòng số 5, chúng ta chỉ cần 5 byte trong số 16 byte mô tả mỗi ký tự. Do đó, mảng tạo ký tự được tổ chức không phải là một mảng gồm các khối 16 byte, mỗi khối mô tả một ký tự, mà là một mảng gồm 16 khối 256 byte, mỗi khối chứa một byte cho mỗi ký tự, sẽ được xuất ra. khi vẽ các đường hiện tại.

Một tổ chức như vậy sẽ cho phép, khi vẽ dòng tiếp theo của raster, một lần để tạo giá trị của thanh ghi TBLPTRH, thanh ghi này sẽ trỏ đến một khối 256 byte và đối với mỗi ký tự, chỉ cần ghi giá trị của nó vào thanh ghi TBLPTRL, do đó lấy địa chỉ byte để hiển thị trên màn hình. Với cách tiếp cận này, chúng tôi có một yêu cầu: bảng tạo ký tự phải được căn chỉnh trên ranh giới 256 byte. Trong chương trình của chúng tôi, trình tạo ký tự sẽ được lưu trữ tại địa chỉ 0x1000.

bảng ký hiệu



(Các ký tự có mã 0x80 đến 0x98 được sắp xếp thành 5 hàng gồm 5 ký tự tạo thành đồ họa logo Vi mạch.)

con trỏ

Thiết bị đầu cuối có khả năng hiển thị con trỏ nhấp nháy ở vị trí hiện tại. Điều này có thể hữu ích để tổ chức đầu vào của người dùng thuận tiện hơn. Tốc độ nhấp nháy của con trỏ được xác định bởi khoảng thời gian TMR0 (con trỏ nhấp nháy ở tần số 2,5 Hz). Đầu ra của con trỏ trong chương trình được tổ chức như sau: khi vẽ dòng ký tự chứa con trỏ, 2,5 lần mỗi giây, ký tự tại vị trí đặt con trỏ được thay thế bằng một ký tự có mã 0x00 (nó là một ký tự được điền), và sau khi vẽ đường, nó sẽ được khôi phục.

Người dùng có thể thay đổi kích thước con trỏ. Kích thước con trỏ càng lớn thì số dòng raster tương ứng với dòng chứa con trỏ sẽ bị thay thế càng lớn.

Trình diễn

Nếu bạn giữ đầu vào RA0 ở "0" khi bật nguồn, chương trình sẽ vào chế độ demo, trong đó ảnh thử nghiệm sẽ được hiển thị (xem ở trên). Ngay sau khi "1" xuất hiện trên RA0, màn hình sẽ bị xóa và chương trình bắt đầu hoạt động bình thường.

Quản lý thiết bị đầu cuối

Giao diện SPI

Thiết bị đầu cuối của chúng tôi được điều khiển bởi một thiết bị bên ngoài thông qua SPI trên bốn đường: notSS, SDI, SDO, SCK. Khi truyền đến đầu cuối (hoặc nhận từ đầu cuối) mỗi byte, dòng notSS phải được đặt thành 0, và khi hoàn thành quá trình truyền (nhận) byte, nó phải được đưa về trạng thái đơn. Dữ liệu (đầu vào và đầu ra) được đặt xung nhịp trên cạnh xuống của đường SCK. Do phần lớn thời gian chương trình bận vẽ hình ảnh, và trong số 31,5 µs, chỉ còn lại khoảng 2,5 µs, tức là 7,5% thời gian. Điều này đặt ra một số hạn chế về tốc độ trao đổi dữ liệu với thiết bị đầu cuối. Chúng tôi đã quản lý để tổ chức chương trình theo cách mà trong 2,5 µs một byte dữ liệu sẽ có thời gian được xử lý hoàn toàn, tức là trong quá trình vẽ một dòng của raster, thiết bị đầu cuối có thể nhận và xử lý đầy đủ một byte. Bằng cách này, tốc độ lặp lại byte không được vượt quá tần suất vẽ đường raster, tức là 1 / 31,5 µs ~ 31,5 KB / s

Những thứ kia. Tần số SPI không được vượt quá **250 kbps** .

Đầu ra màn hình

Để hiển thị một ký tự trên màn hình ở vị trí con trỏ hiện tại, bạn chỉ cần gửi đến thiết bị đầu cuối thông qua SPI mã của nó từ 0x10 đến 0xFF. Các byte 0x00 đến 0x0F được dành riêng cho các lệnh (xem các lệnh điều khiển). Trong đó:

1. ký tự nhận được sẽ được hiển thị ở vị trí con trỏ hiện tại;
2. ký tự này sẽ được đặt thành màu hiện tại;
3. vị trí hiện tại tăng thêm 1. Nếu nó ở cuối dòng, thì nó được chuyển sang đầu dòng tiếp theo. Nếu vị trí ở cuối màn hình (góc dưới bên phải), thì vị trí con trỏ mới sẽ được đặt ở đầu màn hình (góc trên bên trái).

Các lệnh điều khiển

Các byte 0x00-0x0F được dành riêng để gửi các lệnh điều khiển đến thiết bị đầu cuối (thiết lập vị trí con trỏ, màu hiện tại, kích thước con trỏ, v.v.)

Mật mã	Đội	Tùy chọn	Sự mô tả
00	SPI_CMD_NOP	-	Đội không làm gì cả
01	SPI_CMD_SET_X	X	Đặt vị trí con trỏ theo chiều ngang (0..X_SIZE-1) (ví dụ: 1)
02	SPI_CMD_SET_Y	Y	Đặt vị trí con trỏ theo chiều dọc (0..Y_SIZE-1) (ví dụ: 1)
03	SPI_CMD_SET_CURSOR	S	Lựa chọn kích thước con trỏ (0..15)
04	SPI_CMD_SET_COLOR	C	Đặt màu hiện tại (0..15)
05	SPI_CMD_SET_SYMBOL_COLOR	C	Đặt màu của ký tự tại vị trí con trỏ hiện tại (0..15). Màu hiện tại không thay đổi.
06	SPI_CMD_CLRSCR	-	Xóa màn hình, đặt màu hiện tại = 15, đặt con trỏ thành 0,0. Việc thực hiện lệnh này mất nhiều thời gian (khoảng 2,5 ms). Trong thời gian này, thiết bị đầu cuối sẽ không xử lý bất kỳ lệnh nào khác ngoại trừ lệnh SPI_CMD_CHECK. Có hai cách để xác định rằng việc thực thi một lệnh đã hoàn tất: thời gian chờ tạm dừng 2,5 ms hoặc yêu cầu trạng thái thực thi của lệnh này bằng lệnh SPI_CMD_CHECK cho đến khi nhận được phản hồi 0x55
07	SPI_CMD_CHECK	trạng thái (pr.2)	Lệnh kiểm tra trạng thái thực hiện lệnh xóa màn hình. Trả về 0x55 nếu quá trình xóa màn hình hoàn tất (hoặc hiện chưa được thực hiện)
08	SPI_CMD_GET_SYMBOL	char (Ví dụ: 2)	Đọc ký tự từ bộ nhớ video ở vị trí con trỏ hiện tại
09	SPI_CMD_GET_X	X (pr.2)	Đọc vị trí ngang hiện tại của con trỏ
0A	SPI_CMD_GET_Y	Y (ví dụ: 2)	Đọc vị trí con trỏ hiện tại theo chiều dọc
0B	SPI_CMD_GET_CURSOR	S (Ví dụ: 2)	Đọc kích thước con trỏ hiện tại
0C	SPI_CMD_GET_COLOR	C (ví dụ: 2)	Đọc màu hiện tại
0D	SPI_CMD_GET_SYMBOL_COLOR	C (ví dụ: 2)	Đọc giá trị màu của ký tự tại vị trí con trỏ hiện tại
0E	SPI_CMD_GET_X_SIZE	X_SIZE (pr.2)	Nhận kích thước màn hình theo chiều ngang (Ví dụ: 3)
0F	SPI_CMD_GET_Y_SIZE	Y_SIZE (pr.2)	Nhận kích thước màn hình theo chiều dọc (Ví dụ: 3)

Ghi chú :

1. Giá trị của các hằng số X_SIZE và Y_SIZE có thể được lấy bằng các lệnh 0x0E và 0x0F tương ứng;
2. Việc đọc bất kỳ tham số nào từ thiết bị đầu cuối được thực hiện theo ba bước:
   I. một lệnh được gửi đến thiết bị đầu cuối (xem bảng);
   II. thời gian tạm dừng ít nhất 32 µs được duy trì;
   III. byte được đọc.
3. Các lệnh này đã được thêm vào để cho phép mở rộng chương trình đầu cuối. Ví dụ: khi chuyển chương trình sang bộ điều khiển mạnh hơn (PIC24, PIC32, ATMEGA, v.v.), có thể tăng độ phân giải và do đó, số ký tự trên mỗi dòng. Hoặc ai đó quyết định làm lại chương trình để kích thước ký tự không phải là 8x16, mà là 8x8, do đó tăng số lượng dòng văn bản.

Văn bản nguồn

Bạn có thể tìm thấy mã nguồn của chương trình đầu cuối tại đây: terminal.rar

Thư viện

Để thuận tiện cho việc phát triển các ứng dụng điều khiển thiết bị đầu cuối của riêng bạn, một tệp thư viện đã được tạo. Nó nằm trong thư mục APPNOTE. Tệp "vga_terminal.c" được thêm vào dự án và tệp "vga_terminal.h" được bao gồm trong tất cả các tệp mà từ đó nó được cho là gọi các hàm đầu cuối bằng cách sử dụng chỉ thị #include.

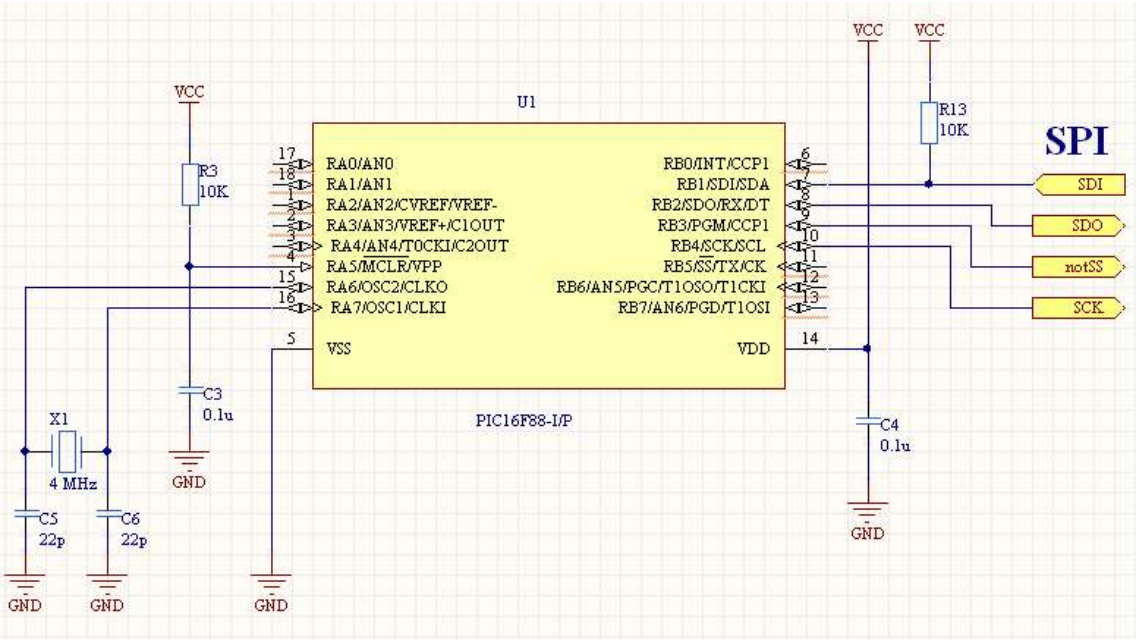
Dưới đây là danh sách các hàm C của mô-đun vga_terminal.

Hàm số	Sự mô tả

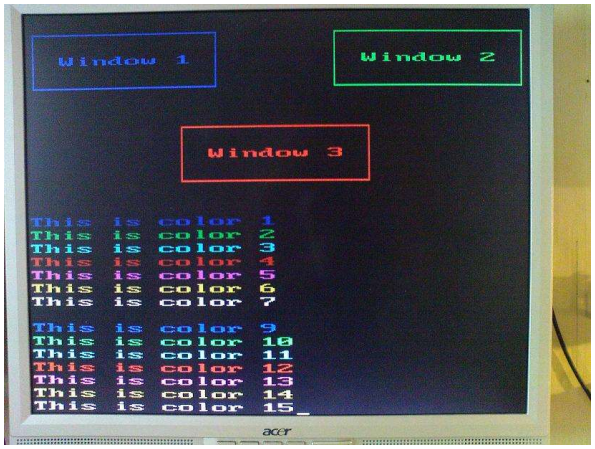
Đầu ra dữ liệu	
void **vga_outchar** (char c);	In ký tự ở vị trí con trỏ hiện tại
void **vga_outcharxy** (char x, char y, char c);	Xuất ký tự tại vị trí con trỏ đã chỉ định.
void **vga_outtext** (const char * t);	Văn bản đầu ra (một chuỗi kết thúc bằng rỗng nằm trong ROM) tại vị trí con trỏ hiện tại
void **vga_outtextxy** (char x, char y, const char * t);	Văn bản đầu ra (một chuỗi được kết thúc bằng null nằm trong ROM) tại vị trí được chỉ định.
void **vga_outstr** (char * s);	Văn bản đầu ra (một chuỗi kết thúc bằng rỗng nằm trong RAM) tại vị trí con trỏ hiện tại
void **vga_outstrxy** (char x, char y, char * s);	Văn bản đầu ra (một chuỗi kết thúc bằng null nằm trong RAM) ở vị trí được chỉ định
void **vga_window** (char x1, char y1, char x2, char y2, char color);	Vẽ một hình hộp chữ nhật trống tại các tọa độ xác định với đường viền có màu xác định.
char **vga_getchar** (vô hiệu);	Đọc ký tự ở vị trí con trỏ hiện tại
Vệ sinh màn hình	
void **vga_clrscr** (void);	Bắt đầu làm sạch màn hình. Sau hàm này, bạn phải tạm dừng trong 2,5 mili giây hoặc kiểm tra việc kết thúc quá trình thực thi của nó bằng hàm vga_check () (cho đến khi nó trả về "1")
char **vga_check** (vô hiệu);	Trả về "1" nếu thiết bị đầu cuối đã sẵn sàng nhận dữ liệu. Nếu không, trả về "0". Nó chủ yếu được sử dụng để kiểm tra việc hoàn thành vệ sinh màn hình.
Quản lý màu sắc	
void **vga_setcolor** (màu ký tự);	Đặt màu hiện tại.
char **vga_getcolor** (void);	Đọc màu hiện tại.
char **vga_getcharcolor** (void);	Đọc màu của ký tự tại vị trí con trỏ hiện tại
Điều khiển con trỏ	
void **vga_gotoxy** (char x, char y);	Di chuyển con trỏ đến vị trí được chỉ định
void **vga_setcursorsize** (kích thước ký tự);	Đặt kích thước con trỏ (0 - ẩn)
char **vga_getx** (void);	Nhận vị trí X hiện tại của con trỏ
char **vga_gety** (void);	Nhận vị trí Y hiện tại của con trỏ
char **vga_getwidth** (void);	Nhận chiều rộng màn hình (tính bằng ký tự)
char **vga_getheight** (void);	Nhận chiều cao màn hình (tính bằng ký tự)

ứng dụng thử nghiệm

Một chương trình PIC16F88 nhỏ thể hiện việc sử dụng thư viện vga_terminal để điều khiển thiết bị đầu cuối: terminal_test.rar Phần cứng SPI được sử dụng để giao tiếp.



Chương trình hiển thị hình ảnh sau:



Người đọc Thủ công mỹ nghệ

Thiết bị đầu cuối có độ phân giải 51x30 ký tự - độc giả Igor Kovalev [mailto:igorkov6@gmail.com] đã thiết kế lại thiết bị đầu cuối để hoạt động ở tần số 16MHz (PIC18F46K20). Bằng cách tăng tốc độ, anh ta đã tăng độ phân giải theo chiều ngang; các ký tự được thu hẹp trực quan, điều này làm cho hình ảnh hấp dẫn hơn

Liên kết

Cổng VGA trên PIC24 - ký tự 60x25, 8 màu, 2 cỡ chữ, bàn phím PS / 2, USART. Lược đồ, nguồn, mô tả tuyệt vời (tương tác)

http://www.micro-examples.com/public/microex-navig/doc/089-pic-pal-tv.html [http://www.micro-examples.com/public/microex-navig/doc/089-pic-pal-tv.html] - Thư viện video PIC PAL - dự án dành cho PIC18F4620 được viết bằng mikorC

http://www.mikrocontroller.net/topic/53140 [http://www.mikrocontroller.net/topic/53140] - đầu cuối văn bản trên điều khiển ATmega8, 40x25, UART 19200 (đầu ra PAL)

http://neil.franklin.ch/Projects/SoftVGA/ [http://neil.franklin.ch/Projects/SoftVGA/] - Một dự án atmega32 khác.

http://mondo-technology.com/dcvideo.html [http://mondo-technology.com/dcvideo.html] - Thiết bị đầu cuối văn bản PIC16F819 với đầu ra video. Độ phân giải 20x10 ký tự, điều khiển: UART 9600

osa / posts / vga_terminal.txt Sửa đổi lần cuối: 18/06/2011 9:02 PM Bởi osa_chief