Search 8052.com...
Search!

User (Email)
Password
Remember Logon
Login

Forgot password?
Create Account

8052 Store
Main
Message Board
Tutorials
8052 FAQs
8052.com Book
News
Code Library
Chips
Challenges
Links
Books
Reviews
Consultants
User Pages
8052.com SBC
8052 Video
8052 TCP/IP
8052 CD-ROM
Disassembler
Site Members
Contact Us
About VIS
Legal Info
Privacy Policy

8051 Television Video

8051 Based TV Display: In the course of a recent 8052.com thread , using '51 variants as audio/video co-processors came up. In response, i spent last Sunday implementing a micro only solution for a modest 256 x 190 pixel monochrome video generator for an equally modest 5" monochrome TV. What resulted was a software based, bit-mapped, NTSC video generator.

Utimately intending to build a VGA class display around a '51 platform, the first step was a direct video television display in hopes of leaving a bit-mapped serial terminal behind.

As an initial platform, i re-used a DS80C420 equipped PCB, removing it's SPI serial chip and installing a "kludge DAC" to drive the TV. &nbspAs this board had no provision for external XRAM memory, the 1kB SRAM, integrated into the microcontroller, was all that was available for Video RAM. &nbspA consequence of this is that the display format can be seen to repeat as this 1k is "looped" for display.

Quick Summary: *The generation of this level of video is easily in the grasp of modern microcontrollers.*

Available processor cycles occur with regular and predictable timing allowing additional tasks such as connected device interface and display formatting to also occur.

Generating a Video Display Typically, monochrome video data displays use a Master Clock set to the desired pixel output rate. &nbspThis clock drives a counter chain connecting to comparitor circuits which decode Horizonal Sync (at the end of each "display line"), Vertical Sync (at the end of each Frame of Horizonal lines), and the "displaying region" of a frame. &nbspThese signals, plus Video Data (shifting out during the displaying time of the cycle) input to a combining network that translates these signals to specific voltage levels.

In the late seventies, single chip controllers appeared which reduced the chip count by including a programable timing chain, comparitors, and memory access controllers. &nbspThese chips became the core of many computer's character and graphic capable displays.

Microcontroller Video: Microcontroller video replaces a Video Timing Chip and the typical outboard electronics with software generating a video output using a simple combiner.

The WEB has many examples of microcontroller character and graphic video TV interfaces. All are simple, and are based across PIC, Scenix, and AVR microcontroller types. &nbsp8051 afficionados have Myke Predko's interlaced NTSC signal generator which produced a 4 by 2 line character display using a video modulator, published in his book, "Customizing and Programming the 8051 Microcontroller". &nbspMyke reports that as he devoted every available cycle of a DS89C520 (11.0592MHz) to the video task which precluded it's originally intentioned use (a serial terminal).

Project Definition: If a useful *Terminal* was to result, Text and Graphic capability would have to have enough depth to generate eye-pleasing display quality. &nbspUsing a DS80C420, some buffers and few resistors, the demonstration platform documented generates all timing and issues all video data. The source code is available in this zip file (METALINK Assy).

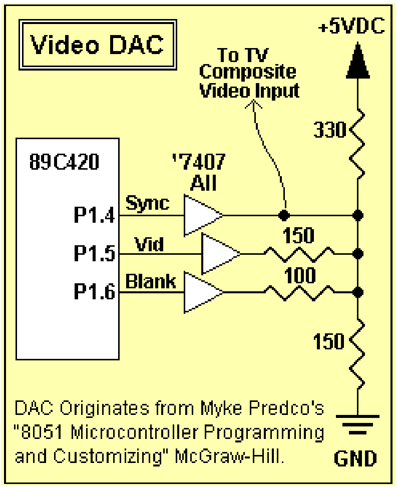
The generated video is "driving" the direct "Video In" of an inexpensive ($30 US) monochrome TV. &nbspThe avoidance of a Video Modulator is probably key to the quality of the display.

Prediction: This approach for video generation can succeed in producing a true monochrome bit mapped "terminal" addressable to the pixel. &nbspThree efforts required for this are;

- 1) Physical Video Generation; (basically what is demonstrating now).
- 2) Logical buffer/bit/pixel translation; (abstracting physical addresses with logical Pixel/Rox addressing)
- 3) Interface Service. (19.2 serial interface intended).

DS80C420 Specific Features: The DS'420 clock doubler results in an cycle rate @ 22.1184 MHz. This rate of execution was needed to achieve the granularity of timing required. Other DS'420 specific implementation includes the "auto-inc" and "auto-toggle" feature for DPTR. &nbspMost of the high speed '51 variants could execute this code with minor alterations.

A final characteristic of the DS'420 is that it only provides 1kB of integrated SRAM. &nbspThis is why the display format in the reference picture depicts repeated pattern. &nbspThe next incarnation will be provided the ~6kB the display requires.


Video DAC
+5VDC
To TV Composite Video Input
330
89C420
'7407 All
P1.4
Sync
P1.5
Vid
150
P1.6
Blank
100
150
DAC Originates from Myke Predco's "8051 Microcontroller Programming and Customizing" McGraw-Hill.
GND

TV Interface: The electrical interface to the TV is a "built out" DAC comprised of a few '7407 open collector outputs driving series resistors which parallel the shunt leg of a voltage divider. This DAC implementation models that used by Myke Predko in his Project 49.

The microcontroller correlates the following video events to the described voltage level; 1) Sync (0.0VDC); 2) Blanking (0.4VDC); (3) Black (0.48VDC); 4) White (1.0VDC).

The DAC shown is an expedient, an alternate "built out" DAC could implement Black, Grey1, Grey2, and White, using two bits for a pixel instead of one. This does have the obvious impact on memory, increasing it by two.

Only one signal line from the microcontroller is activated at a time to produce the four discrete voltage level. &nbspBeing '51 based, port pins could be activated individually but signal transition would include a momentary "glitch" to the "white level". To ensure this does not occur, writes to the Video DAC are "byte-wide" (i.e. MOV P1,#0BFH).

The following extract from the code listing identifies the Port pins used and the byte values required to be written to Port 1 to encode the required Sync, Blank, and Video levels to the connected TV.

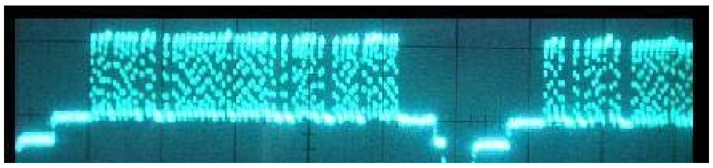
```
;       **********************
;       *** Video I/O Pins ***
;       **********************
;       *** Video DAC
;       ***                 +--- To TV
;       *** hsync ---------+---[330]---+5V
;         *** pix   --[150]--+
;       *** blank --[100]--+
;       ***                 +---[100]--- GND
;       ***
;       *** DAC signals are "exclusive"
;       *** only one active at any time.

hsync   equ     p1.4
pix     equ     p1.5
blank   equ     p1.6

;       *** Video State bytes ***

blank_val       equ     0BFH
sync_val        equ     0EFH
black_val       equ     0DFH
```

NTSC Video Timing: NTSC monochrome television display is the composite of Horizonal and Vertical synchronation pulses, and serial video patterns. These signals are of precise period and voltage level relation.

These signals are:

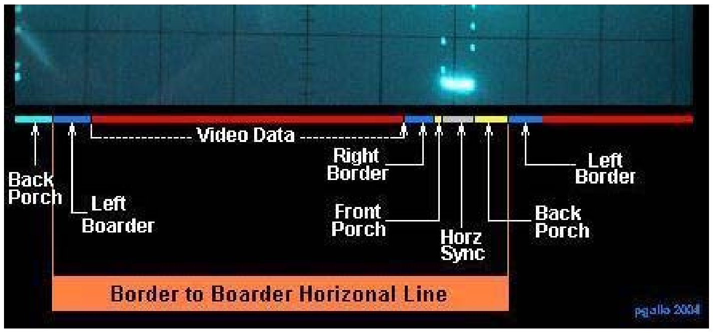
- Vertical Sync - The mechanism that causes the beam to deflect from the bottom right to the top left of the CRT,
- Horizonal Sync - The mechanism that causes the beam to deflect from right to left down one line,
- Blanking - The mechanism by where the beam is made invisible typically during retrace,
- Black - "un-displaying" video,
- White - displaying video.

This project implements a NTSC display format, alternatives such as PAL would require redefinition of specific timing.



Composition of a Horizonal Line: With a 63.5 usec period, a Horizonal Line occurs.

A Horizonal Line starts with:



1) a "Blank" (0.4VDC level) period of 1.4 usec known as the Front Porch,

2) followed by a "Sync Pulse" (0.0VDC) for 4.4 usec during which the beam re-traces to the left side of the CRT (at a location below the just displayed line).

3) The video is then "un-blanked" and a short "black time" occurs (to establish a "left border).

4) Video data is output to the display until time for the right border (black) and then the next horizonal sync occurs. &nbspAny interruption, or instability of timing, is clearly shown in display quality and format.

Code Examples: The following extract demonstrates the generation of the Horizonal Sync period of a Horizonal line.

```
mpl:

;       ***-------------***
;       *** Left Border ***
;       ***-------------***

        mov     p1,#black_val           ;3 Emit Black Video level
        mov     r2,#horz_bytes          ;2 Init # of bytes rotated to Video
```

A black left border is output and # number of Horizonal Video bytes (composed of 8 pixels is initialized.

```
;       ***--------------***
;       *** Active Video ***
;       ***--------------***

;       -- pixel cycle --

vid_rld:
        movx    a,@dptr                 ;2 Fetch a byte of Video pixels
vid_lp:
        rlc     a                       ;1 Shift one pixel value to Carry
        mov     pix,c                   ;2  Move pixel to port pin
        nop                             ;4 equiv of DJNZ
        nop
        nop
        nop
```

A byte of 8 pixels is fetched, and the Most Sig pixel is shifted to the TV interface. A delay of 4 cycles compensates for a DJNZ instruction that terminates the coming video data byte loop. Following this are 5 "inner" pixel cycles which simply rotate the next 5 pixels:

```
;       -- inner pixel cycle --

        nop                             ;1 Wait equiv of a MOVX
        nop
        rlc     a                       ;1 Shift another pixel value to Carry
        mov     pix,c                   ;2  Move pixel to port pin
        nop                             ;4 equiv of DJNZ
        nop
        nop
        nop
```

Noticably the inner pixel cycles compensate for the MOVX instruction that occured in the first pixel cycle. A final pixel cycle occurs with a test to see if the loop re-occurs (because the line is not finished) or if it's time to generate the right border.

```
;       -- pixel cycle --

        nop                             ;1 equiv of movx
        nop

        rlc     a
        mov     pix,c

        djnz    r2,vid_rld
```

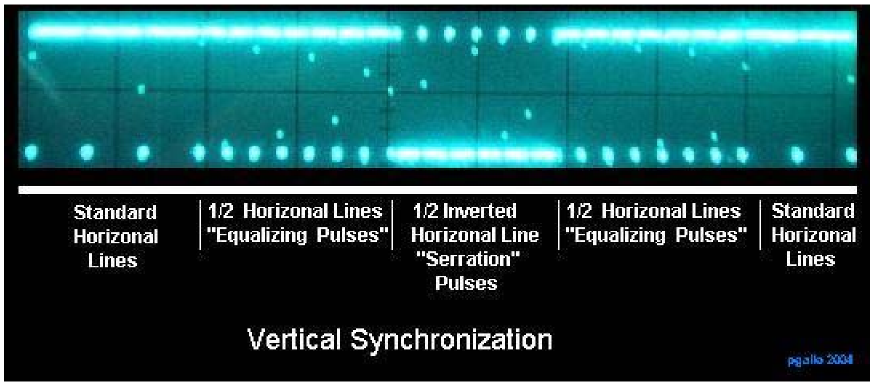
With all video data bytes exhausted, the Right Border is generated and then, the end of line occurs with the Horizonal Sync being generated.

```
;        ***--------------***
;        *** Right Border ***
;        ***--------------***

         mov     p1,#black_val
         nop
         nop

;        ***-----------------***
;        *** Horizonal Synch ***
;        ***-----------------***

         mov     p1,#blank_val          ;Output a "Blank" video state
         mov     r2,#frontporch         ; Init the Front Porch period
         djnz    r2,$                   ;  Loop for duration of Front Porch
         nop

;        *** Horizonal Sync Pulse ***

         mov     p1,#sync_val           ;Output a "Sync" video state
         mov     r2,#synch_width        ; Init the Sync Pulse period
         djnz    r2,$                   ;  Loop for duration of Sync

;        *** Back Porch ***

         mov     p1,#blank_val          ;Ouput a "Blank" video state
         mov     r2,#backporch          ; Init Back Porch period
         djnz    r2,$                   ;  Loop for duration of Back Porch

         djnz    line_counter,mpl               ;Loop thru Visible Horizonal Lines
```

Notably, the Horizonal sync is composed of three delays. &nbspInstead of sitting in "dead loops" waiting, alternative operations can occur. &nbspThe next stage of project incarnation will employ this time to test for and store any waiting serial receive data.

Generating Vertical Sync:

After 243 Horizonal lines have issued it's time to generate the Vertical Sync pattern. Importantly, this implementation is not "interlaced", that is, the 243 lines do not alternate slighly offset every other cycle as a standard Television video source would. &nbspThe code does have routines for this but in use they did not seem to generate the same quality of display as the non-interlaced version.



The Vertical Sync is composed of three sections; 1) a series of seven 1/2 lines (31.8us in length) 2) a series of six reversed direction and polarity 1/2 lines (also 31.8 us); 3) another series of seven 1/2 lines.

The code implements "black" (no video) lines above and below the Vertical Sync area. &nbspThis was done to accomodate the specific TV i am using.

Again, The Vertical Sync, and the Black lines that proceed and follow it, are periods of time spent looping. &nbspConversion of characters to font bitmaps, interpretation of incoming commands, and display scrolling can occur within the time spent doing "blacker than black" (sync-black lines) video generation.

The Vertical Sync routine calls a subroutine "half-line", seven times, a sub-routine "minus line" six times, and again calls the "half-line" routine seven times.