Introduction

At its most basic level controlling the display of a standard monitor is all about precisely controling the timing of a few different signals. Most modern monitors, including LCD monitors, display images by controlling the color of a particular pixel during each clock cycle. After each cycle the next pixel is displayed that way the active pixel sweeps across the screen at a fixed rate. With this we just need to control when to start a new line or when to go back up to the top.

In this document I will explain how to set up the timing to run a standard analog-input VGA monitor. VGA, or video graphics array, is the standard that was developed by IBM which supports a resolution of 640 x 480, for the rest of the document 640x480 will be referred to as VGA. Color is controlled by adjusting the analog inputs for the three color signals, one each for the three primary colors: red, green and blue.

Signal Timing

In order to correctly displaying images on a VGA monitor we must be able to control where the active pixel is in order to be able to send the color signals at the correct time. The maximum resolution a particular monitor is capable of displaying is based on the internal clock frequency, a faster frequency means more pixels can be displayed before the end of the row is reached. For VGA the frequency of the internal clock is 25.175Mhz.

Two control signals are used for timing. One signal, HSync, controls when the active pixel moves to the next line; the second signal, VSync, controls when the active pixel starts back to the top left corner (often called starting a new screen). In VGA the frequency for HSync, 31.47Khz, is calculated by dividing the clock by 800, because there are a total of 800 pixels per line, this number is standard and is based on the number of pixels displayed plus the front and back porches and setup time. VSync is calculated by then dividing the frequency for HSync by 525, because there are a total of 525 lines you must move through before you can display a new screen. This calculates out to make VSync 59.94 Hz, which is in the right range for allowing the screen to be flicker-free for viewing. If you would like to know exactly how the number of pixels per line and lines per screen are broken down you can look up additional information here. (http://www.epanorama.net/documents/pc/vga_timing.html)

Implementation

Implementation of a monitor driver requires using a 25.175Mhz clock, or some multiple of that frequency to be able to correctly time all the signals. The Harrisboard has a 40Mhz clock on it which can be adjusted to a 25MHz signal using the digital clock manager (DCM) on the FPGA. To generate the clock signal for the sample code an instance of a DCM module was created which allowed the clock to be multiplied in frequency by 5/8. How to use the DCM will be covered in the next section of this report. Using 25Mhz for the clock allowed for the display of full VGA. A verilog module using stepdown counters is used to generate *HSync* and *VSync*. Because it is not always the case that data sent to the monitor will be at the correct timing to be displayed, there is a variable *DataValid* that is high only when the active pixel is one of the 640 displayed in each row. Figure 1 shows the timing that results from implementation based on a 25.175Mhz clock, because a 25.00Mhz clock is implemented there is an error in all the signals but it is small enough that it does not effect the communication between the FPGA and the monitor.

	25.175 MHz clock (VGA Standard)
[clock cycles] time	
Hsync period	[800] 31.778 us
H front porch	[8] 317.775 ns
Hsync pulse length	[96] 3.813 us
H back porch	[40] 1.589 us
H Border	[8] 317.775 ns
H active video	[640] 25.422 us
[scans] time	
Vsync period	[525] 16.683 ms
V front porch	[2] 63.555 us
Vsync pulse length	[2] 63.555 us
V back porch	[25] 794.439 us
V Border	[8] 254.220 us
V active video	[480] 15.253 ms
Vsync, Hsync polarity	-,-
H frequency	31.47 KHz
V frequency	59.94 Hz

Figure 1: Timing Calculations

The sample code attached in the index draws a white rectangle on the black background. The way this is implemented is one verilog module (genSyncFullVga.m) generates the signals *HSync*, *VSync*, and *DataValid*, then another module (genSignalswitch.m) uses the clock and *DataValid* signals to generate a horizontal and vertical coordinate which is used with conditional statements to turn the output *Signal* on only when the active pixel is within the boundary of the rectangle desired.

DCM

The Spartan 3 FPGA contains 4 units called Digital Clock Managers that allow you to easily multiply the input clock for the FPGA by a ratio of integers. This is very useful if you would like to run the FPGA at a faster speed than the onboard clock or if you would like to use a specific frequency not easily obtainable by a stepdown counter.

The digital clock manager of the Spartan 3 FPGA is used by creating a new source file in ISE 6.3i and then choosing to create a new IP (CoreGen & Architecture Wizard) file. In the next screen you will chose the clocking section and the Single DCM option. By clicking next and then finish you will create a *.xaw file where the star is the name of your source. The DCM options screen will pop up automatically and you can choose the options you would like. For this implementation you will want to make it the same as Figure 2. Notice that the CLKDV and the CLKFX are check, those are the values that you will specify to divide and multiply the input clock frequency by, respectively. Thus when you click next, you will choose to enter 8 for the CLKDV value and 5 for the CLKFX value. This takes the CLKIN value of 40Mhz and outputs a 25Mhz signal on CLK0. When this is created you can use your *.xaw file by creating an instance of it. When you click on the xaw file you will see an option to view the instance, this will show you what variables are passed to the module. By creating this instance in your toplevel module you can easily use the CLK0 value from the DCM in the rest of your program.

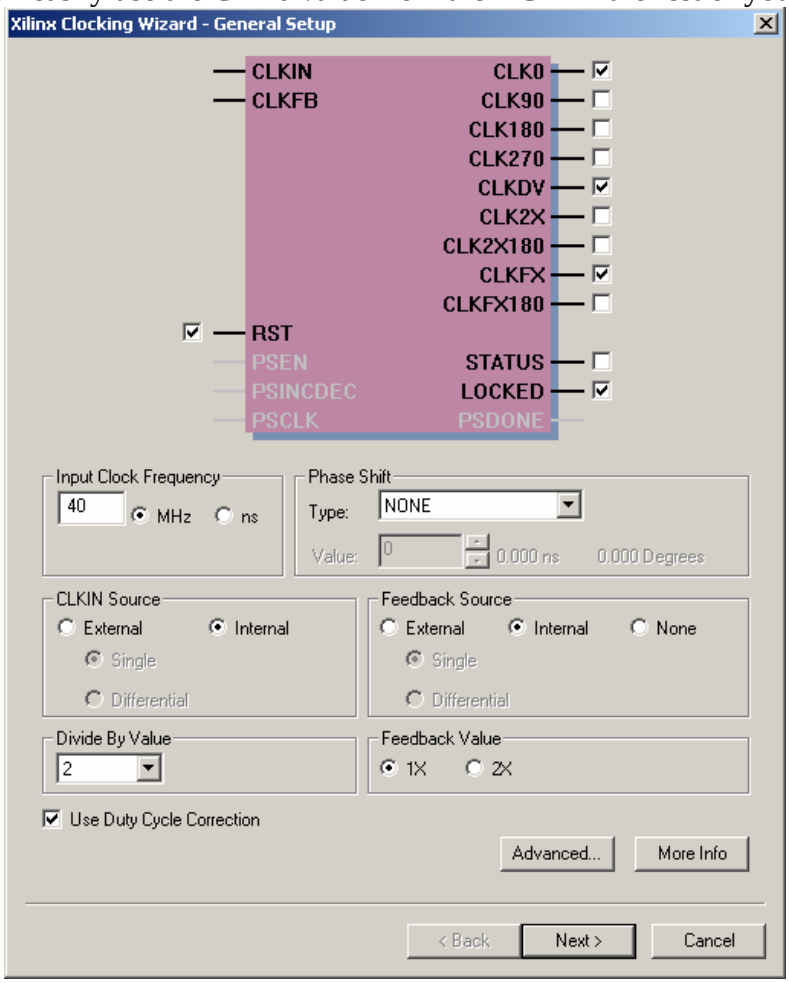


Figure 2 : DCM options

Connection

A standard VGA monitor cable contains 15 pins. The configuration for these 15 pins is shown in schematic 1. The orientation for the pins is based on the male connector, a female connector would be mirrored. To connect the monitor a male monitor cable was split and the 15 different wires were attached to a protoboard, then the monitor can be connected to the male cable.

Additional Information

The following sections contain information on where to find more detailed specifications as well as parts that can be used for implementation of a monitor driver. The 25.175 MHz oscillator can be used if it is not desirable to use the onboard clock with the DCM. The schematic and the sample code is there to give an example of a working model from which future work can be based upon.

Specifications

Xilinx Spartan XCS10 Data Sheet
http://www3.hmc.edu/~harris/class/e155/xilinx.pdf

VGA Timing Information
http://www.epanorama.net/documents/pc/vga_timing.html

Supplier

Part	Vendor	Part #	Price
Epson 25.175Mhz Oscillator	Digi-Key	SE1215-ND	$2.94

www.digikey.com

*** stocked in stockroom

Additional Resources

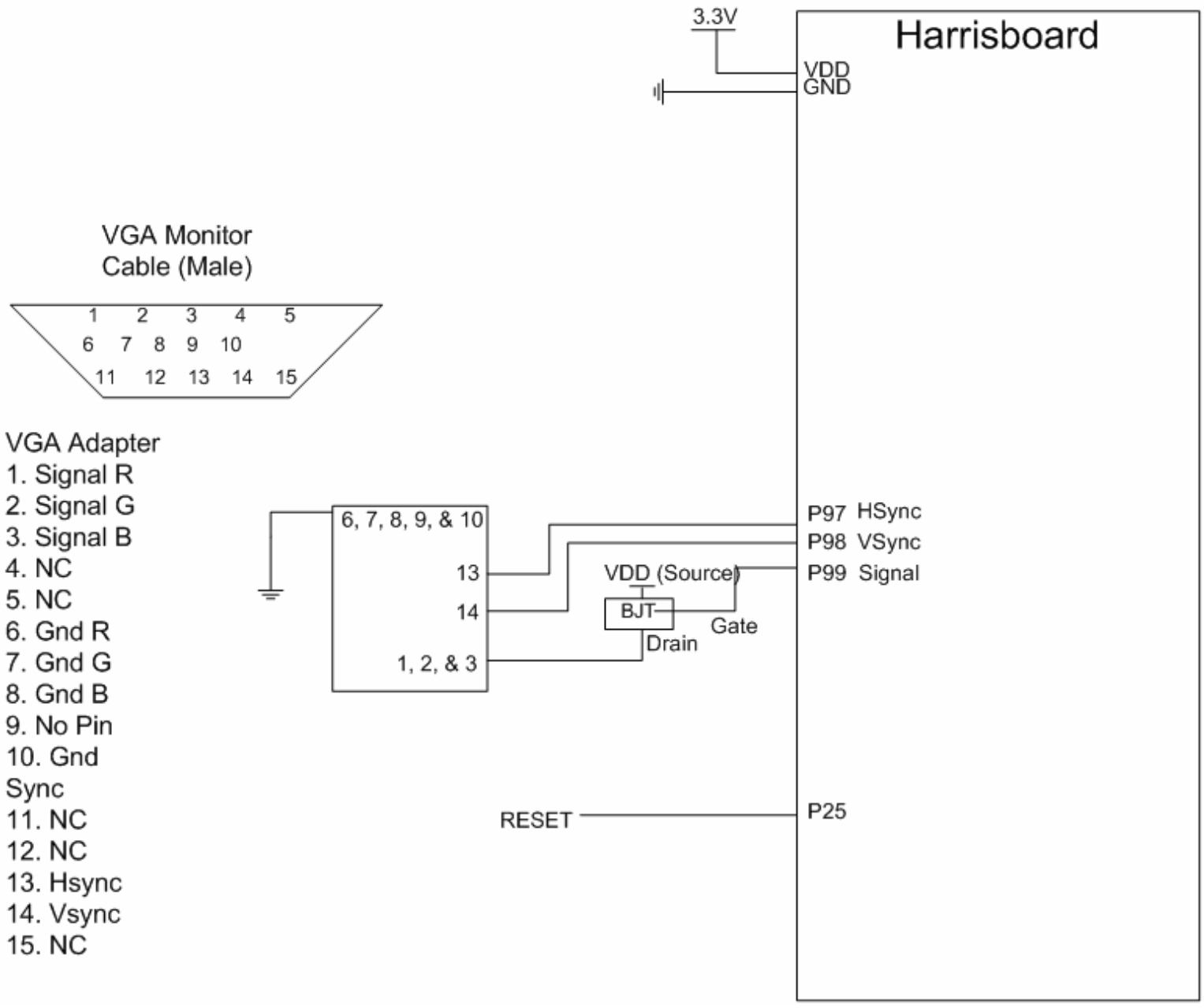
http://www.stanford.edu/class/ee108a/documentation/vga.pdf

Much of the code and information used is from:
http://www3.hmc.edu/~harris/class/e155/projects99/vgamonitordriver.pdf

Using DCM in Spartan 3 FPGA appnote
http://www.xilinx.com/bvdocs/appnotes/xapp462.pdf

Schematics

Schematic 1: Microprocessors Board Pin connections



Sample Code

Listing 1: Toplevel.m

```
//4-11-05
//Daniel Rinzler
//This module controls the inputs and outputs as well as calls the //instances
for the modules that create a VGA monitor driver
module TopLevel(clk,sclk, HSync, VSync, Signal, reset,temp,temp2,temp3);

input         clk;          // 40Mhz
input         reset;
output        sclk;  //25Mhz clock after DCM
output        HSync;
output        VSync;
output Signal;
//Signal is 1 output that is tied to all three the RGB pins on the //VGA cable
to produce white whenever Signal is turned on for a pixel

wire          clkdv,clkfx, clklock;
wire          DataValid;

//Use DCM to create 25Mhz signal
DCMclk dcm1(clk, reset,clkdv,clkfx,sclk,clklock);

// generate monitor timing signals
GenSyncsFullVga GenSyncs1(sclk, HSync, VSync, reset, DataValid);

// generate Signal to monitor
GenSignalSwitch GenSignal1(V, DataValid, Signal, sclk);

endmodule
```

Listing 2: GenSyncsFullVga.m

```
//2-15-05
//Originally written by Michael Cope & Philip Johnson 1999
//modified by Dan Chan, Nate Pinckney & Dan Rinzler Spring 2005
//This module takes the 25Mhz clock and steps it down to turn on
//HSync and VSync at the correct frequencies. It also determines when
//it is possible to send data for each pixel.
module GenSyncsFullVga(clk,HSync,VSync,reset,DataValid);
input         clk;
input         reset;
output        HSync;
output        VSync;
output        DataValid;
//High when according to HSync and VSync data is ready to flow

// 25 Mhz clk period = 40 ns

//Hsync = 31470Hz  Vsync = 59.94Hz
reg    [9:0]  slowdownforHsync;
reg    [9:0]  slowdownforVsync;
reg           HSync;
reg           HData; // High when HSync data is ready to flow
reg           VData; // High when VSync data is ready to flow
reg           VSync;

always @ (posedge clk)
       begin
              slowdownforHsync = slowdownforHsync + 1;
              if((slowdownforHsync == 10'b11_0010_0000) || (reset == 1'b1))//800
                     slowdownforHsync = 0;

              if((slowdownforHsync > 10'b00_0000_1000) && (slowdownforHsync <=
10'b00_0110_1000)) // 8 104
                     HSync = 0;
              else
```

```
                  HSync = 1;

            if((slowdownforHsync >= 10'b00_1001_1000) && (slowdownforHsync <=
10'b11_0001_1000)) // 152 792
                  HData = 1;
            else
                  HData = 0;

      end
//this always block determines when VSync should be driven low, indicating the
//start of a new screen
always @ (negedge HSync)
      begin
            slowdownforVsync = slowdownforVsync + 1;
            if ((slowdownforVsync == 10'b1000001101) || (reset == 1'b1)) //525
                  slowdownforVsync = 0;

            if((slowdownforVsync > 10'b00_0000_0010) && (slowdownforVsync <=
                10'b00_0000_0100)) // 2 4
                  VSync = 0;
            else
                  VSync = 1;

            if((slowdownforVsync >= 10'b00_0010_0101) && (slowdownforVsync <=
                 10'b10_0000_0110)) // 37 518
                  VData = 1;
            else
                  VData = 0;
      end
 assign DataValid = HData && VData;
endmodule
```

Listing 3: GenSignalSwitch.m

```
//2-15-05
//Originally written by Michael Cope & Philip Johnson 1999
//modified by Dan Chan, Nate Pinckney & Dan Rinzler Spring 2005
//This module takes the VSync and DataValid signals and uses them to generate
//the algorithm desired to display a rectangle somewhere on the VGA monitor
//screen with a 640x480 resolution


module GenSignalSwitch(VSync, DataValid, Signal, clk);

input        VSync;
input        DataValid;
output       Signal;
input        clk;

reg    [9:0] col;          // Horizontal coordinate
reg  [9:0] row;           // Vertical coordinate
reg    [9:0] temp;

//This always block assigns column values from 0 to 640 as each different
//pixel is displayed for a particular row
always @ (posedge clk)
begin

      if (DataValid)
      begin
            col <= col + 1;
      end
      else
      begin
            col <= 0;
      end
end
```

```
//Like col, temp counts the number of pixels across each row, but we then use
//temp to incriment a row counter whenever temp = 'd640. This allows us to
//reference the rows by a number from 0 to 480.

always @ (posedge clk)
       begin
              if(!VSync)          //new screen
                begin
                 temp <= 0;
                 row <= 0;
                end
            else
                if (DataValid)
                 temp <= temp + 1;
                if (temp == 9'b101000000)
                 begin
                  row <= row + 1;
               temp <= 0;
                 end
       end


//We must then assign Signal the values we desire. For drawing rectangles it is
//a matter of adjusting the boundaries for the box. However, Any algorithm could
//be implemented

assign Signal = col < 9'b011101010 && col > 9'b001001110 && row > 9'b0_0111_1000
&& row < 9'b1_1110_0000 && DataValid;



endmodule
```

Listing 4: DCMclk.xaw

```
DCMclk instance_name (
    .CLKIN_IN(CLKIN_IN),
    .RST_IN(RST_IN),
    .CLKDV_OUT(CLKDV_OUT),
    .CLKFX_OUT(CLKFX_OUT),
    .CLK0_OUT(CLK0_OUT),
    .LOCKED_OUT(LOCKED_OUT)
    );
```