CS/EE 3710 – Computer Design Lab
VGA system design

Due Friday September 30, 2022

Lab Objectives

The objectives of this lab are to build and demonstrate a simple VGA controller driving a VGA display. I had a colleague who liked to say: “There are two kinds of engineers – those who think VGA is hard, and those who have built a VGAcontroller.” That is to say, it’s a little intimidating when you first see it, but it’s really a very simple protocol for sending output information to a video screen. The steps are:

1. Read and understand the VGA specification. I recommend breaking this down into a VGA timing circuit that generates the timing signals, and a bitGen circuit that defines what color each pixel should be as the information is being sent to the display.

2. Design and implement a VGA control/timing circuit and simulate that circuit using ModelSim.

3. Design and demonstrate a very simple vgaTiming/bitGen circuit that takes input from three switches on the DE1-SoC board and drives the entire screen to the 3-bit color represented by the values on those switches.

4. Design and demonstrate a slightly more interesting bitGen2 circuit that drives the display with the output from your TBird turn signal state machine. That is, in addition to lighting up the LEDs on the DE1-SoC board, you will drive the VGA display as a TBird output device.

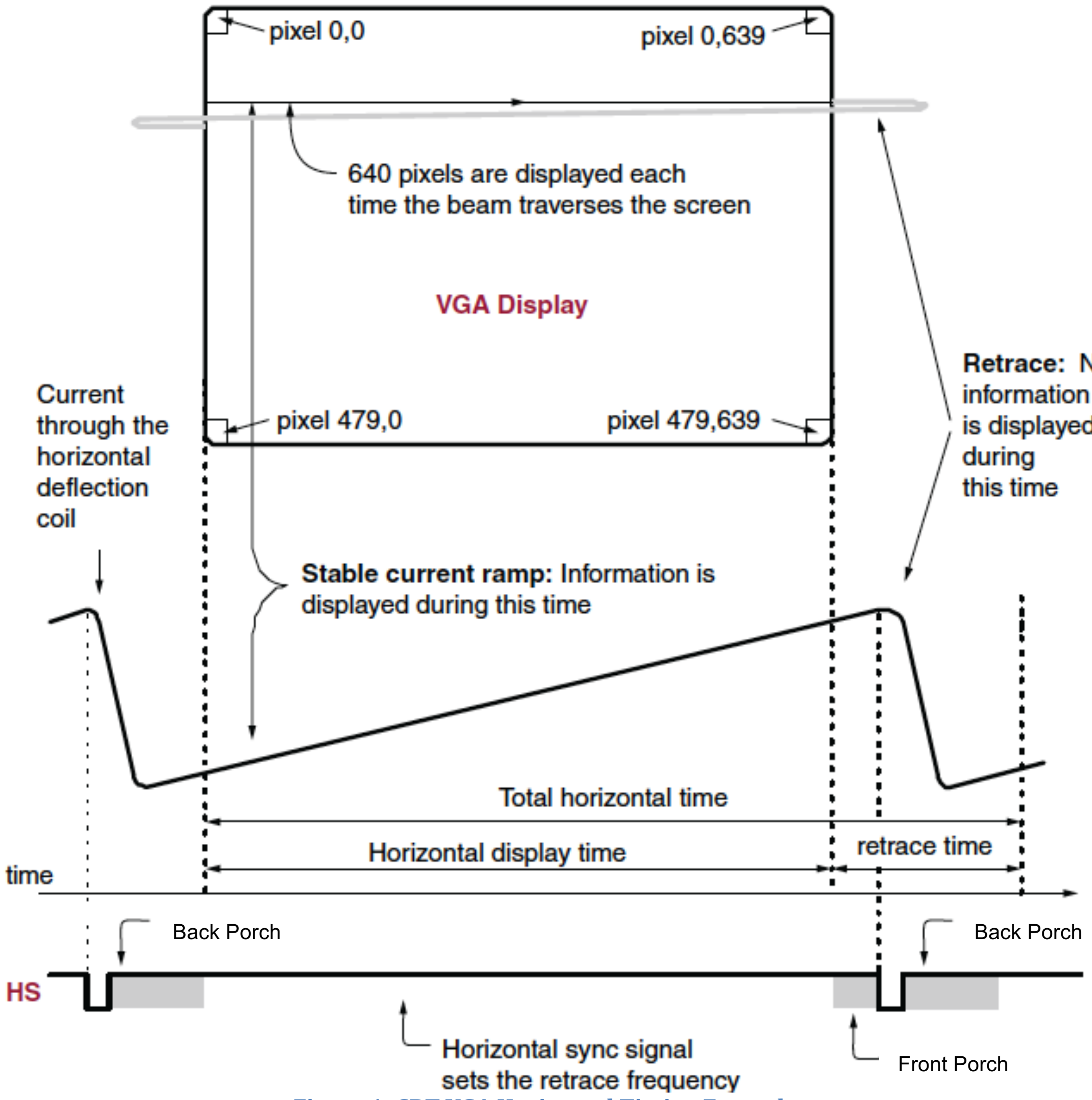
VGA Basics

The term VGA really means one of two things depending on how you use the acronym (it stands for Video Graphics Array, although almost nobody remembers that at this point). It’s either a standard 15-pin connector used to drive video devices (e.g. a VGA cable) or it’s the protocol used to drive information out on that cable (e.g. a VGA interface spec.). The interface defines how information is sent across the wires from your board to the VGA device. The cable defines which pins you use on the standard connector for those signals.

There are some slides on VGA on the Canvas page, and there is a short description of VGA (both the connector and the protocol) in the DE1-SoC board User Guide, also linked to the Canvas page. The most basic thing to know about VGA is that it is a protocol designed to be used with analog CRT (cathode ray tube) output devices. On these devices the electron beam moves across the screen from left to right as you’re looking at the screen at a fixed rate (the refresh rate defines how fast the beam moves), and also moves down the screen from top to bottom at a fixed rate. While it’s moving across and down the screen, you can modify the Red, Green, and Blue values on the VGA interface to control what color is being painted to the screen at the current location. So, painting a certain color on the screen is as easy as keeping track of where the beam is, and making sure the R, G, and B signals are at the right values when the beam is over the point on the screen where you want that color.

If you don’t do anything to stop it, the beam of an analog CRT will move to the right and bottom of the screen and get stuck there. You can force the beam to move back to the left by asserting an active-low signal called hSync (horizontal sync). You can force the beam to move back to the top of the screen by asserting an active-low signal called vSync (vertical sync). Because the beam moves at a fixed rate (defined by the monitor's refresh rate), you can keep track of where the beam is on the screen by counting clock ticks after the hSync and vSync signals.

So, the basics of the VGA control/timer circuit are just a pair of counters to count horizontal ticks and vertical ticks of the VGA clock. How many ticks are there? That depends on how fast your clock is, and how many pixels you want to paint during the time the beam moves across the screen. The basic (ancient) standard for "plain" VGA is 640 pixels on each line, and 480 lines down the screen. This is "640x480" mode. Figure 1 shows a 640x480 screen, and the horizontal sync (hSync) timing required to make it work. After the hSync pulse, you must wait for a certain number of ticks before painting pixels to the screen. This gives the beam time to get back to the left and start moving forward again. This time is called the "back porch" because it's in back of the hSync timing pulse. Then you count 640 pixels as the beam moves. After the 640th pixel, you wait for some amount of time (this is the "front porch" because it's in front of hSync), then assert the hSync signal (asserted low) for a certain amount of time.


pixel 0,0
pixel 0,639
640 pixels are displayed each time the beam traverses the screen
VGA Display
Retrace: No information is displayed during this time
Current through the horizontal deflection coil
pixel 479,0
pixel 479,639
Stable current ramp: Information is displayed during this time
Total horizontal time
Horizontal display time
retrace time
time
Back Porch
Back Porch
HS
Horizontal sync signal sets the retrace frequency
Front Porch
Figure 1: CRT VGA Horizontal Timing Example

The timing for all this depends on the monitor refresh rate. For a monitor with 60Hz refresh in 640x480 mode and a 25MHz pixel clock, you can use the timings in Figure 2. The timings in this figure define the display time (the time when the pixel is one of the 640 visible pixels in a line), pulse width (hSync or vSync), and the front porch and back porch timings. These times (in μs or ms) can also be measured in terms of the number of ticks of the 25MHz pixel clock, or in terms of the number of horizontal lines. That is, the vertical timing can be measured in terms of how many hSync pulses have been seen. The bottom line is that both the horizontal and vertical timing for the vgaTiming are just counters. You may have to enable things or reset things or change things when the counters get to a certain value, but basically they're just counters.

One subtle timing consideration – the timings in Figure 2 describe the pulse width for hSync and for vSync. Those are guaranteed to work. But, it turns out that the analog CRT only looks at the falling edge of the Sync signals to trigger the beam to move. So, in theory you could change the pulse widths and still be fine. But, no reason to mess with established timing parameters.

Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
T_S	Sync pulse time	16.7 ms	416,800	521	32 μs	800
T_{DISP}	Display time	15.36 ms	384,000	480	25.6 μs	640
T_{PW}	Pulse width	64 μs	1,600	2	3.84 μs	96
T_{FP}	Front porch	320 μs	8,000	10	640 ns	16
T_{BP}	Back porch	928 μs	23,200	29	1.92 μs	48

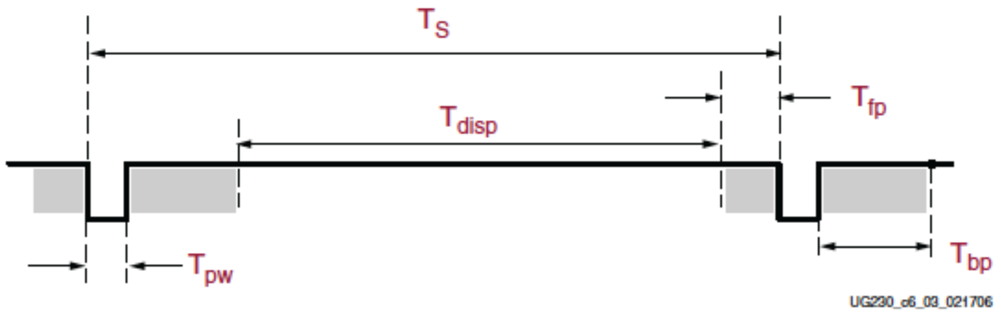


Figure 2: VGA Control Timing at 60Hz refresh and 25MHz pixel clock

I recommend designing a circuit that has two counters: one to keep track of horizontal location of the beam and one that keeps track of vertical location. The horizontal counter should count up in 25MHz ticks, and the vertical counter should count once for each full horizontal line.

As an aside – you might ask whether this is still relevant in a world where an analog CRT is harder and harder to find. You might note, though, that virtually all LCD monitors have VGA connectors on them. That means that even though they work very differently inside, modern monitors all have a VGA interface and will accept, and sync to, VGA standard signals.

Note that if you're going to use a purely synchronous design method (which I highly recommend!), you should NOT expect to have a 25MHz clock at your disposal. The system clock on the DE1-SoC board is 50MHz. You could divide that by two and use that output as the 25MHz clock, but that would NOT be purely synchronous because you would be using a derived signal as your clock. Instead you should generate an enable signal that is asserted every two 50MHz clocks, and use that enable signal in your horizontal pixel counter. For example, the horizontal pixel counter might look like:

```
// An example of a counter that only counts when En is high.
// This particular En is assumed to be high for a single clock
// cycle when it's time to count
always@(negedge clr, posedge clk50MHz)
       if (clr == 0) count = 0;          // clear count if clr is asserted
       else if (En) count = count + 1;   // If En is high, count
              // otherwise, hold previous value (default)
```

Of course, there are potentially lots of details missing here (like when to wrap around back to 0, for example), but this is an example of how you might count at 25MHz if there was a 25MHz En signal available.

For your vertical counter, I recommend generating an enable signal from your horizontal counter once per line and letting your vertical counter count up by one only when it's enabled.

The main thing your VGA control circuit should do is generate hSync and vSync at the appropriate times based on the VGA timing. However, if you're going to know where the beam is on the screen so that your bitGen signal knows what color to put there, you also need to know which pixel and which line the beam is currently over. So, you also need to export the hCount and vCount signals so you can figure out which pixel is being painted. Finally, it's a good idea to generate a signal from your VGA control circuit that is asserted whenever the beam is in a "pixel bright" area of the screen. That is, you are NOT in a front porch, back porch, or sync section of the timing.

I recommend that your VGA control circuit have the following interface:

```
module vgaControl (
  input clk50MHz, clr,            // system clock and clear
  output hSync, vSync,            // sync signals to the VGA interface
  output bright,                  // asserted when the pixel is bright
  output reg [X:0] hCount, vCount // horiz and vertical count (you pick the X!)
                                  // these tell you where you are on the screen
);
```

The body of this module is, of course, up to you. It should generate the outputs based on the 640x480 60Hz refresh timing in Figure 2. You can validate the timing by simulating the vgaTiming circuit in the Quartus/ModelSim simulator.

Some things to consider when you define this module:

- Remember that hSync and vSync are asserted low and should be high when not active.
- hCount and vCount are used by your bitGen circuit so that it knows where you are on the screen. It's handy if the value of hCount is the value of the pixel. That is, if you start counting at 0 when you start the Horizontal Display Time from Figure 1, then the number on the counter will be the pixel number in the line (between 0 and 639).
- In the same way, if you start counting vertical lines in the Vertical Display Time portion of the vertical timing, then the value on the counter will be the line number (between 0 and 479)
- The bright signal can be either asserted high or low – it's used by your bitGen circuit to tell that you're in a section of the screen where you want to turn on the pixel. If you're "not bright" you should force the RGB output to all zeros so that you're not driving the screen.

VGA Colors

The VGA timing information in the previous section controls the motion of the electron beam in the CRT. But what controls the color of the dot as the beam is moving along the CRT? The standard VGA connector tells part of that story.

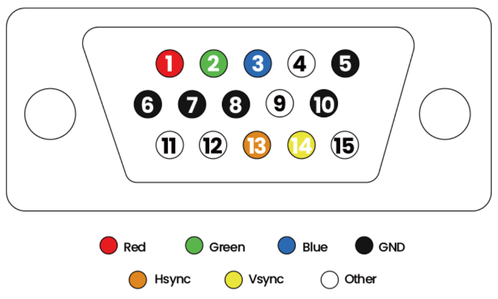


Figure 3: VGA connector showing pin assignments (looking in to the cable, not the port on your board).

You can see the pin assignments on the standard VGA connector/cable. The hSync and vSync timing signals go to pins 13 and 14 respectively. Pins 5, 6, 7, 8, and 10 are ground. Pins 4, 9, 11, and 12 are generally not connected to anything (there are some historical uses for those pins that are generally not used any more. Pines 1, 2, and 3 are the R, G, and B pins. The hSync and vSync signals tell the beam *where* to go. The R, G, and B signals tell the three color beams *what* color to make. These are analog signals with a range of 0 – 0.7v. 0 is that color being all the way off, and 0.7v is making that color all the way on. If you want lots of colors, you need lots of control over the analog values of those signals as the beams move across the screen.

The earliest color displays had a single bit of information for those signals – they were either on (0.7v) or off (0v). This means you had 8 possible colors because with three bits of color there are 8 color combinations of R, G, and B. If you want a few more colors you can build a simple resister divider circuit to take, say, three bits of information for each color and divide the 0.7v into 8 values. That gives you 512 colors (2^9 colors). Actually, many early displays had 8 bits of color for 256 colors allowing the color to fit in a byte. In this version there are 3 bits for R, 3 bits for G, and only 2 bits for B because the human eye is less sensitive to blue so you can get away with fewer intensities in the B color space.

VGA_RED	VGA_GREEN	VGA_BLUE	Resulting Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

Figure 4: Colors from a single-bit per color channel.

Modern machines generally have 24bits of color with 8bits of resolution for each color channel. That's what the DE1-SoC board has. Instead of a resistor-ladder network the DE1-SoC board has a three-channel 8-bit DAC for producing R, G, and B. Actually, it's a three-channel 10-bit DAC so in theory it could do 30-bit color, but only the higher-order 8-bits of each channel are connected to the FPGA, and again, 24 bits / pixel is a good standard. You can see how all those bits are connected to the FPGA in terms of actual pin numbers in the DE1-SoC User Manual.

Some things to note about the VGA DAC:

- There are 8 bits for each color channel, but for this lab you may just want a simpler version with just one bit per color (8 total colors). In that case you need to take that single bit and expand it to 8. A Verilog concatenation is a good way to do this.

- The VGA DAC has a VGA_CLK input. This is the VGA pixel clock – it says how frequently to update the colors. For 640x480 VGA this is the 25MHz clock that your timing circuit uses. In this case it's OK to put that derived clock directly into the VGA_CLK.

- The VGA_BLANK_N signal going into the VGA DAC is an active-low signal that says when to blank the RGB color channels. In other words, when you're NOT in the bright part of the screen. If you're in the front or back porch timing areas, you drop this signal to make sure that the DAC produces 0's on all the color channels. Note that this active-low blanking signal is the same as an active-high "bright" signal…

- The VGA_SYNC_N signal is an active low signal that determines whether the green color channel has a synchronization signal overlayed on top of it. This is used in some systems so that the R, G, and B are the only signals necessary, and the synchronization signal is overlayed on the G channel. We aren't using that, so you can tie this signal low.

- The outputs from your final VGA circuit need to drive the 8-bit VGA_R, VGA_G, VGA_B channels, VGA_CLK, VGA_BLANK_N, and you also need to connect your hSync and vSync to the VGA_HS and VGA_VS signals. See the DE1-SoC User Manual for pin assignments.

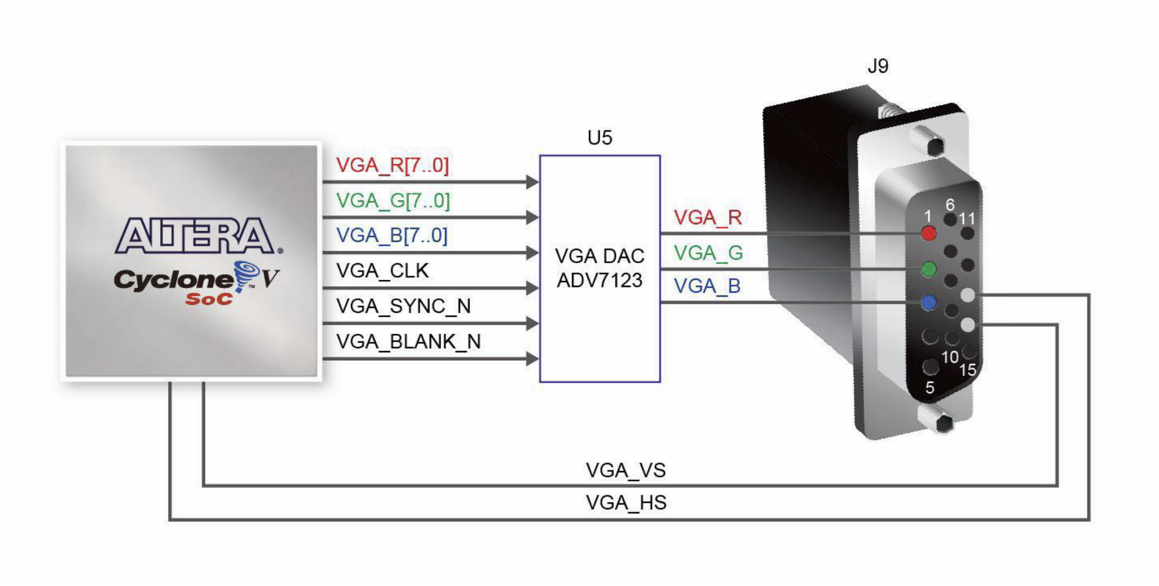


Figure 5: Connection between the FPGA on the De1-SoC board and the VGA DAC

The bitGen Circuit

This is the circuit that takes the hCount, vCount, and bright signals, and decides for each pixel what color should be on the screen. At a minimum a bitGen circuit would have the following interface:

```
module bitGen_minimum (
   input bright,           // asserted if the pixel is in the bright region
   input [X:0] hCount, vCount, // the horizontal and vertical counts
   output [2:0] rgb            // the RGB value of the (hCount,vCount) pixel
// there may be other inputs to help define what the pixel color is...
);
```

How does your bitGen circuit know what the color should be? There are lots of ways, but the three general techniques are:

1. **Bitmapped graphics:** in this technique you have a “frame buffer” that holds the RGB values for every pixel on the screen. You can address this buffer using the hCount and vCount signals to retrieve the color at each of the 640x480 locations on the screen. Generally a frame buffer is dual-ported so that, for example, a CPU can be putting data into the frame buffer and the VGA bitGen circuit can be reading it out at the same time. This is the most memory-intensive technique. 640x480 is 307,200 pixel locations on the screen. If you pack two three-bit rgb pixel values in each byte, that’s still 153,600 bytes of storage to hold the whole 640x480 screen (150kBytes).

2. **Character/glyph graphics:** Instead of storing each of the pixels in the frame buffer, you can break the screen into chunks and store a pointer to a glyph in each screen chunk. For example, if you break the screen into 8x8 pixel chunks, then there are 80x60 chunk locations on the screen (this is 4800 locations total). If you have 256 possible 8x8 glyphs stored somewhere else, then you only need 4800 bytes (4.69kBytes) to store the screen (one 8-bit glyph pointer for each screen chunk), but at each location on the screen you are limited to one of the 256 8x8 glyphs. The glyphs themselves need to be in a separate memory that consists of 2k locations of 8-bits each. So, the total storage requirement is

6848 bytes (6.69kBytes). A lot less than 150kBytes, but less flexible. It’s common to have alphanumeric characters and some graphic glyphs in the glyph memory. You can use the hCount and vCount values not only to tell which chunk you’re in, but also which pixel of the glyph you’re currently in. Hints – If the glyphs are 8 pixels across, which bits of the hCount counter define the pixel within the glyph? How many bits does it take to count to 8 pixels? Which bits of hCount define which glyph you’re in? You only want those bits to change once every 8 ticks of the pixel clock.

3. **Direct graphics:** Instead of storing information about what’s at each screen location, you can design a circuit that checks the current hCount and vCount and draws directly on the screen when you’re in the right spot. For example, if you want to draw a white box at a particular location on a red screen, your bitGen circuit might have the following Verilog code:

```
parameter ON  = 8’b11111111; // 8-bit color channel as bright as possible
parameter OFF = 8’b00000000; // 8-bit color channel is off

// paint a white box on a red background
always@(*)
   if (~bright)
          begin  // force black if not bright
          R = OFF G = OFF; B = OFF;
          end
   // check to see if you’re in the box
   else if (((hCount >= 100) && (hCount <= 300)) &&
            ((vCount >= 150) && (vCount <= 350)))
        begin
          R = ON; G = ON; B = ON;   // All three ON makes white
        end
   else begin
          R = ON; G = OFF; B = OFF; // pure red is the background color
        end
```

Assignments Specifics

The previous text is to help you understand the VGA spec. The specifics of this assignment are as follows.

1. Design and simulate a vgaTiming module to generate hSync, vSync, bright, hCount, and vCount signals. The module should take in the 50MHz system clock that you've used for previous labs. The VGA timing should be based on the 60Hz refresh 640x480 VGA timing described in this handout. Simulate this module with ModelSim. Clearly this will take a lot of simulation to produce any meaningful output – you don't need to simulate more than is necessary to show that things are basically working.

2. Design a bitGen circuit that takes input from vgaTiming, and from three switches on the DE1-SoC board. Based on the value of these switches (i.e. Figure 4), paint the entire 640x480 screen in this color. Changing the switches should change the color of the screen. Demonstrate this circuit to the instructor or TA.

3. Design a bitGen2 circuit that takes input from vgaTiming, and also from your TBird FSM from your previous assignment. Use a bitGen display technique of your own choosing, and display the L and R turn signal lights on the VGA screen. That is, something on the screen should light up when the LEDs on the board light up. It should be a VGA version of your TBird circuit. How this looks is up to you. Demonstrate this circuit to the instructor or TA.

4. Turn in all your commented Verilog code (vgaTiming, bitGen, bitGen2, and any additional Verilog you generated), any testbench code you used for testing, and any schematics that you might have used to combine the vgaTiming and the various bitGen modules. If you modified anything in your TBird FSM, turn in those files too.

Appendix: Other VGA standards

The standard we're using in this lab is the original 640x480 60Hz timing standard. There are many more standards in use now that have much higher resolutions and refresh rates (and are therefore a little trickier to make work. At the expense of more complex bitGen circuits, and perhaps much more memory for bitmapped or glyph-based graphics, you could implement these in your project. Be aware that the "easy" ones have a pixel clock of 50MHz or below. That's the basic clock available on our boards. So, it may be considerably trickier to go beyond SVGA 75Hz refresh…

VGA mode		***Horizontal Timing Spec***				
Configuration	***Resolution(HxV)***	***a(us)***	***b(us)***	***c(us)***	***d(us)***	***Pixel clock(MHz)***
VGA(60Hz)	**640x480**	**3.8**	**1.9**	**25.4**	**0.6**	**25**
VGA(85Hz)	**640x480**	**1.6**	**2.2**	**17.8**	**1.6**	**36**
SVGA(60Hz)	**800x600**	**3.2**	**2.2**	**20**	**1**	**40**
SVGA(75Hz)	**800x600**	**1.6**	**3.2**	**16.2**	**0.3**	**49**
SVGA(85Hz)	**800x600**	**1.1**	**2.7**	**14.2**	**0.6**	**56**
XGA(60Hz)	**1024x768**	**2.1**	**2.5**	**15.8**	**0.4**	**65**
XGA(70Hz)	**1024x768**	**1.8**	**1.9**	**13.7**	**0.3**	**75**
XGA(85Hz)	**1024x768**	**1.0**	**2.2**	**10.8**	**0.5**	**95**
1280x1024(60Hz)	**1280x1024**	**1.0**	**2.3**	**11.9**	**0.4**	**108**

Figure 6: Horizontal timing for additional VGA specs

VGA mode		***Vertical Timing Spec***				
Configuration	***Resolution(HxV)***	***a(lines)***	***b(lines)***	***c(lines)***	***d(lines)***	***Pixel clock(MHz)***
VGA(60Hz)	**640x480**	**2**	**33**	**480**	**10**	**25**
VGA(85Hz)	**640x480**	**3**	**25**	**480**	**1**	**36**
SVGA(60Hz)	**800x600**	**4**	**23**	**600**	**1**	**40**
SVGA(75Hz)	**800x600**	**3**	**21**	**600**	**1**	**49**
SVGA(85Hz)	**800x600**	**3**	**27**	**600**	**1**	**56**
XGA(60Hz)	**1024x768**	**6**	**29**	**768**	**3**	**65**
XGA(70Hz)	**1024x768**	**6**	**29**	**768**	**3**	**75**
XGA(85Hz)	**1024x768**	**3**	**36**	**768**	**1**	**95**
1280x1024(60Hz)	**1280x1024**	**3**	**38**	**1024**	**1**	**108**

Figure 7: Vertical timing for additional VGA specs

Pin Assignments

By now you should know that you'll need to look up pin assignment information in the DE1-SoC User Manual, but just for convenience, here's the connection info for the VGA DAC and VGA connector, the slide switches, and the clock.

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
VGA_R[0]	PIN_A13	VGA Red[0]	3.3V
VGA_R[1]	PIN_C13	VGA Red[1]	3.3V
VGA_R[2]	PIN_E13	VGA Red[2]	3.3V
VGA_R[3]	PIN_B12	VGA Red[3]	3.3V
VGA_R[4]	PIN_C12	VGA Red[4]	3.3V
VGA_R[5]	PIN_D12	VGA Red[5]	3.3V
VGA_R[6]	PIN_E12	VGA Red[6]	3.3V
VGA_R[7]	PIN_F13	VGA Red[7]	3.3V
VGA_G[0]	PIN_J9	VGA Green[0]	3.3V
VGA_G[1]	PIN_J10	VGA Green[1]	3.3V
VGA_G[2]	PIN_H12	VGA Green[2]	3.3V
VGA_G[3]	PIN_G10	VGA Green[3]	3.3V
VGA_G[4]	PIN_G11	VGA Green[4]	3.3V
VGA_G[5]	PIN_G12	VGA Green[5]	3.3V
VGA_G[6]	PIN_F11	VGA Green[6]	3.3V
VGA_G[7]	PIN_E11	VGA Green[7]	3.3V
VGA_B[0]	PIN_B13	VGA Blue[0]	3.3V
VGA_B[1]	PIN_G13	VGA Blue[1]	3.3V
VGA_B[2]	PIN_H13	VGA Blue[2]	3.3V
VGA_B[3]	PIN_F14	VGA Blue[3]	3.3V
VGA_B[4]	PIN_H14	VGA Blue[4]	3.3V
VGA_B[5]	PIN_F15	VGA Blue[5]	3.3V
VGA_B[6]	PIN_G15	VGA Blue[6]	3.3V
VGA_B[7]	PIN_J14	VGA Blue[7]	3.3V
VGA_CLK	PIN_A11	VGA Clock	3.3V
VGA_BLANK_N	PIN_F10	VGA BLANK	3.3V
VGA_HS	PIN_B11	VGA H_SYNC	3.3V
VGA_VS	PIN_D11	VGA V_SYNC	3.3V
VGA_SYNC_N	PIN_C10	VGA SYNC	3.3V

Figure 8: VGA DAC and VGA connector pin assignment information (see Figure 5)

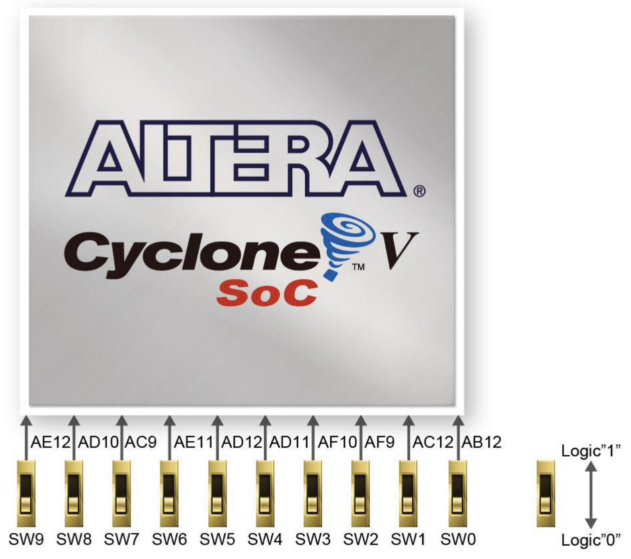


Figure 9: Pin assignments for slide switches on the DE1-SoC

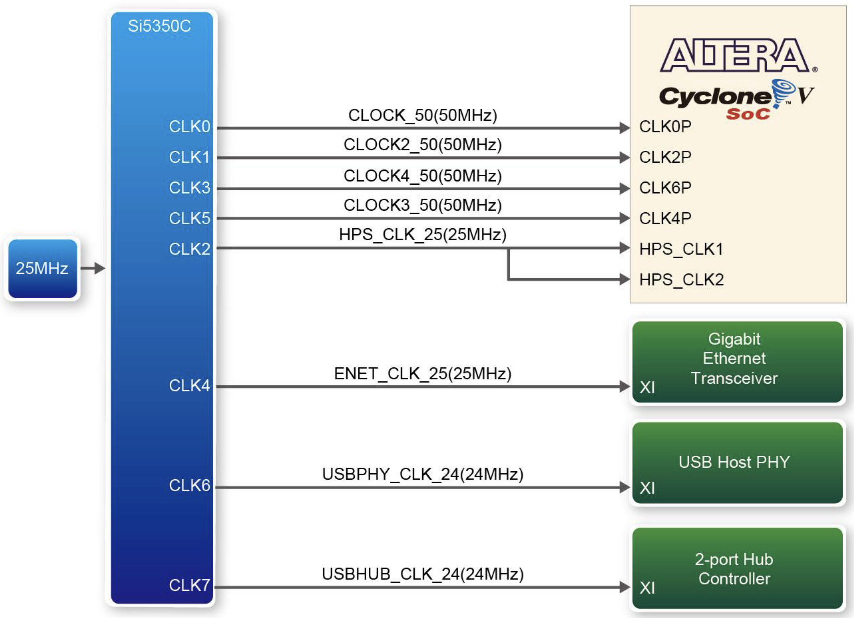


Figure 10: Pin assignments for clock inputs to the FPGA chip. Use only the CLOCKx_50 signals.