EENG2131 - Lab 8

VGA Interface for FPGA

Antonio Veguilla Hernandez

Student ID: P000166790

A picture containing connector

Description automatically generatedFor this lab, we’ll be exploring the analog signals involved with driving an old-school Video Graphics Adapter (VGA) display from our FPGA dev boards.

# Part 1 – VGA Signals

The VGA interface was designed way back in the old days, when displays were Cathode Ray Tubes, shooting an electron beam from the back towards the back-side of the screen, using magnetic fields to steer the electron beam to draw each and every pixel on the display. Even though modern displays are almost all LCDs, the same interface is used (driving a digitizer instead of magnetic coils). Please read this background material to start to understand the five signals that we need to generate to drive the VGA interface:

* <https://digilent.com/reference/programmable-logic/basys-3/reference-manual#vga_port>
* **Before you continue, explain to the instructor how the five VGA signals interact to drive the display.**

# Part 2 – Generate 25 MHz pixel clock

Create a **PixelClock** module that has the following interface:  
Input: Our normal 100 MHz clk signal.  
Output: a 25 MHz clock signal (basically divide the input clock by 4).

First, verify that we have a 25 MHz clock signal in the simulator. We should also verify the two clocks on the FPGA board itself, using an oscilloscope. Create an FPGA interface module that connects the **PixelClock** inputs and outputs to two of the LEDs. Then we can use the oscilloscope to observe the frequencies of the two clocks.

* **Before you continue, show the instructor your Verilog code. Explain how it works to generate the 25 MHz clock. Show how you can use the oscilloscope to observe the two clock signals.**

|  |  |
| --- | --- |
| module PixelClock(clk, clk\_25MHz);input clk;output reg clk\_25MHz = 0;reg [9:1]count = 0;always @ (posedge clk)beginif(count == 1)beginclk\_25MHz <= ~clk\_25MHz;count <= 0;endelsecount <= count + 1;endendmodulemodule testbench\_PixelClock;reg clk;wire clk\_25MHz;PixelClock dut(clk, clk\_25MHz);initial beginclk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10 | clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10clk =1;#10clk =0;#10;endendmodule |

# A screenshot of a computer Description automatically generated with medium confidence

# Part 3 – Generate Horizontal Sync and Vertical Sync signals

Let’s target a 640 x 480 display resolution. This results in the following set of timings. Each value is a count of pixels:

640x480 Timings HOR VER  
 -------------------------------  
 Active Pixels 640 480  
 Front Porch 16 10  
 Sync Width 96 2  
 Back Porch 48 33  
 Blanking Total 160 45  
 Total Pixels 800 525  
 Sync Polarity neg neg

Notice how there are horizontal and vertical “pixels” (or at least allocated pixel timings) outside of the actual 640 x 480 pixel display. These non-display areas are included to give time for the electron beam to transition to the next row / next page.

Here’s another way to look at it. The signal shape is the same for both horizontal and vertical sync. Notice the small discrepancy in the vertical sync pixel counts. We’ll have to try it both ways to see which works best with our displays.

Diagram

Description automatically generated with medium confidence

In order to create these two signals (HSYNC and VSYNC) we should use our pixel clock to drive the horizontal counter, and whenever the horizontal counter rolls-over from 799 to 0, it should trigger the vertical counter. Both counters should be 10-bit counters.

For implementation, we can have each counter start counting at 0, and then trigger changes to our three output values at these counter values. For convenience, we’ll align counter = 0 to be the start of the actual pixel display region, and handle the front-porch, sync pulse, and back-porch at the end of the counter’s range.

Horizontal: (driven by the 25 MHz clock)

* count = 0: Start of actual pixel display region (first column of this row), DE = LOW
* count = 639: End of actual pixel display region (last column of this row), DE = HIGH, start of front-porch
* count = 639 + 16: End of front-porch, start of sync pulse, HSYNC = LOW
* count = 639 + 16 + 96: End of sync pulse, HSYNC = HIGH, start of back-porch
* count = 799: End of back-porch, end of line (increment vertical counter when this rolls over)

Vertical: (increments every time the horizontal counter resets)

* count = 0: Start of actual pixel display region (first row of the display), DE = LOW
* count = 479: End of actual pixel display region (last row of the display), DE = HIGH, start of front-porch
* count = 479 + 10: End of front-porch, start of sync pulse, VSYNC = LOW
* count = 479 + 10 + 2: End of sync pulse, VSYNC = HIGH, start of back-porch
* count = 524: End of back-porch, end of display

Note that DE should only be LOW when both conditions are true:

* Hcount < 640
* Vcount < 480

Let’s create a new Verilog module called **SyncSignals** with this interface:

Input:

* pixelClk – the 25 MHz pixel clock signal
* rst – an active-low reset signal that will reset the two counters

Output:

* Hcount – the 10-bit horizontal counter output
* Vcount – the 10-bit vertical counter output
* HSYNC – the single-bit horizontal sync signal (active low)
* VSYNC – the single-bit vertical sync signal (active low)
* DE – Data Enable - the single-bit “active drawing area” signal (active low)

**Deliverables:**

* Use the simulator to verify the horizontal sync signal. Show the instructor the width of each high and low pulse.
* Use the simulator to verify the vertical sync signal. Show the instructor the width of each high and low pulse. Does it match what the table above says it should be?
* Create an FPGA interface module that connects the 25 MHz clock from the PixelClock module to the SyncSignals module, and then connect the HSYNC, VSYNC, and DE signals to LEDs. Then we can use the oscilloscope to observe the three signals and confirm they are correct.
* **Before you continue, show the instructor your Verilog code. Explain how it works to create the HSYNC, VSYNC, and DE signal. Show how you can use the oscilloscope to observe these signals.**

|  |  |
| --- | --- |
| module FPGA\_Interface(clk,led, JA, btnC, vgaRed, vgaGreen,vgaBlue, Hsync, Vsync, sw);  input clk;  input btnC;  input [15:0]sw;  output [3:0]vgaRed;  output [3:0]vgaBlue;  output [3:0]vgaGreen;  output Hsync, Vsync;  output [15:0]led;  output [7:0]JA;  PixelClock(clk, clk\_25MHz);  assign led[0] =clk;  assign led[1] =clk\_25MHz;  assign JA[1] =clk;  assign JA[2] =clk\_25MHz;  //SyncSignals(clk, btnC);  SyncSignals(clk, btnC, Hcount, Vcount, Hsync, Vsync, DE);  wire [9:0]Hcount, Vcount;  wire DE;  assign vgaRed = (DE) ? 0 : sw[3:0];  assign vgaBlue = (DE) ? 0 : sw[7:4];  assign vgaGreen = (DE) ? 0 : sw[11:8];  endmodule  /////////////////////////////////////////////////////  module SyncSignals(  input clk,  input rst,  output reg [9:0]Hcount = 0,  output reg [9:0]Vcount = 0,  output HSYNC,  output VSYNC,  output DE  );  PixelClock myclk(clk, clk\_25MHz);  wire clk\_25MHz;  assign DE = ~(Hcount < 640 && Vcount < 480);  assign HSYNC = ~(Hcount > 655 && Hcount < 752);  assign VSYNC = ~(Vcount > 488 && Vcount < 492); | always @(posedge clk\_25MHz)  begin  if(rst == 1)  begin  Hcount <= 0;  Vcount <= 0;  end  else begin  Hcount <= Hcount +1;  if(Hcount == 799) begin  Hcount <= 0;  Vcount <= Vcount + 1;  end  if(Vcount == 520) Vcount <= 0;  end  end  endmodule  module testbench\_SyncSignals();  reg clk = 0;  reg rst = 0;  SyncSignals dut1(clk, rst, Hcount, Vcount, HSYNC, VSYNC, DE);  wire HSYNC;  wire VSYNC;  wire DE;  wire [9:0]Hcount;  wire [9:0]Vcount;  always begin  clk = ~clk;  #5;  end  initial begin  rst = 1;  #10  rst = 0;  #10  rst = 1;  #10;  end  endmodule |

A screenshot of a computer

Description automatically generated with medium confidence