ECEN 3002 – Video Project, Part One

Introduction

To truly learn FPGA design, a person must spend time doing design, running the FPGA tools, simulating the design, and recognizing the proper debug techniques. The purpose of this course is not to make you an expert digital designer, but to introduce you to the key aspects of FPGA design, and how to take best advantage of what is offered.

The first goal of the video project is to create a simple VGA based video controller. You will add additional features and capabilities to the project as the semester progresses, as a means to learn more about available design and implementation options and approaches.

For Phase One, you should complete the following:

1) Create a 720p VGA display controller running at 1280 x 720, 60Hz resolution
2) Create accurate simulations of the video timing
3) Create a simple color bar output pattern that displays correctly on your monitor.
4) Use a pushbutton to switch between color and grayscale outputs.

Specific requirements for Part One are given later in this document, and are highlighted in red.

Important: The diagrams in this document show details of a video controller that displays 640 x 480 VGA video. While the diagrams show the fundamental concepts, some of the details for our 720p controller are different. These differences will be highlighted throughout this document.

Basics of VGA video

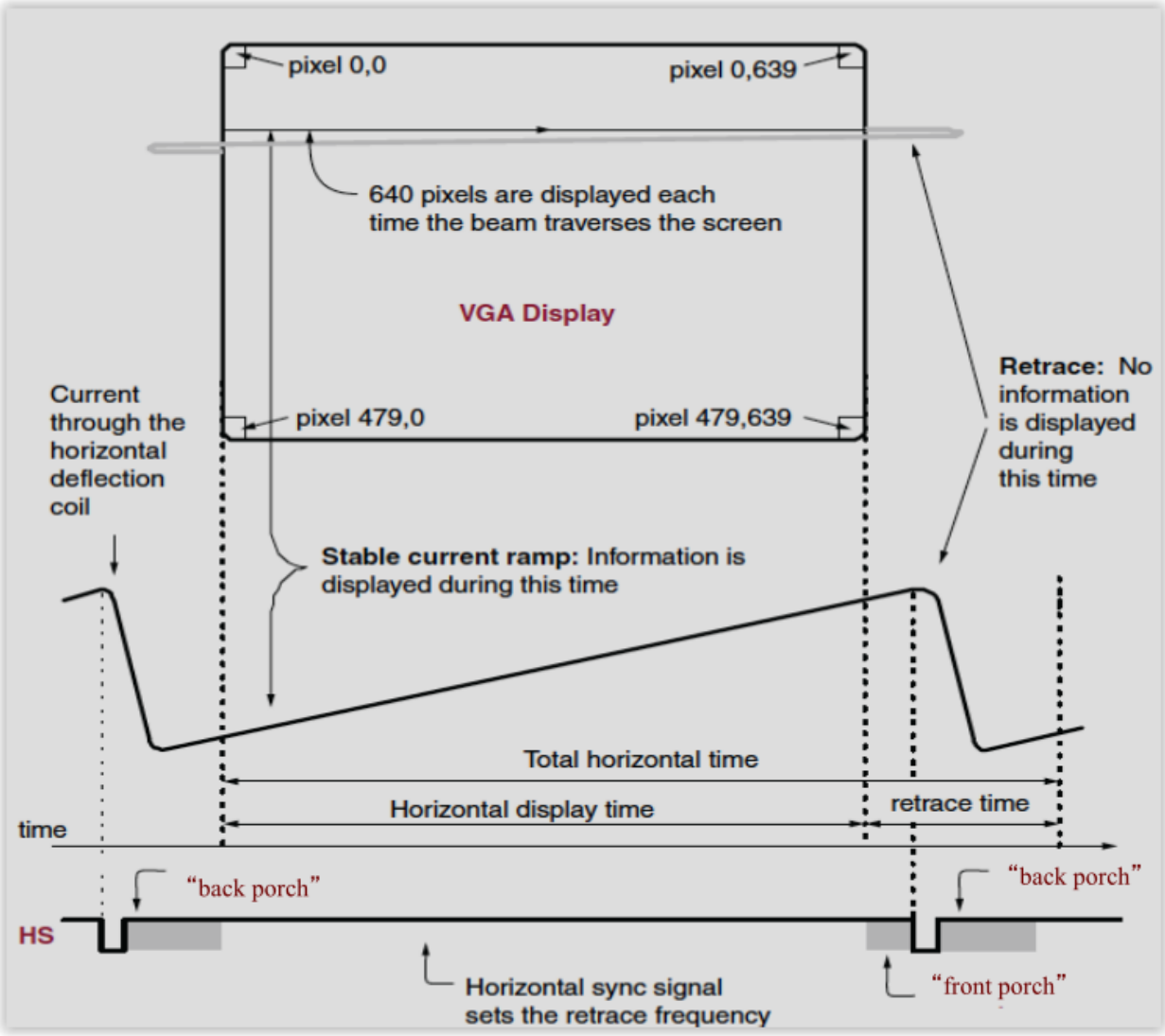
The VGA video standard was developed for use with analog monitors. An electron beam scans across the monitor screen from left to right, and also moves from top to bottom at a given rate. At points along the beam travel, red, green, or blue values are painted on the screen phosphor. By controlling when the various colors are enabled, images are formed.

As the beam moves from left to right, eventually the beam comes to the right side of the screen, and must be moved or reset back to the left side. The Horizontal Sync (Hsync) signal forces the beam back to the left side of the screen. Similarly, eventually the

beam reaches the bottom of the screen, and the Vertical Sync (Vsync) signal is used to move the beam back to the top of the screen.

A video controller generates the proper timing for the Hsync and Vsync signals. By using counters to control horizontal and vertical beam position, one can control both when active video should occur, and when video should be disabled.

The diagram below provides detail for a 640 x 480 VGA controller.



Notice that there is time between Hsync signals where there is no active video. Video is not active during the time when Hsync is asserted low, as well as during a time period before Hsync is active (called the front porch) and during a time period after Hsync deasserts (called the back porch). This implies that the video controller must generate appropriate signals to indicate when active video signals must be driven, and when active video must be turned off. Time when video is off is termed the retrace time.

The same general discussion is also true for the Vsync timing. An active low Vertical Sync pulse, along with its corresponding front and back porch intervals, signals the return of the beam to the top of the screen, and during this retrace period video must be turned off. The frequency of the Vsync signal is called the refresh rate.

Based on class experience from the previous semester, you should design your video controller so that active video starts at the first pixel of a line, and that the horizontal front porch, horizontal synch, and horizontal back porch times occur at the end of a line. Similarly, your vertical front porch, vertical synch, and vertical back porch times should occur at the end of the frame.

Video Timing Parameters

For 1280 x 720, 60Hz VGA, the critical timing values look like this:

https://projectf.io/posts/video-timings-vga-720p-1080p/

Name 1280x720p60

Standard	CTA-770.3
VIC	4
Short Name	720p
Aspect Ratio	16:9

Pixel Clock	**74.250 MHz**
TMDS Clock	742.500 MHz
Pixel Time	13.5 ns ±0.5%
Horizontal Freq.	45.000 kHz
Line Time	22.2 µs
Vertical Freq.	60.000 Hz
Frame Time	6.7 ms

Horizontal Timings

Active Pixels	**1280**
Front Porch	**110**
Sync Width	**40**
Back Porch	**220**
Blanking Tota	**370**
Total Pixels	**1650**
Sync Polarity	**pos**

Vertical Timings

Active Lines	**720**

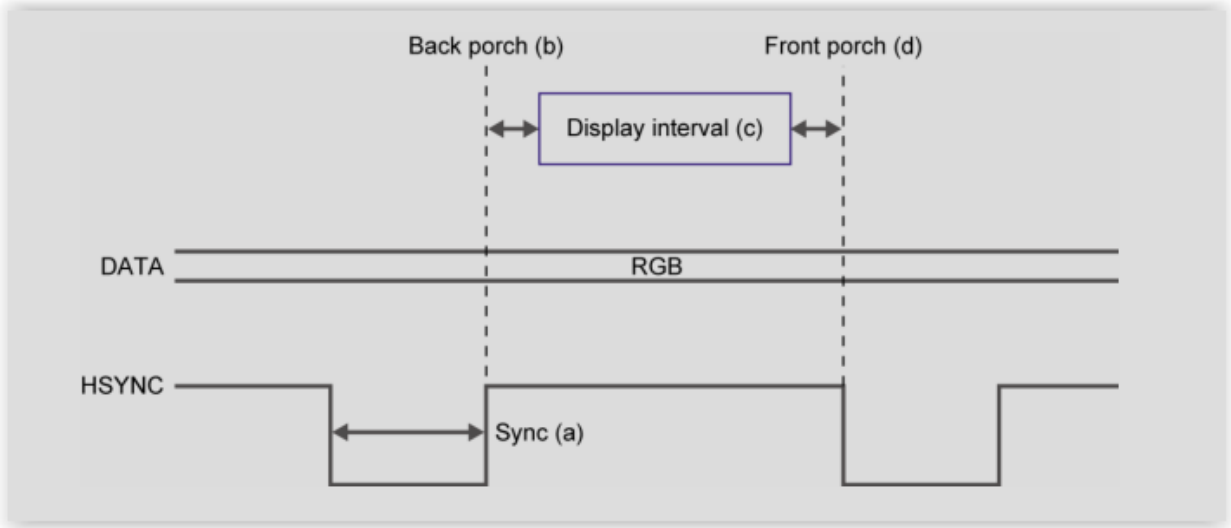
Front Porch	**5**
Sync Width	**5**
Back Porch	**20**
Blanking Total	**30**
Total Lines	**750**
Sync Polarity	**pos**

Active Pixels	921,600
Data Rate	1.78 Gbps

Frame Memory (Kbits)	
8-bit Memory	7,200
12-bit Memory	10,800
24-bit Memory	21,600
32-bit Memory	28,800

It is usually easier to think about timings in terms of pixels and lines, instead of time. A complete horizontal line consists of 1650 pixels, of which only 1280 pixels are actually displayed. Similarly, a complete screen consists of 750 lines, of which only 720 lines are actually displayed.

The diagram below shows the timing relationships for Hsync.

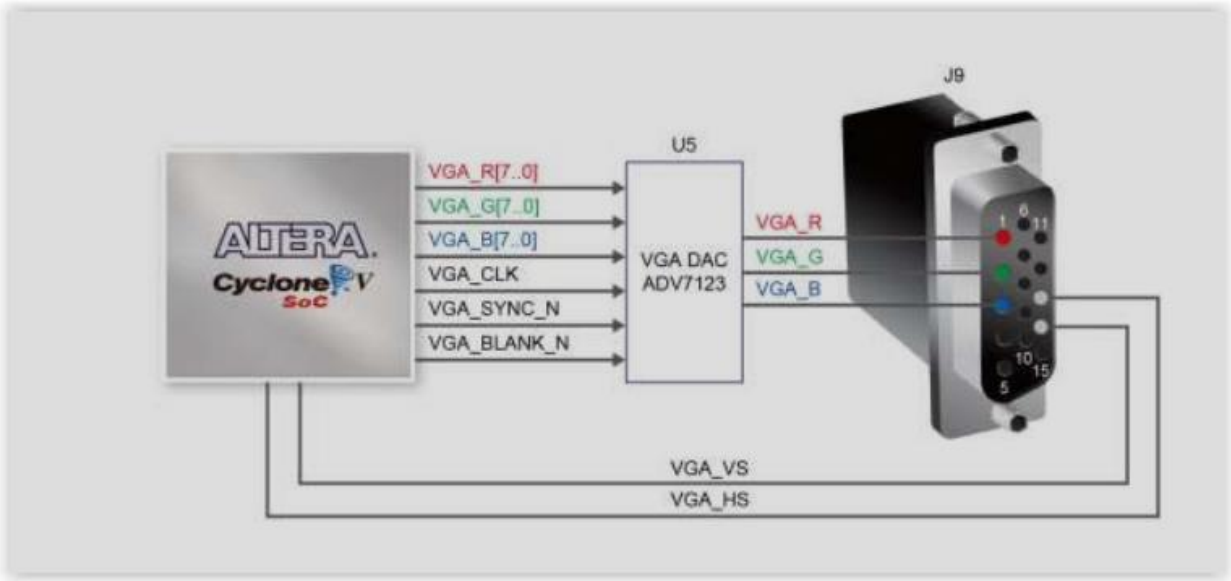


You can also use the timing diagram above for Vsync, recognizing that Vsync pulses signal the end of a complete frame of video, and that the display interval is the entire period where 720 lines of video are written.

At this point I think you have enough information to get started, except that we haven't dealt with how one turns on or off the active video.

VGA and Video DACs

The connections between the Cyclone V FPGA and the 15 pin VGA connector are shown in this diagram taken from the DE10-Standard user's manual.



The box labeled ADV7123 is a video DAC (Digital to Analog Convertor) made by Analog Devices. The datasheet for this DAC is contained in the DE10-Standard documentation package. By writing 8 bit values for each of the red, green, and blue (RGB) pixel values, the DAC will convert that information into the correct analog voltages and apply them to the red, green, and blue connector pins.

When the RGB inputs to the ADV7123 are driven with all logic 0s, video is turned off (0 v corresponds to a black screen).

The video signals that you must drive in your design are:

1. VGA_HS (Horizontal Sync), a positive going pulse for 720p and 1080p resolution.
2. VGA_VS (Vertical Sync), a positive going pulse for 720p and 1080p resolution.
3. VGA_R [7:0] (Red), VGA_G [7:0] (Green), VGA_B [7:0] (Blue)
4. VGA_BLANK_N must be tied high.
5. VGA_SYNC_N signal must be tied low.
6. VGA_CLK should be connected to your pixel clock.

General Design Guidelines

1. Keep all your design fully synchronous. Do not gate clocks.

2. Use a PLL to generate your video clock. Later in the project we will make modifications to the clocking logic, so put your clocking logic in a separate module to make this later transition easier.

If you are not familiar with how to generate and use a PLL, see the section titled PLLs later in this document.

3. All counters and/or state machine logic must have resets. Not only is this good design practice, but will allow your simulations to work. Synchronize the deassertion of reset in a separate reset module.

4. Do not hard code video parameters into your Verilog. Write your code in a manner that will allow easy conversion between VGA resolutions. Place system video parameters in a separate header file, and reference the header file where needed. I will provide you with a document titled Managing Verilog Parameters for Synthesis and Simulation that describes how to you can store parameter values in a header file and reference those values as needed.

5. Your video timing logic should be in its own Video Timing Controller module.

6. Any logic that generates RGB data should be in its own Pixel Generator module.

Design

1. Video Timing Controller (VTC)

You VTC should generate the Hsync and Vsync signals, as well out provide access to the horizontal pixel location and current line value. Also, create an active video signal that is asserted when in the active video region.

Construct your VTC using VGA 720p (1280 x 720), 60Hz timing values. When you have completed this block, construct a simulation that demonstrates correct operation.

If you are at a point where you can connect the DE10-Standard board to a monitor, you can verify at least basic operation of this circuit by driving a constant RGB value to the screen. Remember that RGB values are 8 bits wide. Writing a value of 24'h0 will result in a black screen. Writing a value of 24'hffffff will result in a white screen.

2. Pixel Generator (PG)

The purpose of the PG is to determine what color should be displayed on the screen at a given pixel location. Inputs to the PG include the horizontal and vertical locations from the VTC, and also the active video signal from the VTC. Other required inputs to the PG determine what color pixels to render at the given locations, which will be discussed next.

The outputs of the PG are the RGB values sent to the video DAC.

For this lab, create a color bar pattern to demonstrate correct circuit operation. Write 16 vertical stripes of constant width and varying colors / grayscale intensities.

The PG logic will be enhanced over the course of the semester.

Creating Video to Display

1. **Compute on the fly**

You can construct simple shapes and things on the screen simply by setting your RGB values based on your horizontal and vertical location.

For example, say you wanted to draw a vertical red line whenever your horizontal position was between 100 and 200. Your PG simply needs to look at the horizontal position counter, and set the RGB value to red whenever the counter is in the desired range. If you include the vertical counter in your calculation, you could draw a box.

You can make a box move on the screen by modifying the box position during the vertical refresh interval. One approach is to monitor a pushbutton and if the button is pressed during a vertical sync pulse, modify the vertical and/or horizontal position of the box by a small amount each time.

Compute on the fly video creation is simple, but limited. It requires no video memory, but is very limited in terms of what can be done. Remember that you need to turn off the video drive signals when you are outside the visible display area, and this is done by setting each RGB value to all zeros.

2. **Frame buffer**

A frame buffer is a block of memory that can be programmed with RBG values for each pixel. High performance systems often have two complete frame buffers so that one buffer can be modified while the other buffer is supply the video content. Single frame buffer systems often use a dual port memory approach. One port will be read by the PG to write data to the display, while the other port can be written by a CPU or debug system.

Frame buffer systems are more complex, and often strain FPGA memory capacities. For a 640 x 480 resolution, you need 307,200 locations. If you use the full 8 bits each for red, green, and blue that the DE10-Standard board supports, you will need 921,600 bytes of memory, more than our Cyclone V device contains. We will create a frame buffer using the on-chip memory later in the semester, but will need to emulate “fatter”pixels with fewer color choices to allow this to fit in the FPGA.

You can use memory external to the FPGA, such as the SDRAM chip on the DE10-Standard board. Use of external memory has its own issues, typically performance related. Successful creation of a frame buffer using external memory requires careful

design techniques to guarantee adequate memory bandwidth. Video is very unforgiving when the video data is not available. Creating a frame buffer using the SDRAM memory will be attempted toward the end of the semester.

3. **Character mapping**

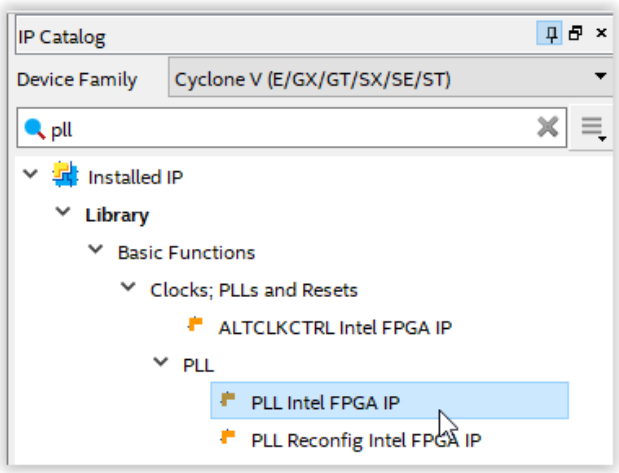
Since a full frame buffer is very memory intensive, there are numerous techniques to allow display of characters (or small game images like asteroids), and these techniques require much less memory. You divide the screen into small sections (for example 8 x 8 pixels), and then access prestored characters or icons / images from memory and display the appropriate pixels. Since the characters only require 8 bits each, and if you have a limited number of characters available, you can display a reasonable character / icon set using under 10Kbytes of memory.

This approach of creating “sprites” or “glyphs” to display is straightforward, but we will not spend time doing this during the semester. Feel free to experiment with this once the lower resolution frame buffer is complete.

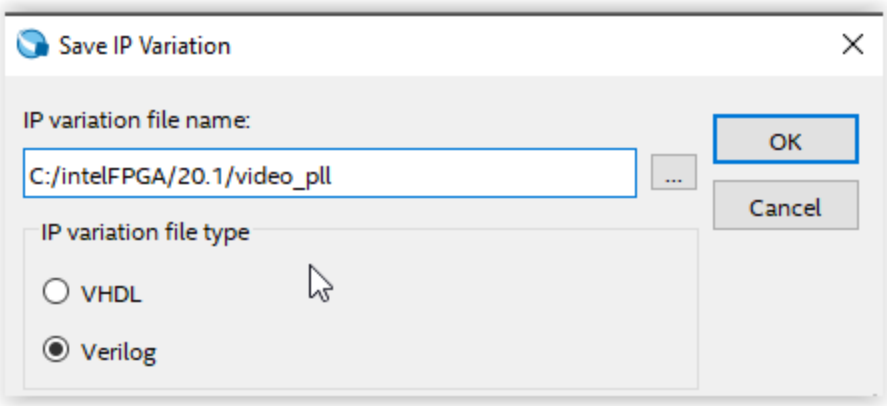
PLLs

A PLL (Phase Lock Loop) is a clocking component that can be used to create clocks of arbitrary frequency, duty cycle, and phase shift. The Cyclone V FPGA family has built in PLLs that can be accessed through the Quartus IP catalog. For this lab we will use a PLL to create a custom clock frequency for our VGA controller design.

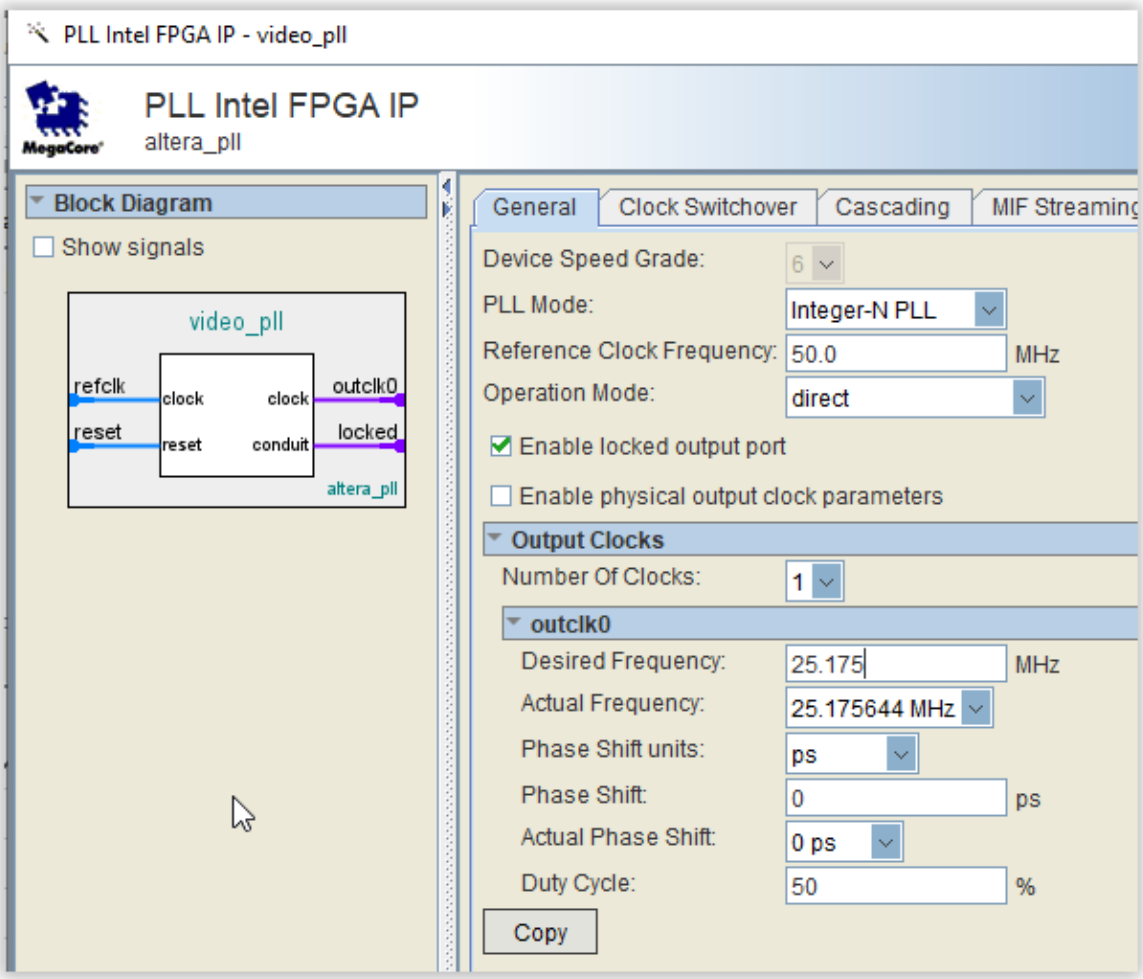
Start by opening your project in Quartus, and go to the IP catalog on the right side of the screen. If you don’t see the IP Catalog pane, open it by View > Utility Windows > IP Catalog. In the IP Catalog search for PLL, and double click on PLL Intel FPGA IP.



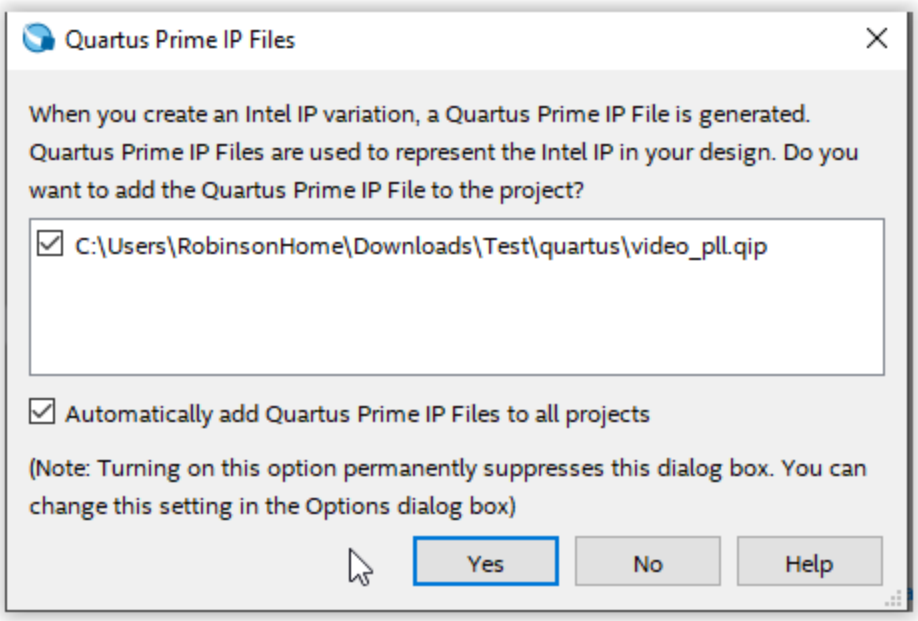
In the Save IP Variation window, provide a name of the PLL, make sure Verilog is selected, and click OK.



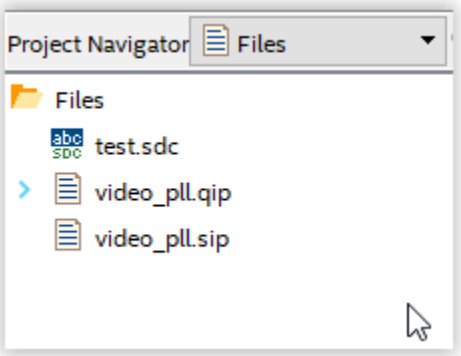
When the PLL configuration dialog opens, in the General tab, change the Reference Clock Frequency to 50 Mhz, and the outclk0 Desired Frequency to 74.25 Mhz (for 720p). Notice that the actual frequency reported will be slightly different, that will not be a problem. Click Finish.



The PLL IP will now be generated. When the window below opens, you can optionally select “Automatically add Quartus Prime Project Files to all projects”, then click Yes.



Change the drop down in the Project Navigator pane to Files, and you will see the PLL files added to your project



In order to use this PLL IP in your design, expand the files under video_pll.qip, and open the file video_pll.v. At the top of the file you will see the instantiation template for the PLL.

```
module video_pll (
      input  wire  refclk,    //  refclk.clk
      input  wire  rst,       //   reset.reset
      output wire  outclk_0,  // outclk0.clk
      output wire  locked     //  locked.export
      );
```

The `rst` polarity is positive high. The `locked` output indicates when outclk_0 is valid, as PLLs do take some time after power on to create a stable output clock.

You should connect the rst input of all PLLs to a value of 1'b1. Simulation of the PLL will not be correct if you use the integer value of 1 (I have no idea why).

Credits

Portions of this lab are based on work done by Professor Eric Brunvand and Vikas Rao at the University of Utah, as well as the Introduction to Video using FPGA apnote from Intel Corporation, authored by H. Martinez and A. Arenas.