Laboratory Exercise 10

VGA output through DDR3 SDRAM frame buffer

Outlines

I. Objectives:

This is an exercise for realizing how to access the control signals for videos such as VSHNC, HSYNC, data_enable, etc. Furthermore, DDR3 SDRAM is used for the frame buffer, whose size is significantly larger than the built-in SRAM. It solves the issue for the SRAM being too small to be a frame buffer, when it comes to high resolution video with RGB channel.

II. Experimental Procedures:

Part 1: VGA on DE1-SoC (quoted from Terasic DE1-SoC user manual P32~P34)

The DE1-SoC board has a 15-pin D-SUB connector populated for VGA output. The VGA synchronization signals are generated directly from the Cyclone V SoC FPGA, and the Analog Devices ADV7123 triple 10-bit high-speed video DAC (only the higher 8-bits are used) transforms signals from digital to analog to represent three fundamental colors (red, green, and blue). It can support up to SXGA standard (1280*1024) with signals transmitted at 100MHz. Figure 1 shows the signals connected between the FPGA and VGA.

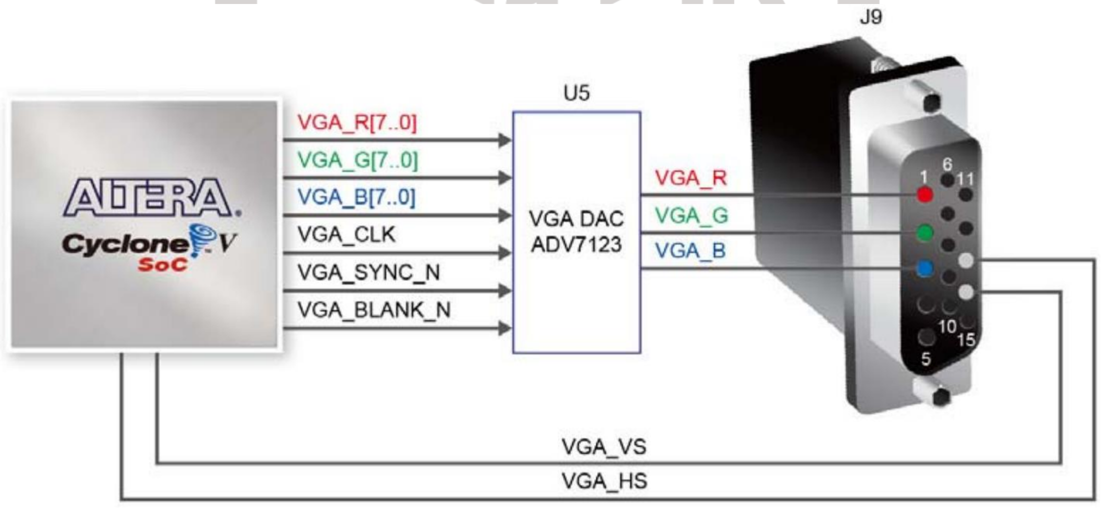


Figure 1. Connections between the FPGA and VGA

The timing specification for VGA synchronization and RGB (red, green, blue) data can be easily found on website nowadays. Figure 2 illustrates the basic timing requirements for each row (horizontal) displayed on a VGA monitor. An active-low pulse of specific duration is applied to the horizontal synchronization (hsync) input of the monitor, which signifies the end of one row of data and the start of the next.

The data (RGB) output to the monitor must be off (driven to 0 V) for a time period called the back porch (b) after the hsync pulse occurs, which is followed by the display interval (c). During the data display interval the RGB data drives each pixel in turn across the row being displayed. Finally, there is a time period called the front porch (d) where the RGB signals must again be off before the next hsync pulse can occur. The timing of vertical synchronization (vsync) is similar to the one shown in Figure 2, except that a vsync pulse signifies the end of one frame and the start of the next, and the data refers to the set of rows in the frame (horizontal timing). Table 1 and Table 2 show different resolutions and durations of time period a, b, c, and d for both horizontal and vertical timing.

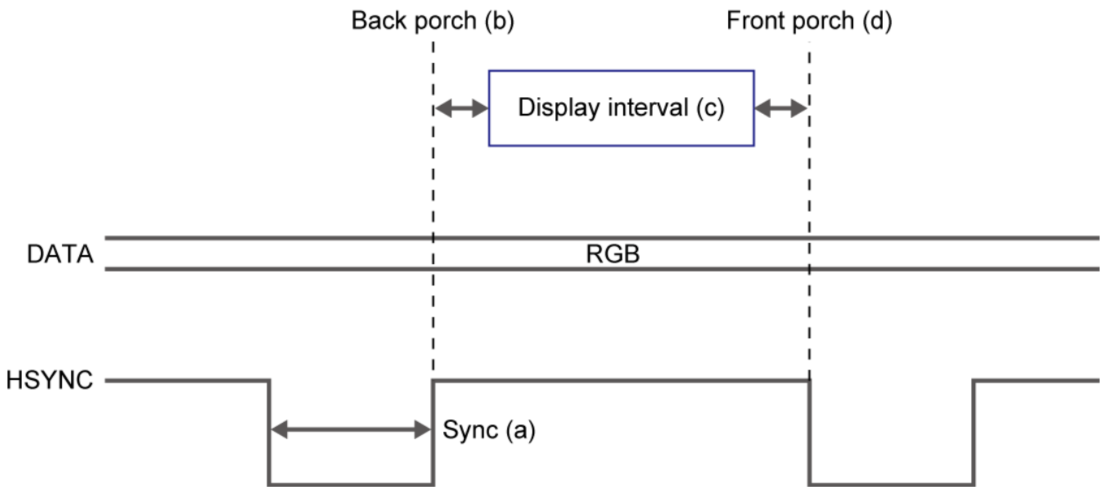
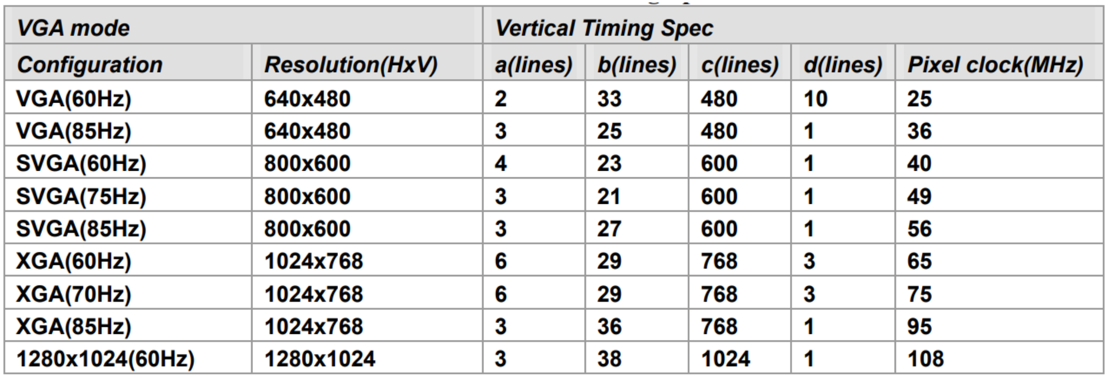


Figure 2 VGA horizontal timing specification

Table 1. VGA Horizontal Timing Specification

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**

Table 2. VGA Vertical Timing Specification



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

Part 2: Display a frame with resolution of 640*480

For this part, TAs will provide you a file, called “input_badge.qxp”. You can view this file(a module) as an encrypted Verilog file, “input_badge.v” which contains a video with I/O shown in Table 3. The contents about how to access the qxp file refers to the end of this lab (page 10-7). The output resolution of the frame is 640*480 with RGB channels. Since the video contains only one frame, and the module continuously outputs that frame, if you connect the signals to VGA output pins correctly, the monitor will show a static picture as Figure 3.

Table 3.

Input		**Output**	
iStart_n	When is set to 0, the module starts to output the control signals and the pixels. (You can set it to be always 0 if you don’t need it.)	***oData_Enable***	Data_enable. When it is 1, the pixel (oData) is valid. It will be 1 when the frame is during the Display interval in Figure 2.
iCLK	25.175MHz Clock	***oData[23:0]***	24-bits pixel. oData[7:0] represents red. oData[15:8] represents green. oData[23:16] represents blue
iRST_N	Reset signal(You can set it to be always 1 if you don’t need it)	**HSYNC**	HYNC signal in Figure 2.
		VSYNC	VSYNC signal in Figure 2.

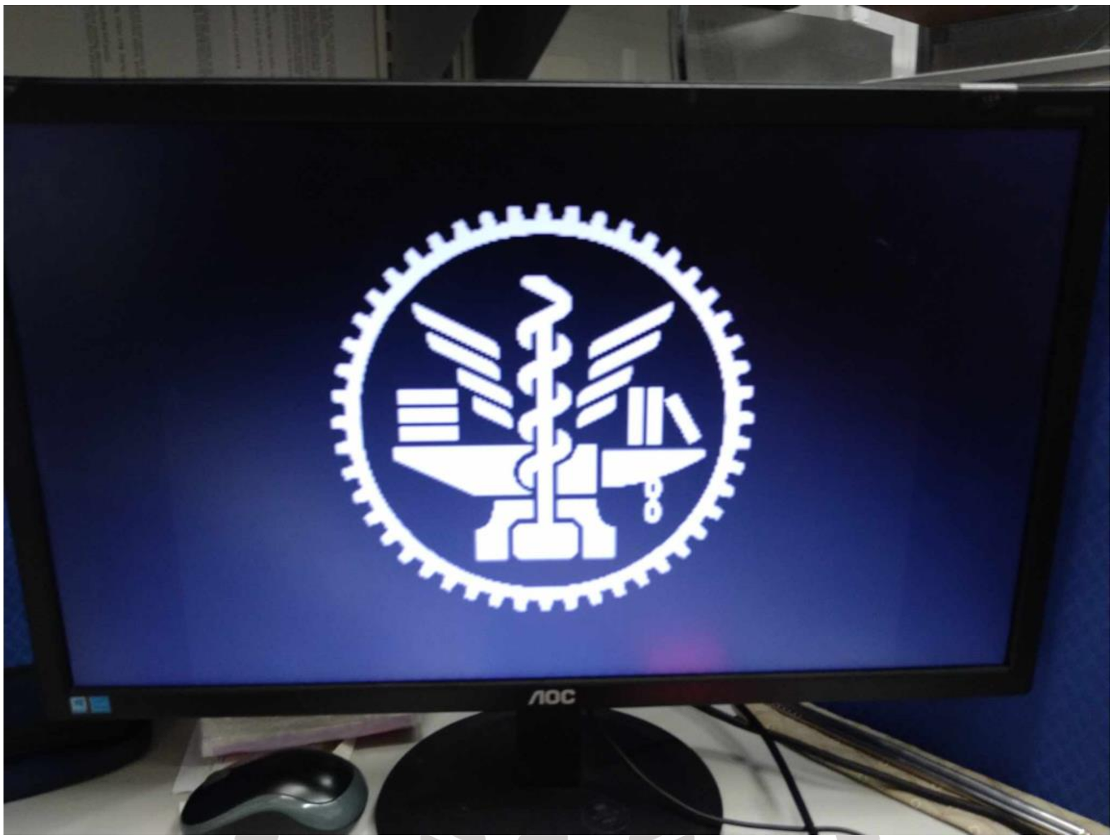


Figure 3.

Noted that the resolution of the frame is 640*480 while the monitor is not. Accordingly, different monitor may contribute to different result (the black region at left part and right part may be filled with stretched frame due to automatic adjustment by monitor).

Part 3: Adding adjustment on the pixels during the flow of the pixels (3 points)

For this part you have to add a white book on the badge in the frame. The white book must be a rectangle, whose four corners are at (203, 207), (203, 197), (248, 197), and (248, 207) respectively. (The coordinate of the top-left corner is (0,0) while the coordinate of the bottom-right corner is (639,479).)

You have to use oData_enable to count which pixels are the ones you want to deal with during the image flow. Subsequently, use SW[0] to choose the original frame or the adjusted frame for the monitor to display. The result should be like Figure 4.

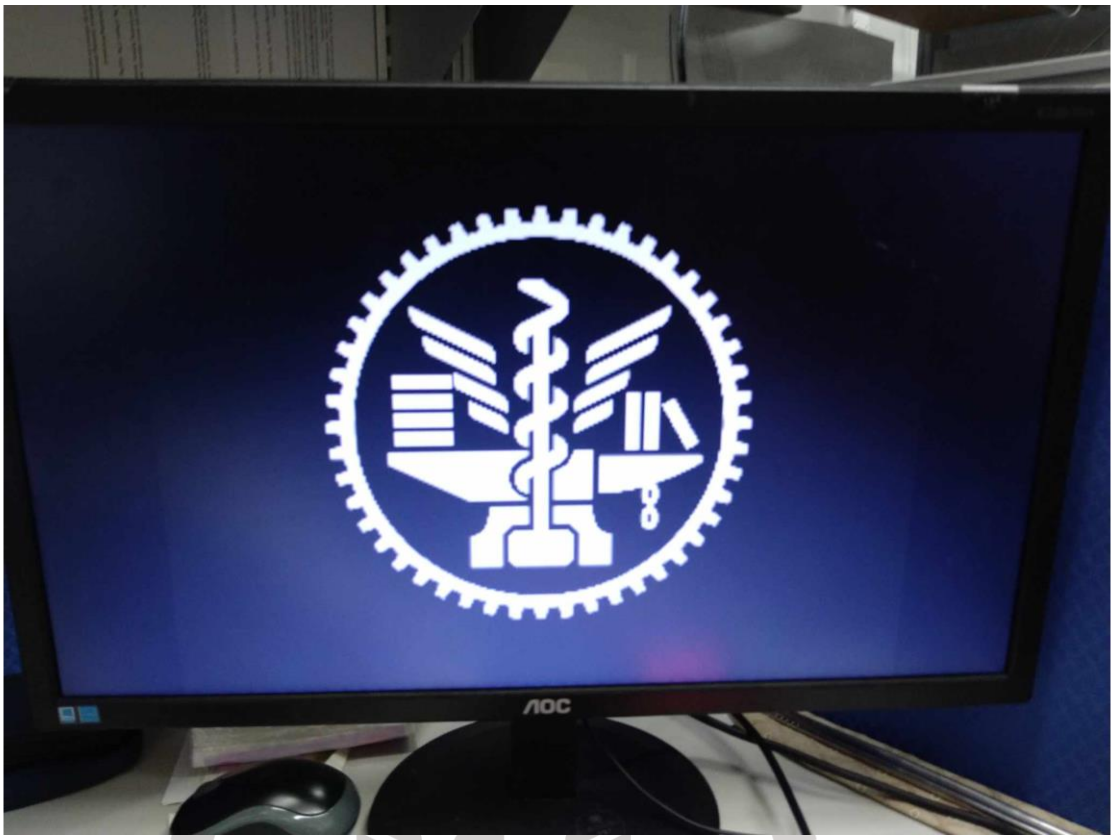


Figure 4.

Part 4: Use SDRAM as frame buffer to flip the image (4 points)

For this part, you have to store and refresh every frame in the image flow in the SDRAM for the purpose of flipping the frames horizontally or vertically, controlled by SW[1:0]. “input_badge.qxp” at this part, or treated as “input_badge.v”, will give you a video different from the one of the previous part, where the badge rolls from the top of the frame to the bottom of the frame repeatedly as shown in the demo video.

You are supposed to cooperate with the other module provided by TAs, “VGA_640_480.v”, which provides control signals for VGA output pins. When it requests pixel, you have to give the pixel stored in your SDRAM at the next clock.

You can use FIFO to cope with the CDC problem, since the input frame is provided at 25.175MHz while the DDR3 SDRAM is working at 100MHz. Because the ROM for badge in “input_badge.qxp” occupies a great magnitude of the memory block in DE1-SoC board, the FIFO you build in MegaWizard should not be too large, or it may lead to a failure of compilation. This warning by TAs simultaneously represents the fact that you cannot use SRAM as frame buffer, because the frame is quite large. Therefore, you must use DDR3 SDRAM as frame buffer.

It is important that you should keep the empty signal of FIFO in mind. If you don’t make some restrictions on the relationship of the FIFO in your SDRAM controller and the “VGA_640_480.v” module, when the module requests pixels at the situation that the FIFO is empty, it may result in column-shifting for the output frame.

“VGA_640_480.v” contains an input iStart_n. When it is set to 1, it is never allowed to request pixels from frame buffer. Consequently, connect the empty signal from FIFO to this pin may be a good solution of the issue about column-shifting. Alternatively, you can design a module to control the VGA output by yourself, too.

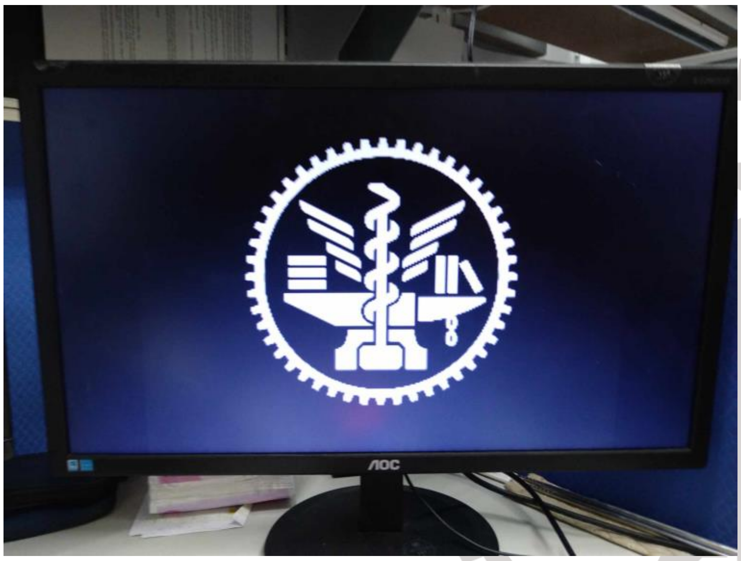


Figure 5. SW[1:0] = 2’b00

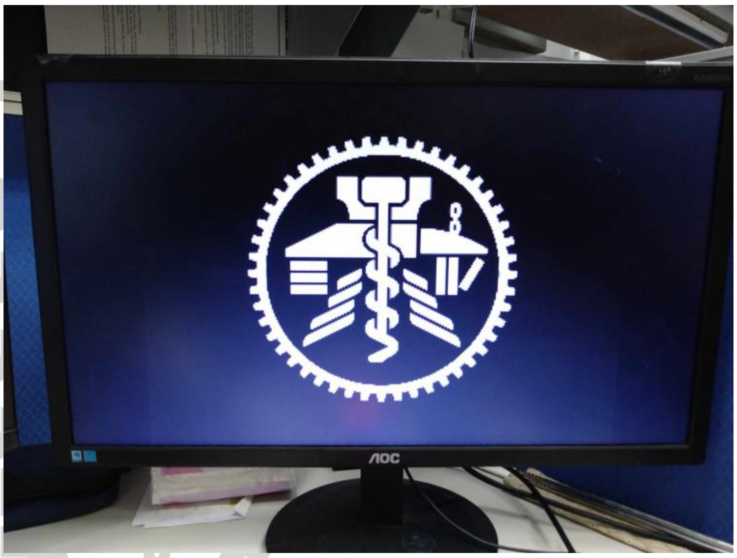


Figure 6. SW[1:0] = 2’b01

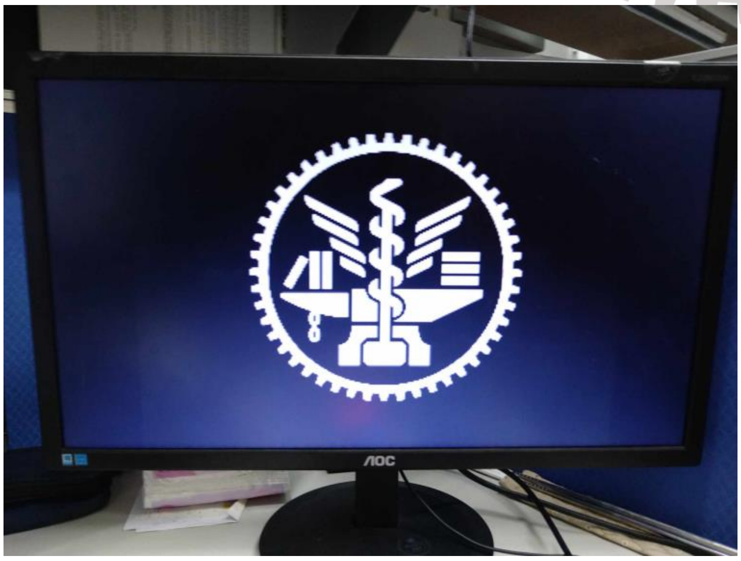


Figure 7. SW[1:0] = 2’b10

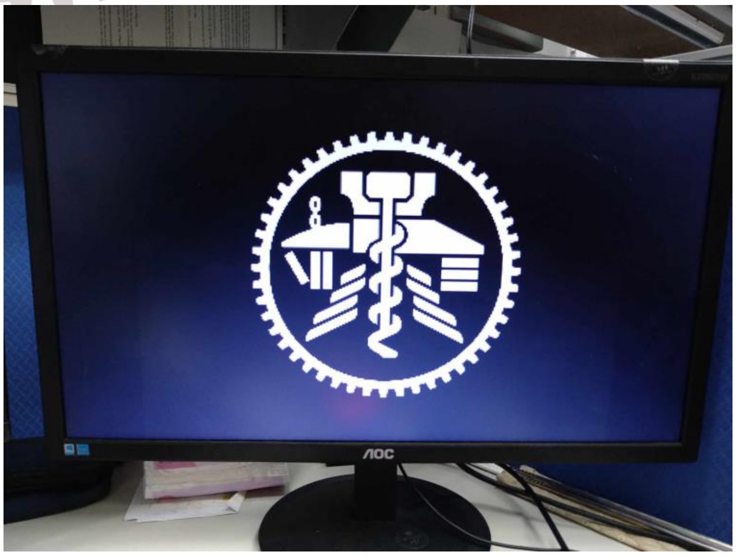


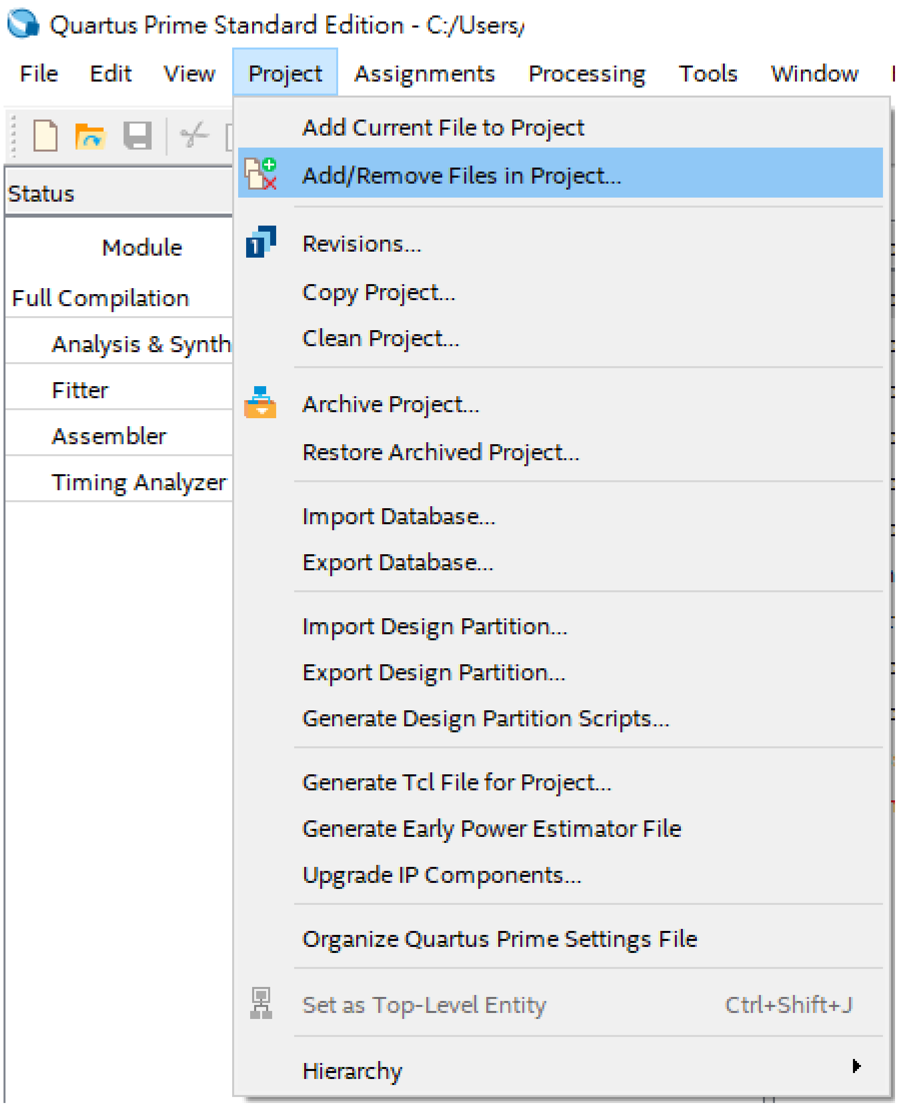
Figure 8. SW[1:0] = 2’b11

Check points:

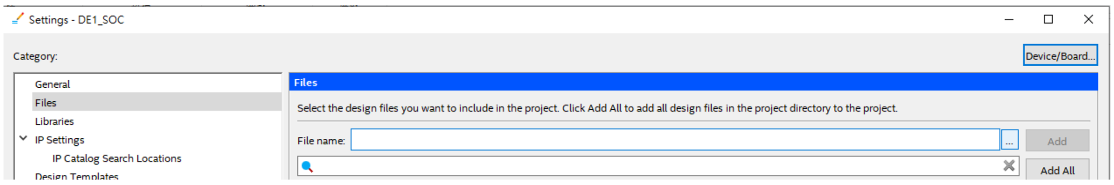
1. When SW[1:0] is 2’b00, the badge should not be flipped.
2. When SW[1:0] is 2’b01, the badge should be flipped vertically, but not horizontally.
3. When SW[1:0] is 2’b10, the badge should be flipped horizontally, but not vertically.
4. When SW[1:0] is 2’b11, the badge should be flipped horizontally and vertically.
5. Whatever the input SW[1:0] is, the badge should be rolling without column shifting.

Appendix: Use qxp file as a black box Verilog module

Step 1: Project->Add/Remove Files in Project


Quartus Prime Standard Edition - C:/Users/
File
Edit
View
Project
Assignments
Processing
Tools
Window
Status
Module
Full Compilation
Analysis & Synth
Fitter
Assembler
Timing Analyzer
Add Current File to Project
Add/Remove Files in Project...
Revisions...
Copy Project...
Clean Project...
Archive Project...
Restore Archived Project...
Import Database...
Export Database...
Import Design Partition...
Export Design Partition...
Generate Design Partition Scripts...
Generate Tcl File for Project...
Generate Early Power Estimator File
Upgrade IP Components...
Organize Quartus Prime Settings File
Set as Top-Level Entity Ctrl+Shift+J
Hierarchy

Step 2: Press the icon with three dots.

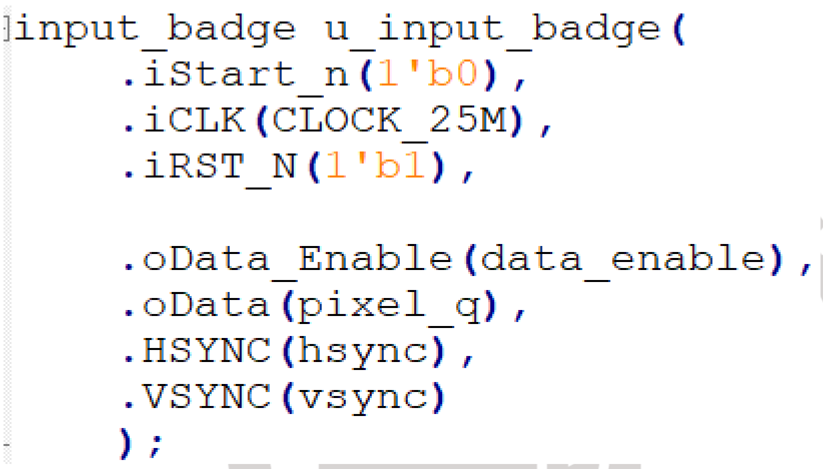


Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.

Step 3: Choose the qxp file “input_badge.qxp”

Step 4: Press “Apply” and then press “OK”.

Step 5: Use the module “input_badge.v” as you are using IPs.

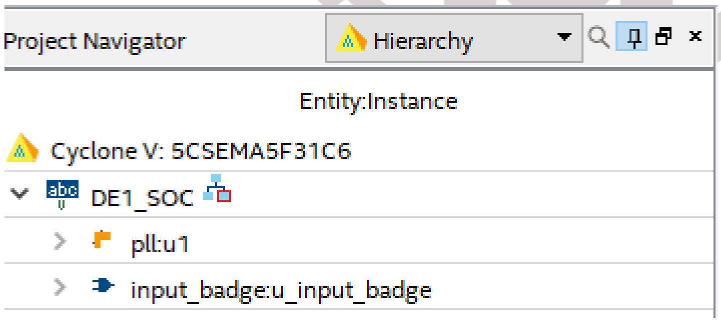


```
input_badge u_input_badge(
      .iStart_n(1'b0),
      .iCLK(CLOCK_25M),
      .iRST_N(1'b1),

      .oData_Enable(data_enable),
      .oData(pixel_q),
      .HSYNC(hsync),
      .VSYNC(vsync)
      );
```

Step 6: Compile your project.

Step 7: After compiling, you can see the module you used in the project navigator.


Project Navigator
Hierarchy
Entity:Instance
Cyclone V: 5CSEMA5F31C6
DE1_SOC
pll:u1
input_badge:u_input_badge