**ECE324 Lab 10: Video Sketch**

Name(s):

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| **Exercise** | **Course outcome** | **Grade** |
| Lab10 Demo | 2.a, 2.d, 5.c, 7.b | /15 |
| Lab10 Code | 2.a, 5.c, 7.b | /5 |
|  | **TOTAL:** | /20 |

2.a. Define engineering problems from specified needs for digital systems including implementation on FPGAs using HDL programming.

2.d. Produce FPGA designs that meet specified needs.

5.c. Collaborate with individuals with diverse backgrounds, skills and perspectives.

7.b. Employ appropriate learning strategies such as communicating with an expert, using external resources, experimentation, simulation, etc.

# Lab 10 Learning objectives

1. Learn how to generate IP using the Xilinx’s Clocking Wizard and Block Memory Generator.
2. Understand video color, BRAM, frame buffer, and pipelining concepts.
3. Learn how to detect and fix a clock period timing violation.
4. Observe how a mouse may be implemented with a Nexys4DDR.
5. Using SystemVerilog HDL, make modifications to a moderately complex design of a video driver circuit, use Xilinx Vivado to implement the circuit in an FPGA chip, download the implemented design to a Nexys4DDR board, and display the results on a video monitor.

# Procedure

1. Vivado setup  
   In Vivado, add design sources hex\_to\_sseg\_p.sv, led\_mux8\_p.sv, and ps2rx.sv, which have been used in previous labs. Also add design sources ps2tx.sv and ps2\_top.sv leveraged from Pong Chu’s SystemVerilog book, and mouse.v leveraged from Pong Chu’s older Verilog book. Also add new design sources BresanhamLineDraw.sv, divideBy3.sv, sketchFrame.sv, configColors.sv, displayMouseSprite.sv, orderedDither.sv, and Lab10\_VideoSketch.sv. Add constraint file Lab10\_VideoSketch.xdc. While you don’t need to add the files mouseSpriteMem.txt and paletteTileMem.txt, make sure they’re in the same directory as your Xilinx project file.
2. Color  
   In the output port declarations of configColors.sv, replace the eight initializations of the palette colors, which are currently shades of grey (R=G=B), with fully saturated colors in the following order: black, blue, green, cyan, red, magenta, yellow, white.
3. Clock IP  
   The sxga (1280x1024) logic with a 60 Hz refresh rate has a specified 108 MHz clock frequency. In a previous lab, you were given a file that had been derived using Xilinx’s clock wizard IP (Intellectual Property) which instantiates a clock primitive that generates 108 MHz, but now you will generate it yourself using the following steps:
4. In Flow Navigator/Project Manager, click on IP Catalog. Under Cores/Vivado Repository/FPGA Features and Design/Clocking, double click on Clocking Wizard.
5. At the top, change Component Name to videoClk108MHz (**use identical capitalization for all names**). Change the Input Clock Primary Port Name from clk\_in1 to CLK100MHZ. Do **not** hit OK yet.
6. In the Output Clocks page, change the Port Name from clk\_out1 to clk108MHz, and make the requested clk\_out1 frequency 108 MHz. After pressing enter, verify to the right that the Actual frequency it will generate is the same as what you requested. Also, at the bottom of the same page, uncheck reset and locked, since we don’t need these signals.
7. In the Summary page, see how 108 MHz will be generated by verifying with a calculator that Input Clock Frequency / Divide Counter \* Mult Counter / MMCM CLKOUT0 Divider Value = 108MHz.
8. Click OK in the bottom right corner, and in the Generate Output Products page press Generate. When you get the “Out of Context…” message, press OK. This message indicates the module is being generated and synthesized without knowledge of the design into which you will later instantiate it, and that’s okay.
9. Under Project Manager/ IP Sources/IP/videoClk108MHz/Instantiation Template, when all the file names have been changed from red to black, you’ll know that the IP is done being generated. In the meantime, double click on videoClk108MHz.veo. Scroll to the bottom, and verify that this template’s instantiation of the clock generator is the same as what is already in Lab9\_VideoSketch.sv, except for the instance name that’s been changed (normally you would copy the instantiation from this template and change the name yourself, but this has already been done for you).
10. Block Memory Generator IP  
    Using Xilinx’s Block Memory Generator, generate a frame buffer that contains 1310720 (1280 columns x 1024 rows) pixels 3 bits each, built out of a composite SRAM (built out of many BRAMs and LUTs):
11. In Flow Navigator/Project Manager, click on IP Catalog. Under Cores/Vivado Repository/Memory & Storage Elements/RAMS & ROMS & BRAM, double click on Block Memory Generator.
12. On the Basic page, change the Component Name to frameBuffer, and change the Memory Type to True Dual Port RAM.
13. On the Port A Options page, ideally you would like to make the Port A Write Depth, which is the number of words in the memory, 1310720 (1280 columns x 1024 rows). However the note to the right of the configuration of Port A Write Depth indicates its maximum allowed value is 1048576. One solution to accommodate this restriction is to pack 2 pixels (6 bits) into each memory word, thus requiring only half the number of memory words (1310720 / 2 = 655360). Change the Port A Write Width to 6 (pushing enter will also change the Read Width to 6), and change the Port A Write Depth to 655360 (pushing enter will also change the Read Depth to 655360). The Block Memory Generator allows you to have a different Read Width and Depth from a Write Width and Depth, so to be able to read one pixel at a time out of the frame buffer, change the Port A Read Width to 3 (which will change the Port A Read Depth to 1310720).
14. Lower on the same Port A Options page, check Core Output Register (leaving Primitives Output Register also checked).
15. On the Port B Options page, change the Port B Write and Read Widths from 6 to 3 (which will change the Port B Write and Read Depths to 1310720, enough words for 1280 columns x 1024 rows). Also uncheck Primitives Output Register (leaving Core Output Register also unchecked).
16. On the Summary page, confirm that the composite SRAM will consume 120 36K Block RAMs (and no 18K Block RAMs), the Port A Read Latency is 3 clock cycles, and the Port B Read Latency is 1 clock cycle.
17. This frame buffer design uses 120 out of the 135 36K Block RAMS in the XC7A100T, but you need to have more than 15 BRAMs to implement the rest of the features you plan to put into this FPGA. By understanding how the Block Memory Generator packs together BRAMs with their limited width capabilities, you realize that the port B width specification of 3 results in using BRAMs that are only 1 and 2 bits wide, so none of the “parity” bits are used, resulting in using at most 32Kbits out of the 36Kbits in each BRAM. However, if three horizontally-adjacent 3-bit pixels were packed into 9-bit memory words, then the Block Memory Generator can use 9-wide BRAMs, which will use the “parity” bits, resulting in fewer BRAMs needed. The number of memory words needed in each row would be 1280 / 3 = 427 (rounded up). So for both Port A and Port B, change both the Write and Read Port Widths to 9, and change both the Write and Read Port Depths to 437248 (427 triColumns x 1024 rows). Verify in the Summary page that the number of 36K BRAMs was reduced from 120 to 107, which meets your objectives.
18. On the Other Options page, under Memory Initialization, check Fill Remaining Memory Locations, and change Remaining Memory Locations (Hex) to 1FF, which initially sets all 9 bits in every word of the SRAM.
19. Click OK in the bottom right corner, and in the Generate Output Products page press Generate. When you get the “Out of Context…” message, press OK.
20. Under Project Manager/Sources/IP Sources/IP/ frameBuffer/Instantiation Template, double click on frameBuffer.veo, scroll to the bottom, and verify that this template’s instantiation is the same as what is already in Lab10\_VideoSketch.sv, except for the instance name that’s been changed, and the names to what the ports are connected.

When all of the IP is generated, synthesize Lab10\_VideoSketch. While waiting, bring down WSU’s sxga (1280x1024) monitor from the top lab shelf (please ask for help if you’re uncomfortable bringing it down and/or putting it back up). Connect the monitor’s VGA cable to the VGA connector near the top right of the Nexys4DDR board. Before turning on the power for the Nexys4DDR, put the blue jumper on the JP1 MODE connector into the center JTAG position (to prevent the Flash from programming the FPGA which could put the Mouse into an undesirable state). Plug the mouse into the USB HOST connector near the top left of the board, and then turn the board power on.

When implementation is done, observe in Project Manager/Project Summary/Implementation that there was a critical warning, which was because the tools can’t make this design run at 108 MHz under worst case temperature and voltage conditions. Fix this (and give yourself timing margin for a future higher frequency design) by adding four more pipeline stages:

1. Go to Project Manager/Sources/IP Sources/IP, and double click on frameBuffer. Leaving everything else alone, add two pipeline stages to Port B by implementing both Primitive and Core Output Registers. After confirming both ports now have a Read Latency of 3 clock cycles, press okay to make the changes to frameBuffer.
2. At the end of sketchFrame.sv, the pipeline flip-flops are missing for all signals in pipeline stages 3-6, so pipelining is not being done (even though the signal names are labeled as if it is being done). For these four stages, add pipeline flip-flops by changing the always\_comb statements to always\_ff @(posedge clk108MHz), and change all blocking assignments (=) to non-blocking (<=).

Resynthesize, and verify the Implementation critical warning is gone, because the additional pipelining enabled the higher frequency. Note in Vivado 2018.2, unfortunately each Project Summary/Implementation message doesn’t get updated by the new Implementation, but you can do an update by clicking on the critical warning message, and it should disappear.

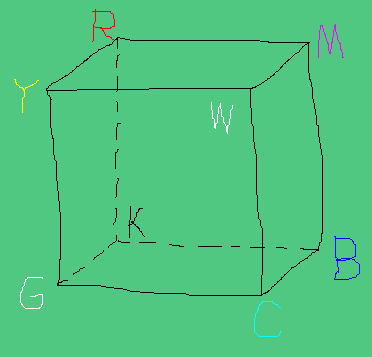
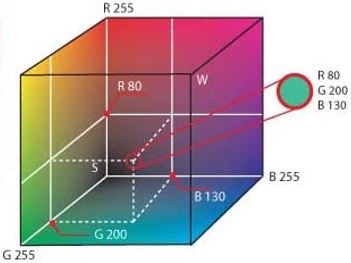
**Put the mouse on a mouse pad or a sheet of paper**; WSU’s mice don’t work well with the black lab benches. Try drawing a line by pressing the left mouse button and moving the mouse. If the cursor doesn’t move, the mouse may have gotten into an undesirable state; unplug the mouse (to remove power from the mouse), re-plug it back in, Program the FPGA again, and try drawing a line again.

To configure different colors of lines, click the right mouse button. At any given time, there can be in the sketching area only 8 colors at most, specified by the 8 palette colors (also called index colors) shown in the upper right corner as 8 vertically-adjacent “buttons”. The background color is always specified by the bottom of the 8 palette “buttons” (palette7). The currently selected color to be drawn is highlighted, but you may select a different palette color by moving the mouse to a different palette “button” and pushing the left mouse button. You can also change any of the 8 palette colors by first selecting which of the 8 you want to change, and then while observing in the lower right of the display the new color in the preview palette “button” and preview line, left clicking the mouse over any of the following locations:

1. Directly to the left of the palette and preview colors are 4096 = 2^12 colors, arranged as 16 cross sections of a 16x16x16 color cube. This is all the native colors available with 4 bits each of red, green, and blue that the Nexys4DDR is capable of providing.
2. Next to the left is a column which provides a grey gradient of 256 shades of grey (8 bits/color). However, when halftoning isn’t enabled, the bottom 4 bits of each color are truncated, so only 16 shades of grey (4 bits/color) are displayed. To produce smoother gradients in the display (at the expense of causing some grain), click the center mouse button which toggles whether halftoning is enabled.
3. Next to the left are three gradient columns of red, green, and blue. Note selecting a color in any of these three columns doesn’t affect either of the two other colors, so you can independently modify red, green, and blue. Three markers indicate the amplitudes of red, green, and blue in the currently selected palette.
4. On the left side of the monitor is an enlarged cross section of a 256x256x256 color cube (displayed as a 16x16x16 color cube if halftoning is disabled). A crosshair marker points to the current palette values of red (horizontal marker) and blue (vertical marker).

The 7-segment displays show the hexadecimal 24-bit RGB color pointed to by the mouse (and under the markers). Clicking the right mouse button again toggles back to the sketch display. You may “erase” the whole frame buffer to the current palette7 color by pushing the red “CPU RESET” button above the DIGILENT insignia (but it doesn’t change any palette changes you’ve made). Play with changing and drawing colors so you can understand how the code provided works.

For your demonstration, change the background color on palette7 to the color displayed in the circle in the left color cube diagram shown below (the decimal color of R=80, G=200, B=130 corresponds to the hexadecimal color 50\_C8\_82 on the 7-segment displays). Don’t expect a perfect color match; different monitors (and printouts with different inks and light sources) will display colors somewhat differently. Similar to the below right diagram (which was generated using Microsoft Paint), roughly sketch the outline of the color cube using 12 black lines, and label all 8 corners with the letter representing the color name (KCMYRGBW), using that same color of text, except use white to label G. Demonstrate this to the professor or lab assistant.



# Deliverables:

1. Demonstrate operation of your completed FPGA design to the instructor or lab assistant.
2. No written lab report is necessary for this lab.
3. Document any SystemVerilog code you modified. Your SystemVerilog code ***must include header comments stating your names, date and class number.*** Any changes to the SystemVerilog code ***must also include comments explaining the operation of the code***.
4. Upload all of the .sv and .xdf text files used for your solution (zipped together if you like), and then hit the submit button just once.

# References

Chu, Pong, *FPGA Prototyping by SystemVerilog Examples*, Wiley 2018, ISBN 9781119282662