San Jose State University
Department of Electrical Engineering
EE178, Fall 2017, Crabill

Laboratory Assignment #6

Objectives

The goal of this lab is for you to implement a line drawing algorithm for integration with a frame buffer and output it to a video display, using the VGA connector on the Digilent Basys3 board. Additionally, you will learn how to obtain a RAM from the Xilinx Vivado IP Catalog. When you successfully complete this lab, you will have developed a piece of intellectual property that you might be able to re-use in the future.

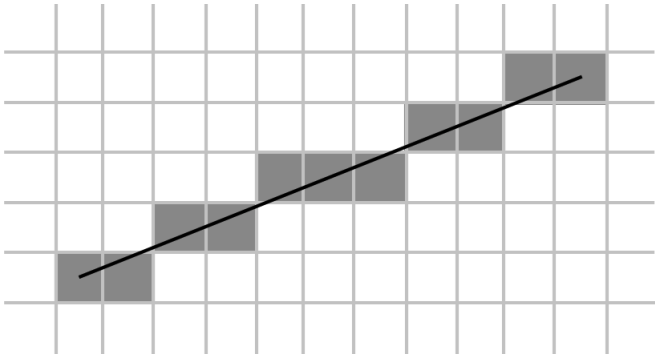

Figure 1: Made in San Jose [Wikipedia]

Now that you are familiar with the tools from Laboratory Assignment #1, you should be able to concern yourself with digital design. Figure 2 shows a symbol of the design you will create. The user interface inputs and outputs are shown on the left and the display outputs are shown on the right.

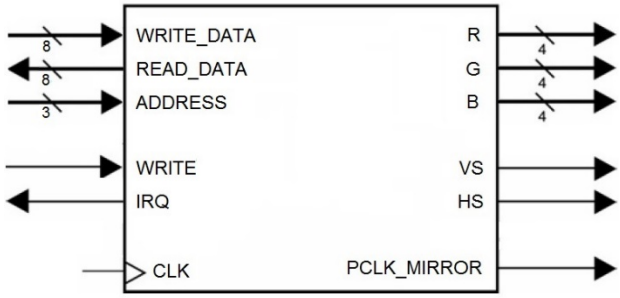

Figure 2: A Symbol of the Design

Bibliography

This lab draws heavily from documents on the Digilent website http://www.digilentinc.com. I would like to thank Digilent for making this material available.

Bresenham's Line Algorithm was developed by Jack Bresenham, in San Jose, in 1962. Most references, such as the Bresenham's Line Algorithm Wikipedia entry, present the algorithm in the context of software. An extremely helpful exposition on hardware implementation of the algorithm, by Stephen Edwards of Columbia University Computer Science Department, is available on his CSEE4840 Embedded System Design website. I would like to thank Stephen Edwards for sharing his material.

Portions of the test bench provided for this lab are based on the TIFF Revision 6.0 Specification from Adobe Systems Incorporated.

Design Description and Requirements

In this design, you are not allowed to use latches. You are allowed to use only one clock. The clock must be a 40 MHz clock signal derived from the 100 MHz clock signal available from the oscillator on the board. You will receive zero points if you do not follow these requirements.

As shown in Figure 2, the display interface consists of six output signals, five of which are used for display signaling, and a sixth which is used to provide an external copy of the 40 MHz clock signal that is derived internal to the design. This external copy of the 40 MHz clock signal exists so that the test bench can operate synchronously with the design logic during simulations.

clk	clock signal, 100 MHz from oscillator
r[3:0]	digital red output, drives a resistive DAC
g[3:0]	digital green output, drives a resistive DAC
b[3:0]	digital blue output, drives a resistive DAC
vs	vertical sync
hs	horizontal sync
pclk_mirror	pixel clock, copy of derived internal clock

In addition to the display interface, there is also a user interface which consists of five signals. These five signals provide read / write access to an eight byte register file. The register file is used to exchange information with the frame buffer and the line drawing algorithm.

write_data[7:0]	eight-bit data input to the register file
read_data[7:0]	eight-bit data output from the register file
address[2:0]	three-bit address input to select a register
write	write enable for the register file
irq	indicates start of vertical blanking interval

The user interface consists of an eight-bit data input, an eight-bit data output, a register select, and a write enable. At all times, the content of the register currently selected by address can be read on read_data. To write the register file, select a register using address, provide data on write_data, and assert write for one pclk cycle. The register map is shown in the following table:

Addr	Name	Access	Description
0	STAX	Read/Write	Starting X coordinate of line. On screen, 0x00 is left, 0xFF is right.
1	STAY	Read/Write	Starting Y coordinate of line. On screen, 0x00 is top, 0xFF is bottom.
2	ENDX	Read/Write	Ending X coordinate of line. On screen, 0x00 is left, 0xFF is right.
3	ENDY	Read/Write	Ending Y coordinate of line. On screen, 0x00 is top, 0xFF is bottom.
4	BUSY	Read/Write	Busy status and command. Read result of 0x00 means idle, and read result of 0x01 means busy. When idle, write of 0x01 is Go command.
5	BEAM	Read/Write	Line drawing pen intensity, 4-bit grayscale ranges from 0x00 for black up through 0x0F for white.
6	MODE	Read/Write	Frame buffer mode: 0x00 is “hold”, frame buffer maintains the image. 0x01 is “erase”, fills frame buffer with black during a scan. 0x02 is “exp decay”, divides intensity of stored image by 2 each scan. 0x03 is “lin decay”, reduces intensity of stored image by 1 each scan.
7	PRNG	Read Only	Pseudo-random number generator (reserved for future use).

The frame buffer mode can be changed at any time by writing MODE. It is recommended to change MODE during the vertical blanking interval, as the frame buffer display fetch is not active at that time.

A line is drawn by writing desired values to BEAM, STAX, STAY, ENDX, ENDY, followed by a write of 0x01 to BUSY. Within two clock cycles, the observable BUSY value must change from 0x00 to 0x01, indicating the line drawing algorithm is active. When the line drawing algorithm has completed, the observable BUSY value changes from 0x01 to 0x00, signaling the hardware is idle and ready to begin drawing another line.

The design must be capable of drawing reasonably good looking lines between arbitrary start and end points. Although no line drawing performance requirement is enforced, your goal should be a draw rate of one pixel per cycle.

Constructing the Baseline Design

Create a project using the following files posted on the class website. Instead of creating source files and copying text, use the add source capability to directly add the files to the project. When adding files, you have the option to copy the files into the project directory – which is advisable, so that the files do not get separated from the project directory.

1. testbench.v (top level testbench)
2. tiff_writer.v (simulation display capture, sub-module used by testbench)
3. vga_example.v (top level design, enhanced version of previous lab)
4. vga_example.xdc (top level design constraint file, enhanced version of previous lab)
5. vga_timing.v (timing controller, sub-module used by vga_example)
6. linedraw.v (line drawing algorithm, sub-module used by vga_example)

After reviewing the files and their contents, you will notice that the timing controller module has a declaration, including port names, but otherwise no contents. Copy in your timing controller from the previous lab.

The design uses a 256x256 frame buffer, with a 4-bit intensity value for each of the 65,536 pixels. The frame buffer is implemented as a dual-ported RAM, with one port for the line drawing algorithm and the other port for the display generation process. You will need to create this RAM using the Xilinx Vivado IP Catalog.

Begin by opening the IP Catalog using the button in the Flow Navigator. In the IP Catalog window, you can search for the Block Memory Generator, or locate it under the Memories & Storage Elements section of the IP Catalog, as shown in Figure 3.

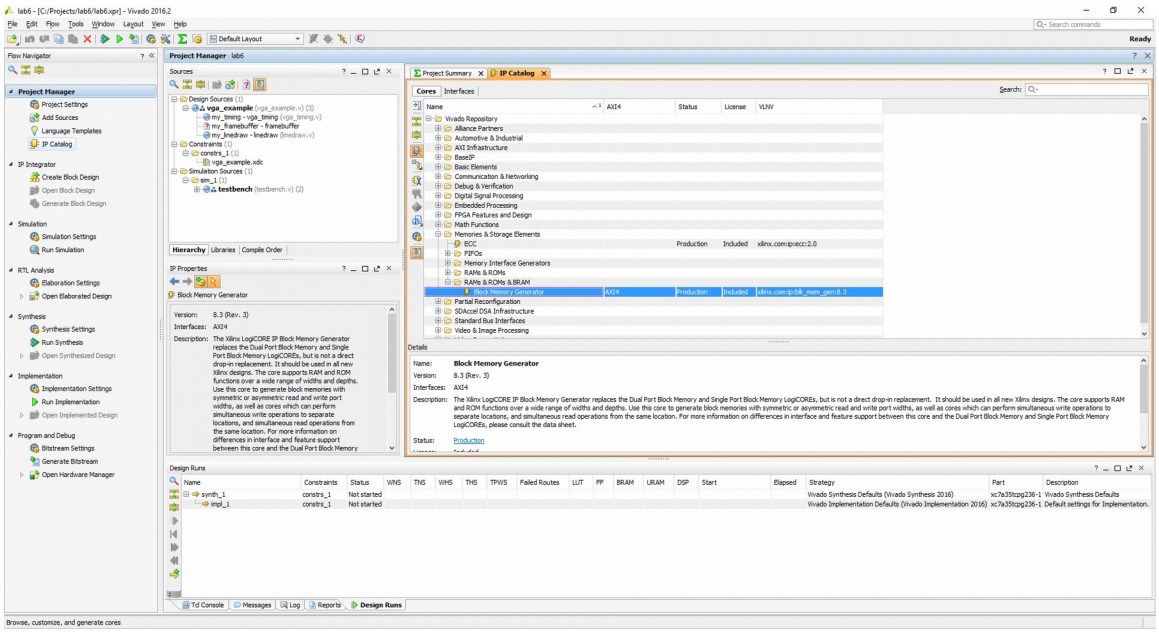

Figure 3: Block Memory Generator in Vivado IP Catalog

Open the Block Memory Generator and configure the RAM for this project. Refer to Figure 4, Figure 5, Figure 6, Figure 7, and Figure 8 – take your time, be accurate, and only click OK after you have reviewed all tabs.

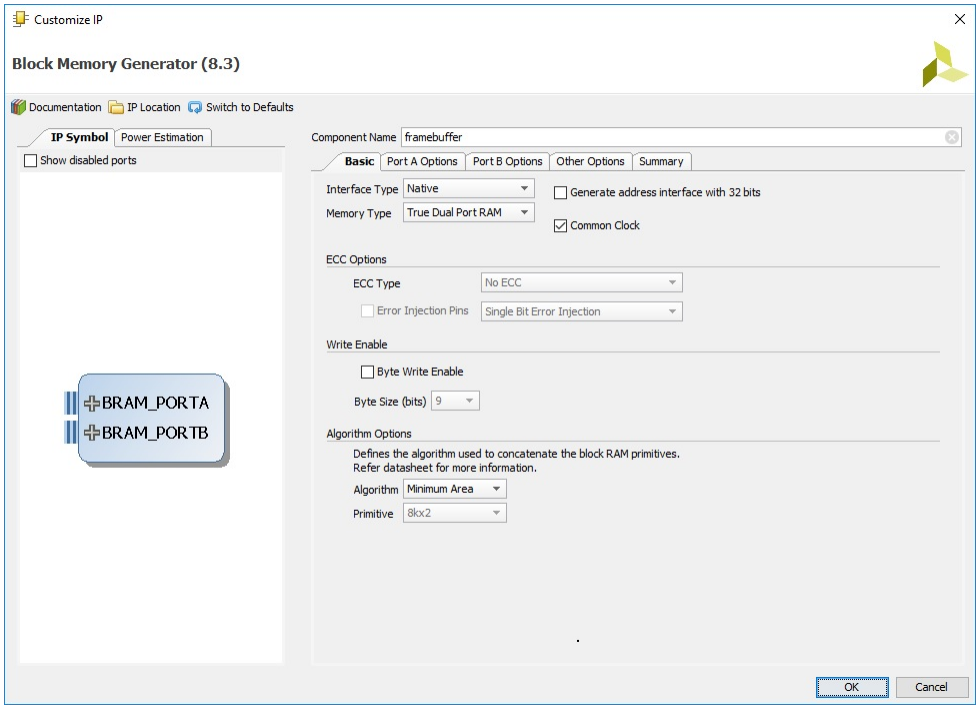

Figure 4: Block Memory Generator, RAM (Basic Tab)

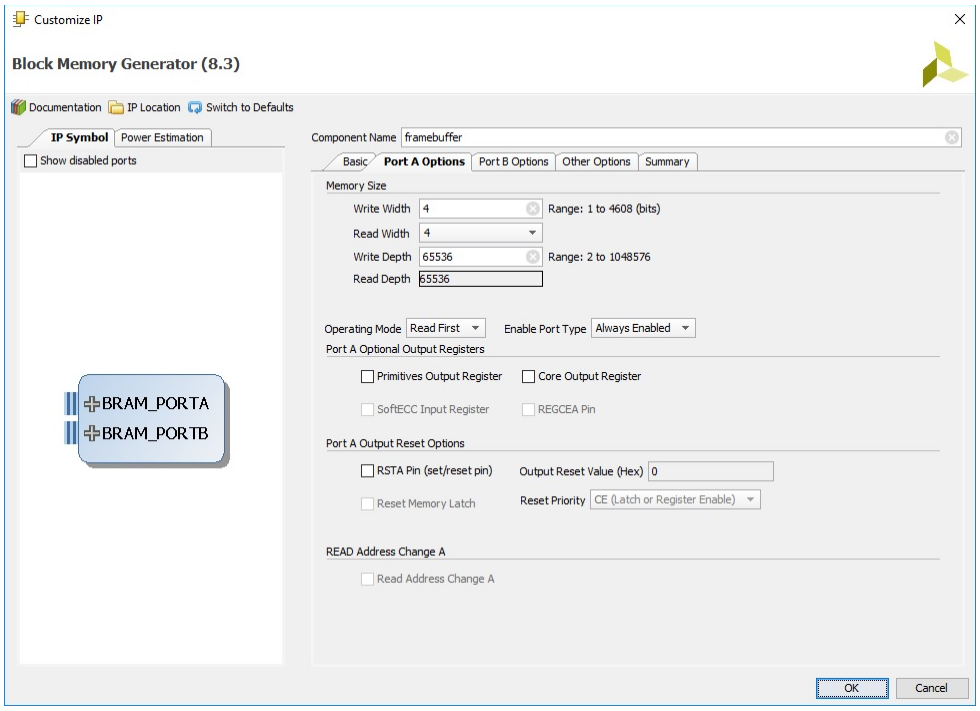

Figure 5: Block Memory Generator, RAM (Port A Options Tab)

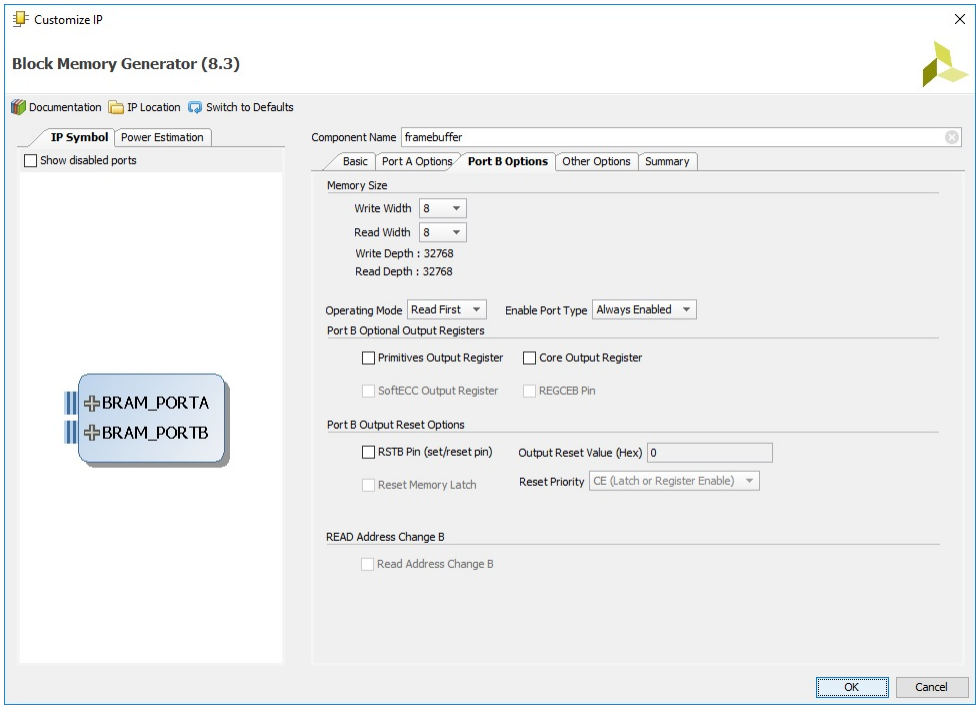

Figure 6: Block Memory Generator, RAM (Port B Options Tab)

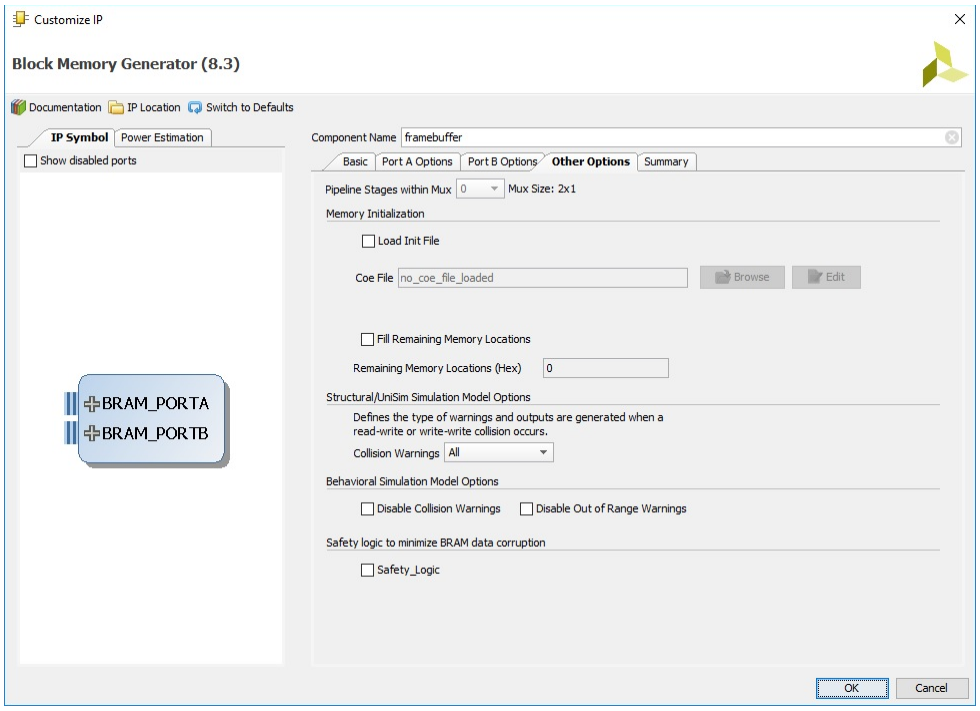

Figure 7: Block Memory Generator, RAM (Other Options Tab)

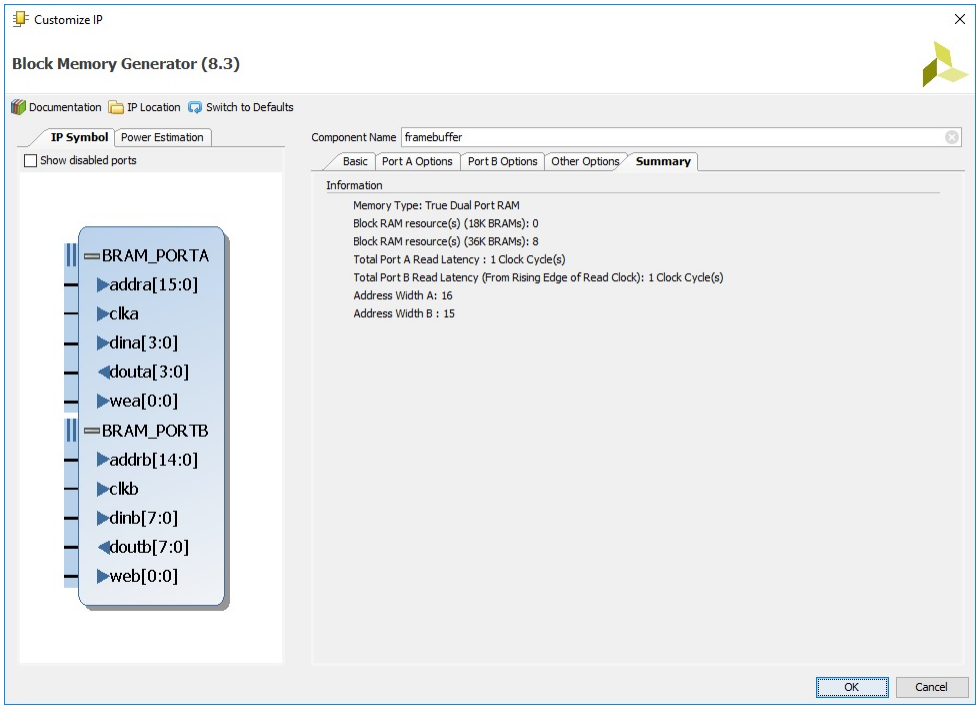

Figure 8: Block Memory Generator, RAM (Summary Tab)

When you click OK, a list of the output products to be generated will appear. Click Generate, which begins a process in which the RAM source code is generated and synthesized.

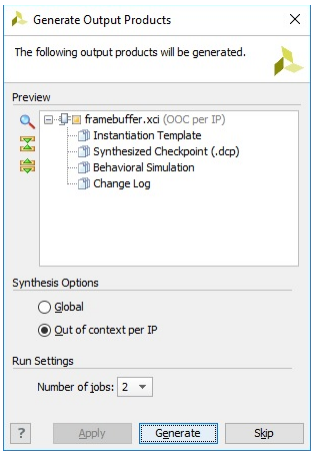

Figure 9: Output Product List

This process is called "out of context" synthesis because the RAM source code is synthesized alone. The purpose of "out of context" synthesis is to reduce time spent in synthesis over the duration of a project by pre-synthesizing modules not likely to change, and preserving the synthesis results, so it need not be done over and over again.

One of the output products listed in Figure 9 is the instantiation template, a time-saving convenience. As shown in Figure 10, you can look in the IP Sources tab of the Sources window to find instantiation template files for any generated IP in your project. Open the instantiation template file for the RAM. You don't need to do anything with this, as the RAM has already been instantiated. However, if you were designing from scratch and wanted to instantiate this generated IP, the instantiation template makes it easy.

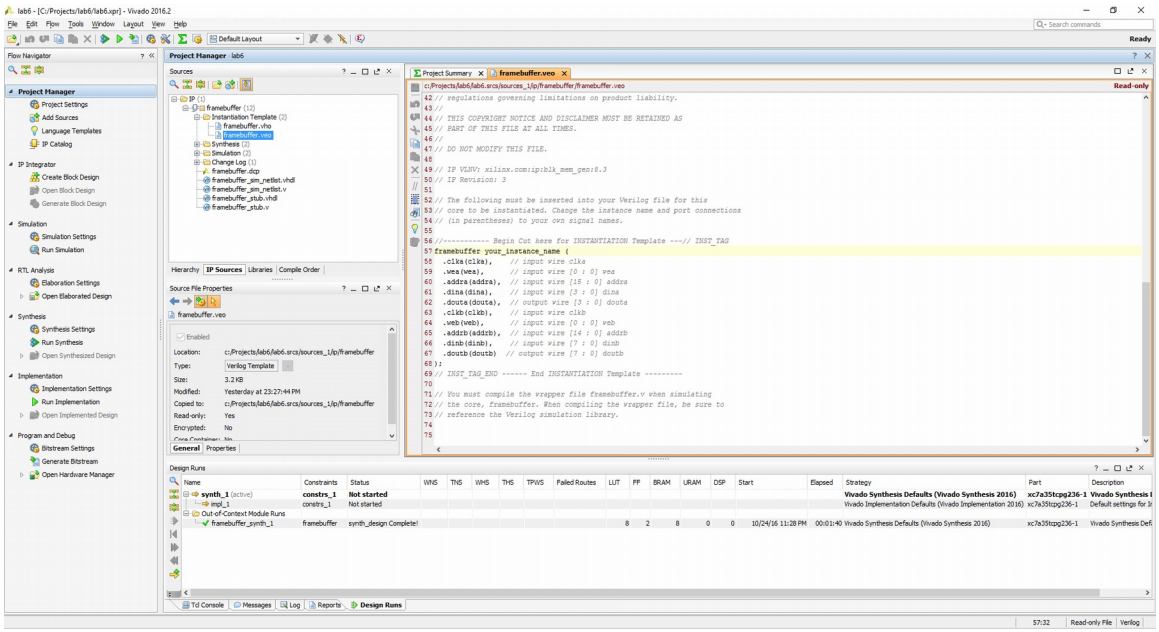

Figure 10: RAM Instantiation Template

Now, with the timing controller added and the RAM generated, only the linedraw module should need work. You will notice that the linedraw module has a declaration, including port names, and some simple contents as a placeholder. The simple contents implement a pixel write function, so that register accesses intended to program a line draw result in a single pixel write at the start location set by STAX and STAY.

The primary goal of this assignment is to implement a line drawing algorithm in this module, replacing the simple contents.

However, before starting work on the linedraw module, it is critical that you verify your baseline design is fully functional. If your timing controller has unresolved bugs, or you have made an error in generating the RAM, you must detect and resolve these through simulation-based verification and hardware validation.

Testing the Baseline Design

You must perform functional simulation of the baseline design with the provided test bench. This is important for two reasons. First, it will give you confidence your baseline design is working properly before you implement it. Second, if the baseline design does not behave as expected when you download it, you will have a mechanism to quickly create additional test cases to help debug the problem. The instructor will not help you debug unless you are able to run a simulation.

The provided test bench automatically generates TIFF output files that show your simulation results as they would appear on the display, frame by frame. The files will be located in one of the project simulation directories, for example lab6\lab6.sim\sim_1\behav and are sequentially named frame000.tif, frame001.tif, etc… Figure 11 shows the expected first frame output of the baseline design with the provided test bench.

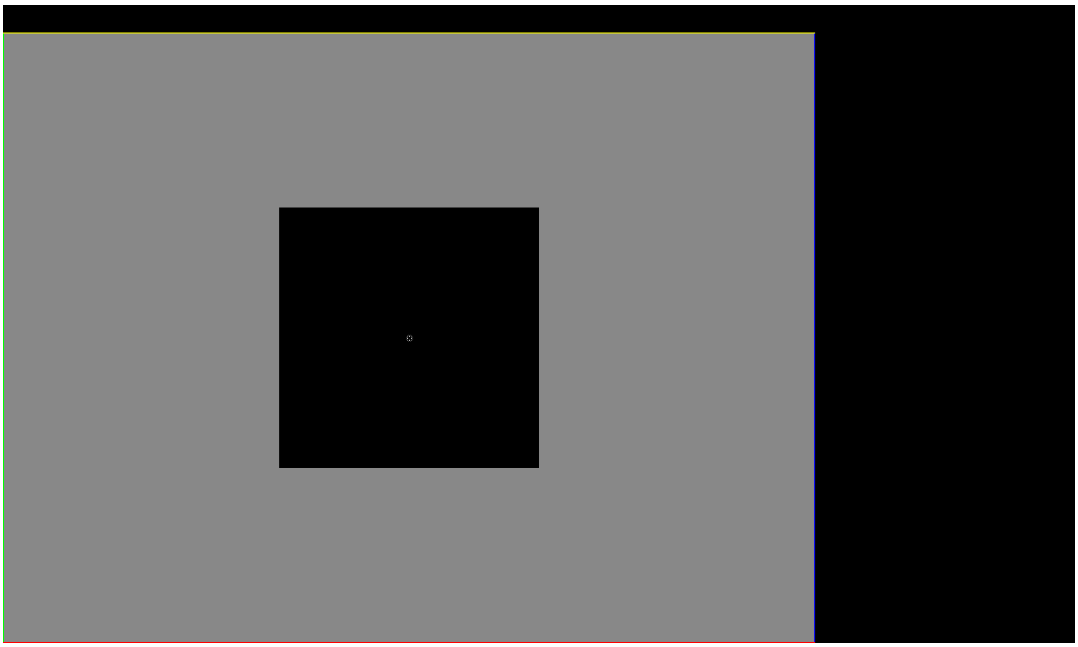

Figure 11: Expected First Frame Output of the Baseline Design with Provided Test Bench

Inspect your results *carefully*. The black regions around the visible area represent the display blanking time and are only partially visible on an actual display. The black square in the middle if the visible area is 256x256 pixels and represents the contents of the frame buffer. There should be a small pattern in the center of this square, shown magnified in Figure 12. Each one of the pixels in the pattern is the start point of a line draw attempt by the test bench.

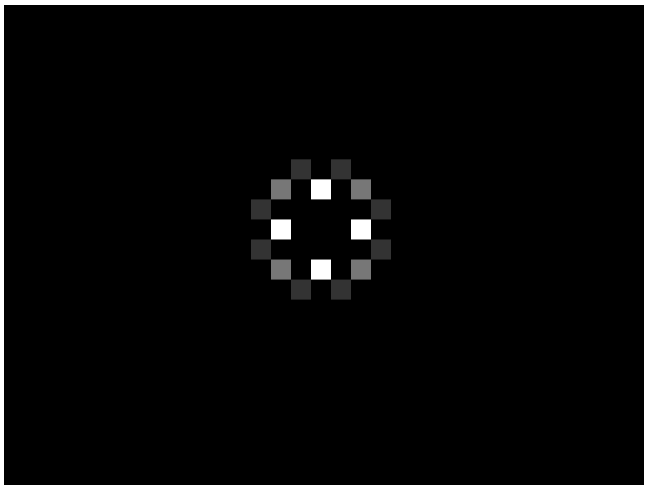

Figure 12: Magnified Pattern in Center of Frame Buffer

Synthesizing and Implementing the Baseline Design

Synthesize and implement your design. Do not forget to check the reports. As a general practice, you will want to review all errors and warnings. These point to areas of concern that you should either address or justify. If the design fails one or more timing specifications the reports will indicate this is the case.

Test your design in hardware. Does the circuit behave as you expect? If it does not, seek assistance before you proceed to implement the linedraw module.

Describing the Design

Once the baseline design is working as expected, the only design source to be edited is the linedraw module. Do not add additional ports, although you may change the data type of ports:

```
module linedraw (
  input wire go,
  output wire busy,
  input wire [7:0] stax,
  input wire [7:0] stay,
  input wire [7:0] endx,
  input wire [7:0] endy,
  output wire wr,
  output wire [15:0] addr,
  input wire pclk
  );

  // Stuff goes here...

endmodule
```

The go and busy signals are for command and status of the line drawing algorithm. The line drawing algorithm must begin in an idle state, with busy = 0. Assuming the line drawing algorithm indicates it is idle, an external stimulus programs it to begin by setting go = 1 for one or more clock cycles. The line drawing algorithm does not need to draw immediately, but it must set busy = 1 the cycle after go = 1 is first detected. When the line drawing algorithm has completed, it must return busy = 0.

Note, if go = 1 is maintained for a long time (as will be the case when we are operating this design with buttons and switches) the expected behavior is for the line drawing algorithm to execute correctly and return idle, then repeat the draw correctly as many times as go = 1 is maintained.

The stax, stay, endx, and endy inputs are each 8-bit values from the register file for indicating the starting and ending coordinates of the line to be drawn.

The line drawing algorithm must "plot" pixels into the frame buffer by controlling the wr and addr signals. The wr signal is the frame buffer write enable, and the addr is the frame buffer address, where addr[15:8] represents the y-coordinate and addr[7:0] represents the x-coordinate (remembering that the frame buffer is 256x256 pixels, so there is an 8-bit field for each axis).

To write a pixel, provide the desired coordinate on addr and hold the wr signal asserted for a clock cycle. The data written into the frame buffer is the pixel intensity, which is controlled through the register file independently of the linedraw module. Also, as a reminder, the origin of the frame buffer is in the upper left corner – a convention carried forward from the video timing controller, as the origin of the display is also in the upper left corner.

You are not expected to invent an algorithm. You should research an algorithm, such as Bresenham's Line Algorithm, and implement it. From the bibliography, I highly recommend the exposition on hardware implementation of Bresehnam's Line Algorithm by Stephen Edwards.

In my own search, I found a number of implementations – some understandable, and others not. When you are evaluating the available references, a critical factor in deciding which to use is understandability. Assume your capture and integration of the algorithm is not going to work right on the first try and that you will need to debug it – if you don't understand how it works, you will have a very difficult time debugging.

Before you begin writing any code, you must have a plan for a circuit that will satisfy the design requirements. Once you have a possible solution, write a description of it in Verilog-HDL and proceed to test it in simulation. Use the provided files for your design. You may change the declared port data types if you need to suit your design description.

Testing the Design

You must perform additional functional simulation of the design with the provided test bench. This is important for two reasons. First, it will give you confidence your design is working properly before you implement it. Second, if the design does not behave as expected when you download it, you will have a mechanism to quickly create additional test cases to help debug the problem. The instructor will not help you debug logic problems (incorrect design behavior) unless you have a block diagram and are able to run a simulation.

Figure 13 shows the expected first frame output of the design with the provided test bench. It consists of sixteen lines, drawn radially outward from the center. Four lines, drawn in white, are horizontal and vertical line tests. Another four lines, drawn in medium gray, are diagonal line tests. The remaining eight lines, drawn in dark gray, are oblique line tests to exercise the drawing algorithm in each of the octants.

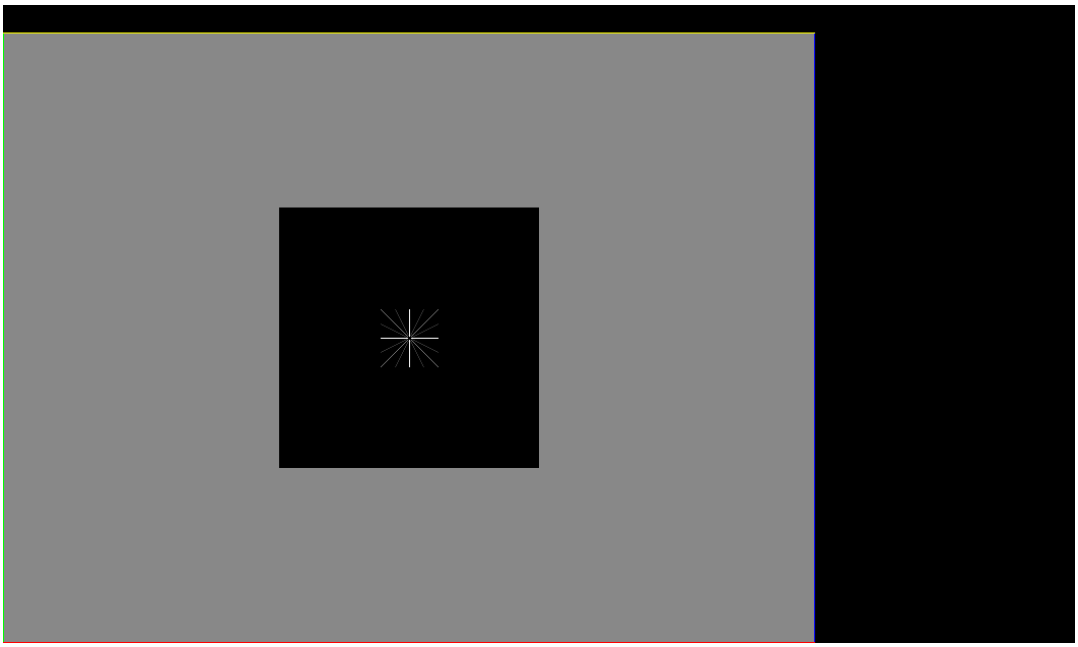

Figure 13: Expected First Frame Output of the Design with Provided Test Bench

Synthesizing and Implementing the Design

Synthesize and implement your design. Do not forget to check the reports. As a general practice, you will want to review all errors and warnings. These point to areas of concern that you should either address or justify. If the design fails one or more timing specifications the reports will indicate this is the case.

Test your design in hardware. Does the circuit behave as you expect? If it does not, seek assistance before you proceed to customize the test bench.

Customizing the Test Bench for Demonstration

You must modify the test bench to draw your first and last initial, somewhere around the perimeter of the visible area. Do not remove the line draw test pattern. For example, in my own implementation, the finished result would display EC in addition to the original test pattern.

Draw your initials with reasonably long lines of a variety of sizes and angles – that is, don't create tiny "pixel art" and don't restrict yourself to only horizontal and vertical lines. Be creative, your goal is to exercise the functionality of your line drawing algorithm implementation. Once you are confident the design works properly and your text is legible in the simulation, demonstrate your final result to the instructor. The instructor will evaluate your design in hardware by drawing a test line.

Laboratory Hand-In Requirements

Once you have completed a working design, prepare for the submission process. You are required to demonstrate a working design. Within four hours of your demonstration, you are required to submit your entire project directory in the form of a compressed ZIP archive. You must include a simulation-generated TIFF file showing the output from your customized test bench. Use WinZIP to archive the entire project directory, and name the archive lab6_yourlastname_yourfirstname.zip. For example, if I were to make a submission, it would be lab6_crabill_eric.zip. Then email the archive to the instructor. Only WinZIP archives will be accepted.

Given the amount of simulation in this lab, the ZIP archive may be extremely large. Often, you can reduce the ZIP archive size if you review the ZIP archive contents and delete any WDB, waveform data base, files in the simulation directories.

Demonstrations must be made on or before the due date. If your circuit is not completely functional by the due date, you should turn in what you have to receive partial credit.