Lab4: VGA Pong

Objective and Overview

This lab exercise is an opportunity to focus on learning about sequential components and designing hardware threads. Hierarchical, synthesizable, procedural modeling in SystemVerilog will be used to specify the design. The design will be simulated, synthesized and downloaded to the DE2-115 FPGA board for demonstration.

Schedule and Scoring

Lab 4 is another two-week exercise. You will want to accomplish some of the design task prior to coming to lab, but the remainder should be easily completed during your lab sessions. **You are expected to show up on time, be present and work on the lab for the entirety of the first lab session.**

31 Oct – 2 November	Task 2 demo is due by end of the lab session. **Do not** stop there!
6 November	Section A alternate Lab date. Final demos due by 7pm on Wednesday, 8 November.
7 November	Democracy Day. No lab.
8 – 9 November	Final demos due by end of the lab session.

Lab 4 is worth 125 points:

- **5 points:** Component Library. Well defined and constructed library file with the HW6 components plus the `RangeCheck`, and `OffsetCheck` components. Each component should be documented in comments. It should also be tested in a single or separate testbenches. These will be checked during your Task 2 demo.
- **10 points:** Task 2 Documentation. Neatly drawn and up-to-date STD and datapath schematic. These will be checked during your Task 2 demo.
- **20 points:** Demonstrate Task 2. Show that you have created a VGA module driving a test pattern to the display. This is to be demonstrated by the end of the 1st session. The test pattern is generated by combinational circuitry built around instantiations of `RangeCheck`, and `OffsetCheck` components. A 5 point deduction will be made each time that you unsuccessfully demonstrate your system.

- **10 points:** Task 3 Plans. Neatly drawn datapath schematic for the hardware thread you intend to construct. Include a schematic showing all the components in your datapath, with all connections neatly labeled. Include FSM control and status points. You do not need to have designed the FSM at this point. Show to the TA as you enter the lab — no later than 6:45, no exceptions!

- **25 points:** Basic Pong Game. A basic game that can play a single point. Ball bounces off top and bottom walls as well as paddles. Paddles move properly.

- **10 points:** Full Game. Pong game that can play multiple points. Ball may be served in same direction after every point.

- **10 points:** Scoring Game. Pong game with scoring to 7-segment displays. Ball is served by player who lost the last point.

- **Up to 5 points:** Bonus points for exceptionally polished game. TAs will award for things like: paddles that bounce the ball in different directions based on position of touch point, game clock speeds up as volley progresses, on-screen score, paddle size varies as game progresses, press KEY[1] to go into a slow mode where the game updates every second (useful to check your collision detection), artificial intelligence (AI) opponents, AI on both sides, etc.
  Bonus points are only awarded after all other game points have been earned. Bonus points are precious — getting 5 bonus points will require an extraordinary effort.

- **10 points:** Task 3 Documentation. Neatly drawn and up-to-date STD and datapath schematic. These will be checked during your Task 3 demo.

- **10 points:** Prelab quiz. I've deliberately placed this on the second page as a way to assess which students carefully read the handout before coming to lab. You are only authorized to tell your lab partner about the quiz – tell your friends to "carefully read the handout" instead. I also want to assess team partner communication.
  To get these points, go to Canvas and take the Prelab Quiz. You can find a link on the front page. Your quiz must be completed before 6:00pm **on Tuesday** regardless of which day your lab session meets. Your score will be the sum of your score and your lab parter's score on the quiz.

- **10 points:** Discussion. The TA will ask you and your partner a few questions about datapath design and your design in particular. Be ready to answer the following questions (as well as similar questions). Make sure your lab partner can answer them as well.
  - How many FSMs did your final design (synthesized version) include? List them.
  - What are the limits to the bit-rate variation, such that your circuit will still correctly receive the message? What parts of the design impose these limits?
  - How is designing this circuit as a hardware thread different from attempting the same project as a single FSM?
  - You could have been given this assignment in a programming course and had to have written a C (or Java, or C++, . . .) program. Imagine how you might go about writing such a program for this lab (but don't write it). Explain the fundamental differences in writing the hardware description you wrote and software description you thought about.

  - What performance differences would likely exist between your hardware description and the hypothetical software description.
  - Why does a language like SystemVerilog exist?

- **5 points:** Your code conforms to the SystemVerilog Coding Standards document, published on Canvas.

No lab report is required for this lab. However, I firmly believe that being organized is one of the keys to successful engineering. Therefore, I expect you to have neatly drawn and up-to-date design documents at all times — a STD for your FSM and schematic of your datapath components. We will check these documents at the various demos. However, you should be using these diagrams to design from, code from and communicate with your lab partner and TA. The TAs will ~~probably~~ refuse to help you find bugs if your documents aren't up-to-date.

Late Penalties (the standard drill)

Late demo or code: You will lose 10% of the points for an item (demo of an individual task or for code) for each day late up to a week late. After a week, you will get zero points. When counting days, we will not include any weekend or holiday day on which there were no TA office hours.

A Note about Teams and Collaboration

By now, you should be getting used to working with a random partner, which is a great skill to have as a modern engineer. Continue to be a good teammate, communicate well, and work well.

Once again, all work you turn in for this lab is expected to be that of your team. You may ask other students for general assistance, but you may not copy their work. Likewise, you may not copy work from other sources.

Theory: VGA Output

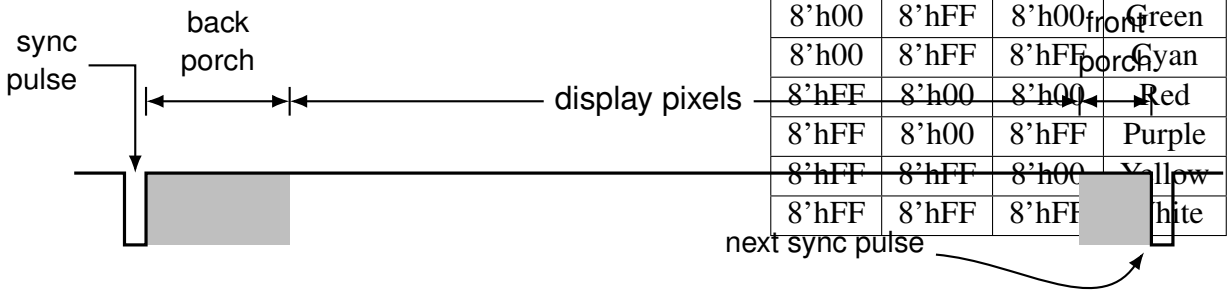
The DE2-115 FPGA boards we use in lab have the capability to output information to a VGA monitor. The VGA monitor displays 480 rows, each of 640 pixels, in a large array. The interface to such a monitor is relatively simple, compared to interfaces to some information arrays that you've learned about recently (memory, perhaps). This interface simply sends the color values for each pixel, one after the other, in a scan of the pixel array. It sends each pixel in a row (left-to-right), followed by the pixels in the next row, from top to bottom. The transmission of a full 640x480 pixels is known as a *frame*. 60 frames are sent each second.

Okay, I'll admit, the interface is a bit more complicated, but not much. The color values for each pixel are 24-bit values, split into 3 color components of 8-bits each. We don't need that much color for this lab, so all 8-bits of each color component can be the same value. The 3 color components indicate if there is any red, green or blue in the final color, according to the scheme shown at right.

The other complicating factor is the synchronization timing. In order to let the monitor know that the entire array is going to be sent, an active-low vertical sync signal is used. Another active-low

synchronization signal, horizontal sync, is used to indicate that a row is ready to be sent. Because these synchronization signals originated in TV and CRT monitors, where an electron beam was magnetically steered across the screen, some additional time was allowed after the synchronization signal before actual display data was sent. This extra time is often referred to as the "back porch" with a corresponding "front porch" after the data is displayed, but before the next synchronization signal is active.

This waveform shows the structure of both the horizontal and the vertial sync signals.

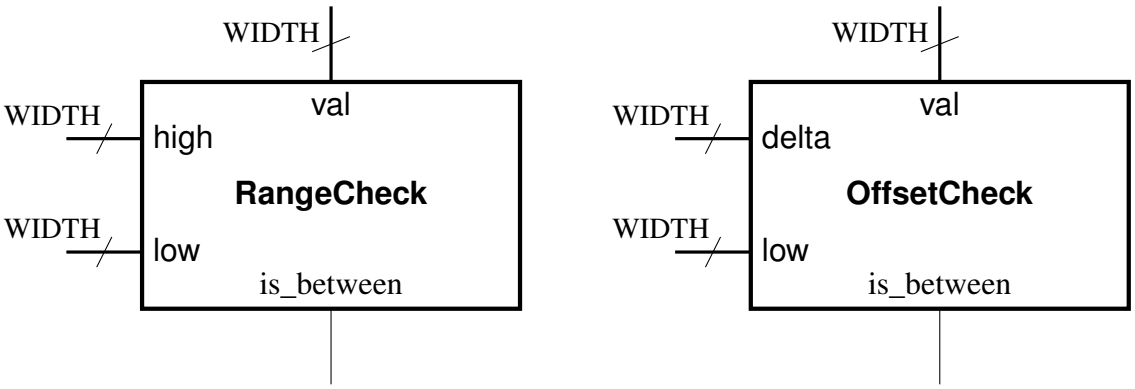


Red	Green	Blue	Result
8'h00	8'h00	8'h00	Black
8'h00	8'h00	8'hFF	Blue
8'h00	8'hFF	8'h00	Green
8'h00	8'hFF	8'hFF	Cyan
8'hFF	8'h00	8'h00	Red
8'hFF	8'h00	8'hFF	Purple
8'hFF	8'hFF	8'h00	Yellow
8'hFF	8'hFF	8'hFF	White

Your Design Problem

Task 1: Datapath Components

You already have some very useful components that you've built as your **`library.sv`** file. Add the following two components to it: **`RangeCheck`** and **`OffsetCheck`**. Make sure to build quality testbenches to ensure they work as intended.



The **`RangeCheck`** module is a souped-up comparator. It checks to see if the input **`val`** lies between the inputs **`low`** and **`high`** (inclusive). The output **`is_between`** is a single-bit value, whereas all the inputs should be of paramaterized **`WIDTH`**.

The **OffsetCheck** module builds on the **RangeCheck**. In this case, the **high** value is replaced by a **delta**, which is an offset from the **low** value. In other words, this module can be built by using an adder to add **low** to **delta** and feeding the sum into a **RangeCheck** module as the **high** input. The output **is_between** is active whenever **val** is greater than or equal to **low** and also less than or equal to **low + delta**.

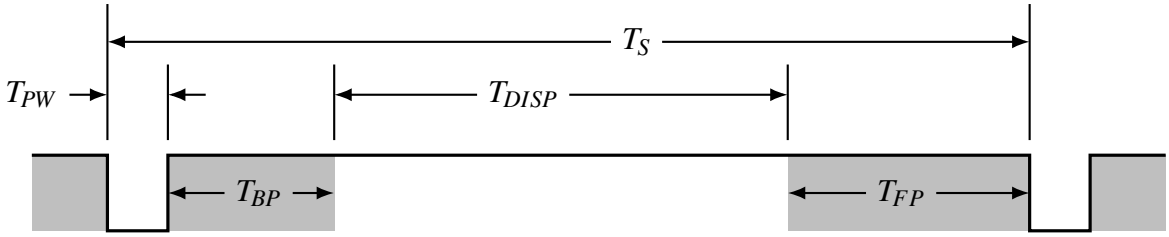
Task 2: Building a VGA Interface

For the next part of the lab, you will build a circuit that generates the VGA timing signals. The module specification is:

```
module vga
  (input  logic CLOCK_50, reset,
   output logic HS, VS, blank,
   output logic [8:0] row,
   output logic [9:0] col);
```

In Lab3, you used **CLOCK_50** to drive your game on the FPGA. At that point, it was only important that it was a fast clock. For the VGA module, the frequency specification is important, as you'll be generating outputs that have to occur with particular timing requirements. Should someone hook up your VGA module to a clock of a different frequency, then the output values would happen at the wrong time.

Your module will output a horizontal and vertical sync signals, **HS** and **VS**. Here are some critical timing parameters of the **HS** signal. Note that the "clocks" column assumes a 50MHz clock.



Symbol	Parameter	Time	Clocks
T_S	Sync pulse time	32 μs	1600
T_{DISP}	Display time	25.6 μs	1280
T_{PW}	Pulse width	3.84 μs	192
T_{FP}	Front porch time	640 ns	32
T_{BP}	Back porch time	1.92 μs	96

During the T_{DISP} time, the 640 pixels of the row will be displayed, which means each pixel will be displayed for 2 clock periods. The pixels are displayed from left to right. At the start of the T_{DISP} time, the **col** output of the **vga** module will be **10'b0**. It will increment every 2 clock periods, such that it will be **10'd639** at the end of the T_{DISP} period. The **col** output is a don't care during all other times – that is, all times except during T_{DISP}.

Every 32 μs, an entire row is displayed, followed by the row below it. The timing for the **VS** signal is shown below. The waveform looks just like that for the **HS** signal above.

Symbol	Parameter	Time	Clocks	Lines
T_S	Sync pulse time	16.7 ms	833,600	521
T_{DISP}	Display time	15.36 ms	768,000	480
T_{PW}	Pulse width	64 μs	3,200	2
T_{FP}	Front porch time	320 μs	16,000	10
T_{BP}	Back porch time	928 μs	46,400	29

Note that during a single **VS** period (i.e. every 16.7 ms), the **HS** sequence discussed above will occur 521 times. 480 of those times will occur during the **VS** display time. The **row** output of the vga module will count from **10'd0** to **10'd479** during the **VS** display time. We don't care what the row value is except during the **HS** display time.

The **blank** signal is a necessary output to the VGA circuitry. It is an active high signal that indicates either the row or column timing is not currently displaying pixels. In other words, during the front porch, back porch or pulse timing of either **VS** or **HS**, **blank** should be active.

The vga module is not overly complex if you think of it in terms of component devices. It should not have any **always_ff** blocks of its own – rather it should instantiate combinational and sequential components (which might have **always_ff** blocks inside them). Once you've built the vga module, it will maintain all your timing for you. At any point in time, you will know the x, y coordinates of the pixel currently being displayed (the value of **col** and **row**).

If you were to pass **col** and **row** into a bit of combinational circuitry, you could easily create a test pattern for display. Imagine dividing the 640 columns into 8, so that each color combination was displayed in one column. Your combinational color generator would output a **8'hFF** on the **red** signal whenever **col** was between **10'd320** and **10'd639**. Hmm. I wonder what component you might use to determine if **col** was between those values . . . . The **green** signal would be **8'hFF** whenever **col** was in the range **10'd160** to **10'd319** or in the range **10'd480** to **10'd639**. I'll let you figure out **blue** on your own.

There is one additional complication to the test pattern. In order to check that your vertical synchronization is correct, you need something different about the pattern from top to bottom. Otherwise, you wouldn't see it if your pattern was rolling up the screen between frames. So, make the bottom half of your test pattern entirely black. To do so, just check **row**. If it is between **10'd240** and **10'd479**, then output **8'h00** for **red**, **green**, and **blue** signals.

The corresponding **ChipInterface** module for your lab would have to include the several VGA signals, so it might look like this:

```
module ChipInterface
  (input  logic CLOCK_50,
   input  logic  [3:0] KEY,
   input  logic [17:0] SW,
   output logic  [6:0] HEX0, HEX1, HEX2, HEX3,
                       HEX4, HEX5, HEX6, HEX7,
   output logic  [7:0] VGA_R, VGA_G, VGA_B,
   output logic        VGA_BLANK_N, VGA_CLK, VGA_SYNC_N,
   output logic        VGA_VS, VGA_HS);
```

The **VGA_R**, **VGA_G**, **VGA_B** signals are simply the **red**, **green**, and **blue** color channels from the combinational circuitry in your test pattern generator. **VGA_VS** and **VGA_HS** are your vertical and horizontal sync signals. **VGA_BLANK_N** is an active low signal that is the same as your active high **blank** signal. **VGA_SYNC_N** should always be active for non-CRT monitors (like the ones in the lab). **VGA_CLK** is used to latch your color values. I recommend connecting it to the inverse of **CLOCK_50**.

The monitors in the lab have a VGA input (and a VGA cable attached) that you can use to display the output of your FPGA board. You might find the Picture-in-Picture feature of the monitor to be of use — ask your TA to demonstrate how to use it.

For Credit: Demonstrate your system (i.e. a **vga** module that outputs a test pattern on the top-half of the screen) to the TA before the end of the first lab session.

Task 3: Pong

A long, long, long time ago, there was a video game called Pong. It was the first game ever released by Atari and the first video game to get any sort of main-stream popularity. It is a 2-dimensional sports game that simulates tennis. Each player manipulates a "paddle," one on the left of the screen and the other on the right. A ball bounces back and forth between the two paddles. If a player misses the ball, the other player scores a point. See `en.wikipedia.org/wiki/Pong` for more information. It is interesting to note that Pong was implemented with digital logic, not a microprocessor. You can find a schematic for the original circuit at `atarihq.com/danb/files/PongSchematic.pdf`. You can also find a nice history (with good illustrations, but many ads) at `www.buzzfeed.com/chrisstokelwalker/atari-teenage-riot-the-inside-story-of-pong-and-t`[1].

[1] Yes, this URL looks like it has been truncated improperly. No worries, it is correct.

Your team will design and implement a hardware thread to play the Pong video game. Be careful to ensure you are designing a datapath using the digital components you have learned about. It may be too tempting to get into "programming" mode and hack together a big always_ff block of procedural code. Such an approach is not going to score points with the TAs, nor get you done any faster.[2]

The display mechanism is a bit different for this lab than is often used in modern computers. Those of you who have some knowledge about computer design might know that most computer displays are represented in memory by a large array of memory bits, with several bytes per pixel. Each pixel's bytes hold a color code. The program works by filling memory with the color codes for the picture it wishes to have displayed. A device called a frame buffer holds the display memory and transmits the contents to the monitor for display. Well, that approach is a fairly general one that won't work very well for this lab.

Instead of building a frame buffer, you are going to design digital circuits that output the appropriate color signals (**`red`**, **`green`**, and **`blue`**) in response to the **`row`** and **`col`** counters coming from the **`vga`** module, much like you did with the test pattern generator. I would recommend that you build a module for the ball and another for each paddle.[3] Each module can emit a signal when **`row`** and **`col`** match the position of their object (ball, left paddle, etc). A color module can take these signals and decide what the **`red`**, **`green`**, and **`blue`** values should be (i.e. **`red`** will be a **`8'hFF`** when the ball matches (along with **`green`** and **`blue`** to make white) or left paddle matches (along with **`green`** to make yellow)).

The objects on the screen can change position only once the entire screen has been displayed. You might be tempted to divide the CLOCK_50 signal's frequency to get a 60Hz clock. Doing so, however, would mean you are putting logic on the clock line, a well-known no-no. Instead, run the CLOCK_50 to all your registers, but only enable them to be updated once per screen draw (for instance, only when **`row == 480`** and **`col == 640`**). I'll refer to this as the game state update period.

Here are the game requirements:

Ball:

The ball is a 4-pixel by 4-pixel square, that is colored white. Its position is updated each game state update period. Upon **`reset`**, the ball is placed in the middle of the court (the screen). It will, when the serve button (**`KEY[3]`**) is pressed, have an initial velocity down and towards the right.

Once each game state update period, the ball moves 2 pixels in a horizontal direction and 1 pixel in a vertical direction. If the top of the ball is in the top row, it will "bounce" and instead of moving 1 pixel up with each update, it will start moving 1 pixel down. Likewise, if the bottom of the ball enters the bottom **`row`** (i.e. **`row == 10'd479`**), then it will bounce and start moving 1 pixel up with each update.

[2] Nor, likely to work at all.

[3] Paddle is easier, start there.

Likewise, the ball shall bounce off the front of the player's paddles. It will do so by changing the amount of horizontal addition from +2 to -2 (or vis. versa) in each update.

Paddle:

Each paddle is 4 pixels wide by 48 pixels tall. The left paddle remains in columns **`10'd60`** to **`10'd63`**. The right paddle remains in columns **`10'd577`** to **`10'd580`**. The left paddle is yellow and the right paddle is cyan.

Each paddle moves up or down based on input from the players. The left player uses **`SW[17]`** to move the paddle up or down and **`SW[16]`** to specify whether it should move (when "up") or not (when "down"). The right player uses **`SW[1]`** to move/not move and **`SW[0]`** to choose the direction.

The paddle does not "wrap" from top to bottom. Instead, it stops if attempting to move up beyond the top of the screen or down beyond the bottom. In each game state update period, the paddle moves up or down by 5 pixels.

Score:

If the ball enters column 0, then the right player earns a point. If the ball enters column 639, then the left player earns a point. When this happens, an on-screen indicator should show that a point has been scored (Use your imagination to design the indicator. It can be as simple as flashing part of the screen a particular color, or more a more complex arrow pointing toward the ball, etc). The total score must be shown on the 7-segment displays, but may also be shown on the screen. When a point is scored, the ball is returned to the center of the screen and will begin moving when the serve button (**`KEY[3]`**) is pressed. The ball should start moving away (and down) from the player who scored the point.

Other inputs:

`KEY[0]` is a reset button.

Implementation Hints

- Remember that the KEY inputs are active when not pushed.
- The VGA module sounds complicated, but it is basically a set of counters with some comparisons (comparator or RangeCheck components) to generate all the outputs.
- Hacking a huge **`always_ff`** is a sure way to complexity and madness. Use the components you've been taught.
- Debugging a VGA module that doesn't output anything can be frustrating because you aren't getting much feedback on what you're doing wrong. Simulation can be your friend here. You can simply use a testbench with a clock flipping every #10 (i.e. period is #20) and have your testbench look like:

```
initial begin
  #40000000 $finish;
end
```

  Then, use the GUI's waveform viewer to examine an entire vertical sync period.
- The TAs might be able to show you how to use SignalTap, a Quartus tool that puts a logic analyzer on your FPGA. Using this tool, you can get real-time feedback as to how signals internal to the chip are behaving. It's kind of like a bunch of oscilloscopes hooked inside the chip. Using it well is somewhat complicated, though.
- The monitors, when displaying VGA full screen, put an extra black bar at the top and bottom of the screen. I found it useful to draw a faint grey line (**`color = 24'h101010`**) in row 0 and 479 (and column 0 and 639) so I knew the boundaries of the pong court.
- Make sure to read the entire lab handout. Each and every paragraph, all the way to the end. Make sure your lab partner has done the same.