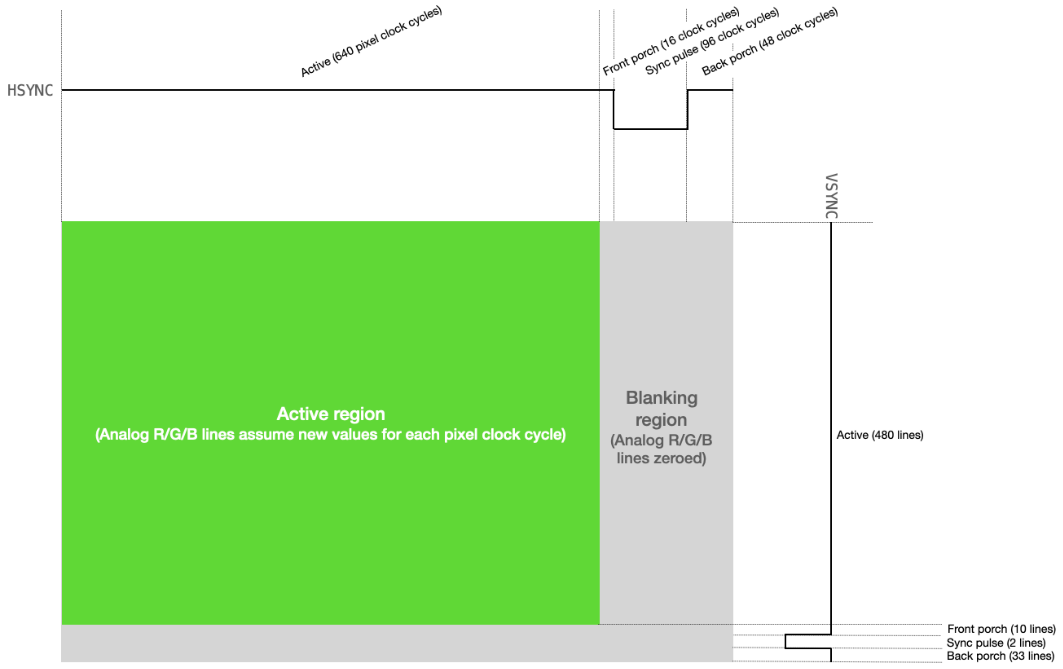
LAB 8 – 3300L

Spring 2025 / Dr. Van Blerkom

This lab will introduce how to display images on the VGA port on the Nexys A7. You will be provided some code that partially implements a simple ball bouncing around the screen. You will need to finish the implementation of the *ball* module. You will also need to implement a larger size ball, by modifying the *ball* module. Finally, you need to add the larger ball to the screen, and add additional partial walls in the middle of the field for the balls to bounce off of.

On the Nexys A7 board, the VGA port follows the standard analog VGA signaling, providing separate red, green, and blue outputs (4 bits each for 12-bit color) along with horizontal (*hsync*) and vertical (*vsync*) sync signals. A typical resolution configuration is 640×480 at 60 Hz, requiring a 25 MHz pixel clock to step through each of the 640 active pixels per line and each of the 480 active lines per frame. To create the 25 MHz pixel clock, we use a 2-bit counter from the 100 MHz system clock, and output a *pixpulse* every fourth system clock. Horizontal and vertical blanking intervals—consisting of a front porch, sync pulse (active low), and back porch—occur between displayed lines and frames, ensuring the monitor's scanning beam resets properly. During active display time, the Nexys A7 drives valid RGB pixel data onto its VGA outputs, and drives *hsync/vsync* low during their respective sync intervals.


Active (640 pixel clock cycles)
Front porch (16 clock cycles)
Sync pulse (96 clock cycles)
Back porch (48 clock cycles)
HSYNC
VSYNC
Active region
(Analog R/G/B lines assume new values for each pixel clock cycle)
Blanking region
(Analog R/G/B lines zeroed)
Active (480 lines)
Front porch (10 lines)
Sync pulse (2 lines)
Back porch (33 lines)

The module *vga_timing* creates two counters, *hcount* and *vcount*, which point to the X and Y location of the current pixel. This module also creates the *hsync* and *vsync* signals for the display.

The module *ball* implements a bouncing ball; the ball can be moving in one of four directions, determined by the xdir and ydir registers:

xdir	ydir	Movement direction
0	0	Left and up
0	1	Left and down
1	0	Right and up
1	1	Right and down

The location of the ball is tracked in the registers *xloc* and *yloc*. The *ball* module is constantly "listening" to the *hcount* and *vcount* values from the *vga_timing* module, and when those counters indicate that the current pixel is inside the ball region, the *ball* module will set the *draw_ball* output to indicate that the pixel color should change to draw the ball.

The *ball* module also keeps track of the immediate neighboring pixels of the ball, to figure out if the ball needs to change direction due to bouncing off a barrier (or another ball). It does this by accepting an input *empty*, which will go low when the current pixel is not empty – if a non-empty pixel is right next to the ball, then the location of that neighboring pixel will be recorded in the *occupied_top, occupied_bot, occupied_lft*, and *occupied_rgt* registers, as described below for the 3x3 ball:

Occupied_top[4] Occupied_lft[4]	Occupied_top[3]	Occupied_top[2]	Occupied_top[1]	Occupied_top[0] Occupied_rgt[3]
Occupied_lft[3]	*Ball pixel*	*Ball pixel*	*Ball pixel*	Occupied_rgt[3]
Occupied_lft[2]	*Ball pixel*	*Ball pixel*	*Ball pixel*	Occupied_rgt[2]
Occupied_lft[1]	*Ball pixel*	*Ball pixel*	*Ball pixel*	Occupied_rgt[1]
Occupied_lft[0] Occupied_bot[4]	Occupied_bot[3]	Occupied_bot[2]	Occupied_bot[1]	Occupied_rgt[0] Occupied_bot[0]

Finally, the *ball* module takes an input *move*, which is a pulse for one pixel time that tells the ball to move to the next location. The move pulse happens at the beginning of the vertical blanking time, so that the next frame of pixels will show the ball in the new location. If moving to the next location would cause the ball to overlap a non-empty pixel, then the ball will change direction and bounce off the obstruction.

The code you are provided only figures out the movement for xdir=0 and ydir=0. For the lab, you need to:

1) Add the code for the other directions to make the ball bounce correctly off each wall.
2) Implement a module *bigball*, which is a copy of *ball*, but modified to make a 21x21 pixel ball.
3) Add *bigball* to the screen so that it interacts with the small *ball*.
4) Finally, you need to add additional partial walls in the center of the screen for the balls to bounce off. The exact sizes and locations are up to you.
5) You can add more balls to the screen if you want (to the limits of the FPGA, that is…) – make sure you start them in different locations.

Optional – make the *bigball* appear as something more interesting than a square…