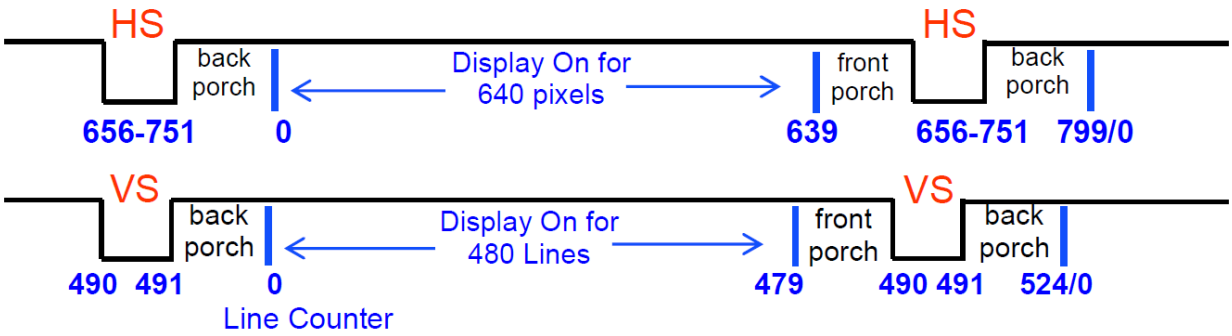
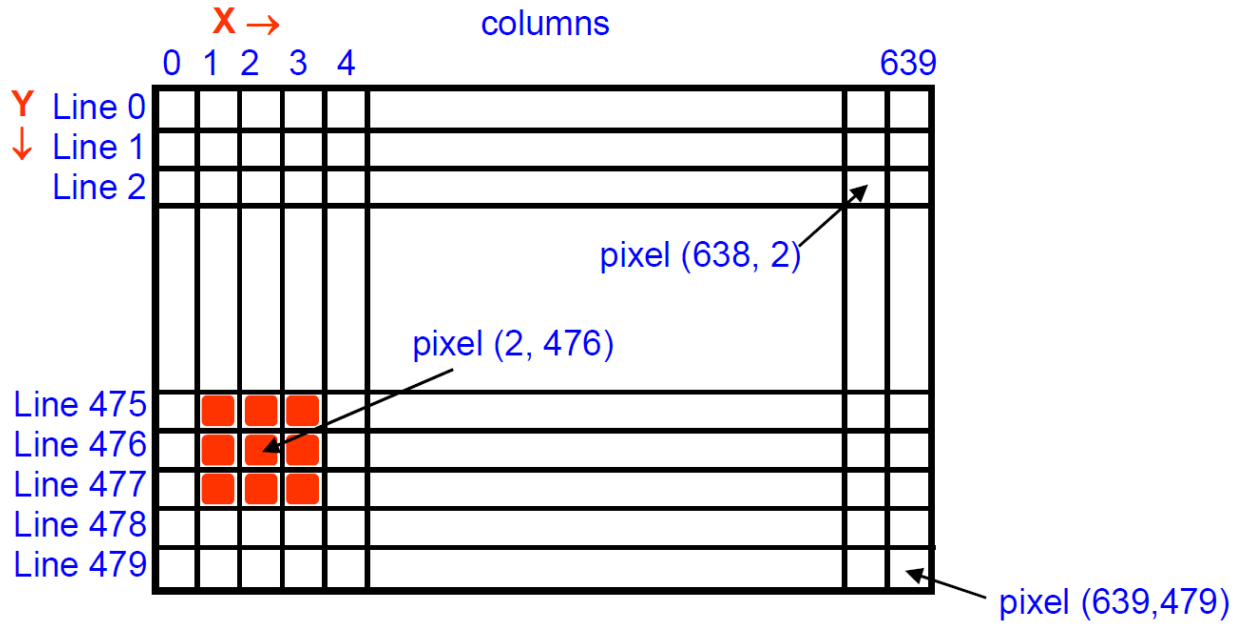
A Primer on Designing Graphics in HDL

The following entities are provided to you:

VGA_controller: This produces timing signals Horizontal Sync (hs) and Vertical Sync (vs) required by any VGA monitor. The timing follows exactly as in the DE2 board manual and the counters follow the outline on Lecture slides of Experiment 7.



Imagine that the "electron beam" is scanning horizontally from top to bottom. By keeping track of the beam position in our counters, we know the exact coordinates of the beam at each pixel time tick. The beam coordinates are labeled - BeamX and BeamY. Recall that the upper-left corner coordinates x,y are (0,0) and y coordinate increases downward on the screen.



The Beam coordinates are available to any drawing and coloring entities. You will never need to modify the **VGA_controller** for future designs!

Color_Mapper: This performs object or shape rendering and coloring - This entity puts out RGB signals to the monitor. It receives the center coordinates of the Ball - namely BallX and BallY. For

example, the red box above has center coordinates (2, 476). When the Beam crosses the box, i.e. the beam coordinates are within (Center + Box Size) and (Center - Box Size), you set (R, G, B) to (1023, 0, 0). If you want a blue background, set (R, G, B) to (0, 0, 1023) outside the box. Similarly, other objects can be rendered by knowing its outer boundary. For example, to draw a round ball we can use the equation for a circle around center (x_0, y_0) with radius r as given by $(x-x_0)^2+(y-y_0)^2 = r^2$. When the beam coordinates (x, y) coincide with a point within the circle, we output the color of the ball for that pixel. When the beam coordinates (x, y) are outside the ball, we output the background color. More complicated functions might manipulate the coordinate values themselves to generate color gradients.

This entity is combinational logic with no timing signals involved. It simply computes RGB values given the current beam position. In this entity, BallX and BallY are the center coordinates of the Ball; BeamX and BeamY are the current beam coordinates. For Experiment 7, this entity needs no modification.

Ball: This entity computes the position of the ball in every frame. One way to keep track of frames is simply by keeping track of tile Vertical Sync (vs) signal. When the vs signal goes from low to high, we can assume the start of a new frame. By changing the position of any object with anchor coordinates (x,y) to (x+StepX. y+StepY) in the succeeding time frames, we create a motion of the object moving in tile direction with an angle arctan(-StepY/ StepX). The boundary conditions are:

StepX = 0 creates vertical motion of size StepY. For positive steps, the motion is downward on the screen and reverse for negative steps.

StepY = 0 creates horizontal motion of size StepX. For positive steps the motion is left to right on the screen and reverse for negative steps.

|StepX| = |StepY| creates 45° diagonal motion. The signs of the steps determine one of four directions: towards lower-right, lower-left, upper-right, or upper-left corners.

The smallest step size is one pixel. Increasing the step size increases speed of the ball. To reduce the speed below one pixel per frame, one can update the position e.g. every two frames to halve the speed of the ball, etc.

Of the given code, this is the only behavioral entity that you need to edit for Experiment 7; your edited version must recognize the commands from the keyboard and change the direction of the ball to up, down, left, or right (not diagonal).

BouncingBall: This entity is the outermost entity that brings the above three entities together and connects the ports to the outside. This entity needs to be modified for Experiment 7 to include your new entities to communicate with the keyboard.