ECE 4514 Fall 2019: Homework 7

Assignment posted on March 28
Assignment due on April 4 11:59 PM

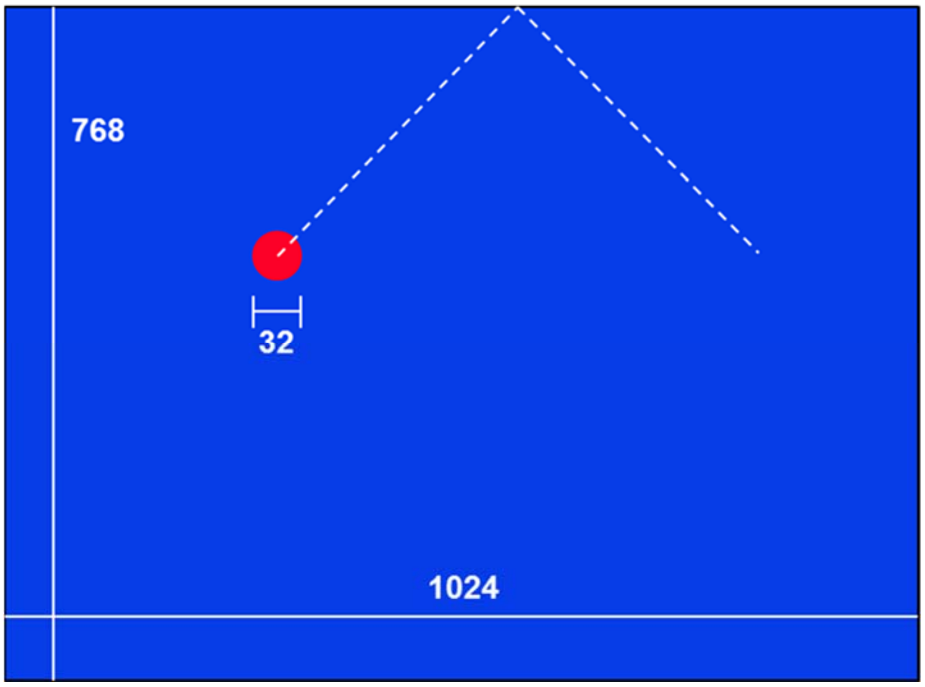
Homework weight: 100 points

In this homework, we continue experimenting with the VGA interface on the DE1-SoC kit. You will receive a solution for the previous assignment, in case you had trouble designing the XGA sync generator. Homework 7 will create and display a moving object on the screen (a bouncing ball). You will design the hardware in such a way that you generate the pixels of the object as the screen pixel counters move through the object.

As with the last homework, this assignment will explain the design specification, but it will reveal little on the actual design steps (writing Verilog, creating a project, mapping to FPGA, downloading the bitstream). It is up to you, as a designer, to translate the following specifications into a working project.

Design Specification

The design specification is illustrated in the figure below. The bouncing-ball application displays a red ball on a blue background. The canvas measures 1024 pixels by 768 pixels, also known as XGA resolution. The canvas is redrawn 70 times per second, which results in a pixel rate of 75 million pixels per second. This means that the FPGA implementation needs to generate pixel data for the Video DAC at a rate of 75 million pixels per second.



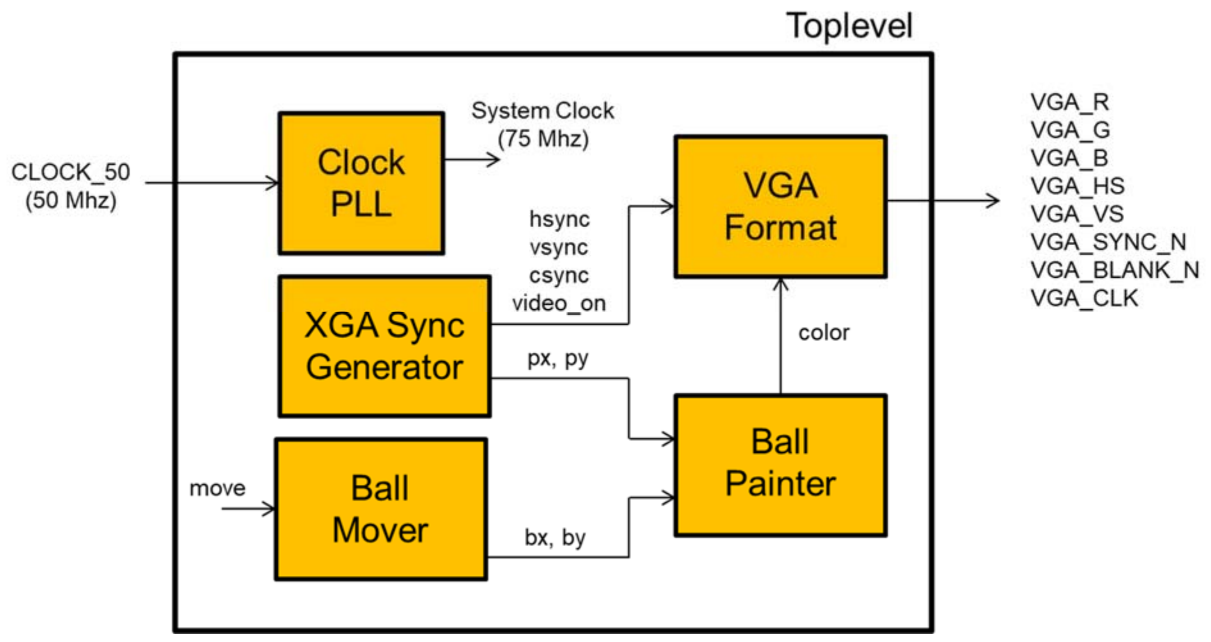
The ball moves 70 steps per second, and each step is taken diagonally in four possible directions: up-and-right, up-and-left, down-and-right, down-and-left. The length of each step is one pixel in the X (horizontal) direction and one pixel in the Y (vertical direction). The ball movement speed coincides with the canvas redraw rate, therefore, control of the ball speed is quite easy: one step per frame. The ball measures 32 pixels in diameter and is obtained by quantizing a circle on a grid.

When the ball reaches the edge of the canvas, it bounces and reverses the moving direction so that the ball stays on the screen. For example, when the ball movement was up-and-right, then bouncing off the top-edge will change the ball movement to down-and-right. The initial position of the ball is at the center of the screen, and the initial movement vector points bottom-right.

The entire design must be made in Verilog, using a meaningful structural decomposition of the design in various sub-modules. Since the pixel rate of this design is faster than the standard system clock, you will again make use of a PLL module, created using the IP generator.

Implementation Guidelines: System Architecture

The figure below gives a basic outline of a possible design. You are not required to follow this outline; it's only provided to point out how to address some of the apparent difficulties in this assignment, such as the drawing of a circle on a canvas, moving it around, detecting bounces, and so forth.



This system architecture includes 5 modules.

1. The **Clock PLL** generates a 75MHz system clock.
2. The **XGA Sync generator** generates horizontal and vertical synchronization signals. This is basically a specialized counter, that scans across all the pixel coordinates on the screen, from left to right and from top to bottom. The current pixel position *(px, py)* is broadcasted to other modules in the system. Other output signals include the horizontal synchronization *hsync*, the vertical synchronization *vsync*, the composite synchronization *csync*, and a marker that shows when the current pixel is inside of the visible data area (*video_on*).
3. The **Ball Mover** bounces a ball over the screen by computing the coordinates of the ball location. The ball mover abstracts the ball into the coordinates of a single centerpoint *bx, by*. Each time this module sees a *move* pulse, it will move the ball one step in the current direction.

The ball mover is responsible for implementing the bouncing logic. The *move* pulse is created by the top-level such that no more than a single pulse per video frame period is created.

4. The **Ball Painter** determines if the current video pixel is part of the ball or not. The ball painter does this by comparing the current pixel position *px, py* with the position of the ball center *bx, by*. The current pixel will be *inside* of the ball if *((px-bx)*(px-bx) + (py-by)*(py-by)) < 32*32*.

   However, instead of implementing this test using arithmetic, it may be more convenient to use a lookup table which stores a ball template. The lookup table is addressed using the relative offset *(px-bx), (py-bx)* to determine if the current pixel position is within the ball area.

   Here is an example of how such a lookup table looks like. The 'X' positions indicate where the ball shape is opaque, the '.' positions indicate where the ball is transparent. You could encode this with '1' and '0', or an equivalent suitable representation such as ball color. Then, each entry of the table is addressed as *(px-bx) + (py-by)*ball_shape_width*, with ball_shape_width indicating the number of pixels used to encode the shape line width. Of course, the addressing logic should take care that out-of-range addresses are properly handled by the ball painter.

```
..................................
............XXXXXXXXX.............
........XXXXXXXXXXXXXXXXX.........
.......XXXXXXXXXXXXXXXXXXX........
.....XXXXXXXXXXXXXXXXXXXXXXX......
.....XXXXXXXXXXXXXXXXXXXXXXX......
....XXXXXXXXXXXXXXXXXXXXXXXXX.....
....XXXXXXXXXXXXXXXXXXXXXXXXX.....
....XXXXXXXXXXXXXXXXXXXXXXXXX.....
.....XXXXXXXXXXXXXXXXXXXXXXX......
.....XXXXXXXXXXXXXXXXXXXXXXX......
.......XXXXXXXXXXXXXXXXXXX........
........XXXXXXXXXXXXXXXXX.........
............XXXXXXXXX.............
..................................
```

5. The **VGA Formatter** combines inputs from the sync generator and the ball painter to create a set of signals compatible with a VGA connector.

Implementation Guidelines: 75 MHz Clock Generator

You will need the same PLL of 75MHz as the one you've used for Homework 6. As a reminder, the steps for creating the PLL module are repeated below:

The module that you need to create a 75MHz clock out of a 50MHz clock is called ALTPLL, and it is created as follows in Quartus.

1. In Quartus, select Tools – IP Catalog.

2. In the IP Catalog, find and select the ALTPLL module (under Clocks, PLLs and Resets). *Note – Intel requires Altera to rename everything as 'Intel'. The newer versions of Quartus (18.x) call this module Intel PLL.*
3. In the dialog, select an output filename for the module that will be created. For example, call it clockgen.v.
4. The following dialog is specific to the PLL module and provides a wealth of options. You need to change only two things. First, select the input clock to be 50MHz. Second, select the output clock to be 75MHz. All the other options can be left at their default value. Press finish.
5. You can then add the PLL to your design project. Note that the top-level of the PLL is quite simple. Open the file clockgen.v in a text editor to inspect it. The input clock signal is inclk0 and should be connected to the 50MHz clock available on the DE1-SoC kit. The output clock signal is c0 and should be used as your system clock. The reset port can be tied to ground, and you can connect the locked output to a LED to verify that the PLL generates a reliable 75MHz clock signal.

```
module clockgen (
     input  wire  refclk,   //  refclk.clk
     input  wire  rst,      //  reset.reset
     output wire  outclk_0, //  outclk0.clk
     output wire  locked    //  locked.export
);
```

What to turn in

Download the repository for Homework 7. You will find a subdirectory called homework6, and a subdirectory XGA graphics. Homework6 contains the solution to homework6. XGA graphics is the design that you need to create, making use of multiple Verilog modules and IP Cores as you see fit.

Before you begin, contemplate the system architecture. Try to generate a static ball first and modify it to a dynamic ball afterwards.

When you are finished, push your repository back to github. Make sure to include all Verilog files as well as IP core module files: you should push the files with extensions qip, sip as well as the folders generated by IP generator.

Good luck!