ECE 4514 Fall 2019: Homework 8

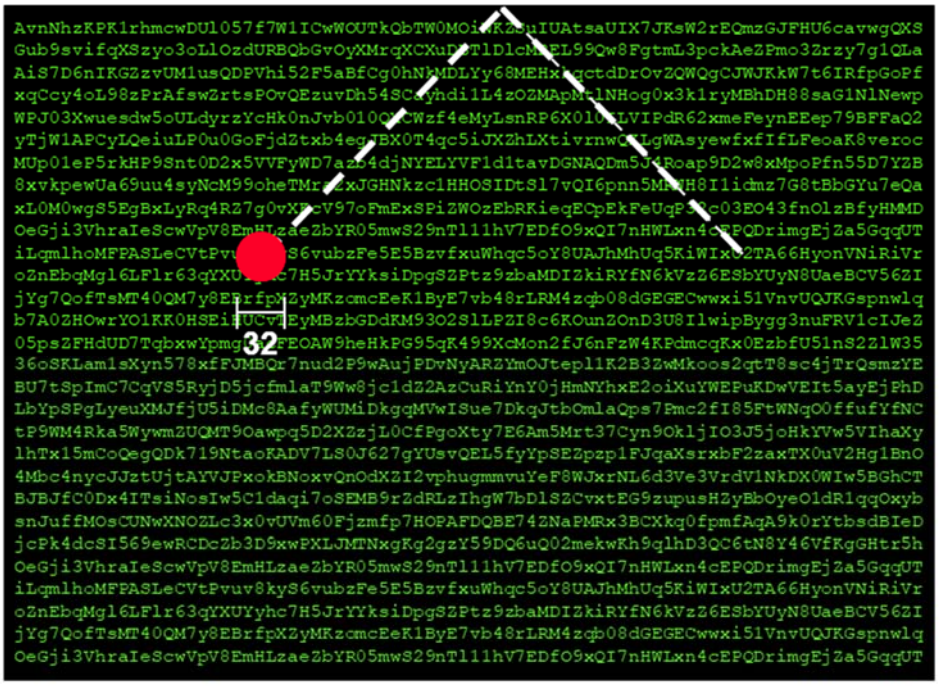
Assignment posted on April 6
Assignment due on April 13 11:59 PM

Homework weight: 100 points

In this homework, we add a third component to the VGA interface on the DE1-SoC kit. As in previous homework, you will receive a solution for homework 7. In homework 8, we will add the capability to display text (ASCII) to the interface. We will also introduce a screen buffer to hold the ASCII codes to be displayed on the screen.

Design Specification

The design specification is illustrated in the figure below. The bouncing ball of homework 7 is back, but this time the background is a display of green characters on a black background. The canvas measures 1024 by 768 pixels. The characters measure 8 pixels (horizontal) by 16 pixels (vertical), so that there fit 128 characters per line, and the display has 48 lines.



The bouncing pattern of the ball is identical to that of homework 7. In fact, you will be able to reuse the solution of homework 7 to build this homework.

The design is operated using four keys, KEY[0] to KEY[3]. Each of these keys has the following function.

- Pressing KEY[0] resets the design. At reset, the display comes up blank and the ball is invisible.
- Pressing KEY[1] fills the screen buffer with random characters. The characters remain static.
- Pressing KEY[2] clears the screen buffer to all blank characters. The characters remain static.
- Pressing KEY[3] toggles the visibility of the bouncing ball. The bouncing ball must move over the characters.

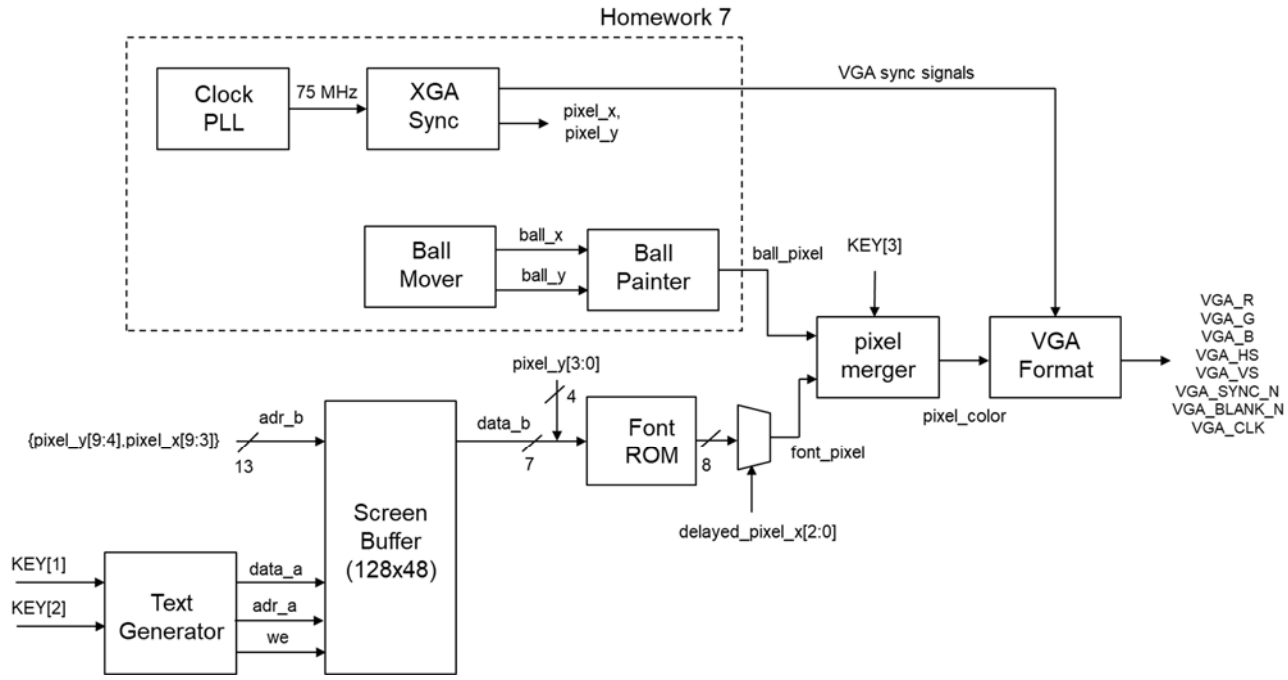
The entire design must be made in Verilog, using a meaningful structural decomposition of the design in various sub-modules. Since the pixel rate of this design is faster than the standard system clock, you will again make use of a PLL module, created using the IP generator.

Implementation Guidelines: System Architecture

To help you start with this design, you receive the following components:

- A sample implementation for homework 7, the bouncing-ball design
- A module that operates as a dual-port ram design, which will serve as screen buffer
- A font ROM, which contains bitmaps for 128 different ASCII characters
- A KEY debouncer module, which can be used to debounce the key presses
- A XGA sync generator, which generates the scanline clock with horizontal and vertical synchronization signals.

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.



The upper part of the design corresponds to the solution for Homework 7. There's a clock PLL and a XGA sync generator which generates pixel_x and pixel_y scan coordinates. There's a Ball Mover and a Ball Painter module, which decides when pixel_x and pixel_y should address a red ball pixel or else a background pixel. In Homework 7, the background was uniformly blue. In this Homework, the background is a text display with 128 columns and 48 rows.

The screen buffer is a dual port memory that holds the ascii code of the characters in each screen position. There are 128*48 = 6144 character positions. These character positions can be expressed in terms of pixel_x and pixel_y, since each character is 8 pixels wide and 16 pixels tall. Hence, pixel_y[9:4]

indicates the row, and pixel_x[9:3] indicates the column in the screen buffer. The characters stored in the screen buffer are 7 bits, and they are fed into a font ROM. The font ROM is a lookup table that holds the pixel data for each character. For example, here is the font data for the 'A' character:

```
// font rom address : font rom data
11'h410: data = 8'b00000000; //
11'h411: data = 8'b00000000; //
11'h412: data = 8'b00010000; //    *
11'h413: data = 8'b00111000; //   ***
11'h414: data = 8'b01101100; //  ** **
11'h415: data = 8'b11000110; // **   **
11'h416: data = 8'b11000110; // **   **
11'h417: data = 8'b11111110; // *******
11'h418: data = 8'b11000110; // **   **
11'h419: data = 8'b11000110; // **   **
11'h41a: data = 8'b11000110; // **   **
11'h41b: data = 8'b11000110; // **   **
11'h41c: data = 8'b00000000; //
11'h41d: data = 8'b00000000; //
11'h41e: data = 8'b00000000; //
11'h41f: data = 8'b00000000; //
```

Using the lower bits of pixel_x and pixel_y, we can then select one single pixel from a character to display on the screen. The font_pixel is merged with the ball_pixel to determine the final pixel color:

Font pixel	**Ball pixel**	**Pixel color**
0	0	Black
1	0	Green
0	1	Red
1	1	Red (ball over text)

The contents of the screen is defined by the Text Generator module. Depending on the key presses, the screen buffer will be filled with all blanks, or with random character data. You are free to decide how to create your 'random' numbers. A very simple pseudorandom generation algorithm is a linear congruential generator, which generates a pseudorandom sequence X(n) from X(0)=0:

$$X(n+1) = (1{,}664{,}525 \cdot X(n) + 1{,}013{,}904{,}223) \bmod (2^{32})$$

There are many other solutions possible, as long as your display text 'looks' random.

What to turn in

Download the repository for Homework 8. You will find a subdirectory called homework7, and a subdirectory XGA_Text. You will also find sample code for the following modules that will help you complete this homework:

- dual_port_ram_sync.v: dual-port screen buffer module
- font_rom.v: font rom data

- keypressed.v: debouncer module for keys
- vesasync.v: XGA sync generator

You have to complete the XGA_Text design. Add new files to that Quartus project as you build up the complete solution. A quick start may be to recreate the homework 7 solution inside of XGA_Text, and then work to add the screen buffer, font rom and text generator into that design.

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!