Final Project: ES4 Spring 2021

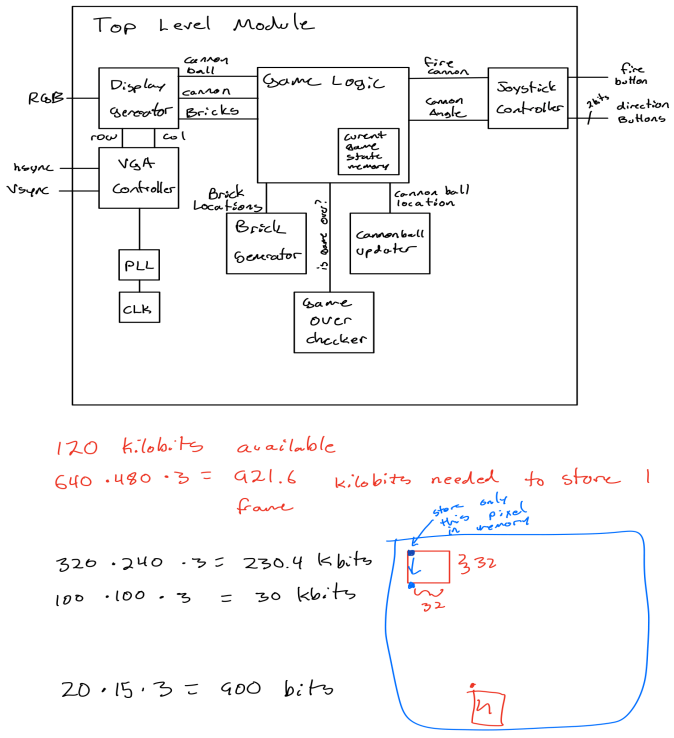
Ibrahima Barry, WIlly Lin

Zach Osman, James Eidson

ECE Tufts University

5/11/2021

1 Overview



This game is a variation of the game brickBreaker where the player fires a cannonball out of a cannon and works to destroy the bricks that are displayed on the screen. There

are three main parts to this project: displays, controls, and game logic. 'Displays' were done using a VGA, 'controls' were done by using an NES gamepad, and 'game logic' done in the 'top' module of our project. VGA sends the current row and column to display on the screen. The NES gamepad takes data in the form of buttons pressed and stores it in a shift-register. A clock is driven for 8 cycles (the number of buttons is 8) and the data signal is synchronized with the clock. When a button is pressed the corresponding bit of the output of the register is set to low. The top module controls the general logic of the game such as drawing the bricks and the cannnon. Top also controls the memory usage of the game. Overall, objects are drawn in the top module, displayed via the display module, and those objects are then controlled via the cannon module (which is connected to the gamepad).

2 Technical Description and Design

2.1 Top Module

The top module is where the main game logic is handled. It's inputs are the 12MHz clock of the FPGA and the data from the the controller. It outputs the necessary signals for the controller and the VGA to work. It's components are the pll, vga, display, and cannon. The main function of top is to draw the game objects. This the code to draw the bricks:

```
TYPE ram_brick IS ARRAY(0 TO 2 ** 5 - 1) OF std_logic_vector(32 - 1 DOWNTO 0);
    SIGNAL ram_block : ram_brick := (0 =>  "00000000000011111100111100111111",
                                     1 =>  "00000000000011111111111111111111",
                                     2 =>  "00000000000011111110000001111111",
                                     3 =>  "00000000000011111001111110011111",
                                     4 =>  "00000000000011110011111111001111",
                                     others => (others => '0'));
```

Where there is a '1' bit is where a brick is drawn. The other details are more complicated but in general $'1'$ = brick drawn $'0'$ = brick not drawn. $0-4$ represent the

row being drawin. One of the major challenges encountered here was figuring out how to destroy a brick when it is hit by a cannon. This means where there is a $'1'$ bit, meaning a block is drawn at that position, we have to invert the bit when it is hit by the ball. The display module now knows not to draw that brick. This was solved by using bit-masking to change the bit.
eg) consider the row of bricks: 00000000000011111100111100111111 suppose the block at index 31 needs to be destroyed then the XOR logical operatoin works well for this.
$00000000000011111100111100111111 \oplus 00000000000000000000000000000010 =$
00000000000011111100111100111101

We create a new bit word of the same size as the ram block with the corresponding bit we want destroyed to be '1' and the other bits to '0'. Then $1 \oplus 1 = 0$ so this will invert the bit we want inverted. The other bits are unchanged as $1 \oplus 0 = 1$ and $0 \oplus 0 = 0$.

Two-Digit Display:
By understanding how to make the two LEDs look they're on simultaneously, two NMOS transistors were used as "switches" turning on and off each digit. Here is the schematic:
Then in the top module (device.vhd) the code to switch between which LED output is on was written. The logic for the left and right digits from (dddd.vhd) were used as inputs to a MUX with the select input being the msb of the counter driver. Then based on whether or not that bit was on or off one 7-bit output was (for the ones-place or tens-place) was outputted. Here is the block diagram for this process:
A zip file of all the source code will be provided.

This is the final implementation on the breadboard:

3 Results and Testing

The behavior of the circuit was farily straightforward to test. One could manually use the DIP switches to do all 2^6 possible number combinations which would take very long (but doable).

Doing this for all possible digits I was sure the circuit functioned correctly. Also a lab TA looked over my work and agreed it worked as intended.

4 Debugging Log

1. A few LEDs on the 7-seg weren't lighting up or they were very dim.
    - Manually setting the LEDs on would work but not otherwise.
    - some possible causes included something wrong with VHDL code, the wiring of the breadboard, and a hardware component not working (not likely).
    - Resolution: I needed to ground both GND pins of the Updiuno. After that the dim effect went away.
    - Lesson: Always check ground connections because even if other parts of the circut make sense you can get unexpected results.
2. problem: The transistors weren't acting as switches for the LED on the 7-seg. properly.
    - I knew there was a problem because both the digits on the 7-seg would always have the same numerical value. Both digits were displaying the value of the ones place.
    - possible causes included wiring of the transistor, confusion of what is the gate, drain and source.
    - The issue was that while the transistors were wired correctly they weren't acting as switches because the power pins of the 7-seg for both digits were

also connected. I.e the transistors weren't controlling the flow of current in a meaningful way since both the digits were always on regardless of the what the transistors were doing.

- the lesson I learned was to really understand what each component does and not just wire it up mindlessly. If I understood the role of the transistors, I would have recognized that the LED's on the 7-seg do not need to be connected to power while the source of the transistor was already connected to power.

3. problem: The LEDs weren't giving values as expected from the input given by the DIP switches.

    - There was a problem because when I toggled the switches the LEDs weren't producing the correct values.
    - Possible causes included: FPGA pin's not connected properly, the logic in the VHDL code.
    - It turned out that the reset signal on my counter was not set to any value. After making this change the outputs seen on the LEDs of the 7-seg were correct.

5 Reflection

1. What was the most valuable thing you learned, and why?
   - I learned how to debug my VHDL code in a way different to how one debugs code in a language like Java or Python. Many times the code can compile but the hardware implementation may not behave the way you want it.

2. What skills or concepts are you still struggling with? What will you do to learn or practice these?
   - I'm still struggling with how to write testbenches for very large complicated code. To practice I will review the textbook and try writing testbenches for other VHDL code I've written.

3. This assignment took about 72 hours (not including several hours tyring to write a testbench) during all of the days in which I worked on it.

6 Work Divison

7 source code