ECE 350
Final Project
Stephen Hughes, Yitaek Hwang

Our group implemented a two-level Frogger arcade game. The project requires multiple files: froggerMips2.asm, processor.v, sdh31_hw5.v, and display_sdh31.v, mul_div_sdh31.v, yh91_hw4.v, clksrc.v, and comparator.v.

External User Input (Keyboard)

sdh31_hw5 is the top-level file. We used the PS2_Interface keyboard controller given to us as part of the skeleton to integrate external user input to our system. We only use four buttons from the keyboard (up, down, left, and right arrow keys). These values taken from the keyboard is then passed onto the processor. When pressed, the PS2 interface would provide 8 bit values to the ps2_out bus. The values are listed below

Up - 0x75
Down - 0x72
Left - 0x6b
Right 0x74

In addition, when a key is pressed, the ps2_key_pressed wire is asserted to indicate a recent key event. The regFile was altered such that at any time, a left/right keypress would automatically write into r1, and an up/down keypress would automatically write into r2. This allowed for multiple writes in the same clock cycle.

Processor

The processor is a modified version of the pipelined processor that we submitted for homework 5. Mainly, registers were specified the hold the following information:

(Frog x and y coordinate registers)
r1: Holds the x coordinate of the frog, can be changed with left and right arrow keys
r2: Holds the y coordinate of the frog, can be changed with up and down arrow keys

(Enemy position registers - They only move horizontally so x-coordinate are saved only)
r3: Base x position of red-enemy in level 1
r4: Base x position of blue-enemy in level 1
r5: Base x position of yellow-enemy in level 1
r6: Base x position of the moving log in level 2
r7: Base x position of the lake in level 2

(End-game registers)
r10: 0 if game is not over, 1 if the game is in a winning state, 2 if the game is in a losing state

r11: Goal line x position, if the frog is within the goal line, game is over or move onto the next level

(Other helpful registers - Useful for MIPS code for branching purposes)
r18: Value of 20000 (large value for displays)
r19: Counter to go from zero to r18
r20: Holds 100 (left bound of the screen)

r22: Comparison register to r21 (stores 1: enemy collision case)
r23: Holds the number of lives
r24: Comparison register to r21 (stores 2: end-game case)
r25: Comparison register to r21 (stores 4: inside the log case)
r26: left screen bound
r27: left screen bound for r1 that goes left due to the log shifting
r28: Stores collision boolean value for enemy collision, end-game, or inside-log cases

sw0: reset key (default state is HIGH)

Hex Segment 6: Stores # Lives Remaining
Hex Segments 3-0: Outputs WIN or LOSE depending on state of game (it displays UUIn and 105E for win/lose)

External User Output (VGA & Computer Monitor)
VGA takes in outputs from the processor that needs to be drawn onto the screen (enemy positions, frogger, etc). The timing logic used for our 640x480, 25 Hz VGA cable was taken from this website: http://martin.hinner.info/vga/timing.html. Also, the code to translate register positions and draw out figures were modeled after this example code on: http://www.academia.edu/6990477/PONG_Game_Source_code_FPGA_Verilog.

The VGA draws pixel by pixel, starting on the top left of the screen, moving rightwards until the end of the horizontal line is detected. Then, the vertical line is incremented by one, and the process repeats itself. The state of the x, y pixels were stored in counterX and counterY, and are used extensively when drawing objects on the screen.

To make the enemy blocks move, we take the x-position and keep subtracting value at a constant rate. Collisions between the frog and any enemy on the screen (moving, stationary, or blue lake) are detected using the isCollision module in display_sdh31.v. The isCollision module is a structural piece of verilog code that takes in eight positions (four for each rectangle), and outputs a high value if the two rectangles are colliding. In addition, there is a isWithin module that structurally determines when one rectangle is entirely inside another (such as end-goal or the brown log/raft).

Colors for the frog, enemies, log, lake, and end goal are assigned using the 8-bit RGB code, which is then sent to the monitor to display.

Getting the width and height of the frog and the enemy blocks were mostly done experimentally, setting values and testing it on the screen. This process was very tedious, because compilation took approximately five minutes.

FroggerMIps2.asm

This code initializes the game by first taking all the positions of the frog and the enemy blocks to the right place, as well as setting the bounds and lives. The positions of the enemies are decremented, both collisions with enemies and with the end of level block are checked; in addition, once the enemy moves off the screen, its position is reset to keep scrolling.

Upon reaching the end of level block, the code transitions to a second level. Here, the lake x position is changed so that it now is displayed on screen. In addition, the log (which was previously placed off screen and was unable to move) now has its x position decremented so that it moves across the screen. If the frog is entirely inside the log, r21 stores the value of 4 (this is done in hardware), so that our assembly instructions can detect this state. Once the state is detected, we decrement the frogs x position by 1 immediately after decrementing the logs position (to give the effect of the frog traveling on the log).

Upon losing all lives, LOSE is written to the 7-segment display, and the game freezes. If the frog has completed both levels, WIN is written to the 7-segment display, and the game once again freezes.

Other Example Programs

With the current I/O interface, almost any arcade game can be built and displayed on the VGA monitor. It is easy to extend the keyboard input to allow for more than just up, down, left, and right commands. Potentially, multiple users could be using the same keyboard for multi player games.

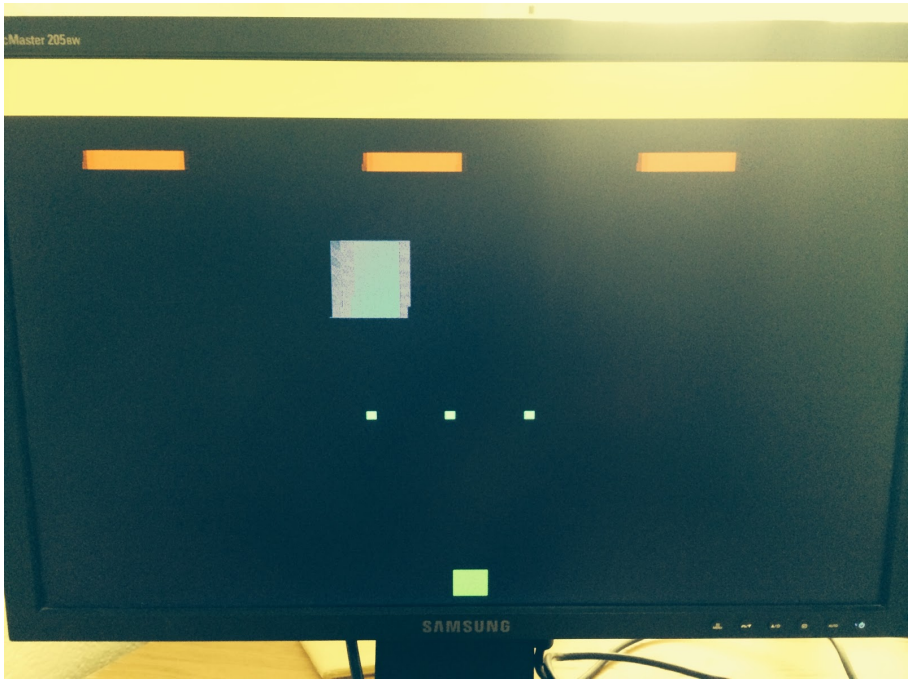

Figure 1: Beginning of Level 1

Figure 2: Beginning of Level 2

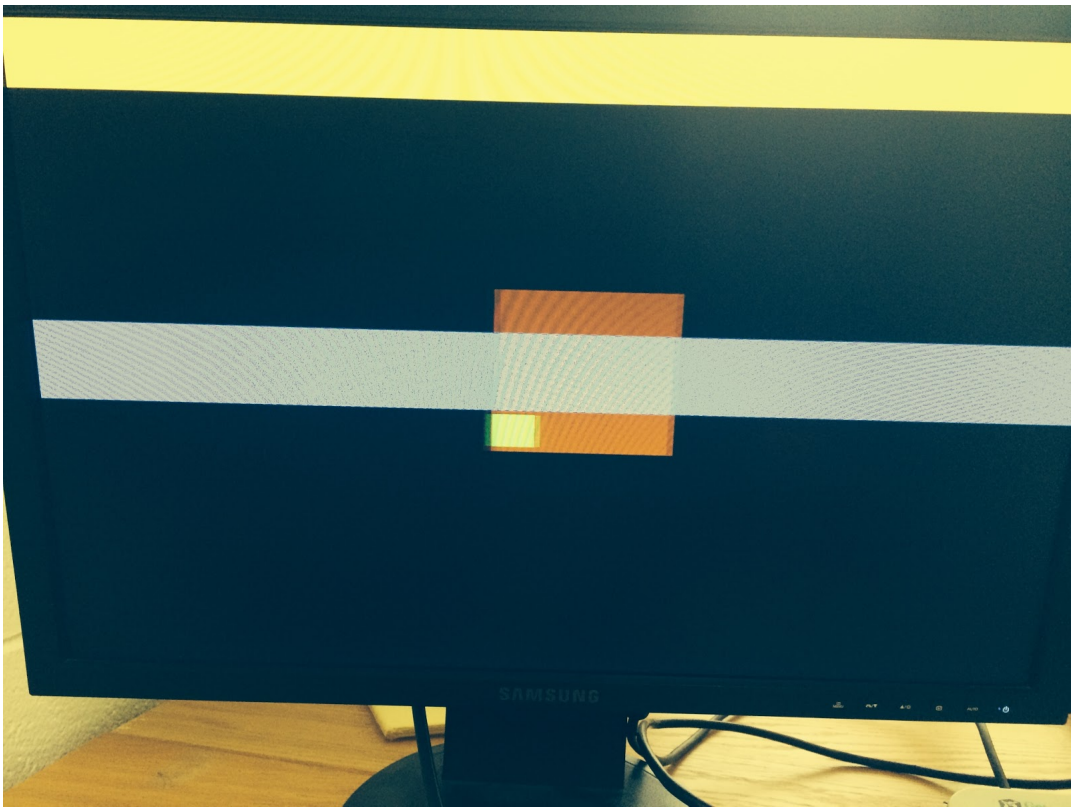

Figure 3: Frogger Traveling on Raft

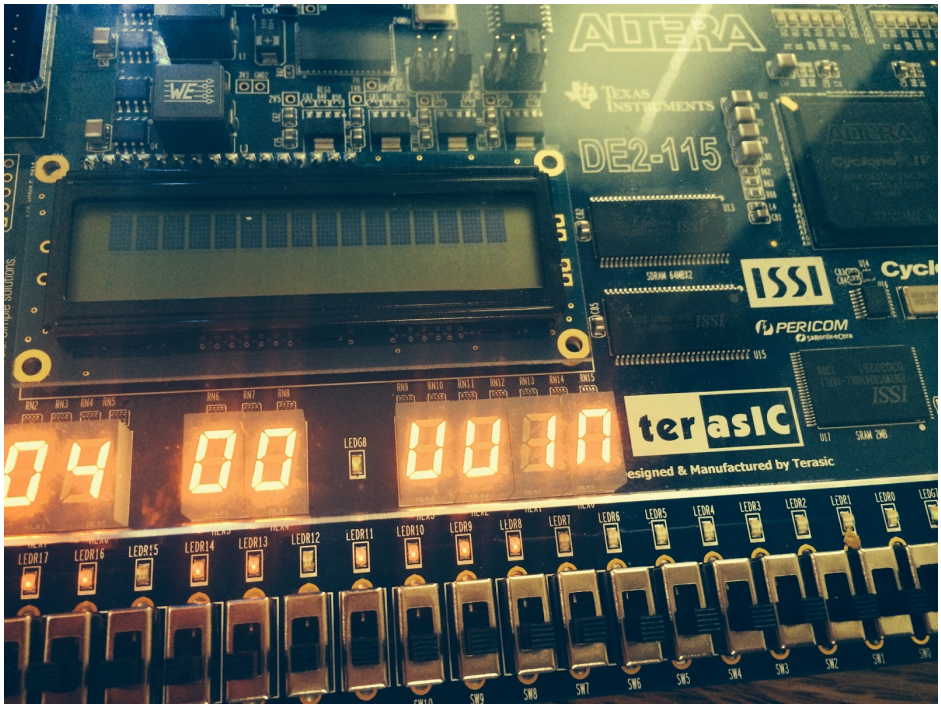

Figure 5: Winning State (4 Lives Left)

Figure 6: Losing State (0 Lives Left)