ECE385 Final Project Report

Frogger in System Verilog

Eric Meyers, Ryan Helsdingen

Section ABG; TAs: Ben Delay, Shuo Liu

May 4th, 2016

emeyer7, helsdin2

I. Introduction

For this ECE 385 project, we recreated the classic game called Frogger. The basic premise of Frogger is to navigate three frogs through obstacles from the bottom to the top of the screen. Frogger must first pass through four lanes of traffic and a river full of lily pads in order to successfully make it to the other side of the map. A frog may die by either colliding with a moving car or falling in the water. There are a total of three frogs that the user must navigate to the other end of the map, and once all three frogs move to their particular ending location, the user wins. If a user dies three times, then the game is over.

This system was developed in System Verilog in Quartus-II on an Altera-DE2-115 FPGA Board, and used software drivers developed in C to communicate with a USB keyboard (to be used as the controller).

II. List of Features

- User controlled frog sprite moves according to grid set on VGA display
  - Up, down, left, or right depending on keyboard input
  - Frog position restricted to on-screen
  - Frog direction held after keyboard direction inputed
  - Frog contains collision algorithms for obstacles
  - A total of three frogs placed into game at three different starting points
    - three endpoints for frogs to get to
    - frog can finish at any endpoint but only one frog can finish per endpoint
- Generation of moving obstacles
  - Four lanes of at least three cars in them
    - Each row moves at specific speed, direction
    - Cars wrap around screen
    - Frogger dies upon impact with car ending the game
  - Four lanes of at least three lily pads in them
    - Each row moves at specific speed, direction
    - Lily pads wrap around screen
    - Frogger lives by stepping onto the lily pad, drowns by falling into the water
    - If the lily pad moves off the screen with Frogger on it, Frogger will fall into the water
- Working game clock at the top of the screen.
  - 60 seconds to complete the level.
  - Clock resets at the start of the game
  - Game ends when the game clock runs out of time
- Color/VGA Monitor Output
  - Detailed sprites give look and feel in graphical user interface
  - Colors allow user to clearly differentiate between obstacle, user controlled frog, and the map background
- Game reset button established that can reset the game at any point.

III. Block Diagram

IV. Purpose of Modules

A. *final_frogger_top*

B. *Color_Mapper*

1) Displays Frogger, Car rows, and Lily pad rows :

2) Sets orientation of Frog Direction:

C. frog

D. car

E. car_row

F. lilypad

G. lilypad_row

H. game_logic

Module game_logic.sv controls the overall finite state machine of the Frogger game. This module monitors number of frogs to successfully get to the other side of the screen. It then sends the game into a win or death state depending on the outcome of the game. This module also controls the value of the game clock. If the game clock runs out of time before the three frogs reach the other side, the game ends and the player loses.

I. hpi_io_intf

This method takes HPI output values from the NIOS II as input, checks for a call to reset, and if no reset, assigns these values to the proper values to be outputted by the top-level module to the CY7C67200 chip. Inputs and outputs used are shown below.

```
input [1:0]  from_sw_address,
 output[15:0] from_sw_data_in,
 input [15:0] from_sw_data_out,
 input        from_sw_r,from_sw_w,from_sw_cs,
 inout [15:0] OTG_DATA,
 output[1:0]  OTG_ADDR,
 output       OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N,
 input        OTG_INT, clk, Reset);
```

J. VGA_controller

The VGA_controller manages output to the monitor. Inputs include clock and reset. Outputs include several sync signals, a pixel clock specified for 25 MHz and 10 bit horizontal and vertical coordinate signals as shown below.

```
input        clk,       // 50 MHz clock
             Reset,     // reset signal
output logic hs,        // Horizontal sync pulse.  Active low
             vs,        // Vertical sync pulse.  Active low
             pixel_clk, // 25 MHz pixel clock output
             blank,     // Blanking interval indicator.  Active low.
             sync,      // Composite Sync signal.  Active low.
                        // We don't use it in this lab, but the video
                        // DAC on the DE2 board requires an input for it.
output [9:0] DrawX,     // horizontal coordinate
             DrawY );   // vertical coordinate
```

V. Circuit Schematics

VI. Finite State Machines

VII. Color & Sprite Generation

VIII. Difficulty

Producing the Frogger game in less than five weeks was no simple task. The team came across a number of issues on a range of the features that were implemented into the game. Most of the problems encountered came about from dealing with the System Verilog software. What would be a simple function in a high-level programming language became rather tedious in System Verilog and required some thought in order to successfully implement.

One of the most difficult tasks in making the game was collision control for the Frogger sprite with obstacles, particularly with the lily pads. The algorithm for collision control turned into a long AND/OR statement for the cars. Collision for the frog with the cars was easy once that long statement was put into place and modified for precision. It was easy because a simple contact between the frog sprite and car sprite killed the frog and ended the game. Creating collision control with the lily pads was far more difficult to implement. The frog should land on the lily pad and live, floating across the screen holding its position fixed to the lily pad until another direction key is pressed. Clock issues came into play when trying to match up the speeds of the two colliding sprites, especially when the two sprites have different frame clocks.

Another issue was generating the high level graphics for the sprites and background. The on-chip memory alone was not enough to hold the amount of colors and pixels of all graphics desired for the project. SDRAM had to be implemented to store the memory for all 640 x 480 pixels on the screen.

A LOT MORE COULD BE ADDED HERE

IX. Conclusion

X. Figures

Appendix