CSE100 Lab 5 – Dino Run

Contents

1 Submission

- Due: Tuesday 8/13/2024 at the end of your lab section.
- Submit your Lab Write-Up to Gradescope.
- Submit your code to the Gradescope autograder. (You have unlimited submission attempts).
- Demonstrate your implementation to a TA.

2 Goals

You will implement a state machine and connect several modules as part of a game called *Dino Run*.

Once you finish your implementation, you will program it to a Basys 3 FPGA Board.

3 Prelab

Read the following sections before starting your lab.

3.1 Overview

Watch this video to understand what to do for this lab: https://youtu.be/f_b3rT6Fiwk.

The video is summarized here:

- At the start of the game, the title screen should be visible, the dinosaur should be playing its running animation, and there should be no obstacles on-screen.
- Upon receiving `start_i`, the title should disappear and the obstacles should begin spawning.
- Upon receiving `up_i`, the dino should jump.
- Upon receiving `down_i`, the dino should duck.
- Upon the dinosaur colliding with an obstacle, the screen should freeze and the dinosaur should display the "hit" state.
- While the screen is frozen from a collision, receiving a `start_i` should reset the score and all the obstacles, and the obstacles should resume spawning and shifting left.

On the BASYS3 board,

- PushButton `btnC` drives the `start_i` signal.
- PushButton `btnU` drives the `up_i` signal.
- PushButton `btnD` drives the `down_i` signal.
- PushButton `btnR` drives the synchronous active-low global reset `rst_ni`.

3.2 Random Number Generator

You need to finish the random number generator in `"rtl/lfsr16.sv"`.

You will use a Linear Feedback Shift Register (LFSR) to generate a random 16-bit binary number. You should adopt your RTL from an earlier lab.

For a 16-bit LFSR, you should use the following taps: `16'b1101000000001000`: https://en.wikipedia.org/wiki/Linear-feedback_shift_register#Example_polynomials_for_maximal_LFSRs

3.3 Edge Detector

You are given an `edge_detector` module. The edge detector will output a 1-cycle pulse when the `data_i` transitions from 0 to 1. You should use this to convert from VGA vsync into the `next_frame` input for each of the game objects.

3.4 Title

You are given a `title` module that takes in an x and y coordinate and combinationally returns whether the pixel at that coordinate is on the title (0-cycle delay). You should use the pixel output when assigning `vga_red_d`, `vga_green_d`, and `vga_blue_d`.

You may run `title_tb` to generate a GIF of how the title will be displayed to the screen.

3.5 Cactus and Bird Obstacles

You are given `cactus` and `bird` obstacle modules. You may instantiate as many obstacles as you like. (For example, you may try 1 bird and 2 cacti).

On reset, the obstacles appear off-screen. If `spawn_i` goes high while the obstacle is off-screen, the obstacle will spawn on the right side of the screen, and begin sliding to the right on every clock pulse that `next_frame` is high. If `spawn_i` goes high while the obstacle is on-screen, then `spawn_i` is ignored.

`spawn_i` should be activated at a random interval determinded by several bits from the LFSR. Each obstacle should have a different key that it waits for. When that key occurs, the obstacle should spawn. For example, you may choose to spawn a cactus when `lfsr16.rand_o[12:8] == 5'b00000` or spawn a bird when `lfsr16.rand_o[12:7] == 6'b101010`.

Additionally, each obstacle takes in a `rand_i` input to determine bird height and cactus style. This `rand_i` should be aquired from the LFSR.

The obstacles take in an x and y coordinate and combinationally returns whether the pixel at that coordinate is on the respective obstacle (0-cycle delay). You should use the pixel output when assigning `vga_red_d`, `vga_green_d`, and `vga_blue_d`.

You may run `obstacle_tb` to generate a GIF of how the obstacle will be displayed to the screen.

3.6 Dino

You are given a `dino` module. When `go_i` goes high, the dino will jump. When `down_i` goes high, the dino will duck. When `hit_i` goes high, the dino will display the hit sprite. The position and animation will update each clock pulse that `next_frame` is high.

The `dino` takes in an x and y coordinate and combinationally returns whether the pixel at that coordinate is on the respective obstacle (0-cycle delay). You should use the pixel output when assigning `vga_red_d`, `vga_green_d`, and `vga_blue_d`.

You may run `dino_tb` to generate a GIF of how the dino will be displayed to the screen.

3.7 Score Counter

You are given a `score_counter` module. On reset, the score is set to 0. When `en_i` is high, the score goes up by 1. When not on the title screen, and not hit, the score should increment by 1 every

frame.

The digits are outputted as binary-coded-decimal.

3.8 Dino Run SystemVerilog Package

You need to finish the `state_t enum` in `dinorun_pkg`. It is common to use SystemVerilog packages to organize parameters and typedefs so that they can be used across a variety of files including RTL and testbenches.

3.9 VGA Timer

You need to finish the VGA Timer in `"rtl/vga_timer.sv"`. You should use your RTL from an earlier lab.

You should pass the x and y coordinates into your game objects to determine which objects should be drawn at which pixels. You may choose which colors correspond with which objects (`dino`, `cactus`, `bird`, `title`). Be sure to add a flip-flop right before the VGA outputs for the best image clarity.

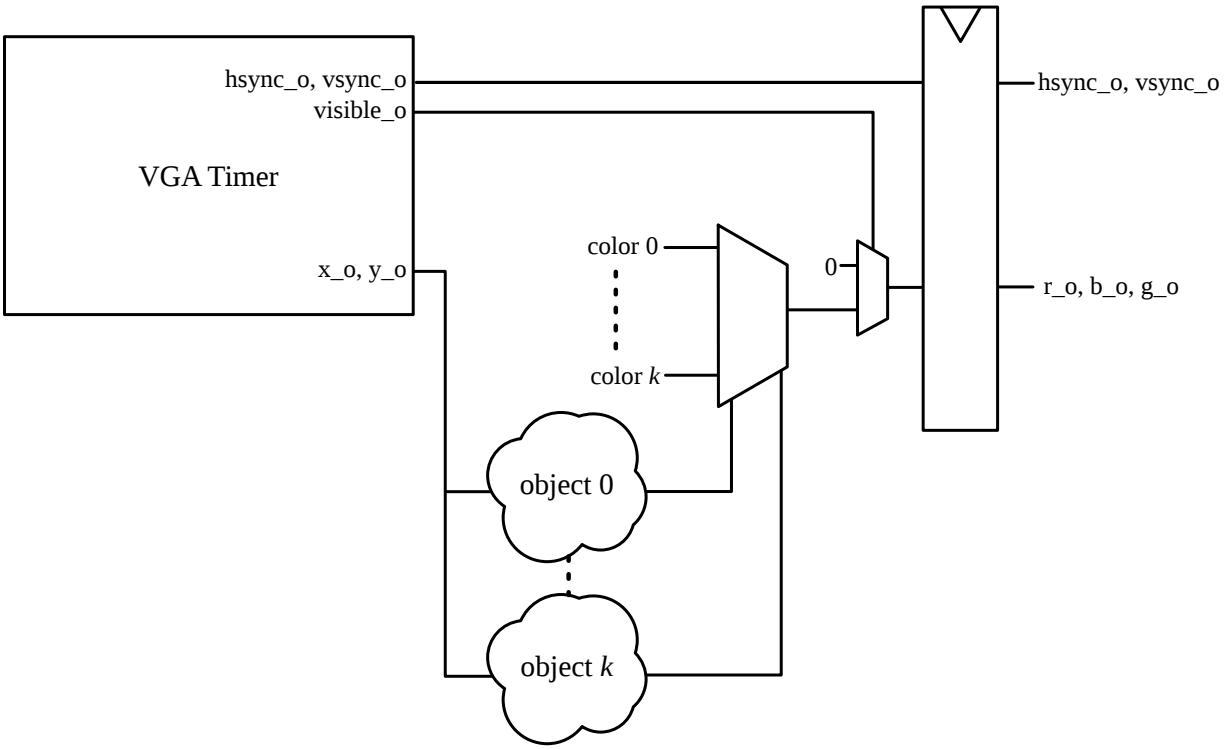

Figure 1: VGA Timing

3.10 7 Segment Display

You need to finish the 7 Segment Display driver in `"rtl/basys3/basys3_7seg_driver.sv"`. You should use your RTL from an earlier lab.

3.11 Dino Run State Machine

You need to finish the state machine in `"rtl/dinorun.sv"` according to the lab instructions.

Read the State Machine section in the lowRISC SystemVerilog style guide: https://github.com/lowRISC/style-guides/blob/master/VerilogCodingStyle.md#finite-state-machines.

Advice: Getting a state machine right usually requires several iterations. It is likely that in demonstrating your design the TA will discover some case that is not properly handled. Often changes to the state machine are not simple. A complete redesign may be necessary. Trying to patch it by changing one signal here or there, or inserting a gate/FF, or more, almost always makes things worse. Please leave yourself enough time. Hurrying will introduce more bugs that you will need to hunt down. It is strongly suggested that your entire design be entered and simulating **before** your second section.

4 Lab

- `"synth/basys3/Basys3_Master.xdc"`
- `"rtl/dinorun_pkg.sv"`
- `"rtl/dinorun.sv"`
- `"rtl/lfsr16.sv"`
- `"rtl/vga_timer.sv"`
- `"rtl/basys3/basys3_7seg_driver.sv"`
- `"rtl/basys3/hex7seg.sv"`

1. Complete the auxiliary modules and ensure they pass the tests mentioned in the README.
2. Draw a state diagram for *Dino Run* using all the states mentioned in `dinorun_pkg::state_t` in `"rtl/dinorun_pkg.sv"`. Be sure to denote on each arrow what condition is required to enter the next state. There is no need to assign an encoding in your state diagram; simply use the `dinorun_pkg::state_t` enum labels.
3. Implement the state machine you drew. Be sure to follow the lowRISC coding style for designing state machines.
4. The Basys3 constraints file `"synth/basys3/Basys3_Master.xdc"` was downloaded from Digilent's GitHub: https://github.com/Digilent/Basys3/blob/master/Resources/XDC/Basys3_Master.xdc. Complete it to match your `basys3` module. Refer to previous labs and the Basys3 Reference Manual if you have questions: https://reference.digilentinc.com/reference/programmable-logic/basys-3/reference-manual.
5. Simulate your design using commands specified in the `"README.md"`. Submit to the autograder until you get 100%. Note that passing the autograder will not guarantee that your implementation is correct.
6. Synthesize your design and program it to the Basys3 using commands specified in the `"README.md"`. Your design may not work the first time. Review the waveforms for any issues that the autograder did not catch.
7. Get checked off by a TA.
8. Finish the writeup.

Important Please remember to turn off the power to BASYS3 board when you are done.

5 Write-Up

1. Include a state bubble diagram. The diagram can be hand drawn or drawn digitally (https://draw.io/).
2. Describe each state of your machine in words.

Example: "The machine starts in `TITLE_SCREEN`. It stays in this state until the `start_i` signal (`btnC`) goes high."