Francis (Haocheng) Feng - 205 339 293
Megan Pham - 505 313 300
CS M152A Lab 2
Dipti Sahu

# Lab 4 Report

## Introduction

Slot machines are very popular in the gambling industry. Partly based on luck, these machines consist of three to four slots and the player wins if the number the slots land on are the same. The reward depends on how many matches you get: doubles, triples, quadruples. For our final project, we wanted to create a slot machine simulation that would utilize the material that we learned this quarter, as well as incorporate new concepts such as using LEDs and random numbers.

## Module Design

In the first submodule, **clks**, we created the four different clocks we needed to use, 9,8 Hz, 10 Hz, 11 Hz, 13 Hz, and a fast clock. To calculate our clocks from 100 Mhz, we used the following logic: say we want to build a 9.8 Hz from 100 MHz. Every time the clock hits 10,200,000, we will toggle a flip-flop. Toggling at 10 million (100 million/ 10 million = 10) gives us a 10 Hz clock. Toggling at 11 Hz (100 million/ 9,090,909 = 11) gives us a 11 Hz clock. For the 12 Hz clock, we toggled at 11 million counts. The fast clock rate we chose was 500Hz, so we toggled the pin at 100,000 counts.

In order to combat the oscillating behavior generated by hardware inconsistencies, we created a module called **debouncer**. This module serves to filter out the noise created by pressing a button and only registering one valid push. By using a flip flop as a “buffer”, we can use the output of the flip flop as the input for the button. This method will allow for a consistent output to all other components in the slot machine.

In order to simulate a slot machine, we built a **random** module that acts as a pseudo-random number generator. The core design principle behind this module is to use the previous state as an input to a linear function that will generate the next state. This methodology will generate a long list of numbers that will look seemingly random when in fact they follow a pattern each run. The module takes in a limit and a seed input. The limit input controls when the random number generation should stop, while the seed input is used to add variation to the random numbers generated. The seed

input is crucial in ensuring that the random numbers have different patterns and will appear to be random.

The seed used in the random module is generated by the **next_state** module. If the slot machine is in play, the module will take the current state and add it to the previous state to create the next state as an output. However, if the slot machine is not in play, then the state will continue looping and stay the same.

The **spin** module generates the decimal number to be displayed on the slot machine. It takes in the four clocks generated by the clks module and uses them as input to generate random numbers. The random module uses four different clocks for a varied number generation. Additionally, they each get their own unique state from the next_state module and will continue to update as long as the slot machine is in play.

We also wanted to incorporate aspects of a Vegas slot machine with its display of lights. This is shown in **blinkLeds.** Utilizing the LEDs above the switches, we made them blink in a sequential trickle-to-the-right manner while the game was running. To implement this, a counter of 31 bits was used, so that the LEDs would not blink too quickly or too slowly.

To keep track of our score, we implemented a **score** module, which would extract digits from the current score that's passed in. For example, if our score is 120, then to output it in the seven-segment display, we would have to get 1, 2, and 0 individually. To do this, we used a mixture of regular and modulus division. Using that same example to get the tenths place digit, we would have to divide the score by 10 to get 12 and then mod it by 10, which would ultimately give us the digit of 2.

Our **jackpot** module is designed to keep track of the reward the users would receive depending on their slot machine results. If two of the slots matched each other, the score would change to 10. If there were triples, we would have a score of 50, and with quadruples, we would have a jackpot (a winning score of 100)!

So far, our modules have been working with the decimal form of the slot machine displays. However, in order to display the time correctly on the FPGA board, we must convert the time from the decimal form into a seven-segment encoding. The module **seven_segments** takes in decimal as input and churns out an 8-bit representation of the seven-segment display for one of the time slots. If the digit is invalid, any number not in the range from 0 to 9, then all 8 bits will be set to 1 which translates to an empty display.

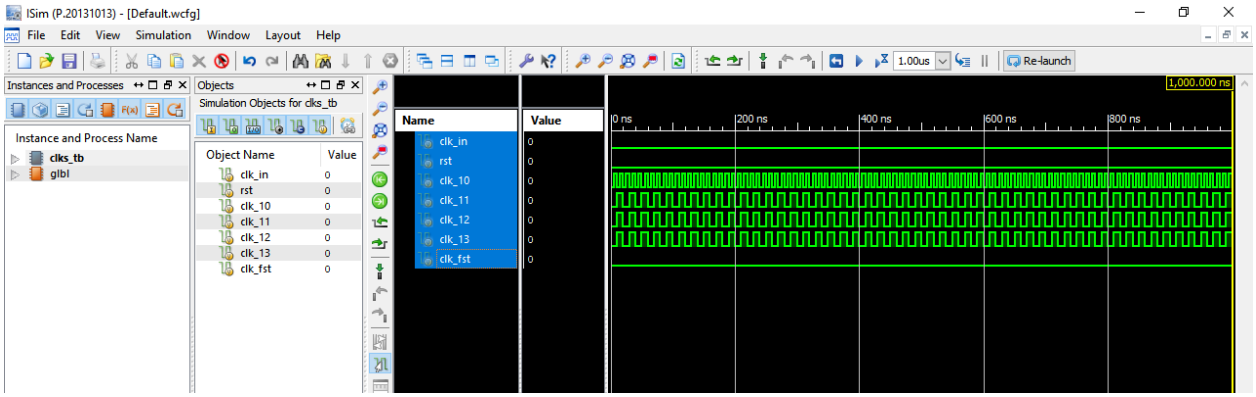
The purpose of the **show_segments** module is to correctly display the seven-segment display on the FPGA board given the inputs from the other modules. At the top level, the module uses a register to keep track of which of the four positions on the display that the module is updating. The register counts from 0 to 3 and repeats as long as the program is being executed. For each position, the appropriate anode is updated. Additionally, the module must be able to handle two different states, score and regular. During score mode, the display will show the current score. During regular mode, the display will show digits according to the state of the slot machine. At each tick, the display will be updated with the most current times in an 8-bit format.

The final module is the main **vegas** module that will call all of the other modules and connect them at the top level. This module first calls the debouncer module to correctly configure the buttons. After that, the blinkLeds module will be called to initialize the sequential blinking. Then the clocks must be set up so that they can be used later by other modules to clock the displays. Next, the spin module is called to initialize and update the current slot display in decimal form. We also call the jackpot module in order to detect when we have matches in our slot and to allocate the correct score. The outputs from jackpot will be passed to the score module in order to prep our digits for the seven_segments module. The output of the spin and score module will then be passed on to the seven_segments module to convert the slot machine display and score display from decimal form to an 8-bit representation. The four 8 bit representation output will be used as input for the show_segments module which will help to display the appropriate seven segments in all four positions for both displays.

## Testbench Design

In our testbench code, we made sure to test the modules that we completed. As modules were intertwined with each other, it made sense to primarily test the different clocks we have to generate our random numbers and blink our displays.

# Simulation Waveforms




# ISE Design Overview Summary Report

| vegas Project Status (03/11/2022 - 10:44:37) | | | |
|---|---|---|---|
| **Project File:** | vegas_sim.xise | **Parser Errors:** | No Errors |
| **Module Name:** | vegas | **Implementation State:** | Programming File Generated |
| **Target Device:** | xc6slx16-3csg324 | **• Errors:** | No Errors |
| **Product Version:** | ISE 14.7 | **• Warnings:** | 90 Warnings (15 new) |
| **Design Goal:** | Balanced | **• Routing Results:** | All Signals Completely Routed |
| **Design Strategy:** | Xilinx Default (unlocked) | **• Timing Constraints:** | All Constraints Met |
| **Environment:** | System Settings | **• Final Timing Score:** | 0 (Timing Report) |

| Device Utilization Summary | | | | [-] |
|---|---|---|---|---|
| **Slice Logic Utilization** | **Used** | **Available** | **Utilization** | **Note(s)** |
| Number of Slice Registers | 391 | 18,224 | 2% | |
| Number used as Flip Flops | 391 | | | |
| Number used as Latches | 0 | | | |
| Number used as Latch-thrus | 0 | | | |
| Number used as AND/OR logics | 0 | | | |
| Number of Slice LUTs | 962 | 9,112 | 10% | |
| Number used as logic | 947 | 9,112 | 10% | |
| Number using O6 output only | 583 | | | |
| Number using O5 output only | 246 | | | |
| Number using O5 and O6 | 118 | | | |
| Number used as ROM | 0 | | | |
| Number used as Memory | 0 | 2,176 | 0% | |
| Number used exclusively as route-thrus | 15 | | | |
| Number with same-slice register load | 0 | | | |
| Number with same-slice carry load | 15 | | | |
| Number with other load | 0 | | | |
| Number of occupied Slices | 340 | 2,278 | 14% | |
| Number of MUXCYs used | 452 | 4,556 | 9% | |

| | | | | |
|---|---|---|---|---|
| Number of LUT Flip Flop pairs used | 980 | | | |
| Number with an unused Flip Flop | 621 | 980 | 63% | |
| Number with an unused LUT | 18 | 980 | 1% | |
| Number of fully used LUT-FF pairs | 341 | 980 | 34% | |
| Number of unique control sets | 30 | | | |
| Number of slice register sites lost to control set restrictions | 113 | 18,224 | 1% | |
| Number of bonded IOBs | 24 | 232 | 10% | |
| Number of LOCed IOBs | 24 | 24 | 100% | |
| Number of RAMB16BWERs | 0 | 32 | 0% | |
| Number of RAMB8BWERs | 0 | 64 | 0% | |
| Number of BUFIO2/BUFIO2_2CLKs | 0 | 32 | 0% | |
| Number of BUFIO2FB/BUFIO2FB_2CLKs | 0 | 32 | 0% | |
| Number of BUFG/BUFGMUXs | 5 | 16 | 31% | |
| Number used as BUFGs | 5 | | | |
| Number used as BUFGMUX | 0 | | | |
| Number of DCM/DCM_CLKGENs | 0 | 4 | 0% | |
| Number of ILOGIC2/ISERDES2s | 0 | 248 | 0% | |
| Number of IODELAY2/IODRP2/IODRP2_MCBs | 0 | 248 | 0% | |
| Number of OLOGIC2/OSERDES2s | 0 | 248 | 0% | |
| Number of BSCANs | 0 | 4 | 0% | |
| Number of BUFHs | 0 | 128 | 0% | |
| Number of BUFPLLs | 0 | 8 | 0% | |
| Number of BUFPLL_MCBs | 0 | 4 | 0% | |
| Number of DSP48A1s | 4 | 32 | 12% | |
| Number of ICAPs | 0 | 1 | 0% | |
| Number of MCBs | 0 | 2 | 0% | |
| Number of PCILOGICSEs | 0 | 2 | 0% | |
| Number of PLL_ADVs | 0 | 2 | 0% | |
| Number of PMVs | 0 | 1 | 0% | |

| | | | | |
|---|---|---|---|---|
| Number of STARTUPs | 0 | 1 | 0% | |
| Number of SUSPEND_SYNCs | 0 | 1 | 0% | |
| Average Fanout of Non-Clock Nets | 3.64 | | | |

| Performance Summary | | | [-] |
|---|---|---|---|
| **Final Timing Score:** | 0 (Setup: 0, Hold: 0, Component Switching Limit: 0) | **Pinout Data:** | Pinout Report |
| **Routing Results:** | All Signals Completely Routed | **Clock Data:** | Clock Report |
| **Timing Constraints:** | All Constraints Met | | |

| Detailed Reports | | | | | [-] |
|---|---|---|---|---|---|
| **Report Name** | **Status** | **Generated** | **Errors** | **Warnings** | **Infos** |
| Synthesis Report | Current | Fri Mar 11 10:44:08 2022 | 0 | 90 Warnings (15 new) | 18 Infos (2 new) |
| Translation Report | Current | Fri Mar 11 10:44:13 2022 | 0 | 0 | 0 |
| Map Report | Current | Fri Mar 11 10:44:20 2022 | 0 | 0 | 8 Infos (0 new) |
| Place and Route Report | Current | Fri Mar 11 10:44:26 2022 | 0 | 0 | 0 |
| Power Report | | | | | |
| Post-PAR Static Timing Report | Current | Fri Mar 11 10:44:30 2022 | 0 | 0 | 3 Infos (0 new) |
| Bitgen Report | Current | Fri Mar 11 10:44:36 2022 | 0 | 0 | 0 |

| Secondary Reports | | | [-] |
|---|---|---|---|
| **Report Name** | **Status** | **Generated** | |
| ISIM Simulator Log | Out of Date | Fri Mar 11 11:09:51 2022 | |
| Post-Synthesis Simulation Model Report | Out of Date | Mon Mar 7 11:57:40 2022 | |
| WebTalk Report | Out of Date | Fri Mar 11 10:44:36 2022 | |
| WebTalk Log File | Out of Date | Fri Mar 11 10:44:37 2022 | |

**Date Generated:** 03/11/2022 - 11:09:52

clock_type.v | score.v | vegas.v | next_state.v | jackpot.v | blinkLeds.v | spin.v

## Conclusion

Overall, we designed and implemented a slot machine for our final project. We were able to implement a pseudo-random generator, blinking lights, and displays. The most difficult parts were that we were on a time crunch and were involved in complications with COVID-19. The random number generator took up most of our time as we were trying to implement a true random number generator instead of a pseudo-random number generator. However, after the project was completed, it was very fun to take a look at the other group projects and interact with our fellow classmates.