**ECE 324 Final FPGA Design Project**

Name(s):

Alex Blake

Jameson Shaw

|  |  |  |
| --- | --- | --- |
| **Demonstration (20 points total)** | **Course outcome** | **Grade** |
| 1. Design features well demonstrated and explained | G-2 | /4 |
| 1. Difficulty of design | C-1 | /4 |
| 1. Success of implementation | C-4, K-2 | /4 |
| 1. Presenter spoke clearly, was easy to understand and answered questions well | G-2 | /4 |
| 1. Overall awesomeness | \* | /4 |
|  | **Demo. Total** | /20 | /20 |
| **Report (20 points total)** | **Course outcome** | **Grade** |
| 1. Design goals and solution approach | C-1, C-3 | /3 |
| 1. Functional specification, including block diagram and state-transition diagrams | C-2 | /8 |
| 1. Discussion of results, conclusion | C-2, C-3 | /3 |
| 1. Timing, utilization summary reports | B-1, K-3 | /2 |
| 1. Acknowledgements and references | I-1 | /1 |
| 1. Verilog (.v), design-constraint (.xdf) files submitted | K-3 | /3 |
|  | **Report Total** | /20 | /20 |
|  | **Project Total** | | **/40** |

\*Not an ABET course outcome

# Course Outcomes

B-1. Identify timing constraints, make time budgeting assumptions, and verify models for digital systems.

C-1. Analyze needs to produce problem definition for digital systems including implementation on FPGAs using HDL programming.

C-2. Carry out design process (such as concept generation, modeling, evaluation, iteration) to satisfy project requirements for digital systems including implementation on FPGAs using HDL programming.

C-3. Design a system that will meet realistic constraints such as economical, manufacturability, safety, etc

C-4. Build prototypes on FPGAs that meet design specifications.

G-2. Deliver well-organized, logical oral presentations, including good explanations when questioned.

I-1. Use resources such as CAD tool tutorials and FPGA board manuals, to learn new material not taught in class.

K-2. Utilize programmable devices such as FPGAs for rapid prototyping and testing.

K-3. Use computer-aided design and analysis software tools for digital systems design on FPGAs.

\* Not an ABET outcome

**Design Goals:**

The original projec plan was to have a working ball maze game displayed through a VGA connection. The ball would accelerate via the movement of the FPGA board by using the built-in accelerometer. The ball maze would be like a puzzle game where there would be a hole for the ball to fall through which would be the end-game condition. The original timeline is listed below:

*Week 1 (April 2nd):*

* *Maze Initiated, Output to VGA*
* *April 4th – Revised Project Proposal*

*Week 2 (April 9th):*

* *Ball sprite created*
* *Moving Plane with buttons*
* *Begin with acceleration*

*Week 3 (April 16th):*

* *Status Review*
* *Finish Acceleration of ball from plane*

*Week 4 (April 23rd):*

* *Make the hole at the end of the maze*
* *Game win-condition*
* *Add additional features? Different maze levels?*

*Week 5 (April 30th):*

* Finalize
* Presentation
* Written Report Due

**Reflection of Design Goals:**

Now that we are preparing the project for presentations, some of our goals were not met. There is a coded win-game condition but is untested due to time constraints. This also includes other features that were supposed to be implemented in the later weeks. Instead, we were happy to accomplish getting the accelerometer to work, considering it was a lot more work than expected. That was probably the hardest part since the SPI module provided from the book examples required a controller module for it to be used with the accelerometer. Given more time, we would have finished the win condition and try to make the game look better. Also, wall detection is still a bit lacking as well if the ball is moving diagonally.

**Solution Approach:**

The project was based on the PacMan lab from class. Most of that lab was leveraged for the visuals and provided a starting platform for the project. The same text files were used and edited slightly but provide a solid base. The ball movement was done mostly from scratch instead of the motion done by the PacMan lab already. The motion eventually would be controlled by the accelerometer but started with the pushbuttons on the board.

**Functional Specifications:**

The only controls in the final game is a reset button which resets everything back to initial conditions and tilting the board. The tilting should correspond with the acceleration the ball experiences on the monitor.

**Block Diagram:**

A close up of text on a white background

Description automatically generated

**Figure 1: Block Diagram of Ball Maze Project**

**BallMaze:**

This module is the top module which ties all the hardware components together. The ROM is tied to this as well as the SPI wires for the accelerometer and the output of the VGA and seven-segment display. This still includes the older methods of ball acceleration using buttons (going to shift registers). This module controls the signals sent to the VGA connection as well. This basically generates everything and then requires the balls location which is all handled by the ball motion module.

**BallMotion:**

This module does the heavy lifting regarding the ball movement. This module was pretty much from scratch. The module is based on calculating ball acceleration, then ball velocity, and then ball position. These are pipelined via flip-flops. The original module used the buttons to change acceleration but towards the end, the accelerometer controlled those instead; this is done by the ReadAccel module. Also, collisions are detected by the BallMaze module which is sent to this module to determine if the ball can move in a given direction.

The ball motion changes by changing the update rate via universal binary counters which was proportional to the velocity. The X and Y coordinates were calculated independently which allows for diagonal movement.

**ReadAccel:**

This module is the brains of the communication between the BallMotion module and the SPI module. The SPI module takes an 8-bit bus and sends each bit per clock cycle via the MOSI wire. The module waits for a start bit to go high which begins the serial communication via the SPI interface. When the module is complete finishing the signal, it outputs a ready signal.

The SPI module works great but requires a controller. It doesn’t know what and when to send information which is a huge timing task that requires an FSM (SPI Controller FSM). The basic design of the FSM works by using a counter between states that increments per clock cycle to ensure it waits long enough between stages as well as when to enable the SPI CS wire. This code is a bit messy but is fully functional (see design below).

This module consists mainly of the FSM but also requires a mod counter which becomes the clock signal for the FSM and clock signal of the SPI module as well. For communication, the frequency must be less than 100MHz as per the data sheet so this was achieved with a mod counter with the value of 250.

**SPI Controller FSM:**

A close up of text on a white background

Description automatically generated

**Figure 2: Block Diagram of SPI Controller FSM**

The SPI Controller (ReadAccel Module) is laid out as shown above. The Controller is broken up into four sections.

**Initialize Accelerometer:**

This is the first section of the FSM to happen and only happens once. The code progresses through 3 stages which must send three different 8-bit values to the accelerometer through the SPI module with timing between: write instruction, address of register, and value to write to register. Then it goes to the Idle section.

**Idle:**

This section consists of only one state which pauses for a short delay of 10 clock cycles. This is the beginning of the read sequence loop and is the top of the FSM that loops indefinitely after the first section.

**Read X-Axis and Read Y-Axis:**

This is the section where the accelerometer data is read from registers corresponding to the correct axes. The SPI controller has to send two 8-bit words (read instruction and register address) to the accelerometer and wait to read from the accelerometer. This value is then sent to the BallMotion module which is sent through a MUX to convert it to acceleration for ball motion calculations. At the end of reading from the y-axis, the next state goes back to Idle and starts the process over again.

**Results:**

As apposed to the original timeline, this is how the project progressed:

*Week 1 (April 2nd):*

* *Maze Initiated, Output to VGA*
* *April 4th – Revised Project Proposal*

*Week 2 (April 9th):*

* *Ball sprite created*
* *Moving Plane with buttons*
* *Begin with acceleration (buttons only)*

*Week 3 (April 16th):*

* *Status Review*
* *Finish Acceleration of ball from plane (still buttons)*

*Week 4 (April 23rd):*

* *Collisions not working*
* *Ball movement has to be rewritten*
* *SPI implemented (replaces buttons for acceleration)*

*Week 5 (April 30th):*

* Finalize
* Presentation
* Written Report Due

We remained on schedule until around week 3 or 4 when collisions became an issue. The original code had the ball move more than 1 unit at a time which made the ball move extremely smooth which we were happy with. Unfortunately, this made wall collision difficult. To fix this, we had to go back and change the ball movement, so it changes the update rate (universal binary counter) instead of how many spots it moves per tick.

The SPI controller was not started until later because it was expected to not be finished in time. To our surprise, and with a little help from a classmate, we were able to get the SPI module given from the book as an example and a controller built to use the accelerometer. This was a big accomplishment. We did not have time after that to finish any other features, so we figured it was a good stopping point for the project even though we wanted to still add more.

**FPGA Utilization:**

![A screenshot of a cell phone

Description automatically generated]()

**Figure 3: FPGA Utilization Table**

**FPGA Timing:**

![A screenshot of a cell phone

Description automatically generated]()

**Figure 4: FPGA Timing Chart**

**Conclusion:**

This was a great project for us to use all the tools learned from the entire course. This includes everything from synchronous flip-flop statements to FSMs made from case statements. The SPI module was definitely the most challenging part of the project and took quite a bit of time to get the timing right. This was a great example of something that would have been much easier to get working with a microcontroller than in an FPGA due to timing. With microcontrollers, a simple delay can be used, and the code executes in sequential order making timing much easier. Instead, this required much more thinking involved to loop between states in a state machine.

If we were to go back and do the project over, we would have fixed the ball motion so that it never moved more than one unit at a time. This would have saved us probably a week of troubleshooting and chasing different solutions that ended up failing. Regardless, it was a good learning experience to know when to give up on a solution and start over again instead of wasting time trying to fix something to save time in the long run.

**Acknowledgements:**

* Lab 9 (PacMan) was leveraged heavily in the project for map design and VGA implementation
* SPI Module provided by the author (code examples) was a starting point for our SPI Controller to communicate with the accelerometer on the FPGA board
* Bryce Tabbut helped us by letting us know that before reading from the accelerometer it needs to be set to measurement mode from standby. We missed this part in the datasheet which means Bryce saved us a lot of time troubleshooting

**References:**

* Chu, Pong, FPGA Prototyping by SystemVerilog Examples, Wiley 2018, ISBN 9781119282662