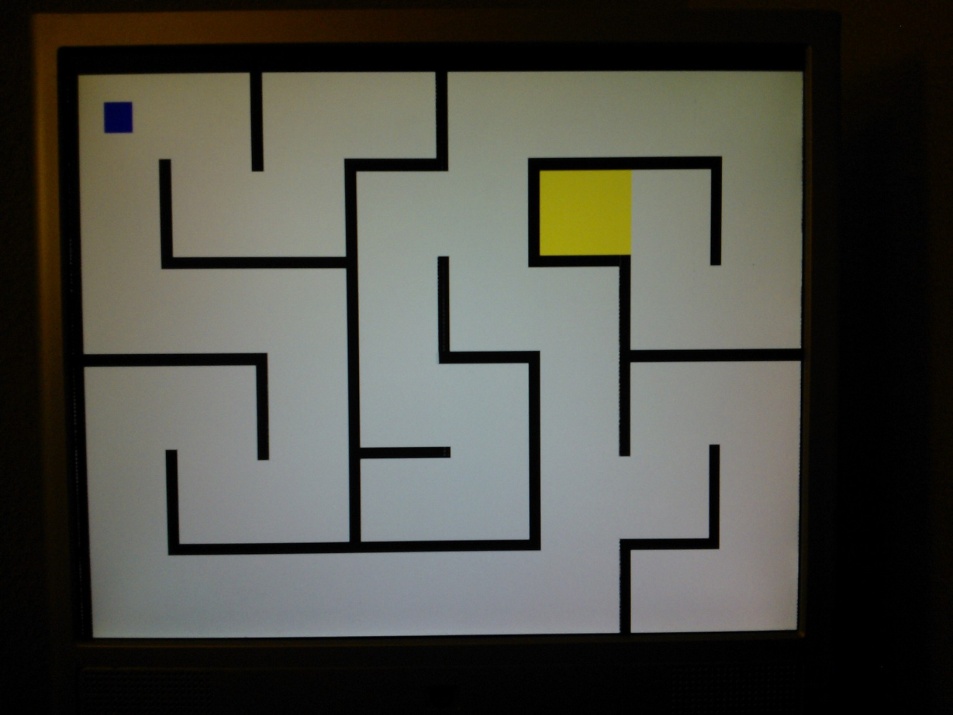
|  |
| --- |
| **Simple Maze Game using the Nexys3 FPGA** |
|  |
| **Uffaz Nathaniel** |
| **12/11/2013** |
|  |

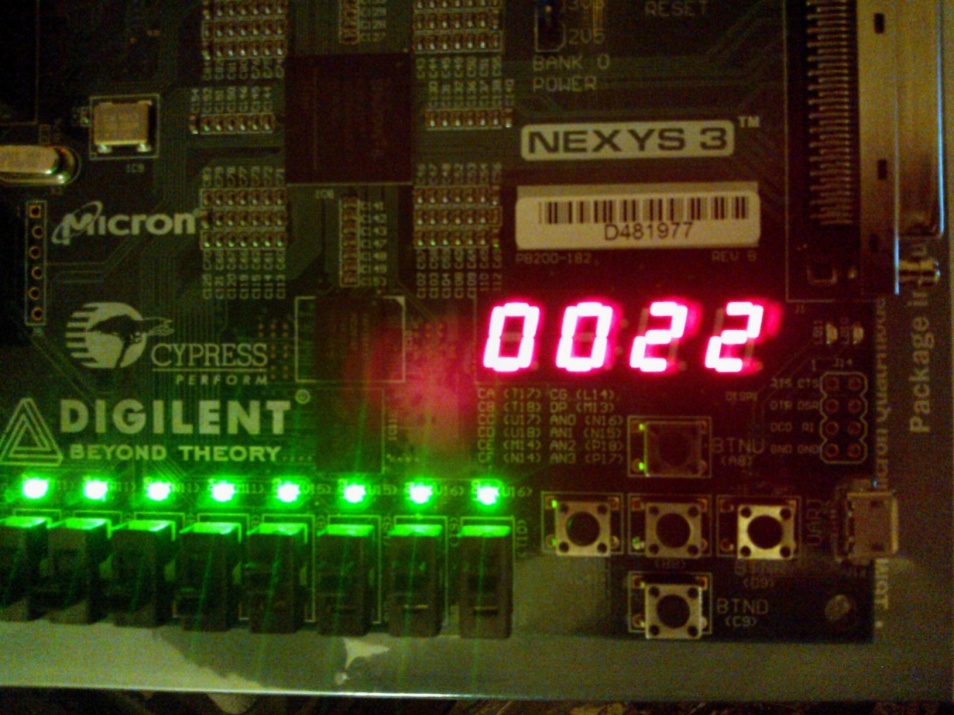
# Project Abstract

For this project, I built a simple maze game using the Nexys3 FPGA that outputs to a screen a preloaded maze along with the destination to reach and a square box indicating the user’s current location. The user is able to use the directional buttons on the FPGA to move up, down, left, or right. The objective of the game is to navigate from a starting location and find your way around the maze to the specified location—indicated by a yellow box. The game is smart enough to recognize collision such that the user doesn’t go wherever they please. *Figure 1* shows what the user sees when the game is first loaded.



**Figure 1 –Output when the game first loads**

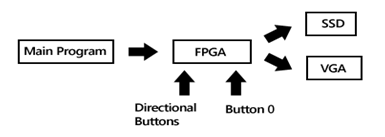
Furthermore, the user is able to reset the game by pressing button zero. There is also a way for the user to keep track of their score—the scoring is based on how quickly the user completes the game (in seconds) and it is displayed on the seven segment display on the FPGA.



**Figure 2 – Seven-segment display showing elapsed time**

# Design Implementation

The design implementation of the project is very simple. All output to the screen is done via a VGA connection. Directional buttons on the Nexys3 board are use to control movement and the middle button is used to perform a reset. Lastly, the seven-segment display (SSD) is used to display how long it takes a user to complete a level. *Figure 3* shows a high-level view of the game design.



**Figure 3 – High-level view of the design**

## Game Logic

*Figure 4* shows a modular view of the logic implemented on the FPGA. The *Logical Layer* is responsible for handling the input from the user as well as determining which RGB vales should be sent to the screen. The *Logical Layer* interacts with the *VGA Controller* to determine the current pixel coordinates being painted which it uses to detect collision and perform other tasks such as movement. The *Logical Layer* also use the pixel coordinates provided by the *VGA Controller* to obtain RGB values which the *Graphics ROM* houses. The *Graphics ROM* serves a crucial role in that the *Logical Layer* uses it extensively for various tasks such as collision detection and what should be output to the screen. Lastly, the output generated from the *Logical Layer* drives the VGA and SSD ports.

**Figure 4 – Interaction of Verilog modules**

## Graphics ROM

The graphics ROM is essentially a long multidimensional register file that is initialized to the values of the image and has various read ports. This file is generated by a console application I wrote in C#.NET. *Figure 5* describes the process used to generate this file.

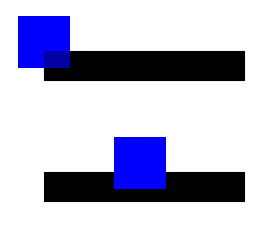
**Figure 5 – Process used to generate Verilog Image ROM file**

To generate the Verilog image ROM file, the image is first loaded into Matlab and the RGB channels are extracted. These extracted channels are then saved to a CSV file which I then run through my console app. Since the original image was a 24-bit image, each channel contains 8-bits. The Nexys3 board is uneven and has 3 bits for the red and green channels and 2 bits for the blue channel. Inside the console application, color values are shifted such that the most significant bits of color are used. A standard Verilog template I wrote is used and after all the parsing is done, a Verilog file is generated.

# Problems/Improvements

## Bug 1

Collision detection is partially functional because intersection detection is not implemented. *Figure 6* shows what I mean by this. So if you are playing the game, if you approach an edge like shown in Figure 6, you can essentially move horizontally and up as long at the detection points (show in *Figure 7*) do not overlap the black pixels. Same concept applies to vertical bars.



**Figure 6 – Intersection of two objects is not tested. User is able to move horizontally and up in the game but not down.**



**Figure 7 - The or Von Neumann neighborhood is used for collision detection**

I incorporated intersection detection but the response from the FPGA was really unpredictable. I believe this is due to the delay/timing issues in the combinatorial logic and the mismatch arising from the adder used to calculate the offset. It was too unpredictable so I decided not to incorporate this feature.

## Improvement 1

Although the image ROM I have implemented is robust and behaves exactly as I intended, it doesn’t really scale. There are a few reasons for this. 1) On a 800x600 resolution, the file size is currently around 30 MB and it takes about 20 minutes to synthesize. I increased the resolution to 800x1200 (2 pictures) and after 4.5 hours of waiting for it to synthesize, I gave up. 2) PROM is limited in its capacity and even a small resolution image with a lot of detail can be too big for “Place and Route”. In a simple image like the maze, I don’t have a lot of detail so the synthesizer is able to significantly reduce the ROM code by employing a multiplexer. 3) Since a lookup is necessary, RAM is perfect for place to store this type of information.

# Testing

Testing for this project was primarily done by playing the game and checking for the intended behavior. Simulation wasn’t really used because I really needed to monitor if image bits being output were correct. It is a little hard to take data from wave outputs and make a visual representation in my head.

# Results/Conclusion

Overall, this project was a huge success. Although I had originally intended to use the ARM Cortex, I ran into serious issues trying to figure out the memory mapped IO and getting the memory controller to work properly. I spent a significant amount of time trying to get the Cortex to do a full image output without the text console. Even though I didn’t employ it in my project, the amount of time I spent debugging/fiddling with it is a huge learning benefit.

Despite all the failures and problems I’ve had, I was able to write pure Verilog code to do what I needed; and at the end have a functional product. This project was a great learning opportunity and so I consider it a huge success!