Logic Master

Jerad Inman, Garrett Bondy, Ryan Peck
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: jdinman@oakland.edu, glbondy@oakland.edu, rpeck@oakland.edu

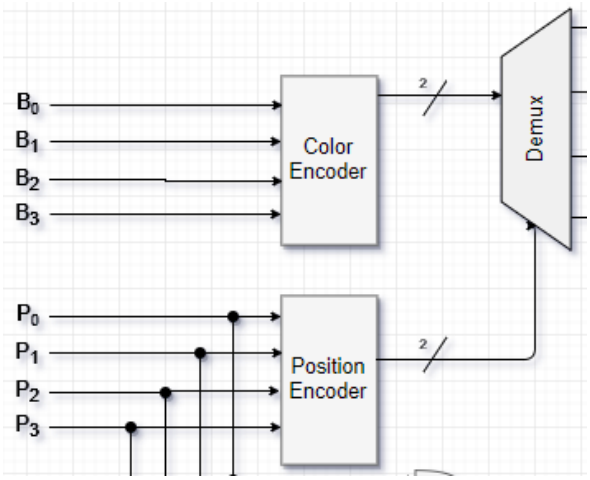
Abstract-This project was based on the board game Mastermind. In Mastermind, the master input a sequence of four colors. The guesser then tries to guess the sequence that the master selected. Pegs display whether the guess was correct or not. For every correct guess in the correct position, a green peg is displayed. For every correct guess in the incorrect position, a yellow peg is displayed. Our project took this idea and digitalised it. The purpose of our project was to create a simple and fun game using techniques taught to us in class and lab. We discovered that creating a comparator is very complex, and just because a simulation gives you correct results, it does not mean that the board will work correctly. In the end, we got the project to work on the FPGA board. While it is not the most user friendly game, it still is fun to play and quite hard. If anyone else wanted to design a similar game, we would recommend to show the guesser's previous guesses, so they have a better chance of guessing the master's sequence.

Introduction

This report will cover the design and the application of our project, Logic Master. The report will contain a description of each of the modules used within the project as well as how that component interacts with the rest of the system. The motivation behind creating this project was to use our knowledge of VHDL from the class, and apply it to a game that is both intuitive and accessible to a user. The game uses many different topics from the class, including decoders, encoders, registers, a demux, equality detectors, and a FSM. Topics that we learned on our own for this project include the comparator and the pmods of the board. Logic Master will use the onboard switches of the FPGA board, external LEDs, a seven segment display, an external three speed switch, and push buttons. The push buttons and switches will be used by the user to input their selections and set the game mode. The seven segment display is used to display the user's choices, and the external LEDs are used to show the results of the players outputs. The application of this project is to provide a game that not only can pass time, but is mentally challenging and intuitive to play.

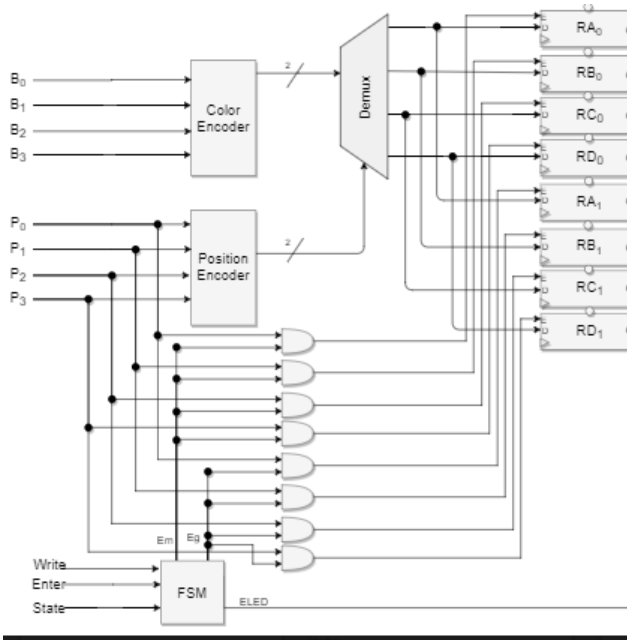
Methodology

A. Inputs



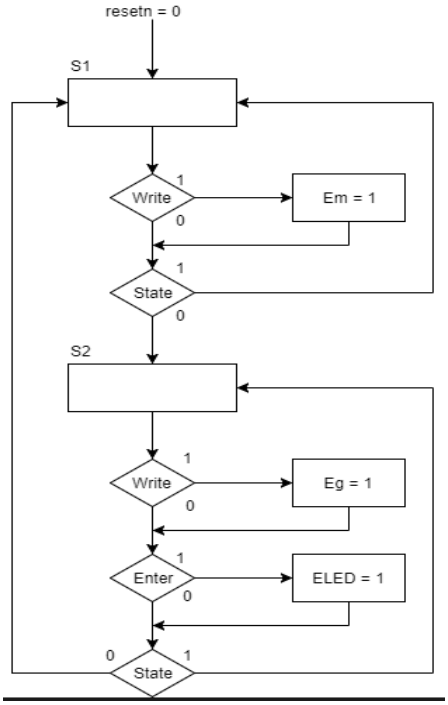
The inputs shown above go into two encoders. The top encoder determines what position the choice is in by reading 3 signals sent by a 3-speed switch. The bottom encoder selects which color was chosen based on the button pressed. The top encoder is a 4-to-2, and the bottom is a 3-to-3. The template for the encoders were created using the file from Dr. Llamoca's website, [1].

B. Demux and Registers



The demux follows the encoders. The color encoder sends its output signal to the input of the demux. The position encoder is used as the select line. Based on the select line, the demux then sends the input signal to the appropriate register. The registers, RA0, RB0, RC0, and RD0, are for the master sequence. The register, RA1, RB1, RC1, and RD1, are for the sequences of the guesses.

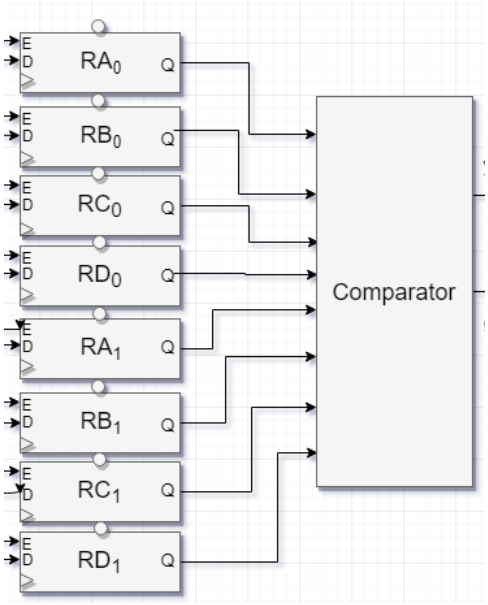
C. FSM



The main function of the FSM is to enable and disable registers. When in State 1, a 'Write' signal is encountered. If 'Write' is 1, then the 'Em' signal becomes 1. This enable signal goes to an and gate with the position inputs to enable only one of the top registers where the signal will be stored. This will continue until the signal 'State' becomes 1.

Once 'State' becomes 1, then the FSM goes to State 2. When in State 2, another 'Write' signal is encountered. If 'Write' is 1, then the 'Eg' signal becomes high. This will then be anded to the position vector to enable one of the the bottom registers. After all of the registers have stored data, the FSM moves to an 'Enter' variable. If this is 1, then it causes the 'ELED' signal to become 1. Finally, if 'State' signal returns to 0, then the FSM returns to State 1.

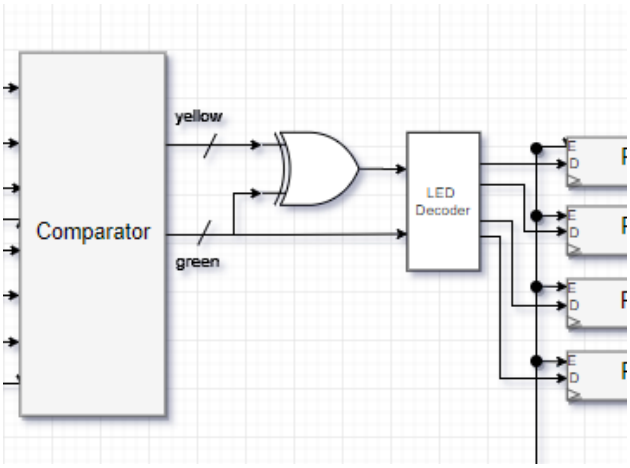
D. Comparator



The main function of the comparator is to analyze the master sequence and the guess sequence. The four colors of the master sequence and the four colors of the guess sequence are sent through equality detectors to produce 16 internal signals. These signals are then sent through a sequence of combinational logic to determine which values are correct in the correct place, and which are correct in the wrong place. Due to possibility of repeating colors, the logic is programmed to prioritize the correctly placed values and eliminate repeated misplaced but correct values. This function then outputs 2 four bit signals, where one indicates correct

value (yellow) and the other indicates correct value and correct place (green).

E. Outputs



The outputs of the comparator represent the unfiltered signals for the yellow and green led indicators. Initially, the yellow signal is filtered through an xor gate to remove unintended yellow indicators. Now, these 2 signals are send into a decoder to be converted, combined, and sent to 4 LED output registers. Finally, these registers can be enabled when in State 2 and 'Enter' is pressed. When enabled, the LED indicators light up in a combination of 4 yellow or green LEDs based on the guess validity.

Experimental Setup

In order to test this program we were able to design a test bench that stores different values and tests guesses against those values. Some of the basic code from our test bench can be seen below. The code on the right is is the master state, as can be seen by st = '0'. This is so that the values are saved in the registers. The code on the left is in the guesser state and is cycling through a series of guesses. In order to test the basic program on the board we used the on board switches instead of the switches attached to the pmod, and the onboard LEDs instead of the ones we ultimately connected to the pmod. This allowed us to verify the code without worrying about possible issues existing outside of the scope of the board. In doing this, we were able to find that the program worked as planned. Then we were able to focus on implementing the peripherals after the fact.

```
stim_proc: process
begin

resetn<='0';
wait for 100ns;
resetn<='1';

st<='0';
a<='1';
b<='0';
c<='0';
r<='1';
g<='0';
bl<='0';
y<='0';
wait for 2*T;
a<='0';
b<='1';
c<='0';
r<='0';
g<='1';
bl<='0';
y<='0';
wait for 2*T;
a<='0';
b<='0';
c<='1';
r<='0';
g<='0';
bl<='1';
y<='0';
wait for 2*T;
a<='0';
b<='0';
c<='0';
r<='0';
g<='0';
bl<='0';
y<='1';
wait for 2*T;
```

```
-------------
st<='1';
wait for T;
a<='1';
b<='0';
c<='0';
r<='1';
g<='0';
bl<='0';
y<='0';
wait for 2*T;
a<='0';
b<='1';
c<='0';
r<='1';
g<='0';
bl<='0';
y<='0';
wait for 2*T;
a<='0';
b<='0';
c<='1';
r<='1';
g<='0';
bl<='0';
y<='0';
wait for 2*T;
a<='0';
b<='0';
c<='0';
r<='1';
g<='0';
bl<='0';
y<='0';
wait for 2*T;
enter<='1';
wait for 10*T;
enter<='0';
a<='1';
b<='0';
c<='0';
r<='0';
g<='1';
bl<='0';
y<='0';
wait for 2*T;
```

Results

This project ended up working as we originally planned, and an image of the timing simulation for our project can be seen at the end of the report. We were able to successfully implement the external portions of the project to use the three speed switch to select what position we used and the external LEDs to display the results of the guesser's choice.

As can be seen in the timing diagram, we were able to successfully store values to the registers. We then inserted a guess and saw if the value matched that of the original input. As also can be seen in the variables r0 - r3 shown above, a correct guess will yield a '1' in the leftmost bit of the the r variable, and will result in one of the green LEDs being turned on. A guess that contains the right color in the wrong place will

yield a ‘1’ in the other bit of the r variable and will lead to a yellow LED being turned on. Unexpected results that we initially received were false output values when testing different sequences. While guessing our values with the switches and push buttons, we were experiencing unexpected values at the output LEDs. We found our problem to be that the value for one of the variables was being changed whenever we were switching between the variables because it was accepting the last pressed value as we switched from one position to the next. In order to combat this issue, we included a read write mode to our FSM, so that the user could see what variable they assigned to each value without overwriting the one variable. Another issue of note was that we created a latch in simulation that did not convert to hardware, which lead to us not being able to change the value of the variable using the FGPA board. In order to resolve this issue we used a register instead of a latch.

Conclusions

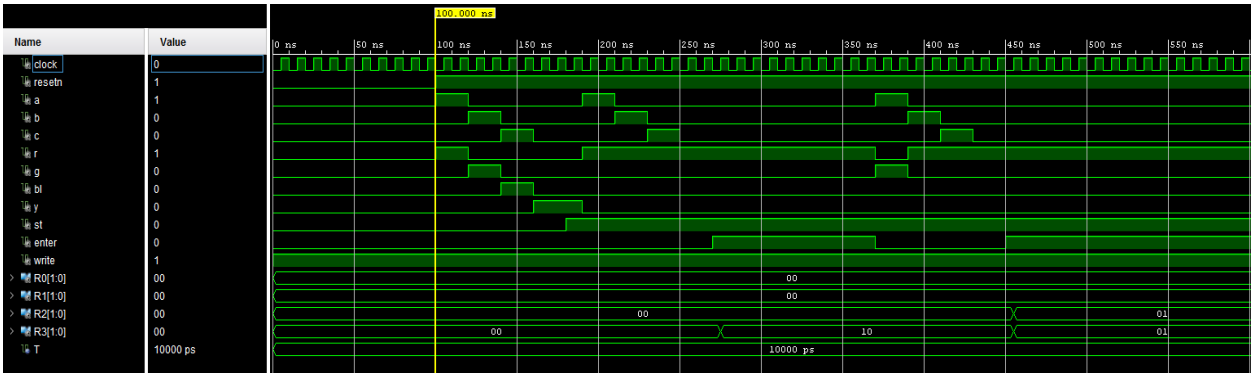
The main take away points from this project is that we were able to produce a working version of our game with the peripherals we originally planned to use. If we were able to expand on this project and had more time to further develop it, there are several changes we would implement. The first change we would make would be to use RGB LED’s for the output, just because it would simplify the design and the code is easily capable of running them. Another change we would make would be to improve the packaging of the game. We would create a more user friendly system that would not require as much instruction to play. Other areas of expansion could be to include a mode in which the guesser can review previous guesses, which would improve the quality of gameplay, and make the game more similar to the original Mastermind game. This could be done by implementing more registers into the system that would store the guesses. In order to do that, we would have to cap the number of guesses that the user gets. This would require a counter to keep track of the number of turns taken by the user. For the purposes of what the original project was to include, we were able to accomplish everything we planned with the project. We got the peripherals to function in coordination with the system and produce the desired inputs and outputs for the system.

References

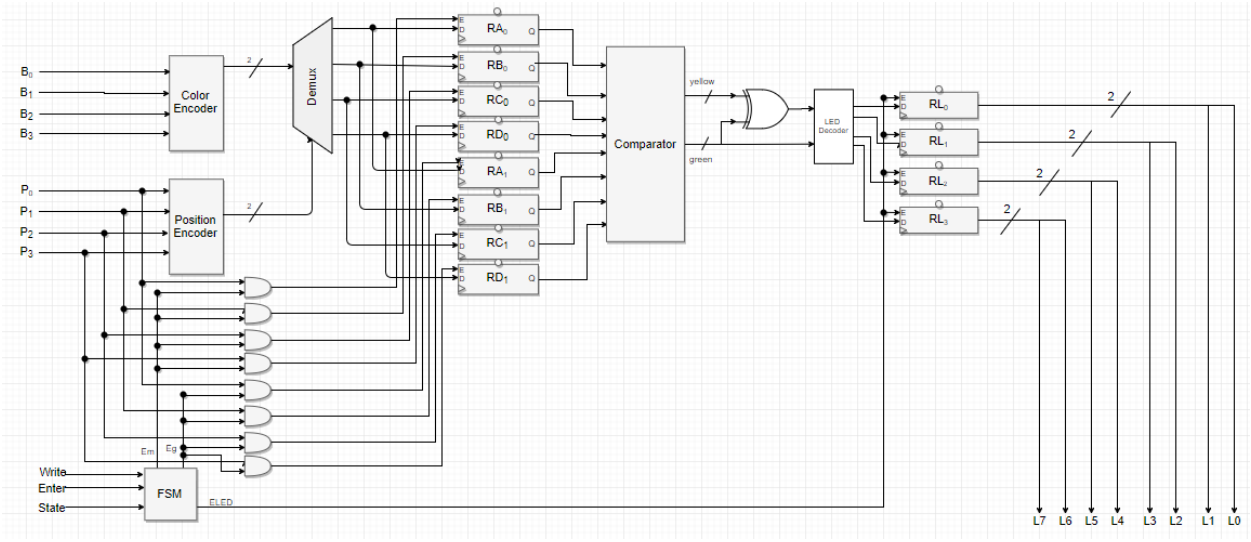
[1] Llamocca, Daniel. "VHDL Coding for FPGAs." VHDL Coding for FPGAs. N.p., n.d. Web. 7 Dec. 2017. http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html

[2] "Nexys 4 DDR Reference Manual." Nexys 4 DDR Reference Manual [Reference.Digilentinc]. N.p., n.d. Web. 7 Dec. 2017.

Timing Simulation



Overview


B0
B1
B2
B3
Color Encoder
2
Demux
P0
P1
P2
P3
Position Encoder
2
RA0
RB0
RC0
RD0
RA1
RB1
RC1
RD1
Comparator
yellow
green
LED Decoder
RL0
RL1
RL2
RL3
2
2
2
2
Write
Enter
State
FSM
Em
Eg
ELED
L7
L6
L5
L4
L3
L2
L1
L0