Lab Final Report

Evan Cole
School of Electrical and Computer Engineering
Oklahoma State University
Stillwater, Oklahoma
evan.cole@okstate.edu

Landon Brown
School of Electrical and Computer Engineering
Oklahoma State University
Stillwater, Oklahoma
landon.d.brown@okstate.edu

I. INTRODUCTION

The purpose of this lab was to put together all that we learned throughout this year. The goal was to get a version of Conway's game of life to work. We were given a few things to help accomplish this goal. A version of the game's rule, A few tools like a mux, a flop. And a HDMI driver for the last step. Conway's Game of Life is a wonderful project that combines almost perfectly everything we have been working on so far in the lab and puts it into one goal. There are many things we had to pull from our prior labs that helped tremendously in the completion of this final lab. Not only did we need combinational and sequential logic from previous labs, but we also needed to look at files, test benches, and other items to finish this lab. We needed to use a version of an FSM and a clock divider that we learned from lab 3. We used multiple things that we learned from lab 2 including shifting.

II. BASELINE DESIGN

For the baseline design, we need to figure out what we need to do to begin with. With this the goal is three options for the game, the first option is the original seed. The second option is the Evolved Seed that loops infinitely, and then the last option is the LFSR that generates a new seed to be used. After figuring all of that out, all we needed to do was put together all of the components that we needed to put together to accomplish this goal. For the first part, we need to use a mux that looks at an input to swap the output so we can easily swap between them. Then we need a flop to act as a register on the clock cycle. To change evolved that comes out of the rule Datapath and then force it into one of the mux inputs. Then LFSR module then finally a FSM. to get it all lopping and working together. The LFSR is done in a separate module that takes in the seed and then shifts all the bits left and with 64,63,61,60 and then xors together.

III. DETAILED DESIGN

To start the design we need to choose the mux that we want to use for this design we need to choose Mux3 or the one that can take in 3 inputs. The input of this mux we need to put in is seed, Seed_evovled, and LFSR_seed, and an input that we named active determines which we are using. This outputs a 64-bit number called comb that then gets moved forward after the mux. After that, we run it throughout the data path and we get comb_evolved after we get that we run it through a flop to swap the comb_evolved to grid Evolved then the FSM does its work with Active from are mux is controlled by the FSM which takes in the input "A" And loops between three states sending out the output active S0 witch just looks at your input and goes to the difference states or loops S0 if *A* set at 00, S1 that turns it in LFSR mode and then S2 with grid evolved also the reset is sent it to S3 which sets it to zero and then goes back to S0. Figure 1 below shows our FSM, and figure two shows our diagram of how everything connects to each other.

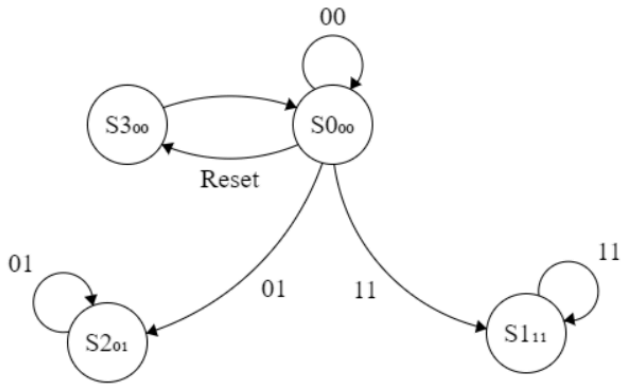


Fig. 1

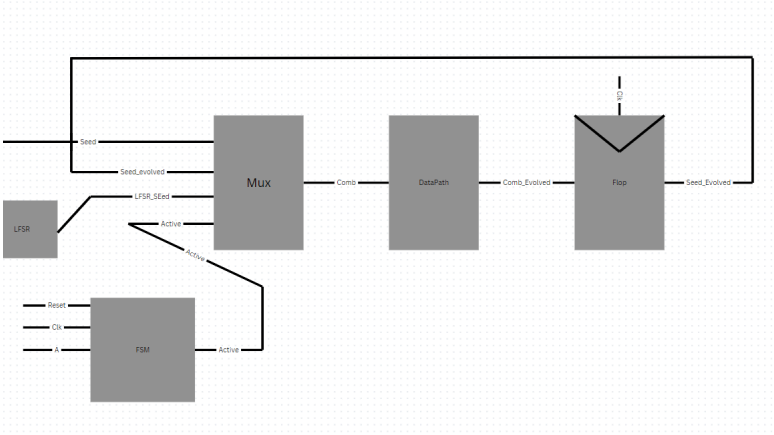


Fig. 2

IV. TESTING STRATEGY

Now we move on to testing. To do that we need to set up a do file and a test bench. This seems daunting at first, but all it really took was some manipulation of the do and test bench files from prior labs, and it ended up working for this lab as well. Once those are set up, we write some lines in the test bench to write the output to a .out file after every clock cycle and we change the inputs and see if the set 64 bit number changes and it does as shown in figure 3.

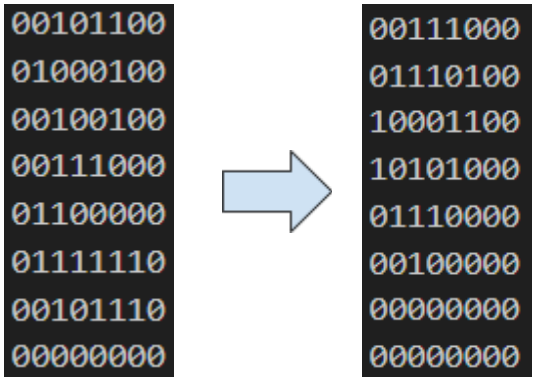
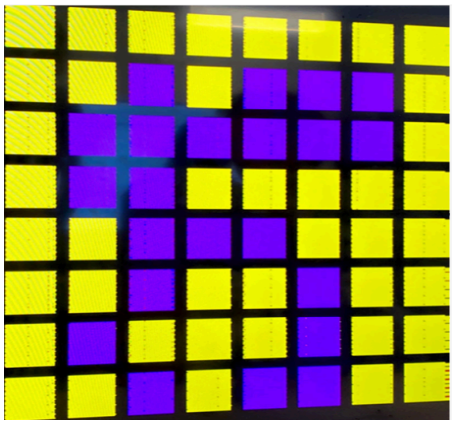


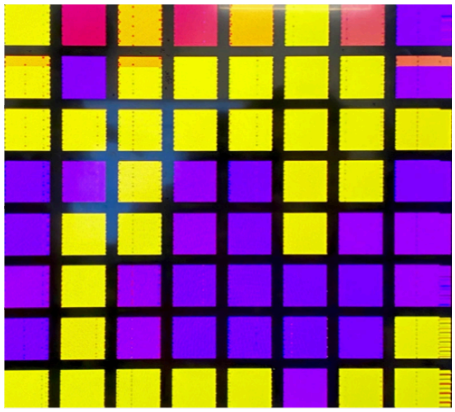
Fig. 3

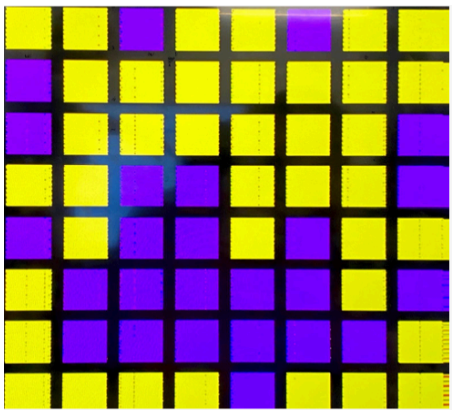
This means that our code is working as intended and the 8 by 8 grid (which is actually just our 64 bit number) is actually following the game rules and changing over time. This was done by running it through the model sim software. To change what mode we use we change the a value in the tb. After we do this and verify that our design is working as intended, we can move on to vivado and the FPGA.

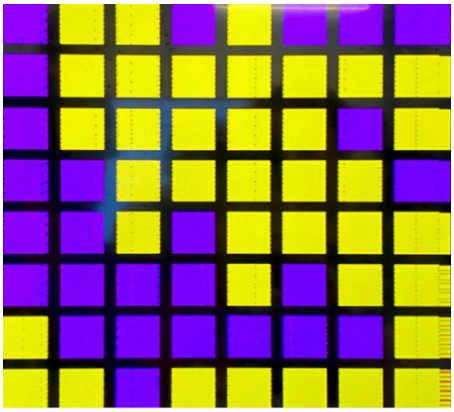
V. EVALUATION

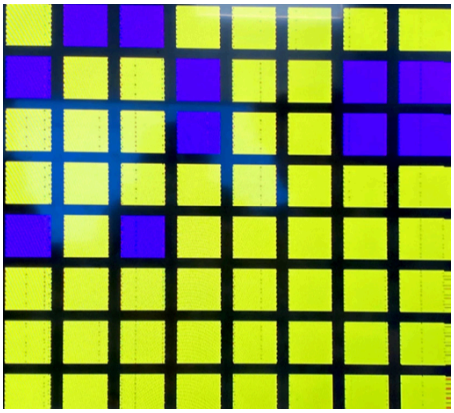
Once it's ready for the board we upload it and set up the inputs and imports. Because the basic clock speed moves much too fast to actually show any output change, we needed to include our clock divider from lab 3 so we could actually see what was going on. We have the first two switches acting at are *A* and reset as the button 3. After that we confirmed it was working. We changed the color of the alive and dead blocks by looking at the HDMI code and changing the 8 bit hex value to the purple one for alive and yellow for dead. Below are pictures of our final output.

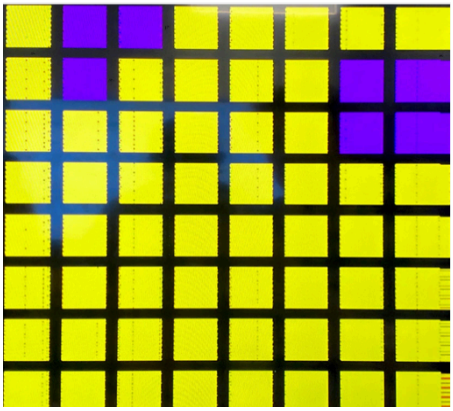












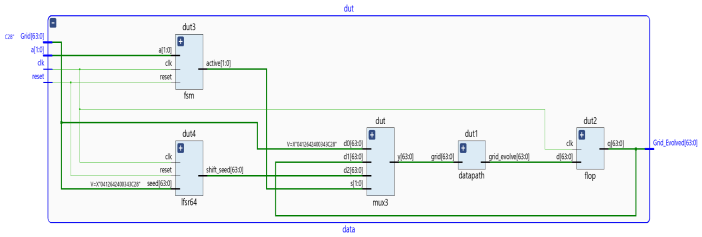


Fig. 4

According to the power report summary that we got from vivado. The entire project takes about 50.5 watts to run on the board. That is about the same as an average laptop uses. It has about a 16 and a half nanosecond delay, and it uses roughly 300 registers.With all of this done we can get the final schematic from vivido you can see it in figure 4 that it looks familiar to are digram in figure 2