CS223 – Digital Design

Cellular Automata (The Game of Life)

Section 4

Mannan Abdul

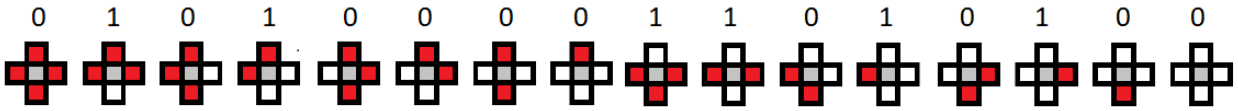
21801066

30/12/2019

Introduction:

In this project, we visit the idea of cellular automata and how they work. To implement the project, we divided it into 2 parts. In the first part, we take a 64 bit input from the user using some switches, each value was shown in its hexadecimal form on the 7 segment display while being entered and afterwards, a partial 16 bit representation of the total 64 bits is shown through the 16 leds. These can be changed to show other parts of the 64 bit value by using 2 switches. In the second part of the project, we show this initial value that we take in a 8x8 Dot Matrix Module on the Beti Board. We then use the rule we calculated according to our ID numbers and apply it to these cells to represent a cellular automaton. These 2 parts combined make up our game in which the objective is to get as low a score as possible as each button press to apply the rule will increment the counter we keep by one.

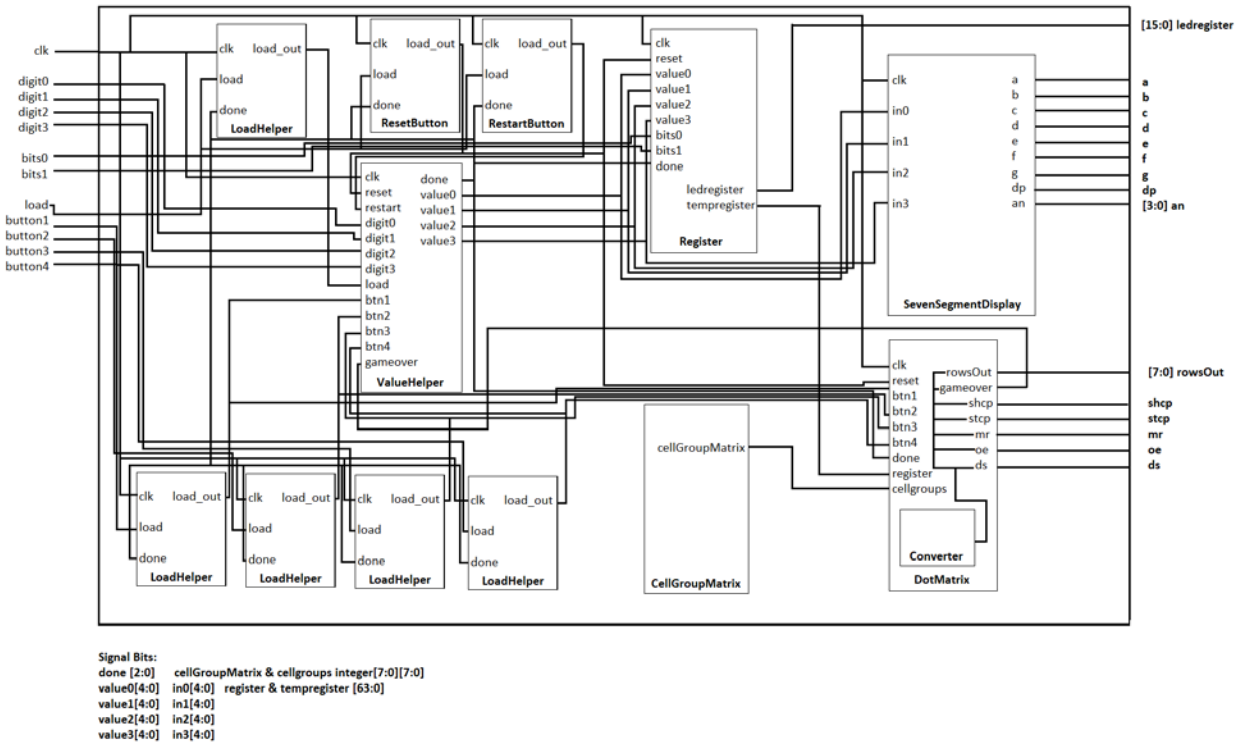
The rule:



The cellular grid:

1	2	1	2	4	3	4	3
3	4	3	4	1	2	1	2
1	2	1	2	4	3	4	3
3	4	3	4	1	2	1	2
2	3	2	3	2	3	2	3
1	4	1	4	1	4	1	4
2	3	2	3	2	3	2	3
1	4	1	4	1	4	1	4

Block Diagram:



Detailed Explanation:

- **Flow of the Program**

We start by letting the users give values that will be saved using 4 switches, this allows them to specify a value between 0 and 15 which basically lets the user specify hexadecimal values that are then stored into the register. The active digit blinks with a period of 1 sec and the middle button is used to move on to the next digit when the value is specified and we want to move on to the next one.

After we have taken 16 values in total, we will store them all in a register which will be of 64 bits to hold these values. That is because we are storing the binary equivalent of the values chosen in the register. We then show these values in their binary form partially on the 16 leds that we have on our Basys 3 board. We use 2 switches named bits0 and bits1 to specify which part of the values we want shown on the leds.

We start the game after we have stored all of the 16 values and we show these values on the 8x8 Dot Matrix display module. We then use the 4 buttons specified

button1, button2, button3 and button4 to apply our rules on the 8x8 grid until the user wins the game.

- **Modules**

AssignValues is the top module in this program, this module takes all the inputs and outputs specified in the constraints and then distributes them to where they are needed in other modules. It also acts as an intermediate between 2 modules and helps pass values along between 2 modules if it is necessary.

Inputs: clk, digit0, digit1, digit2, digit3, bits0, bits1, load, button1, button2, button3, button4

Outputs: a, b, c, d, e, f, g, dp, [3:0] an, [15:0] ledregister, [7:0] rowsOut, shcp, stcp, mr, oe, ds

LoadHelper is a module used to damp a button's output. Usually without damping, the clock checks button presses so quick that it isn't much use to us, so this module only gives a 1 for a button press when the button has been pressed for half a second. This module is used to damp, load, button1, button2, button3 and button4. The module also only works for these button if done is less than 4, done being the count of how many times the values in the 7 segment display have been loaded into the register.

Inputs: clk, load, [2:0] done

Outputs: load_out

ValueHelper is the module where everything related to readying values is done. Values taken from the 4 switches are compiled together here and then sent on to the clock divider to be shown in their hexadecimal form. The active digit that blinks is also done here, using a count, the module gives the value the original value for 0.5 sec and then the module gives a value which translates to all the leds being off for a digit in the 7 segment display for the other 0.5 seconds. It keeps going between the 2 so as to give the impression that the digit is blinking. The score count

of the game is also kept in this module and the final score's blinking is shown the same way as mentioned before albeit with a 0.5 sec period. For smaller cases like what values should be displayed on the 7 segment display when the game is reset or restarted, this module is where all this is handled. This is also the module that keeps track of whether all 16 values have been taken by keeping a count that increments every time 4 values are taken, these are the 4 values that are shown on the 7 segment display at the time. One last thing done in this module is that we also keep a denary count of the score which is shown by taking the modulus of the score and showing it on the 7 segment display for all 4 digits.

Inputs: clk, reset, restart, digit0, digit1, digit2, digit3, load, btn1, btn2, btn3, btn4, gameover

Outputs: [2:0] done, [4:0] value0, [4:0] value1, [4:0] value2, [4:0] value3

SevenSegmentDisplay is the module where we take 4 input values and then we show their representations on the 7 segment display. The only change made to this module from the one provided in the labs was adding a few lines to accommodate hexadecimal values.

Inputs: clk, [4:0] in0, [4:0] in1, [4:0] in2, [4:0] in3

Outputs: a, b, c, d, e, f, g, dp, [3:0] an

Register is the module where we store all the values from the 7 segment display, 16 bits at a time, as we input the values from the switches. This module uses the done output of ValueHelper module to know when 4 values have been input into the 7 segment display. As soon as they have, this module will add the binary equivalent to the register. This module is also where we handle which part of the 64 bit value to show on the leds, this is specified by 2 switches, bits0 and bits1, that can be changed any time to show the value.

Inputs: clk, reset, [4:0] value0, [4:0] value1, [4:0] value2, [4:0] value3, bits0, bits1, [2:0] done

Outputs: [15:0] ledregister, [63:0] tempregister

CellGroupMatrix is the module in which an integer array of 8x8 is made to store the values that each cell is assigned according to our Bilkent IDs. This integer value is between 1 and 4. The purpose of this 8x8 matrix is to help us identify if the rule is applied on a particular cell or not.

Outputs: Integer cellGroupMatrix[7:0][7:0]

DotMatrix this module is where we deal with everything related to the dot matrix display module. We initialize the dot matrix by providing the dot matrix with the value of the register so our initial value is what the dot matrix is first filled up with. We then use the 4 button inputs to check for our rule and if it is being applied on every cell, we change the state of the dot matrix as a whole accordingly. We also have reset as an input and if it is true, then the dot matrix will be empties and if the game is only restarted, then we will input the values of the register to the dot matrix again. This module also checks if whether the dot matrix has been empties and is all off in which case it will assign 1 to the output gameover. Which we will use to determine the behavior of other modules, namely, the blinking of the final score in the ValueHelper module. If we see that the btn1 is pressed, then the module will only apply the rule to the cells which have the integer index 1. The same is done for all the other 3 buttons.

Inputs: clk, reset, btn1, btn2, btn3, btn4, [2:0] done, [63:0] register, integer cellgroups[7:0][7:0]

Outputs: [7:0] rowsOut, gameover, shcp, stcp, mr, oe, ds

RestartButton this module checks if the load button is pressed for 1 sec when done is equal to 4 (which means game has started). If it is, it will restore the game to the initial state of it.

Inputs: clk, load, [2:0] load

Outputs: load_out

ResetButton this module checks if the load button is pressed for 4 secs when done is equal to 4 (which means game has started). If it is, it will restore the program to such a point that it will take all inputs again, and the game will start all over.

Inputs: clk, load, [2:0] load

Outputs: load_out

Converter this module was provided in the Project files, its use is to put convert a 8x8 array into the data we see on the dot matrix module on the Beti Board.

Inputs: clk, [7:0][7:0] data_in

Outputs: [7:0] rowsOut, shcp, stcp,mr, oe, ds

Conclusion:

In this program, we have implemented a cellular automaton game. The user can give the game an initial value through the switches which will be the initial state of the dot matrix. After this initial state has been given, the game will start and the user can then interact with the game using the buttons and switches with the aim of getting all the cells to turn off in the least amount of button presses possible.

The main problem was troubleshooting issues with so much code written and since Vivado doesn't give an error if a module is lacking an input or output when being instantiated, it can also be hard to find that particular error.