30/12/2019

CS223 DIGITAL DESIGN
SECTION 04

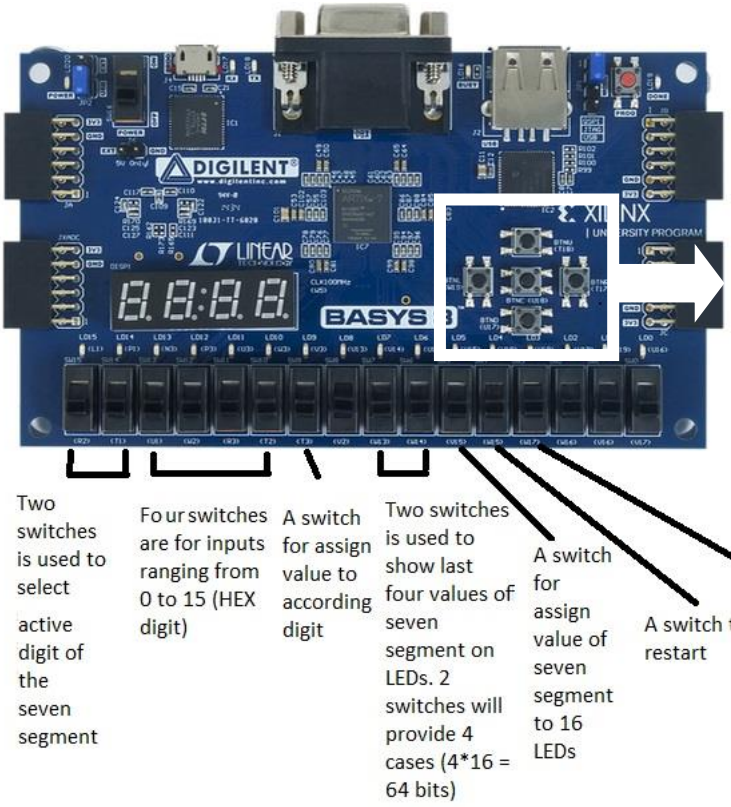
Project Report

Işık Özsoy

21703160

1. INTRODUCTION

This project is an implementation of a game on a Basys 3 FPGA board and BetiBoard using HDL SystemVerilog. We control user interactions by the switches and buttons on Basys 3. 12 switches and 5 buttons are used data input.



Middle button is used to start game (part 2 ends and game starts)

Top button is used to apply game rule to cell labelled with 1

Right button is used to apply game rule to cell labelled with 2

Bottom button is used to apply game rule to cell labelled with 3

Left button is used to apply game rule to cell labelled with 4

Initially, seven segment display on the board shows 0000. User assigns the digits one by one and by using a switch sends the whole number in seven segment display to 16 LEDs. Then it is possible to display last four value of seven segment on LEDs by using 2 switches (2 switches for 4 different cases 4*16 = 64 bits). If user presses the start button, the 64 bits from part 2 will be displayed on 8x8 RGB LEDs on BetiBorad. Then by using 4 buttons, the rule will be applied on the group of cells according to the labelling of cells and the button pressed. The number of the button presses will be shown on seven segment display. When the all cells are off, the game is over and the seven segment will be enabled and show the score for 0.25 second and disabled for 0.25 second. If user turns on the restart switch, the game restarts with the initial state. If the user turns on the reset switch, user starts with part 1 again.

2. BLOCK DIAGRAM

2.1. HLSM Design

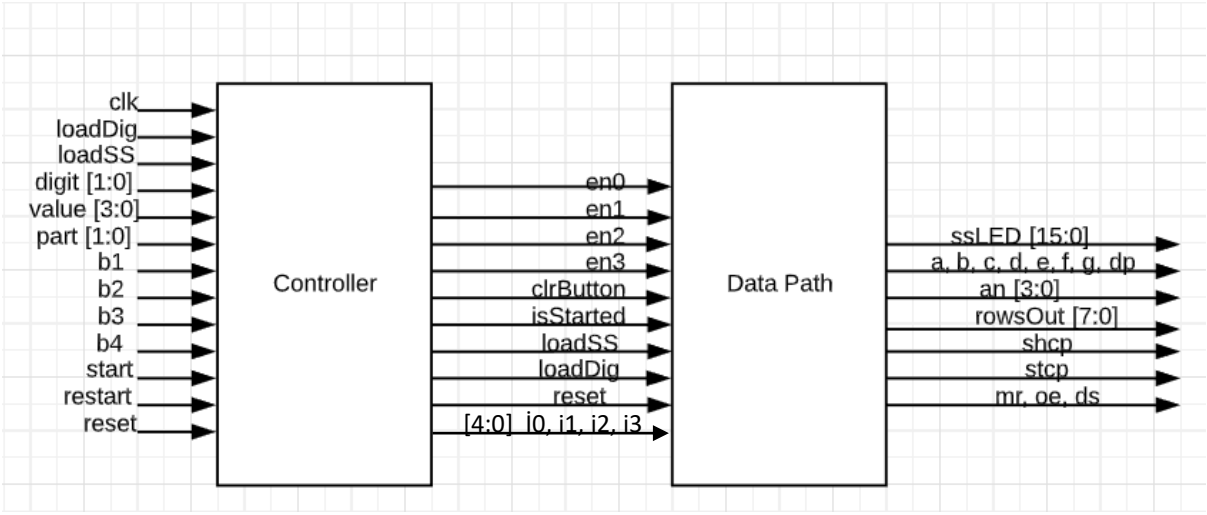


Figure 1: The diagram of High Level State Machine

2.2. Datapath Design

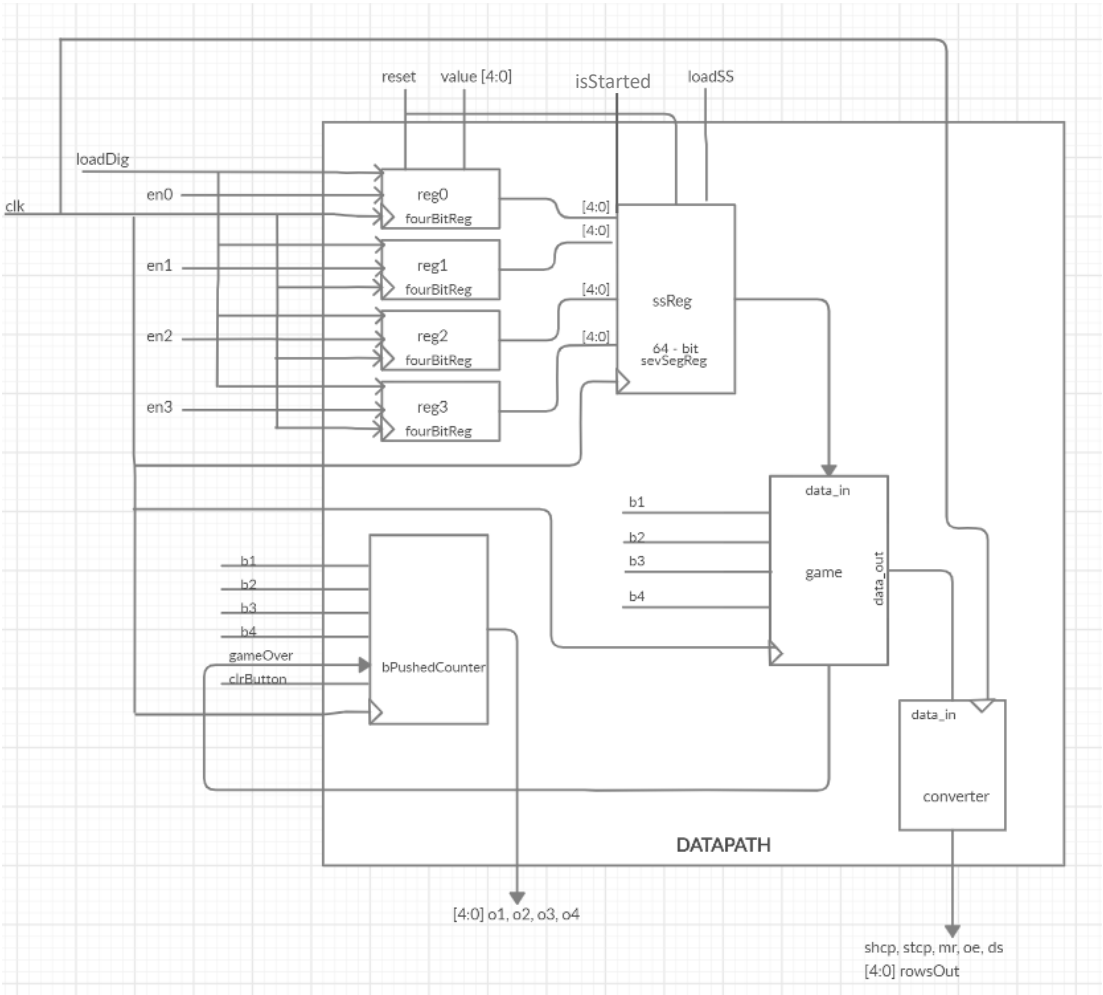


Figure 2: The diagram of the Datapath

3. DETAILED EXPLANATION OF THE WORK

3.1. HLSM Design

HLSM of the game has two main parts:

- Controller
- Datapath

Control unit outputs necessary for the flow. These signals include en0, en1, en2, en3, clrButton, isStarted, loadSS, loadDig, i0 [4:0], i1 [4:0], i2 [4:0], i3 [4:0]. en0, en1, en2, en3 signals determine which digit of seven segment will change. loadDig signal assigns value to selected digit of seven segment in Part 1. loadSS assigns value on seven segment display to last 16 bits of 64 bits sevSegReg in Part 2. When isStartted becomes '1', the game starts, Part 1 and 2 end and part 3 starts. i0, i1, i2, i3 signals are for seven segment display. Seven segment takes its value from those signals. clrButton signals set 0 the counter for counting the button push.

3.2. Control Unit Design

The control unit is an FSM with 4 states. In my FSM design, the init state is the reset. In init state, system waits for the input that starts the game and when start signal comes, the system moves next state, start game state. In this state game starts, 8 x 8 RGB LEDs become on according to the data from previous state (initial state). When one of the buttons is pushed, the system moves to next state, check rule state, in which the rule is applied. Then when the button is released, system goes to the update state. In this state, 8 x 8 LEDs are updated based on the outputs of the game module. If game is over, system passed to game over state. After this step, if user restarts, the game starts with the same initial state and system moves to start game state. If user resets, the system will be in init state.

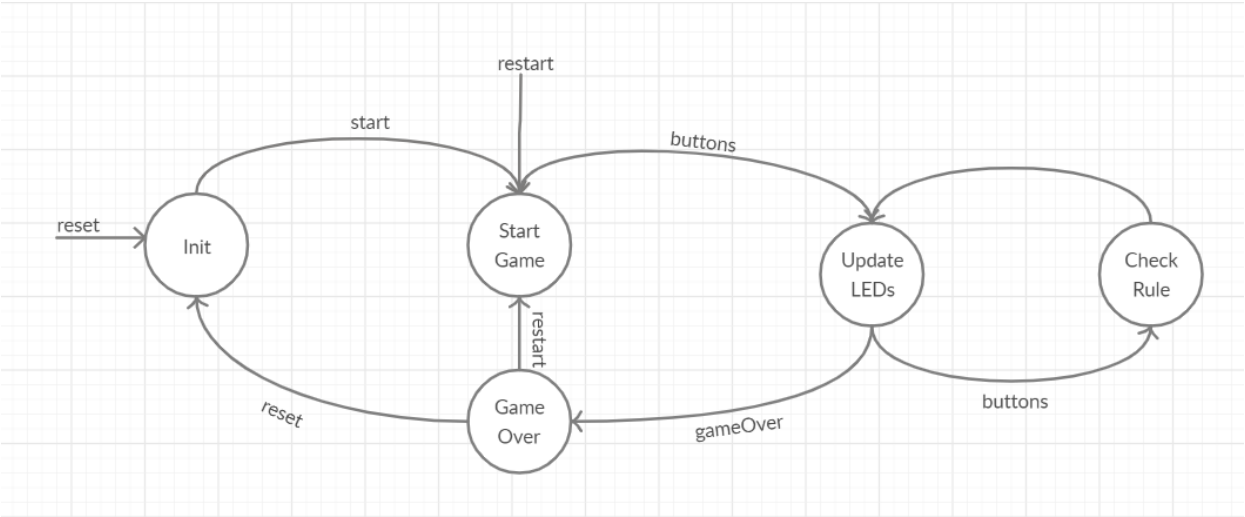


Figure 3: The FSM of the control unit

3.3. Datapath Design

Datapath takes the signal outputted by the control unit and output a, b, c, d, e, f, g, dp, an, ssLED, rowsOut, shcp, stcp, mr, oe, ds signals which correspond to seven segment display signals, 8 x8 RGB LED signals and signals for LEDs on basys 3. Datapath consists of 6 different modules in the top level design.

3.3.1. fourBitRegister Module

This module is for holding the user input for each digit of the seven segment. It takes clk, enable, load and 4 bit value which are provided from user. Enable comes from the controller and it determines the digit of the seven segment display to change. According to the load, the value is saved or not.

3.3.2. SevenSegment Module

Seven segment module is provided by the teacher. In order to make it appropriate for our project some changes are made. One of them is instead of decimal, it is programmed to be able to display digits in hexadecimal.

```
'hA : sseg_LEDs = 7'b0001000;

'hB : sseg_LEDs = 7'b0000011;

'hC : sseg_LEDs = 7'b1000110;

'hD : sseg_LEDs = 7'b0100001;

'hE : sseg_LEDs = 7'b0000110;

'hF : sseg_LEDs = 7'b0001110;
```

It takes clk, four 4 bit value for each digit, isStarted signal to start showing score when the game starts. Furthermore, it takes gameOver signal which shows whether the game is over or not. If the game is over, the seven segment display starts blinking (open for 0.25 second and close for 0.25 second).

3.3.3. sevSegReg Module

This module holds the last 4 values of the seven segment (16 * 4 = 64 bits). It takes clk, clr, load as inputs. In reset state, the reset signal used as clr and 64 bit value is set to zero. Load signal is for adding the current value of the seven segment display is assigned to last 16 bits of 64 bit register.

3.3.4. game Module

The game module determines the 8 x 8 RGB LEDs' outputs. In this module, the specified rule is applied to each cell of the 8 x 8 RGB LEDs. It takes clk, b1, b2, b3, b4 and [7:0][7:0] data_in as inputs. On each button push, the rule applied to input data_in and some leds become on and some becomes off. After those changes the new logic is outputted as data_out which will be shown in 8 x 8

LEDs. gameOver signal is for understanding whether the game is over or not. If all the leds are off, this signal becomes 1 and game finished.

3.3.5. bPushedCounter Module

This is a module for counting the button push. During the game its value is shown in the seven segment display. This module takes clk, clrButton, gameOver, b1, b2, b3, b4 as inputs. gameOver signal can be considered as enable signal, when the game is over this module no longer count. On the restart and reset state,

3.3.6. converter Module

This module is to assign the data_in logic to 8 x 8 RGB LEDs. However, it displays each row in reversed order.

3.4. Testing

I did not write any testbench in order to test my code. I tested it by running it with basys 3 and betiboard. I prepared some cases on paper and decided the expected outputs. Then I compare the outputs that I get from my code with expected outputs. It help me to figure out different problems and make some changes in order to fix them.

4. CONCLUSION

In this project we tried to implement a game by using basys 3 FPGA board. I tried to have a better approach to the problem and optimize the solution as much as possible. I tried to find solution to a problem by dividing them into smaller problems. Instead of doing the project parts one by one, I think about all parts and design my approach according to. Thus, I did not face with any problem because of implementation when I move to next steps of assignment. The hardest part was the writing the array for labelling the 8 x 8 table, this was the converter shows the value in reversed order and I became confused. The one problem which I couldn't solve is that the rule is not applied on each button push. When I pushed the button to move to next state, even if the counter counts the number of push, the rule could not be applied on the each push, however it should work on each button push. Sometimes it worked well but sometimes in order to move to the next state, I needed to push the button more than one times. After this project, I understood the logic of the systemverilog much more better. Although in some parts I got more confused, after discussing the TA and my friends, I figured out those parts as well.

5. REFERENCES

- Seven Segment Display Module
  https://www.unilica.com/index.php?v=album&s=viewer&iid=rxE3Rg3_ppHK3eNw5d2n8ekrW6oWYkZq2-oAWFtIrzs
- Converter Module
  From Unilica, Project Folder