CS223 - SECTION 5
CELLULAR AUTOMATA
SEMESTER PROJECT

ABDULLAH CAN ALPAY
21702686
30.12.2019

Introduction

The aim of this project to learn the use of the 7-segment module and 8x8 matrix on the BetiBoard. This project is basically a game that in the first part, the user enters the values for starting the game via switches and is displayed in the 7-segment module and LEDs. In the second part, the user tries to turn off the lights in the matrix via using buttons. Each button is assigned to specific locations in the matrix by the mod 4 of the last four digits in my student ID (which is 2686 -- %4 --> 2202). Each time the button is pressed, the user's score is increased and displayed in the 7-segment module, and when the game is finished, the user's score will be blinking.

The rule of the game as follows:

0 1 0 1 0 0 0 0 0 0 1 1 1 1 0 0

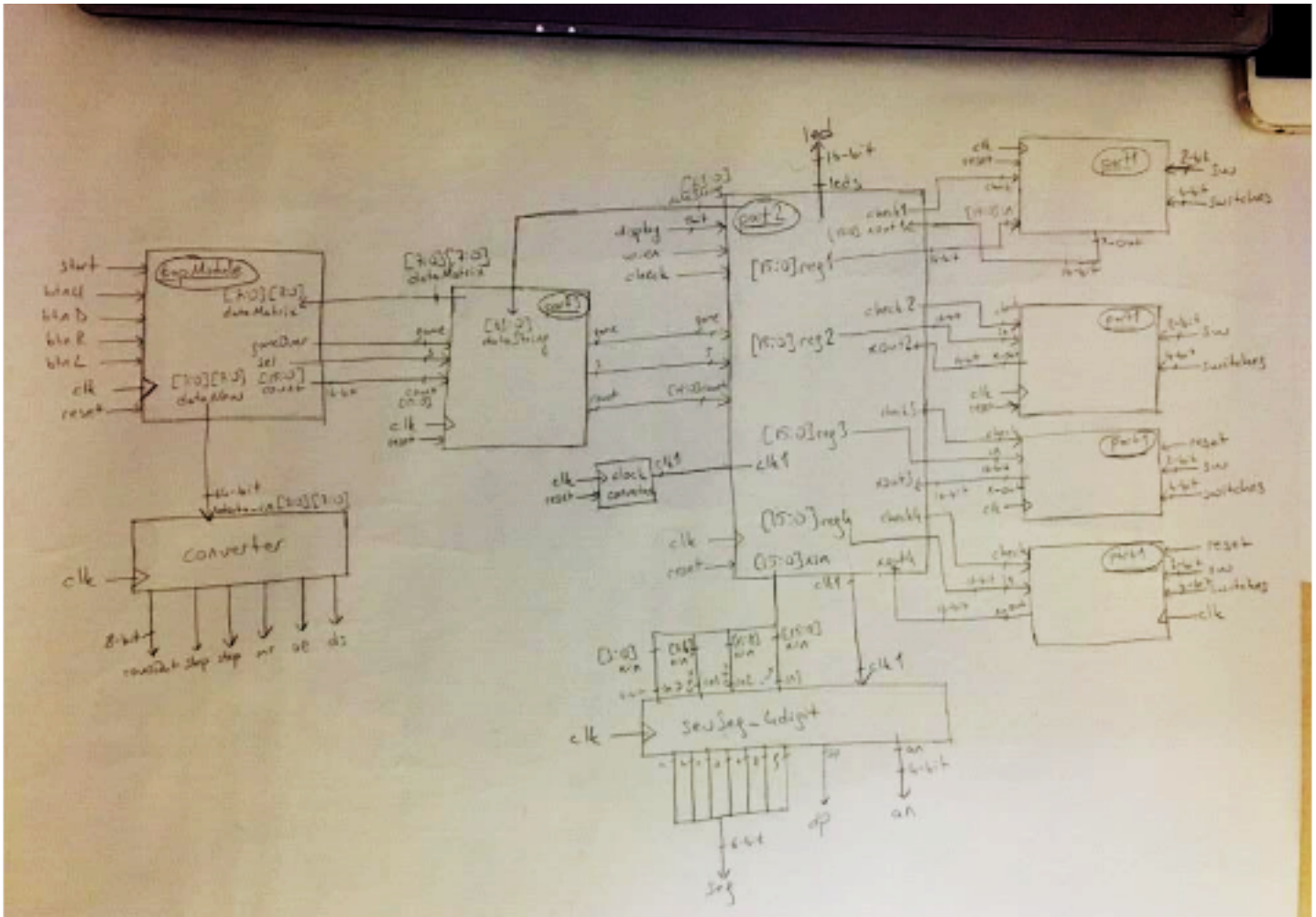
Which indicates when,

North, East & West equals to 1 and South equals 0, or
North & West equals to 1 and East & South equals to 0, or
South & West equals to 1 and North & East equals to 0, or
South & East equals to 1 and North & West equals to 0, or
West equals to 1 and North, South & East equals to 0, or
East equals to 1 and North, South & West equals to 0r

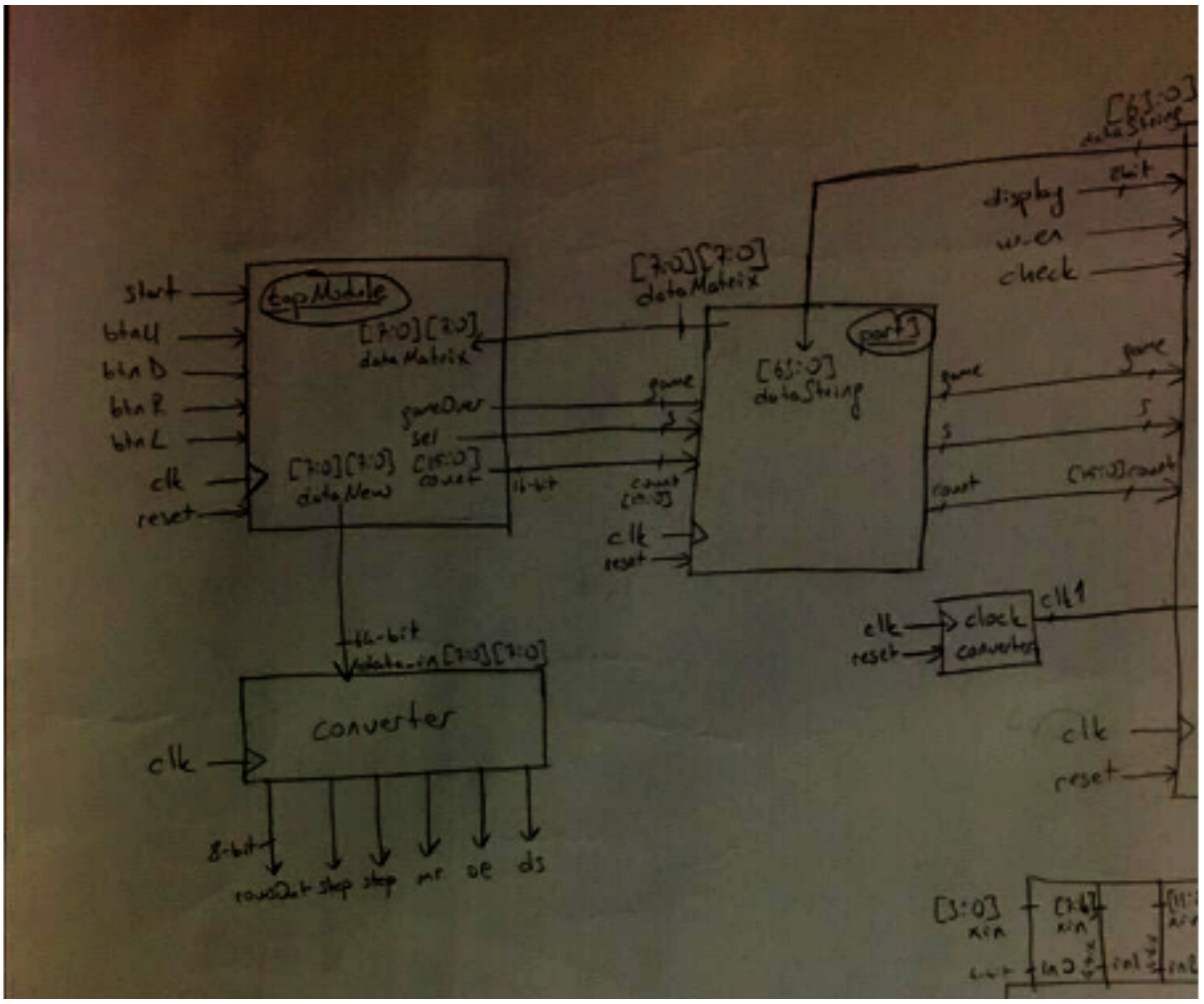
lightens the center cell.

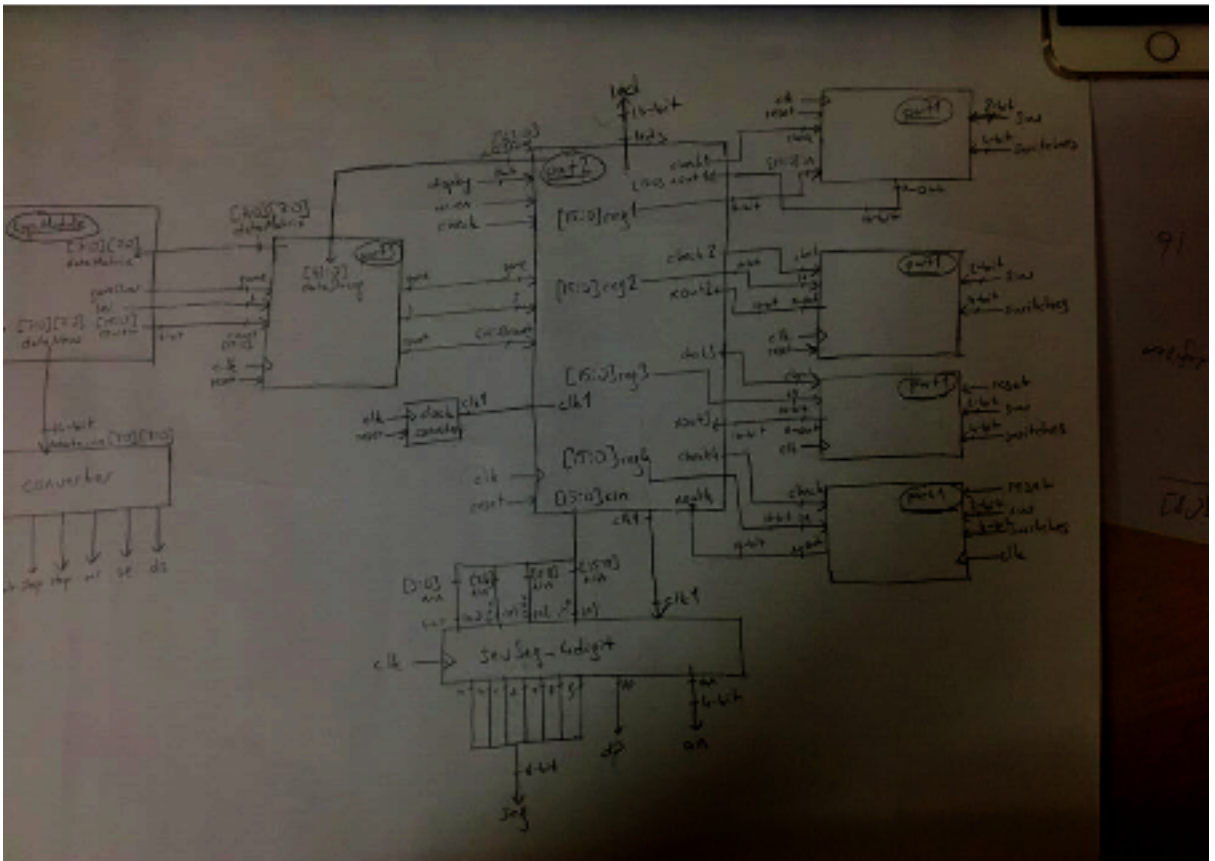
Block Diagram

General Schematic

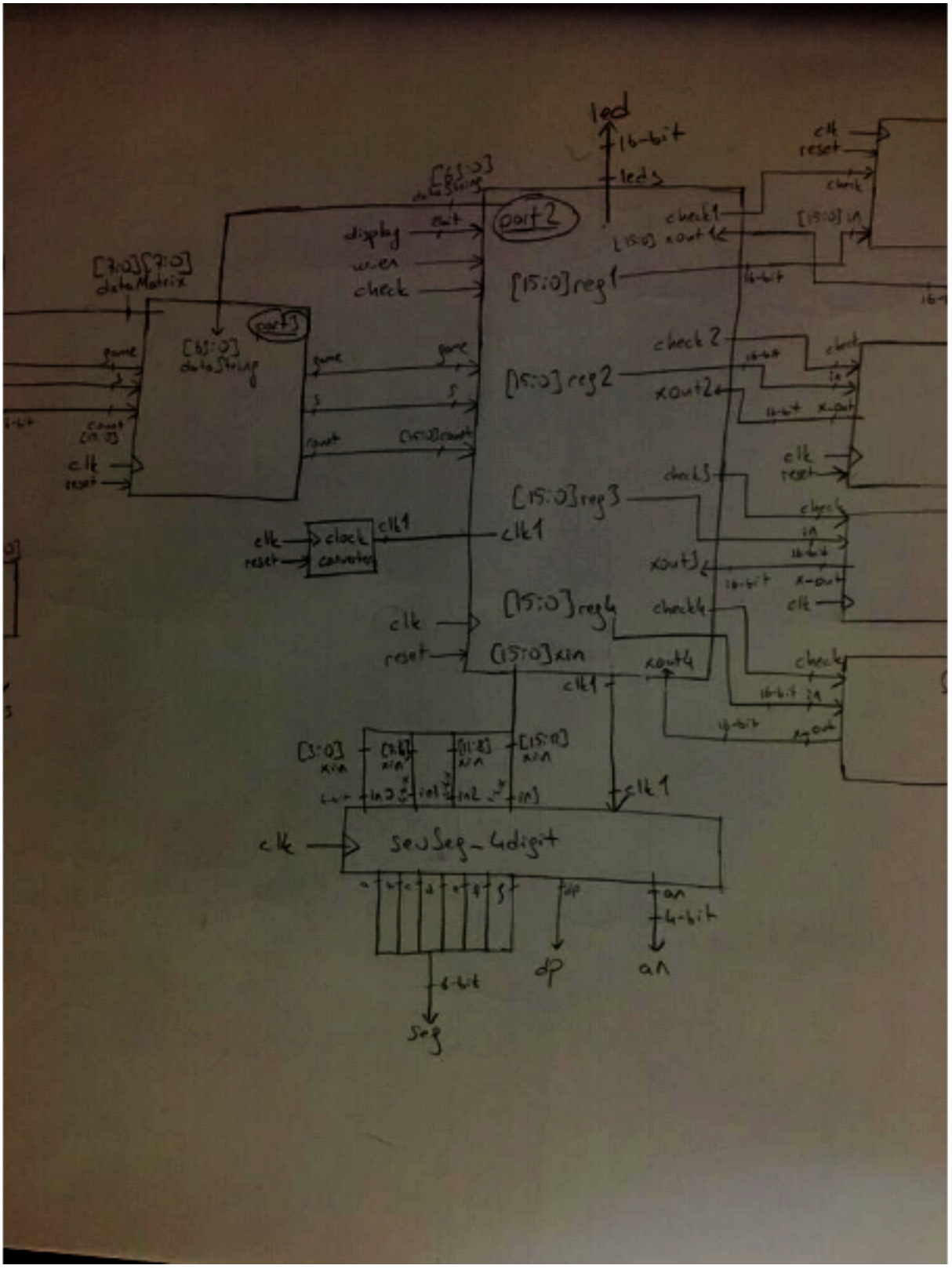


Close-up schematic

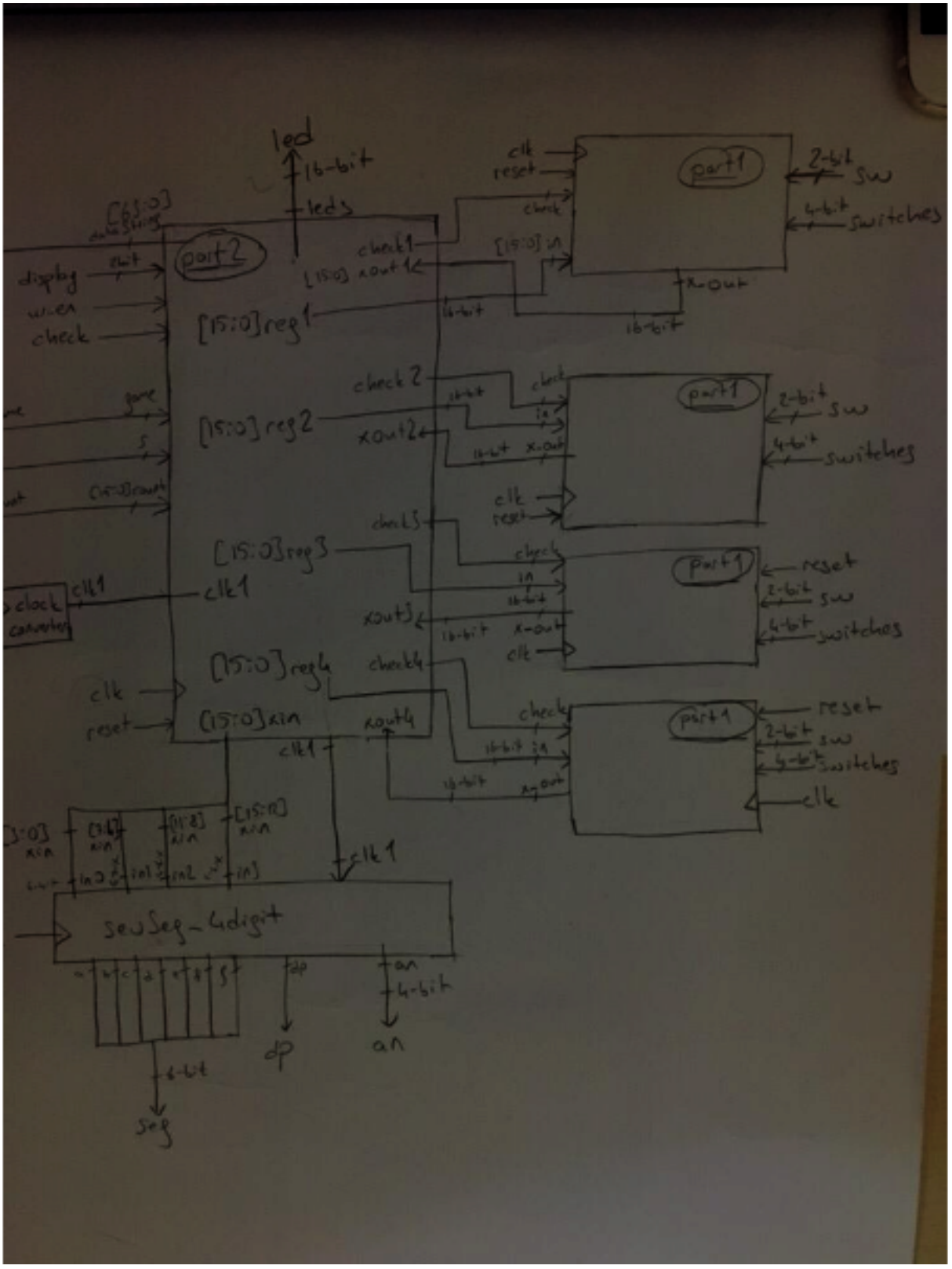




part3 and part2



part2 and part1



Detailed Explanation

This section will be examed according to these topics:

1) Modules' job
2) Input, output & logic variables of the modules and their job
3) Which modules changes which modules' logic variables

1- Module's Job

a) topModule

topModule module is the main module of the game. After the game starts, this module checks the buttons and execute the rule for assigned locations.

topModule calls part3 & converter modules.

b) part3

part3 module changes 64-bit array into an 8x8 matrix.

part3 calls part2 module.

c) converter

converter module displays the 8x8 matrix on the BetiBoard.

d) part2

part2 module creates the 64-bit array for part3.

part2 calls seSeg_4digit, clockConverter & part1 -four times- modules.

e) sevSeg_4digit

sevSeg_4digit module displays the current value on the display screen. This module displays the current score if the game is started; displays the current value of the register, otherwise. Additionally, if the game is finished it starts blinking.

f) clockConverter

clockConverter module creates 0.25 second periods, by changing the clk1.

g) part1

part1 module gets the user's inputs.

2- Input, output & logic variables of the modules and their job

a) topModule

i) Inputs

clk, reset, display[1:0], sw[1:0], check, btnU, btnL, btnR, btnD, w_en, start, switches[3:0]

- **display[1:0], sw[1:0], check, w_en, switches[3:0]** send them to the part3 module.
- **start** is the game started or not
- **btnU, btnL, btnR, btnD** whether the user press the button or not.

ii) Outputs

led, dp, seg[6:0], an[3:0], rowsOut[7:0], shcp, stcp, mr, oe, ds

- Send all of them to the part3 module.

iii) Logic

logic countState; // whether to increase count or not
logic[7:0][7:0] dataMatrix; // original matrix
logic[7:0][7:0] dataNew; // current matrix
logic gameStart, gameOver;// does the game currently been played or not

- **gameStart** is used for displaying count and enabling buttons.

logic[15:0] count; // count the number of times button is pressed
logic sel, u, d, l, r; // whether already entered or not

- sel => is game started?
- u => is last pressed button up?
- d => is last pressed button down?
- l => is last pressed button left?
- r => is last pressed button right?
- u, d, l, r prevents more than one button pressed in sequential order.

int up[15:0]; holds the btnU's responsible locations
int down[15:0]; holds the btnD's responsible locations
int right[15:0]; holds the btnR's responsible locations
int left[15:0]; holds the btnL's responsible locations

b) part3

i) Inputs

game, s, count[15:0], clk,i reset, display[1:0], sw[1:0], check, w_en, switches[3:0]

- **game, s, count, clk, reset, display, sw, check, w_en, switches, leds, dp, seg, an** send them to part2.
- If s is 1, assign dataMatrix by using dataString[63:0]

ii) Outputs

leds, dp, seg[6:0], an[3:0], dataMatrix[7:0][7:0]

- **dp, seg[6:0], an[3:0] & dataString** send to the part2
- **leds** is displaying according to the output of part2

iii) Logic

dataString[63:0]

- **dataString[63:0]** fills with the user's inputs in part2 module and assigned it to the dataMatrix.
- **dataString[63:0]** enters the part2, to update it's value.

c) converter

This module is given for the project.

d) part2

i) Inputs

game, s, count[15:0], clk, reset, display[1:0], sw[1:0],
check, w_en, switches[3:0]

- **display[1:0]** displays the desired register value in the seven Segment.
- **w_en** decides whether save the value into the register or not
- **game, count[15:0]** enters to the sevSeg, thus showing the value
- **display[1:0], sw[1:0], check, switches[3:0]** enters to the part1 module.

ii) Outputs

leds[15:0], dp, seg[6:0], an[3:0], dataString[63:0]

- **leds[15:0]** shows the binary form of the current value displayed in the sevSeg.
- **dataString[63:0]** is updated by reg1, reg2, reg3 and reg4.
- **seg[6:0], an[3:0]** enters the sevSeg_4digit module.

iii) Logic

reg1[15:0], reg2[15:0], reg3[15:0], reg4[15:0], xin[15:0], xout1[15:0], xout2[15:0], xout3[15:0], xout4[15:0], check1, check2, check3, check4, clk1

- **clk1** is a period of 0.25 second which is converted in clockConverter.
- **reg1[15:0], reg2[15:0], reg3[15:0], reg4[15:0]** are the registers which are holding the values that the user enters.
- **xout1[15:0], xout2[15:0], xout3[15:0], xout4[15:0]** are the returns from part1. In other words, these are the current user input. If w_en is on, specified reg will be updated.
- **check1, check2, check3, check4** checks are determined by the display's current value. These are the part1's check conditions.
- **xin[15:0]** is the current value that should shown in the seven segmentation.

e) clockConverter

i) Input

clk, reset

ii) Output

clk1

- New clk which changes every 0.25 seconds.

iii) Logic

No logic, one integer count to divide time into 0.25 second periods.

f) sevSeg_4digit

This module is provided from the lab04 document. Only difference is

Input

game, clk1

- If **game** is 1, the display screen starts blinking with clk1's posedge.

g) part1

i) Input

clk, input reset, sw[1:0], check, in[15:0], switches[3:0]

- If check is not enabled, x_out will be equal to in
- **switches[3:0]** is the value that the user enters.
- **sw[1:0]** is the position that could be changed.

ii) Output

x_out[15:0]

iii) Logic

No logic variable

3- Which module changes which elements

- part1 changes the logic in:
    - part2: xout1, xout2, xout3 & xout4

- clockConverter changes the logic in:
    - part2: clk1

- part2 changes the logic in:
    - part3: [63:0]dataString

- part3 changes the logic in:
    - topModule: [7:0][7:0]dataMatrix

Conclusion

The aim of this project is to demonstrate a game that takes the user's input and displays it on the 8x8 matrix. When the user presses the start button, seven-segment changes and displays the count of button presses.

Considering the schema of the design, topModule elaborates the button presses and increment the counts. With each button presses, 8x8 matrix is updated via dataNew and with each updates, converter fills it again. Hence, sevSeg is connected to project from part2, topModule sends count to part3, and part3 sends it to part2, similarly, gameOver and sel signals went through part3 as well. Likewise, during the coding, there were some difficulties. Firstly, this hierarchy makes it hard to follow. Each module takes too much input and outputs and when unintentionally change something in the code, other modules affected too and it's hard to track it. Secondly, part2 unnecessarily calls part1 for four times, while it can be done in one call.

Overall, the game works perfectly even though it is not the best solution.