D2 Preparation 15.01.2020

ROM

inputs: clk, reset, MDR_bus, load_MDR, load_MAR, CS, R_W
inout: sysbus

The program is in read only memory (ROM).

For load_MAR control signal it sto
It store data in ~~mar and mdr~~ variables.
in look up tables in FPGA.

mar	mdr	At address 0 it the
00000	store, 10000	program ~~store~~ back
00001	load, 10000	to 3 address and
00010	add, 00101	it saves STORE, 5'd16
00011	store, 10000	in mdr variable.
00100	bne, 00110	
00101	~~load, 00010 2;~~	
00110	~~load, 00001 1;~~	

The memory send data to sysbus only if the mar[4] is '1' or '0'. For mar[4] = 1 RAM writes data to bus, otherwise ROM does it. ROM use only 7 addresses, and RAM use 16. which are one half of bus value.

5 bit address: First digit: '1' - RAM writes to bus
'0' ROM writes to bus.
Four digits: address of memory.

Signed by Originator Print Name Date

Signed by Witness Print Name Date

The high impedence 'z' state of sysbus is shown as "7f" state on display. e.g display is turned off.

No, it is not possible to detect 'z' or 'x' states.

32 16 8 4 2 1
1 1 1 1 1

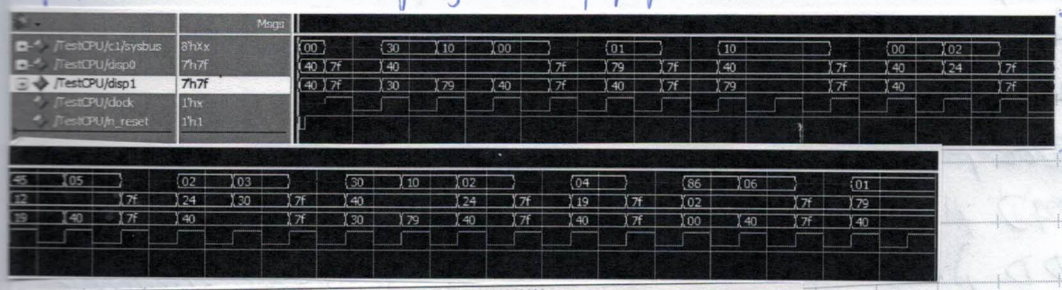
31 - address reserved for switches / segment display
30

Memory Components

0-15 - ROM
16-29 - RAM
30-31 - switch / segment reg

31 &11111
30 &11110

The program in ROM is storing value in ~~0000~~ RAM 0000 adress memory. Then ~~it~~ is the value from ROM 00101 (2) is added and stored again in RAM 0000 memory address. If ACC is 0 the program jumps to 00110 ROM memory



2.1 Simulation

SSR.SV

```
//////////////////////////////////////////////////////////////////////
// Design unit: SSR
//            :
// File name  : ssr.sv
//            :
// Description: Switch/Segment Register

module SSR #(parameter WORD_W = 8, OP_W = 3)
            (input logic clock, n_reset, MDR_bus, load_MDR, load_MAR, CS, R_NW,
             inout wire [WORD_W-1:0] sysbus, input logic [7:0] switches, output logic [3:0]hex1,hex2);

 //`include "opcodes.h"

 logic [WORD_W-1:0] mdr;
 logic [WORD_W-OP_W-1:0] mar;
 logic [WORD_W-1:0] mem [0:(1<<(WORD_W-OP_W-1))-1]; //top half of address range
 //logic [WORD_W-1:0] mem [0:11]; //top half of address range
 int i;

 assign sysbus = (MDR_bus & mar == 4'd30) ? switches : {WORD_W{1'bZ}};// Switches data out to bus

 always_ff @(posedge clock, negedge n_reset)
  begin
  if (~n_reset)
        begin
        mdr <= 0;
        mar <= 0;
        hex1 <= 0;
        hex2 <= 0;
        end
  else
        if (load_MAR)
        mar <= sysbus[WORD_W-OP_W-1:0];
        else if (load_MDR)
              mdr <= sysbus;
              else if (CS & mar == 5'd31) // if 7 segment display address
                    begin
                    if (!R_NW) // Writing, so this signal needs to be low
                    begin
                    hex1 <= mdr[7:4];
                    hex2 <= mdr[3:0];
                    end
                    else mdr <= {hex1,hex2};
                    end
  end


endmodule
```

Signed by Witness ..... Print Name ..... Date

Name ..... Date

```
34
35  assign sysbus = (MDR_bus & mar[WORD_W-OP_W-1] & ~(mar == 5'd31)) ? mdr : {WORD_W{1'bZ}};
36
37  always_ff @(posedge clock, negedge n_reset)
38    begin
39    if (~n_reset)
40      begin
41      mdr <= 0;
42      mar <= 0;
43      end
44    else
45      if (load_MAR)
46        mar <= sysbus[WORD_W-OP_W-1:0];
47      else if (load_MDR)
48        mdr <= sysbus;
49      else if (CS & mar[WORD_W-OP_W-1] & ~(mar == 5'd31))
50        if (R_NW)
51          mdr <= mem[mar[WORD_W-OP_W-2:0]];
52        else
53          mem[mar[WORD_W-OP_W-2:0]] <= mdr;
54    end
```

ROM.sv ←

opcodes

000 load
001 store
010 add
011 sub
100 bne

(101 xor new!)

Program which reads 2 numbers from the switches and displays the sum on the 7 segment display:

```
LOAD,  5'd30   ← get 1st number
STORE, 5'd16   ← store in RAM
LOAD,  5'd30   ← get 2nd number
ADD,   5'd16   ← add to 1st number
STORE, 5'd31   ← display on hex
```

Program which use new xor function in ALU:

xor in ALU.sv:

```
... else if (ALU_xor)
      ecc <= ecc ^ sysbus;
```

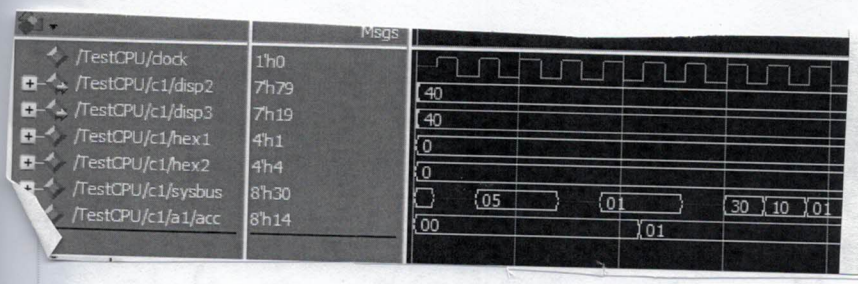
in sequencer.sv:

```
58: if(ALU_xor op == 'XOR)
      ALU_xor = 1'b1;
```

```
47  always_comb
48    begin
49    mdr = 0;
50    case (mar)
51
52      0: mdr = {`LOAD, 5'd5};
53          1: mdr = {`STORE, 5'd16};
54          2: mdr = {`LOAD, 5'd6};
55          3: mdr = {`XOR, 5'd16};
56      4: mdr = {`STORE, 5'd31};
57          5: mdr = 5'b00001;
58          6: mdr = 5'b10101;
59      default: mdr = 0;
60    endcase
61    end
62
63  endmodule
```

Signed by Originator ..... Print Name ..... Date

Signed by Witness ..... Print Name ..... Date



The simulation of xor encrypting program.
The value h.01 (bin 0001) is xor by code 'h15 (10101) and the score 'h14(10100) is displayed on hex display.

Lab Code:	None	Poor	OK	Good	Exceptional	Outstanding
Preparation	0	1	2	3	4	(5)
Progress	0	1	2	3	4	5
Understanding	0	1	2	3	4	(5)
Logbook Use	0	1	2	3	4	5
Demonstrators only						
Print Full Name	Ike Zhan				Signed	Zhan

Lab work 02

16/01/20

3.1
The behaviour of 2.1 program is quite weird. ~~The~~ generally display follows previous simulation but in 'Z state there are wrong numbers displayed, instead of turning off screen on 'Z state there are wrong, random numbers.
~~But~~ During this state, FPGA use this register to store data in sysbus variable which is global variable (type wire).

Simulation	00	Z	30	10	00	Z	01	Z	10	10	Z	00	02	4Z
Behaviour	00	20	30	10	00	00	01	00	10	10	00	00	02	40

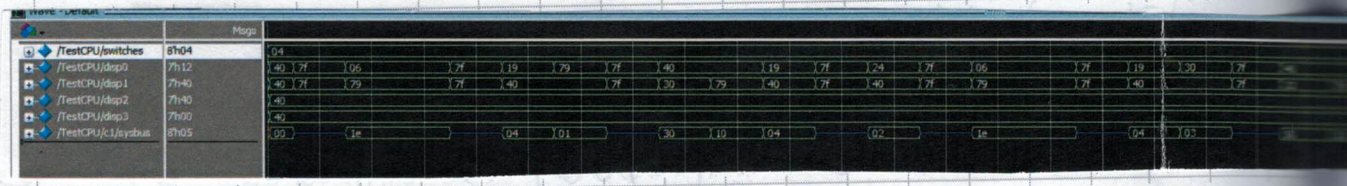
S	45	05	Z	02	03	Z	30	10	02	Z
B	45	05	00	02	03	20	30	10	02	00

Signed by Originator ..... Print Name ..... Date .....

Signed by Witness ..... Print Name ..... Date .....

3.2

For switches = 4 velue i've got simulation:



I've got similar results as in simulation, although for '2 state the display shows '00' and for some states HEX1 display '8 digit instead of '0 digit.

switches velue: 44

Simulation	00	Z	1e	1e	Z	04	01	Z	30	10	04	Z	02	Z	1e	1e	Z
Behaviour	00	00	1e	1e	00	84	01	00	30	10	84	00	02	00	1e	1e	00

The problem with such behaviour was quite simple. Reset button should be pressed before starting a program. In otherhand the outcome of program is random. The correct behaviour display '2 state as 00. The '2 state cannot be detected so it is interpreted as '0 state.

Signed by Originator ..... Print Name ..... Date

Signed by Witness ..... Print Name ..... Date

3.3

The behaviour of sysbus and hex displays is consistent with simulation. 'Z state is displayed as '0' due to impossible measurement of 'Z' state. ~~Probe~~ Program correctly is getting value and encrypting it

3.4

string - oiytmmvk

```
always_comb
  begin
  mdr = 0;
  case (mar)
        0: mdr = {`LOAD, 5'd10};
        1: mdr = {`STORE, 5'd21}; //Store counter
        2: mdr = {`LOAD, 5'd30}; //Get Char
        3: mdr = {`XOR,5'd11}; //Encrypt/Decrypt
        4: mdr = {`STORE, 5'd31}; //Display in HEX
        5: mdr = {`LOAD, 5'd21};
        6: mdr = {`SUB, 5'd9};//Decrement counter
        7: mdr = {`BNE, 5'd9};//Repeat, if 8bit block - end
        8: mdr = 0;
        9: mdr = 1;
       10: mdr = 7;
       11: mdr = 5'b10101;//key
   default: mdr = 0;
  endcase
  end

endmodule
```

This program is taking 8-bit char from switches input an d encrypting it add display into hex.

11001

11100

0010

10000

1111

0101

1010

10_{10}

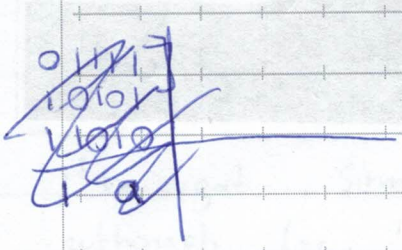
Signed by Originator _____ Print Name _____ Date _____

Signed by Witness _____ Print Name _____ Date _____

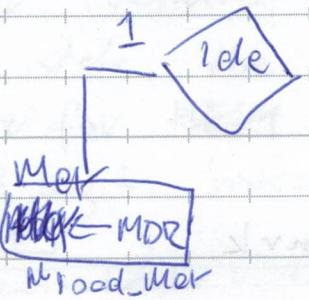
3.3 Testing

switch value	
encrypted value	

key: 10101



New function load data ('LDA)
Load only data of address



10101 10000
1011 10101
10101 11000
11100

11 -> 1 e
12 -> 1 S

01111	01000	11001	
10101	10101	10101	
11010	11101	01100	
16	1 d	D c	

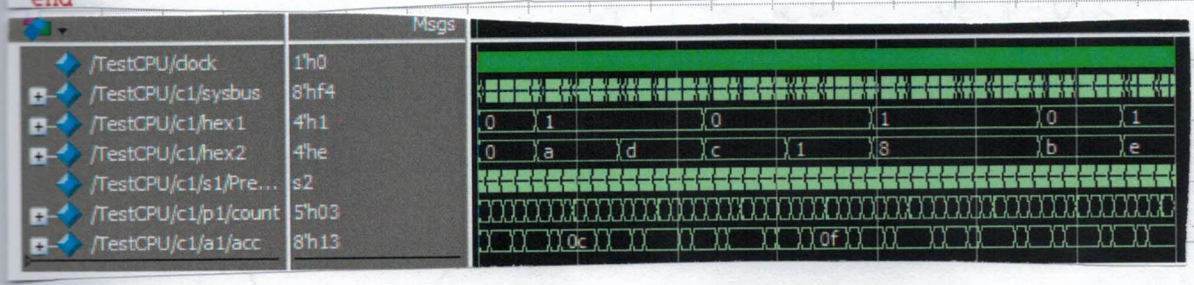
code 10101 encrypt

o	- 01111	0 a
i	- 01000	1 d
y	- 11001	D c
t	- 10100	0 1
m	- 01101	d 8
m	- 01101	1 8
v	- 11110	0 b
k	- 01011	1 C

Signed by Originator _____ Print Name _____ Date _____

Signed by Witness _____ Print Name _____ Date _____

```
always_comb
  begin
  mdr = 0;
  case (mar)
        //load pointer (5'd6) (0)
        0: mdr = {`LOAD, 5'd9};
        1: mdr = {`STORE, 5'd20};
        //save pointer in ram (5'd20) <-- bne function loop (1)
        2: mdr = {`LDE, 5'd20};
        //load from (5'd20) (2)
        3: mdr = {`XOR, 5'd10};
        //xor from (5'd7) (3)
        4: mdr = {`STORE, 5'd31};
        //store in hex (4)
        //4:
        //increment pointer (load, add 1, (can use INC op code)) (5) (6)
        5: mdr = {`LOAD, 5'd20};
        6: mdr = {`INC, 5'd0};
        //mdr = {`STORE, 5'd20};
        //bne 1 (7)
        7: mdr = {`BNE, 5'd8};
        8: mdr = 1;
        9: mdr = 5'd11; //Start data pointer
        10: mdr = 5'b10101;//code
        11: mdr = 5'b01111;//data 1-8 bits to encode
        12: mdr = 5'b01000;
        13:mdr = 5'b11001;
        14:mdr = 5'b10100;
        15:mdr = 5'b01101;
        16:mdr = 5'b01101;
        17:mdr = 5'b11110;
        18:mdr = 5'b01011;
    default: mdr = 0;
  endcase
  end
```



Decrypting the ~~cache~~ 8-bit data by xor gate cipher.

Signed by Originator ..... Print Name ..... Date .....

Signed by Witness ..... Print Name ..... Date .....