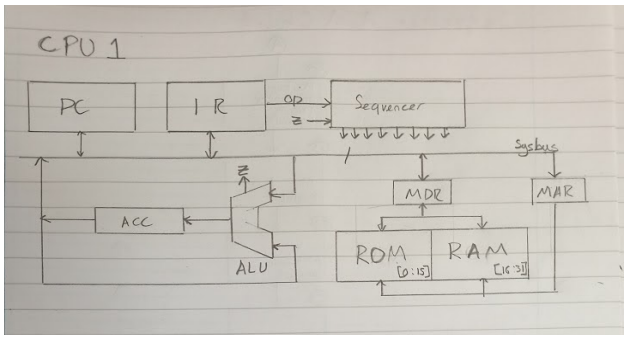
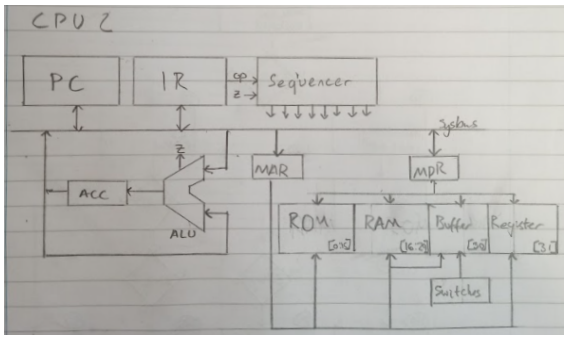
D2: Digital Systems and Microprocessor Design Exercise
https://stackoverflow.com/questions/16424726/what-is-the-difference-between-verilog-and

Understanding the Microprocessor - 23/01/21

Architecture:



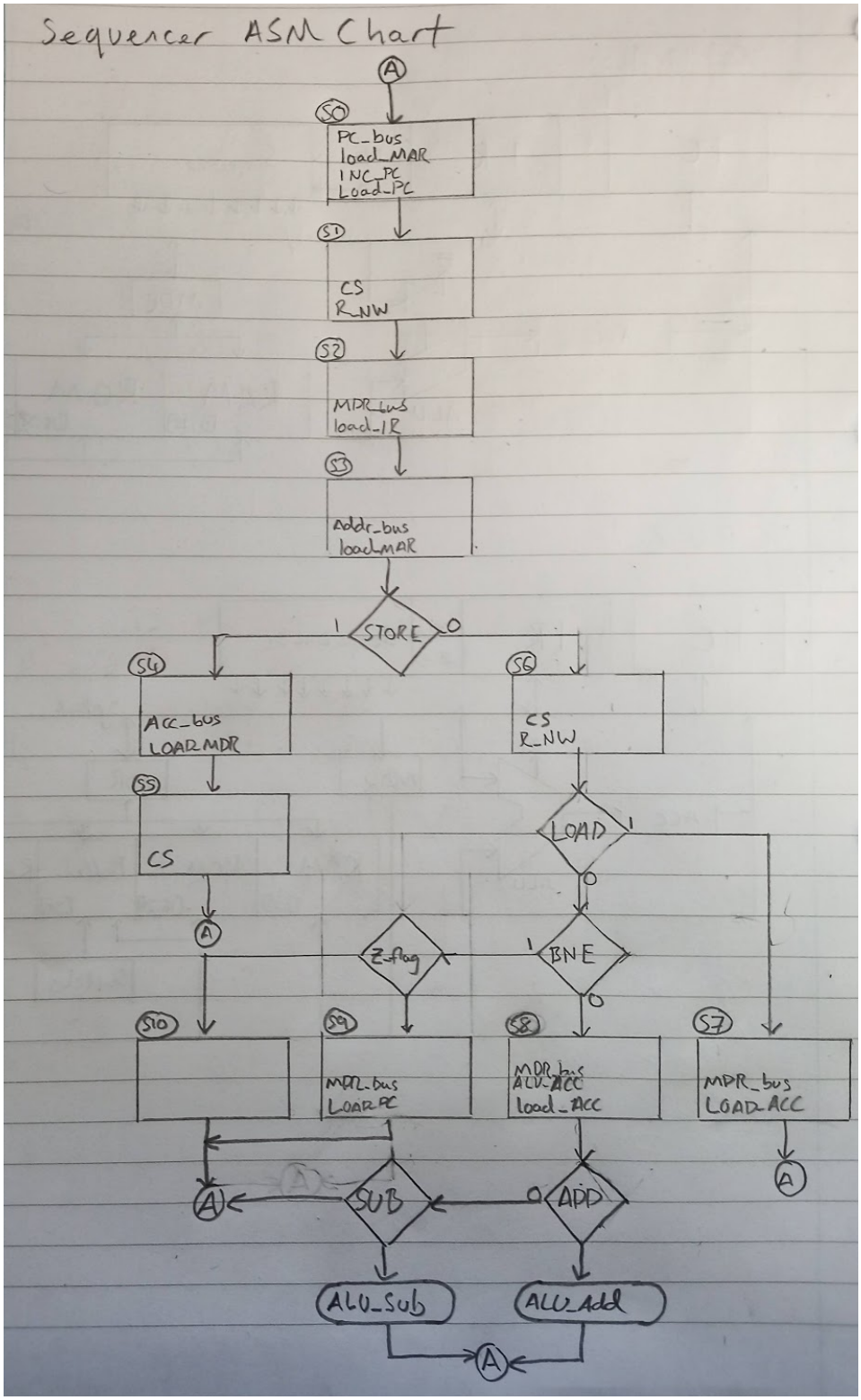


PC:
The program counter is a sequential logic module that records the position of the program through the ROM and outputs the position to the sysbus when the next instruction needs to be loaded.

IR:
The instruction register is a sequential logic module that reads the opcode from the sysbus and passes it to the sequencer.

Sequencer:
The sequencer is a state machine that controls the order of operation for the other modules. This is important to stop any bus contention occurring.

ACC_bus	Drive bus with contents of ACC (enable three-state output)
load_ACC	Load ACC from bus
PC_bus	Drive bus with contents of PC
load_IR	Load IR from bus
load_MAR	Load MAR from bus
MDR_bus	Drive bus with contents of MDR
load_MDR	Load MDR from bus
ALU_ACC	Load ACC with result from ALU
INC_PC	Increment PC and save the result in PC
Addr_bus	Drive bus with operand part of instruction held in IR
CS	Chip select. Use contents of MAR to set up memory address
R_NW	Read, not write. When false, contents of MDR are stored in memory
ALU_add	Perform an add operation in the ALU
ALU_sub	Perform a subtract operation in the ALU


Sequencer ASM Chart
A
S0
PC_bus
load_MAR
INC_PC
Load_PC
S1
CS
R_NW
S2
MDR_bus
load_IR
S3
Addr_bus
load_MAR
STORE
1
0
S4
ACC_bus
LOAD_MDR
S5
CS
R_NW
S6
CS
LOAD
1
0
A
Z_flag
BNE
1
0
S10
S9
MDR_bus
LOAD_PC
S8
MDR_bus
ALU_ACC
load_ACC
S7
MDR_bus
LOAD_ACC
A
A
SUB
0
ADD
ALU_Sub
ALU_Add
A

Basic Operation - 23/01/21

1) The file "rom.sv" contains a program. What does this program do? Where does it store data?

The program in the ROM is stored in the memory addresses 0 to 6. This program uses memory address 16 to store its working, this is the first address in RAM.

2) What happens at address 0 and why? Where does the program loop back to?

At address 0, in the ROM, the program stores the current value in the accumulator (inside the ALU) to address 16, in the RAM.

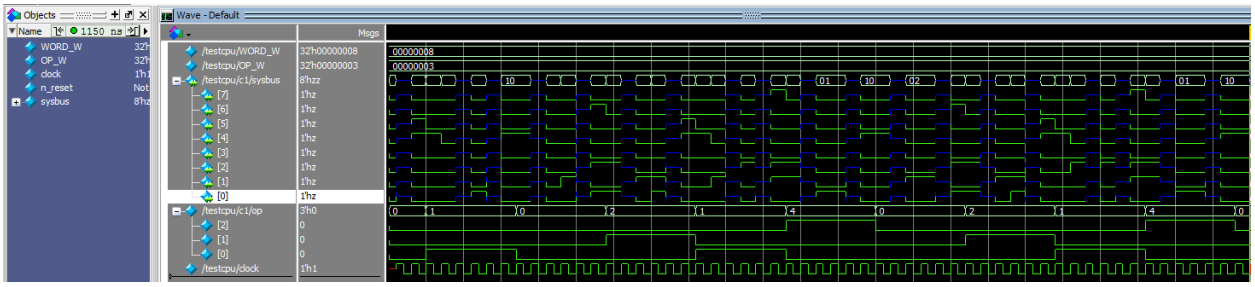
This program will loop back to address one because the BNE opcode sets the mdr to address one and the sequencer to load the program counter from the mdr.

3) The memory address registers in the ram and rom have 5 bits. How are the ram and rom organised to ensure that they do not overlap? How are the addresses decoded?

ROM covers the memory addresses 0 to 15.
RAM covers the memory addresses 16 to 31 (or 16 to 29 for CPU 2).

The three most significant bits of the sysbus is the opcode and so is ignored when looking for the address. The five least significant bits of the sysbus is the mar, which is a five bit address stored in binary.The ROM can only access the sysbus when the value of the most significant bit of the mar is 0.The RAM can only access the sysbus when the value of the most significant bit of the mar is 1. This stops bus contention.

4) How is the high impedance, 'Z', state of sysbus shown in the simulation?



In ModelSim the high impedance state is displayed as a blue line between the two logic levels.

5) SystemVerilog and ModelSim can represent ‘Z’ and ‘X’ states, but are these states “real”? In other words, is it possible to design logic that would detect ‘Z’ or ‘X’ states and display them?

‘Z’ and ‘X’ states are not “real” states as they cannot be verified with only voltage readings, which is all a digital system is capable of doing. However, it is possible to create logic that would identify ‘Z’ or ‘X’ states within itself as it would know at any point in its function when nothing is writing to the bus or if there is bus contention.

Memory Mapped Input and Output - 23/01/21

1) Sketch a diagram of the memory, going from address 0 to address 31, and showing which components are at each address.

Address	Function
[0:15]	ROM
[16:29]	RAM
[30]	Buffer
[31]	Register

2) Add the appropriate lines to cpu2 to instantiate one buffer and one register.

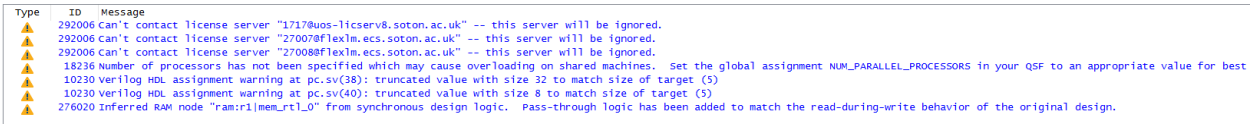
```
//instantiate 1 register and 1 buffer here

buffer #(.WORD_W(WORD_W), .OP_W(OP_W)) b1  (.*);
sequencer #(.WORD_W(WORD_W), .OP_W(OP_W)) s1  (.*);
ir #(.WORD_W(WORD_W), .OP_W(OP_W)) i1  (.*);
pc #(.WORD_W(WORD_W), .OP_W(OP_W)) p1 (.*);
alu #(.WORD_W(WORD_W), .OP_W(OP_W)) a1 (.*);
ram #(.WORD_W(WORD_W), .OP_W(OP_W)) r1 (.*);
rom #(.WORD_W(WORD_W), .OP_W(OP_W)) r2 (.*);
register #(.WORD_W(WORD_W), .OP_W(OP_W)) r3 (.*);
```

3) If the CPU attempts to read from address 30, both the buffer and the RAM will be enabled. Why is this going to be a problem?

This is a problem as both RAM and the buffer will assert their values onto the sysbus as they are both being read. The two values on the same bus will interfere with each other giving a signal that is likely neither of them. This is bus contention.

4) Modify this line so that mdr is not written to the bus when mar contains addresses 30 or 31. The RAM should continue to work correctly for other addresses.

```
//The following line in the ram module copies the contents of mdr to sysbus:
assign sysbus = (MDR_bus & mar[WORD_W-OP_W-1]
                  & ~(mar==30 | mar==31)) ? mdr : 'z;
```



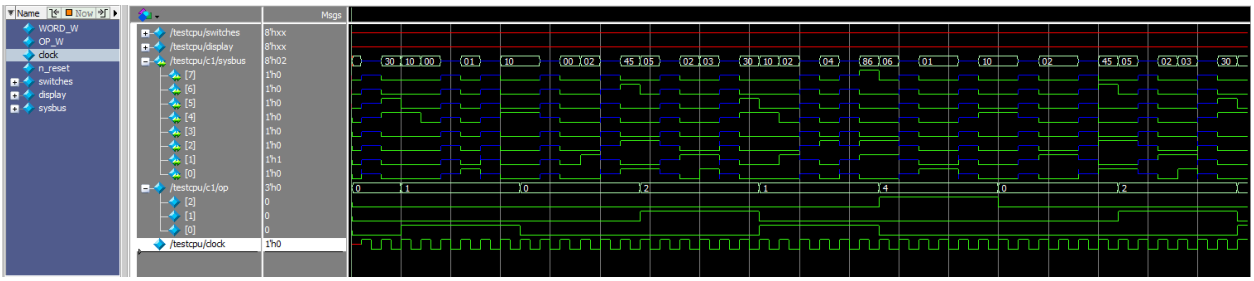
Mark Zwolinski
Sun 24/01/2021 16:11
To: butterworth j.d. (jdb1g20)

```
Logfile for user jdb1g20 attempt 5; ram.sv
Starting ModelSim
Model Technology ModelSim - Intel FPGA Edition vlog 10.5b Compiler 2016.10 Oct  5 2016
Start time: 16:11:40 on Jan 24,2021
vlog -work jdb1g20work -sv ram.sv
-- Compiling module ram

Top level modules:
       ram
End time: 16:11:40 on Jan 24,2021, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
# vsim -L jdb1g20work -c -novopt -do "../do_vsim" ../work.testram
# Start time: 16:11:41 on Jan 24,2021
# Loading sv_std.std
# Loading ../work.testram
# Loading jdb1g20work.ram
# do ../do_vsim
# ---------------------------
# Modelsim Simulation
# ---------------------------
#
# RAM address        16 OK
# RAM address        17 OK
# RAM address        18 OK
# RAM address        19 OK
# RAM address        20 OK
# RAM address        21 OK
# RAM address        22 OK
# RAM address        23 OK
# RAM address        24 OK
# RAM address        25 OK
# RAM address        26 OK
# RAM address        27 OK
# RAM address        28 OK
# RAM address        29 OK
# RAM address        30 OK
# RAM address        31 OK
#
# ---------------------------
# End time: 16:11:41 on Jan 24,2021, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
```

5) Modify “testcpu” so that it uses cpu2 instead of cpu1. You will also need to include switches and display as variables in the testbench.

```
module testcpu;

parameter int WORD_W = 8, OP_W = 3;

logic  clock, n_reset;
logic [WORD_W-1:0] switches;
logic [WORD_W-1:0] display;
wire [WORD_W-1:0] sysbus;

cpu2 #(.WORD_W(WORD_W), .OP_W(OP_W)) c1 (.*);
```

6) What do you see in the Waveform window for display and switches?

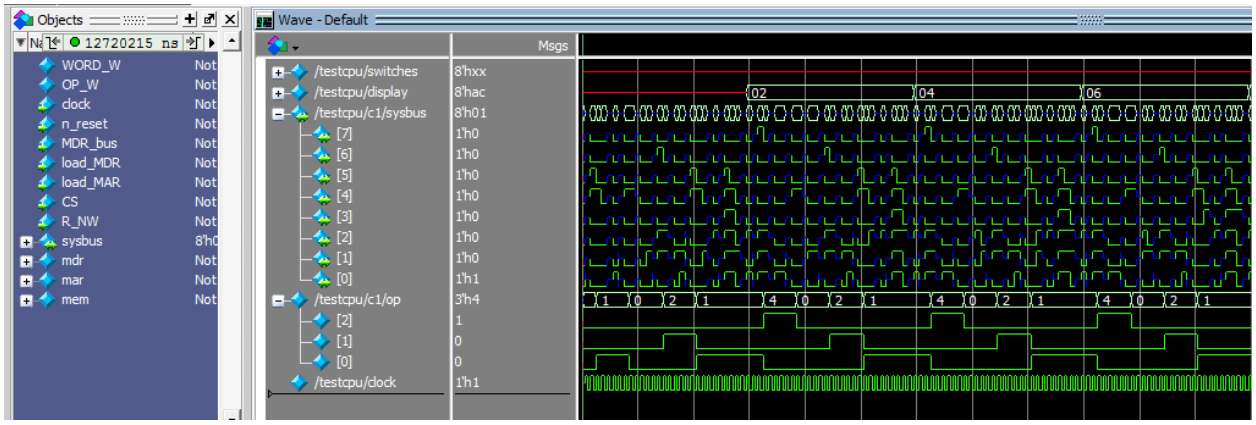


Both the display register and the switches are of unknown value as they are not defined at any point in the testbench or the program.

7) Modify the program in the rom module such that the result in the accumulator is STOREd to address 31 (display) as well as address 16. Note that you will have to move the contents of the ROM at addresses 4 to 6 up by one place and modify the ADD and BNE instructions. What do you now see in the simulation?

```
    0: mdr = {`STORE, 5'd16}; //Store the contents of accumulator
    1: mdr = {`LOAD, 5'd16};  //Load the contents of address into the
accumulator
    2: mdr = {`ADD, 5'd6};    //Add the contents of address to the accumulator
    3: mdr = {`STORE, 5'd16}; //Store the contents of accumulator
    4: mdr = {`STORE, 5'd31}; //Store the contents of accumulator
    5: mdr = {`BNE, 5'd7};    //Branch if result of last arithmetic operation
is not zero
    6: mdr = 2;               //contents used by another instruction
    7: mdr = 1;               //contents used by another instruction
```

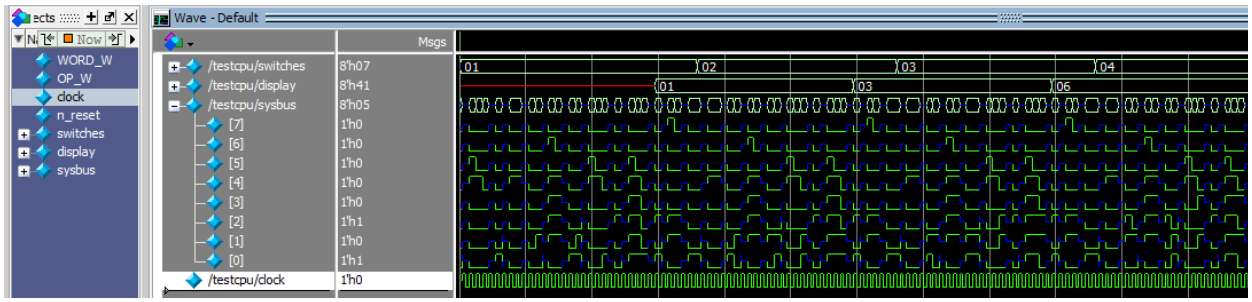
My modified program would output the value to the display register. This clearly shows the program counts up in twos.



8) Make a second change to the program in the rom module such that the data for the ADD operation comes from address 30 – the switches. Modify “testcpu” to apply different values to the switches during the simulation. What do you now see in the simulation?

```
    0: mdr = {`STORE, 5'd16}; //Store the contents of accumulator
    1: mdr = {`LOAD, 5'd16};  //Load the contents of address into the
accumulator
    2: mdr = {`ADD, 5'd30};    //Add the contents of address to the accumulator
    3: mdr = {`STORE, 5'd16}; //Store the contents of accumulator
    4: mdr = {`STORE, 5'd31}; //Store the contents of accumulator
    5: mdr = {`BNE, 5'd6};    //Branch if result of last arithmetic operation
is not zero
    6: mdr = 1;               //contents used by another instruction
```

My new program would take the values from the switches and add it to the current value, outputting it to the display register.



Extending the Microprocessor - 24/01/21

Creating a new version of the CPU, cpu3, I added the bitwise xor and bitwise not (complement) functions to act on the accumulator. This meant defining the opcodes, adding new enable lines between the ALU and sequencer, adding the functionality to the ALU and Sequencer and writing a new program in ROM to use these commands.

ALU:

```
    if (load_ACC)
      if (ALU_ACC)
        begin
        if (ALU_add)
          acc <= acc + sysbus;
        else if (ALU_sub)
          acc <= acc - sysbus;
        else if (ALU_xor)
          acc <= acc ^ sysbus;
        else if (ALU_comp)
          acc <= ~acc;
        end
      else
        acc <= sysbus;
```

Sequencer:

```
 s8: begin
       MDR_bus = 1'b1;
       ALU_ACC = 1'b1;
       load_ACC = 1'b1;
       if (op == `ADD)
         ALU_add = 1'b1;
       else if (op == `SUB)
         ALU_sub = 1'b1;
       else if (op == `XOR)
         ALU_xor = 1'b1;
       else if (op == `COMP)
         ALU_comp = 1'b1;
     Next_State = s0;
       end
```

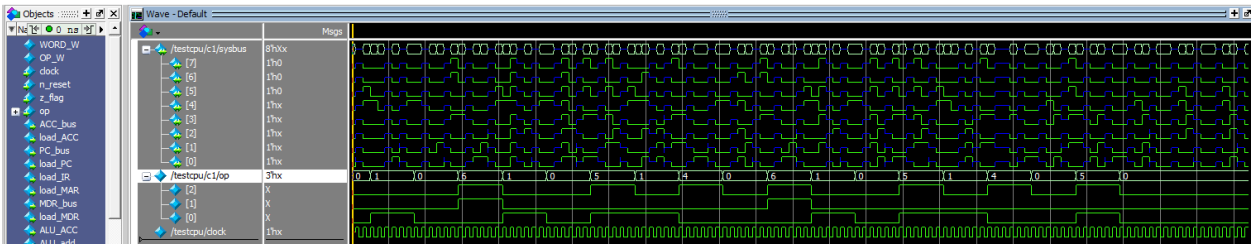
CPU:

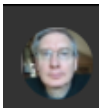
```
module cpu1 #(parameter WORD_W = 8, OP_W = 3)
                  (input logic clock, n_reset,
                   inout wire [WORD_W-1:0] sysbus);

logic ACC_bus, load_ACC, PC_bus, load_PC, load_IR, load_MAR,
MDR_bus, load_MDR, ALU_ACC, ALU_add, ALU_sub, ALU_xor, ALU_comp,
INC_PC, Addr_bus, CS, R_NW, z_flag;
```

ROM:

```
  case (mar)
    0:  mdr = {`STORE,  5'd16}; //Store the contents of accumulator
    1:  mdr = {`LOAD,   5'd16}; //Load the contents of address into the
accumulator
    2:  mdr = {`COMP,   5'd8};  //Complement the contents of the accumulator
    3:  mdr = {`STORE,  5'd16}; //Store the contents of accumulator
    4:  mdr = {`LOAD,   5'd16}; //Load the contents of address into the
accumulator
    5:  mdr = {`XOR,    5'd9};  //Xor the contents of address with the
accumulator
    6:  mdr = {`STORE,  5'd16}; //Store the contents of accumulator
    7:  mdr = {`BNE,    5'd10}; //Branch if result of last arithmetic operation
is not zero
    8:  mdr = 1;                //contents used by another instruction
    9:  mdr = 170;              //contents used by another instruction
    10: mdr = 1;                //contents used by another instruction
    default: mdr = 0;           //rest of ROM is 0
  endcase
```

I then verified that the program was working correctly by simulating it in modelsim and going through each operation. I found that by the end of the first cycle the value in ram was 1'h55, which was to be expected when 1'hFF and 1'hAA are XORed together.





Mark Zwolinski
Sun 24/01/2021 15:02
To: butterworth j.d. (jdb1g20)

```
Logfile for user jdb1g20 attempt 1; alu.sv, sequencer.sv
Starting ModelSim
Model Technology ModelSim - Intel FPGA Edition vlog 10.5b Compiler 2016.10 Oct  5 2016
Start time: 15:02:00 on Jan 24,2021
vlog -work jdb1g20work -sv alu.sv
-- Compiling module alu

Top level modules:
        alu
End time: 15:02:00 on Jan 24,2021, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
Model Technology ModelSim - Intel FPGA Edition vlog 10.5b Compiler 2016.10 Oct  5 2016
Start time: 15:02:00 on Jan 24,2021
vlog -work jdb1g20work -sv sequencer.sv
-- Compiling module sequencer

Top level modules:
        sequencer
End time: 15:02:00 on Jan 24,2021, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
# vsim -L jdb1g20work -c -novopt -do "../do_vsim" ../work.testALU_seq
# Start time: 15:02:01 on Jan 24,2021
# Loading sv_std.std
# Loading ../work.testALU_seq
# Loading jdb1g20work.sequencer
# Loading jdb1g20work.alu
# do ../do_vsim
# ---------------------------
# Modelsim Simulation
# ---------------------------
#
# XOR operation works correctly
#
# ---------------------------
# End time: 15:02:01 on Jan 24,2021, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
================================================================
Starting Verilator
================================================================
Starting Quartus
```