MARMARA UNIVERSITY – FACULTY OF ENGINEERING

LABORATORY ASSIGNMENT #3 REPORT

Student Name, Surname: Hasan Şenyurt – Yusuf Akbulut

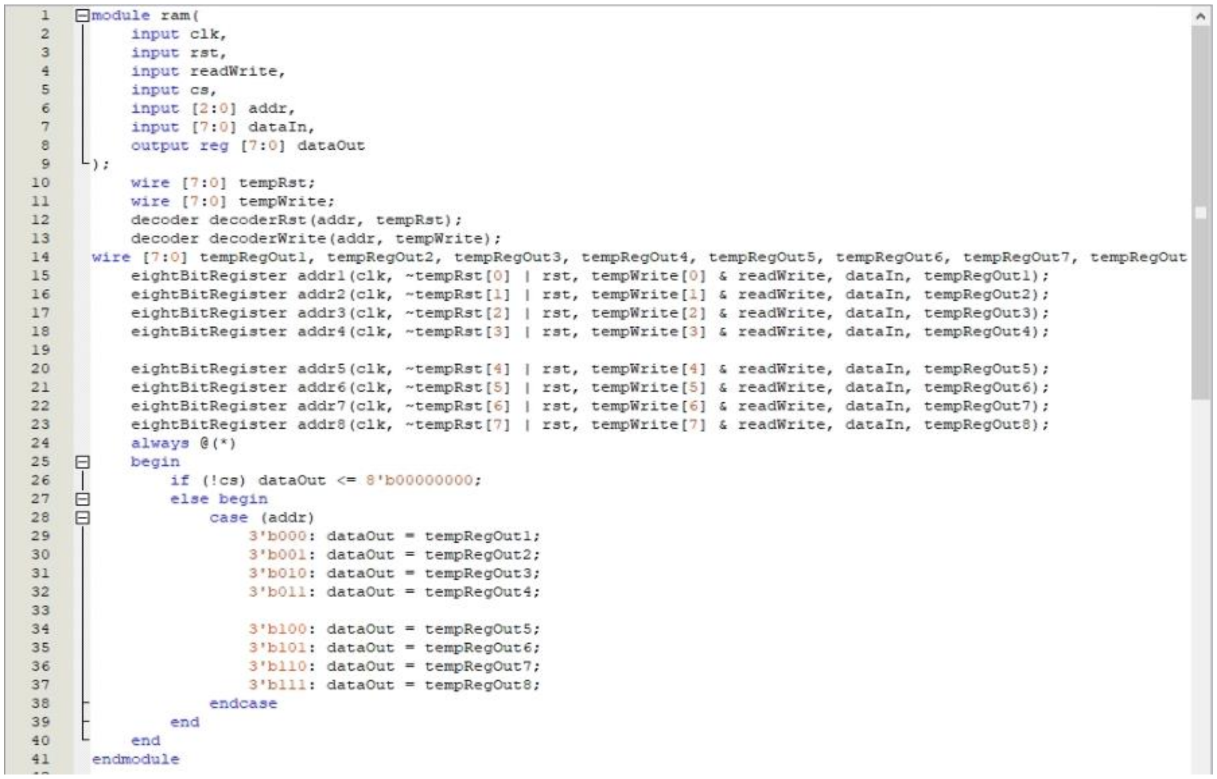
Student ID: 150120531 - 150118023

Department: Computer Engineering

TABLE OF CONTENTS

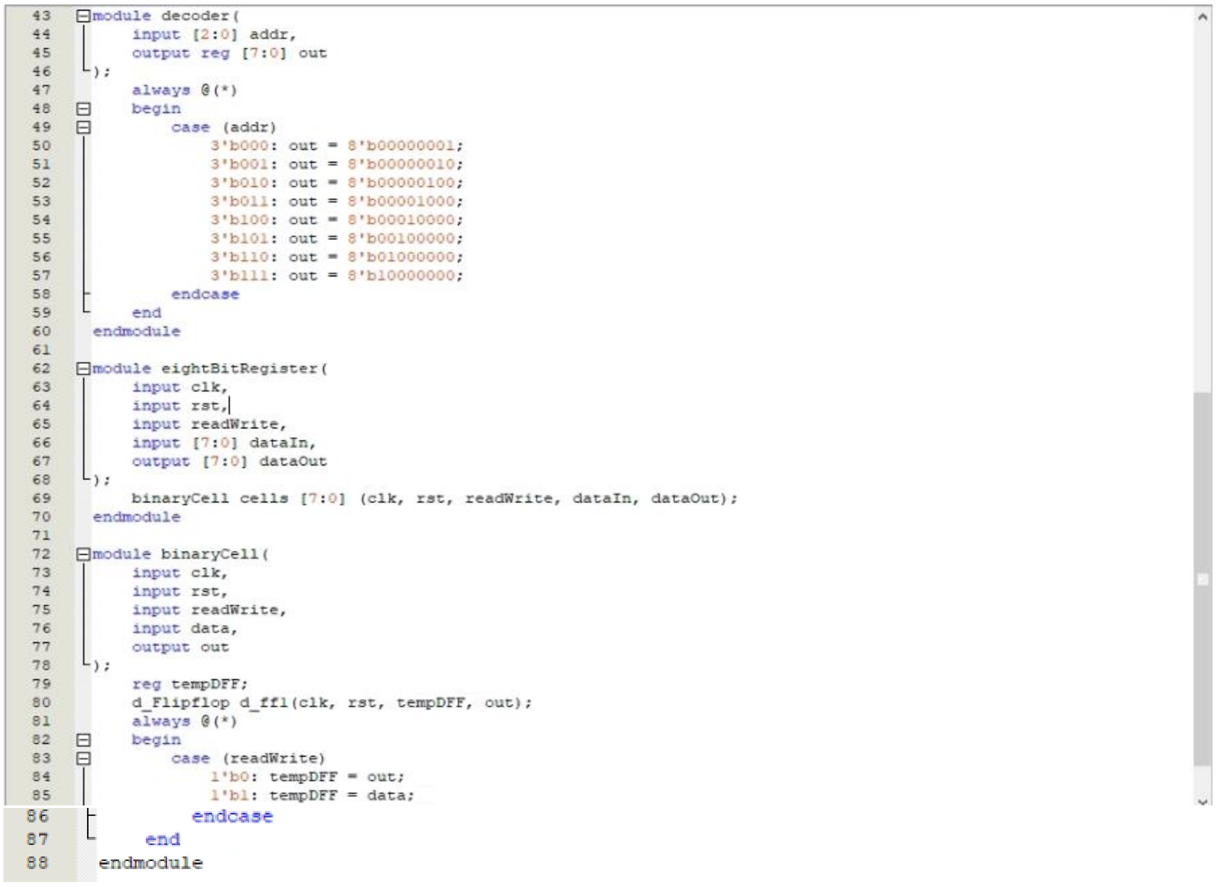
RAM

Ram is a memory unit. We are able to write some data, read some data and also change some data in the ram. The difference between ram and rom comes from that the data is changeable in a ram but not a rom. We designed a ram, tested in the FPGA and finally integrated to our cpu. Our ram is 8-bit.



```
module ram(
    input clk,
    input rst,
    input readWrite,
    input cs,
    input [2:0] addr,
    input [7:0] dataIn,
    output reg [7:0] dataOut
);
    wire [7:0] tempRst;
    wire [7:0] tempWrite;
    decoder decoderRst(addr, tempRst);
    decoder decoderWrite(addr, tempWrite);
wire [7:0] tempRegOut1, tempRegOut2, tempRegOut3, tempRegOut4, tempRegOut5, tempRegOut6, tempRegOut7, tempRegOut
    eightBitRegister addr1(clk, ~tempRst[0] | rst, tempWrite[0] & readWrite, dataIn, tempRegOut1);
    eightBitRegister addr2(clk, ~tempRst[1] | rst, tempWrite[1] & readWrite, dataIn, tempRegOut2);
    eightBitRegister addr3(clk, ~tempRst[2] | rst, tempWrite[2] & readWrite, dataIn, tempRegOut3);
    eightBitRegister addr4(clk, ~tempRst[3] | rst, tempWrite[3] & readWrite, dataIn, tempRegOut4);

    eightBitRegister addr5(clk, ~tempRst[4] | rst, tempWrite[4] & readWrite, dataIn, tempRegOut5);
    eightBitRegister addr6(clk, ~tempRst[5] | rst, tempWrite[5] & readWrite, dataIn, tempRegOut6);
    eightBitRegister addr7(clk, ~tempRst[6] | rst, tempWrite[6] & readWrite, dataIn, tempRegOut7);
    eightBitRegister addr8(clk, ~tempRst[7] | rst, tempWrite[7] & readWrite, dataIn, tempRegOut8);
    always @(*)
    begin
        if (!cs) dataOut <= 8'b00000000;
        else begin
            case (addr)
                3'b000: dataOut = tempRegOut1;
                3'b001: dataOut = tempRegOut2;
                3'b010: dataOut = tempRegOut3;
                3'b011: dataOut = tempRegOut4;

                3'b100: dataOut = tempRegOut5;
                3'b101: dataOut = tempRegOut6;
                3'b110: dataOut = tempRegOut7;
                3'b111: dataOut = tempRegOut8;
            endcase
        end
    end
endmodule
```

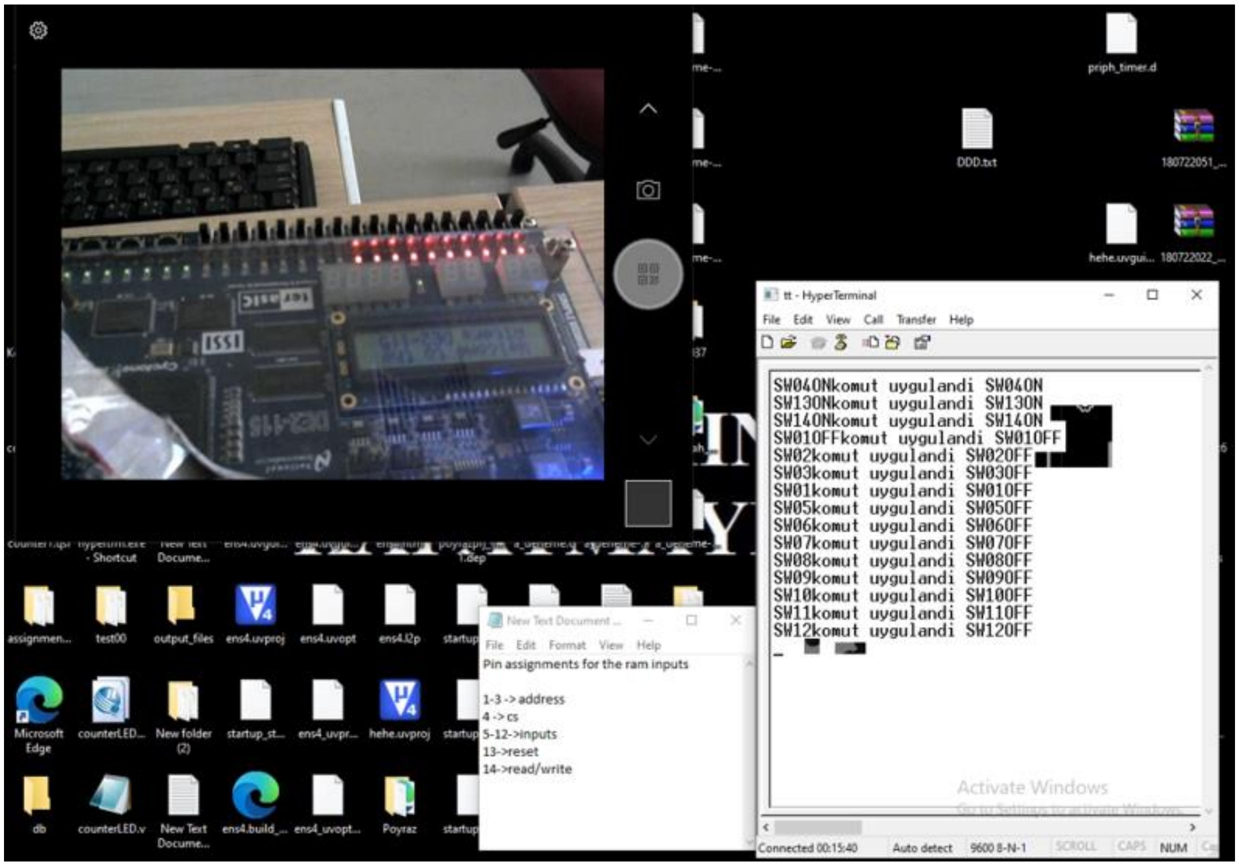
Figure-1: Verilog code for the ram (1)



```
module decoder(
    input [2:0] addr,
    output reg [7:0] out
);
    always @(*)
    begin
        case (addr)
            3'b000: out = 8'b00000001;
            3'b001: out = 8'b00000010;
            3'b010: out = 8'b00000100;
            3'b011: out = 8'b00001000;
            3'b100: out = 8'b00010000;
            3'b101: out = 8'b00100000;
            3'b110: out = 8'b01000000;
            3'b111: out = 8'b10000000;
        endcase
    end
endmodule

module eightBitRegister(
    input clk,
    input rst,
    input readWrite,
    input [7:0] dataIn,
    output [7:0] dataOut
);
    binaryCell cells [7:0] (clk, rst, readWrite, dataIn, dataOut);
endmodule

module binaryCell(
    input clk,
    input rst,
    input readWrite,
    input data,
    output out
);
    reg tempDFF;
    d_Flipflop d_ff1(clk, rst, tempDFF, out);
    always @(*)
    begin
        case (readWrite)
            1'b0: tempDFF = out;
            1'b1: tempDFF = data;
        endcase
    end
endmodule
```

Figure-2: Verilog code for the ram (2)

TESTS FOR RAM:



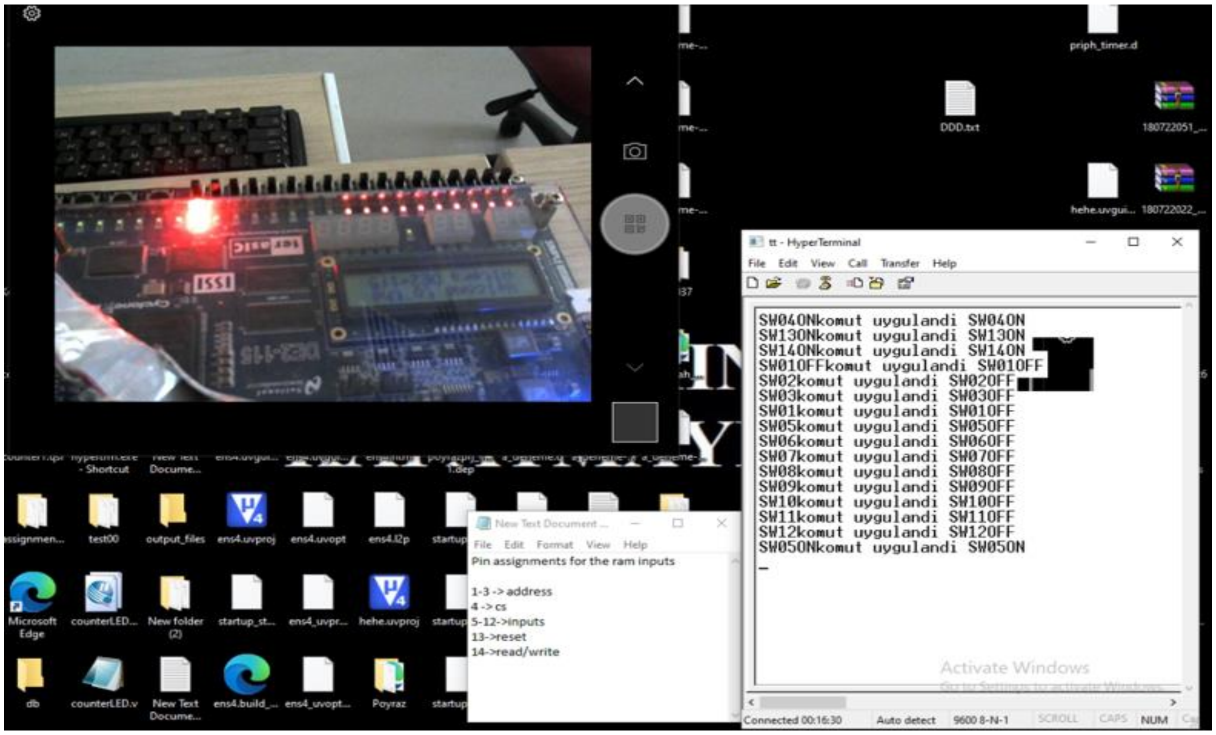
```
SW04ONkomut uygulandi SW04ON
SW13ONkomut uygulandi SW13ON
SW14ONkomut uygulandi SW14ON
SW01OFFkomut uygulandi SW01OFF
SW02komut uygulandi SW02OFF
SW03komut uygulandi SW03OFF
SW01komut uygulandi SW01OFF
SW05komut uygulandi SW05OFF
SW06komut uygulandi SW06OFF
SW07komut uygulandi SW07OFF
SW08komut uygulandi SW08OFF
SW09komut uygulandi SW09OFF
SW10komut uygulandi SW10OFF
SW11komut uygulandi SW11OFF
SW12komut uygulandi SW12OFF
_
```

```
Pin assignments for the ram inputs

1-3 -> address
4 -> cs
5-12->inputs
13->reset
14->read/write
```

tt - HyperTerminal
File Edit View Call Transfer Help
Connected 00:15:40 Auto detect 9600 8-N-1 SCROLL CAPS NUM
New Text Document ...
File Edit Format View Help
Activate Windows

Figure-1:

- Initially, we opened cs, reset and write inputs.
- Then, 000 ram address was set.
- Then , all data inputs between 5-12 switches was set to 0.

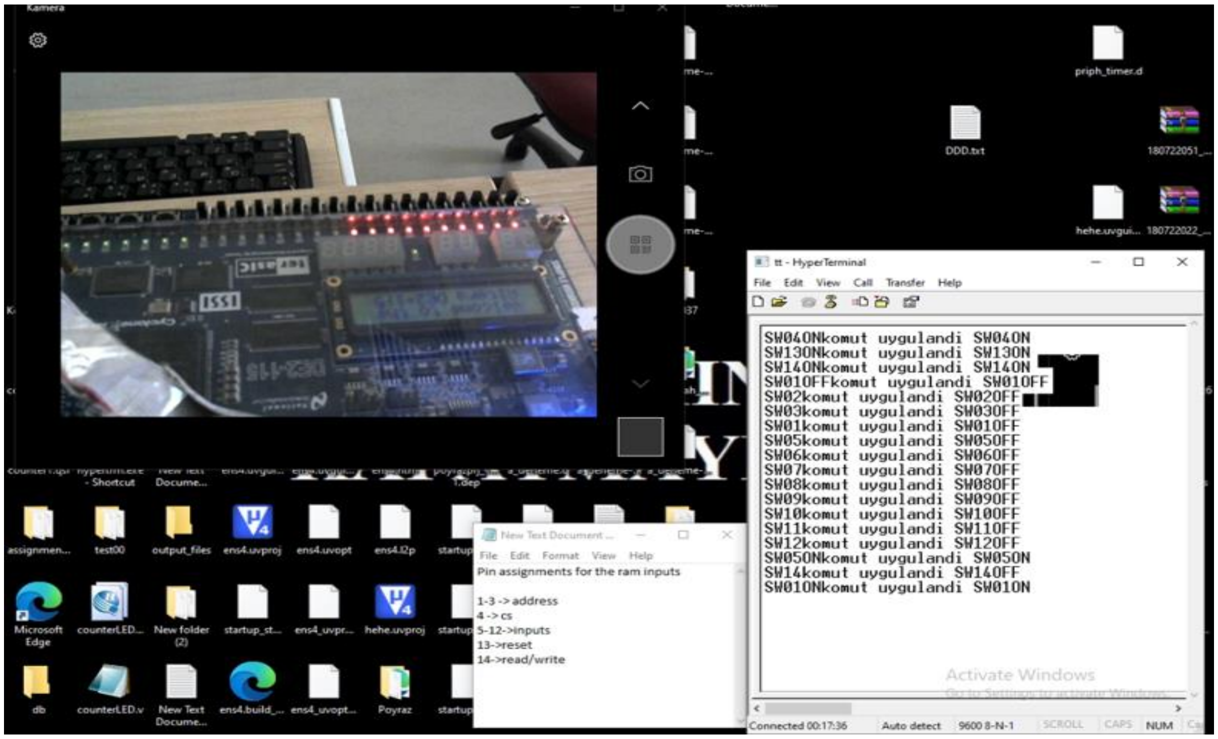


```
SW04ONkomut uygulandi SW04ON
SW13ONkomut uygulandi SW13ON
SW14ONkomut uygulandi SW14ON
SW01OFFkomut uygulandi SW01OFF
SW02komut uygulandi SW02OFF
SW03komut uygulandi SW03OFF
SW01komut uygulandi SW01OFF
SW05komut uygulandi SW05OFF
SW06komut uygulandi SW06OFF
SW07komut uygulandi SW07OFF
SW08komut uygulandi SW08OFF
SW09komut uygulandi SW09OFF
SW10komut uygulandi SW10OFF
SW11komut uygulandi SW11OFF
SW12komut uygulandi SW12OFF
SW05ONkomut uygulandi SW05ON
_
```

```
Pin assignments for the ram inputs

1-3 -> address
4 -> cs
5-12->inputs
13->reset
14->read/write
```

Figure-2:

- We opened the switch 5 that is connected first bit of data input and connected led is turned on.

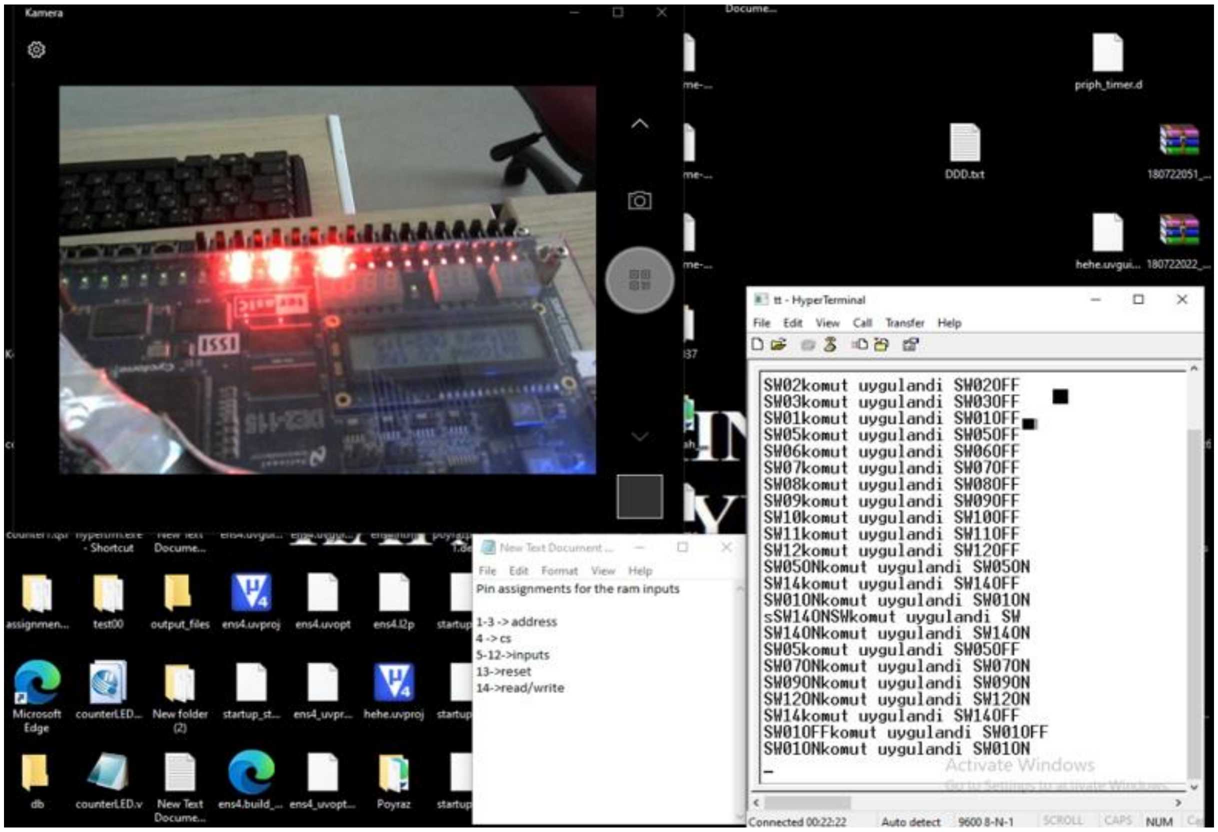


```
SW04ONkomut uygulandi SW04ON
SW13ONkomut uygulandi SW13ON
SW14ONkomut uygulandi SW14ON
SW01OFFkomut uygulandi SW01OFF
SW02komut uygulandi SW02OFF
SW03komut uygulandi SW03OFF
SW01komut uygulandi SW01OFF
SW05komut uygulandi SW05OFF
SW06komut uygulandi SW06OFF
SW07komut uygulandi SW07OFF
SW08komut uygulandi SW08OFF
SW09komut uygulandi SW09OFF
SW10komut uygulandi SW10OFF
SW11komut uygulandi SW11OFF
SW12komut uygulandi SW12OFF
SW05ONkomut uygulandi SW05ON
SW14komut uygulandi SW14OFF
SW01ONkomut uygulandi SW01ON
```

```
Pin assignments for the ram inputs

1-3 -> address
4 -> cs
5-12->inputs
13->reset
14->read/write
```

Figure-3:

- We closed the write signal becase we will change the ram address and previous open switches should not affect the new ram address.
- We chaged the ram address as 001, then all data bits again zero.

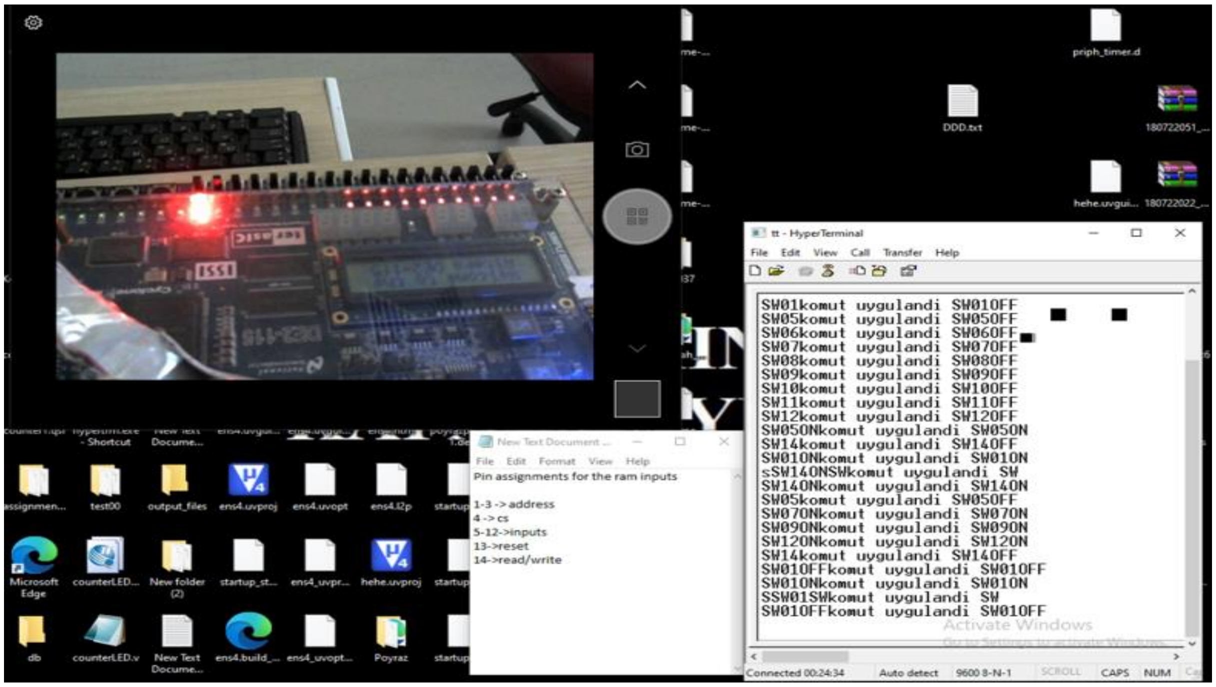


```
Pin assignments for the ram inputs

1-3 -> address
4 -> cs
5-12->inputs
13->reset
14->read/write
```

```
SW02komut uygulandi SW02OFF
SW03komut uygulandi SW03OFF
SW01komut uygulandi SW01OFF
SW05komut uygulandi SW05OFF
SW06komut uygulandi SW06OFF
SW07komut uygulandi SW07OFF
SW08komut uygulandi SW08OFF
SW09komut uygulandi SW09OFF
SW10komut uygulandi SW10OFF
SW11komut uygulandi SW11OFF
SW12komut uygulandi SW12OFF
SW05ONkomut uygulandi SW05ON
SW14komut uygulandi SW14OFF
SW01ONkomut uygulandi SW01ON
sSW14ONSWkomut uygulandi SW
SW14ONkomut uygulandi SW14ON
SW05komut uygulandi SW05OFF
SW07ONkomut uygulandi SW07ON
SW09ONkomut uygulandi SW09ON
SW12ONkomut uygulandi SW12ON
SW14komut uygulandi SW14OFF
SW01OFFkomut uygulandi SW01OFF
SW01ONkomut uygulandi SW01ON
_
```

Figure-4:

- We opened write signal because we need this signal to write some data to a ram address.
- Then we closed the switch 5 (it was open)
- We opened switch 7, 9 and 12 and the corresponding leds were turned on.

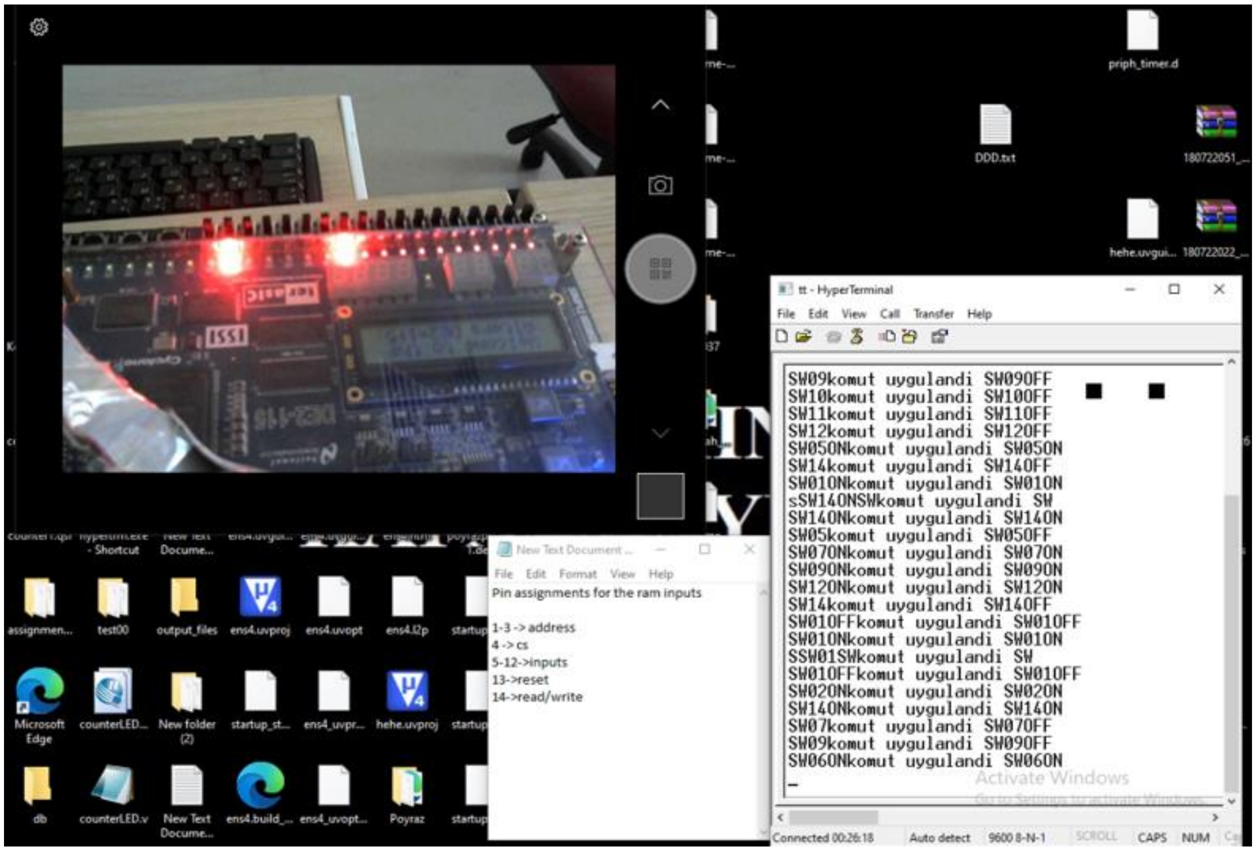


```
SW01komut uygulandi SW01OFF
SW05komut uygulandi SW05OFF
SW06komut uygulandi SW06OFF
SW07komut uygulandi SW07OFF
SW08komut uygulandi SW08OFF
SW09komut uygulandi SW09OFF
SW10komut uygulandi SW10OFF
SW11komut uygulandi SW11OFF
SW12komut uygulandi SW12OFF
SW05ONkomut uygulandi SW05ON
SW14komut uygulandi SW14OFF
SW01ONkomut uygulandi SW01ON
sSW14ONSWkomut uygulandi SW
SW14ONkomut uygulandi SW14ON
SW05komut uygulandi SW05OFF
SW07ONkomut uygulandi SW07ON
SW09ONkomut uygulandi SW09ON
SW12ONkomut uygulandi SW12ON
SW14komut uygulandi SW14OFF
SW01OFFkomut uygulandi SW01OFF
SW01ONkomut uygulandi SW01ON
SSW01SWkomut uygulandi SW
SW01OFFkomut uygulandi SW01OFF
```

```
Pin assignments for the ram inputs

1-3 -> address
4 -> cs
5-12->inputs
13->reset
14->read/write
```

Figure-5:

- We closed the write signal and changed the ram address to 000 again, then only switch 5's output led was turned on as we expected.

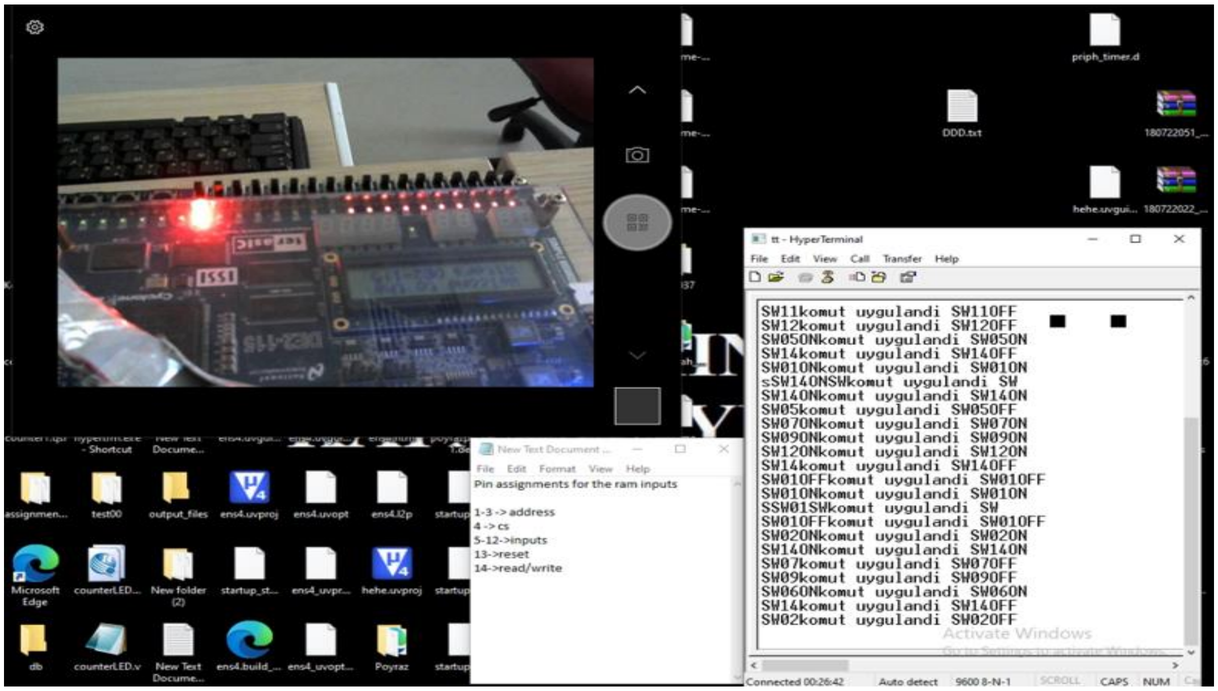


```
SW09komut uygulandi SW09OFF
SW10komut uygulandi SW10OFF
SW11komut uygulandi SW11OFF
SW12komut uygulandi SW12OFF
SW05ONkomut uygulandi SW05ON
SW14komut uygulandi SW14OFF
SW01ONkomut uygulandi SW01ON
sSW14ONSWkomut uygulandi SW
SW14ONkomut uygulandi SW14ON
SW05komut uygulandi SW05OFF
SW07ONkomut uygulandi SW07ON
SW09ONkomut uygulandi SW09ON
SW12ONkomut uygulandi SW12ON
SW14komut uygulandi SW14OFF
SW01OFFkomut uygulandi SW01OFF
SW01ONkomut uygulandi SW01ON
SSW01SWkomut uygulandi SW
SW01OFFkomut uygulandi SW01OFF
SW02ONkomut uygulandi SW02ON
SW14ONkomut uygulandi SW14ON
SW07komut uygulandi SW07OFF
SW09komut uygulandi SW09OFF
SW06ONkomut uygulandi SW06ON
_
```

```
Pin assignments for the ram inputs

1-3 -> address
4 -> cs
5-12->inputs
13->reset
14->read/write
```

Figure-6:

- Still write signal is closed. We changed the ram address to 010 with opening the switch 2.
- Then, we will open the write signal.
- We closed the switch 7 and 9, opened the switch 6.
- Finally only switch 6 (that is connected to 6th bit of data)'s output led is turned on and switch 12's output led is. (Switch 12 was open from previous address)



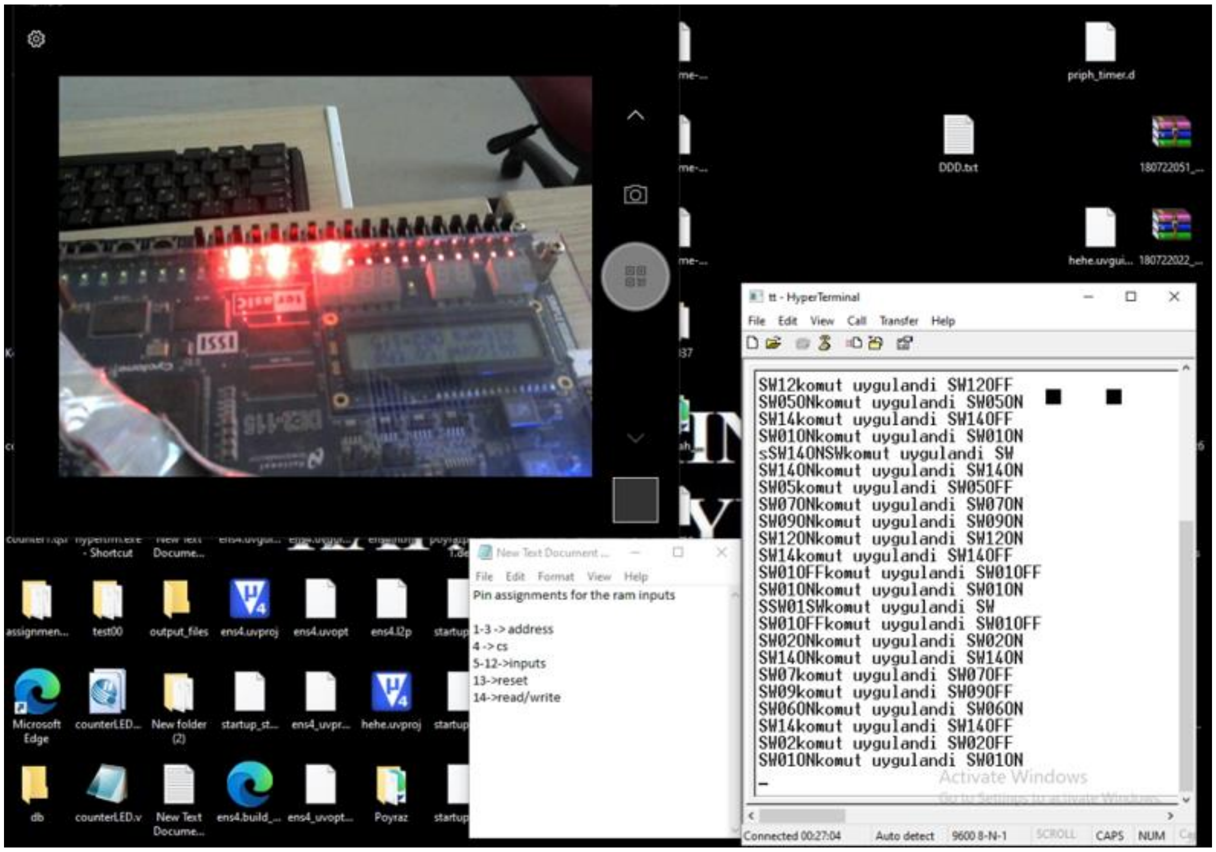
```
Pin assignments for the ram inputs

1-3 -> address
4 -> cs
5-12->inputs
13->reset
14->read/write
```

```
SW11komut uygulandi SW11OFF
SW12komut uygulandi SW12OFF
SW05ONkomut uygulandi SW05ON
SW14komut uygulandi SW14OFF
SW01ONkomut uygulandi SW01ON
sSW14ONSWkomut uygulandi SW
SW14ONkomut uygulandi SW14ON
SW05komut uygulandi SW05OFF
SW07ONkomut uygulandi SW07ON
SW09ONkomut uygulandi SW09ON
SW12ONkomut uygulandi SW12ON
SW14komut uygulandi SW14OFF
SW01OFFkomut uygulandi SW01OFF
SW01ONkomut uygulandi SW01ON
SSW01SWkomut uygulandi SW
SW01OFFkomut uygulandi SW01OFF
SW02ONkomut uygulandi SW02ON
SW14ONkomut uygulandi SW14ON
SW07komut uygulandi SW07OFF
SW09komut uygulandi SW09OFF
SW06ONkomut uygulandi SW06ON
SW14komut uygulandi SW14OFF
SW02komut uygulandi SW02OFF
```

tt - HyperTerminal
File Edit View Call Transfer Help
Connected 00:26:42 Auto detect 9600 8-N-1 SCROLL CAPS NUM
New Text Document ...
File Edit Format View Help

Figure-7:

- We closed the write signal then changed the ram address as 000.
- Again, only switch 5's output led is on as we expected.

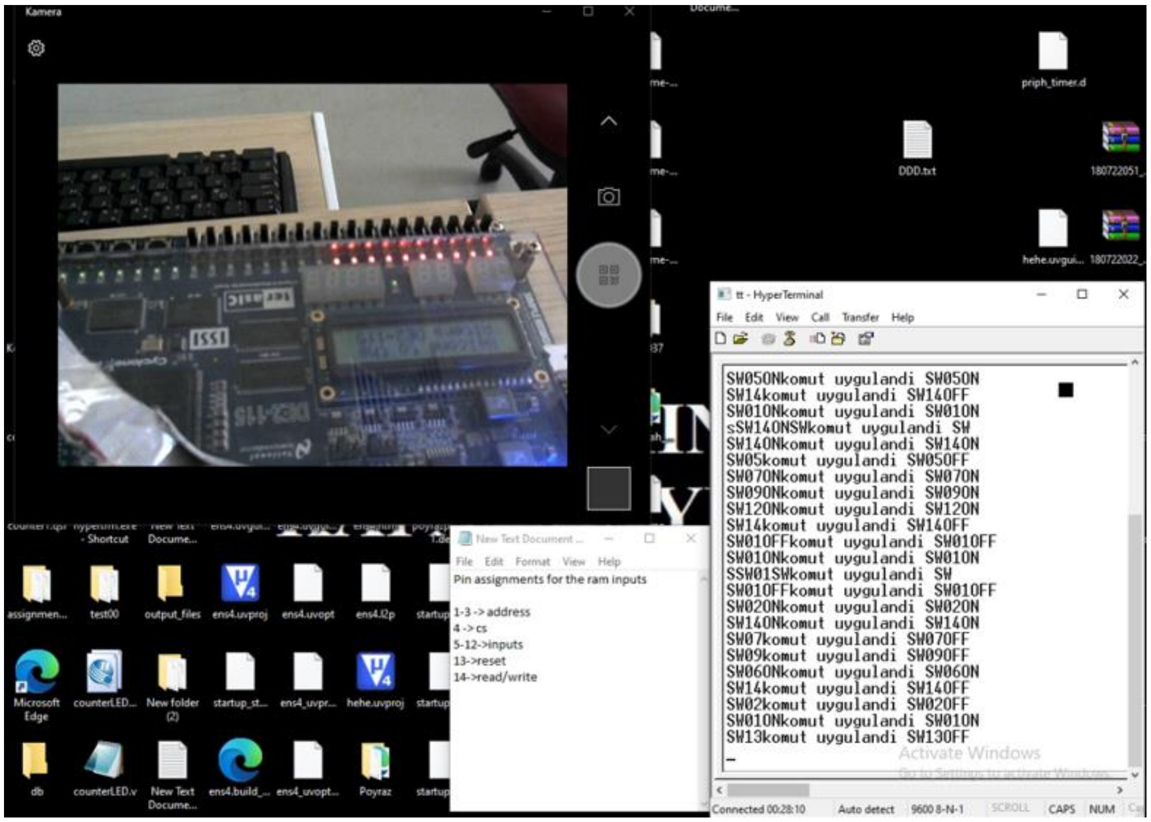

Pin assignments for the ram inputs

1-3 -> address
4 -> cs
5-12->inputs
13->reset
14->read/write

```
SW12komut uygulandi SW12OFF
SW05ONkomut uygulandi SW05ON
SW14komut uygulandi SW14OFF
SW01ONkomut uygulandi SW01ON
sSW14ONSWkomut uygulandi SW
SW14ONkomut uygulandi SW14ON
SW05komut uygulandi SW05OFF
SW07ONkomut uygulandi SW07ON
SW09ONkomut uygulandi SW09ON
SW12ONkomut uygulandi SW12ON
SW14komut uygulandi SW14OFF
SW01OFFkomut uygulandi SW01OFF
SW01ONkomut uygulandi SW01ON
SSW01SWkomut uygulandi SW
SW01OFFkomut uygulandi SW01OFF
SW02ONkomut uygulandi SW02ON
SW14ONkomut uygulandi SW14ON
SW07komut uygulandi SW07OFF
SW09komut uygulandi SW09OFF
SW06ONkomut uygulandi SW06ON
SW14komut uygulandi SW14OFF
SW02komut uygulandi SW02OFF
SW01ONkomut uygulandi SW01ON
_
```

Figure-8:

- We changed the ram address as 001 then switch 7, 9 and 12s' output leds were turned on as we expected.



```
SW05ONkomut uygulandi SW05ON
SW14komut uygulandi SW14OFF
SW01ONkomut uygulandi SW01ON
sSW14ONSWkomut uygulandi SW
SW14ONkomut uygulandi SW14ON
SW05komut uygulandi SW05OFF
SW07ONkomut uygulandi SW07ON
SW09ONkomut uygulandi SW09ON
SW12ONkomut uygulandi SW12ON
SW14komut uygulandi SW14OFF
SW01OFFkomut uygulandi SW01OFF
SW01ONkomut uygulandi SW01ON
SSW01SWkomut uygulandi SW
SW01OFFkomut uygulandi SW01OFF
SW02ONkomut uygulandi SW02ON
SW14ONkomut uygulandi SW14ON
SW07komut uygulandi SW07OFF
SW09komut uygulandi SW09OFF
SW06ONkomut uygulandi SW06ON
SW14komut uygulandi SW14OFF
SW02komut uygulandi SW02OFF
SW01ONkomut uygulandi SW01ON
SW13komut uygulandi SW13OFF
_
```

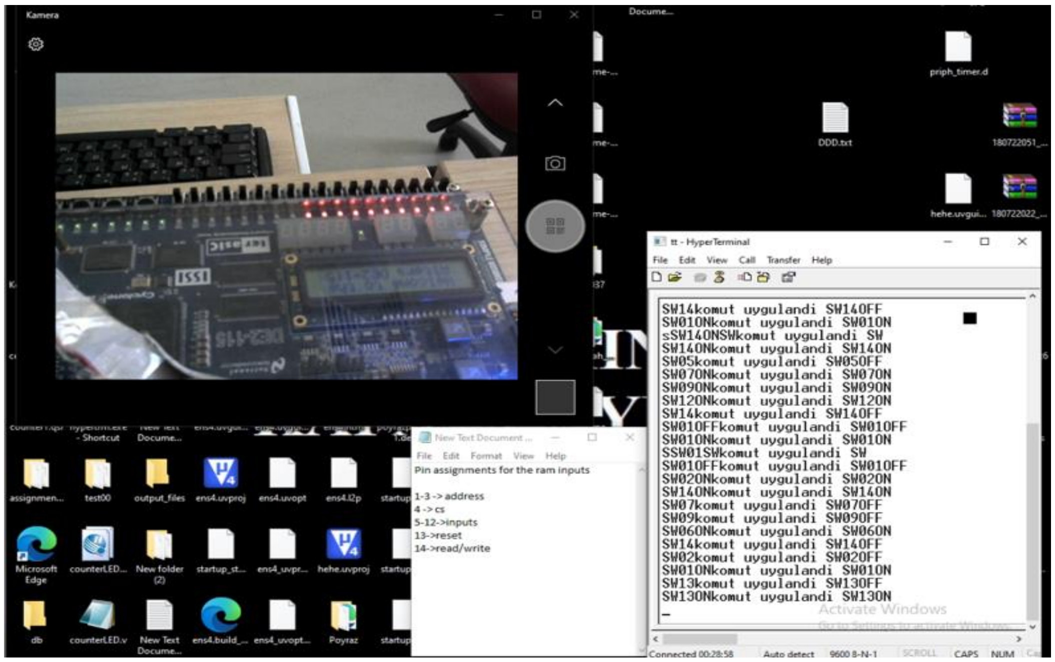
Connected 00:28:10 Auto detect 9600 8-N-1

Pin assignments for the ram inputs

1-3 -> address
4 -> cs
5-12->inputs
13->reset
14->read/write

Figure-9:

- We activated the reset signal then all leds are off as we expected.



```
SW14komut uygulandi SW14OFF
SW01ONkomut uygulandi SW01ON
sSW14ONSWkomut uygulandi SW
SW14ONkomut uygulandi SW14ON
SW05komut uygulandi SW05OFF
SW07ONkomut uygulandi SW07ON
SW09ONkomut uygulandi SW09ON
SW12ONkomut uygulandi SW12ON
SW14komut uygulandi SW14OFF
SW01OFFkomut uygulandi SW01OFF
SW01ONkomut uygulandi SW01ON
SSW01SWkomut uygulandi SW
SW01OFFkomut uygulandi SW01OFF
SW02ONkomut uygulandi SW02ON
SW14ONkomut uygulandi SW14ON
SW07komut uygulandi SW07OFF
SW09komut uygulandi SW09OFF
SW06ONkomut uygulandi SW06ON
SW14komut uygulandi SW14OFF
SW02komut uygulandi SW02OFF
SW01ONkomut uygulandi SW01ON
SW13komut uygulandi SW13OFF
SW13ONkomut uygulandi SW13ON
_
```

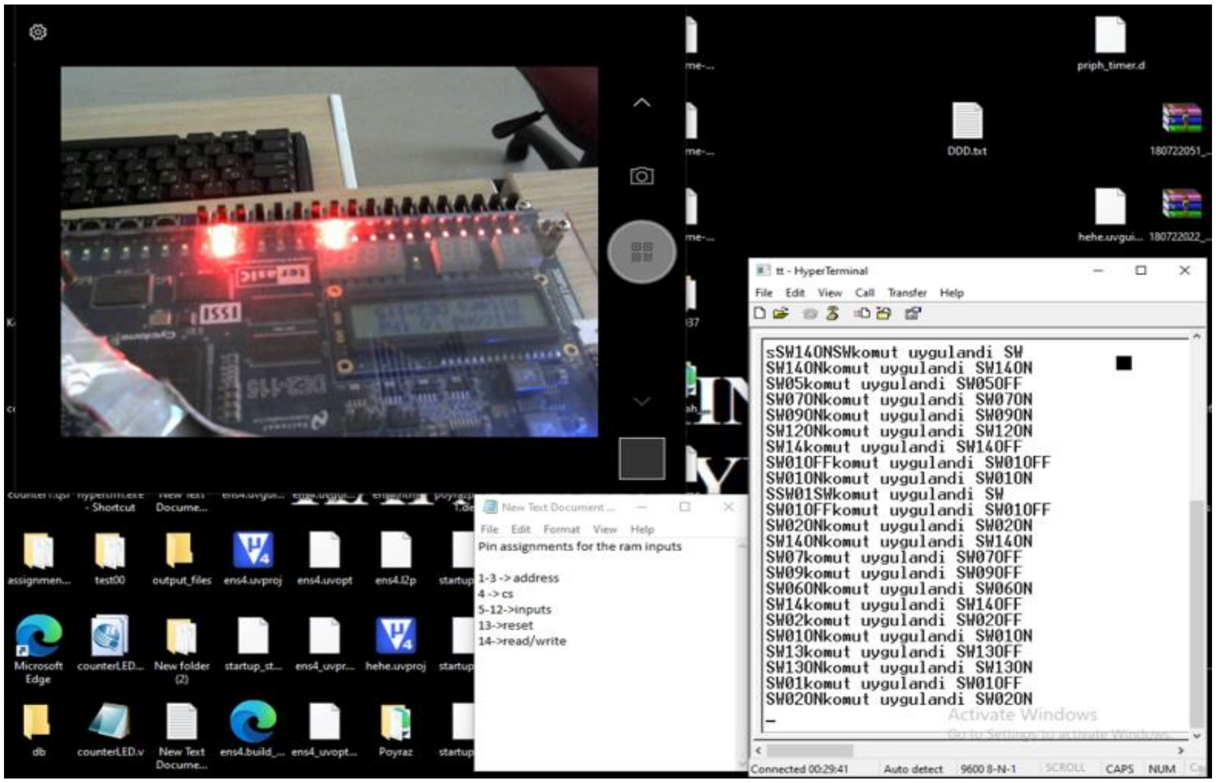
Connected 00:28:58 Auto detect 9600 8-N-1

Pin assignments for the ram inputs

1-3 -> address
4 -> cs
5-12->inputs
13->reset
14->read/write

Figure-10:

- We deactivated the reset signal again but all leds for the current ram address are still off as we expected.



```
sSW14ONSWkomut uygulandi SW
SW14ONkomut uygulandi SW14ON
SW05komut uygulandi SW05OFF
SW07ONkomut uygulandi SW07ON
SW09ONkomut uygulandi SW09ON
SW12ONkomut uygulandi SW12ON
SW14komut uygulandi SW14OFF
SW01OFFkomut uygulandi SW01OFF
SW01ONkomut uygulandi SW01ON
SSW01SWkomut uygulandi SW
SW01OFFkomut uygulandi SW01OFF
SW02ONkomut uygulandi SW02ON
SW14ONkomut uygulandi SW14ON
SW07komut uygulandi SW07OFF
SW09komut uygulandi SW09OFF
SW06ONkomut uygulandi SW06ON
SW14komut uygulandi SW14OFF
SW02komut uygulandi SW02OFF
SW01ONkomut uygulandi SW01ON
SW13komut uygulandi SW13OFF
SW13ONkomut uygulandi SW13ON
SW01komut uygulandi SW01OFF
SW02ONkomut uygulandi SW02ON
_
```

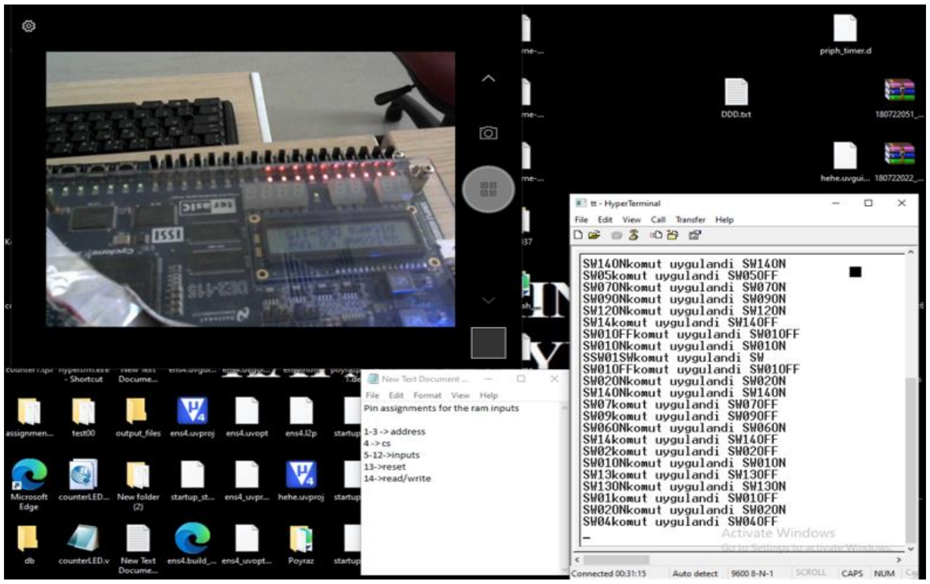
Connected 00:29:41 Auto detect 9600 8-N-1

Pin assignments for the ram inputs

1-3 -> address
4 -> cs
5-12->inputs
13->reset
14->read/write

Figure-11:

- We changed the ram address as 010, correct leds continue to light.

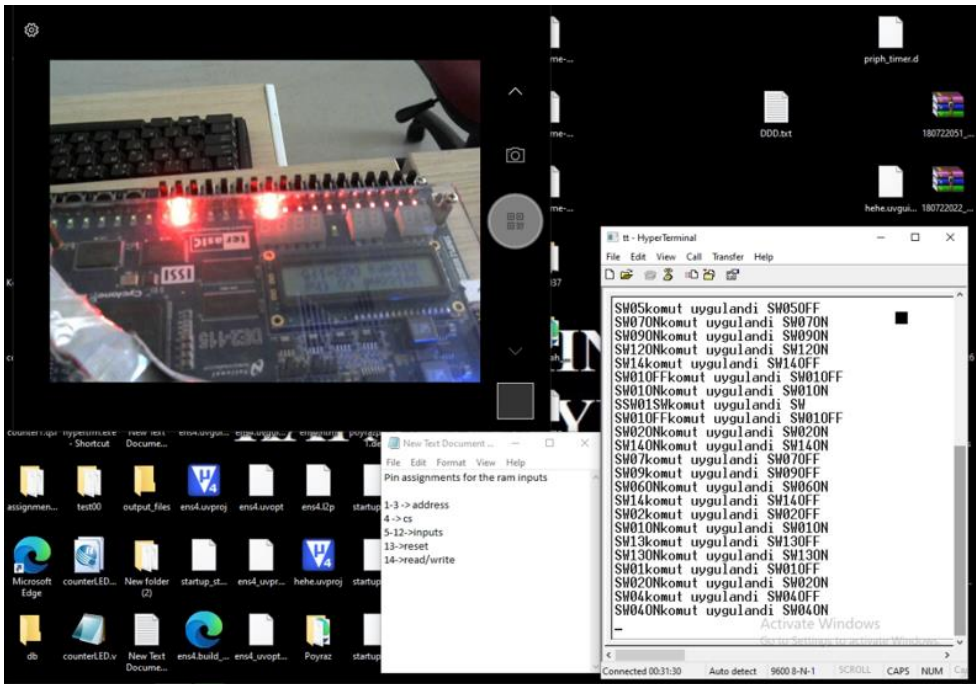


```
Pin assignments for the ram inputs

1-3 -> address
4 -> cs
5-12->inputs
13->reset
14->read/write
```

```
SW14ONkomut uygulandi SW14ON
SW05komut uygulandi SW05OFF
SW07ONkomut uygulandi SW07ON
SW09ONkomut uygulandi SW09ON
SW12ONkomut uygulandi SW12ON
SW14komut uygulandi SW14OFF
SW01OFFkomut uygulandi SW01OFF
SW01ONkomut uygulandi SW01ON
SSW01SWkomut uygulandi SW
SW01OFFkomut uygulandi SW01OFF
SW02ONkomut uygulandi SW02ON
SW14ONkomut uygulandi SW14ON
SW07komut uygulandi SW07OFF
SW09komut uygulandi SW09OFF
SW06ONkomut uygulandi SW06ON
SW14komut uygulandi SW14OFF
SW02komut uygulandi SW02OFF
SW01ONkomut uygulandi SW01ON
SW13komut uygulandi SW13OFF
SW13ONkomut uygulandi SW13ON
SW01komut uygulandi SW01OFF
SW02ONkomut uygulandi SW02ON
SW04komut uygulandi SW04OFF
_
```

Figure-12:

- We closed the cs input connected to switch 4 then current ram address's data will not be selected.



```
Pin assignments for the ram inputs

1-3 -> address
4 -> cs
5-12->inputs
13->reset
14->read/write
```

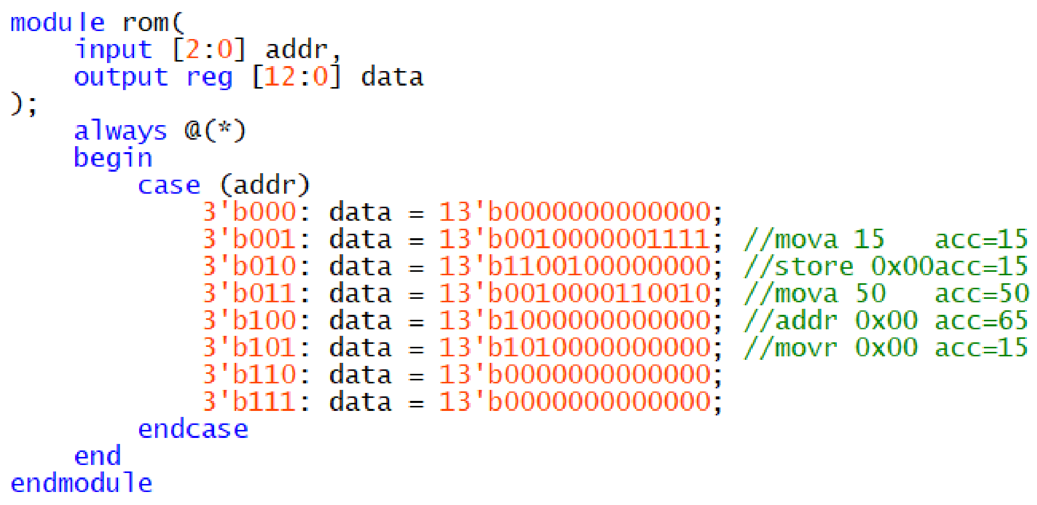
```
SW05komut uygulandi SW05OFF
SW07ONkomut uygulandi SW07ON
SW09ONkomut uygulandi SW09ON
SW12ONkomut uygulandi SW12ON
SW14komut uygulandi SW14OFF
SW01OFFkomut uygulandi SW01OFF
SW01ONkomut uygulandi SW01ON
SSW01SWkomut uygulandi SW
SW01OFFkomut uygulandi SW01OFF
SW02ONkomut uygulandi SW02ON
SW14ONkomut uygulandi SW14ON
SW07komut uygulandi SW07OFF
SW09komut uygulandi SW09OFF
SW06ONkomut uygulandi SW06ON
SW14komut uygulandi SW14OFF
SW02komut uygulandi SW02OFF
SW01ONkomut uygulandi SW01ON
SW13komut uygulandi SW13OFF
SW13ONkomut uygulandi SW13ON
SW01komut uygulandi SW01OFF
SW02ONkomut uygulandi SW02ON
SW04komut uygulandi SW04OFF
SW04ONkomut uygulandi SW04ON
_
```

Figure-13:

- We opened it, leds continued to light as we expected. This characteristics of cs is different from reset signal.

ROM

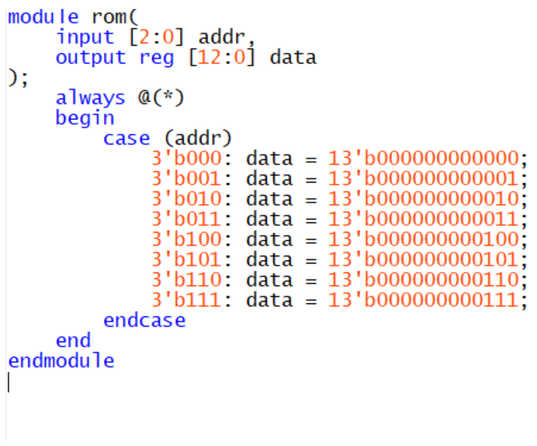
Rom is used to store instruction codes for our CPU. Instructions are 13-bit (1 bit is for literal/address selection, 4 bits are for operation code, 8 bits are for data/address) and 8 instruction is stored inside rom for now. Every instruction is stored at registers and every instruction has addresses. 3 bits are used for addresses because 8 instructions are stored in ROM.



```
module rom(
    input [2:0] addr,
    output reg [12:0] data
);
    always @(*)
    begin
        case (addr)
            3'b000: data = 13'b0000000000000;
            3'b001: data = 13'b0010000001111; //mova 15   acc=15
            3'b010: data = 13'b1100100000000; //store 0x00acc=15
            3'b011: data = 13'b0010000110010; //mova 50   acc=50
            3'b100: data = 13'b1000000000000; //addr 0x00 acc=65
            3'b101: data = 13'b1010000000000; //movr 0x00 acc=15
            3'b110: data = 13'b0000000000000;
            3'b111: data = 13'b0000000000000;
        endcase
    end
endmodule
```

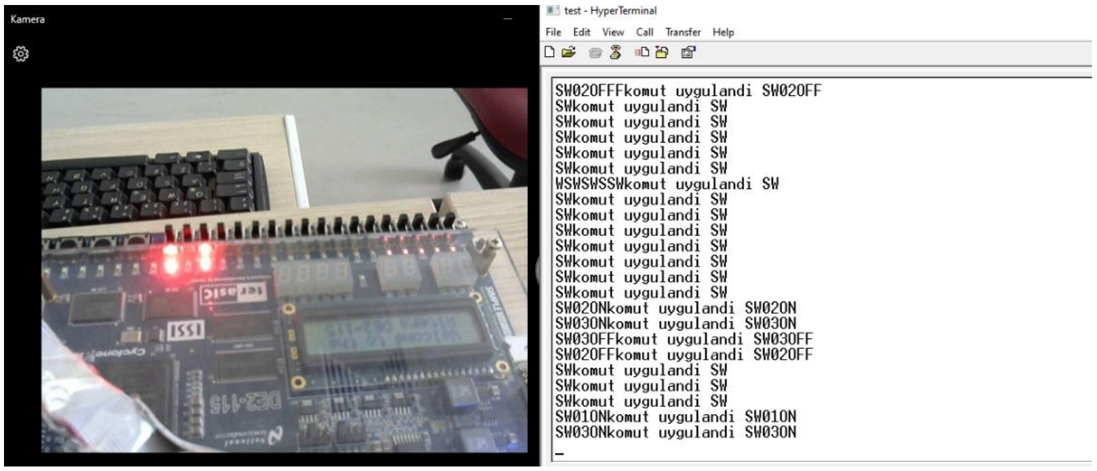
Figure-1: Verilog Code for ROM

3 leds are connected as output for ROM, 3 switches are connected as inputs for ROM.



```
module rom(
    input [2:0] addr,
    output reg [12:0] data
);
    always @(*)
    begin
        case (addr)
            3'b000: data = 13'b0000000000000;
            3'b001: data = 13'b0000000000001;
            3'b010: data = 13'b0000000000010;
            3'b011: data = 13'b0000000000011;
            3'b100: data = 13'b0000000000100;
            3'b101: data = 13'b0000000000101;
            3'b110: data = 13'b0000000000110;
            3'b111: data = 13'b0000000000111;
        endcase
    end
endmodule
```

Figure-2: Tested Verilog Code for ROM



```
SW020FFFkomut uygulandi SW020FF
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
WSWSWSSWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SW020Nkomut uygulandi SW020N
SW030Nkomut uygulandi SW030N
SW030FFkomut uygulandi SW030FF
SW020FFkomut uygulandi SW020FF
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SW010Nkomut uygulandi SW010N
SW030Nkomut uygulandi SW030N
_
```

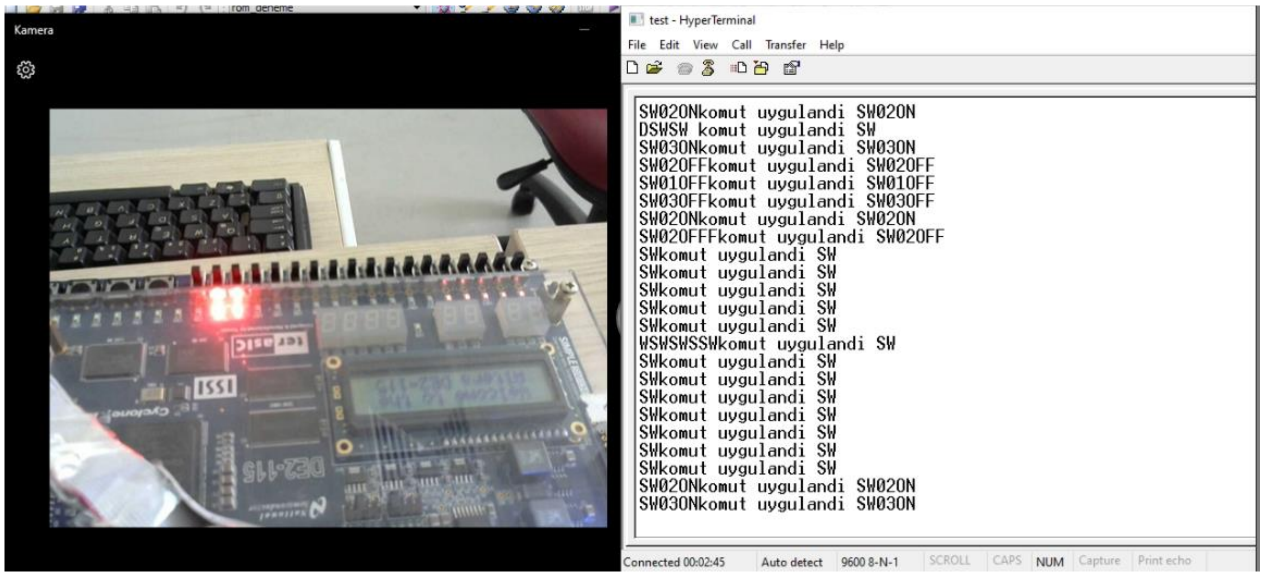
Figure-3: First Test Case for ROM

First test (Figure-3) is done with these values:

Switches (3,2,1): 101

Leds (3,2,1): 101

We can see the correct result when we give 101 as address, led lights gave the result as 101 because in address 101, we have 101 data.



```
SW020Nkomut uygulandi SW020N
DSWSW komut uygulandi SW
SW030Nkomut uygulandi SW030N
SW020FFkomut uygulandi SW020FF
SW010FFkomut uygulandi SW010FF
SW030FFkomut uygulandi SW030FF
SW020Nkomut uygulandi SW020N
SW020FFFkomut uygulandi SW020FF
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
WSWSWSSWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SW020Nkomut uygulandi SW020N
SW030Nkomut uygulandi SW030N
```

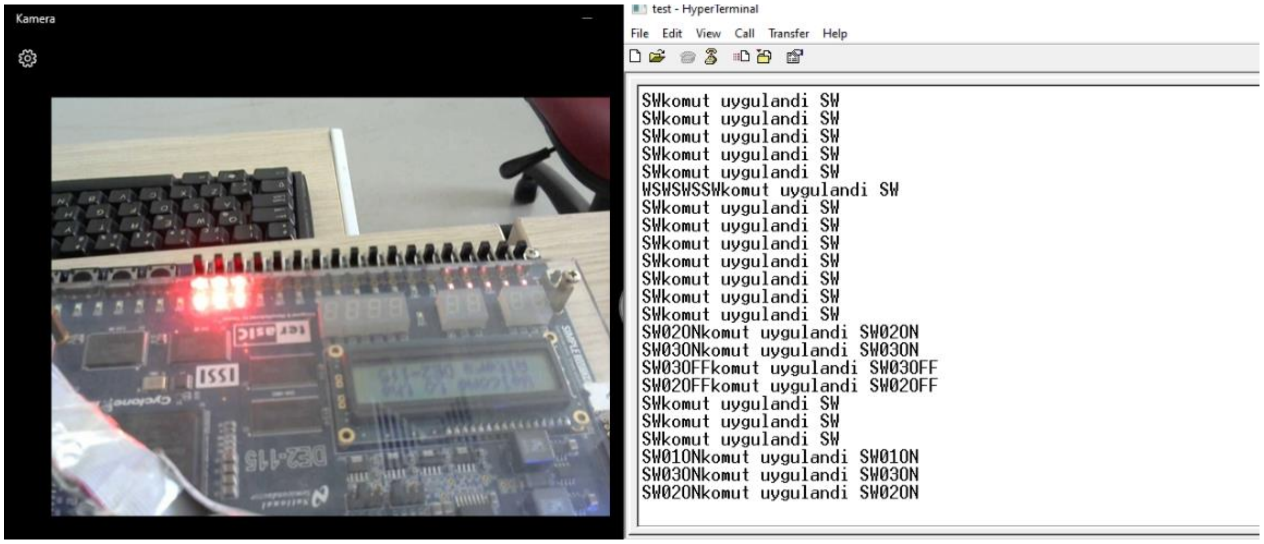
Figure-4: Second Test Case for ROM

Second test (Figure-4) is done with these values:

Switches (3,2,1): 110

Leds (3,2,1): 110

We can see the correct result when we give 110 as address, led lights gave the result as 110 because in address 110, we have 110 data.



```
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
WSWSWSSWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SW02ONkomut uygulandi SW02ON
SW03ONkomut uygulandi SW03ON
SW03OFFkomut uygulandi SW03OFF
SW02OFFkomut uygulandi SW02OFF
SWkomut uygulandi SW
SWkomut uygulandi SW
SWkomut uygulandi SW
SW01ONkomut uygulandi SW01ON
SW03ONkomut uygulandi SW03ON
SW02ONkomut uygulandi SW02ON
```

Figure-5: Third Test Case for ROM

Third test (Figure-5) is done with these values:

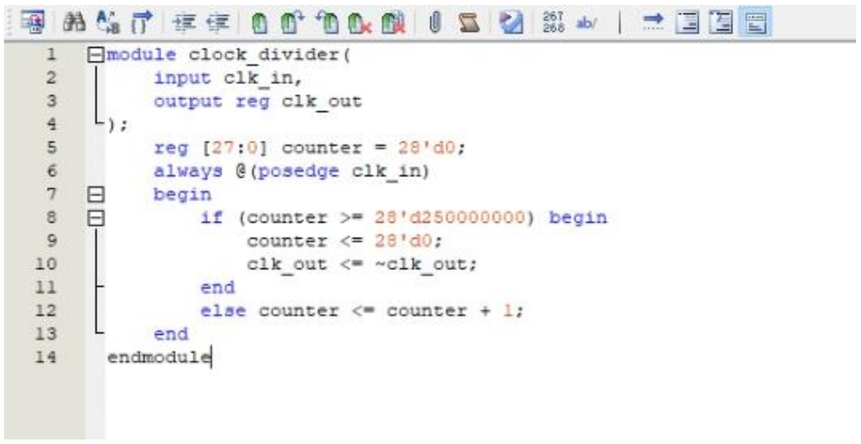
Switches (3,2,1): 111

Leds (3,2,1): 111

We can see the correct result when we give 111 as address, led lights gave the result as 111 because in address 111, we have 111 data.

CLOCK DIVIDER

We use a clock divider in order to decrease the input clock frequency. It takes an input that is clock source which is 50MHz in our FPGA. Clock divider divides that frequency to 250 millions to get requested output clock frequency. It uses a register counter that is initially zero for this purpose. When the process is completed, we obtain 0.1Hz (one clock cyle per 10 seconds) as output frequency. Finally, we integrated the clock divider to our CPU.



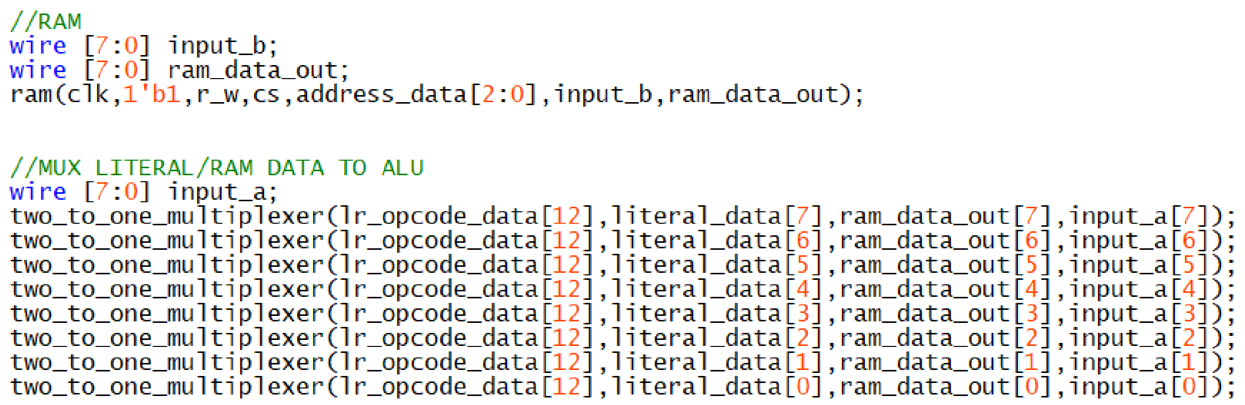
```
module clock_divider(
    input clk_in,
    output reg clk_out
);
    reg [27:0] counter = 28'd0;
    always @(posedge clk_in)
    begin
        if (counter >= 28'd250000000) begin
            counter <= 28'd0;
            clk_out <= ~clk_out;
        end
        else counter <= counter + 1;
    end
endmodule
```

Figure-1: Verilog code for Clock Divider

MUX AND DEMUX

Mux

Two to one multiplexer is used to choose ALU input a. Selection is between the data which comes from RAM (R-Type) and the data which comes directly from instruction register (Literal).



```
//RAM
wire [7:0] input_b;
wire [7:0] ram_data_out;
ram(clk,1'b1,r_w,cs,address_data[2:0],input_b,ram_data_out);

//MUX LITERAL/RAM DATA TO ALU
wire [7:0] input_a;
two_to_one_multiplexer(lr_opcode_data[12],literal_data[7],ram_data_out[7],input_a[7]);
two_to_one_multiplexer(lr_opcode_data[12],literal_data[6],ram_data_out[6],input_a[6]);
two_to_one_multiplexer(lr_opcode_data[12],literal_data[5],ram_data_out[5],input_a[5]);
two_to_one_multiplexer(lr_opcode_data[12],literal_data[4],ram_data_out[4],input_a[4]);
two_to_one_multiplexer(lr_opcode_data[12],literal_data[3],ram_data_out[3],input_a[3]);
two_to_one_multiplexer(lr_opcode_data[12],literal_data[2],ram_data_out[2],input_a[2]);
two_to_one_multiplexer(lr_opcode_data[12],literal_data[1],ram_data_out[1],input_a[1]);
two_to_one_multiplexer(lr_opcode_data[12],literal_data[0],ram_data_out[0],input_a[0]);
```

Figure-1: Multiplexers

Lr_opcode_data[12]: L/R bit.

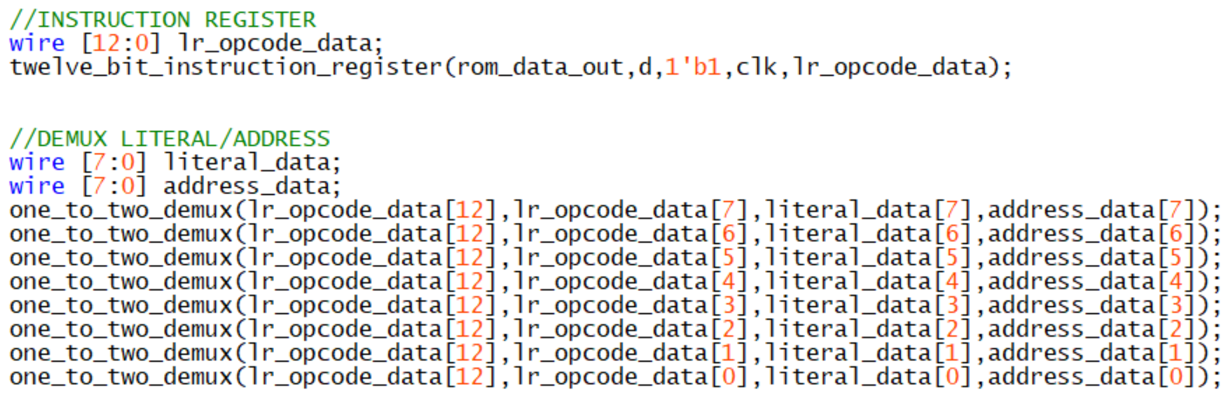
Literal_data: Data which comes directly from instruction register.

Ram_data_out: Data which comes from RAM.

Input_a: Data which will go to ALU input pins.

Demux

One to two demultiplexer is used to decide where data which comes from instruction register goes. The data can go to directly to multiplexer above or RAM to choose address.



```
//INSTRUCTION REGISTER
wire [12:0] lr_opcode_data;
twelve_bit_instruction_register(rom_data_out,d,1'b1,clk,lr_opcode_data);


//DEMUX LITERAL/ADDRESS
wire [7:0] literal_data;
wire [7:0] address_data;
one_to_two_demux(lr_opcode_data[12],lr_opcode_data[7],literal_data[7],address_data[7]);
one_to_two_demux(lr_opcode_data[12],lr_opcode_data[6],literal_data[6],address_data[6]);
one_to_two_demux(lr_opcode_data[12],lr_opcode_data[5],literal_data[5],address_data[5]);
one_to_two_demux(lr_opcode_data[12],lr_opcode_data[4],literal_data[4],address_data[4]);
one_to_two_demux(lr_opcode_data[12],lr_opcode_data[3],literal_data[3],address_data[3]);
one_to_two_demux(lr_opcode_data[12],lr_opcode_data[2],literal_data[2],address_data[2]);
one_to_two_demux(lr_opcode_data[12],lr_opcode_data[1],literal_data[1],address_data[1]);
one_to_two_demux(lr_opcode_data[12],lr_opcode_data[0],literal_data[0],address_data[0]);
```

Figure-1: Demultiplexers

Lr_opcode_data[12]: L/R bit.

Lr_opcode_data [7:0]: The data which comes from instruction register (address/literal).

Literal_data: Data which will go to multiplexer.

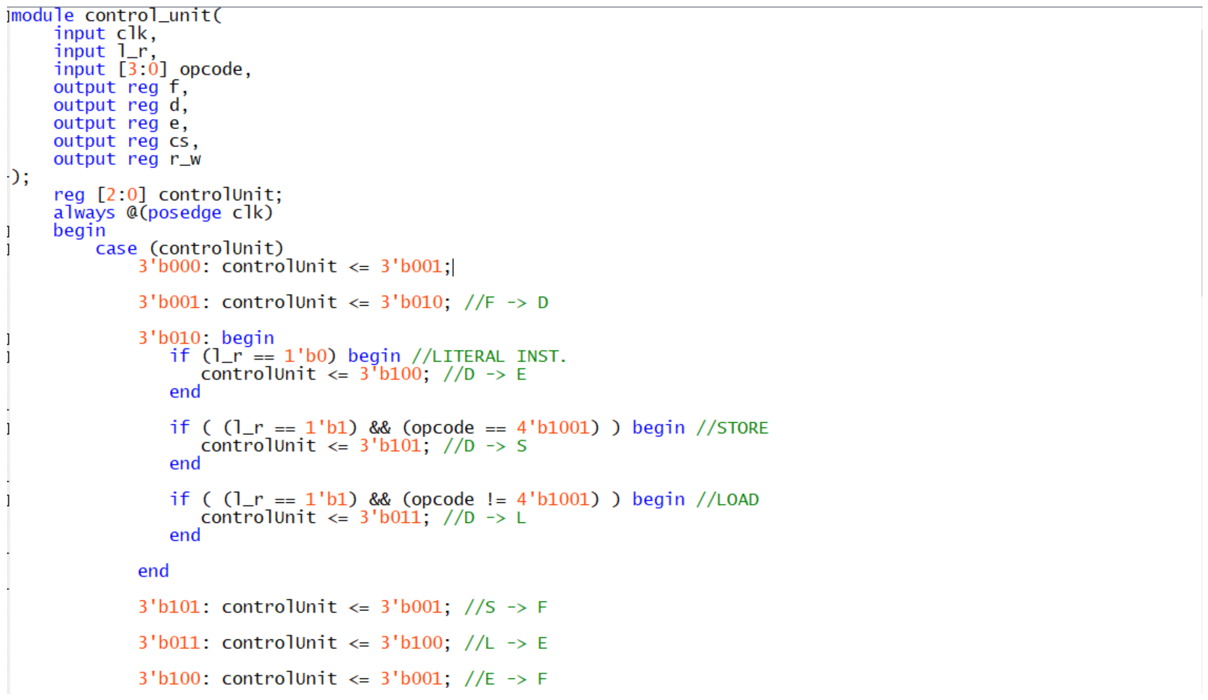
Address_data: Data which will go to RAM as an address.

UPDATED COMPONENTS

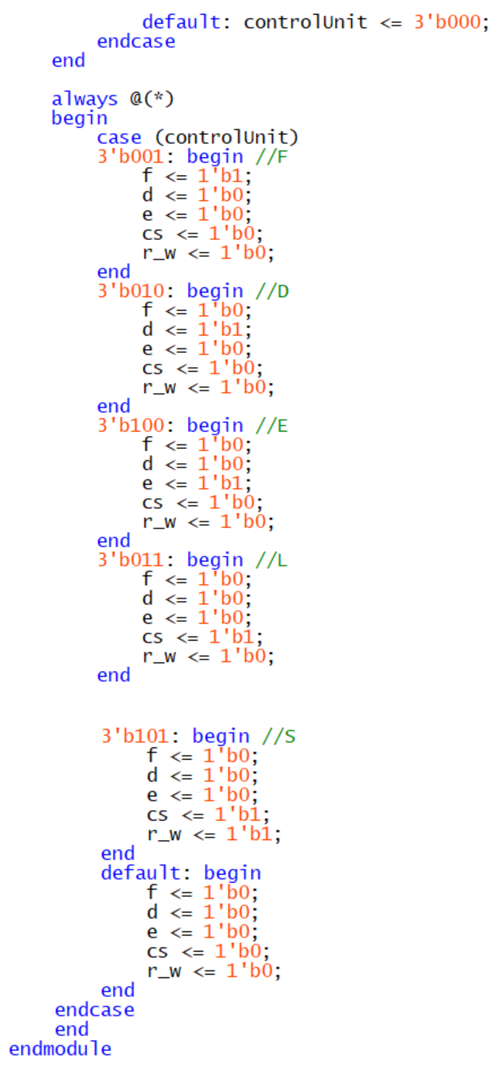
Control Unit

Control unit is changed according to R-Type instructions. We have 2 new states which are ‘LOAD’ and ‘STORE’. When instruction code is R-Type and opcode is 1001, control unit will go to STORE state from DECODE state. When instruction code is R-Type and opcode is not 1001, control unit will go to LOAD state from DECODE state and after that, it will go to EXECUTE state. In LOAD state R/W signal will be 0 (read signal) and ChipSelect signal will be 1. In STORE state R/W signal will be 1 (write signal) and ChipSelect signal will be 1.

LOAD state’s state code is 011, STORE state’s code is 101. The connection between states can be seen in Verilog code (Figure-1) below.



```
module control_unit(
    input clk,
    input l_r,
    input [3:0] opcode,
    output reg f,
    output reg d,
    output reg e,
    output reg cs,
    output reg r_w
);
    reg [2:0] controlUnit;
    always @(posedge clk)
    begin
        case (controlUnit)
            3'b000: controlUnit <= 3'b001;

            3'b001: controlUnit <= 3'b010; //F -> D

            3'b010: begin
               if (l_r == 1'b0) begin //LITERAL INST.
                  controlUnit <= 3'b100; //D -> E
               end

               if ( (l_r == 1'b1) && (opcode == 4'b1001) ) begin //STORE
                  controlUnit <= 3'b101; //D -> S
               end

               if ( (l_r == 1'b1) && (opcode != 4'b1001) ) begin //LOAD
                  controlUnit <= 3'b011; //D -> L
               end

            end

            3'b101: controlUnit <= 3'b001; //S -> F

            3'b011: controlUnit <= 3'b100; //L -> E

            3'b100: controlUnit <= 3'b001; //E -> F
```



```
            default: controlUnit <= 3'b000;
        endcase
    end

    always @(*)
    begin
        case (controlUnit)
        3'b001: begin //F
            f <= 1'b1;
            d <= 1'b0;
            e <= 1'b0;
            cs <= 1'b0;
            r_w <= 1'b0;
        end
        3'b010: begin //D
            f <= 1'b0;
            d <= 1'b1;
            e <= 1'b0;
            cs <= 1'b0;
            r_w <= 1'b0;
        end
        3'b100: begin //E
            f <= 1'b0;
            d <= 1'b0;
            e <= 1'b1;
            cs <= 1'b0;
            r_w <= 1'b0;
        end
        3'b011: begin //L
            f <= 1'b0;
            d <= 1'b0;
            e <= 1'b0;
            cs <= 1'b1;
            r_w <= 1'b0;
        end


        3'b101: begin //S
            f <= 1'b0;
            d <= 1'b0;
            e <= 1'b0;
            cs <= 1'b1;
            r_w <= 1'b1;
        end
        default: begin
            f <= 1'b0;
            d <= 1'b0;
            e <= 1'b0;
            cs <= 1'b0;
            r_w <= 1'b0;
        end
    endcase
    end
endmodule
```

Figure-1: Verilog Code for Control Unit

Test for Control Unit in simulation:

First case (Figure-2): Literal instruction selection (L/R signal = 0)

Second case (Figure-3): R-Type Load instruction (L/R signal = 1 and opcode != 1001)

Third case (Figure-4): R-Type Store instruction (L/R signal = 1 and opcode = 1001)

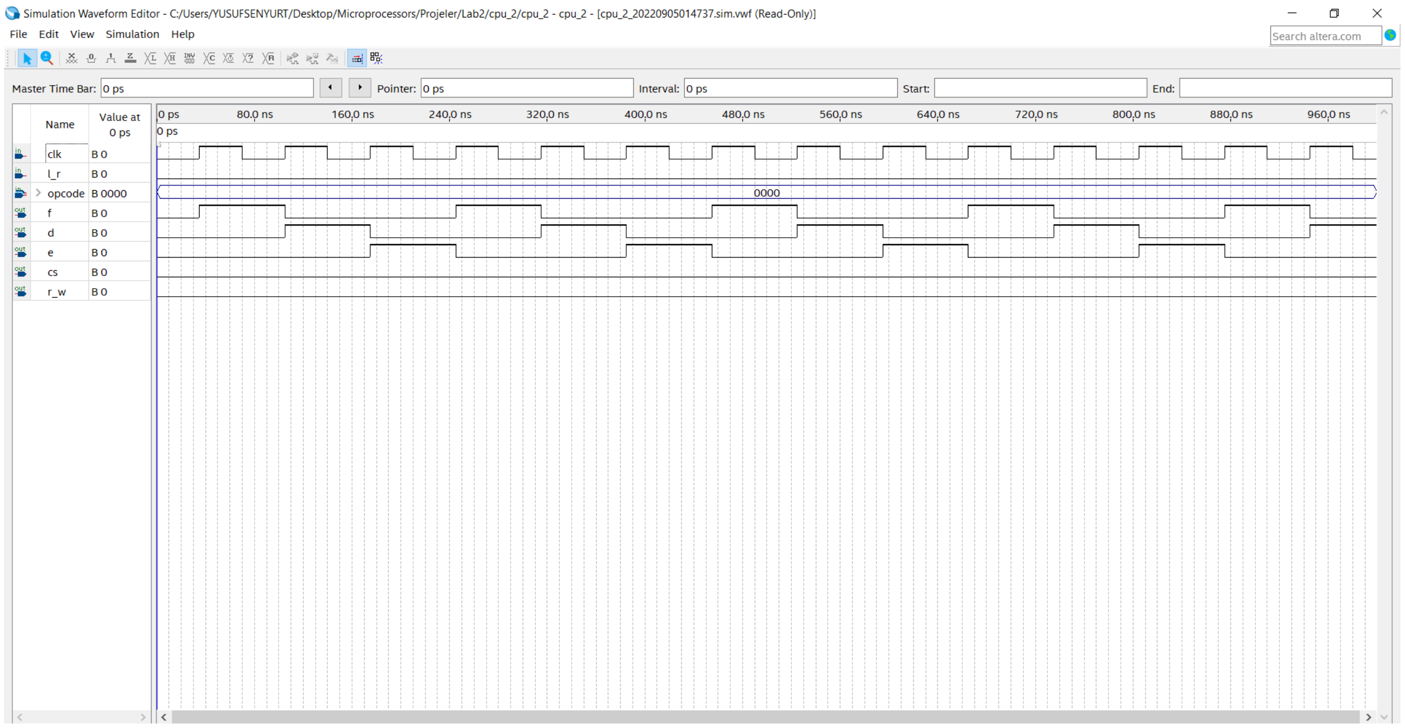


Figure-2: Literal Instruction

In literal instruction case (Figure-2), cs and r_w signals are both set to zero. Program is working according to the former form.

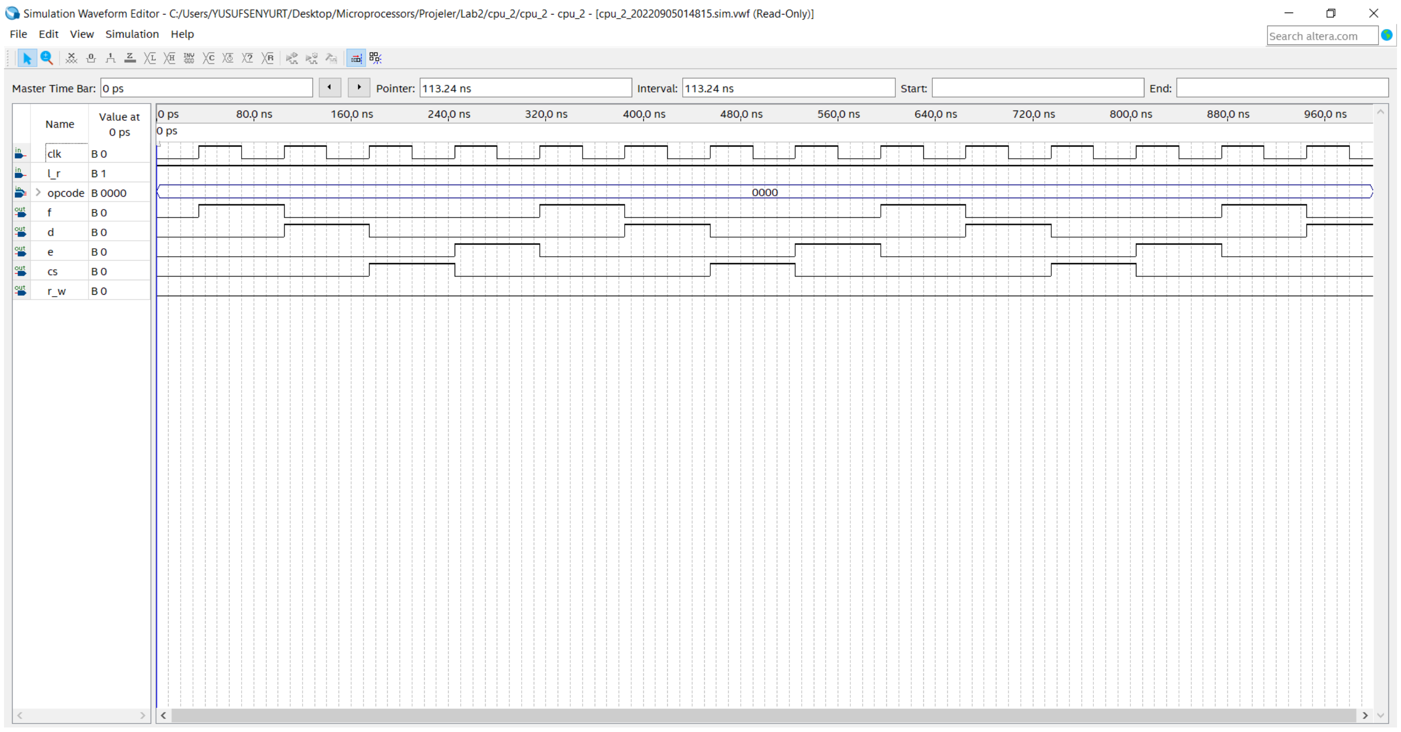

Simulation Waveform Editor - C:/Users/YUSUFSENYURT/Desktop/Microprocessors/Projeler/Lab2/cpu_2/cpu_2 - cpu_2 - [cpu_2_20220905014815.sim.vwf (Read-Only)]
File Edit View Simulation Help
Master Time Bar: 0 ps
Pointer: 113.24 ns
Interval: 113.24 ns
Start:
End:
Name
Value at 0 ps
clk B 0
l_r B 1
opcode B 0000
f B 0
d B 0
e B 0
cs B 0
r_w B 0
0000

Figure-3: R-Type Load Instruction

In R-Type Load instruction case (Figure-3), cs signal is set to 1 to choose RAM and readWrite signal is set to 0 (read signal).

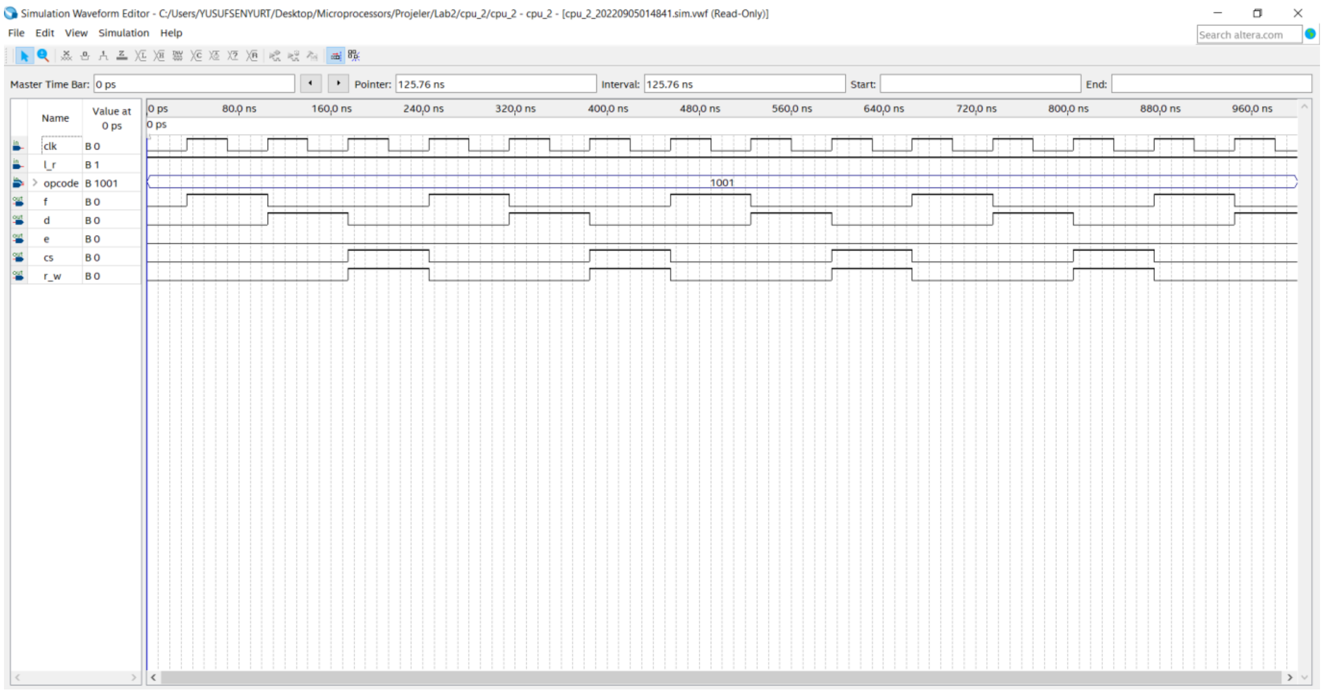

Simulation Waveform Editor - C:/Users/YUSUFSENYURT/Desktop/Microprocessors/Projeler/Lab2/cpu_2/cpu_2 - cpu_2 - [cpu_2_20220905014841.sim.vwf (Read-Only)]
File Edit View Simulation Help
Master Time Bar: 0 ps
Pointer: 125.76 ns
Interval: 125.76 ns
Start:
End:
Name
Value at 0 ps
clk B 0
l_r B 1
opcode B 1001
f B 0
d B 0
e B 0
cs B 0
r_w B 0
1001

Figure-4: R-Type Store Instruction

In R-Type Store instruction case (Figure-4), cs signal is set to 1 to choose RAM and readWrite signal is set to 1 (write signal).

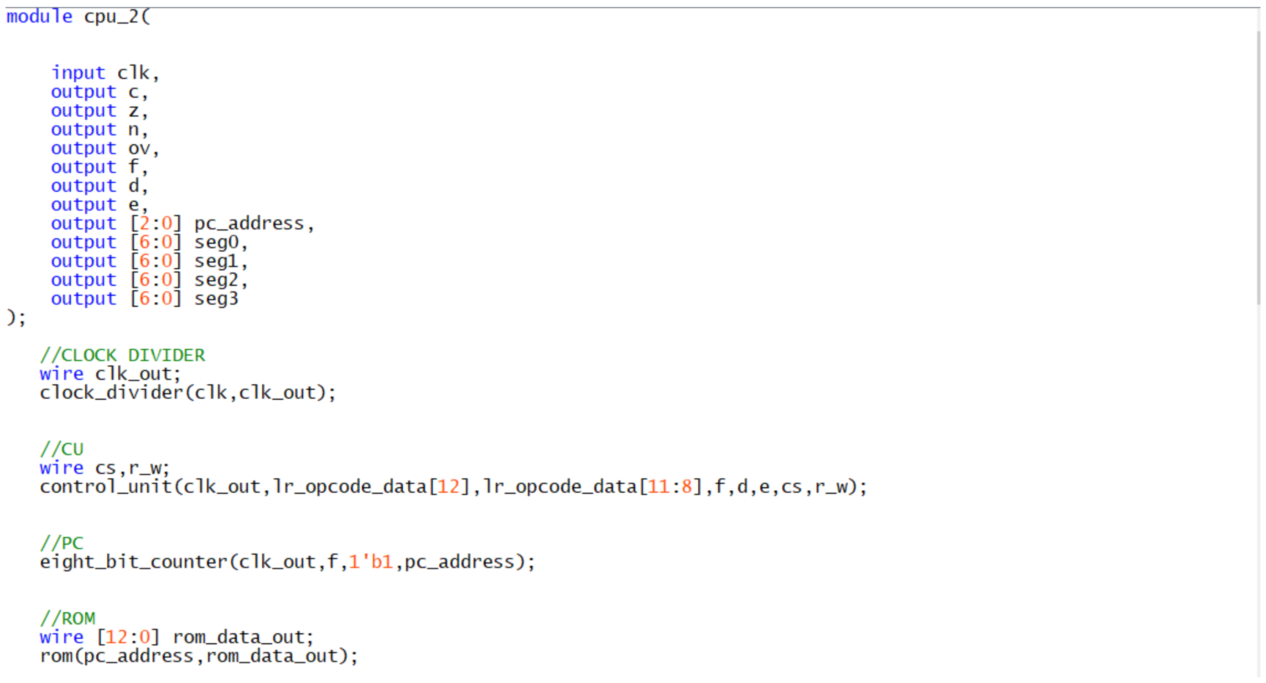
Instruction Register

Instruction register was 12-bit, but it has to be 13-bit now because literal/address selection bit is added to instructions. First bit of 13-bit instruction will be literal/address selection bit from now on.

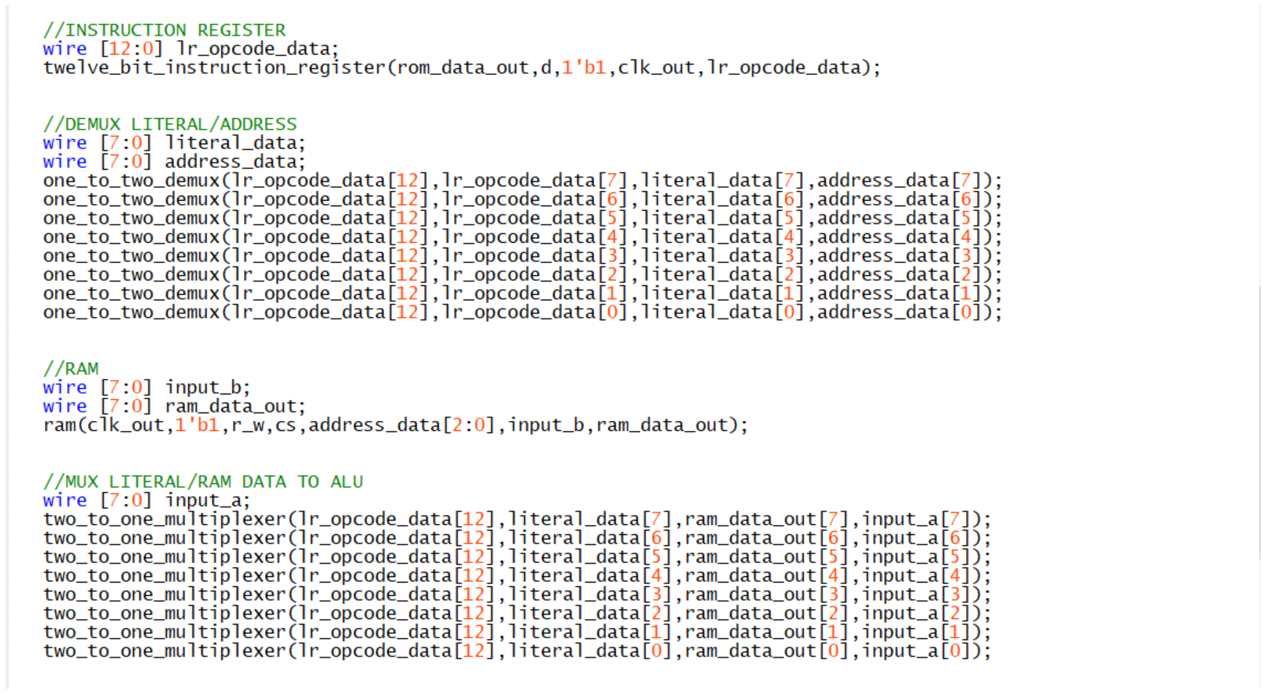
Counter

Counter will count to eight at this point because 8 instructions are stored in ROM in new circuit.

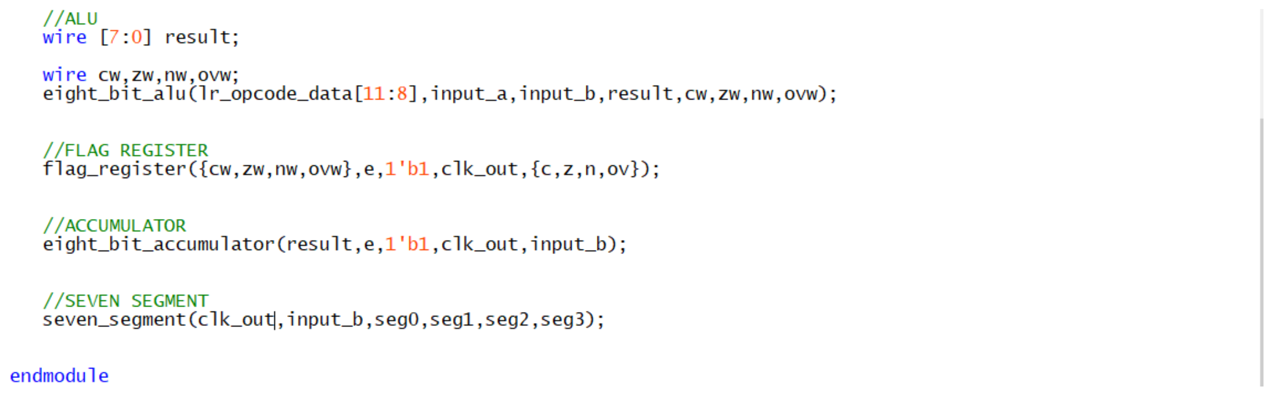
MAIN CODE



```
module cpu_2(


     input clk,
     output c,
     output z,
     output n,
     output ov,
     output f,
     output d,
     output e,
     output [2:0] pc_address,
     output [6:0] seg0,
     output [6:0] seg1,
     output [6:0] seg2,
     output [6:0] seg3
);

    //CLOCK DIVIDER
    wire clk_out;
    clock_divider(clk,clk_out);


    //CU
    wire cs,r_w;
    control_unit(clk_out,lr_opcode_data[12],lr_opcode_data[11:8],f,d,e,cs,r_w);


    //PC
    eight_bit_counter(clk_out,f,1'b1,pc_address);


    //ROM
    wire [12:0] rom_data_out;
    rom(pc_address,rom_data_out);
```



```
    //INSTRUCTION REGISTER
    wire [12:0] lr_opcode_data;
    twelve_bit_instruction_register(rom_data_out,d,1'b1,clk_out,lr_opcode_data);


    //DEMUX LITERAL/ADDRESS
    wire [7:0] literal_data;
    wire [7:0] address_data;
    one_to_two_demux(lr_opcode_data[12],lr_opcode_data[7],literal_data[7],address_data[7]);
    one_to_two_demux(lr_opcode_data[12],lr_opcode_data[6],literal_data[6],address_data[6]);
    one_to_two_demux(lr_opcode_data[12],lr_opcode_data[5],literal_data[5],address_data[5]);
    one_to_two_demux(lr_opcode_data[12],lr_opcode_data[4],literal_data[4],address_data[4]);
    one_to_two_demux(lr_opcode_data[12],lr_opcode_data[3],literal_data[3],address_data[3]);
    one_to_two_demux(lr_opcode_data[12],lr_opcode_data[2],literal_data[2],address_data[2]);
    one_to_two_demux(lr_opcode_data[12],lr_opcode_data[1],literal_data[1],address_data[1]);
    one_to_two_demux(lr_opcode_data[12],lr_opcode_data[0],literal_data[0],address_data[0]);


    //RAM
    wire [7:0] input_b;
    wire [7:0] ram_data_out;
    ram(clk_out,1'b1,r_w,cs,address_data[2:0],input_b,ram_data_out);


    //MUX LITERAL/RAM DATA TO ALU
    wire [7:0] input_a;
    two_to_one_multiplexer(lr_opcode_data[12],literal_data[7],ram_data_out[7],input_a[7]);
    two_to_one_multiplexer(lr_opcode_data[12],literal_data[6],ram_data_out[6],input_a[6]);
    two_to_one_multiplexer(lr_opcode_data[12],literal_data[5],ram_data_out[5],input_a[5]);
    two_to_one_multiplexer(lr_opcode_data[12],literal_data[4],ram_data_out[4],input_a[4]);
    two_to_one_multiplexer(lr_opcode_data[12],literal_data[3],ram_data_out[3],input_a[3]);
    two_to_one_multiplexer(lr_opcode_data[12],literal_data[2],ram_data_out[2],input_a[2]);
    two_to_one_multiplexer(lr_opcode_data[12],literal_data[1],ram_data_out[1],input_a[1]);
    two_to_one_multiplexer(lr_opcode_data[12],literal_data[0],ram_data_out[0],input_a[0]);
```



```
    //ALU
    wire [7:0] result;

    wire cw,zw,nw,ovw;
    eight_bit_alu(lr_opcode_data[11:8],input_a,input_b,result,cw,zw,nw,ovw);


    //FLAG REGISTER
    flag_register({cw,zw,nw,ovw},e,1'b1,clk_out,{c,z,n,ov});


    //ACCUMULATOR
    eight_bit_accumulator(result,e,1'b1,clk_out,input_b);


    //SEVEN SEGMENT
    seven_segment(clk_out,input_b,seg0,seg1,seg2,seg3);

endmodule
```

Figure-1: Main Code of CPU

First of all, every component which needs clock has new clock comes from clock divider. Control unit has output signals that are chip select (cs) and read write (r_w). These signals goes to RAM to choose operation between read and write for RAM. Rom takes pc_address as input from PC counter and gives rom_data_out as output to instruction register.

Instruction register has 13-bit instruction codes which are L/R selection bit, operation code and data/address. lr_opcode_data wire is used for these 13-bit instruction as output of instruction register.

After these operations, one to two demultiplexer decides where data which comes from instruction register goes. The data can go to directly to multiplexer or RAM to choose address. RAM has input_b which comes from b input of ALU (result of accumulator also). This input will be stored if the instruction code includes STORE operation code. RAM gives ram_data_out as output when operation is read (LOAD).

Two to one multiplexer will select the data for input a of ALU. Two inputs of multiplexer come from RAM and LITERAL side of demultiplexer. After that ALU will do its operations, flag registers, accumulator and seven segment will be updated according to results.

Unfortunately, RAM operations of our CPU are not working. Everything beside R-Type operations is working correctly. L-Type instructions, Control Unit (states are working), Clock Divider and ROM are correct.

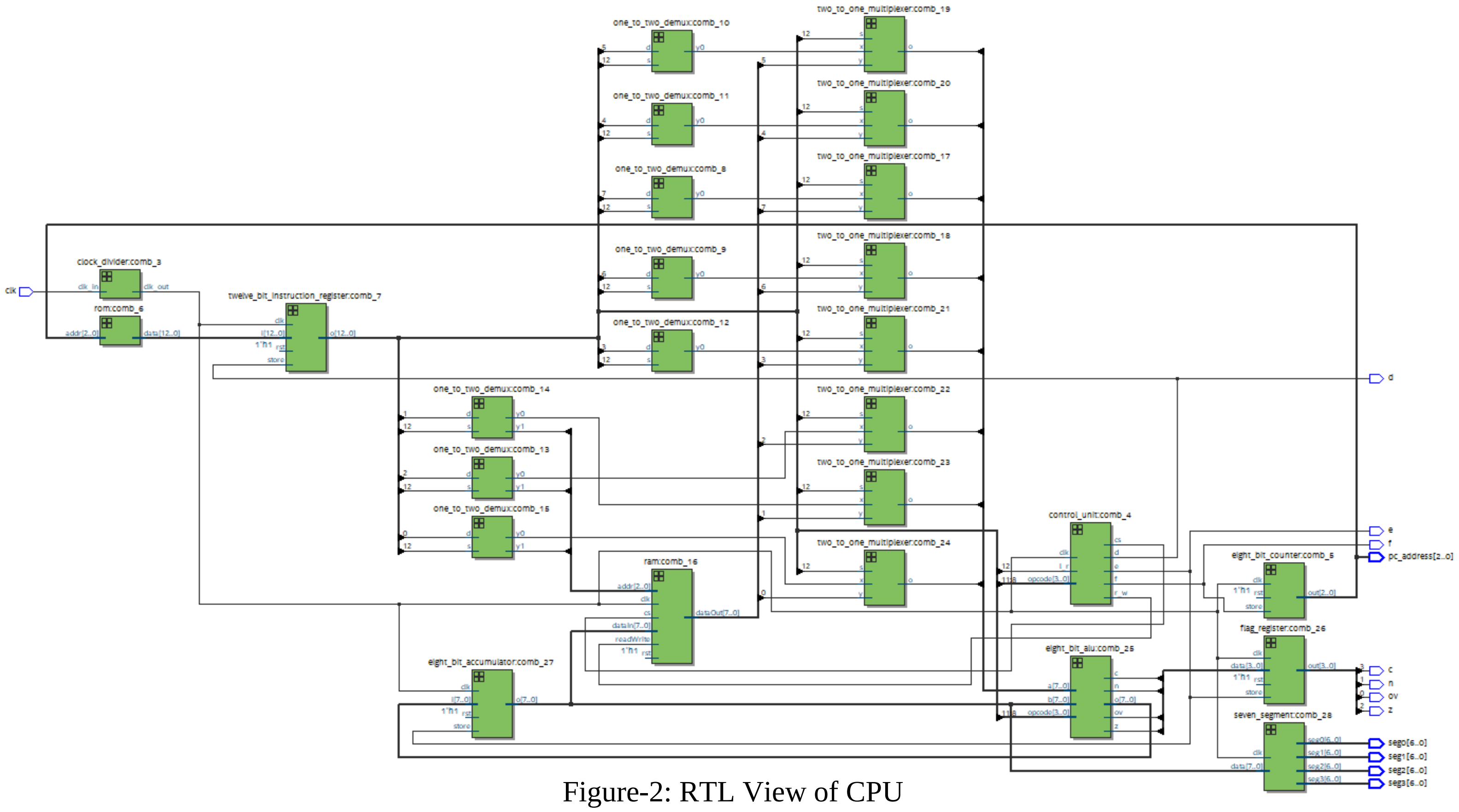


Figure-2: RTL View of CPU