ECE 451-LAB4

Fall 2019

Ram Rohit Gannavarapu

Contents

List of Figures

1. List of items completed

a) Design of 1:2 Decoder
b) Design of 3:8 Decoder using 1:2 decoder blocks
c) Design of ROM block
d) Display frames on 8x8 LED matrix display using ROM module

2. Lab Procedures

ASCII value of the character to be store in the rom modules were retrieved in binary format as shown in the below table.

Letter	ASCII Value in Binary
R	01010010
A	01000001
M	01001101
R	01010010
O	01001111
H	01001000
I	01001001
T	01010100

The lab design starts with designing of a 1:2 decoder module. Then followed by creatinbg a 3:8 decoder block using the 1:2 decoder blocks. ROM module was created which stores the characters showed in the above table in each word line. The ROM module has 8 word lines and 8 output bits. The ROM module is then connected to the 3:8 decoder. Based on the word line selected by the decoder the respected character(binary value of ASCII) is outputted on the bit lines.

3. Schematics/Verilog models

a. Cadence:

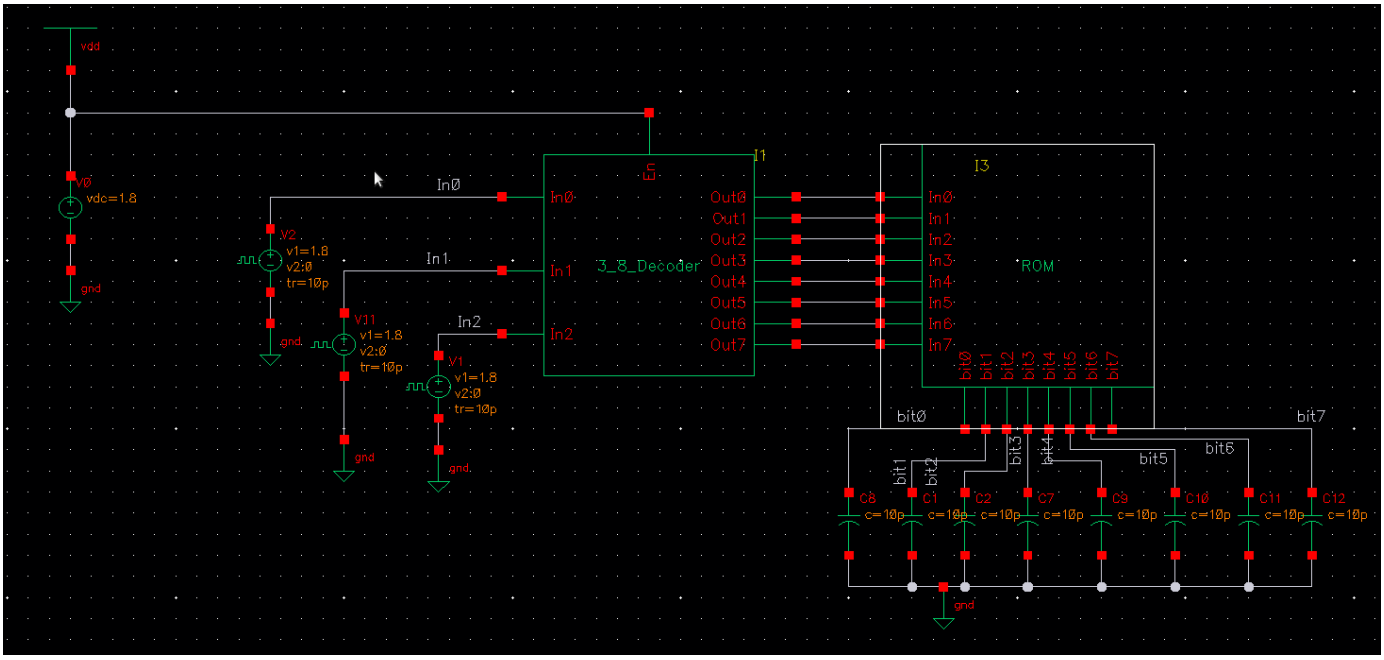

Figure 1: Test Bench

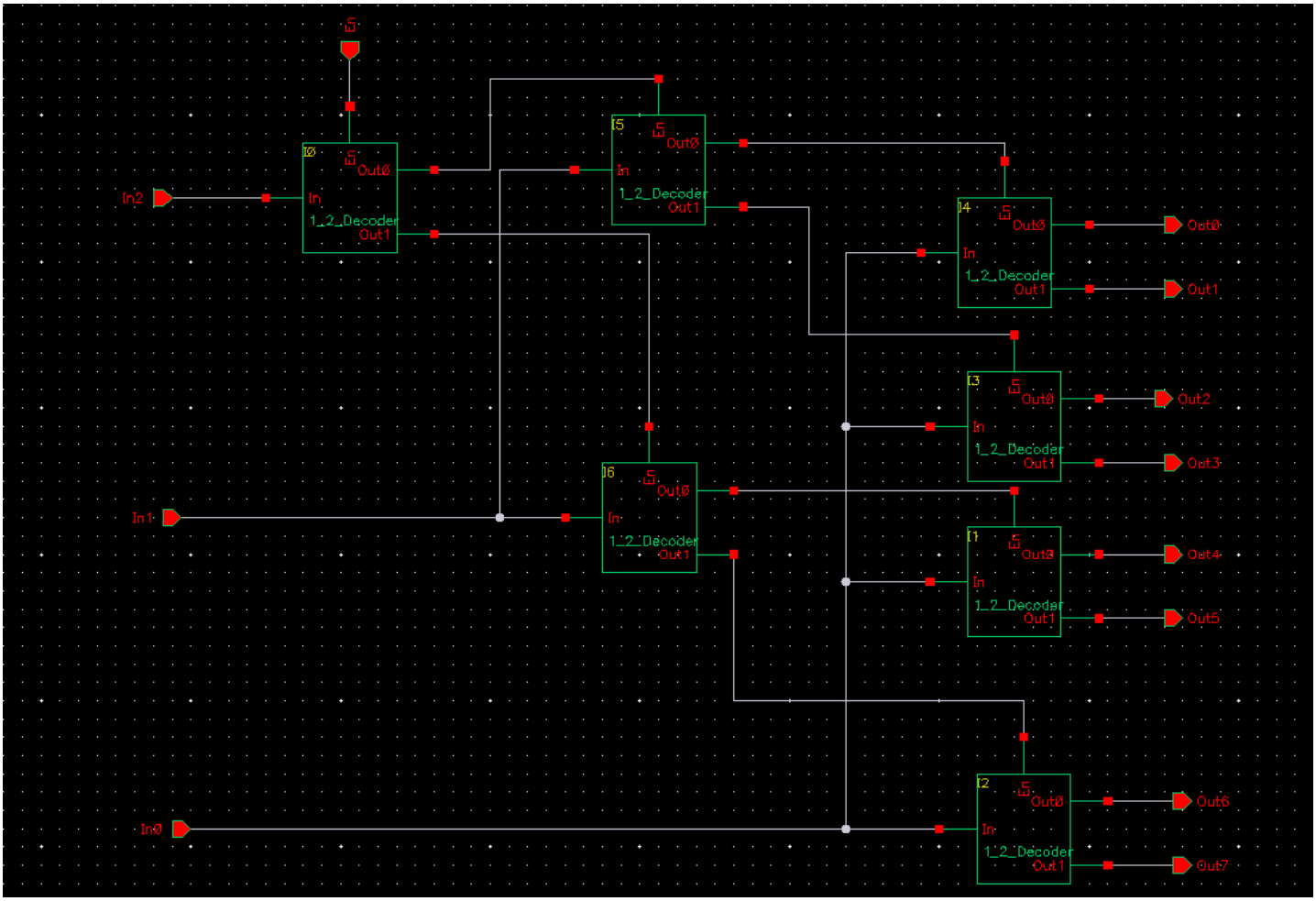

Figure 2: 3:8 Decoder using 1:2 Decoder

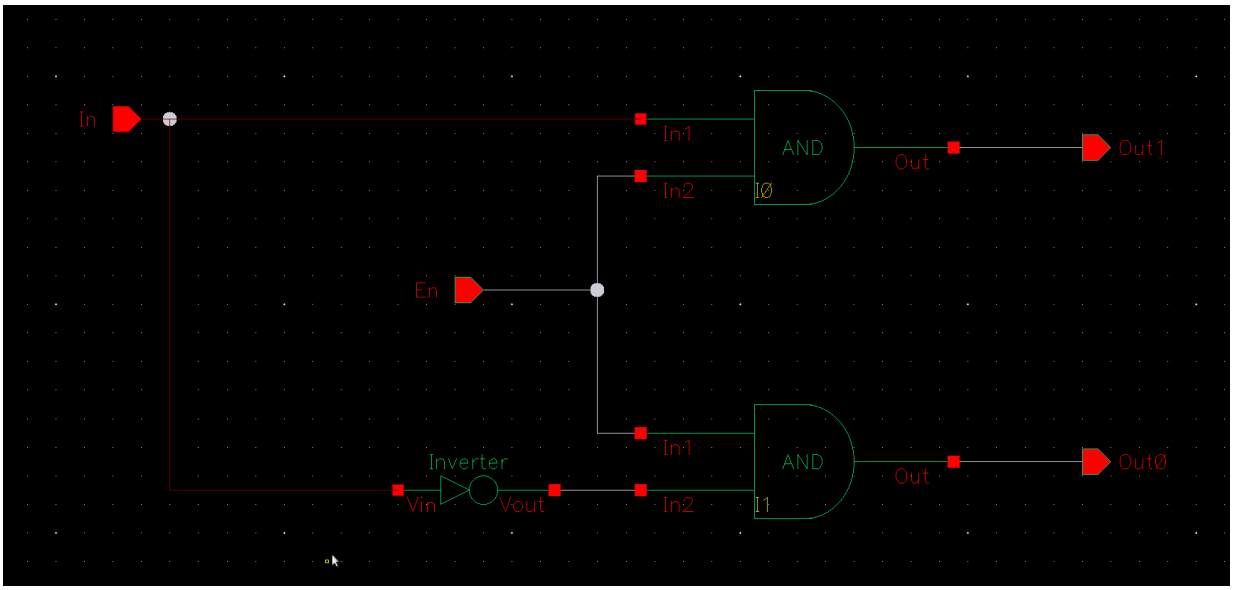

Figure 3: 1:2 Decoder Schematic

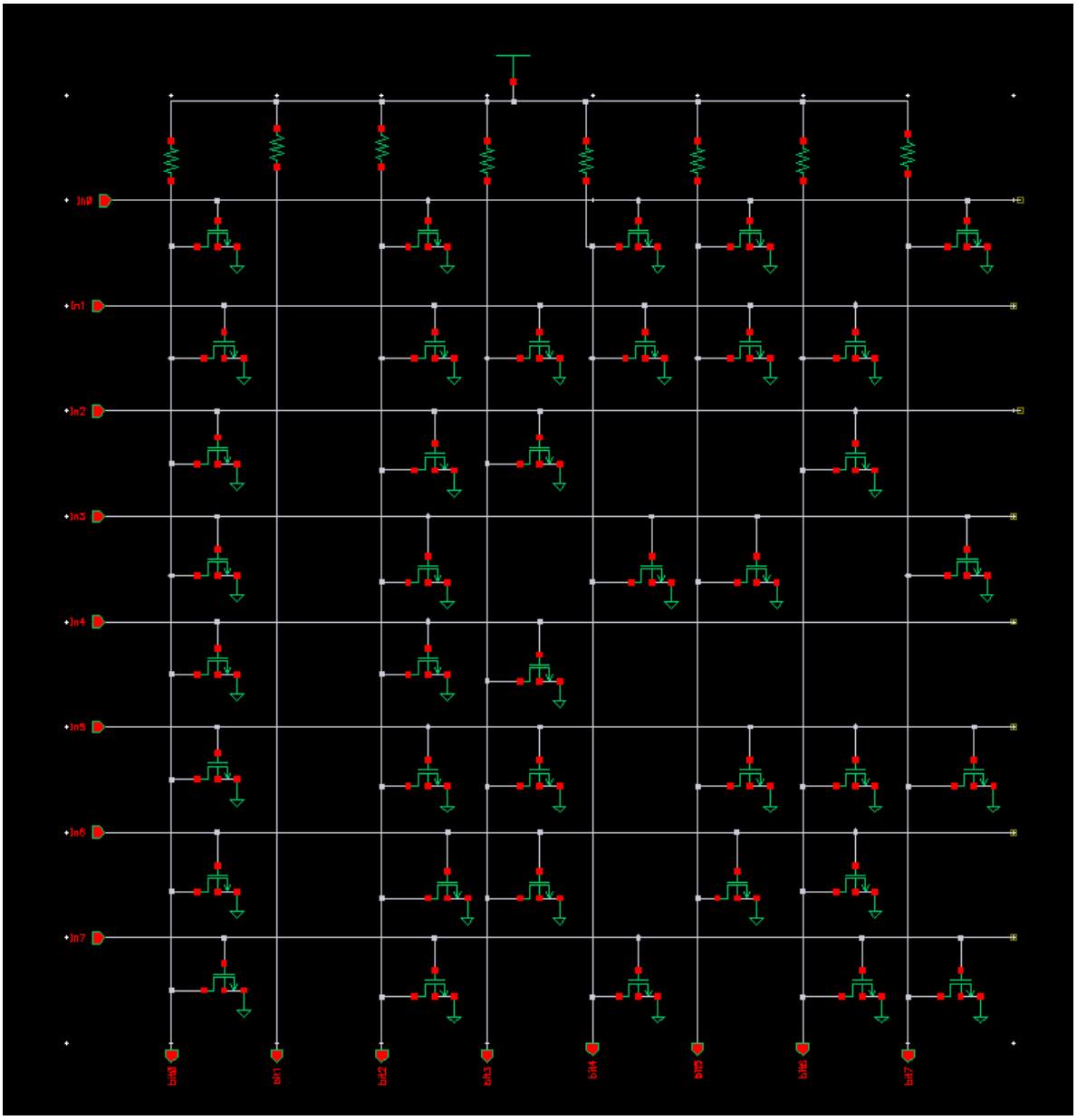

Figure 4: ROM Module for letters R,A,M,R,O,H,I,T (in the same order Top to bottom)

b. Verilog:

Verilog Code file has been attached

Matrix_display.v

4. Simulation Results

Cadence:

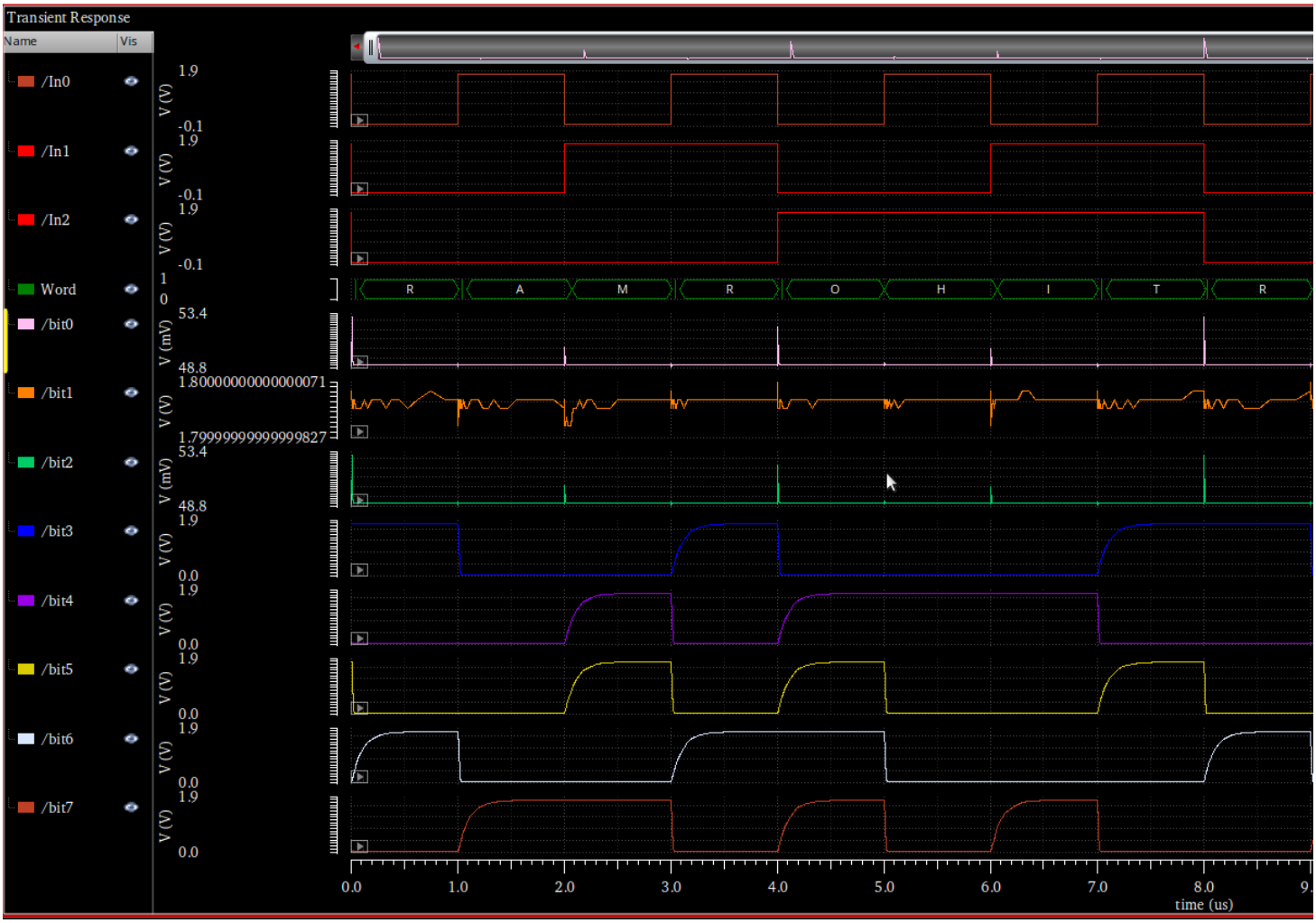

Figure 5: Simulation Result of ROM Module

5. Analysis and Explanations

a. Cadence:

The ROM Module outputs correct values on its outputs based on the Inputs from the decoder module. The Simulation results match with the design and can be verified on the Simulation output were the digital outputs are clubbed and created as a bus and converted to show the output as an ASCII character.

b. Verilog:

display_rom module was created which stores the frames. It has 2 inputs, row number, and frame number and column data as output. Based on the row number the respective column data is sent as the output.

Addressing individual LED is difficult hence in this experiment we try to output each row at a time. And these rows as running at high speeds that we perceive them as a single image. This phenomenon is known as blinkenlights. In the code row_clock has been used to fastly select rows and output the respective data. row_clock also shouldn't be too fast say (50MHz clock of the FPGA) as it would be too fast and the capacitor in the LED display don't get charged at such high speeds and hence we have reduced the speed to 500Hz.

frame_clock was used to toggle between different frames. This clock is comparatively slower and runs at 6.25Hz (0.16 s).

Run input is used to enable change between frames and reset input is used to reset the current frame to the first frame.a

6. Conclusion

a. Cadence:

In this LAB we have successfully designed a 3 to 8 decoder and also a ROM module.

b. Verilog:

In this Lab we were able to successfully design a ROM module and also display the contents onto a 8x8 LED matrix display. We also got a knowledge on how digital displays work at the lowest level through the concept of blinkenlights.

7. Answers to questions posted in the lab

Q) Discuss the concept and advantage of hierarchical design

Hierarchical design is a process in which a Design is divided into small identical blocks and these blocks from the building blocks for the whole design. By this we would be able to achieve faster development times and also debugging issues would be easier and faster as the blocks are reused. Performance of the design also improves as the complexity of the whole design reduces.

Q) How was hierarchical design utilized in this lab?

Hierarchical design was used in the cadence part of the lab where we have created 1:2 decoder and utilized those 1:2 decoder blocks to create a 3:8 decoder.

Hierarchical design was also used partially in the Verilog part of this lab where we have used the clock_divider module to divide the clock for row_clock and frame_clock. We didn't have to rewrite the whole logic instead just created an instance of the module.

Verilog Code:

```
1.              module Matrix_display(clock50MHz, run, reset, row, column, frame);
2.          input run;
3.          input reset;
4.          input clock50MHz;
5.          output reg [7:0] row,column;
6.
7.           wire  row_clock;
8.           wire frame_clock;
9.
10.         output reg [3:0] frame;
11.         wire [7:0] temp;
12.
13.          reg [2:0] count1;
14.
15.         clock_divider #(50000) row_clock_gen(clock50MHz, row_clock);
16.         clock_divider #(4000000) frame_clock_gen(clock50MHz, frame_clock);
17.
18.         display_rom c0(frame,count1, temp);
19.
20.
21.         always @(posedge frame_clock) begin
22.                 if( run == 1)
23.                 begin
24.
25.                 if(frame == 4'b1111 || reset==0)
26.                 begin
27.                     frame  = 4'b0000;
28.                 end
29.                 else
30.                 frame = frame + 4'b0001;
31.             end
32.
33.         end
34.
35.         always @(posedge row_clock)
36.
37.         begin
38.             if(count1 == 3'b111)
39.                 count1  = 3'b000;
40.             else
41.                 count1 = count1 + 3'b001;
42.
43.         end
44.
45.
46.         always @(count1,temp)
47.         begin
48.             case (count1)
49.                 3'b000: begin row = 8'b11111110;column = temp;  end
50.                 3'b001: begin row = 8'b11111101;column = temp;  end
51.                 3'b010: begin row = 8'b11111011;column = temp;  end
52.                 3'b011: begin row = 8'b11110111;column = temp;  end
53.                 3'b100: begin row = 8'b11101111;column = temp;  end
54.                 3'b101: begin row = 8'b11011111;column = temp;  end
55.                 3'b110: begin row = 8'b10111111;column = temp;  end
56.                 3'b111: begin row = 8'b01111111;column = temp;  end
57.             endcase
58.         end
```

```
    endmodule


module display_rom(frame, address, data_out);
    input [2:0] address;
    output [7:0] data_out;
    reg [7:0] data_out;
    input [3:0] frame;

    always @ (address,frame)
    begin
    if(frame == 4'b0000)
        begin
        case (address)

            3'b000: data_out = 8'b11111111;
            3'b001: data_out = 8'b10000000;
            3'b010: data_out = 8'b10000000;
            3'b011: data_out = 8'b10000000;
            3'b100: data_out = 8'b10000000;
            3'b101: data_out = 8'b10000000;
            3'b110: data_out = 8'b10000000;
            3'b111: data_out = 8'b11111111;
        endcase
    end
    else if(frame == 4'b0001)
        begin
            case(address)

                3'b000: data_out = 8'b11111100;
                3'b001: data_out = 8'b00000000;
                3'b010: data_out = 8'b00000000;
                3'b011: data_out = 8'b00000000;
                3'b100: data_out = 8'b00000000;
                3'b101: data_out = 8'b00000000;
                3'b110: data_out = 8'b00000000;
                3'b111: data_out = 8'b11111100;
            endcase
        end

                else if(frame == 4'b0010)
                begin
                    case(address)

                        3'b000: data_out = 8'b11110011;
                        3'b001: data_out = 8'b00000010;
                        3'b010: data_out = 8'b00000010;
                        3'b011: data_out = 8'b00000011;
                        3'b100: data_out = 8'b00000000;
                        3'b101: data_out = 8'b00000000;
                        3'b110: data_out = 8'b00000000;
                        3'b111: data_out = 8'b11110011;
                    endcase
                end

                else if(frame == 4'b0011)
                begin
                    case(address)

                        3'b000: data_out = 8'b11001111;
```

```
                        3'b001: data_out = 8'b00001000;
                        3'b010: data_out = 8'b00001000;
                        3'b011: data_out = 8'b00001111;
                        3'b100: data_out = 8'b00000000;
                        3'b101: data_out = 8'b00000000;
                        3'b110: data_out = 8'b00000000;
                        3'b111: data_out = 8'b11001111;
                    endcase
                end
                        else if(frame == 4'b0100)
                begin
                    case(address)


                        3'b000: data_out = 8'b00111111;
                        3'b001: data_out = 8'b00100000;
                        3'b010: data_out = 8'b00100000;
                        3'b011: data_out = 8'b00111111;
                        3'b100: data_out = 8'b00000000;
                        3'b101: data_out = 8'b00000000;
                        3'b110: data_out = 8'b00000000;
                        3'b111: data_out = 8'b00111111;
                    endcase
                end
                else if(frame == 4'b0101)
                begin
                    case(address)

                        3'b000: data_out = 8'b11111111;
                        3'b001: data_out = 8'b10000000;
                        3'b010: data_out = 8'b10000000;
                        3'b011: data_out = 8'b11111111;
                        3'b100: data_out = 8'b00000001;
                        3'b101: data_out = 8'b00000001;
                        3'b110: data_out = 8'b00000001;
                        3'b111: data_out = 8'b11111111;
                    endcase
                end
                else if(frame == 4'b0110)
                begin
                    case(address)

                        3'b000: data_out = 8'b11111100;
                        3'b001: data_out = 8'b00000000;
                        3'b010: data_out = 8'b00000000;
                        3'b011: data_out = 8'b11111100;
                        3'b100: data_out = 8'b00000100;
                        3'b101: data_out = 8'b00000100;
                        3'b110: data_out = 8'b00000100;
                        3'b111: data_out = 8'b11111100;
                    endcase
                end
                else if(frame == 4'b0111)
                begin
                    case(address)


                        3'b000: data_out = 8'b11110010;
                        3'b001: data_out = 8'b00000010;
                        3'b010: data_out = 8'b00000010;
                        3'b011: data_out = 8'b11110010;
```

```
                    3'b100: data_out = 8'b00010010;
                    3'b101: data_out = 8'b00010010;
                    3'b110: data_out = 8'b00010010;
                    3'b111: data_out = 8'b11110011;
                endcase
            end
            else if(frame == 4'b1000)
            begin
                case(address)

                    3'b000: data_out = 8'b11001000;
                    3'b001: data_out = 8'b00001000;
                    3'b010: data_out = 8'b00001000;
                    3'b011: data_out = 8'b11001000;
                    3'b100: data_out = 8'b01001000;
                    3'b101: data_out = 8'b01001000;
                    3'b110: data_out = 8'b01001000;
                    3'b111: data_out = 8'b11001111;
                endcase
            end

            else if(frame == 4'b1001)
            begin
                case(address)

                    3'b000: data_out = 8'b00100000;
                    3'b001: data_out = 8'b00100000;
                    3'b010: data_out = 8'b00100000;
                    3'b011: data_out = 8'b00100000;
                    3'b100: data_out = 8'b00100000;
                    3'b101: data_out = 8'b00100000;
                    3'b110: data_out = 8'b00100000;
                    3'b111: data_out = 8'b00111111;
                endcase
            end

            else if(frame == 4'b1010)
            begin
                case(address)

                    3'b000: data_out = 8'b10000001;
                    3'b001: data_out = 8'b10000001;
                    3'b010: data_out = 8'b10000001;
                    3'b011: data_out = 8'b10000001;
                    3'b100: data_out = 8'b10000001;
                    3'b101: data_out = 8'b10000001;
                    3'b110: data_out = 8'b10000001;
                    3'b111: data_out = 8'b11111111;
                endcase
            end

            else if(frame == 4'b1011)
            begin
                case(address)

                    3'b000: data_out = 8'b00000100;
                    3'b001: data_out = 8'b00000100;
                    3'b010: data_out = 8'b00000100;
                    3'b011: data_out = 8'b00000100;
                    3'b100: data_out = 8'b00000100;
                    3'b101: data_out = 8'b00000100;
```

```
                        3'b110: data_out = 8'b00000100;
                        3'b111: data_out = 8'b11111100;
                    endcase
                end
                else if(frame == 4'b1100)
                begin
                    case(address)

                        3'b000: data_out = 8'b00010011;
                        3'b001: data_out = 8'b00010010;
                        3'b010: data_out = 8'b00010010;
                        3'b011: data_out = 8'b00010010;
                        3'b100: data_out = 8'b00010010;
                        3'b101: data_out = 8'b00010010;
                        3'b110: data_out = 8'b00010010;
                        3'b111: data_out = 8'b11110011;
                    endcase
                end
                else if(frame == 4'b1101)
                begin
                    case(address)

                        3'b000: data_out = 8'b01001111;
                        3'b001: data_out = 8'b01001000;
                        3'b010: data_out = 8'b01001000;
                        3'b011: data_out = 8'b01001000;
                        3'b100: data_out = 8'b01001000;
                        3'b101: data_out = 8'b01001000;
                        3'b110: data_out = 8'b01001000;
                        3'b111: data_out = 8'b11001111;
                    endcase
                end
                else if(frame == 4'b1110)
                begin
                    case(address)

                        3'b000: data_out = 8'b00111111;
                        3'b001: data_out = 8'b00100000;
                        3'b010: data_out = 8'b00100000;
                        3'b011: data_out = 8'b00100000;
                        3'b100: data_out = 8'b00100000;
                        3'b101: data_out = 8'b00100000;
                        3'b110: data_out = 8'b00100000;
                        3'b111: data_out = 8'b00111111;
                    endcase
                end
                else if(frame == 4'b1111)
                begin
                    case(address)

                        3'b000: data_out = 8'b11111111;
                        3'b001: data_out = 8'b10000000;
                        3'b010: data_out = 8'b10000000;
                        3'b011: data_out = 8'b10000000;
                        3'b100: data_out = 8'b10000000;
                        3'b101: data_out = 8'b10000000;
                        3'b110: data_out = 8'b10000000;
                        3'b111: data_out = 8'b11111111;
                    endcase
                end

```

```
303.            end
304.        endmodule
305.
306.
307.        module clock_divider(
308.            input input_clock,
309.            output reg output_clock
310.            );
311.
312.            parameter limit;    // used as the clock divider
313.            reg [25:0] count = 0;
314.
315.            always@(posedge input_clock)begin
316.                count = count + 1;
317.                if(count == limit) begin
318.                        count <= 0;
319.                        output_clock <= ~output_clock;
320.                end
321.            end
322.        endmodule
```