ECE 451 LAB 4

Blinkenlights

Ishani Gowaikar
CSU ID- 831702439

ECE 451 LAB 4 Blinkenlights.

Objective:

- Use hierarchical design to create complex circuits
- Understand the design and function of a ROM and Decoder
- blinkenlights!

Use a 7xN ROM to write an English word at least 5 ASCII characters long to the output.

LAB Procedures:

Cadence:

- Create a 7-bit ROM. Each word line is a single printable ASCII character (see http://www.asciitable.com).
- Create a 1:2 decoder.
- Use hierarchical design starting with the 1:2 decoder to create a nog$_2$ Nl −1:*N* decoder, able to address each line your ROM.
- Connect the decoder and ROM.
- Simulate the circuit to verify the message is correctly generated at the output.

Verilog:

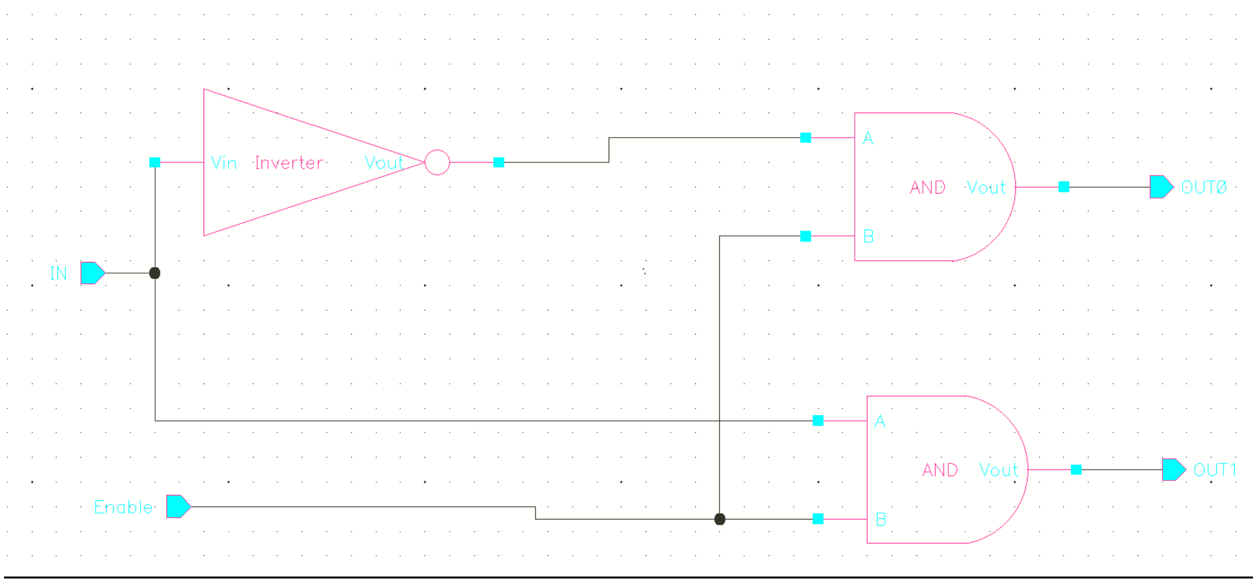
- Create a ROM module containing the frames you will display
- Use the provided clock divider module to create a clock for selecting rows
- Use the provided clock divider module to create a clock for selecting frames
- Use the frame clock to select the current frame inthe ROM
- Use the row clock to select the current row inthe ROM
- Use the output of the ROM for the column output
- Use the row number to generate the row output (remember rows are active low)
- Assign pins and program the board (given in the manual)

List of Items Completed:

a) Design
   - 1:2 decoder
   - 3:8 decoder hierarchical design
   - 7 bit ROM
   - ROM connected to decoder to print ASCII characters.

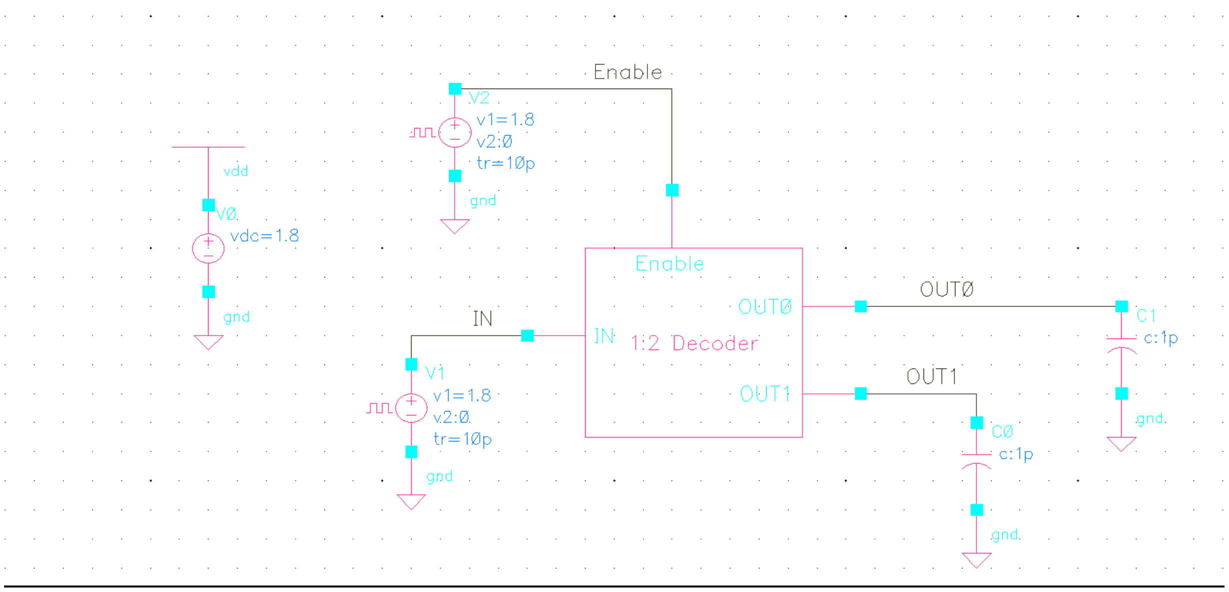
b) Lab Procedures
c) Verilog Code to display 4 different frames.
d) Schematics and test benches
e) Results and Simulations
f) Demo (no given as it was not working on the FPGA board. Could not figure out the problem in time)
g) Analysis and Explanations (includes a brief description of the schematics, results and simulations)
h) Conclusion
i) Answers to Questions

2:1 Decoder:

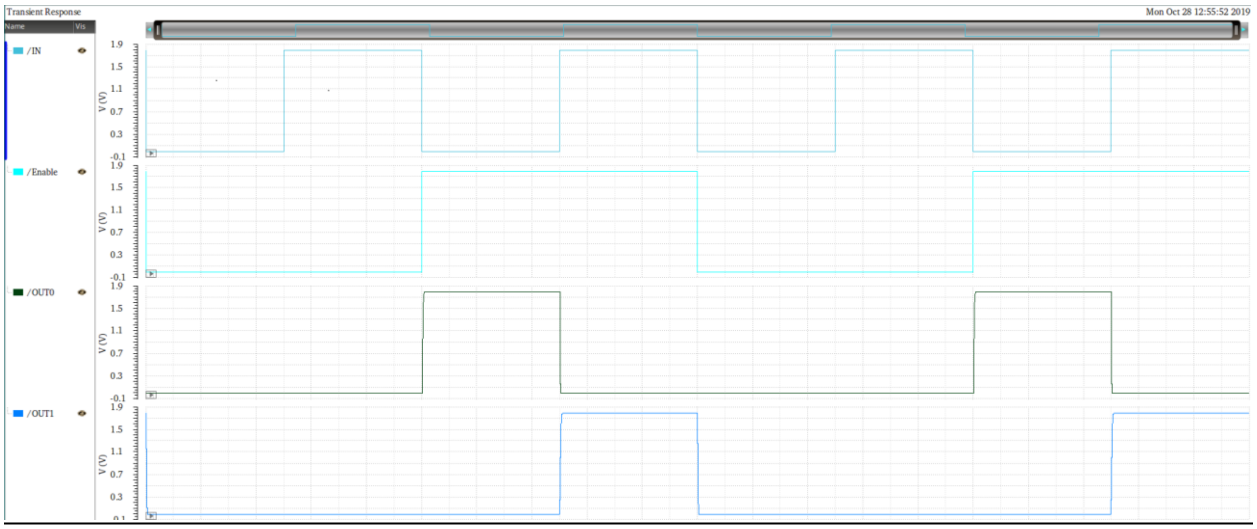


Design of a 2:1 decoder, which is used in the hierarchical design for 3:8 decoder.

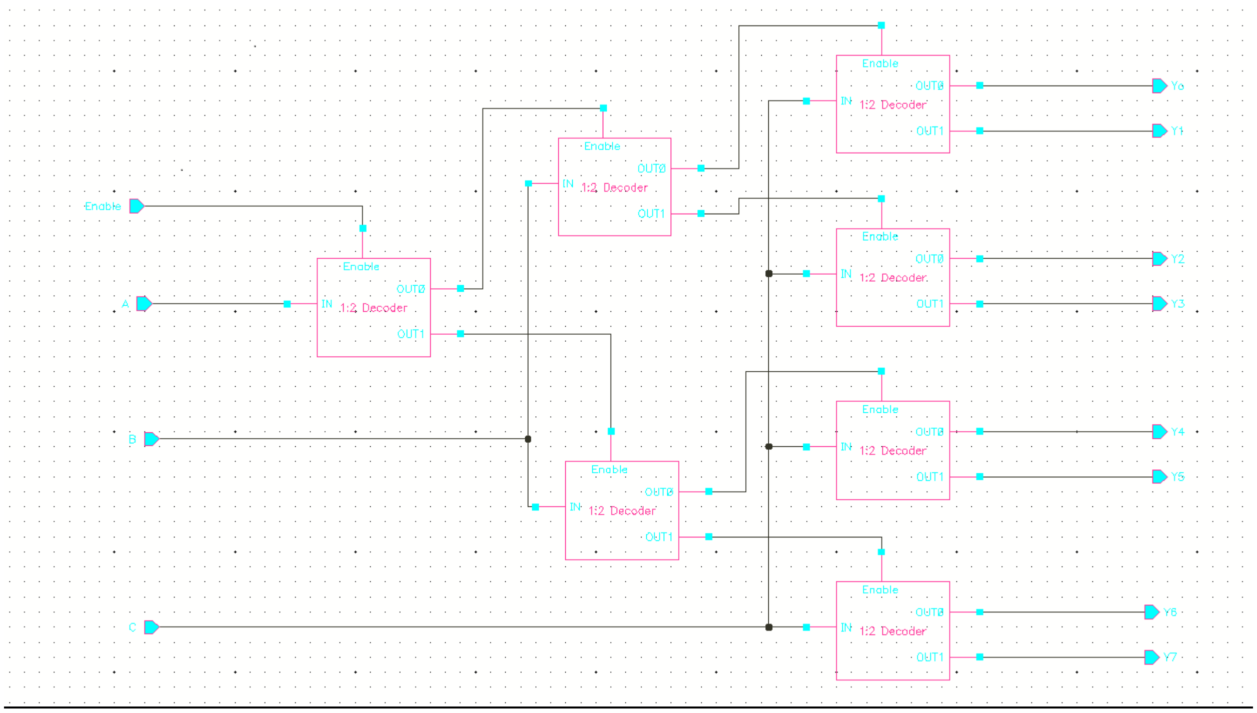
2:1 Decoder test bench:



2:1 Decoder output:

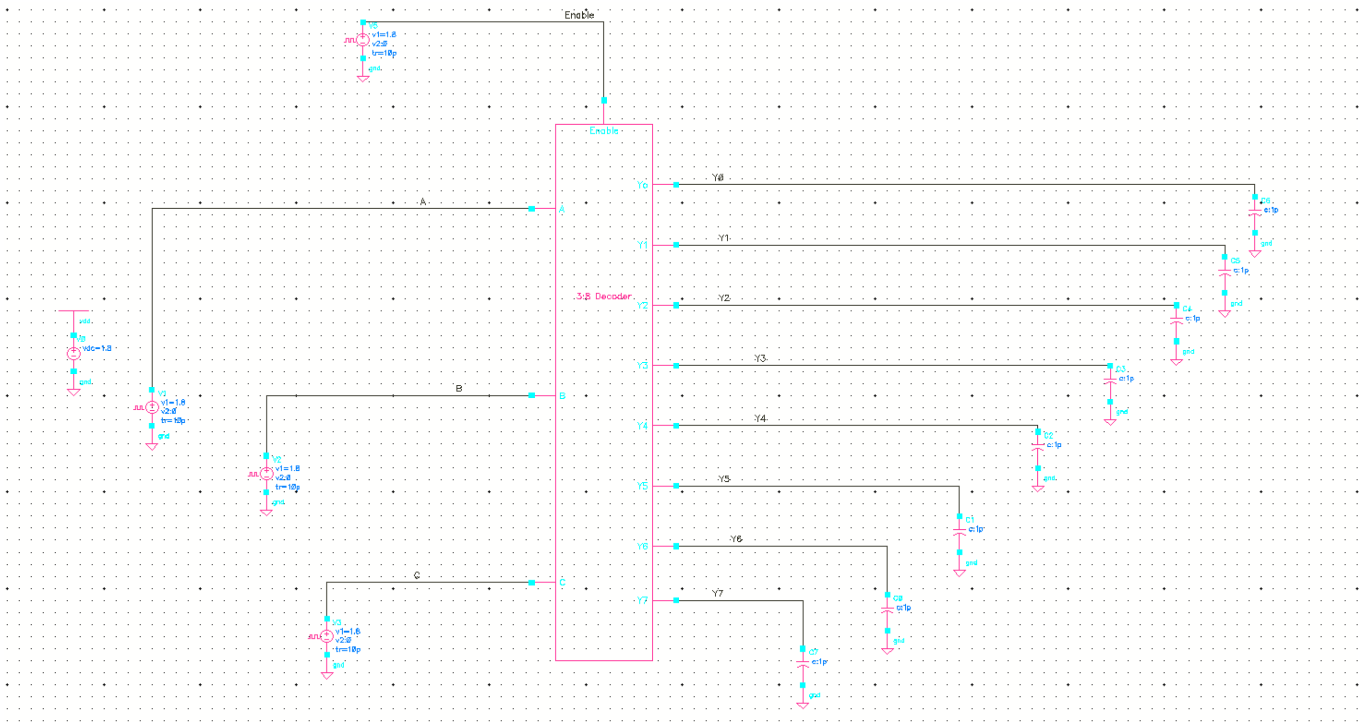


3:8 Decoder:

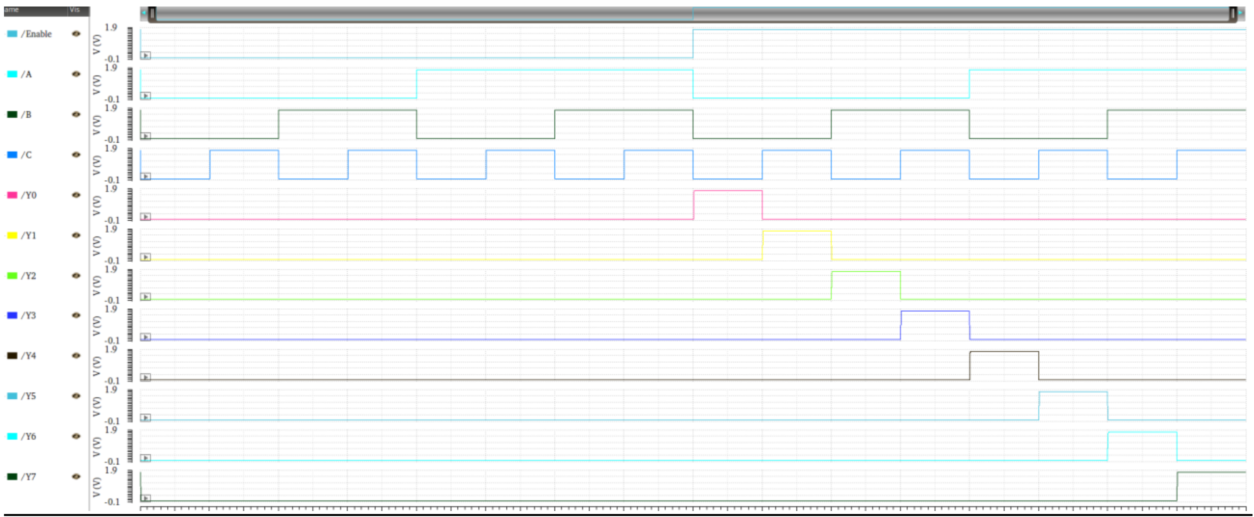


Hierarchical design of a 3:8 decoder, using a 2:1 decoder.

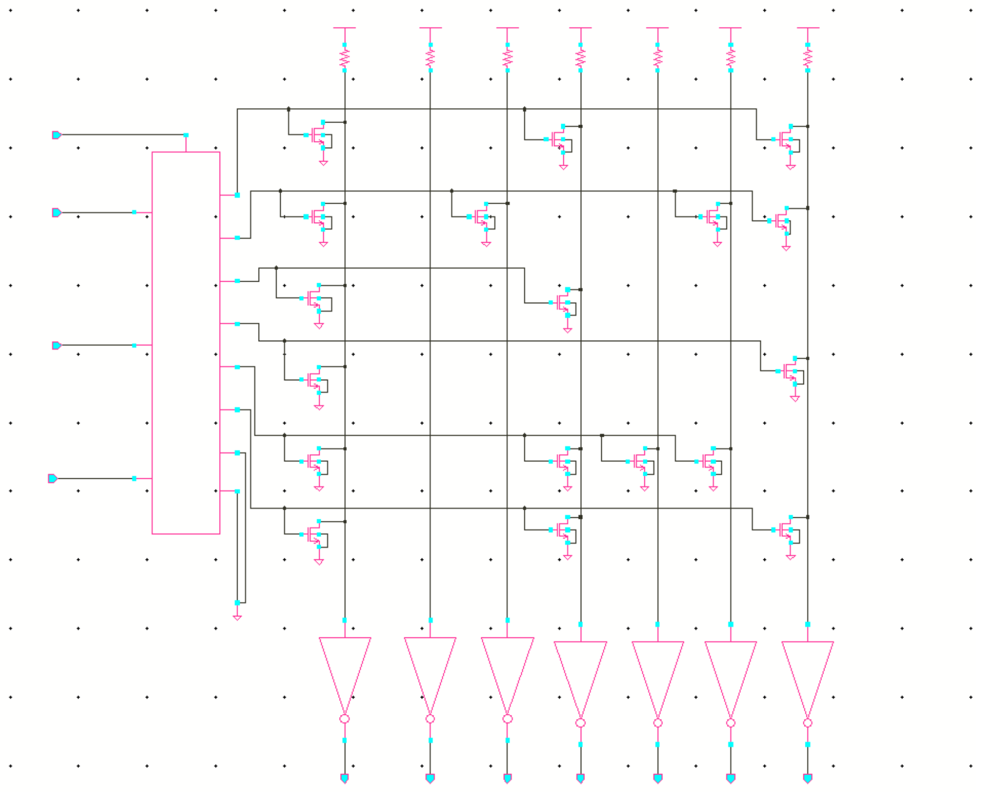
3:8 Decoder test bench:



3:8 Decoder output:

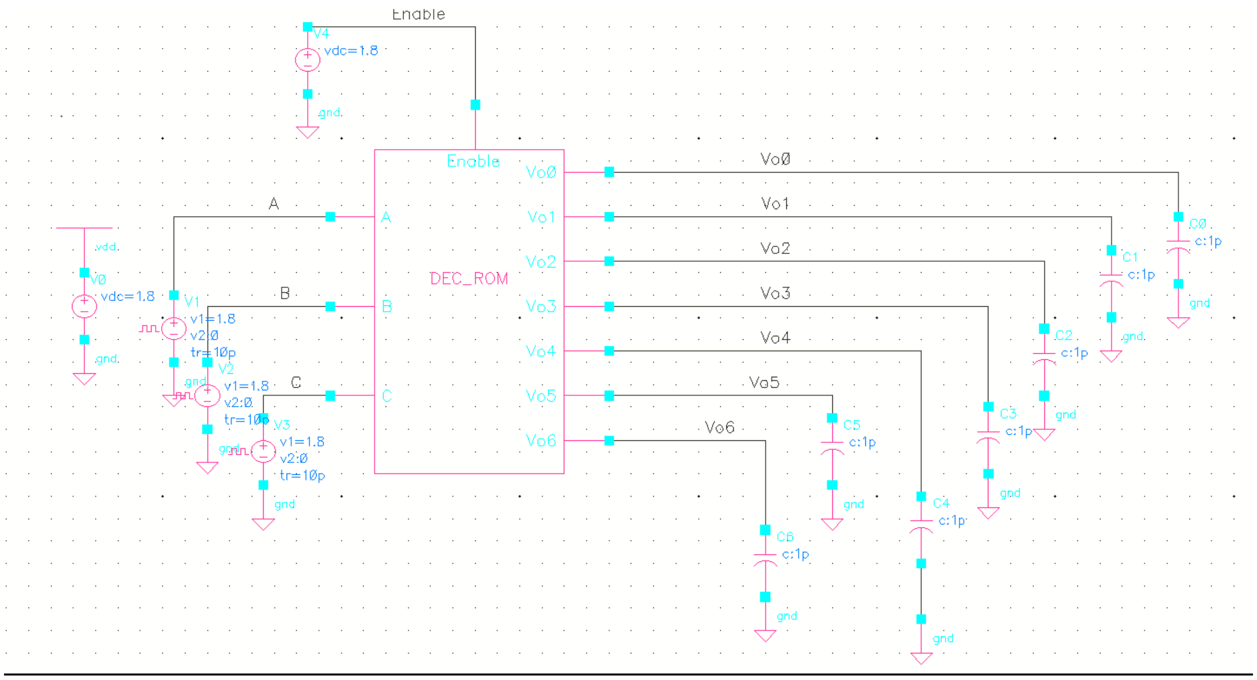


Decoder+ ROM schematic:

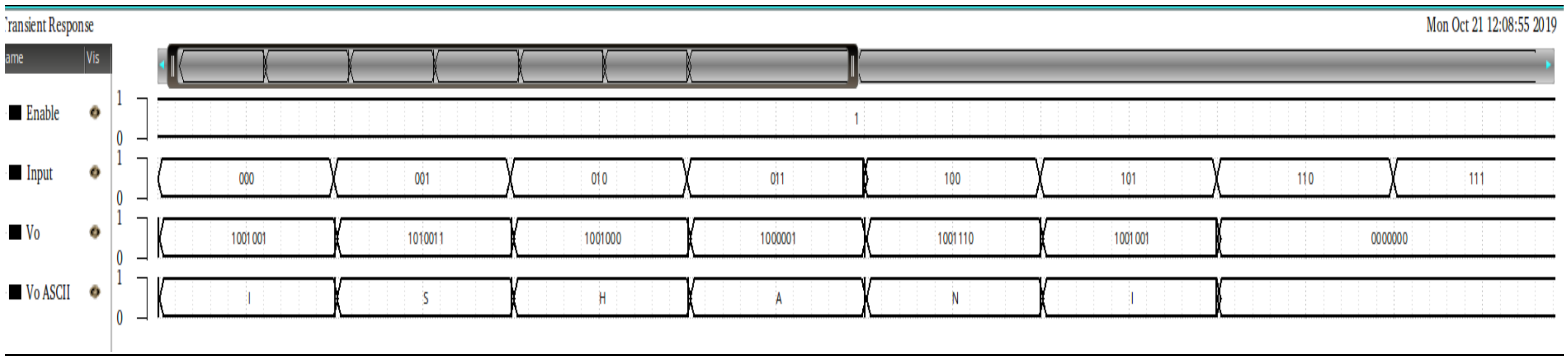


Design of decoder and a 7xN ROM, used to display a 6 letter word in ASCII.

Decoder+ ROM test bench:

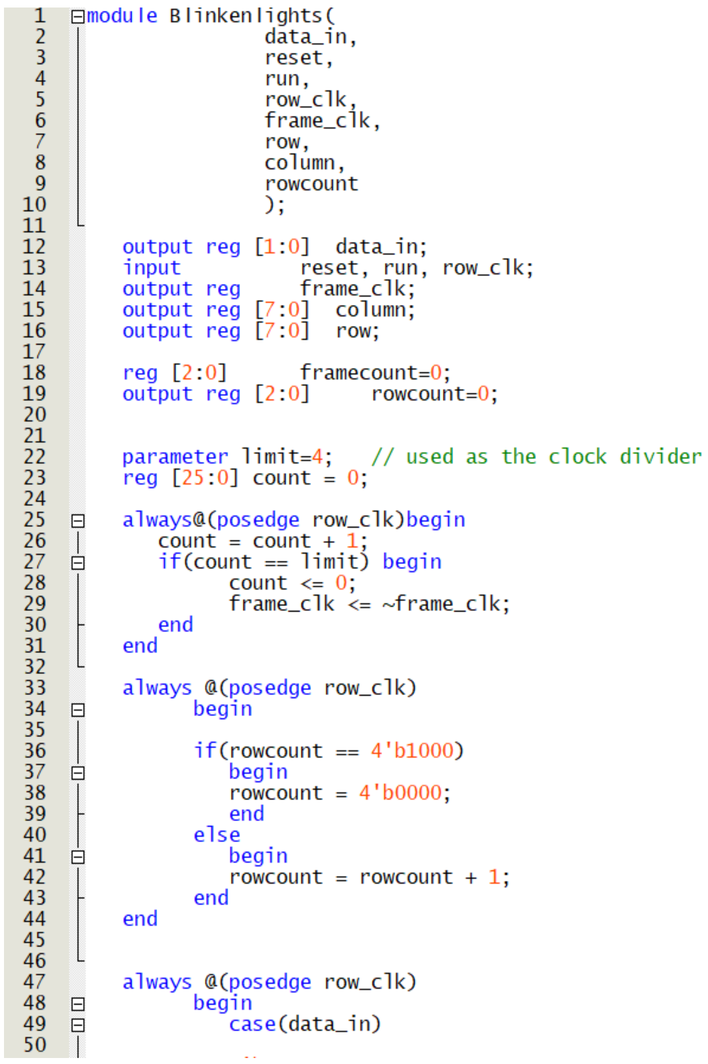


Decoder+ ROM output:

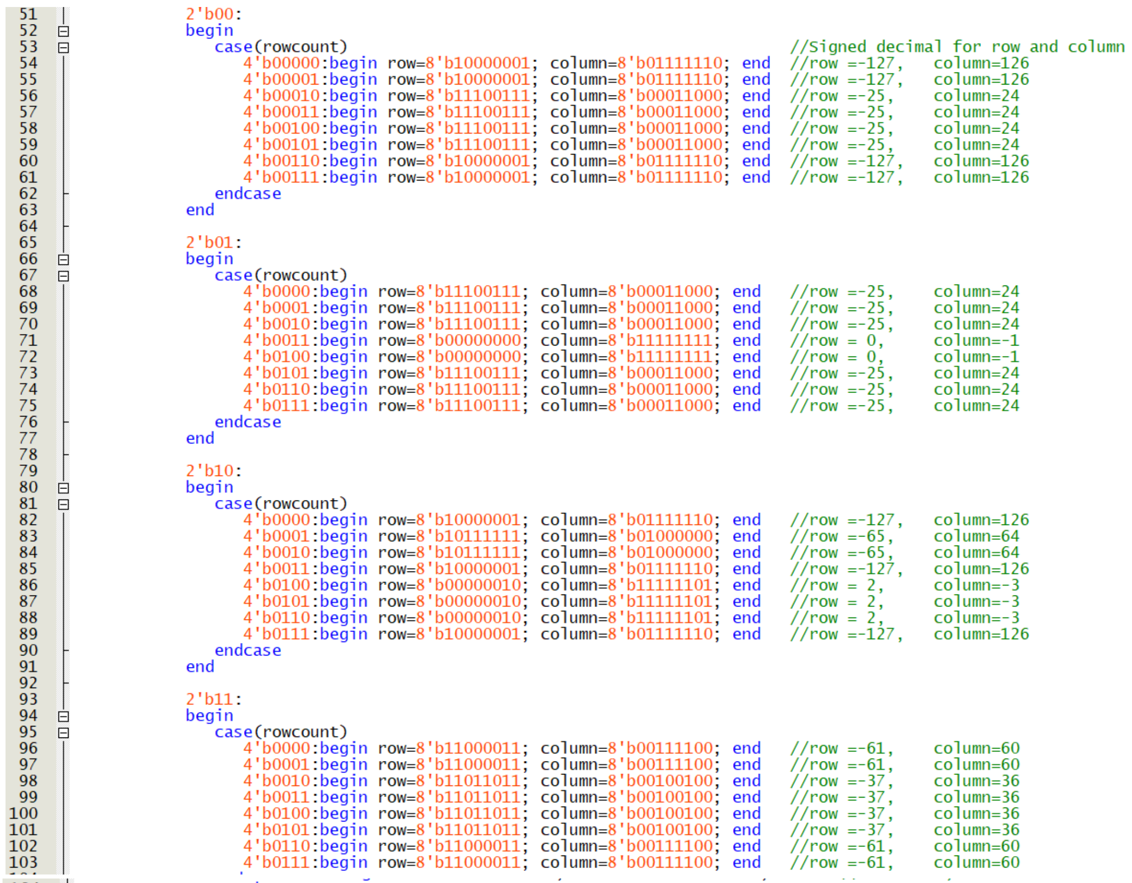


Verilog:

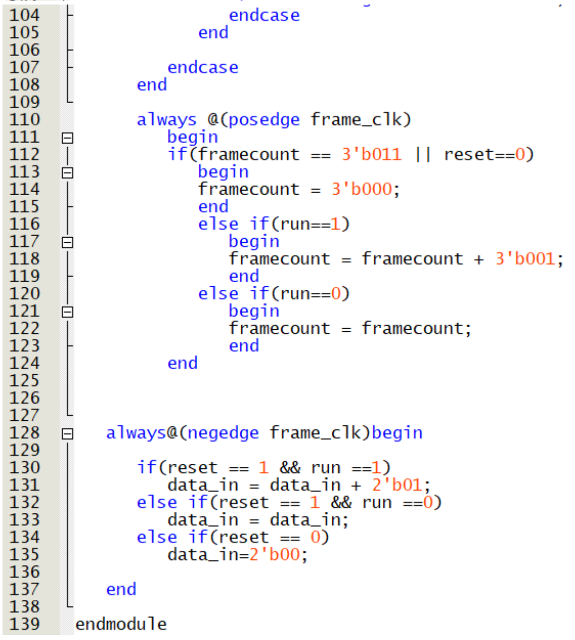
Code:



```
module Blinkenlights(
               data_in,
               reset,
               run,
               row_clk,
               frame_clk,
               row,
               column,
               rowcount
               );

   output reg [1:0]  data_in;
   input          reset, run, row_clk;
   output reg     frame_clk;
   output reg [7:0]  column;
   output reg [7:0]  row;

   reg [2:0]      framecount=0;
   output reg [2:0]     rowcount=0;


   parameter limit=4;   // used as the clock divider
   reg [25:0] count = 0;

   always@(posedge row_clk)begin
      count = count + 1;
      if(count == limit) begin
            count <= 0;
            frame_clk <= ~frame_clk;
      end
   end

   always @(posedge row_clk)
         begin

         if(rowcount == 4'b1000)
            begin
            rowcount = 4'b0000;
            end
         else
            begin
            rowcount = rowcount + 1;
         end
   end

   always @(posedge row_clk)
         begin
            case(data_in)
```



```
            2'b00:
            begin
               case(rowcount)                                                  //Signed decimal for row and column
                  4'b00000:begin row=8'b10000001; column=8'b01111110; end  //row =-127,    column=126
                  4'b00001:begin row=8'b10000001; column=8'b01111110; end  //row =-127,    column=126
                  4'b00010:begin row=8'b11100111; column=8'b00011000; end  //row =-25,     column=24
                  4'b00011:begin row=8'b11100111; column=8'b00011000; end  //row =-25,     column=24
                  4'b00100:begin row=8'b11100111; column=8'b00011000; end  //row =-25,     column=24
                  4'b00101:begin row=8'b11100111; column=8'b00011000; end  //row =-25,     column=24
                  4'b00110:begin row=8'b10000001; column=8'b01111110; end  //row =-127,    column=126
                  4'b00111:begin row=8'b10000001; column=8'b01111110; end  //row =-127,    column=126
               endcase
            end

            2'b01:
            begin
               case(rowcount)
                  4'b0000:begin row=8'b11100111; column=8'b00011000; end   //row =-25,     column=24
                  4'b0001:begin row=8'b11100111; column=8'b00011000; end   //row =-25,     column=24
                  4'b0010:begin row=8'b11100111; column=8'b00011000; end   //row =-25,     column=24
                  4'b0011:begin row=8'b00000000; column=8'b11111111; end   //row = 0,      column=-1
                  4'b0100:begin row=8'b00000000; column=8'b11111111; end   //row = 0,      column=-1
                  4'b0101:begin row=8'b11100111; column=8'b00011000; end   //row =-25,     column=24
                  4'b0110:begin row=8'b11100111; column=8'b00011000; end   //row =-25,     column=24
                  4'b0111:begin row=8'b11100111; column=8'b00011000; end   //row =-25,     column=24
               endcase
            end

            2'b10:
            begin
               case(rowcount)
                  4'b0000:begin row=8'b10000001; column=8'b01111110; end   //row =-127,    column=126
                  4'b0001:begin row=8'b10111111; column=8'b01000000; end   //row =-65,     column=64
                  4'b0010:begin row=8'b10111111; column=8'b01000000; end   //row =-65,     column=64
                  4'b0011:begin row=8'b10000001; column=8'b01111110; end   //row =-127,    column=126
                  4'b0100:begin row=8'b00000010; column=8'b11111101; end   //row = 2,      column=-3
                  4'b0101:begin row=8'b00000010; column=8'b11111101; end   //row = 2,      column=-3
                  4'b0110:begin row=8'b00000010; column=8'b11111101; end   //row = 2,      column=-3
                  4'b0111:begin row=8'b10000001; column=8'b01111110; end   //row =-127,    column=126
               endcase
            end

            2'b11:
            begin
               case(rowcount)
                  4'b0000:begin row=8'b11000011; column=8'b00111100; end   //row =-61,     column=60
                  4'b0001:begin row=8'b11000011; column=8'b00111100; end   //row =-61,     column=60
                  4'b0010:begin row=8'b11011011; column=8'b00100100; end   //row =-37,     column=36
                  4'b0011:begin row=8'b11011011; column=8'b00100100; end   //row =-37,     column=36
                  4'b0100:begin row=8'b11011011; column=8'b00100100; end   //row =-37,     column=36
                  4'b0101:begin row=8'b11011011; column=8'b00100100; end   //row =-37,     column=36
                  4'b0110:begin row=8'b11000011; column=8'b00111100; end   //row =-61,     column=60
                  4'b0111:begin row=8'b11000011; column=8'b00111100; end   //row =-61,     column=60
```



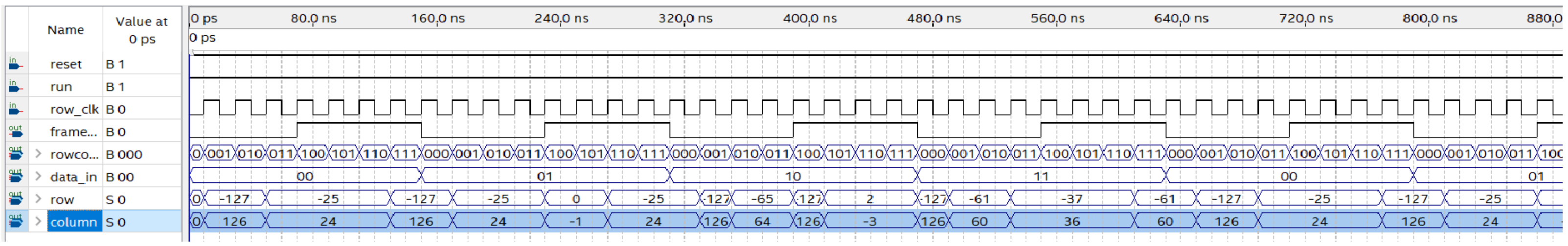
```
               endcase
            end

         endcase
      end

      always @(posedge frame_clk)
         begin
         if(framecount == 3'b011 || reset==0)
            begin
            framecount = 3'b000;
            end
            else if(run==1)
               begin
               framecount = framecount + 3'b001;
               end
            else if(run==0)
               begin
               framecount = framecount;
               end
         end



   always@(negedge frame_clk)begin

      if(reset == 1 && run ==1)
         data_in = data_in + 2'b01;
      else if(reset == 1 && run ==0)
         data_in = data_in;
      else if(reset == 0)
         data_in=2'b00;

   end

endmodule
```

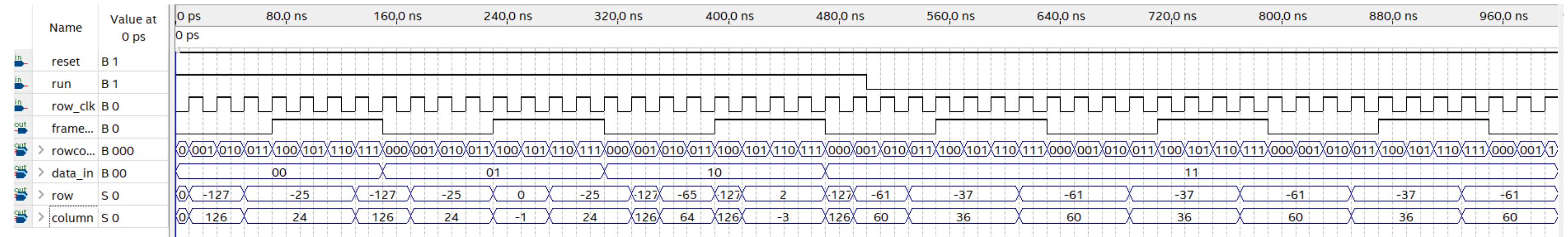
Analysis and Explanations:

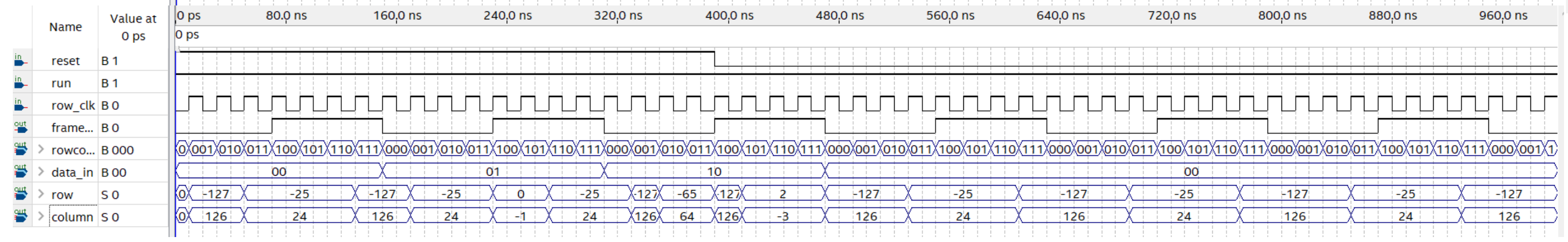
Cadence- By connecting multiple 1:2 decoders, 3:8 decoder was designed (hierarchical design), which was then connected to a 7 bit ROM, the output of this module gives a 5 letter word (e.g. my name- ISHANI). The schematics, test-benches and the results of simulations have been presented above.

Verilog- I have initiated a counter to keep track of all the rows and frames. In the output I have given the period of the system clock as 20ns, which is 50MHz. Therefore, I have provided a period of 320ns, for the data_in LSB, to capture all of the 8 transitions corresponding to each of the row and column. I have also provided the signed decimals representation of each row and column so that it is easier to debug and can be seen easily in the output waveform. The output for the code is given below. The code for the LEDs is assigned signed integer values in the comments for the code. The simulation output follows the row and column values and can be easily deduced from the assigned integer values. Three conditions for results have been given where reset=run=1, reset=1, run=0 and reset=0 and run=1.

Output:







Conclusion:

In this lab, Blinkenlights, which was divided into 2 parts Cadence and Verilog, we have designed a 7XN ROM + decoder to give output as ASCII characters, i.e. output of the block is at least a 5 letter word. This was done using a hierarchical design of 3:8 decoder and the 7 bit ROM. The Verilog code, which was to display at least 4 different frames using a ROM module containing the frames and a clock divider module (given), was implemented as well. This lab taught me about implementation and concept of hierarchical design and decoders as well as the right way of using the clock divider.

Questions:

- Discuss the concept and advantages of hierarchical design.

The concept of hierarchical design is when a design is divided into multiple sub modules, and repeating the operations on the sub modules, dividing them further into more simplified blocks, is called a hierarchical design. This is done to reduce the design complexity, as when the circuit increases in size, so does its complexity. Therefore, to reduce the design complexity and for ease of debugging, hierarchical design is used.

The advantages of a hierarchical design are:

1. Improved functionality of the circuit.
2. Easy debugging of the circuit, as it is easier to debug a smaller module, which is replicated in the design.
3. Faster run time, as designers can work on an individual block, which are much smaller, when compared to the design.
4. Timing issues with a hierarchical design can be spotted and fixed easily.

- How was hierarchical design utilized in this lab?

In this lab, a 3:8 decoder is designed using a basic 2:1 decoder, which is much simpler and smaller compared to the 3:8 decoder. It is a 3 level hierarchical design, where one 2:1 decoder drives the enables of two other 2:1 decoders, and inputs of all the decoders in that level, are connected, thus we get a 3:8 decoder at the output.