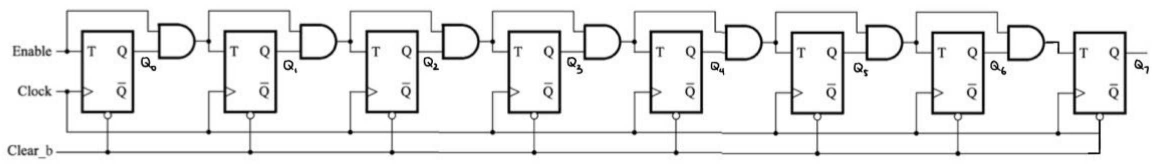
Ryan Chang 1004103748

Prelab 5

PART1

Schematic



Verilog

```
module main_counter (SW, KEY, HEX0, HEX1);

  input[1:0] KEY;
  input[1:0] SW;
  output[6:0] HEX0;
  output[6:0] HEX1;
  wire[7:0] counterToSS;

  counter u0(
    .enable(SW[1]),
    .clock(KEY[0]),
    .clear_b(SW[0]),
    .q(counterToSS[7:0])

  );

  sevenseg S1 (
    .Data(counterToSS[3:0]),
    .Display(HEX0)
  );

  sevenseg S2 (
    .Data(counterToSS[7:4]),
    .Display(HEX1)
  );
```

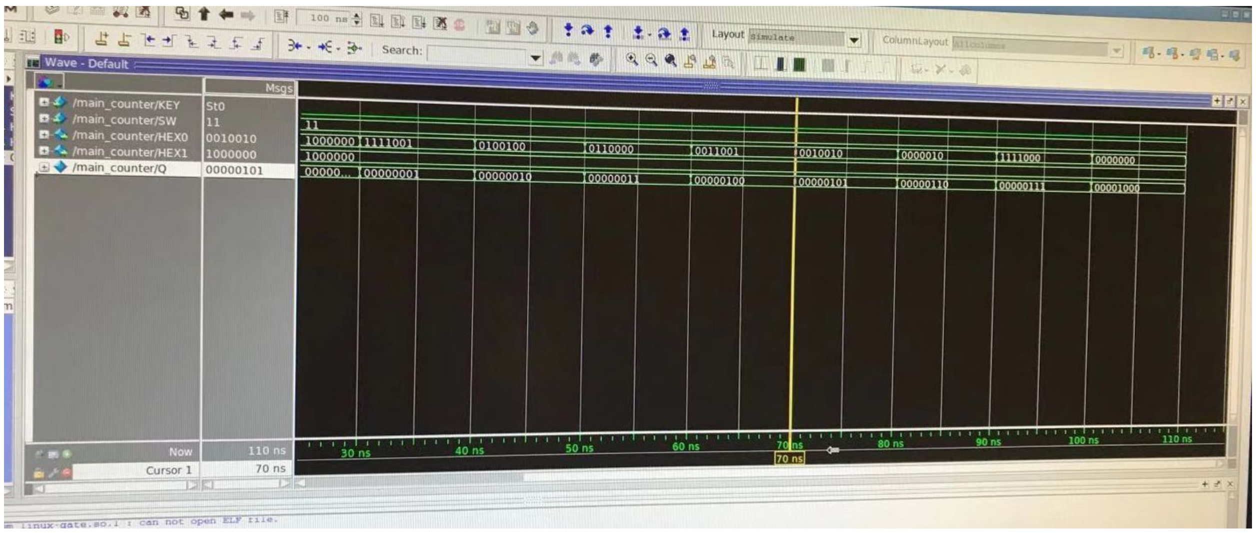
```
endmodule



module counter (enable, clock, clear_b, q);

  input enable, clock, clear_b;
  output wire[7:0] q;

  tflipflop T0 (
    .t(enable),
    .clock(clock),
    .clear_b(clear_b),
    .q(q[0])
  );

  tflipflop T1 (
    .t(enable && (& q[0])),
    .clock(clock),
    .clear_b(clear_b),
    .q(q[1])
  );

  tflipflop T2 (
    .t(enable && (& q[1:0])),
    .clock(clock),
    .clear_b(clear_b),
    .q(q[2])
  );

  tflipflop T3 (
    .t(enable && (& q[2:0])),
    .clock(clock),
    .clear_b(clear_b),
    .q(q[3])
  );

  tflipflop T4 (
    .t(enable && (& q[3:0])),
    .clock(clock),
    .clear_b(clear_b),
    .q(q[4])
  );

  tflipflop T5 (
    .t(enable && (& q[4:0])),
    .clock(clock),
    .clear_b(clear_b),
    .q(q[5])
  );

  tflipflop T6 (
    .t(enable && (& q[5:0])),
    .clock(clock),
    .clear_b(clear_b),
    .q(q[6])
  );

  tflipflop T7 (
    .t(enable && (& q[6:0])),
    .clock(clock),
    .clear_b(clear_b),
    .q(q[7])
  );

endmodule



module tflipflop (t, clock, clear_b, q);

    input t, clock, clear_b;
    output reg q;

    always @(posedge clock, negedge clear_b)
    begin
      if(~clear_b)
        q <= 0;
     else
        q <= q ^ t;
    end

endmodule
```

```
module sevenseg (Display, Data);
   input[3:0] Data;
   output[6:0] Display;

   assign Display[0] = ~Data[3] & ~Data[2] & ~Data[1] &  Data[0] |
        ~Data[3] &  Data[2] & ~Data[1] & ~Data[0] |
         Data[3] &  Data[2] & ~Data[1] &  Data[0] |
         Data[3] & ~Data[2] &  Data[1] &  Data[0];

   assign Display[1] = ~Data[3] &  Data[2] & ~Data[1] &  Data[0] |
         Data[3] &  Data[2] & ~Data[1] & ~Data[0] |
         Data[3] &         Data[1] &  Data[0] |
                Data[2] &  Data[1] & ~Data[0];

   assign Display[2] =  Data[3] &  Data[2] & ~Data[1] & ~Data[0] |
        ~Data[3] & ~Data[2] &  Data[1] & ~Data[0] |
         Data[3] &  Data[2] &  Data[1];


   assign Display[3] = ~Data[3] &  Data[2] & ~Data[1] & ~Data[0] |
             ~Data[2] & ~Data[1] &  Data[0] |
                Data[2] &  Data[1] &  Data[0] |
         Data[3] & ~Data[2] &  Data[1] & ~Data[0];

   assign Display[4] = ~Data[3] &                  Data[0] |
        ~Data[3] &  Data[2] & ~Data[1]        |
             ~Data[2] & ~Data[1] &  Data[0];


   assign Display[5] = ~Data[3] & ~Data[2]         &  Data[0] |
        ~Data[3] & ~Data[2] &  Data[1]        |
        ~Data[3] &        Data[1] &  Data[0] |
         Data[3] &  Data[2] & ~Data[1] &  Data[0];

   assign Display[6] = ~Data[3] & ~Data[2]  & ~Data[1]         |
        ~Data[3] &  Data[2] &  Data[1] &  Data[0] |
         Data[3] &  Data[2] & ~Data[1] & ~Data[0];

Endmodule
```

VSIM CODE

```
# Set the working dir, where all compiled Verilog goes.
vlib work

# Compile all verilog modules in mux.v to working dir;
# could also have multiple verilog files.
vlog -timescale 1ns/1ns main_counter.v

# Load simulation using mux as the top level simulation module.
vsim main_counter
#SW[1] is enable        SW[0] clear_b

# Log all signals and add some signals to waveform window.
log {/*}
# add wave {/*} would add all items in top level simulation module.
add wave {/*}

#SW[1] is enable        SW[0] clear_b
# Set input values using the force command, signal names need to be in {} brackets.
force {SW[0]} 0
run 10ns
force {SW[0]} 1
run 10ns

force {KEY[0]} 0 0, 1 10 -repeat 10
force {SW[1]} 1
run 10ns
run 10ns
run 10ns
run 10ns
run 10ns
run 10ns
run 10ns
run 10ns
run 10ns
```

VSIM SCREENSHOT



Questions

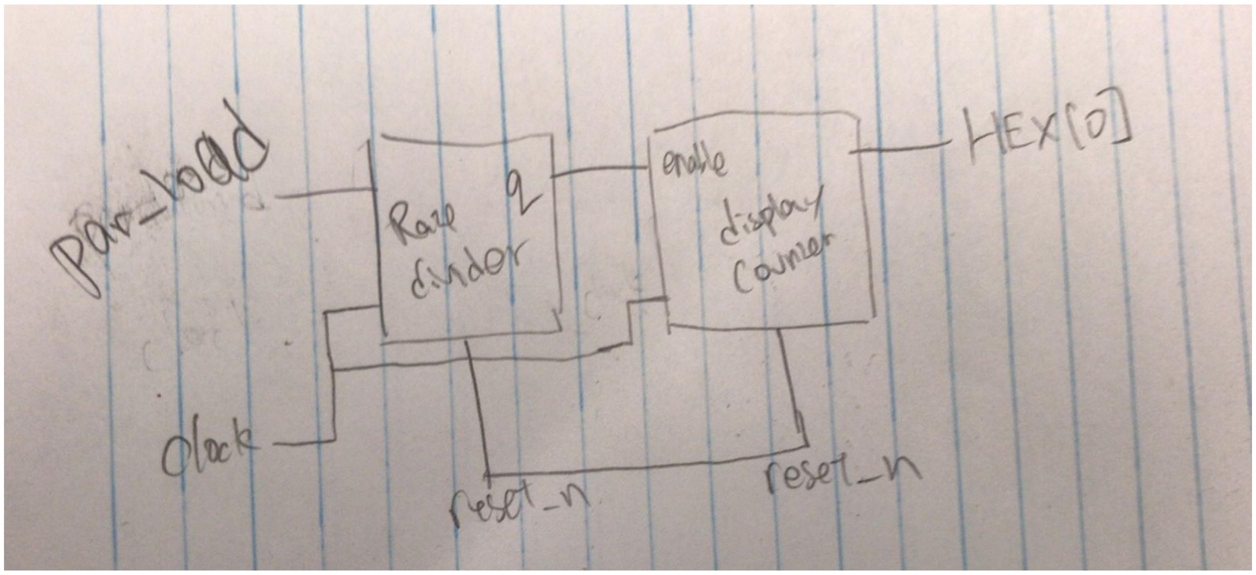
<1 percent

621.89MHz

There are four more counters in our implementation furthermore, there is two seven segment decoders connected to the output of the 8 bit counter.

PART 2

Schematic



Questions

Because you cannot reach F with 4 bits

If (q == 1001)

It will be so fast it will go from 0 to F instantly.

You will need 26 bits to count to 1 HZ from 50MHZ blocks.

VERILOG CODE

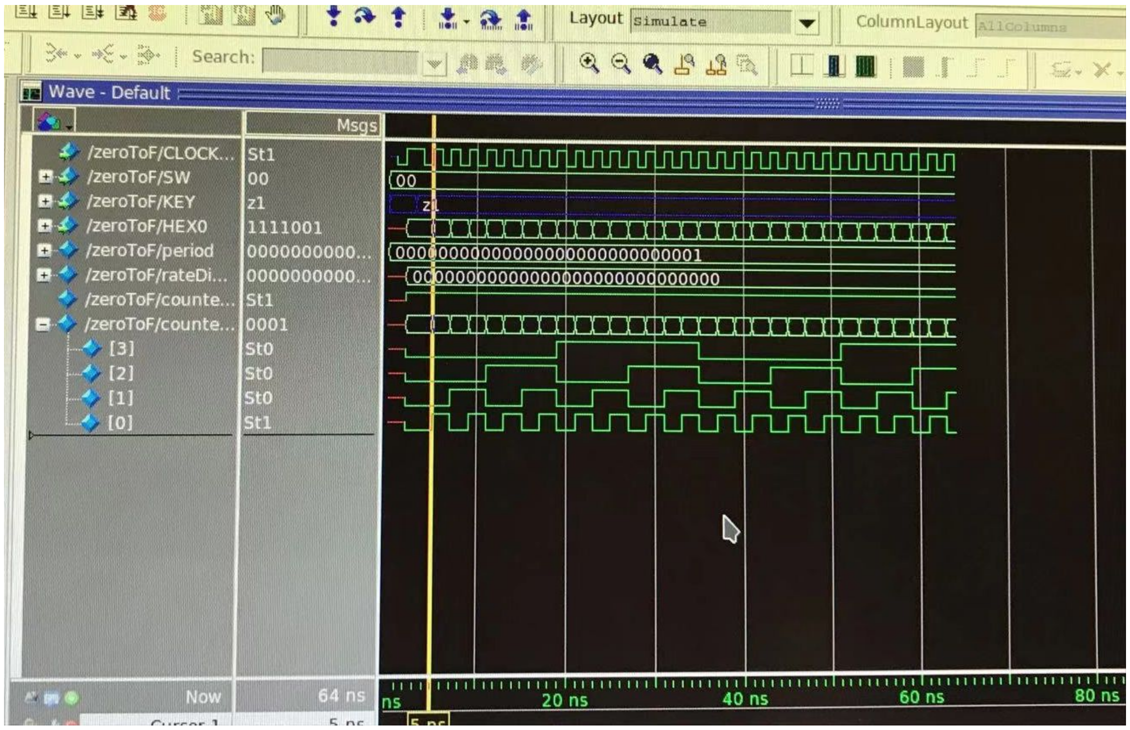
```
module zeroToF (SW, KEY, CLOCK_50, HEX0);
   input CLOCK_50;
         input [1:0] SW;
         input [1:0] KEY;
         input [6:0] HEX0;

   reg [28:0] period;
   wire [28:0] rateDividerOutput;
   wire counterEnable;
   wire [3:0] counterOut;

   always @(SW)
         begin
         case (SW[1:0])
         2'b00: period <= 1;
         2'b01: period <= 50000000;
         2'b10: period <= 100000000;
         2'b11: period <= 200000000;
         default: period <= 1;
         endcase
         end

    assign counterEnable = (rateDividerOutput == 0) ? 1 : 0;

    rateDivider rd1(
          .par_load(period),
          .clock(CLOCK_50),
          .reset_n(KEY[0]),
          .q(rateDividerOutput)
    );

   fourbitcounter fbc1(
          .enable(counterEnable),
          .clock(CLOCK_50),
          .reset_n(KEY[0]),
          .q(counterOut)
    );
   sevenseg ss1(
          .Data(counterOut),
          .Display(HEX0)
   );
endmodule

module rateDivider (par_load, clock, reset_n, q);
   input [28:0] par_load;
   input clock;
   input reset_n;

   output reg[28:0] q;

   always @(posedge clock)
   begin
     if (reset_n == 1'b0)
          q <= par_load - 1'b1;
          else
                  begin
                          if (q == 0)
                                  q <= par_load - 1'b1;
                          else
                                  q <= q - 1'b1;
                  end
   end

endmodule

module fourbitcounter (enable, clock, reset_n, q);

   input enable, clock, reset_n;
   output reg[3:0] q;

   always @(posedge clock)
   begin
     if (reset_n == 1'b0)
          q <= 0;
          else if (enable == 1)
                  begin
                          if (q==4'b1111)
                                  q <= 0;
```

```
                        else
                                q <= q + 1'b1;
        end
    end

endmodule

module sevenseg (Display, Data);
    input[3:0] Data;
    output[6:0] Display;

    assign Display[0] = ~Data[3] & ~Data[2] & ~Data[1] &  Data[0] |
                  ~Data[3] &  Data[2] & ~Data[1] & ~Data[0] |
                   Data[3] &  Data[2] & ~Data[1] &  Data[0] |
                   Data[3] & ~Data[2] &  Data[1] &  Data[0];

    assign Display[1] = ~Data[3] &  Data[2] & ~Data[1] &  Data[0] |
                   Data[3] &  Data[2] & ~Data[1] & ~Data[0] |
                   Data[3] &                 Data[1] &  Data[0] |
                         Data[2] &  Data[1] & ~Data[0];

    assign Display[2] =  Data[3] &  Data[2] & ~Data[1] & ~Data[0] |
                  ~Data[3] & ~Data[2] &  Data[1] & ~Data[0] |
                   Data[3] &  Data[2] &  Data[1];


    assign Display[3] = ~Data[3] &  Data[2] & ~Data[1] & ~Data[0] |
                        ~Data[2] & ~Data[1] &  Data[0] |
                         Data[2] &  Data[1] &  Data[0] |
                   Data[3] & ~Data[2] &  Data[1] & ~Data[0];

    assign Display[4] = ~Data[3] &                  Data[0] |
                  ~Data[3] &  Data[2] & ~Data[1]        |
                        ~Data[2] & ~Data[1] &  Data[0];


    assign Display[5] = ~Data[3] & ~Data[2]               &  Data[0] |
                  ~Data[3] & ~Data[2] &  Data[1]        |
                  ~Data[3] &                Data[1] &  Data[0] |
                   Data[3] &  Data[2] & ~Data[1] &  Data[0];

    assign Display[6] = ~Data[3] & ~Data[2]  & ~Data[1]        |
                  ~Data[3] &  Data[2] &  Data[1] &  Data[0] |
                    Data[3] &  Data[2] & ~Data[1] & ~Data[0];

endmodule
```

VSIM CODE

```
# Set the working dir, where all compiled Verilog goes.
vlib work

# Compile all verilog modules in mux.v to working dir;
# could also have multiple verilog files.
vlog -timescale 1ns/1ns lab5part2.v

# Load simulation using mux as the top level simulation module.
vsim zeroToF
#SW[1] is enable        SW[0] clear_b

# Log all signals and add some signals to waveform window.
log {/*}
# add wave {/*} would add all items in top level simulation module.
add wave {/*}
#SW[1]          SW[0]            reset = KEY[0](active low)
#test 1HZ case
# reset, set initial values
force {SW[0]} 0
force {SW[1]} 0
force {KEY[0]} 0
run 1ns

#go thru one cycle
force {CLOCK_50} 0
run 1ns
force {CLOCK_50} 1
run 1ns
# turn off reset
force {KEY[0]} 1
run 1ns
# clock speed
force {CLOCK_50} 0 0, 1 1 -repeat 2
force {SW[0]} 0
force {SW[1]} 0
run 10ns
run 10ns
run 10ns
run 10ns
run 10ns
run 10ns
```

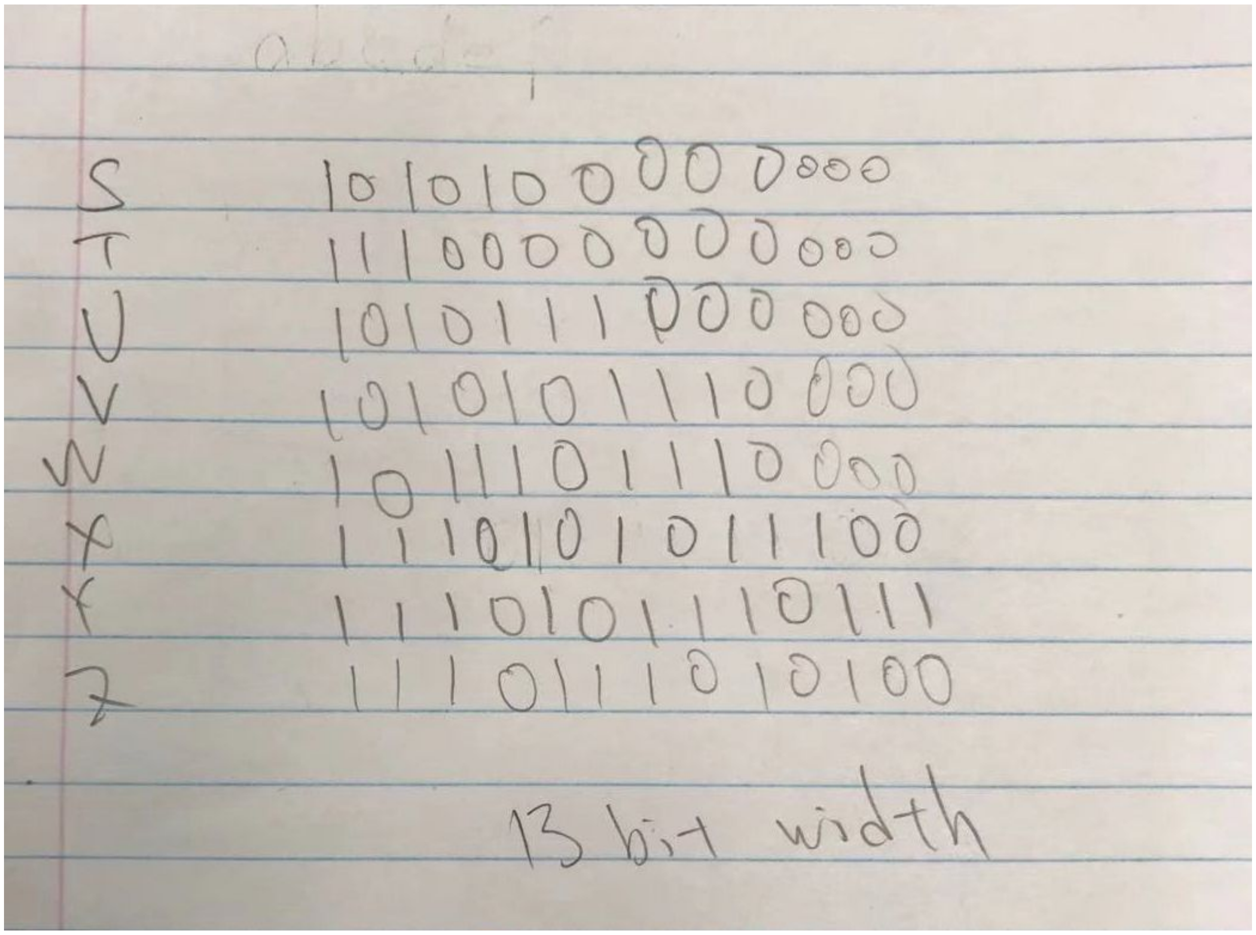
VSIM SCREENSHOT



```
Layout Simulate        ColumnLayout AllColumns

/zeroToF/CLOCK...   St1
/zeroToF/SW         00
/zeroToF/KEY        z1
/zeroToF/HEX0       1111001
/zeroToF/period     0000000000...
/zeroToF/rateDi...  0000000000...
/zeroToF/counte...  St1
/zeroToF/counte...  0001
  [3]               St0
  [2]               St0
  [1]               St0
  [0]               St1

Now                 64 ns
Cursor 1            5 ns
```

PART 3

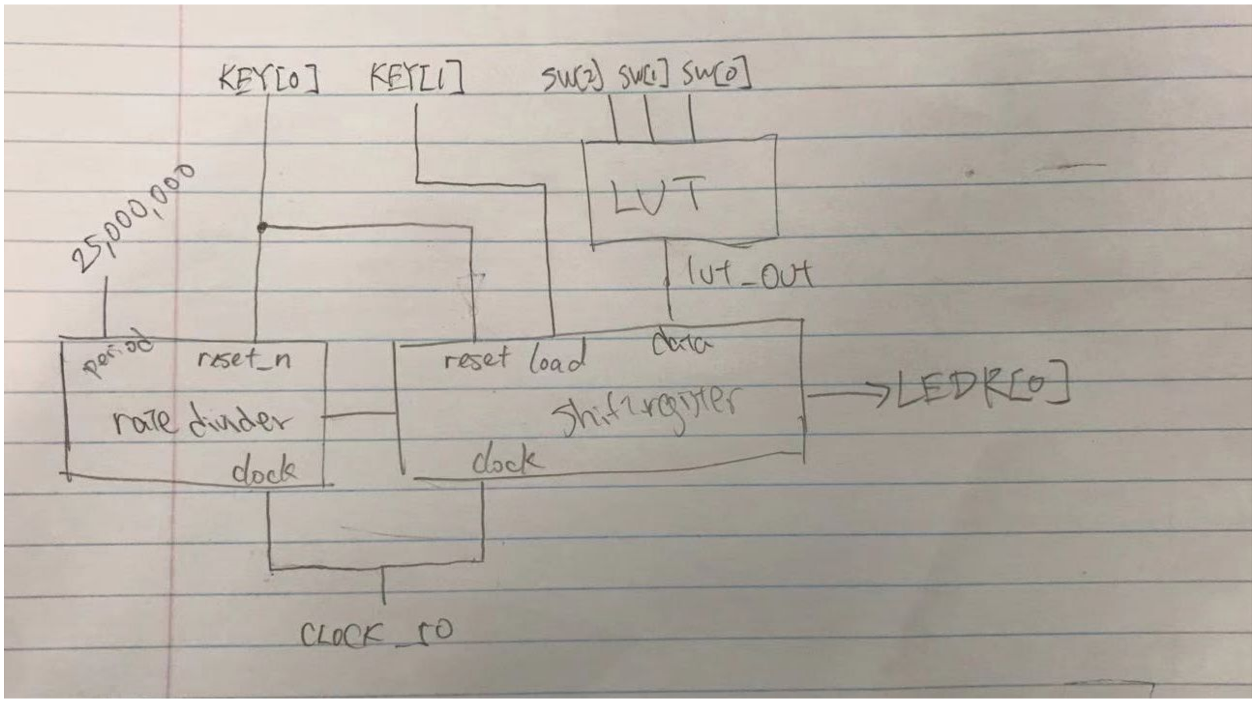
TABLE



S	1010100000000
T	1110000000000
U	1010111000000
V	1010101110000
W	1011101110000
X	1110101011100
Y	1110101110111
Z	1110111010100

13 bit width

SCHEMATIC



VERILOG CODE

```
module morse (KEY, SW, CLOCK_50, LEDR);

    input CLOCK_50;
    input [1:0] KEY;
    input [2:0] SW;

    output [1:0] LEDR;

    wire [28:0] period;
    wire [28:0] rateDividerOut;
    wire [15:0] lutOut;
    wire shiftEnable;


    assign period = 25000000;


    lut u0 (
        .letter(SW[2:0]),
        .morseCode(lutOut)
    );

    rateDivider rd1 (
        .clock(CLOCK_50),
        .period(period),
        .reset_n(KEY[0]),
        .q(rateDividerOut)
    );
    assign shiftEnable = (rateDividerOut == 0) ? 1 : 0;

    shifter s1 (
        .clock(CLOCK_50),
        .load(KEY[1]),
        .enable(shiftEnable),
        .reset(KEY[0]),
        .data(lutOut),
        .out(LEDR[0])
    );

endmodule
```

```
module lut(letter, morseCode);

    input [2:0] letter;
    output reg [15:0] morseCode;

    always @(*)
    begin
        case(letter)
            3'b000: morseCode = 16'b1010100000000000;
            3'b001: morseCode = 16'b1110000000000000;
            3'b010: morseCode = 16'b1010111000000000;
            3'b011: morseCode = 16'b1010101110000000;
            3'b100: morseCode = 16'b1011101110000000;
            3'b101: morseCode = 16'b1110101011100000;
            3'b110: morseCode = 16'b1110101110111000;
            3'b111: morseCode = 16'b1110111010100000;
            default: morseCode = 16'b0000000000000000;
        endcase
    end

endmodule


module rateDivider (clock, period, reset_n, out);

    input clock;
    input reset_n;
    input [28:0] period;

    output reg [28:0] out;


    always @(posedge clock, posedge reset_n)
    begin
        if (reset_n == 1'b0)
            out <= period - 1'b1;
        else
            begin
                if (out == 0)
                    out <= period - 1'b1;
                else
                    out <= out - 1'b1;
        end
    end

endmodule


module shifter(enable, load, clock, reset, data, out);

    input enable, load, clock, reset;
    input [15:0] data;

    reg [15:0] MCodeBit;
    output out;

    always @(posedge load, posedge clock,posedge reset)
    begin
        if (load)
            MCodeBit = data;
        else if (reset)
            MCodeBit = 0;
        else if (enable == 1'b1)
            MCodeBit = MCodeBit << 1'b1;
    end

    assign out = MCodeBit[15];

endmodule
```