Omar Naffaa
ECE 3300L-03
April 25, 2019

Report - Frequency / Period Meter

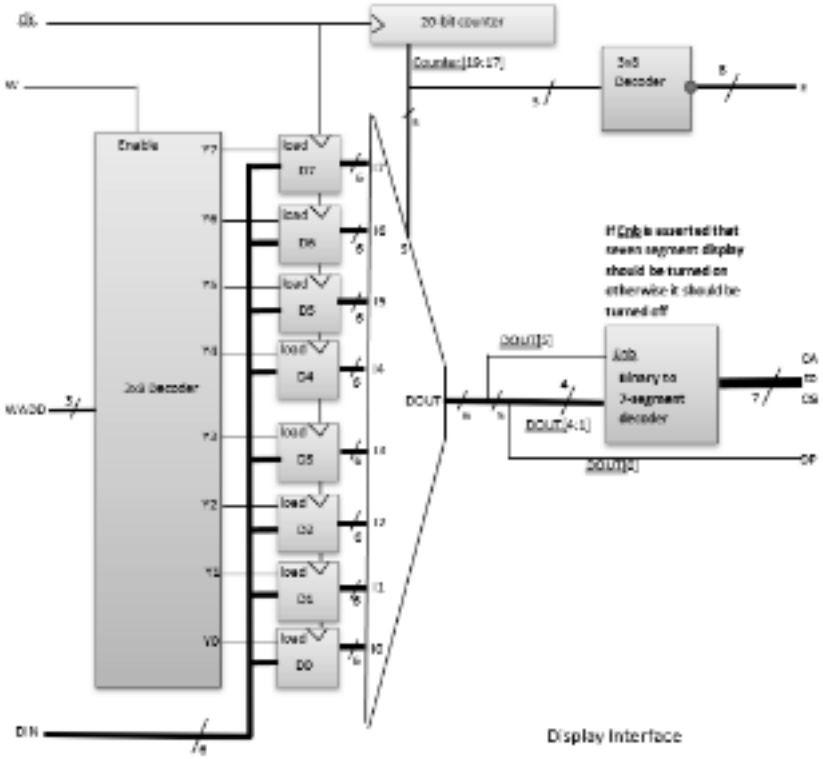
Theory:

The purpose of the experiment was to create a frequency and period meter by combining 5 different modules. A display interface was designed to output to the 7-segment display by multiplexing, which utilizes an effect known as Persistence of Vision (POV) to control all the displays with 7 pins rather than the 56 that would be required to run all the displays. This reduces the amount of pins since the enable pin for each display is toggled briefly, which gives the effect of all the displays being on at the same time although only one display is on at a time. Afterwards, a display controller was created to take a 32 bit BCD value and loads each set of 4 bits (BCD[3:0], BCD[7:4], etc.) into a register and multiplexes the contents of the 8 registers onto the display by toggling the select (WADD), register to load, and the enable (w) using a 3 bit counter. Additionally, a binary to BCD (B2BCD) module was used to take a 10 bit number that represents either a frequency or period in binary and convert it to BCD. This is done by using a select to determine which value will be selected and sent to the display controller, then applying shifts and adding 3 until a comparator is greater than 4. Once this condition is satisfied the 4 bit BCD numbers are captured and sent as a 32 bit BCD value to the display controller. Next a frequency meter was created by creating a module that counts the positive edges of an incoming signal and incrementing a counter, which is then captured in a register F and sent to the B2BCD module in order to be displayed. Finally, a period meter was designed by passing the negative edge of the frequency meter to the period meter in order to synchronize the two modules. The period module then measures the period by starting the measurement at the a negative edge and incrementing a register P until the next negative edge is detected. The output is a frequency and corresponding period on the 7-segment display on the FPGA.

Design:

Display Interface:

Schematic:



Hardware Description Code:

```
module DisplayInterface(input clk, input W, input[2:0] WADD, input[5:0] DIN,
output reg[7:0] E, output reg[6:0] CA_G, output wire DP);
    wire[2:0] S;
    reg[5:0] D0, D1, D2, D3, D4, D5, D6, D7, DOUT; // registers D0 - D7 and DOUT
    reg[19:0] counter;

    assign S = (counter[19:17]);

    // 20 bit Counter - select for encoder and 2nd 3x8 decoder
    always@(posedge clk)
    counter <= counter + 1;
```

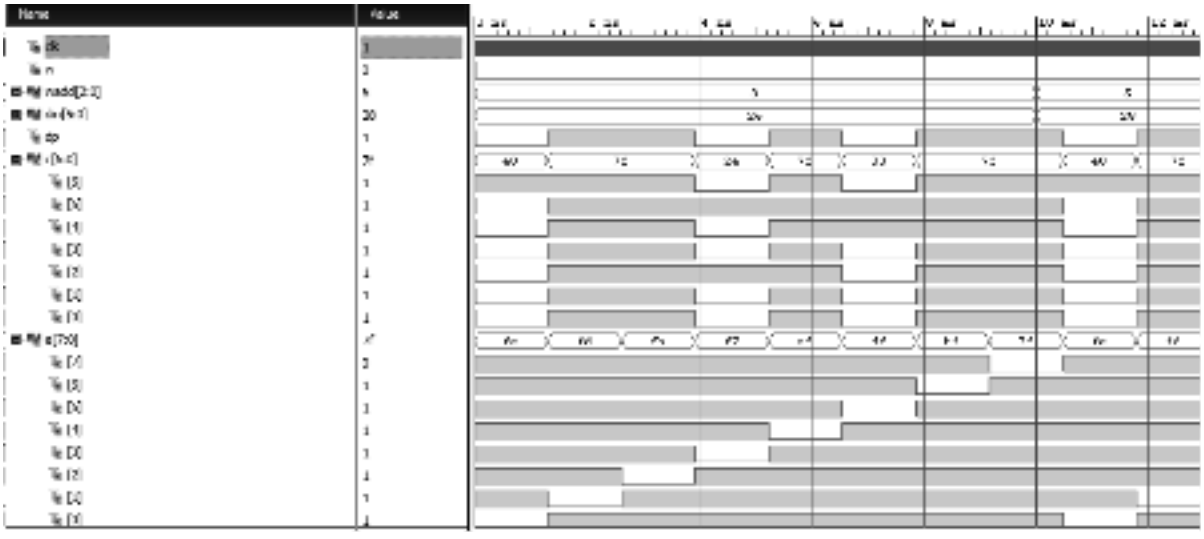
```
// 3x8 Decoder - load DIN to a register based on WADD
always@(posedge clk)
begin
if(EN)
        case(WADD)
        0: D0 <= DIN;
        1: D1 <= DIN;
        2: D2 <= DIN;
        3: D3 <= DIN;
        4: D4 <= DIN;
        5: D5 <= DIN;
        6: D6 <= DIN;
        7: D7 <= DIN;
        endcase
end

// 3x8 Encoder - assign DOUT based on select bits 'S'
always@*
begin
case(S)
        0: DOUT <= D0;
        1: DOUT <= D1;
        2: DOUT <= D2;
        3: DOUT <= D3;
        4: DOUT <= D4;
        5: DOUT <= D5;
        6: DOUT <= D6;
        7: DOUT <= D7;
        default: DOUT <= 0;
endcase
end

// 3x8 Decoder - enable the 7-seg display based on the select
always@*
case(S)
        0: E = 8'b1111_1110;
        1: E = 8'b1111_1101;
        2: E = 8'b1111_1011;
        3: E = 8'b1111_0111;
```

```
            4: E = 8'b1110_1111;
            5: E = 8'b1101_1111;
            6: E = 8'b1011_1111;
            7: E = 8'b0111_1111;
            default: E = 8'b1111_1111;
    endcase

    // Binary to 7-seg decoder (A to G)
    assign DP = (DOUT[0] == 0);

    always@(*)
    begin
    if(DOUT[5])
            case(DOUT[4:1])
            0: CA_G <= 7'b100_0000;
            1: CA_G <= 7'b111_1001;
            2: CA_G <= 7'b010_0100;
            3: CA_G <= 7'b011_0000;
            4: CA_G <= 7'b001_1001;
            5: CA_G <= 7'b001_0010;
            6: CA_G <= 7'b000_0010;
            7: CA_G <= 7'b111_1000;
            8: CA_G <= 7'b000_0000;
            9: CA_G <= 7'b001_1000;
            10: CA_G <= 7'b000_1000;
            11: CA_G <= 7'b000_0011;
            12: CA_G <= 7'b100_0110;
            13: CA_G <= 7'b010_0001;
            14: CA_G <= 7'b000_0110;
            15: CA_G <= 7'b000_1110;
            default: CA_G <= 7'b111_1111;
            endcase
    else
            CA_G <= 7'b111_1111; // hold current state
    end
endmodule
```

Test Fixture:

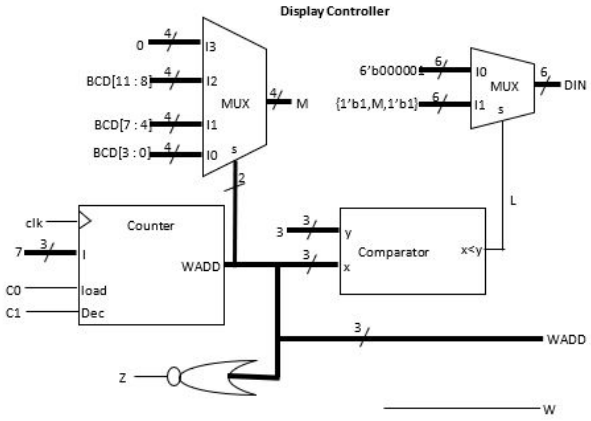
```
module Sim_DisplayInterface();
    reg clk, w;
    reg[2:0] wadd;
    reg[5:0] din;
    wire dp;
    wire[6:0] c;
    wire[7:0] e;
    DisplayInterface uut(clk, w, wadd, din, e, c, dp);
    initial begin
    clk = 0;
    forever# 0.5 clk = ~clk;
    end
    initial begin
    w = 1; wadd = 0; din = 6'b100001;
    #1 w = 1; wadd = 1; din = 6'b1000011;
    #1 w = 1; wadd = 3; din = 6'b100101;
    #1 w = 1; wadd = 5; din = 6'b100111;
    #1 w = 0; wadd = 7; din = 6'b100010;
    #1 w = 0; wadd = 1; din = 6'b100100;
    #1 w = 0; wadd = 3; din = 6'b100110;
    #(1000000) w = 0; wadd = 5; din = 6'b101000;
    #(1000000) $finish;
    end
endmodule
```

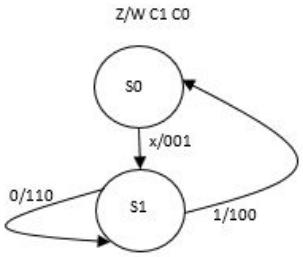
Results:



Display Controller:

Schematic & State Machine:





- Note: The final schematic was modified to hold a 32 bit BCD value rather than a 12 bit BCD value in order to accommodate both the frequency and period meter.

Hardware Description Code:

```
module DisplayController(input clk, input[11:0] BCD, output reg[5:0] DIN,
output reg[2:0] WADD, output reg w);
    // Implementation of state machine diagram
    localparam State0 = 0, State1 = 1;
    reg w, ps = State0, ns, z,c1,c0;

    always @(*)
    begin
            if (WADD == 0)
            z = 1;
            else
            z = 0;
    end
```

```
always @(posedge clk)
        ps <= ns;

always@(*)
begin
case (ps)
        State0:
        begin
        ns = State1;
        w = 0; c1 = 0; c0 = 1;
        end
        State1:
        begin
        if(z)
                begin
                ns = State0;
                w = 1; c1 = 0; c0 = 0;
                end

        else
                begin
                ns = State1;
                w = 1; c1 = 1; c0 = 0;
                end
        end
        default:
        begin
        ns = State0;
        w = 0; c1 = 0; c0 = 0;
        end
endcase
end
```

```
// Counter
always @(posedge clk)
begin
        if (c0)
        WADD = 7; // load the counter
        else if (c1)
        WADD = WADD - 1; // decrement the counter
        else
        WADD = WADD; // default case
end

// multiplexer for BCD values
reg[3:0] mux;
always@(*)
begin
        case (WADD [1:0])
        0: mux = BCD[3:0];
        1: mux = BCD[7:4];
        2: mux = BCD[11:8];
        3: mux = 0;
        default mux = 0;
        endcase
end

// comparator
reg s; // select for 2:1 multiplexer
always@(*)
begin
        if(WADD < 3)
        s = 1;
        else
        s = 0;
end
```

```
        // 2:1 Multiplexer, determines DIN
        always @(*)
        begin
                case (s)
                0: DIN = 1;
                1: DIN = {1'b1, mux, 1'b1};
                default: DIN=0;
                endcase
        end
    endmodule
```

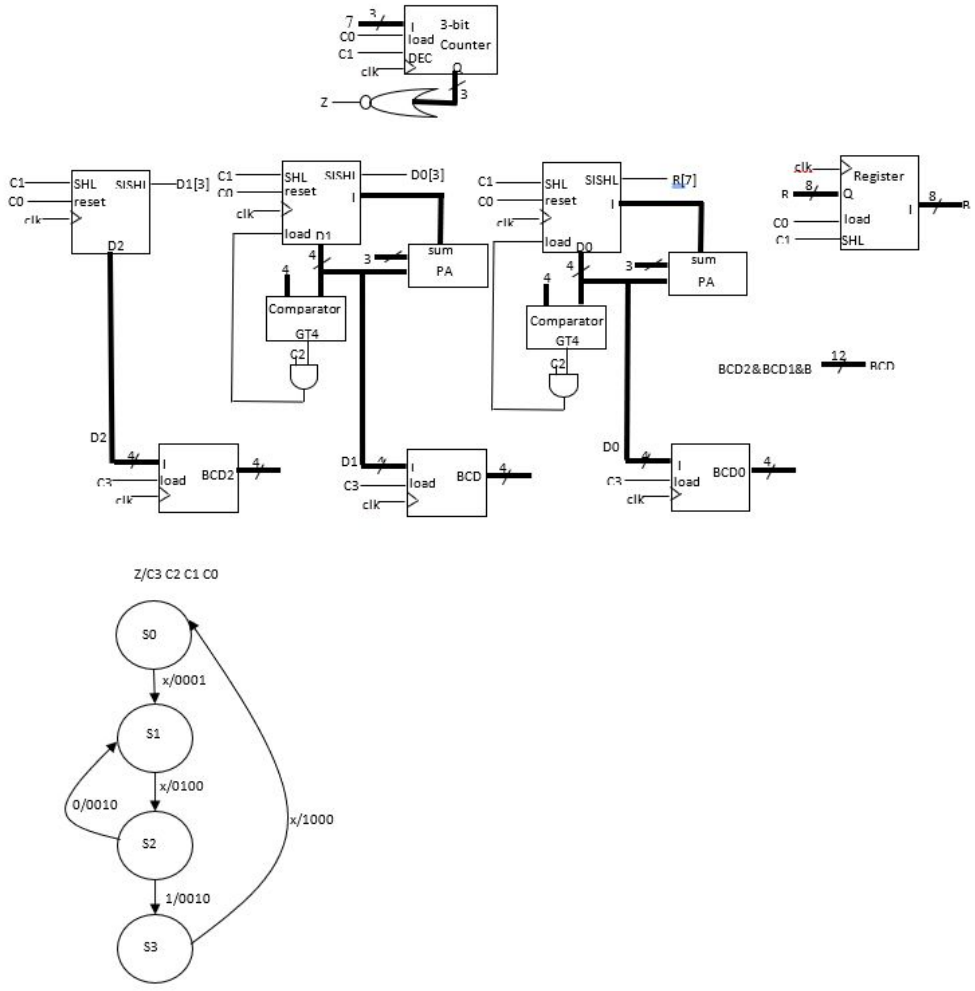
Test Fixture:

```
    module Sim_Display_Controller();
        reg CLK;
        reg [11:0] BCD;
        wire [5:0]DIN;
        wire [2:0] WADD;
        wire w;

        DisplayController uut(CLK, BCD, DIN, WADD, w);

        initial begin
        CLK = 0;
        forever#0.5 CLK = ~CLK;
        end

        initial begin
        BCD = 12'b000000000000;
        #1 BCD = 12'b000100100011;
        #1 BCD = 12'b010101000111;
        #1 BCD = 12'b100110011001;
        #1 BCD =12'b000000000000;
        #1 $finish;
        end
    endmodule
```

Results:

B2BCD:

Schematic & State Machine:



- Note: The B2BCD module was modified to convert a 32 bit binary value into a 32 bit BCD value to accommodate both the frequency and period meter.

Hardware Description Code:

```
module B2BCD(input CLK, input[7:0] B, output[11:0] BCD);
    reg GT4_0, GT4_1;
    reg[2:0] CNT;
    reg[3:0] C, BCD0, BCD1, BCD2, D2, D1, D0;
    reg[7:0] R;

    // BCD Output
    assign BCD = {BCD2, BCD1, BCD0};

    //counter
    always@(posedge CLK)
    if(C[0])
        CNT <= 7;
    else if(C[1])
        CNT <= CNT - 1;

    // Finite State Mealy Machine - determines C[3:0] based on z (counter[0])
    assign Z = ~|CNT;
    localparam [1:0] S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
    reg[1:0] ps = 0, ns;

    always @(posedge CLK)
        ps <= ns;

    always@*
    begin
    case (ps)
        S0:
        begin
            C <= 4'b0001;
            ns <= S1;
        end
```

```
S1:
            begin
                C <= 4'b0100;
                ns <= S2;
            end
            S2:
            begin
                C <= 4'b0010;
                if (Z == 1)
                    ns <= S3;
                else
                    ns <= S1;
            end
            S3:
            begin
                C <= 4'b1000;
                ns <= S0;
            end
            default:
            begin
                C <= 4'b0001;
                ns <= S0;
            end
    endcase
    end

    // Register R
    always@(posedge CLK)
            if(C[0])
                R <= B;
            else if(C[1])
            R <= {R[6], R[5], R[4], R[3], R[2], R[1], R[0], 1'b0};
```

```
// register D0
reg[3:0] D0plus3;
always@(posedge CLK)
if(C[0])
        D0 <= 0;
else if(C[1])
        D0 <= {D0[2], D0[1], D0[0], R[7]};
else if(C[2] & GT4_0)
        D0 <= D0plus3;

// SISHL 2 - register D1
reg[3:0] D1plus3;
always@(posedge CLK)
        if(C[0])
                D1 <= 0;
        else if(C[1])
                D1 <= {D1[2], D1[1], D1[0], D0[3]};
        else if(C[2] & GT4_1)
                D1 <= D1plus3;

// SISHL 3 - Register D2
always@(posedge CLK)
        if(C[0])
                D2 <= 0;
        else if(C[1])
                D2 <= {D2[2], D2[1], D2[0], D1[3]};

// Adder 0
always@(*)
        D0plus3 = D0 + 3;
```

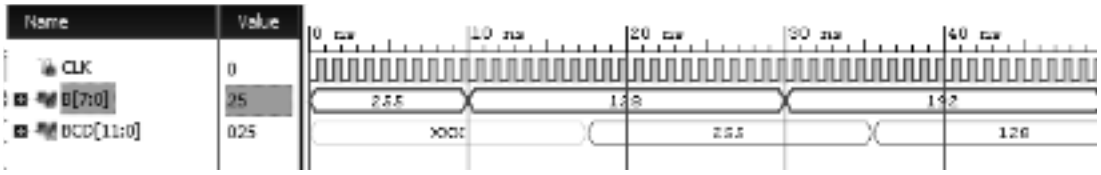
```
        // Adder 1
        always@(*)
                D1plus3 = D1 + 3;

        // Comparator 0
        always@(*)
        if(D0 > 4)
                GT4_0 = 1;
        else
                GT4_0 = 0;

        // Comparator 1
        always@(*)
                if(D1 > 4)
                        GT4_1 = 1;
                else
                        GT4_1 = 0;

        // Load registers BCD2:BCD0 when C[3] = 1
        always@(posedge CLK)
                if(C[3])
                        begin
                                BCD0 <= D0;
                                BCD1 <= D1;
                                BCD2 <= D2;
                        end
    endmodule
```

Test Fixture:

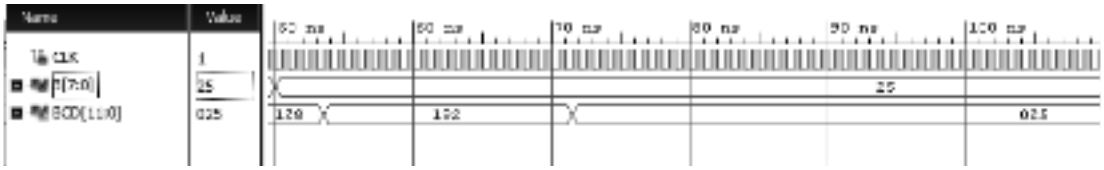
```
    module Sim_B2BCD();
        reg CLK;
        reg[7:0] B;
        wire[11:0] BCD;

        B2BCD uut(CLK, B, BCD);
```

```
        initial begin
                CLK = 0;
                forever#0.5 CLK = ~CLK;
        end

        initial begin
                B = 255;
                #10 B = 128;
                #20 B = 192;
                #20 B = 25;
                #10 $finish;
        end
    endmodule
```

Results:



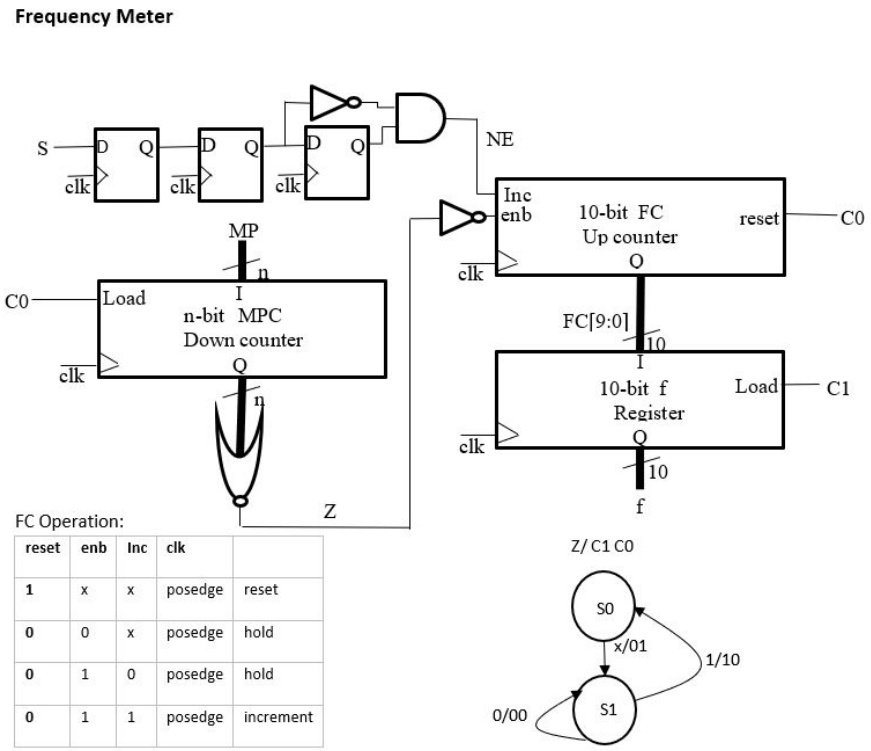
- 0 - 40 ns



- 50 - 100 ns

Frequency Meter:

Schematic & State Machine:



FC Operation:

reset	enb	Inc	clk	
1	x	x	posedge	reset
0	0	x	posedge	hold
0	1	0	posedge	hold
0	1	1	posedge	increment

Hardware Description Code:

```
module FrequencyMeter(input S, input CLK, output reg[9:0] B = 0);
    reg Q1 = 0, Q2 = 0, Q3 = 0, C0, C1, Z;
    wire i;
    reg[16:0] MPC;
    reg[9:0] FC;
    localparam MP = 100000;

    // n-bit MPC Down Counter
    always@(posedge CLK)
    if(C0 == 1)
            MPC <= MP;
    else
            MPC <= MPC - 1;
```

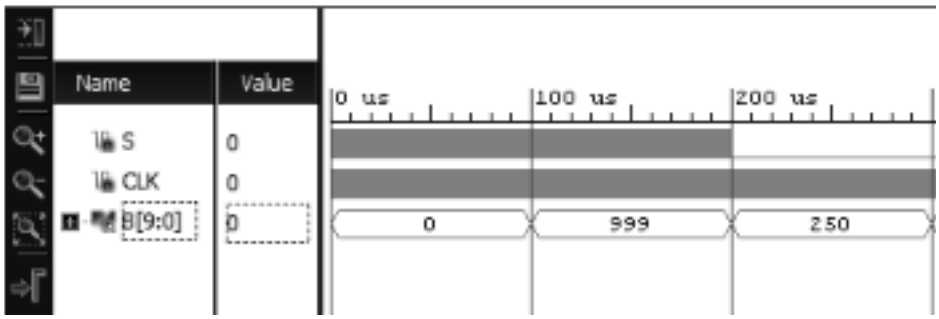
```
always@(*)
begin
if (MPC == 0)
        Z = 1;
else
        Z = 0;
end

// State Machine
localparam S0=1'b0, S1=1'b1;
reg ps = 0, ns;
always @(posedge CLK)
        ps <= ns;
always@(*)
begin
        case(ps)
        S0:
            begin
                ns = S1;
                C1 = 0;
                C0 = 1;
             end
        S1:
           begin
                if(Z == 1)
               begin
                ns = S0;
                C1 = 1;
                C0 = 0;
               end
                else
                begin
                ns = S1;
                C1 = 0;
                C0 = 0;
                end
           end
```

```
            default:
                begin
                    ns = S0;
                    C1 = 0;
                    C0 = 0;
                end
        endcase
end
// 3 Flip Flop Synchronizer
always@(posedge CLK)
begin
            Q1 <= S;
            Q2 <= Q1;
            Q3 <= Q2;
end
assign i = (Q3 & ~Q2);

always @(posedge CLK)
begin
if (C0)
            FC <= 0;
else if (C0==0 && Z ==1)
            FC <= FC;
else if (C0==0 && Z ==0 && i == 0 )
            FC <= FC;
else if (C0==0 && Z ==0 && i == 1 )
            FC <= FC + 1;
end

// 10-Bit F Register
always@(posedge CLK)
if(C1 == 1)
            B <= FC;
else
            B <= B;
endmodule
```

Test Fixture:

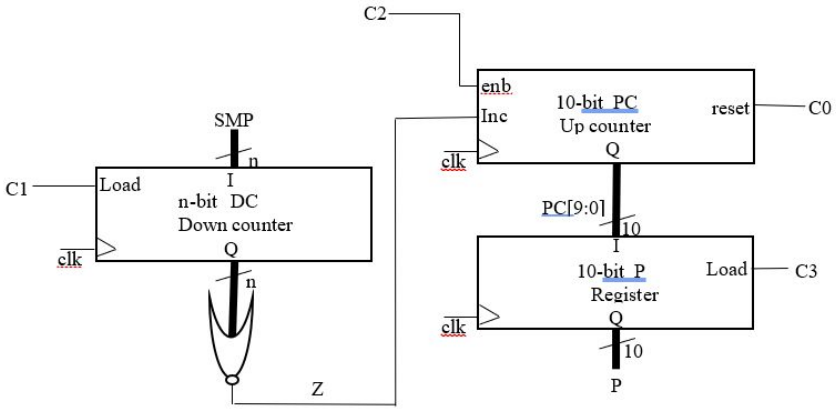
```
module Sim_FrequencyMeter();
    reg S,CLK;
    wire[9:0] B;

    FrequencyMeter uut(S,CLK,B);

    initial begin
        CLK = 0;
        forever #0.5 CLK = ~CLK;
    end

    initial begin
        S = 0;
        repeat(1998)#(50) S = ~S; // Testing 999 kHz - max frequency
        repeat(500)#(200) S = ~S; // Testing 250 kHz
        #(10000)$finish;
    end
endmodule
```

Results:

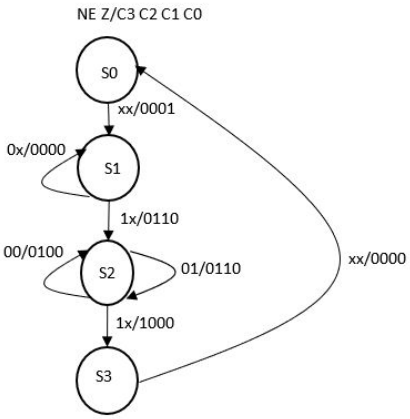


Period Meter:

Schematic & State Machine:



reset	enb	Inc	clk	
1	x	x	posedge	reset
0	0	x	posedge	hold
0	1	0	posedge	hold
0	1	1	posedge	increment



Hardware Description Code:

```
module PeriodMeter(input CLK, input NE, output reg[9:0] P = 0);
    reg Z;
    reg[3:0] C;      // C3 - C0
    reg[6:0] SMP_CNT;
    reg[9:0] PC;
    localparam SMP = 100;

    // n-bit MPC Down Counter
    always@(posedge CLK)
        if(C[1] == 1)
            SMP_CNT <= SMP;
        else
            SMP_CNT <= SMP_CNT - 1;
```

```
always@(*)
begin
    if (SMP_CNT == 0)
        Z = 1;
    else
        Z = 0;
end

// State Machine
localparam[1:0] S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3= 2'b11;
reg[1:0] ps = 0, ns;

always @(posedge CLK)
ps <= ns;

always@(*)
begin
        case(ps)
        S0:
        begin
                ns = S1;
                C = 4'b0001;
        end
        S1:
        begin
                if(NE == 1)
                begin
                ns = S2;
                C = 4'b0110;
                end
                else
                begin
                ns = S1;
                C = 4'b0010;
                end
        end
```

```
            S2:
                begin
                    if(NE == 1)
                    begin
                    ns = S3;
                    C = 4'b1000;
                    end
                    else if(NE == 0 && Z == 0)
                    begin
                    ns = S2;
                    C = 4'b0100;
                    end
                    else if(NE == 0 && Z == 1)
                    begin
                    ns = S2;
                    C = 4'b0110;
                    end
            else
                    begin
                    ns = S2;
                    C = 4'b0000;
                    end
            end
            S3:
            begin
                    ns = S0;
                    C = 4'b0000;
            end
            default:
            begin
                    ns = S0;
                    C = 4'b0000;
            end
            endcase
end
```

```
always @(posedge CLK)
begin
        if (C[0] == 1)
        PC <= 0;
                else if (C[0] == 0 && C[2] == 0)
                PC <= PC;
                else if (C[0] == 0 && C[2] == 1 && Z == 0)
                PC <= PC;
                else if (C[0] == 0 && C[2] == 1 && Z == 1)
        PC <= PC + 1;
end

// 10-Bit P Register
always@(posedge CLK)
        if(C[3] == 1)
                        P <= PC;
        else
                        P <= P;

endmodule
```

Test Fixture:

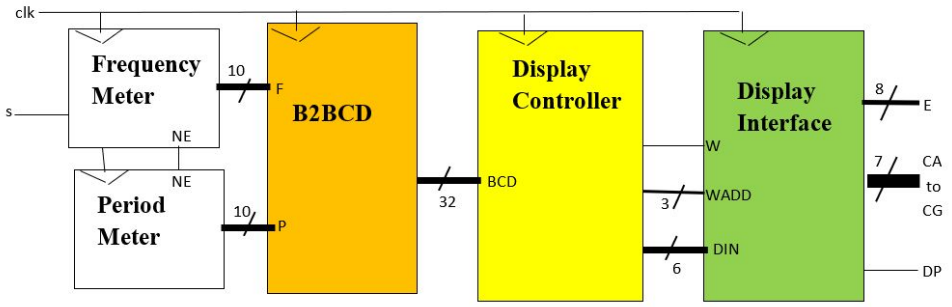
```
module TestBench();
    reg CLK, S;
    wire [9:0] F;
    wire [9:0] P;

    Frequency_Period_Call uut(CLK, S, F, P);

    initial begin
    CLK = 0;
    forever #0.5 CLK = ~CLK;
    end

    initial begin
    S=0;
    repeat(2000)#(50) S = ~S;
    repeat(100)#(1000) S = ~S;
    repeat(8)#(50000) S = ~S;
```

```
        #(100000)$finish;
        end
    endmodule
```

Results:

Driver Module:

Schematic:



Hardware Description Code:

```
module SystemCall(input S, CLK, output[7:0] E, output[6:0] CA_G, output DP);
    wire W, NE;
    wire [2:0] WADD;
    wire [5:0] DIN;
    wire [31:0] BCD;
    wire [9:0] B;
    wire [9:0] P;

    DisplayInterface mDI (CLK, W, WADD, DIN, E, CA_G, DP);
    // Calling modified DisplayController and B2BCD, therefore there are more
    // parameters in the call than there are in the included hardware description code
```

```
        DisplayController mDC (CLK, BCD, DIN, WADD, W);
        B2BCD mB2BCD (CLK, F, P, BCD);
        FrequencyMeter mFM (CLK, S, NE, F);
        PeriodMeter mPM (CLK, NE, P);
    endmodule
```

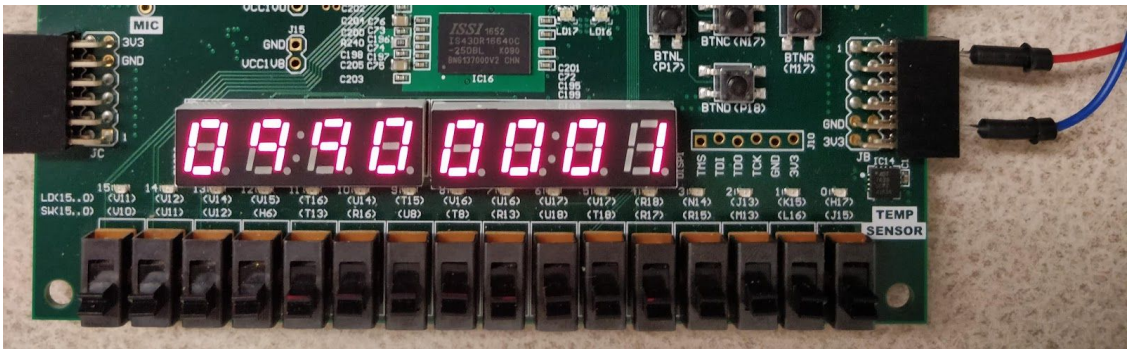
Testing Procedure:

General Procedure:

In order to test our implementation of the system we used a frequency generator to generate different frequencies, which were inputted into the FPGA. This was done by setting the frequency generator in high impedance (high z) mode and setting the low and high amplitude of the voltage to 0v and 3.5v respectively. Shown below are sample images of the data taken, and other data points were observed, recorded, and eventually graphed as shown in the section below.

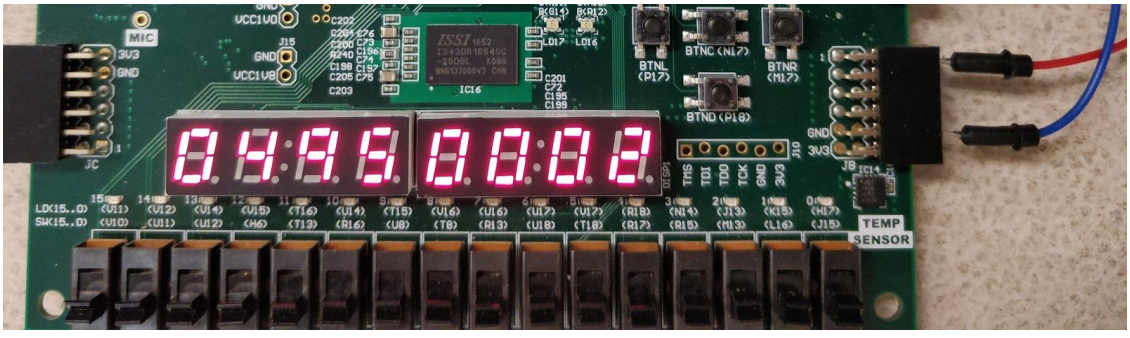
Sample Test Cases:

Sample 1:



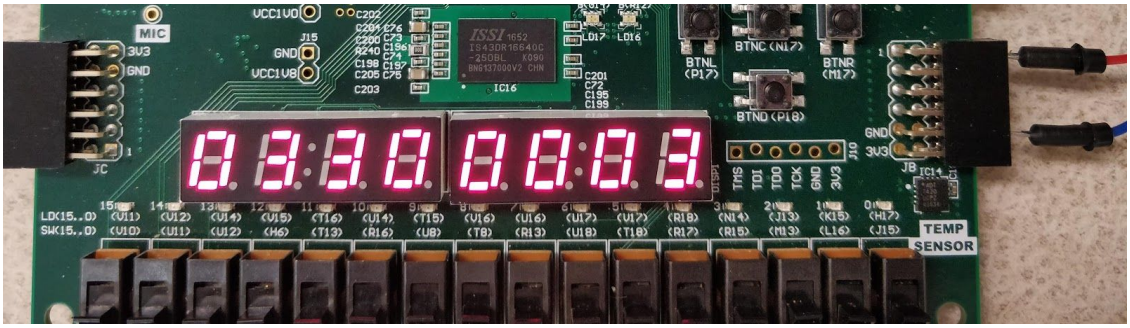
Period = 990 μs, Frequency = 1 KHz

Sample 2:



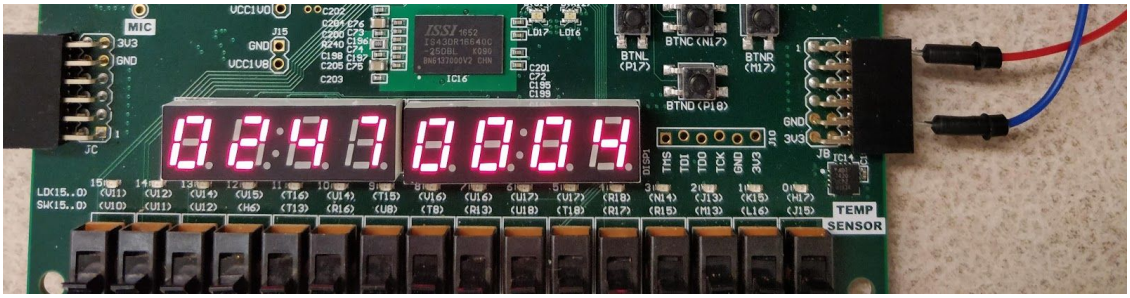
Period = 495 μs, Frequency = 2 KHz

Sample 3:



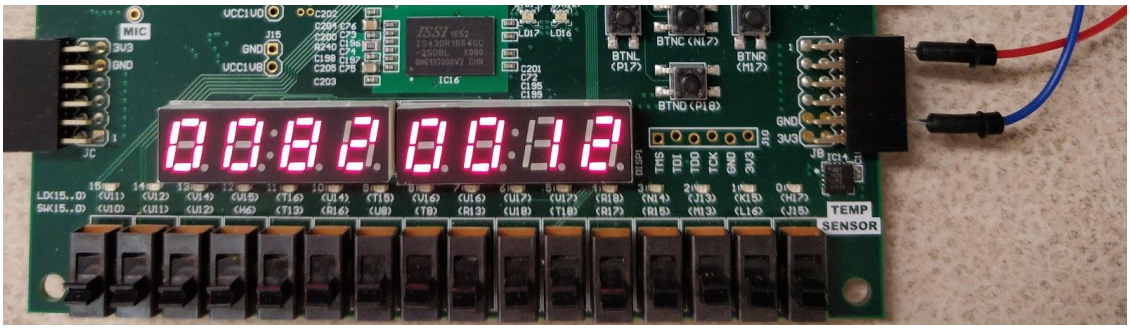
Period = 330 µs, Frequency = 3 KHz

Sample 4:



Period = 247 µs, Frequency = 4 KHz

Sample 5:



Period = 82 µs, Frequency = 12 KHz

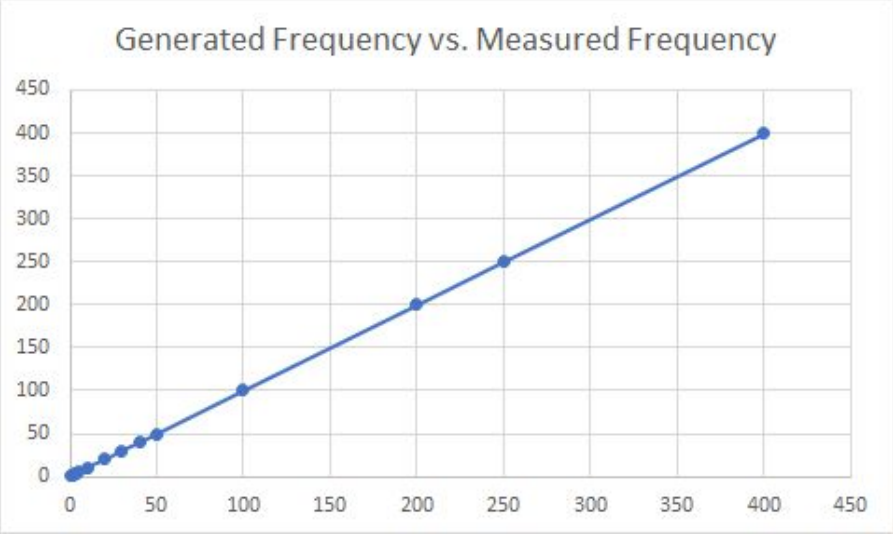
Results / Analysis:

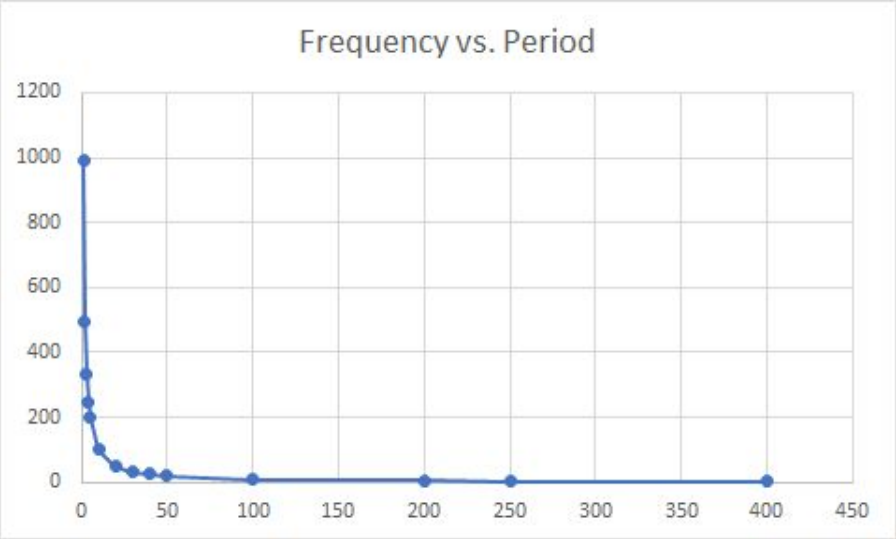
Results:

Data Table:

Generated Frequency	F	P
1	1	990
2	2	495
3	3	330
4	4	247
5	5	198
10	10	99
20	20	49
30	30	32
40	40	24
50	50	19
100	100	9
200	200	4
250	250	3
400	400	2

Graphs:





Analysis:

After testing the system the group was able to verify that the system worked as intended. As mentioned in the testing procedure section of this report a collection of points were collected and plotted in excel, producing two graphs. The first graph proves that our design successfully measures a given frequency between 1 and 999 kHz because the frequency generated by oscilloscope was identical to the frequency read by the FPGA. The second graph shows the relationship between our measured frequency, and the period that was displayed. The graph shows a trend that frequency is described as 1 / period, which matches the theoretical relationship between frequency and period.

Conclusions:

The experiment consisted of designing, testing, and implementing a 5 module system to measure a given frequency as well as its period and verifying our results using an FPGA as well as the Vivado simulator. Each module was simulated individually, and every module worked as intended. This was further verified when testing our system on the FPGA itself, which property measured the desired values. The system performed completely as expected, and no incorrect results were produced regardless of if it was tested with the simulator or the FPGA.

Suggestions:

No suggestions.