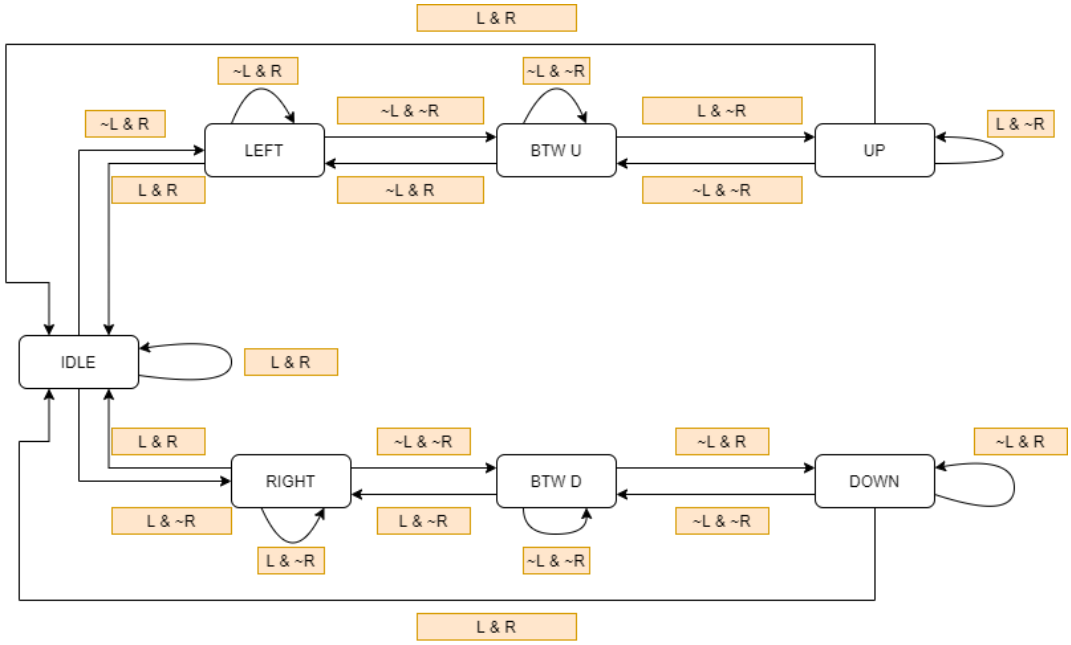
Lab 6 Report

Introduction: In this lab we were tasked with creating a turkey counter system that would use sensors to detect a turkey passing in front of the system. For our lab we use buttons to simulate such sensors. We have to count both up and down, so if a turkey goes left to right then we increment. If the turkey goes right to left then we decrement. Overall, a pretty simple task that was interesting to implement.

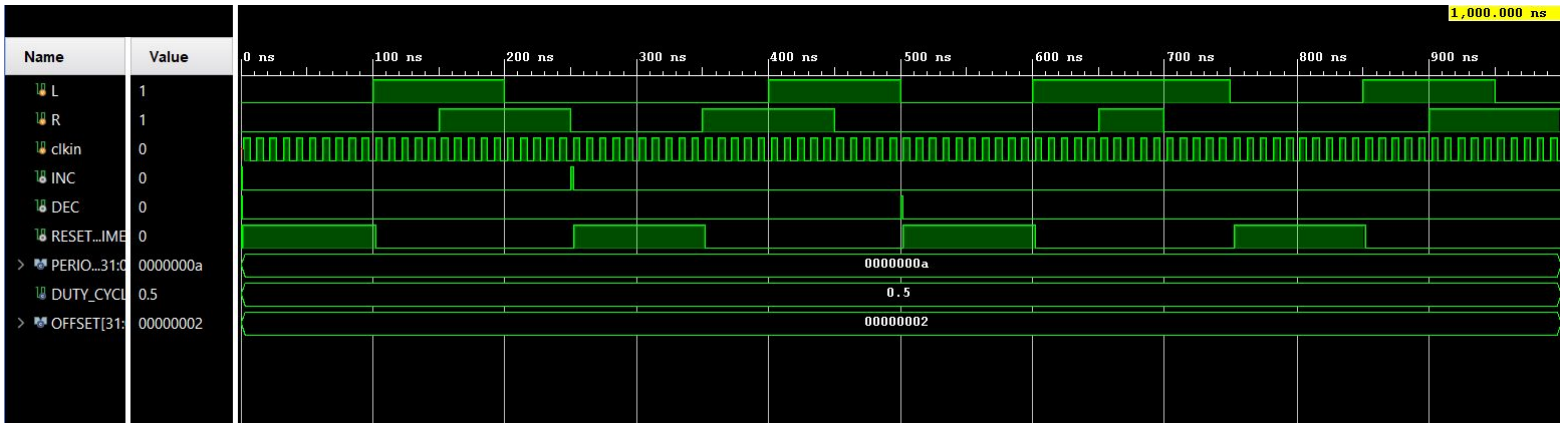
To start I created a basic state machine diagram which I used to create a state table. This was very similar to lab 5 and thus was much easier since I was more familiar with the process. After I used one hot encoding to create the logic for each of the flip flops my state machine would require I coded it in verilog. After that I created a testbench and tested to make sure it was accurate for all the relevant cases.



In my diagram I conceptualized the machine to have two paths, one if the turkey was coming from the right and one if it was coming from the left. In the left path it would increment if it reaches the end, and for the right path it would decrement. However, going down both of these paths does not need to end in an increment or decrement, both can simply end in nothing occurring at all. This was simpler than my previous idea of having the counter increment then decrement again if nothing was supposed to change. The way I have outlined in my diagram saves computations and resources.

State Machine Description: The Idle state is the initial state where it decides where the turkey is coming from, left or right. The RIGHT state is if the right sensor is blocked and leads to the BTWD state where both are blocked. This leads to the DOWN state is the turkey only blocks the left sensor and then decrements if the turkey leaves that sensor. The LEFT path is the exact same except it will increment if the turkey leaves the right sensors meaning it went left to right. Each state leads back to the previous state to stop unnecessary computations.

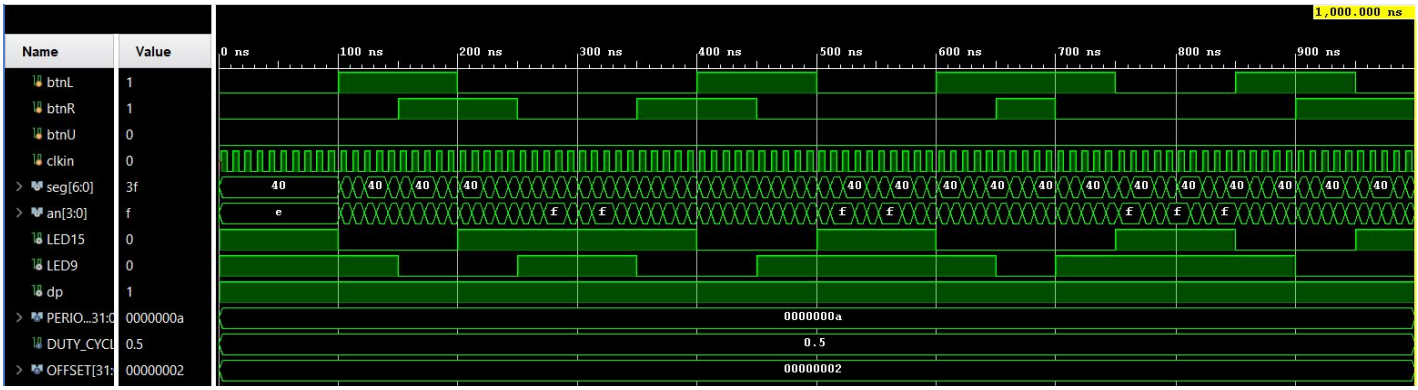
	NS												NS					
	BTN-L/BTN-R					Q0	Q1	Q2	Q3	Q4	Q5	Q6	BTN-L / BTN-R					
States	00	01	10	11									00	01	10	11		
IDLE	IDLE	LEFT	RIGHT			1							Q0	Q1	Q2			
Left	IDLE	LEFT		BTWU			1						Q0	Q1		Q3		
Right	IDLE		RIGHT	BTWD				1					Q0		Q2	Q4		
BTW U		LEFT		BTWU					1					Q1		Q3		
BTW D			RIGHT	BTWD						1					Q2	Q4		
UP	IDLE			BTWU							1		Q0		Q5	Q3		
Down	IDLE	DOWN		BTWD								1	Q0	Q6		Q4		
D0 = (!btnR & !btnL) & (Q0 + Q1 + Q2 + Q5 + Q6)																		
D1 = (btnR & !btnL) & (Q0 + Q1 + Q3 + Q6)																		
D2 = (!btnR & btnL) & (Q0 + Q2 + Q4 + Q5)																		
D3 = (Q1 & btnR & btnL)																		
D4 = (Q2 & btnR & btnL)																		



We can see in the waveform how increment and decrement when the turkey fully passes either left to right or right to left. If one of the sensors is still blocked it will not increment or decrement.

After I had this completed I started on the modules that either fed into the state machine or took its outputs. Most of these I was able to copy from other labs. The time counter I took from the last lab, except I shifted the bits by 2 in order to divide the time by 4. This made it so that the time counted up by 1 second because the qsec input was once every 1/4th of a second. For the turkey counter module I copied the 4bit count module from lab 4. I chose to do this because I needed both up and down counter functionality. Once I had this I copied the selector, hex7seg, and ring counter for the display. Next I started on the top module. However, as I was creating my top module and testing it I realized that I would need a way to invert my negative values since I used two's complement to represent the negative values. I did this by using the 8th bit of the turkey counter and inverting the rest of the bits when the 8th bit was 1. Once I had this I also had to hard code the negative sign for the display. After this I tested it thoroughly until I was satisfied.

Supplementary Materials:



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/17/2020 04:08:46 PM
// Design Name:
// Module Name: top_module
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module top_module(
    input btnL,
    input btnR,
    input btnU,
    input clkin,

    output [6:0] seg,
    output [3:0] an,
    output LED15,
    output LED9,
    output dp
    );
    assign dp = 1'b1;
```

```
    assign LED15 = ~btnL;
    assign LED9 = ~btnR;
    wire clk,digsel,qsec;
    wire Up,Down,ResetTimer;
    wire [7:0] turkeyCount;
    wire [7:0] timeCount;
    wire [3:0] ringCount;
    wire [15:0] selIn;
    wire [3:0] selOut;
    wire [6:0] mux;
    wire [7:0] invOut;
    lab6_clks slowit (.clkin(clkin), .greset(btnU), .clk(clk),
.digsel(digsel), .qsec(qsec));
    statemachine_module fsm(.clk(clkin), .btnL(btnL), .btnR(btnR),
.Up(Up),.Down(Down),.ResetTimer(ResetTimer));

    turkeyCounter_module turkeyC(.clk(clkin), .Up(Up), .Down(Down),
.Q(turkeyCount[7:0]));
    time_counter_module timeC(.clk(clkin), .R(ResetTimer),
.CE(qsec), .Q(timeCount[7:0]));

    neg_inverter_module neg(.in(turkeyCount[7:0]),
.invOut(invOut[7:0]));
    assign selIn[6:0] = invOut[6:0];
    assign selIn[7] = 1'b1;
    assign selIn[11:8] = 4'b0000;
    assign selIn[15:12] = timeCount[3:0];
    ring_counter_module ringC(.clk(clkin), .Advance(digsel),
.Q(ringCount[3:0]));
    selector selec(.sel(ringCount[3:0]), .N(selIn[15:0]),
.H(selOut[3:0]));
    hex7seg hex(.n(selOut[3:0]), .seg(mux[6:0]));

    assign seg[0] = (mux[0] & ~ringCount[2]) | (ringCount[2] &
1'b1);
    assign seg[1] = (mux[1] & ~ringCount[2]) | (ringCount[2] &
1'b1);
```

```
    assign seg[2] = (mux[2] & ~ringCount[2]) | (ringCount[2] &
1'b1);
    assign seg[3] = (mux[3] & ~ringCount[2]) | (ringCount[2] &
1'b1);
    assign seg[4] = (mux[4] & ~ringCount[2]) | (ringCount[2] &
1'b1);
    assign seg[5] = (mux[5] & ~ringCount[2]) | (ringCount[2] &
1'b1);
    assign seg[6] = (mux[6] & ~ringCount[2]) | (ringCount[2] &
1'b0);

    assign an[0] = ~(ringCount[0] & ~ringCount[1]  & ~ringCount[2]
& ~ringCount[3] );
    assign an[1] = ~(~ringCount[0] & ringCount[1]  & ~ringCount[2]
& ~ringCount[3] );
    assign an[2] = ~(~ringCount[0] & ~ringCount[1] & ringCount[2] &
~ringCount[3] & turkeyCount[7]);
    assign an[3] = ~(~ringCount[0] & ~ringCount[1] & ~ringCount[2]
& ringCount[3] & ~ResetTimer);
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/17/2020 01:32:43 PM
// Design Name:
// Module Name: statemachine_module
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module statemachine_module(
    input btnL,
    input btnR,
    input clk,

    output Up,
    output Down,
    output ResetTimer
    );

    wire [6:0] D;
    wire [6:0] Q;
```

```
    FDRE #(.INIT(1'b1) ) IDLE_FF  (.C(clk), .CE(1'b1), .D(D[0]),
.Q(Q[0]));
    FDRE #(.INIT(1'b0) ) LEFT_FF  (.C(clk), .CE(1'b1), .D(D[1]),
.Q(Q[1]));
    FDRE #(.INIT(1'b0) ) RIGHT_FF (.C(clk), .CE(1'b1), .D(D[2]),
.Q(Q[2]));
    FDRE #(.INIT(1'b0) ) BTWU_FF  (.C(clk), .CE(1'b1), .D(D[3]),
.Q(Q[3]));
    FDRE #(.INIT(1'b0) ) BTWD_FF  (.C(clk), .CE(1'b1), .D(D[4]),
.Q(Q[4]));
    FDRE #(.INIT(1'b0) ) UP_FF    (.C(clk), .CE(1'b1), .D(D[5]),
.Q(Q[5]));
    FDRE #(.INIT(1'b0) ) DOWN_FF  (.C(clk), .CE(1'b1), .D(D[6]),
.Q(Q[6]));

    assign D[0] = (~btnL & ~btnR) & (Q[0] | Q[1] | Q[2] | Q[5] |
Q[6]);
    assign D[1] = (btnL & ~btnR) & (Q[0] | Q[1] | Q[3]);
    assign D[2] = (~btnL & btnR) & (Q[0] | Q[2] | Q[4]);
    assign D[3] = (btnL & btnR) & (Q[1] | Q[3] | Q[5]);
    assign D[4] = (btnL & btnR) & (Q[2] | Q[4] | Q[6]);
    assign D[5] = (~btnL & btnR) & (Q[3] | Q[5]);
    assign D[6] = (btnL & ~btnR) & (Q[4] | Q[6]);

    assign ResetTimer = (Q[0]);
    assign Up = (Q[5] & ~btnL & ~btnR);
    assign Down = (Q[6] & ~btnL & ~btnR);
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/17/2020 02:49:51 PM
// Design Name:
// Module Name: turkeyCounter_module
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module turkeyCounter_module(
    input clk,
    input Up,
    input Down,

    output [7:0] Q
);
    wire UTC1,DTC1;
    count4bit count1(.clk(clk), .Up(Up), .Dw(Down), .Q(Q[3:0]),
.DTC(DTC1), .UTC(UTC1));
    count4bit count2(.clk(clk), .Up(Up & UTC1), .Dw(Down & DTC1),
.Q(Q[7:4]));
```

```
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/11/2020 03:36:02 PM
// Design Name:
// Module Name: time_counter_module
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module time_counter_module(
    input clk,
    input R,
    input CE,       //this is set as qsec
    output [7:0] Q
    );
    wire [7:0] D;
    wire [7:0] out;
    wire ce = CE & ~(Q[0] & Q[1] & Q[2] & Q[3]);
    FDRE #(.INIT(1'b0)) Q0_FF (.C(clk), .R(R), .CE(ce), .D(D[0]),
.Q(out[0]));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .R(R), .CE(ce), .D(D[1]),
.Q(out[1]));
```

```
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .R(R), .CE(ce), .D(D[2]),
.Q(out[2]));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .R(R), .CE(ce), .D(D[3]),
.Q(out[3]));
    FDRE #(.INIT(1'b0)) Q4_FF (.C(clk), .R(R), .CE(ce), .D(D[4]),
.Q(out[4]));
    FDRE #(.INIT(1'b0)) Q5_FF (.C(clk), .R(R), .CE(ce), .D(D[5]),
.Q(out[5]));
    FDRE #(.INIT(1'b0)) Q6_FF (.C(clk), .R(R), .CE(ce), .D(D[6]),
.Q(out[6]));
    assign D[0] = out[0] ^ CE;
    assign D[1] = out[1] ^ (CE&out[0]);
    assign D[2] = out[2] ^ (CE&out[0]&out[1]);
    assign D[3] = out[3] ^ (CE&out[0]&out[1]&out[2]);
    assign D[4] = out[4] ^ (CE&out[0]&out[1]&out[2]&out[3]);
    assign D[5] = out[5] ^ (CE&out[0]&out[1]&out[2]&out[3]&out[4]);
    assign D[6] = out[6] ^ (CE&out[0]&out[1]&out[2]&out[3]&out[4]);
    assign Q[0] = out[2];
    assign Q[1] = out[3];
    assign Q[2] = out[4];
    assign Q[3] = out[5];
    assign Q[4] = out[6];
    assign Q[5] = 1'b0;
    assign Q[6] = 1'b0;
    assign Q[7] = 1'b0;


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/17/2020 10:20:16 PM
// Design Name:
// Module Name: neg_inverter_module
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module neg_inverter_module(
    input [7:0] in,

    output [7:0] invOut
    );

    wire [7:0] o;
    wire [7:0] s;
    wire [7:0] c_out;
    assign o[0] = ~in[0] & in[7] | in[0] & ~in[7];
    assign o[1] = ~in[1] & in[7] | in[1] & ~in[7];
    assign o[2] = ~in[2] & in[7] | in[2] & ~in[7];
    assign o[3] = ~in[3] & in[7] | in[3] & ~in[7];
```

```
    assign o[4] = ~in[4] & in[7] | in[4] & ~in[7];
    assign o[5] = ~in[5] & in[7] | in[5] & ~in[7];
    assign o[6] = ~in[6] & in[7] | in[6] & ~in[7];
    assign o[7] = in[7];

    adder add0 (.a(o[0]), .b(in[7]), .c_in(1'b0), .s(s[0]),
.c_out(c_out[0]));
    adder add1 (.a(o[1]), .b(1'b0), .c_in(c_out[0]), .s(s[1]),
.c_out(c_out[1]));
    adder add2 (.a(o[2]), .b(1'b0), .c_in(c_out[1]), .s(s[2]),
.c_out(c_out[2]));
    adder add3 (.a(o[3]), .b(1'b0), .c_in(c_out[2]), .s(s[3]),
.c_out(c_out[3]));
    adder add4 (.a(o[4]), .b(1'b0), .c_in(c_out[3]), .s(s[4]),
.c_out(c_out[4]));
    adder add5 (.a(o[5]), .b(1'b0), .c_in(c_out[4]), .s(s[5]),
.c_out(c_out[5]));
    adder add6 (.a(o[6]), .b(1'b0), .c_in(c_out[5]), .s(s[6]),
.c_out(c_out[6]));
    assign s[7] = in[7];
    assign invOut = s;
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/12/2020 01:09:53 PM
// Design Name:
// Module Name: adder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module adder(
    input a,
    input b,
    input c_in,
    output s,
    output c_out

    );
    assign s = c_in ^ (a^b);
    assign c_out = (a & b) | (b & c_in) | (a & c_in);
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/11/2020 03:54:26 PM
// Design Name:
// Module Name: hex7seg
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



module hex7seg(
    input [3:0] n,
    output [6:0] seg
    );

    m8_1 mSeg0(.in({1'b0,n[0],n[0],1'b0,1'b0,~n[0],1'b0,n[0]}),
.sel({n[3],n[2],n[1]}), .o(seg[0]));             //controls segment
0 for 7 segment display
    m8_1 mSeg1(.in({1'b1,~n[0],n[0],1'b0,~n[0],n[0],1'b0,1'b0}),
.sel({n[3],n[2],n[1]}), .o(seg[1]));            //controls segment
1 for 7 segment display
    m8_1 mSeg2(.in({1'b1,~n[0],1'b0,1'b0,1'b0,1'b0,~n[0],1'b0}),
```

```
.sel({n[3],n[2],n[1]}), .o(seg[2]));                    //controls
segment 2 for 7 segment display
    m8_1 mSeg3(.in({n[0],1'b0,~n[0],n[0],n[0],~n[0],1'b0,n[0]}),
.sel({n[3],n[2],n[1]}), .o(seg[3]));        //controls segment 3
for 7 segment display
    m8_1 mSeg4(.in({1'b0,1'b0,1'b0,n[0],n[0],1'b1,n[0],n[0]}),
.sel({n[3],n[2],n[1]}), .o(seg[4]));              //controls
segment 4 for 7 segment display
    m8_1 mSeg5(.in({1'b0,n[0],1'b0,1'b0,n[0],1'b0,1'b1,n[0]}),
.sel({n[3],n[2],n[1]}), .o(seg[5]));                //controls
segment 5 for 7 segment display
    m8_1 mSeg6(.in({1'b0,~n[0],1'b0,1'b0,n[0],1'b0,1'b0,1'b1}),
.sel({n[3],n[2],n[1]}), .o(seg[6]));                 //controls
segment 6 for 7 segment display

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/11/2020 03:41:08 PM
// Design Name:
// Module Name: ring_counter_module
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module ring_counter_module(
    input clk,
    input Advance,
    output [3:0] Q
    );
    wire [3:0] out;
    FDRE #(.INIT(1'b1)) Q0_FF (.C(clk), .CE(1'b1), .D(out[3]),
.Q(out[0]));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .CE(1'b1), .D(out[0]),
.Q(out[1]));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .CE(1'b1), .D(out[1]),
.Q(out[2]));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .CE(1'b1), .D(out[2]),
```

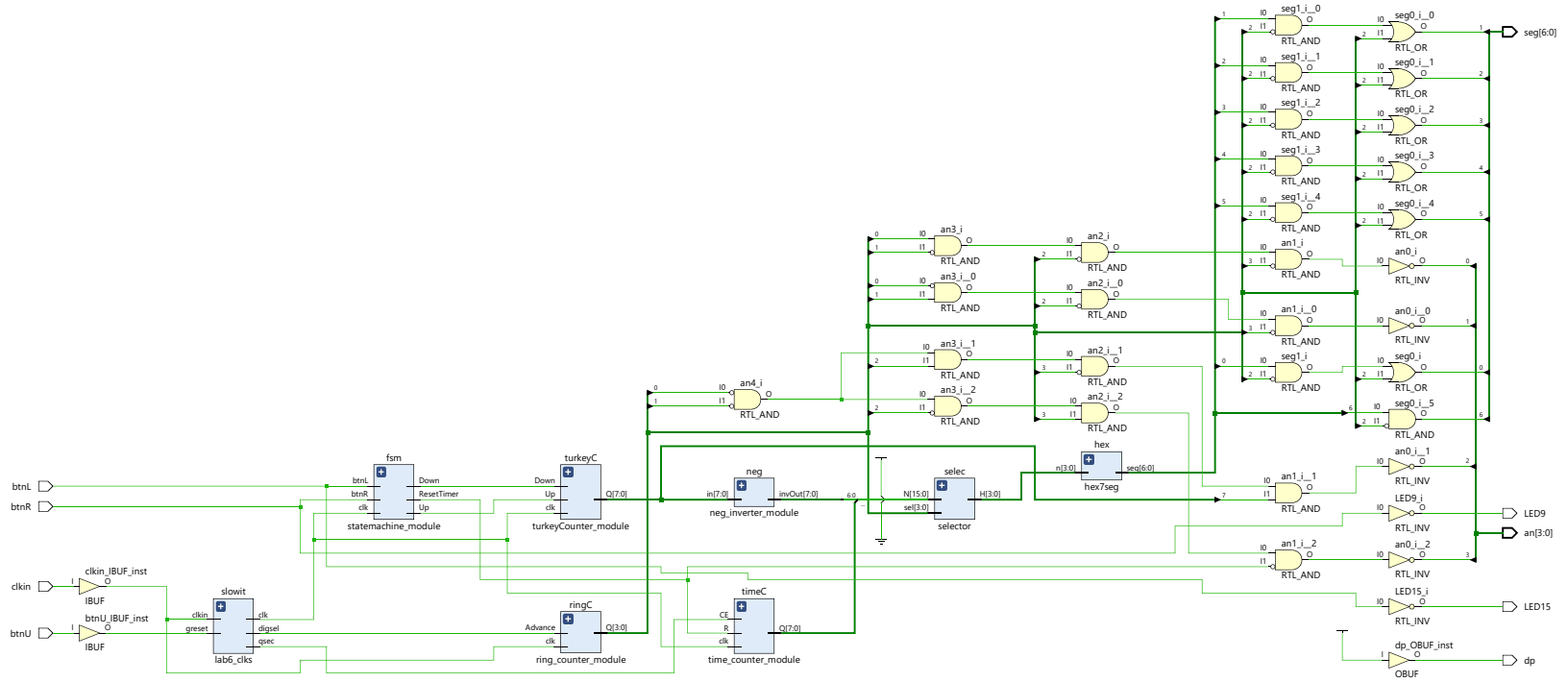
```
.Q(out[3]));

    assign Q = out;
endmodule
```

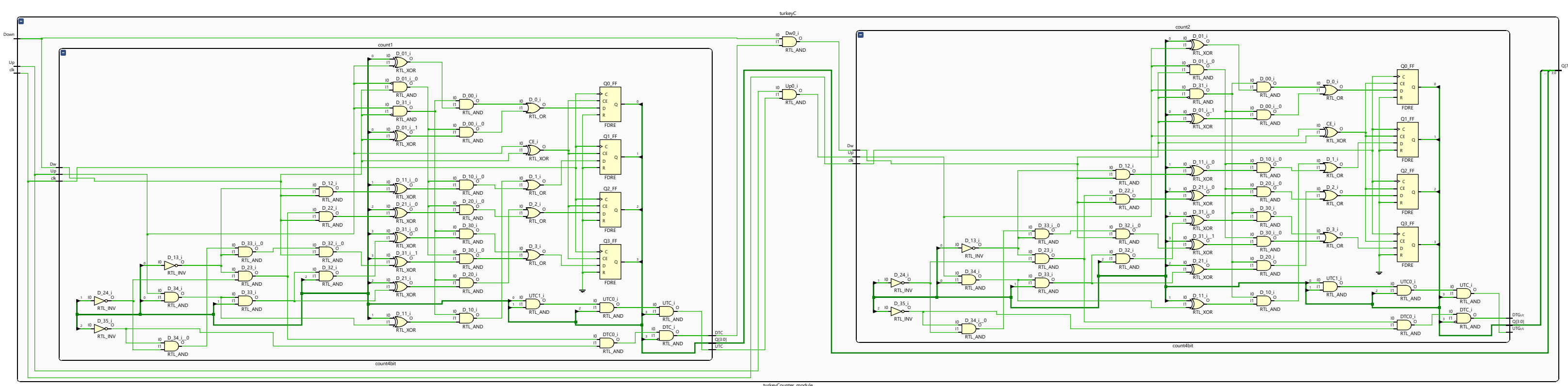
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/11/2020 03:43:17 PM
// Design Name:
// Module Name: selector
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module selector(
    input [3:0] sel,
    input [15:0] N,
    output [3:0] H
    );
    assign H[0] = (sel[0] & ~sel[1] & ~sel[2] & ~sel[3] & N[0]) |
(~sel[0] & sel[1] & ~sel[2] & ~sel[3] & N[4])| (~sel[0] & ~sel[1] &
sel[2] & ~sel[3] & N[8]) | (~sel[0] & ~sel[1] & ~sel[2] & sel[3] &
N[12]);
    assign H[1] = (sel[0] & ~sel[1] & ~sel[2] & ~sel[3] & N[1]) |
(~sel[0] & sel[1] & ~sel[2] & ~sel[3] & N[5])| (~sel[0] & ~sel[1] &
sel[2] & ~sel[3] & N[9]) | (~sel[0] & ~sel[1] & ~sel[2] & sel[3] &
N[13]);
```

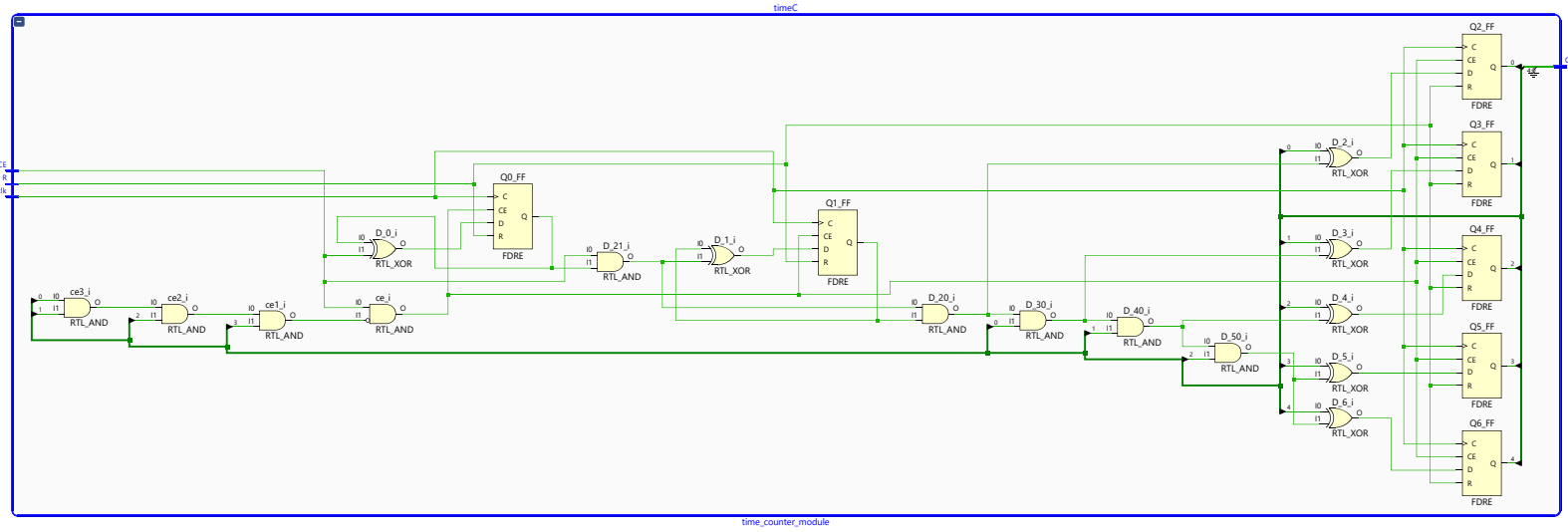
```
    assign H[2] = (sel[0] & ~sel[1] & ~sel[2] & ~sel[3] & N[2]) |
(~sel[0] & sel[1] & ~sel[2] & ~sel[3] & N[6])| (~sel[0] & ~sel[1] &
sel[2] & ~sel[3] & N[10]) | (~sel[0] & ~sel[1] & ~sel[2] & sel[3] &
N[14]);
    assign H[3] = (sel[0] & ~sel[1] & ~sel[2] & ~sel[3] & N[3]) |
(~sel[0] & sel[1] & ~sel[2] & ~sel[3] & N[7])| (~sel[0] & ~sel[1] &
sel[2] & ~sel[3] & N[11]) | (~sel[0] & ~sel[1] & ~sel[2] & sel[3] &
N[15]);
endmodule
```

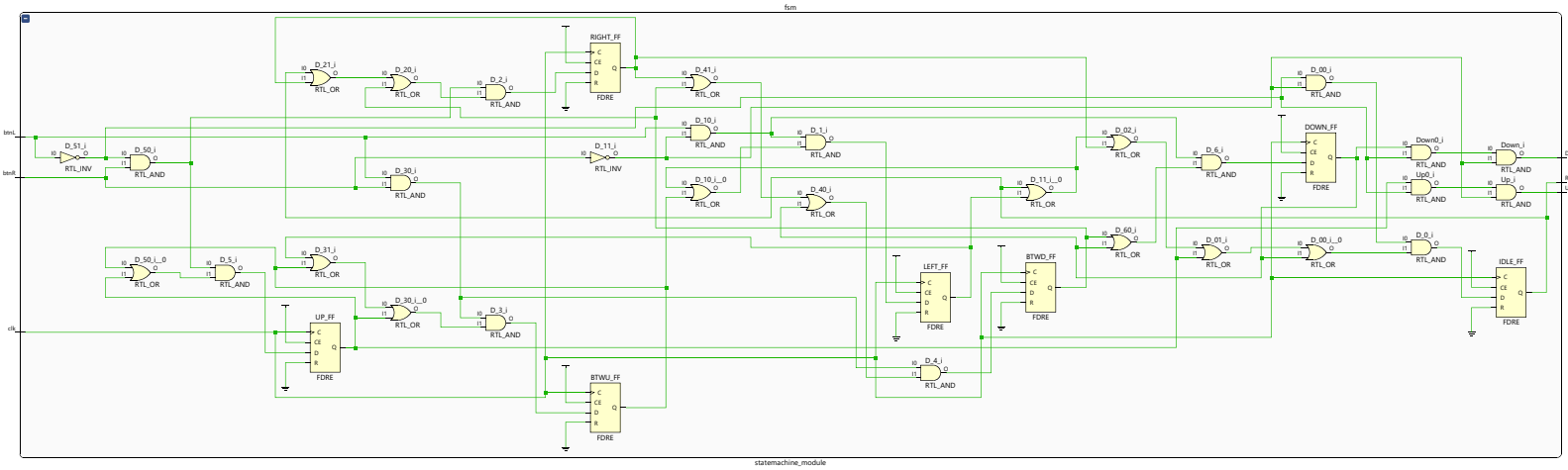
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/16/2020 11:10:56 AM
// Design Name:
// Module Name: count4bit
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module count4bit(
    input clk,
    input Up,
    input Dw,

    output [3:0] Q,     //Current Value of counter
    output UTC,         //is 1 only when counter is 1111
    output DTC          //is 1 only when counter is 0000
    );
    //Loading in Values from Din
    wire [3:0] D;
    wire CE;
    assign CE = Up^Dw;
```

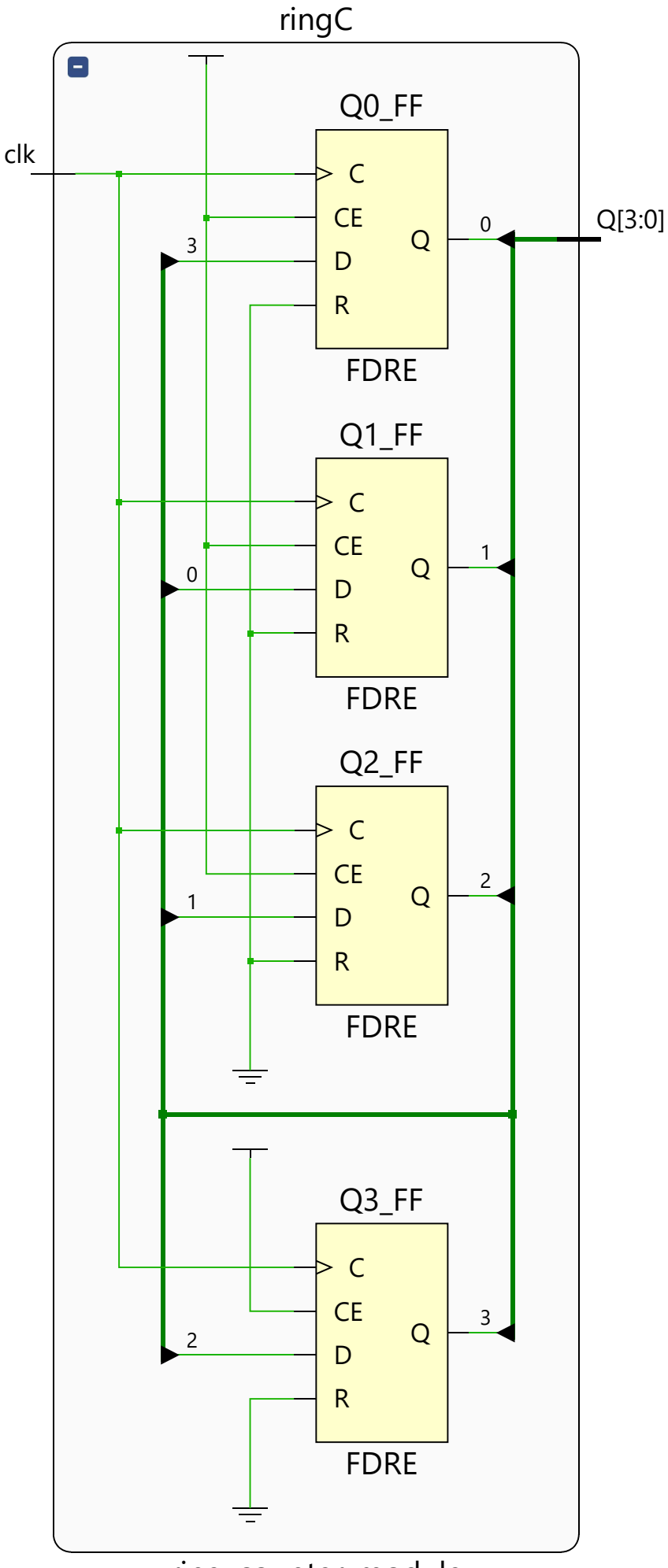
```
    FDRE #(.INIT(1'b0)) Q0_FF (.C(clk), .CE(CE), .D(D[0]),
.Q(Q[0]));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .CE(CE), .D(D[1]),
.Q(Q[1]));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .CE(CE), .D(D[2]),
.Q(Q[2]));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .CE(CE), .D(D[3]),
.Q(Q[3]));

    assign D[0] = (Up & ~Dw & (Q[0] ^ Up))                    | (~Up
& Dw)&(Q[0] ^ Dw);
    assign D[1] = (Up & ~Dw & (Q[1] ^ (Up&Q[0])))             | (~Up
& Dw)&(Q[1] ^ (~Q[0]&Dw));
    assign D[2] = (Up & ~Dw & (Q[2] ^ (Up&Q[0]&Q[1])))        | (~Up
& Dw)&(Q[2] ^ (~Q[1]&~Q[0]&Dw));
    assign D[3] = (Up & ~Dw & (Q[3] ^ (Up&Q[0]&Q[1]&Q[2]))) | (~Up
& Dw)&(Q[3] ^ (~Q[2]&~Q[1]&~Q[0]&Dw));

   assign UTC = (Q[0] & Q[1] & Q[2] & Q[3]);
   assign DTC = (~Q[0] & ~Q[1] & ~Q[2] & ~Q[3]);


endmodule
```

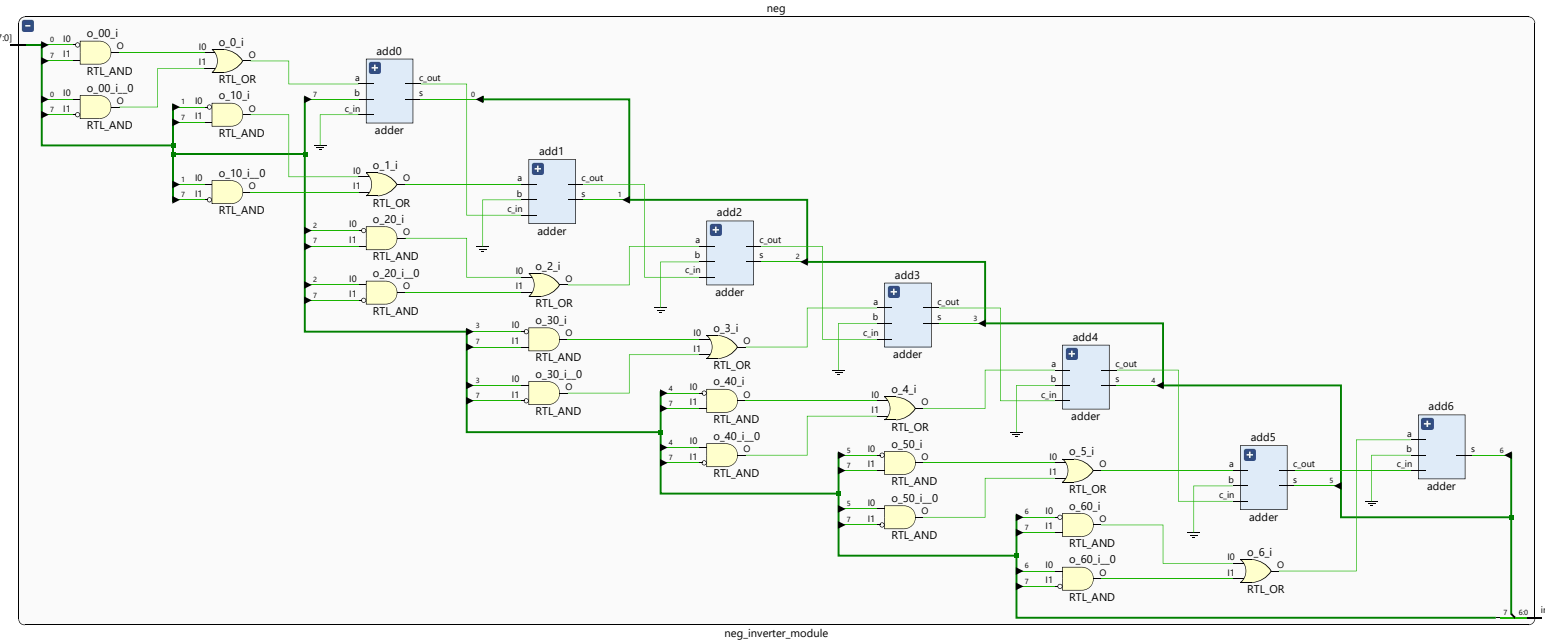

btnL
btnR
clkin
btnU
clkin_IBUF_inst
IBUF
btnU_IBUF_inst
IBUF
slowit
clkin
greset
clk
digsel
qsec
lab6_clks
fsm
btnL
btnR
clk
Down
ResetTimer
Up
statemachine_module
turkeyC
Down
Up
clk
Q[7:0]
turkeyCounter_module
ringC
Advance
clk
Q[3:0]
ring_counter_module
timeC
CE
R
clk
Q[7:0]
time_counter_module
neg
in[7:0]
invOut[7:0]
neg_inverter_module
selec
N[15:0]
sel[3:0]
H[3:0]
selector
hex
n[3:0]
seg[6:0]
hex7seg
an4_i
an3_i
an3_i__0
an3_i__1
an3_i__2
an2_i
an2_i__0
an2_i__1
an2_i__2
seg1_i__0
seg1_i__1
seg1_i__2
seg1_i__3
seg1_i__4
an1_i
an1_i__0
seg1_i
an1_i__1
an1_i__2
seg0_i__0
seg0_i__1
seg0_i__2
seg0_i__3
seg0_i__4
seg0_i__5
an0_i
an0_i__0
seg0_i
an0_i__1
an0_i__2
LED9_i
LED15_i
dp_OBUF_inst
RTL_AND
RTL_OR
RTL_INV
OBUF
seg[6:0]
LED9
an[3:0]
LED15
dp


turkeyC
count1
count2
count4bit
turkeyCounter_module


timeC
CE
R
clk
ce3_i
RTL_AND
ce2_i
RTL_AND
ce1_i
RTL_AND
ce_i
RTL_AND
D_0_i
RTL_XOR
Q0_FF
FDRE
D_21_i
RTL_AND
D_1_i
RTL_XOR
Q1_FF
FDRE
D_20_i
RTL_AND
D_30_i
RTL_AND
D_40_i
RTL_AND
D_50_i
RTL_AND
D_2_i
RTL_XOR
Q2_FF
FDRE
D_3_i
RTL_XOR
Q3_FF
FDRE
D_4_i
RTL_XOR
Q4_FF
FDRE
D_5_i
RTL_XOR
Q5_FF
FDRE
D_6_i
RTL_XOR
Q6_FF
FDRE
Q[7:0]
time_counter_module


fsm
btnL
btnR
clk
RIGHT_FF
D_21_i
D_20_i
D_2_i
D_41_i
D_10_i
D_1_i
D_11_i
D_10_i_0
D_40_i
D_51_i
D_50_i
D_30_i
D_50_i_0
D_5_i
D_31_i
D_30_i_0
D_3_i
UP_FF
BTWU_FF
D_4_i
LEFT_FF
BTWD_FF
D_60_i
D_02_i
D_6_i
D_11_i_0
D_01_i
D_00_i
D_00_i_0
DOWN_FF
Down0_i
Up0_i
Down_i
Up_i
D_0_i
IDLE_FF
Down
ResetTimer
Up
RTL_OR
RTL_AND
RTL_INV
FDRE
statemachine_module


ringC
clk
Q0_FF
C
CE
D
R
Q
FDRE
3
0
Q[3:0]
Q1_FF
C
CE
D
R
Q
FDRE
0
1
Q2_FF
C
CE
D
R
Q
FDRE
1
2
Q3_FF
C
CE
D
R
Q
FDRE
2
3
ring_counter_module


neg
in[7:0]
o_00_i
RTL_AND
o_00_i__0
RTL_AND
o_0_i
RTL_OR
o_10_i
RTL_AND
o_10_i__0
RTL_AND
add0
adder
o_1_i
RTL_OR
o_20_i
RTL_AND
o_20_i__0
RTL_AND
add1
adder
o_2_i
RTL_OR
o_30_i
RTL_AND
o_30_i__0
RTL_AND
add2
adder
o_3_i
RTL_OR
o_40_i
RTL_AND
o_40_i__0
RTL_AND
add3
adder
o_4_i
RTL_OR
o_50_i
RTL_AND
o_50_i__0
RTL_AND
add4
adder
o_5_i
RTL_OR
o_60_i
RTL_AND
o_60_i__0
RTL_AND
add5
adder
o_6_i
RTL_OR
add6
adder
invOut[7:0]
neg_inverter_module