CSE100 Lab 6

1 Lab Overview

For this final Verilog assignment we were tasked with developing a turkey counter which would use two infrared sensors to detect the movement of turkeys as they walk in front of the sensor. The counter would count positively if a turkey moves right to left or negatively if the turkey moves left to right. Additionaly, the turkey counter woould also display the time each turkey takes to cross, up to a maximum of 15 seconds (displayed using a single hexadecimal digit). Several rules were set to make our job easy as circuit designers

- Only one turkey could cross at a time.
- Turkeys were big enough that if a turkey walked across the sensors then both counters would ultimately be activated simultaneously.
- Turkeys move slow when compared to the system clock (which is obvious) so that both infrared sensors couldn't be activated or deactivated at the same instant.

For this assignment we represented the infrared sensors using btnL and btnR in our Basys3 board. Also, the btnU was used as a global reset button. LEDs 15 and 9 were used to represent the low and high infrared sensors of the left and right respectively.

The aspect that differentiates this assignment from others is that we were given minimal guidance, so we had to use our own modules or create new as we saw fit.

Turkeys -as turkeys do- can walk back and forth between sensors. If a turkey, after going back and forth, walks off from the sensor they first came through, the count doesn't change. In the other hand, if after walking back and forth they go through the sensor they didn't first walk through the counter adds or subtracts from the counter depending on the direction.

1.1 State_Machine

The state machine for this assignment is meant to switch from states as the turkey counter receives signals from the infrared sensors (the buttons). Below is the state diagram for the state machine used in this laboratory assignment

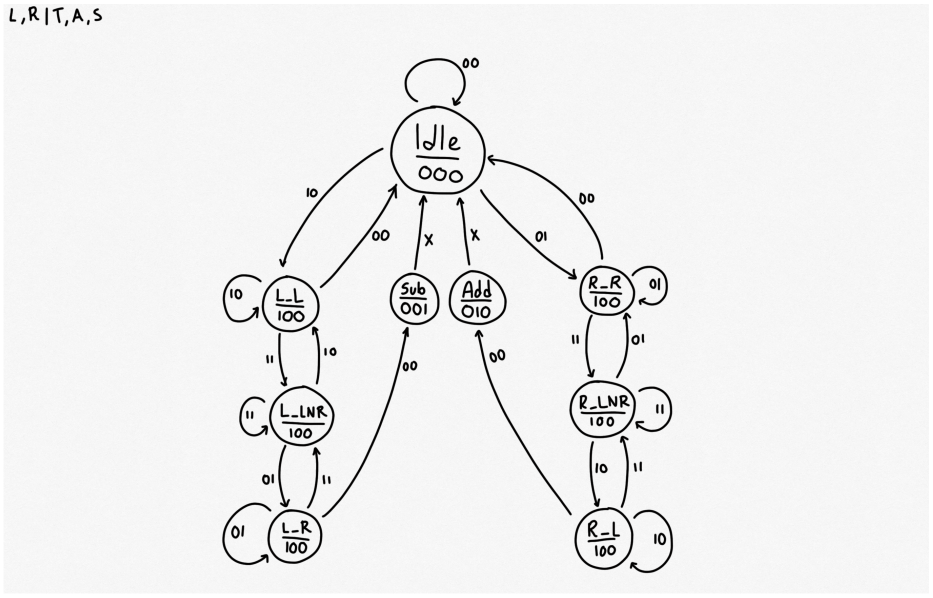


Figure 1: State diagram for laboratory assignment 6. Input is Left (L), Right (R) and outputs are TimeDown (T), Add (A), and Subtract (S).

Below are the following states (with a much in-depth description in the next section)

- Idle: State awaiting for any sensor to be activated. Is the default state and the intital state for this machine.
- L_L: Turkey walking from the left and activating only the left sensor.
- L_LNR: Turkey walking from the left and activating both the left and right sensors.
- L_R: Turkey walking from the left and activating only the right sensor.
- Sub: Turkey that walked from the left to the right. This state sends the subtract signal for the counter.
- R_R: Turkey walking from the right and activating only the right sensor.
- R_LNR: Turkey walking from the right and activating both the left and right sensors.
- R_L: Turkey walking from the right and activating only the left sensor.
- Add: Turkey that walked from the right to the left. This state sends the add signal for the counter.

1.2 Negative inverter

For counting upwards and downwards with a negative sign we needed to design a module that receives the signals from the state machine -Add or Sub- and inverts them if the negative sign is active. For this, it is necessary to also receive the DTC signal of the counter itself, which tells us when the count is at zero (the only value where the sign can change).

1.3 Other modules

Modules in this laboratory assignment were taken from previous assignments, such as hex7seg, m8_1e, Ring_Counter, selector, and Time_Counter. For this lab we used variations from modules of previous assignments, such as Top_Module, countU4L (which became countU8L for an 8 bit counter), and m2_1x8 (which became m2_1x7).

2 Implementation and Logic

2.1 Modification of Previous Modules

This laboratory assignment required that we did some modifications of the modules of previous assignments.

- The m2_1x8 multiplexer got transformed into m2_1x7 multiplexer to be used in the hex7seg to display the negative symbol in the 2nd 7-segment display (if it was turned high).
- A counter countU8L was modified from countU4L which counted the timer down until it reached 0 upon which the signal DTC was sent. This was to use the higher 4 bits for the timer so the clock did not advance as fast.
- The Top_Module was modified to properly connect the different parts of the sequential circuit. The implementation of the Top_Module will be ignored in this report since it is straight forward enough that it doesn't need an explanation.

2.2 Negative inverter

An inverter is needed for the turkey counter since a negative number will count up or down in an opposite way than a positive number. For the Negative module which contained the inverter it was needed that it sent the N signal (negative), when the system had a negative number loaded. A signal positive edge-triggered flip-flop was used for this module. For this, the following logic table and subsequent K-map was used:

Q	Add	Sub	DTC	N
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	X
0	1	1	1	X
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	X
1	1	0	0	1
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Table 1: The flip-flop has input N and clk and output Q. Add and Sub are the signals received from the state machine. DTC is received from the turkey counter (countD8L).

	Sub,DTC				
Q,Add		00	01	11	10
	00	0	0	1	0
	01	0	0	X	X
	11	1	X	X	X
	10	1	0	X	1

Table 2: K-map representing the previous logic table.

From the K-map we can extract the following function for D:

$$N = (Q \cap \neg DTC) \cup (Sub \cap DTC)$$

Once the inverter knows when the system has a negative number loaded, we get the following functions:

$$Dw = Sub \cap \neg N \cup Add \cap N, \; Up = Add \cap \neg N \cup Sub \cap N.$$

Notice that we are using N for the output, and not Q, this is because when we add or subract a number when DTC is active we cannot afford to wait an entire clock cycle for the negative signal to show up, which might lead to errors.

2.3 State Machine

As shown in **Figure 1** the state machine has 9 total states. Below is a more detailed description of the states and their transitions. All inputs and outputs will we refered as XX and XXX respectively, where the input XX is Left (L) and Right (R) -which are signals coming from btnL and btnR respectively-, and output XXX TimeDown (T), Add (A), and Subtract (S).

- Idle: Transitions to L_L when 10 is received, to R_R when 01 is received, and remains in Idle with input 00. Outputs 000.
- L_L: Transitions back to Idle with 00 input and to L_LNR with 11 input. Keeps state if 10 is received. Outputs 100.
- L_LNR: Changes to L_L if 10, to L_R if 01, and stays in L_LNR if 11. Outputs 100.
- L_R: If 01 it stays the same. If 11 it goes back to L_LNR. If 00 it goes to Sub. Outputs 100.
- Sub: Stays in Sub for one clock cycle and shifts to Idle, regardless of input. Outputs 001.
- R_R: Transitions back to Idle upon receiving 00 input and to R_LNR with 11 input. Remains in the same state if 01 is received. Outputs 100.
- R_LNR: Changes to R_L if 10, to R_R if 01, and stays in R_LNR if 11. Outputs 100.
- R_L: If 10 is received it stays the same. If 11 is received it goes back to R_LNR. If 00 is received it goes to Add. Outputs 100.
- Add: Add for one clock cycle and then goes back to Idle, regardless of input. Outputs 010.

This state logic leads to the following state transition table

	P.S.	N.S.				y		
		00	01	11	10	T	A	S
A	Idle	A	F	-	B	0	0	0
B	L_L	A	-	C	B	1	0	0
C	L_LNR	-	D	C	B	1	0	0
D	L_R	E	D	C	-	1	0	0
E	Sub	A	A	A	A	0	0	1
F	R_R	A	F	G	-	1	0	0
G	R_LNR	-	F	G	H	1	0	0
H	R_L	I	-	G	H	1	0	0
I	Add	A	A	A	A	0	1	0

Table 3: State transition table.

Observe that multiple don't-cares are present which are denoted "-". It is important to set these to A as an unexpected state can lead to the entire system to fail. The following functions represent the one-hot encoding for our states:

$$S_0 = \neg(S_1 \cup S_2 \cup S_3 \cup S_4 \cup S_5 \cup S_6 \cup S_7 \cup S_8),\ S_1 = (S_0 \cup S_1 \cup S_2) \cap (L \cap \neg R)$$
$$S_2 = (S_1 \cup S_2 \cup S_3) \cap (L \cap R),\ S_3 = (S_2 \cup S_3) \cap (\neg L \cap R)$$

$$S_4 = S_3 \cap (\neg L \cap \neg R),\; S_5 = (S_0 \cup S_5 \cup S_6) \cap (\neg L \cap R)$$
$$S_6 = (S_5 \cup S_6 \cup S_7) \cap (L \cap R),\; S_7 = (S_6 \cup S_7) \cap (L \cap \neg R),\; S_8 = S_7 \cap (\neg L \cap \neg R)$$

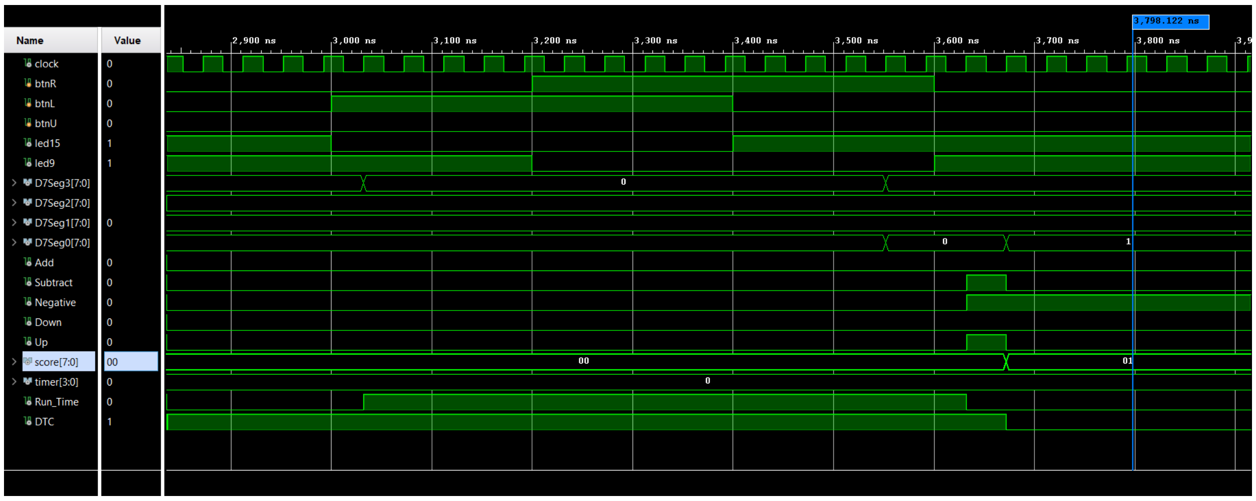
These functions were implemented in Verilog which gave us the desired state machine. Observe that S_0 was simplified as being the default state whenever no other state was active. We can afford to do this generalization as not only will it cover the don't-cares and adjust for unexpected states, but also won't affect our machine negatively as it takes only one clock cycle to got to default. The clock cycles are so fast that it is extremely unlikely for a turkey to walk in front of a sensor within two clock cycles of another turkey walking out.

3 Conclusion and Comments

This assignment was fairly straight forward, yet the difficulty relied on developing the entire circuit with not much instruction. The task was mainly to use all of our previous knowledge and projects to craft a new one that adapted to a completely different scenario. Overall it was a fantastic practice to solidify our understanding of how to develop final state machines.

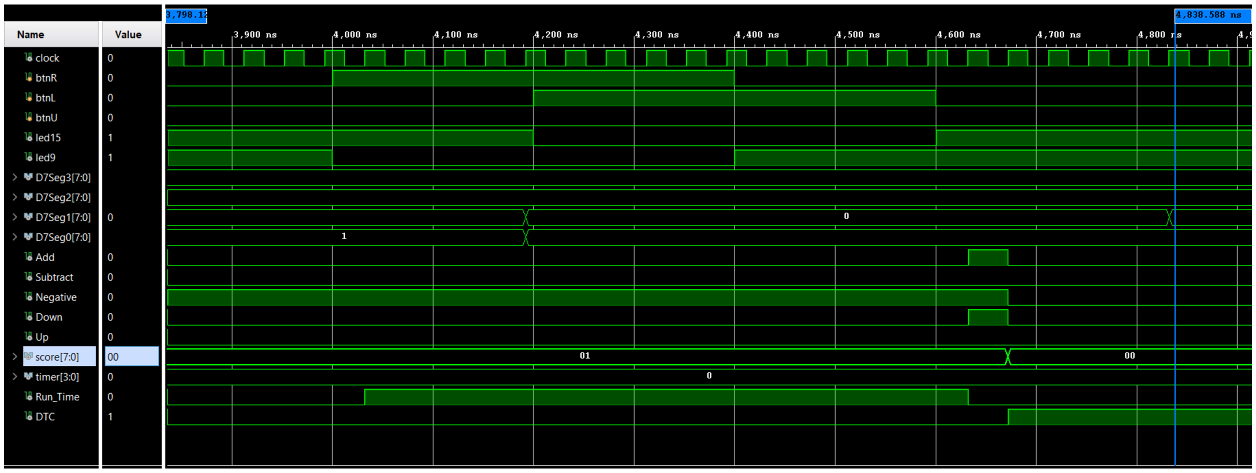
Appendix

Turkey crossing left to right directly.



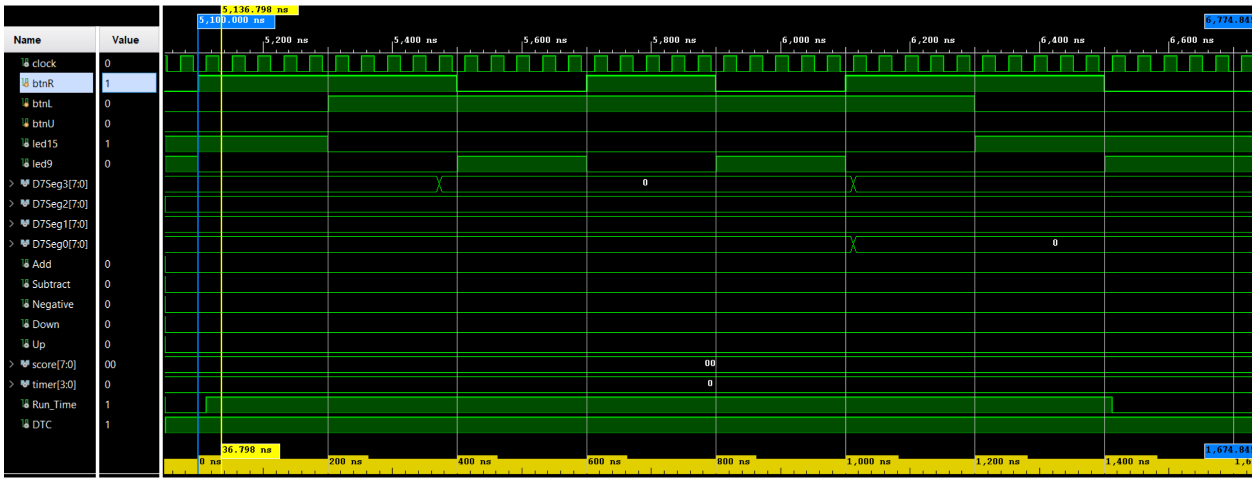
Name	Value
clock	0
btnR	0
btnL	0
btnU	0
led15	1
led9	1
D7Seg3[7:0]	
D7Seg2[7:0]	
D7Seg1[7:0]	0
D7Seg0[7:0]	
Add	0
Subtract	0
Negative	0
Down	0
Up	0
score[7:0]	00
timer[3:0]	0
Run_Time	0
DTC	1

Turkey crossing right to left directly.



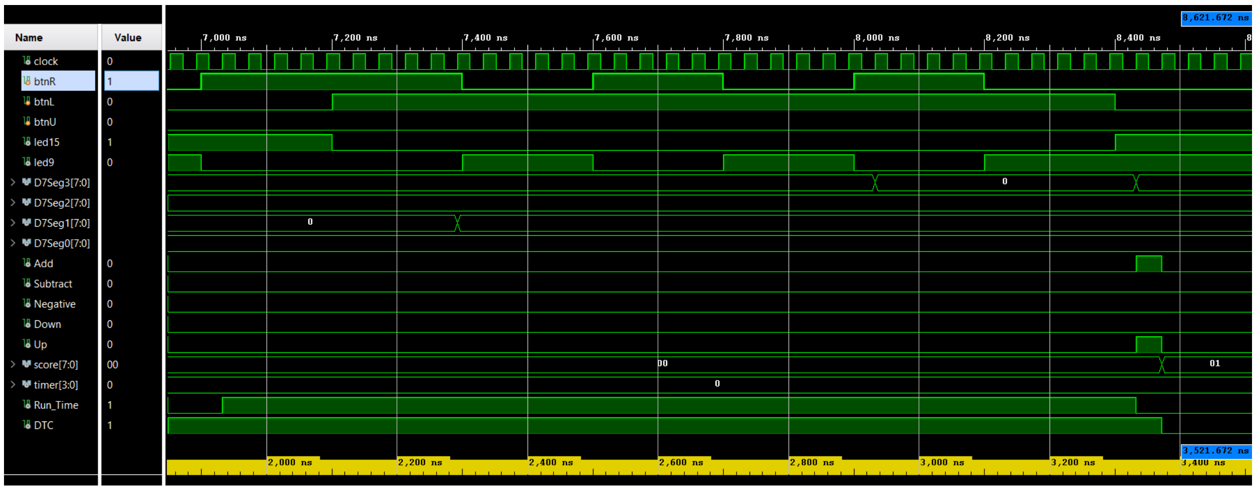
Name	Value
clock	0
btnR	0
btnL	0
btnU	0
led15	1
led9	1
D7Seg3[7:0]	
D7Seg2[7:0]	
D7Seg1[7:0]	0
D7Seg0[7:0]	
Add	0
Subtract	0
Negative	0
Down	0
Up	0
score[7:0]	00
timer[3:0]	0
Run_Time	0
DTC	1

Turkey entering from right and wandering back and forth before retreating to the right.

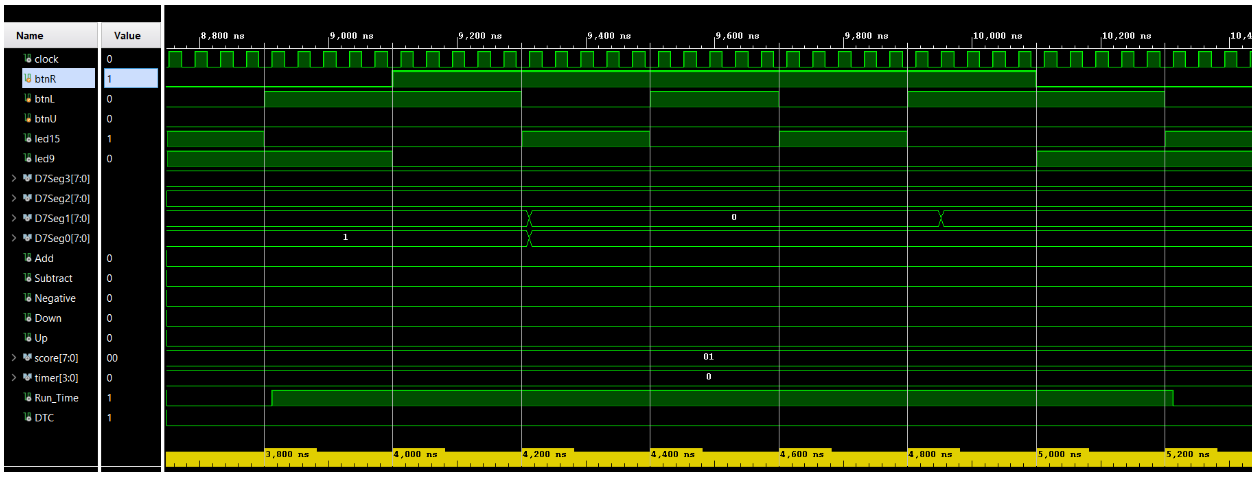


Name	Value
clock	0
btnR	1
btnL	0
btnU	0
led15	1
led9	0
D7Seg3[7:0]	
D7Seg2[7:0]	
D7Seg1[7:0]	
D7Seg0[7:0]	
Add	0
Subtract	0
Negative	0
Down	0
Up	0
score[7:0]	00
timer[3:0]	0
Run_Time	1
DTC	1

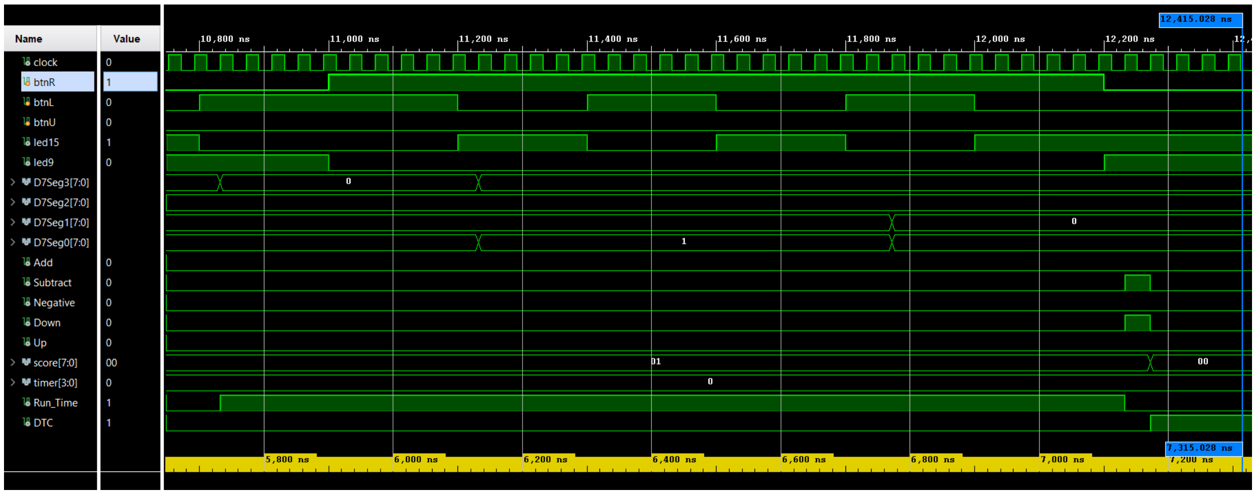
Turkey entering from right and wandering back and forth before crossing to the left.



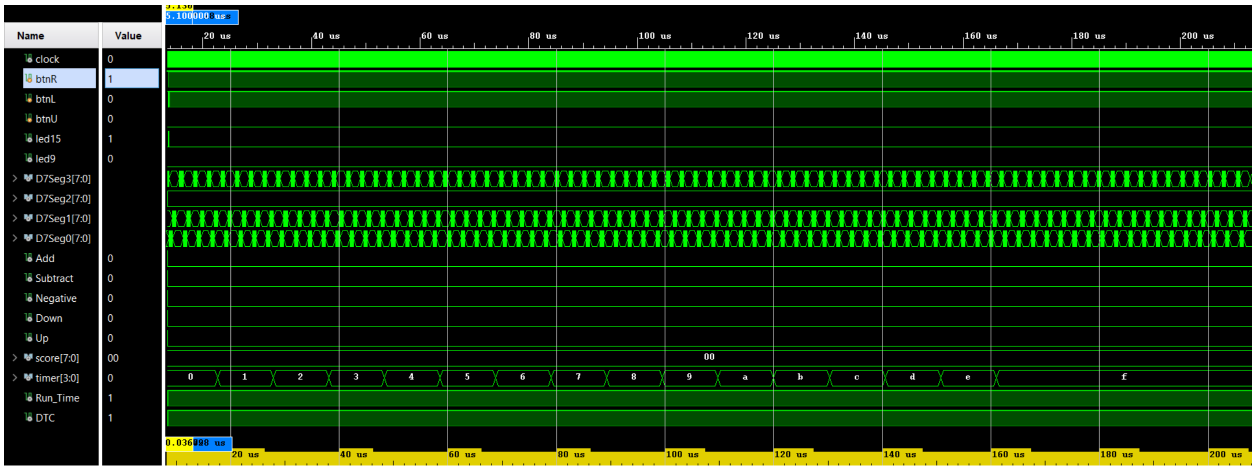
Turkey entering from left and wandering back and forth before retreating to the left.

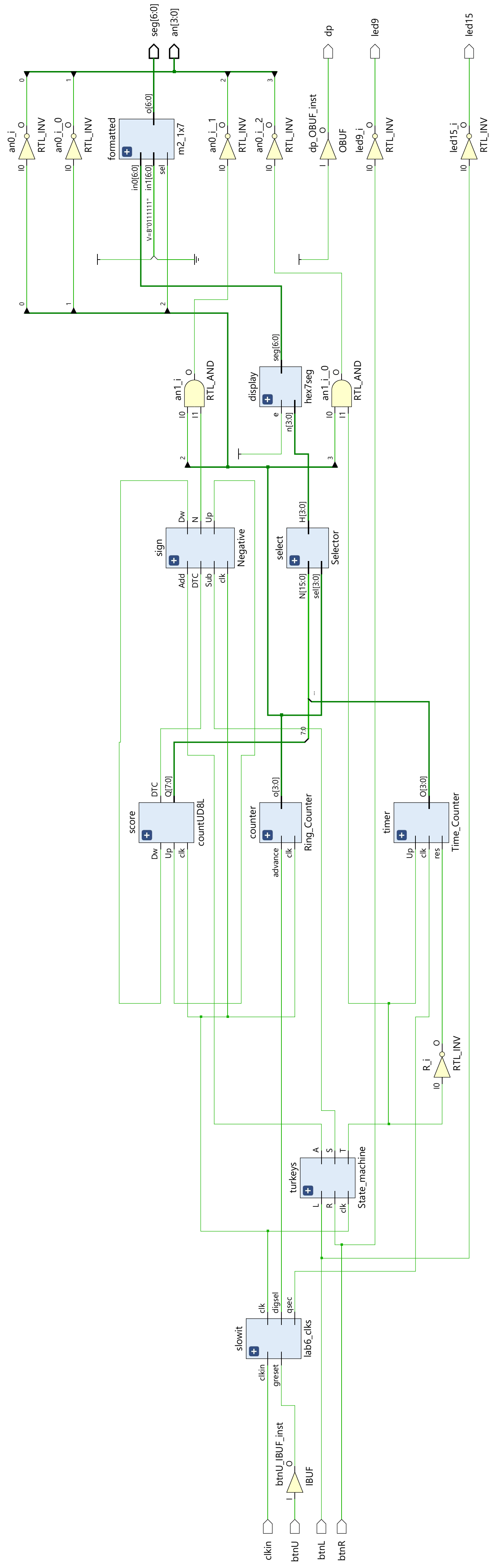


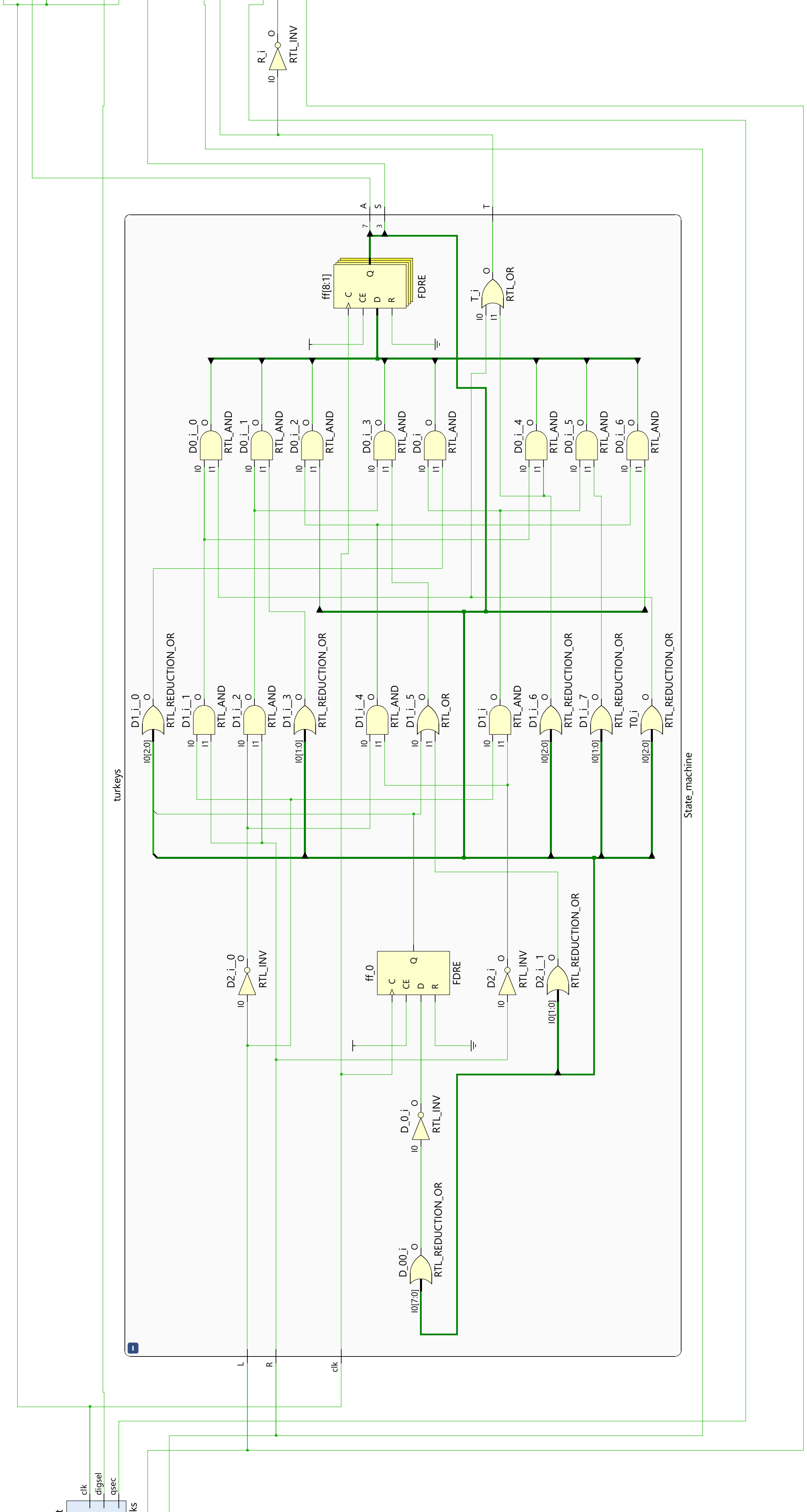
Turkey entering from left and wandering back and forth before retreating to the right.



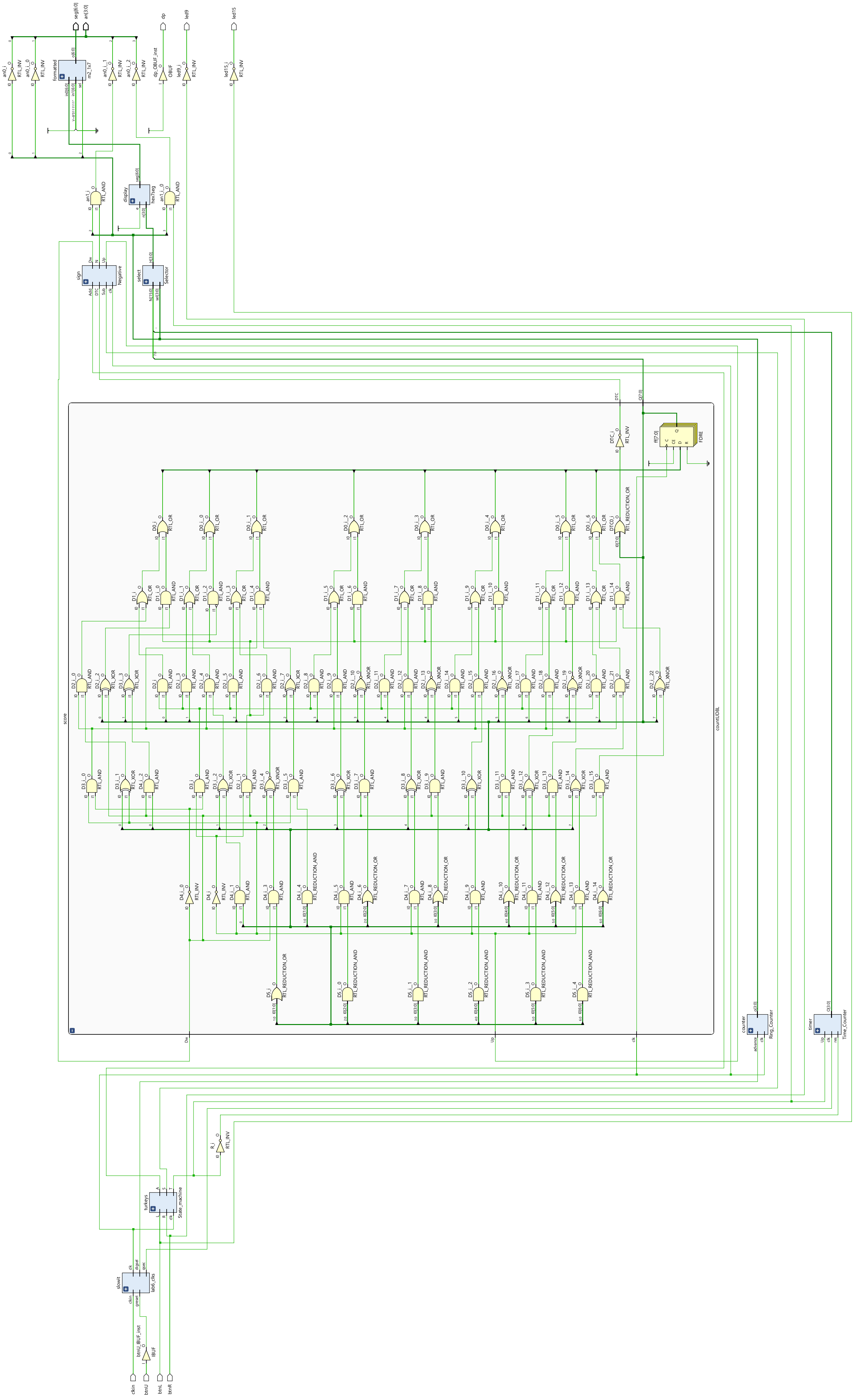
Turkey entering and wandering back and forth until timer reaches F.

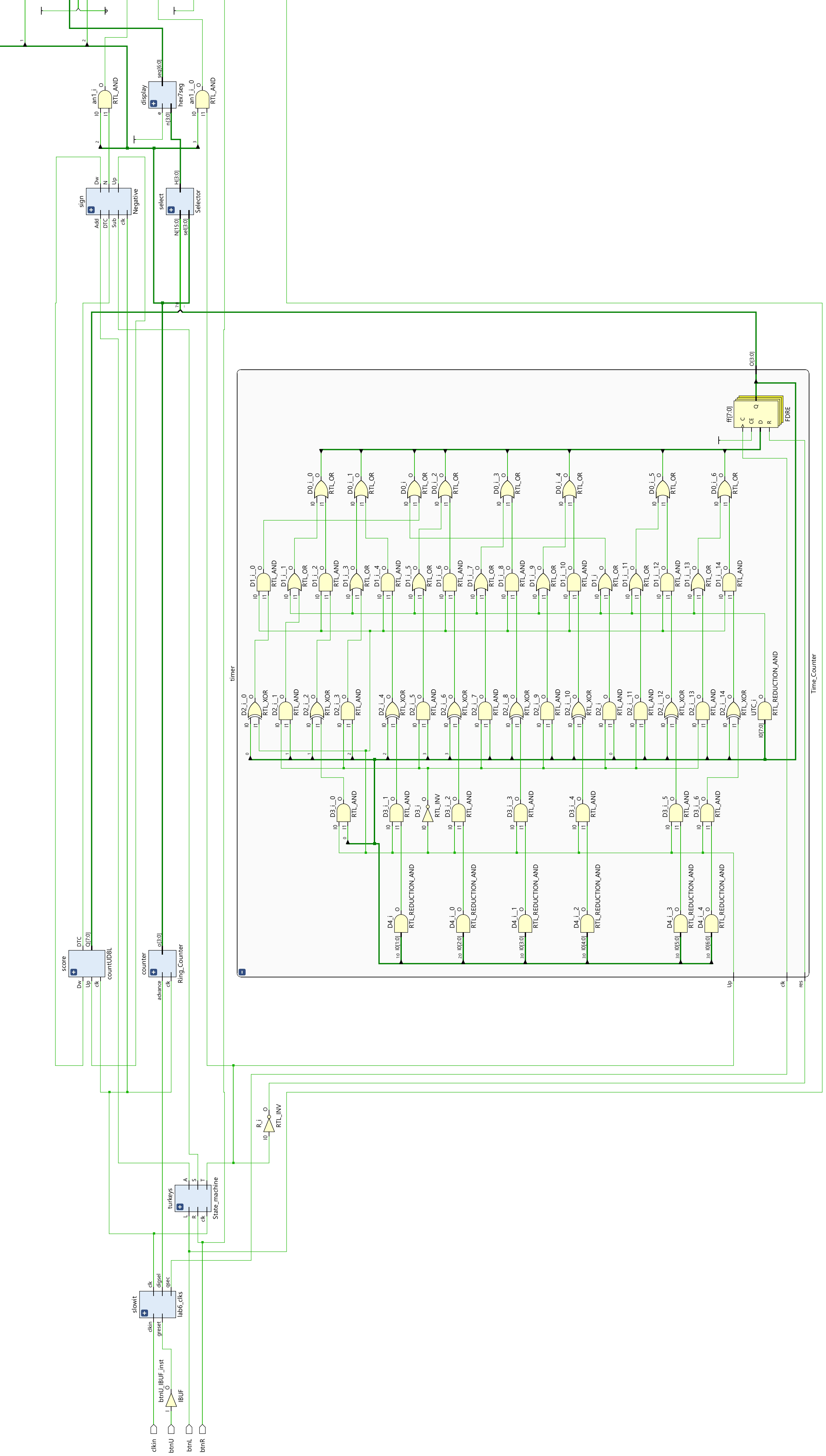



seg[6:0]
an[3:0]
dp
led9
led15
an0_i
an0_i_0
formatted
m2_1x7
an0_i_1
an0_i_2
dp_OBUF_inst
OBUF
led9_i
led15_i
RTL_INV
o[6:0]
in0[6:0]
in1[6:0]
sel
V=8"0111111"
an1_i
RTL_AND
display
hex7seg
seg[6:0]
e
n[3:0]
an1_i_0
sign
Negative
Dw
N
Up
Add
DTC
Sub
clk
select
Selector
H[3:0]
N[15:0]
sel[3:0]
7:0
score
countUD8L
DTC
Q[7:0]
Dw
Up
clk
counter
Ring_Counter
o[3:0]
advance
clk
timer
Time_Counter
O[3:0]
Up
clk
res
R_i
RTL_INV
turkeys
State_machine
A
S
T
L
R
clk
slowit
lab6_clks
clk
digsel
qsec
clkin
greset
btnU_IBUF_inst
IBUF
clkin
btnU
btnL
btnR



R_i
RTL_INV
A
S
T
7
3
ff[8:1]
FDRE
C
CE
D
R
Q
T_i
RTL_OR
D0_i_0
D0_i_1
D0_i_2
D0_i_3
D0_i
D0_i_4
D0_i_5
D0_i_6
RTL_AND
turkeys
D1_i_0
D1_i_1
D1_i_2
D1_i_3
D1_i_4
D1_i_5
D1_i
D1_i_6
D1_i_7
T0_i
RTL_REDUCTION_OR
State_machine
D2_i_0
ff_0
D2_i
D2_i_1
D_0_i
D_00_i
L
R
clk
digsel
qsec


seg[6:0]
an[3:0]
dp
led9
led15
formatted
m2_1x7
display
hex7seg
sign
Negative
select
Selector
score
countUD8L
counter
Ring_Counter
timer
Time_Counter
turkeys
State_machine
slowit
lab6_clks
btnU_IBUF_inst
IBUF
dp_OBUF_inst
OBUF
clkin
btnU
btnL
btnR
RTL_INV
RTL_AND
RTL_OR
RTL_XOR
RTL_XNOR
RTL_REDUCTION_OR
RTL_REDUCTION_AND
FDRE
Dw
Up
clk
DTC
Q[7:0]



an1_i
RTL_AND
display
hex7seg
seg[6:0]
an1_i_0
sign
Negative
Dw
N
Up
Add
DTC
Sub
clk
select
Selector
H[3:0]
N[15:0]
sel[3:0]
timer
Time_Counter
ff[7:0]
FDRE
O[3:0]
RTL_OR
RTL_XOR
RTL_INV
RTL_REDUCTION_AND
UTC_i
Up
clk
res
score
countUD8L
Q[7:0]
counter
Ring_Counter
o[3:0]
advance
R_i
turkeys
State_machine
A
S
T
L
R
slowit
lab6_clks
clkin
greset
digsel
qsec
btnU_IBUF_inst
IBUF
clkin
btnU
btnL
btnR

```
`timescale 1ns / 1ps

module Top_Module(
    input clkin,
    input btnR,
    input btnU,
    input btnL,
    output led15,
    output led9,
    output [3:0] an,
    output [6:0] seg,
    output dp

//    output clk,
//    output N,
//    output A,
//    output S,
//    output U,
//    output D,
//    output DTC,
//    output T,
//    output [7:0] points,
//    output [3:0] cross_time
    );

    wire [15:0] Q;
    wire digsel, qsec, Dw;
    wire [3:0] sel, H;
    wire R;
    wire [6:0] seg_;

    wire N, A, S, clk, D, U, T, DTC;

//    assign points = Q[7:0];
//    assign cross_time = Q[15:12];

    lab6_clks slowit (.clkin(clkin), .greset(btnU), .clk(clk),
.digsel(digsel), .qsec(qsec));

    Negative
```

```
sign(.DTC(DTC),.Add(A),.Sub(S),.clk(clk),.Dw(D),.Up(U),.N(N));
    countUD8L score(.clk(clk),.Up(U),.Dw(D),.Q(Q[7:0]),.DTC(DTC));

    State_machine
turkeys(.L(btnL),.R(btnR),.clk(clk),.T(T),.A(A),.S(S));

    assign R = ~T;
    Time_Counter timer(.Up(T),.clk(qsec),.res(R),.O(Q[15:12]));

    Ring_Counter counter(.clk(clk), .advance(digsel), .o(sel));
    Selector select(.sel(sel), .N(Q), .H(H));
    hex7seg display(.n(H), .seg(seg_), .e(1));
    m2_1x7
formatted(.in0(seg_),.in1(7'b0111111),.sel(sel[2]),.o(seg));

    assign an[0] = ~sel[0];
    assign an[1] = ~sel[1];
    assign an[2] = ~(sel[2] & N);
    assign an[3] = ~(sel[3] & T);

    assign dp = 1;
    assign led15 = ~btnL;
    assign led9 = ~btnR;
endmodule
```

```verilog
`timescale 1ns / 1ps

module State_machine(
    input L,
    input R,
    input clk,
    output T,
    output A,
    output S
    );

    wire [8:0] D, Q;

    assign D[0] =  ~(|Q[8:1]);                // Idle
    assign D[1] =  L &~R & (|Q[2:0]);         // L_L
    assign D[2] =  L & R & (|Q[3:1]) ;        // L_LNR
    assign D[3] = ~L & R & (|Q[3:2]) ;        // L_R
    assign D[4] = ~L &~R & Q[3];              // Sub
    assign D[5] = ~L & R & (Q[0]| |Q[6:5]); // R_R
    assign D[6] =  L & R & (|Q[7:5]);         // R_LNR
    assign D[7] =  L &~R & (|Q[7:6]);         // R_L
    assign D[8] = ~L &~R & Q[7];              // Add

    FDRE #(.INIT(1'b0) ) ff_0 (.C(clk), .R(), .CE(1'b1), .D(D[0]),
.Q(Q[0]));
    FDRE #(.INIT(1'b0) ) ff[8:1] (.C({8{clk}}), .R(),
.CE({8{1'b1}}), .D(D[8:1]), .Q(Q[8:1]));

    assign T = |Q[3:1] | |Q[7:5];
    assign A = Q[8];
    assign S = Q[4];

endmodule
```

```
`timescale 1ns / 1ps

module Negative(
    input DTC,
    input Add,
    input Sub,
    input clk,
    output Dw,
    output Up,
    output N
    );

    wire Q;

    assign N = Q & ~DTC | Sub & DTC;
    FDRE #(.INIT(1'b0) ) ff_n (.C(clk), .R(0), .CE(1), .D(N),
.Q(Q));

    assign Dw = Sub &~N | Add & N;
    assign Up = Sub & N | Add &~N;

endmodule
```

```
`timescale 1ns / 1ps

module Time_Counter(
    input Up,
    input clk,
    input res,
    output [3:0] O
    );

    wire [7:0] D, Q;
    wire UTC;
    assign D[0] = UTC |
       (~Up & Q[0]) |
       ( Up & (Q[0] ^ Up) );
    assign D[1] = UTC |
       (~Up &  Q[1]) |
       ( Up & (Q[1]^ (Up & Q[0])));
    assign D[2] = UTC |
       (~Up &  Q[2]) |
       ( Up & (Q[2]^ (Up & &Q[1:0])));
    assign D[3] = UTC |
       (~Up &  Q[3]) |
       ( Up & (Q[3]^ (Up & &Q[2:0])));
    assign D[4] = UTC |
       (~Up &  Q[4]) |
       ( Up & (Q[4]^ (Up & &Q[3:0])));
    assign D[5] = UTC |
       (~Up &  Q[5]) |
       ( Up & (Q[5]^ (Up & &Q[4:0])));
    assign D[6] = UTC |
       (~Up &  Q[6]) |
       ( Up & (Q[6]^ (Up & &Q[5:0])));
    assign D[7] = UTC |
       (~Up &  Q[7]) |
       ( Up & (Q[7]^ (Up & &Q[6:0])));

    FDRE #(.INIT(1'b0) ) ff[7:0] (.C({8{clk}}), .R(res),
.CE({8{1'b1}}), .D(D), .Q(Q));
    assign O = Q[7:4];
    assign UTC =   &Q[7:0];

endmodule
```

```
`timescale 1ns / 1ps
module countUD8L(
    input clk,
    input Up,
    input Dw,
    output [7:0] Q,
    output DTC
    );
    wire [7:0] D;
    wire UTC;

    assign D[0] =
       (~Up & ~Dw & Q[0]) |
       ( Up & ~Dw & (Q[0] ^ Up)) |
       (~Up &  Dw & (Q[0] ^ Dw));
    assign D[1] =
       (~Up & ~Dw &  Q[1]) |
       ( Up & ~Dw & (Q[1]^ (Up & Q[0]))) |
       (~Up &  Dw &~(Q[1]^ (Dw & Q[0])));
    assign D[2] =
       (~Up & ~Dw &  Q[2]) |
       (~Up &  Dw & (Q[2]^~(Dw & |Q[1:0]))) |
       ( Up & ~Dw & (Q[2]^ (Up & &Q[1:0])));
    assign D[3] =
       (~Up & ~Dw &  Q[3]) |
       ( Up & ~Dw & (Q[3]^ (Up & &Q[2:0]))) |
       (~Up &  Dw & (Q[3]^~(Dw & |Q[2:0])));
    assign D[4] =
       (~Up & ~Dw &  Q[4]) |
       ( Up & ~Dw & (Q[4]^ (Up & &Q[3:0]))) |
       (~Up &  Dw & (Q[4]^~(Dw & |Q[3:0])));
    assign D[5] =
       (~Up & ~Dw &  Q[5]) |
       ( Up & ~Dw & (Q[5]^ (Up & &Q[4:0]))) |
       (~Up &  Dw & (Q[5]^~(Dw & |Q[4:0])));
    assign D[6] =
       (~Up & ~Dw &  Q[6]) |
       ( Up & ~Dw & (Q[6]^ (Up & &Q[5:0]))) |
       (~Up &  Dw & (Q[6]^~(Dw & |Q[5:0])));
    assign D[7] =
       (~Up & ~Dw &  Q[7]) |
       ( Up & ~Dw & (Q[7]^ (Up & &Q[6:0]))) |
       (~Up &  Dw & (Q[7]^~(Dw & |Q[6:0])));

    FDRE #(.INIT(1'b0) ) ff[7:0] (.C({8{clk}}), .R(),
.CE({8{1'b1}}), .D(D), .Q(Q));

    assign UTC =   &Q[7:0];
    assign DTC = ~(|Q[7:0]);

endmodule
```

```
`timescale 1ns/1ps
// Verilog code to test UTC and DTC of your  16 bit counter




module testTC();
  reg clkin, btnR, btnU, btnL;
  wire led15, led9, dp;
  wire [6:0] seg;
  wire [3:0] an;
  wire [7:0] D7Seg3,D7Seg2,D7Seg1,D7Seg0;

    wire clock, Negative, Add, Subtract;
    wire [7:0] score;
    wire [3:0] timer;
    wire Run_Time, DTC, Down, Up;

  Top_Module
   UUT (
      .clkin(clkin),
      .btnR(btnR),
      .btnU(btnU),
      .btnL(btnL),
      .seg(seg),
      .led15(led15),
      .led9(led9),
      .an(an),
      .dp(dp),



      .clk(clock),
      .N(Negative),
      .A(Add),
      .S(Subtract),
      .points(score),
      .cross_time(timer),
      .T(Run_Time),
      .DTC(DTC),
      .D(Down),
      .U(Up)
      );
  show_7segDisplay  showit (.seg(seg),.dp(dp),.an(an),

.D7Seg0(D7Seg0),.D7Seg1(D7Seg1),.D7Seg2(D7Seg2),.D7Seg3(D7Seg3));

// Run this simulation for 3ms. If correct TX_ERROR should be 0 at
the end.
// UTC should be high and then go low at 2,705us and go low at
2,706.3us.

    parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;


    initial    // Clock process for clkin
    begin;

      btnR = 1'b0;
      btnU = 1'b0;
      btnL = 1'b0;

       #OFFSET
        clkin = 1'b1;
       forever
         begin
            #(PERIOD-(PERIOD*DUTY_CYCLE)) clkin = ~clkin;
         end
      end

    initial
    begin
    #3000;
//      Turkey crossing left to right directly.
    btnL = 1'b1;
    #200;
    btnR = 1'b1;
    #200;
    btnL = 1'b0;
    #200;
    btnR = 1'b0;
    #400;
//    Turkey crossing right to left directly.
    btnR = 1'b1;
    #200;
    btnL = 1'b1;
    #200;
    btnR = 1'b0;
    #200;
    btnL = 1'b0;
    #500;
//    Turkey entering from right and wandering back and forth
before retreating to the right.
    btnR = 1'b1;
    #200;
    btnL = 1'b1;
    #200;
    btnR = 1'b0;
    #200;
    btnR = 1'b1;
    #200;
    btnR = 1'b0;
    #200;
    btnR = 1'b1;
    #200;
    btnL = 1'b0;
    #200;
    btnR = 1'b0;
    #500;
//    Turkey entering from right and wandering back and forth
before crossing to the left.
    btnR = 1'b1;
    #200;
    btnL = 1'b1;
    #200;
    btnR = 1'b0;
    #200;
    btnR = 1'b1;
    #200;
    btnR = 1'b0;
    #200;
    btnR = 1'b1;
    #200;
    btnR = 1'b0;
    #200;
    btnL = 1'b0;
    #500
//    Turkey entering from left and wandering back and forth
before retreating to the left.
    btnL = 1'b1;
    #200;
    btnR = 1'b1;
    #200;
    btnL = 1'b0;
    #200;
    btnL = 1'b1;
    #200;
    btnL = 1'b0;
    #200;
    btnL = 1'b1;
    #200;
    btnR = 1'b0;
    #200;
    btnL = 1'b0;
    #500
//    Turkey entering from left and wandering back and forth
before retreating to the right.
    btnL = 1'b1;
    #200;
    btnR = 1'b1;
    #200;
    btnL = 1'b0;
    #200;
    btnL = 1'b1;
    #200;
    btnL = 1'b0;
    #200;
    btnL = 1'b1;
    #200;
    btnL = 1'b0;
    #200;
    btnR = 1'b0;
    #500
//    Turkey entering and wandering back and forth until timer
reaches F.
    btnR = 1'b1;
    #200;
    btnL = 1'b1;
    #200;
    btnR = 1'b0;
    #200;
    btnR = 1'b1;
    #200;
    btnL = 1'b0;
    #200;
    btnL = 1'b1;




//       btnR = 1'b0;
//       btnU = 1'b0;
//       btnL = 1'b0;
//       #1000;
//       btnR = 1'b1;
//       #400;
//       btnL = 1'b1;
//       #500;
//       btnL = 1'b0;
//       #800;
//       btnL = 1'b1;
//       #600;
//       btnR = 1'b0;
//       #200;
//       btnL = 1'b0;
//       #800;
//       btnR = 1'b1;
//       #400;
//       btnL = 1'b1;
//       #300;
//       btnL = 1'b0;
//       #400;
//       btnL = 1'b1;
//       #600;
//       btnR = 1'b0;
//       #200;
//       btnL = 1'b0;
//       #2000;


//       btnL = 1'b1;
//       #200;
//       btnR = 1'b1;
//       #200;
//       btnL = 1'b0;
//       #200;
//       btnR = 1'b0;
//       #400;
//       btnL = 1'b1;
//       #200;
//       btnR = 1'b1;
//       #200;
//       btnL = 1'b0;
//       #200;
//       btnR = 1'b0;
//       #400;
//       btnL = 1'b1;
//       #200;
//       btnR = 1'b1;
//       #200;
//       btnL = 1'b0;
//       #200;
//       btnR = 1'b0;
//       #400;
//       btnL = 1'b1;
//       #200;
//       btnR = 1'b1;
//       #200;
//       btnL = 1'b0;
//       #200;
//      btnR = 1'b0;
//      #400;
//      btnL = 1'b1;
//      #200;
//      btnR = 1'b1;
//      #200;
//      btnL = 1'b0;
//      #200;
//      btnR = 1'b0;
//      #2000;


//      btnL = 1'b1;
//      #200;
//      btnR = 1'b1;
//      #200;
//      btnR = 1'b0;
//      #200;
//      btnL = 1'b0;
//      #400;
//      btnL = 1'b1;
//      #200;
//      btnR = 1'b1;
//      #200;
//      btnR = 1'b0;
//      #200;
//      btnL = 1'b0;
//      #400;

    end
endmodule

module show_7segDisplay (
 input [6:0] seg,
 input dp,
 input [3:0] an,
 output reg [7:0] D7Seg0, D7Seg1, D7Seg2,D7Seg3);

 reg [7:0] val;
```

```
wire AN0, AN1, AN2, AN3;
assign AN0=an[0];
assign AN1=an[1];
assign AN2=an[2];
assign AN3=an[3];

 always @(AN0 or val)
  begin
         if (AN0 == 0) D7Seg0 <= val;
         else if (AN0 == 1) D7Seg0 <= " ";
         else D7Seg0 <= 8'bX;   //  non-blocking assignment
  end

 always @(AN1 or val)
  begin
         if (AN1 == 0) D7Seg1 <= val;
         else if (AN1 == 1) D7Seg1 <= " ";
         else D7Seg1 <= 8'bX;   //  non-blocking assignment
  end

 always @(AN2 or val)
  begin
         if (AN2 == 0) D7Seg2 <= val;
         else if (AN2 == 1) D7Seg2 <= " ";
         else D7Seg2 <= 8'bX;   //  non-blocking assignment
  end

 always @(AN3 or val)
  begin
         if (AN3 == 0) D7Seg3 <= val;
         else if (AN3 == 1) D7Seg3 <= " ";
         else D7Seg3 <= 8'bX;   //  non-blocking assignment
  end

   always @(seg)
   case (seg)
   7'b0111111:
        val = "-";
   7'b1111111:
        val = " ";
```

```
    7'b1000000:
         val = "0";
    7'b1111001:
         val = "1";
    7'b0100100:
         val = "2";
    7'b0110000:
         val = "3";
    7'b0011001:
         val = "4";
    7'b0010010:
         val = "5";
    7'b0000010:
         val = "6";
    7'b1111000:
         val = "7";
    7'b0000000:
         val = "8";
    7'b0010000:
         val = "9";
    7'b0001000:
         val = "A";
    7'b0000011:
         val = "B";
    7'b1000110:
         val = "C";
    7'b0100001:
         val = "D";
    7'b0000110:
         val = "E";
    7'b0001110:
         val = "F";
    7'b0000001:
         val = "-";
    default:
         val = 8'bX;
    endcase
endmodule
```