Lab 6: Turkey Traffic

Gabriel Dimas
20 May 2022
Section C

Description

The purpose of this lab was to investigate further how state machines function with user input. This lab implemented a state machine to sense the direction of a crossing turkey. Whether it crosses left to right or right to left. The sensor created only measures full crosses, not anywhere the turkey entered and exited the same sensor. The signed counter would count up when the turkey crosses right to left and would count down when the turkey crosses left to right. The counter also counts the seconds it takes for the turkey to cross

Design

- **Top_Module_Main**
  - Input: btnL, btnR, btnC
  - Output: [6:0] seg, dp, [3:0]an
  - Implementation: For the top module, connect the counters, the state machine, and the edge detectors together. The inputs for the top module should be used as the inputs for some of the other modules as well. This is where all the modules are connected together to ensure the entire machine is working.
- **StateMachine**
  - input: Left, Right, clk
  - Output: Add, Subtract
  - Implementation: This uses flip-flops to keep track of the user's state. This module also remembers the initial button pressed to ensure that the turkey crossed both beams completely and to prevent a run back from one given side. This memory uses a flip flop to keep track of the initial state. The state machine also keeps track of where the turkey is. If it starts on the left or right side. Depending on the turkey, the output would exclusively be Add or Subtract. Here is a snippet of the state machine:

```
assign Next_IDLE = (IDLE & !left & !right) | (LEFT_BREAK & !left & !right) |
                   (RIGHT_BREAK & !left & !right) | (ALL_BREAK & !left & !right);
FDRE #(.INIT(1'b1)) idle (.C(clk), .R(1'b0), .CE(1'b1), .D(Next_IDLE), .Q(IDLE));

assign Next_LEFT_BREAK = (LEFT_BREAK & left & !right) | (IDLE & left & !right) |
                         (RIGHT_BREAK & left & !right) | (ALL_BREAK & left & !right);
FDRE #(.INIT(1'b0)) left_break (.C(clk), .R(1'b0), .CE(1'b1), .D(Next_LEFT_BREAK), .Q(LEFT_BREAK));

assign Next_RIGHT_BREAK = (LEFT_BREAK & !left & right) | (IDLE & !left & right) |
                          (RIGHT_BREAK & !left & right) | (ALL_BREAK & !left & right);
FDRE #(.INIT(1'b0)) right_break (.C(clk), .R(1'b0), .CE(1'b1), .D(Next_RIGHT_BREAK), .Q(RIGHT_BREAK));

assign Next_ALL_BREAK = (LEFT_BREAK & left & right) | (IDLE & left & right) |
                        (RIGHT_BREAK & left & right) | (ALL_BREAK & left & right);
FDRE #(.INIT(1'b0)) all_break (.C(clk), .R(1'b0), .CE(1'b1), .D(Next_ALL_BREAK), .Q(ALL_BREAK));
```

Figure X

- **Turkey_Counter**
  - Input: Add, Subtract, clk
  - Output: Negative, [7:0] Turkeys
  - Implementation: This module keeps track of the turkey. This design is very simple yet very smart. There are two 4-bit counters that count up in the absolute value. When the turkey positively crosses, the counter increments in the positive direction. If the turkey negatively crosses, the counter decrements in the negative direction. When the counter is signed negatively, the counter's subtraction is an increment in the absolute value. The same goes for when the counter adds: it decrements in the absolute value towards zero. Zero is the point at which the sign is either on or off depending on the input. Here is a snippet of the Turkey_Counter:

```
assign increment = (Add & !negative & !MAX) | (Subtract &  (negative | zero) & !MAX);
assign decrement = (Add &  negative)        | (Subtract & !negative & !zero);
assign MAX = out[0] & out[1] & out[2] & out[3] & out[7] & out[6];
countUD4L counter  (.Up(increment),     .Dw(decrement),     .LD(1'b0), .Q(4'b0),
                    .clk(clk), .UTC(utc), .DTC(dtc), .Qout(out[3:0]));
countUD4L counter2 (.Up(increment&utc), .Dw(decrement&dtc), .LD(1'b0), .Q(4'b0),
                    .clk(clk), .DTC(dtc2), .Qout(out[7:4]));
```

Figure X

- **Time_Counter**
  - Input: clk, CE, Qsec, Reset
  - Output: Signal
  - Implementation: When the CE is HIGH, this module begins a positive count. This uses the Qsec signal and counters to increment the counter in order. When the counter reaches 'F', the counter holds the 'F' value. This is done by setting the enable portion of the counter connected to the UTC wire of the counter.
- **Edge_Detector**
  - Input: clk, btn
  - Output: out
  - Implementation: See **Design:** Edge_Detector *Lab 4: Multiplexers, Full Adders, and Seven Segment Displays*
- **Ring_Counter**
  - Inputs: digsel, clk
  - Outputs: [3:0]out
  - Implementation: See **Design:** Ring_Counter *Lab 4: Multiplexers, Full Adders, and Seven Segment Displays*
- **Selector**
  - Inputs: [3:0]in
  - Outputs: [3:0]out
  - Implementation: See **Design:** Selector *Lab 4: Multiplexers, Full Adders, and Seven Segment Displays*

-

- **hex7seg**
  - Inputs: [3:0]in
  - Outputs: [6:0]out
  - Implementation: See **Design:** hex7Seg *Lab 4: Multiplexers, Full Adders, and Seven Segment Displays*

Testing & Simulation

Testing the design was not much of a difficult task. To ensure it worked, make sure that the timer works as intended and that the state machine is working correctly as well. For the State Machine, it was able to work flawlessly. This is the brain of the entire project, and without it, the lab would not have been able to function. Ensure that no more than one state at a time is being used and that all the outputs have signals attached to them. There was a corner case that needed to be considered: how would the game react if the turkey went forward then back and left the beams? The counter should not count, so there was an added flip flop that keeps track of the initial beam that was broken. For example, if the right beam was broken first, the flip flop combined with an Edge Detector would change to '1' (or '0' if the left was broken first). With some logic, implement the outputs to go high respectively when the turkey unbreaks the first beam. This was a difficult case that was finally able to be resolved. Another problem that needed to be considered as the ability to press both buttons simultaneously and instantaneously. This corner case may be tricky, but with the Basys3 Board, it is highly unlikely to occur. This is because of the very fast clock cycle and the delay of the pushbuttons, a normal human would never be able to press them as quickly as the clock cycle. For this reason, it was safe to assume that there was no need to implement such a case.

Results

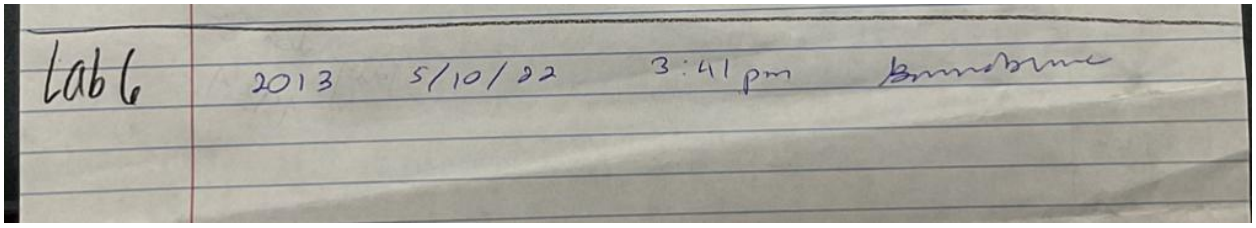
- IDLE: In this state, the machine is idling in the same state until input is given. The sensors are unbroken and nothing is being touched. This state is reached either in a loop or whenever the most recent press is depressed
- LEFT_BREAK: In this state, only the left button is pressed. When only the left button is pressed, this state runs, and the left LED goes HIGH and the timer starts to count (or continues if the left button is pressed first). One can only reach this state from the IDLE state or from the ALL_BREAK state.
- RIGHT_BREAK: In this state, only the right button is pressed. When only the right button is pressed, this state runs, and the right LED goes HIGH and the timer starts to count (or continues if the left button is pressed first). One can only reach this state from the IDLE state or from the ALL_BREAK state.
- ALL_BREAK: One can reach this state by pressing both buttons. As long as both buttons are pressed, this state runs. The buttons do not need to be pressed simultaneously, they only need to be down at the same time. One can only go into this state from the RIGHT_BREAK or LEFT_BREAK states.

The IDLE state was the only state that was used from the previous lab, but other than that, none of them were actually used again. The inputs and outputs were changed and the logic was changed to accommodate this new lab. The end of the **State_Machine** module assigns the outputs to the wires to give out the SUBTRACT and ADD signals.

Conclusion

In the end, the lab turned out to be quite a successful one. Although fairly easy, there are many applications that can occur in the real world. For one, this is important because some technologies rely on the sensor input to prevent catastrophes from happening. For example, this is very prominent in garage door openers because if something blocks the sensor, this prevents the door from shutting on them to prevent serious injuries. This is also very important with sports. Many sports (prominently running sports) use this technology to gauge when a runner completely crosses the finish line or to see if there was a tie between the two to find an absolute winner. In this lab, this application was no different. There were no hard difficulties, besides learning the best way to remember which side the turkey started. Although this was easily resolved. The component that was optimized was the turkey counter. The TAs instructed us to use 2's complement method, but it was much easier to add a bit to the front and call that a sign. When the sign was HIGH, the number was negative, else the number was positive. The deciding factor was when the number was at a state '0' and the down button was pressed. If the up button was pressed at '0', the number would increment positively.

Appendix



Lab 6 Code: 2013, **5/10/2022 3:41 PM**

Below are the Lab 6 Schematics, Verilog Code, and results.

```
module StateMachine(
    input Left,
    input Right,
    input clk,
    output RunGame,
    output Add,
    output Subtract
    );
    wire left, right, INIT_PRESS;
    wire IDLE, LEFT_BREAK, RIGHT_BREAK, ALL_BREAK;
    wire Next_IDLE, Next_LEFT_BREAK, Next_RIGHT_BREAK,
Next_ALL_BREAK, out;
    FDRE #(.INIT(1'b0)) sync1 (.C(clk), .R(1'b0),
.CE(1'b1), .D(Left), .Q(left));
    FDRE #(.INIT(1'b0)) sync2 (.C(clk), .R(1'b0),
.CE(1'b1), .D(Right), .Q(right));

    Edge_Detector btn_press (.clk(clk), .btn(!IDLE),
.out(out));
    FDRE #(.INIT(1'b0)) init_btn (.C(clk), .R(1'b0),
.CE(out), .D(Right), .Q(INIT_PRESS)); //1 if INIT
right, 0 if left



    wire IDLE, LEFT_BREAK, RIGHT_BREAK, ALL_BREAK;
    wire Next_IDLE, Next_LEFT_BREAK, Next_RIGHT_BREAK,
Next_ALL_BREAK;

    assign Next_IDLE = (IDLE & !left & !right) |
(LEFT_BREAK & !left & !right) |
                       (RIGHT_BREAK & !left & !right) |
```

```
(ALL_BREAK & !left & !right);
    FDRE #(.INIT(1'b1)) idle (.C(clk), .R(1'b0),
.CE(1'b1), .D(Next_IDLE), .Q(IDLE));

    assign Next_LEFT_BREAK = (LEFT_BREAK & left &
!right) | (IDLE & left & !right) |
                              (RIGHT_BREAK & left &
!right) | (ALL_BREAK & left & !right);
    FDRE #(.INIT(1'b0)) left_break (.C(clk), .R(1'b0),
.CE(1'b1), .D(Next_LEFT_BREAK), .Q(LEFT_BREAK));

    assign Next_RIGHT_BREAK = (LEFT_BREAK & !left &
right) | (IDLE & !left & right) |
                               (RIGHT_BREAK & !left &
right) | (ALL_BREAK & !left & right);
    FDRE #(.INIT(1'b0)) right_break (.C(clk), .R(1'b0),
.CE(1'b1), .D(Next_RIGHT_BREAK), .Q(RIGHT_BREAK));

    assign Next_ALL_BREAK = (LEFT_BREAK & left & right)
| (IDLE & left & right) |
                              (RIGHT_BREAK & left &
right) | (ALL_BREAK & left & right);
    FDRE #(.INIT(1'b0)) all_break (.C(clk), .R(1'b0),
.CE(1'b1), .D(Next_ALL_BREAK), .Q(ALL_BREAK));

    assign RunGame = !IDLE;
    assign Add = LEFT_BREAK & !left & !right &
INIT_PRESS;
    assign Subtract = RIGHT_BREAK & !left & !right &
!INIT_PRESS;
endmodule
```

```
//status: COMPLETE, EDITED FOR LAB 6
module hex7seg(//hex7seg(n [3:0]. seg
[7:0])
    input [3:0] n,
    input Negative,
    output [6:0] seg //seg[0] = a,
seg[1] = b...etc
    );
    //m8_1(in [7:0], [2:0] sel, [7:0]
out)
    wire [6:0]out;
    wire [7:0] inputs0, inputs1,
inputs2, inputs3, inputs4, inputs5,
inputs6;
    wire [2:0] selectors;
    assign selectors[0] = n[0],
selectors[1] = n[1], selectors[2] =
n[2];
    wire W; assign W = n[3];

    assign inputs0[0] = 1, inputs0[1]
```

```
= 1, inputs0[2] = !W, inputs0[3] = W,
inputs0[4] = 1, inputs0[5] = !W,
inputs0[6] = W, inputs0[7] = 0;
//my: B, vid: A
    m8_1 mux0 (.in(inputs0),
.sel(selectors), .out(out[1]));

    assign inputs1[0] = 1, inputs1[1]
= W, inputs1[2] = 1, inputs1[3] = 1,
inputs1[4] = 1, inputs1[5] = 1,
inputs1[6] = W, inputs1[7] = 0;
//my C, vid: B
    m8_1 mux1 (.in(inputs1),
.sel(selectors), .out(out[2]));

    assign inputs2[0] = !W,
inputs2[1] = 1, inputs2[2] = W,
inputs2[3] = !W, inputs2[4] = 1,
inputs2[5] = W, inputs2[6] = 1,
inputs2[7] = !W;    //my D, Vid: C
    m8_1 mux2 (.in(inputs2),
```

```
.sel(selectors), .out(out[3]));

    assign inputs3[0] = !W,
inputs3[1] = !W, inputs3[2] = 0,
inputs3[3] = !W, inputs3[4] = !W,
inputs3[5] = 1, inputs3[6] = 1,
inputs3[7] = 1;  //my E, vid: D
    m8_1 mux3 (.in(inputs3),
.sel(selectors), .out(out[4]));

    assign inputs4[0] = !W,
inputs4[1] = 0, inputs4[2] = 1,
inputs4[3] = !W, inputs4[4] = 1,
inputs4[5] = 1, inputs4[6] = !W,
inputs4[7] = 1;   //my F, vid: E
    m8_1 mux4 (.in(inputs4),
.sel(selectors), .out(out[5]));

    assign inputs5[0] = !W,
inputs5[1] = 1, inputs5[2] = W,
inputs5[3] = 1, inputs5[4] = 1,
```

```
inputs5[5] = !W, inputs5[6] = !W,
inputs5[7] = 1;      //my A, vid: F
    m8_1 mux5 (.in(inputs5),
.sel(selectors), .out(out[0]));

   assign inputs6[0] = 0, inputs6[1]
= 1, inputs6[2] = 1, inputs6[3] = !W,
inputs6[4] = 1, inputs6[5] = 1,
inputs6[6] = W, inputs6[7] = 1;
//my G, vid: G
    m8_1 mux6 (.in(inputs6),
.sel(selectors), .out(out[6]));

    assign seg[0] = (out[0] |
Negative), seg[1] = (out[1] |
Negative), seg[2] = (out[2] |
Negative), seg[3] = (out[3] |
Negative),
                    seg[4] = (out[4]
| Negative), seg[5] = (out[5] |
Negative), seg[6] = (out[6] &
```

```
!Negative);
endmodule
```

```
//status: COMPLETE
module Selector(
  input [3:0] sel,
  input [15:0] N,
  output [3:0] H
  );

  assign H = ( sel[3] & !sel[2] &
!sel[1] & !sel[0]) ? N[15:12]:
//3
            (!sel[3] &  sel[2] &
!sel[1] & !sel[0]) ? N[11:8] :
//2
            (!sel[3] & !sel[2] &
sel[1] & !sel[0]) ? N[7:4]  :
//1
            (!sel[3] & !sel[2] &
!sel[1] & sel[0])  ? N[3:0]  : 0;
//0
  assign NegDisp = (!sel[3] &  sel[2]
& !sel[1] & !sel[0]);
```

```
//status: OPERATIONAL
module RingCounter(
    input digsel,
    input clk,
    output [3:0]Q,
    output second
    );
    //wire start;
    FDRE #(.INIT(1'b1)) ff1 (.C(clk),
.R(1'b0), .CE(digsel), .D(Q[0]),
.Q(Q[3]));
    FDRE #(.INIT(1'b0)) ff2 (.C(clk),
.R(1'b0), .CE(digsel), .D(Q[3]),
.Q(Q[2]));
    FDRE #(.INIT(1'b0)) ff3 (.C(clk),
.R(1'b0), .CE(digsel), .D(Q[2]),
.Q(Q[1]));
    FDRE #(.INIT(1'b0)) ff4 (.C(clk),
.R(1'b0), .CE(digsel), .D(Q[1]),
.Q(Q[0]));
    assign second = Q[0];
```

```
//status: SEEMING TO WORK
module Edge_Detector(
    input clk, input btn,
    output out
);
    wire feed, temp;
   FDRE #(.INIT(1'b0)) flip_flop1
(.C(clk), .R(1'b0), .CE(1'b1),
.D(btn), .Q(feed));
   assign temp = (btn & !feed);
//src1
   FDRE #(.INIT(1'b0)) flip_flop2
(.C(clk), .R(1'b0), .CE(1'b1),
.D(temp), .Q(out));
endmodule
```

```
//status: OPERATIONAL
module TimeCounter(
    input clk,
    input CE,
    input Qsec,
    input Reset,
    output Signal
    );
    wire [3:0]Q;
    wire utc;
    wire active = CE & Qsec;
    countUD4L counter1 (.Up(active),
.Dw(1'b0), .LD(Reset | Signal),
.Q(4'b0),
                        .UTC(utc),
.clk(clk), .Qout(Q[3:0]));


    assign Signal = Q[1];
```

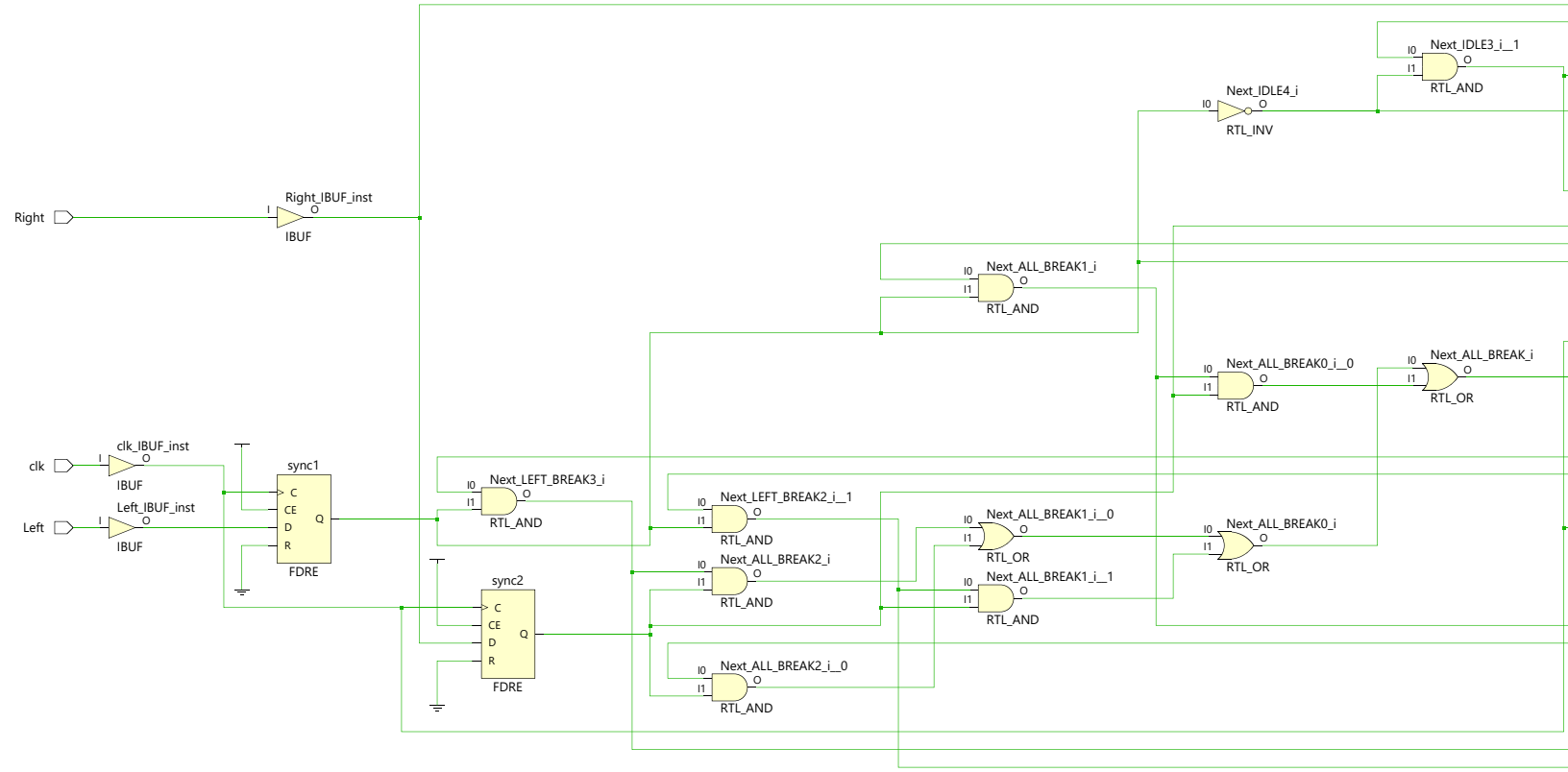
```
//status: COMPLETE
module TurkeyCounter(
    input Add,
    input Subtract,
    input clk,
    output Negative,
    output [7:0]Turkeys
);
    wire utc, utc2, dtc, dtc2, zero, MAX;
    wire [11:0] out;
    wire increment, decrement, negative;
    assign increment = (Add & !negative & !MAX) |
(Subtract &  (negative | zero) & !MAX);
    assign decrement = (Add &  negative)        |
(Subtract & !negative & !zero);
    assign MAX = out[0] & out[1] & out[2] & out[3] &
out[7] & out[6];
    countUD4L counter  (.Up(increment),
.Dw(decrement),     .LD(1'b0), .Q(4'b0),
                        .clk(clk), .UTC(utc),
.DTC(dtc), .Qout(out[3:0]));
    countUD4L counter2 (.Up(increment&utc),
.Dw(decrement&dtc), .LD(1'b0), .Q(4'b0),
                        .clk(clk), .DTC(dtc2),
.Qout(out[7:4]));

    assign zero = dtc&dtc2;
    FDRE #(.INIT(1'b0)) neg (.C(clk), .R(1'b0),
.CE(zero), .D(Subtract), .Q(negative)); //adds a
negative sign
    assign Negative = negative;
```

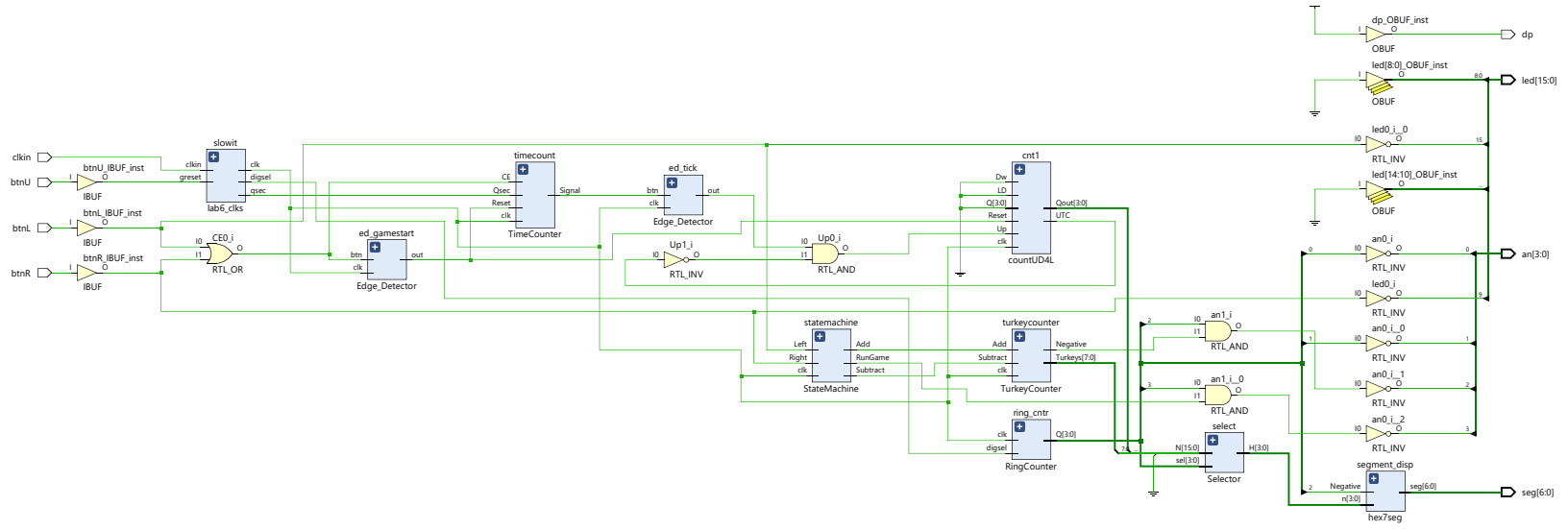
```
    assign Turkeys[4] = 1'b0;
    assign Turkeys[3:0] = out[3:0];
    assign Turkeys[7:5] = out[7:5];


endmodule
```

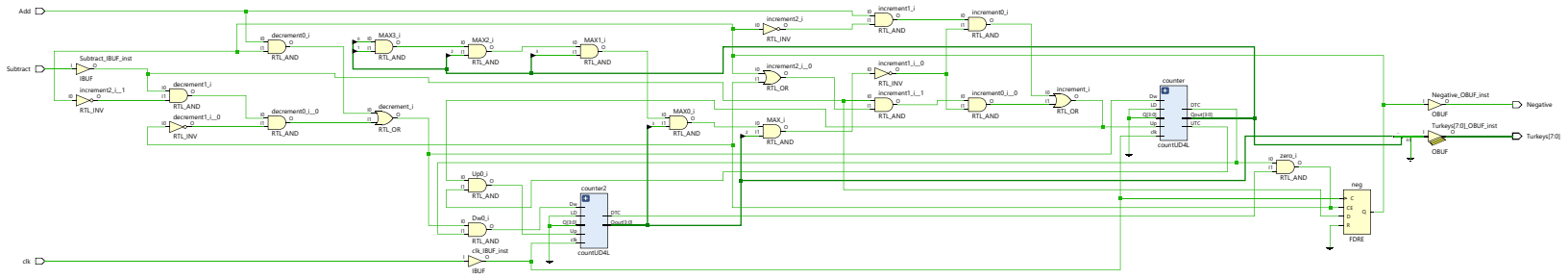
```
module Top_Module_Main(
    input btnU,
    input btnL,
    input btnR,
    input clkin,
    output dp,
    output [6:0] seg,
    output [3:0] an,
    output [15:0] led
    );
    wire qsec, digsel, clk;
    wire [15:0] bit16out;
    lab6_clks slowit (.clkin(clkin), .greset(btnU),
.clk(clk), .digsel(digsel), .qsec(qsec));

    //StateMachine
    wire run_game, add, subtract;
    StateMachine statemachine (.Left(btnL),
.Right(btnR), .clk(clk), .RunGame(run_game), .Add(add),
.Subtract(subtract));

    //TimeCounter
    wire second, start, utc, tick;
    wire [3:0] sec;
    Edge_Detector ed_gamestart (.clk(clk), .btn(btnL |
btnR), .out(start));
    TimeCounter timecount (.clk(clk), .CE(btnL | btnR),
.Qsec(qsec), .Reset(start), .Signal(second));
    Edge_Detector ed_tick (.clk(clk), .btn(second),
.out(tick));
    countUD4L cnt1 (.Up(tick&!utc), .LD(1'b0),
```

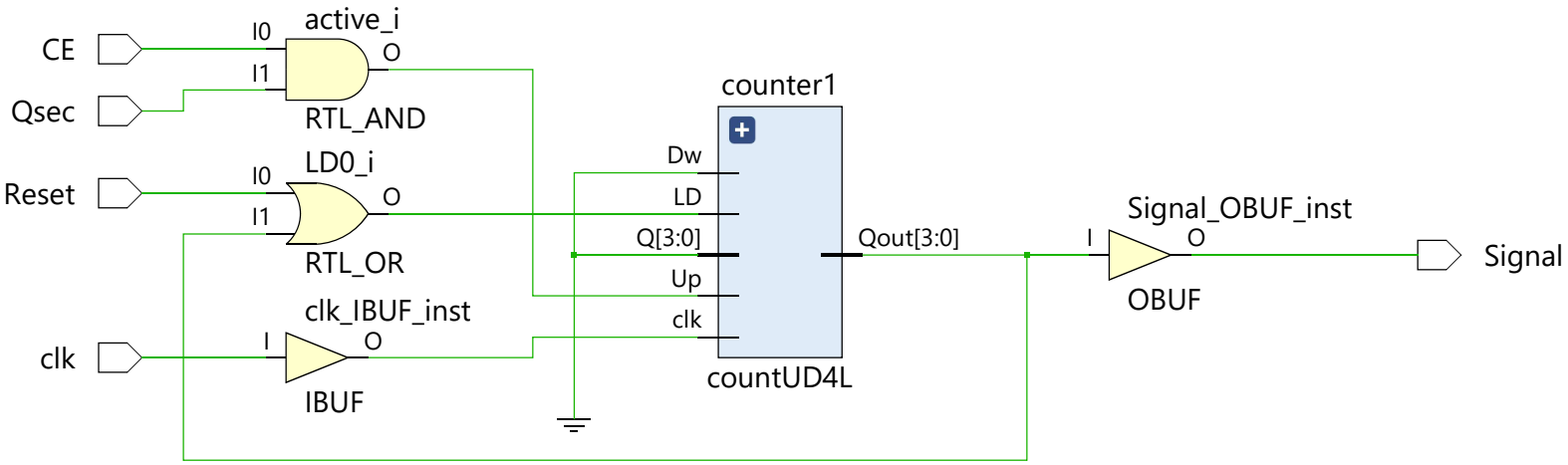
```
.Dw(1'b0), .Q(4'b0), .clk(clk), .UTC(utc),
.Reset(start), .Qout(sec[3:0]));
    assign bit16out[15:12] = sec;

    //TurkeyCounter
    wire negative;
    TurkeyCounter turkeycounter (.Add(add),
.Subtract(subtract), .clk(clk), .Negative(negative),
.Turkeys(bit16out[7:0]));

    //logic for LED
    assign led[8:0] = 9'b000000000;
    assign led[14:10] = 5'b00000;
    assign led[9] = !btnR;
    assign led[15] = !btnL;

    //import
    assign bit16out[11:8] = 4'b0000;
    wire [3:0]Qring; wire negdisp;
    RingCounter ring_cntr (.digsel(digsel), .clk(clk),
.Q(Qring));

    assign an[0] = !Qring[0];
    assign an[1] = !Qring[1];
    assign an[2] = !(Qring[2] & negative);
    assign an[3] = !(Qring[3] & run_game);
    assign dp = 1;
    wire [3:0]sel;
    Selector select(.sel(Qring), .N(bit16out),
.H(sel));
    hex7seg segment_disp (.n(sel), .seg(seg),
```

```
.Negative(Qring[2])); //Negative is the display we're
on


endmodule
```


Right
Right_IBUF_inst
I
O
IBUF
clk
clk_IBUF_inst
IBUF
Left
Left_IBUF_inst
IBUF
sync1
C
CE
D
R
Q
FDRE
sync2
FDRE
Next_LEFT_BREAK3_i
I0
I1
RTL_AND
Next_LEFT_BREAK2_i__1
Next_ALL_BREAK2_i
Next_ALL_BREAK2_i__0
Next_ALL_BREAK1_i
Next_ALL_BREAK1_i__0
Next_ALL_BREAK1_i__1
RTL_OR
Next_ALL_BREAK0_i__0
Next_ALL_BREAK0_i
Next_ALL_BREAK_i
Next_IDLE4_i
RTL_INV
Next_IDLE3_i__1


clkin
btnU
btnL
btnR
IBUF
btnU_IBUF_inst
btnL_IBUF_inst
btnR_IBUF_inst
slowit
lab6_clks
clkin
greset
clk
digsel
qsec
RTL_OR
CE0_i
ed_gamestart
Edge_Detector
timecount
TimeCounter
CE
Qsec
Reset
clk
Signal
ed_tick
Edge_Detector
btn
out
Up1_i
RTL_INV
Up0_i
RTL_AND
cnt1
countUD4L
Dw
LD
Q[3:0]
Reset
Up
Qout[3:0]
UTC
statemachine
StateMachine
Left
Right
Add
RunGame
Subtract
turkeycounter
TurkeyCounter
Negative
Turkeys[7:0]
ring_cntr
RingCounter
Q[3:0]
an1_i
an1_i_0
RTL_AND
select
Selector
N[15:0]
sel[3:0]
H[3:0]
dp_OBUF_inst
dp
led[8:0]_OBUF_inst
led[15:0]
led[14:10]_OBUF_inst
led0_i_0
led0_i
an0_i
an0_i_0
an0_i_1
an0_i_2
an[3:0]
segment_disp
hex7seg
n[3:0]
seg[6:0]


Add
Subtract
clk
Subtract_IBUF_inst
IBUF
increment2_i_1
RTL_INV
decrement1_i
RTL_AND
decrement1_i_0
RTL_INV
decrement0_i
RTL_AND
decrement0_i_0
RTL_AND
decrement_i
RTL_OR
MAX3_i
RTL_AND
MAX2_i
RTL_AND
MAX1_i
RTL_AND
MAX0_i
RTL_AND
increment2_i
RTL_INV
increment2_i_0
RTL_OR
MAX_i
RTL_AND
increment1_i
RTL_AND
increment1_i_0
RTL_INV
increment1_i_1
RTL_AND
increment0_i
RTL_AND
increment0_i_0
RTL_AND
increment_i
RTL_OR
Up0_i
RTL_AND
Dw0_i
RTL_AND
clk_IBUF_inst
IBUF
counter2
countUD4L
counter
countUD4L
zero_i
RTL_AND
neg
FDRE
Negative_OBUF_inst
OBUF
Negative
Turkeys[7:0]_OBUF_inst
OBUF
Turkeys[7:0]


CE
Qsec
Reset
clk
I0
I1
active_i
O
RTL_AND
LD0_i
RTL_OR
clk_IBUF_inst
I
IBUF
counter1
Dw
LD
Q[3:0]
Up
clk
countUD4L
Qout[3:0]
Signal_OBUF_inst
OBUF
Signal