

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

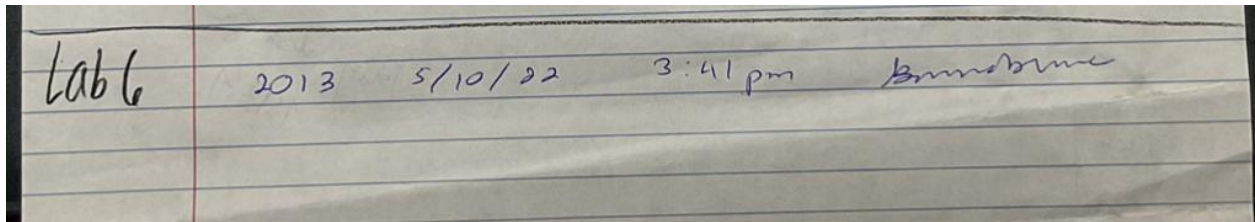
- **IDLE**: In this state, the machine is idling in the same state until input is given. The sensors are unbroken and nothing is being touched. This state is reached either in a loop or whenever the most recent press is depressed
- **LEFT_BREAK**: In this state, only the left button is pressed. When 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.