Hardware Design and Lab

Lab6 Team32 Group Report

Name: 林奕為

Contribution: Slot Machine, Car

Contents

What we have learned from Lab 6

林奕為 ：

In this Lab, I failed to pass the demonstration of Slot Machine, it works out fine to run in an upward and downward direction and can be operated successively, except that for each time you need to press the same button for exactly twice to operate.

During this lab, we have many conditions, one of the FPGA burned, the car's two motor are nearly disabled, the wires are of wrong type so none of them can slot into the car, my computer's OS have issues regarding recent updating that I failed to connect my FPGA board until I roll it back to older version I succeeded etc.
I should figure it out if time permits, but the demonstration is closed anyway. It taught me to strictly follow the spec when designing Verilog code that I could correct the error beforehand to avoid issues and not to take chances during Demo.

Regarding Car, my design successfully passes the basic and bonus track with the help of my teammate so that I'm not doing nothing in this team lab.

In this Lab, I implemented Verilog in peripheral devices and use its feedback signal to further the design scope of FPGA project, it's good to see that my Verilog code can operate on Super sonic devices, IR devices and motor etc. which should help me to facilitate my development of final project.

賴彥丞:

The problem we have is our car has a very weak motor and the cables are not consistent or reliable that what initially we thought was the problem with our code is the hardware components problem, this took us a lot of time. The problem is solved once we borrowed another team's car. I learned some tips and tricks in dealing with the cable issues as well.

I have learned how to set up connections between chips. This kind of remind be of internet protocols. For consistent data transferring, it's not good to just treat both ends like a simple in and out port. It's better to have some validity check to prevent bad connections.

Slot Machine

Design of the Module:

I define 3 states: STATE_IDLE, STATE_UP and STATE_DOWN respectively, when the state Is idle, it means that it's not moving. When pressing one of the buttons, it moves to state_up or state_down, the only difference is how the calculation of the memory address.

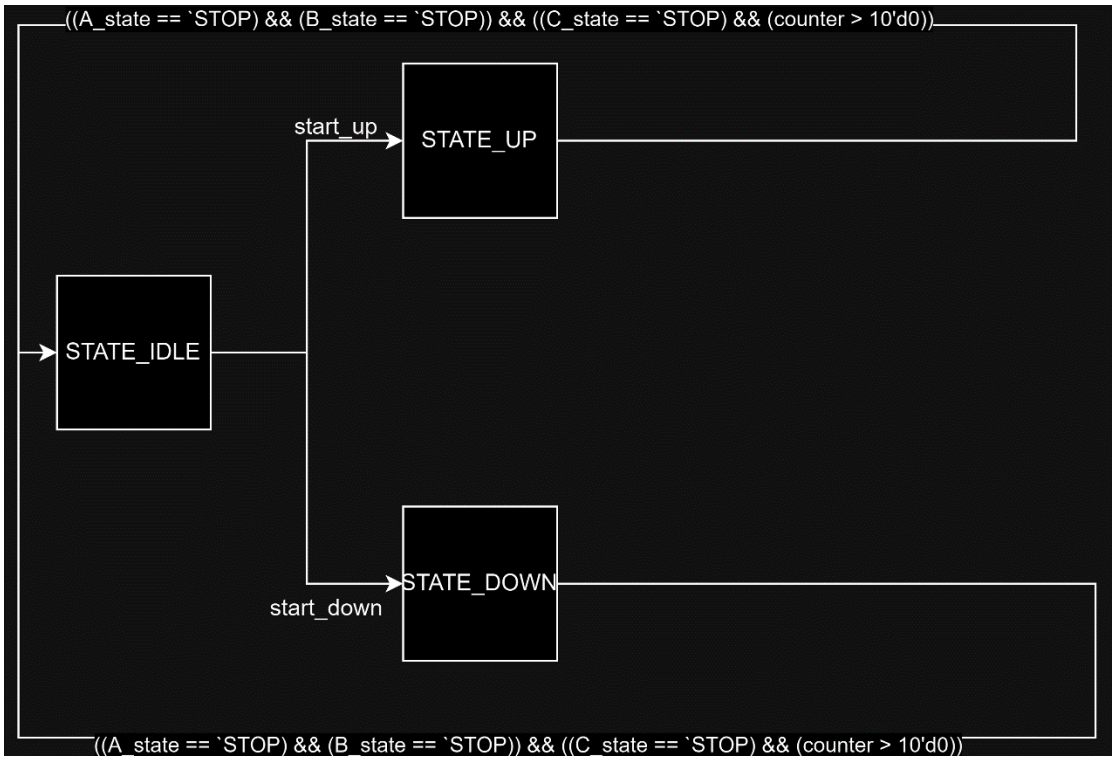
```
assign v_cnt_total = (state == `STATE_UP) ? (v_cnt + v_mem) : (v_cnt +
(16'd239 - v_mem));
```

when stop moving, it will go back to the idle state to wait for the next trigger.

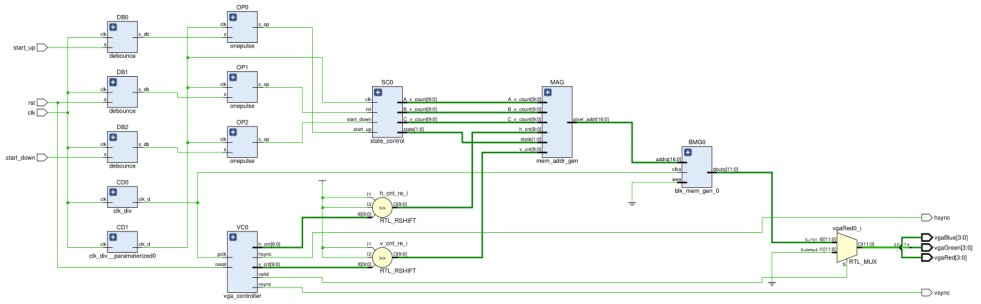
Method of Testing:

I use the screen and FPGA to test my design physically, I observe the moving pattern (whether digits move correctly) and the responsiveness of the button when pressing them, I didn't use a testbench this time.

State Transition Diagram:



Block Diagram:



CAR

Design of the Module:

I define 3 states: TURN_LEFT, TURN_RIGHT and RUNNING, in running state it runs straight with 2 motors with the value of 10'd1000, in TURN_LEFT the left motor is 10'd500 with backward direction and the right motor is 10'd1000 with forward direction, and TURN_RIGHT is similar despite having opposite value and direction of two motors.

If stop signal detected which means dis < 20'd4700, the car will stop otherwise continue moving.

Specifically in IR detection logic, when both left signal and right signal are both equals 1'b0 the car will move the same as the previous state, otherwise the car will turn to left or right if needed.

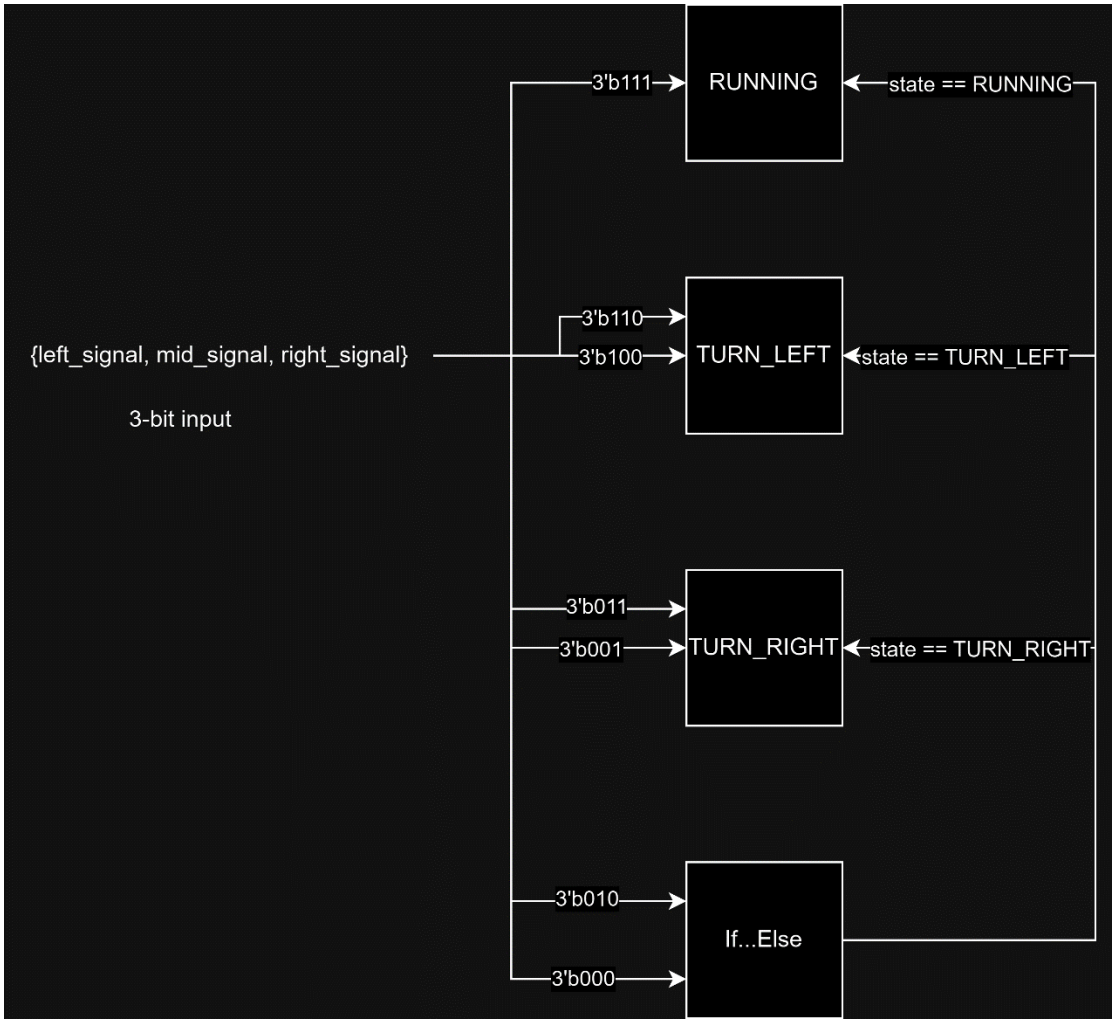
```
always @(*) begin
        next_state = state;
        case({left_signal, mid_signal, right_signal})
            3'b111: next_state = RUNNING;
            3'b110: next_state = TURN_LEFT;
            3'b100: next_state = TURN_LEFT;
            3'b011: next_state = TURN_RIGHT;
            3'b001: next_state = TURN_RIGHT;
            3'b010: begin
                if(state == TURN_LEFT) next_state = TURN_LEFT;
                else if(state == TURN_RIGHT ) next_state = TURN_RIGHT;
                else next_state = RUNNING;
            end
            3'b000: begin
                if(state == TURN_LEFT) next_state = TURN_LEFT;
                else if(state == TURN_RIGHT ) next_state = TURN_RIGHT;
                else next_state = RUNNING;
            end
            default: next_state = RUNNING;
        endcase
    end
```

Method of Testing:

I test the car with physical modules on spot to see if the behavior of the car meets our requirements such as running on the track correctly and successfully detect the distance within 40cm etc. If not, we will either modify the code, check if the wires connect correctly or check if the .xdc files is written right or wrong etc. (From both software and hardware perspective).

Additionally, I add a 4-bit LED signal to represent 3 state, the 3 state is previous mentioned RUNNING, TURN_LEFT and TURN_RIGHT, and last bit of LED represents if the car stopped.

State Transition Diagram:



Block Diagram: