Dylan Thornburg, Alan Rieger
5-23-2023

Lab 7: Counters

Introduction:
For this lab we used dual counters to make a game where the goal is to stop the counters when they are on the same number. We did not use the FPGA for this lab.

Procedures:
This lab was relatively simple compared to the previous one. Almost all the verilog was done in the prelab and all we had to do was connect the modules and run the simulation. We had some problems getting our verilog correct, but we eventually got it.

Questions:
If you were to change your design to have the counters count from 2 to 6 (or 6 to 2) instead of only 1 to 5, list all the things you would need to change.

We would need to change the binary numbers from 1-5 to 2-6 in the UpControl and DownControl modules. This would solve the problem and count from 2-6 instead while still having 5 different possible values.

Conclusion:
Through this lab, we learned how to utilize counters to make a simple game and how to do it all in verilog. We had a couple hiccups along the way including a schematic and a verilog error but all were fixed and now we know for a circuit similar to this one, one counter must have a vcc and the other have ground and not both have ground being inputted. Overall, the lab was interesting to see how counters worked in tandem.

Figure 1: Circuit Schematic

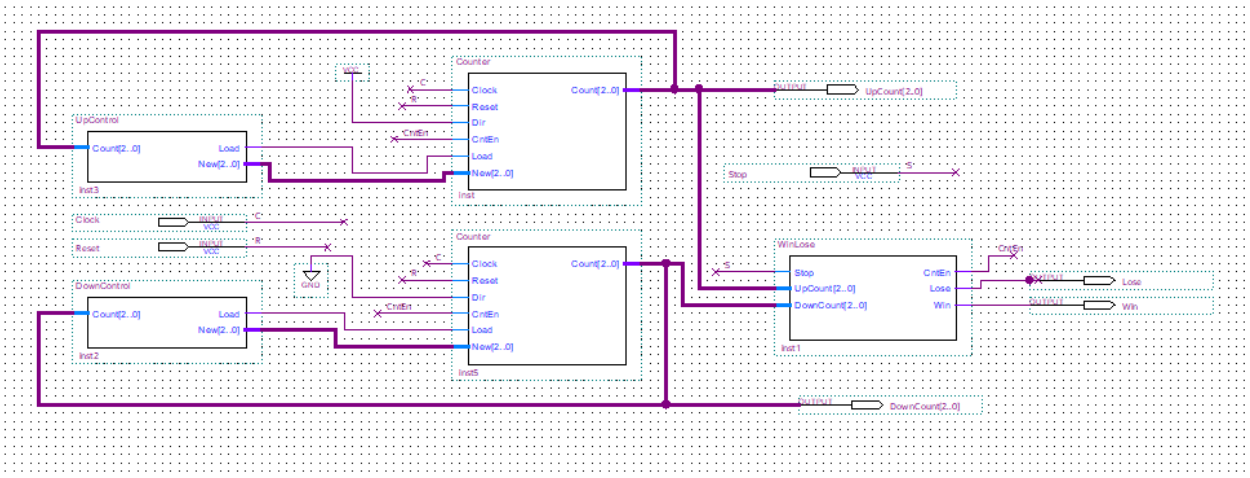

Figure 2: Verilog Code

```
// Counter has been completed for you.
module Counter(
        input Clock,
        input Reset,
        input Dir,
        input CntEn,
        input Load,
        input [2:0] New,
        output reg [2:0] Count);
        always @(posedge Clock) begin
                if (Reset == 1)
                Count <= 3'b001;
                else begin
                if (CntEn == 1) begin
                if (Load == 1)
                        Count <= New;
                else begin
                        if (Dir == 1)
                                Count <= Count + 1;
                        else
                                Count <= Count - 1;
                        end
                end
        end
end
endmodule
// UpControl has been completed for you as well.
```

```
module UpControl(
	input [2:0] Count,
	output Load,
	output [2:0] New);
	// The following statement means if Count equals 5,
	// set Load equal to 1, otherwise set it equal to 0.
	assign Load = (Count == 3'b101) ? 1'b1 : 1'b0;
	assign New = 3'b001;
endmodule
// Fill in DownControl. Note that because the signal
// names are internal to the module, you can reuse
// the same names as in UpControl for simplicity.
module DownControl(
	input [2:0] Count,
	output Load,
	output [2:0] New);
	assign Load = (Count == 3'b001) ? 1'b1 : 1'b0;
	assign New = 3'b101;
endmodule
// WinLose has been partially completed for you.
module WinLose(
	input Stop,
	input [2:0] UpCount,
	input [2:0] DownCount,
	output reg CntEn,
	output reg Lose,
	output reg Win);
	// Fill in the logic for the always block.
	// Refer to Counter for a guide on syntax.
	always @(UpCount or DownCount or Stop) begin
		if(Stop==0)begin
			CntEn=1;
			Win=0;
			Lose=0;
			end
		else begin
			if(UpCount == DownCount)begin
				CntEn=0;
				Win=1;
				Lose=0;
```

```
                    end
                    else begin
                            CntEn=0;
                            Win=0;
                            Lose=1;
                    end
            end
        end
endmodule
```

Figure 3: Waveform Diagram