State Alone Complex

Douglas Zorn

CE 125 winter 2014

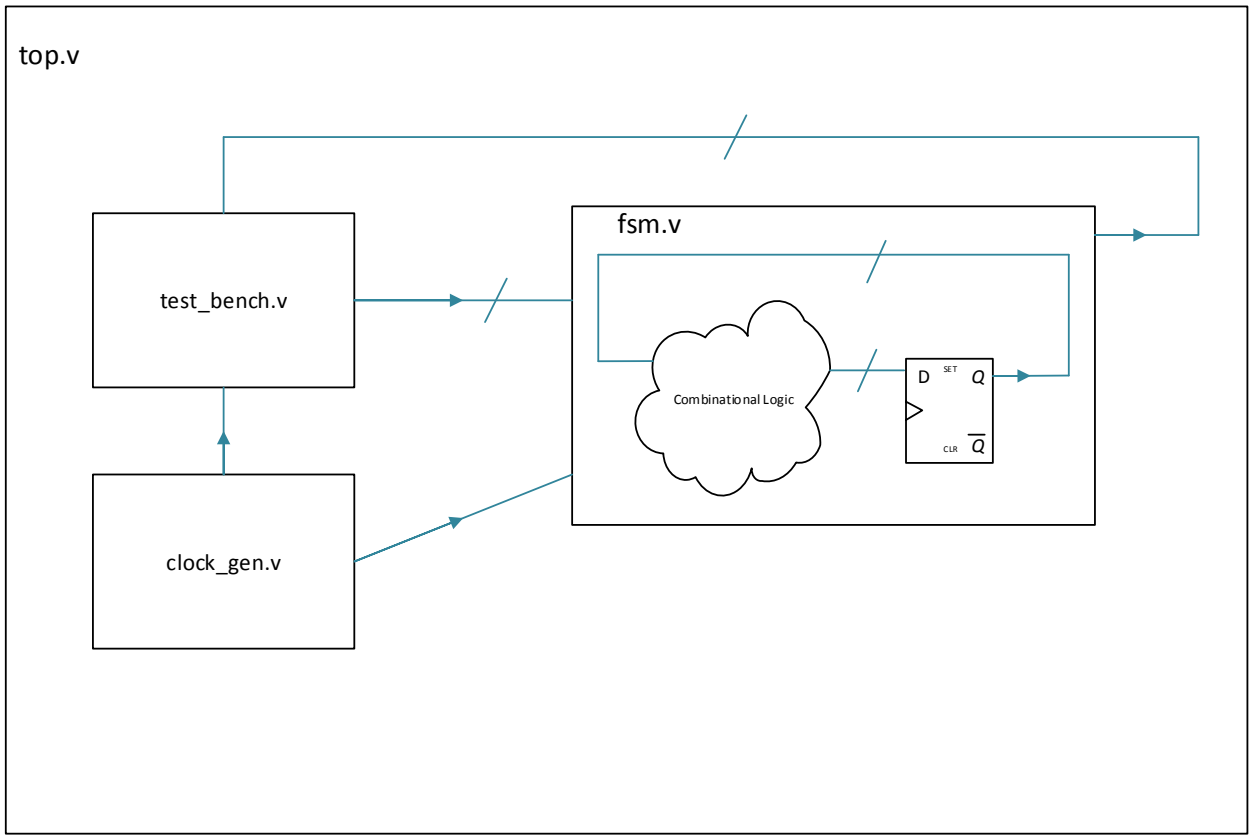
Intro

Lab 3 tasked us with creating a finite state machine that implemented the logic of a 4-way intersection's traffic lights. In this lab we utilized the *synthesis parallel_case* directive to evaluate our combinational logic in parallel instead of sequentially. This was possible due to the use of one-hot encoding of the case statement, which meant that each case was mutually exclusive.

The view from the clouds

There were four modules in total for lab 3. Figure 1 below shows the hierarchy of the modules.

Figure 1



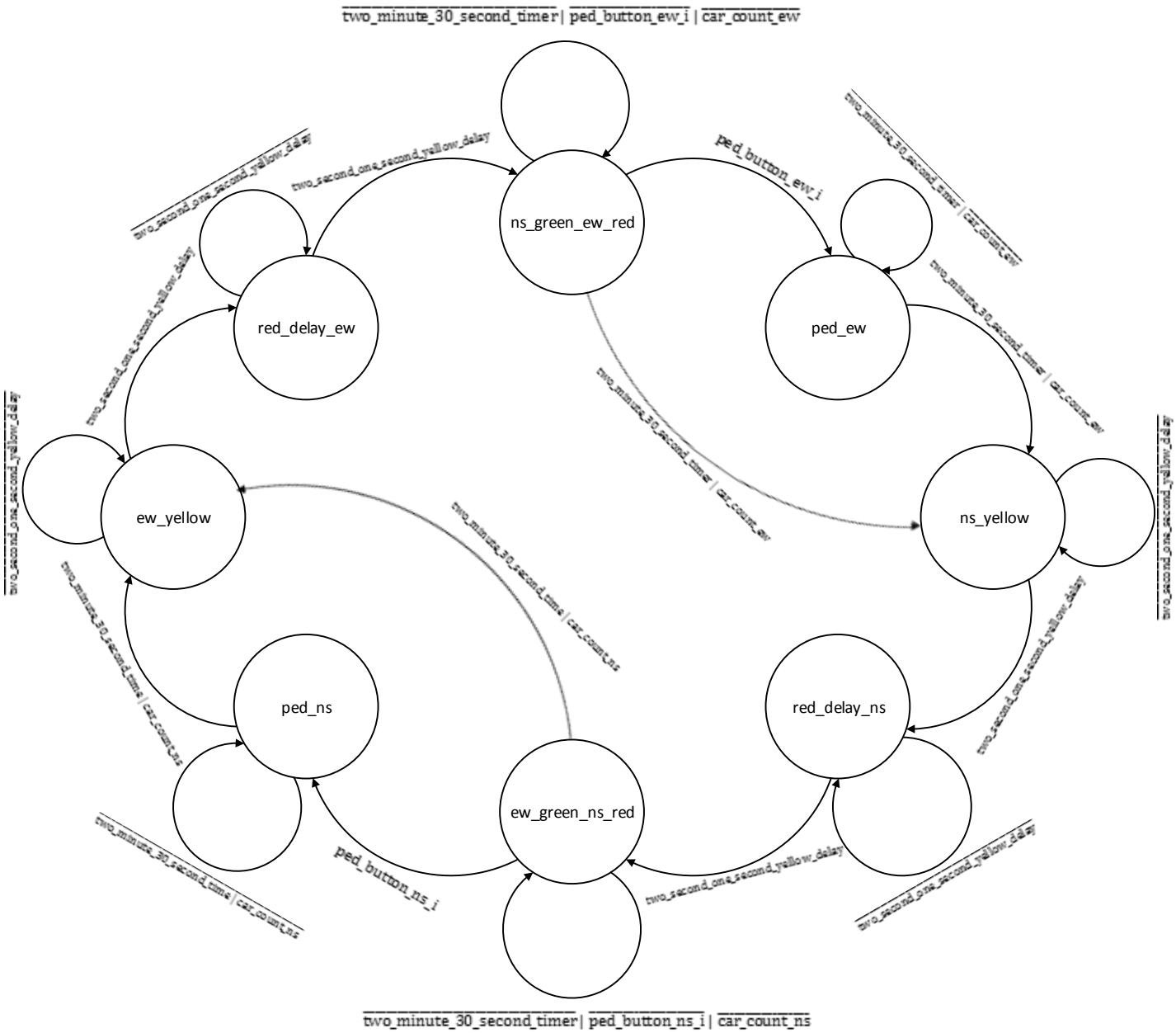
Test_bench.v triggered the initial resets for all the registers. The module also toggled the inputs to fsm.v. Lastly, fsm.v fed its outputs back into test_bench.v so that the values could be printed to an output file for review during debugging and testing.

Clock_gen.v only served to generate a 1 kHz clock to test_bench.v and clock_gen.v. It was convenient to use the parameterized clock I made during lab 2.

Top.v instantiated the three modules, fsm.v, test_bench.v, and clock_gen.v. Top.v needed to exist in order to debug and keep fsm synthesizable.

Fsm.v was the finite state machine for intersection traffic light logic. A picture is worth a thousand words, so it's easiest to understand the logic of the state machine by studying its state diagram. This was my third iteration of the state diagram copied from my lab notebook, almost verbatim. See figure 2 below of the state diagram.

Figure 2



I apologize for the clarity of the text. The program had difficulty converting the diagram to Word and as a result some of the smaller text was blurred. The state machine was implemented using the inverted case method with one-hot encoding. As seen from the diagram there was a total of eight states. This implementation allowed the north/south and east/west states to utilize the same timers.

Moore and Mealy

All but one of the outputs of the fsm module were Moore outputs. Table 1 below describes what state the outputs were functions of.

Table 1

Output	**States**				
green_northsouth_o	ns_green_ew_red	ped_ew			
red_northsouth_o	red_delay_ns	ew_green_ns_red	red_delay_ew	ew_yellow	ped_ns
yellow_northsouth_o	ns_yellow				
green_eastwest_o	ew_green_ns_red	ped_ns			
red_eastwest_o	ns_green_ew_red	red_delay_ew	ns_yellow	red_delay_ns	ped_ew
yellow_eastwest_o	ew_yellow				

The last output, *transition_count_o*, was a counter that incremented every time the north/south green light was triggered. I am not sure if this would be explicitly described as a Mealy output because it was a function of the state and also a timer internal to the fsm module, as opposed to external logic. The condition when the timer incremented was during the red_delay_ew state and the delay timer, *two_second_one_second_yellow_delay,* at specific value.

Conclusion

State machine design starts in a notebook. And in my case I followed this design process. As a result my state machine functioned as I had written it in my notebook. This lab as a result of that design process was not very difficult. I used the low difficulty as an excuse to brush up on my git skills. Looking back at all commits I made really illustrates the whole Verilog process, from notebook to code, as I created each module and it functioned as I had intended to.