Eugene Ngo : 1965514

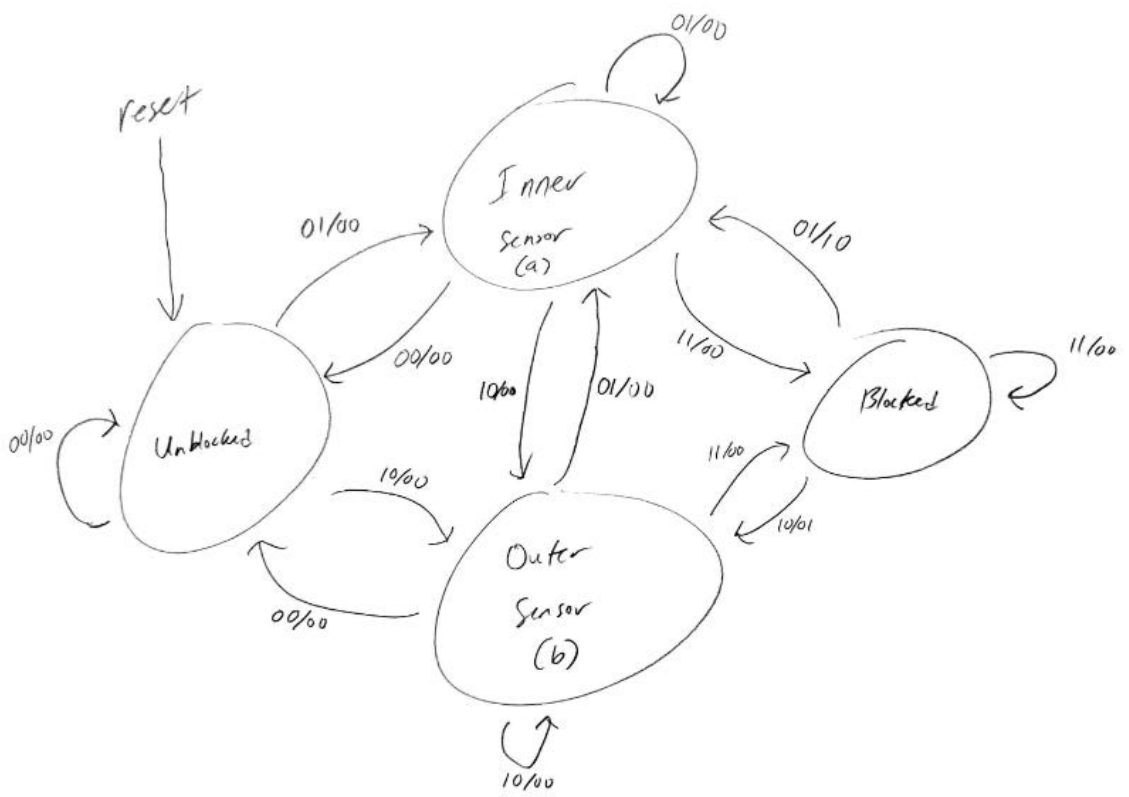
1/13/2023

EE 371 - Lab 1 Report

Procedure

Lab 1 was a one-part lab that refreshed us on creating finite state machines (FSMs) using Quartus. The objective of this lab was to create a parking lot occupancy counter. The GPIO pins on the DE1_SoC were hooked up to switches which represented the inputs from two parking lot sensors. These inputs were also displayed on LEDs. Using the switch inputs, parking lot behaviors could be emulated, such as cars entering and exiting the parking lot. When cars exited or entered the parking lot, the parking lot occupancy counter kept track of the car count. This car count was then displayed on the in-built seven segment display in the DE1_SoC.

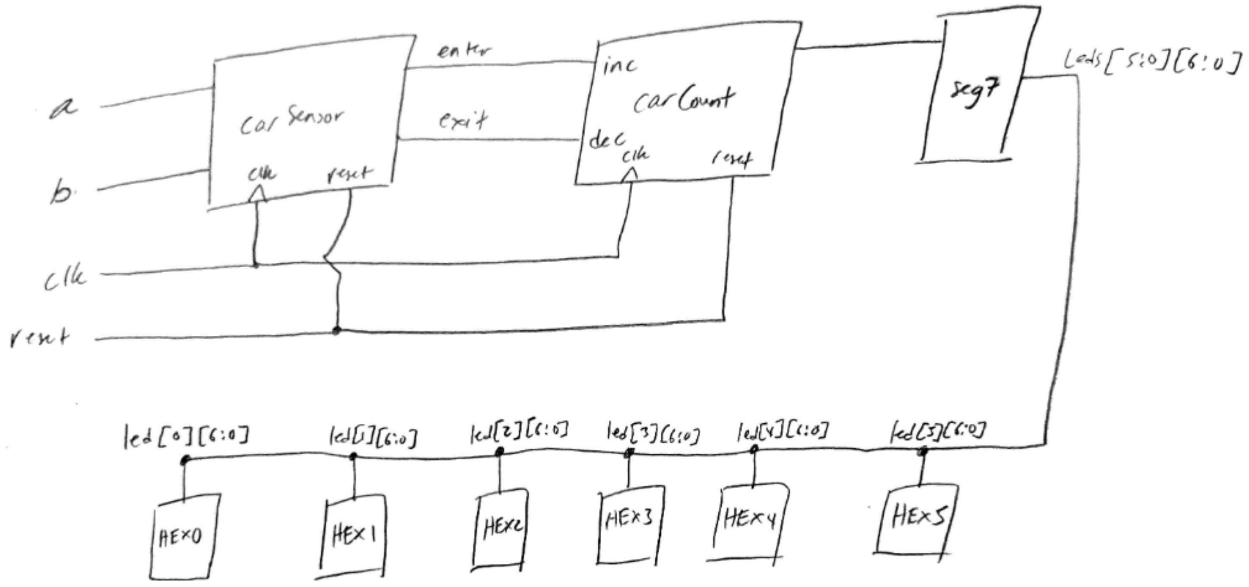
State Diagram:



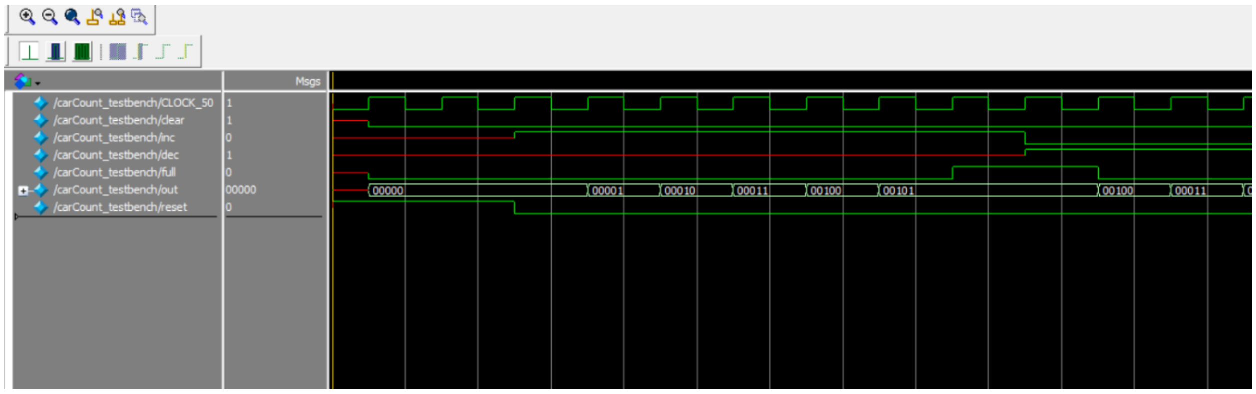
The expected behavior of the parking lot counter was implemented using a finite state machine. Inputs from inner and outer sensors would result in state changes and eventual outputs of a vehicle exiting or entering the parking lot. It was determined that only cars could block both sensors. Thus, the output of the system only adds or removes from the car count when the system transitions from blocked to inner or blocked to outer. The FSM design can be seen above.

The enter and exit output signals from the car detection finite state machine was passed onto a counting unit. The counting unit increments the current count of cars if the enter signal is true and if the current count is below the parking capacity which is 25. The counting unit decrements if the exit signal is true and if the current count is above 0. This design choice was made because car counts outside of this range are not possible and should be ignored. Additionally, it is impossible for both the enter and exit signals to be true at once, thus this case was not considered or implemented. After the counter increments or decrements the count, the count is output. The count value that is generated by the counter is then passed through to the seven segment display unit. The count value is converted from the 5-bit binary encoding to a 7-bit binary encoding for the seven segment display. A count of 0 is converted to "Clear 0". 25 is converted to "Full 25". All other values are converted to numbers they represent.

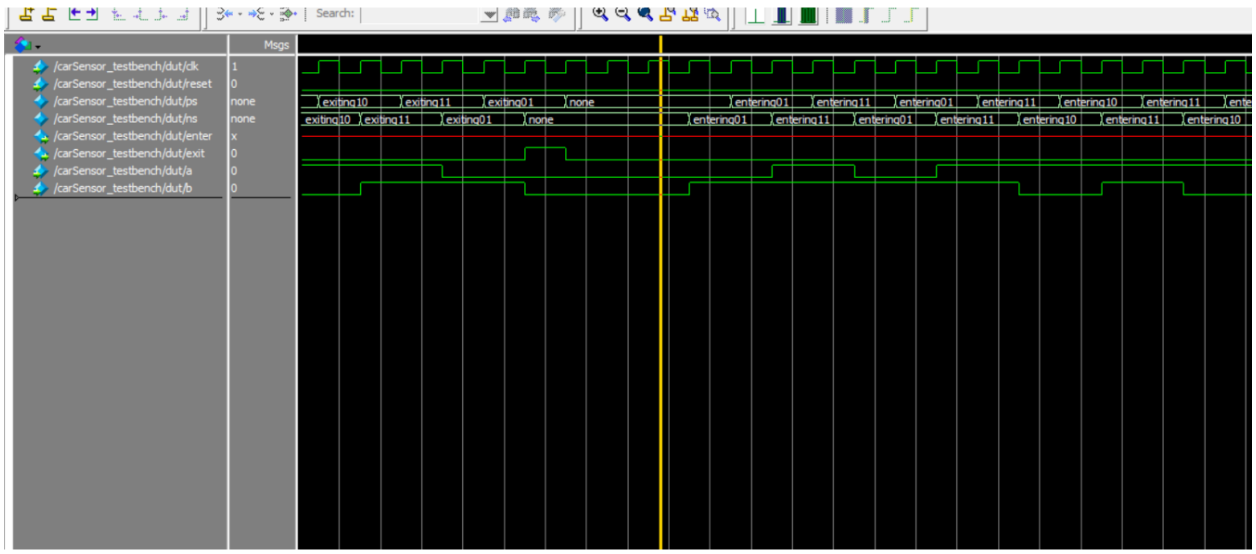
Block Diagram

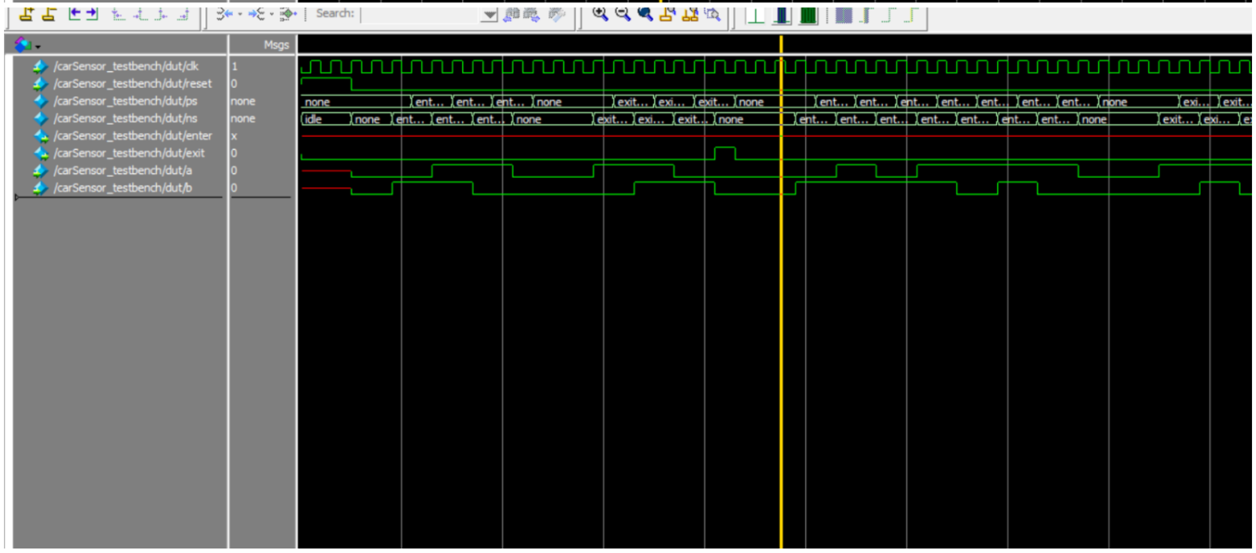


Results

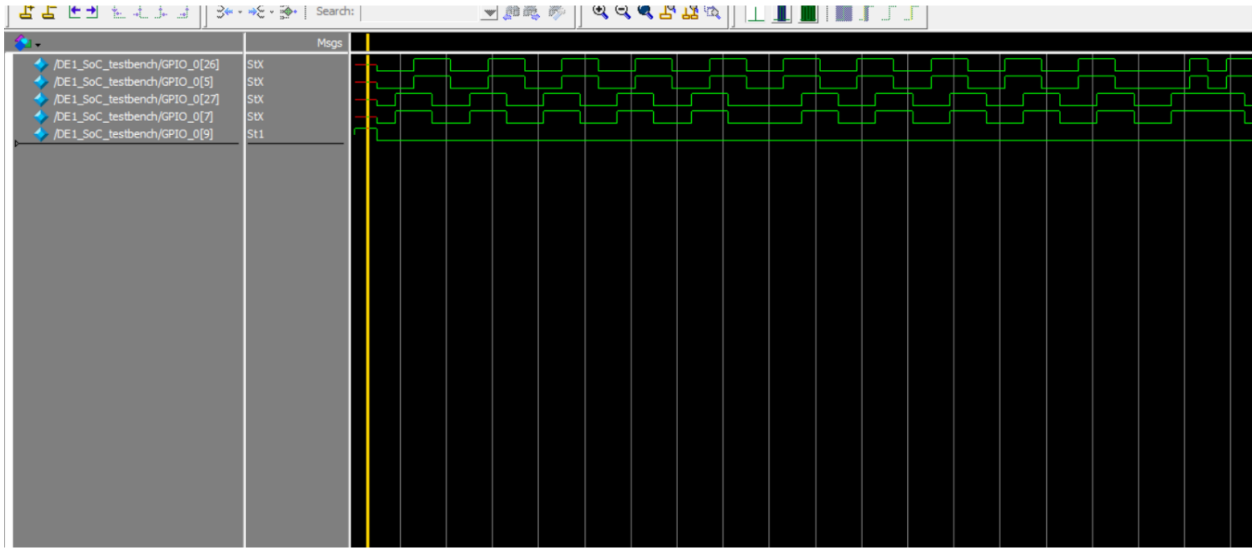

Msgs
/carCount_testbench/CLOCK_50 1
/carCount_testbench/clear 1
/carCount_testbench/inc 0
/carCount_testbench/dec 1
/carCount_testbench/full 0
/carCount_testbench/out 00000
/carCount_testbench/reset 0
00000
00001
00010
00011
00100
00101
00100
00011

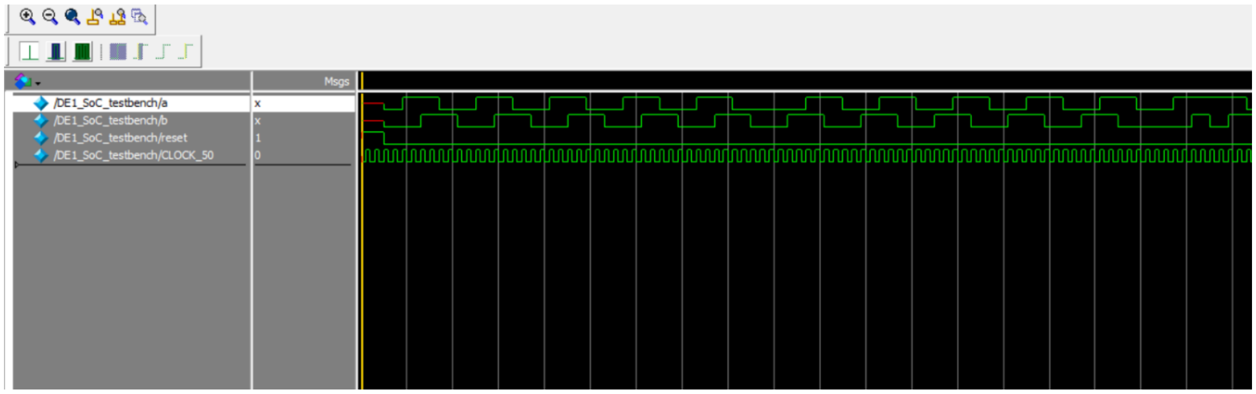
Above is the simulation for the car count which shows the count incrementing to 5 which is set to be the testbench's MAX so after incrementing 5 times, full is set to 1 and is displayed. Then the decrement is tested and decrements the output.


Search:
Msgs
/carSensor_testbench/dut/clk 1
/carSensor_testbench/dut/reset 0
/carSensor_testbench/dut/ps none
/carSensor_testbench/dut/ns none
/carSensor_testbench/dut/enter x
/carSensor_testbench/dut/exit 0
/carSensor_testbench/dut/a 0
/carSensor_testbench/dut/b 0
exiting10
exiting11
exiting01
none
entering01
entering11
entering10

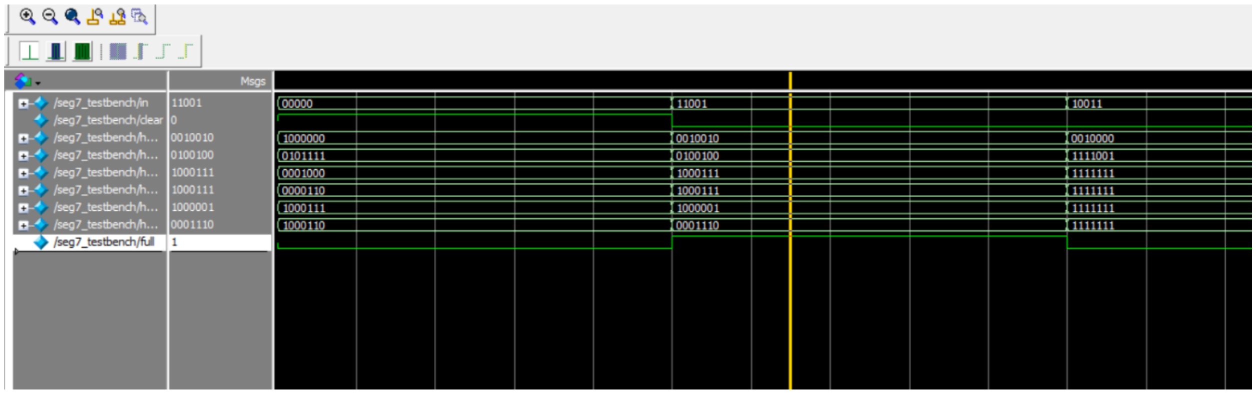

Search:
Msgs
/carSensor_testbench/dut/clk 1
/carSensor_testbench/dut/reset 0
/carSensor_testbench/dut/ps none
/carSensor_testbench/dut/ns none
/carSensor_testbench/dut/enter x
/carSensor_testbench/dut/exit 0
/carSensor_testbench/dut/a 0
/carSensor_testbench/dut/b 0
idle
none
ent...
exit...

Above are two screenshots of the carSensor showing the present state and next state changing appropriately according to the “enter” and “exit” inputs.


Msgs
/DE1_SoC_testbench/GPIO_0[26] StX
/DE1_SoC_testbench/GPIO_0[5] StX
/DE1_SoC_testbench/GPIO_0[27] StX
/DE1_SoC_testbench/GPIO_0[7] StX
/DE1_SoC_testbench/GPIO_0[9] St1

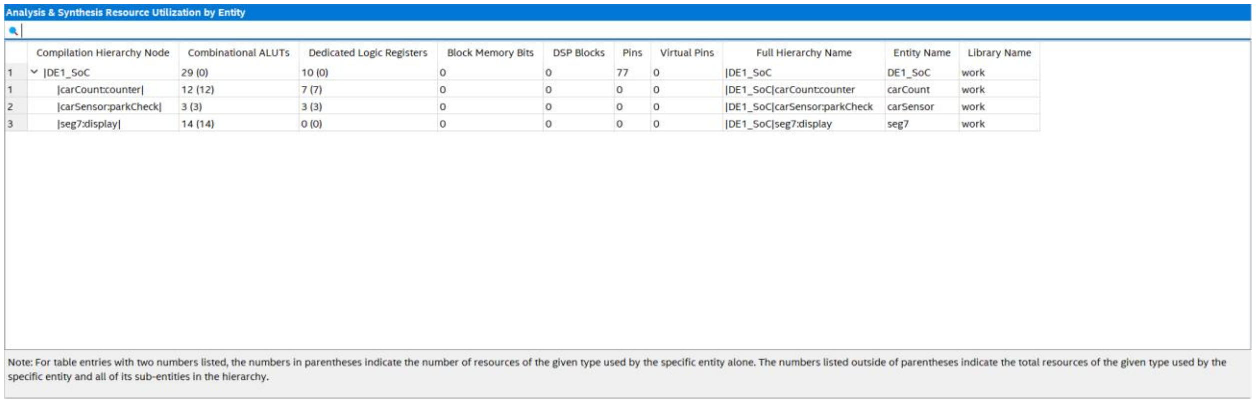


Above are two DE1_SoC screenshots. One shows the GPIO switches and LEDs working together: GPIO 5 switch corresponds to the GPIO 26 LED and the GPIO 7 switch corresponds to the GPIO 27 LED. The second screenshot shows the input signals, a and b, functioning as expected alongside the reset input and the clock.


Msgs
/seg7_testbench/in 11001 00000 11001 10011
/seg7_testbench/clear 0
/seg7_testbench/h... 0010010 1000000 0010010 0010000
/seg7_testbench/h... 0100100 0101111 0100100 1111001
/seg7_testbench/h... 1000111 0001000 1000111 1111111
/seg7_testbench/h... 1000111 0000110 1000111 1111111
/seg7_testbench/h... 1000001 1000111 1000001 1111111
/seg7_testbench/h... 0001110 1000110 0001110 1111111
/seg7_testbench/full 1

Above is the seg7 simulation showing 3 cases: ‘clear’ output, ‘full’ output, and a regular numbered output.

Resource Utilization



Analysis & Synthesis Resource Utilization by Entity

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	DSP Blocks	Pins	Virtual Pins	Full Hierarchy Name	Entity Name	Library Name
1	|DE1_SoC	29 (0)	10 (0)	0	0	77	0	|DE1_SoC	DE1_SoC	work
1	|carCount:counter|	12 (12)	7 (7)	0	0	0	0	|DE1_SoC|carCount:counter	carCount	work
2	|carSensor:parkCheck|	3 (3)	3 (3)	0	0	0	0	|DE1_SoC|carSensor:parkCheck	carSensor	work
3	|seg7:display|	14 (14)	0 (0)	0	0	0	0	|DE1_SoC|seg7:display	seg7	work

Note: For table entries with two numbers listed, the numbers in parentheses indicate the number of resources of the given type used by the specific entity alone. The numbers listed outside of parentheses indicate the total resources of the given type used by the specific entity and all of its sub-entities in the hierarchy.

Overview

This lab was a good refresher on FSMs, the design process and Verilog as a whole. The lab specifications were clear and the material for teaching how to use Labsland was useful. I did not run into any major hang ups while designing the FSM or implementing the FSM, and that is likely because I read the lab spec carefully at the beginning and the lab spec was well written and detailed which helped me design the finite state machine out on paper, and only began implementing the unit in Verilog after I was confident about the logic. Additionally, referencing material from 271 and refreshing myself helped me build the hex display modules. The provided GPIO sheet was quite helpful as well.

Overall, the system works according to how the specification wanted.

Appendix

See following code