Cyclone IV EP4CE115F29 device

Ludovic Provost 300208450

Dexter Day 300192263

Group #16

Laboratory 3: Traffic Light Controller

CEG3155: Digital Systems II

Prof. Rami Abielmona

Due November 8th 2023

Objectives:

The objective of this laboratory is to design and build a synchronous sequential machine in VHDL. Upon completion, the student must be able to:

- Design, realize, and test a finite state machine;
- Design, realize, and test a traffic light controller;
- Demonstrate a complete understanding for synchronous sequential machine design.

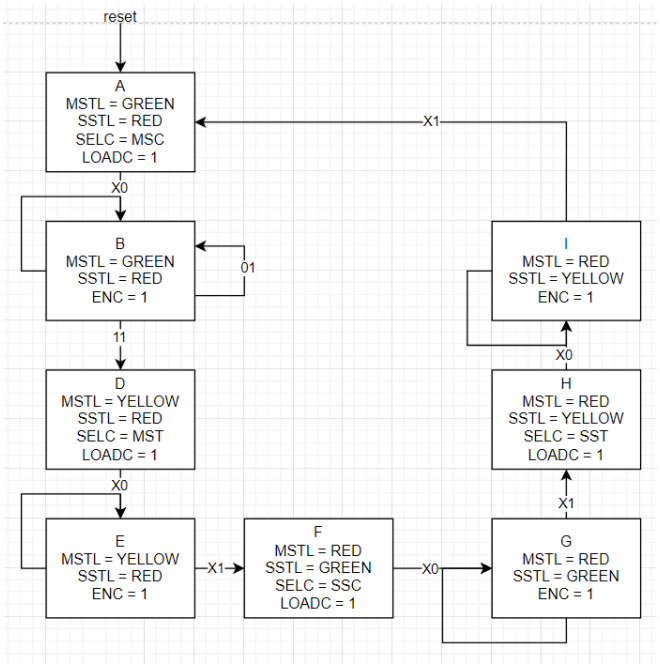
Problem and Solution Discussion:

As was mentioned in the lab manual, groups were free to choose between using a Mealy or Moore machine. We chose to go with a Moore machine, which did leave us with more states (8 instead of 4), but the transitions were less complicated. It is easy to notice that 1 state for Mealy gives 2 states for Moore for this specific problem, because each of the following 4 states must generate 2 different sets of outputs:

1. MSTL = green, SSTL = red
2. MSTL = yellow, SSTL = red
3. MSTL = red, SSTL = green
4. MSTL = red, SSTL = yellow

Since each of the 4 must generate 2 sets of outputs and outputs are only done in states for a Moore machine, 8 total states should be reached. For every MSTL/SSTL combination possible (4), a Moore machine implementation would need 2 states. The first would have outputs to set up and load the timer, and the second would have an output to enable the timer to count.

Our Moore machine controls the timers using 3 outputs: SELC (select counter), LOADC (load counter), ENC (enable counter). The inputs in our state diagram are SSCS and CEXP (counter expired).



State Diagram

From the state diagram above, we were able to complete a state table with which we were able to perform state minimization. Using the equivalence-class partitioning method:

$\Pi_1 = (ABCDEFGHI)$

$\Pi_2 = (A)(BC)(D)(E)(F)(G)(H)(I)$

$\Pi_3 = (A)(BC)(D)(E)(F)(G)(H)(I)$

Originally, state C did not enable the counter (ENC), but by changing that we can minimize the states and since the number of states will go from 9 to 8, we will only need 3 bits instead of 4 to represent the state. This does mean that our counters will need to stop at its maximum and not restart at 0. After the state minimization, we were left with the state table below.

GREEN	001
YELLOW	010
RED	100

SELC	Counter
00	MST (MS yellow light timer)
01	SST (SS yellow light timer)
10	SSC (SS green light counter)

11	MSC (MS green light counter)

Present state	Next state				MSTL			SSTL			SELC		LOAD C	EN C
	00	01	10	11	b2	b1	b0	b2	b1	b0	b1	b0		
A	B	-	B	-	0	0	1	1	0	0	1	1	1	0
B	B	B	B	D	0	0	1	1	0	0	1	1	0	1
D	E	-	E	-	0	1	0	1	0	0	0	0	1	0
E	E	F	E	F	0	1	0	1	0	0	0	0	0	1
F	G	-	G	-	1	0	0	0	0	1	1	0	1	0
G	G	H	G	H	1	0	0	0	0	1	1	0	0	1
H	I	-	I	-	1	0	0	0	1	0	0	1	1	0
I	I	A	I	A	1	0	0	0	1	0	0	1	0	1

State Table

We then performed the state assignment.

Rule 1:
(A,B)x2, (D,E)x2, (F,G)x2, (H,I)x2
Rule 2:
(A,I)x2, (B,D)x2, (E,F)x2, (G,H)x2

Since there were too many outputs and a pattern can already be seen from rule 1 and 2 (which have priority), rule 3 was not performed.
State assignment K-map:

y2\y1y0	00	01	11	10
0	A	B	D	E
1	I	H	G	F

We then completed the transition table.

Present state			Next state									MSTL			SSTL			SELC		LOADC	ENC
			X0			01			11												
y2	y1	y0	Y2	Y1	Y0	Y2	Y1	Y0	Y2	Y1	Y0	b2	b1	b0	b2	b1	b0	b1	b0		
0	0	0	0	0	1	-	-	-	-	-	-	0	0	1	1	0	0	1	1	1	0
0	0	1	0	0	1	0	0	1	0	1	1	0	0	1	1	0	0	1	1	0	1
0	1	1	0	1	0	-	-	-	-	-	-	0	1	0	1	0	0	0	0	1	0
0	1	0	0	1	0	1	1	0	1	1	0	0	1	0	1	0	0	0	0	0	1
1	1	0	1	1	1	-	-	-	-	-	-	1	0	0	0	0	1	1	0	1	0
1	1	1	1	1	1	1	0	1	1	0	1	1	0	0	0	0	1	1	0	0	1
1	0	1	1	0	0	-	-	-	-	-	-	1	0	0	0	1	0	0	1	1	0
1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	1

Transition Table

From the transition table, we derived the following design equations using k-maps:

DFF:
Y2 = (y2cexp') + (y1cexp)
Y1 = (y1cexp') + (y1'y0sscscexp) + (y2'y0'cexp)
Y0 = (y2'y1') + (y2y1)

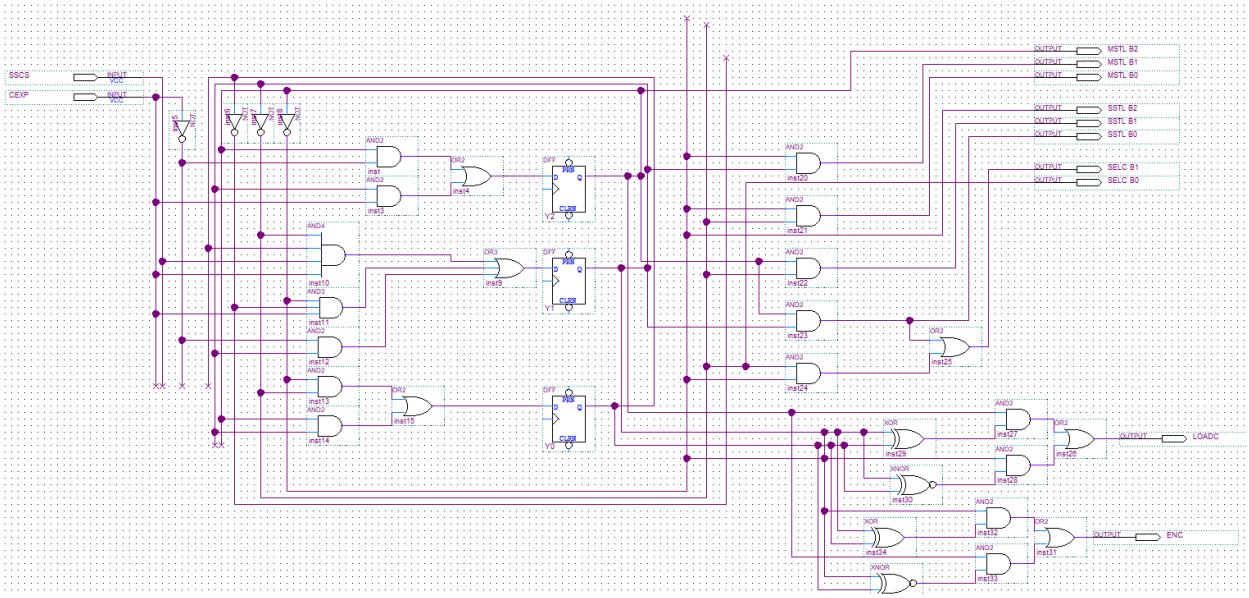
MSTL:
b2 = y2
b1 = (y2'y1)
b0 = (y2'y1')

SSTL:
b2 = y2'
b1 = (y2y1')
b0 = (y2y1)

SELC:
b1 = (y2'y1') + (y2y1)
b0 = y1'

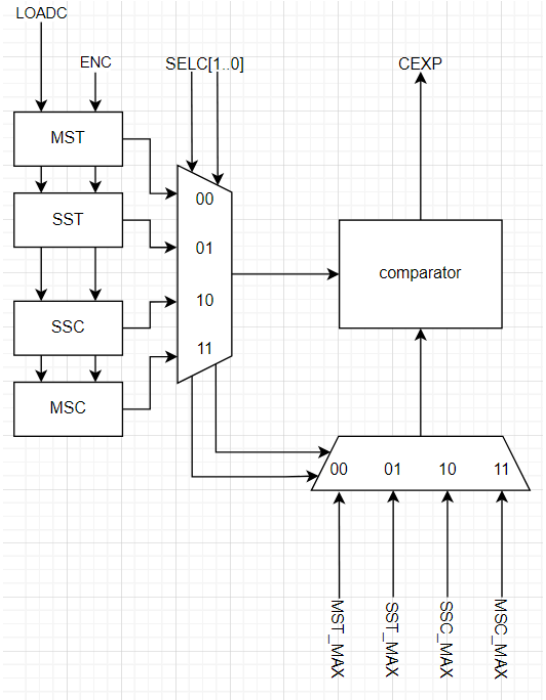
loadc = y2'(y1 xnor y0) + y2(y1 xor y0)
Enc = y2'(y1 xor y0) + y2(y1 xnor y0)

Using these design equations for all flip flops and outputs, the final circuit for the FSM controller could be implemented in VHDL. Shown below is a visual representation of this block.



FSM controller Block diagram (for visual purposes only)

Instead of using the comparator setup shown in figure 9 of the lab manual, we made our own circuit for it to minimize the inputs of the FSM controller.



Comparator and timer selection circuit diagram

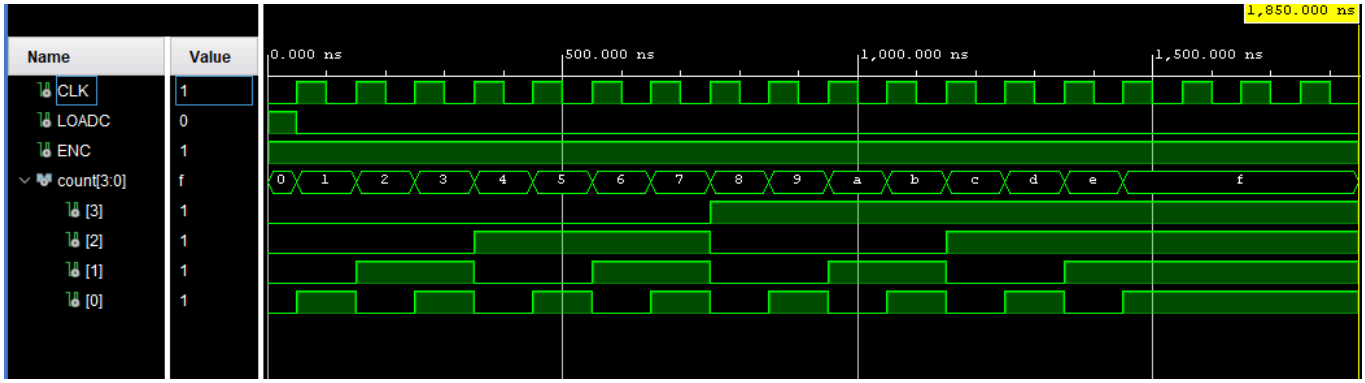
In order to design a 4-bit comparator capable of detecting when a counter was 1. Equal to the max value or 2. Greater than the max value, a simple addition to a normal 4-bit comparator had to be made. Using the given 1-bit comparator connected in series the two

outputs GT and LT could be put together into a nor gate to detect equality between the two 4-bit inputs. Putting this into another or with the GT signal results in an output that reads one whenever the counter value is greater or equal to the input max value. Successful simulation of this block is shown in verification.

Verification:

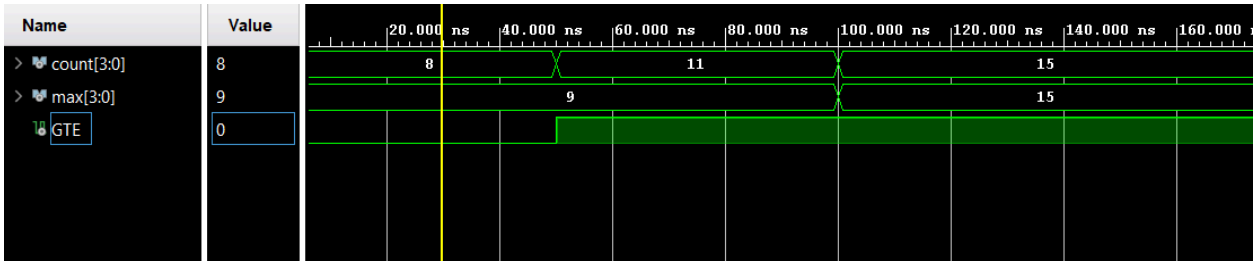
Functional Simulations of major components:

Simulation for 4-bit counter:



We can see that the counter does as we want. It increments every rising edge of the clock and stays at $(F)_{16} = (1111)_2$ until LOADC is asserted.

Simulation for 4-bit comparator:



Testbenches Utilized in the Verification Process:

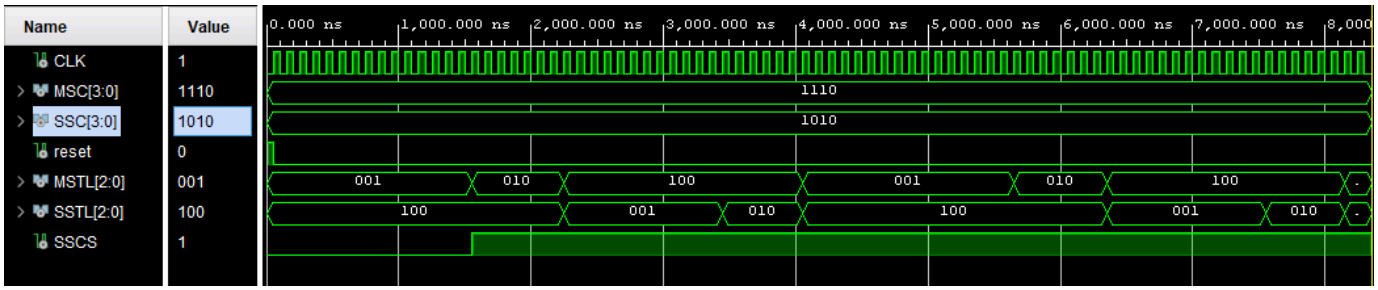
Testbench for 4-bit counter:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity counterTB is
end counterTB;

architecture Behavioral of counterTB is
    signal CLK: std_logic;
    signal LOADC: std_logic := '1';
    signal ENC: std_logic := '1';
    signal count: std_logic_vector(3 downto 0);
begin
    -- clock
    CLK_process: process is
    begin

        CLK <= '0';
        wait for 50 ns;
        CLK <= '1';
        wait for 50 ns;
    end process;

    LOADC <= '0' after 50 ns;

    counter: entity work.counter_4b(struct)
        port map(CLK, LOADC, ENC, count);

end Behavioral;
```

Testbench for 4-bit comparator:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity testbench is
--  Port ( );
end testbench;
architecture Behavioral of testbench is
    component comparator_4b
        port(
            counter, max: in std_logic_vector(3 downto 0);
            GTE : out std_logic
);
end component;
    signal count: std_logic_vector(3 downto 0);
    signal max: std_logic_vector(3 downto 0);
    signal GTE: std_logic;
begin
    test: comparator_4b port map (counter => count, max => max,GTE => GTE );
       process
       begin
       max <= "1001";
       count <= "1000";
       wait for 50 ns;
       max <= "1001";
       count <= "1011";
       wait for 50 ns;
       max <= "1111";
       count<= "1111";
       wait;
end process;
end Behavioral;
```

Top-Level Simulation:



Results are as expected. We can see that the MSTL and SSTL remain the right colors for the right amount of clock cycles. We also notice that the first time the MSTL changed (001→010), its timer had run out but the SSCS was low, indicating no cars on the side street. As soon as SSCS was asserted, MSTL began its cycle through yellow and red lights.

Lab Discussion:

Our first problem was that we needed our counter to stay at its max value once it reached it. We know we needed this because it allowed us to minimize our states. To solve this problem, we followed the FSM steps to design a 4-bit counter and we can see in the simulation for the 4-bit counter that we were able to implement it correctly.

The potential system diagram given to us in the lab manual (figure 9) was not to our liking, as we thought that the SST and MST inputs to the FSM controller were not necessary given that there was already another input (counter expired) that was in charge of checking for counter expiration. Our solution was to use a 4-to-1 MUX with both the timers and counters being the input of the MUX. The circuit diagram for this specific portion of our top-level entity can be found in the problem and solution discussion part of this lab. This turned out well and worked exactly like we planned.

With the implementation of our own 4-bit counter that stays at $(1111)_2$, we realized that our comparator would need to output 1 when the counter value was greater or equal (GTE) to its max value. For this, we realized a 4-bit comparator using 4 1-bit comparators using modular expansion and we connected the output of the comparator (CEXP) to be an or gate of the GT and EQ signals. This meant that CEXP would only be 1 if the counter value was EQ or GT its max value.

Overall after the problems shown above were solved the final implementation of the circuit went smoothly with expected values being displayed for each state and transitions to correct states being successful.

Conclusion and Appendices:

After planning and designing the FSM, implementing circuit blocks, simulating and finally demoing the circuit it can be said that this lab was a success. The final state machine was successfully realized tested and implemented along with the traffic light controller. A complete understanding of synchronous sequential machine design was reached after the successful completion of this lab. All final VHDL source files are commented and available here along with the simulation waveforms: Drive folder of VHDL files for lab 3