Four-way Traffic Control

Isabella Bodmer, Solana Brown, Brikena Dulaj, Dalena Vu
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
Emails: ibodmer@oakland.edu, solanabrown@oakland.edu, brikenadulaj@oakland.edu, dvu@oakland.edu

***Abstract* -- Four-way traffic control circuit replicated with VHDL and breadboard wiring with LED. The main components of the circuit include FSM, counter, and decoder. We had initially run into several errors. However, we were able to complete the circuit, and it had worked as desired with the simulation and through the breadboard circuitry.**

I. INTRODUCTION

For this project, we had decided to replicate a four-way traffic control system, using both VHDL and a breadboard wired to our FPGA boards to copy the LEDs we intend on using in place of the traffic lights. Our project's goal was to successfully emulate a four-way traffic control system using a finite state machine (FSM) and a proper LED setup as its main components and focus. For this project, we had used our prior knowledge of wiring breadboards and LEDs from previous classes. However, some challenges we had encountered were connecting the breadboard to our FPGA boards, altering the constraints file to control the breadboard remotely, and wire our system to be fully functional on its independent clock determined by the counter.

II. METHODOLOGY

A. Overview

Our project's intended way to work is to replicate how a real-life four-way traffic light would function. Our states include each cardinal direction, North, East, South, and West. We set up our circuit to run using a counter, finite state machine (FSM), and a decoder. The counter slows down the FPGAs internal clock to ensure our lights are left on for a reasonable amount of time, not just flashing and undetectable to the eye. The FSM is used to determine which state the circuit is at, which is then fed into a 3 to 12 decoder, which interprets the state and outputs the corresponding LED sequence deciding what LEDs are on and how long.

Our circuit is off until a switch representing enable is flipped. It starts with all light signals being red, while North and South states coincide with each other and West and East states coincide. We then have our North and South states be set to green light signals for ten seconds, all while West and East remain red. After those ten seconds, the North and South states then change the light signal to yellow for five seconds, then all lights turn back to red once those five seconds have passed, functioning similarly to a reset. For that state, all lights will remain red for three seconds. Then the West and East states will then follow the exact timing and process as the North and South states previously, with the green light being on for ten seconds, yellow for five seconds, and then finally turning back to red.

B. FSM

The timing and settings for the lights are based on which state the device is in, making the FSM the central part of our design. To begin designing our FSM, we started by creating an algorithmic state machine (ASM) and a state table. Our ASM shows how our FSM should be behaving and what the outputs for each state should be.

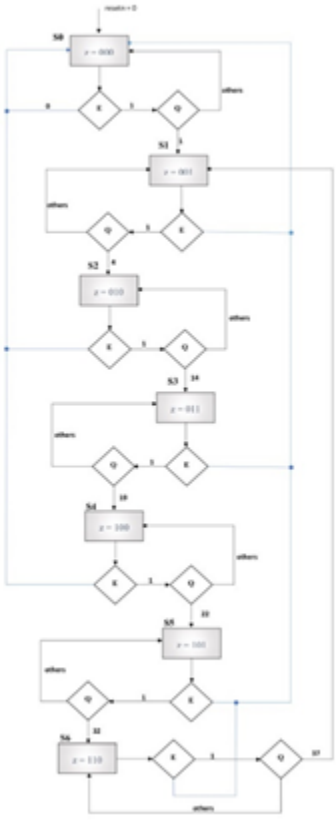

Figure 1. Algorithmic State Machine (ASM)

For our state diagram, we had all agreed on having seven states. It uses a switch representing enable as its input. If the switch is off, all lights are off, and it is on S0. If the enable switch is on, the circuit will progress through the states cycling from S1 to S6, back to S1 after a cycle is completed.

Four-way Traffic Control

Isabella Bodmer, Solana Brown, Brikena Dulaj, Dalena Vu
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
Emails: ibodmer@oakland.edu, solanabrown@oakland.edu, brikenadulaj@oakland.edu, dvu@oakland.edu

		N	E	S	W
	S0				
3s	S1	R	R	R	R
10s	S2	G	R	G	R
5s	S3	Y	R	Y	R
3s	S4	R	R	R	R
10s	S5	R	G	R	G
5s	S6	R	Y	R	Y.

Figure 2. State table

Our state table goes further in-depth with each state's duration and shows one complete cycle of our circuit in "chronological" order. We then created the FSM code to run on our designated process between each state from this state table. The cycle illustrates the output in each case.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity FSM is
  Port (clock,resetn, E: IN std_logic;

        z: OUT std_logic_vector(2 DOWNTO 0));

end FSM;

architecture bhv of FSM is
component Counter
Port (clock,resetn, E: IN std_logic;
              Q: out integer range -1800000000 TO 1800000000);

end component;

type state_type is (S0, S1,S2,S3, S4, S5, S6);
signal state: state_type;
signal Q: integer range -1800000000 TO 1800000000;
begin
f: Counter port map (clock => clock, resetn => resetn, Q => Q, E => E);
Transitions: process (resetn,clock, E, Q)
    begin
        if resetn = '0' then
            state <= S0; --initial state
elsif (clock'event and clock='1') then
if E = '1' then
case state is
when S0 =>
  if Q = -1800000000 then state <= S1; else state <=S0; end if;
when S1 =>
 if Q = -1500000000 then state <= S2; else state <=S1; end if;
when S2 =>
   if Q = -500000000 then state <= S3; else state <=S2; end if;
when S3 =>
  if Q = 0 then state <= S4; else state <=S3; end if;
when S4 =>
 if Q = 300000000 then state <= S5; else state <=S4; end if;
when S5 =>
  if Q = 1300000000 then state <= S6; else state <=S5; end if;
when S6 =>
 if Q = 1800000000 then  state <= S1; else state <=S6; end if;
```

Figure 3. Part one of the FSM code

```
) end case;
   else
   state <= S0;
) end if;
) end if;

) end process;


)   Outputs: process (E, Q, state)
     begin
)    case state is
      when S0 => z <= "000";
      when S1 => z <= "001";
      when S2 => z <= "010";
      when S3 => z <= "011";
      when S4 => z <= "100";
      when S5 => z <= "101";
      when S6 => z <= "110";
)              end case;
)  end process;
) end bhv;
```

Figure 4. Part two of the FSM code

C. *Counter*

The counter intends to slow down the internal clock signal from nanoseconds to seconds. The clock signal will be fed into the counter first, and the counter's output is a slower clock signal that will be fed into the FSM, and then from that point used as that standard measurement of time. We decided to use a six-bit BCD counter [3] initially but then increased the number of bits according to what the output would be in seconds. Since the default clock is 100MHz, this counter's default time would be nanoseconds that were too fast for us to see a difference. The output Q from the counter then goes into the FSM to signal the current state's duration. While Q could have been left in BCD, we agreed that converting it to an integer using the formula (2^n)-1 would be easier to implement.

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

entity Counter is
  Port (clock,resetn, E: IN std_logic;
          Q: out integer range -1800000000 TO 1800000000);


end Counter;

architecture bhv of Counter is
signal Qt: integer range -1800000000 TO 1800000000;
--signal Qt: std_logic_vector(3 downto 0);
begin
process (resetn,clock)
begin
if resetn = '0' then
Qt <= -1800000000;
elsif (clock'event and clock='1') then
if E = '1' then
if Qt =  1800000000 then
Qt <= -1800000000;
else
Qt <= Qt + 1;
end if;
end if;
end if;
end process;
Q <= Qt;

end bhv;
```

Figure 5. Counter code

Four-way Traffic Control

Isabella Bodmer, Solana Brown, Brikena Dulaj, Dalena Vu
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
Emails: ibodmer@oakland.edu, solanabrown@oakland.edu, brikenadulaj@oakland.edu, dvu@oakland.edu

D. Decoder

We decided to use five addresses representing each state embedded into the decoder [3]. Each address will be twelve bits, while each four-bit segment was assigned a LED color, the first four bits being green, then yellow, and then red. Each bit was designated as North, South, East, and West from most to least significant bit. This pattern was repeated for each four-bit segment. When the FSM would output the value corresponding to the state it is in, that value would then feed into the decoder, grab the respective address, and output it so that the 12-bit value can be sent to the LEDs.

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


entity My_Decoder is
port ( clock,resetn, E: in std_logic;
--          z: IN std_logic_vector (2 downto 0);
          R : out std_logic_vector (11 downto 0));
end My_Decoder;

architecture bhv of My_Decoder is
component FSM
 Port (clock,resetn, E: IN std_logic;
       z: OUT std_logic_vector(2 DOWNTO 0));
       end component;
signal z: std_logic_vector (2 downto 0);
signal Q: integer range -1800000000 TO 1800000000;
type state_type is ( S0, S1,S2,S3, S4, S5, S6);
signal state: state_type;
--signal  E: std_logic;
-- |Green| |Yellow| |Red|
begin
u: FSM port map (clock => clock, resetn =>resetn, z => z, E => E);
with z select
                    R <= "001001001001" when "001",   --red red red red
                         "100001100001" when "010",  -- green red green red
                         "010001010001" when "011",  -- yellow red yellow red
                         "001001001001" when "100",   -- red red red red
                         "001100001100" when "101",   -- red green red green
                         "001010001010" when "110",   --red yellow red yellow
                         "000000000000"  when others;


end bhv;
```

Figure 6. Decoder code

E. Top File

For the top file, we implement the components of the decoder, FSM, and counter. We port mapped all of the included signals, inputs, and outputs for each respective component.

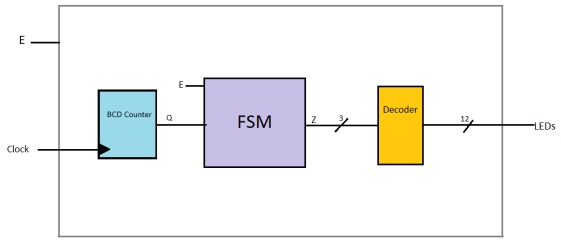

Figure 7. Final Block Diagram

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

entity top is
 Port (clock, resetn, E: in std_logic;
       R : out std_logic_vector (11 downto 0));
end top;

architecture bhv of top is

component My_Decoder is
port ( clock,resetn, E: in std_logic;
--
            R : out std_logic_vector (11 downto 0));
          end component;

signal z: std_logic_vector (2 downto 0);
--signal E: std_logic;
signal Q: integer range -1800000000 TO 1800000000;
type state_type is (S0, S1,S2,S3, S4, S5, S6);
signal state: state_type;

begin

  g: My_Decoder port map (clock => clock, resetn => resetn, R => R, E => E );


end bhv;
```

Figure 8. Top file code

III. EXPERIMENTAL SETUP

Once the code for each module was completed, the next step was to determine how to get the FPGA output values to interface with our LEDs. We chose to use an external breadboard with 12 LEDs to represent the traffic lights, requiring additional research. After looking at the manual for the Nexys A7, we decided to use the JA and JB pins to interface to the breadboard [1]. The top half of the board was grounded through the JA ground port, and the lights on the top half were then wired to JA pins 1-4 and 7-8. Meanwhile, the bottom half of the board was grounded through the JB ground port, and the lights on the bottom half were wired to JB pins 1-4 and 7-8 [1]. The constraint file was then altered to reflect those connections.

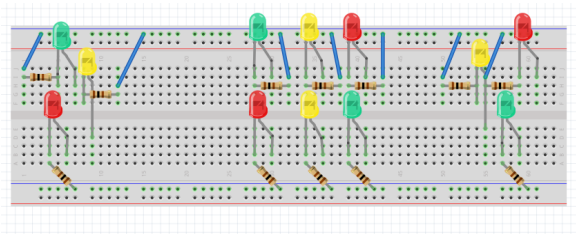

Figure 9. Simulated breadboard setup

The testbench is used to simulate that the codes are functioning as expected. The clock and reset signals are generated in the process statement for this combination circuit. We process the reset signal as one initially, then set it to 0 for the next clock cycle and the following input signals; therefore, the signal is active high.

Four-way Traffic Control

Isabella Bodmer, Solana Brown, Brikena Dulaj, Dalena Vu
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
Emails: ibodmer@oakland.edu, solanabrown@oakland.edu, brikenadulaj@oakland.edu, dvu@oakland.edu

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_arith.ALL;

entity top_tb is
--  Port ( );
end top_tb;

architecture bhv of top_tb is
component top is
 Port (clock, resetn, E: in std_logic;
       R : out std_logic_vector (11 downto 0));
       end component;
-- component Counter is
--  Port (clock,resetn, E: IN std_logic;
--           Q: out integer range 0 TO 63);


--end component;
component FSM is
  Port (clock,resetn, E: IN std_logic;

        z: OUT std_logic_vector(2 DOWNTO 0));
end component;
component Counter
Port (clock,resetn, E: IN std_logic;
             Q: out integer range -1800000000 TO 1800000000);

end component;
  --INPUTS
 signal clock : std_logic := '0';
   signal resetn : std_logic := '0';
   signal E : std_logic := '0';
   --OUTPUTS
  signal R: std_logic_vector (11 downto 0);
  signal z: std_logic_vector(2 DOWNTO 0);
--  type state_type is (S0, S1,S2,S3, S4, S5, S6);
--  signal state: state_type;
```

Figure 10. Part one of the testbench code

```
 signal Q: integer range  -1800000000 TO 1800000000;
    constant T: time:= 10 ns;
  constant DUTY_CYCLE: real:= 0.5;
    constant OFFSET: time:= 1 sec;

 begin
  uut: top port map (
         clock => clock,
          resetn => resetn,
          E => E,
          R => R
        );
   uuv: FSM port map (clock => clock, resetn =>resetn, z => z, E => E);
   f: Counter port map (clock => clock, resetn => resetn, Q => Q, E => E);
--  uuv: counter port map (
--      clock => clock,
--          resetn => resetn,
--          E => E,
--          Q => Q);

    -- Clock process definitions
    clock_process :process
    begin
         wait for T/2;
         clock_loop: loop
             clock <= '0'; wait for (T - (T*DUTY_CYCLE));
             clock <= '1'; wait for (T*DUTY_CYCLE);
         end loop clock_loop;
    end process;

    -- Stimulus process
    stim_proc: process
    begin
       -- hold reset state for 100 ns.
       wait for 10 ns;

       Resetn <= '0';  wait for 100ns;
       Resetn <= '1'; E <= '1'; wait for 37 sec;
       wait;
    end process;

 END;
```

Figure 11. Part two of the testbench code

IV. RESULTS

Initially, our results were shown as expected in the simulation of Vivado. Still, when we generated the bitstream for our circuit and programmed the board, the circuit did not behave as expected. Initially, we could not figure out why the simulation was correct, but the breadboard would not output the right states or time for the LEDs. However, we later found that once we allotted more time for the simulation to run in the console, the circuit worked as expected. Each LED was lit for the exact time as initially intended, and the circuit cycled as it should have. A link for a demo of our circuit is provided with the link below [2].

V. CONCLUSIONS

We believe that many of the takeaways we had from this project are to be flexible and open to change and adapt to our situation. Overall, this project was incredibly difficult for all of us. We often kept running into problems early on in the project and kept changing our plans and scraping our original ideas. It was incredibly frustrating, primarily since we had worked so hard on the project, and it often felt we were at our wit's end. However, after some final tweaking with the time and console, we could correctly program our circuit, and the LEDs worked perfectly. We had learned a lot through this project, specifically about FSMs and using different clock signals for a circuit that runs independently of the program's time. Our prior knowledge of a four-way traffic light that we had gone over in class proved very helpful, even though we used a completely different methodology.

References

[1] Brown, Arthur. "Nexys A7 Reference Manual." *Nexys A7 Reference Manual - Digilent Reference*, reference.digilentinc.com/reference/programmable-logic/nexys-a7/reference-manual.

[2] https://linksharing.samsungcloud.com/qELnwCAHSxzO

[3] Llamocca, Daniel. *VHDL Coding for FPGAs*, Oakland SECS, www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html.