4-Way Traffic Intersection Control

Christopher Thweatt, Joseph Asteefan, John Drabik, Madison Cornett

Introduction

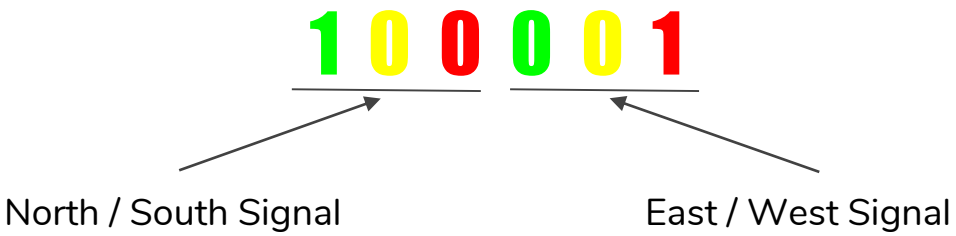
Photo By Christopher Evans/MediaNews Group/Boston Herald

- Proof of Concept: To reduce the footprint and complexity of hardware necessary to prototype and install flexible traffic control systems.

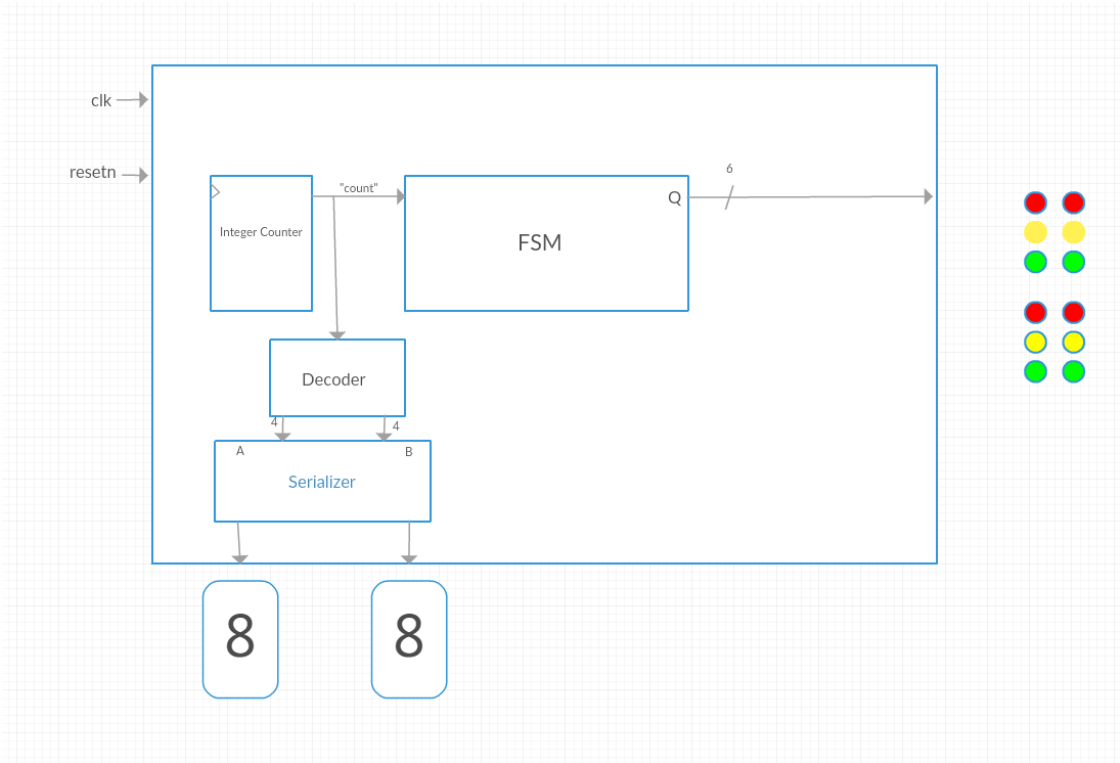
- Traffic control simulation built in VHDL code:
  - Finite State Machine
  - Counter
  - Decoder
  - Serializer
  - Segment and LED Display

Methodology

- Artix-A7 FPGA, bread board, and colored LEDs
- Integer counter
- State machine:
  - Each state represents a different light sequence for each of the signals, and changes dependent on the accumulated count value.
  - Six bit output. Example: 100 (green) for the north/south signals and 001 (red) for the east/west signals.



Top Level Design



Counter

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity counter is
5      port ( clock, resetn: in std_logic;
6             Q: out integer range -1500000000 to 1500000000);
7      end counter;
8
9  architecture bhv of counter is
10     signal Qt: integer range -1500000000 to 1500000000;
11
12 begin
13
14     process (resetn,clock)
15     begin
16         if resetn = '0' then
17             Qt <= -1500000000;
18         elsif (clock'event and clock='1') then
19             if Qt = 1500000000 then
20             Qt <= -1500000000;
21         else
22             Qt <= Qt + 1;
23            end if;
24         end if;
25     end process;
26     Q <= Qt;
27 end bhv;
```

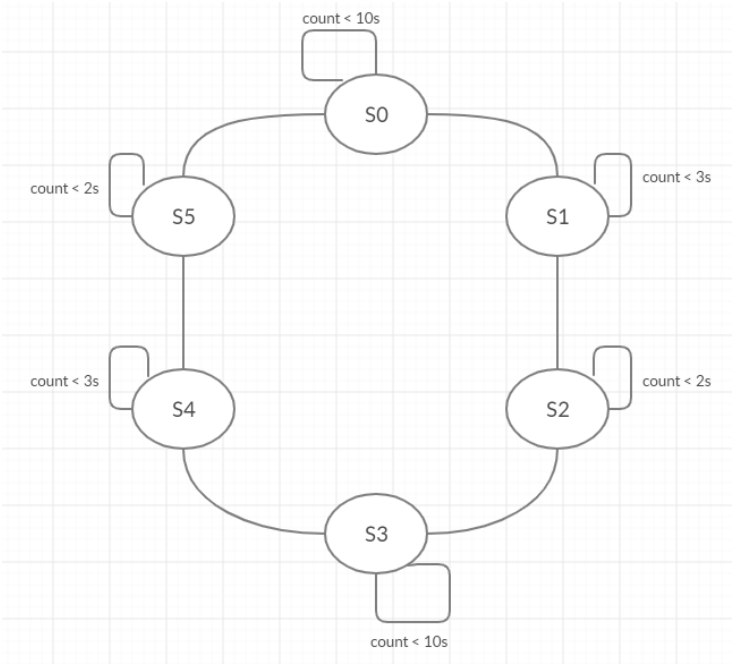
- Initially used 0 to 30 counter
- Integer instead of BCD

State Machine

```
architecture Behavioral of fsm is
    type state_type is (s0, s1, s2, s3, s4, s5);
    signal state: state_type;
begin
    transitions: process (clk, rstn)
    begin
        if rstn = '0' then state <= s0;
        elsif (clk'event and clk = '1') then
            case state is
                when s0 =>
                    if Q = -500000000 then state <= s1;
                    else state <= s0;
                    end if;
                when s1 =>
                    if Q = -200000000 then state <= s2;
                    else state <= s1;
                    end if;
                when s2 =>
                    if Q = 0 then state <= s3;
                    else state <= s2;
                    end if;
                when s3 =>
                    if Q = 1000000000 then state <= s4;
                    else state <= s3;
                    end if;
                when s4 =>
                    if Q = 1300000000 then state <= s5;
                    else state <= s4;
                    end if;
                when s5 =>
                    if Q = 1500000000 then state <= s0;
                    else state <= s5;
                    end if;
            end case;
        end if;
    end process;
```

```
    --100 is green, 010 is yellow, 001 is red
    outputs: process(state)
    begin
        case state is
            when s0 => northeast <= "100001";
                       southwest <= "100001";
            when s1 => northeast <= "010001";
                       southwest <= "010001";
            when s2 => northeast <= "001001";
                       southwest <= "001001";
            when s3 => northeast <= "001100";
                       southwest <= "001100";
            when s4 => northeast <= "001010";
                       southwest <= "001010";
            when s5 => northeast <= "001001";
                       southwest <= "001001";
        end case;
    end process;
end Behavioral;
```

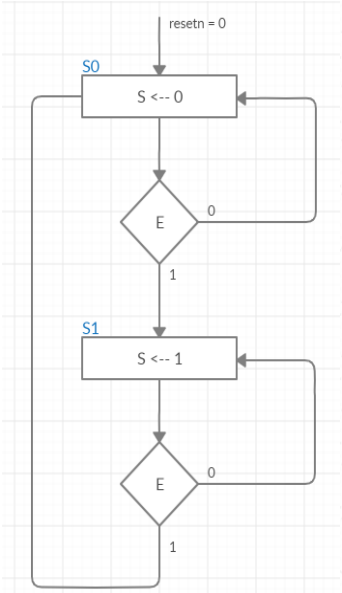
North/South			East/West			State
G	Y	R	G	Y	R	
1	0	0	0	0	1	S0
0	1	0	0	0	1	S1
0	0	1	0	0	1	S2
0	0	1	1	0	0	S3
0	0	1	0	1	0	S4
0	0	1	0	0	1	S5

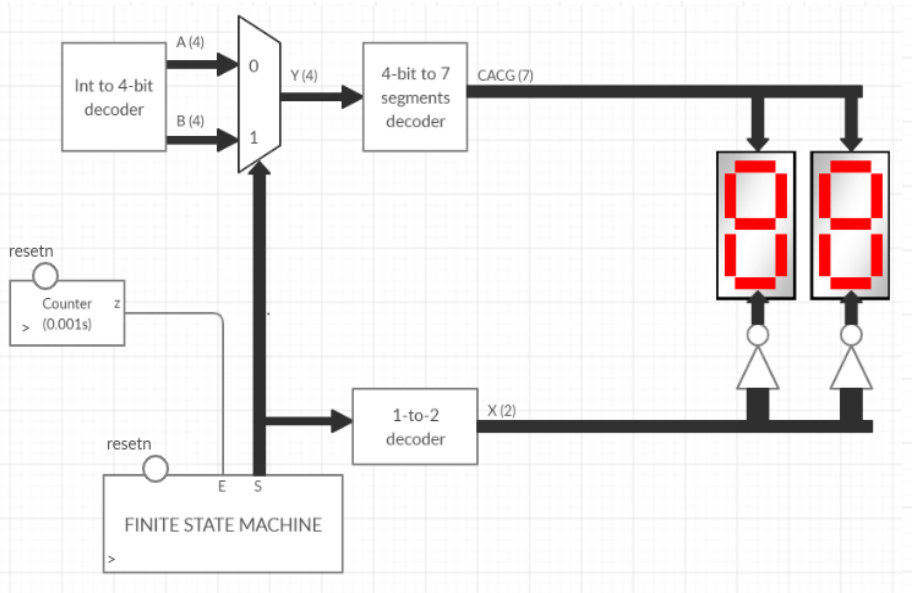


Serializer and Decoder

```
architecture Behavioral of dec is

begin
    process (Q)
    begin
        if Q < -1400000001 then A <= "0000"; B <= "0000";
        elsif Q < -1300000001 then A <= "0001"; B <= "0000";
        elsif Q < -1200000001 then A <= "0010"; B <= "0000";
        elsif Q < -1100000001 then A <= "0011"; B <= "0000";
        elsif Q < -1000000001 then A <= "0100"; B <= "0000";
        elsif Q < -900000001 then A <= "0101"; B <= "0000";
        elsif Q < -800000001 then A <= "0110"; B <= "0000";
        elsif Q < -700000001 then A <= "0111"; B <= "0000";
        elsif Q < -600000001 then A <= "1000"; B <= "0000";
        elsif Q < -500000001 then A <= "1001"; B <= "0000";
        elsif Q < -400000001 then A <= "0000"; B <= "0001";
        elsif Q < -300000001 then A <= "0001"; B <= "0001";
        elsif Q < -200000001 then A <= "0010"; B <= "0001";
        elsif Q < -100000001 then A <= "0011"; B <= "0001";
        elsif Q < -000000001 then A <= "0100"; B <= "0001";
        elsif Q <  099999999 then A <= "0101"; B <= "0001";
        elsif Q <  199999999 then A <= "0110"; B <= "0001";
        elsif Q <  299999999 then A <= "0111"; B <= "0001";
        elsif Q <  399999999 then A <= "1000"; B <= "0001";
        elsif Q <  499999999 then A <= "1001"; B <= "0001";
        elsif Q <  599999999 then A <= "0000"; B <= "0010";
        elsif Q <  699999999 then A <= "0001"; B <= "0010";
        elsif Q <  799999999 then A <= "0010"; B <= "0010";
        elsif Q <  899999999 then A <= "0011"; B <= "0010";
        elsif Q <  999999999 then A <= "0100"; B <= "0010";
        elsif Q < 1099999999 then A <= "0101"; B <= "0010";
        elsif Q < 1199999999 then A <= "0110"; B <= "0010";
        elsif Q < 1299999999 then A <= "0111"; B <= "0010";
        elsif Q < 1399999999 then A <= "1000"; B <= "0010";
        elsif Q < 1499999999 then A <= "1001"; B <= "0010";
        elsif Q < 1500000001 then A <= "0000"; B <= "0011";
        end if;
    end process;
end Behavioral;
```

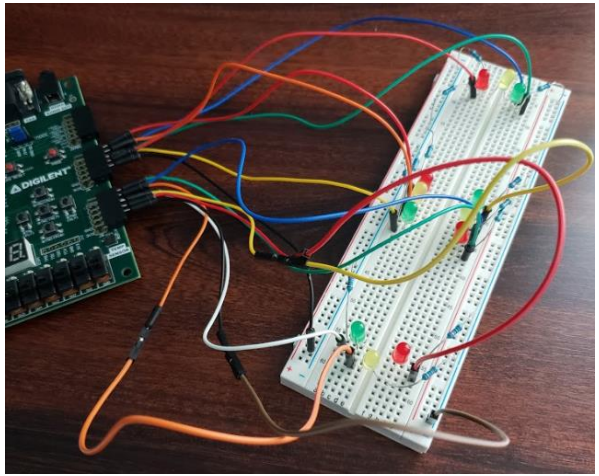




XDC file / Experimental Setup

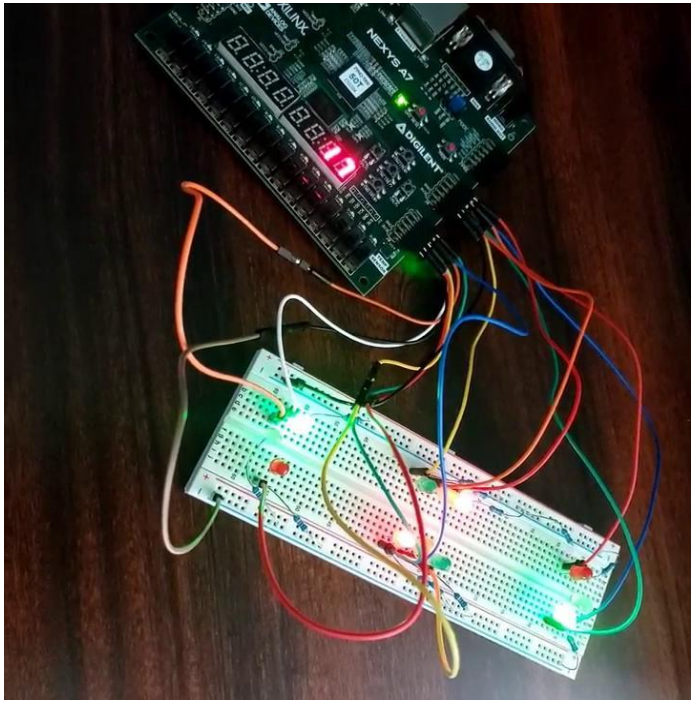
- Clock signal
- 7 segment display
  - First 7 ports
  - AN 0-7
- Buttons
  - CPU reset
- Pmod headers JA and JB
  - JA first 6 ports
  - JB first 6 ports

- Breadboard
- Wires
- Resistors
- External LEDs



Demo

https://drive.google.com/file/d/1FN8xiAmwgKfuOerfLMHYy7eFkHJUG4fz/view?usp=sharing



Conclusion

- Knowledge of VHDL programming and the use of a finite state machine
- Decoders, counters, and serializers
    - Being able to apply prior knowledge and modify existing code to adapt to the situation.
- Learned how to use the different ports to connect to a breadboard
- Used previous knowledge of LEDs and the function of a breadboard to implement the function.
- Strengthened our abilities to solve problems even in the most unideal situations.