



SCROLLING SEVEN SEGMENT BANNER DISPLAY

CONTRIBUTORS:

ANDREW ROBINSON

YAN WANG

JACOB GUBOW

MOTIVATIONS

- To display a desired message to a reader in a way that is compact yet still easy to read.
- To utilize current FPGA technology to effectively convey a message in a hand free low maintenance way.
- To further our knowledge of the FPGA's and Finite State Machine design (FSM) through the creation and development of projects.

COMPONENTS

- Clk2hz
- Mux 8_1
- My_4bitcounter
- My_4bitcounter_e
- My_genpulse_sclr
- My_rege
- Mydec3to8
- Seg_7

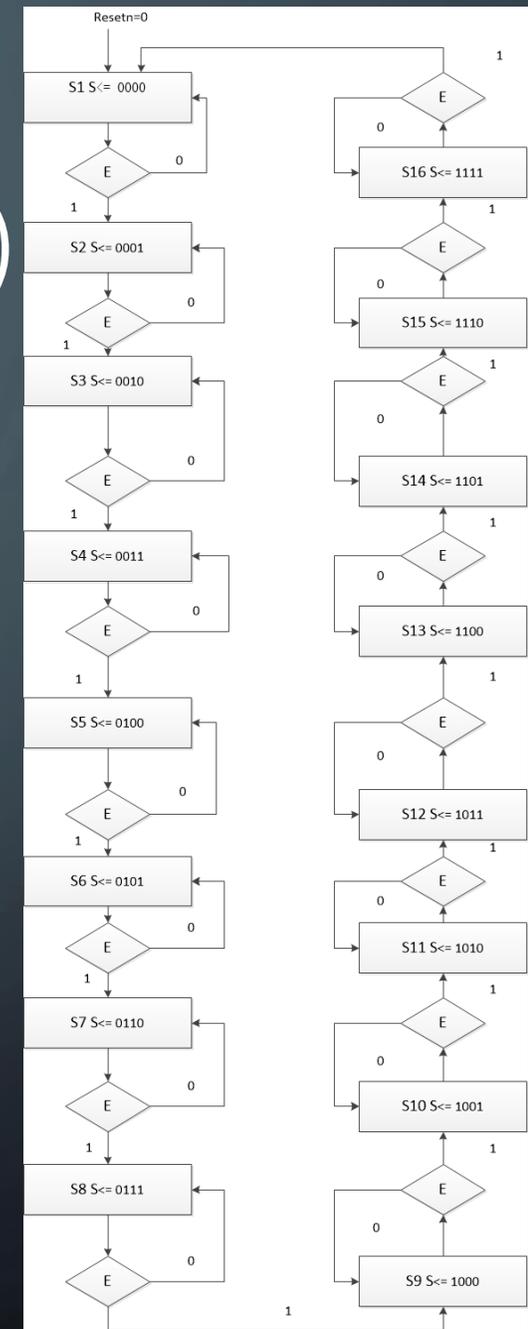
FINITE STATE MACHINE (BCD COUNTER)

```

entity my_4bitcounter_e is
    port (clock, resetn,e: in std_logic;
          q: out std_logic_vector (3 downto 0));
end my_4bitcounter_e;

architecture behaviour of my_4bitcounter_e is
    type state is (S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14,S15,S16);
    signal y: state;
begin
    Transitions: process (resetn, clock, E)
    begin
        if resetn = '0' then -- asynchronous signal
            y <= S1; -- if resetn asserted, go to initial state: S1
        elsif (clock'event and clock = '1') then
            if E = '1' then
                case y is
                    when S1 => y <= S2;
                    when S2 => y <= S3;
                    when S3 => y <= S4;
                    when S4 => y <= S5;
                    when S5 => y <= S6;
                    when S6 => y <= S7;
                    when S7 => y <= S8;
                    when S8 => y <= S9;
                    when S9 => y <= S10;
                    when S10 => y <= S11;
                    when S11 => y <= S12;
                    when S12 => y <= S13;
                    when S13 => y <= S14;
                    when S14 => y <= S15;
                    when S15 => y <= S16;
                    when S16 => y <= S1;
                end case;
            end if;
        end if;
    end process;

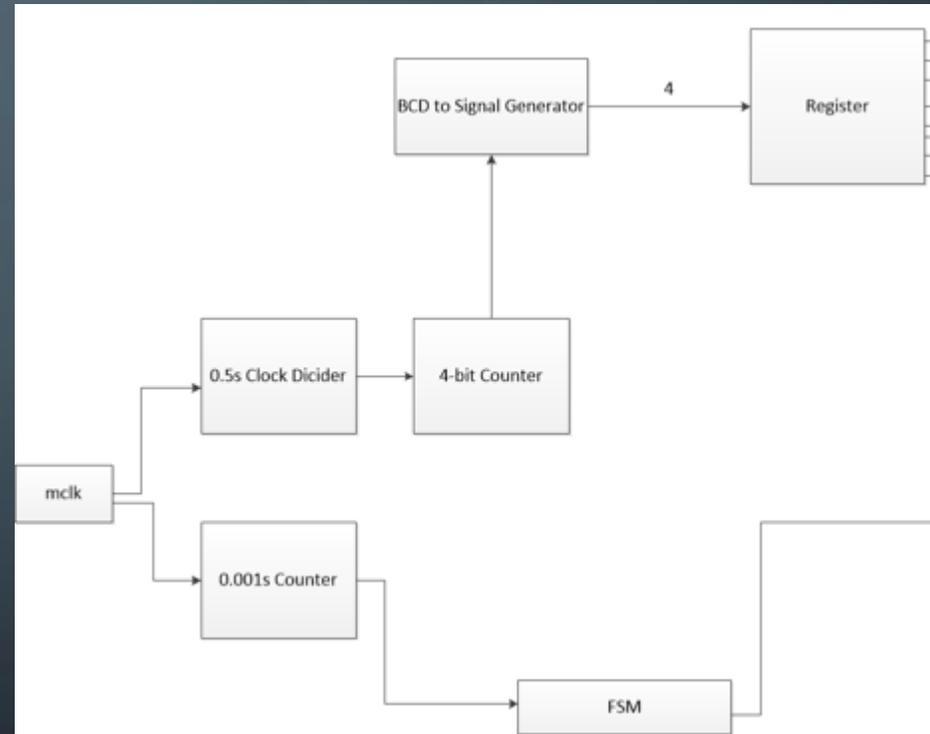
    Outputs: process (y)
    begin
        case y is
            when S1 => q <= "0000";
            when S2 => q <= "0001";
            when S3 => q <= "0010";
            when S4 => q <= "0011";
            when S5 => q <= "0100";
            when S6 => q <= "0101";
            when S7 => q <= "0110";
            when S8 => q <= "0111";
            when S9 => q <= "1000";
            when S10 => q <= "1001";
            when S11 => q <= "1010";
            when S12 => q <= "1011";
            when S13 => q <= "1100";
            when S14 => q <= "1101";
            when S15 => q <= "1110";
            when S16 => q <= "1111";
        end case;
    end process;
end behaviour;
    
```



REGISTERS AND MUX

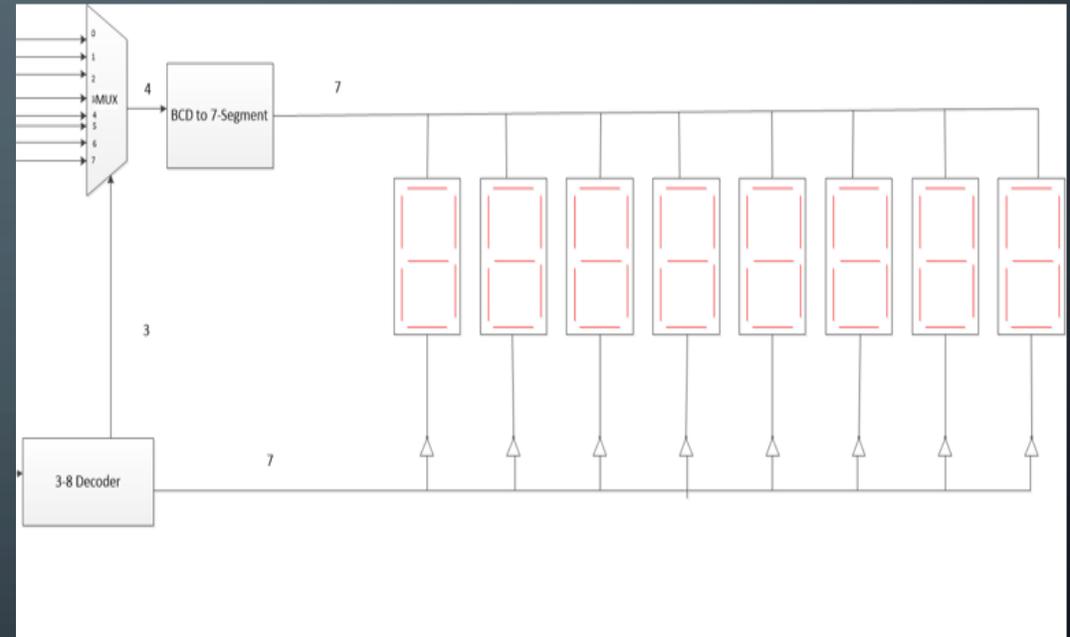
```
signal clkd: std_logic;
signal countout: std_logic_vector (3 downto 0);
signal reg0, reg1, reg2, reg3, reg4, reg5, reg6, reg7: std_logic_vector (3 downto 0);
signal dec: std_logic_vector (7 downto 0);
signal muxout: std_logic_vector ( 3 downto 0);
signal negdec: std_logic_vector (7 downto 0);
signal bittodec: std_logic_vector (3 downto 0);
signal muxsel: std_logic_vector (2 downto 0);
signal e_onems: std_logic;

begin
negdec <= not(dec);
an <= negdec;
--porting from topfile to register
c10: clkHz port map (clk_in => mclk, reset => resetn, clk_out => clkd);
fc: my_genpulse_sclr generic map (COUNT => 10**5)
port map (clock => mclk, resetn => resetn, E => '1', sclr => '0', z => e_onems);
--c0: my_4bitcounter port map (clock => mclk, resetn => resetn, q => bittodec);
c0: my_4bitcounter_E port map (clock => mclk, resetn => resetn, e => e_onems , q => bittodec);
c1: my_4bitcounter port map (clock => clkd, resetn => resetn, q => countout);
--d0: mydec3to8 port map (w => bittodec(2 downto 0), En => mclk, y => dec);
d0: mydec3to8 port map (w => bittodec(2 downto 0), En => '0', y => dec);
r0: my_rege port map (clock => clkd, resetn => resetn, En => SW,D => countout, Q => reg0);
r1: my_rege port map (clock => clkd, resetn => resetn, En => SW,D => reg0, Q => reg1);
r2: my_rege port map (clock => clkd, resetn => resetn, En => SW,D => reg1, Q => reg2);
r3: my_rege port map (clock => clkd, resetn => resetn, En => SW,D => reg2, Q => reg3);
r4: my_rege port map (clock => clkd, resetn => resetn, En => SW,D => reg3, Q => reg4);
r5: my_rege port map (clock => clkd, resetn => resetn, En => SW,D => reg4, Q => reg5);
r6: my_rege port map (clock => clkd, resetn => resetn, En => SW,D => reg5, Q => reg6);
r7: my_rege port map (clock => clkd, resetn => resetn, En => SW,D => reg6, Q => reg7);
m0: mux8_1 port map (s1 => reg0, s2 => reg1, s3 => reg2, s4 => reg3, s5 => reg4, s6 => reg5, s7 => reg6, s8 => reg7, sel => bittodec(2 downto 0), En => SW, y => muxout);
s0: seg_7 port map (bcd => muxout, sevenseg => seg);
end Behavioral;
```

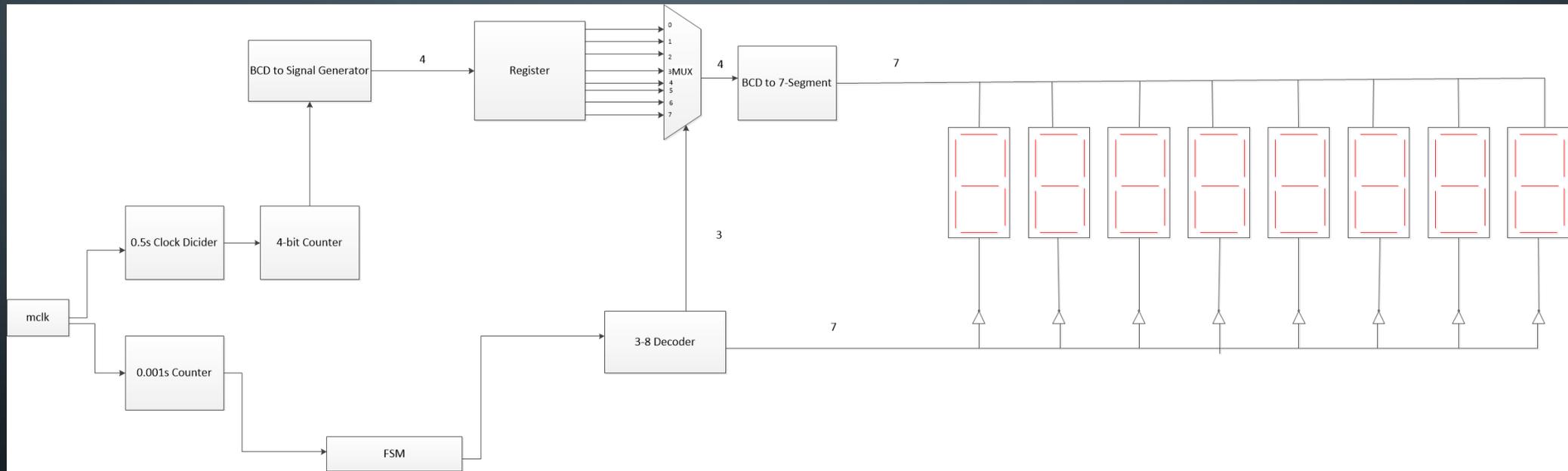


TOP LEVEL OUTPUT

```
signal clkd: std_logic;
signal countout: std_logic_vector (3 downto 0);
signal reg0, reg1, reg2, reg3, reg4, reg5, reg6, reg7: std_logic_vector (3 downto 0);
signal dec: std_logic_vector (7 downto 0);
signal muxout: std_logic_vector ( 3 downto 0);
signal negdec: std_logic_vector (7 downto 0);
signal bittodec: std_logic_vector (3 downto 0);
signal muxsel: std_logic_vector (2 downto 0);
signal e_onems: std_logic;
begin
negdec <= not(dec);
an <= negdec;
--porting from topfile to register
c10: clkHz port map (clk_in => mclk, reset => resetn, clk_out => clkd);
fc: my_genpulse_sclr generic map (COUNT => 10**5)
port map (clock => mclk, resetn => resetn, E => '1', sclr => '0', z => e_onems);
--c0: my_4bitcounter port map (clock => mclk, resetn => resetn, q => bittodec);
c0: my_4bitcounter_E port map (clock => mclk, resetn => resetn, e => e_onems, q => bittodec);
c1: my_4bitcounter port map (clock => clkd, resetn => resetn, q => countout);
--d0: mydec3to8 port map (w => bittodec(2 downto 0), En => mclk, y => dec);
d0: mydec3to8 port map (w => bittodec(2 downto 0), En => '0', y => dec);
r0: my_rege port map (clock => clkd, resetn => resetn, En => SW,D => countout, Q => reg0);
r1: my_rege port map (clock => clkd, resetn => resetn, En => SW,D => reg0, Q => reg1);
r2: my_rege port map (clock => clkd, resetn => resetn, En => SW,D => reg1, Q => reg2);
r3: my_rege port map (clock => clkd, resetn => resetn, En => SW,D => reg2, Q => reg3);
r4: my_rege port map (clock => clkd, resetn => resetn, En => SW,D => reg3, Q => reg4);
r5: my_rege port map (clock => clkd, resetn => resetn, En => SW,D => reg4, Q => reg5);
r6: my_rege port map (clock => clkd, resetn => resetn, En => SW,D => reg5, Q => reg6);
r7: my_rege port map (clock => clkd, resetn => resetn, En => SW,D => reg6, Q => reg7);
m0: mux3_1 port map (s1 => reg0, s2 => reg1, s3 => reg2, s4 => reg3, s5 => reg4, s6 => reg5, s7 => reg6, s8 => reg7, sel => bittodec(2 downto 0), En => SW, y => muxout);
s0: seg_7 port map (bcd => muxout, sevenseg => seg);
end Behavioral;
```

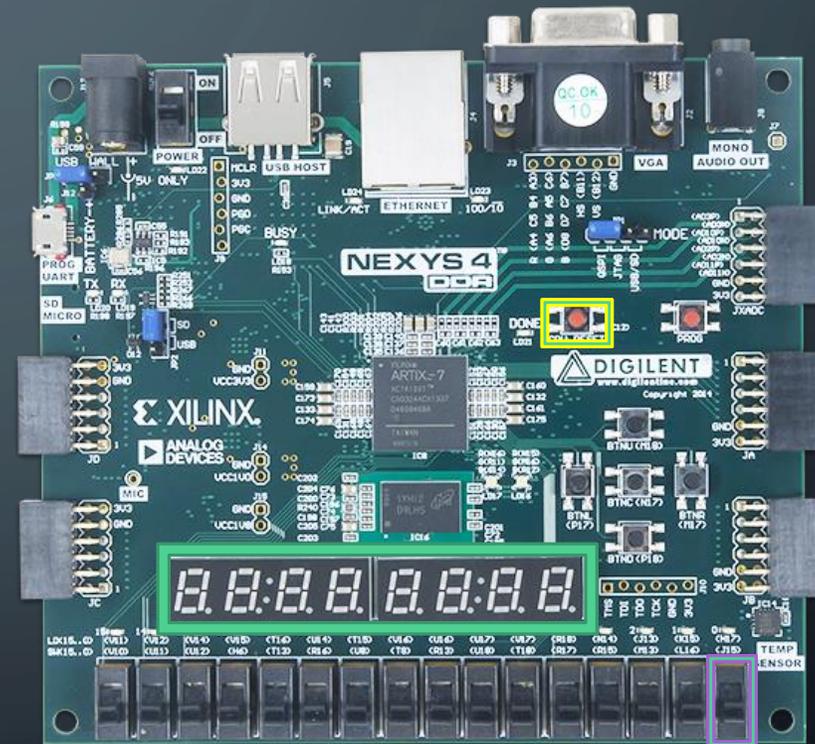


TOP LEVEL DIAGRAM



LAYOUT

- Reset Button
- Enable
- Output/display



SIMULATION - SHIFT REGISTER FUNCTIONALITY

| | | | | | | | | | | | |
|-----------|---|---|---|---|---|---|---|---|---|---|---|
| > s1[3:0] | 8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| > s2[3:0] | 7 | 0 | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| > s3[3:0] | 6 | | 0 | | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| > s4[3:0] | 5 | | | 0 | | 1 | 2 | 3 | 4 | 5 | 6 |
| > s5[3:0] | 4 | | | | 0 | | 1 | 2 | 3 | 4 | 5 |
| > s6[3:0] | 3 | | | | | 0 | | 1 | 2 | 3 | 4 |
| > s7[3:0] | 2 | | | | | | 0 | | 1 | 2 | 3 |
| > s8[3:0] | 1 | | | | | | | 0 | | 1 | 2 |
| > y[3:0] | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

CHALLENGES

- Shifting the first letter from each seven segment display
- Dividing the clock for the different parts of the circuit
- Getting down the correct timing for the different parts of the circuit
- Finding an appropriate message to display because of 7 segment character restrictions.



CONCLUSIONS

- Successfully implement a FSM as a counter
- Successfully scrolled the words “GO GRIZZLIES” across the seven segments
- Improvements:
 - Alter the circuit so that it display more than one message.
 - Use a LCD Display and code into Ascii to give a wider range of characters you can use for the message.
 - Integration with another program such as the display being related to a winning condition.

REFERENCES

- Llamocca, D. *VHDL Coding For FPGAs*. Retrieved from <http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html>



QUESTIONS?

THANK YOU FOR WATCHING AND LISTENING.