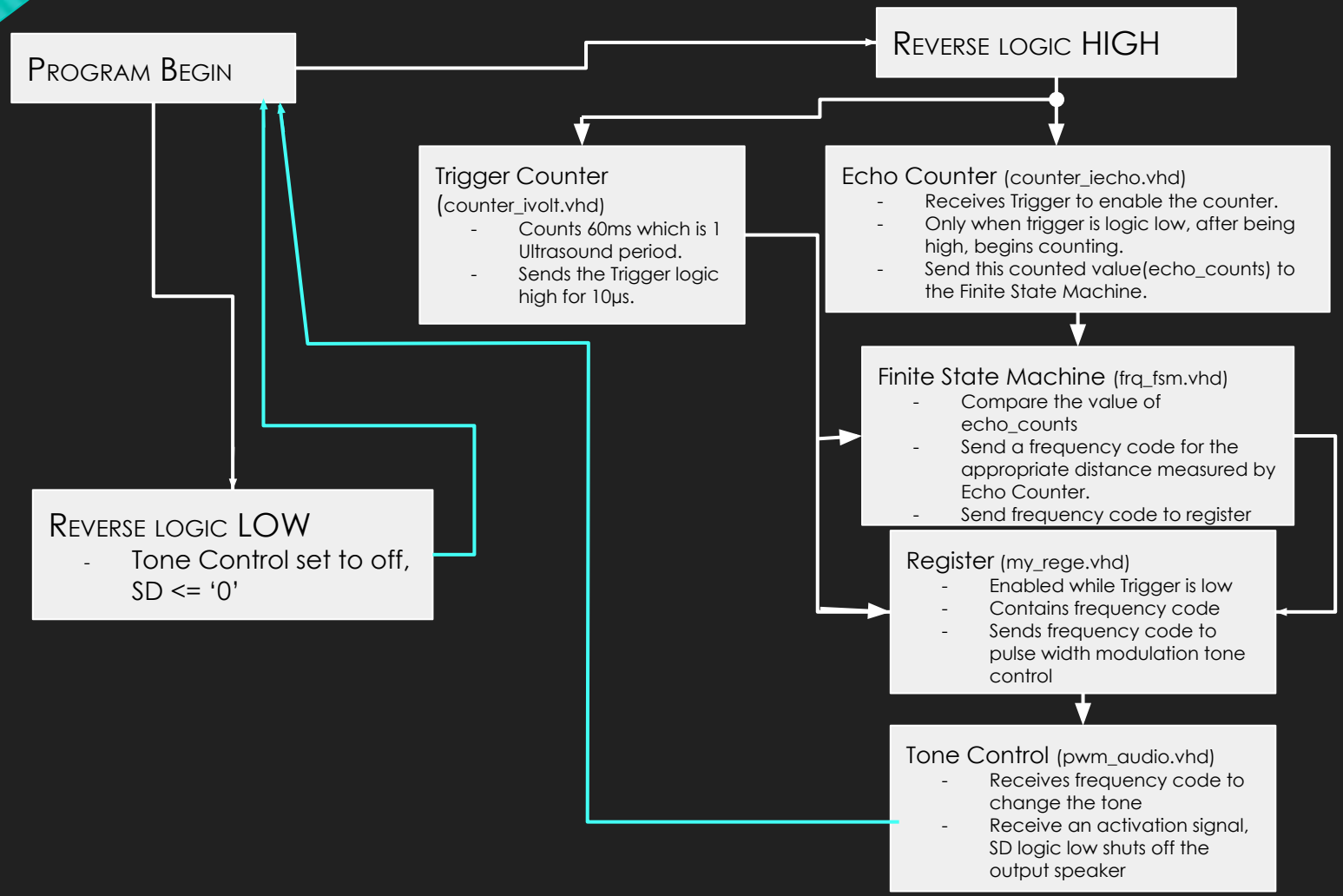
Vehicle Reverse Sensor and Audio Parking Assist System

John Akroush, John Kastran, Andrew Lee

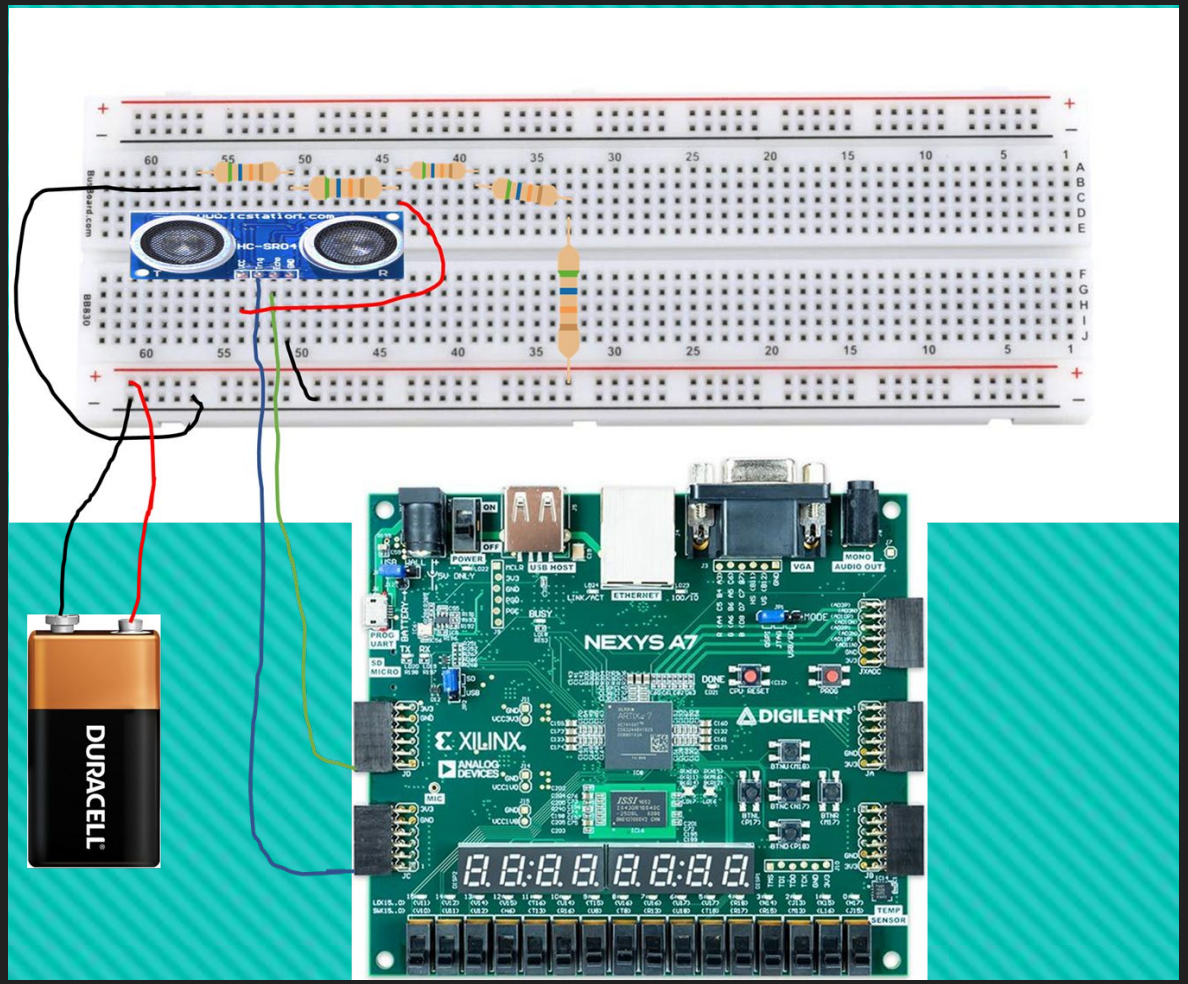
Introduction and Motivation

- "On average, more than 50,000 crashes occur in parking lots and parking garages annually" (ehstoday.com)
- Helpful in situations where you may not be able to see when you are reversing.
- Useful in night-time situations.
- Can help decrease the number of accidents that occur during parking.

Flow Chart


Program Begin
Reverse logic HIGH
Trigger Counter (counter_ivolt.vhd)
- Counts 60ms which is 1 Ultrasound period.
- Sends the Trigger logic high for 10µs.
Echo Counter (counter_iecho.vhd)
- Receives Trigger to enable the counter.
- Only when trigger is logic low, after being high, begins counting.
- Send this counted value(echo_counts) to the Finite State Machine.
Finite State Machine (frq_fsm.vhd)
- Compare the value of echo_counts
- Send a frequency code for the appropriate distance measured by Echo Counter.
- Send frequency code to register
Reverse logic LOW
- Tone Control set to off, SD <= '0'
Register (my_rege.vhd)
- Enabled while Trigger is low
- Contains frequency code
- Sends frequency code to pulse width modulation tone control
Tone Control (pwm_audio.vhd)
- Receives frequency code to change the tone
- Receive an activation signal, SD logic low shuts off the output speaker

Physical Design



List of Components

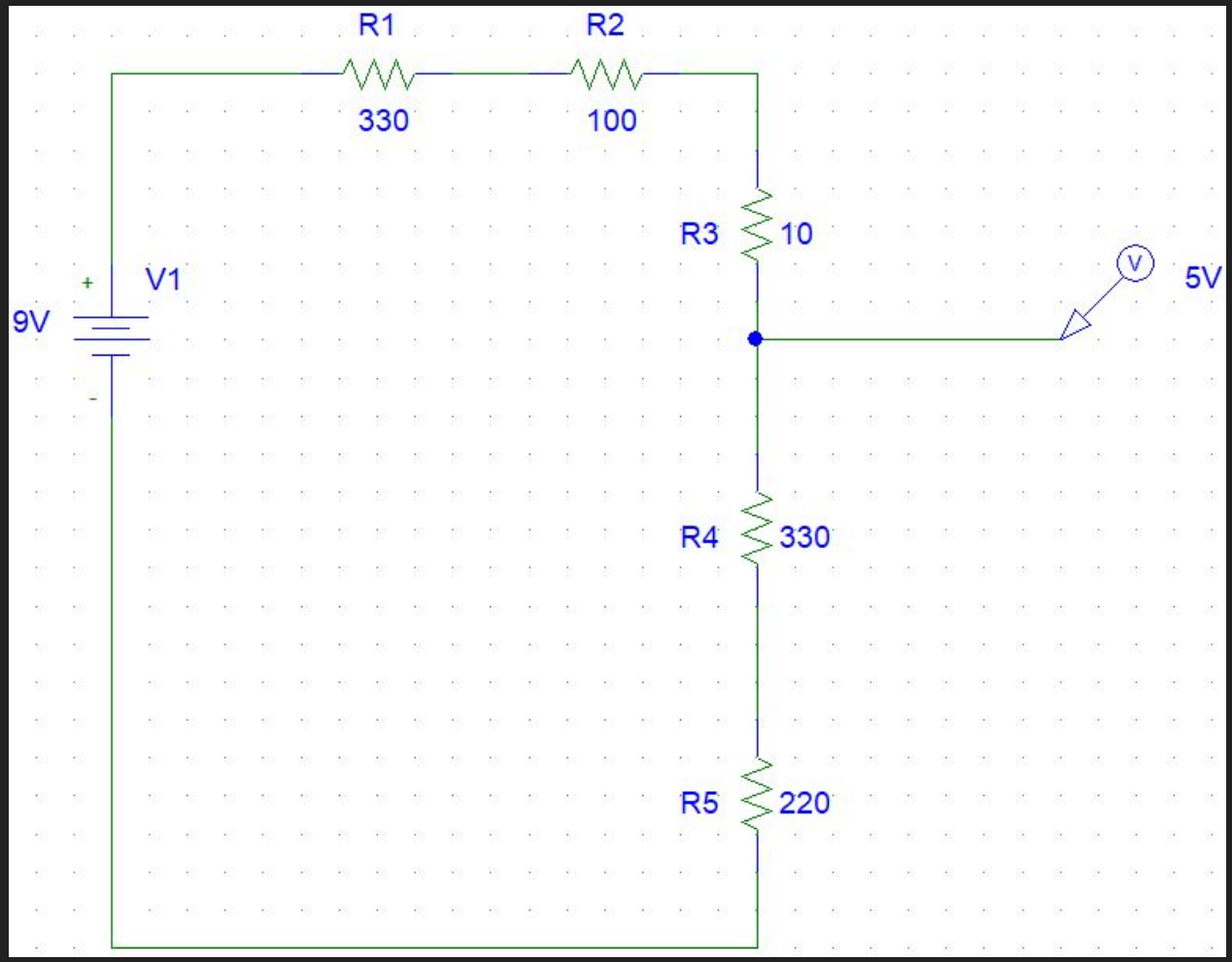
- Ultrasonic Distance Sensor (HC-SR04)
- Nexys A7 50T FPGA
- 10, 100, 220, and 330 Ohm Resistors
- 9V External DC Battery
- Breadboard
- Earphones/speaker
- Wires

Ultrasonic Sensor

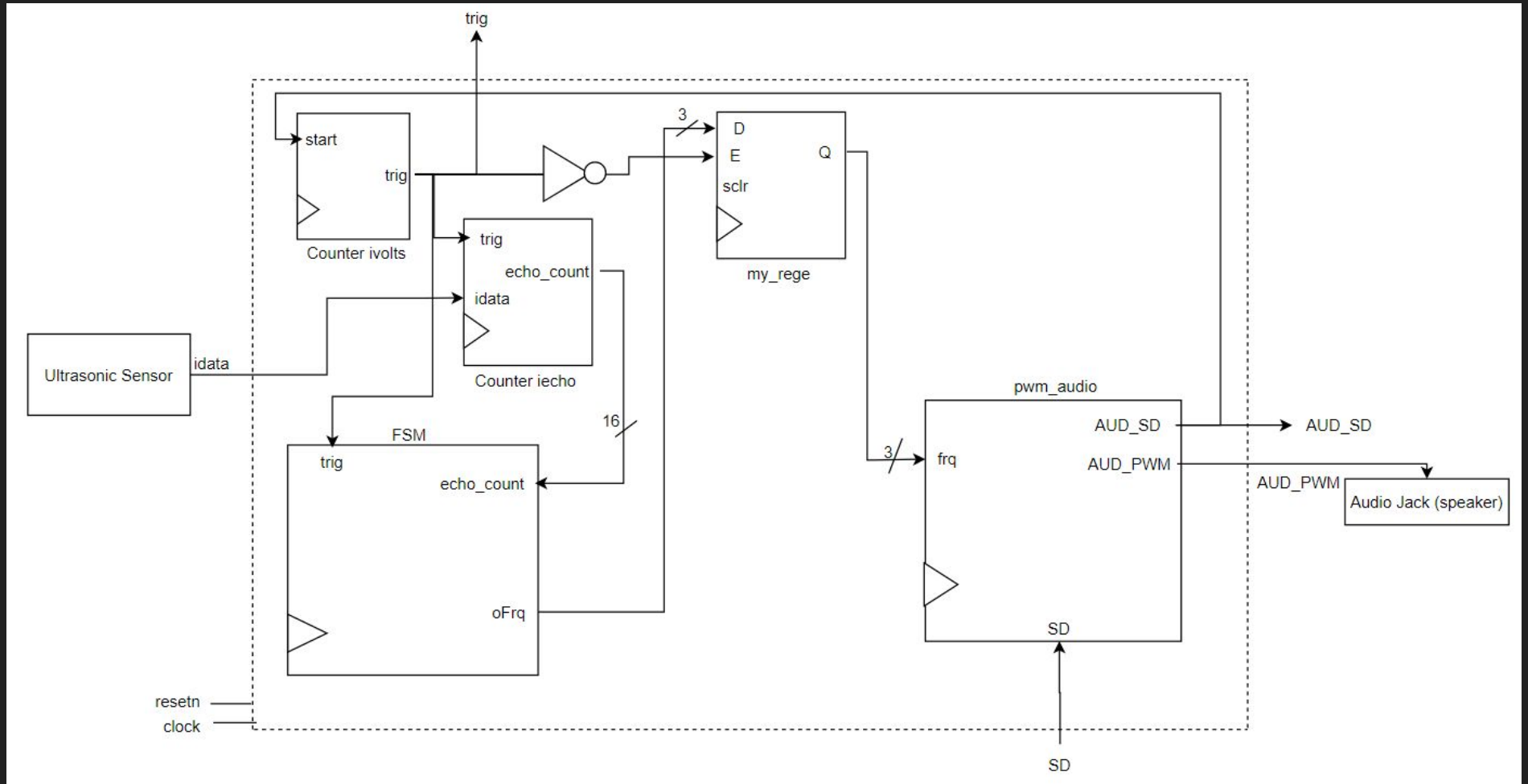
- The Ultrasonic Sensor: HC-SR04
- This sensor requires 5 volts to operate.
- The sensor uses a 10µs trigger pulse (40 kHz) to send a signal, then waits for 60ms to make sure that the trigger signal and the echo returning do not interfere

Obtaining 5V output from 9V Source

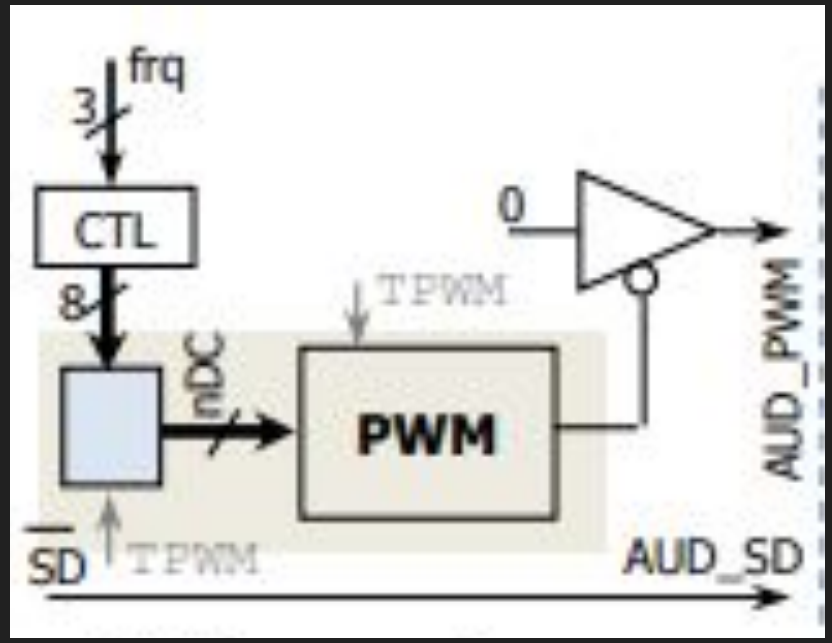
- A Voltage Divider circuit was used to obtain a 5V output for the Ultrasonic Sensor.



Circuit Block Diagram



PWM Block



Counter Code

```
begin

    cntr: process(clock, resetn, start)

        begin
            if resetn = '0' then
                cnt <= (others => '0');
                y <= S0;

            elsif (clock'event and clock = '1') then
                case y is
                    when S0 =>
                        if start = '1' then
                            y <= S1;
                        else end if;

                    when S1 =>
                        if cnt < x"03e8" then
                            cnt <= cnt + 1;
                        else
                            cnt <= cnt + 1;
                            y <= S2;
                        end if;

                    when S2 =>
                        if cnt < x"1b58" then
                            cnt <= cnt + 1;
                        else
                        cnt <= (others => '0');
                            y <= S0;
                        end if;
                end case;
            else end if;

    end process;

    D: process(y)

        begin
        case y is
            when S0 => trig <= '0';
            when S1 => trig <= '1';
            when S2 => trig <= '0';
        end case;

    end process;
```

ivolts

```
begin
echo_count <= cnt;
    counter: process(clock, resetn, trig, idata)
        begin
            if resetn = '0' then
                cnt <= (others => '0');
                y <= S0;

            elsif (clock'event and clock = '1') then
                case y is
                    when S0 =>
                    if trig = '1' then y <= S1;
                          else y <= S0; end if;--cnt <= (others => '0'); y <= S0; end if;
                    when S1 =>
                    if trig = '0' then y<= S2;
                          else y <= S1; end if;
                    when S2 =>
                    if idata = '1' then y <= S3;
                          else y <= S2; end if;
                    when S3 =>
                    if idata = '0' then y <= S0;
                          else cnt <= cnt+1; y<=s3; end if;
                end case;
            else end if;
     end process;
```

iecho

FSM Code

```
begin

--centi <= to_integer(unsigned(echo_count));
--centi <= CONV_STD_LOGIC_VECTOR((echo_count/58),16);
--1342/58 = 85cm
--B54/58 = 50cm
--5AA/58 = 18cm
--244/58 = 10cm
    states: process(centi, clock, resetn, trig)
    begin
         if resetn = '0' then
            y <= S0;
         elsif (clock'event and clock = '1') then
            if trig = '1' then y <= S0; else
                case y is
                    when S0 => if echo_count < x"0244" then  y <= S0;
                               else y <= S1; end if;

                    when S1 => if echo_count > x"1342" then y <= S1;
                               else y <= S2; end if;

                    when S2 => if echo_count > x"0B54" then y <= S2;
                               else y <= S3; end if;

                    when S3 => if echo_count > x"05AA" then y <= S3;
                               else y <= S4; end if;

                    when S4 => if echo_count > x"255" then y <= S4;
                               else end if;
                end case;
            end if;
        end if;
    end process;

    values: process(y)
        begin
            case y is
              when S0 => oFrq <= "001"; --Most Severe
              when S1 => oFrq <="011"; --Danger
              when S2 => oFrq <="100"; --Close
              when S3 => oFrq <="101"; --Something is far behind
              when S4 =>
        end case;
    end process;
```

Top File Code

```
begin

AUD_SD <= AUD_SDtop;
trig <= trigVoltsTop;
regeTrigTop <= NOT (trigVoltsTop);

fa: counter_ivolts port map (clock => clock, resetn => resetn, start => AUD_SDtop, trig => trigVoltsTop);
fb: counter_iecho port map (clock => clock, resetn => resetn, idata => idata, trig => trigVoltsTop, echo_count => ieCountOutTop);

fc: frq_FSM port map (clock => clock, resetn => resetn, trig => trigVoltsTop, echo_count => ieCountOutTop, oFrq => FSMoutTop);
fd: my_rege
        generic map( N => 3)
        port map (clock => clock, resetn => resetn, E => regeTrigTop, sclr => '0', D => FSMoutTop, Q => RegOutTop);
fl: pwm_audio port map ( resetn => resetn, clock => clock, frq => RegOutTop, SD => SD, AUD_PWM => AUD_PWM, AUD_SD => AUD_SDtop);

end Behavioral;
```

Challenges

- Remote collaboration-
  - Working remotely on hardware, while easier with the availability and high functionality of meeting and collaboration software, is still a challenge nonetheless
- FSM module code-
  - The FSM took multiple iterations to find the correct state sequencing
- Timing accuracy with the ultrasonic distance sensor-
  - Without documentation it was impossible to create a functioning circuit, however once we found out how the timing of the sensor was needed it was much easier
- Audio Shutdown-
  - We had a problem finding a way to dynamically update the Audio Shutdown using our circuit

Improvements

- Implementing a distance controlled LED brightness to replicate a vehicle's headlights/rear lights as it would approach oncoming vehicles.
- Making the tone toggle instead of be a static tone.

Video Demonstration

https://youtu.be/KUatEbh2To4

Works Cited

- https://www.ehstoday.com/safety/article/21917821/black-friday-alert-driving-through-a-parking-lot-is-still-driving
- Dr. Llamocca's VHDL Coding for FPGA's Website