Bilkent University
Department of Electrical and Electronics Engineering
EEE-102 Term Project

Smart Intersection with Active Traffic Lights System

Uğur Efe Yiğit

EEE-102 Section 4

22302023

YouTube Link: https://youtu.be/GJ8IfE_5N2o

Objective

The goal of the project was to develop an Active Traffic Light System implementation with the help of FPGA technology to ensure proper traffic flow at intersections by dynamically changing the sequence of the traffic light with respect to real-time vehicle detection. A Basys 3 FPGA board and HC-SR04 ultrasonic sensors have been combined to prioritize the main roads by controlling the green light durations of the joining roads at a 4-way intersection, hence reducing unnecessary wait times, and ensuring traffic efficiency.

Introduction

Intersections can be one of the hardest parts of urban traffic to control; they spur congestion and subsequent increase in emissions for the simple reason that the available sequences in the traffic lights are not good enough. Traditional traffic light control systems have fixed timers that do not change with the prevailing traffic conditions. This project suggests a new solution incorporating the ATLS, a combination of digital design methodologies and sensor technology to come up with a responsive traffic light control system.

The ATLS uses a Basys 3 FPGA board for its robust digital logic capabilities and HC-SR04 ultrasonic sensors to detect vehicle presence. The system is aimed to reduce traffic congestion and wasted idle time by dynamically adjusting the signal phase using real-time information, hence making the traffic control system more efficient and smarter.

Methodology

The ATLS system is designed for an intersection with two main roads (L1 & L3) and two joining roads (L2 & L4). Ultrasonic sensors are placed on the joining roads to detect waiting vehicles. The FPGA controls the traffic light transitions and processes sensor data to manage the traffic lights dynamically.

System Overview:

- Main Roads (L1 & L3): Given higher priority due to constant traffic flow. No sensors are placed on these roads.
- Joining Roads (L2 & L4): Equipped with ultrasonic sensors to detect waiting vehicles. Traffic lights on these roads change based on sensor data.

Hardware Components:

- BASYS 3 FPGA Board: Manages logic for traffic light transitions and interfaces with sensors.
- HC-SR04 Ultrasonic Sensors (2 units): Detect vehicles on joining roads.
- Traffic Light Modules (4 units): Represent traffic lights for each road. Simulated using LEDs.
- MT3608 Step-Up Voltage Regulator Module: Provide the ultrasonic sensors the 5V input they require to function from the BASYS 3's 3.3V output.

Design Specifications

As mentioned in the methodology part of this report, the main components of this project are the ultrasonic sensors and the traffic lights.

Figure 1: The traffic light module which has 3 inputs for the LEDs and a common ground input.

Figure 2: The HCSR-04 ultrasonic sensor module with 5V VCC, trigger and ground inputs and the echo output.

Ultrasonic Sensor Usage

The HCSR-04 sensor essentially provides the data regarding whether a vehicle is waiting at a joining road or not. There are 4 steps to acquire the desired distance output from the ultrasonic sensor. And these are:

1. **Trigger Pulse Generation**: A component, TriggerGenerator, programmed on VHDL, generates a 10-microsecond trigger pulse each 60 milliseconds to start the measurement process. This pulse is directly connected to the trigger input of the ultrasonic sensor using the IO pins of my FPGA.
2. **Reading The Echo Signal:** Another component, EchoDistanceMeasurement, reads the echo output of the signal. This signal is an input of the FPGA board. The echo signal becomes high the moment ultrasonic pulse is sent out and remains high until the echo is received back by the sensor.
3. **Timing the Echo Signal:** A process generates a pulse every microsecond by counting clock cycles of the 100 MHz system clock. Every 50 clock cycles (500 ns for each half-period), the microsecond signal toggles, creating a 1-microsecond pulse. Another process counts the number of microseconds during which the echo signal is high. 2 separate signals echo_p, echo_n is used to properly time the signal as follows. On the rising edge, when echo goes high, the counter starts counting microseconds. On the falling edge of the echo signal, the counter stops, and the round-trip time of the ultrasonic pulse is successfully measured.
4. **Converting Echo Duration to Distance:** Firstly, since we are measuring the round-trip duration of the ultrasonic pulse in microseconds, we need to know the speed of sound in cm/us. By dividing the round-trip time by 2, we can calculate how long does it take for the sound wave to go to the nearest object. This quantity can then be multiplied by the speed of sound to compute the distance measured by the sensor.

   Speed of Sound = 0.0343 cm/us
   Distance = Round-Trip Time / 2 x 0.0343 centimeters.
   Final output of the component: Distance (in 10^-7 m) = Round Trip Time x 1715

Car Detection Logic:

The ultrasonic sensor module outputs a distance in 10^-7 meters. Since the actual width of the roads on my actual model intersection are around 15 centimeters. The 15 centimeters was converted into 10^-7 m format, and then a simple if loop was enough to determine if a car was indeed at the 15-centimeter threshold.

The Traffic Light States and Logic:

There are a total of 14 states in the traffic light logic. Each state represents a specific traffic light configuration. The states include initialization, normal operation phases for each road, and an error state. The transitions between these states are governed by sensor inputs, timing rules, and safety considerations.

Here are the states, listed in their respective orders. and their meanings:

ALL_RED_INIT: Initialization state where all traffic lights are red. (This is not in the loop.)
RY1: Ready state for Main Road 1 (L1), showing Red and Yellow.
GREEN1: Main Road 1 (L1) is green.
YELLOW1: Main Road 1 (L1) is yellow.
RY2: Ready state for Joining Road 1 (L2), showing Red-Yellow.
GREEN2: Joining Road 1 (L2) is green.
YELLOW2: Joining Road 1 (L2) is yellow.
RY3: Ready state for Main Road 2 (L3), showing Red-Yellow.
GREEN3: Main Road 2 (L3) is green.
YELLOW3: Main Road 2 (L3) is yellow.
RY4: Ready state for Joining Road 2 (L4), showing Red-Yellow.
GREEN4: Joining Road 2 (L4) is green.
YELLOW4: Joining Road 2 (L4) is yellow.

ERROR_STATE: The error or night state where the main roads are flashing yellow and joining roads are flashing red, allowing passage for all cars. The color of their flashing lights indicates the priority of roads. Since main roads have priority, they are flashing yellow, indicating that they have the priority of the road. Since joining roads have to yield to cars on the main road, their flashing red light indicates that they can pass but the main road has priority.

Led Control Logic:

The 4 led outputs L1, L2, L3 and L4 are 3-bit outputs where the MSB represents the red light and the LSB represents the green light of that specific traffic light.

Since each state represents a different traffic light configuration, a LED control process assigns L1, L2, L3, L4 their respective logic vectors according to the states. Since there are a total of 14 states, I will not be writing each LED configuration. Here is an example, rest can be found in the appendix on TrafficLogic.vhd.

```
when GREEN4 =>
```

```
L1 <= "100";
L2 <= "100";
L3 <= "100";
L4 <= "001";
```

Conclusion

The ATLS project aimed to increase the efficiency and safety in the management of traffic flow at intersections. A new approach to this was there with the application of the FPGA technology and ultrasonic sensors. The successful implementation and the testing of the ATLS in this project really demonstrated the potential that it brought to substantially improve the traffic flow and reduce the idle time of the vehicle within an intersection. This report outlines the principal findings, the success of the implemented system, and future directions to improve the ATLS. While developing the project, a prototype was created as well to ensure correct state transitions and sensor functionality.

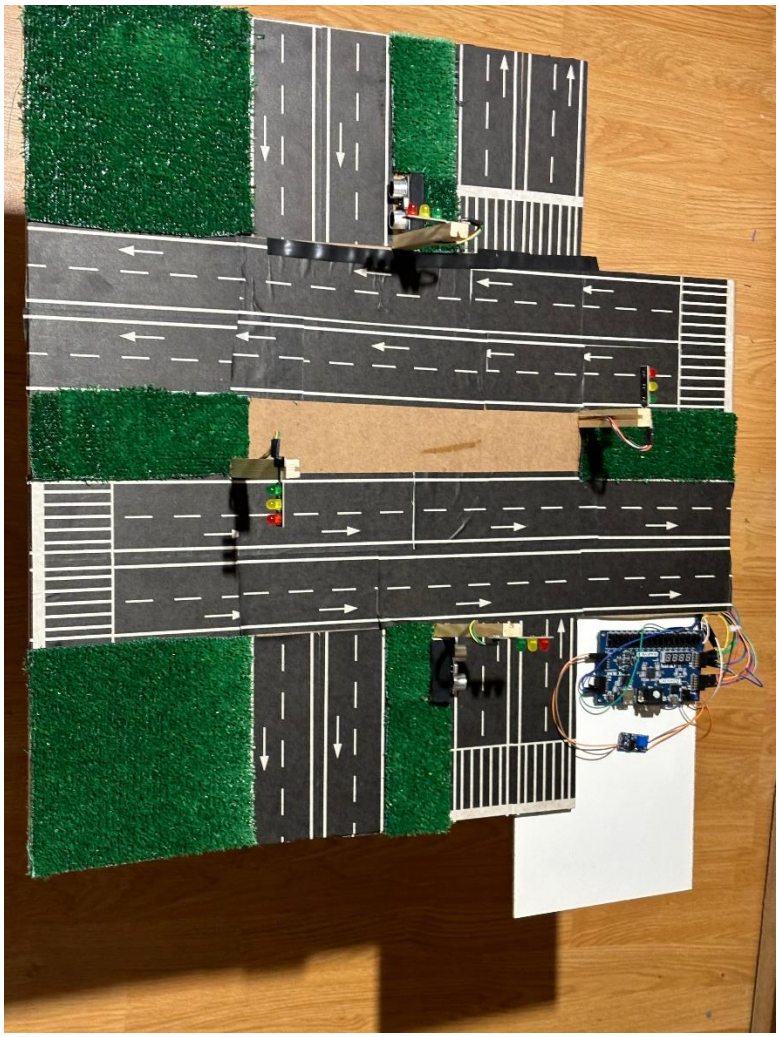

Figure 3: The project setup

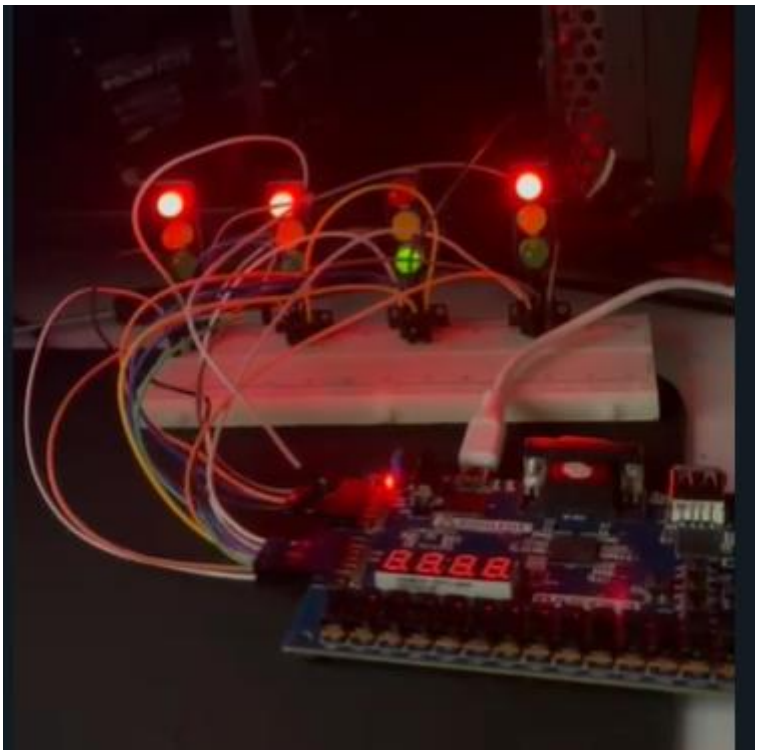

Figure 4: The project prototype

Principal Findings

1) Dynamic Traffic Control

The ATLS module managed the traffic flow effectively and controlled the sequences of the traffic lights dynamically based on real-time vehicle detections. Actually, this adaptive scheme decreased the unnecessary waiting time and thereby increased the intersection's overall efficiency.

2) Accurate Vehicle Detection

The ATLS system detected the presence of the vehicles located on the joining roads accurately, with the help of HC-SR04 ultrasonic sensors. These sensors provided reliable distance measurement that considered whether a vehicle is waiting on the intersection.

3) Efficient FSM Design

The FSM design had the successful approach in the four-roads traffic light state management. The FSM was built with 14 states that guaranteed each road the appropriate traffic signal from

the real-time road data, followed with predefined timing rules. The system gave priority to the main roads (L1 and L3) and dynamically adjusted the green light duration for the joining roads (L2 and L4) based on the presence of the vehicles.

4) Safety and Reliability

The safety of the road users was given paramount importance in the design of the ATLS. The system incorporates a minimum amount of green and yellow phases to guarantee a safe transition of the traffic lights. Additionally, an all-red transition phase was incorporated in the system to safely manage the change of right-of-way between the roads. An error state was also included to handle any unexpected issues, therefore, to ensure that the traffic lights defaulted to a safe configuration in case of a malfunction.

Success of the Implemented System

The ATLS project was successful in several key areas:

Functional Implementation: The system was implemented successfully on the Basys 3 FPGA board, which proved the feasibility of using FPGA technology in real-time traffic management.

Accurate Detection and Control: The ultrasonic sensors used in the project were adequate for the precise detection of the vehicles, and the FSM effectively controlled the traffic lights according to this input.

Improved Traffic Flow: Since the intersection traffic light sequence is being dynamically adjusted according to the real-time condition, the flow of traffic has been significantly improved, thereby reducing idle time, and eventually raising the efficiency of the travel.

Future Work

Although the ATLS project is very strong improvement in the traffic management, there is still some future work that can be done:

More Accurate Vehicle Detection: Further sensors or other more advanced sensor technology can be used in future versions of ATLS to make vehicle detection more accurate. This can be done via the combination of ultrasonic, infrared, and camera-based sensors.

Coordination between Different Intersections: By implementing inter-communication between different intersections, they would be able to provide a traffic management system that is coordinated for the whole area. This will further provide efficient routing of the traffic in a city environment, which will reduce congestion and enhance the flow of the vehicles.

Scalable and Flexible: Generalization of this scheme to manage more than four roads or more lanes at one crossing would certainly increase its domain of use. Additionally, machine learning algorithms will be developed so the system can adapt with time.

User Interface and Monitoring: A user-friendly interface for monitoring and controlling the ATLS in real time should be developed. It should possess a graphical dashboard that shows traffic status, sensor data, and system status.

Smart City Infrastructure: Integration with larger smart city infrastructure, such as ITS and urban data platforms, could yield further benefits. This integration would allow for more holistic management of urban mobility from different sources in the hope of optimally regulating traffic flow and reducing emissions.

Final Thoughts

The ATLS project proved the applicability of the FPGA technology and ultrasonic sensors in developing an intelligent and responsive system to control the traffic lights. By changing the sequence and duration of the traffic lights, the ATLS has successfully improved the traffic flow and reduced the idle time of vehicles waiting at the intersection. This project opens doors for many advancements in intelligent traffic management systems for urban transportation networks, making them efficient, safe, and sustainable. The lessons and experience drawn from the project lay a good foundation for further research and development in the domain of intelligent transportation systems.

Appendix:

TrafficLogic.vhd:

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

entity TrafficLights is
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        u2echo : in std_logic;
        u4echo : in std_logic;
        u2trigger : out std_logic;
        u4trigger : out std_logic;
        error_switch : in std_logic;
        carL2 : out std_logic;
        carL4 : out std_logic;
        L1, L2, L3, L4 : out STD_LOGIC_VECTOR(2 downto 0)
    );
end TrafficLights;

architecture Behavioral of TrafficLights is
    type State_Type is (ALL_RED_INIT,ERROR_STATE,
                        RY1, GREEN1, YELLOW1,
                        RY2, GREEN2, YELLOW2,
                        RY3, GREEN3, YELLOW3,
                        RY4, GREEN4, YELLOW4);

    signal state : State_Type := ALL_RED_INIT;
```

```
    signal count : INTEGER range 0 to 24; -- Maximum cycles for the longest phase
    signal clk_1hz : STD_LOGIC;
    signal DistanceL2, DistanceL4 : integer := 0;
    signal car_at_L2, car_at_L4 : boolean := false;
    signal error : std_logic;
    constant CAR_DETECTION_THRESHOLD : integer := 1500000;  -- Equivalent to 15 cm


begin

    Clock_Divider_inst : entity work.Clock_Divider
        port map (
            clk_in => clk,
            reset => reset,
            clk_out => clk_1hz
        );
    -- Trigger generator for L2
    TriggerGenL2 : entity work.TriggerGenerator
        port map (
            clk => clk,
            trigger_out => u2trigger
        );

    -- Trigger generator for L4
    TriggerGenL4 : entity work.TriggerGenerator
        port map (
            clk => clk,
            trigger_out => u4trigger
        );
```

```
-- Distance measurement for L2
EchoDistL2 : entity work.EchoDistanceMeasurement
    port map (
        clk => clk,
        echo_in => u2echo,
        distance_out => DistanceL2
    );

-- Distance measurement for L4
EchoDistL4 : entity work.EchoDistanceMeasurement
    port map (
        clk => clk,
        echo_in => u4echo,
        distance_out => DistanceL4
    );

ErrorProcess : process(error_switch)
    begin
    if error_switch = '1' then
        error <= '1';
    else
        error <= '0';
    end if;
end process;

-- Car detection process
CarDetectionProcess: process(clk)
begin
```

```
    if rising_edge(clk) then
        if reset = '1' then
            car_at_L2 <= false;
            car_at_L4 <= false;
        else
            -- Update car_at_L2 based on DistanceL2
            if DistanceL2 <= CAR_DETECTION_THRESHOLD then
                car_at_L2 <= true;
            else
                car_at_L2 <= false;
            end if;

            -- Update car_at_L4 based on DistanceL4
            if DistanceL4 <= CAR_DETECTION_THRESHOLD then
                car_at_L4 <= true;
            else
                car_at_L4 <= false;
            end if;
        end if;
    end if;
end process CarDetectionProcess;

   process(clk_1hz, reset, error)
begin
    if reset = '1' then
        state <= ALL_RED_INIT;
        count <= 0;

    elsif error = '1' then
```

```
    state <= ERROR_STATE;  -- Go to ERROR_STATE if switch0 is '1'
    count <= 0;

elsif rising_edge(clk_1hz) then
    case state is

        when ERROR_STATE =>
            -- Toggle count to create flashing effect
            if count = 0 then
                count <= 1;
            else
                count <= 0;
            end if;

        when ALL_RED_INIT =>
            if count < 2 then
            state <= ALL_RED_INIT;
            count <= count + 1;
            else
            state <= RY1;
            count <= 0;
            end if;

        when RY1 =>
            if count < 3 then
            state <= RY1;
            count <= count + 1;
            else
            state <= GREEN1;
```

```
    count <= 0;
    end if;

when GREEN1 =>
    if count < 24 then
    state <= GREEN1;
    count <= count + 1;
    else
    state <= YELLOW1;
    count <= 0;
    end if;

when YELLOW1 =>
    if count < 3 then
    state <= YELLOW1;
    count <= count + 1;
    else
    state <= RY2;
    count <= 0;
    end if;
when RY2 =>
    if count < 3 then
    state <= RY2;
    count <= count + 1;
    else
    state <= GREEN2;
    count <= 0;
    end if;
```

```
when GREEN2 =>
    if car_at_L2 and count < 16 then
    state <= GREEN2;
    count <= count + 1;
    elsif not car_at_L2 and count < 8 then
    state <= GREEN2;
    count <= count + 1;
    else
    state <= YELLOW2;
    count <= 0;
    end if;
 when YELLOW2 =>
    if count < 3 then
    state <= YELLOW2;
    count <= count + 1;
    else
    state <= RY3;
    count <= 0;
    end if;

when RY3 =>
    if count < 3 then
    state <= RY3;
    count <= count + 1;
    else
    state <= GREEN3;
    count <= 0;
    end if;
when GREEN3 =>
```

```
    if count < 24 then
    state <= GREEN3;
    count <= count + 1;
    else
    state <= YELLOW3;
    count <= 0;
    end if;
 when YELLOW3 =>
    if count < 3 then
    state <= YELLOW3;
    count <= count + 1;
    else
    state <= RY4;
    count <= 0;
    end if;

when RY4 =>
    if count < 3 then
    state <= RY4;
    count <= count + 1;
    else
    state <= GREEN4;
    count <= 0;
    end if;
when GREEN4 =>
    if car_at_L4 and count < 16 then
    state <= GREEN4;
    count <= count + 1;
    elsif not car_at_L4 and count < 8 then
```

```
            state <= GREEN4;
            count <= count + 1;
            else
            state <= YELLOW4;
            count <= 0;
            end if;
          when YELLOW4 =>
            if count < 3 then
            state <= YELLOW4;
            count <= count + 1;
            else
            state <= RY1;
            count <= 0;
            end if;
    end case;
    end if;
end process;


-- LED Control Logic
process(state,count)
begin

    -- Control each LED based on the current state
    case state is
       when ERROR_STATE =>
       if count = 1 then
         L1 <= "010"; -- L1 yellow
         L2 <= "100"; -- L2 red
```

```
   L3 <= "010"; -- L3 yellow
   L4 <= "100"; -- L4 red
else
   L1 <= "000"; -- L1 yellow
   L2 <= "000"; -- L2 red
   L3 <= "000"; -- L3 yellow
   L4 <= "000"; -- L4 red
end if;



when ALL_RED_INIT =>
    L1 <= "100";
    L2 <= "100";
    L3 <= "100";
    L4 <= "100";
when RY1 =>
    L1 <= "110";
    L2 <= "100";
    L3 <= "100";
    L4 <= "100";
when YELLOW1 =>
    L1 <= "010";
    L2 <= "100";
    L3 <= "100";
    L4 <= "100";
when GREEN1 =>
    L1 <= "001";
    L2 <= "100";
    L3 <= "100";
```

```
    L4 <= "100";
when RY2 =>
    L1 <= "100";
    L2 <= "110";
    L3 <= "100";
    L4 <= "100";
when YELLOW2 =>
    L1 <= "100";
    L2 <= "010";
    L3 <= "100";
    L4 <= "100";
when GREEN2 =>
    L1 <= "100";
    L2 <= "001";
    L3 <= "100";
    L4 <= "100";
when RY3 =>
    L1 <= "100";
    L2 <= "100";
    L3 <= "110";
    L4 <= "100";
when YELLOW3 =>
    L1 <= "100";
    L2 <= "100";
    L3 <= "010";
    L4 <= "100";
when GREEN3 =>
    L1 <= "100";
    L2 <= "100";
```

```
            L3 <= "001";
            L4 <= "100";
          when RY4 =>
            L1 <= "100";
            L2 <= "100";
            L3 <= "100";
            L4 <= "110";
          when YELLOW4 =>
            L1 <= "100";
            L2 <= "100";
            L3 <= "100";
            L4 <= "010";
          when GREEN4 =>
            L1 <= "100";
            L2 <= "100";
            L3 <= "100";
            L4 <= "001";
        end case;

    end process;

process(car_at_L2, car_at_L4)
    begin
        if car_at_L2 then
          carL2 <= '1';
        else
          carL2 <= '0';
        end if;
        if car_at_L4 then
```

```
          carL4 <= '1';
        else
          carL4 <= '0';
        end if;
 end process;

end Behavioral;
```

ClockDivider.vhd:

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

entity Clock_Divider is
    Port (
        clk_in : in STD_LOGIC;   -- Input clock 100 MHz
        reset : in STD_LOGIC;    -- Reset signal
        clk_out : out STD_LOGIC  -- Output clock at 2 Hz
    );
end Clock_Divider;

architecture Behavioral of Clock_Divider is
    constant CLK_DIVIDER : integer := 25000000; -- Half the period for a 2 Hz clock from
100 MHz
    signal clk_counter : integer range 0 to CLK_DIVIDER := 0;
    signal internal_clk : STD_LOGIC := '0';
begin

    process(clk_in, reset)
```

```
    begin
        if reset = '1' then
            clk_counter <= 0;
            internal_clk <= '0';
        elsif rising_edge(clk_in) then
            if clk_counter < CLK_DIVIDER - 1 then
                clk_counter <= clk_counter + 1;
            else
                clk_counter <= 0;
                internal_clk <= not internal_clk;  -- Toggle the clock every 25,000,000 cycles
            end if;
        end if;
    end process;

    clk_out <= internal_clk;  -- Output the toggled clock

end Behavioral;
```

TriggerGenerator.vhd:

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

entity TriggerGenerator is
    Port (
        clk : in STD_LOGIC; -- 100 MHz internal clock
        trigger_out : out STD_LOGIC
    );
end TriggerGenerator;
```

```
architecture Behavioral of TriggerGenerator is
    signal count : integer := 0;
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if count = 0 then
                trigger_out <= '1';
            elsif count = 1000 then  -- 10 microseconds at 100 MHz
                trigger_out <= '0';
            end if;

            if count = 6000000 then  -- 60 milliseconds at 100 MHz
                count <= 0;
            else
                count <= count + 1;
            end if;
        end if;
    end process;
end Behavioral;
```

EchoDistanceMeasurement.vhd:

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

entity EchoDistanceMeasurement is
```

```
    Port (
        clk : in STD_LOGIC; -- 100 MHz internal clock
        echo_in : in STD_LOGIC;
        distance_out : out integer  -- distance in 10^-7 meters
    );
end EchoDistanceMeasurement;

architecture Behavioral of EchoDistanceMeasurement is
    signal microsecond : STD_LOGIC := '0';
    signal counter : integer := 0;
    signal echo_c : STD_LOGIC := '0';
    signal echo_p : STD_LOGIC := '0';
    signal microsec_counter : integer := 0;
begin
    -- Process to generate a microsecond pulse
    process(clk)
    variable count0 : integer := 0;  -- Adjust for 100 MHz clock
    begin
        if rising_edge(clk) then
            if count0 = 49 then  -- Create a 1 microsecond tick
                microsecond <= not microsecond;
                count0 := 0;
            else
                count0 := count0 + 1;
            end if;
        end if;
    end process;

    -- Process to increment counter on every microsecond when echo is high
```

```
    process(microsecond)
    begin
        if rising_edge(microsecond) then
            echo_p <= echo_curr;
            echo_c <= echo_in;

            if echo_c = '1' then
                counter <= counter + 1;
            end if;

            -- Falling edge detected on the ECHO signal
            if echo_p = '1' and echo_c = '0' then
                -- Calculate the distance based on the counter and the predefined scale factor
                distance_out <= counter * 1715; -- Scale factor, 343*10/2
                counter <= 0; -- Reset the counter
            end if;
        end if;
    end process;
end Behavioral;
```