**Report: Car Speed Detector System**

**Project 2 Name: Car Speed Detector System**

**Name: Parth Patel**

**Enrollment Number: 202411044**

**Introduction**

The Car Speed Detector System is an FPGA-based real-time monitoring project designed to measure and display the speed of a car using two IR sensors. The system uses the time difference between the triggering of two sensors and calculates the car's speed based on the fixed distance between them. The calculated speed is displayed on a multiplexed seven-segment display, while an RGB LED provides immediate visual feedback on the speed category (low, medium, or high). This system finds applications in traffic monitoring, automated toll booths, and speed-limit enforcement systems.

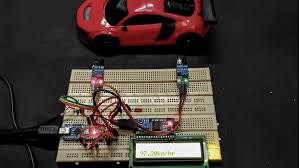
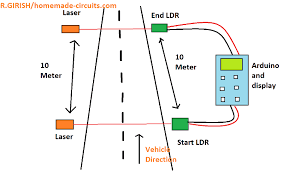
**Objective:**

The aim is to design an FPGA-based system to measure and display the speed of a car using two IR sensors, while providing speed feedback through RGB LEDs and a seven-segment display.

Features:

1. Speed Calculation:
   * Detects the time difference between two IR sensors as the car passes.
   * Uses a known distance between the sensors to calculate speed in cm per clock cycle.
2. Seven-Segment Display:
   * Displays the calculated speed on six multiplexed seven-segment displays.
   * Refreshes displays using a controlled refresh rate.
3. RGB LED Feedback:
   * Displays speed categories:
     + Red: High speed (> 50 cm/clock cycle).
     + Green: Low speed (< 20 cm/clock cycle).
     + Blue: Medium speed.

**Diagram of circuit**



**Design Details:**

1. **Inputs:**
   * **clk: System clock for all operations.**
   * **ir\_sensor1, ir\_sensor2: Signals triggered by cars passing through sensors.**
2. **Outputs:**
   * **abcdefg: Seven-segment display segments.**
   * **an: Enable signals for multiplexing seven-segment displays.**
   * **rgb: RGB LED outputs (3 bits for Red, Green, and Blue).**
3. **Parameters:**
   * **DISTANCE\_CM = 10: Fixed distance between the two IR sensors.**
4. **Registers:**
   * **timer: Global clock cycle counter.**
   * **time1, time2: Time captures for ir\_sensor1 and ir\_sensor2 events.**
   * **time\_diff: Time difference between sensor triggers.**
   * **speed\_reg: Stores the calculated speed.**
   * **digit: Holds the current digit being displayed on the seven-segment display.**
   * **active\_digit: Determines the active display in multiplexing.**
   * **refresh\_counter: Counter to control multiplexing refresh rate.**
5. **Key Functional Blocks:**
   * **State Machine:**
     + **Manages sensor-triggered events.**
     + **Records timestamps for each sensor's activation and resets state.**
   * **Speed Calculation:**
     + **Computes speed using the formula: Speed=Distance (cm)×100Time Difference (clock cycles)\text{Speed} = \frac{\text{Distance (cm)} \times 100}{\text{Time Difference (clock cycles)}}Speed=Time Difference (clock cycles)Distance (cm)×100​**
   * **RGB LED Control:**
     + **Categorizes and outputs speed levels using RGB LEDs.**
   * **Seven-Segment Decoder:**
     + **Converts digits to their respective segment patterns.**
   * **Multiplexing Logic:**
     + **Cycles through six displays for speed visualization.**

**Simulation and Testing:**

1. **Input Conditions:**
   * **Simulate sensor triggers (ir\_sensor1 and ir\_sensor2) at known intervals.**
   * **Vary clock frequencies to observe speed variations.**
2. **Expected Outputs:**
   * **Correct speed displayed on seven-segment displays.**
   * **RGB LED changes according to speed thresholds.**

**Code Analysis:**

1. **Strengths:**
   * **Well-organized structure using state machines and parameterization.**
   * **Efficient multiplexing logic for seven-segment displays.**
   * **Real-time feedback via RGB LEDs enhances user interaction.**
2. **Potential Improvements:**
   * **Add debouncing logic for IR sensors to prevent false triggers.**
   * **Introduce scaling to adapt speed values to different units (e.g., km/hr).**
   * **Optimize timing counter for higher precision using adjustable clock division.**

**Verilog code and it working**

1. Clock Counter

The timer register increments on every clock pulse to keep track of elapsed clock cycles. It acts as a time reference for calculating the difference between the events detected by the two IR sensors.

**always @(posedge clk) begin**

**timer <= timer + 1; // Increment clock cycle counter**

**end**

2. State Machine for Sensor Detection

The system uses a simple state machine to capture the timestamps of events detected by the IR sensors:

* State 0: Waits for ir\_sensor1 to be triggered. When it detects a car, it records the current value of timer into time1 and moves to State 1.
* State 1: Waits for ir\_sensor2 to be triggered. It then records the current value of timer into time2 and resets the state to State 0.

**always @(posedge clk) begin**

**if (state == 0 && ir\_sensor1) begin**

**time1 <= timer; // Capture the time when the first sensor is triggered**

**state <= 1; // Transition to wait for second sensor**

**end**

**if (state == 1 && ir\_sensor2) begin**

**time2 <= timer; // Capture the time when the second sensor is triggered**

**state <= 0; // Reset the state to wait for the first sensor again**

**end**

**end**

3.Speed Calculation

When the system captures timestamps from both sensors, it calculates the time difference and determines the speed using the known distance between the sensors.

always @(posedge clk) begin

if (state == 0 && ir\_sensor2 && (time2 > time1)) begin

time\_diff <= time2 - time1; // Time difference in clock cycles

speed\_reg <= (DISTANCE\_CM \* 100) / time\_diff; // Speed in cm per clock cycle

end

end

**4. RGB LED Control**

The system categorizes the calculated speed into three levels:

* **Red (High Speed)**: Speed greater than 50 cm per clock cycle.
* **Green (Low Speed)**: Speed less than 20 cm per clock cycle.
* **Blue (Medium Speed)**: Speed in between.

verilog

Copy code

always @(posedge clk) begin

if (speed\_reg > 50) // High speed: Red

rgb <= 3'b100;

else if (speed\_reg < 20) // Low speed: Green

rgb <= 3'b010;

else // Medium speed: Blue

rgb <= 3'b001;

end

**5. Seven-Segment Display Decoder**

The abcdefg output drives the segments of the seven-segment display to represent digits from 0 to 9. A case statement determines the segment patterns for each digit.

always @(\*) begin

case (digit)

4'b0000: abcdefg = 7'b1000000; // 0

4'b0001: abcdefg = 7'b1111001; // 1

4'b0010: abcdefg = 7'b0100100; // 2

4'b0011: abcdefg = 7'b0110000; // 3

4'b0100: abcdefg = 7'b0011001; // 4

4'b0101: abcdefg = 7'b0010010; // 5

4'b0110: abcdefg = 7'b0000010; // 6

4'b0111: abcdefg = 7'b1111000; // 7

4'b1000: abcdefg = 7'b0000000; // 8

4'b1001: abcdefg = 7'b0010000; // 9

default: abcdefg = 7'b1111111; // Blank

endcase

end

**6. Multiplexing Seven-Segment Displays**

To display multiple digits of speed, the system uses a time-multiplexing approach. The active\_digit selects which digit is displayed, and an enables the corresponding display.

always @(posedge clk) begin

refresh\_counter <= refresh\_counter + 1;

if (refresh\_counter == 16'd5000) begin // Adjust refresh rate for multiplexing

refresh\_counter <= 0;

active\_digit <= active\_digit + 1;

if (active\_digit == 7) // Reset to 0 after 7 digits

active\_digit <= 0;

case (active\_digit)

0: begin

digit <= (speed\_reg / 100000) % 10; // Display the highest place

an <= 8'b11111110; // Enable display 0

end

1: begin

digit <= (speed\_reg / 10000) % 10; // Display next place

an <= 8'b11111101; // Enable display 1

end

2: begin

digit <= (speed\_reg / 1000) % 10; // Display next place

an <= 8'b11111011; // Enable display 2

end

3: begin

digit <= (speed\_reg / 100) % 10; // Display next place

an <= 8'b11110111; // Enable display 3

end

4: begin

digit <= (speed\_reg / 10) % 10; // Display next place

an <= 8'b11101111; // Enable display 4

end

5: begin

digit <= speed\_reg % 10; // Display least significant digit

an <= 8'b11011111; // Enable display 5

end

default: begin

digit <= 4'b0000;

an <= 8'b11111111; // Disable all displays

end

endcase

end

always @(posedge clk) begin

refresh\_counter <= refresh\_counter + 1;

if (refresh\_counter == 16'd5000) begin // Adjust refresh rate for multiplexing

refresh\_counter <= 0;

active\_digit <= active\_digit + 1;

if (active\_digit == 7) // Reset to 0 after 7 digits

active\_digit <= 0;

case (active\_digit)

0: begin

digit <= (speed\_reg / 100000) % 10; // Display the highest place

an <= 8'b11111110; // Enable display 0

end

1: begin

digit <= (speed\_reg / 10000) % 10; // Display next place

an <= 8'b11111101; // Enable display 1

end

2: begin

digit <= (speed\_reg / 1000) % 10; // Display next place

an <= 8'b11111011; // Enable display 2

end

3: begin

digit <= (speed\_reg / 100) % 10; // Display next place

an <= 8'b11110111; // Enable display 3

end

4: begin

digit <= (speed\_reg / 10) % 10; // Display next place

an <= 8'b11101111; // Enable display 4

end

5: begin

digit <= speed\_reg % 10; // Display least significant digit

an <= 8'b11011111; // Enable display 5

end

default: begin

digit <= 4'b0000;

an <= 8'b11111111; // Disable all displays

end

endcase

end

end

**summary**

The Verilog code implements a robust car speed detection system by combining:

* IR sensor-based time measurement for speed calculation.
* Seven-segment displays for precise visual output of speed.
* RGB LEDs for real-time speed categorization.