WELCOME TO OUR PRESENTATION

Cross Traffic Management

Course Tittle: Electronics Engineering A

Team: 5

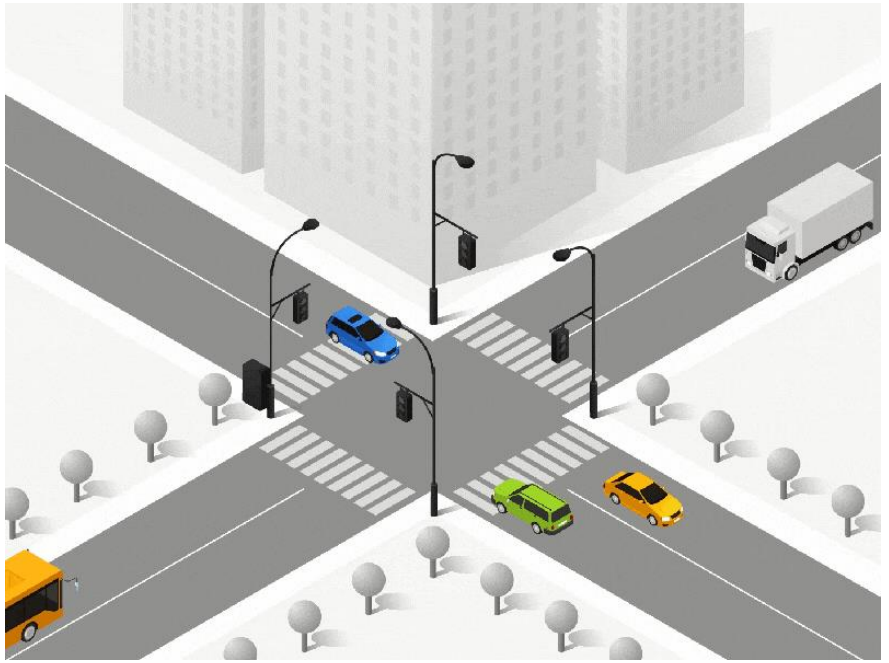
Members:

Md Akram

Abdullah al Forkan

Ali Hisham

Dapsara Kapuge

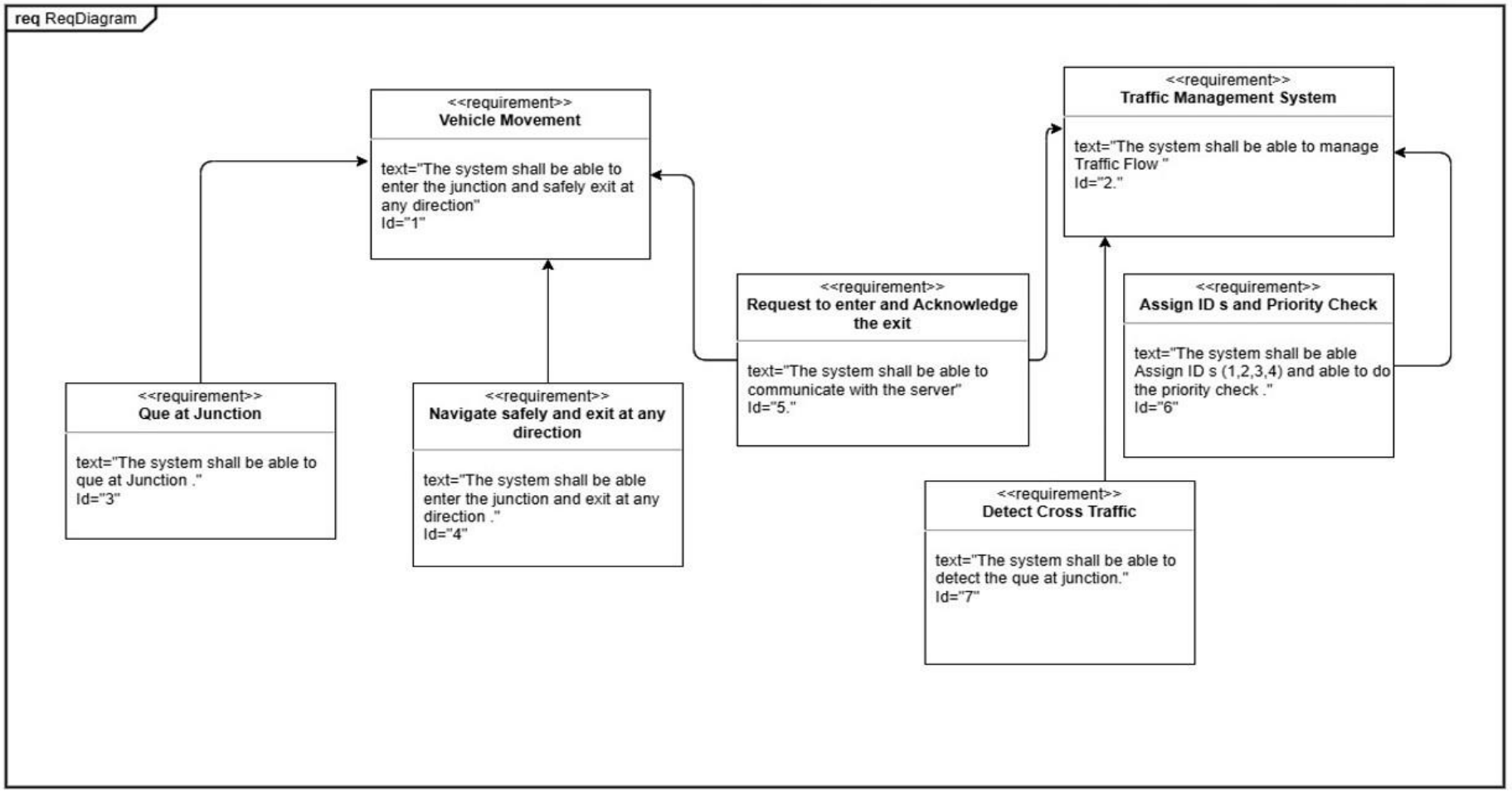


Motivation

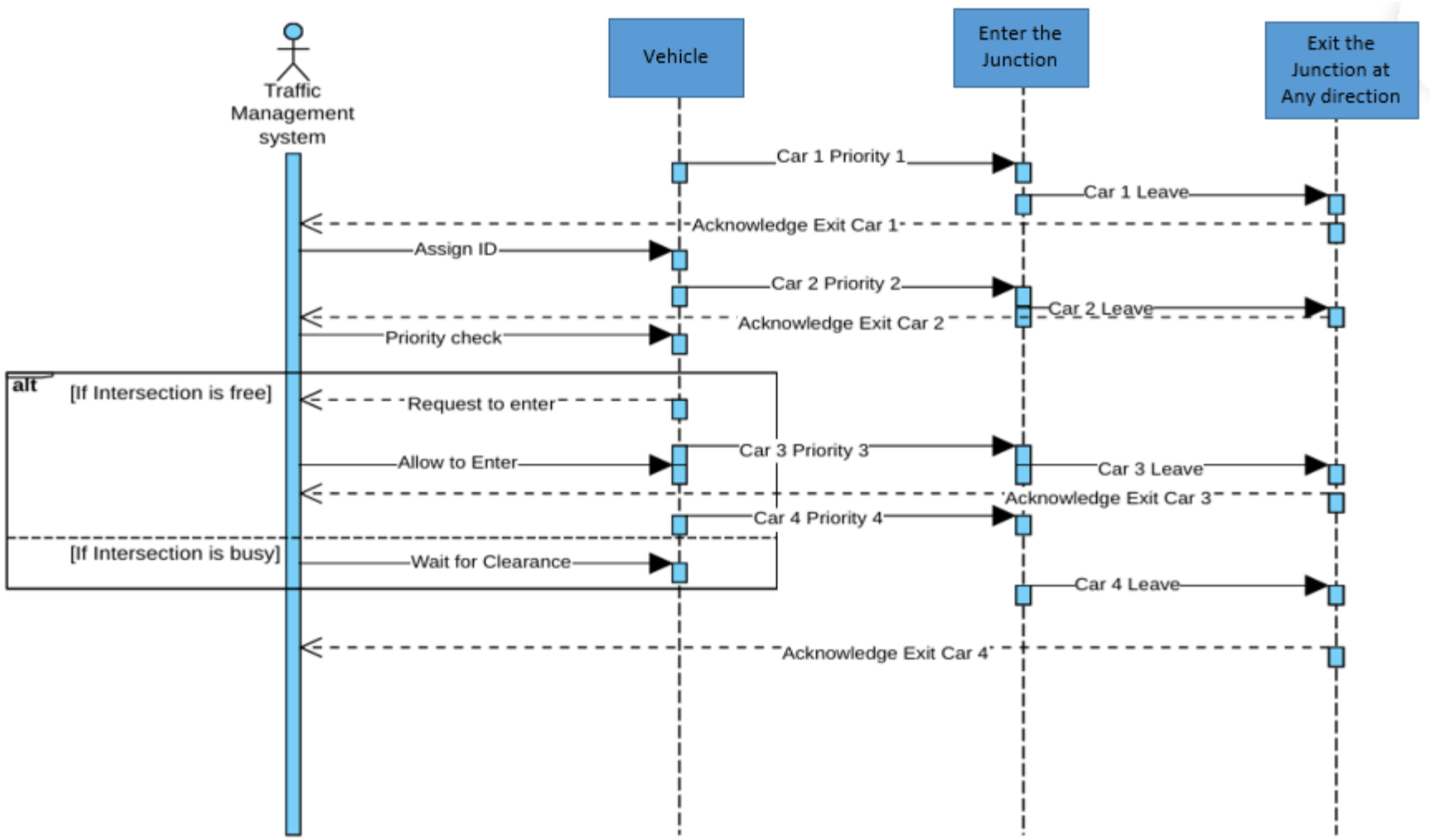
- Ensure Safe and secure Journey
- Reduce Road Accident
- Savings Travel Time

UML Implementation

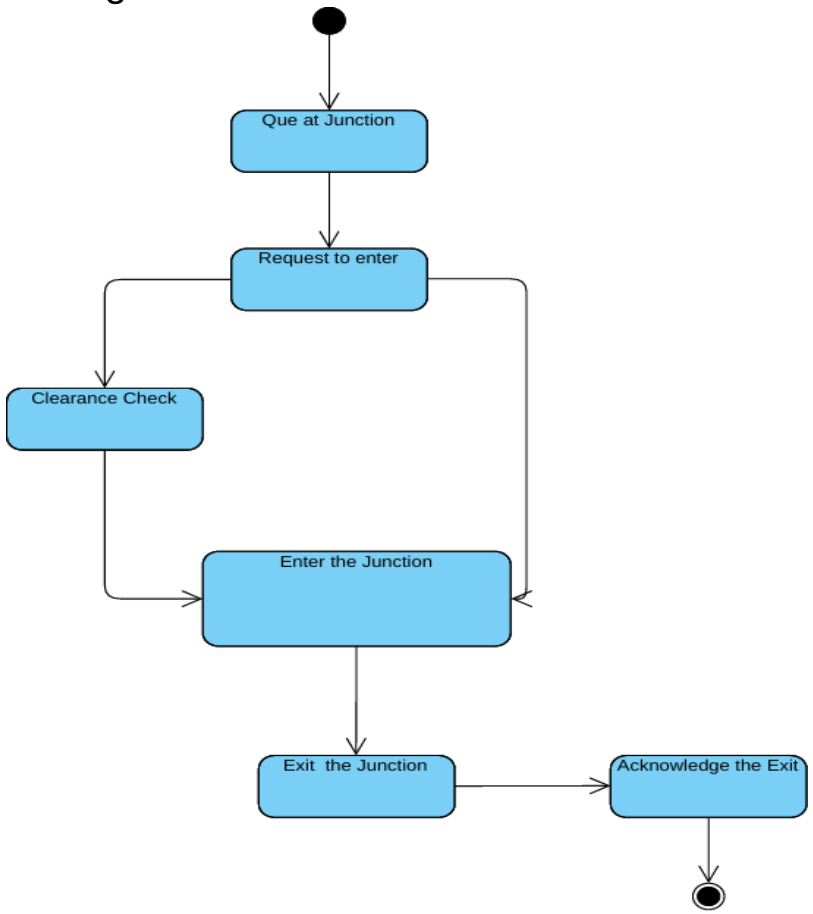
Requirement Diagram



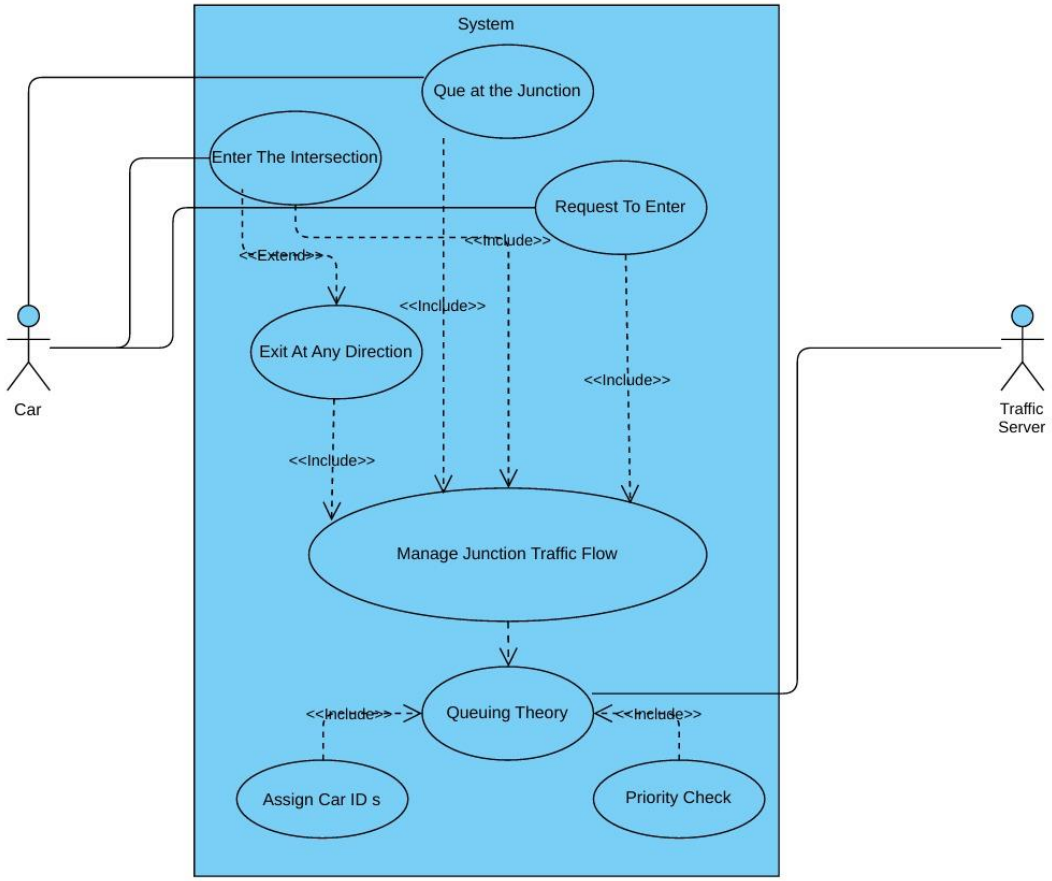
Sequence Diagram

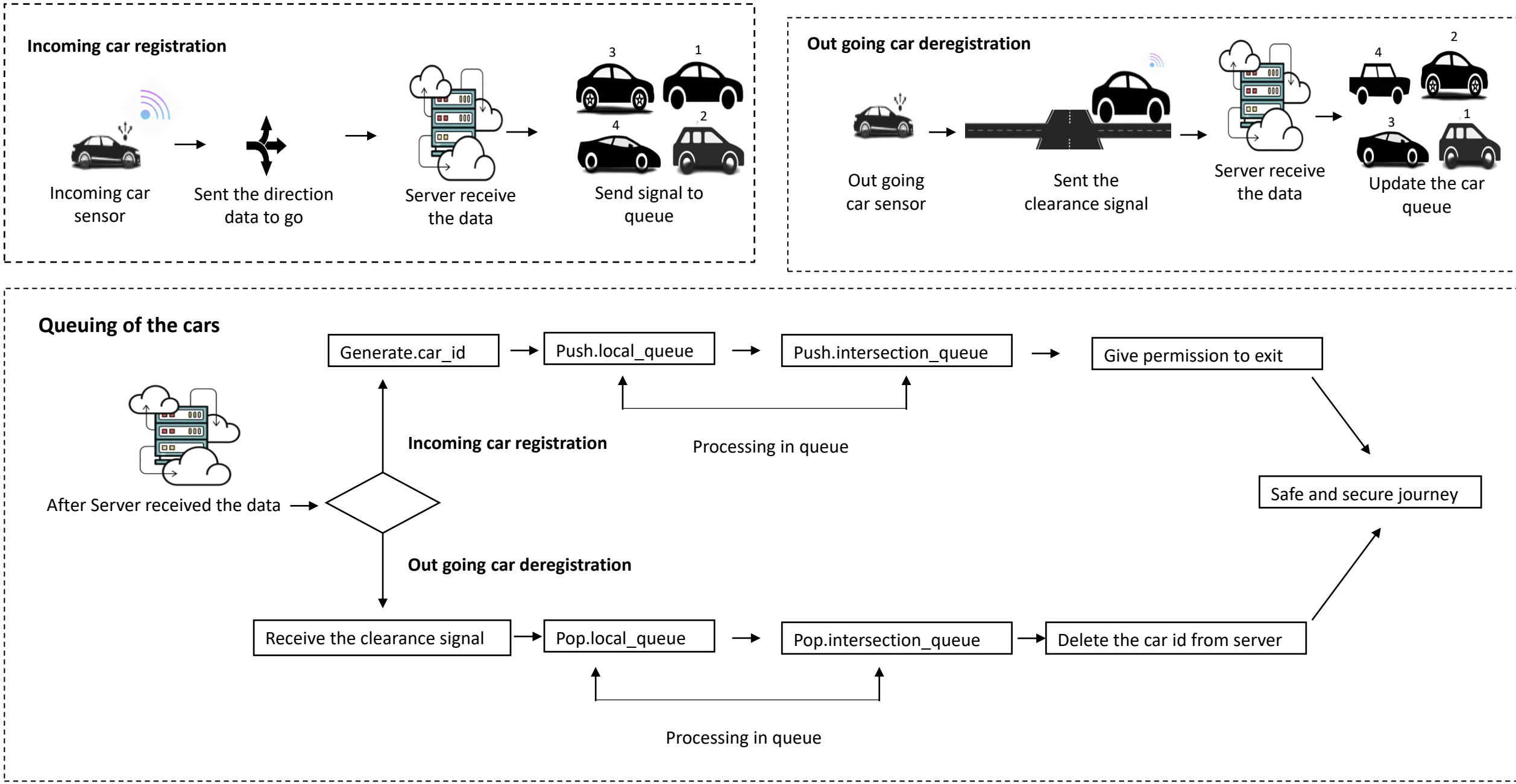


State Machine Diagram



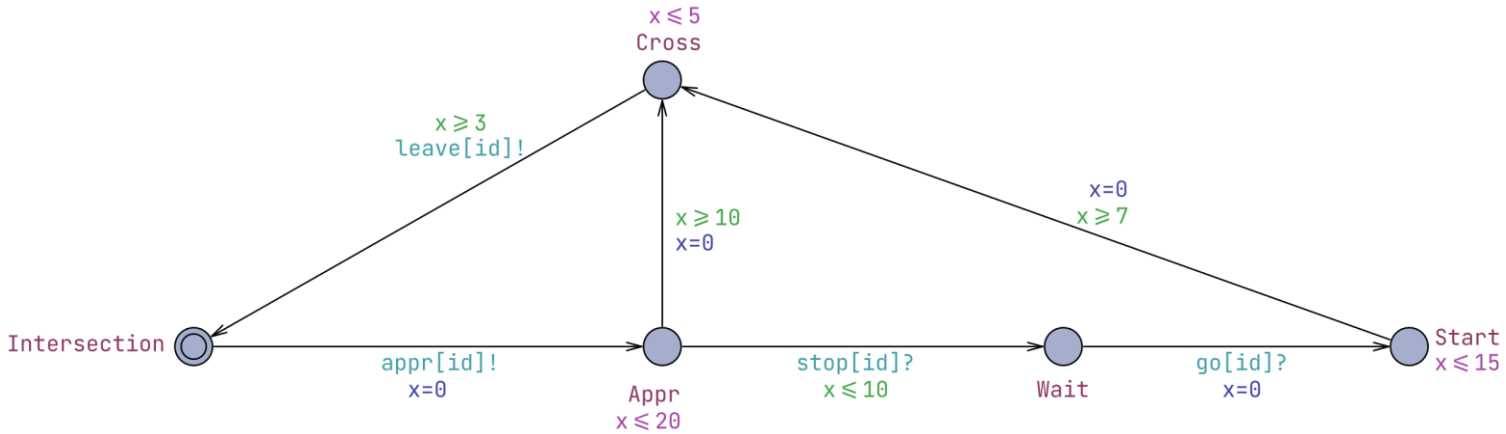
User Case Diagram



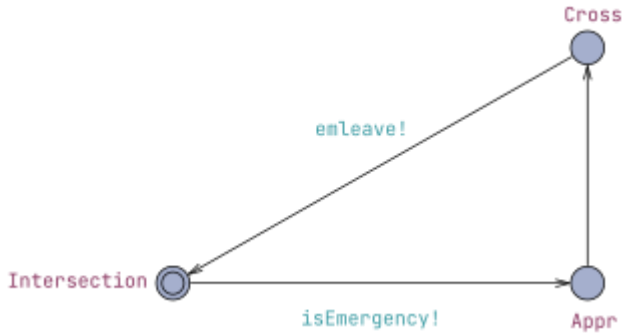


Activity diagram to queue cars

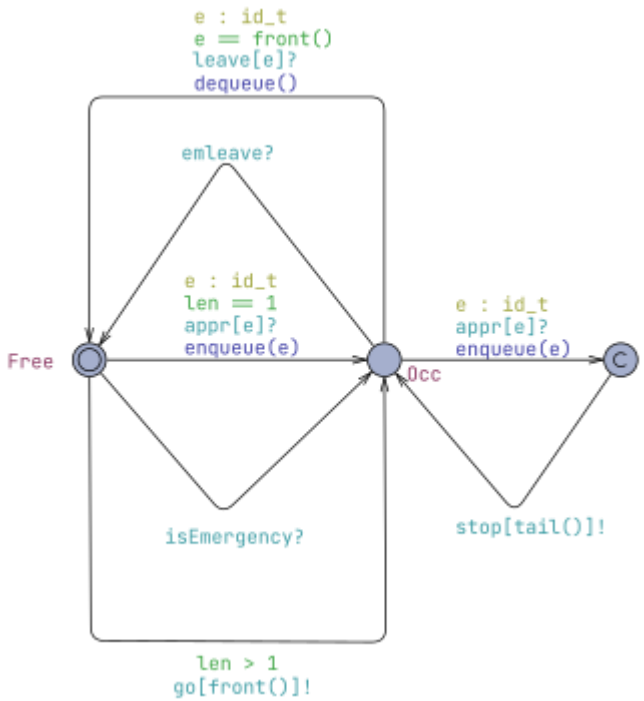
Vehicle



Emergency Vehicle



Server Control

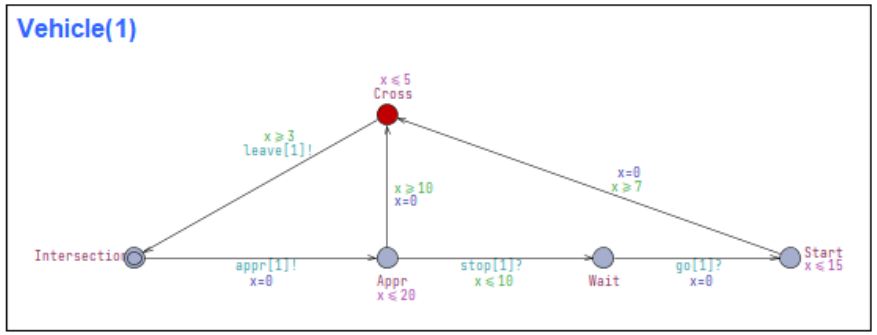


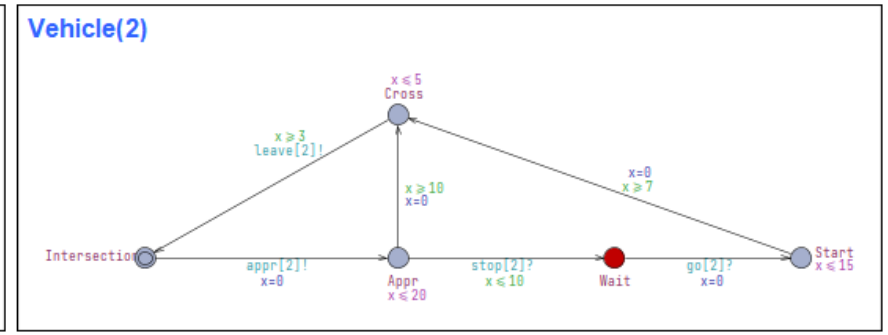
Enqueue & Dequeue

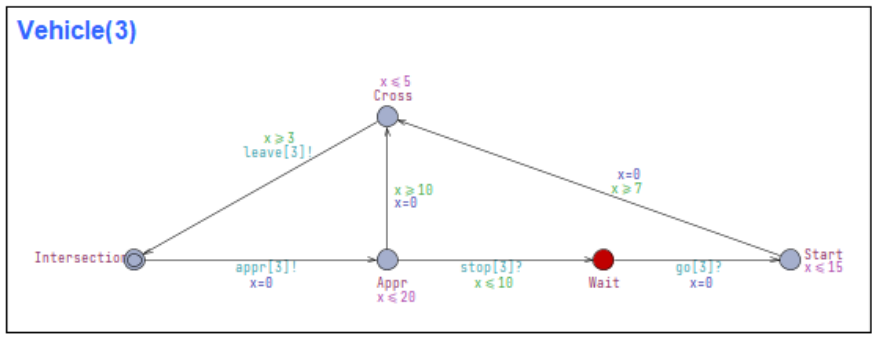
Name: Controll Parameters:

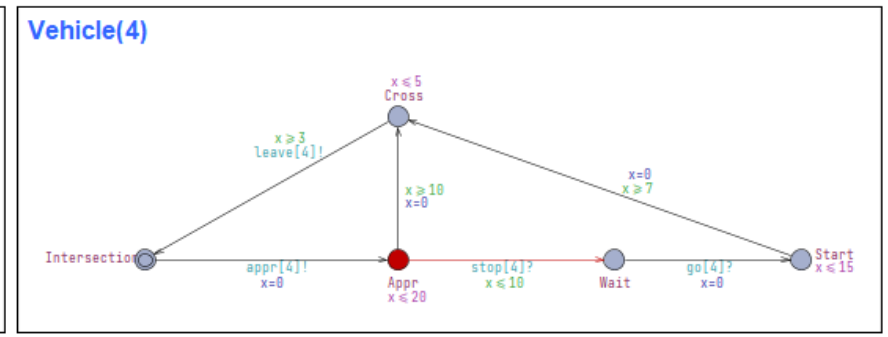
```
// Put an element at the end of the queue
void enqueue(id_t element)
{
        list[len++] = element;
}

// Remove the front element of the queue
void dequeue()
{
        int i = 1;
        len -= 1;
        while (i < len)
        {
                list[i] = list[i + 1];
                i++;
        }
        list[i] = 1;
}
```

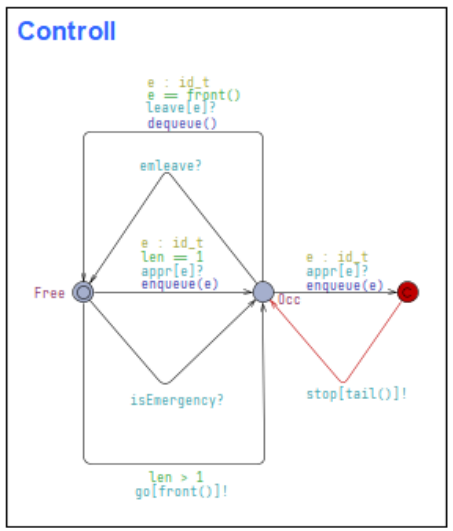
UPPAAL Simulation

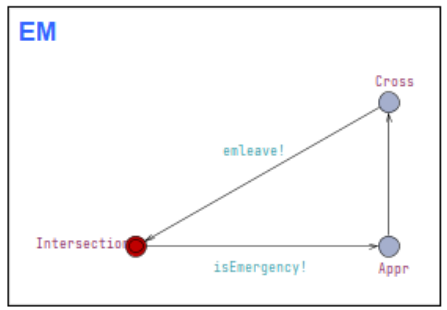


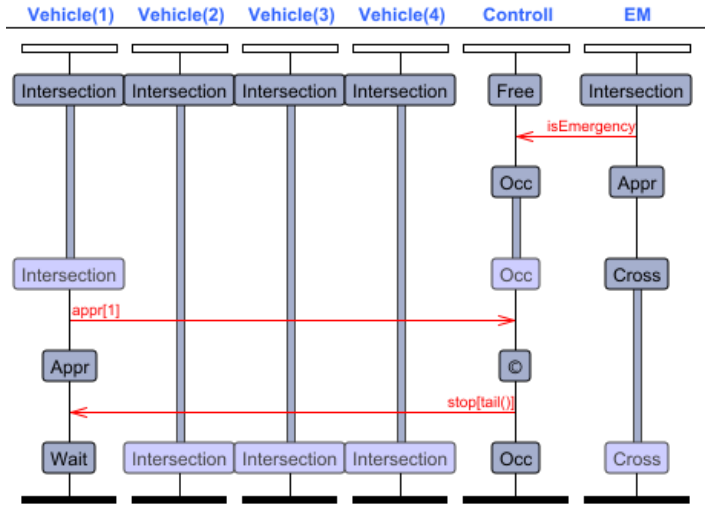












UPPAAL Verify

Overview

```
===== Validation Properties:
E<> Control.Occ
E<> Vehicle(1).Cross
E<> Vehicle(2).Cross
E<> Vehicle(1).Cross and Vehicle(2).Wait
E<> Vehicle(1).Cross and (forall (i : id_t) i ≠ 1 imply Vehicle(i).Wait)
===== Safety Properties:
A[] forall (i : id_t) forall (j : id_t) Vehicle(i).Cross && Vehicle(j).Cross imply i == j
A[] Control.list[N] == 1
===== Liveness Properties:
Vehicle(1).Appr ⟶ Vehicle(1).Cross
Vehicle(2).Appr ⟶ Vehicle(2).Cross
Vehicle(3).Appr ⟶ Vehicle(3).Cross
Vehicle(4).Appr ⟶ Vehicle(4).Cross
===== Deadlock checking:
A[] not deadlock
```

Simulator

CARLA

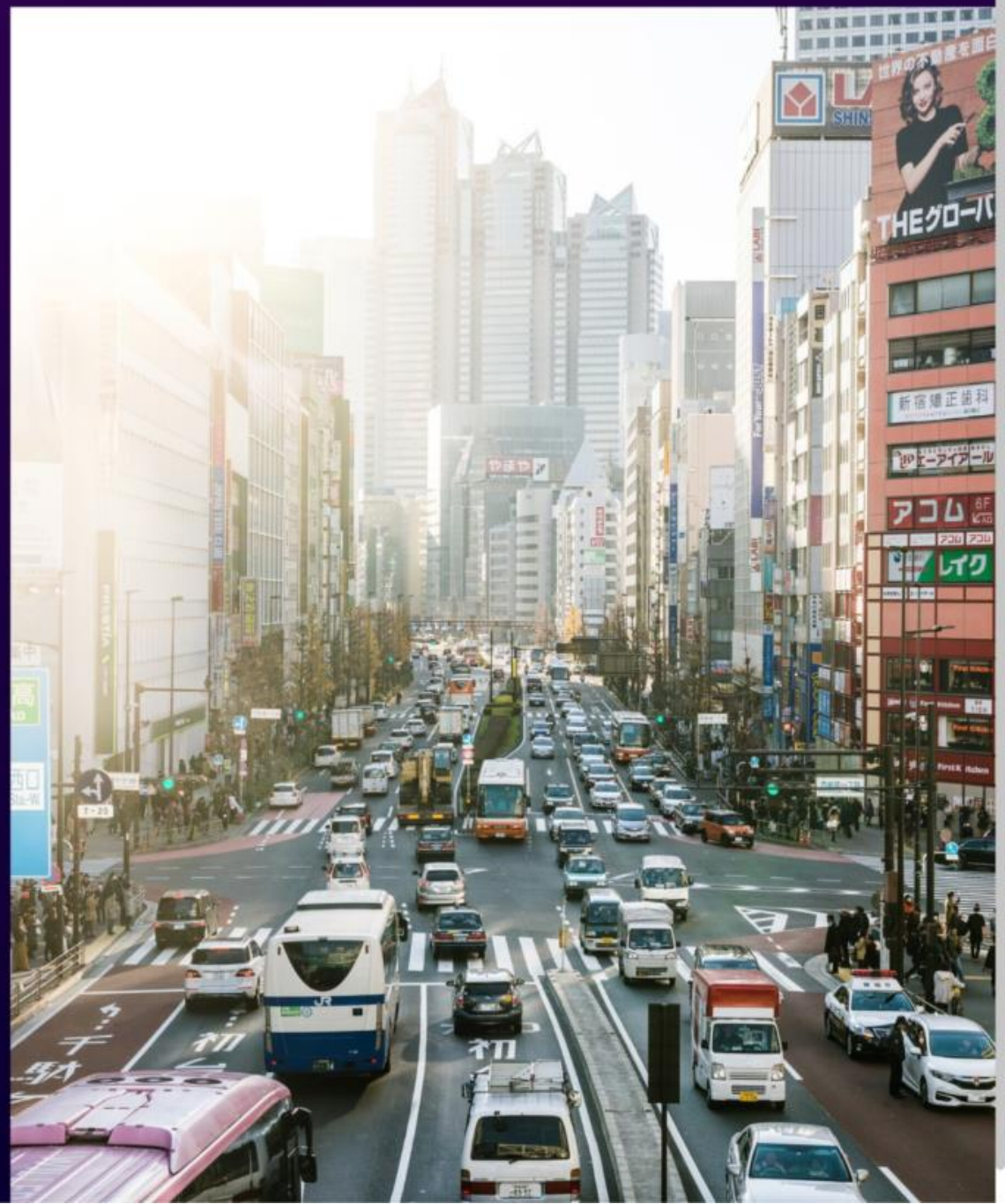
python
python
python
python

User scripts

Traffic Light Simulation Project

Efficient Task Management with FreeRTOS

Ali Hisham



EFFICIENT TASK MANAGEMENT

Agenda

Exploring Task Management with FreeRTOS in Traffic Light Simulation Project

#1 State Machine Analysis

Analyze the state machine design to understand the system's behavior and transitions.

#2 FreeRTOS Setup

Learn how to set up FreeRTOS, an open-source real-time operating system, for task scheduling.

#3 Project Structure and Configuration

Examine the structure of the project and its configuration requirements for efficient task management.

#4 Configuring FreeRTOS

Understand the process of configuring FreeRTOS to optimize task scheduling and resource utilization.

#5 Implementing main.c

Explore the implementation of the main.c file in the Traffic Light Simulation Project for task execution.

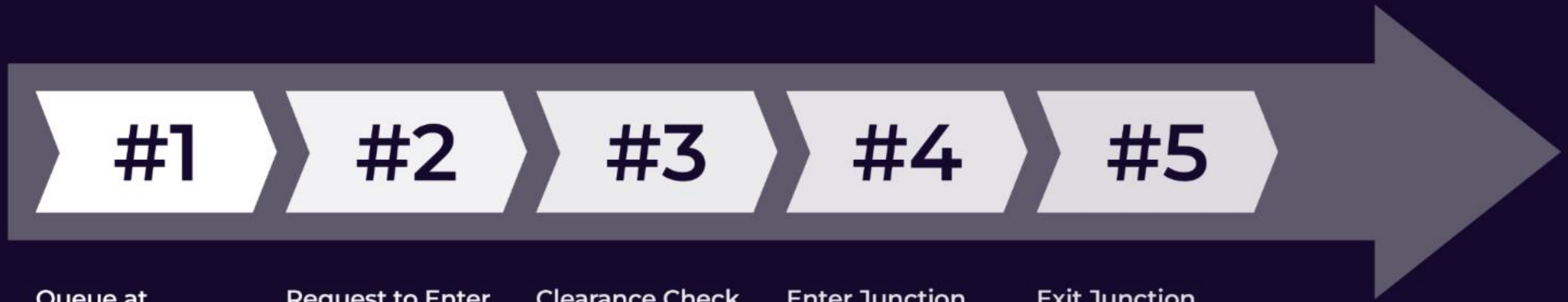
#6 Creating and Configuring Tasks

Learn how to create and configure tasks in FreeRTOS to manage concurrent processes effectively.

TRAFFIC FLOW STATES

State Machine Diagram Steps Explained

Flow of Traffic at an Intersection



Queue at Junction

Cars wait at the junction.

Request to Enter

Cars request permission to enter.

Clearance Check

Check if the light is green.

Enter Junction

Cars proceed when light is green.

Exit Junction

Cars exit the junction.

TASK MANAGEMENT EFFICIENCY

FreeRTOS Setup

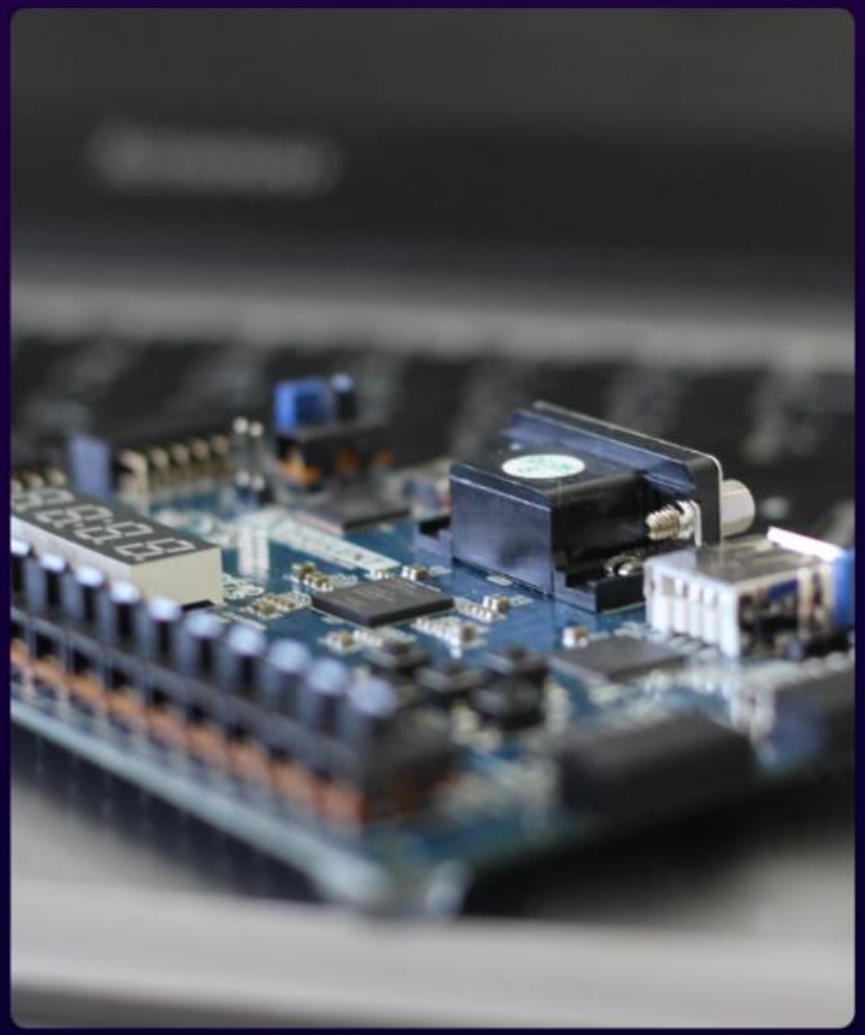
Enhancing Task Management Efficiency in Traffic Light Simulation Project

POSIX Adaptation

Utilized POSIX adaptation to eliminate hardware requirements, providing a more flexible and scalable solution for the project.

Compatibility with Standard OS

FreeRTOS runs seamlessly on standard operating systems, enhancing interoperability and ensuring smooth integration within existing software environments.



Multitasking Capabilities

FreeRTOS was selected for its robust multitasking features, enabling efficient task management in the traffic light simulation project.

Testing and Development Simplification

FreeRTOS simplifies testing and development processes by offering a standardized operating system framework, streamlining project workflows.

SETUP OVERVIEW

Project Structure

Establishing Essential Project Directories and File Setup

#1 **Created project directory**

Established the main project directory to organize project files efficiently.

#2 **Included essential FreeRTOS files**

Added necessary header files and .c source files for FreeRTOS implementation.

#3 **Created a suitable CMakeLists.txt**

Configured the setup to automate the build process, streamlining development tasks and ensuring efficient project compilation.

CONFIGURATION SETTINGS

Configuring FreeRTOS

Optimizing Task Management in FreeRTOSConfig.h

Stack Size

Configured to provide sufficient memory allocation for tasks.

Heap Size

Managed to efficiently allocate memory for task execution.

Task Priorities

Set to manage and prioritize tasks based on importance levels.

Tick Rate

Adjusted for precise timing synchronization among tasks.

Preemption

Enabled high-priority tasks to interrupt low-priority ones for efficient task handling.

Mutexes/Sem

Configured to ensure proper resource access and synchronization of tasks.

Error Handling

Configured stack overflow and Memory allocation failed hooks for effective error management.

TASK MANAGEMENT

Implementing main.c

Creation of Primary Tasks in main.c

#1

Traffic Light Control Task

Manages light states with specific delays to regulate traffic flow efficiently.

#2

Car Generation Task

Generates cars, assigns directions, and queues them for smooth traffic simulation.

#3

Car Passing Task

Processes cars when the traffic light is green, managing their exits strategically.

#4

Independent Task Operations

Each task operates autonomously, facilitated by FreeRTOS for seamless task management.

```c
// This task is used to process cars that are passing through the intersection
void vCarPassingTask(void *pvParameters) {
    Car car; // local variable to hold the car data received from the queue
    for (;;) { // Infinite loop to continuously process cars
        if (trafficLightState == GREEN) { // Check if the traffic light is green
            // xQueueReceive is a FreeRTOS function that receives data from a queue
            // carQueue is the queue we are receiving the data from
            // &car is the address of the car struct we are receiving
            // pdMS_TO_TICKS(100) is the maximum amount of time the function should wait for data to be available in the queue
            // pdTRUE is a macro that indicates that the data was successfully received from the queue
            if (xQueueReceive(carQueue, &car, pdMS_TO_TICKS(100)) == pdTRUE) {
                printf("Car ID: %d is passing through the intersection from %s and turning %s\n",
                       car.id,
                       "East",
                       car.turn == STRAIGHT ? "straight" :
                       car.turn == RIGHT ? "right" :
                       "left");
            }
        }
        vTaskDelay(pdMS_TO_TICKS(100)); // Delay for 100 milliseconds
    }
}
```

```
Traffic Light: YELLOW
Car ID: 2, Approaching from: EAST, Wants to turn: Right
Traffic Light: RED
Traffic Light: GREEN
Car ID: 2 is passing through the intersection from East and turning right
Car ID: 3, Approaching from: EAST, Wants to turn: Left
Car ID: 3 is passing through the intersection from East and turning left
Traffic Light: YELLOW
Traffic Light: RED
Car ID: 4, Approaching from: EAST, Wants to turn: Left
Traffic Light: GREEN
Car ID: 4 is passing through the intersection from East and turning left
Car ID: 5, Approaching from: EAST, Wants to turn: Right
Car ID: 5 is passing through the intersection from East and turning right
Traffic Light: YELLOW
Traffic Light: RED
```

TASK MANAGEMENT

Task Creation and Configuration

Efficient Task Management in Traffic Light Simulation Project

Traffic Light Task

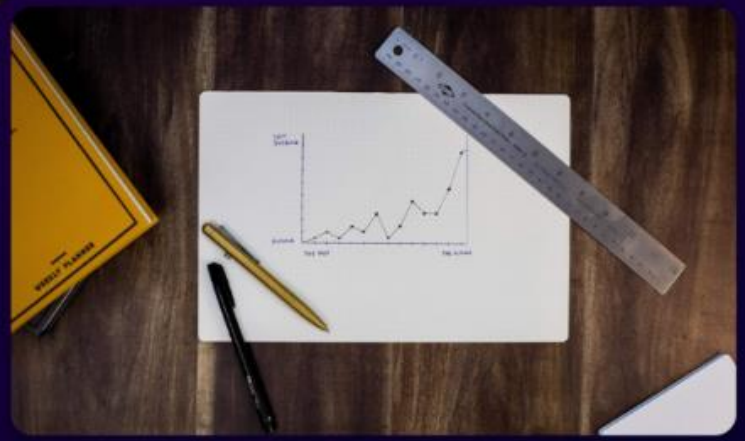
Created with a large stack size and higher priority than idle.

Car Generator & Car Passing Tasks

Configured with large stacks and the same priority level.

Project Development Summary

Analyzed state machine diagram, set up FreeRTOS, implemented main logic, created and configured tasks, debugged stack size issues, and demonstrated efficient task management.



SOFTWARE DEVELOPMENT

Summary

Efficient Task Management with FreeRTOS in Traffic Light Simulation Project

Analyzed State Machine Diagram

Reviewed and analyzed the state machine diagram for the traffic light simulation project.

FreeRTOS Setup and Configuration

Successfully set up and configured FreeRTOS for task management in the project.

Main Logic Implementation

Implemented the main logic of the traffic light simulation project in main.c file.

Task Creation and Configuration

Created and configured tasks within FreeRTOS to manage project functionalities effectively.

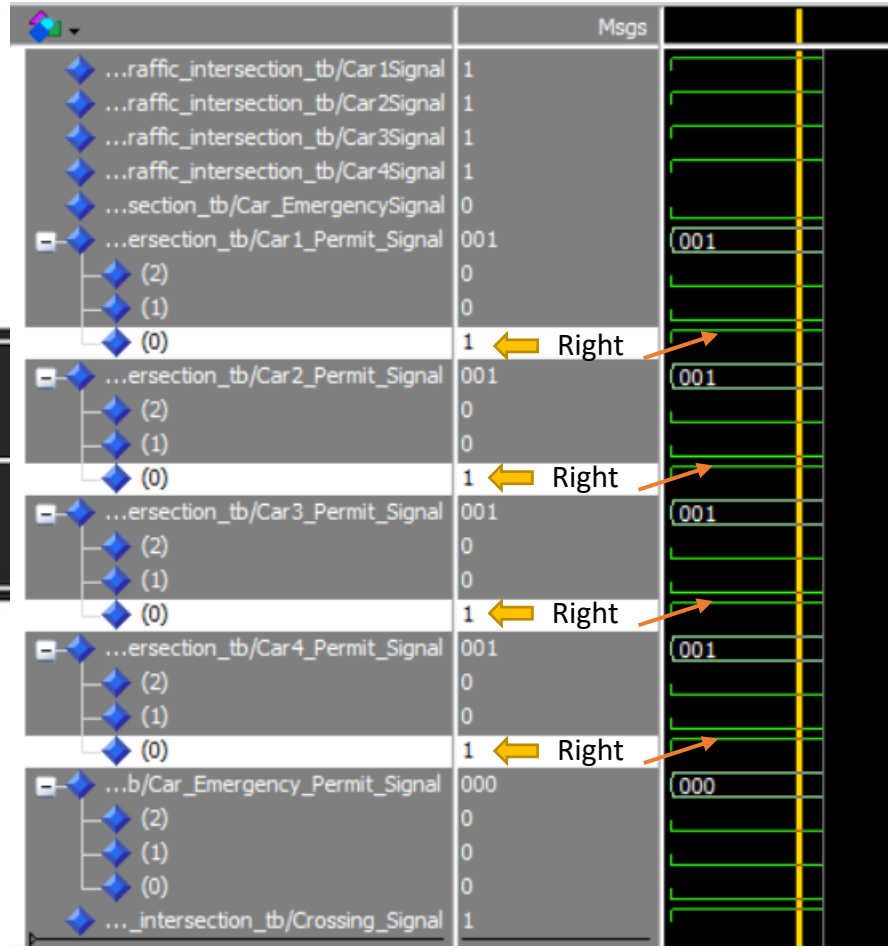
VHDL Implementation

Very High Speed Integrated Circuit Hardware Description Language.

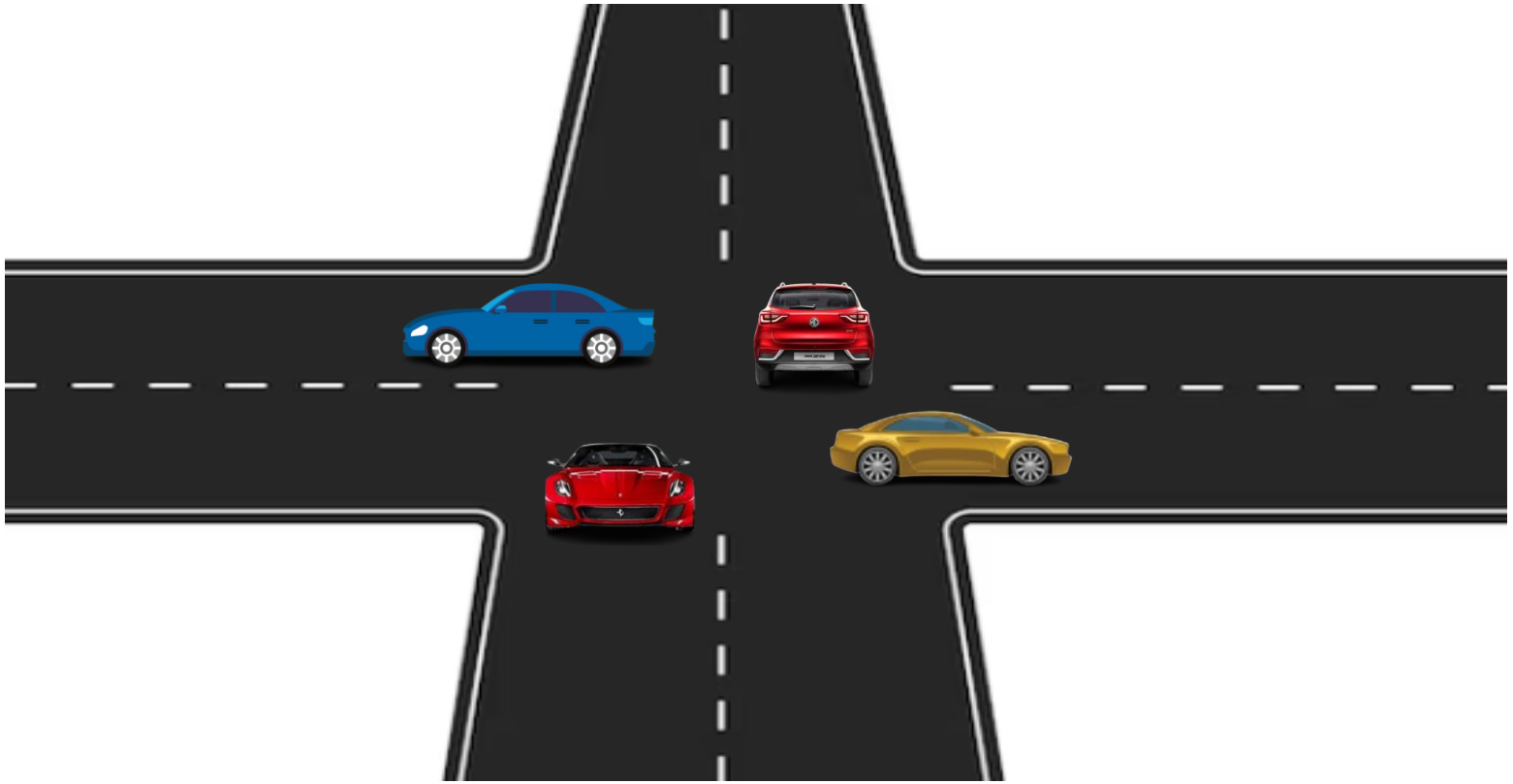
VHDL has many advantages such as:

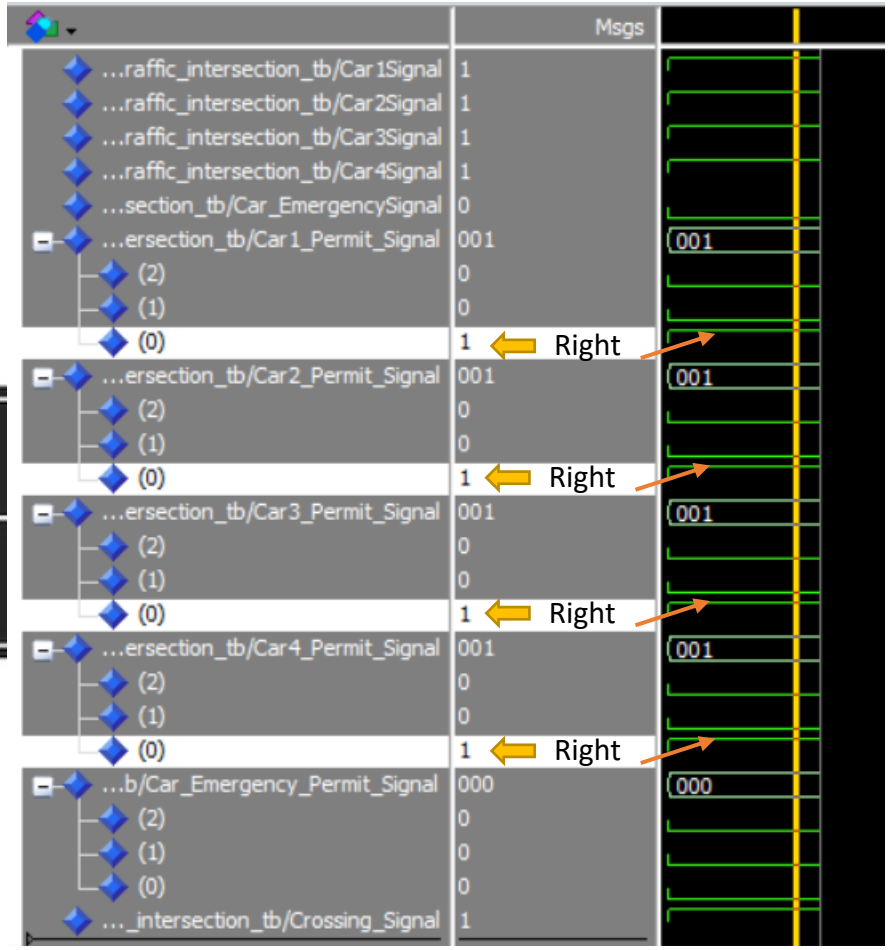
- Portability
- Reusability
- Simulation and Verification
- Design Documentation

Right direction:

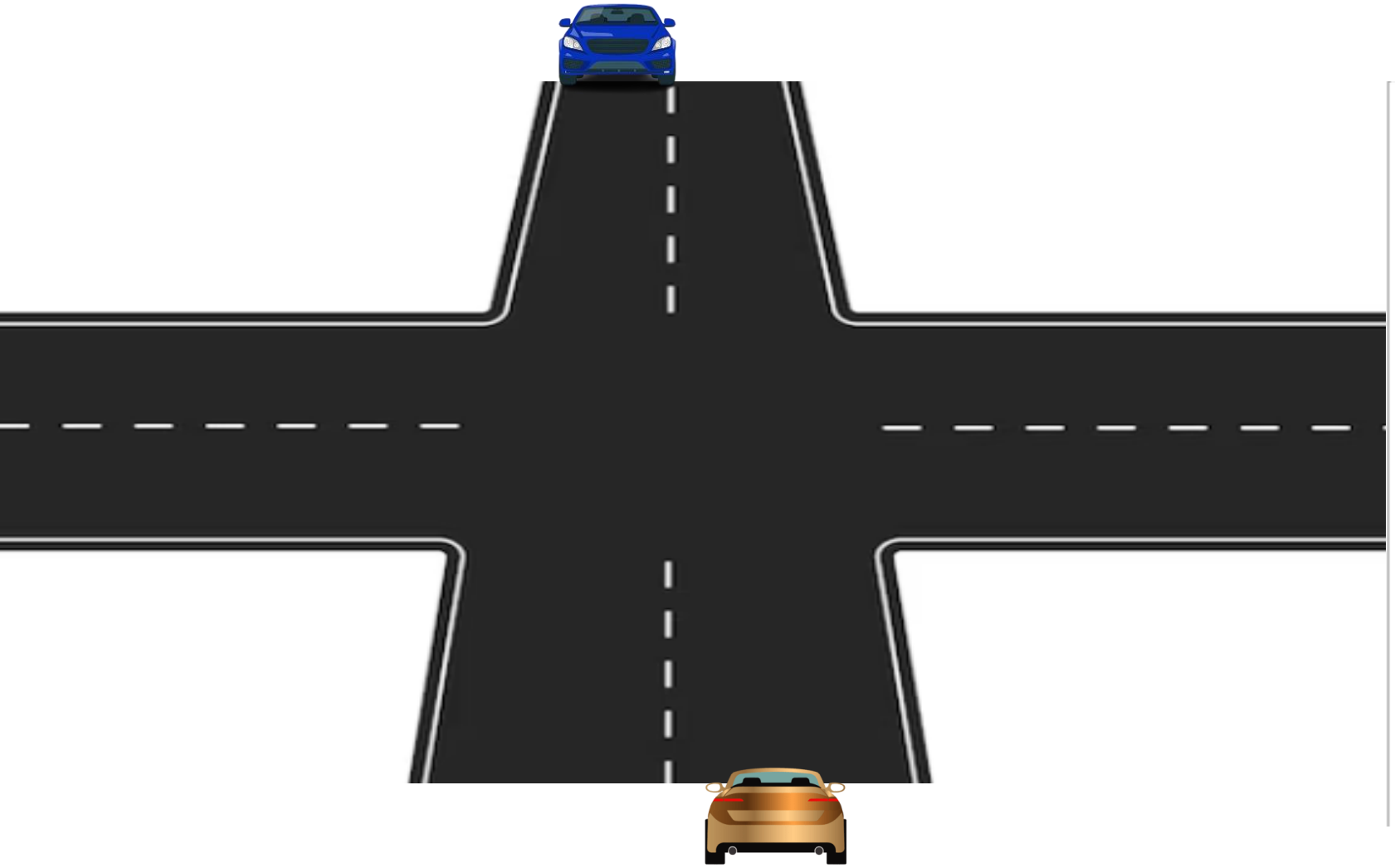


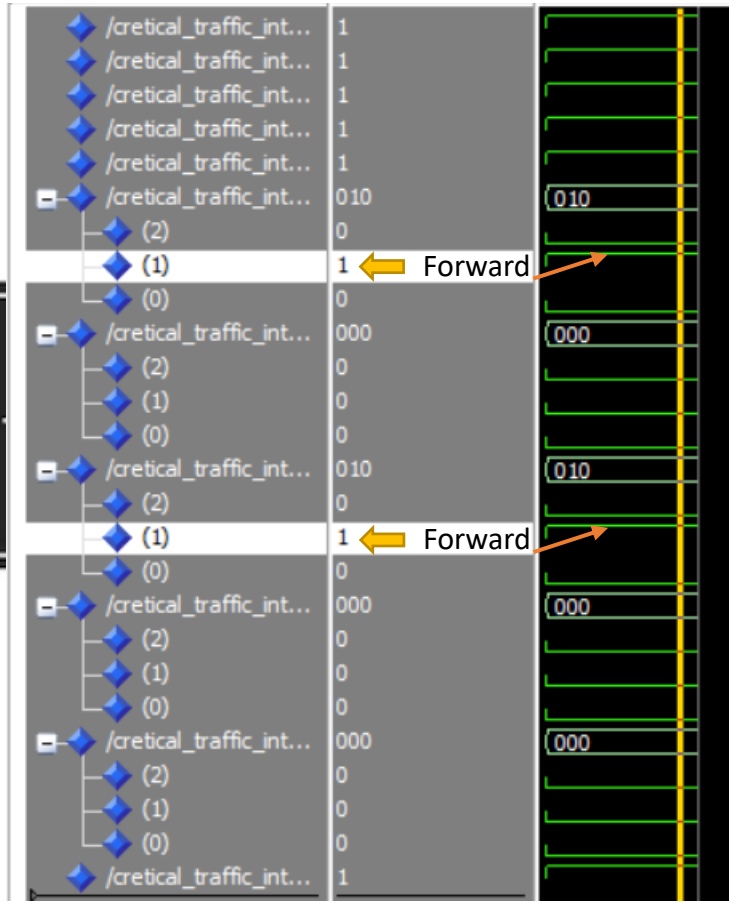
Right direction: All Car are allow to cross the intersection area without any problem



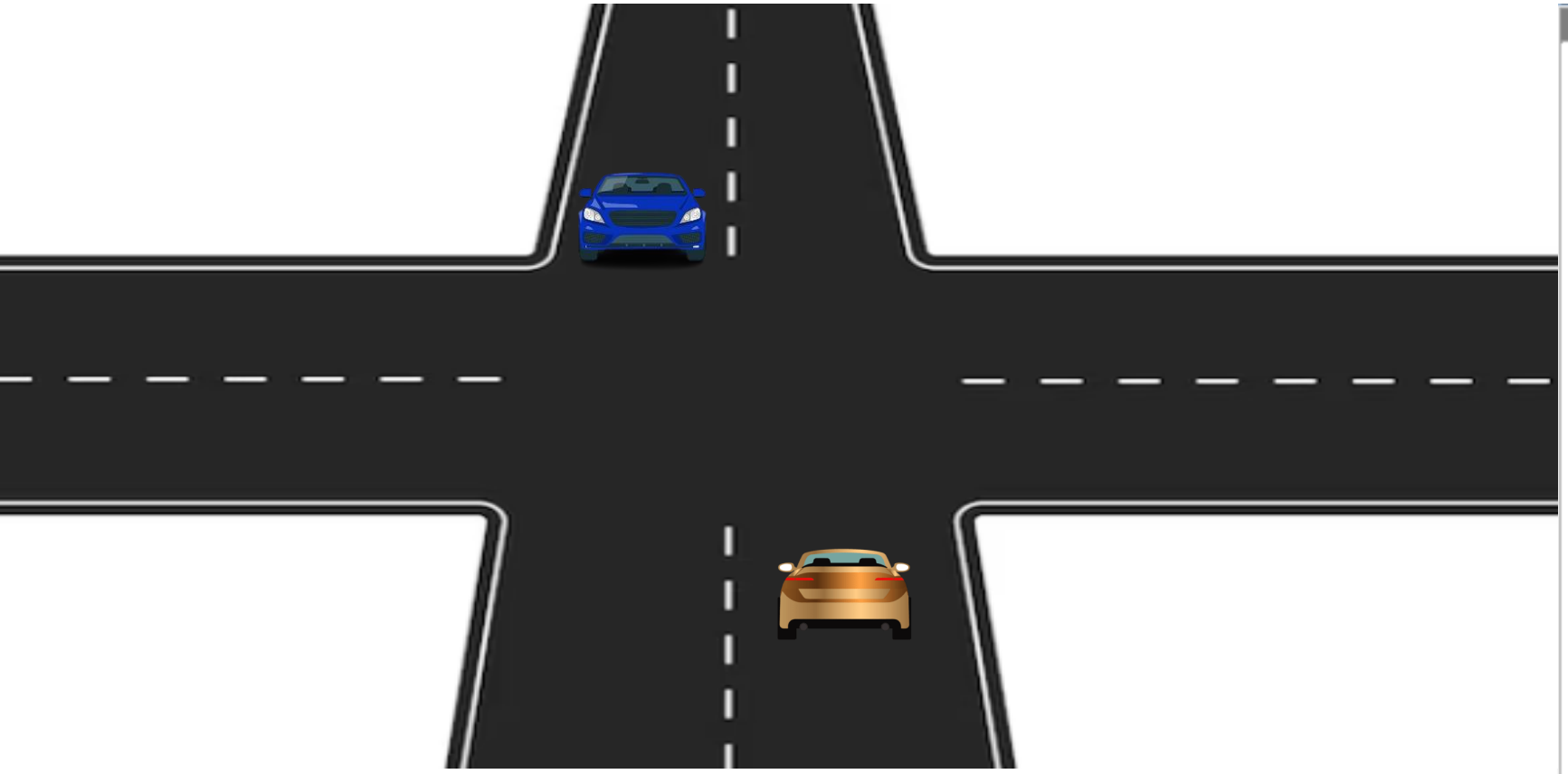


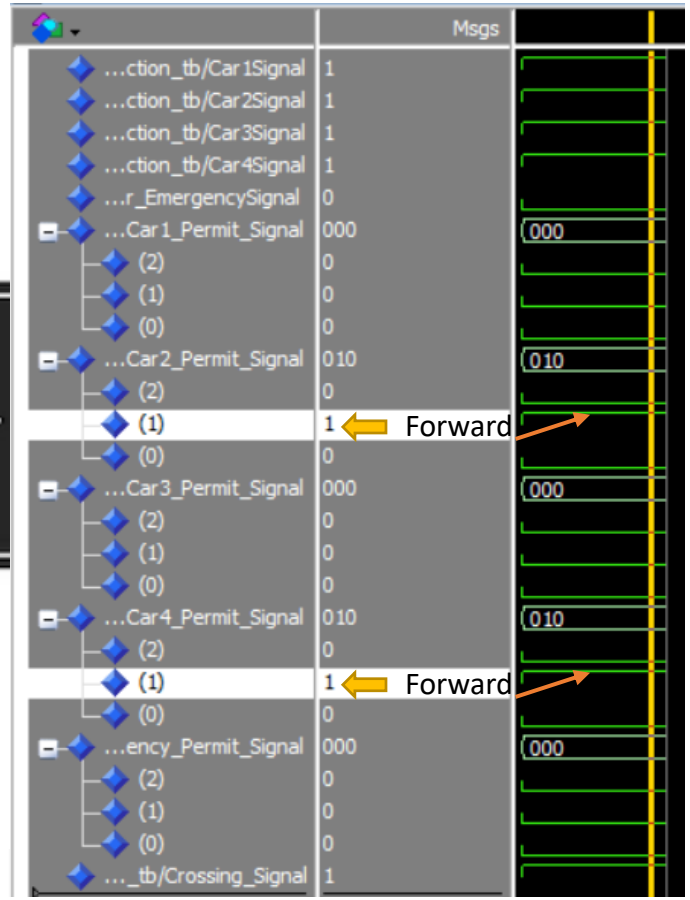
Forward direction: Only opposite side cars allow to cross the intersection area without any problem
So car 1 and car 3 are crossing



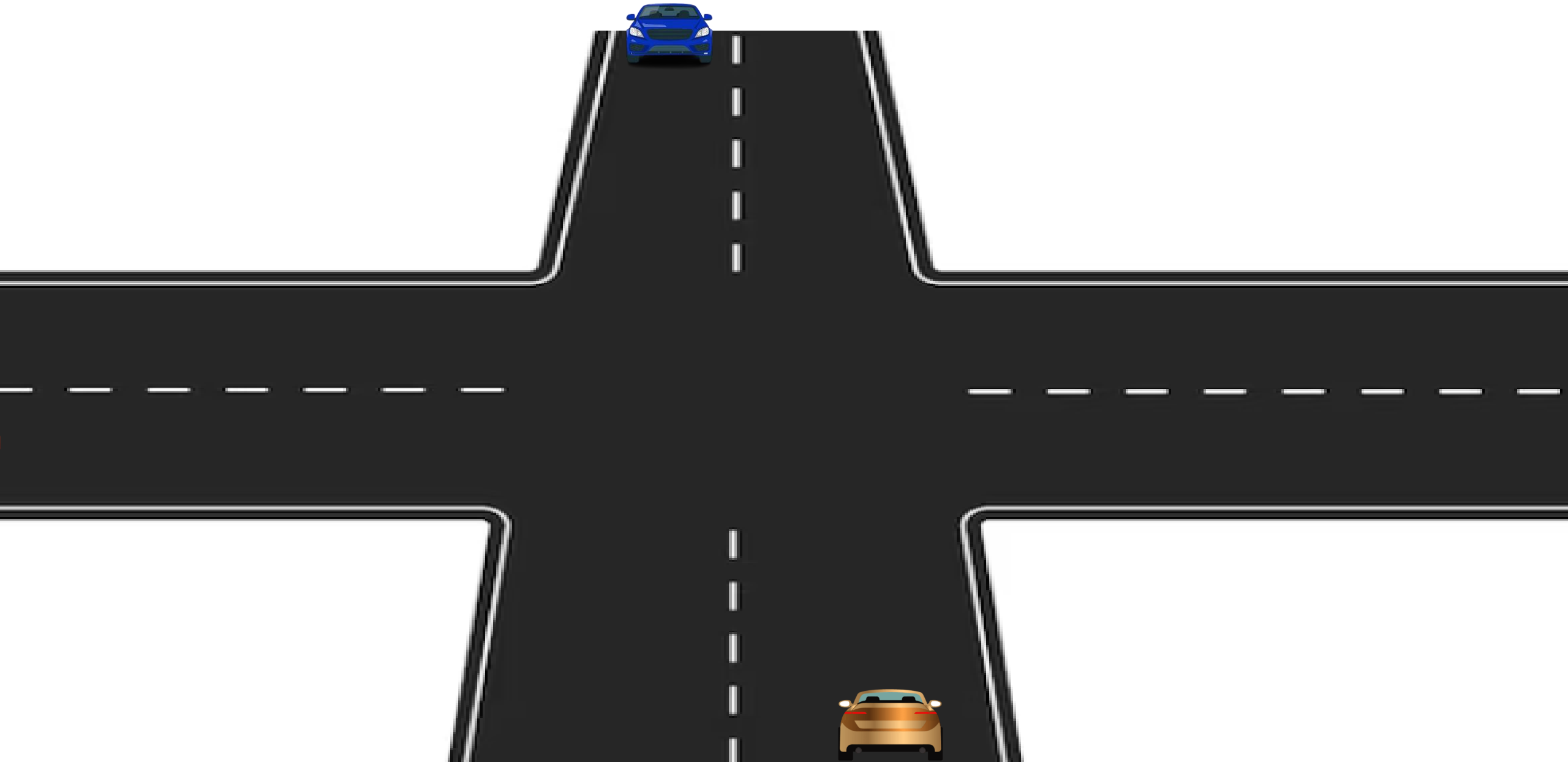


Forward direction: Car 2 and car 4 are crossing

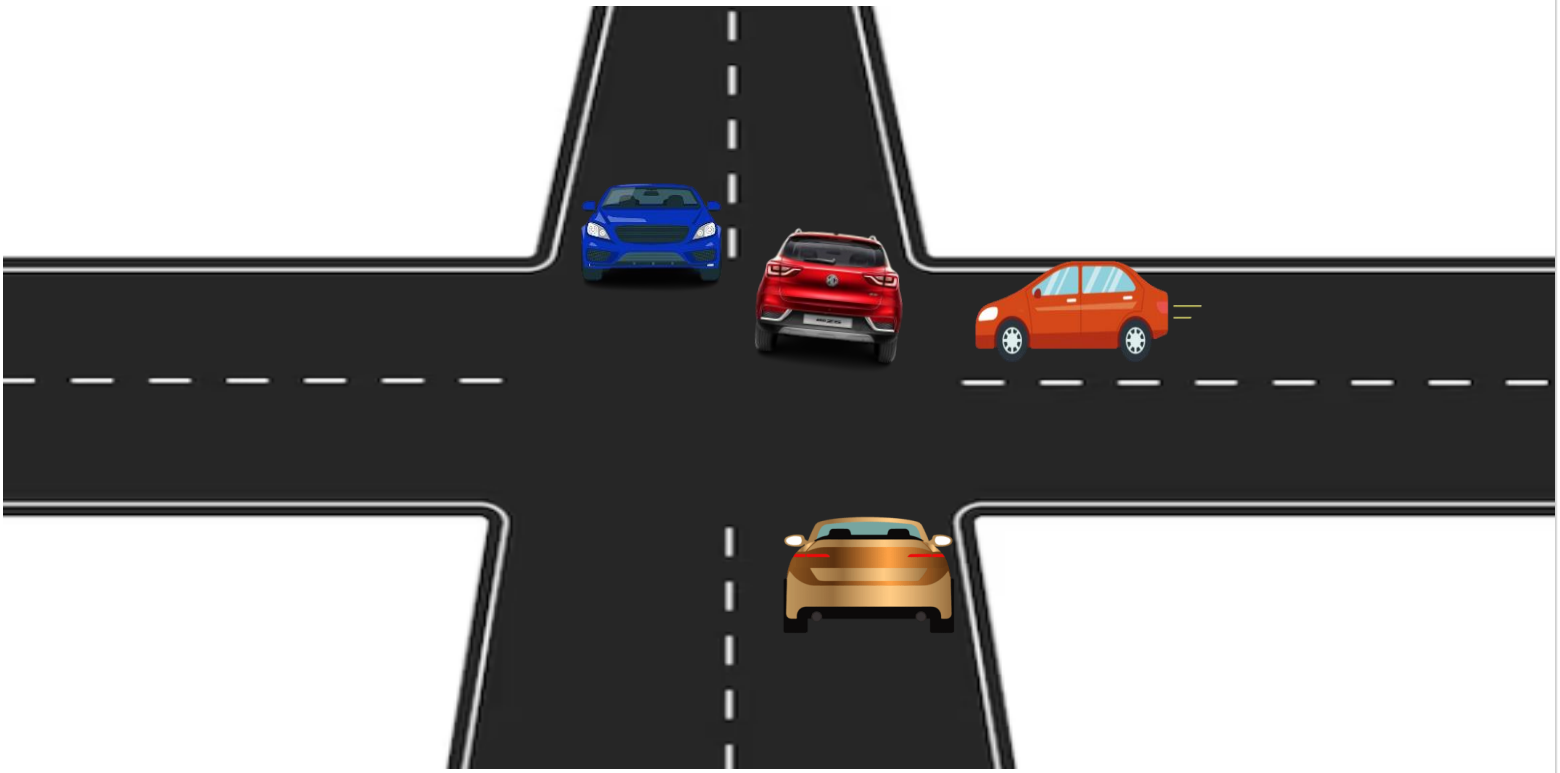


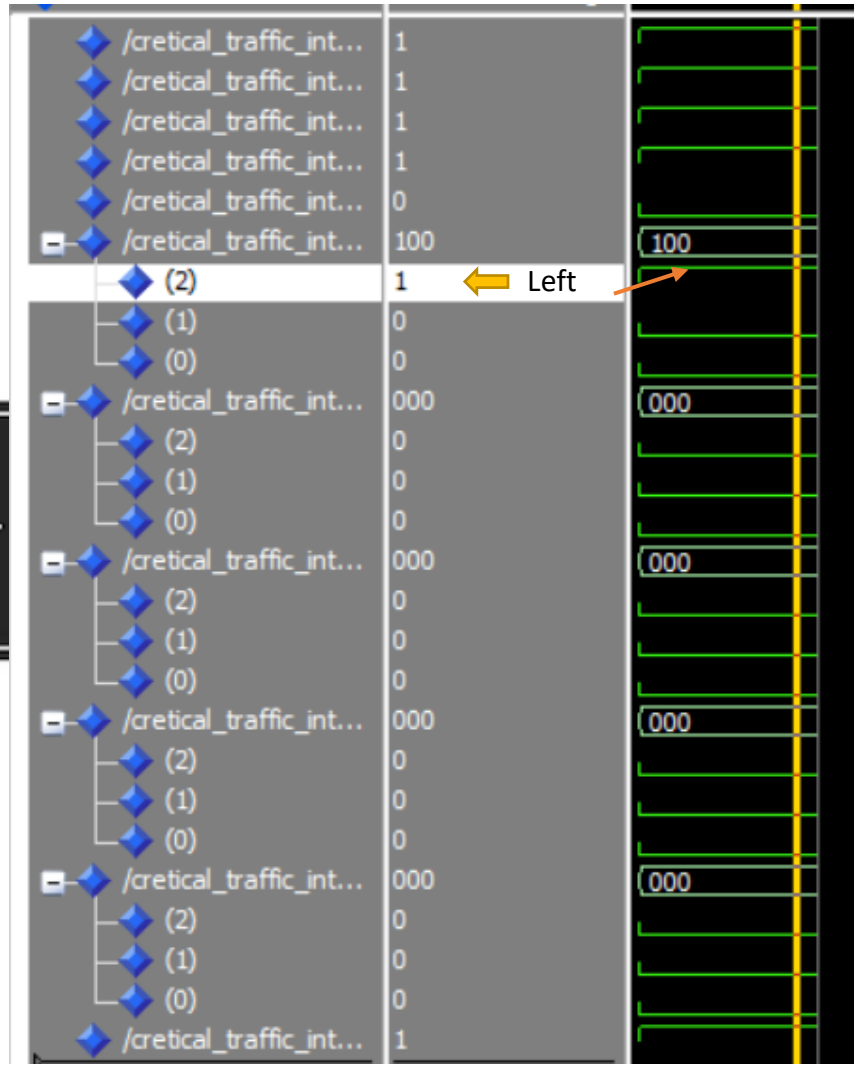


Left direction: There is more problem that's why we will only allow one car at a time

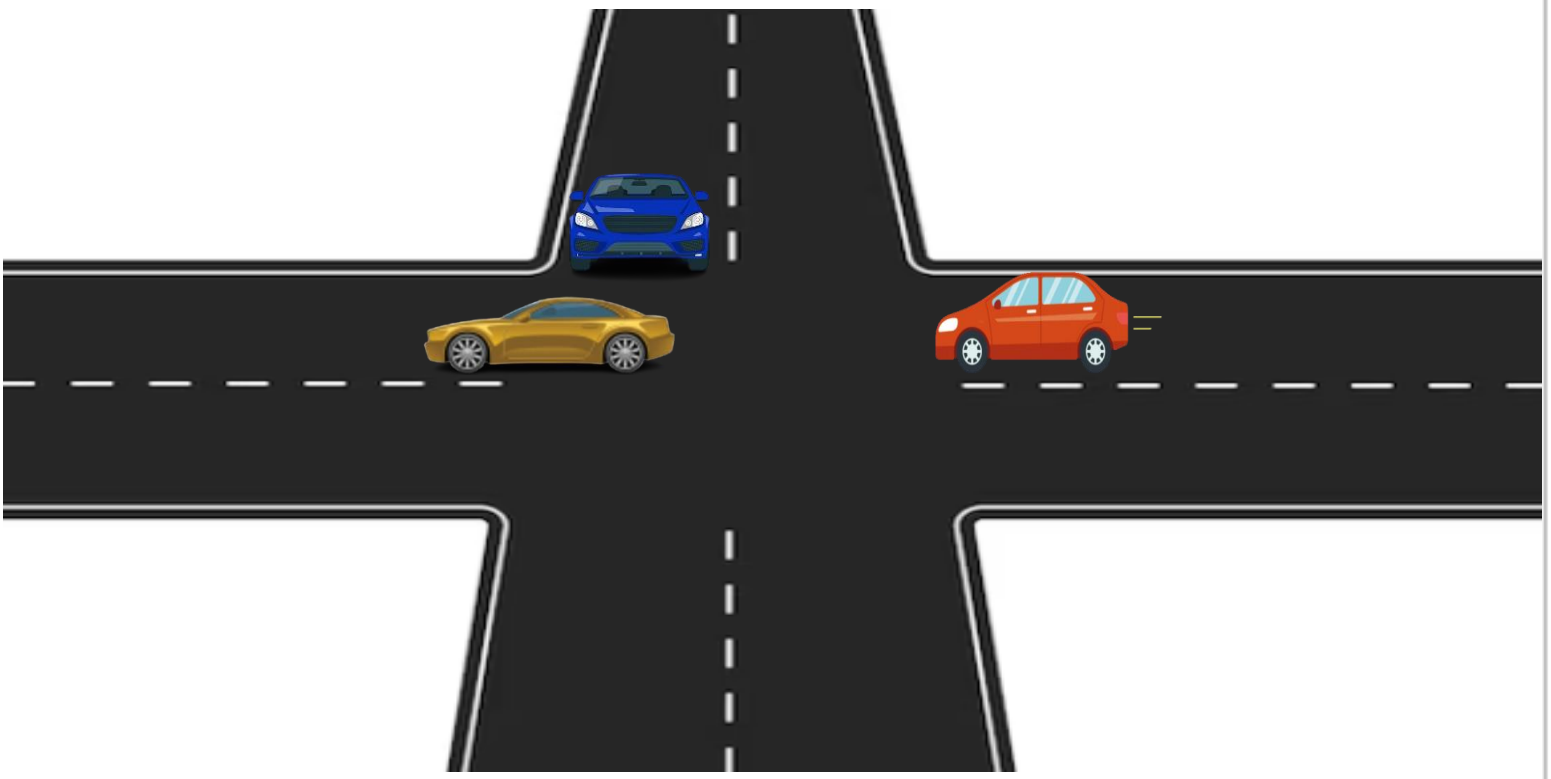


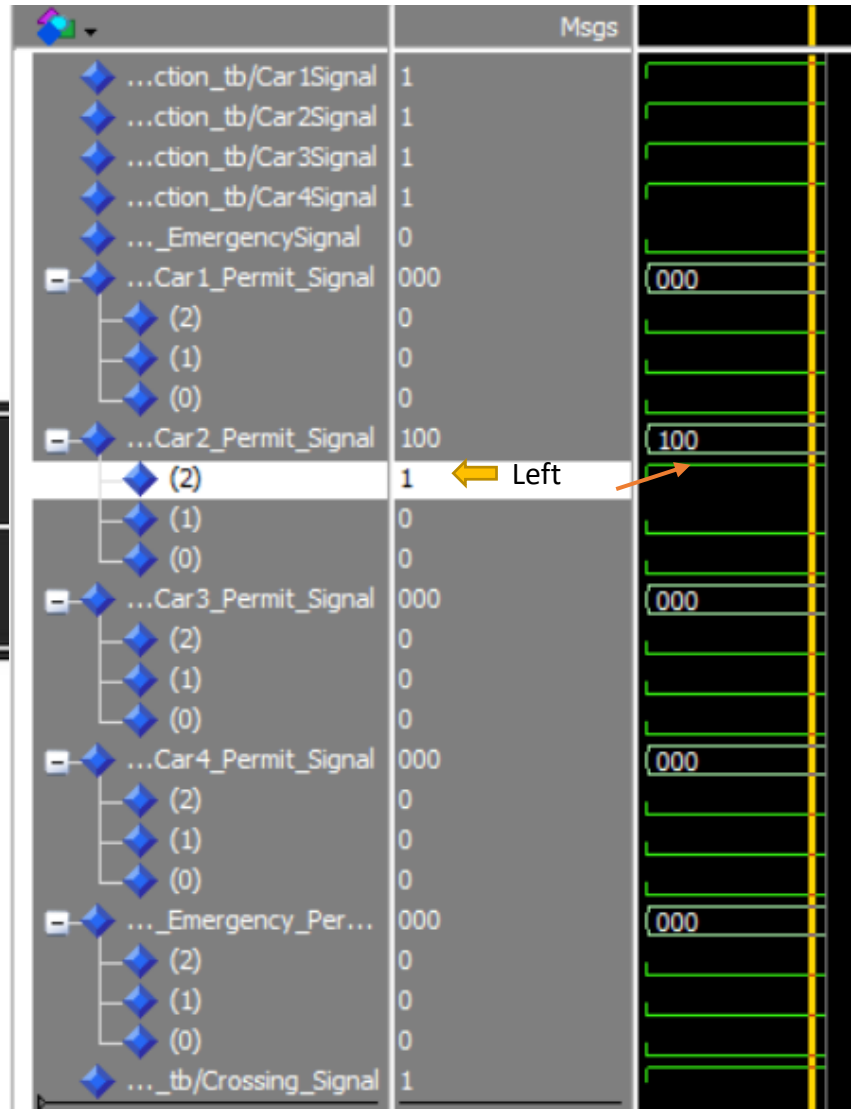
Left direction: Car 1 is crossing



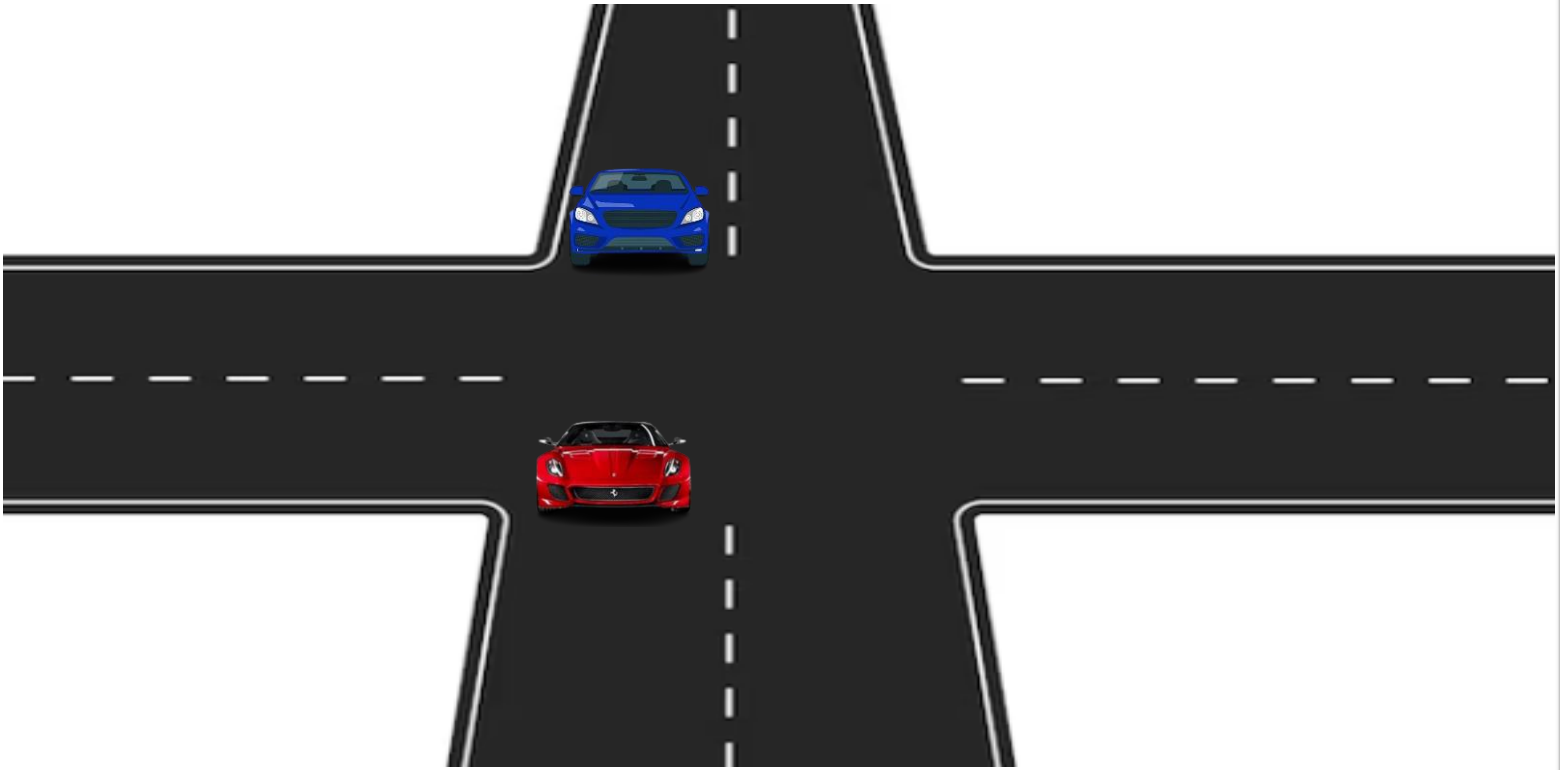


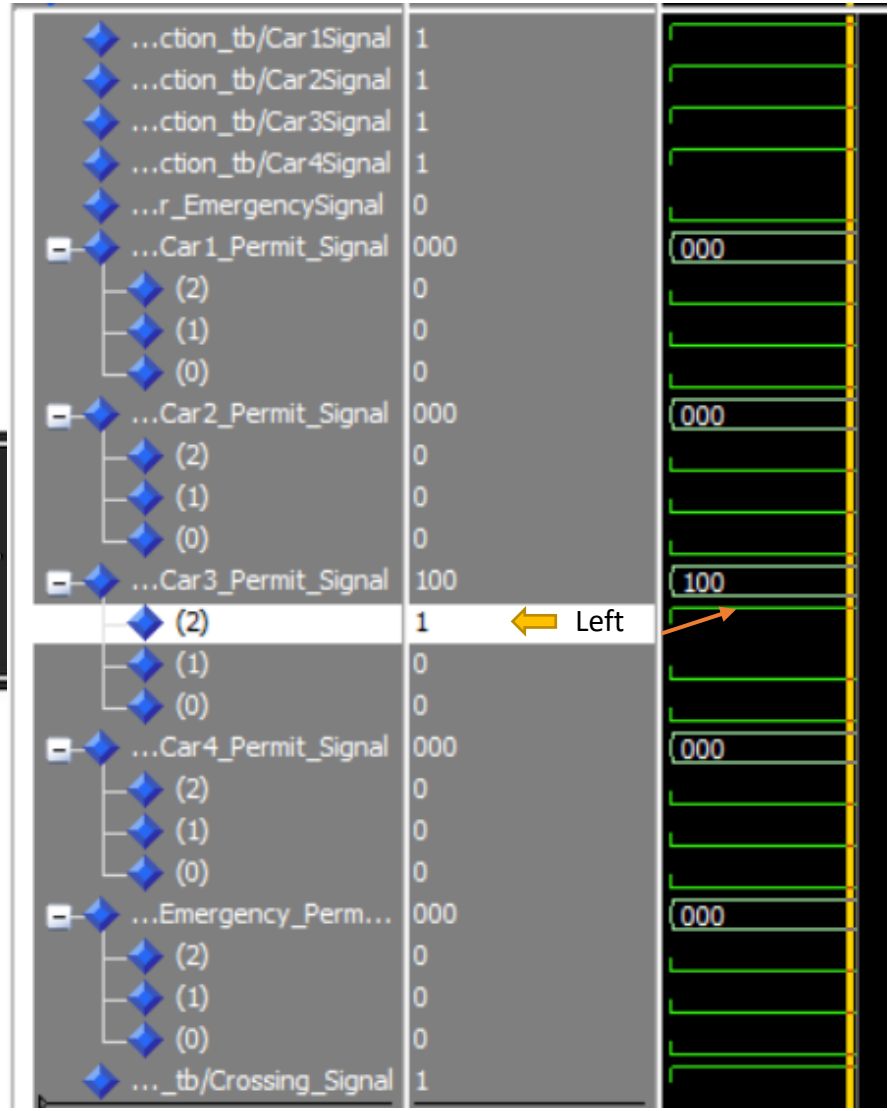
Left direction: Car 2 is crossing



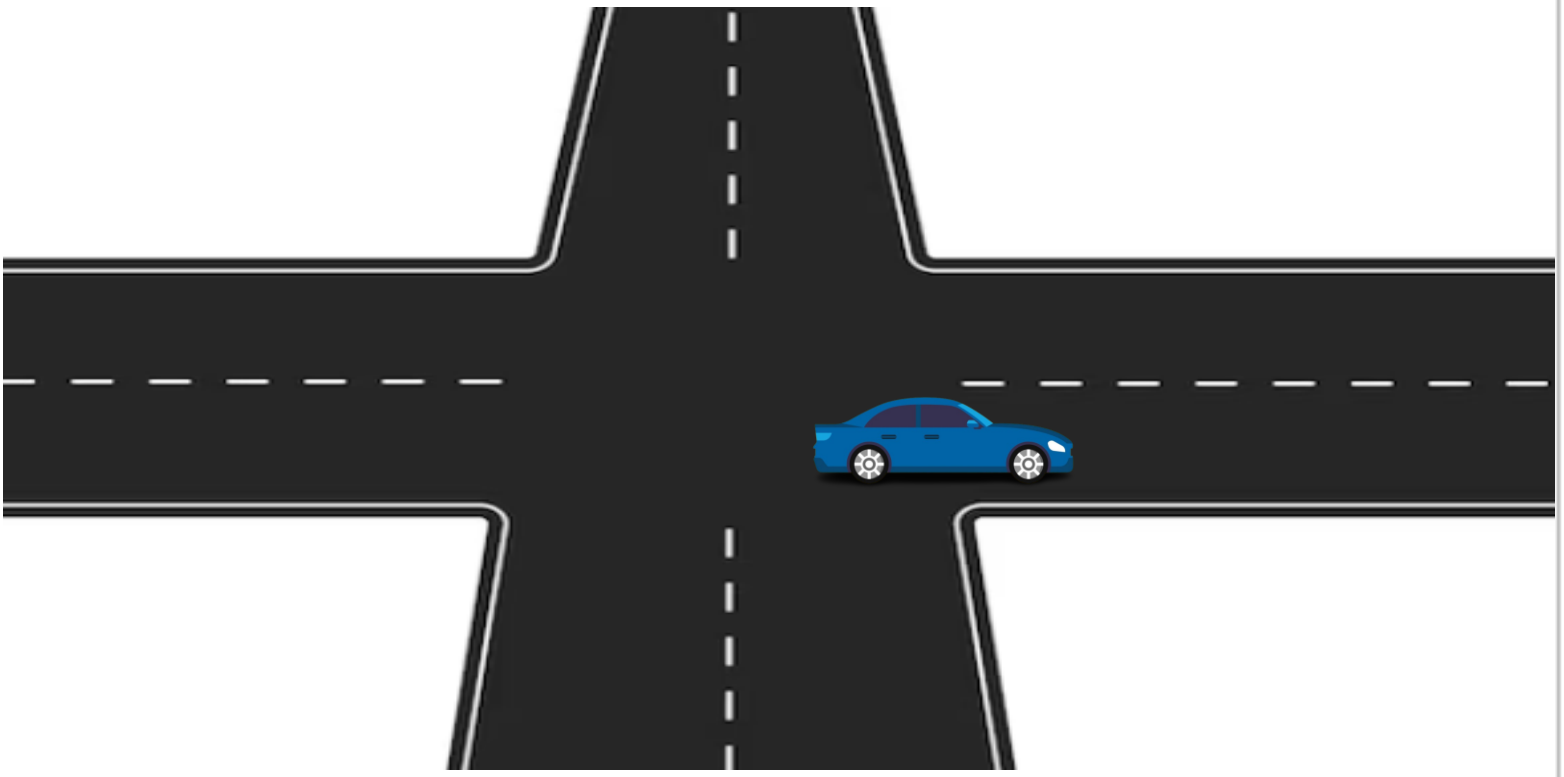


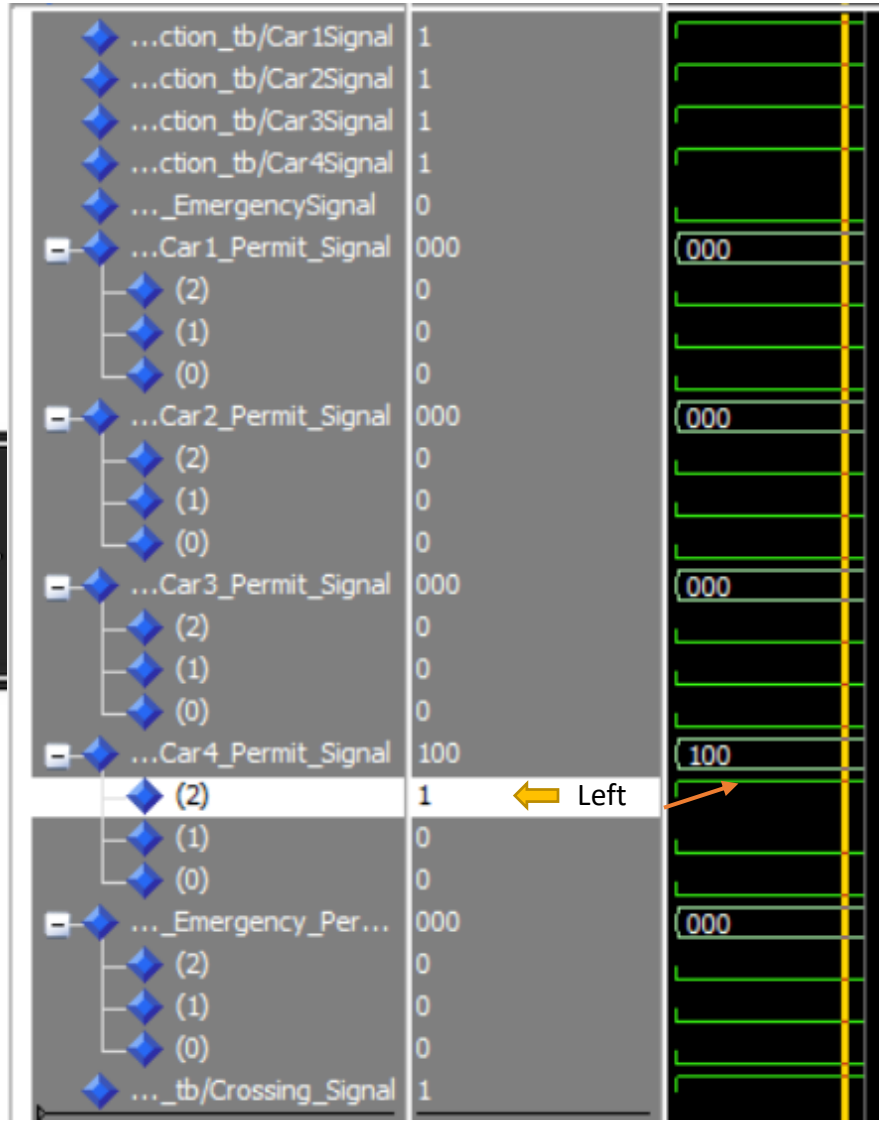
Left direction: Car 3 is crossing



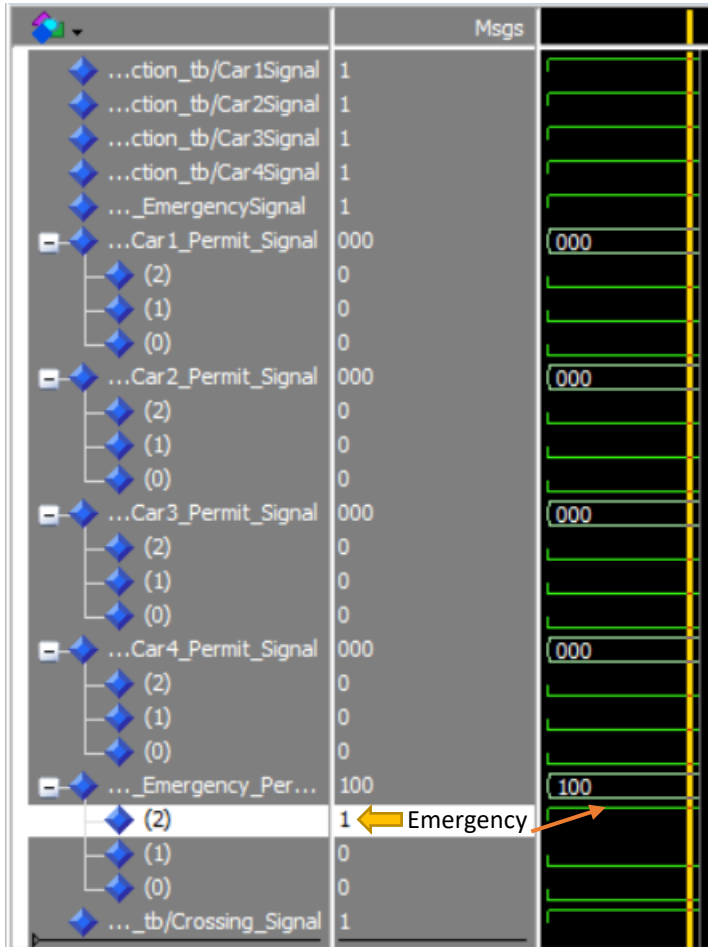


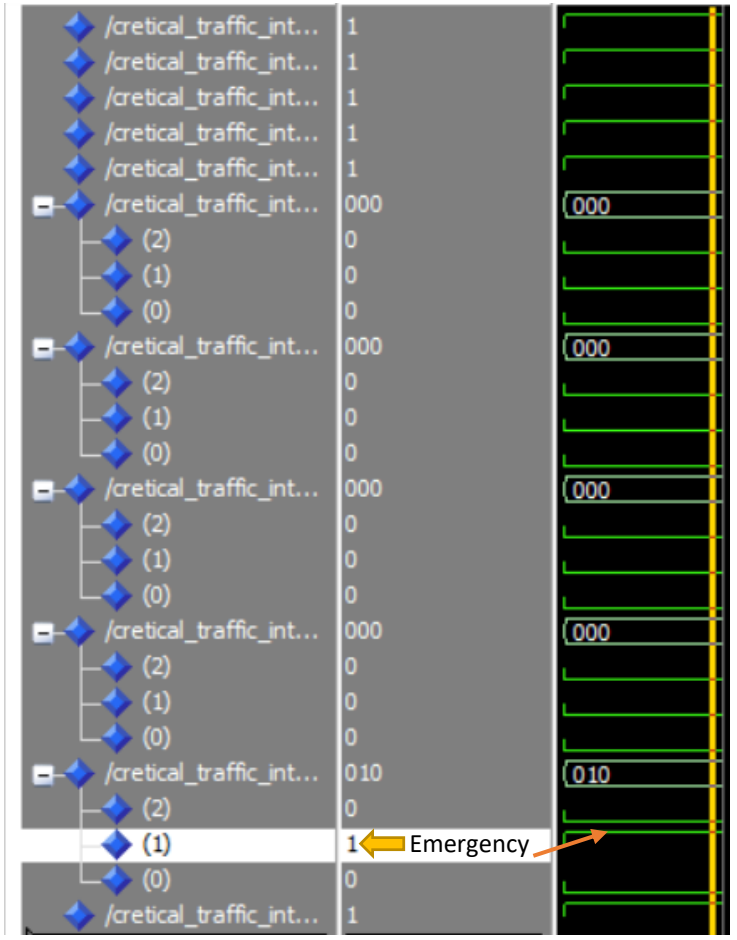
Left direction: Car 4 is crossing

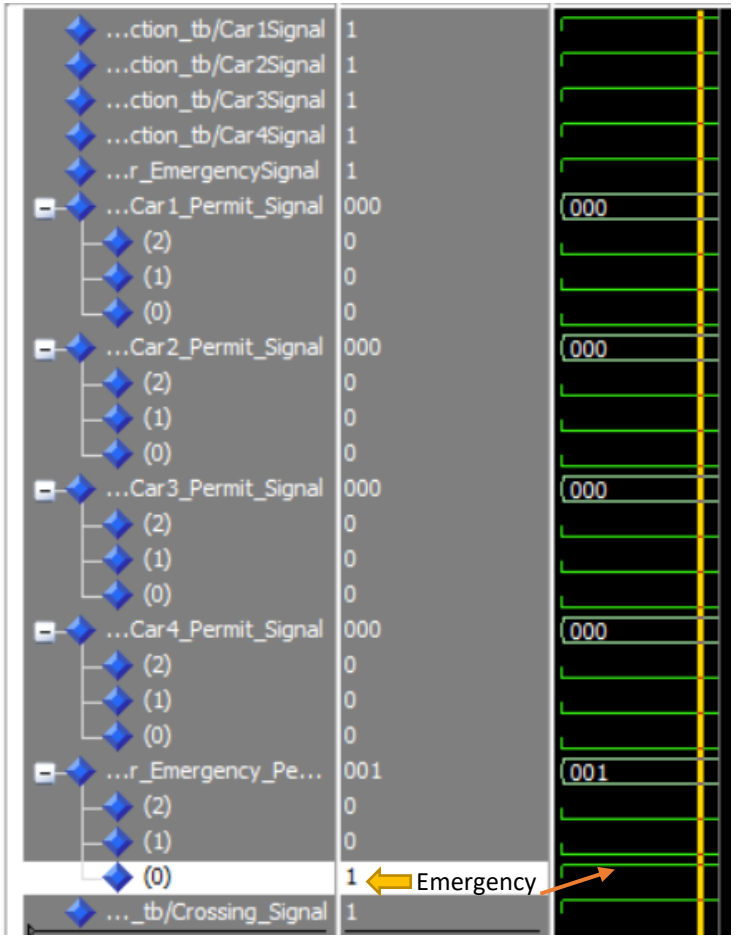




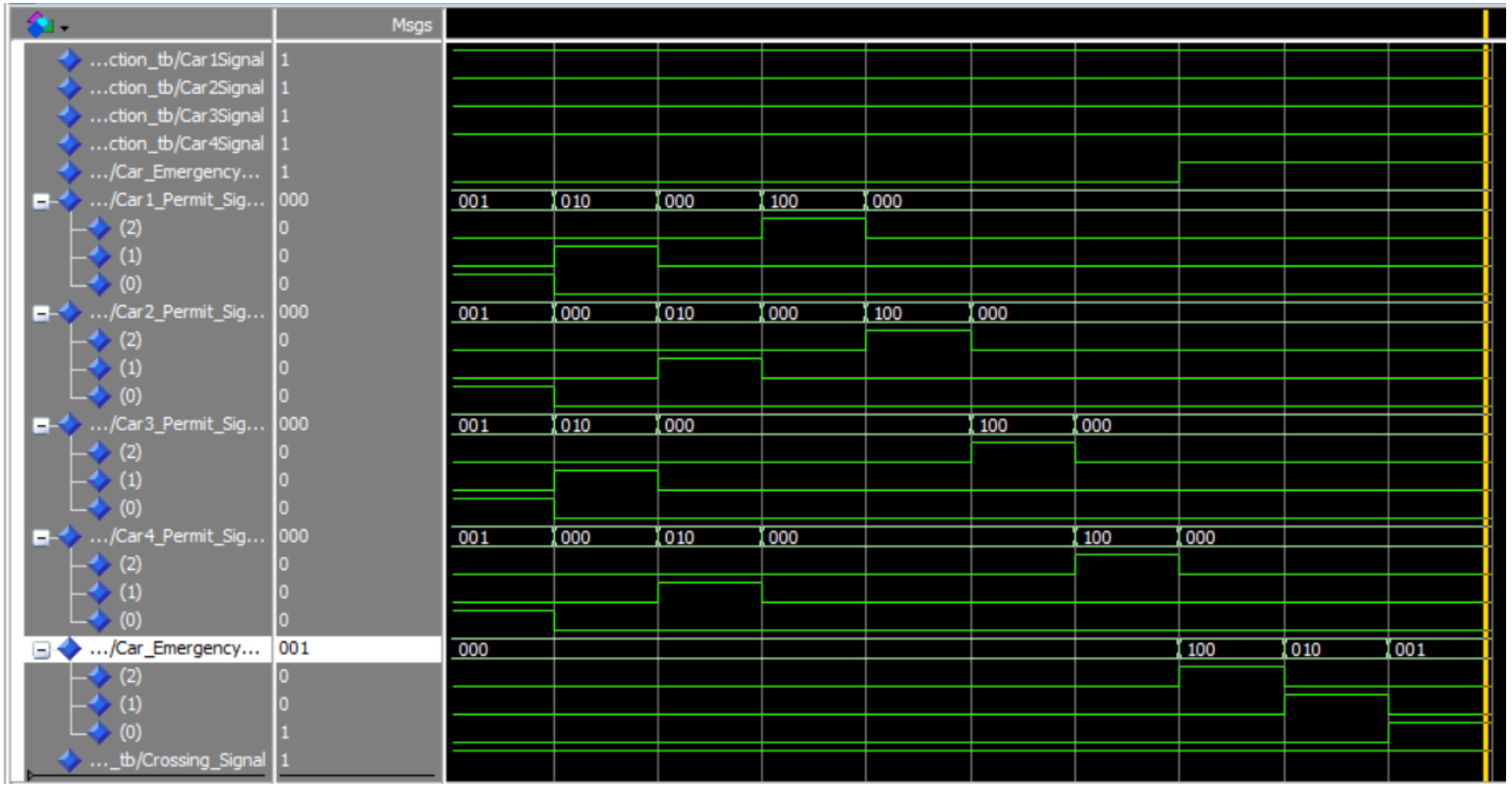
Emergency Car: When emergency car come this time other car is stop and Emergency car are allow to go any direction







VHDL all scenarios simulation together



Conclusion

To sum-up, we can see that by applying several methods
We can able to reduce accident and save travel time and we finally ensure save and secure journey