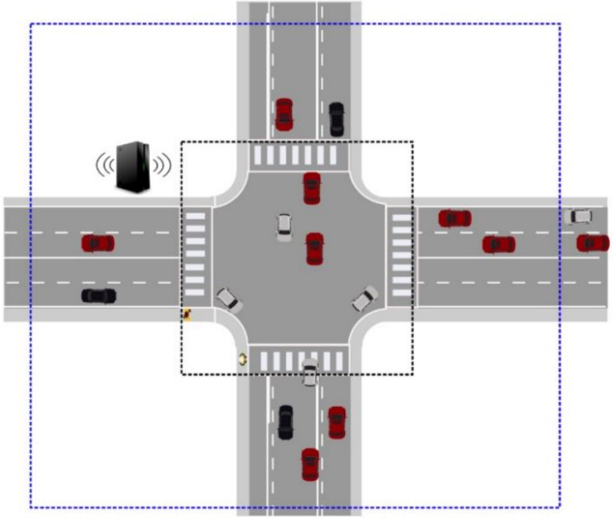
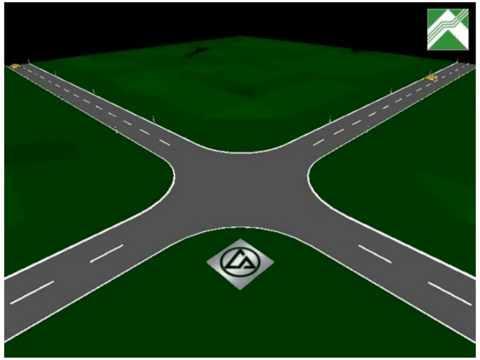
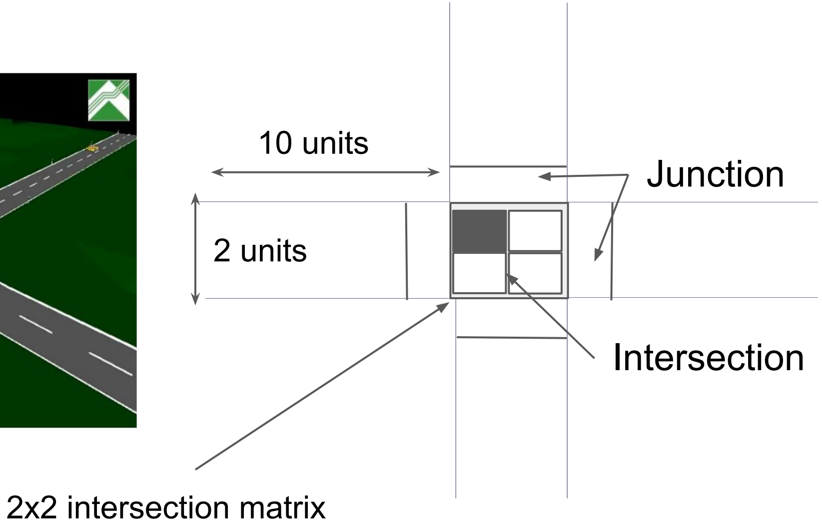
Efficient Cross Traffic Control System

By :- Ravindu Athukorala
Lochana Abhyawardana
Heansuh Lee
Kajeepan Umaibalan

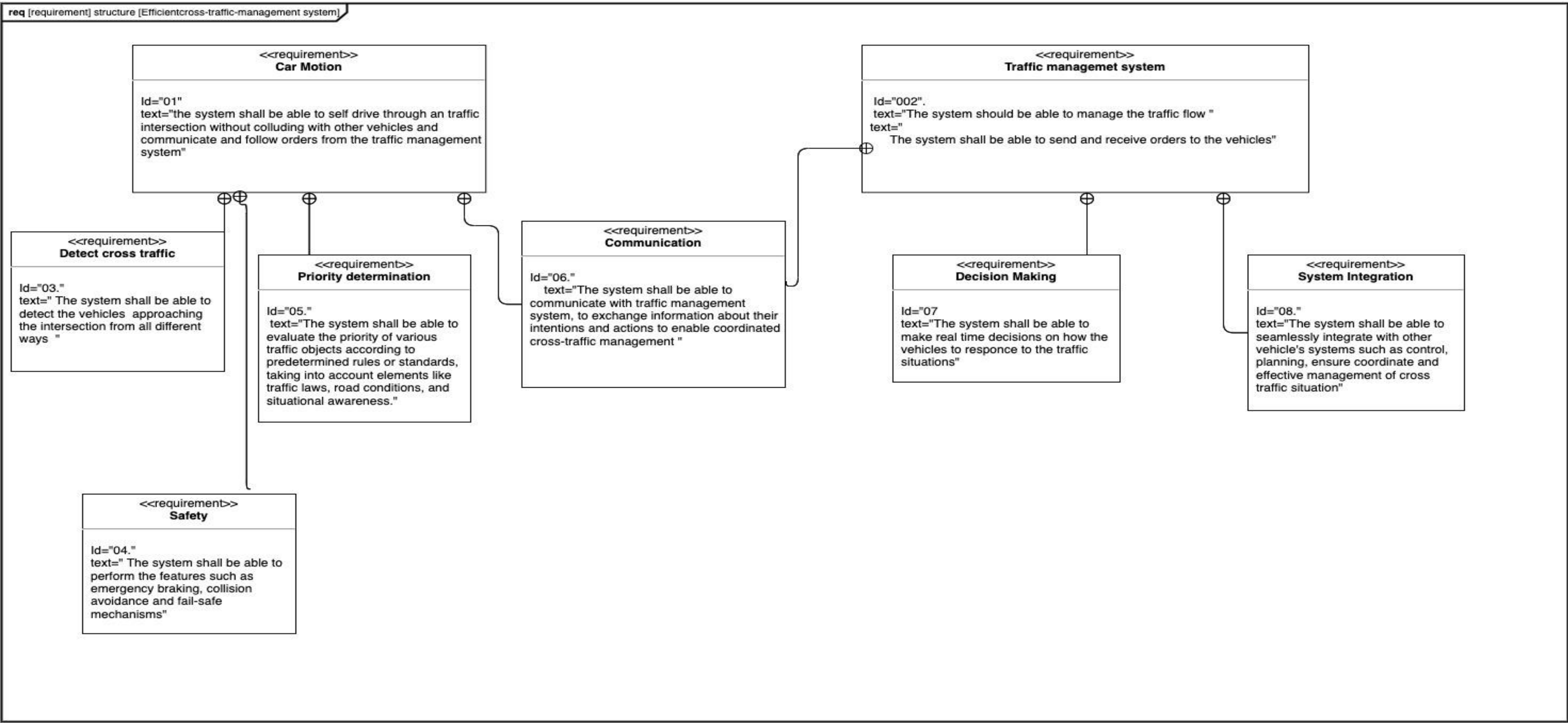
Motivation



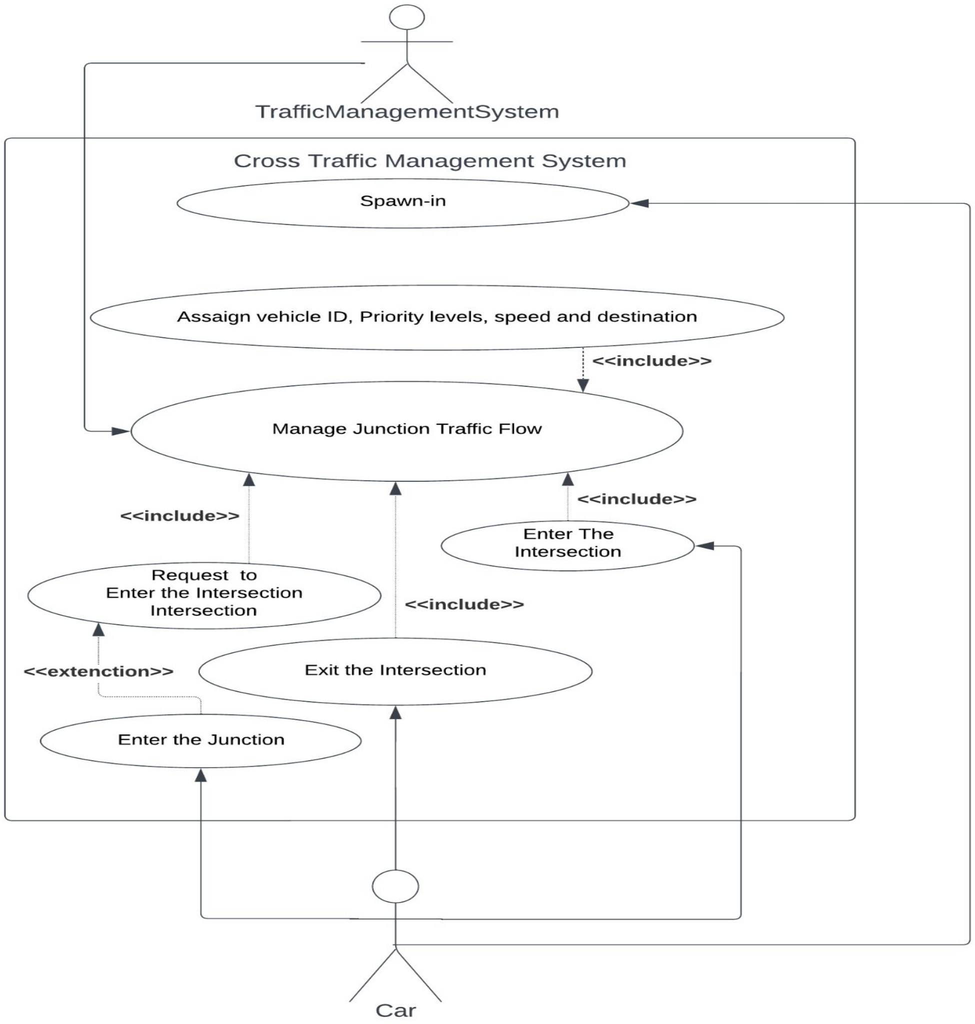



10 units
Junction
2 units
Intersection
2x2 intersection matrix

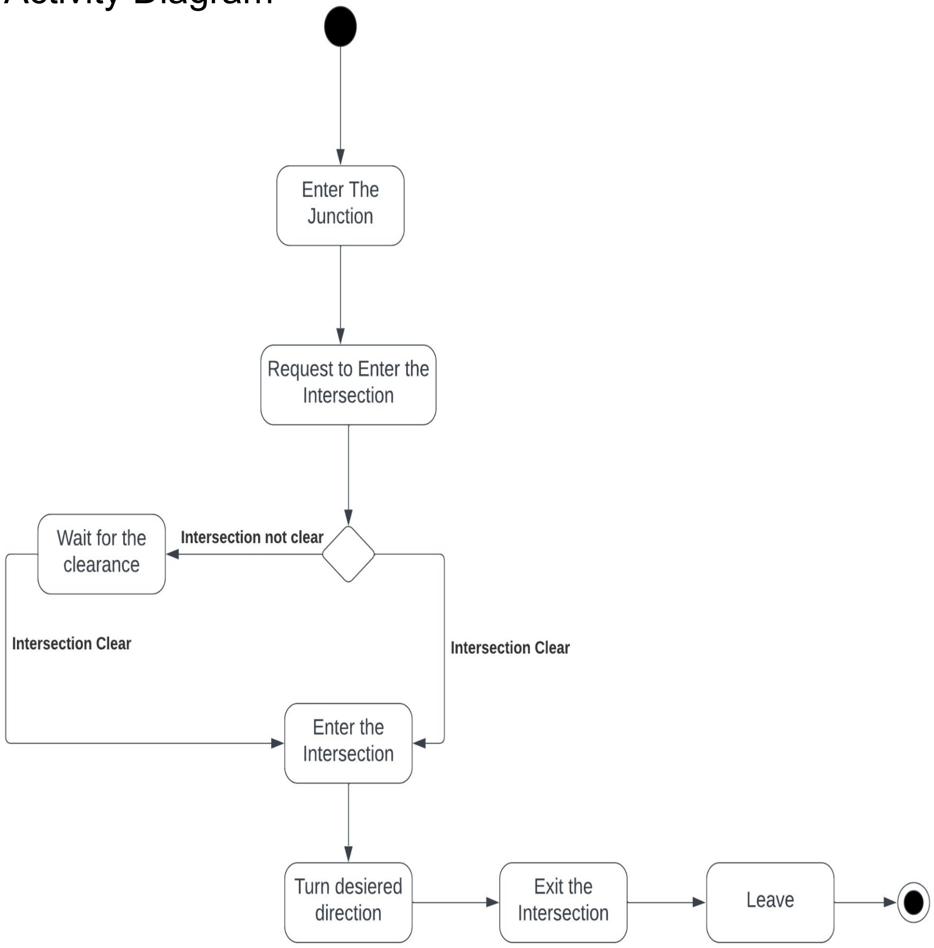
UML Requirement Diagram



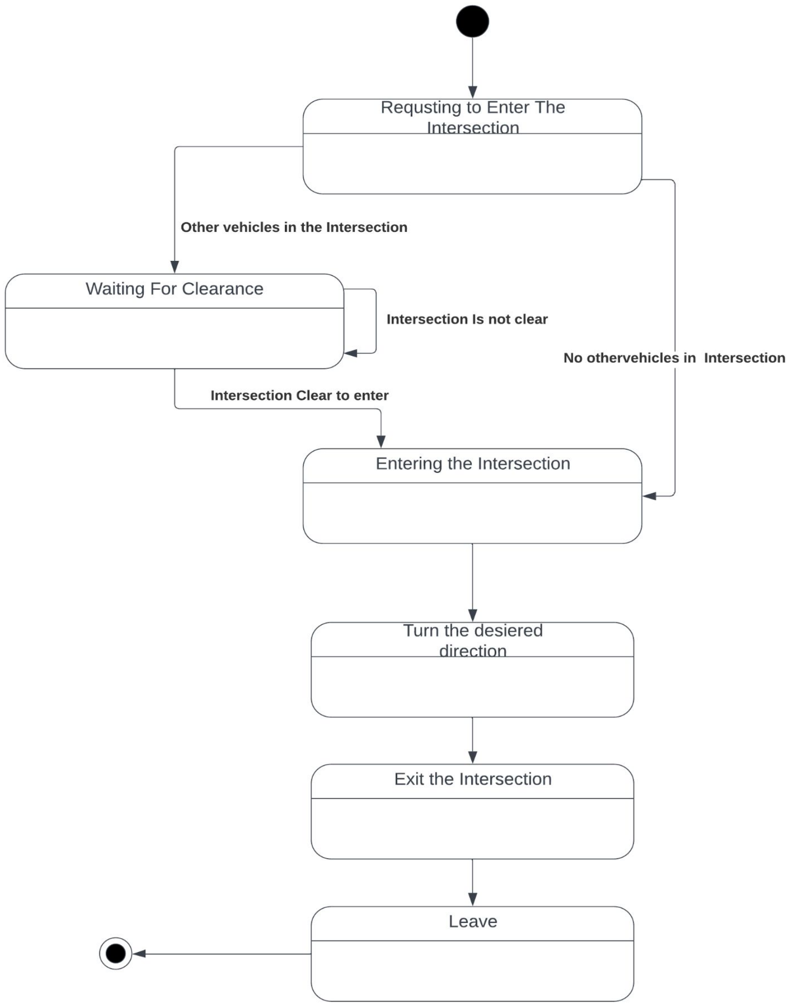
Use Case Diagram



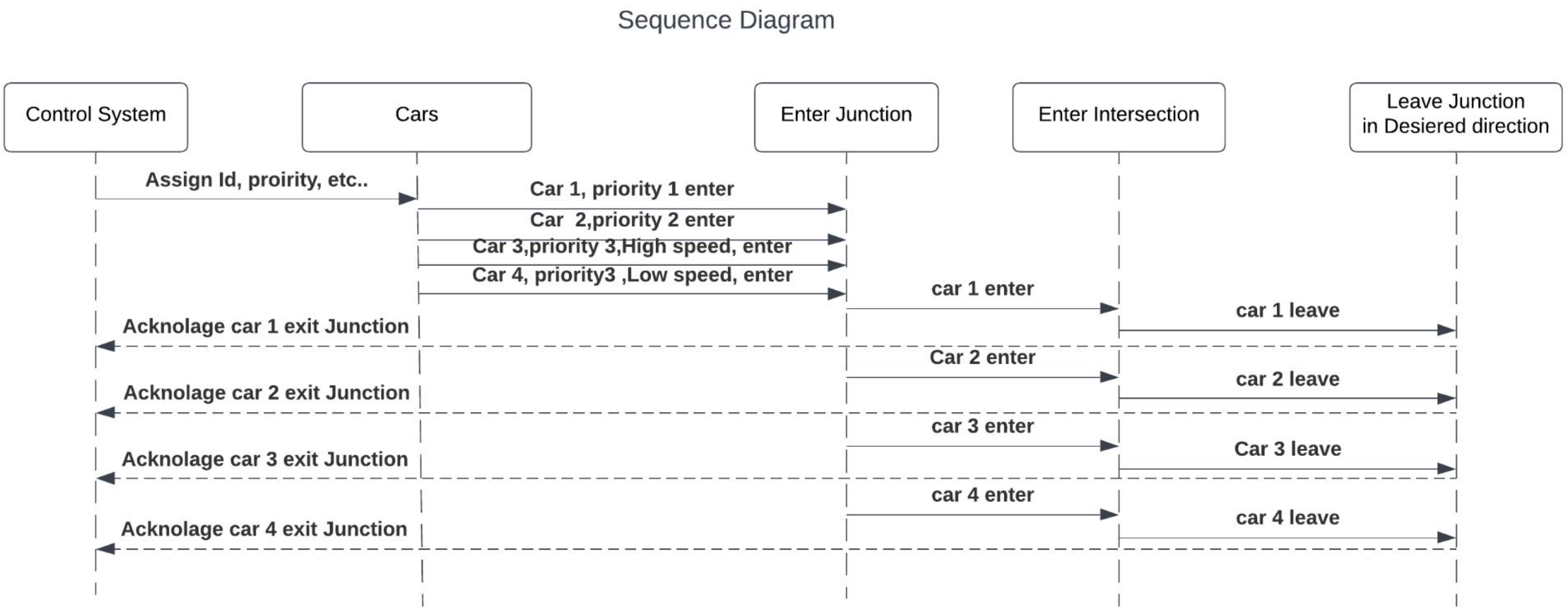
Activity Diagram



State Mechine Diagram



Sequence Diagram



Implementation In C

Spawning Algorithm

- Structs

```c
typedef struct { // custom representation of the data of cars
    int id;
    SpeedLevel speed;
    Direction direction;
    Direction destination;
    PriorityValue priority;
}
```

- Vehicle Spawn Function

```c
void VehicleSpawn() {
```

- Generating Randoms

```c
int randomPriority = rand() % 3;  // Generate a random priority level for the vehicle
```

```c
int randomSpeedLevel = rand() % 3; // Generate a random speed level for the vehicle
```

```c
int id = rand() % 50; //random value from 0 to 50
int directionNum = rand() % 4; // Generate a random direction for the vehicle
```

- Switch case

```
int randomPriority = rand() % 3;  // Generate a random priority level for the vehicle
PriorityValue priority;
switch (randomPriority) {
  case 0:
    priority = emergency;
    break;
  case 1:
    priority = paid;
    break;
  default:
    priority = normal;
    break;
}
```

- Main function

```
int main() {
  srand(time(0)); // random number generator

  for (int j = 0; j < 50; j++) { // Spawn vehicles
    VehicleSpawn();
  }
```

FreeRTOS Implementation

FreeRTOS_Final

```
#include <Arduino.h>
#include <Arduino_FreeRTOS.h>
```

```
Direction destination;
PriorityValue priority = normal;

if ((direction == SOUTH && directionNum == EAST) ||
    (direction == EAST && directionNum == NORTH) ||
    (direction == NORTH && directionNum == WEST) ||
    (direction == WEST && directionNum == SOUTH) ||
    (direction == WEST && directionNum == EAST) ||
    (direction == EAST && directionNum == SOUTH)) {
  destination = (Direction)((directionNum + 1) % 4);
  priority = priority;
} else {
  destination = (Direction)((directionNum + 2) % 4);
}

Serial.print("Vehicle spawned with id: ");
Serial.print(id);

if ((direction == SOUTH && destination == EAST) ||
    (direction == EAST && destination == NORTH) ||
    (direction == NORTH && destination == WEST) ||
    (direction == WEST && destination == SOUTH) ||
    (direction == WEST && destination == EAST) ||
    (direction == EAST && destination == SOUTH)) {
  Serial.print(" || Move with priority: ");
```

```
void setup() {
  Serial.begin(9600);

  xTaskCreate(
    VehicleSpawn,          // Function that implements the task
    "VehicleSpawnTask",    // Text name for the task
    128,                   // Stack size in bytes
    NULL,                  // Parameter to pass to the task
    1,                     // Task priority
    NULL                   // Task handle (not used)
  );

  vTaskStartScheduler();   // Start the FreeRTOS scheduler
}

void loop() {
  // Empty. The tasks are executed in FreeRTOS scheduler.
}
```

/dev/cu.usbmodem14101

Send

```
Vcle spawned with id: 7 || Move with priority: East to North || from East to the destination South
Vehicle spawned with id: 7 || Move with priority: East to North || from East to the destination South
Vehicle spawned with id: 73 || from West to the destination North
Vehicle spawned with id: 30 || Move with priority: North to West || from North to the destination West
Vehicle spawned with id: 44 || from West to the destination North
Vehicle spawned with id: 23 || Move with priority: East to North || from East to the destination South
Vehicle spawned with id: 40 || Move with priority: East to North || from East to the destination South
Vehicle spawned with id: 92 || from West to the destination North
Vehicle spawned with id: 87 || Move with priority: South to East || from South to the destination East
Vehicle spawned with id: 27 || Move with priority: East to North || from East to the destination South
Vehicle spawned with id: 40 || Move with priority: North to West || from North to the destination West
Vehicle spawned with id: 3 || Move with priority: East to North || from East to the destination South
Vehicle spawned with id: 9 || Move with priority: East to North || from East to the destination South
Vehicle spawned with id: 60 || Move with priority: East to North || from East to the destination South
Vehicle spawned with id: 99 || from West to the destination North
Vehicle spawned with id: 16 || Move with priority: South to East || from South to the destination East
Vehicle spawned with id: 97 || from West to the destination North
Vehicle spawned with id: 12 || Move with priority: South to East || from South to the destination East
Vehicle spawned with id: 10 || Move with priority: East to North || from East to the destination South
Vehicle spawned with id: 79 || Move with priority: East to North || from East to the destination South
Vehicle spawned with id: 79 || Move with priority: East to North || from East to the destination South
Vehicle spawned with id: 67 || Move with priority: North to West || from North to the destination West
Vehicle spawned with id: 93 || Move with priority: North to West || from North to the destination West
Vehicle spawned with id: 85 || Move with priority: East to North || from East to the destination South
Vehicle spawned with id: 28 || Move with priority: South to East || from South to the destination East
Vehicle spawned with id: 94 || Move with priority: East to North || from East to the destination South
Vehicle spawned with id: 1 || Move with priority: East to North || from East to the destination South
```

Autoscroll Show timestamp Newline 9600 baud Clear output

Scheduling Algorithm

- **Shortest Remaining Time First (SRTF)**
- **Preemptive** scheduling method
- Derived from the **Shortest Job First (SJF)** scheduling algorithm
- Difference: SRTF is preemptive, but SJF is non-preemptive [1]

2 essential elements, in the crossroad traffic control:

1) **Speed** of the vehicle (1-3 units/time unit) before entering junction
2) **Intersection matrix units**, based on **destination lane** selected (1-3 units)

- Determined in the **junction** of the crossroad traffic control
  i.e. in our model, it is 1 unit before the **intersection**

Priority Allocation

Assumption: All the cars at the junction/intersection NEVER collide, due to their speed variations (i.e. speeds are similar, in junction/intersection)

- Priority based on intersection units > priority based on speed before junction
- Two cars are of same priority based on the intersection units, priority based on speed before junction takes over
  i.e. raising the priority of the car with a higher speed
- Two cars are heading straight, there’s no **collision**, so both cars move together
  i.e. **Priority 2**: **2 units** of intersection matrix units

Visual Implementation Of UPPAAL And VHDL Code

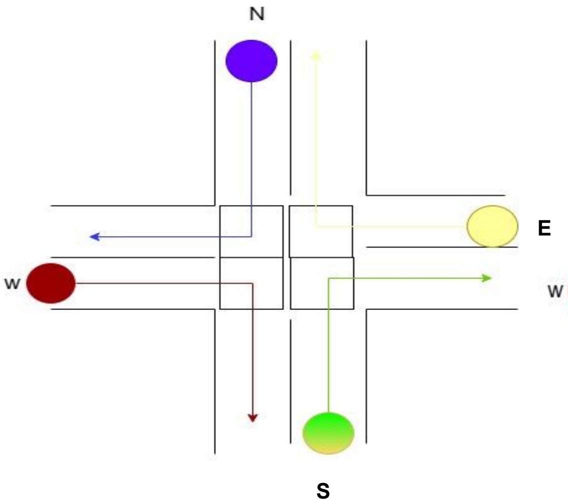
Main scenario

Car 1 - WN

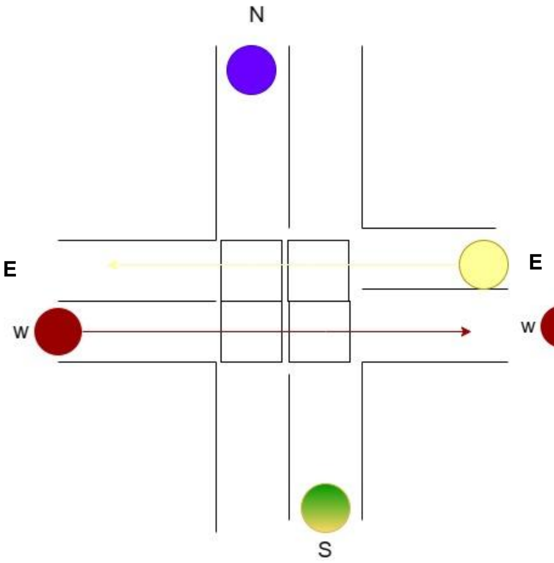
Car 2 - NS

Car 3 - SN

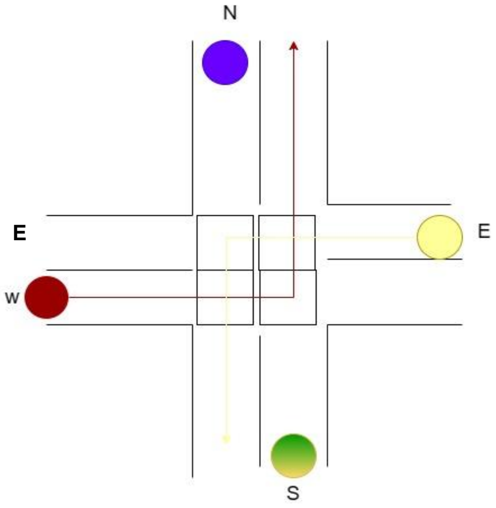
Car 4 - ES



Priority 1



Priority 2



Priority 3

Visual Implementation Of UPPAAL And VHDL Code

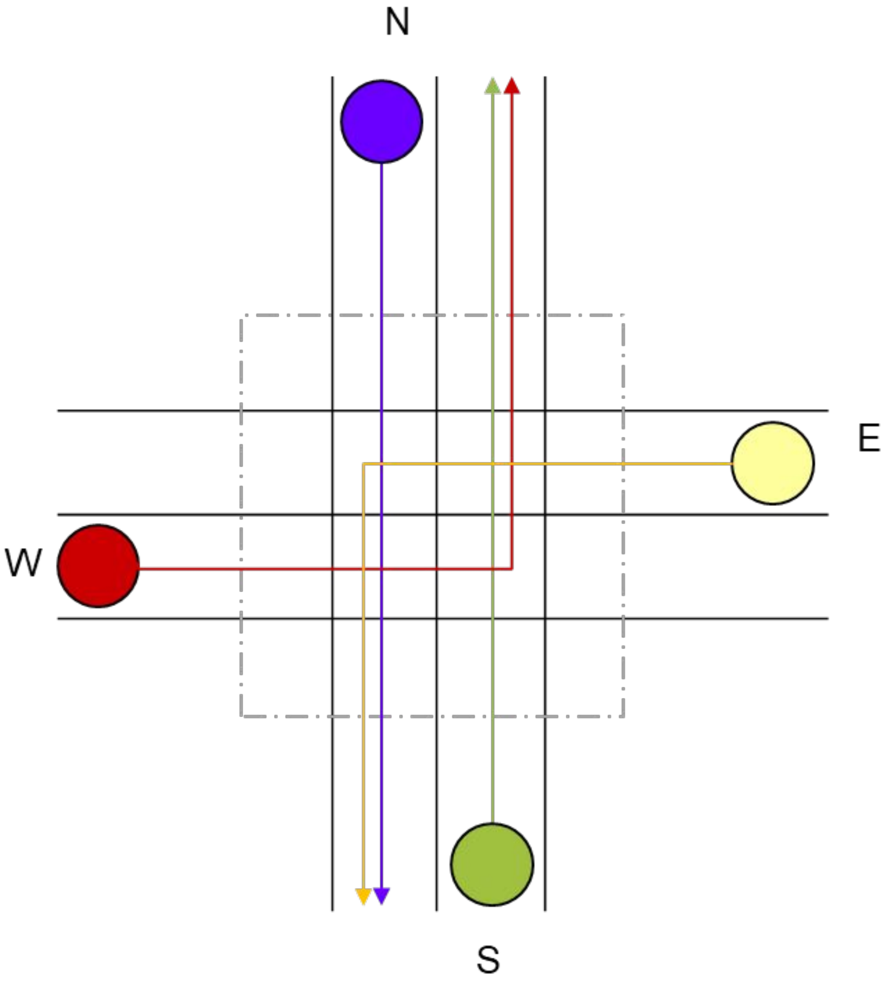
Main scenario

Car 1 - WN

Car 2 - NS

Car 3 - SN

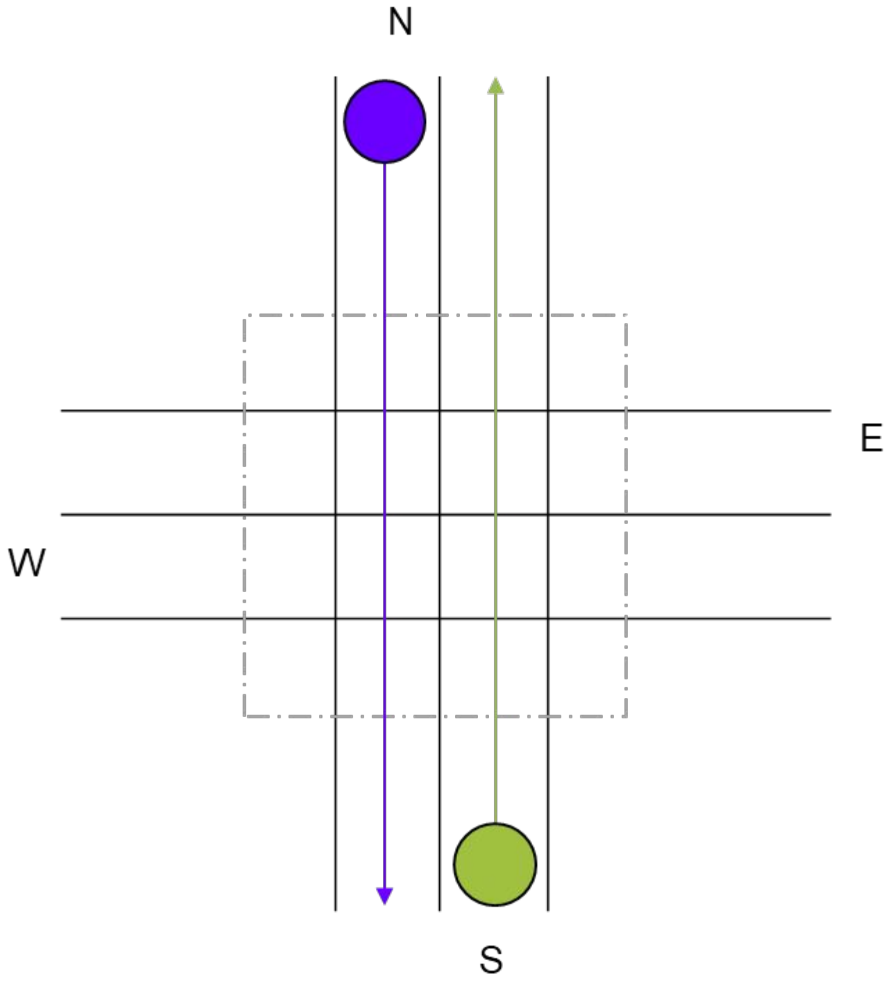
Car 4 - ES



Visual Implementation Of UPPAAL And VHDL Code

Main scenario

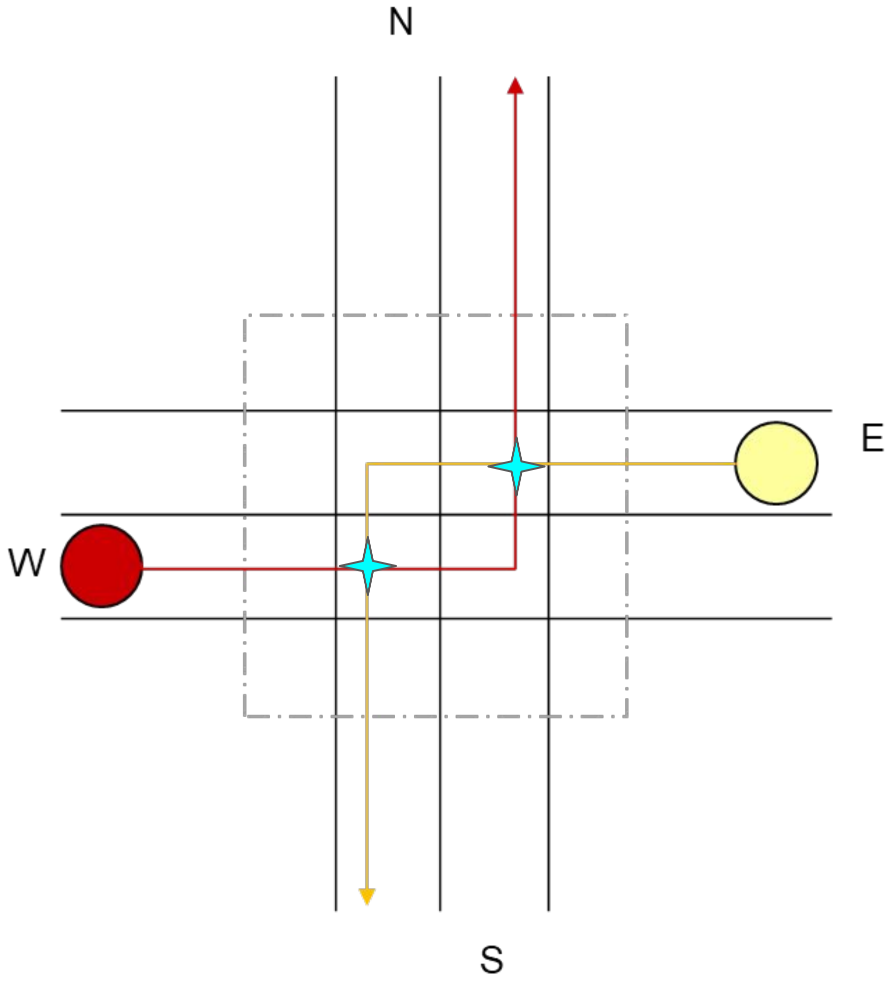
- Car 1 - WN
- Car 2 - NS
- Car 3 - SN
- Car 4 - ES



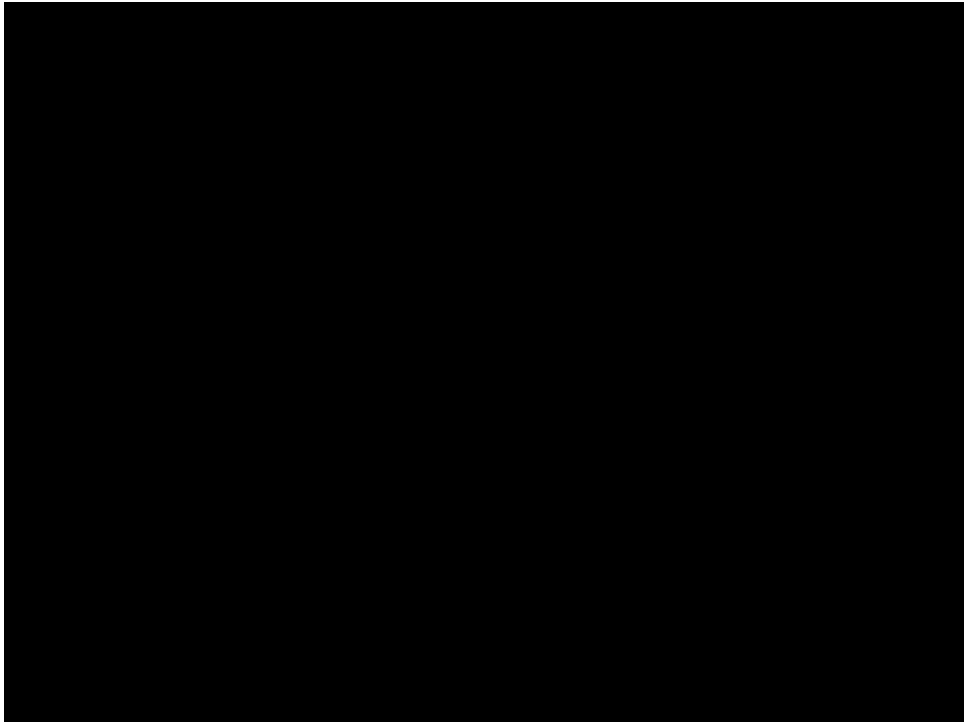
Visual Implementation Of UPPAAL And VHDL Code

Main scenario

- Car 1 - WN
- Car 2 - NS
- Car 3 - SN
- Car 4 - ES



UPPAAL Implementation



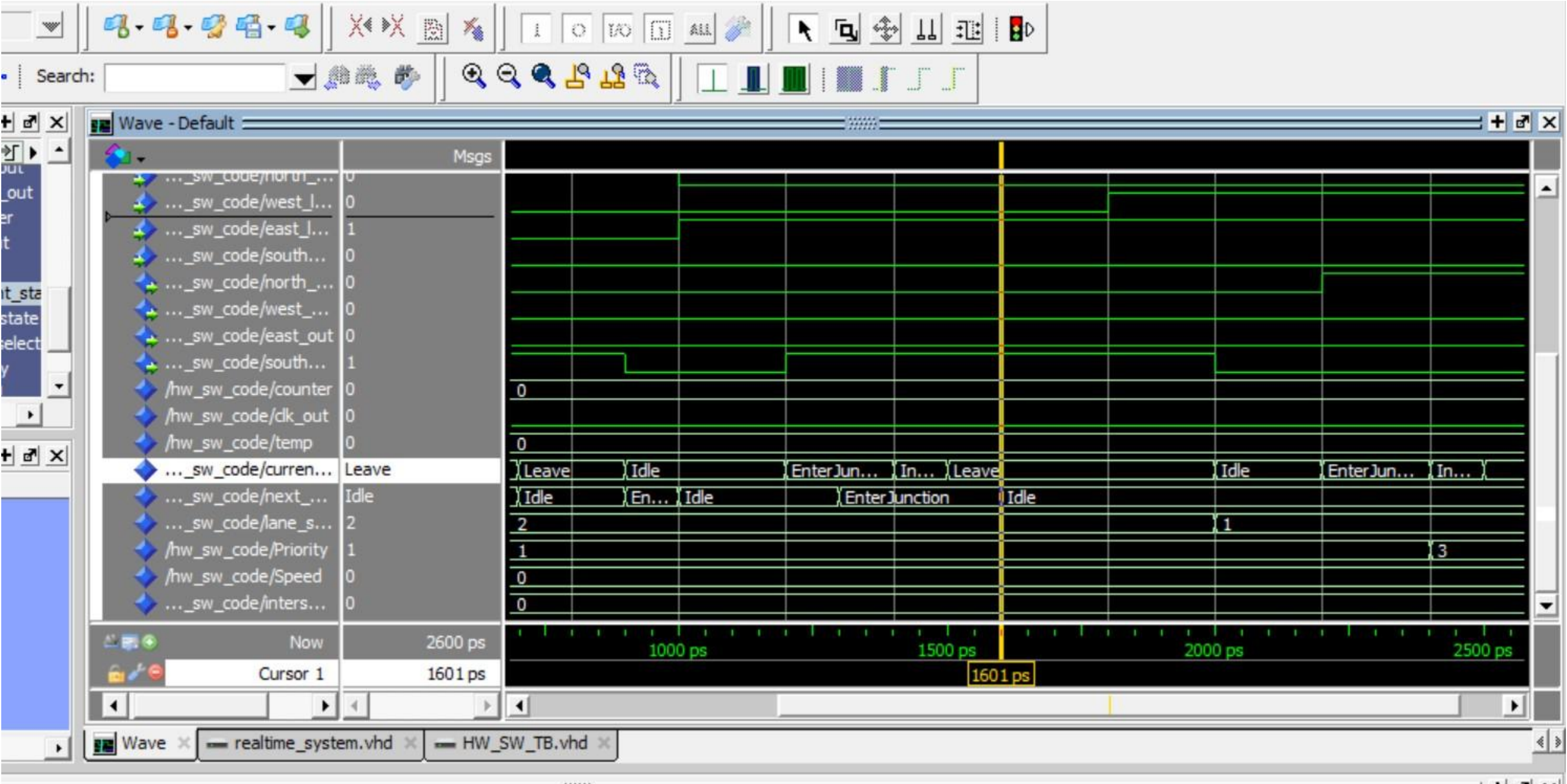
Hardware Codesign with implementation in Modelsim

- Implementation for the scheduling code is done by states and assigning signal values

```
    type StateType is (Idle, EnterJunction, Intersection, Leave);
    signal current_state, next_state : StateType;
    signal lane_selected : integer range 0 to 4 := 0;
    signal Priority : integer range 0 to 4 := 0;
    signal Speed : integer range 0 to 3 := 0;
    signal intersectionMatrix : integer range 0 to 3 := 0;
```

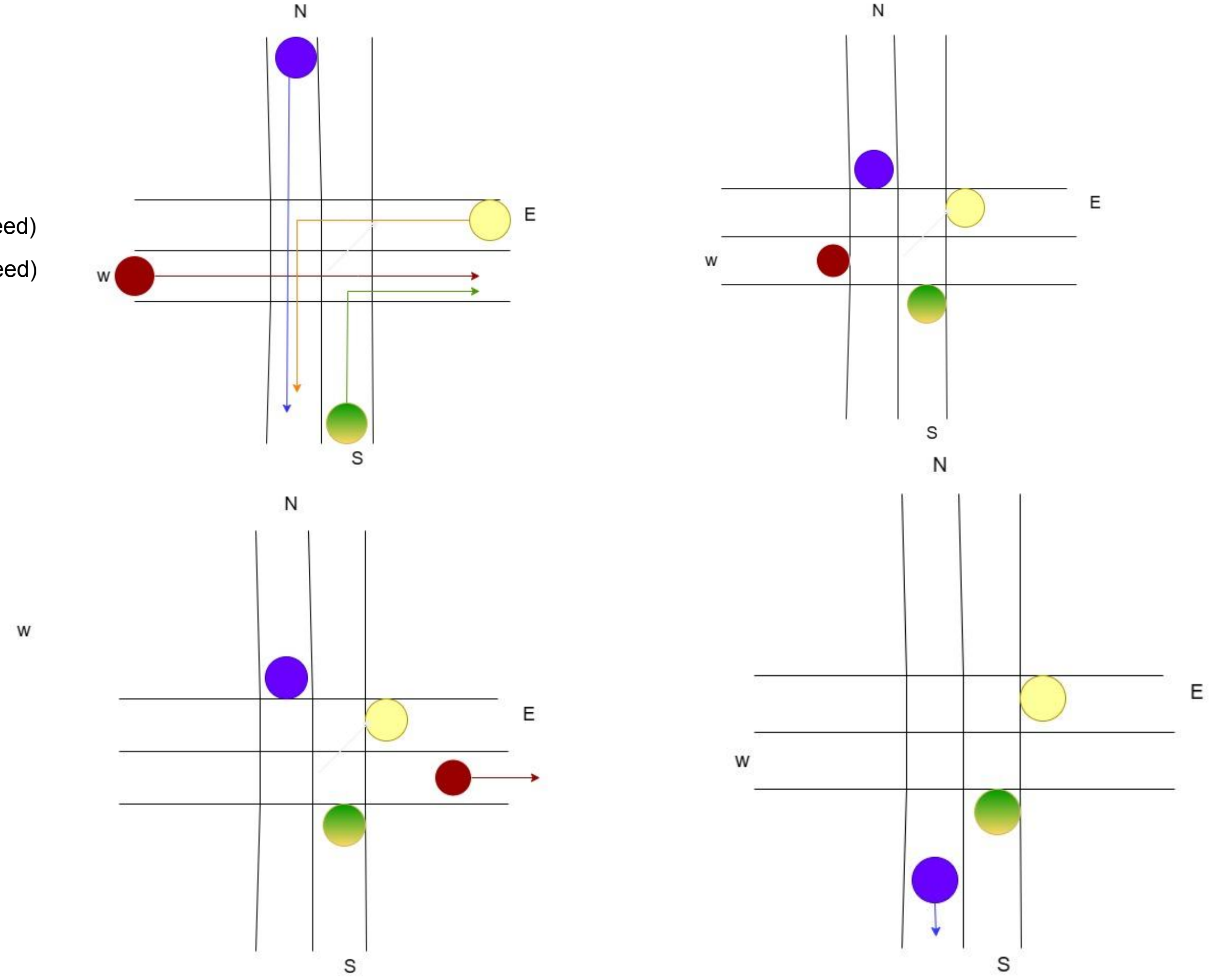
```
process (current_state, car1, car2, car3, car4)
begin
    next_state <= current_state;

    case current_state is
        when Idle =>

            if car1 = '1' then
                if west_lane = '1' then
                    lane_selected <= 1;
                    next_state <= EnterJunction;
                end if;
```

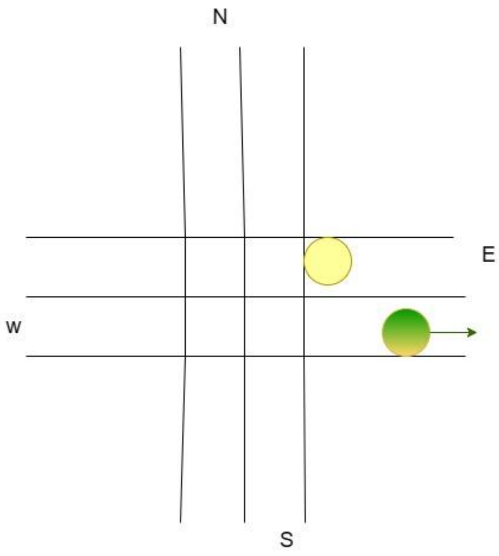
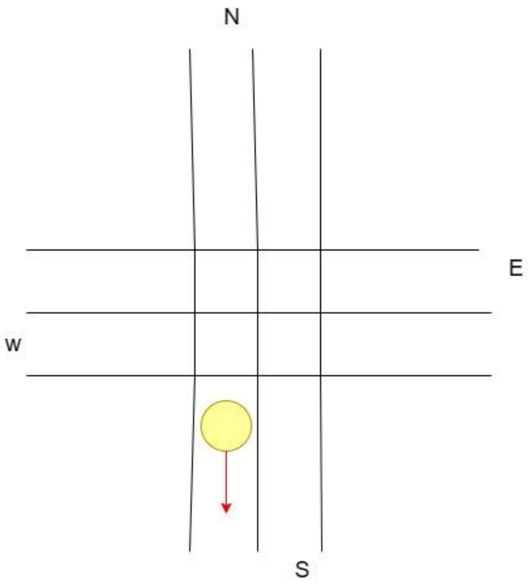
Simulation Model in Modelsim



Scenario 2

- Car 1: priority 1
- Car 2: priority 2
- Car 3: priority 3(High Speed)
- Car 4: priority 3(LOW Speed)




N
E
W
S

N
E
W
S

Summary

This project presents an efficient cross-traffic management control system for autonomous cars. The system utilizes the Short Remaining Time First (SRTF) algorithm to prioritize cars at intersections. It incorporates FreeRTOS C code for task management, VHDL code for system implementation, and UPPAAL modeling for verification. By assigning priority based on predefined rules, the system optimizes traffic flow, ensuring smoother and safer interactions between autonomous vehicles. The project demonstrates improved efficiency and performance, offering potential for future enhancements and advancements in traffic management for autonomous cars.

References

[1] Modern Operating Systems. Andrew S. Tanenbaum, Herbert Bos. Pearson Education, 2015. p.158.