Efficient Cross Traffic Control System

Creative implementation of the algorithm to handle autonomous vehicles at the intersection

1st Ravindu Budhara Athukorala
dept. Electronic Engineering
Hochschule Hamm Lippstadt
Lippstadt, Germany
ravindu.athukorala@stud.hshl.de

2nd Kajeepan Umaibalan
dept. Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt,Germany
kajeepan.umaibalan@stud.hshl.de

3rd Heansuh Lee
dept. Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
heansuh.lee@stud.hshl.de

4th Lochana Abhayawardana
dept. Electronic Engneering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
lochana.abhayawardana@stud.hshl.de

***Abstract*—This paper discusses efficient cross-traffic control system for autonomous vehicles in a junction and an intersection. It focuses on the development of an intelligent traffic management system that leverages real-time scheduling algorithms to optimize the coordination and prioritization of autonomous vehicles at the intersection with the shortest remaining time first (SRTF) scheduling algorithm. By considering the dynamic nature of traffic patterns and the real-time constraints of autonomous vehicles, the proposed system aims to minimize traffic congestion and reduce delays.**

I. Introduction

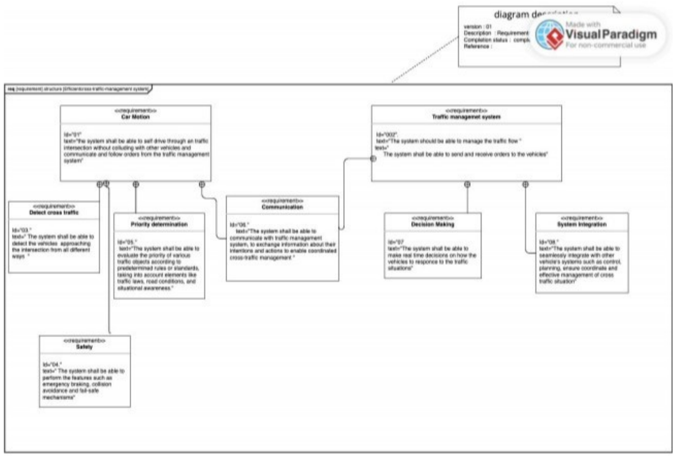
Traditional traffic management systems often lead to congestion, delays, and frustration for drivers. However, our efficient cross-traffic system seeks to change that by leveraging advanced technologies and intelligent algorithms. It will synchronize traffic signals, adapt to real-time conditions, and prioritize the smooth flow of vehicles. Implementing an

efficient cross-traffic system holds tremendous potential. It can drastically reduce commute times, enhance road safety, and minimize carbon emissions. Additionally, by facilitating a seamless intersection experience, it improves the overall quality of life for drivers.

II. SysML and UML Implementations

A. Requirement Diagram

In our requirement diagram, we have mentioned what are our requirements for the project. There are two main blocks: car motion and traffic management system. The main requirement for car motion is "the system shall be able to self drive through an traffic intersection without colluding with other vehicles and communicate and follow orders from the traffic management system.". There are some more sub-requirements linked to car motion, which are the requirements for detect cross-traffic, safety, priority determination and communication. When it comes to the traffic management system, the main requirement of it is "the system shall be able to manage the traffic flow". However, it also has its own sub-requirements responsible for decision-making, system integration and communication.



id="01"
text="the system shall be able to self drive through an traffic intersection without colluding with other vehicles and communicate and follow orders from the traffic management system"

id="03."
text=" The system shall be able to detect the vehicles approaching the intersection from all different ways "

id="05."
text="The system shall be able to evaluate the priority of various traffic objects according to predetermined rules or standards, taking into account elements like traffic laws, road conditions, and situational awareness."

id="06."
text="The system shall be able to communicate with traffic management system, to exchange information about their intentions and actions to enable coordinated cross-traffic management."

id="04."
text=" The system shall be able to perform the features such as emergency braking, collision avoidance and fail-safe mechanisms"

id="002",
text="The system should be able to manage the traffic flow "
text="
The system shall be able to send and receive orders to the vehicles"

id="07
text="The system shall be able to make real time decisions on how the vehicles to responce to the traffic situations"

id="08."
text="The system shall be able to seamlessly integrate with other vehicle's systems such as control, planning, ensure coordinate and effective management of cross traffic situation"

Fig. 1. Requirement Diagram

B. Use Case Diagram

In the diagram, we have an actor named "Car" that interacts with the system. The system is represented as the "Traffic Management System". The main use cases identified in the system are "Detect cross-traffic," "Determine Priority," and "Control Vehicle Movement".

The flow of interaction is shown through arrows between the user and the system. The user initiates the detection process by interacting with the "Detect cross-traffic" use case. The detected cross-traffic information is then sent to the "Determine Priority" use case for analyzing and determining the priority of vehicles. The determined priority information is then transmitted to the "Control Vehicle Movement" use case, which controls the movement of autonomous vehicles based on their priority.

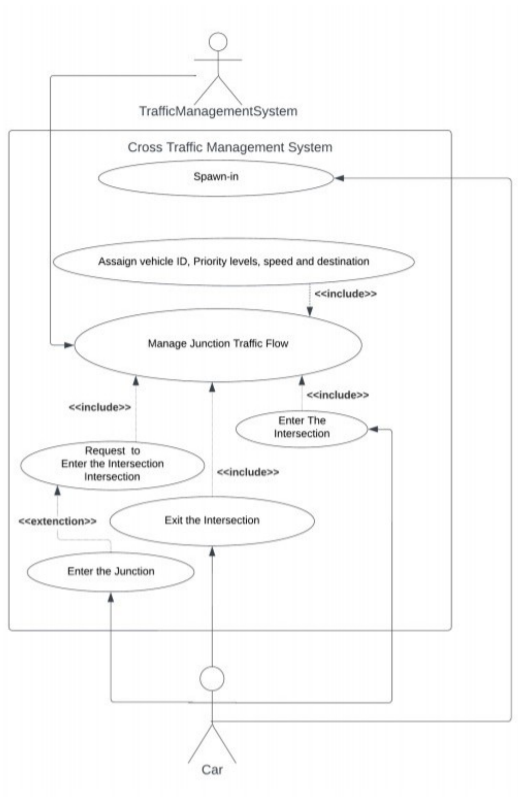

TrafficManagementSystem
Cross Traffic Management System
Spawn-in
Assaign vehicle ID, Priority levels, speed and destination
<<include>>
Manage Junction Traffic Flow
<<include>>
<<include>>
Enter The Intersection
Request to Enter the Intersection Intersection
<<include>>
<<extention>>
Exit the Intersection
Enter the Junction
Car

Fig. 2. Use Case Diagram

C. Activity and State Machine Diagrams

The scenario is in Fig. 3. and Fig. 4., one vehicle from any spawn location enters the junction. Before entering the intersection, the car is requesting to enter the intersection. If the intersection is clear, or if there is no collision detected, it can enter the junction. If the intersection is not clear, the vehicle should wait until the intersection is clear. Then, the vehicle can enter the intersection and steer in the desired direction. Then, it can exit the intersection and leave the junction and reach its desired destination.

D. Sequence Diagram

In Fig. 5., the sequence diagram explains the temporal behaviour of the vehicles in relation to different priority levels. The first vehicle has a priority value of 1, the second vehicle has a priority value of 2, third and fourth car has the same priority level of 3, but, the third vehicle has more speed than the fourth car. Therefore speeder car has a higher priority. Therefore car three enters the intersection before car four. After entering the intersection, each car will exit the intersection in the desired direction.

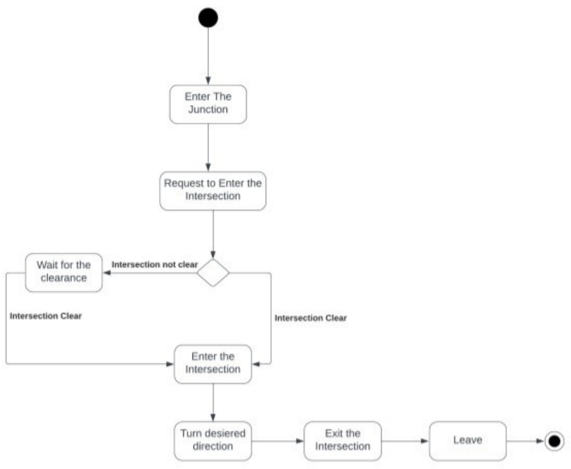


Fig. 3. Activity Diagram

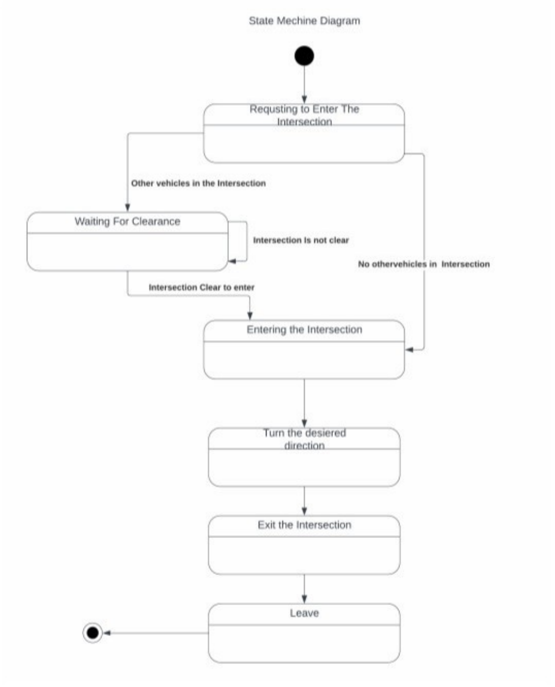

State Mechine Diagram
Requsting to Enter The Intersection
Other vehicles in the Intersection
Waiting For Clearance
Intersection is not clear
No othervehicles in Intersection
Intersection Clear to enter
Entering the Intersection
Turn the desiered direction
Exit the Intersection
Leave

Fig. 4. State Machine Diagram

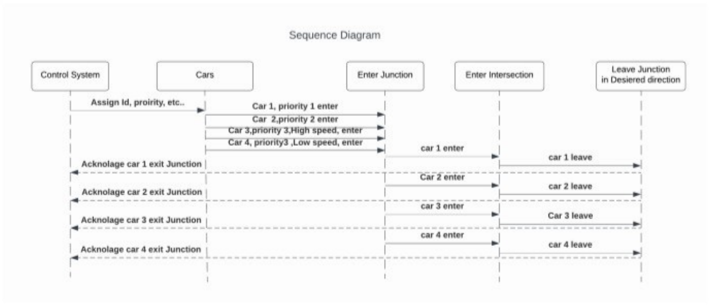

Sequence Diagram
Control System
Cars
Enter Junction
Enter Intersection
Leave Junction in Desiered direction
Assign Id, proirity, etc..
Car 1, priority 1 enter
Car 2,priority 2 enter
Car 3,priority 3,High speed, enter
Car 4, priority3 ,Low speed, enter
car 1 enter
car 1 leave
Acknolage car 1 exit Junction
Car 2 enter
car 2 leave
Acknolage car 2 exit Junction
car 3 enter
Car 3 leave
Acknolage car 3 exit Junction
car 4 enter
car 4 leave
Acknolage car 4 exit Junction

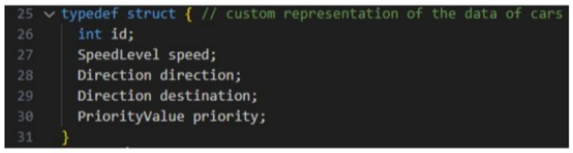
Fig. 5. Sequence Diagram

III. Software Implementation

A. Spawning mechanism in C

This section explains how the spawning mechanism is realised. The spawning mechanism is considered the starting point of the project and it is crucial because when the instance vehicle spawns in the testing environment, the initial location (North, East, South, West) where the vehicle spawns, the final destination(North to East, West to South, etc..) is predetermined. Furthermore Vehicle ID, the priority level should be already assigned. The vehicles have 3 priority levels, In order to make the algorithm simple only 3 constant speed levels were implemented. The basic functionality of this code is generating one random vehicle which contains the previously mentioned functionalities. each entity is assigned a randomised value. Finally, the code is able to re-create the desired amount of vehicles into the system. The most important sections of the c implementation is explained below.

- Structs:
  Structs are used to group logically related data items [1]. In Fig. 6., structs represent the data of the vehicle such as speed, spawn direction, final destination and assigned priority level. These structs can later be called from a function.



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

Fig. 6. Structs

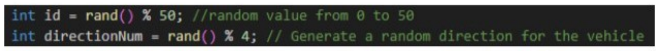
- Vehicle Spawn Function:
  In Fig. 7., considered the heart of the spawning function. This function is responsible for generating a car with a random ID, priority level speed value etc. Furthermore all the required `printf` functions in order to display the output is located inside this function.

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

Fig. 7. Vehicle Spawn Function

- Generating Randoms:
  Here in the Fig. 8., `rand()` is used to generate randomised integers [2]. Remainder (`%`) operator is used to divide and get the reminder from the previously generated random number. As an example, `rand()%3` means this function is dividing the output of rand() function by 3 and giving the reminder as 0 or 1 or 2 as output. Those outputs can be later triggered by the switch statements.

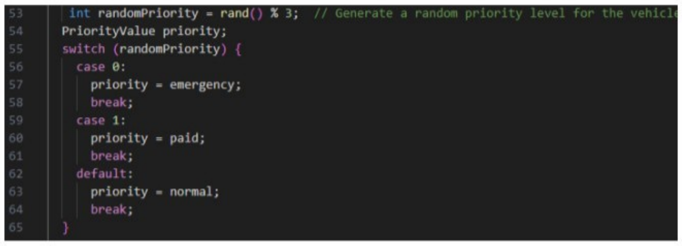
  Furthermore, this method is used for creating random integer values for generating random vehicle IDs, to generate random directions and speeds for the vehicles.



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

Fig. 8. Generating Randoms

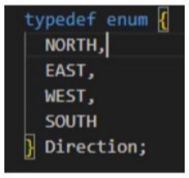
- Switch-statements:
  Switch-statements are used to assign previously generated random values to corresponding Enums. As examples the switch statement is used in assigning random spawn directions and final destinations, generate random priority and speed levels as well as printing out corresponding values as visual output, as shown in Fig. 9.



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

Fig. 9. Switch-statement

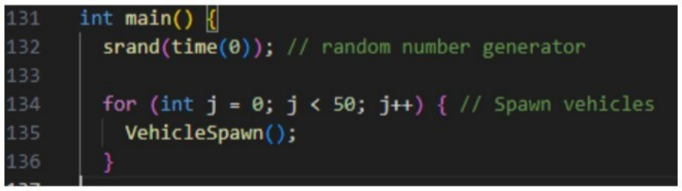
- Enums:
  In Fig.10., enums aid in categorizing, specifying, and differentiating cars by establishing precise criteria for each vehicle, [3].



```c
typedef enum {
  NORTH,
  EAST,
  WEST,
  SOUTH
} Direction;
```

Fig. 10. Enums

- Main Function:
  `srand(time(0))` function is used to seed the random number generator [2]. The reason time is set to zero is to retrieve the current time. The for loop is repeating the `vehicleSpawn()` function for the desired amount of times. Here in this case 50 times used, as shown in Fig. 11.



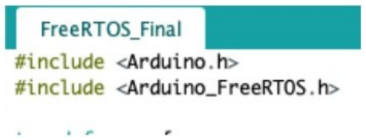
```c
int main() {
  srand(time(0)); // random number generator

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

Fig. 11. Main Function

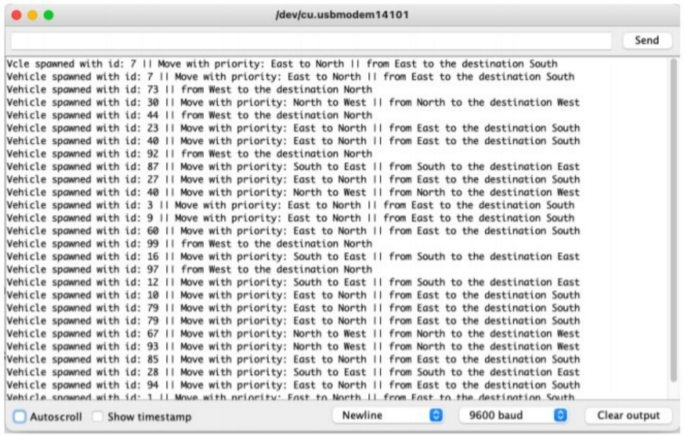
B. Spawning mechanism in FreeRTOS

According to the C code, we made the FreeRTOS implementation using Arduino Uno, as shown in Fig. 12.



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

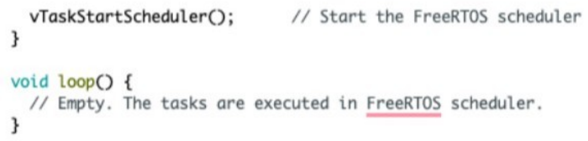
Fig. 12. Libraries For FreeRTOS



```
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
```

Fig. 13. Serial Monitor Simulation

The serial monitor visualization for our implementation can be found in Fig. 13.



```
  vTaskStartScheduler();      // Start the FreeRTOS scheduler
}

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

Fig. 14. Starting the FreeRTOS Scheduler

With reference to Fig. 14., there is an enabling of the FreeRTOS Scheduler in Arduino terminal.

IV. Scheduling Algorithm

Several scheduling algorithms were considered in this cross-traffic management scenario, and eventually, Short Remaining Time First (SRTF) scheduling algorithm was chosen. First Come First Serve (FCFS) was considered at the start of the project, due to the simple idea that any car that enters the junction will be crossing the intersection. Round robin scheduling algorithm was considered, but the fact that equal priority would be assigned did not make sense for our project [4]. However, the idea was more complex and not very adequate in this scenario, which is why SRTF was chosen at the end.

A. *Shortest Remaining Time First (SRTF)*

There are 2 assumptions in consideration of the SRTF scheduling algorithm for our project:

- All the cars at the junction/intersection **never** collide, due to their speed variations (i.e. speeds are similar, in junction/intersection)
- The cars will **never** stop in the junction/intersection, so the cars continuously move at a relatively very slow speed (i.e. the speeds of the vehicles decelerate at the intersection, but do not stop).

SRTF is an extension of Shortest Job First (SJF). Our project idea represents 3 factors, which make SRTF scheduling algorithm the most suitable one:

1) **Speed of the vehicles:** The speed of the vehicles before entering the junction, assigns priority to the vehicles. The faster the speed, the higher is the priority. This is from the value of 1-3, 1 being the lowest and 3 being the highest.
2) **Preemption:** In case of emergency vehicles, paid priority or any factor that can cause a preemption. Since SJF is non-preemptive while SRTF is preemptive, preemption occurs for such vehicles.
3) **Intersection matrix units:** Each vehicle is spawned from one of the North, South, East and West lanes. As each car has a destination lane, and U-turn is not allowed (e.g. North to North), the intersection matrix units can be calculated. The value is between 1-3: 1 being the lowest, 3 being the highest. The priority based on intersection matrix units is prioritized more than the priority based on the speed of the vehicles.

V. Visual Implementation

In order for a better understanding, a scenario was given with these cases, with references to Fig. 15.:

- **Car1:** from West to North; intersection matrix units is 3 units.
- **Car2:** from North to South; intersection matrix units is 2 units.
- **Car3:** from South to North; intersection matrix units is 2 units.
- **Car4:** from East to West; intersection matrix units is 3 units.

The priority, as mentioned previously, is given based on the intersection matrix units in this scenario. The assumption of each vehicle having same speeds must be considered, except for Car1 and Car4, which will be explained later. Car2 and Car3 move straight and there's no way these two vehicles will collide, as shown in Fig. 16. However, Cars 1 and 4 of higher intersection matrix units with lower priority, will be in collision (i.e. in light-blue stars), as shown in Fig. 17., after Cars 2 and 3 have left the junction/intersection. This means they need a separate priority allocation, which will be discussed in details with UPPAAL and VHDL implementations.

VI. UPPAAL Implementation

As shown in Fig. 18., global variables in the UPPAAL implementation are initialized with position at 1, which allows cars to all move together with synchronisation labels, *move14*

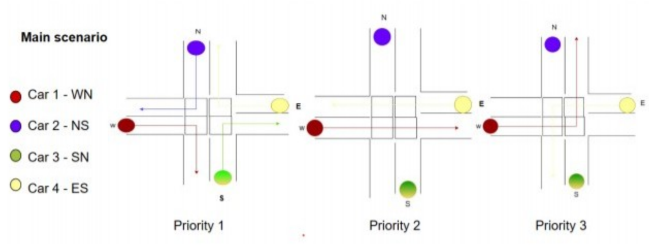


Fig. 15. Visual Implementation of Cross-Traffic Management.

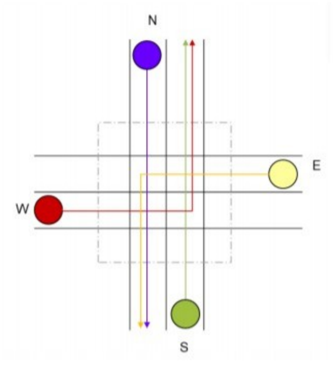


Fig. 16. Visual Implementation of Cars across the intersection.

and *move23* (i.e. cars are enterting the junction at position = 10). On the enabled transitions at the left-column, *move14* and *move23* move Cars 1, 4 and Cars 2, 3 enable state change in a pair in this implementation, from Spawn state. This was done such that Cars 1, 4 pair do not move before Cars 2, 3, with the SRTF algorithm activated when they are in the state enterJunction.

Cars 1 and 4 have different priorities assigned, despite having the same speed units. This is because, in real-life scenario, we know that no car will have the same speed and therefore, faster speed will be given a higher priority. This can also be randomised with the use of round-robin algorithm in the future, such that random assignment of priority can be given for such scenario to make a fair assignment of priority. Therefore, Cars 2 and 3 will leave the lanes first, then Car1 goes next with a temporary increase in priority, and Car3 will leave the last in this scenario.

As shown in Fig. 19., various guards and updates are associated with enterJ boolean value. This takes into account that only when the car enters the junction, priority is assigned to the car. It goes back to Idle state when all the cars leave the junction: that means, SRTF is no longer active when there are no cars in the junction.

VII. VHDL Implementation

Efficient cross-traffic management involves controlling the flow of traffic at intersections to ensure safe and efficient movement of vehicles. Hardware codesign refers to the process of designing both the hardware and software components of a system simultaneously, optimizing them together to meet specific requirements. ModelSim is a popular

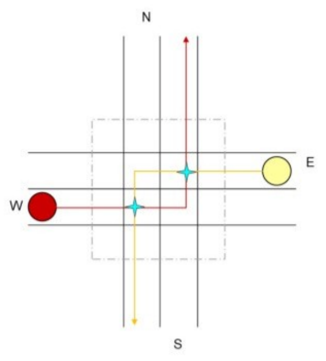


Fig. 17. Visual Implementation of Cars 1 and 4.

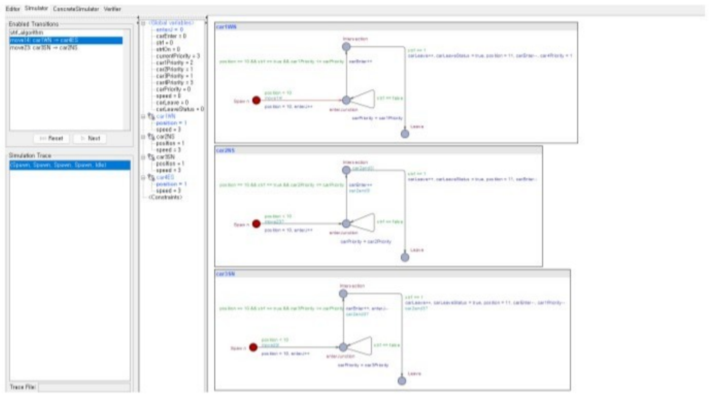


Fig. 18. UPPAAL Implementation of Cross-Traffic Management.

simulation tool used for digital and mixed-signal designs, shown in Fig. 20.

This technical document's major goal is to give readers a thorough grasp of the VHDL design for cross-traffic management, with a particular emphasis on the case of the shortest time to remain first with priority. The purpose of the document is to explain the VHDL design's architecture, design considerations, algorithms, and implementation specifics. It acts as a reference for programmers, engineers, and academics interested in using VHDL to construct cross-traffic management systems.

The whole development process, from the high-level system architecture to the intricate component designs, is covered by the scope of this document. It covers the cross-traffic management system's VHDL implementation, simulation, and performance assessment. The document also emphasizes the process for prioritization and how it affects traffic flow optimization.

Entity declaration in VHDL for a design entity, is named "HW-SW-CODE". The entity has several input and output ports, including clock (clk-in), reset (reset), car sensors (car1, car2, car3, car4), car outputs (car1-out, car2-out, car3-out, car4-out), lane inputs (north-lane, west-lane, east-lane, south-lane), and lane outputs (north-out, west-out, east-out, south-

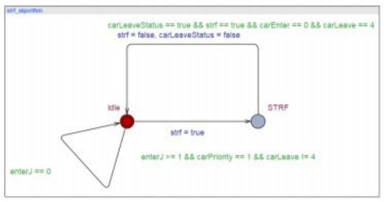

srtf_algorithm
carLeaveStatus == true && strf == true && carEnter == 0 && carLeave == 4
strf = false, carLeaveStatus = false
Idle
STRF
strf = true
enterJ >= 1 && carPriority == 1 && carLeave != 4
enterJ == 0

Fig. 19. UPPAAL Implementation of SRTF.

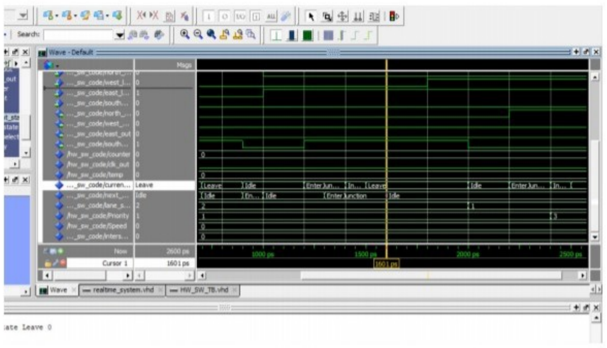


Fig. 20. ModelSim Simulation

out), as shown in Fig. 21.

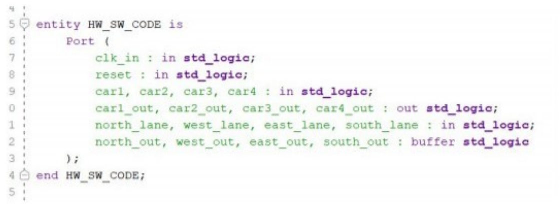
From Fig. 20, these signals and types define the state, lane selection, priority, speed, and intersection information for a vehicle in the cross-traffic management system.

- StateType: Enumeration type representing vehicle states (Idle, EnterJunction, Intersection, Leave).
- Current-state, next-state: Signals storing the current and next state of the vehicle.
- lane-selected: Signal indicating the selected lane (0 to 4).
- Priority: Signal representing the priority level of the vehicle (0 to 4).
- Speed: Signal indicating the speed level of the vehicle (0 to 3).
- intersectionMatrix: Signal representing the availability of lanes in the intersection (0 to 3).

Process in VHDL that is sensitive to changes in the current-state, car1, car2, car3, and car4 signals. The process is responsible for assigning the value of current-state to next-state.

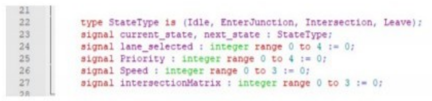
- The process statement specifies that the process is sensitive to changes in the signals listed within the parentheses.
- The begin keyword marks the start of the process block.
- The line next-state ¡= current-state; assigns the value of current-state to next-state.

- The value of current-state is assigned to next-state.
- Based on the value of current-state, the process checks the values of car1, car2, car3, and car4 signals, as well as the corresponding lane signals (west-lane, north-lane, south-lane, east-lane).
- If a car signal (car1, car2, car3, or car4) is high ('1') and the corresponding lane is available, the lane-selected signal is set to the corresponding lane number, and next-state is updated to enterJunction.



```
entity HW_SW_CODE is
    Port (
        clk_in : in std_logic;
        reset : in std_logic;
        car1, car2, car3, car4 : in std_logic;
        car1_out, car2_out, car3_out, car4_out : out std_logic;
        north_lane, west_lane, east_lane, south_lane : in std_logic;
        north_out, west_out, east_out, south_out : buffer std_logic
    );
end HW_SW_CODE;
```

Fig. 21. Inputs and Outputs Declarations



```
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;
```

Fig. 22. Signal Declaration

The cross-traffic management system prioritizes cars according to their remaining time in order to speed up the passage of those with urgent travel requirements, such as emergency vehicles or time-sensitive deliveries. The efficiency of the overall traffic flow is increased, and vehicles traveling closer to their destinations experience less delays.

The cross-traffic management system optimizes traffic flow by combining the shortest time to remain first scenario with priority-based considerations to achieve both efficiency and fairness in handling the needs of various vehicles.

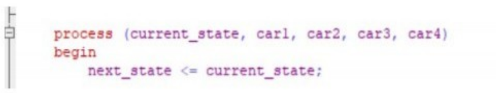
VIII. Evaluation & Conclusion

Despite the fact that there are priority allocations based on the speed of the vehicles before entering the junction, intersection matrix units and preemptive nature of SRTF algorithm, there are some refinements to be made. Combinations of scheduling algorithms, such as FCFS and round-robin algorithms, can be employed with SRTF scheduling algorithm [5].

In order to improve the system, the crossroad should also take into account of different scenarios: for instance, when there is a road blockage or at different times, certain priorities should take over (i.e. time for schools/kindergartens leads to speed control). Then, the intersection matrix units can be divided into further units such that it can predict the car with larger size, would take up more space. This will lead to better prioritization, rather than current 2x2 units square of intersection matrix units (i.e. 1000x1000 units square).

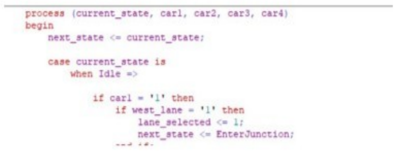
VHDL function can be further improved with better sensors and networking. At the moment, the V2I network has been established in the project, but V2V networks can also be established for better prioritisation.

In conclusion, there was an overall coverage of SysML, UML diagrams for the efficient cross-traffic management, and the SRTF scheduling algorithm was employed. These were successfully implemented with freeRTOS/VHDL fucntions for hardware, and UPPAAL/C code for software implementations. In the end, the project was able to display satisfactory results



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

Fig. 23. Process in VHDL



```
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;
```

Fig. 24. State Change

in accommodating the SRTF algorithm that is based on the factors for priority allocation.

IX. Contributions

- Ravindu:
  FreeRTOS implementation. Requirement Diagram, use case diagram, helping with implementation of the scheduling algorithm
- Kajeepan:
  Project: SRTF implementation in VHDL code and ModelSim simulation
  Paper: Implementation of Hardware software Codesign and explanation of how its implemented according to the Algorithm Proposed.
- Heansuh: Scheduling algorithm choices and evaluation, final UPPAAL model implementation, ideas for implementations, evaluation and conclusion.
- Lochana:
  Project: Spawning Algorithm: implementation in c, State-machine diagram, use case diagram, final refinements of the diagrams.
  Paper: Explanation of activity diagram and sequence diagram, explanation of spawning mechanism in C.

References

[1] GeeksforGeeks, "C structures," 2023. [Online]. Available: https://www.geeksforgeeks.org/structures-c/

[2] S. Overflow, "C++ random int function," https://stackoverflow.com/questions/4982026/c-random-int-function, 2011, accessed:June 9, 2023.

[3] H.-T. Geek, "What are enums (enumerated types) in programming, and why are they useful?" https://www.howtogeek.com/devops/what-are-enums-enumerated-types-in-programming-and-why-are-they-useful/, 2022, accessed on: June 9 2023.

[4] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. Pearson Education, 2015.

[5] "Comparison of cpu scheduling algorithms: Fcfs, sjf, srtf, round robin, priority based, and multilevel queuing," *IEEE Xplore*, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9929533/

X. Affidavit

We hereby confirm that we have written this paper independently and have not used any sources or aids other than those indicated. All statements taken from other sources in wording or sense are clearly marked. Furthermore, we assure that this paper has not been part of a course or examination in the same or a similar version.

Umaibalan Kajeepan
Name,Vorname
Last Name, First Name

Lippstadt 03/07/2023
Ort,Datum
Location, Date

[illegible]
Unterschrift
Signature

Heansuh Lee
Name,Vorname
Last Name, First Name

Lippstadt 03/07/2023
Ort,Datum
Location, Date

[illegible]
Unterschrift
Signature

Abhayawardana Lochana
Name,Vorname
Last Name, First Name

Lippstadt 03/07/2023
Ort,Datum
Location, Date

[illegible]
Unterschrift
Signature

Athukorala Ravindu
Name,Vorname
Last Name, First Name

Lippstadt 03/07/2023
Ort,Datum
Location, Date

[illegible]
Unterschrift
Signature