Optimized Real-time Cross Traffic Management for Autonomous Vehicles

1st Rubayet Kamal
Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
rubayet.kamal@stud.hshl.de

2nd Moiz Zaheer Malik
Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
moiz-zaheer.malik@stud.hshl.de

3rd Mofifoluwa Akinwande
Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
mofifoluwa-ipadeola.akinwande@stud.hshl.de

4th Jonathan Bahry
Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
jonathan.bahry@stud.hshl.de

***Abstract*—This paper presents a comprehensive technical documentation of a centralized traffic controller for autonomous vehicles designed to manage vehicle crossing through intersections. Motivated by the inefficiencies of traditional traffic light systems and the growing integration of autonomous vehicles. The system employs First-Come, First-Serve (FCFS) queueing model and prioritizes pedestrian safety through sensor based detection. The documentation covers the motivation behind the project, the systems engineering approach, UPAAL for timed automata and behavioral modeling. The software is implemented using FreeRTOS to ensure deterministic task scheduling, and the system is validated through hardware using VHDL.**

***Index Terms*—FreeRTOS, FCFS, UPAAL, ModelSim, VHDL**

I. Introduction and Requirements

As cities grow and traffic volumes rise, traditional traffic light systems, while functional, these traffic lights are inefficient and are becoming more insufficient. These systems are leading to reduced flow efficiency, fuel waste and unnecessary delays. Recognizing these limitations, and with the growing relevance of autonomous vehicles.We set out to design an efficient, intelligent traffic control system.

Initially, our approach involved an upgrade or an advanced version of the current traffic light system. Scaling the current system offered some improvements, but it still fell short in addressing the demands of autonomous, coordinated traffic behavior.We then explored a decentralized traffic management approach, relying solely on vehicle-to-vehicle (V2V) communication. Although promising in theory, this concept lacked robustness and consistency leading to many collisions. Through further research, we identified a new approach, we identified a more effective solution. A centralized traffic controller that communicates directly with the vehicles approaching an intersection. In this system, vehicles broadcasts their intent to cross the intersection, allowing the traffic controller to manage traffic on a First Come First Serve (FCFS) queuing model. Pedestrians are always given priority, ensuring that their right of way is fully respected. This system eliminates the need for traditional traffic light and enables more efficient traffic coordination.

The development of our traffic controller system was guided by a comprehensive set of requirements, meticulously captured and organized using SysML requirement diagrams. This approach provides a clear visualization of both functional and non functional requirements. At the core of our system lies the fundamental requirement for traffic controller system (REQ-TRC-001). This overarching capability is supported by four primary functional requirements: a pedestrian detection system (REQ-TRC-001.1), pedestrian handling (REQ-TRC-001.2), arbiter communication handling (REQ-TRC-001.3) and FCFS queuing (REQ-TRC-001.4). The pedestrian detection system represents a critical safety feature of our system. This requirement specifies that the system must be able to detect pedestrians at a safe distance giving enough time for the intended vehicles to come to a complete stop. Complementing this, the pedestrian handling requirement ensures that upon the detection of a pedestrian, the system should broadcast this information to all vehicles within the intersection, maintaining situational awareness and ensuring safe crossing. The arbiter communication handling dictates that all communications and decision making is made by the controller itself to avoid the possibility of any collision. The FCFS queuing requirement enforces that vehicles follow a strict FCFS order, continuously updating their queue status and coordinating with the controller to maintain an efficient traffic flow. To ensure the system reliability and performance, we have also included a constraint related to queuing. This stipulates that under all circumstances, the system should fulfill the FCFS queuing even in the presence of emergency vehicles.

II. Modelling with SysML

To design and analyze our **autonomous traffic controller system**, we employed **Systems Modeling Language (SysML)**. The system is responsible for managing the flow of

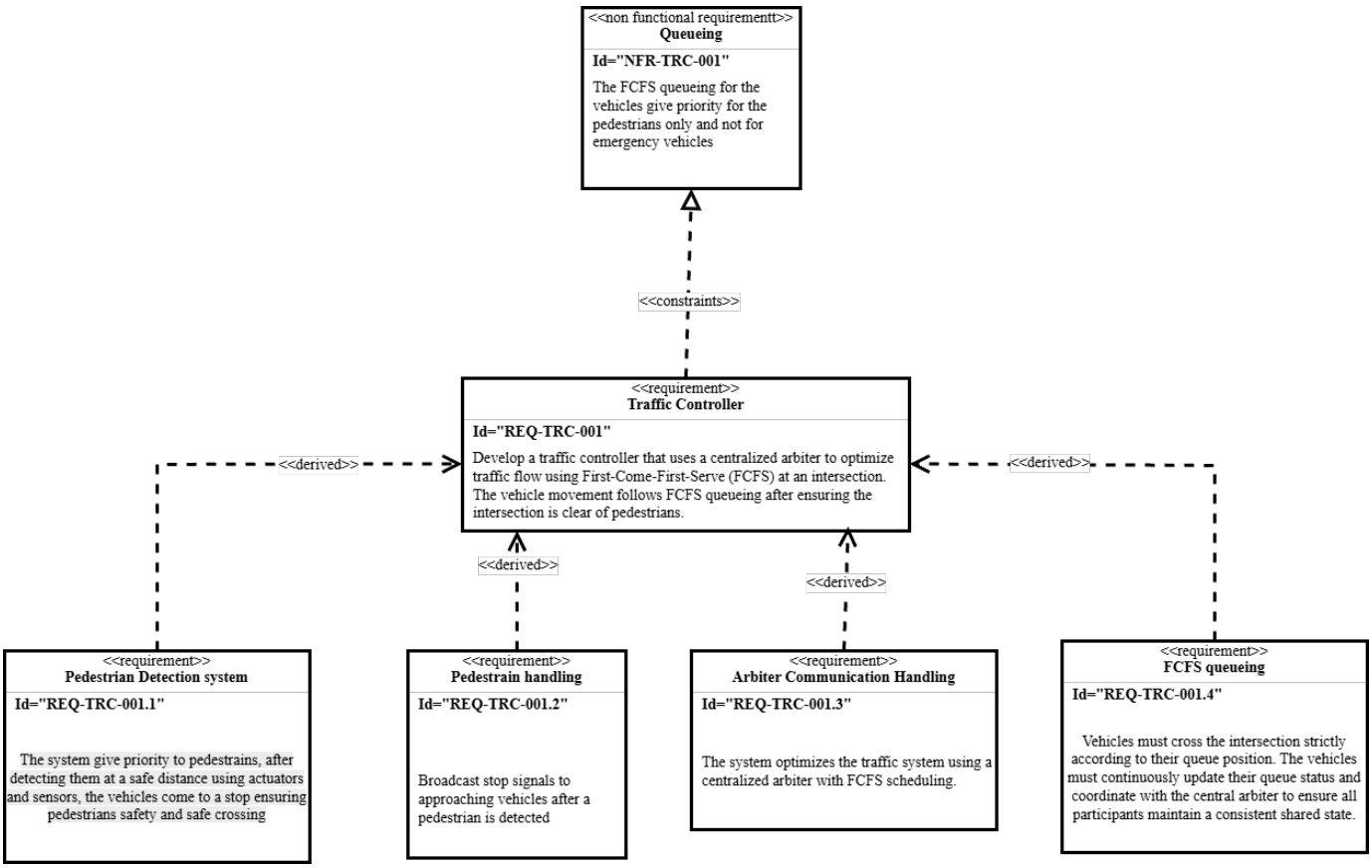

Fig. 1. Requirements Diagram

autonomous vehicles at intersections using a **first-come, first-served queuing mechanism**, while also accounting for **pedestrian detection via sensors and actuators**. Communication between the traffic controller and vehicles ensures coordinated movement, with pedestrian presence acting as a high-priority interrupt.

The following SysML diagrams illustrate key behavioral and interaction aspects of the system:

A. State Machine Diagram

The behavior of our Traffic Controller is comprehensively modeled using SysML State Machine Diagram that can be seen in Fig. 2. This diagram provides a detailed representation of the system's operational states and the transitions between them, offering insight into the decision-making processes and responses to various stimuli during operation. The system begins in the ”ON” state. From this initial state, the system transitions into the ”Idle” state. This state represents the default behavior of autonomous vehicles, where the vehicle is just following the designated path. While in the ”Idle” state, the vehicle is continuously monitoring the environment through various sensors, once the sensors detect an intersection, the system transitions into the ”Approaching” state. During this state, the vehicle transmits the request to pass the intersection. The traffic controller will check the vehicle position. This decision points to two possible states: ”Enter” or ”Queued”. The choice between these states is determined by the position of the autonomous vehicle in the queue. If the position of the vehicle is not the first, the system will enter the ”Queued” state. In this state, the vehicle will wait until it becomes the first in the queue. If the corresponding position is the first, the system will proceed to the ”Enter” state. During this state, the system will check if the route is safe for the vehicle to proceed without causing any collisions. By doing so, the system will either go into ”Wait for car passage” state or ”Ready” state. The traffic controller will check: if the previous car is in the way of the next vehicle leading to a collision, the system will go into ”Wait for car passage” state. The vehicle will remain in this state until the condition is satisfied and the path is clear. Once safe, the system will move to the ”Ready” state. In this current state, the vehicle is ready to exit the intersection, however, in our system pedestrians are given priority, in order for the vehicle to pass, the system has to clear that there are no pedestrians in the designated threshold and the car is able to pass safely. Otherwise, the vehicle will wait in the ”Wait for pedestrian to pass” state until the pedestrian has passed safely. Now, the system will transition into the ”Cross” state, where the vehicle proceeds through the intersection. Upon completing the crossing, the system reaches the final state, ”EXIT”, in which the vehicle detaches

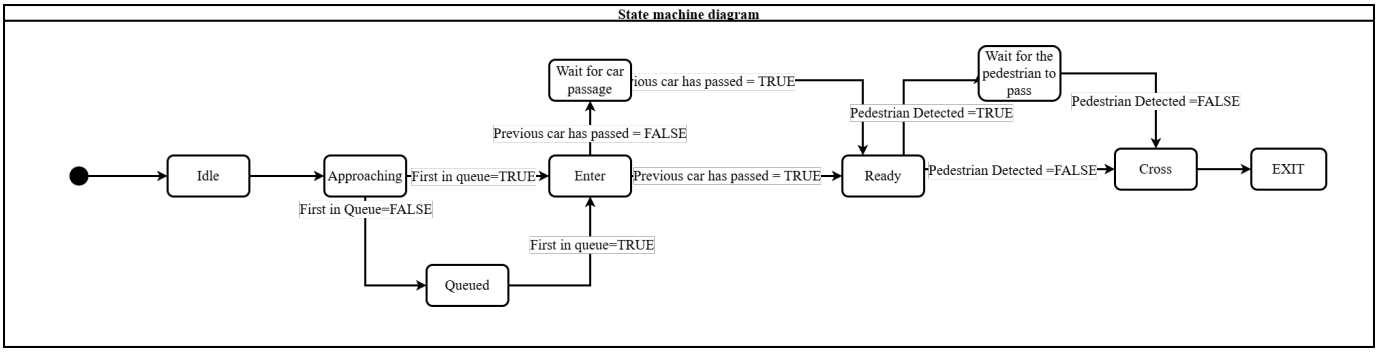

Fig. 2. State Machine Diagram

itself from the traffic control system, allowing the next vehicle to begin its process. This state machine model captures the dynamic nature of our traffic controller system. It illustrates how the system continuously assesses its environments, makes decisions based on sensors and communication inputs, and adapts its behavior accordingly. The clear definitions of states and transitions ensures the system can handle various scenarios it may encounter leading to optimal coordination, safety, and efficiency in intersection management.

B. Use Case Diagram

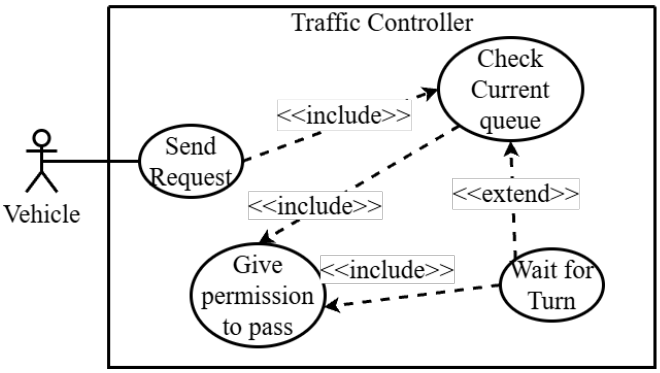

Fig. 3. Use Case Diagram

The Use Case Diagram in Fig. 3 illustrates the interactions between the autonomous vehicle and the traffic controller. The primary actor in this model is the Autonomous Vehicle, which interacts with the controller to navigate the intersection safely. The vehicle initiates the process by sending a request once the vehicle enters the detection range of the controller. This request triggers the controller to check the current queue. If the vehicle is not the first in line, it must wait until it becomes first, that is an extended behavior that is only activated when the vehicle is not the first in the queue meaning this behavior only happen when a this condition is satisfied. Once the vehicle reaches the front of the queue and the path is clear, the system gives permission to pass, allowing the vehicle to safely proceed through the intersection. This diagram effectively captures the communication and decision making processes between the vehicles and the traffic controller.

C. Activity Diagram

The Activity Diagram for Normal Mode illustrates the procedural flow of the traffic controller system as it manages vehicle movement through an intersection. The process begins when a vehicle approaches the intersection and broadcasts its presence. The system then gathers information about surrounding vehicles to determine its relative position in the queue. If the vehicle is not first, it waits in the queue until its turn arrives. Once it is first in line, the system checks for the presence of pedestrians within the intersection zone. If a pedestrian is detected, the vehicle remains in a waiting state until the pedestrian has safely passed. When the path is clear, the vehicle proceeds to cross the intersection and, upon completion, exits the system, allowing the next vehicle to be processed. This diagram effectively captures the logical flow and decision making steps that ensure safe and coordinated traffic movement.

III. UPPAAL Modelling for Deadlock Verification

Traditional traffic intersections use traffic lights to regulate flow. This simulation explores an alternative system: one where cars communicate with each other and with pedestrian indicators to navigate intersections safely. Using UPPAAL, we model timed automata for cars and pedestrians to show how such a system can work. The intersection has four sides—North, South, East, and West—and all sides follow the same behavior. For simplicity, the South side is used to describe the core logic.

A. Intersection Lane Structure and Car Templates

Each side of the intersection has three lanes: left, middle, and right. The left lane is primarily for incoming traffic and is not used actively in this simulation. The simulation focuses on two templates from the South side:

- `CarSM (Car South Middle)`
- `CarSR (Car South Right)`

Each car template includes states such as *Arrived*, *ZebraCrossing*, *GoStraight*, *LeftTurn*, *Stop*, and *Moving*.

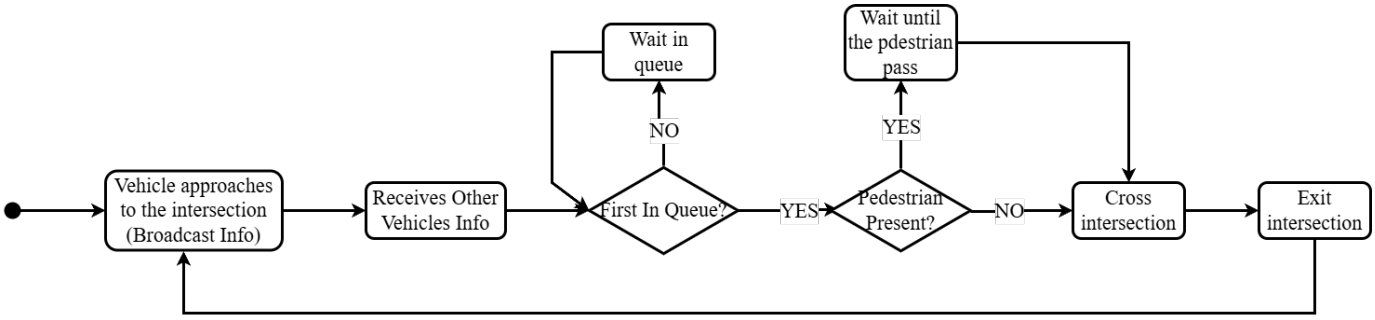

Fig. 4. Activity Diagram for normal mode

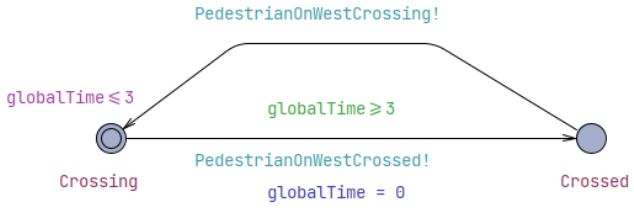

Fig. 5. Pedestrian on North-South Facing On West

B. CarSM Behavior

When a car enters via the South middle lane (CarSM), it checks if a pedestrian is crossing the South zebra crossing. If a pedestrian is crossing, it waits. If not, it moves on. The car then randomly decides whether to go straight (toward North) or turn left (toward West). If it turns, it stops again at the West crossing and checks for pedestrians before moving.

C. CarSR Behavior

The South right lane (CarSR) car always turns right toward East. It first checks the South crossing for pedestrians. After that, it stops again at the East crossing and only continues if no pedestrian is crossing. This flow ensures safe interaction between cars and pedestrians.

D. Pedestrian Modeling

There are four pedestrian templates in the model:

- `PedestrianNorthSouthFacingOnWest`
- `PedestrianEastWestFacingOnNorth`
- `PedestrianEastWestFacingOnSouth`
- `PedestrianNorthSouthFacingOnEast`

Each pedestrian automaton has two states:

- `Crossing`
- `Crossed`

A global timer is implemented via a separate template called `Ticker`, which controls when each pedestrian begins and ends crossing.

The crossing durations are as follows:

- South pedestrians: 10 time units
- East pedestrians: 7 time units
- West pedestrians: 3 time units
- North pedestrians: 14 time units

When a pedestrian starts crossing, it emits a synchronization signal such as:

```
PedestrianOnSouthCrossing!
```

After the crossing is complete, it emits another signal:

```
PedestrianOnSouthCrossed!
```

These synchronization channels are used by car templates to decide whether to wait or proceed, based on pedestrian activity at the crossings.

E. System Integration

The system ties all templates together using synchronization channels and guards. There are no random clocks; all timing is deterministic through the globalTime and anotherglobalTime counters. All car templates (CarSM, CarSR, etc.) and pedestrian templates are declared in the system block and simulate real-time movement across the intersection.

IV. Hardware-Software Co-Design Implementation

The purpose of this section is to discuss the hardware-software co-design part of the implementation. Therefore, obviously the section are divided into two parts: FreeRTOS (Software) in subsection IV-A and Hardware Realisation in IV-B.

A. FreeRTOS

To realize the system in a simulation format with real-time capabilties, we employed a real time operating system (RTOS), in this case the open-source FreeRTOS library. In FreeRTOS, just like in real systems, functions that have to meet a deadline are called tasks. Therefore, to realize a working simulation, we created many critical tasks. This section focuses on some of these important tasks and our solution to resolving conflicts at critical parts of the system.

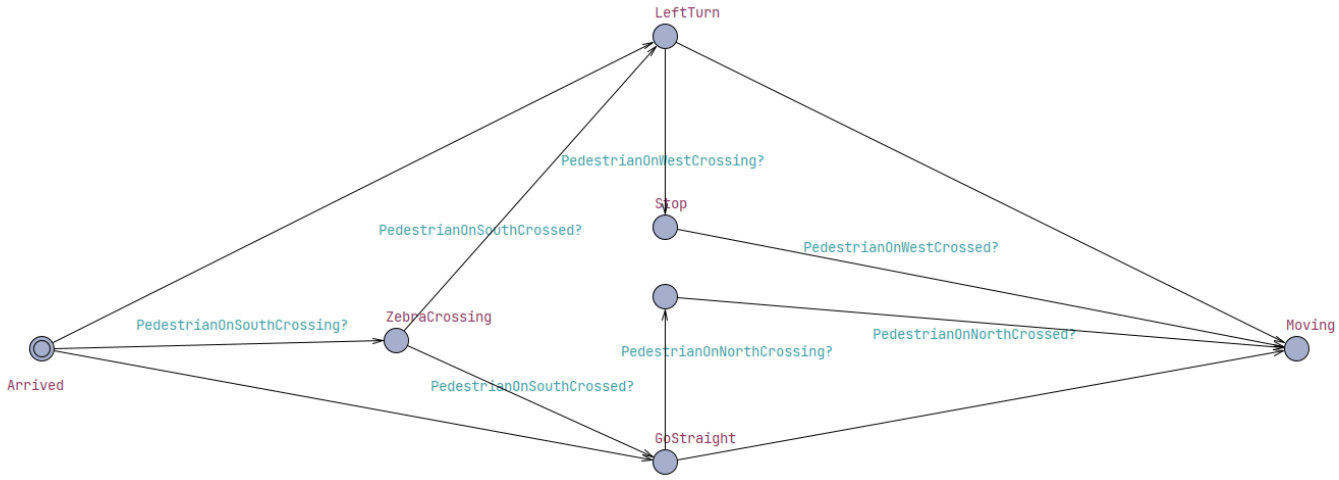

Fig. 6. Model for Cars on Middle Lane of South

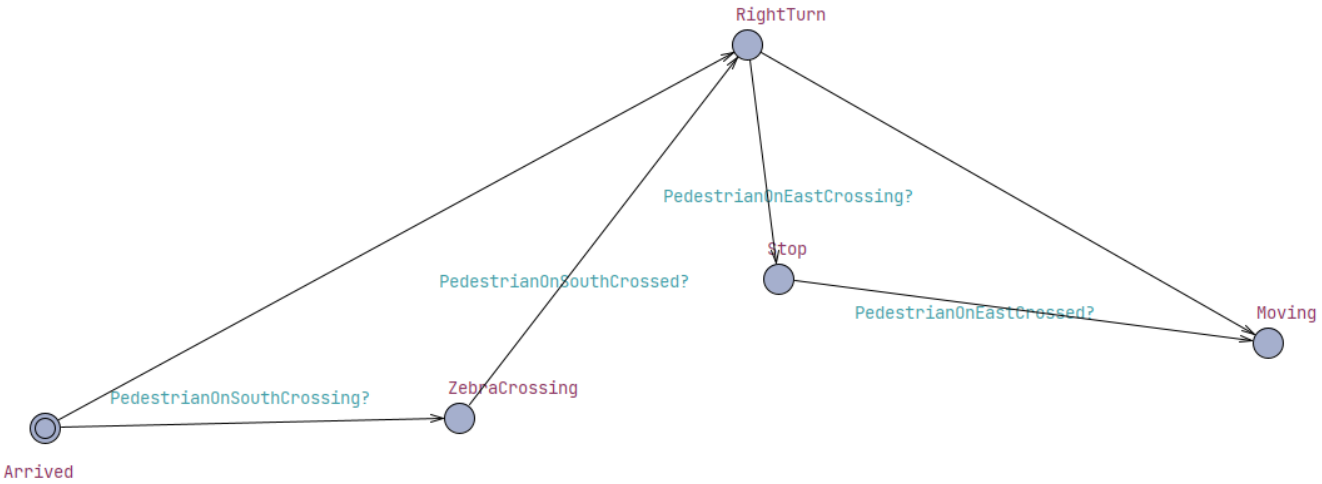

Fig. 7. Model for Cars on Right Side of South

1) vCarGeneratorTask(): This task is responsible for the safe spawning of cars from any of the four lanes. It employs efficient error handling to make sure cars are properly allocated and initialized. A state machine diagram illustrating the flow of this task can be seen in 8.

2) vCarTask(): After the cars have been generated, the Car structure with parameters as seen in fig. 9 is activated. In this task, all varaibles in the struct are initialized. Every action of the car from entering the simulation area, to reaching the critical section, to exiting the simulation is covered in this task and is logged accordingly. Important FreeRTOS features like queues for efficiently inter-task messaging and semaphores to ensure efficient resource sharing of the scheduler time are used as well. A state machine diagram detailing the flow of the task can be seen in fig. 10.

3) log__event(): To visualize the system using the pygame library in Python, it was important to efficiently log all actions of the car from the earlier discussed car task. This task was in charge of this function and creating a CSV log file whose contents were parsed into a Python script to visualize our system.

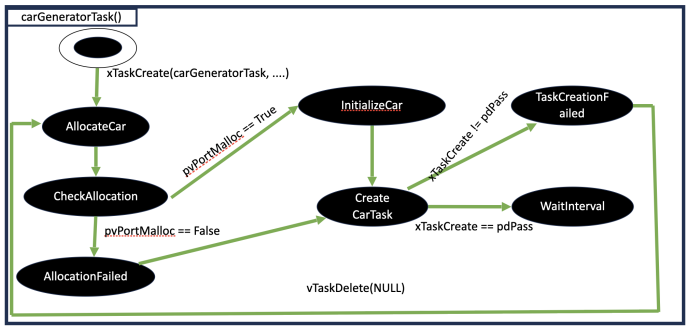

Fig. 8. State machine for Car generator task

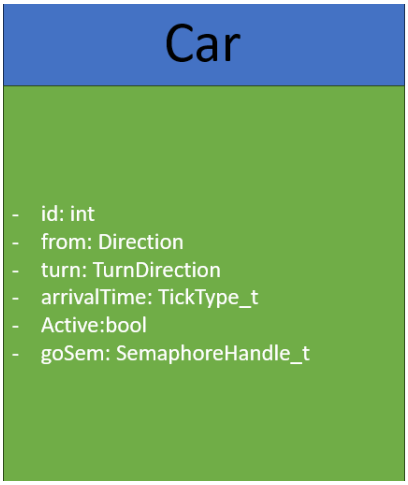

Fig. 9. Class Diagram showing the Car struct

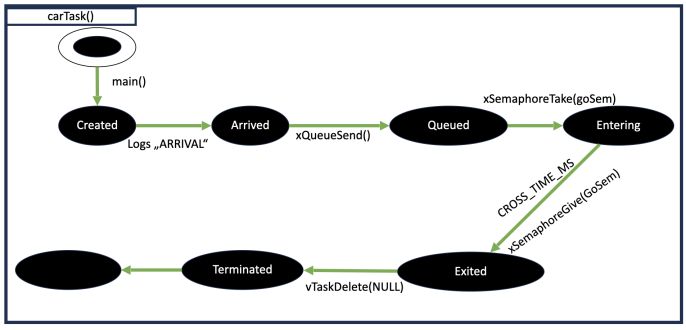

Fig. 10. State machine for Car generator task

4) Conflicts at critical section: It was important to define what part of this system are susceptible to collisions. We call this the critical section. In this case, we define the Intersection as the critical section, because it is a meeting point for all the cars coming from different lanes. There are different approaches to solving this conflict. We decided to use a First Come First Serve (FCFS) algorithm. While seemingly trivial, it ensures proper queuing and ensures fairness for all cars in the system.

To implement this in our FreeRTOS system, we created an Arbiter task (varbiterTask()). The main responsibility of this task was to ”peek” at the head of all four queues, check for the car with the earliest arrival time and release (through a mutual exclusive binary semaphore) that car and only that car to the Intersection. This approach ensures smooth scheduling of the cars and prevents collisions. The sequence of operations can be seen in 11.

B. Hardware Realisation

As traffic control systems require deterministic behaviour, the arbiter logic is realised in hardware. Additionally, the inputs to the arbiter are also realised in a separate hardware module called requesting detection. This is discussed in subsection IV-B1 and IV-B2.

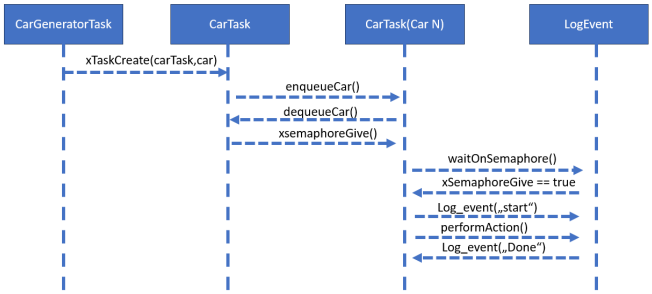

Fig. 11. Sequence diagram for Car generator task

1) Vehicle Arbiter Module: The arbiter controls traffic flow from four directions (North, East, South, and West) by granting access to one direction at a time, based on a simple state machine. It rotates access to each direction in a round-robin fashion , regardless of whether a vehicle is actually waiting. On every rising clock edge, the state rotates to the next one. It doesn't check if a car is waiting as it blindly cycles. Each grant signal is 1 only when the machine state matches that of the request signal resulting in fair rotation. This is designed in VHDL and the block diagram is generated in Vivado as shown in fig, 12.

2) Request Detection Logic: The RequestDetector is designed to monitor the presence of vehicles at each road (North, East, South, West) and generate request signals so that the arbiter can use to decide who should get access. This would typically come from sensors that detect cars approaching the intersection. On each rising edge of the clock the request signal is copied directly from the sensor signal. If a sensor detects a vehicle ,for example, sense_N = '1', then req_N becomes '1'. The schematic can be seen in fig. 13.

V. Outcome

Several challenges were faced in mapping from model created in UPPAAL to FreeRTOS implementation. Because of this some cars were stuck in deadlocks. This led to changing the final development from 3 lanes per road to 2 lanes per road as shown in fig. 14. The complete documentation can be found in https://github.com/RubayetKamal/SS2025_Embedded_Electronic_Enginering_A_Lab_Team1

VI. Conclusion

In this paper, a centralized, real-time cross-traffic management system tailored for autonomous vehicles at intersections is presented. Motivated by the disadvantages of traditional traffic light systems, centralized controller is implemented that enforces a First-Come, First-Serve (FCFS) queuing mechanism while prioritizing pedestrian safety. Using SysML and UPPAAL, system behavior has been modelled and verified deadlock-free execution. Implementation was realized through hardware-software co-design, utilizing FreeRTOS for real-time task scheduling and VHDL for deterministic arbitration. Despite the success of the system, several challenges were

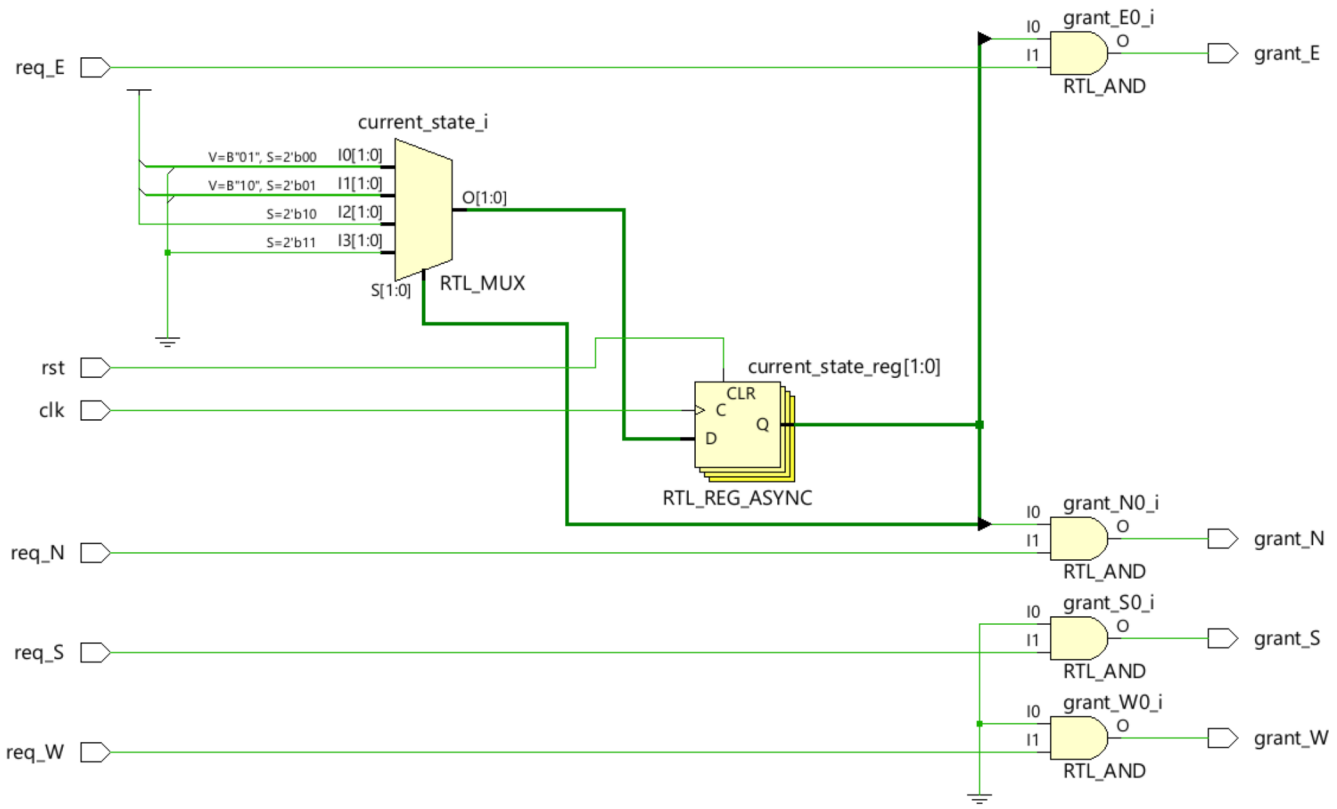

Fig. 12. Schematic for Arbiter Module

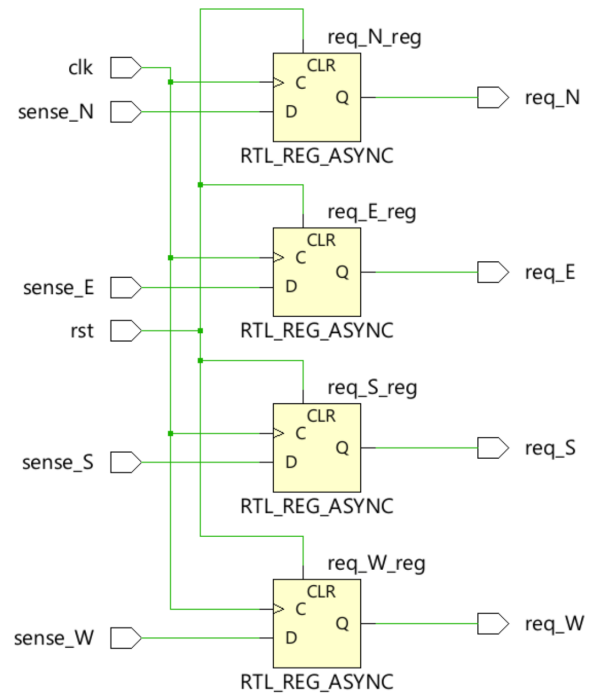

Fig. 13. Module modelling a sensor to detect cars approaching.

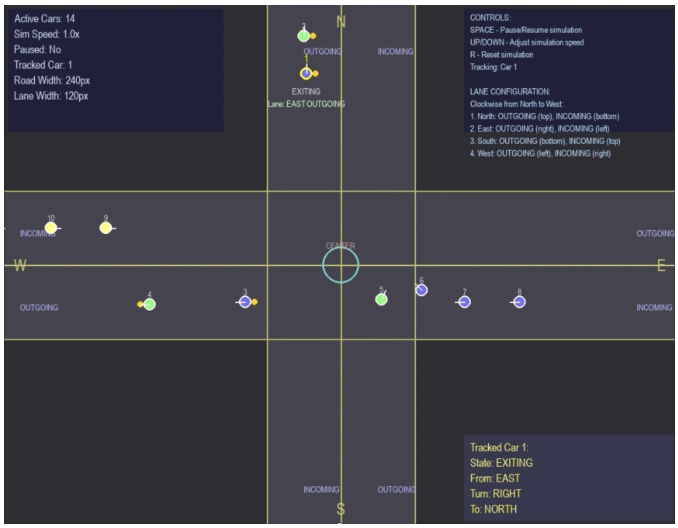

Fig. 14. Output Visualised.

encountered, particularly in aligning the UPPAAL model with the FreeRTOS task structure, which led to simplifications in lane design.

The system demonstrates effective and safe traffic coordination for autonomous vehicles in simulation. For future work, the proposall is on extending the system to handle

dynamic prioritization (e.g., emergency vehicles), supporting more complex intersection layouts, and integrating machine learning techniques for predictive traffic flow optimization. Moreover, a full-scale hardware prototype can be developed and tested in a physical environment to validate the robustness and scalability of the approach.

ACKNOWLEDGMENT

This project has been possible due to the constant feedback of Prof. Dr. Henkler throughout the course. His expertise and guidance of FreeRTOS contributed to completion of the project.

REFERENCES

[1] Vivado Design Suite, Xilinx Inc. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html
[2] ModelSim Simulation Tool, Mentor Graphics. [Online]. Available: https://www.mentor.com/products/fv/modelsim/
[3] FreeRTOS Real-Time Operating System, Real Time Engineers Ltd. [Online]. Available: https://www.freertos.org