Autonomous Traffic Intersection Controller

Electronic Engineering Lab A

Summer Term 2023

1st Jaouaher Belgacem
dept. Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
jaouaher.belgacem@stud.hshl.de

2nd Neaz Mahmud
dept. Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
Neaz.mahmud@stud.hshl.de

3rd Nirojan Navaratnarajah
dept. Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
nirojan.navaratnarajah@stud.hshl.de

***Abstract*—FHWA conducted a study in the united states about car crashes related to intersections in the year 2000 and they found that 1.3 million out of a total equal to 2.8 million hits occurred at the level of signalized intersections which lead to more than 445,000 injuries [1]. Hence, several pieces of research have been dedicated to Intelligent Transportation Systems (ITS) and more specifically to Intelligent intersection management systems [2] [3]. Thus, evolving this field became crucial with the exponential growth of autonomous vehicles in the last couple of years as they will ensure, safer driving that conforms the traffic rules [4], and the establishment of an intelligent intersection controller will enable autonomous cars to drive efficiently.**

In this paper, we will present our solution for an autonomous traffic intersection controller.

***Index Terms*—FreeRTOS, VHDL, Intersection management, Intelligent intersection controller, V2I, autonomous vehicles.**

I. Introduction

Despite, the integrated functionalities of transportation infrastructure, the installed traffic controllers are unable to reduce traffic problems such as stop delays, traffic congestion due to the timing, light switching flows that increase the chances of accidents, and fuel consumption which leads to environmental damage. However, Autonomous vehicles can eliminate most of these issues. Therefore, the installation of autonomous intersection managers will unlock the full potential of self-driving vehicles and make a beneficial impact on our transportation system [4] [5].

In this work, we are proposing an autonomous traffic intersection controller that is Vehicle to Infrastructure (V2I) communication-based. We chose V2I over the Vehicle to Vehicle (V2V) communication model due to the high volume of data transmission which will cause transmission delays and noise. Thus, for this system, the V2I data exchange model will offer more noise resistance.

This paper is divided into four major parts, in the first section we will explain the concept through UML and SysML diagrams. However, the second section is dedicated to the verification and validation of the proposed approach by using UPPAAL. The implementation is composed of two sections which are FreeRTOS and VHDL that we will explain in sections four and five respectively while the last section will conclude the paper.

II. Software Engineering Models

In this section, we are presenting four types of software engineering models in order to explain the ultimate functionality of V2I communication which is designed in the used case diagram, and then focus on the server or interface working flow through the activity diagram. However, the state machine diagram illustrates the different states of the car according to the decision-making results of the intersection manager. Finally, we define the different specifications of the system in a requirements diagram.

A. Use-Case Diagram

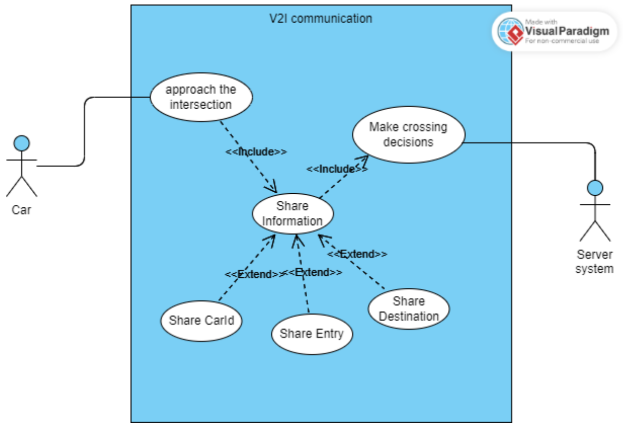


Fig. 1. Use Case diagram of V2I Communication

Figure 1 is presenting the different V2I communication use cases. The traffic coordination is mainly based on the exchange of data between the interface which represent the intersection controller and the self-driving vehicles. The main three use cases are:

- Approach the intersection
- Share information
  - Share CarId

    - Share Entry
    - Share Destination
- Make crossing decision

Thus, cars will approach the intersection and automatically send their data to the interface. The shared information has to include the intersection entry ID (North, South, East, or West), the destination that the car is driving to (Left, Right, or Straight), and the car ID. Therefore, Once, the intersection controller receives these inputs, it will make a crossing decision by confirming the car's request or declining it.

B. Activity Diagram

The proposed model in Figure 2 represents the flow of the interface activity. The interface is actively waiting for receiving data from cars. Once it received the information that we mentioned in the use case diagram, the car ID will be pushed into the queue of the corresponding entry. The intersection interface has to check whether this car is at the head of the queue and if it will cause a collision. If all the requests of the different cars from different entries will not lead to a collision at the level of the intersection, then, the interface will send a crossing confirmation signal to the different cars. As a result, cars are allowed to drive otherwise, they will have to wait and then drive in a sequential way according to their order. The same process will be repeated with other requests, once, the car crossed the intersection.

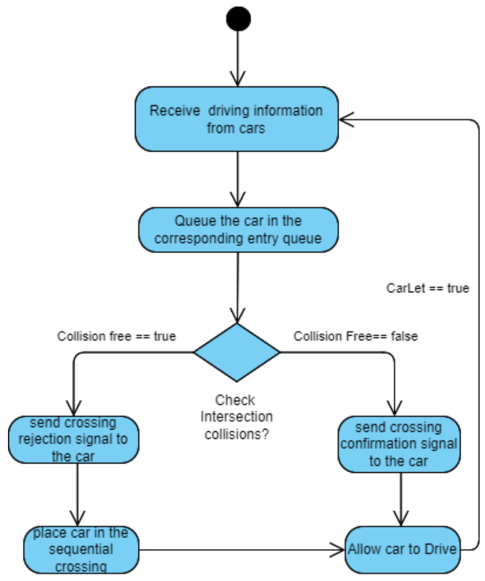

Receive driving information from cars
Queue the car in the corresponding entry queue
CarLet == true
Collision free == true
Collision Free== false
Check Intersection collisions?
send crossing rejection signal to the car
send crossing confirmation signal to the car
place car in the sequential crossing
Allow car to Drive

Fig. 2. Activity diagram of the Intersection Controller Interface

C. State Machine Diagram

This diagram illustrates the different states that autonomous vehicles can have during the crossing process of an intersection. First, the car is in an Idle state, and once it arrives at an intersection area, the car will identify, the intersection entry where it's located and define the desired destination and share them with the interface. As a result, the car ID will be placed in the entry queue. If the car request is collision-free, then the car will be permitted by the intersection manager to drive. Otherwise, the car will be in a waiting state as soon as it receives, a sequential order for intersection crossing. Then, it will be able to drive. Finally, when the car leaves the intersection zone, the whole process will be repeated.

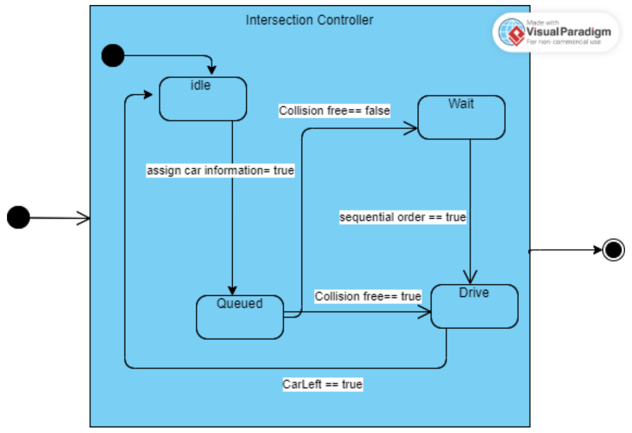
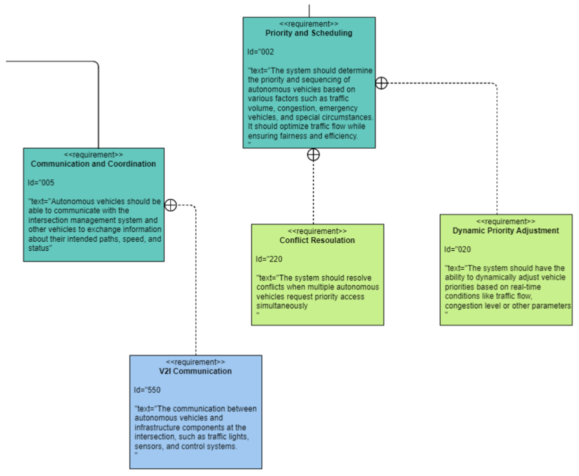


Fig. 3. Finite State diagram of the autonomous vehicles

D. Requirement Diagram

The following requirements in figure 5 and 4 outline the essential functionalities and capabilities expected from an autonomous intersection control system. These requirements are integral to the successful implementation and operation of the system.



<<requirement>>
Priority and Scheduling
Id="002
"text="The system should determine the priority and sequencing of autonomous vehicles based on various factors such as traffic volume, congestion, emergency vehicles, and special circumstances. It should optimize traffic flow while ensuring fairness and efficiency.
"

<<requirement>>
Communication and Coordination
Id="005
"text="Autonomous vehicles should be able to communicate with the intersection management system and other vehicles to exchange information about their intended paths, speed, and status"

<<requirement>>
Conflict Resoulation
Id="220
"text="The system should resolve conflicts when multiple autonomous vehicles request priority access simultaneously
"

<<requirement>>
Dynamic Priority Adjustment
Id="020
"text="The system should have the ability to dynamically adjust vehicle priorities based on real-time conditions like traffic flow, congestion level or other parameters
"

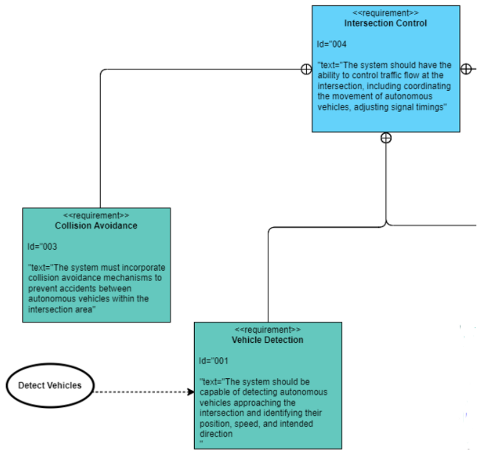
<<requirement>>
V2I Communication
Id="550
"text="The communication between autonomous vehicles and infrastructure components at the intersection, such as traffic lights, sensors, and control systems.
"

Fig. 4. Requirement Diagram section 2

- **Requirement 004: Intersection Control** The system should have the ability to control traffic flow at the intersection, including coordinating the movement of autonomous vehicles and adjusting signal timings. This ensures efficient and smooth traffic flow while maintaining safety and adherence to traffic regulations.
- **Requirement 002: Priority and Scheduling** The system should determine the priority and sequencing of

autonomous vehicles based on factors such as traffic volume, congestion, emergency vehicles, and special circumstances. By optimizing the traffic flow and considering various factors, the system aims to ensure fairness and efficiency in the movement of vehicles through the intersection.

- **Requirement 550: V2I Communication** Effective communication between autonomous vehicles and the infrastructure components at the intersection is essential. This includes establishing reliable communication channels between vehicles and traffic lights, sensors, and control systems. Seamless V2I communication enables synchronized control and coordination for efficient traffic management.
- **Requirement 005: Communication and Coordination** To enable efficient intersection control, autonomous vehicles should be able to communicate with the intersection management system and other vehicles. This communication facilitates the exchange of information about the vehicles' intended paths, speed, and status, allowing for coordinated maneuvers and optimized traffic flow.



<<requirement>>
Intersection Control
Id="004
"text="The system should have the ability to control traffic flow at the intersection, including coordinating the movement of autonomous vehicles, adjusting signal timings"
<<requirement>>
Collision Avoidance
Id="003
"text="The system must incorporate collision avoidance mechanisms to prevent accidents between autonomous vehicles within the intersection area"
Detect Vehicles
<<requirement>>
Vehicle Detection
Id="001
"text="The system should be capable of detecting autonomous vehicles approaching the intersection and identifying their position, speed, and intended direction
"

Fig. 5. Requirement Diagram section 1

- **Requirement 001: Vehicle Detection** Accurate and timely detection of approaching autonomous vehicles is crucial for effective intersection control. The system should be capable of detecting vehicles' presence, identifying their position, speed, and intended direction. This information forms the basis for making informed decisions regarding traffic management and sequencing.
- **Requirement 003: Collision Avoidance** The system must incorporate robust collision avoidance mechanisms to prevent accidents between autonomous vehicles within the intersection area. By employing advanced sensors, algorithms, and real-time decision-making capabilities, the system aims to ensure the safety of all vehicles by proactively avoiding collisions. By addressing these requirements, the autonomous intersection control system can effectively manage traffic flow, optimize vehicle sequencing, enable communication and coordination, detect vehicles accurately, and mitigate collision risks.

III. Uppaal Modelling

In this section, we modeled the proposed solution with Uppaal, in order to verify whether the system will behave as intended and used verification queries to validate the system. Therefore, we used six different automata templates. Four of these templates represent cars from different entries (North, South, East, and West), one automata is dedicated to the intersection interface or manager (Process) and we have a backup server, which will replace the main server in case of failure. All cars from different entries request desired destinations simultaneously (Left, Right, and Straight), and the server makes the decision about placing some cars on the waiting list and enabling others for driving by implementing First-In-First-Out (FIFO) queuing concept as shown in the figure: The different automata templates use synchronization

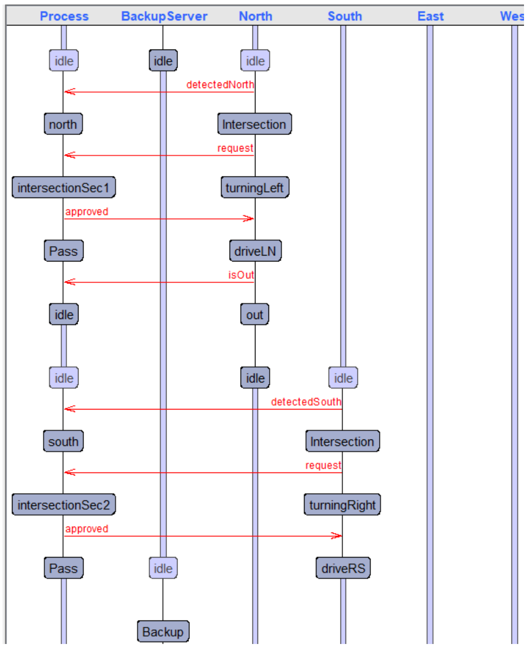
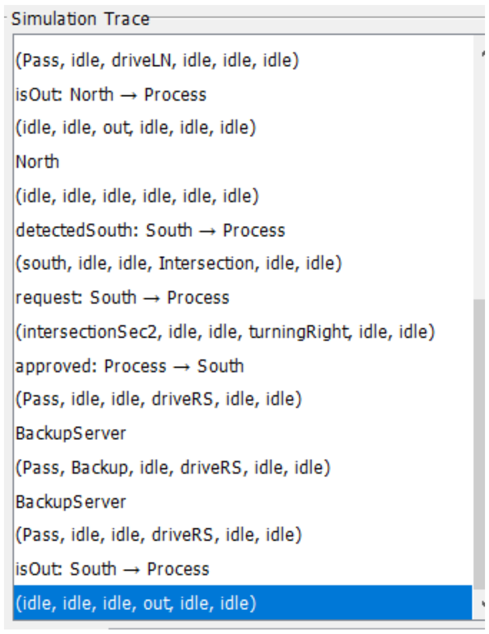


Fig. 6. Activity diagram of the Intersection Controller Interface

signals. Thus, once a car enters an intersection zone, it sends a signal to the server to inform it with its data using (detectedNorth!). then, it asks for crossing permission through a (request!) signal. As a result, the server sends a crossing approval signal (approved!) or rejection signal (wait!) after examining the collision scenarios. According to the received signal, the car can drive and consequently send (out!) signal to the interface once it is left or it has to stop until it's permitted to cross.

Figure 6 shows the simulation history of the system where we can track the requests executions of the different cars as



```
Simulation Trace
(Pass, idle, driveLN, idle, idle, idle)
isOut: North → Process
(idle, idle, out, idle, idle, idle)
North
(idle, idle, idle, idle, idle, idle)
detectedSouth: South → Process
(south, idle, idle, Intersection, idle, idle)
request: South → Process
(intersectionSec2, idle, idle, turningRight, idle, idle)
approved: Process → South
(Pass, idle, idle, driveRS, idle, idle)
BackupServer
(Pass, Backup, idle, driveRS, idle, idle)
BackupServer
(Pass, idle, idle, driveRS, idle, idle)
isOut: South → Process
(idle, idle, idle, out, idle, idle)
```

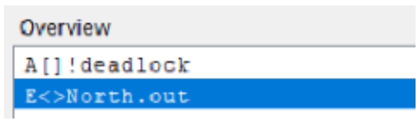
Fig. 7. Simulation History of the different templates

well as the coordination between the main server and the backup server, which stores the data as long as the main server is fully functioning and in case of a failure it will send the final status to the main server once it recovers.

One of the most important features that Uppaal offers are query checking where we can check whether the specified properties are fulfilled. It is crucial to verify and validate specified conditions that our model has to verify according to the system's requirements. In our case, Our system has to be reactive and continuously working and the final places of the different cars' entries are reachable as follows:

- **A[]!deadlock**
- **E<>North.out**

In the following figure, we verified the existence of a deadlock and whether the end place (Out) of a car North template is achieved.



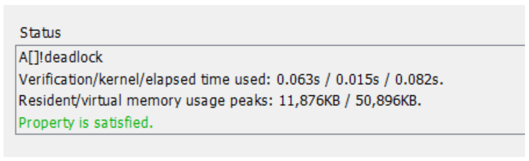
```
Overview
A[]!deadlock
E<>North.out
```

Fig. 8. Activity diagram of the Intersection Controller Interface

For the reachability test, we can apply the same query on the different places of the different templates but in our case, we only verified the reachability of (out) for the North entry.

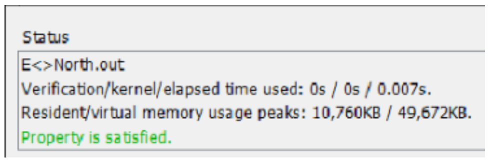
As it is shown in Figure 8, our system is deadlock-free. As a result, our system guaranteed that all the different parts will be able to make progress and no lock will be realized.

The reachability test of the (out) place. This means that Car North will be able to realize all the different transitions.



```
Status
A[]!deadlock
Verification/kernel/elapsed time used: 0.063s / 0.015s / 0.082s.
Resident/virtual memory usage peaks: 11,876KB / 50,896KB.
Property is satisfied.
```

Fig. 9. Deadlock verification result



```
Status
E<>North.out
Verification/kernel/elapsed time used: 0s / 0s / 0.007s.
Resident/virtual memory usage peaks: 10,760KB / 49,672KB.
Property is satisfied.
```

Fig. 10. Reachability verification result

IV. Overview of the Proposed Solution

The project's objective is to manage the traffic at a four-way intersection. Cars can approach the intersection from four directions (North, East, South, West), and they can go straight, turn left, or turn right. Each direction has its own queue to maintain the order of cars. The traffic is managed by allowing the cars to pass through the intersection based on their order in the queue and their desired turn direction.

V. Real-time Systems Implementation (FreeRTOS)

FreeRTOS is a prevalent real-time operating system designed for embedded devices. It provides services such as preemptive multitasking, inter-task communication and synchronization, and memory management. The design of FreeRTOS is minimalist, making it suitable for microcontrollers.

VI. Benefits of FreeRTOS

The benefits of using FreeRTOS for a project such as our Traffic Control System are numerous.

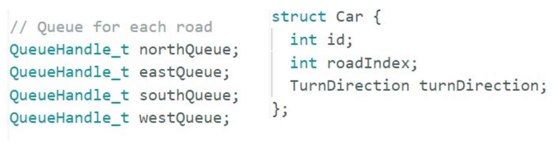
- **Task Management:** FreeRTOS enables us to design our program as a set of independently executing tasks. This is ideal for our project, where several separate activities occur: controlling the intersection, generating cars, and checking for deadlocks.
- **Preemptive Scheduling:** RTOS usually employ preemptive scheduling, enabling them to interrupt one task to execute another task of higher importance. This feature is crucial in real-time systems where specific tasks (like controlling the intersection) might have stringent deadlines.
- **Inter-Task Communication and Synchronization:** FreeRTOS provides several methods for tasks to communicate with each other and synchronize their activities. In our project, we use queues to manage the cars, semaphores to handle deadlocks, and mutexes to protect shared resources.
- **Memory Management:** FreeRTOS offers memory management capabilities, allowing for both static and dynamic allocation, helping optimize memory usage in memory-constrained systems.

- **Portability:** FreeRTOS is designed to be portable and is available for many different architectures. This feature allows switching to a different microcontroller with minimal porting effort.
- **Deterministic Behavior:** Being a real-time operating system, FreeRTOS provides deterministic behavior, i.e., the timing of task execution is predictable and consistent, which is crucial in many embedded systems.

VII. ESP-IDF FreeRTOS Version

The Espressif IoT Development Framework (ESP-IDF) uses a version of FreeRTOS designed to take full advantage of the ESP32's dual-core architecture. It includes support for the creation and management of tasks on either of the ESP32's two cores, freeing up the second core for computation or wireless communication tasks. This makes it ideal for complex, real-time applications like our traffic control system.

A. Code Implementation



```
// Queue for each road
QueueHandle_t northQueue;
QueueHandle_t eastQueue;
QueueHandle_t southQueue;
QueueHandle_t westQueue;
```

```
struct Car {
  int id;
  int roadIndex;
  TurnDirection turnDirection;
};
```

Fig. 11. Data Structs and Enums

1) Data Structures and Enumerations:

- TurnDirection: Enumerated type to represent the turn direction options (LEFT, STRAIGHT, RIGHT).
- Car: Structure to represent a car, containing its ID, the index of the road it is on, and its intended TurnDirection. This is shown in Figure 10.

2) Global Variables and Queues:

- Semaphore and Mutex objects are created to handle the synchronization between different tasks and prevent race conditions.
- Four queues are created to represent each road, using the FreeRTOS Queue API. Each queue stores Car objects.

3) Task Creation and Execution: Three tasks are created using the FreeRTOS Task API:

- **IntersectionControllerTask:** This task manages the intersection. It checks each queue in turn (North, East, South, West), and decides which car can pass based on their booking order and the turn direction. If there is a car in the opposite direction queue that wants to go straight or turn right, they are allowed to pass simultaneously. If not, only the car from the checking queue passes. A mutex is used to protect the carPassed variable to prevent simultaneous writes.
- **FillCarIdDirToQueues:** This task simulates cars approaching the intersection. Each car is given a random ID, a road (based on the road's queue), and a turn direction. This task then adds the car to the appropriate road queue.
- **DeadlockDetectionTask:** This task runs periodically and checks the status of the synchronization primitives to detect possible deadlocks or issues.

All tasks are created with the same priority and are not pinned to any specific core.

B. Why This Approach?

This approach models the real-life scenario of managing a four-way intersection quite accurately. By using a multi-threading environment (FreeRTOS), each aspect of the problem (intersection control, car simulation, deadlock detection) is separated into its own task, promoting the separation of concerns and making the code easier to understand and maintain. The use of semaphores and mutexes ensures safe access to shared resources across different tasks, preventing race conditions.

This design is also flexible and scalable. For instance, additional features such as pedestrian crossings, traffic signals, or different road layouts could be added by creating new tasks or modifying existing ones.

The use of queues allows for the orderly processing of cars at the intersection, and the use of structures and enumerations makes the code more readable and easier to manage.

The deadlock detection mechanism is crucial for detecting potential issues in real-time systems, contributing to overall system reliability.

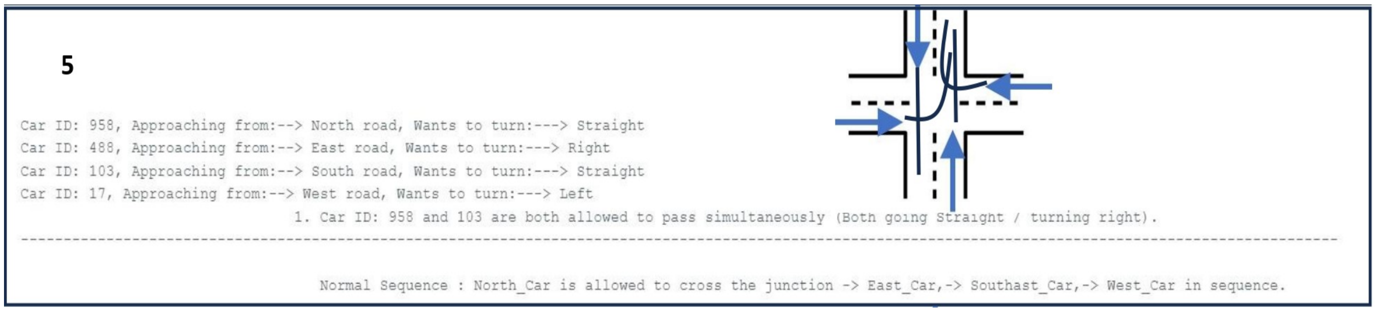
C. Results of FreeRTOS Implementation

The ESP32 microcontroller was successfully programmed with FreeRTOS using the outlined procedures. The results of the implementation are presented in Figure 11, demonstrating the effective utilization of the operating system for task management and synchronization in the traffic control system.

VIII. Software/Hardware Co-design Implementation (VHDL)

A. Purpose

The VHDL implementation was undertaken within the framework of hardware-software co-design, aiming to integrate the functionality of a decision-making mechanism into an Application-Specific Integrated Circuit (ASIC). The primary objective of the VHDL component is to facilitate communication between the ASIC and the freeRTOS operating system, enabling the ASIC to receive decisions regarding the allowance or prohibition of vehicle movement. The ASIC is pre-programmed with an extensive collection of potential intersection scenarios, encompassing various types of crossings that could potentially result in collisions. In instances where the freeRTOS system generates a decision that could lead to a collision, the ASIC module serves to transmit a corresponding message to the freeRTOS operating system, effectively alerting it to the possibility of a collision and prompting a decision modification.



```
Car ID: 958, Approaching from:--> North road, Wants to turn:---> Straight
Car ID: 488, Approaching from:--> East road, Wants to turn:---> Right
Car ID: 103, Approaching from:--> South road, Wants to turn:---> Straight
Car ID: 17, Approaching from:--> West road, Wants to turn:---> Left
                              1. Car ID: 958 and 103 are both allowed to pass simultaneously (Both going Straight / turning right).
--------------------------------------------------------------------------------------------------------------------------------------

                                Normal Sequence : North_Car is allowed to cross the junction -> East_Car,-> Southast_Car,-> West_Car in sequence.
```

Fig. 12. FreeRTOS decision on ESP32 and Arduino IDE

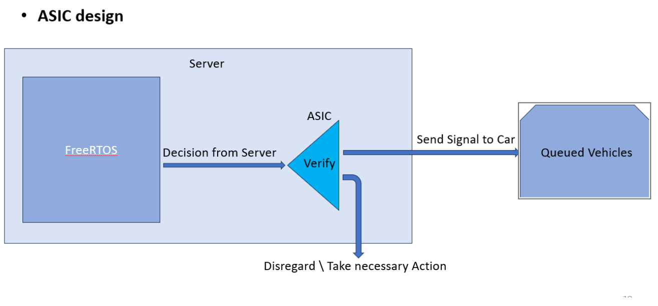


Fig. 13. ASIC Design

B. Entities

The TrafficController entity represents a module that acts as a traffic controller within a digital system. It has the following port connections:

Input Ports:

Car1, Car2, Car3, Car4: These are four input signals representing the presence or absence of cars at four different intersections.

Bidirectional Ports:

Car1Direction, Car2Direction, Car3Direction, Car4Direction: These are bidirectional signals represented as three-bit vectors. They indicate the directions for each car to move. The controller assigns values to these signals based on the traffic conditions.

Output Port:

Collision: This is an output signal indicating whether a collision is detected by the traffic controller.

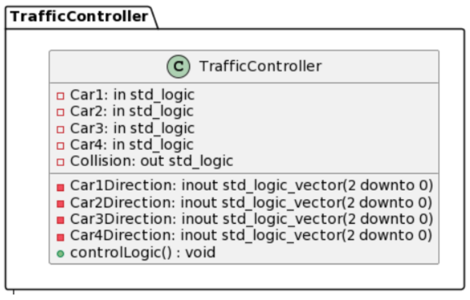

TrafficController
C TrafficController
Car1: in std_logic
Car2: in std_logic
Car3: in std_logic
Car4: in std_logic
Collision: out std_logic
Car1Direction: inout std_logic_vector(2 downto 0)
Car2Direction: inout std_logic_vector(2 downto 0)
Car3Direction: inout std_logic_vector(2 downto 0)
Car4Direction: inout std_logic_vector(2 downto 0)
controlLogic() : void

Fig. 14. Entity declaration

C. Architecture and Logic

The Traffic Controller module interfaces with four input signals, namely ‘Car1‘, ‘Car2‘, ‘Car3‘, and ‘Car4‘, which indicate the presence or absence of cars at distinct intersections. Additionally, the module features bidirectional signals ‘Car1Direction‘, ‘Car2Direction‘, ‘Car3Direction‘, and ‘Car4Direction‘, denoting the permitted direction of travel for each car, as well as an output signal ‘Collision‘, which signifies the possibility of a collision occurring.
The behavior of the Traffic Controller module is established through conditional statements predicated on the values of the input signals. These statements facilitate the handling of diverse traffic scenarios, determining the appropriate directions and collision status. Each scenario corresponds to a specific traffic condition that the controller can effectively manage.

In the project documentation, the behavior of the Traffic Controller is comprehensively explicated by providing an interpretation of each scenario, interpreting the associated conditions and outcomes. Here, each scenario is individually described:

1. Scenario: All cars can go right simultaneously - Conditions: ‘Car1 = ’1’‘, ‘Car2 = ’1’‘, ‘Car3 = ’1’‘, ‘Car4 = ’1’‘ - Directions: Each car (‘Car1Direction‘, ‘Car2Direction‘, ‘Car3Direction‘, ‘Car4Direction‘) is permitted to travel rightward. - Collision: The ‘Collision‘ signal is set to ’0’ to indicate no potential collision.

2. Scenario: Only cars coming from opposite directions can proceed straight simultaneously - Conditions: ‘Car1 = ’1’‘, ‘Car2 = ’1’‘, ‘Car3 = ’0’‘, ‘Car4 = ’0’‘ - Directions: Cars originating from opposite directions (‘Car1‘, ‘Car2‘) are authorized to move straight ahead, while the remaining cars are not permitted to proceed. - Collision: The ‘Collision‘ signal is set to ’0’ since no collision is anticipated.

3. Scenario: Cars coming from opposite directions can turn left simultaneously - Conditions: ‘Car1 = ’0’‘, ‘Car2 = ’0’‘, ‘Car3 = ’1’‘, ‘Car4 = ’1’‘ - Directions: Cars originating from opposite directions (‘Car3‘, ‘Car4‘) are granted permission to turn left, while the remaining cars are not allowed to move. - Collision: The ‘Collision‘ signal is set to ’0’ since no collision is expected.

4. Scenario: Individual Cases - The conditions, directions, and collision outcomes for each individual case are duly documented through explanatory comments in the code. Each

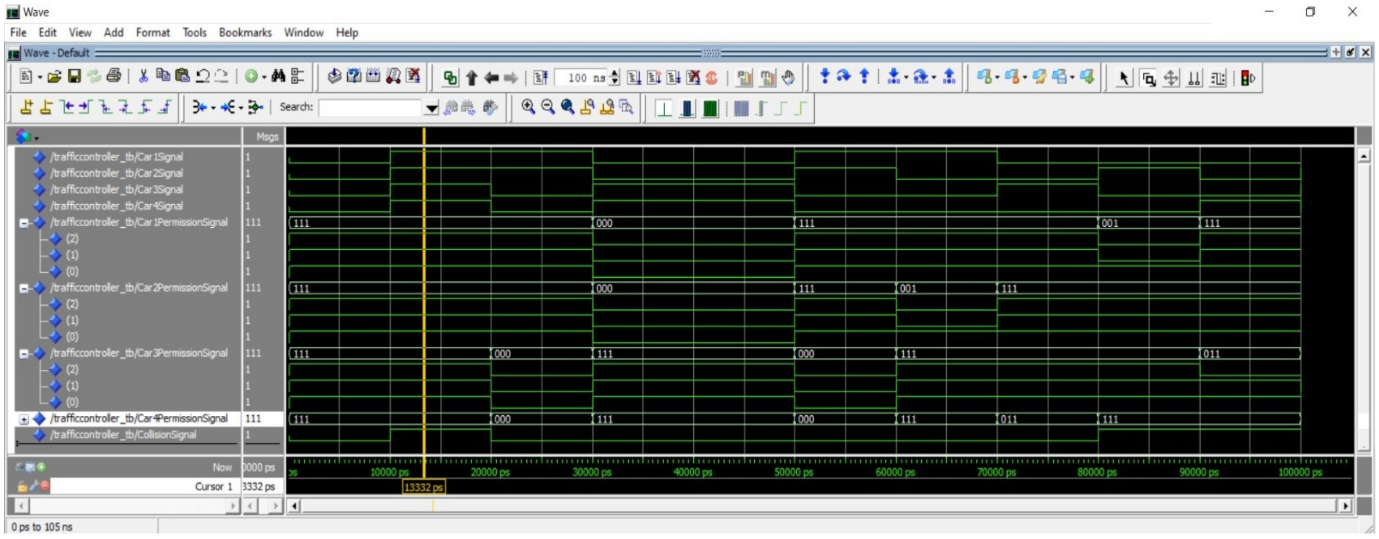

Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/trafficcontroller_tb/Car1Signal
/trafficcontroller_tb/Car2Signal
/trafficcontroller_tb/Car3Signal
/trafficcontroller_tb/Car4Signal
/trafficcontroller_tb/Car1PermissionSignal
/trafficcontroller_tb/Car2PermissionSignal
/trafficcontroller_tb/Car3PermissionSignal
/trafficcontroller_tb/Car4PermissionSignal
/trafficcontroller_tb/CollisionSignal
Now 100000 ps
Cursor 1 13332 ps
10000 ps 20000 ps 30000 ps 40000 ps 50000 ps 60000 ps 70000 ps 80000 ps 90000 ps 100000 ps
0 ps to 105 ns

Fig. 15. Modelsim Simulation

scenario is systematically described, incorporating the specific conditions, directions, and collision expectations.

5. Default Case: - In the event that none of the predefined scenarios align with the input conditions, the default case is executed. - Directions: The direction vectors for all cars (‘Car1Direction‘, ‘Car2Direction‘, ‘Car3Direction‘, ‘Car4Direction‘) are initialized to zero, indicating that no car is permitted to proceed in any direction. - Collision: The ‘Collision‘ signal is set to ’0’ since no collision is projected.

IX. Conclusion

In conclusion, our project documentation presents a solution for an autonomous traffic intersection controller. We addressed the problem of traffic congestion, delays, and accidents at signalized intersections by proposing an intelligent intersection management system. The system utilizes Vehicle-to-Infrastructure (V2I) communication to facilitate the exchange of data between self-driving vehicles and the intersection controller.

We provided a detailed explanation of the system’s functionality using UML and SysML diagrams, including use case diagrams, activity diagrams, state machine diagrams, and requirement diagrams. These diagrams illustrate the various aspects of the system, such as traffic coordination, communication flow, decision-making processes, and system requirements.

Furthermore, we implemented the system using FreeRTOS, a real-time operating system and demonstrated its benefits in task management, preemptive scheduling, inter-task communication, synchronization, and memory management. We also presented the VHDL implementation for hardware-software co-design, integrating the decision-checking mechanism into an Application-Specific Integrated Circuit (ASIC).

Overall, our project aims to enhance traffic management, improve safety, and optimize the movement of autonomous vehicles at intersections. By effectively coordinating and controlling the flow of vehicles, we anticipate a reduction in congestion, accidents, and fuel consumption, leading to a more efficient and sustainable transportation system.

References

[1] U. D. o. Transportation, ”National Agenda for Intersection Safety”, Federal Highway Administration, Washington, 2005.

[2] A. Parker and G. Nitschke, ”How to best Automate Intersection Management,” 2017 IEEE Congress on Evolutionary Computation (CEC), Donostia, Spain, 2017, pp. 1247-1254, doi: 10.1109/CEC.2017.7969448.

[3] M. Rouse, ”intelligent transportation system”, June 2023. [Online]. Available: https://whatis.techtarget.com/definition/intelligent-transportation-system. [Accessed 30.06.2023].

[4] K. K. Daniel J. Fagnant, ”Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations”, Transportation Research Part A, vol. 77, pp. 167-181, 2015.

[5] J. C. S. a. D. C. Chin, ”Traffic-responsive signal timing for system-wide traffic control”, Transportation Research Part C: Emerging, vol. 5, p. 153–163, 1997.

X. Declaration of Originality

I, Jaouaher Belgacem, herewith declare that I have composed the present paper and work by myself and without the use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The paper and work in the same or similar form have not been submitted to any examination body and have not been published. This paper was not yet, even in part, used in another examination or as a course performance. I agree that my work may be checked by a plagiarism checker.

[illegible]

02.07.2023 Lippstadt - Jaouaher Belgacem

I, Nirojan Navaratnarajah , herewith declare that I have composed the present paper and work by myself and without the use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The paper and work in the same or similar form have not been submitted to any examination body and have not been published. This paper was not yet, even in part, used in another examination or as a course performance. I agree that my work may be checked by a plagiarism checker.

[illegible]

02.07.2023 Lippstadt - Nirojan Navaratnarajah

I, Neaz Mahmud, herewith declare that I have composed the present paper and work by myself and without the use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The paper and work in the same or similar form have not been submitted to any examination body and have not been published. This paper was not yet, even in part, used in another examination or as a course performance. I agree that my work may be checked by a plagiarism checker.

[illegible]

02.07.2023 Lippstadt - Neaz Mahmud

Note: It is important to note that all group members have worked on the different parts of this project. As a group, we always chose the best solution of the suggested ones by each member and collaborate to improve it.