Smart Traffic System

1st Syed Rafsan Ishtiaque
Electronic Engineering
Hochschule Hamm-Lippstadt
syed-rafsan.ishtiaque@stud.hshl.de

2nd Yigitcan Aydin
Electronic Engineering
Hochschule Hamm-Lippstadt
yigitcan.aydin@stud.hshl.de

***Abstract*—The rapid development of autonomous vehicles demands the smooth traffic cross overs in intersections. To develop the idea of collision free intelligent traffic intersection, we modelled our project in such a way that it will provide assisting measure to the autonomous vehicles in a multiple road cross overs. The cuing system will provide an upgraded version of autonomous vehicle registration. The system will check the priority of the vehicles and decide the appropriate model to cross the intersection. We have developed the system in C programming language and simulated in VHDL and RTOS**

***Index Terms*—autonomous vehicle, queuing system, free RTOS, VHDL, collision free intersection**

I. Introduction

The fundamental idea of the project was to make a model environment that will provide the efficient traffic intersection. We have included vehicle registration and de-registration mechanism in the traffic intersection. It will collaborate to determine the priority of the vehicles. Then the abstraction model is to categorize the level of the priority. Hence we demonstrate the level two priority for the vehicles to take over allocated time and lane.

II. Requirements of the Project

- Modelling a traffic intersection for autonomous cars using Queuing system
- Best scenario to reduce the traffic congestion in an intersection
- Simulate traffic intersection using C-code (core logic), VHDL and freeRTOS

III. Our Approach

We designed our model considering the amount of vehicles coming from a certain direction in certain unit of time. So the first priority selection will be based on the inward vehicle volume towards the intersection. We attached a counter to collect this data. At this stage, the vehicles will be provided individual identity.

IV. Determining the Level Two Priority

For level two priority, we will consider the number of vehicles going from the intersection outward of individual direction. Here we demonstrate a scenario. First, we gave the EAST direction a level 1 first priority. So, the two subdivision of EAST, that we have annoted as E3W2 and E3S2 will be checked first for level 2 priority.

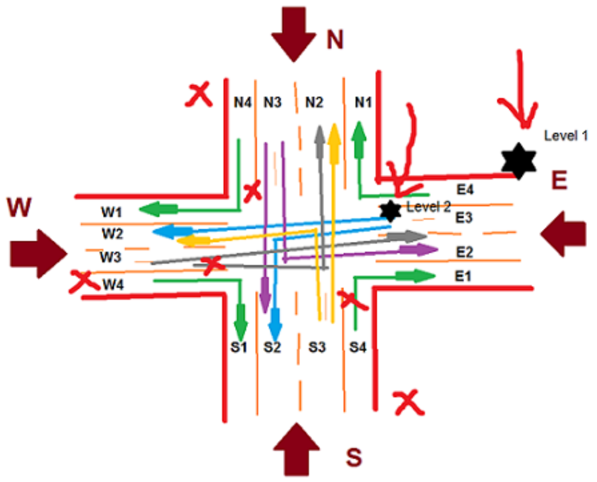


Fig. 1. Determining priority for level 1

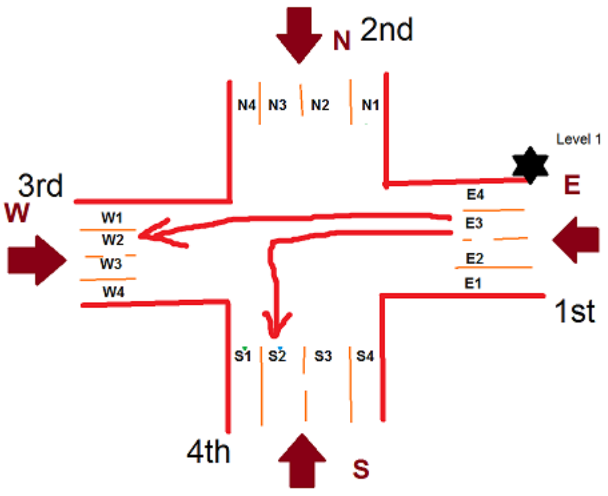


Fig. 2. Determining priority for level 2

V. Interchanging the Level Two Priority

Now we will talk about what will happen when two similar level two prioritized path will interchange between them. In first case, E3W2 will gain higher priority than E3S2. In that sense, we will have the option to pass vehicle in either S3W3 way or W3E2. But we can't let them run at the same time as it will create collision. So, we will check the level two priority for those two paths separately and decide which one will be higher.

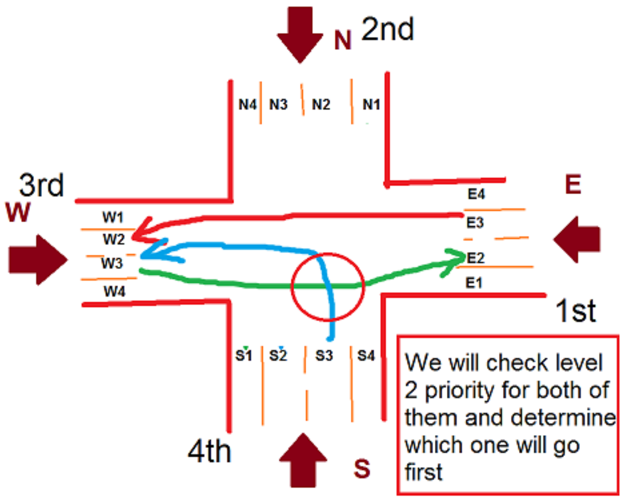

N
2nd
N4
N3
N2
N1
3rd
W
W1
W2
W3
W4
E
E4
E3
E2
E1
1st
S1
S2
S3
S4
4th
S
We will check level 2 priority for both of them and determine which one will go first

Fig. 3. Selecting route when two paths crosses

The same way when level 2 priority of both of the EAST sub paths will be interchanged, here E3S2 will get higher level 2 priority than E3W2, we will see another different scenario. Now we will check the level two priority between W3N2 and N3S2, otherwise there will be always a chance for collision.

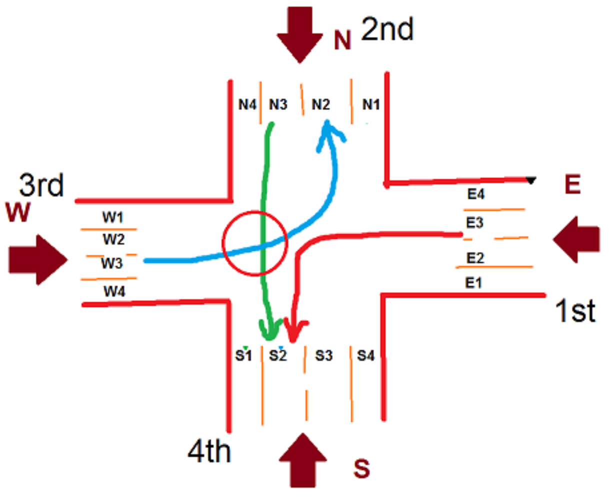


Fig. 4. Selecting route when two paths crosses

VI. VHDL Simulation

Now we have an idea how the traffic modelling will look like. As we mentioned earlier, EAST ¿ NORTH ¿ WEST ¿ SOUTH. So we are not changing the initial priority or level one priority in our simulation. The following diagram represent the modelsim VHDL representation of Scenario 1

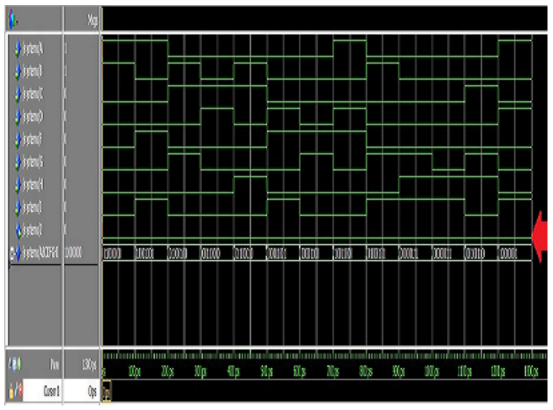
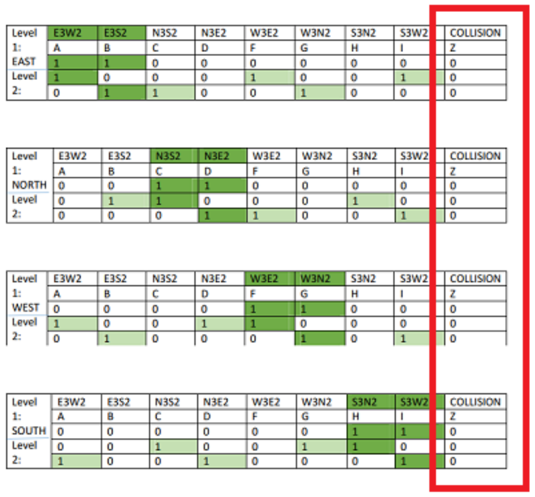


Fig. 5. VHDL simulation for scenario one

VII. Overall collision algorithm and representation

Though we have implemented only one scenario, but this structure of the algorithm remain same for the other cases. So in different scenario, when either NORTH or WEST or SOUTH will have higher priority for level 1, the algorithm we used in our model will determine the possible outcome for traffic intersection. As we can see, as long as we follow this methos, we will not have any collision. In that case, we can say our model is proved to be collision free



Level	E3W2	E3S2	N3S2	N3E2	W3E2	W3N2	S3N2	S3W2	COLLISION
1:	A	B	C	D	F	G	H	I	Z
EAST	1	1	0	0	0	0	0	0	0
Level	1	0	0	0	1	0	0	1	0
2:	0	1	1	0	0	1	0	0	0

Level	E3W2	E3S2	N3S2	N3E2	W3E2	W3N2	S3N2	S3W2	COLLISION
1:	A	B	C	D	F	G	H	I	Z
NORTH	0	0	1	1	0	0	0	0	0
Level	0	1	1	0	0	0	1	0	0
2:	0	0	0	1	1	0	0	1	0

Level	E3W2	E3S2	N3S2	N3E2	W3E2	W3N2	S3N2	S3W2	COLLISION
1:	A	B	C	D	F	G	H	I	Z
WEST	0	0	0	0	1	1	0	0	0
Level	1	0	0	1	1	0	0	0	0
2:	0	1	0	0	0	1	0	1	0

Level	E3W2	E3S2	N3S2	N3E2	W3E2	W3N2	S3N2	S3W2	COLLISION
1:	A	B	C	D	F	G	H	I	Z
SOUTH	0	0	0	0	0	0	1	1	0
Level	0	0	1	0	0	1	1	0	0
2:	1	0	0	1	0	0	0	1	0

Fig. 6. Overall collision algorithm

A. C++ Code Explanation

In the code implementation, we have to use cygwin to run the code since some of the libraries used in the code such as pthread.h and unistd.h are supposed to be used in unix based operating systems.

In order to get simpler code prototype for this system, in the code part we deducted the number of lanes in the each direction to one lane from two. Each directions are assigned as locked directions. As soon as a car spawns at different directions in every 3 seconds (it can be changed as desired), they are also given an ID which is also locked into the traffic system queue.

```
// a queue for each direction.
queue<int> north_lane;
queue<int> east_lane;
queue<int> south_lane;
queue<int> west_lane;
queue<int> intersection_lane;

pthread_t threadID;

// a lock for each queue/direction.
pthread_mutex_t northLock;
pthread_mutex_t eastLock;
pthread_mutex_t southLock;
pthread_mutex_t westLock;
pthread_mutex_t intersectionQLock;

pthread_mutex_t global_id_Lock;
pthread_mutex_t intersectionLock;
```

Since there are a lot of cars spawning, we have to determine for each of them a priority. The priority is determined according to the FIFO (First In First Out) algorithm. The car which has the lower ID is allowed to cross the intersection according to FIFO.

```
car_id = 1; //first car will have ID = 1
pthread_mutex_init(&northLock, NULL);
pthread_mutex_init(&eastLock, NULL);
pthread_mutex_init(&southLock, NULL);
pthread_mutex_init(&westLock, NULL);
pthread_mutex_init(&intersectionQLock, NULL);
pthread_mutex_init(&global_id_Lock, NULL);
pthread_mutex_init(&intersectionLock, NULL);
```

The incoming and outgoing directions of the cars are determined randomly by using random function. The time needed for a car to pass the intersection starts at the spawn point of the car. The difference between starting time and end time of the car's journey through the intersection gives the duration of the waiting at the intersection.

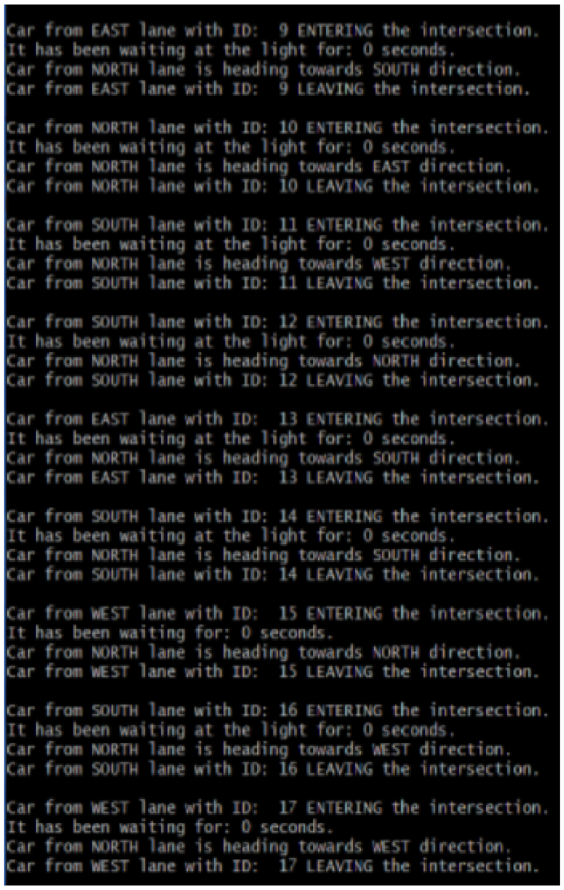
We are also locking the direction queues during the passage of the car through the intersection, in order to prevent the queue information from changing during the process. After the process is complete, we can unlock the queues again to be able to get the new tasks into the program flow.

```
pthread_mutex_lock(&northLock);
pthread_mutex_lock(&global_id_Lock);
localID = car_id++;
pthread_mutex_unlock(&global_id_Lock);
north_lane.push(localID);
pthread_mutex_unlock(&northLock);

sleep(2);
pthread_mutex_lock(&intersectionQLock);
intersection_lane.push(localID);
pthread_mutex_unlock(&intersectionQLock);
```

The car which is currently in front of the queue is allowed to pass. Otherwise, it has to wait until the cars that are in front of the following car finish their tasks first. After finishing their passage, all the information that belongs to the crossing cars are removed from the queue.

During and after the passage at the crossroad, informative messages are printed to the screen in order to monitor the complete process at the intersection.



```
Car from EAST lane with ID:  9 ENTERING the intersection.
It has been waiting at the light for: 0 seconds.
Car from NORTH lane is heading towards SOUTH direction.
Car from EAST lane with ID:  9 LEAVING the intersection.

Car from NORTH lane with ID: 10 ENTERING the intersection.
It has been waiting at the light for: 0 seconds.
Car from NORTH lane is heading towards EAST direction.
Car from NORTH lane with ID: 10 LEAVING the intersection.

Car from SOUTH lane with ID: 11 ENTERING the intersection.
It has been waiting at the light for: 0 seconds.
Car from NORTH lane is heading towards WEST direction.
Car from SOUTH lane with ID: 11 LEAVING the intersection.

Car from SOUTH lane with ID: 12 ENTERING the intersection.
It has been waiting at the light for: 0 seconds.
Car from NORTH lane is heading towards NORTH direction.
Car from SOUTH lane with ID: 12 LEAVING the intersection.

Car from EAST lane with ID:  13 ENTERING the intersection.
It has been waiting at the light for: 0 seconds.
Car from NORTH lane is heading towards SOUTH direction.
Car from EAST lane with ID:  13 LEAVING the intersection.

Car from SOUTH lane with ID: 14 ENTERING the intersection.
It has been waiting at the light for: 0 seconds.
Car from NORTH lane is heading towards SOUTH direction.
Car from SOUTH lane with ID: 14 LEAVING the intersection.

Car from WEST lane with ID:  15 ENTERING the intersection.
It has been waiting for: 0 seconds.
Car from NORTH lane is heading towards NORTH direction.
Car from WEST lane with ID:  15 LEAVING the intersection.

Car from SOUTH lane with ID: 16 ENTERING the intersection.
It has been waiting at the light for: 0 seconds.
Car from NORTH lane is heading towards WEST direction.
Car from SOUTH lane with ID: 16 LEAVING the intersection.

Car from WEST lane with ID:  17 ENTERING the intersection.
It has been waiting for: 0 seconds.
Car from NORTH lane is heading towards WEST direction.
Car from WEST lane with ID:  17 LEAVING the intersection.
```

Fig. 7. Result of the running code

```
 for(int i = 0; i < cars; i++)
{
 direction = getrand(1,5);
 k = getrand(1,5);
 if(k == 1)
```

```
    {
      pthread_create (&threadID, NULL, north, NULL);
    }
    else if (k == 2)
    {
      pthread_create (&threadID, NULL, east, NULL);
    }
    else if (k == 3)
    {
      pthread_create (&threadID, NULL, south, NULL);
    }
    else
    {
      pthread_create (&threadID, NULL, west, NULL);
    }
      sleep(3);
  }
```

REFERENCES

[1] C++ Reference. https://en.cppreference.com/w/

[2] Intelligent Traffic Light System. https://github.com/Anton1123/Intelligent-Traffic-Light-System