Technical University of Cluj Napoca

CAN Bus

Student: Măcean Marius

Group: 30433

Prof: Lia Anca Hangan

19/10/2021

Contents

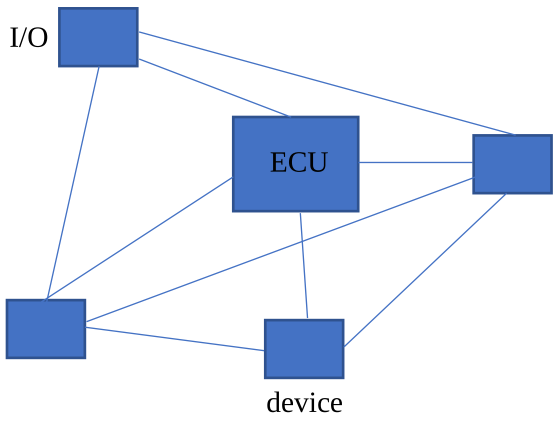
Bibliographic study

CAN = Controller Area Network

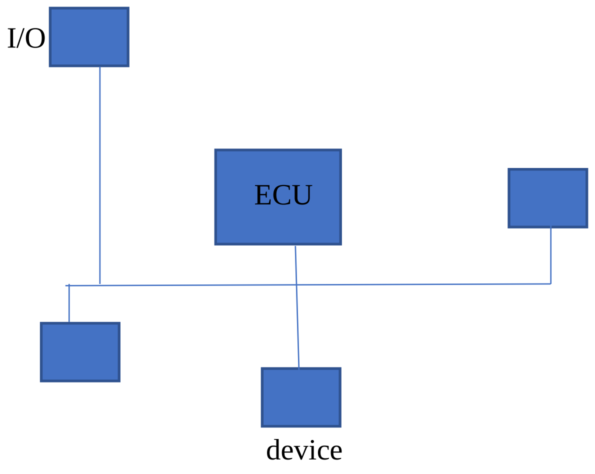
CAN bus is a robust vehicle bus standard designed to allow microcontrollers and devices (embadded in a vehicle) to communicate with each other's applications without a host computer. It is a protocol based on message passing. It can be applied in passagers vehicles, trucks, elevators, lighting control systems, electronic equipment for aviation and navigation etc.

Bosch released the first CAN protocol (in 1986).

Why is CAN needed? A vehicle contains a network of electronic devices that share data and information with one another. Timing is important regarding the functionality of a robust machine that interacts with humans and with other machines. Before CAN was introduced in vehicles, each electronic device was connected to another via wires (point-to-point wiring, to be specific). To ensure that the tasks occur accurately, the protocol sets rules by which electronic devices can exchange data with one another over a common serial bus. It reduced the wiring connections and the overall complexity of the system, as we can see in the following figures.



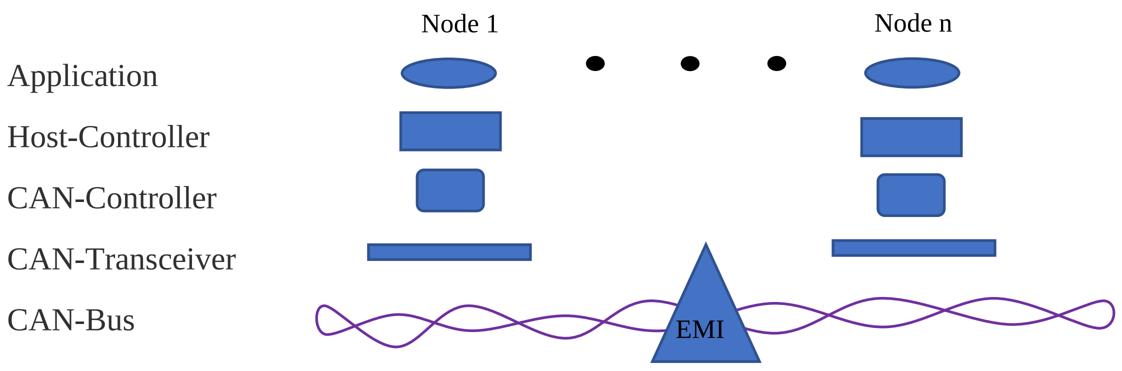
Without CAN



With CAN

ECU = electronic/microcontroller control unit.

A typical CAN network consists of several nodes. Each node has: a host-controller, a CAN-controller and a CAN-tranceiver.



Binary values in the CAN bus are termed as dominant ("0") and recessive ("1") bits. Information is send and received via frames (a specific format). The are 2 types of frames well-known: The Standard CAN (with 11 bit identifier fields) and the Extended frame (29 bits). The bit distribution for standard CAN is shown in the table.

SOF	11-bit Identifier	RTR	IDE	r0	DLC	0…8 Bytes Data	CRC	ACK	EOF	IFS

SOF – Start of Frame bit.

Identifier – to check which node will be next and to check the message that will be sent.

RTR - Remote Transmission Request.

IDE – Identifier Extension.

R0 – Reversed bit.

DLC – Data Length Code.

DATA– Used to store up to 64 data bits of application data to be transmitted.

CRC– Cyclic Redundancy Check.

ACK – Acknowledge field.

EOF– End of Frame (EOF).

IFS – Inter Frame Space.

In automobile test systems, it is essential to take into consideration the test cost, safety, efficiency, feasibility and duration. Therefore, it is required the development of a simulator that can efficiently hold the communication among the ECUs and a dashboard. Such a simulator can be Xilinx for vhdl code or the LABVIEW application (which most likely will be used).

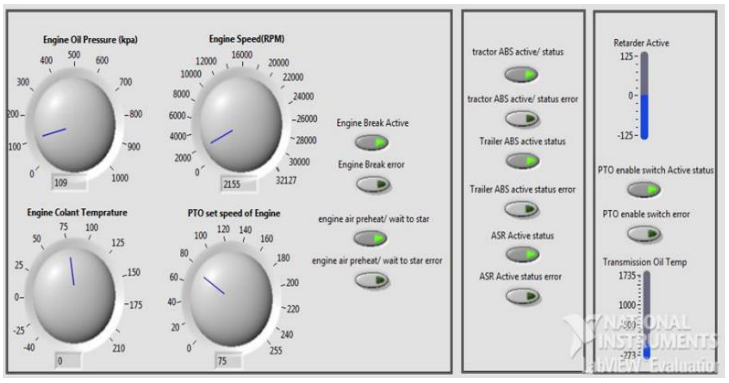
Project Proposal

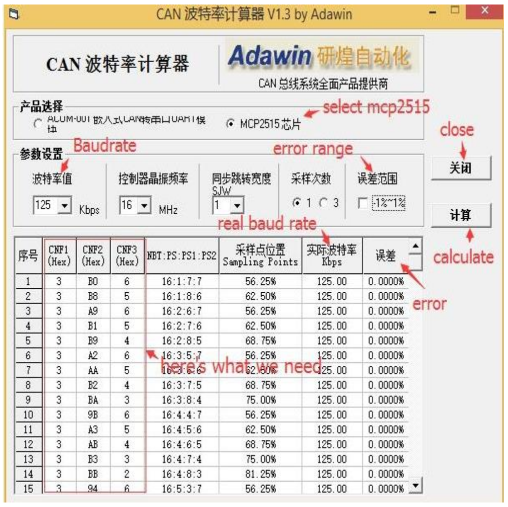
In this project we want to develop a simulator for a CAN bus. The network will replicate the design for a vehicle. We will take into consideration all the bits we need and its purposes. We will use that to make the simulator work. An architecture has to be implemented (consisting of the all the steps and states the simulator has to go through). In the simulation application (LABVIEW, for example) we will set the enviroment for a vehicle. Electronic devices of the vehicle (or even more vehicles as devices) will communicate with each other via a bus, exactly how it is done in a real CAN.

The final simulation product will be user friendly – easy to use. The user will see all the details about each node (device or vehicle) and will be able to test the transmission of information between nodes. Moreover, they will be able to verify the cost, safety, efficiency, feasibility and duration of each step/ transmission. Data will be saved as numbers in hexa.

As an upgrade, the simulator can be implemented on an Arduino board (or more) – depending on the time and price needed to develop such an upgrade. In this case, we will use the Arduino application/ compiler, but also a simulator (the Chinese one, by Adawin).

Finally, the simulation application interface will look something like the examples below.


Engine Oil Pressure (kpa)
Engine Speed(RPM)
Engine Break Active
Engine Break error
Engine Colant Temprature
PTO set speed of Engine
engine air preheat/ wait to star
engine air preheat/ wait to star error
tractor ABS active/ status
tractor ABS active/ status error
Trailer ABS active status
Trailer ABS active status error
ASR Active status
ASR Active status error
Retarder Active
PTO enable switch Active status
PTO enable switch error
Transmission Oil Temp
NATIONAL INSTRUMENTS
LabVIEW Evaluation



产品选择: ( ) ACOM-001 嵌入式CAN转串口UART模块 [✓ MCP2515 芯片]

参数设置: 波特率值 125 Kbps; 控制器晶振频率 16 MHz; 同步跳转宽度 SJW 1; 采样次数 [✓ 1] ( ) 3; 误差范围 ☐ -1%~1%

序号	CNF1 (Hex)	CNF2 (Hex)	CNF3 (Hex)	NBT:PS:PS1:PS2	采样点位置 Sampling Points	实际波特率 Kbps	误差
1	3	B0	6	16:1:7:7	56.25%	125.00	0.0000%
2	3	B8	5	16:1:8:6	62.50%	125.00	0.0000%
3	3	A9	6	16:2:6:7	56.25%	125.00	0.0000%
4	3	B1	5	16:2:7:6	62.50%	125.00	0.0000%
5	3	B9	4	16:2:8:5	68.75%	125.00	0.0000%
6	3	A2	6	16:3:5:7	56.25%	125.00	0.0000%
7	3	AA	5	16:3:6:6	62.50%	125.00	0.0000%
8	3	B2	4	16:3:7:5	68.75%	125.00	0.0000%
9	3	BA	3	16:3:8:4	75.00%	125.00	0.0000%
10	3	9B	6	16:4:4:7	56.25%	125.00	0.0000%
11	3	A3	5	16:4:5:6	62.50%	125.00	0.0000%
12	3	AB	4	16:4:6:5	68.75%	125.00	0.0000%
13	3	B3	3	16:4:7:4	75.00%	125.00	0.0000%
14	3	BB	2	16:4:8:3	81.25%	125.00	0.0000%
15	3	94	6	16:5:3:7	56.25%	125.00	0.0000%

Project Plan

1st meeting – projects presentation and declaring the choice;

2nd meeting – project proposal, plan, bibliography study;

3rd meeting – choose the simulation application (buy and install the products needed), create a basic/ simple simulation to learn the tools;

4th meeting – develop the architecture and design for the simulation (theoretically), describe all steps we are going to do;

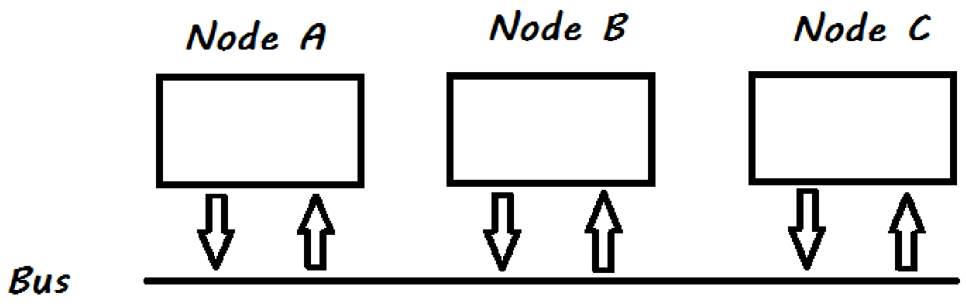
5th meeting – implement all the things presented last meeting, verify its basic functionalities;

6th meeting – apply more detailed tests on the simulator, fix the bugs if any, make it better if possible (with the time/ money left), wrap things up regarding the documantation, based on the last upgrades (implementation, tests).

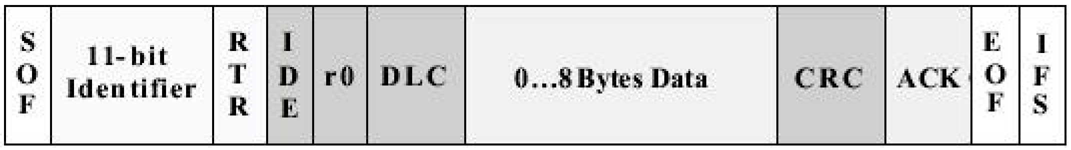
PS. Everything will be written in the documentation.

Analysis

We are going to learn and analyze the CAN protocol. In order to simulate it, we will design a simulation program written in Java or C#. The program will hold entities that represent real components of the protocol. It will also be implemented a GUI part, hence the user will be able to interact the application in an easy and friendly manner. They will be able to test the interaction between some components of a system (a vehicle, for example), based on the CAN protocol. This way, they will understand how CAN works and will have the oportunity to play with it (to provide varied test cases).

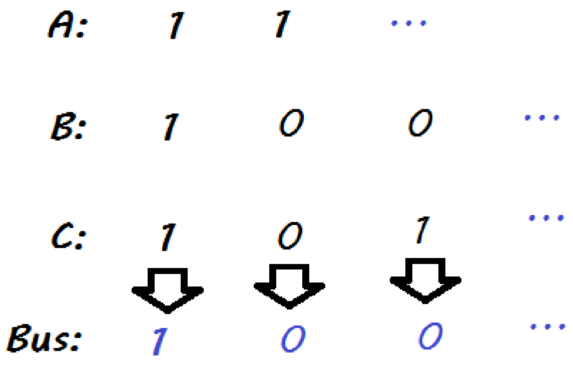


In our simulation, we will have several nodes/ objects (at least 2), which represent components of the system. They are going to transmit and receive messages encoded in Standard CAN Frame (detailed in the Bibiography Study chapter), based on the protocol.



The transmision of the messages is made by taking into consideration the bits transmited. There can be a dominant bit ('0') or a recessive bit ('1'). Let's say that, at the same time, all the components want to send thei messages. However, on the CAN Bus will be written only the most "dominant" message at a time. That is, if all the bits (from all the nodes – same position) transmit '1', then on the Bus will be written '0'; otherwise, if at least one of those bits is '0', then the Bus will take the value '0' and all the nodes that were not trasmitting the value '0' on that particular bit, stop sending the message. In the end, only a message will be sent.

Below it is an exemple of how the transmission of the bits to the Bus should work.



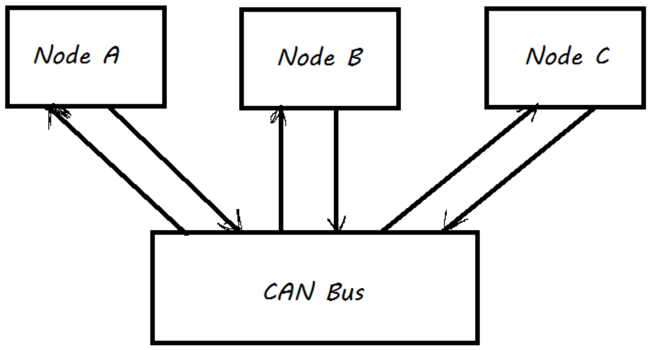
Design

The application will have several Node class objects, let's say 3 (Node A, Node B and Node C), and a Bus class object. The information from the Node classes will be send through the Bus class to the other Node classes, based on the CAN protocol.

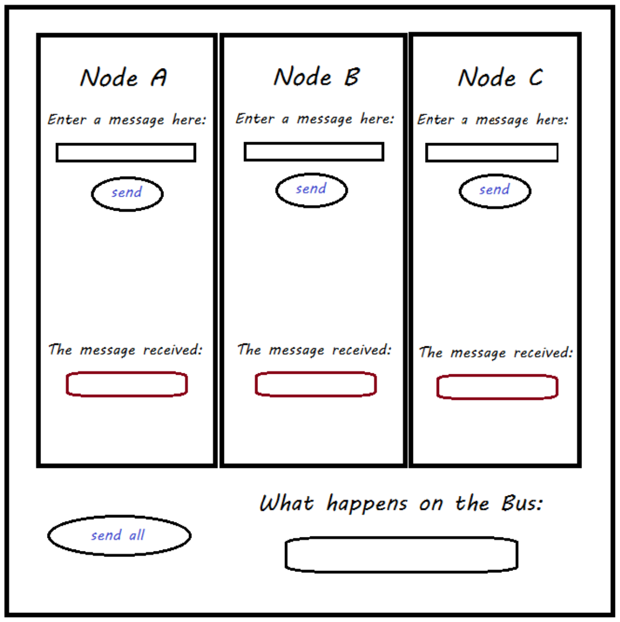
The Node classes will have some attribute, as the Standard CAN Frame require: SOF (1 bit), Identifier (11 bits - array), RTR (1 bit), IDE (1 bit), r0 (1 bit), DLC (4 bits), Data (64 bits / 2 int), CRC (16 bits / short int), ACK (2 bits), EOF (7 bits), IFS (7 bits). For simplicity, a bit will be stored as a boolean value.

For the first test of functionality, we only need the Identifier attribute (which represent the message). The rest of the attributes will play a role in further implementations. Therefore, for the basic simulator, the Bus class will only have the message / identifier attribute.

Below it is represented a basic representation of the class diagram.



The simulator consists of a single web page. The user can easily work with it. The goal is to build the application as interactive as possible. An initial model of the user interface is presented in the sketch below.



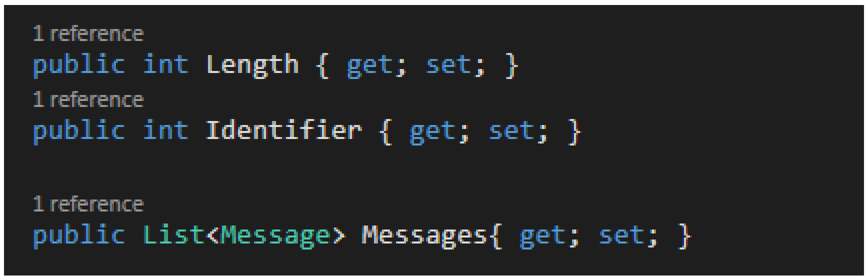
There will be displayed several textboxes, one for each Node in the system. Once the user introduces a message (in standard frame) for every node, the application will make some computations and display the result (the data that will be transmited from a Node to the others).

Implementation

The application will be developed in C#, AspNet framework. The project uses the MVC (model-view-controller) architecture. Most of the frontend will be coded in javascript and jquery. As mentioned in the previous sections, the web page will consists of a number (based on the user input) of textboxes (one for each node). Then the user will type in a message for every node. Finally, the system will write the output: the data passed from the priority node (based on the message identifier) to all of the other nodes, through the bus node.

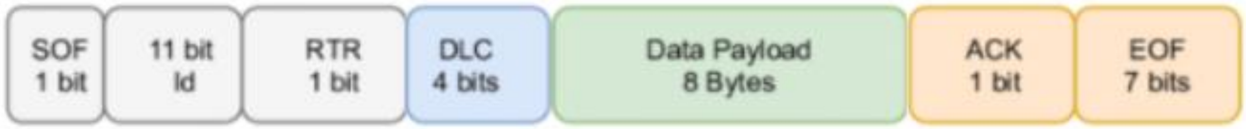
First, we have to decide on some object types. In the application, there will be used 3 types of objects: Node and Message.

The Node will have 3 attributs: Length (the number of messages saved on it – received or ready to be transmited), Identifier (ID, unique for each node – 1/A, 2/B, 3/C, …), Messages (a list of all messages currently on the node). A snippet of the model class code is shown in the following picture.



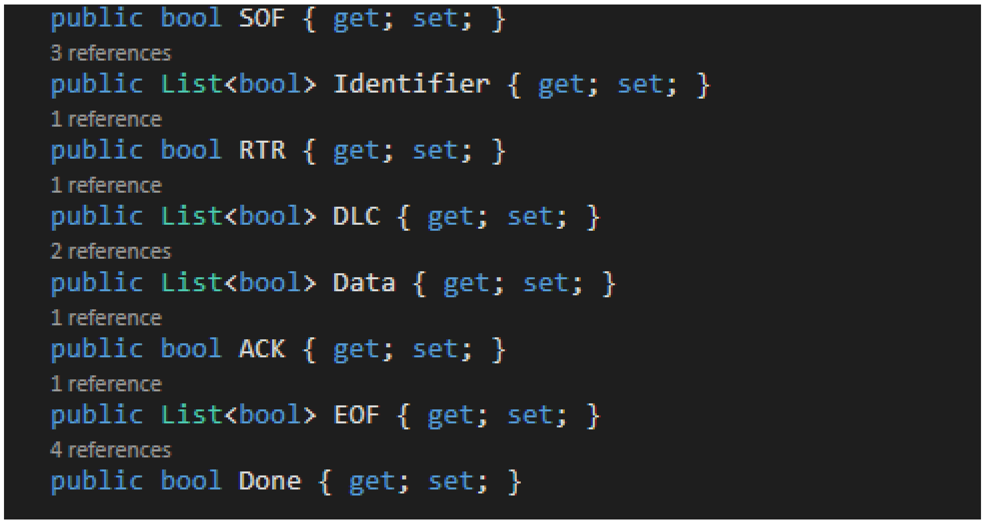
```
1 reference
public int Length { get; set; }
1 reference
public int Identifier { get; set; }

1 reference
public List<Message> Messages{ get; set; }
```

The Message object will hold the data of a message based on CAN protocol. These messages come in 2 shapes (bits arrangement), called frames. For this project, we sticked to the Standard Frame, described in the Analysis Chapter. Also, some of the fields of the frame will not be included in the simulation. Having this into consideration, the frame that will be coded into the Message model, will be a shorter form, as we can observe below.



Therefore, the attributes of the class are:

- SOF (Start of Frame bit). It indicates the start of the message. It is used to synchronize the nodes on a bus. For example, the frame with a dominant bit on this field (‘0’) will have priority in front of the other frames.
- Id (Identifier, on 11 bits). Important role. It determinates which node has access to the bus.
- RTR (Remote Transmossion Request). Specify whether the frame is a data or a remote one. We will only work with data frames in this simulator, therefore the field will be always equal to (‘0’).
- DLC (Data Length Code – 4 bits). Contains the number of bytes being transmitted. Because this is only a simulator, we will only transmit 1 byte = 8 bits on the data payload. This being the case, the value of the field is “0001”.
- Data (8 bits). Used to store up to 64 data bits (8 bytes for this simulation).
- ACK (Acknowledge bit). Firstly, it is on the recessive bit (‘1’). When the message is received correctly by the receiver, the bit is overwritten as a dominant one (‘0’).
- EOF (End of Frame – 7 bits). Marks the end of the frame. Also, indicates an error when dominant (“0000000”).
- Done. Will be used for the algorithms’ logic.


```
public bool SOF { get; set; }
3 references
public List<bool> Identifier { get; set; }
1 reference
public bool RTR { get; set; }
1 reference
public List<bool> DLC { get; set; }
2 references
public List<bool> Data { get; set; }
1 reference
public bool ACK { get; set; }
1 reference
public List<bool> EOF { get; set; }
4 references
public bool Done { get; set; }
```

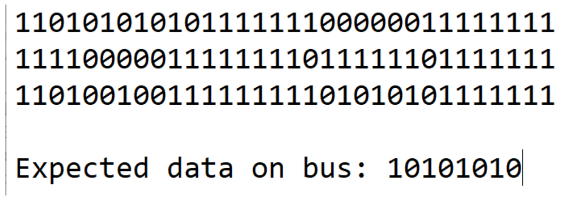
Once the implementation of the objects is set, we can start analysing the algorithms for the simulator. The project will have another layer, called Logics (the “brain” of the application), where all the computations will be made.

The main functions (methods) that will solve the problem of data transmission are:

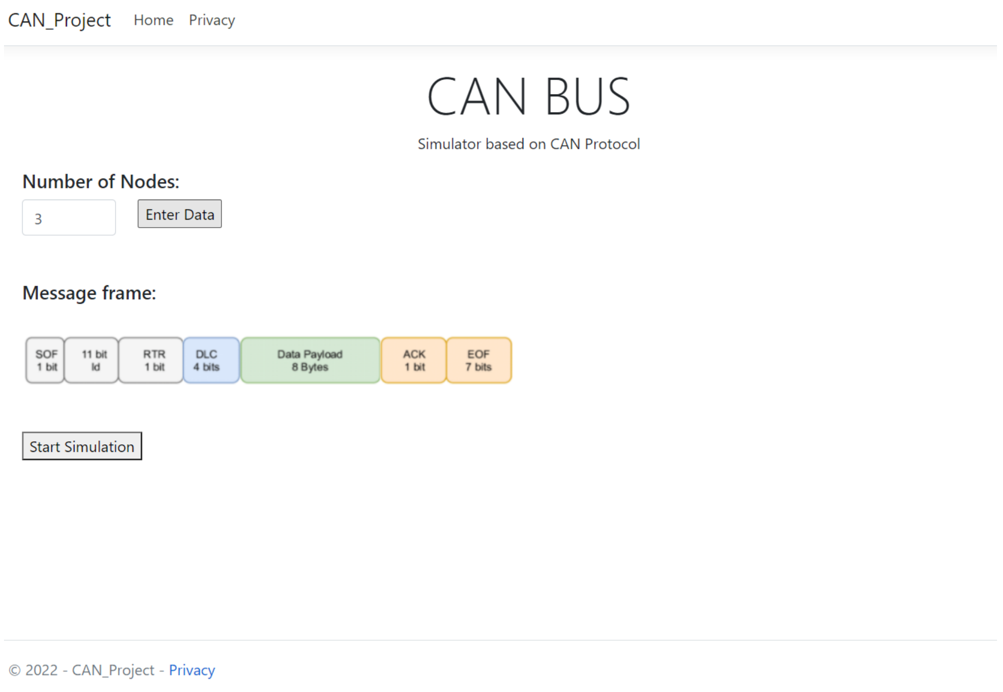
- **InputToNodes.** Here the objects we use in the application are build based on the input from the user. It basicaly converts the input strings of bits into lists of boolean type objects (ones and zeroes).
- **BUSResult.** This function incorporates the most important algorithm of the implementation. It takes the identifier fields of the list of messages (one from each node) and it decides, based on the CAN Protocol, what node should get permission to pass the message. Finally, it saves the data on the BUS (ready to be transmited).
- **BUSOutput.** It simply converts the data from the bus (saved as a list of booleans) into a string ready to be printed as an output to the user.

Testing and validation

In order to properly test the simulator, we prapared some presolved input data. One example of 3 Nodes with a message for each of them is given below (input data and user interface with the result).



```
11010101010111111100000011111111
11110000011111111011111101111111
11010010011111111101010101111111

Expected data on bus: 10101010
```



CAN_Project   Home   Privacy

CAN BUS

Simulator based on CAN Protocol

Number of Nodes:

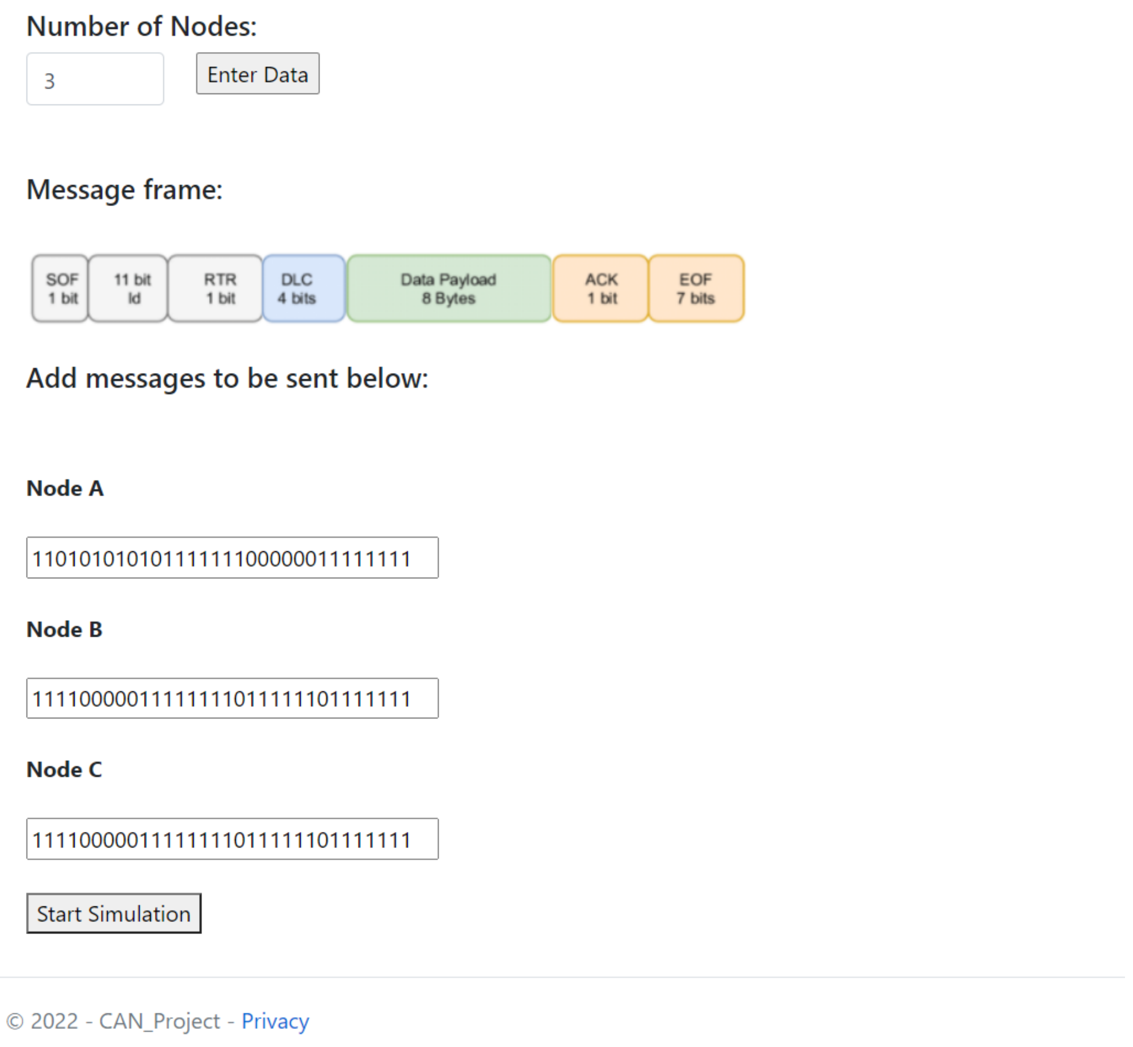
3   Enter Data

Message frame:

Start Simulation

© 2022 - CAN_Project - Privacy

The Interface after introducing the number of nodes. And before starting the simulation.


Number of Nodes:

3

Enter Data

Message frame:

Add messages to be sent below:

Node A

11010101010111111100000011111111

Node B

11110000011111111011111101111111

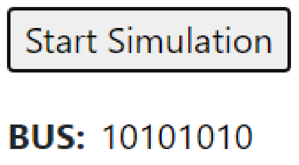
Node C

11110000011111111011111101111111

Start Simulation

© 2022 - CAN_Project - Privacy

The result.


Start Simulation

BUS: 10101010

12

Conclusions

Bulding this project was really interesting. It helped me understand the basics of a technology so much used nowadays, especially in vechicle industry: the CAN Protocol. To be honest, before starting the project, I had no idea what this subject was about and I was not into it either. But now my point of view has changed and from now on I will be more interested in this subject, for sure.

Moreover, the most challenging part of the project was coding the frontend in Jquery. I am sure that all this new knowledge that I gained working on this project will help me a lot in my future work life as an engineer.

Future developments

Obviously, the Simulator Application can be improved. There are a lot of features that can be added to it. For example, all the fields from the standard frame could be added to the message object and used based on the CAN Protocol. Then the extended frame could also be implemented. Besides, the user interface of the application can be built a little more user-friendly, for users that are not very much familiar with the subject.

Overall, this simulator is just a basic version (a prototype) of what it could become. However, I am proud of the result of the project and most certainly will continue to work on it in the future.

Bibliography

- "CAN bus", [Online]. Available: https://en.wikipedia.org/wiki/CAN_bus.
- "CAN protocol: Understading the controller area network", [Online]. Available: https://www.engineersgarage.com/can-protocol-understanding-the-controller-area-network-protocol/.
- Rohini, Shinde. Manisha, Dale. Gajanan, Udas. "CAN based Simulator Design for Vehicles", [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.735.7215&rep=rep1&type=pdf
- "Introduction to can bus and how to use it with arduino", [Online]. Available: https://www.seeedstudio.com/blog/2019/11/27/introduction-to-can-bus-and-how-to-use-it-with-arduino/
- https://www.engineersgarage.com/can-protocol-understanding-the-controller-area-network-protocol/