Controller Area Network (CAN) Elevator Diagnostics & Control Project Report

By: Dinkar Sharma, Anas Yassin, Mike Akl
Course: Engineering Project VI (EECE73125)
Submitted to: Mike Galle and Ali Tehrani
Date: 2017-08-10

Table of Contents

Introduction

The purpose of this report is to describe the development process, including the challenges faced and lessons learned, of the CAN (Controller Area Network) Elevator Diagnostics & Control Design Project. The development process includes phase 1, phase 2, and phase 3. Phase 1 deals with implementation of CAN bus interfacing with CAN nodes on the elevator floors and car controller (CC). In phase 2, this report describes front-end and back-end development of the project website. Lastly, in phase 3 this report covers integration between CAN and supervisor GUI. Each phase encompasses engineering design, implementation, integration, and testing. This report presents challenges during each phase and overcoming those challenges where possible. Finally, the report concludes by outlining outcomes and issues made in terms of the project as a whole.

Project Details

Phase 1

In this phase of the project, a CAN message protocol was designed to allow CAN nodes on the elevator network to communicate. The supervisor is assigned the highest priority of ID = 0x100. This ensures the supervisor has control over the floor and car controller nodes on the elevator network. A plan of CAN node configuration had been created and implemented demonstrating what each floor represents. The CAN node performs the following tasks:

- **Detects the location of the elevator and communicates with all nodes informing the status of the elevator**
- **Determines the up/down requests at any floor and which led is required to light up**
- **Determines the door open/close requests**

A fully functioning CAN network had been created in this phase as well as variety of controlling and controlled devices were involved in the CAN network.

Shared CAN Protocol

Shared CAN Protocol			Shared CAN Protocol								
Node Name	Acronym	Description	Message Name	Transmitter	Receiver	Message ID (hex)	DLC	Byte 0			
								7 6 5 4 3	2	1	0
Master Controller	MC	Process messages from all nodes Command EC to re-position car to specific floor	MC_to_EC	MC	EC	0x100	1		MC_Enable	MC_Floor_Cmd	
Elevator Controller	EC	Process messages from MC and move car to specific floor Read data from position sensor and sends through CAN network	EC_Status	EC	ALL	0x101	1		EC_Status	EC_Car_Position	
Car Controller	CC	Report state of elevator car door Report requested floor on the CAN network Display current car position	CC_Status	CC	MC	0x200	1		CC_Door_State	CC_Floor_Request	
Floor 1 Controller	F1C	Request elevator car Display current car position	F1C_Status	F1C	MC	0x201	1				F1_Call_Request
Floor 2 Controller	F2C	Request elevator car Display current car position	F2C_Status	F2C	MC	0x202	1				F2_Call_Request
Floor 3 Controller	F3C	Request elevator car Display current car position	F3C_Status		MC	0x203	1				F3_Call_Request

Figure 1.1 Shared CAN Protocol Among Each CAN Node On The Elevator Network

Figure 1.1 describes the shared communication protocol developed to communicate with floor nodes, car controller node, and supervisor node. The control bits are specified in byte 0 of the CAN message. These control bits can be manipulated depending on the state of the elevator and incoming floor requests by button presses.

High Level Architecture of CAN Network

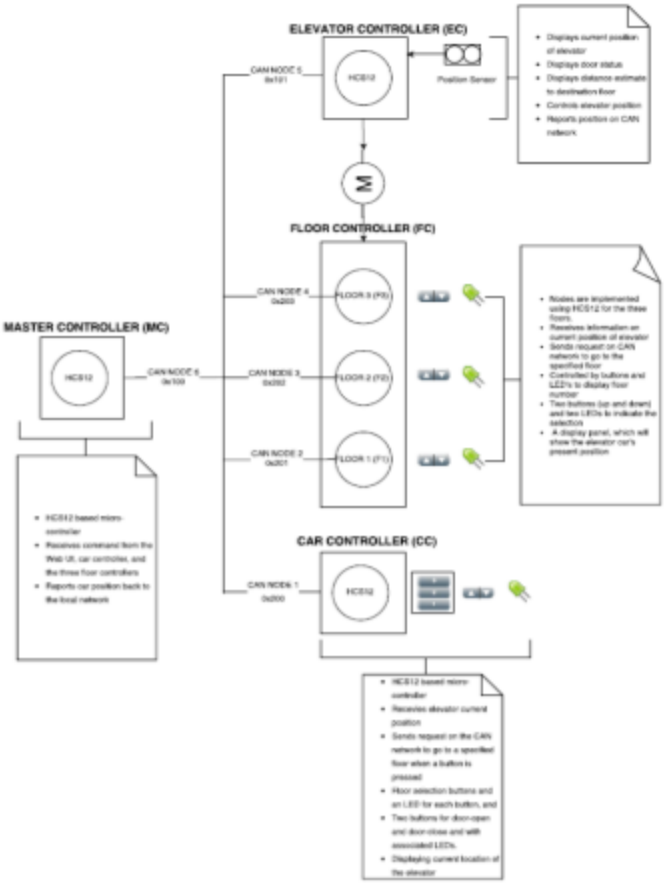

Figure 1.2 High Level Architecture Of CAN Network

Phase 2

In this phase, work on front and back end was completed. It started out with making a website to display the project expectations. As the phase progressed and more web development knowledge was gained, a login page with connection to a MariaDB database was implemented. By the end of the phase, many tables were created to deal with different aspects of the project. A general overview by major milestones will be discussed in this section.
At the beginning of the phase all, the pages were created without use of CSS or bootstrap. The website was very plain and only had logbook entries and some information about the project.

Figure 2.1 - Initial website

By the end of the phase once Bootstrap and CSS was learned the website was completely redesigned. On top of the change in look there were several pages added such as a login page (created with a modal pop up design using Bootstrap library), and request access.

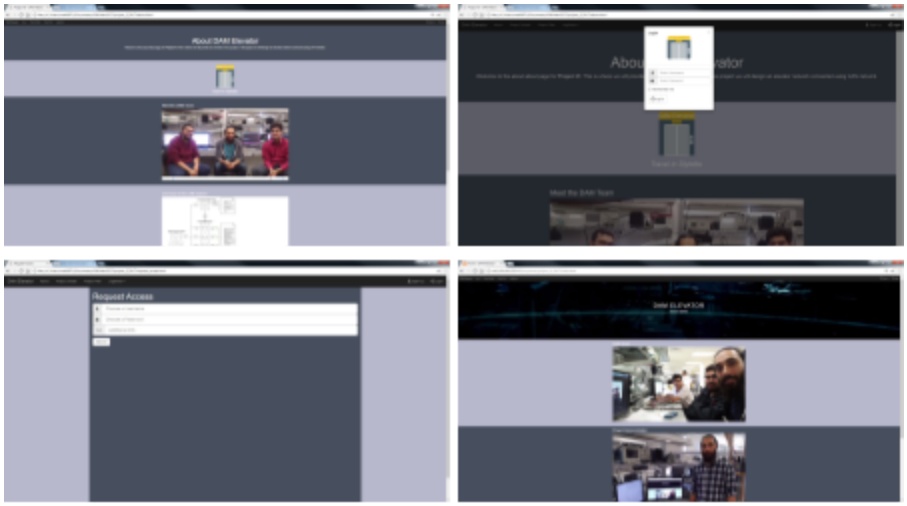

Figure 2.2 - Final design of website with login and request access pages

```
+-----------------------------------+
| Tables_in_elevator_project_2017 |
+-----------------------------------+
| CAN_network                       |
| authorized_users                  |
| cFloor_table                      |
| dState_table                      |
| elevator_network                  |
| queue_table                       |
+-----------------------------------+
```

Figure 2.3 - Overview of the Elevator Database

For login to work a database was created using MariaDB, with an authorized_users table to store usernames and passwords. As shown in figure 3.2 many different tables were created. PDO was used to connect the PHP code to the database. PDO has many methods to be able to make queries and insert data. This proved to be essential in the creation of the elevator control GUI page. The GUI page uses Ajax, PHP and PDO to display the tables to give live information. On every button press there table is updated instantly.

Figure 2.4 - Elevator Control GUI Page

Before applying Ajax on the GUI, every time a button is pressed, the page would reload in order to process the request. Because of that, Ajax was needed to allow the GUI to be updated asynchronously by exchanging data with the database without allowing the whole page to be reloaded.

Phase 3

In this phase, a finite state machine was implemented applying the elevator operation logic. At first, the program is waiting for requests, and then it verifies the status of the door and which floor the elevator is currently. If the door is open, then the elevator waits until it closes; simultaneously, if the requested floor is the same as the current floor, then it waits until another request is received. Once a different floor is requested and the door is closed, the elevator will start moving to the requested floor and open the door. After moving, the requested floor becomes the current floor and the elevator will be waiting again for another requests.

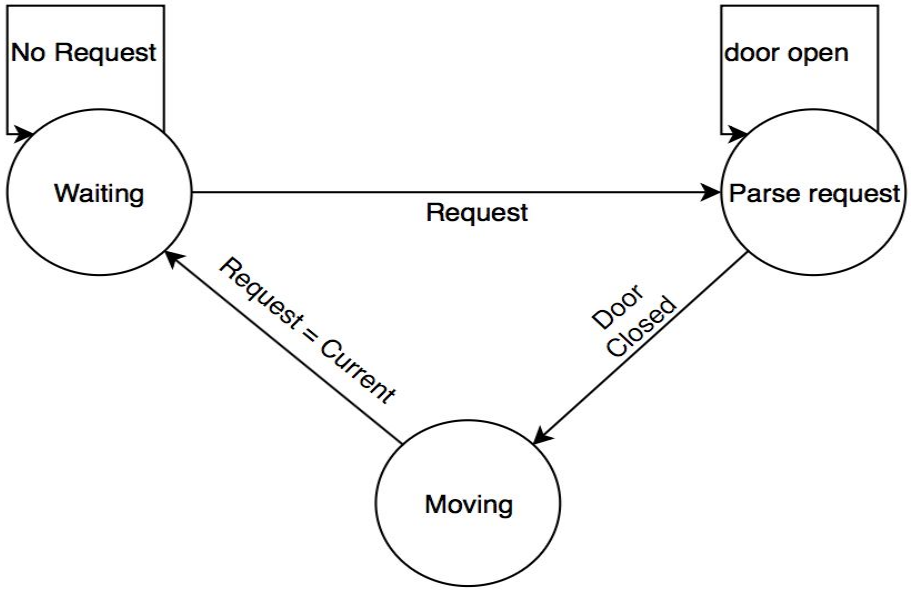

Figure 3.1-State machine of the elevator

The state machine was implemented using two programs that are intended to run all the time as a supervisor node, the supervisor code and queue code. The supervisor code is always reading from the CAN bus, waiting to get a request. Once a request is made, the supervisor code applies some logic to understand who the message is received from and which button has been pressed. Once the message has been dissected and understood the supervisor code inputs it into several tables (CAN_network, and queue_table) in the database. The queue code then takes the request from the queue_table and sends it on the CAN bus to the motor_control. In addition, door state and current floor are being updated on tables,

dState_table and cFloor_table respectively, from the hardware. The motor_control node sends out a message on the CAN bus updating the current floor every 500 milliseconds. The supervisor code then updates the table. Same logic applies for the door state except it comes from the axman board, car controller. If a request comes from the GUI, it is inputted directly into the table, queue_table. Once the request has been inputted into the queue, it works just like before. Coming from the GUI the request is inputted into the elevator_network table.

Issues

The program was not able to process all the requests if they were received while the elevator is moving, so a queue was needed to hold the requests and process them later when the elevator stops moving. A queue_table was created to hold the requests while the elevator is in action, so once the elevator stops moving and ready to process a new request, the program will process the first request in the queue_table. After processing it, the request will be deleted and the second request will become first in the queue_table. However, deleting the processed request was a problem for the supervisor since the only way to delete a row is to count how many rows in the queue_table and delete the first one instantly after processing it. To solve that, a built in method called rowsCount() in the MySQL C++ connector used to determine the number of rows in the queue_table, and then delete the first row always after processing.
Another issue that arose during testing the GUI was that every time a button is pressed, the GUI would refresh the page to process the request. However, this problem was solved by implementing Ajax, which allows the website to read/write to/from the database without refreshing the page.

Conclusion

Overall, the project was successfully completed and the deliverables were met. In order to have a successful project, each team member required knowledge and understanding of the architecture revolving the elevator network. Different programming languages were learned such as HTML, CSS, PHP, and C++. All in all each individual member of the group thoroughly enjoyed this project.

References

[1] Semester 6 Team, *CAN (Controller Area Network) Diagnostics Design (CDD) Project Charter***. 2017.**