Controller Area Network Diagnostic Design Project Report

Kevin Hartwig

James Sonnenberg

Ovidiu Ofrim

Conestoga College

Abstract

This document provides an overview of the three phases and overall outcomes of the Controller Area Network Diagnostic Design project, in partial completion of the Engineering Project VI course, part of the 3rd-year Electronic Systems Engineering program at Conestoga College. Phase I of the project represented CAN communication between nodes and an elevator controller. Phase II contained website design and database communication of the elevator system. Phase III consisted of the integration of the above phases, all phases were completed successfully and on time. The objective of controlling a CAN networked elevator platform through a website interface was achieved.

Controller Area Network Diagnostic Design Project Report

In this report, a detailed overview of the three phases of the project is provided along with a summary of the outcomes of each phase. The final outcome of the project was the design and implementation of a three-floor elevator system using a Controller Area Network (CAN) for communication. This system is comprised of

- three floor nodes, each representing the external face of the elevator at each floor
    - ability to call the elevator to that floor, turning on an LED
    - LEDs that show the current location of the elevator car
- a car controller node, representing the internal control panel of the elevator
    - LEDs that show the current location of the elevator car
    - buttons to open and close the elevator door
- a linux supervisor that monitors the CAN Network[1] and controls elevator through use of a state machine
- a web based user interface that allows control of the elevator remotely

[1] Further references to 'CAN Network' will be abbreviated to 'CAN' to avoid encouragement of RAS syndrome

Phase I

Phase I involved project planning (not detailed in this report), some preliminary setup for the Controller Area Network (CAN), and design and implementation of the CAN.

CAN Setup

Getting CAN communications up and running was the first step before creating the nodes in order to achieve a working elevator. This was accomplished by installing the appropriate drivers and software for the PCAN-USB adapter. Using two PCAN adapters and computers, transmitting capabilities across the CAN bus was tested on all nodes. After confirmation that all nodes were capable of sending and receiving CAN messages, experimentation began on communication from a PC to an Axman[2] board. This process involved writing C code that initializes the CAN communication link that is on board the Axman. Once a stable communication link had been established, the C code was adapted to handle communication between two different Axman boards. This included setting up filters for choosing which node IDs are of importance.

Starting with the CAN protocol, nodes were assigned responsibilities and node IDs. The CAN system layout and message protocol designed are shown in Figures 1 and 2.

[2] Freescale HCS12 development board

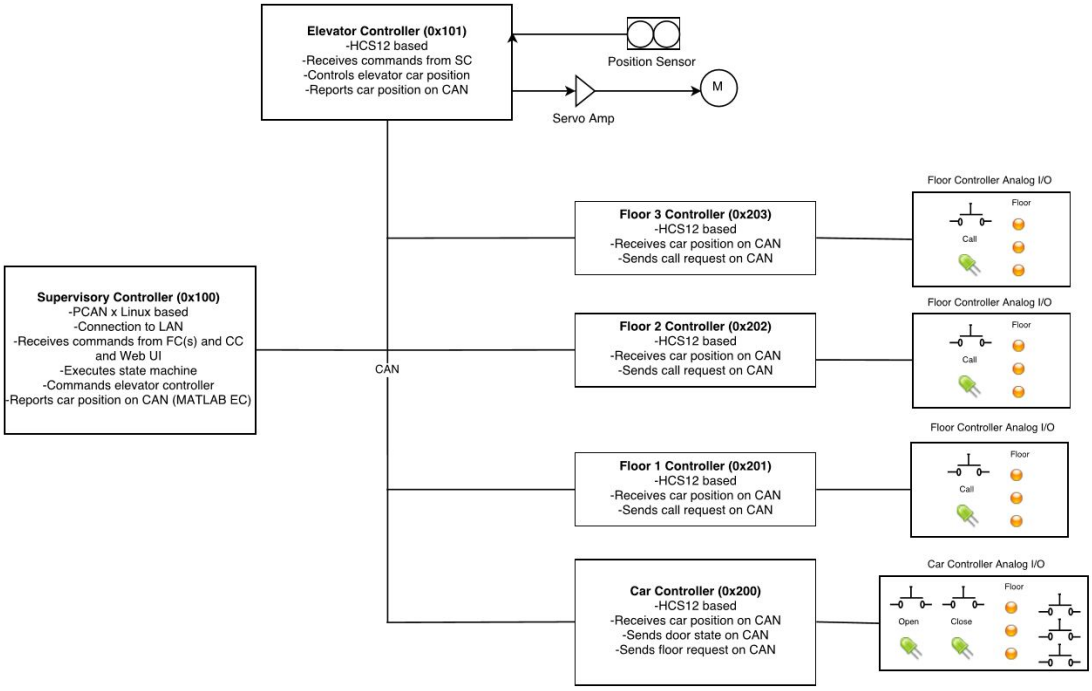

Figure 1: *Diagram of CAN communications*

Shared CAN Protocol - Message Layout

Message Name	Transmitter	Receiver(s)	Message ID (Hex)	DLC	Byte 0							
					7	6	5	4	3	2	1	0
SC_Cmd_To_EC	SC	EC	0x100	1						SC_Enable	SC_Floor_Cmd	
EC_Status	EC	All	0x101	1						EC_Status	EC_Car_Pos	
CC_Status	CC	SC	0x200	1						CC_Door_State	CC_Floor_Req	
CC_Position	SC	CC	0x200	1						CC_Door_State	SC_Current_Floor	
F1_Status	F1	SC	0x201	1								F1_Call_Req
F2_Status	F2	SC	0x202	1								F2_Call_Req
F3_Status	F3	SC	0x203	1								F3_Call_Req

Figure 2: CAN message layout

Floor Nodes

Three floor nodes were created, each of them being represented by a unique Axman HCS12 board. The nodes represented the call button for the elevator on each floor. Each of the nodes was assigned a message ID of 0x201, 0x201, and 0x202, so that the floors could send the appropriate message ID when requesting a floor. Four LEDs were placed on each of the nodes, three of the LEDs represent each of the floor nodes, so that the user can see which floor the

elevator is currently on just by inspecting the node. The fourth LED represents the elevator call button, a user can request the elevator car from a floor node by pressing the button, the LED will stay lit up until the elevator arrives to the current node. All three nodes listen on the CAN network for the elevator car position, and transmits a floor request message when the user requests a floor.

Car Controller

The car controller was additionally run on an Axman. The responsibilities of the car controller were to mimic that of what you would see inside of a real elevator. This included selecting a floor, opening and closing the door, and LEDs for feedback. The car controller works by only accepting messages from the supervisor and only sends messages back to the supervisor. The messages sent to the supervisor were the requests of the user as well as the associated door state, this way the supervisor is able to queue up the requested events and know when it is able to start executing the tasks based off of the door state. The car controller listens to the supervisor so that it can receive periodic updates about the current position of the elevator and update its LED display information.

Linux Supervisor Design & Implementation

The purpose of the Linux Supervisor is to monitor the CAN and control the movement of the elevator car. Monitoring of the CAN is achieved through a USB-CAN adapter called PCAN, as discussed previously. The control of the elevator car movement is achieved through interpretation of signals on the CAN bus from the floor nodes and car controller that feed a finite state machine.

Program Flow

The flow of the main application loop is straightforward, consisting of two main sections:

1. A polled read on the CAN bus
2. A step through a state machine

Before this loop is entered, an initialization phase must be successfully completed. This phase consists of initialization of the PCAN interface, global parameters used to track elevator position, target position, door state, a three element char array[3] used as a FIFO queue for floor requests, and a floor request queue index variable. It also initializes variables used by the state machine to track the current state and the current state transition condition (or "action").

Upon successful initialization, the program enters the main application loop where it cycles indefinitely. Interrupt-based reading on the CAN bus was not necessary in this implementation due to the large receive buffer present in the PCAN low level hardware; polling was chosen for ease of implementation and simplicity.

CAN Bus Read

The PCAN receive buffer is read once per iteration of the main application loop. This means that the receive buffer in the PCAN hardware is not necessarily emptied every iteration. However, because the main loop application cycles between only two main tasks (reading the PCAN Rx buffer and taking a step through the state machine), this is not an issue.

If a message is available in the PCAN Rx buffer, it is immediately parsed. During parsing, message ID (corresponding to the sender) and the message data are used to update global

[3] Only three elements are required as there are only three floors and duplicate floor requests are not permitted.

variables appropriately. For example, if a message is received with ID 0x201, a floor request is processed from Floor 2 (see message protocol for more information).

State Machine

A graphical representation of the state machine integrated in the Supervisory Control program can be seen in Figure 3. There are four states: WAITING, MOVING, DOOR OPEN and DOOR CLOSED. Transitions between states correspond to certain conditions, of which there are six: Floor Arrival; Door is closed; Door is opened; Dequeue is invalid; Dequeue is valid; and Repeat. The transitions in the block diagram are numbered to correspond with the state transition table (see table 1). In program initiation phase, the initial state is set to WAITING, and thus this is the state machine entry point.

The state machine was implemented using a function pointer/lookup table approach. The design steps are outlined below.

1. Functions were defined for each state. Each function executes a short procedure in which it checks global variables and returns a condition.
2. A lookup table was defined to relate transitions between states and the condition required for those transitions (table 1).
3. The condition returned from a state function is then passed along with the current state into a function which accesses the lookup table and determines the next state.

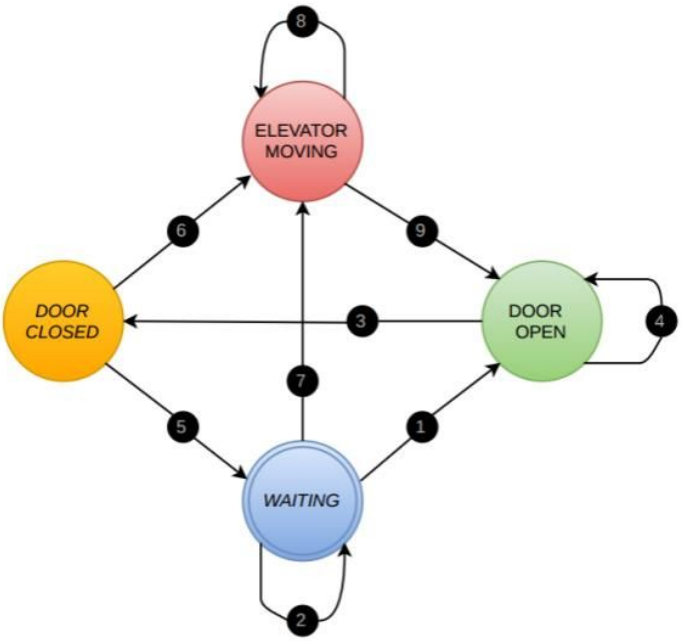

Figure 3: State Machine Block Diagram

#	Present	Condition	Condition Source	Next
1	*Waiting*	*Door is opened*	*CC*	*Door Open*
2	*Waiting*	*Repeat*	*-*	*Waiting*
3	*Door Open*	*Door is closed*	*CC*	*Door Closed*
4	*Door Open*	*Repeat*	*-*	*Door Open*
5	*Door Closed*	*Dequeue Invalid*	*SC (Self)*	*Waiting*
6	*Door Closed*	*Dequeue Valid*	*SC (Self)*	*Moving*
7	*Waiting*	*Dequeue Valid*	*SC (Self)*	*Moving*
8	*Moving*	*Repeat*	*-*	*Moving*
9	*Moving*	*Floor Arrival*	*EC*	*Door Open*

Table 1: State Transition Table

Phase I Outcomes

Functioning CAN, including

i. Three floor nodes

ii. One car controller node

iii. Linux supervisor

Phase II

Phase II involved designing of the web application (Control UI), project website, and design and implementation of an assortment of database tables for communication between the web application and the CAN through the Linux Supervisor.

Project Website Design

The project website is stored on github in a repository. The site contains a home page detailing information about the project members, a project plan page outlining the project schedule and a project details page that goes further in depth regarding the project requirements. A user can register for the website using the signup page, registering to the site places your username and password in the memberInfo table. The login page uses SQL to search the database for the memberInfo table to locate the correct username and password for authentication. Javascript is used to check for similar usernames in the database and to check for an appropriate password length. The site features a GUI interface for controlling the elevator, this page is only available to registered users. A menu bar created using bootstrap is used to navigate around the site.

Control UI

This page of the website was responsible for handling user requests to control the elevator. This page uses a combination of common web development languages including HTML, CSS, Javascript, and PHP. The base template of the page was created with mobile users in mind and has automatic resizing capabilities to adjust for various screen sizes though the use of bootstrap features. This control page can only be accessed by registered accounts and uses sessions to manage this. When a registered user access the page they are greeted with a

welcoming message custom to their username and are to view live information about the elevator. There are 8 buttons total on the control UI which represent all of the buttons on the nodes created in Phase I of the project. This includes the ability to request the elevator from the floor nodes (outside the elevator), open and close the elevator door and request the floors from the car controller (inside the elevator). The buttons sent the request to the supervisor program by updating fields of tables in a shared schema. The tables that were used for writing these requests included *clientRequests,* and *clientQueue.* The clientRequest table consists of a row for each possible request with an adjacent column representing the boolean status. When the status was set to 1 the request was active and queued; when the status is 0 when the request was inactive. The clientQueue table was used to build the queue order, when a request was pushed that event was immediately added to the clientQueue and the supervisor program would read the entries in this table in order from the time they were added to build up its internal queue. When the supervisor finished adding this request it would delete the entry from the table and wait for more request to be added. The page also displays information about the current status of the elevator such as the current position and where the elevator is going to go next (the queue). This was accomplished by reading *currentState* table in the shared schema. This table was updated by the supervisor whenever there was a significant change in the elevator state. There is also a bunch of logging and debugging information (all messages on CAN bus) displayed on the page in a scrollable window. Which was also accomplished by reading a table, the *CANlog* table and displaying its entire contents in the scrollable window.

All of the features on this webpage are dynamic and do not require to reload the page in order to view updates, this was accomplished through the use of Ajax. Ajax has allowed us to

provide live updates of the elevators status and show all messages sent over CAN in real time. The buttons that control the elevator also have used Ajax as a way to disable the button if the request has already been queued (status = 1), as well as change its appearance. Through this approach of changing the buttons based off of database table values the buttons will also change to their appropriate state when their action has been queued by the hardware on its associated node.

Database Reading and Writing: Linux Supervisor Integration

In order to communicate with the UI Control interface, database read/write capabilities had to be added to the Linux Supervisor. This was achieved through implementation of the C++ Connector module from MySQL (see Appendix A). The sample programs offered with the module for reading and writing to a database were modified and adapted for implementation to the Supervisor program.

The overall updates to the program were as follows:

- The supervisor updates the *CANLog* table whenever the system changes in a significant way. Significant changes include change of state machine state and specific signals on the CAN bus.
- The supervisor reads from the *clientQueue* table (representing floor requests from the Control UI) and queues them in the floorRequests queue if valid. As with any FIFO queue, the dequeued item is removed from the list.
- When the elevator car gets to a new floor, the supervisor updates the *clientRequests* table with boolean FALSE (0) for the associated floor's request IDs.

- Modification to the main application loop: every iteration, the Supervisor truncates (clears) and writes the current car position and door state to the *currentState* table.

Phase Outcomes

1. Functional project website featuring:
    i. Information about the team and project
    ii. Individual member logs of weekly project progress
    iii. Members only control page (web application) with the ability to control the elevator and display diagnostic CAN information
2. Read/Write capabilities added to the supervisor
3. Schema created with the following tables (see Appendix B for graphic description):
    i. memberInfo → holds info of member credentials (username, password)
    ii. currentState → holds most recent values for car position and door state
    iii. clientRequests → a row for each of the 7 request IDs and boolean
    iv. clientQueue → FIFO queue of client requests
    v. CANLog → log of all CAN activity

Phase III

All that remained at this point in the project was testing and validating the functionality of both the Control UI and all CAN components as a complete system. Apart from a few code bugs that were quickly identified and rectified, everything functioned as expected.

Phase Outcomes

Full system integration:

i. Fully integrated web based application capable of elevator control and display of CAN diagnostic information through the Linux Supervisor using MySQL databases.

Overall Project Outcomes

This section briefly overviews the higher-level project outcomes, distinct from the individual phase outcomes defined so far in the report.

Outcome 1: Complete System Design and Implementation

The most obvious outcome of this project is the successful design and implementation of a CAN with accompanying remote and diagnostic control capabilities. This required careful planning and task distribution to ensure completion within the designated timeframes for each phase.

Outcome 2: Acquired Knowledge Relative to the Field of Work and Study

In order to complete the tasks required for completion of this project, each member was required to study and put into practice engineering and design principles. For example: research, methodology, preliminary & detailed design, implementation, verification and validation, personal contribution and collaboration, and engineering logging. Additionally, skills for employability and professional responsibility were also developed. These include: work ethic, professional conduct, autonomous learning and project scheduling.

Conclusion

In regards to the graded project outcomes, it is the collective opinion of the group that performance and results met expectations. The knowledge acquired and applied during the duration of the project was a valuable addition to each group member's repertoire of engineering and professional skills. Both Ali Tehrani and Michael Galle are cordially thanked for their oversight and contributions during all phases of the project.

Appendix A

List of links related to the project or accompanying references made in this report:

Link Description	Address
C++ Connector for MySQL	https://dev.mysql.com/downloads/connector/cpp/
Project website[4]	https://jsonnenberg6500.github.io/ProjectS6/
Project website repository	https://github.com/Jsonnenberg6500/ProjectS6
Project code repository ● Elevator nodes code ● Linux supervisor code	https://github.com/kevin-hartwig/CAN-Node-Code

[4] Due to restrictions imposed by GitHub regarding PHP, login and signup capabilities are not available. This in turn restricts access to the Control UI.

Appendix B

This appendix documents the project schema's tables' structures with sample content. Cells that are gray are ***permanent***, meaning these cells make up the structure of the table necessary for the system to function. White cells with italicized content is sample information that is updated by the system during operation.

memberInfo

username	password	email	bio
admin	*password*	*admin@elevator.com*	*I'm the admin*

CANLog

date	time	floorQueue	carPosition	targetPosition	doorState	signalID
2017-08-03	*16:40:32*	*2,1*	*MOVING*	*FLOOR 3*	*CLOSED*	*101*

currentState

currentPosition	doorState	other
1	*0*	*0*

clientQueue

activeID[5]
4
5

[5] See Appendix C for definition of requestIDs which are inserted into this table

clientRequests

requestID[6]	status[7]
1	*0*
2	*0*
3	*1*
4	*1*
5	*0*
6	*0*
7	*0*
8	*1*

[6] See Appendix C for definition of requestIDs represented by each row/index
[7] For the status, "1" represents an active request and "0" represents no active request.

Appendix C

Definition of requests in the *clientRequests* table:

requestID	Description
1	Request Floor 1 (from outside)
2	Request Floor 2 (from outside)
3	Request Floor 3 (from outside)
4	Select Floor 1 (inside elevator)
5	Select Floor 2 (inside elevator)
6	Select Floor 3 (inside elevator)
7	Door Open
8	Door Close