Visualization of Volvo Buses Chassis Electronic Systems via CAN-Bus

Adam Lundström
2016

Bachelor of Science in Engineering Technology
Automotive Engineering

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
Visualization of Volvo Buses’ chassis electronic systems via CAN-bus

Adam Lundström

Luleå University of Technology
Dept. of engineering sciences and mathematics

June 2016
This report is for my bachelor thesis within Automotive Systems Program. I chose this program because of my interest in both vehicles and electronics. The program has an interesting mix of mechanical and electronic courses which makes it a very complete program.

This project felt huge from the beginning and the demands were set high. The progress was slow from the beginning but as time passed the progress went faster and faster. The final result exceeds the demands for this project.

During this project I have especially developed my skills in programming and learned new strategies of constructing systems. I have also learned a lot from working abroad including expressing myself in language other than my native language and have been introduced to a foreign culture.

I would like to thank the following people for making this project possible.

**Bengt Lundström:** Provided important contacts for this project and arranged all trips and residences in Poland.

**Wojciech Romanowski:** My mentor whose knowledge was essential about the buses, electronics and software.

Finally, I want to thank my classmates and everyone that has supported me during this project.
This thesis presents the development of a tool to visualize states and variables of chassis systems on Volvo Buses. The Electronic Control Units (ECUs) of the chassis systems are interconnected via Controller Area Networks (CAN). The systems are separate and the three networks are independent, but interconnected through a Master Control Module (MCM). The tool can simulate a bus on a desktop or be used for software verification of a real bus.

The tool was developed on a 180 MHz ARM microcontroller to which a 7” Thin Film Transistor (TFT) Liquid Crystal Display (LCD) visualizes the data through text and pictures.

Within the project, only key CAN messages were used to interface with the bus and only two push buttons and a potentiometers are used as user interface. These successfully demonstrated the functionality of the tool. Further development would include making use of the touch screen capabilities of the chosen display and add more CAN messages ability.
Sammanfattning


Verktyget är baserat på en 180 MHz ARM mikrokontroller. En 7” Thin Film Transistor (TFT) Liquid Crystal Display (LCD) är ansluten till mikrokontrollen. Displayen visualiserar data genom text och bilder.

viii

SAMMANFATTNING
Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECU</td>
<td>Electronic Control Units</td>
</tr>
<tr>
<td>CAN</td>
<td>Controller Area Network</td>
</tr>
<tr>
<td>MCM</td>
<td>Master Control Module</td>
</tr>
<tr>
<td>SFF</td>
<td>Standard Frame Format</td>
</tr>
<tr>
<td>EFF</td>
<td>Extended Frame Format</td>
</tr>
<tr>
<td>SOF</td>
<td>Start Of Frame</td>
</tr>
<tr>
<td>RTR</td>
<td>Remote Transmission Request</td>
</tr>
<tr>
<td>IDE</td>
<td>Identifier Extension</td>
</tr>
<tr>
<td>DLC</td>
<td>Data Length Code</td>
</tr>
<tr>
<td>ACK</td>
<td>Acknowledgement</td>
</tr>
<tr>
<td>EOF</td>
<td>End Of Frame</td>
</tr>
<tr>
<td>LCD</td>
<td>Liquid Crystal Display</td>
</tr>
<tr>
<td>TFT</td>
<td>Thin Film Transistor</td>
</tr>
<tr>
<td>SD</td>
<td>Secure Digital</td>
</tr>
<tr>
<td>ADC</td>
<td>Analog to Digital Converter</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interface</td>
</tr>
<tr>
<td>SDIO</td>
<td>Secure Digital Input and Output</td>
</tr>
</tbody>
</table>
## Contents

Preface iii  
Abstract v  
Sammanfattning vii  
Acronyms ix  

1 Introduction 1  
1.1 Background ................................................. 1  
1.2 Purpose ..................................................... 2  
1.3 Aim .......................................................... 2  
1.4 Limitations ................................................. 3  

2 CAN-bus 5  
2.1 Introduction to CAN-communication ..................... 5  
2.2 A CAN-message ............................................. 5  
2.3 CAN-bus physical layer .................................. 8  

3 Hardware 11  
3.1 Development board ....................................... 11  
3.2 Screen ...................................................... 13  
3.3 MCM .......................................................... 13  

4 Software 15  
4.1 Introduction to software development ................. 15  
4.2 Timers ....................................................... 16  
4.3 Interrupts .................................................... 17
CHAPTER 1

Introduction

1.1 Background

On a Volvo Bus, the electronic influence is gigantic. Compared to an automobile the bus has to cover almost all of the comfort and safety systems such as active safety, heating and air condition but for 10 times the passengers. It also has to take care of additional features such as recordings from surveillance cameras. All communication between different systems are handled via CAN-bus as this is a very effective way of transporting information between systems.

A Volvo bus has divided its electronic network into few separate independent systems. One of the systems takes care of features like operating the doors and interior lighting while the other system operates for example the engine and ABS system. The systems are independent of each other thanks to that they only exchange information that is necessary. If one component would break the other features will still work. This layout however causes problems for software verification, which is hard to achieve without the whole bus. Using the whole bus is both inconvenient and time consuming and a simulation of the bus’s system would be beneficial. This approach creates the possibility to have a virtual bus which would greatly ease the process of software improvement and development.
1.2 Purpose

This bachelor thesis is completed for Volvo buses. As electronic components are getting faster and cheaper more and more mechanical features are swapped for electronic. There are also environmental and safety demands on these types of products that requires more technical implementation. This results in high pressure on software developers to create stable and reliable software with new features. At Volvo, it is important to be leading in technology and to sustain Volvo’s position in the market. A fast, easy and efficient solution to test and simulate software is required.

1.3 Aim

The requested solution is to have a tool that simulates the bus chassis electronic system for both reading and transmitting messages or CAN-frames. The tool shall be able to read information from the bus’s Master Control Module (MCM) and display information. The MCM is connected to the tool and the 2 other existing CAN-bus networks. Information transmitted from the MCM is supervised by Volvo staff. Information shall be presented in a well-organized and efficient way to a LCD display connected to the tool. A user interface shall be implemented so that the user will have the ability to change values of the signals that is transmitted. The user interface shall consist of two buttons and one potentiometer. A system outline is presented in figure 1.1.

![Figure 1.1: System outline.](image)
1.4 Limitations

The time assigned for this project has led to dividing the project into steps. My responsibility is to create a base in which the amount of signals transmitted and received are reduced and focused on the most important ones for the task. The signals that shall be transmitted are status of parking brake, vehicle speed and ignition status. The signals that shall be displayed on the LCD are door status front and rear, and status of the passenger compartment high temperature alarm.
CHAPTER 1. INTRODUCTION
CHAPTER 2
CAN-bus

2.1 Introduction to CAN-communication

CAN-bus is a serial data communication bus and is used for broadcasting data between nodes on a network. The CAN-bus was first developed by BOSCH 1986 and has become one of the most successful network protocols used in vehicles. Today it is standardized since several years under ISO 11898 [1]. CAN-bus can transmit in speeds up to 1Mb/s but is limited by the networks length. CAN-bus communication is largely implemented in the vehicle industry. This is because of its capability to withstand a tough environment and the fact that it can provide messages fast and reliably. CAN-bus also allows more technically advanced features to be added compared to conventional solutions and at the same time reduce the amount of wires [2].

In contrast to other networks, CAN-bus is not a master and slave network where node A communicates to node B. CAN-bus broadcasts messages to the entire network, which means that every node on the network will acquire these messages. This makes the network less vulnerable for faulty nodes. If one node would break, it is automatically dropped from the network and communication is continued with the functioning nodes [3].

2.2 A CAN-message

The CAN-bus operates in an opposite logic state when broadcasting messages. Usually a logic high is associated with a one and a logic low with zero but the CAN-bus operates in a different way. In CAN-bus a dominant state which corresponds to a high voltage difference is equal to a logic zero and a recessive state
which corresponds to a low voltage difference is equal to a logic one. Signalling is handled by two wires called CANH and CANL. They each represent a high voltage respectively a low voltage which is altered depending on what state is transmitted. When a node is transmitting a recessive state or when nobody obtains the bus, both CANH and CANL converges to \( \approx 2.5V \). When a node is transmitting a dominant state the CANH increases \( \approx 1V \) to \( \approx 3.5V \) and CANL decreases \( \approx 1V \) to \( \approx 1.5V \). [3]

The two states are presented in figure 2.1.

![Figure 2.1: CANH and CANL dominant and recessive state.](image)

Because it is a voltage differential measured by the system it is very robust against noise. As the two wires will be equally affected the voltage difference will be intact and therefore no data is lost. It is even possible to transfer message if one of the wires CANH or CANL has been cut. The flow of current in each wire is equal but in opposite direction. This creates a magnetic field cancellation around the wires which also helps reduce noise.[3]

The inverted logic state is used to avoid collision between messages on the bus, as one node is broadcasting a message on the bus it also listens. To determine which message is the most critical one every message consists of a preprogramed identifier. The identifier also lets the receiver know from where the message is broadcast and what it contains. The identifier then has a priority range where the lowest number has the highest priority. This is because of a logical high is dominant and a logical low is recessive. If two nodes are transmitting simultaneously and one node is transmitting 1 while the other is transmitting 0, the one that transmits 0 always wins the CAN-bus’s attention. This is because a dominant state will overwrite a recessive state. This is called bit arbitration when two or several nodes transmit simultaneously and has to distinguish the most critical message based on the identifier. This is done automatically by the CAN-bus controller. A schematic picture of two nodes transmitting simultaneously is presented in figure 2.2.
Two different types of CAN frames exist, the first one is called Standard Frame Format (SFF) and the other one Extended Frame Format (EFF). In SFF the bit fields transmitted are called:

- Start Of Frame (SOF)
- Remote Transmission Request (RTR)
- Identifier Extension (IDE)
- Data Length Code (DLC)
- Data
- Cyclic Redundancy Check (CRC)
- Acknowledgement (ACK)
- End Of Frame (EOF)

A CAN frame starts with one bit that announce beginning of transmission, this is called the Start Of Frame (SOF) bit. This is then followed by the identifier which is 11 bits long. The next bit is the Remote Transmission Request (RTR) which specifies if additional information form another node is needed. From here the other nodes need to know in which form the message is sent, therefore the Identifier Extension exists (IDE). In SFF configuration the bit is dominant. The next 4 bits is called the Data Length Code (DLC) and specifies how many bytes of data is transmitted, the range is 0 to 8 bytes, then the data is transmitted. To make sure the data was received correctly the message is followed by a Cyclic Redundancy Check (CRC)
Redundancy Check (CRC) that contains the number of bits transmitted. This is now the end of the message but to make sure all data is received the node that transmitted now awaits a dominant state from the other nodes. This is called the acknowledge bit (ACK). This is then followed by the End Of Frame bit (EOF).

In extended frame format the RTR bit is replaced with the Substitute Remote Request (SRR) and after the IDE bit the additional 18 bits identifier is transmitted. This is the main difference between the two formats is the length of the identifier. In the extended frame format it is total of 29 bits while the standard frame format is 11 bits. [3]

Both formats are presented in figure 2.3 and 2.4.

![Standard ID CAN frame](image1)

![Extended ID CAN frame](image2)

**2.3 CAN-bus physical layer**

The CAN controller also supports an option to filter incoming CAN frames [4]. In this particular setup it could be used to specify the frames that contains the requested information. Then all other frames with an identifier that does not fulfill the filter would not be received. This would reduce the amount of information that the microcontroller has to process. Although it would aggravate further development when more signals will be implemented and the other CAN frames will also be necessary. The filtering is instead managed by software and is described further in chapter 4.

In an ordinary CAN-bus system with several nodes the wires CANH and CANL can become quite long depending on what speed is set for transmit. The CAN-line in a bus covers a large distance due to the size of the bus. When distance
become long a possibility emerge that frequencies generated through CAN-line reflects back and creating noise. To prevent this the ISO 11898 standard has implemented termination in the form of two resistors, one at each end of the CAN-line. They each have a resistance of 120 Ω. Figure 2.5 displays a complete CAN-bus network with the resistors placed at each end.

Figure 2.5: CAN-bus network.
3.1 Development board

Microcontrollers are widely used in many different automatic controlled systems such as smart phones, electric tools, toys and remote controls. A microcontroller is generally cheap next to a full size processor and also has a lower power consumption. A development board with a microcontroller is in many ways very alike an ordinary personal computer with reduced performance. It contains a microprocessor, memory and peripherals. A microcontroller usually lacks a operating system and uses embedded or dedicated software. The main application area for microcontrollers are in embedded systems within a larger electrical or mechanical system.

Volvo Buses supplied a development board for this project. The requirements for the setup were that it should be fast, contain lot of storage capability and it should also support two separate CAN-interfaces for future development. The STM32F429IG was the development board that was chosen and is shown in figure 3.1.

The wiring diagrams for the development board is included in appendix A.

3.1.1 Technical specification

The development board is built by HAOYU Electronics and the platform is split into two modules. The top module contains the microcontroller and flash storage. The 32-bit ARM Cortex-M4 core microcontroller is made by ST. The Microprocessor supports a clock speed of 180MHz. This quite high clock speed results in a fast microcontroller that is able to handle large amount and challenging operations. The storage capability which also was one of the deciding factors is 128
CHAPTER 3. HARDWARE

MB large flash storage, 32 MB SD-RAM and 4 MB S-RAM. That is plenty of space for software and means that it will have good potential to be extended in future.

![STM32F429IG presented from above](http://www.elty.pl/pl/p/Plyta-rozwojowa-z-STM32F429I-GT6-Ethernet-CAN-RS485/1741)

Figure 3.1: STM32F429IG presented from above.

The bottom module contains all of the platforms ports and power distribution. The development board supports several connections including:

- One micro USB-port for downloading software to memory.
- One micro USB-port for USART communication.
- Two push buttons and one potentiometer
- Two CAN-interfaces.
- One micro SD-card reader.

Another reason in favour for this particular development board is the support of the micro SD-card reader. This can provide a lot of storage that will be needed for example to store images, which shall be displayed. The amount of images can easily become huge when the touch panel will be implemented in future as all buttons on a touch panel will correspond to an image. Section 2.3 discusses the importance of termination in the CAN-line, however with this development board a 120 Ω resistor is built in before the CAN transceiver. This eliminates the need of a resistor at the development boards end of the CAN-line, although an additional resistor is required at the other end. However, the setup now requires that the

---

development board is placed at one of the ends of the CAN-line. If the system would be extended it will require that additional nodes will be placed either in between or at the other end.

More information about the microcontroller can be found on [5].

3.1.2 User interface

The two physical buttons and the single potentiometer on the development board can act as the user interface. The potentiometer alters the voltage from 0 to 3.3 V simply by turning it, this means that it can obtain several values and is easy to change. That makes it suitable to select vehicle speed that shall be transmitted to the MCM. The two push buttons can theoretically do everything as they are only dependent on software. They are however suitable for the two remaining signals, parking brake status and ignition status.

3.2 Screen

A display is needed to present information to users and was acquired separately. The display is a 7-inch thin film transistor liquid crystal display (TFT LCD) shown in figure 3.2. It is built by HAOYU Electronics, supports resolution of 1024 by 600 pixels and also has a built in touch panel. The touch panel is capacitive type and supports up to 5 touch points. In future development the touch panel will be implemented, which will eliminate the maximum amount of buttons that now is limited to two. Communication between microcontroller and touch panel is through I²C which is often used in short distance between lower speed peripherals and microcontrollers.

3.3 MCM

The MCM is the master control module to which the microcontroller is connected to. The MCM acts as a gateway between the independent systems on the buses. In a Volvo bus 3 different CAN-bus networks exists. Network 1 controls signaling from bus body, the top half of the bus. Network 2 and 3 controls signaling from the bus’s chassis. Network 2 and 3 has different CAN protocols.

The MCM is sending several frames including status about the front and rear passenger door and the passenger compartment high temperature alarm. These signals needs to be read, stored in memory and displayed on the LCD by the microcontroller.
Figure 3.2: LCD screen\textsuperscript{2}.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Figure32.png}
\caption{LCD screen.}
\end{figure}

\textsuperscript{2}http://elty.pl/pl/p/10.1-1024x600-TFT-LCD-Display-with-capacitive-touch-panel/1634
CHAPTER 4

Software

4.1 Introduction to software development

The programming language chosen is C mainly because I have encountered it before in previous courses and felt familiar with it. Partly because it is a well-established general purpose language that easily can be customized. The environment for software development is CoIDE from CooCox. Coocox is based on Eclipse, which I also encountered before. Coocox is a free software development program with support for this specific microcontroller and development board.

My main priority when I developed this software has been to ease further development. I have organized the code into modules for different features to keep the main loop small. The modules are independent of each other resulting in a reliable structure. This is called modular programming and is often used for larger projects. Additionally, creating functions with multiple use. This means using macros instead of creating a function that only deliver the value for a particular point in the data. The macro could be called from the main loop or any other function. The macro would take in which frame and where the desired data is located. The macro will then return value of the data for the user to store. The layout is in such a way that it shall be easy to understand and to continue development.

The code that is explained in this chapter can be seen in appendix B. Due to confidentiality reasons bit placement and details about CAN-bus specific details has been removed.
4.2 Timers

As almost everything in software is dependent on time it would be good to keep track of it, that is why timers exist. If you for example would like to control an LED to flash in a specific frequency, a timer is essential for this use. A timer uses the systems core clock speed to count each cycle. Although the system core in this setup performs 180 million cycles per second a way to control it is necessary. It is done through selecting prescaling and period of the timer. The system core clock will be divided by the assigned prescaler value. Now the frequency is reduced and can be reduced further by selecting a period. This determines when the timer is ended up, the frequency determined by prescaling is incrementing the period.

For example, if a timer should be set to 1 second first a value to divide system core clock needs to be decided. One option would be to divide it by $180 \times 10^6$ and end up with 1Hz frequency. This is not possible in this configuration due to the prescaler variable is declared as 16 bit in the memory. This means that maximum value that can be stored in this variable is 65535. To prevent end up with an odd frequency the prescaler is set to 45000 which corresponds to 4 kHz as shown in equation 4.1.

$$\frac{\text{System core clock}}{\text{Prescaler}} = \frac{180 \times 10^6}{45 \times 10^3} = 4 \times 10^3 \text{ Hz} \quad (4.1)$$

The frequency of 4 kHz results in that the LED will flash every 250 $\mu$s and to achieve correct frequency the period time must be set. The period selects the number of cycles the timer shall make. The period time is also a 16 bit variable and therefore is also limited to 65535. For this case the current frequency of 4 kHz needs to be converted to 1 Hz. With the period time set to 4000 it is incremented at a rate of 4 kHz. The final frequency when the timer has ended becomes 1 Hz. This type of timer is called hardware timer and is very accurate counting. There are however, a few limitations with it. If for example the microcontroller should be able to function as an actual timer. Then it would have to count minutes and maybe hours too. It is not possible to set this type of timer to count minutes or hours by altering the prescaler and period time. Although it is possible to count when the timer has ended up. This is called overflow and is treated like a counter for how many times the timer has ended up. Another limitation is the fact that the number of hardware timers available is limited.
4.3. INTERRUPTS

4.2.1 Software timers

Another way of counting long periods of time or when the application does not require a very precise timing, is to use a software timer. Software timer uses a hardware timer as base for counting in a specific frequency and then increments preprogramed variables every time the hardware timer ends up. A software timer benefits from having multiple usage with only one hardware timer. Another benefit is that several variables can have different time intervals. For example, the hardware timer has a frequency of 10 Hz that corresponds to 100 ms. One variable could be set to 100ms and another to 300 ms. Both variables can be counted simultaneously by the software timer.

Apart from hardware timers the software timer does not have any interrupt function and have to be constantly checked. Due to this it loses some precision as the main loop could take some time to go through.

4.3 Interrupts

An interrupt is a signal delivered to the microprocessor that something needs immediate attention. In the case of an interrupt the microprocessor halts and saves its current position, executes the interrupt instruction, and continues from where it halted. An interrupt could be triggered by software or hardware. An example of an interrupt would be when a hardware timer has ended up. It will set a bit in the interrupt control register indicating that timer has ended and interrupt shall occur. This bit is called interrupt flag and must be reset after the instruction has been executed. Otherwise the microprocessor will always return to what’s called the interrupt handler which keeps the instructions what to perform when interrupt occur.

An interrupt then must have a priority if two should occur simultaneously or if another interrupt would occur inside an interrupt handler. This is preprogramed for each interrupt. For example, in this setup the CAN receive and transmit has high priority compared to timers. This is to avoid losing any data from receiving or transmitting a CAN frame. Features controlled by a timer can generally wait until the CAN frame has ended.

The general strategy when designing the interrupt is to keep it short and easy to execute. The reason for this is that if the microprocessor will be executing instructions in the interrupt handler and at the same time another interrupt will appear it could miss vital information. There is also a possibility that it could be perceived as slow or not reacting at all. Some benefits with using interrupts are almost immediate reaction. Microcontroller does not have to wait for any specific event instead it can perform other tasks in the meantime. This results in that more
instructions can be performed within the same amount of time thus resulting in a more effective work distribution.

4.4 Input and output

4.4.1 Buttons

If the push buttons are supposed to have the ability to do something first task is to assign the button to the correct pin to the microcontroller. According to the wiring diagram in figure 4.1 key 1 corresponds to port A and pin 0 and key 2 corresponds to port C and pin 3.

The wiring diagram in figure 4.1 also shows that key 1 is connected as a pull down and key 2 as a pull up configuration. The difference between the two is what state the pin connected to the button will be in when it is pressed or released. In a pull up configuration as in key 2, the input pin will be read high when button is not pressed and low or ground when the button is pressed. The pull down configuration as in key 1 is the other way around. When button is not pressed the input pin is connected to ground and when pressed connected to $V_{cc}$.

As shown in the wiring diagram in figure 4.1, the switch is directly in line with ground for key 2 and through a resistor to $V_{cc}$ on key 1. The instant that the switch closes when someone pushes the button small sparks will appear. This creates an

---

unstable situation with voltage peaks referred to as bouncing. Bouncing can cause the input pin to quickly alter between high and low resulting in microcontroller interprets it as that button has been pressed several times when in fact it only been pressed once. In figure 4.2 is a schematic picture of a bouncing signal going from low to high.

![Figure 4.2: Schematic picture of bouncing signal](image)

One method to eliminate this problem is to connect a capacitor parallel to the switch as shown in figure 4.3.

![Figure 4.3: Wiring diagram over key 2 with capacitor.](image)

The capacitor would dampen out the bouncing and create a smooth transition from high to low and low to high on the input pin. However, as the board is already created in the previous way and no additional space exist for including a capacitor, another way will be required. In figure 4.2 it is clearly shown that bouncing only occur for a short time and then stabilizes. This phenomenon can be used in such way that when buttons are pressed and interrupt appear a timer is started in the interrupt handler. When timer eventually ends up the signal has stabilized at either high or low level. This approach will now be described in detail.

According to the datasheet for this microcontroller 7 interrupt lines exist that can be set for different pins. [6] Line 0 connects to pin 0, line 1 connects to pin 1

---

and so on. However, this is regardless which port the pin is located, in this case the pins are different for the two buttons and can therefore use two different interrupt lines. It is also necessary to set the trigger for the interrupt; it means selecting when the interrupt shall occur. It can be set to either rising edge, falling edge or both and choosing between rising and falling edge will determine if the microcontroller will react when a button is either pressed or released. In this application a fast response is priority and therefore it will be set so the interrupt will occur when button is pressed. This means for key 1 the trigger will be set for rising edge and key 2 will be set to falling because of pull up and down configurations. Now the interrupt handler can be created and instructed what to perform when one of the buttons is pressed.

The timer can be started several times because of the bouncing and it will be continuously checked in main loop if it has ended up. When it has ended up it means that the signal now has stabilized and desired operation can be performed. When key 1 is pressed the system shall change status of ignition. When key 2 is pressed the system shall change parking brake status. This is achieved by toggling a variable that then tells the system what status to transmit to the CAN-bus.

### 4.4.2 Potentiometer

The potentiometer located on the development board, is used to change the vehicle speed value, which is then communicated through the CAN-line. Wiring diagram over potentiometer is presented in figure 4.4. Potentiometer marked RT1 in figure 4.4 can alter the resistance from 10 to 50 kΩ. Voltage at the potentiometer is fed to pin 2 on port C. This voltage is now an analog value that first of all needs to be converted to digital. The analog to digital converter (ADC) can provide

---

this feature. This will measure the voltage on the specified pin and convert it to a digital number. The ADC also uses interrupt when the conversion has been completed. A timer controls the interval for conversion. In this setup the timer is set to 5 Hz and means that conversion is started every 200 ms. Which number it will be converted to depends on the resolution of the ADC, which in this case is 10 bits. The range of 10 bits is $2^{10} = 1024$ values so the range equals 0 to 1023. The correlation between voltage and which number is obtained can be calculated by equation 4.2.

$$\frac{\text{Resolution}}{V_{cc}} = \frac{\text{ADC value}}{\text{Voltage over potentiometer}}$$  \hspace{1cm} (4.2)

The speed value will be treated as km/h by the MCM and therefore a more appropriate range of values based on project prerequisite from Volvo is needed. Changing the range is achieved by dividing all values converted by the ADC in the interrupt handler. The value is then copied over to a variable that can set the data accordingly for the CAN-bus transmit.

### 4.4.3 CAN-bus

When all parameters are collected and users have the ability to change values they shall transmit. The parking brake, ignition and vehicle speed all have their own designated place in the data transmitted. Values needs to be shifted to the correct position so that the MCM will recognize it. Additionally, the frame ID must be set to the correct value. All CAN frames shall be transmitted at a given time interval. A timer is used to control when to transmit the CAN frames. For further development when the amount of CAN frames transmitted increases the demand for a logic way to store the data becomes more important. To avoid huge amount of variables declared in different places a structure is a good alternative. A structure allows a group of variables to be stored under a single name. This will then need to be declared as a table to get the opportunity to store several frames. Figure 4.5 shows an example of this, the structure is called CAN_Transmit and includes the variables IDE, ExtID, DLC and data. Each variable can be individually set for each table, which in this case has a length of 3.

Now the identifier, extended identifier, data length code and the actual data can be changed simply by calling CAN_Transmit and select desired frame and variable. This microcontroller has a hardware CAN interface that will take care about bit placement meanwhile the microcontroller can perform other instructions. The table can then easily be extended to desired size although the buffer for transmission have to be considered. If one CAN frame is currently transmitted and the system gives command to send another frame the latest will be stored in the buffer. The
buffer will store frames until they are transmitted and then remove it. This buffer has a size of 3 frames which means that if the application requires more than 3 frames the system will need to wait for a free space in the buffer.

The same configuration applies to receiving frames. Here a structure called CANReceive is used with same variables as CANTransmit and is also declared as a table. When a CAN frame arrives all data is stored in the buffer and when message is completely received an interrupt is set. When the interrupt occur, all data is copied over from buffer to the structure and the buffer is cleared ready to accept new messages. Here the only limitation of amount of frames possible to receive is the microcontrollers memory as long as the microcontroller is fast enough to clear the buffer. When all data is copied over to the structure, the microcontroller can begin to process the information. Desired information can be copied from data variable to a separate variable. This method is used as the data variable in CAN is 8 bytes long and to isolate information that is required a smaller variable is used. This variable could then be used to visualize the status, for example the doors.

### 4.4.4 SD-card

The micro SD-card reader provides large storage capability for the microcontroller. Here additional data for example identification for CAN frames could be stored in future development. The micro SD-card also allows to swap out data in an easy way as it can be taken out and connected to a computer. For this project the micro SD-card will be used to store pictures that shall be displayed on the LCD.

The micro SD-card consists of 8 pins and these represent data which is located on pin 1, 7 and 8. Ground on pin 6, supply voltage on pin 4, clock frequency on pin 5 and command line on pin 3. Pin 2 has multiple use as it is used both for data transfer and for detection. Communication between microcontroller and micro SD-card reader is usually through serial peripheral interface SPI, or SD-mode. This microcontroller is using the secure digital input/output interface, SDIO to communicate through SD-mode with the micro SD-card. This is a strict master
and slave network protocol where messages are matched to clock frequency provided by the microcontroller. The microcontroller will transmit commands to the micro SD-card and await response.

At initialization of the micro SD-card the clock is set to 400 kHz for compatibility reasons. 74 clock pulses are then transmitted to the micro SD-card which allows it to enter normal operating mode ready to accept commands. Microcontroller will then send the command for initiate initialization process and await a response. If the response is successful, the clock speed can now be increased to 24 MHz to minimize time for reading and writing to the micro SD-card. Data can now be transferred from the micro SD-card to the microcontroller by sending the command and address for data. A schematic picture of initialization of the micro SD-card is shown in figure 4.6

![Figure 4.6: Schematic picture of initialization of SD-card.](image)

### 4.5 Visualization

To present the information gathered from the MCM in an effective way the TFT LCD is used. Status of passenger compartment high temperature alarm, vehicle speed and door status should be displayed clearly. This is presented in text on the display. The development board supports a RGB interface that connects to the LCD. The microcontroller places information that shall be displayed in a buffer located in the SD-ram. Here the LCD is fetching the information and presents it. Supplied with the screen is a driver with basic functions for displaying characters and set individual pixels to different colours. This function is used for presenting status of previous mentioned variables. This function converts characters into a
ASCII. ASCII stands for American Standard Code for Information Interchange where all letters and signs correspond a number. These numbers are then interpreted by computers and microcontrollers to present text. It then sets individual pixels at the correct position to the colour selected that eventually becomes letters. This is a very useful feature as it eases the complexity of presenting text.

The status of passenger compartment high temperature alarm and doors correspond to a certain number sent by the MCM. These numbers carries over to the function displaying them. This function uses a switch case to match this number to correct status. Each case will display a different status when it’s matched by the variable. Each time the status is updated it clears the part of the screen where it had put the text earlier. If a problem would occur and the transmission or receiving of frames this should be presented on the screen. In the case of receiving frames this is achieved by implementing a timer that is reset each time a frame appears. The timer is set that if 3 frames in a row would be missed the timer runs out and a function is called. This function displays information to the user that a problem with the CAN receiving has occurred. For transmitting which is done in an interrupt handler a different approach is necessary. Here the microcontroller counts how many times it tries to place a frame in the buffer. If the amount of times exceeds a pre-set value, a function for displaying that an error occurred is called.

The display should also present headers for each status so that user easily can see what information is presented. This function is separate from the function that updates status, this is to avoid clearing the header out each time the display is updated. Values are continuously checked in the main loop and to avoid flickering of the screen, the display is only updated when values are changed. For the vehicle speed the number which has been converted from the ADC is first checked if it has changed from previous value. If it has changed that means that display shall be updated. The converted value is then carried over to a separate function that converts number to ASCII and then display the value at correct coordinates.

Connected to the MCM are 2 LED’s, one yellow and one red. The yellow one is switched on when the MCM is receiving parking brake active and is switched off when receiving not active or not receiving this frame. The red one starts to flash when MCM is receiving ignition active and is switched off if ignition is off or not receiving this frame. This is used because of early stage confirmation that MCM interpret transmitted information correctly.

Pictures that shall be displayed are expected to be in BMP file format. The expected colour format is 16 bit RGB 565, meaning that the information about the mix of colours red, green and blue is 16 bits long. Red occupies 5 bits, green 6 bits and blue 5 bits. The number of colours it can produce in this configuration is therefore $2^{16} = 65536$ different colours. The usual configuration on most BMP images are 24 bits which equals $2^{24} = 16777216$ different colours. This configuration
4.5. VISUALIZATION

is called RGB 888 because it is 8 bits each for red, green and blue. The pictures need to be converted from 24 to 16 bit in order to display the colours properly. This is done by shifting bits to their correct length and removing the three least significant bits. For example the colour red is converted by the following method. In 24-bit colour configuration the colour red is 8 bits long and needs to be shorten to 5 bits. By right shifting it with 3 on the 8 bits long red colour configuration it removes the 3 least significant bits. Then left shifting with 11 the 5 least significant bits ends up in the correct place. Here the 6 bits long colour green and 5 bits long colour blue is added. Colours are now presented correctly on the screen and the microcontroller is able to fetch and display pictures from the micro SD-card. Most pictures have a background colour of either white or black. Volvo requested that pictures should be displayed without this background colour. After the RGB conversion has been completed an if statement is declared that checks the RGB number of the colour. If the function for displaying pictures is called with the option to not display white or black colour. Then pixels containing this colour is skipped.

During initialization of components the user shall be presented a welcome screen. Welcome screen consists of a picture displaying a Volvo logo and information about the tool and the creator. The welcome screen is displayed in figure 4.6. This picture is displayed just after the TFT LCD has been initialized and a software timer is also started. The timer controls for how long the welcome screen shall be displayed and is set to 8 seconds. This time is added on top of the time it takes for the components to initialize. After the 8 seconds is past the screen is cleared and background colour set to blue and the function for displaying headers is called.

Figure 4.7: Welcome screen presented on the TFT LCD.
Volvo also requested that the tool shall display a picture of their bus that opened and closed its doors according to the information sent by the MCM. Picture of a Volvo 9700 coach bus was provided and placed on the micro SD-card. The picture of the bus is with doors closed and due to missing pictures of open doors with this particular colour, an improvised interior was made. To minimize the time for updating the LCD only a section of the bus is updated. This section covers the door area which is the only area the LCD needs to update. In figure 4.7 and 4.8 the bus is displayed with doors open and closed.

Figure 4.8: Bus displayed with door closed.

Figure 4.9: Bus displayed with doors open.
CHAPTER 5

Conclusion

5.1 Summary

The tool, developed through this thesis, provides a good base for Volvo to continue the development. The tool can both transmit and receive CAN-frames and also alter the signals that is transmitted by reading the state on buttons and potentiometer. The tool can visualize the data on the screen both in text and through pictures. The tool can read pictures from the micro SD-card and display it on screen and also convert pictures from 24 bit RGB to 16 bit. If an error with the receiving or transmission of CAN-frames occurs the tool visualizes it on the LCD to inform the user. This is important information for the user as the network will be extended in future and communication problem may occur.

Software verification can be performed today using these signals. Using this tool Volvo will not introduce any changes to the software in the bus. The verification can be performed with the actual software that is used in production which is important to get valid results. The only object that needs adding to the CAN-bus network is the tool itself.

The cost of this tool in this moment is about 125€ which is a small expense compared to the functionality it provides. This tool has potential to significantly reduce the time necessary to verify software. The extra cost for the tool itself is therefore justified with shorten development time.
5.2 Further development

To launch and use this tool at Volvo Product Development it needs further development. Development needs to be continued in the following areas.

- Prepare the additional signals
- Implement touch panel
- Further develop user interface to display more information
- Complete system with more nodes.

The signals received and transmitted at this moment are the most critical ones for the use of this tool. To be able to use the tool as requested the remaining signals needs to be added both for reading and sending. One limitation with this current setup is the amount of physical buttons which is 2. If more signals are implemented the current user interface needs to be redesigned. The touch panel is the most likely component that will be implemented as this does not have any limitation for amounts of buttons. When the amount of signals will be increased the way of displaying information needs to be redesigned or add ability to switch between screens. This is important because of the physical size of the screen is not big enough to display all signals. One way of doing this could be to construct a menu which lets the user choose what information shall be displayed on the LCD.
Appendix A

Wiring diagrams

Figure A.1: Wiring diagram of keys, ADC, LED, storage and power \(^1\).

\(^1\)http://www.haoyuelectronics.com/Attachment/HY-STM32F429IG/SOM-STM32F429IG_SCH_V1.1.pdf
Figure A.2: Wiring diagram of communication interface.

Figure A.3: Wiring diagram of extension interface.
APPENDIX B

Software

```c
void Button_Setup(void)
{
    // Key1
    // Enable clock for GPIOA
    RCC_AHB1PeriphClockCmd(RCC_Key1_Port, ENABLE);
    // Initialize structure
    GPIO_InitTypeDef GPIO_Initkey1;
    // Pin 0
    // Mode input
    GPIO_Initkey1.GPIO_Pin = Key1_Pin;
    GPIO_Initkey1.GPIO_Mode = GPIO_Mode_IN;
    GPIO_Initkey1.GPIO_OType = GPIO_OType_PP;
    GPIO_Initkey1.GPIO_Speed = GPIO_Fast_Speed;
    // Initialize pins on GPIO0 port
    GPIO_Init(Key1_Port, &GPIO_Initkey1);

    // Key2
    // Enable clock for GPIOB
    RCC_AHB1PeriphClockCmd(RCC_Key2_Port, ENABLE);
    // Initialize structure
    GPIO_InitTypeDef GPIO_Initkey2;
    // Pin 0
    // Mode input
    GPIO_Initkey2.GPIO_Pin = Key2_Pin;
    GPIO_Initkey2.GPIO_Mode = GPIO_Mode_IN;
    GPIO_Initkey2.GPIO_OType = GPIO_OType_PP;
    GPIO_Initkey2.GPIO_Speed = GPIO_Fast_Speed;
    // Initialize pins on GPIO0 port
    GPIO_Init(Key2_Port, &GPIO_Initkey2);
}
```

Figure B.1: Initialize port and pin of key 1 and 2.
void EXTILine0Config(void)
{
    /* Enable clock for SYSCFG */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

    /* Set EXTI line for key 1 */
    SYSCFG_EXTILineConfig(EXTI_LineSourceGPIOA, EXTI_PinSource0);

    EXTI_InitTypeDef EXTI_InitStruct;
    /* Key (PA0) is connected to EXTI_line0 */
    EXTI_InitStruct.EXTI_Line = EXTI_Line0;
    /* Enable interrupt */
    EXTI_InitStruct.EXTI_LineCmd = ENABLE;
    /* Interrupt mode */
    EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
    /* Triggers on rising edge */
    EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Rising;
    /* Add to EXTI */
    EXTI_Init(&EXTI_InitStruct);

    /* Add IRQ vector to NVIC */
    /* PA0 is connected to EXTI_line0, which has EXTI0IRQ vector */
    NVIC_InitTypeDef NVIC_InitStruct;
    NVIC_InitStruct.NVIC_IRQChannel = EXTI0_IRQn;
    /* Set priority */
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x00;
    /* Set sub priority */
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x00;
    /* Enable interrupt */
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    /* Add to NVIC */
    NVIC_Init(&NVIC_InitStruct);
}

Figure B.2: Setup interrupt for key 1.

void EXTILine3Config(void)
{
    /* Set PC3 for EXTI line 3 */
    SYSCFG_EXTILineConfig(EXTI_LineSourceGPIOC, EXTI_PinSource3);

    EXTI_InitStruct EXTI_InitStruct;
    /* PC3 is connected to EXTI_line3 */
    EXTI_InitStruct.EXTI_Line = EXTI_Line3;
    /* Enable interrupt */
    EXTI_InitStruct.EXTI_LineCmd = ENABLE;
    /* Interrupt mode */
    EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
    /* Triggers on falling edge */
    EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;
    /* Add to EXTI */
    EXTI_Init(&EXTI_InitStruct);

    /* Add IRQ vector to NVIC */
    /* PC3 is connected to EXTI_line3, which has EXTI3_IRQ vector */
    NVIC_InitTypeDef NVIC_InitStruct;
    NVIC_InitStruct.NVIC_IRQChannel = EXTI3_IRQn;
    /* Set priority */
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x00;
    /* Set sub priority */
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x01;
    /* Enable interrupt */
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    /* Add to NVIC */
    NVIC_Init(&NVIC_Init);
    NVIC_Init(&NVIC_InitStruct);
}

Figure B.3: Setup interrupt for key 2.
/* Set interrupt handlers */
/* Handle PA0 interrupt */
void EXTI0_IRQHandler(void)
{
    /* Make sure that interrupt flag is set */
    if (EXTI->PR & EXTI_Line0)
    {
        /* Start timer of 100ms */
        Timers_100ms[KEY1_DELAY]=KEY_BUTTON_DELAY;
        /* Clear interrupt flag */
        EXTI->PR = EXTI_Line0;
    }
}
/* Handle RC3 interrupt */
void EXTI3_IRQHandler(void)
{
    /* Make sure that interrupt flag is set */
    if (EXTI->PR & EXTI_Line3)
    {
        /* Start timer of 100ms */
        Timers_100ms[KEY2_DELAY]=KEY_BUTTON_DELAY;
        /* Clear interrupt flag */
        EXTI->PR = EXTI_Line3;
    }
}

void ButtonsStatusCheck(void)
{
    if(!Timers_100ms[KEY1_DELAY])
    {
        Key1_Pressed=True;
        Timers_100ms[KEY1_DELAY] = TIMER_STOP;
    }
    if(!Timers_100ms[KEY2_DELAY])
    {
        Key2_Pressed=True;
        Timers_100ms[KEY2_DELAY] = TIMER_STOP;
    }
}

#define RGB565CONVERT(red, green, blue) (int) (((red >> 3) << 11) | ((green >> 2) << 5) | (blue >> 3))

Figure B.4: Interrupt handler for key 1 and 2.

Figure B.5: Function that checks if timer has ended up.

Figure B.6: Macro that will convert 24 bit RGB to 16 bit.
void Print_Welcome_Page(void)
{
    /* Clear the LCD and set black background*/
    LCL_Clear(LCD_COLOR_BLACK);
    /*Control FAT file system on SD-card*/
    if (_f_mount(0, 1, Fat) != _FS_OK)
    {
        GUI_Text(0, 100, "FAT mounting ERROR, check SD Card\0", LCD_COLORYELLOW, LCD_COLOR_BLACK);
    }
    /*Read picture from SD-card*/
    res = _f_open(sfile, "0:\logo.bmp", FILEREAD);
    /*If error occurs*/
    if (res != _FS_OK)
    {
        GUI_Text(0, 150, "Missing FILE 0:\logo.bmp at SD CARD\0", LCD_COLORYELLOW,LCD_COLOR_BLACK);
    }
    /*Display picture on LCD and close it*/
    else
    {
        Display_BMP_File(0, 0, sfile, Y_TOP, R_CENTRE);
        _f_close(sfile);
    }
    /*For information about the tool and creator*/
    GUI_Text(400, 360, ""Volvo Chasei Simulation Tool\0", LCD_COLOR_WHITE,LCD_COLOR_BLACK);
    GUI_Text(440, 360, ""Created By Adam Lundstrom\0", LCD_COLOR_WHITE,LCD_COLOR_BLACK);
}

typedef struct
{
    uint32_t StdId;  /*< Specifies the standard identifier. This parameter can be a value between 0 to 0x7FF. */
    uint32_t ExtId;  /*< Specifies the extended identifier. Frame-ID This parameter can be a value between 0 to 0x1FFFFFFF. */
    uint8_t IDE;    /*< Specifies the type of identifier for the message that will be transmitted. This parameter can be a value of 0x2 CAN_IDENTIFIER_TYPE */
    uint8_t RTR;    /*< Specifies the type of frame for the message that will be transmitted. This parameter can be a value of 0x0 CAN_REMOTE_TRANSMISSION_REQUEST */
    uint8_t DLC;    /*< Specifies the length of the frame that will be transmitted. This parameter can be a value between 0 to 8 */
    union
    {
        uint8_t Data[8]; /*< Contains the data to be transmitted. It ranges from 0 to 255. */
        uint64_t data64;
    }
    uint32_t Time;  /*< Time stamp when data has been transmitted */
} CAN_Transmit;

extern CAN_Transmit CANtx[CAN_TX_BUF_LEN];


