TALLINN UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

Department of Computer Engineering

Chair of Computer System Design

IAY70LT

Software Framework for Fuel Sensor FLS Calibration

Master’s Thesis

Student :Joonatan Renel

Student code : 132327 IASM

Supervisor : Peeter Ellervee Ph.D

Co-Supervisor: Toomas Vinter,

Senior Software Developer

Tallinn

2015

Author’s Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

All works and major viewpoints of the other authors, data from other sources of literature and elsewhere used for writing this paper have been referenced.

…………………………… ……………………………..

(Signature) *(Date)*

Abstract

This final thesis deals with developing software for calibrating fuel sensors. The specific fuel sensors in question were developed by the Estonian company Tehnolabor OÜ. These sensors are meant to be installed on large transportation vehicles and their aim is to enable companies to constantly monitor fuel level in the vehicles' fuel tanks. This technology is mainly intended to collect statistics and information about fuel consumption but it can also be used to detect theft and foul play.

The main purpose of this thesis is to develop and document end-user software to calibrate fuel sensors. Since practically all fuel tanks are physically different, each fuel sensor needs to be calibrated to fit a particular fuel tank. This thesis consisted of developing software for two distinct platforms:

1. Software for a dedicated Calibrator device developed using an Atmega644 microcontroller.
2. Software for Microsoft Windows.

One of the main goal was to make calibration data coherent for the end user. Calibrations are usually performed by ordinary workers who have little or no insight into the inner workings of the fuel sensor. Therefore user interface needed to be simple enough to be understood without specific knowledge. Additionally the user interface needed to be sufficiently robust to be used in an industrial environment. The software needed to be designed so, that it could also deal with erroneous results, incorrect or distorted data and user error.

By the end of the final thesis, the fuel sensors developed by Tehnolabor are currently in production. The calibrator device works in conjunction with the sensors and Windows software is also functioning. The windows application can be used to communicate directly with a fuel sensor, or used with an intermediary calibrator connected in between. As a result of this master's thesis, a new way of calibrating vehicle fuel sensors has been designed and tested.

The thesis is in English and contain 63 pages of text, 7 chapters, 19 figures.

Annotatsioon

Käesoleva lõputöö eesmärgiks on arendada ning analüüsida tarkvara Tehnolabori kütuseanduri FLS kalibreerimise tarbeks. Kütuseandurid on peamiselt mõeldud selleks, et neid saaks paigaldada veokitele ning teistele suurtele sõiduvahenditele. Andurite peamiseks eesmärgiks on võimaldada koguda statistikat kütusekulu kohta pikkadel reisidel. Andureid paigaldatakse ka selleks, et avastada võimalikku kütusevargust.

Kütuseandur FLS on disainitud nii, et seda oleks võimalik paigaldada erinevate suuruste ning kujuga kütusepaakidesse. Andur kasutab kütusetaseme mõõtmiseks pikka kondensaator-varrast. Kondensaatori mahtuvus sõltub sellest, kui suur osa vardast on kütuse sees. Kondensaator on kütuseanduris osa impulss-generaatorist. Kütuseanduri peal olev mikrokontroller loeb generaatori impulsside arvu sekundis ja saab seeläbi hinnata kütusetaset. Mida rohkem pulsse sekundi jooksul mõõdetakse, seda suurem osa vardast on kütuse sees.

Selleks et anduri võtaks arvesse paagi kuju, tuleb andurit kalibreerida. Kalibreerimiseks sisestatakse kütuseandur kütusepaaki ning hakatakse määrama kalibratsoonipunkte. Sisuliselt tähendab see, et paaki lastakse teatud koguste kaupa kütust ning fikseeritakse mõõtetulemus. Paljudel varasematel kütuseanduritel on kalibreerimist saanud teostada ainult nii, et anduriga ühendatakse arvuti. Tehnolabor arendas kütuseanduri FLS tarbeks välja kaasaskantava kalibraatori mida saab ühendada otse anduriga.

Lõputöö üheks peamiseks osaks on tarkvara arendamine ning testimine kalibraatori FLSC tarbeks. FLSC kalibraatorit juhib Atmega 644 tüüpi mikrokontroller, millele kirjutati programm keeles C. Lõputöö käigus arendati välja veel Windowsi keskkonnas toimiv graafiline kasutajaliides, mida saab kasutada koos kütuseanduri FLS-i kui ka kalibraatori FLSC-ga. Graafilise kasutajaliidese peamiseks eesmärgiks on visualiseerida kalibratsioonipunkte ning võimaldada kasutajal neid salvestada ning töödelda. Vajaduse korral saab programmiga sisse viia muudatusi. Graafiline kasutajaliides ehitati programmeerimiskeelt C# kasutades ning seda arendati Microsoft Visual Studio 2012 keskkonnas.

Üheks peamiseks eesmärgiks lõputöö käigus oli luua kalibraatorile ning Windowsile intuitiivne kasutajaliides, mida saaksid kasutada ka lihttöölised. Kasutajaliides pidi olema piisavalt lihtne, et seda oleks võimalik kasutada ka töökojas. Samas pidi liides edastama võimalikult palju infot, et kasutaja saaks tuvastada vigu ning ebatäpseid kalibratsiooniandmeid. Lisaks sellele pidi tarkvara olema ise vigadele vastupidav ning võimeline teavitama kasutajat vigadest suhtluses ning andmetes. Kalibraator pidi vastavalt nõuetele olema ka võimeline püsimällu kalibratsiooniandmeid salvestama ning võimaldada kasutajal neid lugeda. Lõputöö käigus tuli lisaks tarkvara arendamisele seda ka testida ning veenduda, et kalibraator töötab tööstuslikus keskkonnas tõrgeteta.

Lõputöö esitamise hetkeks on kütuseanduri FLS jõudnud tootmisesse ning kalibraator töötab kooskõlas kütuseanduriga. Windowsil põhinev „FLS Interface“ graafiline kasutajaliideson samuti testitud ning toimib. Nii andurit kui ka kalibraatorit saab USB kaabli abil ühendada arvutiga, kuid otseühendus anduriga vajab spetsiaalset vahelüli.

Lõputöö tulemusena on välja arendatud kalibreerimistarkvara kütuseandurile FLS. Seadmed toimivad üksteisega kooskõlas ning on loodud uus moodus, kuidas teostada kütuseanduri kalibratsiooni.

Lõputöö in inglise keeles, sisaldab 63 lehelülge, 7 peatükki ning 19 joonist.

1 Table of Contents

Glossary of Terms and Abbreviations

FLS – Fuel sensor (Official designation by Tehnolabor)

FLSC – Fuel Sensor calibrator (Official designation by Tehnolabor)

Atmega644 – 8-bit microcontroller designed by Atmel

AtXmega16A4 – newer generation 8-bit microcontroller designed by Atmel

EA DOGM162 – Designation for LCD display designed by Electronic Assembly

ST7036 – Controller/driver for the EA DOGM162 LCD display

STK500 – Starter kit for Atmel processors that can be used to program Atmel microcontrollers.

.NET – Software framework for MS Windows, developed by Microsoft

C# - Object oriented programming language developed by Microsoft to be used with the .NET framework.

C – A general purpose imperative programming language.

LCD – Liquid Crystal Display

EEPROM - Electrically Erasable Programmable Read-Only Memory

UART – Universal asynchronous receiver/transmitter

ADC – Analogue-digital converter

DAC – Digital-analogue converter

ISR – Interrupt Service Routine

List of Figures

1. Introduction

The aim of this thesis is to write, test and evaluate software for fuel sensor calibrators. The thesis deals with different approaches to creating a calibration interface. One of the main goals in designing this software is to provide the end user with a simple and robust user interface that would be intuitive and easy to operate. An additional aim is to provide a graphical representation of calibration data that could be saved and analysed in a Windows PC environment.

1.1 Motivation

The FLS fuel sensor has been designed by Tehnolabor OÜ in Estonia [1]. The device is meant to be installed on large transportation vehicles that use gasoline or diesel fuel. The fuel sensor outputs voltage in a given range that corresponds to the measured fuel level. Since virtually every fuel tank is physically different, it is necessary to calibrate the fuel sensor to correspond with the given fuel tank. One of the main reasons why companies install such fuel sensors, is to prevent and discover fuel theft. The main users of these devices are logistics companies.

Calibration is usually done by ordinary workers and therefore it is necessary to provide a simple and robust user interface for them. In the past it has been possible to calibrate sensors using a connected PC. Tehnolabor OÜ has developed a portable calibrator device FLSC to be used for calibration in a factory environment. One of the main tasks as part of this final thesis is to create software for the FLSC.

It was important to create an intuitive user interface for the calibrator. The EA DOGM-162 LCD display only has two lines for text with a width of 16 characters each. This meant that there was very little physical space for displaying data. Additionally the calibrator only has 4 buttons. All this required a very compact and minimal user interface. During the design several solutions were tested. The end solution was inspired by mobile phone interfaces. It used a system of menus and submenus to navigate between calibrator functions. Therefore the interface was somewhat more intuitive and easier to learn than earlier iterations.

Designing the GUI for the windows program was somewhat easier because as a Microsoft Windows program, it had a lot more resources available. The main aim was to create a visual representation of calibration data, which could be easily reviewed and modified by the end user. This would provide an alternative to using the FLSC calibrator as well as produce extra functionality. The user could save calibrations to a PC. It would also be possible to upload calibrations from the PC directly into the fuel sensor. The user interface would have to be simple enough that it could be operated by unskilled workers, but it would also have to provide access to more complex diagnostic functions and be able to monitor raw communications between the PC and the fuel sensor. This would make it easier to debug and monitor the fuel sensor. The fuel sensor FLS is not a part of this master's thesis, however it was developed in parallel with the calibration software. Therefore the calibrator as well as the windows program had to be usable also for the sensor's own development for debugging and analysis.

1.2 Task description

The main goal of this thesis is to create a stable and intuitive user interface for calibration. The thesis deals with writing software for the FLSC calibrator and also for a Windows environment. The goals set for the thesis are as follow:

1. Design, test and evaluate software for the FLSC.
2. Design, test and evaluate software for a Windows environment
3. Explore methods for visualising calibration data
4. Perform and analyse test calibrations on fuel sensors.

Design requirements for the FLSC program:

1. The software must comply with the hardware limitations of the Atmega644 microcontroller. (64 Kbytes of In-System Self-programmable Flash program memory – 2 Kbytes EEPROM – 4 Kbytes Internal SRAM) [2]
2. The software must be able to communicate with the FLS sensor via COM port.
3. The software must be able to maintain a connection with the FLS and notify the user if connection has been lost.
4. The software must be able to communicate with a PC through an USB dongle.
5. The software must provide easy access to calibration functions.
6. The software must enable the user to review diagnostics data
7. The software must be capable of presenting data in both English and Estonian
8. The software must be able to save and load calibration data from the EEPROM.

Additionally it was necessary to develop a driver that will work in conjunction with the DOGM-162 LCD display as well as some functions that allowed the program to access hardware.

Design requirements for the Windows program:

1. It should be possible to install and operate the program on a Windows 7 and Windows 8 operating system.
2. The software should be capable of communicating with the FLS through an USB serial converter.
3. The software should provide a graphical representation of calibration data.
4. The software should be able to detect large errors in calibration data.
5. The software should provide access to diagnostic functions
6. The software should be able to save and load calibration data into files.
7. It should be possible to upload modified calibration data directly into the FLS.

1.3 Methodology

The program for the FLSC Calibrator will be written in C and compiled with Atmel Studio 6.2. [3] The windows software will be written in C#, designed and compiled by using Windows Visual Studio 2012 Express edition [4]. An STK500 will be used to program the Atmega644 microcontroller.

1.4 Thesis Overview

This thesis describes the software structure and functionality. It also contains a description of the FLS device as well as its communication protocol. The thesis outlines the work performed on the software and provides an overview of the common calibration procedure. In terms of software analysis, the thesis also contains descriptions of different classes and functions that have been designed.

Chapters:

Chapter 1 Introduction: Introduction and outline of the task

Chapter 2 Fuel Sensor FLS: This chapter deals with the fuel sensor FLS, its working principle and command interface.

Chapter 3 Calibrator FLSC: This chapter deals with the physical aspects of the calibrator FLSC and gives an outline of tasks that are solved by the software.

Chapter 4 FLSC Software Architecture: This chapter deals with the structure and functionality of the program for the FLSC.

Chapter 5 FLSC Functions: The chapter deals with the functions of the FLSC calibrator and how they are presented to the user.

Chapter 6 FLS Interface Program: The chapter deals with the Microsoft Windows based graphical user interface.

Chapter 7 Conclusion: This chapter presents the conclusion of the master’s thesis.

2. Fuel Sensor FLS

As mentioned before, the fuel sensor FLS has been designed by Tehnolabor OÜ [1] and is intended to be installed on large vehicles. The sensor's primary purpose is to avoid fuel theft and help companies collect statistical data about fuel consumption. The sensor has been designed to be compatible with almost all types of fuel tanks and can be used to measure both diesel fuel and gasoline.

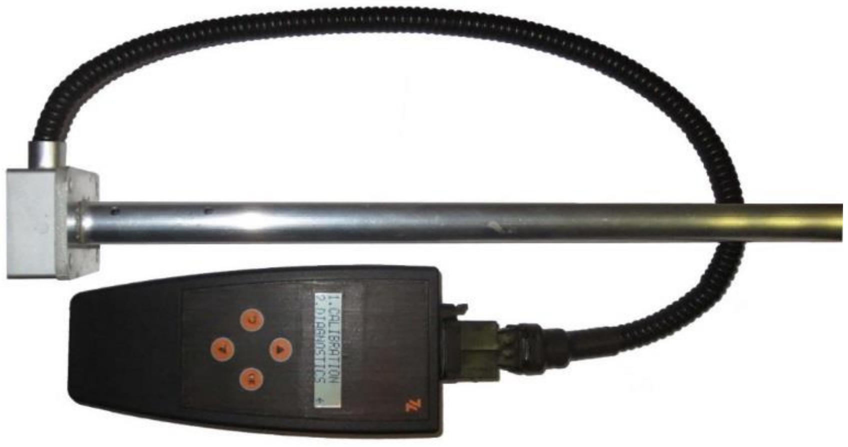


Figure 2-1 Fuel Sensor FLS connected to FLSC Calibrator

2.1 Working Principle

The FLS fuel sensor has a long capacitor tube that is inserted into a fuel tank. If the fuel tank is empty then the environment within the capacitor only consists of air. As fuel is added to the fuel tank, the environment is also filled with diesel fuel or gasoline. This changes the dielectric permittivity of the environment within the capacitor. This in turn changes the capacitance of the capacitor.

The capacitor is connected to an RC generator which produces pulses that are counted by the fuel sensor's CPU. An AtXmega 16A4U Microcontroller serves as the fuel sensor's CPU. In order to get data about the fuel level, the processor counts generator impulses in a given time interval. The CPU calculates the fuel level in litres based on calibration data and outputs this value as voltage and digital information. Calibration data is saved in an EEPROM module that retains data even when power is lost.

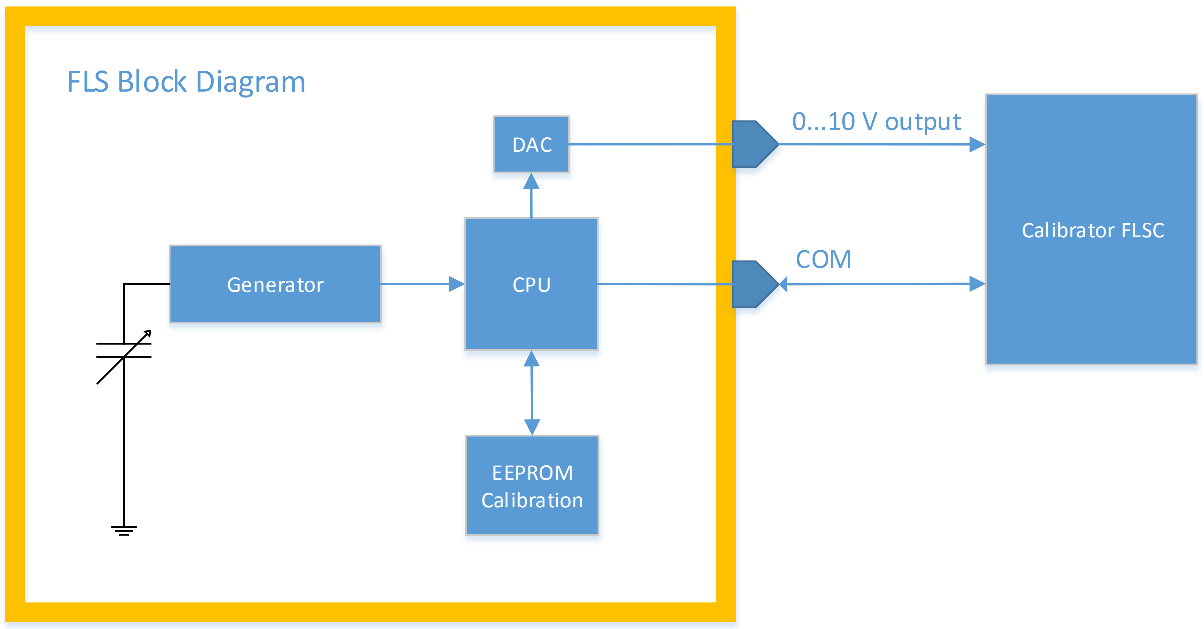


Figure 2-2 FLS Block Diagram

The FLS fuel sensor has two outputs. It has a DC output that gives out voltage typically between 0 and 10 Volts. In its normal configuration 10 volts correspond to a full fuel tank and 0 volts to an empty one. When connected to the sensor, then the FLSC measures FLS output voltage with an analogue-digital converter. The FLS sensor also can report the measured fuel level via COM port as a numeric value

2.2 Communication and Data

In addition to measuring analogue output, the FLSC communicates digitally with the FLS via COM port. In case of using an FLSC Calibrator, data is passed directly between COM ports. If the sensor is connected to a PC, then it is connected via USB port, by using a special converter block in between. The baud rate of the COM port is 9600 Baud/s. The FLS acts as a slave device. It cannot initiate communication itself and does not write anything to the COM port on its own initiative, except for transmitting a “ready” message on activation. The FLS answers to commands that are sent to the device. It does not take more than 200 ms to process a command and send back a reply.

2.3 Command format

Commands that are sent via COM port are in a semi-verbose format. Communication is designed so that it would be easy to debug and read by a human user. All commands that are sent to the FLS begin with the “MK+” prefix. All commands end with a “\r\n” pair of symbols. The FLS responds to a command by first transmitting the original command and then transmits a response followed by -> symbols. If there is a problem with the command then the FLS transmits an “ERROR” message.

Example of communication:

```
FLSC: MK+CAL3=25
FLS:  MK+CAL3=25 -> 2:25L-5447 10.5%  x:5447  z:10.5
```

In this example the sensor is ordered to set the current fuel level to correspond to 25 litres. The FLS responds and transmits the calibration point number, fuel level in litres, measured frequency and how much of the capacitor is submerged in fuel (10.5 %). A full list of FLS commands has been added to the thesis under appendix A.

3. Calibrator FLSC

The FLSC calibrator is designed to be used in conjunction with the FLS fuel sensor. Its main purpose is to provide an easy to use portable device for calibrating fuel sensors. The calibrator has been designed to be robust enough to be used in an industrial environment. The FLSC is intended to be used independently without requiring any external power.

3.1 Working Principle

The calibrator is connected to the fuel sensor FLS by using a special socket. The FLSC provides power for the FLS when connected. The calibrator is powered by two rechargeable lithium-ion batteries that can be charged via USB port. This solution ensures that the calibrator can be used independently with any FLS sensor, so that there is no need to connect extra power to the FLS.

The calibrator communicates digitally with the FLS via serial port. It also measures FLS output voltage with an ADC. The calibrator can also be connected to a PC through a USB dongle. A PC can be used to debug and monitor the calibrator. There is also a special Windows program designed for communicating with the FLSC that is a part of this Master's Thesis. The main purpose of the program is to provide a visual representation of calibration data and enable the user to make modifications.

3.2 Design Decisions

This design was chosen firstly because it supports existing hardware. There are already devices on the market that measure voltage and transmit or record fuel data. [5] The Atmega644 microcontroller that drives the FLSC is an older CPU, but it was readily available and powerful enough for the calibrator. Also there were development tools available that had been tested. The final version of the FLS used a newer AtXmega 16A4U controller, however it was decided not to upgrade the FLSC so that existing CPUs could be utilised. In any case the Atmega644 is sufficient for the FLSC although newer CPUs are currently better supported.

The FLSC was designed with a goal to minimise power consumption so that the calibrator could operate without an external power source. This meant that the clock frequency of the CPU was intentionally kept lower than the maximum frequency of 32 MHz. It was clocked from an external crystal oscillator with a frequency of 7.3728 MHz.

3.3 User Interface

The calibrator has four buttons: OK, BACK, UP and DOWN. A two line LCD display provides information for the user. OK and BACK buttons are used to navigate menus and access different functions of the calibrator. The button OK is also used to pass information to the fuel sensor. The buttons UP and DOWN are for changing parameters and selecting options in menus.

In some modes, it is possible to use the OK button to open a dialog on the display. Pressing and holding the button for at least 3 seconds will open a dialog on the lower half of the display. This can give the user access to special commands and is also used to confirm some user actions.

3.4 Functionality

The calibrator has four basic purposes. Its primary function is to allow the user to perform calibrations on an FLS device and store calibration data. It also provides diagnostic functions and performs power management for the device itself and a connected fuel sensor.

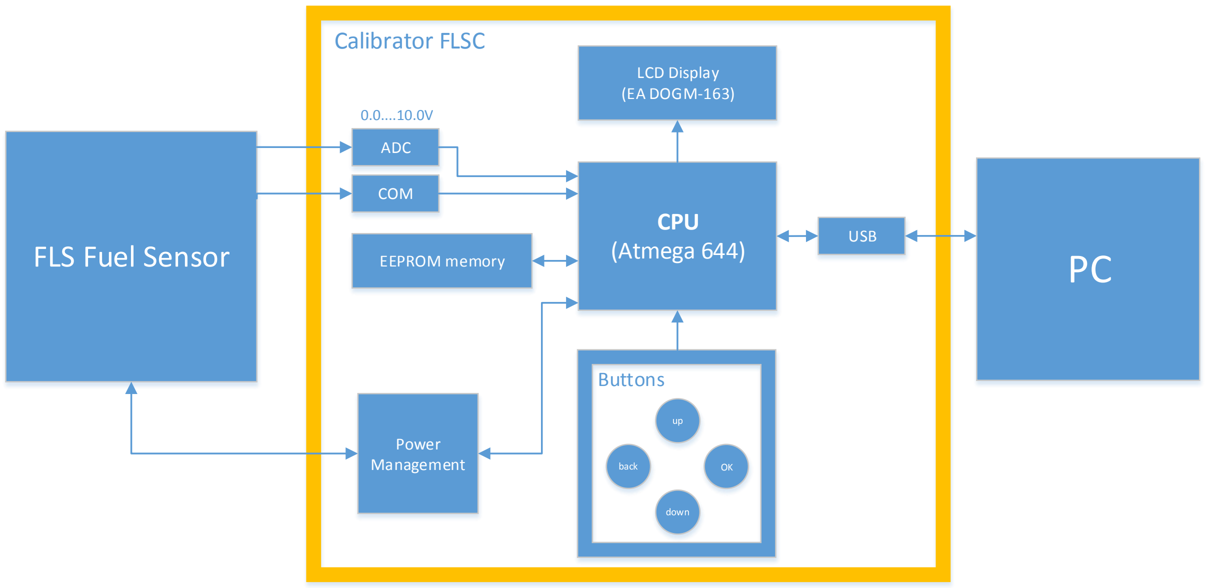


Figure 3-1 Calibrator FLSC Block Diagram

3.4.1 Calibration

Calibration is the primary function of the FLSC. Calibration functions are accessed through the appropriate menu. The user enters calibration points through the calibrator interface. It is necessary to first fixate points corresponding to a full and empty fuel tank. Then it is possible to enter intermediate calibration points. This is usually done by putting the FLS in an empty fuel tank and then adding a known amount of fuel for every calibration point. It is the responsibility of the user to make sure that a correct fuel amount in litres is entered for every calibration point.

During calibration, the FLSC constantly polls the FLS for data and displays parameters on the LCD screen. If there is no response from the FLS, then the calibrator assumes that connection has been lost and calibration will not be possible until it is restored. This is done to prevent incorrect or partial data from being passed between the two devices.

3.4.2 Data Storage

The calibrator has an internal EEPROM module that is used for data storage. The CPU communicates with the EEPROM module in order to store and receive data. The FLSC has 2 KB of EEPROM memory, enough to store a single calibration. There is enough space to store additional calibrations, but currently there is no need for such functionality. A saved calibration can be uploaded to a connected PC, or even directly into a connected FLS device. The FLSC can also download an existing calibration from a fuel sensor.

3.4.3 Diagnostic

The calibrator provides access to diagnostic data about a connected fuel sensor. In diagnostic mode the calibrator constantly polls the FLS for data and displays desired parameters on the LCD display. The FLSC shows information about measured generator frequency, output voltage and internal temperature. It also displays the version of the FLS software as well as some factory set parameters. The FLSC also directly measures output voltage from the FLS by using an ADC. It is possible to compare measured voltage with the digitally transmitted value. This enables the user to debug the FLS.

3.4.4 Power Management

The FLSC calibrator is powered by two mobile phone batteries that can be charged through the USB port. When the calibrator is connected to an FLS, the sensor receives power directly from the calibrator. This helps to ensure that the FLS and the FLSC can work together without requiring an external power source. In its normal mode of operations, the FLS is powered from the vehicle’s battery. The FLSC’s CPU measures battery voltage output with an ADC and gives an indication of battery state. If the voltage falls too low then the calibrator will notify the user and once battery voltage drops below a certain threshold, the calibrator will power down. The FLSC is also programmed to conserve power if the calibrator is inactive and will power off the LCD display’s backlight if the user has been inactive for more than a minute. The FLSC’s battery lasts for about 24 hours.

4. FLSC Software Architecture

One of the main parts of this thesis is the program written for the FLSC microcontroller. The program was written in C and was compiled with Atmel Studio 6.2. Atmel Studio is a software development program that is works on the shell of Microsoft Visual Studio 2010. It has been developed by Atmel to design software for Atmel embedded microprocessors. [3] The final program for the FLSC consisted of a number of .c and .h files that were compiled and linked.

The Atmega644 microcontroller was programmed with an STK500 device. This was done by using Atmel Studio 6.2 that has functions for controlling an STK500.

4.1 Design Decisions

Due to limitations of the Atmega644 microcontroller [2], it was decided not to use C++ or object oriented programming. This is mainly due to memory concerns as well as difficulties with finding a suitable compiler. The program was designed bottom-up so that low level hardware functions were realised and tested first. This included COM port, LCD display, ADCs and EEPROM functions. After the hardware was working and tested, design was focused on implementing higher level functions.

It was decided that the program should run on a single thread because multithreading on the Atmega 644 can be quite difficult and there is very little support offered by Atmel. Timer interrupts were used to take care of periodic processes. This enabled multiple tasks to be accomplished simultaneously and was sufficient for the FLSC. Receiving and processing data over the COM port is also interrupt driven, although actual data processing is also a part of the main cycle. This is done in order to prevent long interrupt service routines that could cause the CPU to lock up.

The FLSC uses serial communication with the FLS. This is again something that is simple to implement with the Atmega644 and the AtXmega 16A4U [2] [6].The communication protocol is in a semi-verbose format, which makes it possible to read transmitted messages and perform debugging. The FLSC can be connected to a PC by using an USB dongle. There is an FT232R [7] serial converter on the calibrator. When the calibrator is connected to a PC then it creates a virtual COM port in the computer. It was decided to use a serial port converter because unlike the newer AtXmega, the Atmega644 lacks built in USB support [2].

It was also necessary to write a driver for the EA DOGM162 LCD display. An existing driver for an AVR processor could not be found and therefore one had to be designed. The LCD display’s interface is relatively simple and designing a driver was not overly difficult. The EA-DOGM-162 has an integrated ST 7036 controller interface. It supports an 8-bit, 4-bit and SPI (**Serial Peripheral Interface**) operations. In a very early version, the calibrator used an 8-bit connection. Later it was decided to use SPI instead because it needed a smaller amounts of pins to be connected. Therefore both modes are supported by the display driver.

Using Atmega’s EEPROM was an obvious choice for storing calibration data. The processor has 2KB of EEPROM available which is more than enough space for storing calibration points. Since the user might want to review calibrations at a later time, it was necessary to use memory that wouldn’t be erased after shutting off power to the FLSC.

Each mode of the calibrator has a set of functions associated with it. Every mode has a main process that is called during every program cycle. This process contains routines for displaying data on the screen and handling user input as well as sending commands to the FLS. Every mode has its own function for handling received data. This function is used to parse messages from the FLS and take actions based on received data.

4.2 Main Program Cycle

The main program contains the start-up routine for the microcontroller and the main processing loop. Here follows a diagram describing the main program.

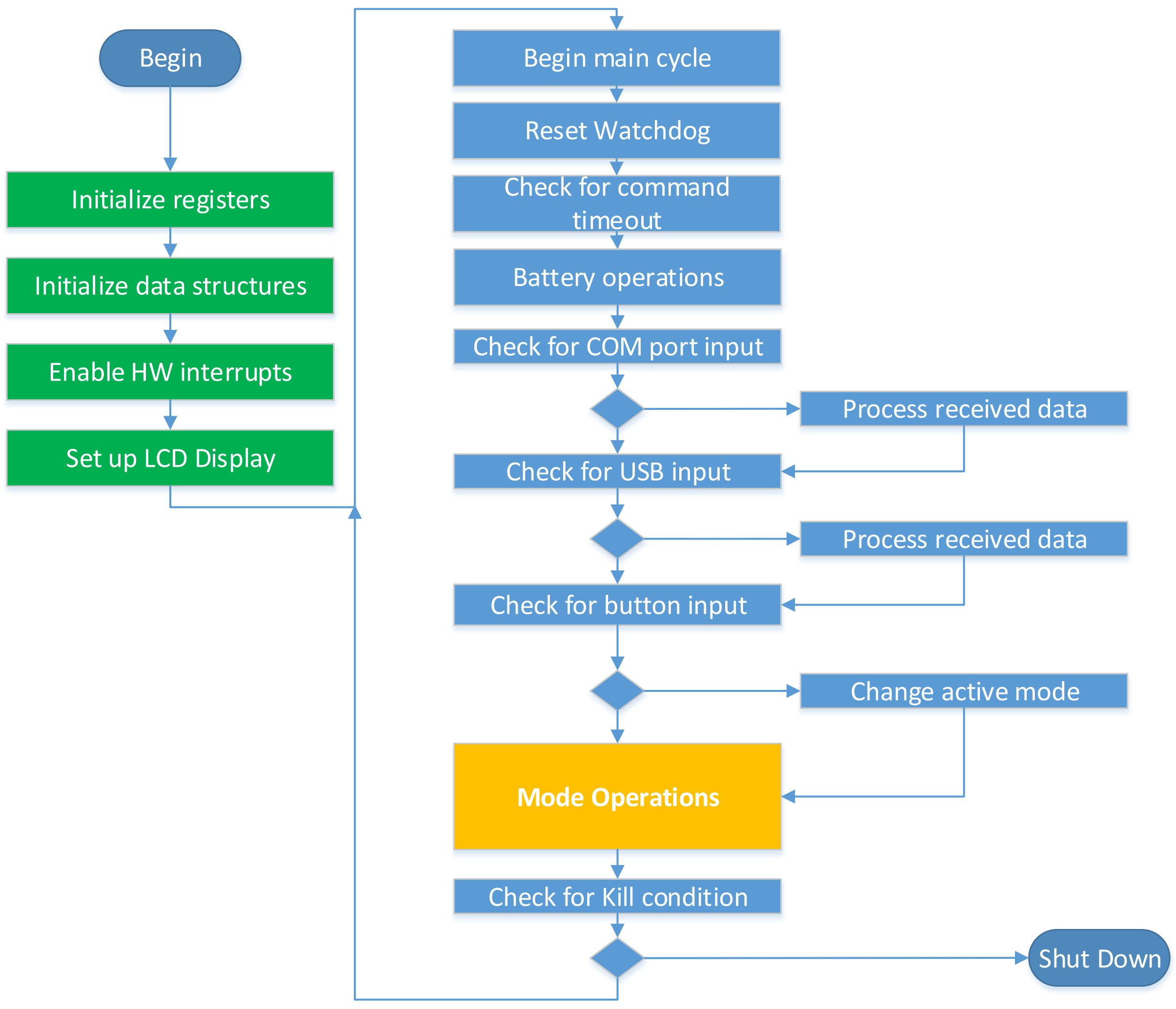


Figure 4-1 FLSC Main Program cycle

The program starts with initializing registers for the microcontroller. These registers control the mode of input/output pins. The program also initializes registers for configuring UARTs and ADC-s. The program then initializes variables in data structures and resets all flags. After this done, the program enables hardware interrupts that are needed for the internal timer and UART modules. After this the controller itself has been fully initialized and the program continues to setup the LCD display. It takes a few hundred milliseconds to turn on the display and set the mode of operations. This concludes the start-up procedure and the program moves on to its main cycle.

At the beginning of the main cycle the internal watchdog timer is reset. The main purpose of the watchdog is to detect if the calibrator program has stopped functioning or is stuck in an infinite loop. If the watchdog timer is not reset, then the calibrator controller will restart itself. The program then checks for a command timeout flag. If this flag is set then it means that the calibrator has not received a response to a command sent to a connected FLS. It then assumes that connection with the FLS has been lost and will display a message on the LCD screen. Next come battery operations, where the program checks battery state. If the battery is low then the calibrator will periodically print out a notification on the display. If the battery is critically low, then the calibrator will notify the user and shut down after 60 seconds if an external power source has not been connected.

The main program will now check for input from two communication channels. Firstly it checks if there has been a command sent from the main COM port that is usually connected to the FLS. Receiving individual characters over the COM port is not handled in the main processing cycle. Low level data communication is entirely interrupt driven. Instead the main cycle handles a case where a complete command string ending with the “r/n” characters has been received. This raises an internal flag that the main program reads. If the flag is set then the program parses received data and takes corresponding action. This process is identical for the USB port, since the calibrator has an FT232R serial converter connected to the USB socket. [7]

After handling received data, the main program cycle checks for button input. The program checks for flags that are set through timer interrupts. If the user has pressed the BACK or OK buttons, then the calibrator changes its active menu or mode. After the calibrator has performed these checks, the program moves on to active mode operations. The exact nature of these depend on the currently active menu or user mode.

Finally the calibrator checks for “kill” flag which is set if the user wants to shut down the calibrator. If this is set, then the calibrator will shut down. If the flag is not set, then the main processing cycle will begin from the start.

4.3 Hardware Interrupts

The Atmega 644 microcontroller has two internal 8-bit timer/counters and one 16-bit counter. Timer/Counter0 is a general purpose 8-bit Timer/Counter module with PWM support. It allows accurate program execution timing, event management and wave generation. [2, pp. 86-102] In this case the timer is used for scheduling events inside the calibrator's program. The timer is clocked from the main system clock, an external 7.3728 MHz crystal oscillator. The clock input for the timer has been divided by 256. Once the 8 bit counter overflows, it generates a hardware interrupt which is executed by an interrupt service routine within the program.

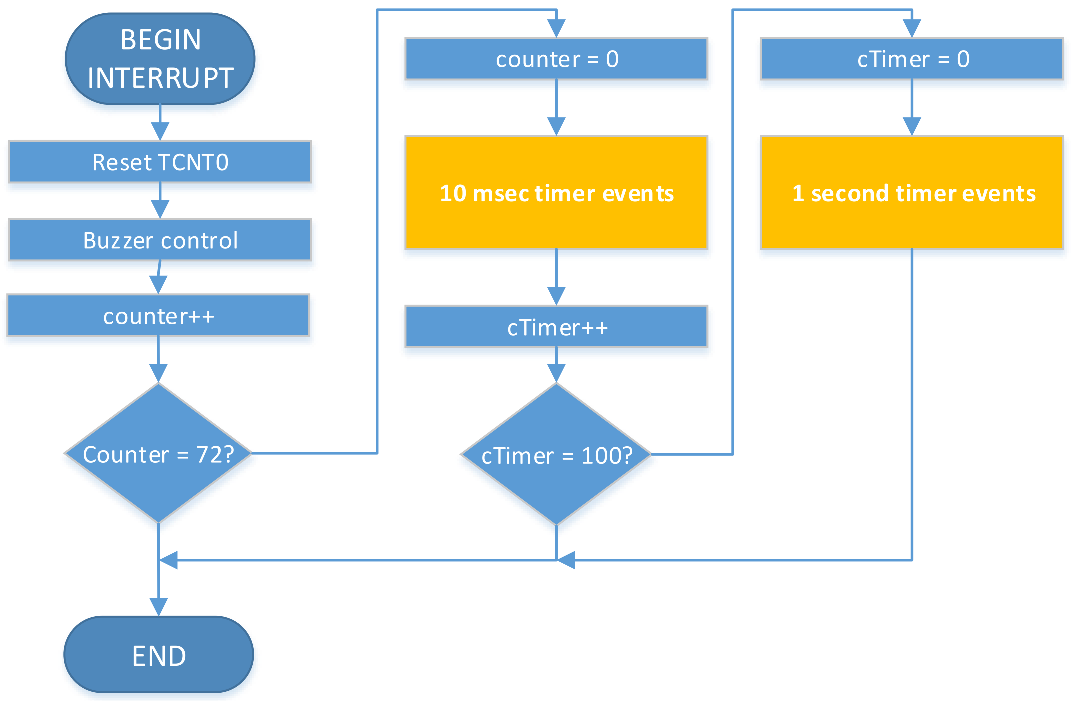


Figure 4-2 Timer Overflow Interrupt

The interrupt service routine resets the counter value, but not to zero. It sets it to a preconfigured value that has been chosen in order to better produce time intervals in the millisecond range. There is also a virtual counter variable, which is used to measure 10 millisecond time intervals. The aim is to have a set of events that are executed every 10 milliseconds. There is also another variable for measuring 1 second intervals in a similar way. This timer is also used to generate delays in the program code where needed.

4.4 Timer Events

At certain intervals the main timer triggers a set of events. There are two different routines that are periodically executed – one every 10 milliseconds and one every second. The 10 ms routine is used to manage operations that happen on a short timescale. The first thing the 10 ms routine does, is check for command timeout from the COM port. The FLS sensor must answer to a command within 800 milliseconds. If there is no answer within that timeframe, then the calibrator will assume that connection has been lost.

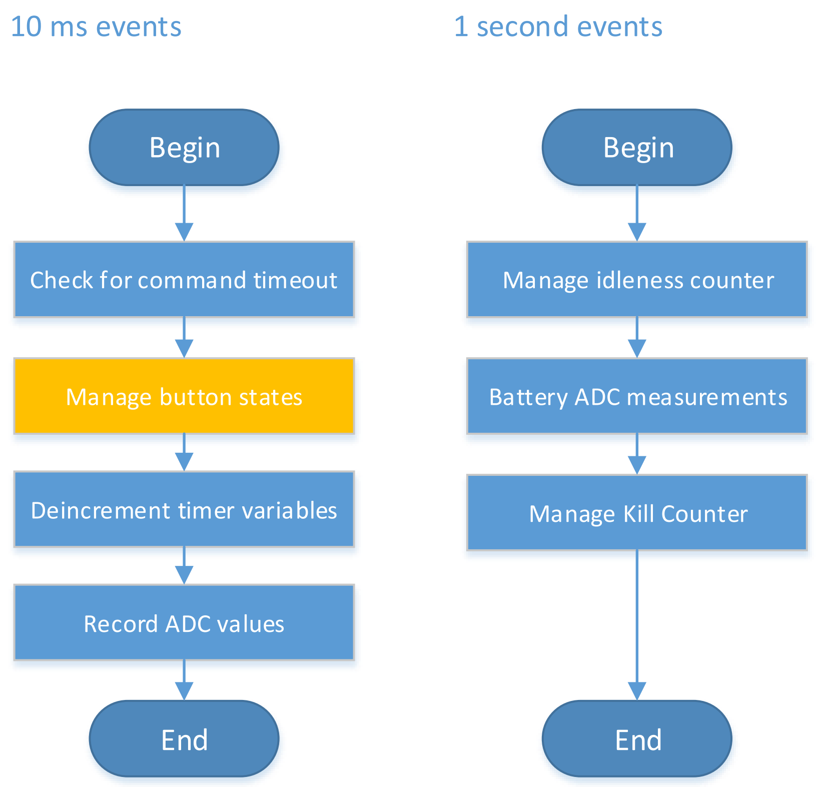


Figure 4-3 Timer triggered events

The 10 millisecond timer is also used to check for button states. The calibrator has four buttons for user input. The 10 ms timer triggers input measurements from each button. If the timer detects that the button is being held down, then it sets a flag to indicate a button press. If the button is configured to be triggered by the falling edge of the input pin, then the timer will set the flag once it sees that the button has been released. This kind of an approach can detect keypresses that last for longer than 10 ms. This helps to prevent debounce effects which would become a problem if the keypresses were interrupt driven.

The 10 millisecond timer also performs timekeeping for some of the calibrator’s functions. It decrements counter values that keep track of time intervals. Lastly the timer performs ADC measurements from the FLS-s analogue input. The calibrator has a small array that continuously keeps track of ADC measurements.

The 1 second timer takes care of less precise timekeeping functions. The calibrator has an idleness counter which is reset each time a button is pressed. If no buttons have been pressed for more than 120 seconds, then the calibrator turns off the LCD display’s backlight to conserve energy. The 1 second timer also takes care of ADC measurements of battery voltage. If the battery voltage drops below 3.6 V then the calibrator will notify the user of a low battery. If the battery voltage drops below 3.45 V then the timer will set a flag for the calibrator to shut down.

4.5 Communication protocol

The FLSC has two communication channels. It is connected to the FLS sensor via COM port and can be connected to a PC via USB dongle. The calibrator has an FTR-232 chip that creates a virtual COM port between the PC and the microcontroller.

4.5.1 Communication with FLS

The calibrator communicates with the FLS to pass commands and ask for data. The FLS does not initiate communication, it acts as a slave device for the FLSC. Commands for the FLS are in a semi-verbose format for debugging purposes. All commands begin with the prefix „MK+“. In some modes the FLSC continuously polls the fuel sensor for data. The FLS must answer to a command within 800 ms or the calibrator will assume that connection has been lost. If a command is a result of a user action, then the calibrator will also notify of the success or failure communication. It displays either an „OK“ or an „ERROR“ message on the display depending on the result.

The COM port operates at a baud rate of 9600 baud/s. Data is sent to the FLS by writing characters to the UART register of the Atmega microcontroller. The microcontroller will automatically transmit a character, if it detects that the register has been accessed. [2] All commands to the FLS are ended by the linebreak symbols.

Receiving characters via COM port is interrupt-driven. The characters are in 8-bit ASCII format. If a character is received by the microcontroller, it is written to an input buffer [2]. This triggers an interrupt that is cleared by an interrupt service routine. The interrupt service routine appends the received character into a buffer. The buffer is 256 bytes long and if it overflows then further incoming characters are simply ignored. If the program detects an linebreak character, (number 10 in the standard ASCII code [8]), then it sets a flag indicating that a full message has been received. This flag is read and cleared by the main program cycle. The main cycle then processes received data.

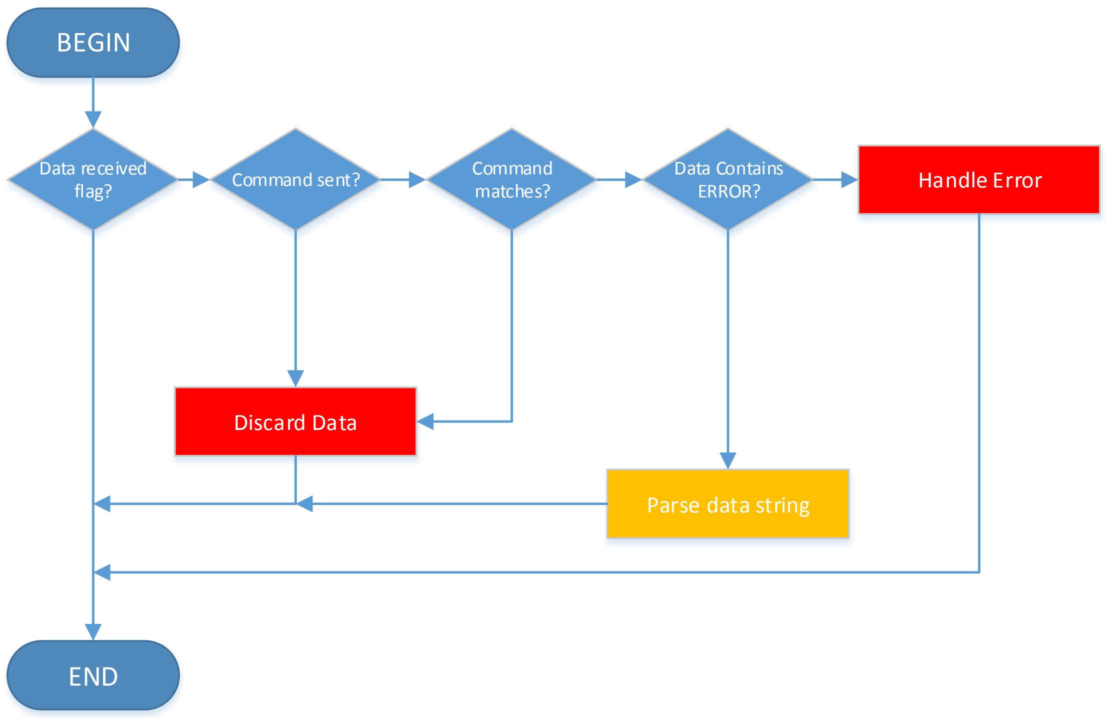


Figure 4-4 Block Diagram for COM port data input

The data packet must also echo back the original command that was sent by the FLSC. If no command has been sent out by the FLSC or the header of the message is different from the command, then a message will simply be ignored. Additionally if the received data contains an „ERROR“ message, then it will not be analysed further and the program simply sets a flag for a communications error.

Example of communication:

```
FLSC: MK+CAL3=25
FLS:  MK+CAL3=25 -> 2:25L-5447 10.5%  x:5447  z:10.5
```

This ensures that the FLSC knows if the FLS answered to its most recent command. In earlier versions of the calibrator, there were problems with mixed up messages. The calibrator would sometimes send out multiple commands in a sequence and could not understand which command was being answered by the FLS. This led to the calibrator parsing data incorrectly and displaying faulty or garbled information. To prevent such mixups, the calibrator can currently only send out one command at a time and cannot send another until the previous one has been answered or timed out. The echo mechanism provides additional security against mixups.

Once the program has made sure that received data is valid and in a correct format, then it is passed to a parser function. The exact function depends on the FLSC-s current mode. This will also determine how data is handled as different functions may use the same commands for communication. When parsing large data strings, the parser functions search for characters followed by a semicolon ';' in order to parse different values. Even if parts of the data string are corrupted, then it might still be possible to extract valid data. The following data string contains a general description of a calibration point, measured generator frequency and its percentage compared to the maximum value.

Example of message from FLS:

```
FLS:  MK+CAL3=25 -> 2:25L-5447 10.5%  x:5447  z:10.5
```

4.5.2 Communication with PC

The FLSC can be connected to a PC by using a USB dongle. It creates a virtual COM port in the PC that can be accessed with COM port monitoring software or the Windows Program for FLSC. The calibrator can receive commands from the PC. When debug mode is enabled, the calibrator will also mirror all communication with the FLS to the PC.

All commands that are sent to the FLSC from the PC also have to begin with the prefix „MK+“. Receiving data from the PC works similarly to the FLS. Low level data communication is interrupt driven and incoming characters are appended to a data buffer. It is possible to send commands to the FLSC through the virtual COM port.

It is also possible to upload a full calibration from the PC by using the FLSC Interface program. The loading cycle begins by notifying the calibrator with a command : „MK+PC_SEND_BEGIN“. The calibrator will then expect a number of calibration points to be transmitted by the PC. The FLSC will erase any previous calibration data and save new calibration points in its memory. The upload cycle is finished by transmitting the command: „MK+PC_SEND_END“.

4.6 LCD Display driver

The FLSC uses an EA-DOGM162-a 8-bit LCD display to display information to the user. The EA-DOGM162 is a two-line character display. It has 16 character slots per line. It can be configured to run with only a single character line that uses larger characters. The display has a built-in character set consisting of 248 pre-defined characters. It is also possible to define 8 custom characters. [9]

4.6.1 SPI Interface and Low Level Communication

As a part of this master's thesis, it was necessary to write a driver for the LCD display that would work on the Atmega microcontroller. The LCD display is configured to use SPI (Serial Peripheral Interface) for receiving commands. Under the SPI configuration there are 4 data pins that are connected to the microcontroller. These are SI(*Serial Input*), CLK (*clock*), RS (*Register Select*) and CSB (*Chip Select*).

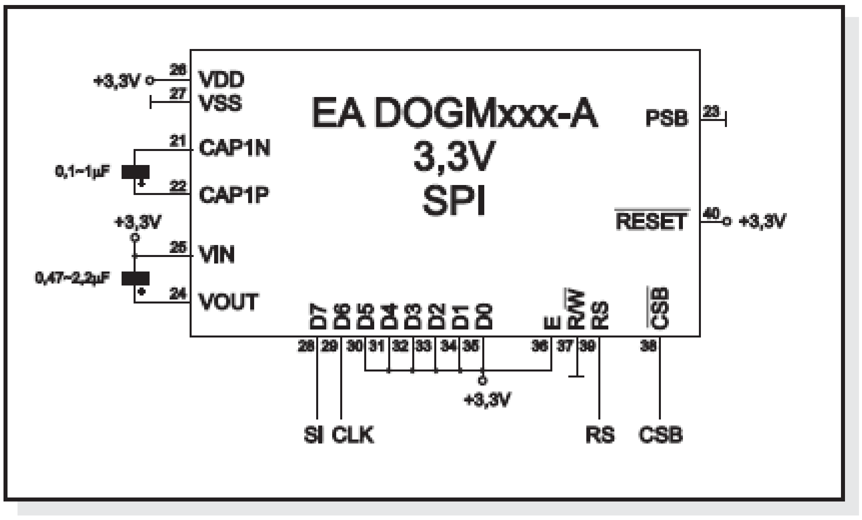


Figure 4-5 SPI COnfiguration of EA DOGM162-A LCD Display

The Serial Input Pin is used to transmit individual data bits, that are recorded on the clock's rising edge. The clock pin is used to generate time intervals for separating individual bits. The Register Select pin is used to specify the command mode, and the chip select pin is used to activate the LCD module. In the beginning of operations there needs to be a falling edge on the CSB pin. This is used to synchronize the display and begin operations. During operations, the CSB pin is driven low. Note that this pin cannot simply be connected to the ground. [9]

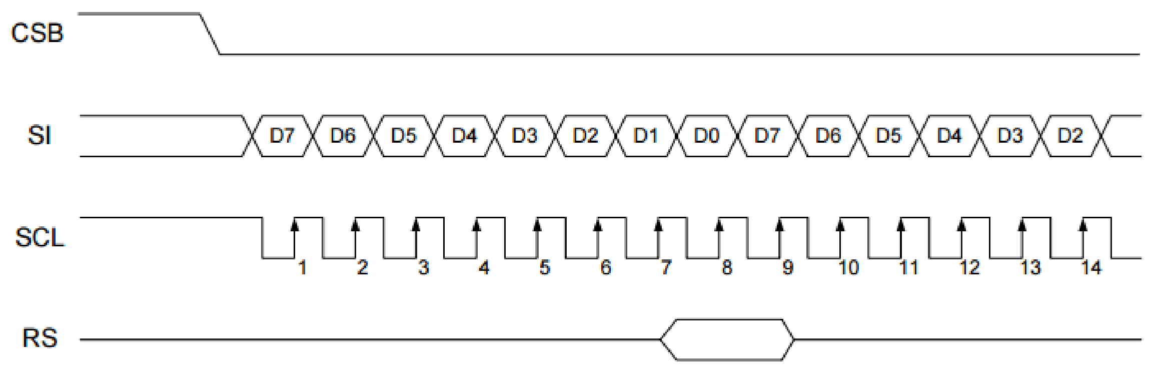


Figure 4-6 SPI operations for the ST 7036 interface

It takes 8 clock cycles to transmit one byte of data. At the end of the data byte the RS pin must be in its desired mode. The interface does not have a pin for busy checking in SPI mode. The minimum clock pulse length is 200ns. [10] Since the Atmega644 is clocked at 7.3728 MHz then there is no need to generate delays in the program code. The time period for accessing and changing register values already takes care of the necessary delays that are required for transmitting data.

Data communication with the LCD is write-only. This means that it is not possible to for example read data from the LCD display. In order to keep track of characters that are written on the LCD screen, the program saves them in an array which can be accessed by the calibrator functions.

4.6.2 Initializing the Display

The display is initialized at the start of the FLSC program. During the initialization the display's mode and parameters are set. The initialization takes about 300 ms.

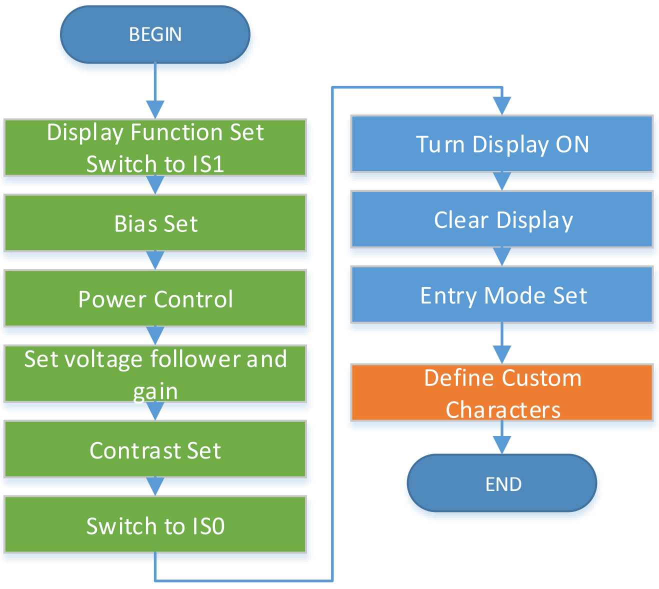


Figure 4-7 LCD initialization routine

When starting the program first initializes the display and commits the function set command. This configures the display to 8-bit configuration with 2 lines and a normal-sized font. This operation also selects Instruction Table 1 for the following operations. The display has 3 instruction tables which provide access to different functions. Instruction codes are 8 bits long. Also the RS pin is used to differentiate between instructions. [9]

IS0 : Standard Instruction set during normal operations. It is possible to access CGRAM (*Character Generator RAM*) and therefore define custom characters.
IS1 : Contains advanced setup instructions for setting the bias and contrast of the display.

IS2 : Not used in this design. Contains functions for configuring the position of double height characters. [9]

In addition there are a set of functions that are always available no matter which instruction set is currently active. [9]

Once function setting is complete, the routine sets the bias of the display to 1/5. After this the routine performs power control and the LCD is configured to run on 3.3 V. Additionally the contrast of the display is configured. This concludes the setup configuration of the LCD. The instruction set is switched back to IS0.

After the setup is complete, the program turns on the display, clears its memory and sets the entry mode. At this point the setup for the display is complete and it is ready for use. The setup function does one more thing after this – it defines 4 custom characters for displaying the battery state in a graphic manner.

4.6.3 Writing on the Display

Writing characters on the display is fairly simple. The EA-DOGM162 display has an address space with a length of 128 slots. Since the LCD display has 2 lines with 16 characters each, only 32 of these addresses are assigned to actual positions on the display. The rest can be used for general purpose data storage. It is possible to write an 8-bit value to a memory slot which corresponds with a character from the display's character set. There are 256 characters to choose from. The characters from 32 – 127 correspond roughly with standard ASCII printable characters although there are some exceptions. [9] Characters from memory positions 0 – 7 are custom characters that can be defined by using the onboard Character Generator.

In order to write a single character on the display screen, the display driver first executes an instruction to set the DDRAM address to the corresponding position. During this the RS pin must be cleared. Then it executes another instruction to write a new value to the DDRAM address. During this the RS pin is held high. The driver also writes a character into the corresponding slot of the character array that keeps track of changes on the display screen.

4.7 EEPROM Management

The ATmega644 contains 2 Kbytes of data EEPROM memory. It is organized as a separate data space. Single bytes bytes can be read and written. In order to access the EEPROM, the program needs to write the desired address to the EEPROM address registers. Data is read and written from the the EEPROM Data Register. Finally the EEPROM Control Register is used to execute the desired operation. [2]

The FLSC uses the Atmega's EEPROM for storing calibration points. Data written in the EEPROM is not lost when the processor is powered down and it is possible to use this for long term storage. The main purpose of storing calibration points in EEPROM is to make sure that the user can review performed calibrations. It is possible to download and view a calibration from a connected FLS sensor. If a user wishes to calibrate several sensors with the same data, then the user can upload a calibration to the FLS directly from the FLSC. Calibrations can also be uploaded to a connected PC and vice versa.

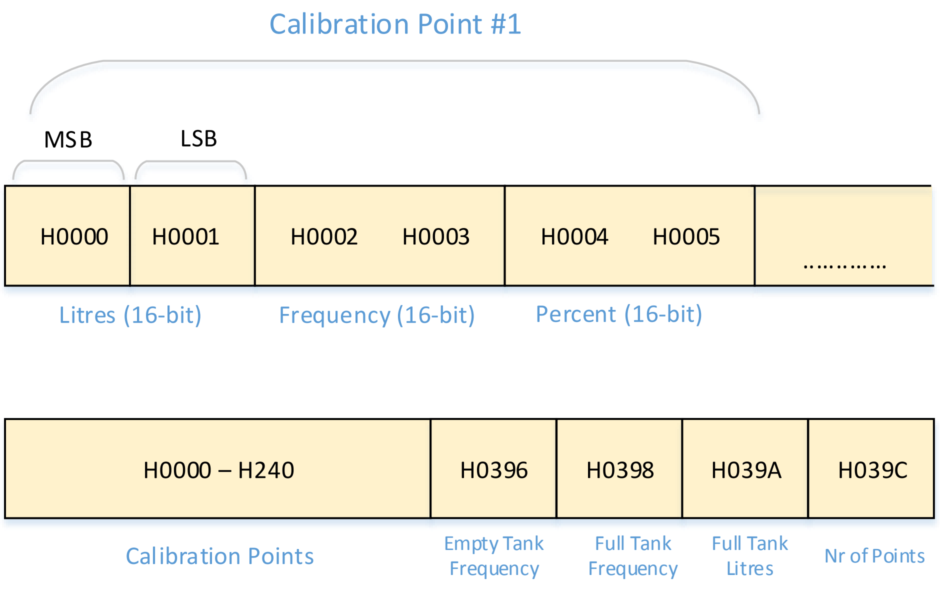


Figure 4-8 EEPROM Memory Structure

It is possible to save up to 96 calibration points in the FLS. Therefore the FLSC does not allow to save more than 96 points for a calibration. Each possible calibration point has a dedicated memory slot in the EEPROM. Every valid calibration must consist of at least two points – one point corresponding to an empty tank and one to a full tank. Values for the full and empty tank are stored in a separate location at the end of the EEPROM address space.

The total number of points is also written into EEPROM, so that the program knows which slots contain valid data.

A calibration point has four variables associated with it.

1. The number of the point (1-96)
2. The amount of fuel in litres (0 – 2000L)
3. The generator frequency (0 - 10000)
4. Percentage of a full tank (0.0 – 100.0 %)

The number of the calibration point is not written to EEPROM. Instead the EEPROM address space has been divided into 96 slots, each corresponding to a possible calibration point. If the program needs to retrieve a point then it will simply read data from the corresponding slot. Each calibration point takes 6 bytes of storage space. The space for a single calibration point is divided into three 16-bit „WORD“ spaces that consist of 2 8-bit EEPROM address slots. It therefore takes 2 EEPROM reading cycles to read and write this data.

Whenever the program stores a new empty tank frequency to the calibrator, then calibration data in EEPROM memory is reset. Data is not erased, but the „number of points“ memory slot is set to 0. As new calibration points are added to the memory, this value is incremented. It is assumed that when a new empty tank value is configured, then the user has started a new calibration and in any case it is highly unlikely that old calibration data would still be valid. Empty tank fuel amount in litres is always presumed to be 0 L so this value is not stored in EEPROM. The full tank point is also stored separately, but updating this value does not reset the rest of the calibration points.

In addition to calibration data, the language selection for the calibrator is also stored in EEPROM. This means that the user does not have to set the langugage each time he starts up the FLSC device.

Low level EEPROM operations have been defined in the header file „avr/header.h“ which is a factory made EEPROM driver for the Atmega series. [11]

5. FLSC Functions

The user can access calibration functions by selecting different modes of the FLSC. Each mode has its own set of data that is displayed to the user. Each mode has its own method of communicating with a connected FLS sensor. The modes of the FLSC are organised into a system of menus that can be navigated by using buttons. Functions are grouped together into general categories such as setup, calibration, diagnostic, data storage etc.

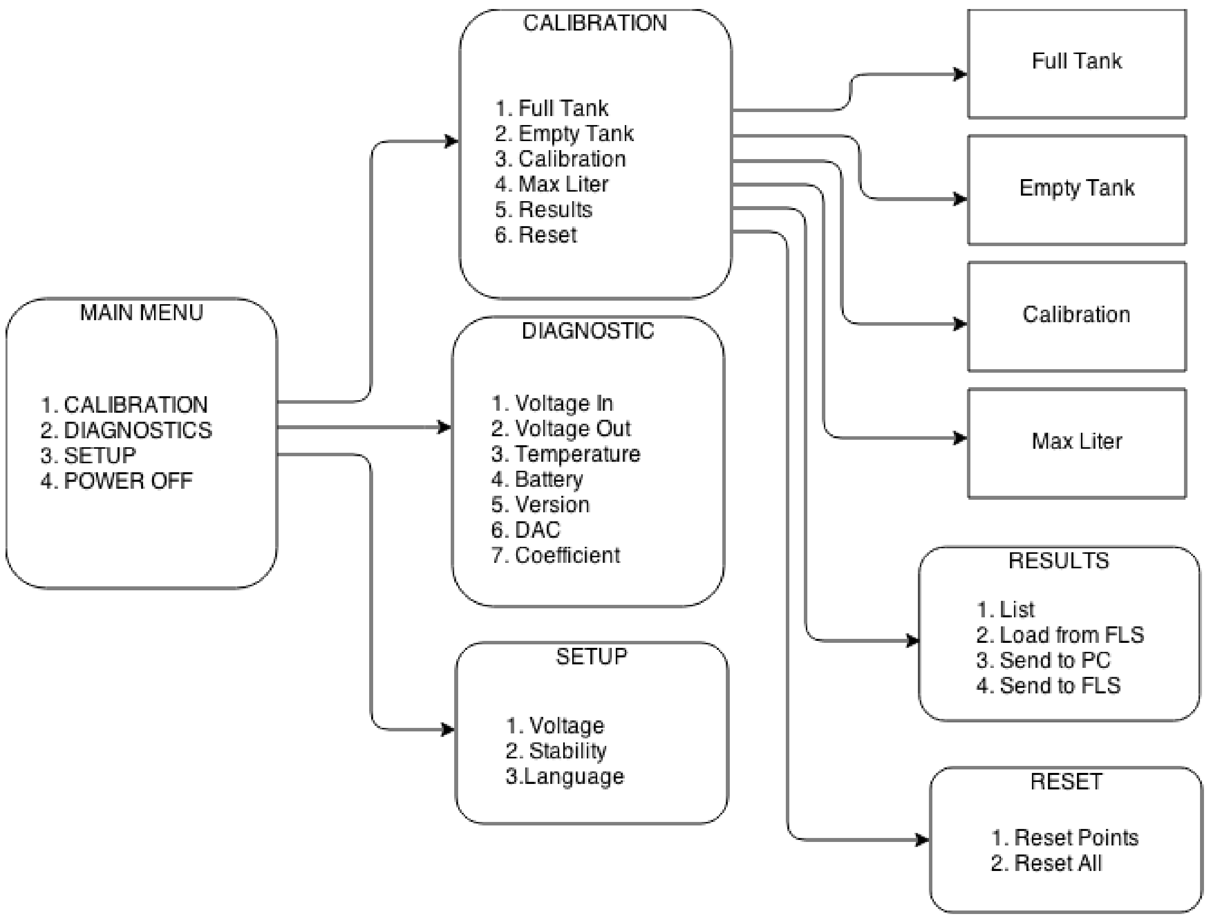


Figure 5-1 FLSC menu map

Figure 5-1 outlines the general menu structure of the FLSC. When the calibrator is powered on the user sees the Main Menu that contains four selections: calibration, diagnostics, setup and power off. The first three options take the user into the specified submenu. Choosing the fourth option will power down the device. The user can change the selection by using the up and down buttons. Pressing OK will navigate the user to the next menu and pressing BACK will return to the previous one.

The main purpose of designing this menu structure was to make navigating between modes easy and intuitive for the end user. In very early versions of the calibrator, the user had a dedicated “MODE” button that was used to cycle between different functions. When the number of modes became larger, this kind of an approach became too tedious. By organising different functions into a menu system it was possible to group functions together and make the system easier to navigate.

Most FLS calibrations begin by submerging the fuel sensor completely in fuel. Therefore the first step for the user is to set the full tank configuration. Under the current structure the user only needs to press the OK button twice to reach the beginning of the calibration. This way more commonly used functions take less effort to reach than the less commonly used setup and diagnostic functions. In an industrial environment this makes working less tedious and somewhat less time consuming.

5.1 Calibration Menu

The functions grouped under CALIBRATION deal with managing calibration points for the FLS. This is the most commonly used menu for general operations. This menu also provides access to internally stored calibration data which can be accessed and viewed to verify calibration results. There are a total of 6 possible modes to select from.

Usually calibration of a fuel tank begins by selecting “Full Tank” to fixate the maximum fuel level. Then the empty tank is set and after that the user can begin inserting calibration points. Once all calibration points are set the user can ask the FLS to estimate the size of the tank based on calibration data. Finally the user can use the FLSC to review calibration results.

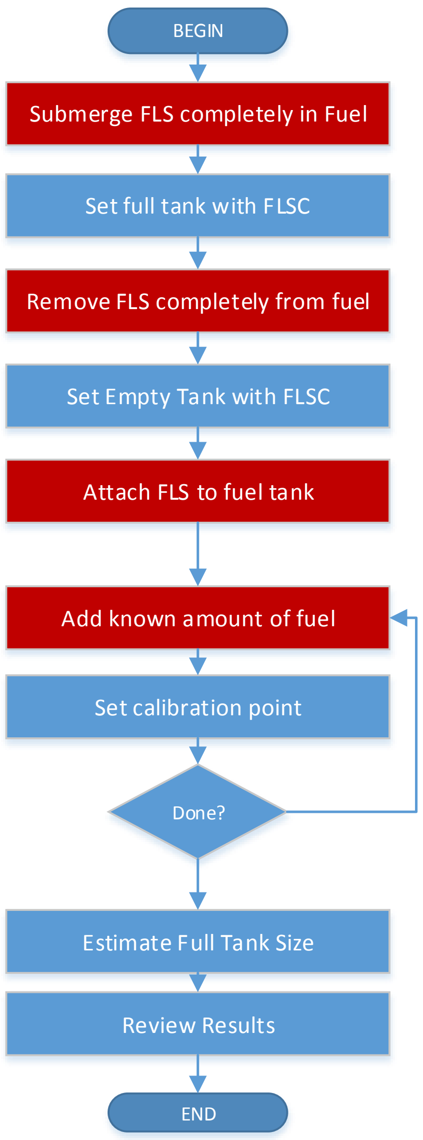


Figure 5-2 Typical Calibration Procedure

5.1.1 Full Tank Mode

In this mode the user records a situation where the fuel sensor is completely submerged in fuel. The user can also set an approximate size of the fuel tank in litres, but this is not mandatory. Information about the full tank point is displayed on the lower part of the screen. Here follows an example of the display when in Full Tank mode.

Example of displayed during full tank calibration :

```
Full Tank
--------------------
312L-7508      7484
```

On the lower left, the amount of fuel is displayed in litres. This value corresponds to the maximum amount of fuel that can be stored in a fuel tank and can be adjusted by the user with the UP and DOWN buttons. The currently measured generator frequency is displayed on the lower right (highlighted in red in the example).

In full tank mode the calibrator polls the fuel sensor for data every 3 seconds. This is done to keep data up to date as well as maintain a connection with the FLS. If there is no reply to the query, then a „NOT CONNECTED“ message will be displayed.

The FLSC polls for data with the command „MK+INPUT“ See Appendix A for full list of FLS commands.

Example of communication :

```
CAL:MK+INPUT

FLS:MK+INPUT -> x:7484,7484  z:98.3,98.3  v:9.83V
    out:307L,98.3%,1805  temp:1440,1.0000 s0:2516,0
```

As soon as the FLSC is connected to a fuel sensor, it asks for data about the full tank calibration point that is saved in the sensor's memory. The corresponding frequency is displayed on the lower left corner of the display (highlighted in green in the example). In case of an uncalibrated FLS, the default factory frequency value is 9000. If connection with an FLS is lost and then reestablished, the calibrator asks for data again. This helps to ensure that even when using multiple FLS sensors, then the calibrator will always display correct information for the device that is currently connected.

The FLSC asks for Full Tank data with the command „MK+CALIB=F“

Example of communication:

```
CAL:MK+CALIB=F

FLS:MK+CALIB=F -> F:312L-7508 100.0%
```

In order to set a new full tank configuration, the user must press the OK button. Optionally the user can set an approximate size for the fuel tank. It is important that the connected fuel sensor is completely submerged in fuel. After the user has pressed OK, then the calibrator will send a command to the FLS to fixate the full tank and will then wait for a reply.

The full tank is fixated with the command „MK+CAL4=x“, where x is the desired fuel amount corresponding to full tank.

Example :

```
CAL:MK+CAL4=500
FLS:MK+CAL4=500 -> F:500L-7483 100.0%  x:7483  z:98.3
```

If the calibrator receives a good response, then it will display an „OK“ message, and the point is also saved in EEPROM memory.

5.1.2 Empty Tank Mode

This mode enables the user to fixate a situation where the fuel sensor is completely removed from fuel. This is usually the second step in calibration. This mode is somewhat analogous to Full Tank mode, however the fuel level corresponding to Empty Tank is always set to 0L. It cannot be changed by the user.

Just like with the full tank, when the calibrator is connected to a sensor, then the calibrator will send a query about the Empty Tank point currently saved in memory. The FLS responds with the frequency corresponding to the Empty Tank. In case of an uncalibrated sensor, the default frequency for empty tank is 1000.The empty tank configuration is asked with the command „MK+CALIB=E“

In the lower right corner, the calibrator shows the current measured frequency. Just like in Full Tank mode the calibrator continuously polls the FLS for data.

The user can press OK to set empty tank. The calibrator then sets it with the command „MK+CAL3=0“.

Example:

```
CAL:MK+CAL3=0
FLS:MK+CAL3=0 -> E:0L-6019 0.0%  x:6019  z:0.0
```

If the empty tank point is successfully set, then it is also saved in the calibrator’s EEPROM.

5.1.3 Calibration Mode

The purpose of this mode is to enable the user to insert calibration points. In case of a typical calibration procedure this is the third step that is performed. The fuel sensor is inserted into the fuel tank with empty and full points already set. Then the user will connect the FLSC calibrator with the sensor and begin adding fuel to the fuel tank.

In calibration mode, the calibrator displays data on the lower part of the display. It shows the number of the calibration point, and the selected fuel level. The number of the lower right indicates how much of the length of the fuel sensor is covered in fuel. In the example it is 10.5 percent full.

Example of display during calibration:

```
Calibration
----------------
2:25L       10.5
```

The user adds a known amount of fuel to the tank (ex. 25L) and then selects the appropriate amount of fuel (25L in the example) by using the UP and DOWN buttons. The user can specify a fuel amount between 1 and 2000L. However it is not possible to insert a smaller amount than for the previous point as this would indicate a logical error.

Once the fuel amount has been set then the user should wait for the frequency indicator to stabilize. Then the user can press OK on the calibrator to fixate the calibration point. If adding a point is successful then an „OK“ message is displayed and the newly added point is also written into EEPROM memory. Adding a calibration point is done with the command „MK+CAL3=x“, where x is the selected fuel level in litres.

Example:

```
CAL:MK+CAL3=25

FLS:MK+CAL3=25 -> 2:25L-5447 10.5%  x:5447  z:10.5
```

After inserting a calibration point, the calibrator displays data about the point on the lower row of the display. The FLSC keeps track of the previously inserted calibration point. If the user moves all the way down to the previous fuel amount (25L in the example), then data about the previous point is displayed.

```
Calibration 5447
----------------
2:25L-10.5% 10.5
```

It is possible to add up to 96 calibration points in the FLS fuel sensor.

5.1.4 Tank size Estimation

By selecting “Max Litre” from the Calibration menu, the user can order the FLS to estimate the size of the tank based on inserted calibration data by linearizing inserted points. The FLS does this automatically and recalculates a new value every time a new calibration point is inserted. When the user enters this mode, then the calibrator will ask for this value with the command “MK+CAL5”. The result is displayed in the lower left corner of the LCD.

Example:

```
CAL:MK+CAL5
FLS:MK+CAL5 -> Max:306L,306L
```

In this example, the FLS has calculated the fuel tank to have a volume of 306 L. This value can be overriden by the user. It can be changed with the UP and DOWN buttons. In order to confirm a new value, the user must press OK. It is not possible to set a value that is smaller than the fuel level in the largest calibration point. The calibrator overrides the estimate value with the command MK+CAL5=x, where x is the new value in litres.

Example of communication:

```
CAL:MK+CAL5=315
FLS:MK+CAL5=315 -> Max:315L,306L
```

If this is successful then the new value will also be written into EEPROM. Note that the FLS's answer also contains the original calculated value separated by a comma.

5.1.5 Reset

Reset functions are accessed through the calibration menu. If the user navigates to this menu then there are two options: Reset Points and Reset All. Reset Points only deletes all calibration points saved into the FLS. It retains data for full and empty tank configurations. Selecting Reset All will restore the FLS to a factory configuration. Resetting points is done by the FLSC with the command „MK+CALRES3". Resetting the entire configuration is done with the command „MK+CALRES". Before executing a reset, the user will be have to confirm the choice. Changes are also reflected in the calibrator's EEPROM.

5.2 Results Menu

The Results menu can be accessed through the Calibration menu. As the name suggests, this is where the user can review calibration results. This menu provides access to data saved in EEPROM and enables the user to manage this data. There are four options available for the user under the Results Menu: List, Load from FLS, Send to PC and Send to FLS. The calibrator can store one full calibration at a time and these functions allow the user to review, upload and download full calibrations.

5.2.1 List

Selecting List enables the user to view calibration points saved in EEPROM. Every time a new point is inserted into the FLS, it is also saved in EEPROM. In List mode the user can cycle between calibration points and view them. Calibration points are displayed on the screen. The number of the point, the amount of fuel and its corresponding frequency are shown. By using the UP and DOWN keys, the user can scroll up and down. Up to 96 calibration points can be saved in the EEPROM.

Example of display during listing calibration points:

```
1:25L   -   5565
----------------
2:50L   -   5789
```

By pressing the OK button, the user can switch between displaying the frequency and percentage values.

Example of display showing percentage values.

```
1:25L   -   10.6
----------------
2:50L   -   20.9
```

Full and empty tank points do not have a number preceding them. Instead they are denoted by the letters E: and F: respectively.

Example of listing full and empty tanks in case of an un-calibrated FLS:

```
E:0L    -   1000
----------------
F:1000L -   9000
```

5.2.2 Load from FLS

This option in the Results menu enables the user to download a calibration directly from a connected fuel sensor. The calibrator deletes existing calibration data and downloads new calibration points directly from the FLS. Due to communication and data processing delays, this takes some time although usually not more than 5 or 10 seconds. During the loading process the calibrator displays the message „LOADING“ and also shows a progress bar. Once the loading procedure is complete, then the user can view downloaded points.

During loading the calibrator will ask for calibration points with the command „MK+CALIB=x“, where x is the number of the calibration point. First the calibrator asks for a full and empty tank point. Every FLS must have these points configured. An un-calibrated FLS will respond with the default values. After receiving full and empty tank points the FLSC will begin to ask for further calibration points. The calibrator will parse the responses and save calibration points to the EEPROM. If the FLS responds with „NON“ then the FLSC will finishsince it has reached the end of points.

Example of download procedure:

```
CAL:MK+CALIB=E
FLS:MK+CALIB=E -> E:0L-5804 0.0%

CAL:MK+CALIB=F
FLS:MK+CALIB=F -> F:443L-7414 100.0%

CAL:MK+CALIB=1
FLS:MK+CALIB=1 -> 1:25L-5839 2.1%

CAL:MK+CALIB=2
FLS:MK+CALIB=2 -> 2:50L-6027 13.8%

CAL:MK+CALIB=3
FLS:MK+CALIB=3 -> 3:75L-6203 24.7%

CAL:MK+CALIB=4
FLS:MK+CALIB=4 -> 4:100L-6203 24.7%

CAL:MK+CALIB=5
FLS:MK+CALIB=5 -> 5:NON
```

During a download, the main thread is locked and the download must either finish or fail to continue operations.

5.2.3 Send to PC

This option allows the user to upload calibration data into a PC. The FLSC can be connected to a PC with a USB cable. It will create a virtual COM port in the connected PC and data can be read directly or by using the FLS Interface program, which is also a part of this master’s thesis. The calibrator will upload calibration data that is saved in its internal EEPROM. It does not transmit data directly from a connected FLS. In order to make sure that the desired data is transmitted, the user can press „Load from FLS“ and then select „Send to PC“ from the Results menu. This will ensure that data from the connected sensor is uploaded.

Since data processing times and memory capacity are not a problem for the PC, then the calibration points are transmitted all at a time. The points are separated by commas and newline characters. Each calibration point contains its number, fuel level, frequency and percentage. These values are separated by „-„ characters. The sending cycle is initiated by sending the message `#SEND_BEGIN and #SEND_END.`

Example of sending cycle :

```
#SEND_BEGIN

E:0-6035-0.0%,
1:25-6251-14.9%,
2:50-6409-25.9%,
3:75-6554-35.9%,
4:100-6679-44.6%,
5:125-6796-52.7%,
6:150-6915-60.9%,
7:175-7020-68.2%,
8:200-7115-74.8%,
9:225-7205-81.0%,
10:250-7295-87.3%,
11:275-7373-92.7%,
12:300-7447-97.8%,
F:315-7478-100.0%,
#SEND_END
```

If the FLS Interface program is running in the connected PC, it will receive calibration data and display them in a graph. The FLS Interface program can then be used to view and modify calibration data.

5.2.4 Send to FLS

It is possible to upload a calibration that is saved in FLSC's EEPROM into a connected FLS sensor. This function might be required if the user wants to calibrate multiple sensors for identical fuel tanks or wants to upload a calibration that has been saved and modified in a PC. Unfortunately due to the limitations of the FLS sensor's command framework, this process takes a lot of time. It has been accomplished with some unusual methods and involves resetting the FLS and reprogramming a new calibration into it.

The upload cycle begins by ordering the FLS to reset itself with the command MK+CALRES. After this is complete, the calibrator will begin the upload by first transmitting the full tank point, followed by the empty tank and other calibration points. The FLS does not have a command for entering a calibration point directly. Therefore the FLS is ordered to first to force its output to a given value with the command MK+INPUT=xxxx, where xxxx is the desired frequency. Then a new calibration point is inserted with the command MK+CAL3=zzz, where zzz is the point’s amount of fuel in litres.

Basically the FLSC simulates a calibration cycle without actually gathering input from the fuel sensor’s capacitor. This solution is not very elegant, but it has been shown to work. The upload cycle is somewhat long and there is a progress bar displayed during the process. Unfortunately it is not possible to change this approach unless the FLS program itself was changed. Since the device is already in production, this is unlikely to happen. In any case uploading an existing calibration into the FLS is not a commonly used function of the FLSC.

Example of progress bar:

```
SENDING
----------------
<======         >
```

Example of uploading a point to FLS:

```
CAL:MK+INPUT=6683
FLS:MK+INPUT=6683 -> x:6683*6683  z:33.9,33.9  v:3.40V
    out:69L,34.0%,630  temp:1216,1.0002 s0:2452,2

CAL:MK+CAL3=75
FLS:MK+CAL3=75 -> 3:75L-6683 33.9%  x:6683  z:33.9
```

5.3 Diagnostic Menu

The Diagnostic menu gives the user access to several parameters of the FLS. From here the user can view direct measurement results and also review the FLS's software version. This menu is useful for debugging purposes. There are total of 7 data sets that can be viewed. The user can switch between data sets by pressing the UP and DOWN buttons.

The data sets are:

1. Bat: FLSC-s remaining power and battery voltage. (Ex. 82%-3.99V)
2. In: FLS output voltage (0.0 – 20.0 V)
3. Out: FLS's measured frequency and the average frequency for a given period. (ex. 1234,1235)
4. Temp: FLS's measured temperature.
5. Ver: Displays the FLS's software version
6. DAC: – Shows the configuration of the FLS's Digital analogue converter.
7. Coef: – Shows a conversion coefficient for the FLS.

In order to renew data, the FLSC sends a MK+INPUT command to the FLS every 1 second. The response to this command is parsed for data.

Example:

```
CAL:MK+INPUT
FLS:MK+INPUT -> x:7448,7448  z:97.9,97.9  v:9.53V
    out:300L,95.3%,1750  temp:1439,1.0000 s0:2552,0
```

The command „MK+VER" is used to ask for the version of the FLS's software.

Example :

```
CAL:MK+VER
FLS:MK+VER -> Ver: FLS.10.22      19.2M, Aug 14 2014, 14:47:53
```

5.4 Setup Menu

The setup menu contains functions for changing the configuration of both the FLS and the FLSC calibrator. It is possible to change the output voltage range of the FLS from here as well as a stability coefficient for the sensor. The user can also change the language of the FLSC's display.

5.4.1 Voltage Mode

This mode enables the user to change the voltage output range of the FLS. When the user enters Voltage mode then the FLSC asks the sensor for its current voltage range. The default voltage range is between 0.0 – 10.0V, with 0.0 volts corresponding to an empty tank and 10.0V to a full one.

The calibrator queries for voltage values with the commands MK+CAL1 for the lower limit and MK+CAL2 for the upper one.

Example:

```
CAL:MK+CAL1
FLS:MK+CAL1  -> Lo:0.0V

CAL:MK+CAL2
FLS:MK+CAL2 -> Hi:10.0V
```

When the user presses the OK button then the calibrator will display a dialogue on the lower part of the LCD asking if the user wishes to change the voltage range.

Example of voltage mode display:

```
Voltage
----------------
NO   Change? YES
```

If the user confirms his choice, then he will be able to change the upper and lower boundaries of output voltage. The lower boundary can be between 0.0 and 5.0V. The upper boundary can be between 5.0 and 20.0V. The calibrator asks the FLS to change voltage boundaries with the commands MK+CAL1=x and MK+CAL2=y where x and y represent the desired lower and upper voltage boundary respectively.

5.4.2 Stability Mode

The FLS sensor does not output a directly measured value. Instead it has a period during which measurements are made. The FLS then calculates an average frequency value for this period of time. This value is transformed into output voltage. This helps to smooth out disturbances in the fuel level which can happen due to outside interference. Due to the fact that the FLS is designed to be function on a moving vehicle, vibrations and manuevers can cause great fluctuations in the fuel level. Stabilization helps to reduce noise in the output signal.

The calibrator enables the user to adjust the time interval for averaging. For this the user must enter stability mode in the setup menu. The averaging period is measured in seconds and can be anything between 0 and 120 seconds. When the user enters stability mode, then the FLSC will ask for the currently saved value with the command „MK+STAB“. The default averaging period is 60 seconds.

Example:

```
CAL:MK+STAB
FLS:MK+STAB -> Stab:65
```

The user can adjust this value and choose a new interval. A shorter interval will result in more direct measurements, the output signal will react more rapidly to fuel level changes. A longer interval means that the FLS gives out a smoother more stable signal, but takes longer to react to sudden fuel level changes. The user can press OK to confirm the new stability coefficient. The coefficient is altered with the command „MK+STAB=x“, where x is the desired new value.

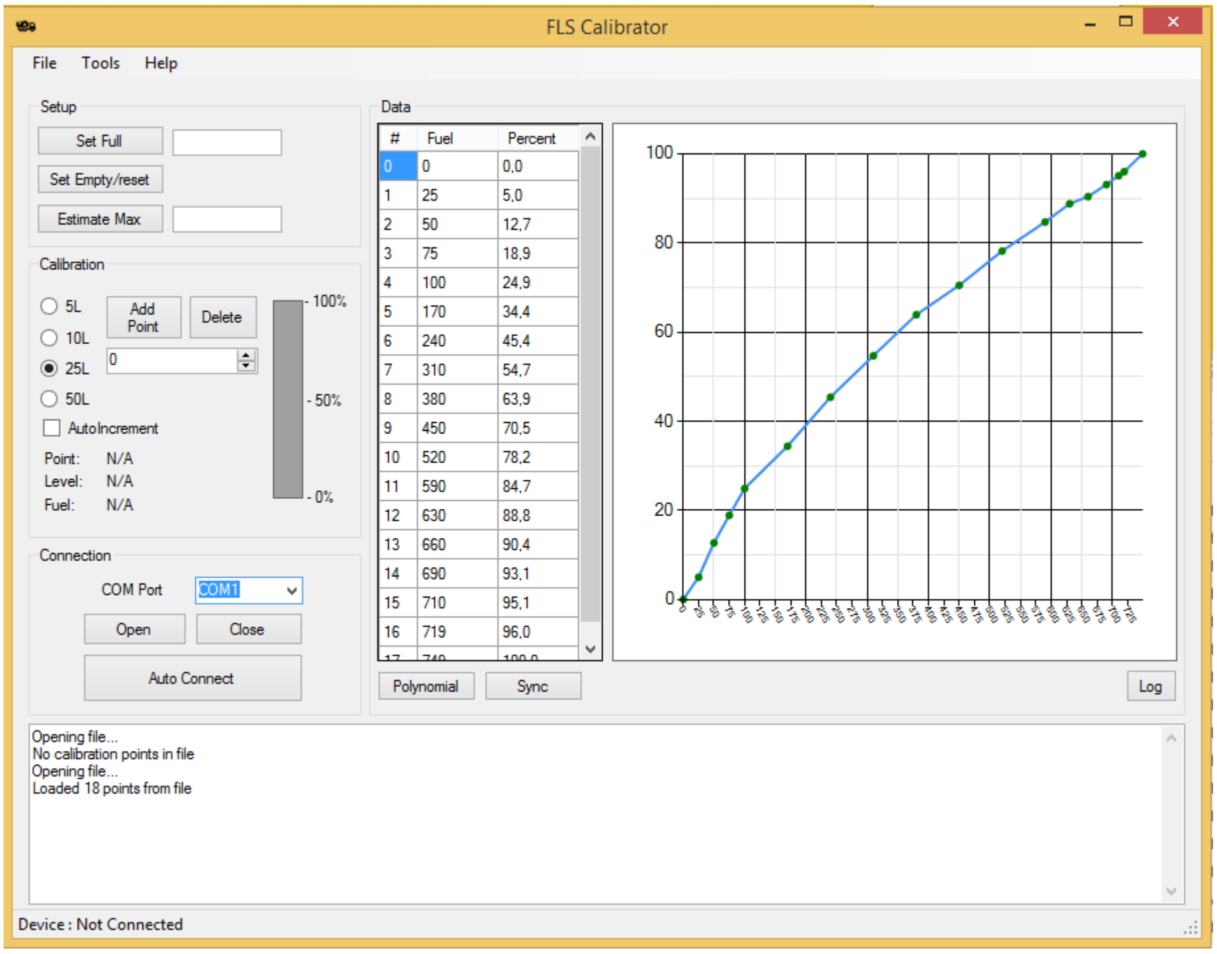
6. FLS Interface Program

The FLS Interface program is used to calibrate Tehnolabor's FLS Fuel Sensor. The program should be compatible with FLS software version starting from 10.23. The FLS Interface program can be used to calibrate the FLS sensor by using the FLSC calibrator device. The program can also be used to communicate with a connected FLSC. The program allows the user to visualize calibration data. It can also be used to analyse and modify calibrations. The program was written in C# using Visual Studio 2012 for development and testing. It has been designed to run on Windows 7 32-bit and Windows 7 64-bit although other supported Windows versions should also be compatible. The program requires the .NET 4.0 framework to run.

6.1 Graphical User Interface

The FLS interface provides information and controls through a Windows Graphical User Interface. The GUI has been built with .NET 4.0 Windows components by using Microsoft Visual Studio Express 2012. [12] The GUI is separated into 2 main parts: the large data panel on the right and the control panel on the left. The data panel consists of a datagridview for listing calibration points and a graph for plotting a calibration. The control panel enables the user to setup a calibration, insert new calibration points and also handle the connection to an FLS or FLSC device. The drop-down menus in the upper left part of the interface provide access to more advanced functions.

The GUI was designed mainly to provide a graphical representation of calibration data. It is easier for a human user to process data that is visually presented. This makes it easier to detect obvious mistakes in calibration data. A calibration point that differs from the general trendline can be difficult to spot in case it is represented in text. However in a line graph such a point can be easily identified. Also the GUI was designed as an alternative to the FLSC calibrator. One of the goals in designing the GUI, was to provide at least the same functionality as the FLSC. The program can also be used together with the FLSC. It is possible to upload calibrations from the FLSC and make modifications with the program. It is not necessary to use the FLS Interface program for making the FLS sensor work, but it does provide extra functions.



( ) 5L ( ) 10L (●) 25L ( ) 50L

☐ AutoIncrement

Point:	N/A
Level:	N/A
Fuel:	N/A

COM Port: COM1

#	Fuel	Percent
0	0	0,0
1	25	5,0
2	50	12,7
3	75	18,9
4	100	24,9
5	170	34,4
6	240	45,4
7	310	54,7
8	380	63,9
9	450	70,5
10	520	78,2
11	590	84,7
12	630	88,8
13	660	90,4
14	690	93,1
15	710	95,1
16	719	96,0
17	740	100,0

```
Opening file...
No calibration points in file
Opening file...
Loaded 18 points from file
```

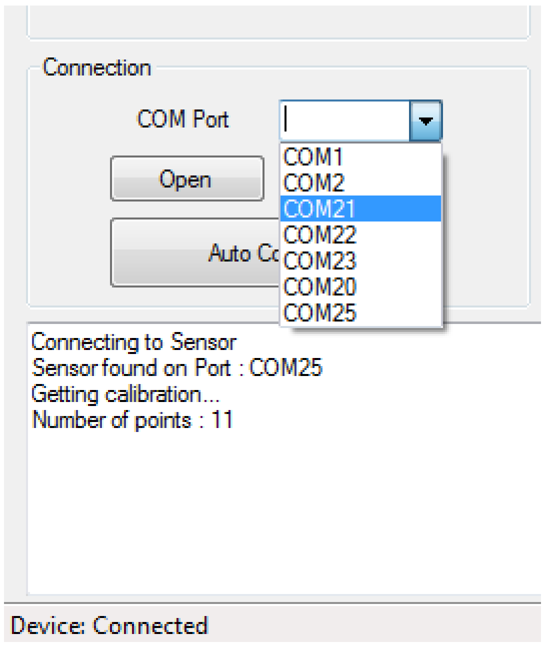
Device : Not Connected

Figure 6-1 FLS Interface Graphical User Interface

6.1.1 Connection Panel

Before using the device, it is necessary to establish a connection to the FLS device. The FLS should be connected to a USB port by using a special intermediary connector. The FS232 USB serial converter chip is used to create a virtual COM port in the PC. [7] In order to connect to the FLS sensor, the user can press the “Auto Connect” button in the Connection Panel on the lower left corner of the program. The program will then check all available COM ports and attempt to connect to the FLS Sensor or an FLSC calibrator.

It is also possible to connect manually. For this it is necessary to choose a COM port from the drop-down menu also located in the Connection Panel. The user can then try to connect the device on the specified COM port, by pressing the “Open” Button. The user can also close a connection by pressing the Close button.



```
Connecting to Sensor
Sensor found on Port : COM25
Getting calibration...
Number of points : 11
```

Device: Connected

Figure 6-2 FLS Interface connection panel

Once connection has been established, the program will load and display calibration data from the FLS fuel sensor automatically.

6.1.2 Setup Panel

The "Setup" panel in the FLS Interface is used to set full and empty points for the FLS. Usually the full tank is calibrated first. The fuel sensor is completely submerged in fuel and the situation is recorded. Just like with the FLSC, it is possible to specify the full size of the fuel tank in litres, but this is not mandatory. The user can enter the full tank amount to the textbox after the "Set Full" Button. If this field is empty, then the sensor will initially use the default fuel tank size of 1000 litres. This value can be adjusted later in the calibration process.

Calibrating the Empty Tank value is usually done next. This will also reset any calibration points except the full tank point. For this the fuel sensor should be completely removed from fuel. The user can then press the "Set Empty/reset" button on the Setup panel. The fuel amount for this operation is always presumed to be 0 litres. Clicking the "Estimate Max" button will estimate tank size based on calibration data..

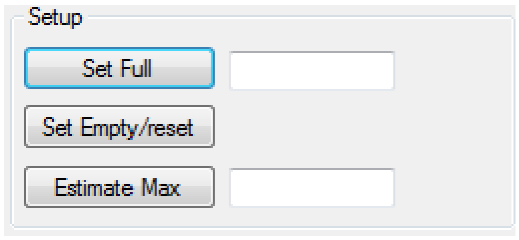


Figure 6-3 FLS Interface setup panel

6.1.3 Calibration Panel

The calibration panel enables the user to set calibration points. The Calibration panel holds the following elements :

1. Buttons for setting the fuel step - These control the step of the fuel that is added to the fuel tank between inserting points.The values are 5L, 10L, 25L and 50L. For example if the step is set to 25L, then 25 litres of fuel should be added to the fuel tank between each point.

2. Auto Increment checkbox - If this box is checked, then the program will automatically increment the amount of fuel after each point is set. Usually during calibrations workers add the same amount of fuel to the tank for each point. Using autoincrement makes calibrations more convenient.

3. Fuel amount field : This shows the amount of fuel for the current calibration point. This value can be changed by pressing the up and down arrows, which will increase or decrease the amount by the increment value specified. It is also possible to enter the number manually for fine tuning purposes.

4. Buttons “Add Point”, “Delete” - Pressing the “Add Point” button will add the currently configured point to the calibration data. Pressing the “Delete” button will remove the last calibration point from the FLS’s memory.

5. Fuel Level Gauge - This displays the current fuel level of the sensor. Hovering the mouse over the gauge will display the exact percentage.

6. Data field - This displays the current point data - point number, the fuel level and the

calculated fuel amount from the sensor. The Fuel amount might not correspond to the actual real-time value. It shows the fuel level with a stabilization algorithm taken into account, which is designed to keep the value as steady as possible. If the fuel level changes rapidly then it could take some time for the fuel level indicator to stabilize.

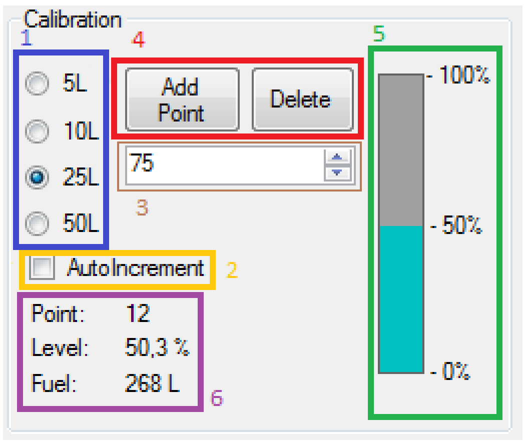


Figure 6-4 FLS Interface calibration panel

6.1.4 Data Panel

The data panel contains two items for displaying calibration points: the data table and the data graph.

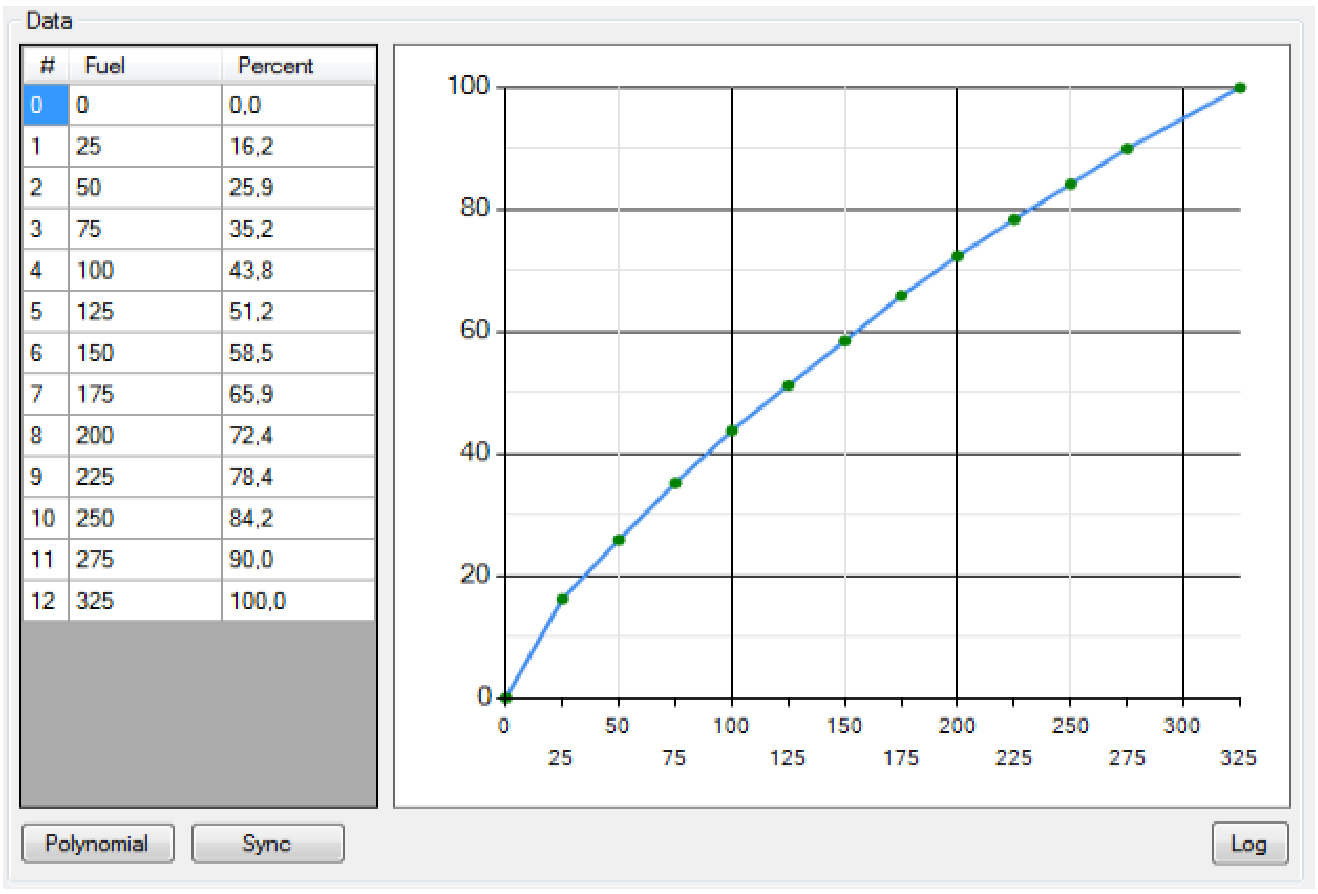
Data Table

The data table displays information about calibration points saved in the fuel sensor. Every point has a number, amount of fuel and a percentage value associated with it. It is possible to change the Fuel and Percent values manually. When these are adjusted, then the program will attempt to synchronize data between the program and the Fuel Sensor.

Pressing the sync button will delete all calibration data in the fuel sensor and override it with the values displayed in the table. This is useful if for instance the user wants to upload a previously saved calibration file into the sensor. The user can open a .csv file and then press Sync to directly upload the points into the sensor. Reprogramming the sensor can take some time, especially if there are many calibration points. There is usually no need to press the Sync button when only individual point values are changed. The program modifies these automatically.

Data Graph

The data graph displays the same information as in the data table. It helps to provide a visual representation of the calibration data. The shape of the graph line depends on the shape and size of the calibrated fuel tank. In the example, the data represents a cylindrical fuel tank. Green dots on the graph represent points that the program believes to be correct. Red dots represent points that the program believes to be erroneous. This might not always be correct assessment however, especially in the case of complex fuel tanks. The user can display a polynomial line on the graph by pressing the Polynomial button. This line represents an approximation of the fuel tank calibration curve and can help identify points which deviate from the main curve.



#	Fuel	Percent
0	0	0,0
1	25	16,2
2	50	25,9
3	75	35,2
4	100	43,8
5	125	51,2
6	150	58,5
7	175	65,9
8	200	72,4
9	225	78,4
10	250	84,2
11	275	90,0
12	325	100,0

Figure 6-5 FLS Interface data panel

6.2 Software Architecture

The FLS Interface program is written in C# and is object oriented. It uses several .NET classes for building the graphical user interface. The *MainForm* class takes care of the primary user form. [13]There are several additional classes that have been created for analyzing and presenting data. Also the program has two *backgroundworkers* that take care of periodic processes without locking up the main thread. [14] Additionally there are classes for subforms.

6.2.1 MainForm class

The MainForm class takes care of initializing and presenting the main user interface. The class is split into two partial classes. One partial class takes care of the visual components, while the other one contains more complex back-end functions. The MainForm class takes care of all button presses, text fields and indicators. It also manages the *dataGridView* and *Graph* objects that are used to display calibration points.

BackGroundWorkers

The MainForm class contains 2 backgroundworkers. These are processes that can be run without locking up the main thread. The first backgroundworker takes care of continously acquiring data from the FLS. It polls the FLS for data every 2 seconds and signals the main form to display new data. The data contains information about the measured fuel level, measured generator frequency and also diagnostic values. This worker also takes care of maintaining a connection with the FLS. If there is a time out then it signals the main form that connection has been lost. Since there are delays in data communication, this is performed in the background in order to prevent the form from freezing every 2 seconds.

The second backgroundworker is used to handle data syncing with the FLS. During syncing the FLS is reset and a new calibration is loaded into the device. Since this operation takes a noticeable amount of time, then it is run in the background to prevent the main window from freezing. During syncing, some of the buttons disabled as a simple measure to prevent data collisions.

6.2.2 Program Classes

The FLS Interface uses several additional classes to analyse information and parse input data. These classes are declared outside of the MainForm Class.

Calib Class

The Calib class provides a wrapper for the List that contains calibration points. It contains a .NET List object and methods to add, remove and read points from the list. These methods also contain data validity checks and make it easier to manage the list. The List is also bound to the *datagridView* and *Chart* objects. [15] [16]

CalPoint Class

The CalPoint class is a simple class for storing calibration points as objects. The class contains the point's number, frequency, percentage and fuel level values. It also provides getters and setters to read these values. Storing calibration points as objects makes it easy to use them as part of a List object and therefore makes them compatible with the graph and datagridview structures.

CommandParser class

This class is used to parse data from the FLSs responses. It has methods to extract integers and doubles from the response strings. It also extracts calibration points as well as parses diagnostic data. Originally the response strings were designed to be parsed in ordinary C. It proved to be surprisingly difficult to parse these in C# by using managed code. In the end the CommandParser class proved to be successful although it became somewhat larger than initially expected. The FLSC was modified to pass on calibration points as a comma separated list, so it could be parsed easily.

6.2.3 Polynomial class

The FLS Interface can interpolate a polynomial from calibration points and display this polynomial on the data display graph. The main reason for doing this is to find a function that approximately follows the shape of the calibration curve. If there are points that deviate greatly from this line then the program points them out to the user as potentially erroneous. This enables the user to quickly discover mistakes. Points which are believed by the program to be erroneous are highlighted in red on the graph.

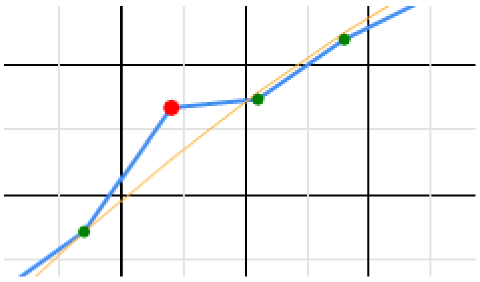


Figure 6-6 FLS Interface displays erroneous point

The approximation polynomial is a second degree polynomial in the form $ax^2 + bx + c$. In order to find the coefficients a,b and c the program solves the following linear equation:

$$\begin{cases} \sum_{i=1}^{n} y_i = a\sum_{i=1}^{n} 1 + b\sum_{i=1}^{n} x_i + c\sum_{i=1}^{n} x_i^2 \\ \sum_{i=1}^{n} x_i y_i = a\sum_{i=1}^{n} x_i + b\sum_{i=1}^{n} x_i^2 + c\sum_{i=1}^{n} x_i^3 \\ \sum_{i=1}^{n} x_i^2 y_i = a\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i^3 + c\sum_{i=1}^{n} x_i^4 \end{cases}$$ [17]

The Polynomial class contains methods for constructing and solving this system of linear equations. It does this by using Cramer's formula. The algorithm finds minors of the equation system and calculating their determinants. The determinants are calculated through Laplace's equations. [18]

6.2.4 SerialPortHandler class

The SerialPortHandler class has been implemented as a separate DLL library. The class provides a wrapper for the *SerialPort* class of the .NET framework. It handles data sending and receiving. It also provides exception handling for the SerialPort class. The handler was created to provide easy access to a serial port. It provides methods for opening and closing a serial port.

The handler expects to receive data strings that are terminated with a linebreak character. It discards data that is not properly terminated. The handler provides an event that is fired every time a full data string is received. This event can be subscribed to by an outside process. It is also possible to send data to an open serial port. The *SerialPortHandler* has a rather useful method *GetAnswerTo*. This method takes a string as input and returns the response from the FLS. If there is no response then a nullpointer is returned instead.

7. Conclusion

In the beginning of this master's thesis, I set a goal of creating working software for calibrating the FLS fuel sensor. Currently the FLSC provides an easy method for calibrating sensors. It fulfills all requirements that were set for the original device. The software uses the Atmega644 processor to its full capacity. There are no major detected problems in running hardware. Data communication with the FLS works well and the initial problem with data collisions has been solved.

The FLSC was designed to become a handheld tool for calibrating the FLS fuel sensor. As such it had access to very limited resources, a small number of inputs, a small LCD screen and limited computing power. The physical dimensions of the LCD did not allow for much data to be displayed. Therefore it was necessary to prioritize and select which data would be visible to the user. In the end it was chosen to only display data that was directly relevant to the calibration function being performed. Other functions were hidden deeper in the menu system. The menu system itself proved to be successful in allowing the user to navigate the calibrator. The menu proved to be intuitive and easy to use.

The FLS Interface program on the other hand had a very large number of resources at its disposal. The program could present data graphically, in a way that was impossible for the FLSC. The program also enabled the user to save calibrations in the PC and retrieve them for later operations. The program had all the functionality of the FLSC, but it also gave access to extra features. Therefore in addition to complementing the FLSC, it was also designed to be a viable replacement.

As a result of this master's thesis, there is now an existing software framework for Tehnolabor's FLS fuel sensor. This software now forms a whole. All the included devices have been tested together and currently there are no major problems with any of them. On the side of software, the devices are ready for production.

I would like to thank my supervisor Peeter Ellervee for providing excellent assistance during this master's thesis. Additionally I would like to thank my co-supervisor Toomas Vinter for helping me with Atmel processors and the master's thesis in general.

References

[1] Tehnolabor OÜ, "Tehnolabor web page," [Online]. Available: http://www.tehnolabor.ee.

[2] Atmel corporation, "Atmega644 Datasheet 2593O–AVR–02/12," 2012. [Online]. Available: http://www.atmel.com/Images/doc2593.pdf.

[3] Atmel, "Atmel Studio 6," [Online]. Available: http://www.atmel.com/microsite/atmel_studio6/.

[4] Microsoft, "MSDN - Visual Studio 2012," [Online]. Available: https://msdn.microsoft.com/en-us/library/vstudio/dd831853%28v=vs.110%29.aspx.

[5] Vepamon OÜ, "FUEL LEVEL SENSOR LLS 20160 datasheet," 2013. [Online]. Available: http://omnicomm-online.com/downloads/Datasheets/Datasheet%20for%20LLS%20model%2020160%20v2.13.pdf.

[6] Atmel, "AtXmega AU microcontroller datasheet," [Online]. Available: http://www.atmel.com/Images/Atmel-8331-8-and-16-bit-AVR-Microcontroller-XMEGA-AU_Manual.pdf.

[7] Future Technology Devices, "FT232R Datasheet," [Online]. Available: http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf.

[8] "ASCII table," [Online]. Available: http://www.asciitable.com/.

[9] ELECTRONIC ASSEMBLY GmbH, "DOG SERIES 3.3V Datasheet," 2012. [Online]. Available: http://www.lcd-module.com/eng/pdf/doma/dog-me.pdf.

[10] Sitronix, "ST7036 Dot Matrix LCD controller/driver datasheet," [Online]. Available: http://www6.in.tum.de/pub/Main/TeachingWs2013MSE/st7036.pdf.

[11] Atmel, "AVR Libc Reference Manual for eeprom.h," [Online]. Available: http://www.atmel.com/webdoc/AVRLibcReferenceManual/group__avr__eeprom.html.

[12] Microsoft, "Microsoft ASP.NET," [Online]. Available: http://www.asp.net/.

[13] Microsoft, ".NET Form class overview," [Online]. Available: https://msdn.microsoft.com/en-us/library/system.windows.forms.form%28v=vs.110%29.aspx.

[14] Microsoft, ".NET BackGroundWorker class description," [Online]. Available: https://msdn.microsoft.com/en-us/library/system.componentmodel.backgroundworker%28v=vs.110%29.aspx.

[15] Microsoft, ".NET DataGridview class description," [Online]. Available: https://msdn.microsoft.com/en-us/library/system.windows.forms.datagridview%28v=vs.110%29.aspx.

[16] Microsoft, ".NET chart class description," [Online]. Available: https://msdn.microsoft.com/en-us/library/system.web.ui.datavisualization.charting.chart%28v=vs.110%29.aspx.

[17] eFunda, Inc., "efunda," [Online]. Available: http://www.efunda.com/math/leastsquares/lstsqr2dcurve.cfm.

[18] D. Poole, Linear Algebra: A Modern Introduction, 2002.