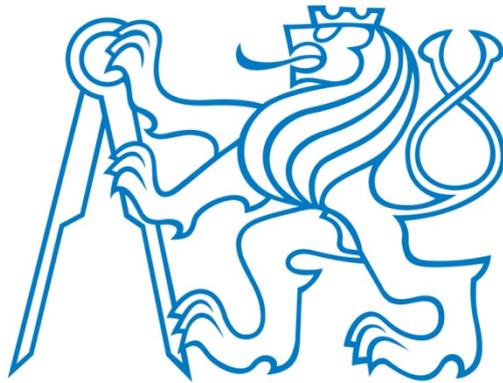


CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF ELECTRICAL ENGINEERING
DEPARTMENT OF CONTROL ENGINEERING



MASTER'S THESIS

Automatic Traffic Counter

Prague 2009

Bc. Miroslav Macháček, BEng

Declaration

I, Miroslav Macháček declare that this thesis is my own work and that, to the best of my knowledge, it contains no material previously published, or substantially overlapping with material submitted for the award of any other degree at any institution, except where due acknowledgment is made.

January 17th 2009, Prague

Signed: 

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v příloženém seznamu.

V Praze dne 17.1.2009


podpis

Acknowledgements and dedications

I would like to thank Ing. Michal Kutil, the supervisor of this thesis, for his guidance and comments, and for always being kind and helpful. I must also express my thanks to Taranis Invest, s.r.o. company for providing facilities for in-the-field tests. The biggest words of thanks go to my parents for easing up the years of my studies, and for dedicating so much effort to raise me and teach me, and generally for helping me out even during my adulthood. Without them, I would not have been able to write any thesis at all.

This thesis is dedicated to my family and all my friends (especially those from Čabuzi village).

Assignment

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Miroslav Macháček**

Studijní program: Elektrotechnika a informatika (magisterský), strukturovaný
Obor: Kybernetika a měření, blok KM1 - Řídicí technika

Název tématu: **Automatický sčítač dopravy**

Pokyny pro vypracování:

1. Seznamte se s rodinou mikrokontrolérů Microchip PIC18. Nastudujte možnosti programování a možnosti periférií.
2. Navrhněte a realizujte obvod pro snímání indukčních smyček pro detekci vozidel v pohybu. Při návrhu berte v úvahu nutnost rychlé reakce detektoru vzhledem k možným vyšším rychlostem vozidla. Dbejte zároveň na ochranu zařízení proti přepětí a statické elektřině.
3. Realizujte systém pro sběr informací o rychlostech a délkách vozidel včetně navázání na databázový systém a webového rozhraní pro zobrazování statistik o dopravě.
4. Vše odzkoušejte a vypracujte podrobnou dokumentaci HW a SW včetně návodu k obsluze.

Seznam odborné literatury:

Dodá vedoucí práce

Vedoucí: Ing. Michal Kutil

Platnost zadání: do konce zimního semestru 2009/2010



prof. Ing. Michael Šebek, DrSc.
vedoucí katedry



doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 3. 9. 2008

Abstract

The aim of this thesis was to create a system for traffic data acquisition based on inductive loops built within the roadway. The first part of the thesis involves development of an inductive loop detector for sensing whether a car is above the inductive loop. The second part comprises the realization of a device, which employs inductive loop detectors, for measuring speed and length of vehicles. The device logs the vehicle data, and transfers it to a server via a GPRS connection. The last two parts deal with implementation of a server application for receiving and saving remote data from stations to the database, together with a web interface for showing statistical information and charts about stations and their traffic.

Abstrakt

Diplomová práce popisuje návrh systému pro sběr dopravních dat pomocí indukčních smyček ve vozovce. První část práce popisuje vývoj detektoru indukční smyčky pro zjištění, zda se vozidlo nachází nad smyčkou. Druhá část obsahuje návrh zařízení pro měření rychlosti a délky vozidla pomocí detektorů indukčních smyček včetně průběžného nahrání posbíraných dat na server pomocí GPRS spojení. Poslední dvě části práce se zabývají vývojem serverové aplikace pro příjem a ukládání dat z jednotlivých stanic do databáze a implementací webového rozhraní pro zobrazování dopravních statistik a průběhů na jednotlivých stanicích.

Table of Contents

Declaration	I
Acknowledgements and dedications	II
Assignment	III
Abstract.....	IV
Abstrakt	IV
Table of Contents	V
List of Figures.....	VIII
List of Tables.....	X
List of Abbreviations.....	XI
1 Introduction.....	1
2 Motivation and Methodology	3
2.1 Reason for this Thesis	3
2.2 Review of Vehicle Detection Technologies	4
2.2.1 Non-Intrusive (Over-Roadway).....	4
2.2.2 Intrusive (In-Roadway)	4
2.3 Inductive Loops in General	5
2.4 Vehicle Detection Principle Used.....	6
2.4.1 Changes in Inductance.....	6
2.4.2 Resonant Frequency	6
2.4.3 Inductance Values and Calculations.....	7
2.5 Traffic Measurement Principle Used.....	8
2.5.1 Signals from Detectors	8
2.5.2 Measurement of Time Intervals.....	9
3 Microchip PIC and Equipment	10
3.1 PIC18 Family.....	10
3.1.1 Description	10
3.1.2 Configuration Bits	11
3.1 Microchip MPLAB Development Environment	12
3.2 ASIX PRESTO Programmer	12
3.2.1 Description	12
3.2.2 Flashing Environment	13
3.3 CCS C Compiler.....	14
3.3.1 Description	14
3.3.2 Installation	15
3.3.3 Compiling from the command line.....	16
3.3.4 Setting the Configuration Bits	16
3.3.5 Interrupts.....	17
3.3.6 General Structure of Source Code	18
3.4 Tiny Bootloader.....	19
3.4.1 Compiling Bootloader Source Code.....	20
3.4.2 Usage	21
3.4.3 Changes to the User's Source Code	22
4 Inductive Loop Detector	23
4.1 Device Description	23
4.2 Loop Oscillator Circuit.....	26
4.2.1 Circuit based on Comparator.....	26

4.2.2	Circuit based on Operational Amplifier	27
4.2.3	Surge Protection	28
4.3	Microcontroller.....	29
4.3.1	PIC18F2420.....	30
4.3.2	Frequency measurement.....	31
4.3.3	Sample Buffer.....	32
4.3.4	Calibration	33
4.3.5	Vehicle Recognition	34
4.3.6	LED Outputs.....	36
4.3.7	Setting Device Parameters.....	36
5	Traffic Counter.....	38
5.1	Device Description	39
5.2	Firmware Implementation	41
5.2.1	Vehicle Measurement.....	41
5.2.2	Date and Time	43
5.2.3	Measuring Temperature.....	44
5.2.4	Vehicle and Temperature Samples Buffer	45
5.3	Device Settings and Parameters	47
5.4	Command-line Interface.....	48
5.4.1	Character Buffer	48
5.4.2	Buffer Processing	49
5.4.3	Usage	50
5.5	GPRS Modem.....	50
5.5.1	Description	50
5.5.2	Communication	51
5.5.3	AT Commands.....	52
5.5.4	Modem Configuration	52
5.5.5	GPRS Connection to Server	53
5.5.6	Server Protocol	54
5.5.7	Transferring Samples.....	55
6	Server Applications.....	56
6.1	Server Installation.....	56
6.2	Database Model	56
6.3	Traffic Server.....	58
6.3.1	Sockets in PHP	59
6.3.2	Handling Multiple Clients	59
6.3.3	File Formats.....	60
6.3.4	Binary Data Conversion	62
6.3.5	Event Log	62
6.3.6	Server Configuration	63
6.3.7	Running the Server	63
6.4	Traffic Statistics.....	64
6.4.1	jQuery	65
6.4.2	JpGraph.....	68
6.4.3	Traffic Charts API	68
6.5	Database Benchmarking.....	70
7	Conclusion	72
7.1	System Summary	72
7.2	Tests in the field	72
7.3	Further Work	73

8	References and Bibliography	74
9	Software and Packages Used	76
Appendix 1	Content of the Attached CD	77
Appendix 2	Inductive Loop Detector Schematic	78
Appendix 3	Inductive Loop Detector PCB	79
Appendix 4	Inductive Loop Detector Bill of Materials	80
Appendix 5	Inductive Loop Detector User's Manual	81
Appendix 6	Traffic Counter Schematic.....	86
Appendix 7	Traffic Counter PCB.....	87
Appendix 8	Traffic Counter Bill of Materials.....	89
Appendix 9	Traffic Counter User's Manual.....	90
Appendix 10	Traffic Counter: List of Commands	96
Appendix 11	Database Create Script	98
Appendix 12	Example Source Code for CCS C Compiler	100

List of Figures

Figure 1.1 System Overview	1
Figure 2.1 Inductive Loop and Vehicle Presence.....	5
Figure 2.2 Detection and Object Orientation.....	6
Figure 2.3 Vehicle Measurement Principle	8
Figure 2.4 Signals From Loops	8
Figure 3.1 PIC18F2320 in PDIP package	10
Figure 3.2 Microchip MPLAB IDE - layout	12
Figure 3.3 ASIX PRESTO Programmer	13
Figure 3.4 ASIX-UP Flashing Environment	13
Figure 3.5 PIC Standard PDIP Programming Pinouts.....	14
Figure 3.6 CCS C Compiler IDE.....	15
Figure 3.7 CCS C Compiler Fuses Quick View	16
Figure 3.8 CCS C Compiler Interrupt Quick View	17
Figure 3.9 Tiny Bootloader Memory Layout	19
Figure 3.10 Tiny Bootloader Source Code Modification	20
Figure 3.11 Tiny Bootloader PC Application.....	22
Figure 4.1 Inductive Loop Detector	23
Figure 4.2 ILD System Overview	24
Figure 4.3 ILD Board Layout.....	24
Figure 4.4 ILD Loop Connector.....	25
Figure 4.5 ILD Main Connector	25
Figure 4.6 Oscillator Circuit with Comparator.....	26
Figure 4.7 Oscillator Circuit with Operational Amplifier	27
Figure 4.8 Simple Conversion from Sine Wave to TTL Signal	28
Figure 4.9 Overvoltage Protectors.....	28
Figure 4.10 Overvoltage Protection.....	29
Figure 4.11 ILD Operation Diagram	30
Figure 4.12 PIC18F2420 pinout	31
Figure 4.13 CCP module in Capture mode.....	31
Figure 4.14 Vehicle Recognition State Machine.....	34
Figure 4.15 State Change Hysteresis.....	35
Figure 5.1 Traffic Counter.....	38
Figure 5.2 Traffic Counter System Overview	39
Figure 5.3 Traffic Counter Board Layout.....	40
Figure 5.4 PC and GPRS Modem Connectors	40
Figure 5.5 Detectors and Microcontroller Connection.....	41
Figure 5.6 Vehicle Measurement State Machine.....	42
Figure 5.7 Maxim DS1302 Basic Circuit	43
Figure 5.8 Maxim DS18B20 Pinout and Circuit	45
Figure 5.9 Samples Memory Layout	46
Figure 5.10 Buffer Processing Algorithm	49
Figure 5.11 TENcom SPEEDER RS GPRS Modem	51
Figure 5.12 GPRS Connection Process	53
Figure 5.13 Client-Server Communication Diagram	55
Figure 6.1 Database Model.....	57
Figure 6.2 Traffic Server Operation	58

Figure 6.3 Binary File Structure	61
Figure 6.4 Traffic Statistics Web Application.....	64
Figure 6.5 Datepicker plugin for jQuery	66
Figure 6.6 Tablesorter Plugin for jQuery	67
Figure 6.7 Simple Line Plot via JpGraph Library	68
Figure 6.8 Charts Generated in Traffic Charts API.....	70

List of Tables

Table 3.1 Microchip PIC Configuration bits (selection)	11
Table 3.2 List of Important Devices Supported by ASIX PRESTO	13
Table 4.1 ILD Connector Description	25
Table 4.2 Frequency Configuration	36
Table 4.3 Device Sensitivity Configuration	37
Table 5.1 DS1302 Driver API	43
Table 5.2 Memory Space and Buffer Sizes	46
Table 5.3 Device Parameters	47
Table 5.4 Speeder RS Specification	51
Table 5.5 List of Important AT Commands	52
Table 5.6 Client Authorization Headers	54
Table 6.1 Binary File Headers	61
Table 6.2 Unpack Function Little-endian Parameters	62
Table 6.3 Server Configuration Attributes	63
Table 6.4 Traffic Charts API Variables	69
Table 6.5 Traffic Charts API	69
Table 6.6 MySQL Query Benchmark	71

List of Abbreviations

ANSI	American National Standards Institute
CAN	Controller Area Network
CSV	Comma-Separated Values
CTU	Czech Technical University
DCE	Data Communication Equipment
DIP	Dual-In-line Package
DOM	Document Object Model
DTE	Data Terminal Equipment
EEPROM	Electrically Erasable and Programmable Read-Only Memory
FEE	Faculty of Electrical Engineering
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile communications
HTML	HyperText Markup Language
ICSP	In-Circuit Serial Programming
IDE	Integrated Development Environment
ILD	Inductive Loop Detector
ITS	Intelligent Transportation Systems
LED	Light Emitting Diode
LPR	Licence Plate Recognition
MSSP	Master Synchronous Serial Port
PC	Personal Computer
PCB	Printed Circuit Board
PDIP	Plastic Dual-In-Line Package
PHP	Hypertext Preprocessor Language
PIC	Programmable Intelligent Computer
PLL	Phase-Locked Loop
PWM	Pulse-Width Modulation
RISC	Reduced Instruction Set Computer
RTC	Real-Time Clock
SPI	Serial Peripheral Interface

SQL	Server Query Language
TTL	Transistor-Transistor Logic
TVS	Transient Voltage Suppressor
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
UTC	Coordinated Universal Time
WIM	Weight-In-Motion

1 Introduction

The overview of the system is depicted in Figure 1.1. The thesis deals with the development of an Inductive Loop Detector (ILD – Chapter 4) used for sensing the presence of a conductive metal object near the loop. In other words, to find out whether a vehicle enters/is above the loop. The detector has to have the ability to cope with different loop inductances as well as temperature drifts, and therefore, an auto-calibration feature had to be introduced.

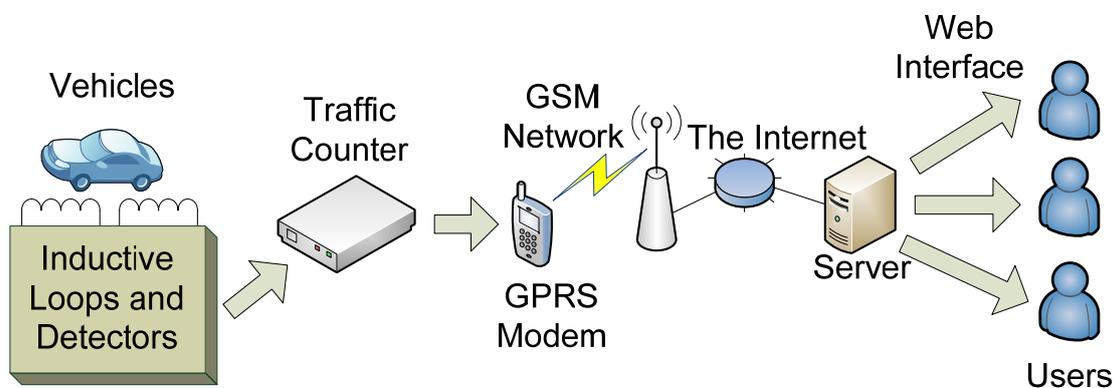


Figure 1.1 System Overview

For gathering traffic data such as speed and length of vehicles, another device has been developed. Traffic Counter (Chapter 5) uses simple principle to measure speed and length of a vehicle by placing two inductive loops (including two ILDs) within a traffic lane. The device can gather data from up to 2 traffic lanes, samples are stored in an internal memory, and from time to time, the buffer in binary form is transferred over a GPRS connection to the server. Furthermore, station temperature data is also logged and transferred to the server. The Traffic Counter provides a command-line interface so that the user can configure and monitor the device.

A server application (Traffic Server in Chapter 6.3) for receiving data from stations has been written in PHP language, which allows the developer to use high-level functions without the necessity of taking care of matters whose are implicit in PHP such as memory management. Another advantage is the possibility of running the application on different platforms such as Linux or Microsoft Windows, because both operating systems support PHP scripting. The server application gathers binary data

from stations, puts them into files, and also uploads them to the database for further processing.

For an user-friendly output, a PHP web interface (Traffic Statistics - Chapter 6.4) has been created. Charts and statistical information on stations such as speed and length categories, number of vehicles per day/hour, with the possibility of selecting date range and other constraints, can be generated via the web interface.

2 Motivation and Methodology

In nowadays, road and highway traffic has become a great concern because of continuous increase in traffic, which calls for new roads to be built. One matter is to build them, the second one is to optimise the current transportation system, and therefore, to improve efficiency of existing roads. Because of that, it is essential to gather traffic data, not only for statistical information but mainly for analysis and further traffic control.

There are many technologies for gathering traffic data (Chapter 2.2). One of well-working principles is acquisition based on inductive loops embedded in the roadway, which is the concern of this thesis. Chapter 2.4 and Chapter 2.5 deal with the principles of that method.

Later on, the traffic data collected can be used for analysis, prediction, and control of traffic in higher-level technology – ITS, whose aim is to increase safety, and to reduce road congestions and fuel consumption. The traffic data can also be used as a source for information, guidance, and assistance systems for drivers.

2.1 Reason for this Thesis

Vehicle detection and measurement based on inductive loops has been in place for tens of years, and it is the most used technology. Over the world, especially in the US, it is quite easy to buy an inductive loop detector device and a traffic counter device. However, in the Czech Republic, it is not so easy since companies either get those devices from abroad, or develop and use them for their own purpose without further providing/selling them.

Therefore, the purpose of this thesis was to develop both detector and counter, based on simple and well-available components. Given that the inductive loop detector can also be used at stop bar places in parking where vehicle presence needs to be found out, or in automation for metal proximity sensing, it was a major concern to develop own and low-cost detector.

2.2 Review of Vehicle Detection Technologies

Vehicle detection technologies are non-intrusive and intrusive. The non-intrusive principle uses a sensor built in the roadway, the latter employs a sensor mounted above or at the side of the roadway. Both technologies have their pros and cons. An in-depth discussion on both principles can be found in [2, 3, 5].

2.2.1 Non-Intrusive (Over-Roadway)

Non-intrusive technology uses microwave, ultrasonic, active and passive infrared, acoustic, and laser principle. An advantage is the ease of installation, above or at the side of the roadway. A disadvantage is the need for sensor cleaning, and the susceptibility to weather condition such as fog, heavy rain and others. Some of over-roadway detection devices use more than one sensors in order to overcome the drawbacks of each sensor.

In nowadays, video image processors have been incorporated within traffic sensors. They are easy to set up, and are able to monitor several traffic lanes at a time. Moreover, it is also possible to introduce Licence Plate Recognition (LPR) to the system since an image of a vehicle is already present in digital form.

2.2.2 Intrusive (In-Roadway)

Examples of this category are inductive and magnetic loops, magnetometers, tape-switches, and piezoelectric cables. Installation and maintenance of one of those sensors means disruption of traffic. However, especially reliability is usually better than in non-intrusive sensors since in-roadway sensor measurement is not susceptible to weather conditions. The latest inductive loop detectors also provide a better accuracy than over-roadway sensors.

Recently, there has been a demand for Weight-in-Motion (WIM) technology [5], which is capable to measure speed, length, and gross+axle weight of a vehicle. The WIM usually combines piezoelectric and inductive-loop sensors into one system, and it is mainly used to detect overweight vehicles.

2.3 Inductive Loops in General

The detection is based on sensing a change of loop inductance when a vehicle approaches the loop. The loop is embedded in the pavement where saw cuts are made to place the wire into the roadway. A change in magnetic flux caused by a vehicle (i.e. its chassis) is depicted in Figure 2.1. There are several approaches of detecting the change of inductance such as resonant and phase-shift methods.

The resonant principle is based on an oscillator circuit where the inductive loop is its part. The change in inductance causes a change in resonant frequency. Therefore, in the detector, frequency measurement has to be carried out, which is a simple task since it can be achieved easily with a microcontroller. The resonant method (Chapter 2.4) has been used in this thesis (ILD device in Chapter 4).

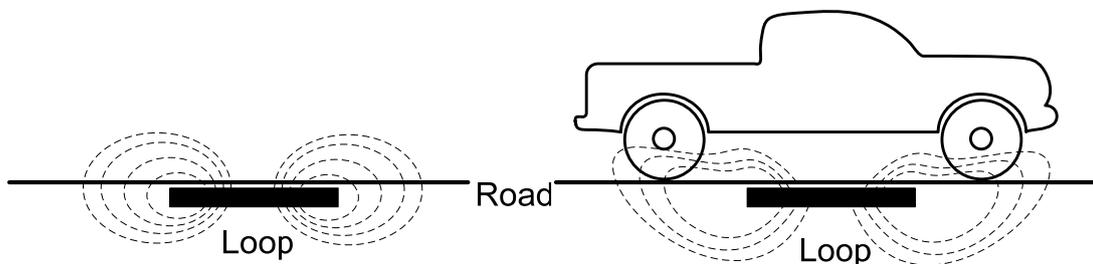


Figure 2.1 Inductive Loop and Vehicle Presence

Vehicle measurement can be performed by a single inductive loop. The advantage is that it is simpler than two-loop system, because of saw cuts and measurement equipment. However, this method can only estimate the vehicle speed based on statistical information [4], there is no possibility to measure length of the vehicle.

In contrast, the two-loop method is capable to measure vehicle speed and length quite accurately. This method has been used in this thesis, and is discussed in Chapter 2.5 and Chapter 5.

Furthermore, it can also be achieved to specify the vehicle class by calculating the number of axles of the vehicle [8]. The loop frequency is sampled in periods of milliseconds, and the resulting curve contains information on the axle count. Similarly, the same method can be used to produce a vehicle signature [6, 7], which can be used for vehicle matching between two remote places.

2.4 Vehicle Detection Principle Used

2.4.1 Changes in Inductance

The inductive loop detector behaves as a tuned electrical circuit. When a vehicle passes above the loop, the vehicle induces eddy currents in wire of the loop, which decreases its inductance. The outcome is a change in oscillator frequency of the circuit. Figure 2.2 shows the effect of orientation of a metal object on the detection. There is a misconception that the detection is based on metal mass. Instead of that, surface metal area has the biggest impact.

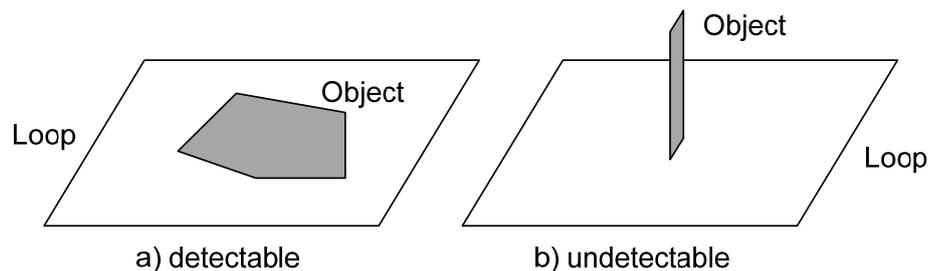


Figure 2.2 Detection and Object Orientation

In addition, there is a ferromagnetic effect caused by the iron mass of the engine, transmission, or differential. This effect causes an increase of inductance when such part of a vehicle enters the loop. Nonetheless, the surrounding metal of the vehicle always has a stronger impact, and therefore, the overall inductance will decrease in spite of that.

2.4.2 Resonant Frequency

The resonant frequency of LC tank circuit is:

$$f_{osc} = \frac{1}{2\pi\sqrt{LC}} \quad [Hz, H, F] \quad (2.1)$$

In order to maintain fast responses of the detector, the resonant frequency should be set approximately between **20 kHz and 120 kHz**.

The Quality factor of the loop is expressed by

$$Q = \frac{2\pi f \cdot L_s}{R_s} \quad [-, Hz, H, Ohm] \quad (2.2)$$

Where L_s = Loop series inductance, R_s = Loop series resistance

The resulting frequency is given by:

$$f_{osc} = \frac{1}{2\pi\sqrt{LC\left(1 + \frac{1}{Q^2}\right)}} \quad [Hz, H, F, -] \quad (2.3)$$

An important thing to note is that the resonant frequency changes slightly when there is resistance in the circuit. In this application, when the inductive loop in the road is about 20-40 meters long, its resistance might be up to several Ohms, which influences the oscillation frequency a bit. Nevertheless, the task of the ILD is not to measure the value of the loop inductance accurately, but to detect the change in inductance only.

2.4.3 Inductance Values and Calculations

The following formulas have been used to determine loop inductance and dimension (refer to [20] for further description).

a) Circular loop

$$L = N^2 R \mu_0 \mu_R \left[\ln\left(\frac{8R}{a}\right) - 2 \right] \quad (2.4)$$

where L = Inductance [H]

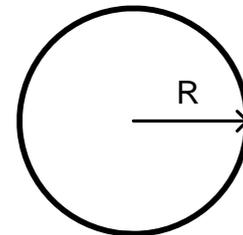
N = Number of turns [-]

R = Radius of the circle [m]

$\mu_0 = 4\pi \cdot 10^{-7}$ = Permeability of vacuum [H/m]

μ_R = Relative permeability [-]

a = Radius of the wire [m]



b) Rectangular loop

$$L = \frac{N^2 \mu_0 \mu_R}{\pi} \left[-2(w+h) + 2\sqrt{h^2 + w^2} - h \cdot \ln\left(\frac{h + \sqrt{h^2 + w^2}}{w}\right) - w \cdot \ln\left(\frac{w + \sqrt{h^2 + w^2}}{h}\right) + h \cdot \ln\left(\frac{2w}{a}\right) + w \cdot \ln\left(\frac{2h}{a}\right) \right] \quad (2.5)$$

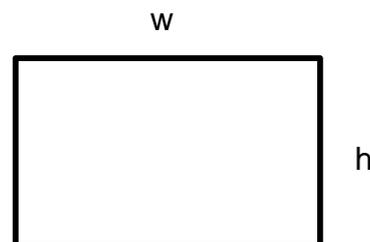
where L = Inductance [H]

N = Number of turns [-]

w = Width of the rectangle [m]

h = Height of the rectangle [m]

a = Radius of the wire [m]



2.5 Traffic Measurement Principle Used

The principle is shown in Figure 2.3. There are 2 inductive loops built within the roadway. Each loop has length l meters, and there is d meters distance between the loops. Each loop is connected to one inductive loop detector and signals carrying vehicle presence are used.

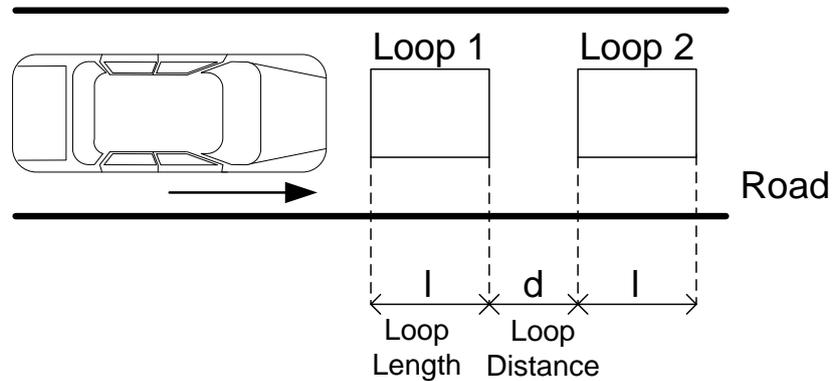


Figure 2.3 Vehicle Measurement Principle

2.5.1 Signals from Detectors

The measurement is carried out through a sequence of impulses loop detectors (see Figure 2.4). In order to avoid crosstalks and other noise interferences, the sequence is strictly given:

1. Vehicle enters the first loop
2. Vehicle enters the second loop
3. Vehicle leaves the first loop
4. Vehicle leaves the second loop

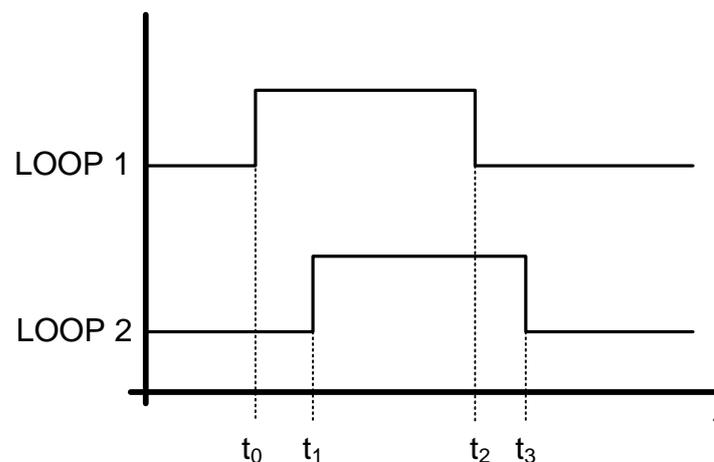


Figure 2.4 Signals From Loops

Other sequences will be ignored. Thus, there arises a **limitation of this method:**

Vehicles shorter than Loop Distance will be not be measured.

However in practice, loop distance is approximately 2 meters, vehicles shorter than that value are not considered as vehicles. Therefore, this limitation is just a minor drawback.

2.5.2 Measurement of Time Intervals

The measurement starts when a vehicle enters the first loop (t_1). When the vehicle enters the second loop (t_2), its speed can be calculated:

$$v_1 = \frac{l + d}{t_2 - t_1} \quad [m.s^{-1}, m, s] \quad (2.6)$$

Subsequently, When the vehicle leaves the first loop (t_3), its length can be calculated:

$$l_1 = v_1(t_3 - t_1) - l \quad [m, m.s^{-1}, s] \quad (2.7)$$

Another speed and length can be calculated when the vehicle leaves the first and the second loop respectively (t_3, t_4):

$$v_2 = \frac{l + d}{t_4 - t_3} \quad [m.s^{-1}, m, s] \quad (2.8)$$

$$l_2 = v_2(t_4 - t_3) - l \quad [m, m.s^{-1}, s] \quad (2.9)$$

In theory, the both inductive loop detectors are supposed to behave the same and to detect a vehicle in the same point of its chassis. Therefore, the following equations are satisfied:

$$\text{Speed: } t_2 - t_1 = t_4 - t_3 \quad (2.10)$$

$$\text{Length: } t_3 - t_1 = t_4 - t_2 \quad (2.11)$$

However, in the real application, the **equalities will never be satisfied**. For instance, detector responses are not exactly the same, or a vehicle does not enter both loops in the same position. In order to make vehicle speed and length more accurate, calculated values v_1, v_2 and l_1, l_2 **can be averaged**.

3 Microchip PIC and Equipment

Microchip PIC microcontrollers are popular because of low cost, availability, various collections of source codes, and free/cheap development tools. Those 8-bit microcontrollers have Harvard architecture (program and data memory spaces are separated) which together with RISC instruction set ensures that the most instructions are executed within one machine cycle.

3.1 PIC18 Family

3.1.1 Description

PIC18 family (example shown in Figure 3.1) is based on the previous PIC16 family, fixes its drawbacks, and introduces upgrades such as:

- Much deeper call stack (31 levels)
- The call stack may be read or write
- Conditional branch instructions
- Indexed addressing mode

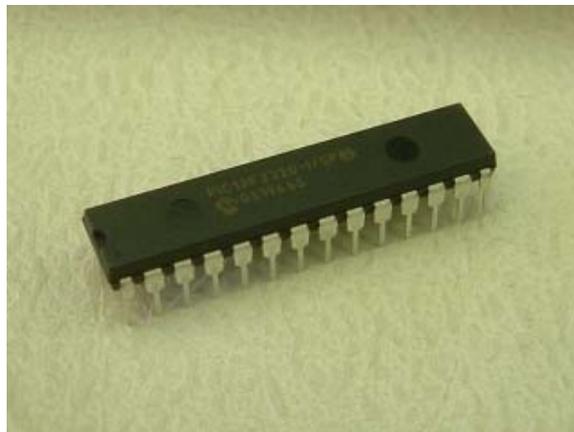


Figure 3.1 PIC18F2320 in PDIP package

The most important is the deeper call stack and highly optimised instruction set optionally with extended instructions. This delivers the possibility of using a higher-level programming language such as C language.

The PIC18 microcontrollers can (with some exceptions) run at a clock speed up to 40MHz, and they are generally equipped with the following on-board peripherals:

- Watchdog timer
- Internal clock oscillator

- PLL for multiplying the clock frequency
- 8/16 Bit Timers (3)
- Internal EEPROM memory
- Synchronous/Asynchronous Serial Interface USART
- MSSP Peripheral for I²C and SPI Communications
- Compare/Capture and PWM modules
- Analog-to-digital converter
- Analog Comparator

There are many variants of PIC18 microcontrollers having different on-board peripherals (an example of a model in brackets):

- CAN (18F2480)
- USB (18F2455)
- Ethernet (18F66J60)

3.1.2 Configuration Bits

Every PIC microcontroller contains the configuration bits (so-called fuses) whose are used to tell the microcontroller how it should handle external environment. These bits are mapped at the start of program memory at 0x300000, and have to be set during programming the chip. The fuses set behavior of PIC on-board peripherals such as Watchdog, Oscillator, Memory protection, and others.

Some important fuses are listed in Table 3.1, a full description of fuses can be found in the Microchip PIC device's data sheet [15, 16]. Detailed process of setting the fuses can be found in Chapter 3.3.4.

Fuse	Description	Fuse	Description
XT	Crystal oscillator (<4Mhz)	HS	Crystal oscillator (>4Mhz)
XTPLL	Crystal oscillator (<4Mhz) with frequency multiplier	H4	Crystal oscillator (>4Mhz) with 4x frequency multiplier
NOWDT	no Watchdog Timer	NOLVP	Disable low voltage programming pin
NODEBUG	don't enable debug pins for ICD	MCLR	Master Clear Reset pin enabled
NOPROTECT	Code not protected from reading	NOEBTRB	Boot block not protected from table reads
NOWRT	Program memory not write protected		

Table 3.1 Microchip PIC Configuration bits (selection)

3.1 Microchip MPLAB Development Environment

MPLAB IDE (Figure 3.2) is a free toolset for managing projects for Microchip microcontrollers: writing source code, compiling, simulating, and debugging. It contains interface for additional Microchip or third-party development tools such as C compilers and PIC emulators/real-time debuggers. Application contains integrated MPASM assembler language compiler.

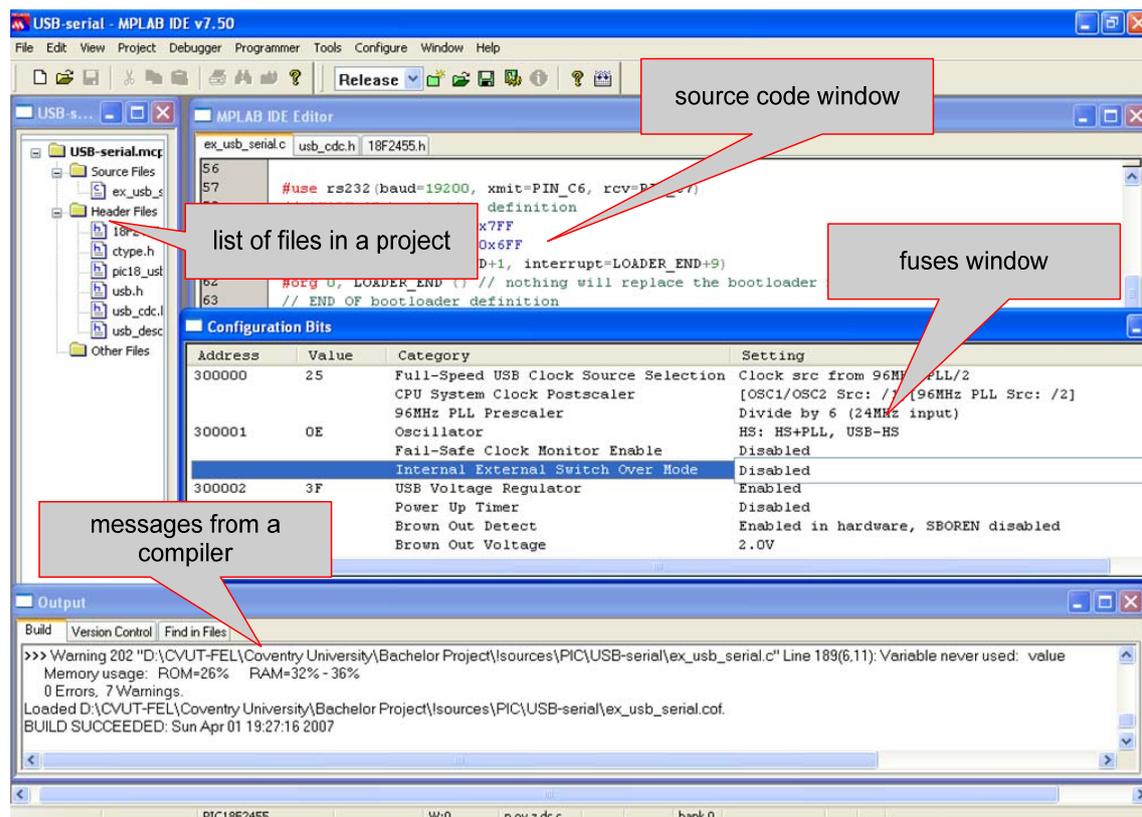


Figure 3.2 Microchip MPLAB IDE - layout

3.2 ASIX PRESTO Programmer

3.2.1 Description

ASIX PRESTO Programmer (Figure 3.3) is an USB 1.1/2.0 programmer which supports many devices ranging from Serial EEPROM/Flash memories through 8-32 Bit microcontrollers to CPLD/FPGA microprocessors. An important list of devices supported can be found in Table 3.2. The programmer supports ICSP programming, and since it is usb-powered, it does not need any power supply for programming

microcontrollers. Given that the device uses an USB interface chip from FTDI company [9], it is possible to use the programmer either under Windows or Linux.



Figure 3.3 ASIX PRESTO Programmer

Manufacturer	Devices Supported	Manufacturer	Devices Supported
Microchip	PIC16, PIC18, dsPIC, PIC24, PIC32	Atmel	AVR, 8051
Atmel, Philips, NXP	ARM	Xilinx, Altera, Lattice	CPLD and FPGA devices
Various	Serial EEPROM and Flash memories		

Table 3.2 List of Important Devices Supported by ASIX PRESTO

3.2.2 Flashing Environment

The programmer has its own environment *Asix UP* [25] in both English and Czech languages (Figure 3.4) for flashing microcontrollers, which can also be run under Linux.

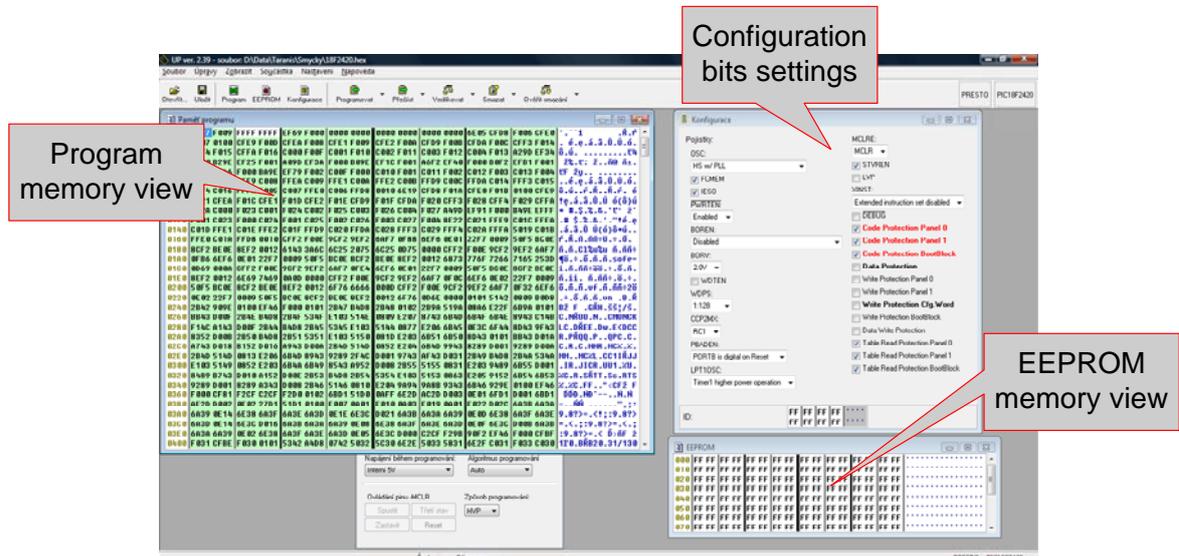


Figure 3.4 ASIX-UP Flashing Environment

Within the application, a hex file, containing compiled source code, has to be loaded, and a microcontroller can be flashed with the file afterwards. It should be noted that even though the hex file contains settings for Configuration bits (Chapter 3.1.2) of the microcontroller, the application allows the user to change those bits just before flashing (see *Configuration bits settings* in Figure 3.4). Figure 3.5 [18] contains standard programming pinouts for PDIP package.

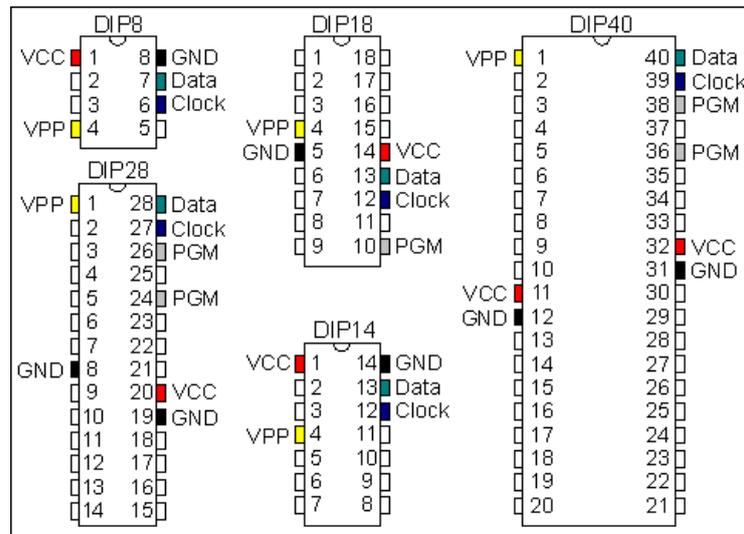


Figure 3.5 PIC Standard PDIP Programming Pinouts

3.3 CCS C Compiler

3.3.1 Description

CCS C Compiler [27] is an ANSI C compiler developed exclusively for Microchip PIC microcontrollers. It comes with an extensive collection of libraries (RS232, CAN, SPI, I2C, USB and others) as well as many examples, which make it very easy to use.

It can be bought with or without IDE (Figure 3.6), it should be noted that the compiler itself can be integrated into Microchip MPLAB IDE (Chapter 3.1). However, the advantage of the original IDE is that it fully supports compiler features such as prompt, list of fuses/interrupts (see Chapter 3.3.4 and Chapter 3.3.5), statistics and memory usage, whose improve and speed up development process. Thus, given that I have used the IDE for this thesis, I think that it is worth to get the IDE as well.

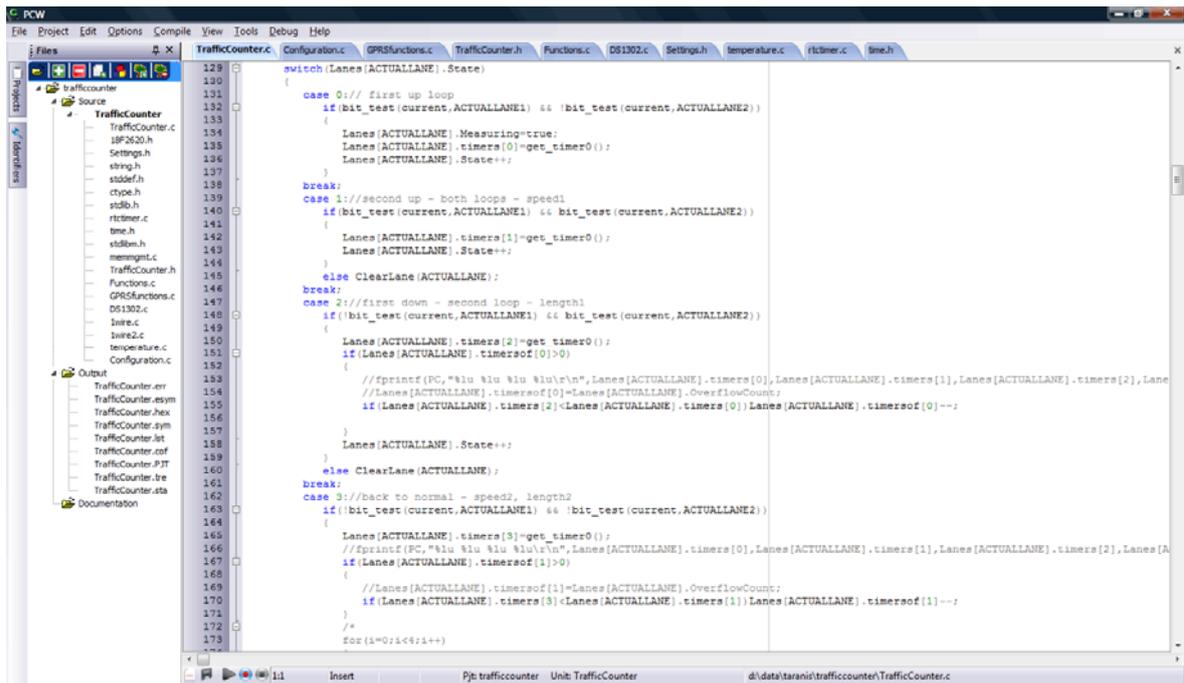


Figure 3.6 CCS C Compiler IDE

The compiler has to be obtained for particular PIC family, or as a combination of them:

- **PCB** - for 12-bit PIC MCUs (PIC10, PIC12)
- **PCM** - for 14-bit PIC MCUs PIC16
- **PCH** - for 14-bit PIC MCUs PIC18
- **PCD** - for 24-bit PIC MCUs PIC24/dsPIC

It should be noted that the PIC18 family only has been used for this thesis, and hence, PCWH IDE+PCH compiler have been used for development.

3.3.2 Installation

The compiler is as default installed into *C:\Program Files\PICC* directory. There is the following structure of its subfolders:

- **Devices** – header files (.h) for PIC microcontrollers
- **Drivers** – libraries for various devices (e.g. memories, sensors, displays), drivers for CAN and USB modules
- **Examples** – example source codes

3.3.3 Compiling from the command line

We do not need any IDE for compiling a source code since it can also be compiled directly from the command line:

- we can run “C:\Program Files\PICC\Ccsc.exe” and select a .C file which contains *main()* function of the PIC application
- in case of compiling directly from the command line there is the following format “Ccsc.exe *options filename*” where *options* are described in [10].

example : *Ccsc.exe +FH C:\picsources\file.c*

3.3.4 Setting the Configuration Bits

The compiler produces a hex file ready to be flashed into a microcontroller. The hex file also contains Configuration bits (Chapter 3.1.2). Therefore, in order to get the fuses configured, a special keyword in the source code followed by fuse description has to be used:

```
#fuses HS, NOWDT, NOPROTECT, NOLVP, NOPBADEN, WRTB, NOCPD, NOWRTC
```

In case of using the CCS Compiler IDE (Chapter 3.3.1, Figure 3.6), a list of fuses available (Figure 3.7) for selected target microcontroller can be viewed by choosing *View>Valid Fuses* from the IDE menu toolbar.

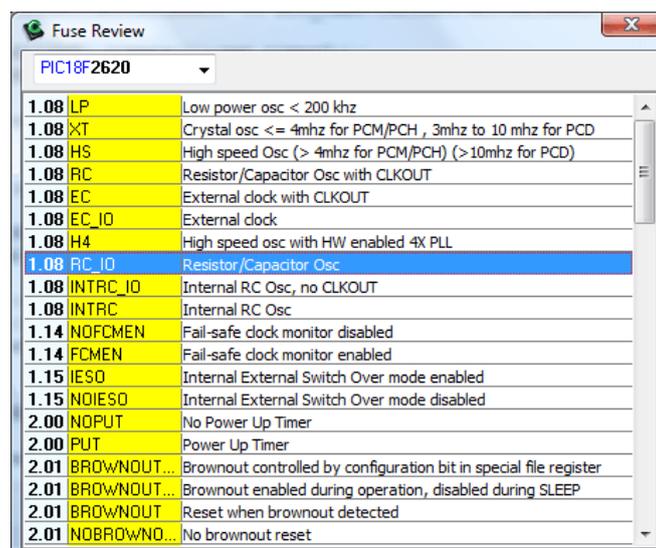


Figure 3.7 CCS C Compiler Fuses Quick View

3.3.5 Interrupts

The Microchip PIC family offers various internal/external interrupts such as timer overflow, external edge trigger, serial line receive and others. Similarly, to previous subchapter, the CCS Compiler IDE provides a quick list (Figure 3.8) of interrupts available for particular microcontroller.

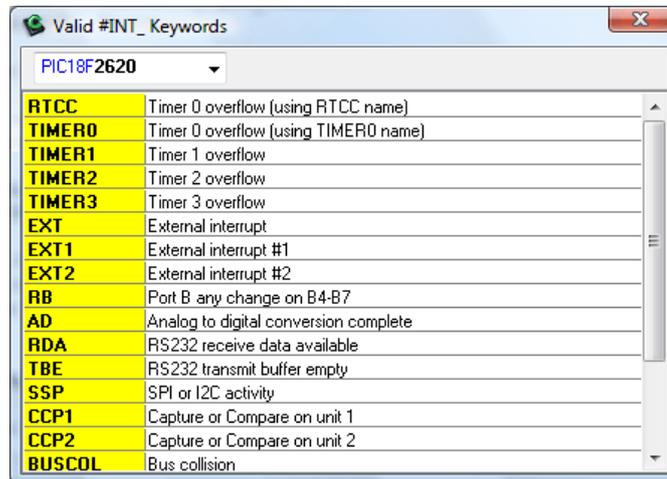


Figure 3.8 CCS C Compiler Interrupt Quick View

PIC18 family introduces priority interrupts, where an interrupt can either be low or high priority. High-priority interrupt events will interrupt any low-level interrupts that may be in progress. Entire interrupt logic can be found in [15].

3.3.5.1 Setting an Interrupt

The compiler provides function for dealing with interrupts. At first, a peripheral which will fire an interrupt has to be set:

```
setup_timer_1(T1_INTERNAL | T1_DIV_BY_8);
```

Subsequently, the appropriate interrupt, including general interrupt flag, has to be enabled:

```
enable_interrupts(INT_TIMER1);
enable_interrupts(GLOBAL);
```

Finally, an interrupt service routine has to be put somewhere in the source code:

```
#int_TIMER1
void TIMER1_isr()
{
    // this routine is executed when timer1 overflows
}
```

3.3.5.2 High-priority Interrupt

If the user intends to use high-priority interrupts, the following has to be placed at the beginning of source code:

```
#device HIGH_INTS=true
```

After that, the user has to mark some particular interrupt with high-priority:

```
#int_CC1 HIGH
void CC1_isr()
{ // interrupt handler function
}
```

It shall be pointed out that according to my experience, if the user enables high-priority interrupts, and he does not set any interrupt to be high-priority, the **microcontroller behaves strange and unpredictably** when the compiled source code is flashed and run. This might either be a compiler bug or a bug of a microcontroller.

3.3.6 General Structure of Source Code

Like other C language compilers, source codes consists of .c and .h files, and there also has to be *main* function within the main .c file. All microcontroller firmware codes written within this thesis have the following structure and order:

- Include device header file (ex.: *include <18f2420.h>*)
- Set internal device configuration (*#device* command)
- Set configuration bits (*#fuses* command)
- Include other project files (.h .c) and function prototypes
- Interrupt handler routines, other functions
- Main function (*void main()*)

An example of source code can be found in Appendix 12.

3.4 Tiny Bootloader

Generally, the term “bootloader” means a small block of code that is executed once a device is reset or powered on. In terms of embedded systems, the bootloader is used for downloading a firmware to a device without the need for programming equipment (i.e. programmer).

Tiny Bootloader [26] is a bootloader for Microchip PIC16, PIC18, dsPIC30 families that provide self-programming. It allows flashing the microcontroller via its serial line interface, which is usually present within every application. This speeds up development markedly, because the chip does not have to be taken out of a board.

Given that the source code of the bootloader is freely available and is written in assembler language for PIC, it can easily be adapted to any PIC device running at various clock-rate and having various serial line speed.

Assets of the bootloader are its size, which is merely 100 words, and also the fact that it comes with a PC application for writing a hex file into the microcontroller. The following subchapters describe process of compiling and employment of the Tiny Bootloader.

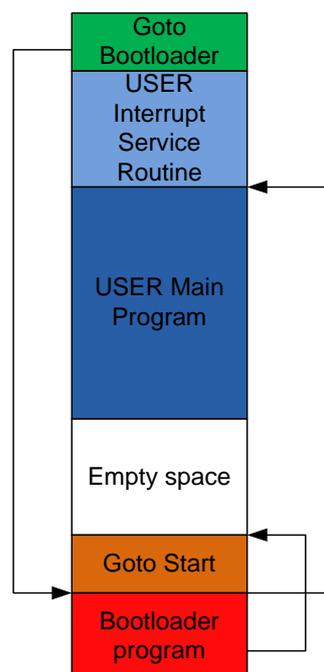


Figure 3.9 Tiny Bootloader Memory Layout

The bootloader memory structure is depicted in Figure 3.9. It resides at the end of program memory. When the microcontroller is powered on, it waits for a while for an initialization char over its serial line. When there is nothing received, user

firmware is executed. Otherwise, the bootloader is run and begins to communicate with the PC.

3.4.1 Compiling Bootloader Source Code

The bootloader comes with source code written in assembler languages (MPASM) for PIC, and several precompiled hex files. In order to use it with a PIC having a custom Configuration (baud rate, fuses, frequency), the source code has to be modified and compiled before flashing it into the microcontroller.

At first, Microchip MPLAB IDE (Chapter 3.1) has to be installed, since it contains assembler language compiler. The compiler has a **limitation** on path length of compiled files. Therefore, it is suggested to place all bootloader files directly in a folder located in the root directory of a drive. Otherwise, in case of long path to files, the compilation of the source code will fail.



Figure 3.10 Tiny Bootloader Source Code Modification

Source codes particularly for PIC18 family are located under */picsource/pic18/* directory. If the user wants to use an universal bootloader, *tinybld18F.asm* file has to be modified (see Figure 3.10) according to the following:

- Select device by writing its model name (i.e 18f2420). It is also **necessary** to change the device in MPLAB environment from Menu Toolbar : *Configure > Select Device...* Otherwise, it will be compiled **incorrectly**.
- Modify crystal speed and serial line baud rate

- Set Configuration bits. A complete list of fuses suitable for a device, including fuse description, can be found in device's *.inc* file (e.g. *P18F2420.INC*) under *C:\Program Files\Microchip\MPASM Suite*.

An example of device configuration taken from the beginning of *tinybld18F.asm* file:

```
LIST      P=18F2420
xtal EQU 20000000
baud EQU 115200
CONFIG OSC = HS, FCMEN = ON, IESO = ON, PWRT = ON
CONFIG BOREN = OFF, WDT = OFF, MCLRRE = ON, LPT1OSC = OFF
CONFIG PBADEN = OFF, CCP2MX = PORTC, STVREN = ON
CONFIG XINST = OFF, DEBUG = OFF, LVP=OFF
```

Once the source code has been successfully compiled, it can be flashed into the microcontroller.

3.4.2 Usage

The Tiny Bootloader contains a PC application (Figure 3.11) for downloading the user firmware to the microcontroller. The use of the application is straight-forward. The user has to select a COM port where the microcontroller is connected to, and the communication speed (i.e. baud rate). A hex file containing compiled user firmware has to be opened via *Browse* button. After that, the target microcontroller has to be put in reset by disconnecting its power supply or by pulling low its MCLR pin. Consecutively, the user has to release the microcontroller from reset state (e.g. pull MCLR pin high), followed by clicking on *Write Flash* button. Finally, the user application should be flashed into the microcontroller. If the user does not click on *Write Flash* Button (i.e. PC application will not send initialization to the chip), the user firmware in a microcontroller will be executed.

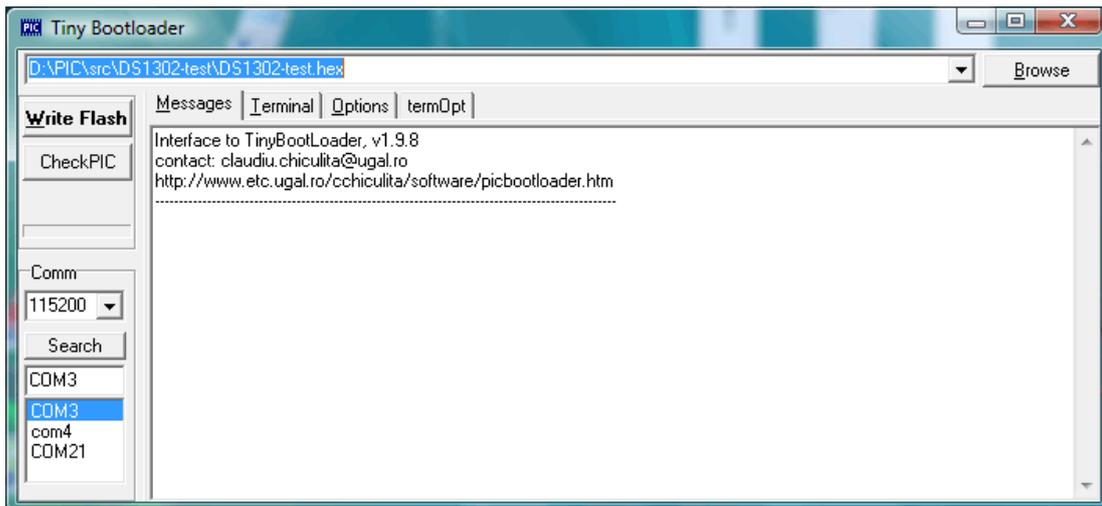


Figure 3.11 Tiny Bootloader PC Application

3.4.3 Changes to the User's Source Code

The Tiny Bootloader resides at the very end of program memory. Thus, when compiling the user source code, the compiler has to be aware of the fact that nothing can be placed at the end of program memory where the bootloader program resides. Otherwise, the bootloader code would be overwritten and therefore destroyed.

Within the CCS C Compiler, this can be achieved by placing the following definition in source code right after device and fuses definitions.

```
#include <18F2420.h>
#device adc=10
#FUSES NOWDT //No Watch Dog Timer
#FUSES WDT128 //Watch Dog Timer uses 1:128 Postscale
#FUSES H4 //High speed osc with HW enabled 4X PLL
#FUSES PROTECT //Code not protected from reading
#use delay(clock=4000000)

// START OF BOOTLOADER DEFINITION
#define MAX_FLASH getenv("PROGRAM_MEMORY")
#define LOADER_SIZE 0xFF //tinybld size + a bit more (200 bytes is
enough)
#build(reset=0x0000:0x0007)
#org MAX_FLASH-LOADER_SIZE , MAX_FLASH-1 void boot_loader(void) {}
// END OF BOOTLOADER DEFINITION
```

4 Inductive Loop Detector

Inductive Loop Detector (ILD) (Figure 4.1) is a device developed for finding out whether a vehicle is above an inductive loop. Particularly, the presence of a conductive metal object (i.e. axles or a chassis of a vehicle) is recognized by a change in inductance of a loop embedded in the roadway (Chapter 2.4).



Figure 4.1 Inductive Loop Detector

4.1 Device Description

Figure 4.2 depicts a system structure. During the development of this device, it was mandatory to keep in mind that the device had to respond in the shortest time interval possible. Therefore, given that it is quick and easy to implement, one of the fastest ways for measuring inductance was to measure frequency change of a resonant LC (so-called *tank*), whose oscillation frequency changes as one of the values of its components changes.

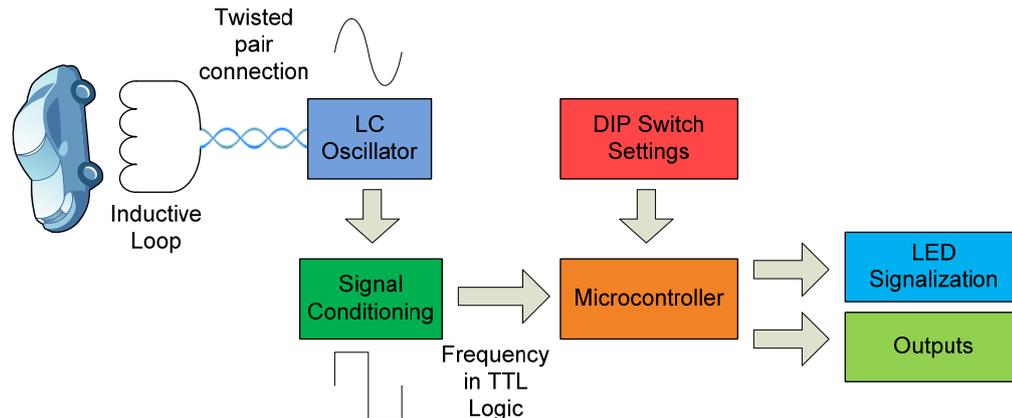


Figure 4.2 ILD System Overview

Furthermore, in order to measure loops having a broad range of inductance, the device contains auto-calibration feature, which also compensates small differences in inductance caused by temperature drifts. The device itself contains an analog part for measuring inductance, and a digital part for analysis, calculations, calibration and decisions. The latter comprises a Microchip PIC18F2420 microcontroller (Chapter 4.3) running at the highest possible frequency (40 MHz), taking into account the amount of calculations to be carried out, and some minor peripherals described in Chapter 4.3.6 and Chapter 4.3.7.

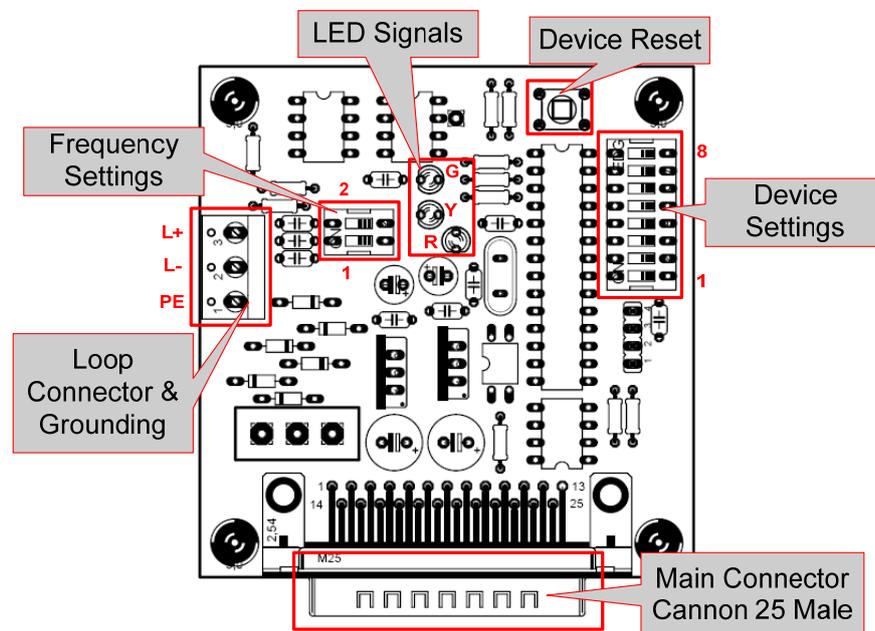


Figure 4.3 ILD Board Layout

Moreover, given that loops are supposed to be placed within the roadway, the device contains surge protectors (transient voltage suppressors – Chapter 4.2.3) on the loop inputs. In order to let the environment know about the state of the device

and the presence of a vehicle above the loop, there are several LED diodes on the ILD, whose function is described in Chapter 4.3.6. Similarly, the device parameters, such as sensitivity, can be set via on-board DIP switches.

Figure 4.3 depicts the board layout. An inductive loop can be either connected through the main connector or the loop connector (Figure 4.4). The device should also be **grounded** via its pin 1.

The device has to be powered through its main connector (Figure 4.5), where inputs and outputs can also be found. Their purpose is described in Table 4.1.

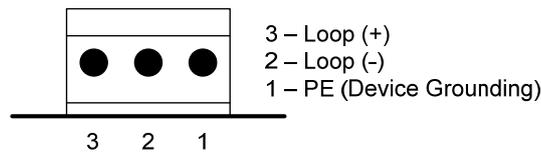


Figure 4.4 ILD Loop Connector

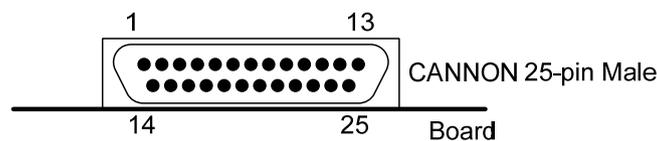


Figure 4.5 ILD Main Connector

Pin No.	Name	Direction	Description
1, 14	L1	In	Inductive Loop (+)
2, 15	PE	In	Grounding to the earth
3, 16	L2	In	Inductive Loop (-) (=GND)
5, 18	DC-	In	Negative Voltage Power Supply (-7 to -12V)
6, 19	GND	In	System Ground
7, 20	DC+	In	Positive Voltage Power Supply (+7 to +12V)
9, 22	RST	In	External Reset (with opto-isolation – against EXTGND)
10, 23	VEHI	Out	Vehicle Above 100ms Impulse (with opto-isolation – against EXTGND)
11, 24	VEH	Out	Vehicle Above Permanent (with opto-isolation – against EXTGND)
12, 25	EXTGND	In	Ground for opto-isolation
13	VEHTTL	Out	Vehicle Above – TTL signal (against GND)

Table 4.1 ILD Connector Description

4.2 Loop Oscillator Circuit

In order to detect changes in inductance of a loop, a resonant circuit has been introduced. Detection is carried out by measuring a resonant frequency of the circuit that consists of parallel combination of inductor and capacitor. The capacitor value is fixed, whereas inductance (of the loop) changes as a conductive metal object comes near the loop. Hence, the presence of a vehicle near the loop causes change in the frequency of the oscillator. Later on, this signal is processed by a microcontroller described in Chapter 4.3.2.

Two different oscillator circuits have been examined. It should be noted that **the second circuit** has been employed in the final version of the ILD. Their pros and cons are discussed in the following two subheads.

4.2.1 Circuit based on Comparator

The schematic (based on [19]) is shown in Figure 4.6. The main component is a comparator LM311. Components L1 and C1 creates a parallel LC circuit. The advantage of this circuit is that it can be powered by single power supply only, and that the output is already in TTL logic, which simplifies a connection to the microcontroller.

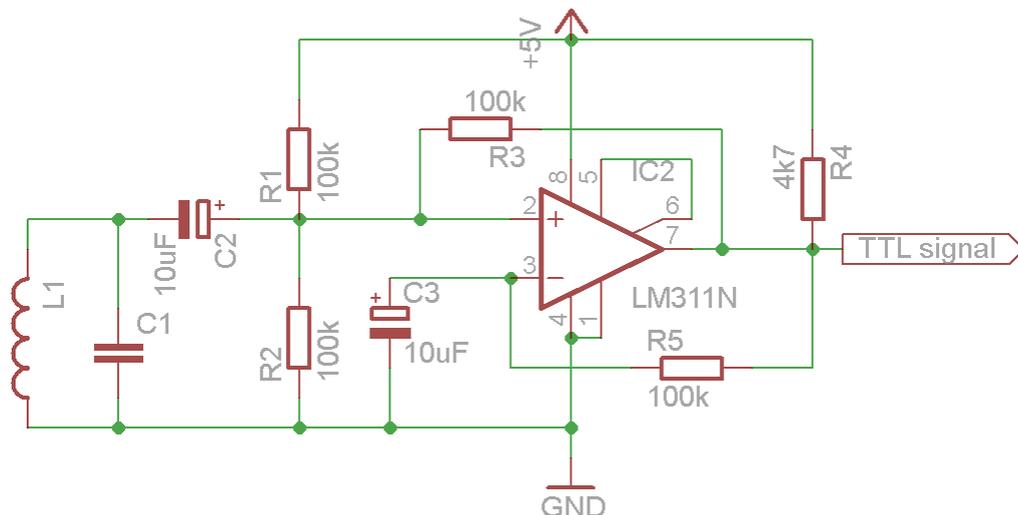


Figure 4.6 Oscillator Circuit with Comparator

This circuit was originally used for measuring small capacitors and inductors. Unfortunately, when a loop (several meters of wire) having **non-zero resistance** is used in this circuit, the output **signal gets noisy**. This causes a small

variation in the output frequency, and therefore, a lower sensitivity of detection (because the signal has to be averaged or filtered), which is a major **drawback** of this circuit.

4.2.2 Circuit based on Operational Amplifier

With regard to the issue mentioned in the previous subchapter, another oscillator circuit has been examined. The circuit in Figure 4.7 employs an operational amplifier, components C1 and L1 forms a parallel LC circuit. Resistor R3 controls the energy fed into the LC circuit.

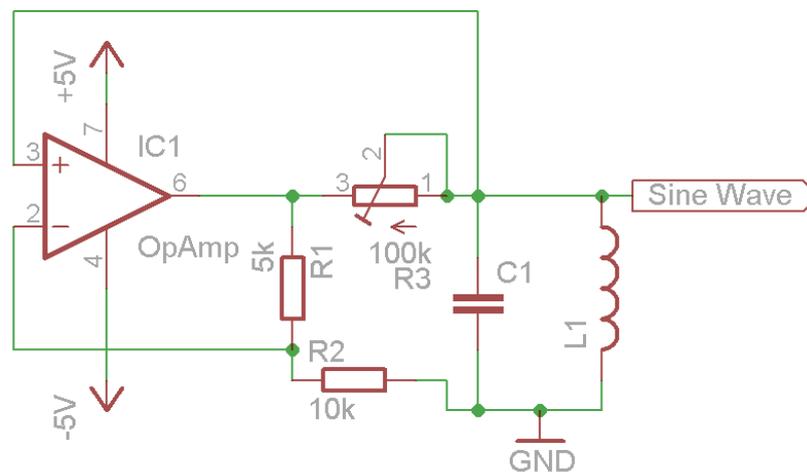


Figure 4.7 Oscillator Circuit with Operational Amplifier

The schematic overcomes the drawback of the previous circuit. However, a disadvantage is the need for dual power supply. In addition, the output signal is a sine wave, which needs further conversion to TTL logic (Figure 4.8). In spite of the fact that more components are necessary to be introduced, **this circuit has been used** in the analog part of the ILD.

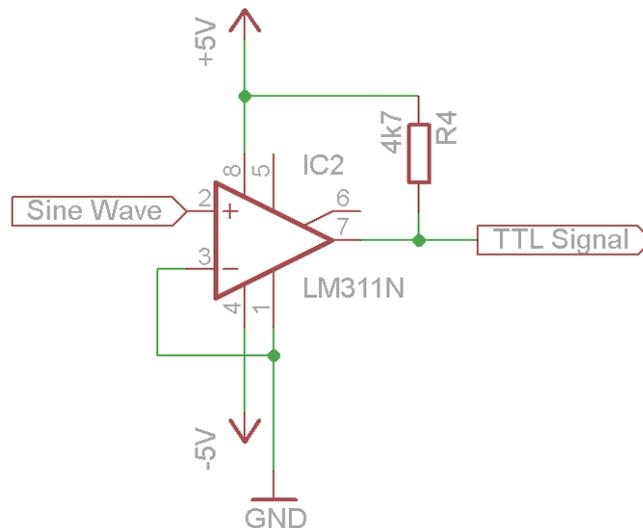


Figure 4.8 Simple Conversion from Sine Wave to TTL Signal

4.2.3 Surge Protection

The inductive loop built within the roadway is connected with the oscillator through twisted-pair cable. Due to outdoor environmental conditions, the circuit needs to be protected against overvoltage caused by electrostatic discharge coming from the loop. The following two types of surge protectors have been used:

- a) Gas Discharge Tube - Figure 4.9a
- b) TVS Diode (Transil) - Figure 4.9b



a)



b)

Figure 4.9 Overvoltage Protectors
a) Gas Discharge Tube, b) TVS Diode

Gas Discharge Tubes are gas-filled components which protect against overvoltage from tens of Volts (typically 100V). They are characterized by low capacitance (approx. 1pF) and high currents (thousands A) they are able to withstand. In spite of that, TVS Diodes can protect the circuit even from several

Volts. Hence, it is a good practice to combine both types of protectors. Figure 4.10 shows the protection used. Normal diodes in bridge improve frequency behaviour of TVS Diodes.

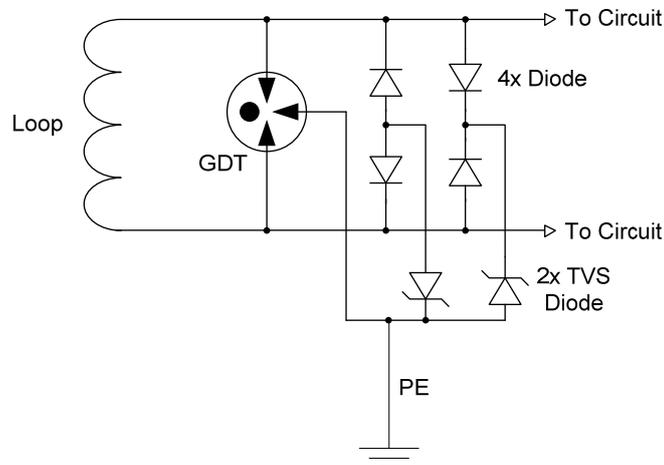


Figure 4.10 Overvoltage Protection

4.3 Microcontroller

Chapter 4.2 described the approach of measuring the inductance of the loop. The sine-wave output of the oscillator has been converted into a TTL signal whose frequency is to be measured and processed by the microcontroller. This chapter describes the microcontroller, its peripherals used, and the firmware implementation.

The operation diagram is depicted in Figure 4.11. After the device is powered on, the microcontroller peripherals such as Timers, Ports are configured. Afterwards, the device waits until there is a signal from loop oscillator circuit, and waits 5 seconds so that it gathers some frequency samples (Chapter 4.3.3), and frequency mean value and deviation (Chapter 4.3.4) are calculated. If the samples deviation is lower than some particular threshold, the device is calibrated and vehicle-detecting mode is set. Otherwise, the deviation implies that there is something happening to the inductive loop, and the device tries to recalibrate itself again. Even during vehicle-detecting mode, due to temperature and other impacts, the device needs to check occasionally whether it requires recalibration.

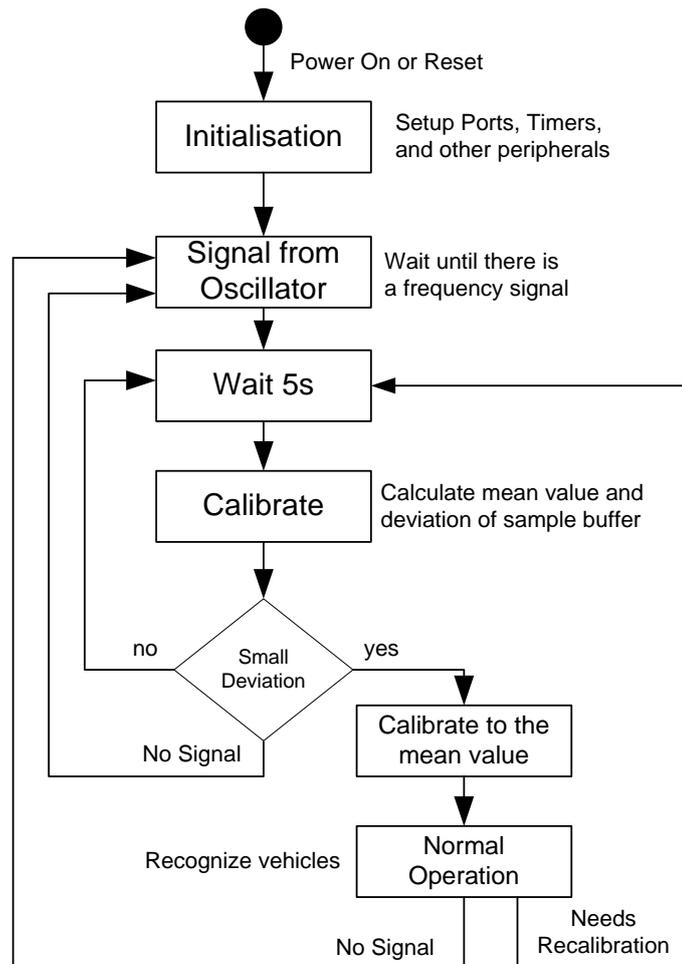


Figure 4.11 ILD Operation Diagram

4.3.1 PIC18F2420

PIC18F2420 is a 28-pin microcontroller which has 16Kb flash program memory and 768 bytes of RAM. Its pinout is shown in Figure 4.12. PORTB (pins 21-28) contains internal pull-up resistors, which can be switched on/off by software. They are convenient for switches – there is no need for external resistors.

There is a 10MHz crystal oscillator connected to the microcontroller. The microcontroller uses its frequency multiplier (H4 fuse in Table 3.1) which yields 40MHz input clock to the microcontroller and 10MHz of effective clock (further details in [15]).

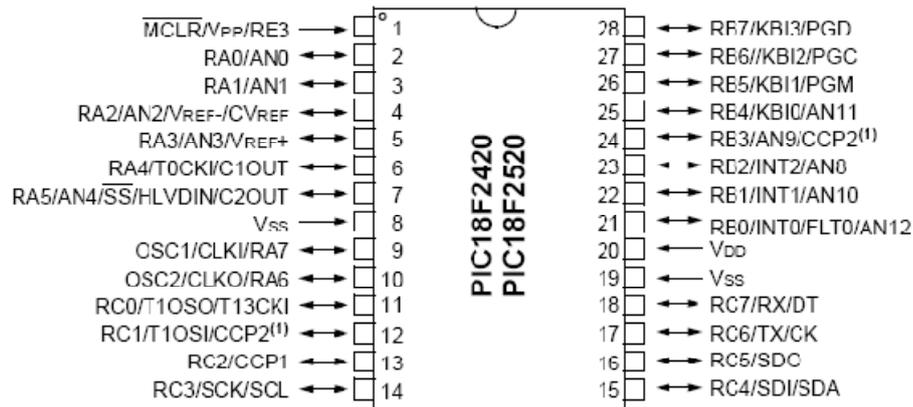


Figure 4.12 PIC18F2420 pinout

4.3.2 Frequency measurement

Capture/Compare/PWM module (CCP) is a PIC on-board peripheral for timing inputs/outputs. It can be configured in Capture mode that allows interrupt-on-change when rising/falling edge appears on the CCP1 input (pin 13). Figure 4.13 depicts the function of CCP module when it is configured in Capture mode.

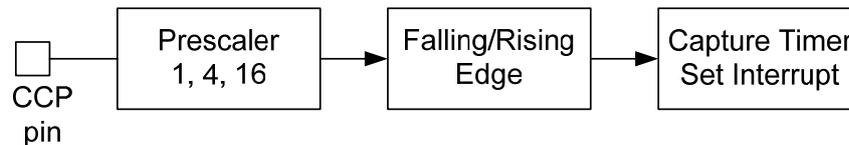


Figure 4.13 CCP module in Capture mode

The following has to be set to configure the CCP:

- Switch to Capture Mode
- Select Timer for capturing (Timer1/Timer3)
- Select occurrence:
Rising edge, Falling edge, every 4th Rising edge, every 16th Rising edge
- Enable CCP interrupt

Given that the frequency of the oscillator is expected between 20 kHz and 145 kHz (Chapter 2.4.2), it is convenient to set the signal prescaler to 16 so that the interrupt routine is not called so often. Furthermore, dividing the signal by 16 also produces **signal filtration**, which is desired.

Therefore, the snippet in C language (for CCS C Compiler – Chapter 3.3) for CCP configuration is the following:

```

setup_ccp1(CP_CAPTURE_DIV_16|CCP_USE_TIMER3);
setup_timer_3(T3_INTERNAL|T3_DIV_BY_1);
enable_interrupts(INT_CCP1);
enable_interrupts(GLOBAL);

```

CCP interrupt routine (high-priority – Chapter 3.3.5.2):

```

#int_CCP1 HIGH
void CCP1_isr()
{
    unsigned int16 CCPtemp,CCPactual;
    static unsigned int16 CCPprev;
    CCPtemp=CCP_1; // get the captured value of Timer3
    CCPactual=CCPtemp-CCPprev; // contains time duration between 16 ticks
of the input signal
    // further processing
    CCPprev=CCPtemp;
}

```

Variable CCPactual contains count of Timer3 ticks between 16 periods of the input signal. Hence, the signal frequency is calculated:

$$f_s = \frac{16 \frac{f_{osc}}{4}}{CCPactual} = \frac{4.40 \cdot 10^6}{CCPactual} \quad [Hz] \quad (4.1)$$

4.3.3 Sample Buffer

In order to collect frequency samples for calibration and for recognizing vehicles, samples buffering has been introduced. For both purposes, the only properties necessary to know are the mean value and deviation of samples gathered. Thus, samples can be saved into a circular buffer, which rolls from its end to the beginning. The following outlines the circular buffer implementation:

```

#define SAMPLES_NO 64
unsigned int16 Samples[SAMPLES_NO];
unsigned int8 SamplePointer=0;

```

Writing into the buffer:

```

Samples[SamplePointer]=CCPactual;
SamplePointer++;
if(SamplePointer>=SAMPLES_NO) SamplePointer=0;

```

There is no need for calculating frequency according to Equation 4.1. All computations can be worked out with counter values, which is much faster because samples do not have to be converted to frequency value.

Hence, *Samples* array is continuously filled with **counter values** gathered according to Chapter 4.3.2, and it contains **the most recent 64** frequency samples.

4.3.4 Calibration

At the start-up and every time an inductive loop is connected, calibration process has to be carried out. If there is a loop connected to the device, the device gathers samples into a calibration circular buffer (see Chapter 4.3.3 for algorithm used) for 5 seconds, and calculates the mean value and the deviation of the samples:

$$\bar{x} = \frac{1}{N} \sum_{i=0}^{N-1} x_i \quad (4.2)$$

$$s = \sqrt{\frac{1}{N-1} \sum_{i=0}^{N-1} (x_i - \bar{x})^2} \quad (4.3)$$

In C language:

```

unsigned int16 GetMean(unsigned int16 arr[],char length)
{
    unsigned int32 Sum=0;
    char i;
    for(i=0;i<length;i++)Sum+=arr[i];
    Sum/=length;
    return (unsigned int16)Sum;
}

unsigned int16 GetDeviation(unsigned int16 arr[],unsigned int16
meanvalue,char length)
{
    unsigned int32 Sum1=0,Sum2=0;
    char i;
    for(i=0;i<length;i++)
    {
        Sum1=(unsigned int16)(abs((signed int32)arr[i]-meanvalue));
        Sum1*=Sum1;
        Sum2+=Sum1;
    }
    return (unsigned int16)((float)(1.0/(length-1.0))*(float)Sum2);
}

```

It shall be noted that there is no square root function at the end of *GetDeviation* function. Again, the deviation value does not have to be calculated accurately, and square root math function takes up plenty of machine cycles to be worked out.

Calibration is performed during the following events:

- Power up/Reset
- A loop is disconnected and connected again
- The mean value changes during vehicle-recognition mode
- Vehicle is above the loop continuously for 5 minutes

It should be noted that the oscillator frequency is **limited to 20kHz to 145kHz**. If frequency is out of range, calibration will not be performed and the device will behave as if there was no signal (see Chapter 4.3.6). Calibration is successful if the value obtained by *GetDeviation* **function is lower than Deviation Threshold** described in Chapter 4.3.7.

4.3.5 Vehicle Recognition

When the device is successfully calibrated, vehicle-recognition mode is set. The device calculates the mean value and the deviation (Chapter 4.3.3) of Sample buffer, and based on the calibrated mean value, it decides whether a vehicle is above the loop or not.

Recognition is implemented as a state machine depicted in Figure 4.14. There are 2 states – No vehicle (*VehOff*), Vehicle (*VehOn*). Transition between them is done according to Figure 4.15.

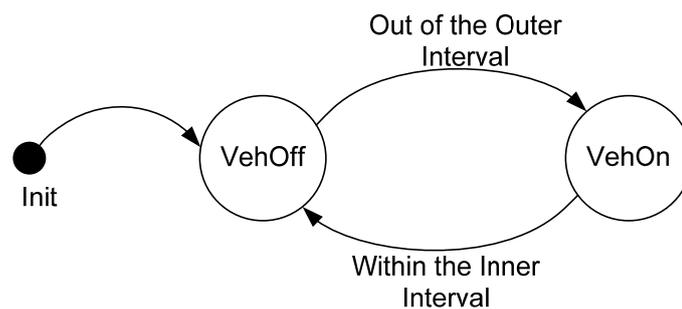


Figure 4.14 Vehicle Recognition State Machine

Actual mean (average) value of frequency is continuously calculated (*GetMean* function) from the sample buffer and compared to the calibrated value. Averaging process provides **filtration of unwanted detections**.

If the actual value gets out of the outer interval (i.e. a metal object is close to the inductive loop), it is considered as vehicle above, and transition to *VehOn* state is made. Similarly, when the actual value returns back to the inner interval, it implies that the vehicle has left the loop, and *VehOff* state is set. In order to provide **setting the sensitivity** of the device, both intervals can be set via on-board switches (Chapter 4.3.7).

The device is automatically recalibrated if the device remains in *VehOn* state longer than 5 minutes.

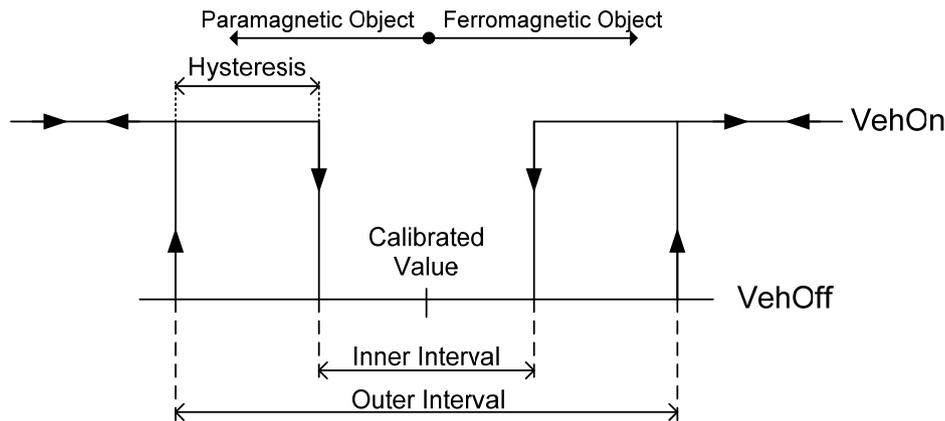


Figure 4.15 State Change Hysteresis

Calibrated value in Figure 4.15 is a value of CCP1 counter obtained according to frequency measurement described in Chapter 4.3.2. When a vehicle enters the loop, the loop inductance decreases (see Chapter 2.4.1) and therefore, oscillator frequency is increased according to Equation 2.1 and 2.3 - the higher frequency, the lower counter value.

It would be sufficient to implement left side of interval in Figure 4.15 only, because a vehicle always causes lower counter value. However, **the purpose of the ILD is to be versatile**, and therefore, to detect ferromagnetic materials (Chapter 2.4.1) as well. Hence, a symmetrical interval, capable of detecting both types of materials, has been introduced. Code snippet in C language of the state machine follows:

```

if(VehicleAbove)
{ // vehicle is above
  if(AverageCounter >= (CalibCounter - CalibCounterOff) &&
AverageCounter <= (CalibCounter + CalibCounterOff))
  { // vehicle is leaving
    VehicleAbove = false;
  }
}
else
{ // no vehicle
  if(AverageCounter <= (CalibCounter - CalibCounterOn) ||
AverageCounter >= (CalibCounter + CalibCounterOn))
  { // vehicle is going above
    VehicleAbove = true;
  }
}
}

```

CalibCounterOn and *CalibCounterOff* are hysteresis offsets, they provide to set the device sensitivity (Chapter 4.3.7). Both can be changed via on-board DIP switches (Chapter 4.3.7).

4.3.6 LED Outputs

There are three LED diodes (position shown in Figure 4.3) on the ILD – Green, Yellow, Red. Each of them has a different meaning also depending on the device calibration state. Their purpose is described in Appendix 5.

4.3.7 Setting Device Parameters

The detector is equipped with 2 blocks of DIP switches (position shown in Figure 4.3) used for configuring the device.

a) Oscillator frequency

DIP switches in Figure 4.3 set the loop oscillator's frequency by adding parallel capacitors to the oscillator LC tank (see C1 and L1 in Figure 4.7). Frequency decreases as capacitance grows (Equation 2.1). There is a fixed **10nF** capacitor. In addition, there are 2 switches for adding **33nF** and **100nF** capacitors to the circuit. It is necessary to keep in mind that the oscillator frequency has a limited range of 20 kHz to 145 kHz in the firmware (Chapter 4.3.2).

When 2 detectors are working close to each other, **crosstalks** may occur. In other words, one detector may influence the other one. Therefore, to **avoid** the crosstalk, operating frequencies have to differ from each other. In order to determine the frequency easily, the device expresses the actual operating frequency by yellow LED diode blinking (see Appendix 5). Loop frequency can be selected by frequency switches (see Figure 4.3 for position). Their combinations and possible inductance values are listed in Table 4.2.

DIP Switch On	Total Capacitance [nF]	Loop Inductance Range [uH]
none	10	120 - 6000
1	43	30 – 1500
2	110	10 - 600
1, 2	143	8 - 450

Table 4.2 Frequency Configuration

b) Detection Sensitivity

The device, apart from previously described frequency setting, also allows to set the sensitivity of detection. In other words, how small metal object can be detected. The calibration deviation threshold (Chapter 4.3.4) also changes according to the sensitivity configuration - The higher the sensitivity, the higher deviation threshold that allows the device to be calibrated.

There are 8 DIP switches, the switches 7, 8 are used only - Table 4.3. The reset of switches are not used and may be used in the future.

DIP Switch On	Device Sensitivity	Calibration Deviation Threshold
None of 7,8	Low	Low
7	Medium Low	Medium
8	Medium High	Medium
7, 8	High	High

Table 4.3 Device Sensitivity Configuration

5 Traffic Counter

Previous chapter deals with recognition whether a vehicle is above an inductive loop. With the help of the Inductive Loop Detector, we can therefore determine the vehicle presence and position.

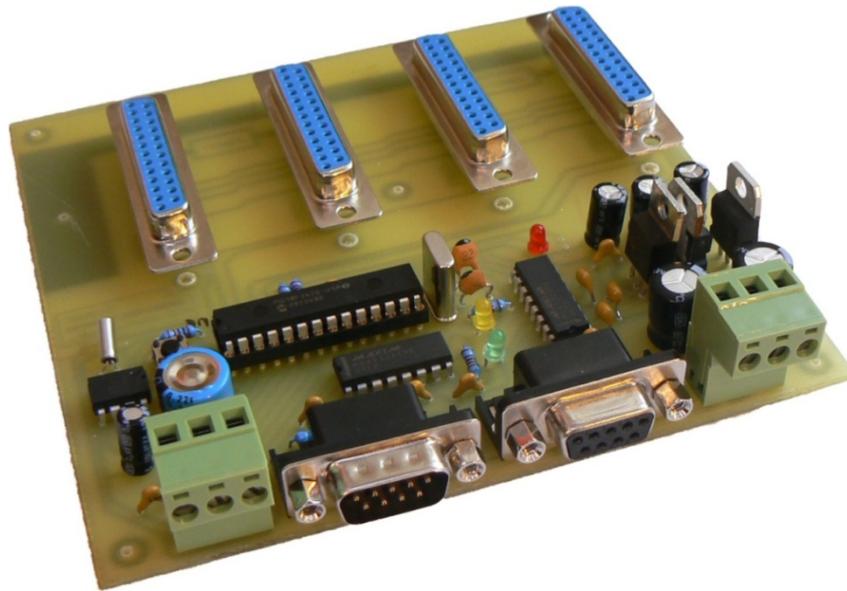


Figure 5.1 Traffic Counter

The Traffic Counter (Figure 5.1) is a device developed for measuring speed and length of vehicles. Maximally two traffic lanes can be monitored, each of them has to be equipped with 2 inductive loops which has to be away from each other some particular distance so that measurement can be carried out (Chapter 2.5).

Figure 5.2 contains an overview of the Traffic Counter. The device incorporates an on-board RTC chip (Chapter 5.2.2) so that date and time is also logged when a vehicle is measured. Furthermore, there is ambient temperature monitoring (Chapter 5.2.3) and both indoor and outdoor temperatures can be logged. The device can be configured via the implemented command-line interface (Chapter 5.4) running on its PC RS-232.

As the last task performed, in case a GPRS modem is connected to the device, vehicle and temperature samples gathered are transferred to the server in particular time intervals.

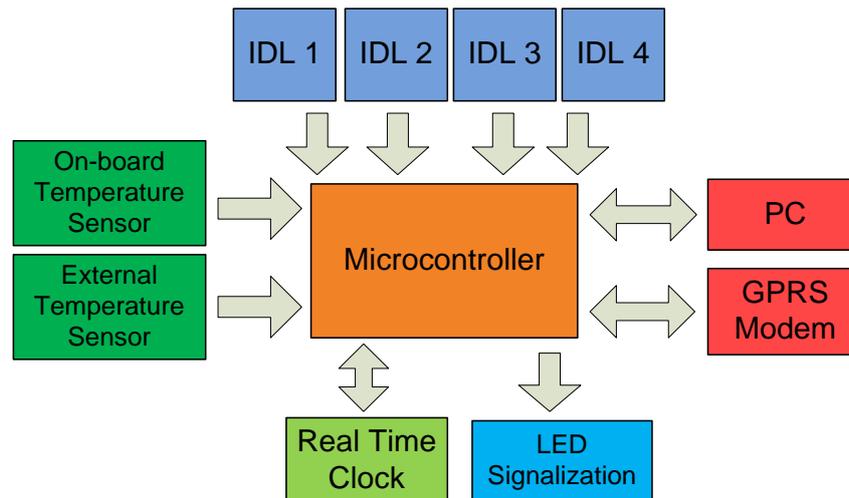


Figure 5.2 Traffic Counter System Overview

5.1 Device Description

The device is based on PIC18F2620, it is exactly the same microcontroller as PIC18F2420 (Chapter 4.3.1) except for ROM and RAM sizes, which are 32kB and 4KB. The microcontroller has been chosen because of large memory since the device has to buffer vehicle samples meanwhile it is not connected to the Internet via a GPRS modem.

The layout of the device is shown in Figure 5.3. The device needs $\pm 15\text{V}$ DC supplied by an external power supply through a power connector. There are 4 connectors for ILDs (Chapter 4), which are used for measuring vehicles.

The device contains two RS-232 connectors for PC and a GPRS modem. Finally, there are 2 temperature sensors, one for board (indoor) temperature, the latter (should be placed outside the box and connected by cable to *external temperature sensor connector*) for outdoor temperature.

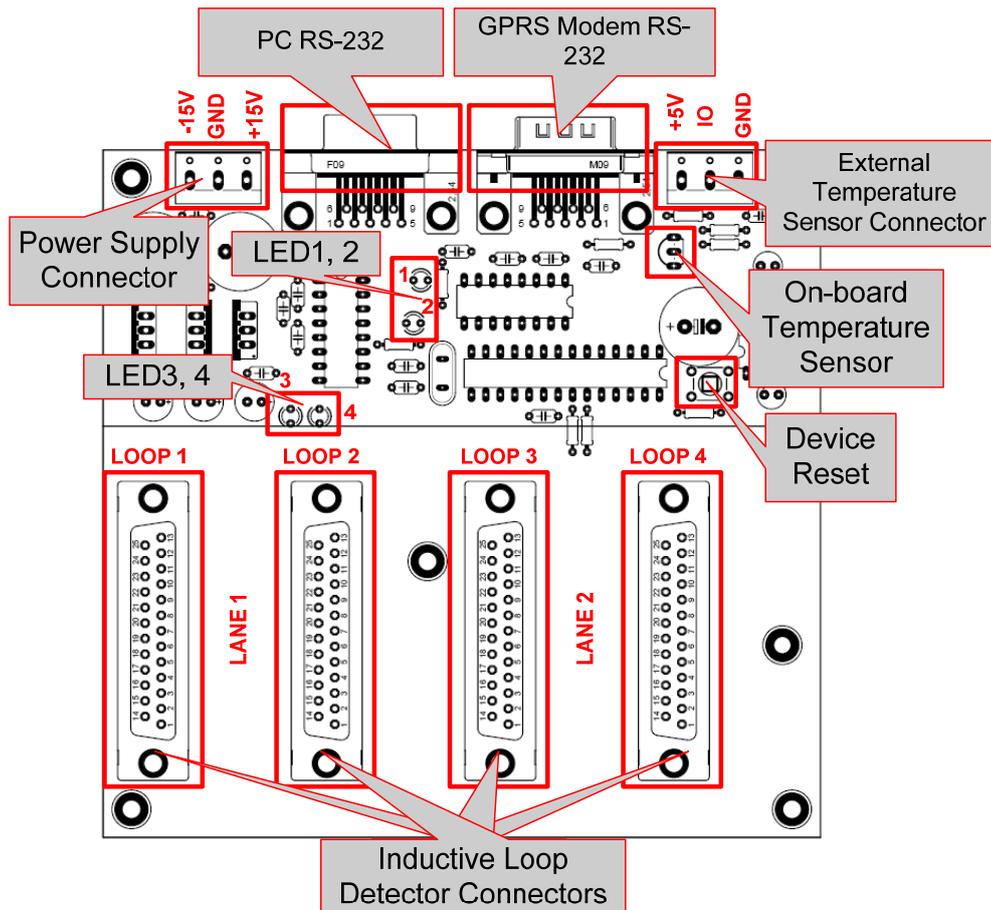


Figure 5.3 Traffic Counter Board Layout

Communication connectors and their pinout is shown in Figure 5.4, a straight cable should be used for connecting the PC and a modem. Both connectors have a standard RS-232 pinout.

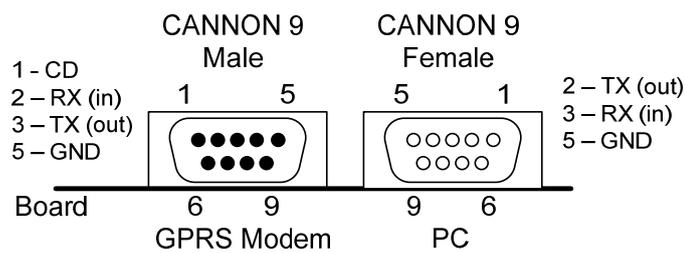


Figure 5.4 PC and GPRS Modem Connectors

The user is notified about the state of the device by 4 on-board **LEDs** whose purpose is described in Appendix 9.

5.2 Firmware Implementation

5.2.1 Vehicle Measurement

The device implements loop signals measurement as described in Chapter 2.5. There are 4 inductive loop detectors connected to the Traffic Counter, each detector provides vehicle TTL signal (VEHTTL) on its pin 13 (see Table 4.1). Those signals are connected to the microcontroller's PORTB (Figure 5.5) that has interrupt-on-change feature on its 4th to 7th bits.

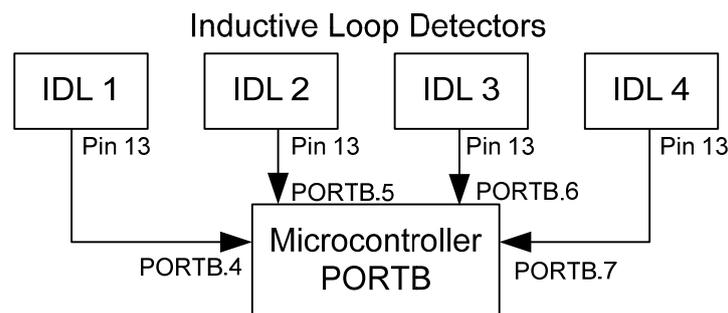


Figure 5.5 Detectors and Microcontroller Connection

The PORTB interrupt is called once rising/falling edge has occurred on pins PORTB.4 to PORTB.7. In order to find out what event has happened, the previous state has to be memorized. The following snippet outlines the PORTB interrupt routine (does not include vehicle measurement algorithm):

```
#int_RB HIGH
void RB_isr(void)
{
    int8 current;
    static int8 last;
    static int8 bitevent;
    current=input_b();
    bitevent=(current ^ last)>>4; // get change
    if((bitevent & 3) > 0)//something with lane0 - event on ild1 or ild2
    {}
    if((bitevent & 12) > 0)//something with lane1 - event on ild3 or ild4
    {}
    last=current;
}
```

Vehicle measurement is carried out via state machine as depicted in Figure 5.6. Measurement starts when a vehicle enters the first loop. A state transition is made according to Figure 2.4, and in case of any unexpected state of the detector inputs, transition to the idle state is made.

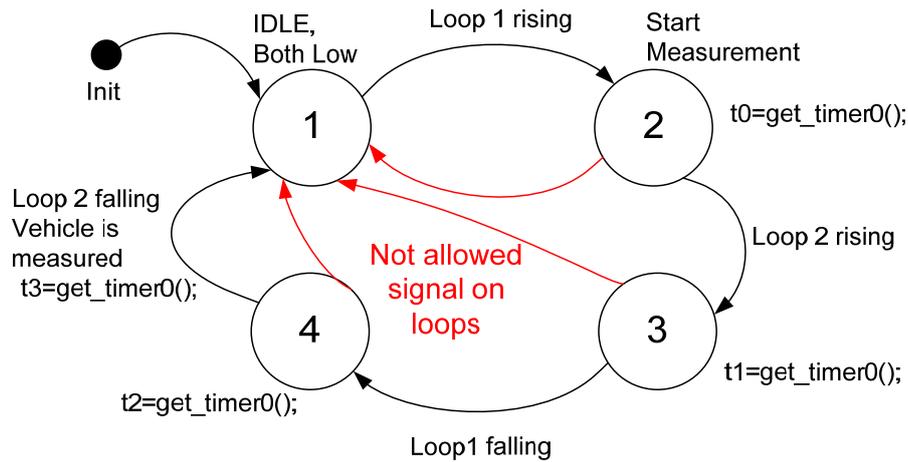


Figure 5.6 Vehicle Measurement State Machine

Time measurement is performed via **timer0** which is a 16-bit timer. After every transition, timer0 ticks are saved into t0-t3 variables. For the purpose of vehicle measurement, a slower timer is better. The highest prescaler value of the timer is **256**, thus, the timer has the following properties (with regard to 20Mhz crystal clock of the device):

- Timer tick 51,2us
- Overflow every 3355ms

Given that 3,3 seconds is not enough to measure long vehicles, the number of timer overflows (*of*) is counted and its value is added to length calculations:

$$v_1 = \frac{3600000(\text{loopdistance} + \text{looplength})}{512(t_1 - t_0)} \quad [\text{kmph}, \text{dm}, \text{dm}, \text{ticks}, \text{ticks}] \quad (5.1)$$

$$l_1 = \frac{512 \cdot v_1 (t_2 - t_0 + \text{of} * 65536)}{360000} - 10 \cdot \text{looplength} \quad (5.2)$$

[cm, kmph, ticks, ticks, dm]

Similar calculations are done with t2, t3 values (to calculate v_2 , l_2) if measurement averaging is set by the user (refer to *MEASUREAVG* in Chapter 5.3 and Appendix 10).

Vehicle samples (including timestamp described in Chapter 5.2.2) are saved into RAM buffer as discussed in Chapter 5.2.4 to RAM storage.

5.2.2 Date and Time

It is essential that the device saves a timestamp along with vehicle's speed and length. For that purpose, it was mandatory to employ a RTC peripheral to establish a time base.

Maxim DS1302 [12] is a RTC chip in PDIP8 package with a simple 3-wire TTL interface. It provides time keeping calendar functions up to the year 2099. A basic circuit is depicted in Figure 5.7. In order to keep time base even if power supply is not connected, an external back-up capacitor can be attached on pin 8. The chip contains internal diode and resistors charger (*Trickle Charger* in the device data sheet) for the capacitor so it is charged when the main power supply is available. When a capacitor is fully charged, the time keeping **will be functional up to several days**.

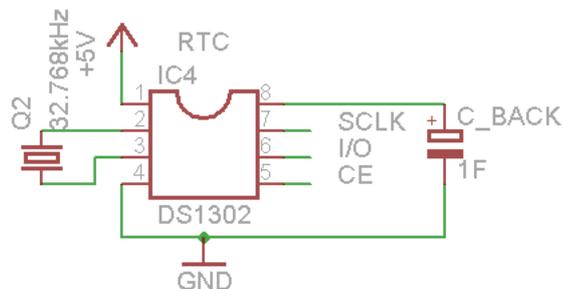


Figure 5.7 Maxim DS1302 Basic Circuit

The CCS C Compiler (Chapter 3.3) comes with a DS1302 driver (*DS1302.c*) and standard C language library *time.h*. Therefore, the following only describes the functions of the driver. Further information on the 3-wire interface can be found in the device data sheet.

Function	Description
<code>rtc_init()</code>	Initialize the chip
<code>rtc_set_datetime()</code>	Write date and time to the chip
<code>rtc_get_date()</code>	Read date
<code>rtc_get_time()</code>	Read time

Table 5.1 DS1302 Driver API

It should be noted that the chip does not allow to set seconds. Hence, when calling `rtc_set_datetime` function, it always sets seconds to 0.

For purpose of saving vehicle samples into buffer, the sample size matters because there is a limited RAM size of the microcontroller. Therefore, it is more efficient to save a Unix timestamp having 4 bytes rather than saving date and time separately. The compiler's *time.h* contains *mktime* function which accepts *struct_tm*, and which returns a timestamp (i.e. number of seconds since January 1, 1970 00:00:00 UTC). All date and time functions of the compiler time library have years since 1900 – for instance, number 108 means the year 2008 etc. Months and days begin at 0, not 1. The following demonstrates how to get a timestamp:

```
struct_tm ActualTime;
unsigned int32 timestamp;
ActualTime.tm_year=year+100;
ActualTime.tm_mon=month-1;
ActualTime.tm_mday=day-1;
ActualTime.tm_hour=hour;
ActualTime.tm_min=min;
ActualTime.tm_sec=sec;
timestamp=mktime(&ActualTime);
```

5.2.3 Measuring Temperature

As well as gathering vehicle samples, the Traffic Counter provides taking indoor and outdoor temperature samples by two Maxim DS18B20 [13] digital temperature sensors. One is placed on the board, the latter should be connected by 3 wires to the external sensor connector (shown in Figure 5.3), and placed outside the case. The period of taking temperature samples can be set by *TEMPTIME* attribute (Table 5.3) via the command-line interface described in Chapter 5.4.

DS18B20 is a 3-pin variable precision temperature sensor in TO-92 package, it communicates via Maxim 1-Wire interface, which is a master-slave, half-duplex protocol on a single wire. Figure 5.8 contains DS18B20 pinout and basic circuit. There has to be a pull-up resistor on the data lane for proper work of the 1-Wire. The description of the interface is out of scope of this thesis since the CCS C Compiler contains a driver for the sensor.

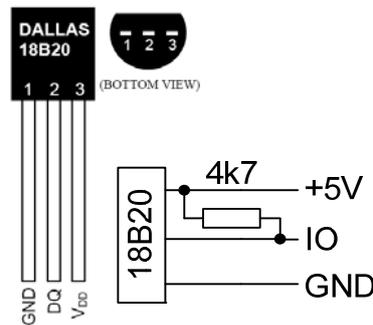


Figure 5.8 Maxim DS18B20 Pinout and Circuit

Several 1-Wire devices can be placed on a single 1-Wire bus. However, for ease of employing 2 temperature sensors, the both are placed on separated 1-Wire buses.

The device driver has been modified so that it handles two different buses and also allows for having no sensor connected (in case of the external sensor). The following describes the modified driver API:

ds1820_init() – initiates the on-board sensor

ds1820_init2() – initiates the external sensor

ds1820_read(int1 sensor) – returns sensor temperature in degree Celsius,
 sensor=0 -> on-board, =1 -> external
 if sensor is not connected, returns -128

5.2.4 Vehicle and Temperature Samples Buffer

Both types of samples need to be saved into RAM memory, where they are uploaded from later on (see GPRS modem in Chapter 5.5). So that the sample size is as small as possible, some reductions have been made.

Given that number of lanes within a station will surely never be higher than 16, we need just **4 bits** for that value. Moreover, length of a vehicle will certainly be lower than 40 meters, thus, only **12 bits** are required to hold the length value in centimetres (12 bits ~ 4096 cm). Those two variables can be assembled into one 16-bit variable.

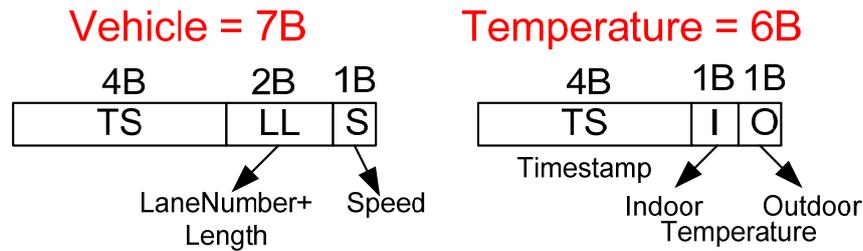


Figure 5.9 Samples Memory Layout

Structure of a vehicle (7 bytes) and a temperature (6 bytes) sample is shown in Figure 5.9. The CCS C Compiler uses **Little-endian byte ordering** (the least significant bit/byte is stored at the memory with the lowest address). In order to provide access to single bytes of each sample, the entire sample is encapsulated in a *union*, which provides access to different variables at the same location in RAM memory (see Chapter 5.5.7 for usage demonstration):

```
typedef union {
    int8 bytes[7];
    struct {
        unsigned int32 DateTime;
        unsigned int16 LaneLength; // lower 12 bits = length in cm, higher 4
        bits=lane
        unsigned int8 Speed;
    } vehicle;
} Vehicle;
Vehicle VehicleBuffer[VEHICLE_BUFFER_LENGTH];
```

Finally, variable *VehicleBuffer* is the main buffer, where vehicle samples are saved into, and from where the data is read for server upload. Temperature buffer is implemented in the similar manner. Table 5.2 summarizes some combinations of vehicle and temperature buffer sizes with regard to the available RAM memory space, the combination in bold, which provides a reasonable ratio between both types of samples, has been used.

Number of Vehicle Samples	Number of Temperature Samples	Total RAM Usage [%]
430	30	95
400	70	94
350	100	90

Table 5.2 Memory Space and Buffer Sizes

5.3 Device Settings and Parameters

In order to save settings such as inductive loop dimensions, server's ip address, station id, it is necessary to save the configuration into a memory which keeps its content even without power supply. Fortunately, the microcontroller contains an internal 1kB EEPROM memory. Device configuration (properties listed in Table 5.3) can be changed via PC serial interface (Chapter 5.4).

It should be noted that configuration is load from EEPROM memory after power-up. If the **user wants to save the configuration** to EEPROM, the *WRITE* command (refer to Appendix 10) has to be issued.

Item	Value Type	Description
AutoStart	bool	Start measurement after power-up
SID	uint16	Station ID
MeasureAvg	bool	Measure Average (see averaging in Chapter 2.5.2)
LoopDist	uint8	Distance between loops [dm]
LoopLen	uint8	Length of a loop [dm]
LaneNum	uint8	LaneNumber (see Chapter 6.2)
GprsSend	bool	Send samples to the server
GprsTime	uint8	Send period [minutes]
GprsPin	uint16	PIN code of the modem SIM card (0 for none)
ServerIP	uint32	IP address of the server (xxx.xxx.xxx.xxx)
ServerPort	uint16	Server port
TempTime	uint8	Time interval of taking temperature samples [minutes]

Table 5.3 Device Parameters

There is a data structure in the program which can be written/read into the EEPROM memory, example:

```
typedef struct _Config{
    int1 AUTOSTART;
    int1 MEASUREAVG;
    int1 GPRSEND;
    unsigned int8 GPRSTIME;
}Config;
Config Configuration;
```

The following two functions implements reading and writing from the internal EEPROM memory, whole data structure at particular offset can be written:

```
void EEPROM_READ(int *ptr,int num,int addr)
{
    int count;
    for (count=0;count<num;count++)
    {
        ptr[count]=READ_EEPROM(addr+count);
    }
}
void EEPROM_WRITE(int *ptr,int num,int addr)
{
    int count;
    for (count=0;count<num;count++)
    {
        WRITE_EEPROM(addr+count,ptr[count]);
    }
}
```

To save the configuration structure, the following command is used:

```
EEPROM_READ(&Configuration,sizeof(Configuration),0);
```

5.4 Command-line Interface

In order to configure the device (see Chapter 5.3), a simple command-line interface has been implemented. It receives characters from PC serial line, buffers them, and when a delimiter character is sent, the entire buffer is processed. **The list of available commands** can be found in Appendix 10.

5.4.1 Character Buffer

Each character received through the device HW UART causes a *RDA* interrupt routine to be called. The character is saved into a receive buffer. If a '\r' character is received, the content of the buffer is copied (so that serial line receive works when the buffer is processed) and is processed in the program main loop (see Chapter 5.4.2). The following function receives a character when it is ready, saves it into the buffer, and sets *process* flag.

```
inline void PCBufferIsr()
{
    char c;
    c=fgetc(PC);
    if(c=='\n')
    {
        if(PCBufferPointer>0 && PCBuffer[PCBufferPointer-1]=='\r')
        PCBufferPointer--;
        PCBuffer[PCBufferPointer]='\0';
        strcpy(PCBufferCopy,PCBuffer);
        PCBufferPointer=0;
        PCBufferProcess=true;
    }
}
```

```

}
else
{
PCBuffer[PCBufferPointer]=c;
if(PCBufferPointer<PC_BUFFER_LENGTH-1)
{
PCBufferPointer++;
}
}
}

```

5.4.2 Buffer Processing

There are 2 types of command – with or without value. Character buffer contains entire string received over the serial line. Figure 5.10 depicts the processing algorithm. At first, it is verified that the command exists. If the command cannot contain a value, a response to the command is generated.

Otherwise, if the command received contains a value, it means that the users is going to change device configuration. Hence, the value is parsed and saved.

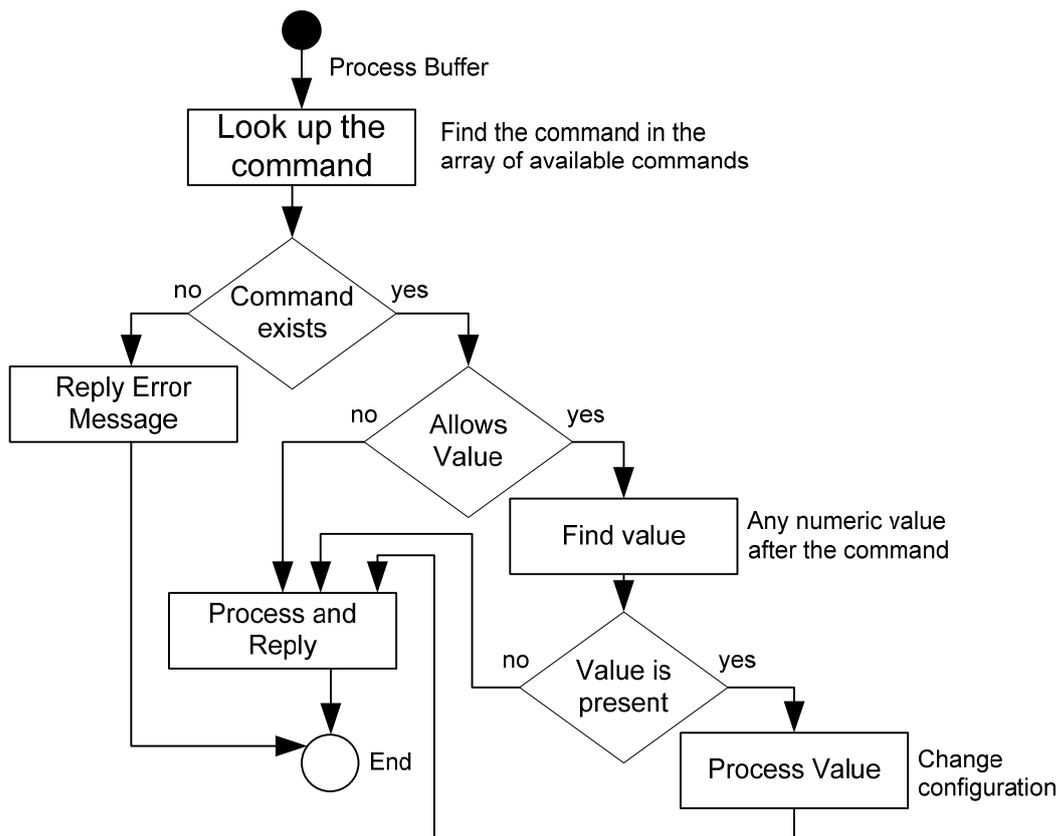


Figure 5.10 Buffer Processing Algorithm

The value can be various depending on the command. It is mostly a numeric value – bool, uint8, uint16. However, in some cases, it can also be a datetime or an ip address.

5.4.3 Usage

The serial line at the PC side shall be set according to the following:

- **115200 baud, 8 bits, no parity, 1 stop bit, no handshake**

The list of all available commands can be found in Appendix 10. All commands have to end with `\r` or `\r\n` characters. Otherwise, the device will not reply. If the received command exists, the device replies with the command, otherwise an “*unknown command*” string is sent back.

In case of commands with value, if the user sends the command without any value, it is considered as reading the command value and the device replies with the value. If the user sends a command+space followed by a value, this value is parsed and saved into an appropriate configuration property.

Some examples of commands follow:

- Get actual temperature: *GETTEMP\r*
- Get the device date and time: *DT\r*
- Set the device date and time (2008-12-12 15:43): *DT 0812121543\r*
- Set the server IP address: *SERVERIP 192.168.1.1\r*

5.5 GPRS Modem

A GPRS modem has been introduced for transferring data to the server (see Traffic Server in Chapter 6.3). The modem is connected with the Traffic Counter via Modem connector depicted in Figure 5.3.

5.5.1 Description

TENcom SPEEDER RS [24] is shown in Figure 5.11. The modem communicates via a standard RS-232 interface.



Figure 5.11 TENcom SPEEDER RS GPRS Modem

Modem highlights are listed in Table 5.4. The modem can be run at different baud rates, and SW or HW handshaking can be used. The modem provides embedded UDP and TCP/IP stack, which makes all network operations transparent to the user.

Property	Value
Power Supply	7-30V DC
RF	900/1800/1900MHz
GSM	Voice: FR, EFR, HR SMS: Text, PDU, MO/MT
GPRS	Mode: Class B Multislot Class 12 (4Rx, 4Tx, Max 5 Slots) Speed: 85,6kb/s Tx and Rx
Interface	RS-232, standard AT command set Speed: 9600 – 115200 baud selectable SW or HW handshake UDP, TCP/IP stack, PPP, PAD, CMUX

Table 5.4 Speeder RS Specification

5.5.2 Communication

The modem is a DCE with a Cannon 9 female connector. There is a male connector on the Traffic Counter board so that it is a DTE. Connection between the devices should be via a 1:1 RS-232 cable. The communication properties have been chosen as the following (modem factory defaults in brackets):

- **19200 (115200) baud, 8 bits, no parity, 1 stop bit, no handshake (RTS/CTS)**

Even though the modem provides HW handshaking via RTS/CTS signals, no handshaking has been used. However, the modem sets the *CD* (Carrier Detect – pin 1) signal once a GPRS connection has been established.

5.5.3 AT Commands

The modem communicates via a standard modem AT command set. Each command starts with “*AT*” prefix and is processed by the device when `\r\n` (CRLF) string is received. The response from the modem also ends with `\r\n`. Modem configuration needs to be saved into its EEPROM memory (*AT&W* command). Table 5.5 contains a list of selected AT commands, full command set can be found in [17].

Command	Answer(s)	Description
AT	OK	Find out whet
AT&V	List settings	Print actual settings
AT&W	OK	Write settings to memory
AT+IPR= <i>val</i>	OK	Set baud rate, example: <i>AT+IPR=19200</i>
AT+IFC= <i>x,y</i>	OK	Set handshaking mode, ex.: <i>AT+IFC=0,0</i>
AT+CPIN=" <i>val</i> "	OK ERROR	Sets PIN code for SIM card
AT\$PADDST=" <i>ip</i> ", port	OK	Set Server IP address and Port, ex.: <i>AT\$PADDST="192.168.1.1",12345</i>
ATD*99#	CONNECT ERROR NO CARRIER	Open GPRS connection
+++		Escape sequence, switch from data to AT command mode
ATH	OK	Close GPRS connection

Table 5.5 List of Important AT Commands

5.5.4 Modem Configuration

Before using the modem, it needs to be configured for active TCP connections. The following describes configuring the modem for TCP mode, for UDP and PPP modes, the reader should refer to [16].

1. Baud rate and handshaking

```
AT+IPR=19200\r\n
AT+IFC=0,0\r\n
```

2. Network access point

```
AT+CGDCONT=1,"IP","internet"\r\n
```

3. Active TCP connection

```
AT$HOSTIF=2\r\n
AT$ACTIVE=1\r\n
```

4. Target IP and port (example)

```
AT$PADDST="192.168.1.1",1000\r\n
```

5. Write settings

```
AT&W\r\n
```

5.5.5 GPRS Connection to Server

Samples are transferred to the server (if *GprsSend* property is true) in intervals given by *GprsTime* property (Table 5.3). At first, *GprsPin*, *ServerIp* and *ServerPort* properties have to be configured.

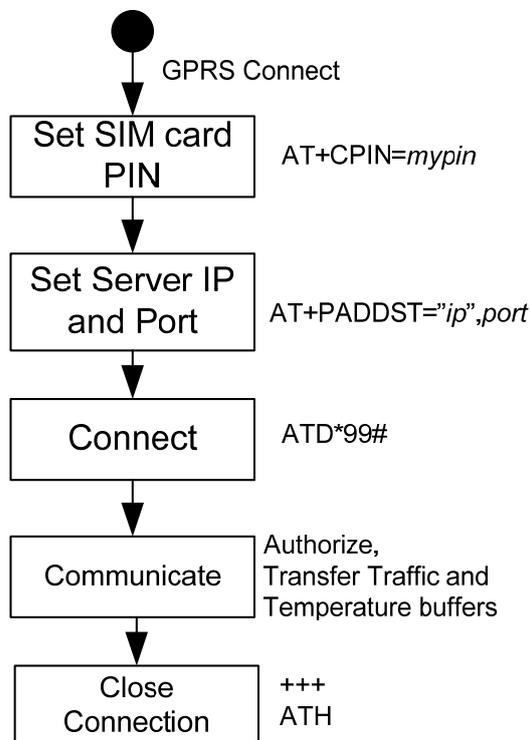


Figure 5.12 GPRS Connection Process

Figure 5.12 outlines how a GPRS connection is established, used, and released.

It is implemented as a state machine. In every state, there is a timeout for a reply from the modem. If the modem replies *OK*, another state is set. In case of error, the gprs connection is closed and entire process starts over.

Once a PIN code is set properly, the modem connects to GSM network, and replies *OK*. After the *ATD*99#* command is issued, a GPRS connection to the server is about to be established. This may take several up to several tens of seconds. The modem sends *CONNECT* in case it has been successfully connected to the server. If a connection is closed by the server or GSM network,

connected to the server. If a connection is closed by the server or GSM network,

the modem sends *NO CARRIER* message. If a connection cannot be established, the modem replies *ERROR*.

5.5.6 Server Protocol

Communication protocol is described in Figure 5.13. When a connection to the server application (Traffic Server in Chapter 6.3) has been established according to process described in Chapter 5.5.5, the client (e.g. Traffic Counter station) sends headers to be recognized and authorized by the server. Headers are sent in text mode (*id=value*) and are separated by semicolon. There are 8 mandatory headers (Table 5.6). If they are not sent, the client is disconnected.

Header	Description
SID	Station ID
PVER	Protocol Version
FWVER	Firmware Version
HWVER	Hardware Version
DTFROM	Acquisition Start - unix timestamp
DTTO	Acquisition End - unix timestamp
SV	Number of vehicle samples to be sent
ST	Number of temperature samples to be sent

Table 5.6 Client Authorization Headers

Hence, the header string look similarly to the following:

```
SID=1;PVER=10;FWVER=9;HWVER=10;DTFROM=1228150807;DTTO=1228151807;SV=2;ST=2
\r\n
```

Afterwards, in case of a proper authorization, a binary mode is set and vehicle and temperature samples are transferred (see Chapter 5.5.7). Every time a reply from a second side is expected, there is a timeout so that connection is closed when an error occurs. If server sends *ERROR* or *BYE*, connection is also terminated. All text messages are delimited by `\r\n`.

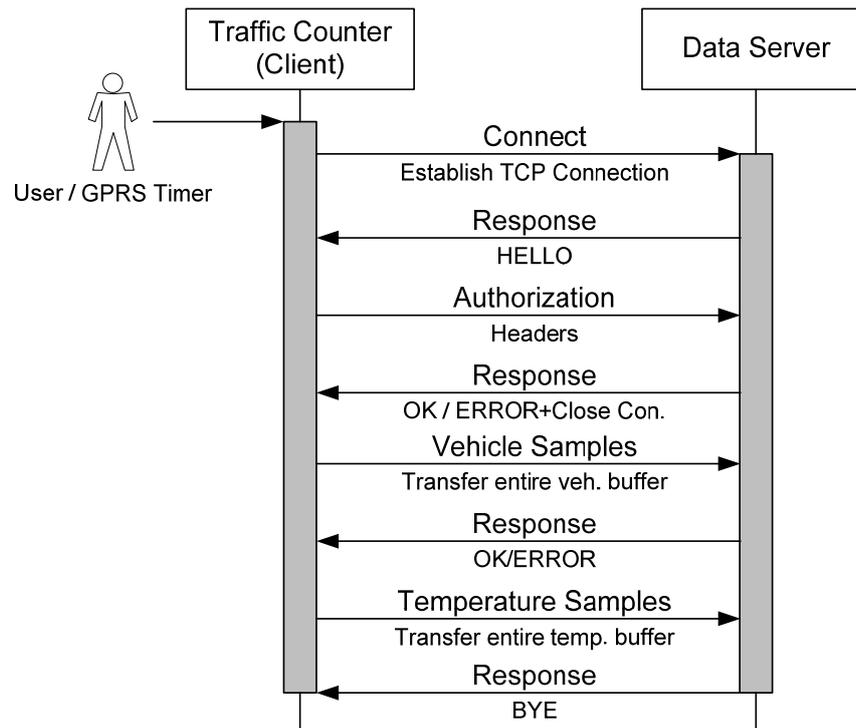


Figure 5.13 Client-Server Communication Diagram

5.5.7 Transferring Samples

Vehicle and temperature buffers are implemented and filled as described in Chapter 5.2.4. Once a connection to the server has been established and the station has been authorized by the server (see headers in Chapter 5.5.6), both buffers are transferred to the server in binary form.

Both sample arrays have items encapsulated within *union* structure, thus, all data can be accessed via *bytes* array. The following outlines the function for transferring data in binary form. Every byte of the vehicle sample is sent to the GPRS modem. Temperature buffer is handled in a similar way:

```

void BinarySendVehicle(unsigned int16 index)
{
    char i;
    for(i=0;i<7;i++)fputc(VehicleBuffer[index].bytes[i],GPRS);
}
  
```

6 Server Applications

Previous chapters deal with traffic data acquisition and uploading to the server in binary form over a TCP connection. Thus, there has to be a mainframe application (Traffic Server – Chapter 6.3) for receiving and processing at the server side. The application communicates with Traffic Counter station over the Internet, places the received data into files, and uploads them to the database as well.

The latest part is a web-interface (Traffic Statistics - Chapter 6.4) for viewing a state of stations and their traffic information. Both applications/scripts are written in PHP language [30] so that the entire server side can be run on various operating systems such as Microsoft Windows or Linux.

6.1 Server Installation

In order to get server applications working, the following packages have to be installed and configured (both tasks are out of scope of this thesis and can be found on the Internet):

- Apache Web Server [29] (version 2 and higher)
- PHP Preprocessor Language [30] (version 5 and higher)
- MySQL Database Server [31] (version 5 and higher)

Moreover, for proper function of the server application, it is also mandatory to install the following modules:

- Mod_rewrite for Apache
- GD library plugin for PHP
- JpGraph library for PHP [35]
- jQuery Javascript Library [22], [33]

6.2 Database Model

The database structure has been designed keeping in mind the purpose of traffic data and its later processing. Table attributes are mainly used within the web-interface described in Chapter 6.4. Database table layout (designed in [32]) is shown in Figure 6.1, SQL create script can be found in Appendix 11. The model contains the following tables:

- **stations**: contains nodes (i.e. Traffic Counter – Chapter 5), each node uploads data over the internet.
- **temperatures**: indoor and outdoor temperature samples of stations
- **uploadedfiles**: every time a station uploads a file, the file details are inserted so that the user can check how stations upload data
- **lanes**: each station consists of one or more traffic lanes.
- **trafficdata**: raw vehicle samples – speed and length of vehicles for every lane

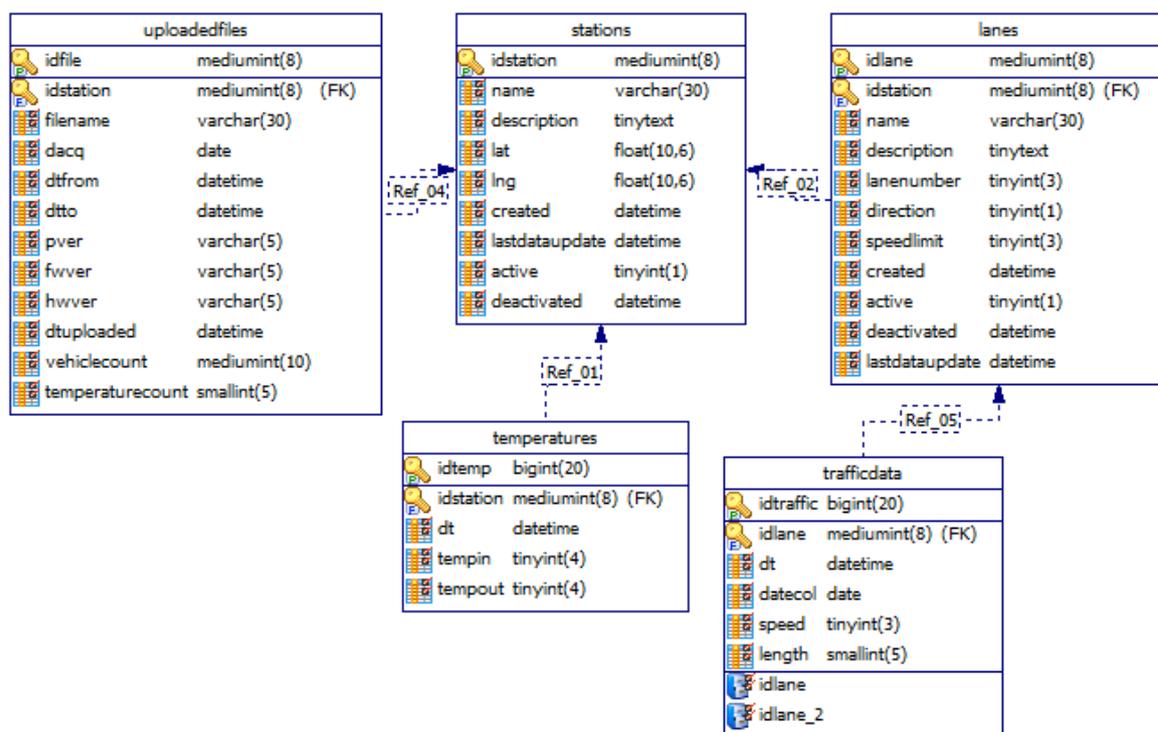


Figure 6.1 Database Model

The following columns are important since they have to be set properly in the Traffic Counter device at a station (see Chapter 5):

- **stations.idstation** – an unique id of a station (*SID* parameter in Appendix 10)
- **lanes.lanenumber** – id of lane within a station (*LANENUM* parameter in Appendix 10)

Even though *lanes.idlane* is a unique identifier, it is easier to use *lanes.lanenumber* which is only unique per station. The reason is that *lanenumber* is

an offset which can be modified when adding more lanes to the station. See property *LANENUM* in Chapter 5.3 for station configuration of *lanenumber* item.

6.3 Traffic Server

The Traffic Server is a PHP script which receives data from station over network, puts it into files and also upload to the MySQL database (having the table structure according to the model in Figure 6.1). It implements multi-client TCP server through socket connections.

Figure 6.2 demonstrates the operation of the server. At first, a socket server is established, and then it waits for a new client or data from any currently connected client. Communication and protocol are described in Chapter 5.5.6.

When a new client is connected, authorization in terms of receiving headers in Text mode has to be made. Once the headers have been successfully acquired from the client, the server awaits vehicle and temperature data in binary form. In case of any error in communication, the client is disconnected. Once all vehicle or all temperature samples have been received, they are converted to text form, saved into text files, and uploaded to the database.

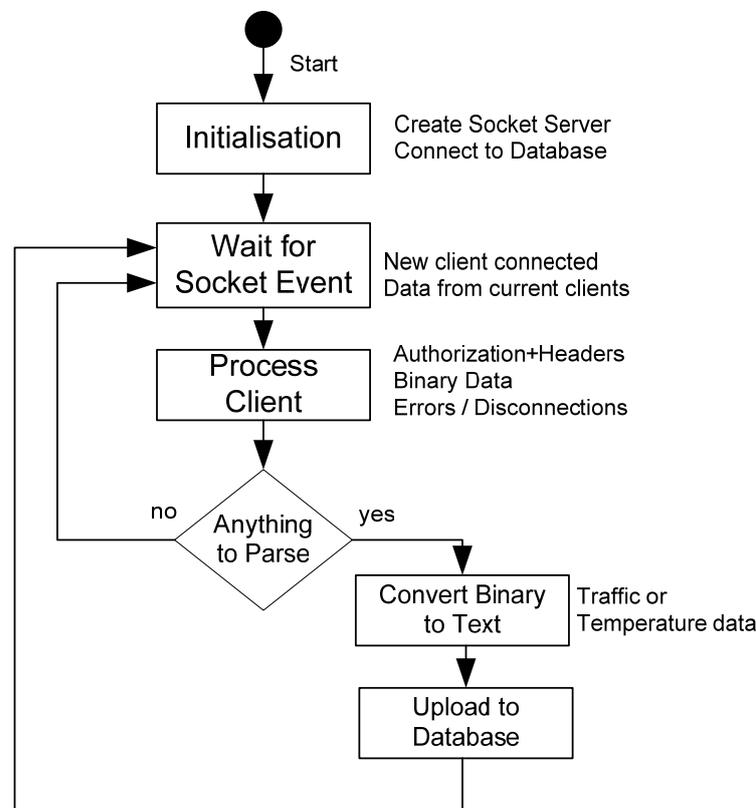


Figure 6.2 Traffic Server Operation

6.3.1 Sockets in PHP

Sockets in PHP are similar to C language sockets. The following PHP code snippet sets up a TCP socket for listening on port 10000:

```
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
if (!is_resource($socket)) {
    echo 'Unable to create socket: '.
    socket_strerror(socket_last_error());
    exit(1);
}
// try to re-use address and port
if (!socket_set_option($socket, SOL_SOCKET, SO_REUSEADDR, 1))
{ echo 'Unable to set option on socket: '.
socket_strerror(socket_last_error());
exit(1);
}
if (!socket_bind($socket, 0, 10000)) {
    echo 'Unable to bind socket: '. socket_strerror(socket_last_error());
    exit(1);
}
socket_listen($socket, 5);
```

Unfortunately, PHP language does not provide threads, but it supports both non-blocking and blocking sockets. Given that the data transfer from a station takes only several seconds and is carried out in a period of several minutes, blocking socket can be used without any problems.

Connection or client disconnection can be recognized by reading from the socket where the event occurred:

```
$result = socket_read($clients[$i]['socket'],7,PHP_BINARY_READ);
if($result === FALSE || $result==' ' ||
(socket_last_error($clients[$i]['socket']) == 104)) {}
```

6.3.2 Handling Multiple Clients

The server application provides the possibility of having several clients (stations) connected at a time. Even though blocking sockets have been used, a simple principle for dealing with multiple clients has been employed.

The script waits until there is either a new client connected or data from one of the current clients is available. The following outlines the main loop of the data server script:

```
while(1){
    $read[0] = $socket; // copy the main socket to detect a new client
    for($i=1; $i<count($clients)+1; ++$i) {
        if($clients[$i] != NULL)
            { // copy sockets of all current clients
                $read[$i+1] = $clients[$i]['socket'];
            }
        $ready = socket_select($read, $write = NULL, $except = NULL, $tv_sec =
        NULL); // blocks
```

```

if(in_array($socket, $read)) { // a new client connected
}
for($i=1; $i<$max_clients+1; ++$i){
    if(in_array($clients[$i]['socket'], $read)){ // process data from
current clients
    }
}
}}

```

The *\$read* array contains all client sockets including the server socket. Function *socket_select* blocks until at least one event occurs on *\$read* array, and removes those sockets from *\$read* array which do not have any event. Function parameters have to be passed by reference – the reason for *\$write*, *\$except*, *\$tv_sec* variables.

6.3.3 File Formats

Once all vehicle samples have been acquired from the station, the file containing those samples is saved into *BINARY_FILES_PATH* directory. Afterwards, the binary file is processed and uploaded to the database. In case of a successful upload, the file is moved to *BINARY_FILES_PATH_PARSED* directory, and is also converted from binary to text form and saved into *DATA_FILES_PATH*.

Filename is always the following: *SID-DATETIME.EXT*, where

SID = Station ID, DATETIME = MySQL timestamp, EXT=bin/txt

Example of traffic file: *1-2008-12-04_22_21_41.bin*

Example of temperature file: *1-2008-12-04_22_21_41_T.bin*

a) Binary File

The structure of a binary file is depicted in Figure 6.3. The file contains *Prefix* “TRAFDATA” in ASCII form so that the file can be distinguished from other files. *Number of Headers (N)* is a count of all consecutive headers. After that, there is *N* of header items (*Header 1 to N*), each consists of *Header ID* and a variable-length *value* depending on the particular header – see Table 6.1. After the last header item, there are traffic (each 7 bytes) / temperature (6 bytes) samples in the same format as it is sent by the station in (see Chapter 5.5.2).

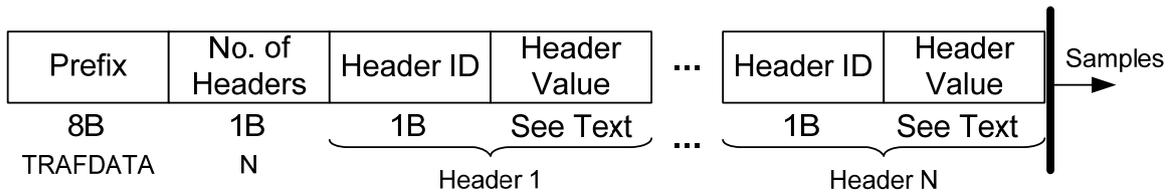


Figure 6.3 Binary File Structure

Header ID	Size [B]	Name	Description
0	2	SID	Station ID
1	1	PVER	Network Protocol Version
2	1	FWVER	Station Firmware Version
3	1	HWVER	Station Hardware Version
4	4	DTFROM	Acquisition Start Unix timestamp
5	4	DTTO	Acquisition End Unix timestamp
6	1	TYPE	Type of Data: 0=traffic, 1=temperature
7	2	SMPLNO	Number of Samples

Table 6.1 Binary File Headers

b) Text File

Its format is straight-forward and self-explanatory. Every line contains one value, headers begin with * character. Samples are separated by comma, and have the following format (Traffic text file example is below):

Traffic: lanenumber; datetime; speed; length

Temperature: datetime; tempin; tempout

```
* Traffic Data File
* Created on:2008-12-04 23:19:43
* SID=1
* PVER=1.0
* FWVER=0.9
* HWVER=1.0
* DTFROM=2008-12-01 18:00:07
* DTTO=2008-12-01 18:16:47
* TYPE=0
* SMPLNO=2
1;2008-12-01 18:00:07;55;500
1;2008-12-01 18:00:10;65;306
```

6.3.4 Binary Data Conversion

As described in Chapter 5.5.7, traffic and temperature data is transferred from stations in binary form, and it is necessary to unpack them before uploading to the database. PHP language contains *unpack* function for conversion from binary format. The following code snippet contains a function for vehicle sample conversion.

```
function UnpackVehicleSample($bistring)
{
    $unpackeddata=unpack("V1datetime/v1lanelength/C1speed", $bistring);
    $vehiclesample=array();
    $vehiclesample['unixtimestamp']=$unpackeddata['datetime'];
    $vehiclesample['dt']=gmdate("Y-m-d H:i:s", $unpackeddata['datetime']); //
convert unix timestamp
    $vehiclesample['datecol']=gmdate("Y-m-d", $unpackeddata['datetime']);
    $vehiclesample['lane']=( $unpackeddata['lanelength'] )>>12; //the highest 4
bits
    $vehiclesample['length']=$unpackeddata['lanelength']&0xffff; //the lowest
12 bits
    $vehiclesample['speed']=$unpackeddata['speed'];
    return $vehiclesample;
}
```

The *unpack* function is capable of converting data from little-endian, big-endian, and machine byte order representations. The Traffic Counter device uses **little-endian** byte order as described in Chapter 5.2.4. Table 6.2 contains *unpack* function unpacking format (char=1byte, short=2bytes, long=4bytes).

Code	Description	Code	Description
c	Signed Char	C	Unsigned Char
v	Unsigned Short	V	Unsigned Long

Table 6.2 *Unpack Function Little-endian Parameters*

6.3.5 Event Log

Client communication events are into files in LOG_FILES_PATH folder (Chapter 6.3.6). Filename is “#date.log” for normal logs and “ERR-#date.log” for error logs, where #date is a variable containing the date of logging – e.g. *2008-12-24.log*.

The log file is a csv file having the following structure:

Datetime;Client_ID;Event;Message

When a client is authorized, all headers obtained are listed in the log.

Example of log file:

```

2008-12-15
16:06:22;SID:1;PVER:10;FWVER:10;HWVER:10;DTFROM:1228150807;DTTO:1228151807
;SV:2;ST:2;
2008-12-15 16:07:16;CLIENT:1;ERROR;Connection refused by the client
2008-12-15 16:07:30;SID:1;DISCON

```

6.3.6 Server Configuration

File *ServerSettings.php* contains definitions (listed in Table 6.3) necessary to be set before the server is run. The server script uses some function from the web-interface and vice versa. Therefore, it is mandatory to set an appropriate path. All paths are **absolute** and need to end with \ or / character (depending on Operating System).

Definition Name	Type	Description
DEBUG	bool	Debug messages will be displayed in console output
SERVER_PORT	int	Port number the server will be running at
WEB_PATH	string	Path to the web-interface directory (to the folder where <i>index.php</i> file is)
BINARY_FILES_PATH	string	Where to store binary files which have not been parsed yet
BINARY_FILES_PATH_PARSED	string	Where to store parsed binary files (traffic and temperature)
DATA_FILES_PATH	string	Where to store text data files (traffic and temperature)
LOG_FILES_PATH	string	Where to store server log files

Table 6.3 Server Configuration Attributes

6.3.7 Running the Server

The PHP script has to be executed not as a website but as a command line script. At first, the server has to be configured – see Chapter 6.3.6.

a) Linux

The executable script has to have the following at the beginning (issue *which php* command for finding out the path):

```
#!/usr/bin/php
```

It also requires executable permission for running:

```
chmod +x Server.php
./Server.php
```

b) Windows

Consider that the PHP environment is installed in `C:\php\` directory.

The server can be started issuing the following command:

```
C:\php\php.exe C:\path_to_server\Server.php
```

6.4 Traffic Statistics

Traffic Statistics (a screenshot in Figure 6.4) is a PHP web application developed for administration of stations, their lanes, and for their monitoring including generating graphical outputs (i.e. charts).

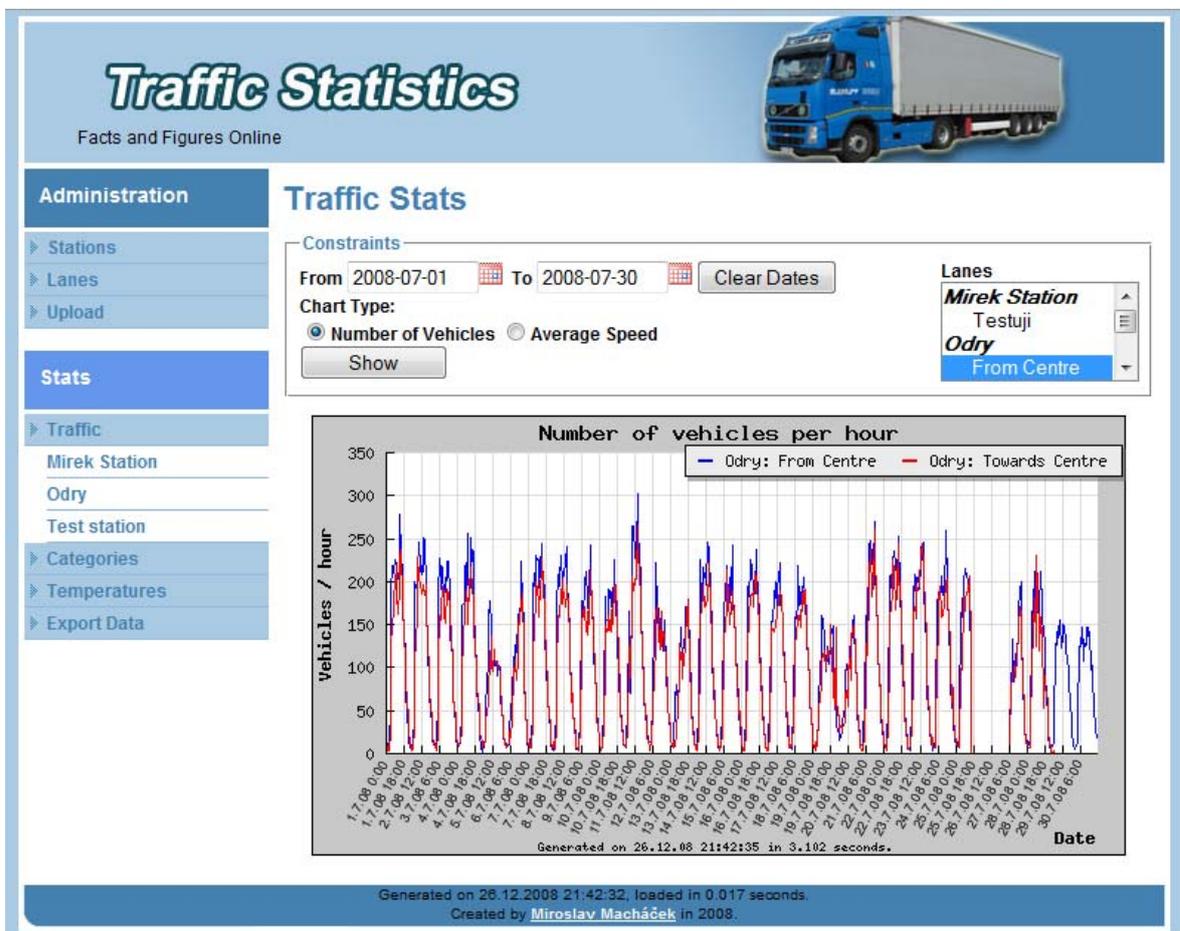


Figure 6.4 Traffic Statistics Web Application

The application employs jQuery UI [33] Javascript Library for sorting tables and for easy selection of date intervals, and JpGraph [35] PHP library for generating charts. The following outlines the features of the application:

- **Station Management**
Create/edit station, its description, view station last update date
- **Lanes Management**
Create/Edit station lanes, their lane number, description, view lane last update
- **Upload history**
Binary files upload, view upload history and details
- **Stats**
Lane Charts – number of vehicles per hour/day, average speed per hour/day, speed and length categories
Station Charts – indoor and outdoor temperature
- **Export**
Raw lane data export into CSV files

6.4.1 jQuery

jQuery is a open-source Javascript library [22] which simplifies HTML document access, event handling and animations through abstractions of low-level functions and attributes. Particularly jQuery UI is its part which provides easy-to-implement user interface - effect such as resizing, sorting, drag & drop and others.

This thesis employs two plugins from jQuery UI – **Datepicker** and **Tablesorter**. Therefore, facts mandatory to get those two plugins working are mentioned only. For an extensive description of jQuery features, the reader should refer to [22] and [33].

6.4.1.1 Usage

The heart of the entire Library is the file *jquery.js*. It has to be included in HTML source code at first:

```
<script type="text/javascript" src="jquery.js"></script>
```

jQuery functions can be used as soon as the DOM is ready, after which the jQuery code can be run. The following demonstrates how to achieve that:

```
<script type="text/javascript">
$(document).ready(function(){
// executed once the DOM is ready, put user's jquery code here
$("a").click(function() {
    alert("Hello world!");
});
});
</script>
```

Which does the same as the following code (without jQuery):

```
<a href="" onclick="alert('Hello world')">Link</a>
```

6.4.1.2 Datepicker

It is a plugin embedded in jQuery UI package. It provides an easy date selection via a small calendar menu. It can also be configured so that it allows to select a date range.

The Datepicker is bound to an input tag:

```
<input type="text" size="10" value="" id="startDate"/>
```

After that, the following (to be placed into *ready* function – see Chapter 6.4.1.1) causes a calendar to pop up when the text field is focused, the outcome is shown in Figure 6.5.

```
$("#singleDate").datepicker({
  maxDate: 0,
  dateFormat: $.datepicker.W3C,
  showOn: "both",
  buttonImage: "images/calendar.gif",
  buttonImageOnly: true,
  firstDay: 1
});
```

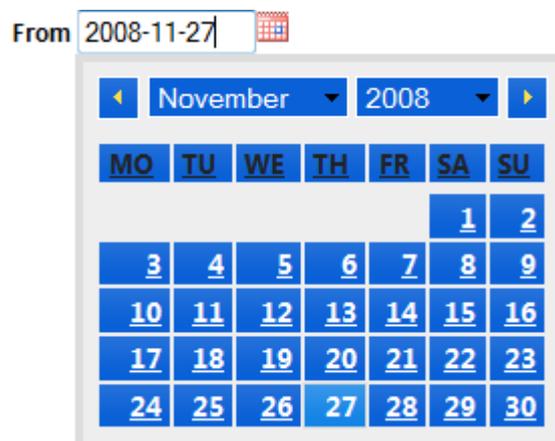


Figure 6.5 Datepicker plugin for jQuery

6.4.1.3 Tablesorter

Tablesorter [34] provides sorting feature and a nice graphical layout of HTML tables via jQuery library. It is a former plugin of jQuery UI since it has been separated from jQuery. Figure 6.6 demonstrates the result of Tablesorter plugin.

Lane Number [▲]	Direction [◆]	Name, Desc [◆]	Last Data Update [◆]	Speed Limit [◆]	Functions
0	1	Towards Centre Traffic and LPR	2008-12-04 23:25:57	50	Traffic Stats Temperature Stats Edit
1	0	From Centre LPR and speed	2008-12-04 20:53:07		Traffic Stats Temperature Stats Edit

Figure 6.6 Tablesorter Plugin for jQuery

The HTML table should contain *THEAD* and *TBODY* tags so that the *tablesorter* can be applied on the table. The following code outlines the table structure:

```
<table id="myTable">
<thead>
<tr>
<th>Last Name</th><th>First Name</th><th>Email</th><th>Due</th><th>Web
Site</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td><td>John</td><td>jsmith@gmail.com</td><td>$50.00</td>
<td>http://www.jsmith.com</td>
</tr>
</tbody>
</table>
```

Tablesorter javascript file should also be included in the HTML file:

```
<script type="text/javascript" src="ui/jquery.tablesorter.js"></script>
<script type="text/javascript" src="ui/jquery.metadata.js"></script>
```

If the users wants to use jQuery inline function, *metadata.js* needs to be included. Subsequently, jQuery code has to be added to *ready* function described in Chapter 6.4.1.1.

```
$.tablesorter.defaults.widgets = ['zebra'];
$("#MyTable").tablesorter({sortList: [[0,0]]});
```

The code above sets the first column of table *MyTable* as the sorting column, and *zebra* widget makes different color of every second row of the table.

Inline functions provide the functionality of changing *Tablesorter* properties through code placed within the object tag. For instance, the following disables sorting feature on the associated column:

```
<th class="{sorter: false}">Functions</th>
```

6.4.2 JpGraph

Is an object-oriented PHP library for generating various types of charts such as line, bar, pie and ring plots. A chart can either be saved as an image file or shown directly within a web site. Within this thesis, the second method, alongside line and pie plots, has been used.

The usage of the JpGraph library is straight-forward and well-documented in [23], and it would be beyond the range of this thesis to provide a description of the library. The following PHP code only demonstrates the simplicity of using the library, Figure 6.7 contains the chart generated:

```
<?php
include ( "../jgraph.php" );
include ( "../jgraph_line.php" );
$ydata = array(11,3, 8,12,5 ,1,9, 13,5,7 ); // Some data
$graph = new Graph(350, 250,"auto");
// Create the graph. These two calls are always required
$graph->SetScale( "textlin");
$lineplot =new LinePlot($ydata); // Create the linear plot
$lineplot ->SetColor("blue");
$graph->Add( $lineplot); // Add the plot to the graph
$graph->Stroke();// Display the graph
?>
```

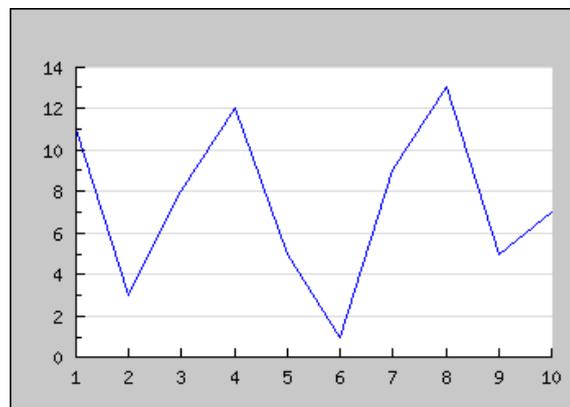


Figure 6.7 Simple Line Plot via JpGraph Library

6.4.3 Traffic Charts API

In order to provide the transparency of JpGraph functions, an API for generating traffic charts based on database samples has been implemented. Scripts for chart generating are in /charts/ folder under the main web-interface directory (i.e. where *index.php* can be found). Each file generates different type of chart which is generated right into the screen, and variables are passed over query string - \$_GET array whose variables are listed in Table 6.4.

GET variable	Description
idlane	Traffic lane(s) to be shown in the chart
idstation	Station(s) to be shown in the chart
datefrom, dateto	Range of date to be shown
daily	=1 -> show data per day =0 -> show data per hour
interval	day interval – i.e. how many days to show
showlength	=1 -> length categories will be generated =0 -> speed categories will be generated

Table 6.4 Traffic Charts API Variables

Variables *datefrom* and *dateto* are dates in MySQL format (ex: 2008-12-24), and they limit date period to be shown in the chart. In some charts, *idlane/idstation* are **array** variables (ex: *idlane[]*). This allows to plot data from several lanes/station in one chart.

Filename	Description	GET variable
VehiclesPerDay.php	Number of Vehicles in a lane per day/hour	idlane[] datefrom, dateto daily
AvgSpdPerHour.php	Average speed in a lane per hour	idlane[] or idstation datefrom, dateto
PercentagePie.php	Speed and length categories of a lane	idlane datefrom interval showlength
AvgTempPerHour.php	Average temperatures	idstation[] datefrom, dateto

Table 6.5 Traffic Charts API

In order to show traffic data of several lanes in one graph, *idlane* GET variable is passed as an array, and the URL may look similarly to the following:

```
VehiclesPerDay.php?idlane[]=1&idlane[]=2&datefrom=2008-10-10&dateto=2008-11-10&daily=1
```

If it is desired to embed the chart within a web page, this URL query can be put in an *img* tag within the HTML code:

```

```

Configuration of Traffic Charts API is under *charts/chartfunctions.inc* where also speed and length categories can be changed. Examples of charts generated via the Traffic Charts API are depicted in Figure 6.8.

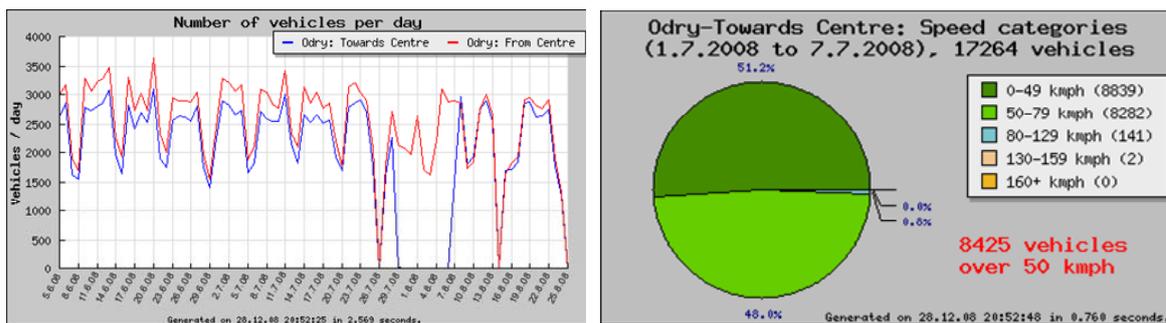


Figure 6.8 Charts Generated in Traffic Charts API

6.5 Database Benchmarking

Table *trafficdata* (model in Chapter 6.2) has been filled with roughly **400000** pre-generated samples. This table contains data of vehicle samples of all stations, and queries for generating charts are executed on this table. Therefore, its performance is vital. The following code has been run in order to measure query execution time, the query is taken from chart-generating function:

```
<?php
$mysql=MySQLConnect();
$strttm = explode(' ', microtime());
$strttm = $strttm[1] + $strttm[0];
// BEGIN there should be operations to be measured
$q=mysql_query("SELECT COUNT(*) FROM trafficdata WHERE idlane='1' AND
datecol BETWEEN '2008-07-01' AND ADDBDATE('2008-07-01',7-1) AND length
BETWEEN '500' AND '899'");
// END
$endtm = explode(' ', microtime());
$tottm = $endtm[0] + $endtm[1] - $strttm;
$buf=sprintf('%f', $tottm);
echo $buf."<br>\n";
list($count)=mysql_fetch_row($q);
echo $count;
MySQLClose($mysql);
?>
```

It turned out that adding an extra column *datecol*, containing a date extracted from *dt* column, **decreased** the query time, because almost every query for

generating a chart is date limited. In addition, adding a multi-column index to *trafficdata* table on (*idlane,datecol*) also **improved** the query time. Table 6.6 summarizes the effect of good/bad indexes on execution time. It proves the fact that a wrong index slows down the query – in this case, because *dt* column contains datetime – which is almost everytime unique, the index misleads the database engine.

In case the table contains several indexes, *EXPLAIN* keyword can be used in the query. The database returns a list of reasonable/used indexes.

Index On	Execution Time [s]
No Indexes	0.36
idlane,dt	1.6
idlane, datecol	0.11

Table 6.6 MySQL Query Benchmark

7 Conclusion

7.1 System Summary

The outcome of this thesis is a fully functional, stand-alone system for traffic data acquisition such as speed and length of vehicles. The system developed consists of 4 parts.

The first part, the Inductive Loop Detector, finds out the presence of a vehicle with the help of a wired loop embedded within the roadway. However, given that it generally detects metal objects, it can be used in various types of automation such as proximity detection and others. The second part, Traffic Counter, employs 4 loop detectors, and gathers traffic data from 2 lanes. The data collected is in particular time interval transferred to the server via a GPRS modem.

The last part of the system comprises two server applications. The first one receives data from multiple stations and uploads them to the database. The latter is a web interface for monitoring traffic at stations and generating statistical and graphical outputs.

Once the entire system has been configured, it automatically gathers traffic data from lanes of stations, transfers them to the server where samples are used for statistical information such as speed and length categories and lane occupancy.

7.2 Tests in the field

The entire system has been placed in the field for some period to gather traffic data. The Inductive Loop Detector has been tested with an inductive loop built in a tar roadway. The loop was rectangular with dimensions 2 m x 2 m, and inductance of approx. 100uH. The loop frequency was perfectly stable having almost zero variation, hence, the overall device sensitivity is very high. A small vehicle (1000 kg weight, 3,7 m length), even though the loop inductance was so low, caused a change in inductance of 2%. Such a substantial change can be detected by the device without any problems. To summarize, the ILD device could also be used to detect motorbikes or even bicycles (with different loop size) in normal traffic.

Speed accuracy of the Traffic Counter has been examined with the help of a GPS device that provided reliable speed information. The Traffic Counter speed measurement error increases rapidly at speed lower than approx. 25 kmph. It is caused

by different reactions of both ILDs in the traffic lane. The speed error remains at approx. 5% for speeds above the threshold. It should be noted that the test was conducted up to speeds of 90 kmph.

7.3 Further Work

As further extension of the thesis, some features could be added to the Inductive Loop Detector. Because of its universality, there are various ways how to use that device. Therefore, a convenient feature would be to allow the user to set parameters, such as trigger delay, via the unused on-board switches.

Moreover, the IDL could be modified to continuously sample loop frequency when a vehicle passes over the loop, and create a vehicle signature, which could be used for recognizing number of axles of the vehicle. Similarly, it could also be employed in vehicle matching at two remote places, as mentioned in Chapter 2.3.

With regard to the Traffic Counter, in order to increase the number of monitored traffic lanes, an auxiliary microcontroller peripheral (i.e. port expander) for increasing the number of detectors connected could be employed. A possibility of saving data on a SD card would come in useful at places without a GPRS connection.

As for the traffic statistics web-interface, different charts can be added, and integration into ITS, for further data analysis and traffic control, could be performed.

8 References and Bibliography

- [1] Pavel Příbyl, Radim Mach. *Řídicí systémy silniční dopravy*. 1st printing, Prague: Vydavatelství ČVUT, 2003. ISBN 80-01-02811-9
- [2] Lawrence A. Klein. *Traffic Detector Handbook: Third Edition - Volume I*. Milton K. Mills, David R.P. Gibson. 3rd Edition, Wash. D.C.: Federal Highway Administration; U.S. Department of Transportation. 2006. Publication No. FHWA-HRT-06-108
- [3] Lawrence A. Klein. *Traffic Detector Handbook: Third Edition - Volume II*. Milton K. Mills, David R.P. Gibson. 3rd Edition Wash. D.C.: Federal Highway Administration; U.S. Department of Transportation. 2006. Publication No. FHWA-HRT-06-139
- [4] Carlos Sun, Stephen G. Ritchie. *Individual Vehicle Speed Estimation Using Single Loop Inductive Waveforms*. Berkeley: University of California. 1999. ISSN 1055-1417
- [5] Luz Elena Y. Mimbela. *A Summary of Vehicle Detection and Surveillance Technologies used in Intelligent Transportation Systems* [online]. Lawrence A. Klein, Ph.D., P.E. New Mexico: The Vehicle Detector Clearinghouse. 2007. Available from <<http://www.nmsu.edu/~traffic/>> [viewed 17 January 2009]
- [6] Virgil F. Totten. *Application of Vehicle Detector Waveforms in Vehicle Re-Identification and Evaluating Detector Installation Performance*. Indiana: Purdue University. 2008
- [7] Seyed Mohmamad Tahib. *Vehicle Re-Identification Based On Inductance Signature Matching*. University of Toronto. 2001
- [8] Carlos Sun. *An Investigation in the Use of Inductive Loop Signatures for Vehicle Classification*. California PATH, University of California. 2000. Report No. UCB-ITS-PRR-2000-4
- [9] Future Technology Devices International Limited. *USB Chips* [online]. Available from <<http://www.ftdichip.com>> [viewed 14 January 2009]
- [10] Custom Computer Service, Inc. *C Compiler Reference Manual* [online]. Available from <<http://www.ccsinfo.com/downloads.php>> [viewed 1 January 2009]
- [11] Custom Computer Service, Inc. *Developers forum* [online]. Available from <<http://ccsinfo.com/forum>> [viewed 1 January 2009]

- [12] Maxim Integrated Products, Inc. *DS1302 Trickle-Charge Timekeeping Chip Data Sheet* [online]. Available from <http://www.maxim-ic.com/quick_view2.cfm/qv_pk/2685/t/al> [viewed 1 January 2009]
- [13] Maxim Integrated Products, Inc. *DS18B20 Programmable Resolution 1-Wire Digital Thermometer Data Sheet* [online]. Available from <http://www.maxim-ic.com/quick_view2.cfm/qv_pk/2812/t/al> [viewed 1 January 2009]
- [14] Maxim Integrated Products, Inc. *MAX232 Multichannel RS-232 Drivers/Receivers Data Sheet* [online]. Available from <http://www.maxim-ic.com/quick_view2.cfm/qv_pk/1798> [viewed 1 January 2009]
- [15] Microchip Technology Inc. *PIC18F2420/2520/4420/4520 Data Sheet* [online]. Available from <<http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010270>> [viewed 1 January 2009]
- [16] Microchip Technology Inc. *PIC18F2525/2620/4525/4620 Data Sheet* [online]. Available from <<http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010284>> [viewed 1 January 2009]
- [17] Enfora L.P. *Enfora Enabler-G GSM/GPRS Radio Modem AT Command Set Reference*, version 1.13, published on 7th June 2004
- [18] Shane Tolmie. *MicrochipC.com PIC micros and C*, [online]. Available from <<http://www.microchipc.com>> [viewed 1 January 2009]
- [19] Midland Amateur Radio Club Inc. *LC Meter*, [online]. Available from <<http://www.marc.org.au>> [viewed 1 January 2009]
- [20] Tecnick.com s.r.l. *Inductance Calculator*, [online]. Available from <www.technick.net> [viewed 1 January 2009]
- [21] The PHP Group. *PHP Language Manual*, [online]. Available from <<http://www.php.net/manual/en/>> [viewed 1 January 2009]
- [22] John Resig and the jQuery Team. *jQuery JavaScript Library*, [online]. Available from <<http://jquery.com>> [viewed 1 January 2009]
- [23] Aditus Consulting. *JpGraph Documentation*, [online]. Available from <<http://www.aditus.nu/jpgraph/documentation.php>> [viewed 1 January 2009]
- [24] TENcom, s.r.o. *GPRS Modem Configuration Manual*, [online]. Available from <<http://www.tencom.cz/download.php>> [viewed 1 January 2009]

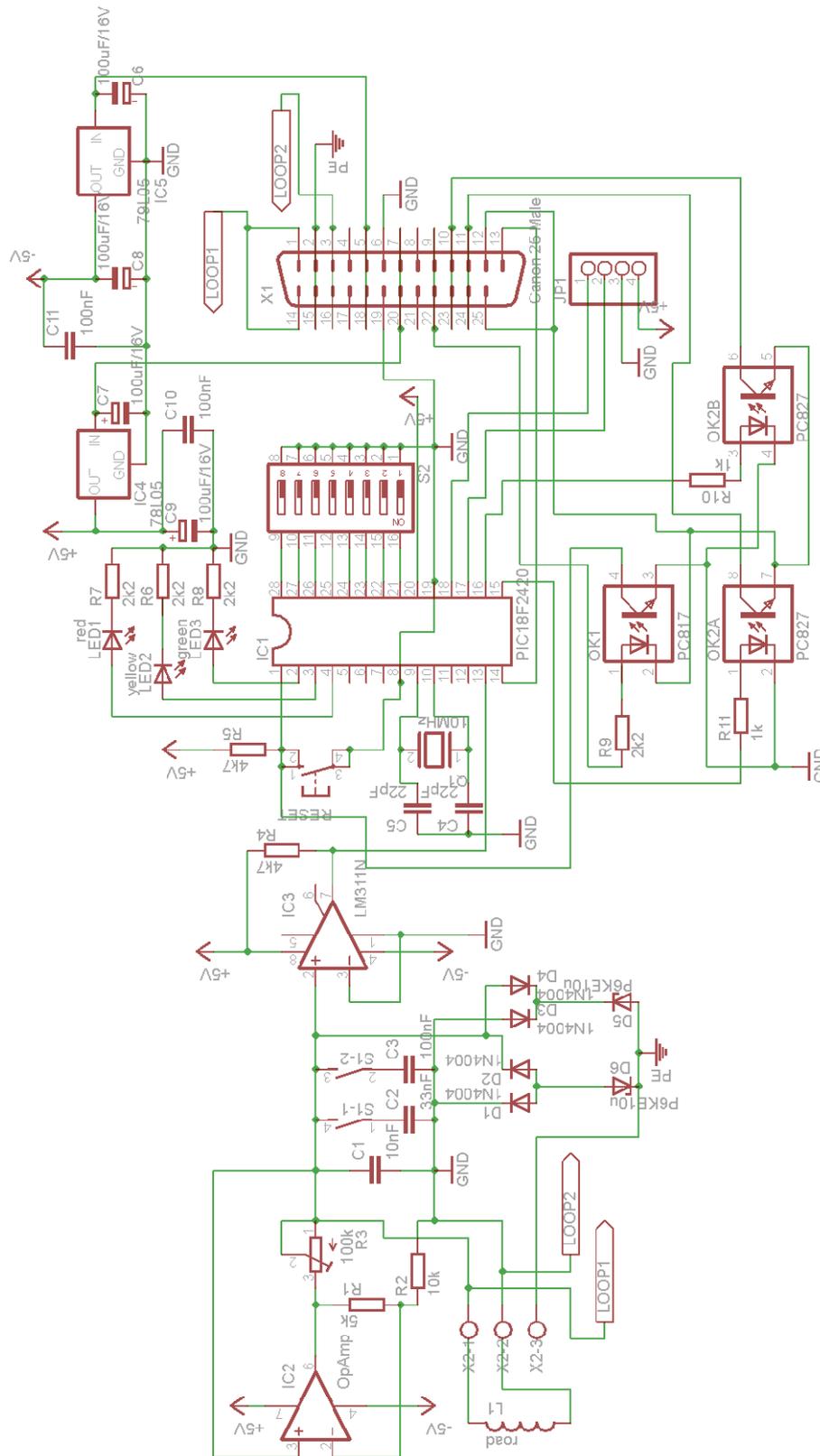
9 Software and Packages Used

- [25] ASIX, s.r.o. *ASIX UP* [online]. Available from <http://tools.asix.net/prg_presto.htm> [viewed 1 January 2009]
- [26] Claudiu Chiculita. *Tiny PIC bootloader* [online]. Available from <<http://www.etc.ugal.ro/cchiculita/software/picbootloader.htm>> [viewed 1 January 2009]
- [27] Custom Computer Services, Inc. *PCWH C Compiler for Microchip PIC* [online]. Available from <<http://ccsinfo.com>> [viewed 1 January 2009]
- [28] Microchip Technology Inc. *MPLAB Integrated Development Environment* [online]. Available from <http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002> [viewed 1 January 2009]
- [29] The Apache Software Foundation. *The Apache HTTP Server Project*, [online]. Available from <<http://httpd.apache.org>> [viewed 1 January 2009]
- [30] The PHP Group. *PHP Language*, [online]. Available from <www.php.net> [viewed 1 January 2009]
- [31] The Sun Microsystems, Inc. *MySQL Database Server*, [online]. Available from <www.mysql.com> [viewed 1 January 2009]
- [32] MicroOLAP Technologies Ltd. *MicroOLAP Database Designer for MySQL*, [online]. Available from <<http://www.microolap.com/products/database/mysql-designer>> [viewed 1 January 2009]
- [33] Paul Bakaus and the jQuery Team, Inc. *jQuery UI* [online]. Available from <<http://ui.jquery.com/>> [viewed 1 January 2009]
- [34] Christian Bach. *Tablesorter Plugin for jQuery* [online]. Available from <<http://tablesorter.com/docs>> [viewed 1 January 2009]
- [35] Aditus Consulting. *JpGraph* [online]. Available from <<http://www.aditus.nu/jpgraph/>> [viewed 1 January 2009]
- [36] CadSoft Computer GmbH. *EAGLE* [online]. Available from <www.cadsoft.de/> [viewed 1 January 2009]

Appendix 1 Content of the Attached CD

- **/DP2009-Machacek.pdf** This thesis in Adobe PDF format
- **/Docs/** Documentation, Data Sheets, Manuals
- **/Install/** Installation and Setup Executables
- **/PCB/** **PCBs, Schematics, Bill of Materials**
 - **ILD** Inductive Loop Detector
 - **TrafficCounter** Station Device
- **/Src/** **Source Codes and Executables**
 - **DataServer/** PHP TCP Server
 - **ILD/** Inductive Loop Detector
 - **TrafficCounter/** Station Device
 - **TrafficStats/** PHP Web Application

Appendix 2 Inductive Loop Detector Schematic



Appendix 4 Inductive Loop Detector Bill of Materials

Quantity	Devices	Component
1	SW1	microswitch
4	D1.D2.D3.D4	1N4007
1	R9	1k
3	R5.R6.R7	1k8
1	R2	2k7
1	DIPSW2	2xdipswitch
1	R11	4k7
1	JP1	4xjumper
1	DIPSW1	8xdipswitch
1	XT	10MHz
3	R8.R10.R12	10k
1	C5	10nF
2	C8.C9	22p
1	C6	33nF
1	IO2	78L05
1	IO1	79L05
5	C10.C11.C12.C13.C7	100nF
2	C3.C4	100u/10V
2	C1.C2	100uF/16V
2	R3.R4	820R
1	CON1	ARK2500V-A-3P
1	E\$1	Cannon 25 Male
1	LED3	GRN 2mA
1	E\$21	LM311
2	D5.D6	P6KE10V UNIDIR
1	OK1	PC817
1	OK2	PC827
1	E\$24	PIC18F2420
1	LED1	RED 2mA
1	E\$20	TL072
1	LED2	YEL 2mA

Appendix 5 Inductive Loop Detector User's Manual

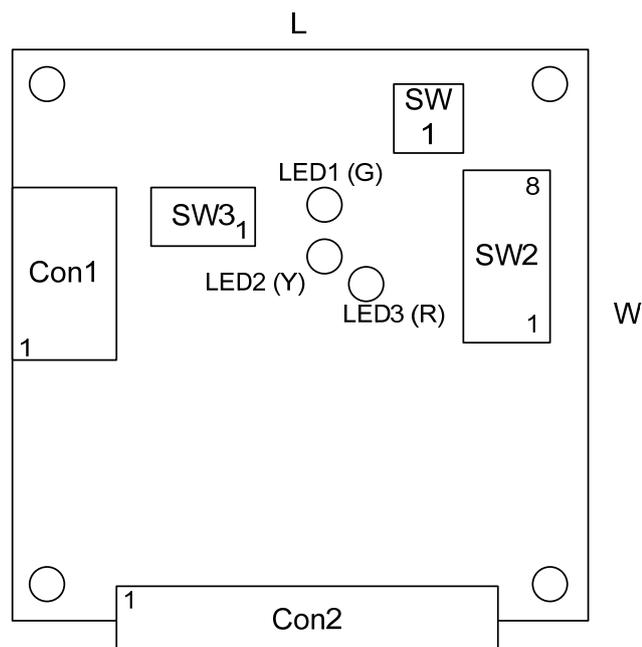


1) Introduction

The Inductive Loop Detector is a microcontroller-based device for detecting an electrically conducting metal object near the inductive loop. The main purpose of the device is to detect vehicle presence by sensing an inductance change caused by the vehicle passing above the loop embedded in the roadway.

The device provides self-tuning feature so that a wide range of loop inductances is achieved, the loop operating frequency can be set by on-board switches. The user is notified about the state of the device by 3 on-board LEDs.

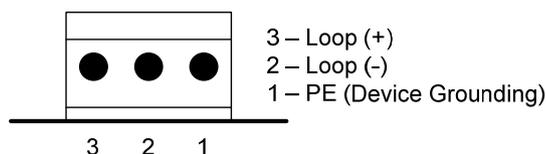
2) Technical Data



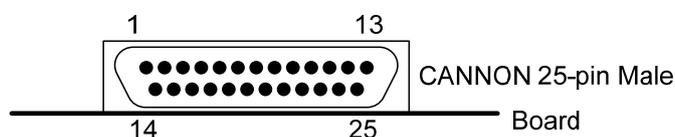
Power Requirements	+15V DC, approx. 100mA
Board Dimensions	71 mm (L) x 73 mm (W) x 23 mm (H)
Mounting	Pillar: 4x 3mm hole (distance 61 mm, 63 mm) Connector Plug-in
Connectors	Con1: ARK2500V-A-3P ARK2500F-A-3P for cable side Con2: Cannon 25-pin Male
Surge Protection	On Loop inputs: Gas Discharge Tube TVS Diodes
Tuning	Fully Automatic Inductance Range (Theoretical): 8 to 6000 uH
Loop Operating Frequency	20-145kHz 4 selectable positions via SW3 switches
Sensitivity	4 selectable positions via SW2 switches: Low, Medium Low, Medium High, High
Digital Inputs	Opto-isolated (<i>External Reset</i>)
Digital Outputs	TTL outputs (<i>Vehicle Above</i>) Opto-isolated outputs (<i>Vehicle Above, Vehicle Above Impulse</i>)
LEDs	Green – Vehicle Presence Yellow – Calibration, Operating Frequency Red - Error

3) Connectors

CON1 – Loop and Grounding (front view)



CON2 – Main Connector (front view)



Pin No.	Name	Direction	Description
1, 14	L1	In	Inductive Loop (+)
2, 15	PE	In	Grounding to the earth
3, 16	L2	In	Inductive Loop (-) (=GND)
5, 18	DC-	In	Negative Voltage Power Supply (-7 to -12V)
6, 19	GND	In	System Ground
7, 20	DC+	In	Positive Voltage Power Supply (+7 to +12V)
9, 22	RST	In	External Reset (with opto-isolation – against EXTGND), Input Current = 5mA
10, 23	VEHI	Out	Vehicle Above 100ms Impulse (with opto-isolation – against EXTGND)
11, 24	VEH	Out	Vehicle Above Permanent (with opto-isolation – against EXTGND)
12, 25	EXTGND	In	Ground for opto-isolation
13	VEHTTL	Out	Vehicle Above – TTL signal (against GND)

4) Settings

SW1 – Reset Button

SW2 – Device Parameters

Sensitivity	Pin Switch On	Sensitivity
pins 7, 8	none	Low
	7	Medium Low
	8	Medium High
	7, 8	High

SW3 – Loop Frequency Settings

Pin Switch On	Frequency
none	High
1	Medium High
2	Medium Low
1, 2	Low

5) LED purpose

Device State	LED	Action	Description
No Signal	Red	On	There is no loop connected or the oscillator frequency is out of range (has to be 20 kHz – 145 kHz)
Calibrating	Yellow	On	Calibration in progress
Calibrating	Red	Blink	Blinks once if calibration was not successful
Calibrating	Yellow	Blink	Blinks for 2 seconds in case of successful calibration
Vehicle-recognition	Green	Off	No Vehicle above the loop
Vehicle-recognition	Green	On	Vehicle above
Vehicle-recognition	Yellow	Blink	Blinks according to the actual loop frequency (e.g. 30 kHz = 3x, 80 kHz = 8x)

6) Operating Instructions

I. Principle of Operation

The detection is based on sensing a change of loop inductance when a metal part approaches the loop. Loop inductance depends on loop dimensions, number of turns, feeder length, and the ambient environment.

II. Start Up

1. Connect the loop either via Con1 or via Con2. Feeder cable to the loop should be **twisted in pair** so that crosstalks and other interferences are as lower as possible.
2. Connect the power supply (through Con2)

III. Calibration

The calibration is fully automatic. When a power is supplied to the device, or a reset is performed, the detector will automatically tune itself to the loop it is connected to. In case the loop is not in steady state (i.e. metal parts are passing by the loop), the detector tries to recalibrate.

Once the device has been calibrated, detection of metal objects is turned on. The detector also handles environmental effects (temperature etc.) which cause slow changes in inductance by auto-recalibrating on the fly.

IV. Operation

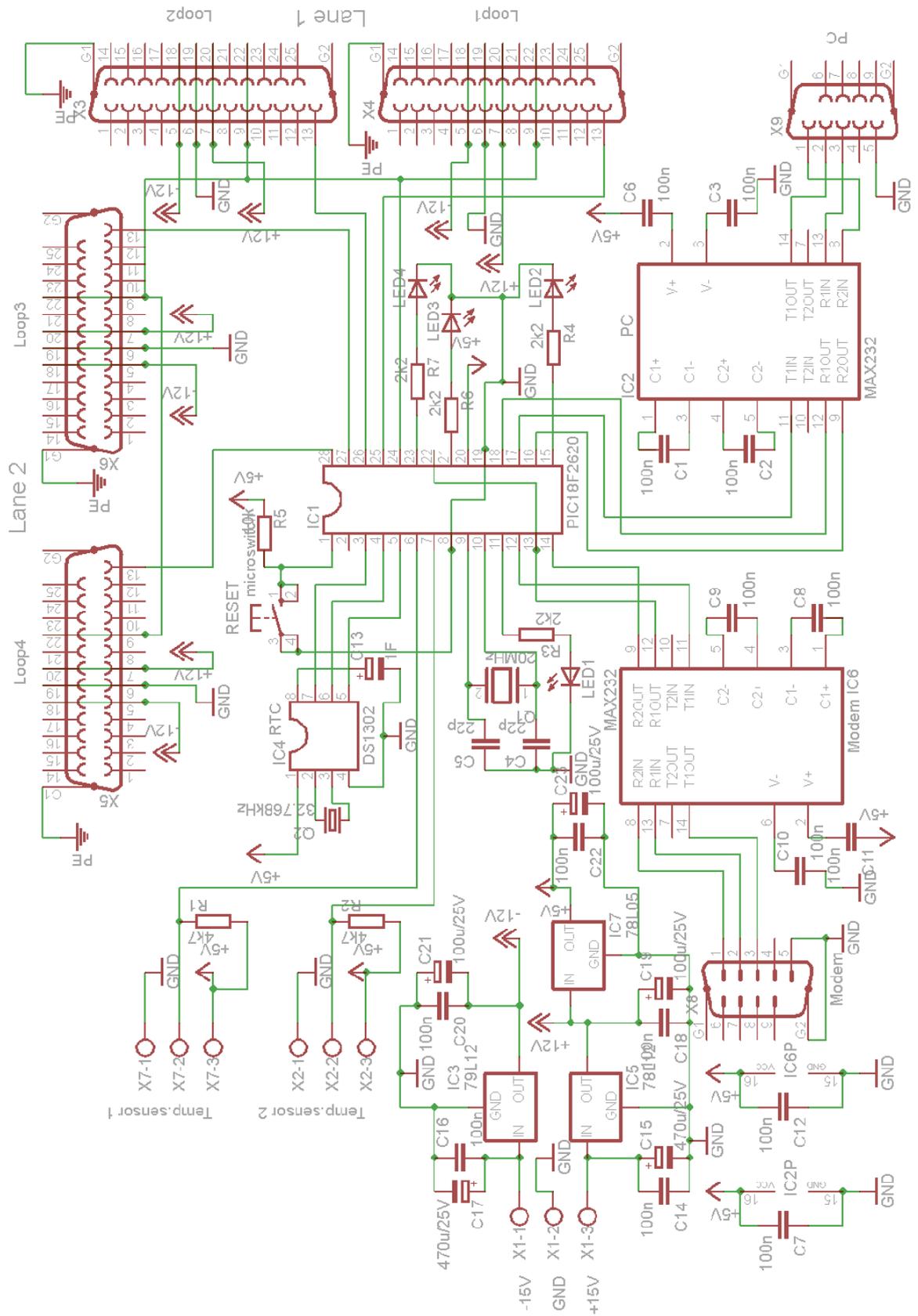
If there is no loop connected, or the loop frequency is out of range, the red LED lights. The yellow LED is turned on when a calibration is taking place. If calibration is not successful, the red light blinks once.

During the normal operation when the device is calibrated, the yellow LED signalizes the current loop frequency by blinking. The green LED shows whether a metal part is near the loop.

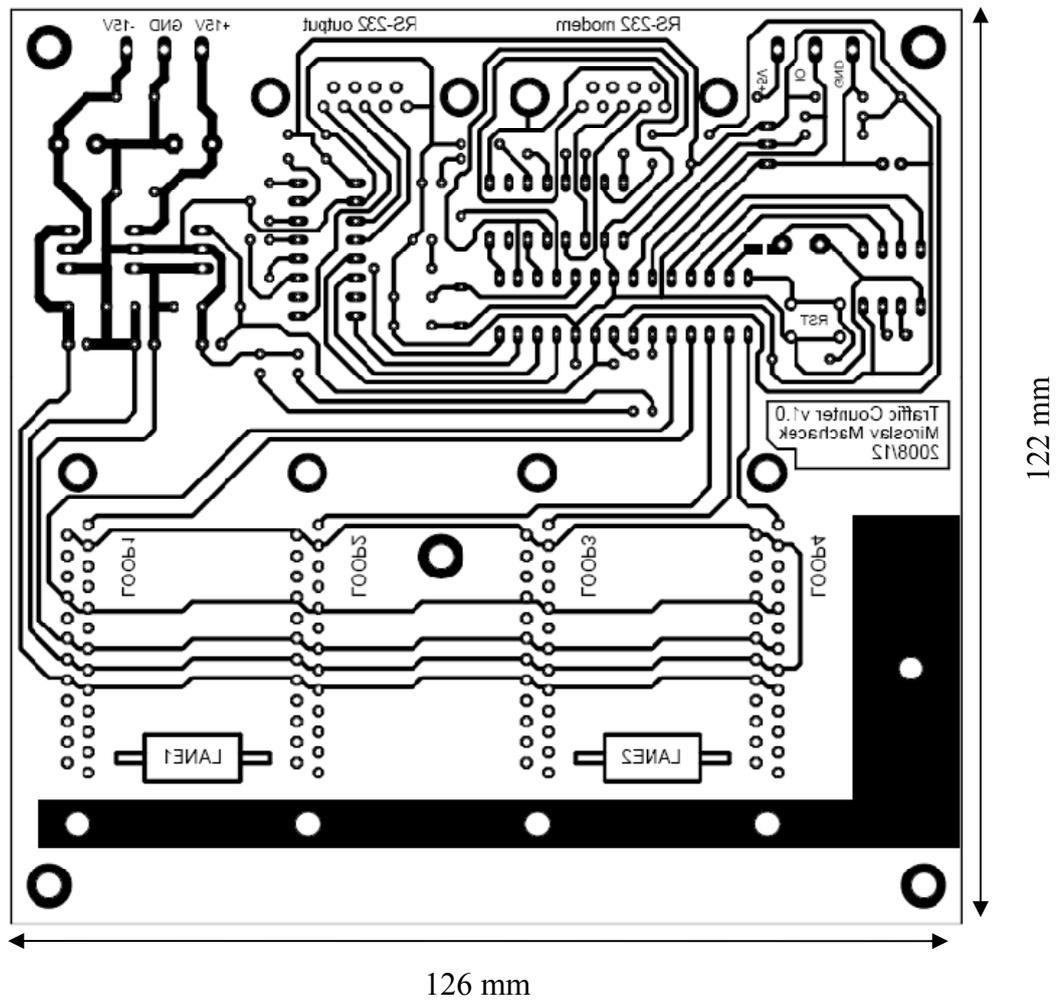
7) Troubleshooting

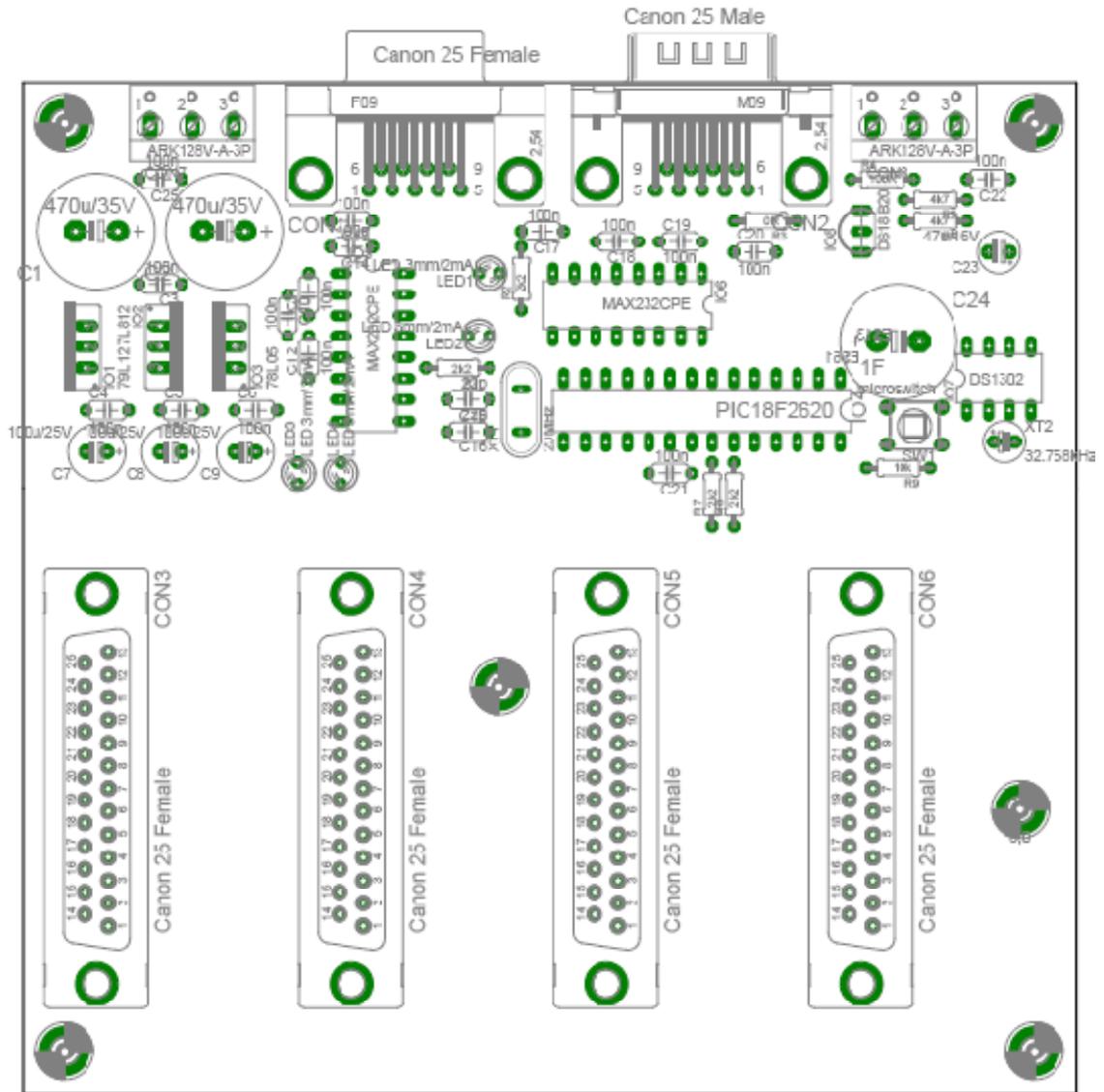
Fault	Possible Cause	Hint
No LED lights on power up	Wrong power connection	Check power feed
Red LED is still on	Faulty loop of feeder connection	Check loop installation and connection
	Loop operating frequency out of range	Select different frequency settings (SW3)
The detector cannot calibrate (e.g. Red LED blinks occasionally)	Crosstalks with adjacent detector	Select different frequency settings (SW3)

Appendix 6 Traffic Counter Schematic



Appendix 7 Traffic Counter PCB

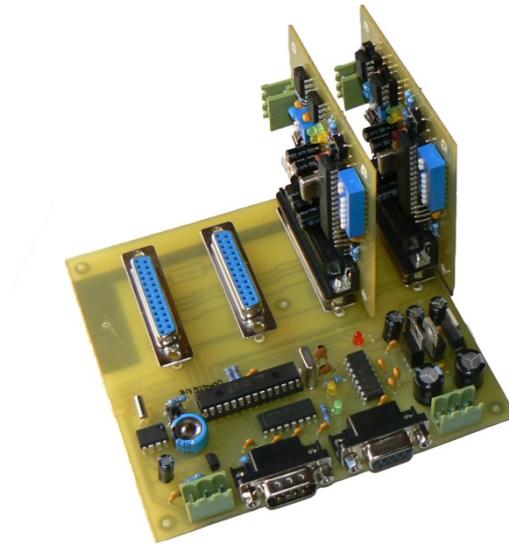




Appendix 8 Traffic Counter Bill of Materials

Quantity	Devices	Component
1	R3	0R
1	C24	1F
4	R1.R2.R7.R8	2k2
2	R5.R6	4k7
1	IO2	78L12
1	R9	10k
1	XT1	20MHz
2	C15.C16	22p
1	XT2	32.768kHz
1	C23	47u/16V
1	IO3	78L05
1	IO1	79L12
1	R4	100R
16	C3.C4.C5.C6.C10.C11.C12.C13.C14.C17.C18.C19.C20.C21.C22.C25	100n
3	C7.C8.C9	100u/25V
2	C1.C2	470u/35V
2	CON7.CON8	ARK2500V-A-3P
1	CON1	Cannon 25 Female
4	CON3.CON4.CON5.CON6	Cannon 25 Female
1	CON2	Cannon 25 Male
2	IO8, External Temperature Sensor	DS18B20
1	IO7	DS1302
4	LED1.LED2.LED3.LED4	LED 3mm/2mA
2	IO5.IO6	MAX232CPE
1	IO4	PIC18F2620
1	SW1	microswitch

Appendix 9 Traffic Counter User's Manual

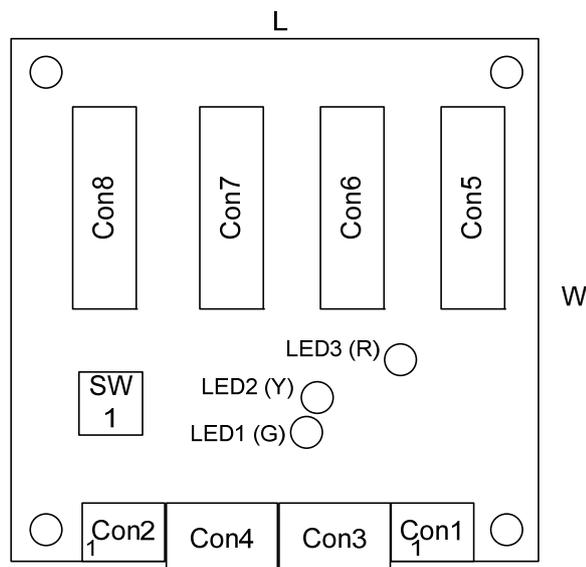


1) Introduction

The Traffic Counter is a device for measuring speed and length of vehicles. The device allows to gather data from up to 2 traffic lanes. Each lane is monitored by 2 Inductive Loop Detectors that are connected with 2 inductive loops build in the roadway surface of the lane.

The device, apart from vehicle data, also takes samples of indoor and outdoor temperature, and all data is transferred via a GPRS modem to the server. Device parameters can be configured via a command-line interface running over RS-232 line.

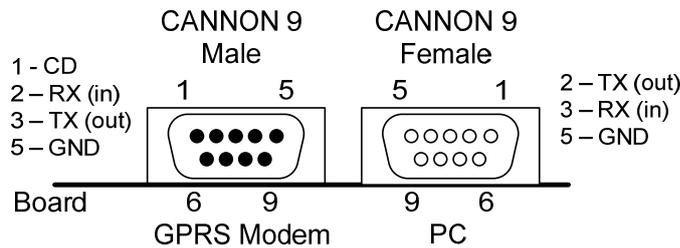
2) Technical Data



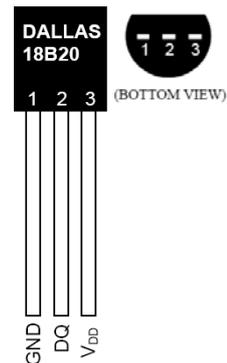
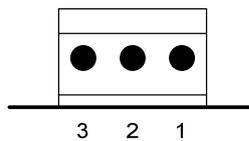
Power Requirements	+/-15V DC, approx. 400mA
Board Dimensions	126 mm (L) x 125 mm (W) x 23 mm (H) (excluding Detector boards)
Mounting	Pillar: 4x 3mm hole (distance 116 mm, 112 mm)
Connectors	Con1, Con2: ARK2500V-A-3P (ARK2500F-A-3P for cable side) Con3: Cannon 9-pin Female Con4: Cannon 9-pin Male Con5-8: Cannon 25-pin Female
Traffic Lanes	Up to 2 lanes monitored
Inductive Loop Detectors	Up to 4 detectors, 2 needed for each traffic lane
Communication	GPRS Modem: RS-232, 19200, 8N1, no handshake PC: RS-232, 115200, 8N1, no handshake
Date and Time	RTC chip on board, time keeping will be working several after power supply disconnect
Traffic Data	Capture Date, Time, Lane, Speed, Length
Sample Buffer Memory	up to 400 Vehicle and 70 Temperature Samples transferred to server over GPRS in time intervals
Temperature Sensors	2 Digital sensors: 1 On-Board (Indoor) 1 On External Connector (Outdoor)
Device Configuration	Over PC RS-232, Command-line Interface
Buttons and Switches	Device reset button
LEDs	Green – Device State Yellow – Vehicle Sampling Red – GPRS State

Connectors

Con3, Con4 (Modem and PC Serial Lines - front view):



Con1, Con2 (Power Supply, External Temperature Sensor – front view):



Pin No.	Con1 (Power Supply In)	Con2 (Ext. Temp. Sens.)
1	-15V DC	+5V (sensor pin 3)
2	GND	IO (sensor pin 2)
3	+15V DC	GND (sensor pin 1)

Con5-8 (Loop Detectors): refer to *Inductive Loop Detector User's Manual*

3) LED signalization

LED	Color	Event	Description
1	Green	On	Device is powered
		Blink	Data acquisition is in progress (<i>START</i> command)
2	Yellow	Blink	Vehicle sample has been taken
3	-	Not Used, Future Purpose	
4	Red	On	GPRS connection being initialized
		Slow	GPRS connection has been established, vehicle and temperature samples being transferred
		Blink	and temperature samples being transferred
		Fast	20 fast blinks means GPRS connection error, retry in 2 minutes

4) Operating Instructions

I. Start Up

1. Connect external temperature sensor over a 3-wire cable to Con2. Place the sensor outside the box so that it measures outdoor temperature.
2. Plug in 2 or 4 loop detectors – Lane1 = Con5,6 ; Lane2= Con7,8
Connect corresponding loops to the detectors
3. Set operating frequencies of the detectors so that there are no crosstalks between them (Rule of thumb: The closer two loops, the bigger frequency span). Set appropriate sensitivity of each detector.
4. Plug PC serial line in Con4
5. Attach +-15V DC power supply to Con1
6. Configure the device according to paragraph 5)

II. GPRS Modem Configuration

The modem interface is configured in default: **115200, 8N1, RTS/CTS**. This has to be changed to: **19200, 8N1, No handshake**. In order to reconfigure the modem interface, connect the modem to PC terminal and issue the following:

```
AT+IPR=19200\r\n
```

```
AT+IFC=0,0\r\n
```

And write out the new settings: *AT&V\r\n*, and reconfigure your PC terminal according to those new settings. Change TCP stack settings:

```
AT$HOSTIF=2\r\n
```

```
AT$ACTIVE=1\r\n
```

Subsequently, set GPRS properties according to your GSM provider (*internet* is an example):

```
AT+CGDCONT=1,"IP","internet"\r\n
```

And write out the settings to modem memory:

```
AT&W\r\n
```

5) Configuration

The device is configured via its PC serial line. Connect a PC to Con4, and set the following in PC's terminal application: **115200, 8N1, no handshake**. Once a device is powered on, a message "*Traffic Counter*" is sent to the PC, and the user can communicate with the device via commands.

A command consists of 2 parts – *Text* and *Value*. If *Value* is not present, it is considered as reading. Otherwise, the value is processed by the device and saved into configuration. A list of available commands can be found in **Appendix 10**. The following 3 paragraphs summarize basic configuration.

I. Managing Configuration and Device

Command	Description
LOADDEF	Loads factory defaults (always use for new devices)
SHOWSET	Shows current configuration
READ	Reads configuration from EEPROM memory
WRITE	Writes current configuration to EEPROM memory
DT	Read/Write date and time
RESTART	Restarts the device

Note: Do not forget to issue *WRITE* command if you have changed any device parameter!

II. Measurement Configuration

The following commands have to be used in order to configure the device for taking vehicle samples:

1. Set Lane Number, Loop Length and Distance between loops:
Commands: *LANENUM, LOOPLEN, LOOPDIST*
2. Start or stop measurement:
Commands: *START, STOP, AUTOSTART*
3. Listen to measured vehicles (samples will be sent to the PC)
Commands : *LISTEN, NOLISTEN*
4. Check vehicle buffer:
Commands: *VEHCOUNT, SHOWVEH, CLEARVEH*

III. Server Upload

The following should be configured for proper data upload to the server:

1. Station ID, Lane Number (for appropriate server database upload)

Commands: *SID, LANENUM*

2. Server IP address and Port:

Commands: *SERVERIP, SERVERPORT*

3. Enable GPRS transfer, Time period, SIM card PIN code:

Commands: *GPRSEND, GPRSTIME, GPRSPIN*

4. Try to connect to the server

Commands: *GPRSNOW*

Appendix 10 Traffic Counter: List of Commands

Command	Description
Commands without write values	
LISTEN	Vehicle samples will be sent to the PC for 5 minutes
NOLISTEN	Stop listening vehicle samples
VEHCOUNT	Show number of vehicle samples in buffer
UPTIME	Show number of minutes of device uptime
RESTART	Restart the device
RESETDET	Reset Inductive Loop Detectors
LOADDEF	Load default settings
READ	Read settings from EEPROM memory
WRITE	Write settings to EEPROM memory
SHOWSET	Show actual settings
SHOWVEH	Show the content of vehicle buffer
CLEARVEH	Clear vehicle buffer
SHOWTEMP	Show content of temperature buffer
CLEARTEMP	Clear temperature buffer
GETTEMP	Show current indoor and outdoor temperature
JOKE	Easter Egg
VER	Show network protocol, firmware and hardware versions
HELP	Show list of available commands
GPRSNOW	Connect and transfer buffers (veh., temp.) to the server
GPRSDISCON	Close and reset GPRS modem
START	Begin measurement
STOP	Stop measurement
Commands for Configuration (read/write value)	
AUTOSTART	Start Measurement after power-up 0=no, 1=yes

DT	Set/Get actual date and time (set command without seconds) format: <i>YYMMDDHHmm</i> ex. 0812301545 = 2009-12-30 15:45:00
MEASUREAVG	Measure speed and length by averaging 0=no, 1=yes
GPRSEND	Transfer samples to the server 0=no, 1=yes
GPRSTIME	Time interval of transferring [minutes]
GPRSPIN	PIN code for a SIM card in the GPRS modem 0 for no PIN
LOOPDIST	Distance between loops [dm]
LOOPLEN	Length of loop [dm]
LANENUM	Lane offset – LaneNumber of the first lane LaneNumber+1 of the second lane
SERVERIP	IP address of the server [xxx.xxx.xxx.xxx]
SERVERPORT	Server port [0-65535]
TEMPTIME	How often to get temperature samples [minutes, 0=don't take any samples]
SID	Station ID

Appendix 11 Database Create Script

```

--
-- Diagram Name: ERmodel
-- Created on: 25.12.2009 17:02:54
--
SET FOREIGN_KEY_CHECKS=0;
-- Drop table stations
DROP TABLE IF EXISTS `stations`;

CREATE TABLE `stations` (
  `idstation` mediumint(8) UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` varchar(30) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `description` tinytext CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL
COMMENT 'description',
  `lat` float(10,6),
  `lng` float(10,6),
  `created` datetime NOT NULL,
  `lastdataupdate` datetime COMMENT 'last data update',
  `active` tinyint(1) NOT NULL DEFAULT '1',
  `deactivated` datetime,
  PRIMARY KEY(`idstation`)
)
ENGINE=MYISAM
ROW_FORMAT=dynamic;

-- Drop table uploadedfiles
DROP TABLE IF EXISTS `uploadedfiles`;

CREATE TABLE `uploadedfiles` (
  `idfile` mediumint(8) UNSIGNED NOT NULL AUTO_INCREMENT,
  `idstation` mediumint(8) UNSIGNED NOT NULL,
  `filename` varchar(30) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `dacq` date COMMENT 'date of acquisition',
  `dtfrom` datetime NOT NULL COMMENT 'dt of the first sample of the file',
  `dtto` datetime NOT NULL COMMENT 'dt of the last sample of the file',
  `pver` varchar(5) CHARACTER SET utf8 COLLATE utf8_general_ci COMMENT 'network
protocol version',
  `fwver` varchar(5) CHARACTER SET utf8 COLLATE utf8_general_ci,
  `hwver` varchar(5) CHARACTER SET utf8 COLLATE utf8_general_ci,
  `dtuploaded` datetime NOT NULL,
  `vehiclecount` mediumint(10) UNSIGNED COMMENT 'how many vehicle samples',
  `temperaturecount` smallint(5) UNSIGNED COMMENT 'how many temperature samples',
  PRIMARY KEY(`idfile`)
)
ENGINE=MYISAM
ROW_FORMAT=dynamic;

-- Drop table lanes
DROP TABLE IF EXISTS `lanes`;

CREATE TABLE `lanes` (
  `idlane` mediumint(8) UNSIGNED NOT NULL AUTO_INCREMENT,
  `idstation` mediumint(8) UNSIGNED NOT NULL,
  `name` varchar(30) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `description` tinytext CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `lanenumber` tinyint(3) UNSIGNED NOT NULL,
  `direction` tinyint(1) NOT NULL DEFAULT '0' COMMENT 'going towards what',
  `speedlimit` tinyint(3) UNSIGNED COMMENT 'maximal speed in a lane in kmph',
  `created` datetime NOT NULL,
  `active` tinyint(1) NOT NULL DEFAULT '1',
  `deactivated` datetime,
  `lastdataupdate` datetime,
  PRIMARY KEY(`idlane`)
)

```

```

ENGINE=MYISAM
ROW_FORMAT=dynamic;

-- Drop table temperatures
DROP TABLE IF EXISTS `temperatures`;

CREATE TABLE `temperatures` (
  `idtemp` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT,
  `idstation` mediumint(8) UNSIGNED NOT NULL,
  `dt` datetime NOT NULL,
  `tempin` tinyint(4) NOT NULL COMMENT 'indoor temperature',
  `tempout` tinyint(4) NOT NULL COMMENT 'outdoor temperature',
  PRIMARY KEY(`idtemp`)
)
ENGINE=MYISAM
ROW_FORMAT=fixed;

-- Drop table trafficdata
DROP TABLE IF EXISTS `trafficdata`;

CREATE TABLE `trafficdata` (
  `idtraffic` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT,
  `idlane` mediumint(8) UNSIGNED NOT NULL,
  `dt` datetime NOT NULL,
  `datecol` date NOT NULL COMMENT 'contains date of dt',
  `speed` tinyint(3) UNSIGNED NOT NULL COMMENT 'kmph',
  `length` smallint(5) UNSIGNED NOT NULL COMMENT 'in 10s of cm',
  PRIMARY KEY(`idtraffic`),
  INDEX `idlane`(`idlane`, `datecol`),
  INDEX `idlane_2`(`idlane`, `dt`)
)
ENGINE=MYISAM
ROW_FORMAT=fixed;

SET FOREIGN_KEY_CHECKS=1;
-- Drop View TrafficOverview
DROP VIEW IF EXISTS `TrafficOverview`;
CREATE VIEW `TrafficOverview` AS
SELECT stations.idstation, stations.name, stations.lastdataupdate, (SELECT
ROUND(AVG(tempout)) FROM temperatures WHERE dt BETWEEN (NOW()-INTERVAL 1 HOUR)
AND NOW()) AS tempout,
(SELECT ROUND(AVG(tempin)) FROM temperatures WHERE
temperatures.idstation=stations.idstation AND dt BETWEEN (NOW()-INTERVAL 1 HOUR)
AND NOW()) AS tempin,
(SELECT COUNT(*) FROM trafficdata LEFT JOIN lanes USING (idlane) WHERE
lanes.idstation=stations.idstation AND dt BETWEEN (NOW()-INTERVAL 1 DAY) AND
NOW()) AS vehday,
(SELECT COUNT(*) FROM trafficdata LEFT JOIN lanes USING (idlane) WHERE
lanes.idstation=stations.idstation AND dt BETWEEN (NOW()-INTERVAL 1 HOUR) AND
NOW()) AS vehhour
FROM stations ORDER BY stations.name
;

```

Appendix 12 Example Source Code for CCS C Compiler

```
#include <18f2455.h>
#device ADC=10
#fuses HSPLL,PLL5,CPUDIV3,NOVREGEN,NOWDT,NOPROTECT,
NOLVP,NODEBUG,NOPBADEN,WRTB,MCLR,NOCPD,NOWRTC
#use fast_io(B)
#use delay(clock=24000000)
#use rs232(baud=115200,xmit=PIN_C6,rcv=PIN_C7,STREAM=RS232)
#define TICKS_PER_SECOND 732 //how often per second the timerirq routine
is called
unsigned int16 timing = 0;
#INT_rtcc
void timerirq()
{
    if (++timing == TICKS_PER_SECOND)
    {
        timing = 0;
        output_toggle(PIN_B0);
        if(input(PIN_B0))
        {
            //set_pwm1_duty(0);
        }
        else
        {
            //set_pwm1_duty(1023);
        }
    }
}
#INT_ext1
void extint()
{
    printf ("extint\n");
}
void main()
{
    char c;
    fprintf(RS232, "load\r\n");
    set_tris_b(2);
    output_low(PIN_B0);
    ext_int_edge(1, L_TO_H);
    setup_ccpl(CCP_PWM);
    setup_timer_2(T2_DIV_BY_16, 256, 16);
    set_pwm1_duty(512);
    set_timer0(0);
    setup_timer_0(RTCC_INTERNAL|RTCC_DIV_32|RTCC_8_BIT);
    enable_interrupts(INT_TIMER0);
    enable_interrupts(INT_EXT1);
    enable_interrupts(GLOBAL);

    while (1)
    {
        if(kbhit())
        {
            c = getc();
            printf("echo: %c\r\n", c);
            if(c=='0')set_pwm1_duty(0);
            else if(c=='1')set_pwm1_duty(10);
            else if(c=='2')set_pwm1_duty(1023);
        }
    }
}
```