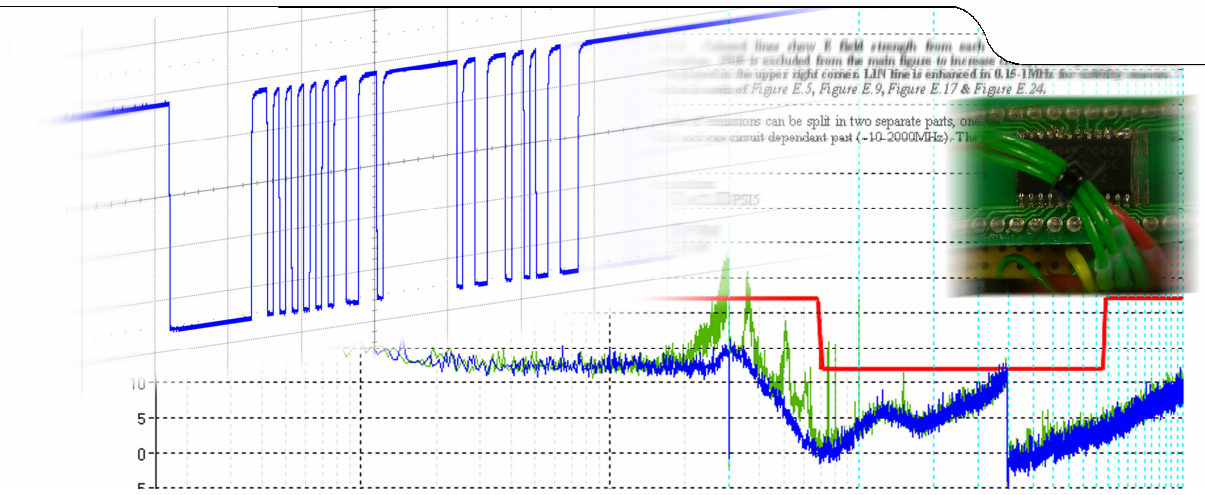
CHALMERS

MÄLARDALEN UNIVERSITY



Sensor communication characteristics

a comparison of LIN, PSI5, ACL and PWM

Master of Science Thesis

Andreas Englund
Department of Energy & Environment
Elektrotekniklinjen 180p
CHALMERS UNIVERSITY OF TECHNOLOGY

Peter Larsson
Department of Computer Science and Electronics
Datateknik och elektronik 180p
MÄLARDALEN UNIVERSITY

Volvo Technology

Göteborg 2006

Abstract

The thesis report presents a technical comparison of four different methods for sensor communication in vehicles. Local Interconnect Network (LIN), Peripheral Sensor Interface (PSI5), Analog Current Loop (ACL) and Pulse Width Modulation (PWM) have been investigated.

The focus of the study is EMC characteristics, where radiated emission and susceptibility are investigated in a laboratory setup. Included in the thesis are implementation and documentation of a functional physical test system, much suitable for further investigation of the four studied communication methods as well as rapid implementation of other methods.

The results indicate substantial differences between LIN, PSI5, ACL and PWM both in terms of basic functionality and electromagnetic behavior. Best in the EMC tests was PWM. Of special interest are indications that communication circuitry might possibly be a bigger problem than the actual communication.

Sammanfattning

Detta examensarbete presenterar en teknisk jämförelse mellan fyra olika metoder för sensorkommunikation i fordon. Teknikerna som har undersökts är ”Local Interconnect Network” (LIN), ”Peripheral Sensor Interface” (PSI5), analog strömslinga (ACL) och pulsviddsmodulering (PWM).

Tyngdpunkten i studien ligger på EMC, där strålade emissioner och känslighet för strålning har undersökts i en testmiljö. I examensarbetet ingår konstruktion och dokumentation av ett fungerande testsystem, väl anpassat för ytterligare undersökningar av de fyra kommunikationsmetoderna liksom för en snabb implementering av andra metoder.

Resultaten från undersökningarna ger markanta skillnader mellan LIN, PSI5, ACL och PWM, både när det gäller tekniska egenskaper och EMC. PWM utmärker sig positivt i EMC-testerna. Av speciellt intresse är indikationer på att kretsarna som sköter kommunikationen eventuellt är ett större EMC-problem än störningar från kommunikationen.

Preface

During our work at Volvo Technology we have had plentiful of support and guidance from our supervisors Magnus Granström, Eilert Johansson and Catharina Levinsson for which we are very grateful.

We would like to thank Kenneth Alklind and Ulf Herbertsson at Volvo 3P EMC laboratory for all the help and patience with our numerous questions.

Torbjörn Thiringer at CTH was always ready to help us out on short notice, and provided interesting and rewarding discussions during the work. Thank you for all the support.

This thesis was an initiative of the TC Sensor, a team of experts from several Volvo companies: Trucks, Buses, Aero, Construction Equipment, Penta, Powertrain and 3P. We are grateful your trust to do this job.

Your help was most appreciated.

Andreas Englund
Peter Larsson

Table of contents

Abbreviations

µC	Micro Controller, in this report Freescale STAR12
ACL	Analog Current Loop, 4-20mA
ATD	Analog To Digital converter
B field	Magnetic field
BDM	Background Debug Module
DMTU	Dual Master Translation Unit, RS-232 voltage converter
E field	Electric field
EMC	Electro Magnetic Compatibility
I/O	digital Input/Output
IC	Integrated Circuit
LP	Low pass (filter)
PAS	Peripheral Acceleration Sensors, used in PSI5 systems
PCB	Printed circuit board
PDU	Power Distribution Unit
PSI5	Peripheral Sensor Interface
SCI	Serial Communication Interface
SPI	Serial Peripheral Interface

1. Introduction

1.1. Background

All modern vehicles include complex electronic systems. A lot of these are communication systems, carriers of information from one part to another in a vehicle. As the demands for environmental friendliness, quality and cost effectiveness increases, so must the electronic system's ability to solve the new situations increase. This is directly related to sensors, as more information must be collected at many positions inside and outside the vehicle. To carry all the information required, the benefits of a more intelligent communication system than those presently in service are obvious:

1. Less cable harnesses. Desirable because of:
    a. The lower number of wires used, the lower the costs.
    b. Easier assembling in vehicles.
    c. Less space requirements.
    d. Weight reduction, less fuel is consumed by the vehicle.
2. Analog voltage signals are not as reliable as desired, especially not over great distances or in some difficult areas of the vehicles.
3. Unreliable technology is expensive due to the need of more workshop visits. In addition to this, the fault detection is difficult in traditional electronic systems.

In the development of future generations of vehicles, it is important to take these aspects into consideration. Accordingly, there is a need for a compilation - a report - which can assist management, designers, engineers and others in the process of choosing which communication method should be used for a specific task, and which should not.

Initially, the thesis required different skills, such as understanding of electronics, architecture, programming, EMC and communications. The thesis students were both finishing senior year at different universities. They both had the same master, electronics, but with different specialization. The educations provided the thesis team with tools and experience how to get necessary knowledge to complete the project.

1.2. Related work

The automotive industry has used sensor systems for a long time. Therefore many methods and standards have been introduced for various purposes, with varying success. The methods frequently used (e.g. analog voltage) are well documented and there exist extensive know-how. There is, however a need of more perspective expertise to be able to choose wisely when to use the different techniques. The information available in separate organizations was demanded.

The first task to select which areas to focus on was done by the Technical Committee Sensor at Volvo AB. The committee chose LIN, PSI5, ACL and PWM for further investigation and comparison. Some aspects as cost and availability were investigated outside the scope of this thesis.

1.3. Thesis formulation

The thesis work should be performed as a project work to fit in the structure of Volvo's project organization. Two thesis students should work 20 weeks, 40 hours per week. Listed below are the essential moments of the project:

- Study LIN, PSI5, ACL and PWM in detail.
- Implement the communication methods mentioned above.
- Expose the test setup for EMC tests.
- Evaluate and analyze test results.
- Technical comparison.
- Report and present outcome of the project.

1.4. Delimitations

The thesis project is performed within the field of sensor communication methods, but with a number of strict delimitations. The reason to delimit the work according to the list below is to define in detail what is included and what may be expected of the project.

1. The thesis should be limited to studies within the communication methods:
    a. LIN
    b. PSI5
    c. ACL
    d. PWM
2. Wire based communication.
3. Only technical aspects should be studied.

1.5. Purpose

Future vehicles will require more sensor information and information transmissions. The increase in complexity, costs and space requirement must be matched and mastered by better design principles. Still the designs will be limited, to be able to offer high quality products to affordable prices to the customer. To be able to decide which communication method(s) should be used, a fair comparison is required. Most fair is here achieved by following standards and specifications for each method: how each method is intended to operate with basic configuration, as recommended by manufacturers.

2. Method

2.1. Analysis of problem

In this section an analysis of the premises after information gathering are explained in detail. The list in 1.3 is expanded.

- Study LIN, PSI5, ACL and PWM in detail.
  - LIN
    - Established consortium responsible, specification available.
    - Circuit information available.
    - Software available but not adapted to this hardware.
    - Hardware available and requires minor modifications.
  - PSI5
    - Bosch responsible for standard, specification available.
    - E-mail contact established for technical issues with Bosch.
    - Circuit information not extensive, but sufficient.
    - Software not available but needed.
    - Hardware components available but not assembled.
  - ACL
    - Several used methods available, analog 4-20mA current loop chosen.
    - Software not available but needed for this implementation.
    - Hardware components available from electronics supplier can be ordered.
  - PWM
    - No standard available, communication definition needed.
    - Software not available but needed for this implementation.
    - Hardware available and requires minor modifications.
- Implement the communication methods mentioned above.
  - Extensive hardware design and assembling required.
  - Component purchase and gathering required.
  - Controlling PDU.
    - Environment familiarization.
    - Programming.
    - Debugging.
- Expose the test setup for EMC tests.
  - Study test routines.
  - Plan tests.
    - Find facilities.
    - Schedule synchronization with laboratory.
    - Visit facilities.
    - Hardware preparation.
    - Software preparation.
  - Perform tests.
- Evaluate and analyze test results.
- Technical comparison.
- Report and present outcome of the project.

2.2. Solution – a brief implementation overview

For each of the methods that should be tested, the test setup should be designed as a small, fully functional and representative system. The system should consist of one master and one slave unit, where the slave simulates a sensor. The system should be a common platform, capable of communicating through LIN, PSI5, ACL and PWM. To achieve this, a solution consisting of PDU's and replaceable adapter modules for each communication method is chosen.

3. Communication in vehicles

3.1. Communication methods

LIN

Local Interconnect Network (LIN) is a communication method developed by an industrial group, consisting mainly of companies from the automotive industry. It is used for two-way communications with sensors and other electronic devices in vehicles. [1]

The LIN system is divided into sub-groups for the purpose of making it easy to work with. The three are:

1. LIN Physical layer.
2. LIN Protocol layer.
3. LIN Application layer.

Some of the main features of LIN are that it is made to be cheap to implement and manufacture, and that it should support a number of nodes (e.g. sensors) in a single network, and thereby reducing harnessing, weight, complexity and costs.

All communication is transmitted in one wire. The master uses an identifier to address one or several nodes in the network which then respond, refer to *Figure 3.1, Figure 3.2 & Figure 3.3.* All nodes attach a calculated checksum byte last in the response, so the master can verify correct transmission. The control of the bus communication is performed by the master unit, which has advanced software for different scenarios. The information is transferred in the wire as digital voltage levels, high or low states.

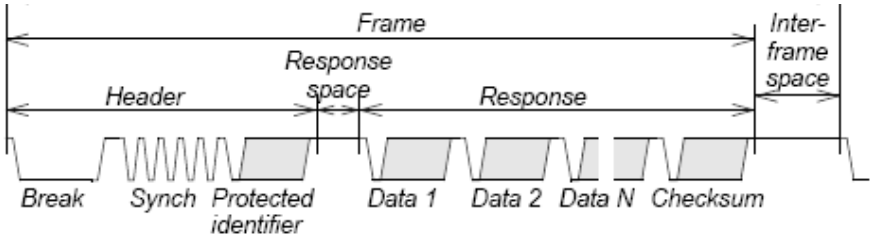


Figure 3.1. The principal structure of a LIN frame.

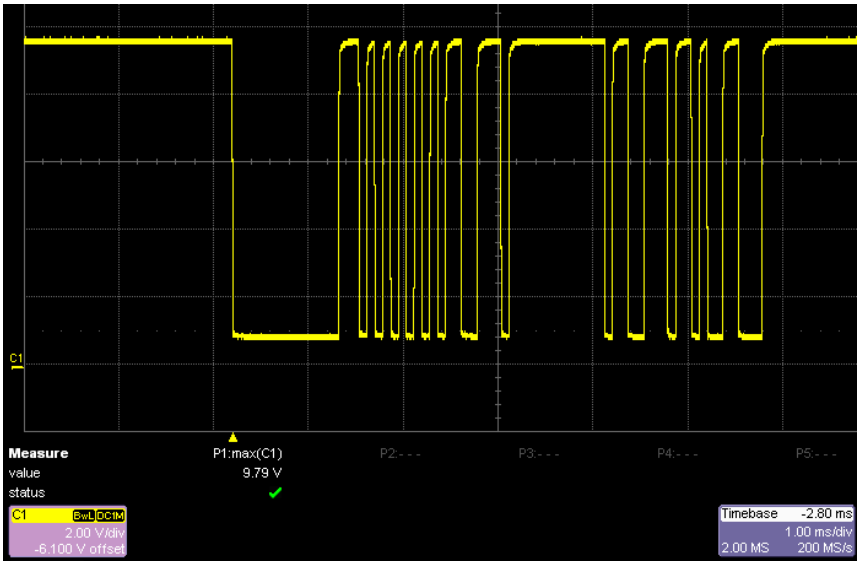


Figure 3.2. The structure of a LIN frame as presented by oscilloscope.

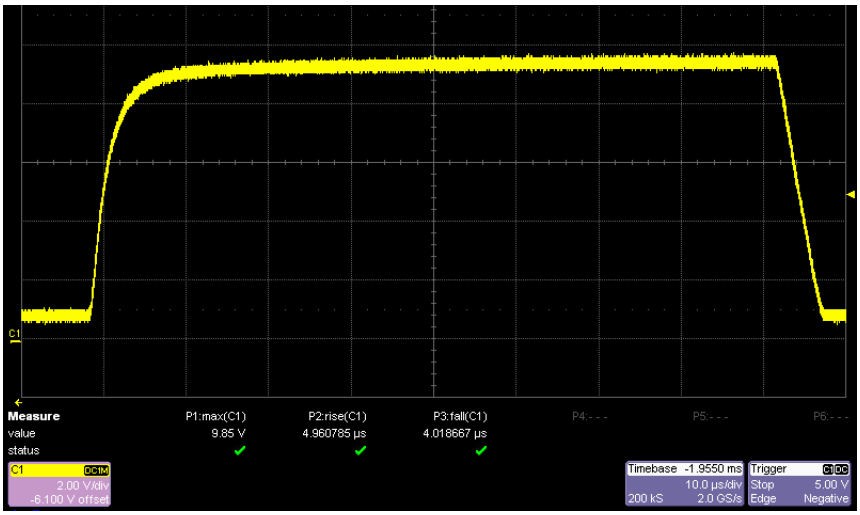


Figure 3.3. Enhanced view of a single LIN bit.

☐ *PSI5*

The Peripheral Sensor Interface (PSI5) is an open standard, developed by Autoliv, Bosch and Continental Temic, to meet the specific requirements of airbag systems [2]. It is widely used in the automotive industry today. The system operates by a two-wire, digital current interface, where the power to the sensors is distributed on the same wire as the sensors data output. Below is a brief description of how a simple PSI5 system operates.

A current I_{low} (refer to *Figure 3.4, Figure 3.5 & Figure 3.6*) of typically 10mA originating from the master is driven through the circuit. The sensor node uses the current to power its internal circuits. When a measurement is done by the sensor, the information is translated into data.

The data is Manchester coded, which gives the benefit of constant average power consumption regardless of the information transmitted. The data is transmitted to the master by altering the current in the system. The altering current, which varies about 20 mA from I_{high} to I_{low} is decoded by the master. Refer to *Figure 3.4* for an example of Manchester coding.

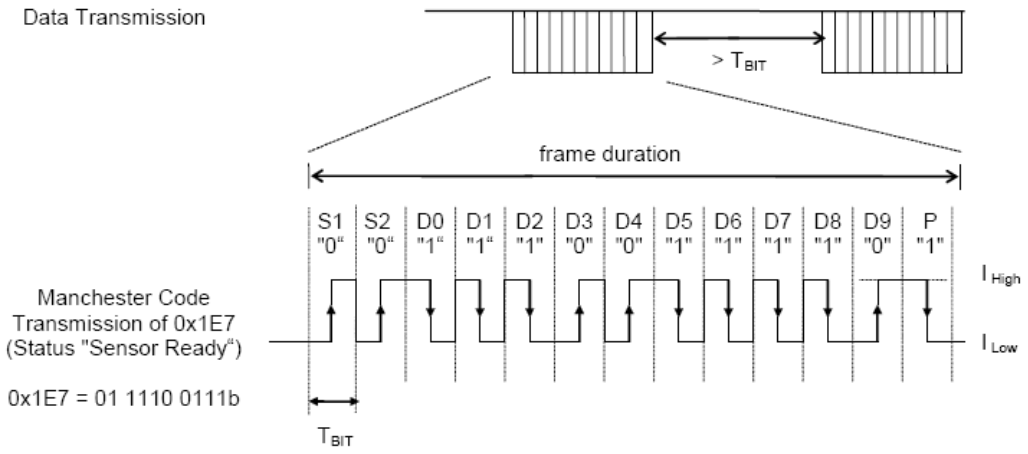


Figure 3.4. PSI5 Manchester coded data frame with 10 data bits (D0-D9), two start bits (S1, S2) and one parity bit (P).

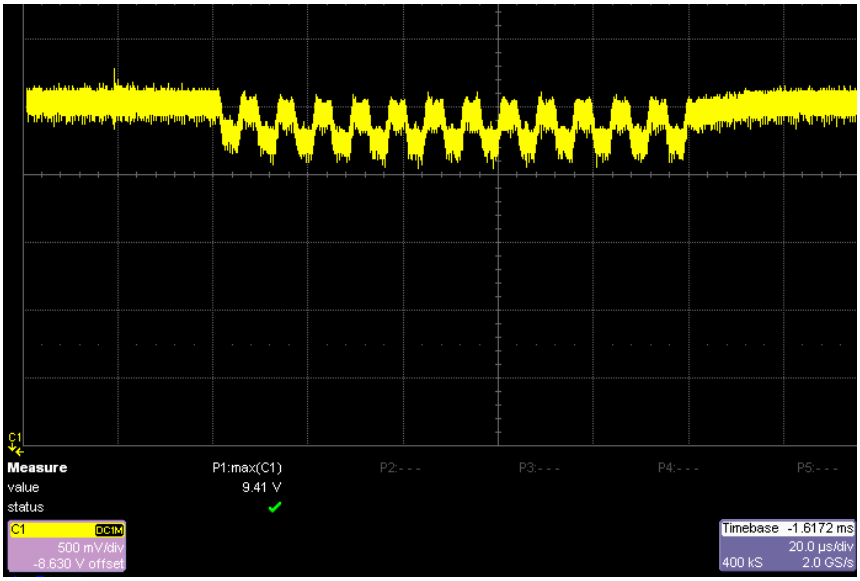


Figure 3.5. PSI5 bits are current encoded, but also visible as voltage levels. Note the voltage scaling and offset.

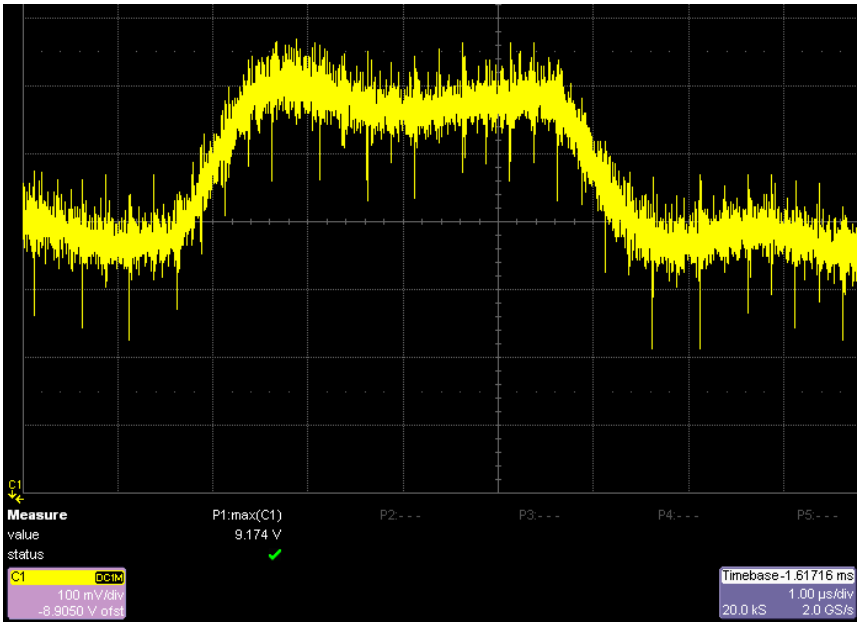


Figure 3.6. Enhanced view of a PSI5 single bit. The voltage dips occur with a separation of 0.5µs, which is equal to the frequency of the PSI5 clock.

ACL

Analog current loop (ACL) is a current based communication method, frequently used in process industry. The concept of operation is a closed 12V current loop with a slave unit altering the current (4-20mA), which is measured by the master unit. If the slave unit is a sensor, the sensor data is converted to a current level in the circuit.

For example, it is possible to let a reading at 19% of a sensors total scale to be transmitted by ACL through setting the current to 7,04mA which is 19% of ACL full scale (4-20mA).

PWM

PWM is short form of “Pulse Width Modulation”. There are no wide accepted standardized methods for this type of communication, and it is therefore free to implement as needed, refer to *Figure 3.7 & Figure 3.8* for an example of PWM communication. [3]

The idea of PWM is to use a single line on which the voltage is altered between high and low states. The length of every period, T_p, within a system is constant. Information is transmitted by altering the time the signal is high. In PWM, a period is followed immediately by another one.

For example, it is possible to let a reading at 30% of a sensors total scale to be transmitted by PWM, by setting 30% of T_p to high state. This period is then followed by another identical one, until a new reading of the sensor is ready for transmission. Digitally, the quote between the time periods $T_{high\ state}$ / $T_{low\ state}$ is equal to the information transmitted. Analogically, the quote between the electrical powers $P_{transmitted}$ / $P_{maximum}$ is equal to the information transmitted. Since any given PWM-signal can be handled digitally or analogically, the two methods can be used within one single system.

The transmission speed of PWM is defined in Hz, and should be adapted to the environment and requirements of the system.

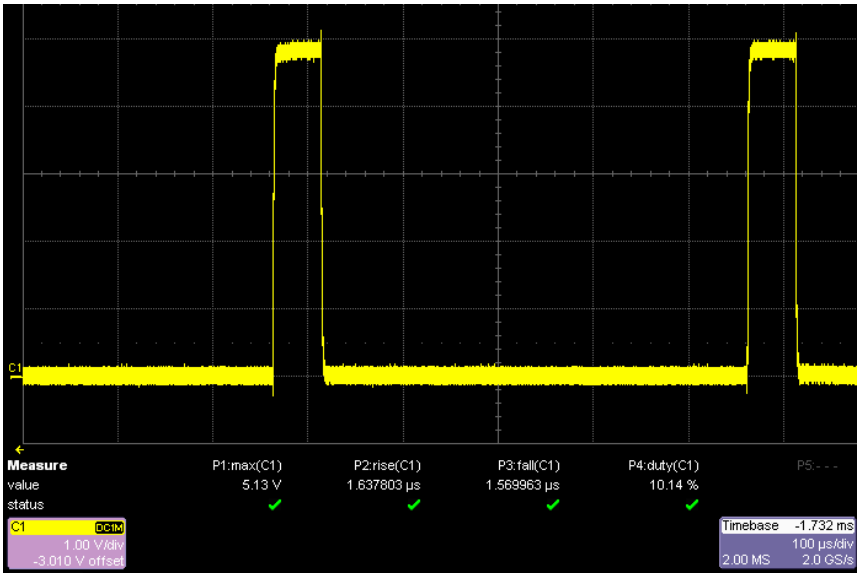


Figure 3.7. PWM periods, 2kHz.

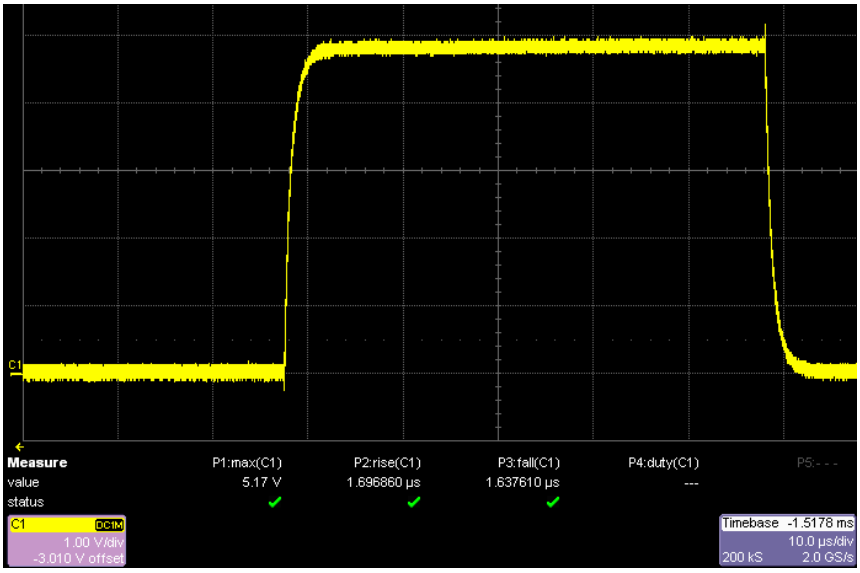


Figure 3.8. Enhanced view of edges in PWM signal.

3.2. Feature comparisons

For an overview of LIN, PSI5, ACL and PWM, refer to *Table 3.1*.

Table 3.1. Features of the communication systems.

Feature		LIN	PSI5	ACL	PWM
No. of wires required to sensor (incl. power & ground cables)	One-way comm.	3	2	2	3
	Two-way comm.	3	*	-	4
Analog/digital		Digital	Digital	Analog	Analog/digital
Electrical interface of communication		Voltage	Current	Current	Voltage
Number of slave nodes in each circuit		1-15	1-many**	1	1
Message transmission frequency approx.		<250Hz	~4.4kHz	>2kHz	>2kHz
Bit rate		20kbps	125kbps	-	-
Transmission distance		40m	Few meters	>>40m	>40m
Supply voltage level typical		12/24V	12/24V	24V	5-24V
Sensor diagnostic possibility		Yes	No	No	No
Verification of correct transmission		Checksum	Parity bit	-	-
Requires µC		Yes	Yes	No	No

* In special configurations, two-way communication can be possible.
** Limited by requirements on information flow and electrical characteristics.

4. Electro magnetic compatibility

4.1. Emissions and susceptibility

In an electrical system where currents and voltages are changing, electric (E field) and magnetic (B field) fields are created. The E field is proportional to voltage and the B field is proportional to current. The field's frequency spectrum is proportional to the frequency spectrum of the voltage or current it origin from [6]. This is called emissions.

In the reversed way a current or voltage can be induced into a conductor by a field. A system's ability not to be affected by fields is called susceptibility.

There are Volvo regulations that define which emission and susceptibility behavior are accepted and which are not [7]. This is important for safety, technical and legal reasons: equipment must not disturb or be disturbed by other equipment, and must not be harmful to living biological tissue.

4.2. EMC Laboratory

The laboratory used for testing in this project is located in Volvo 3P facilities at O2, Lundby. It is used for EMC testing within AB Volvo and was at dispose for one week. During all times it was operated by staff, which contributed with extensive knowledge and experience in EMC testing. Used hardware is listed in *Table 4.1* and shown in *Figure 4.1*.

Table 4.1. Equipment used in EMC laboratory.

Description	ID
Spectrum analyzer, HP 8593	EMC 60389
Bilog antenna, Schaffner	ANT 60330
Logper. antenna,	ANT 60104
Active monopole antenna, EMCO 3301B	ANT 60097
Amplifier, HP 8447D	EMC 60116
Amplifier, Miteq	EMC 60445
Tektronix 2232	KOS 60466
Schaffner NSG 5000	EMC 60435
Power supply, SCR	LAG 60044
Power supply, Delta elektronika	LAG 60448

Description	ID
Signal generator, R&S	EMC 60130
Multimeter, Tektronix TX3	UID 60435
Crank pulse generator	EMC 60183
Oscilloscope, Tektronix THS 720A	KOS 60466
Millivoltmeter, R&S URV5	EMC 60086
Amplifier, MPD	EMC 60135
Amplifier, ENI	EMC 60098
Directional coupler	EMC 60006
Directional coupler	EMC 60182
BCI probe Schaffner	EMC 60452

Figure 4.1. Left: Laboratory equipment. Right: Inside test chamber.

Each of the listed equipment has a limited frequency range and therefore changes between different setups were necessary. During emission frequency scans the antenna was changed at 30MHz, and the amplifier at 400MHz. The latter is visible as an amplitude drop of approximately 12dB originating from different amplifiers noise characteristics, refer to *Figure 4.2*.

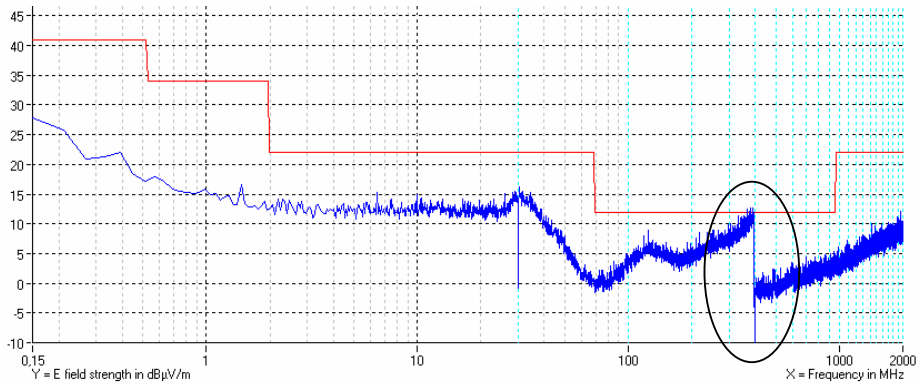


Figure 4.2. The marking shows the 12dB drop in amplifier noise.

5. Implementation

5.1. Communication hardware

For detailed information about electronic parts of the test setup system refer to *Appendix – Schematics.*

5.1.1. Boxes

For reliable laboratory results, it was desirable to achieve appropriate shielding towards E fields for two reasons.

- For emission testing only emissions originating from inter box cabling (refer to *Figure 5.1*) were of interest.
- For susceptibility testing only effects of interference introduced in inter box cabling were desirable to study.

To shield electronics high quality zinc boxes were used. The number of openings (holes) in shielding was kept to a minimum.

Required power supply for electronics was arranged in each box by battery packs. Each battery pack contained nine standard AA battery providing 13.5V. Inter box cabling consists of two wires, red and black. Each cable was a 0.81mm^2 (16 leads) stranded tinned copper wire with PVC coating.

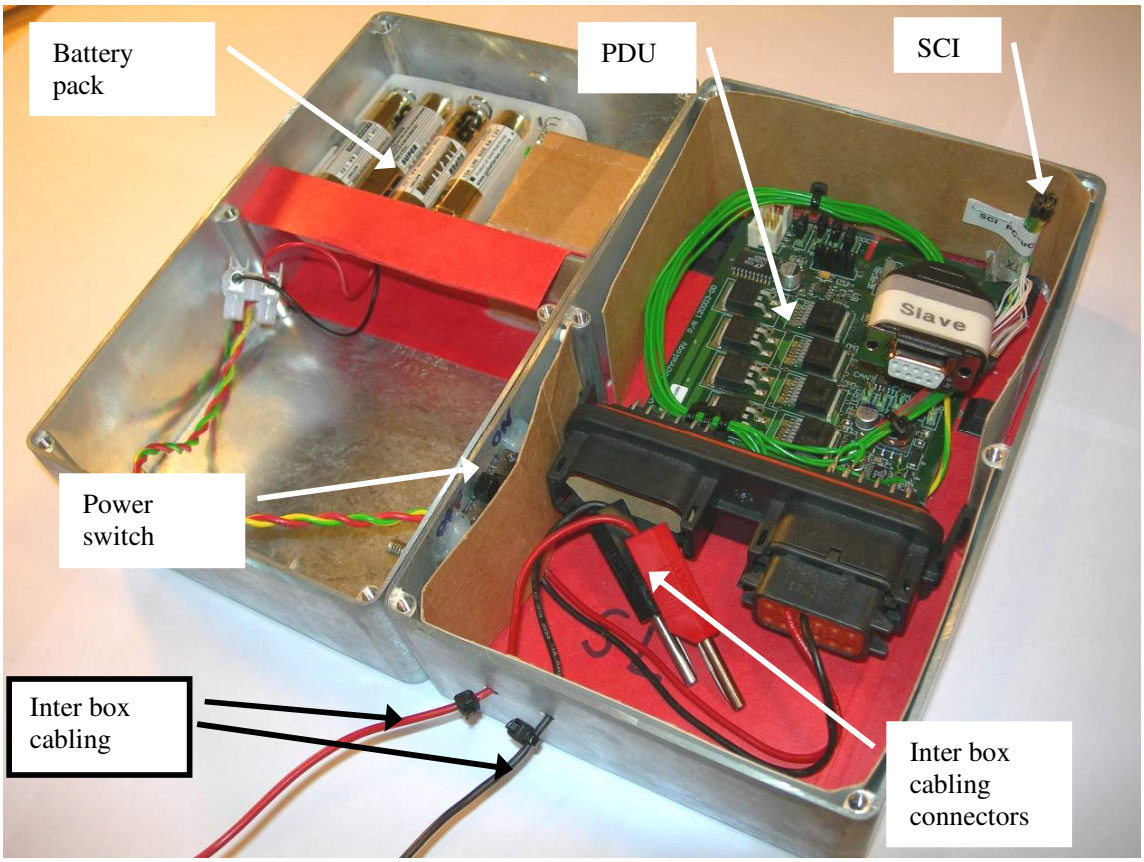


Figure 5.1. Picture shows major parts inside slave box.

5.1.2. PDU's

One PDU (refer to *Figure 5.2*) was used in each box. A PDU is a professionally manufactured circuit board designed for test purposes. It is designed for various applications of use, this project mainly used:

1. µC, Freescale STAR12, versatile 16MHz processor.
2. BDM connector for programming and debugging. Some modifications were necessary, refer to PDU documentation.
3. DC/DC converter with internal voltage regulation.
4. LIN, circuits required for LIN communication.

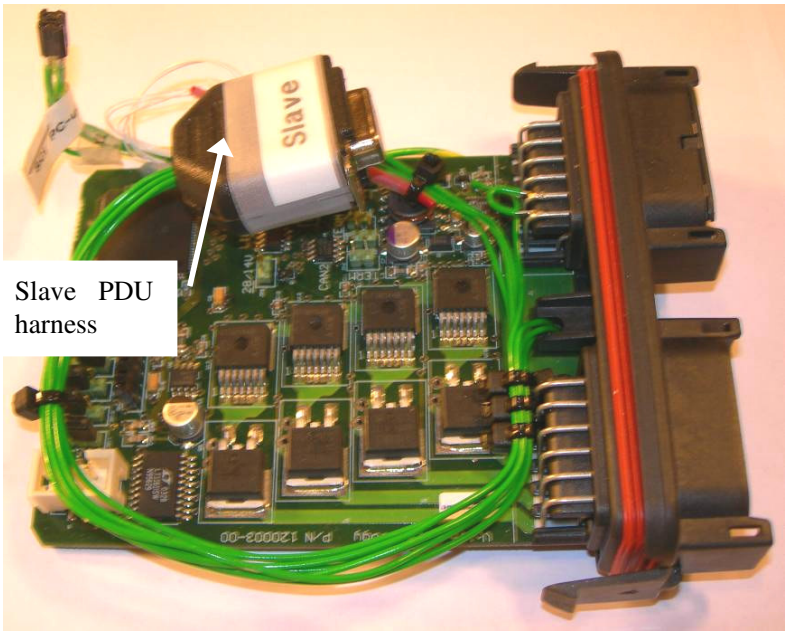


Figure 5.2. A PDU.

In addition to mentioned features both PDU's were equipped with SCI connectors to allow connection to PC for software purposes. The master PDU was further upgraded with a stop switch and a red LED. During laboratory tests two problems were known but considered unimportant for test outcome:

1. µC malfunction in ATD0 module. Resolved by use of ATD1.
2. µC malfunction in Chip select pin of SPI0 module. Resolved by use of port A output pin.

5.1.3. Communication specific hardware

LIN implementation used Infineon TLE 6259 for the necessary signal translation from standard SCI to LIN bus. The IC was already implemented on the PDU's. No hardware modification was required. Two adapter harnesses were used to connect PDU's to inter box cabling.

PSI5 implementation required additional hardware. One adapter module was built with Bosch CG974 and IQD IQEXO-3 and served as link between master PDU and inter box cabling. CG974 is controlled by a microcontroller's SPI interface and translates the PAS sensor information to SPI. PSI5 slave was replaced with a real sensor, Bosch PAS4 SMB180.

ACL implementation required additional hardware. The master adapter module was built with Burr Brown RCV420. The IC transmits ACL bus information to analog 0-5V connected to µC ATD module. ACL slave module was built with Burr Brown XTR115 transceiver. Maxim MAX522CPA converts digital output from µC to analog input on XTR115.

PWM implementation used µC's I/O module as receiver (master) and transmitter (slave) to create a PWM bus. No hardware modification was required. Two adapter harnesses, each with appropriate protective circuitry were used to connect PDU's to inter box cabling.

5.2. Communication software

All numbers occurring in this section are decimal if not labeled differently.

All code used in master and slave (refer to *Appendix – C-code for master PDU*, and *Appendix – C-code for slave PDU*) was programmed in ANSI C, and internally arranged as modules for each of the four communication methods. Refer to *Table 5.1* for a function overview.

Some key properties of the software:

- LIN was implemented for 12,1kb/s, 12V.
- PSI5 was implemented for PAS4, 125kbaud, and SPI communications to PDU at 4MHz.
- ACL was implemented with current level update frequency and current readings at 2kHz.
- PWM was implemented with signal generation and readings at 2kHz.

For software development, debugging and in-software navigation, the following resources were used:

1. C-compiler and project builder: IAR Embedded Workbench, User Interface Version: 2.31D, IAR Workbench target descriptor for M68HC12 Version 2.43A/WIN.
2. Processor debugging software: Trace32 ICD HC12 S12 USB.
3. Accompanying processor debugging hardware: Lauterbach Power debug interface / USB, Lauterbach PDM-adapter.
4. SCI-input/output handler: Terminal Version 1.9b. Settings: Baud rate = 1200, 8 data bits, no parity, 1 stop bit, no handshaking.

Table 5.1. Overview of master and slave software. Arrows indicate which functions can interact to each other. The function numbering in the table is used in the C-code. Refer to *Appendix – C-code for master PDU* and *Appendix – C-code for slave PDU*.

Master			**Slave**	
C function #	Description		C function #	Description
10	Reference mode for ambient emission readings. No communications take place. Uses LIN harness for setup.	⇔	10	Reference mode for ambient emission readings. No communications take place. Uses LIN harness for setup.
20	LIN. Reference mode for emission readings. No LIN communications take place.	⇔	20	LIN. Reference mode for emission readings. No LIN communications take place.
21	LIN. Requests and check readings from slave. Red light if error.	⇔	21	LIN. Checks requests and transmit readings back to master. All PI:s are accepted as valid.
30	PSI5. Reference mode for emission readings. No PAS communications take place.	⇔		Uses a PSI5 PAS4 sensor as slave. Slave PDU is switched off, no software required.
31	PSI5. Checks readings from slave (=sensor). Red light if error.	⇔		
40	ACL. Reference mode for emission readings. No ACL communications take place.	⇔	40	ACL. Reference mode for emission readings. No ACL communications take place.
		⇔	41	ACL. Regulates current in loop to a varying pattern 4-20mA.
42	ACL. Checks current in loop at a low level, approx. 4mA. Red light if error.	⇔	42	ACL. Regulates current in loop to a low level, approx. 4mA.
43	ACL. Checks current in loop at a medium level, approx. 10mA. Red light if error.	⇔	43	ACL. Regulates current in loop to a medium level, approx. 10mA.
44	ACL. Checks current in loop at a high level, approx. 16mA. Red light if error.	⇔	44	ACL. Regulates current in loop to a high level, approx. 16mA.
50	PWM. Reference mode for emission readings. No PWM communications take place.	⇔	50	PWM. Reference mode for emission readings. No PWM communications take place.
52	PWM. Checks duty cycle of signal from slave at a low level, approx. 10%. Red light if error.	⇔	52	PWM. Transmits a low level duty cycle of approx. 10%.
53	PWM. Checks duty cycle of signal from slave at a medium level, approx. 50%. Red light if error.	⇔	53	PWM. Transmits a medium level duty cycle of approx. 50%.
54	PWM. Checks duty cycle of signal from slave at a high level, approx. 95%. Red light if error.	⇔	54	PWM. Transmits a high level duty cycle of approx. 95%.

5.2.1. Limit definitions for susceptibility

For susceptibility testing the red LED was used. When it was not lighted, communication was up and working. When lit, some type of error had occurred. Below is a description of what was interpreted by the master unit as an error.

LIN

The master transmitted a specific identifier, to which the slave responded with a certain predefined value and a checksum. If the received value and checksum was correct, the master accepted the communication and initiated a new round. No bit errors were accepted.

PSI5

When the sensor unit was powered, it transmitted actual readings automatically. In this case, it was an accelerometer in rest, so the data was predictable. The sensor data was not entirely stable, so binary data values 0±1 (three different values) was accepted. As the data was 10-bit, 3/1024 (0.3%) of the entire scale was defined as correct data values.

ACL

The slave PDU "transmitted" a predefined static current level at 10.0mA. To define the current level as correct, the master PDU required the converted 8-bit current level to be 121-124, which corresponds to 1.6% of the entire scale.

PWM

The slave PDU transmitted a predefined static PWM pattern with 50.0% duty cycle. The master PDU used two criteria's to determine a correct signal. Firstly it did not accept voltage dips or peaks during expected static conditions. Secondly it required the duty cycle to be within 49.9-50.2%, which corresponds to approximately 0.4% of the entire scale.

5.3. EMC Test setup/Test procedures

5.3.1. Hardware orientation

The hardware setup was as shown in *Figure 5.3*. Refer to *Appendix – Emission test procedures* and *Appendix – Susceptibility* for detailed instructions how to use hardware, software and test procedures. The red LED on the master unit was monitored by remote camera. A picture of the actual setup is shown in *Figure 5.4*.

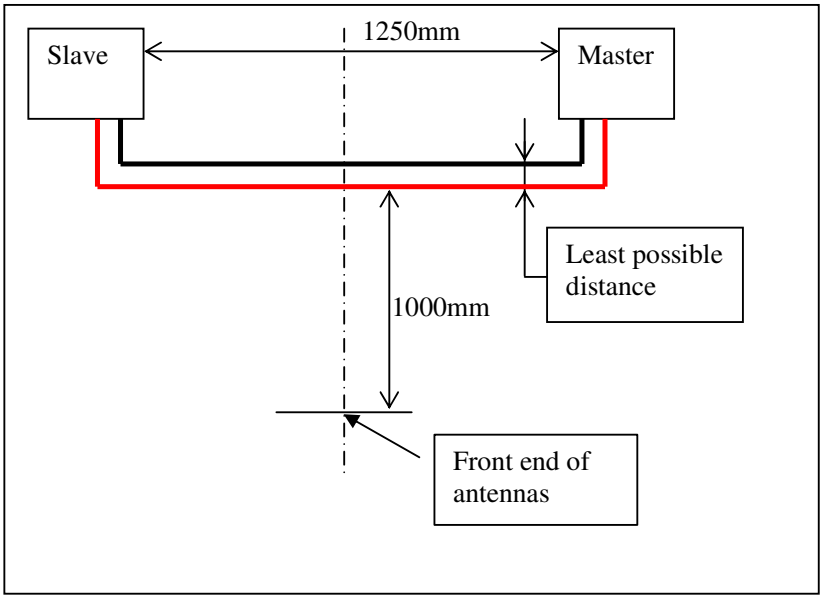


Figure 5.3. Hardware setup used for EMC-testing. 1.5m inter box cabling (black and red cables in figure) were exposed during test.

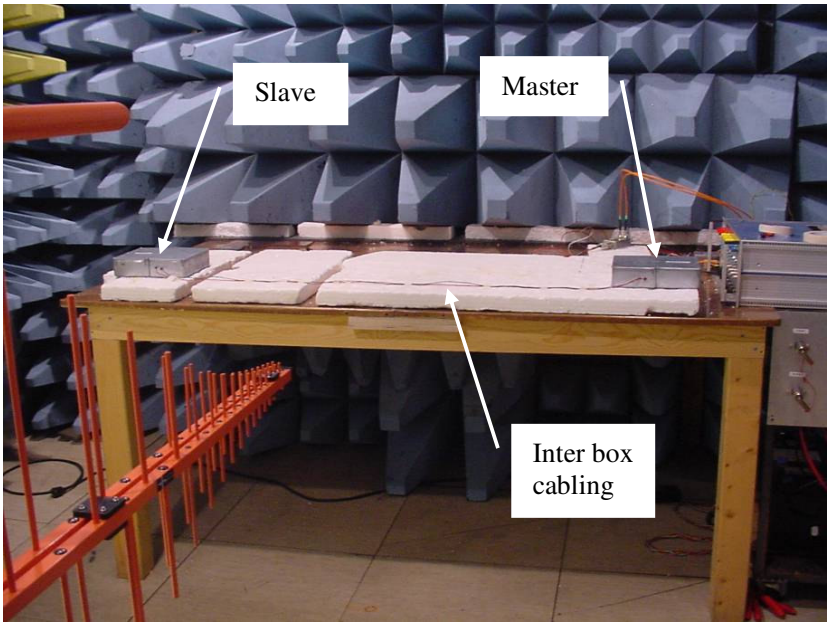


Figure 5.4. The system described in *Figure 5.3*.

During tests, only the master and slave unit with appropriate communication harness or module, and inter box cables (one red and one black) were present in the test environment.

5.3.2. A brief instruction how to use software

Refer to *Appendix – Emission test procedures* and *Appendix – Susceptibility* for detailed instructions how to use software and test procedures.

The main concepts of operation for EMC testing are:

1. Set up hardware and surroundings as described in *5.3.1*.
2. Initialize software in slave by powering it up, followed by transmitting a number from a PC via the SCI-interface. If communication was successful, the PDU responds immediately with the same function number as it received. If not successful, it responds with the number 255.
3. Initialize software in master by powering it up, followed by transmitting a number from a PC via the SCI-interface. If communication was successful, the PDU responds immediately with the same function number as it received. If not successful, it responds with the number 255.
4. Perform tests as desired. When doing susceptibility tests (functions 21, 31, 42-44, 52-54 in master and functions 21, 42-44, 52-54 in slave), a red LED light on master indicates error. If communication fails, the red LED is lighted for about a second if no more errors occur.
5. After a completed test, the internal stop switch in master may be used to investigate number of errors. When the switch is put in stop mode, the system transmits a 2-byte number indicating the total number of errors, followed by hexadecimal 0xFF. This is repeated until power is reset.

5.3.3. EMC test overview

EMC tests were performed in five days during the period from 10th to 19th of October in Volvo 3P facilities at Lundby, (refer to *4.2*). All tests were performed following Volvo group EMC standard, STD 515-0003 version 2 [7]. STD 515-0003 is based on CISPR25 [4] and fulfill requirements of 2004/104/EC [5]. In STD 515-0003, chapter six and seven are of interest, "Radiated emission" and "Radiated susceptibility" respectively.

Measuring of radiated emission was done as narrowband measurement for all methods. For those methods that not complied with limit (refer to *Figure 5.5*) average measurement was done. The aim was to find out if emissions were broad- or narrowband. If average measurement showed that an emission was broadband, accordingly broadband measurement was performed. The CISPR25 definition for narrow- and broadband signal was used.

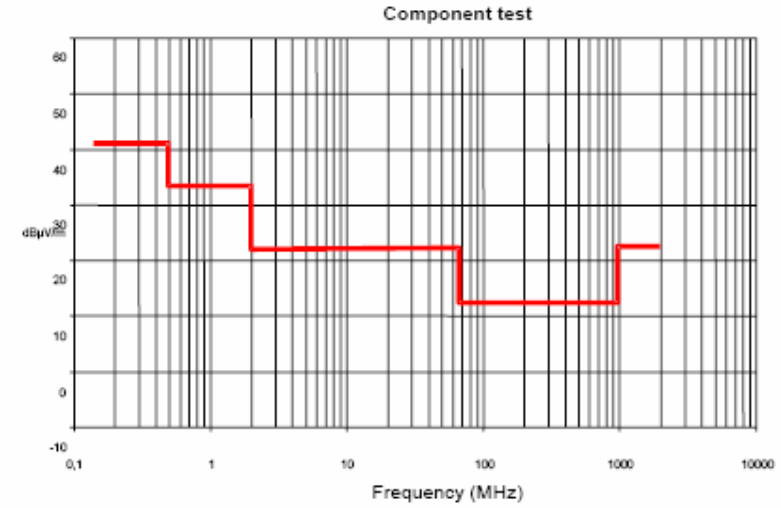


Figure 5.5. Limit line showing emission requirement level of a component test, narrowband.

According to Volvo standard STD 515-0003, emission tests are performed within 0.15-2000MHz, where studied cables and harnesses are 1.5±0.05m. This test setup was used for this projects emission studies.

In a full-scale susceptibility test three different modes of modulation are used (refer to *Figure 5.6*): continuous wave, amplitude modulated, and pulse modulation. Each is applied with increasing E field strength in predefined levels. The susceptibility tests performed in this project were limited to continuous wave modulation, due to available laboratory time.

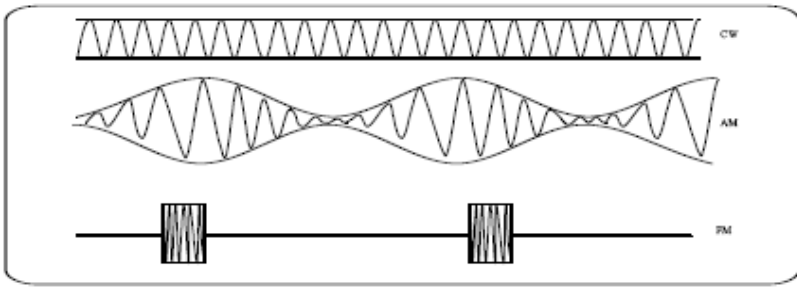


Figure 5.6. The three different modulations used in susceptibility testing.

6. Results

6.1. Emissions

In all figures in this section red line represents the Volvo. Preferably no measured emission should exceed that limit at any point.

The results from vertical and horizontal emission readings in each case are very much comparable and similar to each other (refer to *Appendix – Complete EMC emission test results*). Horizontal polarization for the test setup system and the four communication methods are chosen as representative for consideration in this section.

The test system (PDU's) used affected the noise level as in *Figure 6.1*. Some major peaks in E field can be observed between 20 and 100MHz. Of these peaks, the ones at 30 and 100MHz are of special interest due to a substantial exceeding of the limit. The reason why the green and blue line differs is leakage of emissions through inter box cables from PDU's. To improve this, a better platform than the current PDU is necessary. The green line is to be considered as a reference line in all emission tests, which the different communication methods are unlikely to fall below.

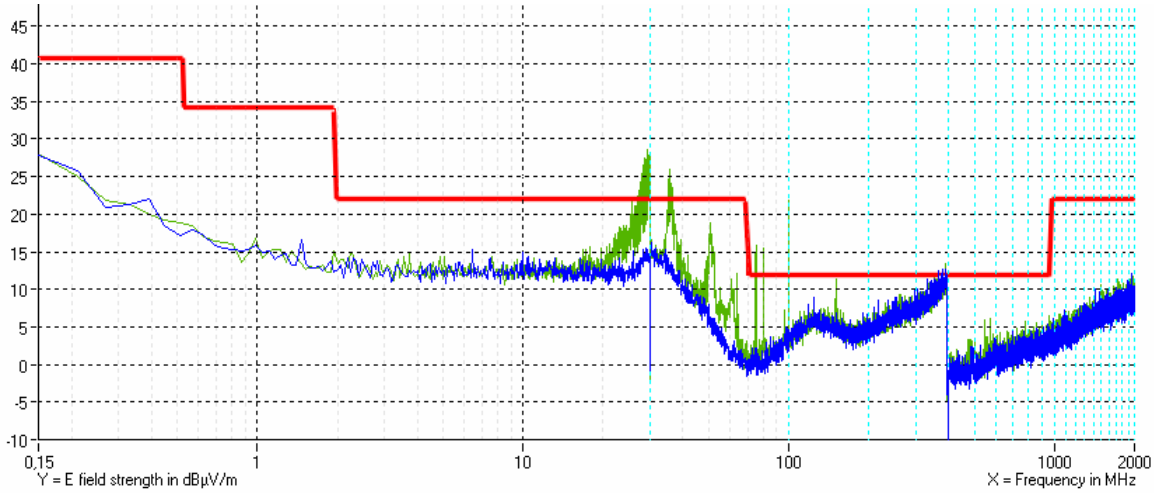


Figure 6.1. The blue line shows ambient noise level in test cell. The green line shows noise level with both PDU's running, but with no communication taking place between them. The information is a mix of *Figure E.1* & *Figure E.2*.

After the required circuitry was installed for each communication method, a noise reference measurement for all four communication methods reveals minor differences, of little or no importance, in the range 0.15-20MHz (refer to *Figure 6.2*). From 20MHz and forward, LIN, ACL and PWM continue to be quite similar, while PSI5 differs remarkably with massive E fields across the frequency spectrum. In 20-150MHz, LIN, ACL and PWM are close to, and at some points above the limit. From 150MHz they are well hidden in the reference noise level.

The activated but not communicating LIN circuitry provided interesting results: except for one major E field peak at 100MHz, it was equivalent or better than the PDU reference in the entire range 0.15-2000MHz. Refer to *Figure 6.2*, & *Figure E.3*.

After the PSI5 master circuitry was activated, but the PSI5 PAS-sensor was left without power and therefore no communication took place, some very interesting results are revealed. The curve (refer to *Figure E.9* and upper right picture in *Figure 6.2*) shows high level E fields, many larger than 15dBµV/m over limit in the range 20-1100 MHz. There are peaks at 8, 12 and 16MHz, that might be

traced to harmonics of PSI5's required clock circuit frequency at 2MHz. The other noise is not easy to explain or interpret.

At the low frequencies, 0.15-0.30MHz, ACL demonstrated a high but decreasing offset level compared to the PDU reference curve (refer to *Figure 6.2*). In the mid frequencies, 20-150MHz, it showed a number of peaks exceeding the limit.

The PWM measurement (refer to *Figure 6.2*) showed a curve very similar to the PDU reference's. The explanation is that PWM implementation required little additional circuitry (some protective circuits and passive filters only) which was only minor hardware change from the PDU reference measurement.

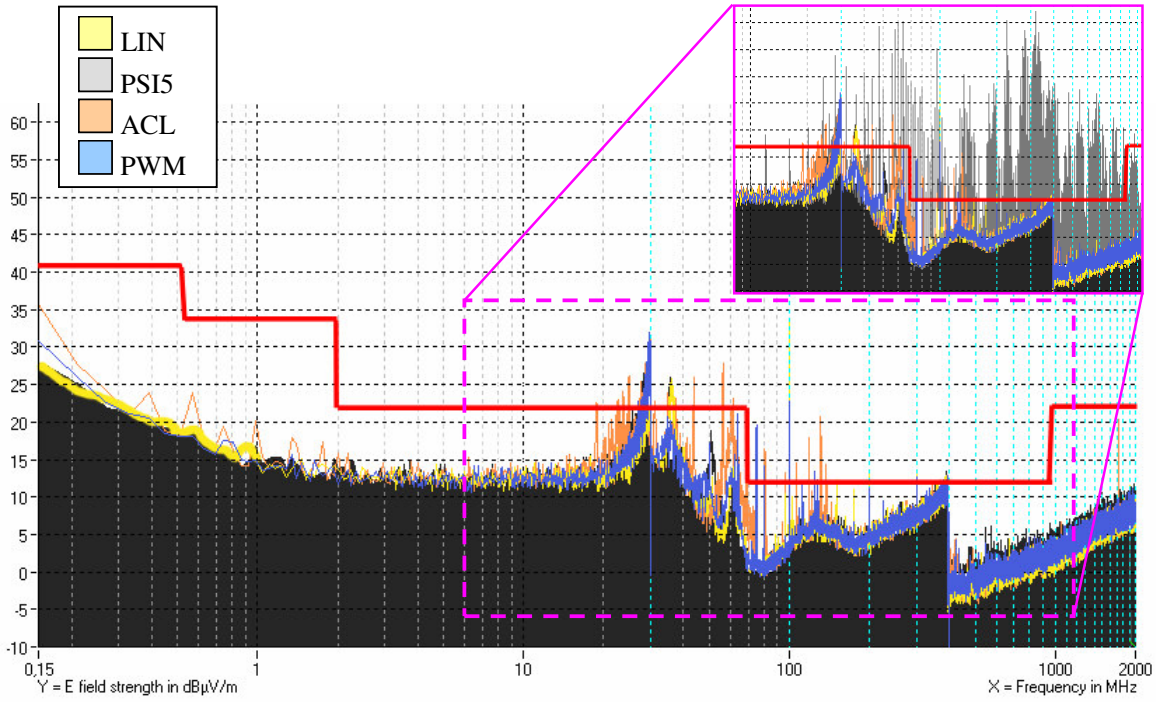


Figure 6.2. The black area represents the PDU reference noise. Colored lines show E field strength from each communication method with no communication taking place. PSI5 is excluded from the main figure to increase clarity, but a partial view including PSI5 is displayed in the upper right corner. LIN line is enhanced in 0.15-1MHz for visibility reasons. The information is a mix of *Figure E.2*, *Figure E.3*, *Figure E.7*, *Figure E.15* & *Figure E.23*.

After the communication is activated, the E fields are generally not changing much compared to the idle case (refer to *Figure 6.2* & *Figure 6.3*). The most notable differences are in frequencies ranging between 0.15-1MHz and 60-80MHz.

Refer to *Figure 6.3*. The LIN E field increased significantly in 0.15-1MHz, and was the only communication method to exceed the limit in this range. This is likely to originate from the actual communication, which in this implementation uses bit frequency at 12kHz.

PSI5 did not improve its E field 20-1000MHz performance when communication was activated. Neither did it worsen (refer to Figure E.9). The results were so poor, a suspicion arose that implementation was incorrect. Bosch contributed with a "Smartbox", a laboratory unit capable of PSI5 PAS communication. It was built on a metallic box, and therefore considered comparable to the master unit used in this project. The master unit was switched off, and disconnected. The Smartbox was placed next to the master unit and connected to the inter box cabling. (Refer to *Figure E.12*.) The

results of the test can be studied in *Figure E.11*. Except for absence of what was suspected to be clock pulse multiples in the previous PSI5 test, the E field behavior was almost identical and very poor 20-1000MHz. The lower frequencies, 0.15-20MHz, were well under the limit at all times.

Being so bad in some frequency ranges, PSI5 was exposed to average and broadband measurements. Refer to *Figure E.13* & *Figure E.14*. The E field emissions from PSI5 were broadband, and still over the limit.

The communicating ACL system showed a curve quite similar to the ACL idle. There were two frequency ranges where it differed slightly, in 25-30MHz and 300-400MHz. Refer to *Figure 6.3.*

Refer to *Figure 6.3*. The PWM E field increased in 0.15-1MHz. This is likely to originate from the actual communication bit frequency at 2.0kHz. Some additional E field noise occurred at 40-150MHz with peaks well over the limit.

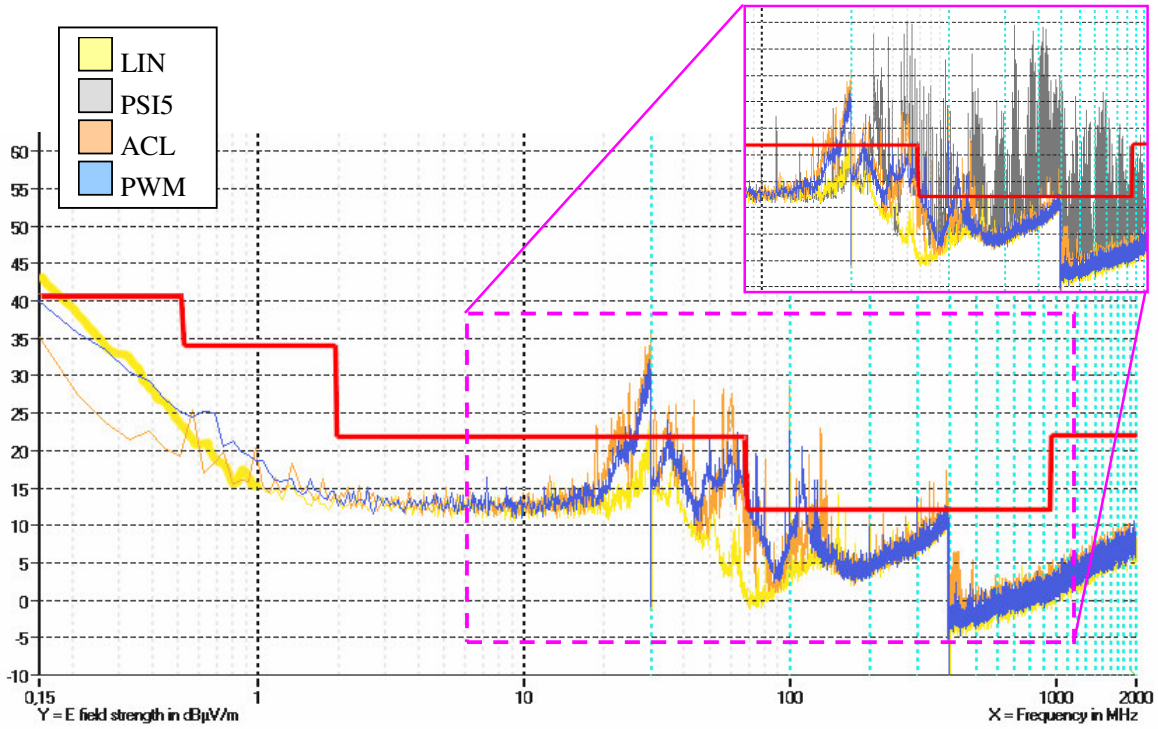


Figure 6.3. Colored lines show E field strength from each communication method with active communication. PSI5 is excluded from the main figure to increase clarity, but a partial view including PSI5 is displayed in the upper right corner. LIN line is enhanced in 0.15-1MHz for visibility reasons. The information is a mix of *Figure E.5*, *Figure E.9*, *Figure E.17* & *Figure E.24*.

The results of emissions can be split in two separate parts, one communication dependant part (0.15-~10MHz) and one circuit dependant part (~10-2000MHz) [8]. The ranking lists in descending order (best first).

Communication (0.15- ~10MHz):

1. ACL PSI5
3. PWM
4. LIN

Circuitry (~10-2000MHz):

1. LIN
2. PWM
3. ACL
4. PSI5

6.2. Susceptibility

The results are presented for each test level as graphs and tables. In the graphs, the lines for each method have been separated at each level for increased visibility, but they still represent the same applied E field strength. The lines represent full functionality of the system (which was determined by the absence of red LED light on the master unit). Interruption of the lines shows where failures occurred. The tables describe for which frequencies and E field strengths a method failed. Volvo requirements [7] for components states persistence to E fields as: 30V/m for level 1, 60V/m for level 2 and 100V/m for level 3.

The susceptibility results for 400-2000MHz are presented in *Figure 6.4*, *Figure 6.5* & *Table* **6.1**. LIN and PWM showed a very good performance with no failures at any level. PSI5 worked well at most frequencies, except in 400-680MHz where failures appeared. ACL had many failures at 50 and 100V/m and some still at 20V/m.

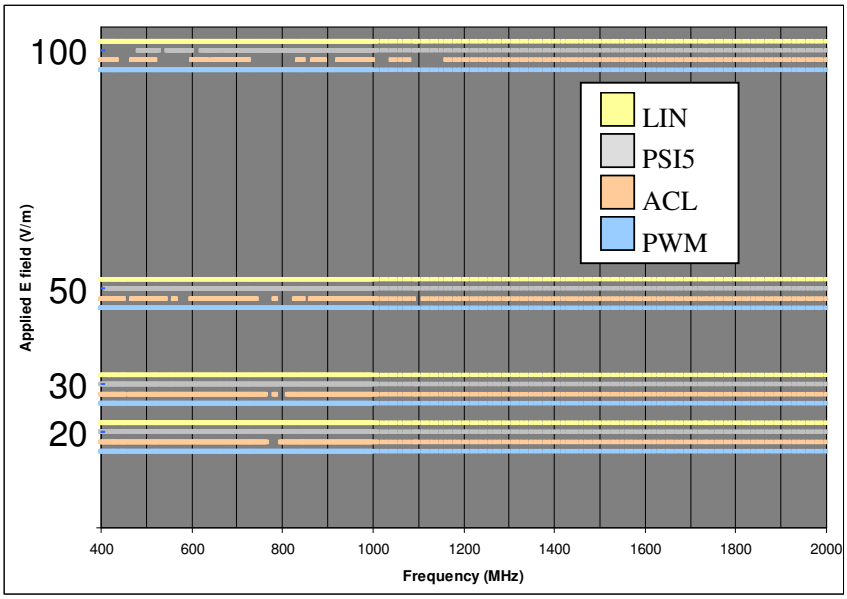


Figure 6.4. The results of susceptibility testing 400-2000MHz. Horizontal polarization. Interruption of the lines shows at which frequencies communication failed.

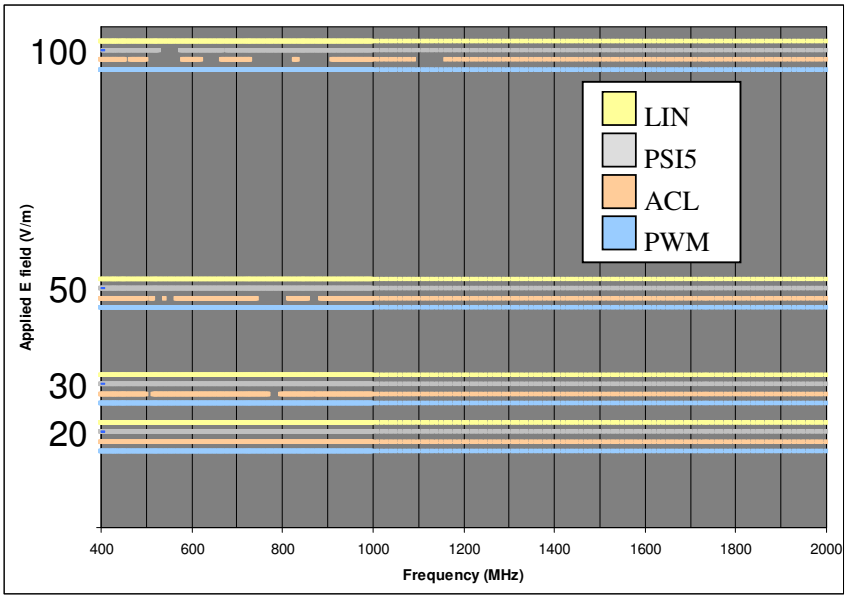


Figure 6.5. The results of susceptibility testing 400-2000MHz. Vertical polarization. Interruption of the lines shows at which frequencies communication failed.

Table 6.1. The results of susceptibility testing 400-2000MHz. The content is a union (of failures) of horizontal and vertical polarization.

	Applied field strength (V/m)			
	100	**50**	**30**	**20**
LIN	No errors.	No errors.	No errors.	No errors.
PSI5	(400-481)(514-515) (530-575)(604-621) (664-677)	(400-409)	(400-401)	No errors.
ACL	(424-427) (438-467) (502-599) (622-665) (728-832) (836-921) (934-937) (1005-1035) (1090-1155)	(452-467) (502-509) (518-565) (568-597) (746-827) (848-885) (1095-1105)	(456-461) (502-515) (766-811) (872-873)	(768-795)
PWM	No errors.	No errors.	No errors.	No errors.

The results from susceptibility tests in 0.15-400MHz are presented in *Figure 6.6* & *Table* 6.2. PSI5 did not perform as well as previously, ACL was even worse. Huge gaps of malfunction, even at low E field levels, are visibly obvious in *Figure 6.6*. LIN failed at one narrow frequency range. PWM still performed great.

PSI5 and ACL demonstrated great incapability at 30V/m. Due to this, the levels 50 and 75V/m were not completely tested, and the actual results of those levels might differ from the ones in *Figure 6.6*.

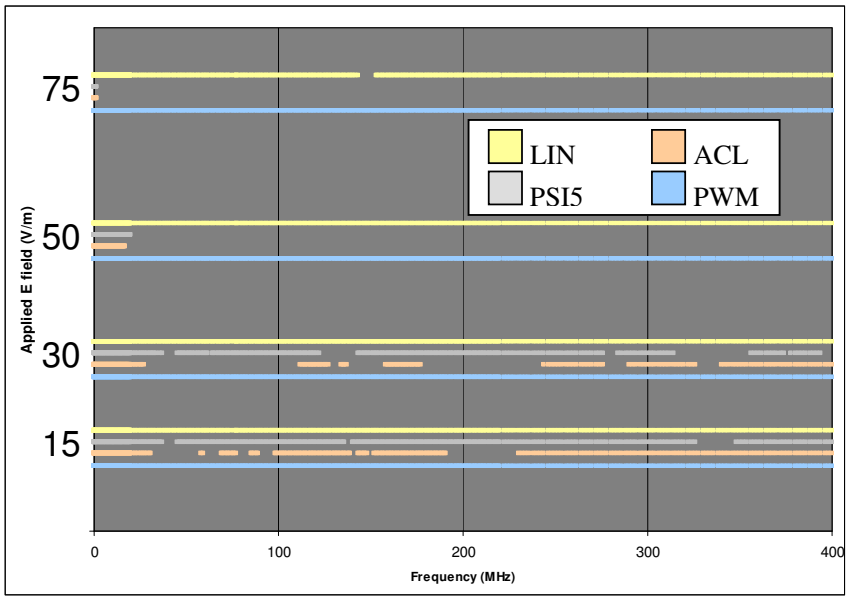


Figure 6.6. The results of susceptibility testing 0.15-400MHz. Interruption of the lines shows at which frequencies communication failed.

Table 6.2. The results of susceptibility testing 0.15-400MHz. The content is a union (of failures) of horizontal and vertical polarization.

	Applied field strength (V/m)			
	75	**50**	**30**	**15**
LIN	(144-152)	No errors.	No errors.	No errors.
PSI5	(1.5-400)	(20-400)	(38-44) (63-64) (123-142) (278-283) (316-355) (376-377) (396-400)	(38-44) (137-139) (328-347)
ACL	(1.5-400)	(16.8-400)	(19.3-19.6) (28-111) (128-133) (138-157) (178-243) (278-289) (328-339)	(32-57) (60-68) (78-84) (90-97) (140-142) (149-151) (192-229)
PWM	(16.8-18.4) (19.2-19.9)	(19.8-19.9)	No errors.	No errors.

The results from susceptibility tests can be summarized in a ranking list in descending order (best first).

1. PWM
2. LIN
3. PSI5
4. ACL

7. Summary and conclusions

7.1. EMC test conclusions

Is it the communication that causes emissions?
The result of the emission tests shows more over-the-limit peaks in the high frequency range 10-200MHz than in the low frequencies 0.15-1MHz (where only one communication method exceeded the limit). The conclusion from the results of this study is that in general, circuits are more of a problem than the actual communication.

What causes the PSI5 emission noise?
The PSI5 emission results are interesting as they differ remarkably from the other three communication methods. What is considered to be harmonics of PSI5 2MHz clock in the implemented test setup, vanished when the test setup master node was replaced by Bosch Smartbox. Bosch managed to limit the negative side effects of the clock, but that did not improve the noise at 20-1100MHz. The conclusion is that the noise must then be a result of other aspects within the Bosch PSI5 chipsets.

Which type of communication is best in susceptibility?
The results of susceptibility tests reveal great differences in performance. The most varying outcome can be found in the lower frequencies, at 0.15-400MHz where PWM followed by LIN performed great while PSI5 and ACL did really badly. The conclusion from this is that digital voltage-based communication methods are superior to analog and digital current-based.

Is there a connection between emissions and susceptibility?
When investigating the results, it could not be established that the emission peaks at a certain frequency range for a specific method resulted in any susceptibility weakness for the same or nearby frequencies.

7.2. Recommendations and reflections

Controlling emissions
We recognize the emission problem as two different problem areas, one communication dependant part (0.15-~10MHz) and one circuit dependant part (~10-2000MHz). This separation should make it easier to handle emission problems.

- Circuit dependant emissions can possibly be delimited by:
  1. Choice of communication method.
  2. Better PCB layout with shorter circuits, elimination of loops, common ground layout.
  3. Choice of components.
  4. Improved shielding and input/output filtering.
  5. Correct shield connection to vehicle ground.
- Communication dependant emissions are controlled by:
  1. Choice of communication method.
  2. Wire length.
  3. Wire capacitance.
  4. Wiring type: shielded or unshielded, twisted pair or separated cables.
  5. Dedicated return cable instead of common return path.

The major effect of minor hardware modifications
PWM was tested with and without protective circuits. The protective circuits serve as µC inputs barrier against high voltage peaks. A side effect of these circuits is first order LP-filter properties. These simple filters have a significant impact on emissions; consider *Figure E.21* & *Figure E.24*. By trimming the hardware it might be possible to greatly improve performance of PWM as well as other communication methods.

The relevance of the test system used
The software used in the setup maintains communication similar to a real case system. It is more difficult to create a representative hardware environment as implementation and conditions vary greatly.

To test maximum performance for each communication method, it would have been better to build a specific hardware for each of them. In that case no compromises or limitation in design would be necessary. If, as in this thesis, the aim is to study and compare the effects and characteristics of the actual communication rather than circuits and electronics, a single hardware setup capable of all the different communication methods is to prefer. The test system used belongs in the latter category.

PSI5
PSI5 circuitry is capable of autonomous sensor communication, and only responds to a µC when it is told to do so. This is due to the high speed SPI interface between µC and PSI5 circuitry, compared to the sensor communication. This key property allows one µC to operate several PSI5 circuits.

ACL
ACL have been used by the process industry for almost 30 years. At current time, it would not have been used unless quite reliable. The results of this study show ACL as unreliable under certain conditions, such as E field susceptibility. This is somewhat of a contradiction, so we suspect that a lot of more effort must be done in implementing ACL for achieving a reliable communication. That would however most likely be expensive compared to the alternative communication methods, so if ACL does not work in an easy implementation, maybe ACL is better replaced by something else.

A nice feature offered by ACL is the easy way to implement different modes. For example:

- If a sensor was not required to deliver high resolution information, it could be adjusted for 1-5mA circuits resulting in low power consumption. A possible problem with low currents can be oxidation in connectors.
- If a sensor or application needs more power it can be adjusted in a similar way to operate from 15-20mA. In this case, 15mA can be consumed by the slave node.
- For better resolution, it is better to use as wide current range as possible.

7.3. Recommendations matrix

Refer to *Table 7.1* for recommendations when to use each communication method.

Table 7.1. Recommendations which communication method to use for desired properties.

Feature	***Weight***	**LIN**	**PSI5**	**ACL**	**PWM**
Low E field emissions from communication	*4*		X	X	X
Low E field emissions from circuitry	*4*	X			
Robustness to E fields	*5*	X			X
Advanced diagnostics	*4*	X			
Bus mode	*3*	X	X		
Point-to-point mode	*1*	X	X	X	X
High transmission speed	*2*		X	X	X
Operable at 24V power supply	*3*	X	X	X	X
Total point (Weight * X), max 26		*20*	*13*	*10*	*15*

7.4. Possible sources of error

The test setup
PDU's were used in a previous project, and had several components of no use to this study. The communication methods investigated maybe would have performed better or more adequate if implemented directly on the PCB, with full control over the physical properties.

Susceptibility limits
For this project, there were different ways to decide if communication was correct or not. Since the communication methods varied in operational principles, there was no obvious way to compare them entirely fair. Even though extra tolerance was given to ACL due to analog operation, it did not perform as good. Another question is whether it is right to accept greater tolerances for analog systems, which an effect as analog systems have floating limits which values are correct and which are not.

Static values
PWM and ACL were tested for susceptibility at a specific level in the middle of their respective measure scale. It was not studied if the susceptibility test results would differ if the current level (ACL) or duty cycle (PWM) were altered.

7.5. Future work

- It would be very interesting to implement and compare an alternative system to wire based transmissions. Several examples of existing products can be found at http://www.jdsu.com. They offer an optical fiber based system where both power and sensor data is transmitted simultaneously via a single fiber. A system like that would have great opportunities to deliver excellent performance regarding EMC and harsh conditions. Fibers are also light weight.
- It would also be of interest to add CAN to this comparison.
- The effects of different types of cabling, e.g. twisted pair or shielding, could be studied further.
- What impact does the implemented communication speed have? This is relevant for all the four communication methods.
- What impact does different communication signal voltage/current levels have?
- Further investigation of PSI5's EMC problems.

7.6. Project summary

The first phase of the project; problem formulation, planning, studies of protocols and specifications, brought intensive desktop work. This was sometimes tough, as we had little experience in the field at this detailed level.

The second phase was implementation and EMC testing. This was time critical in order to be ready when the EMC laboratory was available to our work. As always, designing and soldering takes a lot more time than anticipated, but eventually we had a complete test platform, ready to test. At this phase, it helped a lot not to have tight economic restrictions, we could buy the tools, equipment and components not available.

The third phase was analyzing and report formulation. The time for this can probably not be under estimated, depending on the required level of details in the report.

In our opinion the thesis project went smoothly. We were given lots of room to solve the problems we encountered, while we had solid help at hands when needed.

8. References

8.1. Literature

[1] LIN consortium: “LIN Specification Package Rev. 2.0”
http://www.lin-subbus.org/

[2] PSI5: Technical Specification v1.0a 2005-07
http://www.psi5.org

[3] PWM: Mohan et al., “Power Electronics – Converters, Applications and Design”, Wiley, 1995

[4] International Electrotechnical Commission’s CISPR 25: "Radio disturbance characteristics for the protection of receivers used on board vehicles, boats, and on devices - Limits and methods of measurement"
http://www.iec.ch/

[5] European commission EMC directive: “Commission directive 2004/104/EC”,
http://europa.eu.int/eur-lex/lex/LexUriServ/site/en/oj/2004/l_337/l_33720041113en00130058.pdf

[6] Cheng, David K, “Fundamentals of Engineering Electromagnetics”, Prentice Hall, New Jersey, 1993

[7] Standard Volvo Group STD515-0003, “Electro-magnetic compatibility, EMC”, 2006

[8] Oral Communication (2006-10): Eilert Johansson, Technology Area Director Electrics, Volvo Technology Corporation, Göteborg.

8.2. Figures

Figure 3.1 is from [1], p3
Figure 3.4 is from [2], p11
Figure 5.5 is from [7], p16
Figure 5.6 is from [7], p17

A Appendix – C-code for master PDU

```
/*****************************************************************************************************
*** COPYRIGHT (c) Volvo Technological Development Corp. 2006                                       ***
*** The copyright of the computer program(s) herein is the property                                ***
*** of Volvo Technological Development Corporation, Sweden.                                        ***
*** The program(s) may be used and copied only with written permission                             ***
*** from Volvo Technological Development Corporation, or in accordance                            ***
*** with the terms and conditions stipulated in the agreement under                               ***
*** which the program(s) have been supplied.                                                      ***
*****************************************************************************************************/

/*****************************************************************************************************

    Master

*****************************************************************************************************/

/*****************************************************************************************************
*** Included files                                                                                 ***
*****************************************************************************************************/
#include <r912dp256.h>
#include <i912dp256.h>
#include "SCI.h"
#include "types_hc12.h"

/*****************************************************************************************************
*** Defines                                                                                        ***
*****************************************************************************************************/
#define TDRE  128 & SCI0SR1                     //Adressing a specific bit: Ready to transmit.
#define RDRF  32 & SCI0SR1                      //Adressing a specific bit: receive data complete.
#define FE    2 & SCI0SR1                       //Adressing a specific bit: Framing error.
#define TC    64 & SCI0SR1                      //Adressing a specific bit: Transfer complete

/*****************************************************************************************************
*** Function declarations                                                                          ***
*****************************************************************************************************/
void _10_Reference(void);
void _20_LIN_reference(void);
void _21_LIN_Request(void);
    int receive(int unsigned *FakeData, int PI);
    void SendHeader(int PI);
    void PrepareToTransmit(void);
    void PrepareToreceive(void);
    //21 also uses "void stopbutton(int unsigned number_error);"
void _30_PSI5_Reference(void);
    void init_CG974(void);
void _31_PSI5_receive(void);
    int ReadSensor(void);
    //31 also uses "void init_CG974(void)"
    //31 also uses "void stopbutton(int unsigned number_error)"
void _40_ACL_Reference(void);
    void init_ATD(void);
void _42_ACL_receive_low(void);
    int unsigned sample_ATD(void);
    //42 also uses "void init_ATD(void)"
    //42 also uses "void stopbutton(int unsigned number_error)"
void _43_ACL_receive_mid(void);
    //43 also uses "void init_ATD(void)"
    //43 also uses "int unsigned sample_ATD(void)"
    //43 also uses "void stopbutton(int unsigned number_error)"
void _44_ACL_receive_high(void);
    //44 also uses "void init_ATD(void)"
    //44 also uses "int unsigned sample_ATD(void)"
    //44 also uses "void stopbutton(int unsigned number_error)"
void _50_PWM_Reference(void);
void _52_PWM_receive_short(void);
    //52 also uses "void stopbutton(int unsigned number_error)"
void _53_PWM_receive_mid(void);
    //53 also uses "void stopbutton(int unsigned number_error)"
void _54_PWM_receive_long(void);
    //54 also uses "void stopbutton(int unsigned number_error)"
void stopbutton(int unsigned number_error);

/*****************************************************************************************************
***              PDU main function                                                                ***
*****************************************************************************************************/
void main(void){
    int unsigned SCI_in = 0;
    //Start of PortA init for "Stopbutton"
    RDRIV = 0;
    PORTA = 0;
    DDRA = 127;
    PORTA = 95;
    //End of PortA init
```

```
    //Start of SCI init
    DDRS=     254;                                  //1111 1110 Set direction of Port S

    SCI0BDH=  5;                                    //Baudrate
    SCI0BDL=  32;                                   //Baudrate
    SCI0CR1=  4;                                    //0000 0100  Control register
    SCI0CR2=  12;                                   //0000 1100  Control register
    SCI0SR2=  2;                                    //0000 0010  Status register
    //End of SCI init

    while(1){
        if(RDRF){                                   //If SCI input received
            SCI_in = SCI0DRL;                       //Read SCI input and go to appropriate function.
            switch(SCI_in){
                case 10:
                    _10_Reference();
                case 20:
                    _20_LIN_reference();
                case 21:
                    _21_LIN_Request();
                case 30:
                    _30_PSI5_Reference();
                case 31:
                    _31_PSI5_receive();
                case 40:
                    _40_ACL_Reference();
                case 42:
                    _42_ACL_receive_low();
                case 43:
                    _43_ACL_receive_mid();
                case 44:
                    _44_ACL_receive_high();
                case 50:
                    _50_PWM_Reference();
                case 52:
                    _52_PWM_receive_short();
                case 53:
                    _53_PWM_receive_mid();
                case 54:
                    _54_PWM_receive_long();
                default:
                    SCI0DRL = 255;
                    while((SCI0SR1 & 64) == 0){}
                    break;
             }
        }
    }
}  /* End of main */

/*** REFERENCE ********************************************************************************/
void _10_Reference(void){
    int unsigned number_error = 0;
    SCI0DRL = 10;                                   //Confirm function call.
    while((SCI0SR1 & 64) == 0){}                    //Wait until done.

    while(1){}                                      //Trap
    return;
}

/*** LIN ********************************************************************************/
void _20_LIN_reference(void){
    SCI0DRL = 20;                                   //Confirm function call.
    while((SCI0SR1 & 64) == 0){}                    //Wait until done.

    //SCI init (needed for LIN)
    SCI0BDH = 0x00;
    SCI1BDH = 0x00;
    SCI0BDL = 0xC3;                                 //Set BR to 2400
    /* Control register 1 */
    SCI0CR1 = 0x00;
    SCI1CR1 = 0x00;
    /* Control register 2, enable reception and transmition */
    SCI0CR2 = 0x0C;
    SCI1CR2 = 0x0C;
    //End of SCI init

    //LIN init
    SCI1CR2 = 0x00;
    DDRS    = 36;                                   //xxxx x1xx
    PTS = PTS | 4;                                  //activates LIN chipset
    //End of init LIN

    while(1){}                                      //Trap
    return;
}

/*** LIN ********************************************************************************/
void _21_LIN_Request(void){
```

```
    int unsigned number_error = 0, i, j, Redlight = 0, FakeData = 0;
    int Status =0;
    SCI0DRL = 21;                                   //Confirm function call
    while((SCI0SR1 & 64) == 0){}                    //Wait until done.

    //SCI init (needed for LIN)
    SCI0BDH = 0x00;
    SCI1BDH = 0x00;
    SCI0BDL = 0xC3;                                 //Set BR to 2400
    /* Control register 1 */
    SCI0CR1 = 0x00;
    SCI1CR1 = 0x00;
    /* Control register 2, enable reception and transmition */
    SCI0CR2 = 0x0C;
    SCI1CR2 = 0x0C;
    //End of SCI init

    //LIN init
    SCI1CR2 = 0x00;
    DDRS    = 36;                                   //xxxx x1xx
    PTS = PTS | 4;                                  //activates LIN chipset
    //End of init LIN

    while(1){
        for(j=0; j<=255; j++){
            SendHeader(j);                          //Transmit header.
            Status = receive(&FakeData, j);         //Receive response.

            if((Status == -1 ) || (FakeData != j)){   //If received is not ok, then...
                PORTA = PORTA | 32;                 //...turn on red LED...
                Redlight = 100;                     //...set counter...
                number_error++;                     //...increase nr of errors.
                if(number_error > 65000)            //Don't increase beyond 65000 errors.
                    number_error=65000;
            }

            for (i=0; i<2000; i++){}                //interframe spacing

            if(Redlight == 0)                       //If redlight has been turn on long enough...
                PORTA = PORTA & 223;                //...then turn off red LED
            if(Redlight > 0)                        //If countdown in progress...
                Redlight--;                         //...decrease redlight.
        }

        if(PORTA > 127){                            //Check if stop button is pressed.
            PTS = PTS & 251;                        //Deactivate LIN chipset
            stopbutton(number_error);
        }
    }
    return;
}

int receive(int unsigned *FakeData, int PI){
    int i = 0, ok = 1, Checksum, CalcChecksum;

    // Data
    PrepareToreceive();                             //Prepare to receive response from slave.
    while((!(RDRF || FE)) && (i<2000)){             //Wait for the response.
        i++;
    }

    if((FE != 0) || (i >=2000))                     //If response transmission error...
        return -1;                                  //...then exit.

    *FakeData = SCI0DRL;

    // Checksum
    PrepareToreceive();                             //Prepare to receive checksum from slave.
    i = 0;
    while(!(RDRF || FE) && (i<2000)){               //Wait for the response.
        i++;
    }

    if((FE != 0) || (i >=2000))                     //If checksum transmission error...
        return -1;                                  //...then exit.
    Checksum = SCI0DRL;

    // Calculate Checksum
    CalcChecksum = *FakeData + PI;
    if(CalcChecksum > 255){
        CalcChecksum = CalcChecksum & 255;
        CalcChecksum++;
    }

    if((Checksum + CalcChecksum) != 0xFF)           //If checksum calculation failed...
        return -1;                                  //...then exit.
    //End of checksum calculation
```

```
    return 0;                                   //Return 0 if everything ok!
}

void SendHeader(int PI){
    // Break
    SCI0BDL = 0xC3;                             //Equals BR=2400
    PrepareToTransmit();
    SCI0DRL=0x00;
    while(!(TC)){}                              //Check Transfer-Complete-flag.

    // Synch
    SCI0BDL = 0x82;                             //Equals BR=9600
    PrepareToTransmit();
    SCI0DRL=0x55;                               //Send synch byte
    while(!(TDRE)){}                            //Check TDRE-flaggan

    // PI
    PrepareToTransmit();
    SCI0DRL=PI;                                 //Send PI
    while(!(TC)){}
    return;
}

void PrepareToTransmit(void){
    int temp;
    temp = SCI0SR1;                             //Clear flags
    temp = SCI0DRL;
    SCI0CR1 = 0;
    SCI0CR2 = 8;                                //TE Enable
    return;
}

void PrepareToreceive(void){
    int temp;
    temp = SCI0SR1;                             //Clear flags
    temp = SCI0DRL;
    SCI0CR1 = 0;
    SCI0CR2 = 4;                                //RE Enable
    return;
}

/*** PSI5 ********************************************************************************************/
void _30_PSI5_Reference(void){
    int unsigned number_error = 0;
    SCI0DRL = 30;                               //Confirm function call
    while((SCI0SR1 & 64) == 0){}

    //Init SPI
    MODRR = MODRR & 239;                        //Set MODRR[4] = 0
    SPI0CR1 = 86;                               //0101 0110
    SPI0CR2 = 16;                               //0001 0000
    SPI0BR  = 16;                               // 0000 0010 Prescaler = 2

    RDRIV = 0;                                  //All ports on full effect
    DDRA = 127;
    PORTA = PORTA | 64;                         //Set CS high
    //End of init SPI

    init_CG974();

    while(1){}

    return;
}

void init_CG974(void){
    int unsigned i;
    i = SPI0SR;

    // Program
    while(!(SPI0SR & 32)){}
    PORTA = PORTA & 191;
    SPI0DR = 94;
    while(!(SPI0SR & 32)){}
    SPI0DR = 1;                                 //PAS type here
    while(!(SPI0SR & 32)){}
    for (i=0; i<1; i++){}                       //CS delay loop
    PORTA = PORTA | 64;
    for (i=0; i<15; i++){}                      //Interframe spacing

    // Read back programmed data
    while(!(SPI0SR & 32)){}
    PORTA = PORTA & 191;
    SPI0DR = 126;
    while(!(SPI0SR & 32)){}
    SPI0DR = 0;
    while(!(SPI0SR & 32)){}
    for (i=0; i<1; i++){}                       //CS delay loop
```

```
    PORTA = PORTA | 64;
    for (i=0; i<15; i++){}                          //Interframe spacing

    // End of programming
    while(!(SPI0SR & 32)){}
    PORTA = PORTA & 191;
    SPI0DR = 12;
    while(!(SPI0SR & 32)){}
    SPI0DR = 0;
    while(!(SPI0SR & 32)){}
    for (i=0; i<1; i++){}                           //CS delay loop
    PORTA = PORTA | 64;
    for (i=0; i<15; i++){}                          //Interframe spacing

    return;
}

/*** PSI5 ********************************************************************************************/
void _31_PSI5_receive(void){
    int Status = 0, Redlight = 0;
    int unsigned i, number_error = 0;
    SCI0DRL = 31;                                   //Confirm function call
    while((SCI0SR1 & 64) == 0){}

    //Init SPI
    MODRR = MODRR & 239;                            //Set MODRR[4] = 0
    SPI0CR1 = 86;                                   //0101 0110
    SPI0CR2 = 16;                                   //0001 0000
    SPI0BR  = 16;                                   //0000 0010 Prescaler = 2

    RDRIV = 0;                                      //All ports on full effect
    DDRA = 127;
    PORTA = PORTA | 64;                             //Set CS high
    //End of init SPI

    init_CG974();

    // Power on
    while(!(SPI0SR & 32)){}
    PORTA = PORTA & 191;
    SPI0DR = 50;
    while(!(SPI0SR & 32)){}
    SPI0DR = 1;
    while(!(SPI0SR & 32)){}
    for (i=0; i<1; i++){}                           //CS delay loop
    PORTA = PORTA | 64;
    for (i=0; i<15; i++){}                          //Interframe spacing
    //End of power on

    for(i=0; i<4; i++){                             //Fake readings, not used.
        ReadSensor();
    }

    while(1){
        Status = ReadSensor();

        if(Status == -1){                           //If response isn't ok, then...
            PORTA = PORTA | 32;                     //...turn on red LED....
            Redlight = 1000;                        //...set redlight counter...
            number_error++;                         //...increase nr of errors...
            if(number_error > 65000)                //Don't store more than 65000 errors.
                number_error=65000;
        }

        if(Redlight == 0)                           //If redlight countdown is complete, then...
            PORTA = PORTA & 223;                    //...turn off red LED
        if(Redlight > 0)                            //If redlight countdown in progress, then...
            Redlight--;                             //...decrease redlight.

        if(PORTA > 127){                            //Check if stop button is pressed.
            stopbutton(number_error);
        }
    }
    return;
}

int ReadSensor(void){
    int unsigned i, spi_high, spi_low;
    i = SPI0SR;

    //Send and receive SPI info.
    while(!(SPI0SR & 32)){}
    PORTA = PORTA & 191;
    SPI0DR = 128;
    while(!(SPI0SR & 128)){}
    spi_high = SPI0DR;
    SPI0DR = 0;
    while((SPI0SR & 128) == 0){}
```

```
    //End of send and receive SPI info.

    PORTA = PORTA | 64;
    for (i=0; i<807; i++){}                     //Interframe spacing for 2kHz
    spi_low = SPI0DR;

    if((spi_high & 128) != 0){                  //Error if TFF set
        return -1;
    }

    i = spi_high*256 + spi_low;                 //Assemble 16-bit information
    i = i & 1023;                               //Delete first 6 bits

    if((i == 0) || (i == 1020) || (i == 4)){    //If sensor data is 0, 1024 or 4, it's ok.
        return 0;
    }

    return -1;
}

/*** ANALOG CURRENT LOOP ******************************************************************************/
void _40_ACL_Reference(void){
    int unsigned number_error = 0;
    SCI0DRL = 40;                               //Confirm function call
    while((SCI0SR1 & 64) == 0){}

    init_ATD();
    while(1){}
    return;
}

void init_ATD(void){
    int i;
    ATD1CTL2 = 192;
    ATD1CTL3 = 12;
    ATD1CTL4 = 227;
    ATD1DIEN = 0;
    for(i=0; i<1000;i++){}
    return;
}

/*** ANALOG CURRENT LOOP ******************************************************************************/
void _42_ACL_receive_low(void){
    int unsigned number_error = 0, FakeData, Redlight = 0, i;
    SCI0DRL = 42;                               //Confirm function call
    while((SCI0SR1 & 64) == 0){}

    init_ATD();
    while(1){

        FakeData = sample_ATD();                //Sample from ACL receiver.

        if((FakeData < 52) || (FakeData > 55)){ //Equals 4mA
            PORTA = PORTA | 32;                 //Turn on red LED.
            Redlight = 1000;
            number_error++;
            if(number_error > 65000)
                number_error=65000;
        }

        if(Redlight == 0)
            PORTA = PORTA & 223;                //Turn off red LED
        if(Redlight > 0)
            Redlight--;

        for (i=0; i<345; i++){}                 //Interframe spacing for 2kHz

        if(PORTA > 127){                        //Check if stop button is pressed.
            stopbutton(number_error);
        }
    }
    return;
}

int unsigned sample_ATD(void){
    int i;
    ATD1CTL5 = 5;                               //Start conversion
    for(i=0;i<50;i++){}                         //Delayloop

    return ATD1DR0H;
}

/*** ANALOG CURRENT LOOP ******************************************************************************/
void _43_ACL_receive_mid(void){
    int unsigned number_error = 0, FakeData, Redlight = 0, i;
    SCI0DRL = 43;                               //Confirm function call
    while((SCI0SR1 & 64) == 0){}
```

```
    init_ATD();

    while(1){

        FakeData = sample_ATD();                  //Sample from ACL receiver.

        if((FakeData < 121) || (FakeData > 124)){ //Equals 10mA
            PORTA = PORTA | 32;                   //Turn on red LED.
            Redlight = 1000;
            number_error++;
            if(number_error > 65000)
                number_error=65000;
        }

        if(Redlight == 0)
            PORTA = PORTA & 223;                  //Turn off red LED.
        if(Redlight > 0)                          //Continue redlight countdown.
            Redlight--;

        for (i=0; i<345; i++){}                   //Interframe spacing for 2kHz.

        if(PORTA > 127){                          //Check if stop button is pressed.
            stopbutton(number_error);
        }
    }
    return;
}

/*** ANALOG CURRENT LOOP ******************************************************************************/
void _44_ACL_receive_high(void){
    int unsigned number_error = 0, FakeData, Redlight = 0, i;
    SCI0DRL = 44;                                 //Confirm function call
    while((SCI0SR1 & 64) == 0){}

    init_ATD();
    while(1){
        FakeData = sample_ATD();                  //Sample from ACL receiver.
        if((FakeData < 160) || (FakeData > 170)){ //Equals 16.72mA
            PORTA = PORTA | 32;                   //Turn on red LED.
            Redlight = 1000;
            number_error++;
            if(number_error > 65000)
                number_error=65000;
        }
        if(Redlight == 0)
            PORTA = PORTA & 223;                  //Turn off red LED
        if(Redlight > 0)                          //Continue redlight countdown.
            Redlight--;
        for (i=0; i<345; i++){}                   //Interframe spacing for 2kHz
        if(PORTA > 127){                          //Check if stop button is pressed.
            stopbutton(number_error);
        }
    }
    return;
}

/*** PWM **********************************************************************************************/
void _50_PWM_Reference(void){
    SCI0DRL = 50;                                 //Confirm function call
    while((SCI0SR1 & 64) == 0){}

    DDRA =  DDRA & 239;
    while(1){}
    return;
}

/*** PWM **********************************************************************************************/
void _52_PWM_receive_short(void){
    int unsigned a, b, number_error = 0, Redlight = 0;
    float dutycycle, A, B;
    SCI0DRL = 52;                                 //Confirm function call
    while((SCI0SR1 & 64) == 0){}
    DDRA =  DDRA & 239;

    while(1){
        //Sample PWM input
        a = 0;
        b = 0;
        while((PORTA & 16 ) != 0 ){}              //Wait for a new period.
        while((PORTA & 16) == 0){}
        while(((PORTA & 16) != 0) && (a < 1000)){
            a++;
        }
        b=a;
        while(((PORTA & 16) == 0) && (a < 3000)){
            a++;
        }
        //End of sampling of PWM input
```

```
        B = b;
        A = a;
        dutycycle = (B / A);                    //Calculate dutycycle

        if((dutycycle < 0.100) || (dutycycle > 0.103)){    //Warning: dutycycle must not be 33%!
            PORTA = PORTA | 32;                 //Turn on red LED.
            Redlight = 1000;
            number_error++;
            if(number_error > 65000)
                number_error=65000;
        }

        if(Redlight == 0)
            PORTA = PORTA & 223;                //Turn off red LED.
        if(Redlight > 0)                        //Continue redlight countdown.
            Redlight--;

        if(PORTA > 127){                        //Check if stop button is pressed.
            stopbutton(number_error);
        }
    }
    return;
}

/*** PWM ************************************************************************************/
void _53_PWM_receive_mid(void){
    int unsigned a, b, number_error = 0, Redlight=0;
    float dutycycle, A, B;
    SCI0DRL = 53;                               //Confirm function call
    while((SCI0SR1 & 64) == 0){}
    DDRA =  DDRA & 239;

    while(1){
        //Sample PWM input
        a = 0;
        b = 0;
        while((PORTA & 16 )!= 0 ){}             //Wait for a new period.
        while((PORTA & 16) == 0){}
        while(((PORTA & 16) != 0) && (a < 1000)){
            a++;
        }
        b=a;
        while(((PORTA & 16) == 0) && (a < 3000)){
            a++;
        }
        //End of sampling of PWM input

        B = b;
        A = a;
        dutycycle = (B / A);                    //Calculate dutycycle

        if((dutycycle < 0.499) || (dutycycle > 0.502)){    //Warning: dutycycle must not be 33%!
            PORTA = PORTA | 32;                 //Turn on red LED.
            Redlight = 1000;
            number_error++;
            if(number_error > 65000)
                number_error=65000;
        }

        if(Redlight == 0)
            PORTA = PORTA & 223;                //Turn off red LED
        if(Redlight > 0)
            Redlight--;

        if(PORTA > 127){                        //Check if stop button is pressed.
            stopbutton(number_error);
        }
    }
    return;
}

/*** PWM ************************************************************************************/
void _54_PWM_receive_long(void){
    int unsigned a, b, number_error = 0, Redlight=0;
    float dutycycle, A, B;
    SCI0DRL = 54;                               //Confirm function call
    while((SCI0SR1 & 64) == 0){}
    DDRA =  DDRA & 239;

    while(1){
        //Sample PWM input
        a = 0;
        b = 0;
        while((PORTA & 16 )!= 0 ){}             //Wait for a new period.
        while((PORTA & 16) == 0){}
        while(((PORTA & 16) != 0) && (a < 1000)){
            a++;
```

```
        }
        b=a;
        while(((PORTA & 16) == 0) && (a < 3000)){
            a++;
        }
        //End of sampling of PWM input

        B = b;
        A = a;
        dutycycle = (B / A);                    //Calculate dutycycle

        if((dutycycle < 0.949) || (dutycycle > 0.952)){    //Warning: dutycycle must not be 33%!
            PORTA = PORTA | 32;                 //Turn on red LED.
            Redlight = 1000;
            number_error++;
            if(number_error > 65000)
                number_error=65000;
        }

        if(Redlight == 0)
            PORTA = PORTA & 223;                // Turn off red LED
        if(Redlight > 0)
            Redlight--;

        if(PORTA > 127){                        //Check if stop button is pressed.
            stopbutton(number_error);
        }
    }
    return;
}

/*** STOPBUTTON ************************************************************************************/
void stopbutton(int unsigned number_error){
    int unsigned outputbyte = 0;
    //Re-initiation of SCI. Set register for sending.
    DDRS=     254;                              //1111 1110 Set direction of Port S
    SCI0BDH=  5;                                //Baudrate
    SCI0BDL=  32;                               //Baudrate
    SCI0CR1=  4;                                //0000 0100  Control register
    SCI0CR2=  12;                               //0000 1100  Control register
    SCI0SR2=  2;                                //0000 0010  Status register

    //Send 0xFF followed by number_error until unit power off.
    while(1){
        SCI0DRL=255;                            //Send 0xFF
        while((SCI0SR1 & 64) == 0){}
        outputbyte = number_error / 256;
        SCI0DRL = outputbyte;                   //Send high byte of number_error
        while((SCI0SR1 & 64) == 0){}
        outputbyte = number_error & 255;
        SCI0DRL = outputbyte;                   //Send low byte of number_error
        while((SCI0SR1 & 64) == 0){}
    }
    return;
}
```

B Appendix – C-code for slave PDU

```
/*****************************************************************************************************
*** COPYRIGHT (c) Volvo Technological Development Corp. 2006                                       ***
*** The copyright of the computer program(s) herein is the property                                ***
*** of Volvo Technological Development Corporation, Sweden.                                        ***
*** The program(s) may be used and copied only with written permission                             ***
*** from Volvo Technological Development Corporation, or in accordance                             ***
*** with the terms and conditions stipulated in the agreement under                                ***
*** which the program(s) have been supplied.                                                       ***
*****************************************************************************************************/

/*****************************************************************************************************

    Slave

*****************************************************************************************************/


/*****************************************************************************************************
*** Included files                                                                                 ***
*****************************************************************************************************/
#include <r912dp256.h>
#include <i912dp256.h>
#include "SCI.h"
#include "types_hc12.h"

/*****************************************************************************************************
*** Defines                                                                                         ***
*****************************************************************************************************/
#define TDRE  128 & SCI0SR1                     //Adressing a specific bit: Ready to transmit.
#define RDRF  32 & SCI0SR1                      //Adressing a specific bit: receive data complete.
#define FE    2 & SCI0SR1                       //Adressing a specific bit: Framing error.
#define TC    64 & SCI0SR1                      //Adressing a specific bit: Transfer complete

/*****************************************************************************************************
*** Function declarations                                                                           ***
*****************************************************************************************************/
void _10_Reference(void);
void _20_LIN_reference(void);
void _21_LIN_Respond(void);
    void PrepareToRespond(void);
    void PrepareToreceive(void);
    int CheckHeader(int unsigned *PI);
void _40_ACL_Reference(void);
    void Init_DAC(void);
    void Analog_out(int adcvalue);
void _41_ACL_transmit_cycle(void);
    //41 also uses "void Init_DAC(void)"
    //41 also uses "void Analog_out(int adcvalue)"
void _42_ACL_transmit_low(void);
    //42 also uses "void Init_DAC(void)"
    //42 also uses "void Analog_out(int adcvalue)"
void _43_ACL_transmit_mid(void);
    //43 also uses "void Init_DAC(void)"
    //43 also uses "void Analog_out(int adcvalue)"
void _44_ACL_transmit_high(void);
    //44 also uses "void Init_DAC(void)"
    //44 also uses "void Analog_out(int adcvalue)"
void _50_PWM_Reference(void);
void _51_PWM_transmit_cycle(void);
void _52_PWM_transmit_short(void);
void _53_PWM_transmit_mid(void);
void _54_PWM_transmit_long(void);

/*****************************************************************************************************
***              PDU main function                                                                  ***
*****************************************************************************************************/
void main(void){
    int unsigned SCI_in = 0;
    //Start of PortA init for PWM
    RDRIV = 0;
    DDRA = 255;
    PORTA = 0;
    //End of PortA init

    //Start of SCI init
    DDRS=     254;     //1111 1110 Set direction of Port S

    SCI0BDH=  5;       //Baudrate
    SCI0BDL=  32;      //Baudrate

    SCI0CR1=  4;       //0000 0100  Control register
    SCI0CR2=  12;      //0000 1100  Control register
    SCI0SR2=  2;       //0000 0010  Status register
    //End of SCI init
```

```
    while(1){
        if(RDRF){                                   //If SCI input received
            SCI_in = SCI0DRL;                       //Read SCI input
            switch(SCI_in){
                case 10:
                    _10_Reference();
                case 20:
                    _20_LIN_reference();
                case 21:
                    _21_LIN_Respond();
                case 40:
                    _40_ACL_Reference();
                case 41:
                    _41_ACL_transmit_cycle();
                case 42:
                    _42_ACL_transmit_low();
                case 43:
                    _43_ACL_transmit_mid();
                case 44:
                    _44_ACL_transmit_high();
                case 50:
                    _50_PWM_Reference();
                case 51:
                    _51_PWM_transmit_cycle();
                case 52:
                    _52_PWM_transmit_short();
                case 53:
                    _53_PWM_transmit_mid();
                case 54:
                    _54_PWM_transmit_long();
                default:
                    SCI0DRL = 255;
                    while((SCI0SR1 & 64) == 0){}
                    break;
             }
        }
    }
    return;
}  /* End of main */

/*** REFERENCE ********************************************************************************/
void _10_Reference(void){
    int unsigned number_error = 0;
    SCI0DRL = 10;                                   //Confirm function call.
    while((SCI0SR1 & 64) == 0){}

    while(1){}
    return;
}

/*** LIN **************************************************************************************/
void _20_LIN_reference(void){
    SCI0DRL = 20;                                   //Confirm function call.
    while((SCI0SR1 & 64) == 0){}

    //Start of SCI init (needed for LIN)
    SCI0BDH = 0x00;
    SCI1BDH = 0x00;
    SCI0BDL = 0xC3;                                 //Set BR to 2400
    /* Control register 1 */
    SCI0CR1 = 0x00;
    SCI1CR1 = 0x00;
    /* Control register 2, enable reception and transmission */
    SCI0CR2 = 0x0C;
    SCI1CR2 = 0x0C;
    //End of SCI init

    //Start of LIN init
    SCI1CR2 = 0x00;
    DDRS = 4;
    PTS = PTS | 4;                                  //Activate LIN chipset
    //End of init LIN

    while(1){}                                      //Trap
    return;
}

/*** LIN **************************************************************************************/
void _21_LIN_Respond(void){
    int unsigned PI;
    int Checksum;
    SCI0DRL = 21;                                   //Confirm function call.
    while((SCI0SR1 & 64) == 0){}

    //Start of SCI init (needed for LIN)
    SCI0BDH = 0x00;
    SCI1BDH = 0x00;
```

```
    SCI0BDL = 0xC3;                                 //Set BR to 2400
    /* Control register 1 */
    SCI0CR1 = 0x00;
    SCI1CR1 = 0x00;
    /* Control register 2, enable reception and transmission */
    SCI0CR2 = 0x0C;
    SCI1CR2 = 0x0C;
    //End of SCI init

    //Start of LIN init
    SCI1CR2 = 0x00;
    DDRS = 4;
    PTS = PTS | 4;                                  //Activate LIN chipset
    //End of init LIN

    while(1){
        if(CheckHeader(&PI) == 0){                  //If header ok, then...
            PrepareToRespond();                     //...set the SCI module to transmit mode.
            SCI0DRL = PI;                           //Send data.
            while(!(TC)){}

            // Calculate checksum
            Checksum = PI + PI;
            if(Checksum > 255){
                Checksum = Checksum & 255;
           Checksum++;
            }
            Checksum = 255 - Checksum;              //Invert number.

            SCI0DRL = Checksum;                     //Send checksum.
            while(!(TC)){}
        }
    }
    return;
}

void PrepareToRecieve(void){
    int temp;
    temp = SCI0SR1;                                 //Clear flags
    temp = SCI0DRL;
    SCI0CR1 = 0;
    SCI0CR2 = 4;                                    //RE Enable
    return;
}

void PrepareToRespond(void){
    int temp;
    temp = SCI0SR1;                                 //Clear flags
    temp = SCI0DRL;
    SCI0CR1 = 0;
    SCI0CR2 = 8;                                    //TE Enable
    return;
}

int CheckHeader(int unsigned *PI){
    int unsigned i = 0;

    // Break
    SCI0BDL = 0xC3;                                 //Equals BR=2400
    PrepareToRecieve();
    for(i=0; (!(RDRF || FE)) && (i < 3000); i++){}
    if(FE != 0 || i >= 3000)
        return -1;
    if(SCI0DRL != 0)
        return -1;

    // Synch
    SCI0BDL = 0x82;                                  //Equals BR=9600
    PrepareToRecieve();
    for(i=0; (!(RDRF || FE)); i++){}
    if(FE != 0)
        return -1;
    if(SCI0DRL != 0x55)
        return -1;

    // PI
    PrepareToRecieve();
    for(i=0; (!(RDRF || FE)); i++){}
    if(FE != 0)
        return -1;
    *PI = SCI0DRL;

    return 0;
}

/*** ANALOG CURRENT LOOP ********************************************************************/
void _40_ACL_Reference(void){
    int unsigned i;
```

```
    SCI0DRL = 40;                                   //Confirm function call.
    while((SCI0SR1 & 64) == 0){}

    Init_DAC();
    while(1){
         Analog_out(0);
         for (i=0; i<10000; i++){}                  //Interframe spacing for 2kHz readings
    }
    return;
}

void Init_DAC(void){
    MODRR = MODRR & 239;                            //Set MODRR[4] = 0
    SPI0CR1 = 86;
    SPI0CR2 = 16;
    SPI0BR  = 16;                                   //Prescaler = 2
    return;
}

void Analog_out(int adcvalue){
    int unsigned i;
    i = SPI0SR;
    while(!(SPI0SR & 32)){}
    SPI0DR = 33;
    while(!(SPI0SR & 32)){}
    SPI0DR = adcvalue;
    while(!(SPI0SR & 32)){}
    return;
}

/*** ANALOG CURRENT LOOP ************************************************************************/
void _41_ACL_transmit_cycle(void){
    int unsigned i,j;
    SCI0DRL = 41;                                   //Confirm function call.
    while((SCI0SR1 & 64) == 0){}

    Init_DAC();
    while(1){
        for(i = 50;i <= 255;i++){                   //50 for 4.00 mA
            Analog_out(i);
            for (j=0; j<1000; j++){}                //Time between tranmissions for ~2kHz
        }
    }
    return;
}

/*** ANALOG CURRENT LOOP ************************************************************************/
void _42_ACL_transmit_low(void){
    int unsigned i;
    SCI0DRL = 42;                                   //Confirm function call.
    while((SCI0SR1 & 64) == 0){}

    Init_DAC();
    while(1){
        Analog_out(50);                             //50 for 4.00 mA
        for (i=0; i<1000; i++){}                    //Time between tranmissions for ~2kHz
    }
    return;
}

/*** ANALOG CURRENT LOOP ************************************************************************/
void _43_ACL_transmit_mid(void){
    int unsigned i;
    SCI0DRL = 43;                                   //Confirm function call.
    while((SCI0SR1 & 64) == 0){}

    Init_DAC();
    while(1){
        Analog_out(126);                            //126 for 10.03 mA
        for (i=0; i<1000; i++){}                    //Time between tranmissions for ~2kHz

    }
    return;
}

/*** ANALOG CURRENT LOOP ************************************************************************/
void _44_ACL_transmit_high(void){
    int unsigned i;
    SCI0DRL = 44;                                   //Confirm function call.
    while((SCI0SR1 & 64) == 0){}

    Init_DAC();
    while(1){
        Analog_out(210);                            //210 for 16.72 mA
        for (i=0; i<1000; i++){}                    //time between tranmissions for ~2kHz
    }
    return;
}
```

```
/*** PWM ************************************************************************************************/
void _50_PWM_Reference(void){
    SCI0DRL = 50;                                   //Confirm function call.
    while((SCI0SR1 & 64) == 0){}

    while(1){}
    return;
}

/*** PWM ************************************************************************************************/
void _51_PWM_transmit_cycle(void){
    int unsigned i, pwm_lo, pwm_hi;
    SCI0DRL = 51;                                   //Confirm function call.
    while((SCI0SR1 & 64) == 0){}

    while(1){
        for(i = 77;i < 690;i++){
            PORTA = (PORTA | 16);
            for (pwm_hi = 0;pwm_hi < i;pwm_hi++){}
            PORTA = (PORTA & 239);
            for (pwm_lo = 0;pwm_lo < (690 - i);pwm_lo++){}
        }
    }
    return;
}

/*** PWM ************************************************************************************************/
void _52_PWM_transmit_short(void){                  //Generates 10% PWM
    int unsigned pwm_lo, pwm_hi;
    SCI0DRL = 52;                                   //Confirm function call.
    while((SCI0SR1 & 64) == 0){}

    while(1){
        PORTA = (PORTA | 16);
        for (pwm_hi = 0;pwm_hi < 83;pwm_hi++){}
        PORTA = (PORTA & 239);
        for (pwm_lo = 0;pwm_lo < 745;pwm_lo++){}
    }
    return;
}

/*** PWM ************************************************************************************************/
void _53_PWM_transmit_mid(void){                    //Generates 50% PWM
    int unsigned pwm_lo, pwm_hi;
    SCI0DRL = 53;                                   //Confirm function call.
    while((SCI0SR1 & 64) == 0){}

    while(1){
        PORTA = (PORTA | 16);
        for (pwm_hi = 0;pwm_hi < 414;pwm_hi++){}
        PORTA = (PORTA & 239);
        for (pwm_lo = 0;pwm_lo < 414;pwm_lo++){}
    }
    return;
}

/*** PWM ************************************************************************************************/
void _54_PWM_transmit_long(void){                   //Generates 95% PWM
    int unsigned pwm_lo, pwm_hi;
    SCI0DRL = 54;                                   //Confirm function call.
    while((SCI0SR1 & 64) == 0){}

    while(1){
        PORTA = (PORTA | 16);
        for (pwm_hi = 0;pwm_hi < 787;pwm_hi++){}
        PORTA = (PORTA & 239);
        for (pwm_lo = 0;pwm_lo < 41;pwm_lo++){}
    }
    return;
}
```

C Appendix – Emission test procedures

All numbers occurring in this section are decimal if not labeled differently.

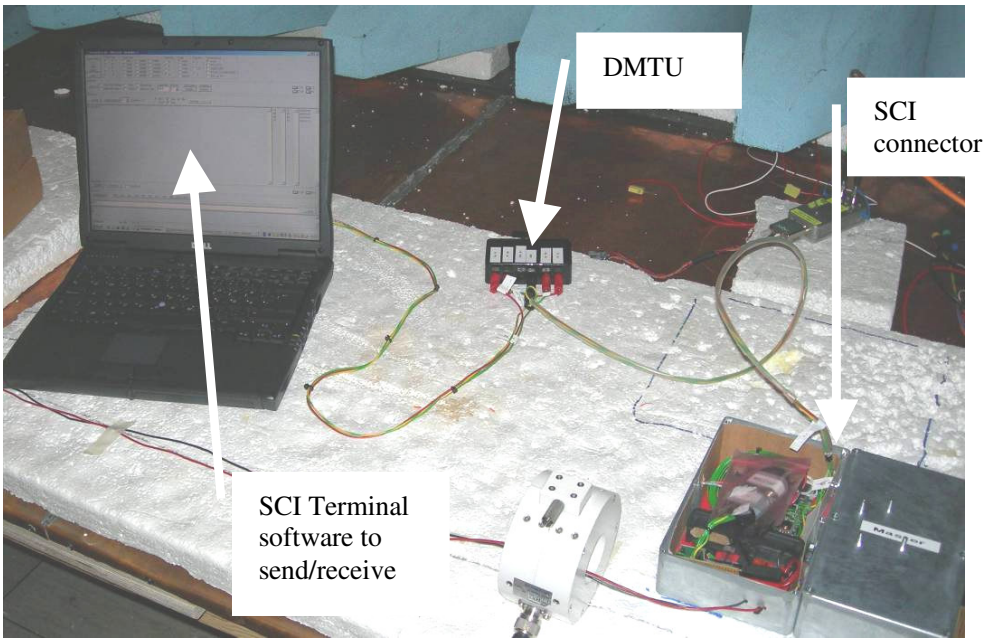


Figure C.1. The picture shows proper use of SCI interface.

Reference measurement

1. M: Demount cover of box.
2. S: Demount cover of box.
3. M: Power off.
4. S: Power off.
5. M: Remove harness inside box.
6. S: Remove harness inside box.
7. M: Connect LIN-harness. **Pay attention to plastic cover around connectors.**
8. S: Connect LIN-harness. **Pay attention to plastic cover around connectors.**
9. S: Connect SCI-connector from PC.
10. S: Power on.
11. S: Transmit ”10” from PC via SCI to Slave.
12. S: Check that response from slave is ”10”.
13. If not, power off slave and repeat #10-12.
14. S: Detach SCI harness.
15. S: Assemble cover of box.
16. M: Connect SCI-connector from PC.
17. M: Check that red switch is in run mode.
18. M: Power on.
19. M: Transmit ”10” from PC via SCI to Master.
20. M: Check that response from master is ”10”.
21. If not, power off master and repeat #16-19.
22. M: Detach SCI harness.
23. M: Attach cover on box.

LIN

Reference measurement

1. M: Demount cover of box.
2. S: Demount cover of box.
3. M: Power off.
4. S: Power off.
5. M: Remove harness inside box.
6. S: Remove harness inside box.
7. M: Connect LIN-harness. **Pay attention to plastic cover around connectors.**
8. S: Connect LIN-harness. **Pay attention to plastic cover around connectors.**
9. S: Connect SCI-connector from PC.
10. S: Power on.
11. S: Transmit ”20” from PC via SCI to Slave.
12. S: Check that response from slave is ”20”.
13. If not, power off slave and repeat #10-12.
14. S: Detach SCI harness.
15. S: Assemble cover of box.
16. M: Connect SCI-connector from PC.
17. M: Check that red switch is in run mode.
18. M: Power on.
19. M: Transmit ”20” from PC via SCI to Master.
20. M: Check that response from master is ”20”.
21. If not, power off master and repeat #11-14.
22. M: Detach SCI harness immediately and note that the red LED is switched off.
23. M: Attach cover on box.

Emission measurement

1. M: Demount cover of box.
2. S: Demount cover of box.
3. M: Power off.
4. S: Power off.
5. M: Remove harness inside box.
6. S: Remove harness inside box.
7. M: Connect LIN-harness. **Pay attention to plastic cover around connectors.**
8. S: Connect LIN-harness. **Pay attention to plastic cover around connectors.**
9. S: Connect SCI-connector from PC.
10. S: Power on.
11. S: Transmit ”21” from PC via SCI to Slave.
12. S: Check that response from slave is ”21”.
13. If not, power off slave and repeat #10-12.
14. S: Detach SCI harness.
15. S: Assemble cover of box.
16. M: Connect SCI-connector from PC.
17. M: Check that red switch is in run mode.
18. M: Power on.
19. M: Transmit ”21” from PC via SCI to Master.
20. M: Check that response from master is ”21”.
21. If not, power off master and repeat #11-14.
22. M: Detach SCI harness immediately and note that the red LED is switched off.
23. M: Attach cover on box.

PSI5

Reference measurement

1. M: Demount cover of box.
2. S: Demount cover of box.
3. M: Power off.
4. S: Power off.
5. M: Remove harness inside box.
6. S: Remove harness inside box.
7. M: Connect PSI5-harness. **Pay attention to plastic cover around connectors.**
8. S: Connect PSI5-sensor.
9. S: Assemble cover of box.
10. M: Connect SCI-connector from PC.
11. M: Check that red switch is in run mode.
12. M: Power on.
13. M: Transmit ”30” from PC via SCI to Master.
14. M: Check that response from master is ”30”.
15. If not, power off master and repeat #11-14.
16. M: Detach SCI harness.
17. M: Attach cover on box.

Emission measurement

1. M: Demount cover of box.
2. S: Demount cover of box.
3. M: Power off.
4. S: Power off.
5. M: Remove harness inside box.
6. S: Remove harness inside box.
7. M: Connect PSI5-harness. **Pay attention to plastic cover around connectors.**
8. S: Connect PSI5-sensor.
9. S: Assemble cover of box.
10. M: Connect SCI-connector from PC.
11. M: Check that red switch is in run mode.
12. M: Power on.
13. M: Transmit ”31” from PC via SCI to Master.
14. M: Check that response from master is ”31”.
15. If not, power off master and repeat #11-14.
16. M: Detach SCI harness.
17. M: Attach cover on box.

ACL

Reference measurement

1. M: Demount cover of box.
2. S: Demount cover of box.
3. M: Power off.
4. S: Power off.
5. M: Remove harness inside box.
6. S: Remove harness inside box.
7. M: Connect ACL -harness. **Pay attention to plastic cover around connectors.**
8. S: Connect ACL -harness. **Pay attention to plastic cover around connectors.**
9. S: Connect SCI-connector from PC.
10. S: Power on.
11. S: Transmit ”40” from PC via SCI to Slave.
12. S: Check that response from slave is ”40”.
13. If not, power off slave and repeat #10-12.
14. S: Detach SCI harness.
15. S: Assemble cover of box.
16. M: Connect SCI-connector from PC.
17. M: Check that red switch is in run mode.
18. M: Power on.
19. M: Transmit ”40” from PC via SCI to Master.
20. M: Check that response from master is ”40”.
21. If not, power off master and repeat #16-19.
22. M: Detach SCI harness.
23. M: Attach cover on box.

Emission measurement of cyclic values

Other emission tests are performed in the same way, simply change the function number below for the one desired.

1. M: Demount cover of box.
2. S: Demount cover of box.
3. M: Power off.
4. S: Power off.
5. M: Remove harness inside box.
6. S: Remove harness inside box.
7. M: Connect ACL-harness. **Pay attention to plastic cover around connectors.**
8. S: Connect ACL-harness. **Pay attention to plastic cover around connectors.**
9. S: Connect SCI-connector from PC.
10. S: Power on.
11. S: Transmit ”41” from PC via SCI to Slave.
12. S: Check that response from slave is ”41”.
13. If not, power off slave and repeat #10-12.
14. S: Detach SCI harness.
15. S: Assemble cover of box.
16. M: Connect SCI-connector from PC.
17. M: Check that red switch is in run mode.
18. M: Power on.
19. M: Transmit ”40” from PC via SCI to Master.
20. M: Check that response from master is ”40”.
21. If not, power off master and repeat #16-19.
22. M: Detach SCI harness.
23. M: Attach cover on box.

PWM

Reference measurement PWM

1. M: Demount cover of box.
2. S: Demount cover of box.
3. M: Power off.
4. S: Power off.
5. M: Remove harness inside box.
6. S: Remove harness inside box.
7. M: Connect PWM -harness. **Pay attention to plastic cover around connectors.**
8. S: Connect PWM -harness. **Pay attention to plastic cover around connectors.**
9. S: Connect SCI-connector from PC.
10. S: Power on.
11. S: Transmit ”50” from PC via SCI to Slave.
12. S: Check that response from slave is ”50”.
13. If not, power off slave and repeat #10-12.
14. S: Detach SCI harness.
15. S: Assemble cover of box.
16. M: Connect SCI-connector from PC.
17. M: Check that red switch is in run mode.
18. M: Power on.
19. M: Transmit ”50” from PC via SCI to Master.
20. M: Check that response from master is ”50”.
21. If not, power off master and repeat #16-19.
22. M: Detach SCI harness.
23. M: Attach cover on box.

Emission measurement of cyclic values

Other emission tests are performed in the same way, simply change the function number below for the one desired.

1. M: Demount cover of box.
2. S: Demount cover of box.
3. M: Power off.
4. S: Power off.
5. M: Remove harness inside box.
6. S: Remove harness inside box.
7. M: Connect PWM-harness. **Pay attention to plastic cover around connectors.**
8. S: Connect PWM-harness. **Pay attention to plastic cover around connectors.**
9. S: Connect SCI-connector from PC.
10. S: Power on.
11. S: Transmit ”51” from PC via SCI to Slave.
12. S: Check that response from slave is ”51”.
13. If not, power off slave and repeat #10-12.
14. S: Detach SCI harness.
15. S: Assemble cover of box.
16. M: Connect SCI-connector from PC.
17. M: Check that red switch is in run mode.
18. M: Power on.
19. M: Transmit ”50” from PC via SCI to Master.
20. M: Check that response from master is ”50”.
21. If not, power off master and repeat #16-19.
22. M: Detach SCI harness.
23. M: Attach cover on box.

D Appendix – Susceptibility test procedures

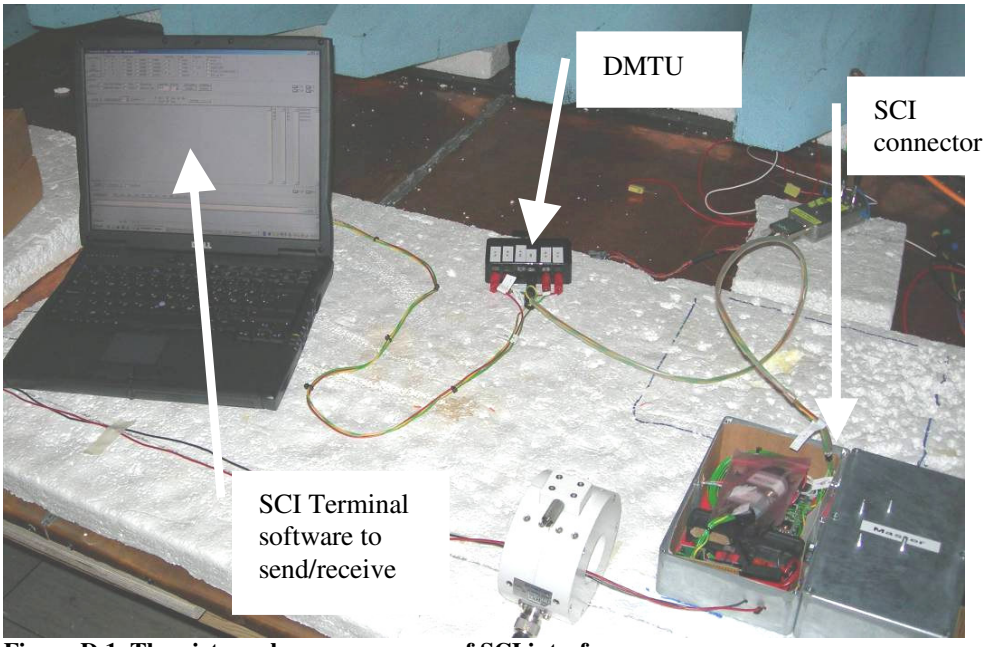


Figure D.1. The picture shows proper use of SCI interface.

LIN

1. M: Demount cover of box.
2. S: Demount cover of box.
3. M: Power off.
4. S: Power off.
5. M: Remove harness inside box.
6. S: Remove harness inside box.
7. M: Connect LIN-harness. **Pay attention to plastic cover around connectors.**
8. S: Connect LIN-harness. **Pay attention to plastic cover around connectors.**
9. S: Connect SCI-connector from PC.
10. S: Power on.
11. S: Transmit ”21” from PC via SCI to Slave.
12. S: Check that response from slave is ”21”.
13. If not, power off slave and repeat #10-12.
14. S: Detach SCI harness.
15. S: Assemble cover of box.
16. M: Connect SCI-connector from PC.
17. M: Check that red switch is in run mode.
18. M: Power on.
19. M: Transmit ”21” from PC via SCI to Master.
20. M: Check that response from master is ”21”.
21. If not, power off master and repeat #11-14.
22. M: Detach SCI harness immediately and note that the red LED is switched off.
23. M: Attach cover on box.

PSI5

1. M: Demount cover of box.
2. S: Demount cover of box.
3. M: Power off.
4. S: Power off.
5. M: Remove harness inside box
6. S: Remove harness inside box.
7. M: Connect PSI5-harness. **Pay attention to plastic cover around connectors.**
8. S: Connect PSI5-sensor.
9. S: Assemble cover of box.
10. M: Connect SCI-connector from PC
11. M: Check that red switch is in run mode.
12. M: Power on.
13. M: Transmit ”31” from PC via SCI to Master.
14. M: Check that response from master is ”31”.

15. If not, power off master and repeat #11-14.
16. M: Detach SCI harness.
17. M: Attach cover on box.

ACL (Medium current)

Other susceptibility tests are performed in the same way, simply change the function number below for the one desired.

1. M: Demount cover of box.
2. S: Demount cover of box.
3. M: Power off.
4. S: Power off.
5. M: Remove harness inside box.
6. S: Remove harness inside box.
7. M: Connect ACL-harness. **Pay attention to plastic cover around connectors.**
8. S: Connect ACL-harness. **Pay attention to plastic cover around connectors.**
9. S: Connect SCI-connector from PC.
10. S: Power on.
11. S: Transmit ”43” from PC via SCI to Slave.
12. S: Check that response from slave is ”43”.
13. If not, power off slave and repeat #10-12.
14. S: Detach SCI harness.
15. S: Assemble cover of box.
16. M: Connect SCI-connector from PC.
17. M: Check that red switch is in run mode.
18. M: Power on.
19. M: Transmit ”43” from PC via SCI to Master.
20. M: Check that response from master is ”43”.
21. If not, power off master and repeat #16-19.
22. M: Detach SCI harness.
23. M: Attach cover on box.

PWM (Medium duty cycle)

Other susceptibility tests are performed in the same way, simply change the function number below for the one desired.

1. M: Demount cover of box.
2. S: Demount cover of box.
3. M: Power off.
4. S: Power off.
5. M: Remove harness inside box.
6. S: Remove harness inside box.
7. M: Connect PWM -harness. **Pay attention to plastic cover around connectors.**
8. S: Connect PWM -harness. **Pay attention to plastic cover around connectors.**
9. S: Connect SCI-connector from PC.
10. S: Power on.
11. S: Transmit ”53” from PC via SCI to Slave.
12. S: Check that response from slave is ”53”.
13. If not, power off slave and repeat #10-12.
14. S: Detach SCI harness.
15. S: Assemble cover of box.
16. M: Connect SCI-connector from PC.
17. M: Check that red switch is in run mode.
18. M: Power on.
19. M: Transmit ”53” from PC via SCI to Master.
20. M: Check that response from master is ”53”.
21. If not, power off master and repeat #16-19.
22. M: Detach SCI harness.
23. M: Attach cover on box.

E Appendix – Complete EMC emission test results

This appendix contains a compilation of the test results. All graphs were automatically produced by the laboratory's equipment. Red line in figures represents the Volvo maximum emissions requirement.

Emission testing

All graphs were automatically produced by the laboratory's equipment. Red line in figures represents the Volvo maximum emissions requirement.

References

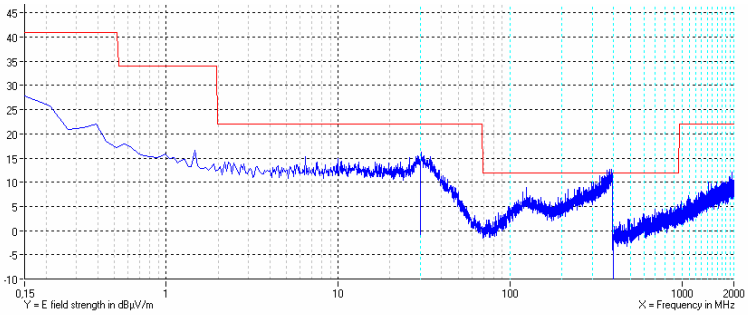


Figure E.1. Ambient emissions, both master and slave were switched off. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Horizontal polarization.

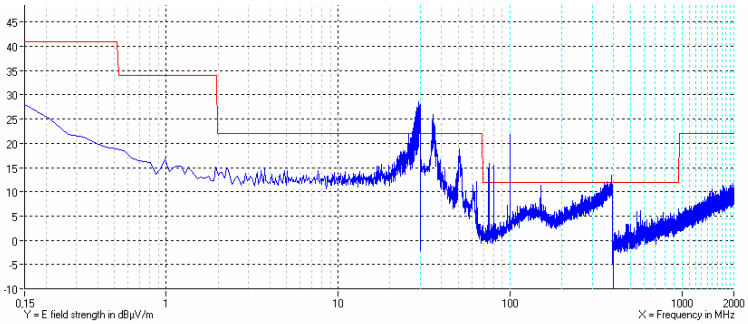


Figure E.2. Reference emissions, both master and slave were running function #10, and connected with LIN harness. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Horizontal polarization.

LIN

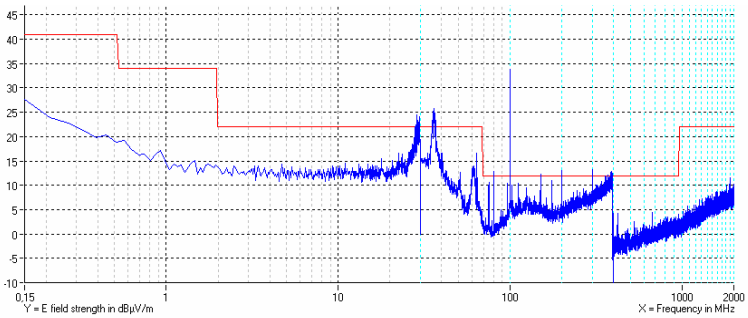


Figure E.3. LIN reference emissions, both master and slave were running function #20. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Horizontal polarization.

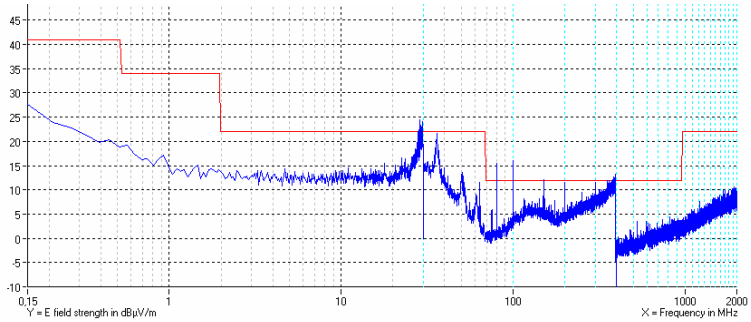


Figure E.4. LIN reference emissions, both master and slave were running function #20. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Vertical polarization.

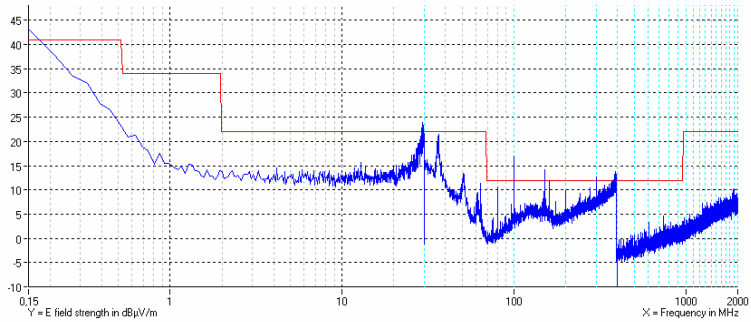


Figure E.5. LIN communication emissions, both master and slave were running function #21. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Horizontal polarization.

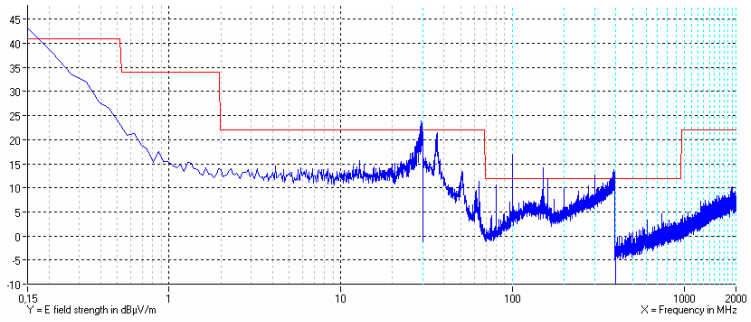


Figure E.6. LIN communication emissions, both master and slave were running function #21. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Vertical polarization.

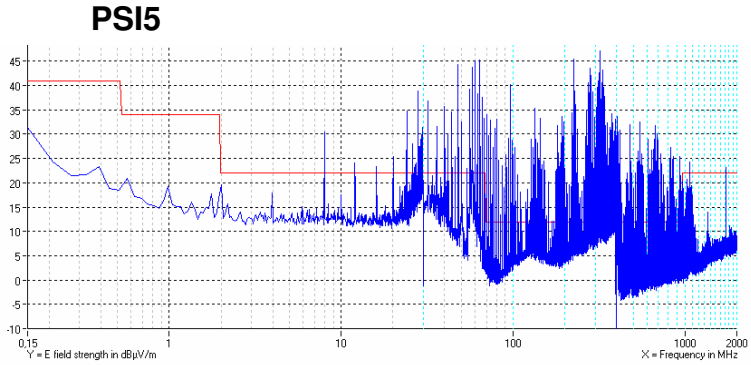


Figure E.7. PSI5 reference emissions, master was running function #30. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Horizontal polarization.

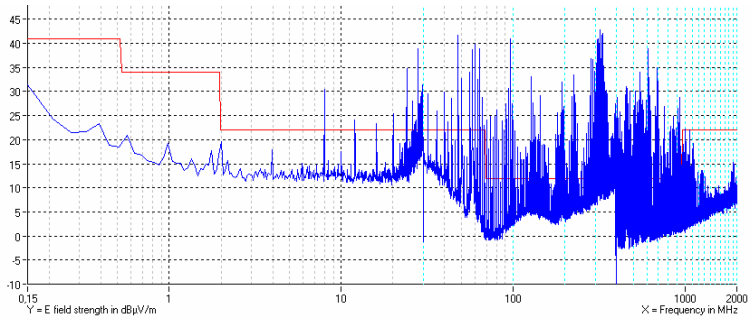


Figure E.8. PSI5 reference emissions, master was running function #30. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Vertical polarization.

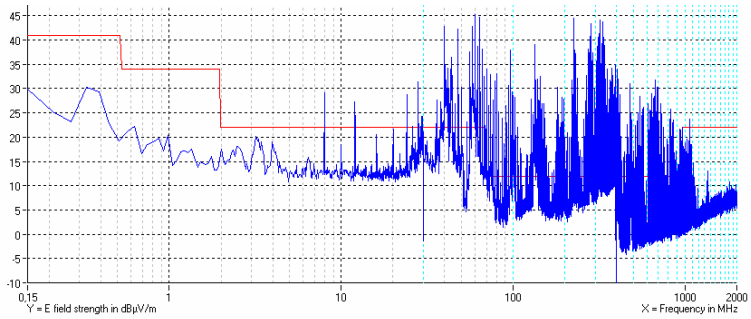


Figure E.9. PSI5 communication emissions, master was running function #31. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Horizontal polarization.

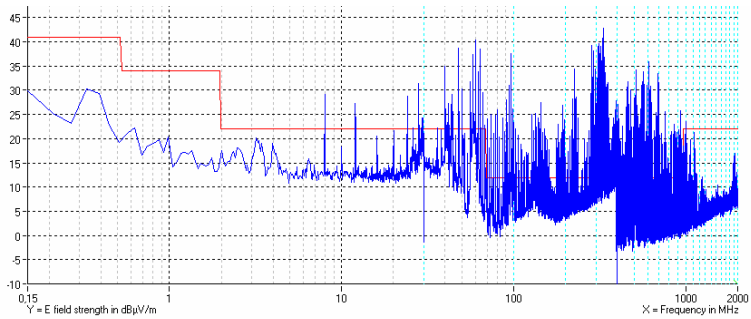


Figure E.10. PSI5 communication emissions, master was running function #31. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Vertical polarization.

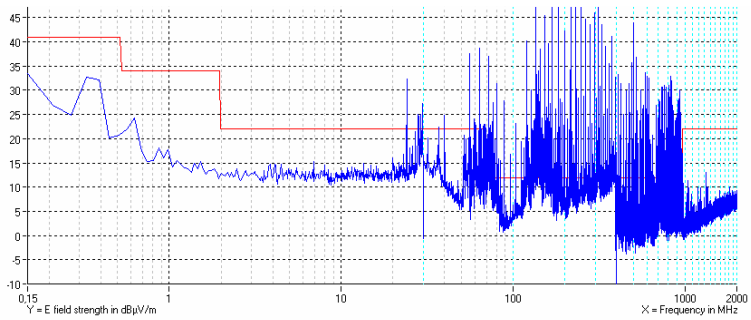


Figure E.11. Bonus measurement with Bosch Smartbox replacing master. PSI5 communication emissions. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Vertical polarization.

Figure E.12. The test setup with Bosch Smartbox.

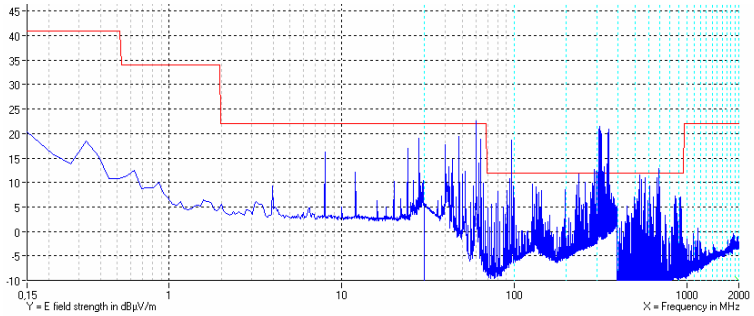


Figure E.13. PSI5 Average measurement 0.15-2000MHz. Vertical polarization.

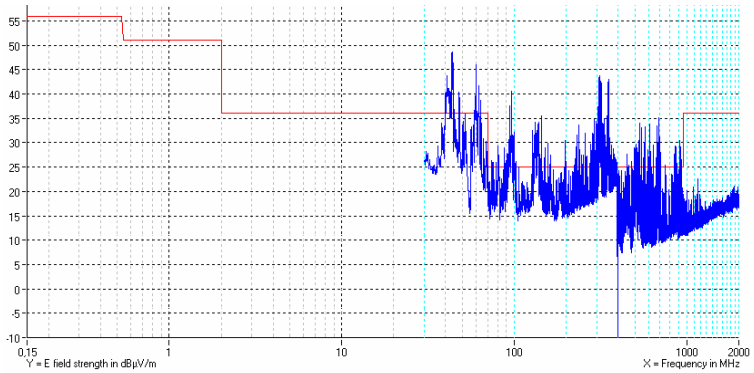


Figure E.14. PSI5 Broadband measurement 30-2000MHz. Vertical polarization.

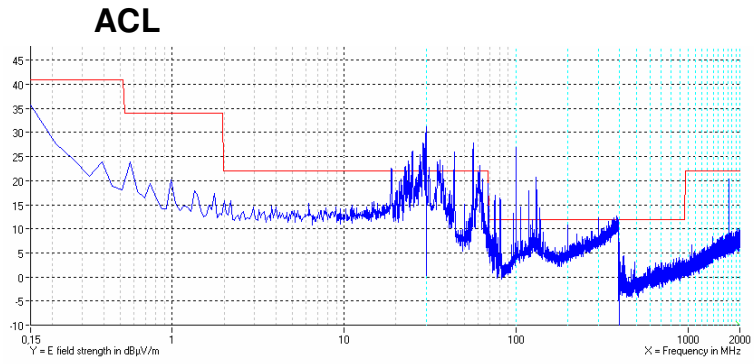


Figure E.15. ACL reference emissions, master and slave were running function #40. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Horizontal polarization.

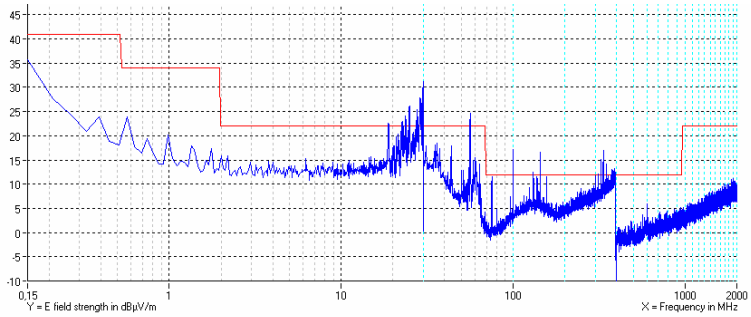


Figure E.16. ACL reference emissions, master and slave were running function #40. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Vertical polarization.

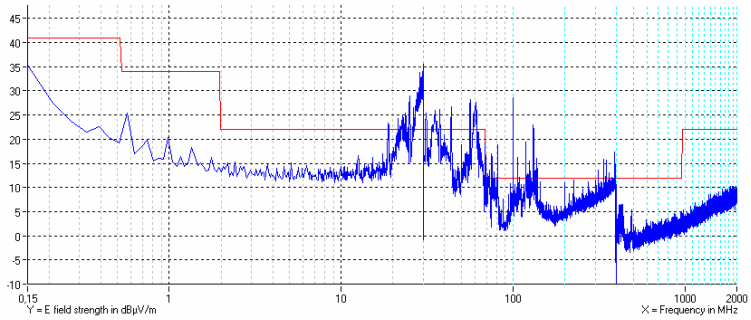


Figure E.17. ACL communication emissions, master was running function #40, slave function #41. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Horizontal polarization.

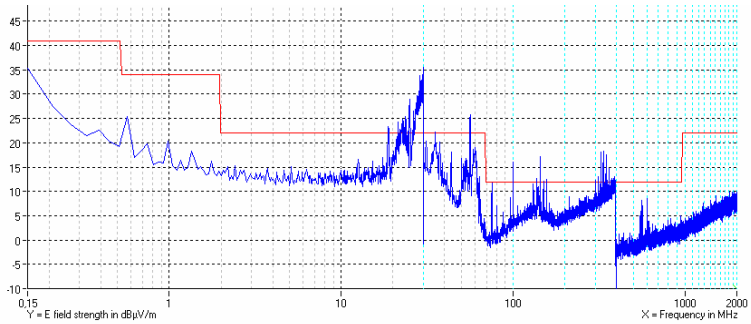


Figure E.18. ACL communication emissions, master was running function #40, slave function #41. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Vertical polarization.

PWM without protective circuits

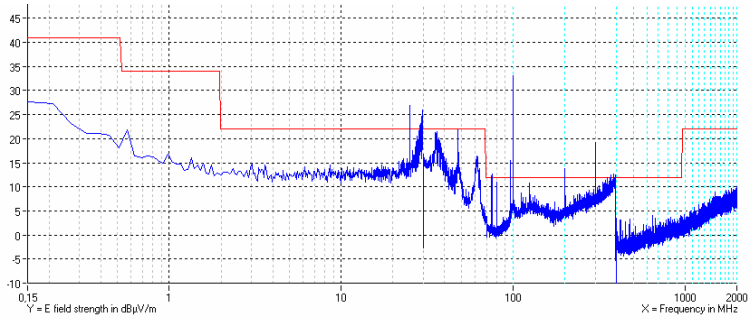


Figure E.19. PWM reference emissions, master and slave were running function #50. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Horizontal polarization.

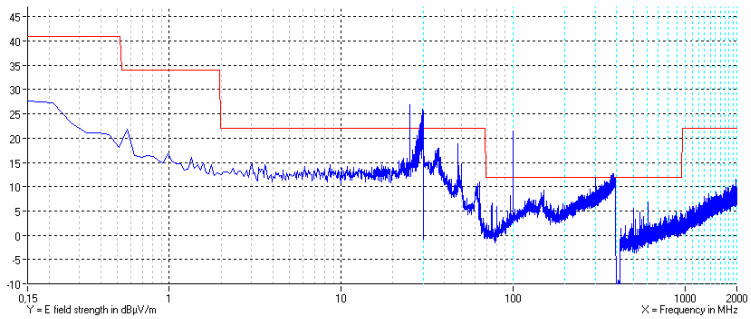


Figure E.20. PWM reference emissions, master and slave were running function #50. Laborative error at 400-405MHz, plot info at those frequencies isn't valid. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Vertical polarization.

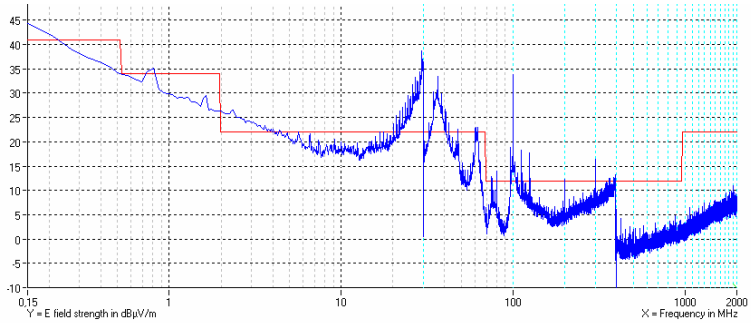


Figure E.21. PWM communication emissions, master was running function #50, slave function #51. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Horizontal polarization.

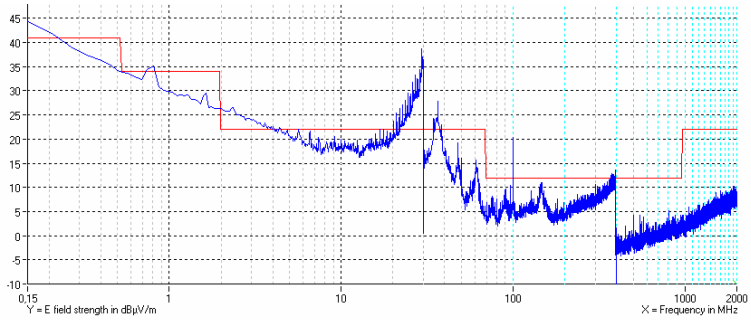


Figure E.22. PWM communication emissions, master was running function #50, slave function #51. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Vertical polarization.

PWM with protective circuits

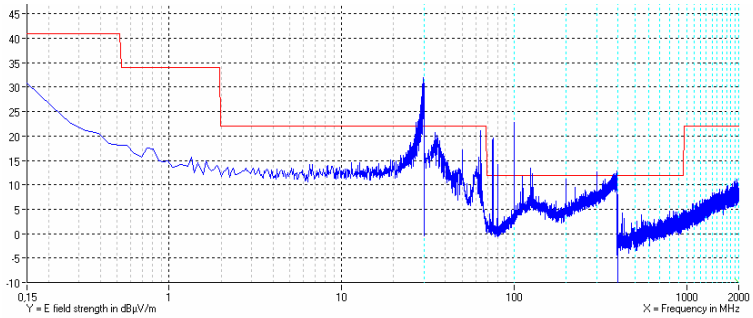


Figure E.23. PWM reference emissions, master and slave were running function #50. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Horizontal polarization.

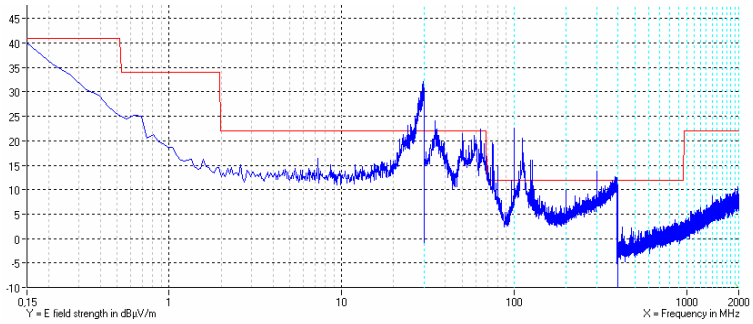


Figure E.24. PWM communication emissions, master was running function #50, slave function #51. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Horizontal polarization.

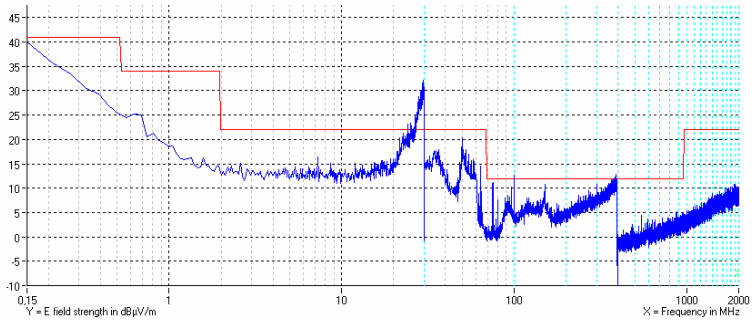


Figure E.25. PWM communication emissions, master was running function #50, slave function #51. X-axis unit: frequency in MHz. Y-axis unit: E field strength in dBµV/m. Vertical polarization.

F Appendix – Schematics

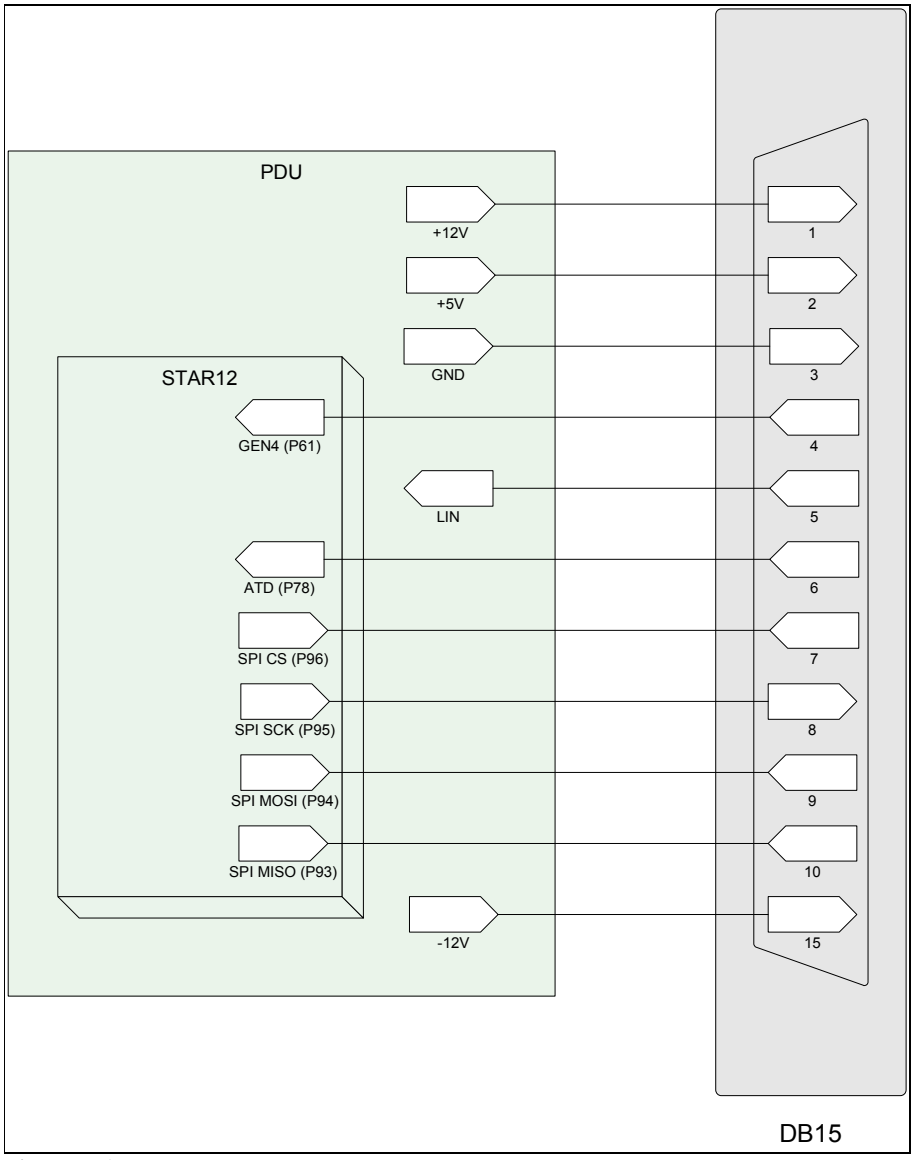


Figure F.1. Master PDU harness.

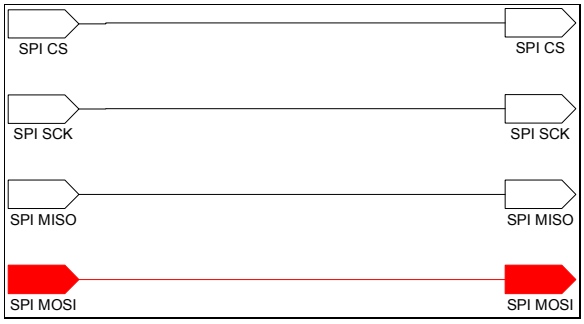


Figure F.2. Master SPI harness

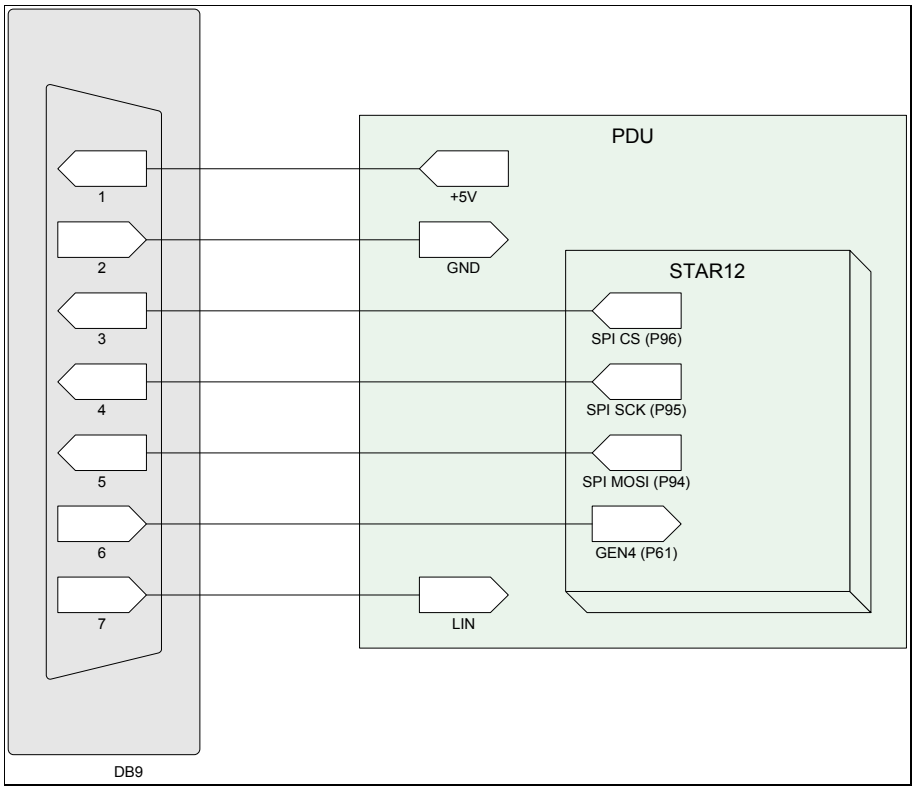


Figure F.3. Slave PDU harness.

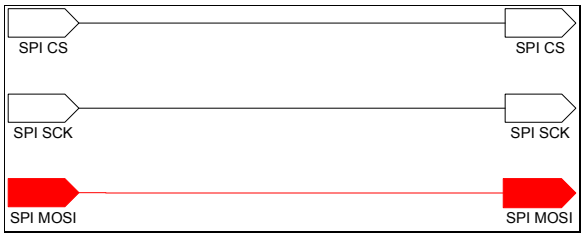


Figure F.4. Slave SPI harness.

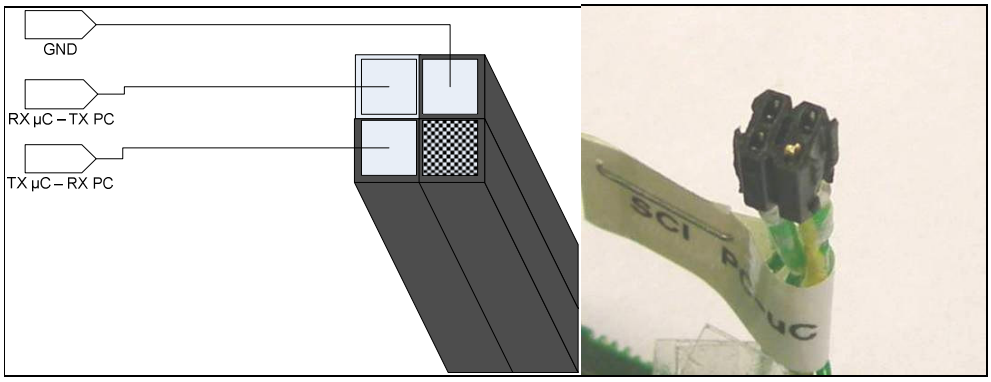


Figure F.5. SCI connector in PDU's.

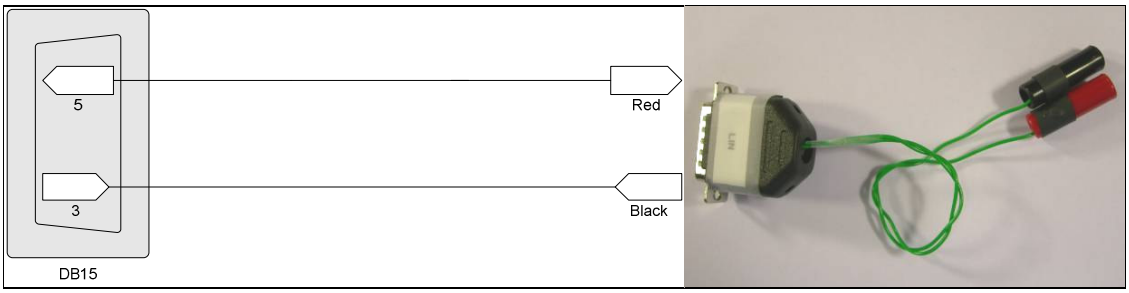


Figure F.6. LIN master harness.

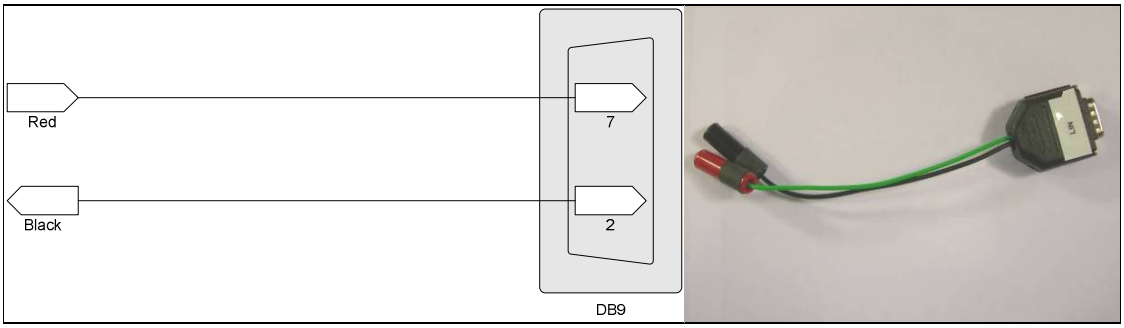


Figure F.7. LIN slave harness.

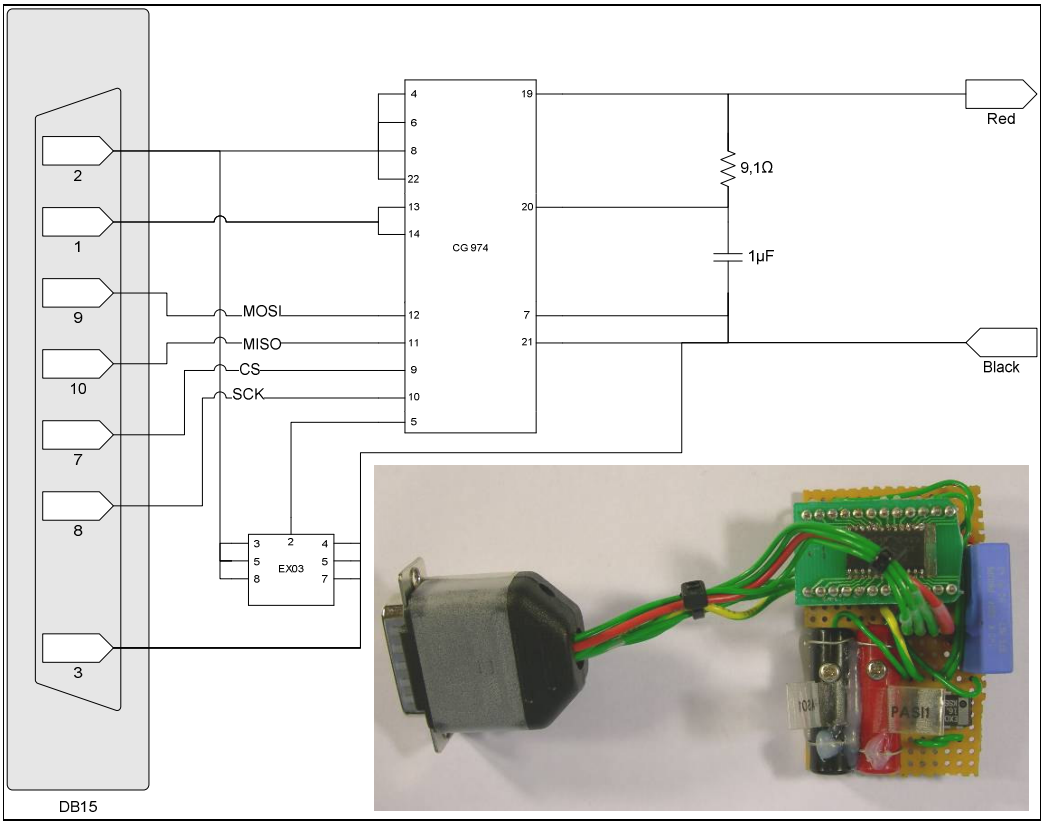


Figure F.8. PSI5 master module.

Figure F.9. PSI5 slave module.

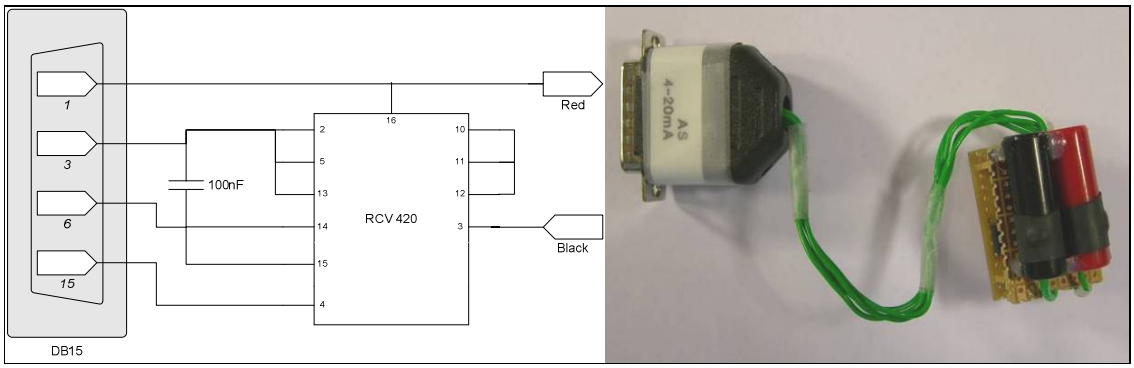


Figure F.10. ACL master module.

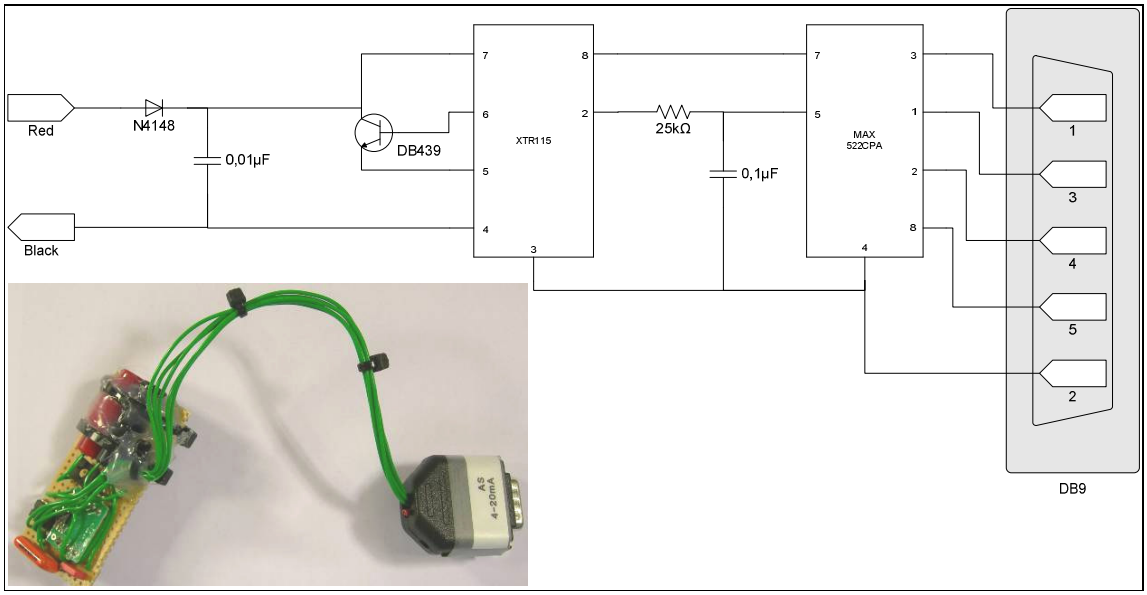


Figure F.11. ACL slave module.

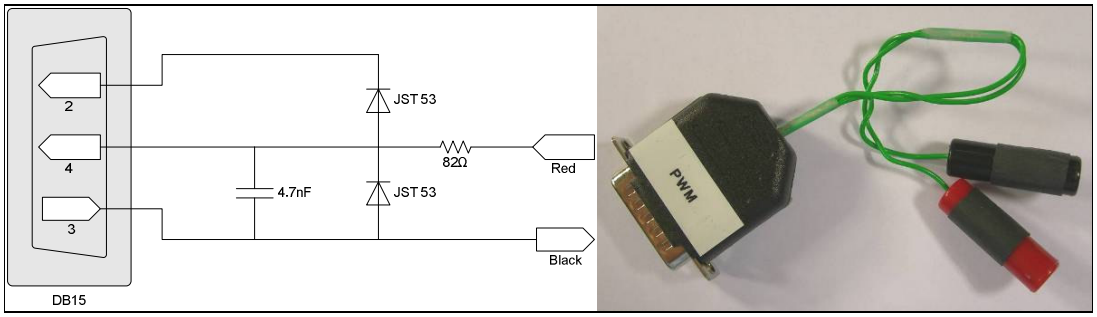


Figure F.12. PWM master harness.

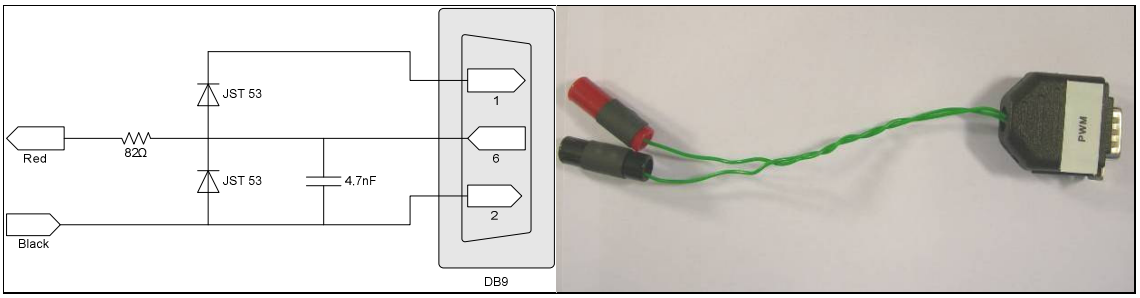


Figure F.13. PWM slave harness.

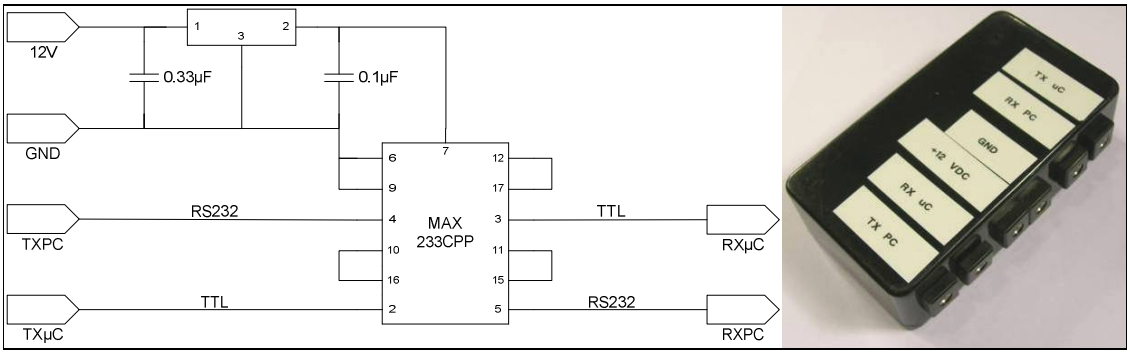


Figure F.14. DMTU.