

Automotive Data Logger

Simon Hodgson (9940447)

ELD030

Final Year Project Report

MEng Electronic and Computer Systems Engineering

Supervisor: Mr Bill Gabb

Department of Electronic and Electrical Engineering

Loughborough University

May 2003

ABSTRACT

The majority of modern motor vehicles feature digital communication networks as an integrated part of their design. These networks provide the connection between various electronic systems and provide easy access to many varied parameters related to the vehicles performance. The Automotive Data Logger is an attempt to access and record this information, over a period, allowing access at some point in the future. Whilst providing an additional analogue interface for the collection of parameters not available via this information network.

TABLE OF CONTENTS

Overview and Project Aims	1
Project Aims	1
Possible applications	1
CAN and it's use in the Automotive Industry.....	2
Why has CAN been selected for Automotive Applications?.....	2
What features are usually found in CAN connected automotive systems?	2
Why do modern cars require data networks?	3
Electronic Stability Programme (ESP)	4
The future of data networks in Automotive Applications.....	5
Other applications of CAN	6
Brief Outline of CAN Protocol.....	6
How CAN fits into the OSI 7 Layer Communications Network Model.....	6
CAN Physical Layer	7
CAN Message Frame format	9
Error Frame Format	10
Overload Frame Format	11
Hardware Design - Block Diagram	12
Hardware Design Description	13
Schematic Design	13
Construction	13
Power Supply	13
Microcontroller	14
RS-232	14
CAN.....	15
Flash RAM	15
LCD	15
Input Buttons.....	16
Accelerometers	16
iMEMS technology.....	17
Software.....	19
Software Development and Configuration Control.....	19
Embedded Real-time Software	19
Log CAN Data	20
Log ADC Data	21
LCD Manager	21

IO Manager	21
UART Manager	22
DataFlash Operation	22
Interrupts.....	23
Real-time software – CAN Test utility	23
PC Software – Download Application	24
PC Software – Matlab Routines	25
ADC data processing	25
CAN Data processing.....	25
Data storage format	25
EEPROM Index Format.....	26
DataFlash Format	26
Serial Communications Protocol.....	26
Software Development Tools Used	27
Compiler	27
Programmer	27
Source Code Control	27
Experimentation and Proving of Design	29
Testing Overview.....	29
Testing of the Analogue Accelerometer Sensors	30
Collection of sample CAN data.....	32
Problems Encountered During Testing.....	32
CAN Message IDs present	32
Interpretation of the Data	33
Sample of information downloaded from Data Logger	33
Project Management.....	36
Time.....	36
Finance.....	37
Further work	40
Known Problems	40
Suggested Extensions	40
Bibliography	i
Acknowledgements.....	ii
Appendixes	iii
List of acronyms used in this document	iii
Source Code.....	iii
Schematics.....	iii

OVERVIEW AND PROJECT AIMS

Project Aims

The original aim of the project was to construct an Automotive Data Logger with the following features:

- CAN Bus Interface
- Analogue signal interface
- At least 16Mb of on-board Flash Memory
- PC Interface to allow download of data
- Powered from the 12v power source in a car

The CAN interface was selected as it is found on many modern cars, and allows easy access to the wide range of sensors already fitted to the modern motor vehicle. The analogue interface is to allow for the addition of users own sensors should they find the information they require is not available via the CAN bus.

Possible applications

The Data Logger is designed to be as versatile as possible, making it suitable for a number of different uses, these could include:

- Digital replacement for the Tachograph Disc recorder used in commercial vehicles. This would be much more difficult to tamper with than the current mechanical models.
- Automotive equivalent of the Black Box recorder as fitted to modern aircraft. If fitted to all cars it could provide the police and insurance companies with information on how the car was being driven before an accident, along with some environmental data –for example the temperature may have been recorded as below freezing possibly making the road conditions icy.
- Performance Tuning – The CAN bus provides easy access to many engine and braking parameters, this information could be useful to people race tuning ordinary road cars.
- Recording Diagnostic Information, CAN bus is increasing being used to replace the propriety diagnostic information systems currently found on many vehicles, this information could provide valuable diagnostic information to technicians attempting to repair faults on cars.
- Driving style analysis, for driving instructors and racing drivers. The datalogging of steering angle, vehicle speed and engine rpm would make it possible to ascertain for example, whether the driver was changing gear to late or too early, when going into a bend.

CAN and it's use in the Automotive Industry

Why has CAN been selected for Automotive Applications?

There are a number of reasons why CAN has been selected for use in automotive applications, these include:

- High noise immunity – the twisted pair data bus wires provides good immunity to electrical noise, CAN must operate in engine bays which are electrically noisy environments.
- Excellent error detection – should the system be affected by noise, parity and other error checks will detect most errors present. Statistically CAN will only miss one error in every 1 billion operating hours¹. This is important as CAN is frequently used to connect safety critical systems such as ABS, EBD etc.
- Fault Isolation – Nodes have a BIT capability enabling them to detect when they are at fault and subsequently disable themselves – leaving the other nodes on the bus running.
- Mature technology – the CAN protocol is nearly 20 years old, making it proven technology, therefore reducing the risk to engineering projects introducing it into their products.
- Wide Support – CAN Controllers are found integrated into many Microcontrollers and Microprocessors. There are also a large number of development tools on the market – for example CAN bus analysers.

What features are usually found in CAN connected automotive systems?

CAN is put to use in a wide variety of applications within modern cars. The Audi A4 used for project testing has 3 separate CAN buses, these are:

- Drive Train, a 500Kbps bus to which the following systems are connected:
 - Engine Control
 - ABS Braking
 - Steering Column
 - Dashboard
 - Automatic Gearbox (if fitted)
- Convenience 100 kBit/s low speed CAN, Air conditioning, electric windows & sunroof, security (keyless entry, alarm system)
- Infotainment (GPS and navigation systems, CD, Radio, DVD etc)

¹ Source: Atmel CAN tutorial

Why do modern cars require data networks?

Modern motor cars are not only complicated mechanical systems, but also rely on many complex electronic systems. These systems bring a number of improvements:

- Comfort for passengers and the driver. Systems such as active roll control and air suspension are designed to increase passenger comfort and reduce road noise.
- Increase in fuel efficiency and decrease in harmful pollutant emissions. Precise digital control over the fuel and air mixture as well as ignition timing ensures that the fuel is burnt as efficiently as possible thereby increasing the efficiency of the engine and reducing the emissions it produces.
- Increase in component life. On board monitoring of the engine and other components can alert the driver to potential problems. This allows the driver to have the any problems investigated immediately, rather than allowing further deterioration followed by complete failure thus rendering the car unserviceable. Careful monitoring of the engine parameters has also lead to the introduction of variable servicing intervals, meaning services are carried out when they are actually needed rather than at set mileages or times. This generally means the vehicle will be serviced fewer times over it's life saving time, resources and reducing the amount of environmentally unfriendly oils and fluids to be disposed of, and making the vehicle more reliable.
- Most importantly – Greater safety for the driver and their passengers, examples of network connected safety systems include:
 - ABS – Anti lock braking system, releases and reapplies the brakes in quick succession so that the driver can maintain directional control of the vehicle whilst maintaining full brake pressure.
 - Airbags – When the rapid deceleration of a crash is detected the vehicle's airbags deploy before the driver and passengers impact with the steering wheel and dashboard in front of them. Modern cars may also be fitted with side airbags – the car is able to detect the direction of impact and deploy the relevant airbags. At the same time the airbag is deployed a seatbelt pre-tensioning system may also be triggered, this is a small explosive charge in the seatbelt spool which tightens the seatbelt before the occupant is thrown forwards. Finally, many cars may also automatically turn on the hazard warning lights and unlock any locked doors to permit rapid escape.

The electronic systems are typically arranged as a series of Electronic Control Units (ECUs) each of which performs a specific function.

Most ECUs are able to perform their primary functions without communication with the other ECUs for example the engine ECU collects data from the engine and performs the engine control function without the need for data from the other ECUs.

However the addition of a digital network allows a large number of parameters from the vehicle measured by a series of different ECUs to be exchanged in real-time. One example being the dashboard ECU, this collects data from several units and displays it for the driver, e.g. engine RPM from the engine ECU and speed from the braking ECU.

Electronic Stability Programme (ESP)

ESP provides an excellent example of how data networks in modern cars are improving safety. Invented by Bosch, it has become available in new cars in the last few of years. Although initially introduced into higher end models, it is likely it will filter down to even the entry level models over the coming years.

ESP is essentially a software system that builds upon two safety systems already commonly found on many modern cars, these systems being:

- ABS – Antilock Braking System, aims to prevent wheels locking when the driver applies full braking force on low-friction road surfaces, by controlling the brake pressure.
- Traction Control – Aims to prevent wheel spin when the driver applies too much engine torque on slippery road surfaces, by controlling the engine output either from a standing start or whilst turning through a bend.

The aim of ESP is to keep the vehicle travelling in the direction the driver has commanded. For example if a driver makes a sudden lane change when travelling at high speed on a motorway (perhaps to avoid an obstacle in the road) there is a tendency for the car to over steer causing the vehicle to spin round into the path of oncoming traffic. ESP collects data from the following sources:

- Wheel speed sensors
- Steering Angle sensors
- Yaw rate sensor
- Accelerator Position
- Brake Pressure

All of this information is collated by the ESP ECU, from data transmitted over the cars drive train CAN bus from several separate ECUs.

From this information, the ESP system can deduce where the driver would like the vehicle to go and where it is actually going. In order to correct the difference between the two vehicle positions EPS is able to apply corrective steering by actuating each of the wheel brakes separately, in some situations it may also alter engine torque.

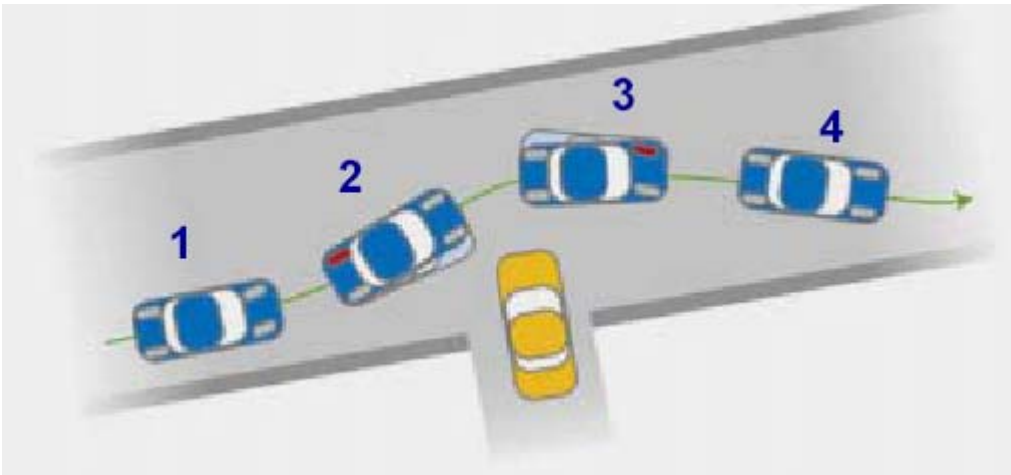


Figure 1 - ESP in action, avoiding an obstacle²

Figure 1 demonstrates how ESP takes control of the vehicle helping the driver avoid an obstacle:

1. The driver applies the brakes, and makes a sudden steering command to the left. There is a tendency for the vehicle to under steer and attempt to carry straight on.
2. The ESP recognises that the vehicle is under steering and applies the back left brake as a corrective measure. The vehicle goes in the direction the driver commanded.
3. There is now a tendency for the driver to over steer to the right, causing the vehicle to spin. The ESP again recognises this and this time applies the front left brake to correct the over steer.
4. The vehicle is stable once again.

The future of data networks in Automotive Applications

As the design of cars progresses communications networks are being increasingly utilised for varying aspects of in-vehicle systems. One of the principle reasons for doing this is to allow the designers to easily multiplex several signals onto a single wire – thereby decreasing the amount of cabling that has to be installed in a vehicle. The move towards the increased use of networks for automotive applications has been brought about by the ever-decreasing cost of microprocessors, and the increasing need for data information exchange between various components of the vehicles electronics system.

Three other networking systems, currently being deployed in new vehicle designs, are worth mentioning here:

- LIN (Local Interconnect Network) Bus – This is a low data rate (20Kbit) single wire system, that works with a single master transmitter and many slave receivers. It is being used to multiplex signals such as the brake lights and indicator lights onto a single wire.

² Source: Bosch Website

- MOST (Media Oriented Systems Transport)– This is a very high speed network (up to 24Mbit) the physical layer being fibre optic. It is being deployed in infotainment applications to carry information such as compressed video. Typical devices that may connect to a MOST bus system include DVD player, GPS, Digital Radio, CD Changer and mobile phone.
- TTP (Time Triggered Protocol) – This protocol is currently being hailed as the replacement for CAN. Devices on the bus are each allocated specific time slots in which they may transmit data – this replaces the bus arbitration access mechanism used by CAN with one that can guarantee access to the bus at specific times. This is especially important use in the ‘drive-by-wire’ systems currently in development, where the driving controls are no longer mechanically linked to the wheels, engine and brakes.

Other applications of CAN

As a networking protocol CAN has also found it’s way in to many other applications, these include railway, industrial control and automation, building automation, maritime and medial applications. It would also be feasible to use the Data Logger in these applications.

Brief Outline of CAN Protocol

How CAN fits into the OSI 7 Layer Communications Network Model

The OSI (Open Systems Interconnection) model defines a standard model for computer networks. The vast majority of network protocols can be fitted onto this model.

7. Application Layer	Implemented by HLPs
6. Presentation Layer	Partially Implemented by Higher Level Protocols
5. Session Layer	
4. Transport Layer	
3. Network Layer	
2. Data Link Layer	CAN Protocol
1. Physical Layer	

Table 1 - CAN and the OSI 7 Layer Model

The CAN protocol itself only actually implements the lowest two layers of the OSI 7 layer model. The physical layer defines the way bits are physically formatted for transmission over the physical medium. The Data Link defines the format of the lowest level of data packet, such as how the packer header is formatted, and the number of bytes in each packet.

In CAN networks, it is common to run the application straight on top of these two layers. However there are also a number of higher-level protocols that have been defined. These protocols simplify using CAN for more complicated applications – for example they permit the

transmission of streams of data bigger than 8 bytes, handling the splitting into smaller packets for transmission over the CAN network, and the re-assembly of the packets and data on the receiver.

Some examples of higher-level protocols used with CAN include:

- CANopen
- DeviceNet
- CAN Kingdom
- OSEK/VDX
- SAE J1939
- Smart Distributed System

Some of these higher-level protocols are part of wider specification that includes operating systems and network management. One of the aims of higher-level protocols is to increase the operability between ECUs from different manufactures, increasing commonality and reducing development effort for new vehicles and systems through code re-use. This is becoming increasingly important as the complexity of automotive electronic systems increases.

As the Data Logger only implements the capture of data at the Data Link Layer of the network, further detail of these protocols is outside the scope of this document.

CAN Physical Layer

The most common transmission medium for CAN bus is a twisted pair of wires, although it has been demonstrated operating over single wire, fibre optic and radio. The main advantage of using twisted pair is it's excellent noise immunity, this is especially important in automotive applications as the numerous electronic and electrical systems in today's modern vehicles generate high levels of electrical noise allowing possible EMI problems to arise.

Data is transmitted over the physical channel using Non-Return to Zero (NRZ) bit stuffed encoding. A stuff bit is inserted when five consecutive bits have the same value. This provides the receiver with sufficient edges to perform bit level synchronisation.

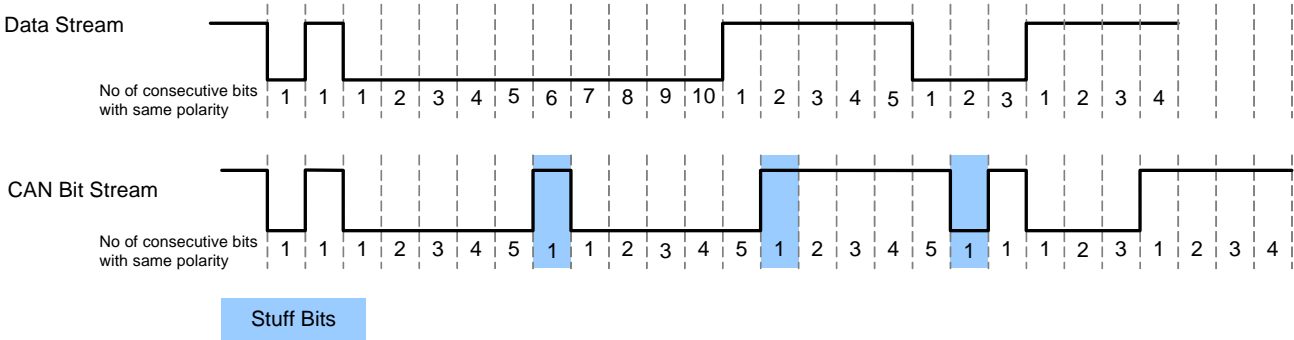


Figure 2 - CAN Bit Stuffing³

Rather than the normal convention of 0's and 1's the CAN protocol defines a dominant (equivalent to logic 0) and a recessive state (equivalent to logic 1) these states are used to implement a 'wired AND' access mechanism.

This wired AND system operates during the arbitration stage of message transmission, when stations on the network are competing for access to the bus. Where two stations attempt simultaneous access to the bus, the station transmitting the dominant state (logic zero) will win access to the bus. The other stations will then only be able to listen to the bus, and must attempt to re-send their message once the bus is free again, as illustrated in Figure 3.

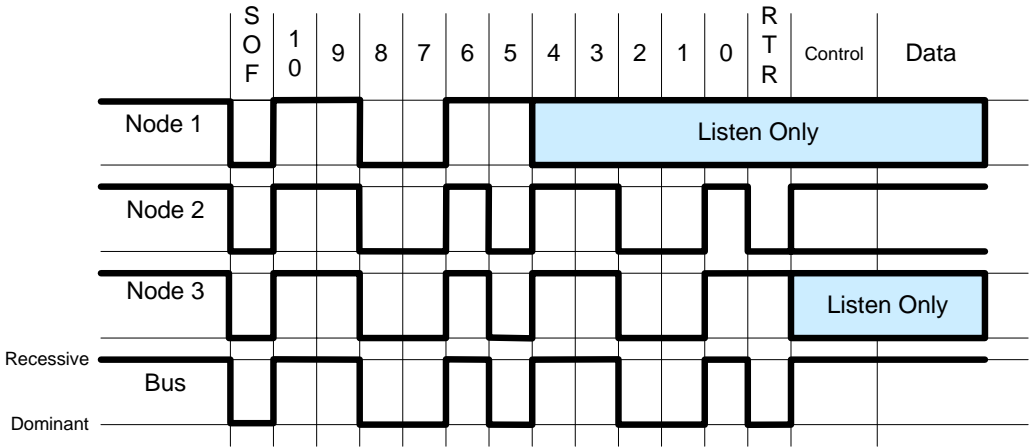


Figure 3 - Wired AND Access mechanism⁴

This access mechanism allows a system designer to assign a priority to network messages. Those with lower identifiers will always win access to the bus.

This message prioritisation is very useful in a system where some network messages have a much higher associated importance than other messages on the network. For example data for ABS braking information will carry a higher importance than engine temperatures, as the consequences of the data not getting through are much higher. This access mechanism is

³ Source – Atmel CAN Tutorial

⁴ Source: <http://www.can-cia.de/can/protocol/>

known as CSMA/CD & AMP, Carrier Sense Multiple Access / Collision Detection and Arbitration by Message Priority.

CAN Message Frame format

There are several different types of frame that may be present on a CAN bus:

- Data Frame – Carries data from the transmitter to the receiver
- Remote Frame – Transmitted by a station on the bus to request the transmission of a data frame with the same identifier.
- Error Frame – Transmitted by any station on the bus, on detection of an error
- Overload Frame – Used to increase the delay between Data and Remote Frames

The CAN protocol specifies two formats of frame that may be transmitted over the bus. These are known as Version A and Version B or Standard and Extended format.

SOF	ARBITRATION	CTRL	DATA	CRC	del	ACK	del	EOF	IFS
1	12-32	6	0-64	15	1	1	1	7	>=3

Figure 4 - CAN Data Frame Structure

Figure 4 shows the structure of a CAN Data Frame, this is made up of the following components:

- SOF – Start of Frame, a single dominant bit (logic 1), used to signal that start of the transmission of a frame.
- Arbitration and Control Fields – the contents of these vary depending on which protocol version is being used 2.0A or 2.0B, see Figure 5 and Figure 6

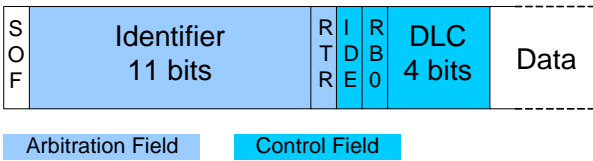


Figure 5 - CAN A Arbitration & Control Fields

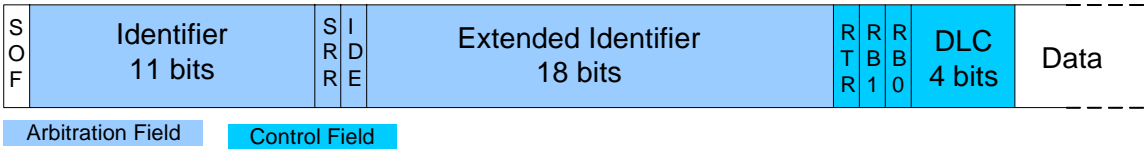


Figure 6 - CAN B Arbitration & Control Fields

- Identifier – 11 bits for protocol version A, 29 bits for protocol version B
- RTR – Remote transmission request, dominant in a data frame, recessive in a remote frame
- IDE – Version A/B Identifier, dominant for version A, recessive for version B

- DLC – Data length code, number of bytes in the Data Field
- RB0, RB1 – Reserved Bits
- SRR – Substitute Remote Request, only used in version B, always recessive
- Data Field – Contains the message data, consist of 0-8 bytes of data
- CRC – 15 bit cyclic redundancy check field, follow by a single recessive delimiter
- ACK – Set recessive when a message is being transmitted, set to dominant when in messages send in the ACK slot – in response to a correctly received message. This is also followed by a single recessive delimiter.
- EOF – End of Frame, seven recessive bits to mark the end of the frame.
- IFS – Inter Frame space, there must be a space of at least three bits between the transmission of each message on the bus.

Remote Frames follow the same pattern as the data frames, except the Remote Transmission Request is recessive and there is no Data field.

Error Frame Format

CAN error frames are transmitted by any unit on the bus that detects that the frame it has just received contains an error. Receiving units can detect the following types of error:

- *Bit stuffing errors* – where more than five consecutive bits have the same value
- *CRC Errors* – When the receiver calculates the CRC value for the received frame and it doesn't match the CRC value transmitted with the frame. This indicates one or more bits have been altered during transmission of the frame.
- *Violation of frame format* – For example a delimiter is not found where one is expected.
- *ACK Errors* – Where a unit transmits a frame, but receives no acknowledgement of it's correct reception.

Error frames give no indication as to the type of errors that have occurred, only that one has occurred and that the transmitted unit should attempt to retransmit that frame.

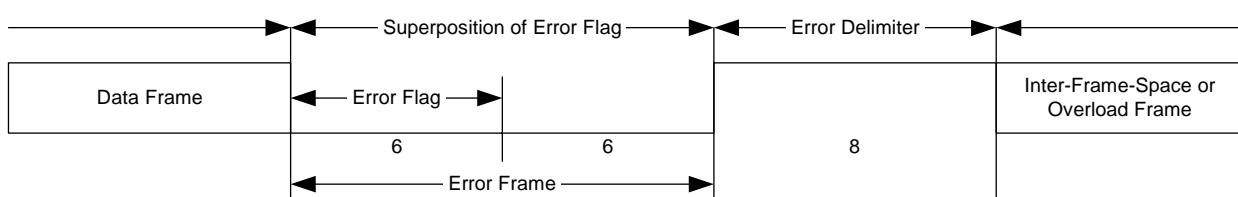


Figure 7 - CAN Error Frame

A mechanism exists within the CAN controller to count the number of transmit errors, if this number exceeds 256 consecutive errors then the transmitter should disable it's self. This prevents a single faulty unit on the bus, taking over the bus, with a stream of continuous re-transmission attempts

Overload Frame Format

Overload frames are sent by units on the CAN bus to introduce a delay, and slow down the rate that data is being transmitted by the other units.

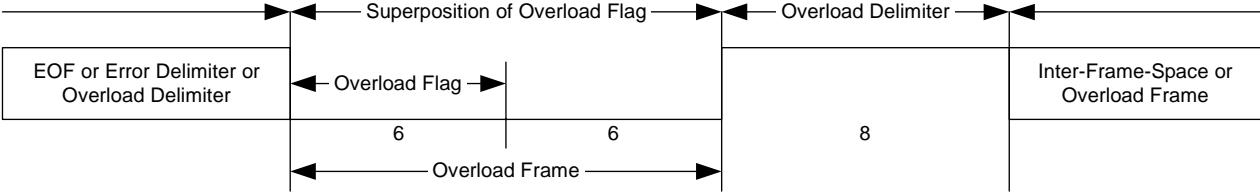
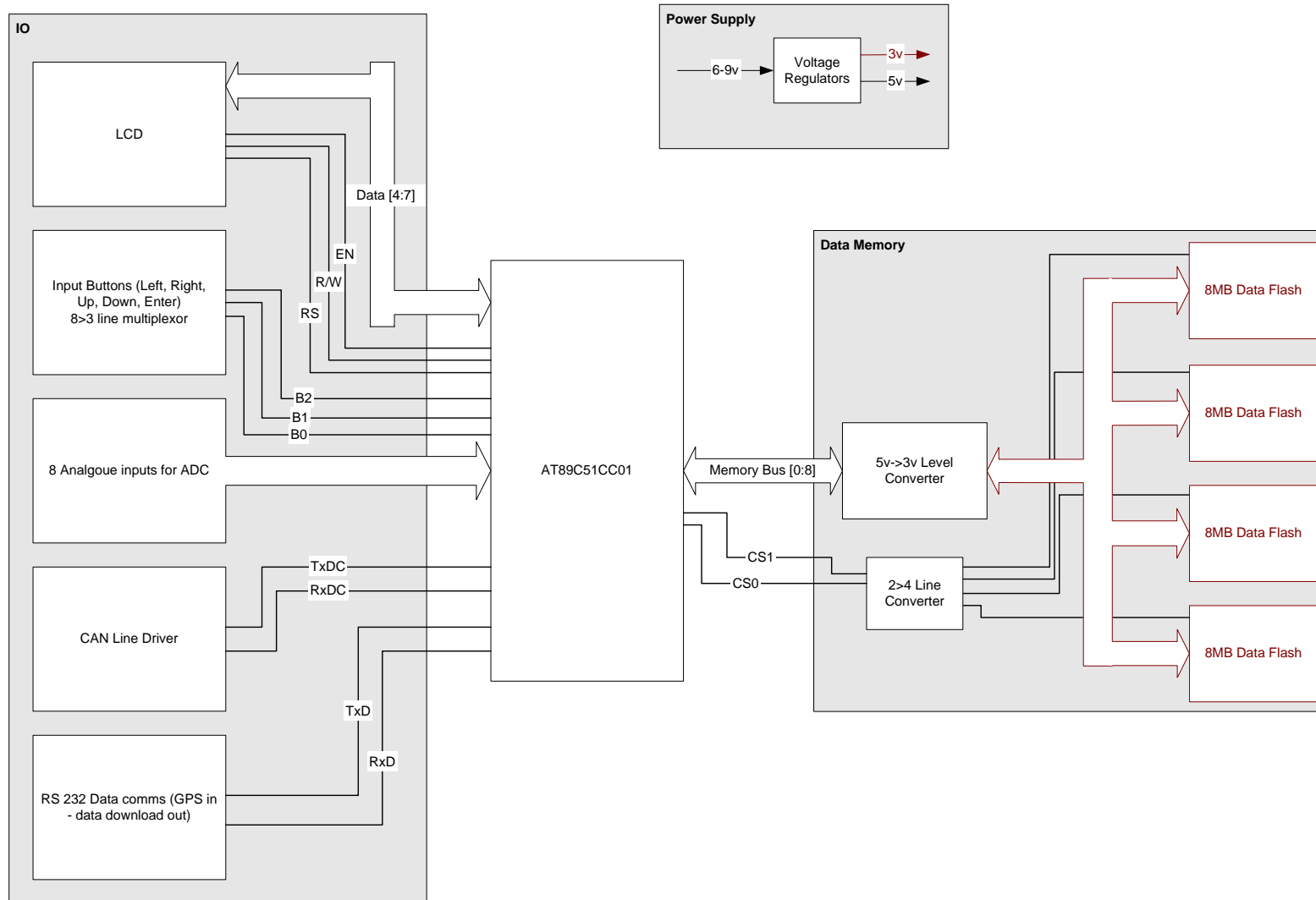


Figure 8 - CAN Overload Frame

HARDWARE DESIGN - BLOCK DIAGRAM



HARDWARE DESIGN DESCRIPTION

Schematic Design

The schematics were designed using the software package Orcad 9.2. A separate schematic diagram has been created for each board. The schematics are included in the appendix at the end of this document.

Construction

The hardware is constructed on 3 separate boards, the main motherboard is wire wrap – this allowed for easy modification during the testing and construction stage of the project, and has proved very robust.

Due to the very small pin pitch (0.2mm) on the Atmel DataFlash devices (TSOP 40 package) they were mounted on a PCB, along with the other surface mounted components required to interface to the Flash RAM.

The accelerometers were only available in a lead less 8 pin micro-package, and hence these have also been mounted on a two separate PCB daughter boards. In order to measure acceleration in all 3 axis's one of these daughter boards is mounted vertically.

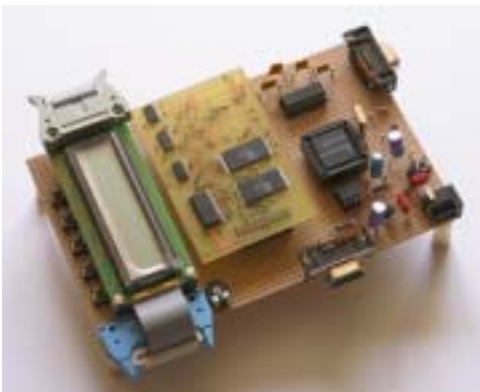


Figure 9 - The assembled hardware

Power Supply

The motherboard requires a 6v supply, connected via the commonly used 2.1mm power jack. This is then fed through two voltage regulator ICs one to provide 5v and the second to provide 3v. The addition of two 22uF capacitors provides smoothing of the power output.

The 6v supply may be sourced either from a standard mains power supply – similar to those supplied with a variety of low voltage electrical products, or a similar unit which plugs into a cars 12v cigarette lighter socket thereby providing the required 6v supply..

Microcontroller

The Atmel AT89C51CC01 Microcontroller has been selected for this project because it has a large number of integrated peripherals ideally suited to this data logging device, these include:

- 8051 with 32K Flash Program RAM and 1.2K of Application RAM
- UART to enable communication with a PC.
- CAN Controller
- Analogue to Digital Converter (8 multiplexed channels)
- 2K EEPROM Data Memory – allows storage of configuration data, error counters and indexing for main DataFlash memory.

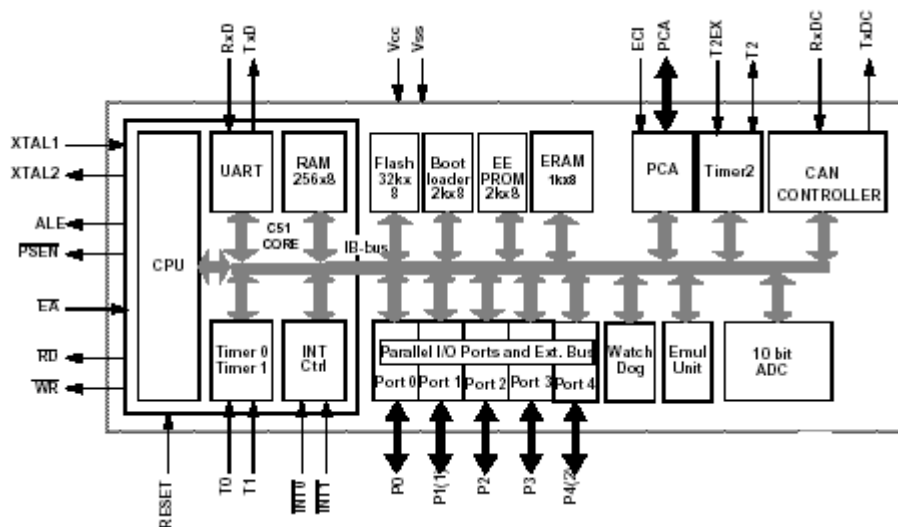


Figure 10 - Block Diagram of the Microcontroller⁵

An external 12MHz crystal oscillator generates the Microcontroller clock signal, 12MHz was selected as it divides down easily to a suitable frequency for the CAN baud rate generator. The Microcontroller has a fuse bit which allows the developer to operate it in X2 mode, in this mode instructions only require six machine cycles instead of twelve, hence the software executes twice as fast. The Data Logger utilises this X2 mode.

The Microcontroller is capable of In System Programming (ISP) via either the CAN bus or the UART (RS-232 Port) depending on the boot loader present. The Data Logger is loaded with a boot loader to permit programming via the UART. A jumper on the motherboard allows the user to select if the ISP boot loader or the main program executes at reset/power up.

RS-232

Communications with the PC are provided via RS-232 this interface was chosen as it is simple to implement and is fitted as standard to laptop and desktop PCs.

⁵ Source: AT89C51CC01 Data Sheet

The MAX232 line driver provides the translation between the Microcontroller 0-5v output signals and the +/-12v signals required to drive RS232.

CAN

Most of the CAN controlling electronics provided are integrated into the Microcontroller, only one external line driver component is required. The connector used for the CAN is a 9 pin DSUB connector, with the pin out as suggested by the CAN Industry Association (this is the same pin out as the Atmel Evaluation Board used during for the development of this project)

Note: the CAN standards do not define a connector type or pin out.

The CAN controller built into the Microcontroller provides all of the low level CAN interfacing functions, such as error checking, and data buffering. This frees the processor to concentrate on the data processing tasks.

A 120 Ohm terminating resistor is provided if the Data Logger is the last device connected to the CAN bus, this can be selected via a jumper on the motherboard.

Flash RAM

The Data Logger is fitted with two 64Mbit Atmel DataFlash devices for data storage. These Flash RAM devices are serial flash storage devices, that differ from standard RAM as they do not require any address pins, therefore keeping their pin count low (important as the Microcontroller only has a limited number of pin available for IO). They can be driven by a parallel or serial interface, the parallel interface was selected as data can be written at a much higher rate to it.

The biggest interfacing problem with the Flash RAM has been that it uses 3v for the power supply and signal levels, this requires the addition of a 5-3v level converter between the Microcontroller and Flash devices, this also required another control line to indicate the read/write direction.

The chip select is provided by a 2-4 line decoder, this is a 3v device with 5v tolerant input pins to provide the interface between the Flash and Microcontroller.

LCD

The LCD is a self contained module that requires no external interfacing components between it and the Microcontroller. In order to reduce the number of IO pins required it is configured in four bit mode (this means each character has to be sent to the LCD as two separate four bit nibbles)

Input Buttons

Five input buttons are provided to allow menu selection and configuration via the LCD. They are interfaced to the Microcontroller via an 8-bit priority encoder that encodes the five buttons onto three lines, again reducing the number of IO pins required on the Microcontroller.

Accelerometers

The accelerometer initially chosen for the project was the Analog Devices ADXL150. This is a single axis $\pm 50g$ device with an analogue output signal of 0-5v. This presented two problems, firstly the range of the accelerometers, $\pm 50g$ is far wider than a car would experience under normal conditions. A much more realistic range would be $\pm 2g$ but in order to use these accelerometers an 8 bit ADC high gain amplifiers would be required, this makes the circuit more complex and is likely to introduce more noise. Secondly, the amplifiers would require their output to be limited to 3v as the ADC interface is 3v only.

However early in 2003 Analog Devices launched a brand new accelerometer, the ADXL311, this accelerometer has an ideal range of $\pm 2g$. The accelerometers are powered from a 3v supply and produce an output signal 0-3v, they are pre-configured with a 0g offset of 1.5v. This means an output voltage of 0v is indicative of acceleration measurement of $-2g$. Additionally these accelerometers are 2 axis devices, so only two devices are required to provide data from the X,Y and Z planes.

$\pm 2g$ is greater than a car would typically experience during normal driving conditions. If the user were specifically looking to collect and analyse data from a crash situation where higher accelerations are experienced then accelerometers with a greater range would be required.

The accelerometers use Analog Devices iMEMS (Integrated Micro-Electro Mechanical Systems) technology. These devices are based on the same technology are already fitted to many cars to provide de-acceleration information for air-bag sensors.

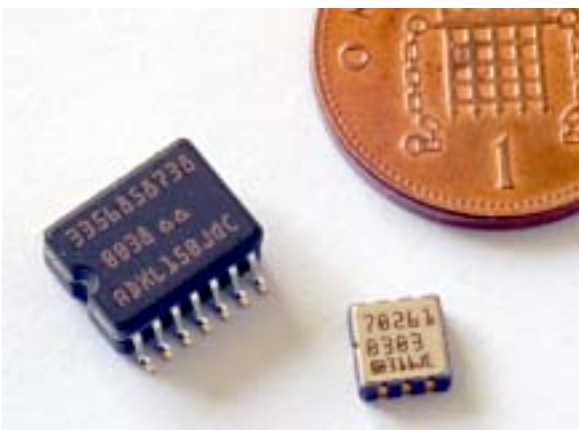


Figure 11 – Analog Devices ADXL150 (left) and ADXL311 (right) accelerometers

Often when modelling vehicle acceleration and behaviour, it is helpful to consider the vehicle as a point source. In order to do this it is important that the accelerometers are placed as

close as possible to the centre of gravity of the vehicle. As the purpose of this project was simply to collect data, rather than perform a detailed analysis of that data, the positioning of the accelerometers was not considered important during the testing.

iMEMS technology

MEMS stands for Micro-Electro-Mechanical Systems, it is the integration of mechanical elements with electronics in the same silicon package. The mechanical section is manufactured using the same silicon etching and deposition processes that are used to construct the electronic section of the device.

The iMEMS accelerometers work by having a beam suspended by two springs that is free to move back and to. Attached to this beam are a series of 'fingers', these moving fingers are interspaced by fingers which are fixed to the main body of the device.

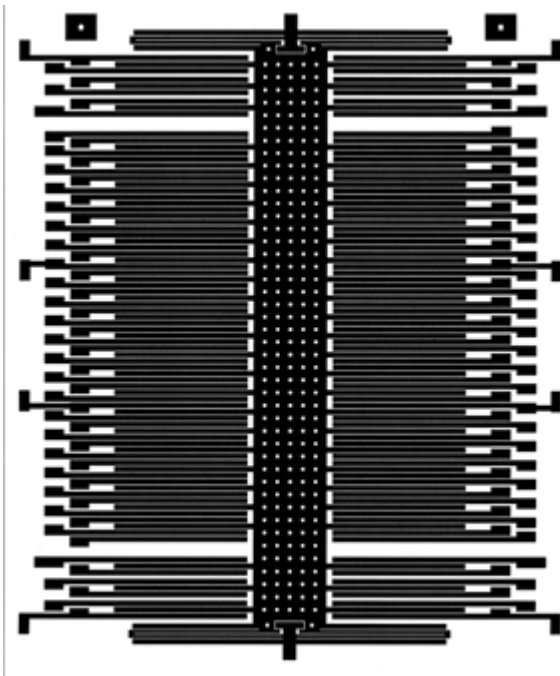


Figure 12 - Silhouette Plot of iMEMS accelerometer⁶

As the device is subjected to accelerating forces, the beam moves back-and-to, this alters the differential capacitance between these fingers on either side of the beam, this difference in capacitance is measured to produce a value for acceleration.

⁶ Source: Analog Devices Website

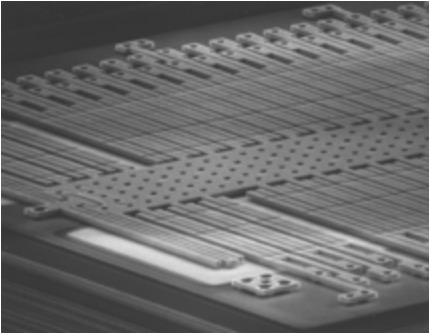


Figure 13 - Scanning Electron Microscope view of iMEMS accelerometer⁶

MEMS is an area of technology where there is currently a lot of research. Researchers have been able to construct a wide variety of mechanical systems, and MEMS based devices are likely to find their way into many applications such as:

- Machining of inductors and tuneable capacitors
- Micro machined Scanning Tunnelling Microscopes
- Detection of hazardous chemical and biological agents
- Solid state gyro meters

There are of course likely to be many uses, most of which we have not even thought about yet. Some researchers are predicting that MEMS technology may be as important as the invention of the integrated circuit. The low inertia of MEMS components means it is possible to construct mechanisms that can be accelerated to high speeds without a large amount of energy.

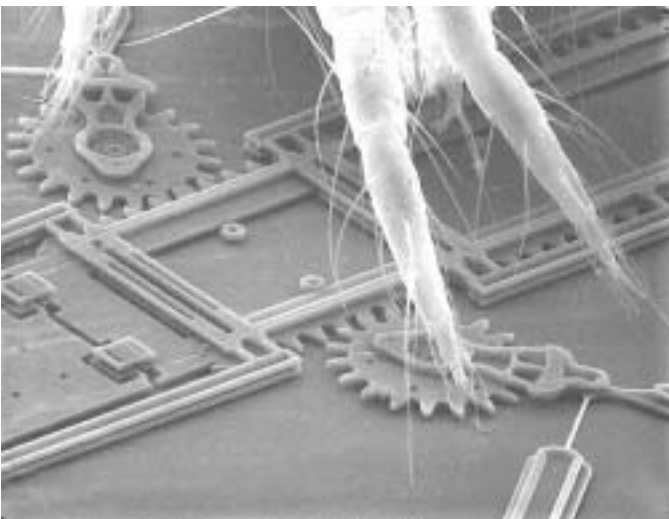


Figure 14 - MEMS gears with the legs of a spider mite⁷

MEMS technology is used in yaw rate sensors fitted to cars with ESP, it works in a similar way to the accelerometers, using a system of masses suspended by springs to measure the rotational speed of the vehicle.

⁷ Courtesy Sandia National Laboratories, SUMMIT™ Technologies, www.mems.sandia.gov

SOFTWARE

There are four main components to the software system that supports the Data Logger:

- 8051 Real-time data logging software, this is the main software and is responsible for reading data from the various sources and storing it in Flash RAM.
- 8051 Real-time CAN test pattern generator. This was only used for testing of the CAN data logging functions.
- PC Download – A Visual Basic application used to retrieve data from the Data Logger.
- PC Analysis – A series of Matlab routines to process and analyse the data once it has been downloaded to the PC.

Software Development and Configuration Control

The software has been developed using a modularised build-and-fix approach. This has been used as this was the authors first major embedded real time software project.

The software designed is based on modularised approach, with a central kernel routine calling a number of separate, self-contained modules.

All of the embedded software has been kept in a configuration control system to aid development. The system known as CVS (Concurrent Version System) allows the files to be 'checked in' after modifications, adding a description of the changes. This has two major advantages:

- If a change stops the software building, or running then it is trivial to ask the system to return a list of the changes between the last 'check in' and the previous one.
- Using the Build-and-Fix development method it is rather easy to get into a state where the software either no longer builds or functions but the developer cannot remember what they have changed! CVS allows developers to trivially 'roll back' the source code to the last known working point and start again.

Additionally an automated nightly backup of this source code repository has been carried out, a script runs which compresses the repository into a single file and uses FTP (file transfer protocol) to copy the source code to a remote server.

Embedded Real-time Software

The embedded code for the 8051 processor is by far the most complex piece of software in the project. It is based around a single master loop, which calls each module in turn.

Each of the modules poll buffers and global variables in turn to determine whether they are required to do any processing. In order to keep up with the high rate data logging requirements of the CAN and ADC these logging function are called much more frequently than some of the other less demanding functions.

Modules must return to the main loop as quickly as possible – large time consuming tasks are split into smaller tasks, and each small task is completed every time the module is executed. Figure 15 shows the order the modules are executed.

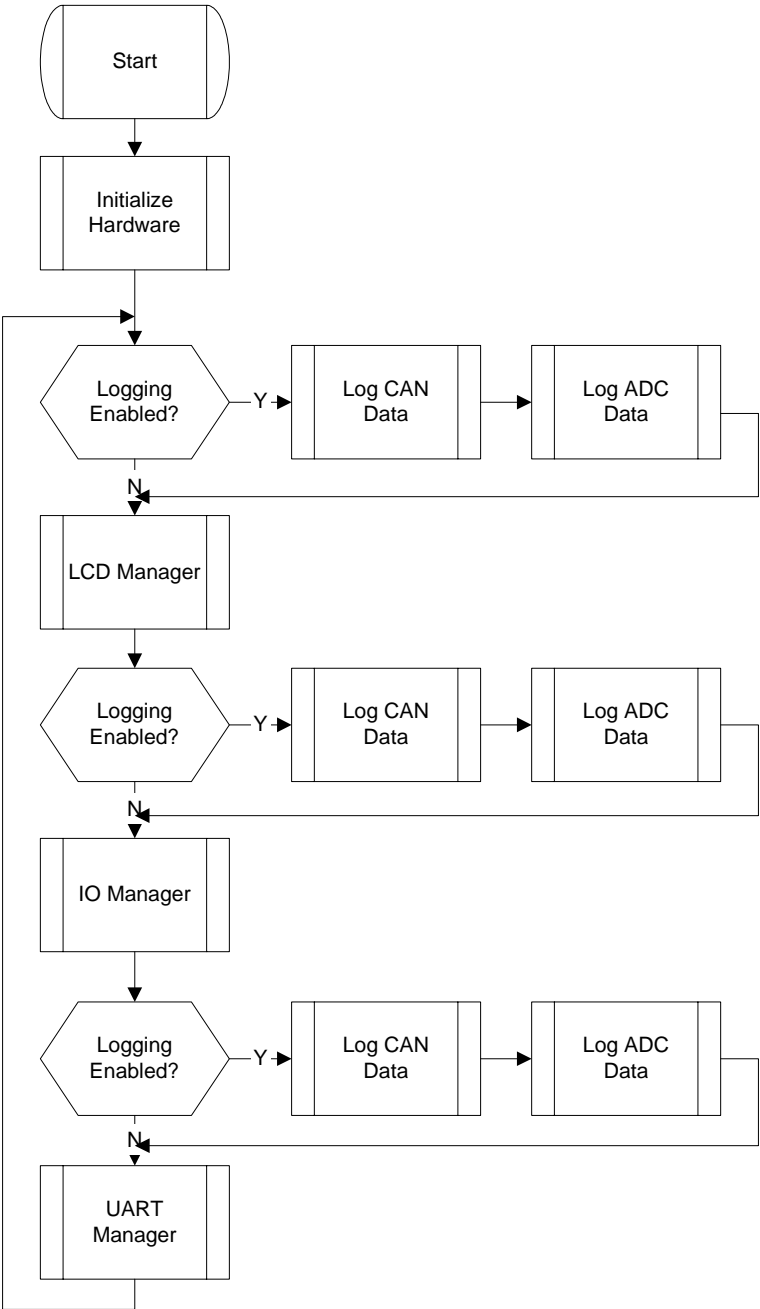


Figure 15 - Main software loop

Log CAN Data

This function polls two buffers (one for CAN version 2A the other for CAN version 2B) any data in the buffers is written to DataFlash, along with it's message ID and a timestamp. The timestamp is provided by the CAN controller, and is used to re-construct a time axis. There are 15 message reception 'slots', each of these can be programmed to transmit or receive data,

they can also be combined to form reception buffers. For this data logging application the message slots are configured as two buffers as illustrated in Figure 16.

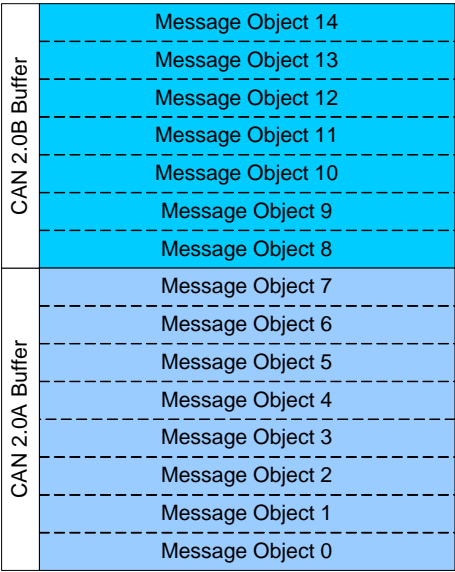


Figure 16 - CAN Buffer configuration

If the user knows that they only need to log CAN bus messages with a specific message ID then it is possible to program the reception buffers with a filter, so that only messages with that ID are recorded, thereby saving memory.

Log ADC Data

This function checks to see if the ADC readings have been updated and if they have it writes their values to the DataFlash.

The actual reading of the ADC data is carried out by an interrupt service routine. The interrupt is generated every 10ms by one of the built-in 16 bit counter/timers. The readings are then stored in RAM until the Log ADC Data module is executed.

LCD Manager

The LCD Manager checks whether any of the other functions have modified the LCD strings. If they have been modified then it clears the display and outputs the new data.

IO Manager

This is principally responsible for reading the status of the input buttons. The value of the input buttons is polled every time this function is called. It performs software based switch de-bouncing to ensure that bouncing switch contacts do not appear as multiple button presses. The IO manager is also responsible for managing the menu system, updating the data that is to be displayed on the LCD and performing enabling/disabling for the various functions.

UART Manager

The UART manager checks the input buffer for any commands arriving from a connected PC, and sends data in reply. If it is required to send a large amount of data in response to a request from the PC, it sends a small section of the data each time it is called, to avoid holding up the other processes.

DataFlash Operation

As the DataFlash is serial RAM and has no address bus, specialised drivers have been developed to provide a level of hardware abstraction to the main software. The main software only needs to call two functions flash_read() and flash_write(). These two functions handle the commands to write to buffers, tracking of pages and address etc.

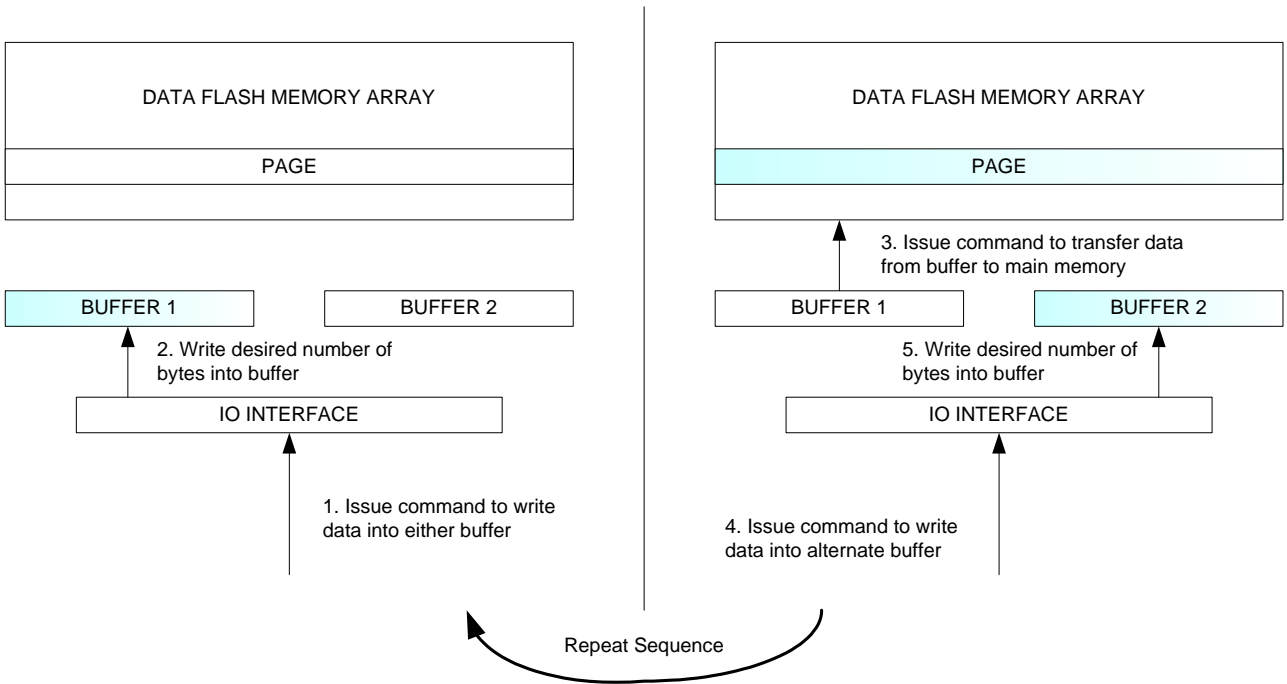


Figure 17 - Continuous Write Algorithm for DataFlash⁸

The Figure 17 illustrates demonstrates how, by using the two buffers built into the DataFlash devices it is easy to achieve a sequence of continual writes to the DataFlash.

Although the DataFlash is a serial memory device, through careful use of the buffers, it is capable of pseudo-random access, but it is very slow. However, it is ideally suited to the Data Logging application as data is written and read back in large blocks.

⁸ Source: Atmel DataFlash Datasheet

Interrupts

The use of interrupts is avoided wherever possible. Only one interrupt is used for this project, this is triggered by the overflow of a timer, it is used to sample data from the analogue interface at regular time intervals.

Not using interrupts is beneficial as it is much easier to predict how time critical sections of the software will perform. The CAN and UART IO ports both have buffers to allow data to be stored whilst the processor is executing other functions.

Real-time software – CAN Test utility

In addition to the main software used in the Data Logger a separate utility has been developed to aid the testing of the CAN logging feature of the main software. This test utility is very simple, it configures the CAN baud rate to 500kbps and then sits in a continuous loop sending out CAN messages, the ID counts up to 7FF before rolling over and the data counts to FFFFFFFF before looping, the data count is transmitted twice in each CAN message.

This program was used to test the following features on the Data Logger:

- The receiver baud rate was set correctly
- CAN version 2.0A and 2.0B messages could be successfully received and stored in memory.
- CAN Messages could be received in the correct order
- CAN Messages could be received at full data rate without message lost due to buffer overflows.



Figure 18 - Atmel AT89C51CC01 Evaluation Kit

The test program was loaded into an evaluation board provided by Atmel, and a small CAN network was created between the evaluation board and the Data Logger. This allowed CAN communications to be fully tested before real-world trials were carried out on the test car.

PC Software – Download Application

In order to allow the data to be transferred from the Data Logger's on board memory card to a PC for further analysis and processing a 'Download' Application has been developed.

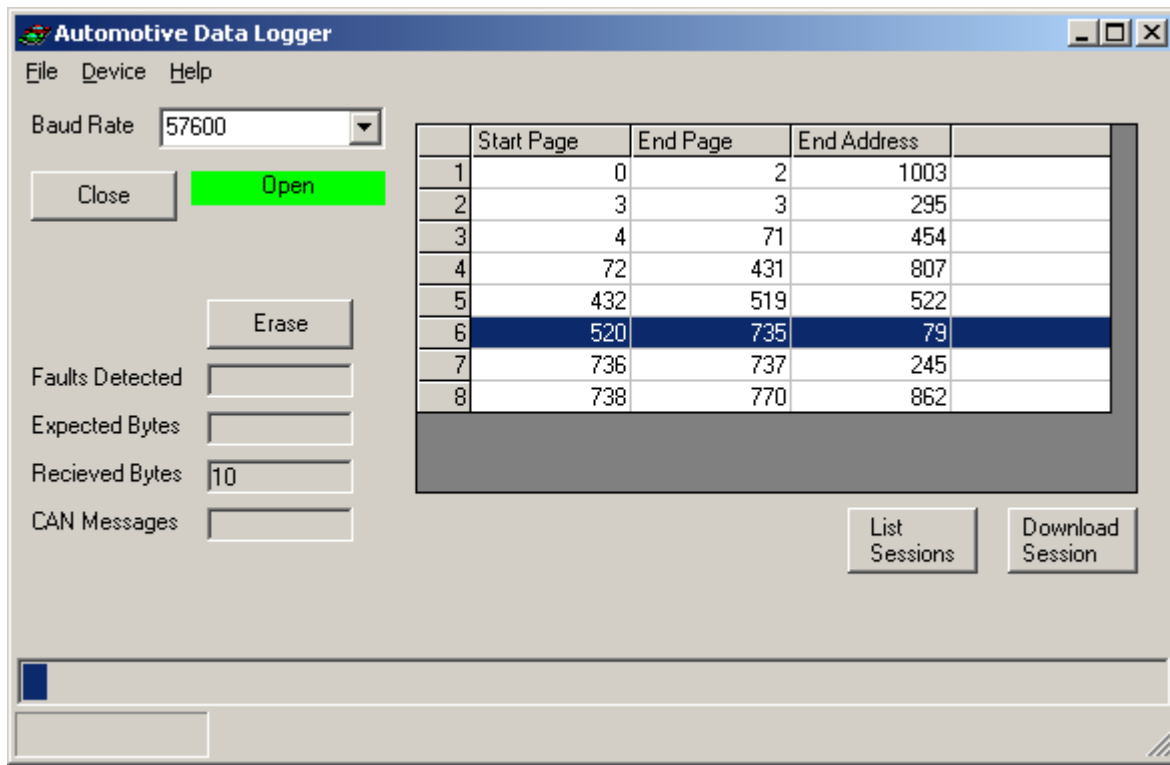


Figure 19 - Download Application User Interface

This application is written in Visual Basic and has a user interface which allows the user to select the data session they wish to download – the data from this session is then downloaded from the memory card, the different categories of data are separated out and written to disk in hexadecimal format that can be read in by other programs. The 4 files that are written for each session are:

- can_a_data.txt – Contains all of the CAN 2.0A protocol data received.
- can_b_data.txt – Contains all of the CAN 2.0B data received.
- adc_data.txt – Contains all of the accelerometer data received from the ADC in CSV format
- raw_data.txt – Contains a copy of the raw data, in the same format as stored on the memory card.

A set of files is generated for each session and stored in a separate folder, labelled with the session number.

During software development, the PC download application was modified to include some code to check that the test patterns being generated on the evaluation board were being received correctly.

PC Software – Matlab Routines

ADC data processing

The Matlab routine for processing the analogue data from the accelerometers reads the data from the input file into a matrix then creates a time axis matrix using the fact that readings are taken from the accelerometers every 10ms. In addition to this, a simple moving average filter is applied to attenuate some of the noise present in the analogue signals.

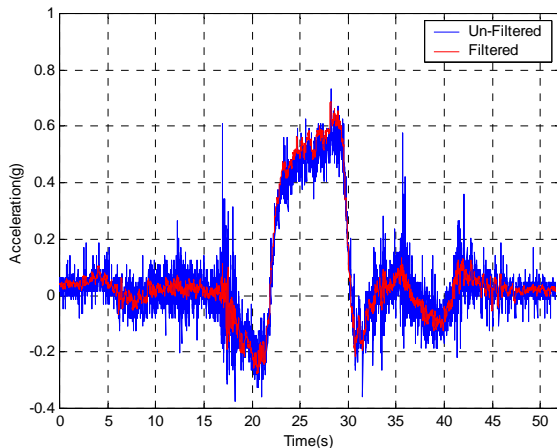


Figure 20 - Accelerometer Values before and after filtering

The most likely source of this noise is from the other electronics on the Data Logger board. It has been noticed during development that the noise levels on the analogue signals are greatly increased when the unit is recording data from the CAN interface as well as analogue data.

CAN Data processing

The main tasks the Matlab routine performs are:

- Reading all of the data from a file into a Matlab Matrix to permit analysis and manipulation
- Separating messages of a specific message ID from the rest of the data
- Extracting specific bits from that message into a matrix so it can be plotted
- Turning the time stamp values from the 16 bit timer into values suitable for use as a time axis knowing that the counter increments at a frequency of 31.25kHz and that the timer rolls back to zero at 65535 (2^{16})

Data storage format

Logged data is stored in the two 8MB DataFlash devices on the memory card, this data has an index which is stored in the processor's on board EEPROM, so that it can be manipulated on a

per-byte basis. This index enables the Microcontroller to find specific sessions within the DataFlash.

EEPROM Index Format

Each index entry in the EEPROM is stored in a 16 byte data structure, even though all of the 16 bytes are not used, using 16 bytes makes it simple to ensure that the entries do not cross page boundaries in the EEPROM, thus making it much simpler to write the data to EEPROM.

The following data structure is used:

```
struct session_config {  
    unsigned short start_page;           /* 2 bytes*/  
    unsigned short end_page;           /* 4 */  
    unsigned short end_buff;           /* 6 */  
    unsigned short can_msg_count;      /* 8 */  
    unsigned short adc_msg_count;      /* 10 */  
  
    unsigned char adc_enable;          /* 11 */  
  
    unsigned char pad[5];  
};
```

DataFlash Format

The DataFlash stores 3 different types of data:

- ADC readings from the accelerometers
- CAN Version A Data
- CAN Version B Data

CAN Data is stored in the following format:

- 2 byte Data Type Identifier (0103 for version A, 0104 for version B)
- 2 byte timestamp
- 2 – 4 byte CAN Identifier (depending on CAN A or CAN B)
- 8 bytes of CAN data

ADC data is stored in the following format:

- 2 byte Data Type Identifier (0105)
- 3 bytes of data (one byte for each analogue input)

All the fields apart from the Data Type Identifier have any values that are 01 byte stuffed, to enable the software to differentiate between actual data and identifiers. The Data Type Identifier 0101 should never be used.

Serial Communications Protocol

The serial communications protocol is very simple and shares a lot in common with the format of the DataFlash, this allows data to be read straight from the DataFlash and transmitted over the serial link without requiring any processing. The following Data Type Identifiers are valid for serial communications:

Data Type Identifier	Meaning	Sent by PC/Data Logger
0102	Number of bytes to be transmitted	Data Logger
0103	Start of CAN A Message	Data Logger
0104	Start of CAN B Message	Data Logger
0105	Start of ADC values	Data Logger
0106	Number of Sessions in EEPROM Index	Data Logger
0107	Start of Session Information	Data Logger
0102	Send Session List	PC
0104	Erase all session Information	PC
0105	Send Session Data	PC

Table 2 - Data Type Identifiers for Serial Protocol

Software Development Tools Used

The software development tools used for this project are specialised and are described below:

Compiler

There are many different compilers available for the processors based on the Intel 8051, Loughborough University provides and recommends the use of the Kiel compiler. This provides a complete Windows based development environment. However for this Data Logger project the compiler chosen was an open source based one called SDCC, this avoided licensing costs and allowing the author to develop the software working from home. Whilst this is a command line based package, the editor used (EditPlus v2) allows easy integration with external compilers, effectively providing an excellent IDE (Integrated Development Environment)

Programmer

The Microcontroller is programmed whilst fitted to the board via the RS-232 interface. The software used to do this is called FLIP (FLexible In-system Programmer) and is provided as a freeware download from the Atmel website⁹.

Source Code Control

Source code versioning control was provided by a package known as CVS (Concurrent Versioning System) with web access to the source code repository via an application known as

⁹ http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2767

Chora. The web interface provides a very simple method of retrieving check-in log files and old version of the source code.

EXPERIMENTATION AND PROVING OF DESIGN

Testing Overview

Once the hardware construction and software implementation had been completed to a sufficiently robust level, the Data Logger was tested on a car. The car chosen was a Audi A4 Avant, this is a modern design with a CAN bus fitted, however the design of the car is now 2 years old and hence the CAN bus implementation is not quite as complex as on some of the newer models.



Figure 21 - Audi A4 Avant¹⁰

Access to the CAN bus was gained by attaching two probes to the multi-pin connector on the ABS ECU (see Figure 22). This ECU is located on the right hand side of the engine compartment. A wire was then fed under the bonnet, through the window and to the passenger seat area. The Data Logger was powered via a 12-6v converter, which plugged into the car's cigarette lighter socket.

¹⁰ Source: Audi

2 testing sessions were carried out, the first of these over 4th and 5th April in Northampton, the 2nd was arranged on the 18th April in Leamington Spa. A second session was required following resolution of a few minor issues discovered during the first testing session.



Figure 22 - Connecting the Data Logger to the CAN bus

Testing of the Analogue Accelerometer Sensors

The accelerometers were tested using two methods, the first relied on the fact that the earth's gravitational field provided a constant acceleration of 0.98m/s^2 downwards. By rotating the Data Logger about each axis it was possible to see this acceleration component increasing and decreasing.

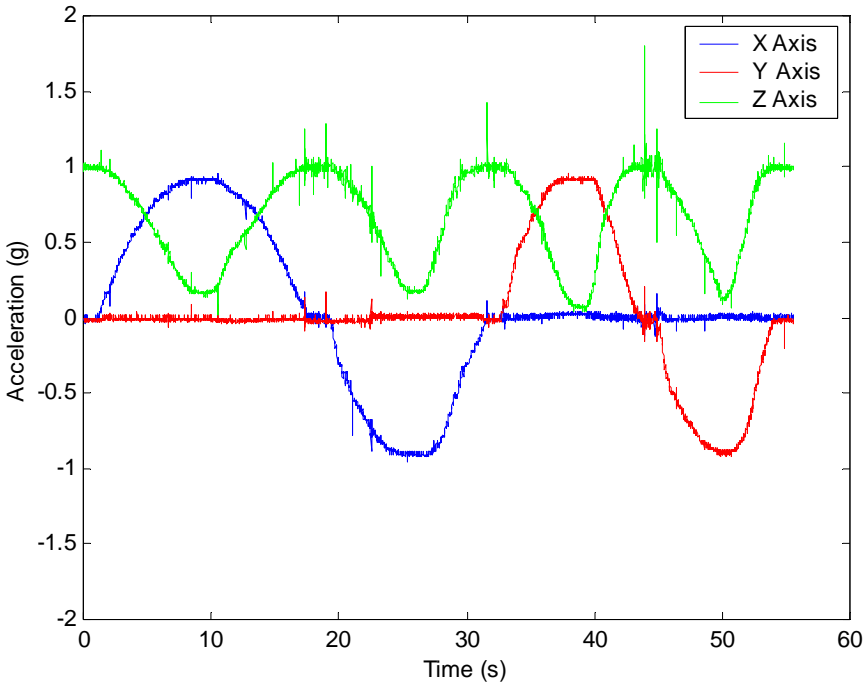


Figure 23 - Acceleration due to gravity

The second test method, was to mount the Data Logger in a vehicle and take it for a drive around a set course, then compare the results to those expected.

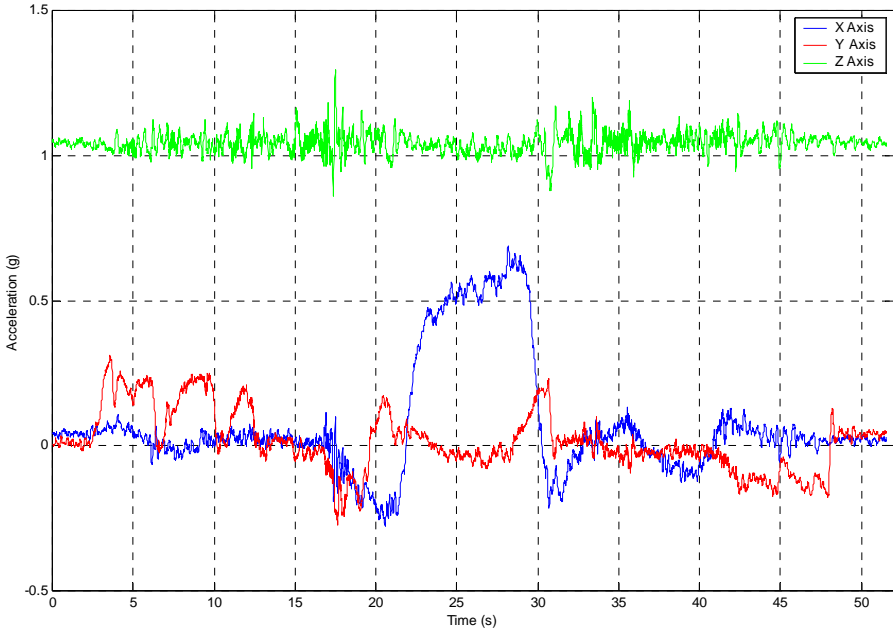


Figure 24 - Accelerometer Data from roundabout test run

The diagram above shows sample readings from one the test runs performed in the car. The data is plotted after the noise reduction filter has been applied.

The analogue to digital conversions are performed to 8 bit precision, the Microcontroller is also capable of doing 10 bit conversions, however in order to reduce noise the Microcontroller must be put into a powered down state whilst the conversion is performed. These high precision conversions also take longer than the standard 8 bit conversions and require more data storage space, given the tight time restrictions in place 8 bit precision conversions were deemed sufficient. This allows the measurement of acceleration down to a resolution of 0.015g

Collection of sample CAN data

Once connected to the vehicle, the Data Logger was used to record samples of data in several different situations. Unfortunately the exact formatting of the data in the CAN bus is considered by the vehicle manufacturers to be proprietary information, hence very little information is available to aid with the interpretation of the CAN data. When the downloaded data was analysed, 20 different CAN message IDs were found to be present. An attempt at reverse engineering this data to extract some useful and interesting variables has been made, with some success. However the meaning of much of the data remains unidentified.

Problems Encountered During Testing

Overall the initial testing session on 5th April went very well, with the DataLogger capturing CAN bus data on the first attempt. The following small problems were encountered:

- Without any proper tools attempts to analyse the data in Matlab and Excel were difficult. This led to the development of some routines in Matlab to import the data and extract the relevant bits.
- The DataLogger did not record any time information with the data. This made it very difficult to plot data with different CAN IDs on the same chart – as each message ID appears to be transmitted at a different time intervals. This has since been resolved by additionally logging a time stamp values supplied by a 16 bit timer present on the on-board CAN controller.
- Once the CAN bus was connected and data was present the DataLogger stopped recording information from the ADC. This turned out to be a software bug caused by having interrupts for the CAN controller unnecessarily enabled.

With all these above issues resolved, a second test run was completed successfully on 18th April. During this second run video was also recorded to aid with analysis.

CAN Message IDs present

The following message IDs were found:

0050	00C2	01A0	0280	0288	0320	0380
0420	0480	0488	04A0	04A8	0520	0580
0588	05A0	05D0	05D2	05D8	05E0	

From the data collected Bus utilisation has been calculated to be approximately 13.4% whilst this figure may appear to be low, a low level of bus utilisation is the only way to ensure that stations get access to the network with a minimum of delay. At this rate the Data Logger can store about 10 minutes of data in each 8Mb DataFlash device – 40 minutes if all four are fitted.

```
00C2 3AA5 5883000045000024
05D8 3AB4 0504003F15000001
05D0 3AC5 C014613018406406
0420 3AE4 028FA19CB0000000
0520 3B01 0204014151261200
05D2 3B12 00D6749379574155
00C2 3B22 5883000045100014
0280 3B69 4917BC0D17001E17
0288 3B79 0BB5180000585100
05E0 3B89 31932C200A6B0000
0380 3B98 006F000010000000
0480 3BA8 1D009FE735000000
```

Figure 25 - Sample data from can_a_data.txt

The figure above shows a small sample of the CAN 2A data, as captured during one of the test sessions. The 3 columns are:

1. CAN ID
2. 16 Bit CAN timestamp
3. 8 bytes of CAN Data

Interpretation of the Data

Attempts to reverse engineer that data obtained from the CAN bus, have so far yielded the following parameters:

- Steering Angle
- Engine RPM
- Brake Light Switch
- Brake Pressure
- Speeds for each wheel
- 3 Engine Parameters, one of these is most likely fuel consumption.

Other information that is likely to be present includes:

- Engine Temperature
- Fuel Quantity
- Engine Torque

Sample of information downloaded from Data Logger

Figure 26 shows a graph plotted from one of the data logging sessions. The following narrative describes how the car was driven for this test:

- The car started in a stationary position with the engine running
- 1st gear was selected and the car accelerated to 30 mph through 2nd and 3rd gear. The increase in engine rpm as the driver accelerates through each gear can be clearly seen.

- When the car reached 30 mph (top of 1st peak on blue speed line, about 12s) the driver backed off the accelerator and the car began to slow down for the roundabout.
- The car decelerates at a higher rate as the driver applies the brake (16s).

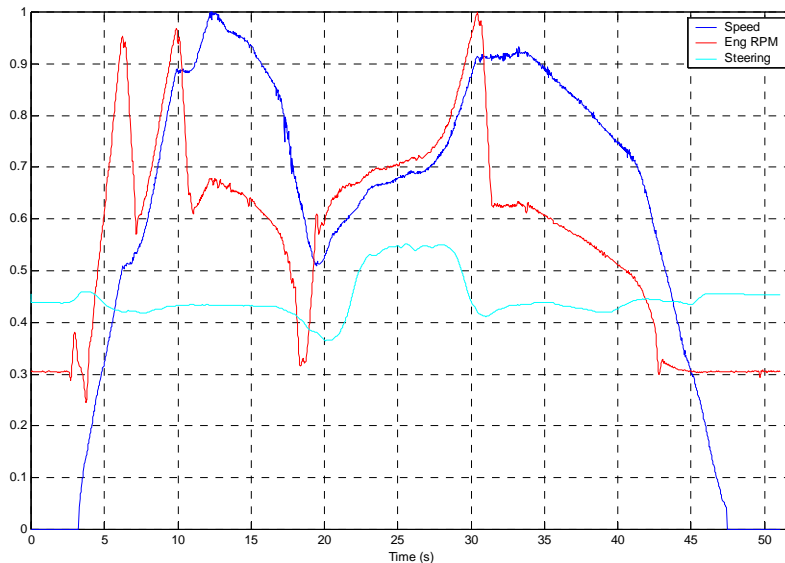


Figure 26 - Acceleration to 30 mph, then round a roundabout

- Once the driver was happy there was no other traffic on the roundabout he accelerated again and held a constant speed as possible whilst negotiating the roundabout.
- On reaching the desired exit of the roundabout, the driver then accelerated back to 30 mph.
- Once 30 mph was reached the driver held the speed steady for a few seconds then released the accelerator and allowing the engine to go into overrun as the car decelerates.

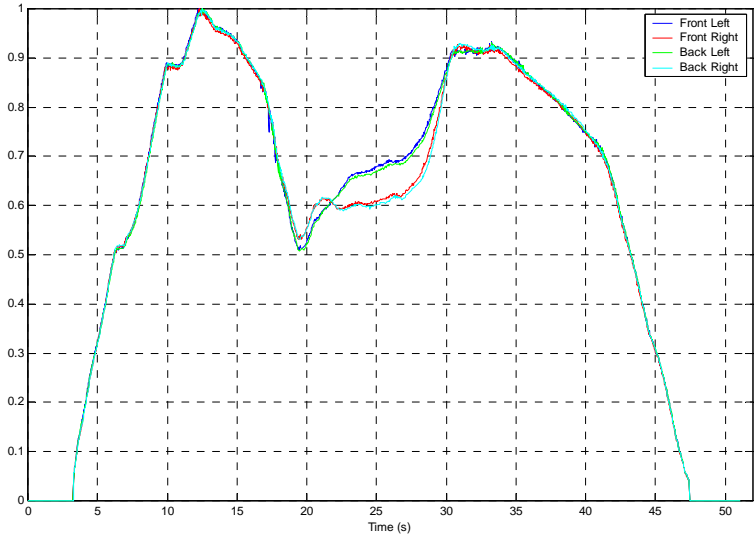


Figure 27 - Wheel Speeds

Figure 27 clearly shows that all four wheels rotate at the same speed whilst the vehicle is travelling in a straight line, but as soon as the car turns left (about 17s), the right hand wheels move faster, and as the car swings right and continues round the roundabout (22s) the left wheels are rotating at a faster rate. This effect is observed because the set of wheels on the outside of a bend travel round an arc with a bigger radius and therefore must travel further in the same time as the wheels on the smaller inner arc.

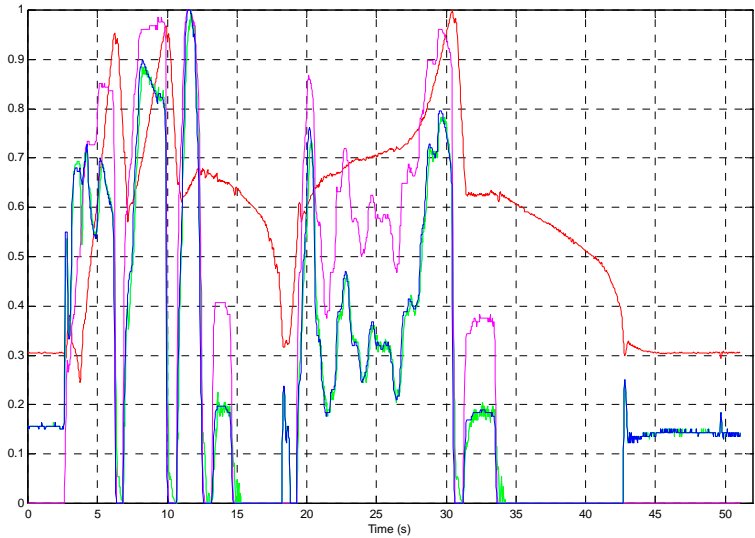


Figure 28 - Engine Parameters

The figure above is a plot of engine related parameters. Apart from engine RPM (shown in red) it has not been possible to ascertain exactly what they all are. One is likely to be fuel consumption, another may be recording the mass flow of air taken in by the engine.

PROJECT MANAGEMENT

Time

The initial time plan was created in the planning stage of the project, in October 2002, and was included in the interim report.

The times included on the initial plan were very rough estimates, based on limited previous experience.

At the initial stage apart from the obvious report deadlines that were fixed, the most important milestone was considered to be ordering the components before the Christmas holidays and exam period, this was to allow for delays in obtaining the parts, this period being 6 weeks where little time would be available to devote to the project.

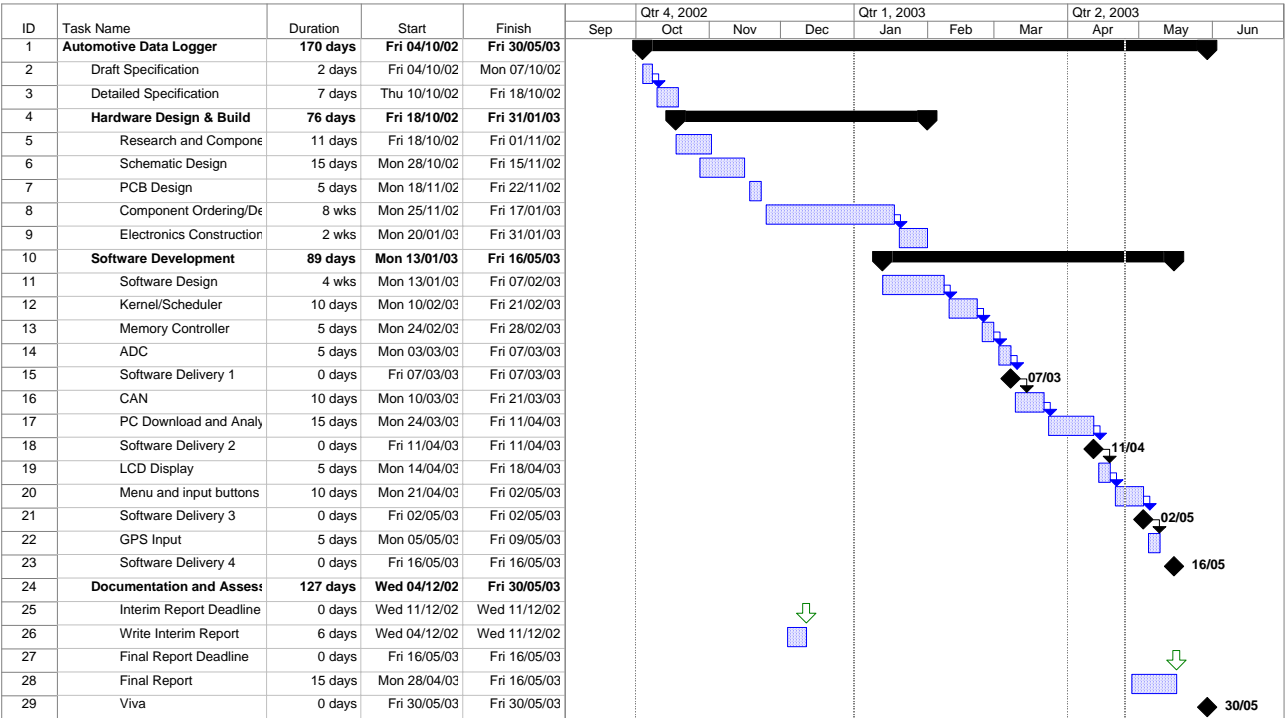


Figure 29 - The initial time plan

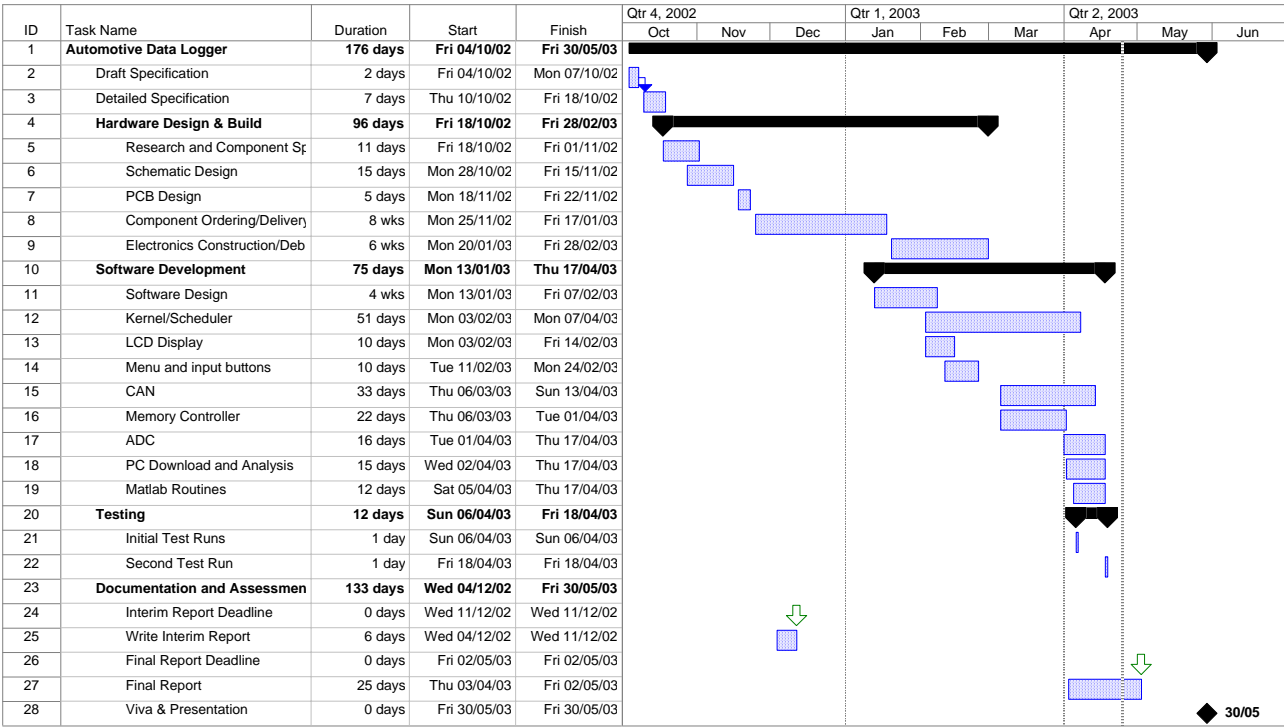


Figure 30 - Actual time planning followed

The Gantt chart above shows how the project actually developed, it can be seen that the original plan was largely adhered to. The largest deviation from the original plan was the software development stage. The original software development plan was to follow an evolutionary delivery development model, however once the software development had commenced it was considered easier to follow a build-and-fix approach, with modules generally being developed in parallel.

The software development times for Figure 30 have been taken from the log files of the software configuration management system, using the first and last check in dates for a module.

Finance

The budget allocated by the department for this project was £75. The following components were donated free:

- Atmel DataFlash components 2 @ £16.52 each
- Analog Devices Accelerometers 2 @ £3.40 each

Description	Qty	Price	Sub Total
Memory PCB	1	£9.94	£9.94
Accelerometer PCB	2	£3.24	£6.48
AT89C51CC01 Microcontroller	1	£11.00	£11.00
CAN Line Driver	1	£1.48	£1.48
RS-232 Line Driver	1	£2.47	£2.47
3v Regulator	1	£1.05	£1.05
9 Way PCB Mount Socket	2	£0.33	£0.66
Tactile PCB Mount Switch	6	£0.25	£1.50
Quad 2-Input OR Gate	1	£0.37	£0.37
Priority Encoder	1	£0.93	£0.93
44 Way Wire Wrap PLC Socket	1	£9.63	£9.63
2x16 Char LCD Module	1	£10.83	£10.83
PCB 2x10 Socket	1	£1.06	£1.06
3-5v Level Converter	1	£0.89	£0.89
3-to-8 Line Decoder	1	£0.54	£0.54
8 way buffer	1	£0.32	£0.32
Wire Wrap Board	1	£1.46	£1.46
Wire Wrap Wire	10	£0.21	£2.10
16 Pin wire wrap DIL Socket	1	£0.78	£0.78
8 Pin wire wrap Dil Socket	1	£0.39	£0.39
2.5 mm power socket	1	£1.54	£1.54
5v LM78 voltage regulator	1	£0.22	£0.22
10K Trim Pot	1	£0.18	£0.18
12MHz Crystal	1	£0.60	£0.60
22pF Capacitor	2	£0.01	£0.02
22uF Capacitor	2	£0.01	£0.02
100nF Capacitor	2	£0.01	£0.02
1uF Capacitor	4	£0.01	£0.04
10K Resistor	10	£0.01	£0.10
PCB Supports	4	£0.15	£0.60
M4 Bolts	4	£0.12	£0.48
M2 Nuts & Bolts	3	£0.60	£1.80
		TOTAL	£69.50

These prices are all based on the cost of single items. If the Data Logger were being manufactured in production quantities on a single PCB, using only surface mount components then it is estimated the component cost would be around £50.

FURTHER WORK

Known Problems

The following problems are known to exist with the Data Logger and would require further development work to eradicate them:

- LCD Updates sporadic. – Occasionally the LCD fails to update or displays incorrect information, this could possibly be a timing issue
- The interface between the DataFlash busy signal and the Microcontroller is currently an ACT32M Quad 2 Input OR gate, as the busy signal is at a logic one when the DataFlash device is not busy this device should be replaced with the equivalent AND gates. However the DataFlash components allow the busy status to be retrieved via a status register, so a software modification has been implemented to counter this problem.
- Corruption of data during read from Data Flash . It should be possible to issue a single read command and then read all of the data from the a single Data Flash device, however on some occasions this process appears to fail and 0's are read. A software work-around for this was developed and incorporated to allow the read command to be re-issued every 50 bytes, thereby reducing data loss to a minimum, but slowing the read process down.
- CAN bus buffer overflows when reading data from a 500Kbit stream at 100% bus utilisation. To solve this apparent problem some research would have to be carried out into optimising the software, and possibly increasing the processor clock speed to 20MHz(it is currently 12MHz, but up to 20MHz is supported) This problem is only present when bus utilisation is at or near 100%

Suggested Extensions

The following features are suggested as possible future extensions to the project.

- Extending the software to allow data to be written in both Data Flash devices (rather than one or the other)
- Automatic detection of presence of DataFlash devices and their capacity.
- Expansion of the menu system
- GPS interface – To allow the addition of positional information to be stored alongside the performance data collected.
- Implementation of a higher-level protocol – This will be necessary if the data logger is to be able to retrieve information from the next generation of cars currently in development.

BIBLIOGRAPHY

- Ayala, Kenneth J: The 8051 microcontroller: architecture, programming, and applications, (West, 1991)
- Jurgen, Ronald K: Automotive Electronics Handbook. 2nd Edition, (McGraw Hill, 1999)
- Ribbens, William B: Understanding Automotive Electronics. 6th Edition, (Newnes, 2003)
- Tanenbaum, Andrew S: Computer Networks, 3rd Edition (Prentice Hall, 1996)

MEMS

<http://www.memsnet.org/mems/applications.html>

Bosch ESP

<http://www.kraftfahrzeugtechnik-heute.de/k/en/start/index.jsp>

Atmel Wireless & Microcontroller

<http://www.atmel.com>

Analog Devices Inc.

<http://www.analog.com>

SDCC Compiler

<http://sdcc.sourceforge.net/>

8051 Development

<http://www.intel.com/design/MCS51/>

<http://www.8052.com>

ACKNOWLEDGEMENTS

I would like to thank the following people and companies for all of the assistance and support that they have given to me throughout this project.

- Bill Gabb, Project Supervisor
- Dave Wayment, PCB Manufacture
- Roger Tomlinson, Surface mount soldering
- Tim Caine, for providing the car to carry the tests runs on
- Atmel Wireless & Microcontroller, for the supply of the DataFlash RAM components
- Hitex, with whom the liaison with Atmel was conducted, and for providing the development kit.
- Analogue Devices Inc. for the supply of the Accelerometers

APPENDIXES

List of acronyms used in this document

ABS	Anti-lock Braking System	IO	Input/Output
BIT	Built In Test	LIN	Local area INterconnect
CAN	Control Area Network	MEMS	Micro Electro-Mechanical System
CSV	Comma Separated Values	MOST	Media Oriented Systems Transport
CVS	Concurrent Version System	MPH	Miles per Hour
DVD	Digital Versatile Disc	OSI	Open Systems Interconnection
EBD	Electronic Brake-force Distribution	PC	Personal Computer
ECU	Electronic Control Unit	RAM	Random Access Memory
ESP	Electronic stability program	ROM	Read Only Memory
EEPROM	Electrically Erasable Read only Memory	RS-232	Serial Data communication standard
FLIP	FLexible-In-system Programming	TTP	Time Triggered Protocol
GPS	Global Positioning System	UART	Universal Asynchronous Receiver/Transmitter
HLP	Higher Level Protocol		
ISP	In-System Programming		

Source Code

All of the source code for the project is included on the enclosed CD-ROM.

Schematics

The following schematic diagrams have been included:

- Motherboard Board
- Memory Daughter Board
- Accelerometer Daughter Boards