School of Engineering Science
Burnaby, BC V5A 1S6
nvision-tech@sfu.ca

December 19, 2003

Dr. Andrew Rawicz
School of Engineering Science
Simon Fraser University
Burnaby, British Columbia
V5A 1S6

Re: ENSC 340 Project Post Mortem for an Automotive Diagnostic Tool

Dear Dr. Rawicz:

The attached document, *Post Mortem for an Automotive Diagnostic System*, outlines the process upon the completion of ENSC 340 (Engineering Project). We designed a system to interface with existing diagnostic ports on late-model automobiles, thereby providing hobby mechanics and sport enthusiasts with inexpensive access to detailed information from the car's computer.

This document provides an overview of the current state of the device, problems encountered during development, and future plans for the device. In addition, we compared our estimated and actual budgetary and schedules, as well as the marketability of our product. Each group member also discussed the technical experienced gained through this project.

NVision Technologies is a group of four talented, enthusiastic and creative third-year engineering students: Jozsef Dudas, Seema Jaffer, Deanna Lee, and Byron Thom. Should you have any questions or concerns, please feel free to contact me via telephone at 604-773-9712 or email at nvision-tech@sfu.ca.

Sincerely,

Joe Dudas

Jozsef Dudas
President and CEO
NVision Technologies

Enclosure: Post Mortem for an Automotive Diagnostic System

Post Mortem for an

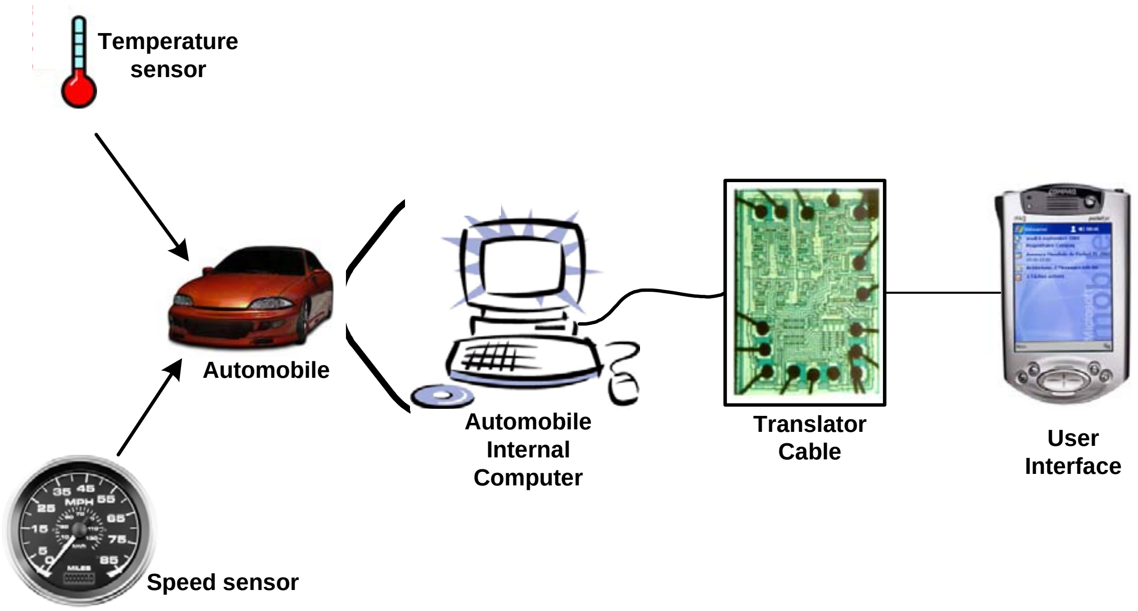
Automotive Diagnostic Tool

NVision Technologies:	Jozsef Dudas Seema Jaffer Deanna Lee Byron Thom
Contact Person:	Jozsef Dudas NVision-Tech@sfu.ca
Submitted to:	Dr. Andrew Rawicz – ENSC 340 Steve Whitmore – ENSC 305 School of Engineering Science Simon Fraser University
Issue Date:	December 19, 2003
Revision:	1

Table of Contents

1 Introduction

NVision Technologies has successfully passed over numerous obstacles and challenges in the completion of an Automotive Diagnostic Tool. For the last four months, our group has been devoted to producing the first phase of a low-cost, user-friendly device that is capable of providing users with three channels of interaction: sensor data, trouble codes, and computer reprogramming, via a personal digital assistant (PDA). This device controls all the data flow between the users and the automobile. Figure 1 shows the



system overview of the Automotive Diagnostic Tool.

Figure 1: System Overview

This document describes the current functionality of the Automotive Diagnostic Tool, problems encountered, as well as future plans for this device. Budgets, timelines and marketability is also discussed to evaluate team and product performance. Finally, each group member shares his/her valuable experience gained in this project.

During this course, we have faced enormous challenges in design, implementing, and integrating the project. The design of this device has also been revised to improve on cost, performance and functionality. Despite all the obstacles met, we are able to present a system that is just short of fulfilling all the major requirements we proposed.

2 Current State of Device

2.1 Hardware

Current implementation of the hardware meets all specifications set forth in the functional specification document.

The interface to the J1850 bus is capable of performing the level conversion and meeting the wave-shaping requirements as per the standards. Furthermore, communications with the iPaq via RS232 are functional via the transceiver, and the power supply is capable of supplying adequate power without overheating.

Providing a programming interface to write firmware to the micro-controller proved to be a difficult challenge, however, several adaptors were constructed and the product can now be programmed via a variety of commercial programmers.

2.2 Firmware

In its current state, all of the firmware works in simulation. Most of the firmware works on hardware, but there are timing issues a small part of the code.

The following diagram illustrates the firmware architecture.

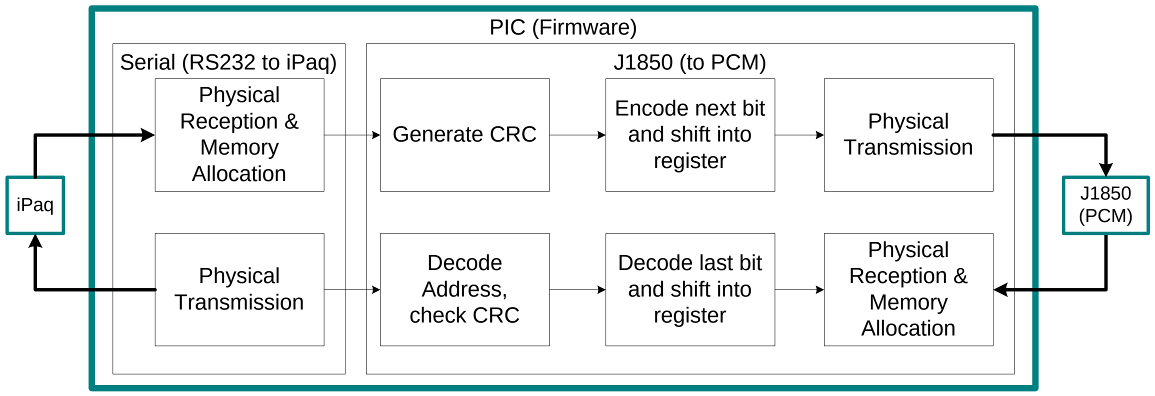

PIC (Firmware)
Serial (RS232 to iPaq)
J1850 (to PCM)
Physical Reception & Memory Allocation
Generate CRC
Encode next bit and shift into register
Physical Transmission
iPaq
J1850 (PCM)
Physical Transmission
Decode Address, check CRC
Decode last bit and shift into register
Physical Reception & Memory Allocation

Figure 2: High-Level Firmware

The translator cable's serial interface (with the iPaq) works. It was tested both in simulation and on the hardware and this component is fully functional. The following sections will describe the functionality of the J1850-related code.

2.2.1 Simulation of J1850

After simulation and timing issues arose, as we could not meet our smallest time frame (64 microseconds, which is about 600 clock cycles), the firmware was redesigned.

The firmware design evolved into interrupts with priority-levels, as some tasks were the most important to run (i.e. toggling the output pin on the port to communicate with the car's computer). Other tasks were secondary, such as setting up the next bit timer values for variable pulse width (VPW) modulation. Lastly, some tasks could occur whenever there was time, such as performing cyclical redundancy checks.

After redesigning the firmware, we simulated again, and now all of the firmware components work. These components include the RS232 and J1850 interfacing. The following lower level logic was tested in MPLAB simulation:

- Generate/check cyclical redundancy checks (CRCs)
- Decoding addresses of received packets (from the J1850 side)
- Shifting bits in from reception (from the J1850 side) and decoding symbols (start of frame, '1', '0', or end of frame)
- Encoding symbols and shifting bits out to transmission (to the J1850 side)
- Toggling ports to mimic transmission and reception

2.2.2 Hardware

The main difference between simulation and hardware is as follows: there are more functions that are required in hardware. When testing on hardware, we needed to provide information to the iPaq (for debugging and packet information), which added more clock cycles. We were already cutting it close in simulation, and now testing on the hardware, we noted a consistent time shortage to complete tasks. In its current design, on this microcontroller, we were unable to meet the real-time requirements for the J1850 VPW reception (we were able to meet transmission requirements, alone).

The following components functioned well on the hardware:

- RS 232
- J1850 Transmission

The only component that did not work on the hardware (due to timing reasons) was the J1850 Reception code.

2.3 Software

The Software Application Layer controls user interaction with the PCM via the Translator Cable. The software is multi-threaded with the serial interface to the Translator Cable and the GUI running in parallel. The serial thread controls communications with the iPaq's built-in serial connection. Packets from the GUI are sent to the serial thread where they are buffered and sent out to the Translator Cable. When packets destined for the GUI are sent to the iPaq through the serial connection, the serial thread sends a proxy for the GUI to fetch the sent data from the serial thread's memory buffer.

On the GUI side, the application software is meant to be robust with the coding of commands configurable by the user. By incorporating this feature, the GUI enables the user to access commands that are not part of the standard J2190/J2178 command set but specific to a particular model/make of car.

The software display has two different types of modes for the data coming from the PCM. For certain sensors, it makes sense to have a display that is easily updated with streaming data. A gauge display was used for this purpose. As well, digital data for up to four sensors can also be displayed at a time for user who wants a one shot reading of certain sensors.

2.4 System Integration

All of our problems arose when trying to integrate the firmware subsections and are described above.

3 Problems Encountered

3.1 Hardware

The primary issue in hardware development was programming. In particular, one PIC programmer was toasted during the making of this product, and replacement parts had to be found in a rush. In addition, in an attempt to make debugging easier, we borrowed a Microchip programmer from Dr. Rawicz. However, this required us to create yet another intricate adaptor to connect to the board, and the wiring board had to be partially reworked.

3.2 Firmware

The biggest obstacles we ran into with the firmware was due to its real-time nature.

3.2.1 **Timing Issues with the Firmware Code**

The biggest problem we encountered was with timing issues with the code. The smallest frame cycle we have to send a *high* or *low* on the J1850 bus is only 64µs. With 10 million instruction cycles per second (10 MIPS), the time it takes to run one instruction cycle is 0.1µs. It means we are limited to much less than 640 cycles. Our worst-case scenario would include transmission and reception, where the reception performs bit-by-bit arbitration, because the J1850 bus is a one-line bus. In this scenario, we must perform the following tasks:

- toggle the pin on the port,
- shift out the next bit and calculate its timer value for the next toggling,
- decode the last received symbol and shift it in

Other tasks, which include the cyclical redundancy checking and decoding of the address of the J2190 packet, can be done whenever there is time.

Since we were unable to meet the initial timing requirements, we employed a nested-interrupt architecture where the most important code was executed, then another interrupt may cut-off the current interrupt in order for the VPW transmission and reception to occur. Even after this redesign, we were unable to fully receive on the J1850 bus, due to timing restrictions.

3.2.2 MPLAB IDE Simulator

MPLAB IDE is Microchip's Integrated Development Environment that allows us to program the PIC microcontroller using C. Although we were able to simulate our code in MPLAB, certain peripheral functions cannot be tested in simulation and can only be tested on hardware.

3.2.3 MPLAB In-Circuit Debugger

We obtained an MPLAB In-Circuit debugger from Dr. Andrew Rawicz. Although the programmer worked, the in-circuit debugger did not work. Without this debugging tool, we turned to other methods of debugging, such as employing our RS232 to iPaq interface to obtain debugging data. These other methods ate up cycles, but sufficed.

3.2.4 CRC Checking

In general, the SAE standards caused us a lot of problems because they are poorly written and they all cross-referenced eachother. For example, we had difficulties understanding the CRC checking stated in the SAE J1850 standard. It took us a long time to find out what the equation really means. The following is the equation given:

$$\frac{X^{8*}D(X)+X^{n}+X^{n+1}+\ldots+X^{n+7}}{P(X)}=Q(X)+\frac{R(X)}{P(X)}$$

The information we obtained from the standard is that *D(X)* is the Data Segment Polynomial, *P(X)* is the CRC division polynomial, and $\overline{R(X)}$ is the CRC byte. The Remainder Polynomial R(X) is determined from this Modulo 2 division equation. First, we had difficulties understanding what the X^n means. Later on we find out that it is just 2^n so we know which bit is set in binary form. Secondly, once we understood the X, we didn't know how to interpret X^{8*} because we didn't know what the * is. Initially we interpreted that as a complement, because is less confusing and easier to read than $X^7 + X^6 + \ldots + X^0$ in written form. However, our interpretation was incorrect. The asterisk meant to shift *D(X)* 8 bits to the left. Thirdly, we had to find out what Modulo 2 division is, because it is different than the usual division we have always been dealing with. We put a lot of effort trying to understand the SAE standards. Others problems encountered with the SAE standards are discussed in their related sections.

3.3 Software

As with any project, many problems had to be overcome. The main issues that needed addressing revolved around the construction of the J2190/J2178 packets that were necessary to communicate with the PCM. The communication packaging set forth by the SAE necessary to communicate with the car is complex and involve many cross-references from various standards. Other issues that had to be dealt with included difficulties with the integrated development environment due to unfamiliarity with the differences between Visual C++ and Embedded Visual C++. Mr. Thom, the software lead, had no prior experience with Embedded Visual C++. With all support documentation found relating directly to VC++ instead of Embedded VC++, which has a stripped-down version of the Microsoft Foundation Classes; incorporating even simple features such as drawing a line became an arduous task to find a work-around for a proven method that works in Visual C++, but not its embedded counterpart.

3.4 Group Dynamics and Organization Chart

There were no major group dynamics issues between the team members of NVision Technologies. Although most of us had a very tight schedule due to other academic courses or thesis/co-op work, we managed to meet at least once a week to update each other on our progress. Joe was a natural leader for our team and Seema was excellent at organizing meetings and coordinating schedules to keep everyone updated and on track.

The team consisted of the Hardware, Software and Firmware group, where Joe was responsible for the hardware, Byron was responsible for the software, and Seema and Deanna share the worked on the firmware. Initially we planed to divide the work among the group members equally, but because of the close correlation between different parts, work sometimes could not be separated as easily and as equally as we had wished.

With the lack of time working together due to our tight schedules, some members ended up with a larger workload than other team members. For example, in the Firmware group, since Seema can spend more time working with Joe because of their common schedules, she has better understanding with the system and the firmware communications with the hardware. Deanna ended up working on firmware that could have been separated from the hardware, and did more documentation work. Our team was slightly restructured due to the nature of our work, to make sure every team member has contributed his/her part towards the team.

4 Future Plans

4.1 Hardware

The device must receive a significant upgrade to be fully SAE compliant. In particular, additional physical layer interfaces must be added to connect to vehicles supporting all four communications standards (J1850 VPW, J1850 PWM, ISO1440, and CAN), all of which have unique electrical interfaces. Fortunately, the remaining standards are fairly simple in their electrical implementation and could be designed using a few discrete components.

Furthermore, to make this a viable commercial product, the hardware must be miniaturized to a reasonable form factor. This shrinking could easily be accomplished through the exclusive use of surface-mount parts and printed wiring boards. Also, protection circuitry such as Electrostatic Discharge prevention and safety circuitry will have to be designed into the final product.

4.2 Firmware

Since the firmware is a stand-alone unit that cannot be modified by the user, it must be robust to withstand long hours of operation, without issues such as buffer overflows.

Due to our timing issues on the current microcontroller, in future work, the code will be optimized again and may be run on a newer, faster chip to support upgrades and redesigns.

In the future, the firmware will also need to support all four of the communication standards with the car's computer, which includes variable pulse width (VPW) modulation, pulse width modulation (PWM), controller area network (CAN), and the latest ISO communication standards.

4.3 Software

The future development of the Automotive Diagnostic Tool centres on the development of application specific software that harnesses the power and functionality afforded by the SAE Standards. Most of the added functionality that distances the automotive diagnostic tool from other available devices is found in software. Future revisions of the software will include the ability to download diagnostic data from the PCM. As well, the ability to change car parameters will also be implemented.

In the future, the software can also be ported to different operating systems such as the Palm OS and eventually PocketPhone.

5 Budget and Scheduling

5.1 Budget

Table 1 shows the estimated cost and the actual cost used in the development of the Automobile Diagnostic Tool.

Table 1: Estimated Budget vs. Actual Cost

Required Components	Estimated Cost (C$)	Actual Cost (C$)

OBD-II Interface Cable	$35.00	$21.64
FPGA	$60.00	N/A
Bus interface	$5.00	$0.00
Power Supply and Miscellaneous Parts	$30.00	$0.00
Prototype Circuit board	$30.00	$12.00
Cables, connectors and Adaptors	$30.00	$49.78
Enclosure	$20.00	N/A
User Interface (PDA)	$50.00	$200.00
Prototyping Equipment	$150.00	$150.00
Development Tools	$0.00	$0.00
Cost Over-run Contingency (20%)	$82.00	N/A
Sub Total	**$492.00**	**$433.42**
SAE Documentations	N/A	$572.65
Total Cost	**$492.00**	**$1,006.07**

The detail breakdown of the estimated and actual costs can be found in the Appendix. As shown in Table 1, some of the required components are removed in the actual cost (indicated by N/A) while other components are added. This modification is made to improve both cost and design. Some many argue that the cost was not improved because the actual cost has doubled the estimated cost. This is because approximately $600 was spent on the SAE documentations, which we initially plan to borrow it from local libraries. However, since SFU library does not have a copy of the standards and it may be a useful reference, we decided to purchase the documentations. This payment will be reimbursed by faculty.

Without any funding, out total cost matches very closely with our predicted cost. Moreover, our predicted cost for the PDA is $50.00 while the actual cost is $200.00. This is because initially one of the group members decided to pay for the PDA so he can keep it for his own use later on. Since we are within budget, the group has decided to pay for the PDA. Otherwise, we will be significantly under our predicted budget.

Table 2 shows the funding summary of our project. All costs of this project are well covered by sources of funding including NSERC and ESSEF.

Table 2: Funding Summary

Total Cost	$1006.07.00
NSERC	($250.00)
ESSEF	($200.00)
Reimbursement on SAE Documentations	($572.65)
Net Amount Paid	**($16.58)**

5.2 Schedule

Table 3 shows the Gantt chart for an estimated timeline and the actual scheduling of this project. Figure 3 shows the expected deadlines and actual deadlines for deliverables as well as major milestones. As indicated in the figures, the team matches the planned deadlines very closely at first. However, the time it takes for implementation is out of our expectation, especially for the firmware and the software. We predicted to finish firmware in 22 days; however, it took twice as long to finish all the implementation. Because of the delay in implementation, the starting dates for system integration, testing and debugging, and the post-mortem report were all postponed.

Table 3: Gantt chart

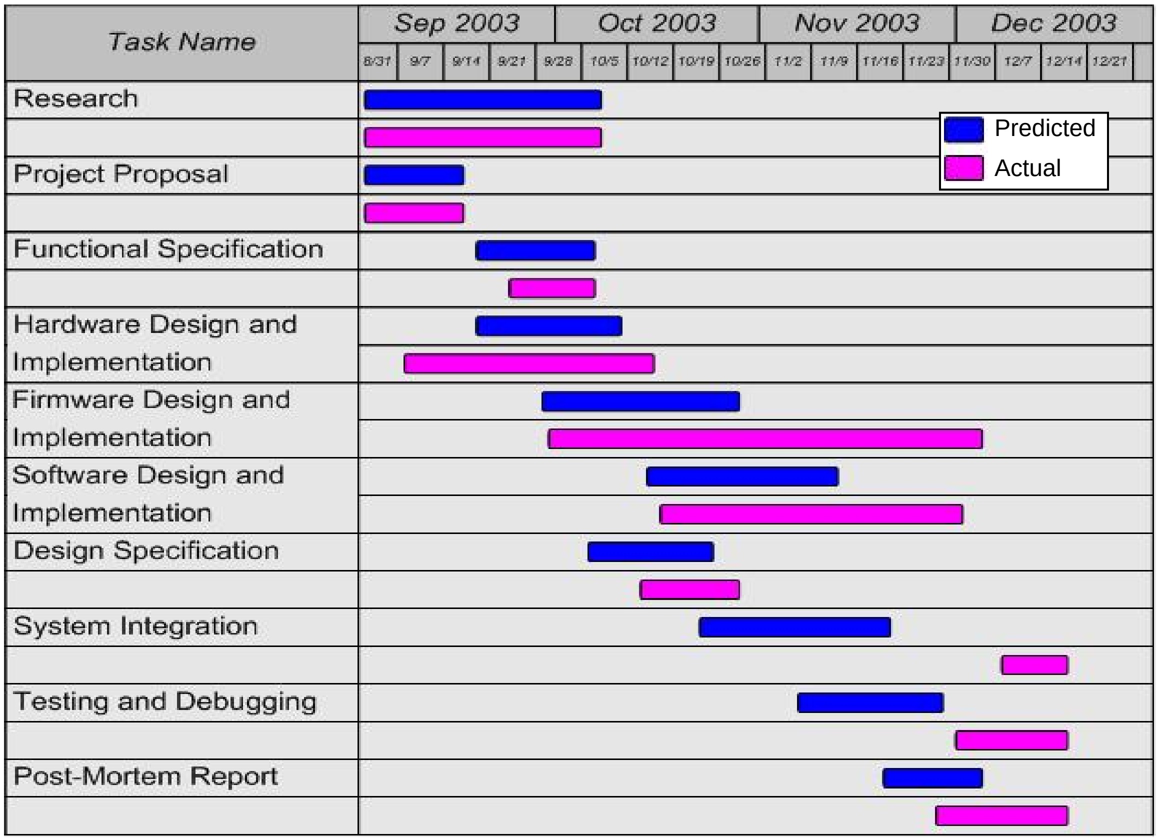


Figure 3 shows the estimated and predicted milestone. All implementations were expected to be finished by mid-November. However, as mentioned before, implementations take longer and expected, and some time was allocated for exam preparations.

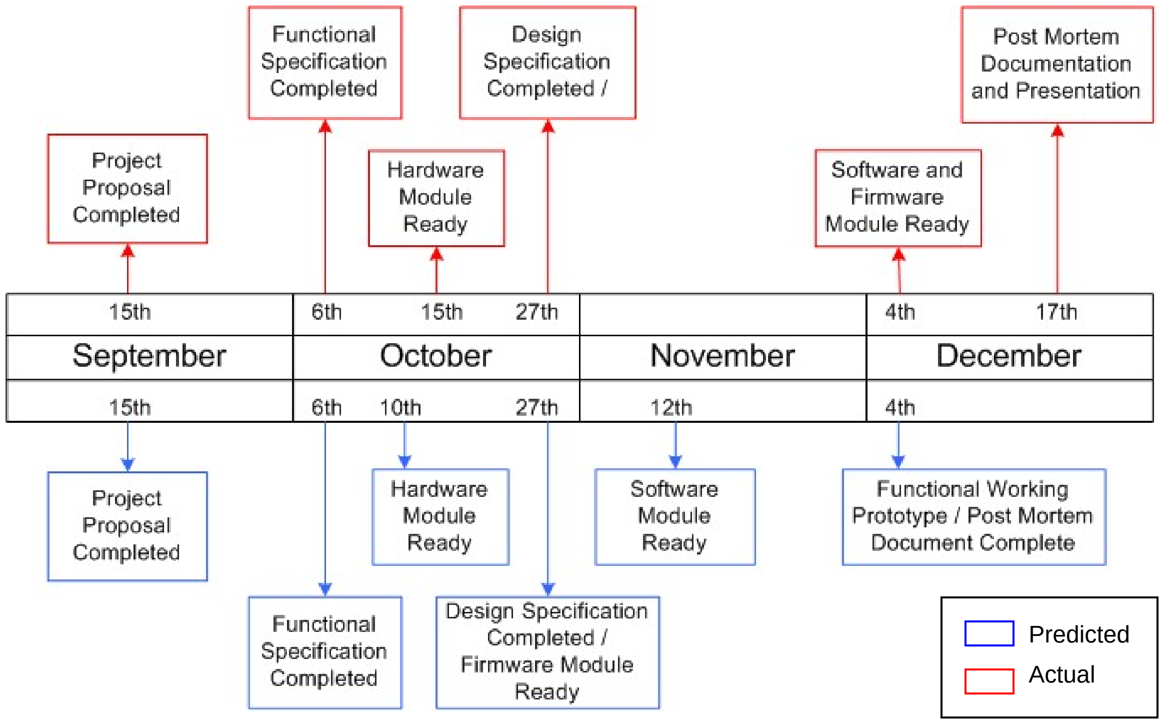

Project Proposal Completed
Functional Specification Completed
Hardware Module Ready
Design Specification Completed /
Software and Firmware Module Ready
Post Mortem Documentation and Presentation
15th
6th
15th
27th
4th
17th
September
October
November
December
15th
6th
10th
27th
12th
4th
Project Proposal Completed
Hardware Module Ready
Software Module Ready
Functional Working Prototype / Post Mortem Document Complete
Functional Specification Completed
Design Specification Completed / Firmware Module Ready
Predicted
Actual

Figure 3: Project Milestone Chart

6 Individual Experiences

6.1 Jozsef Dudas – Project Manager and Hardware Lead

As a general overseer and technical contributor to this project, I learned a great deal about team dynamics, project management, and software design.

As the leader I strived to keep each group member happy, with a satisfactory workload and sufficiently interesting work. This proved challenging as there were often lulls in between frequent storms, and it was important to keep everyone up to date. I found that always keeping people in the loop creates a much more cohesive working unit and produced more effective work.

Some of the biggest lessons I learned were with the technical aspects of this project: embedded software is considerably different from anything we are used to at this school. I am now acutely aware of that fact the microcontrollers, real-time systems, and C-compilers don't mix well. A large proportion of our headaches stemmed from the fact that our code wasn't running fast enough. On a slightly different note, I learned that Windows CE is a very different beast from its desktop cousins. The libraries are quite stripped down, the user interface is far more difficult to design for, and the overall documentation is worse than poor. Often, Microsoft claimed their systems worked very differently that they actually did, or made no reference to the embedded version at all. However, I did learn that Windows CE is still a very useful tool for a variety of project.

In all, I feel that these lessons will make me much more prepared for future projects in that I have a better understanding of some of the problems that may arise and I can better allocate time and resources to solving them.

6.2 Seema Jaffer – Firmware Co-Lead

In the past courses I have taken in Engineering Science at Simon Fraser University, projects could always be optimized enough to fit on a chip, code always worked in the end if you tried, and the point was to get things working. When I began this project, more than four months ago, I thought that the purpose of this course was to come up with a novel idea and make it work, and it was going to be like all the other courses because it would work. I did not think that we could make mistakes.

Between 305 and 340, I learned a lot more than I bargained for. I also learned that there is so much more to learn. At our current level in third year, we have learned how to apply knowledge but we never knew how to manage a project, co-ordinate a large amount of work between several people, and do your best in three months.

We all make mistakes. Some are more weighted than others. Besides these mistakes that may have affected our project, I still speak with my group members and I did try my best. Although I have heard it before, I never really considered that 'you can only do your best'.

6.3 Deanna Lee – Firmware Co-Lead

It was a very enjoyable experience to work with the team, although at the end we didn't have a fully functional product. I am very glad that the team works very well together, there were no major group dynamics issues, and that we became close friends than before working together in this project.

One minor group dynamics issue we had was that work cannot be split up equally sometimes, so of us had to write more code while others had to do more documentations. It was sometimes frustrating that a group member got stuck with his or her problem and I can't get involved because I am occupied by something else. It makes me feel like I am not contributing enough when my group members are trying very hard. This is the problem with a small team like us and we don't have enough to get involved in everything. However, working on a small team has its advantages. The bonds between group members are tighter and we are being supportive for each other.

There are several things I learn through this project. One thing I learned is that technical documentations can be very confusing and takes a lot of time to digest. I spent at least two weeks to understand one equation in the SAE documentation. It was even harder to interpret the confusing documentations when they have cross-reference with each other. Cross-referencing increases the confusion level at an exponential rate. Another thing I achieved is that I brushed up on C programming. I have perfected the use of arrays and pointers programming.

One important lesson I gained is that when doing a project, we should start from the basic and build on it, instead of trying to make everything perfect from the beginning. Otherwise, we won't get things done.

6.4 Byron Thom – Software Lead

I found this project quite trying. I had difficulties with the integrated development environment and was unfamiliar with GUI programming prior to taking on this course. In addition, as a group we learned some lessons about project management especially regarding divisions of tasks and the ability to work separately on different issues while maintaining close communications with other group members. As well, I think the best lesson I learned in retrospect when comparing how we foresaw the development of the tool and how the development really occurred. I think that four months is an insufficient amount of time to truly encompass all the tasks that are required of the course without sacrificing quality or sleep.

A. Appendix 1: Detail breakdown of Predicted Cost

Description	Comments	Our Cost
OBD-II Interface Cable		
Connector Shell	Standard connector, not easy to find	$ 30.00
Cable	6', shielded, 4 conductor	$ 5.00
	Section Total	$ 35.00
Smart Cable	*note, all parts for example so far only*	
FPGA	Decide on size required	$ 60.00
Support Hardware	Oscillator, power supply, bypass caps	$ 30.00
Transciever hardware	Some caps, resistors and transistors	$ 5.00
Circuit board	Perf-board, plated, or wire-wrap	$ 30.00
Enclosure	Plastic box, etc	$ 20.00
Cable	The cable part of 'Smart Cable'	$ 5.00
Connectors	DB-9 for serial, etc	$ 5.00
	Section Total	$ 155.00
Main Controller Unit		
Compaq iPaq 3650	Colour display, easy programming…	$ 50.00
Serial Cable for PDA	Needed for serial comms, not 100% necessary	$ 20.00
	Section Total	$ 70.00
Prototyping Equipment		
Spare PCM	For experimenting, and testing	$ 150.00
uController tools	Compilers, debuggers, etc	$ -
FPGA tools	VHDL Compilers, downloaders, etc	$ -
WinCE Dev Tool	eMbedded Visual C++, etc	$ -
	Section Total	$ 150.00
	Sub Total	$ 410.00
	Contingency percentage	20%

Total	$ 492.00

B. Appendix 2: Detail breakdown of Actual Cost

Description	**Comments**	**Our Cost**
OBD-II Interface Cable		
Connector Shell + Pins	Standard connector, not easy to find	$ 21.64
Cable	6', shielded, 4 conductor	$ -
	Section Total	$ 21.64
Smart Cable		
Support Hardware	Oscillator, power supply, bypass caps	$ -
Transciever hardware	Some caps, resistors and transistors	$ -
Circuit board	Perf-board, plated, or wire-wrap	$ 12.00
Connectors	DB-9 for serial, etc	$ -
SOIC-DIP Adaptors	Used to mount components on breadboard	$ 30.00
	Section Total	$ 42.00
Main Controller Unit		
Compaq iPaq 3650	Colour display, easy programming…	$ 200.00
Serial Cable for PDA	Needed for serial comms, not 100% necessary	$ 19.78
	Section Total	$ 219.78
Prototyping Equipment		
Spare PCM	For experimenting, and testing	$ 150.00
uController tools	Compilers, debuggers, etc	$ -
WinCE Dev Tool	eMbedded Visual C++, etc	$ -
	Section Total	$ 150.00
Reference Materials		
SAE Handbook	References for SAE Standard	$ 572.65
	Section Total	$ 572.65

Total	$ 1,006.07