Diagnostic System for Electronic Fuel Injection Engines

Pedro Maria de Sousa Melo Correia Duarte

Electronics and Computers Engineering Department, Instituto Superior Técnico – Uni. Técnica de Lisboa
Av. Rovisco Pais, 1049-001 Lisboa
pedro.duarte@binarylabs.pt

Abstract—This paper summarizes the study and prototyping of a multi-protocol low-cost console for multiplexed mechatronic peripherals in automotive environments. The work developed included the selection of the protocols to be used, the microcontroller selection and programming, and the development of different peripherals for functional demonstration.

I. INTRODUCTION

Over the last 30 years the impact of electronic components in the automotive industry has increased dramatically. Electronic peripherals are nowadays used in all sorts of functional modules inside a vehicle ranging from safety-critical systems to multimedia appliances. In order to minimize the cable harness used inside vehicles multiplexing is being used. However, different applications require different types of buses and there is, as of the writing of this paper, no device able to connect to all of those. This project consists on a single low-cost application towards this end connecting a CAN bus interface and a LIN bus interface with an on-board computer using USB.

The project includes both the hardware development and the software for all of the modules designed. As of modules, three different and independent areas can be emphasised: the Main Board – console with the interfaces to all buses (CAN and LIN) and to the on-board computer (USB); the on-board computer that will host the monitoring application and user interface; and the mechatronic nodes which demonstrate the functionality of the application.

II. PROTOCOLS

SAE, Society of Automotive Engineers has defined several categories to classify buses used in automotive environments. The classes of interest to this project are class A, B and C which are based upon the bit rate of the serial protocol other than the end towards which it will be used. After careful analysis of the following protocols: IEBus; MI-bus; J1708; LIN subbus; J1850; CAN; IDB-C; MOST; MML; D2B; FlexRay; Byteflight; BST; Intellibus; DSI; SmartwireX and IDB-1394. [1]

Due to node cost, bit-rate and popularity of the bus, the final choices were CAN bus and LIN subbus. These protocols reach all the 3 SAE classes being LIN a class A protocol and CAN either class B or C, depending of the configured bus bitrate.

The following table illustrates some bus specific data that was taken into account in order to make the final decision. [2]

<table>
<thead>
<tr>
<th>Bus</th>
<th>Max Bit Rate</th>
<th>Max Nodes</th>
<th>Popularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1708</td>
<td>100 kbps</td>
<td>NA</td>
<td>Medium</td>
</tr>
<tr>
<td>LIN</td>
<td>19.2 kbps</td>
<td>16</td>
<td>High</td>
</tr>
<tr>
<td>MI</td>
<td>20 kbps</td>
<td>8</td>
<td>Medium</td>
</tr>
<tr>
<td>IEBUS</td>
<td>18 kbps</td>
<td>50</td>
<td>Medium</td>
</tr>
<tr>
<td>J1850-X</td>
<td>10 / 40 kbps</td>
<td>32</td>
<td>Medium / High</td>
</tr>
<tr>
<td>CAN</td>
<td>1 Mbps</td>
<td>32</td>
<td>High</td>
</tr>
<tr>
<td>IDB-C</td>
<td>250 kbps</td>
<td>16</td>
<td>Medium</td>
</tr>
<tr>
<td>MOST</td>
<td>25 Mbps</td>
<td>64</td>
<td>High</td>
</tr>
<tr>
<td>MML</td>
<td>110 Mbps</td>
<td>16</td>
<td>Medium</td>
</tr>
<tr>
<td>D2B</td>
<td>25 Mbps</td>
<td>50</td>
<td>Medium</td>
</tr>
<tr>
<td>FlexRay</td>
<td>10 Mbps</td>
<td>NA</td>
<td>High</td>
</tr>
<tr>
<td>Byteflight</td>
<td>10 Mbps</td>
<td>NA</td>
<td>High</td>
</tr>
<tr>
<td>BST</td>
<td>250 kbps</td>
<td>62</td>
<td>Medium</td>
</tr>
<tr>
<td>Intellibus</td>
<td>12.5 Mbps</td>
<td>64</td>
<td>Medium</td>
</tr>
<tr>
<td>DSI</td>
<td>150 kbps</td>
<td>16</td>
<td>Medium</td>
</tr>
</tbody>
</table>

Although more information, such as node cost should be taken into consideration, there is almost a direct relation between the bit rate of the bus and equivalent cost of each peripheral node.

III. MAIN BOARD

The Main Board is one of the three distinct areas of the project. On this chapter the circuit synthesized for the Main Board of the application is explained. The main board is the hardware responsible for the communication with the on-board computer, the CAN clusters and LIN clusters.

A. Overview

The microcontroller chosen, 18F4550 from Microchip, was selected regarding the output drivers and ports needed. However, the hardware for the TTL voltage level conversion necessary by each protocol is not included. The final board required at least one transceiver for CAN, 1 transceiver for LIN and a standalone controller for the CAN driver. Other components such as coupling capacitors, voltage regulators and status LEDs are also contemplated here.

B. Microcontroller
The main board of the application was designed to be as low cost as possible. Although it would be expected to implement FlexRay, MOST, CAN, LIN and USB communication in a future version, the lack of documentation and hardware drivers for FlexRay, the cost of MOST interfaces as well as the lack of output for all of these protocols using a single MCU or DSP forced the final choices to be: CAN, LIN and USB interfaces.

As a world leader of MCU products, Microchip was selected as the provider for the microcontroller. It offers several solutions with multiple EUSART for LIN communication, CAN drivers for CAN communication and Hi-speed USB interfaces for USB communication.

Although other vendors such as Texas Instruments and ST were considered, the lack of development tools and the cost of the microcontrollers led to the selection of Microchip as the MCU provider. However, for FlexRay and MOST implementation, another comparison should be made. The interface requirements for CAN and LIN are far inferior to the ones that are expected for these latter protocols and so is the operating frequency.

As the USB communication is the greatest restraint on the selection of the MCU, the selection of models was firstly based on those who have an USB interface. However, there is no Microchip MCU that supports both USB and CAN interfaces. Microchip workaround was to create a standalone controller MCP2510 that could receive commands through a SPI interface and output it to the CAN bus. The MCP2510 was recently upgraded to its second generation version - MCP2515.

The MCU that was now sought should implement an USB interface, SPI port and a LIN compatible EUSART. Once again there was no MCU that had all these interfaces implemented in independent ways. All the available microcontrollers shared the SPI TX pin with the EUSART RX pin. The possible solutions considered were to either emulate a LIN compatible EUSART in software or to implement an intelligent use of the I/O port that was shared by SPI and the EUSART.

The hardware requirements led, in the end, to the choice of the microcontroller PIC18F4550. It includes an internal PLL that is able to produce a 96 MHz clock signal from a 4 MHz input needed for USB high-speed communication (12 Mbps), has a EUSART compatible with the LIN break signal and an SPI port for 10 Mbps communication.

The 18F4550 is high-performance, low power consumption device that comes in 40 or 44 pin packages. It has three 8-bit ports, one 7-bit port and one 4-bit wide port for multipurpose functionality although some pins are multiplexed with an alternate function from the peripheral features on the device. In general, when a peripheral is enabled, those pins may not be used as general purpose I/O pins. The microcontroller also supports three distinct 16-bit timers and one 8-bit timer that may be set up as counters and may have prescaler and postscaler features. Interrupts are also supported although only two priority levels (High and Low) are implemented. Flag-polling must be implemented in order to emulate greater priority hierarchy.

The microcontroller implements special circuitry that allows for PWM generation, serial communication using RS232, SPI, I2C and USB protocols. Its general purpose ports may also be used for software emulation of such ports although the lack of dedicated registries may reduce significantly the transmission speed limits.

C. Transceivers

Both CAN and LIN buses require coupling to the data buses. The microcontroller outputs TTL levels and therefore dedicated transceivers are required. The Microchip solutions for the two cases are the MCP2551 for CAN and the MCP201 for LIN.

1) MCP2551 - CAN:

![Fig. 1 Block Diagram of MCP2551](image)

[3] On the TX circuitry, as CAN protocol requires a 2-wire differential medium, the conversion from the TTL logic levels to CANH and CANL signals is done by a driver that controls two transistors connected to either pull-up or pull-down resistors. On idle mode the transistors are OFF and the CANH and CANL are imposed by the bus.

On RX circuitry, the values of CANH and CANL are compared and the result of the exclusive OR is outputted to the RXD pin. If the voltage is different between both pins – CANH and CANL – a TTL High is detected, if the voltage is the same a TTL Low is detected.

2) MCP201 - LIN:

![Fig. 2 Block Diagram of the MCP201](image)

[4] In the TX circuitry, the TX pin has an internal pull-up resistor to Vreg setting the bus at High (recessive) state. When TXD is low, the LIN pin is in Low (dominant) state. If the Thermal Protection module detects an over temperature while the transmitter is imposing the dominant state, the transmitter is turned off until a normal temperature level is detected.

The RX circuitry is a standard CMOS output stage and follows the state of the LIN pin. The RXD pin is internally connected to Wake-up logic to provide built-in wake-up functionality.

D. CAN Controller
The MCP2515 is the standalone controller that implements receive and transmit buffers, masks for message ID filtering, driver rules for arbitration, driver rules for frame interpretation and driver rules for frame generation. It has 18 pins, most of them for output of internal conditions and bus status. The controller requires a dedicated oscillator in order to be able to define a precise bit time. It also has 4 dedicated pins to serial communication and 2 pins dedicated to transceiver interface.

![Fig. 3 Block Diagram of the MCP2515](image)

[5] The CAN protocol uses a Non Return to Zero encoding which does not encode a clock within the data stream, therefore the receiver’s clock must be configured by the receiving nodes and synchronized to the transmitter’s clock. For accurate detection, the receiver must implement some kind of PLL synchronized to data transmission edges, in order to synchronize and maintain the receiver’s clock. The CAN protocol includes bit stuffing to ensure that an edge occurs at least every 6 bit times to maintain the Digital PLL synching.

### Bit Time:

CAN defines the most elementary time unit as Time Quanta – TQ – and the Nominal Bit Rate (NBR) as the number of bits per second transmitted by an ideal transmitter without resynchronization. The NBR is made up of non-overlapping segments each made up of several TQ described in Fig 4.

![Fig. 4 Nominal Bit Segments](image)

Since it was used a 20 MHz crystal as the controller clock and taking into account the specifications of CAN bus for the TQ distribution among nominal bit segments, the final distribution was:

- Synch Segment: 1 TQ
- Propagation Segment: 2 TQ.
- Phase Segment 1: 4 TQ.
- Phase Segment 2: 3 TQ.

### Interruption Handling:

In order to minimize the queries to the controller, the RXnBF pins provide a comfortable way to detect an incoming message. As soon as the RX buffer receives a message, an interruption may be triggered (if previously configured for that effect) and inform the MCU that there is new information to be fetched.

On this application the RXnBF for buffers 0 and 1 will be activated and connected to general purpose I/O pin on the MCU.

### E. Status LEDs

The main board of the application will have a visual output of its internal state defined by four status LEDs. The first two leds – LED 1 and LED 2 – will be used for USB interface diagnostic and the blinking code is the same as the defined by Microchip USB driver for the 18F4550 USB interface.

<table>
<thead>
<tr>
<th>USB status</th>
<th>LED1</th>
<th>LED2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suspended State</td>
<td>Toggle</td>
<td>Equal to LED1</td>
</tr>
<tr>
<td>Detached State</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>Attached State</td>
<td>ON</td>
<td>ON</td>
</tr>
<tr>
<td>Powered State</td>
<td>ON</td>
<td>OFF</td>
</tr>
<tr>
<td>Default State</td>
<td>OFF</td>
<td>ON</td>
</tr>
<tr>
<td>Address State</td>
<td>Toggle</td>
<td>OFF</td>
</tr>
<tr>
<td>Configured State</td>
<td>Toggle</td>
<td>Opposed to LED1</td>
</tr>
</tbody>
</table>

### F. External Connectors

The main board requires several connectors to interact with three different buses, for future testing and debugging and for firmware updates with the ICD2.

Although there is not strict connector imposed to be used on CAN and LIN bus, the most common are the same D9 connectors used in other serial communication protocols such as RS232. As the type of connector used does not impose restrictions on the type of connectors that are to be used by the other nodes, on this board the interfaces will be implemented with RJ connectors for size reduction purposes, except for the USB interface.

- CAN and LIN will use a RJ10 female connector since they only require up to four optional pins.
- ICD2 will use a RJ12 female connector since this is the default connector used by Microchip programmer: ICD2.
- USB will use a female USB type-B connector, used mostly on peripheral devices such as the main board.

### G. Power Supply

The Main Board requires a power supply of 200 mA and 5 V. Since the power must be obtained from the car battery and this is always above 6 V and is supposed to be usually between 11.5 V and 13 V, the converted needed is a buck or step-down DC-DC converter. Ordinary power regulators such as 7805 have very high power dissipation due to the internal configuration. They are designed to dissipate the remaining voltage and as the input voltage tends to be greater, the power consumption is even bigger.

The chosen converter was the LT1776 since this buck is a low power consumption regulator. The block diagram shows the internal oscillator, the control circuitry used to generate
the PWM signal that controls the switch and the protection circuitry all in a monolithic die. This device also supports input voltages up to 60V, far higher than ordinary power regulators. [6]

The LT1776 is supposed to be assembled in a configuration similar to the one depicted in Fig. 5, making this the layout chosen for the power supply of the Main Board.

![Diagram of LT1776](image)

**Figure 1**

**H. Software**

The microcontroller firmware flowchart can be summarized to the diagram in Fig. 6.

![Flowchart of the LIN routine](image)

**Fig 6. Main Board: Flowchart of the LIN routine**

Although LIN is a popular protocol, there are very few implementations of a complete LIN 2.0 driver. Application Notes AN1099 from Microchip implements a complete driver for the MCU families PIC18XXXX. The driver had to be adapted not only for the MCU used, but also for compatibility with the dynamic schedule-table structure engineered for this particular project.

The flowchart of the LIN communication routine only focuses 1 of the 2 exiting routines: the polling routine (for process of successful received messages) and the interruption routine (for communication with the other nodes).

As for the interruption service routine, since the LIN driver is fully implemented in software unlike the driver for CAN used on this project, all of the stages of transmission are contemplated.

The driver assumes that the master node may be in 6 different modes:

- **SynchBreak**: Waiting for a SynchBreak (Break) character consisting of at least 13 Low bits. This data configuration flags the framing error bit on the RCSTA register of the MCU and is the only time that the data is taken into consideration after detecting an error.
- **SynchByte**: After the SynchBreak character, the master sends a Synch character for baud rate detection purposes on slave nodes. The Synch character has the hexadecimal code 0x55 in order to maximize the transitions between logical levels.
- **IdentifierMode**: After the SynchByte mode the master sends the identifier byte of the scheduled message. The identifier has only 6 effective bits being the last 2 for parity check. If the master detects a valid identifier it processes the associated header and determines whether to set Xmit/Receive mode or to set SynchBrake mode.
- **ReceiveMode**: While on ReceiveMode the master is supposed to accept all incoming bytes and according to the length of the expected message
verify the CRC byte. In case of a successful reception the successful reception flag is set.

- **Xmit**: While on Xmit mode the master keeps transmitting the data bytes stored in the source buffer. It compares the sent byte with the received byte since the RXIE has been disabled. In case of success the master sets the successful transfer flag. In case of error, the transmission error flag.

- **Sleep**: While on sleep mode the master only monitors a wakeup signal. In case of a successful wakeup signal is detected it starts the EUSART and sets the baud rate register SPBRG.

USB Communication

The USB driver used on this project is a modified version of the provided driver by Microchip™ on their demonstration board PICDEM USB. It is configured as a polling driver that keeps checking for newly received data frames. The polling routine is the one mentioned on the main routine: ProcessIO().

The driver loads a structure of 64 bytes with the received data named DataPacket. This structure had to be modified to be compatible with the protocol developed for this project. The protocol will develop be reference as to MBP – Multi Bus Protocol.

### Table III

<table>
<thead>
<tr>
<th>Start</th>
<th># bytes</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Type</td>
<td>Descriptor of the type of message.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>BUS</td>
<td>Identify which bus the message is intended to be sent to.</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>PORT</td>
<td>For future usage, in case multiple ports of the same bus are implemented.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For instance two CAN interfaces.</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>DeviceID_MAJ</td>
<td>For device identification, Vendor ID for instance.</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>DeviceID_MIN</td>
<td>For device identification, Product ID for instance.</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>InstructionID</td>
<td>Instruction ID used for the destination protocol bus.</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>length</td>
<td>Length in bytes of the data field.</td>
</tr>
<tr>
<td>17</td>
<td>39</td>
<td>data</td>
<td>Data Field. May only contain part of the data do be transmitted. The length of the valid data on this field is determined by the length field.</td>
</tr>
<tr>
<td>56</td>
<td>8</td>
<td>reserved</td>
<td>Reserved for future upgrades on the protocol.</td>
</tr>
</tbody>
</table>

CAN Communication

CAN communication has no software driver implementation in this firmware version. The device used for the driver – MCP2515 – was already explained. The MCP2515 uses the SPI interface to communicate with the MCU and requires a dedicated crystal for Time Quanta generation.

The driver has the SPI port configured to run at 5 MHz and requires configuration information from the MCU. The MCP2515 needs to be set to configuration mode for special registers access, namely for the ones responsible for the amount of TQ available to each of the nominal bit time segments.

The configuration routine starts with a reset command, hexadecimal code 0xC0. The reset command sets the register values to the Power-On Reset. The following command sets the controller to configuration mode by acting upon the CANCTRL register. On configuration mode access to CNF1, CNF2, CNF3 and filtering registers is gained.

CNF1 register defines the length of Synchronization Jump Width – SJW – and the baud rate prescaler. The SJW is required for unreliable oscillators with poor accuracy and in this particular project may be set to its minimum: 1 TQ. The baud rate prescaler is used to define the length of the TQ as a multiple of the oscillator period.

After configuring the bit timing, the interruptions generation must be customized in order to be able to monitor interrupts whenever a message arrive instead of keep polling the controller. The registers responsible for interruption configuration are: BFPCTRL, CANINTE and CANINTF.

The CANINTE enables the interruptions that will trigger the generic INT pin. This pin will be only used for error detection while the BFPCTRL register configures the RXnBF pins as interrupt pins for successfully received messages.

IV. ON-BOARD COMPUTER

On this chapter the on-board computer will be analysed. Decisions will be made towards the best machine to host the application as well as the application itself.

A. Hardware

The environment inside the automobile is problematic. The temperatures may vary from -40°C to 90°C depending on the geographic location and proximity to the engine. Also, the electric current and voltage level provided by the battery may oscillate depending on the number of devices using the power supply such as headlights, start-up engine and the sound system amplification among others. The mechanical impact of the road conditions and driving skills also imply certain restraints to the On-board Computer components such as the mobility of the needles in magnetic storage devices. Taking all this factors in consideration the system sought for good performance would be a real time computer, running a light kernel operating system and with a Flash memory card as a HDD.

As the system is supposed to be suspended during the period of time when the automobile is not working and have a fast wake-up when the automobile is turned on, the hardware
selected should be able to run and embedded operating system with a few seconds maximum reboot time such as Windows CE or Embedded Linux. Also the system should provide USB ports for communication with the main board and a touch screen for driver interface. The power supply should be oriented for low power consumption and have high tolerance for the battery transients.

**Processor**

There are two major companies devoted to low power X86 processor design: VIA and Intel. However, VIA has a larger catalogue on this area and has been the main supplier for Car-Computer systems in the last years. A comparison based on the available models from VIA was made and the figures of merit calculated. [7]

<table>
<thead>
<tr>
<th>CPU</th>
<th>CPU MHz</th>
<th>RAM MHz</th>
<th>TDP Watt</th>
<th>CPU* TDP</th>
<th>CPU^*RAM / TDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIA C7</td>
<td>2000</td>
<td>800</td>
<td>20</td>
<td>100,00</td>
<td>80000,00</td>
</tr>
<tr>
<td>Via C7-M</td>
<td>1800</td>
<td>400</td>
<td>18</td>
<td>100,00</td>
<td>40000,00</td>
</tr>
<tr>
<td>Via C7-M</td>
<td>2000</td>
<td>400</td>
<td>20</td>
<td>100,00</td>
<td>40000,00</td>
</tr>
<tr>
<td>Via C7-M</td>
<td>1600</td>
<td>400</td>
<td>15</td>
<td>106,67</td>
<td>42666,67</td>
</tr>
<tr>
<td>VIA Eden</td>
<td>600</td>
<td>400</td>
<td>5</td>
<td>120,00</td>
<td>48000,00</td>
</tr>
<tr>
<td>Via C7</td>
<td>1800</td>
<td>533</td>
<td>15</td>
<td>120,00</td>
<td>63960,00</td>
</tr>
<tr>
<td>Via C7</td>
<td>1500</td>
<td>533</td>
<td>12</td>
<td>125,00</td>
<td>66625,00</td>
</tr>
<tr>
<td>Via C7-M</td>
<td>1500</td>
<td>400</td>
<td>12</td>
<td>125,00</td>
<td>50000,00</td>
</tr>
<tr>
<td>VIA Eden</td>
<td>500</td>
<td>400</td>
<td>3,5</td>
<td>142,86</td>
<td>57142,86</td>
</tr>
<tr>
<td>VIA Eden</td>
<td>400</td>
<td>400</td>
<td>2,5</td>
<td>160,00</td>
<td>64000,00</td>
</tr>
<tr>
<td>VIA Eden</td>
<td>800</td>
<td>400</td>
<td>5</td>
<td>160,00</td>
<td>64000,00</td>
</tr>
<tr>
<td>VIA Eden</td>
<td>1200</td>
<td>400</td>
<td>7</td>
<td>171,43</td>
<td>68571,43</td>
</tr>
<tr>
<td>VIA Eden</td>
<td>1000</td>
<td>400</td>
<td>5</td>
<td>200,00</td>
<td>80000,00</td>
</tr>
<tr>
<td>VIA Eden</td>
<td>1000</td>
<td>400</td>
<td>5</td>
<td>200,00</td>
<td>80000,00</td>
</tr>
<tr>
<td>ULV</td>
<td>1500</td>
<td>400</td>
<td>7,5</td>
<td>200,00</td>
<td>80000,00</td>
</tr>
<tr>
<td>Via C7-M</td>
<td>1000</td>
<td>400</td>
<td>5</td>
<td>200,00</td>
<td>80000,00</td>
</tr>
<tr>
<td>ULV (770)</td>
<td>1500</td>
<td>400</td>
<td>7,5</td>
<td>200,00</td>
<td>80000,00</td>
</tr>
<tr>
<td>Via C7-M</td>
<td>1200</td>
<td>400</td>
<td>5</td>
<td>240,00</td>
<td>96000,00</td>
</tr>
<tr>
<td>ULV</td>
<td>1000</td>
<td>400</td>
<td>3,5</td>
<td>285,71</td>
<td>114285,71</td>
</tr>
<tr>
<td>Via C7-M</td>
<td>1000</td>
<td>400</td>
<td>3,5</td>
<td>285,71</td>
<td>114285,71</td>
</tr>
<tr>
<td>ULV</td>
<td>500</td>
<td>400</td>
<td>1</td>
<td>500,00</td>
<td>200000,00</td>
</tr>
</tbody>
</table>

Through basic analysis of the table above it is possible to verify that VIA EDEN [15] processors have the best ratio CPU clock / Maximum power consumption. Excluding the ULV (Ultra Low Voltage) versions, which are extremely hard to find, the best option would be a VIA EDEN 1.0GHz with 400 MHz FSB for RAM communication. This processor has a ratio of CPU Clock * RAM FSB / Watt equivalent to the 2.0GHz C7 computer, a state-of-the art processor. However, VIA EDEN 1.0GHz is not sold with passive cooling in ITX form factor and a fan is extremely undesired in a car computer since it means more power consumption and mechanical-intolerant systems. The only VIA EDEN 1.0GHz fanless distribution had a nano ITX form factor and was twice as expensive as the VIA EDEN 1.2GHz.

The chosen processor was the second on the list, the 1.2 GHz VIA EDEN which was distributed with passive cooling on an ITX motherboard. Although there is an increase of 2 W in the processor power consumption, there is also an increase in Clock speed and the safety of a passive cooling system.

**Power Supply**

The power supply for automotive applications is already extremely reliable. The main suppliers in the automotive industry are Oppus and Morex. Prices range from €30 to €80 and reflect the functionality of the power supply. Most low cost powers supplies do not provide the connector for ITX motherboards or the voltage tolerance that is expected.

M2-ATX-HV is a 140 W, 6 V to 32 V wide input DC-DC power supply. It has intelligent detection of ignition allowing for special commands to the motherboard ON/OFF switch, is prepared for engine cranks and fits in most popular SBC form-factor PC enclosures. Another extremely useful functionality is the battery monitor. M2-ATX constantly checks the battery level and if it crosses safety limits M2-ATX does a full shutdown until secure levels are reached again.

**B. SOFTWARE**

The software for the On-board Computer was divided into three sections:

- Operating System
- Graphical User Interface
- Application Kernel

As for the list items 2 and 3 two different languages where chosen: C++ and HTML/JavaScript.

The application layer consisted of an USB interface (for Main Board communication), Multi Bus Protocol driver (for command processing) and a graphical interface (for the vehicle driver). Once again the new trends were analysed and a web interface was chosen. Some renowned companies such as Symantec or Hewlett Packard started using web interfaces for end-user input and output. The reason has to do with shorter developing time and compatibility with web services. As for USB communication, C++ was a comfortable choice since the Microchip driver for the USB device is written in C++. Also C++ has all the advantages of C programming plus support for objects and garbage collection features. The communication between the C++ USB driver and the web interface is done with ActiveX technology.

1) ActiveX

In order to integrate C++ modules with the user interface, a third interface mechanism was needed. The chosen technique was an ActiveX interface. ActiveX interface allows for the C++ module to be accessed from an HTA, VBA or even a webpage, meaning that multiple interfaces could be developed regardless of the language used in the core module. ActiveX is another name for Object Linking and Embedded Automation and is mostly used for reusable objected oriented software components such as the one developed in C++.
2) HTA – Web Interface

The HTA is the Microsoft answer for HTML-based trusted applications. It consists on a trusted application with permissions to manipulate the file system structure or even system registry. In HTA, HTML is used for layout description and JScript or VBScript are used for data processing and scripting capabilities. HTA have been used by Symantec for Antivirus GUIs and Hewlett Packard Scanning tools GUIs. Although it is not recommended for exhaustive processing, HTA in combination with ActiveX modules developed on lower level languages such as C++ or C# is a very powerful tool.

In this project in particular the flexibility of HTA allows easy development for third party modules, layout customization or application upgrades.

3) C++ and C#

C++ is the object oriented approach for C. Although its syntax complexity for classes manipulation is target of significant criticism, its overall performance, support for assembly and C coding and support for classes, turns it into a good choice for this project. The Microchip driver is also written in C++ making the development of the data handling code easier. C++ also supports ActiveX interfaces which are necessary for data exchange with the HTA GUI.

C# is the .NET response to object-oriented C. With full access to the .NET framework, garbage collector and much more user friendly syntaxe than C++, C# is becoming a strong high-level language with no more than 3 years of existence. It was used for the ECU emulator done in order to test one of the peripherals explained on the next chapter.

C. OPERATING SYSTEM

There multiple light operating systems that can be used for embedded applications ranging from Embedded Linux to Microsoft Windows CE. However, with the release of MS Windows CE 6.0 [16] a platform builder was made available allowing for greater customization of the kernel thus making the operating system more flexible and application-oriented. Recent cellular phones, Personal Digital Assistants, automotive built-in computers or even systems for motion-sensitive door controllers such as those used on malls and other commercial areas have been running on distributions of Windows CE 6.0. The most popular distributions at the moment are Windows Mobile and Windows Automotive.

Windows CE 6.0 kernel may run under 1 megabyte of memory and can be programmed into a ROM preventing any disk storage. Also Windows CE is a real-time operating system with deterministic interrupt latency supporting up to 256 priority levels for interruptions. CE 6.0 raised most of the limitations from CE 5.0 such as the 32 process limit (now 32768) or 32 megabyte virtual memory limit which now is up to 2GB per process. [8]

Windows Mobile is credited as one of the reasons for the PDA market increase in 40% in the first quarter of 2007. With the support for the .NET framework the number of applications for Windows CE is rising exponentially and it is foreseen that it will be the most popular OS for embedded systems in a near future.

It was based on this fact that the decision to use a Microsoft platform was made. As development for Windows CE required more man-weeks than were previously assigned to software development, the OS used was Microsoft Windows XP SP2 for 2 reasons:

- The migration from Windows XP to Windows CE is not difficult.
- In the worst case the software could run on a light distribution of Windows XP called Windows XP Embedded which provided the best of both worlds.

Windows XP Embedded is a customizable version of Windows XP SP2 on which drivers and unnecessary functionality may be removed in order to minimize the kernel load. The only restraint of Windows XP Embedded resides on the fact that it can only run on X86 architecture processors such as Intel’s Pentium4 while CE may run on a multitude of processors such as ARM, MIPS and Hitachi SuperH.

V. EXAMPLE APPLICATIONS

In order to test the application, three demonstration applications were designed. The first one uses the LIN bus and the others use CAN bus.

A. LIN – Suspension Control

LIN – Suspension Control is intended to control the strength of the four shock absorbers installed on the automobile. The strut system is a Kayaba AGX with 4 mode of operation. The modes are selectable by rotating a small circle on top of each shock absorber.

![Fig. 7 – Shock Absorber mode selection](image)

The desired mode of operation is selected by inserting a small screw-driver in the white straight line and rotate de inner black circle so that the small white dot is in the desired quadrant. In the figure above the white dot is selecting the mode 1.

The idea behind this first application is to read and write the new mode for each of the shock absorbers. The command will be sent via the USB interface to the Main Board, the Main Board will understand the command as a LIN command and pass it through the LIN interface. On the strut side, the LIN slave that subscribed the message identifier will process the desired value for each quadrant and output a control signal to each servo motor controlling the shock absorbers.
The software for the On-board Computer will be a simple plug-in for the software. The plug-in consists of an HTML file for the layout of the application: CFGHTML.html and scripting file: CFGHTML.js. Upon the loading of the module the following user interface will be displayed and different modes may be selected or configured to each shock absorber.

Fig. 8 Suspension Control GUI

The hardware implementation was done with a microcontroller PIC18F4585, a LIN transceiver MCP201 and four Futaba S3003 servos.

The specification of the wave form must be sent as well as the desired mode for servo motors other than the default may be installed. The wave form used for position control on servo motors is a PWM wave usually with a 50Hz frequency and a duty cycle ranging from 2% up to 12%.

The new values are written in the non-volatile EEPROM of the chip so that they can be restored after a Power-On Reset.

B. CAN – ECU Datalogger

In the beginning of 1990, HONDA started monitoring approximately 30 sensors, ranging from critical functionality, such as the engine temperature, to optional devices. Examples of optional devices are the Knock sensor, Air Conditioned, or Electrical Load Detector (ELD).

It is of interest to be able to monitor these values and process the data as the automobile driver may wish. For instance, when the engine has not the right air fuel ratio, the O2 sensor may output a signal of rich or lean air fuel mixture. It is important for the driver to be able to realise this in order to be able to fix the problem. Also, the only temperature warning in an automobile is a simple analogue pointer that keeps rising in case of overheating. Most temperature damages are due to lack of perceptive warning about the internal temperature of the engine. If there was a way to monitor this sensor and act accordingly over its output, most damages would probably be avoided.

The following section describes the procedures made in order to be able to retrieve this information from the vehicle by reverse-engineering of the ECU.

ECU reverse engineering

The Signal Processing in these ECUs is done by an OKI MCU – 66207. While it is possible to track all sensors signals from the Sensor connectors back to the sensor itself and analyse the signal conditioning by careful examination of the conditioning circuitry, it is not possible to read the code of a code protected MCU and determine the algorithm that is used by the MCU to process the data. This was supposed to make the task of reading the sensors via the MCU impossible. However, a design flaw was found on the OKI microcontrollers allowing the retrieval of program memory.

OKI MCUs can use both an internal or external ROM for program memory storage. The selection of the memory to be used is done by setting the digital input EA (pin 27) to either 0 V or 5 V. Since the MCU do not latch the value of the EA pin on reset, it is possible to redirect the data from the original ROM to the serial port and read it with a computer by flipping the state of EA and load a special routine from an external EPROM. Several software applications have been developed in order to read the code from the MCU and the binary files are available to download in the Internet, on pgmfi.org site.

The code has been mapped out by several individuals and the locations in the RAM where the various sensors values are well known.

In order to make the MCU run from an external ROM it is necessary to program with the original code a ROM of 32 kB (equal to the internal ROM of the OKI 66207). The ROM used was the EEPROM 27C256 with a time response below 170 ns. This time constraint was defined based on observation of the factory-chipped ECUs that already have code running from external ROMs (for fine tuning purposes).

The code changes consisted only on:
- Serial Port Reception ISR
- Checksum validation

The code inside the OKI MCU validates the checksum of the program memory and in case of defective coding the ECU enters a limp-home mode where very poorly chosen, yet safe, spark advance and injector pulse values are set. This mode is notified by a CEL/MIL on the dashboard.

In order to bypass this verification, the code is altered with an unconditional jump to the end of the checksum verification.

The idea behind changing the original MCU code is to be able to rewrite the Serial Port ISR and output through the serial port the value of specific RAM positions based on the command sent. This way it is possible to access sensor specific data taken that the RAM addresses for these values are already known.

The user interface to access this data is the one depicted in Fig. 8.
This application was tested with a C# emulator of the ECU code programmed to the external EEPROM. The application connects to a serial port on the host computer and reads the port was if the ISRs on the MCU were being triggered. When a valid commands is detected the code emulates the microcontroller code and all the values may be adjusted at will in order to verify the data display in the ECU Datalogger GUI.

C. CAN – Thermometers

In order to give the ability to read the temperature inside the vehicle and compare it with the temperature outside the vehicle a board was designed. This board does not require an additional module since it is supposed to be included in all the distributions of the complete system.

The thermometers used were digital thermometers so that the electromagnetic noise would not corrupt the value of the temperature analogue transmission. The chosen thermometers were DALAS DS18S20 and these are 9 bit digital thermometers which require a custom driver for communication. [9]

The thermometers support multiplexing with other devices of the same types, on the same bus, require a 1-wire data bus and support parasite power (getting power from the data line in IDLE state).

The thermometers require a maximum temperature conversion time of 750 ms counting from the CONVERT TEMPERATURE command. This interval should be respected especially in parasite powered mode.

D. Media Player

A last module was implemented in order to provide multimedia functionality to the application.

The module was fully implemented in HTML and JavaScript just like all the other modules and accessed the ActiveX library of Windows Media Player 11.

Through this ActiveX interface it was possible to implement a fully functional media player to play either video or audio files. Full Screen, shuffle, interactive progress bar, volume control playlists management and other functionality was also implemented in order to allow the driver to easily navigate through his or her media library.

VI. CONCLUSIONS AND IMPROVEMENTS

Conclusions

- The first conclusion is that a microcontroller has a great limitation on the number of port available. The low operating frequency does not allow for efficient serial port emulator on general purpose digital ports. Hence a DSP or small processor should be chosen for enhanced systems.
- Another conclusion is that LIN nodes should be used whenever possible for the lowest possible node cost. Not only they do not require an external oscillator for precise Time Quanta generation, but they also do not need power supply circuitry since the transceiver provides a 5 V, 20 mA power output. This allows the reduction of the entire component list to a single MCU and transceiver for fully functional LIN 2.0 node.
- Regarding CAN, microcontrollers with a built-in CAN driver should be considered for mass production. The available standalone CAN controllers such as the one used MCP2515 require a dedicated crystal and do not have a build-in transceiver thus increasing the total cost of the node, power dissipation and consumption.
- Regarding the communication with the ECU, this approach for the Honda specific OBD1 connector is fast enough for the most demanding applications, reaching a theoretical maximum of 1920 sensor queries per second. This mechanism is fast enough for real time tuning of the injection and ignition advance tables. However, in order to maximize the precision of the table entries, the addition of four new sensors would be required: EGT – exhaust gas temperature for control of the explosion quality and temperature, Knock sensor to detect explosions that occur before the specified time, a wideband O2 sensor for more accurate measurement of the quantity of Oxygen on the exhaust and a atmospheric pressure sensor for correct calculation of intake’s air quantity.

Future Improvements

- The On-board computer should have very low power consumption and a very short recovery time
from suspension mode. A future approach to this type of interface should be implemented with a low power processor, probably an ARM running an operating system such as Windows CE 6.0.

- The software running on the On-board Computer should be designed to meet the new timing requirements of the new protocols. For instance for MOST, which has a large bandwidth reaching up to 25 Mbps, the software should be able to handle each of the streaming data packets. A suggestion for the new language used is C# which is fully supported by Windows CE 6.0 and allows the creating of fast ActiveX components.

- For the Main Board, an interface for two more buses should be added: FlexRay for critical application and MOST for multimedia data streaming. In order to become fully compatible with the actual products on the market, an interface for K-line (ISO 9141) for low speed peripheral communication should also be implemented.

- Concerning the implemented protocols, minor changes on the software should be made in order to correctly handle all versions of CAN and LIN, especially LIN 2.1 which was released while the thesis was being developed. Some improvements on LIN node configuration should also be implemented such as message ID assignment via NAD addressing.

- The last suggestion would be the implementation, within the ECU CAN node, of a NVRAM for real-time programming of ECU MCU code. Since the EA pin of the OKI 66207 is not latched at startup it is possible to keep loading new maps in real time and change the injectors and ignition in real time for optimum engine tuning.

ACKNOWLEDGMENT

I believe that I must mention all the people and companies who made my work possible or at least eased the way as much as they could. Among these there are three special acknowledgements that have to be made to the ones whose contribution was invaluable. These are my parents and grandparents who gave me all the support they could, both financially and emotionally, throughout my academic process; my supervisor Dr. Moisés Piedade who accepted to guide my work and whose passion for electronics has been a key factor on my personal motivation on this course; and at last but not less important than the others, my colleague Paulo Louro who assisted me on every doubts I came across regarding both hardware and software development.

REFERENCES

[9] Dallas - Maxim, DS18S20 High-Precision 1-Wire Digital Thermometer Data Sheet, Feb 21 2003