

Electrical System Development for American Le Mans Racing

Manminder Hayre* Marcelo Martinelli* Yasuhide Sakamoto*

ABSTRACT

The electrical system developed for the American Le Mans Series (ALMS) racing car, used a modular approach for both the engine control unit complete with associated satellite components and the interface and management of 3rd party devices. The basis of the system applied the modular approach by expanding the base engine control system; while still allowing continual development of the engine including different injectors and sensors, with optimized peripheral units for the different engine hardware.

Within the control system a shift to model-based code and simulation for software test and validation was developed and implemented. This resulted in the ability to provide strategies at an average of three times the rate compared with the previous system, and ultimately providing the higher efficiency of overall system application to changing targets.

要 旨

アメリカンルマンシリーズ(ALMS)用レースカーの電気トリカルシステムの開発をおこなった。ECUはソフトウェアのコントロール機能を中心とするようにし、ドライバやユニットは独立したモジュールタイプを採用した。モジュールタイプを採用することによりインジェクタや各センサなどの仕様変更や追加に対し必要なモジュールの対応のみで使うことが可能となり、拡張を容易におこなうことができる。今後、開発のフレキシビリティおよび効率向上を計ることができると考えられる。

ソフトウェアの開発ではシミュレーションやモデルの使用に努めた。その結果、従来に比べ3倍程度時間を短縮してソフトウェアストラテジ投入が可能になり、高効率な電気トリカルシステム開発ができるようになった。

1. Introduction

The designs of previous systems were very engine-specific with algorithms closely coupled to the specific hardware used on the engine. Additionally, the PC tools for the control system were specific products that all users had to be familiar with.

To economize on the resources used to produce and support a new generation of control systems and provide a more efficient development structure allowing multiple applications, a completely new approach was required.

The key points to be addressed with the new system were:

- (1) The main component of the system i.e. Engine Control Unit (ECU), to have a life exceeding five years. Allowing the latest technologies to be applied without this system core being outdated.
- (2) Provide enough expansion for not only engine control but also driveline control.
- (3) Reduce the time from concept to track application for system operation and control algorithm development.
- (4) Develop as close to ECU independent control algorithms as possible.
- (5) The ability to apply the system to different or changing

engine and vehicle specifications without the need for completely new hardware or software.

With the original system assumptions of driveline control and diversity in control applications, the correct steps to fulfilling the requirements for the first race application to be implemented with the Le Mans Prototype 2 (LMP2) project were in place. Figure 1 shows the Acura LMP2 car during initial testing of the new engine and control system.



Fig. 1 Acura LMP2 car

* HONDA PERFORMANCE DEVELOPMENT INC.

2. Engine Control Unit (ECU)

The design of the ECU incorporated very powerful Power PCs, using floating point computation and operating at clock speeds of up to 1.2 GHz, providing digital signal processing (DSP) and parallel processing with the potential of 600+ Million Instructions Per Second (MIPS). This would provide over 10x greater processing than the outgoing control system and the ability to easily implement auto-coding. Added to this was one of the latest Field Programmable Gate Arrays (FPGAs) for Input/Output (I/O) processing and control. To these basics, up to 1GB of memory, multiple 400 Mbit/sec Low Voltage Differential Signaling (LVDS) high speed links and the implementation of four separate Controller Area Network links (CAN) were added allowing the addition of peripheral and 3rd party drive stages or data input units to the control system. In comparison to any previous system used for a motor racing application, this system far surpassed the processing, control speed and expandability. Figure 2 shows the ECU main component design.

3. Injector Driver Unit (IDU)

The distributed approach allowed the development of a custom designed injector driver stage for the LMP2 engine injectors resulting in a LMP2 specific IDU, while provisional engine development was being carried out on the dynamometer with the standard IDU.

Through investigations on the fuel bench, testing different injectors and their responses to different injector drive stages, an evolution of the standard peak and hold design was set. To this design, the extra functionality of injector diagnostics was added. Different injector current and voltage waveforms were processed and a signature for a working or broken injector identified. This signature resulted in using a simple but effective means of providing information on the movement of the injector pintle and hence a means to identify a non-operating injector. To allow firmware up-dates, CAN

communication was incorporated into the design. High Density Integrated Circuit (HDIC) technology was used to address the size and weight reduction. This resulted in a successfully designed, developed and tested injector driver unit.

4. Knock Detection Unit (KDU)

With the use of gasoline for the LMP2 engine, there was a requirement to optimize the ignition to maintain the high compression ratio under all conditions. Also, due to the inconsistency of fuels at different racing tracks and the need to run at the optimum performance for long durations, a knock detection and control system was needed.

As with the IDU development, the distributed architecture of the control system allowed engine development to proceed with the available control system components before the addition of a usable KDU and software could be developed. As with the IDU, the additional goals of reducing component size and weight led to the implementation of HDIC for the KDU as well. To provide reliable working hardware without the need for extensive research and development, the design of the KDU used an existing Knock IC. This device processed one of four broadband analogue accelerometer inputs switched by a multiplexer. Figure 3 shows the architecture of the system with the Knock IC interfacing to the ECU via an FPGA.

To select the correct frequency and optimize the detection, high bandwidth knock sensor data and cylinder pressure data were gathered. The high sampling rate data was post processed using an in-house developed numerical computing model of the hardware to yield a target for the optimum filter pass frequencies and window angles. The upper filter frequency was dictated by the operating range of the sensor. The lower frequency was to encompass the 1st order frequency of knock without dropping into a high area of noise seen with lower frequencies. The band pass filter ranges where a result of an iterative process of data analysis.

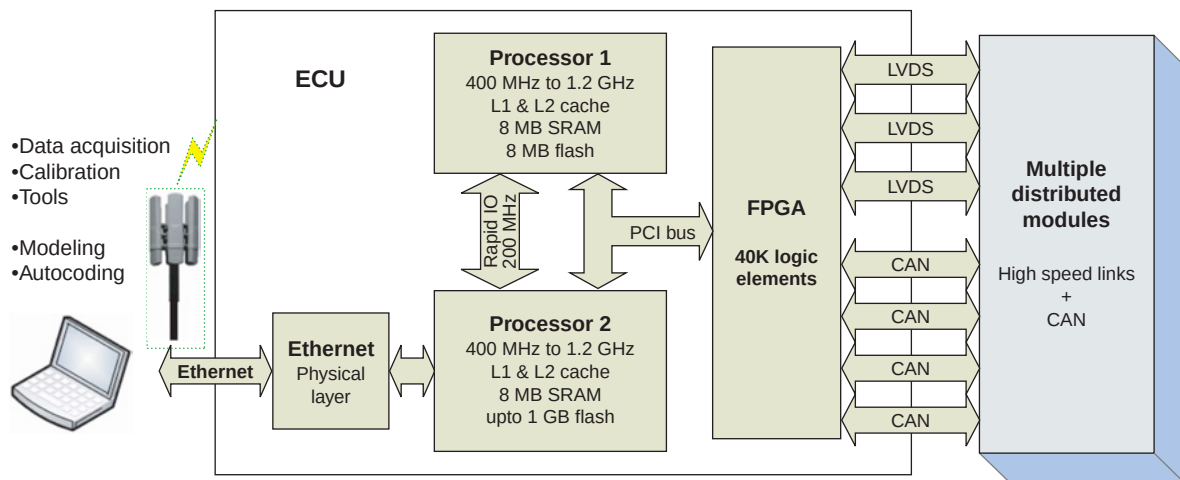


Fig. 2 ECU architecture

Figure 4 shows a plot of the minimum detectable cylinder pressure oscillations against engine speed for different data filtering ranges to determine the most suitable filter type and range. The three lines plot, Plot 1. the best signal to noise ratio with a high pass filter of >20 kHz. Plot 2. the chosen band-pass filter of 12 KHz to 20 kHz and Plot 3. the worst with a band-pass 6 kHz to 17 kHz filter. The optimum filter and frequency range, Plot 2, was chosen because it minimized the detectable cylinder pressure oscillations while also remaining within the knock sensors' operating range yielding a clear detectable knock event.

The development of the algorithm was split into an ignition open loop and a knock feedback numerical computing model both developed using a multi-domain simulation and model based design system platform. The Ignition model provided a baseline knock threshold ignition timing and integrated with the other ignition algorithms affecting the engine torque. The feedback model used two knock thresholds. One threshold caused an immediate retard with the other using a statistical means for control. The knock calibration continually evolved with data available from dynamometer and subsequent track testing. This led to the knock signal noise and probability of knock being determined through statistical analysis leading to a further evolution of the system.

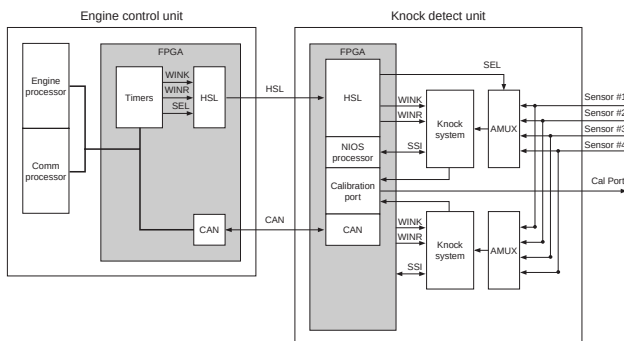


Fig. 3 KDU embedded architecture

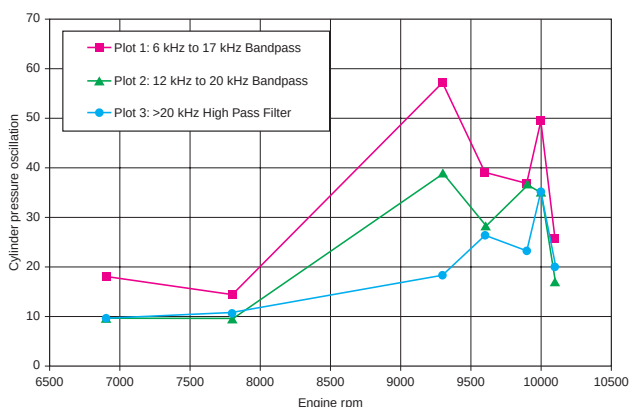


Fig. 4 Filter performance for knock data

5. System Communication Architecture

The distributed nature of the engine control system along with all 3rd party chassis electronic components required a unique communications and control architecture unlike the standard interfacing scheme previously used.

A configurable CAN interface was developed in order to meet the changing requirements for data sharing between engine ECU, chassis logger and telemetry radio. Figure 5 shows the System communication architecture. CAN buses were segregated into control and monitoring classifications in order to make the overall system more robust and fault-tolerant in the event of a link failure or link overload condition. CAN 4, blue lines, denotes monitoring only link. CAN 1, red line, denotes system control messages link. CAN 2 and 3, denotes 3rd party links with CAN 3 used for specifically for timing critical information. The use of CAN for driver/dash switches as well as expansion I/O, was a means to reduce overall wiring weight and still maintain the ability to attach additional sensors for testing and development purposes during the course of the season. Message transmission rates between the different buses varied from 10 Hz to 1 kHz.

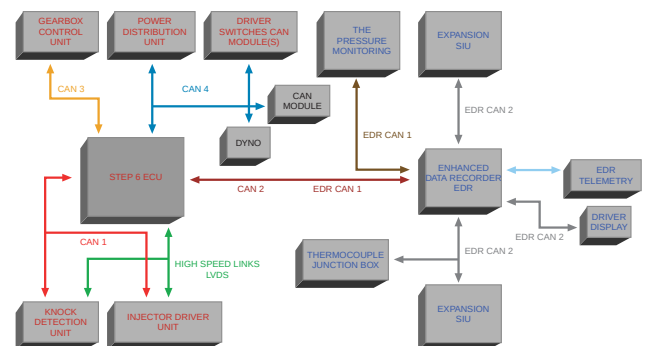


Fig. 5 System communication architecture

6. Embedded Software Structure

The embedded code was split into three layers to provide abstraction specifically for auto-coding, which was new for this system:

- (1) HAL – Hardware Abstraction Layer
 - (a) Lowest layer of software. Defines hardware drivers, input/output conditioning, interface to memory, register addresses, communications etc.
- (2) VAL – Vehicle Abstraction Layer
 - (a) Middle layer of software. Converts low level signals into engineering units.
 - (b) Characterizes sensors like wheels speed, crank and cam, temperature, pressure, knock, etc. Converts calibrated injection end angle into a start angle.
- (3) APP – Application Layer
 - (a) Highest layer of software. Contains all high level

functionality. Fuel and ignition control, pit lane speed limiter, yellow mode, idle control, shift control, knock control, engine torque control, etc.

(b) This layer is auto-coded.

A Real Time Operating System (RTOS) was implemented for one of the processors to handle higher level controls, such as the Ethernet communications and other items. The main processor, however, required the development of bespoke drivers which provided the basics of the Hardware Application Layer (HAL). The result of this software structure together with the hardware design and firmware management was the application of a structured software code base. Figure 6 shows the flow of information organized in the different software layers. e.g. the decision on the injector angle is made in the APP layer. This decision interfaces with the VAL layer which translates the angle to a value associated with the calculation parameters i.e. the start angle. This is then translated to an actual time or/and event controlled by the ECU hardware in the HAL layer. Similarly, the raw inputs are read by the hardware specific devices. i.e. analogue to digital converters at the HAL layer. These are then translated to pulses or volts, i.e. engineering units at the VAL layer then used in the algorithms for temperature, pressures or engine speed at the APP layer.

As part of the software structure direction, the goal of reducing the time from concept to application code and hardware independency was realized by implementing 100% of the ECU embedded code application layer in model format.

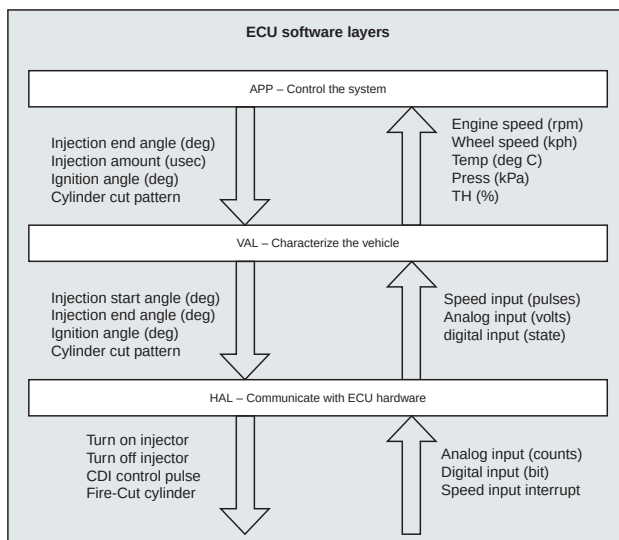


Fig. 6 Embedded software layers

7. Code Testing

To complete the loop for coding efficiency a custom configured Commercial Test Hardware and associated Open Source Software (CTHOSS) system was utilized to expand our testing capabilities. In conjunction with

this and to compliment the ECU models we also had to develop plant models for testing and simulation. For the engine model, we started with a basic model of a V8 engine. To this, enhancements were made and then a transmission, drive shafts, uprights and basic tire model where added. The platform for the testing and implementation of the model-based environment was an industry standard numerical computing tool, multi-domain simulation and model based design platform in conjunction with the CTHOSS. The CTHOSS interface fitted in with our entire simulation and auto-coding environment. The same plant models where used offline as well as in hardware in the loop simulation. The two methods in which we used the system, connecting directly to an ECU were:

- (1) Hardware in the Loop Simulation (HILS) - CTHOSS houses a plant model of engine and vehicle. ECU operates as if driving a real car.
- (2) Bench tester – CTHOSS houses a model that maps buttons and dials, displayed on the screen of a PC, to ECU I/O.
 - (a) User can manually exercise functionality, test hardware and check communications between different hardware modules.
 - (b) User can run track accumulated or dynamometer data through the ECU.
 - (c) Automated tests are run through the ECU to exercise existing functionality.
 - (d) Representative signals had to be created for ECU excitation. i.e. variable reluctance cam and crank, wheel speed, knock, etc.

With the above approach, all the hardware that made up the vehicle electrical system could be connected together for bench testing prior to dynamometer or track running. i.e.

- (1) ECU, gearshift controller, knock detection, injector driver, CDI, etc.
- (2) Real system loads like injectors, spark plugs, shift actuator, all sensors, etc.

8. Conclusion

The development of the system provided new directions in many areas. On the controller side, the ECU used two power PCs running at speeds that are more associated with desktops. The implementation of the HDIC technology and the use of a multiplexed knock IC for the knock allowed the use of new technology with existing components but in a different environment. The use of injector movement signatures for the diagnostics again was new for the LMP project. The handling of the different component communications utilizing CAN at a much greater level than previously used, provided the ability to quickly add 3rd party components and expand the complete vehicle system but still provide diagnostic abilities through component fault interrogation and hence isolation. Figure 7 shows the extent of the electronic components required for an LMP car. On the software side, the implementation of model-based code and auto-coding the application layer was the basis of

the new development environment and the goal of concept to code time reduction resulted in the release of a strategy at three times the rate as in the previous system approach. This was achieved by using the test hardware and having the ability to simulate the track environment better. Coupled with this was the development of the plant models for the engine driveline and system expansion, which provided more accurate simulations and hence the ability to provide a calibrated system for different circuits more feasible.



Fig. 7 LMP2 vehicle cockpit layout

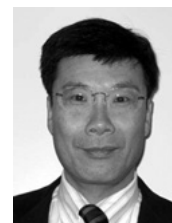
■ 著 者 ■



Manminder Hayre



Marcelo Martinelli



Yasuhide Sakamoto