THE EVOLUTION OF POWERTRAIN MICROCONTROLLERS AND ITS IMPACT ON DEVELOPMENT PROCESSES AND TOOLS

Gary Miller - Motorola
Kevin Hall - Hewlett Packard
Wayne Willis - Hewlett Packard
Wilfried Pless - Hewlett Packard

ABSTRACT

As the new generation of RISC powertrain MCUs propagate through the automotive development cycle, there will likely be more difficulty in debugging the ECU reliably and efficiently. Simply stated, there is less support for the development process in the new high-performance single-chip RISC MCUs, which could create critical and costly delays in the development cycle. Additionally, as powertrain MCUs continue to evolve, superscalar or multiple-issue RISC implementations may be used as the central processor. With the capability to issue multiple instructions in one clock cycle, this will only magnify the development support problem. Thus it is essential to address this impending problem with a strategy that both automotive and tools developers can agree. A strategy for development standards is presented in this paper, and a new powertrain MCU development interface standard is proposed.

Evolution of Powertrain Microcontrollers

With the advent of the new 32-bit powertrain MCUs driven by high performance, integration and low cost, there are unique challenges and trade-offs for both SW/HW development engineers and development tools vendors. Ever increasing performance needs have led to the migration to RISC processors with on-chip instruction and data caches, or high-speed RAM and Flash, to meet the high memory bandwidth requirements of these architectures. High frequency one clock cycle instruction and data buses are on-chip to support the memory bandwidth requirements to the internal memory system.

The challenges offered with the emergence of the new 32-bit MCUs are to provide the visibility needed for logic analyzer, processor emulation and calibration systems, while not compromising performance and cost saving features. The new MCUs solve this problem by providing trade-offs to the SW/HW development engineers. Development engineers can choose from a number of options which provide an increasing level of visibility, but with an impact of a reduction in performance and features. Consequently there is no standard approach used since each is customized to the user's own preferences and needs. This presents a compatibility challenge to development tools vendors.

Impact on SW/HW Development

The integration of high-speed RAM and flash onto the powertrain MCU creates a paradigm shift for processor run control and logic analysis tools. With high-speed memory on-chip, memory substitution techniques may no longer be truly transparent due to the significant latency increase in accessing off-chip memory relative to on-chip memory. Thus comprehensive emulator techniques become problematic for doing run control operations without impacting the customer's system e.g. reading and writing memory and internal registers, and halting and running the microprocessor. Additionally, since accesses to internal memory do not, by default, show up on external pins, there is insufficient visibility for logic analysis.

With integrated memory, there is an expectation by automotive developers that some applications will not require the external address and data signals. For these applications, a cost-saving technique is to use these pins for general-purpose I/O. Run control and logic analysis issues are further complicated by removal of the external address and data signals altogether. As a result of the powertrain MCU evolution, tools vendors must re-approach their solution technology in new and cooperative ways with the silicon vendors.

Impact on Calibration

ECU calibration comprises data acquisition of intermediate calculated variables, co-processing or analysis of critical parametrics, and tuning of calibration constants either on a dynamometer or in-vehicle. In today's powertrain MCUs, where RAM and flash are easily accessible, the calibration process is essentially non-obtrusive. But with high-speed on-chip memory and the removal of the external address and data signals altogether, calibration issues may impact both the SW/HW developer and the calibrator. Depending upon the method used for acquiring variables and updating constants, there can be a range of impacts on MCU overhead. The developer must trade-off options which provide an increasing level of acquisition and tuning capability, but with an impact of a reduction in performance and features.

DEVELOPMENT NEEDS FOR POWERTRAIN APPLICATIONS

Figure 1 illustrates at a high level view the automotive development process steps for a mechatronic-based system. This diagram identifies the 3 major sub-systems being developed and the associated steps encountered in each sub-system's development. The diagram is also useful in demonstrating the expansiveness and complexity of the automotive development process, and how different skills and organizations are involved. Requirements are initially extracted from the vehicle level to powertrain needs, then to electronic hardware, and finally to module software. As the design sub-systems come together into prototypes, they are then qualified and calibrated to verify conformance with the initial requirements.

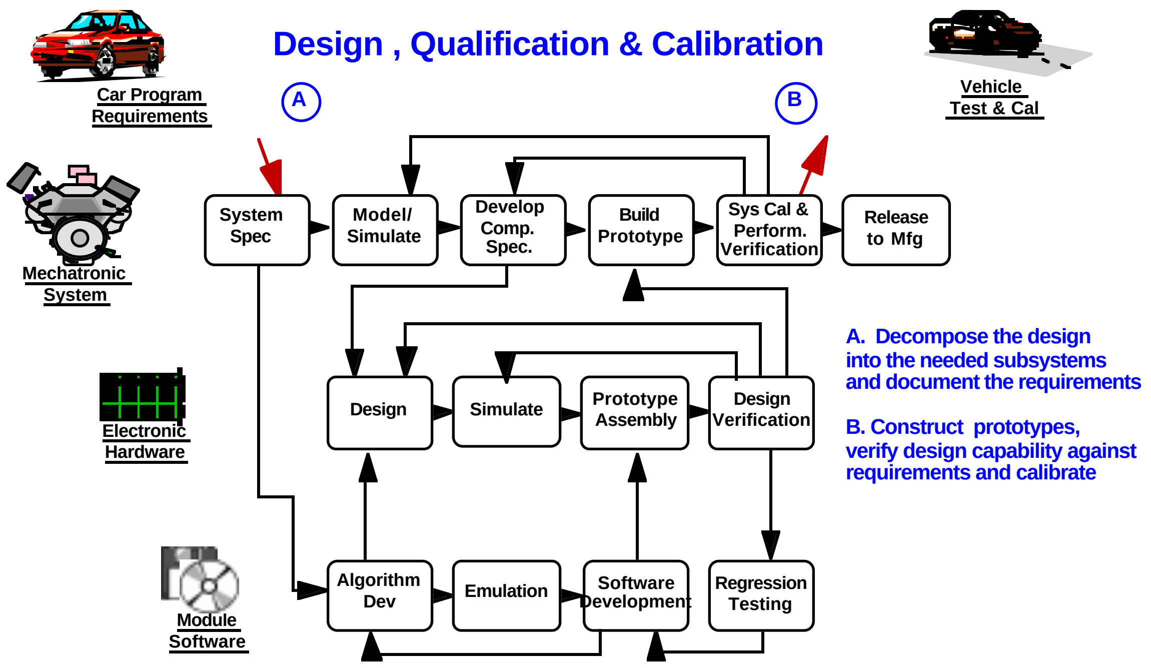


Figure 1: Mechatronic System Development Lifecycle

Figure 2 illustrates the timeline and steps associated with a typical powertrain program. Across the program timeline the design content evolves to become more complete as the developers and tools change.

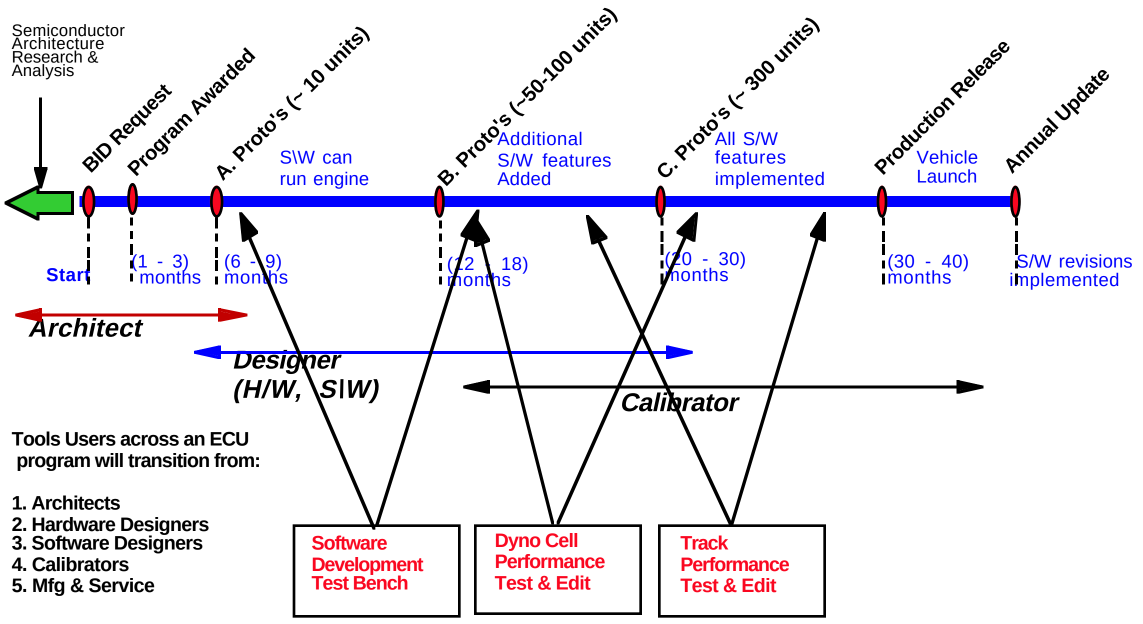


Figure 2: Program Timeline

Basic Development Needs

Automotive developers have certain needs of their development tools in order to accomplish their jobs. For *logic analysis* the basic needs are:

1. To access program trace information with acceptable impact *to* the system under test. The developer needs to be able to interrogate and correlate program flow to real world interactions
2. To retrieve information on how data flows through the system with acceptable impact to the system under test, and to understand what system resource(s) are creating and accessing data
3. To assess whether system SW is meeting the required performance with acceptable impact to the system under test

For *run control* the basic needs are:

1. To query and modify all register and memory locations
2. To support breakpoint features in debuggers, either as HW or SW breakpoints depending on the architecture

Note that the capability for run control, logic analysis and memory overlay or substitution comprise what is traditionally known as processor or in-circuit emulation.

For *calibration* the basic needs are:

1. To be able to positionally (crank shaft synchronous) acquire data relating to calibration factors as they are being used or modified during high speed transient events with acceptable impact to the system under test.
2. To acquire time synchronous data relating to calibration factors with minimal impact to the system under test
3. To coherently modify table(s) of calibration data in real time while the ECU is running the vehicle

For *algorithm development* the basic need is to rapidly prototype new ECU algorithms and perform conformance testing in a vehicle e.g. new fuel strategy such as direct injection. The new algorithm is typically implemented in a special emulation module which is connected to the ECU and acts as a co-processor. Control strategy SW is modified to make a procedure call to the special emulation module at the appropriate time, and after waiting for result calculations, the SW proceeds with normal control strategy.

Table 1 describes the ECU development phases and further demarcates the development needs and tools which are used in each phase.

HW Development

- Bring up board for first time
- Debug HW interfaces (e.g. RAM, serial, A/D, etc.)
- Using bench code stimulate interface to test timing, noise margins, etc.
- *Logic analyzer* central to development effort, including timing, scope and stimulus functions
- Prototype module developed only for bench environment to be used with bench-top engine emulator
- May implement low-level drivers interfacing to HW

SW Development (may be provided outside company)

- Starts with requirement specifications developed by separate algorithm development team (may even be provided by an OEM)
- Majority of work is accomplished on bench-top with prototype ECU and bench-top engine emulator
- More focus on *BDM debuggers* and less on logic analyzers and parametrics
- May start development with high level HW modules or VHDL simulators
- Start using calibration tools as a high level view of how SW is responding to stimulus (to view calibration parameter usage)
- Some routines may be debugged on dynamometer
- Initial calibration constants supplied from historical information

HW/SW Integration (measurement intrusion becomes a more significant factor)

- Resolve problems/bugs related to inadequate specifications
- Start to measure performance and other ECU parametrics
- Start to exercise asynchronous events and resolve race conditions
- Test corner cases in real time
- May use PRUs or PSEs to aid visibility (depending upon development philosophy)
- Need higher dynamic range of abstraction from tools, from capturing HW events, correlating events to program/data flow, to statistical analysis for SW routines

Calibration

- Tuning 5 to 15 thousand constants which impact drivability
- 3 types of calibration
  - Initial constants supplied in SW development phase
  - Dynamometer calibration for dynamic tuning of engine
  - In-vehicle for final calibration

Dynamometer

- Approximately 30-50% of constants are calibrated with dynamometer (i.e. ignition control – misfire, knock, torque, etc.)
- May use PRUs to aid visibility (depending upon development philosophy)
- Interface should be non-intrusive and meet synchronized data bandwidth requirements (i.e. 40 bytes/1msec and more in future). Block transferred out each synchronized event and transferred in periodically. ASAP standard exists for dynamometer calibration.
- Common technique for updating constants is external to MCU memory emulation (position synchronized bank switched).

In-Vehicle

- Approximately 50-70% of constants are determined in in-vehicle calibration (i.e. cold/hot start, altitude compensation, etc.)
- Performed with production-intent module
- Interface may be intrusive and should meet time and vehicle correlated data bandwidth requirements (i.e. 40 bytes/10 msec and more in future). Block transferred in/out periodically
- Common technique for updating constants is external to MCU memory emulation. Serial interface also used.

Table 1: Electronic Development Phases and Development Needs and Tools

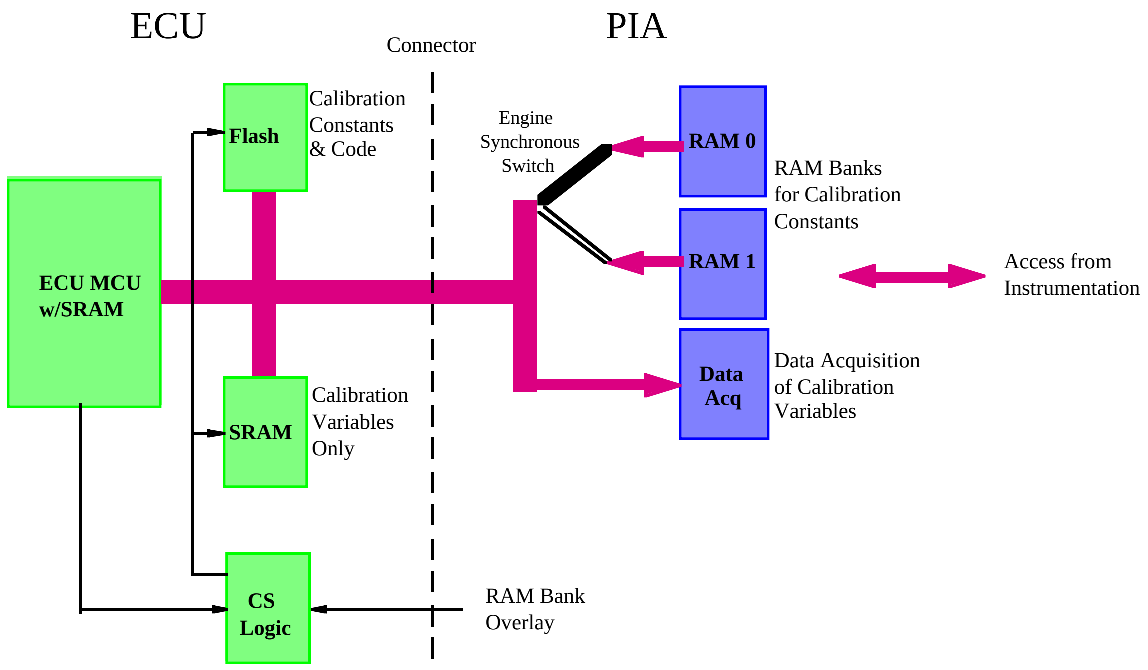
It should be noted that there are different philosophies with regard to ECU development for powertrain MCUs comprising fast on-chip instruction and data memories. One approach is to provide more MCU visibility to enhance the development process by using a Port Replacement Units (PRU) in module versions up through in-vehicle calibration. This adds risk since the PRU may be removed in the production module, which may impact ECU parametrics. Another approach is to transition to the production-intent module as early as possible in the development process. This approach provides lower risk but with potentially more difficulty during the development process. There are also varying approaches in between these two which trade-off the development process and risk.

Calibration Performed with External to MCU Memory Emulation

The predominant technique used today to perform calibration on the dynamometer and in-vehicle is with external to MCU memory emulation. A popular solution is Hewlett Packard's Parallel Interface Adapter (PIA) as shown in Figure 3. On the left side of Figure 3 you see the relevant pieces of the ECU, and on the right side the main building blocks of the PIA are shown.

When calibration is performed visibility is needed of intermediate calculated values (calibration variables) such as calculated ignition time, actual RPM, sensor temperatures, knock sensor value, etc., Many of these variables are calculated every cycle of each cylinder. The PIA provides instrumentation access to these variables through data acquisition hardware. By snooping the MCU's external bus for writes to calibration variables in the ECU SRAM, data acquisition hardware provides an image of the calibration variables for instrumentation access.

After parametric analysis is performed the two RAM banks of the PIA allow instrumentation to update calibration constants for ECU access. Chip select logic on the ECU is used to overlay flash memory containing calibration constants with new constants in RAM bank 0/1. So instead of reading calibration constants from the flash, the MCU reads them from one of the RAM banks. Which one of the banks calibration constants are read from is determined by a switch triggered by engine TDC occurrence. The position synchronous switch allows for coherent changes to calibration constants.



ECU
Connector
PIA
Flash
Calibration Constants & Code
Engine Synchronous Switch
RAM 0
RAM Banks for Calibration Constants
RAM 1
ECU MCU w/SRAM
Access from Instrumentation
Data Acq
Data Acquisition of Calibration Variables
SRAM
Calibration Variables Only
CS Logic
RAM Bank Overlay

Figure 3: Calibration Example using a PIA

THE EVOLUTION OF POWERTRAIN MCUs

Since the first application of semiconductors in engine control the powertrain MCU has continuously evolved. As requirements have exploded due to more intensive control strategies and environmental restrictions, the powertrain MCU has evolved into a very high-performance and cost-effective solution. This evolution has been good for automotive manufacturers and their customers, but a result has been the creation of a new paradigm which will negatively impact development processes and tools. In the remainder of this paper the impact on development processes and tools will be investigated. Additionally, options for new powertrain MCUs will be discussed to simplify development processes and tools.

Today's ECUs

Powertrain MCUs used today are predominantly Complex Instruction Set Computers (CISC). An example architecture is Motorola's CPU32-based MCU as shown in Figure 4. CISC MCUs do not require a high bandwidth instruction path since typical instructions require multiple clocks to execute. Furthermore a unified bus architecture is sufficient in many applications. Thus delays for off-chip accesses and visibility cycles do not significantly impact overall MCU performance.

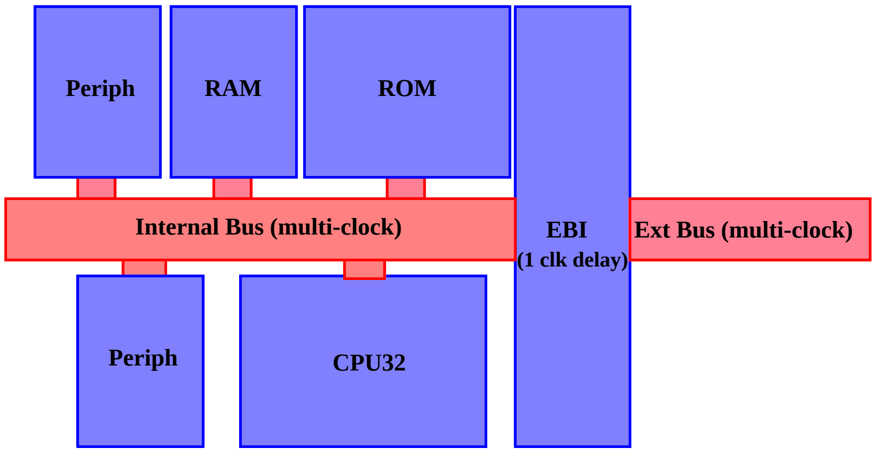


Figure 4: Motorola's CPU32-Based MCU

Figure 5 shows typical development tool usage for this generation of MCUs. Full program/data flow visibility is readily available with no intrusion, although a PRU may be required for single-chip MCUs. Calibration may be accomplished with external to MCU memory emulation with little or no intrusion. SW development is supported through the Background Debug port.

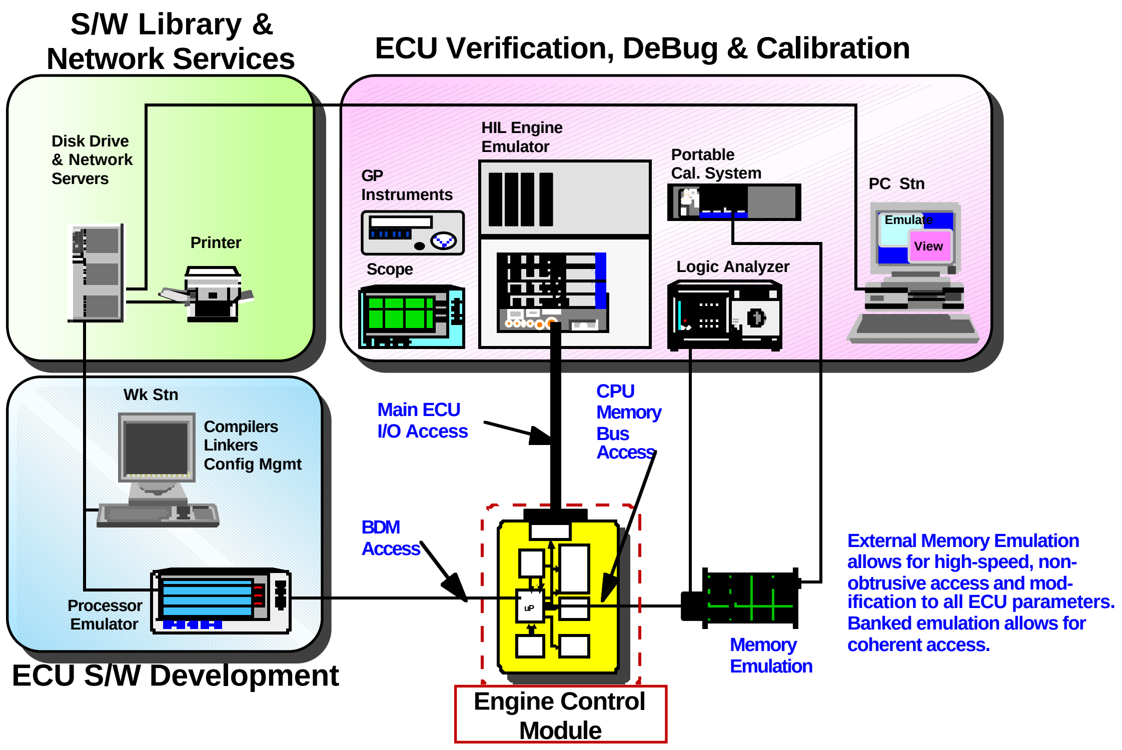


Figure 5: ECU Development Bench for Today's MCUs

Some recent CISC implementations as shown in figure 6 have evolved to include a high bandwidth data path to improve performance. This evolution signals a trend in engine control MCUs towards more performance with the addition of embedded, single clock buses. *It also represents a paradigm shift in the development model since there is less external bus visibility of internal data or instruction flow information, or a reduction in performance if flow information is shown in real time on the external bus.* A reduction in performance results when transactions on the fast embedded bus are prohibited and re-directed to the internal bus for the purpose of increased visibility on external bus pins. This increases bus traffic on the internal/external bus and slows performance. An alternate approach is to pin out the embedded bus for data visibility.

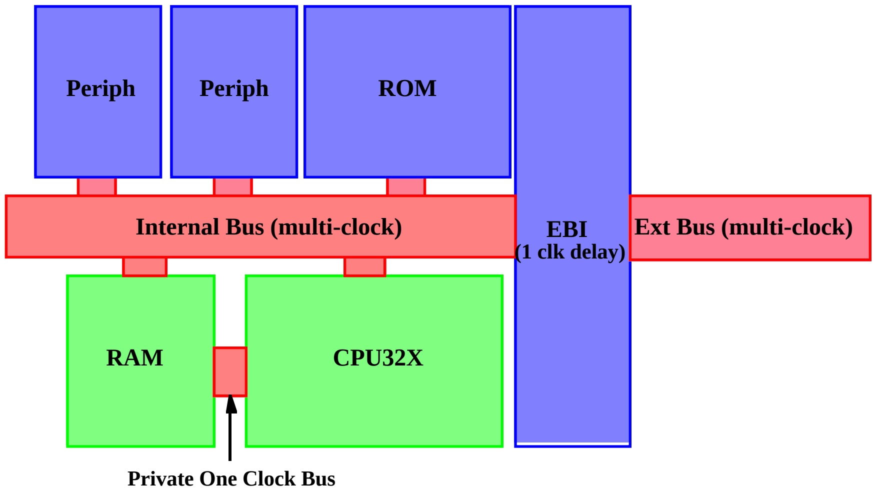


Figure 6: Motorola's CPU32x-Based MCUs

Migration to RISC

Ever increasing performance needs for automotive engine control applications have precipitated a migration to Reduced Instruction Set Computers (RISC). RISC architectures by nature require high bandwidth data and instruction paths to ensure high performance, with most instructions being executed in one clock cycle. With the advent of RISC architectures into engine control there are performance, features and development trade-offs to understand and exercise wisely.

RISC processors achieve high performance by utilizing multiple independent execution units which may be executing instructions in parallel. A scoreboarding technique may be used to determine instruction data dependencies. Instructions may even complete out of sequence if there are no data dependencies. RISC implementations with a Harvard architecture comprise separate instruction and data buses. Autonomous instruction fetch and data load/store execution units are employed to maintain the high instruction/data bandwidth. The result is that multiple instructions may complete or retire on a single clock cycle. Additionally instruction and data buses require high-bandwidth (1 clock cycle access), which may result in a performance or visibility "bottle neck" through the external bus interface. Thus development support for program trace is more complicated since not all program/data flow is typically echoed on the external bus.

To meet instruction fetch demands of RISC architectures an on-chip instruction cache (1 clock access) is typically utilized, with burst fetches to the off-chip instruction memory. And to meet data access demands of RISC architectures fast on-chip RAM (1 clock access) is typically utilized. Figure 7 illustrates an example of an internal Harvard/external unified architecture as implemented on Motorola's MPC509 PowerPC MCU. This architecture allows for fast on-chip access to data/instructions to achieve high performance.

For this and similar MCUs automotive system developers are required to make development trade-offs which impact performance. Enabling instruction show cycles (show all cache hits, show only changes of flow, or show only indirect changes of flow) provide visibility of cache accesses needed for program trace, but at the cost of more bus traffic and lower performance. Processor state information is provided externally each clock to indicate if the current instruction is sequential, branch direct taken, branch indirect taken, exception taken, or branch not taken. Information is also provided on how many instructions complete each clock. With this state information and visibility of branch addresses, referred to as *branch trace messaging*, the program flow can be reconstructed by development tools in a post-analysis manner.

Enabling data show cycles (show all internal accesses or show only internal writes) provide data flow information but with lower performance. It should be noted that if show all is enabled for both data and instructions then the MPC509 operates in the performance class of its CISC predecessors.

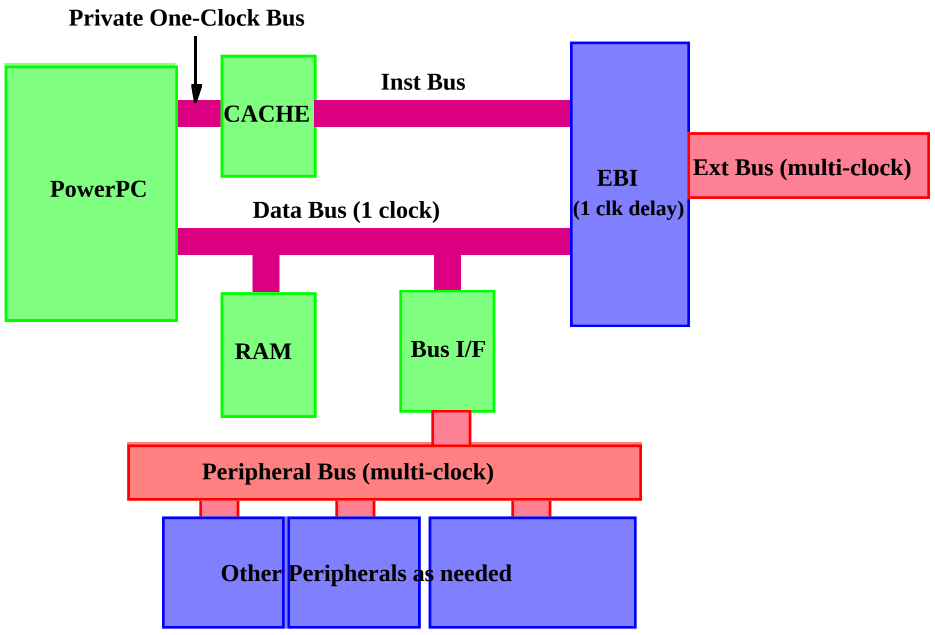


Figure 7: Motorola's MPC509 PowerPC MCU

With fast on-chip data memory developers also trade-off visibility needed for data acquisition of calibration variables against performance. Visibility is gained by data write showcycles, also called *data write messaging*, or data "echo" cycles on the external bus. External data write showcycles reduce external bus bandwidth available and typically needed for instruction fetches, and can also slow the internal data bus.

New Trade-offs for Tomorrow's ECUs

More recently fast instruction memories have also been integrated on RISC MCUs for the improved performance/integration/cost of a single-chip solution. Refer to figure 8 for an example of Motorola's MPC555 PowerPC architecture. The challenges offered with the emergence of the new 32-bit single-chip RISC MCUs comprise providing the visibility needed for calibration, logic analyzer and processor emulation while not compromising performance and cost saving features. The MPC555 also addresses this problem by providing trade-offs to the development engineers. Development engineers can choose from a number of options which provide an increasing level of visibility, but with a reduction in performance and features.

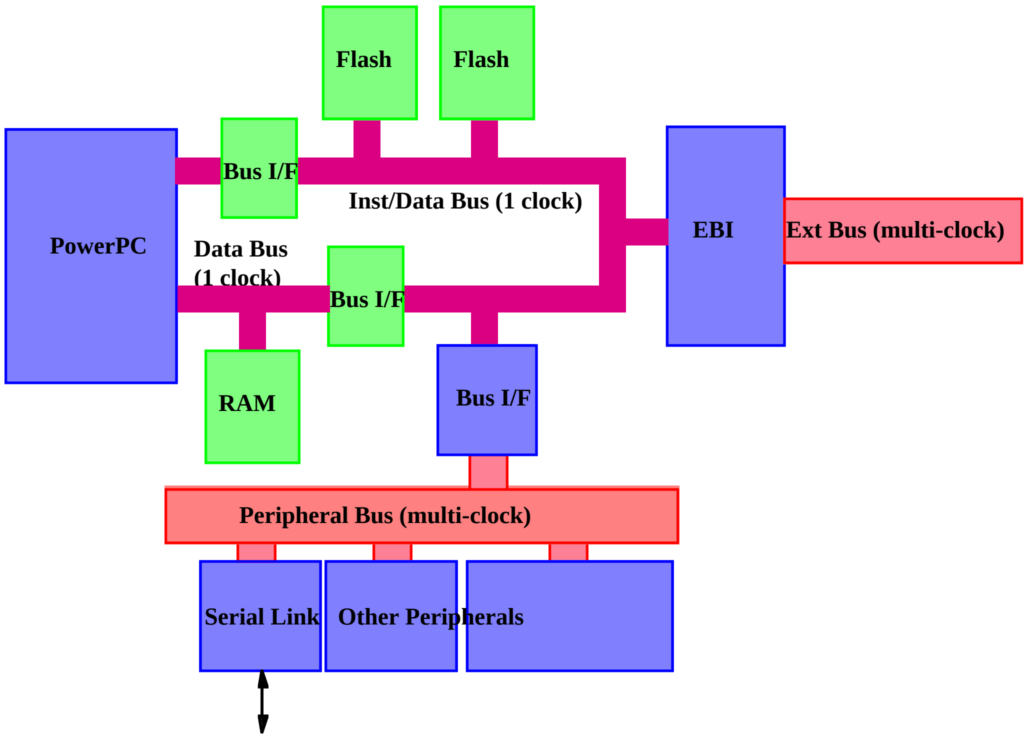


Figure 8: Motorola's MPC555 PowerPC MCU

To provide visibility for logic analyzer support the MPC555 device should be configured in one of the expanded modes offered. The Addr/Data/GPIO pins can be configured as GPIO (single-chip mode), as Addr and GPIO (allows instruction showcycles for program trace) or as Addr and Data (for complete visibility). An external PRU can be designed to provide needed GPIO if an expanded mode is used. Although these options provide a range of solutions which may meet needed development and functional capabilities, expanded mode options may only be acceptable to some powertrain developers for preliminary development stages. Additionally, there would be a varying degree of performance degradation, depending upon which expanded mode configuration and showcycle configuration is selected. Showcycle options and processor state visibility information offered on the MPC555 are similar to the MPC509.

To perform calibration several options are available on the MPC555, depending upon if the device is configured in an expanded mode or in single chip mode. Calibration performed with the device in an expanded mode configuration may be accomplished through external to MCU memory emulation. This can be accomplished with an ECU configuration similar to Figure 3, with the addition of a PRU and the exception that on-board flash would not be required. By using a feature of the MPC555 to overlay data accesses within a programmable region in the internal flash, these accesses can be mapped directly to an external RAM bank. There may only be minimal performance impact for accessing constants externally as the on-board flash is optimized for fast instruction fetches, but requires multiple clocks for data accesses.

Calibration performed with the device in the single-chip configuration may be accomplished through an on-chip serial link such as a CAN link. Due to data acquisition requirements for calibration on a dynamometer it may only be practical to perform in-vehicle calibration through the serial port. Even so, the processor overhead required for managing the data transfer through the serial link should be considered when evaluating this option, as it may be too intrusive. The CAN link on the device has 16 transmit/receive message buffers of 8 bytes, for a total of 128 transmit/receive bytes which can be queued. This feature allows for a minimal number of interrupts to support data acquisition and calibration.

Table 2 summarizes some of the development trade-offs for the MPC555 and characterizes performance impact and related issues.

	Addr/Data/ GPIO Usage	Performance Impact	Processor State Info	Comments
Program Flow	Addr & GPIO	Performance impact estimated in the 0% to 5% range (code dependent) for branch trace messaging	Latched each clock by tools	Post processing required by tools; PRU may be required; Performance impact may not be acceptable to some developers
Program and Data Flow	Addr & Data	Performance impact estimated to be larger than 5% (code dependent) for branch trace and data write messaging	Latched each clock by tools	Post processing required by tools; PRU may be required; Performance impact may not be acceptable to some developers
Calibration Option #1 - Parallel Interface and external to MCU memory emulation	Addr & Data	Minimal performance impact for accessing RAM banks with 0 wait state external memory; additional performance impact dependent on number of cal variables being acquired	N/A	Dynamometer or in-vehicle calibration; PRU may be required; Performance impact may not be acceptable to some developers
Calibration Option #2 - Serial Interface managed by CPU	GPIO	Performance impact primarily dependent on interrupt rate required for instrumentation	N/A	In-vehicle calibration; Performance impact may not be acceptable to some developers; Requires cal constant pointers in RAM to re-map desired constants from Flash to RAM

Table 2: Development Trade-offs for the MPC555

Figure 9 shows possible development tool usage for the MPC555. In comparison to the current development tools use model, as shown in Figure 5, full program/data flow visibility will result in intrusion. The development tools will be more complicated since post analysis is necessary. Calibration may be accomplished either by external to MCU memory emulation, or possibly via a serial link. SW development is supported through a Background Debug port, although port definition has evolved.

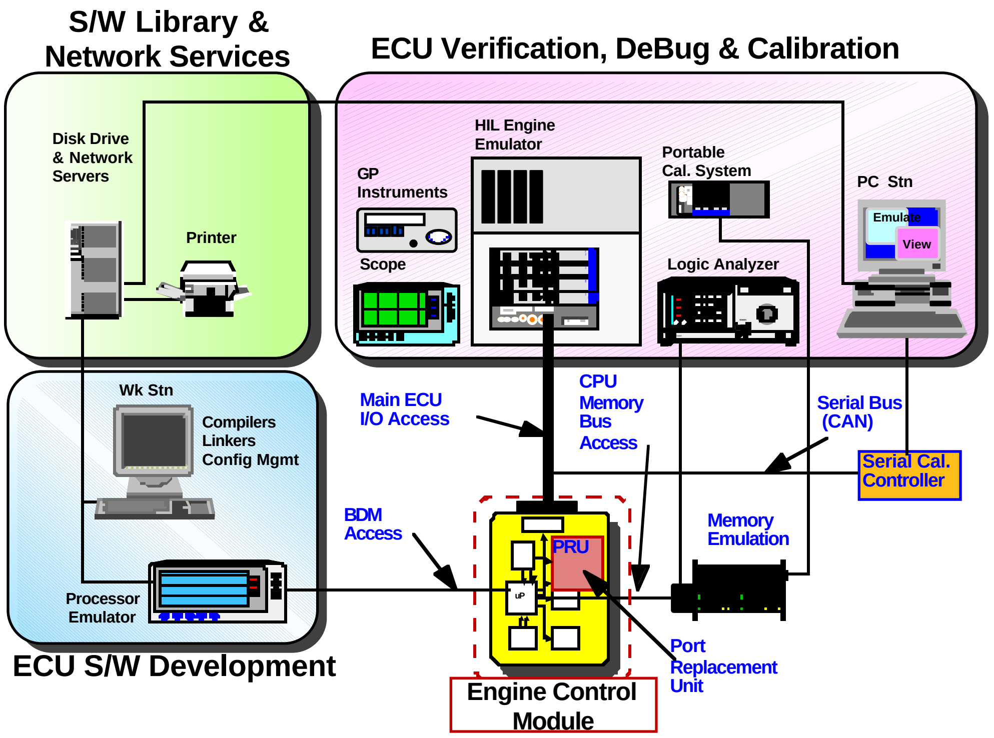


Figure 8: ECU Development Bench for the MPC555

ADDRESSING DEVELOPMENT PROBLEMS FOR TOMORROW'S POWERTRAIN MCUs

As the new generation of RISC powertrain MCUs propagate through the automotive development cycle (which can take several years) there will likely be more complexity and difficulty in debugging the ECU reliably and efficiently. Simply stated, there is less support for the development process in the new high-performance single-chip RISC MCUs, which could create critical and costly delays in the development cycle. Additionally, as powertrain MCUs continue to evolve, superscalar or multiple-issue RISC implementations may be used as the central processor. With the capability to issue multiple instructions in one clock cycle, this will only magnify the development support problem. Thus it is essential to address this impending problem with a strategy with which both automotive and tools developers can agree.

Recommendations for Tomorrow's ECUs

A Real-time Embedded Application Development Interface (READI) standard has been proposed by Motorola. The Nexus Consortium, comprising world-wide leaders in semiconductor ICs and development tools, has adopted and enhanced the READI standard. Key milestones regarding the Nexus standard are

- Alpha release on 16 June 98
- Beta release on 16 October 98
- Nexus Consortium announcement at Convergence '98 on 19-21 October 98
- 1.0 release on 18 December 98

The proposed standard can be down-loaded from the website www.nexus-standard.org. The intent for the standard is to improve the way embedded systems (MCUs, MPUs and DSPs) are developed. Automotive-specific needs, such as calibration, are covered in the development interface standard.

The proposed standard is JTAG-based (IEEE 1149.1 standard). It defines a standard development use for the 5 JTAG pins, which are already available on the majority of embedded processors. JTAG pins are used to access standardized development registers and for read/write access to internal memory-mapped resources during runtime. The standard also defines an auxiliary port (2 or more pins), which provides for high-speed, packet-based messaging. The auxiliary port is needed for run-time visibility and controllability such as program trace, data trace, memory substitution, read/write access to internal memory-mapped, and other high-bandwidth information transfers (vendor defined).

Table 3 summarizes the major problems which were discussed in this paper. It also lists the recommendations for solving these major problems using the Nexus standard.

Problem/Task	**Method(s) used for Today's MCUs**	**Method(s) used for RISC MCUs**	**Nexus Standard**
Use Model: HW Dev. and HW/SW Int. High performance on-chip instruction cache or flash eliminates visibility needed for program trace. Need program trace visibility with acceptable impact to the system under test.	CPU32-based MCUs use configurable showcycles to provide visibility of fetches to the outside world with no performance impact. Requires that external address and data pins are available.	Customer may trade off performance for visibility. As more access types are configured for showcycles performance can degrade accordingly for some architectures. Branch trace messaging is the minimal configuration needed for visibility and may result in a performance impact in the 0% to 5% range (code dependent). Requires that external address pins are available.	Defines an auxiliary port for outputting branch trace and data write messaging at high speeds. The port is capable of meeting bandwidth requirements to support messaging with no performance impact. Bandwidth requirements can be met via an expandable port size or clock rate. Implementations may include a message queue, which eliminates the need to stall the processor for most scenarios.
Use Model: HW/SW Int. High performance on-chip SRAM eliminates visibility needed for data trace. There are 2 types of data visibility needed: (1) Which process wrote which data parameter and what new value was written? (2) For a chosen data parameter which process accessed it? Need data trace visibility with acceptable impact to the system under test.	CPU32-based MCUs use configurable showcycles to provide visibility of data accesses to the outside world with no performance impact. Requires that external address and data pins are available.	Customer may trade off performance for visibility. Branch trace and data write messaging is the minimal configuration needed for data write visibility (#1) and may result in a performance impact of larger than 5% (code dependent). Branch trace messaging and watchpoints are the minimal configuration for data read visibility (#2) and may result in a performance impact in the 0-5% range (code dep.).	See box above. Watchpoint are also supported.
Use Model: Cal Data acquisition in ECU (time and position synchronous) while running an engine or vehicle. Need access to calibration variables with acceptable impact to the system under calibration. Today's bandwidth needs are > 40 bytes/1 msec for position synchronous and > 40 bytes/10 msec for time synchronous.	External to MCU memory emulation allows for direct access to calibration constants with no performance impact. Requires that external address and data pins are available. Time synchronous data may also be accessed by a serial link.	Can use similar method as used with today's MCUs, but there is a performance impact dependent on number of calibration variables (stored in on-chip fast RAM) being acquired . Requires that external address and data pins are available.	Auxiliary port may provide data acquisition using data write messaging (transparent to SW) or data messaging (SW writes to designated address(es), which are queued and transferred).

Use Model: Cal Updating calibration constants coherently in ECU while running an engine or vehicle. Need to switch between at least two maps or banks at a position synchronized event. Need calibration accessibility with acceptable impact to the system under calibration.	External to MCU memory emulation allows for instrumentation access to calibration constants with low (or not any) performance impact. Position synchronized bank switching hardware as implemented on a PIA is used for control of emulation memory banks. Requires that external address and data pins are available.	Can use similar method as used with today's MCUs.	Provides access via JTAG or auxiliary port to internal calibration tuning block. Allows for capability of a single-chip solution (no PIA required).
Use Model: HW Dev. and HW/SW Int. Muxing debug pins with GPIO can make logic analysis tools exhibit unpredictable behaviors that developers consider undesirable	Not applicable.	No work-around. Customer must put up with these anomalies.	Early tool vendor involvement will help customer understand potential problems which may be introduced in the development process by such muxing schemes. Auxiliary port is reset configurable.
Use Model: HW Dev., HW/SW Int. & Cal. MCU vendors do not support standard development interface/methodology across MCU architectures	Powertrain MCU vendors do not always provide the same development interface and features across their product line. This makes it difficult for automotive developers to upgrade since it requires an investment in capital and engineering.		An industry-wide development standard would developers to leverage existing development tools and methodologies