Use of Nexus Based Software Development Tools for Powertrain ECU's
Agenda

- Introduction (Marissa Jadrosich, IEEE-ISTO)
- Nexus Overview (Randy Dees, Freescale)
- Nexus Interface in Automotive uC’s (Randy Dees, Freescale)
- ECU Software Debugging and Trace (Udo Zoettler, Lauterbach)
- Calibration and Rapid Prototyping (Todd Collins, ETAS)
- Common Instrumentation Solutions (Norm D’Amico, GM)
- Tools in Use (Norm D’Amico, GM)
- Benefit of the Nexus Standard (Norm D’Amico, GM)
- QA
Introduction

Marissa Jadrosich, IEEE-ISTO
Nexus Overview

Randy Dees, Freescale
What is Nexus?

• Nexus is an industry standard debug standard for embedded microcontrollers.

• Growing out of a white paper written in 1998 from Freescale (then part of Motorola) along with Agilent Technologies (then HP) on a new debug standard, the Nexus Consortium joined forces with the IEEE-ISTO to release the first version of the standard, the IEEE-ISTO 5001-1999.

  • The standard was updated in 2003 to address enhancements and minor oversights (IEEE-ISTO 5001-2003).
  • An additional update was released in 2012 that adds support for the IEEE 1149.7 and Nexus messaging over an Aurora protocol link.

• The standard covers not only the debug protocol but also the pin interface and connectors, driving a new level of standardization.

• To date, Nexus is implemented on MCU devices from several semiconductor companies on a wide variety of core types.

• Information on the Nexus Consortium can be obtained at www.nexus5001.org
Nexus Overview

• Nexus provides a real-time debug and calibration interface

• Nexus utilizes a packet based message structure to transmit MCU debug information, such as program flow, process, or MCU data, out of the processor on a high speed, variable width bus.
  • Bus width and speed dictated by bandwidth requirements for a given MCU.

• Support multiple processor cores over a single interface

• The Nexus standard defines 4 levels of features, Class 1, 2, 3, and 4.
  • Each class has a minimum set of features that must be implemented. Additional features are optional.

• Program flow trace uses branch trace messaging.
The Nexus standard defines debug from Class 1 (static debug) all the way through to Class 4 with memory substitution and port replacement.
Nexus Class Definition – Class 1

Class 1
- Run Time Control
  - Read/Write MCU registers / memory
  - Set / Clear Breakpoints
  - Stop / Start code execution
  - Control entry into / exit from debug mode (from reset and user modes)
  - Stop execution on hitting a breakpoint and enter debug mode
  - Single step instructions
  - Read Nexus device ID

Class 2
- Dynamic Debug

Class 3
- Data Trace

Class 4
- Advanced Debug
Nexus Class Definition – Class 2

Class 2

All class 1 features plus:

- **Ownership Trace Messages** – Real time process / task ownership tracing
- **Watchpoint Messaging** – Trigger a nexus message on an event
- **Program Trace Messages** – Real time, non intrusive instruction trace

Optional Features:
- **Port Replacement** (of slow GPIO)
Nexus Class Definition – Class 3

Class 3
All class 2 features plus:
- **Real Time Data Access** – Registers / memory can be read/written real time
- **Real Time Data Trace (WRITES)**

Optional Features:
- **Real Time Data Trace (READS)**
- **Transmission of additional data used for data acquisition**
Nexus Class Definition – Class 4

Class 4
All class 3 features plus:

- **Watchpoint Triggering** – Allows a watchpoint to trigger trace event
- **Memory Substitution** – MCU can run code from memory in development tool (ROM emulation)
- **Over-Run Control** – Allows Nexus to stop core if buffers will overflow.

Optional Features:
- Start memory substitution on watchpoint
So How Does It Work?

• Nexus messaging is based on change of flow / data operations.
  • This significantly reduces the number of pins required for a Nexus implementation.

• Looking at program trace
  • A change of flow is anything that disrupts the linear code execution flow (branches taken, exceptions and interrupts).
  • Each time a change of flow event happens, the nexus module issues a BTM (Branch Trace Message) to the development tool.
  • Each BTM contains key information including
    • Number of instructions executed since last change of flow
    • For indirect branches / exceptions – information on relative branch target or exception vector address.
  • When the trace is complete, the debugger is then able to re-construct the program flow since it knows the start point and any changes of flow.
  • Initially and periodically a SYNC message is transmitted with full absolute address. This gives the debugger a start and intermediate reference point.
Nexus Auxiliary Output Messages

- Nexus Messages consist of a 6-bit TCODE followed by a variable number of packets (the number of packets for each TCODE is defined in the standard)

- Packet Types
  - Variable: Variable length packets, minimum length is 1. Must end on a port boundary. End defined by MSEx protocol
  - Vendor-Fixed: Fixed length field length is defined by chip vendor (by Nexus standard). Can be 0 length for unneeded fields.
  - Vendor-Variable: Variable length field, can be vendor defined as 0 length for unneeded fields. Must end on a port boundary. End defined by MSEx protocol.

- Messages can be Sync or Non-sync
  - Sync messages include full address and are required under certain conditions
  - Non-sync only include relative address change
Types of Trace Messages

- **Ownership Trace Messages**
  - Provides capability to trace the process, task, or thread switching

- **Program Trace Messages**
  - Provides trace of code execution

- **Data Trace Messages**
  - Provides capability to trace processor (or client) memory accesses, including address and values

- **Watchpoint Trace Messages**
  - Provide exact timing when events occur

- **Data Acquisition Trace Messages**
  - Provides user program control of data transmitted from the device

- **In-Circuit Trace Messages**
  - Provides custom trace capabilities for information that can’t be handled by any of the above message types.
Trace Message Example

- Direct branches (conditional or unconditional) are all taken branches whose destination is fixed in the instruction opcode.
- Messages for taken direct branches includes how many sequential instructions were executed since the last taken branch or exception.
- Indirect Branches are program discontinuities that are calculated or not known at compile time.

### Direct BTM

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Instruction Count</th>
<th>Message Source</th>
<th>TCODE (0x3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendor defined</td>
<td>1-8 bits</td>
<td>Vendor defined</td>
<td>6 bits</td>
</tr>
</tbody>
</table>

- Optional timestamp to record time event occurred
- Compressed (delta) address
- Number of instructions since last BTM
- For multi processor systems to identify which processor sent message
- Message transfer code

### Indirect BTM

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>U-ADDR</th>
<th>Instruction Count</th>
<th>Message Source</th>
<th>TCODE (0x3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendor defined</td>
<td>Variable length</td>
<td>1-8 bits</td>
<td>Vendor defined</td>
<td>6 bits</td>
</tr>
</tbody>
</table>
Address Compression

Full address is transmitted periodically, but address compression is used to reduce the number of bits that must be transmitted in the majority of trace message.

A1 = Previous trace message full address
A2 = Current trace message full address
CA2 = Current trace message compressed address

Compression (by Nexus Module)

<table>
<thead>
<tr>
<th>Address</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 (0x0002F124)</td>
<td>0000 0000 0000 0010 1111 0001 0010 0100</td>
</tr>
<tr>
<td>A2 (0x0002F256)</td>
<td>0000 0000 0000 0010 1111 0010 0101 0110</td>
</tr>
<tr>
<td>CA2 = A1 XOR A2</td>
<td>0000 0000 0000 0000 0000 0011 0111 0010</td>
</tr>
</tbody>
</table>

In compressed address CA2, high order 0’s are suppressed → Transmits 0x372

Decompression (by Development Tool)

<table>
<thead>
<tr>
<th>Address</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 (0x0002F124)</td>
<td>0000 0000 0000 0010 1111 0001 0010 0100</td>
</tr>
<tr>
<td>CA2 (0x00000372)</td>
<td>0000 0000 0000 0000 0000 0011 0111 0010</td>
</tr>
<tr>
<td>A2 = A1 XOR CA2</td>
<td>0000 0000 0000 0010 1111 0010 0101 0010</td>
</tr>
</tbody>
</table>
Nexus Ports

- The Nexus standard defines 2 possible Nexus port configurations
  - Auxiliary port only
  - Combined JTAG / Auxiliary port. In this implementation, read/write access is performed over the JTAG port. The standard allows Nexus messages to be sent over JTAG, however this is not realistic due to JTAG bandwidth
- Auxiliary port can be the “low-speed” Parallel Nexus interface or the high-speed Serial (Nexus (via Aurora interface)

<table>
<thead>
<tr>
<th>Combined JTAG / Aux</th>
<th>Nexus AUX In / Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTAG (4-6 pins)</td>
<td>AUX-in (3-5 pins)</td>
</tr>
<tr>
<td>AUX-in (1 pin)</td>
<td>AUX-out (6-17 pins)</td>
</tr>
<tr>
<td>Control and Data Read / Write messages</td>
<td>Program, Data, Ownership Trace, Watchpoint messages</td>
</tr>
<tr>
<td></td>
<td>Control / Data request messages</td>
</tr>
<tr>
<td></td>
<td>All Nexus output messages</td>
</tr>
</tbody>
</table>
Xilinx® Aurora Protocol
A scalable, hi-speed serial, link-level interface
Common protocol for single and multi lane channels
Serial Full Duplex or Serial Simplex operation
System-Synchronous or Asynchronous operation
Arbitrary data transfers: packets or words
Optional Flow Control & Expedited Messaging
8B/10B Data Encoding

Nexus over Aurora
The parallel Nexus Message Data Output and Message
Start/End Output information is encapsulated into an
Aurora Protocol Data Unit (PDU).
Can be simplex or duplex.
Aurora is a protocol for taking parallel data and serializing it over a LVDS connection
The Aurora protocol handles dividing the incoming parallel data into different lanes
Aurora Protocol

- Aurora transmits continuous data. If there are no Nexus messages, Idle messages are transmitted.

- Clock Compensation (CC) symbols are transmitted periodically.

- User PDUs are the actual Protocol Data Units (Nexus trace messages).

<table>
<thead>
<tr>
<th>Link Layer Frame Encapsulation</th>
<th>2 octets</th>
<th>12 octets</th>
<th>2 octets</th>
<th>2 octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCP</td>
<td>User PDUs</td>
<td>CC</td>
<td>User PDUs</td>
<td>CRC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8B/10B Encoding</th>
<th>2 octets</th>
<th>12 octets</th>
<th>2 octets</th>
<th>2 octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>/SCP/</td>
<td>/User PDUs/</td>
<td>/CC/</td>
<td>/User PDUs/</td>
<td>CRC/</td>
</tr>
</tbody>
</table>
Nexus Read/Write Access

- Normally, access to memory must be done through the core, one location at a time.
  - Requires that the core be stopped
  - Tool must write the address to a register
  - For writes, the tool must load the data into another register
  - Then an instruction be forced into the core to perform the actual transfer
  - For reads, the data would be read from a register after the instruction executed

- Nexus read/write access allows better throughput of data reads or writes by performing block reads or writes of memory
  - Address is written to a start R/W address register
  - For write data is written to R/W Data register
  - R/W Control register written with number of words to transfer, transfer word size
  - RDY pin signals when the tool can write additional data.
  - Can be done while the core is running
Nexus Trace Advantages

- Inherent Multi-core support including core identification in trace messages
- Instruction Trace via Branch Messaging to reduce bandwidth requirements.
- Data Trace support (reads and/or writes)
- Optional Support for time-stamping of messages
- Minimum definitions for feature supported
Nexus Interface in Automotive uC’s

Randy Dees, Freescale
• Optional IEEE 1149.7 powers up in pass through mode and can be enabled by the tools using only the pins available in the 1149.7 2-pin mode.
• 4 to 16 MDO pins, either 1 or 2-pin MSEO.
MPC5500 Nexus Interfaces

- **ETPU**
  - Engine 1
  - Engine 2
  - R/W Reg, Halt, step, cont
  - Program, Data, Ownership, Watchpoint Trace
  - Buffer
  - ETPU NDEDI

- **e200z6**
  - Program, Data, Ownership, Watchpoint Trace
  - Read/Write Access
  - Buffer
  - eDMAC
  - e200z6 NZ6C3

- **MMU Cache**
  - R/W Reg, R/W Data, Halt, step, cont

- **XBAR**
  - Off-chip Mem & I/O
  - On-chip Mem & I/O

- **NPC**
  - Data, Watchpoint Trace
  - Buffer
  - DMA NXDM

- **JTAG Controller JTAG Port**
  - RDY, TDI, TCK, TDO, TMS, JCOMP
  - EVTI, MSEQ[0:1], MCKO, MDO (4 or 12), EVTO

RDY used only for R/W Access
• Optional IEEE 1149.7 powers up in pass through mode and can be enabled immediately using the 1149.7 2-pin mode.
• Parallel or serial interfaces can be selected. It is preferred that they not share the same pins. The parallel interface can be set to 4, 8, 12, or 16 bits wide.
MPC5744P Block Diagram and Features

Key Functional Characteristics
- 2 x e200z4 in delayed lock step operating up to 180 MHz Power Architecture e200z4 computational cores
- Embedded Floating point unit
- ISO 26262 – ASIL-D assessment
- 4 x 12-bit Analog to Digital converters (16 channels each)
- 2.5MB Flash memory with ECC
- 384KB SRAM with ECC
- End-to-End ECC on data paths
- ECC implemented in peripheral memories (FlexCAN, FlexRay)

Nexus Debug Features
- Nexus Class 3+
- e200z4 Nexus client
- 2 X Nexus XBAR bus trace clients (one on DMA and the Zipwire interface. A second bus trace client for FlexRay)
- 4-bit parallel Nexus interface on low-pin count 144 QFP package and higher package (144QFP and 257 BGA)
- 2-lanebe Nexus Aurora Trace port on larger package (257 BGA)
• IEEE 1149.7 powers up in pass through mode and can be enabled immediately using the 1149.7 2-pin mode.

• 1, 2, 4, 6, or 8 lanes of Aurora can be implemented. (only 2 or 4 planned today)

• Aurora Clock must be supplied from external tool (625 MHz or 1.2GHz).

• Trace output can go to either the physical interface (NAL/NAP/pins) or memory.
MPC5746M 4M Block Diagram

Key Functional Characteristics
- Two independent 200 MHz Power Architecture z4 computational cores
  - Single 200 MHz Power Architecture e200z4 lockstep
  - Delayed lock-step for ASIL-D safety
- Single I/O Core 200 MHz Power Architecture z4 core
- 4M Flash with ECC
- 320k total SRAM with ECC
  - 128k of system RAM (incls. 64k standby on 292 PBGA package)
  - 192k of tightly coupled data RAM
- 6 ΣΔ converters for knock detection, 8 SAR converters – 60 total ADC channels
- GTM – 120 timer channels
- eDMA controller – 64 channels

Key Nexus Features
- Nexus trace to memory on Production and emulation devices
- Nexus Aurora Interface (2 or 4-lanes) on Emulation Device
- Expanded Trace memory on Emulation Device (1MB)
- Nexus Class 3+
MPC57xxM Trace Block diagram

1.25GHz Clock from external debug tool to MCU
1.25G bits/sec Aurora trace lanes from MCU to debug tool
   MPC5746M – 2 or 4 lanes
   MPC5744P - 2 lanes
MPC5746M Emulation Device Nexus Trace Flow

- **NAR**: Nexus Aurora Router
- **NAL**: Nexus Aurora Link (formatter)
- **NAP**: Nexus Aurora Physical Interface
- **Nexus Clients**
  - NZ4C3 e200z4 core processing Unit
  - NXMC Nexus Crossbar bus trace client
  - Sequence Processing Unit
**Aurora (8B10B) Signal Terminations**

<table>
<thead>
<tr>
<th>Tool</th>
<th>In target system</th>
<th>On-chip circuitry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock transmitter</td>
<td>10 pF</td>
<td>100 Ω</td>
</tr>
<tr>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100 Ω</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10 pF</td>
<td></td>
</tr>
<tr>
<td>Aurora receiver</td>
<td>10 pF</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>100 Ω</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100 Ω</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10 pF</td>
<td></td>
</tr>
<tr>
<td>Two to four Aurora lanes are supported.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aurora transmitter</td>
<td>10 pF</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>100 Ω</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100 Ω</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10 pF</td>
<td></td>
</tr>
</tbody>
</table>

- Debugger terminations typically included on chip (e.g., Xilinx FPGA)
- MCU input decoupling capacitors need to be located close to device

Aurora Input Clock 1.25 GHz

MCU

Aurora Data Lanes (1.25 Gbps)
ECU Software Debugging and Trace

Udo Zoettler, Lauterbach
Real time debugging and Trace Analysis

- Embedded software in engine and transmission controllers needs to be tested and validated for performance, safety and quality assurance.
- The use of the NEXUS debug interface on the microcontroller allows the Lauterbach debugger real time access to the cores to display source code and to have read/write access to memory without interference with the cores' real time code execution.
Lauterbach NEXUS class3 debugger

NEXUS class3 debugger features:

- Run control: Single Step, Run, Break
- Source code listing
- Read/write real time memory access
- Flash programming
- Real time Trace: Program execution, Data r/w history
- Advanced features: Function Runtime Analysis, Code Coverage Analysis, History based trace debugging
Run control, Source code listing, r/w real time memory access
Real time trace: Dual Core Program execution
Real time trace: dual core program execution, data r/w history
dual core program execution, data r/w history: merged display
Dual Core Function Runtime Analysis: Chart Display
Code Coverage Analysis: HLL Function Level
Code Coverage Analysis: Source Code / Instruction Level
Trace history based debugging: Context Tracking System
Calibration and Rapid Prototyping

Todd Collins, ETAS
Electromechanical systems comprise a substantial segment of the feature & function set of modern automobiles & trucks. As a function of the rising demands on drivability, convenience, safety, and environmental impact, the number and complexity of electronically controlled/implemented vehicle functions is increasing steadily.

Today’s mid-size cars are commonly equipped with engine and transmission controls, electronic controlled brakes, occupant safety controls as well as driver assistance and infotainment systems resulting in 40 or more electronic control units (ECU’s) on one vehicle.

A modern engine control module can process 250 million instructions per second, and the control software may contain over 500,000 lines of code and upwards of 20,000 function parameters.

A complex vehicle ECU network is interconnected to share data between ECU’s utilizing many different communication buses including CAN, Flexray, LIN, and Ethernet.
Calibration is an essential part of the development process of systems and vehicles from the first prototypes in the lab until the validation & release of production.

During Calibration, an optimized set of ECU parameters is determined for a new vehicle & is commonly used in the development of the various on vehicle control systems including: Engine, transmission, hybrid systems, battery, electric motor, chassis, braking, and vehicle control systems.

During Calibration, the tuning of vehicle characteristics is performed to meet the customer/vehicle requirements for fuel consumption, comfort, drivability, stability, performance and legislative standards (EPA, CARB, etc...)

Calibration is performed live/real time, while the vehicle/system is running in a lab environment, in a dyno environment, and in extreme conditions (temperature, weather, altitude, etc...).
Availability of ECU parameters/variables in real time to the calibration engineer is an essential aspect of a calibration system.

During calibration, ECU parameter measurement is also essential, as thousands of variables can be simultaneously measured. Some hybrid and vehicle control systems require high speed measurements of variables/parameters changing faster than 50uS.
Calibrated function:
\[ y = m \times x + b \]
Use of Nexus in vehicle calibration
Rapid Prototyping in vehicle control systems

- Rapid prototyping (RP) of new control algorithms is a common practice in the Automotive industry.
- Use of an RP system enables the test of new control algorithm’s / strategies before all the production pieces/parts are available.
- The ability to quickly evaluate and revise a new function is key to efficient software design.
- External bypass is one of the more common methods of RP whereby new functions run on external RP hardware. This can include connection to sensors and actuators via the ECU or via external RP hardware I/O.
- For an automotive real time system, the round trip time from the input to calculation to the output is critical.
- The use of Nexus provides high speed real time interface directly to the microcontroller, enabling the bypass with complex control algorithm’s calculated off-board the main target micro controller.
The availability of a Nexus interface on the Microcontroller enables the Calibration and RP systems to have a real time high speed access to the software parameters/variables.

Via the Nexus interface, a calibration engineer has access to the ECU data via a PC based graphical user interface, and can view and tune the ECU software/parameters in real time.

Via the Nexus interface, an RP engineer has access to the ECU SW in real time. Algorithms or parts of algorithms can be bypassed to test new control strategies. RP system can also include additional IO hardware to provide additional ECU IO when needed.

The Nexus 5001 standard has continued to evolve, enabling access to the ever increasing complex ECU control software, the ever increasing bandwidth demands, and the ever increasing high speed requirements for modern vehicle control systems.

The introduction of Aurora in the 2012 revision to the Nexus 5001 standard will further enable the increasing demands of future vehicle electronic control systems.
Common Instrumentation Solutions

Norm D’Amico, General Motors
Instrumentation Strategies

- GM Powertrain has developed instrumentation strategies that are used across entire generations of ECU programs, including:
  - ECMs (Engine Control Modules)
  - TCMs (Transmission Control Modules)
  - Hybrids (Hybrid Control Modules)

- Today’s webinar will touch on two of the common strategies (Gen 2 & Gen 3) with a focus on the tools used in the development of ECU software and calibrations
Tool Use Cases

- There are 3 primary use cases employed at GM in the development of ECU software and calibration
  - Calibration: Measurement and calibration tools
  - Software Development: Software debugging / trace tools
  - Feature Development: Rapid prototyping tools

- The next few slides illustrate how these tools are being used on the Gen 2 and Gen 3 controller programs at GM
Gen 2 Overview

• External instrumentation – instrumentation was outside of the development ECU

• Each ECU supplier was required to implement a mezzanine / daughter board for instrumentation

• All 3 tool interfaces could not be used simultaneously
  • Maximum of 2 tools could be connected at the same time
Gen 2 Implementation

- Measurement & Calibration

![Diagram of Gen 2 Implementation]
Gen 2 Implementation

- Measurement & Calibration with Debug & Trace

![Diagram of Gen 2 Implementation](image_url)
Gen 2 Implementation

- Measurement & Calibration with RCP

Diagram:
- Nexus Class 3, JTAG, A/D bus
- MPC55xx
- Development Controller
- Custom Tier-1 Mez / Daughter Board
- Cal RAM
- Nexus JTAG
- Glenair connector
- ETAS ETKS3
- dSPACE GSI POD
- Developer’s PC
- Ethernet connections
Gen 3 Overview

- Internal instrumentation – instrumentation was inside of the development ECU

- Common internal instrumentation (COTS) eliminated the need for supplier designed daughter board

- All 3 tool interfaces can be used simultaneously
Gen 3 Implementation

- Measurement & Calibration
Gen 3 Implementation

- Measurement & Calibration with Debug & Trace
Gen 3 Implementation

- Measurement & Calibration, Debug & Trace, and RCP

[Diagram of Gen 3 Implementation with connections and components]

- Nexus Class 3+, JTAG, A/D Bus
- MPC56xx
- VertiCal Base
- dSPACE DS543
- ETAS XETKV2
- Samtec Connector
- LEMO connector
- LVDS connector
- T&D
- M&C
- Developer’s PC
- ethernet or USB
- ethernet
- ethernet
Tools in Use

Norm D’Amico, General Motors
Gen 2 Measurement & Calibration

Gen 2 ECU

ETAS ETKS3
Gen 2 Measurement & Calibration with Trace & Debug

Lauterbach LA-7690
Lauterbach LA-7610
Lauterbach Glenair Adapter
ETAS ETKS3
Gen 2 ECU
Gen 2 Measurement & Calibration with RCP
Gen 3 Measurement & Calibration

Gen 3 ECU

ETAS XETKV2 (inside)
Gen 3 Measurement & Calibration with Trace & Debug
Gen 3 Internal Instrumentation (XETKV2)
Gen 3 Instrumentation (multiple XETKV2s)
Gen 3 Bench
Gen 2 Development Vehicle
Gen 2 Development Vehicle (close up)
Benefit of the Nexus Standard with Respect to Tools

Norm D’Amico, General Motors
Lower Development Costs

- Elimination of the “one off” instrumentation solutions that were only used on a single program
  - Seems obvious – but multiple instrumentation solutions were used simultaneously at GM prior to the common “generational” strategy

- Programs share development equipment – efficiency/cost gains
  - As older programs ramp down their equipment needs also diminish
  - Newer programs utilize common equipment from older programs

- Don’t need skills / training for multiple tool sets

- Common tools used across the organization allows developers (software and calibration) to move across programs quite easily
  - Increases Powertrain’s ability to move people more effectively
Benefits to Common

- Powertrain’s instrumentation strategy being used as a model for other parts of GM

- ECU development in other parts of GM are being asked to follow Powertrain’s common instrumentation approach

- The Nexus Standard has played a significant role for realizing these benefits at GM
Questions and Answers

During the Webinar, Send questions to the Host using the Chat Window
Thank You!

For more information on Nexus 5001 Forum, please visit:

www.nexus5001.org