

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



A PROGRAM OF THE IEEE
INDUSTRY STANDARDS AND
TECHNOLOGY ORGANIZATION

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 Members



Fulcware
Corporation



Inspiring Innovation and Discovery

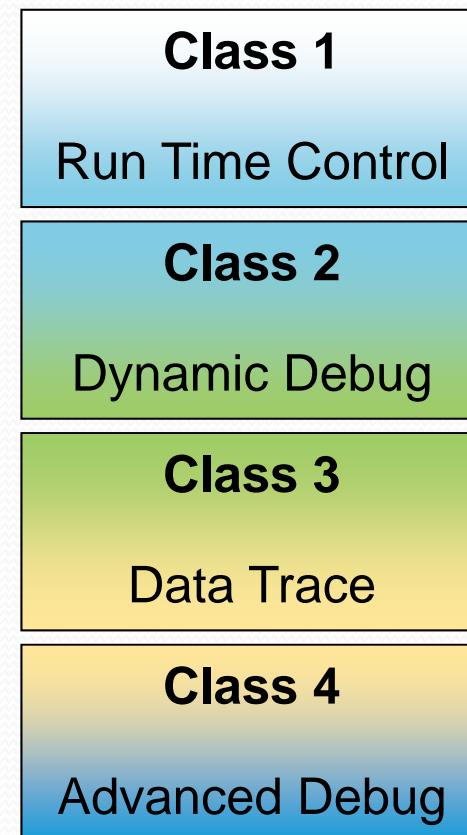


Nexus Overview

- 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.

Nexus Class Definition

The Nexus standard defines debug from Class 1 (static debug) all the way through to Class 4 with memory substitution and port replacement



Run Control

**Memory and
Port Substitution**

Nexus Class Definition – Class 1

Class 1

Run Time Control

Class 2

Dynamic Debug

Class 3

Data Trace

Class 4

Advanced Debug

Class 1

- 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

Nexus Class Definition – Class 2

Class 1

Run Time Control

Class 2

Dynamic Debug

Class 3

Data Trace

Class 4

Advanced Debug

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 1

Run Time Control

Class 2

Dynamic Debug

Class 3

Data Trace

Class 4

Advanced Debug

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 1

Run Time Control

Class 2

Dynamic Debug

Class 3

Data Trace

Class 4

Advanced Debug

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 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

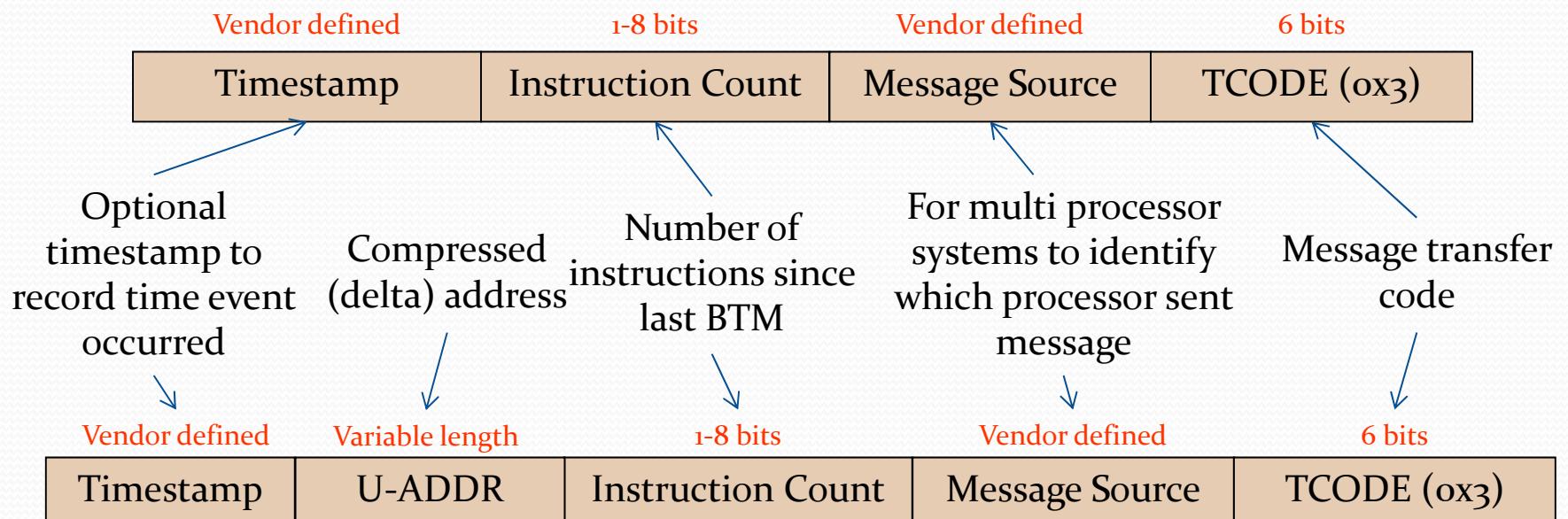
Types of Trace Messages

- 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



Indirect BTM

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)

A1 (0x0002F124)	0000	0000	0000	0010	1111	0001	0010	0100
A2 (0x0002F256)	0000	0000	0000	0010	1111	0010	0101	0110
CA2 = A1 XOR A2	0000	0000	0000	0000	0000	0011	0111	0010

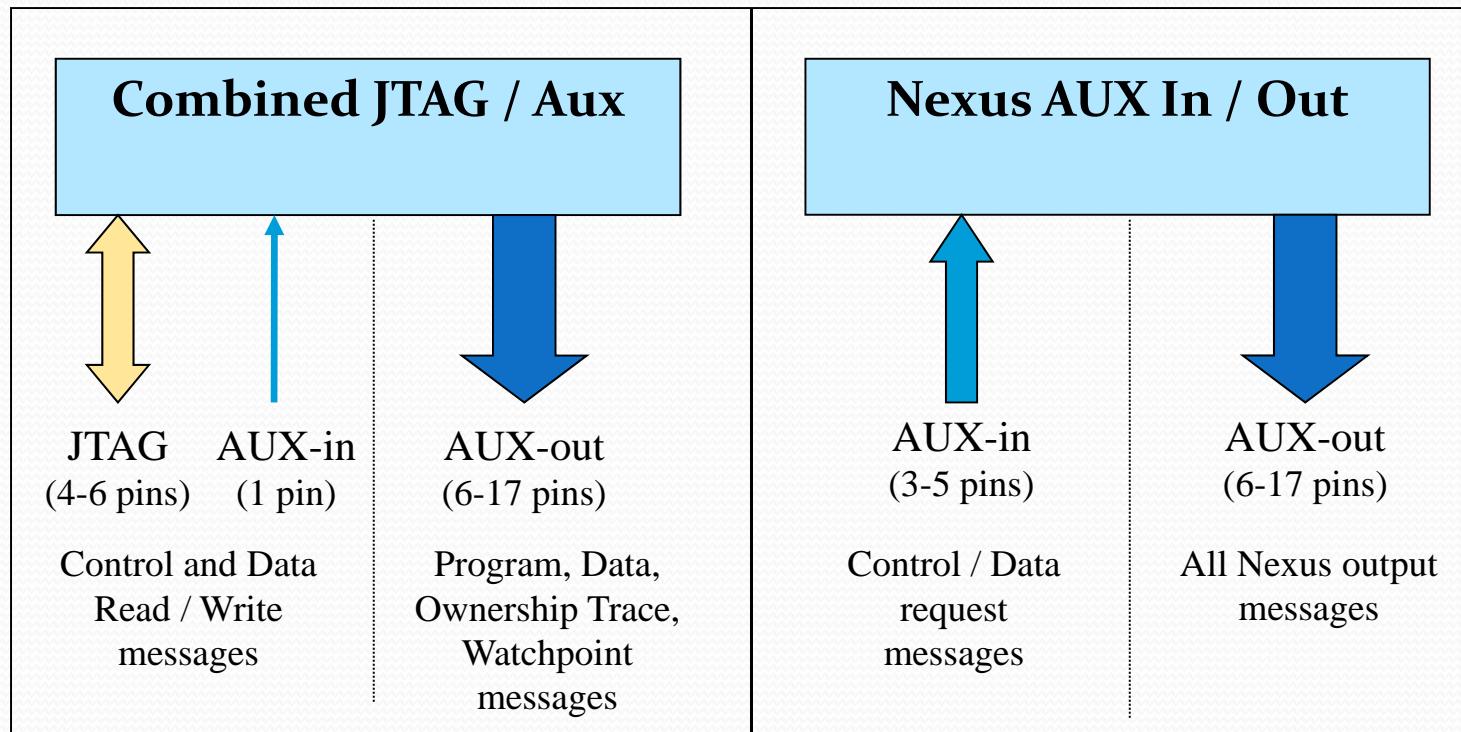
In compressed address CA₂, high order 0's are suppressed → Transmits ox372

Decompression (by Development Tool)

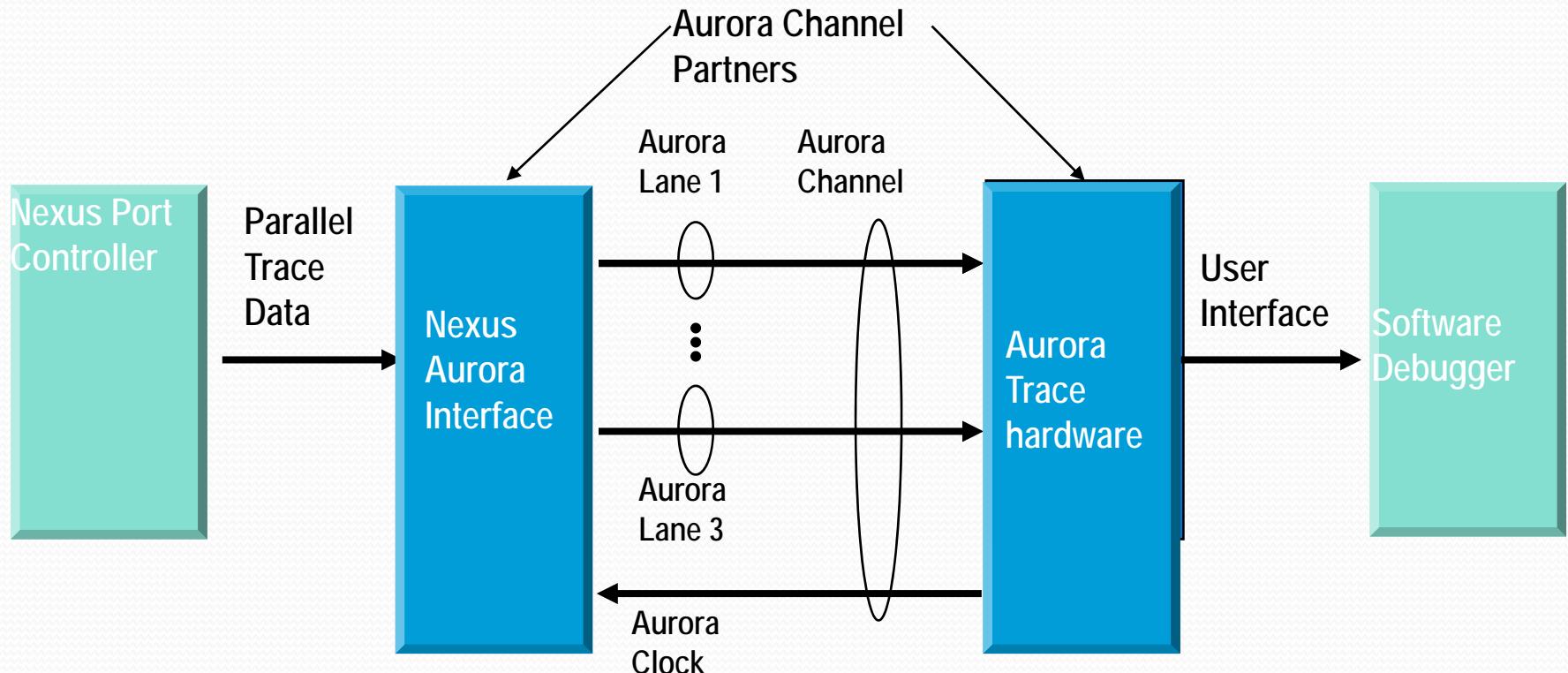
A1 (0x0002F124)	0000	0000	0000	0010	1111	0001	0010	0100
CA2 (0x00000372)	0000	0000	0000	0000	0000	0011	0111	0010
A2 = A1 XOR CA2	0000	0000	0000	0010	1111	0010	0101	0010

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)



Nexus Aurora Trace



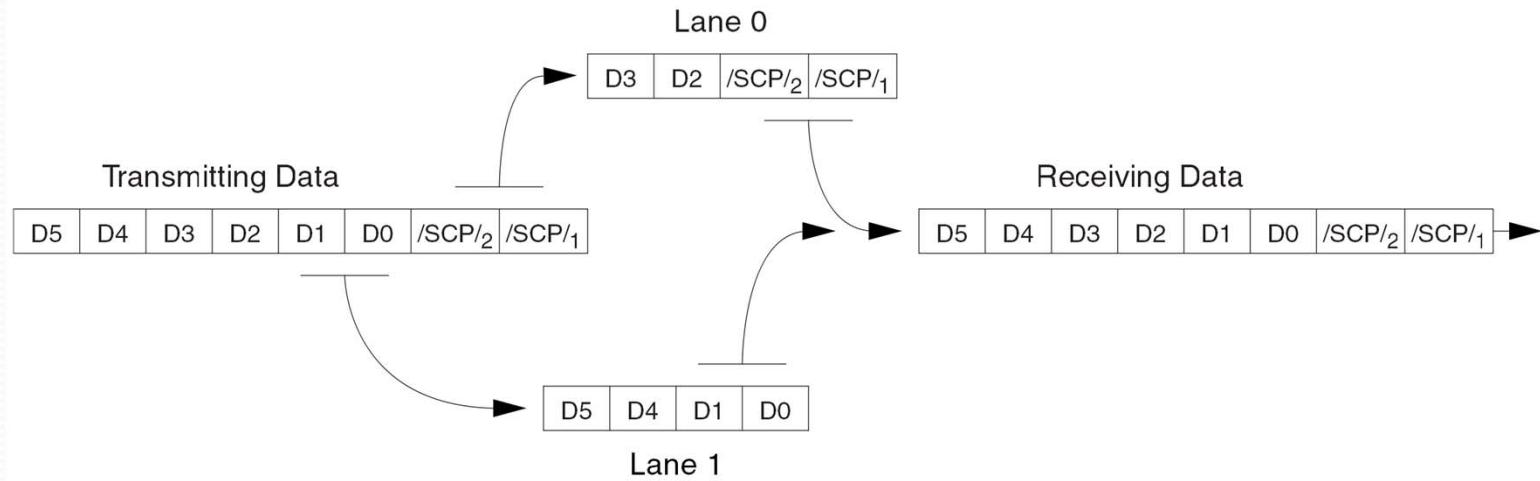
Xilinx® Aurora Protocol

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 Lane Striping

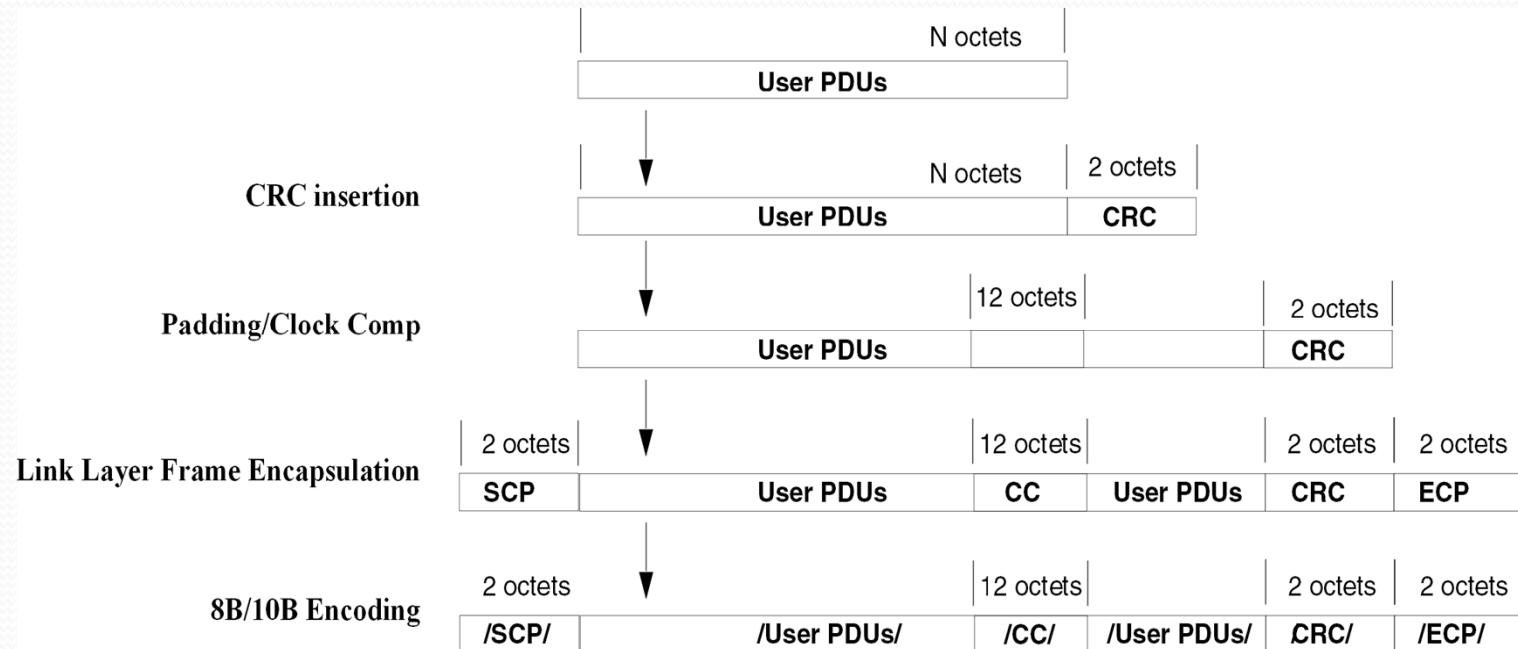


Aurora is a protocol for taking parallel data and serializing it over a LVDS connection

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)



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

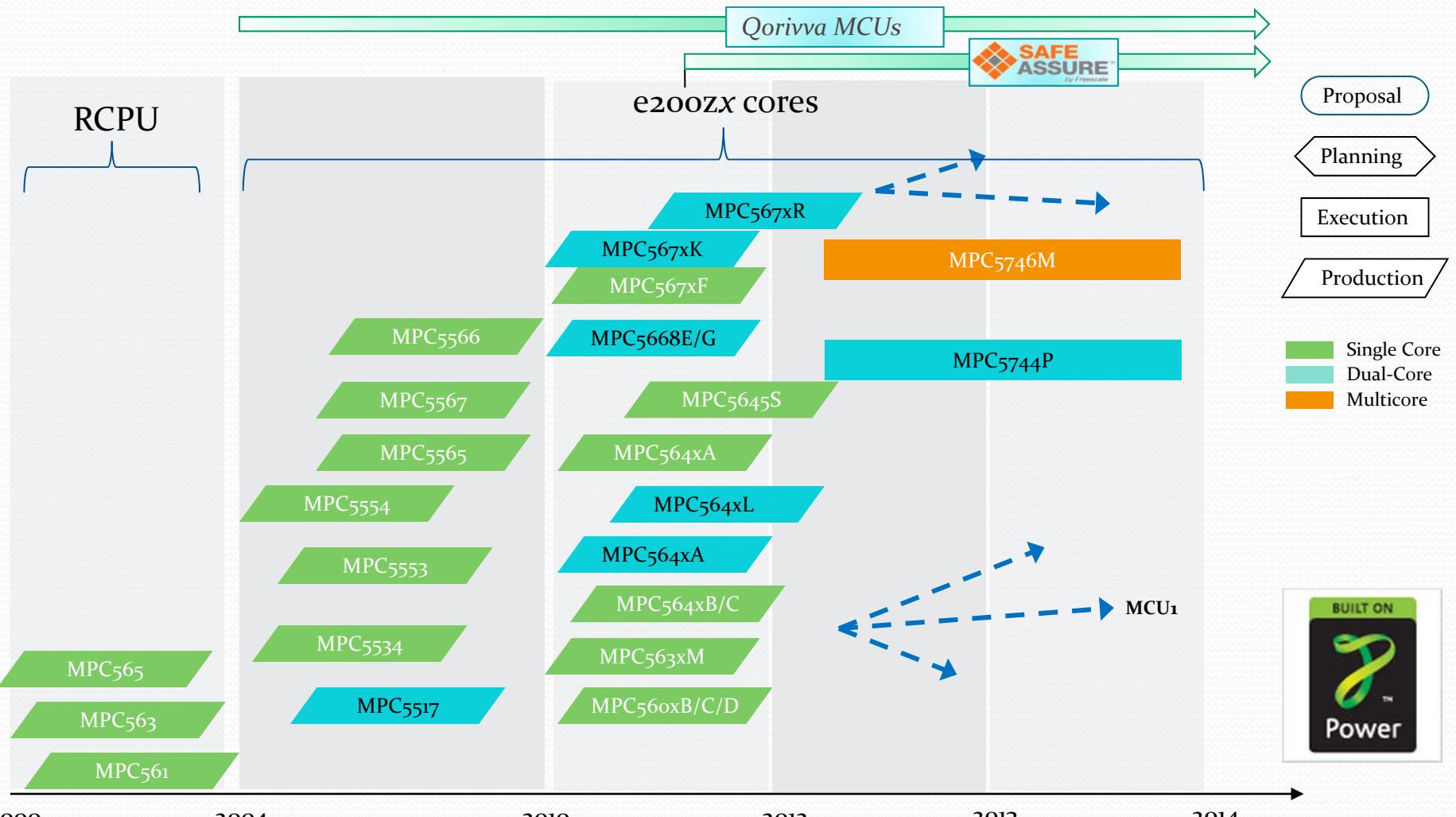
- Inherent Multi-core support including core identification in trace messages
- Instruction Trace via Branch Messaging to reduce bandwidth requirements.
- Data Trace support (reads and/or writes)
- Optional Support for time-stamping of messages
- Minimum definitions for feature supported

Nexus Interface in Automotive uC's

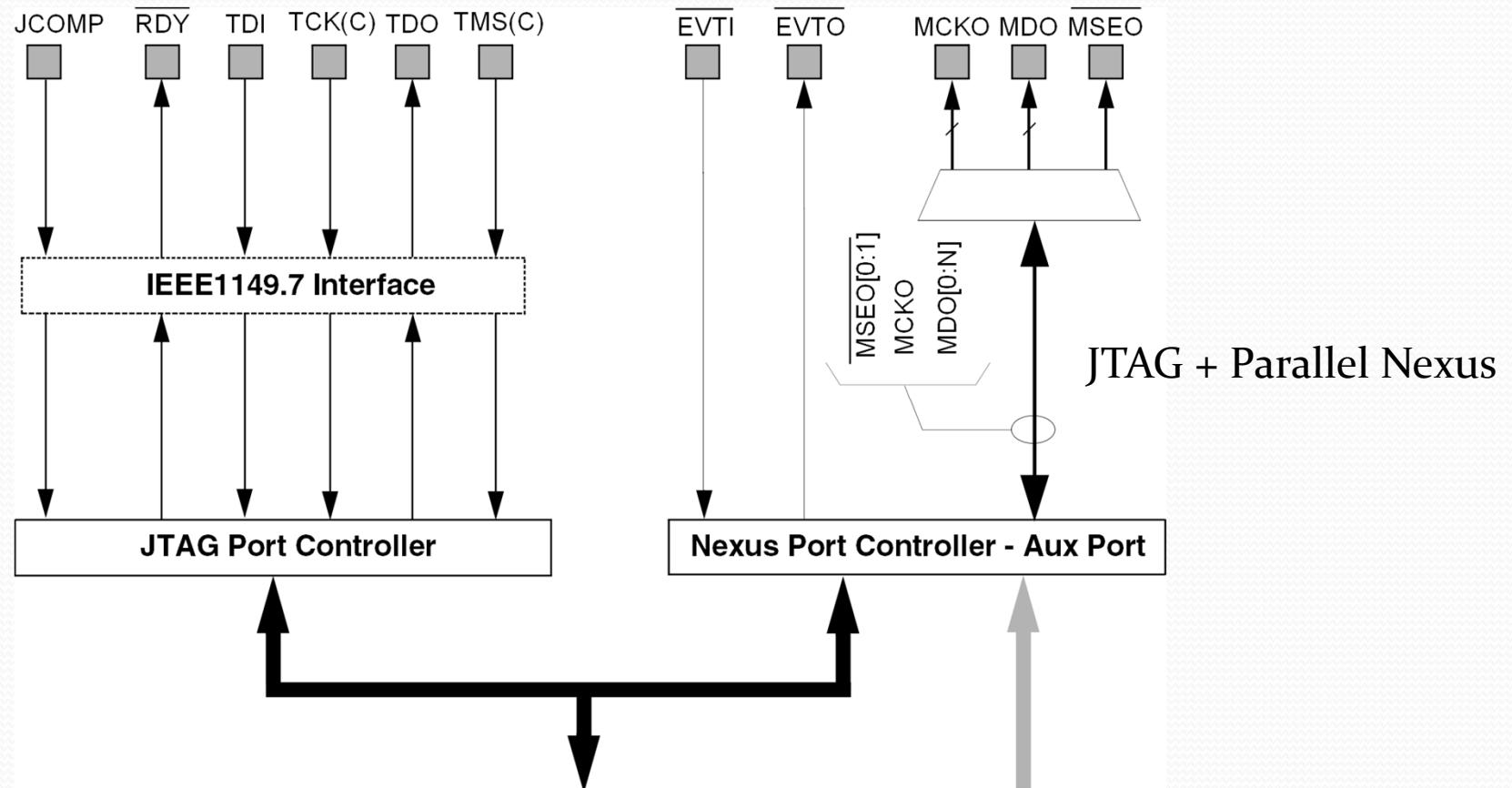
Randy Dees, Freescale



Automotive Nexus MCU Roadmap

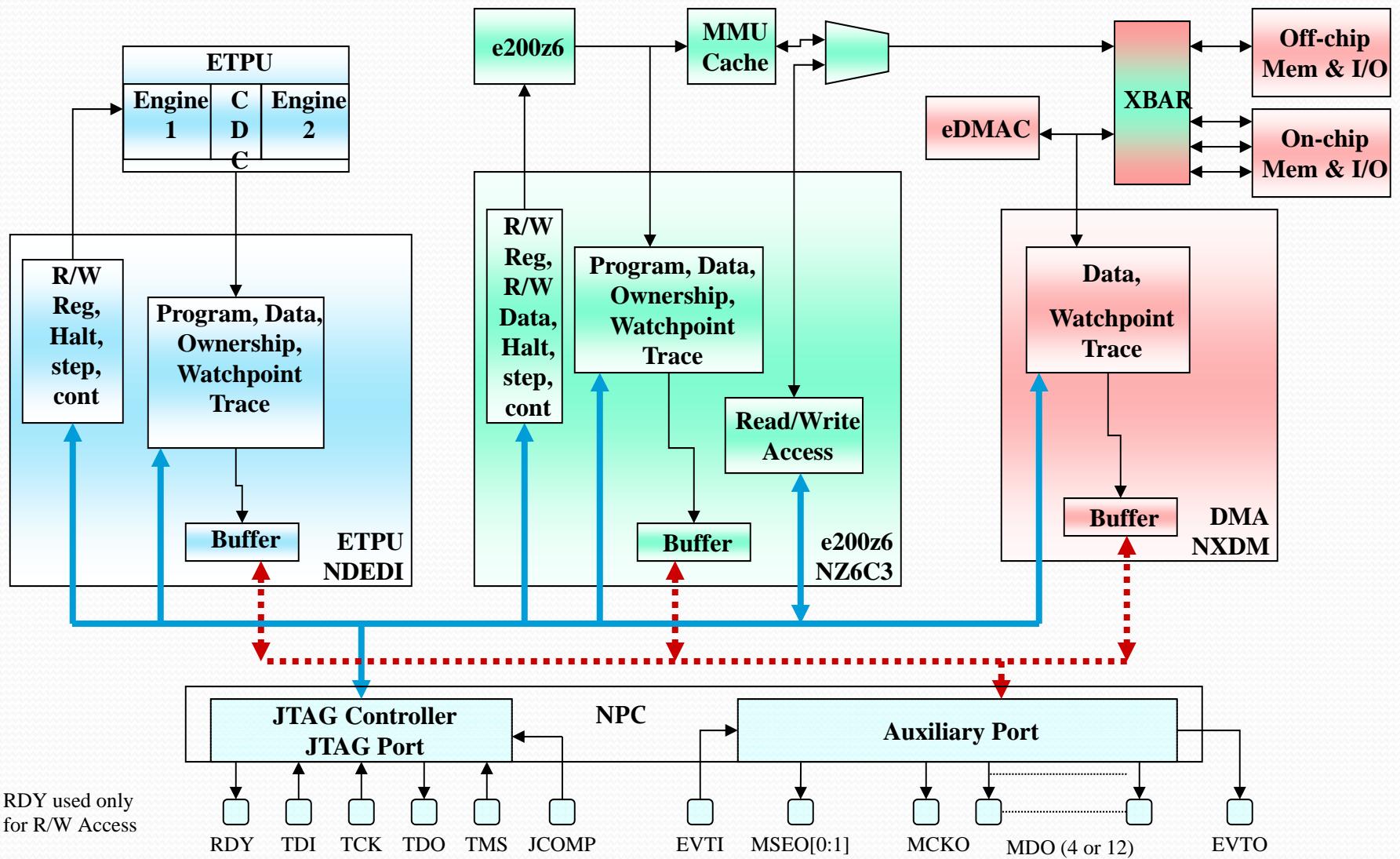


Low-End Nexus Trace Solution

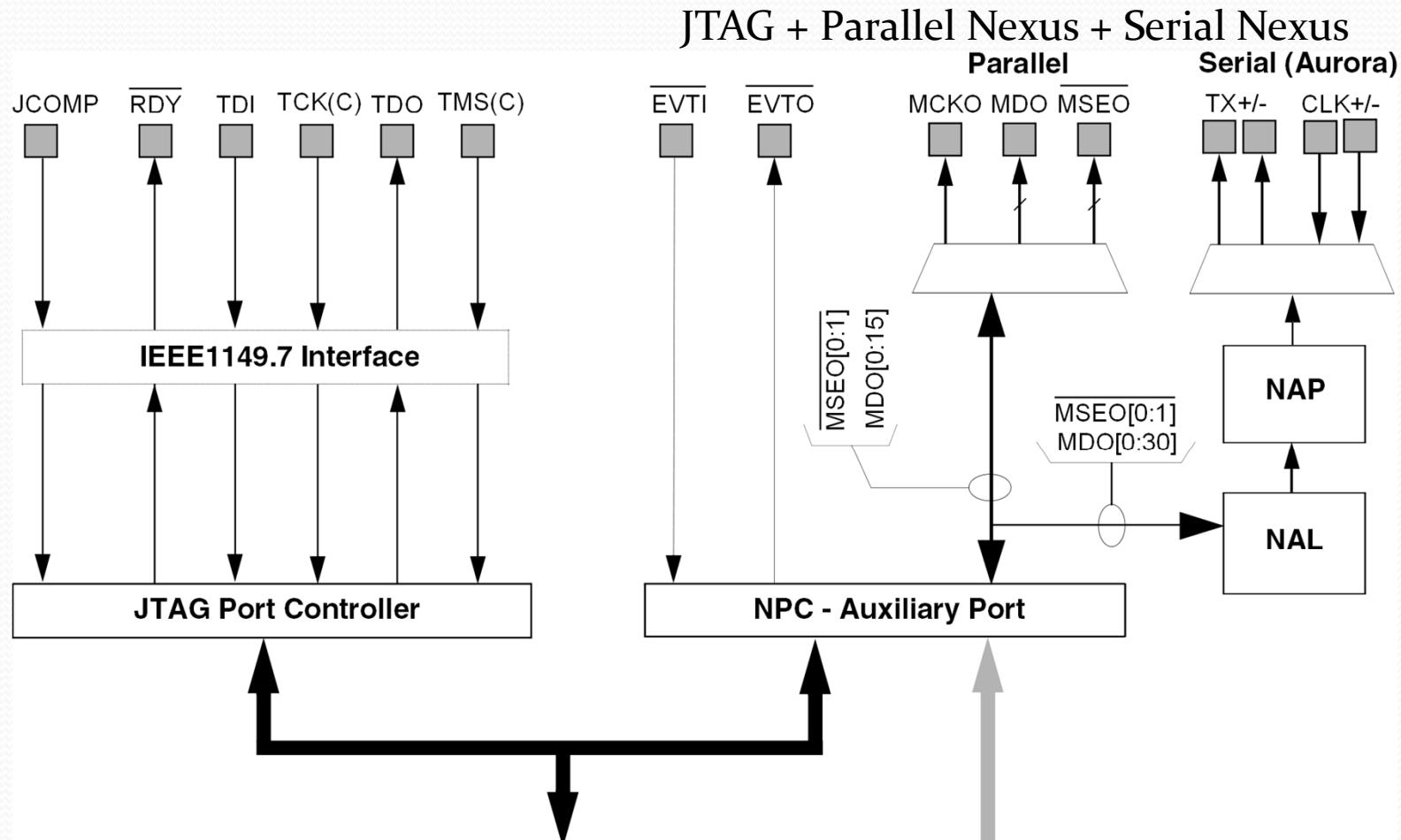


- Optional IEEE 1149.7 powers up in pass through mode and can be enabled by the tools using only the pins available in the 1149.7 2-pin mode.
- 4 to 16 MDO pins, either 1 or 2-pin MSEO.

MPC5500 Nexus Interfaces



Mixed Nexus Trace Solution



- 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

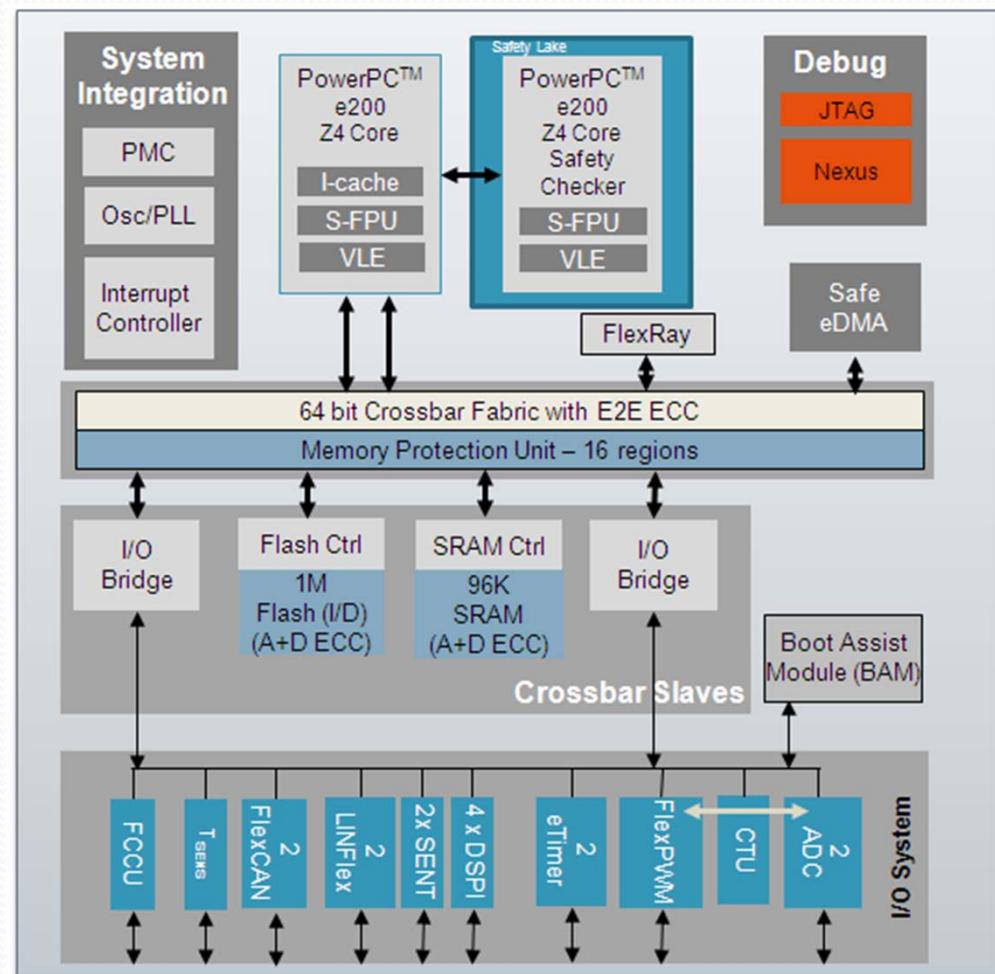


Key Functional Characteristics

- 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

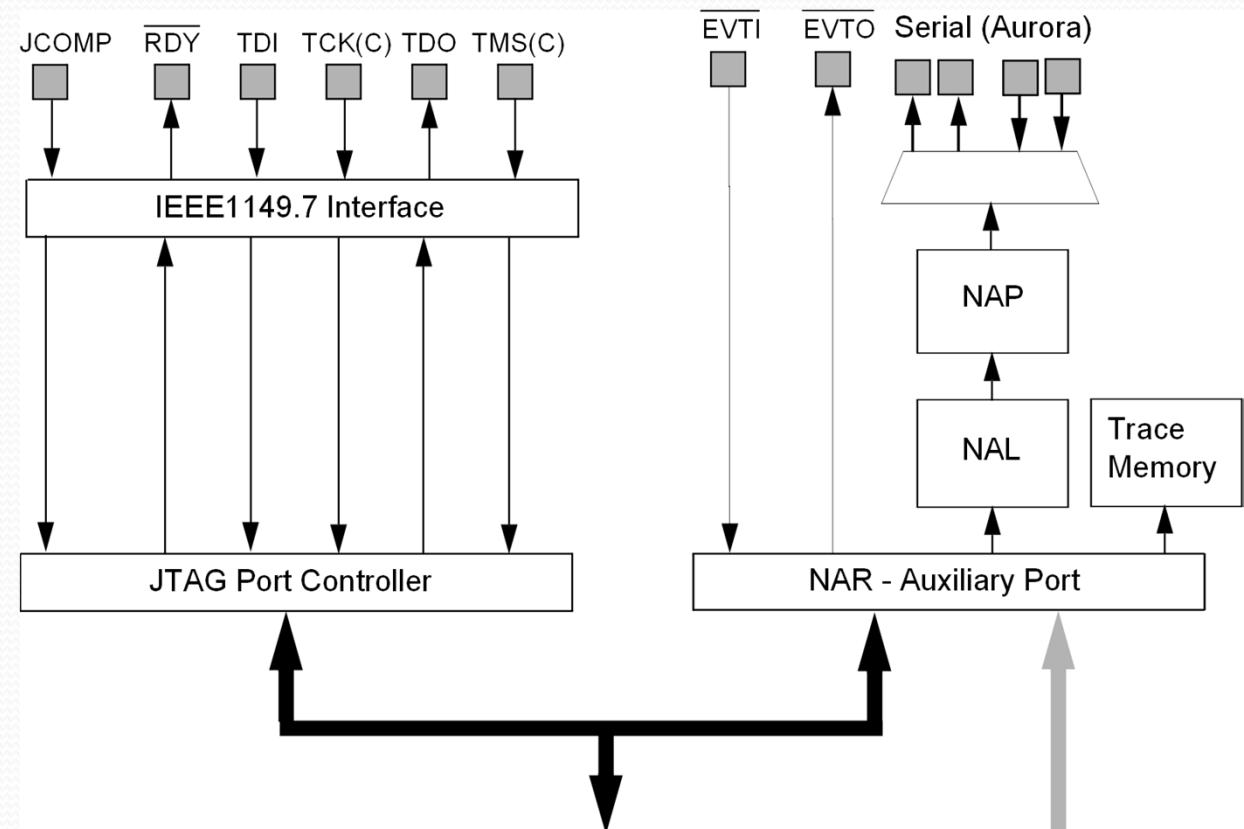
- Nexus Class 3+
- e200z4 Nexus client
- 2 X Nexus XBAR bus trace clients (one on DMA and the Zipwire interface. A second bus trace client for FlexRay)
- 4-bit parallel Nexus interface on low-pin count 144 QFP package and higher package (144QFP and 257 BGA)
- 2-lane Nexus Aurora Trace port on larger package (257 BGA)



Serial Nexus Debug Solution

- 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.

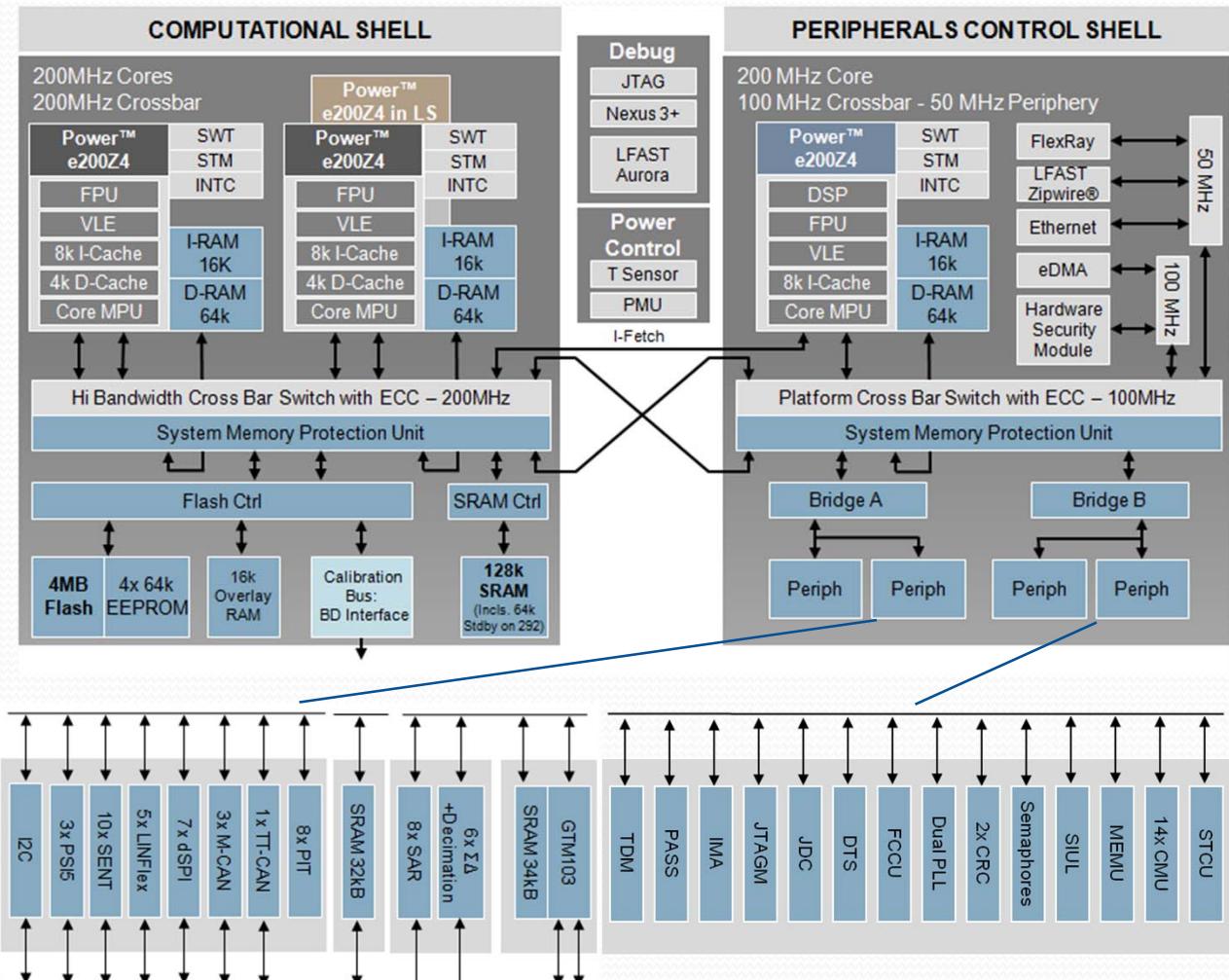
JTAG + Serial Nexus



MPC5746M 4M Block Diagram

Key Functional Characteristics

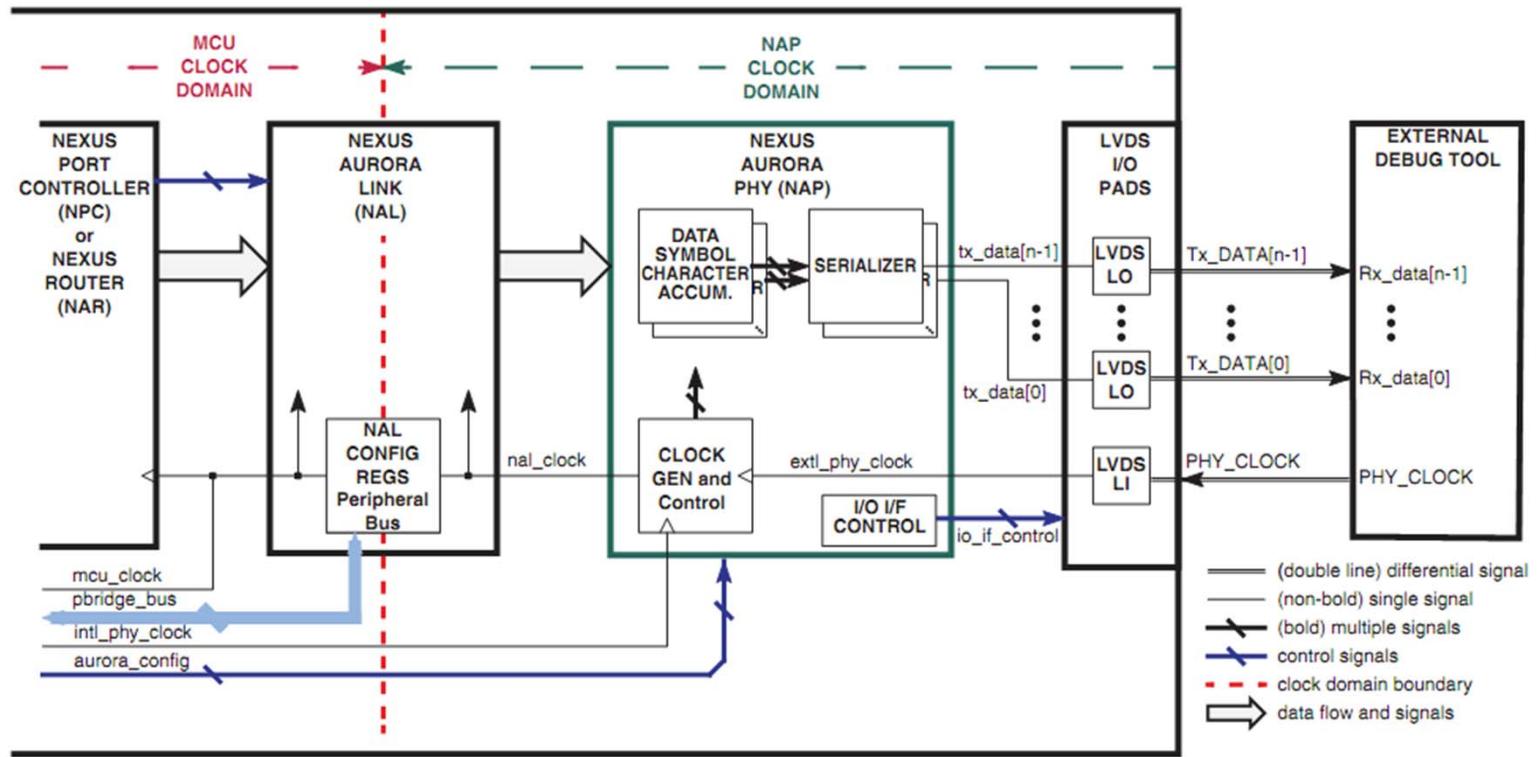
- Two independent 200 MHz Power Architecture z4 computational cores
 - Single 200 MHz Power Architecture e200z4 lockstep
 - Delayed lock-step for ASIL-D safety
- Single I/O Core 200 MHz Power Architecture z4 core
- 4M Flash with ECC
- 320k total SRAM with ECC
 - 128k of system RAM (incls. 64k standby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



Key Nexus Features

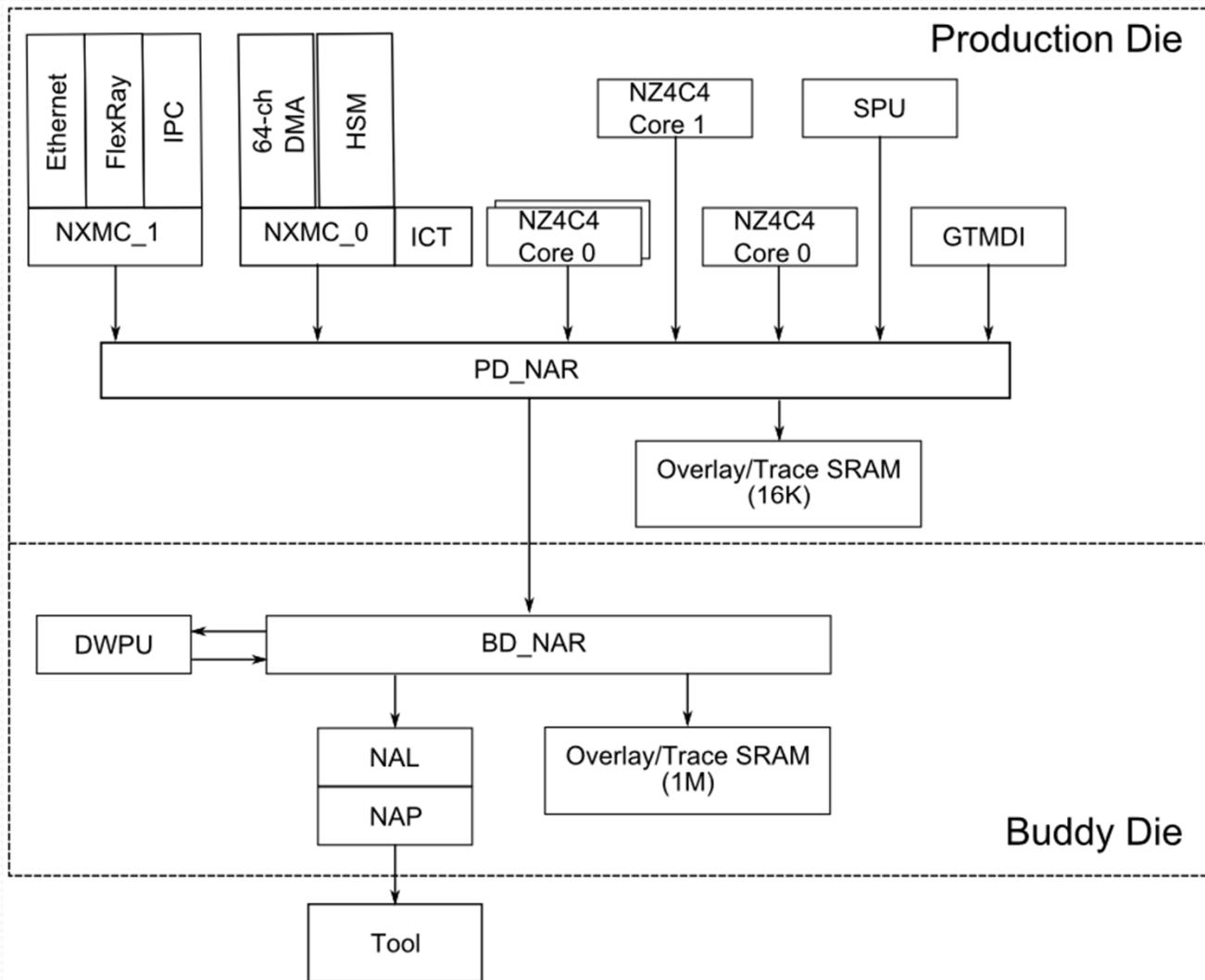
- Nexus trace to memory on Production and emulation devices
- Nexus Aurora Interface (2 or 4-lanes) on Emulation Device
- Expanded Trace memory on Emulation Device (1MB)
- Nexus Class 3+

MPC57xxM Trace Block diagram



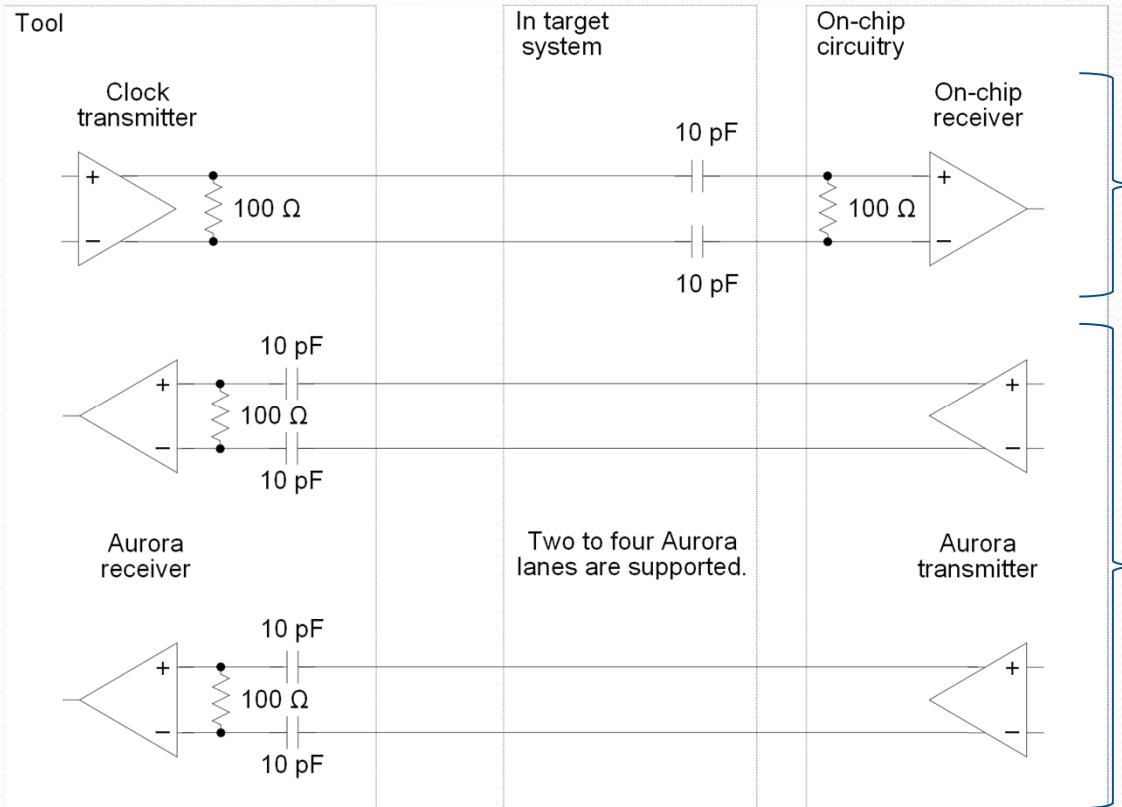
1.25GHz Clock from external debug tool to MCU
 1.25G bits/sec Aurora trace lanes from MCU to debug tool
 MPC5746M – 2 or 4 lanes
 MPC5744P - 2 lanes

MPC5746M Emulation Device Nexus Trace Flow

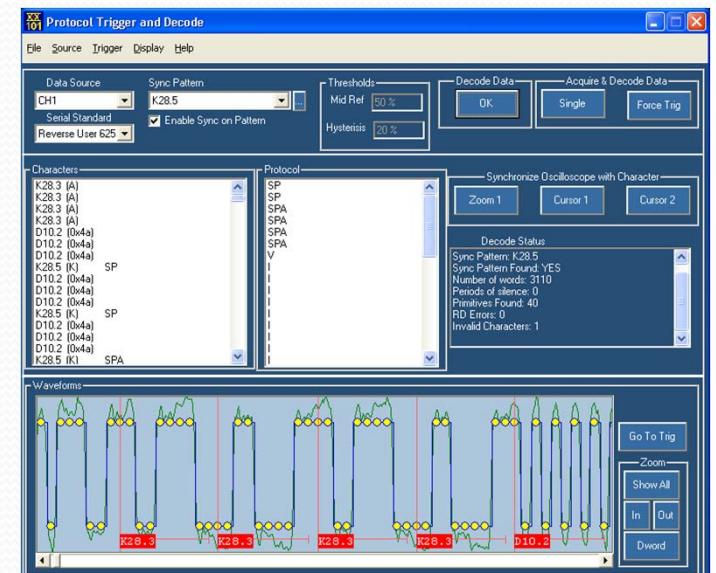


- NAR: Nexus Aurora Router
- NAL: Nexus Aurora Link (formatter)
- NAP: Nexus Aurora Physical Interface
- Nexus Clients
 - NZ4C₃ e20oz4 core processing Unit
 - NXMC Nexus Crossbar bus trace client
 - Sequence Processing Unit

Aurora (8B10B) Signal Terminations



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



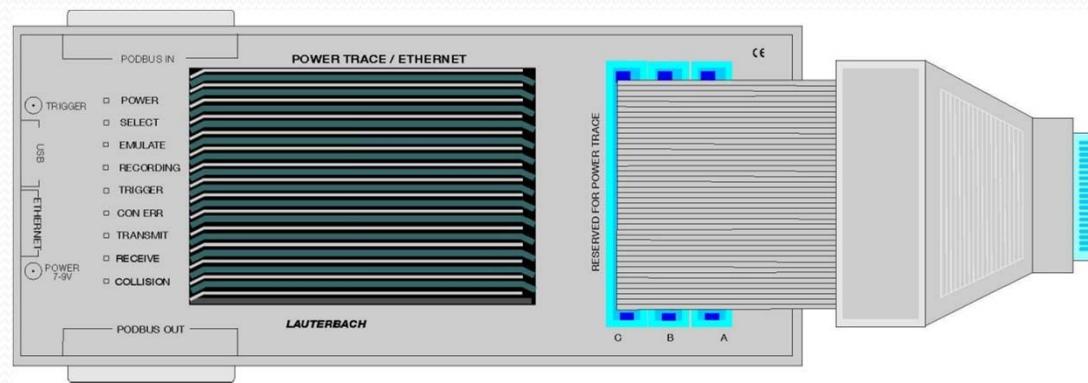
ECU Software Debugging and Trace

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.

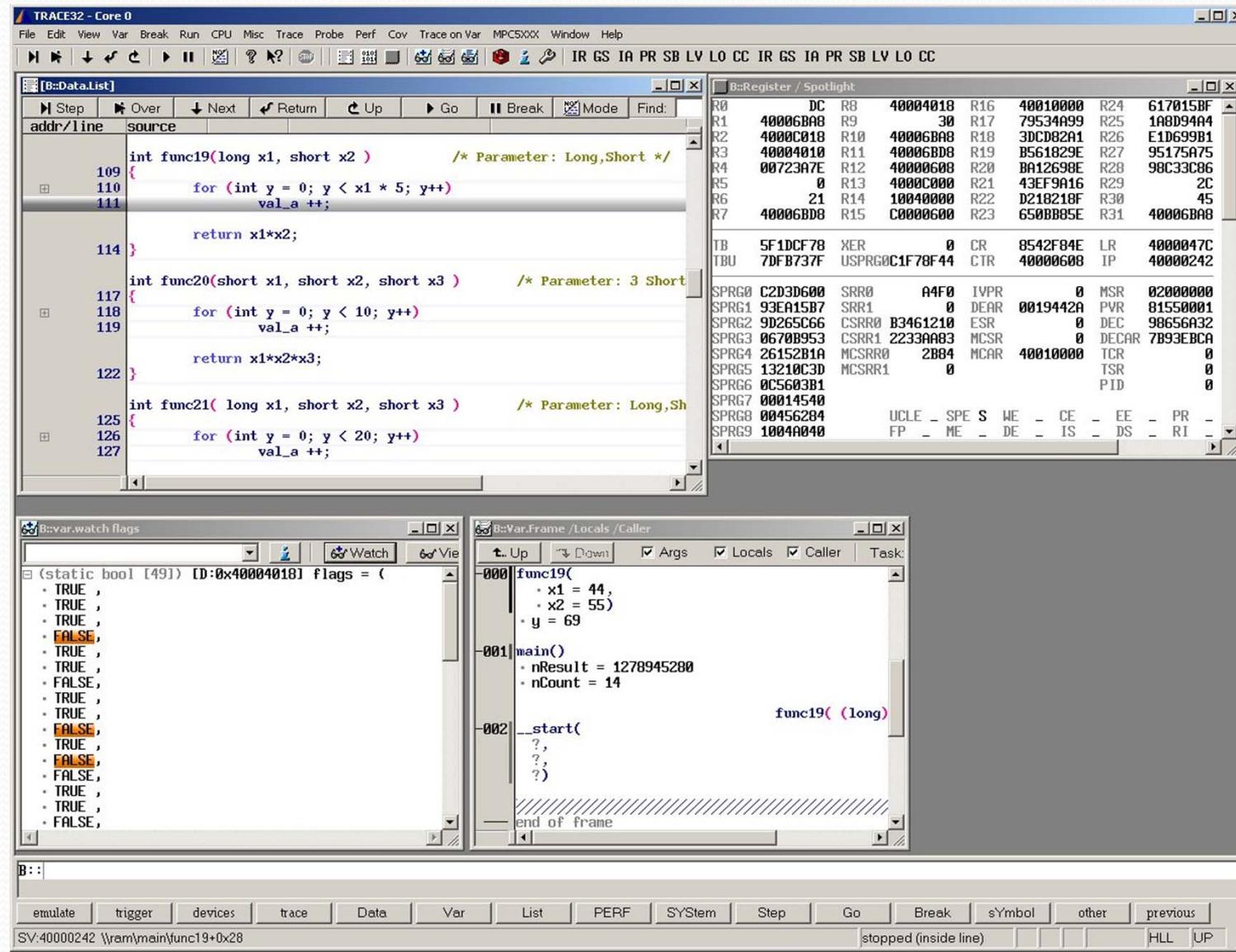


Lauterbach NEXUS class3 debugger

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

Two windows showing real-time traces for a dual-core program execution. Both windows have a header bar with tabs: Setup, Goto..., Find..., Chart, Profile, MIPS, More, Less, record, run, address, cycle, data, symbol, ti.back, ti.zero.

The left window shows the trace for core B (Analyzer.List def ti.zero ti.back /t). The right window shows the trace for core A (Real def ti.zero ti.back).

Core B Trace (Left Window):

```

record run address          cycle   data      symbol    ti.back  ti.zero
51   for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ;
     | cmp r1,#0x12           ; i,#18
     | bne 0x403260B4
     | strb r2,[r12,r1]       ; i,i,#1
     | add r1,r1,#0x1          ; i,i,#1

51   for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ;
     | cmp r1,#0x12           ; i,#18
     | bne 0x403260B4
     | strb r2,[r12,r1]       ; i,i,#1
     | add r1,r1,#0x1          ; i,i,#1

51   for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ;
     | cmp r1,#0x12           ; i,#18
     | bne 0x403260B4
     | strb r2,[r12,r1]       ; i,i,#1
     | add r1,r1,#0x1          ; i,i,#1

51   for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ;
     | cmp r1,#0x12           ; i,#18
     | bne 0x403260B4
     | strb r2,[r12,r1]       ; i,i,#1
     | add r1,r1,#0x1          ; i,i,#1

51   for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ;
     | cmp r1,#0x12           ; i,#18
     | bne 0x403260B4
     | strb r2,[r12,r1]       ; i,i,#1
     | add r1,r1,#0x1          ; i,i,#1

51   for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ;
     | cmp r1,#0x12           ; i,#18
     | bne 0x403260B4
     | strb r2,[r12,r1]       ; i,i,#1
     | add r1,r1,#0x1          ; i,i,#1

51   NR:403260B4 ptrace \\\demo\\demo\\sieve+0x14 <0.005us 0.

-0000049653
      strb r2,[r12,r1]
      add r1,r1,#0x1          ; i,i,#1

51   for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ;
     | cmp r1,#0x12           ; i,#18
     | bne 0x403260B4

53   for ( i = 0 ; i <= SIZE ; i++ )
     | mov r2,#0x0
     |
     | if ( flags[ i ] )
     | ldrb r1,[r12,r2]
     | cmp r1,#0x0
     | beq 0x40326130
     |
     | primz = i + i + 3;
     | lsl r1,r2,#0x1          ; r1,i,#1
     | add r3,r1,#0x3          ; r1,i,primz
     | add r1,r2,r3            ; r1,i,primz
     | register int i, primz, k;
     | int anzah;

```

Core A Trace (Right Window):

```

record run address          cycle   data      symbol    ti.back  ti.zero
51   for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ; <0.005us -117.185us
51   for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ; <0.005us -117.185us
51   for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ; 17.578us -99.607us
51   for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ; 23.437us -76.170us
51   for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ; 23.437us -52.734us
51   for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ; 23.437us -29.297us
51   for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ; 23.437us -5.860us
51   for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ; 5.860us 0.000us
53   for ( i = 0 ; i <= SIZE ; i++ ) <0.005us 0.000us
55   if ( flags[ i ] ) <0.005us 0.000us
57   primz = i + i + 3; <0.005us 0.000us
58   k = i + primz; <0.005us 0.000us
register int i, primz, k;
int anzah;

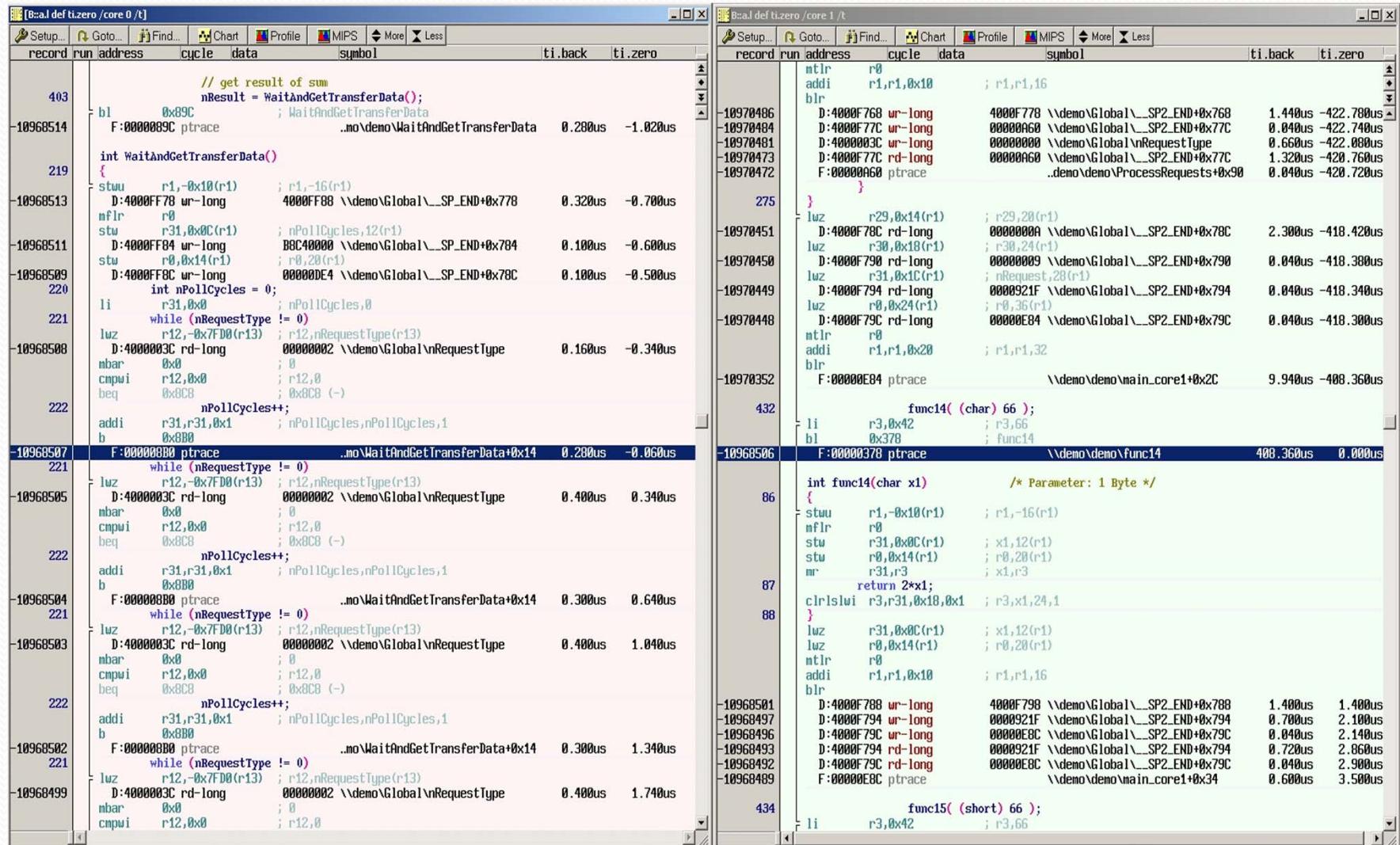
anzahl = 0; <0.005us 0.000us
while ( k <= SIZE ) <0.005us 0.000us
{
  flags[ k ] = FALSE; <0.005us 0.000us
  k += primz; <0.005us 0.000us
  while ( k <= SIZE ) <0.005us 0.000us
  {
    flags[ k ] = FALSE; <0.005us 0.000us
    k += primz; <0.005us 0.000us
    while ( k <= SIZE ) <0.005us 0.000us
    {
      flags[ k ] = FALSE; <0.005us 0.000us
      k += primz; <0.005us 0.000us
      while ( k <= SIZE ) <0.005us 0.000us
      {
        flags[ k ] = FALSE; <0.005us 0.000us
        k += primz; <0.005us 0.000us
        while ( k <= SIZE ) <0.005us 0.000us
        {
          flags[ k ] = FALSE; <0.005us 0.000us
          k += primz; <0.005us 0.000us
          while ( k <= SIZE ) <0.005us 0.000us
          {
            flags[ k ] = FALSE; <0.005us 0.000us
            k += primz; <0.005us 0.000us
            while ( k <= SIZE ) <0.005us 0.000us
            {
              flags[ k ] = FALSE; <0.005us 0.000us
              k += primz; <0.005us 0.000us
            }
          }
        }
      }
    }
}

```

Bottom status bar:

- B:: anzah = 2
- emulate trigger devices trace Data Var List PERF SYStem Step Go Break Register sYmbol FPU MMX MMU TRANSLation CACHE other previous
- C-T:-0000049653 -759.395ms | C-Z: 0.000 stopped MIX UP

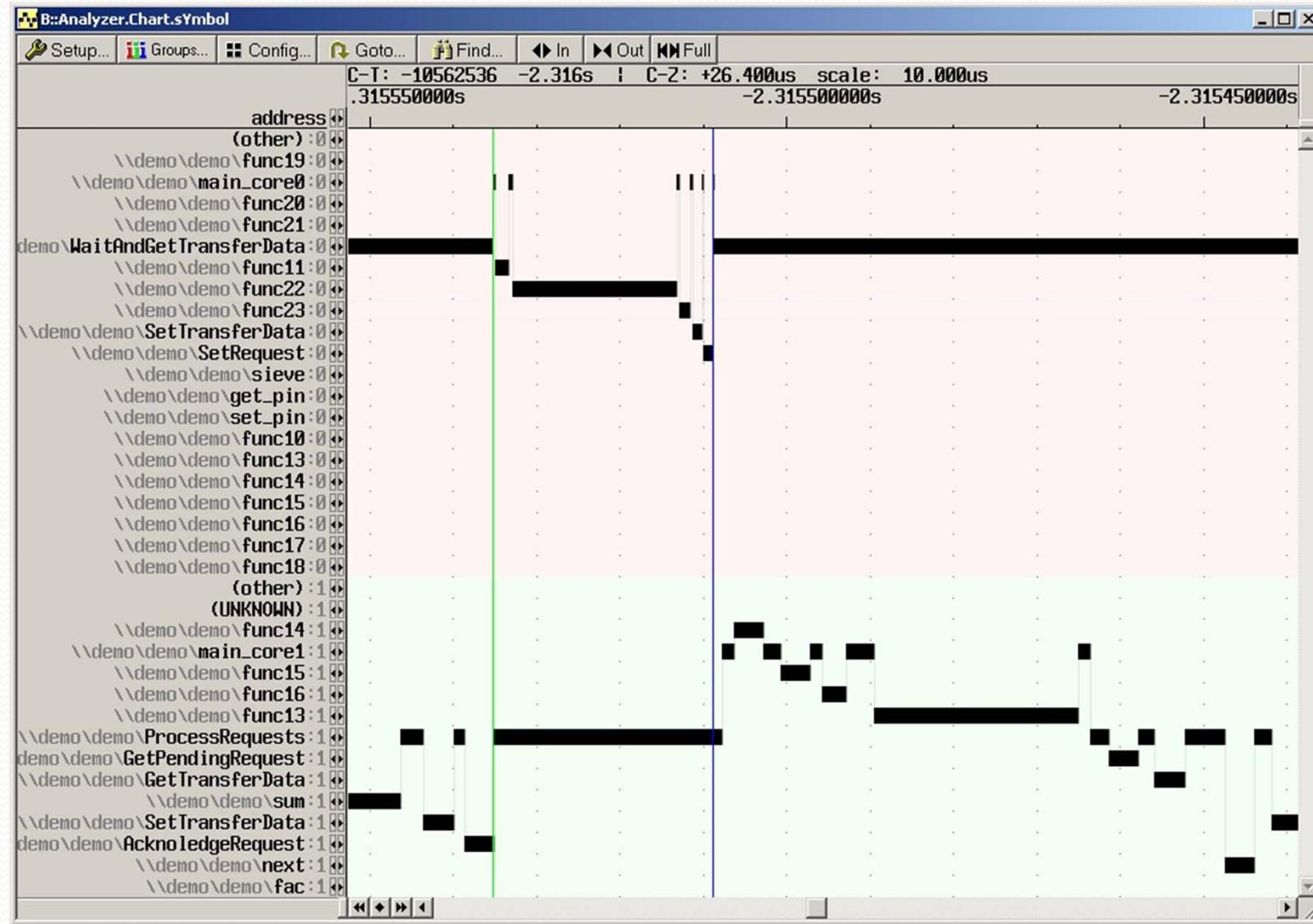
Real time trace: dual core program execution, data r/w history



dual core program execution, data r/w history: merged display

record	run	address	cycle	data	symbol	ti.back	ti.zero
-00000460	0	D:4000003C rd-long	00000003	\demo\Global\nRequestType		0.400us	-0.340us
	0	mbar 0x0		; 0			
	0	cmpwi r12,0x0		; r12,0			
	0	beq 0x8C8		; 0x8C8 (-)			
222	0	nPollCycles++;					
	0	addi r31,r31,0x1		; nPollCycles,nPollCycles,1			
	b	0x8B0					
-00000459	0	F:000008B0 ptrace		..mo\WaitAndGetTransferData+0x14	0.300us	-0.040us	
-00000458	1	F:00000314 ptrace		\demo\demo\func13	2.100us	0.000us	
221	0	while (nRequestType != 0)					
	0	lwz r12,-0x7F0(r13)		; r12,nRequestType(r13)			
-00000457	0	D:4000003C rd-long	00000003	\demo\Global\nRequestType	0.400us	0.360us	
-00000456	0	F:000008B0 ptrace		..mo\WaitAndGetTransferData+0x14	0.300us	0.660us	
1							
1							
1							
72	1	int func13(int a, int c, int e)		/* arguments and locals stack-tracking */			
	{						
	1	stwu r1,-0x20(r1)		; r1,-32(r1)			
-00000455	1	D:4000F758 wr-long	4000F778	\demo\Global__SP2_END+0x758	0.720us	0.720us	
	0	mbar 0x0		; 0			
	0	cmpwi r12,0x0		; r12,0			
	0	beq 0x8C8		; 0x8C8 (-)			
222	0	nPollCycles++;					
	0	addi r31,r31,0x1		; nPollCycles,nPollCycles,1			
	b	0x8B0					
-00000453	1	D:4000F760 wr-quad 000000070000002A \demo\Global__SP2_END+0x760			0.060us	0.780us	
-00000451	1	D:4000F768 wr-quad 0000000600000003 \demo\Global__SP2_END+0x768			0.060us	0.840us	
-00000449	1	D:4000F770 wr-quad 0000000200000001 \demo\Global__SP2_END+0x770			0.060us	0.900us	
221	0	while (nRequestType != 0)					
	0	lwz r12,-0x7F0(r13)		; r12,nRequestType(r13)			
-00000447	0	D:4000003C rd-long	00000003	\demo\Global\nRequestType	0.400us	1.060us	
	1	mflr r0					
	1	stwu r26,0x8(r1)		; f,8(r1)			
-00000446	0	F:000008B0 ptrace		..mo\WaitAndGetTransferData+0x14	0.300us	1.360us	
	1	stu r0,0x24(r1)		; r0,36(r1)			
-00000445	1	D:4000F77C wr-long	00000358	\demo\Global__SP2_END+0x77C	0.500us	1.400us	
	0	mbar 0x0		; 0			
	0	cmpwi r12,0x0		; r12,0			
	0	beq 0x8C8		; 0x8C8 (-)			
222	0	nPollCycles++;					
	0	addi r31,r31,0x1		; nPollCycles,nPollCycles,1			
	b	0x8B0					
221	0	while (nRequestType != 0)					
	0	lwz r12,-0x7F0(r13)		; r12,nRequestType(r13)			
	1	mr r31,r3		; a,r3			
	1	mr r30,r4		; c,r4			
	1	mr r29,r5		; e,r5			
	1	int b, d, f;					

Dual Core Function Runtime Analysis: Chart Display



Code Coverage Analysis: Source Code / Instruction Level

		code	label	mnemonic	comment
	coverage	addr/line			
never	423	void main_core1()			
never	SF:00000E58	9421FFF0	main_cor..:stwu	r1,-0x10(r1)	; r1,-16(r1)
never	SF:00000E5C	7C0802A6	mflr	r0	
never	SF:00000E60	93E1000C	stw	r31,0x0C(r1)	; nLoopCount,12(r1)
never	SF:00000E64	90010014	stw	r0,0x14(r1)	; r0,20(r1)
never	424		unsigned int nLoopCount = 0;		
never	SF:00000E68	3BE00000	li	r31,0x0	; nLoopCount,0
never	425		enable_timebase();		
never	SF:00000E6C	4BFFFC6D	bl	0xAD8	; enable_timebase
			while (1)		
			{		
ok	428			func13(1, 2, 3);	
ok	SF:00000E70	38600001	li	r3,0x1	; r3,1
ok	SF:00000E74	38800002	li	r4,0x2	; r4,2
ok	SF:00000E78	38A00003	li	r5,0x3	; r5,3
ok	SF:00000E7C	4BFFF499	bl	0x314	; func13
ok	430			ProcessRequests();	
ok	SF:00000E80	4BFFFB51	bl	0x9D0	; ProcessRequests
ok	432			func14((char) 66);	
ok	SF:00000E84	38600042	li	r3,0x42	; r3,66
ok	SF:00000E88	4BFFF4F1	bl	0x378	; func14
ok	434			func15((short) 66);	
ok	SF:00000E8C	38600042	li	r3,0x42	; r3,66
ok	SF:00000E90	4BFFF515	bl	0x3A4	; func15
ok	436			func16((long) 66);	
ok	SF:00000E94	38600042	li	r3,0x42	; r3,66
ok	SF:00000E98	4BFFF53D	bl	0x3D4	; func16
ok	438			nLoopCount++;	
ok	SF:00000E9C	3BFF0001	addi	r31,r31,0x1	; nLoopCount,nLoopCount,1
ok	439	}			
ok	SF:00000EA0	4BFFFFD0	b	0xE70	
never	440	}			
never	SF:00000EA4	83E1000C	lwz	r31,0x0C(r1)	; nLoopCount,12(r1)
never	SF:00000EA8	80010014	lwz	r0,0x14(r1)	; r0,20(r1)
never	SF:00000EAC	7C0803A6	mtlr	r0	
never	SF:00000EB0	382410010	addi	r1,r1,0x10	; r1,r1,16
never	SF:00000EB4	4E800020	blr		

Trace history based debugging: Context Tracking System

The screenshot shows a debugger interface with four windows:

- B:CTS.List**: Shows a list of trace records. Record 72 is selected, displaying code from `func13` and local variable values (a=37, c=52, e=0). Other records show loops and variable assignments.
- B:Register**: Registers window showing CPU register values. For example, R0 = 0D48, R1 = 40000030, etc.
- B:Var.View %oe nRequestType nTransferData**: Variables window showing `nRequestType = 1` and `nTransferData = 0`.
- B:Var.Frame /Locals /Caller**: Call stack frame window showing the execution flow between `WaitAndGetTransferData()` and `main_core0()`.

Calibration and Rapid Prototyping

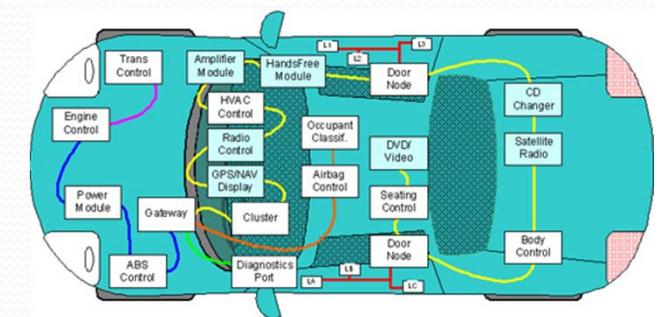
Todd Collins, ETAS



A PROGRAM OF THE IEEE
INDUSTRY STANDARDS AND
TECHNOLOGY ORGANIZATION

Use of control systems in vehicles

- 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

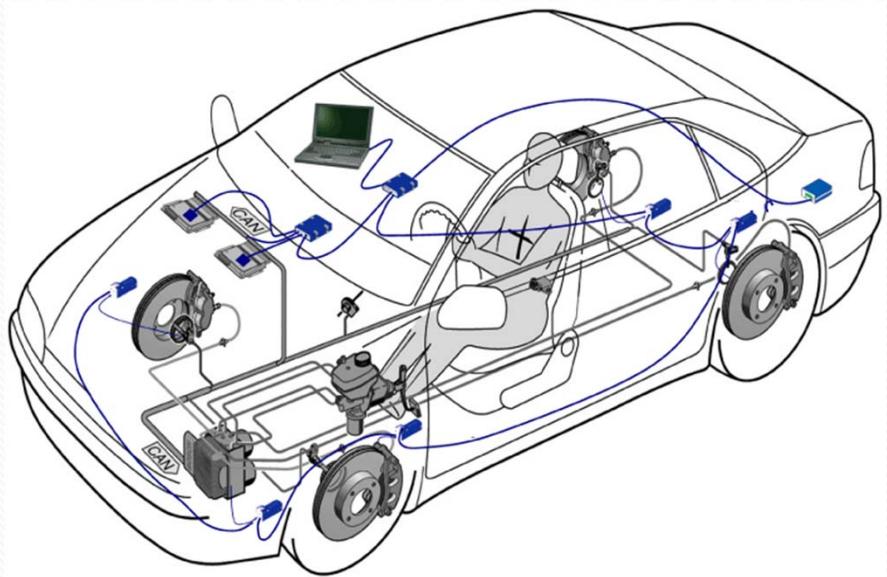


Vehicle Calibration

- 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 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...)

Vehicle Calibration Cont'd

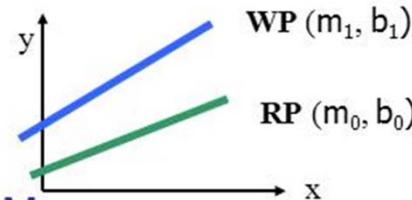
- Availability of ECU parameters/variables in real time to the calibration engineer is an essential aspect of a calibration system
- During calibration, ECU parameter measurement is also essential, as thousands of variables can be simultaneously measured. Some hybrid and vehicle control systems require high speed measurements of variables/parameters changing faster than $50\mu\text{s}$



Simple calibration example

Calibrated function:

$$y = m * x + b$$



Flash

Data

Reference Page
Parameter m_0
Parameter b_0

Code

RAM

Data

Working Page
Parameter m_1
Parameter b_1

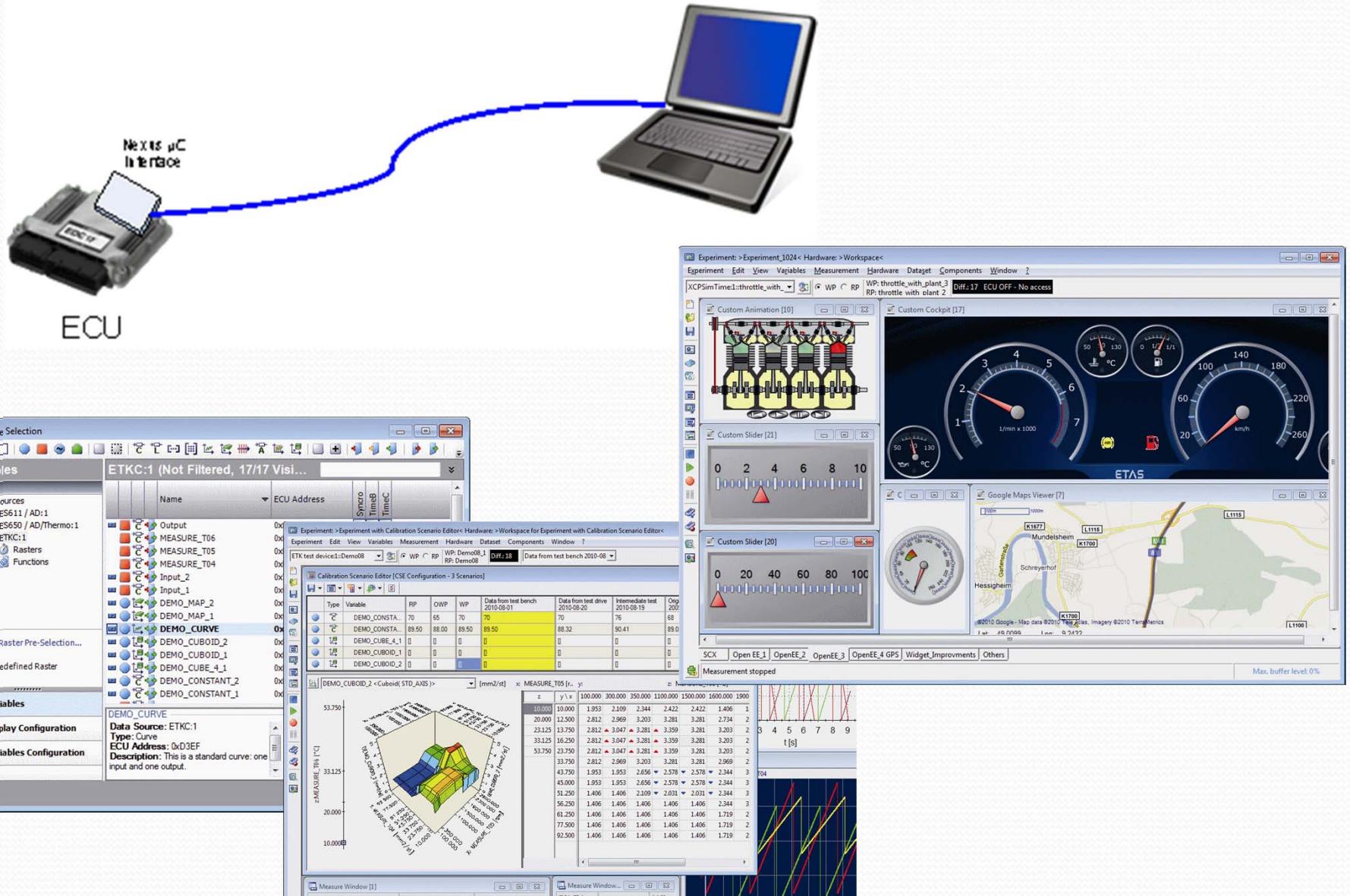
Code

ECU Running on
Reference Page
RP

Page switch

Working Page
WP

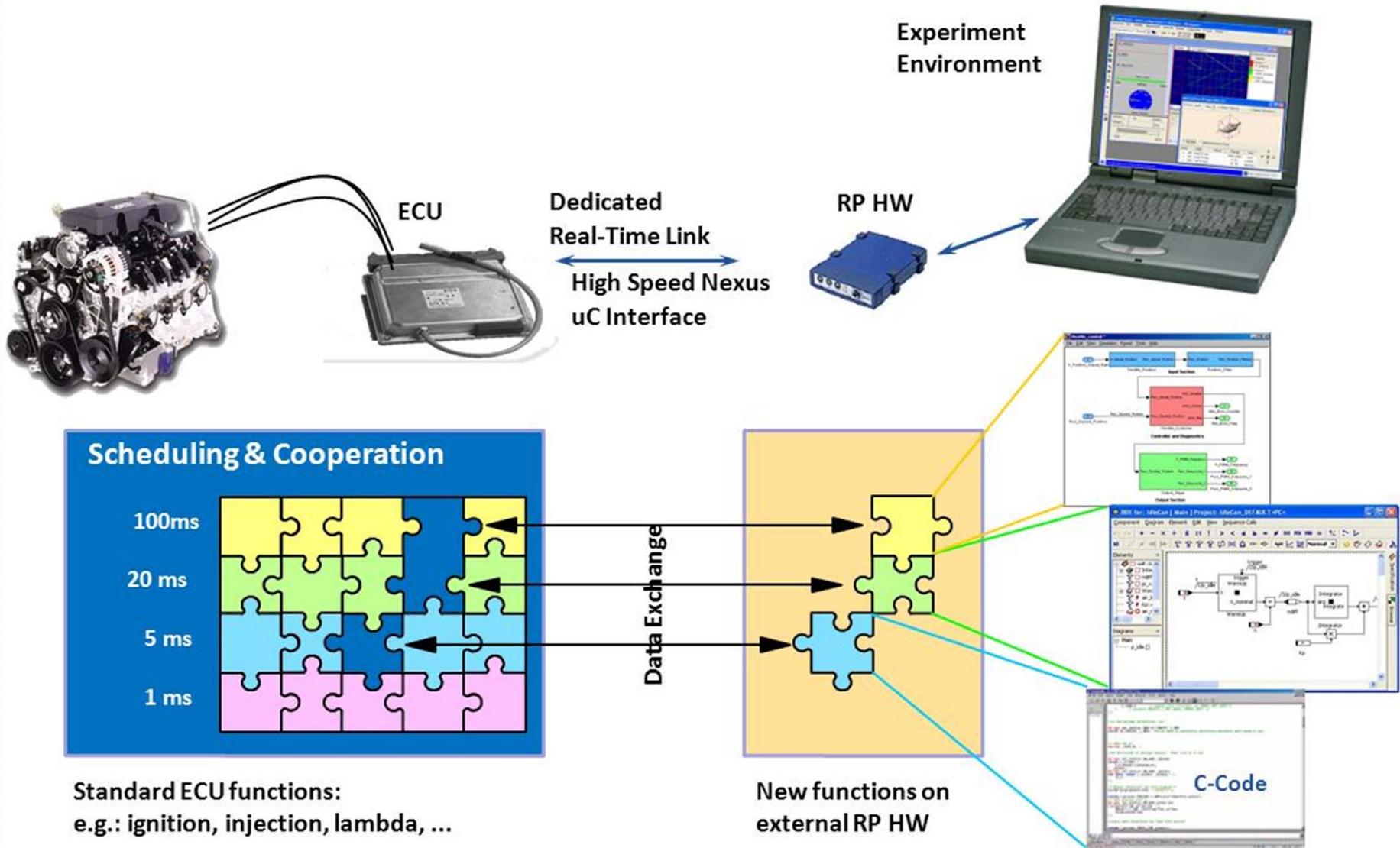
Use of Nexus in vehicle calibration



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

Rapid Prototyping example



Use of Nexus interface in Calibration & RP

- 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

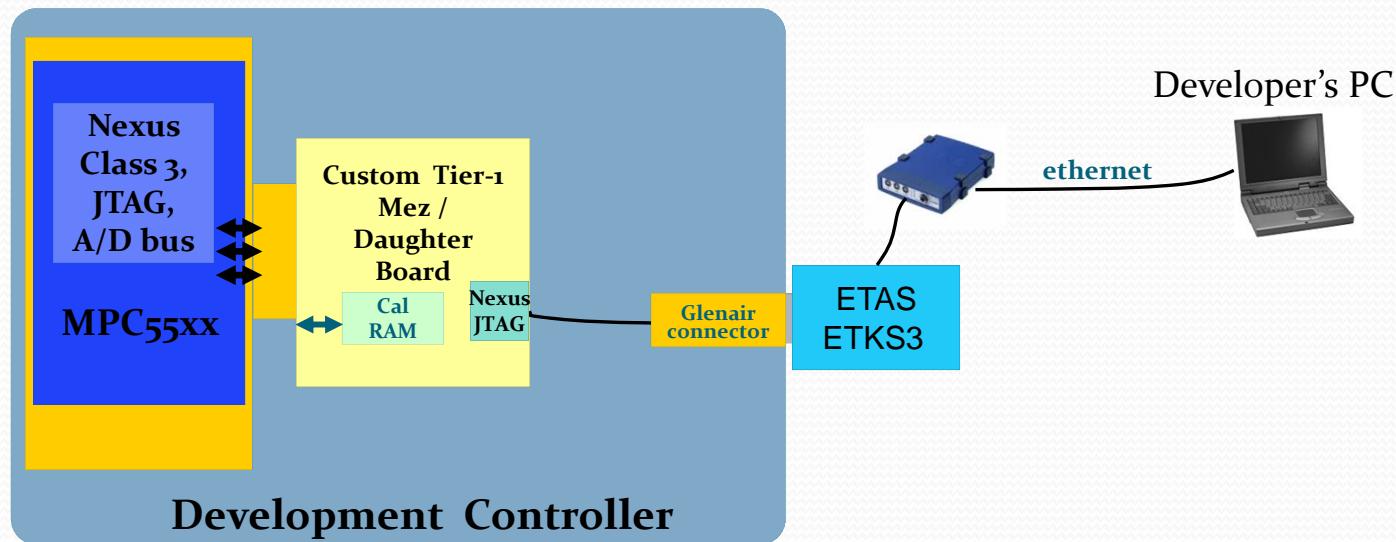
- There are 3 primary use cases employed at GM in the development of ECU software and calibration
 - Calibration: Measurement and calibration tools
 - Software Development: Software debugging / trace tools
 - Feature Development: Rapid prototyping tools
- The next few slides illustrate how these tools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

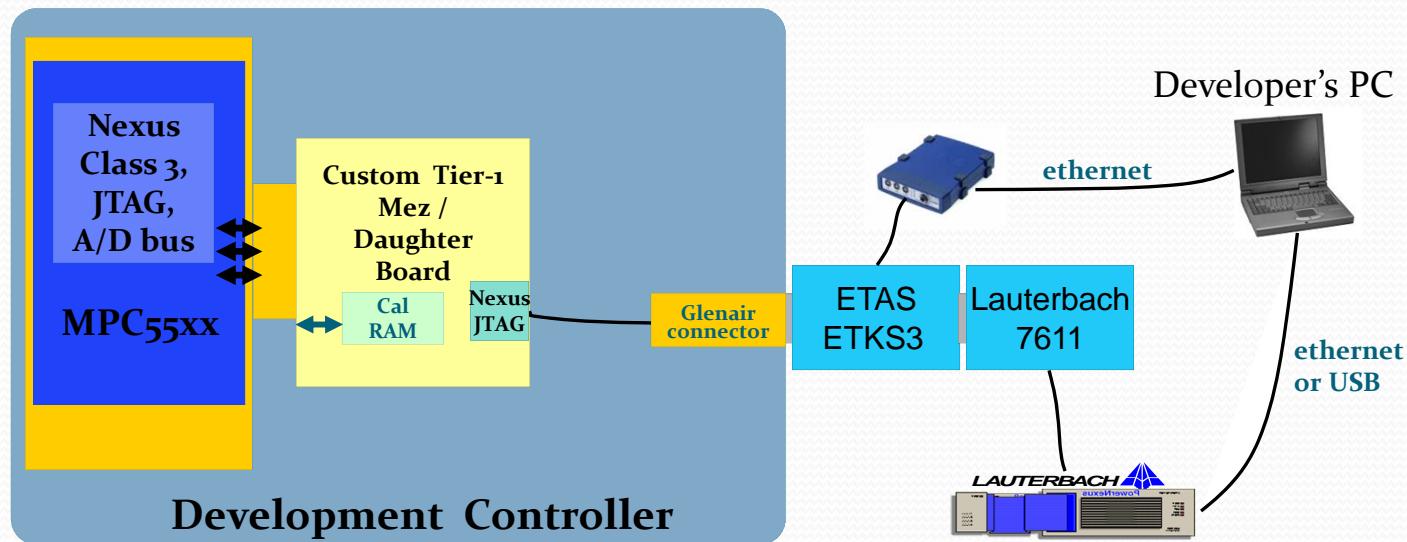
Gen 2 Implementation

- Measurement & Calibration



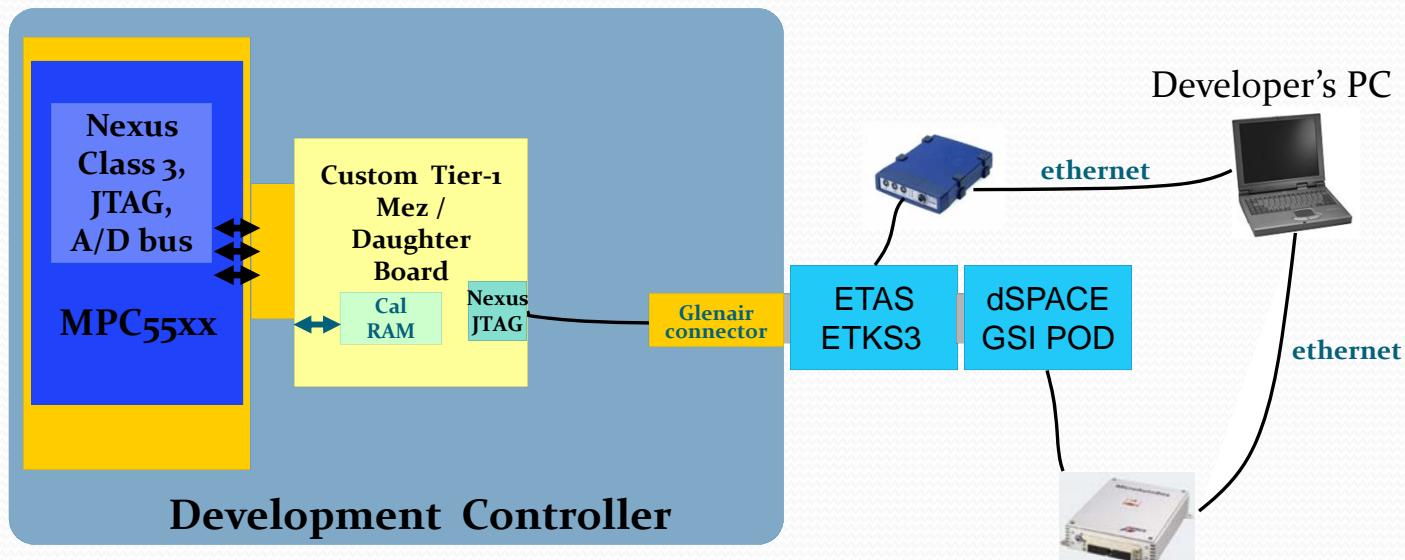
Gen 2 Implementation

- Measurement & Calibration with Debug & Trace



Gen 2 Implementation

- Measurement & Calibration with RCP

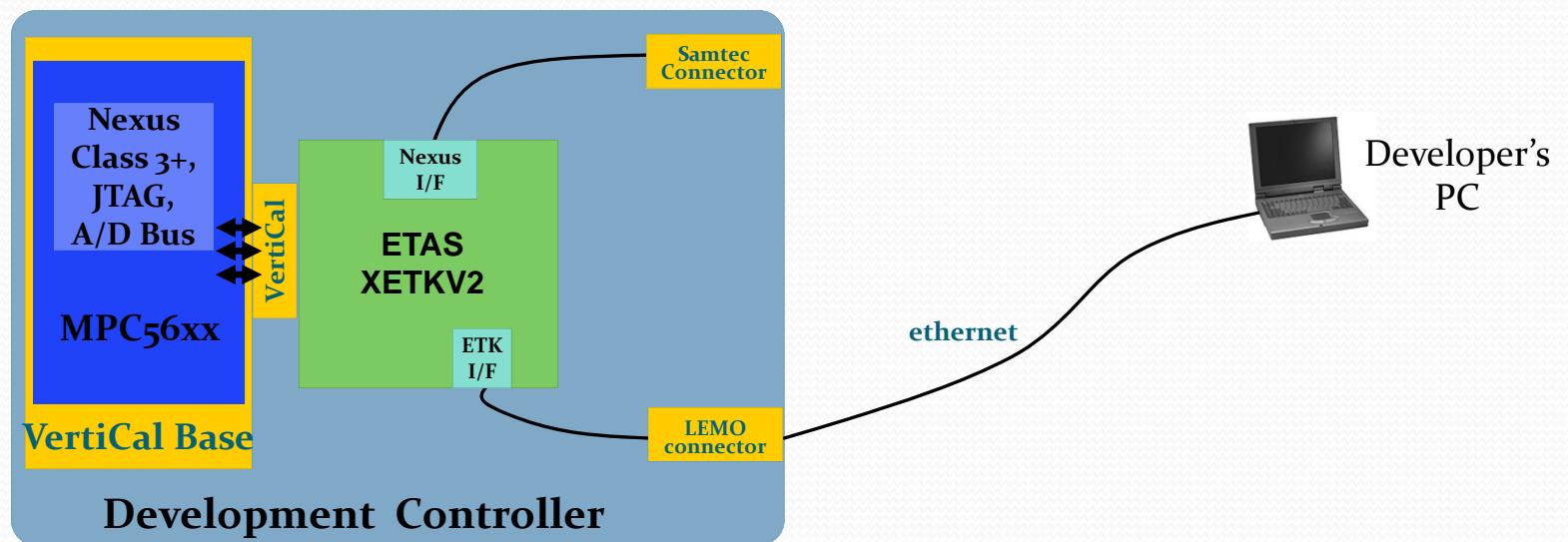


Gen 3 Overview

- 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

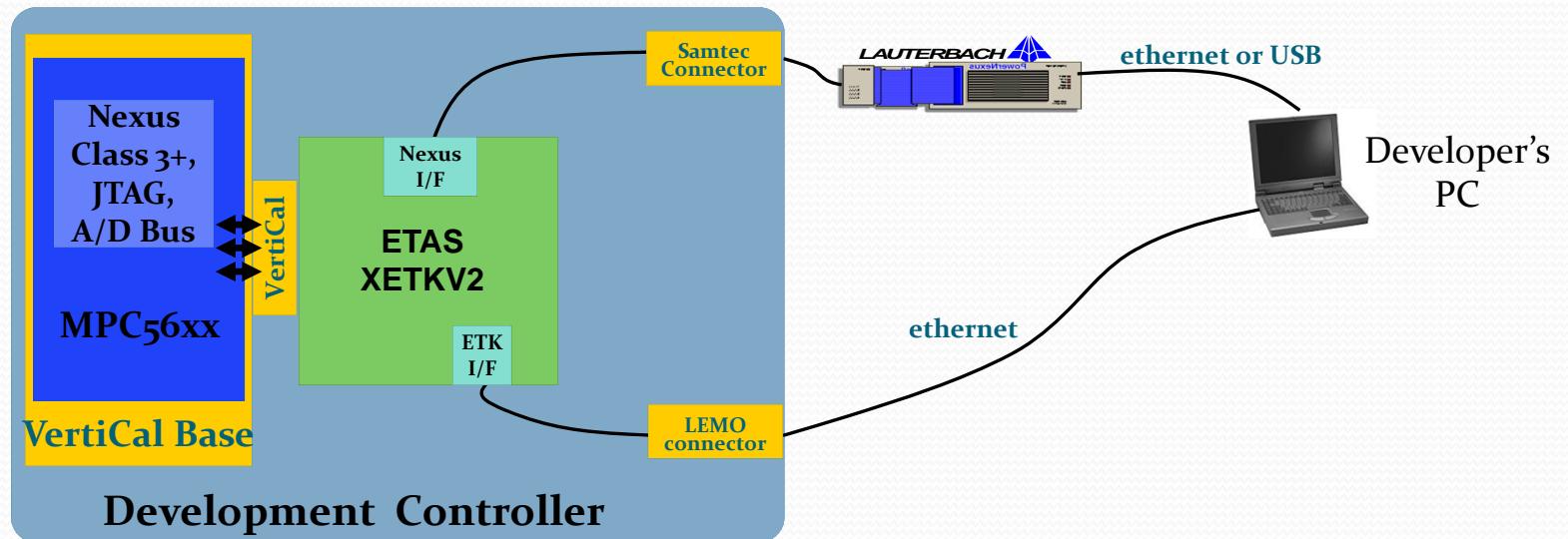
Gen 3 Implementation

- Measurement & Calibration



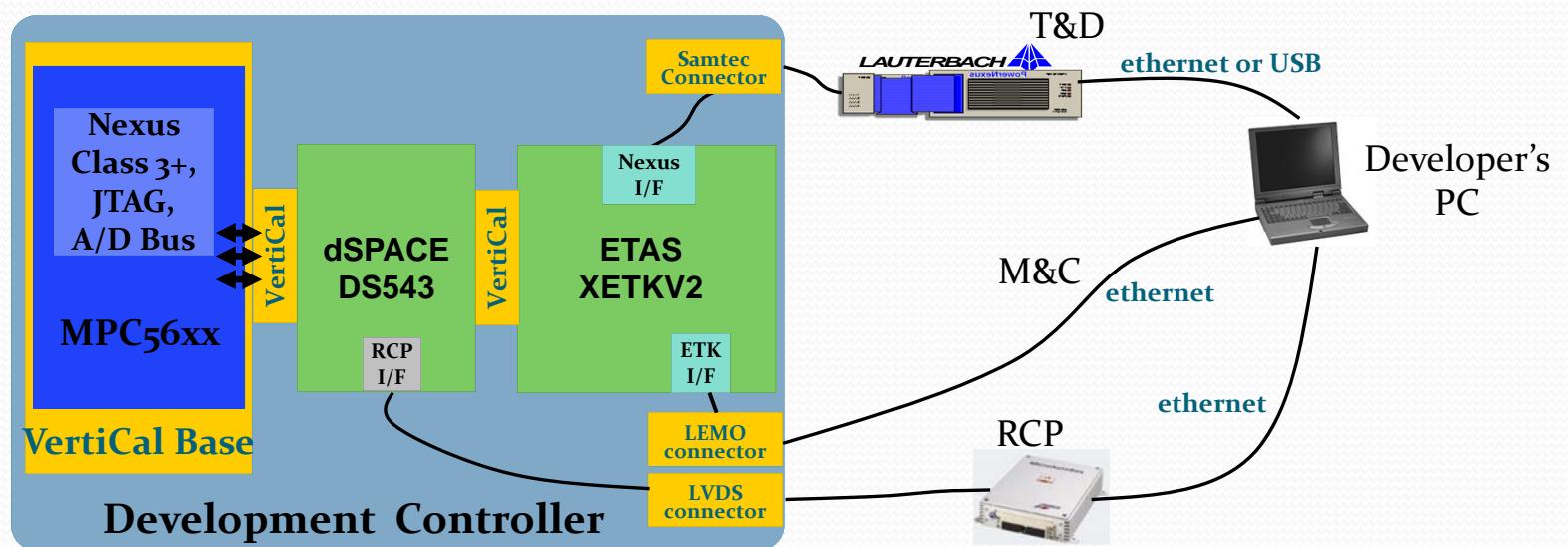
Gen 3 Implementation

- Measurement & Calibration with Debug & Trace



Gen 3 Implementation

- Measurement & Calibration, Debug & Trace, and RCP



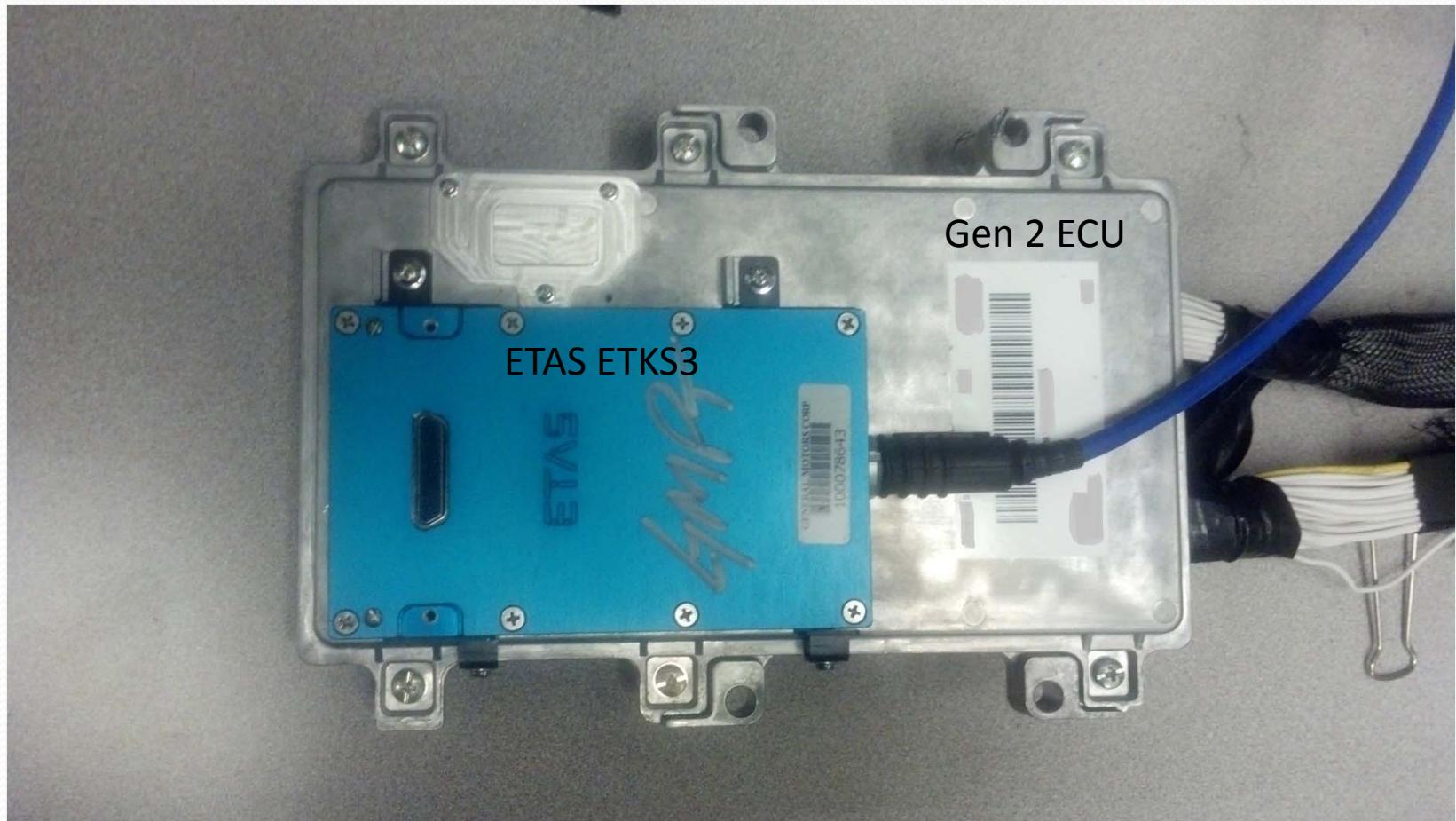
Tools in Use

Norm D'Amico, General Motors

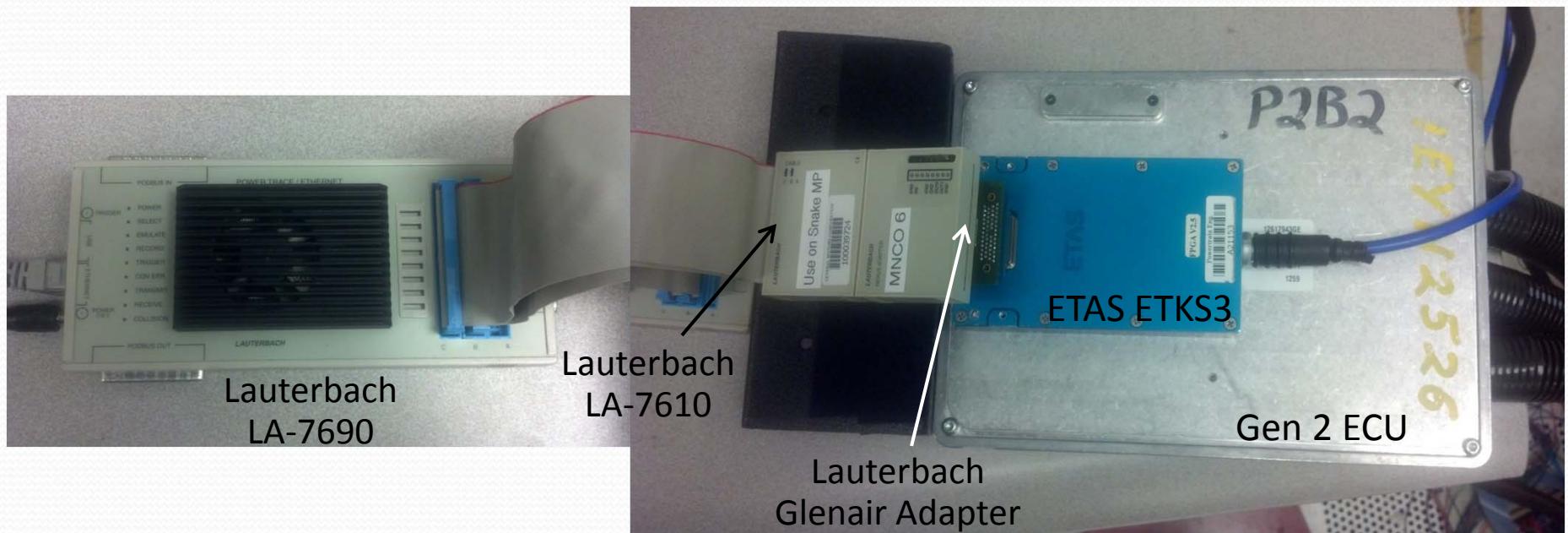


66
A PROGRAM OF THE IEEE
INDUSTRY STANDARDS AND
TECHNOLOGY ORGANIZATION

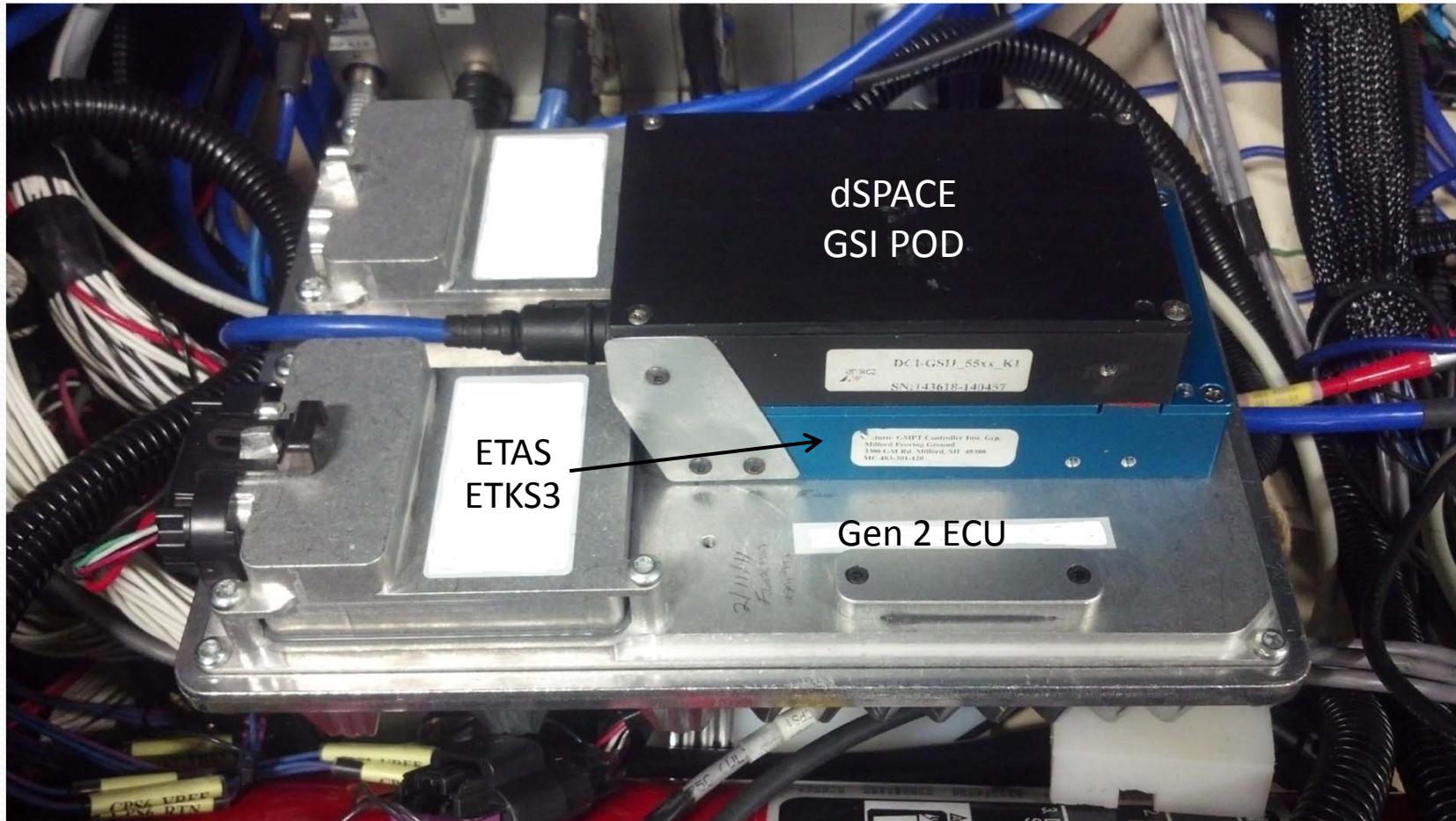
Gen 2 Measurement & Calibration



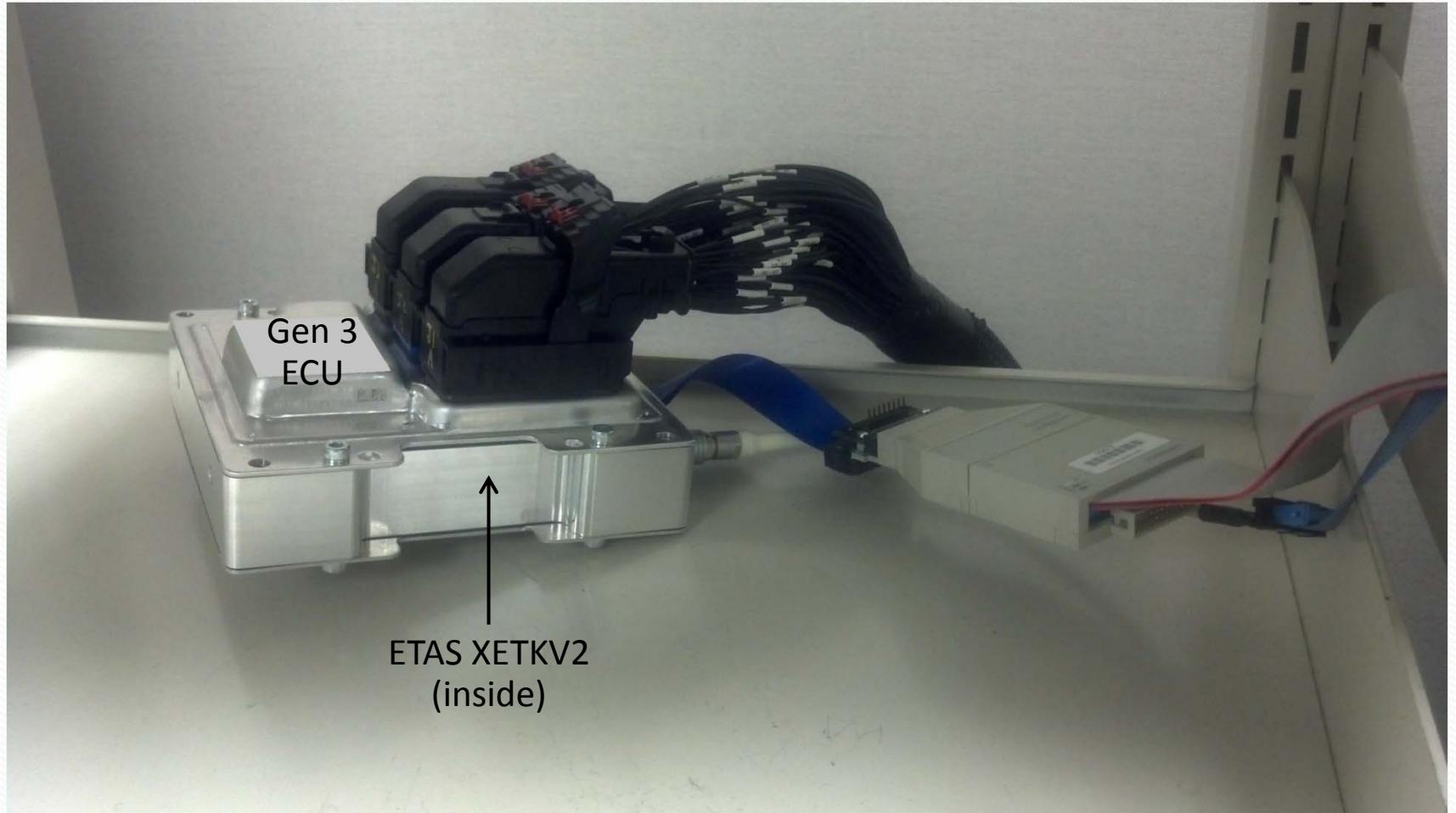
Gen 2 Measurement & Calibration with Trace & Debug



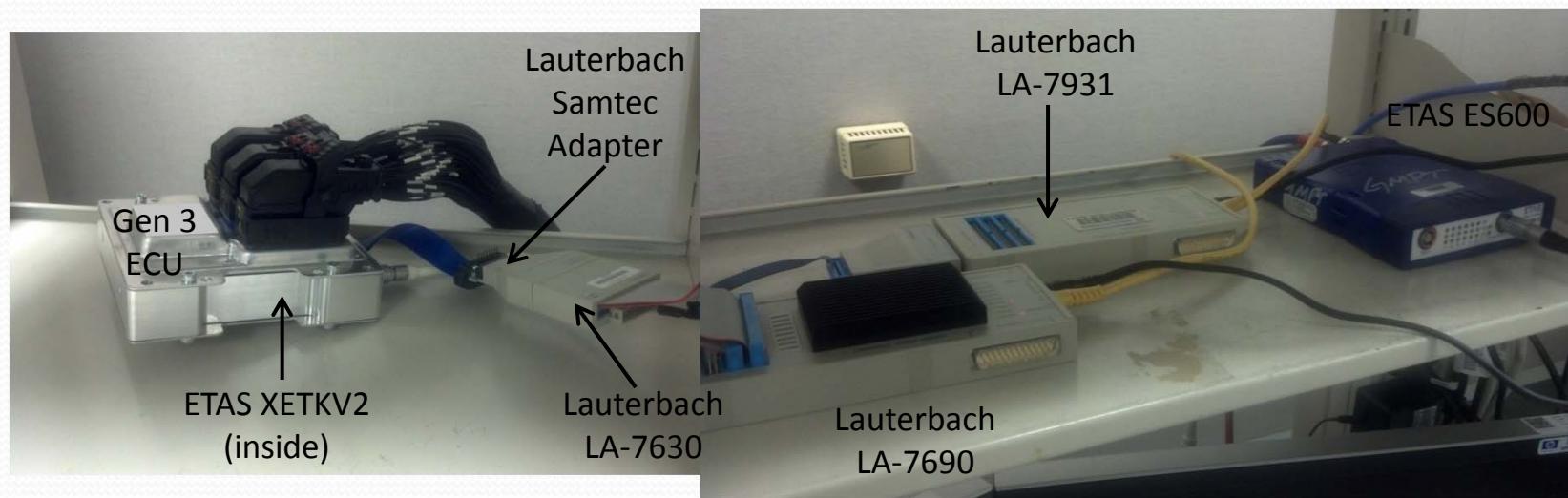
Gen 2 Measurement & Calibration with RCP



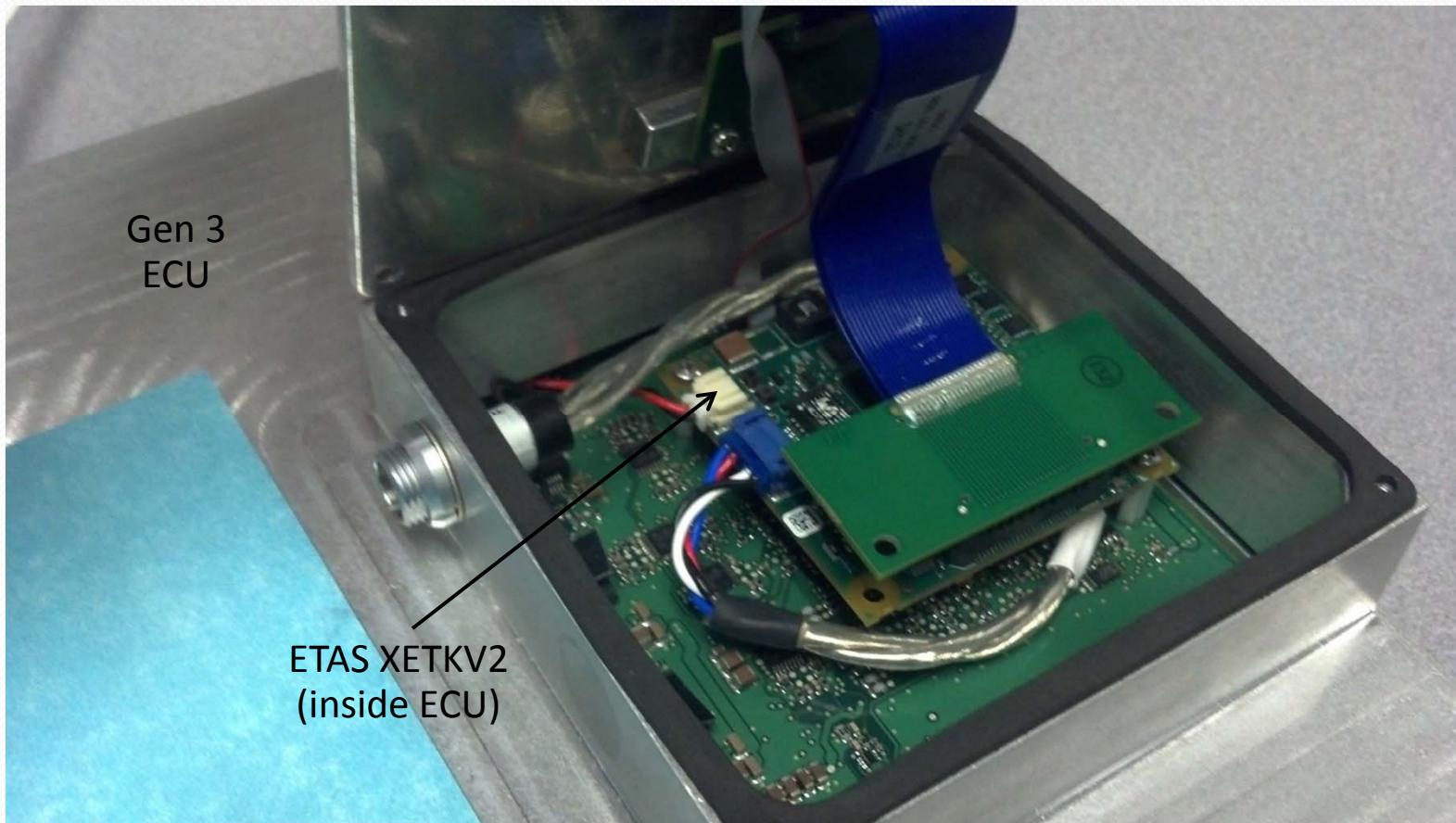
Gen 3 Measurement & Calibration



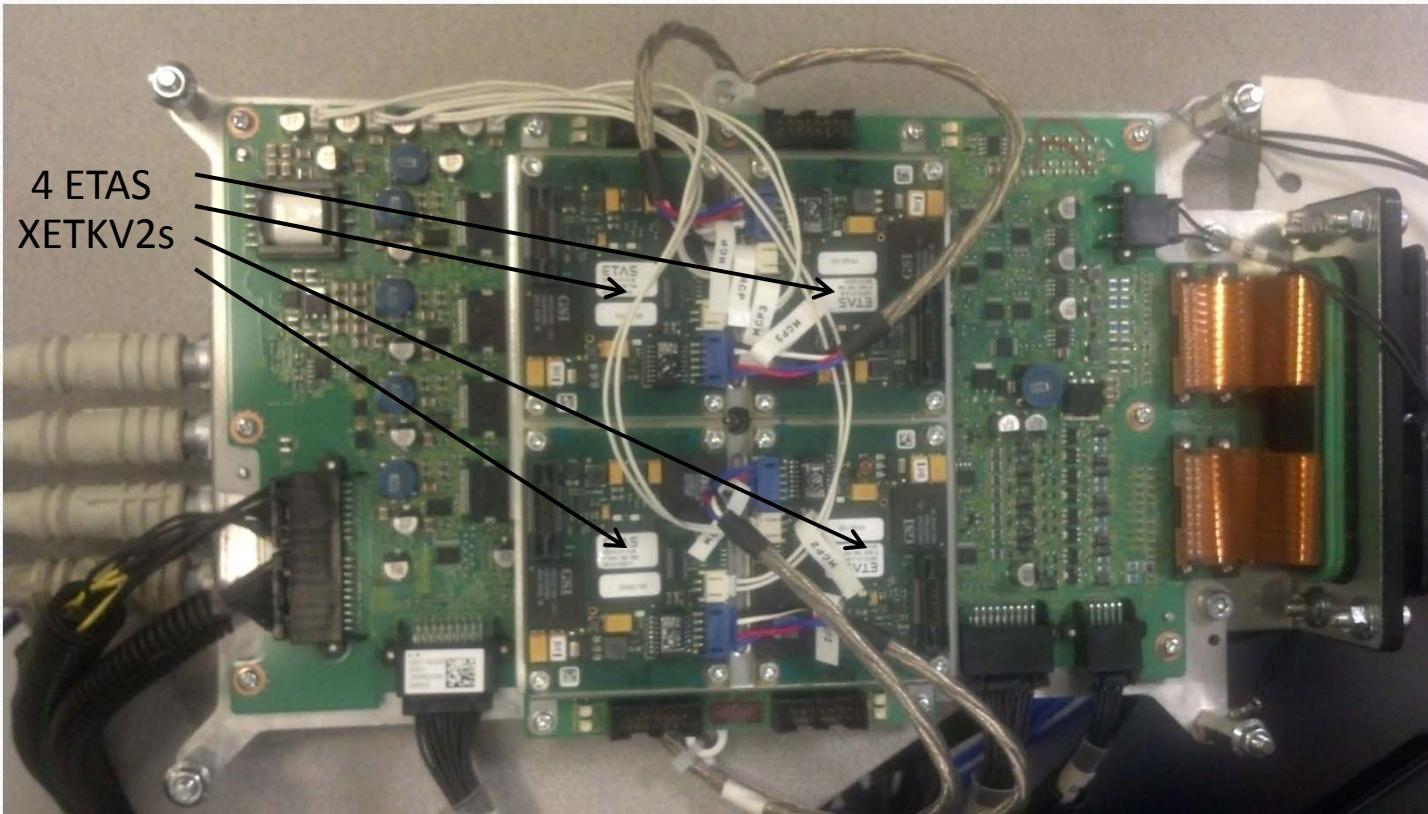
Gen 3 Measurement & Calibration with Trace & Debug



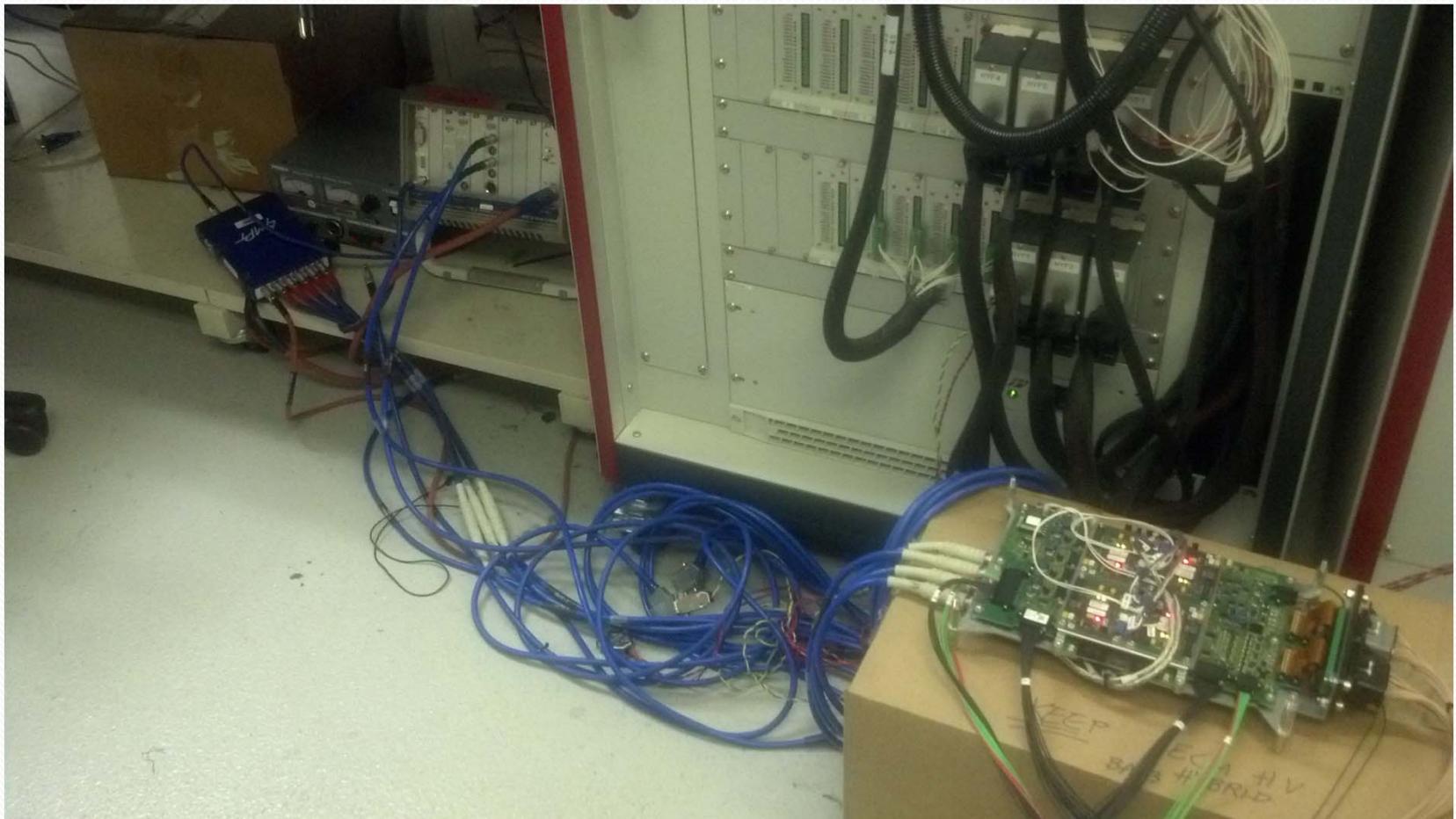
Gen 3 Internal Instrumentation (XETKV2)



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



A PROGRAM OF THE IEEE
INDUSTRY STANDARDS AND
TECHNOLOGY ORGANIZATION

Thank You!

For more information on Nexus 5001 Forum, please visit:

www.nexus5001.org

