FTF | FREESCALE TECHNOLOGY FORUM 2015

Performance Monitoring
for Automotive MCUs

FTF-ACC-F1184

Randy Dees | Automotive MCU Applications

J U N E . 2 0 1 5

External Use

Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Magniv, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Qorivva, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid and Xtrinsic are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Airfast, BeeKit, BeeStack, CoreNet, Flexis, Layerscape, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink and UMEMS are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © 2015 Freescale Semiconductor, Inc.

Abstract

- As microcontrollers become more complex, the use of debug features becomes more important. While Nexus trace features can be used for many types of application validation, the e200zx cores on the MPC57xx devices also include a performance monitor that allows additional capabilities
- This session will cover an overview of traditional system validation use cases and includes an in-depth view of the performance monitor including features, instructions, registers, overflow operation, and predefined events that can be used for a more robust system validation

Agenda

- Session Overview
- Nexus Overview
- Nexus and Debug Use Cases
- Performance Monitor Overview
- Performance Monitor Instructions
- Performance Monitor Registers
- Performance Monitor Events
- Performance Monitor Use Cases
- Summary

Session Overview

Agenda

- **Session Overview**
- Nexus Overview
- Nexus and Debug Use Cases
- Performance Monitor Overview
- Performance Monitor Instructions
- Performance Monitor Registers
- Performance Monitor Events
- Performance Monitor Use Cases
- Summary

Automotive Market Trends

- **Exponential MCU performance demands**
  - Need 3–5x increase in performance to support software and emission filtering
  - Consolidation of external ASICs into MCU
  - On-chip DSP to meet tighter emission regulations
  - High speed inter-processor communications, tightly coupled memory and Nexus Aurora allows 4x faster debug/trace capability
- **Emerging safety and security standards**
  - Enable ASIL-C or ASIL-D functional safety
    - Simplify ISO 26262 adoption
  - Tamper detection and encryption to deter code tampering

Automotive Requirements

- Traditionally, Powertrain MCUs have required the most advanced debug capabilities, but this is changing
- "Calibration" features (data collection and application tuning) are becoming required across the board
  - RADAR applications require real-time access to data for system confirmation
  - All motor control applications
  - Lighting controllers
  - Even Gateway type applications are using debug features for "tuning" of the system
- Software validation and qualification requires debug/trace capabilities. Some specialized testing requires the use of the core performance monitors
- All of the debug features are required to help insure safety systems (ISO 26262)

ISO 26262 — Introduction to the Basics

- ISO 26262 is the new *functional safety standard* for series production passenger cars
- It is applicable to electric / electronic systems where malfunctioning behavior of such systems can cause harm
- The standard focuses on systems ("items") consisting of sensors, microprocessors and actuators
- Four Automotive Safety Integrity Levels (ASIL A to D) determined through hazard analysis and risk assessment at vehicle level
- The ASILs imply a specific set of requirements and safety measures to be applied for avoiding an unreasonable residual risk
- Smart microprocessor solutions & collateral can significantly reduce the effort required to build functional safe systems complying with ISO 26262

NXP

The Four Pillars

SAFE ASSURE™ by Freescale

Safety process

- Integrating functional safety into product development process
- Select products defined and designed from the ground up to comply with the standards

Safety hardware

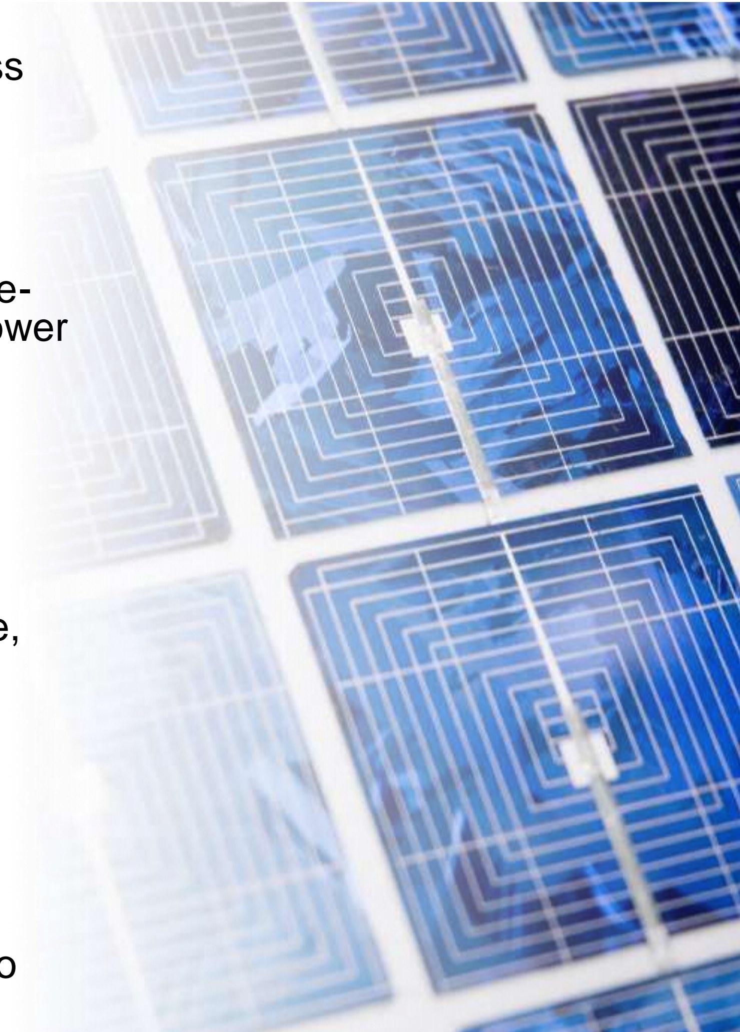
- Built-in safety functions (self-testing, monitoring and hardware-based redundancy) in Freescale microcontrollers (MCUs), power management ICs and sensors
- Additional system-level safety functionality from Freescale analog solutions (checking MCU timing, voltages and error management)

Safety software

- A comprehensive set of automotive functional safety software, including AUTOSAR OS and associated microcontroller abstraction layer (MCAL) drivers, as well as core self-test capabilities
- Partnerships with leading third-party software providers for additional safety software solutions

Safety support

- From customer-specific training and system design reviews to extensive safety documentation and technical support



Objective

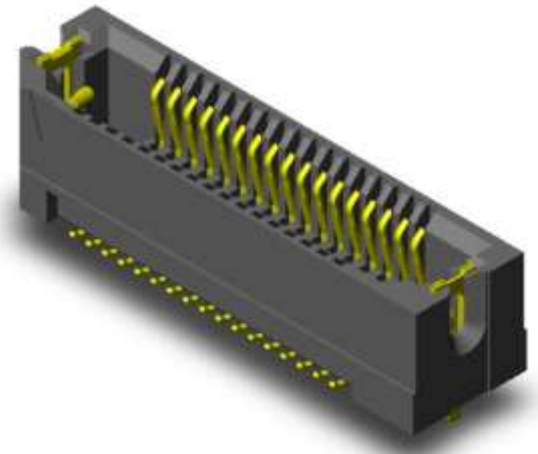
Use the Performance Monitor included in many of the e200z*x* cores to extend the system development capabilities provided by the Nexus interface on the Freescale Automotive Power Architecture® Microcontrollers.

Nexus Overview

Agenda

- Session Overview
- **Nexus Overview**
- Nexus and Debug Use Cases
- Performance Monitor Overview
- Performance Monitor Instructions
- Performance Monitor Registers
- Performance Monitor Events
- Performance Monitor Use Cases
- Summary

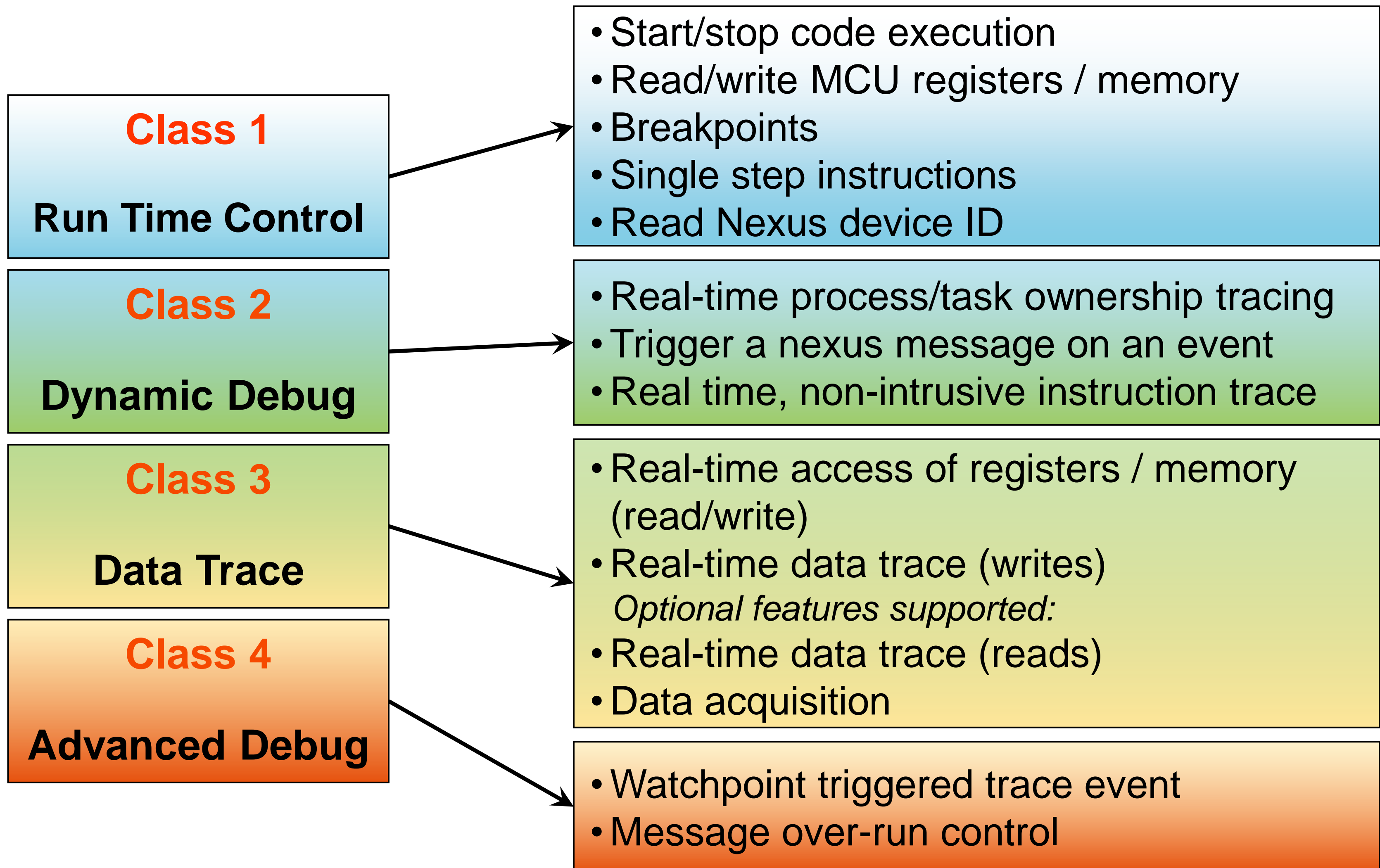
Multi-Core Debug Overview



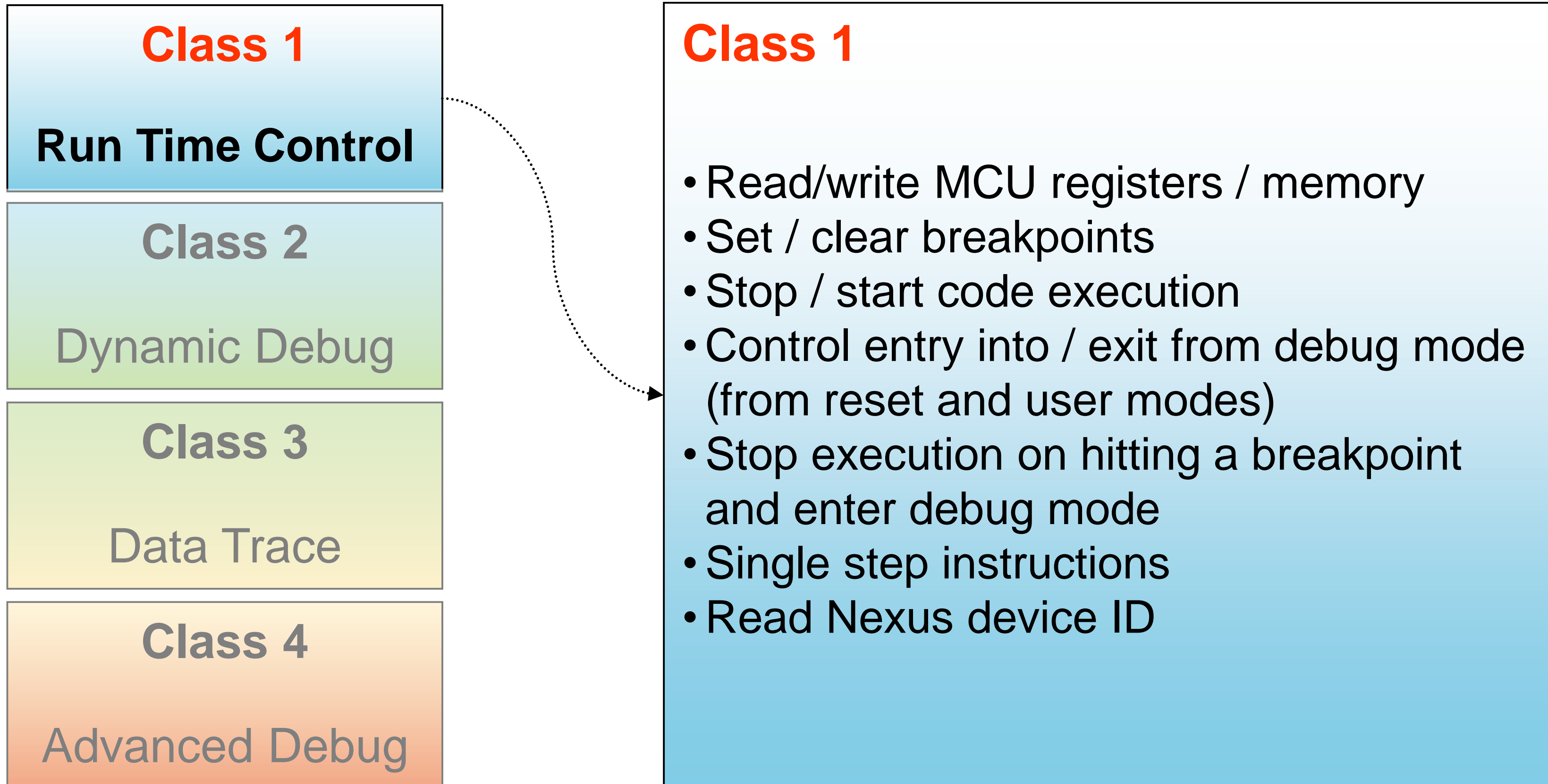
- Single debug interface for run control, trace, and measurement data
- Supports IEEE 1149.1 (JTAG) and IEEE 1149.7 (cJTAG) for run control
  - Freescale uses TAP sharing on the Automotive Power Architecture MCUs to allow access to all internal clients (processor cores, timer cores, and other clients)
- Trace based on the IEEE-ISTO 5001 Nexus debug standard
  - Nexus inherently supports multi-core debug information
  - Class 2+ or Class 3+ support on most clients
  - Parallel or high-speed Serial solutions depending on requirements

Current Nexus Members

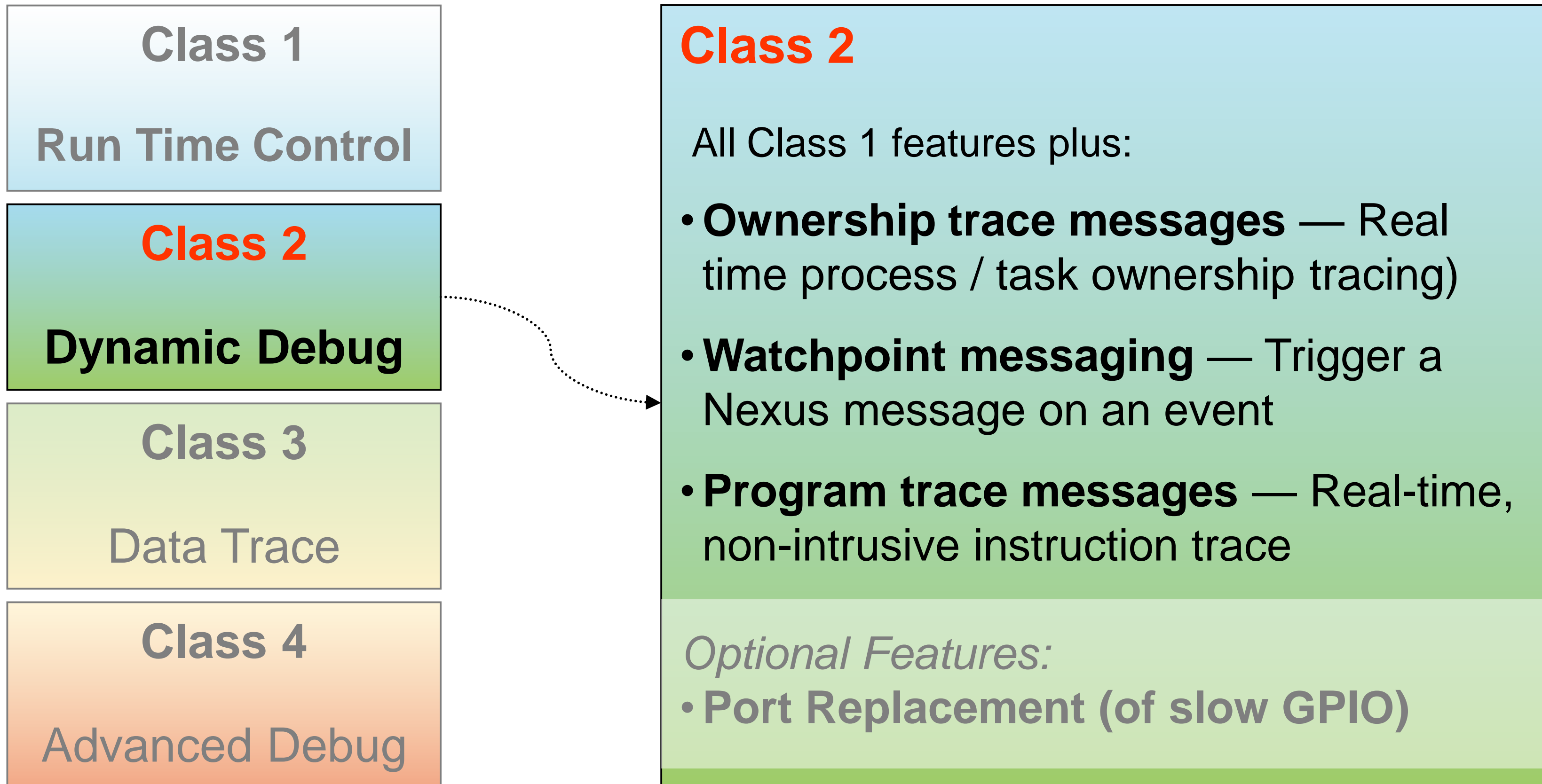
IEEE-ISTO 5001 Nexus Classes (MPC57xx Support)



Nexus Class Definition — Class 1

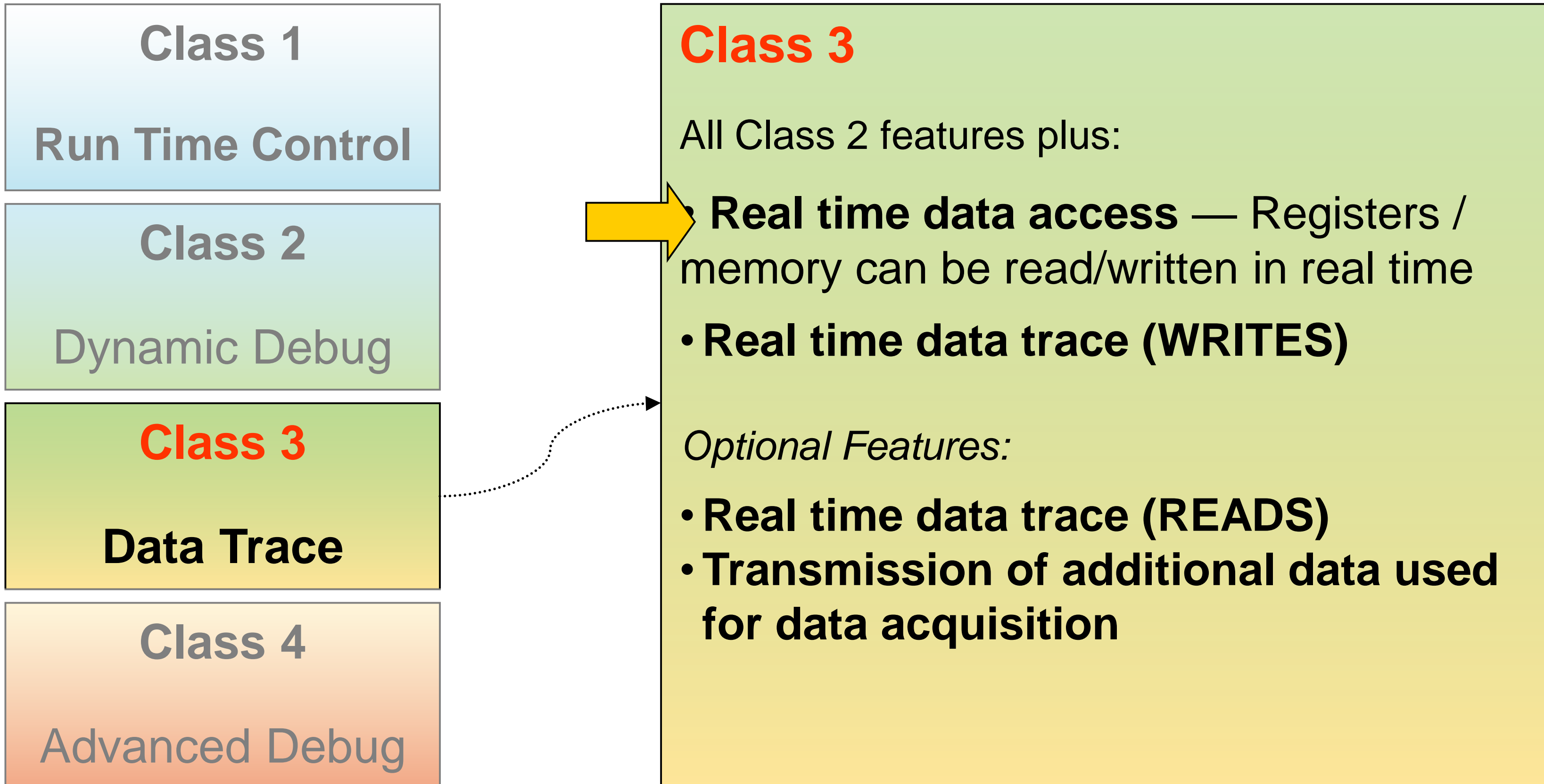


Nexus Class Definition — Class 2

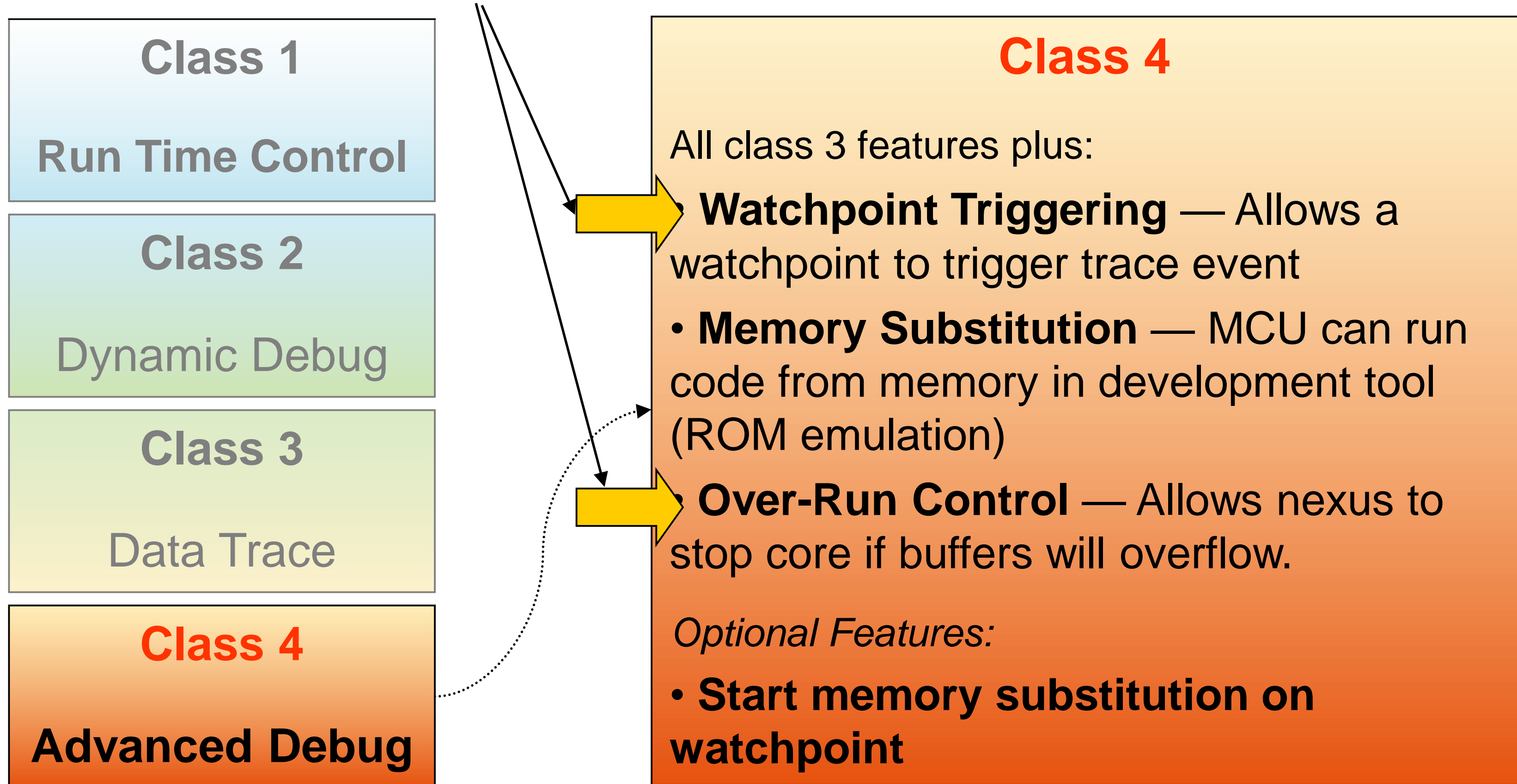


Nexus Class Definition — Class 3

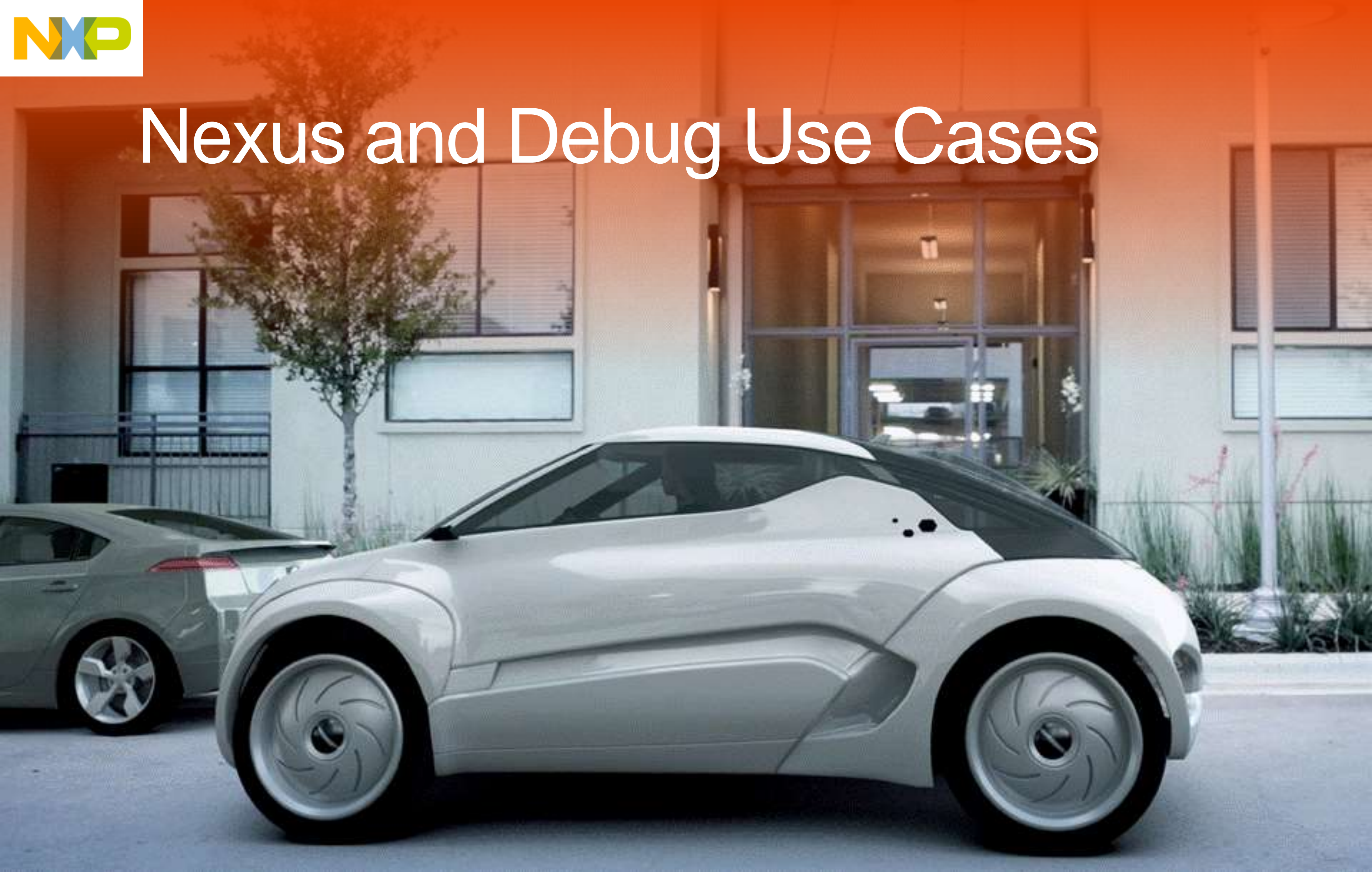
Features that Freescale supports on many automotive devices that support only Class 2



Nexus Class Definition — Class 4



Nexus and Debug Use Cases



Agenda

- Session Overview
- Nexus Overview
- **Nexus and Debug Use Cases**
- Performance Monitor Overview
- Performance Monitor Instructions
- Performance Monitor Registers
- Performance Monitor Events
- Performance Monitor Use Cases
- Summary

Customer Debug Use Cases

Debug uses cases can be broken down into 2 basic types of debug feature requirements.

Static debug: Static debug is also called stop-mode debug. These types of features allow debug of an MCU by stopping and starting core operation by an external tool.

Dynamic debug: Dynamic debug require features that allow real-time access of the MCU and real-time monitoring of internal MCU operations while the MCU is in operation.

Customer uses cases can then be traced back to the Nexus classes and features.

Static Debug Use Cases

- **Basic hardware debug**
  - The standard JTAG boundary scan is used for the testing on the initial board design, board bring-up, and then production board testing
- **Boot Loader development**
  - Development of boot code often requires stop mode debugging
- **Driver development**
  - In new system development, peripherals require drivers to be developed and tested. This will use many static debug features
- **Application crash analysis**
  - The use of breakpoints and other debug features allow analysis of unexpected program failures that result in exceptions that were not anticipated

Static Debug Features

Use Case	JTAG	Nexus Class 1						
	BSDL	Read/write registers	Read/write memory	Enter debug from reset	Enter debug	Exit debug mode	Single step instruction	Breakpoint
Basic hardware debug	✓							
Board test	✓							
Bootloader development		✓	✓	✓	✓	✓	✓	✓
Driver development		✓	✓	✓	✓	✓	✓	✓
Application crash analysis		✓	✓		✓	✓	✓	✓

Customer Dynamic Debug Use Cases

- **Real Time defect debug**
  - Many times to resolve software issues, advanced debug techniques are required to determine the source of erroneous program execution. This requires the use of program trace, data trace, watchpoint messaging and sometimes even real-time access of memory to determine the source of the problem
- **Execution (code) coverage**
  - For software quality, code coverage testing is required to determine which program code was executed and which program code was not executed. This also uncovers whether all conditions of conditional instructions are executed
- **Execution profiling**
  - Execution (time) profiling can be used to can be used to determine performance bottlenecks at the source code level. This information can be used to decide whether functions require better optimization, better algorithms, or if techniques such as in-lining would increase the performance of the system
- **Function Profiling**
  - Function profiling is similar to the overall execution profiling, but looks at each function individually

Customer Dynamic Debug Use Cases

- **RTOS profiling**
  - Monitoring of the performance of the Real-time operating system is required to identify sequences of tasks, interrupt service routines or other operating system tasks to assist the software designer in determining optimum task and interrupt priorities and to assist in allocating tasks, functions, or interrupt servicing between cores in a multi-core environment
- **Application data collection**
  - Many user applications require that data be collected from the application code. This can then be used to analyze the system operation and environmental performance criteria
- **Shared resource profiling**
  - Data areas (variables) may be shared by multiple cores in an MCU or by different functions. In some cases, synchronized access is required. Therefore, it is necessary to be able to measure the use of these shared resources to determine is system performance is degraded by a core or process waiting for data to become available

Dynamic Debug Features

1. Optional
2. Currently being used.
3. Only the address portion of the data trace is required, the data values are not needed.
4. Data trace of performance monitor data (if available).

Use Case	Nexus Class 2			Nexus Class 3		Optional		Notes
	Program Trace	Watchpoint messaging	Ownership trace	Data Trace	Real time memory access	Data Acquisition (Instrumentation)	Timestamps	
Real Time defect tracking	✓	✓	✓	✓	✓			
Execution (code) coverage	✓							
Execution profiling	✓						✓	Core performance monitor can provide additional information.
Function profiling	✓						✓	Core performance monitor can provide additional information.
RTOS profiling			✓[1]	✓[2]		✓[1]	✓	
Application data collection				✓[3]		✓	✓	
Shared resource profiling				✓			✓	
Cache profiling	✓							Core performance monitor
IRQ latency profiling		✓						
Core loading	✓			✓[4]				Core performance monitor

Performance Monitor Overview

Agenda

- Session Overview
- Nexus Overview
- Nexus and Debug Use Cases
- **Performance Monitor Overview**
- Performance Monitor Instructions
- Performance Monitor Registers
- Performance Monitor Events
- Performance Monitor Use Cases
- Summary

Concept

The e200zx core performance monitor provides the ability to count predefined events and processor clocks associated with particular operations, such as cache misses, miss-predicted branches, or the number of cycles an execution unit stalls.

Goals

- Improve system performance by monitoring software execution and then recoding algorithms for more efficiency. For example:
  - Memory hierarchy behavior can be monitored and analyzed to optimize task scheduling or data distribution algorithms

- Help system developers bring up and debug their systems
- Allow statistics on application code in production environment

Performance Monitor Features

- Four configurable 32-bit performance monitor counters each capable of counting selected CPU subsystem events
- Performance Monitor interrupt
- Ability to configure performance monitor resources for debugger use
- Hardware input signals for qualification of counting by individual counters
- Hardware output signals to indicate counter overflows
- Dedicated watchpoint outputs for each counter with programmable periodicity
- Trigger On/Off control for each counter based on subsystem events
- Can be used in conjunction with the Sequence Processing Unit counters for additional flexibility (additional 16 32-bit counters)

Performance Monitor Availability

Device	Core	Performance Monitor version	SPU	JTAG Access
MPC564xA	e200z4	Generation 1	No	No
MPC564xL	e200z4	Gen 1	No	No
MPC567xF	e200z760	Gen 1	No	No
MPC567xK	e200z759	Gen 1	No	No
MPC5676R	e200z7	Gen 2	No	No
MPC574xC	e200z4	Gen 2	No	Limited?
MPC574xG	e200z4	Gen 2	No	Limited?
MPC574xP	e200z4	Gen 2	No	Limited
MPC5746R	e200z4	Gen 2	Yes	Yes
MPC5777C	e200z759	Gen 1	No	No
MPC5777M	e200z7/e200z4	Gen 2	Yes	Limited

Performance Monitor Instructions

Agenda

- Session Overview
- Nexus Overview
- Nexus and Debug Use Cases
- Performance Monitor Overview
- **Performance Monitor Instructions**
- Performance Monitor Registers
- Performance Monitor Events
- Performance Monitor Use Cases
- Summary

Special Purpose PM Instructions

- The performance monitor core registers are not memory mapped, they are accessed via special instructions
  - As such these are not a shared resource and cannot been accessed by other cores in a multi-core system
- Special instructions for reading/writing PM registers
  - mfpmr (move from performance monitor register)
  - mtpmr (move to performance monitor register)
- Compiler version that includes these instructions is needed

Read / Write Example

```
void setup_pmu(void)
{
        // PMLCa0 — event 2, clock count
        __mtpmr(144, 0x00020000);
}

unsigned int get_pmc0(void)
{
        return(__mfpmr(16));
}
```

Performance Monitor Registers

Agenda

- Session Overview
- Nexus Overview
- Nexus and Debug Use Cases
- Performance Monitor Overview
- Performance Monitor Instructions
- **Performance Monitor Registers**
- Performance Monitor Events
- Performance Monitor Use Cases
- Summary

Performance Monitor Registers

Supervisor registers	Supervisor PMR Number	User Registers[1]	User PMR Number	Register
PMC0	16	UPMC0	0	(User) Performance Monitor counter 0
PMC1	17	UPMC1	1	(User) Performance Monitor counter 1
PMC2	18	UPMC2	2	(User) Performance Monitor counter 2
PMC3	19	UPMC3	3	(User) Performance Monitor counter 3
PMGC0	400	UPMGC0	384	(User) Performance Monitor global control register 0
PMLCa0	144	UPMLCa0	128	(User) Performance monitor local control a0
PMLCa1	145	UPMLCa1	129	(User) Performance monitor local control a1
PMLCa2	146	UPMLCa2	130	(User) Performance monitor local control a2
PMLCa3	147	UPMLCa3	131	(User) Performance monitor local control a3
PMLCb0	272	UPMLCb0	256	(User) Performance monitor local control b0
PMLCb1	273	UPMLCb1	257	(User) Performance monitor local control b1
PMLCb2	274	UPMLCb2	258	(User) Performance monitor local control b2
PMLCb3	275	UPMLCb3	259	(User) Performance monitor local control b3

1. Read only registers (in user mode)

Global Control Register 0 [gen 1]

- Global configuration
  - Freeze all counters
  - Enable/disable interrupt
  - Freeze counters on enabled condition
  - Time base select
  - Time base Enable

8.3.3 Global Control Register 0 (PMGC0)

The performance monitor global control register PMGC0 shown in Figure 8-1 controls all performance monitor counters.

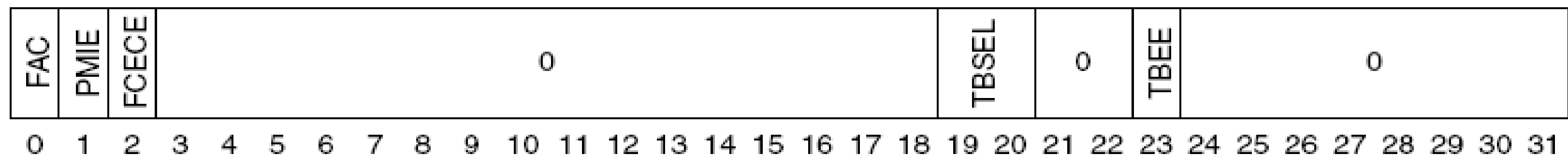
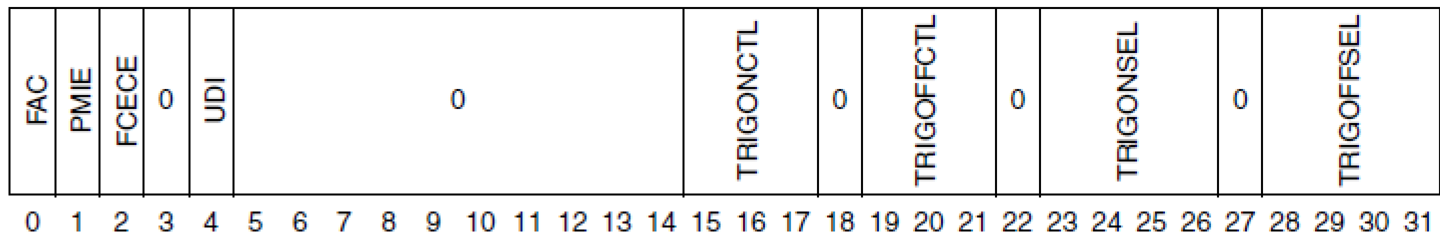


Figure 8-1. Performance Monitor Global Control Register (PMGC0)

PM Global Control Register 0 [gen 2]

- Performance Monitor Global control register
  - Freeze all counters (FAC)
  - PM interrupt enable/disable (PMIE)
  - Freeze counters on enabled condition or event (FCECE)
  - Use debug interrupt (UDI)
  - Trigger-On Control Class (TRIGONCTL)*
  - Trigger-Off Control Class (TRIGOFFCTL)*
  - Trigger-On select (TRIGONSEL)*
  - Trigger-Off select (TRIGOFFSEL)*

*new for c55 generation — MPC57xx



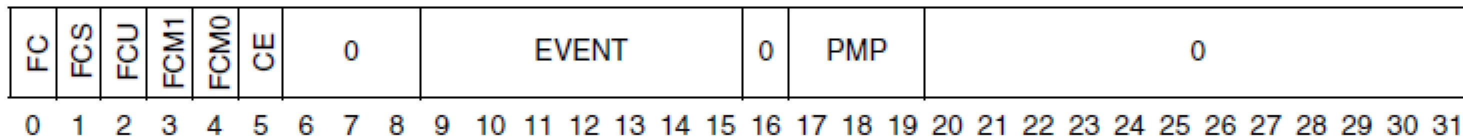
Trigger On/Off Control Class

TRG{on/off}CTRL	Setting Description
0b000	Trigger on/off control is disabled if TRIGONSEL/TRIGOFFSEL is 0b0000 (counting not affected by triggers)
0b001	Reserved
0b010	Trigger-on/off based on selected processor interrupt (TRIGxxSEL=0b0000) [xx=ON or OFF
0b011	Trigger-on/off based on selected hardware signal (selected by TRIGxxxSEL)
0b100	Trigger-on/off based on selected watchpoint occurrence (watchpoint #0-15 selected in TRIGxxxSEL)
0b101	Trigger-on/off based on selected watchpoint occurrence (extension for watchpoint #16-31 selected in TRIGxxxSEL)
0b11x	Reserved

No triggering will occur while PMGC0FAC or PMLCanFC is set to ‘1’.

Local Control A Registers (gen 1/gen2)

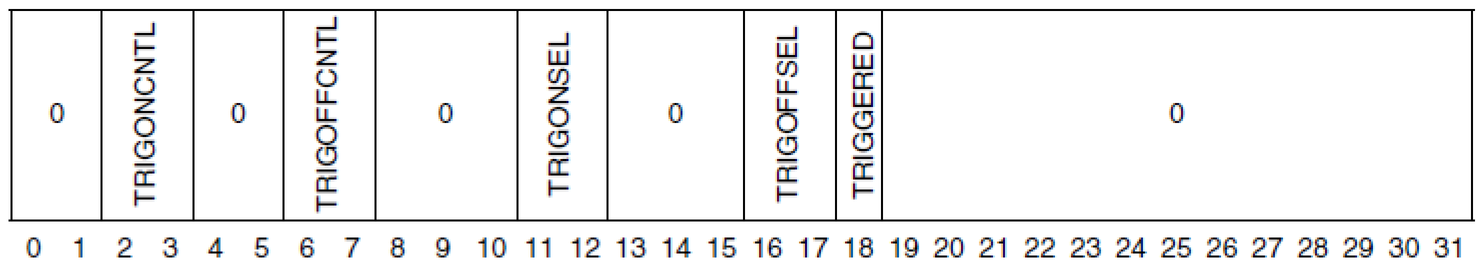
- The local control A registers (PMLCa0-PMLCa3) function as event selectors and give local control for the corresponding performance monitor counters
- Individual event configuration
  - Freeze counter (FC)
  - Freeze counter in Supervisor mode (FCS)
  - Freeze counter in User state (FCU)
  - Freeze counter while Mark is set (FCM1) or cleared (FCM0)
  - Overflow condition enable (CE)
  - Event selection (EVENT)
  - Performance monitor watchpoint periodicity (PMP)



PMR - 144, 145, 146, 147; Read/Write; Reset[1] - 0x0

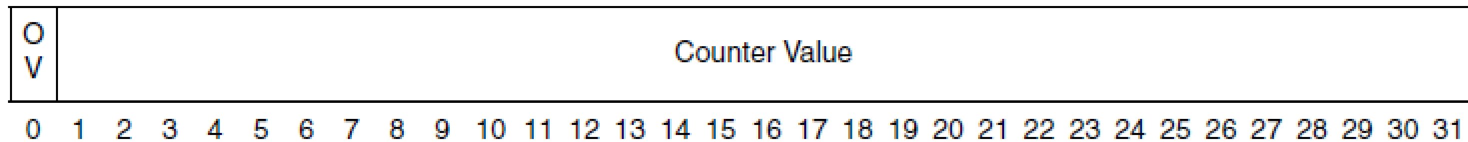
Local Control B Registers (gen1/gen2)

- Local control B registers PMLCb0-PMLCb3) specify triggering condition for the corresponding performance monitor counter. PMLCb is used in conjunction with the corresponding PMLCa register. When triggering is used to enable and/or disable a counter, and the trigger condition is generated by an instruction-based watchpoint, the instruction which generates a trigger-on condition will not generally be counted for most events it generates. An instruction which generates a trigger-off condition will generally still have it's events counted.
- Trigger on/off control class, overflow trigger control, and trigger stat



Counter Registers

- The performance monitor counter registers are 32-bit counters that can be programmed to generate overflow event signals when they overflow. Each counter can be configured to count selected processor events
- Event counter and overflow



Overflow

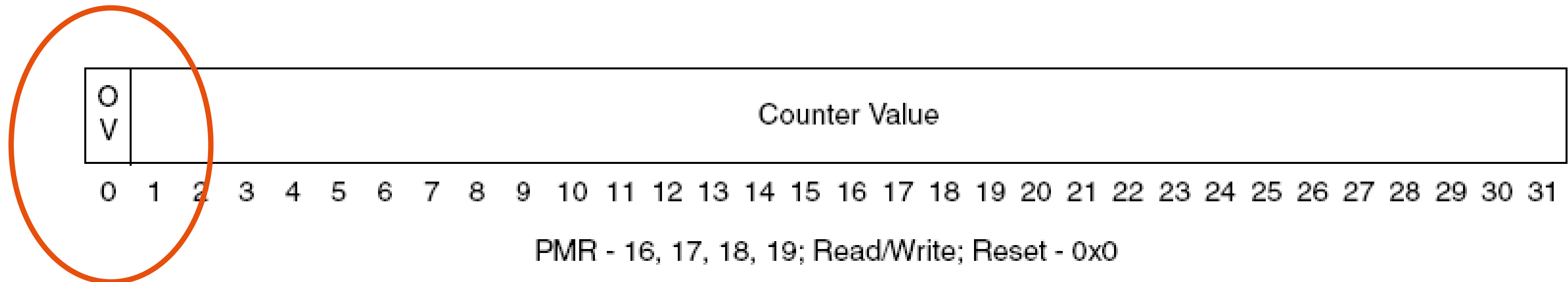


Figure 8-4. Performance Monitor Counter Registers (PMC0–PMC3)

- Bit 0 used for overflow indication
- It is not a sticky bit!!
- Optional interrupt can be generated when bit is set
- Events 97–100 can be used to count overflows to build a 64-bit counter
  - In this case the range is twice the range of interrupt detection

Performance Monitor Events

Agenda

- Session Overview
- Nexus Overview
- Nexus and Debug Use Cases
- Performance Monitor Overview
- Performance Monitor Instructions
- Performance Monitor Registers
- **Performance Monitor Events**
- Performance Monitor Use Cases
- Summary

Event Categories — General Events

Event types	Event number	Event	Include Speculative Counts?	Description
General Event s	0	Nothing	No	Register counter holds current value
	1	Number of processor cycles	No	Every processor cycle not in Waiting, Halted, or Stopped states and not in a debug session.
	2	Instructions completed	No	Completed instructions. 0 or 1 per cycle.
	3	Processor cycles with 0 instructions issued	No	Cycles with no instructions entering execution.
	4	Processor cycles with 1 instruction issued	No	Cycles with one instruction entering execution.
	5	Processor cycles with 2 instruction issued [1]	No	Cycles with two instruction entering execution.
	6	Instructions fetched	Yes	Fetched instruction words. 0, 1, or 2 per cycle. (note that an instruction word may hold 1 or 2 instructions, or 2 partial instructions when fetching from a VLE region)

1. Dual issue CPUs only

Event Categories — General Events

Event types	Event number	Event	Include Speculative Counts?	Description
General Events	7	PM event transitions	-	0 to 1 transitions on the *p_pm_event input.*
	8	PM cycles during event	-	Processor cycles that occur when the *p_pm_event input is asserted*

1. Dual issue CPUs only

Event Categories — Instruction Types Completed

Event types	Event Number	Event	Include Speculative Counts?	Description
Instruction Types Completed	10	Branch instructions completed	No	Completed branch instructions, includes branch and link type instructions
	11	Branch and link type instructions completed	No	Completed branch and link type instructions
	12	Conditional branch instructions completed	No	Completed conditional branch instructions
	13	Taken Branch instructions completed	No	Completed branch instructions which were taken. Includes branch and link type instructions.
	14	Taken Conditional Branch instructions completed	No	Completed conditional branch instructions which were taken.
	15	Load instructions completed	No	Completed load, load-multiple type instructions

Event Categories — Instruction Types Completed

Event types	Event Number	Event	Include Speculative Counts?	Description
	16	Store instructions completed	No	Completed store, store-multiple type instructions
Instruction Types Completed (continued 1)	17	Integer instructions completed	No	Completed integer instructions (not a load-type/store-type/branch/mul/div, EFPU)
	18	Multiply instructions completed	No	Completed Multiply instructions (non-EFPU)
	19	Divide instructions completed	No	Completed Divide instructions (non-EFPU)
	20	Divide instruction execution cycles	No	Cycles of execution for all Divide instructions (non-EFPU)
	21	EFPU FP instructions completed	No	Completed EFPU FP instructions

Event Categories — Instruction Types Completed

Event types	Event Number	Event	Include Speculative Counts?	Description
Instruction Types Completed (continued 2)	62	EFPU instructions completed	No	Completed EFPU instructions. Does not include load and store instructions.
	63	Number of return from interrupt instructions	No	Includes all types of return from interrupts (i.e. **se_rfi, se_rfci, se_rfdi, se_rfmci)**

Event Categories

Event types	Event number	Event	Include Speculative Counts?	Description
Pipeline Stalls	22	Cycles decode stalled due to no instructions available	Yes	No instruction available to decode
	23	Cycles issue stalled, not due to empty instruction buffer	Yes	Cycles the issue buffer is not empty but 0 instructions issued
Load/Store Events	24	Store buffer full stalls	Yes	Stall cycles due to store buffer full
Data Cache, and Data Line Fill Events	25	Dcache linefills	Yes	Counts dcache reloads for any reason, including touch-type reloads. Typically used to determine approximate data cache miss rate (along with loads/stores completed).
	26	Dcache load hits	Yes	

Event Categories

Event types	Event Number	Event	Include Speculative Counts?	Description
Fetch, Instruction Cache, Instruction Line Fill, and Instruction Prefetch Events	27	Icache linefills	Yes	Counts icache reloads due to demand fetch. Used to determine instruction cache miss rate (along with instructions completed)
	28	Number of Instruction fetches	Yes	Counts fetches that write at least one instruction to the instruction buffer. (With instruction words fetched (com:6), can used to compute instruction words-per-fetch, or with icache linefills, can compute icache hit rate)
BIU Interface Usage	29	BIU instruction-side transfers	Yes	instruction-side transaction beats
	30	BIU instruction-side cycles	Yes	instruction-side transaction clock cycles
	31	BIU data-side transfers	Yes	data-side transaction beats
	32	BIU data-side cycles	Yes	data-side transaction clock cycles
	33	BIU single-beat write cycles	Yes	single beat write transaction clock cycles

Event Categories

Event types	Event Number	Event	Include Speculative Counts?	Description
Chaining Events	34	PMC0 rollover	No	$PMC0_{OV}$ transitions from 1 to 0.
	35	PMC1 rollover	No	$PMC1_{OV}$ transitions from 1 to 0.
	36	PMC2 rollover	No	$PMC2_{OV}$ transitions from 1 to 0.
	37	PMC3 rollover	No	$PMC3_{OV}$ transitions from 1 to 0.
Interrupt Events	38	Interrupts taken	No	-
	39	External input interrupts taken	No	-
	40	Critical input interrupts taken	No	-
	41	Cycles in which MSR_{EE}=0	No	-
	42	Cycles in which MSR_{CE}=0	No	-

Event Categories — Watchpoint Events

Event types	Event Number	Event	Include Speculative Counts?	Description
Watchpoint Events	43–58	Watchpoint #0 to 15 occurs	No	Assertion of watchpoint 0 to 15 detected
	59	Watchpoint #27 occurs	No	Assertion of watchpoint 27 detected

Event Categories — Cache Array Events

Event types	Event Number	Event	Include Speculative Counts?	Description
Cache Array Events	64	Cycles Dcache data array enabled for a read		
	65	Cycles Icache data array enabled for a read		
	66	Cycles DMEM array enabled for a read		
	67	Cycles IMEM array enabled for a read		
	68	Cycles DMEM array and Dcache data array both enabled for a read		
	69	Cycles IMEM array and Icache data array both enabled for a read		
	70	Cycles data access is stalled due to Dcache latewrite buffer		
	71	Cycles data access is stalled due to full DMEM write buffer		
	72	Cycles data access is recycled		Dcache recycled lookup
	73	Cycles instruction access is recycled		Icache recycled lookup

Performance Monitor Use Cases

Agenda

- Session Overview
- Nexus Overview
- Nexus and debug Use Cases
- Performance Monitor Overview
- Performance Monitor Instructions
- Performance Monitor Registers
- Performance Monitor Events
- **Performance Monitor Use Cases**
- Summary

NXP

Instructions Per Clock (IPC)

- Use counter 0 & 1 to build a 64-bit counter to count **processor clocks**
  - PMLCa0 set to event 1 (processor cycles)
  - PMLCa1 set to event 34 (overflow of counter 0)
- Use counter 2 & 3 to build 64-bit counter to count **instructions completed**
  - PMLCa2 set to event 2 (instructions completed)
  - PMLCa3 set to event 36 (overflow of counter 2)

IPC = Instructions / Processor clocks

Lauterbach TRACE32 BenchMarkCounter

- TRACE32 has a GUI for assigning the Performance Monitor events to the PM counters.
- Type “BMC” to get a simple interface for collecting data.
- In addition to the standard PM use cases, these can be combined with trace and monitoring the PM registers during runtime for more capabilities

Freeze stops the counters in debug mode
AutoInit clears the counters when exiting debug mode

B::BMC

control: RESet, Init, Freeze, AutoInit, Attach
profile: PROfile
snoop: SNOOPer, List, SnoopSet, PROfileChart
SELect: CNT0, TREE, sYmbol, sYmbol
CLOCK

counter name	event	size	value	ratio	ratio	ov
CNT0	PROC-CYC (Processor cycles (C0))	32BIT	11040039	OFF		
CNT1	INST-CMP (Instructions completed (C0-1))	32BIT	12816147	X/CNT0	116.087%	
CNT2	BR-CMP (Branch instructions completed (C2-3))	32BIT	1304548	X/CNT0	11.816%	
CNT3	BC-CMP (Conditional branch instructions completed (C2-3	32BIT	1100001	X/CNT0	9.963%	

erformanceBenchmark: Optimization, Cache, and –SDA effects on performance → Measure Execution Time and IPC

Optimizing myBenchmark Loop

2.50
2.00
1.50
1.00
0.50
0.00

ipc
time (us)
-Onone
ipc
time (us)
-Onone, -sda
ipc
time (us)
-Ospeed, -sda

non-cached
cached

See FTF-ACC-F1185 Performance Optimization Tips and Hints for Automotive Power Architecture

General Performance indicator

Get a feel for general performance of code and compiler. Must be careful no overflows occur. Shows how well single/dual issue is working and when stalled.

- PMLCa0 set to event 1 (processor cycles)
- PMLCa1 set to event 3 (cycles with 0 instructions issued)
- PMLCa2 set to event 4 (cycles with 1 instruction issued)
- PMLCa3 set to event 5 (cycles with 2 instructions issued)

B::BMC

control: RESet | Freeze | Init | AutoInit | Attach
profile: PROfile
snoop: SNOOPer | List | SnoopSet | PROfileChart
SELect: CNT0 | TREE | sYmbol | sYmbol
CLOCK

counter name	event	size	value	ratio	ratio	ov
CNT0	PROC-CYC (Processor cycles (C0))	32BIT	11040039	OFF		
CNT1	PROC-CYC-0II (Processor cycles with 0 instructions issu	32BIT	2660645	X/CNT0	24.099%	
CNT2	PROC-CYC-1II (Processor cycles with 1 instructions issu	32BIT	3942641	X/CNT0	35.712%	
CNT3	PROC-CYC-2II (Processor cycles with 2 instructions issu	32BIT	4436753	X/CNT0	40.187%	

erformanceBenchmark: Before and After Compiler Optimization → measure instructions issued, IPC and exec time

200%
180%
160%
140%
120%
100%
80%
60%
40%
20%
0%

0 issued | 1 issued | 2 issued | ipc | Execution Time

183 us
95 us

no opt
opt

See FTF-ACC-F1185 Performance Optimization Tips and Hints for Automotive Power Architecture

Number of loads versus stores

- In the below snapshot, Counter1 was set up to handle overflows of Counter0. (it was not needed in this example.)

B::BMC

control: RESet, Freeze, Init, AutoInit, Attach | profile: PROfile | snoop: SNOOPer, List, SnoopSet, PROfileChart | SELect: CNT0, TREE, sYmbol, sYmbol | CLOCK

counter name	event	size	value	ratio	ratio	ov
CNT0	INST-CMP (Instructions completed (C0-1))	32BIT	12816147	OFF		
CNT1	PMC1-OV (PMC1 overflow)	32BIT	0	X/CNT0	0.000%	
CNT2	LD-CMP (Load instructions completed (C2-3))	32BIT	3203517	X/CNT0	24.995%	
CNT3	ST-CMP (Store instructions completed (C2-3))	32BIT	1900002	X/CNT0	14.825%	

Interrupt Performance

- Use counter 0 & 1 to build a 64-bit counter to count processor clocks
  - PMLCa0 set to event 1 (processor cycles)
  - PMLCa1 set to event 34 (overflow of counter 0)
- Use counter 2 to measure the number of interrupts taken
  - PMLCa2 set to event 38 (interrupts taken)
- Use counter 3 to measure the number of critical interrupts taken
  - PMLCa3 set to event 40 (cycles with external [core] interrupts disabled)

B::BMC

control: RESet, Freeze, Init, AutoInit, Attach
profile: PROfile
snoop: SNOOPer, List, SnoopSet, PROfileChart
SELect: CNT0, TREE, sYmbol, sYmbol
CLOCK

counter name	event	size	value	ratio	ratio	ov
CNT0	PROC-CYC (Processor cycles (C0))	32BIT	11040039	OFF		
CNT1	INST-CMP (Instructions completed (C0-1))	32BIT	12816147	X/CNT0	116.087%	
CNT2	INT (Interrupts taken (C2-3))	32BIT	0	X/CNT0	0.000%	
CNT3	MSREE_0 (Cycles in which MSR[EE]=0 (C2-3))	32BIT	0	X/CNT	0.000%	

Code Segment

- Use counter 0 & 1 to build a 64-bit counter to count processor clocks
  - PMLCa0 set to event 1 (processor cycles)
  - PMLCa1 set to event 34 (overflow of counter 0)
- Use counter 2 & 3 to build 64-bit counter to count processor clocks
  - PMLCa2 set to event 1 (processor cycles)
  - PMLCa3 set to event 36 (overflow of counter 2)
  - Use PMLCa2[FC] bit to control the counting of specific code segment(s)

Supervisor / User Mode

- Use counter 0 & 1 to build a 64-bit counter to count processor clocks
  - PMLCa0 set to event 1 (processor cycles)
  - PMLCa1 set to event 34 (overflow of counter 0)
- Use counter 2 & 3 to build 64-bit counter to count processor clocks
  - PMLCa2 set to event 1 (processor cycles)
  - PMLCa3 set to event 36 (overflow of counter 2)
  - Use PMLCa2[FCS]/[FCU] bit to control the counting of specific code segment(s)

Summary

Agenda

- Session Overview
- Nexus Overview
- Nexus and debug Use Cases
- Performance Monitor Overview
- Performance Monitor Instructions
- Performance Monitor Registers
- Performance Monitor Events
- Performance Monitor Use Cases
- **Summary**

Generation 1 (primarily c90/MPC56xx) to Gen 2 (c55/MPC67xx) differences

- Global configuration register changes, adding debug interrupt capability and control of registers via the JTAG port
- Events change
  - Not all events can be used with all counters
  - Event numbers have changed
    - C90 overflow was events 97–100, in c55 they are events 34–37
  - Reduced number of events
    - C90 event numbers go up to 158
    - C55 event numbers go up to 73
  - JTAG/ONcE register access added during runtime

Considerations

- Be careful when comparing events against each other
  - For example, you cannot add up the different stall events to get total cycles stalled. Different types of stalls can happen in same cycle
- Overflow mechanism in non-traditional
- Don’t be bogged down in details. Customer should be controlling engines not worrying about Dcache stream hit rates
- Beware on differences between c90 and c55 versions of the PM

Session Summary

- The e200zx core Performance Monitor can enhance the MCU system development capabilities above the capabilities provided by the IEEE 5001 Nexus trace interface.
  These features can be used to enhance software quality and performance.

Now Start Developing Your MCU Software

FTF Recommended Sessions

FTF-ACC-F1185 Performance Optimization Tips and Hints for Automotive Power Architecture

www.Freescale.com

© 2015 Freescale Semiconductor, Inc. | ***External Use***