FTF | FREESCALE TECHNOLOGY FORUM
POWERING INNOVATION

Key Features of Qorivva 32-bit MCUs for Body Applications

FTF-AUT-F0546

Alasdair Robertson
MSG Applications Engineer

23 June 2011

Agenda

- Overview of Qorivva family and MPC564xB/C (5 minutes)
  - Roadmaps and main features
- Introducing dual core (15 minutes)
  - Why dual core?
  - Configuring and using dual core
  - Debugging a dual core system
- Smart peripherals for lighting (20 minutes)
  - Requirements & lamp driver control
  - MPC564xB/C features
- Gateway communications and questions (5 minutes)

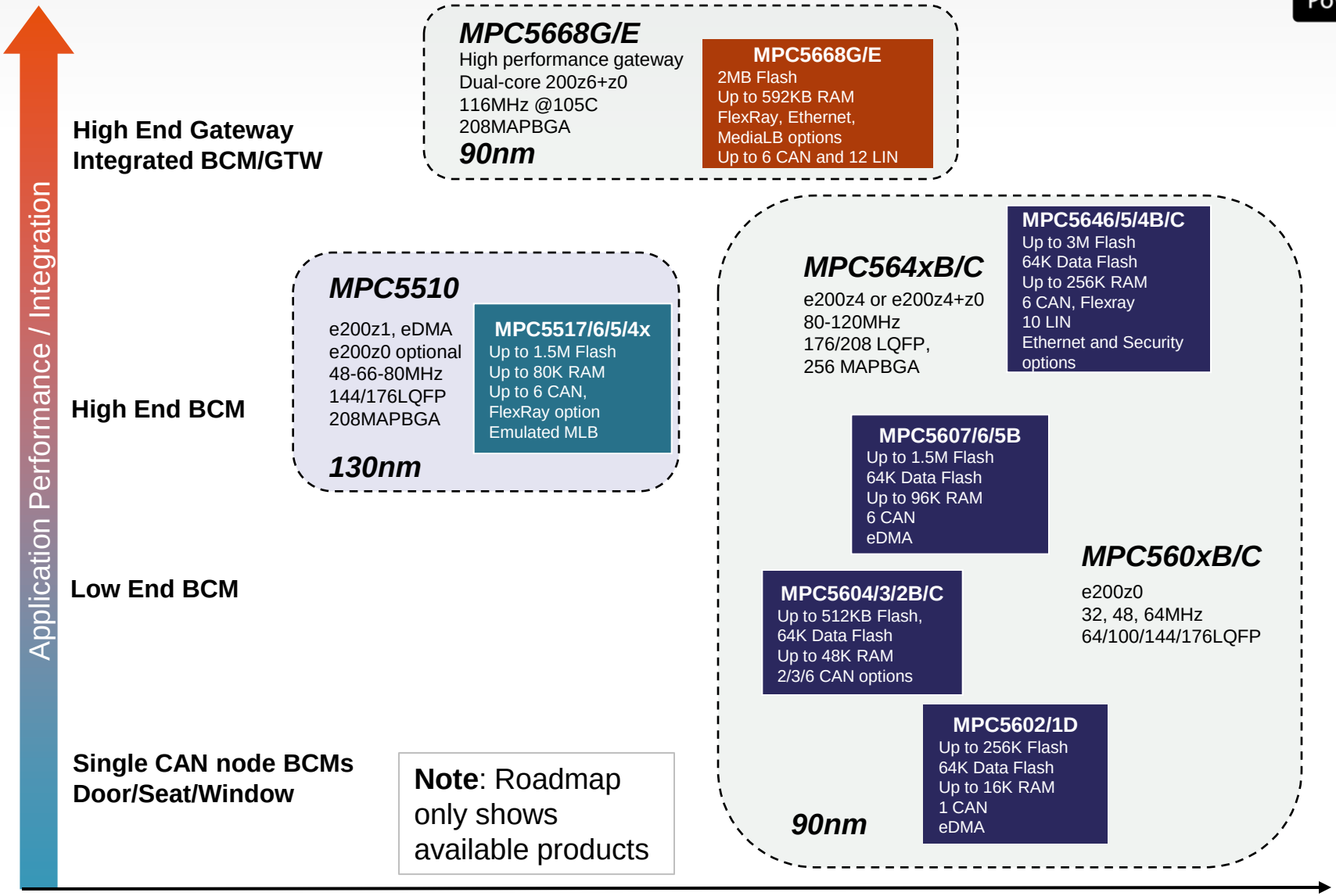
Freescale on Facebook
Tag yourself in photos
and upload your own!

Tweeting?
Please use hashtag
#FTF2011

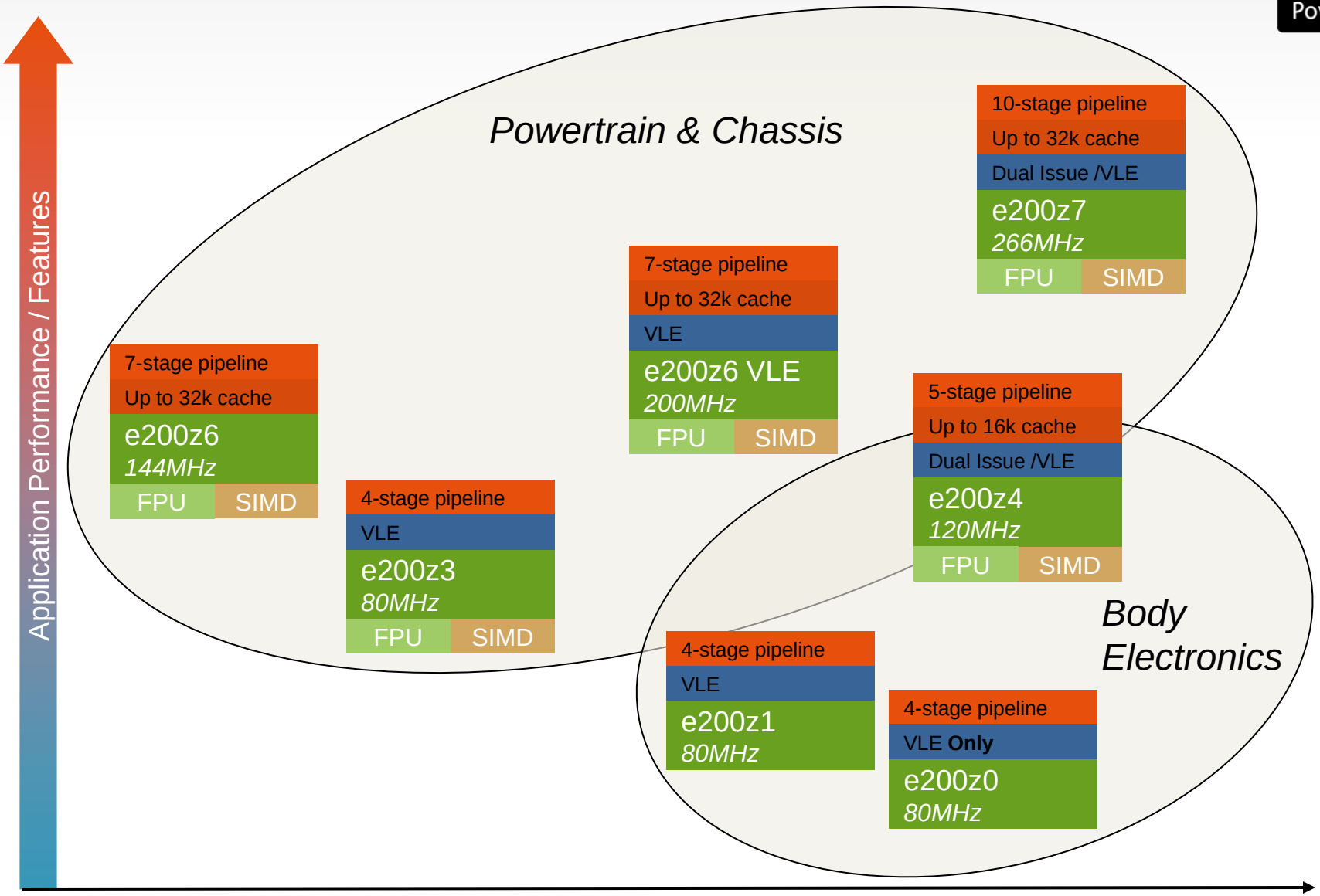
Qorivva & MPC564xB/C Overview

- 32-bit body roadmap and family compatibility
- Core roadmap and core compatibility
- MPC564xB/C feature set
- Part number decrypter
- Architected for performance (IOP, XBAR, DMA)

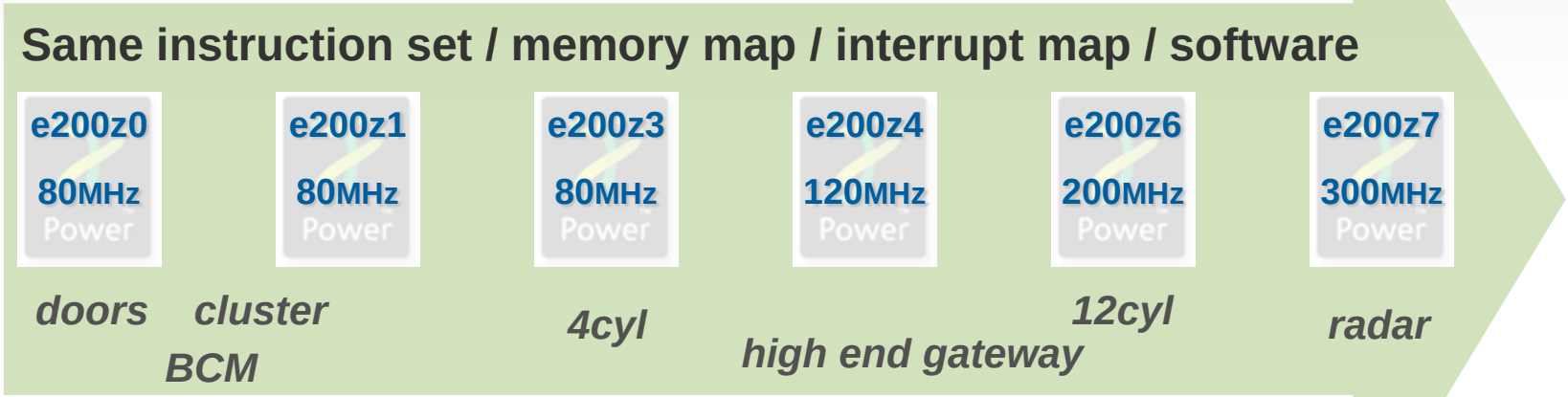
Overview: 32-bit Body Electronics MCU Roadmap



Overview: Qorivva e200z Core Roadmap

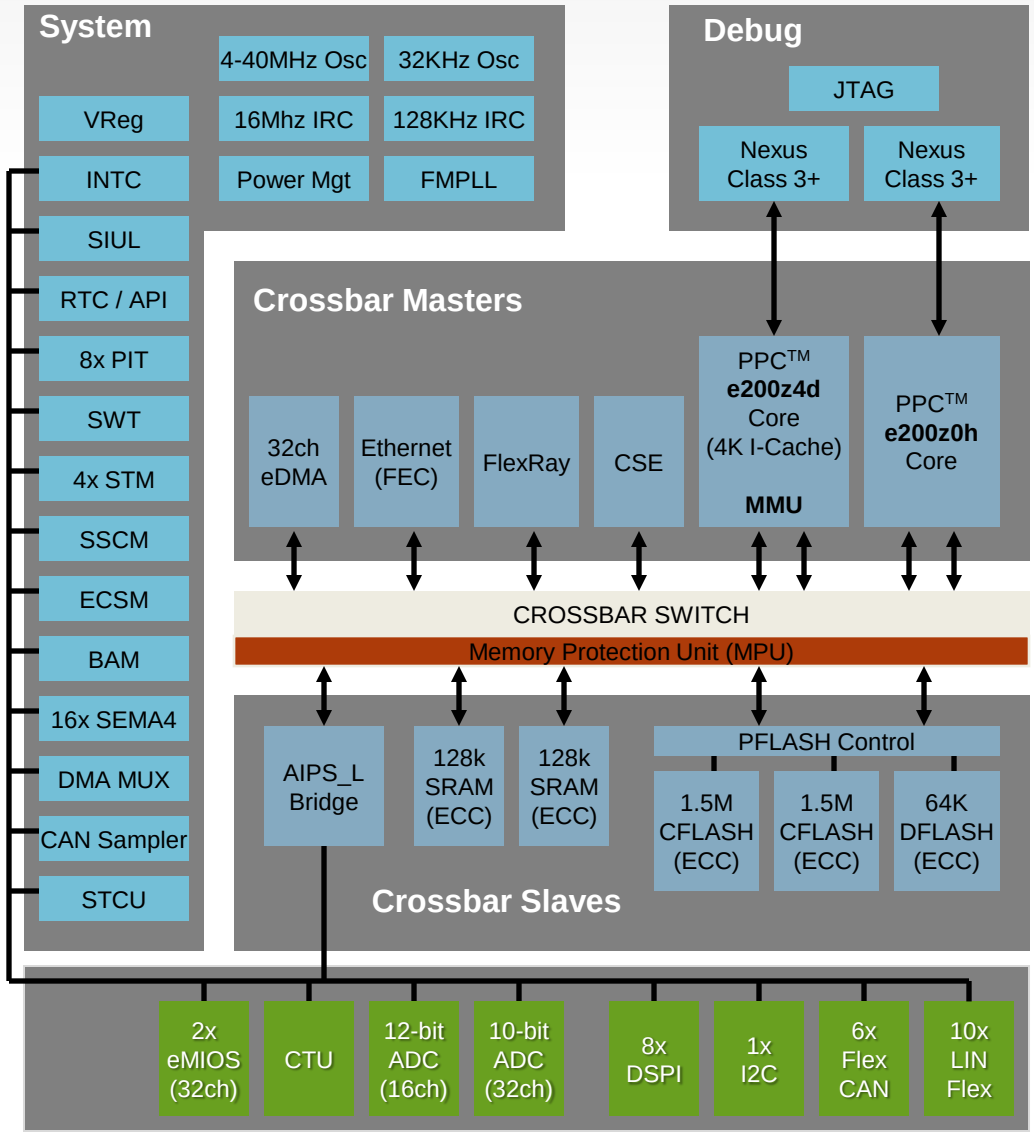


Overview: Qorivva e200z Automotive Platform



- Highly modular core
  - Add DSP, FPU, cache
  - Bigger pre-fetch buffers
  - Packaging, module library, peripherals
  - Single and dual core options
- Cost reduction through maximum re-use
  - Same tools, drivers, application code
  - Build a cross-application platform with 1 core

Overview: MPC564xB/C – Part 1



Crossbar & MPU

- SPP XBAR with 8 masters and 5 slaves
- Memory protection unit with 8 regions, 32byte granularity for all crossbar access

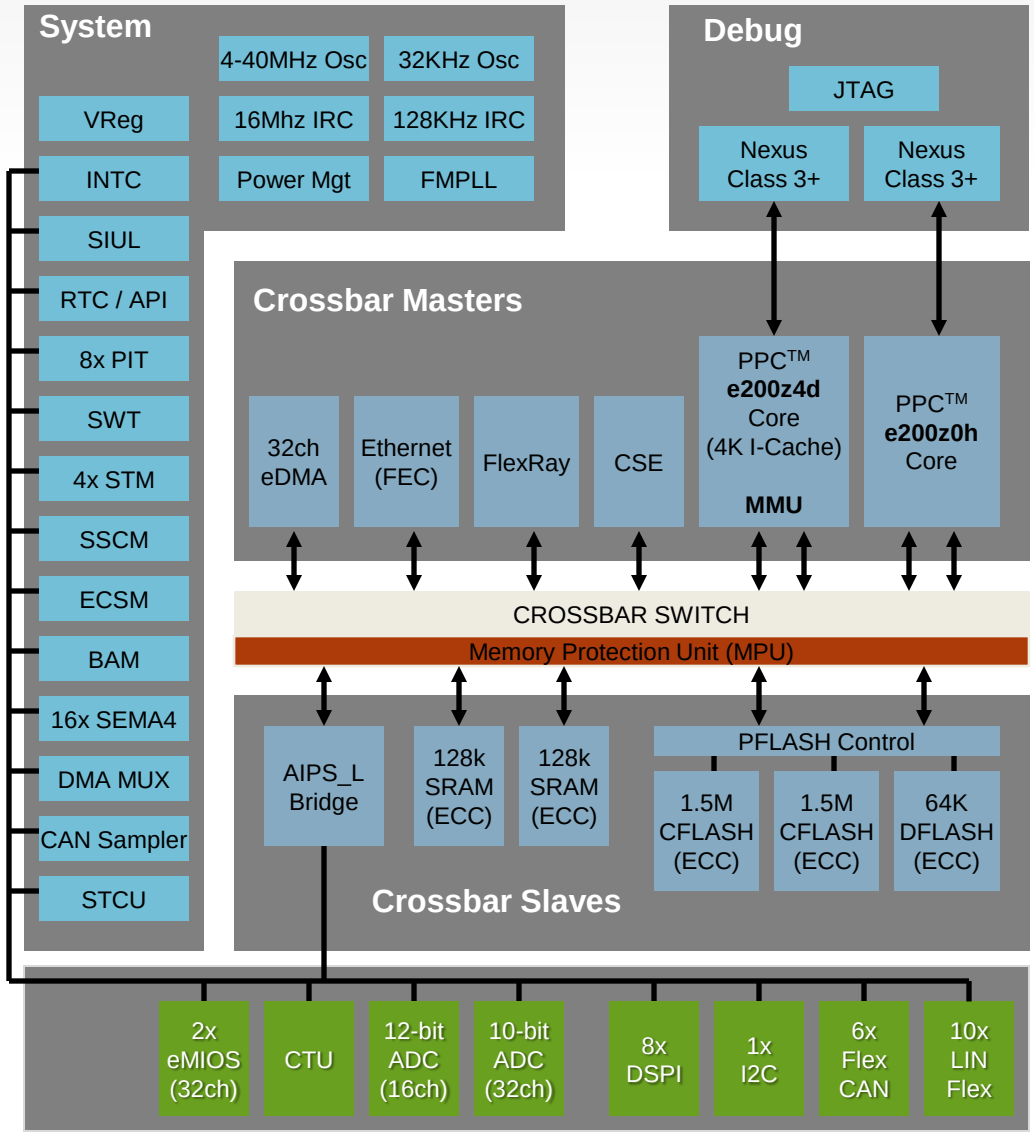
Core(s)

- e200z4d (dual issue) Harvard core running up to 120MHz
- MMU (e200z4 only)
- Optional e200z0h core running up to 80MHz
- Book VLE instruction set for superior code density (e200z0 is VLE only)
- Vectored interrupt controller

Debug

- Basic RTC via JTAG
- Nexus 3+ debug (Instruction & Data trace) on both cores
- Can be used to provide Nexus 3+ trace for other e200z0 Bolero family devices

Overview: MPC564xB/C – Part 2



Memory

- 3Mbyte embedded code Flash
- 64Kbyte embedded data Flash optimized for EEPRom emulation
- 256Mbyte user SRAM
- ECC on Flash and user SRAM (single bit correct, double bit detect)
- Highly optimized memory / XBAR configuration for maximum performance

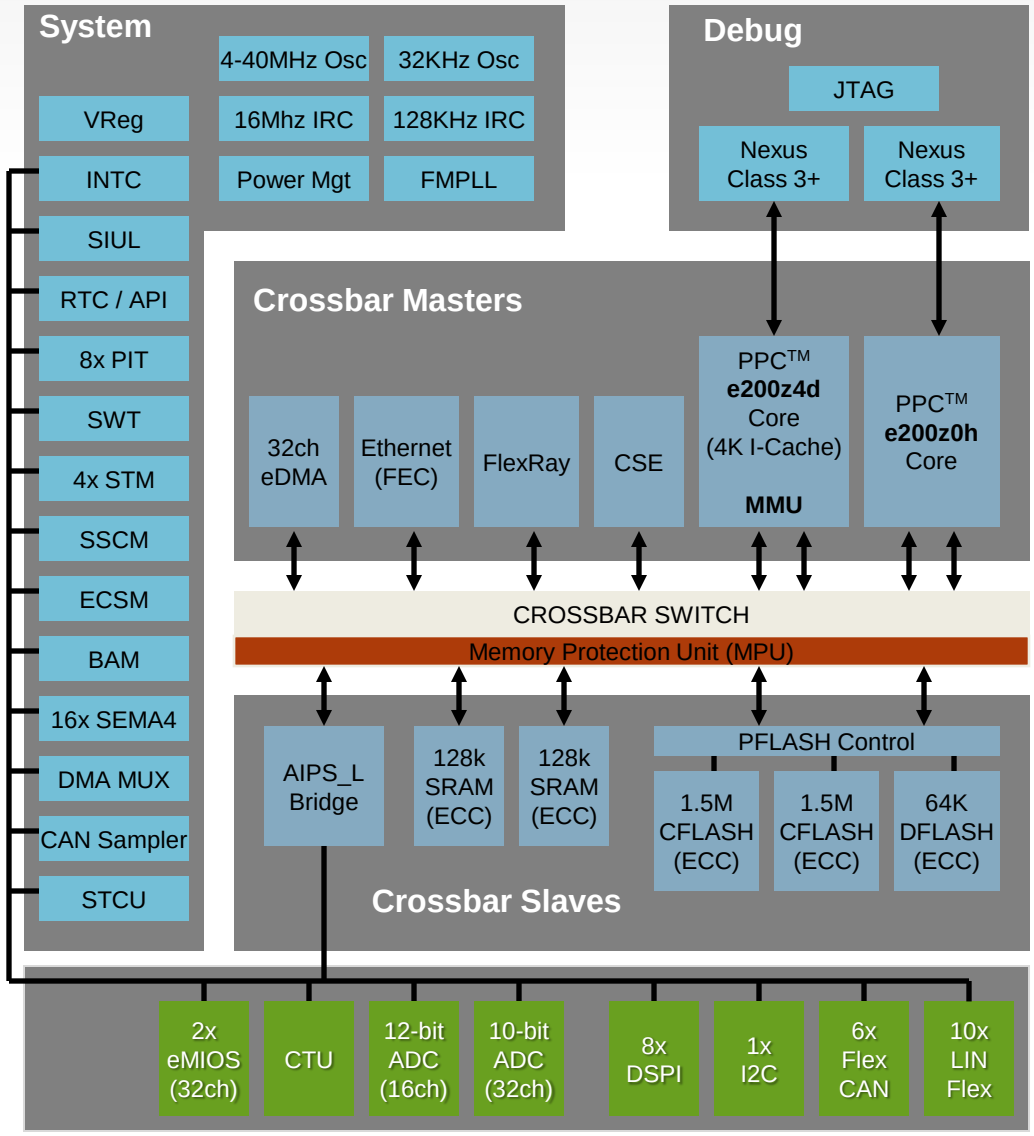
DMA

- 32ch DMA controller for memory to memory transfers without core intervention
- Can be interrupt driven from peripherals

Other Masters

- Fast Ethernet Controller (FEC) offering 100Mbps data rate
- FlexRay master
- CSE (Cryptographic Services Engine) for enhanced security

Overview: MPC564xB/C – Part 3



Flexible Clocking Structure

- 16MHz internal reference clock
- 12KHz internal reference clock
- 4-40MHz external oscillator / module
- 32Khz external oscillator
- PLL with frequency modulation

Additional Communications

- 8x DSPI with DSI and eMIOS serialization
- 1x I2C
- 6x enhanced FlexCAN (Full 2.0 spec, 64 message buffers)
- Can sampler for enhanced wakeup on CAN event
- 10x LinFLEX with DMA support

Analog

- 32 ch 10-bit ADC / 16 ch 12-bit ADC

Timed I/O

- 2x 32ch eMIOS
- Cross trigger unit to allow triggering of ADC conversion from eMIOS event (DMA trigger or specific CTU trigger)

Overview: MPC5600 Part Numbers

M	- MC Qualification Status
P	- Power™ Core
C	
56	- Automotive Platform (C90)
4	- e200z4 Core
6	- Flash Size (Core dependent)
C	- Product Family
E	- Data Flash (blank if none)
F	- FlexRay (blank if none)
M	- 125CTemp Spec
L	- 176LQFP Package
U	

Optional fields

Qual Status

- M: MC Status
- S: Auto Qualified
- P: PC Status

Auto Platform

- 56: PPC in 90nm
- 57: PPC in 65nm

Core Version

- 0: e200z0
- 1: e200z1
- 2: Reserved
- 3: e200z3
- 4: e200z4
- 5: Reserved
- 6: e200z6
- 7: e200z7
- 8: Reserved
- 9: Reserved

Flash Size (Z0, Z1 based MCUs)

- 0…Reserved
- 1….128 or 192kByte
- 2….256kByte
- 3…320 or 384kByte
- 4…512kbyte
- 5…768kByte
- 6…1024kByte
- 7…1.5MByte
- 8…2MByte
- 9…3MByte

Flash Size (Z3, Z4, Z6, Z7 based MCUs)

- 0…Reserved
- 1….512kByte
- 2….768kByte
- 3…1024kByte
- 4…1.5MByte
- 5…2MByte
- 6…3MByte
- 7…4MByte
- 8…6MByte
- 9…8MByte

Prod/fam Name

- M: Monaco
- A: Andorra
- R: San Marino
- T: Taipan
- R: Tiger
- E: Copper
- V: Viper
- W: Cobra (dual core)
- B: Bolero standard
- C: Bolero gateway
- S: Spectrum
- P: Pictus steering
- Y: Tokay
- L: Leopard
- K: Komodo
- N: Lance

Temp Spec

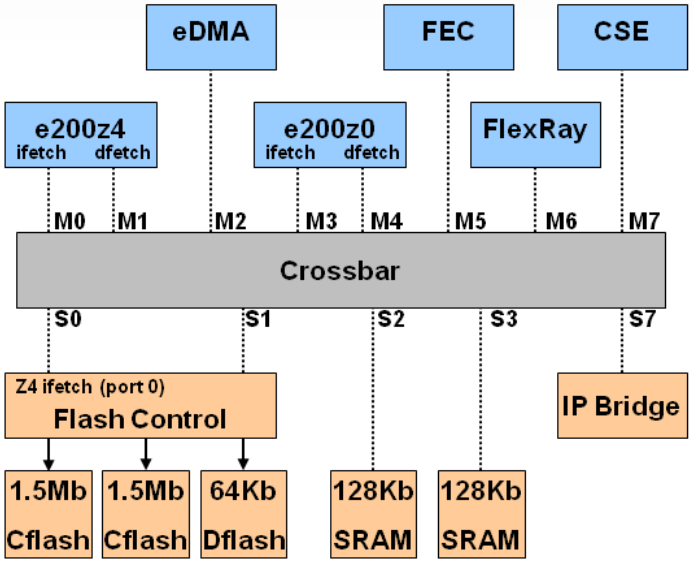
- C: 85C
- V: 105C
- M: 125C

Package Code

- LL: 100LQFP
- LQ: 144LQFP
- LU176LQFP
- MG: 208MAPBGA
- VZ: 324PBGA
- MW: 496 CSP

Overview: MPC564xB/C Architected for Performance

- Crossbar
  - 5 simultaneous master / slave accesses (to different slaves) permitted with no contentions
  - Master 0 to Slave 0 dedicated to e200z4 instruction fetches
- eDMA
  - Removes CPU load for data transfers
- Smart Peripherals
  - Queues, Interrupt & DMA support, cross peripheral interaction (CTU) all reduce core load
- I/O Processor (e200z0)
  - Highly flexible and customer specific usage (main core monitor, task partitioning, interrupt handler)



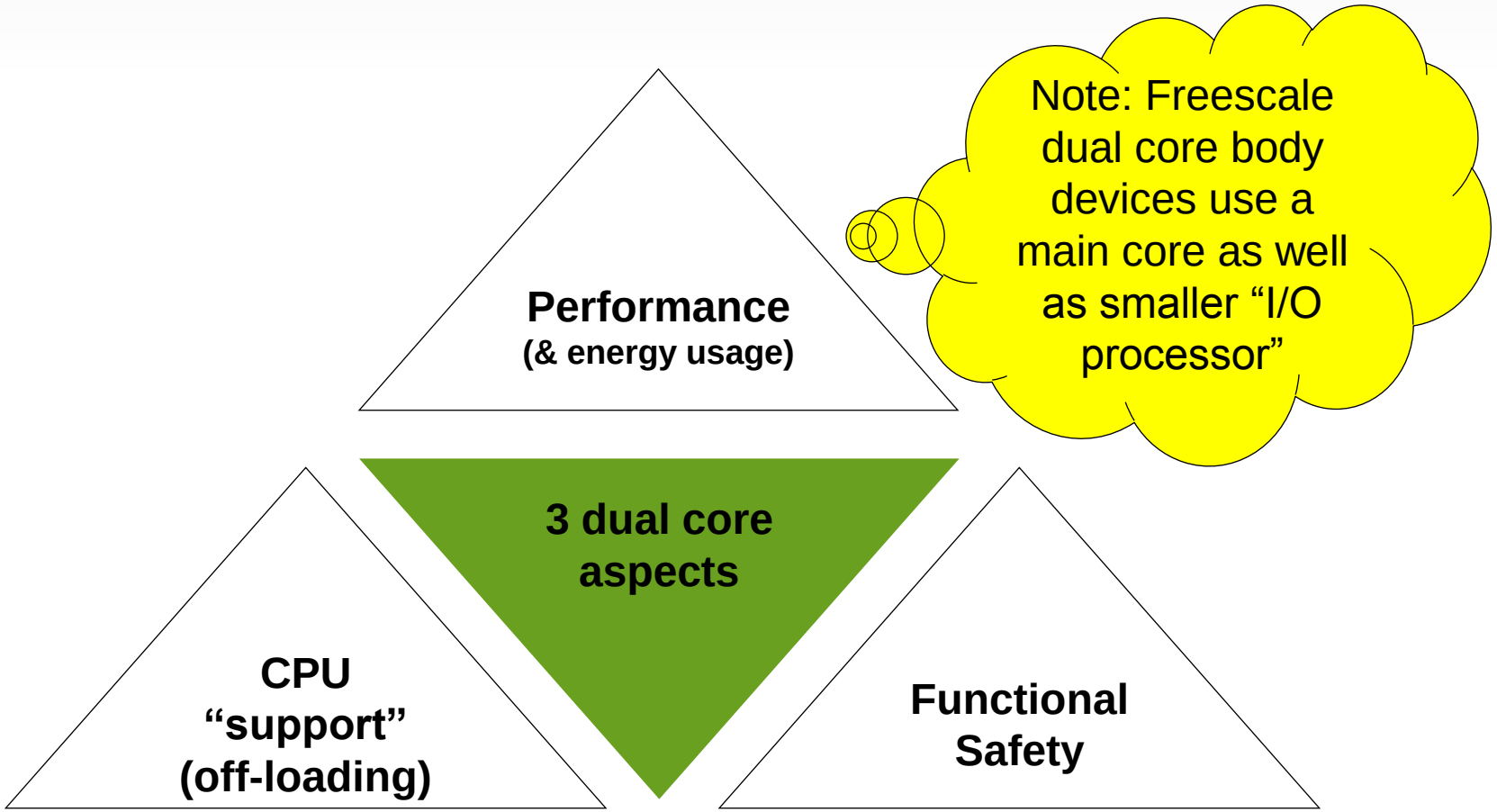
FTF | FREESCALE TECHNOLOGY FORUM
POWERING INNOVATION

Dual Core

Dual Core

- Why dual core?
- Using dual core on the MPC564xB/C. Three-step approach:
  - Planning
  - Setup
  - Using

Dual Core: Why Dual Core? – Three Aspects



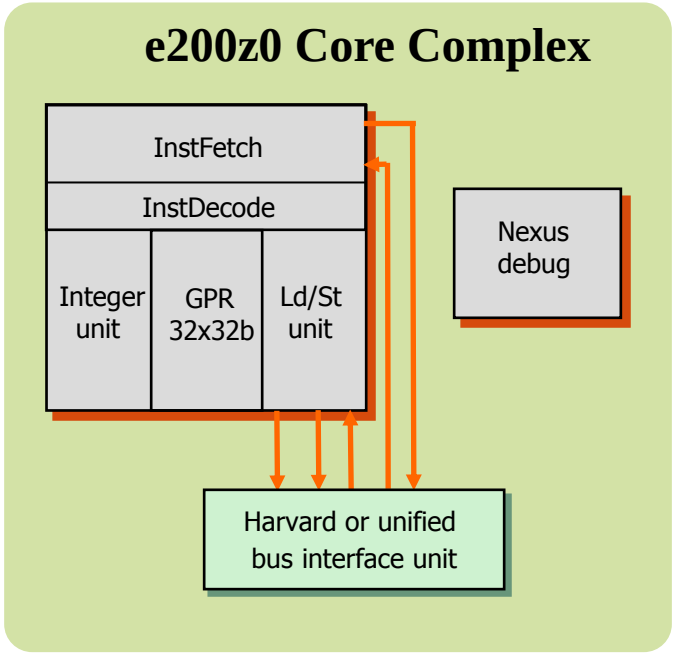
Dual Core: Introducing the I/O Processor

The MPC564xB/C utilizes several features to improve system performance:

- **Smart peripherals** provide DMA serviced queues as well as cross triggering
- **DMA** can move large blocks of data or service smart peripherals with minimal core intervention
- **The IOP** is a complete standalone secondary core that has full access to all of the MPC564xB/C peripherals and memory

About the IOP

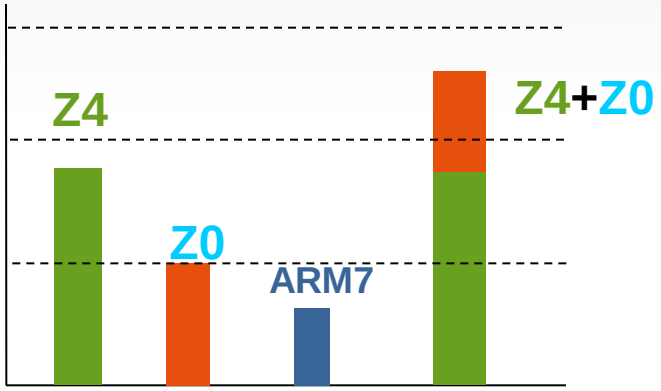
- The smallest e200z core implementation
- 32-bit, VLE only core
- Single-issue machine
- 4-stage pipeline
- No 64-bit core timer
  - No decrementer / watchdog on core
- No MMU
- Nexus 3+ debug, first used on MPC564xB/C



Dual Core: Why Do I Need an IOP?

IOP boosts system performance

- By offloading significant stand-alone tasks from the main core, system performance is increased
- True multi-tasking is possible - with careful task partitioning, the effect of peripheral / crossbar clashes can be minimized
- Performance is increased without increasing system clock speed



Low-power support

- The dual core architecture allows one core to be stopped for power saving
- The lower current IOP can be used to recover from low power mode and then decide whether to wake the main core

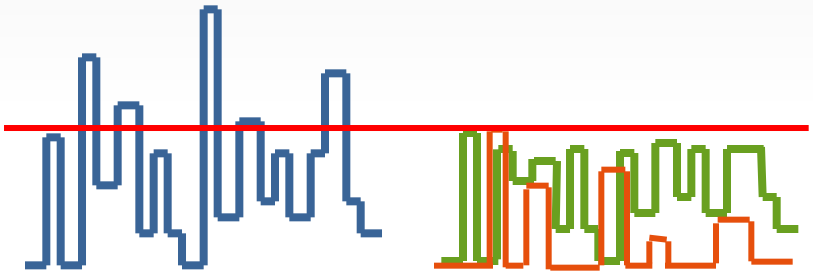
Full chip cyclic wake up

Smaller second core or 2 cores running reducing time ON

Dual Core: Why Do I Need an IOP – 2?

Current consumption / EMC

- Overall clock frequency is reduced
- Reduces EMC
- Reduces power as main core is running slower and IOP runs only when required

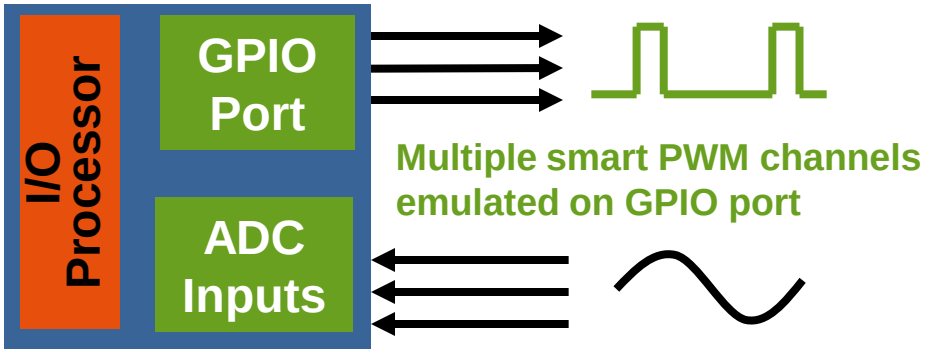


Single core, high frequency operation

Main core running at reduced speed and Second core periodically helping

Flexible and expandable

- The dual core solution provides a cost-effective way of reducing multiple CPU system cost
- Although some customers are not currently using the IOP, it also can be viewed for future expansion
- Ideal for peripheral emulation



Multiple smart PWM channels emulated on GPIO port

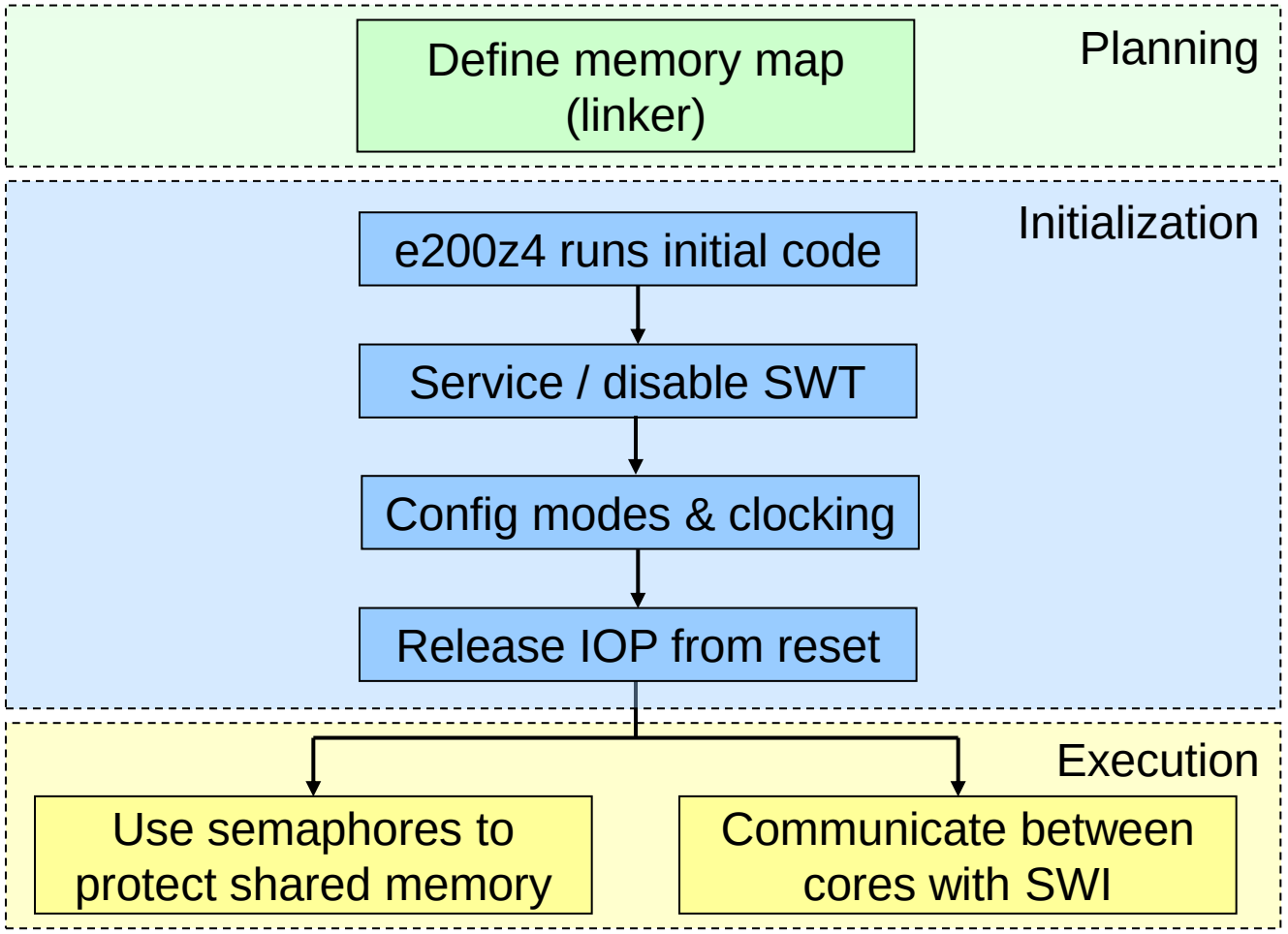
Dual Core: Application Examples of IOP Usage

- **Gateway function**
  - The e200z0 performs a lot of the gateway communication, handling multiple incoming / outgoing packets; the e200z4 essentially is left to handle other BCU functions
- **Safety applications**
  - The e200z0 monitors the operation of the e200z4
  - Potentially checking critical calculations and acting as a smart watchdog
  - If there are any anomalies, the system can be put into a “safe” mode
- **Multi-function**
  - Both cores perform completely separate functions
- **Smart peripheral extension**
  - e200z0 is used to provide additional “functions” like PWM

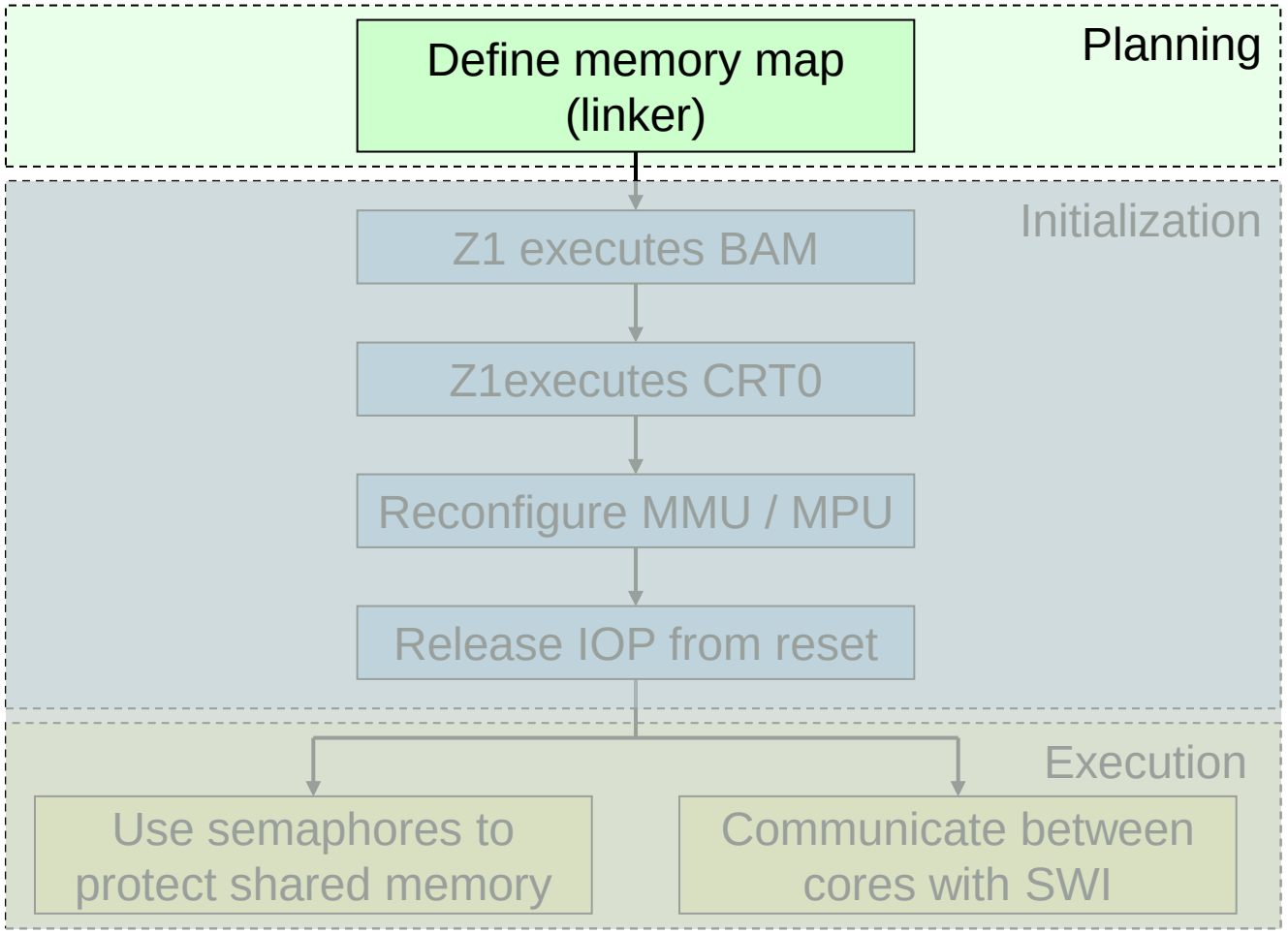
Note that the MPC564xB/C is not really designed for an SMP based multi core system where the O/S schedules tasks / threads to multiple cores. (Needs to be more deterministically setup than an SMP system)

Dual Core: Required Steps to Using the IOP

- Three-Step Approach



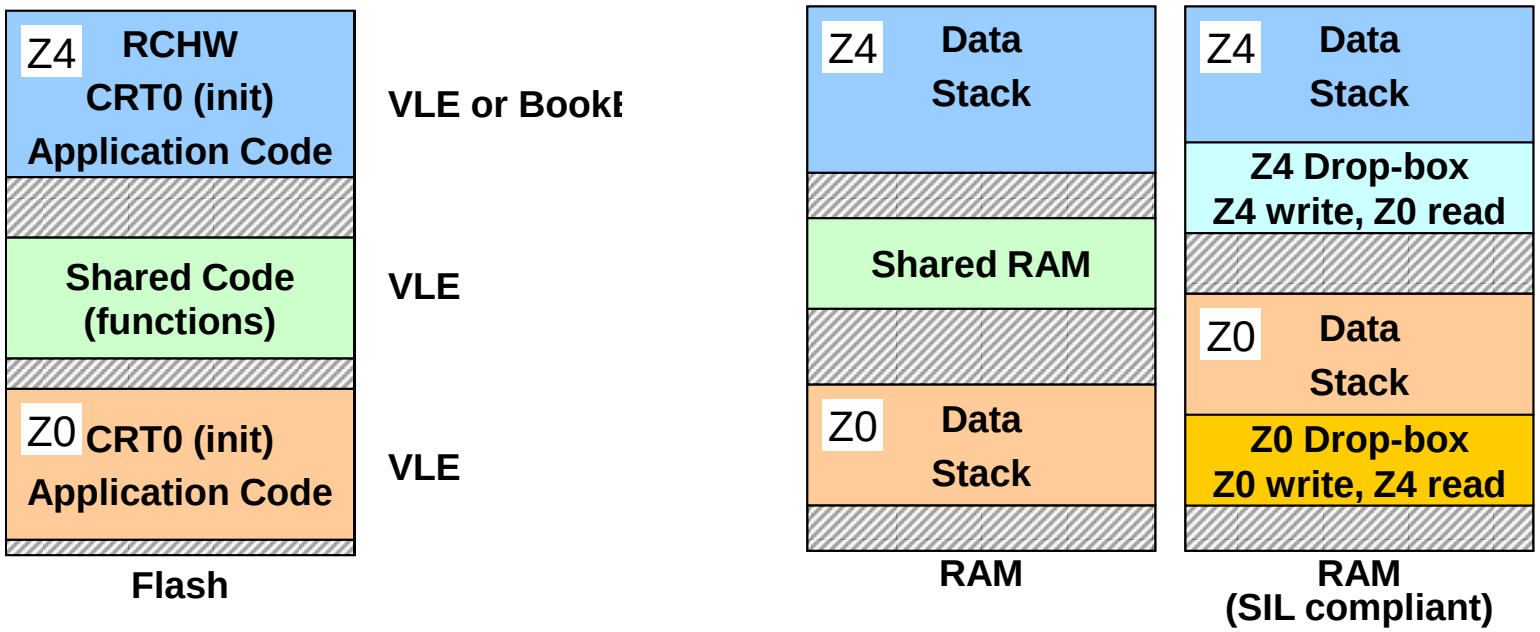
Planning: Defining the Memory Map



Planning: Defining the Memory Map

The memory map must be planned and include:

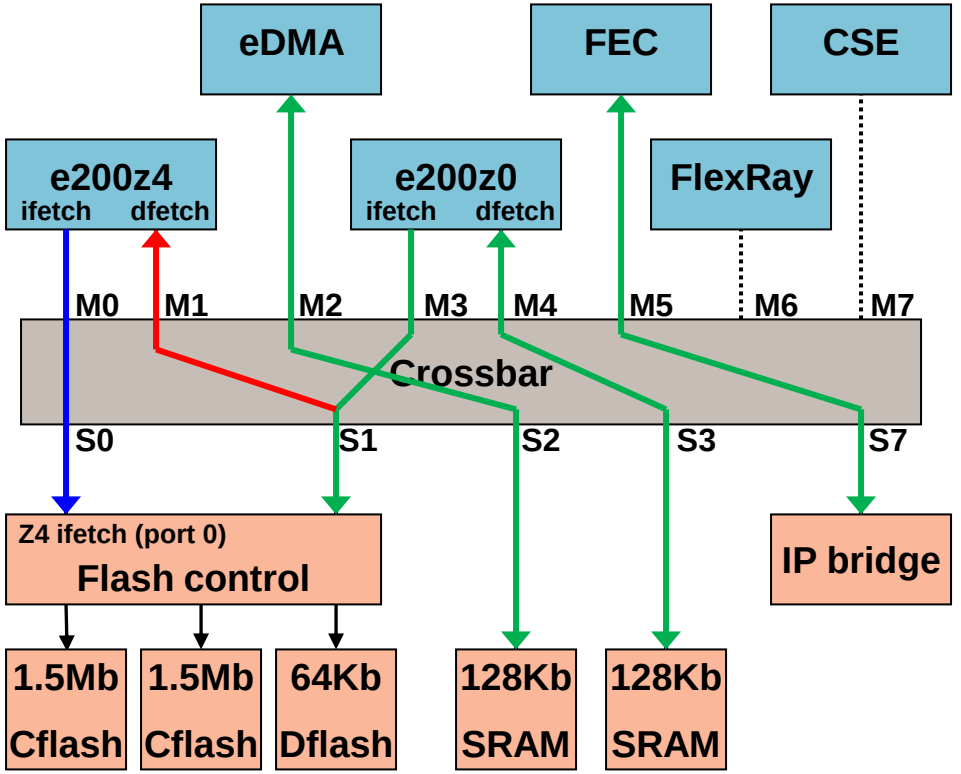
- Dedicated stack, code and data areas for each core
- Common or shared data area for shared variables
  - Note IC61508 compliance states that in multi-core systems, no more than one core can have write access to any area of memory



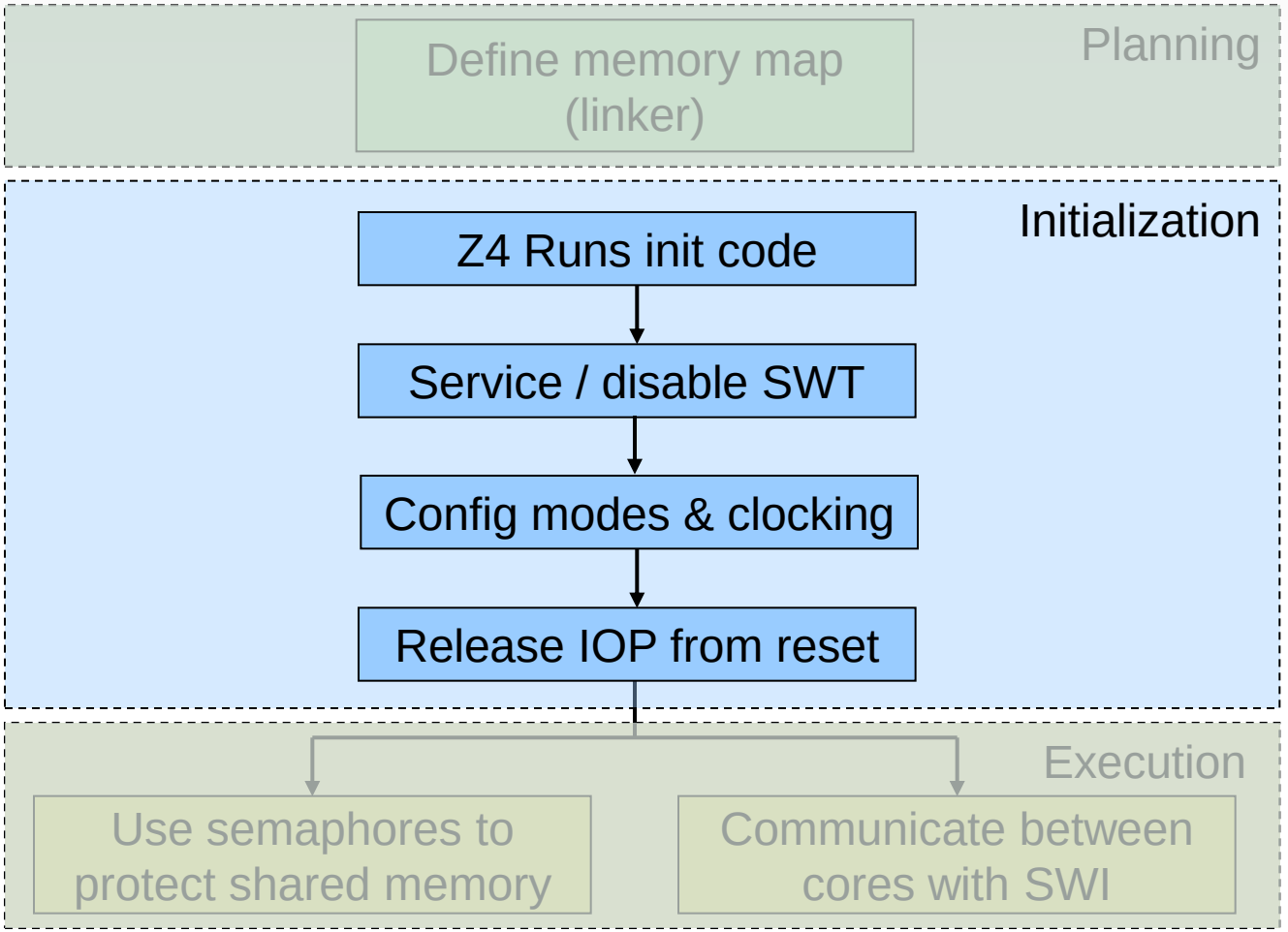
Planning: XBAR Configuration for Performance

On the MPC564xB/C devices, the crossbar and memory architecture is optimized for dual core operation:

- Dedicated z4 ifetch port
- Separate flash ports
- Separate SRAM blocks
- No contention unless attempt made to access same slave from multiple masters
- Flexible configuration to deal with arbitration and parking to maximize performance for your application



Initialization: Overview



Initialization: Initialization & Watchdog

Before the e200z0 can be released from reset, the e200z4 core must perform some system initialization and configuration tasks:

Initialization code (e200z4) — Assembly

- Initialise SRAM (ECC)
- Re-configure MMU for FLASH
- Configure MMU for peripherals
- Setup Stack / Heap
- Copy constants / initialised variables

Service / Disable the Watchdog — "C"

- Default timeout period of approximately 10ms

Initialization: Modes & Clocking / Starting e200z0

Modes and Clocking "C"

- Enable desired operating modes
  (Modes have to be enabled before they
- Enable desired peripherals in R
  (All peripherals are clock gated
- Configure peripheral group clock divi
- Configure system clock divid) for desired clock speed
- Configure clock source and PLL for
- Latch changes with mode chang

Important thing to notice is the amount of initialization required by the e200z4 before the e200z0 can run

Enable e200z0 "C"

- Write DPM boot and DPM key (see next slide)

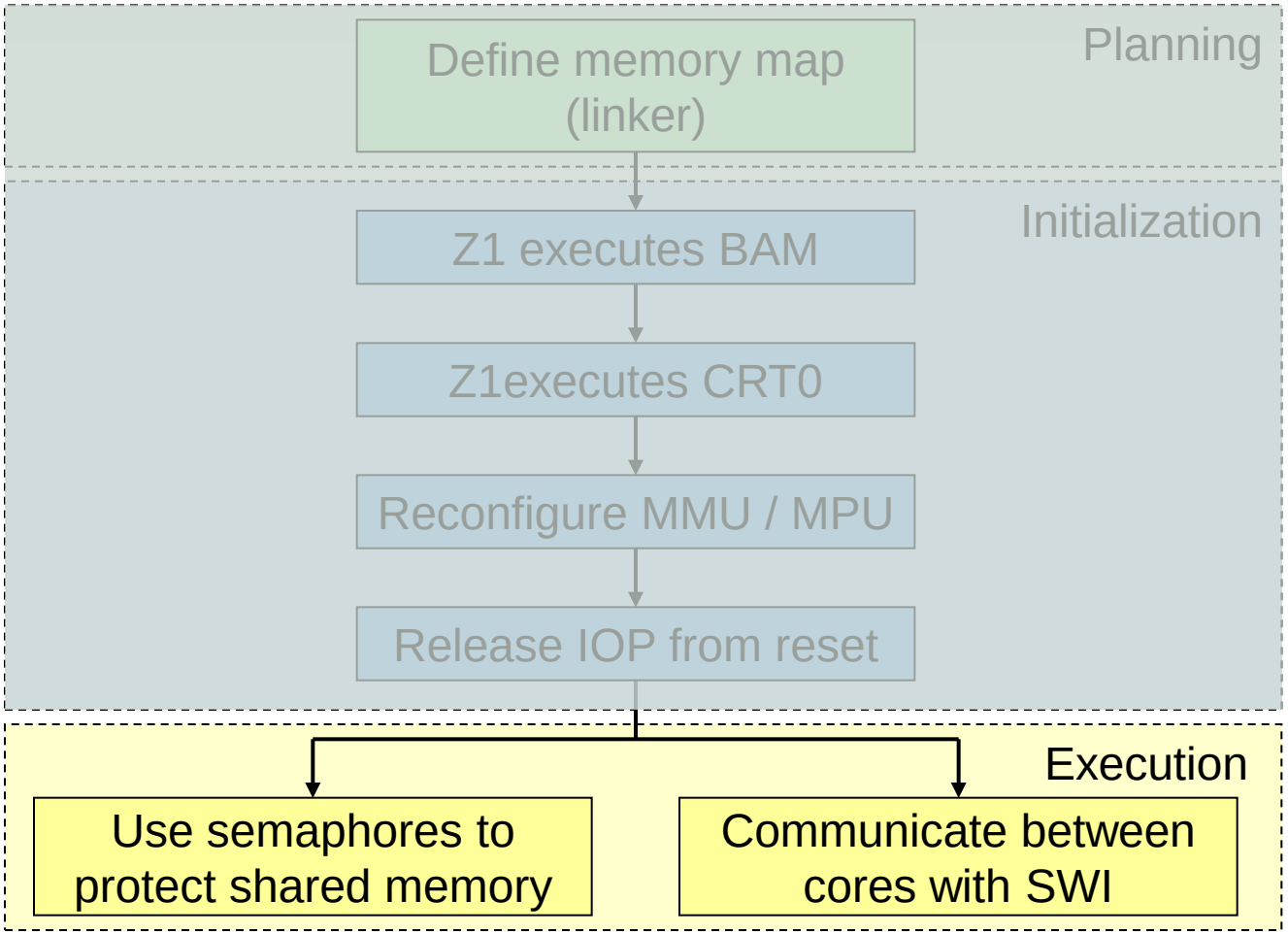
Initialization: Releasing e200z0 from Reset

- The e200z0 start address must be 4 byte aligned and written to the SSCM.DPMBOOT register
- A key and inverse key are then written to the SSCM.DPMKEY register to action the boot request

```
SSCM.DPMBOOT.R = 0x00180000;  /* Start address of Z0 (2nd flash block)  */
SSCM.DPMKEY.R  = 0x00005AF0;  /* Write key 1                            */
SSCM.DPMKEY.R  = 0x0000A50F;  /* Write key 2                            */
```

- **Important notes**
  - Once the e200z0 has been released from reset, it cannot be reset again unless the MCU is reset or enters *standby* mode
  - In *standby* mode, the core to wake-up can be selected via the SB_CPU bit within the RGM_STBY register. The core not used for boot can be enabled as above
  - One core should be the master at all times in control of operating mode transition and clock control

Execution: Semaphores and SWI



Execution: Question

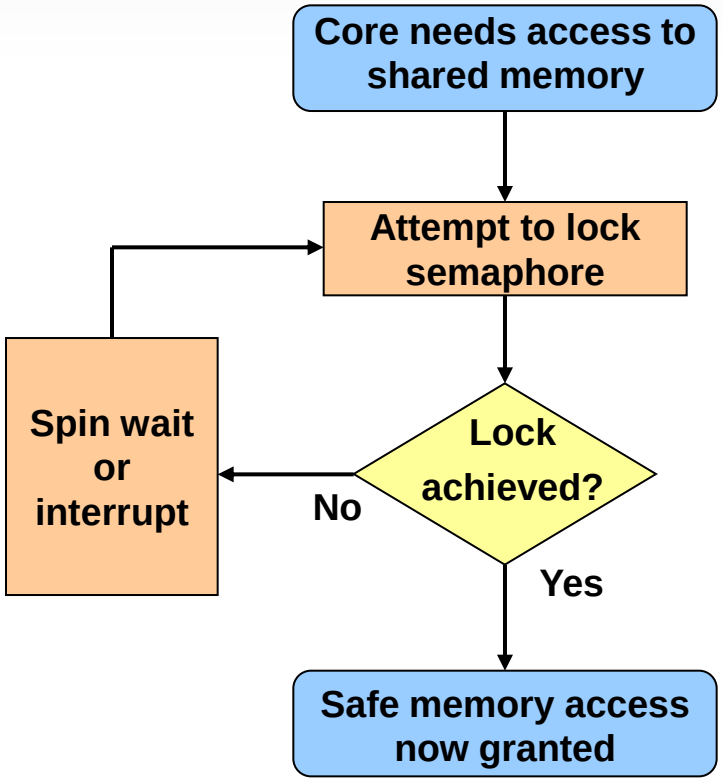
Why are semaphores / software interrupts required?

Some Answers:

- Ensures memory coherency and protection
- Cross core communication
- Task dependency / synchronization notification

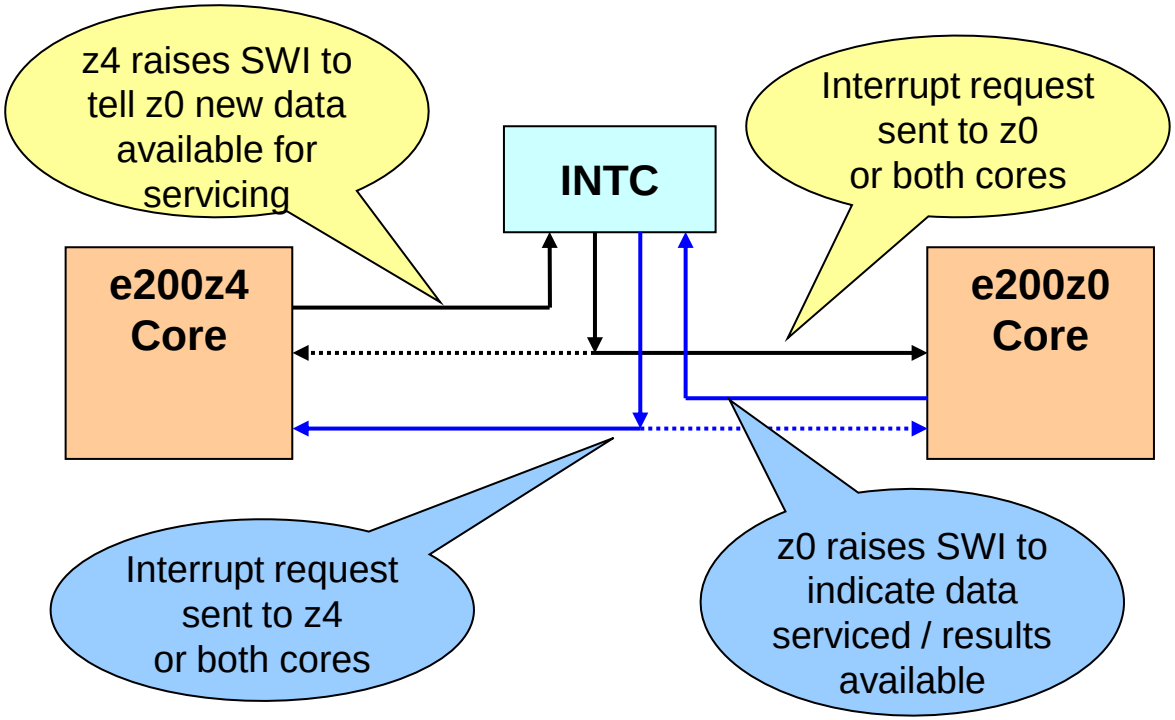
Execution: Semaphores

- Hardware enforced flags provide a mechanism for multi-processor systems to determine which core has access to shared memory
- They do **not** physically prevent the cores from accessing a particular memory region
  - Software must check the semaphores before accessing a shared memory resource
- There are 16 configurable semaphore gates on MPC564xB/C, allowing up to16 controlled shared memory regions
- Semaphores can be reset or interrupt requests cleared by using a mechanism similar to servicing a watchdog



Execution: Software Interrupts

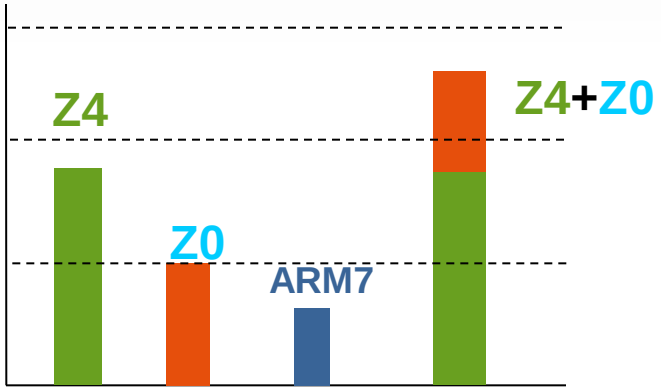
- The MPC564xB/C has 8 software interrupts that may be used to trigger cross core communication or signal to a core there is data ready to be serviced
- Each SWI has programmable priority and can be routed to both or just one of cores



Note: Interrupt vectors and handlers not shown for simplification purposes

Execution: Performance

- Note that overall system performance is not equal to 100% Z4 + 100% Z0
- Why?
  - Mainly down to system architecture
  - Any resource contention slows system down
  - Z0 clock divider means that at > 80Mhz, the Z0 runs at half clock speed
- However the architecture is optimized as much as possible for dual core performance



FTF | FREESCALE TECHNOLOGY FORUM
POWERING INNOVATION

Lighting

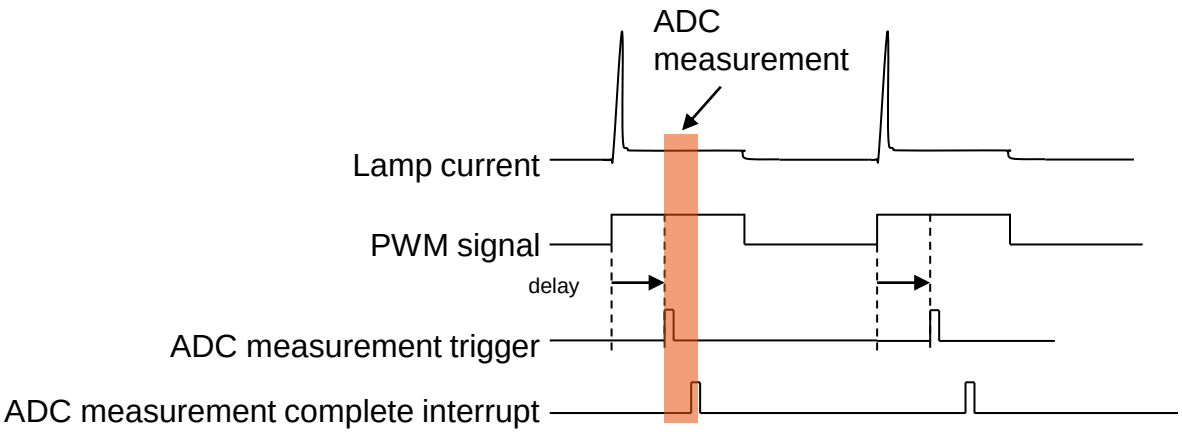
Smart Peripherals for Lighting

- Overview
  - Requirements for lighting
  - Direct and serial based PWM lamp driver control
  - Chaining lamp drivers
- Implementation
  - MCU software requirements for lamp driver control
  - Performance examples
  - Smart features on the MPC564xB/C ideal for lamp driver control
    - DSI
    - CTU
    - ADC watchdog
    - DMA channel linking

Lighting: The Requirement

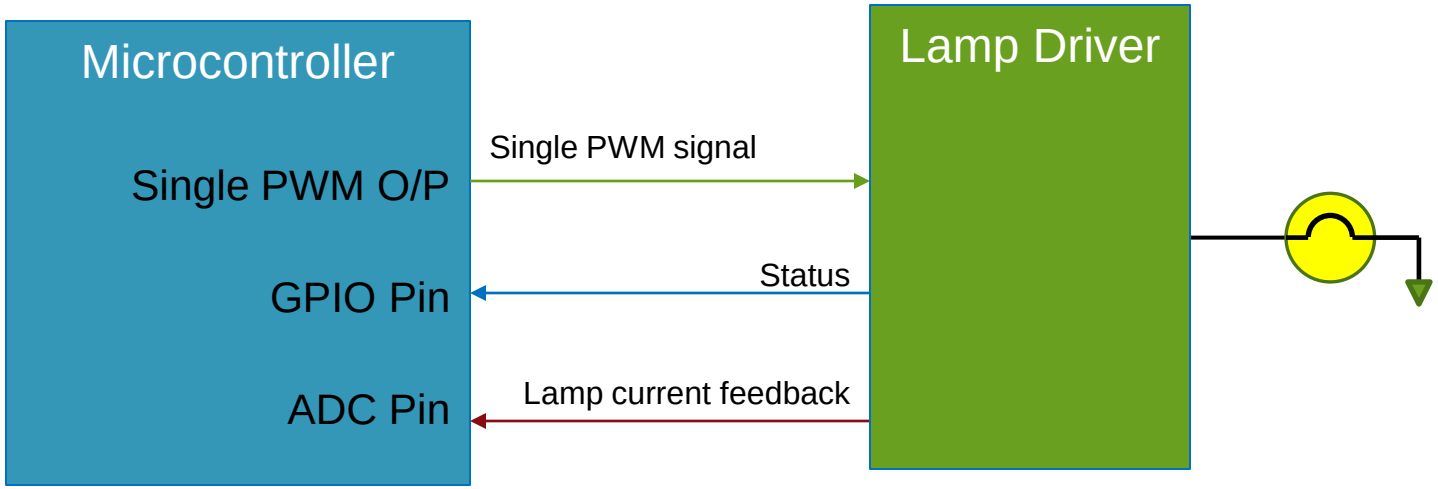
- Ability to control vehicle lighting via external lamp drivers using PWM-like signals to control the current (brightness)
  - 100Hz PWM for incandescent lighting
  - 160Hz PWM for LED lighting

  } 0.5% resolution on duty cycle
- Ability to measure the lamp current during the active phase of the PWM but after the initial inrush current
- Using as few pins and as little CPU bandwidth as possible



Lighting: Phase 0 – Theoretical Minimum

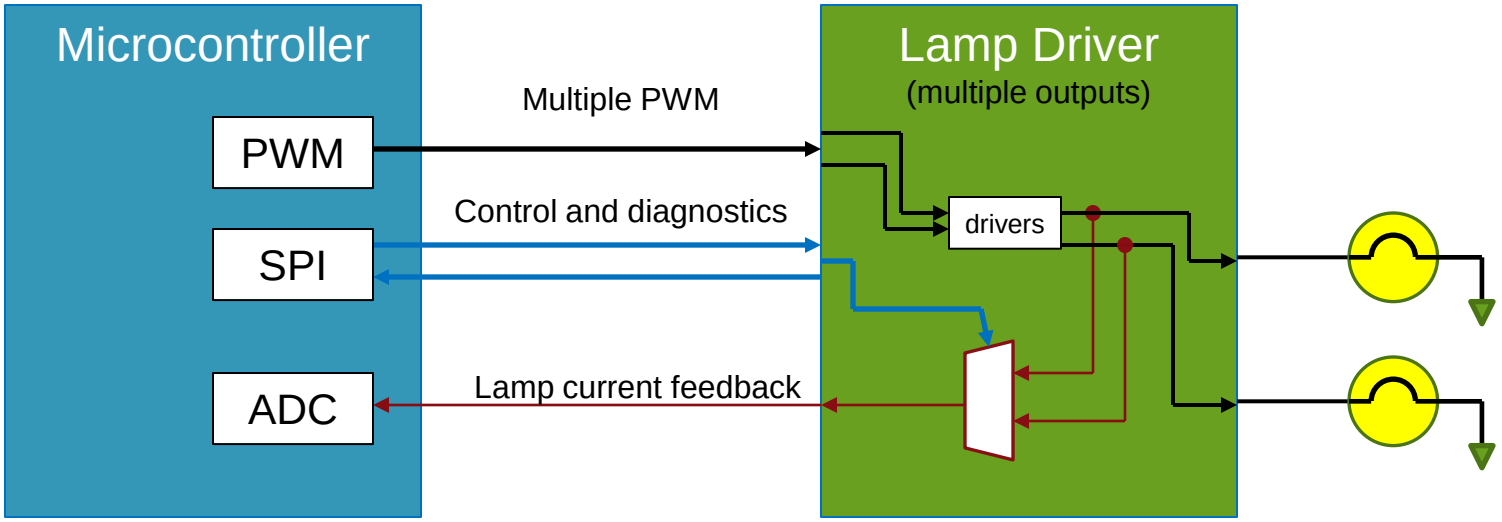
- Single PWM signal generated by MCU and fed into lamp driver
  - could be generated by hardware module or by software
- The Lamp driver provides:
  - Analog feedback to MCU ADC
  - Status feedback to GPIO pin (for error condition monitoring)



However in practice, a single output lamp driver would rarely be used!

Lighting: Phase 1 – Discrete PWM

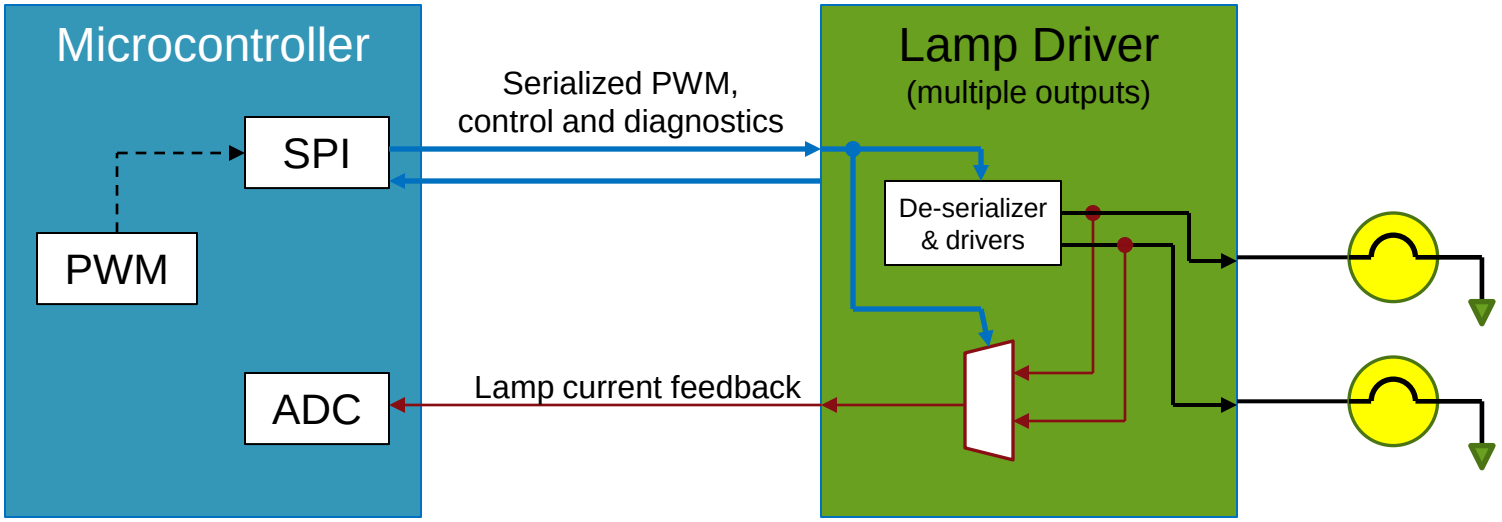
- Multiple PWM signal to control multiple lamps
  - could be generated by hardware module or by software
- Control and status via SPI
  - Single ADC feedback channel with SPI controlled mux
  - Status feedback via SPI



Lighting: Phase 2 – Serialized PWM

- Rather than discrete PWM, the PWM signals are transmitted over SPI
  - One PWM signal per bit of SPI data (example: 5-bits data can control 5 lamps)
  - Significant pin savings over discrete PWM solution

- The SPI is also used for control of ADC mux and diagnostics

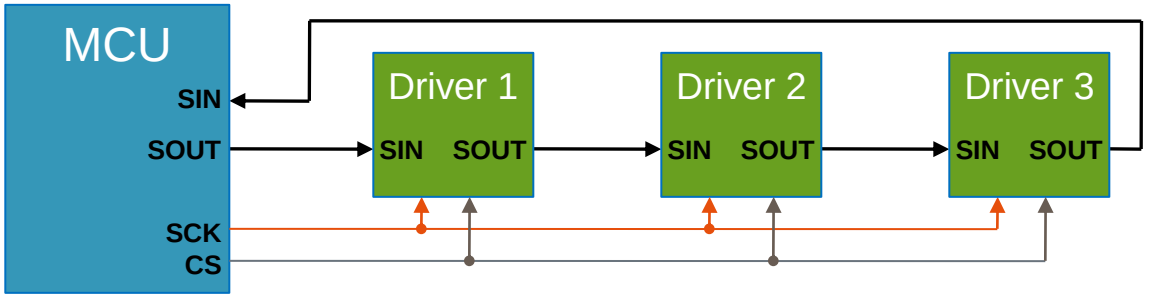


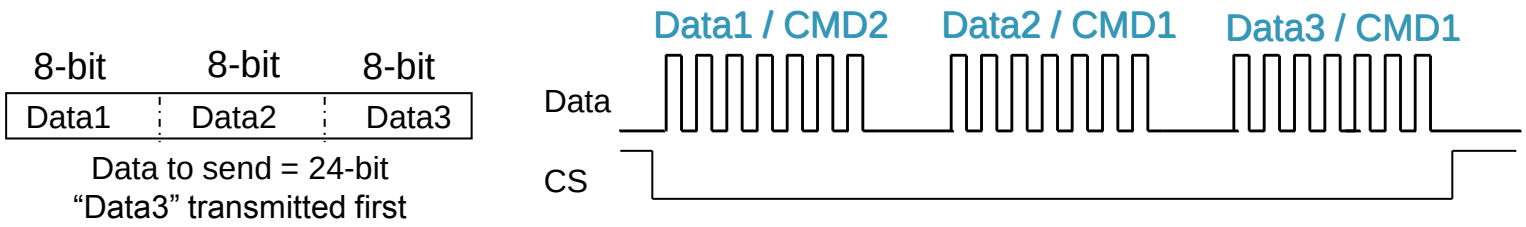
Lighting: Chaining Lamp Drivers – Part 1

- Safety requirements dictate that the left and right side of the vehicle have independent light control circuits
- However within the same side of the vehicle, lamp drivers are often daisy chained
  - Typical lamp drivers have 8-bits of SPI data (some have 16-bits)
    - 3-5 bits associated with corresponding lamp outputs (1 bit per output)
    - Bit(s) to control whether the data is a PWM output or a control signal

Lighting: Chaining Lamp Drivers – Part 2

- The example below shows how to chain 3 drivers (each with 8-bit SPI)
  - Common chip select and clock (saves pins over multiple CS implementation)
  - 3 SPI frames are transmitted:
    - Frame1: SPI command asserts chip select. Data is for lamp driver3 (but enters driver1)
    - Frame2: CS still asserted. Data is for driver2
    - Frame3: SPI command de-asserts chip select at end of frame. Data is for driver1





Significant pin savings are achieved using serialized, daisy chained lamp drivers compared to discrete PWM-based solution

Lighting: MCU High-Level Requirements

- With serialized PWM, the MCU needs to:
  - Take lighting events (brake signal, turn signal) and turn these into serialized PWM events to be routed to the relevant lamp driver
  - Write out the SPI PWM data at a rate sufficient to meet the PWM frequency and resolution. Safety requirements dictate that PWM signals are still transmitted even if PWM is in static state
    - *For 100Hz PWM signal with 0.5% resolution, SPI transmission period is 50us (10ms @ 0.5% resolution)*
  - Write out SPI frames to configure the ADC mux
  - Read back the ADC data from the lamp driver and modify the PWM waveform if required

- If this is done totally in software, there is a significant CPU bandwidth requirement

Lighting: MPC564xB/C Implementation

So how is all of this implemented?

What features of the MPC564xB/C can help me?

- Looking at some examples, we will assume:
  - The lamp drivers are fully SPI based (serialised PWM)
  - Lamp drivers are daisy chained (8-bits SPI data per lamp driver)

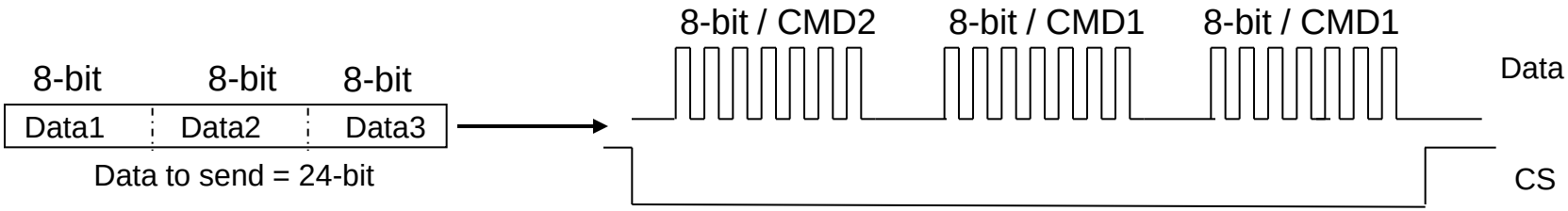
Lighting: Virtual PWM RAM Table

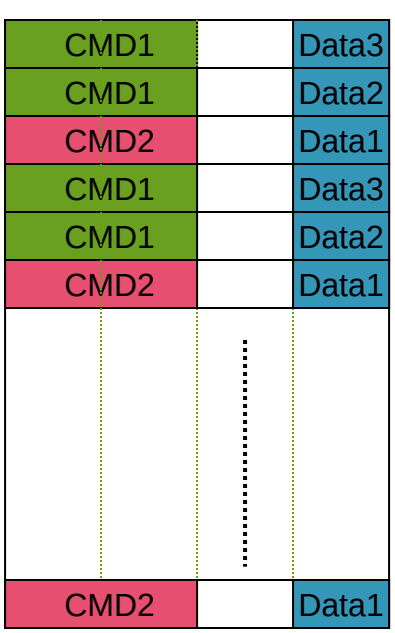
- With a 100 MHz PWM signal at 0.5% resolution, the PWM waveform is made up of 200 points
  - In reality, there are a maximum of two transitions per period (on and off)
  - One way to easily represent the waveform is via a 200 entry table in RAM which is then copied to the SPI at the desired rate
    - Easy for software to update the table but demands regular data transmission to SPI
- For serial chained lamp drivers, the table increases based on number of drivers
- On the MPC564xB/C, the **DMA** can be used to continually transfer the contents of the RAM table to the DSPI with no CPU intervention once setup
  - CPU is then left to service the RAM table

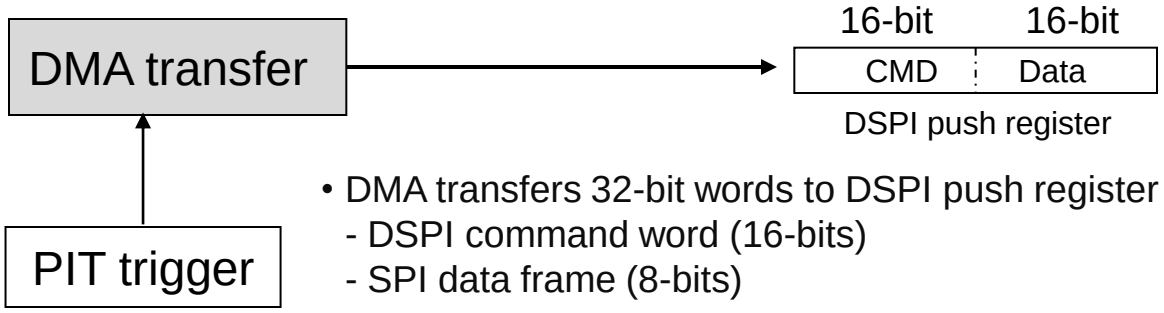
Lighting: MPC564xB/C Example 1

RAM table contains full DSPI frame (command and data)

- Single PIT triggered DMA channel used to transfer DSPI frame to push register





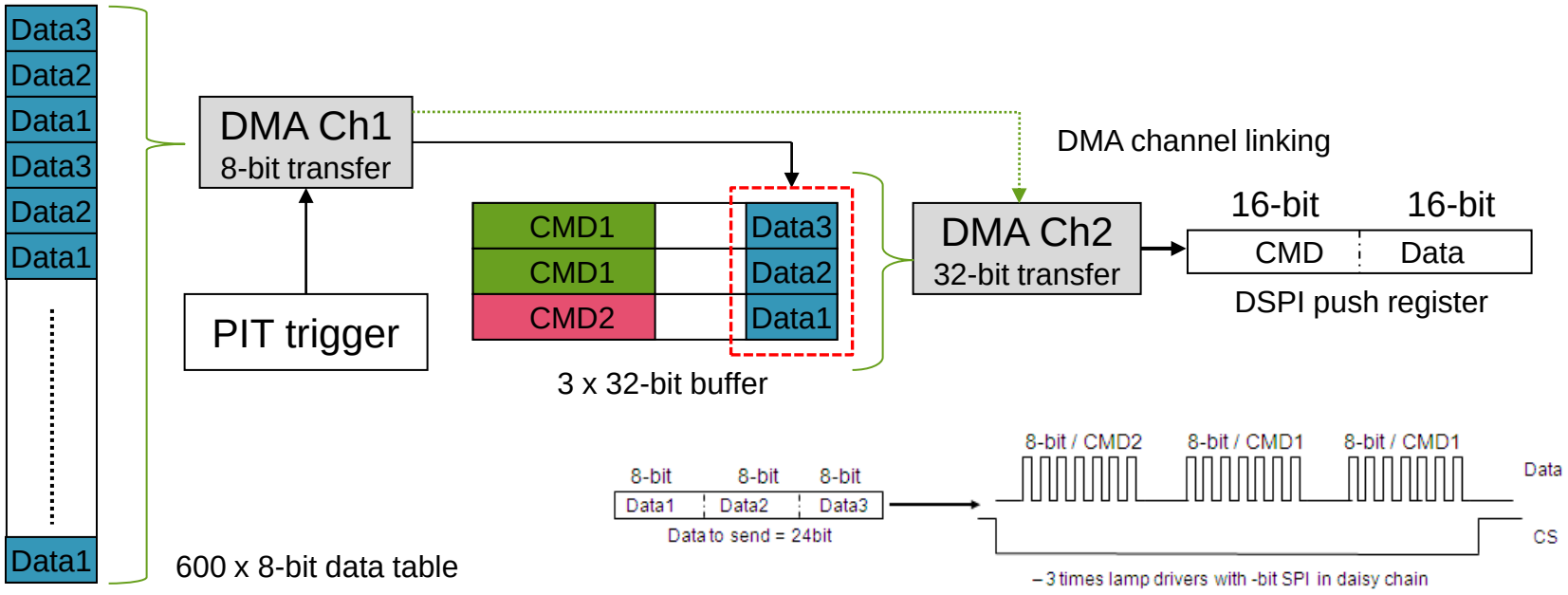


- DMA transfers 32-bit words to DSPI push register
  - DSPI command word (16-bits)
  - SPI data frame (8-bits)
- Wasted space in RAM table with this approach
  - Command words 1 & 2 are never changed
  - 8-bits space not used between CMD and data
- For a 600 entry table (3 lamp drivers), this wasted space equates to 1800 bytes

Lighting: MPC564xB/C Example 2

RAM table only contains 8-bit data values

- Separate 3x32-bit RAM buffer containing command words
- 2 DMA channels (with channel linking) used to form SPI frames and fill push register
  - DMA CH1 transfers 8 bytes data from data table into 3-entry 32-bit buffer
  - DMA CH2 (triggered by CH1) then transfers full 32-bit DSPI frame to push register
- Only **3 bytes RAM wasted** per 3 lamp driver chain but more DMA bandwidth used



Lighting: Improving Performance with MPC564xB/C

- In previous examples, the CPU still has to:
  - Update the table in RAM to modify the PWM signal
  - Manage the ADC MUX configuration and trigger the ADC measurements

- The MPC564xB/C has some additional features to help manage this:
  - CTU (cross trigger unit)
  - ADC watchdog
  - DSI mode
  - Automated de-serialization

Lighting: Cross Trigger Unit (CTU)

- CTU Features
  - Uses the eMIOS OPWMT (PWM triggered) mode which can generate a CTU trigger at any point in the PWM period
    - The trigger point is set to measure the lamp current after initial inrush current
  - The CTU uses the trigger to start a single ADC conversion on a nominated ADC channel.
  - All of this occurs with no CPU intervention

- However…
  - This relies on the eMIOS being used to generate the PWM signal
  - In examples above, the PWM was manually generated using a RAM table sent to the DSPI

Lighting: ADC Watchdog

- ADC Watchdog Features:
  - Allows an upper and lower measurement “watchdog” to be set
  - If a measurement is received out-with these limits, an interrupt can be triggered
  - System does not have to process ADC results unless they are out of spec
  - Further reduction in CPU bandwidth requirements

Lighting: Direct PWM Serialization (DSI Mode)

- DSI mode provides automatic serialization of eMIOS channel outputs
  - All 32 outputs from eMIOS_0 can be serialized (intended for OPWMT mode)
  - Configurable eMIOS channel muxing determines bit order
  - 32-bit SPI frames are supported
- Automated process
  - Removes the need for a virtual RAM table and manual DSPI transmission
  - Each time an eMIOS output changes, a new DSPI frame is transmitted
    - CTU can be used to trigger ADC conversion
    - ADC watchdog feature can be used to monitor current feedback results
- CPU required for:
  - Updating the PWM signals directly (duty cycle)
  - Configuring the lamp driver ADC mux (via DSPI CSI or ASDR frames)
  - Acting on any out of spec ADC measurements

Lighting: DSPI De-serialization

- Allows automatic monitoring of the returned DSPI data
  - Once all bits have been received, they are stored in a de-serialized data register (DSI.DDR)
  - This register can be compared automatically with
    - Data Polarity (value) Register (DPIR) ⟶ Bitwise compare (XNOR)
    - De-serialized Mask Register (DIMR) ⟶ Bitwise mask (AND)
  - If there is a bit-wise mismatch between the polarity data and received data (and the bits are not masked), the DDIF flag is set
    - Interrupt can be enabled on DDIF flag event
- An interrupt can be triggered if the returned data does not meet certain constraints, otherwise no CPU processing is required.
  - Useful for automated monitoring of lamp driver status return data
- OEMs are currently running trials with these new features and expect to see significant reductions in CPU bandwidth requirements

Lighting: DSPI De-serialization 2

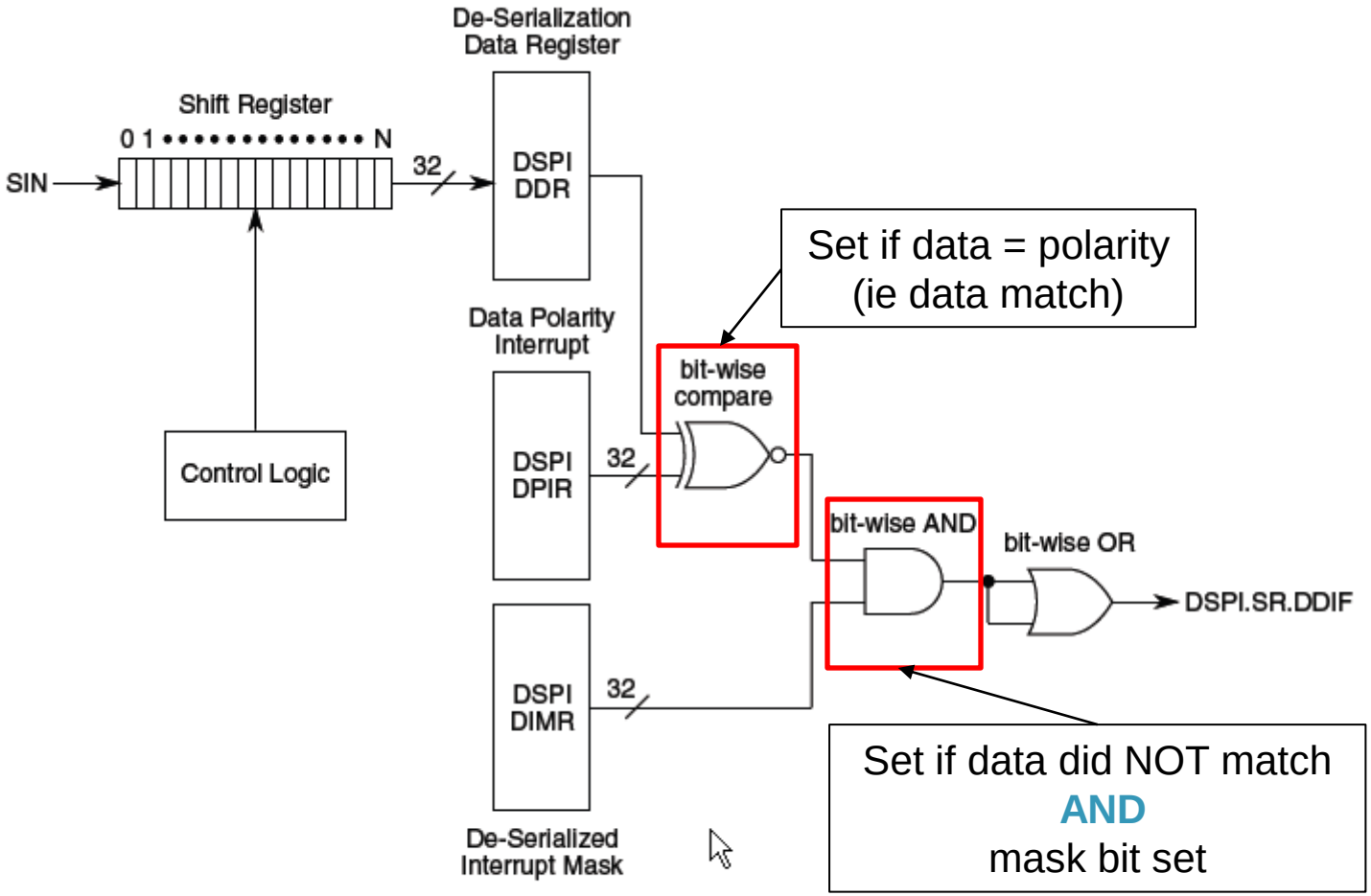


Figure 6. DSPI ASP deserialization block level view

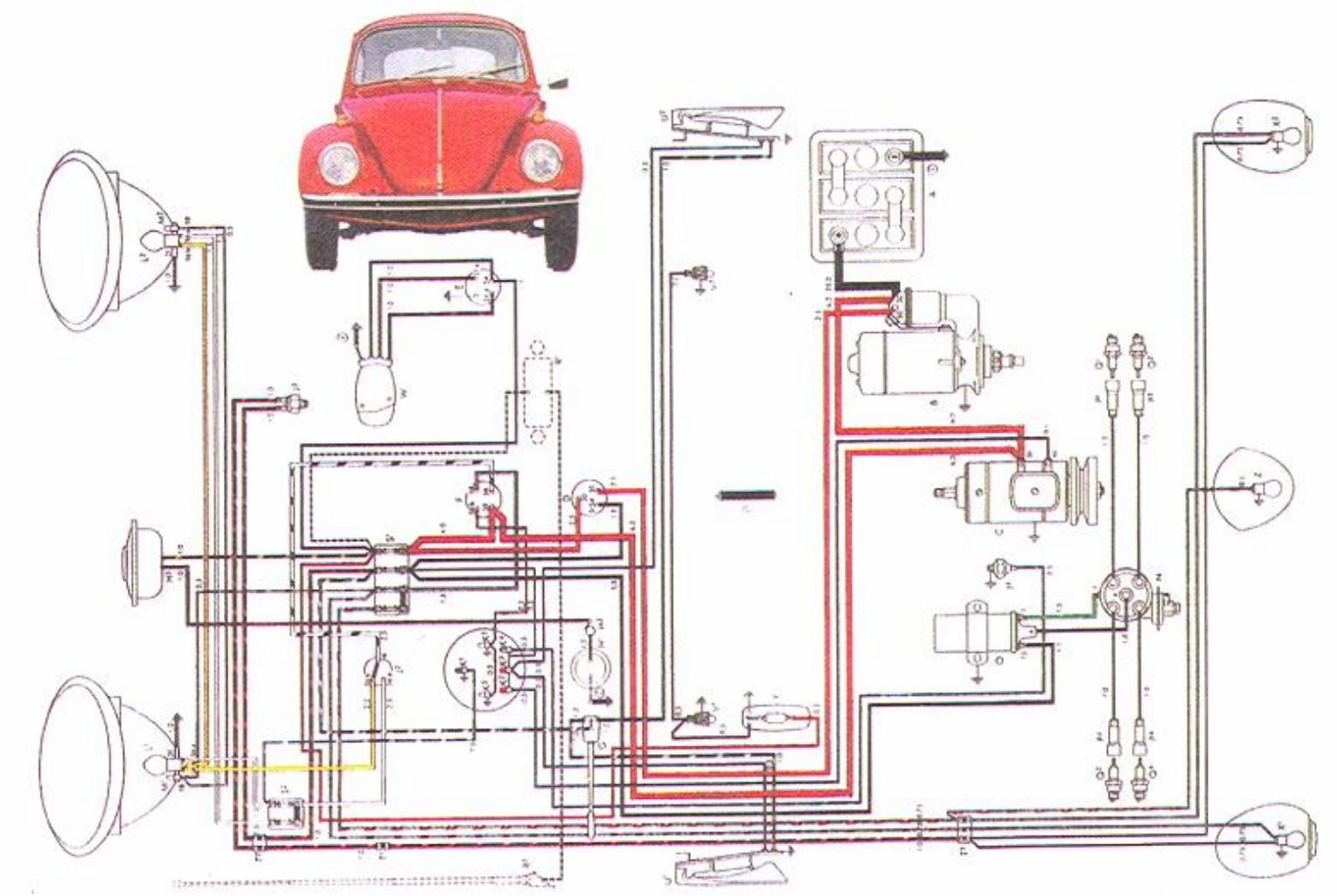
FTF | FREESCALE TECHNOLOGY FORUM
POWERING INNOVATION

Gateway

Gateway

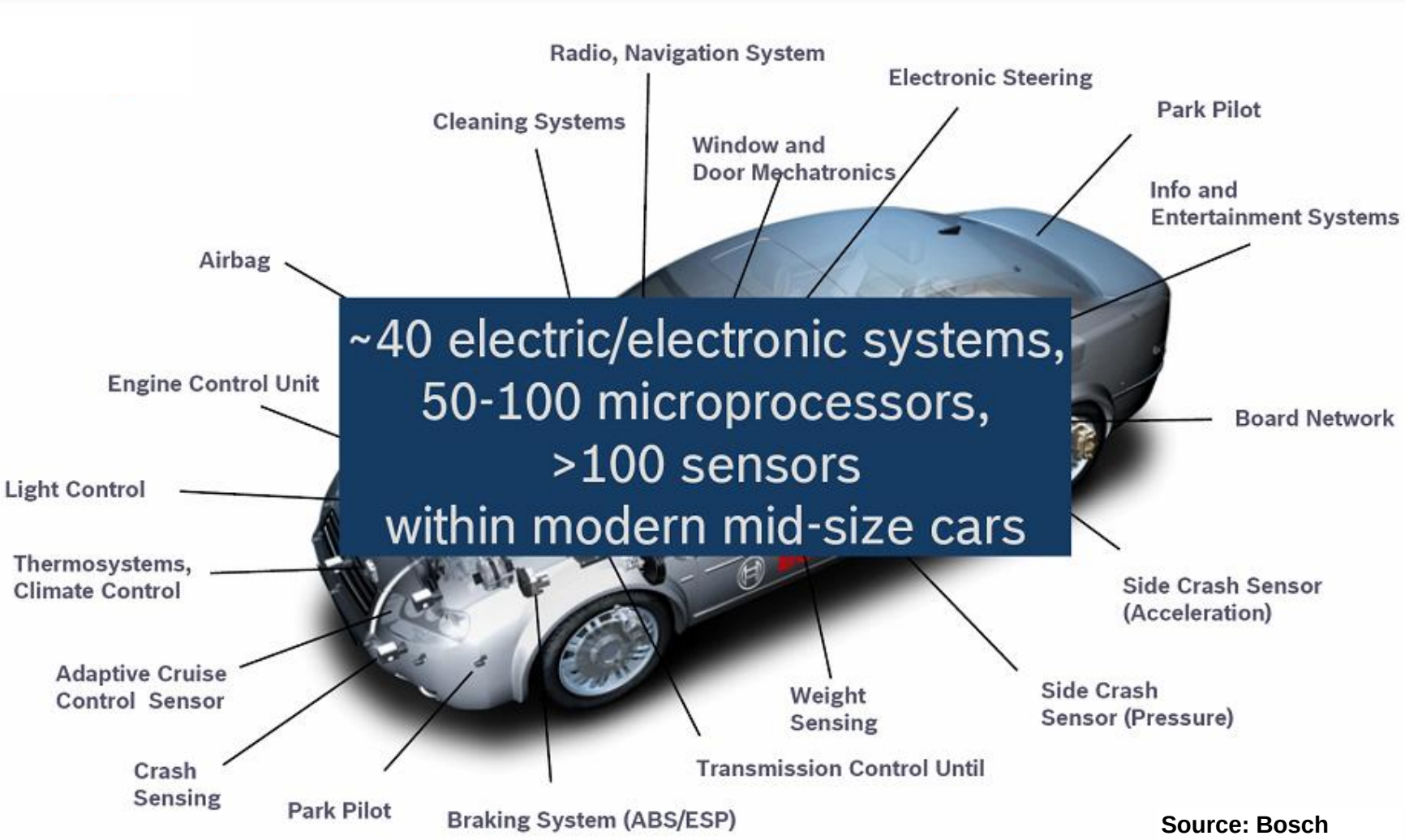
- Overview
- Future gateway systems
- MPC564xB/C features for gateway

Gateway: History of Automotive Networks: Discrete Wiring



Source: Bosch

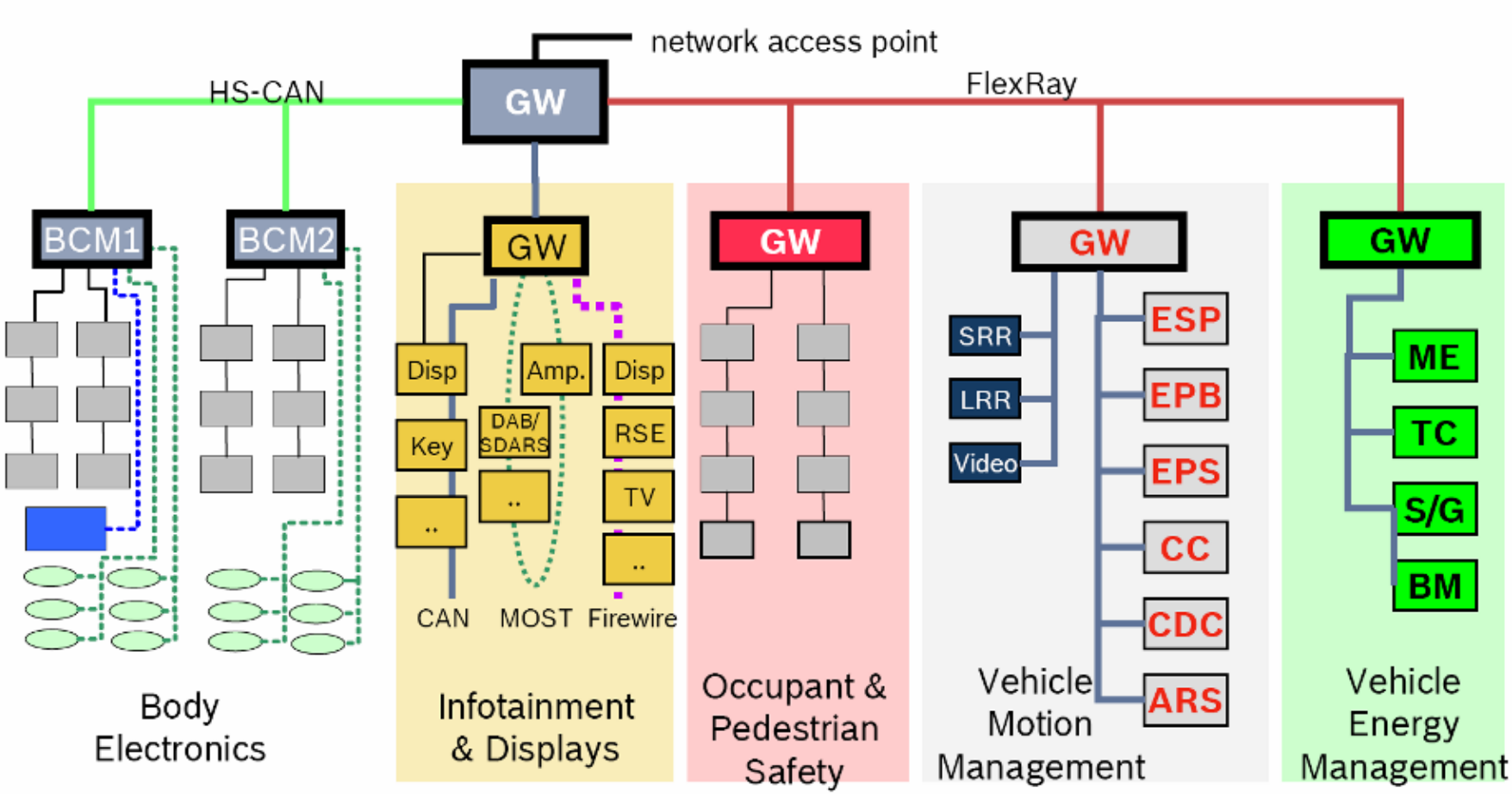
Gateway: Today: Complex Electric / Electronic System



Source: Bosch

Source: Public presentation of
Th. Lindenkreuz (Bosch) at
electronica automotive conference 2006

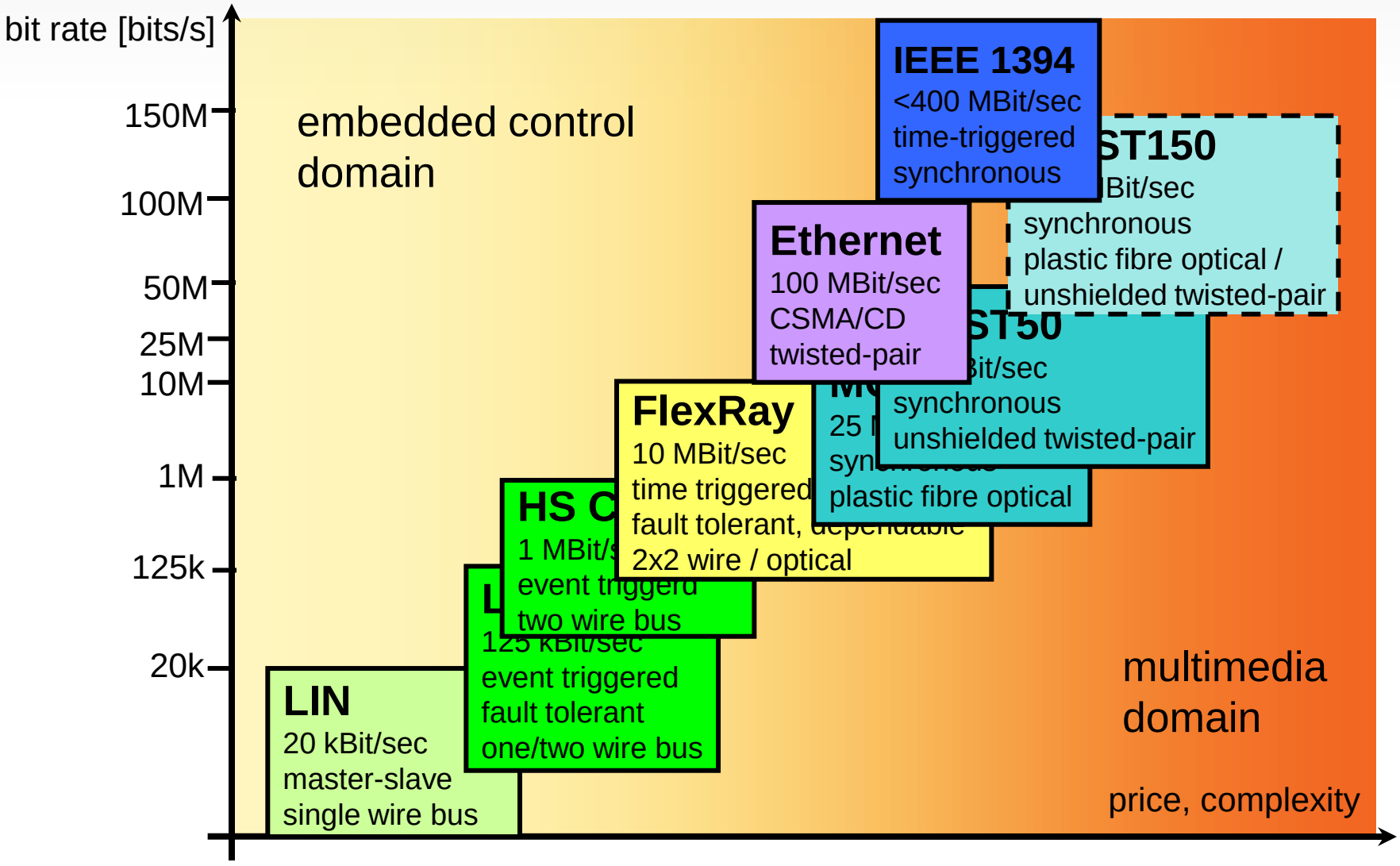
Gateway: Today & Near-Future Architecture: Bosch Vision



More or less centralized cross domain communication

Source: Bosch
Public presentation of
Th. Lindenkreuz at electronica
automotive conference 2006

Gateway: Communication Protocols Landscape



Gateway: MPC564xB/C for Gateway

- Ideal for mid- to high-end gateway applications:
- Communications
  - LIN, Ethernet, FlexRay
- Dual Core
  - Software message filtering
  - Consistency checking of messages
  - Simple routing
    - Frame based (receive a frame on one bus and retransmit on another)
    - Signal based (consolidate data from multiples buses / messages into single frame)
- Other
  - Low power mode support with CAN sampler wakeup
  - Sufficient bandwidth to support some BCM and gateway features

Summary and Q&A

- The Freescale product portfolio offers good choices of devices specifically tailored for automotive body applications
- The high-end MPC564xB/C devices offer:
  - High-performance architecture with rich feature set
  - Smart peripherals, DMA and IOP to offload core tasks
  - Smart DSI mode with CTU and ADC watchdog for lamp driver control
  - LIN, CAN, FlexRay and Ethernet make this ideal for gateway
- Combining these features allow a master gateway controller to also have sufficient bandwidth for body tasks like lighting control

Session materials will be posted @
www.freescale.com/FTF
Look for announcements in the FTF Group
on LinkedIn or follow Freescale on Twitter


freescale™