MPSoCs for Mixed-Criticality Systems: Challenges and Opportunities

Mohamed Hassan
Towards Mixed Criticality

1943: Colossus
1981: IBM's Acorn
1989: NEC's UltraLite
2000s: Smart Phones
2010s: Wearables
Now-Near: IoT/Smart Homes
Towards Mixed Criticality
Easy Tasks of Yesterday and the Challenges of Tomorrow

Up until recent years:
- Small inputs
- Small networks
- No/limited real-time use cases

From today onwards:
- Large inputs, image/video processing
- Very deep networks
- Safe, real-time embedded apps

None of today’s hardware can solve the challenges we are facing
• No longer solely hosting isolated safety-critical tasks
  • Execute tasks with different criticalities
  • Criticality $\alpha$ consequences of failure to meet requirements

High-criticality tasks
  • Airbag Control Unit (ACU)
  • Anti-lock Braking System (ABS)
  • Engine Control Unit (ECU)
• No longer solely hosting isolated safety-critical tasks
  • Execute tasks with different criticalities
  • Criticality $\alpha$ consequences of failure to meet requirements

---

**Medium-criticality tasks**
- Navigation System
- Instrument Cluster
- Cruise Control

---

Mixed Criticality Systems
Mixed Criticality Systems

**MOTIVATION**

- No longer solely hosting isolated safety-critical tasks
  - Execute tasks with different criticalities
  - Criticality $\alpha$ consequences of failure to meet requirements

- **Low-criticality tasks**
  - Air Conditioning Unit
  - Connectivity Box
  - Infotainment Unit
Increased need for performance and mixed criticality as we move from assisted to autonomous driving systems.
Mixed Criticality Systems
Why MPSoCs?

• Low cost
• High performance
• Energy Efficiency
• Low time-to-market (3rd party IPs)
• Simplicity and Modularity
MPSoCs

**MOTIVATION**

Hennessy & Patterson, Turing Lecture, A New Golden Age for Computer Architecture
Heterogenous MPSoCs

MPSoCs

Shared cache(s)

Memory Controller

Off-chip Memory/ies

PE_1

PE_2

PE_3

PE_4

PE ...

PE_P

Shared IO

Heterogenous MPSoCs

Shared cache(s)

Memory Controller

Off-chip Memory/ies

CPU

ASIC1

FPGA

ASIC2

GPU

DSP
Why Heterogenous MPSoCs?

- Variety of processing capabilities
  → Best-suits MCS conflicting requirements
Complementary SoC processor requirements

- High performance compute
- Infotainment
- Cluster
- Driver assist
- Vehicle interface
- User experience

Cost     Quality     Ecosystem     Temperature

Real-time control
- Safe
- Secure
- Responsive
- Reliable
- Fast boot

Translating System-Level Requirements → SoC Level

- Exploding Performance Requirements
  - Rise of heterogeneous architectures & right-sized compute
  - Cache coherency & End-to-end QoS of critical importance

- Real-Time Sensor Processing
  - Different IPs with differing requirements
  - Ensuring communication happens without any deadlocks

- Ultra-High Safety & Reliability
  - Pressure to comply to industry standards – ISO 26262
  - Functional Safety – Performance – Area Tradeoffs

Automotive Applications Require Different SoC Architectures

High-End ADAS
- UFD01, Ethernet AVB, MPI, HDMI, PCIe, SATA, ADC
- Embedded Vision
- Security
- Sensor Fusion
- Requires Functional Safety

Infotainment
- USB, UFD01, Ethernet AVB, MPI, HDMI, PCIe, SATA, ADC, UFS, eMMC
- Real-time Multimedia
- Security
- Sensor Fusion

MCU
- IP: Ethernet 10/100/1000, ADC, I²C peripherals
- Medium Density HM

Need For Heterogeneous Computing

- Image Acquisition: Noise removal, Pixel processing, Image pyramids
- Feature Extraction: Optical flow, Edge detection, Gradient detection
- Feature Processing: Segmentation & filtering, Object tracking, Object detection
- Pattern Recognition: Feature reduction, Feature classification, Augmentation
- Feedback and Action: Computation & processing, Feedback loop, Avoidance signalling

- CPU
- DSP, Accel
- GPU, ISP

- Smaller amounts of data
- Highly structured data
- Complex computation/Item
- Lots of data
- Simple computation/Item
- Massive parallelism
Complementary SoC processor requirements

High performance compute
- Infotainment
- Cluster
- Driver assist
- Vehicle interface
- User experience

Cost  Quality  Ecosystem

Compute, Control, Sense
Real-time control

Translating System-Level Requirement

Exploding Performance Requirements
- Rise of heterogeneous computing
- Cache cold

Real-Time Sensor Processing
- Different I/O
- Ensuring high-speed processing

Ultra-High Safety & Reliability
- Pressure to comply to industry standards - ISO 26262
- Functional Safety - Performance - Area Tradeoffs

Safety and Security

Automotive Applications Require Different SoC Architectures

High-End ADAS  Infotainment  MCU

ARM® Cortex®-A  ARM® Cortex®-R  ARM® Cortex®-M

Computing

- Smaller amounts of data
- Highly structured data
- Complex computation/item

- Lots of data
- Simple computation/item
- Massive parallelism

Feedback loop
- Pattern recognition
- Feedback and action
- Noise removal
- Segmentation & filtering
- Feature reduction
- Image pyramids
- Computation & processing
- Gradient detection
- Object detection

Source: Synopsys, ARM

©ARM 2016
Heterogenous MPSoCs with Real-time Processors
Heterogenous MPSoCs with Real-time Processors
Where Are We?

WMC13-17

Uniprocessor: 23
Multiprocessor: 19
45%
Where Are We?

WMC13-17
Where Are We?

WMC13-17

Uniprocessor: 23
Multiprocessor: 19 (45%)
- Shared Resources: 13 (~31%)
- SoCs: 1 (~2%)
Since Vestal’s 2007 paper, there has been a series of publications. Most of these papers address single processor platforms and independent components.

Overall, MCS review [Burns and Davis]
Where Are We?

*Overall, MCS review [Burns and Davis]*
Where Are We?

*Overall, MCS review [Burns and Davis]*
MPSoc-Based MCS: Four Aspects

- Traditional MCS Model
- Timing Interference
- Data Sharing
- Security
Traditional Model

WCET (LO)

normal mode

degraded mode
Traditional Model

- WCET (LO)
- WCET (HI)

normal mode

degraded mode
Problems with the Model

**Suspension**
- Lower-criticality tasks are suspended upon switching to a higher-mode (not acceptable in industry [P. Graydon and I. Bate, WMC 2013])

**Overheads**
- Switching leads to huge overheads (usually overlooked) [L. Sigrist et al, RTAS 2015]

**Sources of Uncertainty**
- Of special importance for MPSoCs (more next)
1. MPSoCs create switching alternatives

- Different modes of operation at different cluster of PEs?
1. MPSoCs create switching alternatives

- Different modes of operation at different cluster of PEs?

Flexible software implementations

- Software complexity increases with mixed criticality applications
- Split-Lock on Cortex-A76AE is designed to be transparent to software
- Armv8.2 architectural support for virtualization and Type-2 hypervisors
1. MPSoCs create switching alternatives

- Different modes of operation at different cluster of PEs?
- Migrate instead of switching?
  - Dynamic Reconfiguration (IEC61508-7)

**C.3.13 Dynamic reconfiguration**

*The logical architecture of the system has to be such that it can be mapped onto a subset of the available resources of the system. The architecture needs to be capable of detecting a failure in a physical resource and then remapping the logical architecture back onto the restricted resources left functioning. Although the concept is more traditionally restricted to recovery from failed hardware units, it is also applicable to failed software units if there is sufficient ‘run-time redundancy’ to allow a software re-try or if there is sufficient redundant data to make the individual and isolated failure be of little importance. This technique must be considered at the first system design stage.*
1. MPSoCs create switching alternatives

- Different modes of operation at different cluster of PEs?
- Migrate instead of switching?

2. MPSoCs open the door for customized solutions

- Using specialized PEs is a norm in MPSoCs
- Dedicating a PE for the runtime monitoring
  - faster detection of exceptional events → react in a timely manner
- PE can be further tailored to optimize the behavior of the monitoring techniques
1. Common assumption: “uncertainty in WCET does not come from the system itself; rather, it comes from our inability to measure (or compute) it with complete confidence”

• Well, this may not be completely true for MPSoCs
  ➢ In SMPs, which core (or cores) executing a task does not affect its measured execution time.
  ➢ In MPSoCs, this decision directly affects the level of certainty in its WCET:
    Real-time vs High-performance PEs?
    Use scratchpads vs caches?
2. Scalability challenges associated with these scheduling and monitoring techniques.

3. Mode switching in MPSoCs may incur task migrations or reassignment of heterogeneous cores to tasks

➢ the effects of these decisions on the switching overhead need to be quantified.
MPSoC-Based MCS: Four Aspects

- Traditional MCS Model
- Timing Interference
- Data Sharing
- Security
Challenge: operations of one PE affect the temporal behavior of other PEs, which complicates the timing analysis of the system.

Most of the MCS scheduling techniques do not incorporate these interferences in their scheduling or analysis.

Approaches focusing on shared resources mostly assume SMPs.
Challenge: operations of one PE affect the

7.4.2.7 Where the software is to implement both safety and non-safety functions, then all of the software shall be treated as safety-related, unless adequate independence between the functions can be demonstrated in the design. [IEC61508-3]
1. Which memory levels should be shared amongst which cores
   • Does the GPU share the LLC with the CPU?

2. How to distribute the cache architecture?
   • Would implementing a NUCA be adequate for MCS (e.g., helping in achieving different levels of isolation)?

3. Different types of on-chip memories
   • Both caches and SPMs
   • Most of the currently available approaches focus on a single type

4. Different types of available off-chip memories
   • DDR, GDDR, RLDRAM, LPDDR, QDR.
   • Investigating the cooperation of these types is also worth investigating
MPSoCs Challenges

1. The interference exaggerates with the increase in the number of PEs
2. Understanding the architectural details of shared resources is inevitable to derive realistic bounds.
3. Each type of PEs has its own memory access behavior, which complicates the analysis, leading to more pessimism
   - Data-intensive PEs (e.g. multimedia/DSP processors) can saturate system queues
   - A requirement- and criticality-aware arbitration is a must to deliver differential service to PEs
task = \langle CL, D, WCET(CL) \rangle

calculated/measured in isolation

Traditional Model

Timing Interference
Extending Traditional Model

**Task**

\[ \text{task} = \langle \text{CL}, D, WCET \rangle \]

**Calculated/Measured in Isolation**

\[ \text{WCET} = \text{WCCT} + \text{WC \# reqs} \times \text{WC Interference} \]

**Guaranteed Bound by the Core Scheduling**

\[ \text{guaranteed bound provided by the arbiter has to be less than this one} \]

**Bring the Deadline and Criticality Down to the Arbitration**

**Execution Time Decomposition**

**Solution**

- To account for shared resources interference in multicore
- Guaranteed bound provided by the arbiter has to be less than this one
1. Combine tasks with same criticality into classes

2. Two-tier Hierarchical arbitration to split inter-class from intra-interference
   - Criticality awareness

3. Harmonic WRR with optimal service assignment
   - Requirement awareness

CArb: Criticality- and Requirement-Aware Arbiter
CArb: Postponing (or Eliminating) Switching

WCET (LO) + WCET (HI) = CP₂

Computation + Memory

C₁, C₂, C₃, C₁, C₂, C₃, C₁

Normal mode + CArb

Normal mode + PCarb

Degraded mode

Timing Interference
CArb: Postponing (or Eliminating) Switching
CArb: Postponing (or Eliminating) Switching Timing Interference

(normal mode + CArb)
(normal mode + PCArb)
(degraded mode)

CArb: Postponing (or Eliminating) Switching
CArb: Postponing (or Eliminating) Switching

- Lower-critical tasks are not suspended
- Higher-critical tasks meet their requirement
- Postponed switching; thus decreasing overheads
- Lower-critical tasks receive no memory guarantees
CArb: Postponing (or Eliminating) Switching Timing Interference

- Lower-critical tasks are not suspended
- Higher-critical tasks meet their requirement
- Postponed switching; thus decreasing overheads
  - Lower-critical tasks receive no memory guarantees

How much increase in computation time?

$\text{WCET (LO)}$ $\text{WCET (HI)}$

20
CArb: Postponing (or Eliminating) Switching Timing Interference

A set of schedules that provide some guarantees to $l$ tasks while mitigate execution-time increase in higher-CL tasks

- Lower-critical tasks are not suspended
- Higher-critical tasks meet their requirement
- Postponed switching; thus decreasing overheads
- Lower-critical tasks receive no memory guarantees
A set of schedules that provide some guarantees to \( l \) tasks while mitigate execution-time increase in higher-CL tasks

- Lower-critical tasks are not suspended
- Higher-critical tasks meet their requirement
- Postponed switching; thus decreasing overheads
- Memory guarantees for lower-critical tasks

C Arb: Postponing (or Eliminating) Switching
C Arb: Postponing (or Eliminating) Switching

A set of schedules that provide some guarantees to $l$ tasks while mitigate execution-time increase in higher-CL tasks

- Lower-critical tasks are not suspended
- Higher-critical tasks meet their requirement
- Postponed switching; thus decreasing overheads
- Memory guarantees for lower-critical tasks
CArb: Postponing (or Eliminating) Switching Timing Interference

Monitor Execution time

- Normal CL-mode + CArb
- Normal CL-mode + CArb sch2
- Normal CL-mode + pCArb

< WCET(\(CL\))

< WCET^{sch2}(\(CL\))

< WCET^{max}(\(CL\))

Switch to (\(CL+1\)) mode

WCET (LO)

WCET (HI)
MPSoC-Based MCS: Four Aspects
Common Approach

Data Sharing

Ignore
- Adopts an independent-task model $\rightarrow$ No communication amongst tasks

Prevent
- Enforcing complete isolation between tasks.
  - At the shared cache: strict cache partitioning and coloring
  - At the DRAM: bank privatization
• May result in a poor memory or cache utilization
  • e.g.: a task has conflict misses, while other partitions may remain underutilized
• Does not scale with increasing number of cores
  • e.g.: number of PEs \( \leq \) number of DRAM banks
• Not viable in emerging systems due to increased functionality and massive data
Solution:
No caching of shared data

[Hardy et al., RTSS’09]
[Lesage et al., RTNS’10]
Another Solution:
Task scheduling on shared data
[Calandrino and Anderson, ECRTS’09]
[Chisholm et al., RTSS’16]
The mainstream solution is to provide shared memory and prevent incoherence through a hardware cache coherence protocol, making caches functionally invisible to software.
Coherence is the Industry’s Choice
today's SoCs include a mix of CPU cores, computing clusters, GPUs and other computing resources and specialized accelerators. Getting heterogeneous processors to communicate efficiently is a daunting design challenge. **A popular approach is to use high-performance and power-efficient shared-memory communication and a sophisticated on-chip cache-coherent interconnect.** This presentation will introduce a new technology that automates the architecture design process, supports CHI and ACE in one design, and uses advanced machine-learning algorithms to create an optimal pre-verified cache-coherent solution.
Coherence is the Industry’s Choice

Autonomous driving requirements are mandating the simultaneous use of multiple types of processing units to efficiently execute sophisticated image processing, sensor fusion, and machine learning/AI algorithms. This presentation introduces a new coherency platform technology that enables the integration of heterogeneous cache coherent hardware accelerators and CPUs, using a mixture of ARM ACE, CHI, and CHI Issue B protocols, into systems that meet both the requirements of high compute performance and ISO 26262-compliant functional safety.
Unpredictability in Sharing Data
Unpredictability in Sharing Data

- Inter-core coherence interference on same cache line
- Inter-core coherence interference on different cache lines
- Inter-core coherence interference due to write hits
- Intra-core coherence interference

Unpredictability in Sharing Data
PMSI: Predictable Cache Coherence
Performance Gains of Coherence

- Uncache all
- Single core
- Uncache shared
- PMSI
- MSI
- MESI

Data Sharing
• **Time-based Cache Coherence**
  • Configurable timers for critical/non-critical cores

• **Fixed Priority Arbitration**
  • If both critical and non-critical requesting same cache line → critical gets it

• **Allows for simultaneous data sharing**
  • Both intra- and inter-criticality

• ** Bounds WCL for critical cores while improving the BW of non-critical cores**
MPSoC-Based MCS: Four Aspects

- Traditional MCS Model
- Timing Interference
- Data Sharing
- Security
Security is a nightmare challenge on its own for all computing systems. It is even more scary for MCS. Three specific challenges for MPSoC-based MCS.
MPSoCs open the door for customized solutions

**MPSoCs Opportunities**

**ARM Cortex-M35P with Physical Security**

- Nested vectored interrupt controller
- Coreset interface
- Memory protection unit
- ITM trace
- Instruction Cache
- Coreset interface
- Wake-up interrupt controller
- DSP
- FPU
- Data watchpoint
- Breakpoint unit
- Serial wire
- MTB trace

**NXP’s QorIQ SoC with Trust Architecture**

- General Purpose Processor
- General Purpose Processor
- DDR Controller
- Security Subsystem
- Cryptography
- Key Management
- Secure Communication
- Tamper Detection
- Power Management
- Internal BootROM
- SPI, UART, CAN
- SD/MMC
- PCIe, USB, SATA
- Coreset/Reset
- OS, Business Integrity Check
- Memory Subsystem
- Bus Subsystem
- Security Subsystem
- DDR Controller
- Insecure Boot
- Real Time Debug
- Watchdog

---

**Opportunities**

- ARM Cortex-M35P
- NXP’s QorIQ SoC
- Physical Security
- Trust Architecture
MPSoCs Challenges

- Cyber-Physical Nature of MCS
- Heterogeneity of MPSoCs
- Shared Components (Again!)
Cyber-physical Nature

- MCS manage sensitive tasks in critical domains: power grids, cars, factories, nuclear plants
- Any security breach could lead to catastrophic consequences
- Hackers gained access to locked cars by only eavesdropping a single signal from the original remote keyless entry unit of the car
Heterogeneity of MPSoCs

- Each PE can be a 3rd-party IP (40% at Intel!)
- PEs share system components and interact with each other → new across-PEs threats
- Stuxnet attack exploited the authentication of the Siemens programmable logic controller to access a Windows machine
Shared hardware components in MPSoCs

- Historically, security was not considered as a concern for MCS because of isolation
- Not the case anymore
- Researchers were able to control sensitive (considered secure) engine control by compromising the (considered insecure) radio unit
  - Reason? Sharing the CAN
Possible Directions

- Identifying new vulnerabilities of MPSoCs, which did not exist in traditional platforms
- Developing cost- and performance-effective methodologies to prevent or mitigate them
- Adopting security as a first-class citizen in designing MPSoCs for MCS (secure-by design concept).
- Scheduling techniques
Our focus so far has been in uniprocessors.

Computing systems → MCS

SoC is the choice for Automotive and IoT

Traditional MCS Model

Data Sharing

Timing Interference

Security