Providing QoS policies for mixed-criticality applications on RISC-V based MPSoCs

Raúl de la Cruz* (raul.delacruz@collins.com), Gonzalo Salinas and Alejandro G. Gener
Connected & Real-time Systems Group, Collins Aerospace Applied Research & Technology, Ireland

RISC-V Summit Europe
5-9 June 2023, Barcelona, Spain

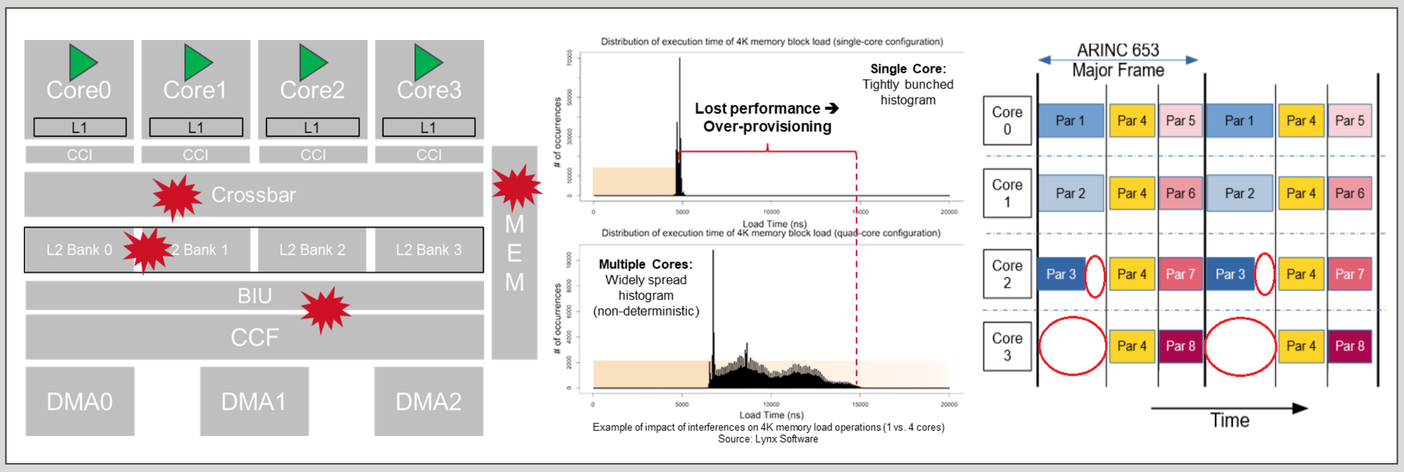
Vision & Value

Business Need:

- MPSoCs are required to adopt autonomous systems but its timing behavior is highly **non-deterministic** and **hinders certification process** [1].
- **Reduce time to market** providing evidence in a faster way to the certification authorities.

Technical Challenge:

- **Avoid deactivation of resources or cores** to avoid contention scenarios and ease certification.
- Escape from **long provisioning** in RTOS/HV and **rigid schedulers** (ARINC-653) to guarantee timeliness.
- Boost **mixed-criticality performance** on MPSoCs.



"For both cases, we collect memory access time and application's execution time **while one core executes the benchmark and others execute a stressing benchmark over the same DRAM controller**." (FAA TC-16/51, p. 48)

Methodology

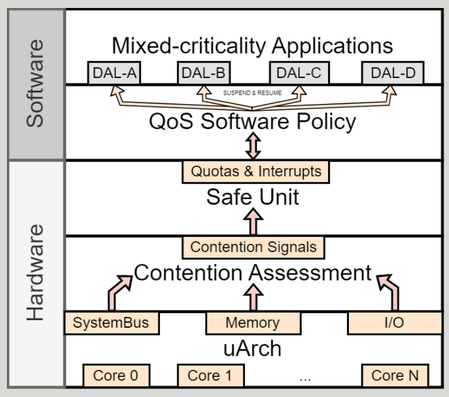
QoS Methodology

QoS on custom Multicore uarch

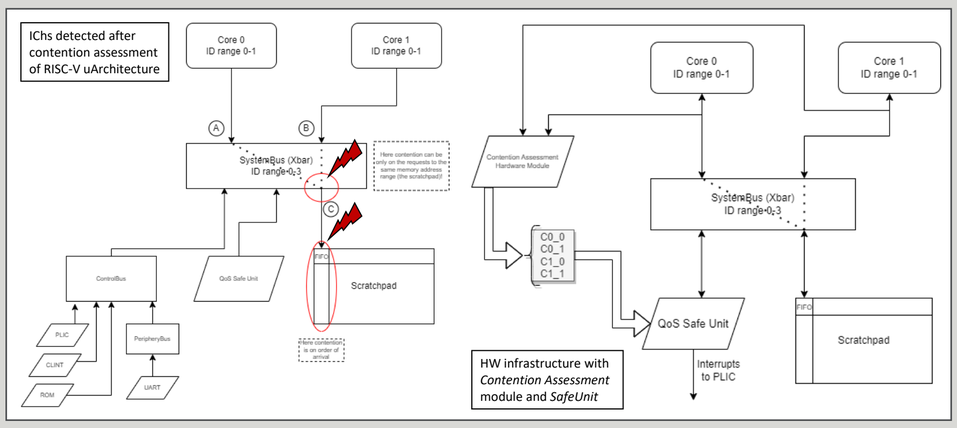
- Specialized stack to assess the contention at runtime and stop contenders when quota exceeded

How is it achieved?

- *Contention Assessment* module to spot master creating contention.
- *SafeUnit*[3,4] to account contention and maximum quota assignment
- *SW policies* to enforce QoS means over a mixed-criticality system.



HW infrastructure



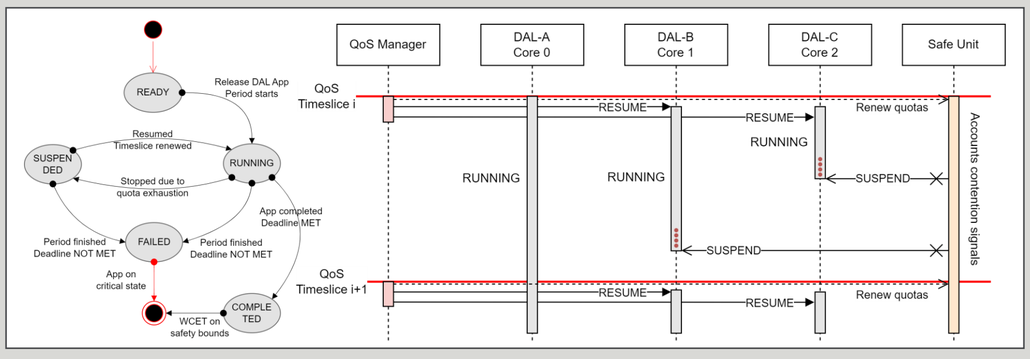
RISC-V testbed architecture

- 4-cores *RocketChip* deployed on Zynq UltraScale+
- I-cache no coherent and main memory (scratchpad) as shared resource
- TL-UL and TL-UH as Xbar protocols (single and burst transactions) [2]

Types of contention identified

- *Serialization* of messages on TileLink crossbar (buffered/enqueued)
- Locking of scratchpad on *read burst transactions* (multi clock cycle)

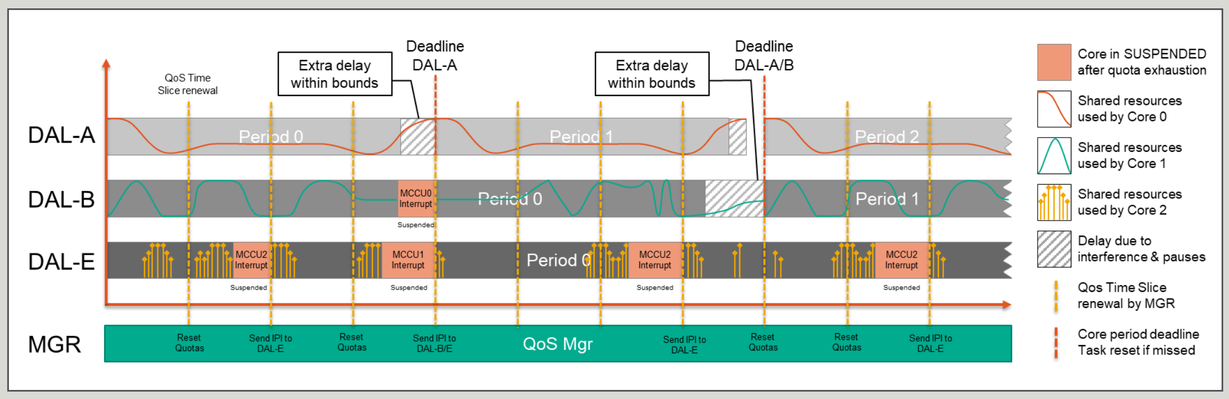
SW layer for Mixed-criticality



SW stack designed to provide high level of contention mitigation

- QoS policy based on *Timeslices* (q) providing reactive and sensitive response
- Cores assigned to *criticality levels* (DALs) $a_i^{dal} \subseteq \{\tau_1, \dots, \tau_n\} \,/\, c_j \mapsto \{a_1^{dal}, \dots, a_m^{dal}\}$
- *Monitoring* resets policies resuming halted cores after quota exhaustion
- *Chained-rule* quota mechanism where rules are triggered based on DALs

Evaluation on Flexible Scheduler



Flexible scheduling - tasks are scheduled at different frequencies

- *Contention Assessment* sniffs *TileLink* traffic and *MCCU* accounts contention cycles raising interrupts
- *Manager* initializes policy quotas and reroutes SUSPEND interrupts to contending cores (RUNNING)
- *QoS TimeSlice* quotas are renewed periodically waking up SUSPENDED cores (CLINT & IPI interrupts)

Usecase objectives

- Implement a *mixed-criticality system* with *tasks released at specific rates* (periods) *with stringent deadlines*. Each DAL core has *different periods and deadlines for their major frames*, breaking the strict *ARINC-653* alignment.
- Demonstrate that the *QoS Time Slice policy* can provide a *deterministic behaviour* for very high assurance level applications (DAL-A/B) that *run periodically* providing *very high performance*.

	Isolation (SOLO)		
Profile	Core 0	Core 1	Core 2
Workload	ub_complex_1	ub_complex_2	ubenchmark2
Parameters	NREPS=8	NREPS=9	NREPS=100
Elapsed time (SOLO)	8.13ms	18.65ms	67.60ms
PROC_CYCLES	254141	582705	2112463
INST_RETIRED	92445	82501	253629
INT_STORE_INST_RET	4072	0	28160
INT_LOAD_INST_RET	3312	7	56327
DATA_CACHE_MISS	7383	2906	84486
INS_CACHE_MISS	3062	29336	112645
Deadline	10.00ms	20.00ms	250.00ms
Grace time	1.87ms	1.35ms	182.40ms
Major Frames	100 periods	50 periods	4 periods

Software architecture

- **Core 0**: DAL-A high priority app with strong deadline of *10ms (100Hz)*.
- **Core 1**: DAL-B app (BCET *~18ms)* with deadline of *20ms (50Hz)*.
- **Core 2**: DAL-E app acting as aggressive contender with BCET *~70ms (4Hz)*.
- **Core 3**: Manager core that manages QoS Timeslice policy.
- **Scheduler** is run for 1 sec (TimeSlice frequency = 1ms – 31,250 cycles)

Multiple chained and hierarchical quotas

- **MCCU0**: Contention over *C0* (*C0_1 + C0_2*) stops *CORE1*
- **MCCU1**: Contention over *C0* (*C0_1 + C0_2*) stops *CORE2*
- **MCCU2**: Contention over *C1* (*C1_2*) stops *CORE2*

~20% of TimeSlice quota (31,250 cycles)

	MCCU QUOTAS & Xbar Configuration					
MCCUs	Event 1	Event 2	Quota	Max delay	Suspends	Exhausted
MCCU 0	C0_1	C0_2	6500 cycles	0.208ms	CORE 1	42 times
MCCU 1	C0_1	C0_2	2300 cycles	0.074ms	CORE 2	946 times
MCCU 2	C1_2	LOW	2800 cycles	0.090ms	CORE 2	38 times
MCCU 3	LOW	LOW	0 cycles	0.000ms	NONE	0 times

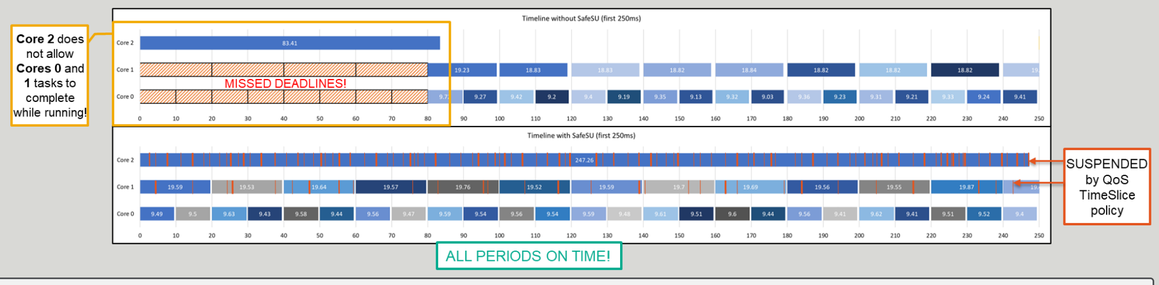
Timing analysis disabling and enabling *QoS TimeSlice* policy

	Isolation	Parallel without SafeIPs					Parallel with SafeIPs				
Timing Analysis	BCET	WCET	Delay	Slowdown	Missed	Deadline met	WCET	Delay	Slowdown	Missed	Deadline met
Core 0 (DAL-A)	8.13ms	>10ms	MISSED	MISSED	37/100	NO	9.63ms	1.50ms	18.45%	0/100	YES
Core 1 (DAL-B)	18.65ms	>20ms	MISSED	MISSED	17/50	NO	19.87ms	1.22ms	6.54%	0/50	YES
Core 2 (DAL-E)	67.60ms	83.42ms	15.82ms	23.40%	0/4	YES	248.13ms	180.53ms	267.06%	0/4	YES

All periods are completed on time!!!

Stopped >900 times by MCCUs during TimeSlices

Trace of the actual scheduling disabling and enabling *QoS TimeSlice* policy



Conclusions

Improvements to foster mitigation response and performance

1. Include *Timeslice* mechanism on HW providing frequency concept
2. Periodic renewal of core quotas after timeslice expiration
3. Interrupts to suspend & resume cores (INT_SUSPEND, INT_RESUME)

Benefits of the QoS Policies proposed

1. Contention cycle-accurate with visibility of SoC behavior
2. Agnostic QoS mechanism for multiple cores and HV/RTOS
3. Advanced QoS policies to guarantee deadline compliance
4. Enhanced flexibility over *ARINC-653* rigid static schedule
5. Flexible scheduler hosting mixed-criticality apps and periods
6. Mitigations for *DoS* attacks that cause contention on ICh

References

[1] Woodrow Bellamy. "Avionics Industry Advances Toward DAL A Multicore Adoption". In: aviationtoday.com (2020).
[2] SiFive. TileLink specification. https://starfivetech.com/uploads/tilelink_spec_1.8.1.pdf.
[3] Guillem Cabo et al. "SafeSU: an Extended Statistics Unit for Multicore Timing Interference". In: 2021 IEEE European Test Symposium (ETS). 2021. DOI: 10.1109/ETS50041.2021.9465444.
[4] Pablo Andreu et al. End-to-End QoS for the Open Source Safety-Relevant RISC-V SELENE Platform. 2022. DOI:10.48550/ARXIV.2210.04683.
[5] H2020 SELENE consortium. SELENE RISC-V open source hardware platform. https://gitlab.com/seleneriscv-platform. 2021.