Tampereen yliopisto
Tampere University

Predictable Computer Architecture

The Foundation of Hard Real-Time Systems

Antti Nurmi
Tampere University
antti.nurmi@tuni.fi

Outline

- Terminology
- Unpredictable Computers
- Our Research

Terminology

What are Real-Time Systems? (1/5)

- Computing systems with **explicit timing requirements**.

What are Real-Time Systems? (2/5)

- Consider:
  - “System X should process as many FPS as possible.”

  vs.
  - “System Y must respond to incoming event Z within 10 µs.”

What are Real-Time Systems? (3/5)

- Two broad categories for seriousness of the system timing requirements [1]:
  - Soft:
    - Missing deadlines degrades performance.
    - Inconvenient rather than fatal.
    - E.g.: Video streaming for **sports events**.

What are Real-Time Systems? (4/5)

- Hard:
  - Missing deadlines is a system failure.
  - Expensive things break, people die.
  - E.g.: Video streaming for **remote brain surgery**.

What are Real-Time Systems? (5/5)

- Related: Mixed-Criticality Systems.
- Consider an airplane as one system:[1]
  - Flight controls
  - In-flight information

[1] Daniel Schwen, CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons

Real-Time Operating Systems (RTOS) (1/2)

- A category of operating system specifically designed for implementing real-time systems.
  - Generally more ligthweight that full general-purpose operating systems (GPOS), closer to bare-metal.
  - Examples: Zephyr, FreeRTOS, many proprietary solutions.
  - **Linux is not an RTOS** [2].

Real-Time Operating Systems (RTOS) (2/2)

- Typical RTOS utilities:
  - Task/thread creation & management
    - (Preemptive) context switching
    - Prioritized scheduling
  - Synchronization & shared resource control
    - Mutexes, locks, semaphores, etc.
  - Timing utilities
    - Timers, SysTicks
  - Inter-process communication (IPC)

Unpredictable Computers

What makes a computer (un)predictable? (1/2)

- Fundamentally, digital computers are perfectly deterministic.
- The most significant advances in computer architecture aim to improve **average-case** performance.
  - Caches, multicores, branch prediction, etc.

What makes a computer (un)predictable? (2/2)

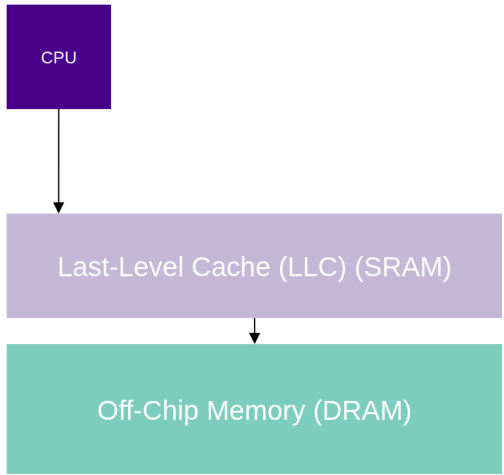
- Real-time systems need to be designed around **worst-case** performance and predictable timing behaviour [3].
- Many average-case optimizations are directly detrimental to worst-case performance.

Example: Memory Hierarchies (1/5)

- **How long does a generic CPU memory access take?**
- Assume an SRAM access latency of 3 cycles.
- Here: 3 cycles, **always**.

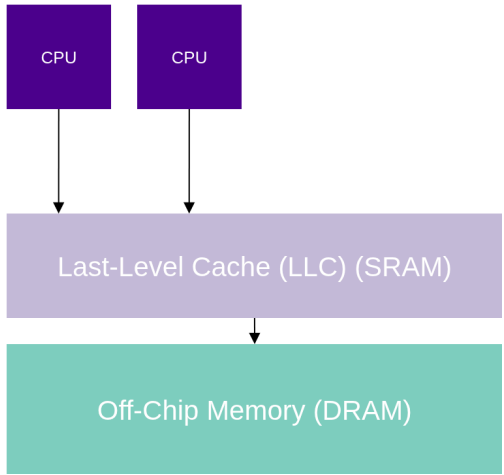
Example: Memory Hierarchies (2/5)

- What if your system needs a lot of memory? (e.g, for running Linux?)
- Assume DRAM accesses (inc. LLC refill) take from 1000 to 5000 cycles.
- LLC hit: 3 cycles
- LLC miss: 1000-5000 cycles



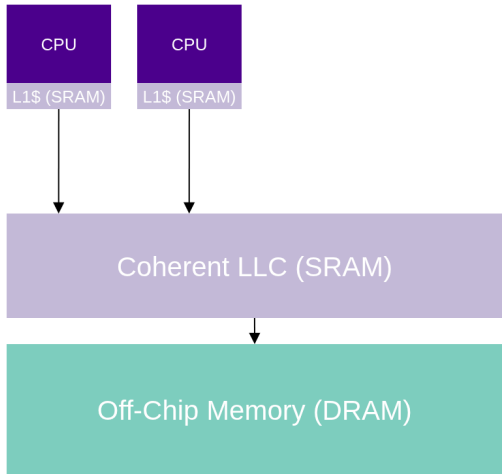
Example: Memory Hierarchies (3/5)

- What about multicores?
- Similar latency, but more cache misses because of conflicting access patterns.
- Hardware resource contention can be catastrophic for time-predictability [4].



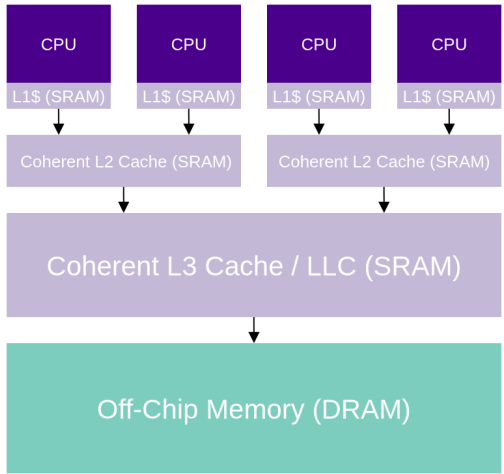
Example: Memory Hierarchies

- Adding L1 caches for better cache hit ratio.
- Coherency required from LLC.
- Access times now:
  - L1 hit: 3 cycles
  - LLC hit, L1 refill: 100 cycles
  - LLC miss, refill: 1000-5000 cycles



Example: Memory Hierarchies (5/5)

- Adding more cores, more cache layers.
- Access times now:
  - L1 hit: 3 cycles
  - L2 hit, L1 refill: 100 cycles
  - LLC hit, L2 refill: 300 cycles
  - LLC miss, refill: 1000-5000 cycles

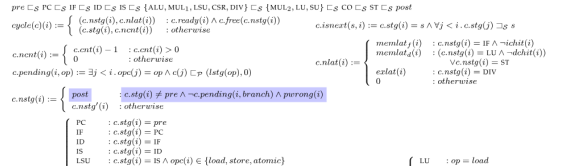


Example: Branch Prediction (1/2)

- Long instruction pipelines depend on **branch prediction** to maintain performance.
- Can be extremely simple or extremely complex.
  - Simple: Always guess yes, ~70 % accuracy.
  - Complex: TAGE [5], neural networks [6], > 95 % accuracy.
- **But**, branch prediction is inherently probabilistic.

Example: Branch Prediction (2/2)

- Proving time-predictability for systems with branch prediction/speculative execution is an active research area [7].



Unpredictable Software (1/2)

- Predictable systems require predictable hardware **and** predictable software.
- **Threads** are the dominant model for concurrent programming.
  - Threads are also notoriously nondeterministic [8].
  - Nicer models are available.

Unpredictable Software (2/2)

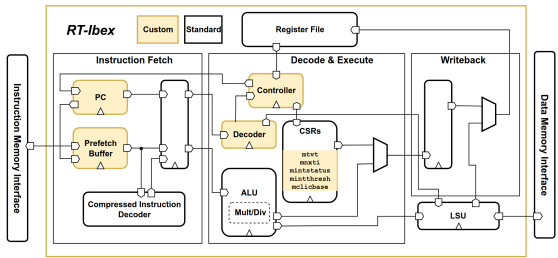
- We have focused on RTIC [9], [10].
  - Lightweight, minimal runtime overhead.
  - Static guarantees provided by Rust, RTIC itself.
  - Solid formal foundation based on the stack resource policy (SRP) [11].

Our Research

(So Far)

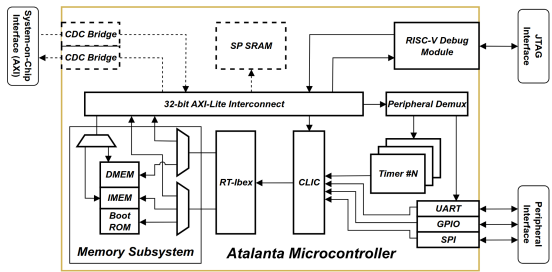
Predictable System Design (1/2)

- Atalanta: our initial work to build a predictable RISC-V microcontroller system.
- Custom Ibex fork for advanced interrupt controller support, Arm-style ISR fetch.
- Integrated to (then) brand-new CLIC [12], other open components.



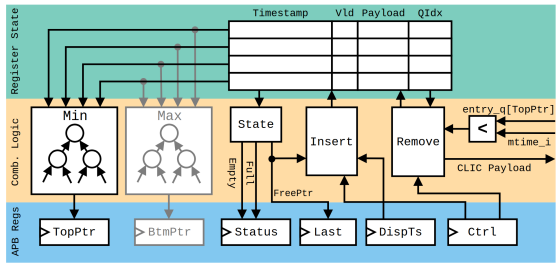
Predictable System Design (2/2)

- Results:
  - Competitive (~21) cycle interrupt latency.
  - Solid platform for subsequent research.
- Published in [13], poster presented at 2024 European RISC-V Summit in Munich



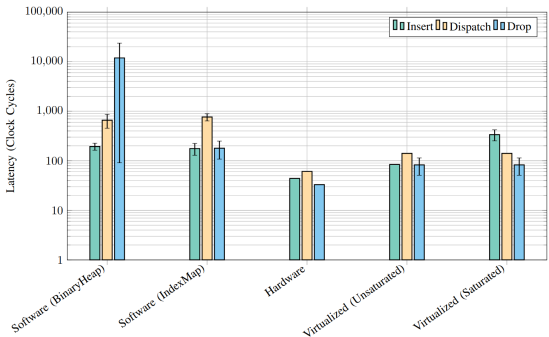
Hardware Priority Queues (1/2)

- Priority queues (PQ) are fundamental to implementing task schedulers.
- Software-based PQs have a logarithmic time-complexity for at least one of its access operations.
  - A hardware PQ can provide constant access times.



Hardware Priority Queues (2/2)

- Initial implementation in [14], improved follow-up in [15].
- Results:
  - Proof of HW feasibility.
  - Latency & jitter comparison against SW data structures.



Fast & Heterogeneous Interrupts (1/3)

- Reactive real-time systems need to do a lot of context switching (CS).
  - Highly repetitive low-level routine.
  - Hardware acceleration has the potential to save a lot of instructions.

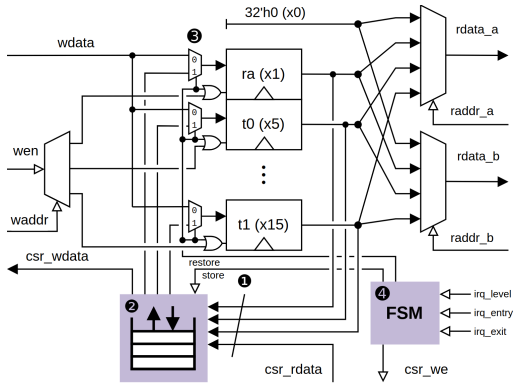
```
trampoline:
  addi  sp,     sp,  -(4 * 9)
  sw    ra,     0(sp)
  sw    t0,     4(sp)
  sw    a0,     8(sp)
  sw    a1,     12(sp)
  sw    a2,     16(sp)
  sw    a3,     20(sp)
  sw    t1,     24(sp)
  csrr  t0,     mepc
  csrr  t1,     mcause
  sw    t0,     28(sp)
  sw    t1,     32(sp)
  csrsi mstatus, 8    // enable global irqs
  jal   functional_isr
  csrci mstatus, 8    // disable global irqs
  lw    t0,     28(sp)
  lw    t1,     32(sp)
  csrw  mcause, t1
  csrw  mepc,   t0
  lw    ra,     0(sp)
  lw    t0,     4(sp)
  lw    a0,     8(sp)
  lw    a1,     12(sp)
  lw    a2,     16(sp)
  lw    a3,     20(sp)
  lw    t1,     24(sp)
  addi  sp,     sp,  (4 * 9)
  mret
```

Entry, 12 Instructions

Exit, 13 Instructions

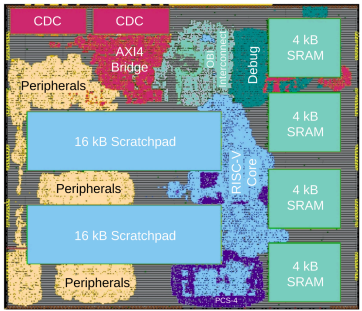
Fast & Heterogeneous Interrupts (2/3)

- Our initial work presented a stacked register file to eliminate the need for CS [16].
- Extension in [17] presented the heterogeneous interrupt (HETI) scheme.
  - Combines accelerated and conventional interrupts.



Fast & Heterogeneous Interrupts (3/3)

- Results:
  - Representative ASIC implementation on TSMC 22-nm.
  - ~25 % instruction & clock cycle reduction in CS-heavy benchmark.



Future Work (1/2)

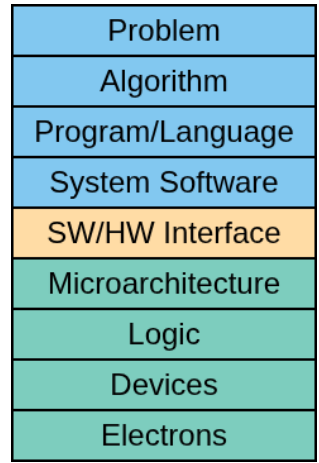
- Potentially a new RISC-V CPU implementation for better intergrating our custom features.
- More work on hardening and evaluating the presented solutions in ASIC physical design.
  - Hopefully with tapeouts & demonstrators in future projects.

Future Work (2/2)

- New publication on hardware-based earliest-deadline first (EDF) scheduling accepted for publication in 2026 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS).
  - Privileged information for now :)

Interested in Contributing?

- Broad research field that relates to most layers of the transformation hierarchy.
- Got inspiration for a related MSc. thesis?
  - Let's have chat!
- Interested in paid work @TAU?
  - No RA openings **right now**, but hopefully we can hire again very soon.



References

[1] G. C. Buttazzo, **Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications**, 3rd ed., vol. 24. in Real-Time Systems Series, vol. 24. New York, NY: Springer Nature, 2011.

[2] F. Reghenzani, G. Massari, and W. Fornaciari, “The Real-Time Linux Kernel: A Survey on PREEMPT_RT,” **ACM computing surveys**, vol. 52, no. 1, pp. 1–36, 2019.

[3] S. A. Edwards and E. A. Lee, “The Case for the Precision Timed (PRET) Machine,” in **Proceedings of the 44th Annual Design Automation Conference**, in DAC '07. Asso-

ciation for Computing Machinery, 2007, pp. 264–265. doi: 10.1145/1278480.1278545.

[4] D. Oliveira, W. Chen, S. Pinto, and R. Mancuso, “Shared Resource Contention in MCUs: A Reality Check and the Quest for Timeliness,” in **36th Euromicro Conference on Real-Time Systems (ECRTS 2024)**, R. Pellizzoni, Ed., in Leibniz International Proceedings in Informatics (LIPIcs), vol. 298. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, pp. 5:1–5:25. doi: 10.4230/LIPIcs.ECRTS.2024.5.

References

[5] A. Seznec and P. Michaud, “A Case for (Partially) Tagged Geometric History Length Branch Prediction,” **Journal of Instruction-Level Parallelism**, vol. 8, p. 23, 2006.

[6] D. A. Jiménez and C. Lin, “Neural methods for dynamic branch prediction,” **ACM Trans. Comput. Syst.**, vol. 20, no. 4, pp. 369–397, Nov. 2002, doi: 10.1145/571637.571639.

[7] A. Gruin, T. Carle, C. Rochange, H. Cassé, and P. Sainrat, “MINOTAuR: A Timing Predictable RISC-V Core Featuring Speculative Execution,” **IEEE Transactions on Comput-**

References

ers, vol. 72, no. 1, pp. 183–195, 2023, doi: 10.1109/TC.2022.3200000.

[8] E. Lee, “The Problem with Threads,” **Computer**, vol. 39, no. 5, pp. 33–42, 2006.

[9] P. Lindgren, P. Dzialo, and H. Lunnikivi, “Hardware support for Static-Priority Stack Resource Policy based scheduling,” in **2023 IEEE 32nd International Symposium on Industrial Electronics (ISIE)**, 2023, pp. 1–5. doi: 10.1109/ISIE51358.2023.10228088.

References

[10] H. Lunnikivi, Z. Madaoui, P. Dzialo, and P. Lindgren, “Modular RTIC: Lightweight Real Time for Customized Architectures,” **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, vol. 33, no. 11, pp. 2952–2960, 2025, doi: 10.1109/TVLSI.2025.3595712.

[11] T. Baker, “A stack-based resource allocation policy for real-time processes,” in **[1990] Proceedings 11th Real-Time Systems Symposium**, 1990, pp. 191–200. doi: 10.1109/REAL.1990.128747.

References

[12] R. Balas, A. Ottaviano, and L. Benini, “CV32RT: Enabling Fast Interrupt and Context Switching for RISC-V Microcontrollers,” **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, vol. 32, no. 6, pp. 1032–1044, 2024, doi: 10.1109/TVLSI.2024.3377130.

[13] A. Nurmi, P. Lindgren, A. Kalache, H. Lunnikivi, and T. D. Hämäläinen, “Atalanta: Open-Source RISC-V Microcontroller for Rust-Based Hard Real-Time Systems,” in **Architecture of Computing Systems**, D. Fey, B. Stabernack, S. Lankes, M.

References

Pacher, and T. Pionteck, Eds., Cham: Springer Nature Switzerland, 2024, pp. 316–330.

[14] A. Nurmi, P. Lindgren, T. Szymkowiak, and T. D. Hämäläinen, "AnTiQ: A Hardware-Accelerated Priority Queue Design with Constant Time Arbitrary Element Removal," in **2023 26th Euromicro Conference on Digital System Design (DSD)**, 2023, pp. 462–469. doi: 10.1109/DSD60849.2023.00070.

[15] A. Nurmi, H. Lunnikivi, P. Lindgren, and T. D. Hämäläinen, "Towards Predictable Ultra-Low Latency End-Nodes with Hardware-Accelerated Abstract Timers," in **2025 IEEE Nordic**

References

Circuits and Systems Conference (NorCAS), 2025, pp. 1–6. doi: 10.1109/NorCAS66540.2025.11231291.

[16] A. Nurmi, A. Kalache, and T. D. Hämäläinen, "Hardware Solutions for Eliminating Context Switching Latency in Processor-Based Hard Real-Time Systems," in **2024 IEEE Nordic Circuits and Systems Conference (NorCAS)**, 2024, pp. 1–6. doi: 10.1109/NorCAS64408.2024.10752471.

[17] A. Nurmi, A. Kalache, H. Lunnikivi, P. Lindgren, and T. D. Hämäläinen, "Efficient and Predictable Context Switching for Mixed-Criticality and Real-Time Systems," **IEEE Transactions**

References

on Very Large Scale Integration (VLSI) Systems, vol. 33, no. 11, pp. 2907–2915, 2025, doi: 10.1109/TVLSI.2025.3612433.

Thank you!

Questions?