Tampereen yliopisto
Tampere University

Computer Architecture for System-on-Chip (SoC) Design

Antti Nurmi
Tampere University
antti.nurmi@tuni.fi

whoami

- Doctoral Researcher @SoC Hub Research Centre.
- Worked professionally with RISC-V system design since 2020.
- Research focus on hardware architectures for embedded real-time systems, specifically:
  - Predictable computer architecture,
  - Optimization of interrupt-driven systems,
  - Hardware support of operating system functions.
- Publications: https://orcid.org/0000-0003-3533-9832

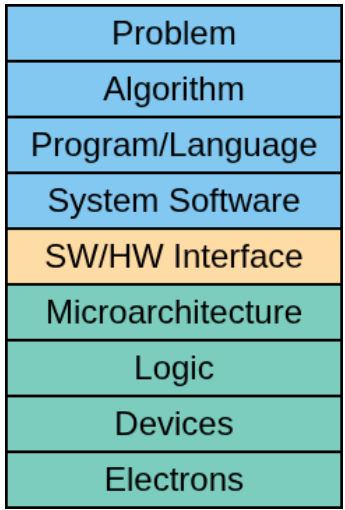
Outline

- Computing Fundamentals
- Real Processors
- Introduction to System-on-Chips (SoC)

Computing Fundamentals

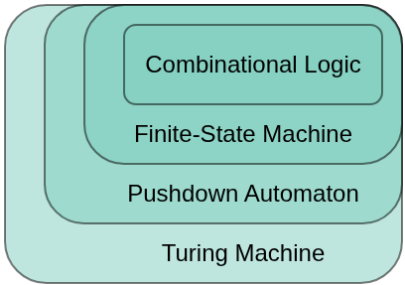
Motivation

- Why do we want to compute things?
- Many levels to consider; optimal solutions require awareness of all.
- Course goal: To understand how computing *systems* are built across multiple abstraction levels.



What is a “Complete” Computer? (1/2)

- Fundamentally, *any* computable algorithm can be computed on a Turing Machine.
- … but only given *infinite* **memory, time and energy.**



What is a “Complete” Computer? (2/2)

- Many things are Turing-complete even unintentionally[1], including:
  - Microsoft Excel,
  - Habbo Hotel,
  - The `printf` format string,
  - x86's MOV instruction.

[1] https://en.wikipedia.org/wiki/Turing_completeness#Unintentional_Turing_completeness

Constraints (1/2)

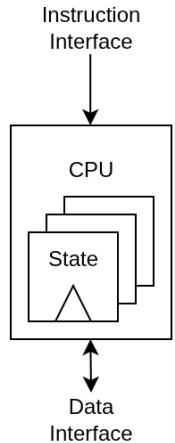
- Real systems do not have infinite memory, time, or energy.
- The **constraints** of a given system should define how it is designed and specialized.

Constraints (2/2)

- For example,
  - Battery-powered systems are inherently *energy-constrained*.
  - Real-time systems are inherently *time-constrained*.
    - E.g., video streaming for 4K @ 30 FPS implies an **explicit** time-constraint for processing.
  - Embedded systems are often *memory-constrained* in addition to being energy-constrained.

The Instruction-Set Processor (1/2)

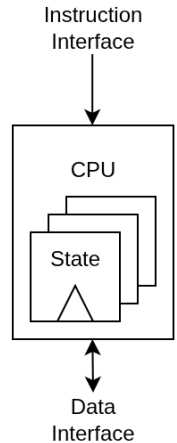
- A.K.A. the Central Processing Unit (CPU).
- The simplest and most ubiqutous implementation of a “full computer”.
- Fundamantally a Finite-State Machine (FSM).



The Instruction-Set Processor (2/2)

An abstract processor consists of:

- An **architectural** state.
  - stored in **registers**
  - programmer-visible
- An interface to read instructions from from **memory** in a given address.
- An interface to access data in **memory** at a given address.



Types of Registers (1/3)

General-purpose registers (GPRs).

- CPU-internal, holds operands of CPU instructions.
- Hardware is general-purpose, but programming conventions dedicate certain GPRs to special functions.
  - RISC-V Example: GPR `x2` holds the stack pointer (`sp`)
- Can be used directly by programmer/compiler.
  - Example:

```
1 add x3, x1, x2
2 bne x6, x7, 0x1000
```

Types of Registers (2/3)

Special-purpose registers

- CPU-internal, holds machine state information, among others.
  - RISC-V: Control and status registers (CSRs)
  - Accessible through dedicated **csr** instructions.
    - Example:

```
1 csrw mtvec, x5
2 csrci mstatus, 8
```

Types of Registers (3/3)

Memory-mapped registers

- External to CPU, common interface to peripherals, accelerators.
- Accessed through memory operations.
  - Example:

```
1  li t0, 0x0380
2  li t1, 0x80000000
3  sw t1, 0(t0)
```

Instruction-Set Architecture (ISA) (1/2)

- The specification of operations that a processor architecture supports.
- Reduced or Complex (RISC or CISC)
  - RISC trade-off: program complexity
  - CISC trade-off: hardware complexity

Instruction-Set Architecture (ISA) (2/2)

Instructions may:

- Modify the architectural state.
  - E.g., multiply two internal operands
- Access memory.
  - **Load** or **Store**
- Alter program flow.
  - Conditionally: **Branch**
  - Unconditionally: **Jump**

Architecture vs. Microarchitecture (1/3)

- Assume the execution of an instruction I transforms the architectural processor state from $AS \rightarrow AS'$.

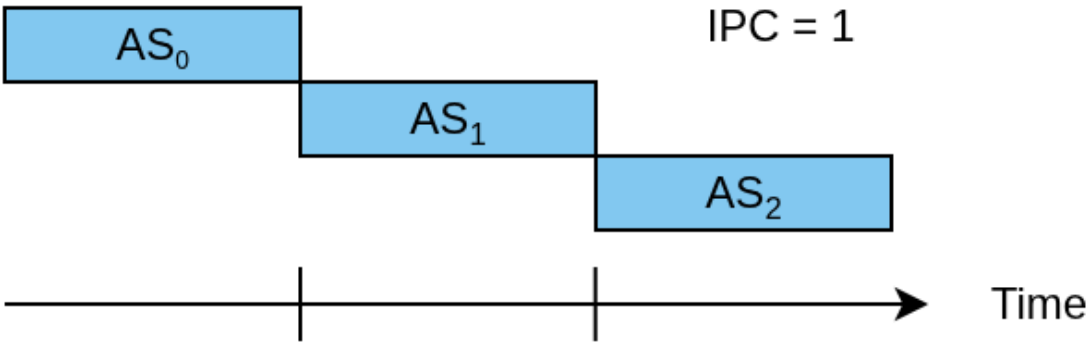
Architecture vs. Microarchitecture (2/3)

- In **single-cycle** microarchitectures, this transition always completes within one clock cycle with no intermediate states.
  - All instructions have identical execution time.
  - Longest **instruction** defines max. clock frequency.
  - Cycles-per-instruction (CPI) = 1, but clock cycle is very long.

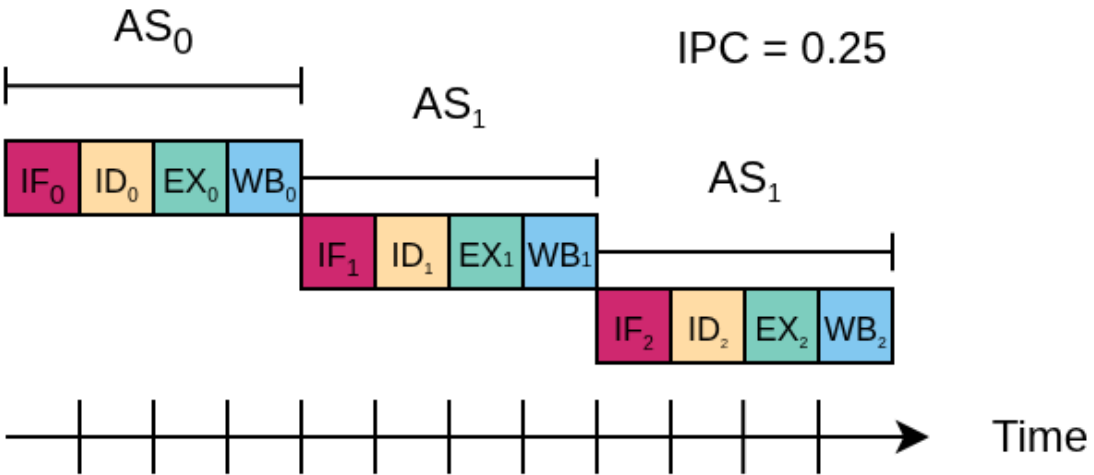
Architecture vs. Microarchitecture (3/3)

- **Multi-cycle** architectures allow for **microarchitectural** intermediate states.
  - I.e., $AS \rightarrow AS + MS_1 \rightarrow AS + MS_2 \rightarrow AS + MS_n \rightarrow AS'$
  - Microarchitectural states are not programmer-visible.
  - Instructions can have a varied execution times.
  - Longest **combinational path** defines max. clock frequency.
  - CPI > 1, but clock cycle is short.

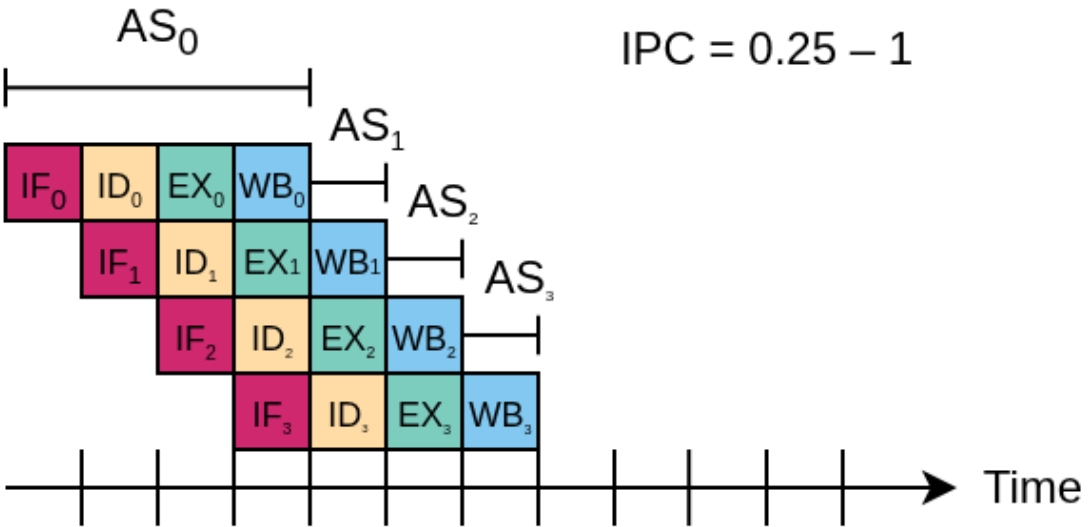
Single-Cycle Microarchitecture



Multi-Cycle Microarchitecture



Pipelined Microarchitecture



Textbook RISC Microarchitecture [1]

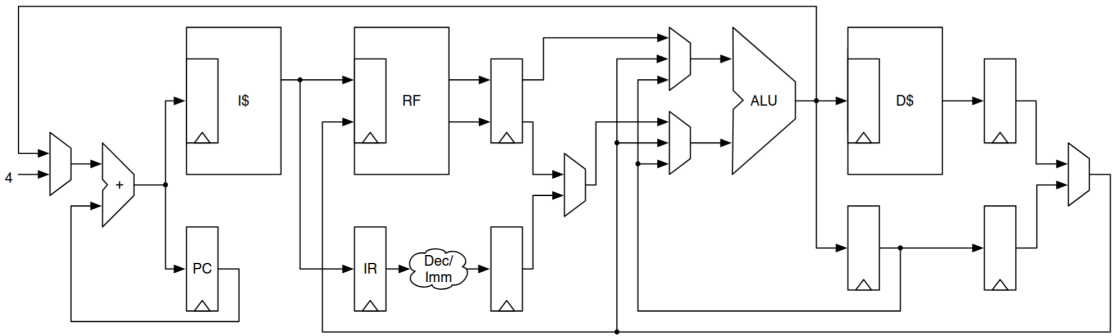


Fig. 1. A textbook style 5-stages RISC-V processor pipeline.

A Better* RISC Microarchitecture [1]

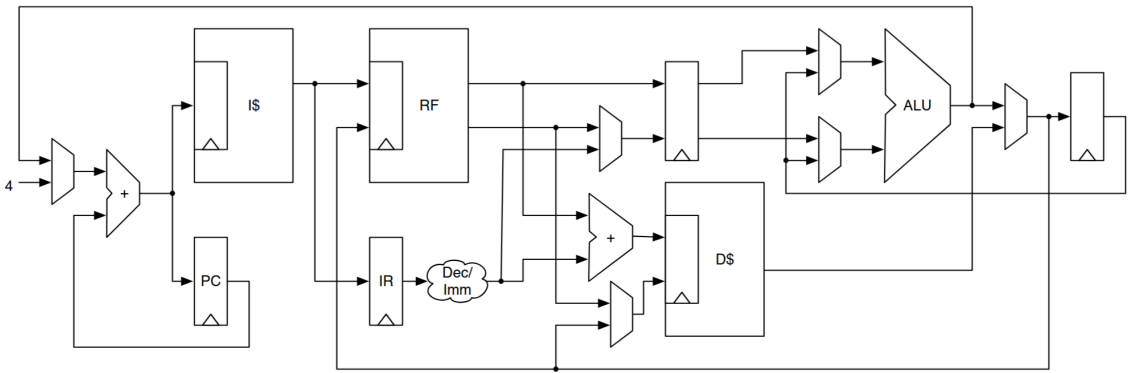


Fig. 2. A 3-stage RISC-V processor pipeline.

Advanced Microarchitectures (1/2)

- Deeper pipelines
  - 20+ stages feasible in big CPUs.
  - Usability depends on branch prediction.
- Multi-issue execution
  - Issue & execute multiple instructions from single stream.
  - More data depencences, needs more involved scheduling.
    - Statically by compiler: VLIW
    - Dynamically by hardware

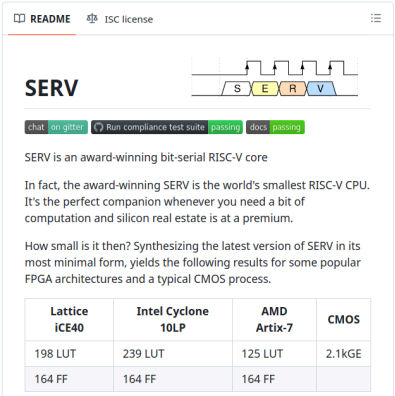
Advanced Microarchitectures (2/2)

- Out-of-Order (OoO) execution
  - Instructions can be executed OoO, retired in-order to preserve sequential program semantics.
  - Internally implemented as dataflow processors.
- Out of scope here, but an excellent lecture series by Prof. Onur Mutlu from ETH Zürich covering these and much more is available here: https://www.youtube.com/watch?v=ubhxKNlOlRg&list=PL5Q2soXY2Zi9Eo29LMgKVcaydS7V1zZW3

Real Processors

SERV — The SErial Risc-V[2]

- Smallest RISC-V in the world.
- Bit-serial for extreme area saving.
- May look like a toy, but there is genuine demand for tiny CPUs.
  - Recall: Slide 8 – Constraints
- Featured in Springer Nature [2] as the first full RISC-V implemented with flexible electronics.


README ISC license

SERV

S E R V

chat on gitter Run compliance test suite passing docs passing

SERV is an award-winning bit-serial RISC-V core

In fact, the award-winning SERV is the world's smallest RISC-V CPU. It's the perfect companion whenever you need a bit of computation and silicon real estate is at a premium.

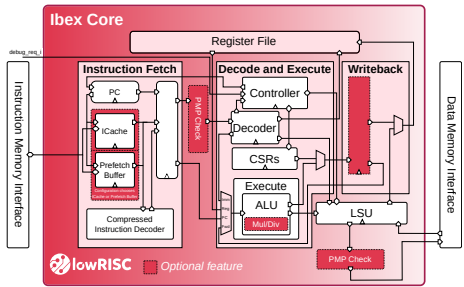
How small is it then? Synthesizing the latest version of SERV in its most minimal form, yields the following results for some popular FPGA architectures and a typical CMOS process.

Lattice iCE40	Intel Cyclone 10LP	AMD Artix-7	CMOS
198 LUT	239 LUT	125 LUT	2.1kGE
164 FF	164 FF	164 FF	

[2] https://github.com/olofk/serv

Ibex

- Very popular open-source[3] project, used on this course.
- Lot of microarchitecture parameters supported.
- Performance comparable to low-end Arm CPUs.
- Area: ~15–60 kGE



[3] https://github.com/lowRISC/ibex

CVA6 and Xuantie C910 [3]

- Application-class CPUs
- CVA6: 6 stages
  - Area: 2282 kGE
  - CM/MHz: 2.19
  - $F_{\max}$: 1.3 GHz
- C910: 12 stages
  - Area: 3992 kGE
  - CM/MHz: 4.86
  - $F_{\max}$: 1.7 GHz

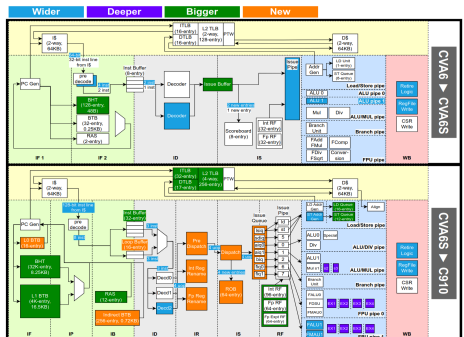
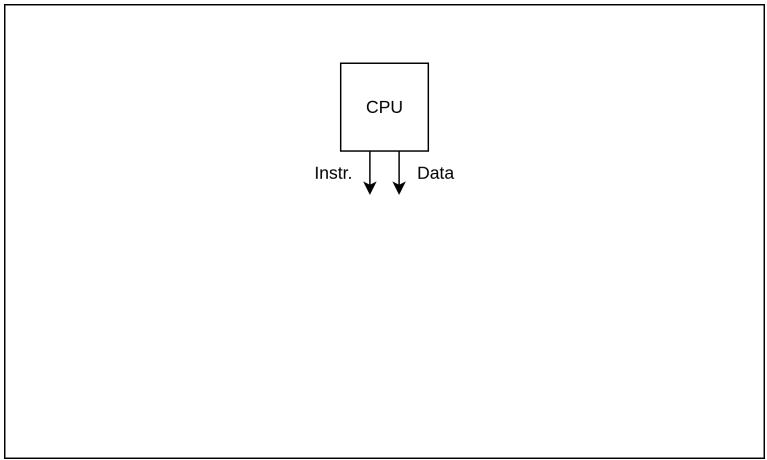


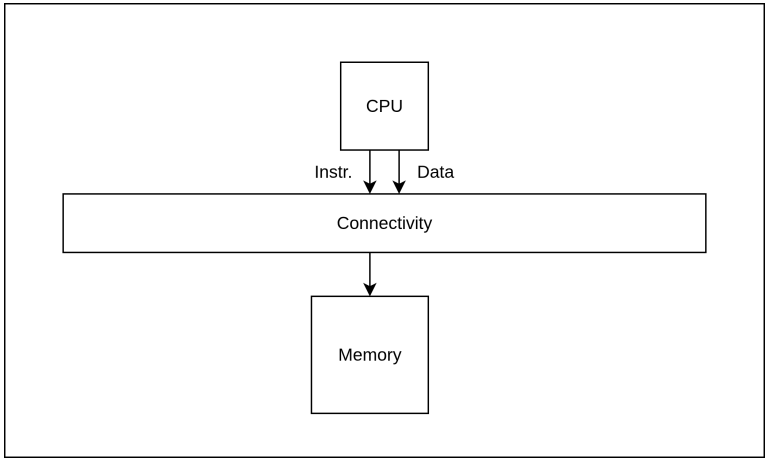
Figure 1: Architecture comparison of CVA6, CVA6S, and C910 cores

System-on-Chips (SoCs)

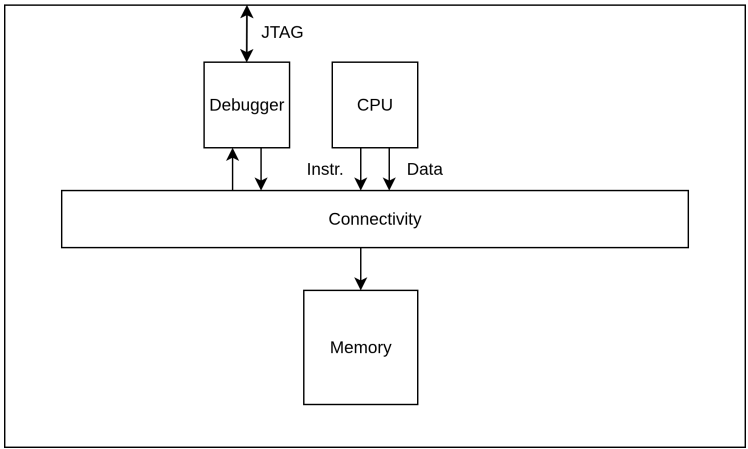
Building a Usable Computing System (1/9)



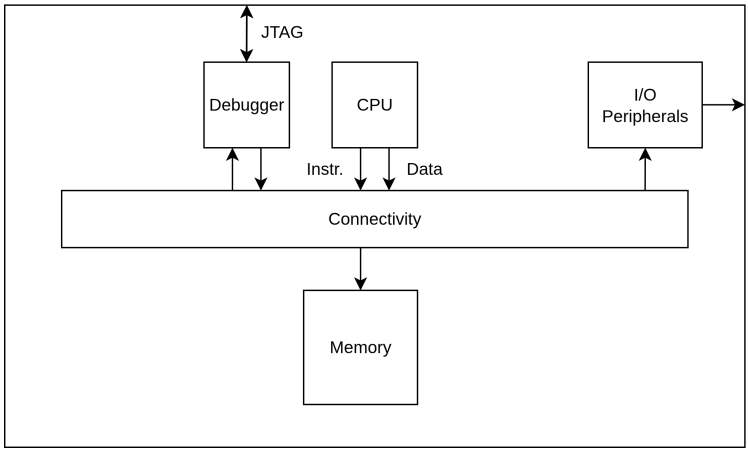
Building a Usable Computing System (2/9)



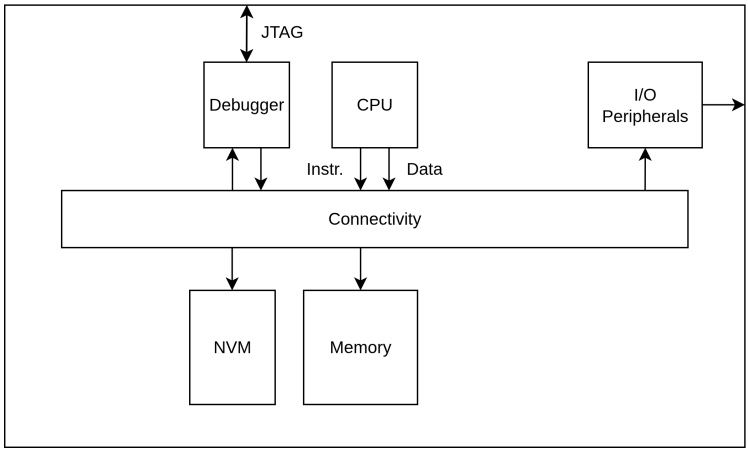
Building a Usable Computing System (3/9)



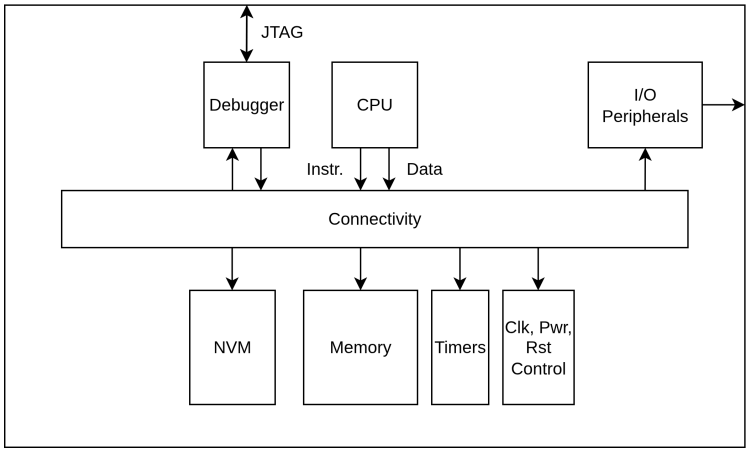
Building a Usable Computing System (4/9)



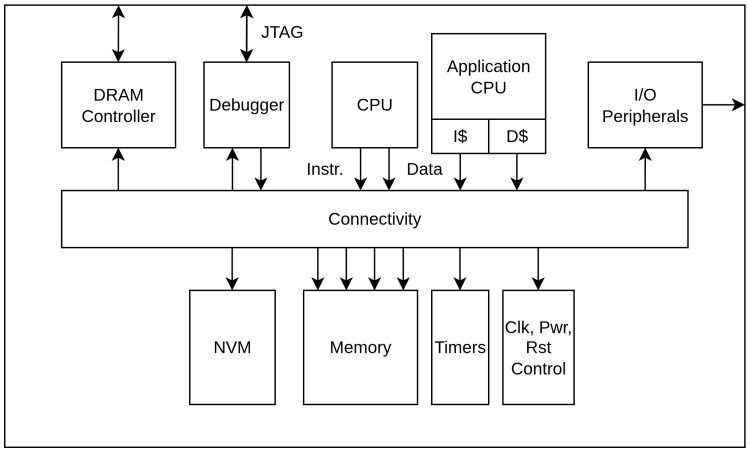
Building a Usable Computing System (5/9)



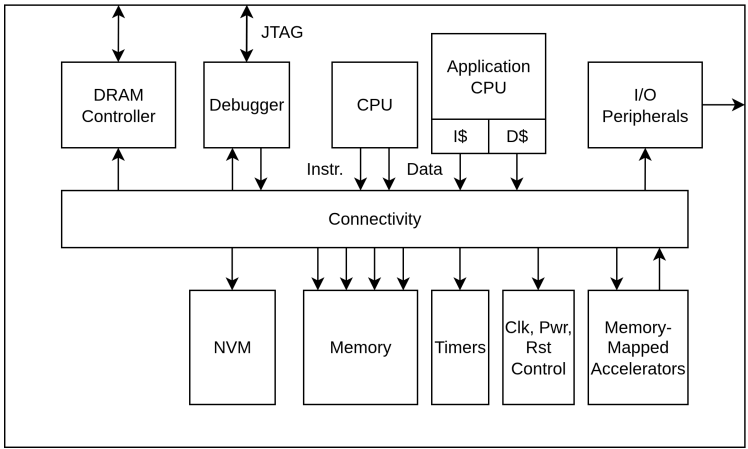
Building a Usable Computing System (6/9)



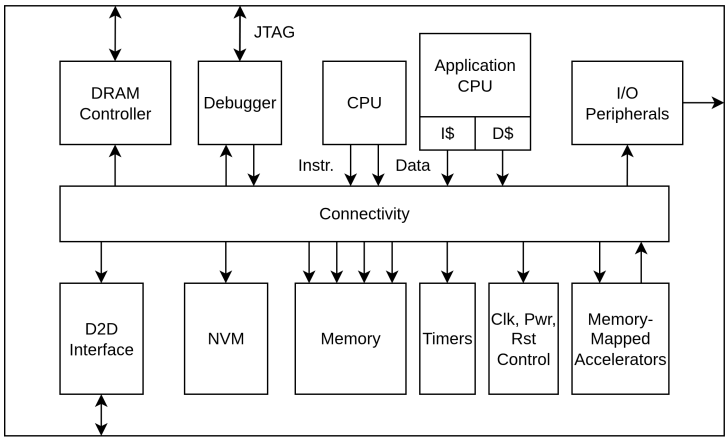
Building a Usable Computing System (7/9)



Building a Usable Computing System (8/9)



Building a Usable Computing System (9/9)



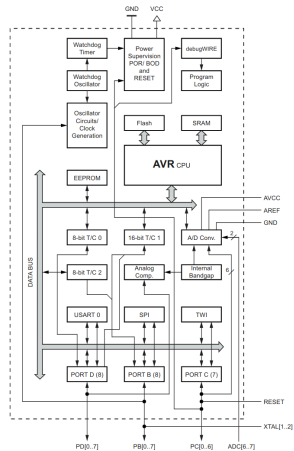
Definition – System-on-Chip (SoC)

- “An integrated circuit that combines most or all key components of a computer or electronic system onto a single microchip.”[4]

[4] https://en.wikipedia.org/wiki/System_on_a_chip

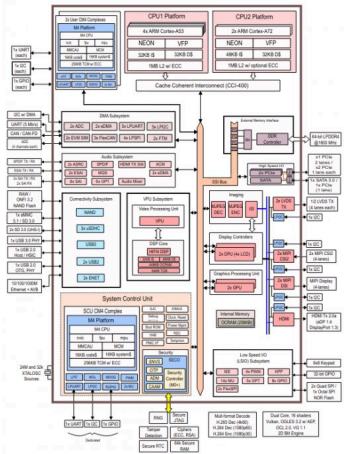
From MCU to SoC

- Technically, microcontrollers (MCUs) are very simple “SoCs”.
- Example: Atmega328p [4]
  - One 8-bit CPU.
  - Some on-chip memory (inc. NVM).
  - Some I/O and timers.
- SW complexity: reasonable.



From MCU to SoC

- However, “SoC” more commonly means something like this:
- Example: NXP iMX.8
  - 2 × Appliaction CPU **clusters**.
  - Dedicated subsystems for video, audio, graphics, connectivity…
  - More small CPUs.
  - A **lot** of I/O.
- SW complexity: dumpster fire [5].



Procuring Intellectual Property (IP) (1/2)

- Reinventing the wheel is not good business.

Procuring Intellectual Property (IP) (2/2)

- This is why companies source IP from 3rd parties.
  - Traditionally: commercial vendors like Arm, Arteris, etc.
  - New trend: open-source IP
    - Common Modules
    - CPU Cores
    - AXI Interconnects
    - and many more!

Summary

- CPUs are the fundamental building blocks of computer systems.
  - Available in many shapes and sizes.
- The performance of SoCs comes from creating a **heterogeneous, application-specific** set of computation capabilities.
- Don’t reinvent the wheel. Source non-critical IP from somewhere else, focus on developing the differentiating factors in your system.

References

[1] M. Schoeberl, “Wildcat: Educational RISC-V Microprocessors,” in **Architecture of Computing Systems: 38th International Conference, ARCS 2025, Kiel, Germany, April 22–24, 2025, Proceedings**, Kiel, Germany: Springer-Verlag, 2025, pp. 189–202. doi: 10.1007/978-3-032-03281-2_13.

[2] E. Ozer **et al.**, “Bendable non-silicon RISC-V microprocessor,” **Nature**, vol. 634, no. 8033, pp. 341–346, Oct. 2024, doi: 10.1038/s41586-024-07976-y.

[3] Z. Fu **et al.**, “Ramping Up Open-Source RISC-V Cores: Assessing the Energy Efficiency of Superscalar, Out-of-Order

References

Execution,” in **Proceedings of the 22nd ACM International Conference on Computing Frontiers**, in CF '25.: Association for Computing Machinery, 2025, pp. 12–20. doi: 10.1145/3719276.3725186.

[4] Microchip Technology Inc., “ATmega328P Automotive Microcontrollers ATmel-7810 Datasheet.” Chandler, AZ, USA, 2015. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

References

[5] A. Baumann **et al.**, “The multikernel: a new OS architecture for scalable multicore systems,” in **Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles**, in SOSP '09. Big Sky, Montana, USA: Association for Computing Machinery, 2009, pp. 29–44. doi: 10.1145/1629575.1629579.

Thank you!

Questions?