FPGAs as Tools and Architectures at ETH Systems
FPGAs as Tools and Architectures at ETH Systems

Real-Time Tracing and Verification
- The FPGA as a tool.
- Analysing a multi-Gb trace stream in real time.

BRISC – Research Architecture for Large Systems
- The FPGA as an architecture.
- A platform for hardware and software research.
- Expose the coherent interface to an FPGA, with lots and lots of fast IO links.
Real-Time Tracing and Verification
We're Going to Build a *Large Program Collider*

Collide *instructions* at 0.99c, and observe the decay products.
Programmers Once (Thought They) Understood Computer Architecture

Figure 1.4
Hardware organization of a typical system. CPU: Central Processing Unit, ALU: Arithmetic/Logic Unit, PC: Program counter, USB: Universal Serial Bus.

systems, but all systems have a similar look and feel. Don’t worry about the complexity of this figure just now. We will get to its various details in stages throughout the course of the book.
Symmetric Multiprocessors Were *Fairly* Simple
Concurrent Code Makes Architecture Visible

- Consider message passing.
  - Pretty much the simplest thing you can do with shared memory.
  - Systems like Barrelfish rely on it.
- When are barriers required?
- You can't write good code, without sufficiently understanding the hardware.
- We're combining components in new ways.
Message Passing with Shared Memory

Write: \*x = 42
Write: \*y = 1

Read: \*y = 1
Read: \*x = 42

\*x = 02
\*y = 0
Message Passing with a Write Buffer

Write: \( *x = 42 \)
Write: \( *y = 1 \)

\( x = 42 \)
\( y = 1 \)

Read: \( y = 1 \)
Read: \( x = 0 \)
Message Passing with a Barrier

CPU

Write: *x = 42
Write: *y = 1

WB

*x = 42
*y = 1

RAM

*x = 02
*y = 0

CPU

Read: *y = 1
Read: *x = 42
Of Course, CPUs Aren't *That* Simple

9 hops

CPU WB L1
L2
PCI
RAM
Coherent Interconnect

CPU WB L1
L2
L3
You Can't Trust the Hardware

- seL4 was verified *modulo* a hardware model.
- The Cortex A8 has bugs:
  - Cache flushes don't work.
  - As of today, these “errata” are **still** not public.
  - We rediscovered these by accident.
- Non-coherent memory is coming.

Source: Chip Errata for the i.MX51, Freescale Semiconductor

| ENGcm09630 | ARM: Load and Store operations on the shared device memory regions may not complete in program order | No fix scheduled | 12 |
| ENGcm07788 | ARM: A RAW hazard on certain CP15 registers can result in a stale register read | No fix scheduled | 14 |
| ENGcm04786 | ARM: ARPROT[0] is incorrectly set to indicate a USER transaction for memory accesses generated from user tablewalks | No fix scheduled | 16 |
| ENGcm04785 | ARM: C15 Cache Selection Register (CSSELR) is not banked | No fix scheduled | 18 |
| ENGcm07784 | ARM: Cache clean memory ops generated by the Preload Engine or Clean by MVA to PoC instructions may corrupt the memory | No fix scheduled | 19 |
| ENGcm07786 | ARM: Under a specific set of conditions, a cache maintenance operation performed by MVA can result in memory corruption | No fix scheduled | 21 |
| ENGcm07782 | ARM: Clean and Clean/Invalidate maintenance ops by MVA to PoC may not push data to external memory | No fix scheduled | 23 |
| ENGcm04765 | ARM: Incorrect L2 cache eviction can occur when L2 is configured as an inner cache | No fix scheduled | 25 |
| ENGcm04761 | ARM: Swap instruction, preload instruction, and instruction fetch request can interact and cause deadlock | No fix scheduled | 26 |
| ENGcm04759 | ARM: NEON load data can be incorrectly forwarded to a subsequent request | No fix scheduled | 28 |
| ENGcm04760 | ARM: Under a specific set of conditions, processor deadlock can occur when L2 cache is servicing writes allocate memory | No fix scheduled | 30 |
| ENGcm10230 | ARM: Clarification regarding the ALP bits in AMC register | No fix scheduled -Clarified in RM | 32 |
| ENGcm10700 | ARM: If a Perf Counter OVFL occurs simultaneously with an update to a CP14 or CP15 register, the OVFL status can be lost | No fix scheduled | 33 |
| ENGcm10716 | ARM: A Neon store to device memory can result in dropping a previous store | No fix scheduled | 35 |
| ENGcm10701 | ARM: BTB invalidate by MVA operations do not work as intended when the IBE bit is enabled | No fix scheduled | 37 |
| ENGcm10703 | ARM: Taking a watchpoint is incorrectly prioritized over a precise data abort if both occur simultaneously on the same address | No fix scheduled | 39 |
| ENGcm10724 | ARM: VCVT.Q2.32 u32 can return wrong result for the input 0xF00.0F01 in one specific configuration of the floating point unit | No fix scheduled | 41 |
And Then There's Rack Scale...

[Diagram of rack-scale architecture with various components such as CPUs, L1, L2, L3, RAM, Coherent Interconnect, PCI, NIC, TOR, and Backhaul]
There's a Lot of Data Available

- Program trace
- Cache dumps
- Port mirroring
- Event triggers
- Openflow
- Backhaul
ARM High-Speed Serial Trace Port

- Streams from the *Embedded Trace Macrocell*.
- Cycle-accurate control flow + events @ 6GiB/s+
- Compatible with FPGA PHYs.
- Well-documented protocol.
  - Aurora 8/10
- Available on ARMv8

Image: Teledyne Lecroy
The HSSTP Hardware

- The official tool is CHF10,000 per core.
- The cable run is maximum 15cm.
- It's PHY-compatible with common FPGAs
- A CHF6k FPGA could easily handle 10.
  - 15x cheaper!
- We have a development prototype.
HSSTP Testbench
Fancy Triggering and Filtering

- The ETM has sophisticated filtering e.g. Sequencer.
- \( Bn \) and \( Fn \) can be just about any events on the SoC.
- States can enable/disable trace, or log events.
- A powerful facility for pre-filtering
Filtering and Offload in an FPGA

- We'll need to intelligently filter high-rate data.
- We're using an FPGA for the physical interface already.
- How much processing could we do?
- We have expertise in the group with FPGA query offloading
  - We have a Master's student working on this.
What Could We Do With This Data?
Hardware Tracing for Correctness

Are HW operations right?

- Real time pipeline trace on ARM.
- Can halt and inspect caches.
- HW has “errata” (bugs).
- Check that it actually works!
- Catch transient and race bugs.

\[
\exists v, a \rightarrow pa
\]

\[
\exists v, a \rightarrow pa
\]

5Gb/s

Filter at line rate

Log & process offline

Check temporal assertions
Hardware Tracing for Performance

- Should see N coherency messages.
- Do we?
  - The HW knows!

Is URPC optimal?

Log & process offline

 URPC[0] = x;
 URPC[1] = 1;
while(!URPC[1]);
 x = URPC[0];
Properties to Check: Security

- Runtime verification is an established field.
- Lots of existing work to build on.
- What properties could we check efficiently?
- How could we map them to the filtering pipeline?

/* A very simple TESLA assertion. */
TESLA_WITHIN(example_syscall, 
previously(security_check(ANY(ptr), 
o, op) == 0));

http://www.cl.cam.ac.uk/research/security/ctsrd/tesla/
That's a lot of data, how can we process it?

This is what rack-scale systems are for!

We have a software pipeline, thanks to a Master's student: Andrei Pârvu.
Properties to Check: Memory Management

- Could we check this?

```c
void *a = malloc();
...
{a is still allocated}
free(a);
```

It's always been true that...

...if \( x \) is freed now, then...

...before this free...

...there were no frees of \( x \), since it was allocated.
Checking LTL with Automata

This is a well-studied problem, and standard algorithms exist:

\[ Gp \ \text{free}(x) \rightarrow P \ \text{!free}(x) \ S \ x = \text{malloc}; \]
Bound Variables and Multiple Automata

- So far only one \( x \) value.
- Every \( x \) needs an automaton instance.

\[
\begin{align*}
Gp & \; \text{\$free(1)} \rightarrow P \; \text{!\$free(1)} \; S \; 1 = \text{\$malloc}; \\
Gp & \; \text{\$free(2)} \rightarrow P \; \text{!\$free(2)} \; S \; 2 = \text{\$malloc}; \\
Gp & \; \text{\$free(3)} \rightarrow P \; \text{!\$free(3)} \; S \; 3 = \text{\$malloc};
\end{align*}
\]

- Requires dynamic allocation.
- Not trivial in HW.
A Streaming Verification Engine

Sources
- HSSTP
- Packet Capture

Capture
- ETM Sequencer
- FPGA Capture

Processing
- Dataflow Engine
- FPGA Offload

Properties
- TESLA malloc()
- pairing
- Coherence correctness

Constraints

Requirements
Software Pipeline Performance
LTL checking in software

![Graph showing time in seconds against number of events (1000s) for different conditions: no double allocation, no double frees, and no leaks.](image)
Software Pipeline Performance
Trace parsing in software

![Graph showing software pipeline performance](image-url)
Offloading Verification

- Think regular expressions for infinite streams.
- As for REs, we compile a checking automaton.
- Run the automaton in real time and look for violations.
- FPGAs are good at state machines.
Offloading Parsing

- Currently the bulk of the runtime.
- Not as straightforward on the FPGA.
- Current student project.
An Instrumented Rack-Scale System

- 64 SoCs x 5Gb/s = 320Gb/s trace output.
- Online checkers (e.g. automata) will be essential at this scale.
- We're going to build this:
  - A rack of ARMv8 cores & FPGAs.
BRISC
A deadly embrace

Product hardware is designed for current application workloads running on Linux.

Innovation (and research) in system software is constrained by available commodity hardware.
The Gap.

For many commercially relevant workloads, cores spend much of their time in the OS.

**BUT:**

- Processor architects ignore OS designers
  - Simply don’t understand the OS problem
  - Cores rarely evaluated with >1 app running anyway
- HPC people try to remove the OS
  - And then blow the rest of their s/w development budget putting it back in a user library.
- and OS design people?
  - Complain among themselves and try and deal with it
  - Don't even try to influence hardware
A deadly embrace

Product hardware is designed for current application workloads running on Linux.

Innovation (and research) in system software is constrained by available commodity hardware.
Solution: BRISC

- A hardware *research* platform for system software
  - Massively overengineered wrt. products
  - Highly configurable building block for rackscale
Sketch

100 Gb Ethernet

Large server-class SoC

≥ 0.5TB DDR4

SATA, PCIe, UART, NVMe, USB

Coherence

High-end FPGA (e.g. Xilinx Zynq ZU17EG)

~ 32GB DDR4

UART, USB, SD

As many 100Gb QSFP+ cages as possible
All kinds of uses for this...

- Plug lots together for rack-scale computing
- Use the FPGA for data processing offload
- Emulate large distributed NVRAM
- Sequester processors using the FPGA
- Runtime verification of program trace
- Experiment scaling coherency
- Build a dataprocessing network switch
- etc. etc. etc.
Higher goal: research amplification

• Seed the research community
  – Remove major barrier to innovation at a stroke

• Precedents:
  – PlanetLab
  – Berkeley Unix
  – …
Questions?
Checking LTL with Automata

This is a well-studied problem, and standard algorithms exist:

\[
\text{Gp } \neg \text{free}(x) \rightarrow P \land \neg \text{free}(x) \land x = \text{malloc};
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{Gp P, at t-1} & P, at t & \text{Gp P, at t} \\
\text{„P was true until t-1“} & \text{„P is still true at t“} & \text{„P has always been true“} \\
\hline
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\hline
\end{array}
\]