Toward a Trustable, Self-Hosting Computer System

Gabriel L. Somlo, Ph.D.

CReSCT
Cyber Resilient Supply Chain Technologies
2020 Virtual Workshop

Idaho National Laboratory

IEEE

CERT
Carnegie Mellon University
Software Engineering Institute

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM20-0353

Computer Systems are Everywhere

Embedded Computers with *exotic* enclosures and peripherals, e.g.:

- comms
- navigation
- artillery

A Tech Based Supply Chain Workaround?

- Trustworthines of computer hardware is a ***Big Question***
  - Microchips in particular!
- Development / Design / Production supply chains are problematic
  - Multinational corporations
  - Opaque relationships, abundance of NDAs
  - International, highly mobile workforce
- Non-destructive testing & reverse engineering of microchips is ***HARD***
  - Unlike *software*

Hardware Attack Surface

- ASIC Fabrication (Malicious Foundry)
  - masks reverse engineered and modified to insert malicious behavior
    - privilege escalation CPU backdoor
    - compromised random number generator
  - problematic to test/verify *after the fact*!
  - mitigated by using FPGAs instead!

- Compilation (Malicious Toolchain)
  - generates malicious design from clean sources

- Design Defects (Accidentally or Intentionally Buggy HDL Sources)
  - Spectre
  - Meltdown

Field Stripping a Computer

Applications (incl. compiler)		Software (e.g., Linux, BSD, seL4)
System Runtime Libraries		
Kernel		
Hypervisor (optional)		
CPU ISA & I/O Registers		Hardware (e.g., x86, ARM, RISC-V)
Microarchitecture		
Register Transfer Level (RTL)		
ASICs	FPGAs	

Self-hosting:

- a system’s capability to produce new versions of itself, from bounded sources, without reliance on external third-party support*
- the software stack is self-hosting

* Assuming the hardware can be trusted!!!

Field Stripping a Computer

Applications (incl. compiler)		Software (e.g., Linux, BSD, seL4)
System Runtime Libraries		
Kernel		
Hypervisor (optional)		
CPU ISA & I/O Registers		Hardware (e.g., x86, ARM, RISC-V)
Microarchitecture		
Register Transfer Level (RTL)		
ASICs	FPGAs	

Self-hosting:

- a system's capability to produce new versions of itself, from bounded sources, without reliance on external third-party support*
- the software stack is self-hosting

* Assuming the hardware can be trusted!!!

Goal: Extend self-hosting property to encompass hardware, including hardware source-language (HDL) compiler!

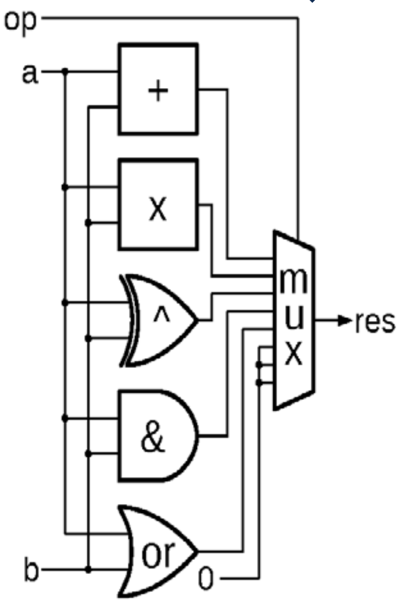
Hardware Development and Compilation Stages

Source Code

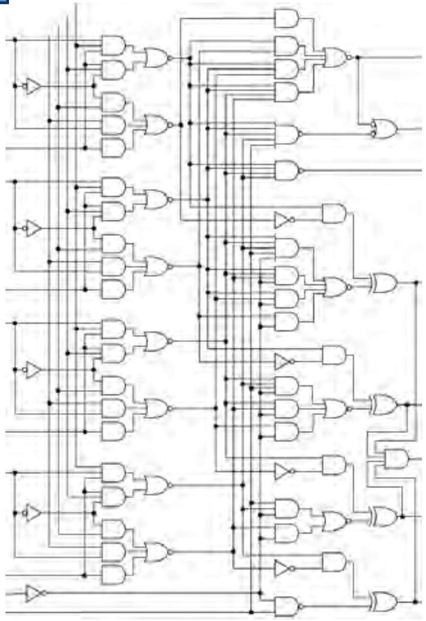
```
module alu_mod (
  // operator:
  input  alu_op_t     op,
  // operands:
  input  logic [31:0] a, b,
  // result:
  output logic [31:0] res);

  always_comb begin
    unique case (op)
      ALU_ADD: res = a + b;
      ALU_MUL: res = a * b;
      ALU_XOR: res = a ^ b;
      ALU_AND: res = a & b;
      ALU_OR : res = a | b;
      default: res = 32'b0;
    endcase
  end
endmodule: alu_mod
```

Elaboration

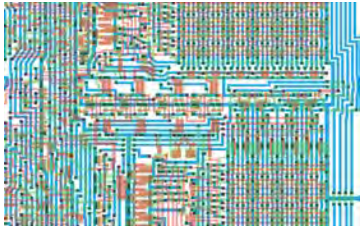


Synthesis, Optimization



Technology Mapping,

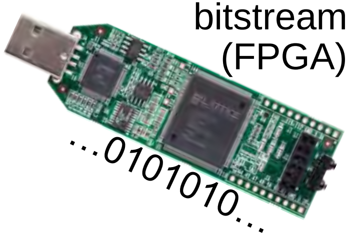
Place-and-Route



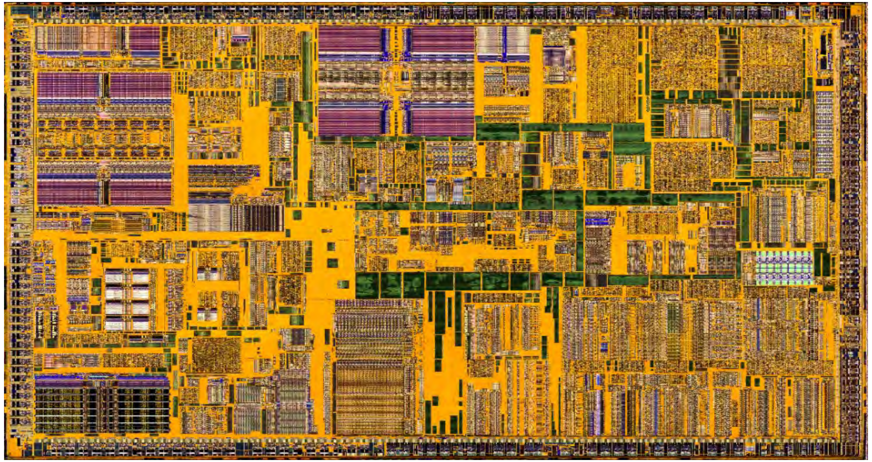
mask (ASIC)

or

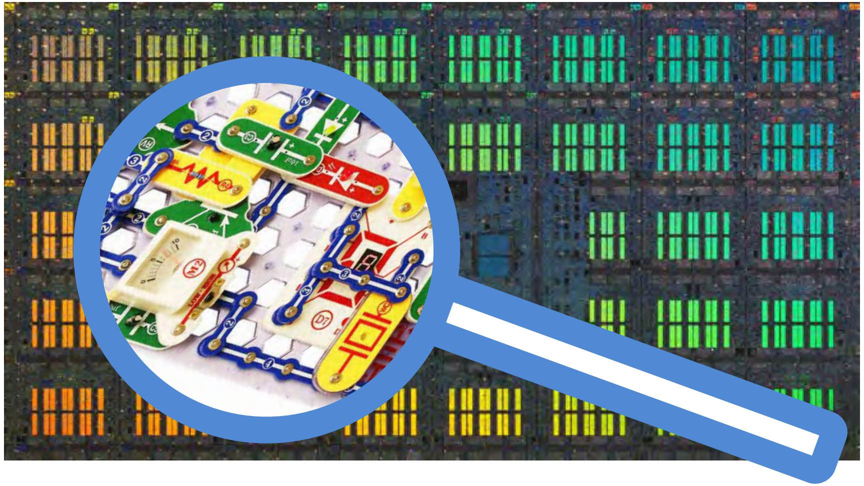
bitstream (FPGA)



ASICs vs. FPGAs

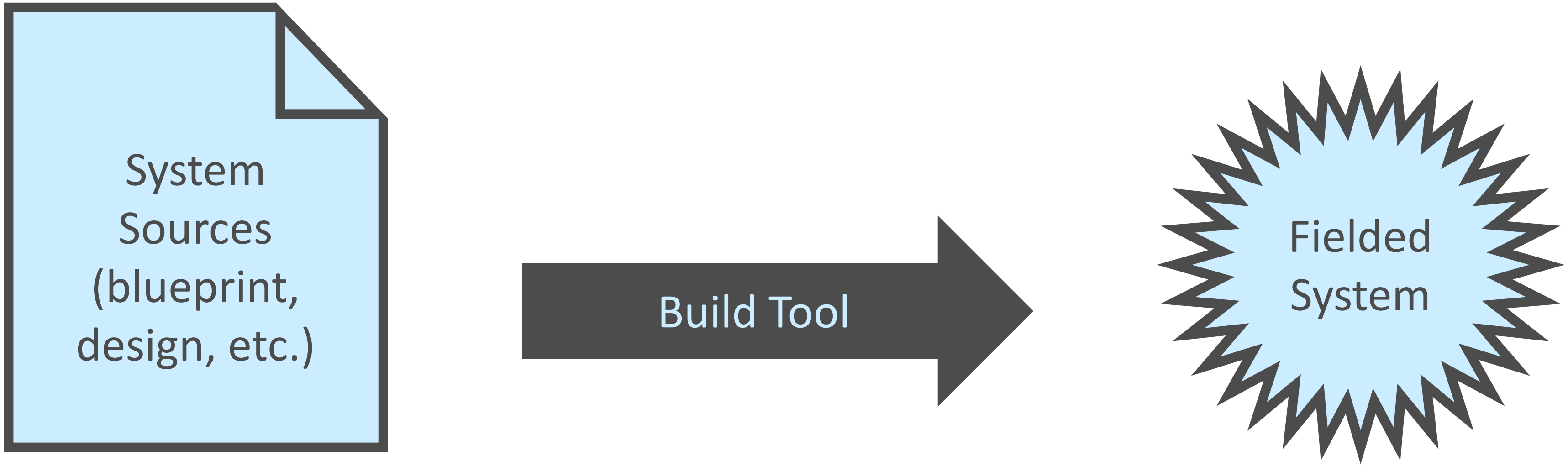


- Application Specific Integrated Circuits
- dedicated, optimized etched silicon
  - [photolithographic masks]
- “hard” IP cores

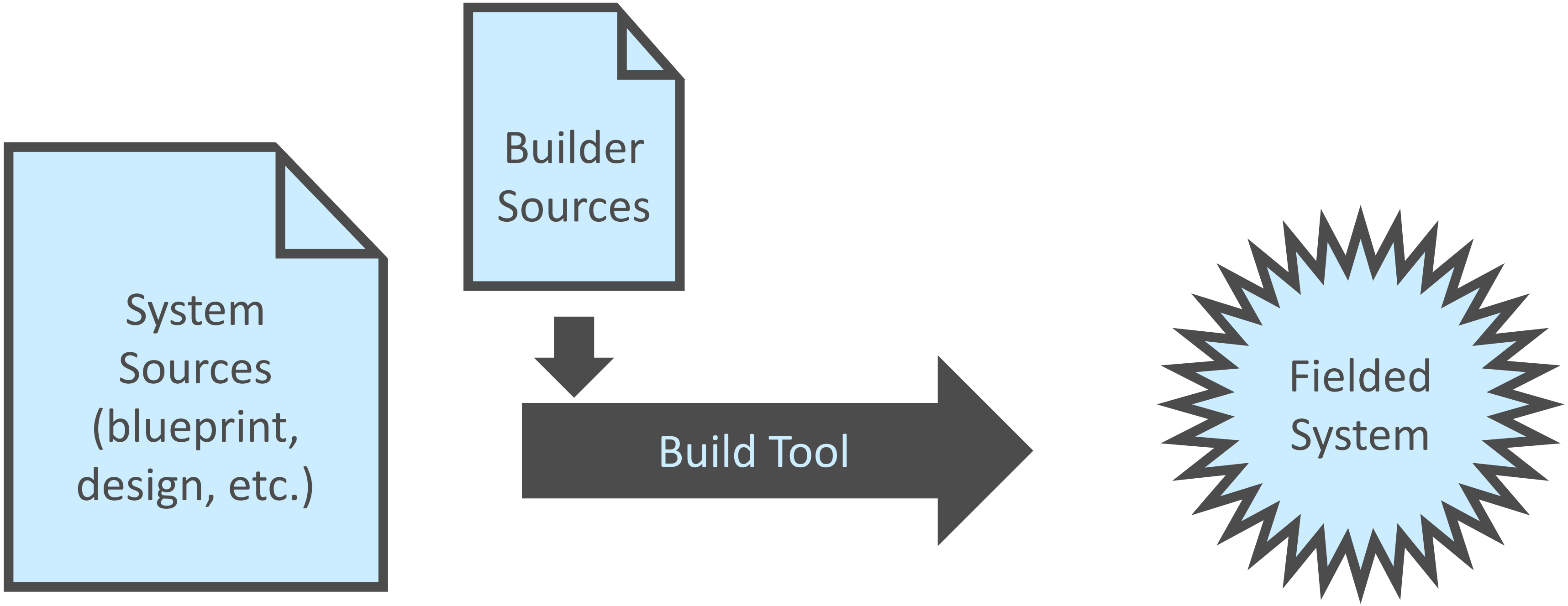


- Field Programmable Gate Arrays
- grid: programmable blocks, interconnect
  - bitstream
- “soft” IP cores

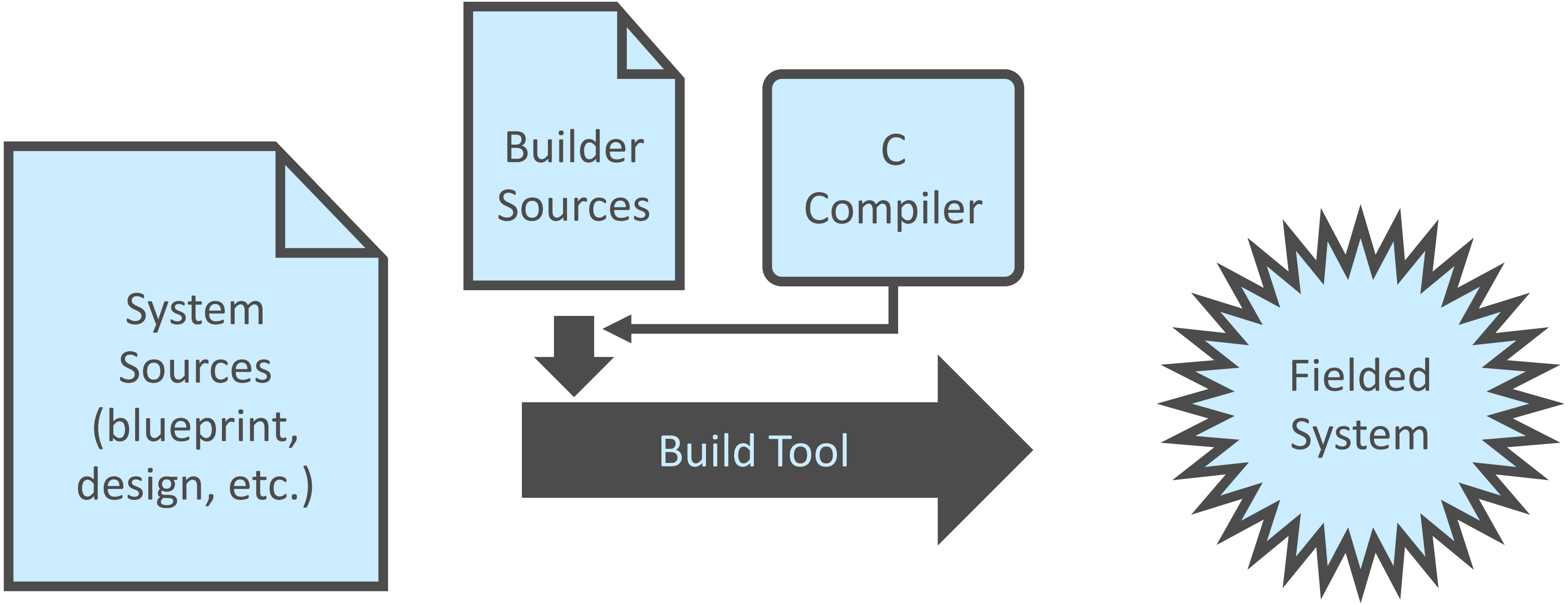
Anchoring Trust for Fielded Systems



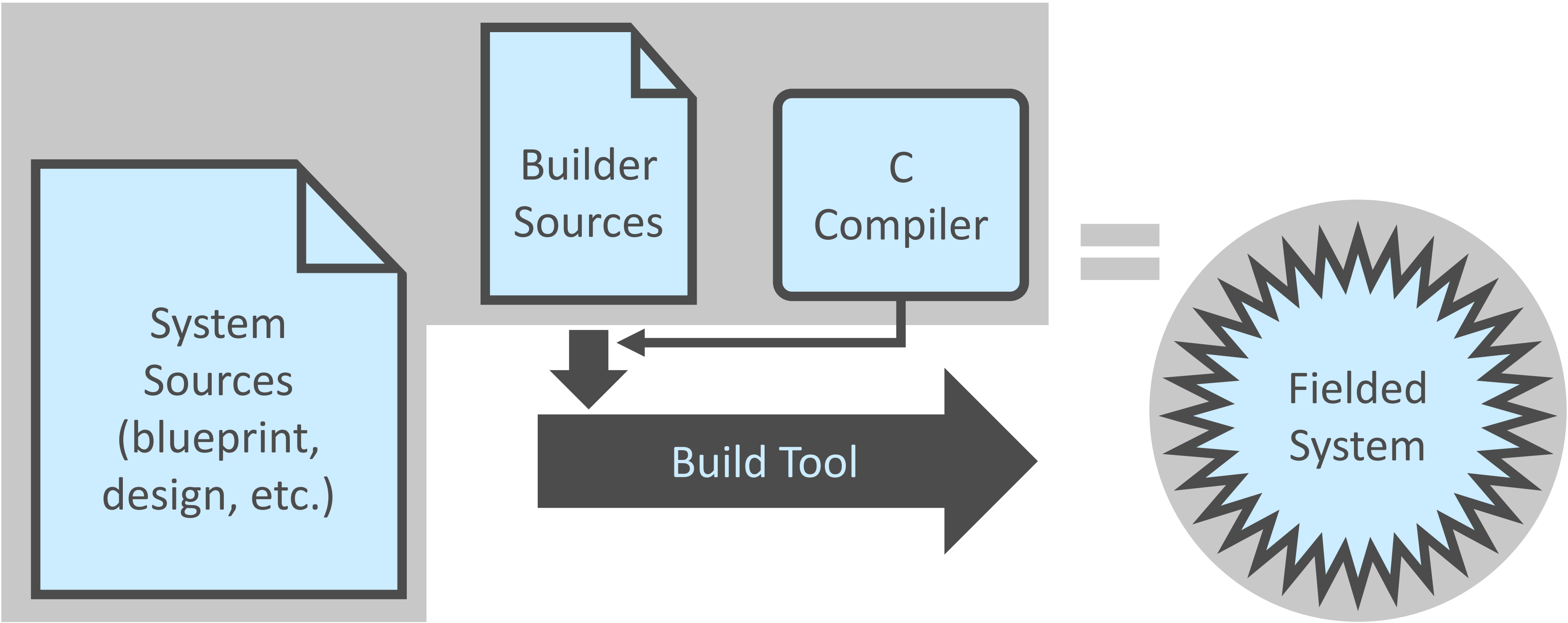
Anchoring Trust for Fielded Systems



Anchoring Trust for Fielded Systems



Anchoring Trust for Fielded Systems



Bootstrapping a Trustworthy RISC-V Cleanroom System

Host (x86/Linux):

- Use DDC to verify we have a clean C (cross-)compiler
- Build clean HDL compiler toolchain, for both x86 and rv64
- Cross-compile target rv64 OS (kernel, libraries, utilities)
- Build rv64 SoC FPGA bitstream, from HDL sources

Target (rv64/Linux):

- Boot up FPGA-based rv64 computer into cross-compiled OS
  - rv64/Linux system is *self-hosting* from this point forward!
- Natively rebuild FPGA bitstream, kernel, libraries, and applications
  - we now have a trustworthy cleanroom
  - guaranteed to “honestly” compile any imported sources (HDL and/or software)!

List of Ingredients

Physical Hardware: FPGA development board (based on Lattice ECP5 series chip):

- Versa-5G or TrellisBoard

Free/Open HDL toolchain (Verilog-to-bitstream):

- Yosys (compiler), Project Trellis (bitstream utilities), NextPNR (place-and-route tool)

Free/Open RISC-V 64-bit CPU:

- Rocket Chip

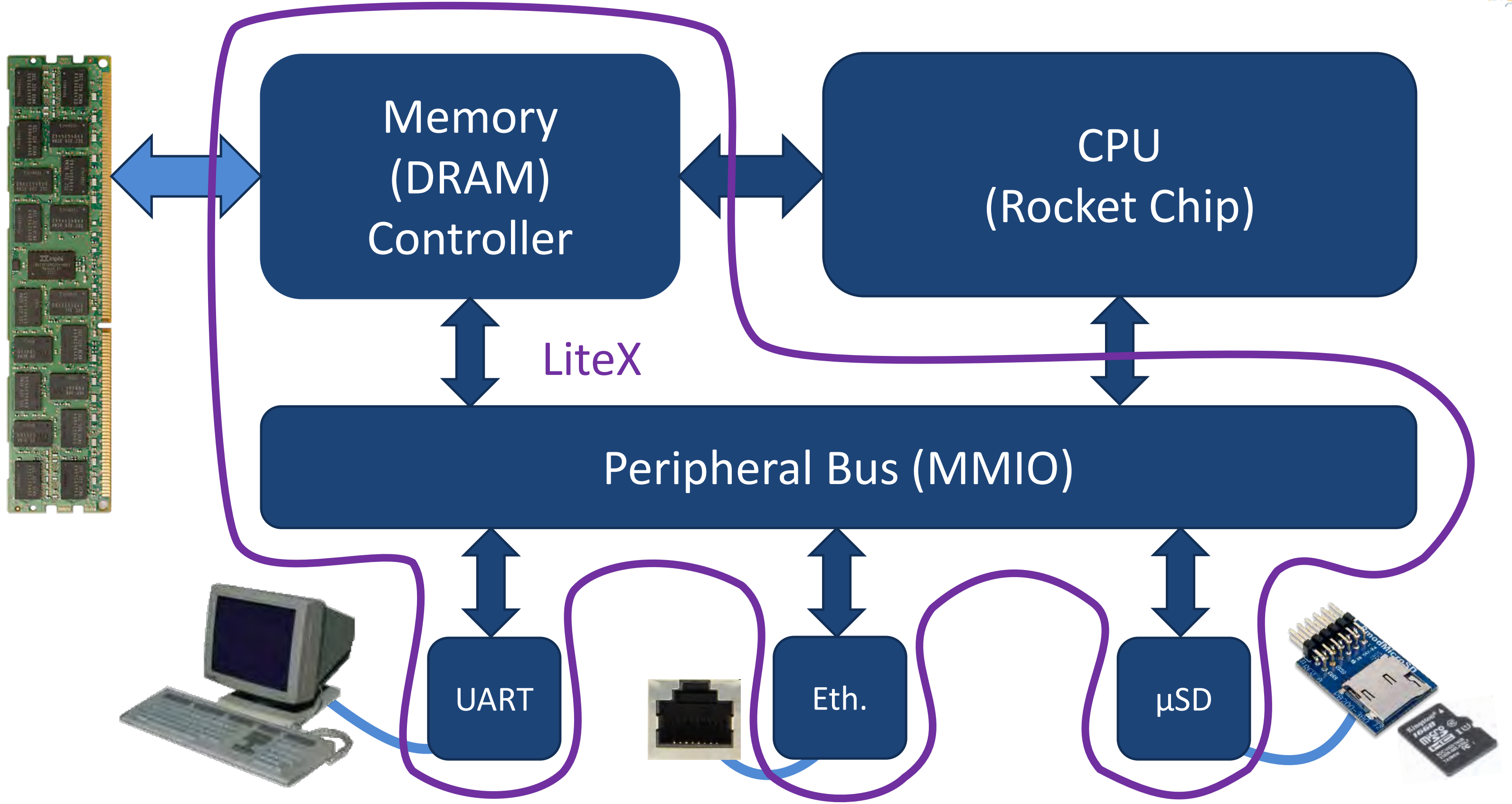
Free/Open system-on-chip (SoC) environment (e.g., system bus, peripherals):

- LiteX

Free/Open software stack (e.g., Linux kernel, glibc runtime, GCC compiler):

- Fedora-riscv64

LiteX + Rocket 64-bit FPGA-based Linux Computer



Benchmarks

CPU	MHz	CoreMark	Linpack (KFLOPS)		nbench					Notes
					P5-90		K6-223			
			Single	Double	Int	Float	Mem	Int	Float	
P5	90	-	-	-	1.00	1.00	-	-	-	reference, P5-90
K6	233	-	-	-	-	-	1.00	1.00	1.00	reference, K6-233
Xeon	2400	12489.07	1679090	1618198	109.59	112.60	34.36	23.05	62.46	native, E5645
rv64gc	-	1468.86	21520	20964	13.38	1.67	2.80	3.81	0.93	QEMU on E5645
u54mc	1400	2079.59	112832	88496	18.21	12.97	3.81	5.19	7.19	SiFive Unleashed
P5	133	282.63	13227	8923	1.77	0.90	0.35	0.53	0.50	Dell Dim. GsMT5133
Rocket	65	47.45	48	31	0.31	.003	.077	.079	.001	LiteX: no FPU
Rocket	60	103.89	84	79	0.47	.003	0.11	0.12	.001	LiteX: gateware FPU
Rocket	50	103.58	5709	4492	0.92	0.67	0.19	0.26	0.37	lowRISC: FPU, cache

Next Steps

NEAR	MID	FAR
Performance Optimizations	Formal Analysis & Verification	Hardware Assurance BCPs
• Early prototype HDL is a target-rich environment for further performance improvements, e.g.,: • 64bit AXI system bus • separate RAM and MMIO data paths	• Starting from a *bounded* set of sources, 100% *as trustworthy as* the fielded system. • Goal: measure *actual* ability to trust the system by conducting source code analysis!	• Supply chain complexity mitigated by hardware openness, auditability

In Conclusion…

- ***Side-stepping*** supply chain questions re. hardware assurance

- ***FPGAs*** mitigate against malicious foundry (silicon) backdoors

- ***Field Stripping*** computers (from complete sources) to determine trustability of:
    - build tools
    - fielded end-product systems

Demo: Linux on Rocket+LiteX (on ECP5 FPGA)

http://www.contrib.andrew.cmu.edu/~somlo/BTCP

Backup Slides

C Compiler vs. "Trusting Trust": Problem & Workaround

- self-propagating C compiler hack (Ken Thompson)
  - malicious compiler inserts Trojan during compilation of a *victim program*
    - clean source → malicious binary
      - including *compiler's own* sources!
    - compiler source hack *no longer needed* after 1st iteration!
- David A. Wheeler's defense: Diverse Double Compilation
  - suspect compiler A: sources S_A, binary B_A
  - trusted compiler T: binary B_T

$S_A \rightarrow B_A \rightarrow X$ $\qquad$ $S_A \rightarrow B_T \rightarrow Y$

- X and Y are functionally identical, but different binaries

$S_A \rightarrow X \rightarrow X_1$ $\qquad$ $S_A \rightarrow Y \rightarrow Y_1$

- X_1 and Y_1 must be identical binaries (since X, Y were functionally identical)!

Related Topics

Diminishing distinction between civilian and military/industrial security posture:

- Bruce Schneier blog post: https://www.lawfareblog.com/myth-consumer-security
- Ability to source trustworthy microchips drowned out by consumer market
- https://youtu.be/1uCy-T22el8?t=132

Right To Repair:

- automobiles, electronics, agricultural machinery
- issues of ownership, control, trust: all aspects of security