RESEARCH REVIEW 2019

Field Stripping a Weapons System: Building a Trustworthy Computer

Gabriel L. Somlo, Ph.D.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Q:

Could you ask a vendor for *full* software *and* hardware *sources* to *any* system or solution contracted by the DoD, *today*?

Myth:

“It is no longer possible for a single person to fully understand how a computer works.”

© 2019 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Field Stripping

From dictionary.com:

To take apart (a weapon) for cleaning, lubrication, and repair or for inspection

© 2019 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Field Stripping: What About Modern Weapons Systems?

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

- artillery
- navigation
- comms

Non-destructive testing & reverse engineering is relatively easy with *software*

- less so with *microchips*!

© 2019 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

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!

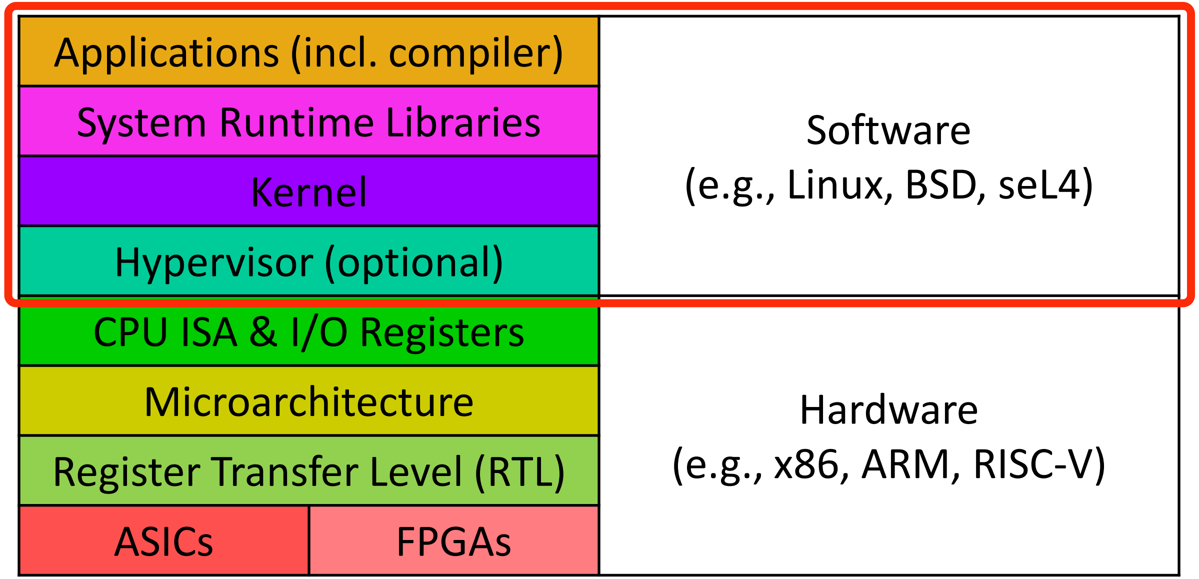
- Compilation (Malicious Toolchain)
  - generates malicious design from clean sources

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

© 2019 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Field Stripping a Computer



Self-hosting:

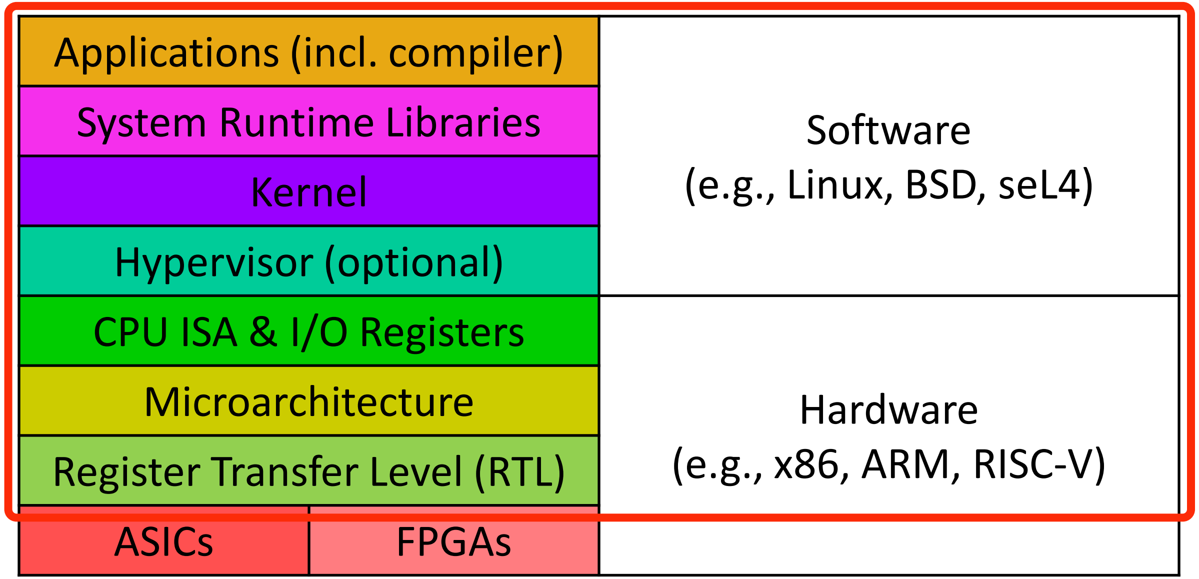
- a system's capability to produce new versions of itself, from bounded sources, without reliance on external third-party support*
- the software stack is self-hosting

* Assuming the hardware can be trusted!!!

© 2019 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Field Stripping a Computer



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!

© 2019 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

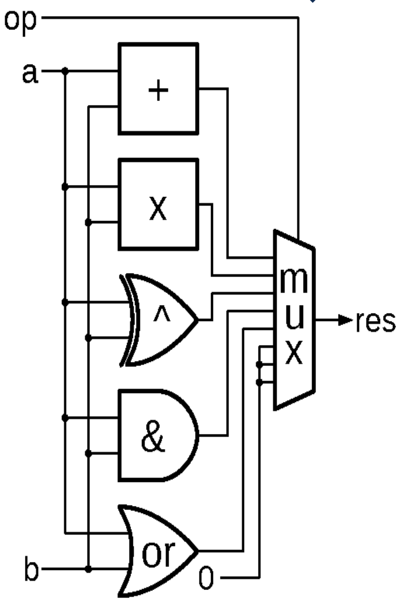
Hardware Development and Compilation Stages

Source Code

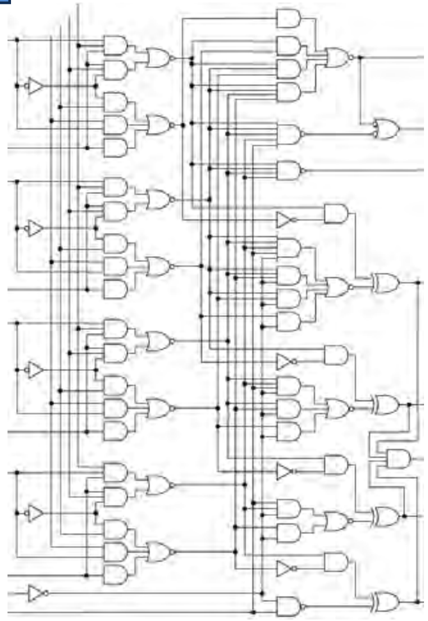
```
module alu_mod (
  // operator:
  input  alu_op_t     op,
  // operands:
  input  logic [31:0] a, b,
  // result:
  output logic [31:0] res);

  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

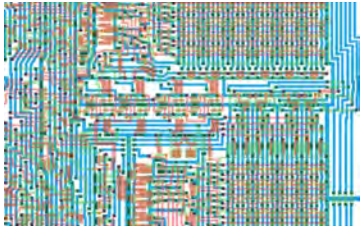


Synthesis, Optimization



Technology Mapping,

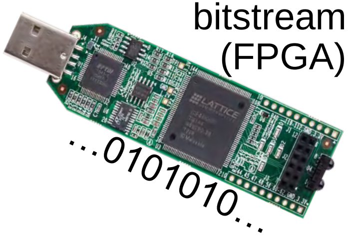
Place-and-Route



mask (ASIC)

or

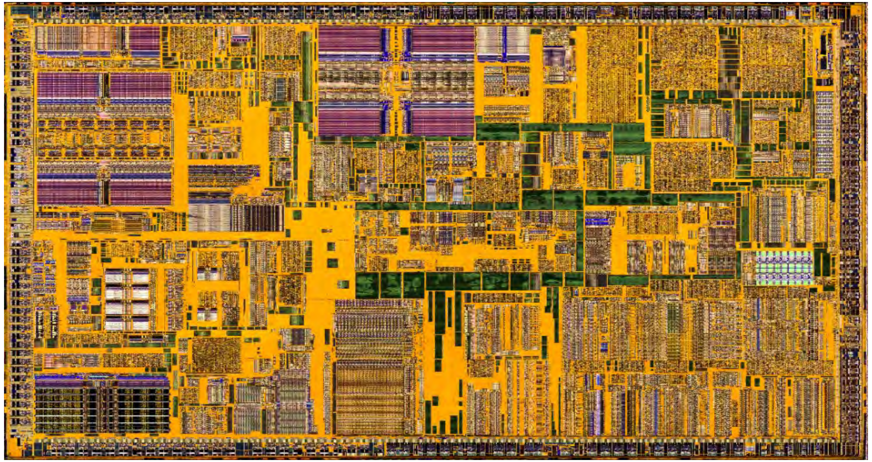
bitstream (FPGA)



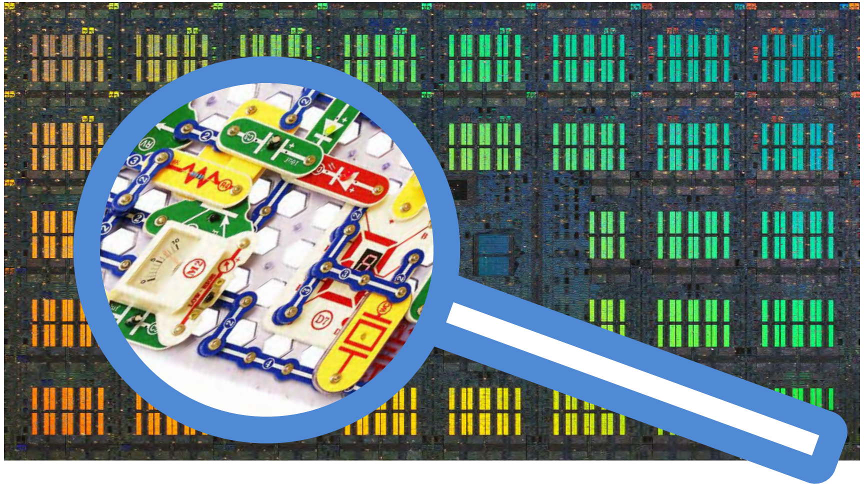
© 2019 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

ASICs vs. FPGAs



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

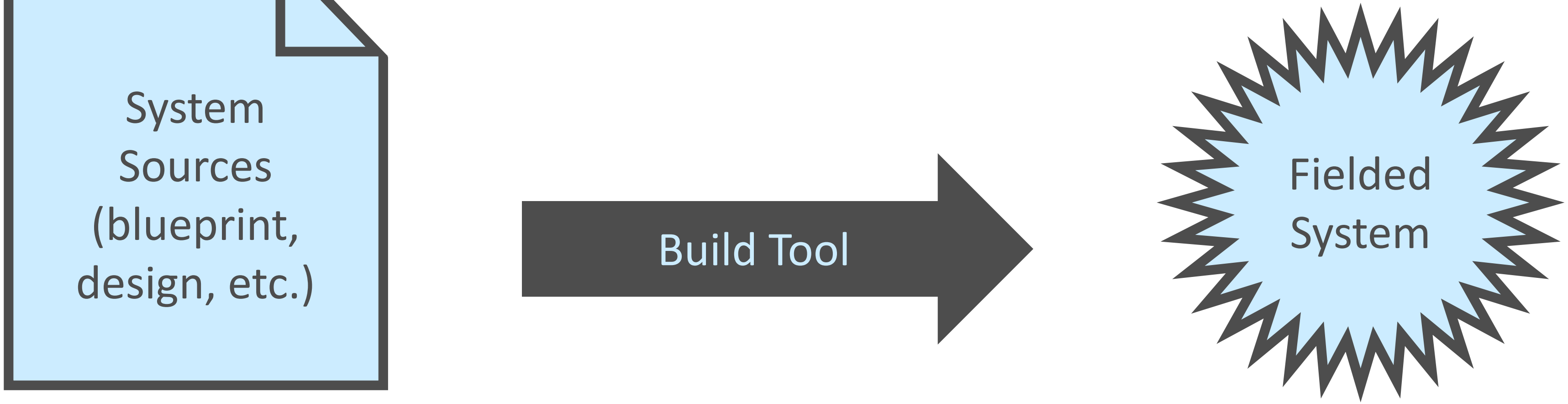


- Field Programmable Gate Arrays
- grid: programmable blocks, interconnect
  - bitstream
- “soft” IP cores

© 2019 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

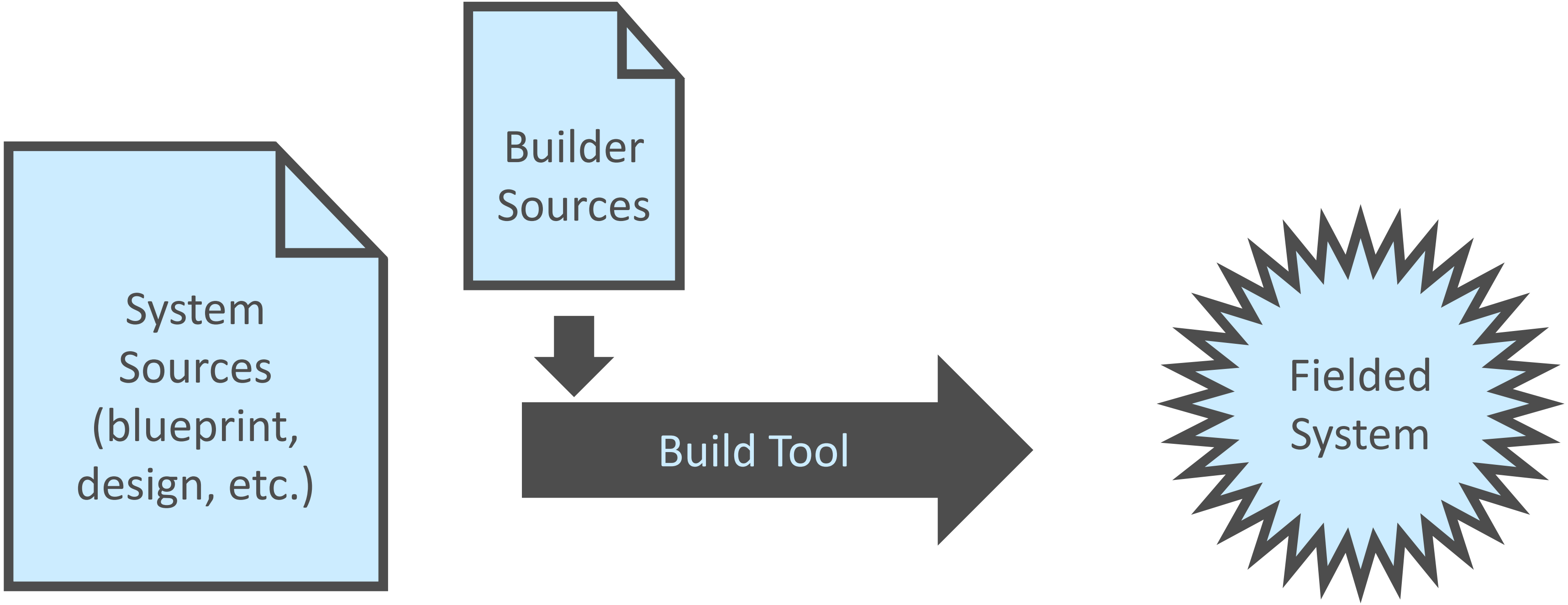
Anchoring Trust for Fielded Systems



© 2019 Carnegie Mellon University

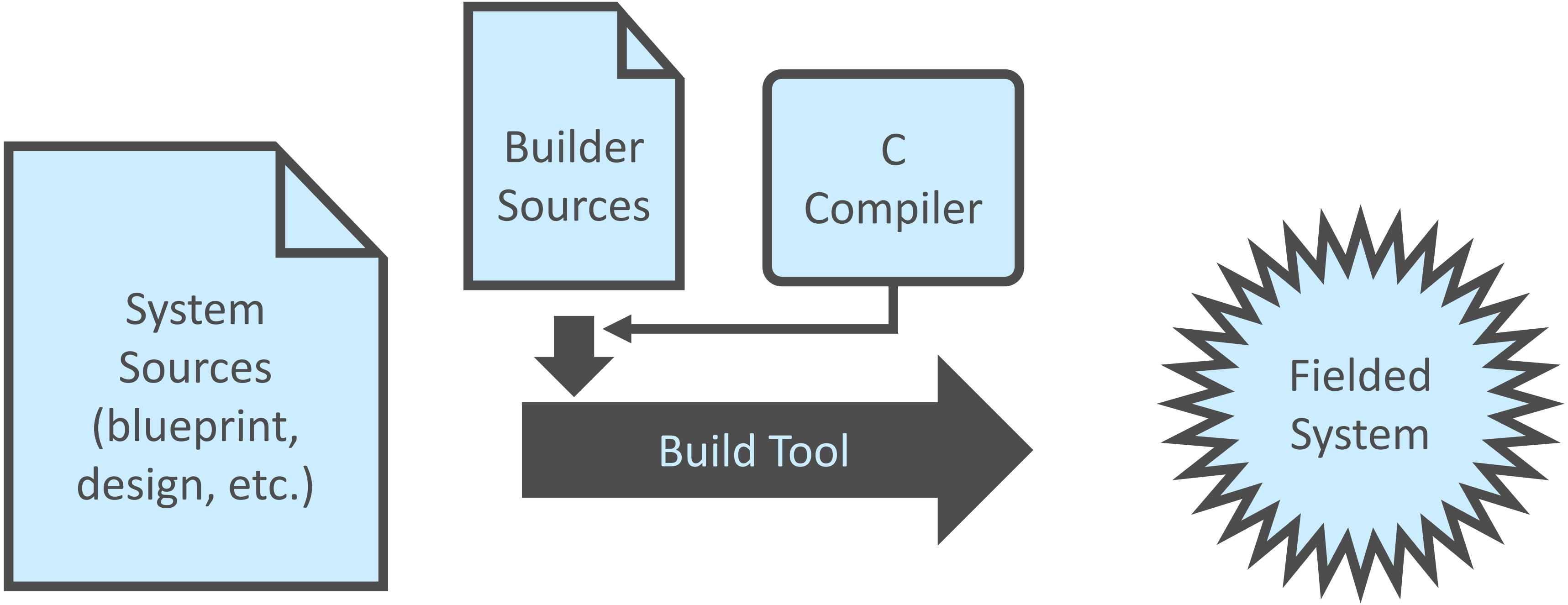
[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Anchoring Trust for Fielded Systems



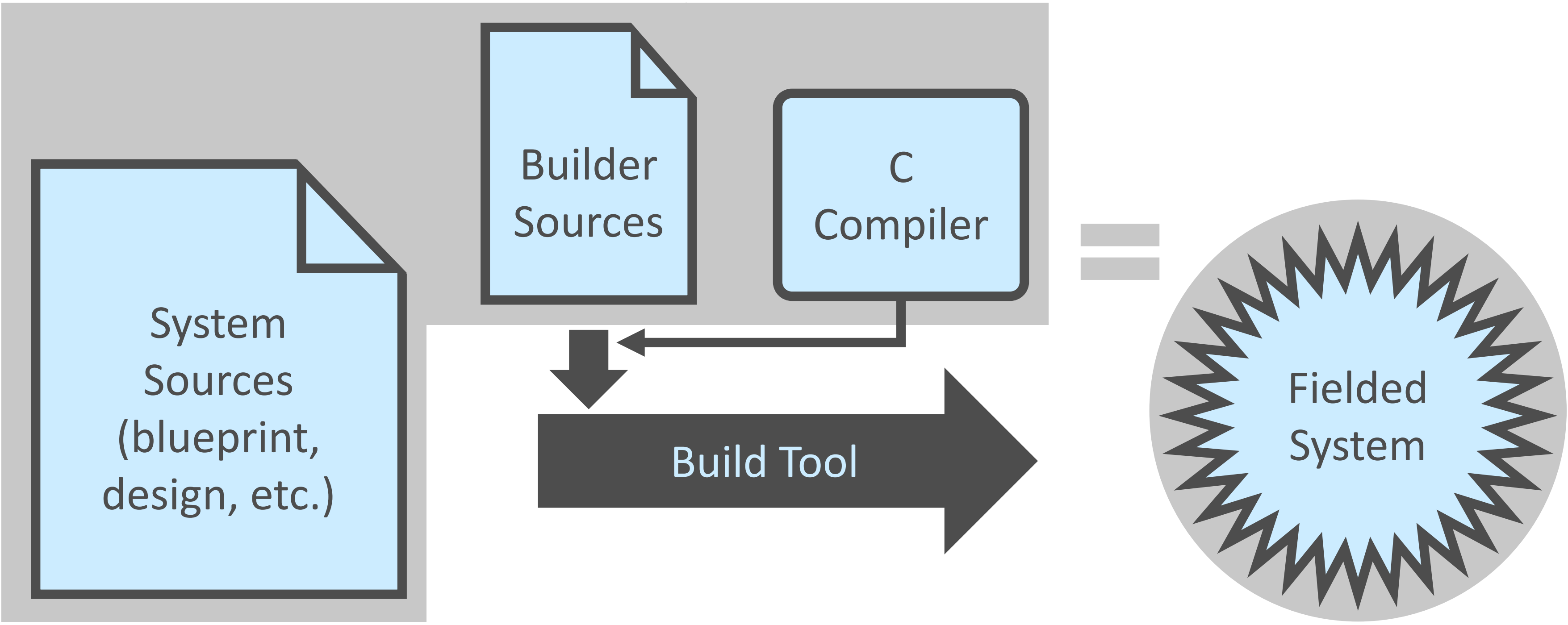
© 2019 Carnegie Mellon University
[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Anchoring Trust for Fielded Systems



© 2019 Carnegie Mellon University
[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Anchoring Trust for Fielded Systems



© 2019 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Bootstrapping a Trustworthy RISC-V Cleanroom System

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

- [rv64/Linux]: Boot up FPGA-based rv64 computer into cross-compiled OS
  - rv64/Linux system is *self-hosting* from this point forward!
- [rv64/Linux]: 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)!

© 2019 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

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

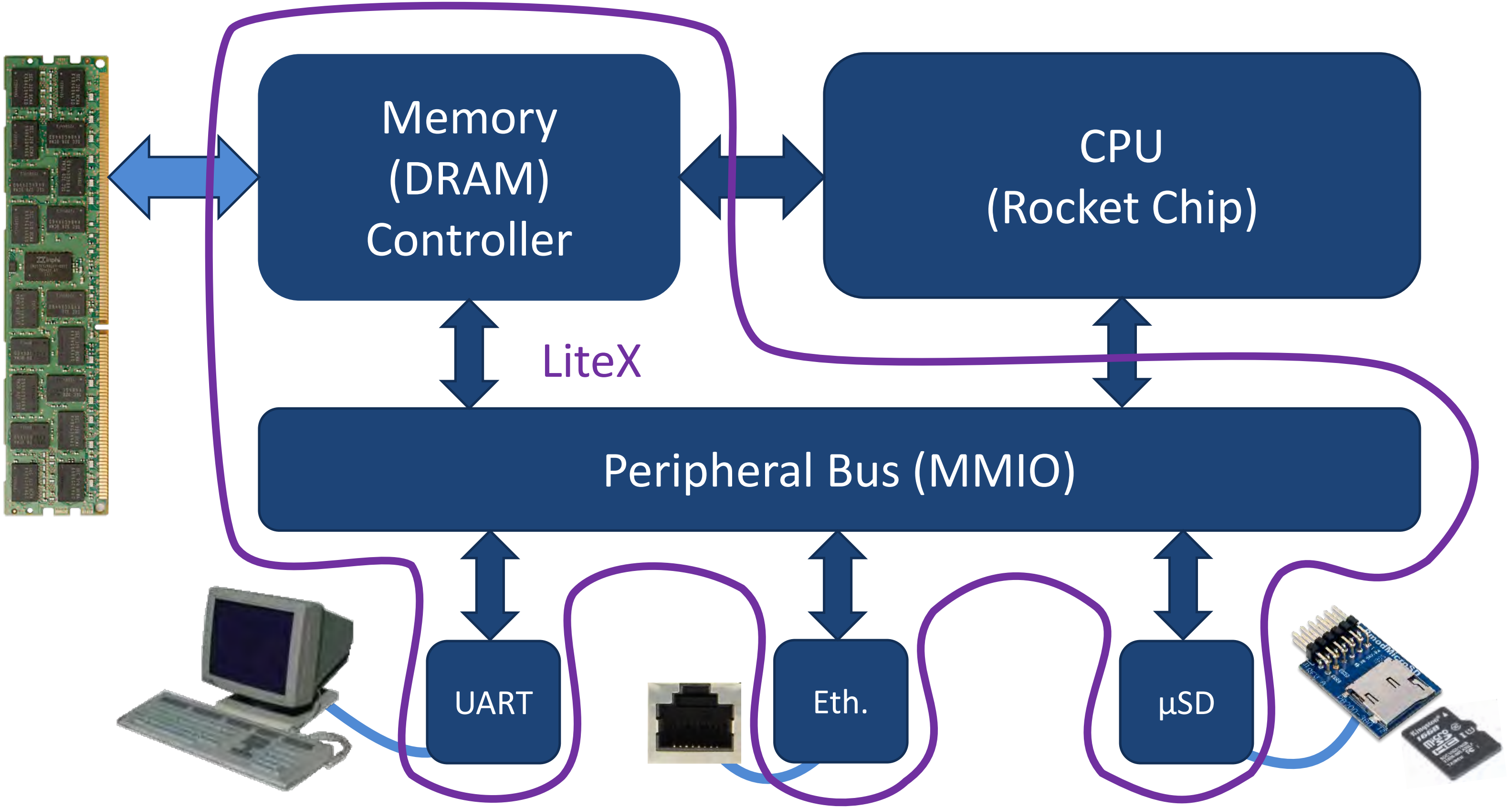
Free/Open software stack (e.g., Linux kernel, glibc runtime, GCC compiler):

- Fedora-riscv64

© 2019 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

LiteX + Rocket 64-bit FPGA-based Linux Computer



© 2019 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Next Steps

NEAR	MID	FAR
Performance Optimizations	Formal Analysis & Verification	Hardware Assurance BCPs

NEAR — Performance Optimizations

- Early prototype HDL is a target-rich environment for further performance improvements, e.g.,:
  - 64bit AXI system bus
  - separate RAM and MMIO data paths

MID — Formal Analysis & Verification

- 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!

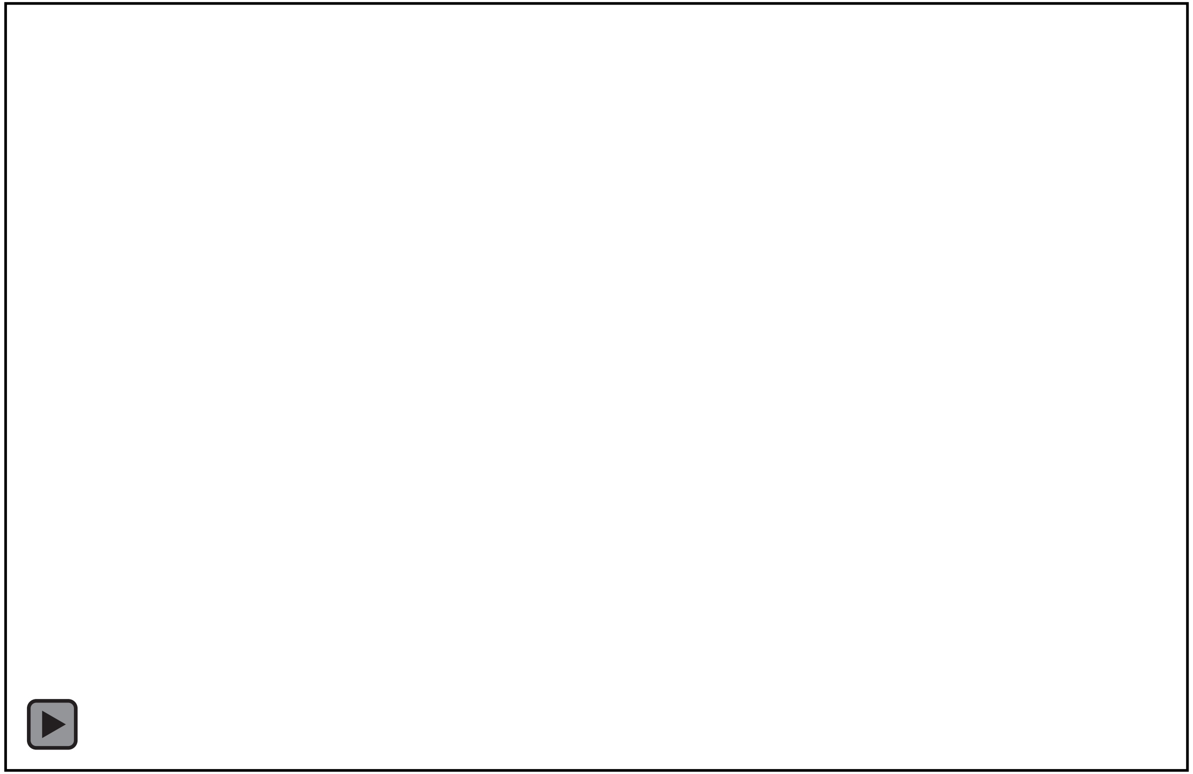
FAR — Hardware Assurance BCPs

- Cyber weapons as trustworthy as kinetic, despite supply chain complications!

© 2019 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Demo: Linux booting on Rocket+LiteX on ECP5 FPGA



© 2019 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Backup Slides

© 2019 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Related Topics

Diminishing distinction between civilian and military/industrial security:

- Bruce Schneier blog post: https://www.lawfareblog.com/myth-consumer-security
- DoD 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

© 2019 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

C Compiler vs. "Trusting Trust": Problem and 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)!

© 2019 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.