Building a Trustworthy Computing Platform

Gabriel L. Somlo, Ph.D.
<glsomlo@cert.org>

SEI, CERT Division
Carnegie Mellon University
Pittsburgh, PA 15213

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.

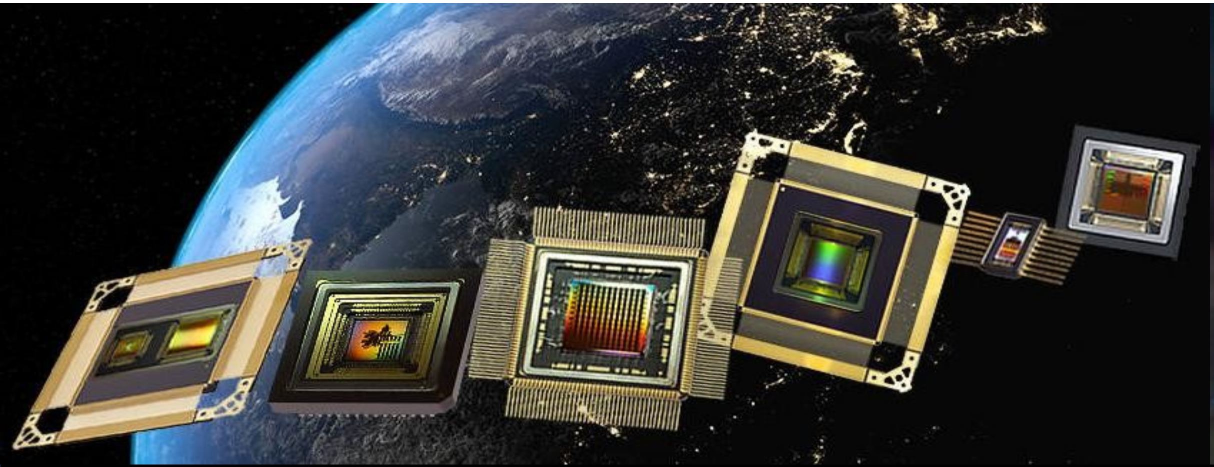
DM19-0566

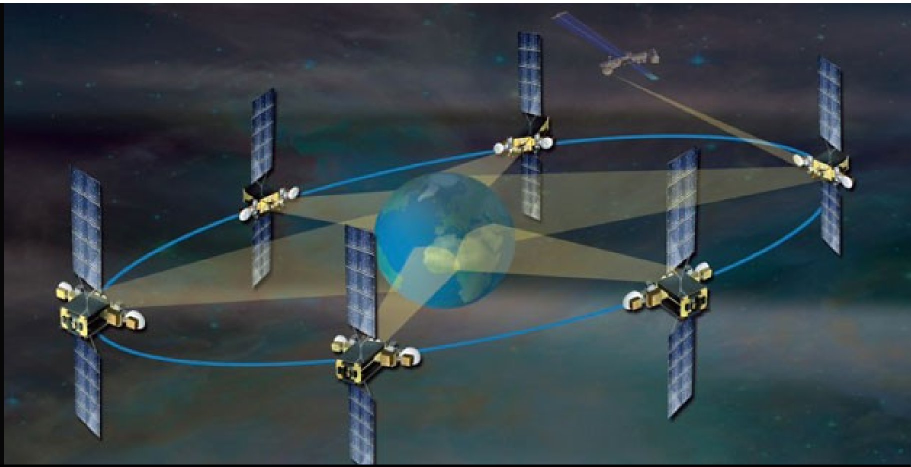
Bottom Line Up Front

- Research: µChip Trojans, Backdoors easy to insert!
- Need trustworthy comp. platform (Soft & Hardware)
  - Trust anchor: full set of sources to *all* components
    - available to (buildable by) system owner
    - *Self-hosting* property: no external black-box dependencies!
- Obtaining full, buildable sources to proprietary vendor HDL toolchains is challenging
- Goal: Proof of Concept with Open Source EDA toolchain

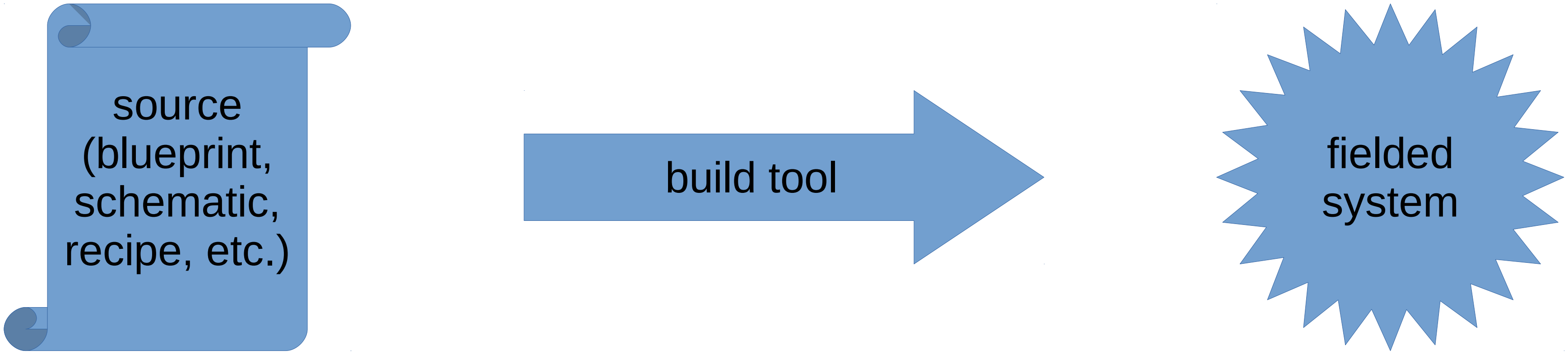
Motivation

- Off-shored µChip Design, Development, Fabrication
- Known (methods to insert) hardware privilege escalation bugs during all stages of Hw. lifecycle
- Need capability to validate Hardware+Software as comprehensive, self-contained stack

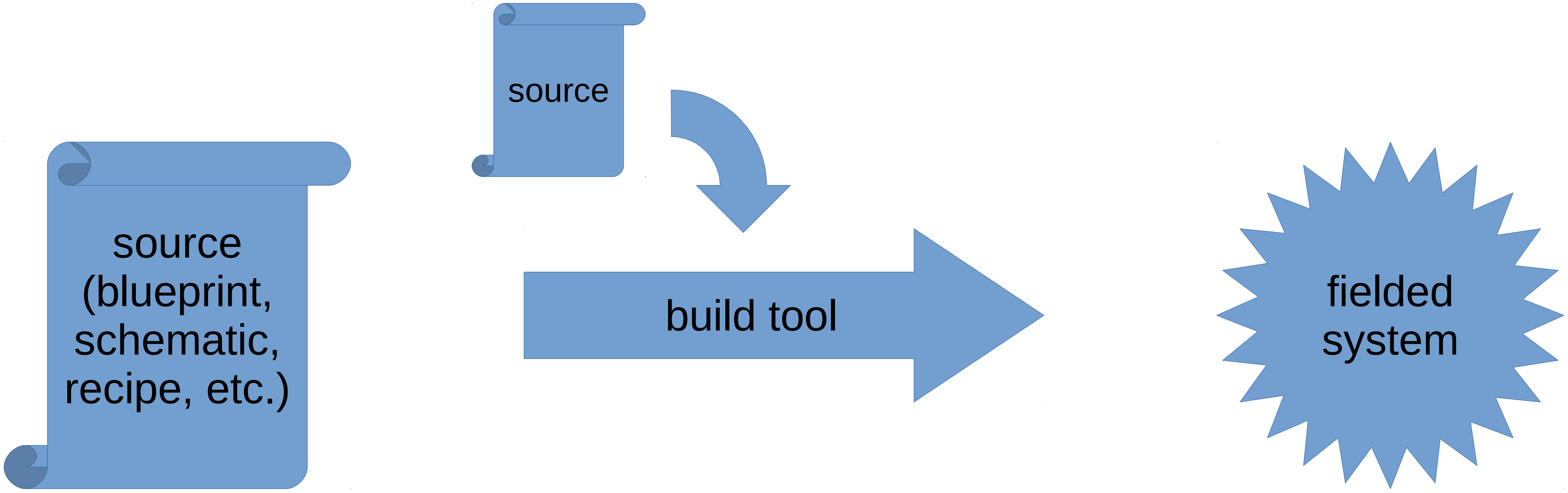




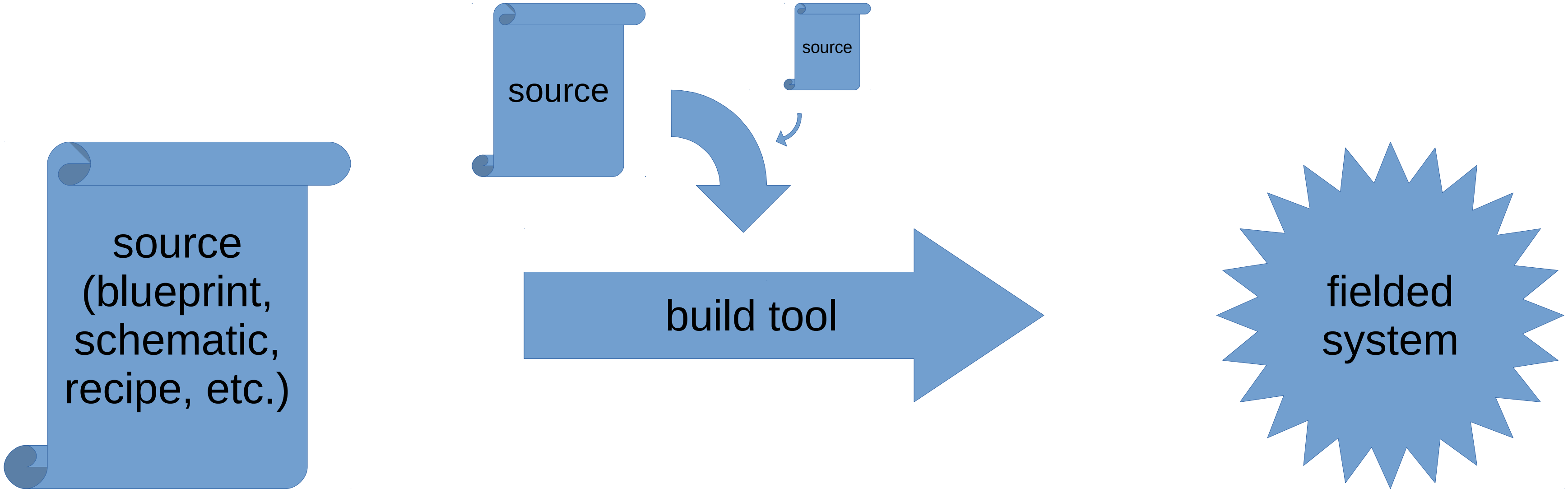
Trust Anchors for Fielded Systems



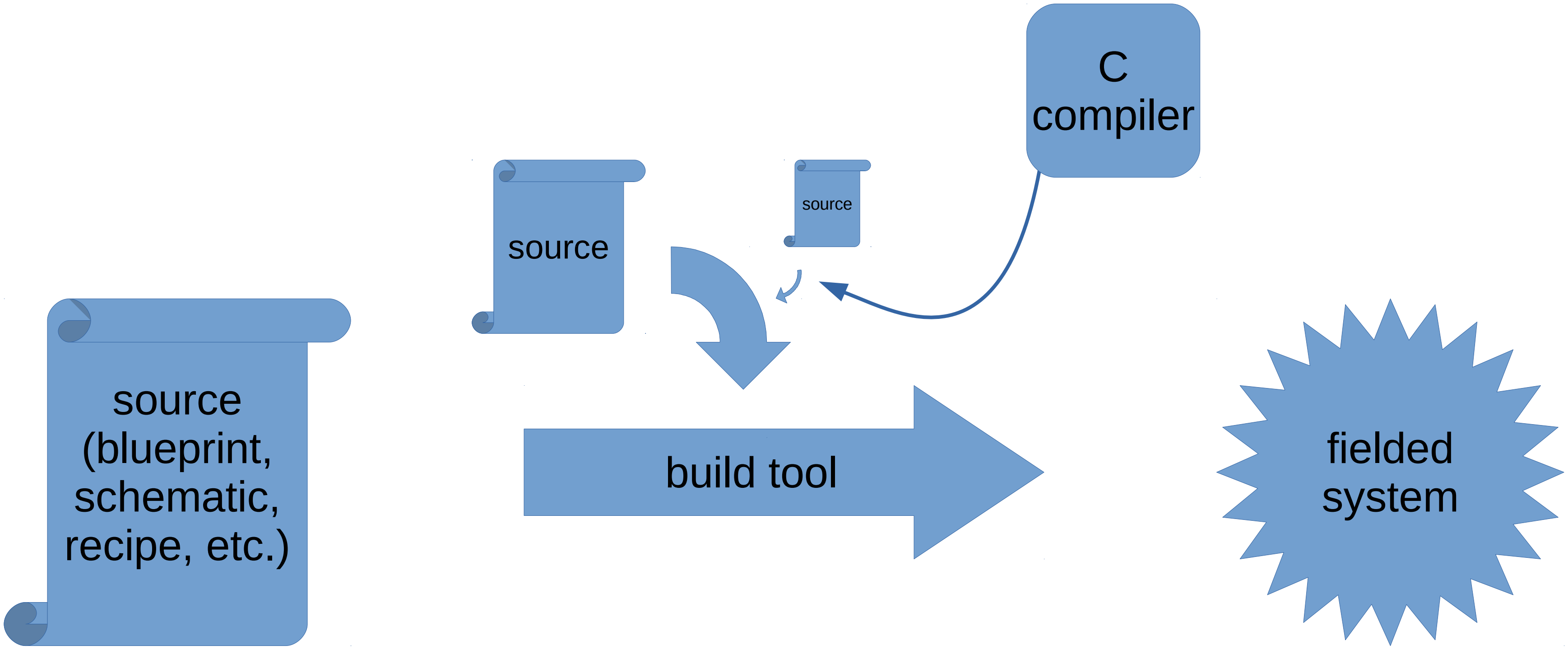
Trust Anchors for Fielded Systems



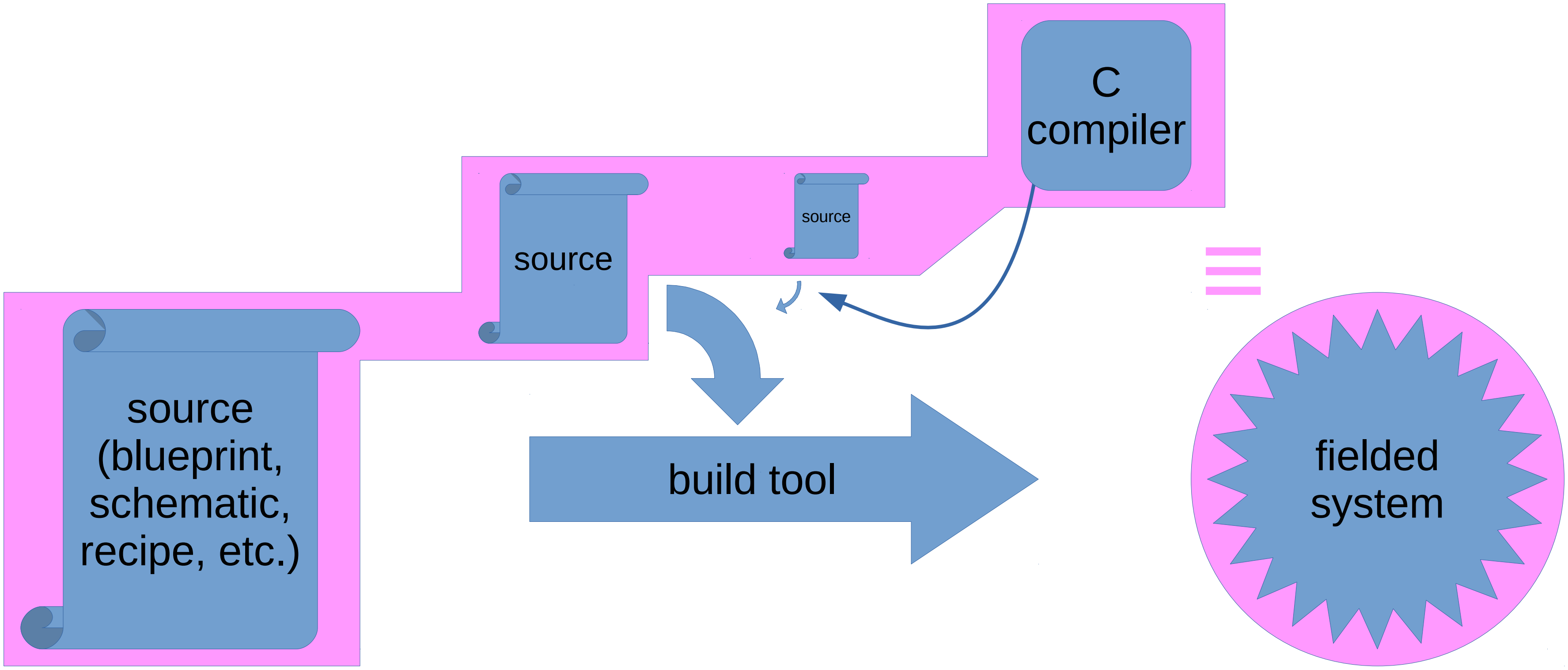
Trust Anchors for Fielded Systems



Trust Anchors for Fielded Systems



Trust Anchors for Fielded Systems



Trusting Trust: Problem and Solution

- Self-propagating compiler hack (Ken Thompson)
  - Malicious C compiler inserts Trojan during *victim program* build
    - 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: source S_A, binary B_A
  - Trusted compiler T: binary B_T

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

- X and Y are functionally identical, but different binaries

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

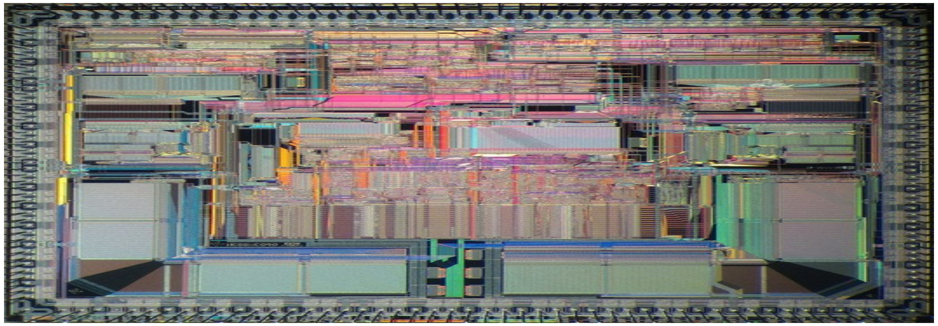
- X_1 and Y_1 must be identical binaries (since X, Y functionally identical)

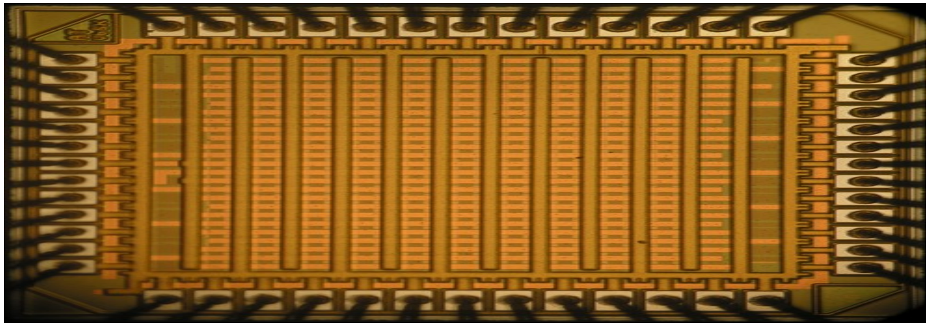
Attack Surface Spans All Stages

- Even for steps performed inside the US!

- Flaws in HDL sources (design):
  - e.g., Meltdown & Spectre (unintentional or malicious)
- Compromised compiler tool chain:
  - Clean HDL source → Malicious masks or bitstream
- Malicious ASIC foundry:
  - Insertion of Trojan privilege escalation backdoor
  - Compromise of encryption strength

ASIC vs. FPGA





- Application Specific Integrated Circuit
- Dedicated, optimized etched silicon
  - Photolitho. masks
- "Hard" IP cores

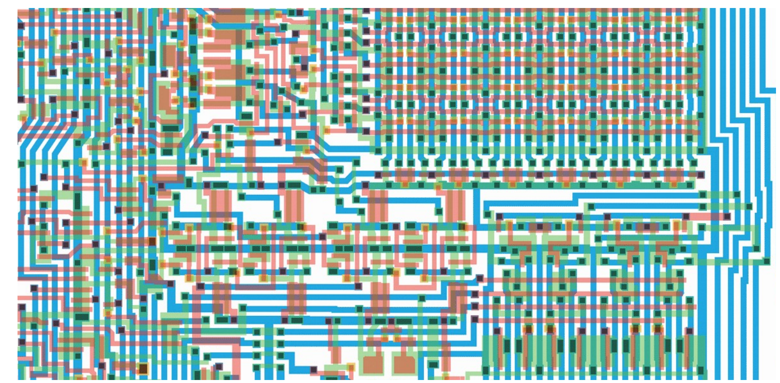
- Field Programmable Gate Array
- Grid of programmable blocks + interconnect
  - Bitstream
- "Soft" IP cores

µChip Design, Devel., Fabrication

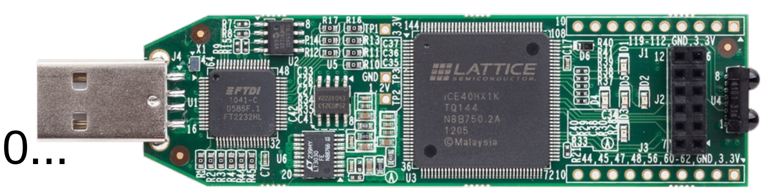
- HDL (Hw. Descr. Language): Verilog, VHDL
  - Functional / Declarative *programming*!
  - Compiled (via tool chain) into masks or bitstream

```
module alu_mod (
  input  alu_op_t     op,
  input  logic [31:0] a, b,
  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
```



...0101010...

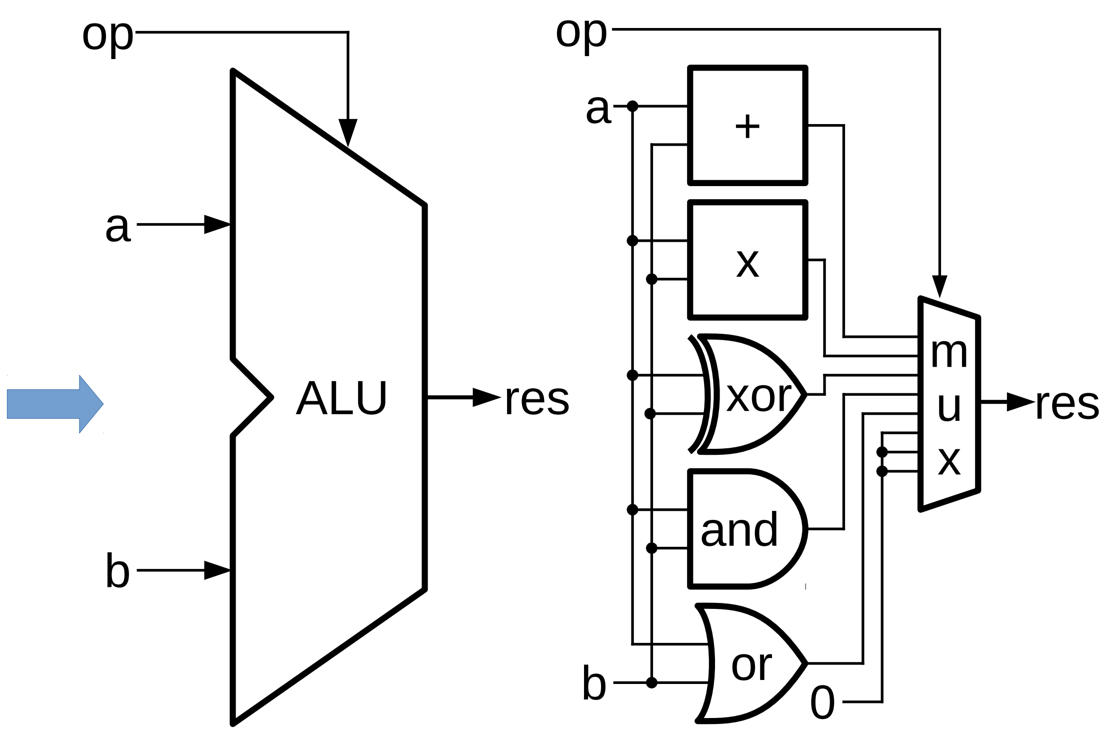


µChip Design, Devel., Fabrication

- Elaboration Stage
  - HDL constructs → Standard Library Blocks

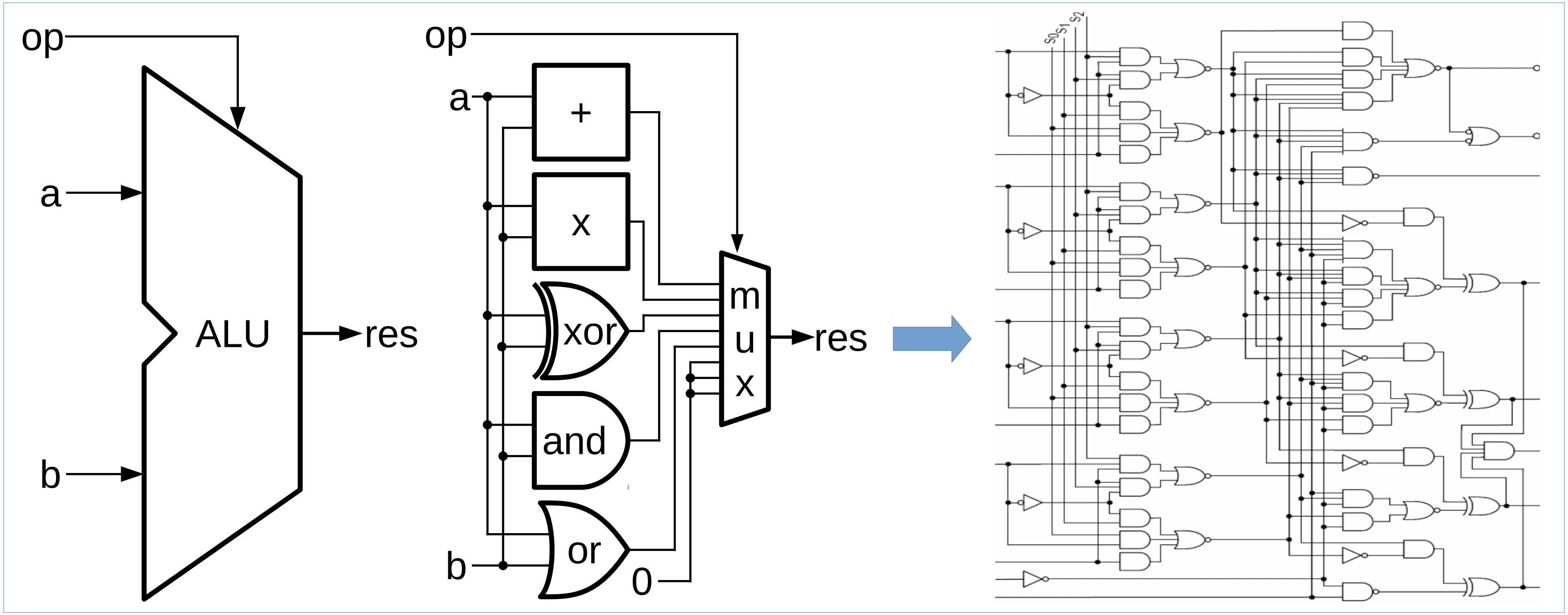
```
module alu_mod (
  input   alu_op_t     op,
  input   logic [31:0] a, b,
  output  logic [31:0] res);

  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
```



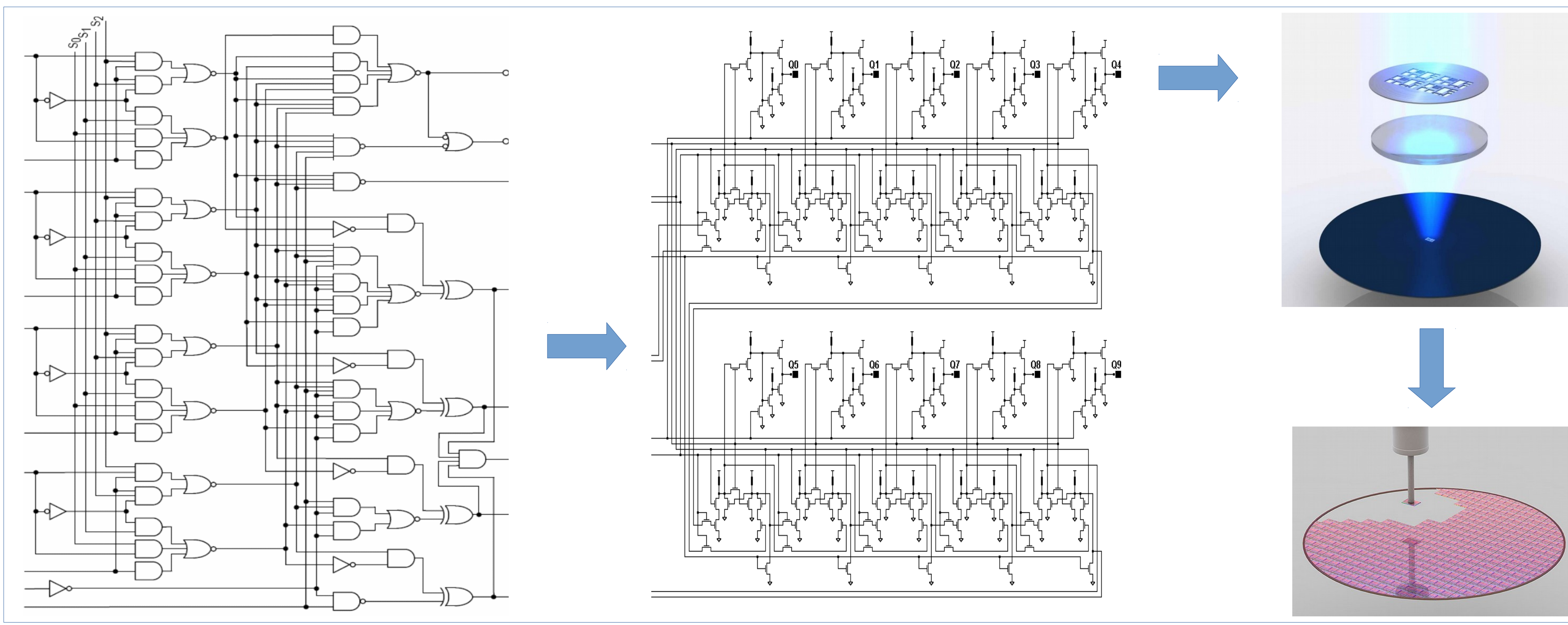
µChip Design, Devel., Fabrication

- Logic Synthesis and Optimization:
  - Library Blocks → Logic Gates



µChip Design, Devel., Fabrication

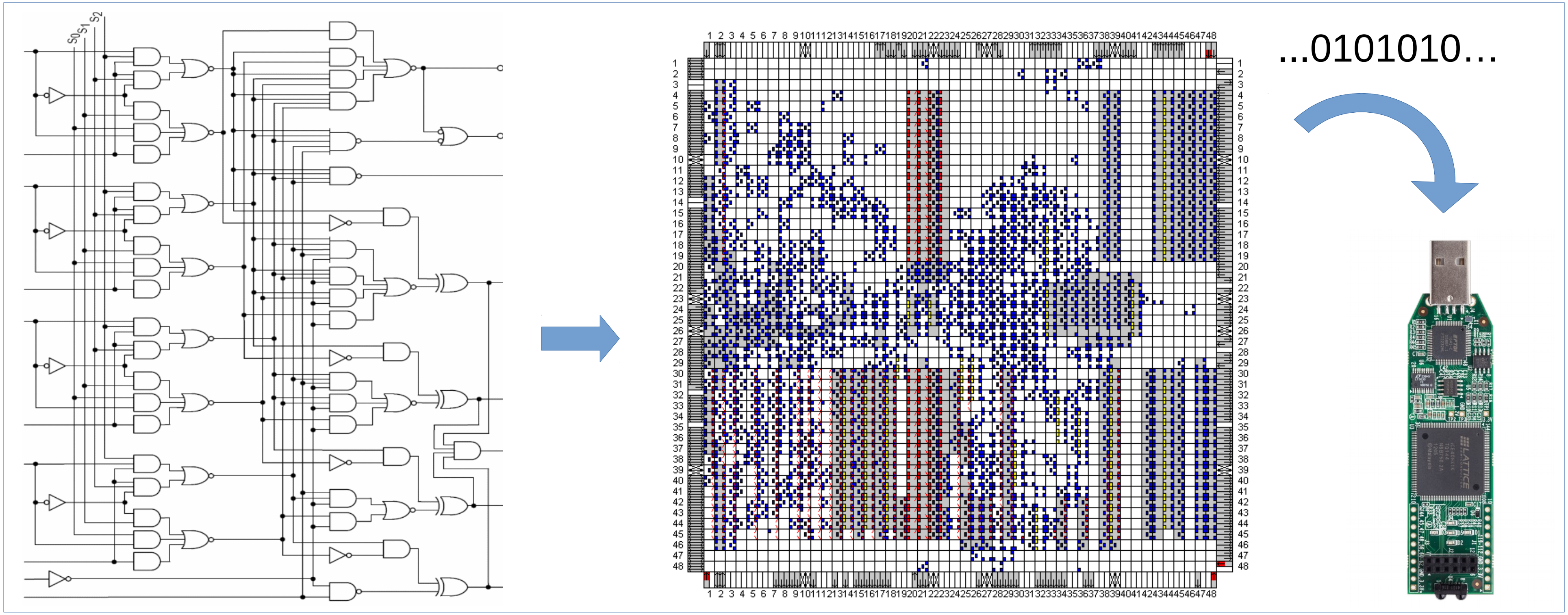
- Technology Mapping, Placement & Routing:
  - ASIC: Gates → Transistors → Photolitho. Masks



µChip Design, Devel., Fabrication

- Technology Mapping, Placement & Routing:
  - FPGA: Gates → CLBs → Bitstream

    (CLB = Configurable Logic Block)



Recommendations

- Prefer FPGAs and Soft IP cores
  - Most severe malicious-foundry attacks based on understanding purpose, functionality of ASIC masks
    - Using FPGAs withholds knowledge necessary for targeting privilege escalation attacks!
  - Improve sustainment, operational/acquisition agility
    - Fix HDL design, tool-chain flaws via bitstream update!
      - Like firmware update, but at *even lower* level
  - Modern FPGA performance sufficient for e.g., embedded, or basic development platform

Recommendations

- Retain ability to *field strip* our cyber-weapons!
  - Require capability to rebuild system from *sources*
    - *Including* tool chain sources: HDL & software compilers!
  - Show of *good faith* from upstream supplier(s)
  - Built-in sustainment capability from *day one*
    - Solve “Trusting Trust” concerns
      - Available source code (to everything) acting as trust anchor

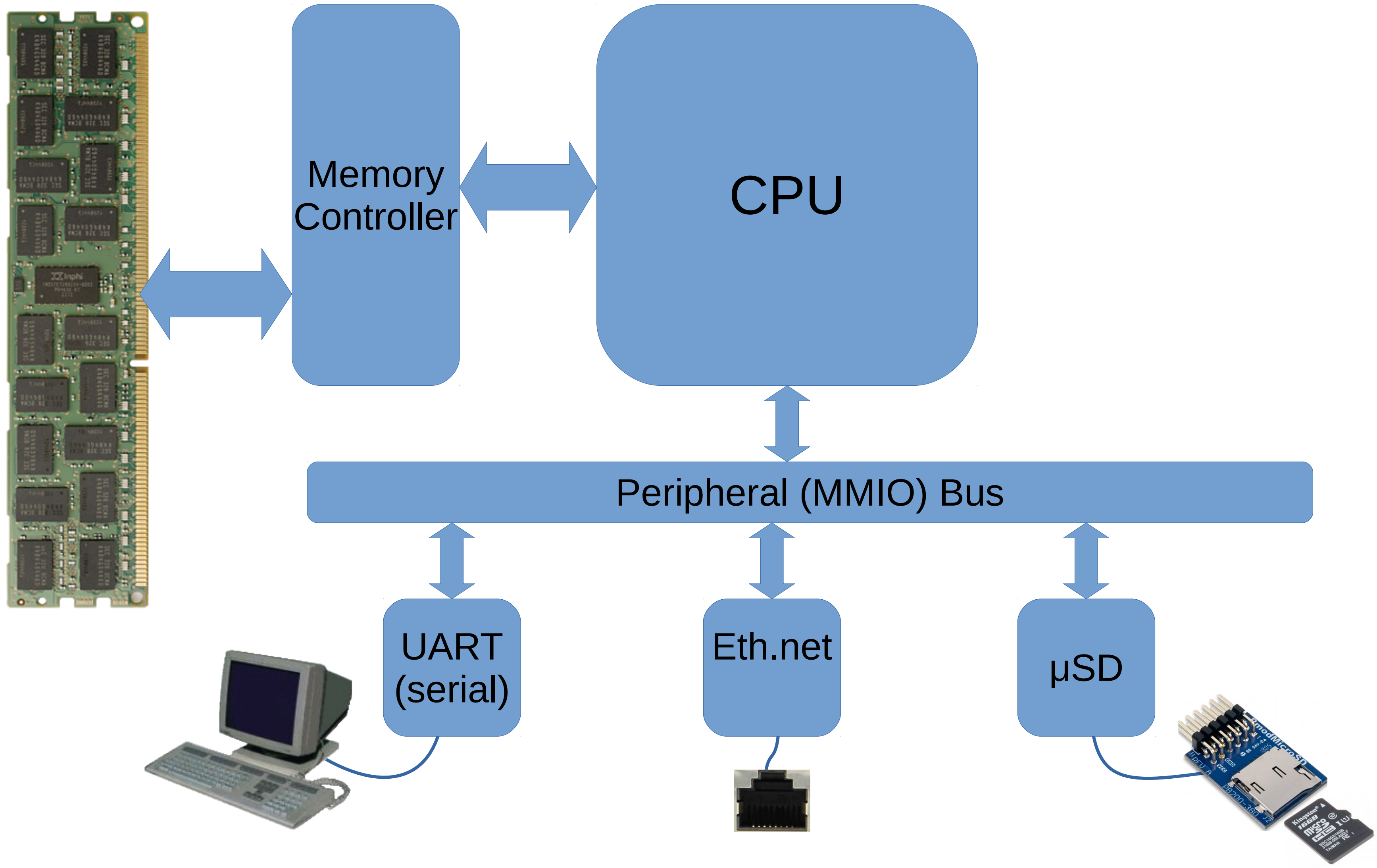
Bootstrapping a Trustworthy Platform

- Use DDC to obtain a clean C [cross-]compiler
- [Cross-]compile HDL compiler toolchain
- Cross-compile target OS (kernel, glibc, utilities)
- Build FPGA bitstream with HDL toolchain
- Boot target OS on FPGA
  - Self-hosting from this point forward
    - Any system component can be (re)built on the system itself!
  - Trust anchor: the cumulative set of source code
    - HDL, OS (kernel, glibc, utilities), and Compilers (C & HDL)

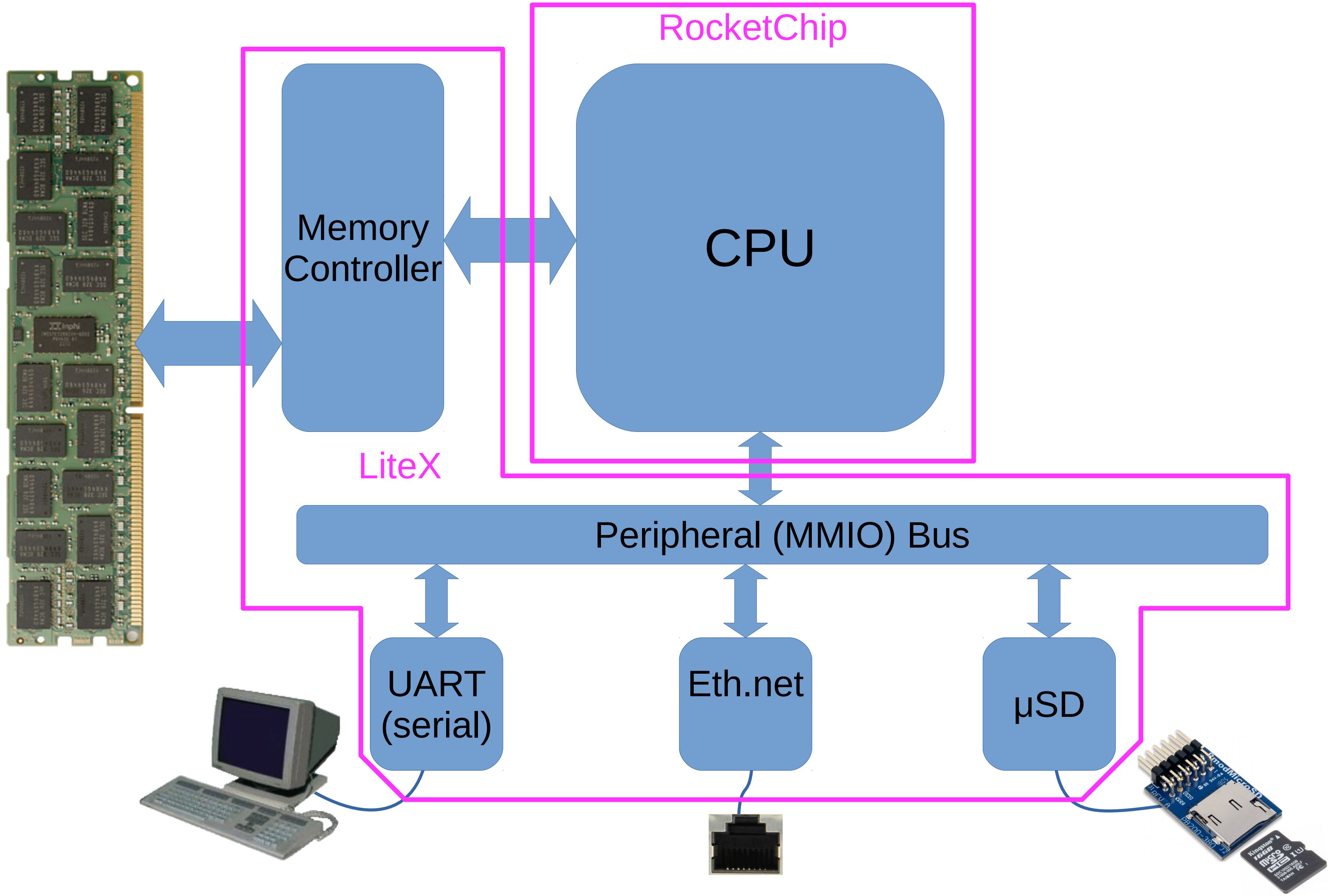
List of Ingredients

- FPGA development board
  - Lattice ECP5 Versa: LFE5UM5G-45F-VERSA
- Free/Open HDL (Hardware Description Language) toolchain
  - Verilog front-end: https://github.com/YosysHQ/yosys
  - ECP5 device db. & bitstream tools: https://github.com/SymbiFlow/prjtrellis
  - Place & Route back-end: https://github.com/YosysHQ/nextpnr
- Free/Open 64-bit CPU (RISC-V ISA)
  - RocketChip: https://github.com/freechipsproject/rocket-chip
- Free/Open System-on-Chip (SoC) environment (sys. bus & peripherals)
  - LiteX: https://github.com/enjoy-digital/litex
- Software stack (Linux, GCC, glibc)
  - Fedora: https://fedoraproject.org/wiki/Architectures/RISC-V

Simplified Computer Architecture



Simplified Computer Architecture



Project Status

- LiteX builds & runs on ECP5 FPGA with Free/Open toolchain
  - Boots Linux on 32-bit (VexRiscv) CPU
- LiteX + 64-bit RocketChip builds and runs in simulation, expected to run on FPGA within weeks
- Next: boot Fedora (only available in 64bit flavor)
  - Port yosys/trellis/nextpnr packages from x86 to rv64
  - Build LiteX + 64-bit Rocket ***ON*** LiteX + 64-bit Rocket!

Next Steps

- Need FPGA development board w. more RAM!

- Collaborate with Formal Verification experts on measuring trustworthiness of comprehensive source bundle (now provably equivalent to deployed FPGA SoC embedded system)

Extra Slides

Software vs. Hardware Programming

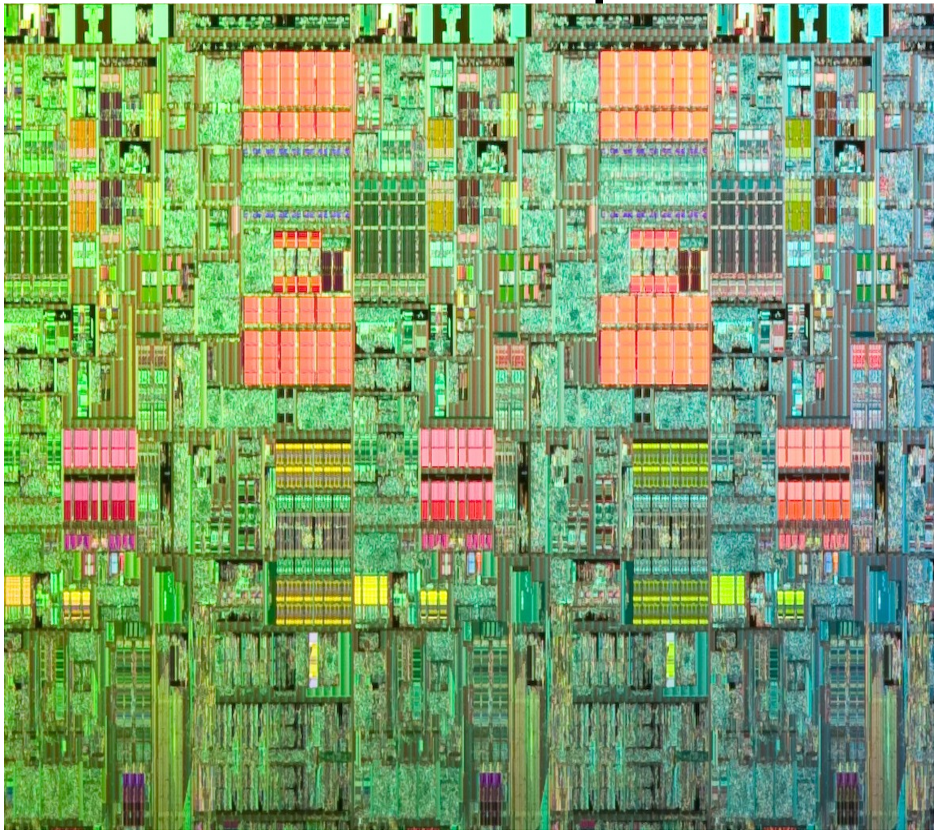
Microprocessor	FPGA
Architectural design	Architectural design
Choice of language (C, JAVA)	Choice of language (Verilog, VHDL)
Editing programs	Editing programs
Compiling programs (.DLL, .OBJ)	Compiling programs
	Synthesizing programs (.EDIF)
Linking programs (.EXE)	Placing and routing programs (.VO, .SDF, .TTF)
Loading programs to RAM	Loading programs to FPGA
Debugging P programs	Debugging FPGA programs
Documenting programs	Documenting programs
Delivering programs	Delivering programs

Image credit: Ed Klingman, "FPGA programming step by step", 2004

Malicious Foundry Attack Examples

- A2 Trojan
  - 20 transistors + 1 capacitor
  - Incremental charge via unpriv. instr. sequence
  - Flip bit reg. – privilege flag
- Dopant-level Trojan
  - Swap PNP ↔ NPN polarity on selected transistors
  - Visually undetectable
  - Predictably weaken RNG randomness

> 1.5 bn components!



Confusion re. “Open Source”

- Open Source Intelligence (OSINT)
  - Used in Military, Intelligence, Government, Law Enforcement communities
  - Data collected from public sources (vs. trusted, classified, access-controled sources)
  - Often associated with *decreased reliability*!
- Free or Open Source Software (F/OSS)
  - Software development principles & methodology
  - High quality: Linux, BSD, Apache, Firefox, etc.
  - Serious participants: RedHat, IBM, Google, etc.
    - Competitors collaborate to avoid “reinventing wheel”

Glossary

- **CPU**, **ISA**: Central Processing Unit; implements a specified Instruction Set Architecture (e.g. x86, ARM, PowerPC).
- **GPU**, **SIMD**: Graphics (rather, vector) Processing Unit; a processor operating on vector data, running a Single Instruction on Multiple Data simultaneously; originally targeted at graphics acceleration, useful in high performance computing.
- **ASIC**: Application Specific Integrated Circuit; dedicated etched silicon implementing a specified microelectronic design.
- **Hard IP Core**: well-delimited functional unit of an ASIC, based on Intellectual Property provided by a specific vendor.
- **FPGA**, **CLB**: Field Programmable Gate Array; in itself a special-purpose ASIC, with the application or purpose of dynamically and reconfigurably implementing a given microelectronic design. An FPGA consists of a grid of identical Configurable Logic Blocks that can communicate with each other through a programmable interconnect.
- **Bitstream**: stream of bits populating memory cells that control the CLBs and programmable interconnect on an FPGA, determining the exact nature of the design to be implemented.
- **Soft IP Core**: well-delimited functional unit of a microelectronic design, based on Intellectual Property from a specific vendor, incorporated into a design laid out on top of an FPGA using Bitstream.
- **SoC**: System-On-a-Chip; instead of soldering multiple, frequently-used-together ASICS and/or FPGAs together on a Printed Circuit Board (PCB), they are connected together on the same set of masks, and etched onto the same silicon die. This saves space, reduces power, and improves reliability.