Introduction to FPGAs using Homebrew Automation

Brandon Blodget, Patrick Lloyd, & Bob Smith

Who are We?

- Tinkerers with a bunch of software, hardware, firmware, and gateware experience (mostly Bob and Brandon, though 😉)
- Engineers at OLogic, Inc.
- ClusterFighters!

What is an FPGA?

> To control a processor, you program it with instructions and tell it what to **do**...
>
> to control an FPGA, you describe a circuit and tell it what to **become**.

- Dalai Lama, probably

What is an FPGA?

- Short for **F**ield-**P**rogrammable **G**ate **A**rray
- One type of **Programmable Logic Device (PLD)**
    - Implements arbitrary digital logic equations (i.e. Boolean operations)
    - Most are re-programmable
    - Some are non-volatile, while others require external memory to save configuration

What is an FPGA?

- Other PLD's include:
  - PLA - Programmable Logic Array
  - PAL - Programmable Array Logic
  - GAL - Generic Array Logic
  - SPLD - Simple Programmable Logic Device
  - CPLD - Complex Programmable Logic Device

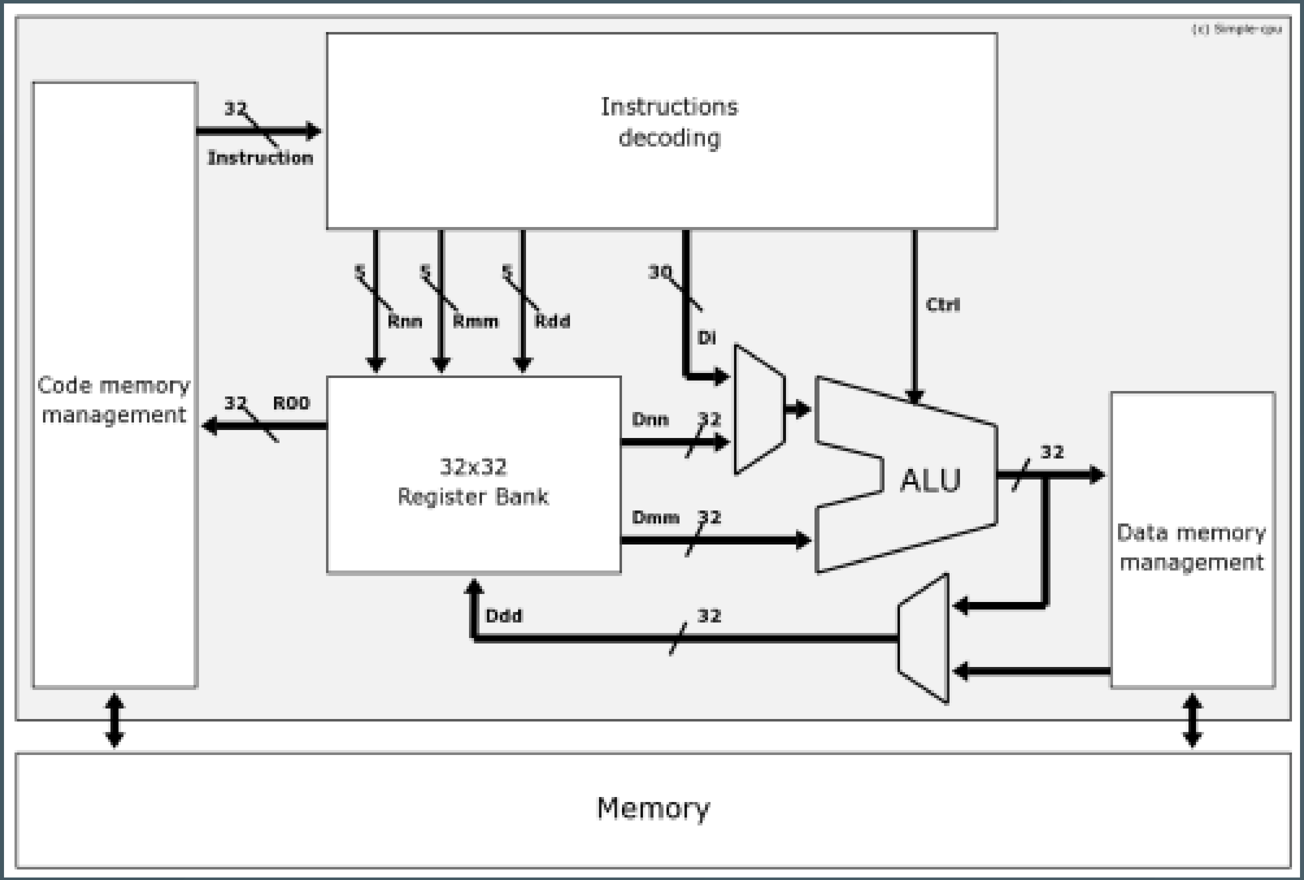
(Who names these things?)

Players

FPGA vendors by market share:

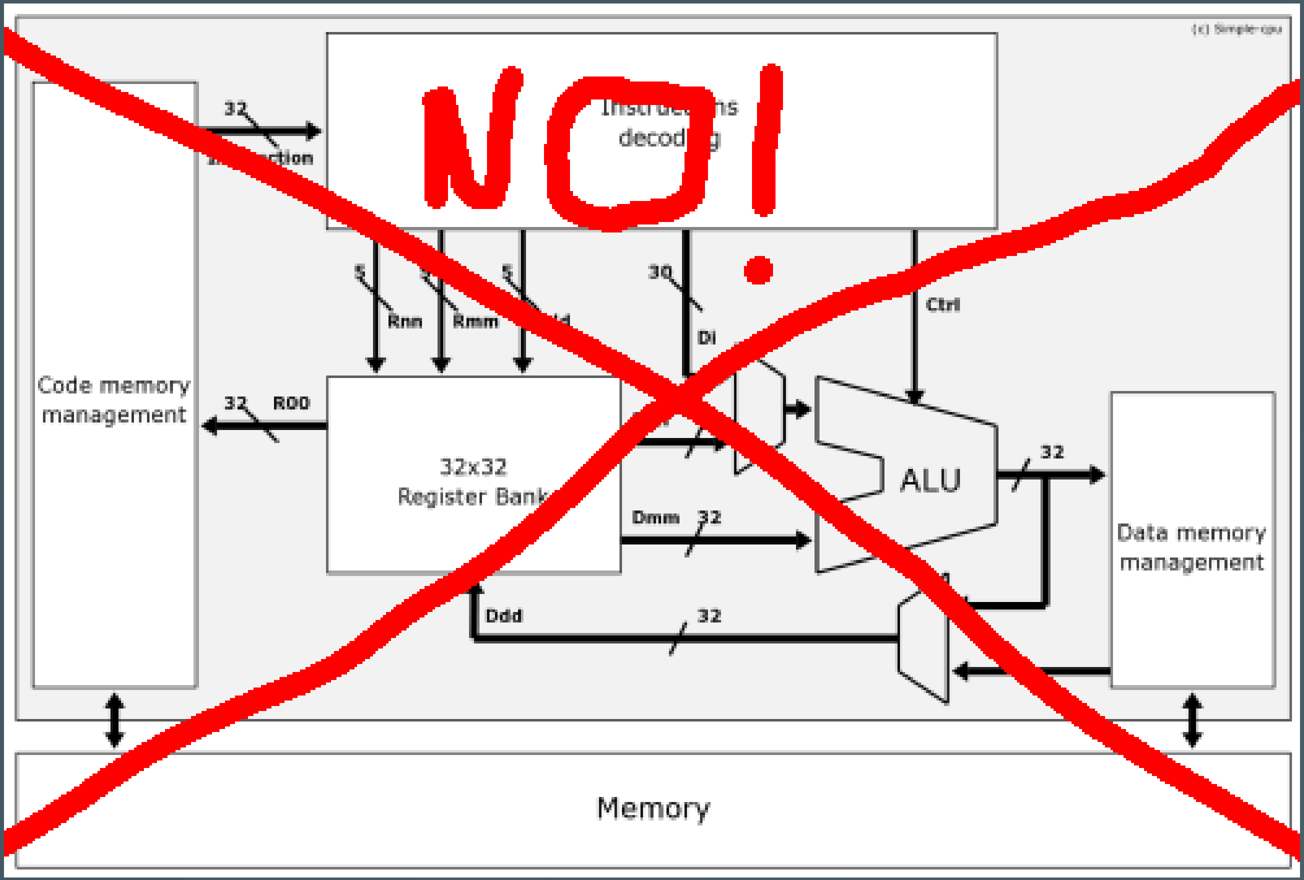
- **Xilinx** - ~50%
- **Intel** (Altera) - ~40%
- **Lattice** - >5%
- **Microchip** (Microsemi (Actel)) - <5%
- Everyone else (QuickLogic, Gowin, etc.) - ~1%

What's *Inside* an FPGA?



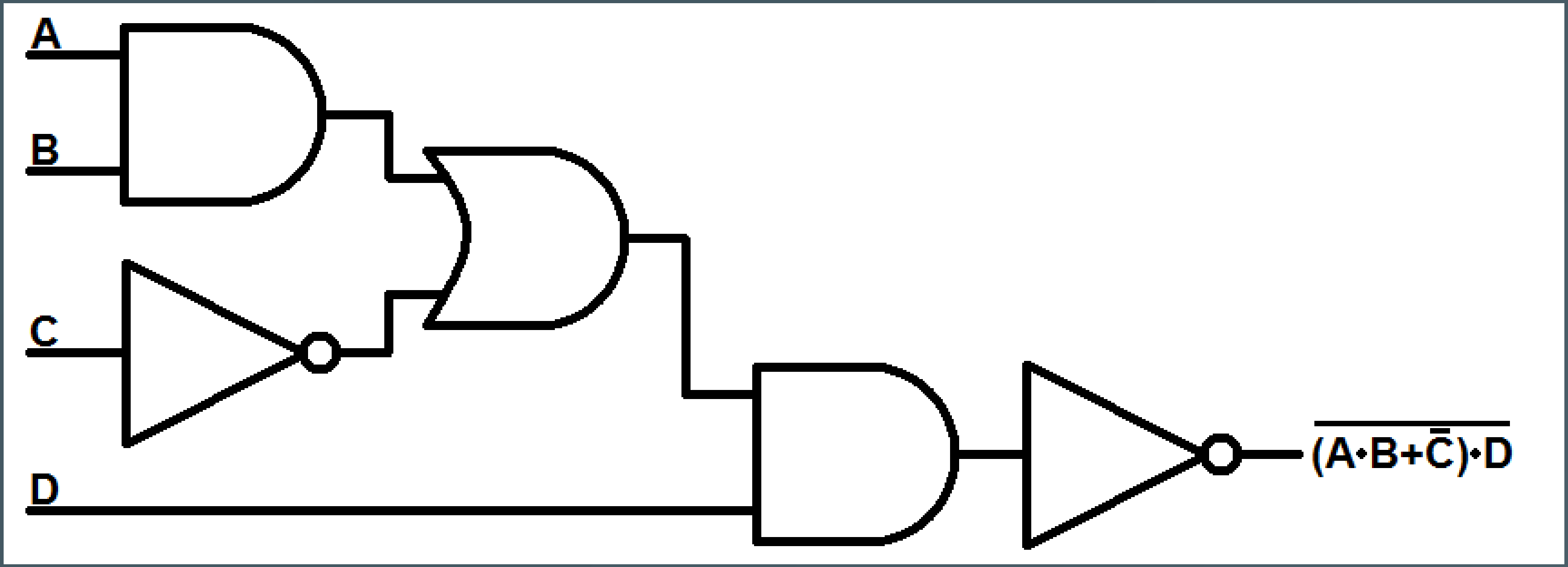
- Is it a special type processor?

What's *Inside* an FPGA?



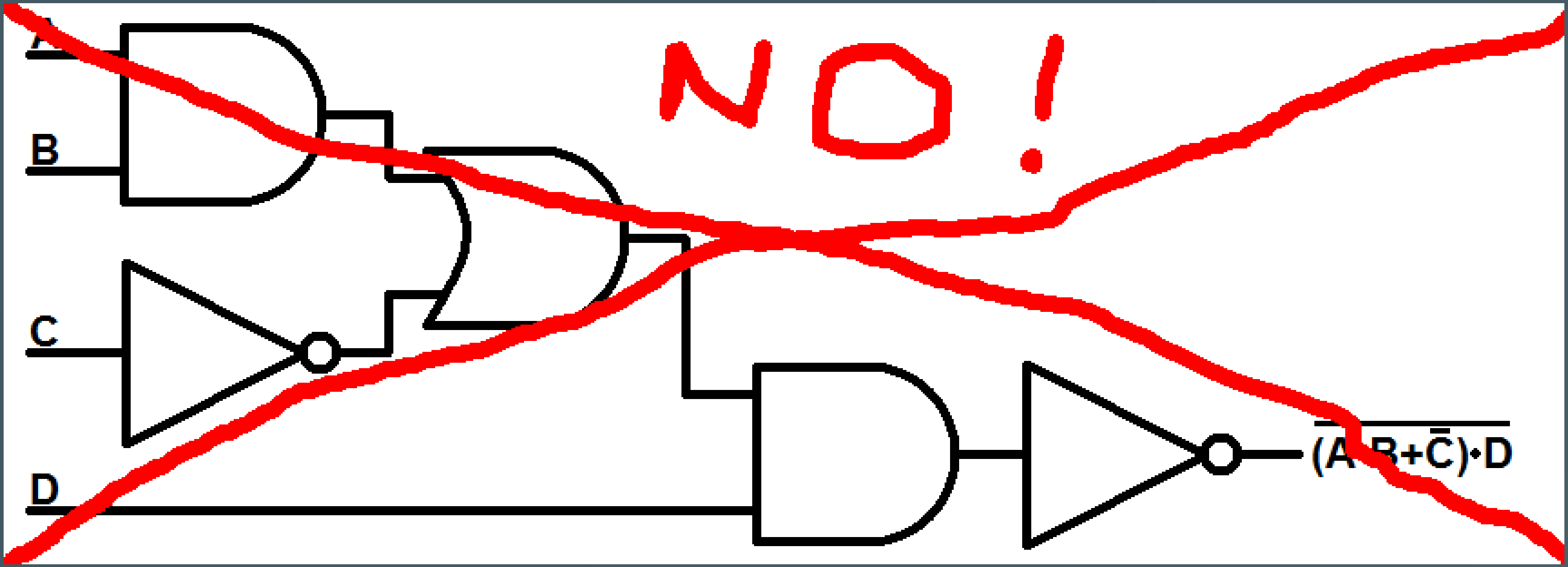
- But an FPGA can be used to *implement* CPUs (known as "soft cores")

What's *Inside* an FPGA?



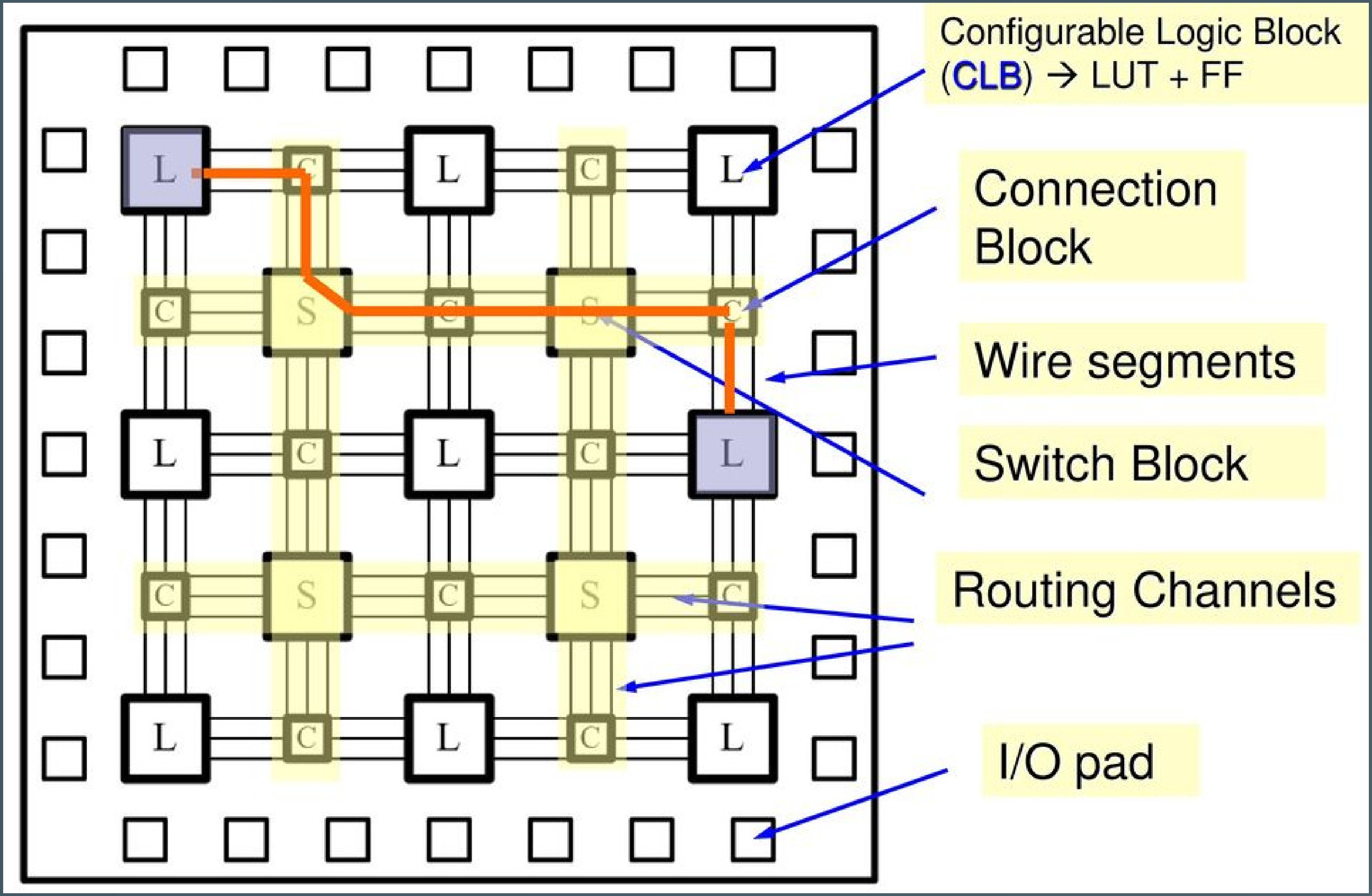
- Is it a big collection of AND / OR / NOT / NOR / NAND / NOR gates? Perhaps even a *gate array* of sorts?

What's *Inside* an FPGA?

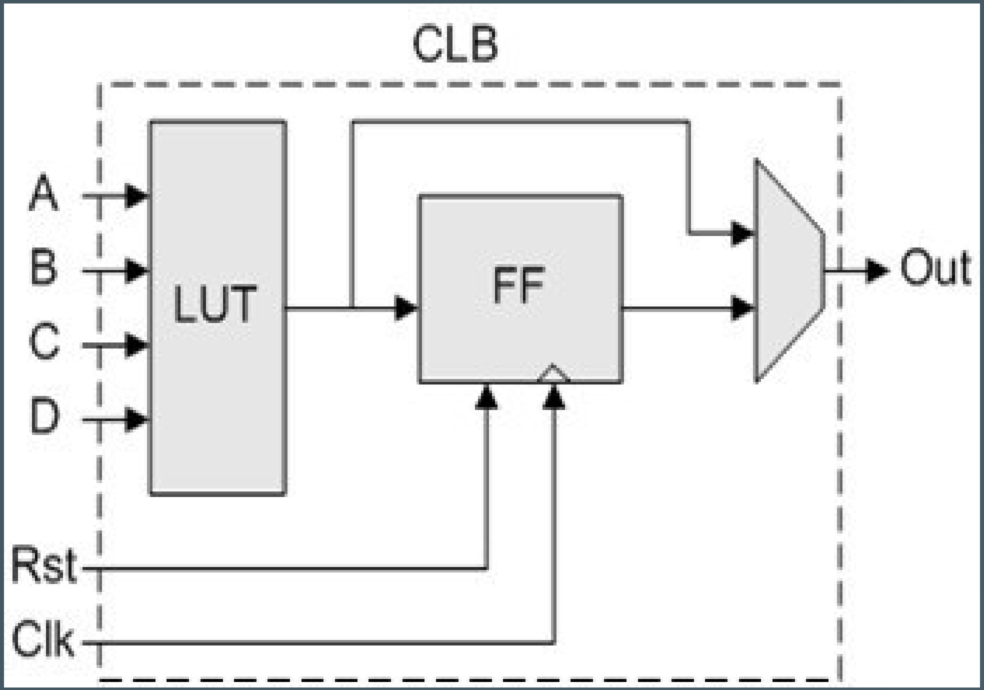


- But an FPGA can be used to *implement* arbitrary logic equations

What's *Inside* an FPGA?



Configurable Logic Blocks



- Lookup table (LUT)
- D-Type Flip-Flop (D-FF)
- Multiplexer (MUX)

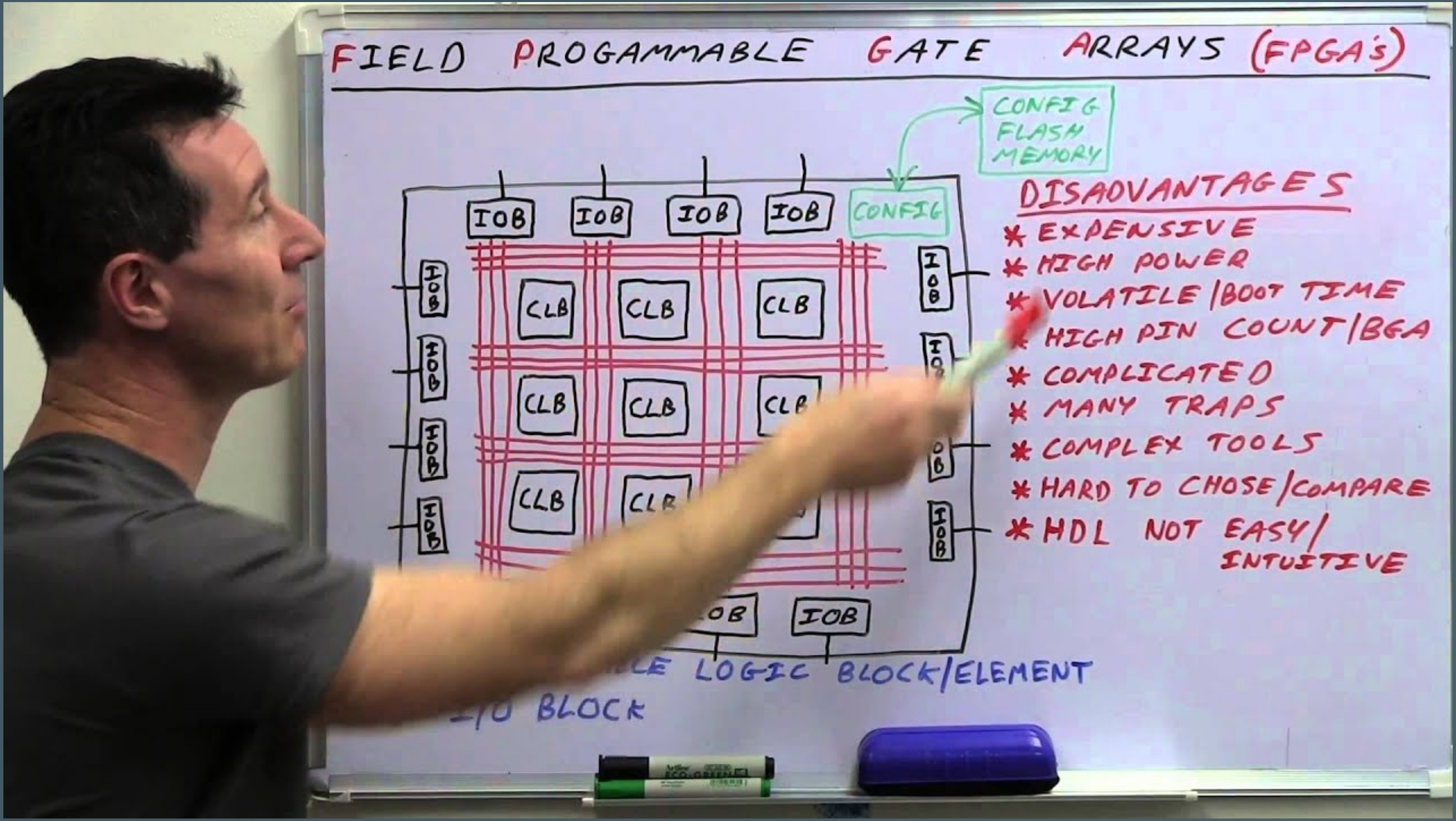
Open-Source Applications

- ASIC prototyping [RISC-V]
- Chip emulation [MiSTer]
- DSP - filters, transforms, convolution, decimation, digital synthesis, etc.
    - Software-defined radio [1] [2] [3]
    - Audio Synthesizers [1] [2]

Open-Source Applications

- Accelerator cores
  - Cryptography - [MD5] [AES]
  - Video encode / decode [H.264]
  - Very high speed networking [Intel 100G NIC]
- In general, FPGAs are great at applications that can leverage fixed-point math, high memory bandwidth, pipelining, & parallelism

It can't be all good, right?



Disadvantages

FPGAs are ill-suited for certain tasks:

- Floating point operations
- Protocol handling
- Complex rulesets
- P-complete problems that are sequential & not easily parallelized (think "Conway's Game of Life")
- Developing solutions quickly (maybe)

The Case for Robotics

- Access to *tons* of flexible, reconfigurable I/O pins
- Timers & counters are trivial to impement
    - PWM for motor control (brushed, BLDC, servos)
    - Pulse decoding (encoders, IR remotes, RC controllers, SONAR)
- Swappable, on-demand peripherals like UART, SPI, I2C, 1-wire, etc.

The Case for Robotics

- PID controllers
    - Very fast execution and consistent timing
    - Fixed point
- Finite state machines
    - Line following
    - Obstacle avoidance

FPGA Workflow

- Design entry
- Simulation
- Synthesis
- Technology mapping
- Placement
- Routing
- Bitstream generation
- Flashing device

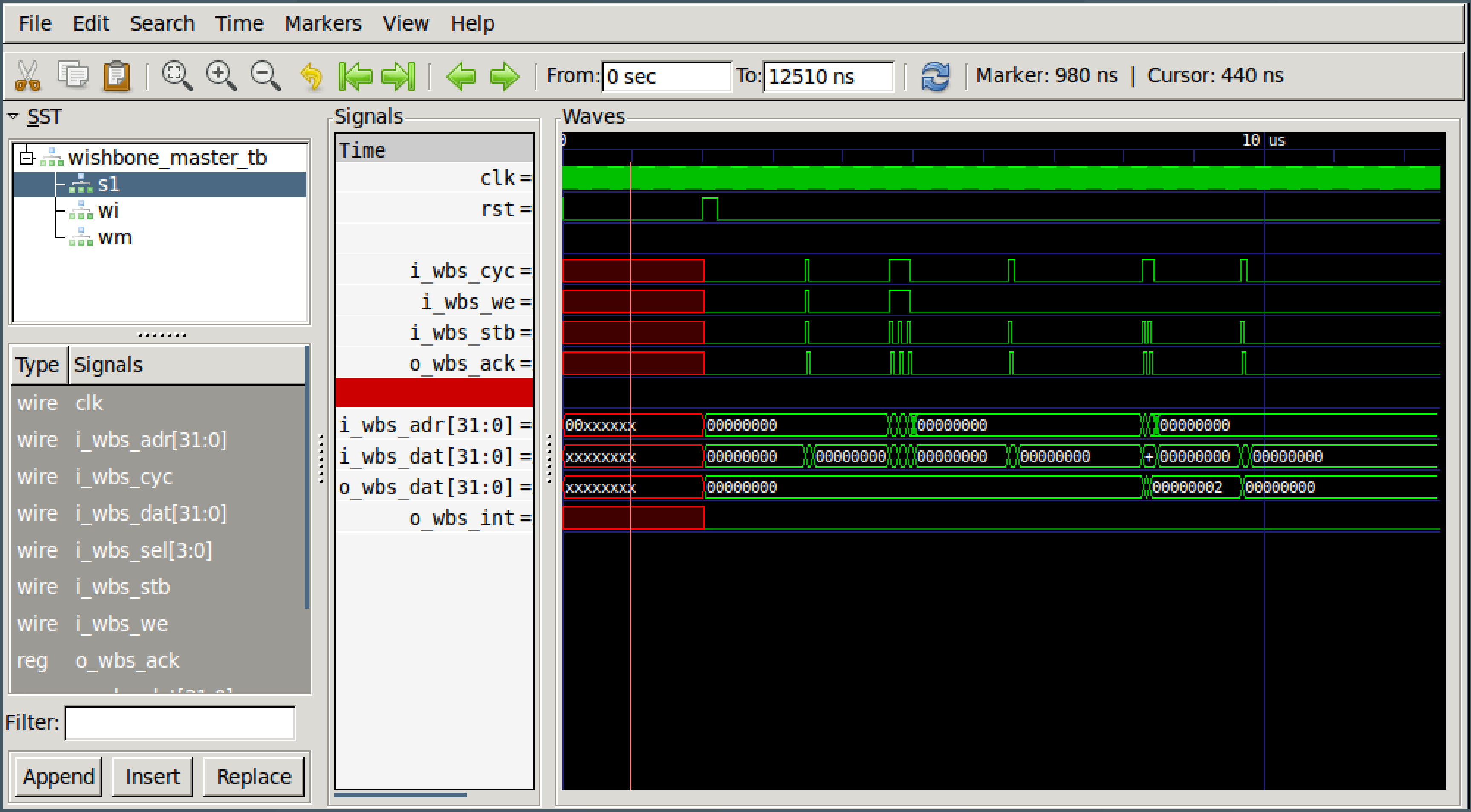
Design Entry

- FPGA internals are *described* using a hardware description language (HDL)
    - **Verilog** (C-like, weakly-typed)
        - Popular in open source, consumer electronics, & the west coast
    - **VHDL** (Ada-like, strongly typed)
        - Popular in defense / aerospace, academia, & the east coast
- Tools: Your favorite text editor

Simulation

- Allows designs to be verified as individual blocks or as a full system
- Only simulates *functionality* of a design, not the physics. This means a design can work in simulation but fail when trying to work with real hardware
- Tools:
  - Simulator - Icarus Verilog, Verilator
  - Waveform Viewer - GTKWave

Simulation



Type	Signals
wire	clk
wire	i_wbs_adr[31:0]
wire	i_wbs_cyc
wire	i_wbs_dat[31:0]
wire	i_wbs_sel[3:0]
wire	i_wbs_stb
wire	i_wbs_we
reg	o_wbs_ack

Synthesis

- Convert Verilog into generic logic circuits. Also known as behavioral to RTL conversion.
- Some logic optimization happens here
- Output saved as an intermediate file format not really intended for human interaction
- Tools: Yosys
    - `synth` command provides good set of defaults that can be used as basis for synthesis scripts
        - `yosys> read_verilog mydesign.v # import design`
        - `yosys> synth -top mytop # default synthesis`

Technology Mapping

- Synthesis output needs to be mapped to the specific hardware architecture of the FPGA (i.e. CLBs, DSP, SERDES, etc.)
- Tools: Yosys
    - Multiple scripts are used to map the design
        - `yosys> dfflibmap # map flip-flops`
        - `yosys> abc # map logic`
    - Supports **extensible, custom techmaps!**
    - Returns text file to be consumed by P&R tool

Placement & Routing

- Tool consumes techmapped design, tries to fit it into the FPGA, and connect everything together
- NP-hard optimization problem (think "travelling FPGA salesman")
- Timing constraints factor in at this step
- Tools:
  - Nextpnr - Actively developed, still buggy, GUI
  - Arachne-pnr - No more development, works for ICE40 only (but reliably)

Placement & Routing

Bitmap Generation

- The P&R tool generates a binary file known as the **bitstream**
- Morally equivalent to an ELF or EXE
- Each bit controls the configuration state and initial conditions of every CLB and interconnect in the device
- Previously very secret sauce
- Slowy being reverse engineered by hackers *fuzzing* the vendor tools

Device Flashing

- Chip-specific (SPI, JTAG, or some custom protocol)
- SRAM (fast, volatile) vs. external flash (slow, non-volatile)
- Tools:
  - `iceprog` - ICE40 dev boards only
  - `openocd` - Popular tool to program & debug with
  - `flashrom` - External SPI/I2C flash

Holy cow we're done!

Just kidding! Now we can finally start.

How to Write Less Verilog

- Code reuse has historically been challenging for FPGA designs
- How to abstract away complexity:
  - High-level synthesis tools
    - **MATLAB** HDL Coder, VivadoHLS (**C/C++**), Intel HLS Compiler (**C**), Synphony (**C**), Migen (**Python**), Litex (**Python**)
  - Configurable IP Megablocks
    - Vendor clock & PLL configuration
    - Schematic representation

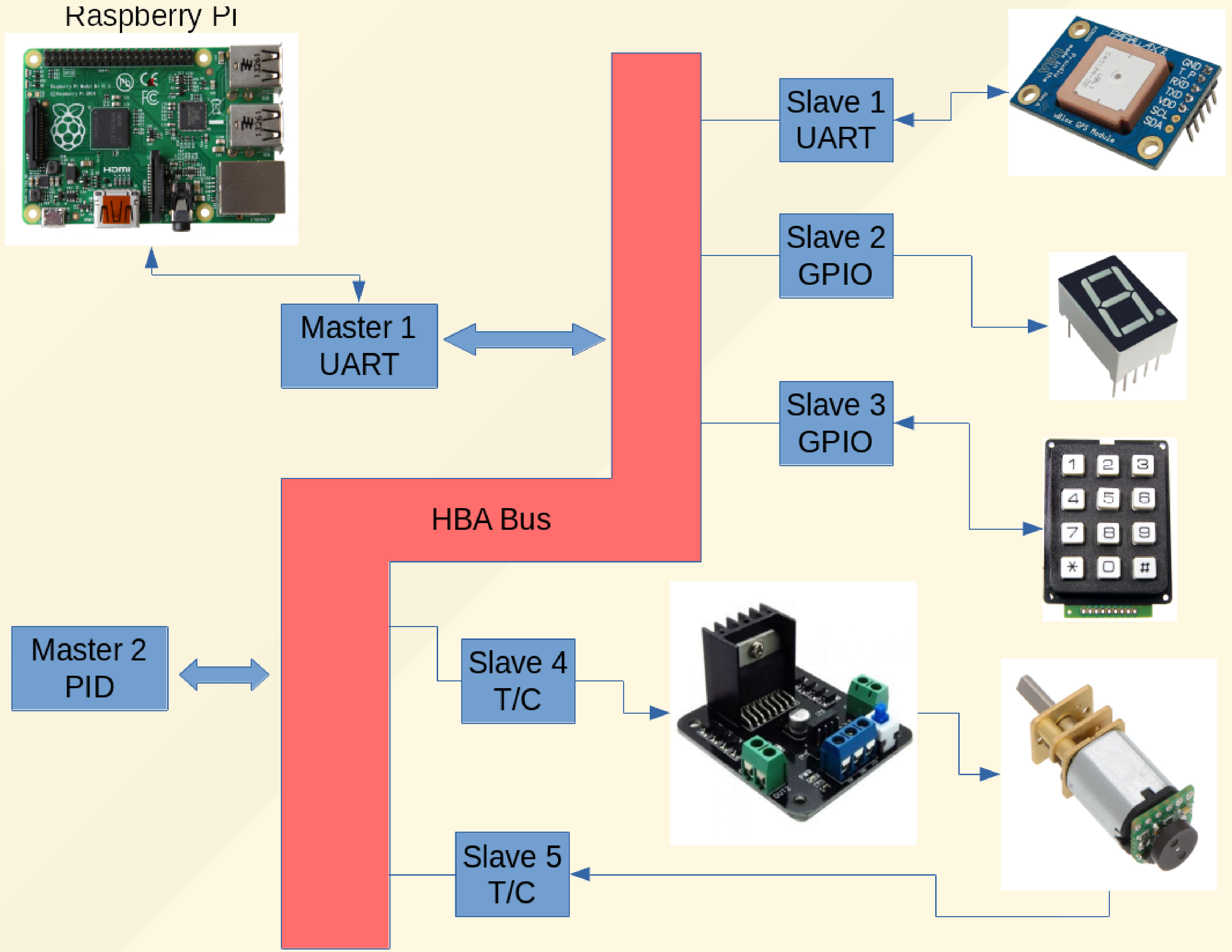
How to Write Less Verilog

- Connect the FPGA to a Linux computer & leverage the strengths of both devices

Homebrew Automation

HBA Gateware (FPGA)

- Small, modular peripherals that can be reused
- Standardized interface to a simple bus
- Up to four master peripherals
  - E.g. Raspberry Pi interface or PID controller
- Up to sixteen slave peripherals
  - e.g. timer/counter, SPI, UART


Raspberry Pi
Master 1
UART
Master 2
PID
HBA Bus
Slave 1
UART
Slave 2
GPIO
Slave 3
GPIO
Slave 4
T/C
Slave 5
T/C

HBA Software (Linux)

- UNIX-like interface design - everything is ASCII and can be piped ('|') into other tools
- Abstracts all the FPGA complexity into command and data registers for each peripheral (think I2C)
- Three basic commands:
  - `hba_get [PERIPHERAL] [REGISTER]` - read
  - `hba_set [PERIPHERAL] [REGISTER]` - write
  - `hba_cat [PERIPHERAL] [REGISTER]` - open stream

HBA Hardware

- Romi and a custom board
- Show pics of stuff

Class Details

- Time
- Place
- Cost

Questions?