https://github.com/hbrc-fpga-class/class-material/blob/master/presentation.pdf

Introduction to FPGAs using Homebrew Automation

Brandon Blodget, Patrick Lloyd, & Bob Smith

About This Talk

- Lots of information - $\frac{\Delta slides}{\Delta time} \propto log(time)$
- Please save questions for the end

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)**
  - Used to implement arbitrary digital logic equations (i.e. Boolean logic)
    - Combinational logic (gates, stateless)
    - Sequential logic (flip-flops, stateful)
  - 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

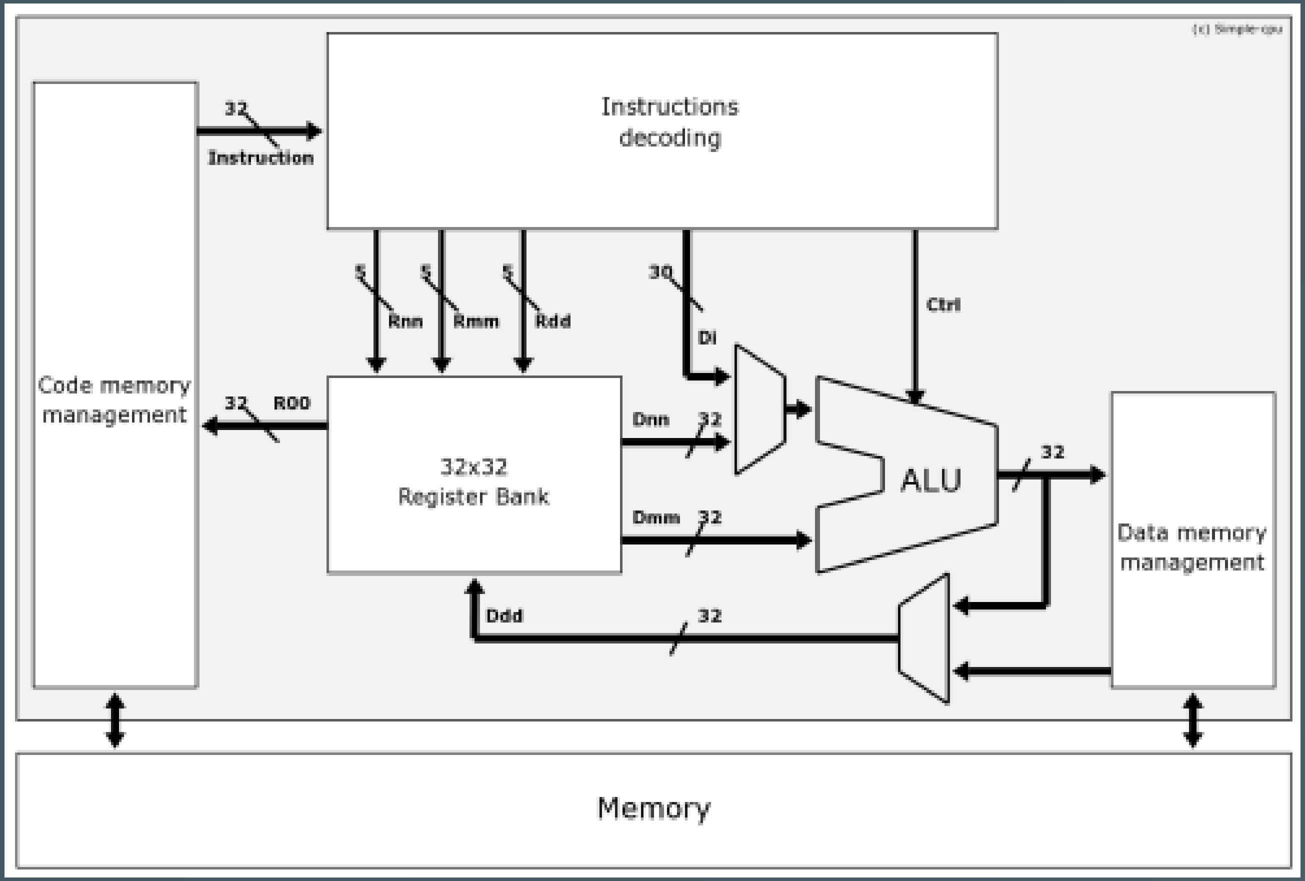
(Who names these things?)

Players

FPGA vendors by market share:

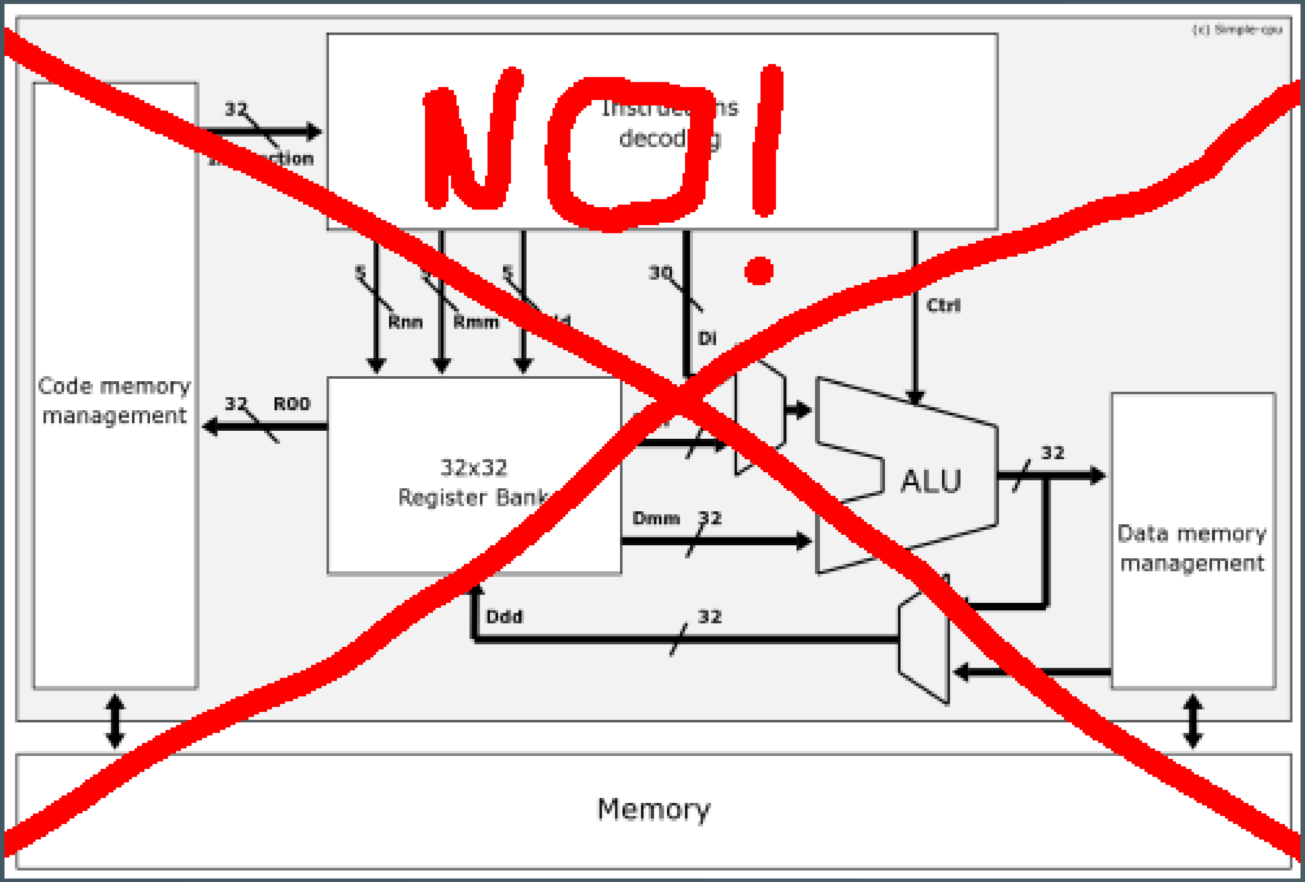
- **Xilinx** - ~50%
- **Intel** (Altera) - ~40%
- **Lattice** - >5%
- **Microchip** (Microsemi (Actel)) - <5%
- Everyone else (QuickLogic, Gowin, etc.) - ~1%

What's *Inside* an FPGA?



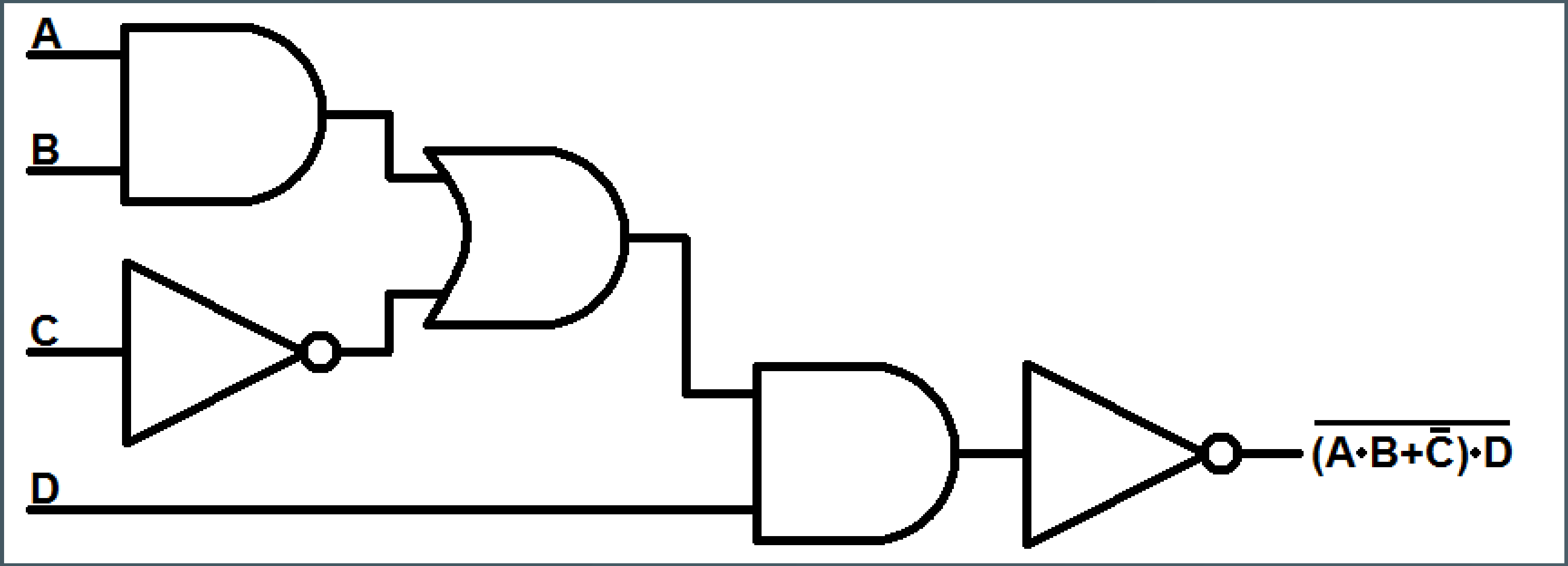
- Is it a special type processor?

What's *Inside* an FPGA?



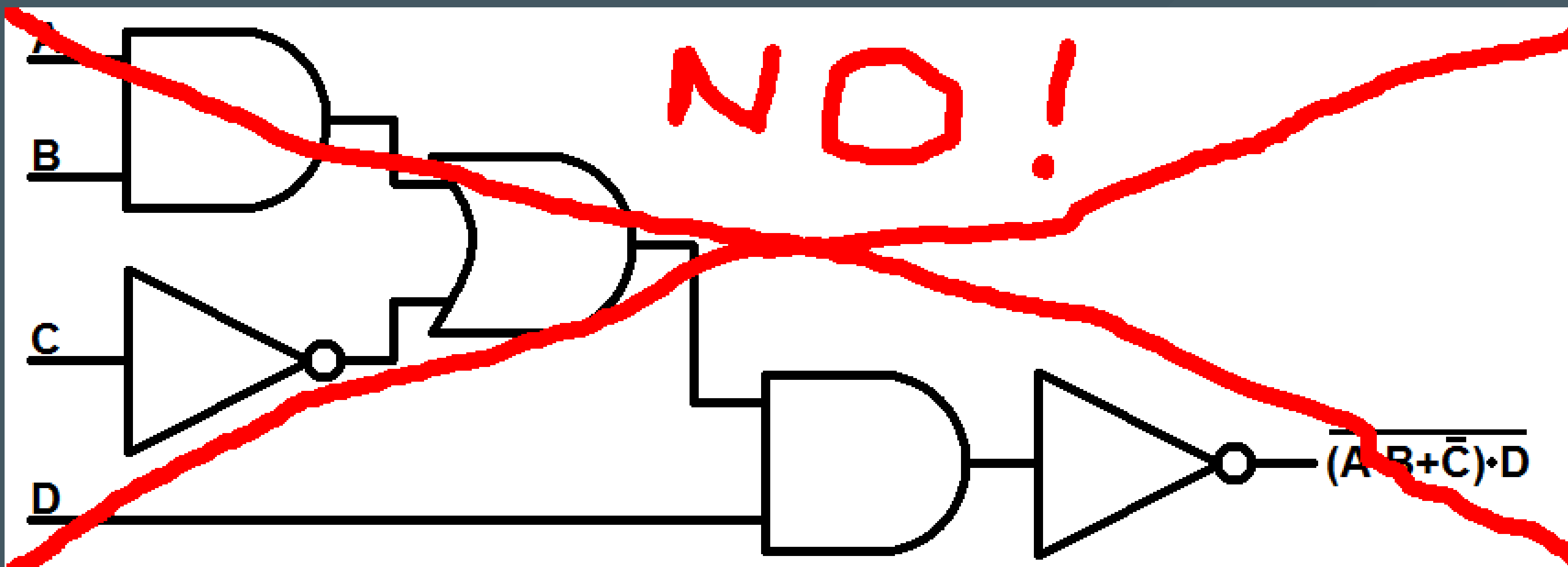
- But an FPGA can be used to *implement* CPUs (known as "soft cores")

What's *Inside* an FPGA?



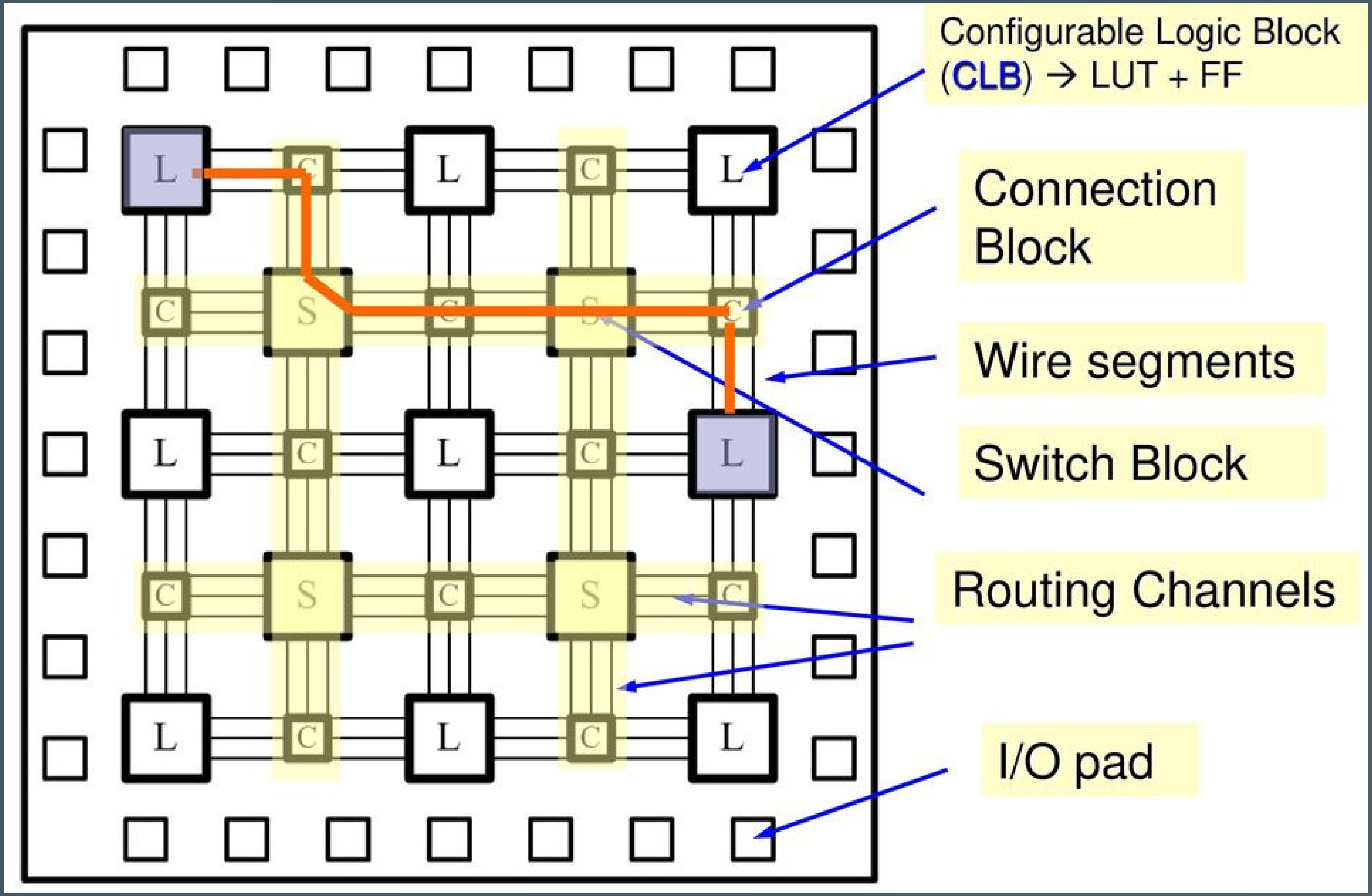
- Is it a big collection of AND / OR / NOT / NOR / NAND / NOR gates? Perhaps even a *gate array* of sorts?

What's *Inside* an FPGA?

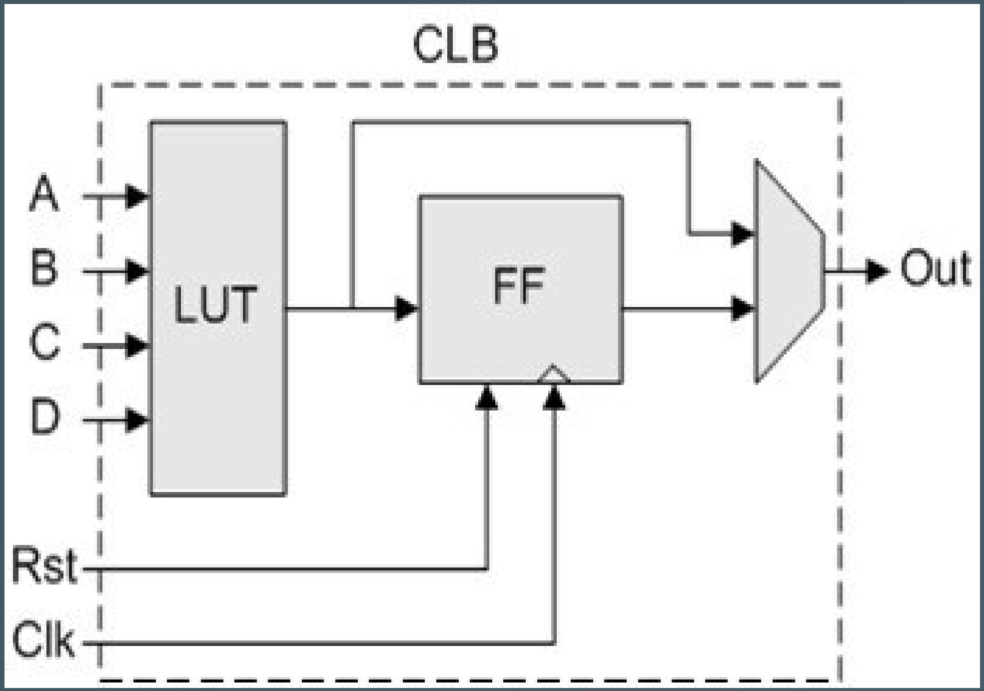


- But an FPGA can be used to *implement* arbitrary logic equations

What's *Inside* an FPGA?



Configurable Logic Blocks



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

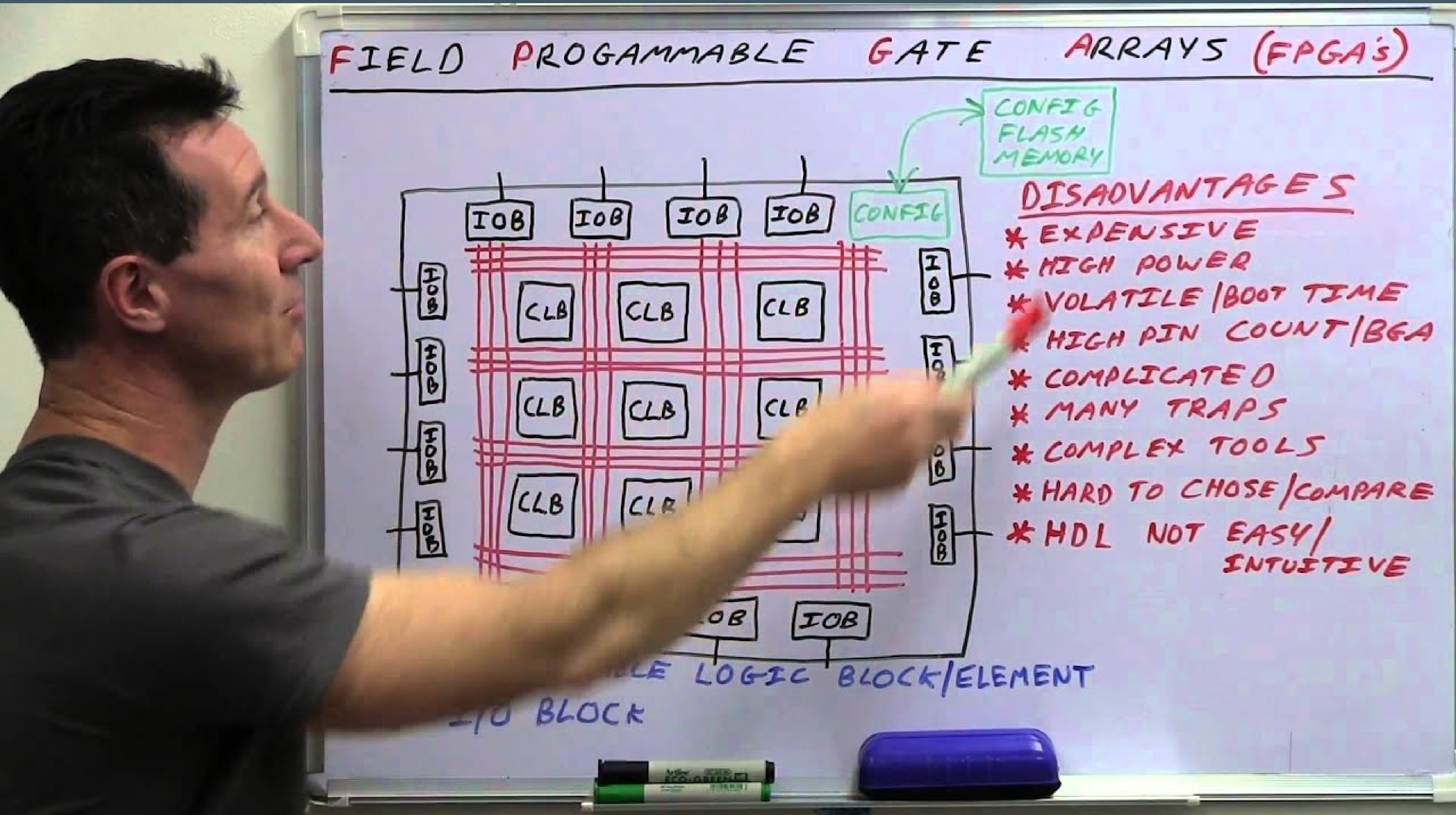
Open-Source Applications

- 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?


FIELD PROGAMMABLE GATE ARRAYS (FPGA's)
CONFIG FLASH MEMORY
CONFIG
IOB
CLB
DISADVANTAGES
* EXPENSIVE
* HIGH POWER
* VOLATILE / BOOT TIME
* HIGH PIN COUNT / BGA
* COMPLICATED
* MANY TRAPS
* COMPLEX TOOLS
* HARD TO CHOSE / COMPARE
* HDL NOT EASY / INTUITIVE
LOGIC BLOCK / ELEMENT
BLOCK

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

1. Design entry
2. Simulation
3. Synthesis
4. Technology mapping
5. Placement & Routing
6. Bitstream generation
7. Flashing device

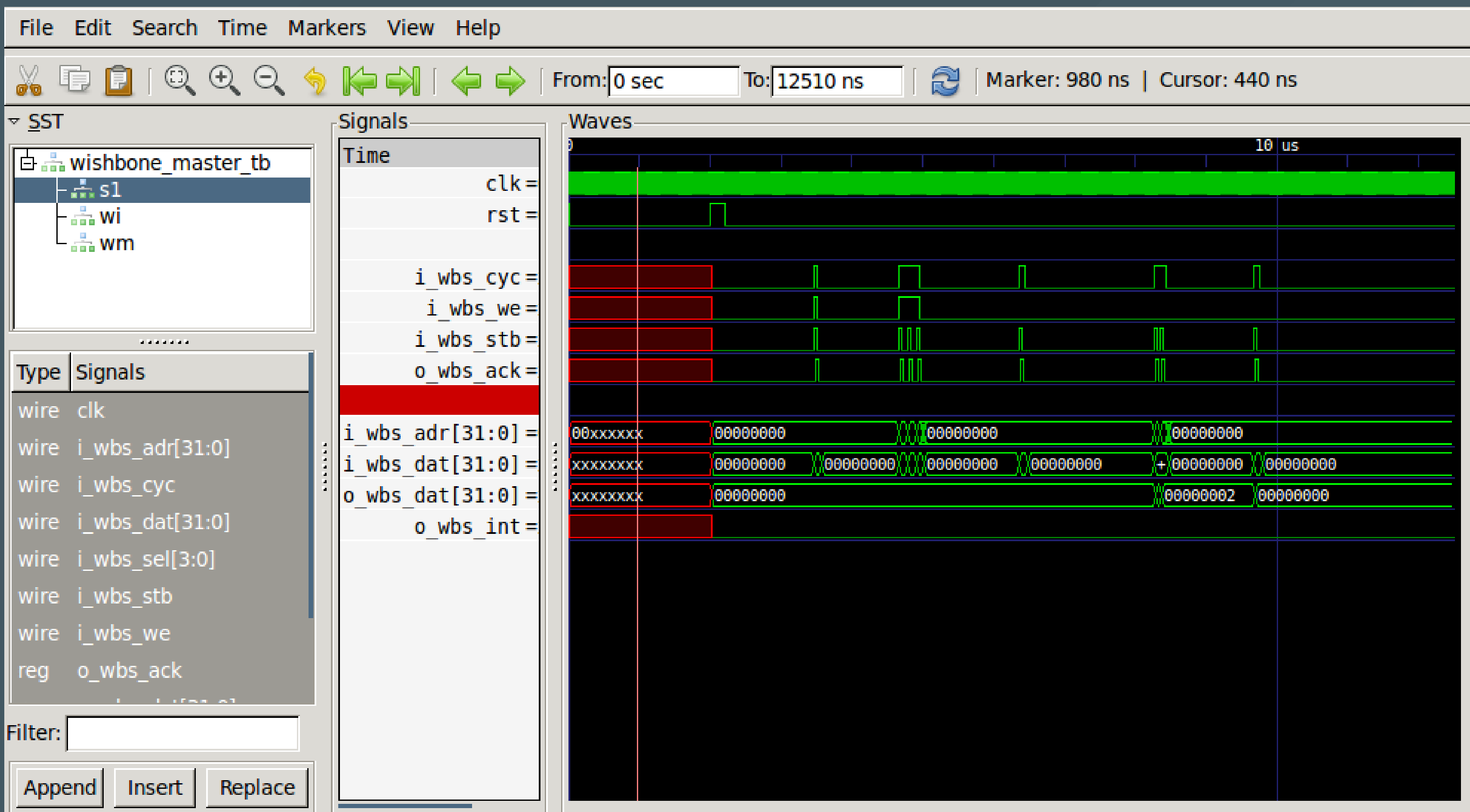
1. Design Entry

- 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

2. 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

2. Simulation



3. Synthesis

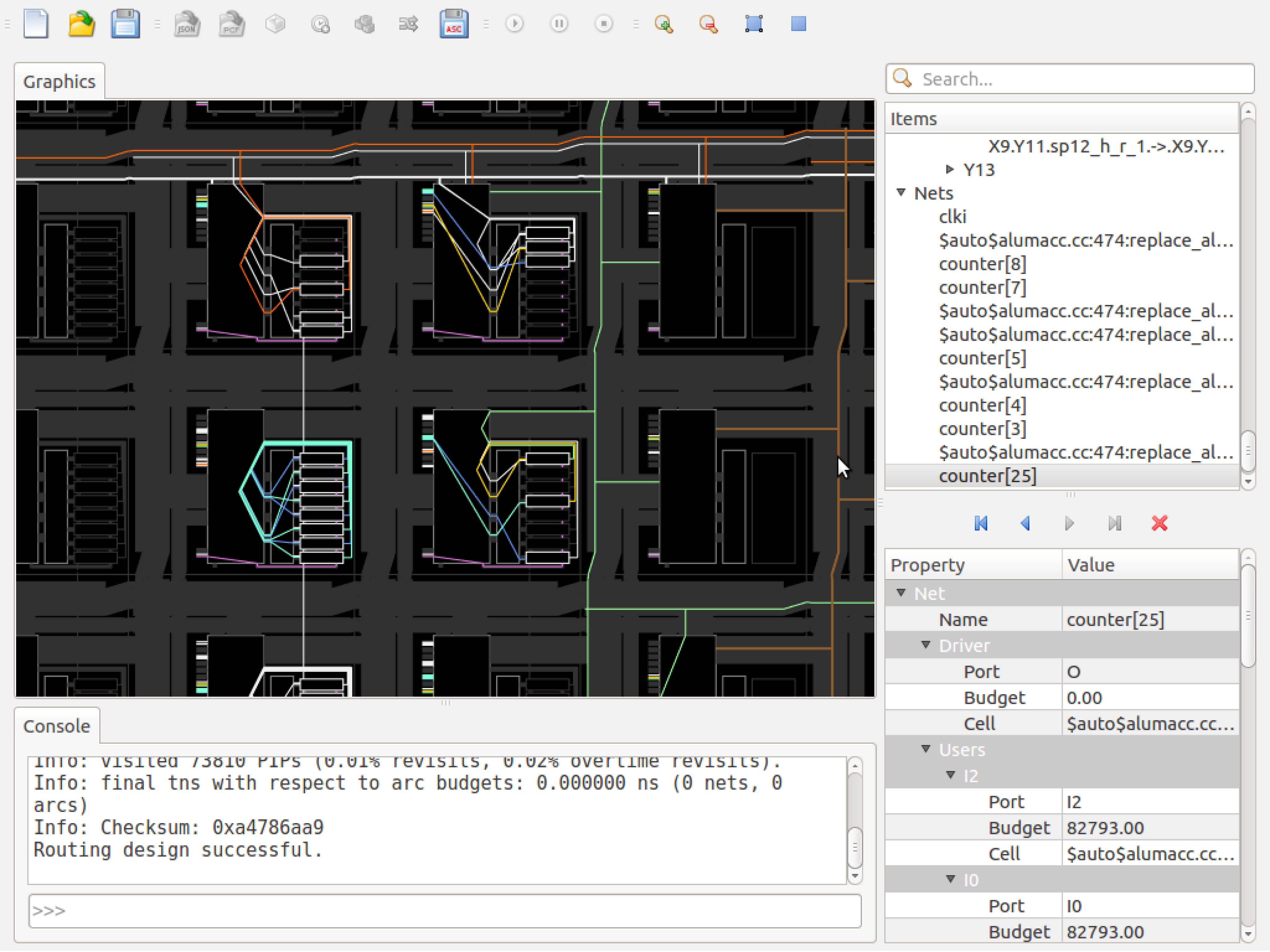
- Translates Verilog & RTL schematics into generic logic circuits.
- Some logic optimization happens here
- Output saved as an intermediate file format not really intended for human interaction
- Tools: [Yosys]
    - `synth` command provides good set of defaults that can be used as basis for synthesis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

4. 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

5. 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)


Graphics
Search...
Items
X9.Y11.sp12_h_r_1.->.X9.Y...
Y13
Nets
clki
$auto$alumacc.cc:474:replace_al...
counter[8]
counter[7]
$auto$alumacc.cc:474:replace_al...
$auto$alumacc.cc:474:replace_al...
counter[5]
$auto$alumacc.cc:474:replace_al...
counter[4]
counter[3]
$auto$alumacc.cc:474:replace_al...
counter[25]
Property
Value
Net
Name
counter[25]
Driver
Port
O
Budget
0.00
Cell
$auto$alumacc.cc...
Users
I2
Port
I2
Budget
82793.00
Cell
$auto$alumacc.cc...
I0
Port
I0
Budget
82793.00
Console
Info: final tns with respect to arc budgets: 0.000000 ns (0 nets, 0 arcs)
Info: Checksum: 0xa4786aa9
Routing design successful.
>>>

6. 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

7. 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

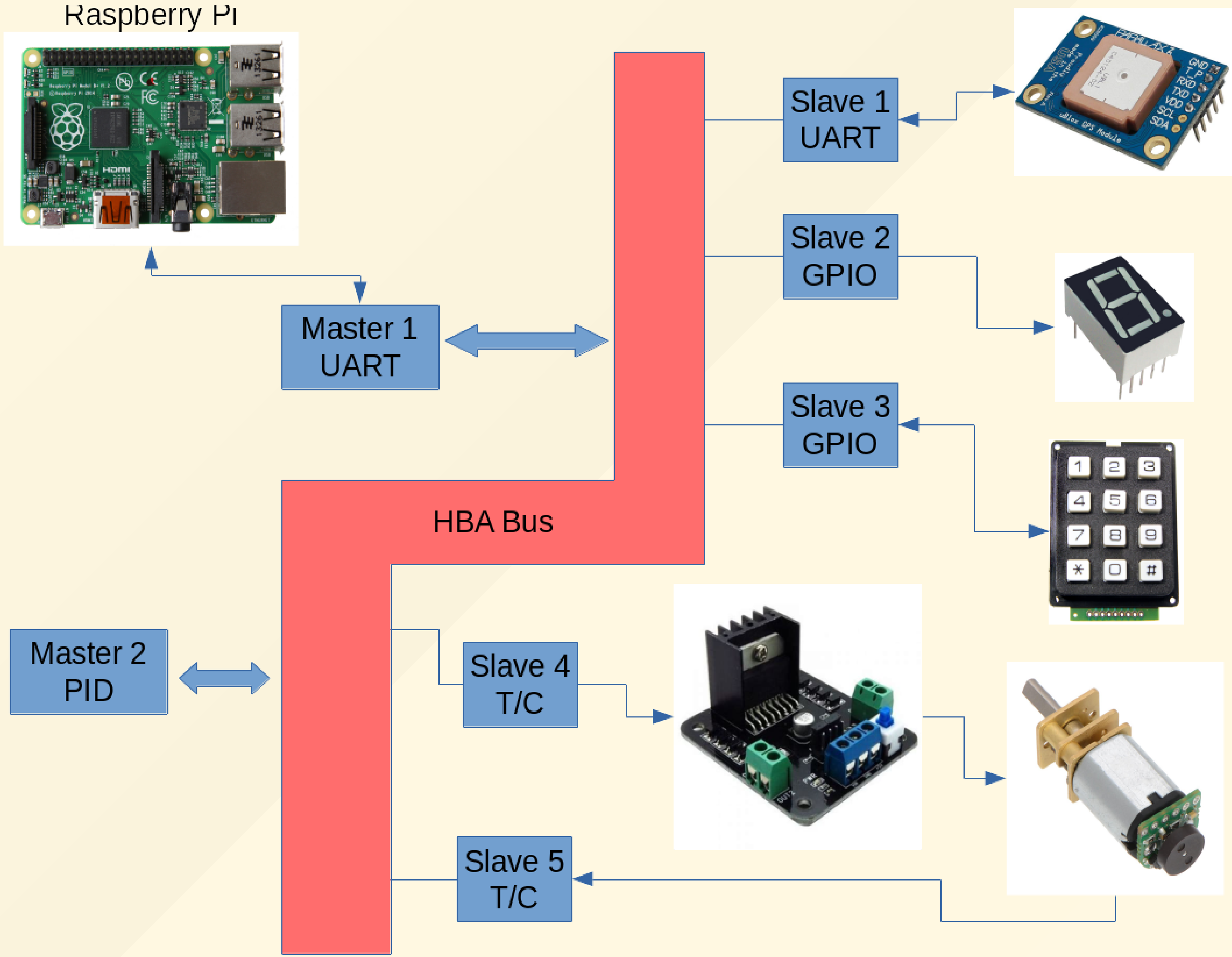
How to Write Less Verilog

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

Homebrew Automation

HBA Gateware (FPGA)

- Based on TinyFPGA BX (Lattice ICE40)
- Consists of small, modular peripherals
- Provides 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
HBA Bus
Slave 1
UART
Slave 2
GPIO
Slave 3
GPIO
Master 2
PID
Slave 4
T/C
Slave 5
T/C

HBA Software (Linux)

- UNIX-like interface design - everything is ASCII
- Interface presented as command and data resources for each peripheral (think I2C registers)
- Only a few basic basic commands:
    - `hbaloadso <plug-in.so>`
    - `hbaget <name|ID#> <resource_name>`
    - `hbaset <name|ID#> <resource_name> <value>`
    - `hbacat <name|ID#> <resource_name>`
    - `hbalist [name]`

HBA Software (Linux)

Motor controller example:

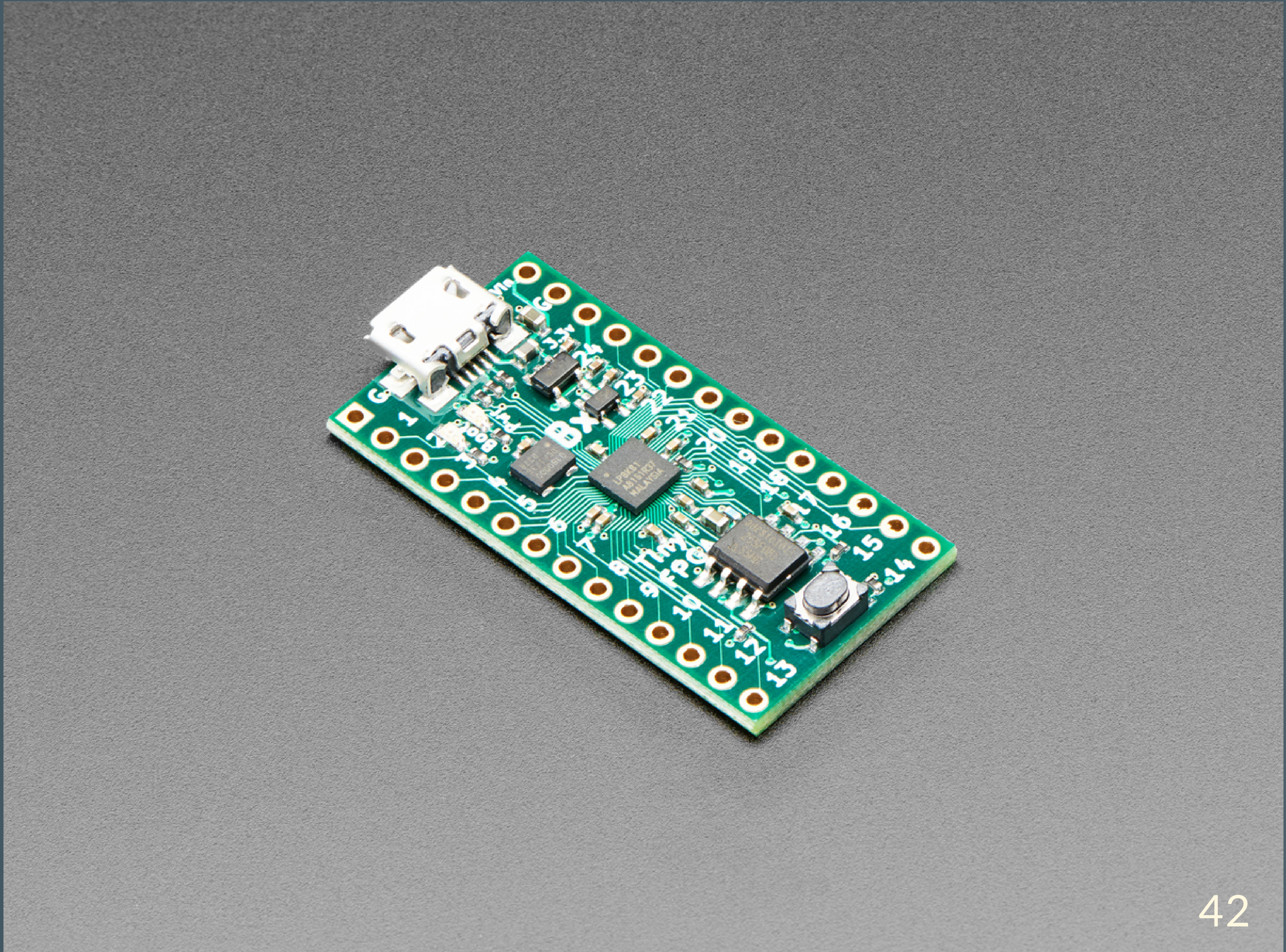
- `mode`: a character in (B)rake, (F)orward, re(V)erse, or (R)otate
- `speed`: the desired speed in ticks/second or meters/second
- `direction`: 0 to 100 with 0=hard left, 50=straight, 100=hard right
- `watchdog`: Brake if no mode, speed, or dir command in this many milliseconds

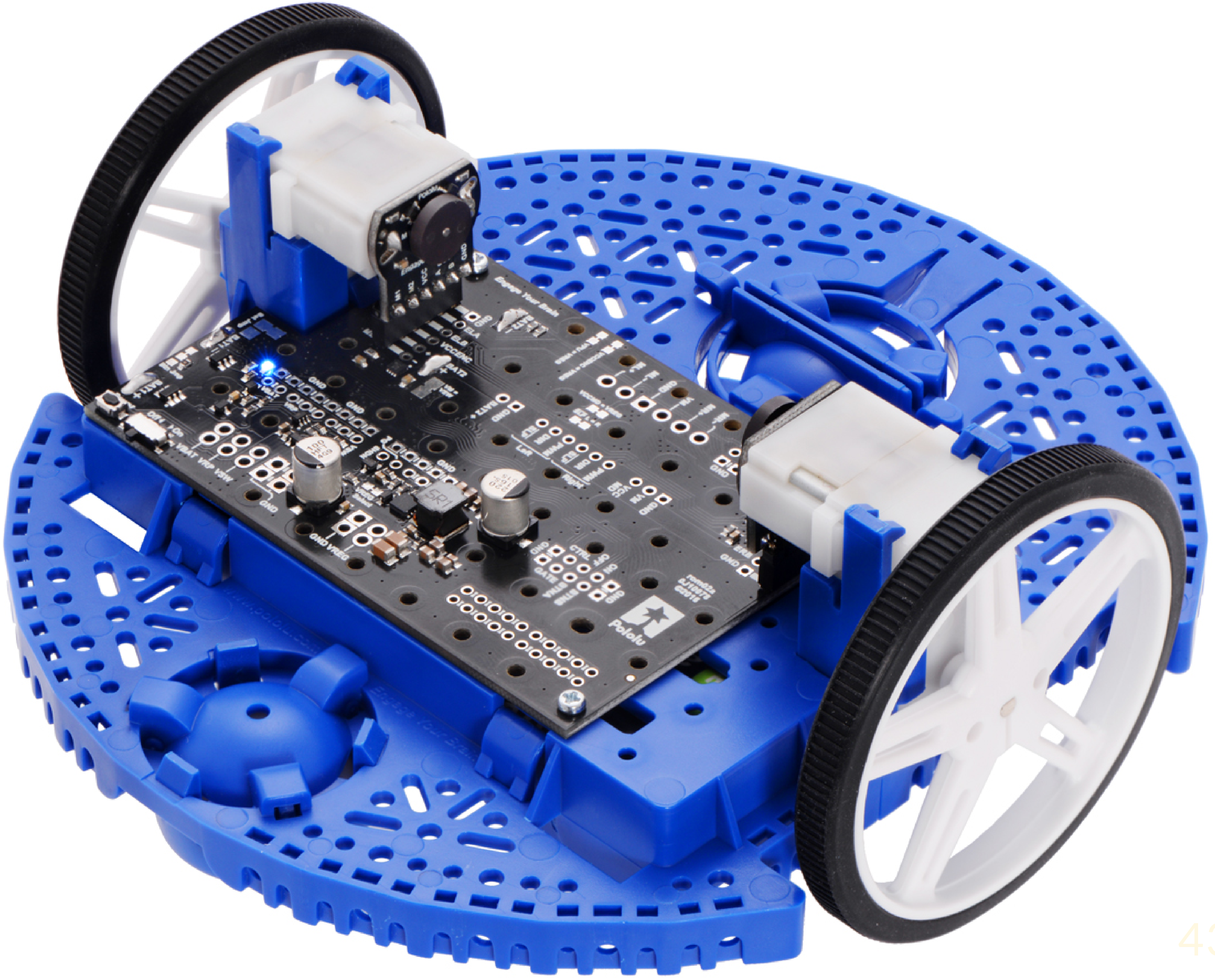
HBA Software (Linux)

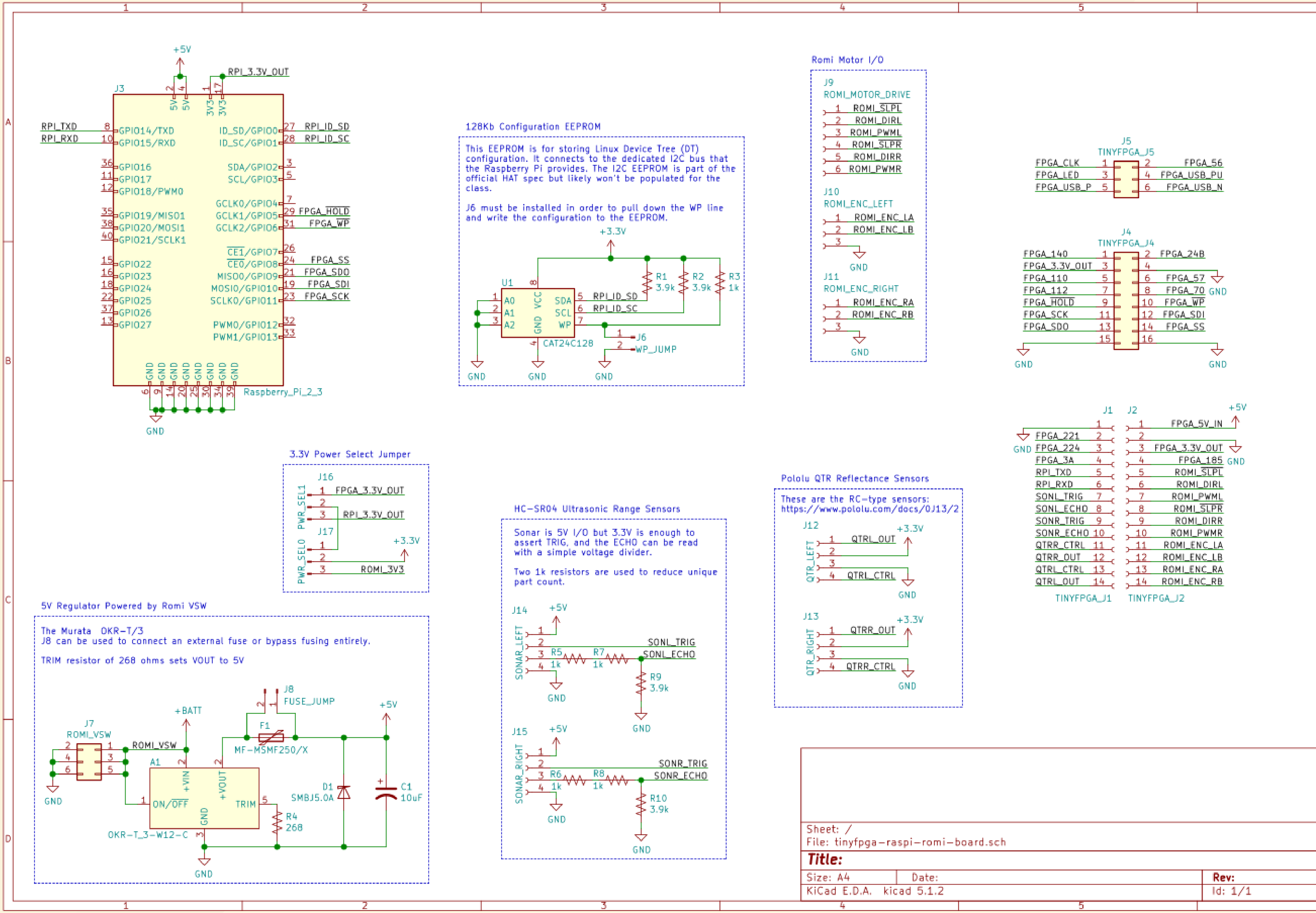
```
hbaset motors mode b     # brake
hbaset motors speed 50
hbaset motors dir 50
hbaset motors mode f     # and off we go...
while true
do
  sleep 2
  hbaset motors dir 100
  sleep 0.75
  hbaset motors dir 50
done
```

HBA Hardware

- Raspberry Pi 3 Model B+ - $45 (incl. extras)
- TinyFPGA BX - $40
- Pololu Romi Chassis Kit - $30
- Pololu Romi Encoder Pair Kit - $10
- Pololu Motor Driver and PDB - $20
- 2x HC-SR04 SONAR sensors - $10
- 2x Pololu QTR IR reflectance sensors - $5
- Custom PCB (power supply & interconnect) - $10?


BX
FPGA





J3
Raspberry_Pi_2_3
+5V
RPI_3.3V_OUT
RPI_TXD
RPI_RXD
GPIO14/TXD
GPIO15/RXD
GPIO16
GPIO17
GPIO18/PWM0
GPIO19/MISO1
GPIO20/MOSI1
GPIO21/SCLK1
GPIO22
GPIO23
GPIO24
GPIO25
GPIO26
GPIO27
ID_SD/GPIO0
ID_SC/GPIO1
SDA/GPIO2
SCL/GPIO3
GCLK0/GPIO4
GCLK1/GPIO5
GCLK2/GPIO6
CE1/GPIO7
CE0/GPIO8
MISO0/GPIO9
MOSI0/GPIO10
SCLK0/GPIO11
PWM0/GPIO12
PWM1/GPIO13
RPI_ID_SD
RPI_ID_SC
FPGA_HOLD
FPGA_WP
FPGA_SS
FPGA_SDO
FPGA_SDI
FPGA_SCK
GND
128Kb Configuration EEPROM
This EEPROM is for storing Linux Device Tree (DT) configuration. It connects to the dedicated I2C bus that the Raspberry Pi provides. The I2C EEPROM is part of the official HAT spec but likely won't be populated for the class.
J6 must be installed in order to pull down the WP line and write the configuration to the EEPROM.
+3.3V
U1
CAT24C128
R1 3.9k
R2 3.9k
R3 1k
J6
WP_JUMP
Romi Motor I/O
J9
ROMI_MOTOR_DRIVE
ROMI_SLPL
ROMI_DIRL
ROMI_PWML
ROMI_SLPR
ROMI_DIRR
ROMI_PWMR
J10
ROMI_ENC_LEFT
ROMI_ENC_LA
ROMI_ENC_LB
J11
ROMI_ENC_RIGHT
ROMI_ENC_RA
ROMI_ENC_RB
J5
TINYFPGA_J5
FPGA_CLK
FPGA_LED
FPGA_USB_P
FPGA_56
FPGA_USB_PU
FPGA_USB_N
J4
TINYFPGA_J4
FPGA_140
FPGA_3.3V_OUT
FPGA_110
FPGA_112
FPGA_HOLD
FPGA_SCK
FPGA_SDO
FPGA_24B
FPGA_57
FPGA_70
FPGA_WP
FPGA_SDI
FPGA_SS
3.3V Power Select Jumper
J16
PWR_SEL1
FPGA_3.3V_OUT
RPI_3.3V_OUT
J17
PWR_SEL0
+3.3V
ROMI_3V3
HC-SR04 Ultrasonic Range Sensors
Sonar is 5V I/O but 3.3V is enough to assert TRIG, and the ECHO can be read with a simple voltage divider.
Two 1k resistors are used to reduce unique part count.
J14
SONAR_LEFT
SONL_TRIG
SONL_ECHO
R5 1k
R7 1k
R9 3.9k
J15
SONAR_RIGHT
SONR_TRIG
SONR_ECHO
R6 1k
R8 1k
R10 3.9k
Pololu QTR Reflectance Sensors
These are the RC-type sensors:
https://www.pololu.com/docs/0J13/2
J12
QTR_LEFT
QTRL_OUT
QTRL_CTRL
J13
QTR_RIGHT
QTRR_OUT
QTRR_CTRL
J1
J2
TINYFPGA_J1
TINYFPGA_J2
FPGA_221
FPGA_224
FPGA_3A
RPI_TXD
RPI_RXD
SONL_TRIG
SONL_ECHO
SONR_TRIG
SONR_ECHO
QTRR_CTRL
QTRR_OUT
QTRL_CTRL
QTRL_OUT
FPGA_5V_IN
FPGA_3.3V_OUT
FPGA_185
ROMI_SLPL
ROMI_DIRL
ROMI_PWML
ROMI_SLPR
ROMI_DIRR
ROMI_PWMR
ROMI_ENC_LA
ROMI_ENC_LB
ROMI_ENC_RA
ROMI_ENC_RB
5V Regulator Powered by Romi VSW
The Murata OKR-T/3
J8 can be used to connect an external fuse or bypass fusing entirely.
TRIM resistor of 268 ohms sets VOUT to 5V
J7
ROMI_VSW
+BATT
J8
FUSE_JUMP
F1
MF-MSMF250/X
A1
OKR-T_3-W12-C
ON/OFF
+VIN
+VOUT
TRIM
R4 268
D1 SMBJ5.0A
C1 10uF
Sheet: /
File: tinyfpga-raspi-romi-board.sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad 5.1.2
Id: 1/1

Class Details

- Where: **Hacker Dojo**
- When: **Wednesday, June 19, 2019 - 7:00pm**
- Cost - **$200** (we'll refund you if materials end up being less)

Questions?

All HBA material is on GitHub:

- https://github.com/hbrc-fpga-class/class-material/blob/master/presentation.pdf