Software Defined Printed Circuit Boards

Jonathan Bachrach

EECS UC Berkeley

August 25, 2016
Software Defined Printed Circuit Boards
“Circuit Board Design for Programmers”

Prof Jonathan Bachrach with Richard Lin @ EECS

Do you want to

Have you ever wanted to design a circuit board but were intimidated?
Have you been frustrated by the tedium of circuit design apps?
Are you a programmer and want circuit board design to be like software design?
Do you want to design boards at the speed of rapid fabrication?

Well this class is for you...

CS194/294-126
upper div / grad / studio / project class

Fall 2016, TuTh 10-11:30a, Jacobs 220
4 Units: 3 hr lecture and 3+ hrs lab time / week.
http://www-inst.eecs.berkeley.edu/~cs194-126

“making the easy things easy and the hardware things software”
Industrial Revolution of Personalized Fabrication

- cheap fabbing – 3D Printers + laser cutters + milling machines
- cheap and available parts – sparkfun, adafruit, seeed studio, ...
- standard interfaces – arduino, mbed, etc
- collective development – makerspaces
Motivation

IoT
good work labs

Robotics
Utah Valley University
Problem

Current Design Flows are
- time consuming
- inaccessibly complicated
- not reusable

Situation:
- idle printers
- domain experts without access to fabrication
High level design with software generators
Use software to define hardware and software

Because
- highly parametric
- massive reuse
- abstract away low-level details
- easy design space exploration
- extreme integration across EE and CS (and ME)
Declaratively design embedded systems

Inputs
- list of peripherals
- application in C++

Outputs
- (multiple) board files
- compiled firmware
- networking layer
JITPCB Movie

JITPCB

IROS 2016

JONATHAN BACHRACH
DAVID BIANCOLIN
AUSTIN BUCHAN
DUNCAN HALDANE
RICHARD LIN
Current Breadboard Prototyping

- ungainly format
- fragile
- manual and error prone efforts – pin conflicts
- doesn’t scale
Current Modular Circuit Prototyping

- modular spaghetti and expense
- arduino frankenboards – why have interfaces? supports modules
- no perfect premade modules
- still manual and error prone efforts – pin conflicts
- doesn’t scale
- CAD is tedious and only WYSIWYG
- fabrication is slow and expensive
Enter OtherMill or MCB

- JIT manufacturing – mills blank copper boards while you wait
- produces structurally sound electronics in desired form factors
- relatively inexpensive (approx $3000 for mill and $5 for boards)
- 20 $in^2$/ hour
- 1 or 2 layers and no solder mask for now
- still requires labor intensive Eagle design
othermill is an order of magnitude lower cost and time
Chisel is ...

- Best of hardware and software design ideas
- Embedded within Scala language to leverage mindshare and language design
- Not Scala -> Verilog
- Algebraic construction and wiring
- Hierarchical, object oriented, and functional construction
- Abstract data types and interfaces
- Bulk connections
- Multiple targets
  - Simulation and synthesis
  - Memory IP is target-specific
Methods

- SOB architecture
- stanza
- hardware description
- software model
Template

- Peripherals
- IO processors
  - one master and rest slaves
  - peripherals attach to pins
- Network

Benefits

- Modular and Scalable
- Easier SW and Routing
- best of scripting and production languages
  - easy to understand and powerful to use
  - gradual types -> easy parameteric types
- simple orthogonal concepts
  - functions, objects, pipes, and namespace separated
  - use concepts in unlimited ways – serendipity
  - entire language including optimizing native compiler in 20K LOC
- powerful macros for conventional syntax
  - almost entire stanza syntax written as macros
  - better DSL hosting language

* developed by Patrick Li @ EECS Berkeley
Stanza by Contrast

- like Python but
  - types and on and on and on ...
- like Scala but
  - thinner – native compiler + runtime in 20K LOC
  - simpler – fewer base concepts
  - more separated – orthogonal functions / objects
- like Clojure but
  - conventional syntax – love the parens but ...
  - thinner – native
  - more powerful macro system – not just name macros
- like Dylan but
  - improved gradual types – parameteric types
  - better multimethod namespaces – fewer name clashes
  - more powerful macros – syntax written in it
  - has pipes – generalized control flow mechanism
- components
- modules
- layout
- autorouter
**Component Descriptions:**

```haskell
defcomponent LED ("LED", "LED5MM") :
  pad A
  pad K

defcomponent Resistor ("RCL", "0204/7") :
  pad 1
  pad 2
```

**Module Description:**

```haskell
defmodule PullUpLED () <=: Module:
  ;; interface
  inherit gnd
  inherit v3
  sig io
  ;; subcomponents
  mod l  : LED("red")
  mod r  : Resistor("1k")
  ;; netlists
  sig up  = [r.1, l.A]
  sig v3  = [r.2]
  sig gnd = []
  sig io  = [l.K]
  ;; layout
  lay HBox([Rot(r,90), Rot(led,90)], 1.0)
```
- mBED
- drivers
- virtualized peripherals – RPC calls
- bootloader
C++ Object Oriented Library for Embedded
ARM consortium and works across chips / boards
Standard and nicely designed

```cpp
I2C master(SDA_PIN, SCL_PIN);
master.frequency(I2C_FREQ);
master.read(SLAVE_ADDR,(char*) &SW1_dat,1);
master.write(SLAVE_ADDR,(const char*) &LED1_dat,4);

DigitalIn SW0_dev(PTB1);
PwmOut LED0_dev(PTB10);
```
Peripheral Interface (Stanza):

defn bool_out (name:Symbol, hw:Symbol -> PeripheralModule) :
    Peripheral(name, "BoolOut", hw,
        [Pin('DigitalOut, pad#io)],
        [Port("value", IntType(0))])

Peripheral Interface (C++):

// Automatically generated Peripheral and Master class
class BoolOutPeriph : public Peripheral {
    public:
        int value;
};
class BoolOutPeriphMaster : public BoolOutPeriph {
    // Initialize and synchronize local class variables
    // to Slave instantiations over network
};

// User completed Slave functionality class
class BoolOutPeriphSlave : public BoolOutPeriph {
    public:
        BoolOutDevice(PinName pin) : dout_pin(pin) { }
        void step(void): { dout_pin.write(value); }
    protected:
        DigitalOutPin dout_pin;
};
- program logic agnostic to device placement
  - user defines init and main on master
  - transform input to output data through net addressable registers
- generate HW + SW together
  - HW IDs
  - device / data layout
- use RPC for remote control/status registers
- bootloader loads code on i2c according to ids

```c
#include "master.h"

void main (void) {
  init(); // init user code / drivers
  for (;;) {
    motor0.torque = f(motor0.pos, ...);
    motor1.torque = g(motor1.pos, ...);
    step(); // step drivers
  }
}
```
master.h (CPU16.h)

```c
MotorMaster motor0, motor1;
void pull(void) { ... } ... void push(void) { ... }
```

CPU18.h

```c
MotorSlave motor0(pin3, ...);
void pull(void) { ... } ... void push(void) { ... }
```

CPU20.h

```c
MotorSlave motor1(pin3, ...);
void pull(void) { ... } ... void push(void) { ... }
```
- programmed over I2C
- code organized by IO processor from low to high with cpu-id.elf
- token passed down i2c bus programming processors in order
- IOP with token gets programmed
Results

- pong
- robot arm
- mobile robot
Peripheral layout.

IOP layout.

Board front.

Board back.
Robot Arm


System of three boards.
Mobile Robot.

Board detail.
Mobile Robot with Nucleo
Arduino Boards with JITPCB

- physically sound
- automatic pin assignment
- scalable
- voltage + resistance + current annotations
- solve for missing quantities
- check for correct wiring
- solve for power supply
Fabricated: The New World of 3D Printing, by Hod Lipson, Melba Kurman
Makers: The New Industrial Revolution, by Chris Anderson
Makers, by Cory Doctorow
The Third Industrial Revolution, The Economist
JITPCB, by Jonathan Bachrach, David Biancolin, Austin Buchan, Duncan Haldane, Richard Lin