

# A Library and Platform for Bitstream Manipulation

---

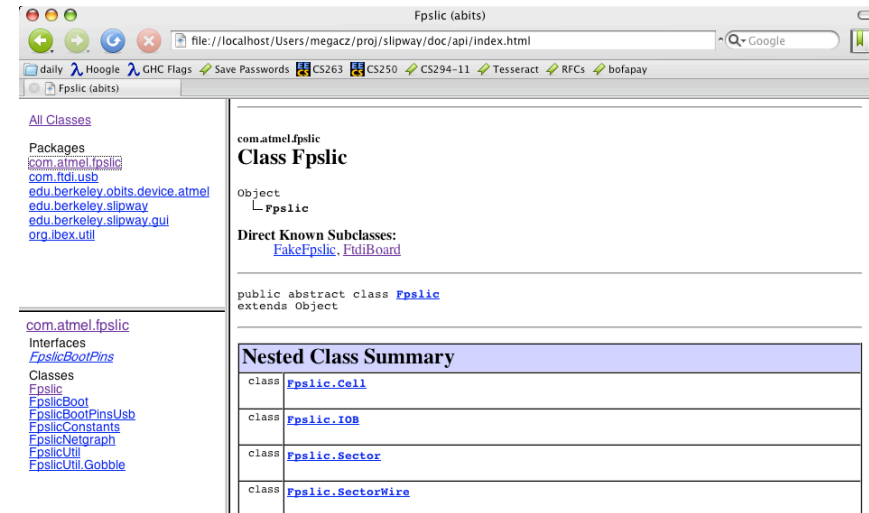
Adam Megacz  
UC Berkeley  
megacz@cs.berkeley.edu

FCCM  
23-Apr-2007

# The Library and Platform

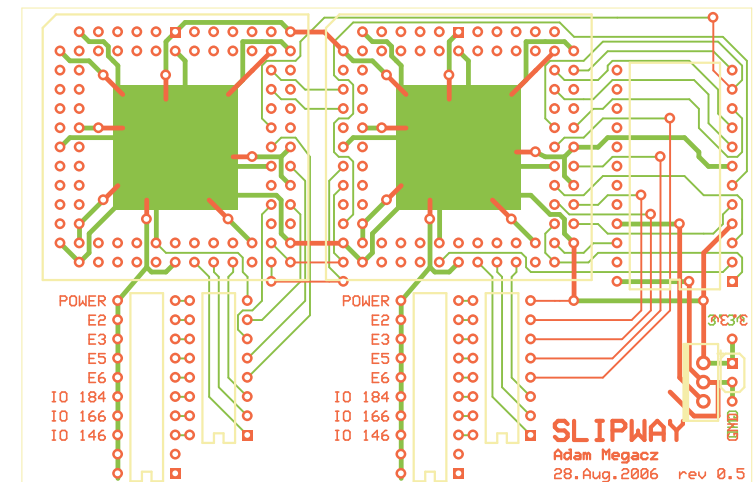
## • Abits

- Java library for Atmel At94k FPGAs
- bitstream creation/modification
- on-line partial reconfiguration
- 100% open source (BSD license)



## • Slipway

- Development board reference design
- ~USD\$60 for pcb+parts
- Assemble by hand in ~30min



# Background

---

# Background

---

- Microprocessors have *publicly documented* instruction sets

# Background

---

- Microprocessors have *publicly documented* instruction sets
  - Users *expect* and even *assume* this.

# Background

---

- Microprocessors have *publicly documented* instruction sets
  - Users *expect* and even *assume* this.
- Until ~1997, FPGAs typically did as well

# Background

---

- Microprocessors have *publicly documented* instruction sets
  - Users *expect* and even *assume* this.
- Until ~1997, FPGAs typically did as well
- Post-1997: manufacturers begin to treat FPGA “instruction sets” as a trade secret

# Background

---

- Microprocessors have *publicly documented* instruction sets
  - Users *expect* and even *assume* this.
- Until ~1997, FPGAs typically did as well
- Post-1997: manufacturers begin to treat FPGA “instruction sets” as a trade secret
  - Claim that users don’t mind this.



# So What?

---

# So What?

---

- Ability to treat *code as data* is one of the most powerful aspects of the von Neumann architecture.

# So What?

---

- Ability to treat *code as data* is one of the most powerful aspects of the von Neumann architecture.
  - Last 50 years: rich body of knowledge emerges for dealing with code as data: compilers, program transformation, linkers, loaders, garbage collectors, debuggers.

# So What?

---

- Ability to treat *code as data* is one of the most powerful aspects of the von Neumann architecture.
  - Last 50 years: rich body of knowledge emerges for dealing with code as data: compilers, program transformation, linkers, loaders, garbage collectors, debuggers.
  - Last 7 years: Hotspot JVM takes *runtime code generation* from “obscure research topic” to “standard feature”

# So What?

---

- Ability to treat *code as data* is one of the most powerful aspects of the von Neumann architecture.
  - Last 50 years: rich body of knowledge emerges for dealing with code as data: compilers, program transformation, linkers, loaders, garbage collectors, debuggers.
  - Last 7 years: Hotspot JVM takes *runtime code generation* from “obscure research topic” to “standard feature”
    - ➔ Performance cost of dynamic language features drops.

# So What?

---

- Ability to treat *code as data* is one of the most powerful aspects of the von Neumann architecture.
  - Last 50 years: rich body of knowledge emerges for dealing with code as data: compilers, program transformation, linkers, loaders, garbage collectors, debuggers.
  - Last 7 years: Hotspot JVM takes *runtime code generation* from “obscure research topic” to “standard feature”
    - ➔ Performance cost of dynamic language features drops.
      - ➔ A whole generation of business software is written in higher-level languages.

# So What?

---

- Ability to treat *code as data* is one of the most powerful aspects of the von Neumann architecture.
  - Last 50 years: rich body of knowledge emerges for dealing with code as data: compilers, program transformation, linkers, loaders, garbage collectors, debuggers.
  - Last 7 years: Hotspot JVM takes *runtime code generation* from “obscure research topic” to “standard feature”
    - ➔ Performance cost of dynamic language features drops.
      - ➔ A whole generation of business software is written in higher-level languages.
      - ➔ Programmer productivity rises.

# So What?

---



# So What?

---

- FCCMs/FPGAs are one of the leading alternatives to the von Neumann computation model

# So What?

---

- FCCMs/FPGAs are one of the leading alternatives to the von Neumann computation model
  - No *inherent* barrier to code-as-data paradigm.

# So What?

---

- FCCMs/FPGAs are one of the leading alternatives to the von Neumann computation model
  - No *inherent* barrier to code-as-data paradigm.
    - However, *practical* barrier of bitstream secrecy.

# So What?

---

- FCCMs/FPGAs are one of the leading alternatives to the von Neumann computation model
  - No *inherent* barrier to code-as-data paradigm.
    - However, *practical* barrier of bitstream secrecy.
  - FPGA languages, compilers and tools have *not evolved as quickly* as those for software.

# So What?

---

- FCCMs/FPGAs are one of the leading alternatives to the von Neumann computation model
  - No *inherent* barrier to code-as-data paradigm.
    - However, *practical* barrier of bitstream secrecy.
  - FPGA languages, compilers and tools have *not evolved as quickly* as those for software.
  - FPGA design productivity *has not kept pace* with software design productivity growth.

# So What?

---

- FCCMs/FPGAs are one of the leading alternatives to the von Neumann computation model
  - No *inherent* barrier to code-as-data paradigm.
    - However, *practical* barrier of bitstream secrecy.
  - FPGA languages, compilers and tools have *not evolved as quickly* as those for software.
  - FPGA design productivity *has not kept pace* with software design productivity growth.
  - Coincidence?

# Atmel At94k Background

---

- **Positive**

- Fine-grained, “sea of gates”
  - Fast connections to eight nearest neighbors
  - 10-wire routing channel for each row/column
- Partial reconfiguration on an extremely fine grain
- On-die AVR microcontroller (manages partial reconfig)

- **Negative**

- Manufactured on an old 0.35 $\mu$ m process
- Nominal clock rate is ~100Mhz
- Largest device is 2300 CLBs (CLB = FF+4LUT)

# Abits

---

- Library for configuring Atmel At94k FPGAs
  - Written in Java
  - Bitstream creation, modification, parsing
  - On-line partial reconfiguration
  - 100% open source, BSD license

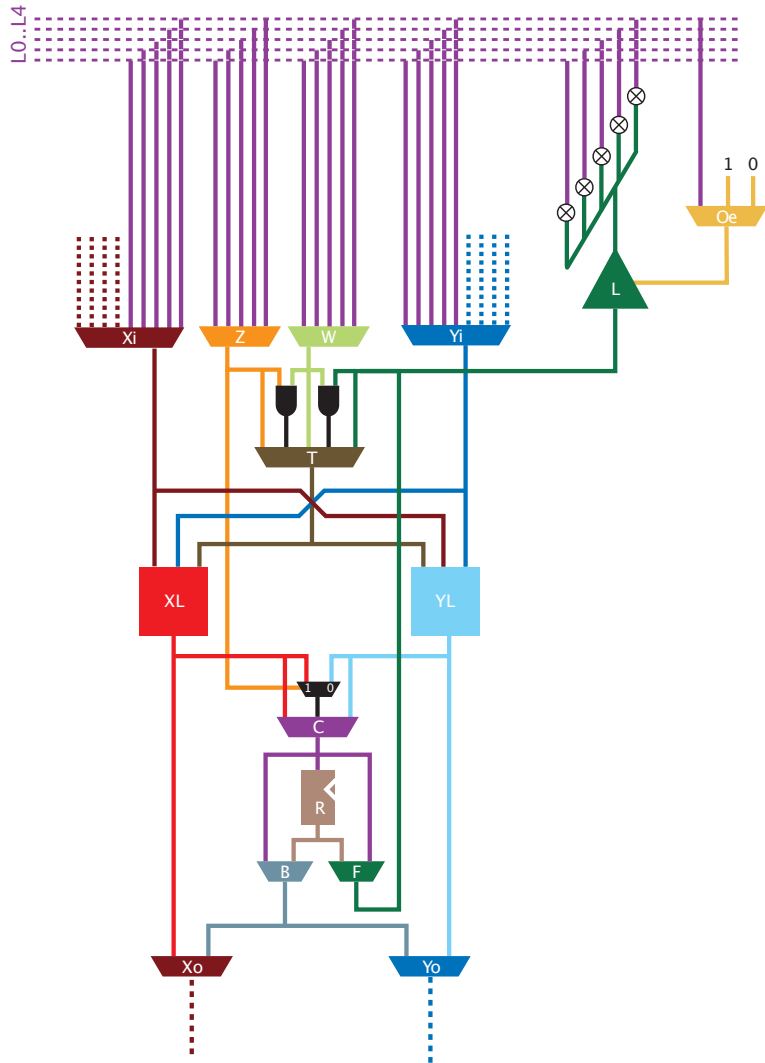


# Abits: API

---

- Same API for bitstreams and live devices
  - bitfile on disk
  - bitstream in memory
  - running device accepting partial reconfiguration commands
- Heap-efficient
- Very low-level
- 4 user-visible classes, ~55 methods

# Abits: Example



```
void foo(Fpslic fpslic) {
```

```
    Cell cell =  
        fpslic.getCell(10,10);
```

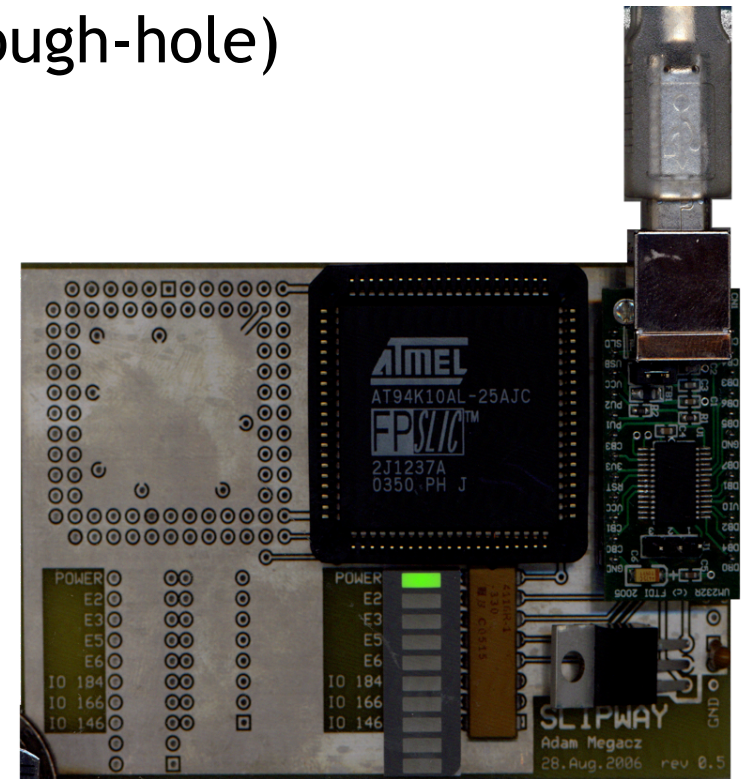
```
    // X-Lut computes constant 1  
    cell.xlut(0xff);
```

```
    // Y-Lut computes (Xin & Yin)  
    cell.yi(SOUTH);  
    cell.ylut(LUT_SELF & LUT_OTHER);
```

```
    // write to device (or file)  
    fpslic.flush();
```

# Slipway

- Reference design for development board
  - USD\$60 for pcb+parts
  - Assemble by hand in <30min (all through-hole)
  - Board masks are BSD licensed
- USB interface
  - Provides hard reset, configuration and 1Mbit/sec serial communication
  - Bus powered
  - No creaky parallel ports
  - *No drivers!*
  - Host-side library is also BSD licensed

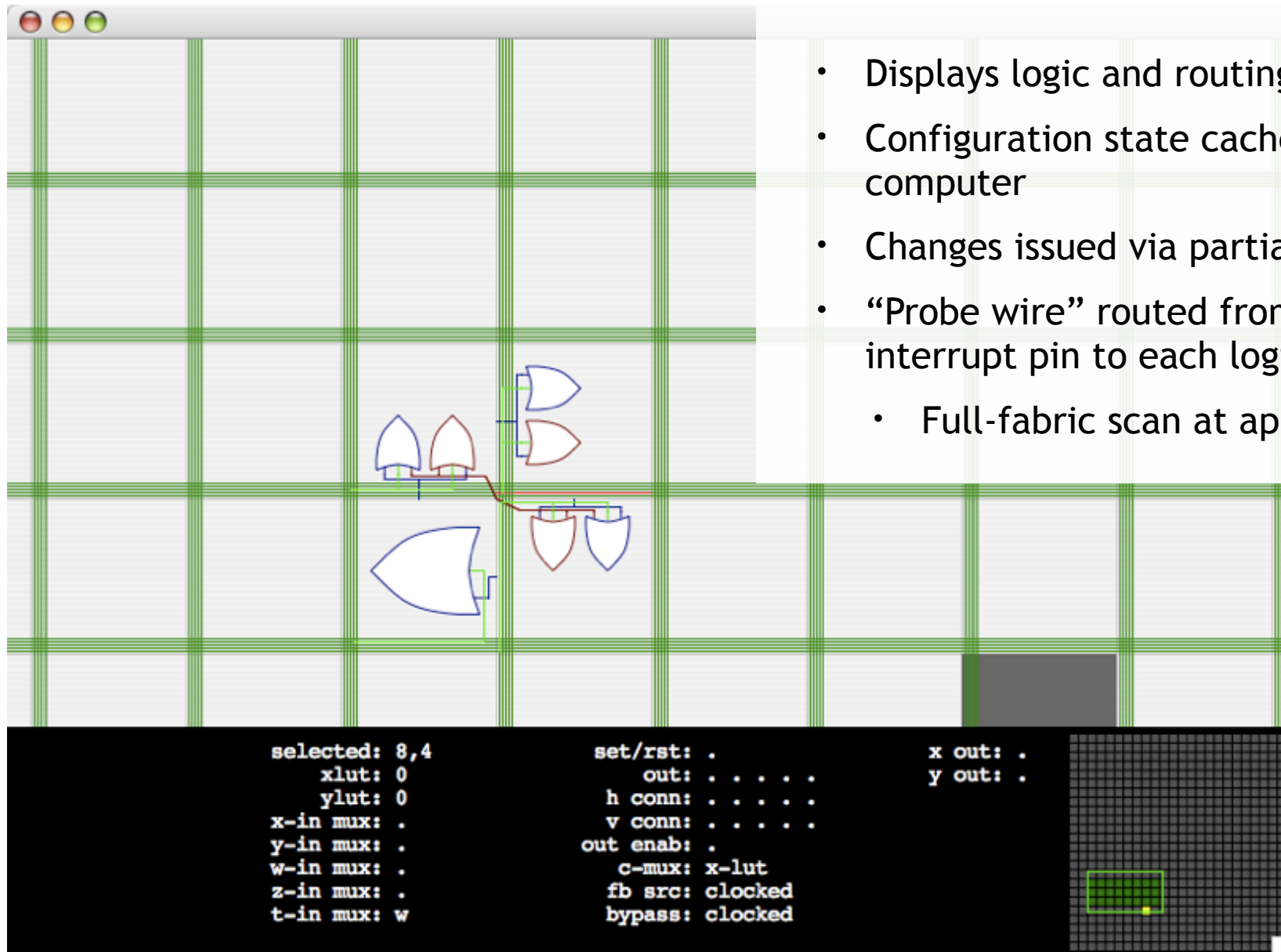


# The Applications

---

- Live fabric editor
  - Mirrors configuration state of device
  - Scan device state by reconfiguring routing of “debug wire”
- Asynchronous (clockless) FIFO
  - Performance scales smoothly with temperature changes
  - Room temperature data “velocity” of 533 Mshifts/sec
  - Performance gain due to a logic block configuration which *cannot be produced using the manufacturer’s tools.*
- High-speed event counter
  - Reliably count events occurring at >600MHz

# Live Fabric Editor



The screenshot displays the Live Fabric Editor interface. The main area shows a grid of logic cells with several cells containing logic components (AND, OR, and NOT gates) and routing paths. The bottom panel contains a terminal window with the following configuration parameters:

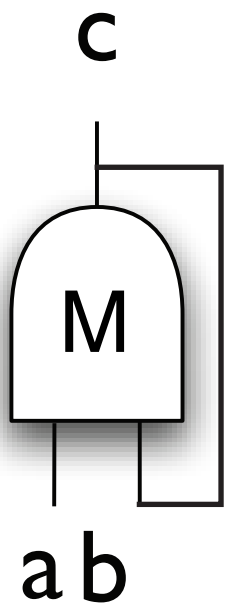
```
selected: 8,4
xlut: 0
ylut: 0
x-in mux: .
y-in mux: .
w-in mux: .
z-in mux: .
t-in mux: w
set/rst: .
out: . . . . .
h conn: . . . . .
v conn: . . . . .
out enab: .
c-mux: x-lut
fb src: clocked
bypass: clocked
x out: .
y out: .
```

A small grid icon is visible in the bottom right corner of the terminal window.

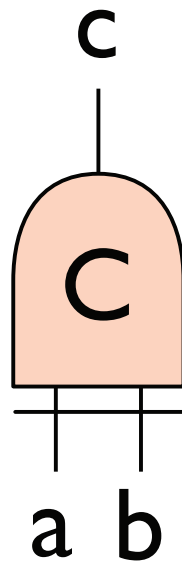
- Displays logic and routing
- Configuration state cached on host computer
- Changes issued via partial reconfiguration
- “Probe wire” routed from microcontroller interrupt pin to each logic cell
  - Full-fabric scan at approximately 5Hz

# Asynchronous (clockless) FIFO

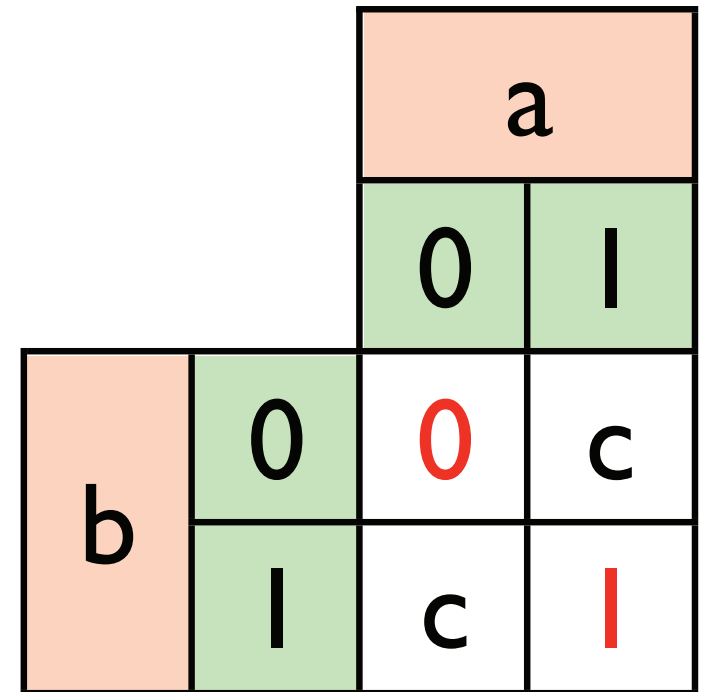
- Fundamental component: Muller C-Element



=

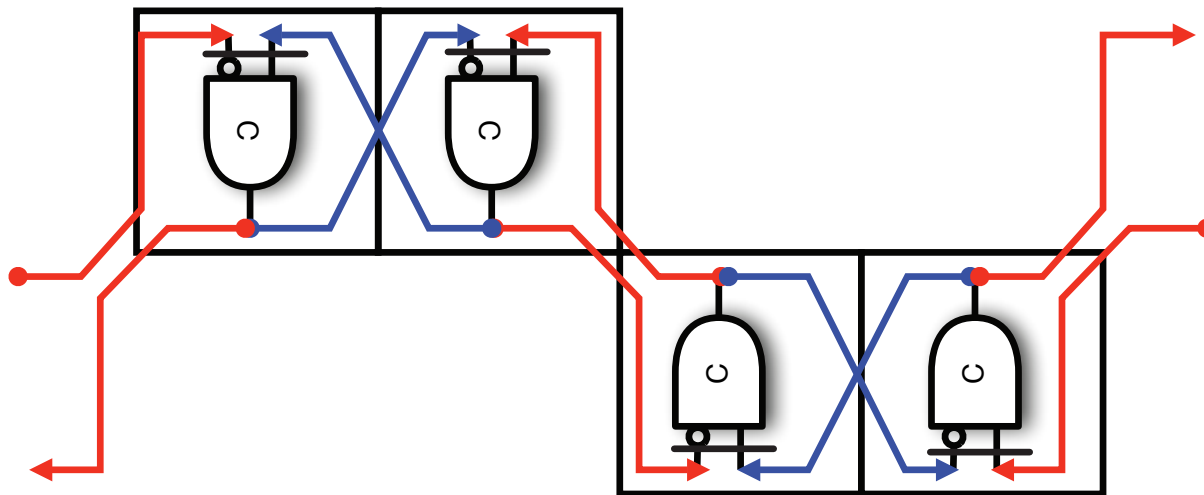


=



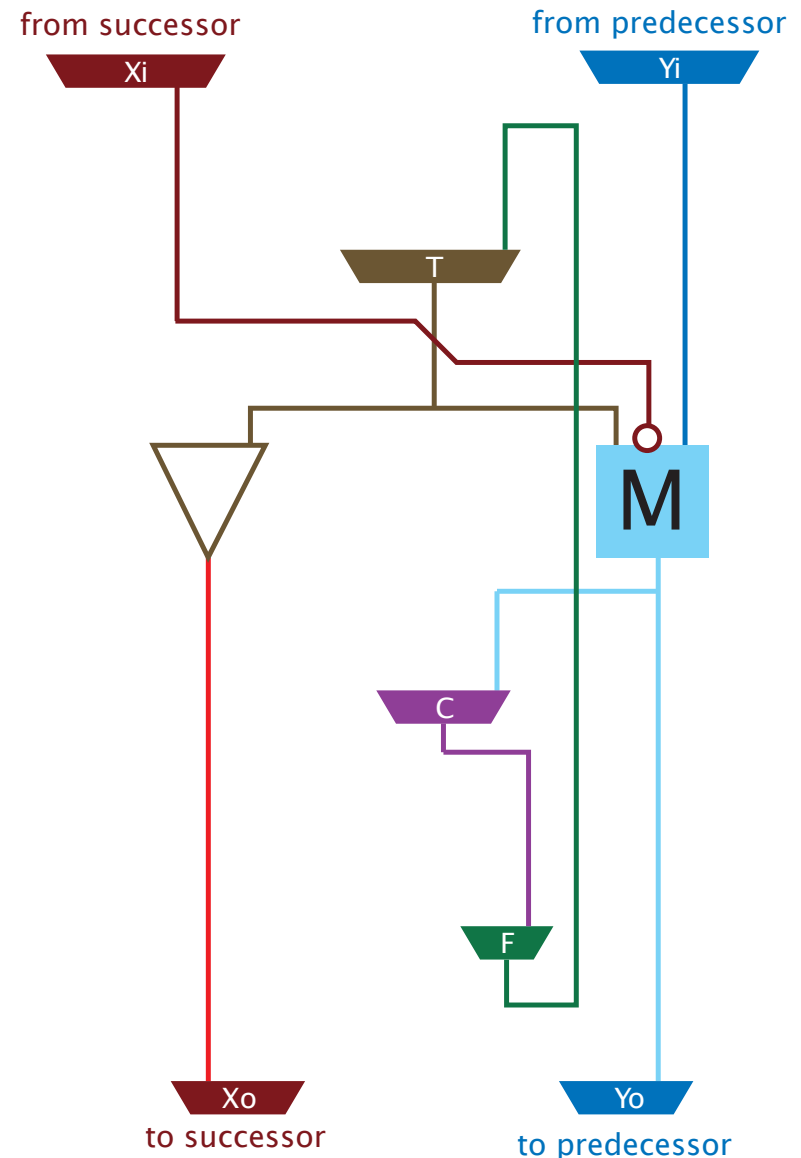
# Asynchronous (clockless) FIFO

- Chain of Muller C Elements



# Asynchronous (clockless) FIFO

- Muller C-Element Configuration
  - Utilizes *internal combinational feedback* feature of Atmel CLB
  - Manufacturer tools cannot exploit this feature
- Achieves peak token velocity of 533Mstages/sec

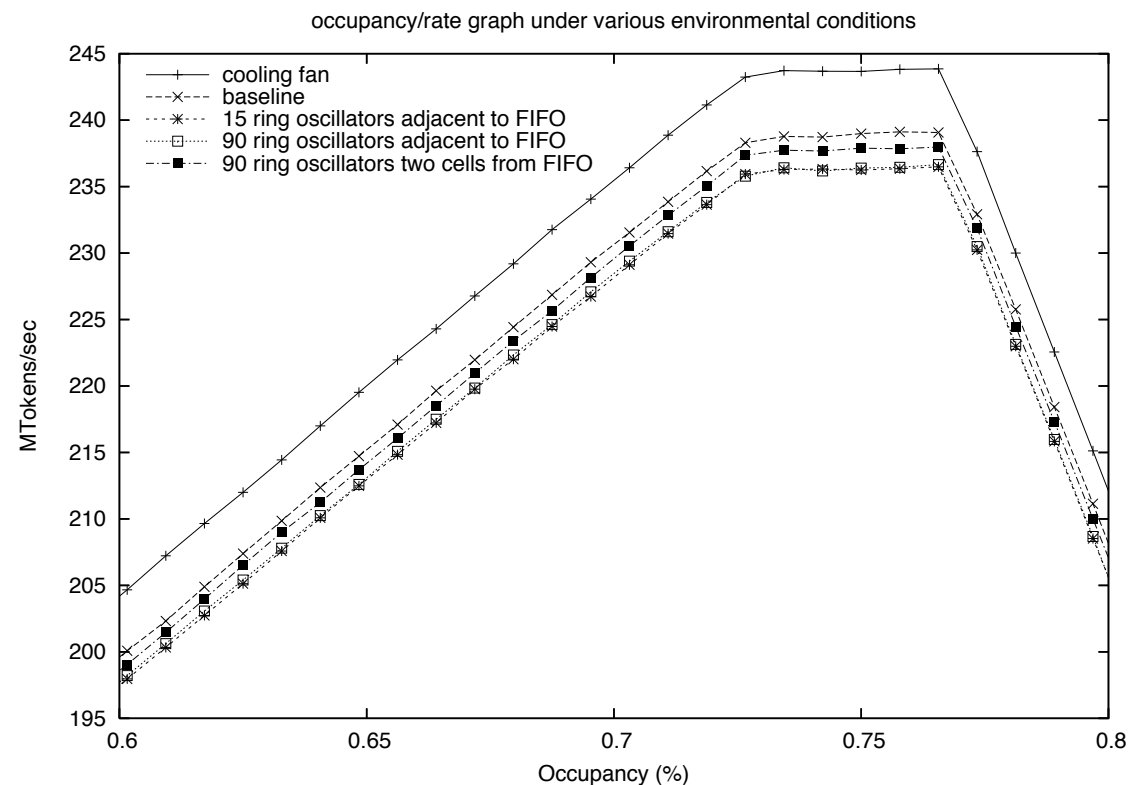




# Asynchronous (clockless) FIFO

- As with all clockless ring FIFOs, the *occupancy/rate* graph is divided into three slope regions

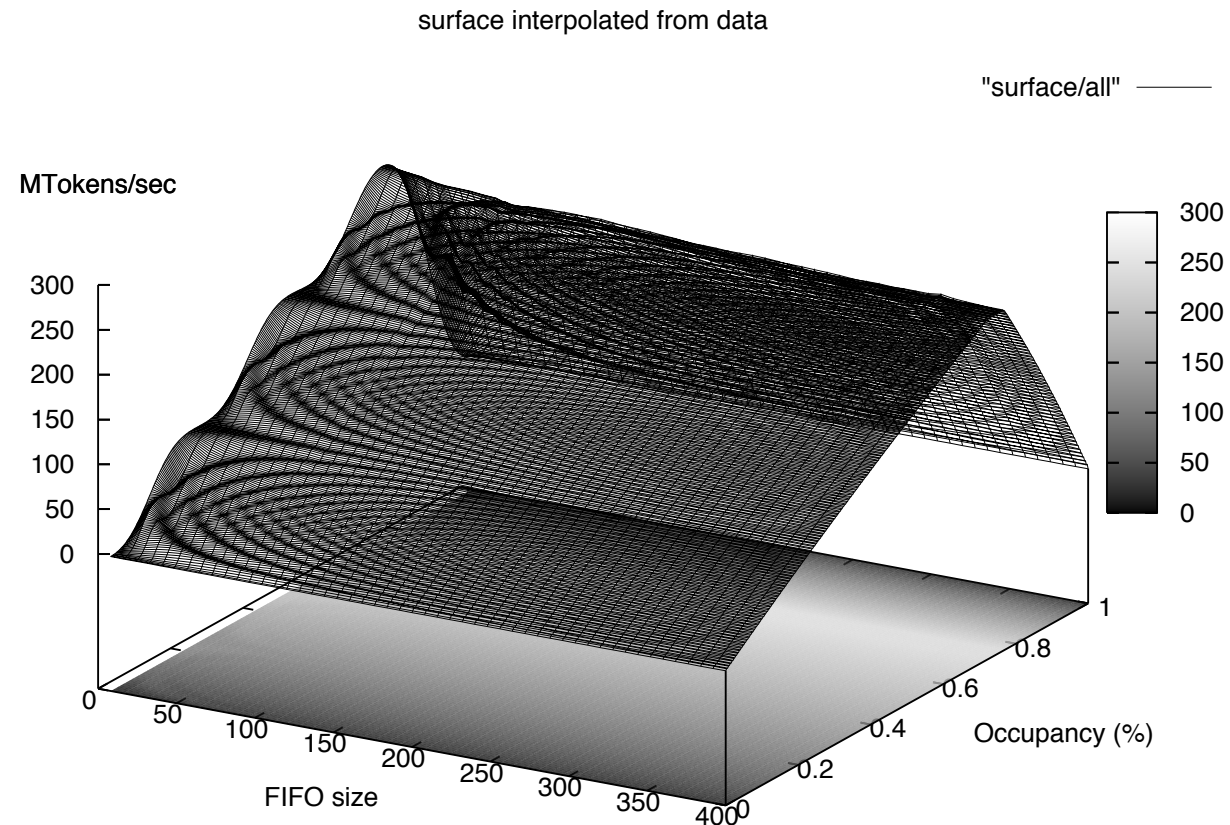
- Left side: limited by *forward propagation time*
- Plateau: limited by *communication*
- Right side: limited by *stage recovery time*



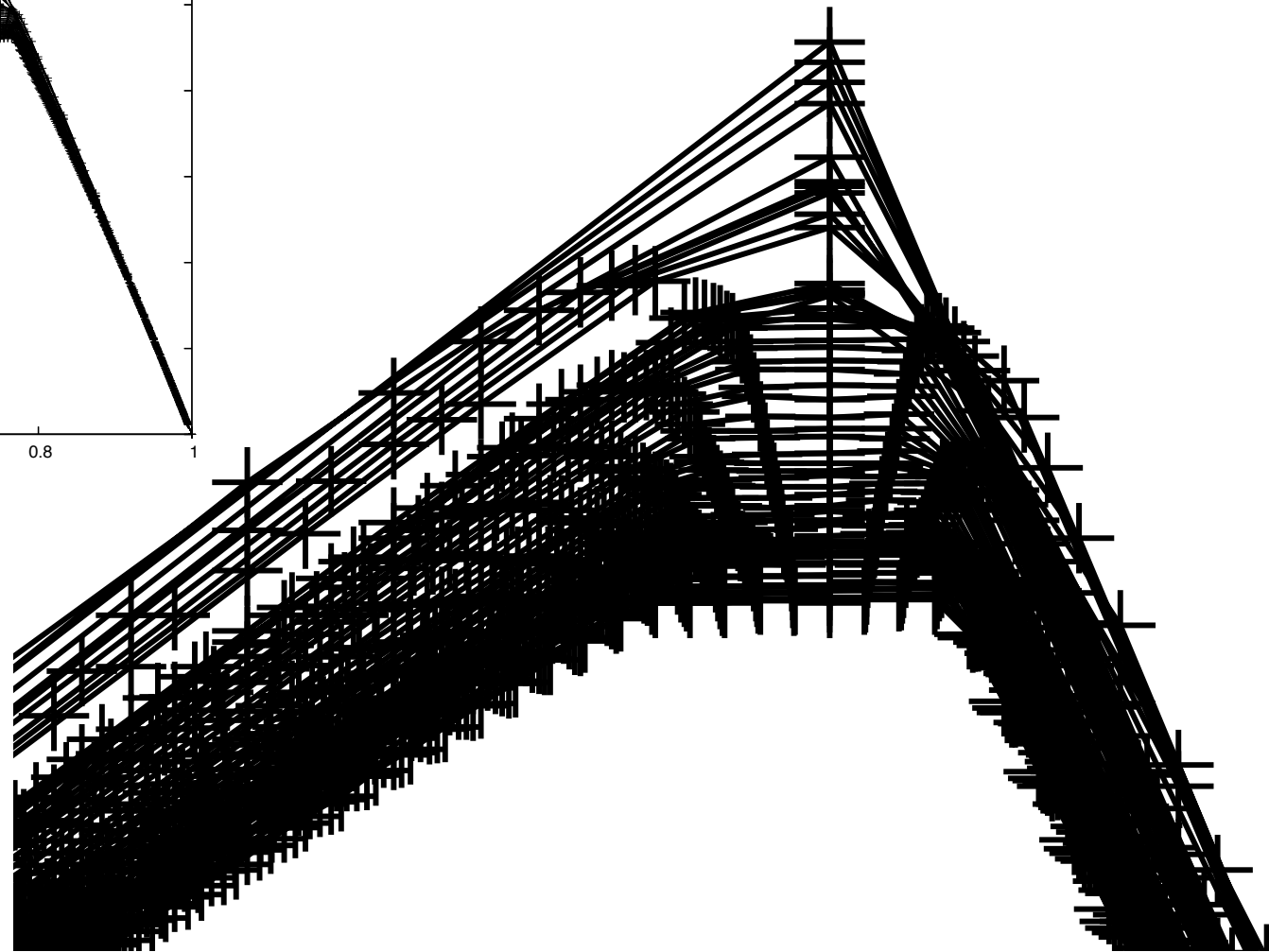
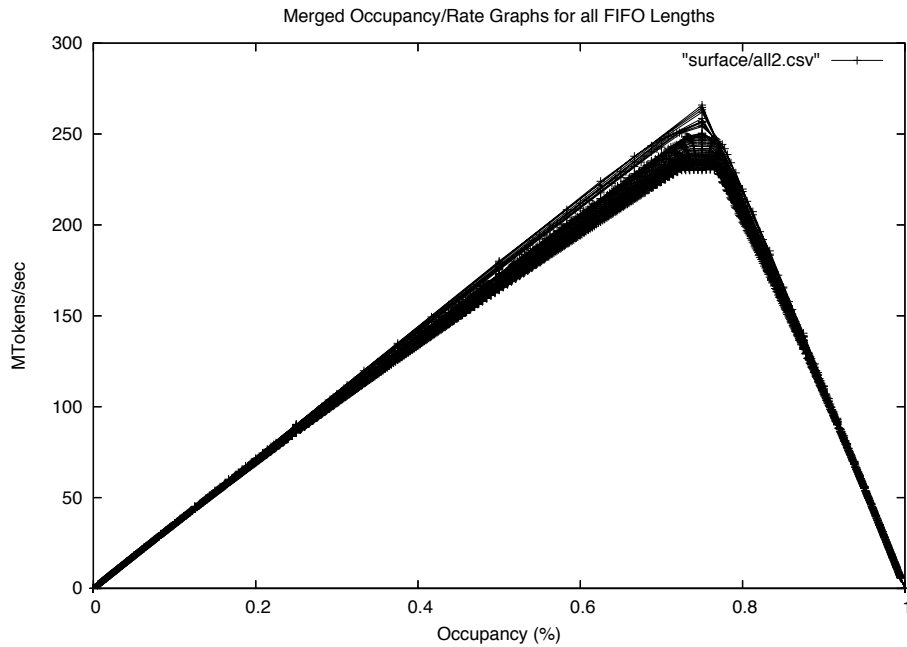
# Asynchronous (clockless) FIFO

- Rate/occupancy graphs are well-studied for fixed FIFO sizes

- Typically in custom VLSI, fixed number of stages
- Reconfigurable hardware lets us try *all 400 possible sizes*
  - Much higher resolution on combined rate/occupancy graph

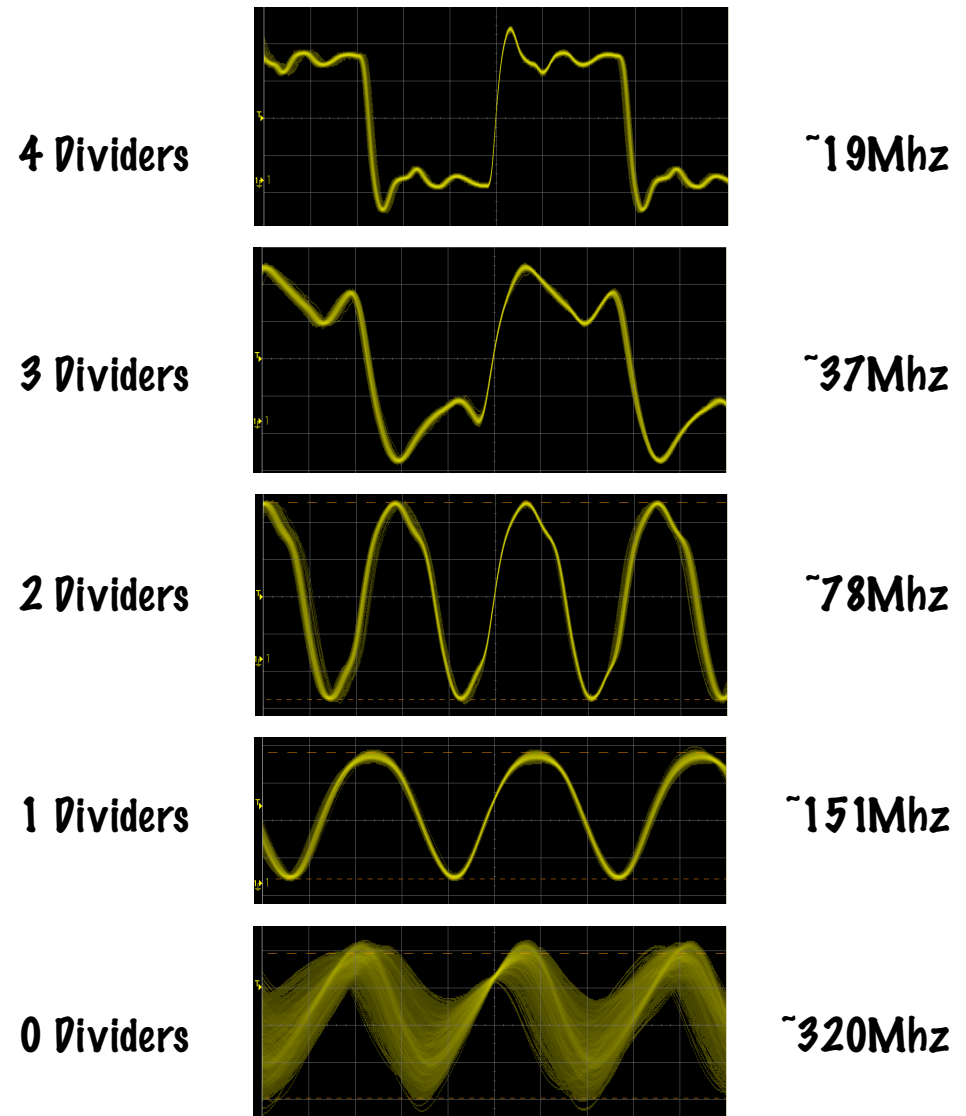


# Asynchronous (clockless) FIFO



# High-speed event counter

- High frequency signals cannot be brought out to pads
  - Signal distortion, missed edges
- Solution: on-chip 1-bit clockless counter

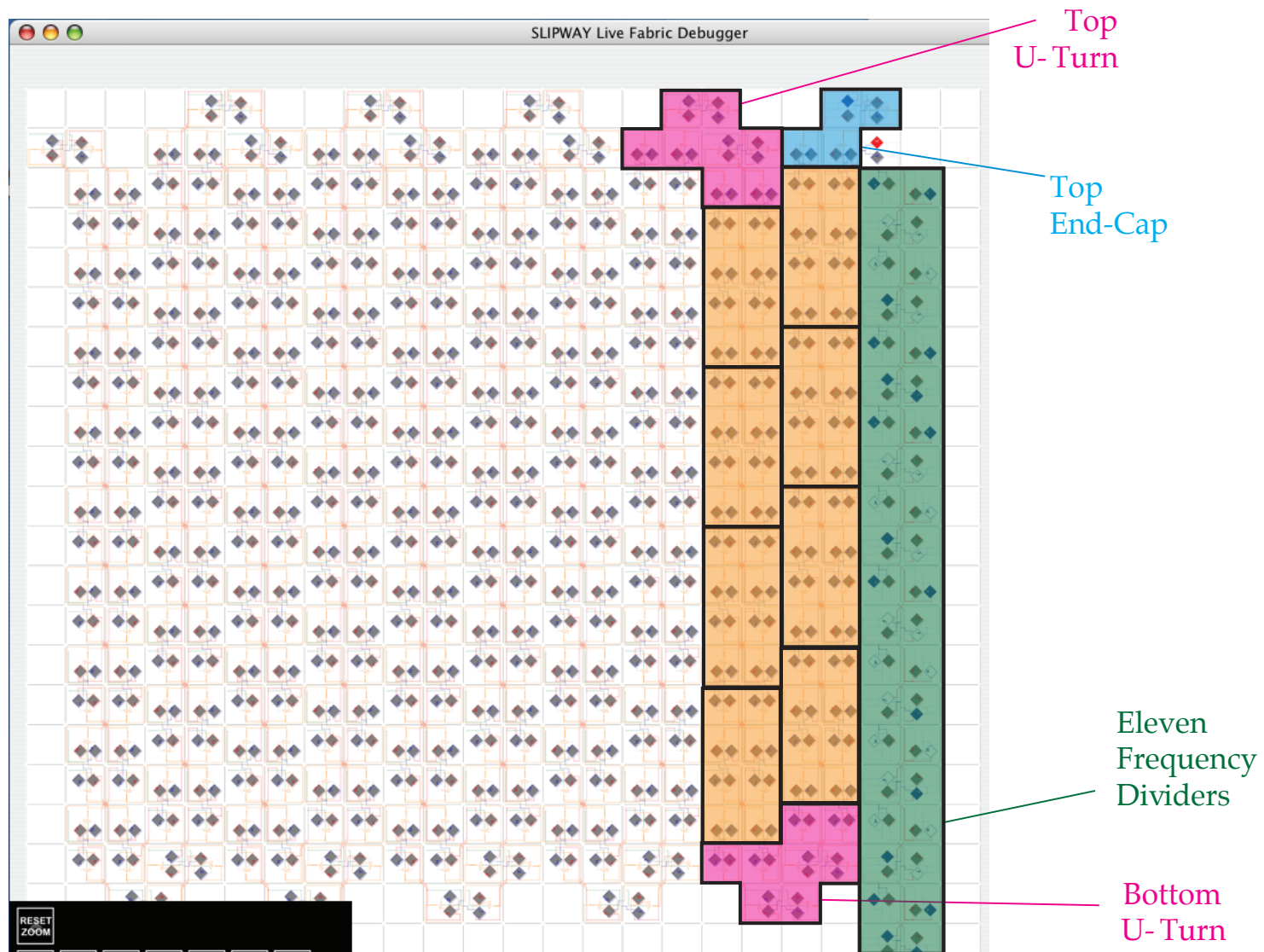


# High-speed event counter

---

- Chain of self-timed counters
- Can reliably count events at ~600Mhz
  - Exceeds rate of two-cell ring oscillator
  - Approaches rate of single-cell oscillator
  - Vastly exceeds toggle rate of flip flops
- Crude but useful “on-chip oscilloscope”

# High-speed event counter



# Summary

---

- Bitstream manipulation can be made easy
- Bitstream manipulation enables new applications
- Bitstream manipulation opens up new research areas
- Public bitstream documentation improves quality of tools
- Evidence: abits library, slipway board
  - Existence proof

# What is Next?

---

- Currently: adding support for more devices & vendors
- Potentially: foundational component of a *completely* open-source FPGA toolchain



# Questions?

---

<http://research.cs.berkeley.edu/project/slipway/>