From Silicon to Compiler

Reverse-Engineering the CoolRunner-II Bitstream Format

Dr. Andrew Zonenberg (@azonenberg)
Senior Security Consultant

IOActive
Hardware | Software | Wetware
SECURITY SERVICES

Outline

- Introduction / motivation
- [C|S]PLD architecture overview
- Block diagram and die overview
- Detailed functional analysis
- Live demo
- Sorry, no cute cat pictures ☹

About Me

- Full-stack hacker, transistors to web apps
- RPI ‘15 PhD CS
  - Created and taught a new class on hardware RE
  - Thesis was on hardware-based OS for SoCs
- Active contributor to siliconpr0n
- Been with IOA since January

IOActive™

Motivation

- Programmable logic is everywhere these days
  - Cheaper than ASICs, dominating low volume
- It's full of black boxes
- No visibility into the compilers (RoTT anyone?)
- Can't dev on platforms official compilers don't run on
- Can't debug or RE bitstreams
  - Vendors want you to think bitstream RE is hard
  - But is it really?

Methodology

```
azonenberg@foxacid:/opt/Xilinx/14.7$ du -h --summarize .
18G     .
```

- Nobody wants to read gigabytes of spaghetti in IDA
  - Plus, it's against the EULA (if you care about that stuff)
- Nobody ever said I can't look at the silicon, though…

Open source… by any means necessary

IOActive™

Our target – Xilinx XC2C32A

- Dirt cheap – just over $1 each
- Choice of QFN, QFP, or csBGA
- Nice “big” 180nm process (4-metal UMC eNVM)
- Small bitstream (~12K)
- Simple product term arch with no hard IP cores
- Old-ish chip, but still in volume production
- Vendor tools are free of charge

IOActive™

JEDEC programming files

- ASCII container format, kind of like ihex but binary
- Xilinx CPLD tools use these instead of raw binaries
- Coolrunner-II JED files have comments!
  - These give (somewhat cryptic) hints as to which bits control which parts of the chip
  - But no details on the coding for the bits ☹

```
Note Block 0 *
Note Block 0 ZIA *
L000000 11111111*
L000008 11111111*
L000016 11111111*
L000024 11111111*
L000032 11111111*
L000040 11111111*
L000048 11111111*
L000056 11111111*
L000064 11111111*
L000072 01111110*
```


IOActive™

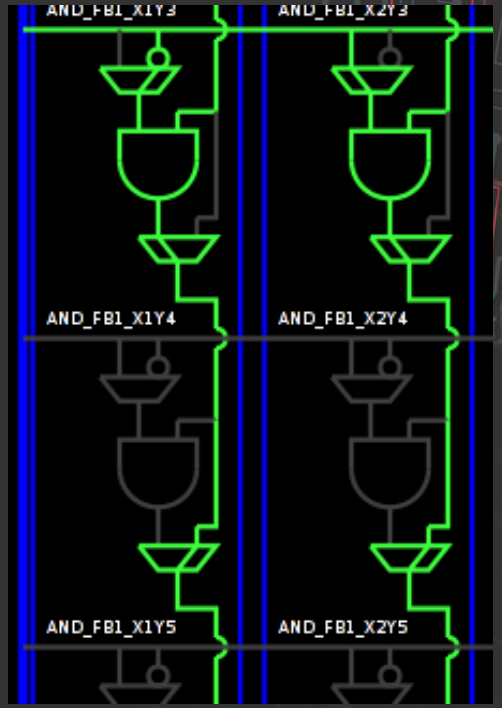
Sum of Products

- Canonical expression of digital equations
- (A & !B & C) | (!D & !E & F) | ...

IOActive™

SPLD architecture

- We can make *programmable* AND / OR gates!
  - [A|1] & [B|1] & [C|1] & [D|1] …
  - [A|0] | [B|0] | [C|0] | [D|0] …
- Grid of gates with a 2:1 mux at each input
  - Inputs on one axis
  - Outputs on the other
  - One SRAM cell per input
  - Logically cascaded but often implemented as tree



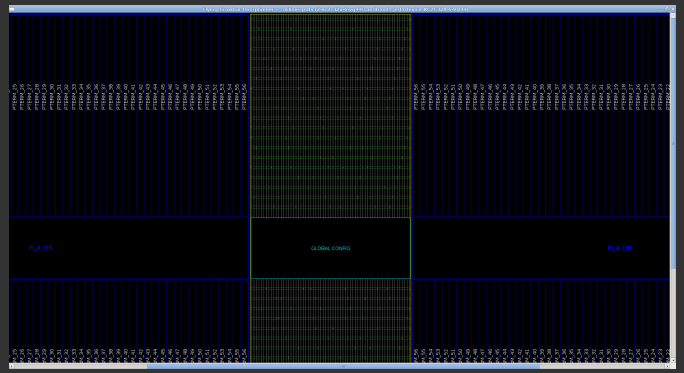
IOActive™

SPLD architecture

- Take N signals from FFs and input pins
- Inputs, plus complements, go to 2N x M AND array
- Product terms go to M x R OR array
- Sum terms go to FF or output pins

CPLD architecture

- SPLDs scale poorly, how to improve?
- Grid of SPLDs connected by a crossbar
  - Global bus with signals from all FFs and GPIOs
  - Pick a subset and feed to each SPLD



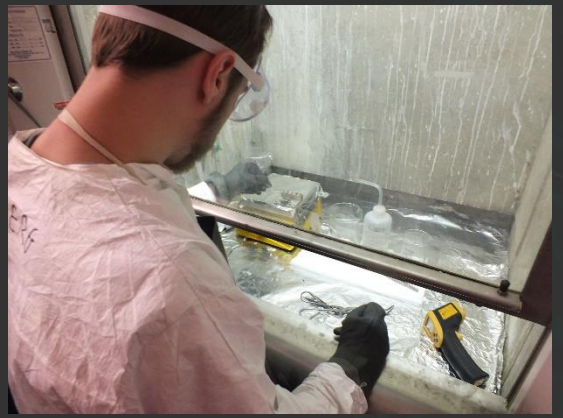
IOActive™

XC2C32A high-level arch

- 32 GPIOs + 1 input-only
- 2 function blocks (SPLDs)
  - 16 GPIOs and 16 FFs
  - 40 inputs from global routing, chosen from 65 available
  - 80x56 AND array
  - 56x16 OR array
  - Some product terms are special (covered later on)

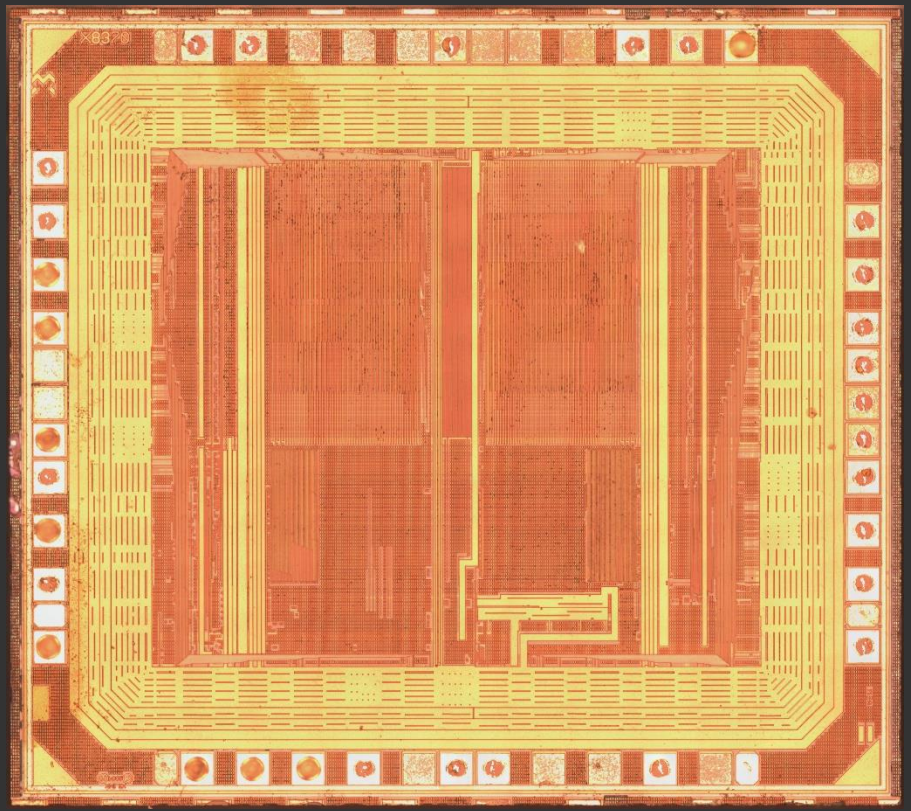
Time to put on our lab coats

- Decapping and imaging process has been beaten to death in lots of other talks so we won't cover it here
  - Check out siliconpr0n.org or my class for a refresher (http://security.cs.rpi.edu/courses/hwre-spring2014/)
- This talk is about high level RE, not lab techniques



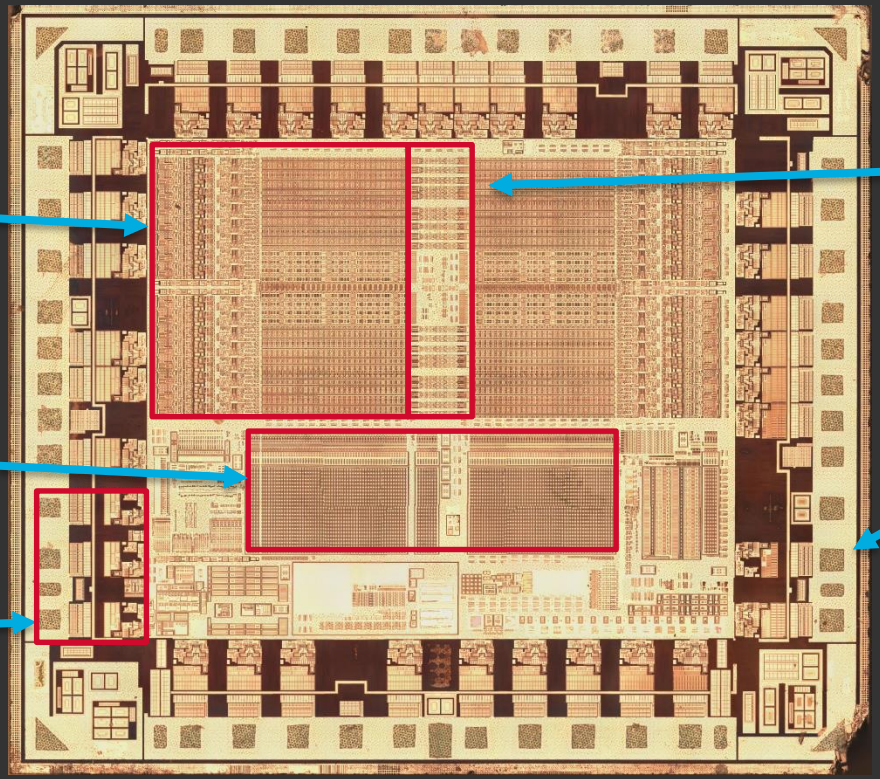
IOActive™

XC2C32A M4 overview



IOActive™

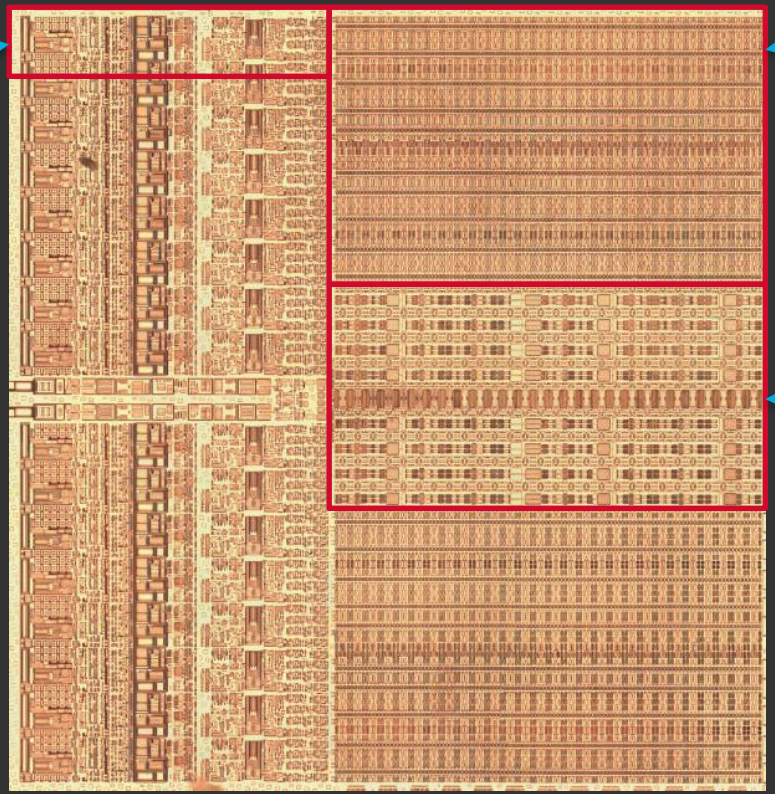
XC2C32A implant overview (Dash etch)



IOActive

Closeup of function block

Macrocell (1 of 16)



AND array (top half)

OR array

IOActive

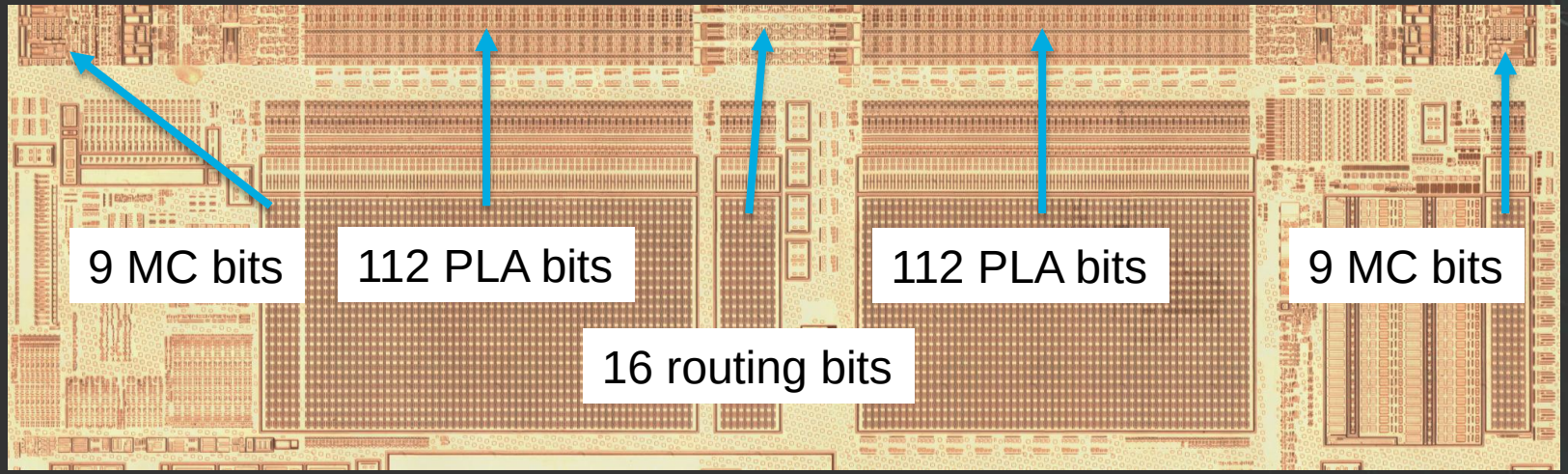
Config bit structure

- Some info in programming algorithm docs
  - Bit order in JED ≠ bit order on die (virtual addressing)
  - (48+1) rows x 260 cols (last row is config metadata)
  - 7 lock bits (0x2A = locked, 0x7F = unlocked)
  - 2 “done” bits (0x3 = blank chip, 0x2 = valid bitstream)
- 258 data columns + “transfer bits”
  - Leftmost and rightmost bit indicate “row valid”
- FF = blank chip so expect active-low for most stuff

IOActive™

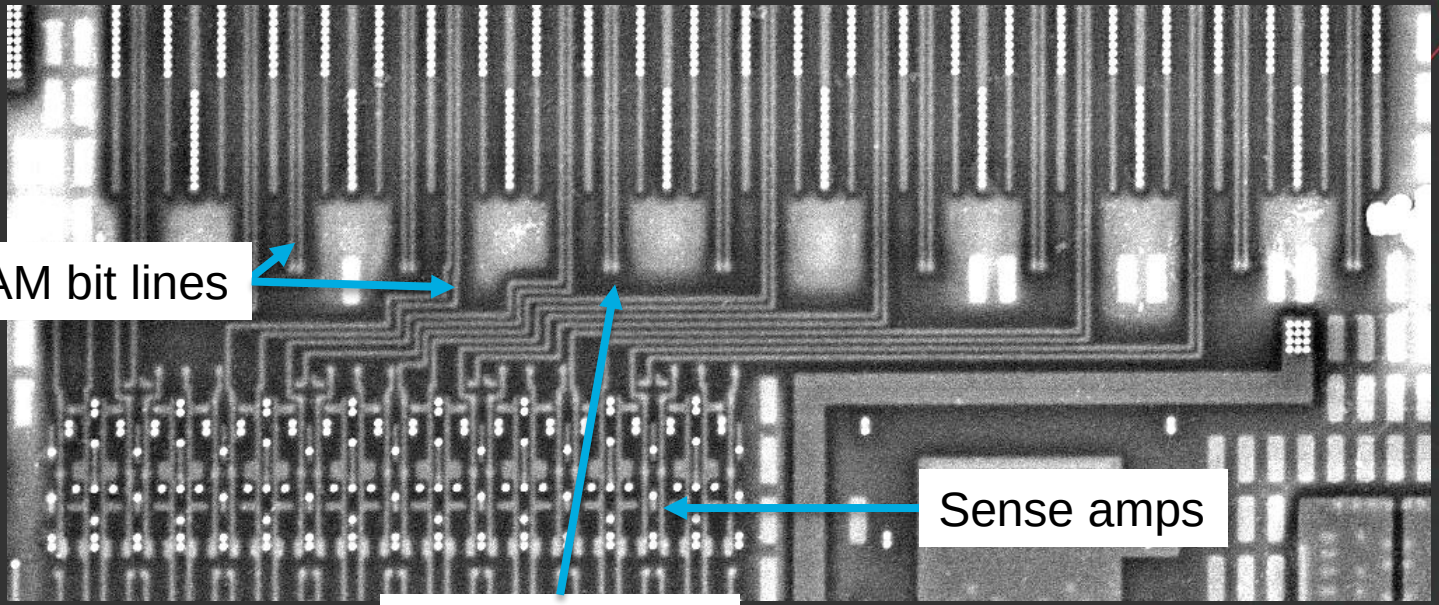
Config memory (implant)

Data flows up from EEPROM to SRAM during boot process



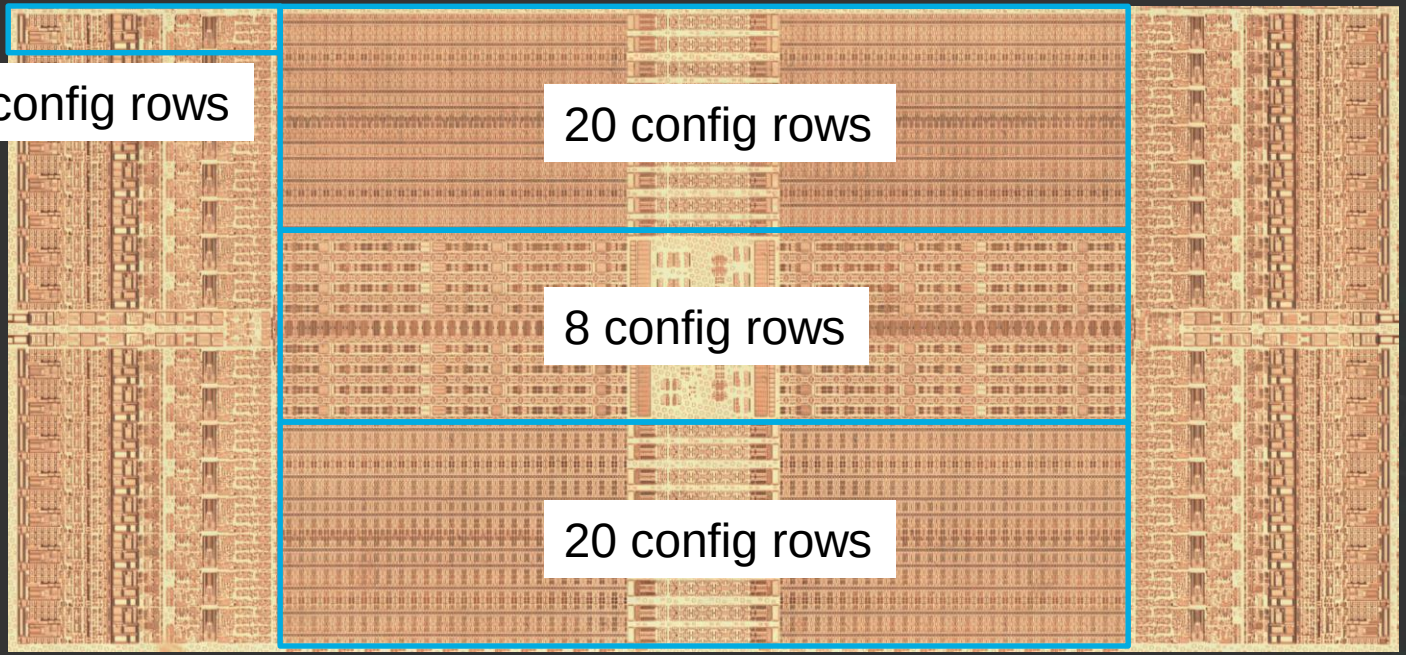
IOActive

Config memory (M2 near routing)



IOActive

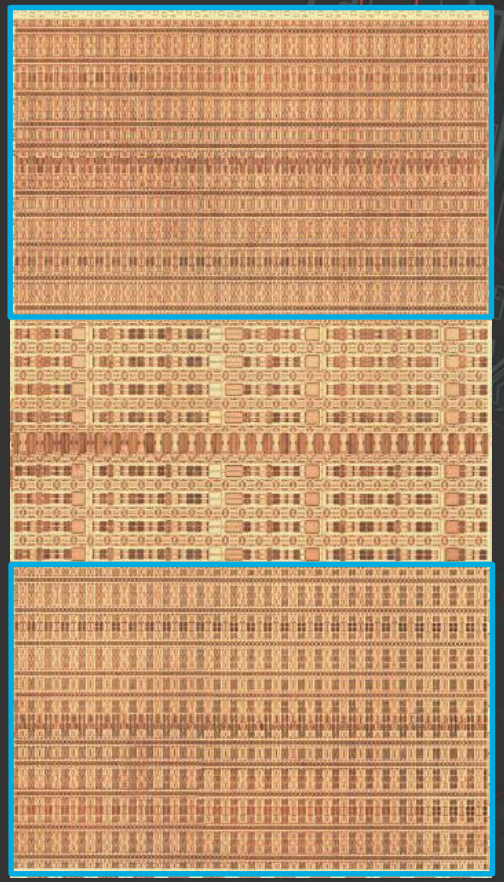
Main logic array



IOActive

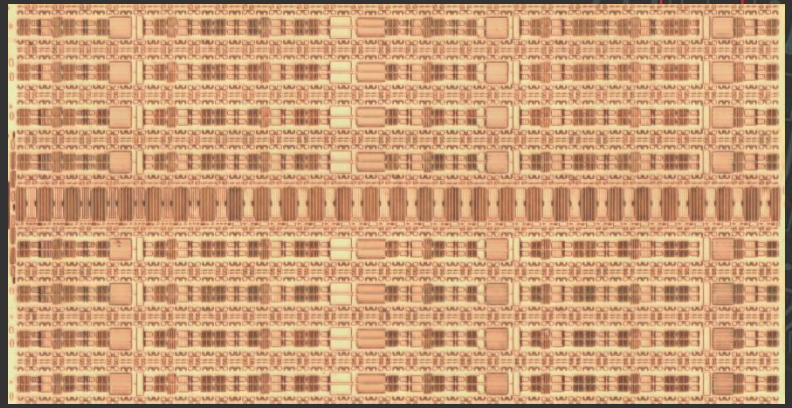
PLA AND array

- 56 pterms x 40 rows (+comps) flanking OR block
- Count config bits: 112 wide x 2 blocks x 20 rows
- Conclusions
  - Each row of config bits = 1 input from xbar
  - 2 bits per product term for X and !X
  - Empirical testing shows one-hot coding



IOActive™

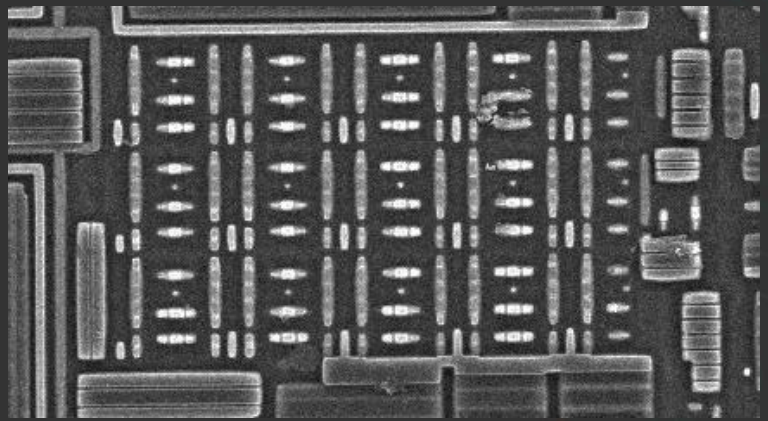
PLA OR array



- 56 AND terms x 16 OR outputs
- Count config bits: 112 wide x 8 high
- Conclusions
  - One bit to select a particular product term
  - Two OR terms interleaved in one config row

Macrocells

- 27 config bits per MC (9x3 grid*)
- Makes sense, EEPROM is 9 bits wide
- 3 rows * 16 MC = 48 rows (full height)
- Functionally partially RE’d
- These bits also control IOBs



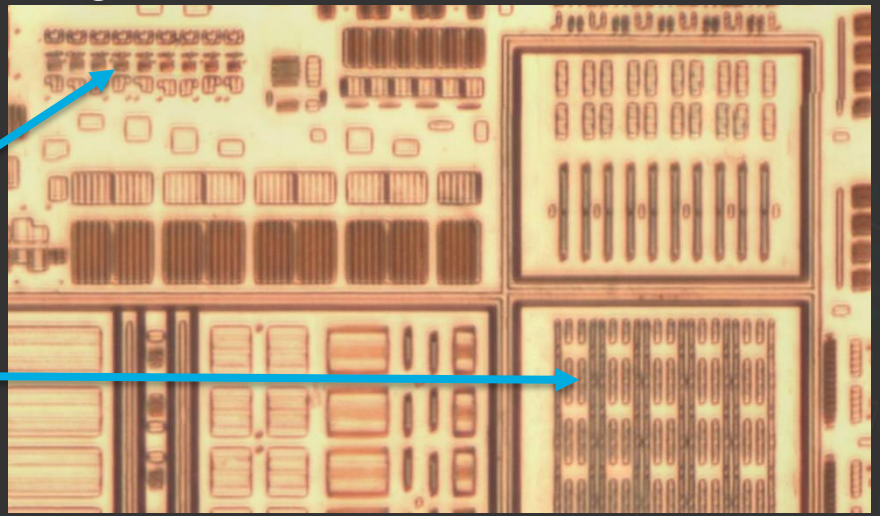
* Actually 9x2 + 9x1

Security bits

- Nine lock/done bits are in last row of config EEPROM
- Column position is same as right hand macrocells
- Coincidence? I think not!

Nine 6T SRAM cells ;)

Macrocell EEPROM

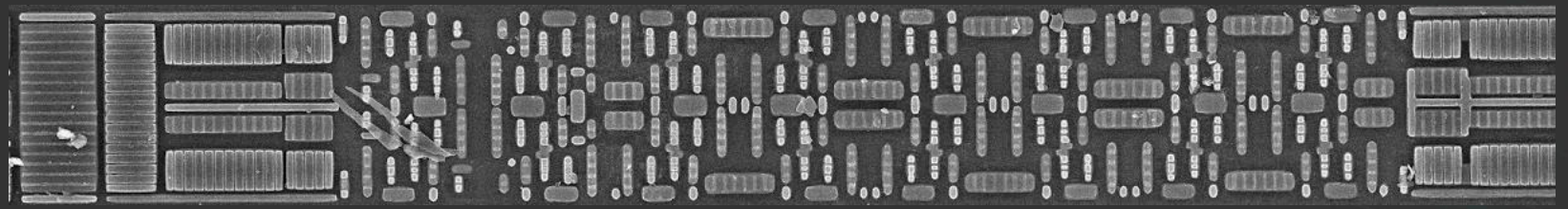


IOActive

Global routing

- Located between left and right AND array
  - 20 bits high x 16 bits wide (x 2 high for top/bottom half)
- Somehow selects 20 out of 65 signals to each FB
  - Half of each row selects a single left/right output
- But how do you do a 65:1 mux with 8 bits?
  - Dense coding makes no sense (would need 7 bits)
  - Obviously can’t be one-hot (would need 65 bits)
  - Datasheet is of no help here

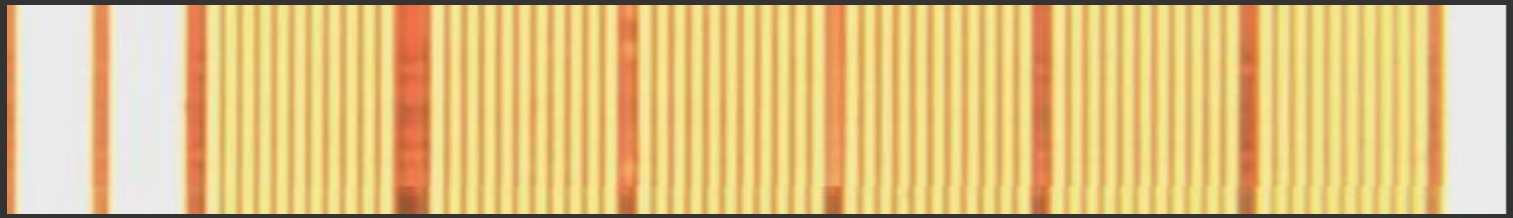
Global routing row – implant (Dash etch)



IOActive™

Global routing row – metal 4

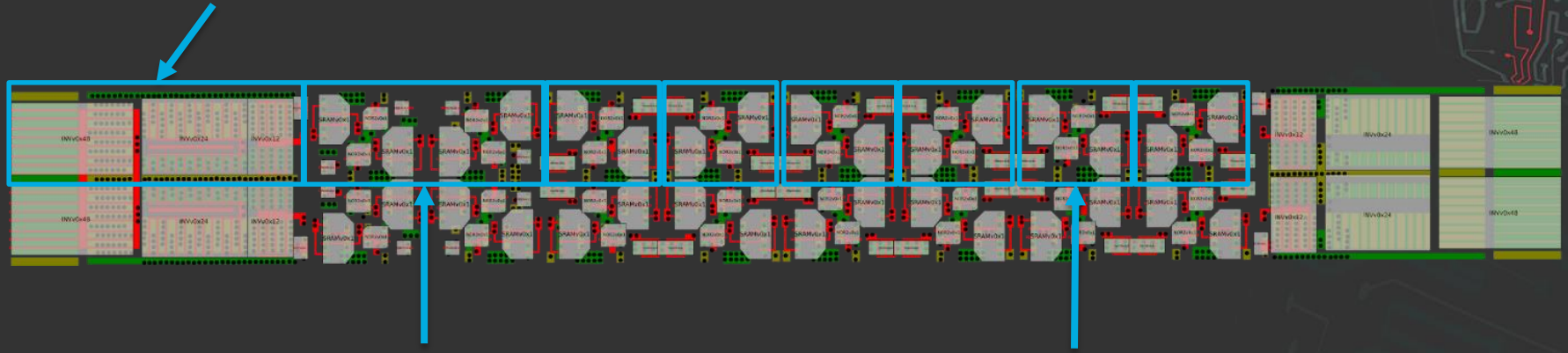
- Six groups of ~11 signals (rightmost is only 10)



IOActive™

Global routing row – vectorized

High-fanout inverter



These two are different

Six identical blocks

IOActive

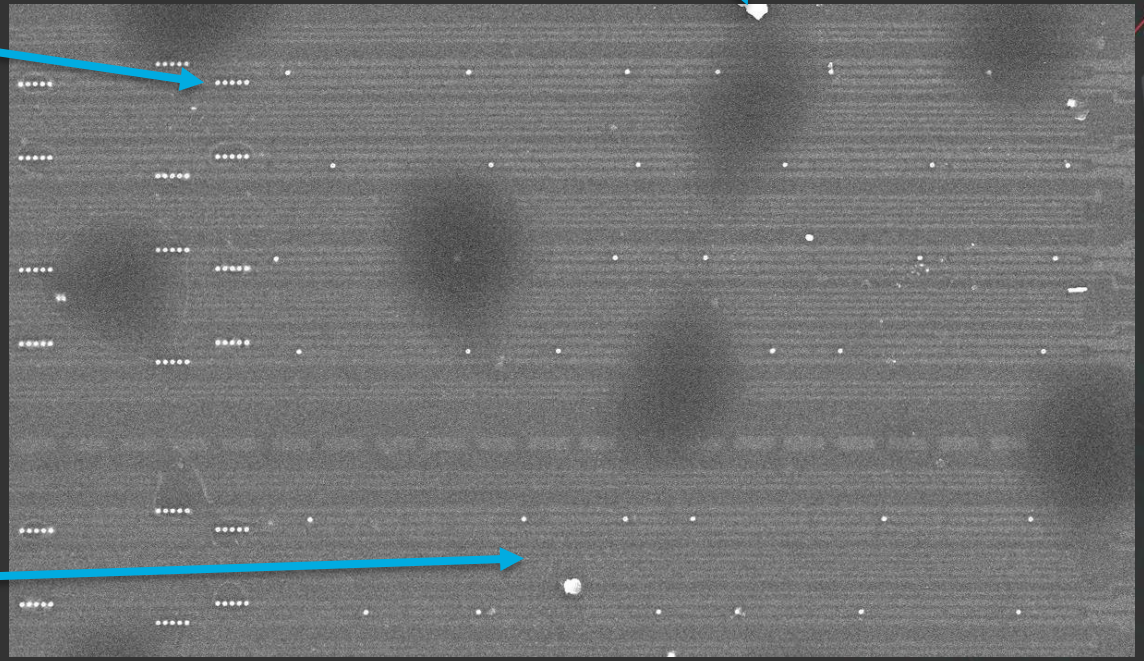
Thoughts so far

- Big driver at each side goes out to the PLA
  - Three-stage inverter (logically one NOT gate)
- Each of the eight blocks contains two SRAM cells
  - Maybe one for leftbound and one for rightbound?
- Six identical blocks, six groups of wires on M4
  - This seems promising…

Let’s take a look at M3…

Dust particle ☹

Power/ground vias



Six vias per row
But not identical!

IOActive™

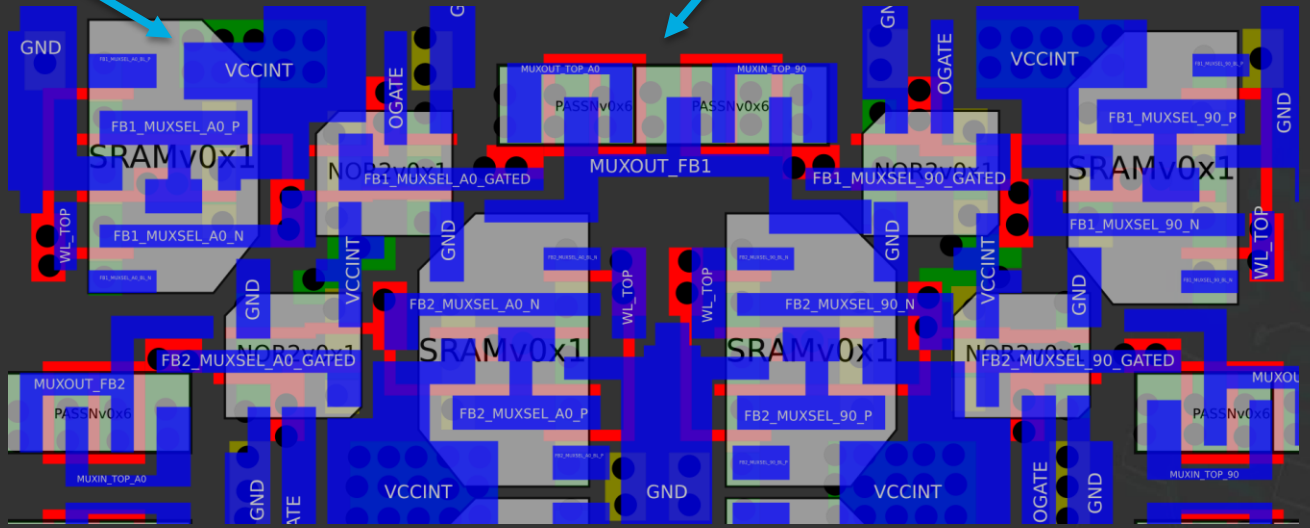
Examining the pattern

- Each row of the routing fabric has six M3-M4 vias
  - Exactly one via under each of the M4 bus groups
  - But in a different place for each row
- Now we're getting somewhere
  - Routing matrix is not a full crossbar!
  - Sparse crossbar with only 6 connections possible per row
  - Different combinations for each one

Looking back at poly/M1

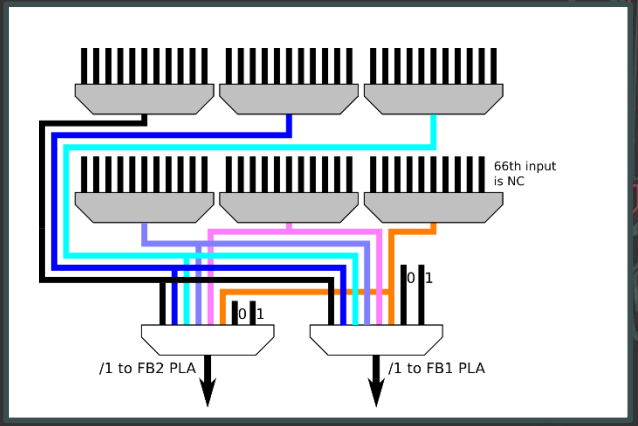
Config bit

Pass transistor



IOActive

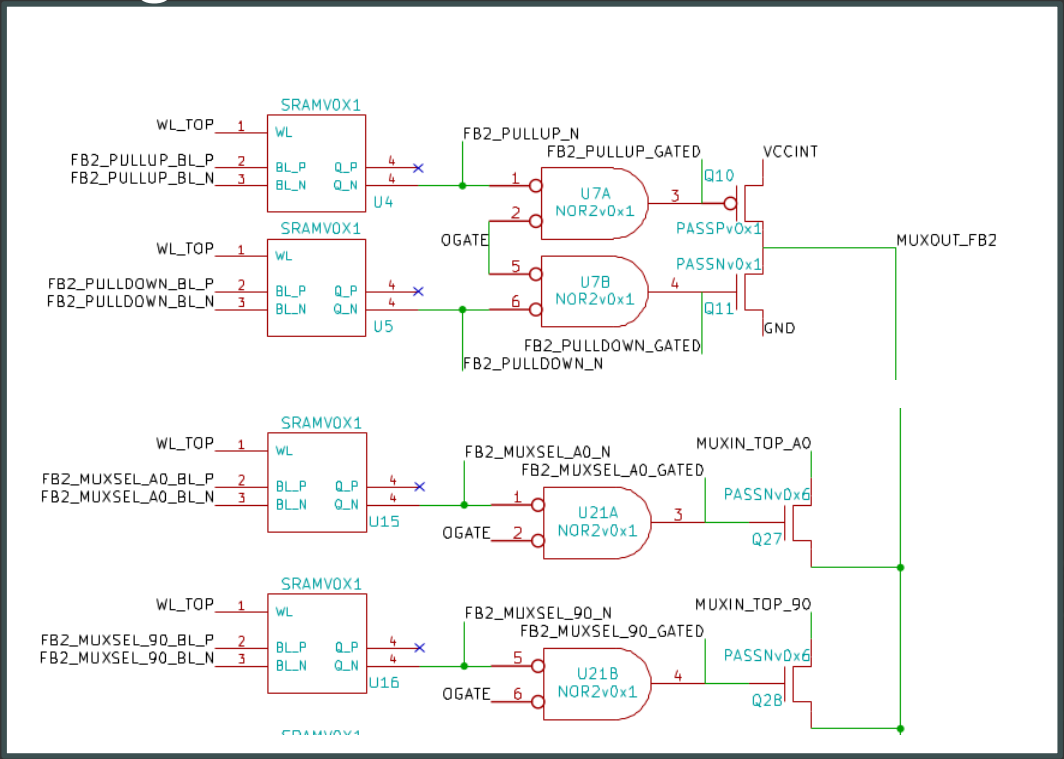
Putting it all together



- Each row is an 8:1 tristate bus mux
  - Constant zero (active low)
  - Constant one (active HIGH)
  - One signal from each group on M4 (6x active low)
- Global “OGATE” signal forces output high
  - NOR2s are to avoid bus fights between nets
- Rows are not identical, each one has different subset
  - Can use max-flow to assign signals to rows

IOActive™

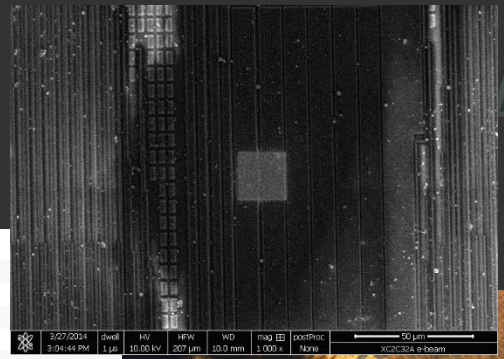
Global routing schematic

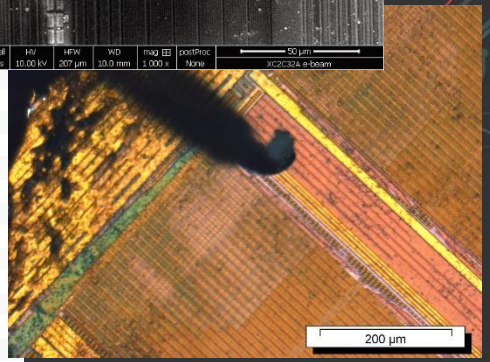


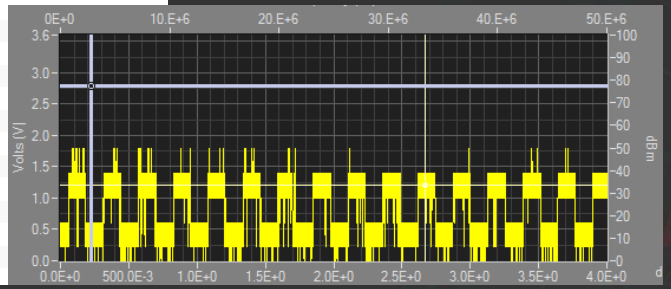
IOActive

Figuring out order of M4 bus


```
01 `timescale 1ns / 1ps
02 module test(clk_2048khz, led);
03
04  //Clock input
05  (* LOC = "P1" *) (* IOSTANDARD = "LVCMOS33" *)
06  input wire clk_2048khz;
07
08  //LED out
09  (* LOC = "P38" *) (* IOSTANDARD = "LVCMOS33" *)
10  output reg led = 0;
11
12  //Don't care where this is placed
13  reg[17:0] count = 0;
14  always @(posedge clk_2048khz)
15   count <= count + 1;
16
17  //Probe-able signal on FB2_5 FF at 2x the LED blink rate
18  (* LOC = "FB2_5" *) reg toggle_pending = 0;
19  always @(posedge clk_2048khz) begin
20   if(count == 0)
21    toggle_pending <= !toggle_pending;
22  end
23
24  //Blink the LED
25  always @(posedge clk_2048khz) begin
26   if(toggle_pending && (count == 0))
27    led <= !led;
28  end
29
30 endmodule
```







IOActive™

A few last bits on layout

- M1/poly = gates
- M2 = local vert routing, SRAM BL, OGATE
- M3 = local horz routing, SRAM WL, mux outputs
- M4 = input bus. Bit ordering L-R:
  - FB1 GPIOs
  - Global input
  - FB2 GPIOs
  - FB1 FFs
  - FB2 FFs

Full PLA structure

- We now know almost enough to configure the PLA
- One last gotcha – most pterms are dual purpose!
  - All can be used as general purpose logic
  - 48 have special connections to macrocells (3 each)
    - Used for set/reset, clock enable, etc
  - 4 have special connections to the whole FB
    - Use these to avoid use of lots of per-MC PTs
  - 4 appear to have no special use
  - Datasheet describes functions, but not which is which

Per-FB product terms

- Control Term Clock (CTC)
  - Per-FB clock (kinda like BUFH)
- Control Term Reset (CTR)
  - Per-FB reset
- Control Term Set (CTS)
  - Per-FB set
- Control Term Enable (CTE)
  - Per-FB output enable (for parallel buses etc)

Per-macrocell product terms

- Product Term A (PTA)
  - Can be used as FF set/reset
- Product Term B (PTB)
  - Can be used as IOB output enable
- Product Term C
  - Can be used as FF clock
  - Must be used as CE for DFFCE cells
  - Goes to macrocell XOR gate (see next slide)

Macrocell XOR gate

- Output from the PLA doesn't go directly to FF/IOB
- First, PLA output is XORed with…
  - Product Term C (or its complement)
  - Constant 1
  - Constant 0
- This allows for efficient adders, plus bypassing the OR array for slight speedup
- So… which product term is PTC?

Identifying PTC

- We can configure the PLA exactly as we want
- Copy black-box MC config bits from ISE bitstream that is known to use PTC
- Scan one PT at a time until we hit it
- Turns out, PTC for cell N is term 3N + 10

Identifying other pterms

- Still working on this
- Probably PTB, PTA are either
  - PTC+1,PTC+2
  - PTC-1,PTC-2
- And CT* are likely among the first few terms
- Too busy with “real work” to test this yet ☹

IOActive™

Global config bits

- There are 22 global config bits at middle of die
- 3 of them control I/O bank Vt
  - One global bit (XC2C32 backward compat)
  - One per-bank bit (used for 32A bitstreams)
- Rest control other misc stuff
  - Global clock (3), set/reset (2), output enable (8 bits)
  - IOB termination mode (1 bit, pullup/keeper)
  - Input-only pin config (2 bits)
  - Coding of most unknown as of now

libcrowbar

- BSD-licensed library for manipulating CR-II bitstreams
- Needs a lot of work and refactoring, but (kinda) works
- Only supports the 2c32a for now
- http://redmine.drawersteak.com/projects/achd-soc/repository/show/trunk/src/crowbar

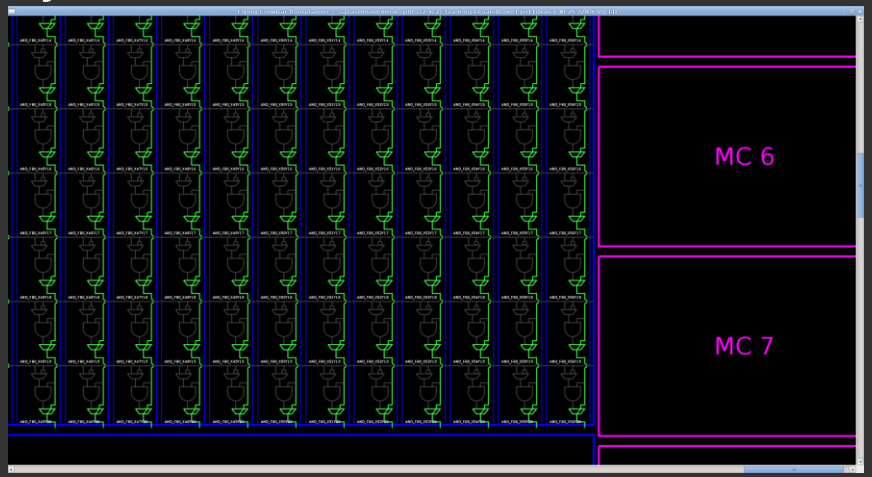
Libcrowbar usage example

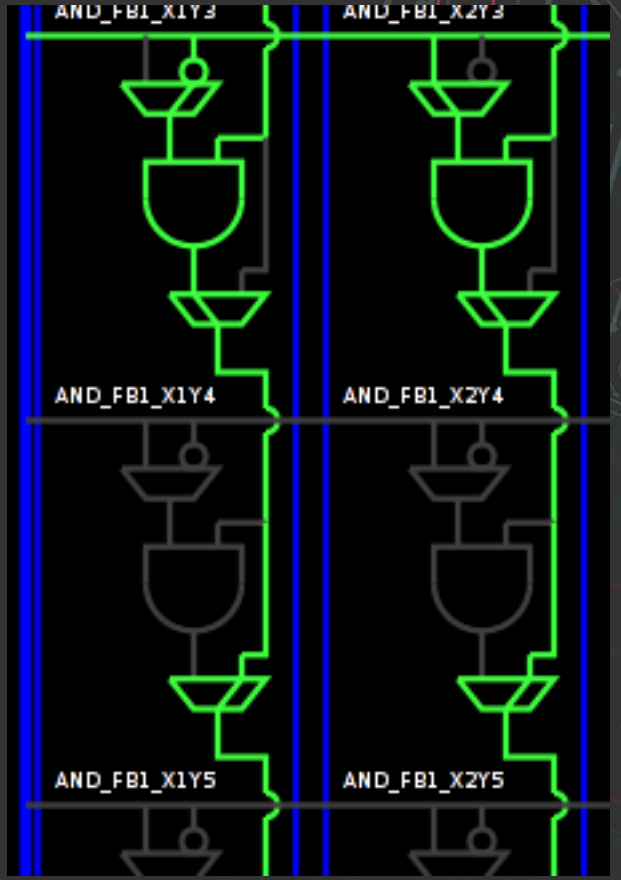
```
//Get the IOBs
FCCoolRunnerIIIOB* iob_gpiol0 = static_cast<FCCoolRunnerIIIOB*>(device.GetIOBForPin("P8"));
FCCoolRunnerIIIOB* iob_gpiol1 = static_cast<FCCoolRunnerIIIOB*>(device.GetIOBForPin("P6"));
FCCoolRunnerIIIOB* iob_led1 = static_cast<FCCoolRunnerIIIOB*>(device.GetIOBForPin("P38"));
FCCoolRunnerIIIOB* iob_led2 = static_cast<FCCoolRunnerIIIOB*>(device.GetIOBForPin("P37"));
printf("IOB assignment\n");
printf("    IOB for FTDI_GPIOL0 is at %s\n", iob_gpiol0->GetNamePrefix().c_str());
printf("    IOB for FTDI_GPIOL1 is at %s\n", iob_gpiol1->GetNamePrefix().c_str());
printf("    IOB for LED1 is at %s\n", iob_led1->GetNamePrefix().c_str());
printf("    IOB for LED2 is at %s\n", iob_led2->GetNamePrefix().c_str());

//GPIOL0: LVCMOS33 input
iob_gpiol0->m_iostandard = FCCoolRunnerIIIOB::LVCMOS33;
iob_gpiol0->SetInZ(FCCoolRunnerIIIOB::INZ_INPUT);
iob_gpiol0->SetOE(FCCoolRunnerIIIOB::OE_FLOAT);
iob_gpiol0->SetTerminationMode(FCCoolRunnerIIIOB::TERM_FLOAT);
iob_gpiol0->m_bSchmittTriggerEnabled = false;

//GPIOL1: unregistered LVCMOS33 output
iob_gpiol1->m_iostandard = FCCoolRunnerIIIOB::LVCMOS33;
iob_gpiol1->SetInZ(FCCoolRunnerIIIOB::INZ_FLOAT);
iob_gpiol1->SetOE(FCCoolRunnerIIIOB::OE_OUTPUT);
iob_gpiol1->m_outmode = FCCoolRunnerIIIOB::OUTPUT_DIRECT;
iob_gpiol1->SetTerminationMode(FCCoolRunnerIIIOB::TERM_FLOAT);
iob_gpiol1->SetSlewRate(FCCoolRunnerIIIOB::SLEW_FAST);
iob_gpiol1->m_bSchmittTriggerEnabled = false;
```


IOActive™

fcplan

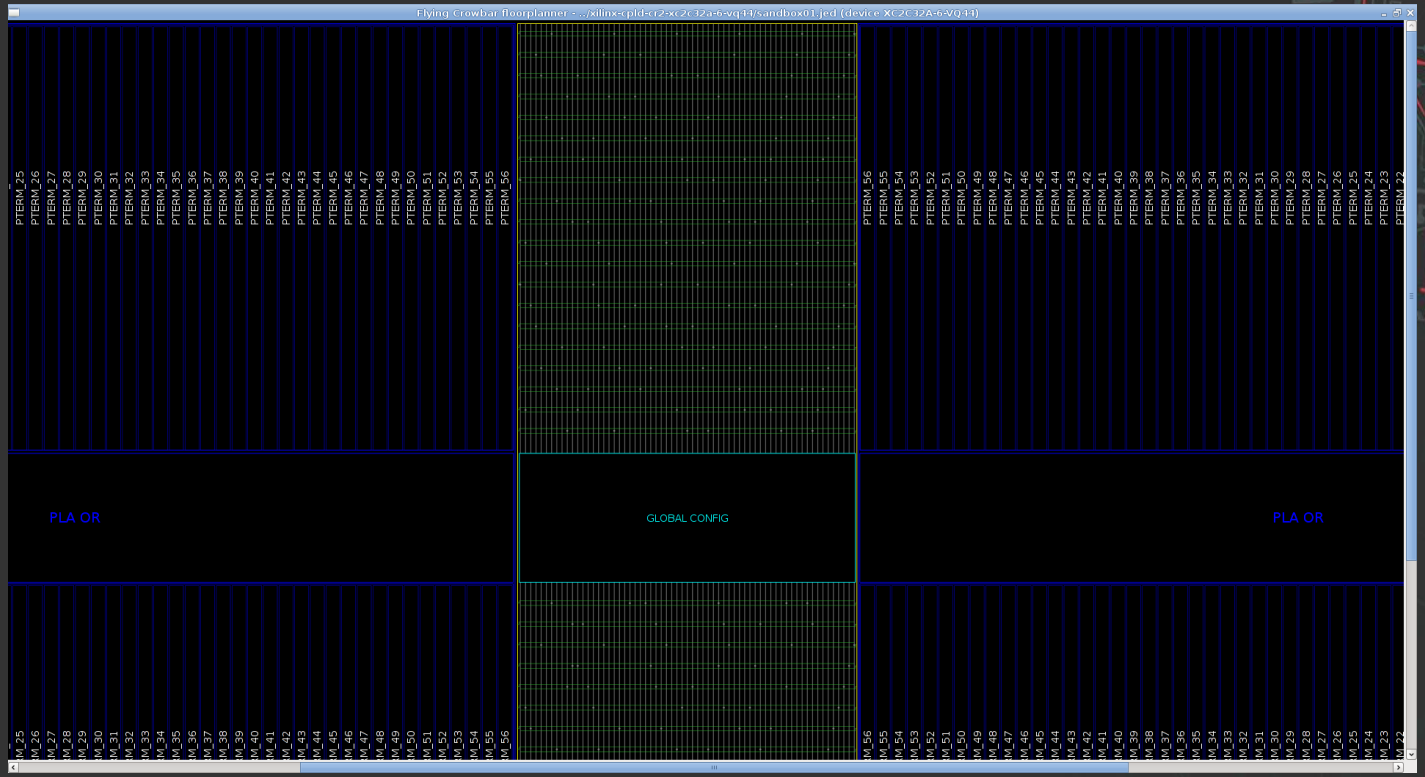
- Floorplanner / physical layout viewer
- Currently supports AND array and global routing only





IOActive™

fcplan



IOActive™

Acknowledgements

- John McMaster (siliconpr0n)
  - CNC microscopy, sample prep
- Ray Dove (RPI)
  - SEM/FIB access and training
- Bryant Colwill (RPI)
  - Cleanroom/microprobe access and training
- Siliconpr0n.org team
  - Lots of helpful advice and feedback

IOActive™

Live demo

Future work (RE)

- Figure out remaining special product terms
- Figure out rest of macrocell bits
- Figure out rest of global bits
- Support larger devices
  - Routing matrix via ROM varies across device densities
  - New macrocell features in ≥ 128 MC devices

Future work (toolchain)

- Add support for OR array etc to fcplan
- Yosys -> libcrowbar bridge for RTL synthesis
- Improve decompiler
  - Higher-level structure extraction (adders, etc)

Questions?

IOActive™