CSCI 4974 / 6974
Hardware Reverse Engineering

Lecture 19: FPGA architecture

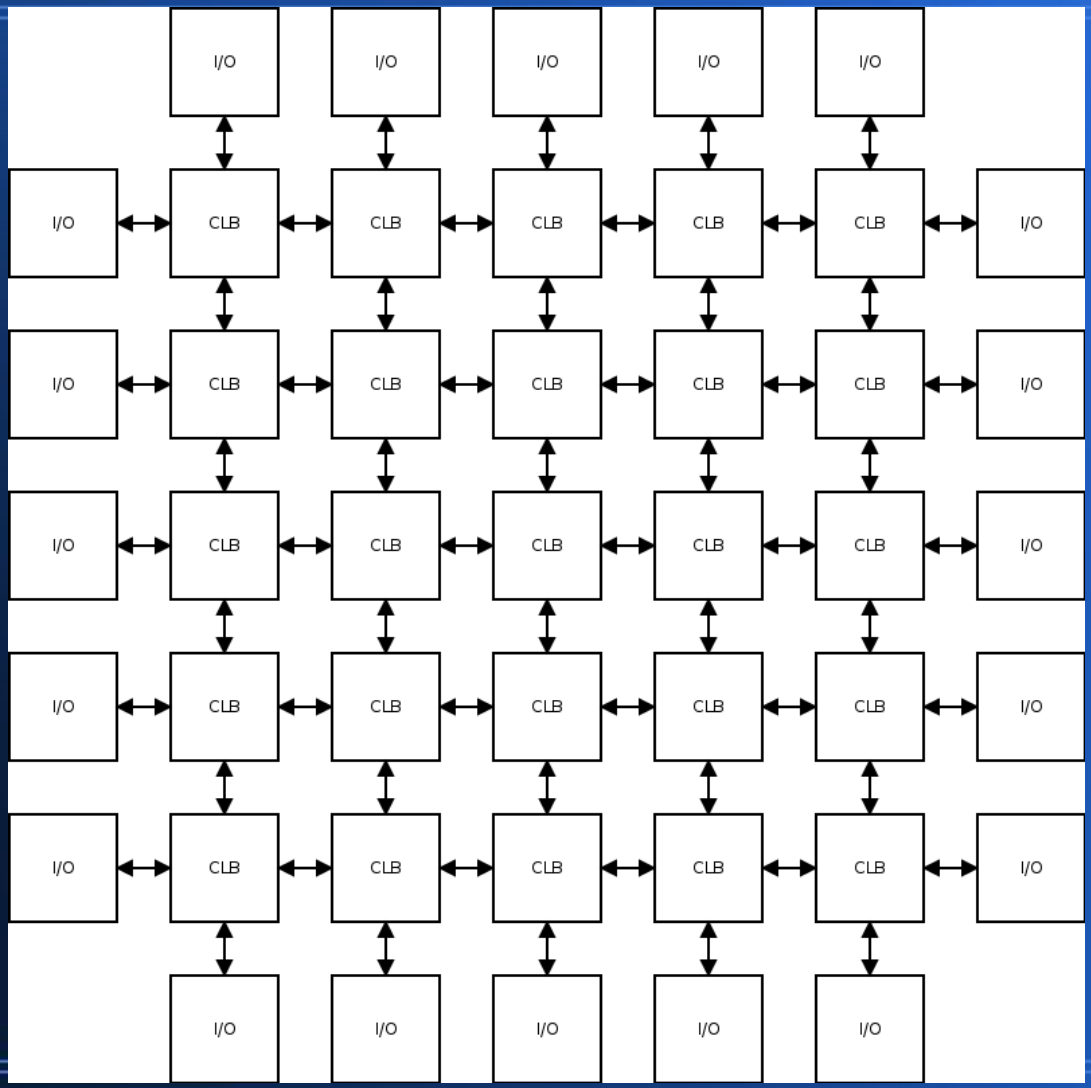
What is an FPGA?

- Programmable logic like a CPLD
- 2D array topology
  - O(n) interconnect scaling vs $O(n^2)$ of a CPLD
- Basic building block is a lookup table (LUT)

Today's lecture

- Overview of generic FPGA architecture
- In-depth discussion of two Xilinx architectures
  - Spartan-3A
  - Spartan-6
- High-level Xilinx FPGA bitstream structure
- A few die shots but no in-depth analysis
  - Once I finish my CPLD work all bets are off ;)

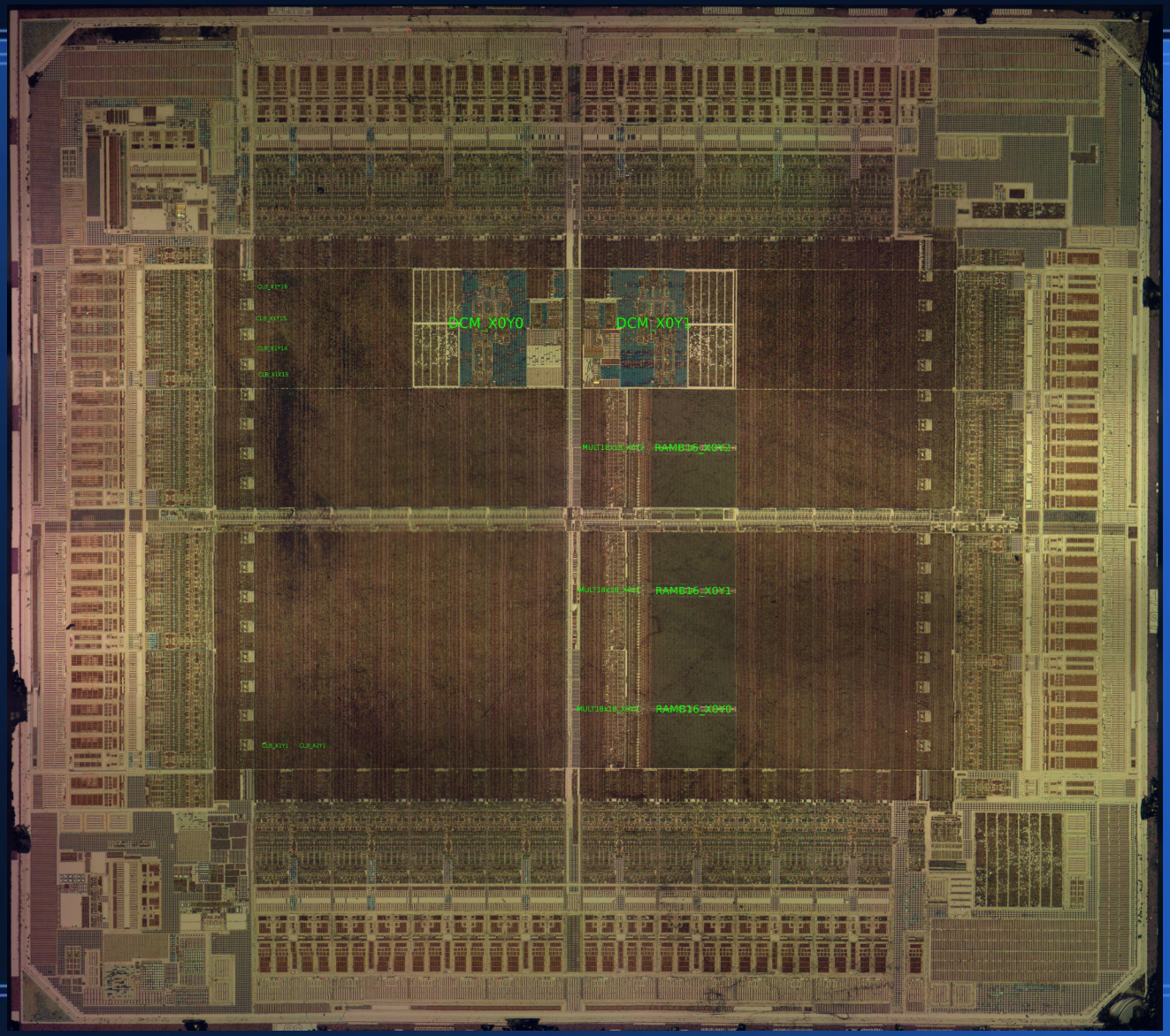
High-level overview



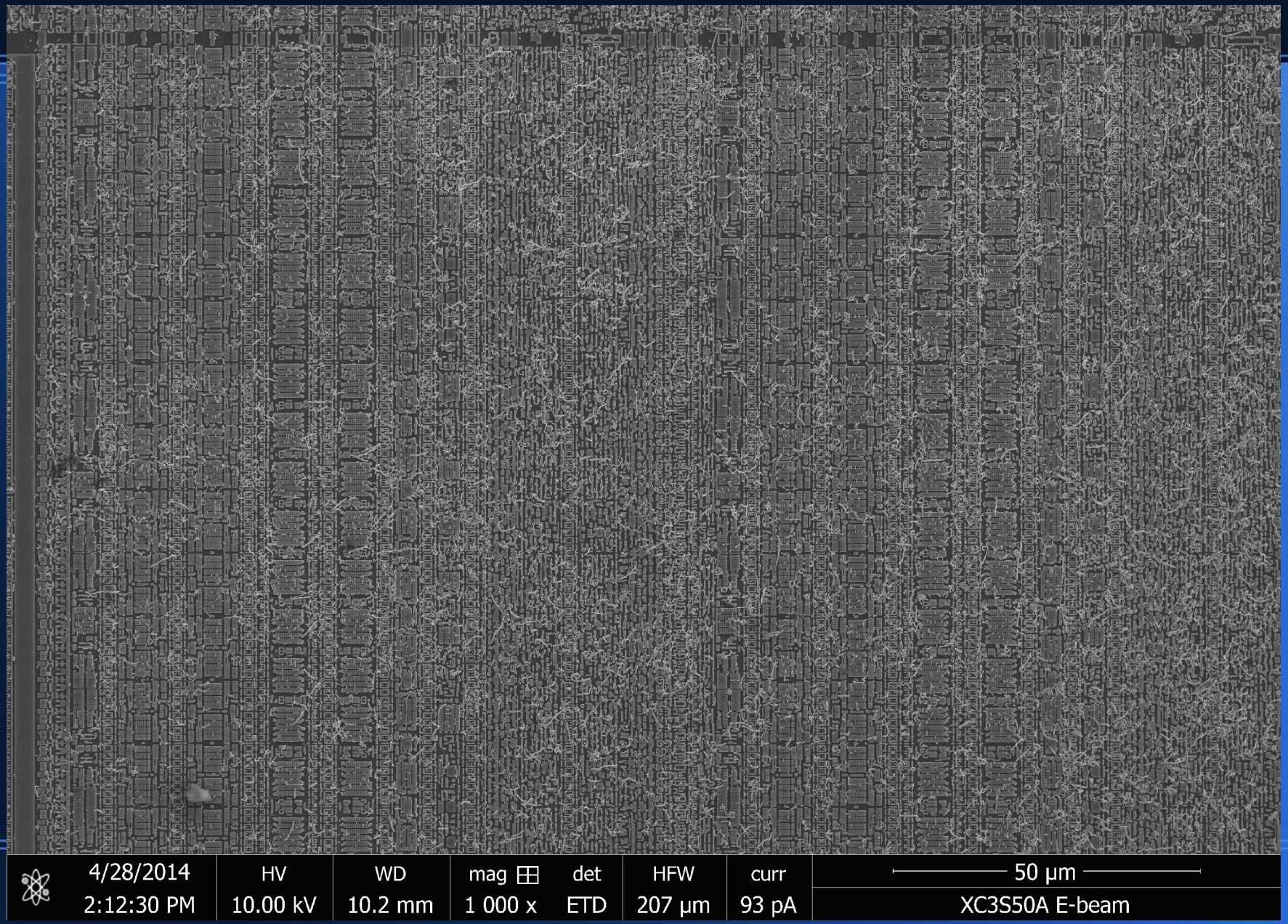
Configurable Logic Block (CLB)

- Contains one or more *slices* and a switch box
- Number of slices varies by device family
  - Spartan-3A has four slices per CLB
  - Spartan-6 has two slices per CLB
- Slices implement actual logic
- Switch box is a crossbar
  - Route signals from slice I/Os to other CLBs
  - Forward multi-hop signals to the next CLB

Xilinx XC3S50A (16 x 12 CLBs)



Spartan-3A CLB (active layer)



Slice

- One or more lookup tables (LUTs)
- One or more D flipflops

Lookup table (LUT)

- Basic building block for combinatorial logic
- Small async SRAM array (16-64 bits)
- Load SRAM with truth table
- Feed inputs to address lines
- Use output in other logic
- Typically 4-6 inputs and 1-2 outputs

Xilinx Spartan-3A slices

- SLICEL (logic slice), two per CLB
  - Two 4:1 LUTs
  - Ripple-carry generation logic for adders
  - Two 2:1 muxes for making wide-input functions
  - Two D flipflops
- SLICEM (memory slice), two per CLB
  - All features of SLICEL
  - Writable LUTs, can be 16-bit shift reg or SRAM

Xilinx Spartan-6 slices

- Three types, but only two slices per CLB
- CLBs alternate in columns
  - SLICEX + SLICEL
  - SLICEX + SLICEM

SLICEX (basic logic)

- One in every CLB
- Four LUTs, configurable as 6:1 or 5:2
- Eight D flipflops

SLICEL (logic)

- One in each SLICEL+X CLB
- All features of SLICEX
- Three 2:1 muxes
- Ripple-carry chain

SLICEM (memory)

- One in each SLICEM+X CLB
- All features of SLICEL and SLICEX
- Can also use LUTs as:
  - 64-bit single port SRAM
  - Merge two LUTs to make 64-bit dual port
  - 32-bit shift register

Block RAM

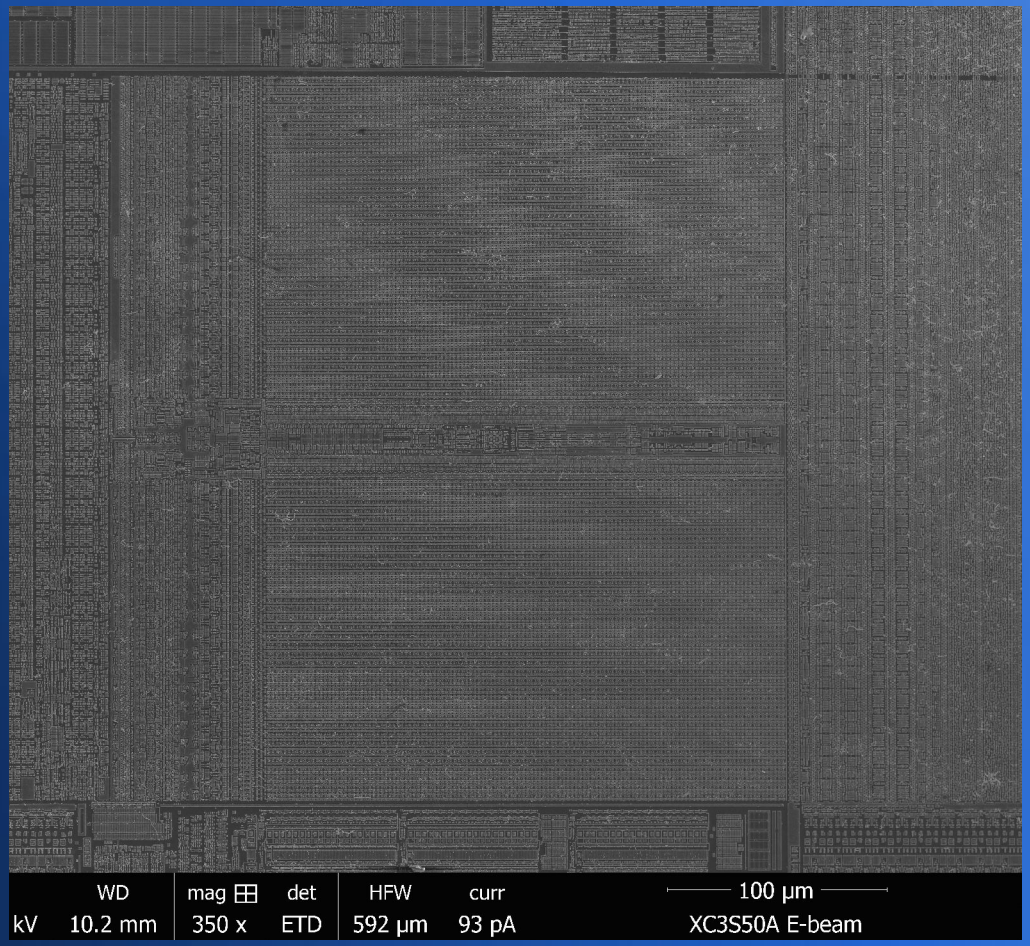
- LUT-based RAM is asynchronous and distributed across the chip
- But uses a lot of die area
- Bulk synchronous SRAM is more area-efficient

Spartan-3A block RAM

- 18kbit dual port synchronous SRAM
- Variable topology
  - 16k x 1 up to 512 x 36
  - Optional parity bit (no parity gen/check logic)
- Four CLBs tall
- Some routing resources shared with multipliers
- Memory can be initialized to arbitrary values

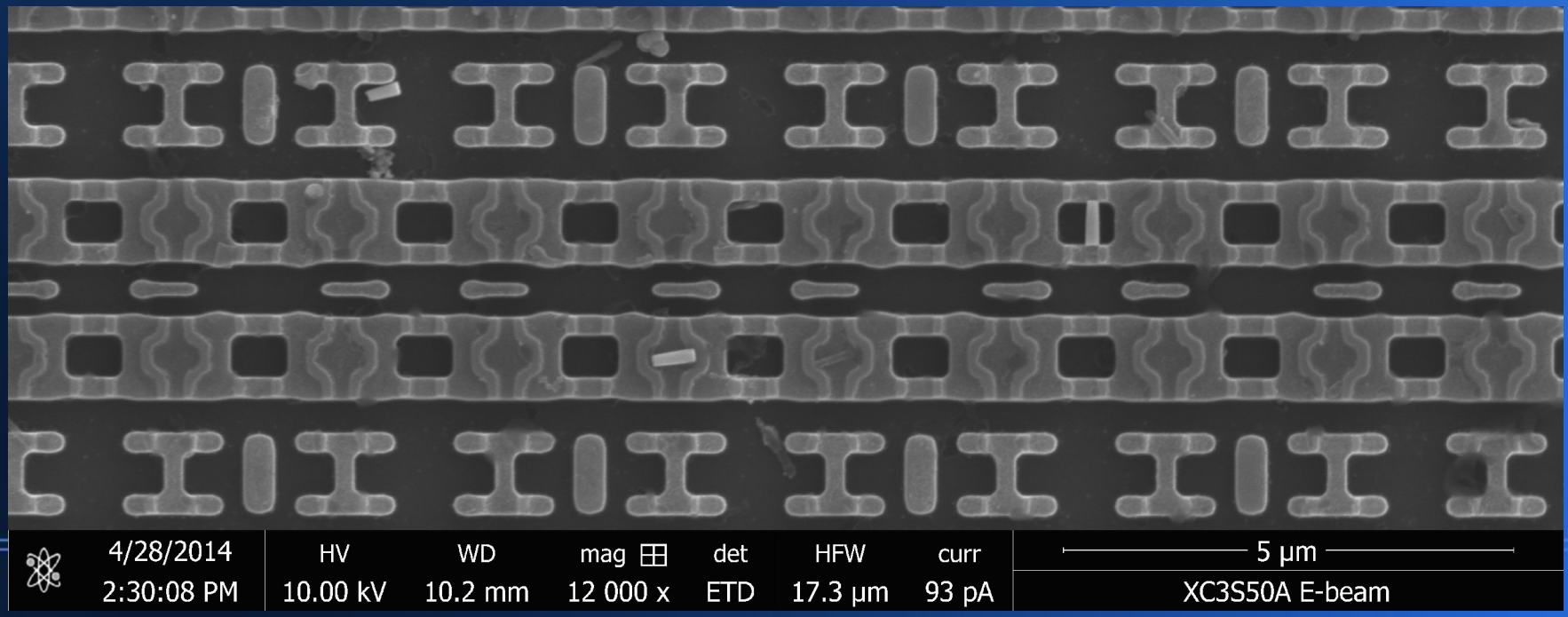
Spartan-3A block RAM

- 2 blocks x
  72 rows x
  128 cols
  = 18kbits



Spartan-3A block RAM

- 8T dual port SRAM (1.84 x 2.07 µm = 3.8 µm^2)
- Inverters between PMOS “H” and NMOS area
- Two sets of access transistors, one per port



Spartan-6 block RAM

- Based on Spartan-3A block RAM
- Can be split into two 9kbit blocks
  - Some silicon bugs related to these :(
- Separate routing, does not compete with mults

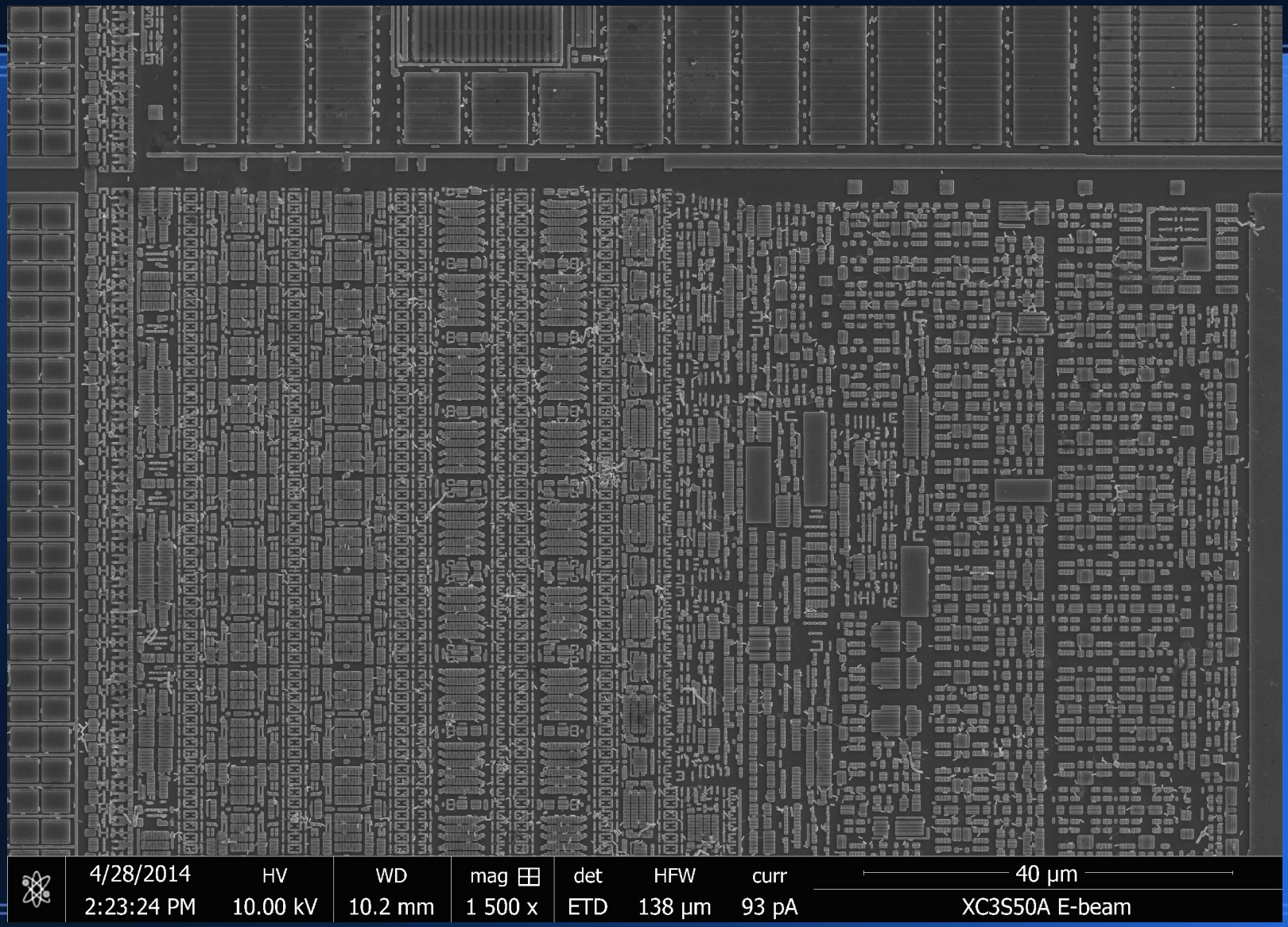
Multipliers

- FPGAs are commonly used for DSP stuff
- This means lots of multiply-add operations
- Multipliers need lots of LUTs
- Hard-wired ones save die area and run faster

Spartan-3A MULT18x18

- 18 x 18 => 36 bit twos complement multiplier
- Pipeline registers on inputs and outputs
- Cascadable to handle larger numbers

Spartan-3A MULT18x18



Spartan-6 DSP48A1

- 18 + 18 => 18 bit pre-adder
- 18 x 18 => 36 bit twos complement multiplier
- 36 + 48 => 48 bit adder/accumulator
- Cascadable to handle larger numbers
- Four stages of pipelining

SERDES

- FPGA logic is slow but can easily parallelize
- I/O pin counts are limited
- Parallel data on chip => serial data at pins

GPIO SERDES

- Every I/O pin on most modern FPGAs can do basic parallel/serial conversion
- Spartan-3A is 1:1 (normal) or 2:1 (DDR) only
- Spartan-6 can do
  - Up to 4:1 in one IOB for single-ended
  - Cascade both IOBs for up to 8:1 on differential

High-speed SERDES

- GPIO SERDES have to be small to fit in every I/O cell. This limits their capabilities
- Typically not super fast (~1 GHz max)
- Getting faster requires more advanced features that take up die area

High-speed SERDES

- Dedicated transceivers (Spartan-6 LXT etc) are on specific pins not usable as GPIO and often have separate power rails
- Much higher max speeds possible (3 Gbps in Spartan-6, up to 30+ Gbps in Virtex-7)
- Lots more features (pre-emphasis, clock recovery PLLs, higher serialization rates up to 30:1 or more)

Yield enhancement

- FPGAs are large dies and often push fab limits
- Yields are typically poor, especially to start
- What if we could still use chips with some defects?
- Chip vendors are very tight-lipped about these techniques, much of this section is speculation

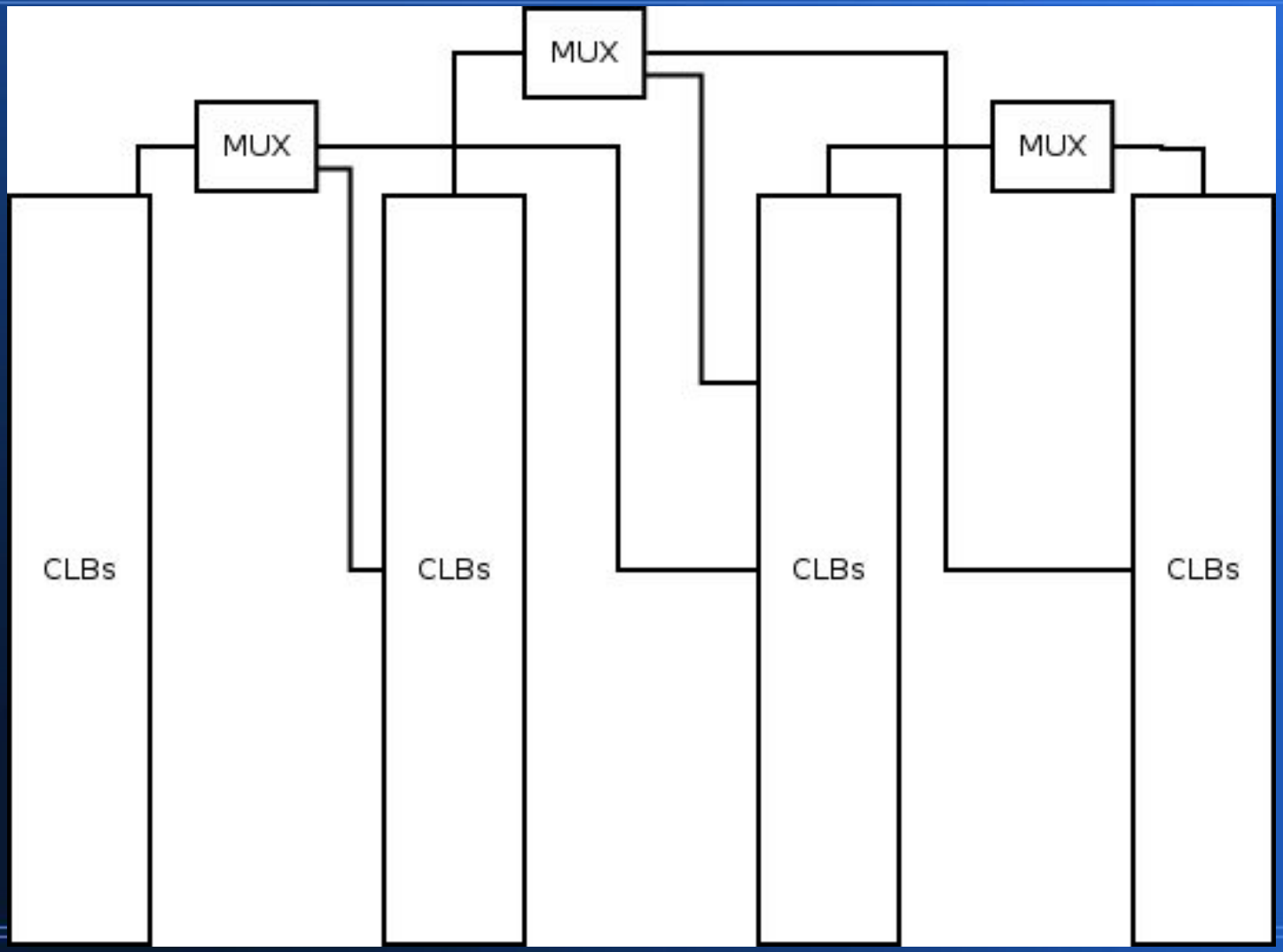
Yield enhancement

- Transceivers are very sensitive to fab issues
- Make another version of the chip with no transceivers and sell the bad ones!
- Evidence:
  - Spartan-6 LXT has unbonded I/O pads
  - Spartan-6 LX has no SERDES and extra GPIOs, plus SERDES-sized hole in CLB array
  - Bitstream size and static power are identical

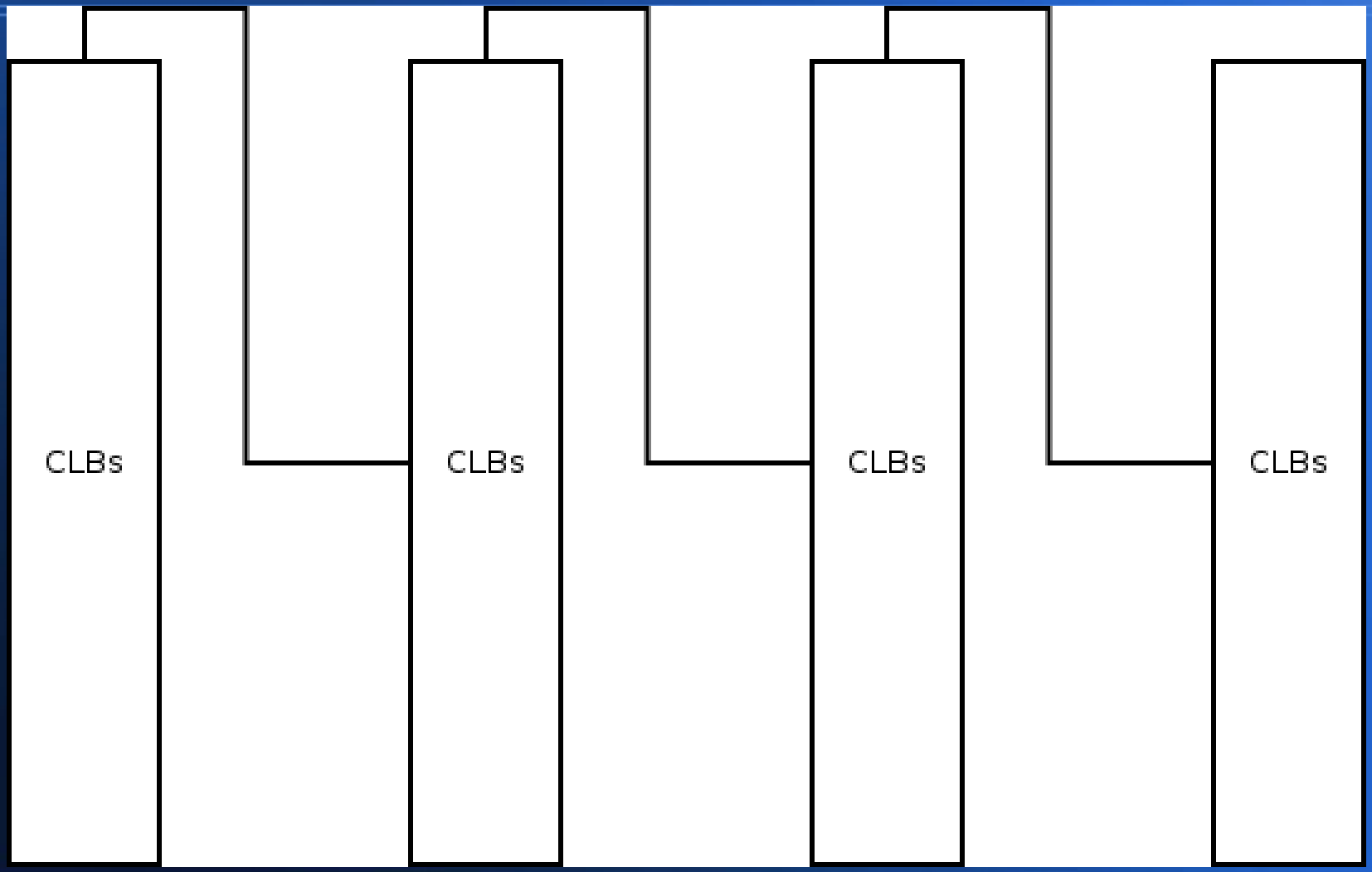
Yield enhancement

- Divide rows or columns of logic into groups
- Allow up to one unit per group to fail
- XC7A75T may be binned XC7A100T
  - Exactly 3/4 the BRAM/DSP
  - Almost exactly 3/4 the CLBs
  - Same static power
  - Same bitstream length
  - JTAG IDCODE has two bits swapped

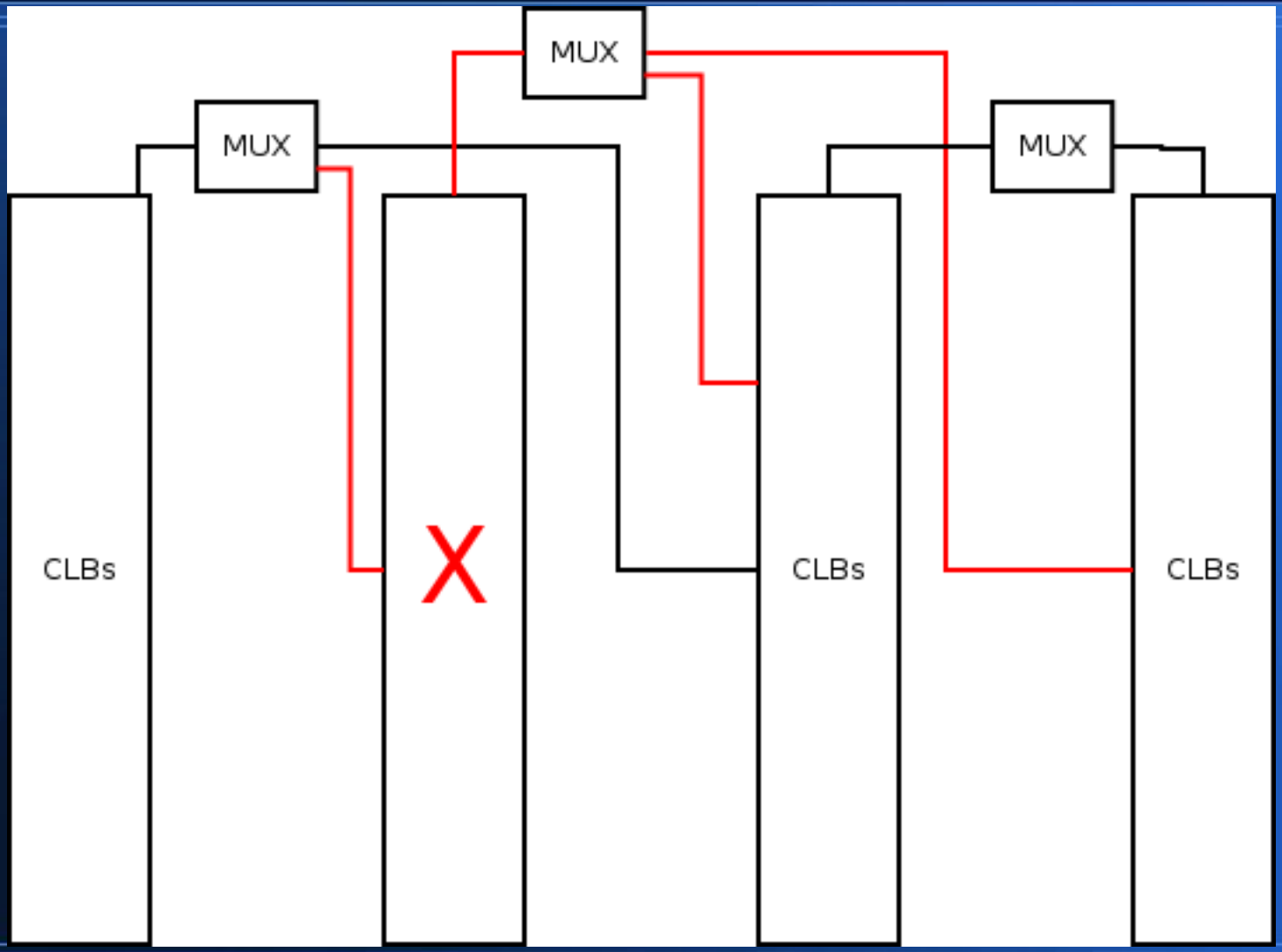
No bad CLBs



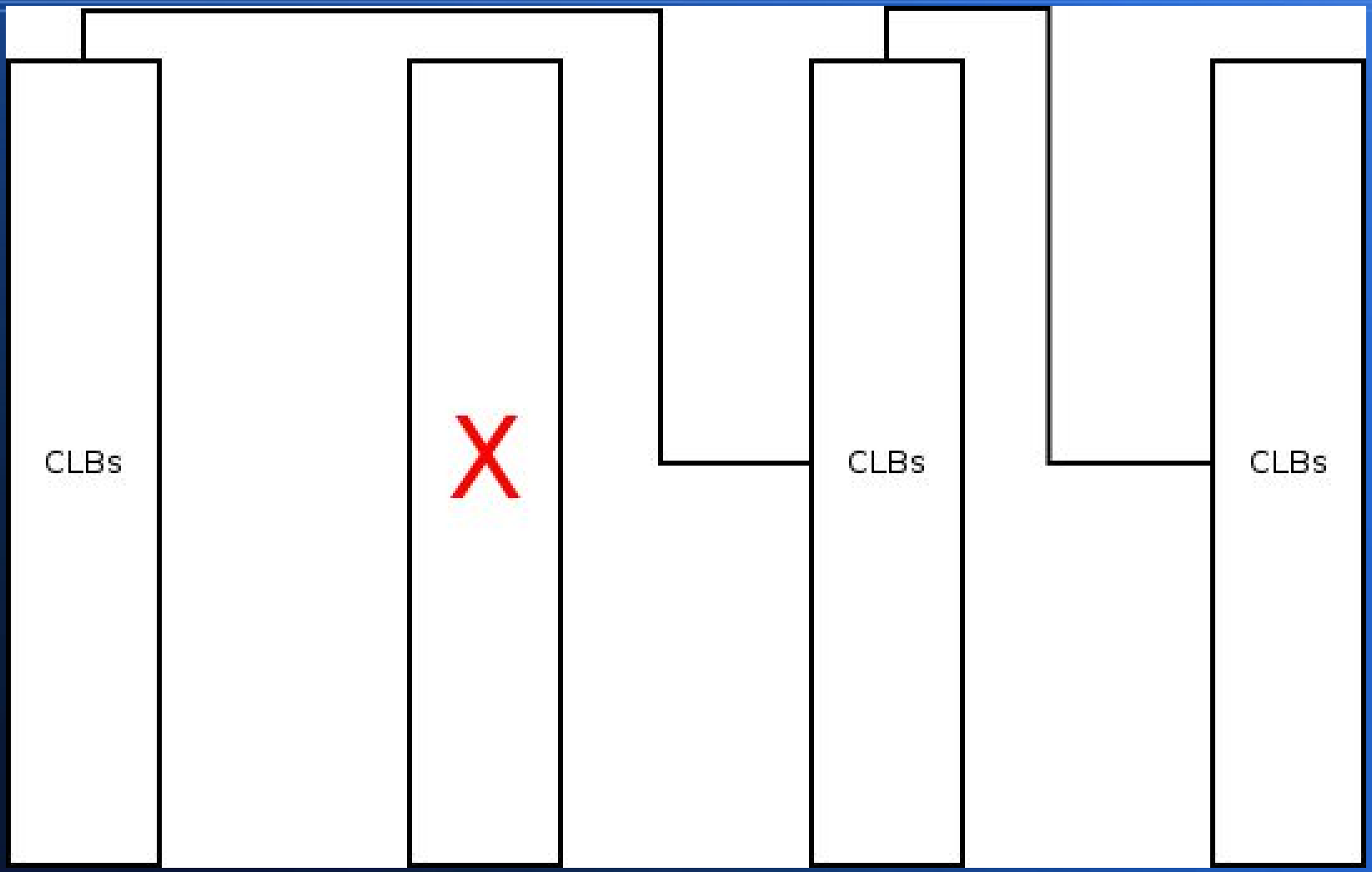
Equivalent circuit - 4 columns



One bad column



Equivalent circuit - 3 columns



Boot process

- Use mode pins to select config source (serial flash, parallel flash, wait for JTAG, etc)
- Wipe existing config, if any
- Start reading bitstream
- Look for sync header
- Execute bitstream on the config state machine

Xilinx bitstream structure

- A Xilinx bitstream file consists of header data plus the actual FPGA configuration data.
- The FPGA bitstream itself is essentially microcode for a simple state machine and consists of a series of scripted register reads and writes.
- All of the high-level structure is documented, however the logic array configuration is not.

Generic Xilinx .bit file structure

- Xilinx .bit files are more than just config data!
- Magic number
  - 00 09
  - 0f f0 0f f0 0f f0 0f f0
  - 00 00 01
- Five config records
  - Record type (2 bytes, lowercase letter + null)
  - One length byte

Generic Xilinx .bit file structure

- Record 'a': Bitstream description
  - “foo.ncd;UserID=0xFFFFFFFF”
- Record 'b': Device name
  - “6slx9tqg144”
- Record 'c': Compilation date
  - “2014/04/15”
- Record 'd': Compilation time
  - “10:44:45”

Generic Xilinx .bit file structure

- Record 'e': Padding before actual bitstream
- All of the .bit headers are for dev tools etc only; the actual device never sees any of this data.
- Typically only the raw binary bitstream, not the full .bit file, is stored on firmware flash etc.

Spartan-6 bitstream structure

- Treated as a sequence of 16-bit words
- 16 dummy words (0xFFFF) to flush pipeline
- Sync word (aa99 5566) for endian detection
- Configuration frames
  - 3-bit type (always 1 or 2)
  - 2-bit opcode (0=nop, 1=read, 2=write)
  - 6-bit register ID
  - 5-bit word count

Spartan-6 bitstream structure

- Type 1 frames are single register reads/writes
  - Documented in UG380 page 94+
- Type 2 frames are bulk data (config stuff only)
  - Length field in header word is ignored
  - Actual length is the next two data words
  - Len[31:28] is always zero
  - Bulk data follows

Example XC6SLX9 bitstream

- Type 1 WRITE to register 0x0f (CWDT), 1 words, Value = ffff
- Type 1 WRITE to register 0x13 (GENERAL1), 1 words
  - Multiboot start address low: 44
- Type 1 WRITE to register 0x14 (GENERAL2), 1 words
  - Multiboot SPI opcode: 0x6b
  - Multiboot start address high: 0x0
- Type 1 WRITE to register 0x15 (GENERAL3), 1 words, Value = 44
- Type 1 WRITE to register 0x16 (GENERAL4), 1 words
  - Golden SPI opcode: 0x6b
- Type 1 WRITE to register 0x17 (GENERAL5), 1 words, Value = 0
- Type 1 WRITE to register 0x05 (CMD), 1 words, Command = NULL

Example XC6SLX9 bitstream

- Type 1 WRITE to register 0x18 (MODE), 1 words, Value = 3100
- Type 1 WRITE to register 0x10 (HC_OPT), 1 words, Value = 5f
- Type 1 WRITE to register 0x05 (CMD), 1 words, Command = IPROG
  - IPROG reset (to address 44)
  - Sync word = aa995566
- Type 1 WRITE to register 0x05 (CMD), 1 words, Command = RCRC
- Type 1 WRITE to register 0x0d (FLR), 1 words, Value = 380
- Type 1 WRITE to register 0x0a (COR1), 1 words, Value = 3d0c
- Type 1 WRITE to register 0x0b (COR2), 1 words, Value = 9ee
- Type 1 WRITE to register 0x0e (IDCODE), 2 words, ID code = 04001093
- Type 1 WRITE to register 0x07 (MASK), 1 words, Value = cf

Example XC6SLX9 bitstream

- Type 1 WRITE to register 0x06 (CTL), 1 words, Value = 81
- Type 1 WRITE to register 0x1c (CCLK_FREQ), 1 words, Value = 3cc8
- Type 1 WRITE to register 0x0c (PWRDN), 1 words, Value = 881
- Type 1 WRITE to register 0x21 (EYE_MASK), 1 words, Value = 0
- Type 1 WRITE to register 0x10 (HC_OPT), 1 words, Value = 1f
- Type 1 WRITE to register 0x0f (CWDT), 1 words, Value = ffff
- Type 1 WRITE to register 0x19 (PU_GWE), 1 words, Value = 5
- Type 1 WRITE to register 0x1a (PU_GTS), 1 words, Value = 4
- Type 1 WRITE to register 0x18 (MODE), 1 words, Value = 3100
- Type 1 WRITE to register 0x13 (GENERAL1), 1 words
  - Multiboot start address low: 0

Example XC6SLX9 bitstream

- Type 1 WRITE to register 0x14 (GENERAL2), 1 words
  - Multiboot SPI opcode: 0x0
  - Multiboot start address high: 0x0
- Type 1 WRITE to register 0x15 (GENERAL3), 1 words, Value = 0
- Type 1 WRITE to register 0x16 (GENERAL4), 1 words
  - Golden SPI opcode: 0x0
- Type 1 WRITE to register 0x17 (GENERAL5), 1 words, Value = 0
- Type 1 WRITE to register 0x1d (SEU_OPT), 1 words, Value = 1be2
- Type 1 WRITE to register 0x1e (EXP_SIGN), 2 words, Value = 00000000
- Config frame starting at 0x146: Type 1 WRITE to register 0x01 (FAR_MAJ), 2 words
- Value = 00000000

Example XC6SLX9 bitstream

- Type 1 WRITE to register 0x05 (CMD), 1 words, Command = WCFG
- Type 2 WRITE to register 0x03 (FDRI), 170157 words *[This is the array config!]*
- Type 1 WRITE to register 0x01 (FAR_MAJ), 2 words, Value = 010e0017
- Type 1 WRITE to register 0x05 (CMD), 1 words, Command = WCFG
- Type 2 WRITE to register 0x03 (FDRI), 130 words
- Type 1 WRITE to register 0x05 (CMD), 1 words, Command = GRESTORE
- Type 1 WRITE to register 0x05 (CMD), 1 words, Command = LFRM
- Type 1 WRITE to register 0x05 (CMD), 1 words, Command = GRESTORE
- Type 1 WRITE to register 0x05 (CMD), 1 words, Command = START
- Type 1 WRITE to register 0x07 (MASK), 1 words, Value = ff
- Type 1 WRITE to register 0x06 (CTL), 1 words, Value = 81

Example XC6SLX9 bitstream

- Type 1 WRITE to register 0x00 (CRC), 2 words, CRC = 00245bba
- Type 1 WRITE to register 0x05 (CMD), 1 words, Command = DESYNC

Code protection

- FPGA bitstreams are typically loaded from external flash (SPI or parallel)
- This makes dumping the code trivial
  - Can then be copied or reverse engineered
- Three main approaches:
  - No countermeasures at all (not worth stealing)
  - Encrypt bitstream
  - Store bitstream in flash on die

Bitstream encryption

- OTP/fuse memory
  - Nonvolatile, but can't be changed
  - Can potentially be extracted by delayering to poly and imaging fuse cells
- Battery backed SRAM
  - Needs battery backup but can be updated
  - Harder to extract
  - Can be zeroized by self-destruct system

Reverse engineering

- The actual configuration data is totally undocumented and FPGA vendors want to keep it that way (“security by obscurity”)
- “In fact, FPGA manufacturers have no tools that can be used to recover a netlist from a bitstream. Given the sheer size of modern FPGAs and the number of configuration bits involved, recovering an entire design from a bitstream is unlikely, probably requiring the resources of a state actor. In general, the bitstream generation process serves as a type of design obfuscation”
  - Xilinx WP365 page 5

Reverse engineering

- Several published papers have succeeded in reversing parts of various FPGA vendors' bitstream formats and completely debunked the claims of security by obscurity
- “From the bitstream to the netlist”
  http://www.univ-st-etienne.fr/salware/Bibliography_Salware/FPGA%20Bistream%20Security/Article/Note2008.pdf
- Sadly, few if any of these tools have been updated for current-generation parts. RE is quite possible but will need custom tool dev.

Other proposed countermeasures

- The paper mentioned on the previous slide suggests using a Spartan-3AN can avoid exposing the bitstream to an outside user.
- Not so fast ;)
- Probe the bond wires and sniff the bitstream

Questions?

- TA: Andrew Zonenberg <azonenberg@drawersteak.com>
- Image credit: Some images CC-BY from:
  - John McMaster <JohnDMcMaster@gmail.com>