18-643 Lecture 2: Basic FPGA Fabric

James C. Hoe
Department of ECE
Carnegie Mellon University

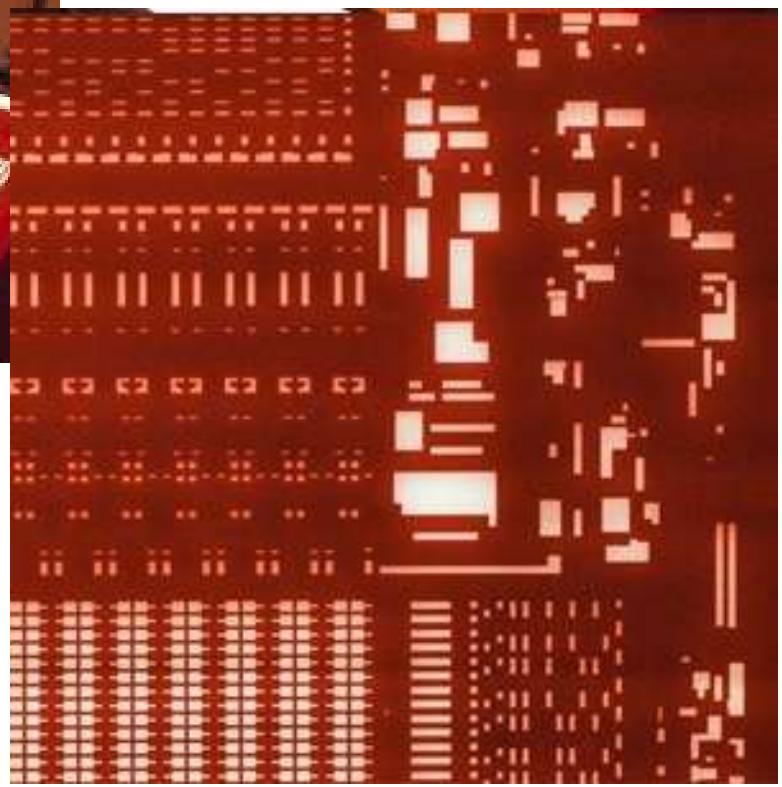
Housekeeping

- Your goal today: know enough to build a basic FPGA (even if not a very good one)
- Notices
  - Complete survey on Canvas, due noon, 9/6
  - Handout #2: lab 0, due noon, 9/11
  - Make friends, make teams, due noon, 9/11
- Readings (see lecture schedule online)
  - Altera 2006 white paper (see lecture schedule)
  - skim databooks referenced for more details

What it means:

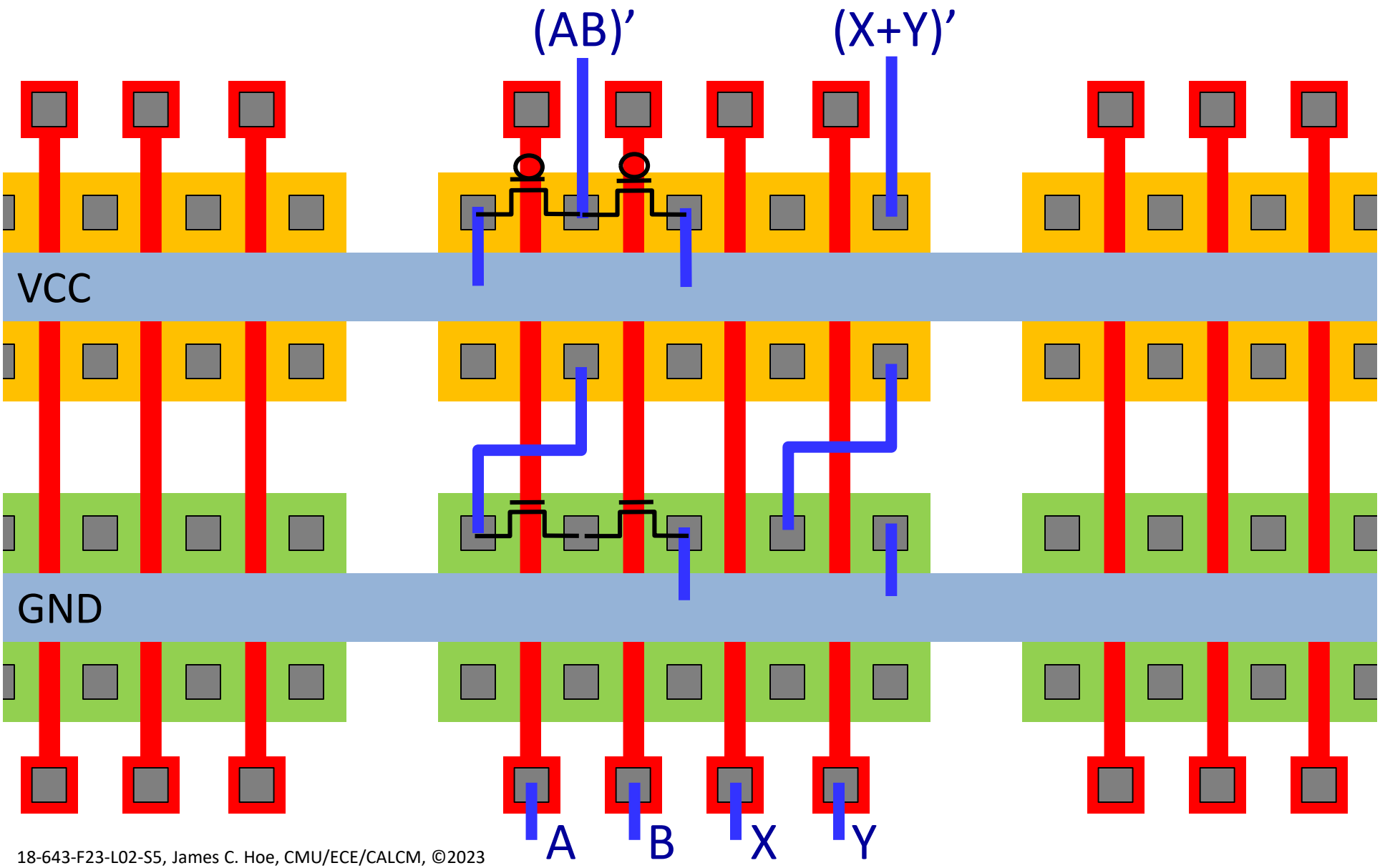
“Field Programmable” “Gate Array”

SSI→ MSI→ LSI→ VLSI



From Quora, “How did people design integrated circuits in early years?”

How to democratize 100K gates



Idea behind Gate Arrays

- Mass produce identical gate array wafers
- Finish into any design by custom metal layers (2)
  - so called Mask-Programmable GA (MPGA)
  - reduced design effort (more automation, no layout)
  - reduced mask and fab cost
  - faster fab turn-around
- Proliferation of ASIC design starts
  - don't need volume for economy of scale
  - small design team could keep up with Moore's law

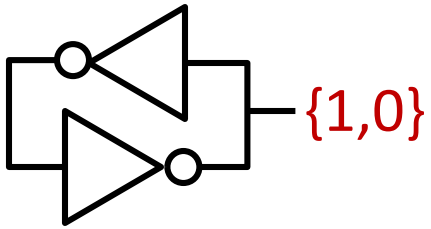
Of course, not as efficient as full-custom or standard-cell designs

How about no mask, no fab? i.e., "field programmable"

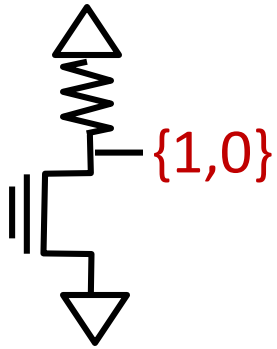
- Again, mass produce identical devices but this time fully-finalized
- Then what can still be changed after fab?

bits

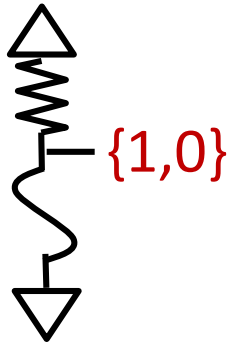
– SRAM



EPROM

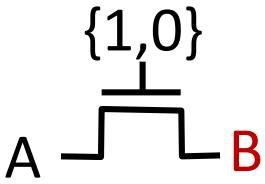


(anti)fuse

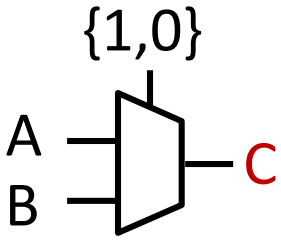


connections

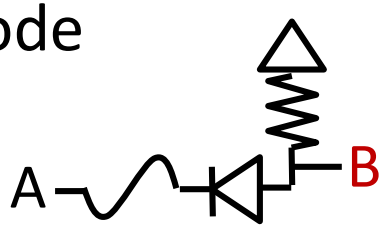
– pass gate



mux



diode

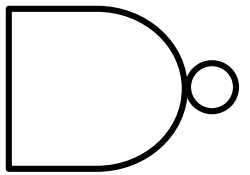


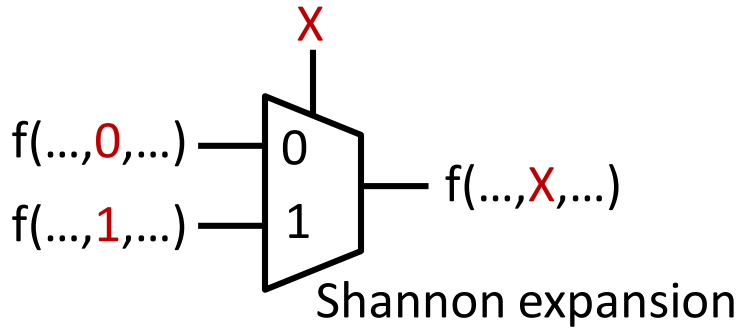
programmable vs reprogrammable

Configurable “Logic Gates”

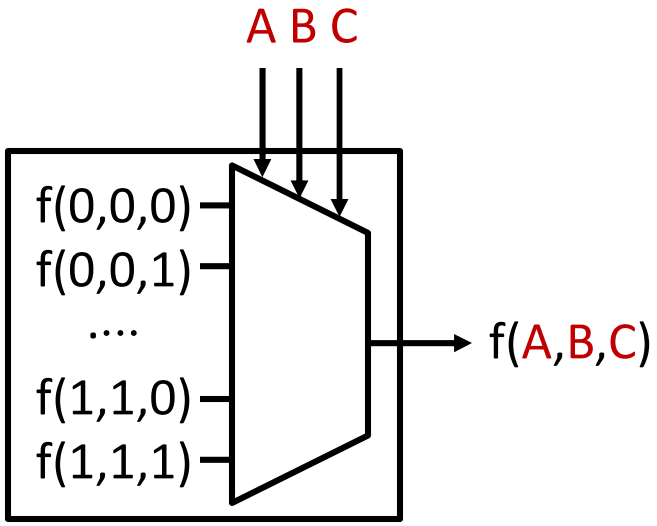
Reconfigurable Logic

- Arbitrary logic (combinational and sequential) can be formed by wiring up enough NANDs or muxes





- Lookup table as universal logic primitive
  - arbitrary n-input function from 2^n-entry table
  - this is 8-by-1 bit "memory"



Size of Lookup Tables (aka LUTs)

- n-input function from 2^n-entry LUT
- Count only the 6T SRAM cells, an n-LUT has $6 \cdot 2^n$ T
- Some points of reference
  - 2-input NAND = 4T
  - 3-input NAND = 6T
  - 3-input full-adder (a, b, c_{in})
    - $s = a \oplus b \oplus c_{in}$ = 8T
    - $c_{out} = bc_{in} + ac_{in} + ab$ =18T
  - 10-input 5-bit adder = 130T
  - basic flip-flop=16T

muxes? (compare to 2 LUTs per latch)

n-LUT	T-count
2	24
3	48
4	96
5	192
6	384
7	768
8	1536
9	3072
10	6144

Choosing LUT Granularity

- Small LUTs
  - + shorter propagation delay (per LUT)
  - – a given fxn consumes many LUTs (comes with wiring cost and delay)
  - – high "interpretation overhead" if too small
- Big LUTs
  - – longer propagation delay (per LUT)
  - + a given fxn consumes fewer (but bigger) LUTs
  - – high "interpretation overhead" if too large (and fxn has exploitable structure, e.g., 5-bit ripple add)
  - – wastage if not all input are used in a LUT

this kind

this kind

Where is the sweetspot?

A Quantitative Look at LUT Sizing
e.g., 2006 Altera White Paper on Stratix-II ALMs

Figure 4. Delay-Cost Tradeoff with LUT Size

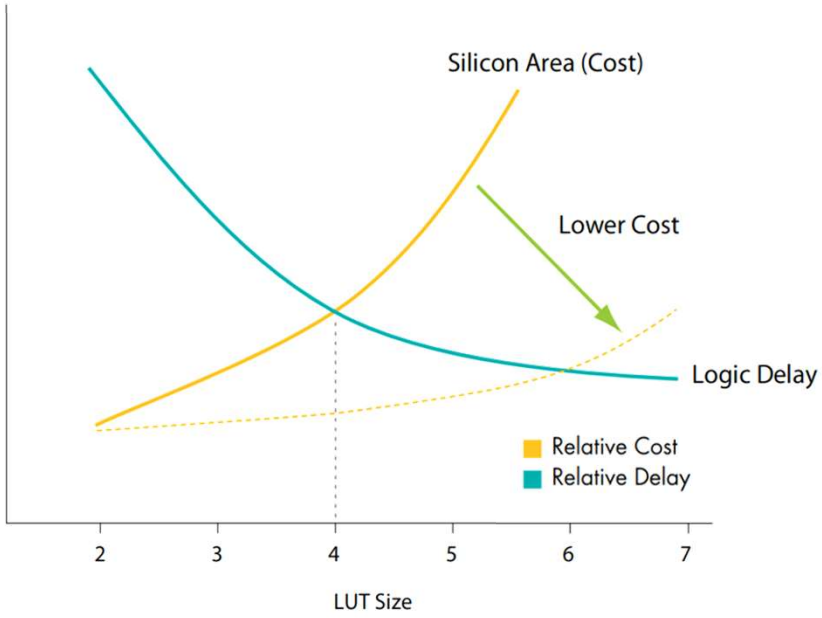
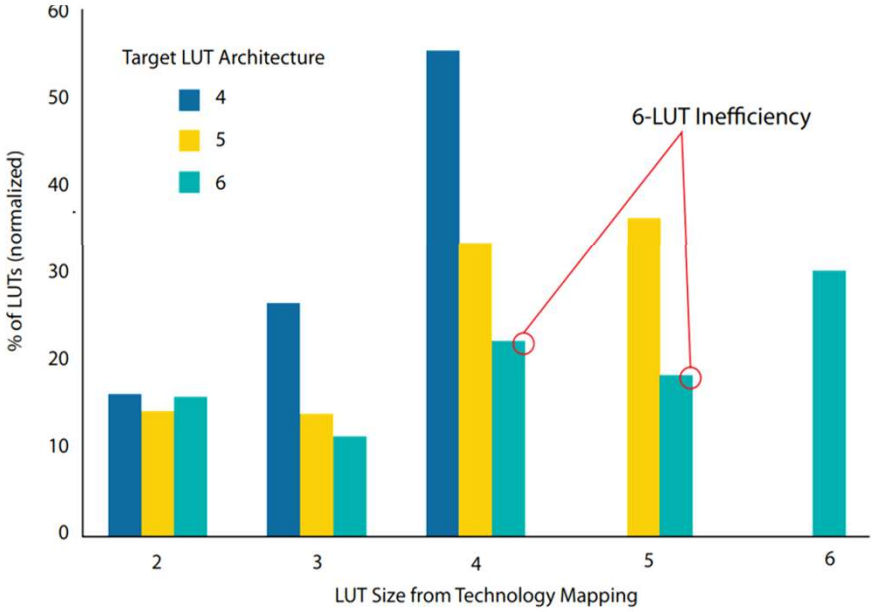


Figure 5. Distribution of Functions Generated by Synthesis



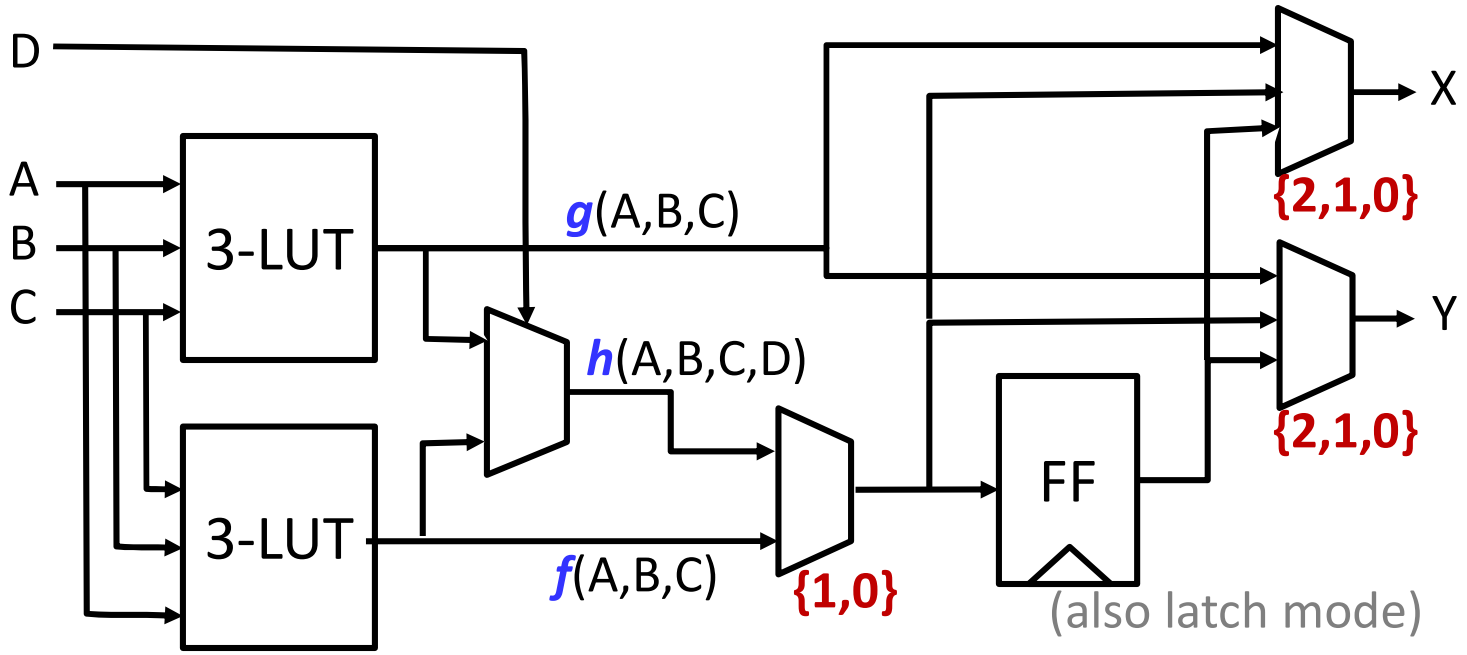
Large-enough functions have shorter total delay using bigger LUTs

But, bigger LUTs cost more and prone to "internal fragmentation"

4-LUTs 50+% fully utilized
6-LUTs less than 40% fully utilized

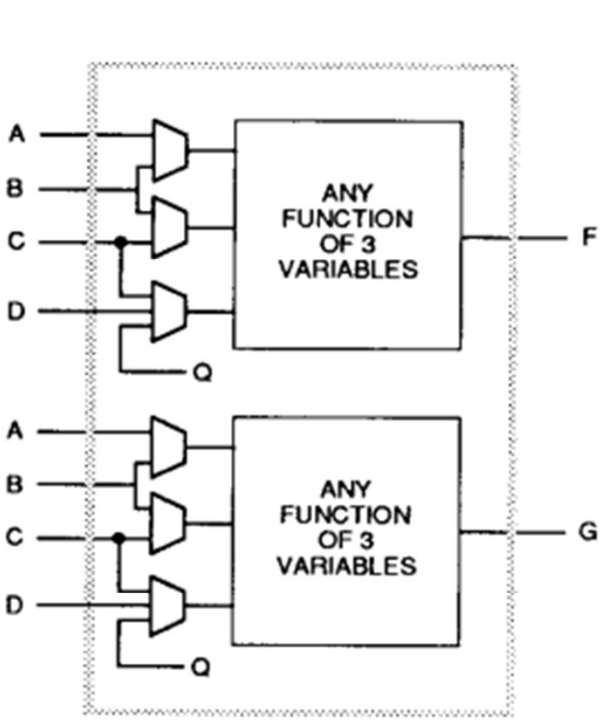
No one LUT size optimal
$\Rightarrow$ "adaptive" LUT approach

LUT-based Configurable Logic Block (simplified sketch)



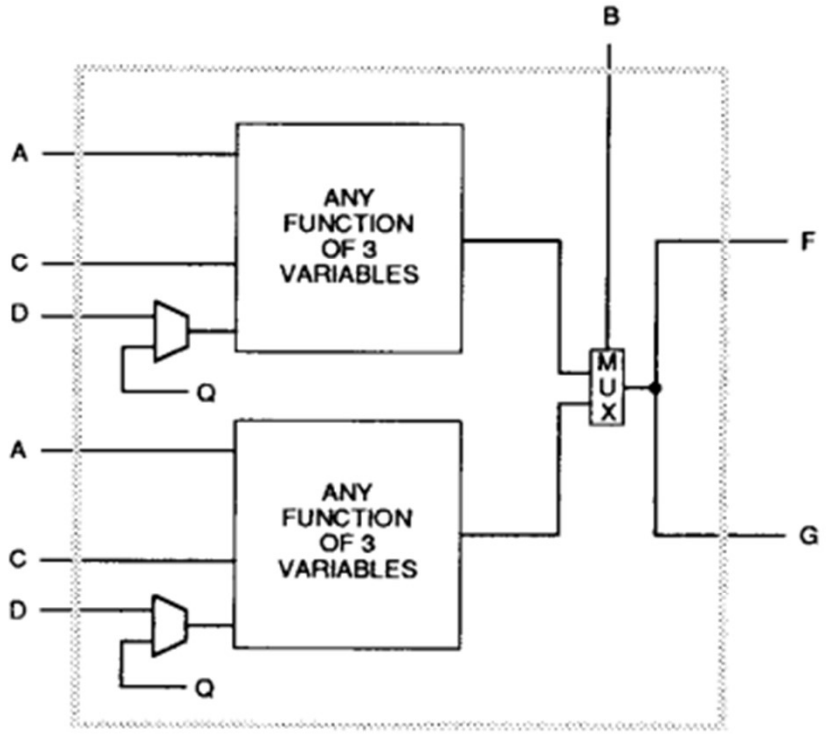
- 2 fxns (*f* & *g*) of 3 inputs OR 1 fxn (*h*) of 4 inputs
- hardwired FFs (too expensive/slow to fake)
- Just 10s of these in the earliest FPGAs

Xilinx XC2000 CLB (1980s)



OPTION 2

2 FUNCTIONS OF 3 VARIABLES



OPTION 3

DYNAMIC SELECTION OF 2 FUNCTIONS OF 3 VARIABLES

[XC2064, XC2018 Logic Cell Array]

Contemporary Xilinx CLB Architecture

- each 6LUT is two 5LUTs
- LUTs can also be used as small SRAMs
- special paths for addition and multiplexer

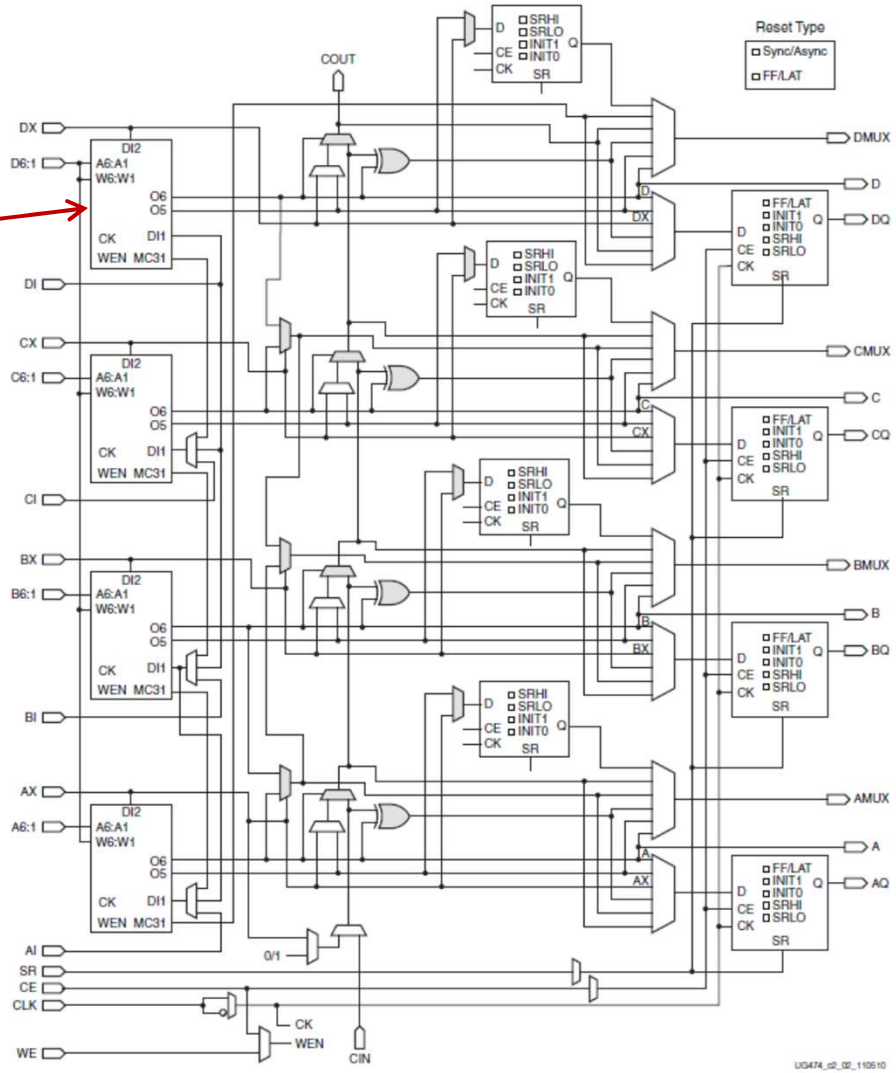


Figure 2-3: **Diagram of SLICEM**

2 slices per CLB

Largest devices (many $K each) have several 100K slices

Largest extreme in 2022 has over 1M slices

[Figure 2-3: 7 Series FPGAs CLB User Guide]

Still Coarser Logic Blocks?

- So called Coarse-Grain Reconfigurable Arrays (CGRAs) based on complete adders or ALUs
  - native arithmetic units have low interpretation overhead if you are doing arithmetic
  - poor fit if you are working with narrow data or bit-level manipulations
- Even coarser is to use many tiny processors
  - still a spatial computing paradigm
  - not programmed with RTLs
  - converging with software multicores

More on this later on

Brief Aside: Mapping Logic To LUTs

- Start from primary output and input to registers, cover logic graph with cuts of less than K input edges
- K-cuts corresponding to K-LUT realizable functions

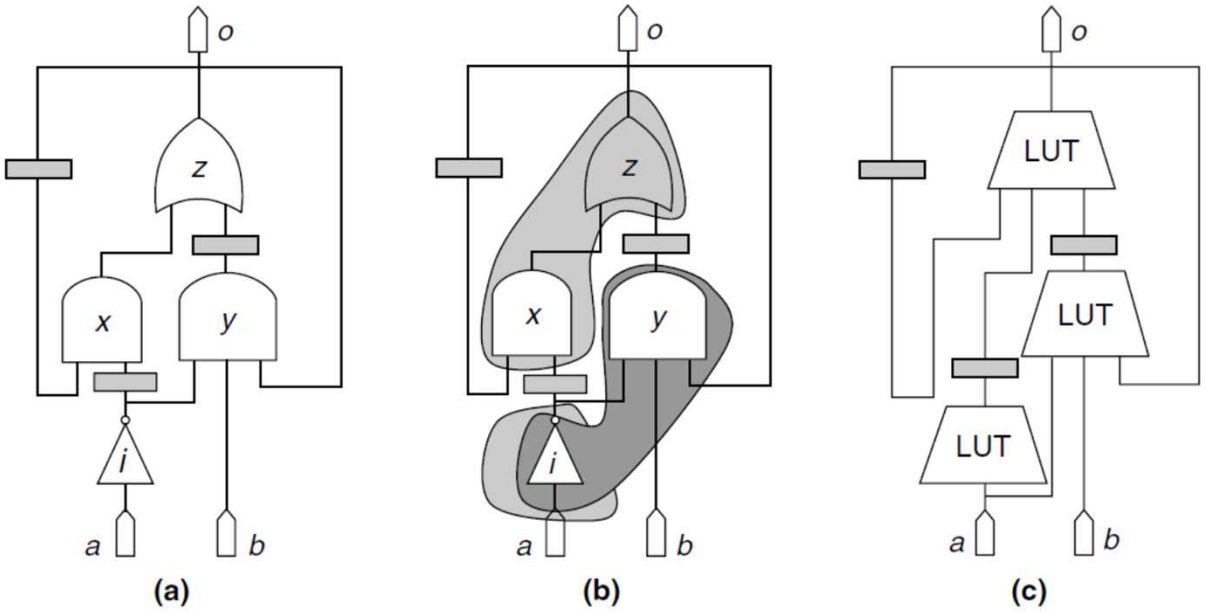
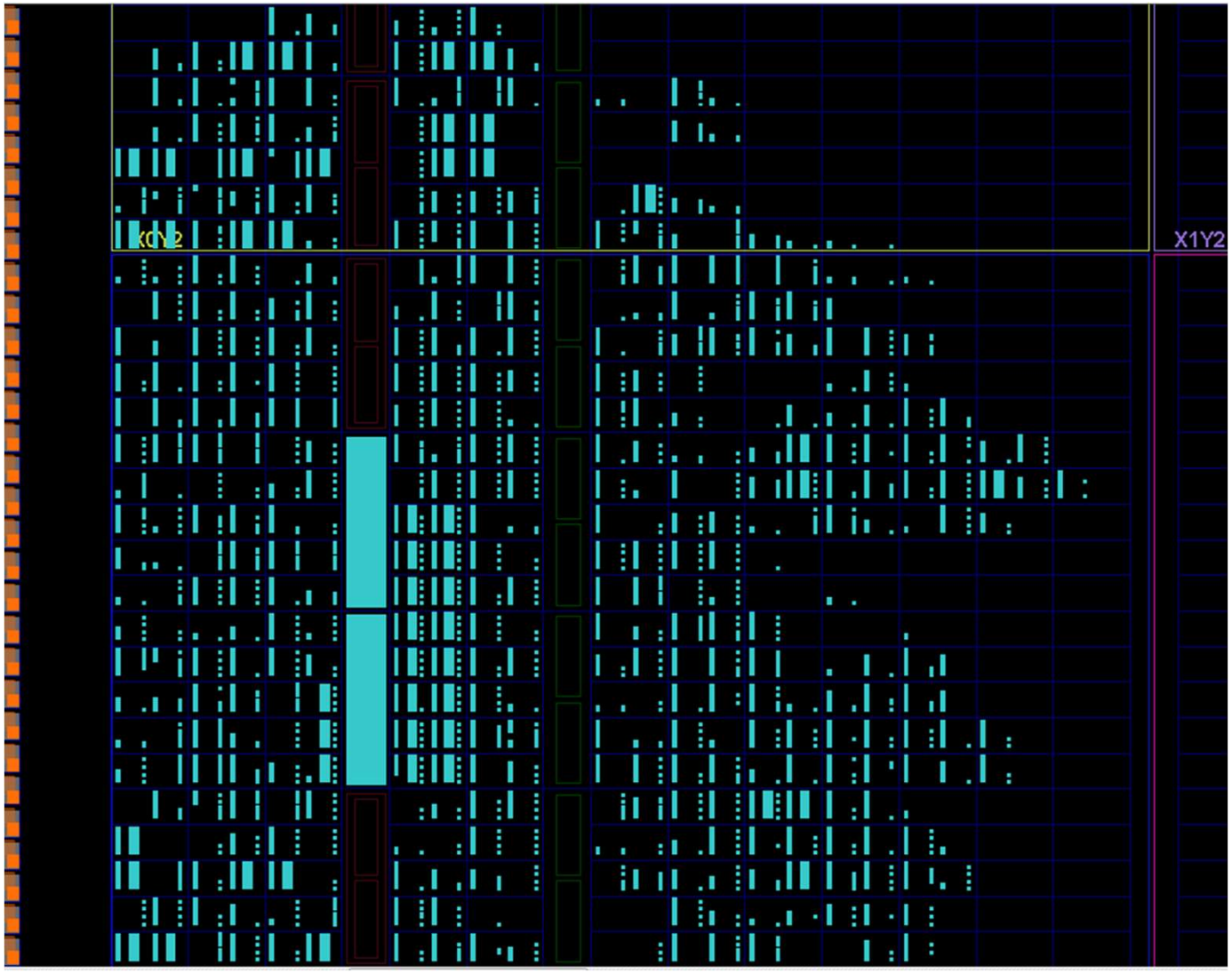


FIGURE 13.1 ■ Structural technology mapping: (a) original network, (b) covering, and (c) mapping solution.

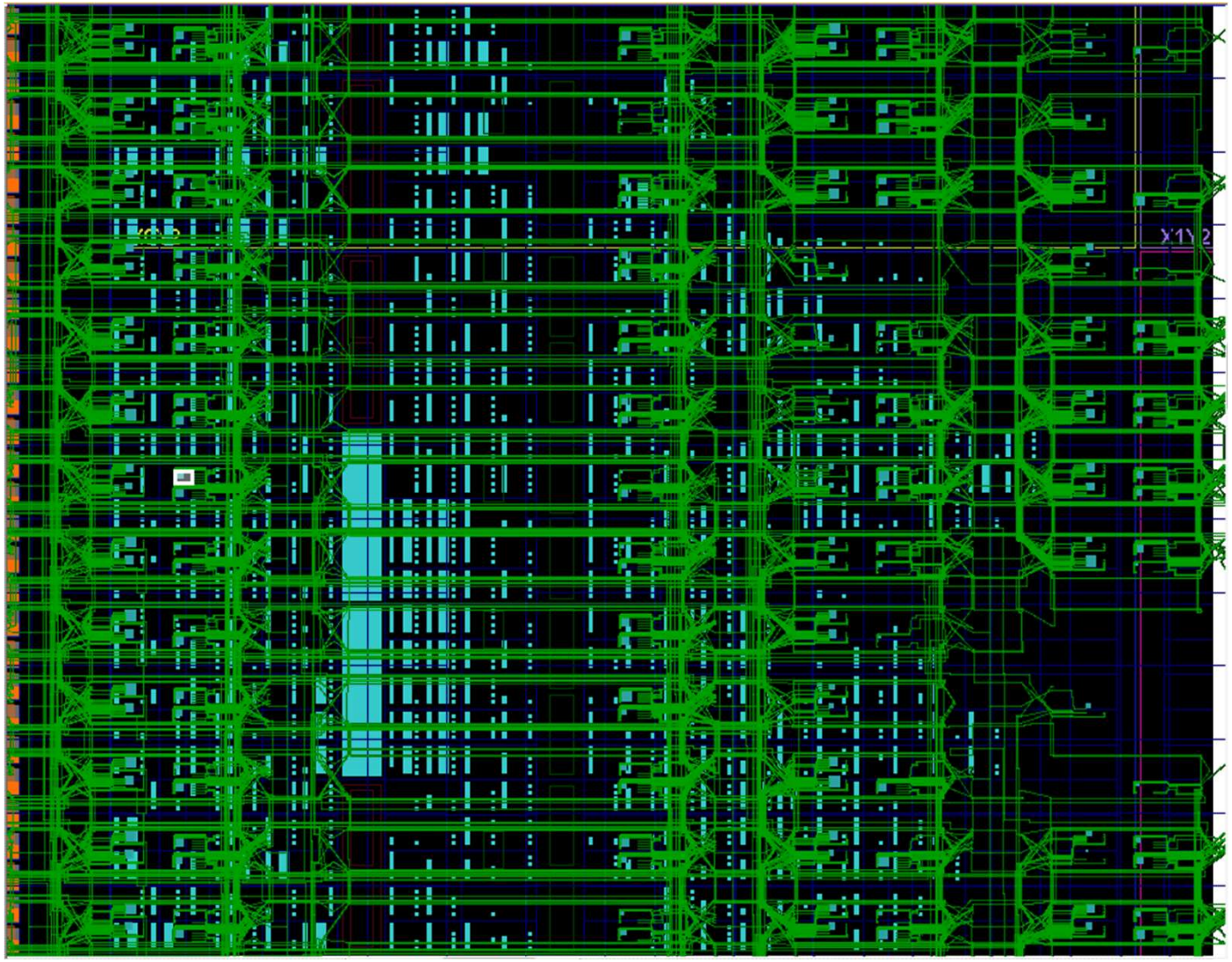
[Figure 13.1: "Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation"]

Placement



[Vivado Implementation Screenshot]

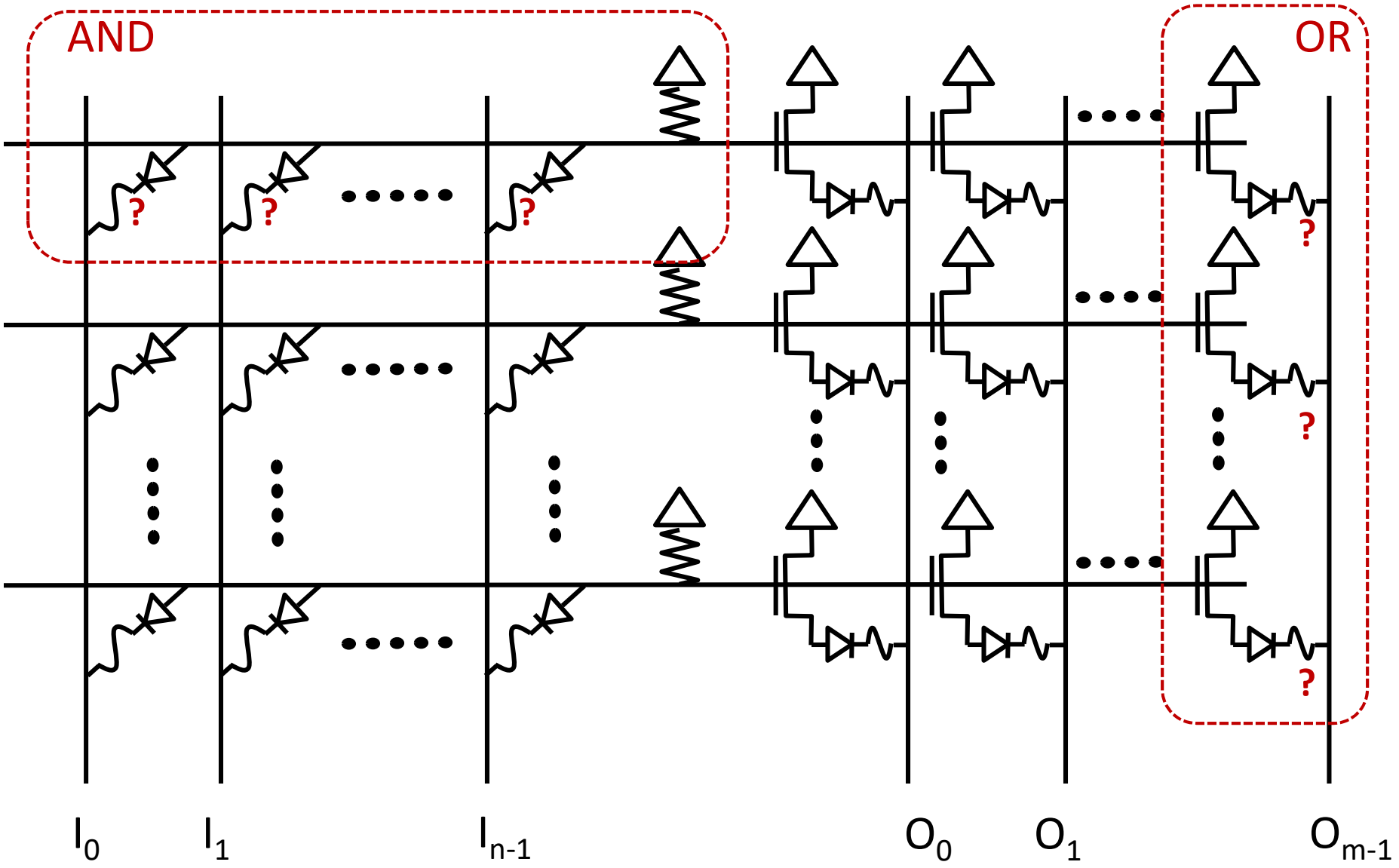
… and Route



[Vivado Implementation Screenshot]

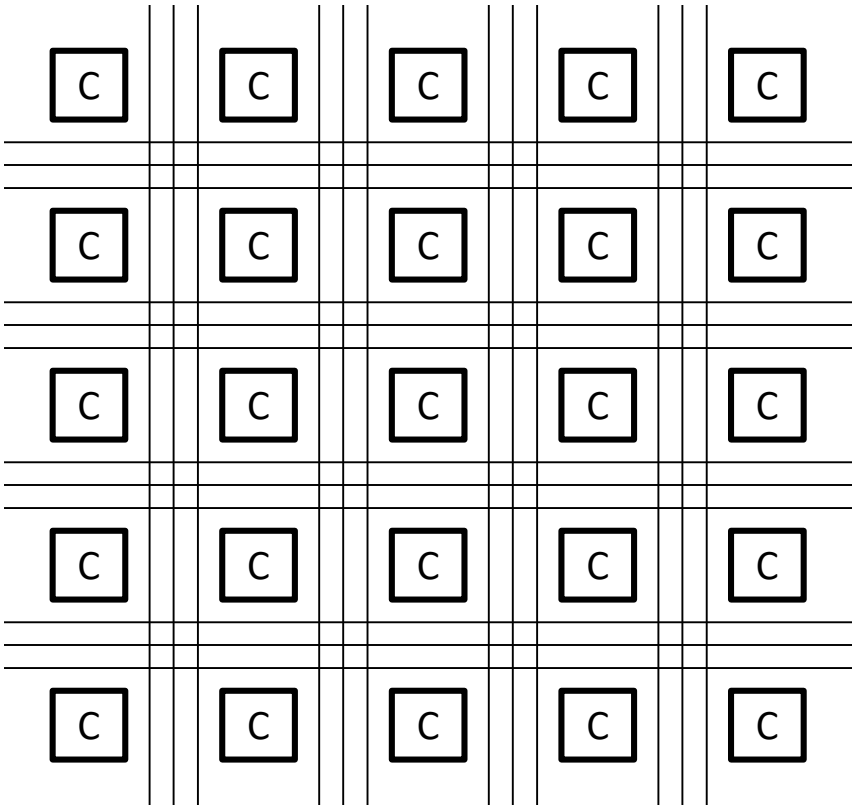
Configurable “Wires”

PLA-style Configurable Routing

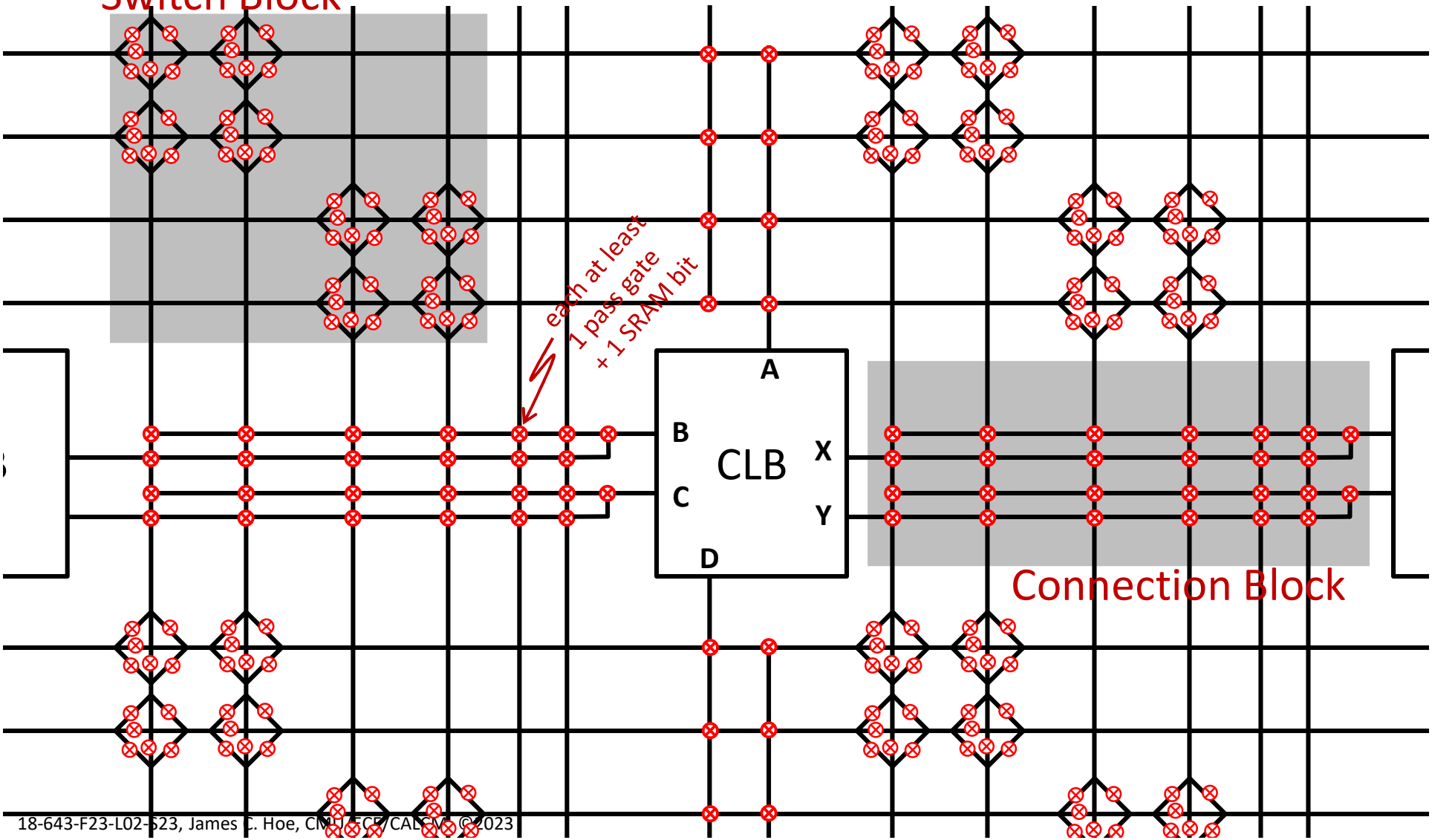


Island Style Routing Architecture

- CLB islands in sea of interconnects
- Flexible routing to support ASIC style netlists
- Note regularity in structure



Configurable Routing (1980s Xilinx simplified)

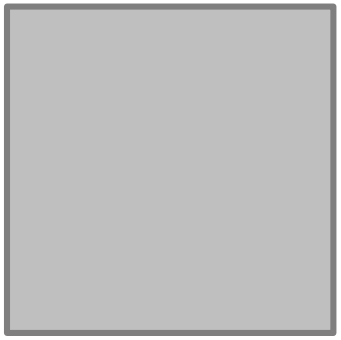


Reconfigurable Routing is Expensive!

- Routing resource area is on par with logic
- Each configurable connection is
  - area of configuration bit
  - area of configurable connection

  and don't forget propagation delay
- Too much: cost for everyone who doesn't need it
- Too little: congestion leaves unreachable CLBs unused
  - worse for larger arrays/designs (why?)
  - buy a $10K FPGA and only get to use 70%?

Rent's Rule



- $T \propto g^p$
  - T = number of inputs and outputs
  - g = number of internal components
  - p typically between 0.5 (regular) and 0.8 (random)
- In a square, perimeter=$4 \cdot \text{area}^{0.5}$
  - unless regular, I/O signals grow faster than available routes exiting a design area
- Need hierarchy of progressively longer additional routing resources

long routes also reduce delay when going far

Virtex-II Routing Architecture

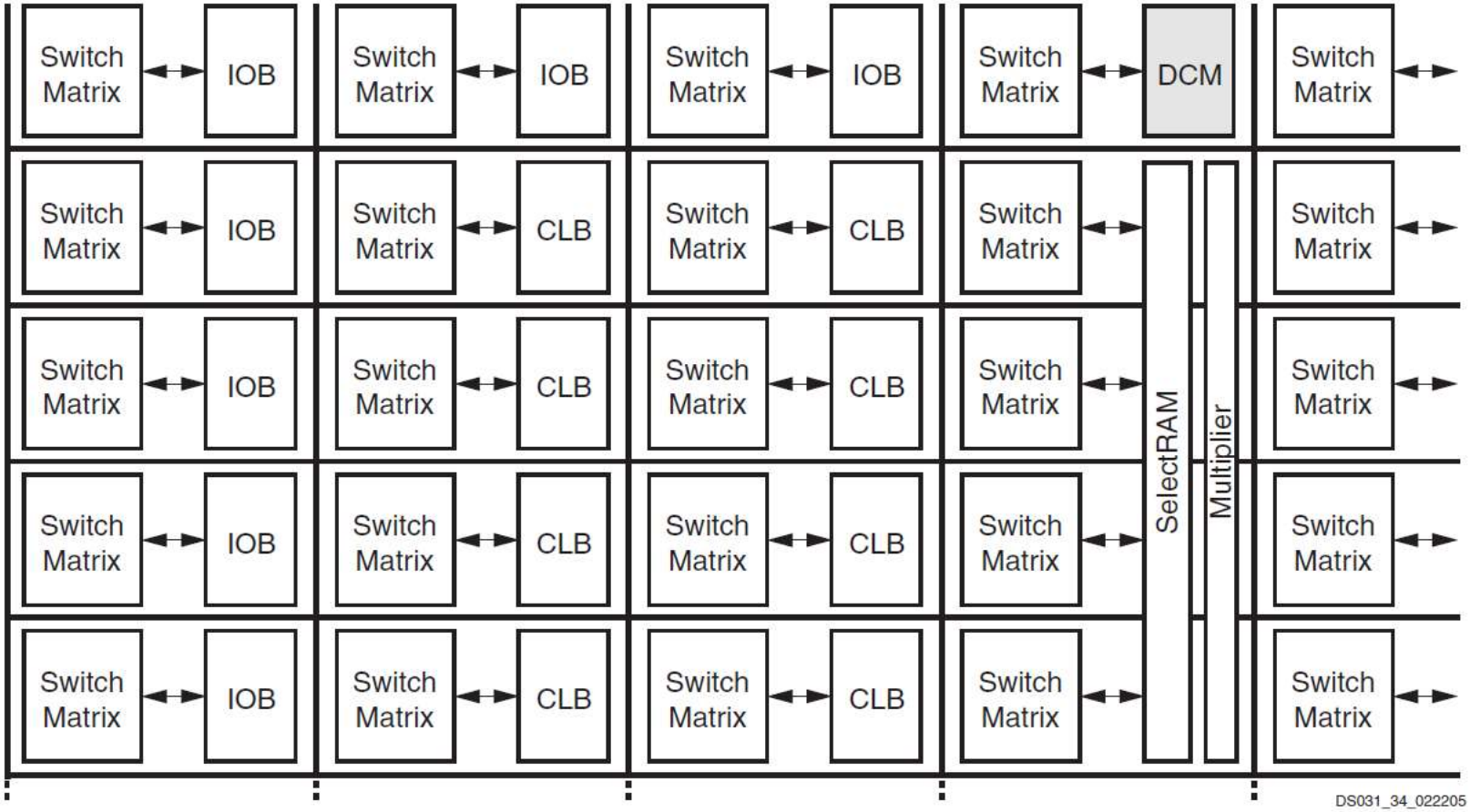


Figure 48: **Routing Resources**

[Figure 48: Virtex-II Platform FPGAs: Complete Data Sheet]

Virtex-II Routing Architecture

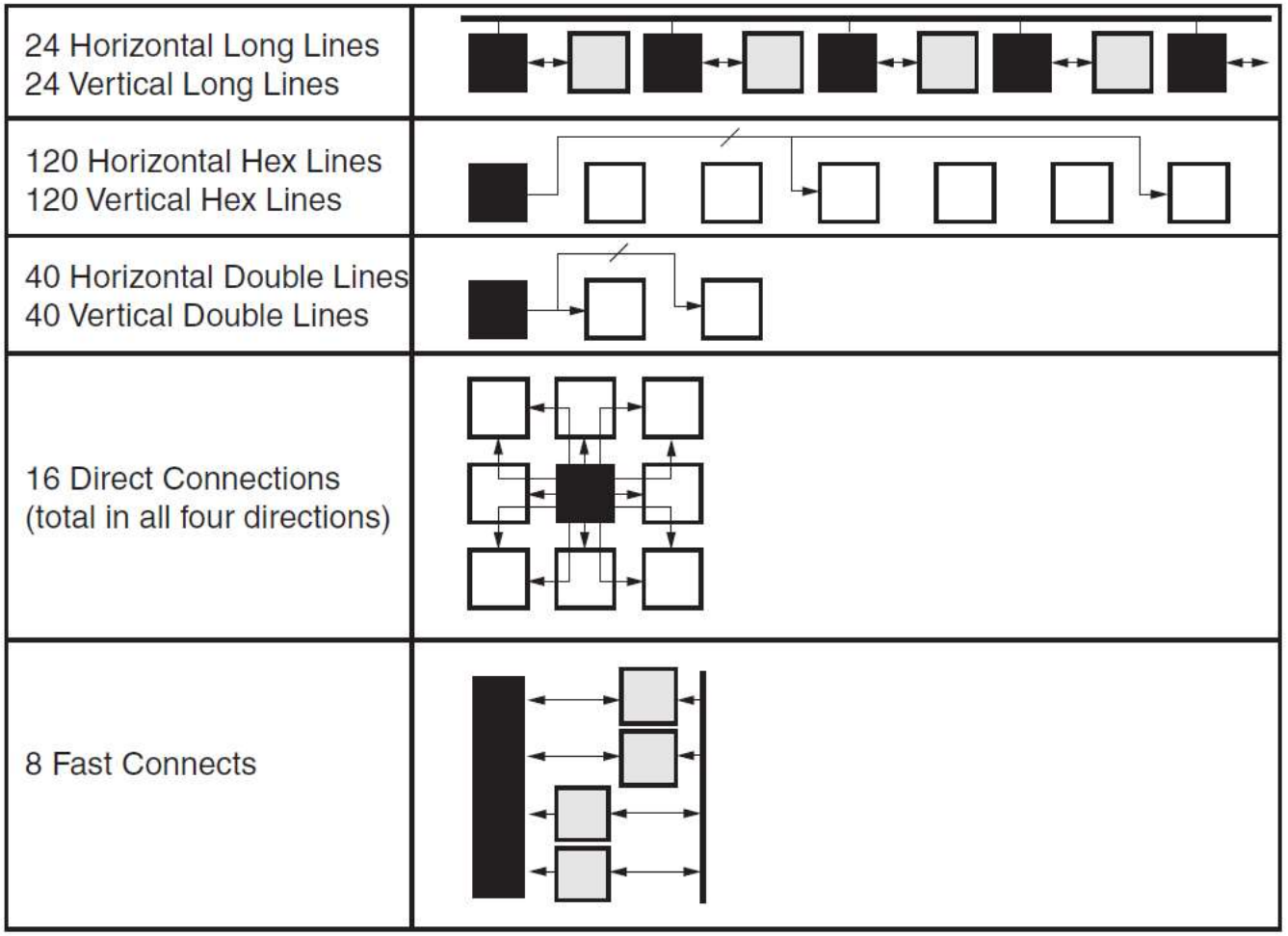


Figure 49: **Hierarchical Routing Resources**

Later architectures extended in reach and in diagonals

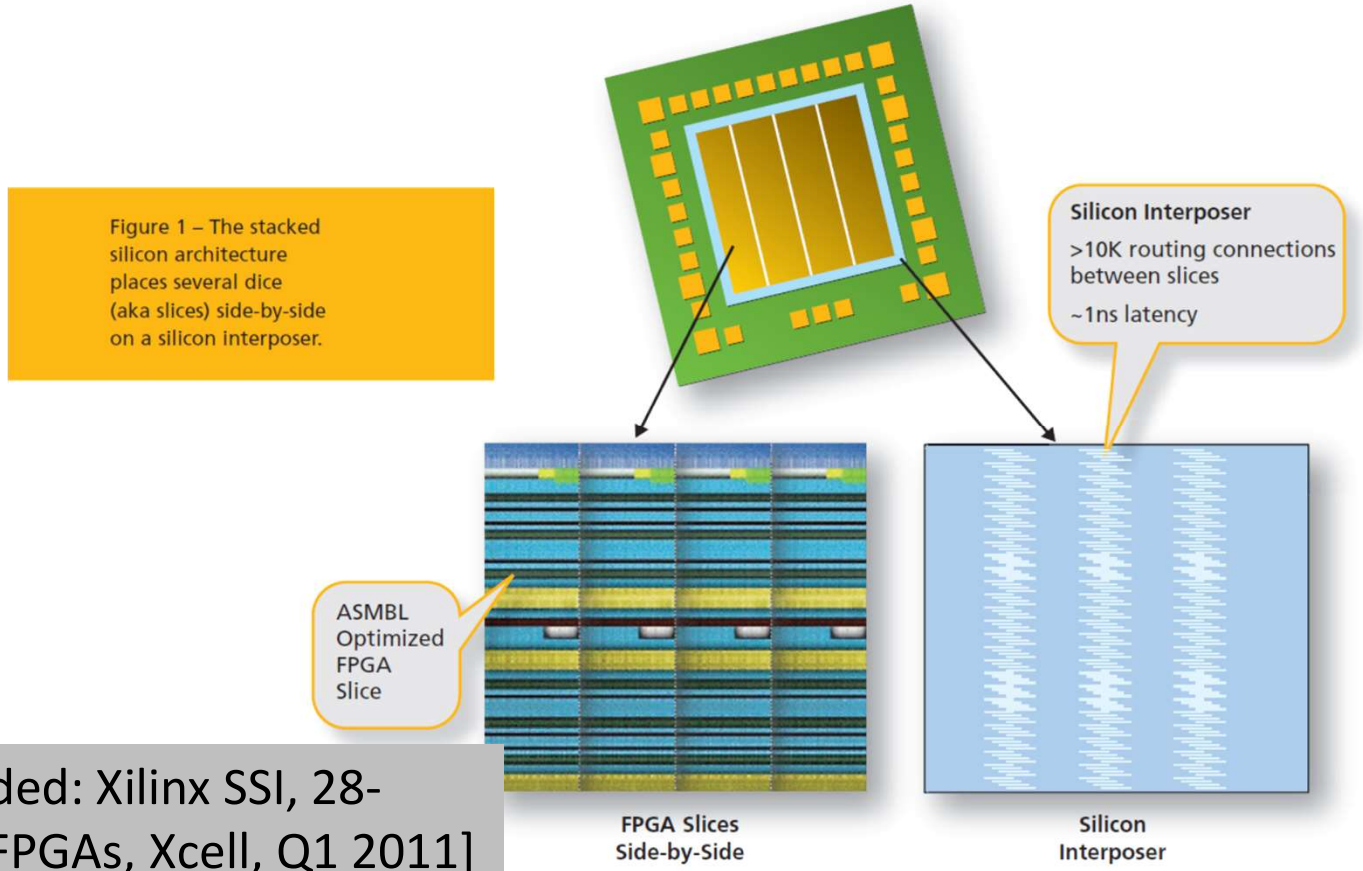
Separate, dedicated clock trees

[Figure 49: Virtex-II Platform FPGAs: Complete Data Sheet]

Between-Die Routing in 2.5D IC

Virtex7 Stacked Silicon Interconnect (SSI), 2011

- Longest routes go across dies carried on interposer
- No change to design tool and abstraction

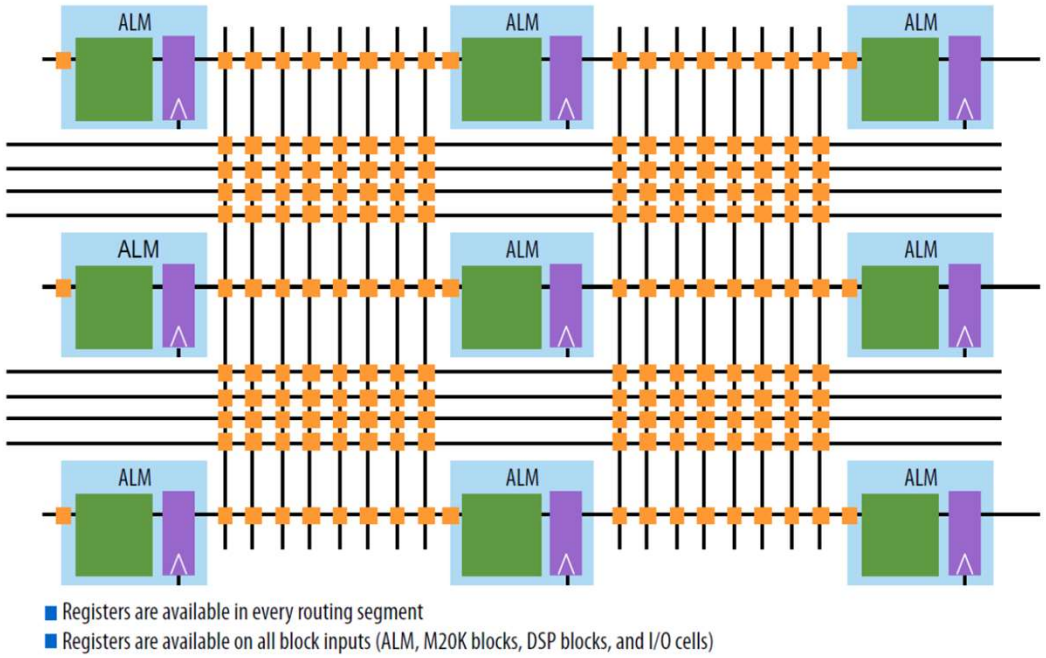


[Figure 1, Stacked & Loaded: Xilinx SSI, 28-Gbps I/O Yield Amazing FPGAs, Xcell, Q1 2011]

Intel Stratix-X HyperFlex

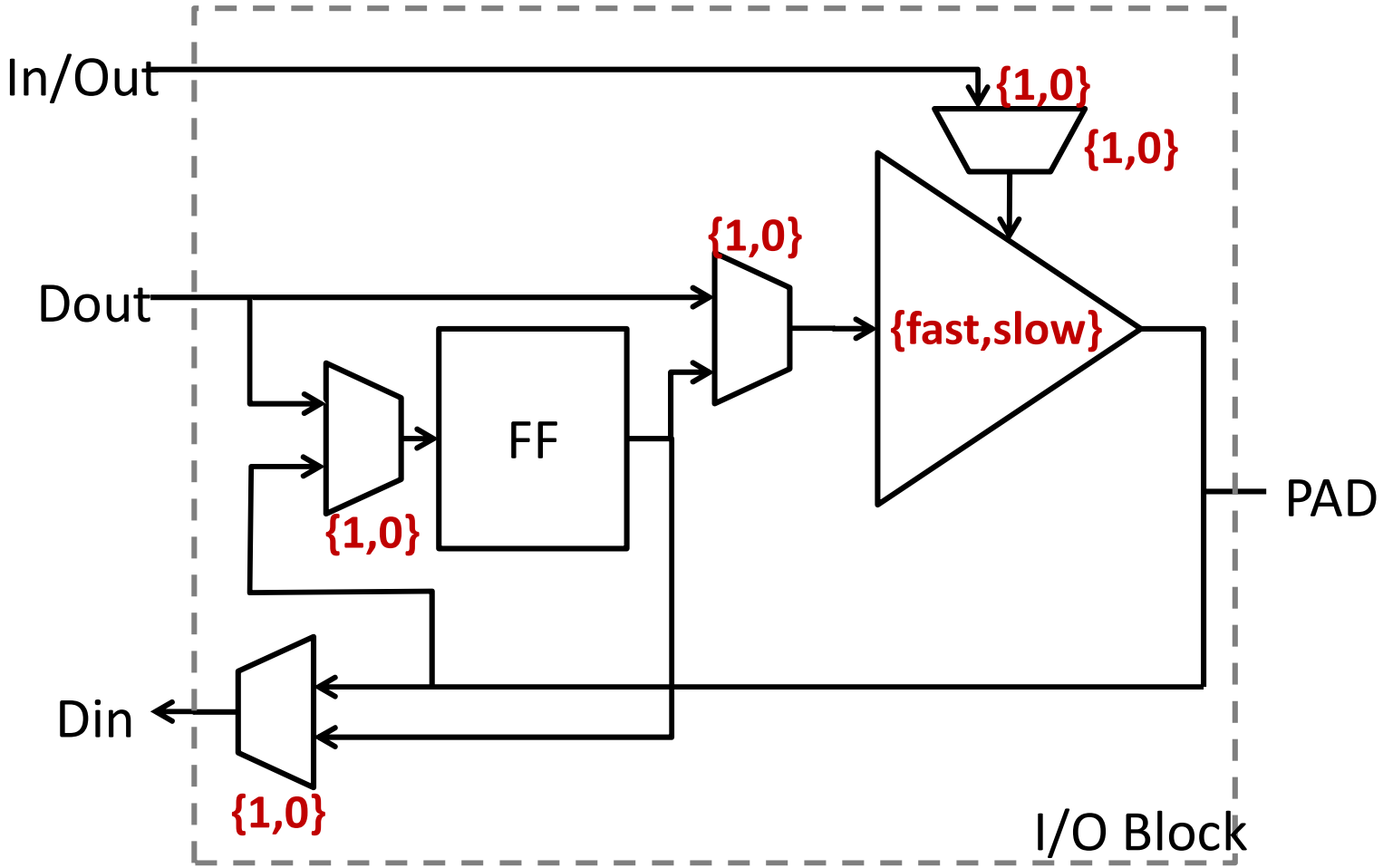
- Long routes need buffered repeaters; very long routes need pipelining
- Add (bypassable) pipeline registers throughout
- RTL designs have to be pipelined explicitly to benefit; high-level synthesized designs leverage directly
- a high-freq strategy — e.g., 0.5xlogic at 2xfreq for perf. parity

Figure 2. "Registers Everywhere" HyperFlex Architecture



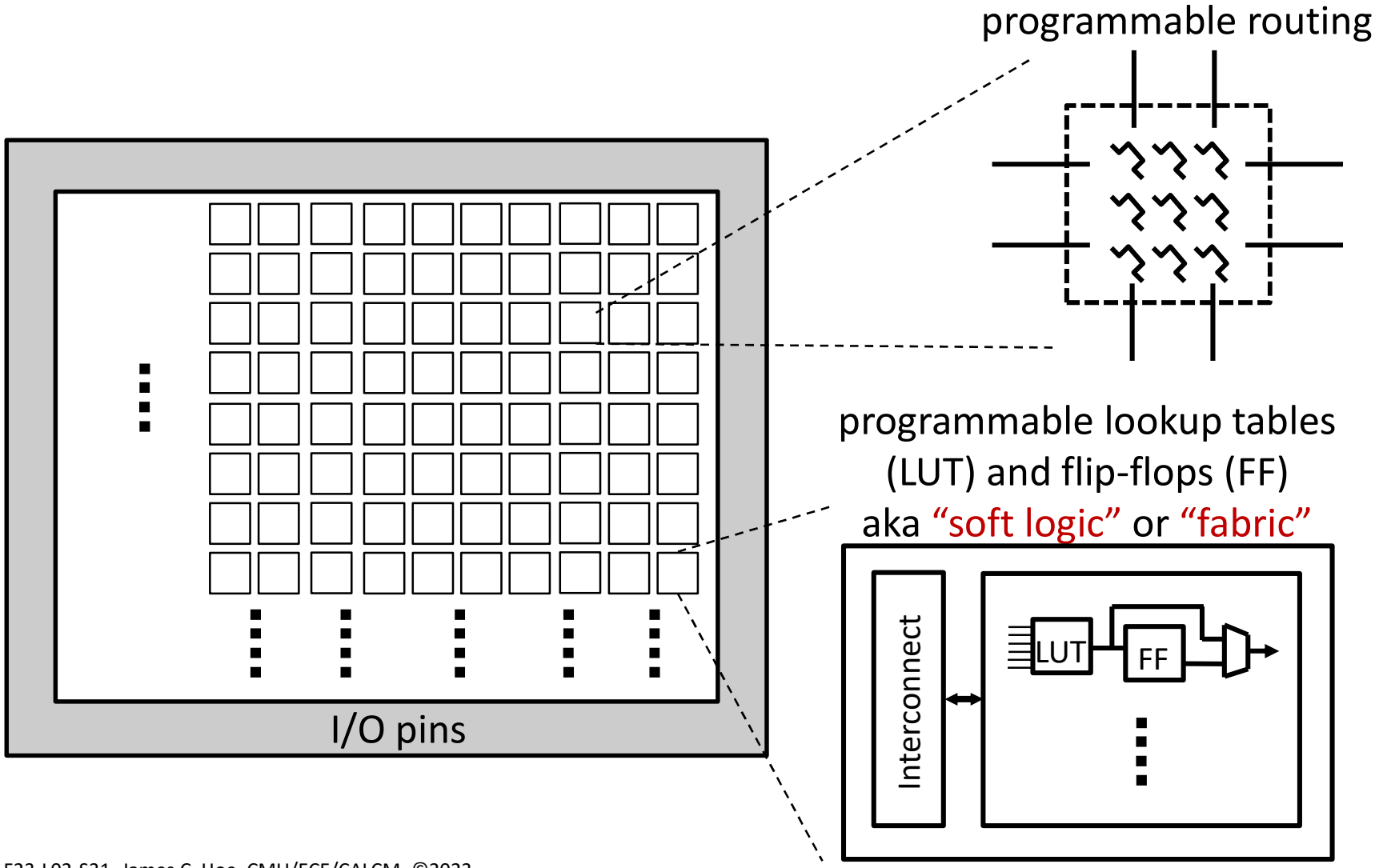
[Figure 2: Understanding How the New HyperFlex Architecture Enables Next-Generation High-Performance Systems]

Don't Forget Configurable I/O



- real devices more complicated
- modern devices support special signaling and protocols

Putting it all together: an Universal ASIC



Bitstream defines the chip

- After power up, SRAM FPGA loads bitstream from somewhere before becoming the "chip"

  a bonus "feature" for sensitive devices that need to forget what it does

- Many built-in loading options
- Non-trivial amount of time; must control reset timing and sequence with the rest of the system
- Reverse-engineering concerns ameliorated by
  - encryption
  - proprietary knowledge

Return to this later in term . . . .

Parting Thoughts

- Birth of FPGAs rooted entirely in digital logic and ASIC concerns; today, you can use an FPGA without knowing any of this stuff
- You can find a lot of specific details on-line (databooks and research papers)
- So far still just the basic fabric . . . .

  . . . more next time

  - saving "configuration" for later in term
  - won't say anything about low-level EDA