CS152 – Computer Architecture and Engineering

Lecture 3 – Field Programmable Gate Arrays

2003-09-02

Dave Patterson
(www.cs.berkeley.edu/~patterson)

www-inst.eecs.berkeley.edu/~cs152/

Review: Verilog

- Verilog allows both structural and behavioral descriptions, helpful in testing
- Some special features only in Hardware Description Languages
  - # time delay, nonblocking assignments, initial vs. always, forever loops
- Syntax a mixture of C (operators, for, while, if, print) and Ada (begin… end, case…endcase, module …endmodule)
- Verilog can describe everything from single gate to full computer system; you get to design a simple processor

Multiple Review

- Multiply: successive refinement to see final design
  - 1st iteration:
    64-bit Adder,
    64-bit Multiplicand shift register,
    32-bit Multiplier shit register,
    64-bit Product register
  - 3rd iteration:
    32-bit Adder,
    64-bit Product/Mutiplier shift register,
    32-bit Multiplicand Register
  - There are algorithms that calculate many bits of multiply per cycle
    (see exercises 4.36 to 4.39 in COD)

Outline

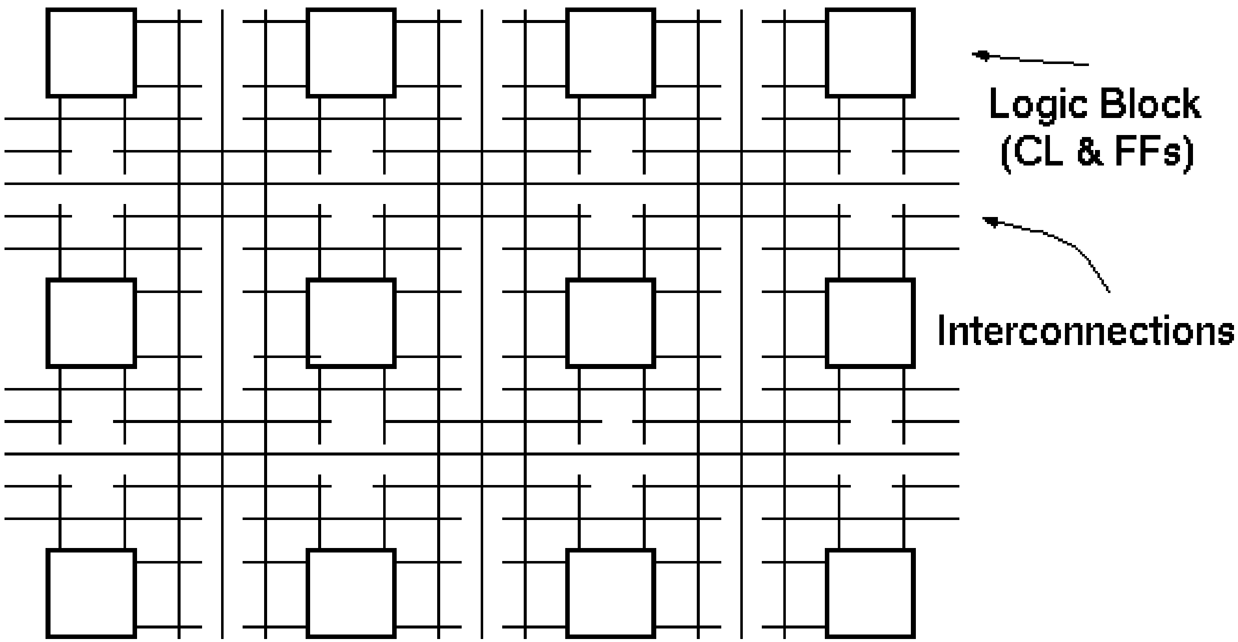
- FPGAs Overview
- Why use FPGAs?
  (a short history lesson).
- FPGA variations
- Internal logic blocks.
- Designing with FPGAs.
- Specifics of Xilinx Virtex-E series.

FPGA Overview

- Basic idea: 2D array of combination logic blocks (CL) and flip-flops (FF) with a means for the user to configure both:

  1. the interconnection between the logic blocks,

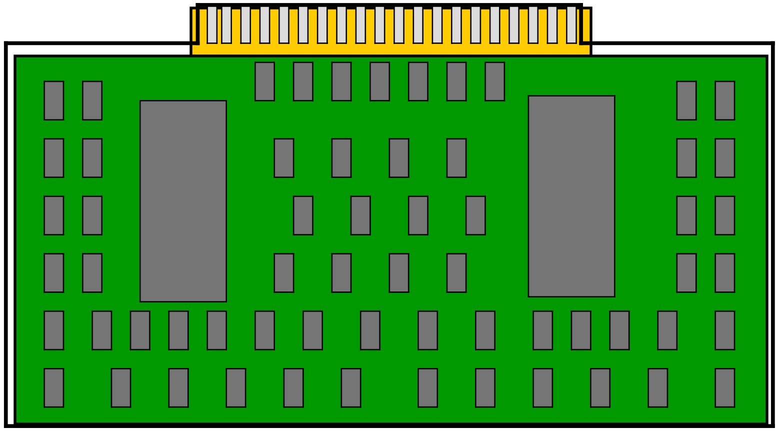
  2. the function of each block.



Simplified version of FPGA internal architecture

Why FPGAs? (1 / 5)

- By the early 1980’s most of logic circuits in typical systems were absorbed by a handful of standard large scale integrated circuits (LSI ICs).
  - Microprocessors, bus/IO controllers, system timers, ...
- Every system still needed random small “glue logic” ICs to help connect the large ICs:
  - generating global control signals (for resets etc.)
  - data formatting (serial to parallel, multiplexing, etc.)
- Systems had a few LSI components and lots of small low density SSI (small scale IC) and MSI (medium scale IC) components.



Printed Circuit (PC) board with many small SSI and MSI ICs and a few LSI ICs

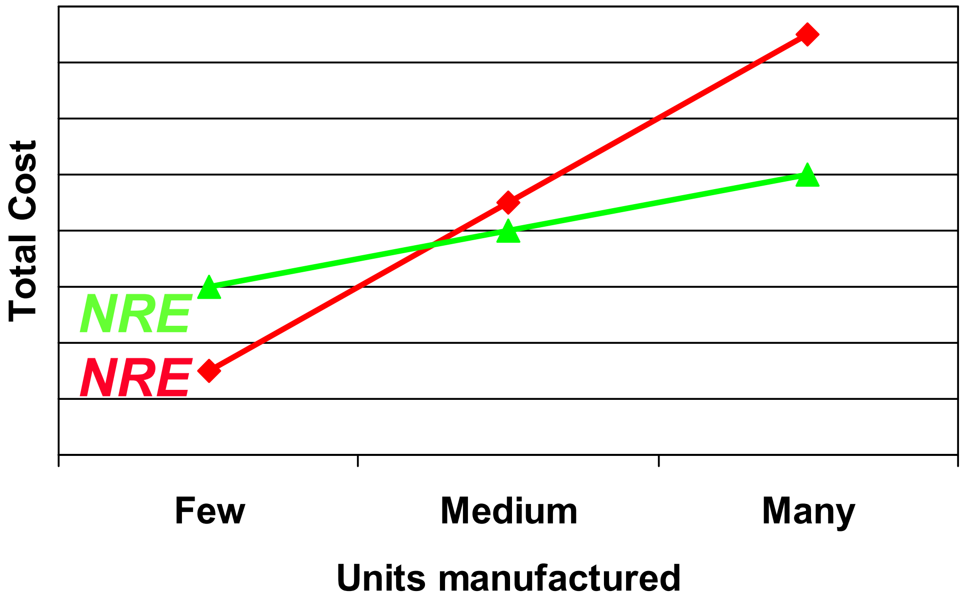
Why FPGAs? (2 / 5)

- **Custom ICs sometimes designed to replace glue logic:**
  - **reduced complexity/manufacturing cost, improved performance**
  - **But custom ICs expensive to develop, and delay introduction of product ("time to market") because of increased design time**
- **Note: need to worry about two kinds of costs:**

  1. cost of development, "Non-Recurring Engineering (NRE)", fixed

  2. cost of manufacture per unit, variable

  Usually tradeoff between NRE cost and manufacturing costs



Why FPGAs? (3 / 5)

- Therefore custom IC approach was only viable for products with very high volume (where NRE could be amortized), and not sensitive in time to market (TTM)
- FPGAs introduced as alternative to custom ICs for implementing glue logic:
  - improved PC board density vs. discrete SSI/MSI components (within around 10x of custom ICs)
  - computer aided design (CAD) tools meant circuits could be implemented quickly (no physical layout process, no mask making, no IC manufacturing), relative to Application Specific ICs (ASICs)
    (3-6 months for these steps for custom IC)
    - lowers NREs (Non Recurring Engineering)
    - shortens TTM (Time To Market)
- Because of Moore's law the density (gates/area) of FPGAs continued to grow through the 80's and 90's to the point where major data processing functions can be implemented on a single FPGA.

Why FPGAs? (4 / 5)

- FPGAs continue to compete with custom ICs for special processing functions (and glue logic) but now try to compete with microprocessors in dedicated and embedded applications
  - Performance advantage over microprocessors because circuits can be customized for the task at hand. Microprocessors must provide special functions in software (many cycles)
- MICRO: Highest NRE, SW: fastest TTM
- ASIC: Highest performance, worst TTM
- FPGA: Highest cost per chip (unit cost)

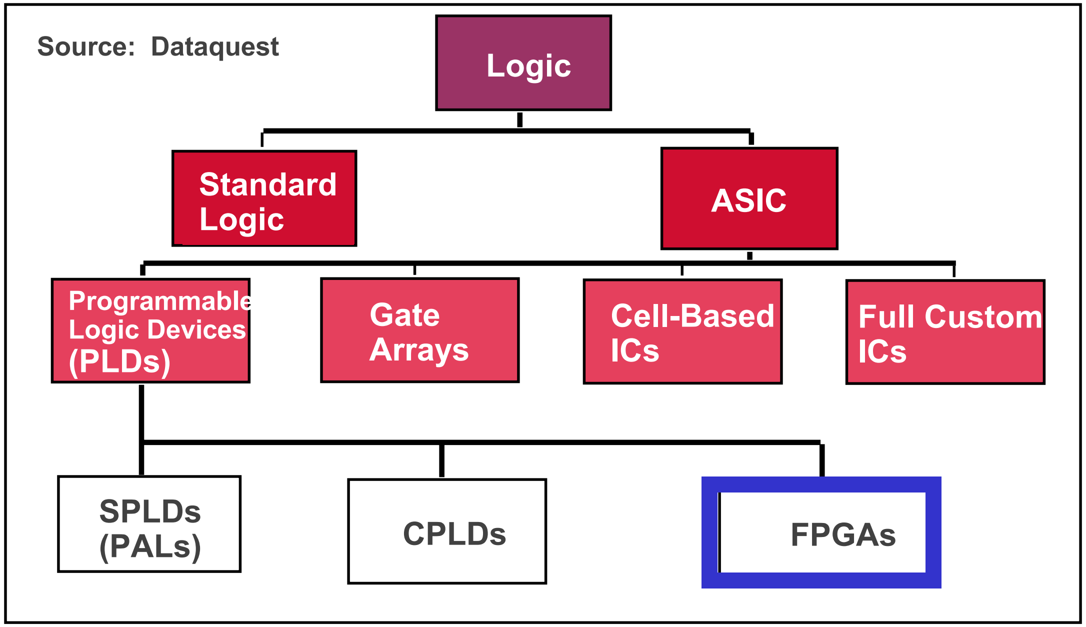
Why FPGAs? (5 / 5)

- As Moore's Law continues, FPGAs work for more applications as both can do more logic in 1 chip and faster
- Can easily be "patched" vs. ASICs
- Perfect for courses:
  - Can change design repeatedly
  - Low TTM yet reasonable speed
- With Moore's Law, now can do full CS 152 project easily inside 1 FPGA

Administrivia

- Prerequisite Quiz Results
- Lab 1 due tomorrow
- How many bought $37 PRS Transmitor ? from behind ASUC textbook desk (Chem 1A, CS 61ABC, 160)
    - Can sell back to bookstore

Where are FPGAs in the IC Zoo?



Acronyms

SPLD = Simple Prog. Logic Device
PAL = Prog. Array of Logic
CPLD = Complex PLD
FPGA = Field Prog. Gate Array

(Standard logic is SSI or MSI buffers, gates)

Common Resources

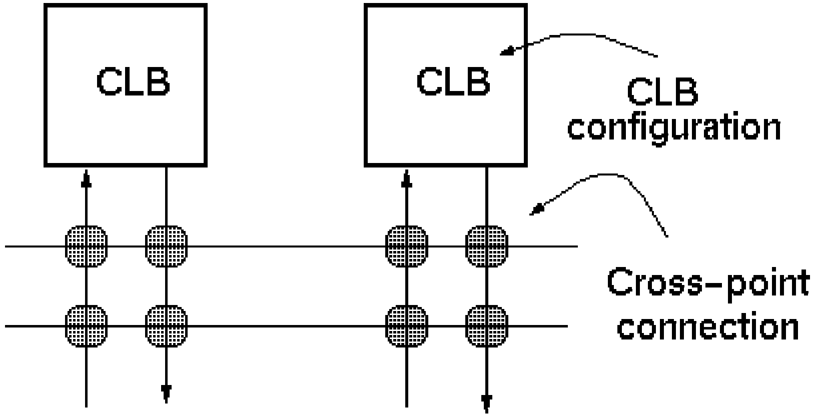
Configurable Logic Blocks (CLB)
- **Memory Look-Up Table**
- **AND-OR planes**
- **Simple gates**

Input / Output Blocks (IOB)
- **Bidirectional, latches, inverters, pullup/pulldowns**

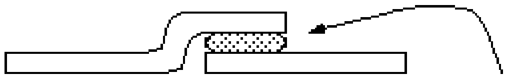
Interconnect or Routing
- **Local, internal feedback, and global**

FPGA Variations

- Families of FPGA’s differ in:
  - physical means of implementing user programmability,
  - arrangement of interconnection wires, and
  - basic functionality of logic blocks
- Most significant difference is in the method for providing flexible blocks and connections:



- **Anti-fuse based** (ex: Actel)



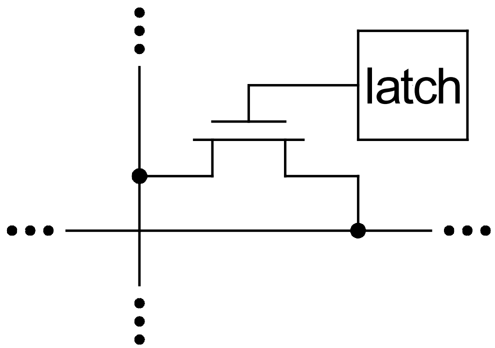
\+ Non-volatile, relatively small

\- fixed (non-reprogrammable)

(Almost used in 150 Lab: only 1-shot at getting it right!)

User Programmability

- **Latch-based** (Xilinx, Altera, …)



  +reconfigurable

  - volatile

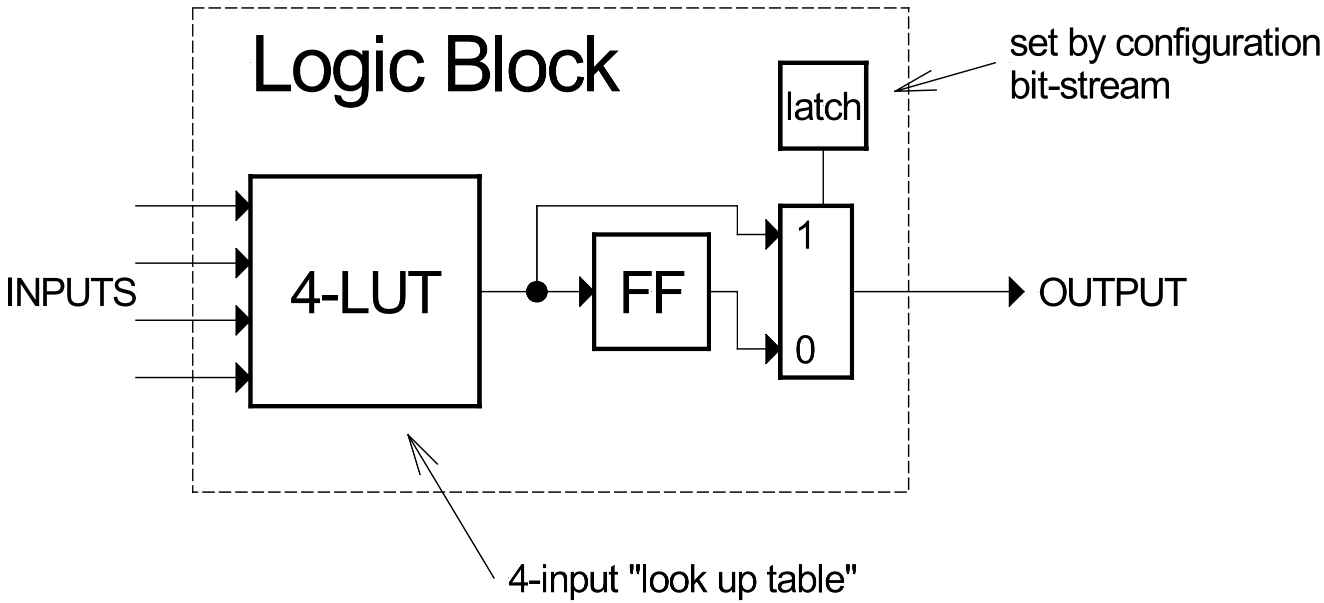
  - relatively large die size

- Note: Today 90% die is interconnect, 10% is gates

- Latches are used to:
  1. make or break cross-point connections in interconnect
  2. define function of logic blocks
  3. set user options:
     - within the logic blocks
     - in the input/output blocks
     - global reset/clock

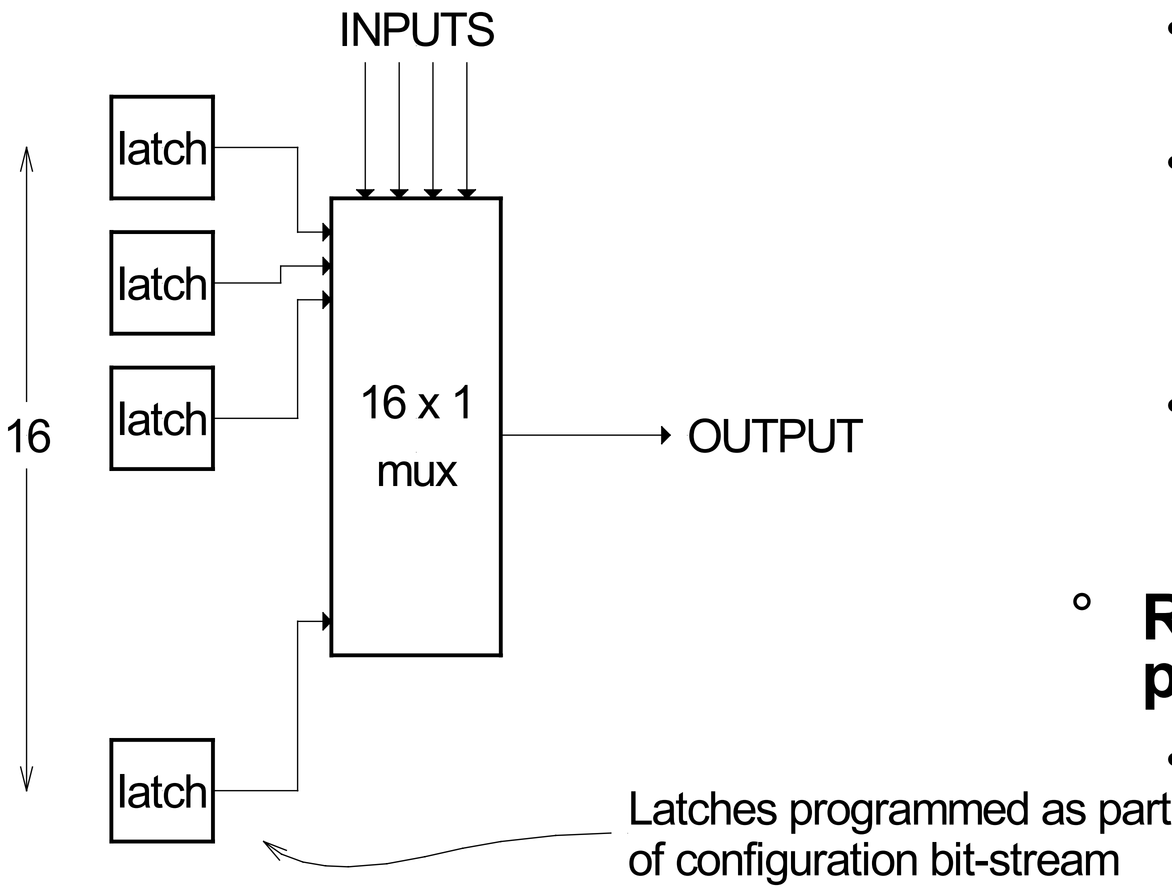
- "Configuration bit stream" loaded under user control:
  - All latches are strung together in a shift chain
  - "Programming" => creating bit stream

Idealized FPGA Logic Block



- 4-input ***Look Up Table*** **(4-*****LUT*****)**
  - implements combinational logic functions
- Register
  - optionally stores output of LUT
  - Latch determines whether read reg or LUT

4-LUT Implementation



- **n-bit LUT is actually implemented as a 2^n x 1 memory:**
  - **inputs choose one of 2^n memory locations.**
  - **memory locations (latches) are normally loaded with values from user's configuration bit stream.**
  - **Inputs to mux control are the CLB (Configurable Logic Block) inputs.**
- **Result is a general purpose "logic gate".**
  - **n-LUT can implement *any* function of n inputs!**

LUT as general logic gate

- An n-lut as a direct implementation of a function **truth-table**
- Each latch location holds value of function corresponding to one input combination

Example: 2-lut

INPUTS	AND	OR
00	0	0
01	0	1
10	0	1
11	1	1

• • •

Implements *any* function of 2 inputs.

How many functions of n inputs?

Example: 4-lut

INPUTS		
0000	F(0,0,0,0)	← store in 1st latch
0001	F(0,0,0,1)	← store in 2nd latch
0010	F(0,0,1,0)	←
0011	F(0,0,1,1)	←
0011		
0100	•	
0101	•	
0110	•	
0111		
1000		
1001		
1010		
1011		
1100		
1101		
1110		
1111		

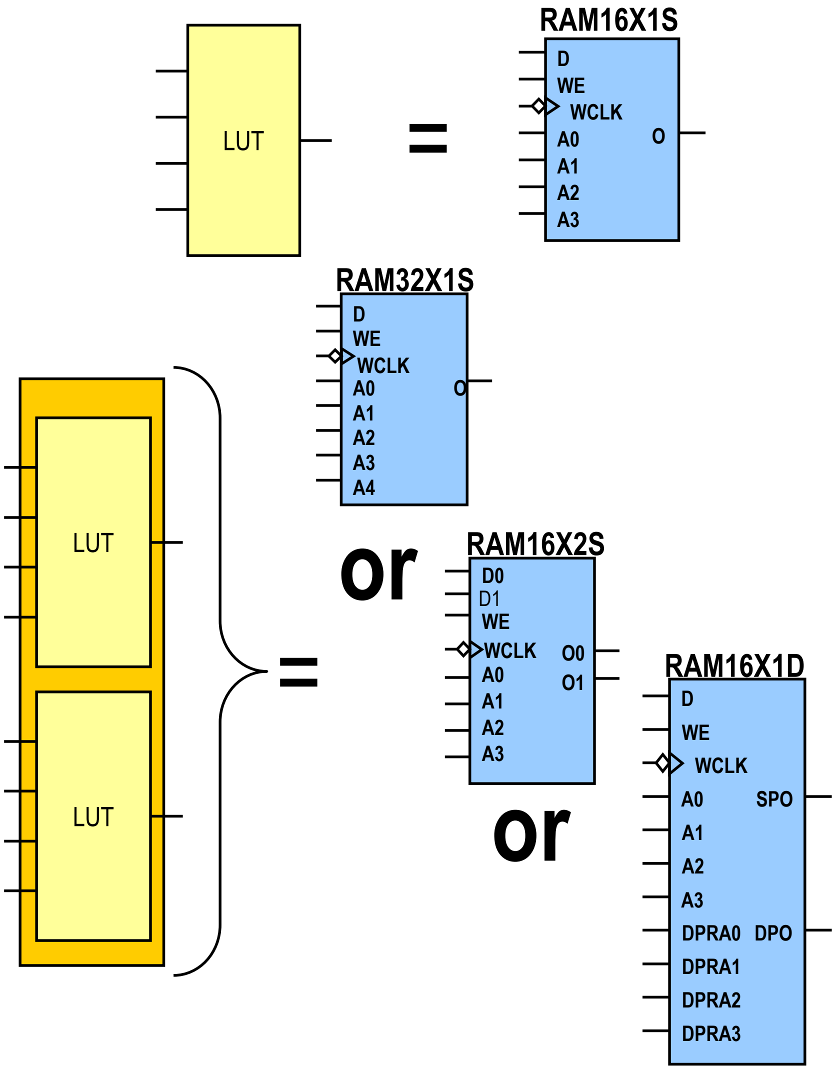
More functionality for "free"?

- Given basic idea
  - LUT built from RAM
  - Latches connected as shift register
- What other functions could be provided at very little extra cost?

1. Using CLB latches as little RAM vs. logic
2. Using CLB latches as shift register vs. logic

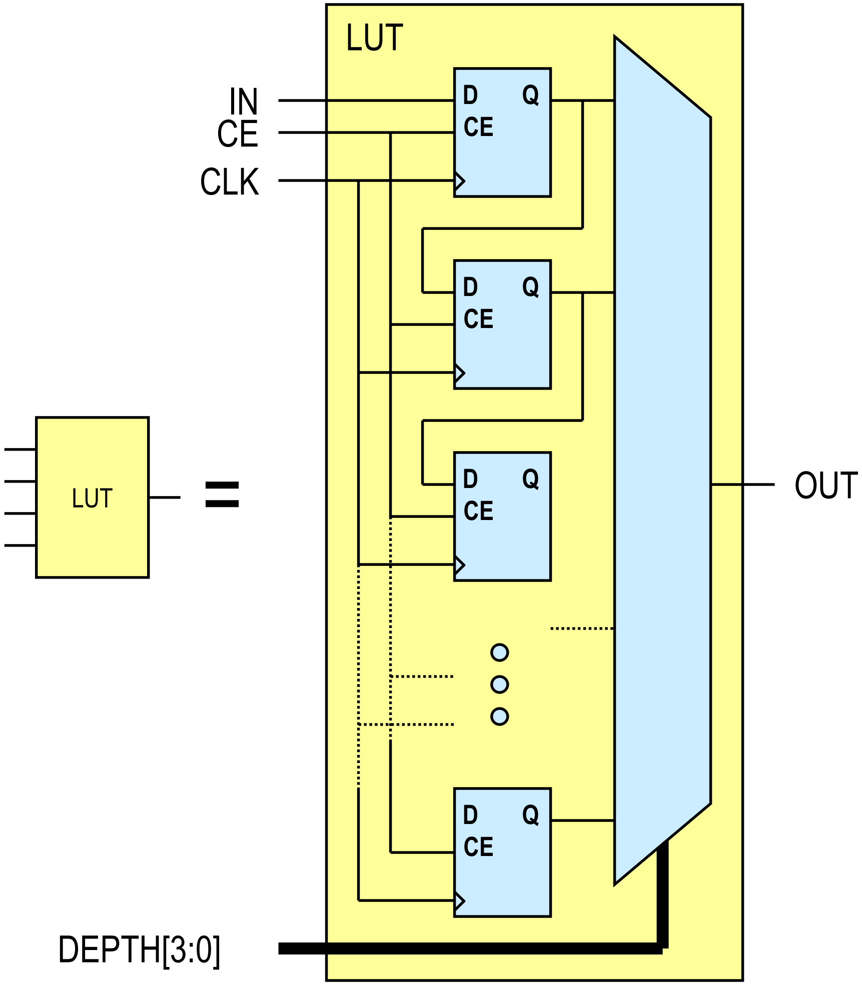
1. "Distributed RAM"

- CLB LUT configurable as Distributed RAM
  - A LUT equals 16x1 RAM
  - Implements Single and Dual-Ports
  - Cascade LUTs to increase RAM size
- Synchronous write
- Synchronous/Asynchronous read
  - Accompanying flip-flops used for synchronous read

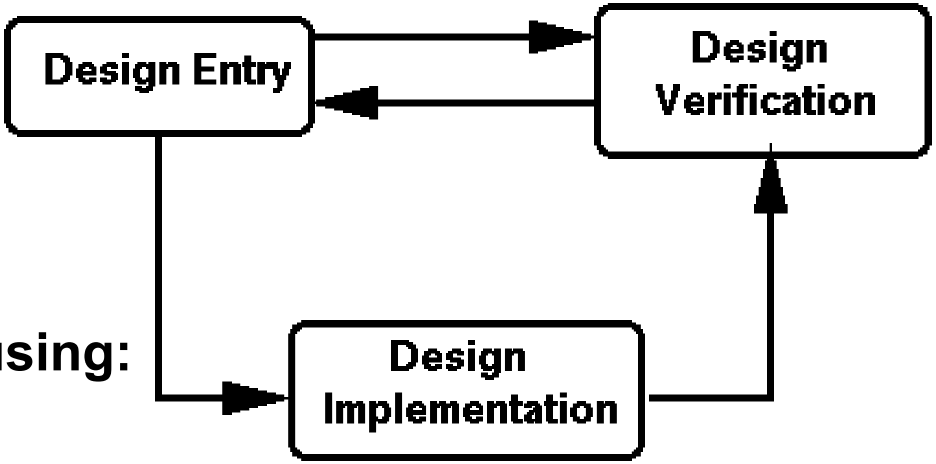


2. Shift Register

- Each LUT can be configured as shift register
  - Serial in, serial out
- Saves resources: can use less than 16 FFs
- Faster: no routing
- Note: CAD tools determine with CLB used as LUT, RAM, or shift register, rather than up to designer



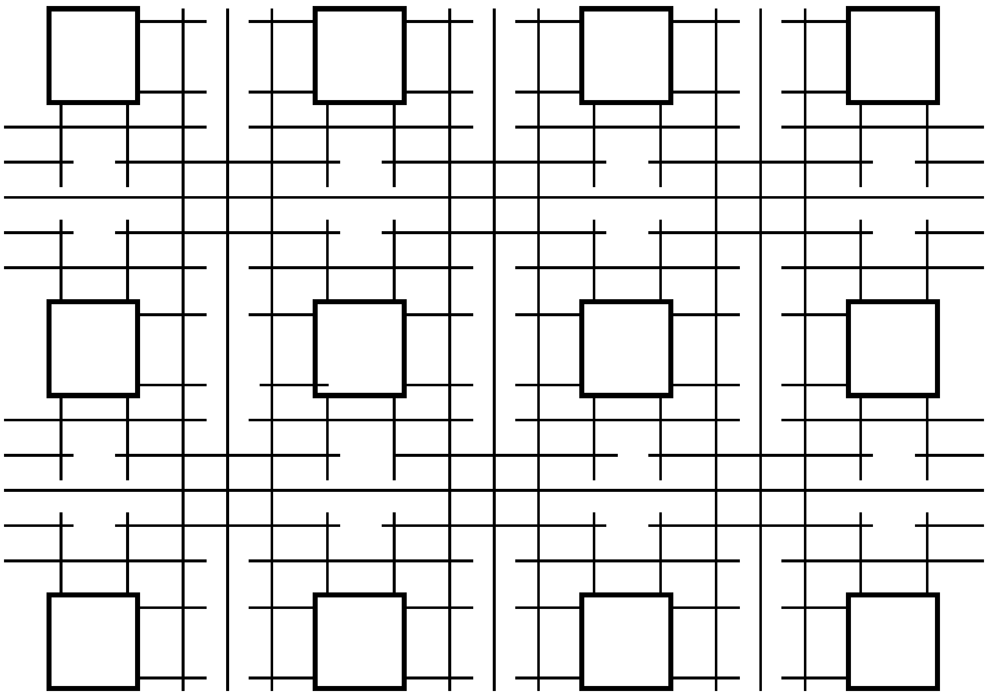
How Program: FPGA Generic Design Flow



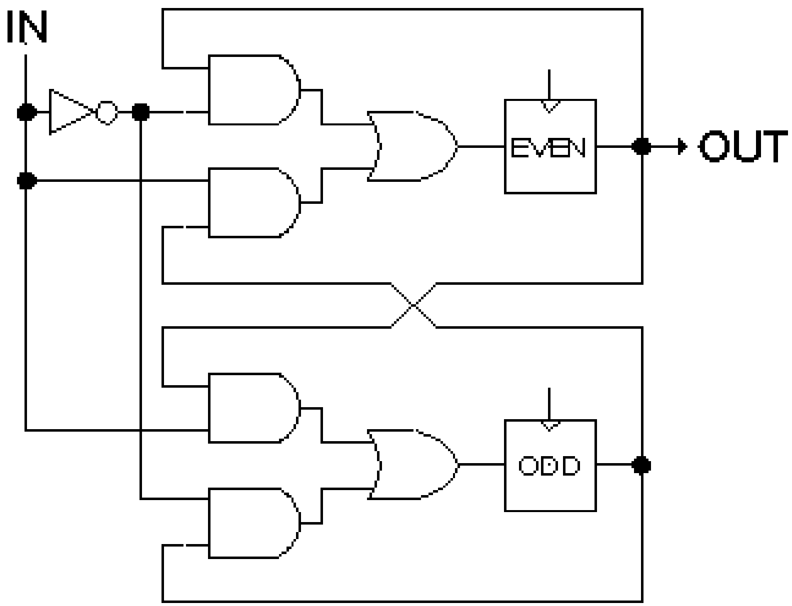
- **Design Entry:**
  - **Create your design files using:**
    - **schematic editor or**
    - **hardware description language (Verilog, VHDL)**
- **Design "implementation" on FPGA:**
  - ***Partition, place, and route*** **("PPR") to create bit-stream file**
  - **Divide into CLB-sized pieces, place into blocks, route to blocks**
- **Design verification:**
  - **Use Simulator to check function,**
  - **Other software determines max clock frequency.**
  - **Load onto FPGA device (cable connects PC to board)**
    - **check operation at full speed in real environment.**

Example Partition, Placement, and Route

- Idealized FPGA structure:



- Example Schematic Circuit:



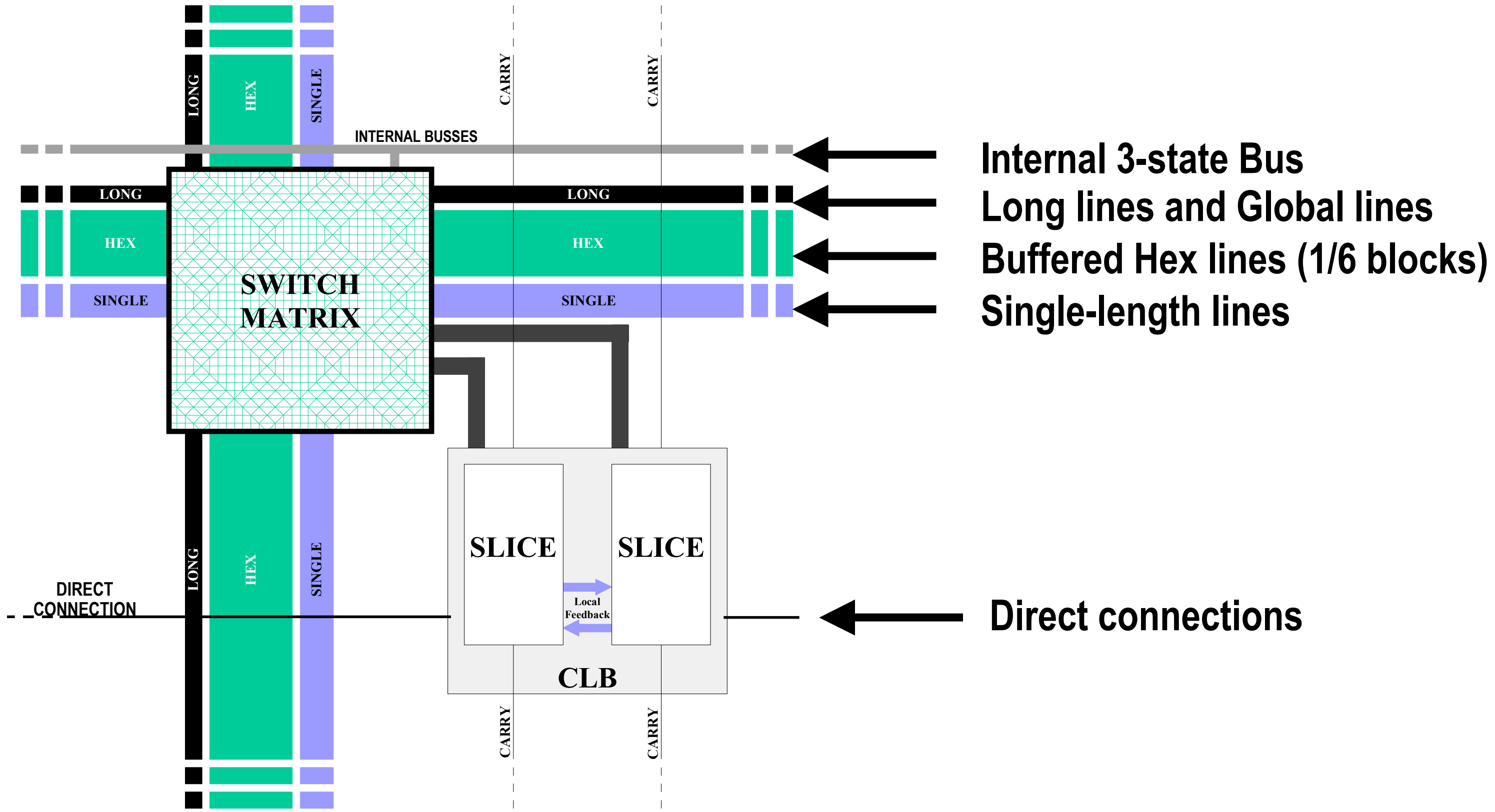
Circuit combinational logic must be "covered" by 4-input 1-output "gates".

Flip-flops from circuit must map to FPGA flip-flops.
(Best to preserve "closeness" to CL to minimize wiring.)

Placement in general attempts to minimize wiring.

Xilinx Vittex-E Routing Hierarchy

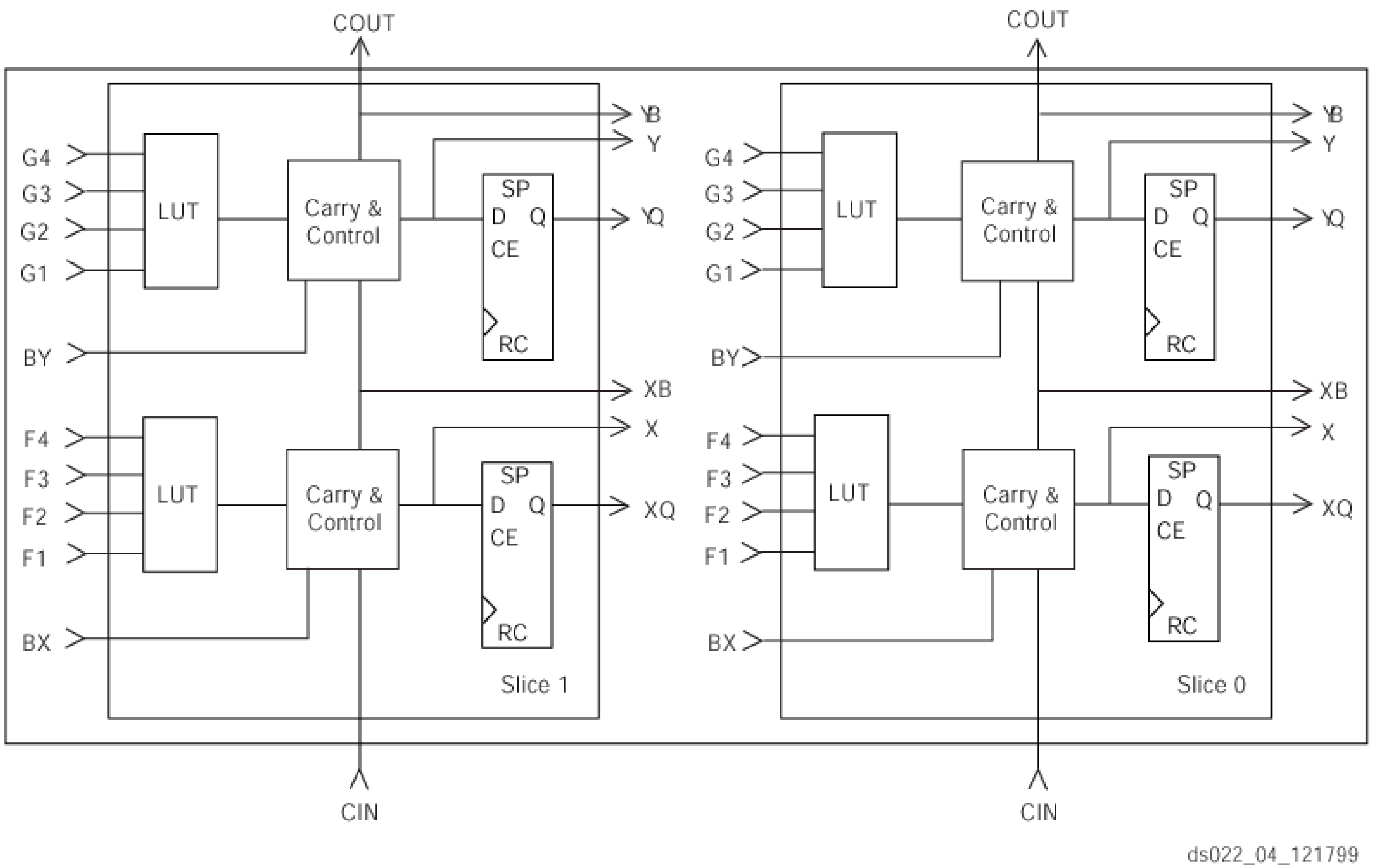
Note: CAD tools do PPR, not designers



- 24 single-length lines
  - Route GRM signals to adjacent GRMs in 4 directions
- 96 buffered hex lines
  - Route GRM (general routing matrix) signals to another GRMs six blocks away in each of the 4 directions
- 12 buffered Long lines
  - Routing across top and bottom, left and right

Virtex-E Configurable Logic Block (CLB)

2 “logic slices” / CLB, two 4-LUTs / slice => Four 4-LUTs / CLB



Peer Instruction

° How would you place ASIC, FPGA, and Microprocessors+software from best to worst?

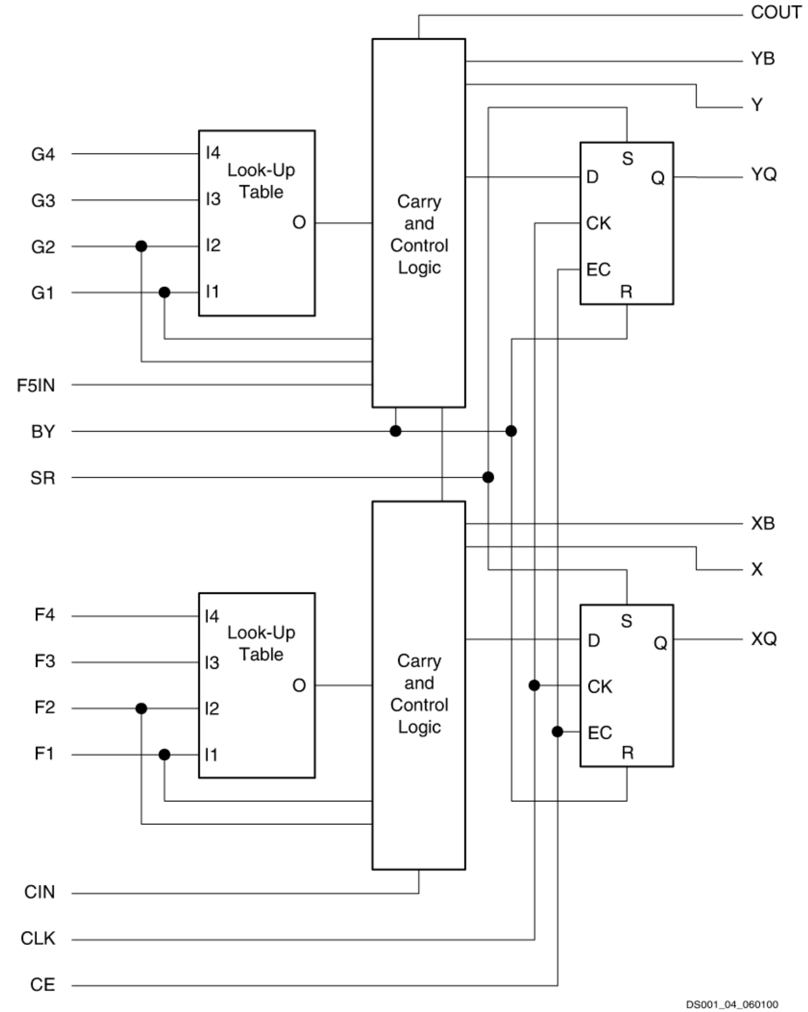
- Performance?
- Non Recurring Engineering?
- Unit cost?
- Time To Market?

1. ASIC, FPGA, MICRO
2. ASIC, MICRO, FPGA
3. FPGA, ASIC, MICRO
4. FPGA, MICRO, ASIC
5. MICRO, ASIC, FPGA
6. MICRO, FPGA, ASIC

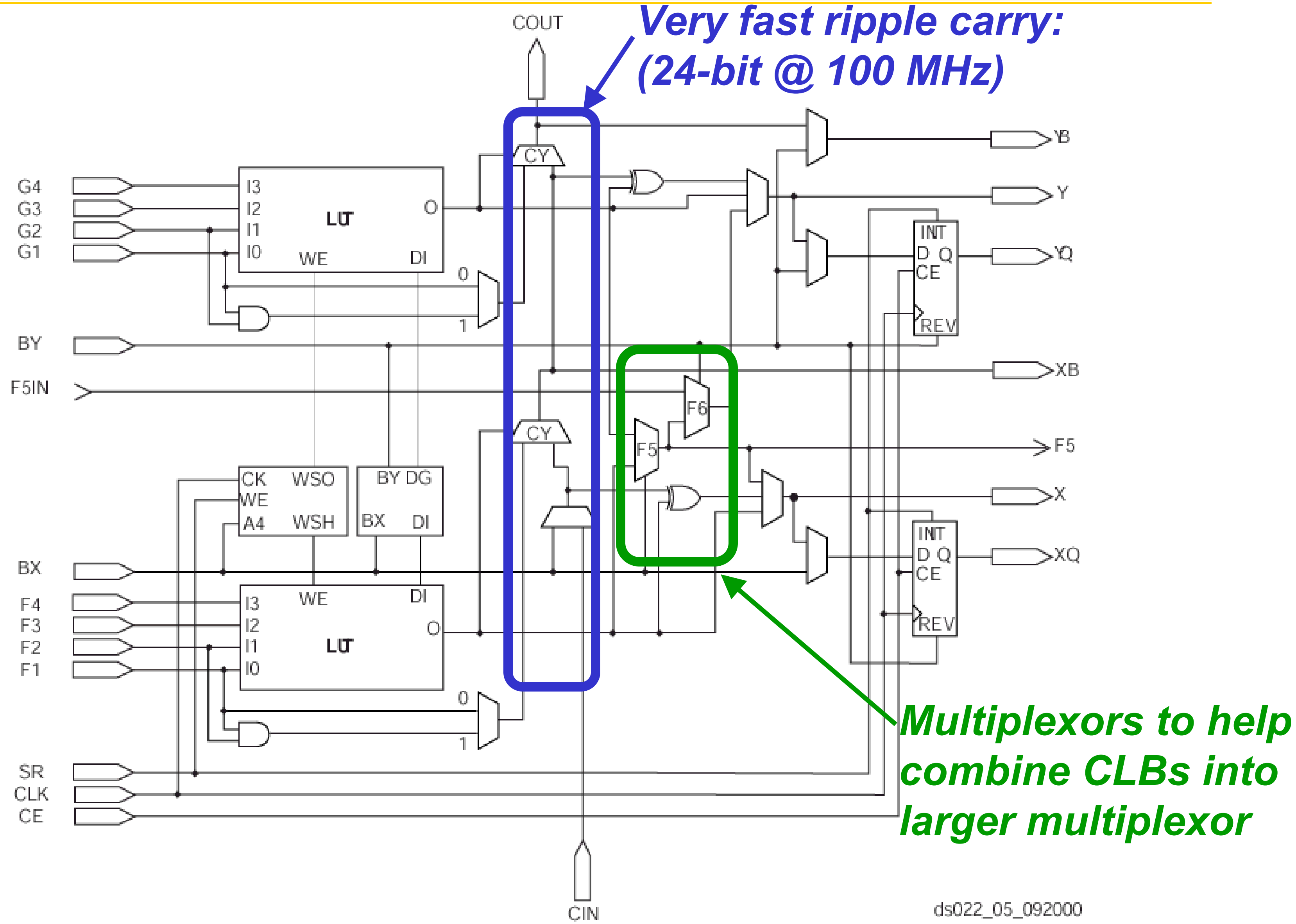
Virtex-E CLB Slice Structure

° Each slice contains two sets of the following:

- Four-input LUT
  - Any 4-input logic function
  - Or 16-bit x 1 sync RAM
  - Or 16-bit shift register
- Carry & Control
  - Fast arithmetic logic
  - Multiplexer logic
  - Multiplier logic
- Storage element
  - Latch or flip-flop
  - Set and reset
  - True or inverted inputs
  - Sync. or async. control

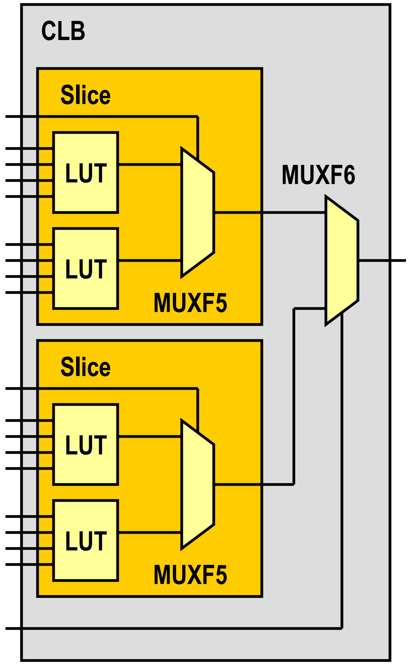


Details of Virtex-E Slice

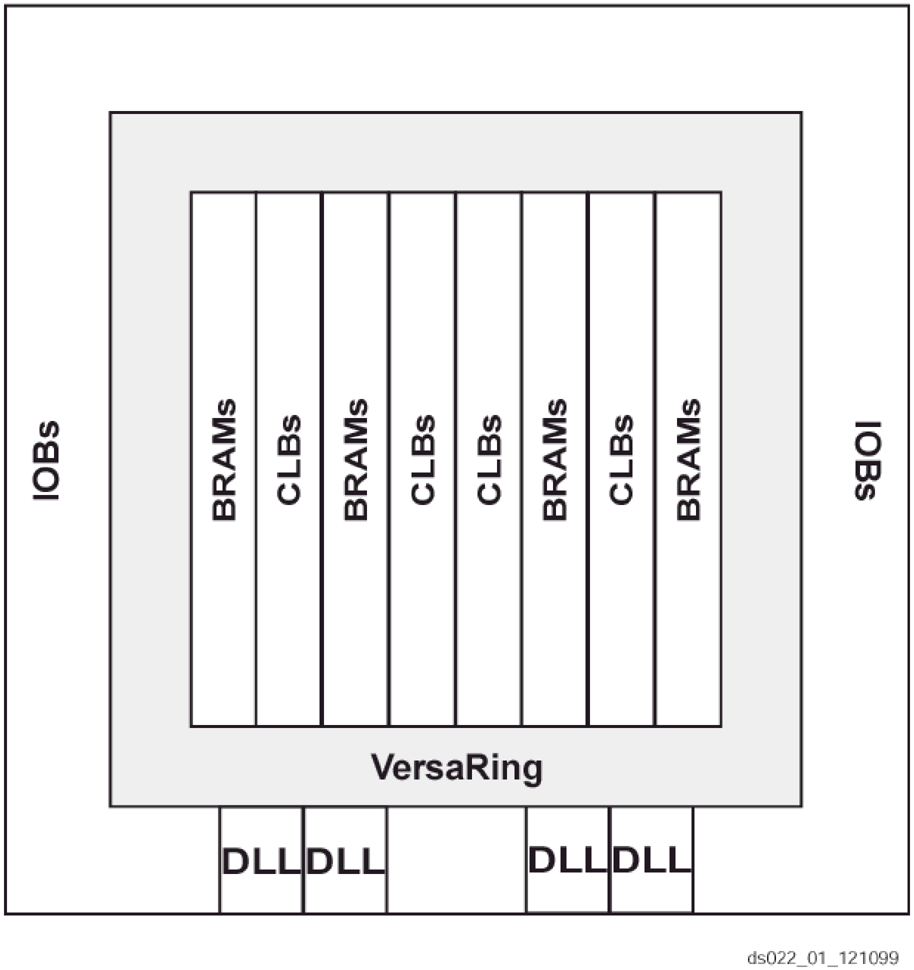


Virtex-E Dedicated Expansion Multiplexers

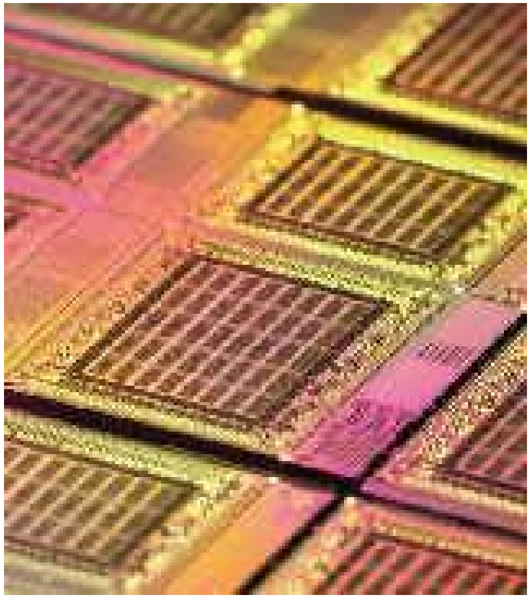
- Since 4-LUT has 4 inputs, max is 2:1 Mux (2 inputs, 1 control line)
- MUXF5 combines 2 LUTs to create
  - 4x1 multiplexer
  - Or any 5-input function (5-LUT)
  - Or selected functions up to 9 inputs
- MUXF6 combines 2 slices to form
  - 8x1 multiplexer
  - Or any 6-input function (6-LUT)
  - Or selected functions up to 19 inputs
- Dedicated muxes are faster and more space efficient



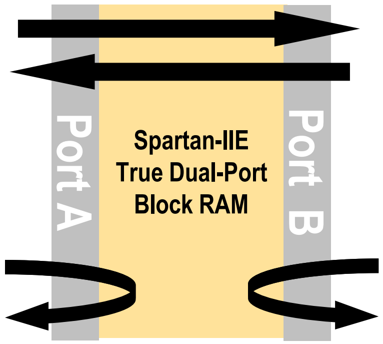
Xilinx Virtex-E Chip Floorplan



- Input / Output Blocks (IOBs)
- Configurable Logic Blocks (CLBs)
- Block RAMs (BRAMs) (discussed soon)
- Delay Locked Loop (DLL) (discussed soon)
- "VersaRing" =



Block RAM (Extra RAM not using LUTs)

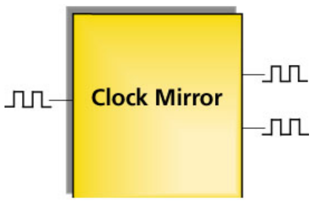


- Most efficient memory implementation
  - Dedicated blocks of memory
- Ideal for most memory requirements
  - Virtex-E XCV2000 has 160? blocks
    - 4096 bits per blocks
  - Use multiple blocks for larger memories
- Builds both single and true dual-port RAMs
- CORE Generator provides custom-sized block RAMs
  - Quickly generates optimized RAM implementation

Virtex-E Block RAM

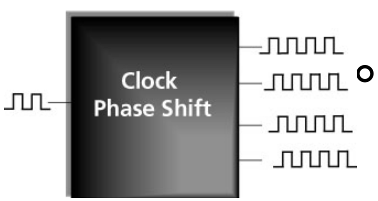
- Flexible 4096-bit block… Variable aspect ratio
  - 4096 x 1
  - 2048 x 2
  - 1024 x 4
  - 512 x 8
  - 256 x 16
- Increase memory depth or width by cascading blocks

Virtex-E Delay Lock Loop (DLL) Capabilities

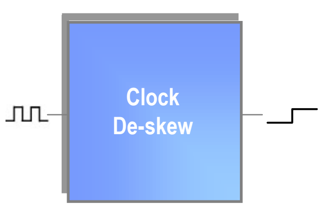


- Easy clock duplication
  - System clock distribution
  - Cleans and reconditions incoming clock

- Quick and easy frequency adjustment
- Single crystal easily generates multiple clocks



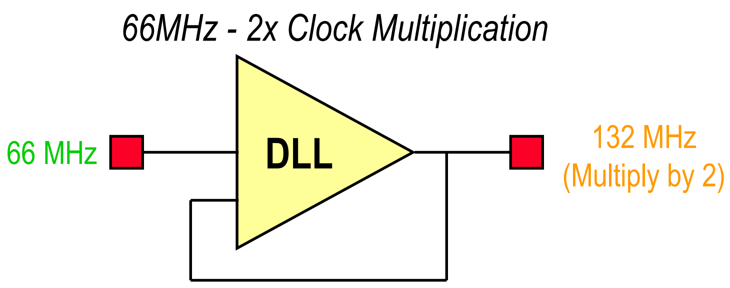
- Excellent for advance memory types

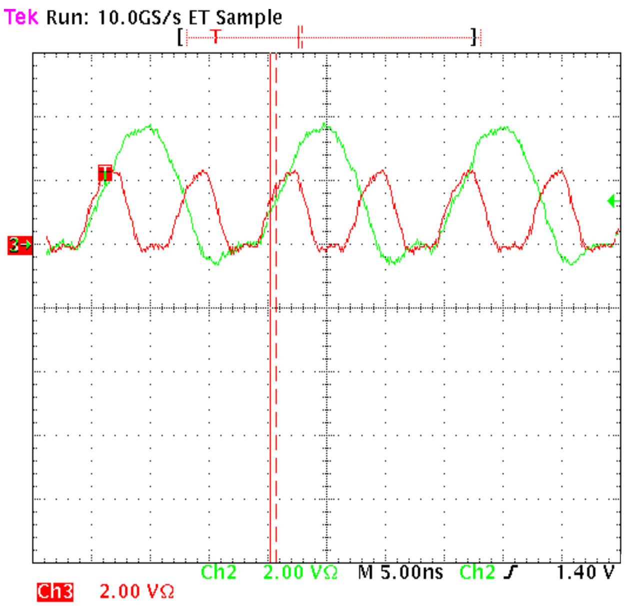


- De-skew incoming clock
- Generate fast setup and hold time or fast clock-to-outs

DLL: Multiplication of Clock Speed

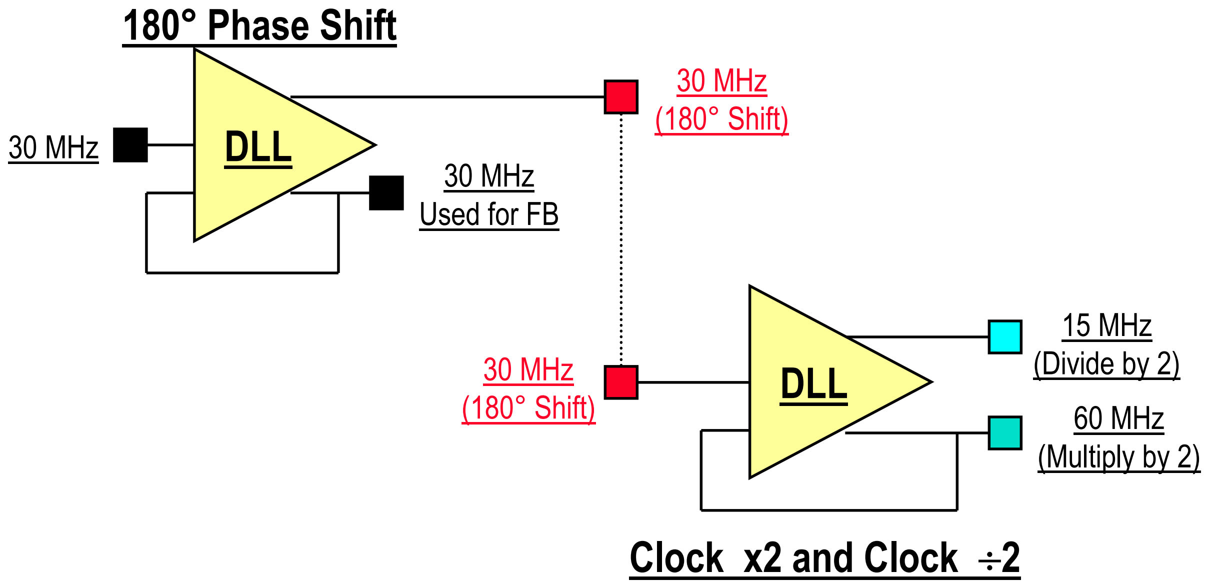
- Have faster internal clock relative to external clock source
- Use 1 DLL for 2x multiplication
- Combine 2 DLLs for 4x multiplication
- Reduce board EMI
  - Route low-frequency clock externally and multiply clock on-chip





DLL: Division of Clock Speed

- Selectable division values
  - 1.5, 2, 2.5, 3, 4, 5, 8, or 16
- Cascade DLLs to combine functions
  - Combine DLLs to multiply and divide to get desired speed
- 50/50 duty cycle correction available



Clock Management Summary

° All digital DLL Implementation
  • Input noise rejection
  • 50/50 duty cycle correction

° Clock mirror provides system clock distribution

° Multiply input clock by 2x or 4x

° Divide clock by 1.5, 2, 2.5, 3, 4, 5, 8, or 16

° De-skew clock for fast setup, hold, or clock-to-out times

Virtex-E Family of Parts

Table 1: **Virtex-E Field-Programmable Gate Array Family Members**

Device	System Gates	Logic Gates	CLB Array	Logic Cells	Differential I/O Pairs	User I/O	BlockRAM Bits	Distributed RAM Bits
XCV50E	71,693	20,736	16 x 24	1,728	83	176	65,536	24,576
XCV100E	128,236	32,400	20 x 30	2,700	83	196	81,920	38,400
XCV200E	306,393	63,504	28 x 42	5,292	119	284	114,688	75,264
XCV300E	411,955	82,944	32 x 48	6,912	137	316	131,072	98,304
XCV400E	569,952	129,600	40 x 60	10,800	183	404	163,840	153,600
XCV600E	985,882	186,624	48 x 72	15,552	247	512	294,912	221,184
XCV1000E	1,569,178	331,776	64 x 96	27,648	281	660	393,216	393,216
XCV1600E	2,188,742	419,904	72 x 108	34,992	344	724	589,824	497,664
XCV2000E	2,541,952	518,400	80 x 120	43,200	344	804	655,360	614,400
XCV2600E	3,263,755	685,584	92 x 138	57,132	344	804	753,664	812,544
XCV3200E	4,074,387	876,096	104 x 156	73,008	344	804	851,968	1,038,336

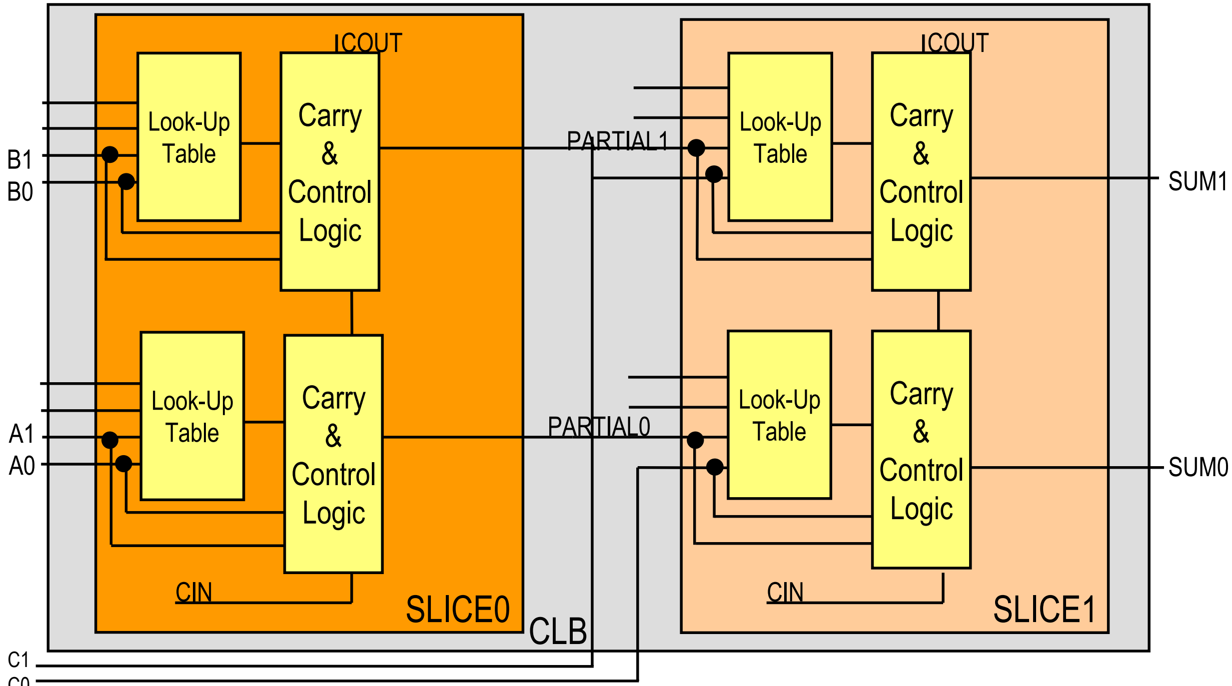
Summary: Xilinx FPGAs

° How they differ from idealized array:

- In addition to their use as general logic "gates", LUTs can alternatively be used as general purpose RAM or shift register
  - Each 4-LUT can become a 16x1-bit RAM array
- Special circuitry to speed up "ripple carry" in adders and counters
  - Therefore adders assembled by the CAD tools operate much faster than adders built from gates and LUTs alone.
- Many more wires, including tri-state capabilities.

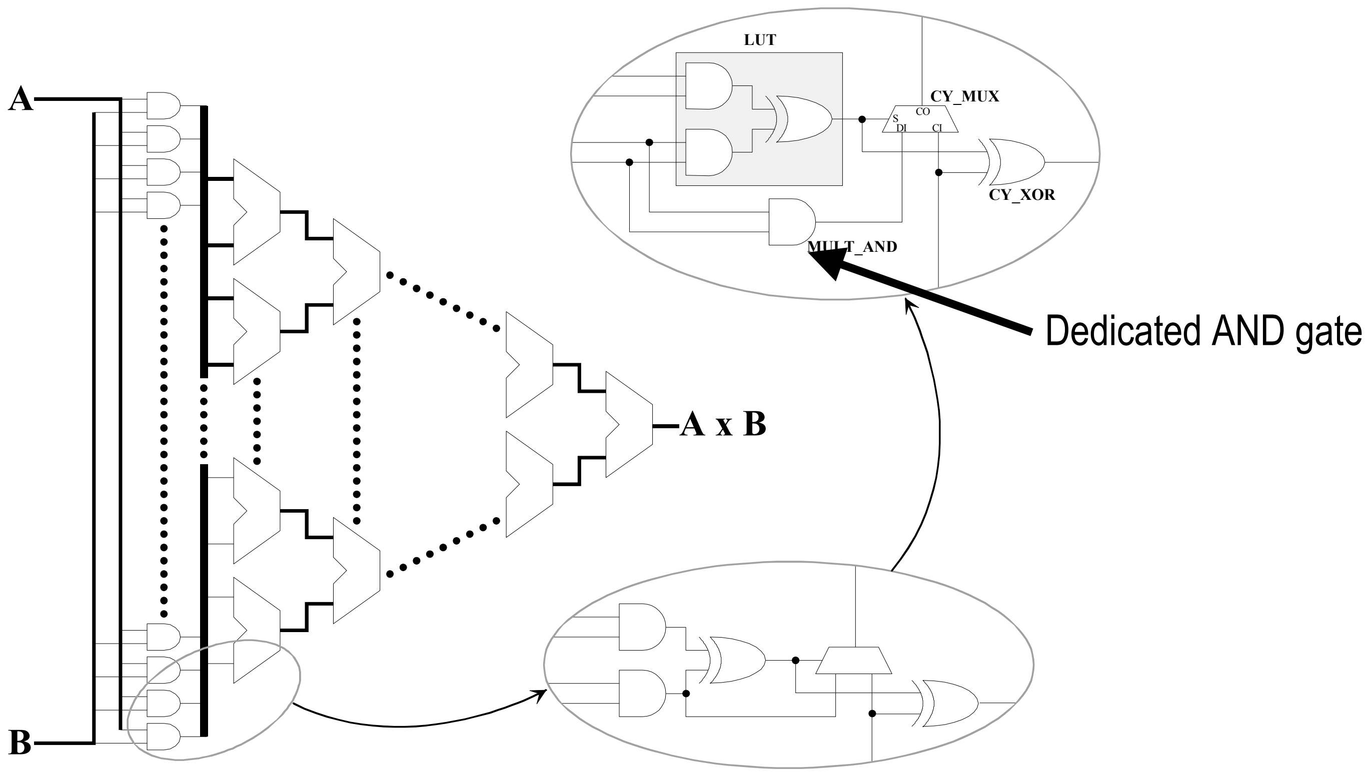
Backup Slides FYI

3 Operand Adder Function



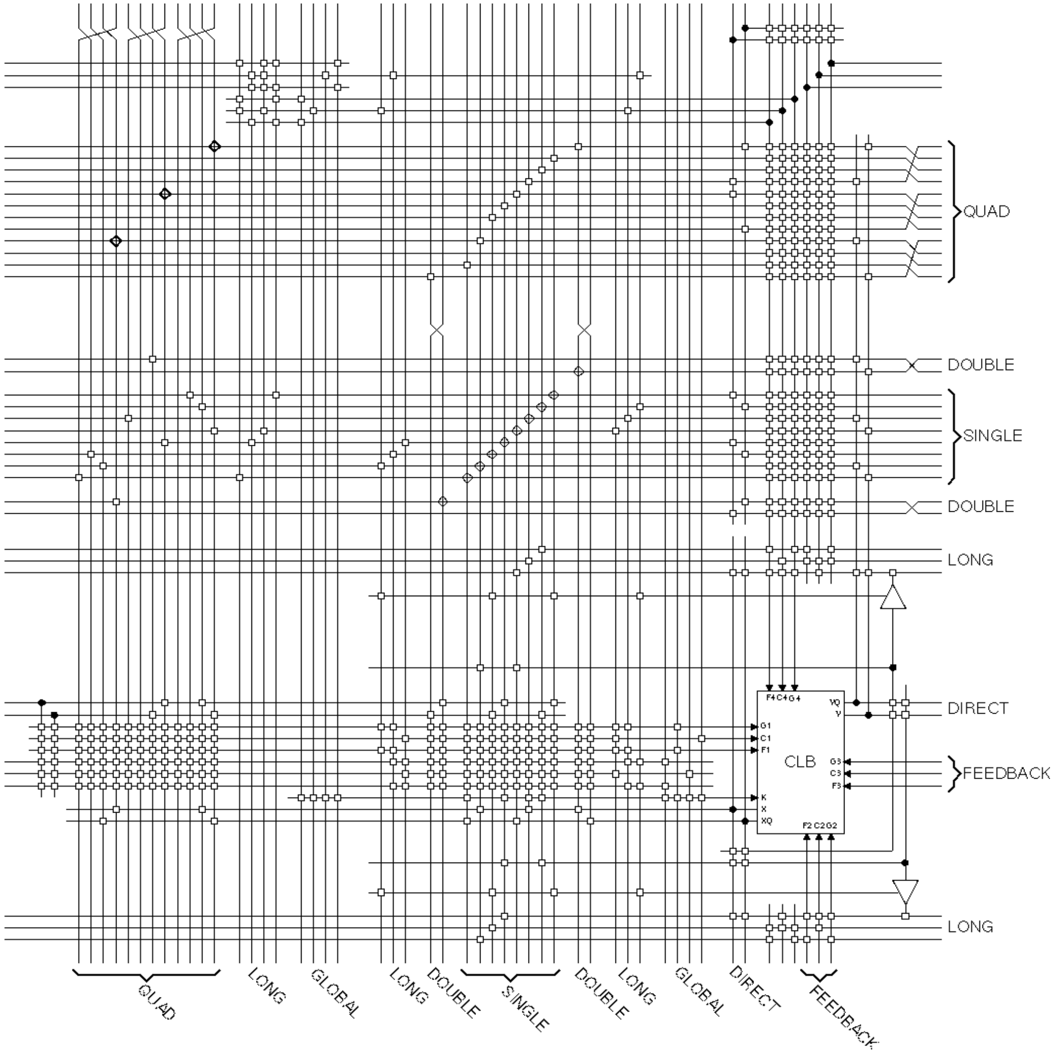
- A, B, C are two-bits wide
  - SUM = A + B + C or PARTIAL + C, where PARTIAL = A + B
  - Implementation
    - First 2-operand sum ‘A+B’ is performed in Slice 0
    - Second 2-operand sum ‘PARTIAL + C’ is performed in Slice 1
  - Fast local feedback connection within the CLB
    - Very small delay for on PARTIAL

Dedicated CLB Multiplier Logic



- Dedicated AND gate
- Highly efficient 'Shift & Add' implementation
  - For a 16x16 Multiplier
    - 30% reduction in area and one less logic level

Xilinx FPGAs (interconnect detail)



Virtex-E Input/Output block (IOB) detail