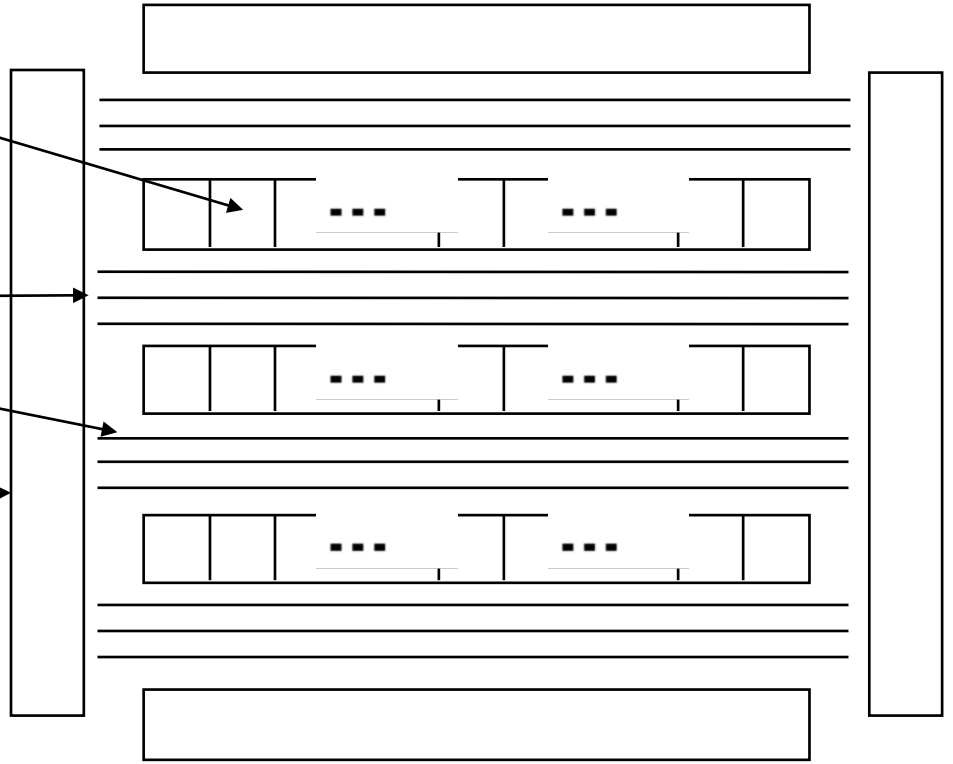
# Evolution of implementation technologies

- Logic gates (1950s-60s)
- Regular structures for two-level logic (1960s-70s)
  - muxes and decoders, PLAs
- Programmable sum-of-products arrays (1970s-80s)
  - PLDs, complex PLDs
- Programmable gate arrays (1980s-90s)
  - densities high enough to permit entirely new class of application, e.g., prototyping, emulation, acceleration

**trend toward higher levels of integration**

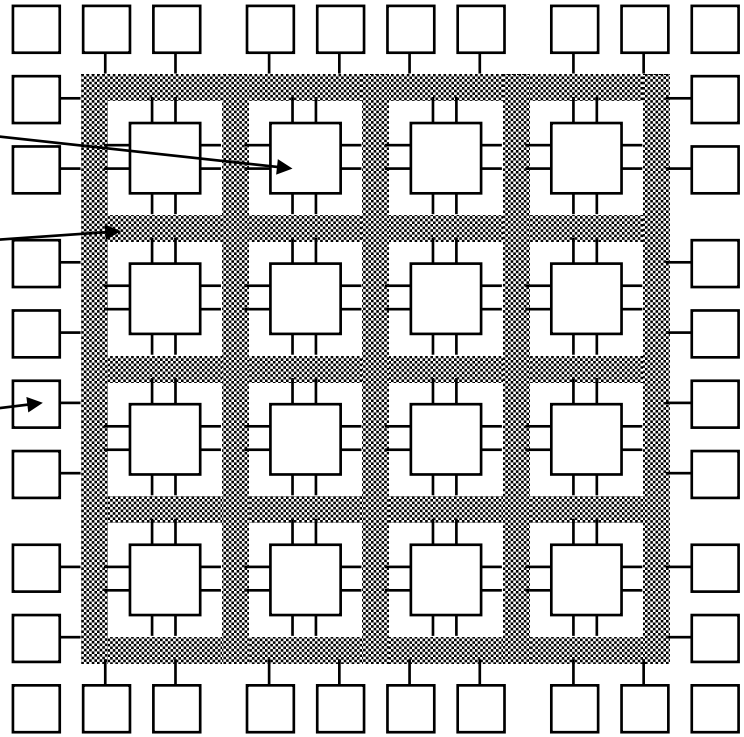
# Gate Array Technology (IBM - 1970s)

- Simple logic gates
  - combine transistors to implement combinational and sequential logic
- Interconnect
  - wires to connect inputs and outputs to logic blocks
- I/O blocks
  - special blocks at periphery for external connections
- Add wires to make connections
  - done when chip is fabbed
    - "mask-programmable"
  - construct any circuit
- Coming back as "Structured ASICs"



# Field-Programmable Gate Arrays

- Logic blocks
  - to implement combinational and sequential logic
- Interconnect
  - wires to connect inputs and outputs to logic blocks
- I/O blocks
  - special logic blocks at periphery of device for external connections



- Key questions:
  - how to make logic blocks programmable?
  - how to connect the wires?
  - *after the chip has been fabbed*

# Enabling Technology

- Cheap/fast fuse connections
  - small area (can fit lots of them)
  - low resistance wires (fast even if in multiple segments)
  - very high resistance when not connected
  - small capacitance (wires can be longer)
- Pass transistors (switches)
  - used to connect wires
  - bi-directional
- Multiplexors
  - used to connect one of a set of possible sources to input
  - can be used to implement logic functions

# Programming Technologies

- Fuse and anti-fuse
  - fuse makes or breaks link between two wires
  - typical connections are 50-300 ohm
  - one-time programmable (testing before programming?)
- EPROM and EEPROM
  - high power consumption
  - typical connections are 2K-4K ohm
  - fairly low density
- RAM-based
  - memory bit controls a switch that connects/disconnects two wires
  - typical connections are .5K-1K ohm
  - can be programmed and re-programmed easily (tested at factory)

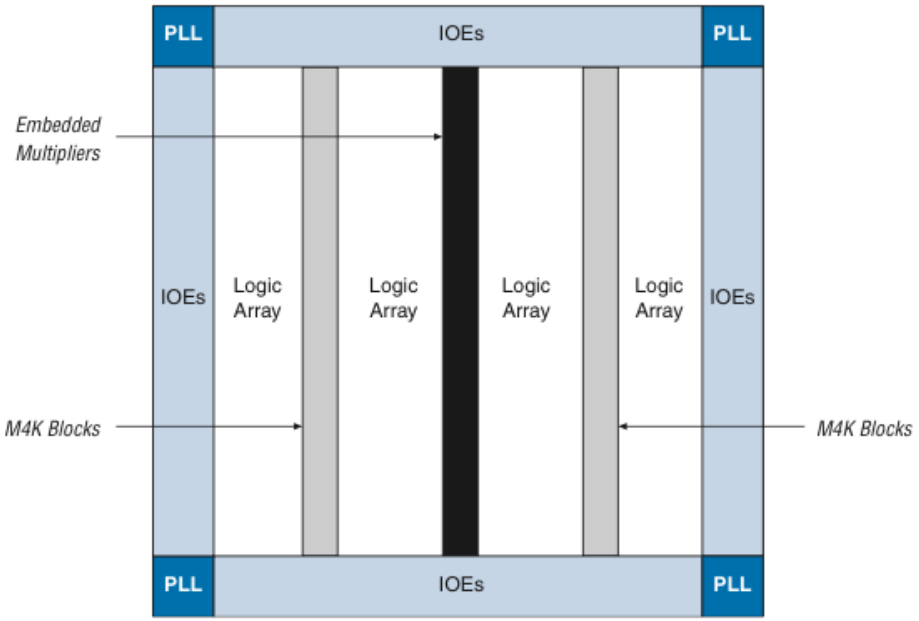
# Tradeoffs in FPGAs

- Logic block - how are functions implemented: fixed functions (manipulate inputs) or programmable?
  - support complex functions, need fewer blocks, but they are bigger so less of them on chip
  - support simple functions, need more blocks, but they are smaller so more of them on chip
- Interconnect
  - how are logic blocks arranged?
  - how many wires will be needed between them?
  - are wires evenly distributed across chip?
  - programmability slows wires down – are some wires specialized to long distances?
  - how many inputs/outputs must be routed to/from each logic block?
  - what utilization are we willing to accept? 50%? 20%? 90%?

# Altera Cyclone II (Low-cost Stratix-II FPGA)

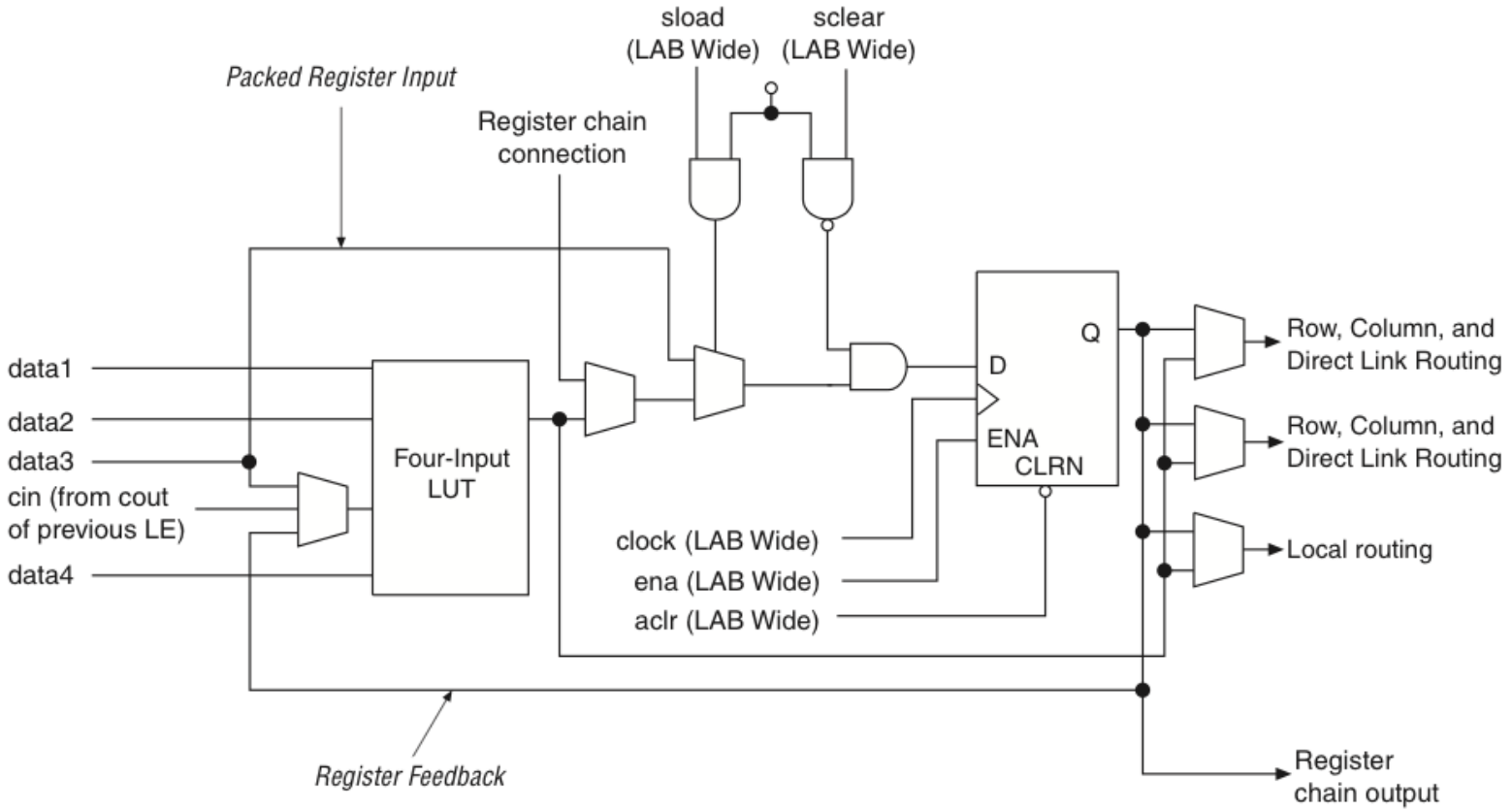
- LE is basic block
  - LUT + register + configurable routing
- LAB: 16 LEs
  - Internal carry chain
  - Shift register chain
  - Shared control signals:
    - clocks, resets, enables
  - Internal connect
- FPGA: Rows and Columns of LABs
  - With memories and multipliers
  - I/Os and PLLs on periphery

*Figure 2–1. Cyclone II EP2C20 Device Block Diagram*



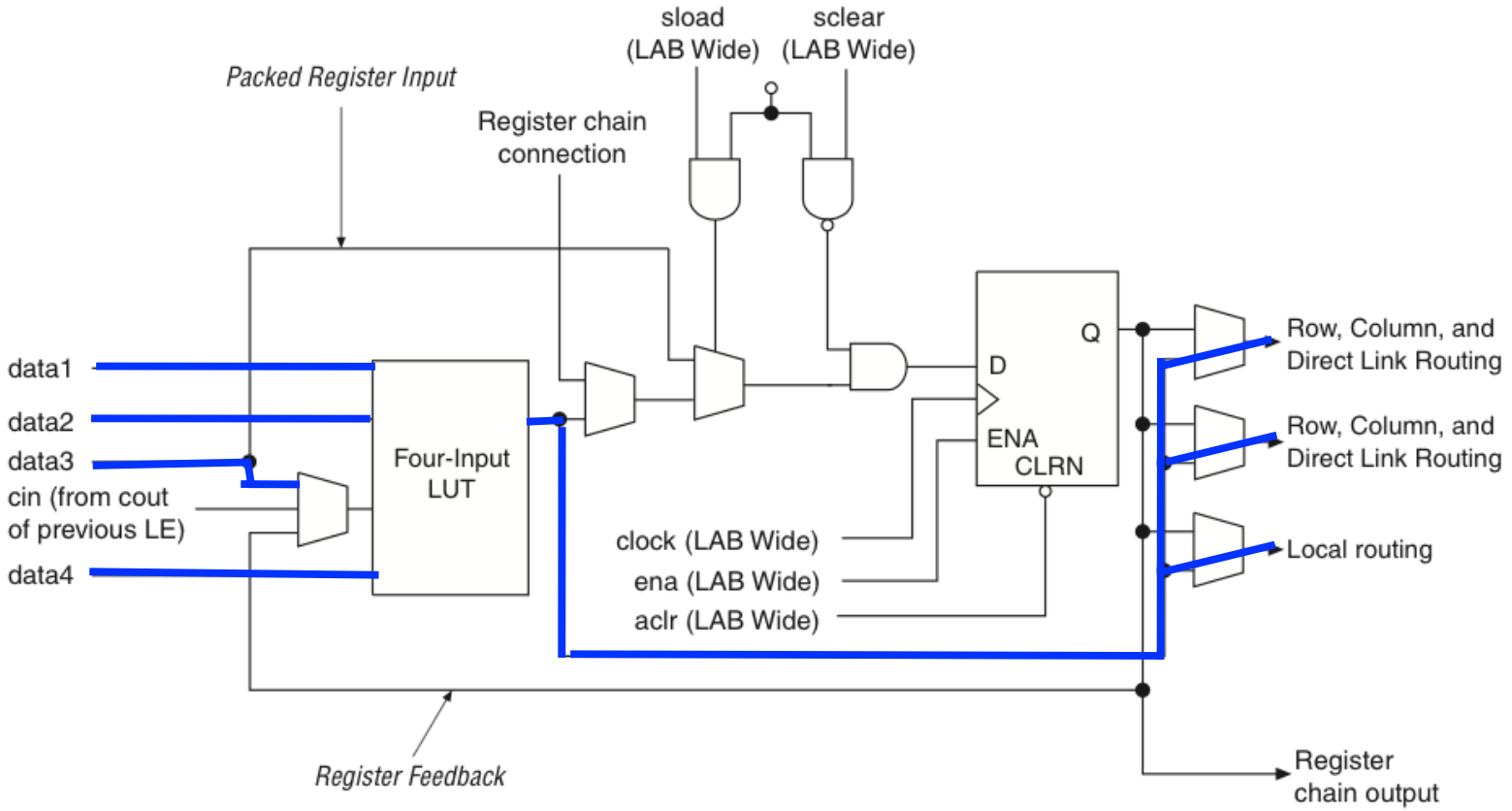
# Cyclone II LE used for Logic

***Figure 2–3. LE in Normal Mode***



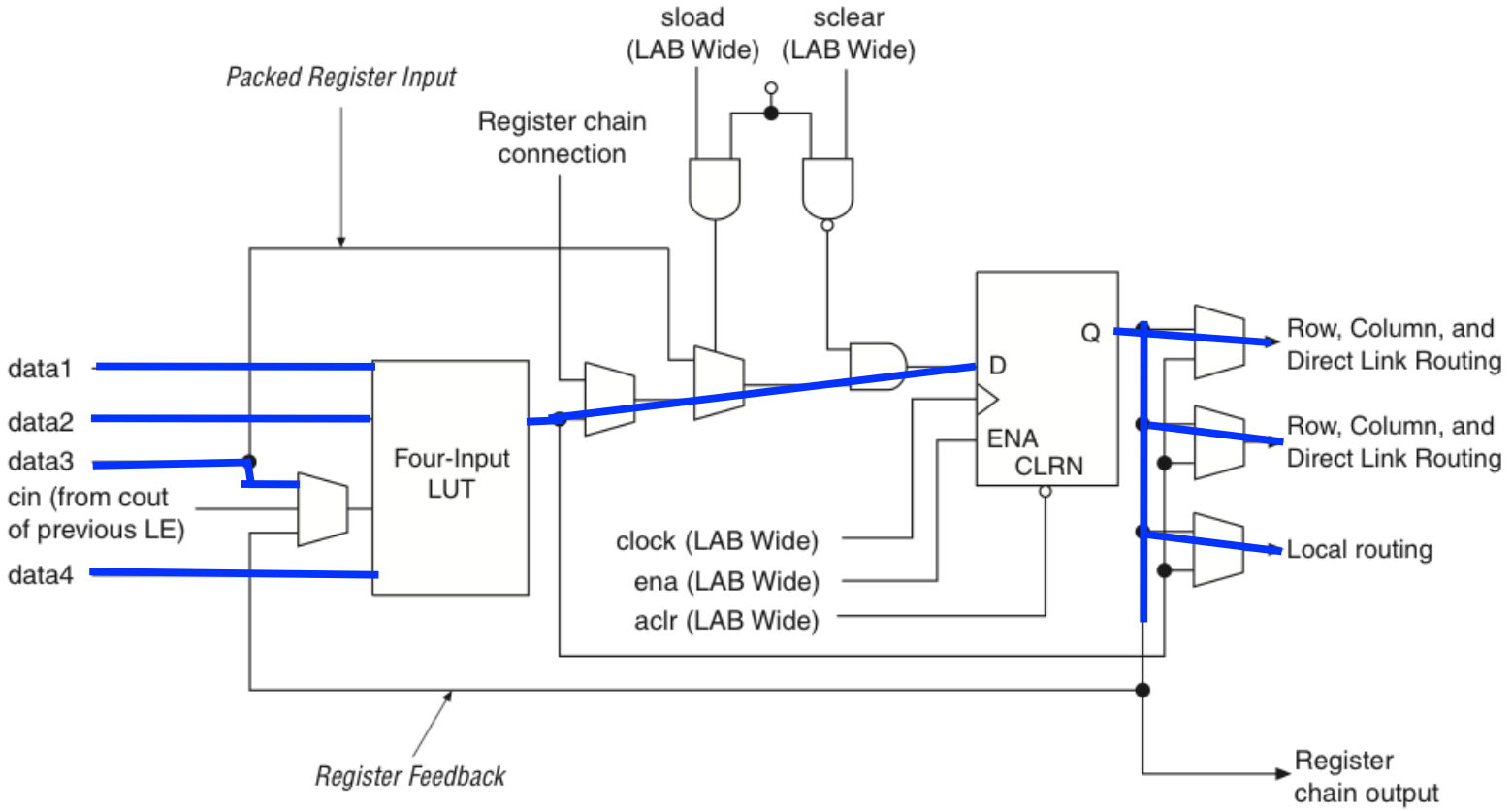
# Combinational Logic

***Figure 2–3. LE in Normal Mode***



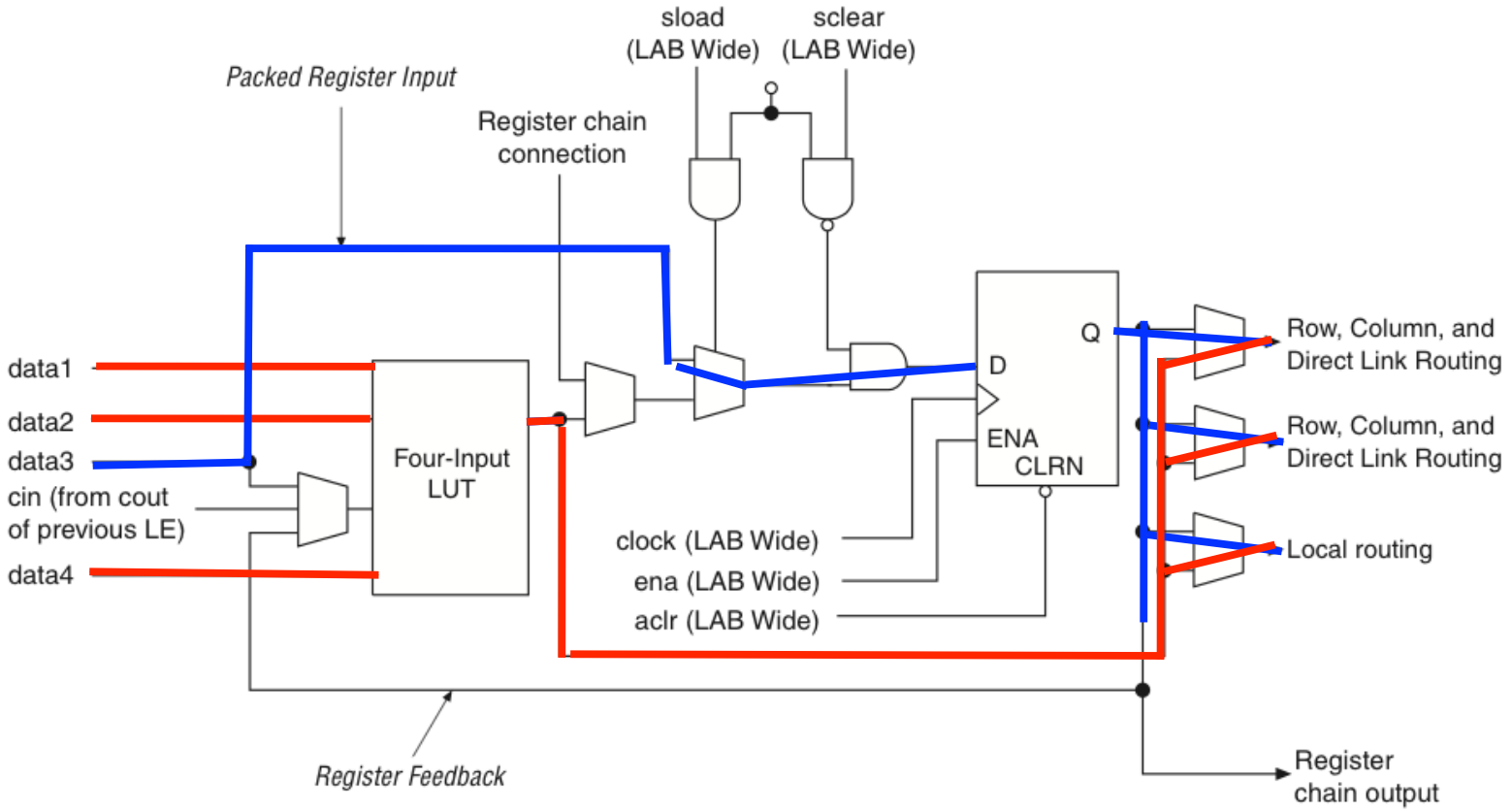
# Registered Logic

***Figure 2–3. LE in Normal Mode***



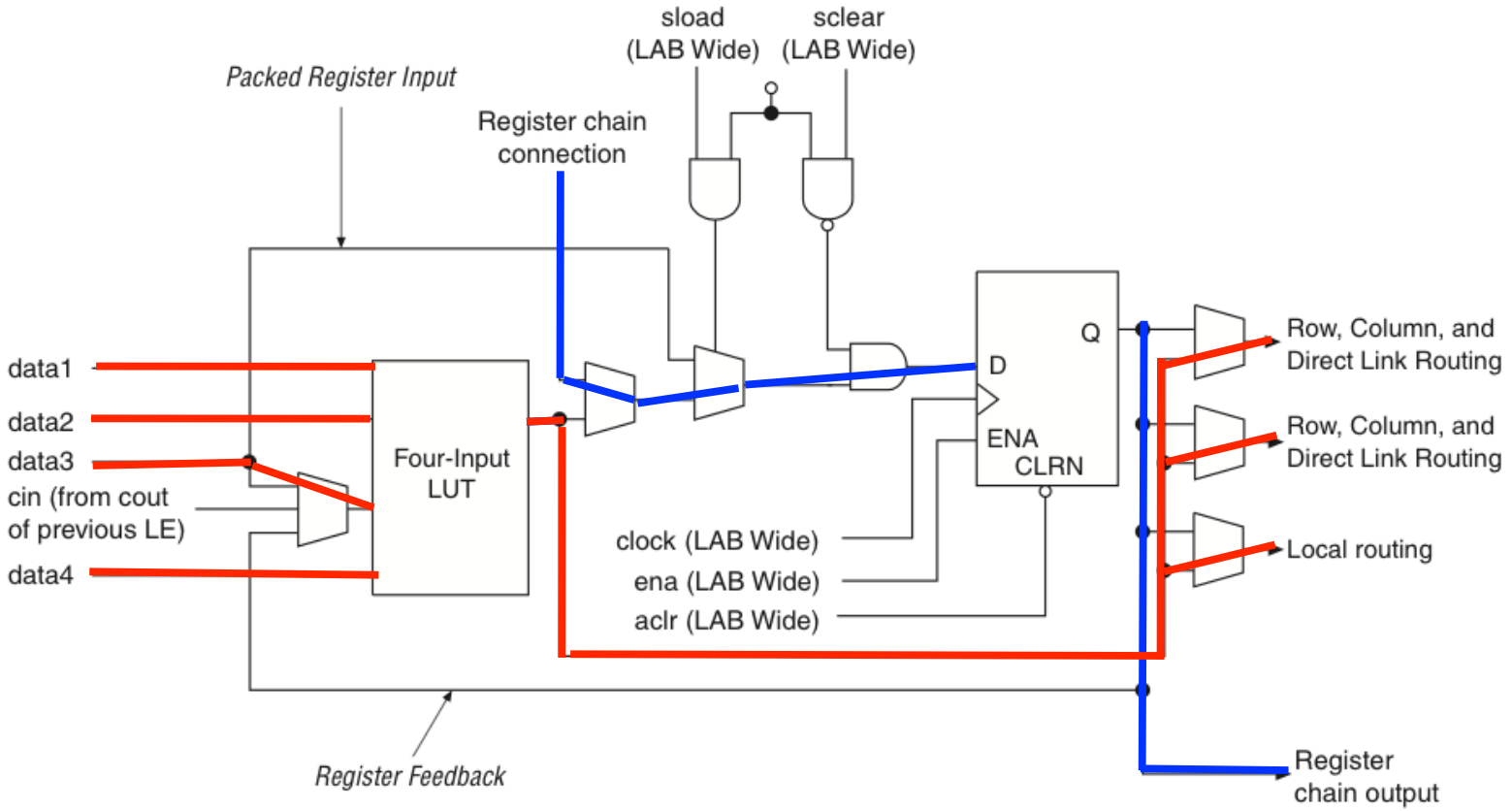
# "Packed" Mode – LUT and Register

***Figure 2–3. LE in Normal Mode***



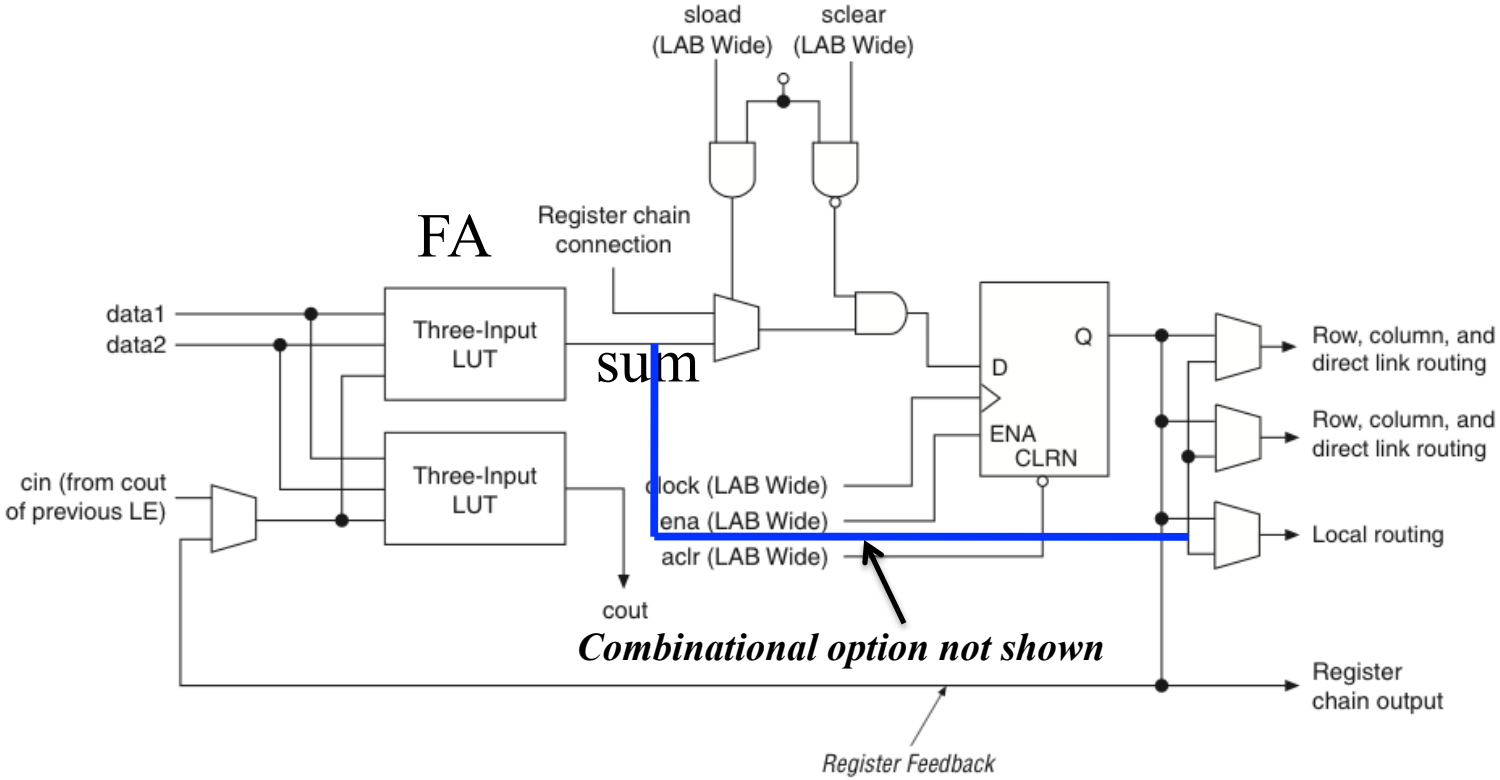
# Shift Register Mode + Logic

***Figure 2–3. LE in Normal Mode***



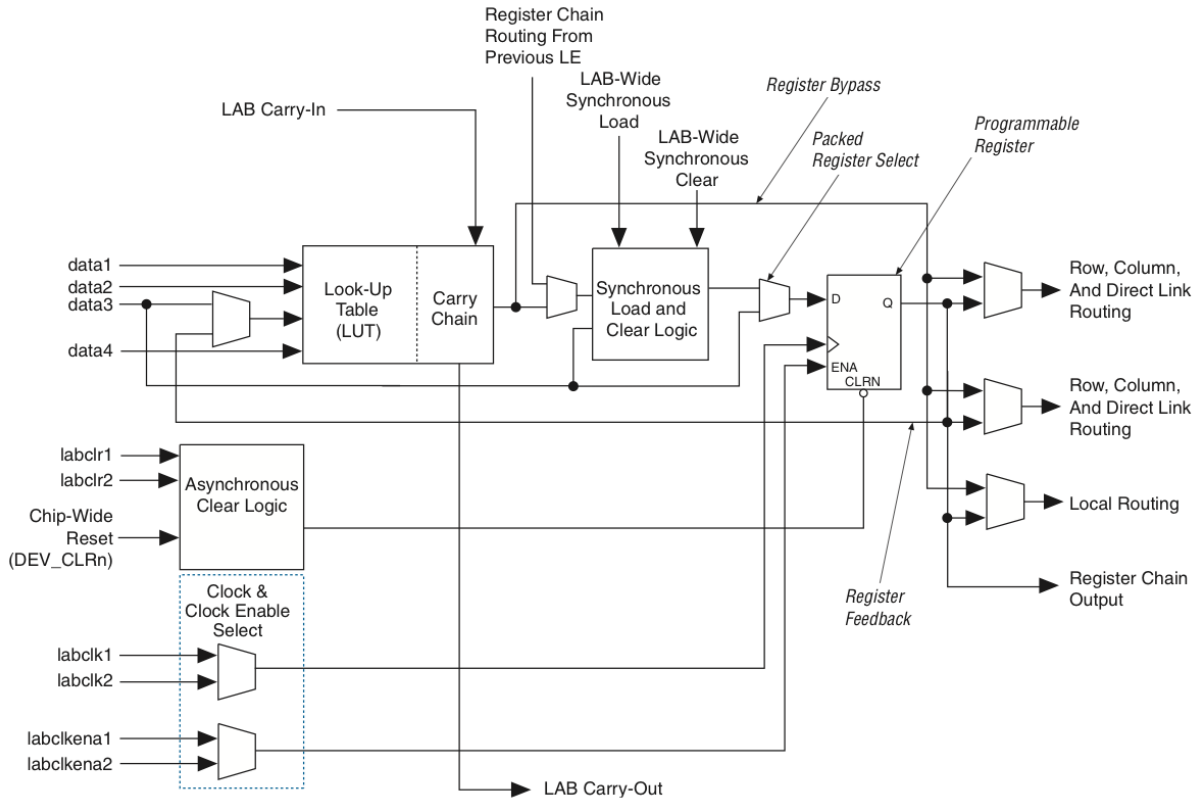
# Cyclone II LE used for Arithmetic

***Figure 2–4. LE in Arithmetic Mode***

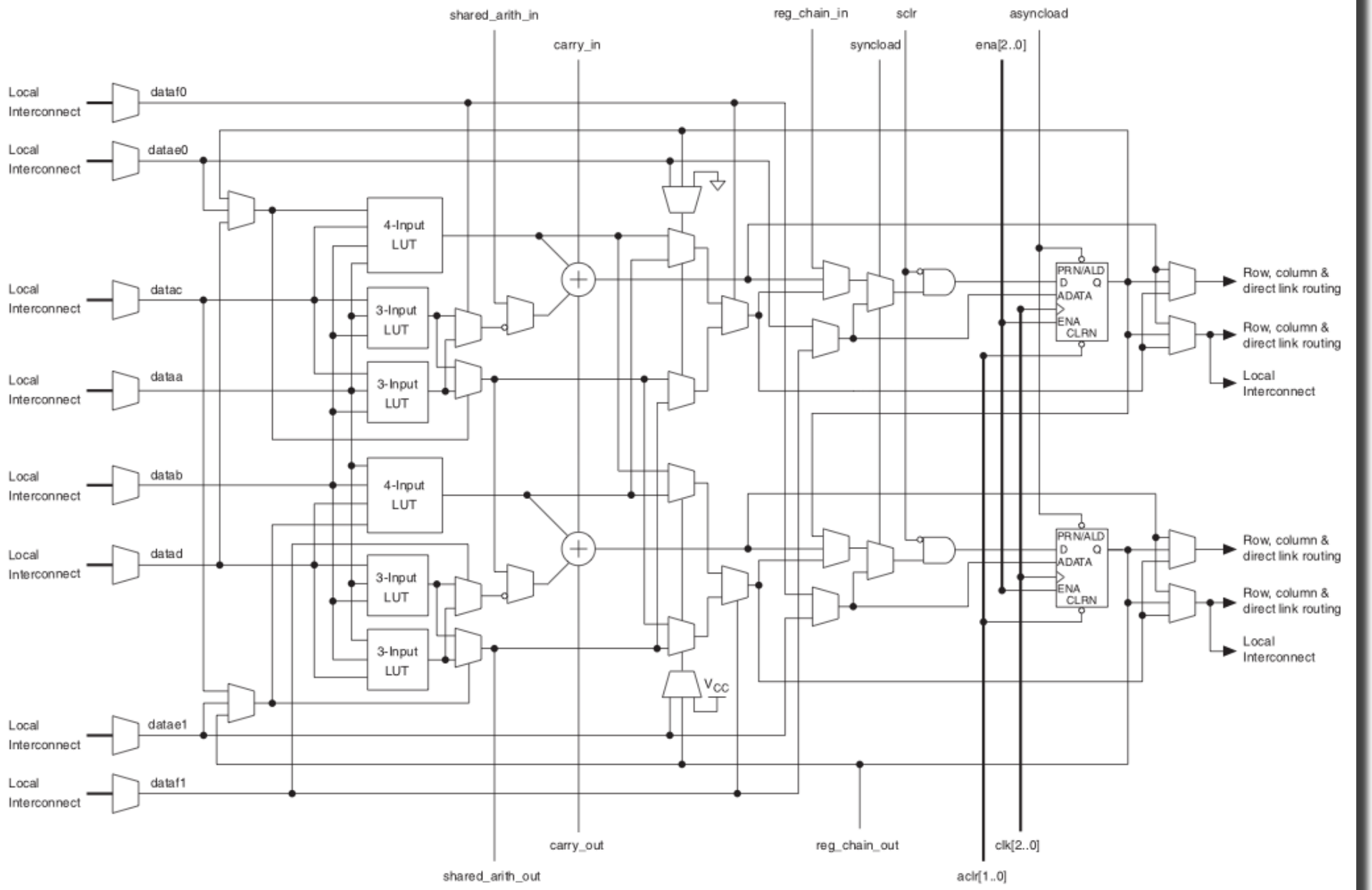


# Full Cyclone II LE

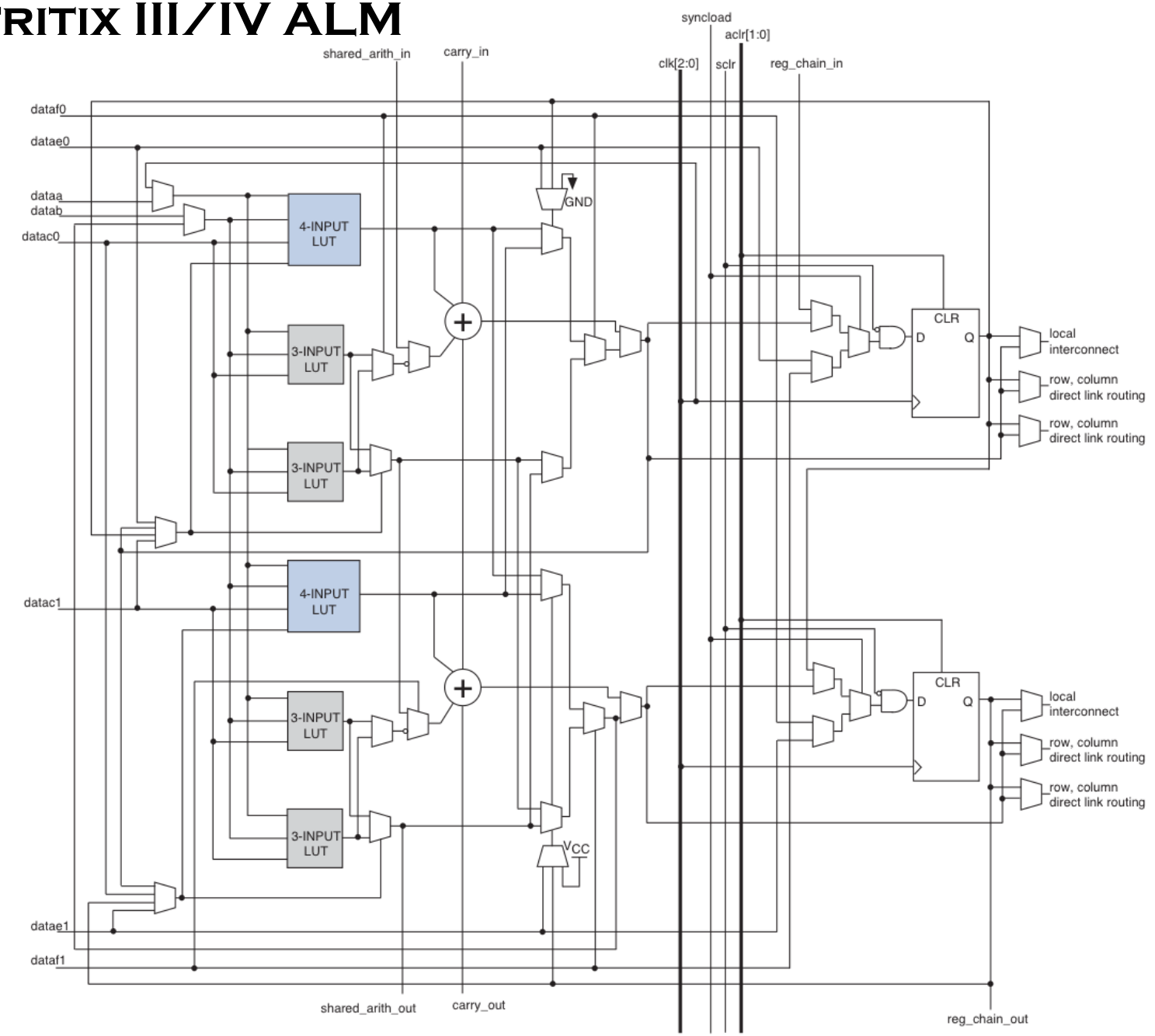
***Figure 2–2. Cyclone II LE***



# Stratix II ALM

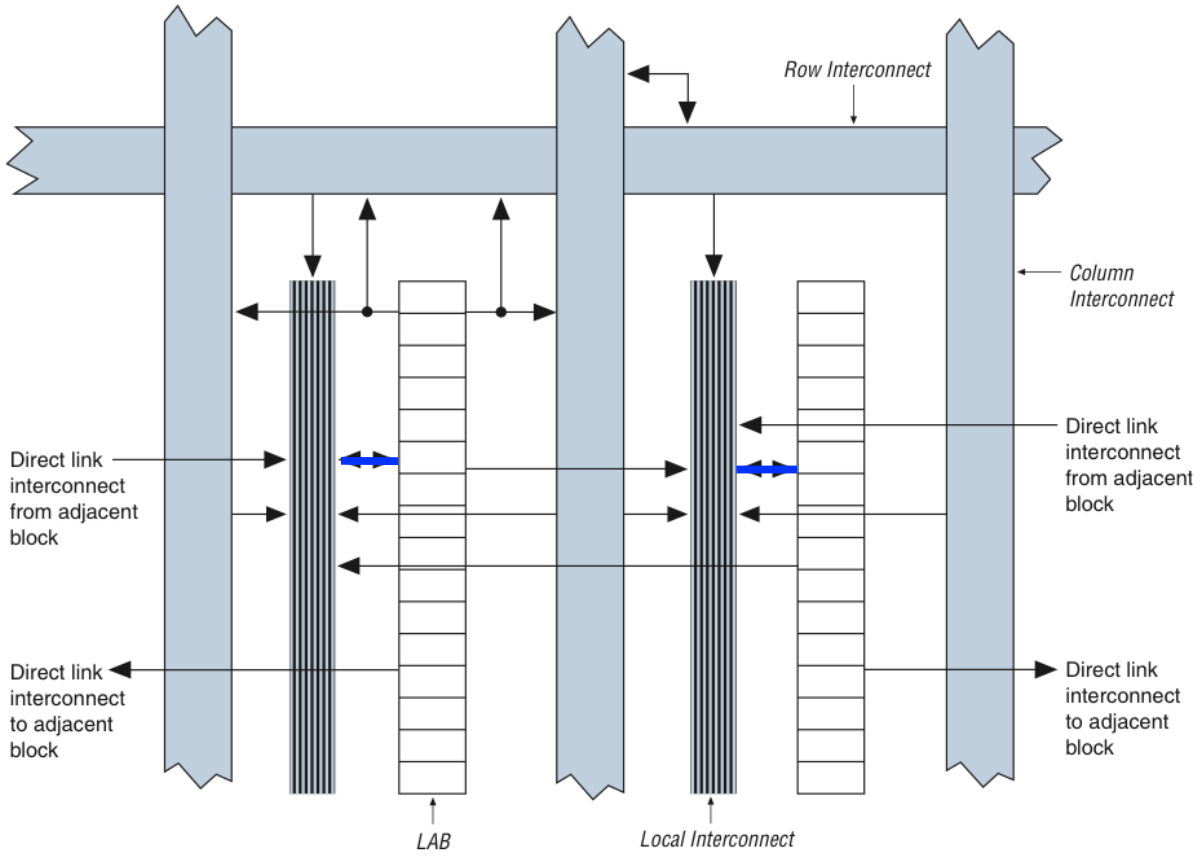


# Stritix III/IV ALM



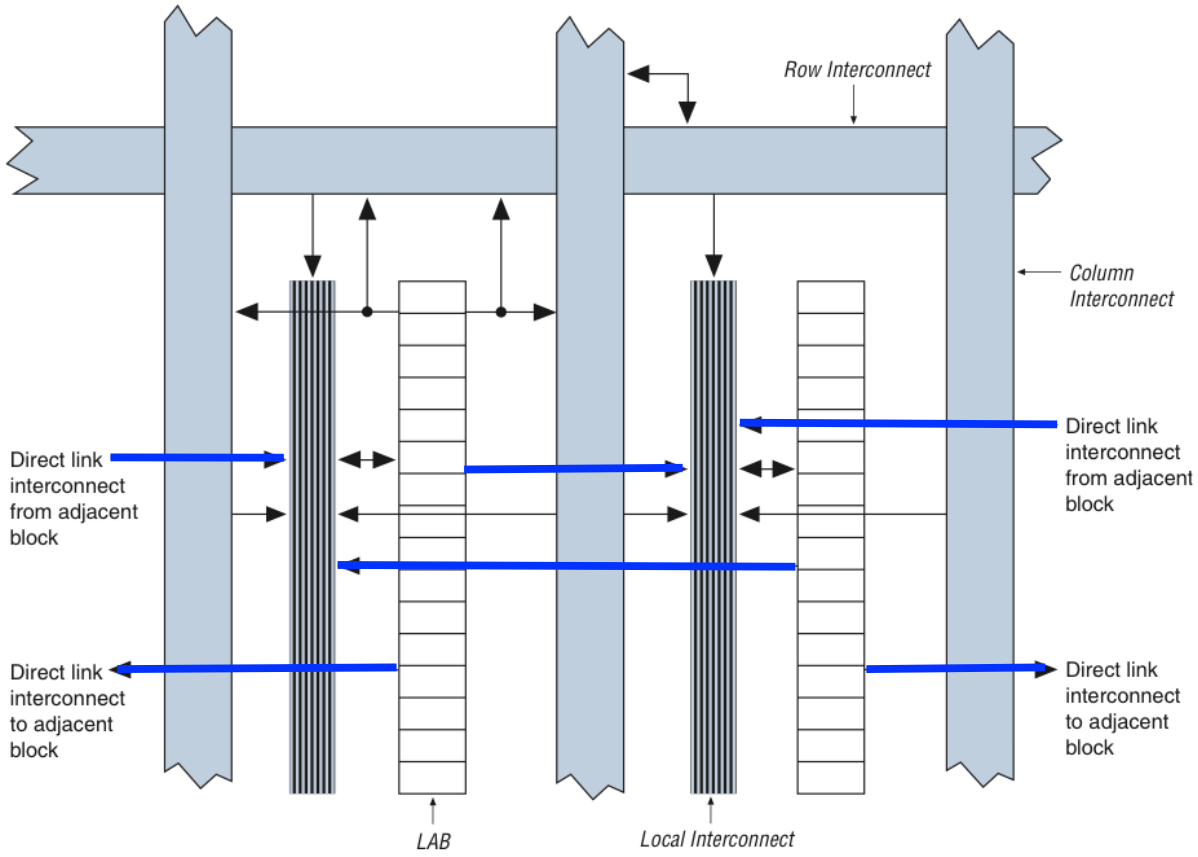
# LAB Local Interconnect – Local connections

*Figure 2–5. Cyclone II LAB Structure*



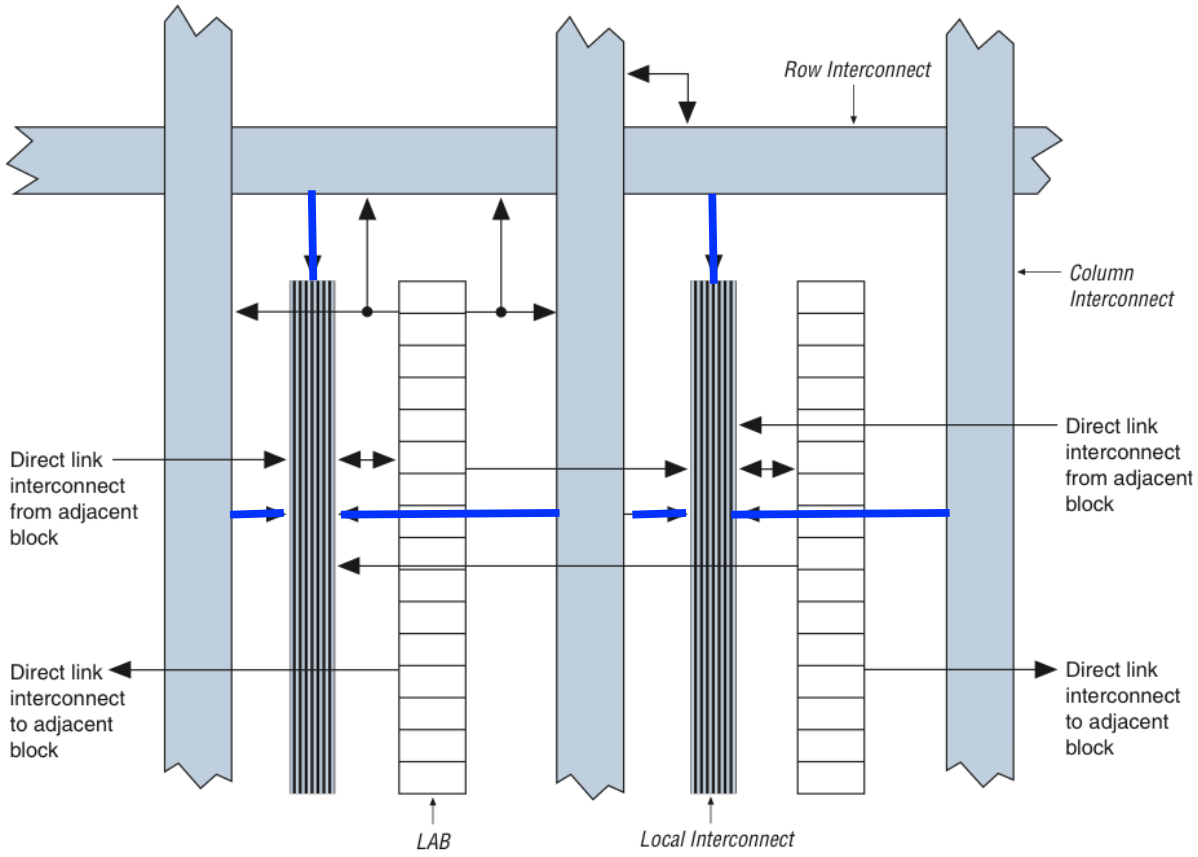
# Local Interconnect – Connect to adjacent LABs

*Figure 2–5. Cyclone II LAB Structure*



# Local Interconnect – Connect to Row/Column Interconnect

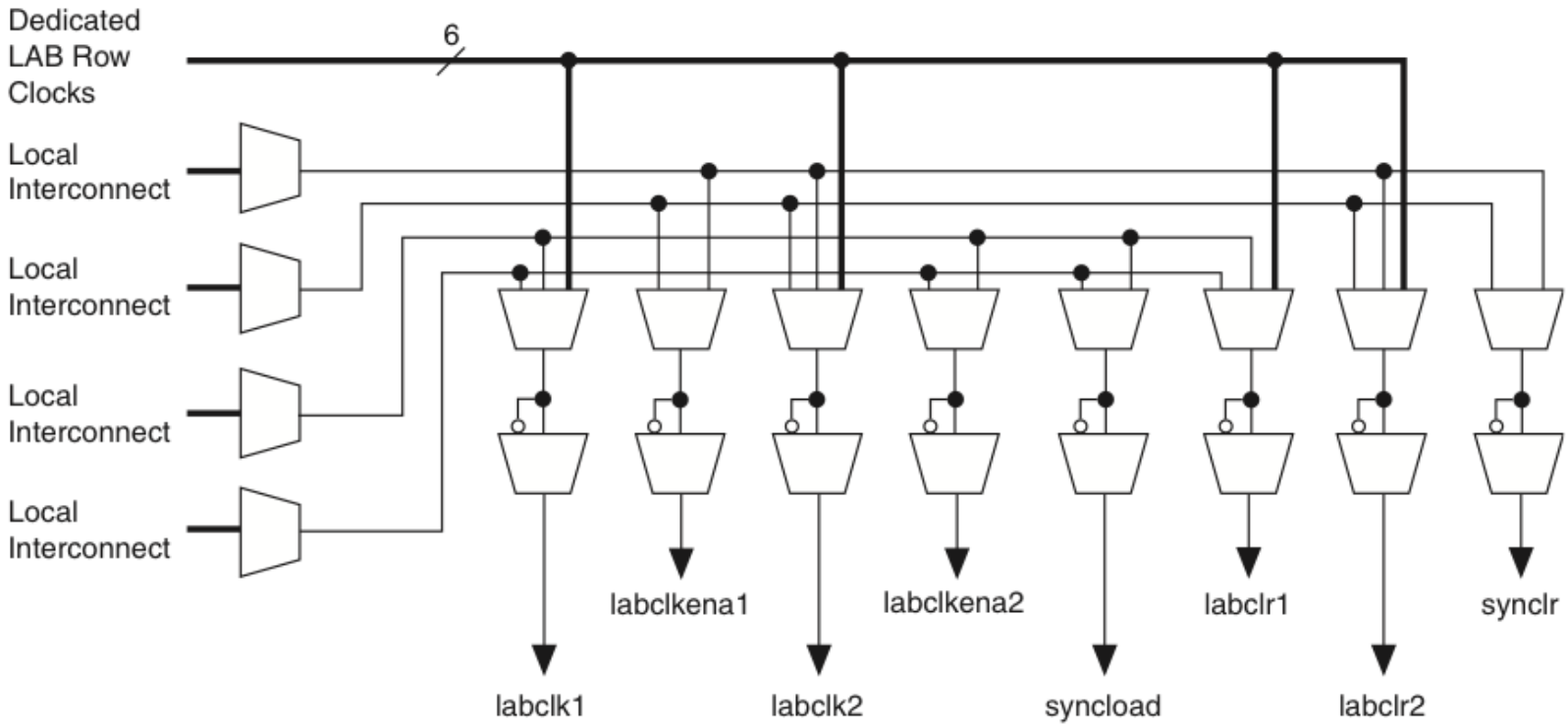
*Figure 2–5. Cyclone II LAB Structure*



# Lab Control Signals

- Shared by LEs in LAB
  - Constraint on placement

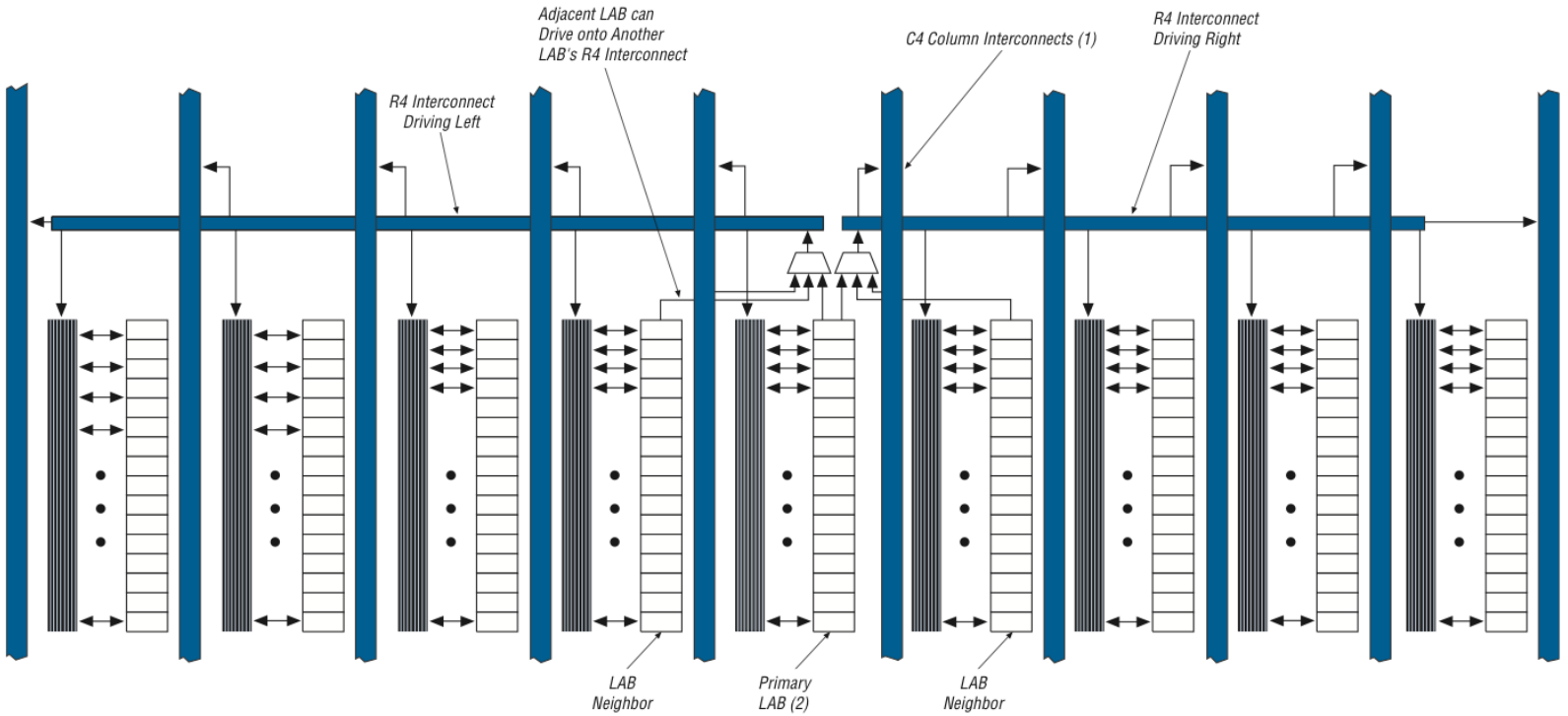
***Figure 2–7. LAB-Wide Control Signals***



# Row Interconnect

- R4 Interconnects span 4 columns
- R24 Interconnects span width of device
- LABs, memories, multipliers can drive R4s

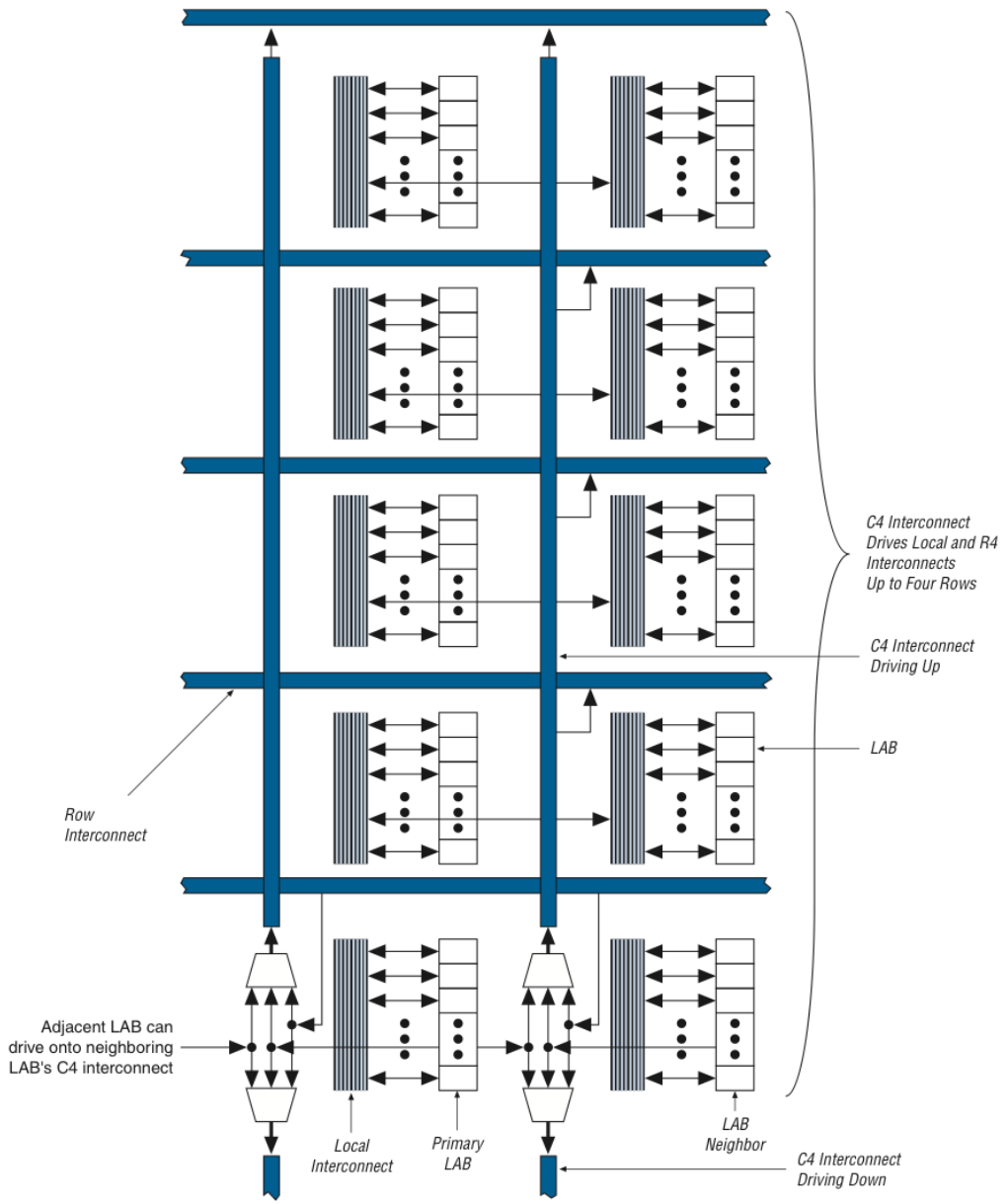
***Figure 2–8. R4 Interconnect Connections***



# Column Interconnect

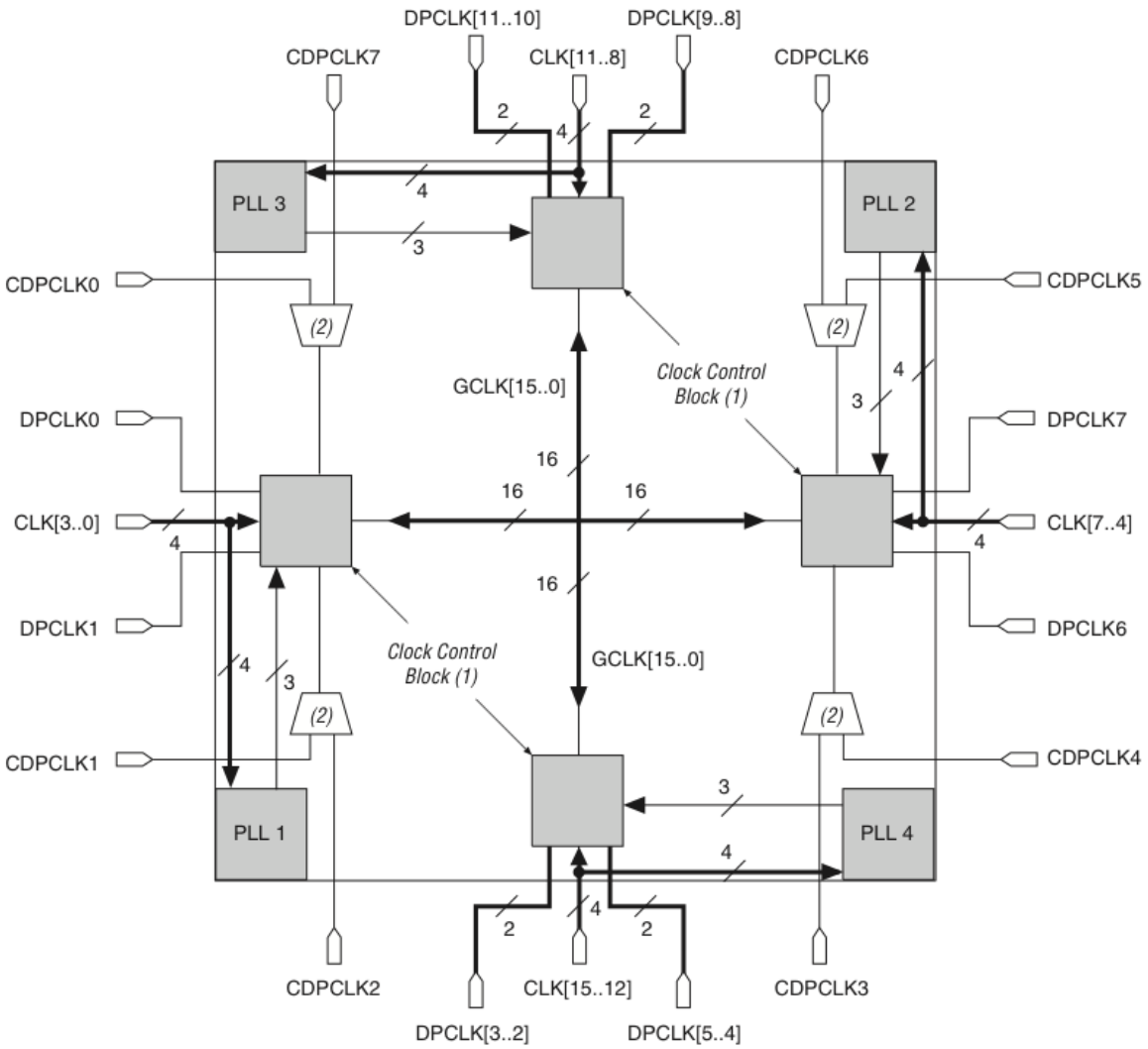
- C4 – spans 4 LAB rows
- C16 – spans 16 LAB rows
- LABs directly connected via row interconnect
  - indirectly via column interconnect

*Figure 2–10. C4 Interconnect Connections* *Note (1)*



# Clocks

*Figure 2–12. EP2C15 & Larger PLL, CLK[], DPCLK[] & Clock Control Block Locations*



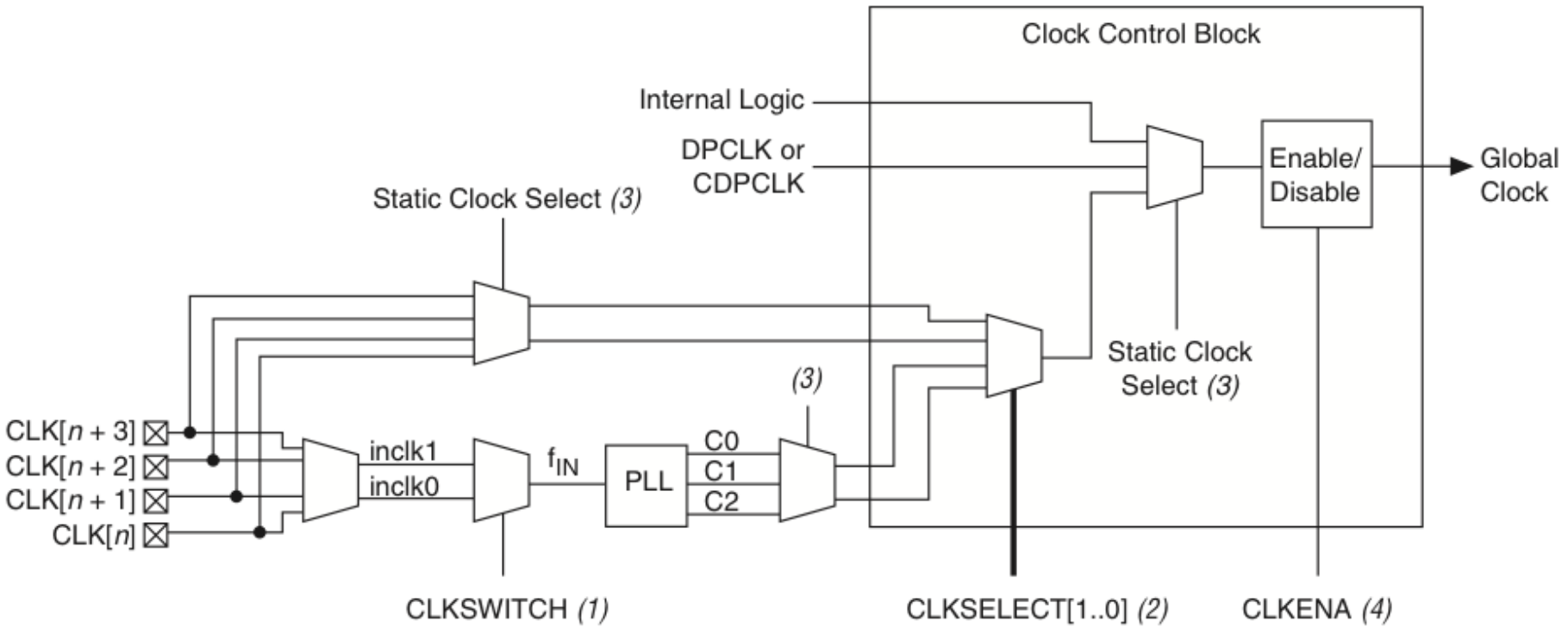
# Clocks

- Dedicated clock pins
  - low-skew
- Dual-purpose clock pins
  - have programmable delays
  - drive clock control block
- 8 – 16 global clocks
  - Used for all high-fanout control signals
    - clock, reset, enable
  - Driven by clock control block
  - Also locally by logic

# Clock Control Block

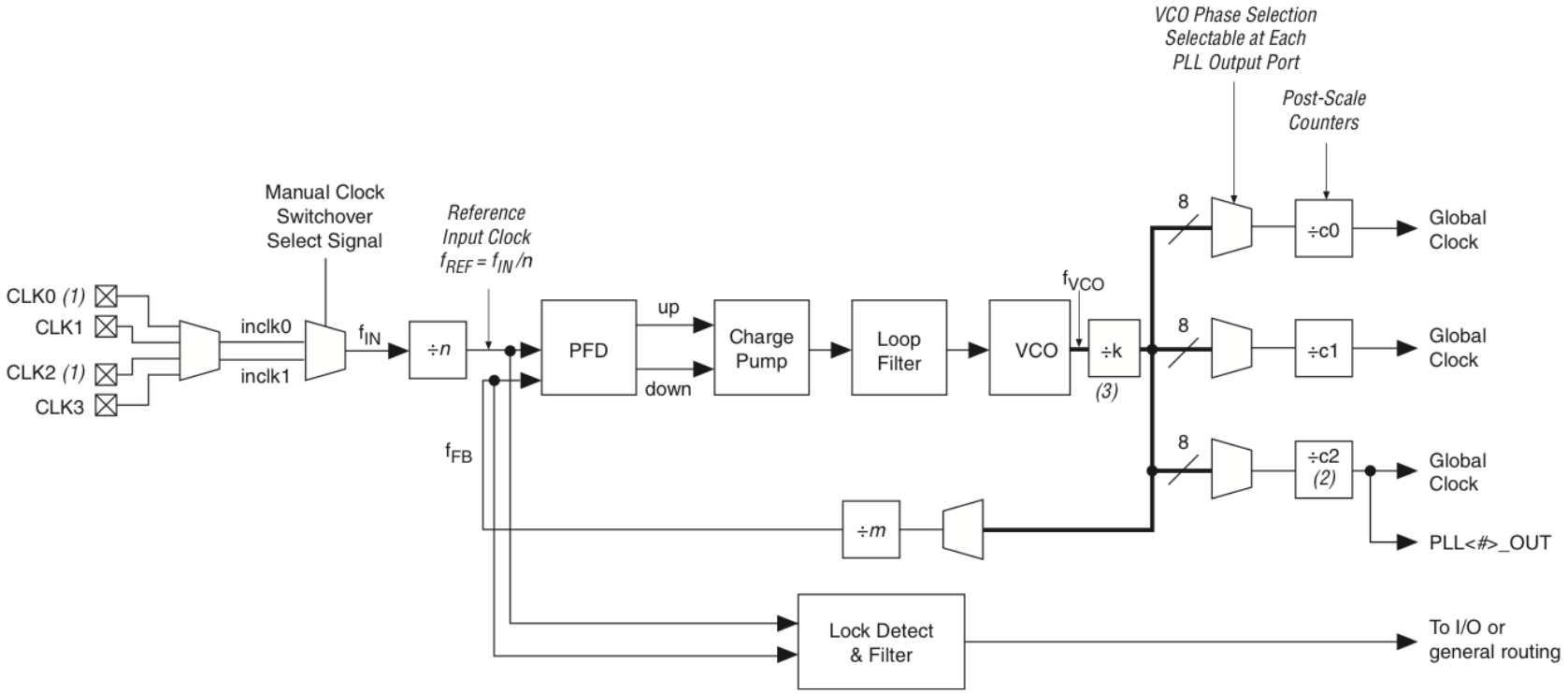
- Selects global clock input
- One per global clock

***Figure 2–13. Clock Control Block***



# PLL

***Figure 2–16. Cyclone II PLL*** *Note (1)*



# 4K Memory Block

*Table 2–6. M4K Memory Features*

| Feature | Description |
|---|---|
| Maximum performance *(1)* | 250 MHz |
| Total RAM bits per M4K block (including parity bits) | 4,608 |
| Configurations supported | 4K × 1<br>2K × 2<br>1K × 4<br>512 × 8<br>512 × 9<br>256 × 16<br>256 × 18<br>128 × 32 (not available in true dual-port mode)<br>128 × 36 (not available in true dual-port mode) |
| Parity bits | One parity bit for each byte. The parity bit, along with internal user logic, can implement parity checking for error detection to ensure data integrity. |
| Byte enable | M4K blocks support byte writes when the write port has a data width of 1, 2, 4, 8, 9, 16, 18, 32, or 36 bits. The byte enables allow the input data to be masked so the device can write to specific bytes. The unwritten bytes retain the previous written value. |
| Packed mode | Two single-port memory blocks can be packed into a single M4K block if each of the two independent block sizes are equal to or less than half of the M4K block size, and each of the single-port memory blocks is configured in single-clock mode. |
| Address clock enable | M4K blocks support address clock enable, which is used to hold the previous address value for as long as the signal is enabled. This feature is useful in handling misses in cache applications. |
| Memory initialization file (**.mif**) | When configured as RAM or ROM, you can use an initialization file to pre-load the memory contents. |
| Power-up condition | Outputs cleared |
| Register clears | Output registers only |
| Same-port read-during-write | New data available at positive clock edge |
| Mixed-port read-during-write | Old data available at positive clock edge |

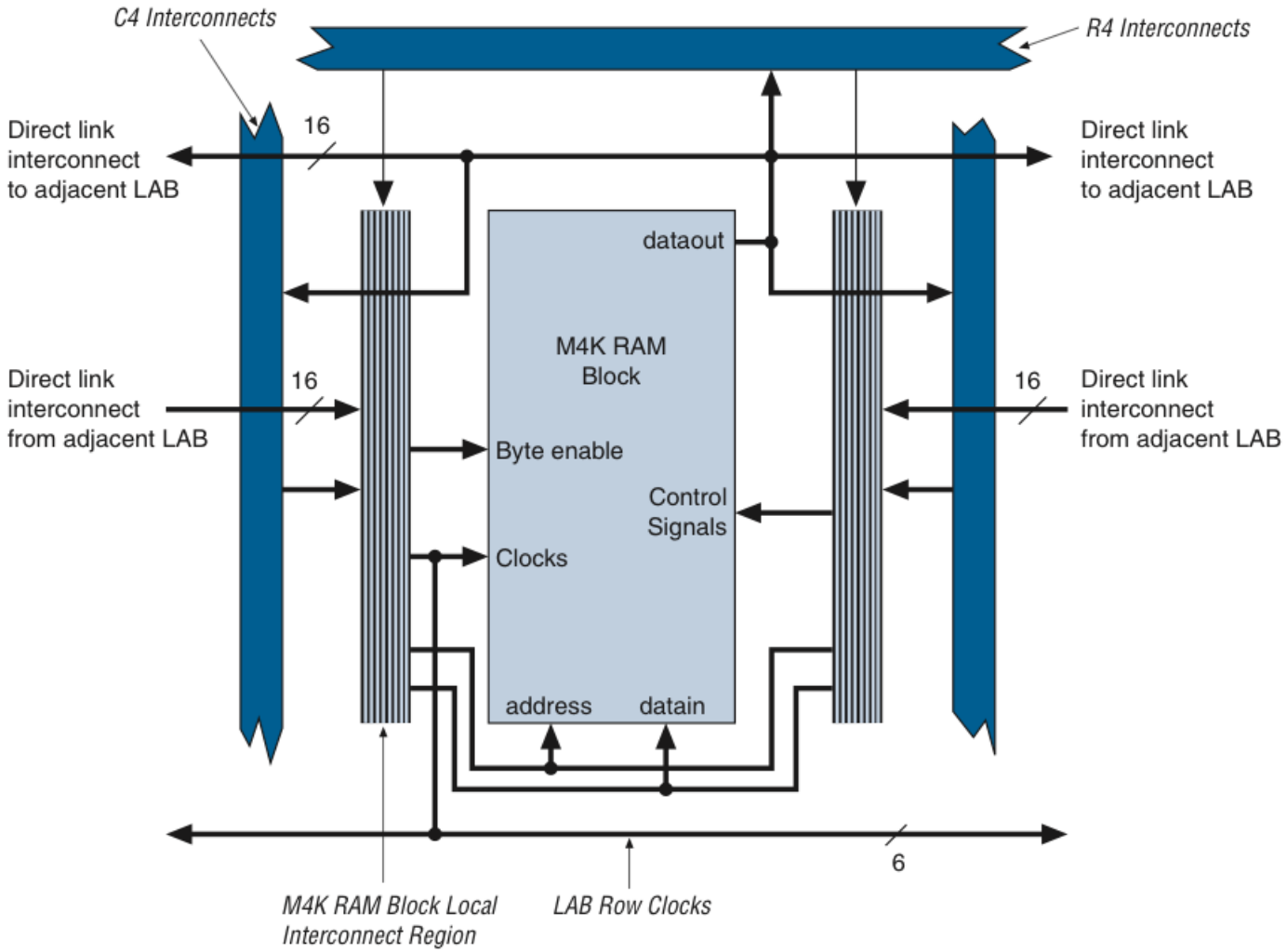
# 4K Memory

*Table 2–7. M4K Memory Modes*

| Memory Mode | Description |
|---|---|
| Single-port memory | M4K blocks support single-port mode, used when simultaneous reads and writes are not required. Single-port memory supports non-simultaneous reads and writes. |
| Simple dual-port memory | Simple dual-port memory supports a simultaneous read and write. |
| Simple dual-port with mixed width | Simple dual-port memory mode with different read and write port widths. |
| True dual-port memory | True dual-port mode supports any combination of two-port operations: two reads, two writes, or one read and one write at two different clock frequencies. |
| True dual-port with mixed width | True dual-port mode with different read and write port widths. |
| Embedded shift register | M4K memory blocks are used to implement shift registers. Data is written into each address location at the falling edge of the clock and read from the address at the rising edge of the clock. |
| ROM | The M4K memory blocks support ROM mode. A MIF initializes the ROM contents of these blocks. |
| FIFO buffers | A single clock or dual clock FIFO may be implemented in the M4K blocks. Simultaneous read and write from an empty FIFO buffer is not supported. |

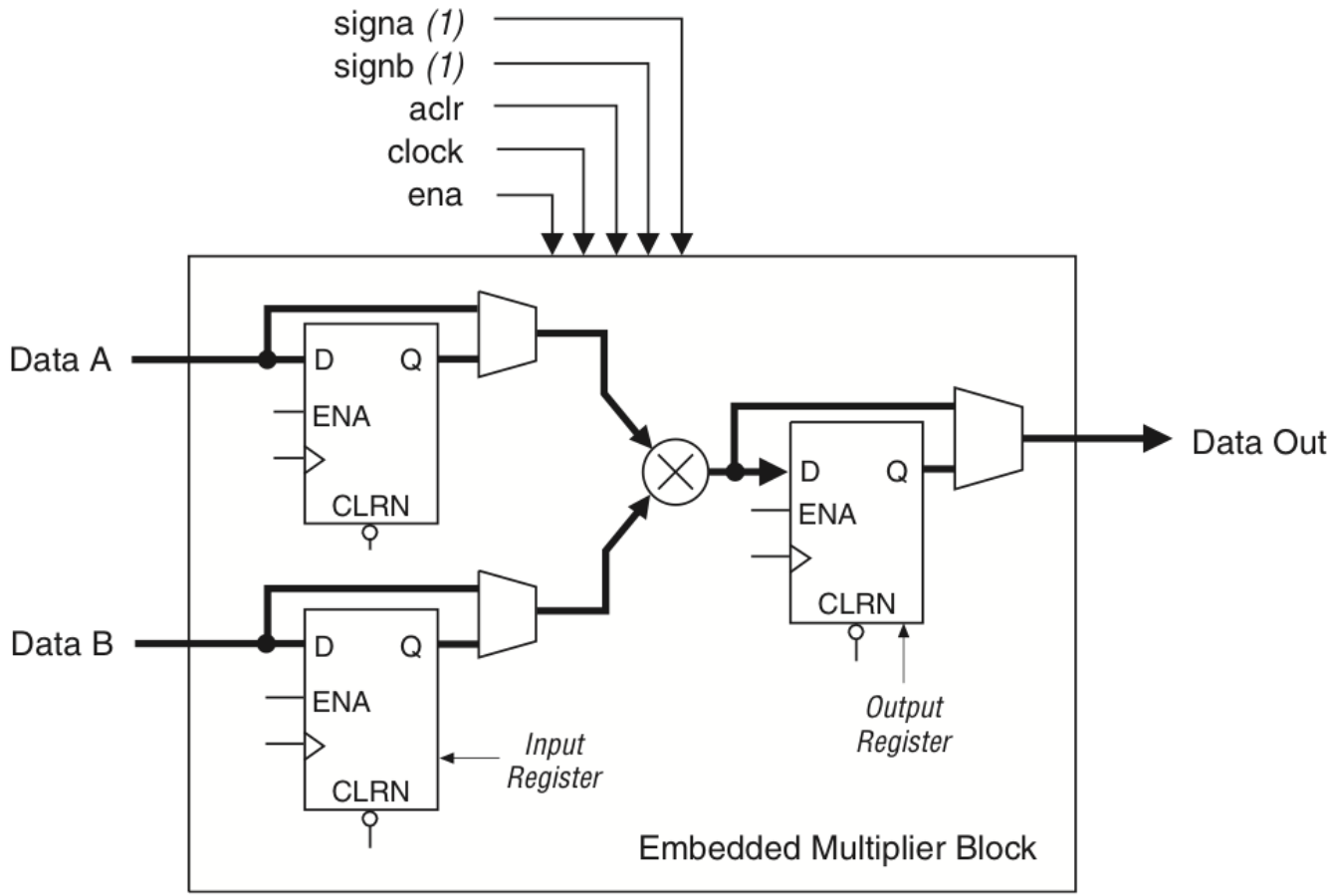
# Memory Connections

***Figure 2–17. M4K RAM Block LAB Row Interface***



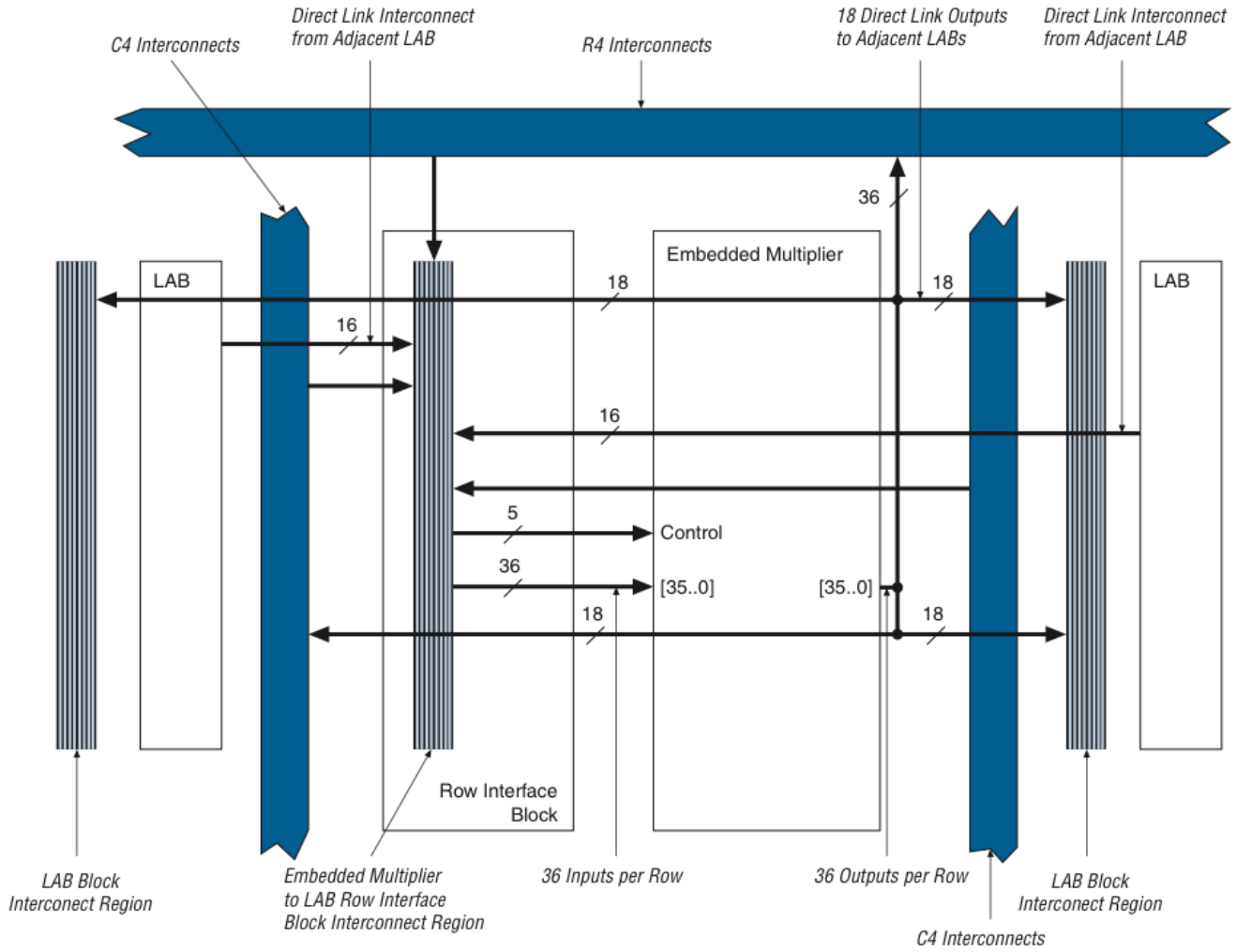
# Multiplier

***Figure 2–18. Multiplier Block Architecture***



# Multiplier Connections

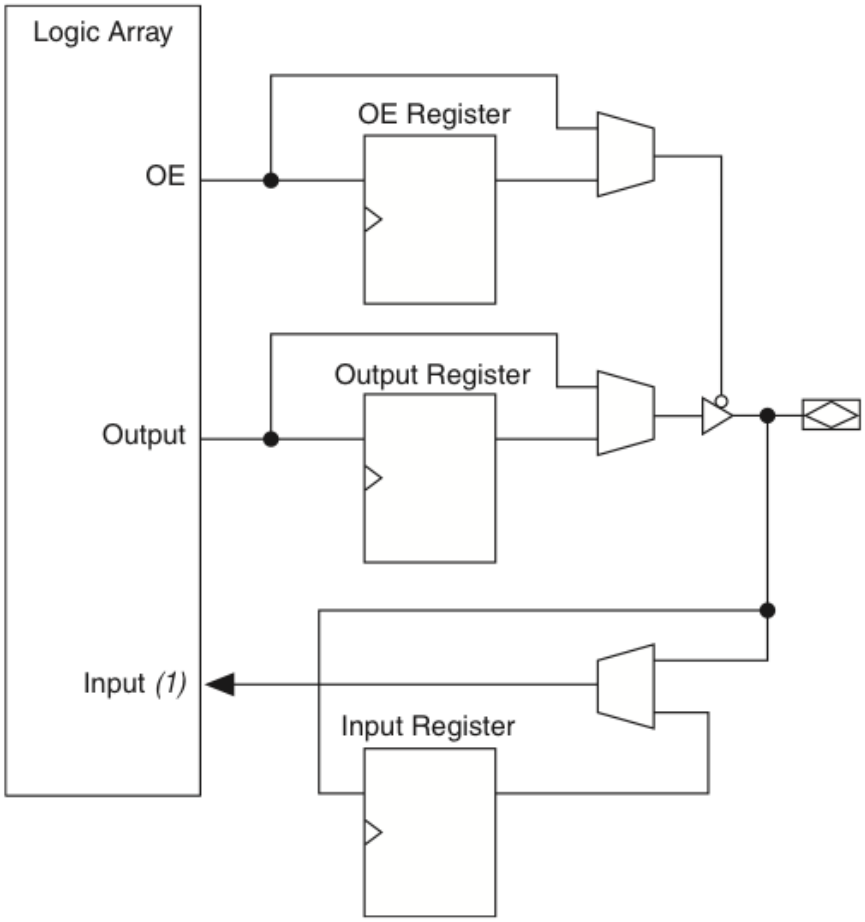
***Figure 2–19. Embedded Multiplier LAB Row Interface***



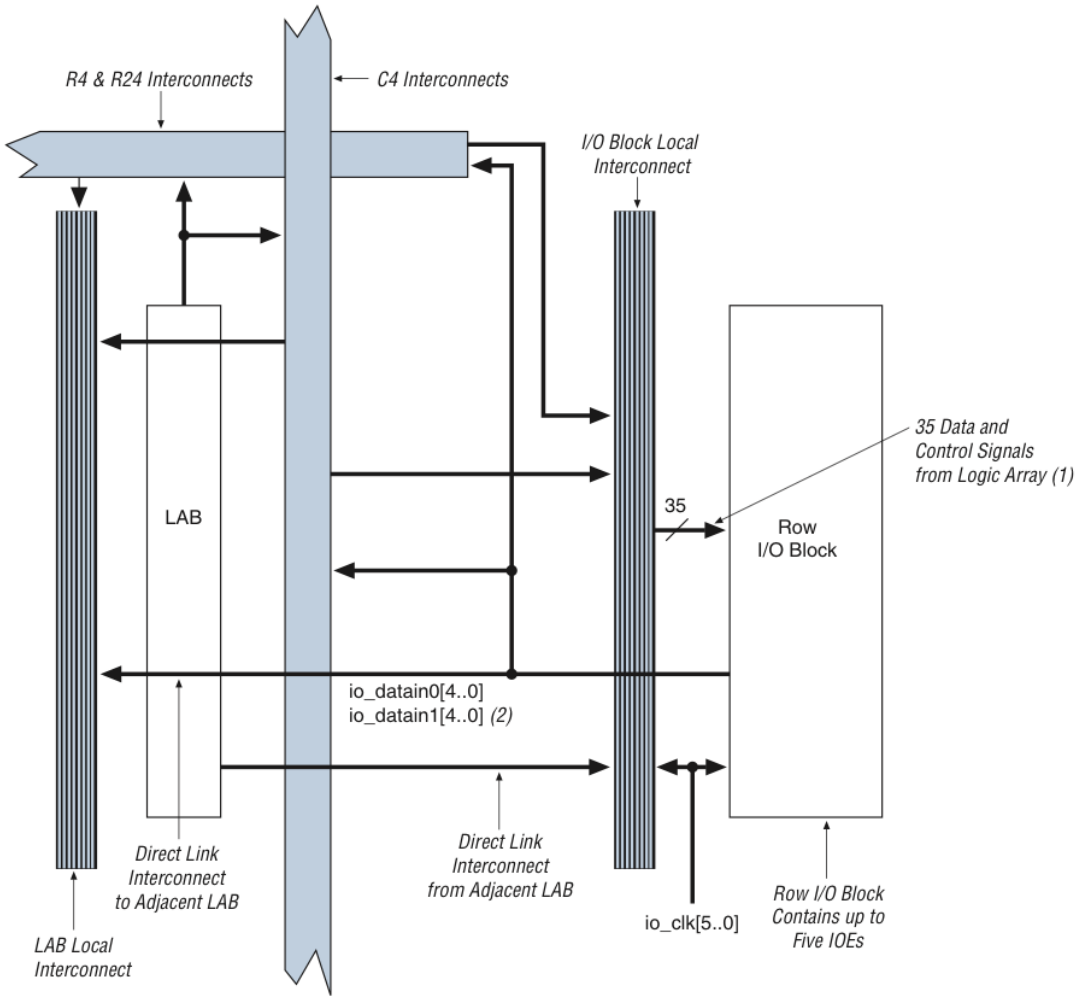
# I/O Block

- Output pin
  - Combinational
  - Registered
- Input pin
  - Combinational
  - Registered
- In/Out (tri-state) pin
  - Mix of both

*Figure 2–20. Cyclone II IOE Structure*

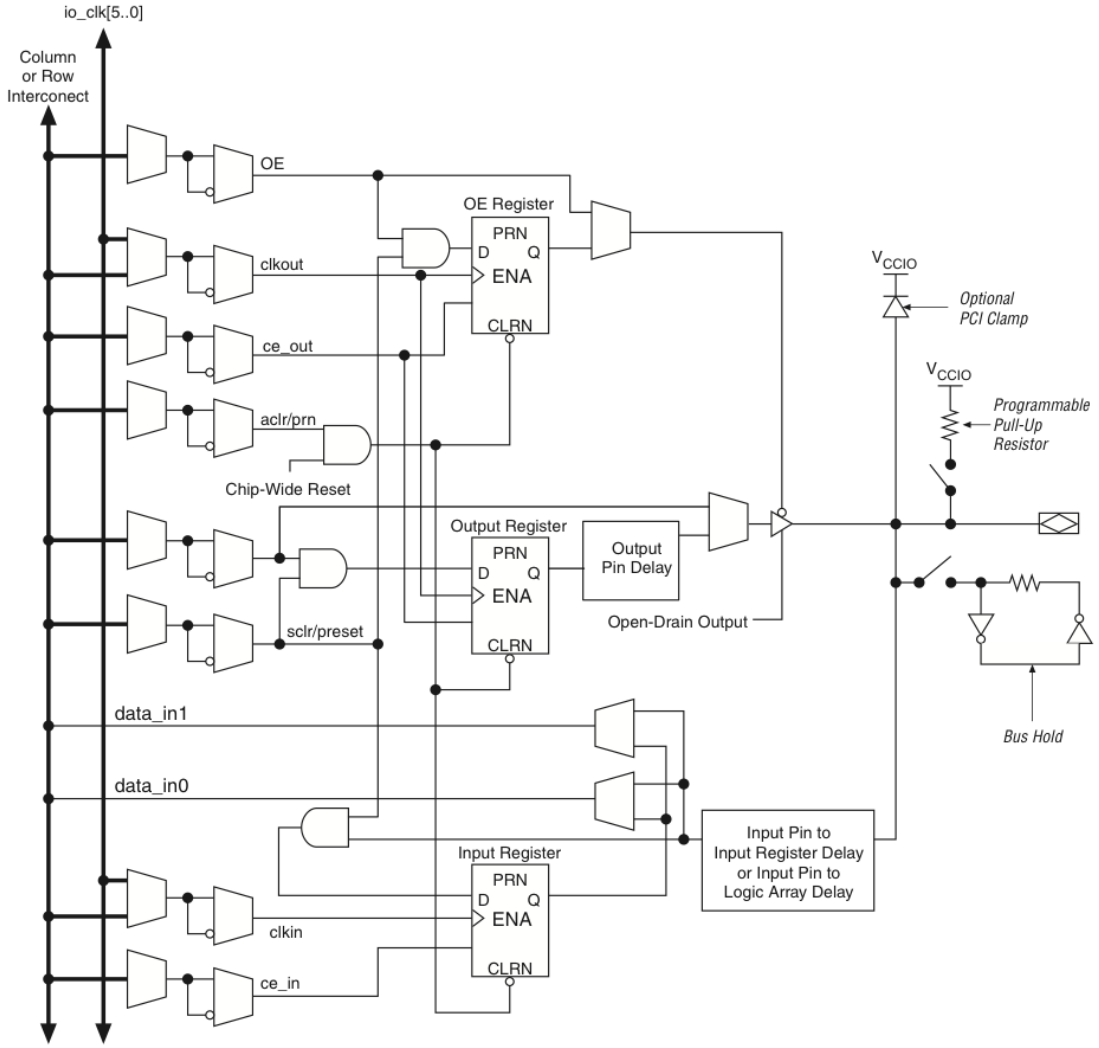


# Connections to I/O Block (5 pins)



# More I/O Detail

*Figure 2–25. Cyclone II IOE in Bidirectional I/O Configuration*



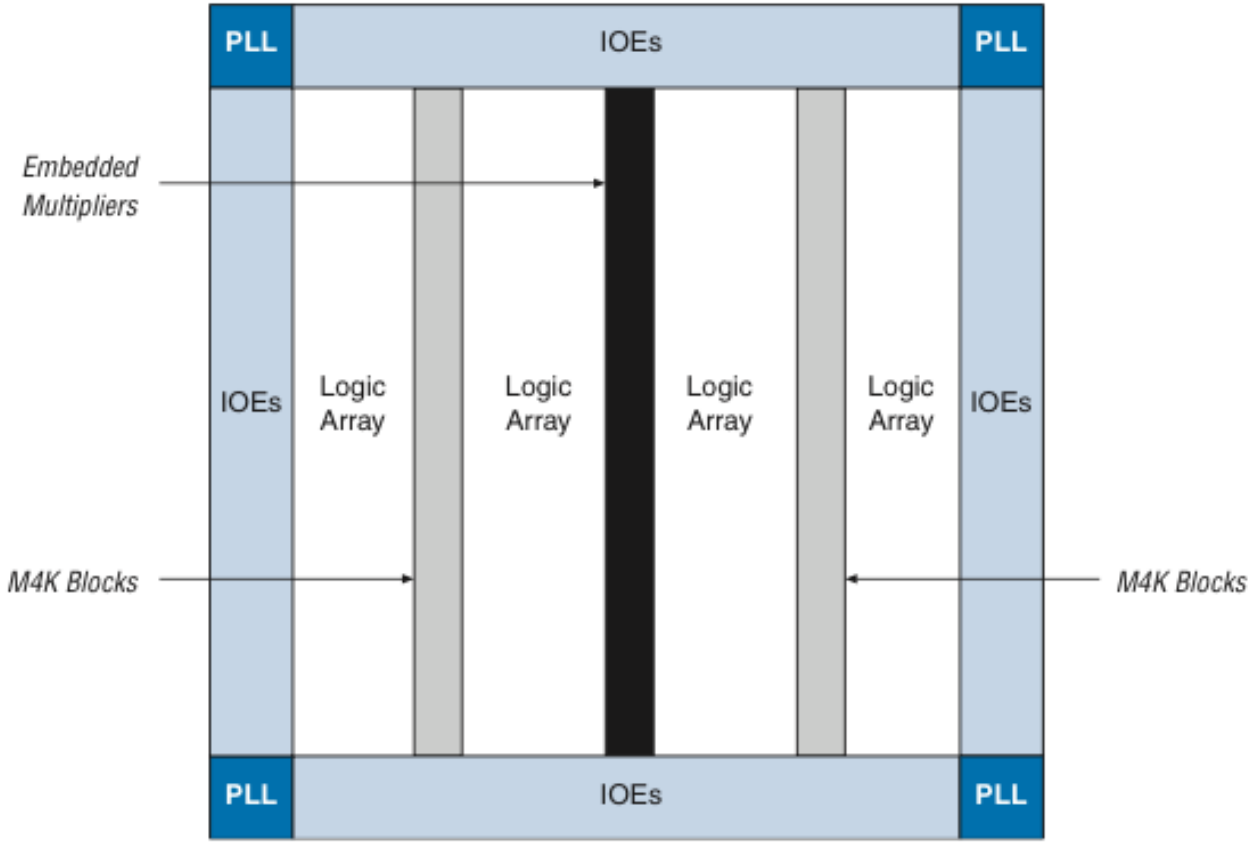
# Configurable I/Os

- Single-ended and differential
- Voltage and current mode
  - External reference voltage
- Range of voltages: 1.5v, 1.8v, 2.0v, 2.5v, 3.3v
- Range of standards: LVTTL, LVCMOS, SSTL, HSTL, LVDS, LVPECL
- Programmable drive strength: 4 – 20 mA
  - Interface requirement
  - Slew rate control

- I/Os arranged in Banks
  - 4 – 8 per chip
  - Banks each have a different power bus

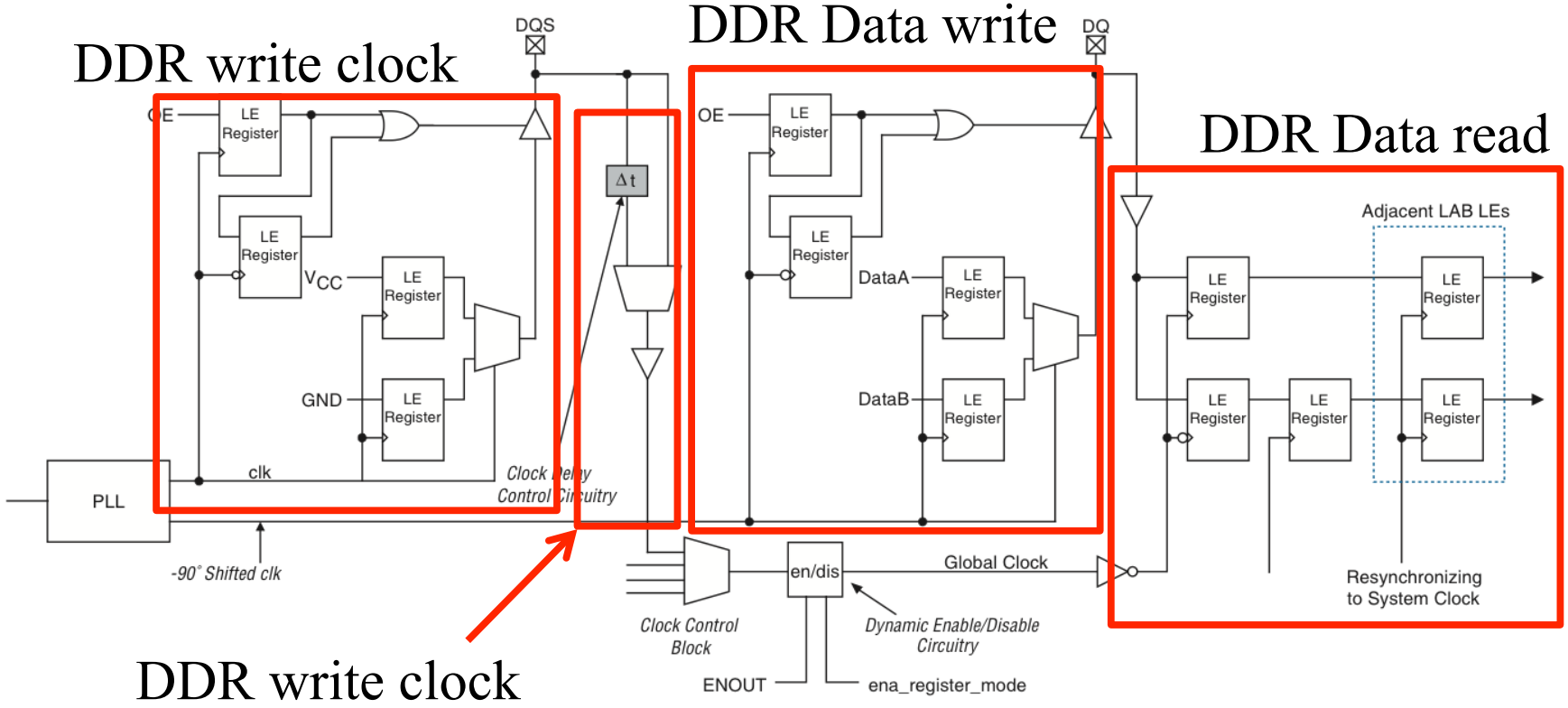
# Cyclone Floorplan

*Figure 2–1. Cyclone II EP2C20 Device Block Diagram*



# Support for Memory Interfaces

*Figure 2–27. DDR SDRAM Interfacing*



# Details of One Xilinx Virtex Slice

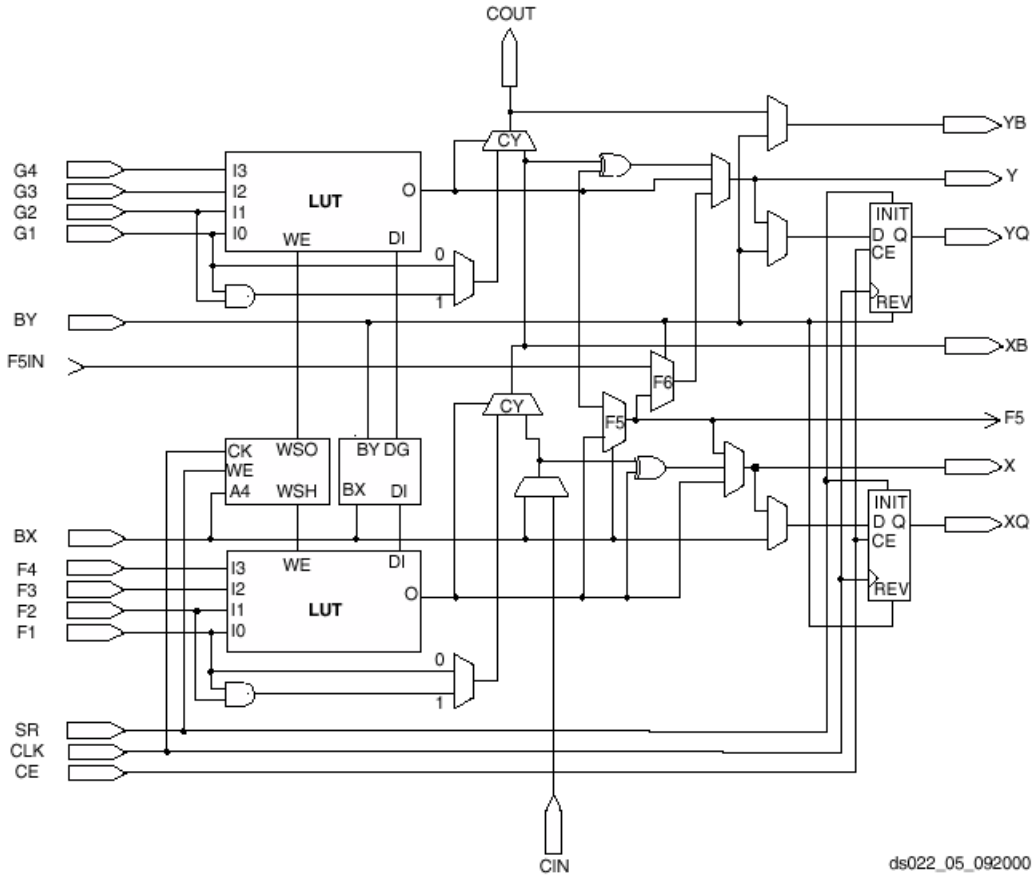



*Figure 5:* **Detailed View of Virtex-E Slice**

# Implements any Two 4-input Functions

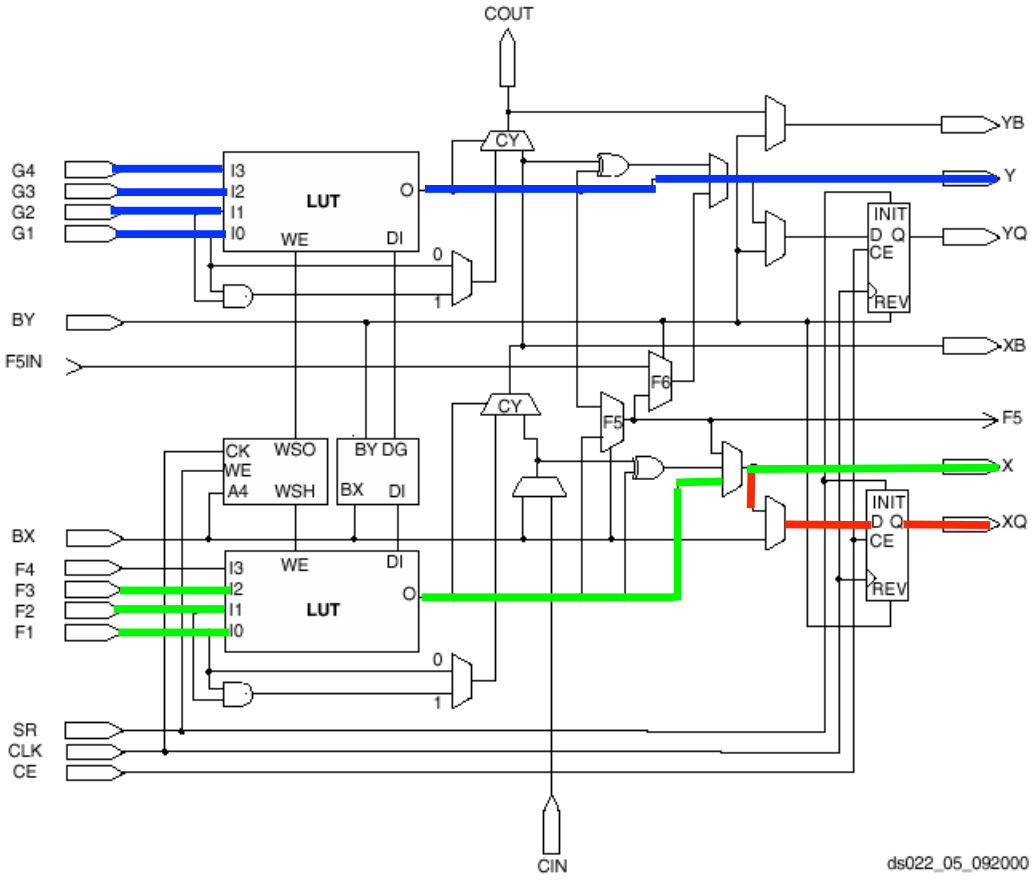


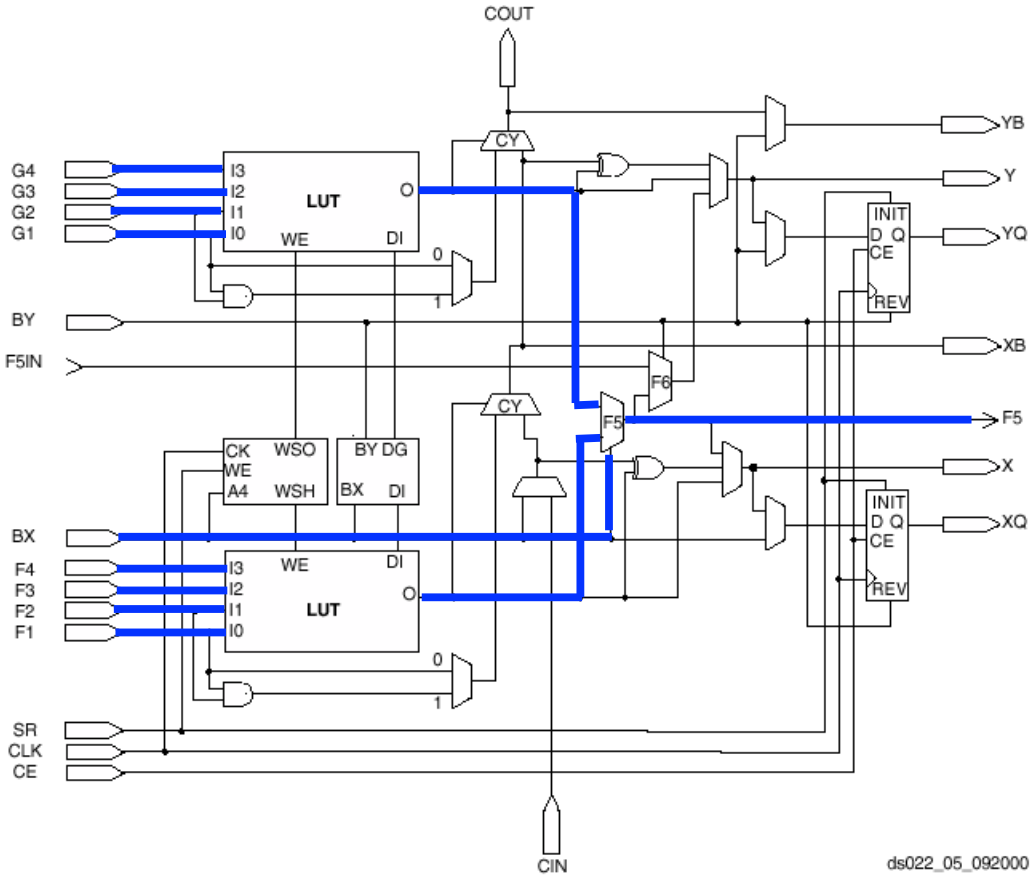

*Figure 5:* **Detailed View of Virtex-E Slice**

4-input function

3-input function;

registered

# Implements any 5-input Function

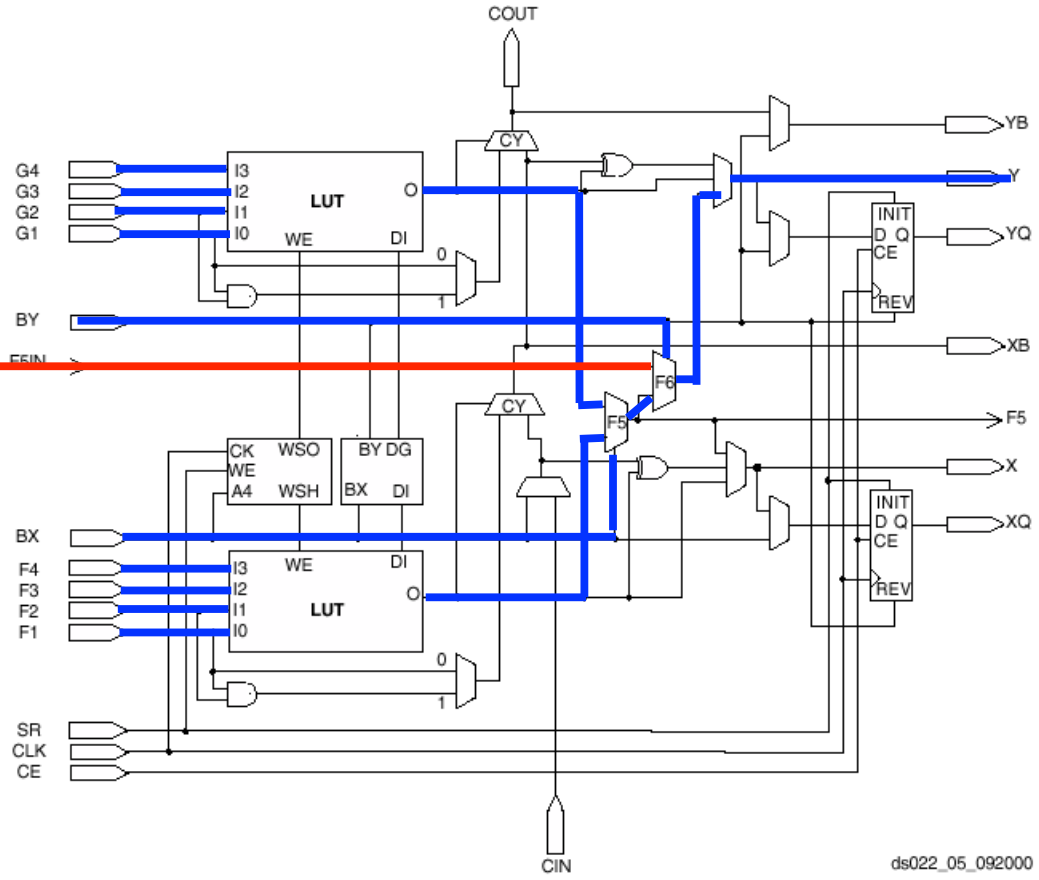



5-input function

*Figure 5:* **Detailed View of Virtex-E Slice**

# Two Slices: Any 6-input Function

from other slice




6-input function

*Figure 5:* **Detailed View of Virtex-E Slice**

# Fast Carry Chain: Add two bits per slice

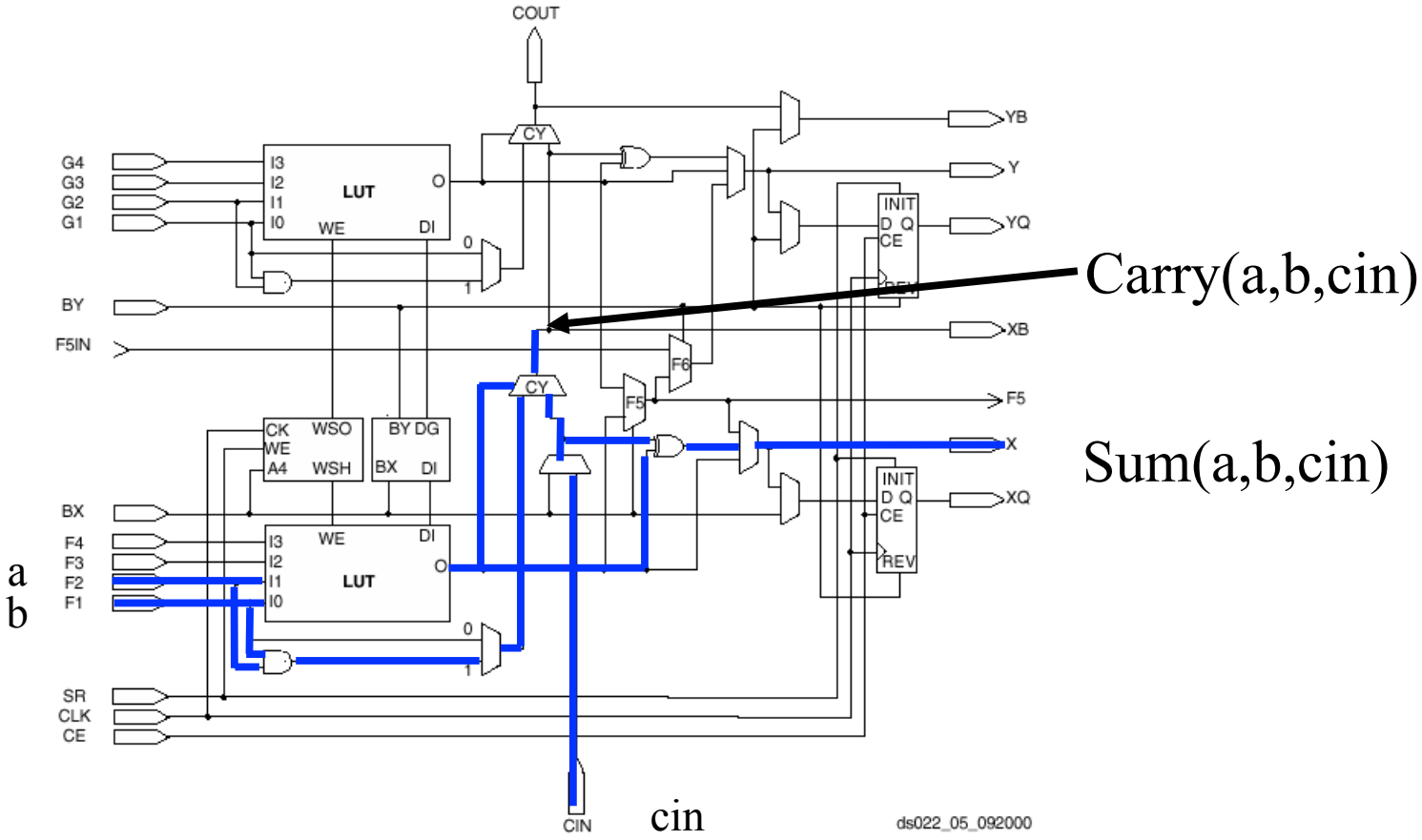



*Figure 5:* **Detailed View of Virtex-E Slice**

# Lookup Tables used as memory (16 x 2) [ Distributed Memory ]

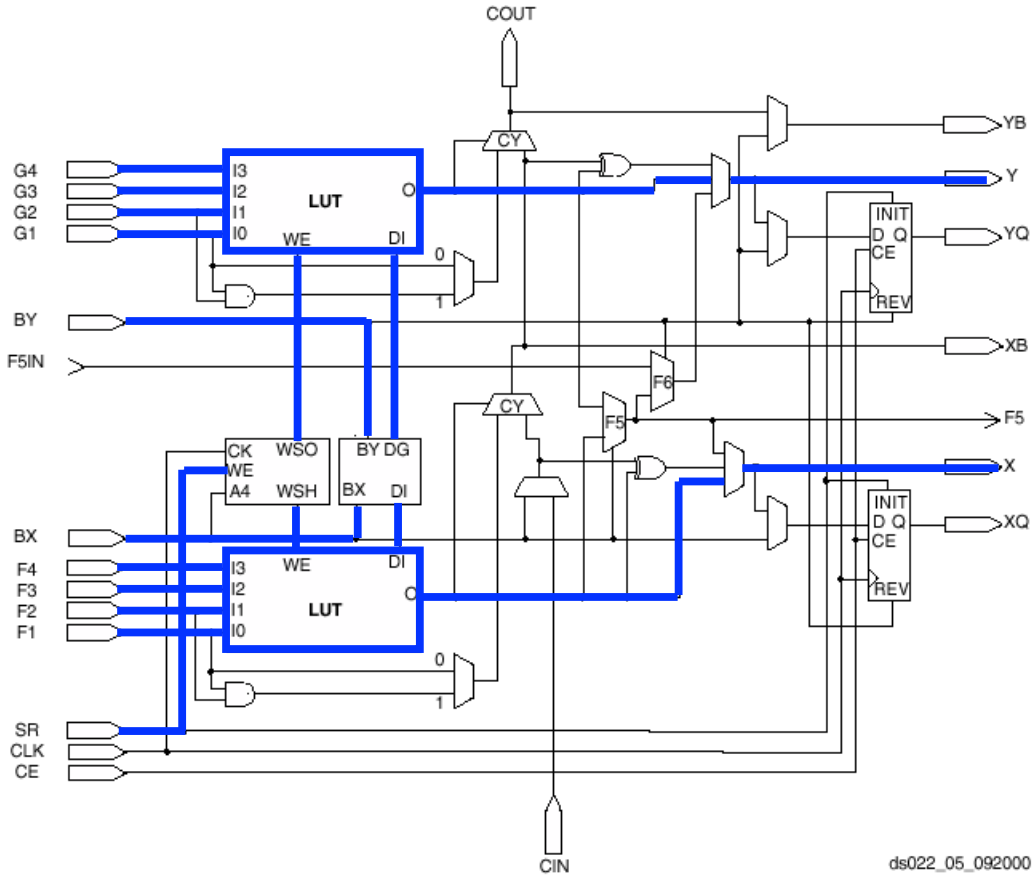



*Figure 5:* **Detailed View of Virtex-E Slice**

# Computer-aided Design

- Can't design FPGAs by hand
  - way too much logic to manage, hard to make changes
- Hardware description languages
  - specify functionality of logic at a high level
- Validation - high-level simulation to catch specification errors
  - verify pin-outs and connections to other system components
  - low-level to verify mapping and check performance
- Logic synthesis
  - process of compiling HDL program into logic gates and flip-flops
- Technology mapping
  - map the logic onto elements available in the implementation technology (LUTs for Xilinx FPGAs)

# CAD Tool Path (cont'd)

- Placement and routing
  - assign logic blocks to functions
  - make wiring connections
- Timing analysis - verify paths
  - determine delays as routed
  - look at critical paths and ways to improve
- Partitioning and constraining
  - if design does not fit or is unroutable as placed split into multiple chips
  - if design it too slow prioritize critical paths, fix placement of cells, etc.
  - few tools to help with these tasks exist today
- Generate programming files - bits to be loaded into chip for configuration

# CAD Tools

- Verilog (or VHDL) use to specify logic at a high-level
  - combine with schematics, library components
- Synthesis: Synopsys/Synplicity/Quartus/ISE/Precision
  - compiles Verilog to logic
  - maps logic to the FPGA cells
  - optimizes logic
- APR - automatic place and route (simulated annealing)
  - provides controllability through constraints
  - handles global signals
- STA - measure delay properties of mapping and aid in iteration

# Applications of FPGAs

- Implementation of random logic
  - easier changes at system-level (one device is modified)
  - can eliminate need for full-custom chips
- Prototyping
  - ensemble of gate arrays used to emulate a circuit to be manufactured
  - get more/better/faster debugging done than possible with simulation
- Reconfigurable hardware
  - one hardware block used to implement more than one function
  - functions must be  mutually-exclusive in time
  - can greatly reduce cost while enhancing flexibility
  - RAM-based only option
- Special-purpose computation engines
  - hardware dedicated to solving one problem (or class of problems)
  - accelerators attached to general-purpose computers