

# Field Programmable Gate Array

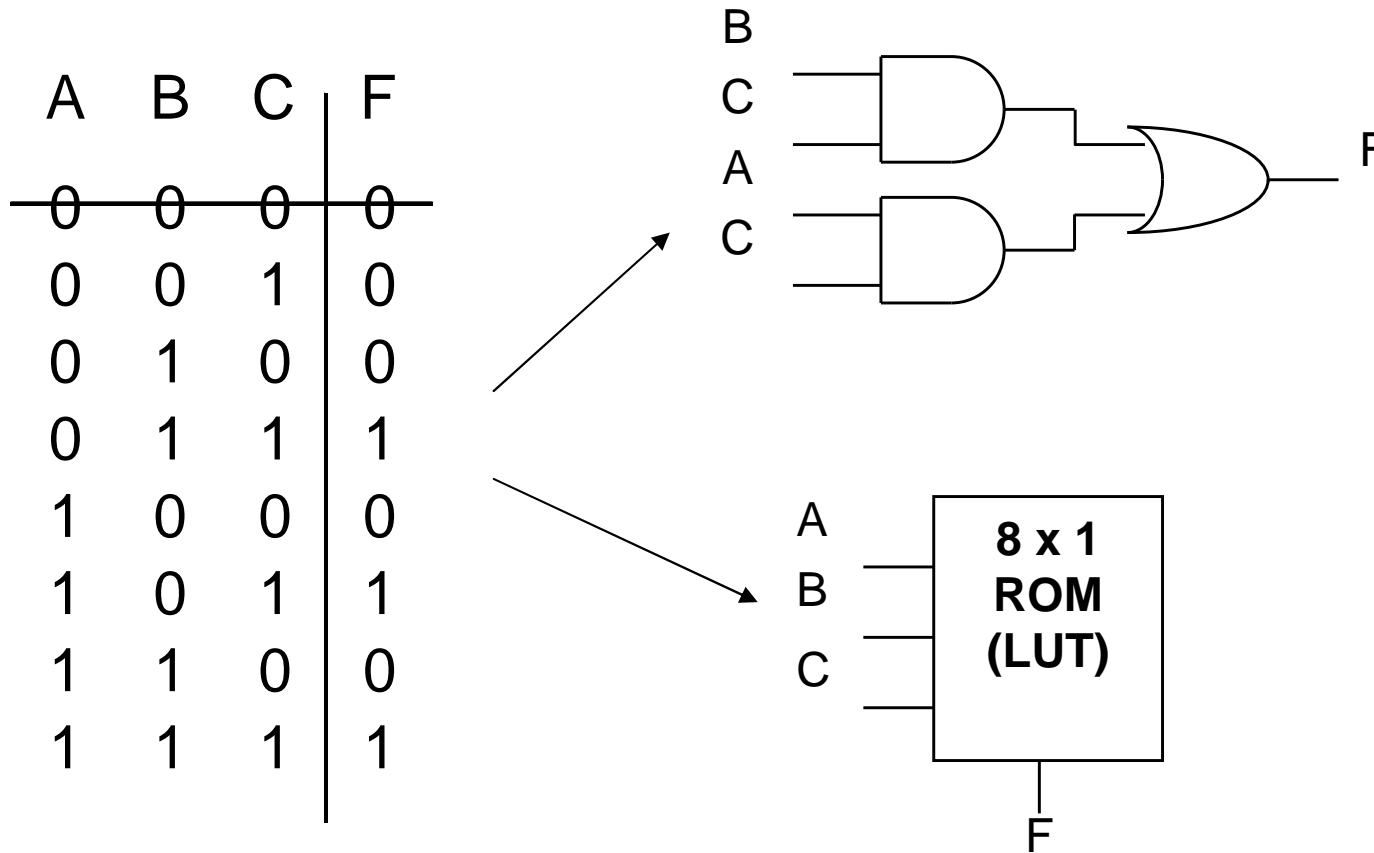
FPGA

Slides from BYU Computer Engineering

**BYU**

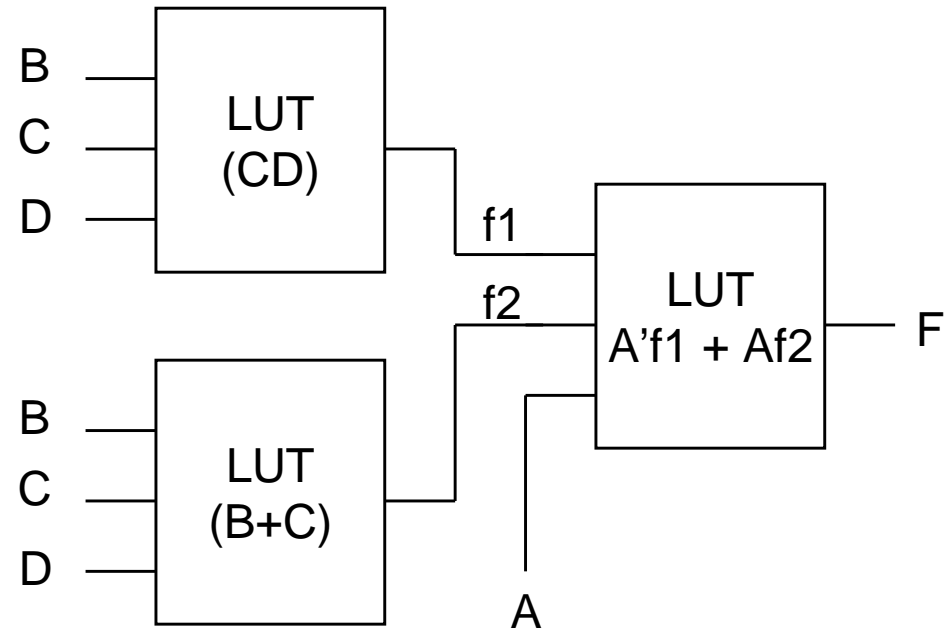
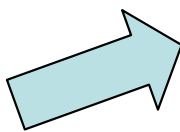
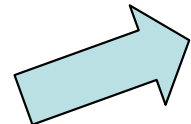
Computer Engineering  
Electrical Engineering

# Using ROM as Combinational Logic



# Mapping Larger Functions To ROMs

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

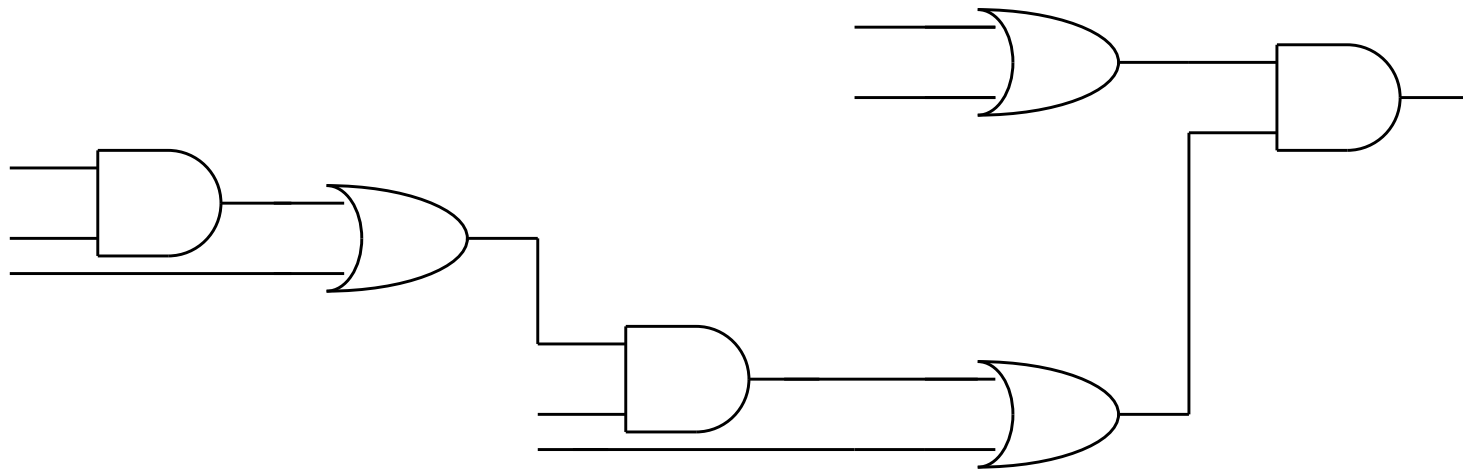


Very similar to how we decomposed functions to implement with MUX blocks...

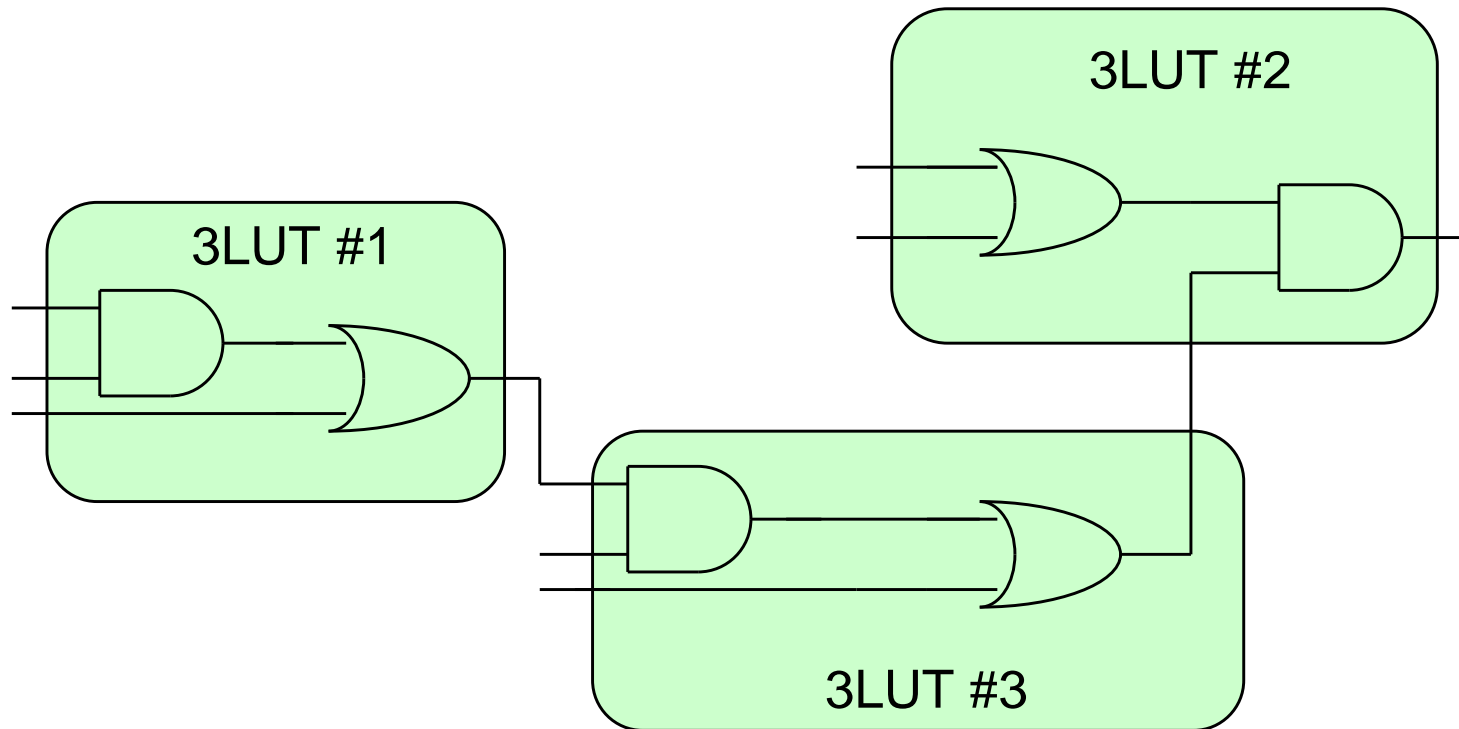
# ROM vs. LUT

- A ROM can be used as a lookup table (LUT)
- An FPGA contains many, many such LUTs
  - Possibly hundreds of thousands
- A 3LUT has 3 inputs and 1 output
- A 4LUT has 4 inputs and 1 output
- 4LUTs are the most common...

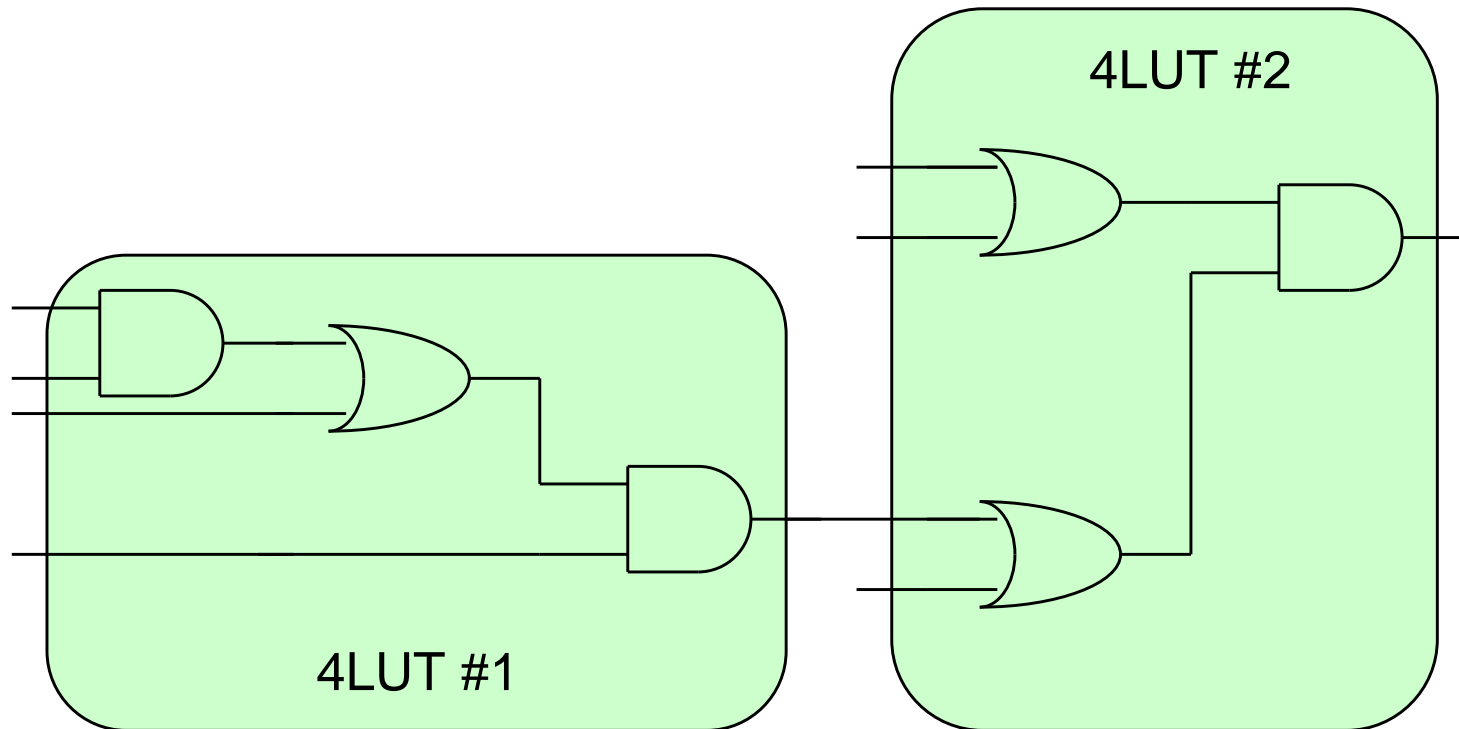
# Mapping a Gate Network to LUTs



# Mapping a Gate Network to 3LUTs



# Mapping Same Network to 4LUTs



# Mapping Equations to LUTs

- How many 4LUTs do the following functions require?

$$F = a$$

$$F = abcd$$

$$F = a'b'c'd + a'bcd' + a'b'c'd' + a'b'cd$$

$$F = a + b + c + d$$

$$F = (a + b + c + d)(a' + b' + c' + d')$$

- Each equation requires a single 4LUT!

Number of LUTs isn't a function of equation complexity,  
but a function of number of unique inputs

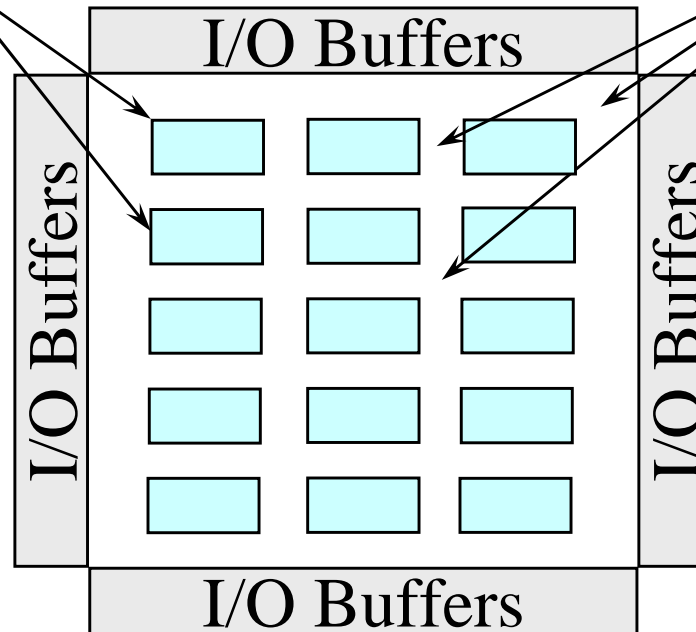


# FPGAs - What Are They?

Programmable Logic Elements  
(LEs)

Programmable wiring areas

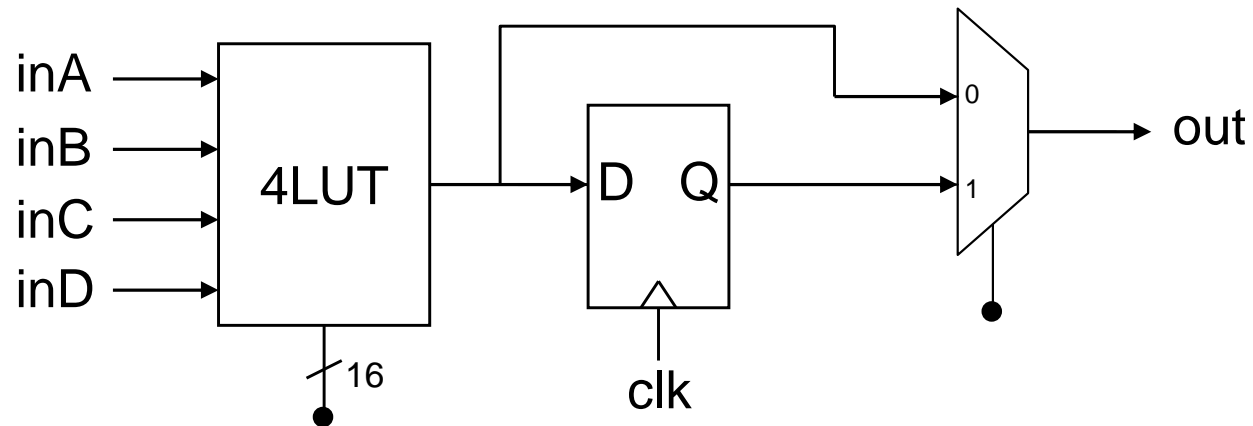
I/O Buffers communicate  
between FPGA and  
the outside world



An FPGA is a Programmable Logic Device (PLD)  
It can be programmed to perform any function desired.

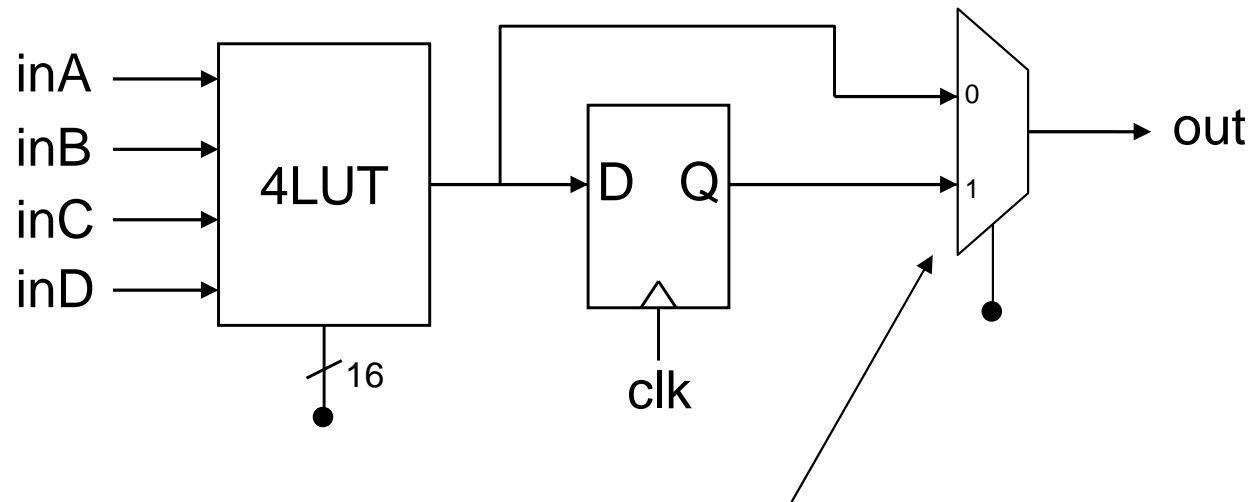
# Programmable Logic Elements (LEs)

- Typically contain a LUT, wires, and storage



# Programmable Logic Elements (LEs)

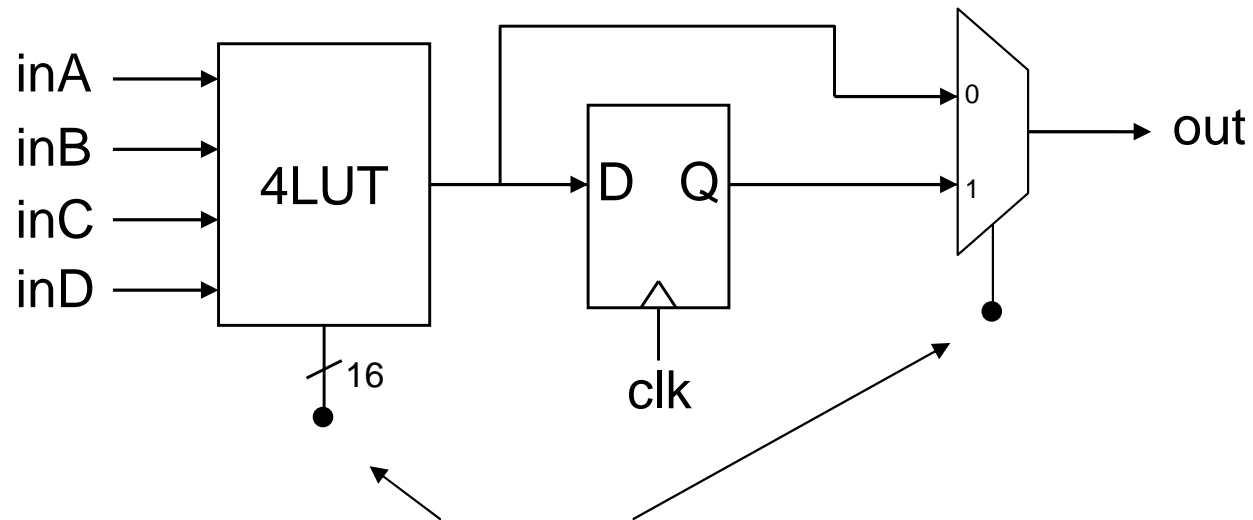
- Typically contain a LUT, wires, and storage



This mux provides a registered or unregistered function of the 4 input variables

# Programmable Logic Elements (LEs)

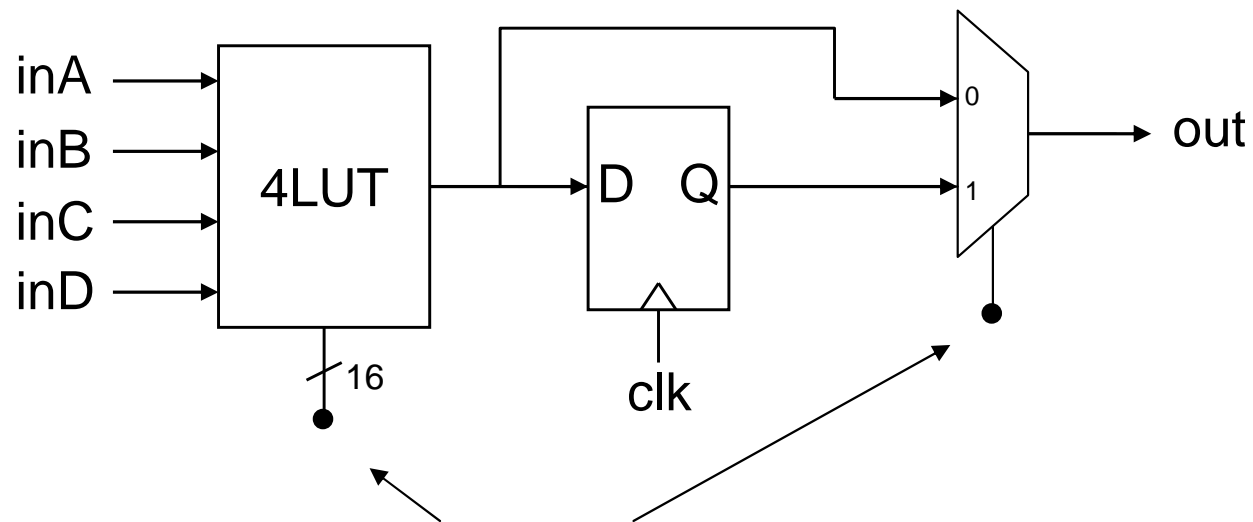
- Typically contain a LUT, wires, and storage



Black dots indicate programming bits  
(configuration bits)

# Programmable Logic Elements (LEs)

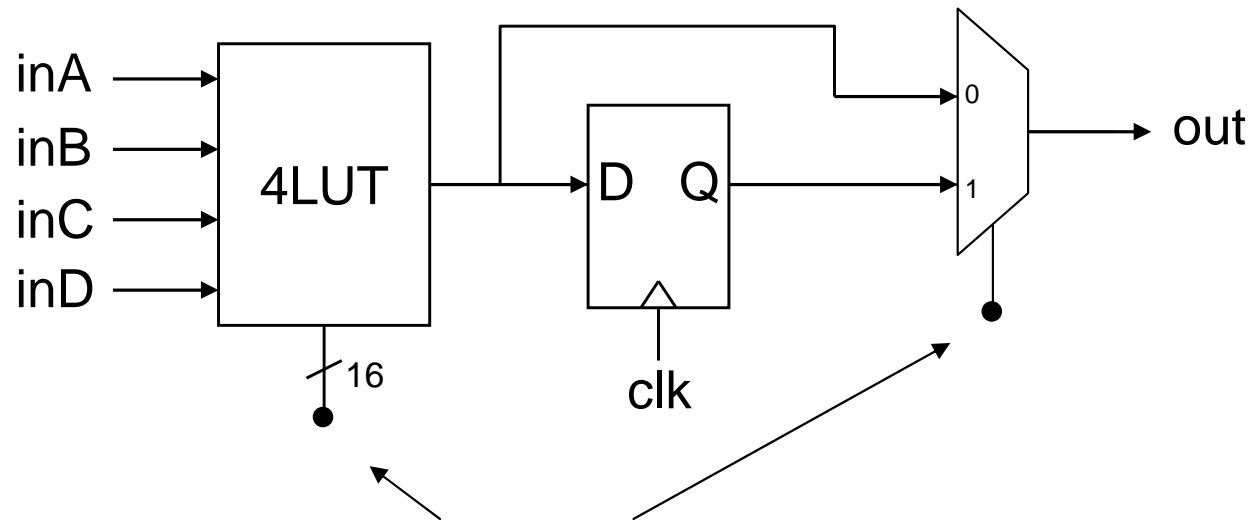
- Typically contain a LUT, wires, and storage



On power up, configuration bits are loaded into FPGA  
This customizes its operation (LUT function, MUX selection)

# Programmable Logic Elements (LEs)

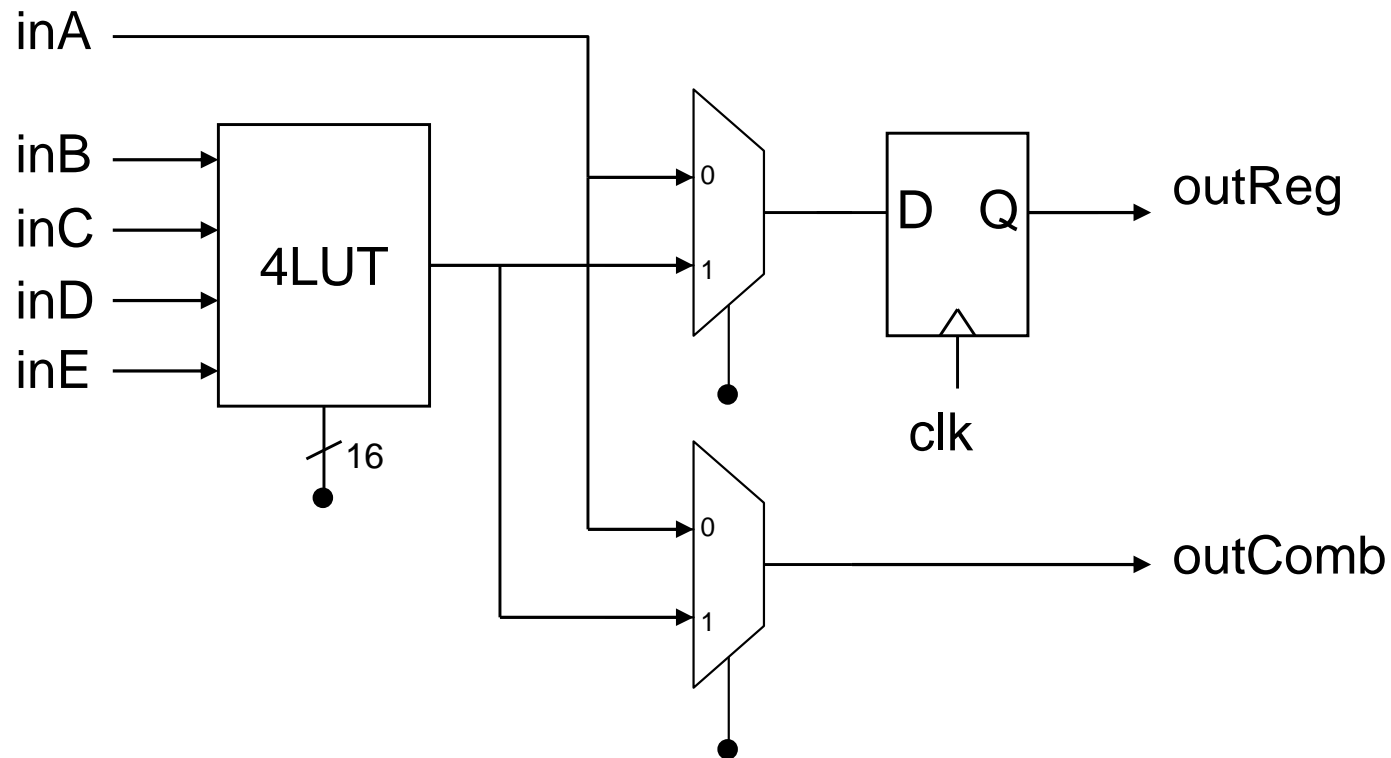
- Typically contain a LUT, wires, and storage



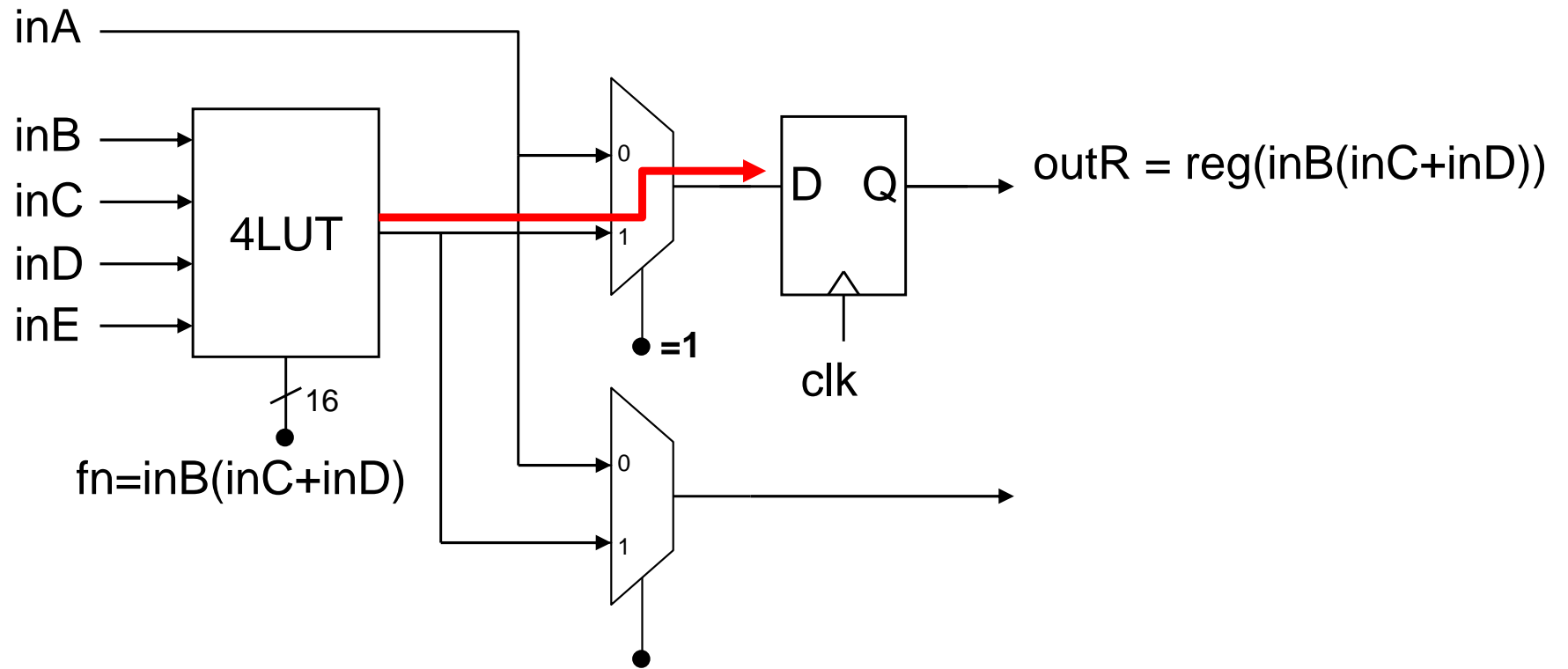
Typically, configuration bits are not changed during circuit operation  
(but, some FPGA's are *dynamically reconfigurable*)

# Another LE Structure

- Provides two outputs
  - One combinational, one registered

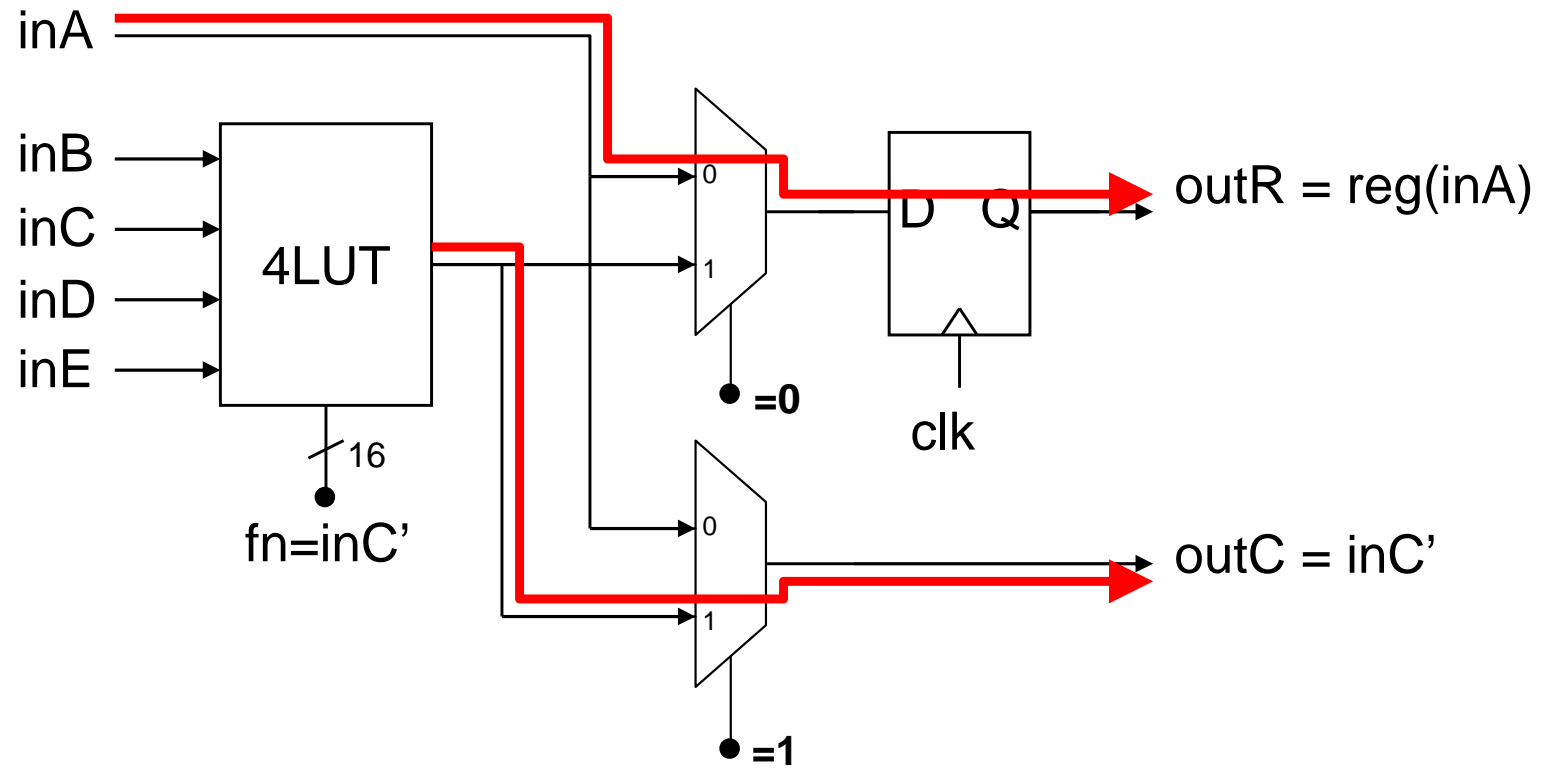


# One Configuration

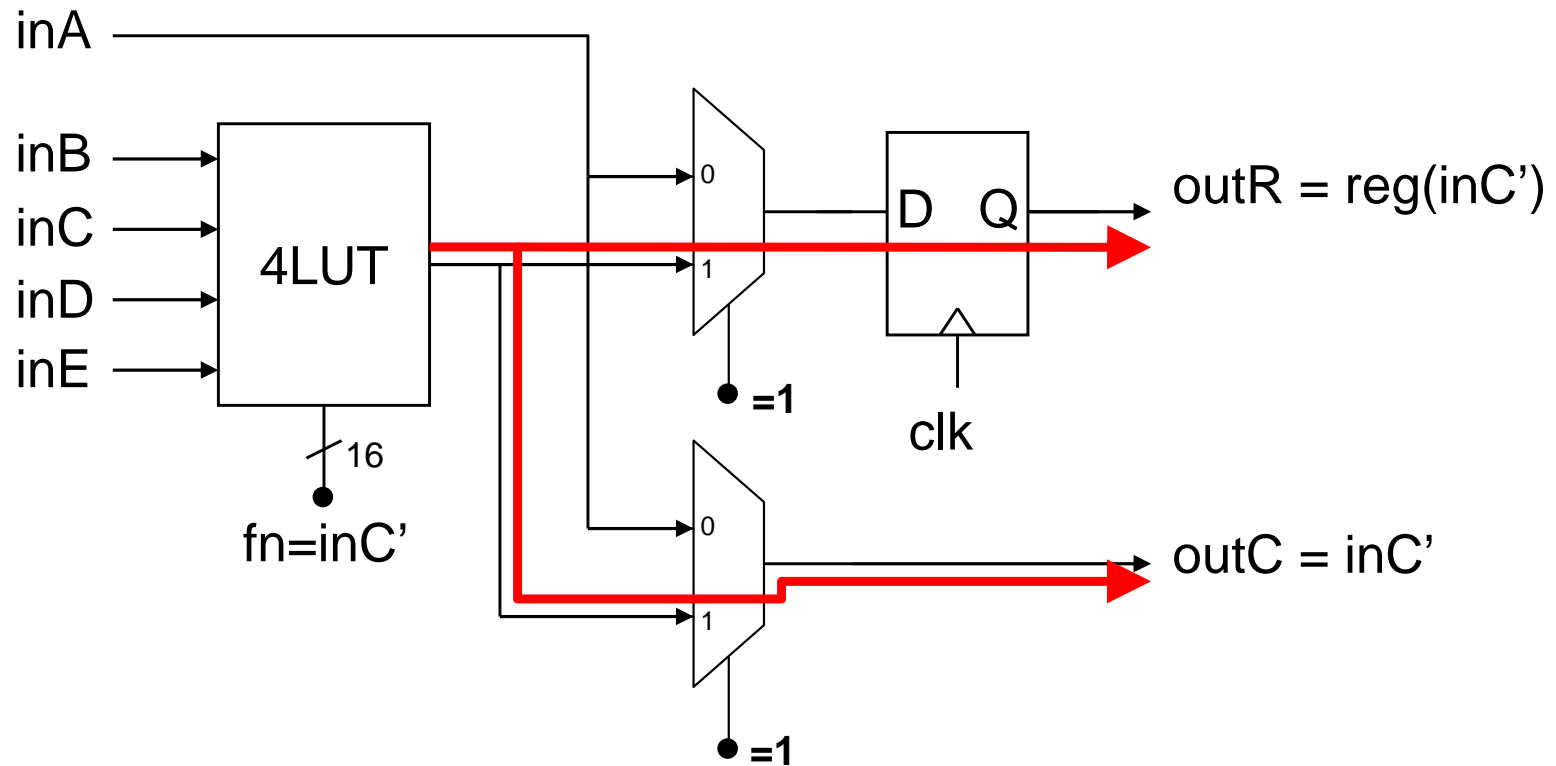




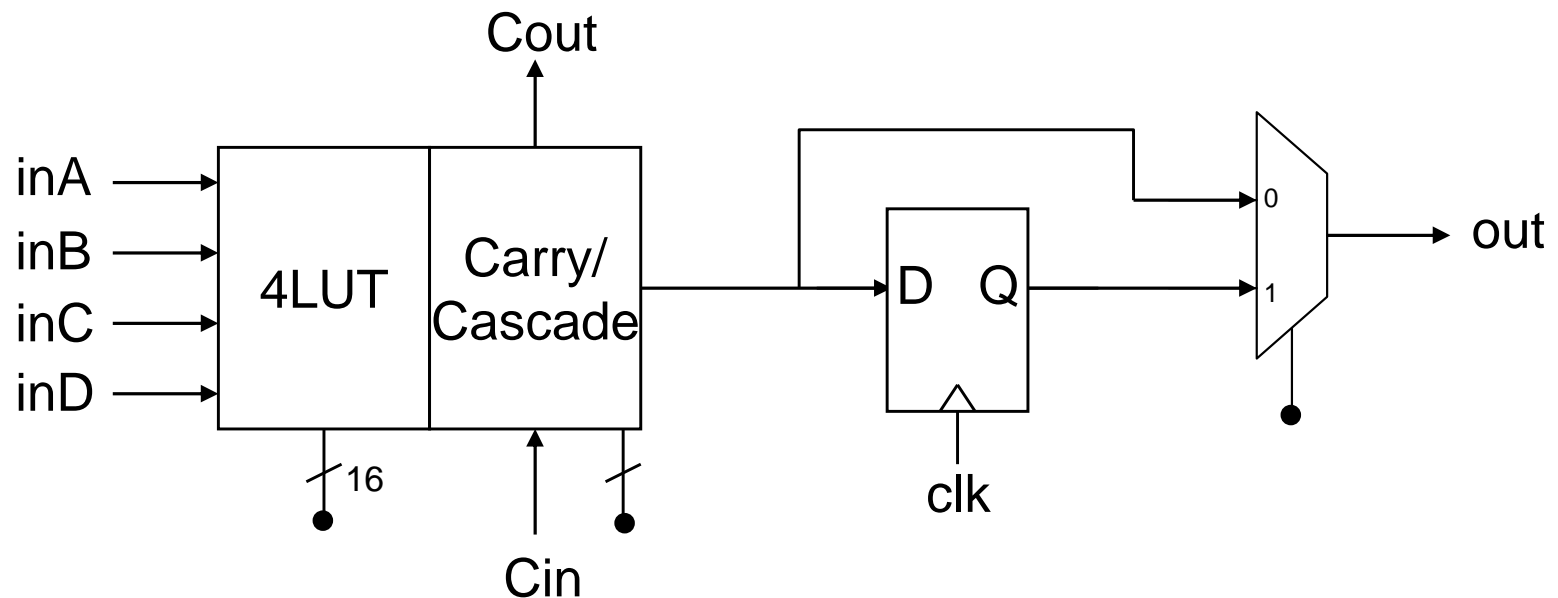
# Another Configuration



# Yet Another Configuration



# An LE With Carry/Cascade Logic

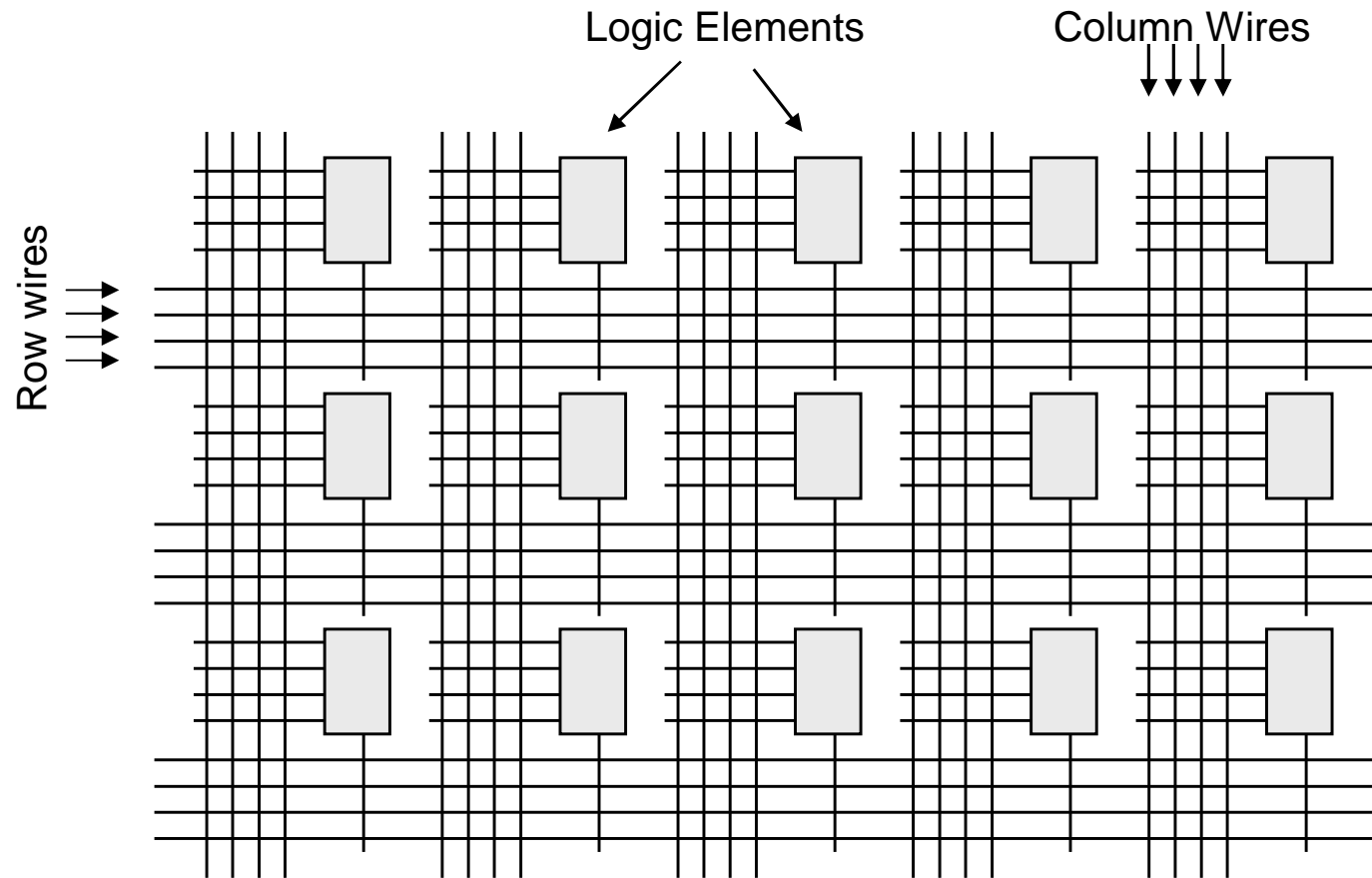


Can do two functions at once (Sum and Cout)

Carry/cascade logic optimized for add/subtract and wide AND, OR, ...

Cin and Cout have dedicated connections to neighboring LEs  $\Rightarrow$  fast carry chains  $\Rightarrow$  fast arithmetic

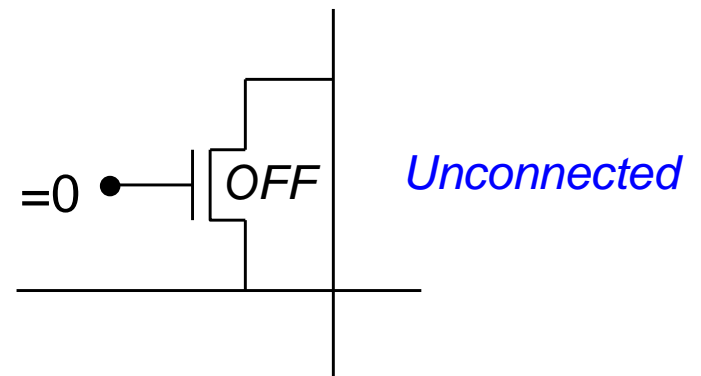
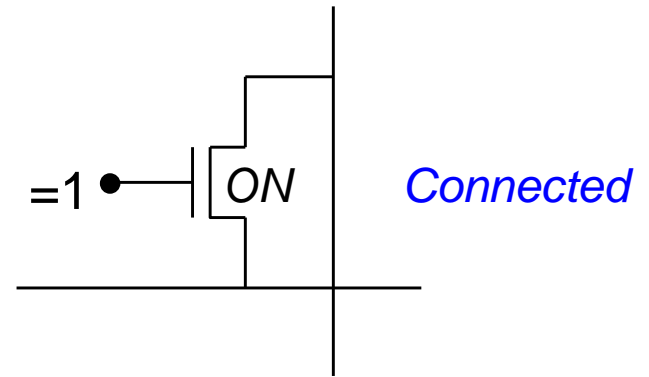
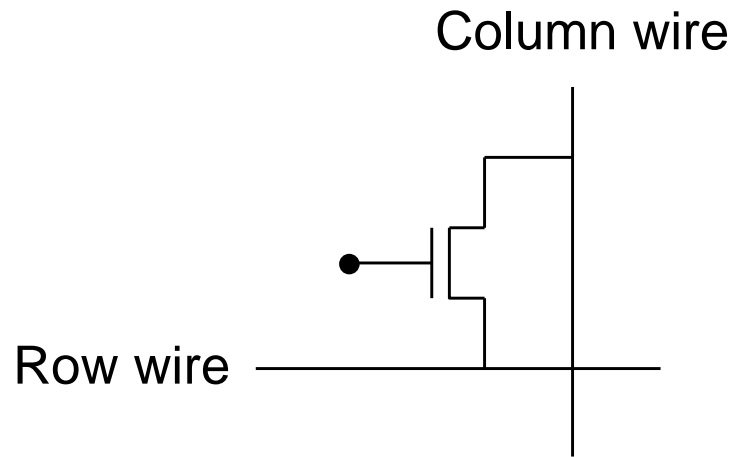
# An FPGA Architecture (Island Style)



Each LE is configured to do a function

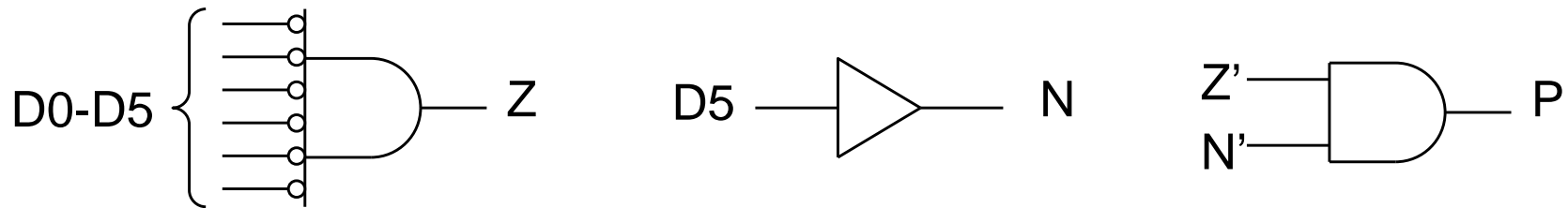
Wire intersections are programmed to either connect or not

# Programmable Interconnect Junction



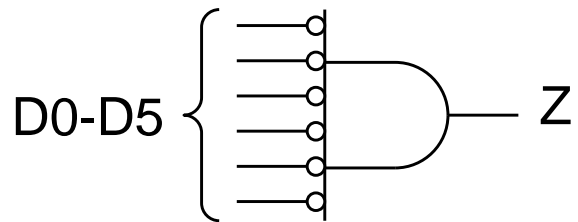
# Example Problem

- Generate the N, Z, P status flags for a 6-bit microprocessor

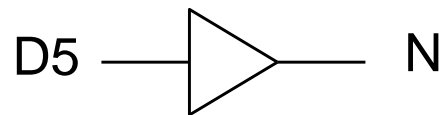


# Example Problem

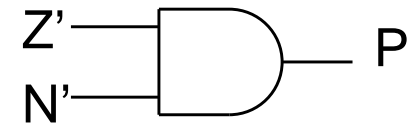
- Generate the N, Z, P status flags for a 6-bit microprocessor



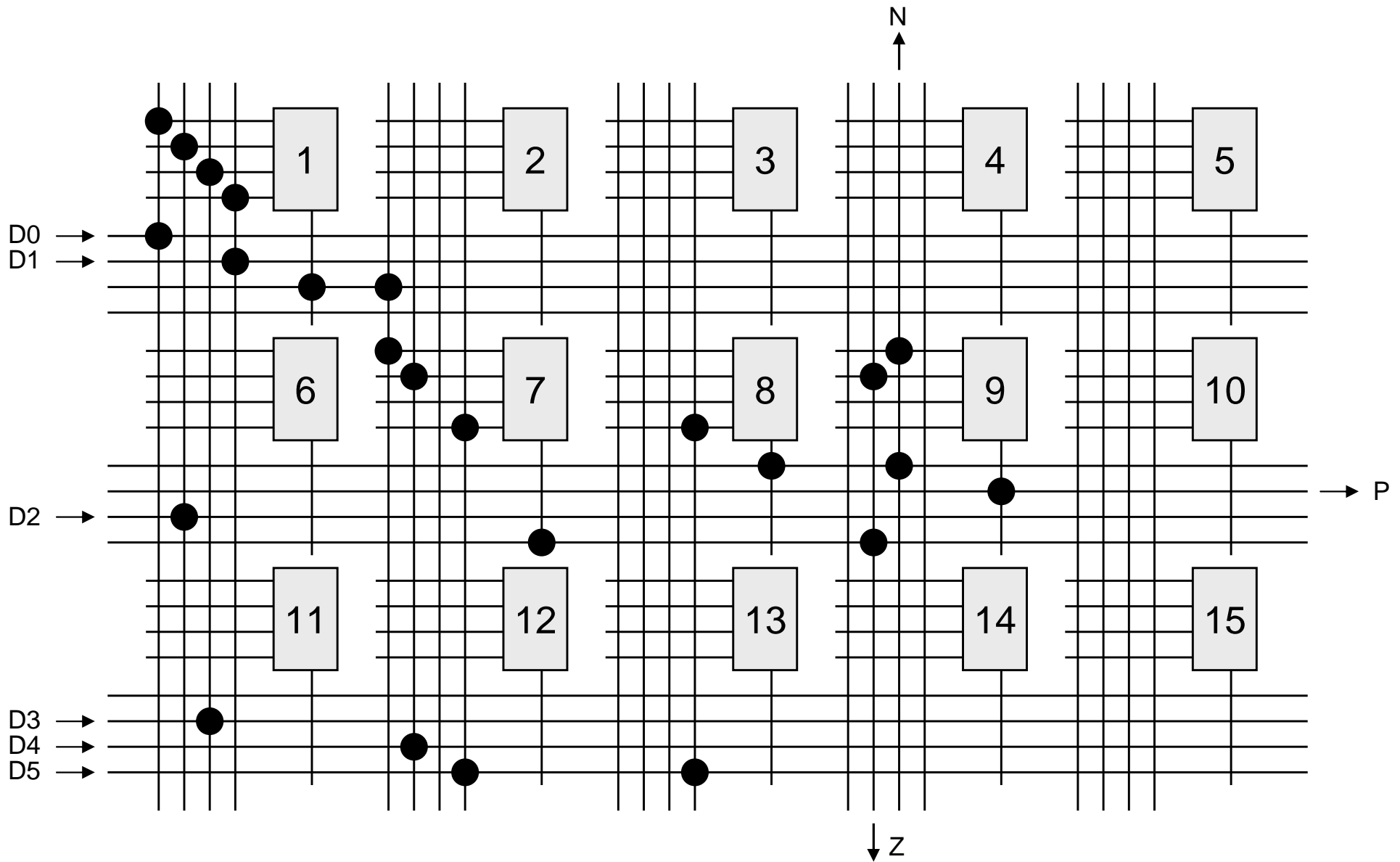
Will require 2 4LUTs



Can be done with wiring only or with 1 4LUT



Will require 1 4LUT



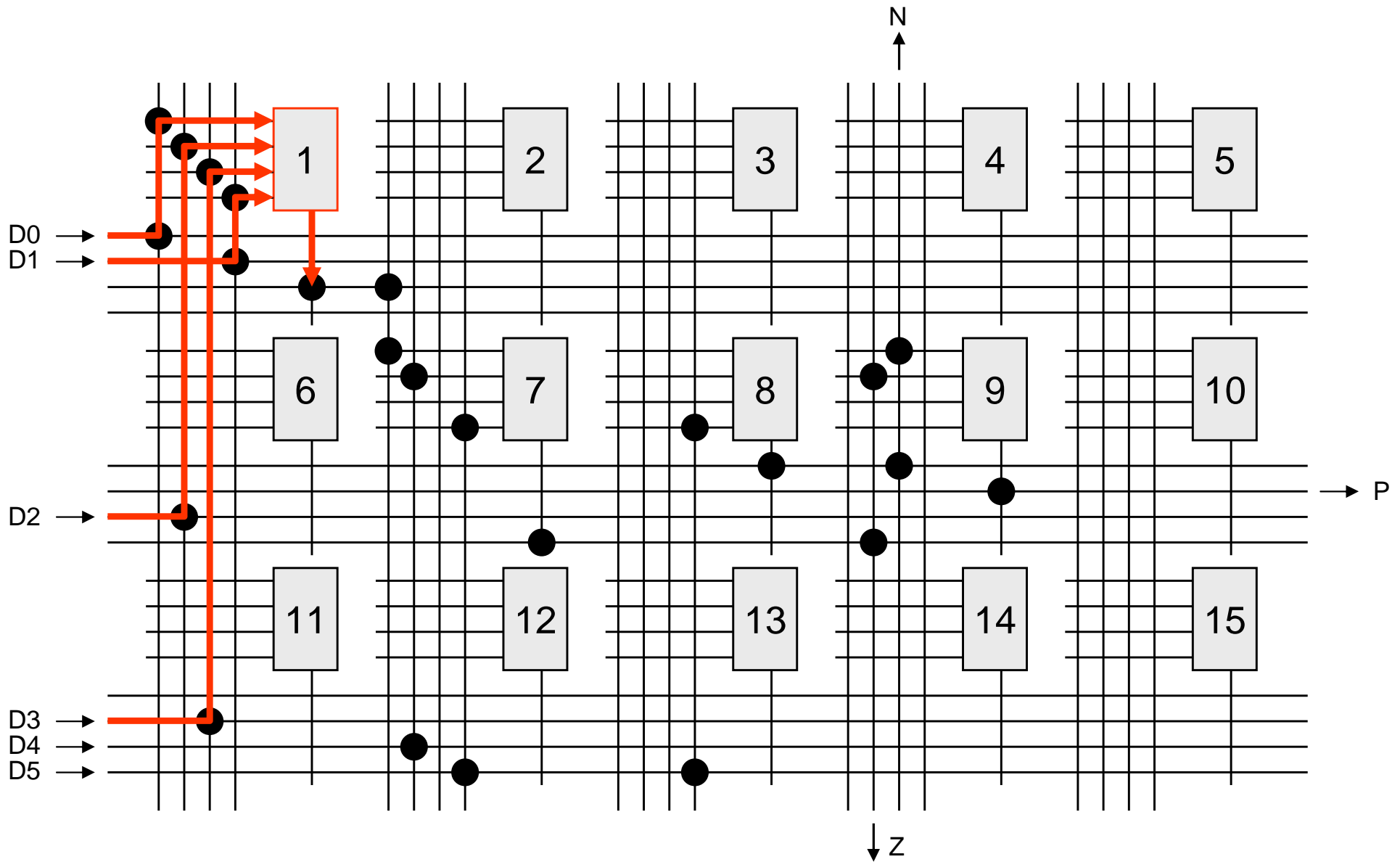
LUT #1:  $F1 = D0' \cdot D1' \cdot D2' \cdot D3'$

LUT #7:  $F2 = F1 \cdot D4' \cdot D5' \Leftrightarrow \mathbf{Z}$  output

LUT #8:  $F3 = D5 \Leftrightarrow \mathbf{N}$  output

LUT #9:  $F4 = Z' \cdot N' \Leftrightarrow \mathbf{P}$  output



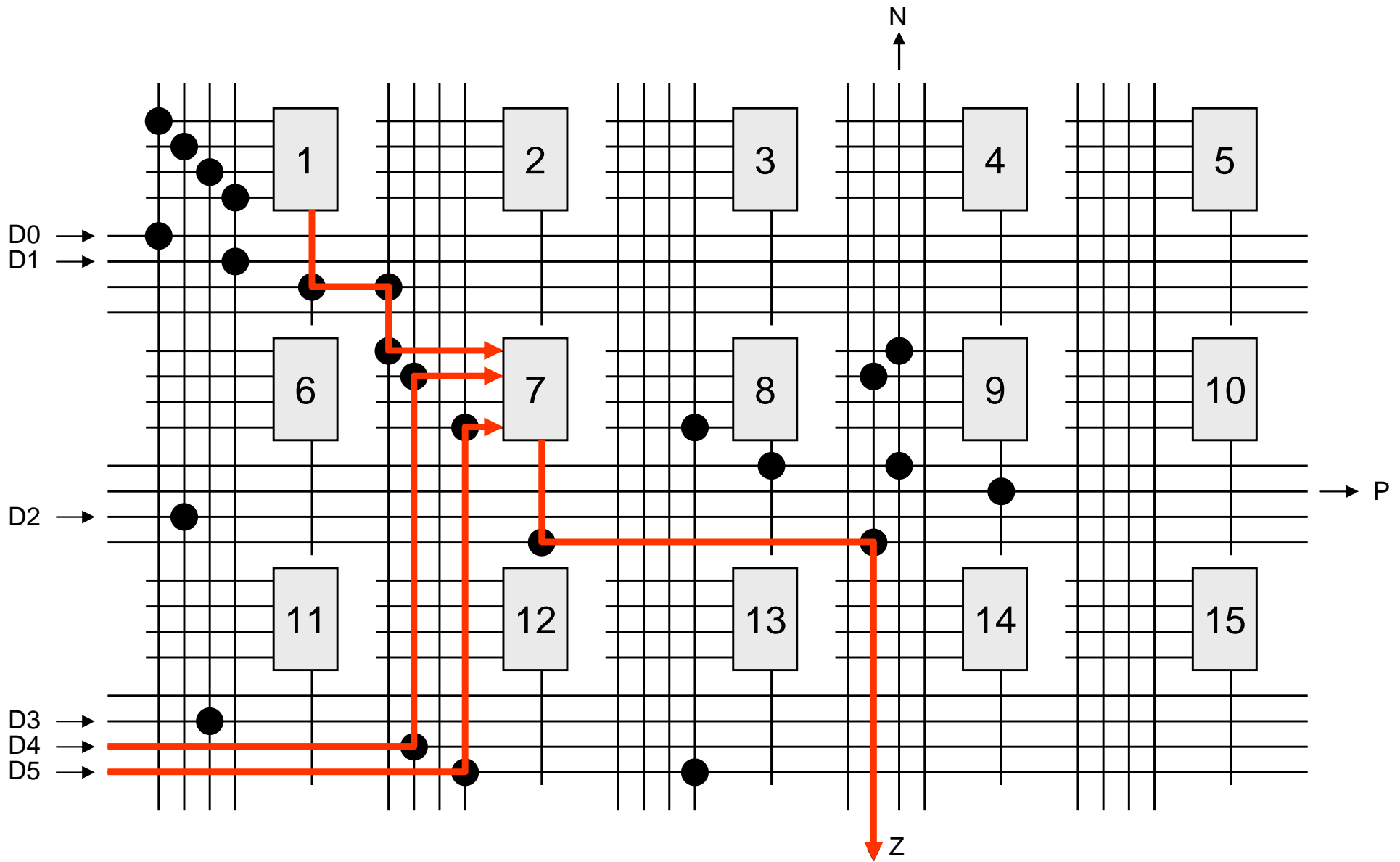


LUT #1:  $F1 = D0' \cdot D1' \cdot D2' \cdot D3'$

LUT #7:  $F2 = F1 \cdot D4' \cdot D5' \Leftrightarrow Z \text{ output}$

LUT #8:  $F3 = D5 \Leftrightarrow N \text{ output}$

LUT #9:  $F4 = Z' \cdot N' \Leftrightarrow P \text{ output}$

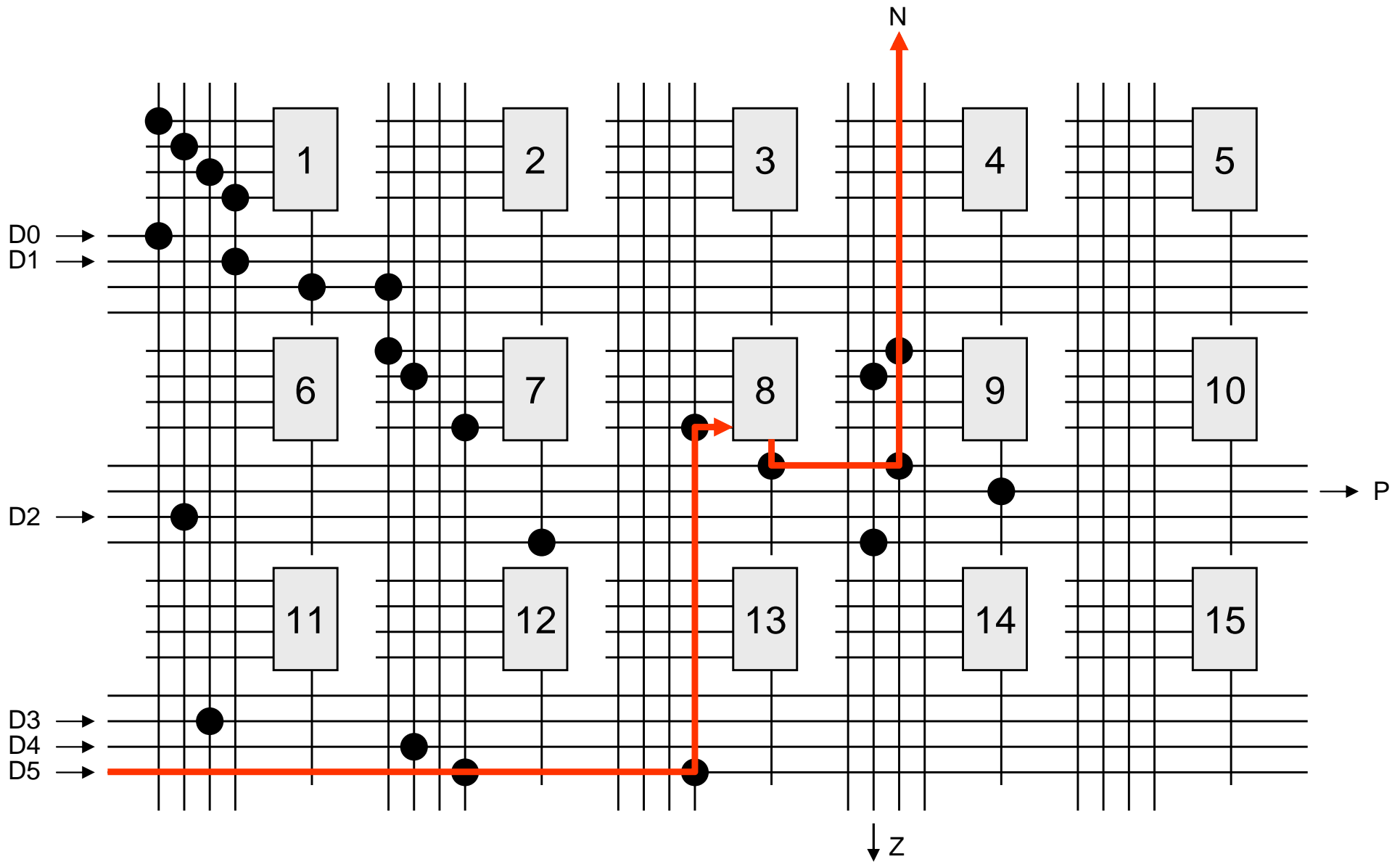


LUT #1:  $F1 = D0' \cdot D1' \cdot D2' \cdot D3'$

LUT #7:  $F2 = F1 \cdot D4' \cdot D5' \Leftrightarrow Z \text{ output}$

LUT #8:  $F3 = D5 \Leftrightarrow N \text{ output}$

LUT #9:  $F4 = Z' \cdot N' \Leftrightarrow P \text{ output}$

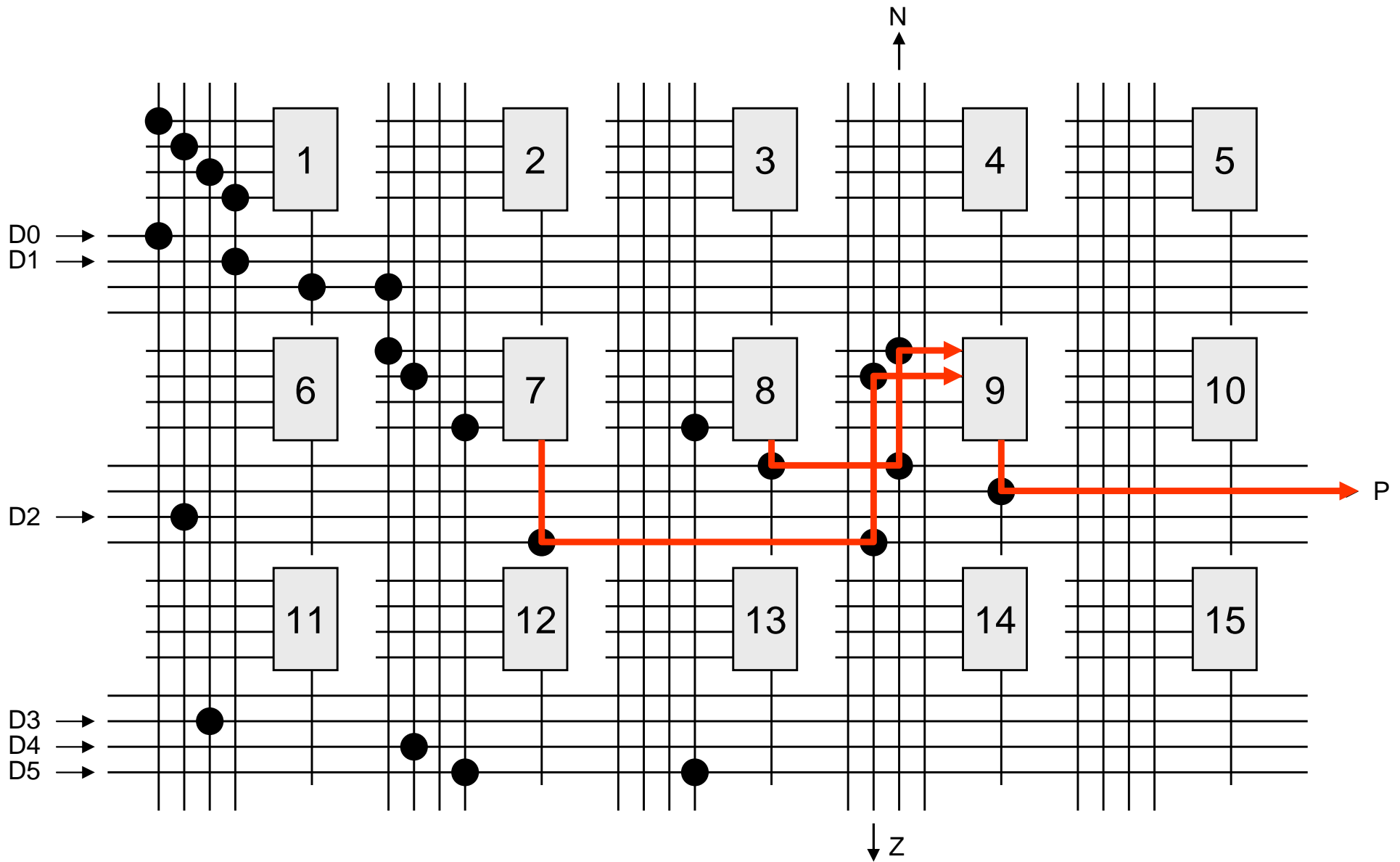


LUT #1:  $F1 = D0' \cdot D1' \cdot D2' \cdot D3'$

LUT #7:  $F2 = F1 \cdot D4' \cdot D5' \Leftrightarrow Z \text{ output}$

LUT #8:  $F3 = D5 \Leftrightarrow N \text{ output}$

LUT #9:  $F4 = Z' \cdot N' \Leftrightarrow P \text{ output}$



LUT #1:  $F1 = D0' \cdot D1' \cdot D2' \cdot D3'$

LUT #7:  $F2 = F1 \cdot D4' \cdot D5' \Leftrightarrow \mathbf{Z}$  output

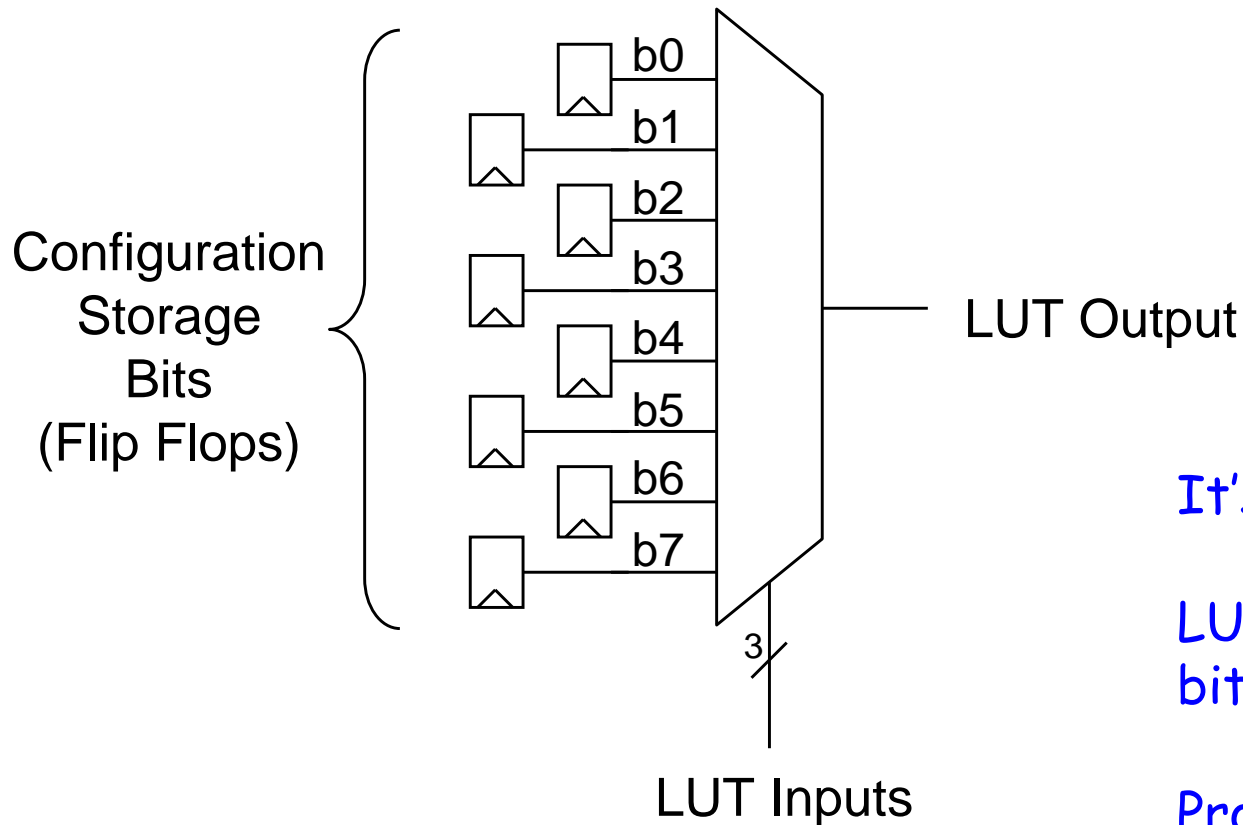
LUT #8:  $F3 = D5 \Leftrightarrow \mathbf{N}$  output

LUT #9:  $F4 = Z' \cdot N' \Leftrightarrow \mathbf{P}$  output

# Configuring an FPGA

- Most FPGAs have a configuration input pin
  - Configuration bits are shifted into FPGA using this pin, one bit per cycle
  - Configuration bits in FPGA linked into a long shift register (SIPO)
- Examples on following slides are *conceptual*
  - Commercial devices slightly different

# Structure of a 3LUT



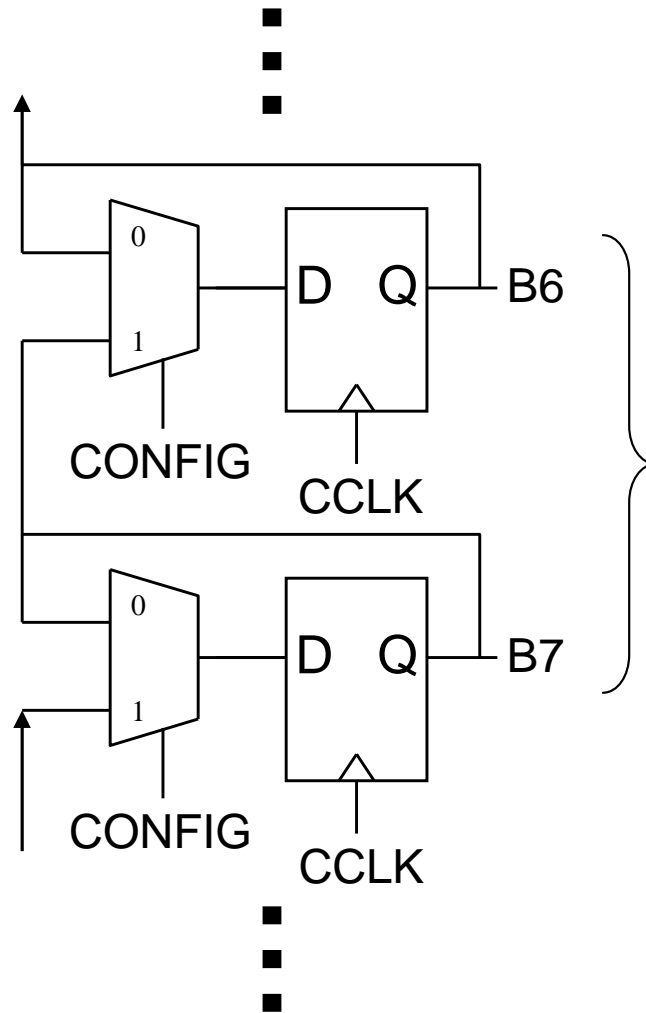
It's just an 8:1 MUX

LUT inputs select which config bit is sent to LUT output

Programming LUT function ⇔ setting configuration bits

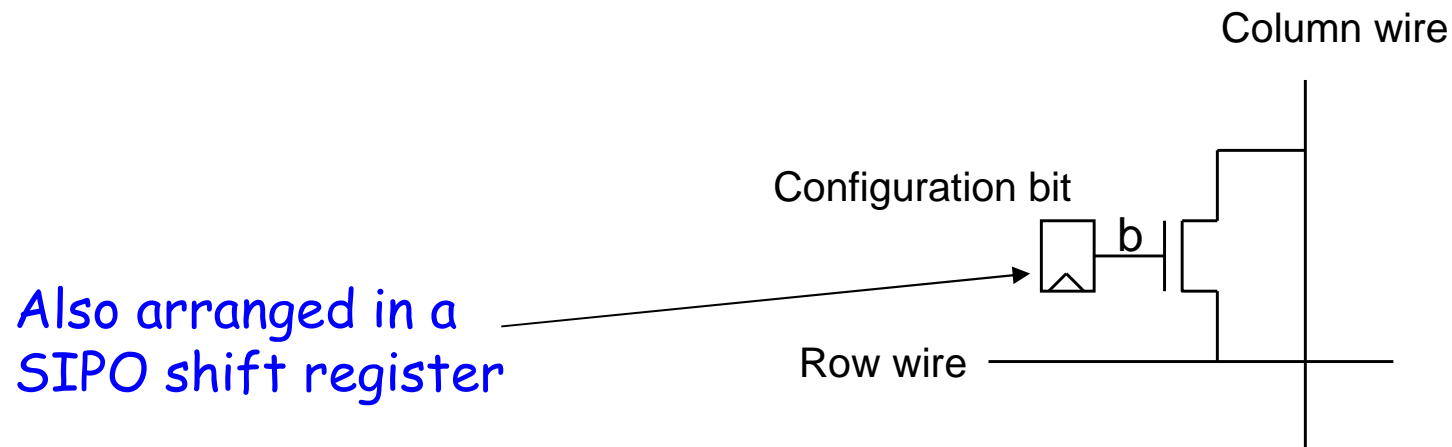
# How are the Configuration Bit Flip Flops Loaded?

A serial-in/parallel-out (SIPO) shift register



These are the configuration bits which the LUT selects from

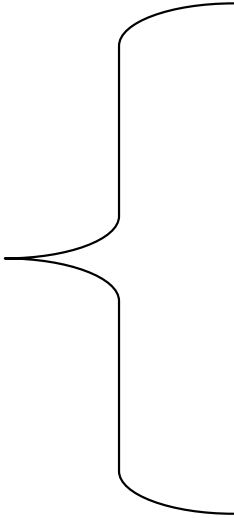
# Configuring the Programmable Interconnect



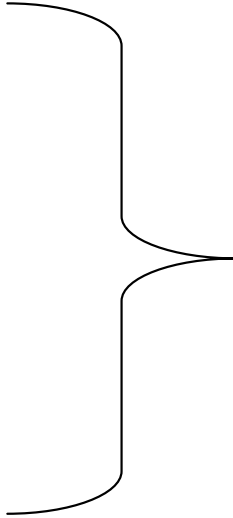


# Additional FPGA Features

Found in commercial FPGAs



At this point, can use following section's slides  
on advanced FPGA features or  
simply show data sheets for commercial FPGA's



# Configurable Input/Output

- I/O blocks allow internal FPGA signals to connect to external pins
- Configurable I/O block features:
  - Can be configured as inputs or outputs
  - Electrostatic discharge (ESD) protection
  - Input signal conditioning (voltage levels, ...)
  - Output drive
    - Fast, slow
    - Strong, weak
    - Tri-state
  - Handles a variety of electrical standards
    - LVTTTL
    - LVCMOS
    - PCI
    - LVDS
    - ...
- I/O blocks are configured when the rest of the chip is configured

# Configuration Storage

- Actual configuration storage is *not* flip flops
  - Many use SRAM memory cells
- Different technologies
  - Program once
    - Fuse, anti-fuse configuration
  - Program multiple times
    - SRAM-based configuration
- Some chips can be *partially reconfigured*, even while rest of chip is running

# Programmable Interconnections

- Expensive to put programmable connections at *every* wire junction
  - Commercial parts use partially populated junctions
  - Little or no loss of routing flexibility

# Hierarchical Routing

- Collection of short, medium, and long wires
  - Both rows and columns
- Signals use wires that go distance needed
- CAD tools make determinations
- Like alleys, streets, expressways in a city
  - Longer distance ⇔ more limited access points

# Clustered LE's

- Cluster:  $\geq 1$  LE
  - Example: Cluster of 4
  - Dedicated wires within groups of 4 LE's
- Closely related to hierarchical routing
- Higher performance
- Different vendors  $\Leftrightarrow$  different clustering

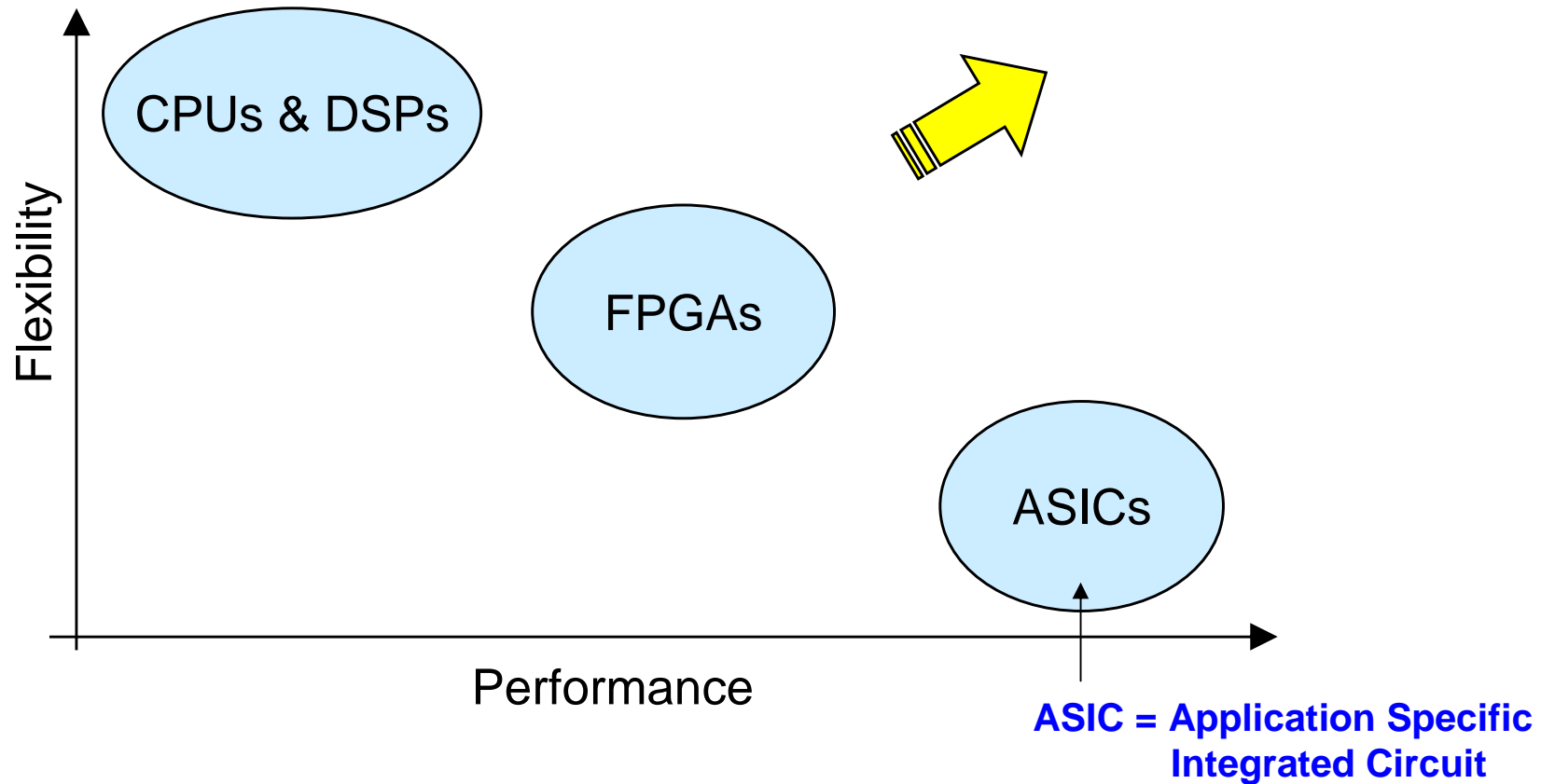
# Embedded Function Units

- Place the following into the FPGA fabric:
  - Multipliers
  - Memories (Block RAM)
  - I/O interfaces (*e.g.* gigabit serial links or Ethernet PHY/MAC)
  - CPU's (PowerPC, ARM, ...)
- Result: higher speed, higher density, lower power designs



# FPGAs Compared To Other Technologies

# Performance vs. Flexibility



Goal: the performance of ASIC's with the flexibility of programmable processors.

# FPGAs vs. CPUs

## Flexibility

- Any function can be programmed to run on a CPU
  - Programming is relatively simple to do
- Any function can also be configured onto an FPGA
  - Design is much more difficult and time consuming since it's a digital logic design
  - "Hardware is harder than software"
- Both CPUs and FPGAs can be reprogrammed (reconfigured) in the field
- The winner: CPUs

# FPGAs vs. CPUs

## Performance

- 10+ years of research  $\Leftrightarrow$  10x-100x performance advantage for FPGAs!
- Custom hardware
  - No fetch, decode, or memory access overhead
  - Significant parallelism (e.g., pipelining)
- Case in point: 1998 SONAR beamformer on an FPGA
  - BYU Configurable Computing Lab research
  - $2 \times 10^9$  FLOP/sec *sustained*
  - 10x-80x faster than comparable CPU technology
- The winner: FPGAs

# FPGAs vs. ASICs

## Flexibility

- *ASIC* is a *static* hardware design
  - Wires and transistors fixed at manufacture time
  - Cannot be upgraded (reprogrammed) in the field
- *FPGA* can be configured (and reconfigured)
  - *FPGA* platform ⇔ used for a variety of applications
    - *CCM* = Custom Computing Machine (based on *FPGAs*)
- The winner: *FPGAs*

# FPGAs vs. ASICs

## Performance

- Both ASICs and FPGAs are *custom* hardware designs
- FPGA *configurability* comes at great cost
  - FPGA density is *much* lower than that of an ASIC
  - Low density means logic must be physically spread out
- Result:
  - FPGAs are slower, physically larger, and more power hungry than a custom ASIC chip
  - For some/many applications FPGAs may...
    - ...be too slow
    - ...not hold enough logic
    - ...be too power hungry
- The winner: ASICs

# FPGAs vs. ASICs

## Cost

- Once the design is complete:
  - FPGAs can be purchased off-the-shelf and programmed in seconds
    - Negligible up-front costs
    - Inexpensive for small volume
    - Per part cost is high
  - ASICs require time-consuming, expensive manufacturing process
    - Fabrication facility (> \$1 billion)
    - Significant up-front costs
    - Expensive for small volume
    - Per part cost is very low, when volume is high
- The winner: Depends on volume and application!

# FPGA vs. ASIC Production Cost Example

- FPGA

- Non-recurring engineering (NRE) costs = \$0
- Per-part cost = \$20 (prices vary from a few \$'s to \$1000's)
- Cost to produce 100 = \$2K
- Cost to produce  $10^6$  = \$20M

- ASIC

- NRE costs = \$5M
- Per-part cost = \$1
- Cost to produce 100 = \$5M + \$100  $\approx$  \$5M
- Cost to produce  $10^6$  = \$5M + \$1M = \$6M

Volume produced makes a big difference!



# FPGAs vs. ASICs

## Time to Market and Bug Fixes

- Manufacturing Time
  - Shorter time to market makes money sooner
  - FPGA = seconds to configure
  - ASIC = weeks to months
- Fixing Bugs
  - FPGA = modify design + reconfigure FPGA
    - Reconfigurability can make debugging easier
  - ASIC = modify design + remanufacture
- The winner: FPGA

# FPGAs vs. ASICs - Summary

- Two very different technologies
  - Flexibility: FPGA wins
  - Performance: ASIC wins
  - Risk: ?
  - Cost: ?
- Complex Decision
  - Only partially based on technical issues
  - Significant business issues
    - Cost
    - Risk
    - Time-to-market
    - Market characteristics (elasticity, price sensitivity, ...)

# Design for FPGAs

1. Draw schematics or write HDL
2. CAD tool maps design to LUTs + FFs
3. CAD tool generates *bitstream*
4. Load bitstream to *configure* the FPGA

# Design for ASICs

1. Draw schematics or write HDL
  2. CAD tool maps design to physical silicon layout
  3. Send layout data to silicon *fab*
  4. Finished chip does intended function
- Step 1 fairly similar for FPGAs and ASICs
    - Major differences in later steps

# For More Information

- FPGA companies' WWW sites tend to have lots of information
  - Catering to a wide range of customers
    - Novice to expert
    - Low volume to high volume
  - Data sheets
  - Application notes
  - Success stories
  - CAD tools