Memory & Programmable Logic

Logic and Digital System Design - CS 303

Sabancı University

Memory Unit

- A device
  - to which binary data is transferred for storage and
  - from which data is available when needed for processing
- When data processing takes place
  - data from the memory is transferred to selected register in the processing unit
  - Intermediate and final results obtained in the processing unit are transferred back to the memory for storage

Memory Unit

- Used to communicate with an input/output device
  - binary information received from an input device is stored in memory
  - information transferred to an output device is taken from memory
- A collection of cells capable of storing a large quantity of binary information
- Two types
  - RAM (random-access memory)
  - ROM (read-only memory)

Classification

- RAM
  - We can read stored information (read operation)
  - Accepts new information for storage (write operation)
  - Perform both read and write operation
- ROM
  - only performs read operation
  - existing information cannot be modified
  - “Programming” the device
    - specifying the binary information and storing it within the programmable device
  - a.k.a. programmable logic device

Programmable Logic Devices

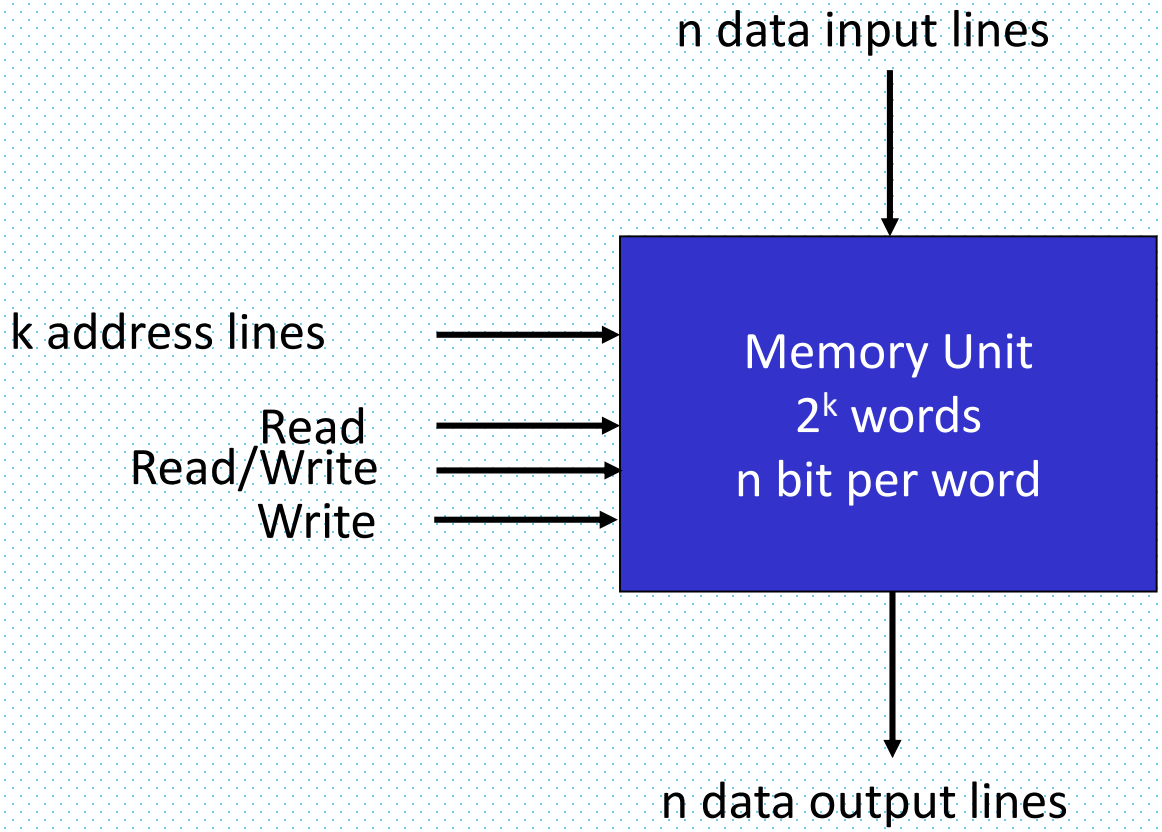
- PLD
  - ROM is one example
  - Programmable Logic Array (PLA)
  - Programmable Array Logic (PAL)
  - Field Programmable Gate Array (FPGA)
- PLD is
  - an integrated circuit (IC) with internal logic gates
  - Interconnection between the gates can be programmed through fuses
  - At the beginning they are all intact
  - By programming we remove some of them, while keeping the others

Random Access Memory (RAM)

- RAM : the reason for the name
  - The time it takes to transfer information to or from any desired random location is always the same.
- Word
  - group of bits in which a memory unit stores information
  - At one time, memory move in and out certain number of bits
  - 8 bit – byte
  - 16 bit
  - 32 bit
  - Capacity is usually given in number of bytes

Memory Unit

- Block diagram

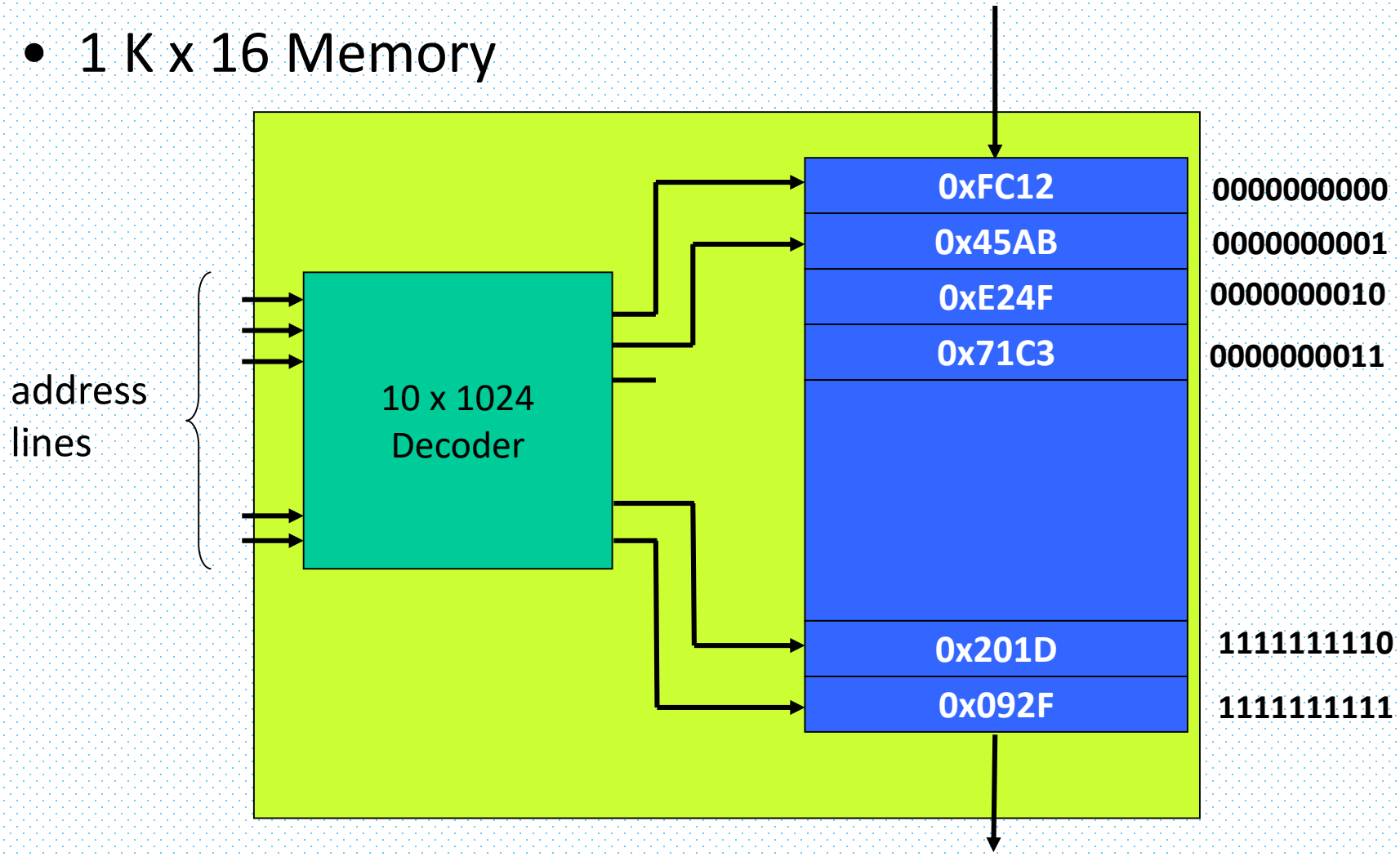


Specification

- A memory unit is specified by
  1. the number of words it contains
  2. number of bits in each word
- Each word (or location for a word) in memory is assigned an identification number
  - address
  - 0 to 2^k-1
- Selection
  - selection process to read and write a word is done by applying k-bit address to the address lines
  - A decoder accepts the address and selects the specified word in the memory

Memory Map and Address Selection

- 1 K x 16 Memory



$1\ K = 2^{10}$ $1\ M = 2^{20}$ $1\ G = 2^{30}$ $1\ T = 2^{40}$

Write and Read Operations

- Write
  - transfer in
- Read
  - transfer out
- Steps for write operation
  1. Apply the binary address of the desired word to the address lines
  2. Apply the data word that is be stored in memory to the (data) input lines
  3. Activate the “write” input

Read Operation

- Steps
  1. Apply the binary address of the desired word to the address lines
  2. Activate the “read” input
  - The desired word will appear on the (data) output lines
  - reading does no affect the content of the word

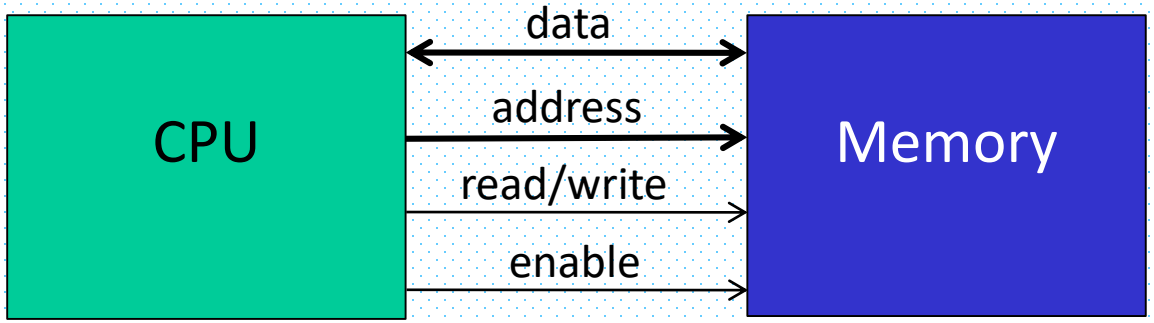
Control Inputs to Memory Chip

- Commercial memory components usually provide a "memory enable" (or "chip select") control input
- memory enable is used to activate a particular memory chip in a multi-chip implementation of a large memory

Memory Enable	Read/Write	Memory Operation
0	X	None
1	0	write
1	1	read

Timing

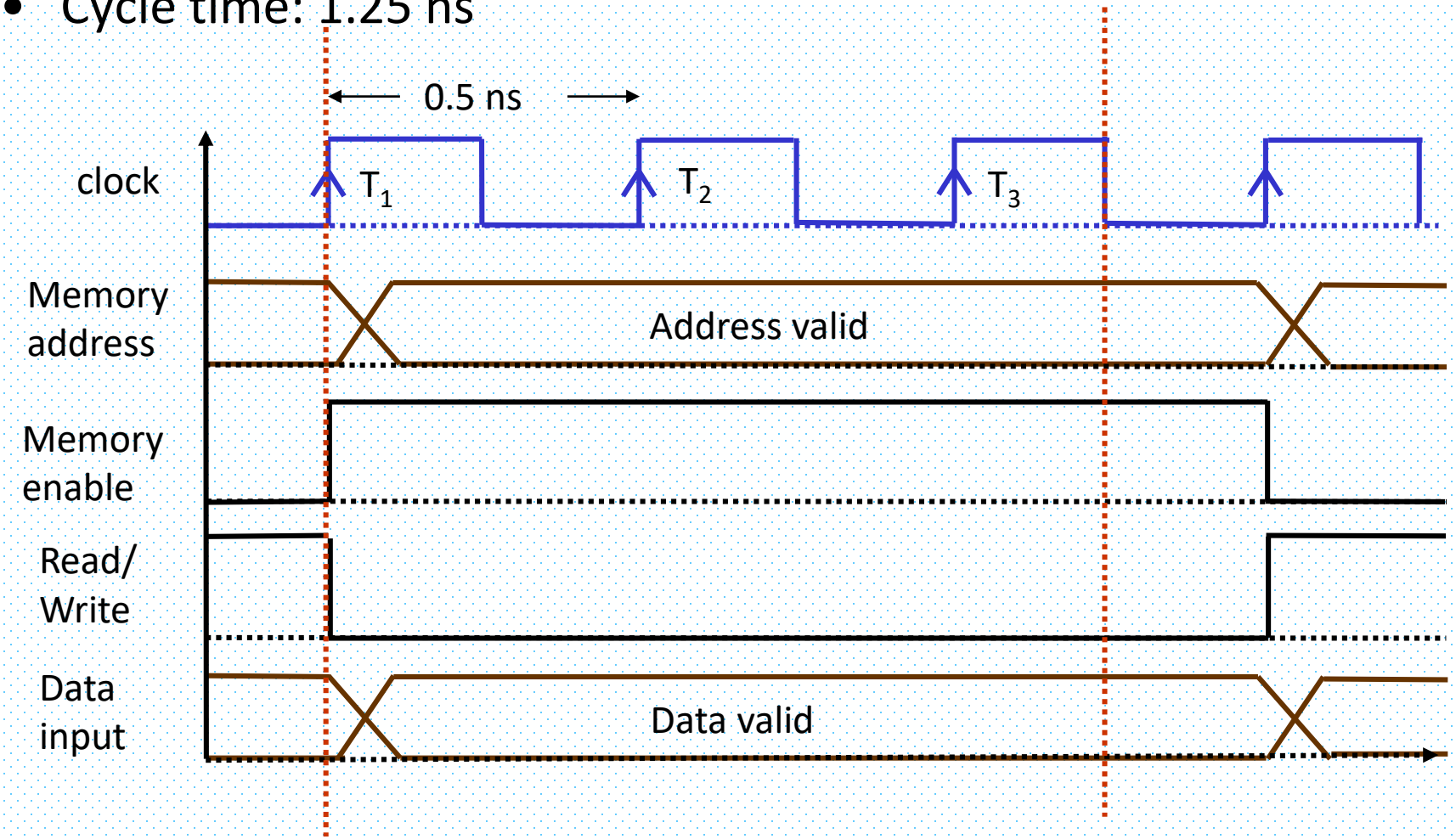
- Memory does not have to use an internal clock
  - It only reacts to the control inputs, e.g., “read” and “write”
  - operation of a memory unit is controlled by an external device (e.g. CPU) that has its own clock
- Access time
  - the time required to select a word and read it
- Cycle time
  - the time required to complete a write operation



Write Cycle

CPU must devote three clock cycles for each write operation

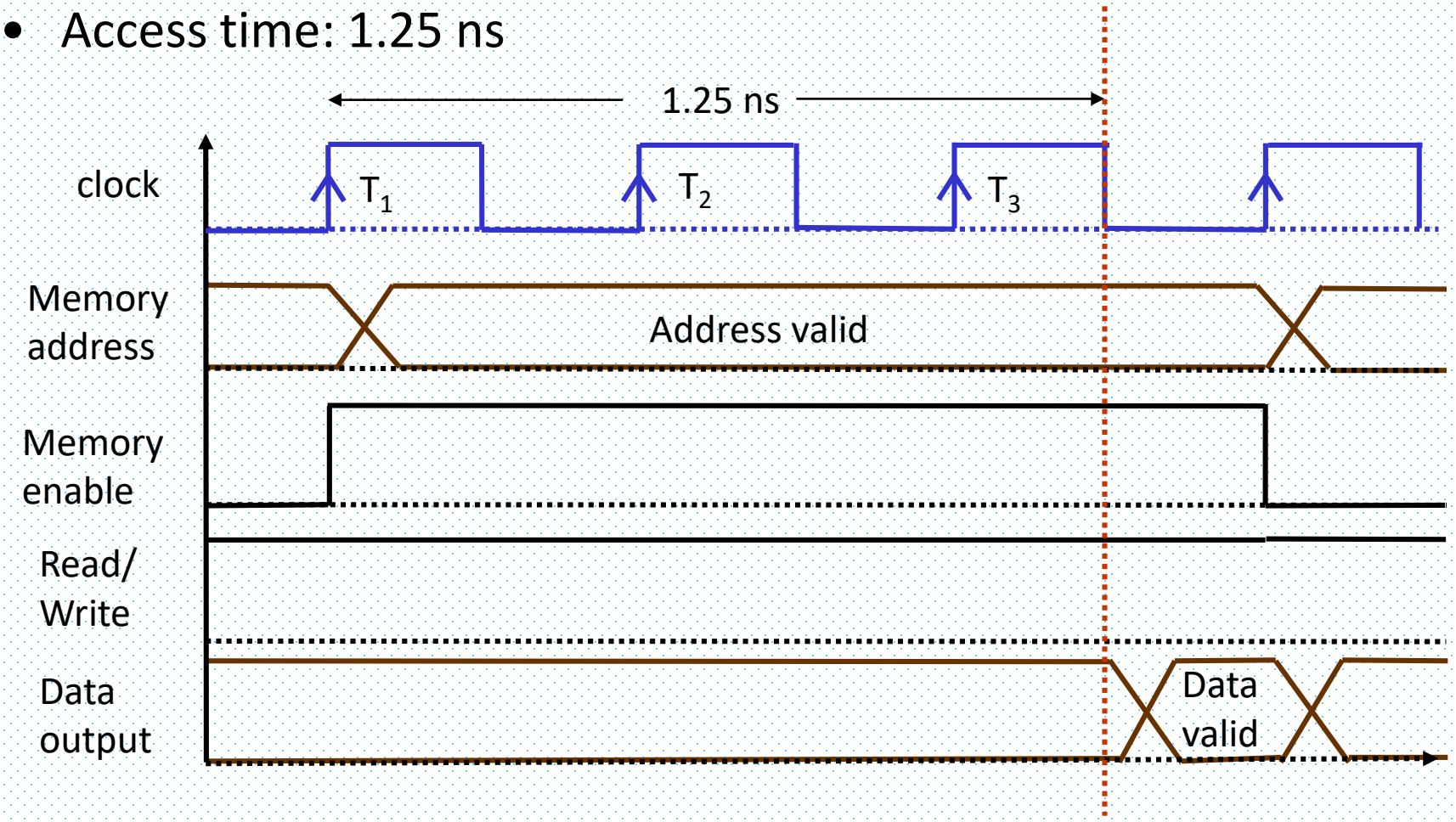
- CPU clock: 2 GHz
- Cycle time: 1.25 ns



Read Cycle

The desired word appears at output, 1.25 ns after memory enable is activated

- CPU clock: 2 GHz
- Access time: 1.25 ns



Types of Memory 1/2

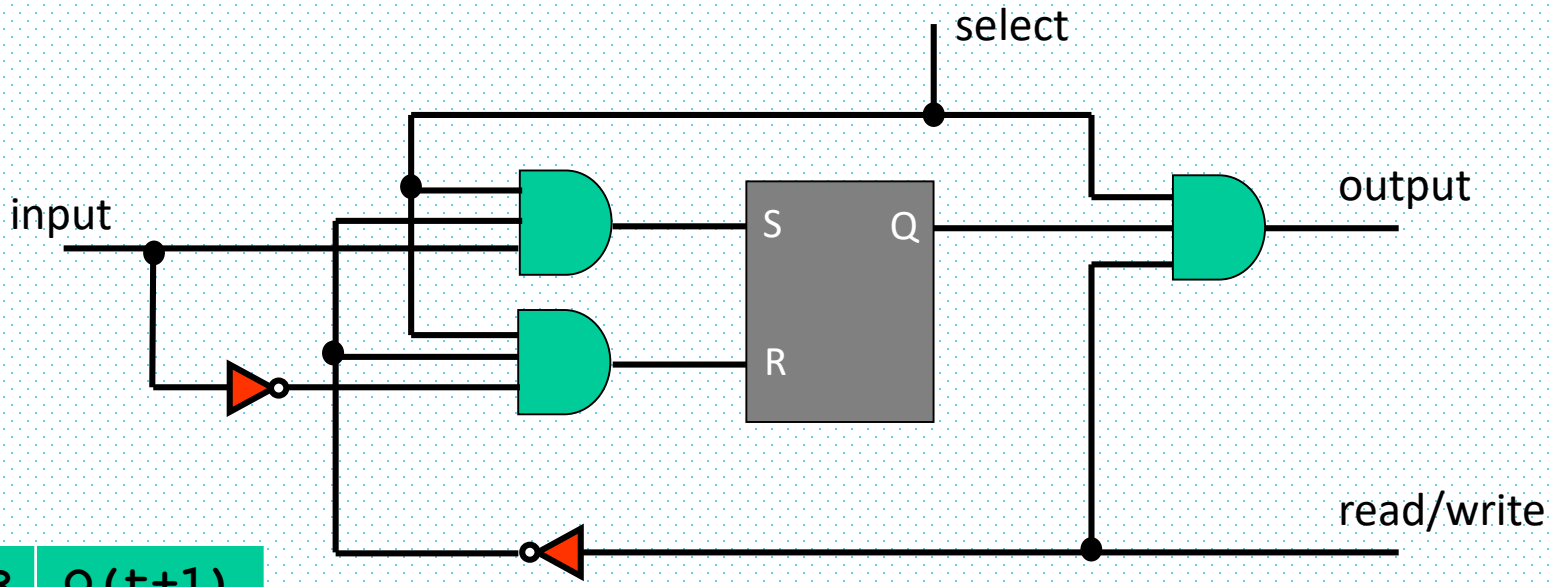
- RAM
  - access time is always the same no matter where the desired data is actually located
- Sequential-access memory
  - Access time is variable
  - e.g., magnetic disks, tapes
- RAM
  - SRAM (static RAM)
    - latches, stores information as long as power is on
  - DRAM (dynamic RAM)
    - information is stored as charge on a capacitor
    - refreshing is necessary due to discharge

Types of Memory 2/2

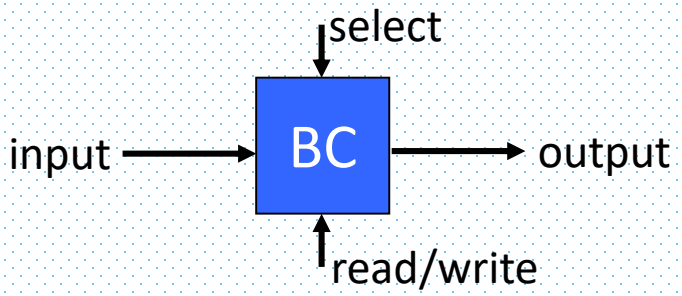
- Volatile memory
  - When the power is turned off, the stored information is lost
  - RAM (SRAM or DRAM)
- Nonvolatile memory
  - retains the stored information even after removal of power
  - magnetic disks
    - data is represented as the direction of magnetization
  - ROM
    - programs needed to start a computer are kept in ROM

Memory Cell

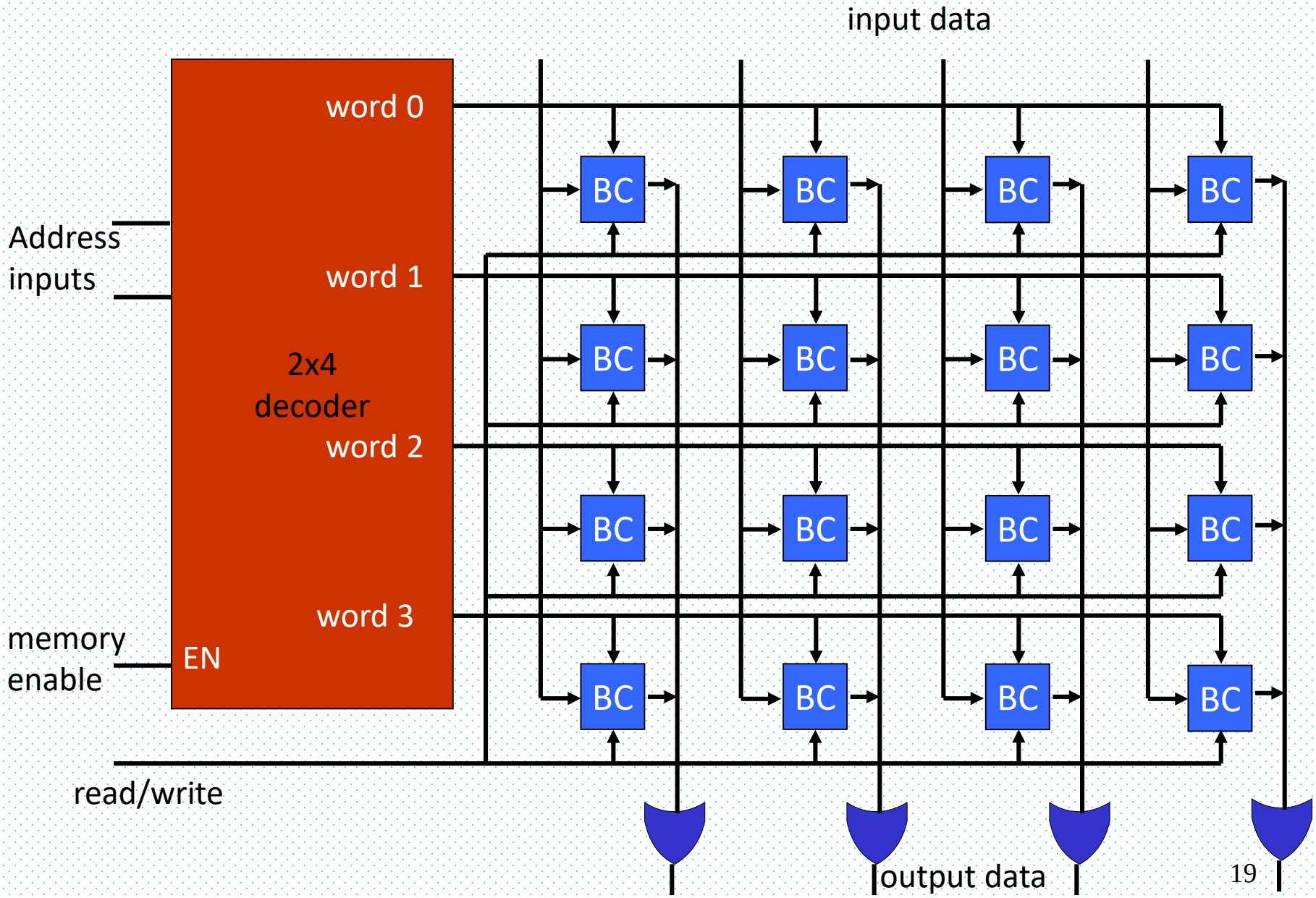
- Equivalent logic of a memory cell for storing one bit of information



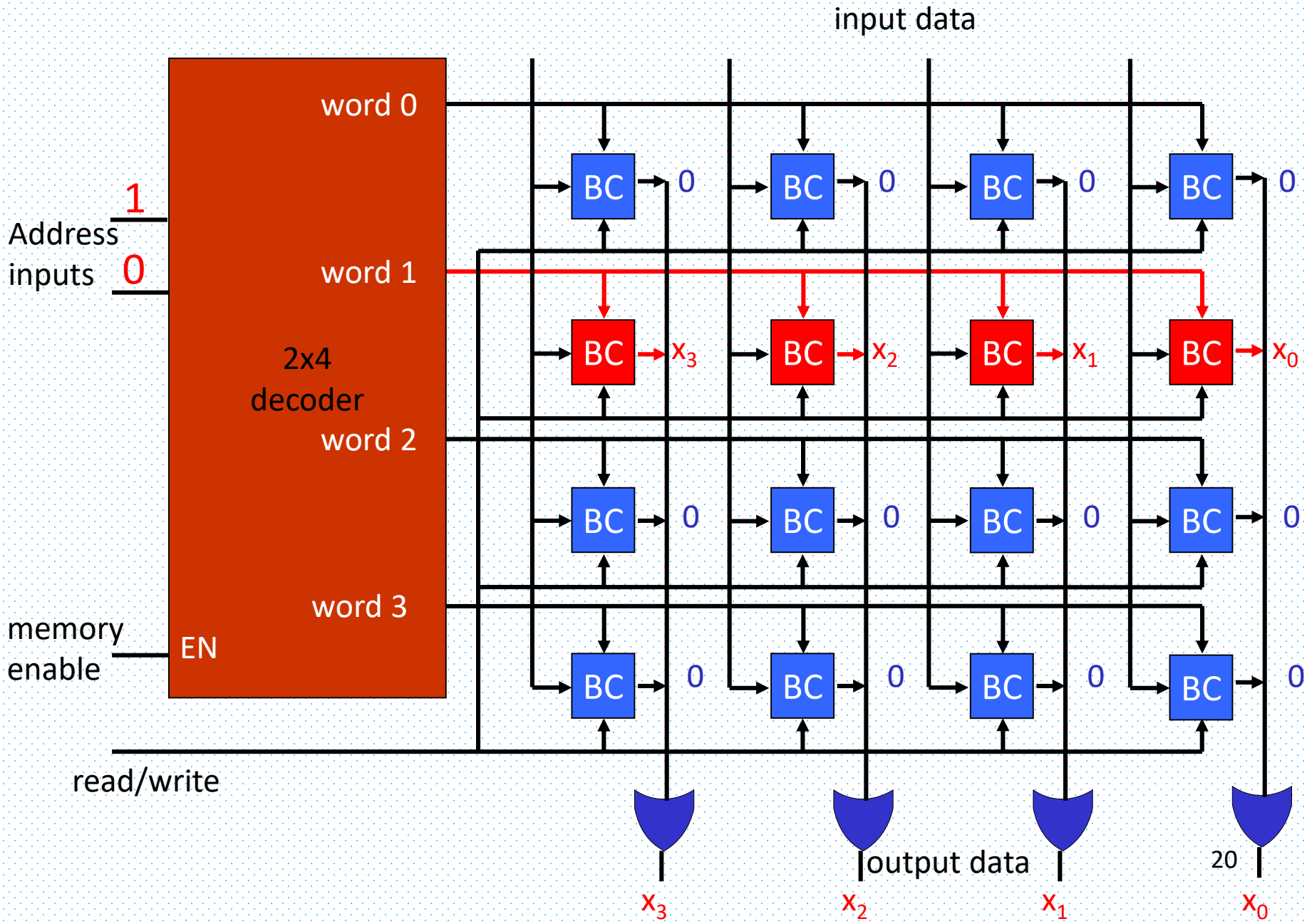
S	R	Q(t+1)
0	0	Q
0	1	0
1	0	1
1	1	X



4 x 4 RAM

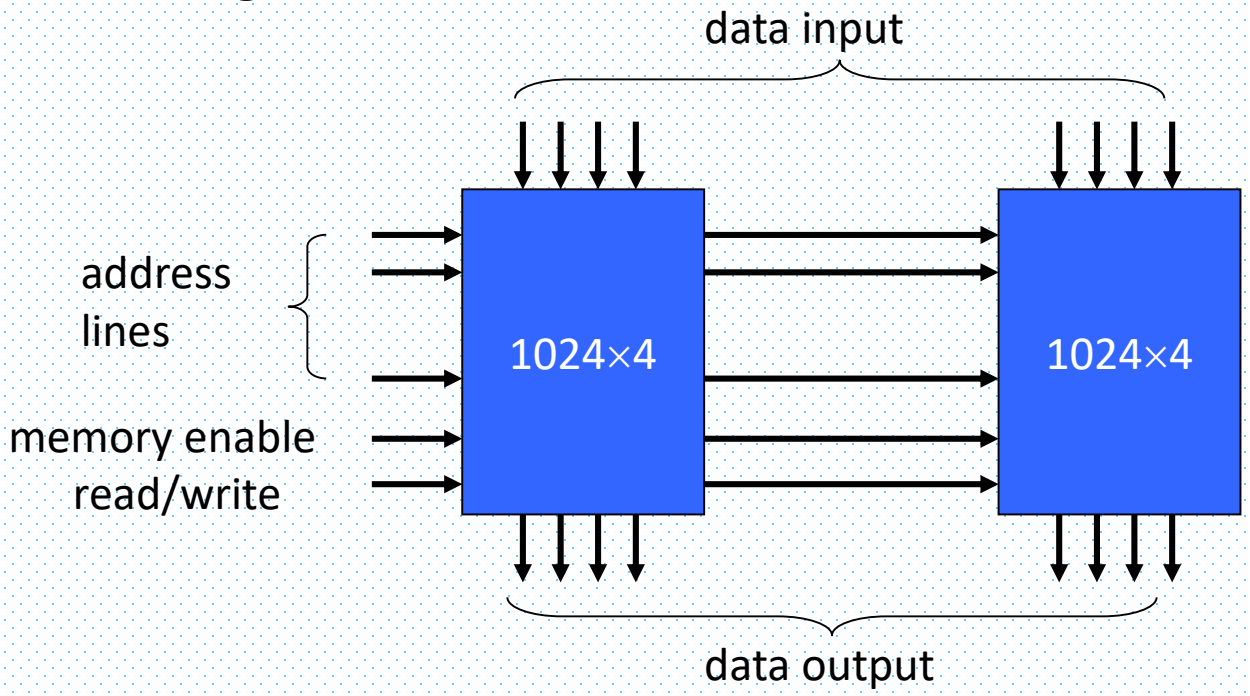

input data
word 0
word 1
word 2
word 3
2x4
decoder
Address
inputs
memory
enable
EN
read/write
BC
output data

4 x 4 RAM

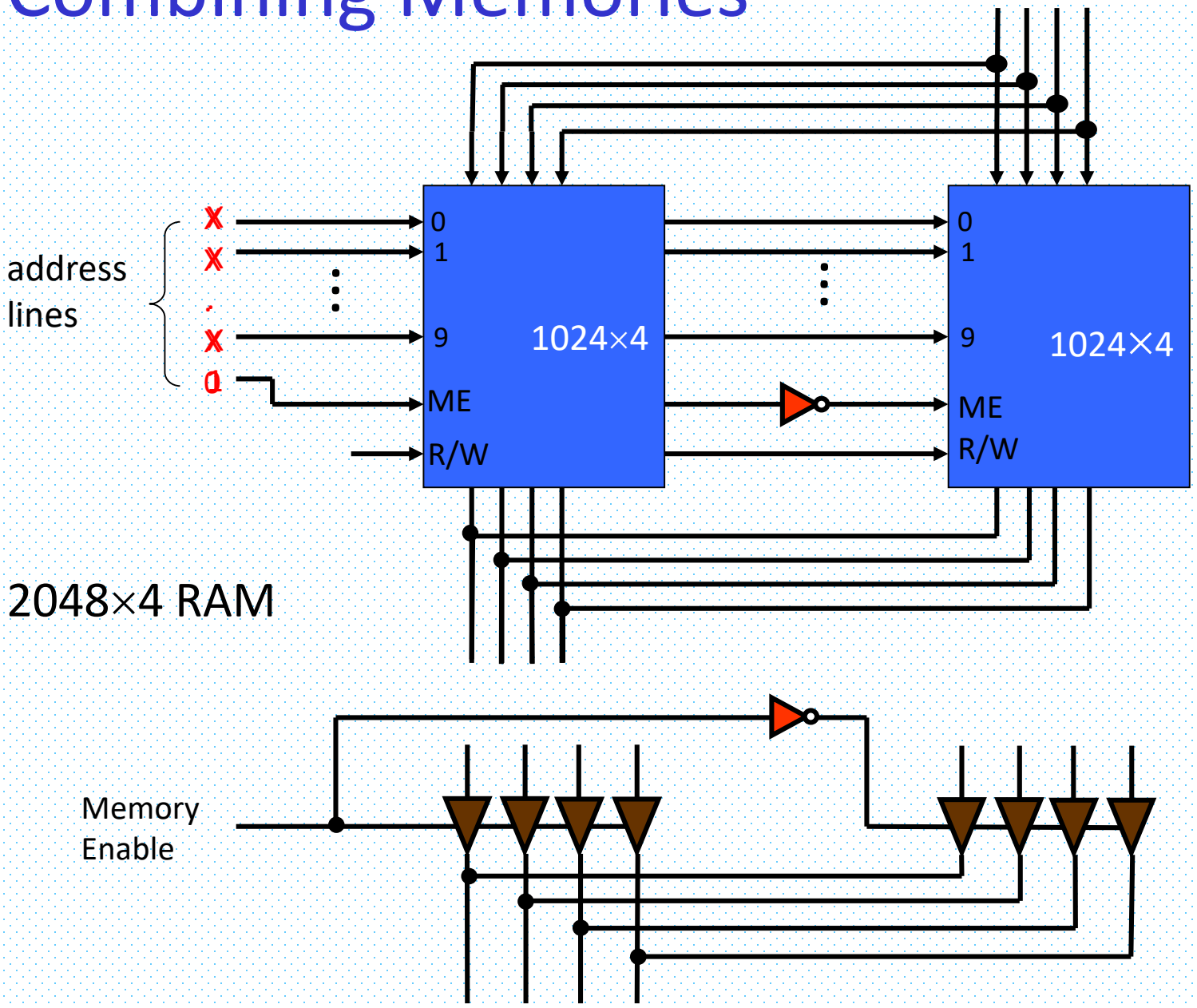

input data
word 0
word 1
word 2
word 3
2x4
decoder
1
Address
inputs
0
memory
enable
EN
read/write
BC
0
x_3
x_2
x_1
x_0
output data
20

Commercial RAMs

- Physical construction
  - Capacity of thousands of words
  - each word may range from 1 to 64 bits
- Example:
  - We have memory chips of 1024×4
  - Logical construction: 1024×8



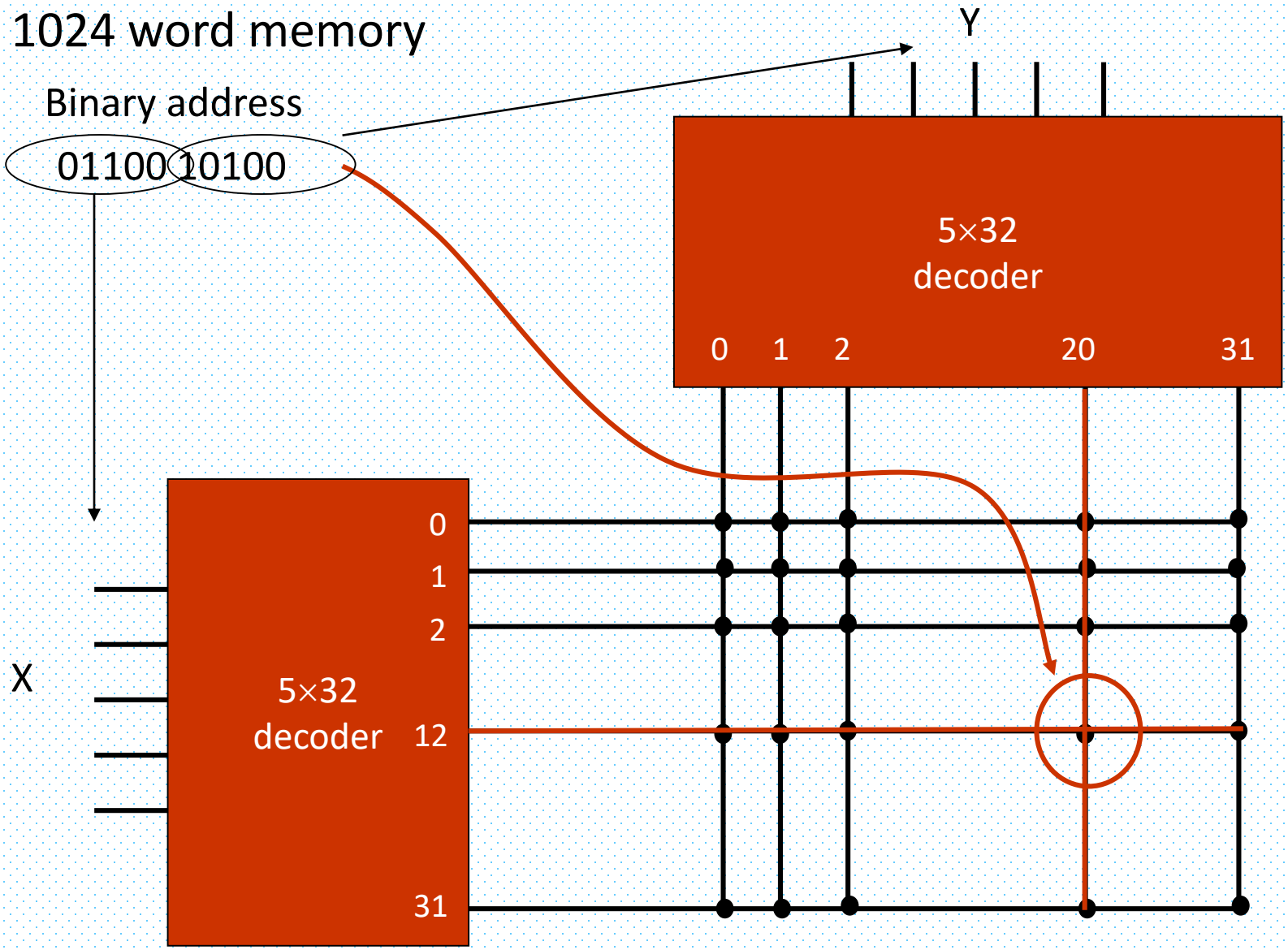
Combining Memories



Coincident Decoding

- A memory with 2^k words requires a $k \times 2^k$ decoder
- $k \times 2^k$ decoder requires 2^k AND gates with k inputs per gate
- There are ways to reduce the total number of gates and number of inputs per gate
- Two dimensional selection scheme
  - Arrange the memory words in a two-dimensional array that is as close as possible to square
  - Use two k/2-input decoders instead of one k-input decoder.
    - One decoder performs the row selection
    - The other does the column selection

Example: Coincident Decoding

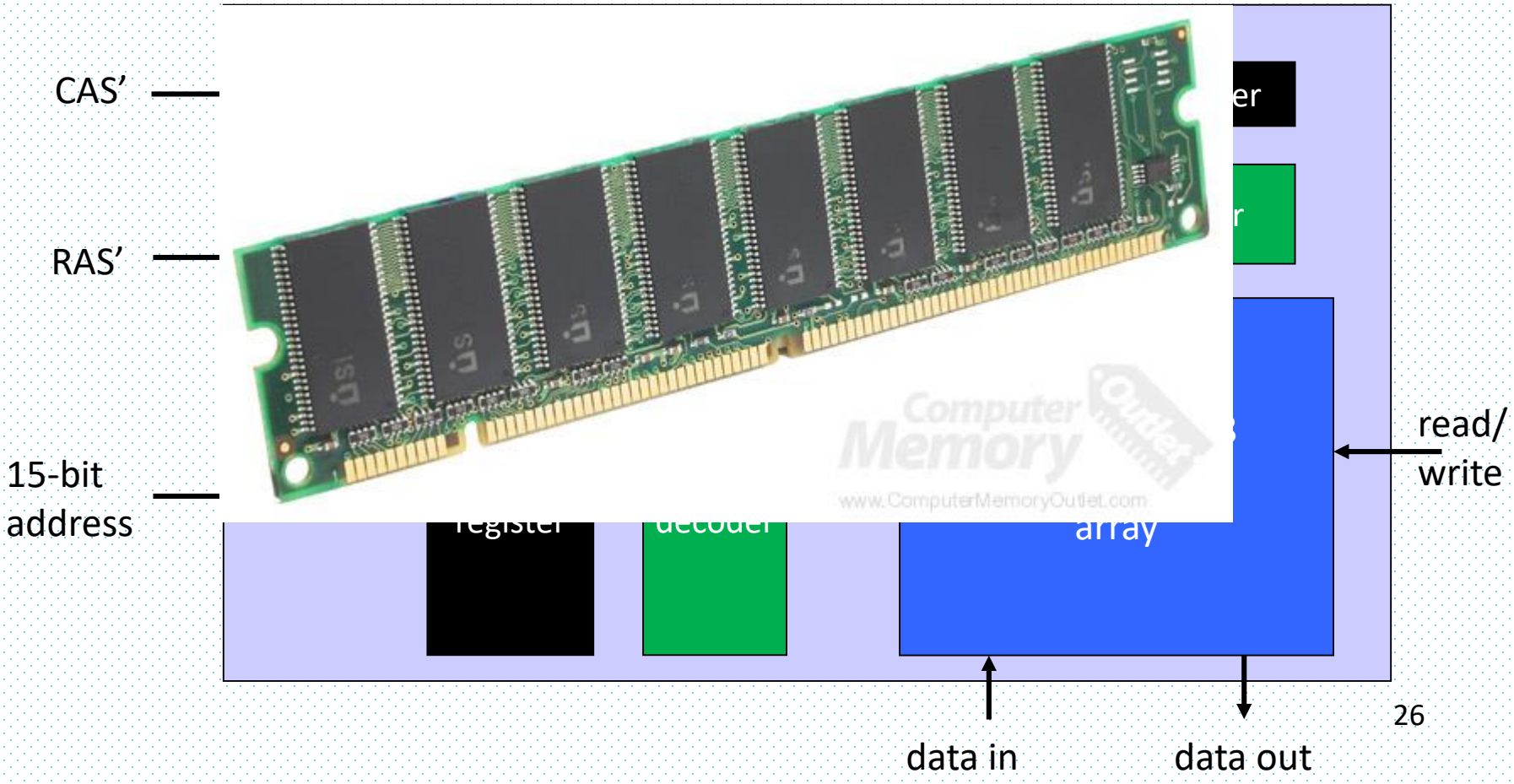


DRAMs

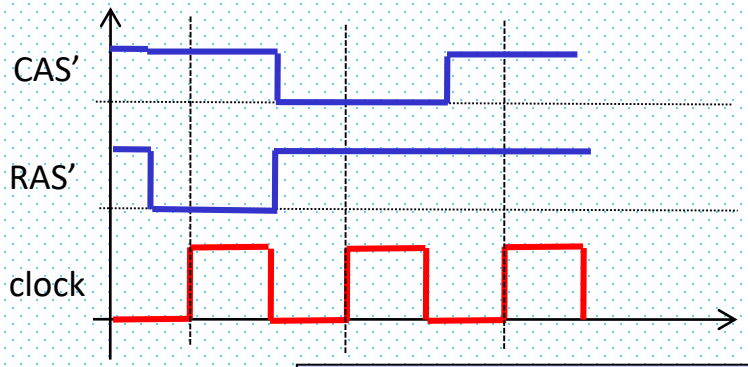
- SRAM memory is expensive
  - One cell typically contains four to six transistors
  - Usually used for on-chip cache memories and embedded systems (cameras, smart phones, etc.)
- DRAM is much less expensive
  - One MOS transistor and a capacitor
  - Four times the density of SRAM in a given chip area
  - cost per bit storage is three to four times less than SRAM
  - low power requirement
  - Perfect technology for large memories such as main memory
  - Most DRAMs have short word sizes

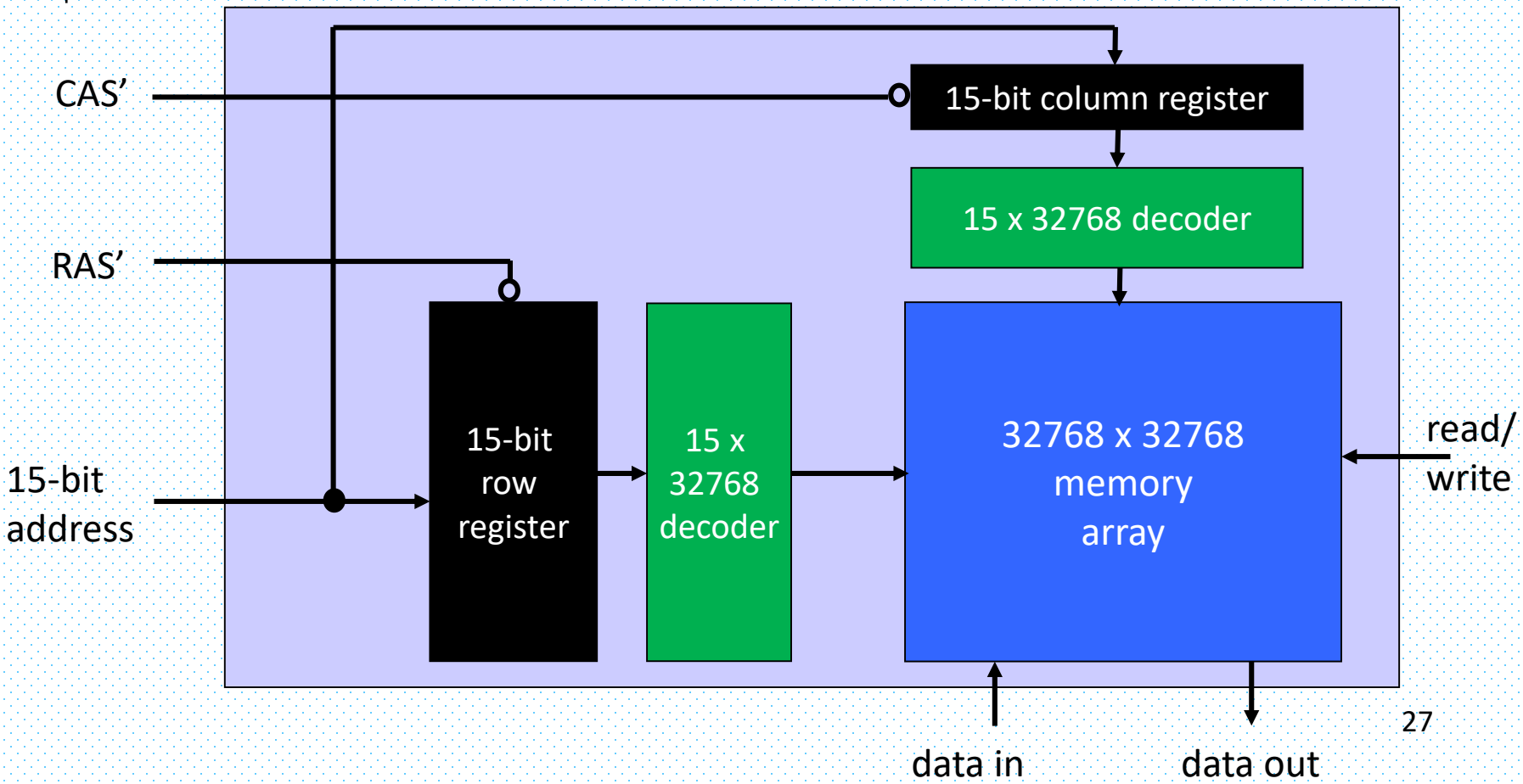
DRAMs and Address Multiplexing

- In order to reduce number of pins on a memory chip, the same pins are used for both row and column addresses
- Example: 1 GB DRAM



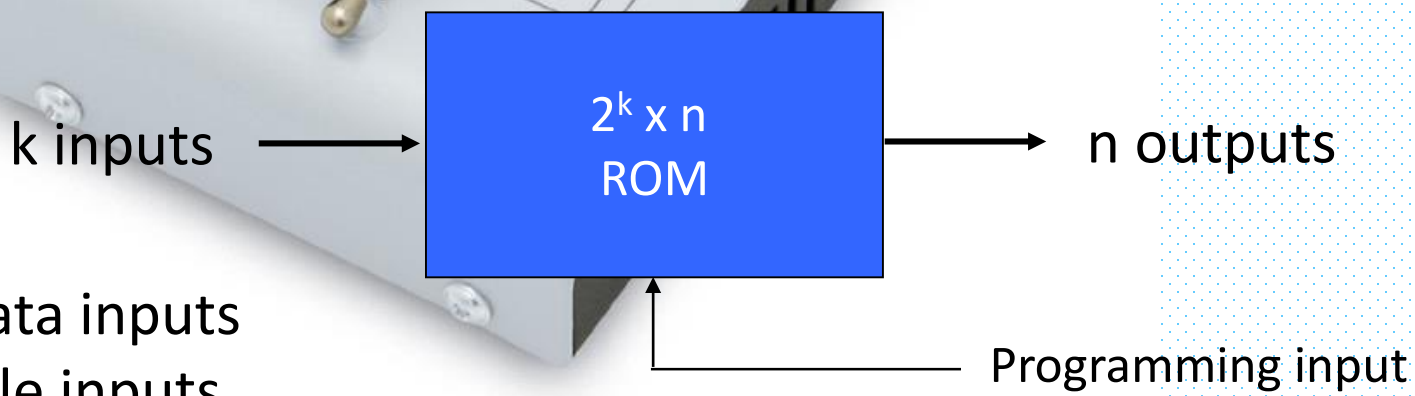
DRAMs and Address Multiplexing





Read-Only Memory

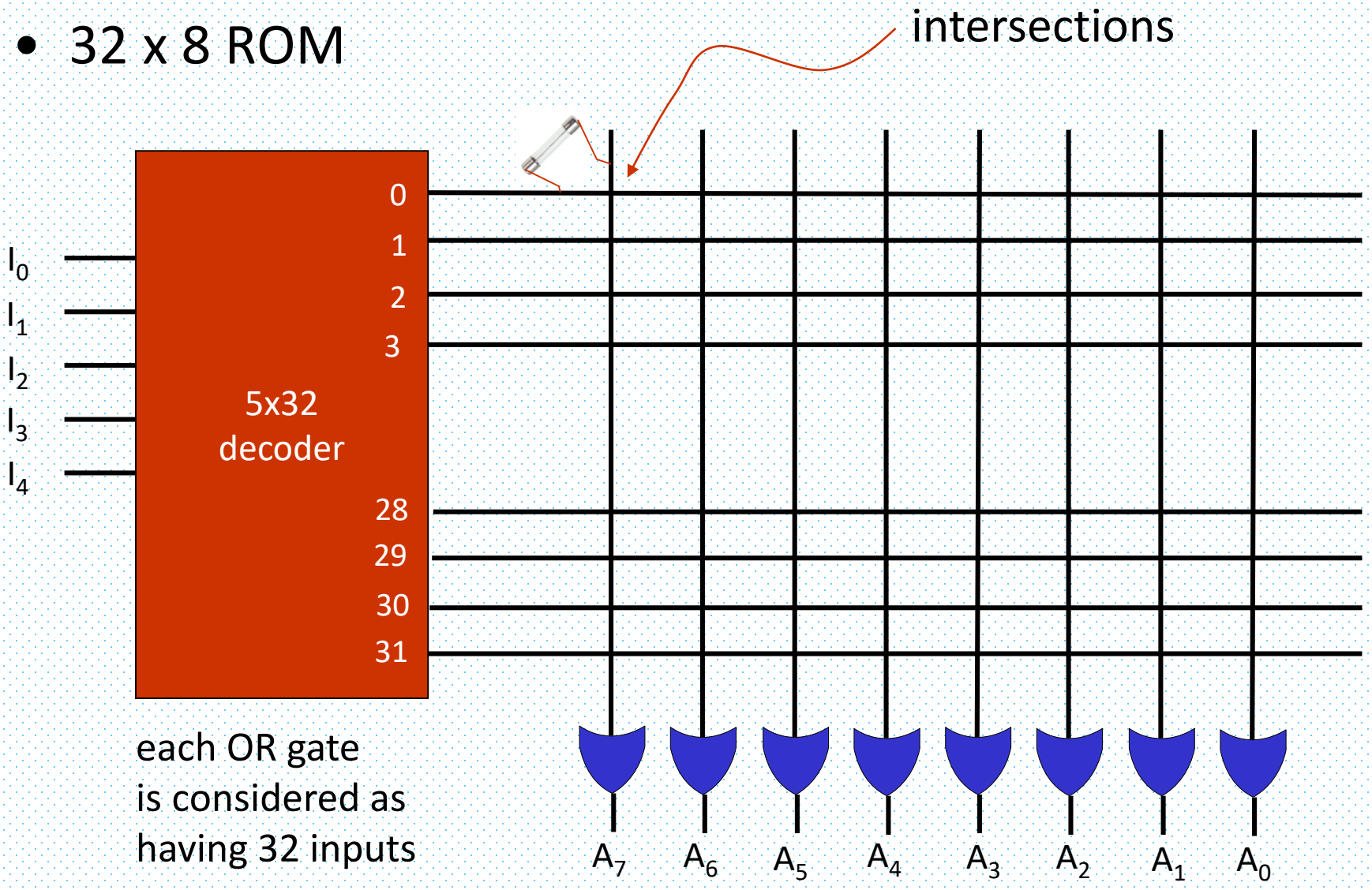
- ROM
  - memory device in which permanent binary information is stored
  - Binary information is programmed into the device
  - It then is embedded in the unit to form the required interconnection pattern
  - nonvolatile
- Block diagram



  - no data inputs
  - enable inputs
  - three-state outputs

Example: ROM

- 32 x 8 ROM



each OR gate
is considered as
having 32 inputs

Example: ROM

- Number of connections
  - 32×8 ROM has $32 \times 8 = 256$ internal connections
- In general
  - $2^k \times n$ ROM will have a $k \times 2^k$ decoder and n OR gates
  - Each OR gate has 2^k inputs
  - inputs of every OR gate are initially connected to each output of the decoder
- These intersections are programmable
  - they are initially closed (connected to the input of OR gate)
  - A fuse is used to connect two wires
  - During programming, some of these fuses are blown by applying high voltage.

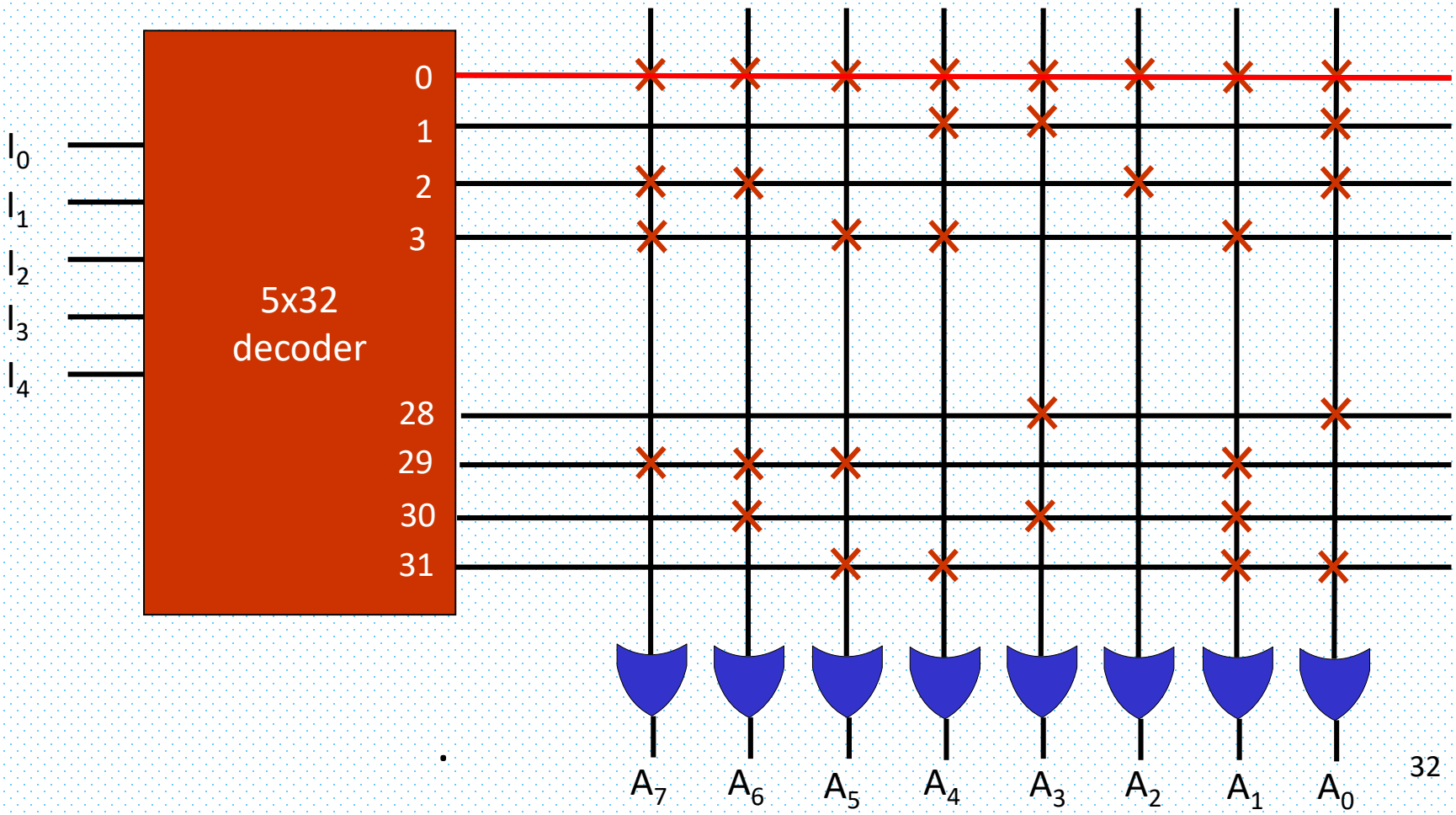
Programming ROM

- Internal storage specified by a table
- Example: 32 × 8 ROM

Inputs					Outputs							
I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
…	…	…	…	…	…	…	…	…	…	…	…	…
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

Programming ROM

Inputs					Outputs							
I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	1	0	1	1	0	1	1	0

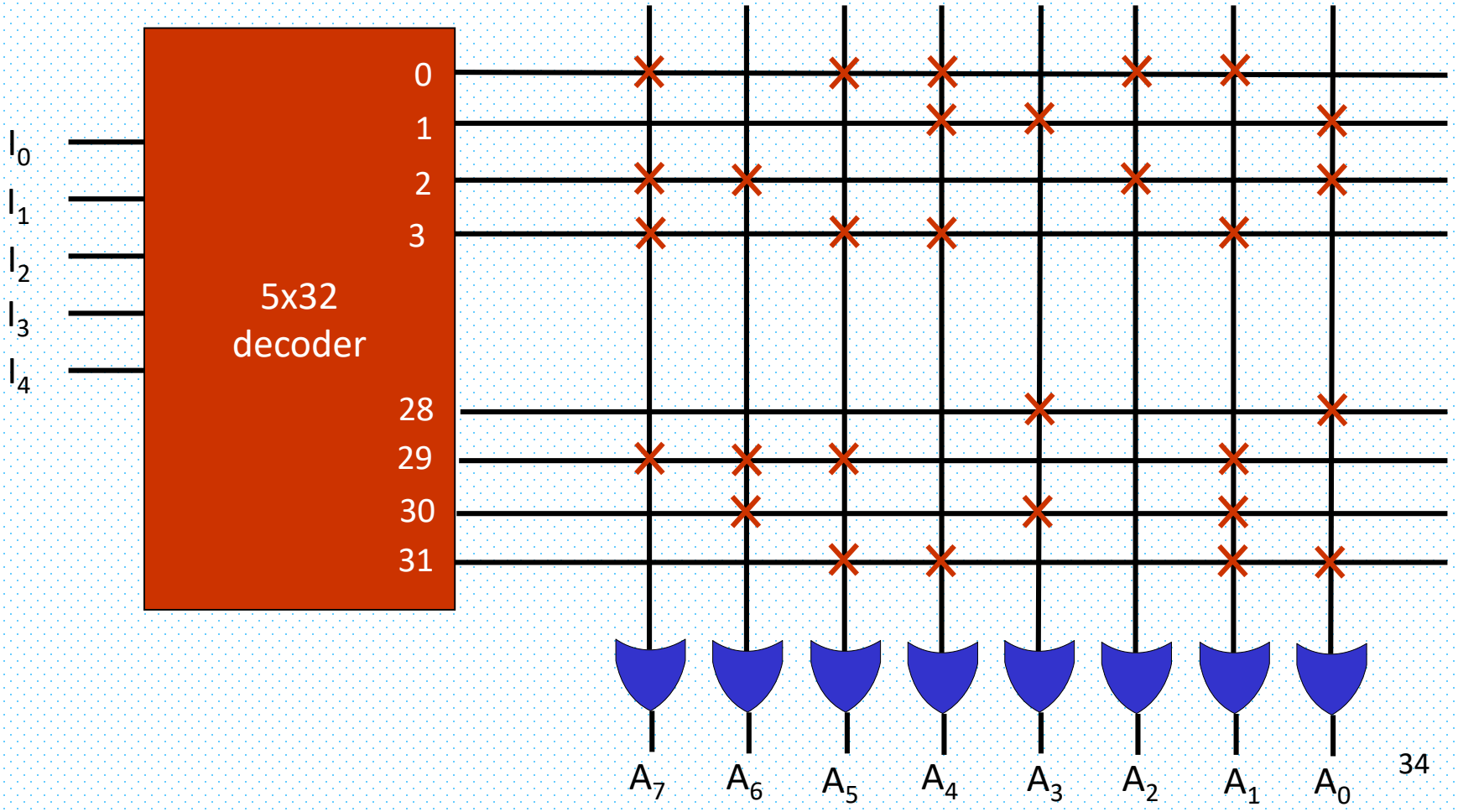


Combinational Circuit Design with ROM

- Formerly,
  - we have shown that a $k \times 2^k$ decoder generates 2^k minterms of k input variables
- Furthermore,
  - by inserting OR gates to sum these minterms, we were able to realize any desired combinational circuit.
- A ROM is essentially a device that includes both the decoder and the OR gates within a single device.
  - first interpretation: a memory unit that stores words

Combinational Circuit Design with ROM

- ROM (cont.)
  - Second interpretation: a programmable device that can realize any combinational circuit



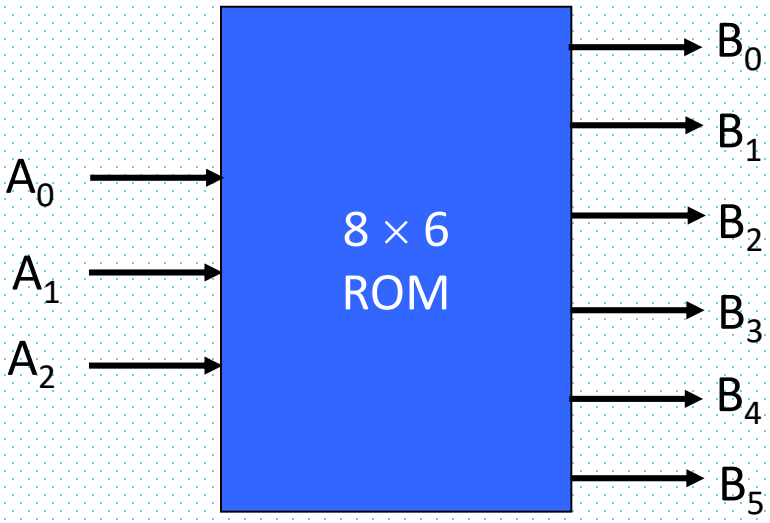
Combinational Circuit Design with ROM

- Example: Truth table

Inputs			Outputs					
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	1	0
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	1	1	1
1	1	1	1	1	0	0	0	1

Example: Design with ROM

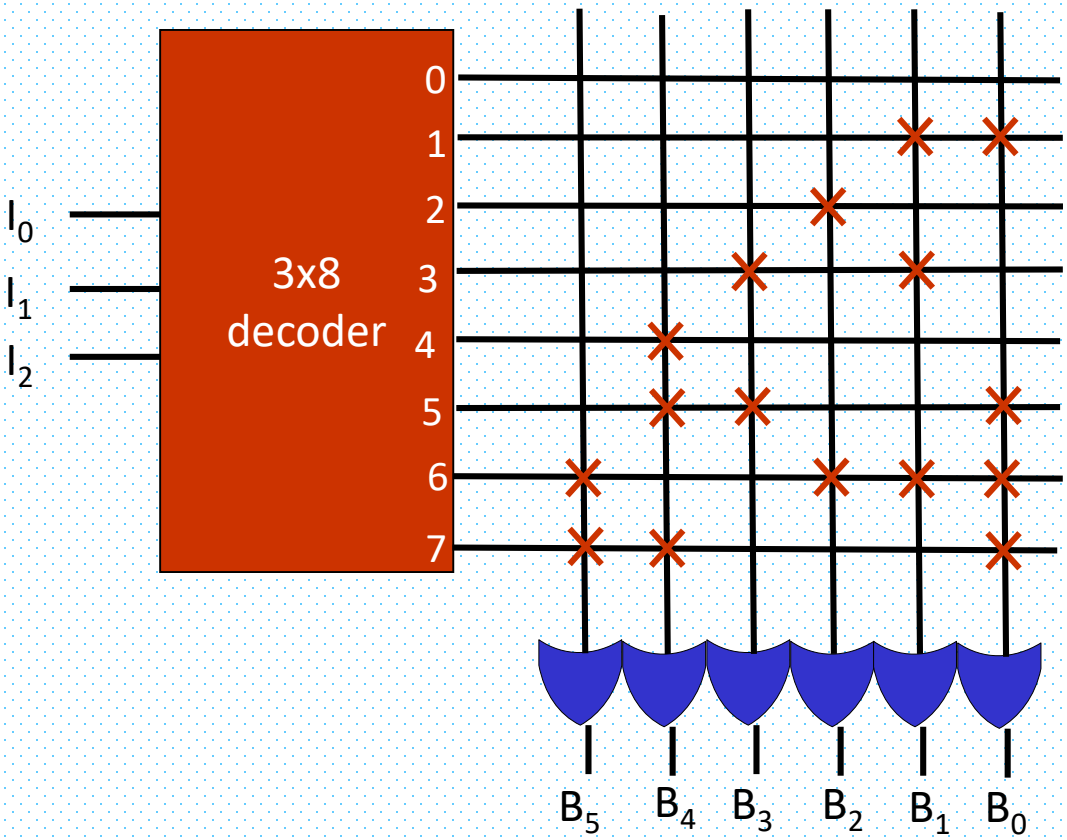
8 x 6 ROM would suffice



Inputs			Outputs					
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	1	0
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	1	1	1
1	1	1	1	1	0	0	0	1

ROM Truth Table

Example: Design with ROM



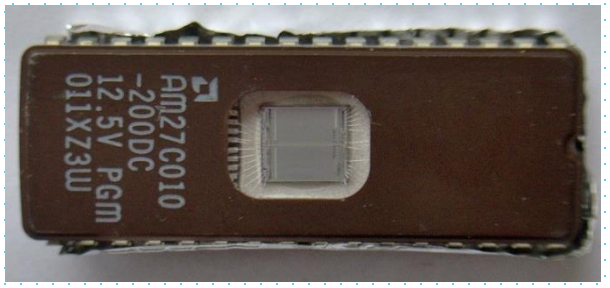
Inputs			Outputs					
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	1	0
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	1	1	1
1	1	1	1	1	0	0	0	1

Types of ROM 1/2

- Programming can be done in different ways
  - Mask programming:
    - customer provides the truth table
    - manufacturer generates the mask for the truth table
    - can be costly, since generating a custom mask is charged to the customer.
    - economical only if a large quantity of the same ROM configuration is to be ordered.
  - Field Programmable
    - Programmable ROM (PROM):
    - Customer can program the ROM by blowing fuses by applying high voltage through a special pin
    - Special instrument called PROM programmer is needed.

Types of ROM 2/2

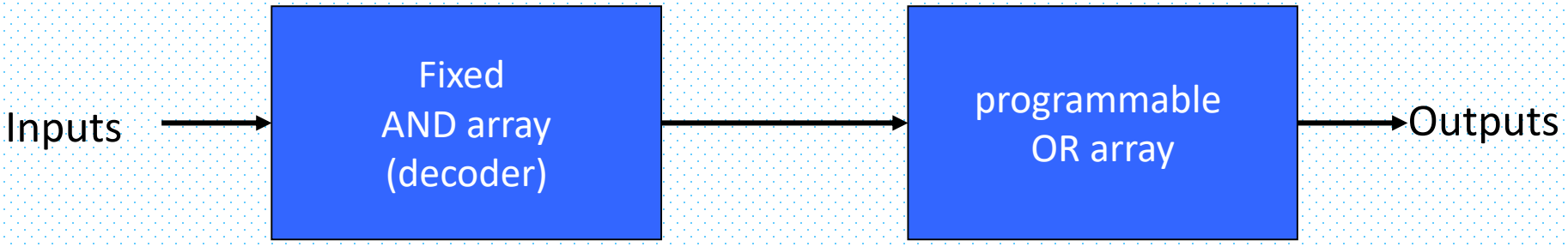
- Programming ROM and PROMs is irreversible.
- Erasable PROM (EPROM)
  - can be programmed repeatedly.
  - EPROM is placed under a special ultra-violet light for a given period of time
  - At the end, the former program is erased
  - After erasure, EPROM becomes ready for another programming



- Electronically erasable PROM (EEPROM or E^2PROM)

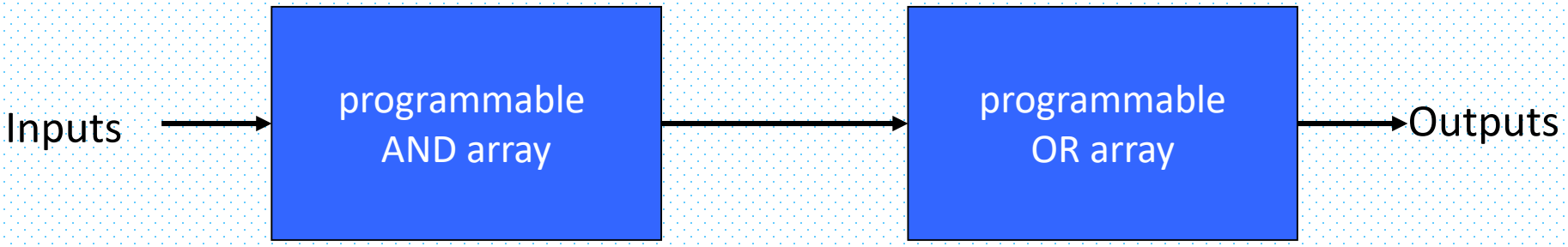
Programmable Logic Devices

- EPROM is an example of combinational **p**rogrammable **l**ogic **d**evice (PLD)
- Configuration of EPROM

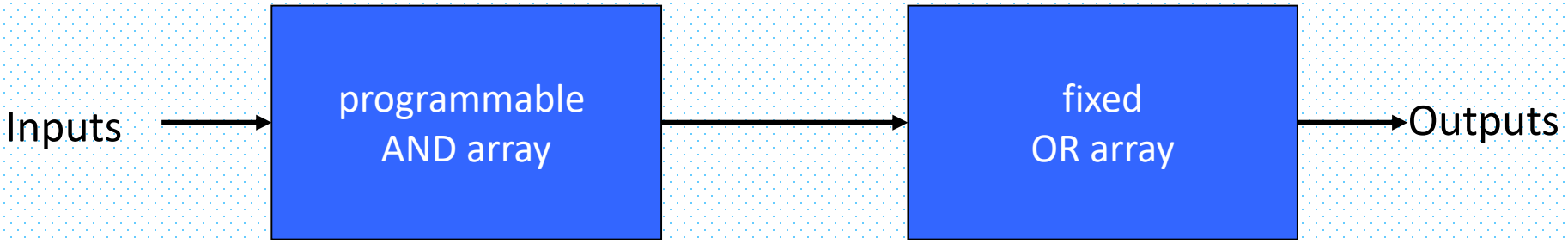


Other PLDs

- Two other types

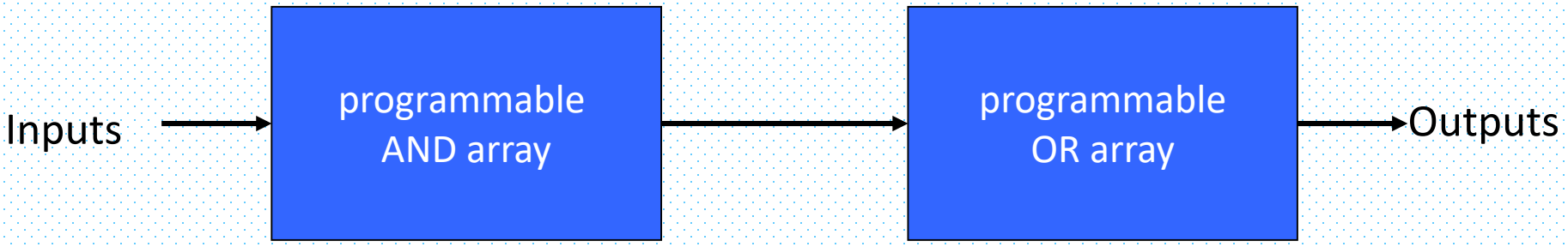


Programmable Logic Array (PLA)



Programmable Array Logic (PAL)

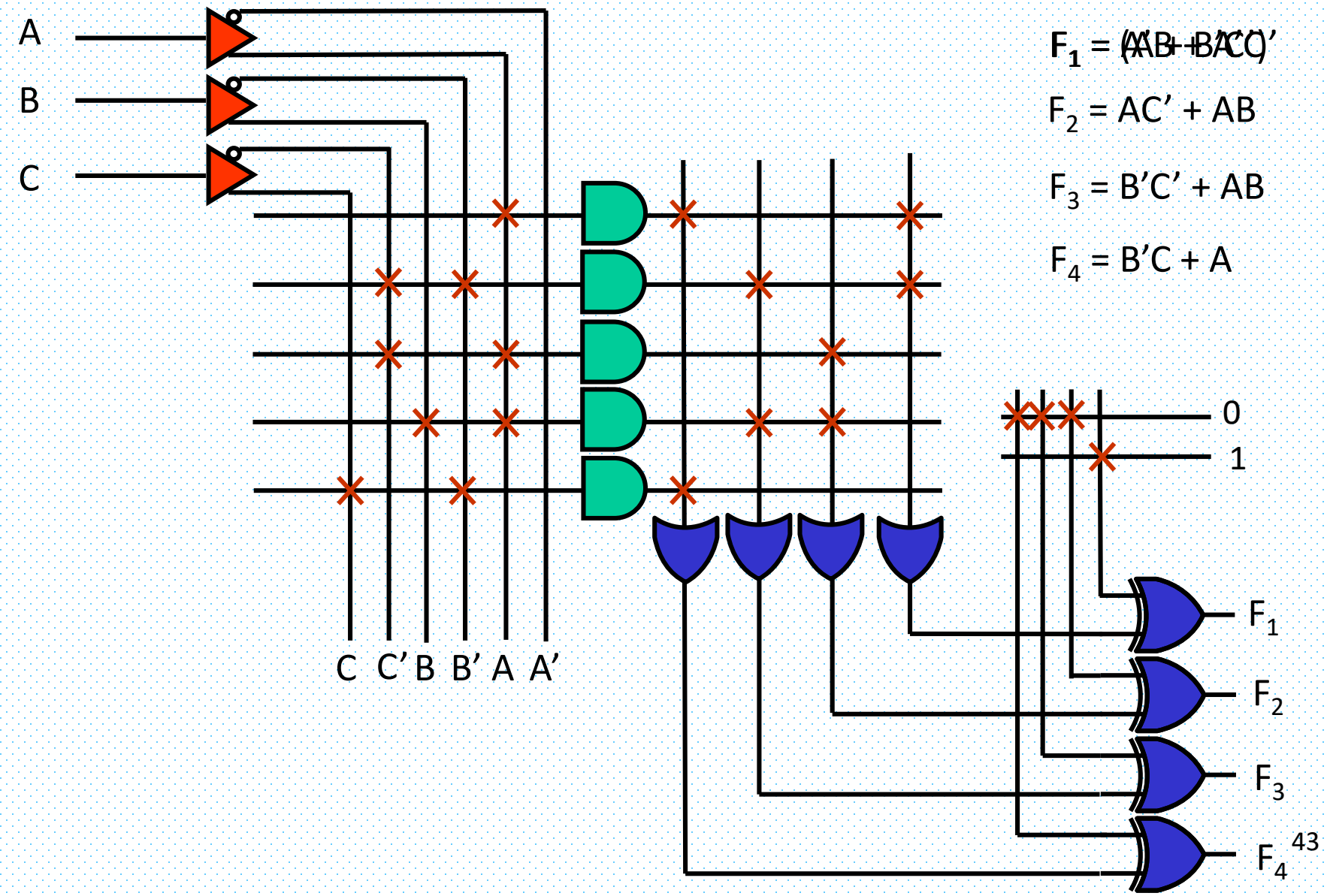
Programmable Logic Array (PLA)



- Similar to PROM
- However, PLA does not generate all the minterms
- Decoder is replaced by an array of AND gates
  - can be programmed to generate any product term of input variables
- The product terms are then connected to OR gates
  - that provide the sum of products

PLA: Example

- PLA: 3 inputs, 5 product terms and 4 outputs



$F_1 = (A'B + B'C)'$

$F_2 = AC' + AB$

$F_3 = B'C' + AB$

$F_4 = B'C + A$

PLA Programming Table

$F_1 = (A + B'C')'$ $F_3 = B'C' + AB$

$F_2 = AC' + AB$ $F_4 = B'C + A$

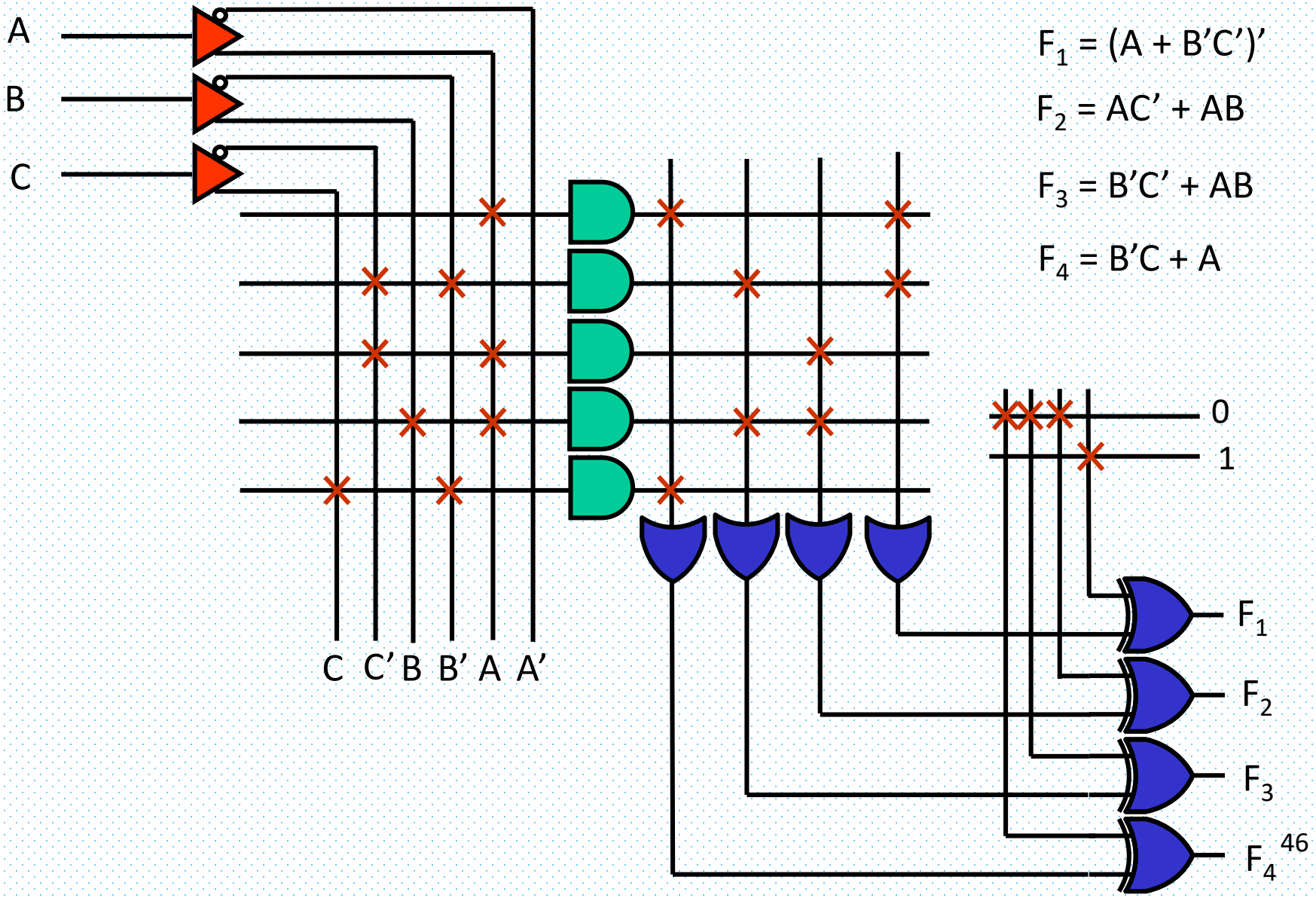
					Outputs			
		Inputs						
	Product Term	A	B	C	F_1	F_2	F_3	F_4
	1							
	2							
	3							
	4							
	5							

Size of PLA

- Specified by
  - number of inputs, number of product terms, number of outputs
- A typical IC PLA (F100)
  - 16 inputs, 48 product terms, and 8 outputs
- n input, k product terms, m output PLA has
  - k AND gates, m OR gates, m XOR gates
  - $2n \times k$ connections between input and the AND array
  - $k \times m$ connections between the AND and OR arrays
  - 2m connections associated with XOR gates
  - $(2n \times k + k \times m + 2m)$ connections to program

PLA: Example

- PLA: 3 inputs, 5 product terms and 4 outputs



$F_1 = (A + B'C')'$

$F_2 = AC' + AB$

$F_3 = B'C' + AB$

$F_4 = B'C + A$

Programming PLA

- Optimization
  - number of literals in a product term is not important
  - When implementing more than one function, functions must be optimized together in order to share more product terms
    - multiple output optimization (espresso)
  - both the true and complement of each function should be simplified to see which one requires fewer number of product terms

http://web.eecs.umich.edu/~ksewell/espresso/

Example: Programming PLA

- Two functions
  - $F_1 (A, B, C) = \Sigma(0, 1, 2, 4)$
  - $F_2 (A, B, C) = \Sigma(0, 5, 6, 7)$

A \ BC	00	01	11	10
0	1	1	0	1
1	1	0	0	0

$F_1 = A'B' + A'C' + B'C'$

$F_1 = (AB + AC + BC)'$

A \ BC	00	01	11	10
0	1	0	0	0
1	0	1	1	1

$F_2 = AB + AC + A'B'C'$

$F_2 = (A'C + A'B + AB'C')'$

Example: Programming PLA

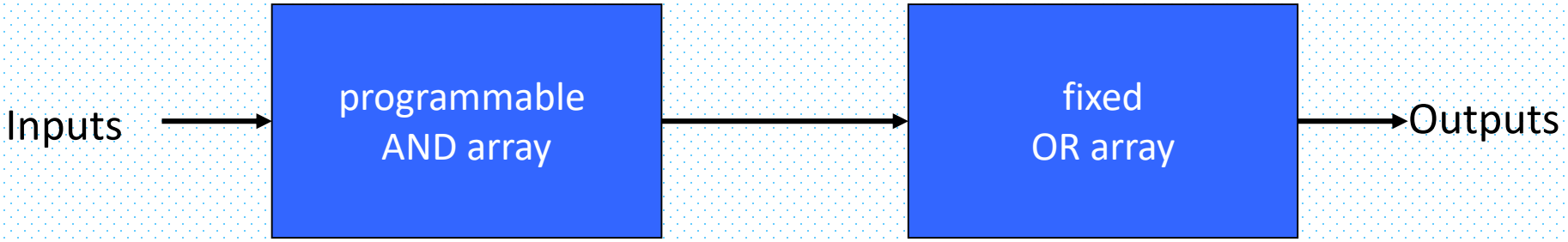
- PLA programming table

$F_1 = (AB + AC + BC)'$

$F_2 = AB + AC + A'B'C'$

					Outputs	
		Inputs			C	T
	Product Term	A	B	C	F_1	F_2
AB	1	1	1	-	1	1
AC	2	1	-	1	1	1
BC	3	-	1	1	1	-
A'B'C'	4	0	0	0	-	1

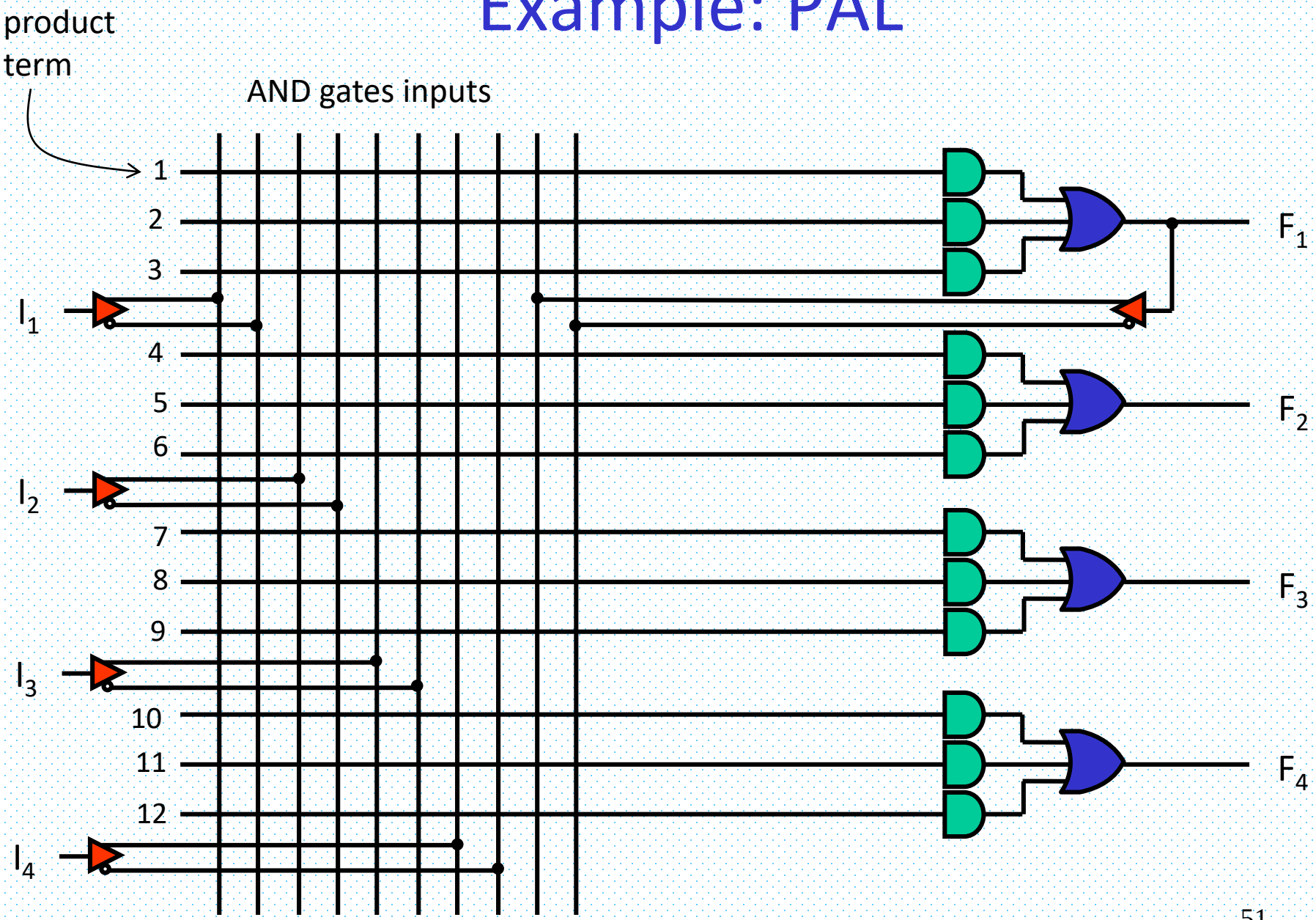
Programmable Array Logic (PAL)



Programmable Array Logic (PAL)

- Easier to program than PLA
- But, not as flexible
- A typical PAL
  - 8 inputs, 8 outputs, 8-wide AND-OR array

Example: PAL


product
term
AND gates inputs
1
2
3
I1
4
5
6
I2
7
8
9
I3
10
11
12
I4
F1
F2
F3
F4

Design with PAL

- Each Boolean function must be simplified to fit into each section.
- Product terms cannot be shared among OR gates
  - Each function can be simplified by itself without regard to common product terms
- The number of product terms in each section is fixed
  - If the number of product terms is too many, we may have to use two sections to implement the function.

Example: Design with PAL

- Four functions
  - $A(x, y, z, t) = \Sigma (2, 12, 13)$
  - $B(x, y, z, t) = \Sigma (7, 8, 9, 10, 11, 12, 13, 14, 15)$
  - $C(x, y, z, t) = \Sigma (0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$
  - $D(x, y, z, t) = \Sigma (1, 2, 8, 12, 13)$
- First step is to simplify four functions separately
  - $A = xyz' + x'y'zt'$
  - $B = x + yzt$
  - $C = x'y + zt + y't'$
  - $D = xyz' + x'y'zt' + xy't' + x'y'z't$

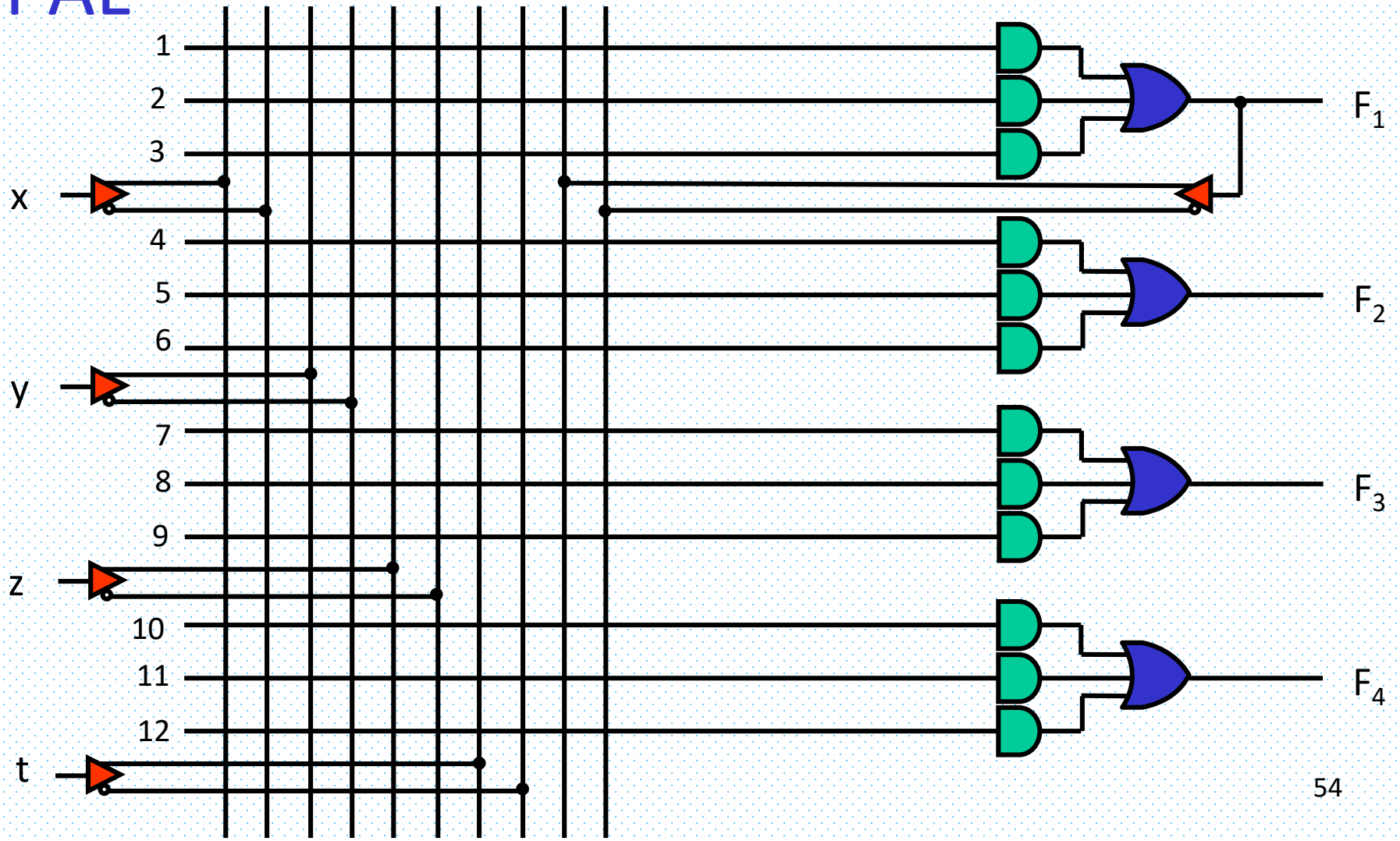
Example: Design with PAL

A = xyz’ + x’y’zt’

B = x + yzt

C = x’y + zt + y’t’

D = xyz’ + x’y’zt’ + xy’t’ + x’y’z’t = A + xy’t’ + x’y’z’t



Example: Design with PAL

– D = A + xy't' + x'y'z't

Product Term	AND Inputs					Outputs
	x	y	z	t	F_1	
1	1	1	0	-	-	A = F_1 = xyz' + x'y'zt'
2	0	0	1	0	-	
3	-	-	-	-	-	
4	1	-	-	-	-	B = F_2 = x + yzt
5	-	1	1	1	-	
6	-	-	-	-	-	
7	0	1	-	-	-	C = F_3 = x'y + zt + y't'
8	-	-	1	1	-	
9	-	0	-	0	-	
10	-	-	-	-	1	D = F_4 = F_1 + xy't' + x'y'z't
11	1	0	-	0	-	
12	0	0	0	1	-	

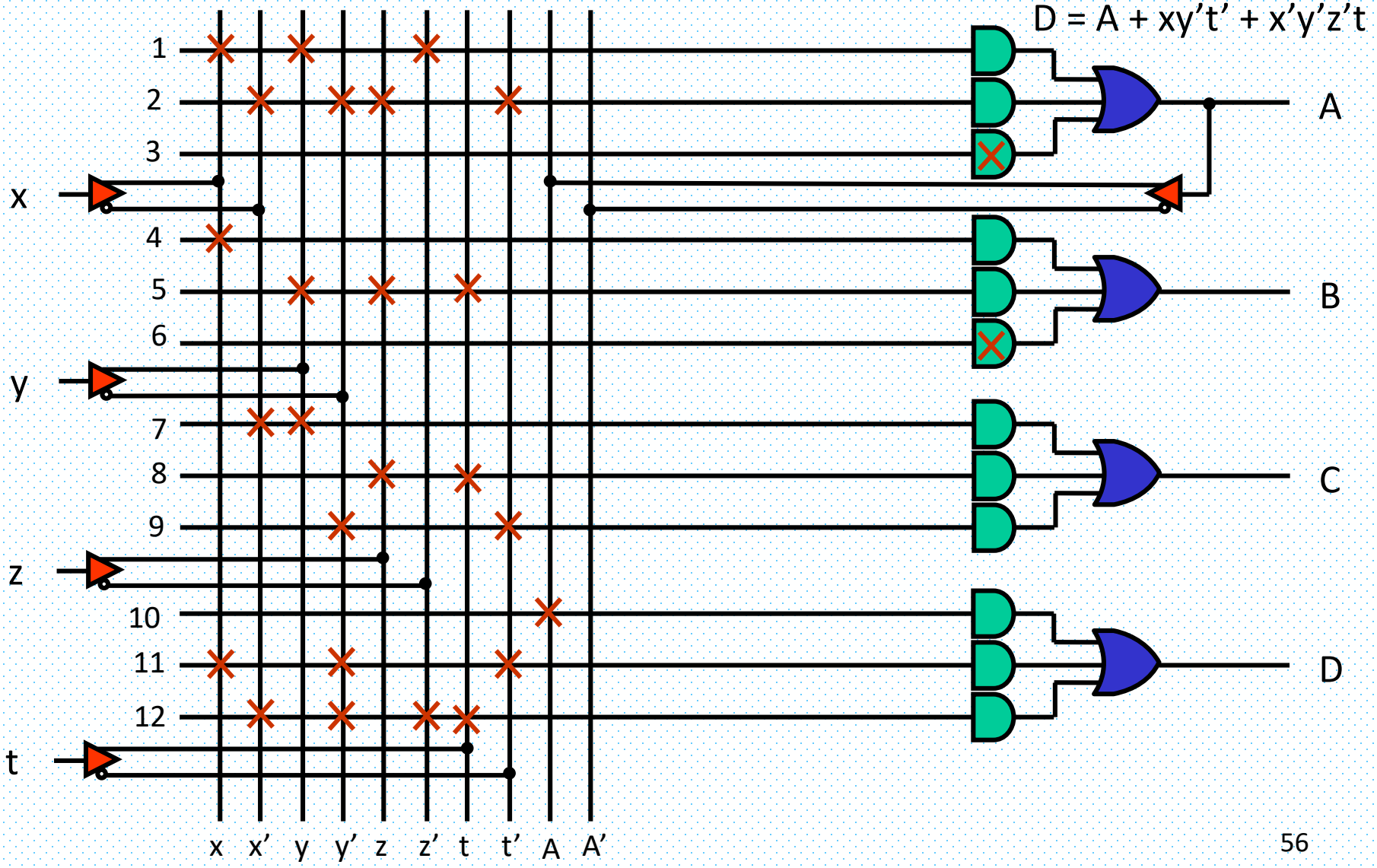
Example: Design with PAL

A = xyz’ + x’y’zt’

B = x + yzt

C = x’y + zt + y’t’

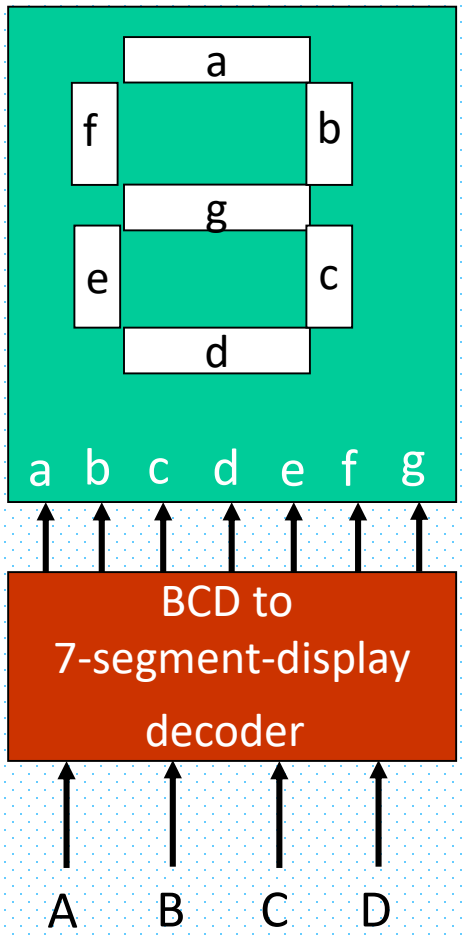
D = A + xy’t’ + x’y’z’t



PAL: BCD to 7-Segment -Display Decoder

- $(ABCD)_{10}$ → (a b c d e f g)

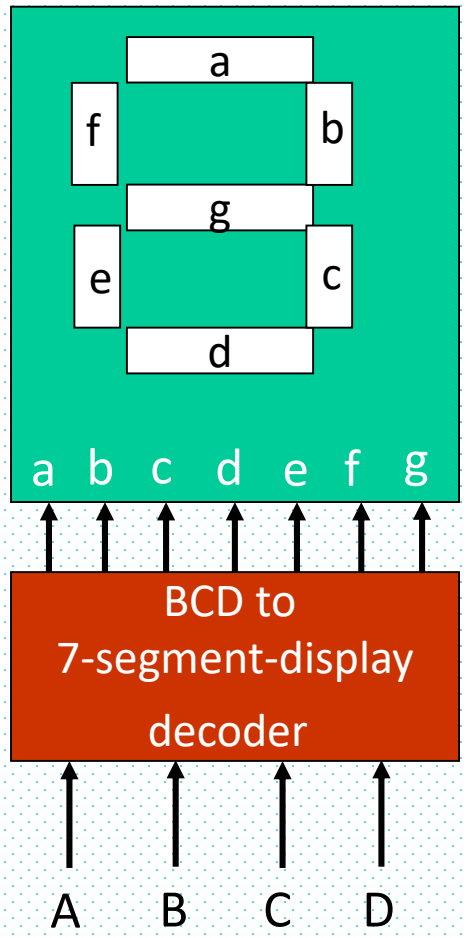
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
X	X	X	X	X	X	X	X	X	X	X



PAL: BCD to 7-Segment -Display Decoder

- $(ABCD)_{10}$ → (a b c d e f g)
  - a = A + BD + C + B'D'
  - b = C'D' + CD + B'
  - c = B + C' + D
  - d = B'D' + CD' + BC'D + B'C + A
  - e = B'D' + CD'
  - f = A + C'D' + BD' + BC'
  - g = A + CD' + BC' + B'C
  - we need 4 inputs, 7 outputs, at most 5 product terms per output
  - P16H8: 10 inputs, 8 outputs, 7 product terms per output.

P14H8: 14 inputs, 8 outputs (2 have four product terms, 6 have 2 product terms)



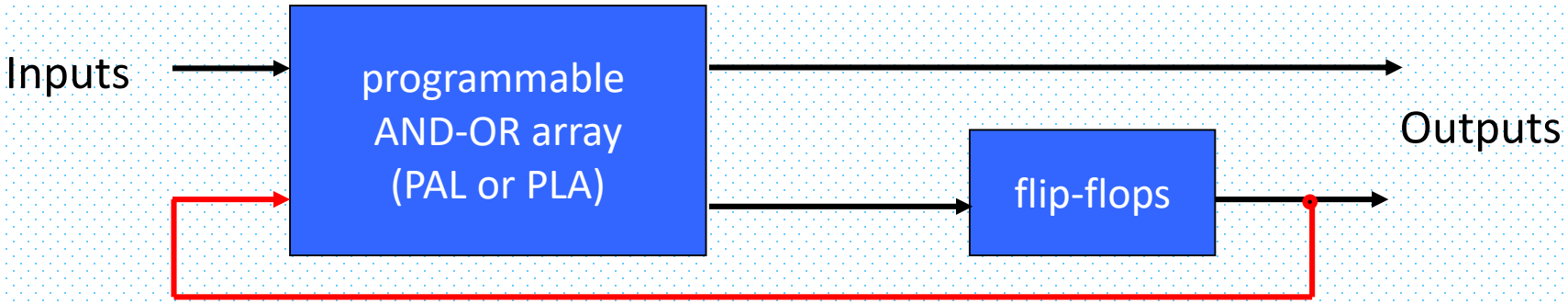
7-Segment-Display Decoder

- Different way to optimize
  - multiple output optimization → espresso supports this
  - a = BC'D + CD + B'D' + A + BCD'
  - b = B'C' + C'D' + CD + B'D'
  - c = B'C' + BC'D + C'D' + CD + BCD'
  - d = B'C + BC'D + B'D' + BCD' + A
  - e = B'D' + BCD'
  - f = BC'D + C'D' + A + BCD'
  - g = BD' + B'C + A + BC'D
  - a = A + BD + C + B'D'
  - b = C'D' + CD + B'
  - c = B + C' + D
  - d = B'D' + CD' + BC'D + B'C + A
  - e = B'D' + CD'
  - f = A + C'D' + BD' + BC'
  - g = A + CD' + BC' + B'C
  - 9 product terms in total (previous one has 15)

Sequential PLDs

- So far, we have seen PLD that can realize only combinational circuits
- However, digital systems are designed using both combinational circuits (gates) and flip-flops.
  - With PLDs, we need to use external flip-flops to realize sequential circuit functions.
- Different types
  1. Sequential (or simple) programmable logic device (SPLD)
  2. Complex programmable logic device (CPLD)
  3. Field programmable gate array (FPGA)

SPLD

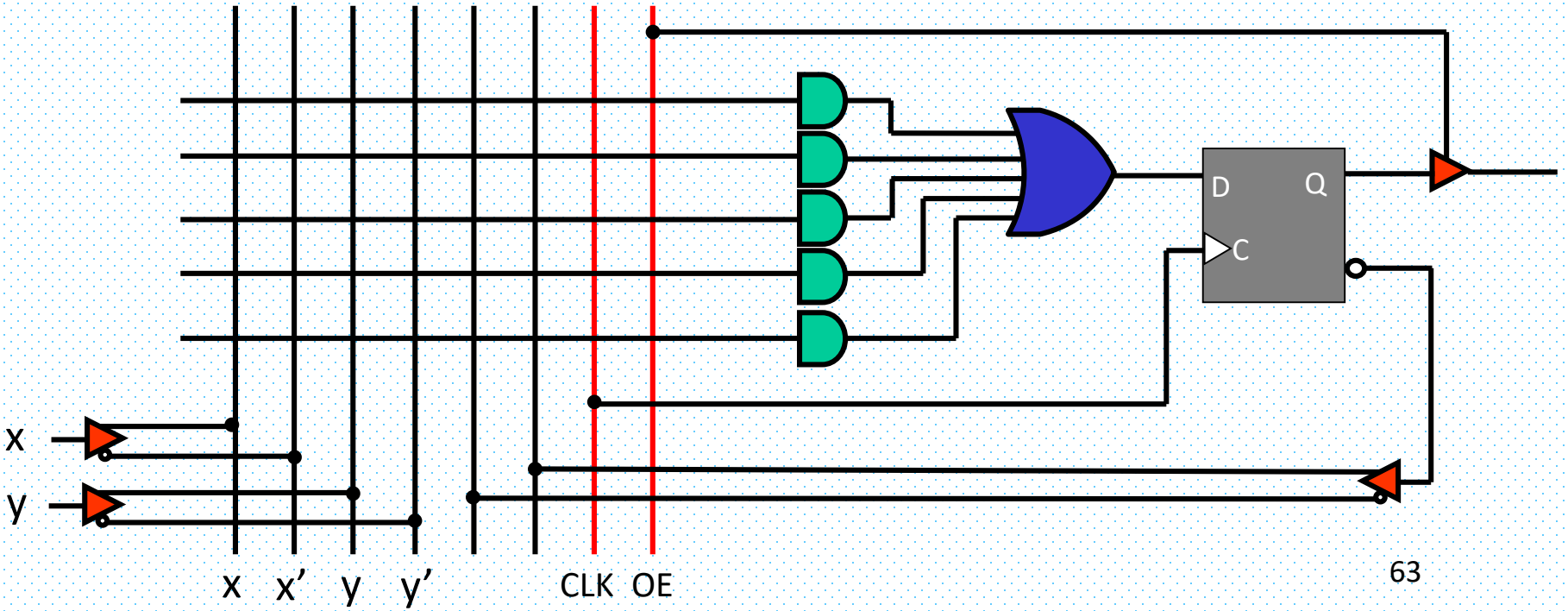


Sequential Programmable Logic Device (SPLD)

- Additional programmable connections are available to include flip-flop outputs (current state) in the product terms.
- Example: AMD 22V10
  - 24 pin device, 10 output logic macrocells
  - The number of product terms allocated to an output varied from 8 to 16

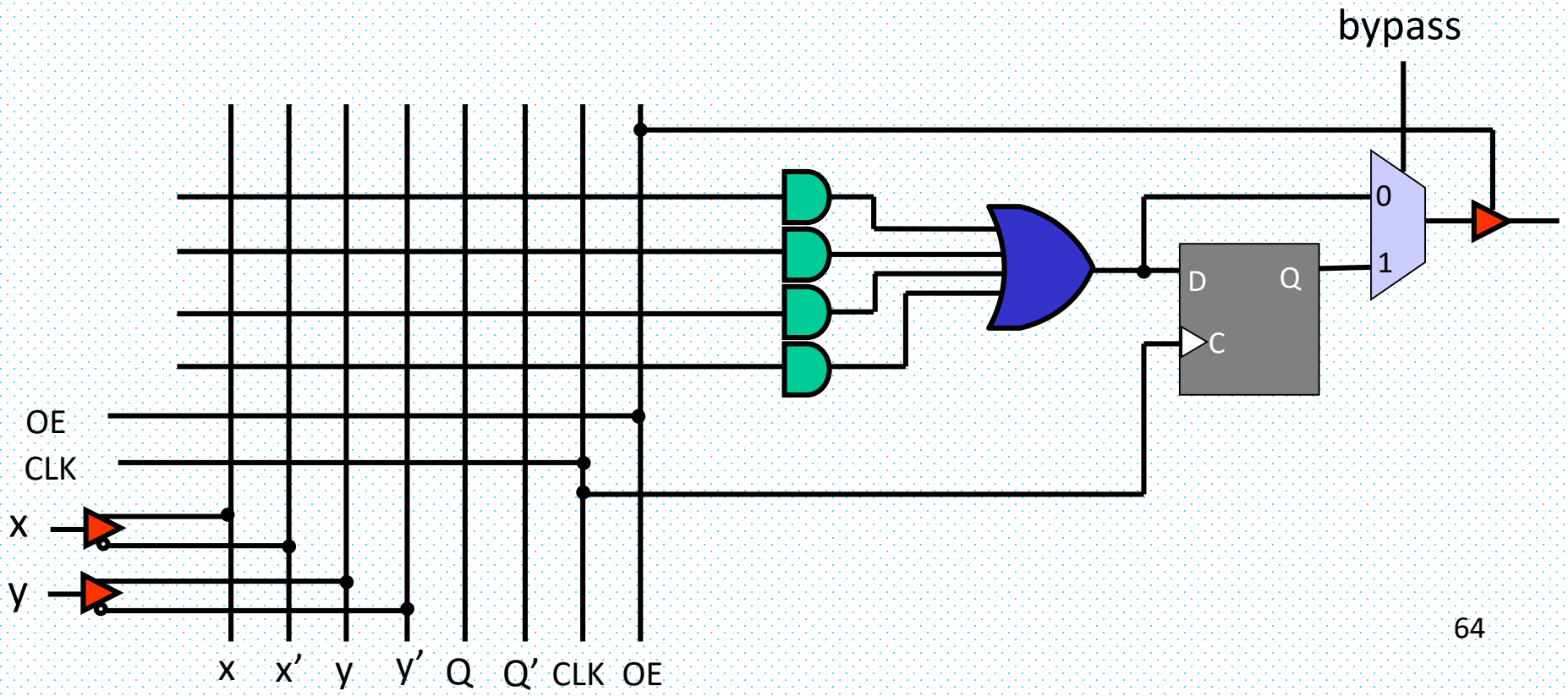
SPLD Macrocell

- SPLD is usually PAL + D flip-flops
- Each section in SPLD is called macrocell.
- A macrocell
  - sum-of-products combinational logic + optional flip-flop
  - 8-10 macrocells in one IC package

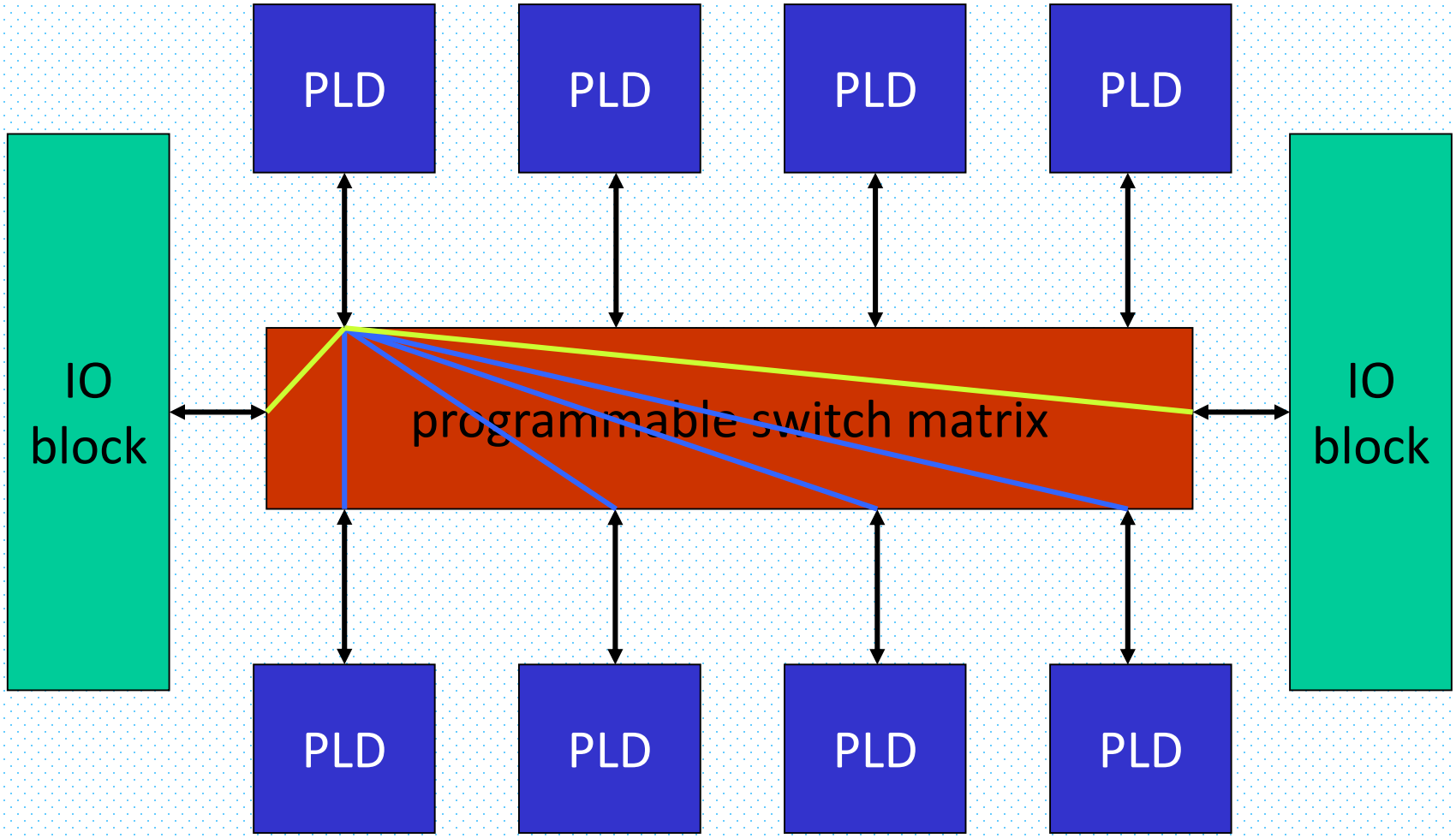


Additional SPLD Functionalities

- Additional SPLD Functionalities
  - Bypass circuitry for the output (bypassing) flip-flop
  - selection of clock edge polarity
  - XOR gate for selection of true or complement of output



Complex Programmable Logic Device

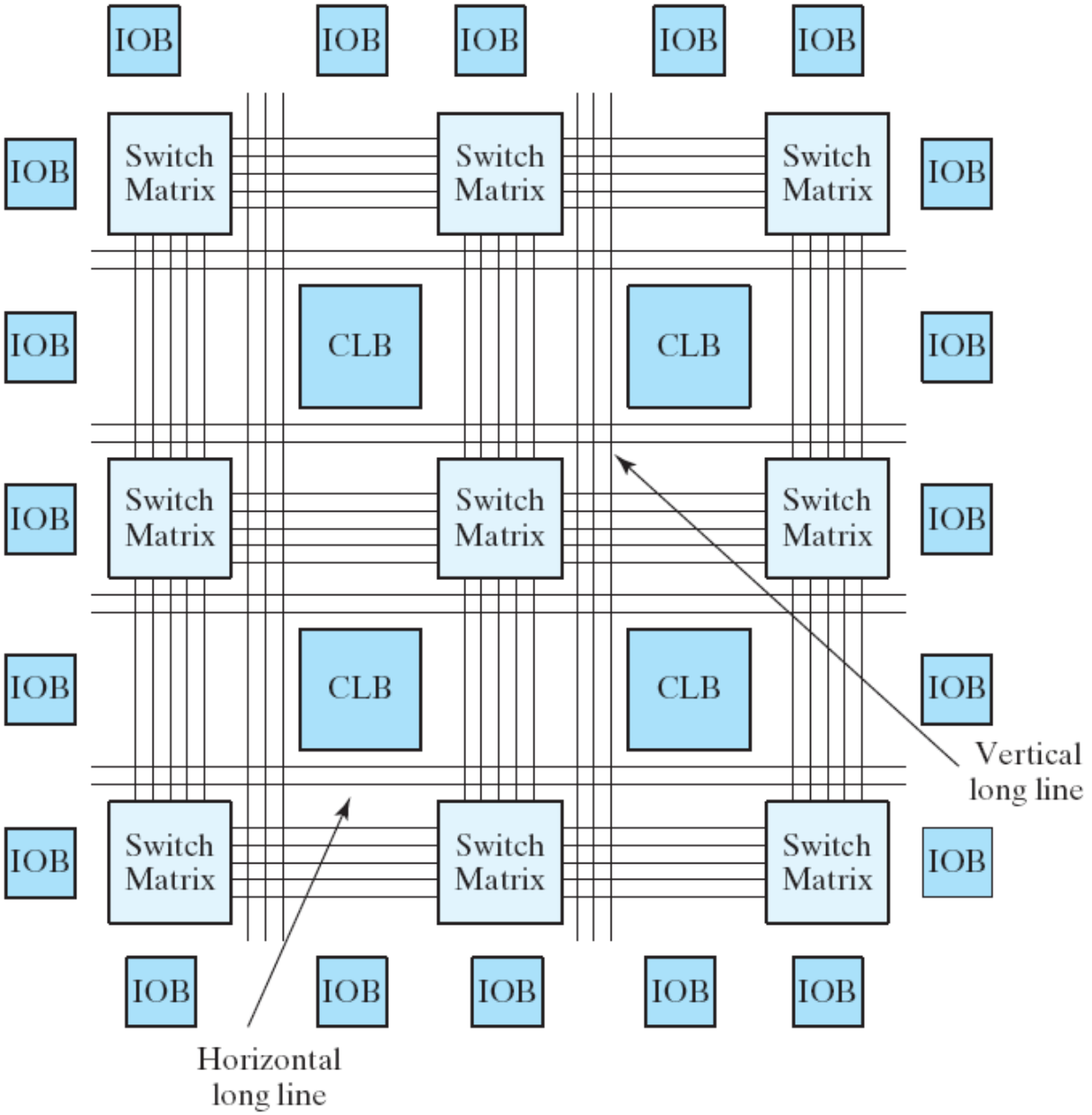


Example: Altera MAX 7000-series CPLD with 2500 gates

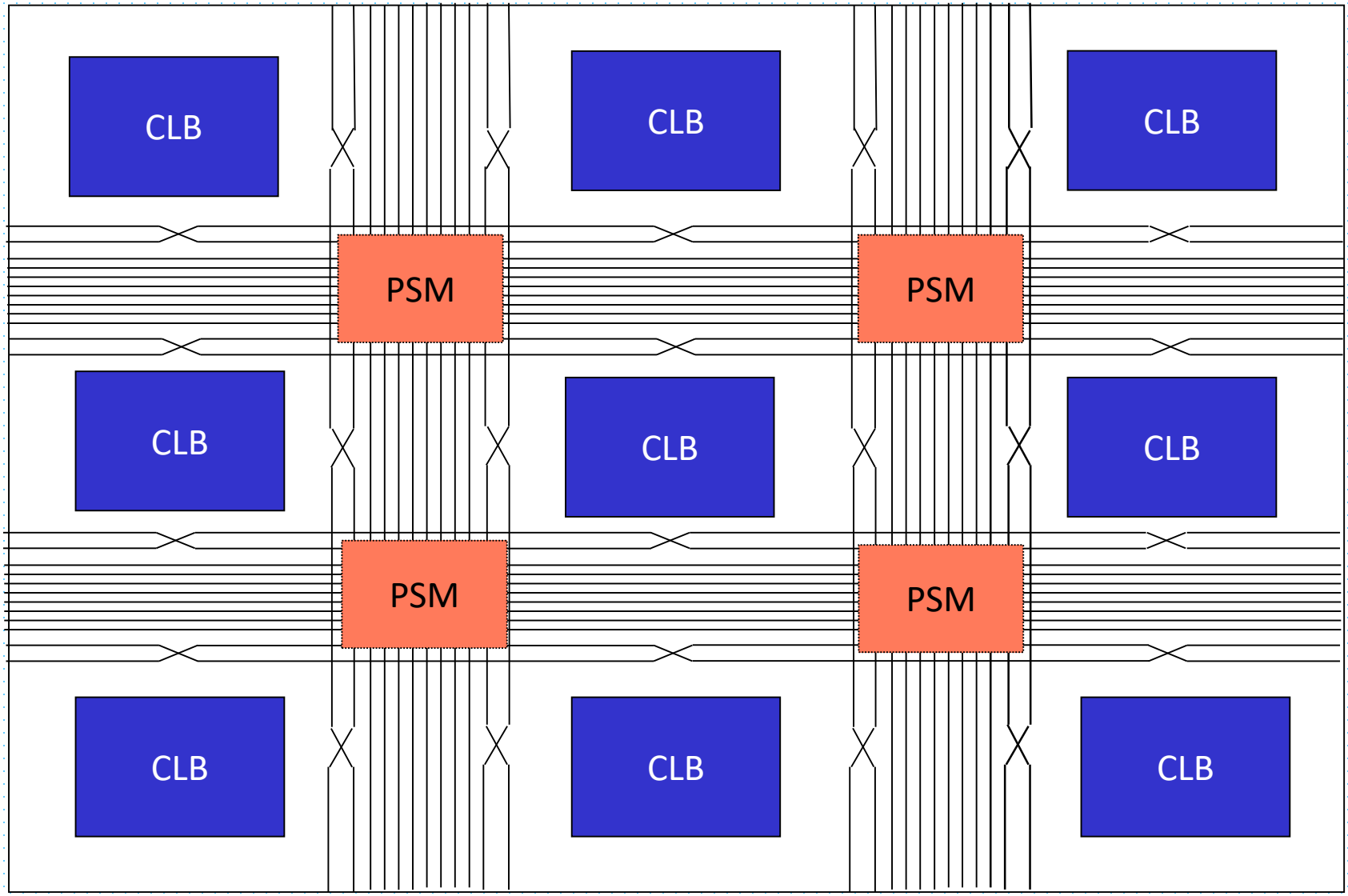
FPGA

- Field Programmable Gate Array
  - FPGA is a VLSI circuit
  - Field programmable means user can program it in his own location
  - Gate array consists of a pattern of gates fabricated in an area of silicon
    - pattern of gates are repeated many (thousand) times
  - one thousand to millions of gates are fabricated within a single IC chip

Basics of FPGA



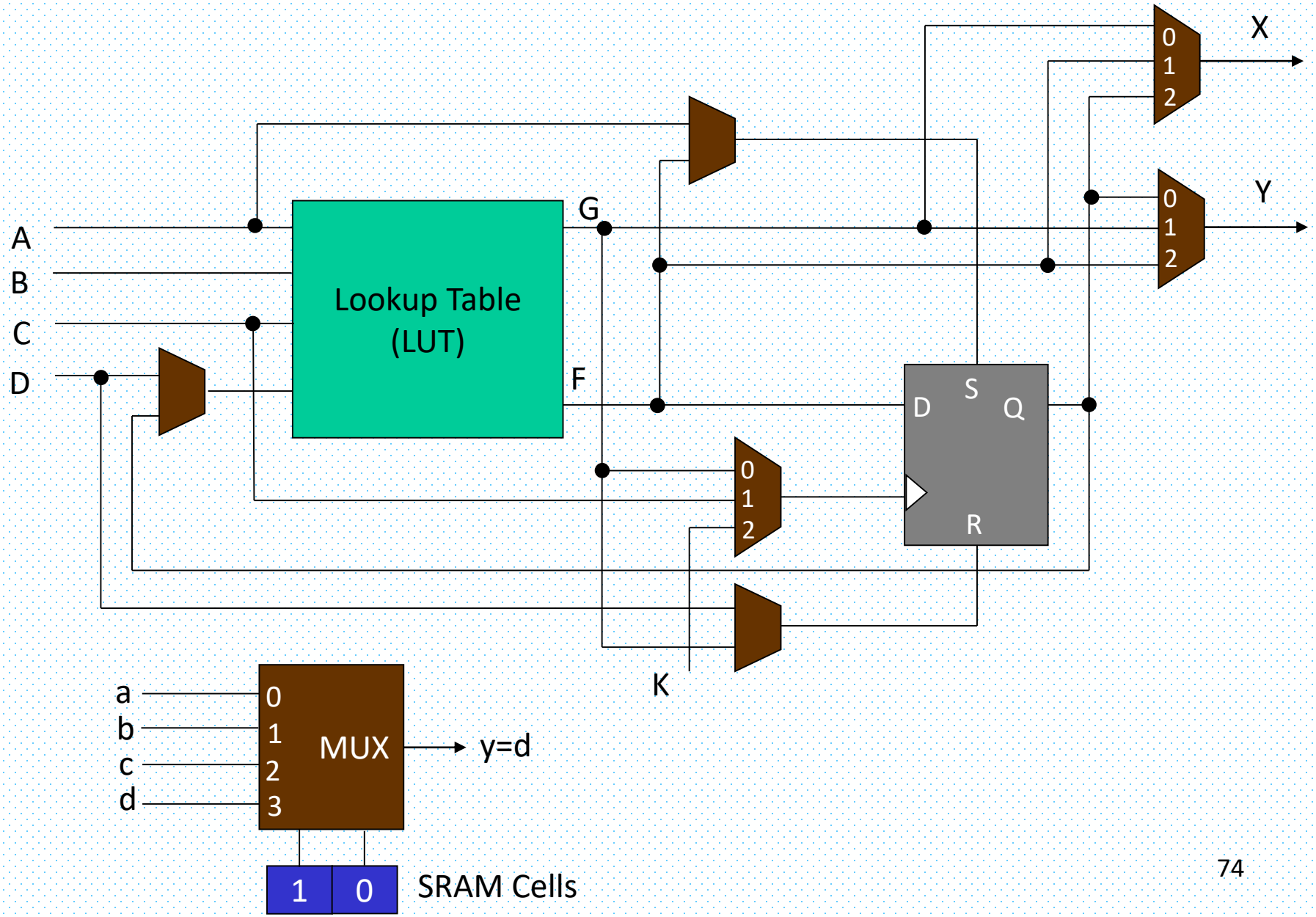
Basics of FPGA



Basics of FPGA

- A typical FPGA consists of an array of hundreds, thousands of configurable logic blocks (CLB)
  - CLBs are connected to each other via programmable interconnection structure
  - CLBs are surrounded by I/O blocks for basic communication with outside world.
- CLBs consist of look-up tables, multiplexers, gates, and flip flops
- Look-up table
  - is a truth table stored in an SRAM
  - provides the combinational circuit functions for the logic block.
  - It is like a ROM implemented as SRAM

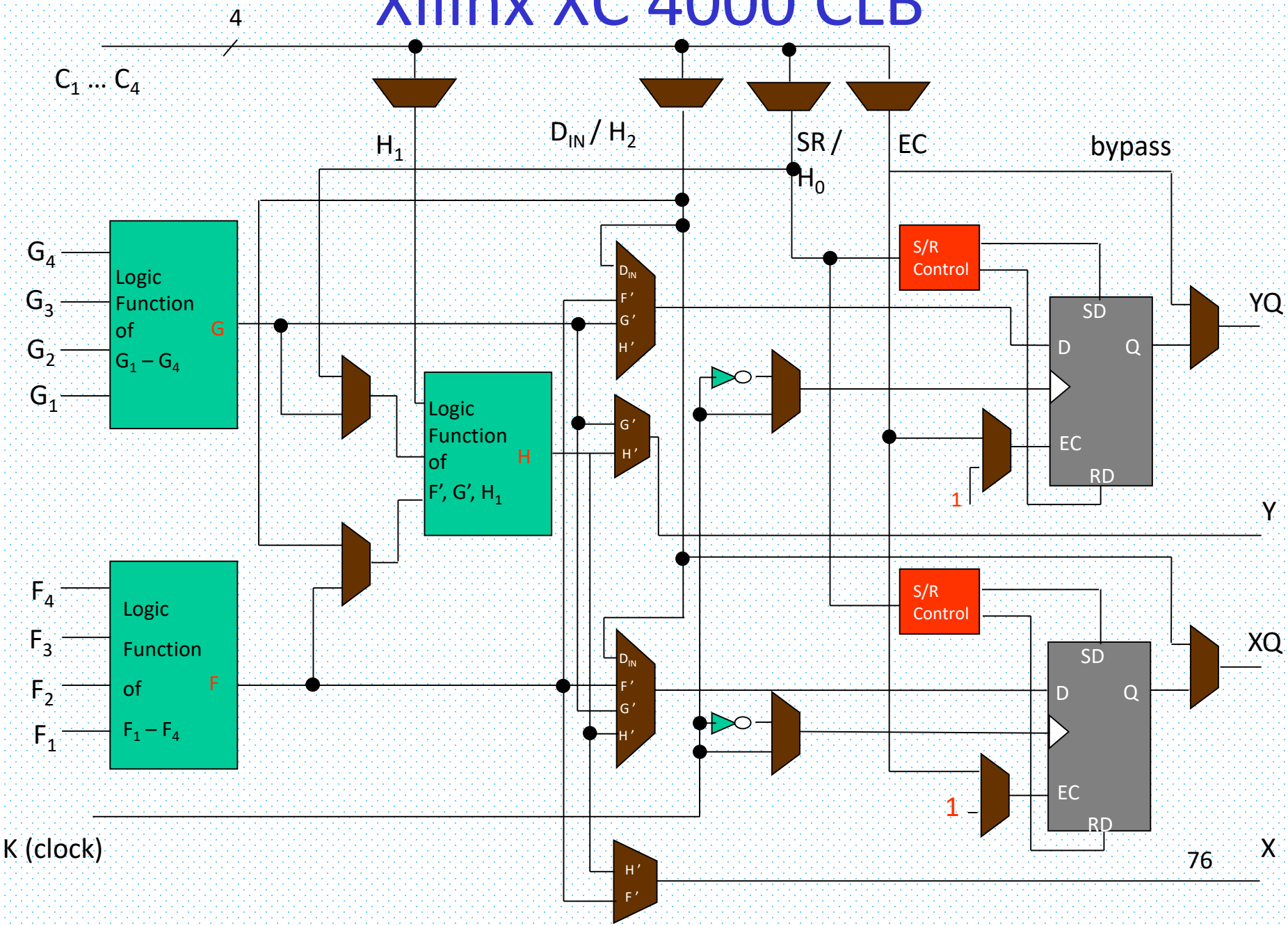
Xilinx FPGA – CLB (Partial View)



Inside CLB

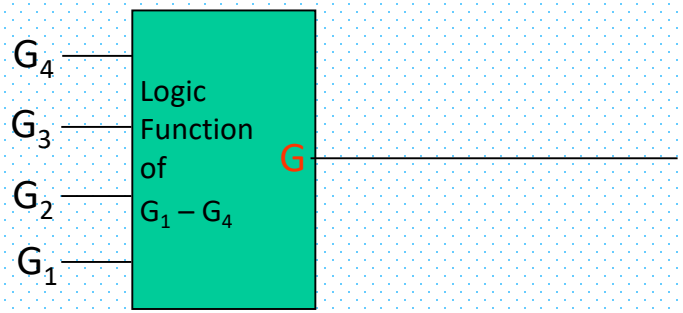
- Lookup Table
  - 16×2 ROM (implemented as SRAM)
  - Implements two four-variable Boolean functions
  - Can also be configured as memory (RAM)
- Multiplexers
  - 2^k input multiplexers
  - controlled by k SRAM cells
- Flip-flop
  - provides operation as a sequential system
  - can be configured as a latch as well.

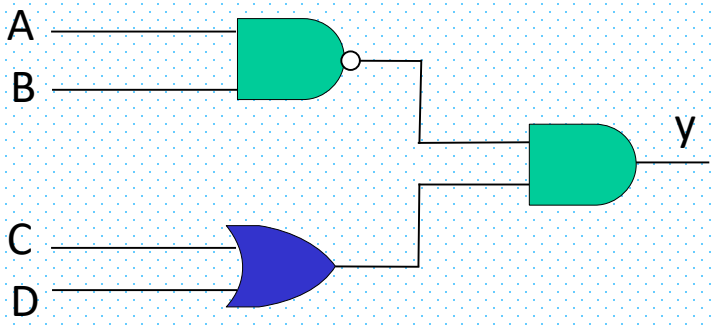
Xilinx XC 4000 CLB



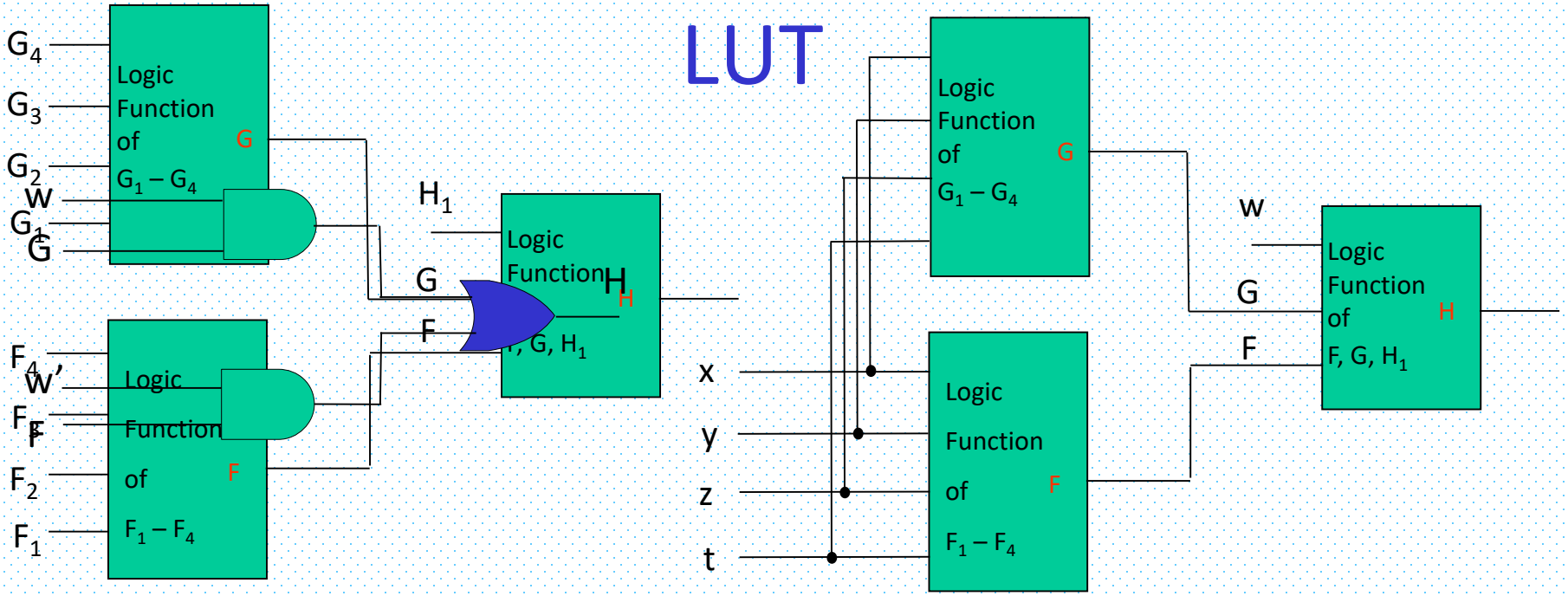
Lookup Tables (LUT)

- Combinational circuits are implemented using LUT



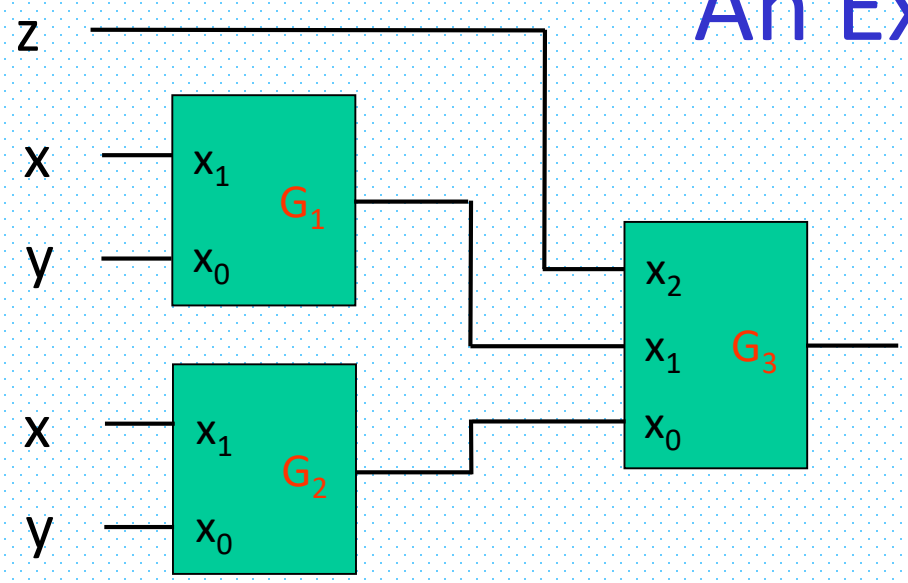


G_4(A)	G_3(B)	G_2(C)	G_1(D)	G(y)
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



- Using three function generators F, G, and H
  - A single CLB can implement <u>any</u> Boolean function of five variables
    - General form of Boolean function of five variables:
      $H = F(x, y, z, t) \cdot w' + G(x, y, z, t) \cdot w$
  - <u>Some</u> functions of up to nine variables
    - Nine logic inputs: F_1, F_2, F_3, F_4, G_1, G_2, G_3, G_4, and H_1.

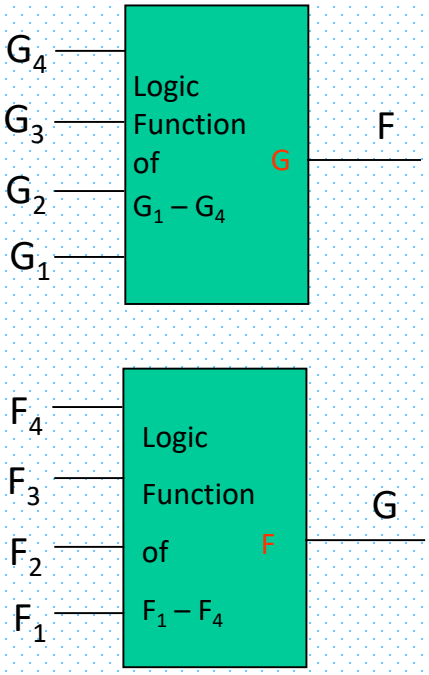
An Example



$$G_3 = G_1(x_1, x_0)x_2 + G_2(x_1, x_0)x_2'$$

- $F(x, y, z) = \Sigma(0, 1, 2, 4)$
- $F(x, y, z) = x'y'z' + x'y'z + x'yz' + xy'z'$
- $F(x, y, z) = (x'y')z + (x'y' + x'y + xy')z'$
- $G_1(x_1, x_0) = x'y'$
- $G_2(x_1, x_0) = x'y' + x'y + xy' = x' + y'$
- $G_3(x_1, x_0) = (x'y')z + (x'y' + x'y + xy')z'$

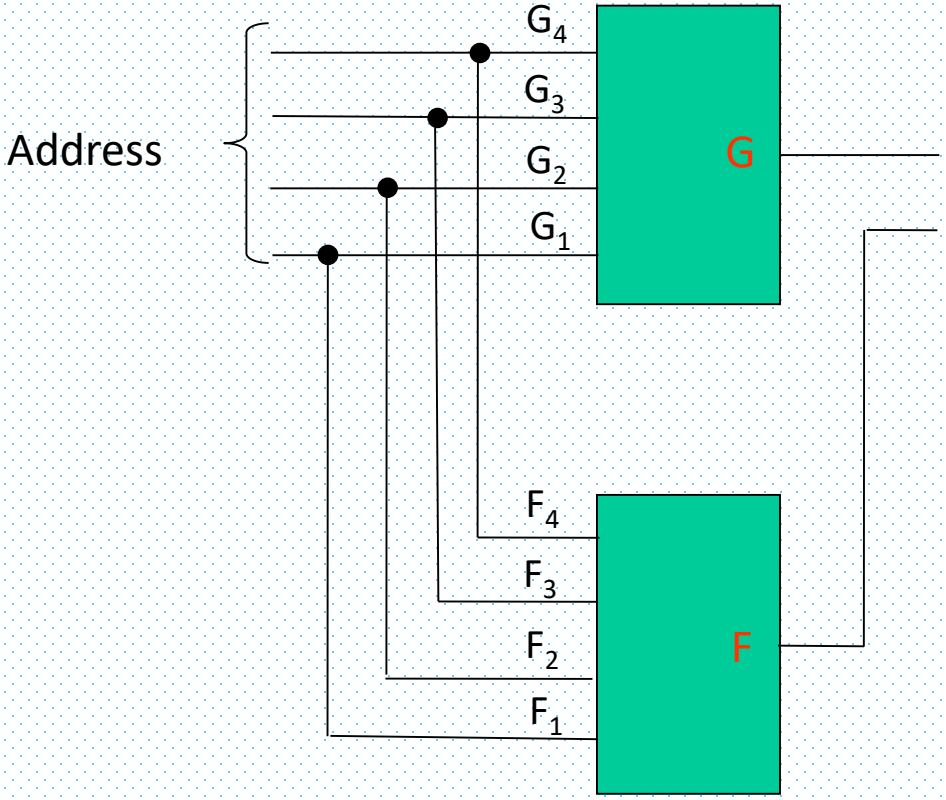
LUT as Memory



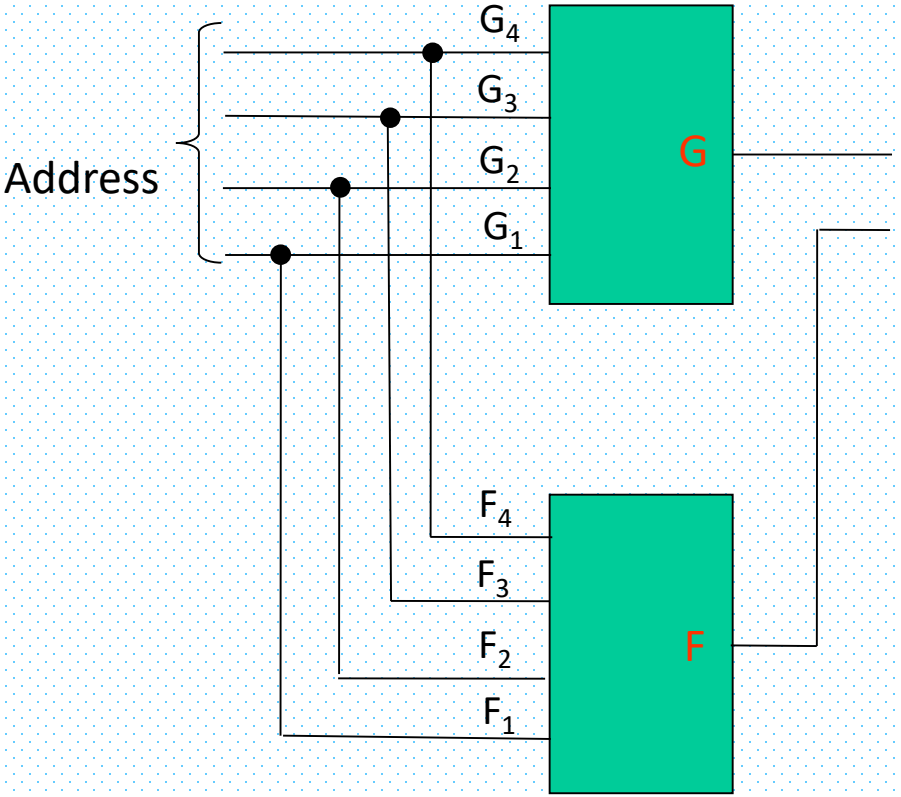
- LUTs can be configured as memory block
  - One 16x2 memory module
  - One 32x1 memory module
  - Dual-ported 16x1 memory module
  - Synchronous-edge-triggered and asynchronous-memory interfaces are supported

LUT as Memory

16x2 configuration



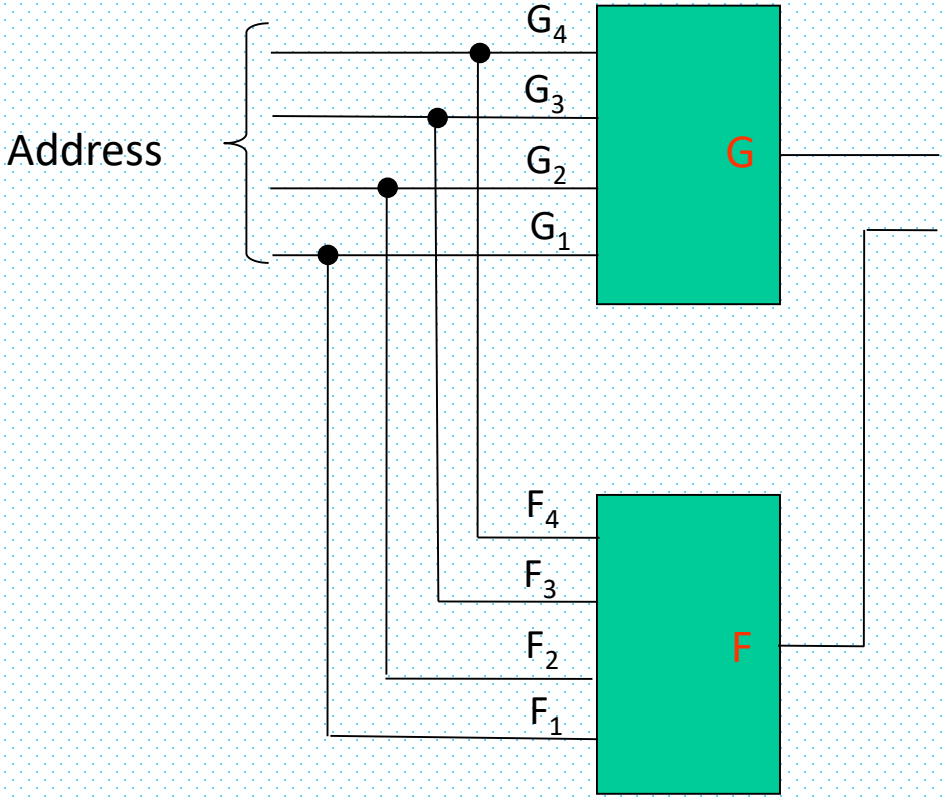
16x1 dual-ported configuration



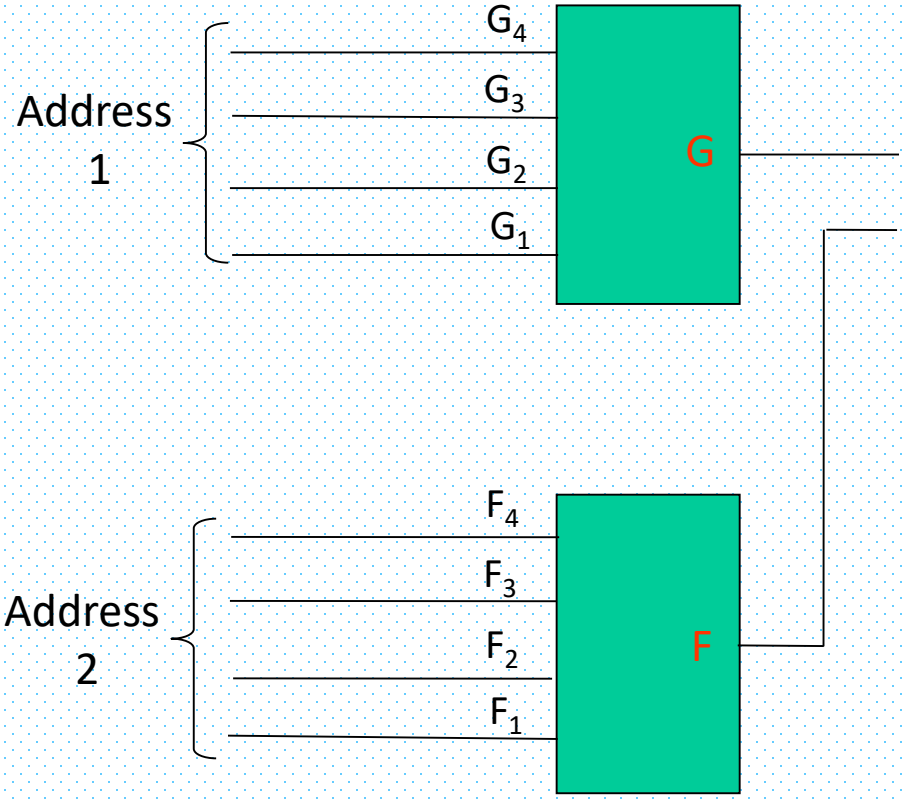
One write

LUT as Memory

16x2 configuration



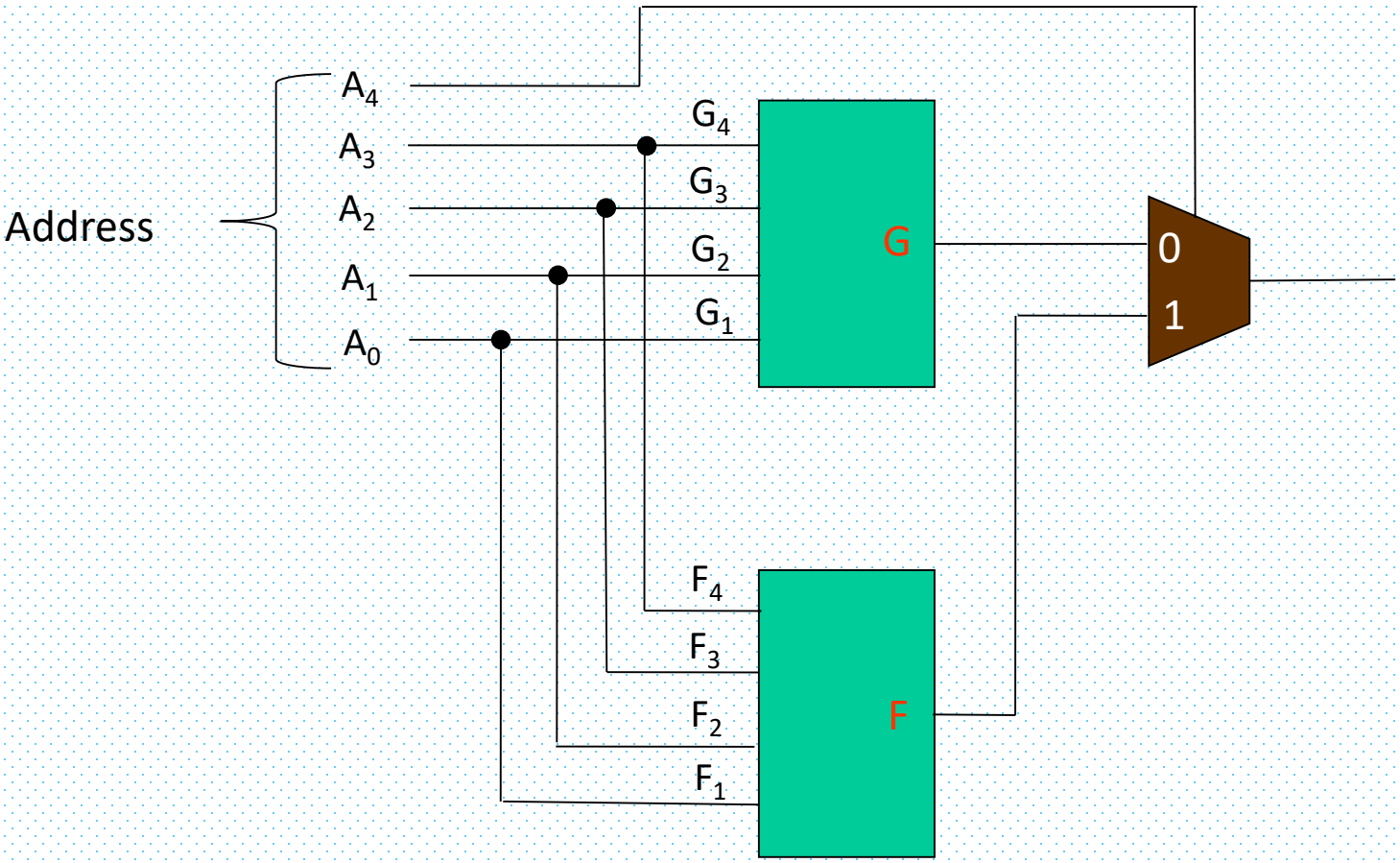
16x1 dual-ported configuration



Two reads at the same time

LUT as Memory

32x1 configuration

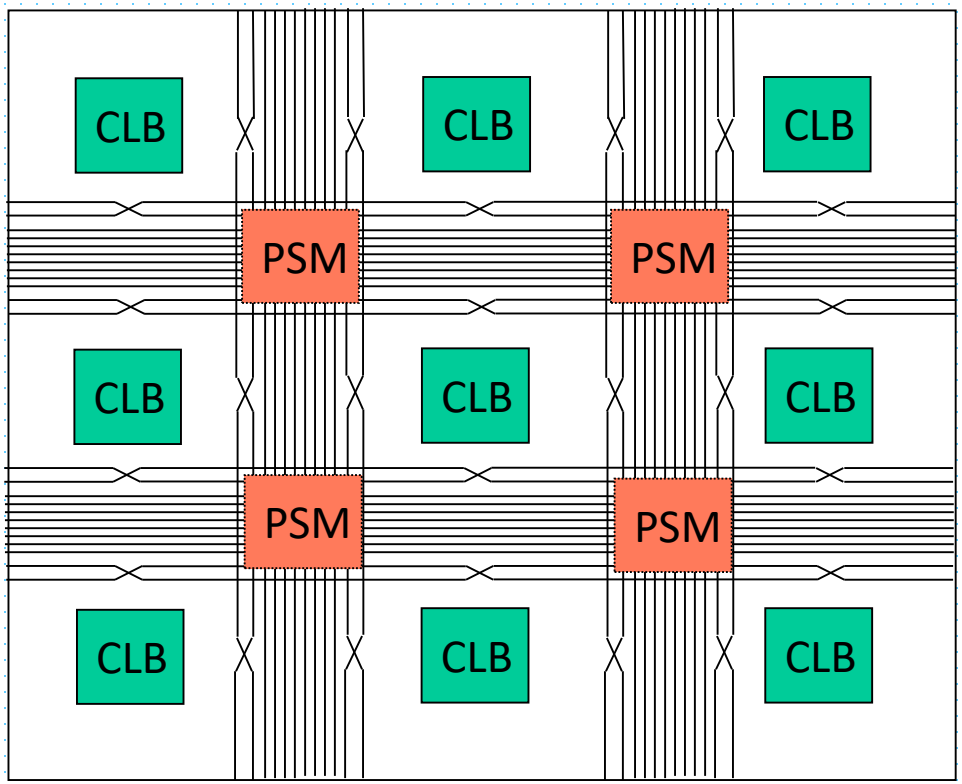


A Simple Memory on FPGA

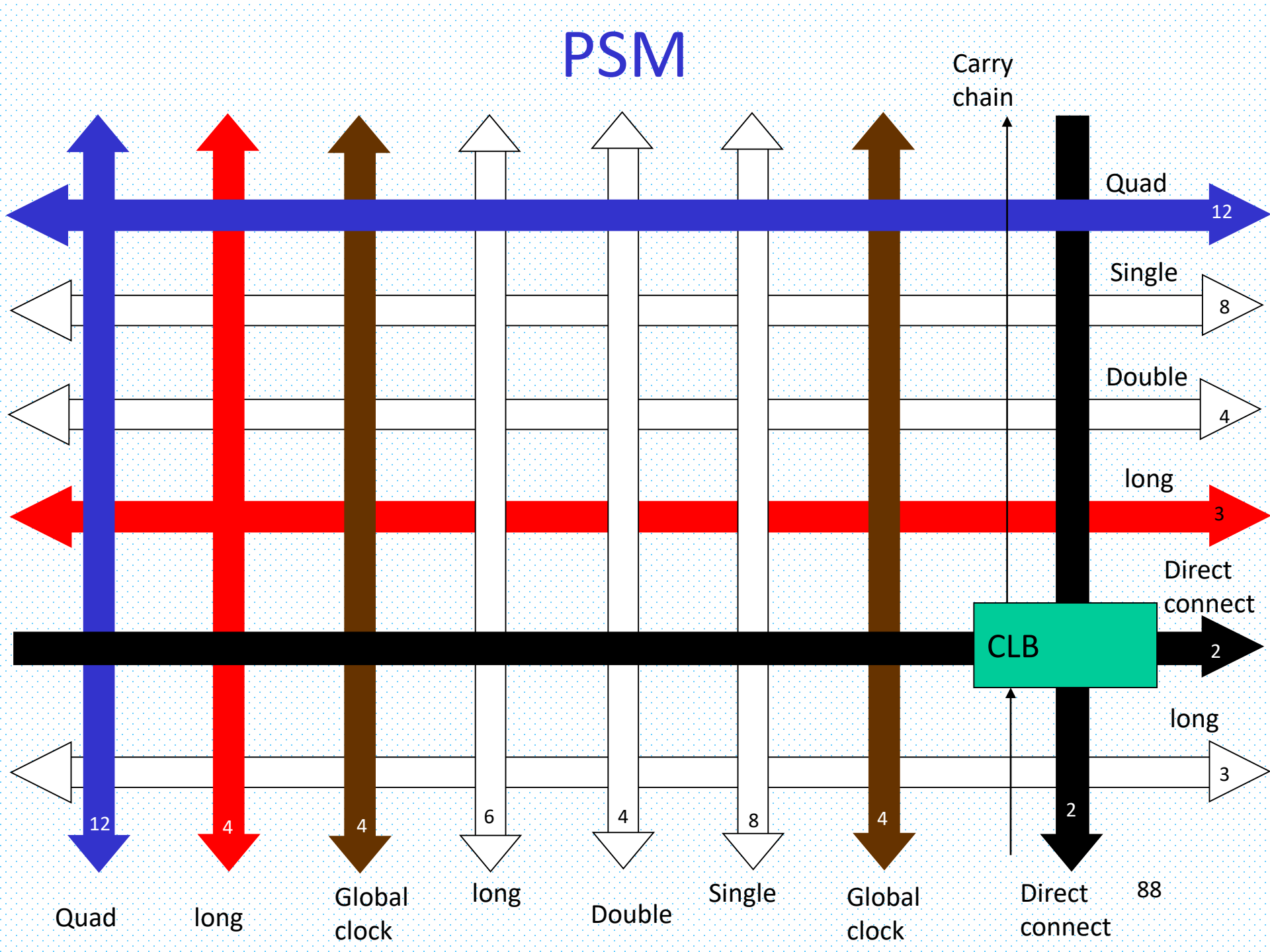
```
module simple_memory(clk, reset, dat_in, wr_adr, wr_en, dat_out, rd_adr);
input  clk, reset;
input  [15:0]  dat_in;
input  [7:0]   wr_adr;
input  wr_en;
output [15:0] dat_out;
input  [7:0]   rd_adr;

// synthesis attribute ram_style of my_memory is distributed
reg [15:0]    my_memory[0:255];
reg [15:0]    dat_out;

always @(posedge clk)
begin
    if(wr_en)
       my_memory[wr_adr] <= dat_in;
    dat_out <= my_memory[rd_adr];
end
endmodule
```

Switch Matrix

- Different levels of interconnections
  - Direct interconnects between CLBs. Short wires to connect adjacent CLBs in both directions.
  - Switch matrix



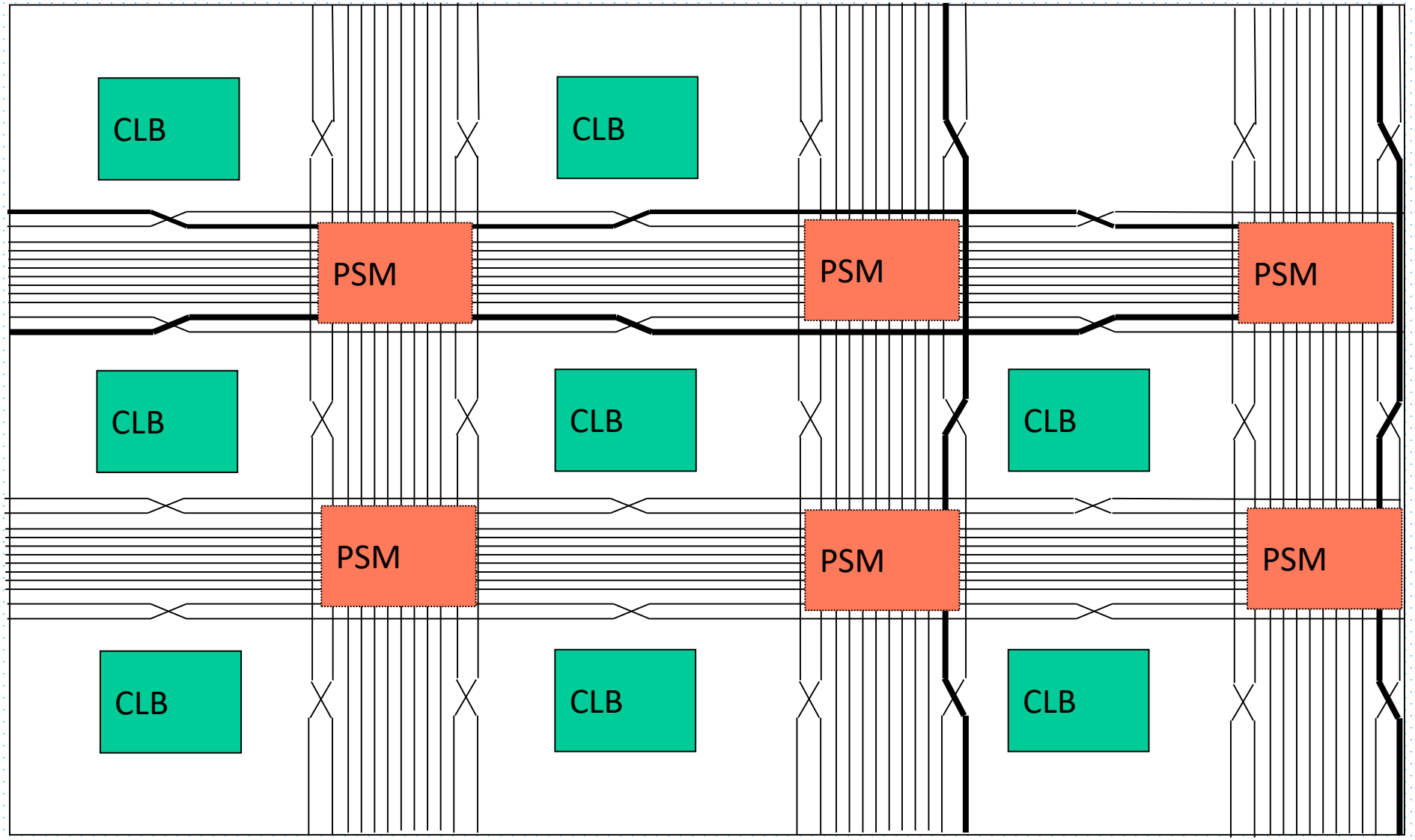
- PSM: **P**rogrammable **S**witch **M**atrix
- PSM can be configured to connect a horizontal wire to a vertical one.
- One wire can be connected to multiple wires
- This way output of a CLB can be routed through multiple PSMs to the input of another CLB.


PSM
Carry chain
Quad
12
Single
8
Double
4
long
3
Direct connect
CLB
2
long
3
12
4
4
6
4
8
4
2
Quad
long
Global clock
long
Double
Single
Global clock
Direct connect
88

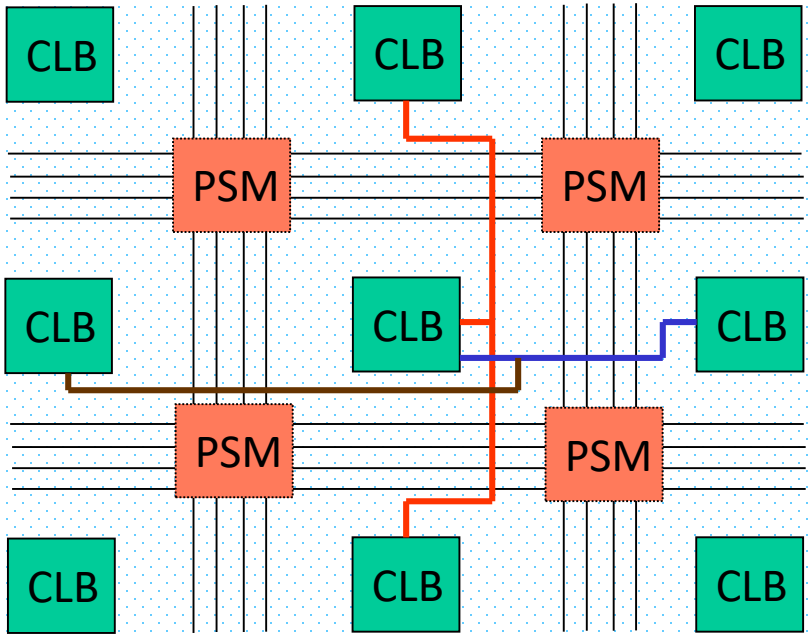
Types of Lines

- Single-length: connects adjacent PSMs
- Double-length: connects every other PSM
- Quad-length: traverse four CLBs before passing through a PSM.
- Long: runs entire chip.
  - Using tri-state buffers within the CLBs, long lines can be configured as buses.
- Local connections use direct interconnect or single length lines in order to avoid to many switching points
- Global Nets: Low skew signal paths used to distribute high fan-out signals such as clock and reset signals.

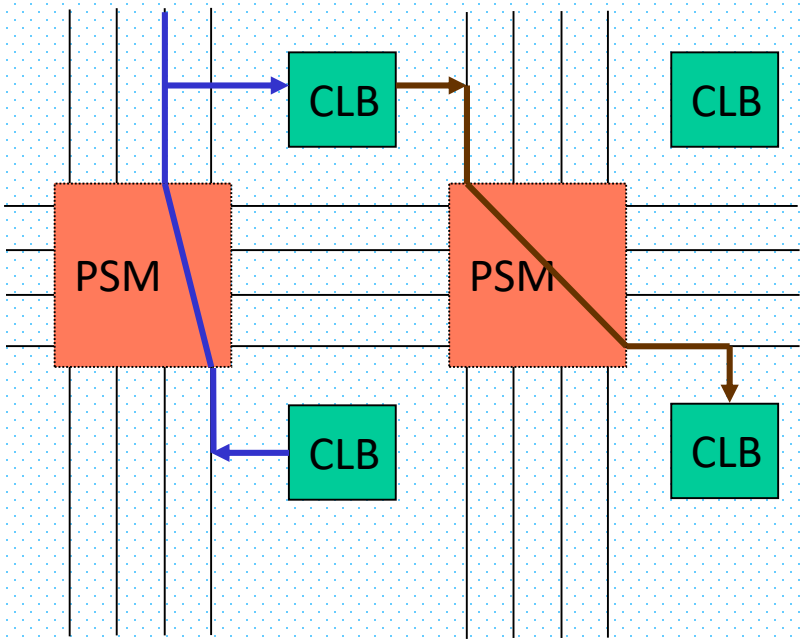
Double Length Lines



Example Interconnects

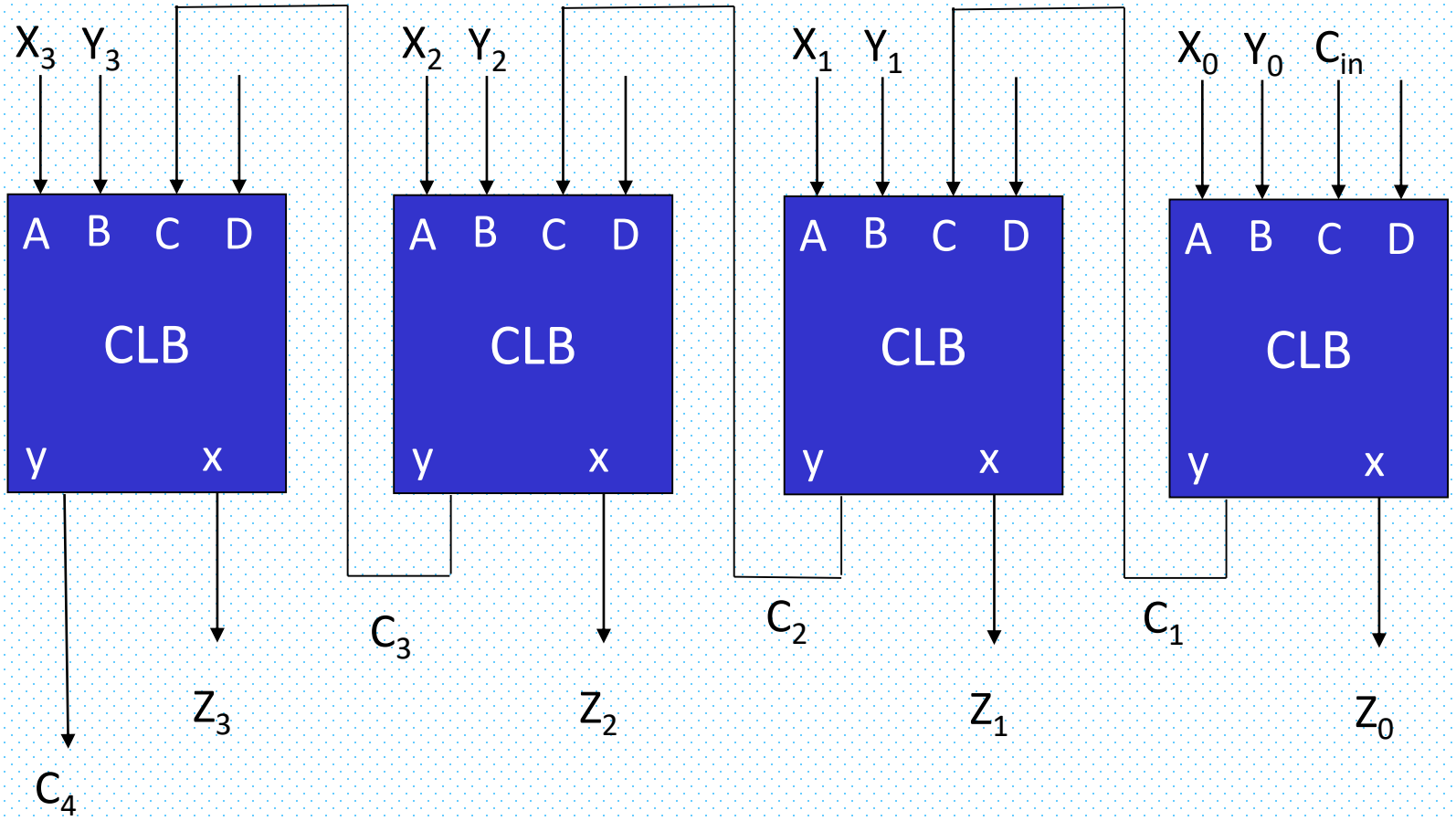


Direct interconnects between adjacent CLBs

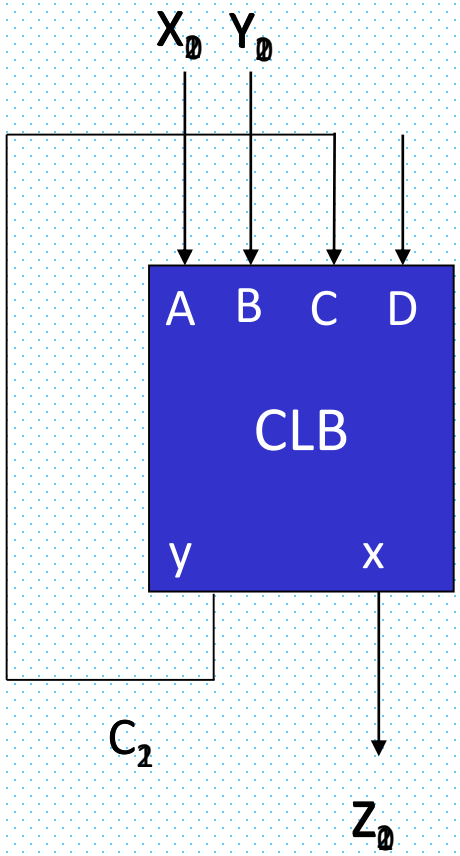


General-purpose interconnects

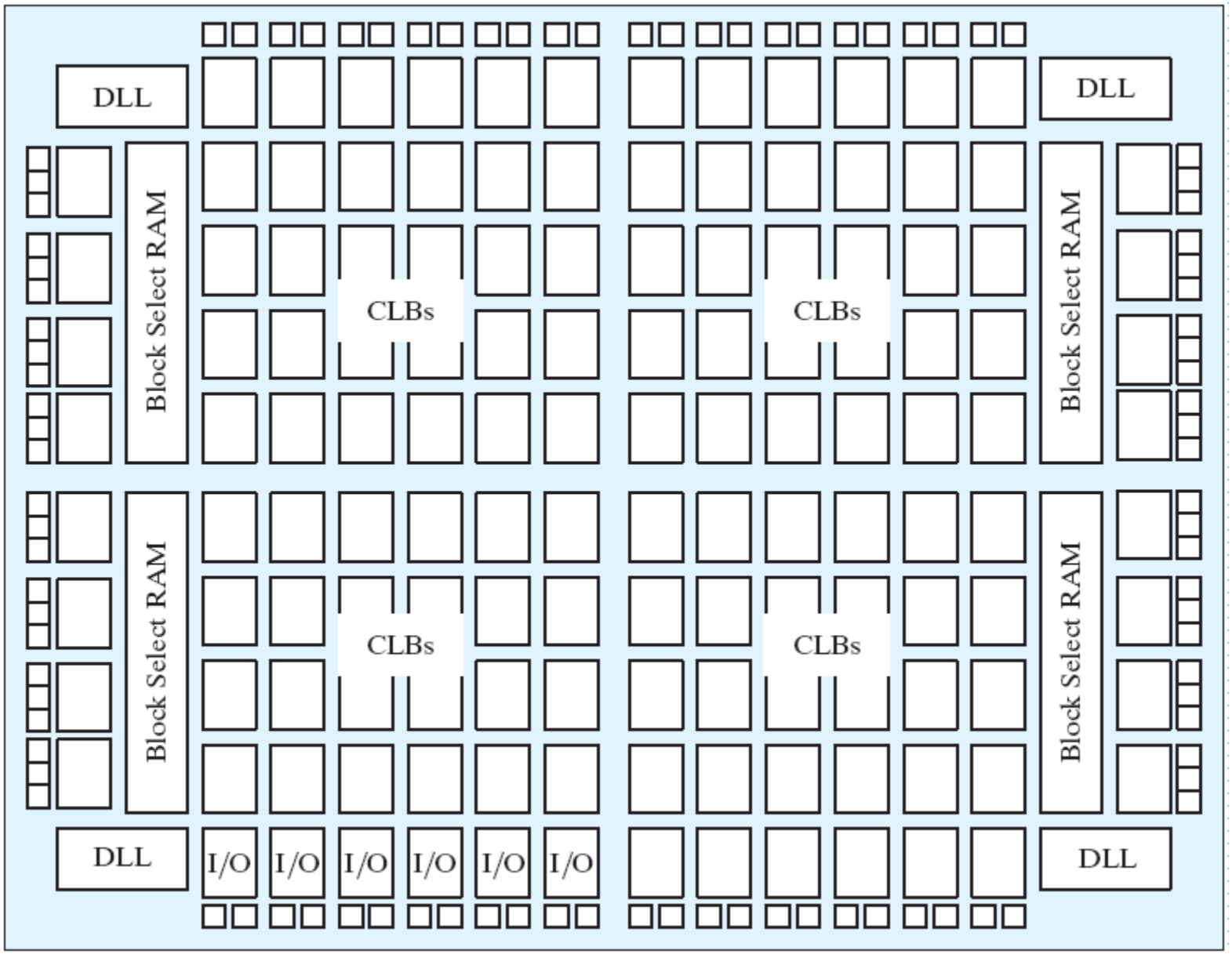
4-bit Adder with CLBs 1/2



4-bit Adder with CLBs 2/2



Spartan II Architecture



Xilinx Spartan II Characteristics

- Density: up to 200,000 system gates
  - Up to 5292 logic cells
  - Each cell contains a LUT
- Operating voltage: 2.5 V
- Operating frequency: 200 MHz
- On-chip **block** memory
  - not made up of look-up tables
  - Does not reduce the amount of logic
  - Improve performance by reducing the need to access off-chip storage.
- 0.22/0.18-µm CMOS technology
  - Six layers of metal for interconnect

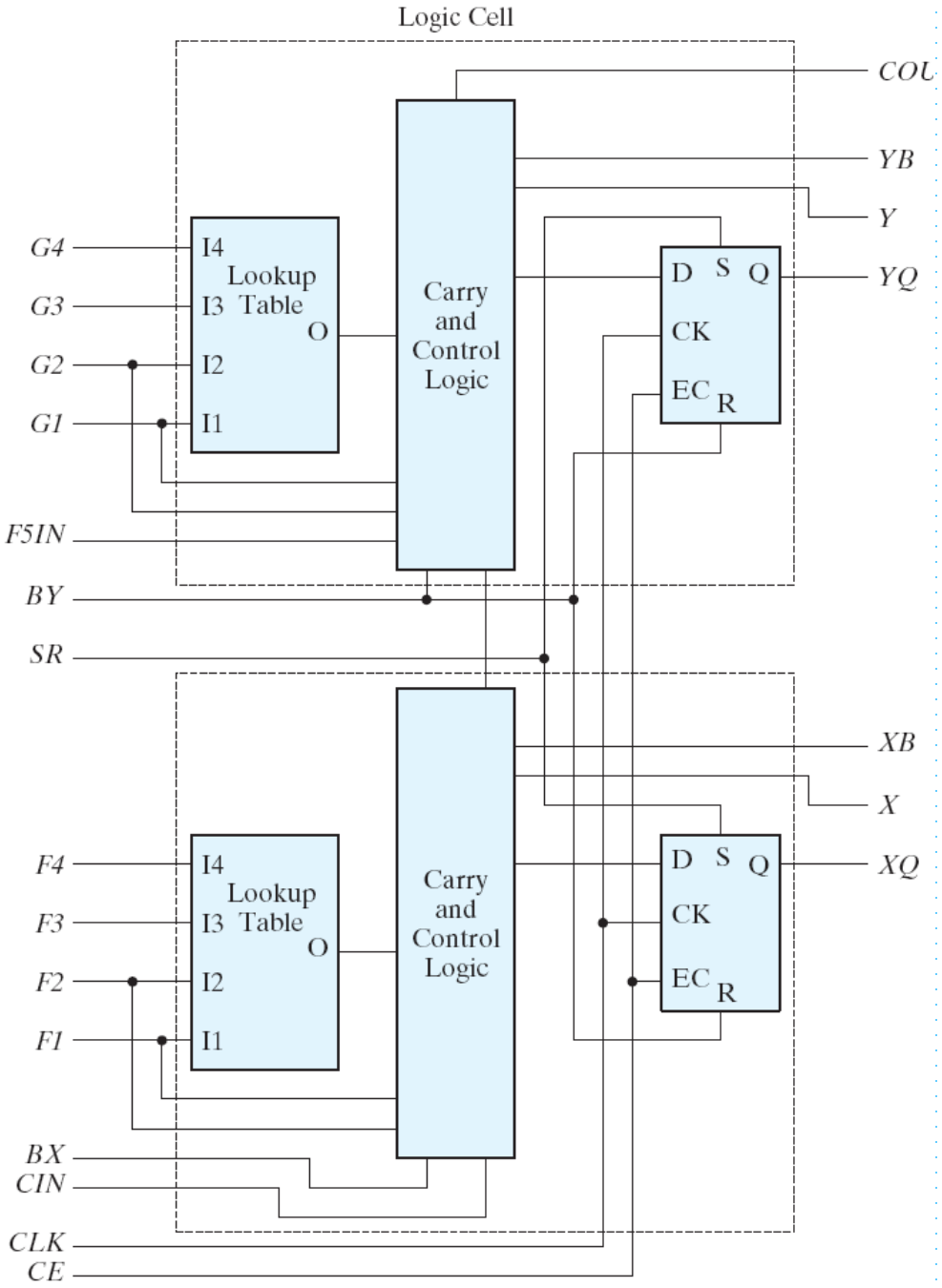
A Simple Memory on FPGA

```
module simple_memory(clk, reset, dat_in, wr_adr, wr_en, dat_out,
   rd_adr);
input  clk, reset;
input  [15:0]  dat_in;
input  [7:0]   wr_adr;
input  wr_en;
output [15:0] dat_out;
input  [7:0]   rd_adr;

// synthesis attribute ram_style of my_memory is block
reg [15:0]    my_memory[0:255];
reg [15:0]    dat_out;

always @(posedge clk)
begin
    if(wr_en)
       my_memory[wr_adr] <= dat_in;
    dat_out <= my_memory[rd_adr];
end
endmodule
```

Xilinx Spartan II Characteristics

- Reliable clock distribution
  - Clock synchronization through delay-locked loops (DLLs)
  - DLLs eliminate clock distribution delay
  - DLLs provide frequency multipliers, frequency dividers
- Different architecture
  - Four quadrants
  - Each quadrant is associated with 4096-bit block RAM
  - There are FPGAs with up to 14 blocks of block RAM (56K bits total block memory)

Slice

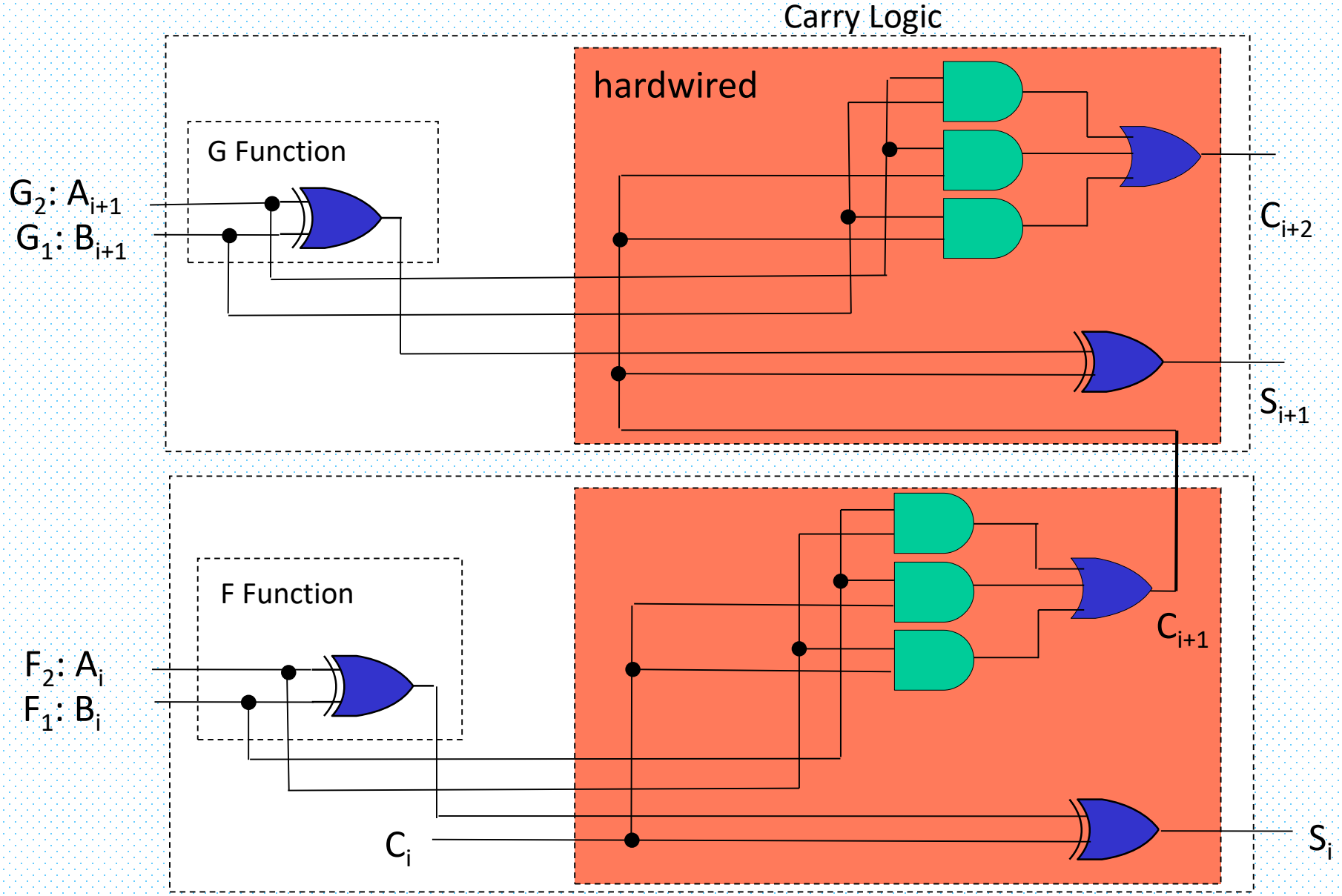


- A logic cell contains a four-input lookup table, logic for carry and control and a D type flip-flop
- Each slice contains contains two cells
- Each CLB contains two slices.

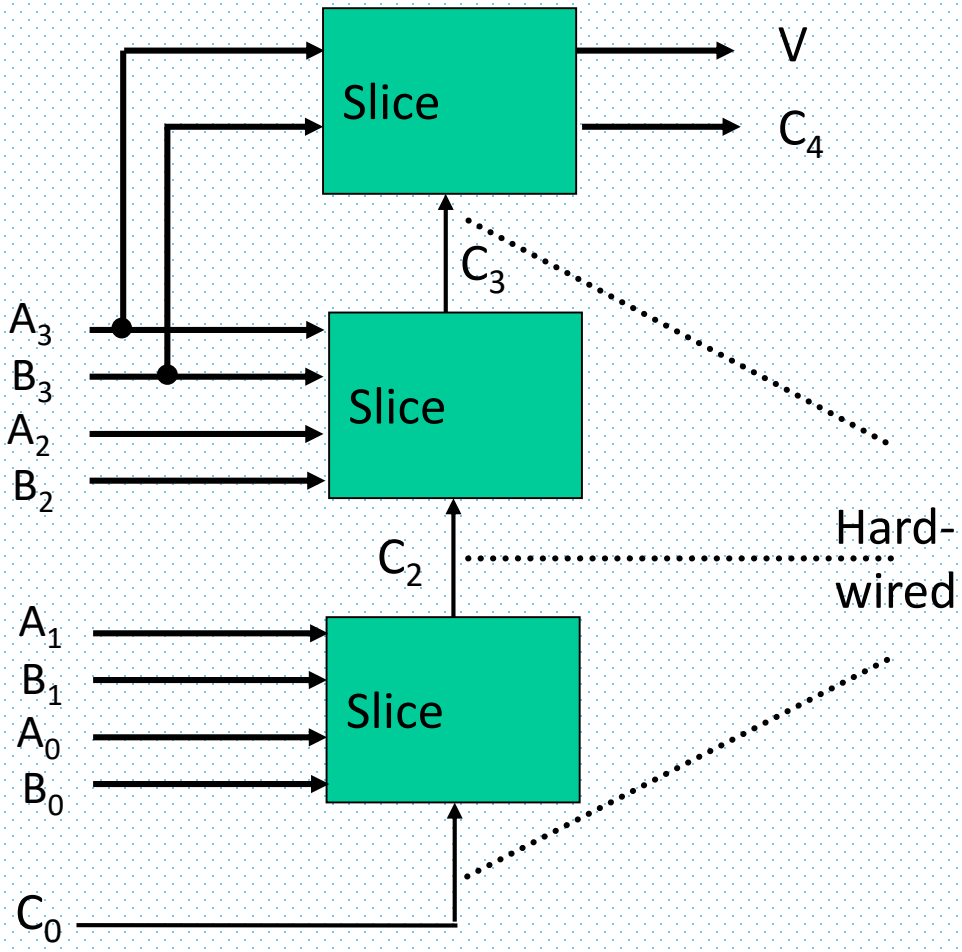
Carry Logic

- Lookup tables can be used to generate the sum bits.
- Each slice can be programmed to implement both carry and sum for two bits.
- The carry lines between cells are hardwired (not programmed) to provide for fast propagation of carries
- The carry logic can be programmed to implement subtracter, incrementer/decrementers, 2's complementers, and counters

A Slice as Two-Bit Full Adder



Connections for 4-bit Adder



- If we want to detect a possible overflow, we add the 3rd slice.
- The 2nd slice outputs C_3 instead of C_4 (How?)
- In the 3rd Slice, C_4 can be re-computed using the carry logic.
- Overflow is computed using the G function generator in the 3rd Slice
- Overflow: $V = C_3 \oplus C_4$
- 4-bit adders can easily be expanded 8 or 16-bit adders
- Adder modules are available in Xilinx library

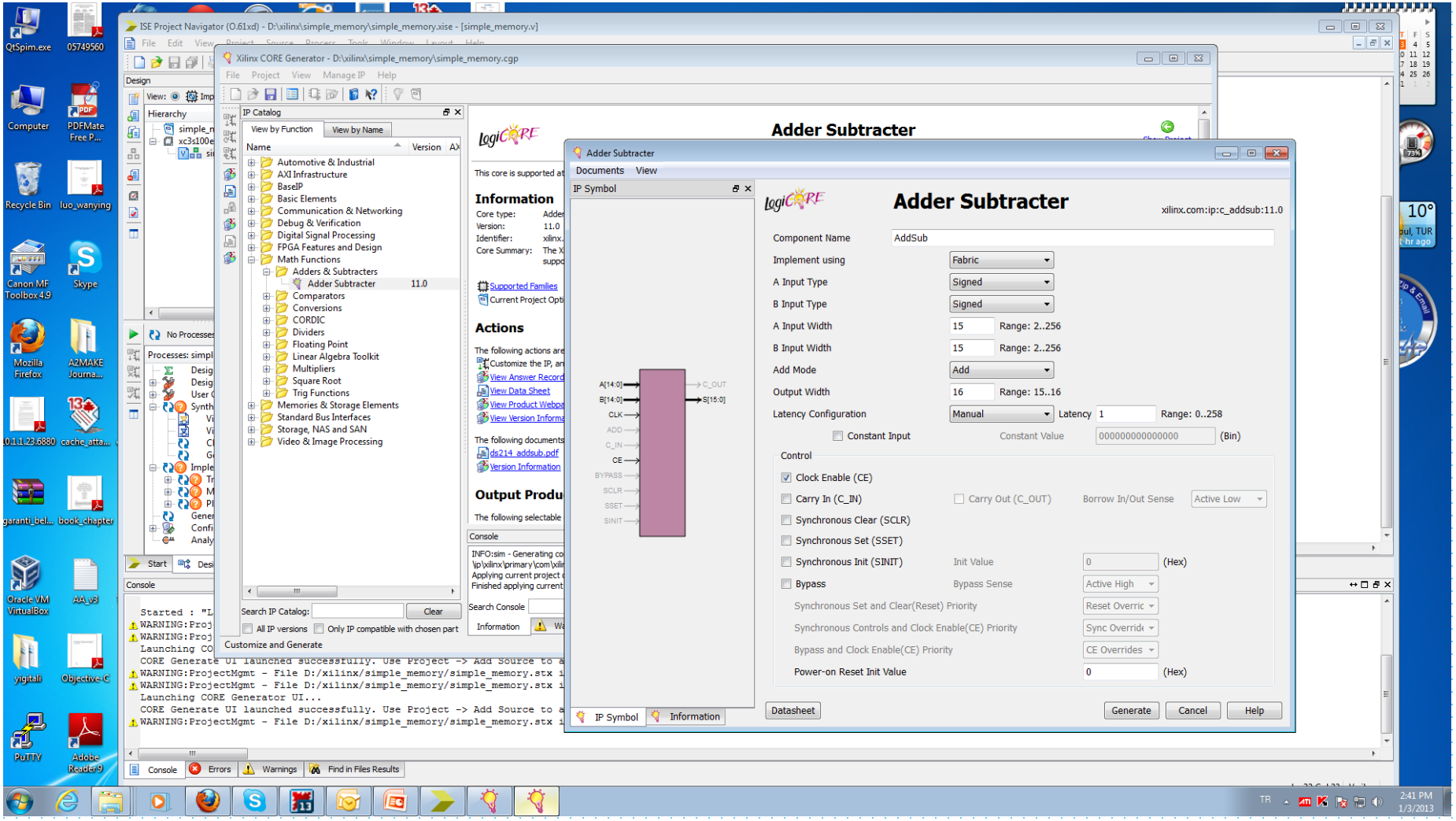
Accessing Carry Logic

- All major synthesis tools can infer carry logic for arithmetic functions
  - Addition (SUM <= A + B)
  - Subtraction (DIFF <= A - B)
  - Comparators (if A < B then…)
  - Counters (count <= count +1)

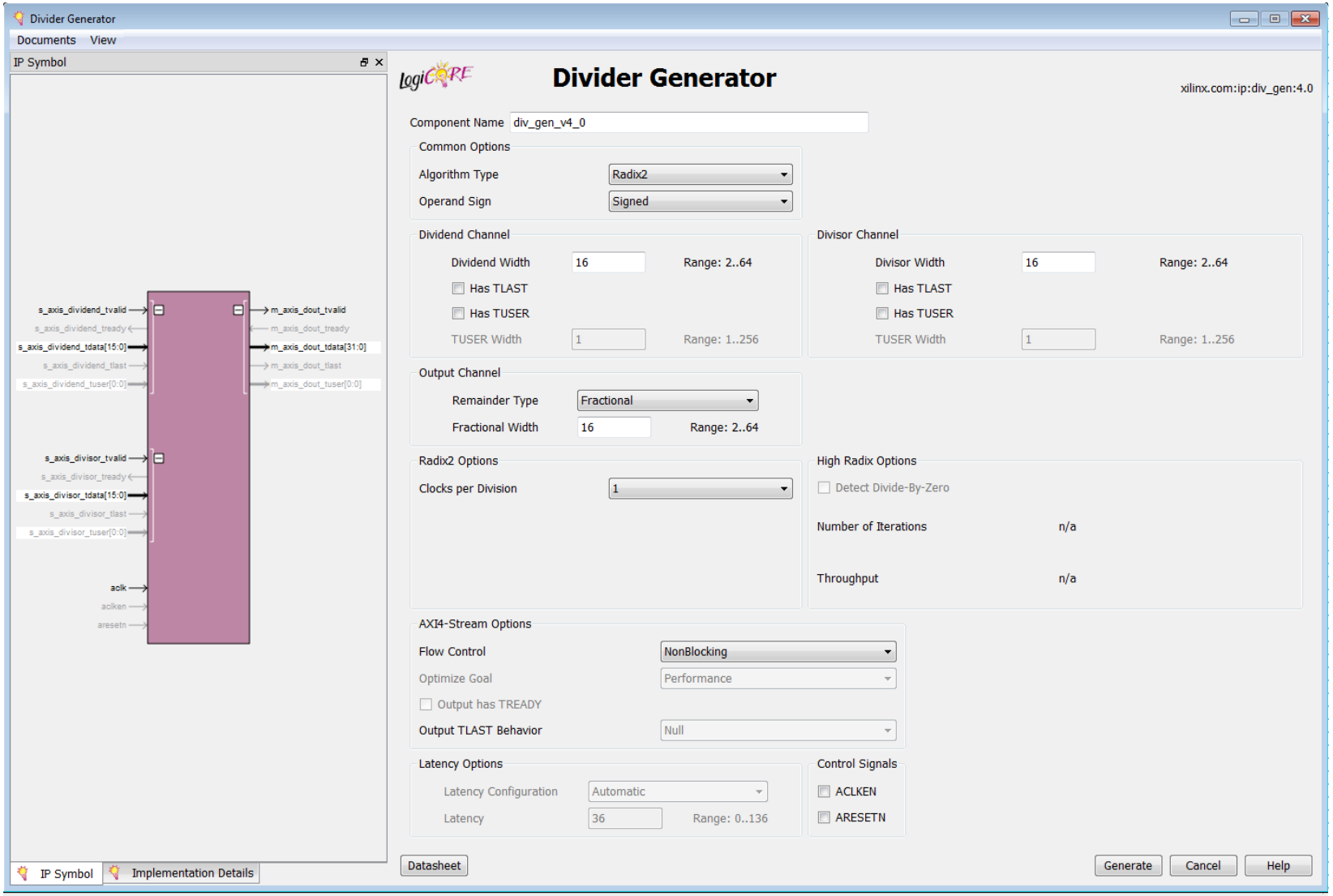
Core Generators

- Many vendors typically supply implementation of common building blocks that are optimized to the structure of their hardware components.
  - Xilinx, in fact, has a core generator utility that can create common building blocks from parameterized descriptions provided by the user
  - Adders, subtractors, multipliers, memories, etc. are such building blocks
- FPGA as a sea of LUTs and flip-flops
  - A gate-level design can be placed on the array by mapping combinational components to LUTs, sequential to flip-flops

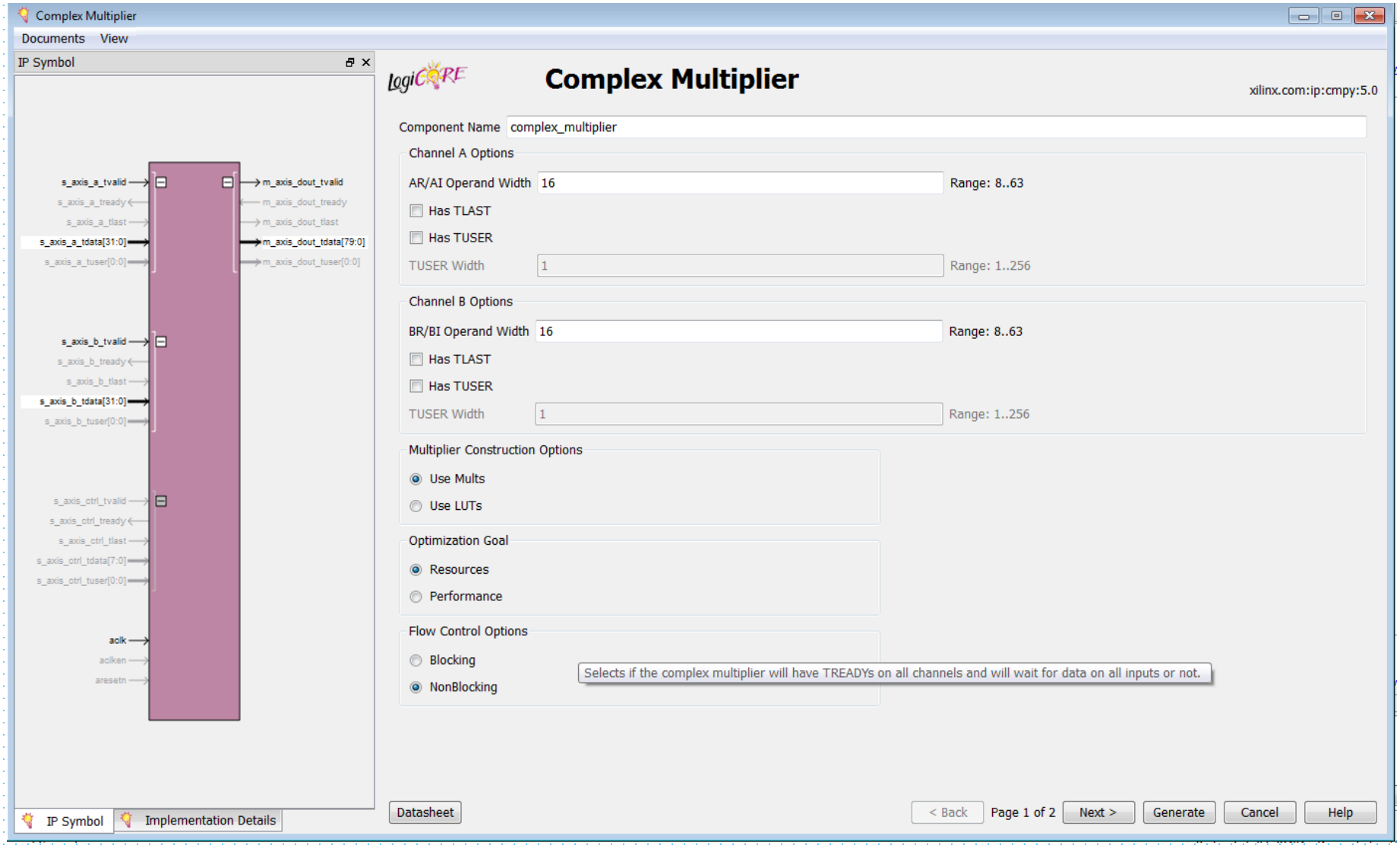
Adder/Subtracter with Core Generator



Divider with Core Generator



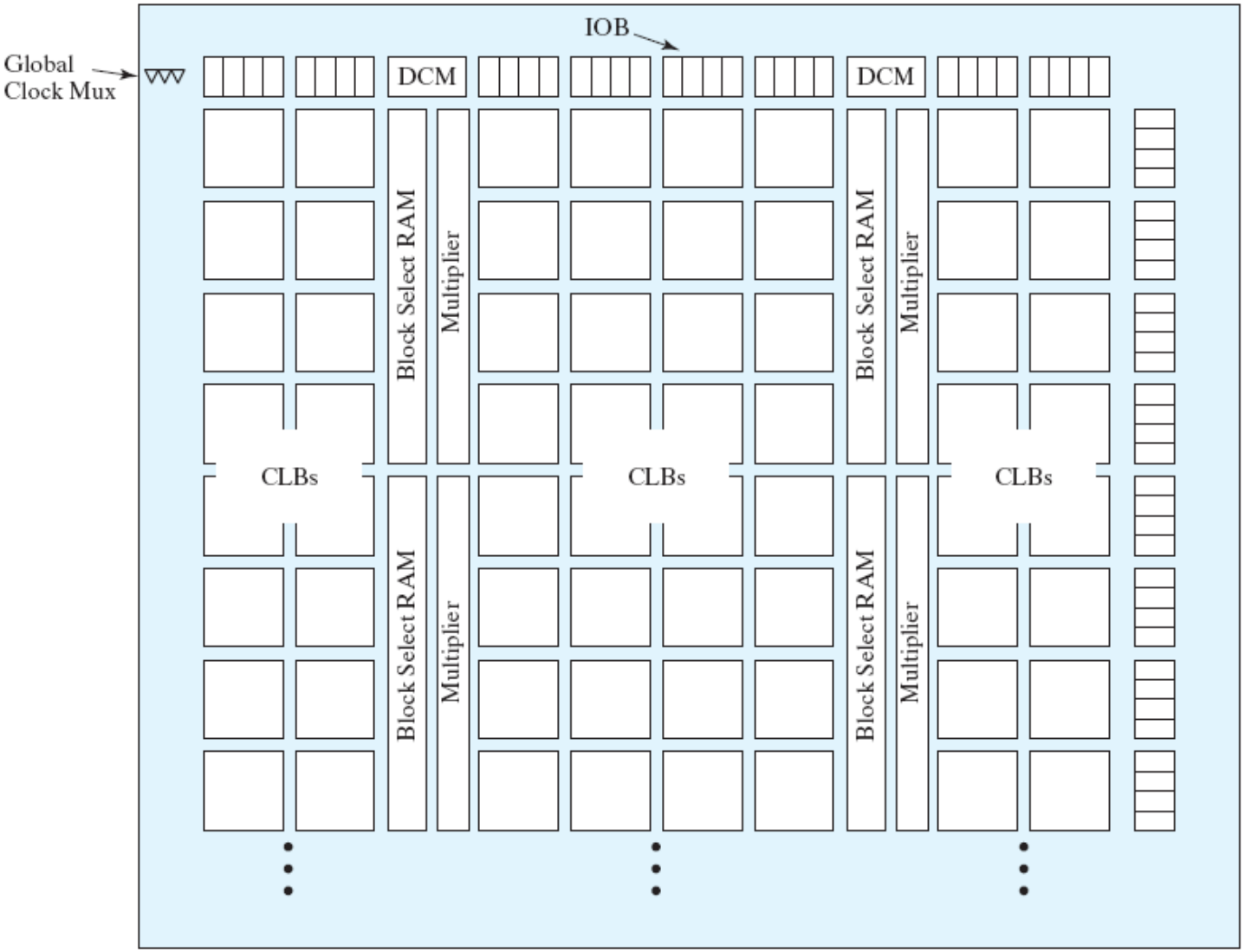
Multiplier with Core Generator



Xilinx Virtex FPGAs

- Leading edge of Xilinx technology
  - 65 nm technology
  - 1 V operating voltage
  - Up to 330,000 logic cells
  - Over 200,000 internal flip-flops
  - 10 Mb of block RAM
  - Hardwired units: multipliers, DSP units, microprocessors (powerPC)
  - 550-MHz clock technology

Xilinx Virtex FPGAs

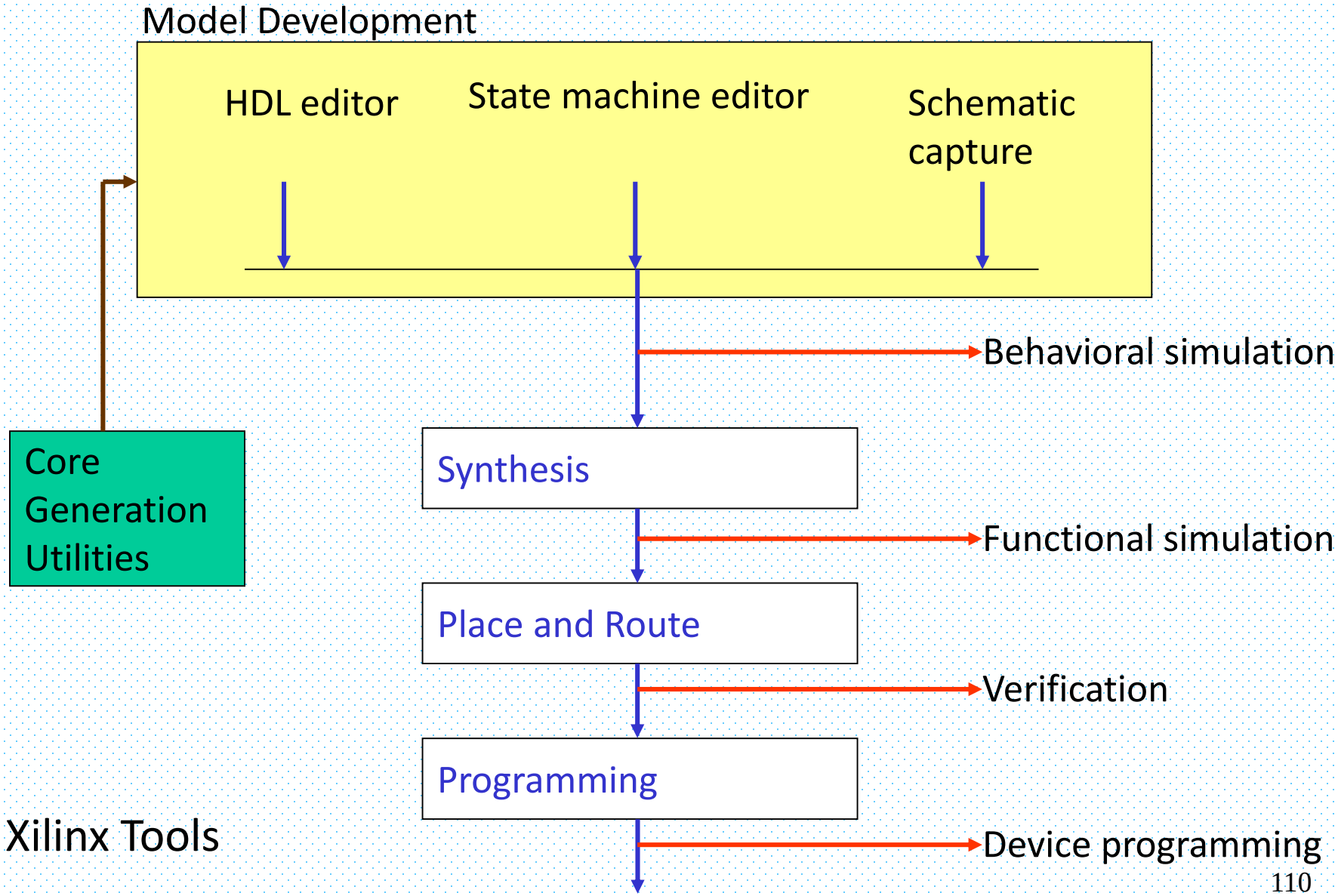


DCM: Clock Manager

Design with Programmable Devices

- Requires CAD tools
- Entry tools: entering a design
  - schematic entry package
  - FSM (finite state machine)
  - Hardware description languages (HDL)
    - VHDL, Verilog, ABEL,
- Synthesis tools
  - allocate
  - map
  - configure
  - connect logic blocks

FPGA Design Flow 1/4



FPGA Design Flow 2/4

- Model development:
  - VHDL code
  - State-machines may be described in a graphical manner and translated into VHDL code.
  - Traditional schematic capture can be translated into VHDL source.
- Behavioral Simulation
  - Before synthesis; for testing functional correctness
- Synthesis
  - The design is synthesized to a library of primitive components such as gates, flip-flops, and latches
- Functional Simulation
  - To find out preliminary performance estimates
  - For example, timing information can be obtained from known properties of FPGA components
  - Still not too accurate

FPGA Design Flow 3/4

- Place and Route:
  - The design is mapped to the primitives in the target chip
  - In FPGA, there are function generators (LUTs), flip-flops, and latches
  - Each primitive must be assigned to a specific CLB (Placement)
  - Connections between CLBs that implement the primitives must be established (routing)
  - Accurate timing can be obtained in Verification step (Post-placement and routing simulation)
  - The configuration bits are generated.

FPGA Design Flow 4/4

- Programming:
  - The configuration data (bit stream) is finally loaded into the target FPGA chip.

- These steps are fairly generic although the terminology used here is adopted from Xilinx.

Xilinx Tools: Design Flow