Chapter 10: Design Options of Digital Systems

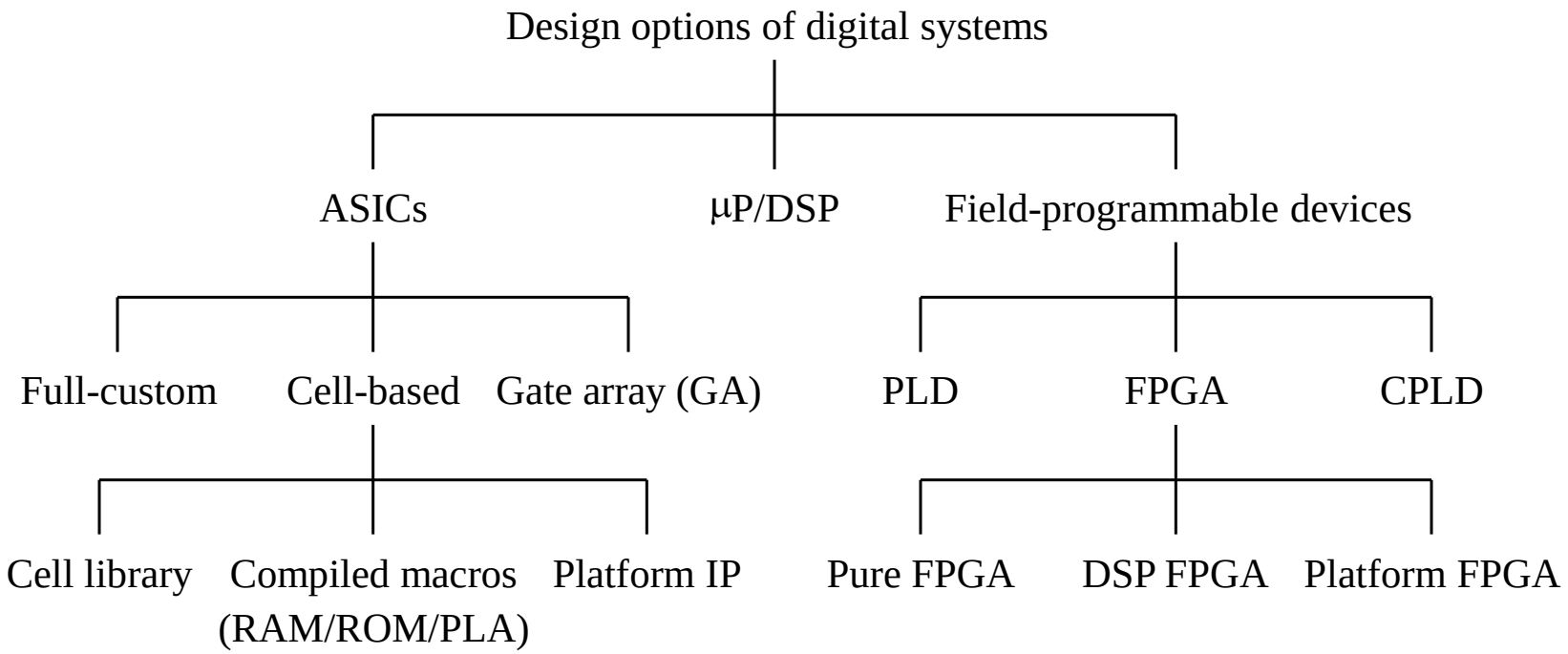
Prof. Soo-Ik Chae

Objectives

After completing this chapter, you will be able to:

- Describe the features of system-level design options
- Describe basic structures and features of cell-based ASICs
- Describe basic structures and features of gate array ASICs
- Describe basic structures and features of programmable logic devices (PLDs)
- Describe basic structures and features of field-programmable gate arrays (FPGAs)
- Understand how to model PLAs (including PLDs)
- Understand the issues of voltage tolerance and voltage compliance

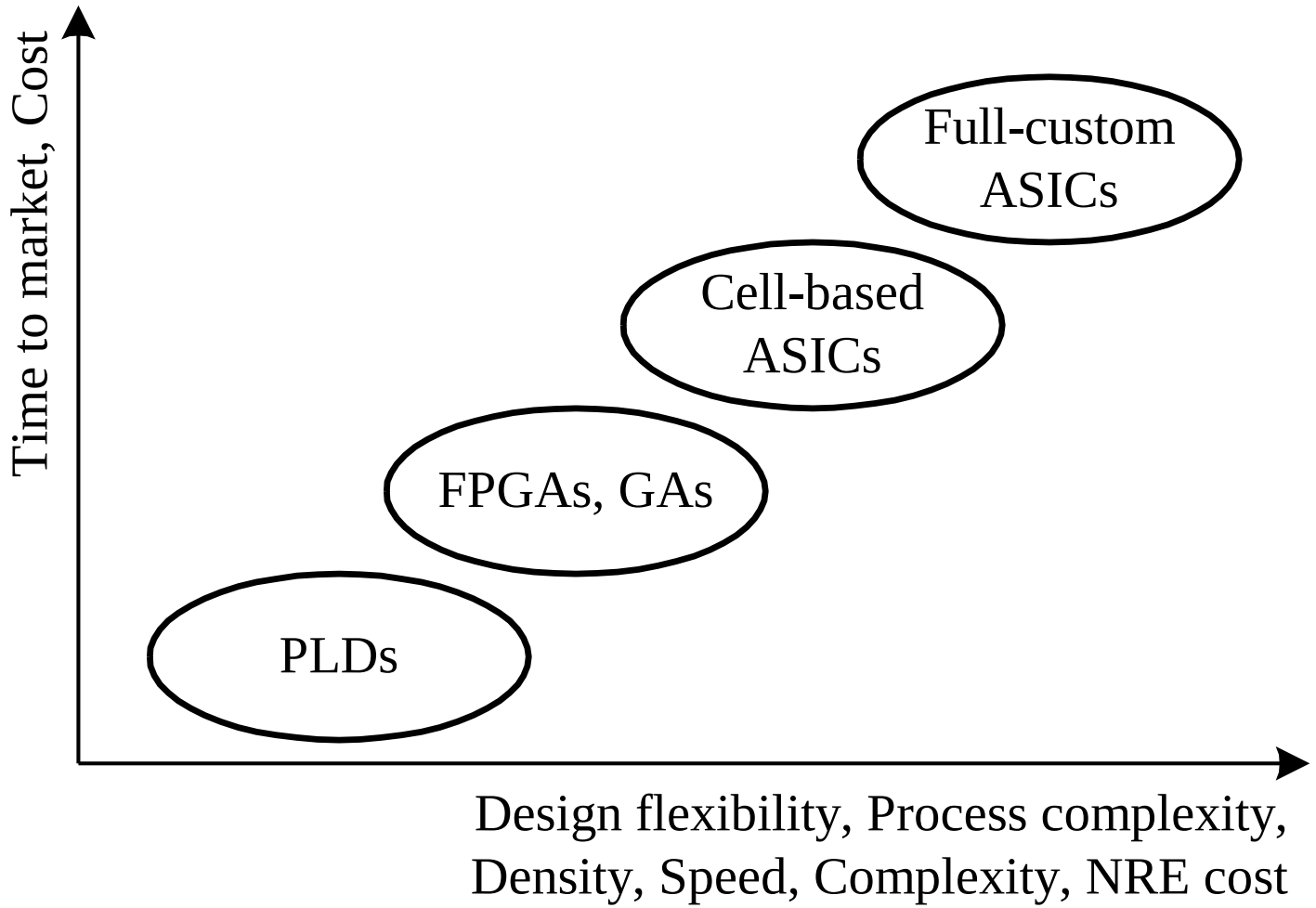
Implementation Options of Digital Systems



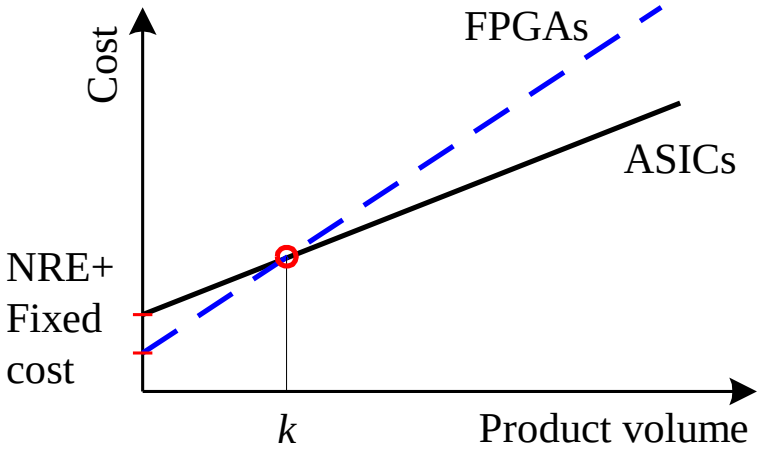
System-Level Design Options

- PCB (printed-circuit board)-based platforms
  - They consist of discrete standard components such as μp/μC, peripherals, and memory devices.
  - NRE cost is low but final product may cost too much to be accepted.
  - It might need additional CPLD/FPGA to customize the required logic.
- SoPC (System on a programmable chip)-based platforms
  - They consist of hard or soft IP being configured into a system required for a given application.
  - It might need additional customized logic modules.
  - Examples are Xilinx VirtexII pro, Spartan 3 (MicroBlaze), and Altera Cyclone Series (NIOS).
- Soft IP+ cell_library-based platforms
  - They consist of soft IP being configured into an optimized system hardware.
  - An example is Tensilica's Xtensa.

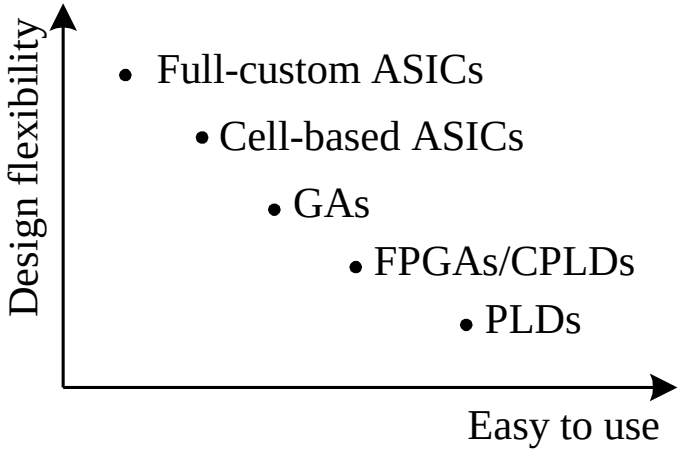
Design Alternatives of Digital Systems



Comparison of Design Options

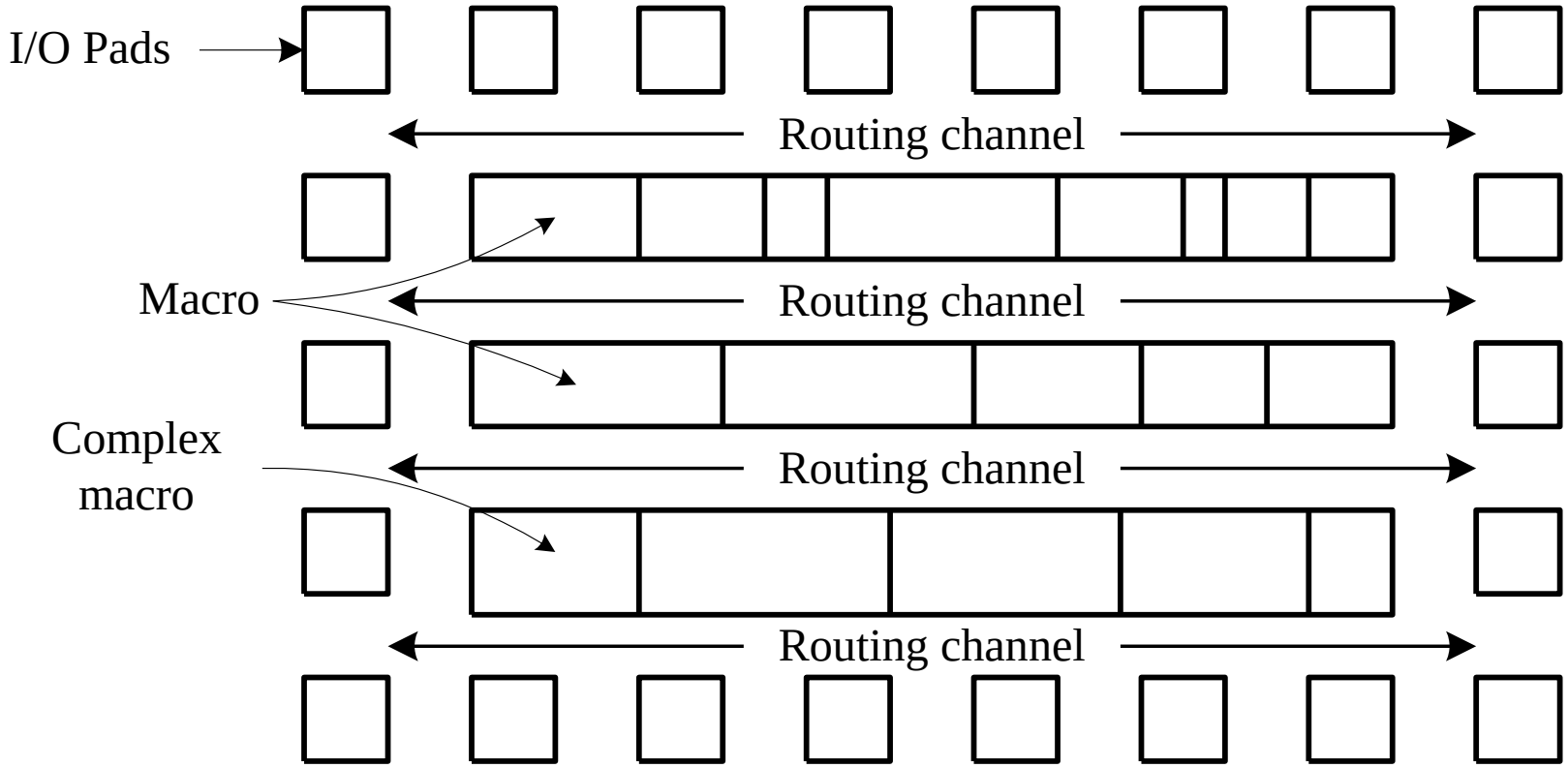


(a) Cost versus product volume



(b) Comparison of various digital system design options

Basic Structure of Cell-Based ASICs

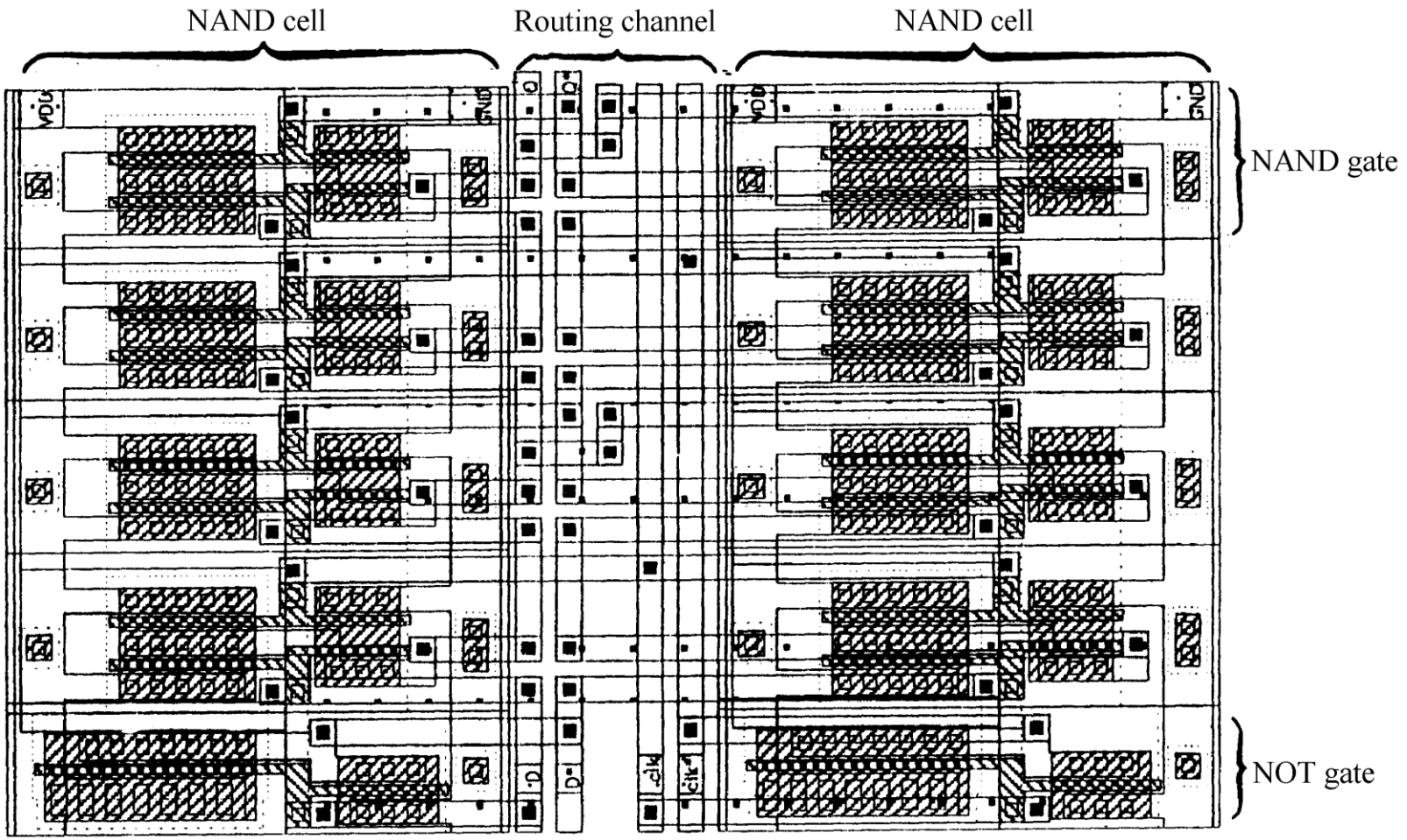


Basic Cell Types

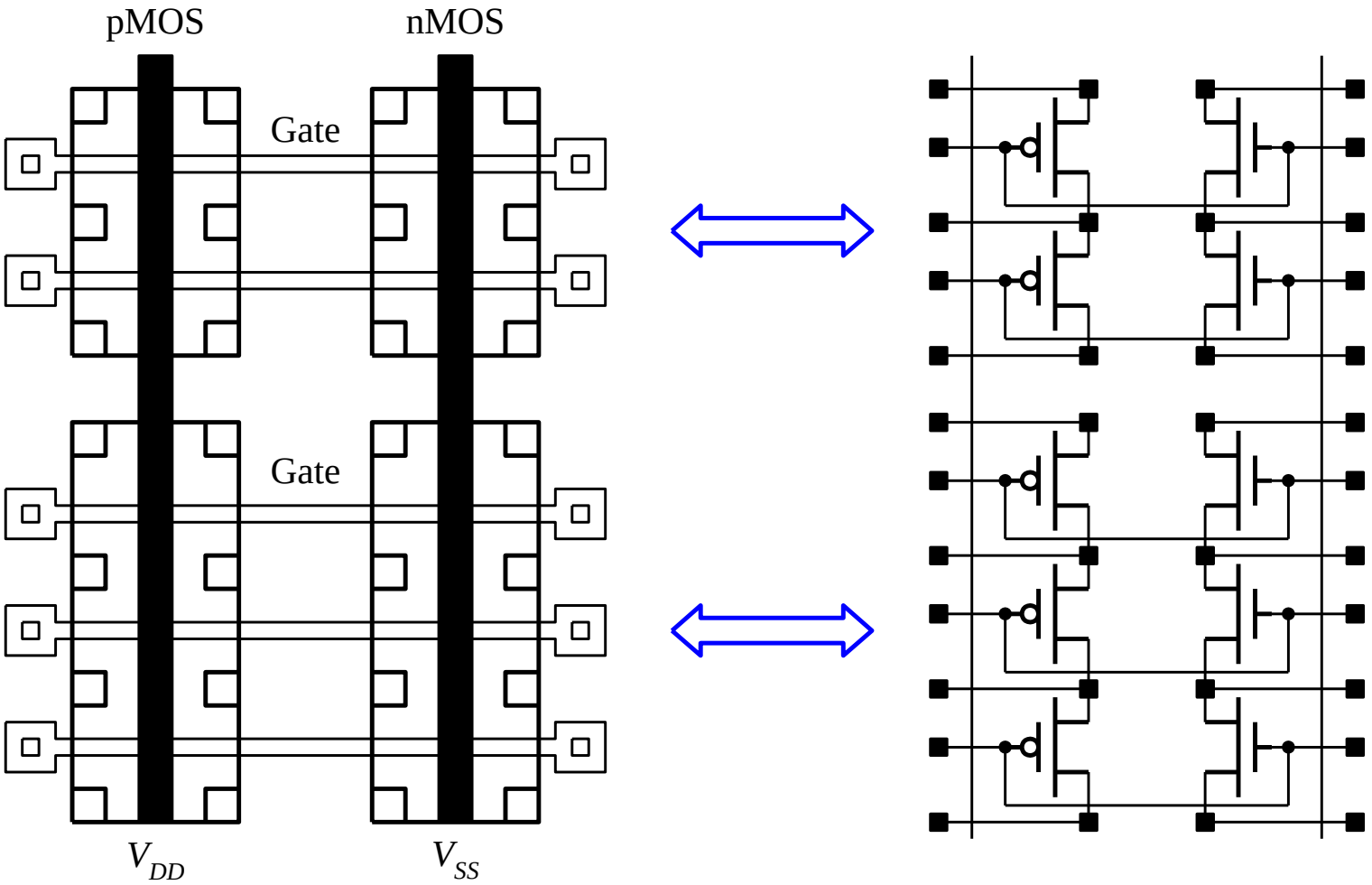
Standard cell types	Variations
Inverter/buffer/tristate buffer	1X, 2X, 4X, 8X, 16X
NAND/AND gate	2 ~ 8 inputs
NOR/OR gate	2 ~ 8 inputs
XOR/XNOR gate	2 ~ 8 inputs
MUX/DeMUX	2 ~ 8 inputs (inverted/noninverted output)
Encoder/Decoder	4 ~ 16 inputs (inverted/noninverted output)
Schmitt trigger circuit	Inverted/noninverted output
Latch/register/counter	D/JK(sync./async. clear/reset)
I/O pad circuits	Input/Output(tristate/bidirectional)

A Cell-Based D-Type Flip Flop

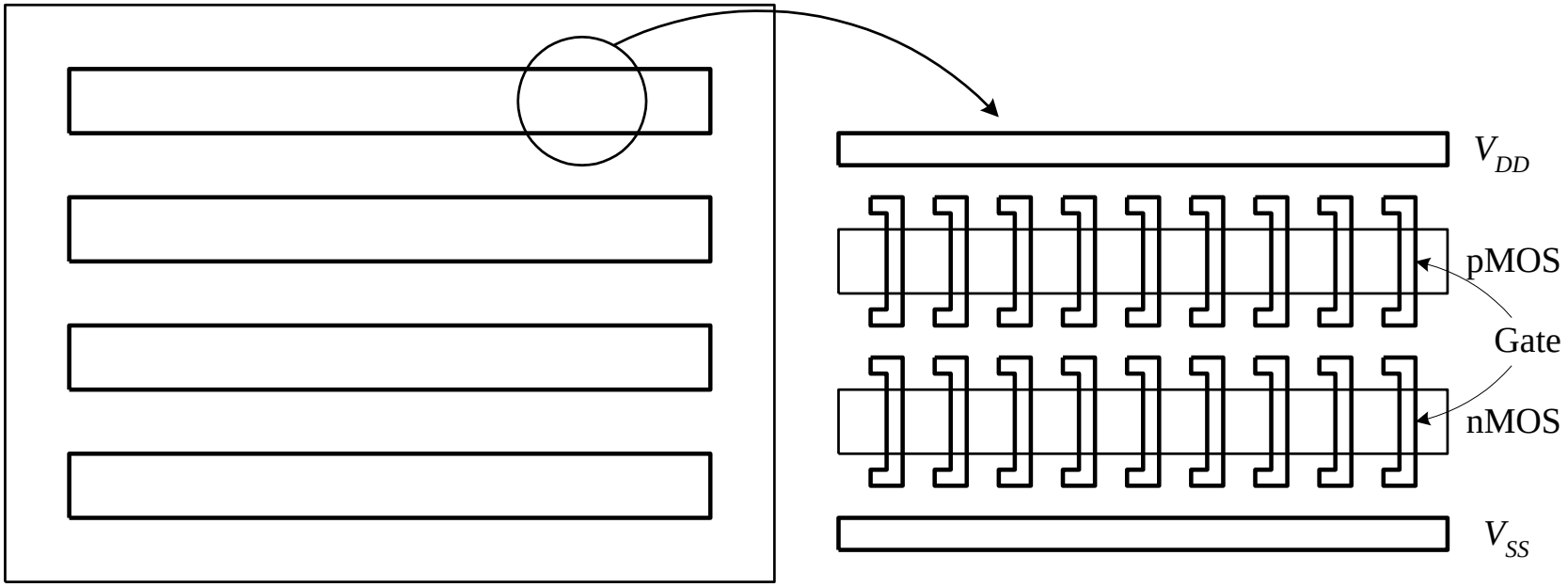
Abutting: fixed height



Basic Structures of Gate Arrays

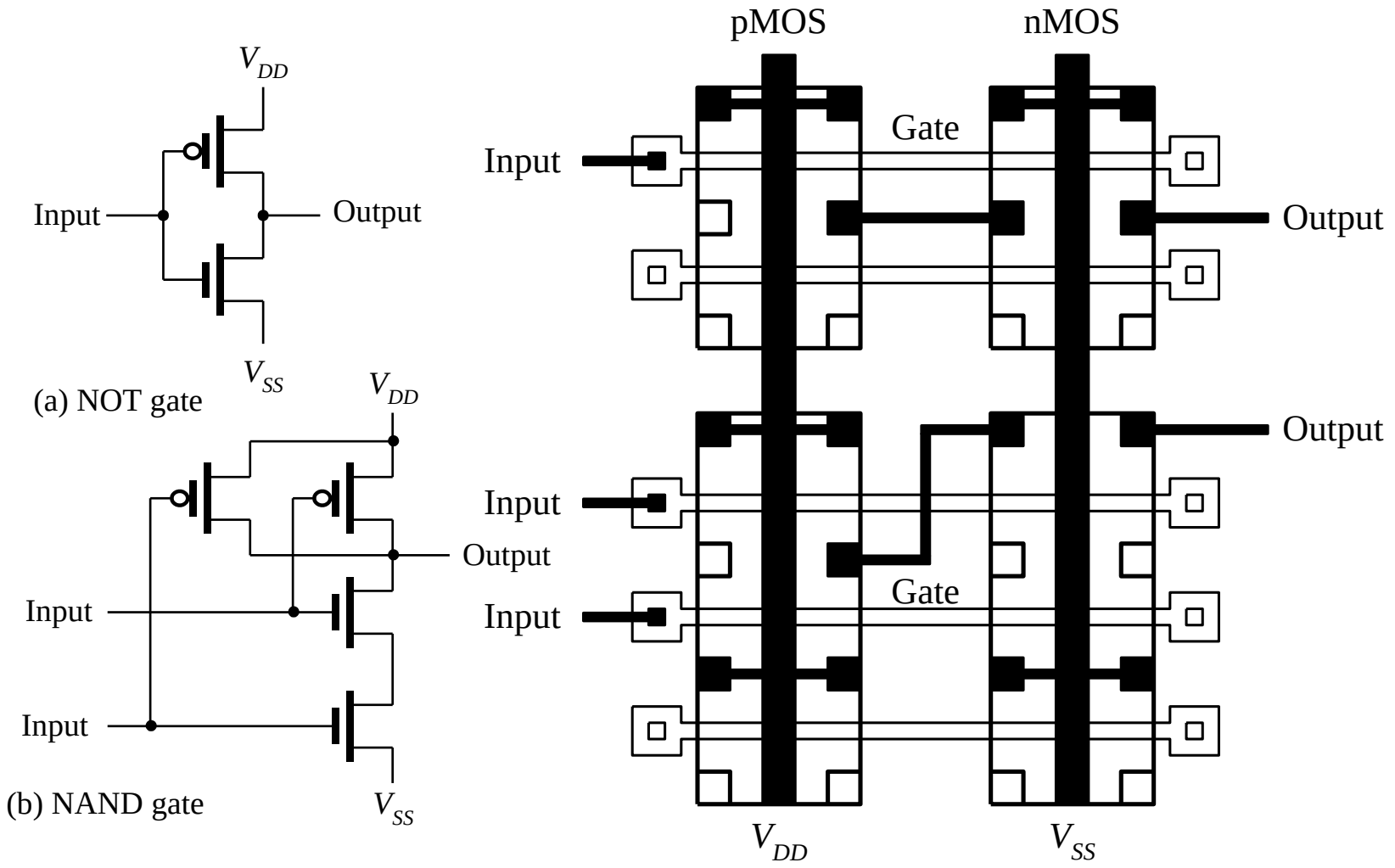


Basic Structures of Gate-Array ASICs

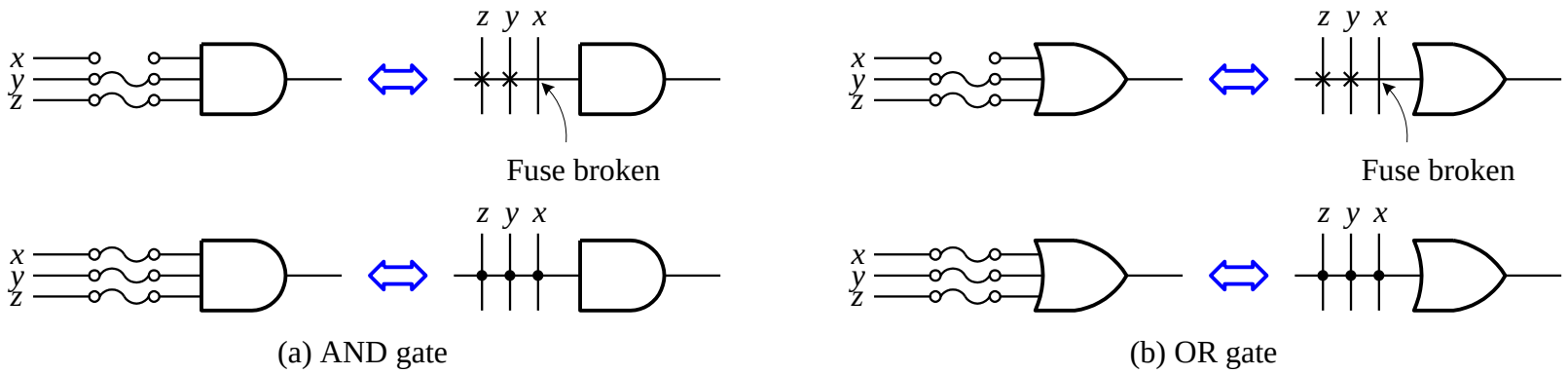


Sea of gates? No routing channels

Basic Structures of Gate Arrays

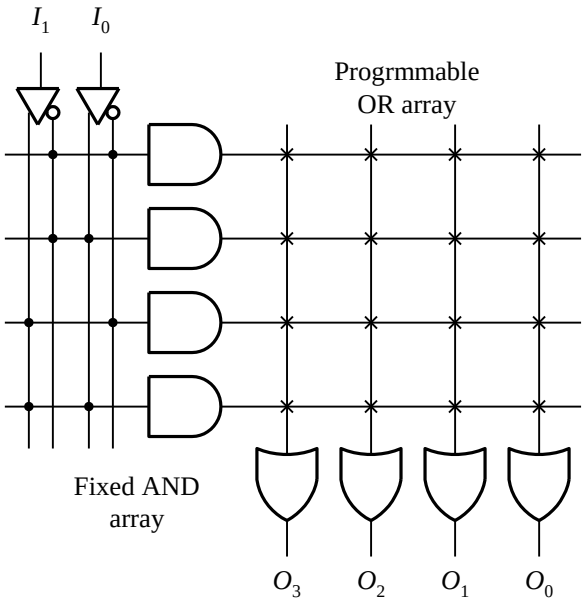


Basic Structures of Programmable Logic Devices

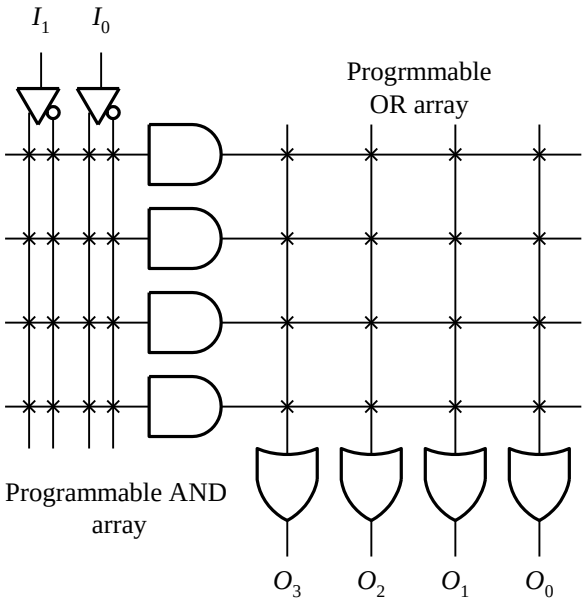


Shorthand notations for describing the structures of PLDs

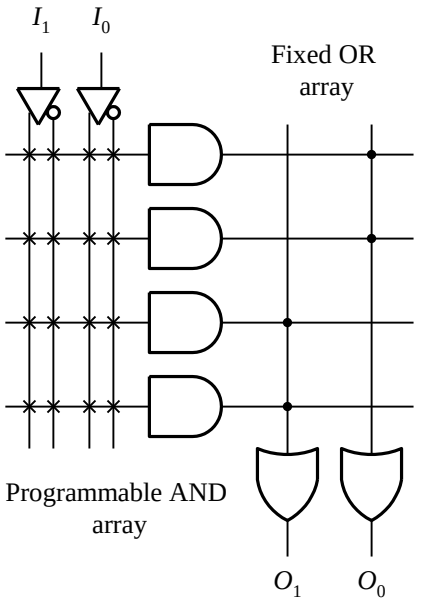
Basic Structures of Programmable Logic Devices



(a) ROM structure

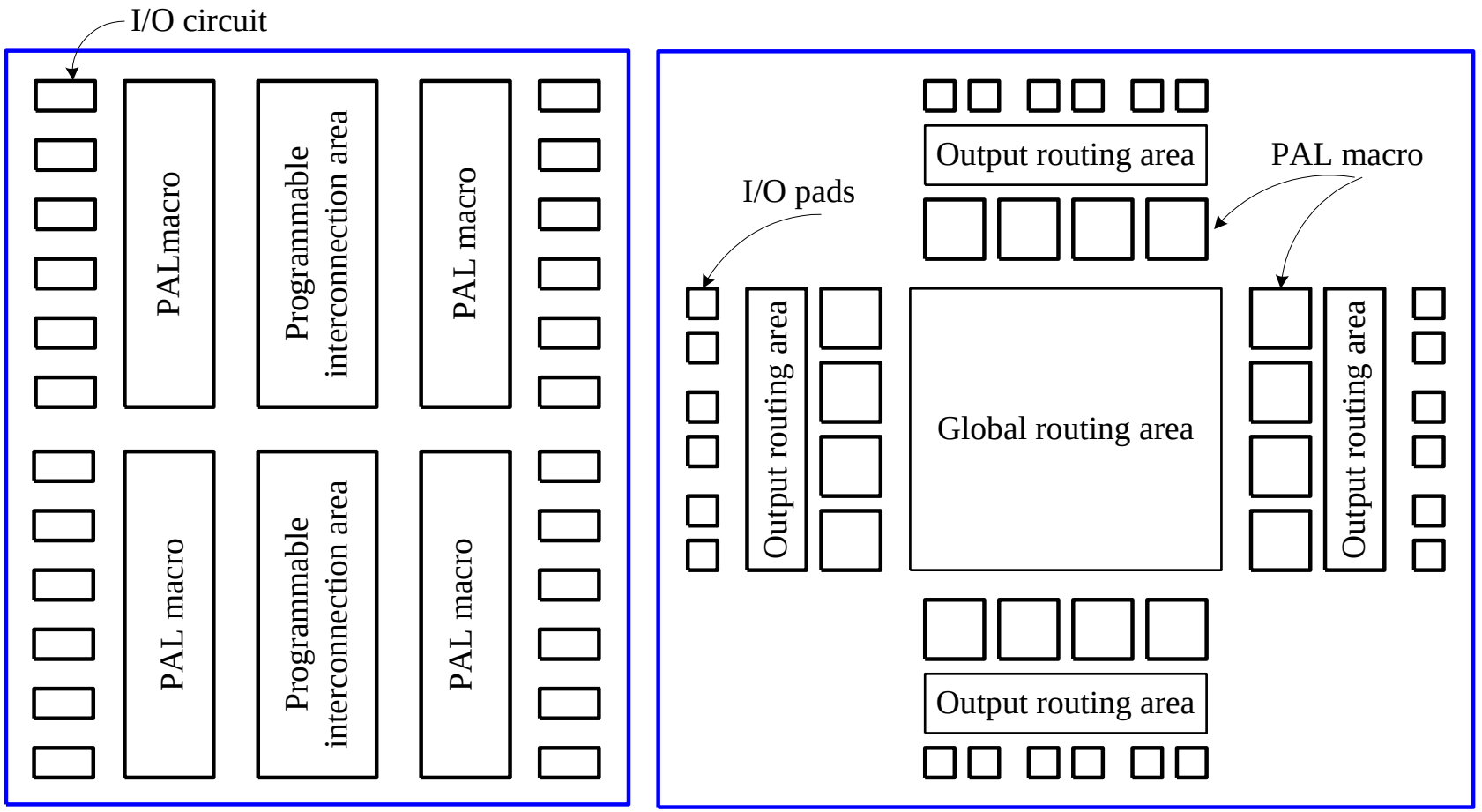


(b) PLA structure



(c) PAL structure

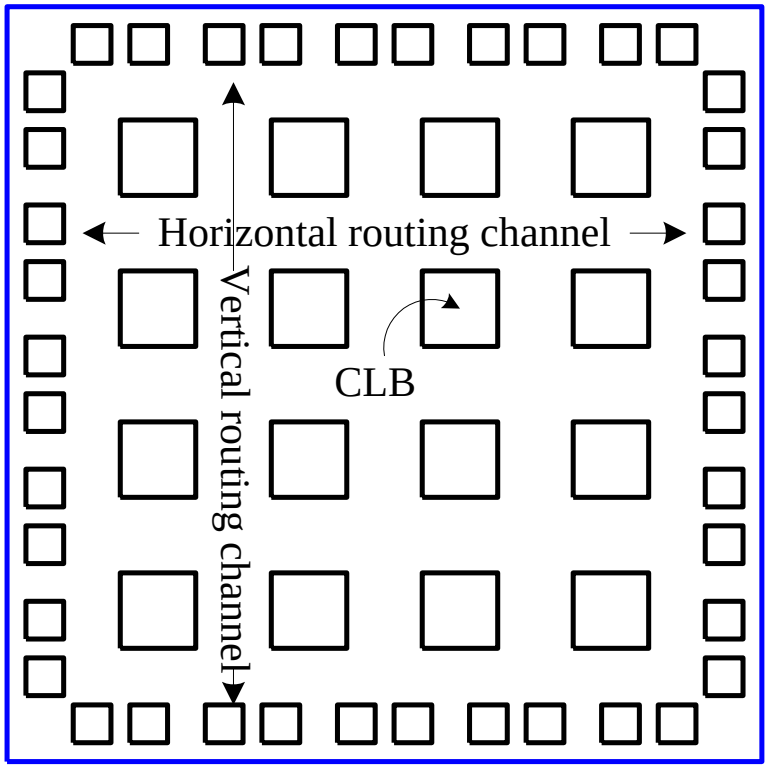
Basic Structures of CPLDs



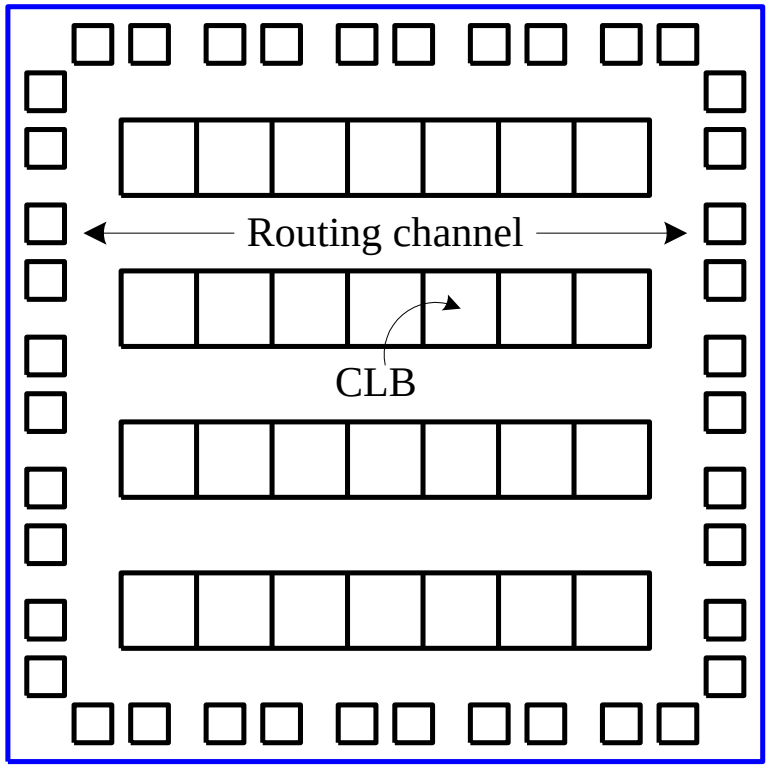
(a) CPLD basic structure

(b) pLSI basic structure

Basic Structures of FPGAs

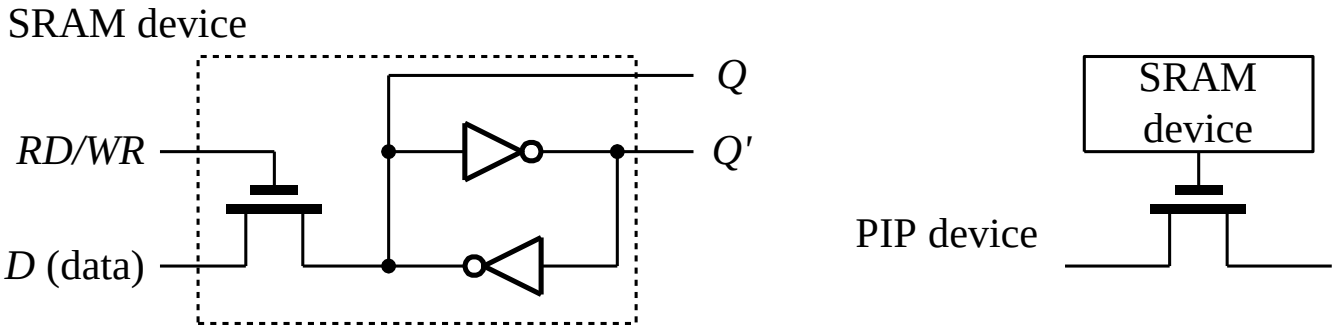


(a) Matrix type

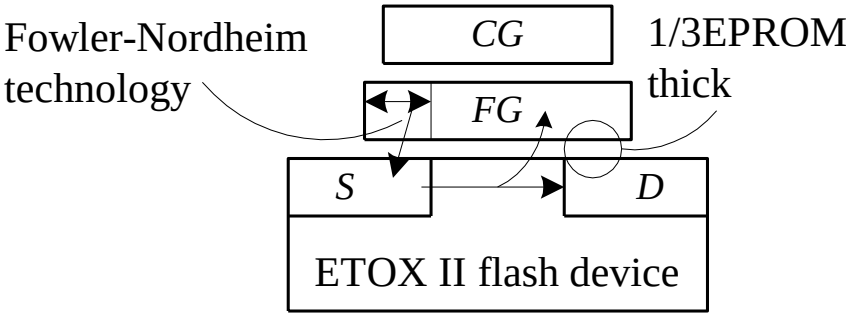


(b) Row type

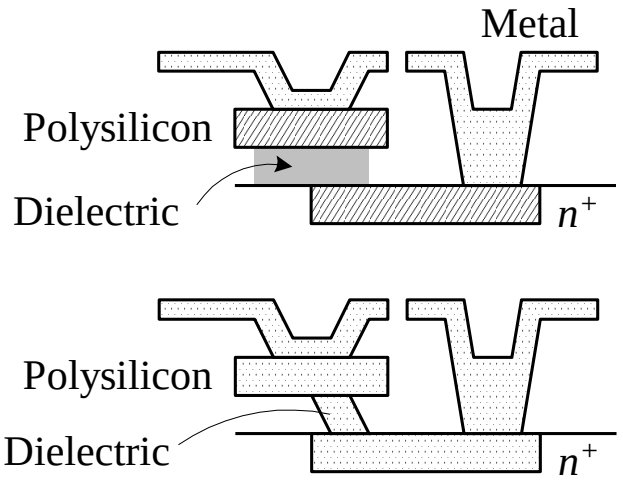
Basic Structures of Programmable Interconnections



(a) SRAM device



(b) Flash device

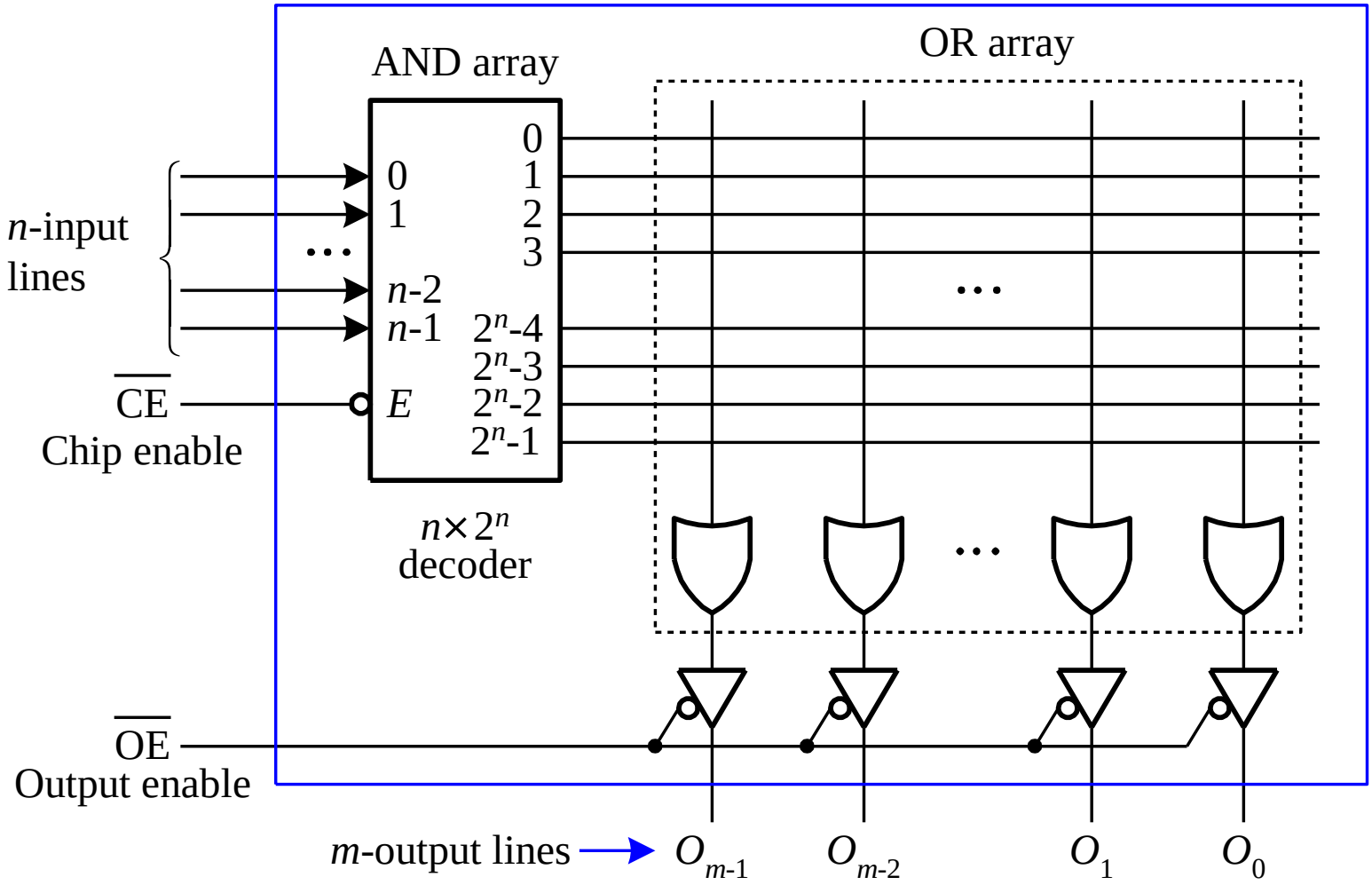


(c) Antifuse device

Comparison of programmable interconnections

	SRAM	Flash	Antifuse
Process technology	CMOS	Standard 2-level polysilicon	New type polysilicon
Programming approach	Shift register	FAMOS	Avalanche
Area	Very large	Large	Small
Capacitance	≈ 2 kΩ	≈ 2 kΩ	≈ 500 Ω
Resistance	≈ 50 fF	≈ 15 fF	≈ 5 fF

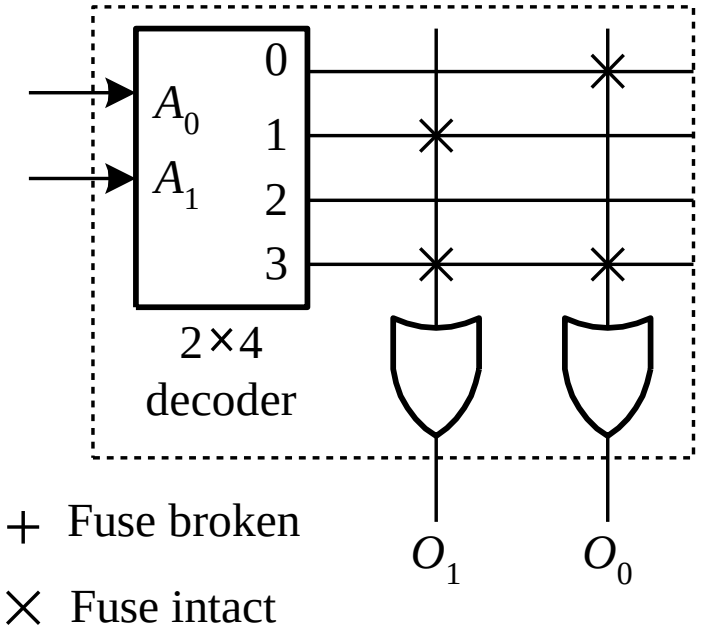
Basic Structures of ROMs



Basic Applications of ROMs

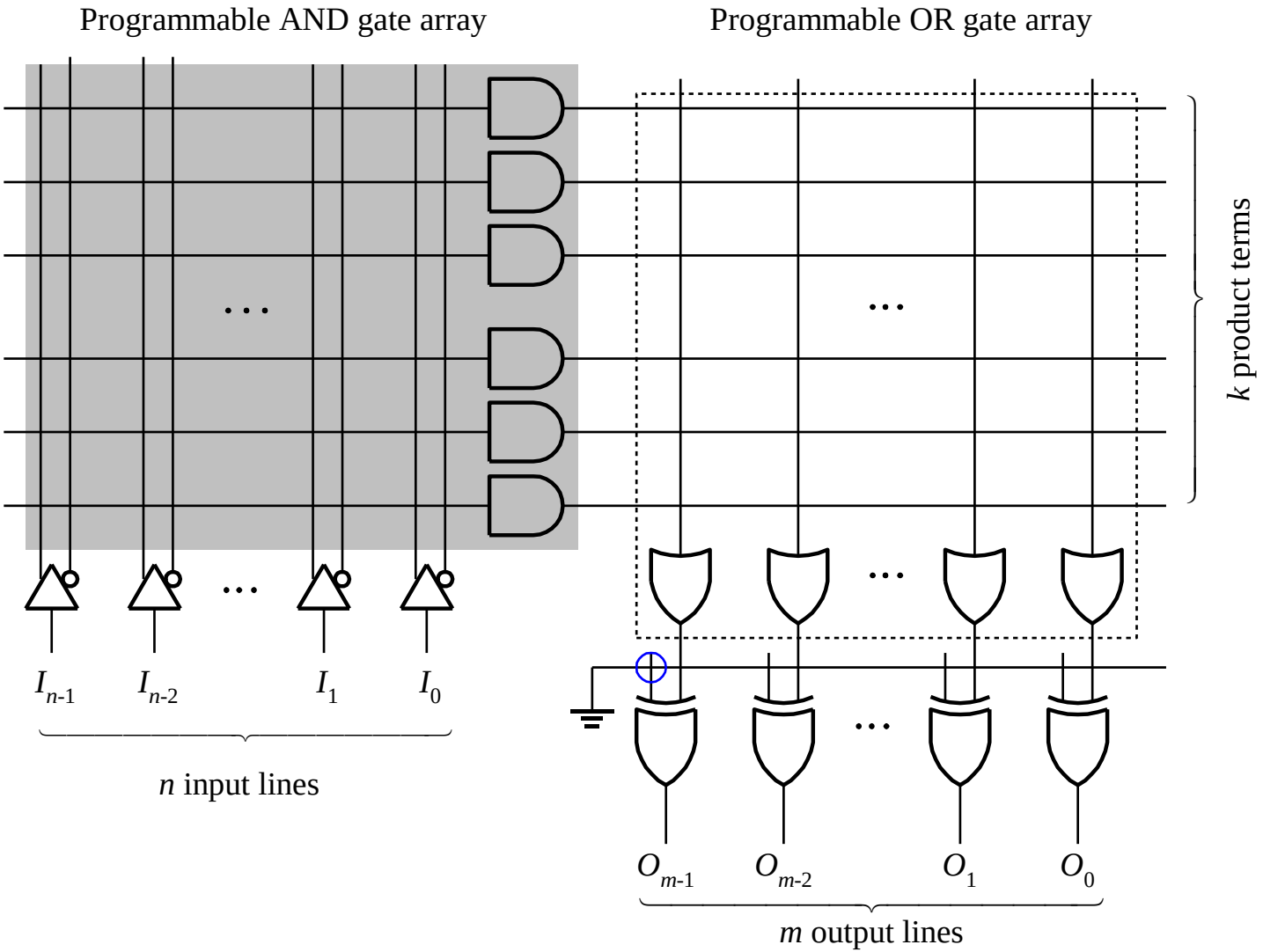
Input		Output	
A_1	A_0	O_1	O_0
0	0	0	1
0	1	1	0
1	0	0	0
1	1	1	1

(a) Truth table



(b) Logic circuit

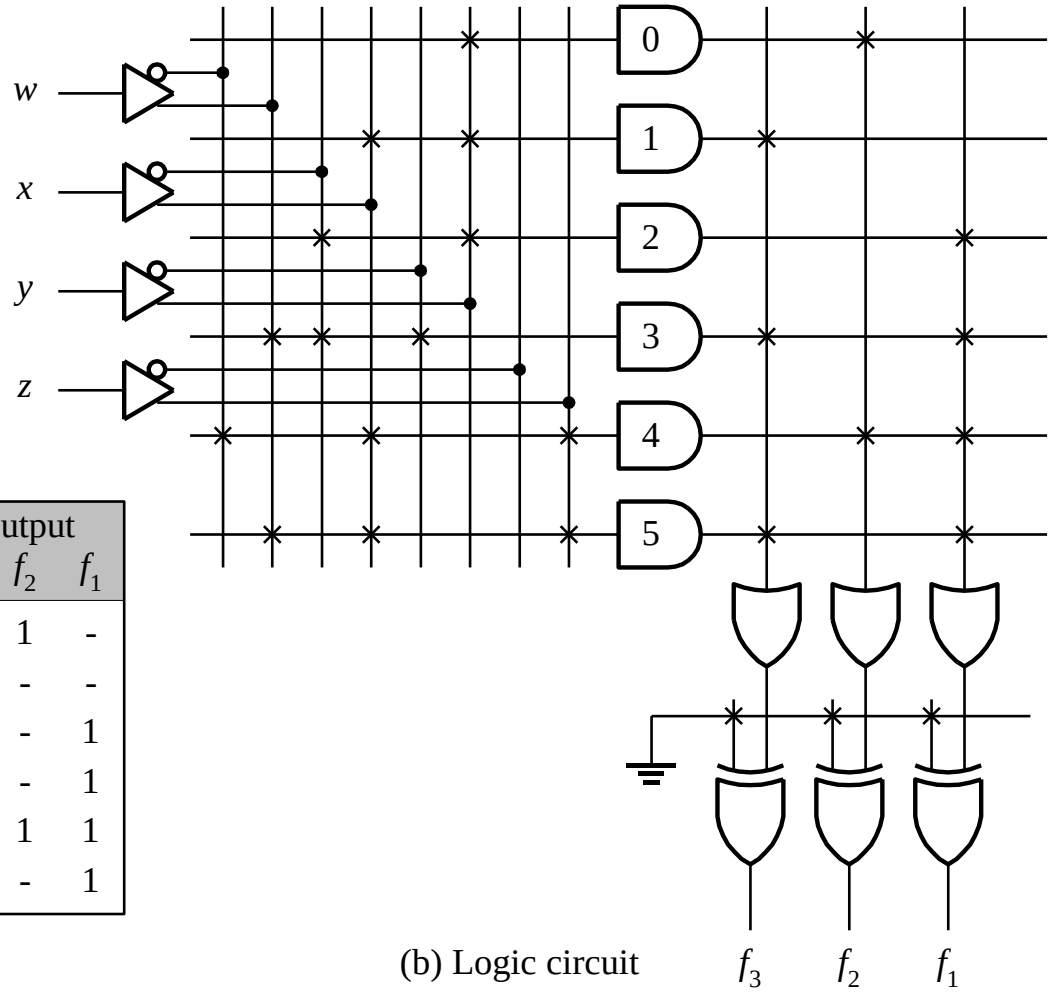
Basic Structures of PLAs



Basic Applications of PLAs

Product term		Input w	x	y	z	Output f_3	f_2	f_1
y	0	-	-	1	-	-	1	-
xy	1	-	1	1	-	1	-	-
$x'y$	2	-	0	1	-	-	-	1
$wx'y'$	3	1	0	0	-	1	-	1
$w'xz$	4	0	1	-	1	-	1	1
wxz	5	1	1	-	1	1	-	1

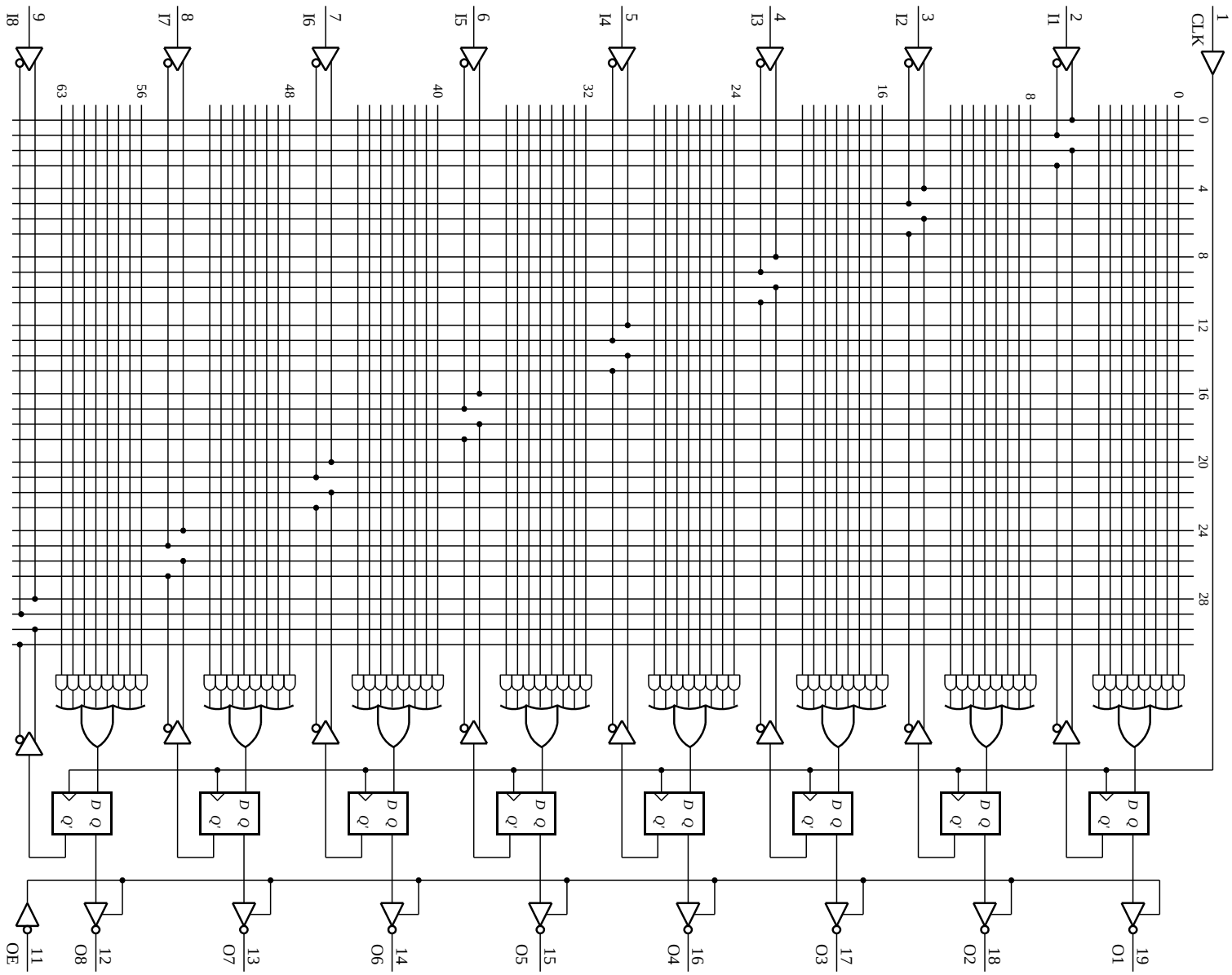
(a) PLA programming table



(b) Logic circuit

Basic Structures of PALs

PAL devices	Package pins	AND-gate inputs	Primary inputs	Bidirectional I/Os	Registered outputs	Combinational outputs
PAL16L8	20	16	10	6	0	2
PAL16R4	20	16	8	4	4	0
PAL16R6	20	16	8	2	6	0
PAL16R8	20	16	8	0	8	0
PAL20L8	24	20	14	6	0	2
PAL20R4	24	20	12	4	4	0
PAL20R6	24	20	12	2	6	0
PAL20R8	24	20	12	0	8	0


I8 9
I7 8
I6 7
I5 6
I4 5
I3 4
I2 3
I1 2
CLK 1
63
56
48
40
32
24
16
8
0
0
4
8
12
16
20
24
28
D Q
Q'
OE 11
O8 12
O7 13
O6 14
O5 15
O4 16
O3 17
O2 18
O1 19

PLA Modeling

- PLA device structures can be one of the following types:
  - SOP (sum of product) are composed of an AND plane and an OR plane.
    - AND plus OR array.
    - NAND plus NAND array.
  - POS (product of sum) are composed of an OR plane and an AND plane.
    - OR plus AND array.
    - NOR plus NOR array.
- An AND plane or an OR plane is also called a personality array (or memory).

PLA Modeling (p.303 in LRM)

- A PLA device
  - is modeled by a group of system tasks.
  - is modeled by an appropriate combination of two system tasks.
- A PLA device can be modeled as:
  - asynchronous: the evaluations are executed whenever any input is changed.
  - synchronous: the evaluation time can be controlled in a predictable way.
- The output terms are updated without any delay.

PLA Modeling

```
$async$logic$format(mem_id, input_terms, output_terms);
$sync$logic$format(mem_id, input_terms, output_terms);
```

- PLA modeling system tasks
  - input_terms: nets or variables
  - output_terms: variables
  - logic: and | or | nand | nor

Array or Plane formats (17.5.4 in LRM)

- Array format: only 1 or 0 is used to denote whether inputs is taken or not
- Plane format: complies with Espresso
  - 1 or 0: true or complement value is taken
  - x: denote that the worst-case input is taken
  - z, ?: denote don't care

Array format		Plane format	
Asynchronous	Synchronous	Asynchronous	Synchronous
$async$and$array	$sync$and$array	$async$and$plane	$sync$and$plane
$async$nand$array	$sync$nand$array	$async$nand$plane	$sync$nand$plane
$async$or$array	$sync$or$array	$async$or$plane	$sync$or$plane
$async$nor$array	$sync$nor$array	$async$nor$plane	$sync$nor$plane

PLA Modeling

```
$async$logic$array(mem_type, input_terms, output_terms);
$sync$logic$array(mem_type, input_terms, output_terms);
```

```
wire a1, a2, a3, a4, a5, a6, a7;
reg b1, b2, b3;
wire [1:7] awire;
reg [1:3] breg;
reg [1:7] mema [1:3]; // define personality memory
   // asynchronous AND plane
   $async$and$array(mema, {a1, a2, a3, a4, a5, a6, a7}, {b1, b2, b3});
   // or using the following statement
   $async$and$array(mema, awire, breg);
   // synchronous AND plane
   forever @(posedge clock)
      $sync$and$array(mema, {a1, a2, a3, a4, a5, a6, a7}, {b1, b2, b3});
```

PLA Modeling

- Logic array personality is declared as an array of regs.
  - The width is equal to the number of input terms.
  - The depth is equal to the number of output terms.
- Logic array personality is loaded into memory using
  - system tasks $readmemb or $readmemh.
  - the procedural assignment statements.
- Logic array personality can be changed dynamically during simulation.

PLA Modeling

- The array format
  - uses only 1 or 0 to denote the input value is taken or not.
- The plane format (complies with Espresso)
  - 1 or 0 : the true or complement input value is taken.
  - x : the worst case input value is taken.
  - ? or z: don't care.

PLA Modeling

- Define PLA personality from file.
  - In this example only one and plane is considered.

```
// an example of the usage of the array format.
module async_array(a1, a2, a3, a4, a5, a6, a7, b1, b2, b3);
input   a1, a2, a3, a4, a5, a6, a7 ;
output b1, b2, b3;
reg     [1:7] mem[1:3];   // memory declaration for array personality
reg     b1, b2, b3;
initial begin
// setup the personality from the file array.dat
$readmemb ("array.dat", mem);
// setup an asynchronous logic array with the input
// and output terms expressed as concatenations
$async$and$array (mem,{a1, a2, a3, a4, a5, a6, a7},{b1, b2, b3});
end
endmodule
```

b1 = a1 & a2
b2 = a3 & a4 & a5
b3 = a5 & a6 & a7

```
1100000
0011100
0000111
```

PLA Modeling

```
module sync_array(a0, a1, a2, a3, a4, a5, a6, a7, b0, b1, b2);
input    a0, a1, a2, a3, a4, a5, a6, a7;
output   b0, b1, b2;
reg      b0, b1, b2;
reg [7:0] mem[0:2];
// an example of the usage of array format
initial begin   // using procedural assignment statements.
  $readmemb ("array.dat", mem);
  forever @(posedge clk)
      $async$and$array(mem, {a0,a1,a2,a3,a4,a5,a6,a7}, {b0,b1,b2});
  end
end
endmodule
```

PLA Modeling – example

```
module pla(a0, a1, a2, a3, a4, a5, a6, a7, b0, b1, b2);
input    a0, a1, a2, a3, a4, a5, a6, a7;
output   b0, b1, b2;
reg      b0, b1, b2;
reg [7:0] mem[0:2];
// an example of the usage of array format
initial begin   // using procedural assignment statements.
  mem[0] = 8'b11001100;
  mem[1] = 8'b00110011;
  mem[2] = 8'b00001111;
  $async$and$array(mem, {a0,a1,a2,a3,a4,a5,a6,a7},
                                {b0,b1,b2});
  end
endmodule
```

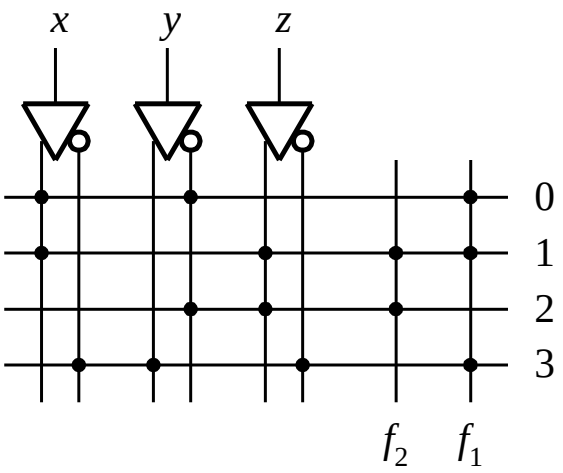
```
A = {a0, a1, a2, a3, a4, a5, a6, a7}
B = {b0, b1, b2}
mem[0] = 8'b11001100;
mem[1] = 8'b00110011;
mem[2] = 8'b00001111;

A = 11001100 -> B = 100
A = 00110011 -> B = 010
A = 00001111 -> B = 001
A = 10101010 -> B = 000
A = 01010101 -> B = 000
A = 11000000 -> B = 000
A = 00111111 -> B = 011
```

A PLA Modeling Example --- Logic Circuit

Product		Input x	y	z	Output f_2	f_1
xy'	0	1	0	-	-	1
xz	1	1	-	1	1	1
$y'z$	2	-	0	1	1	-
$x'yz'$	3	0	1	0	-	1

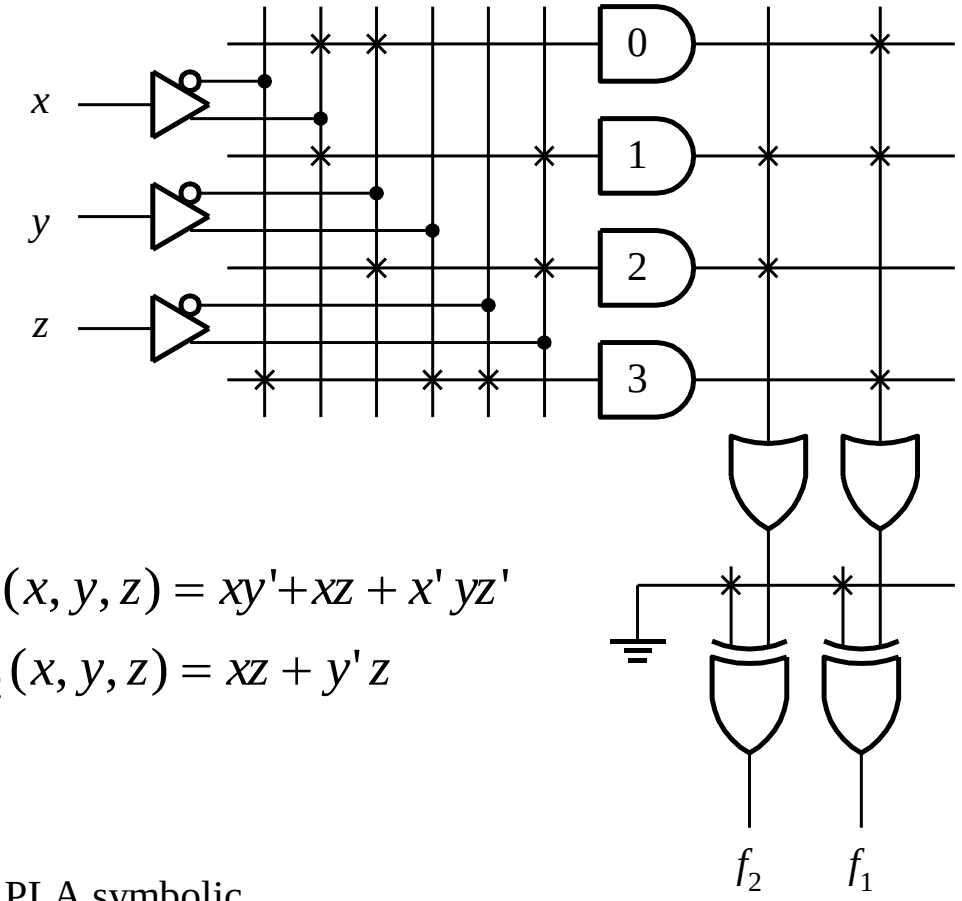
(a) PLA programming table



(b) PLA symbolic diagram

$$f_1(x, y, z) = xy' + xz + x'yz'$$

$$f_2(x, y, z) = xz + y'z$$



(c) logic circuit

A PLA Modeling Example – The Array Format

```
module pla_example(input x, y, z, output reg f1, f2);
// An example of the usage of the array format.
reg    p0, p1, p2, p3;   // internal minterms
reg    [0:5] mem_and[0:3];
reg    [0:3] mem_or[1:2];
wire  x_n, y_n, z_n;
assign x_n = ~x, y_n = ~y, z_n = ~z;
initial begin: pla_model_array
   mem_and[0] = 6'b100100;     // define AND plane
   mem_and[1] = 6'b100010;
   mem_and[2] = 6'b000110;
   mem_and[3] = 6'b011001;
   $async$and$array(mem_and, {x, x_n, y, y_n, z, z_n}, {p0, p1, p2, p3});
   mem_or[1] = 4'b1101;          // define OR plane
   mem_or[2] = 4'b0110;
   $async$or$array(mem_or, {p0, p1, p2, p3}, {f1, f2});
end
endmodule
```

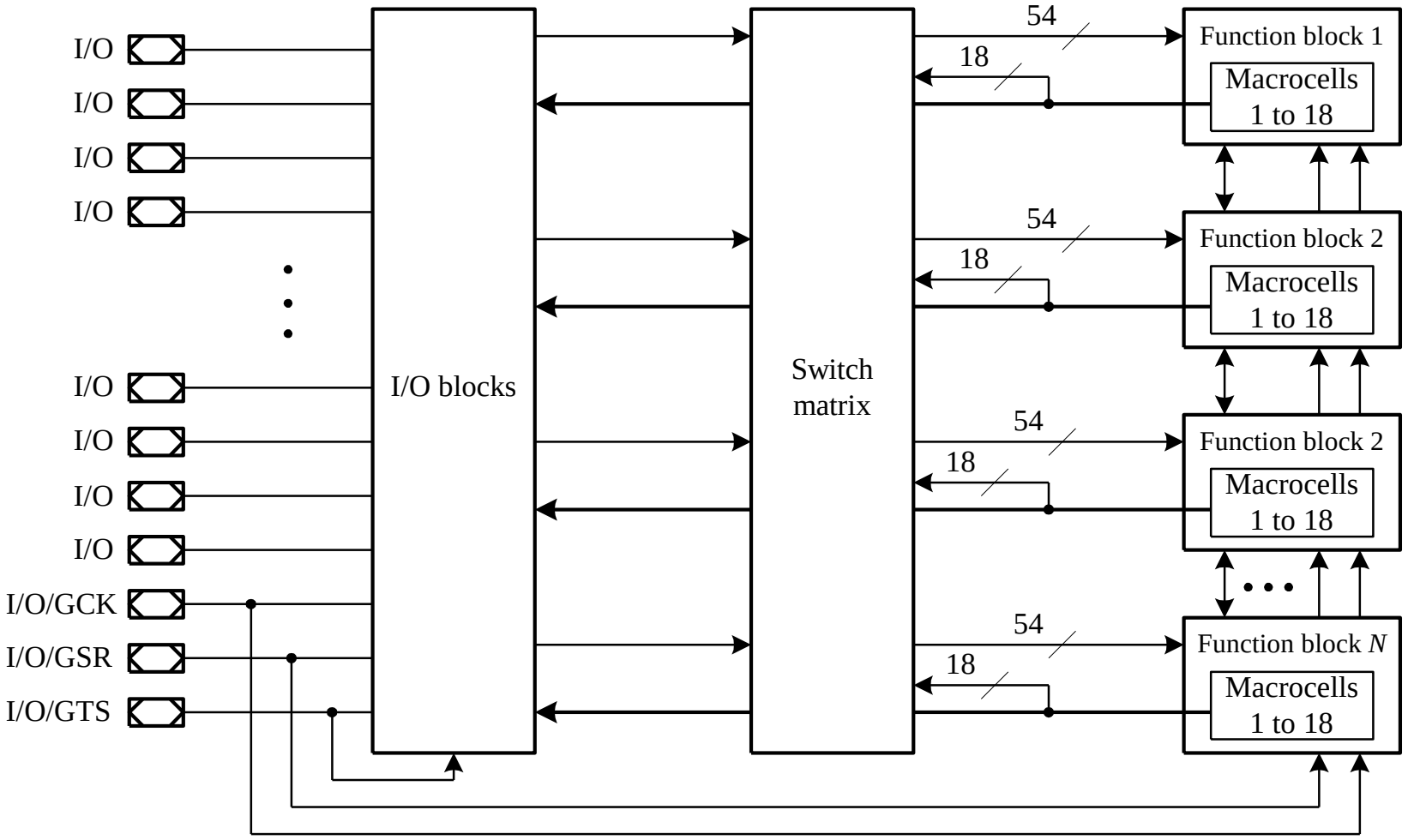
```
# 0 ns   x x x x x
# 5 ns   0 0 0 0 0
# 10 ns 0 0 1 0 1
# 15 ns 0 1 0 1 0
# 20 ns 0 1 1 0 0
# 25 ns 1 0 0 1 0
# 30 ns 1 0 1 1 1
# 35 ns 1 1 0 0 0
# 40 ns 1 1 1 1 1
```

A PLA Modeling Example – The Plane Format

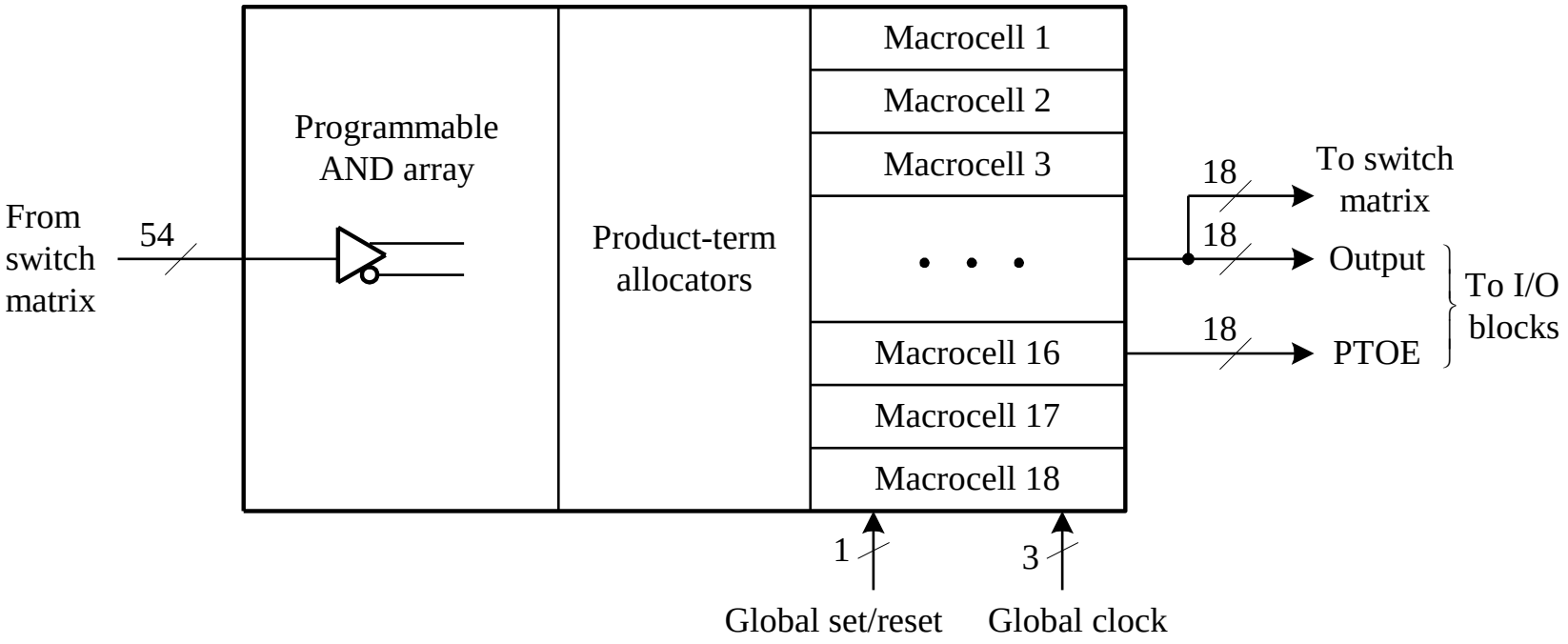
```
module pla_example(input x, y, z, output reg f1, f2);
reg    p0, p1, p2, p3;             // internal minterms
reg    [0:2] mem_and[0:3];   // and plane personality matrix
reg    [0:3] mem_or[1:2];     // or plane personality matrix
// An example of the usage of the plane format
initial begin: pla_model_plane
  $async$and$plane(mem_and, {x, y, z}, {p0, p1, p2, p3});
  mem_and[0] = 3'b10?;     // define AND plane
  mem_and[1] = 3'b1?1;
  mem_and[2] = 3'b?01;    // 000110 in array format
  mem_and[3] = 3'b010;
  $async$or$plane(mem_or, {p0, p1, p2, p3}, {f1, f2});
  mem_or[1] = 4'b11?1;     // define OR plane
  mem_or[2] = 4'b?11?;     // 0110 in array format
end
endmodule
```

```
# 0 ns   x x x x x
# 5 ns   0 0 0 0 0
# 10 ns 0 0 1 0 1
# 15 ns 0 1 0 1 0
# 20 ns 0 1 1 0 0
# 25 ns 1 0 0 1 0
# 30 ns 1 0 1 1 1
# 35 ns 1 1 0 0 0
# 40 ns 1 1 1 1 1
```

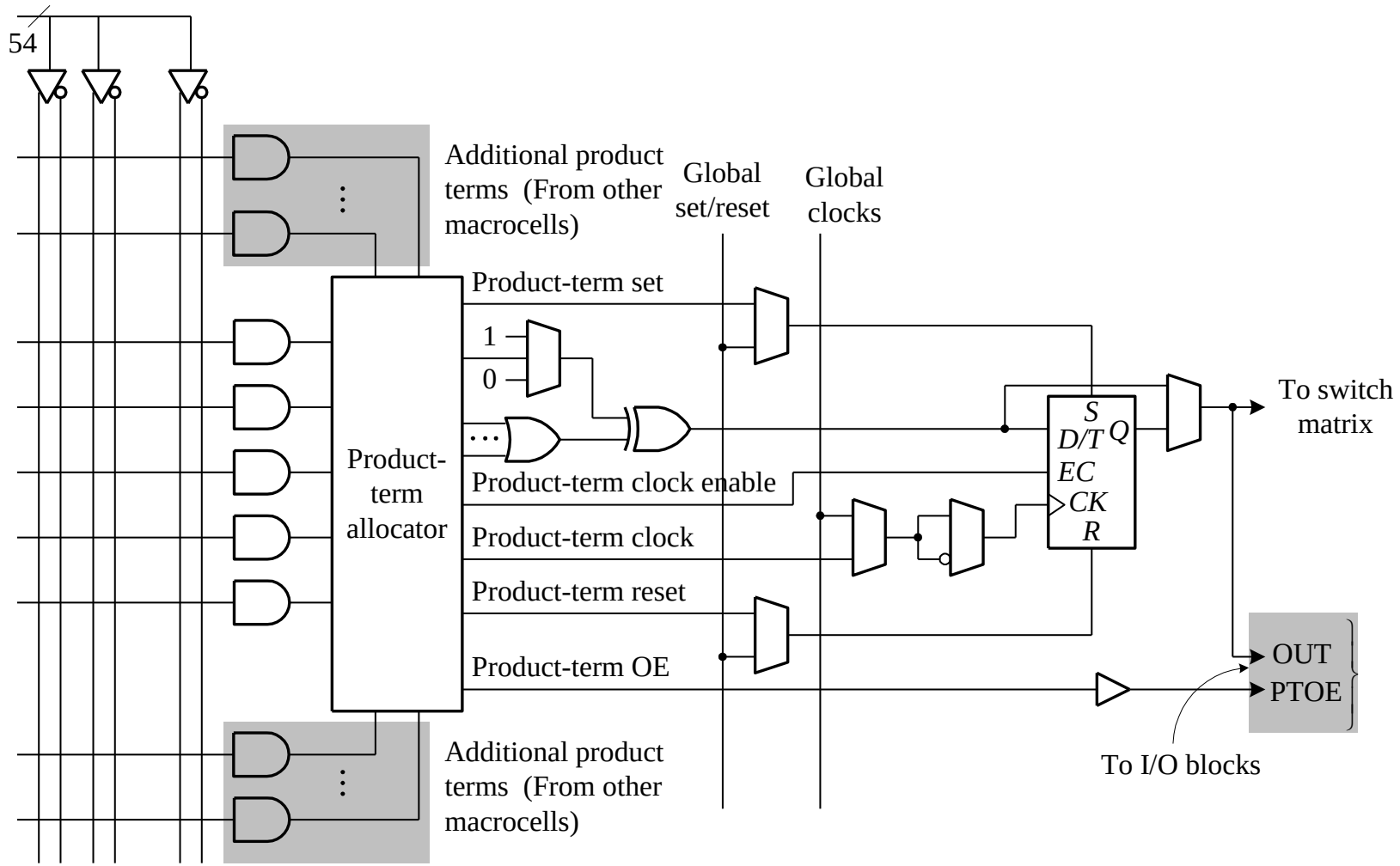
Xilinx Basic Structure of CPLD: XC9500XL



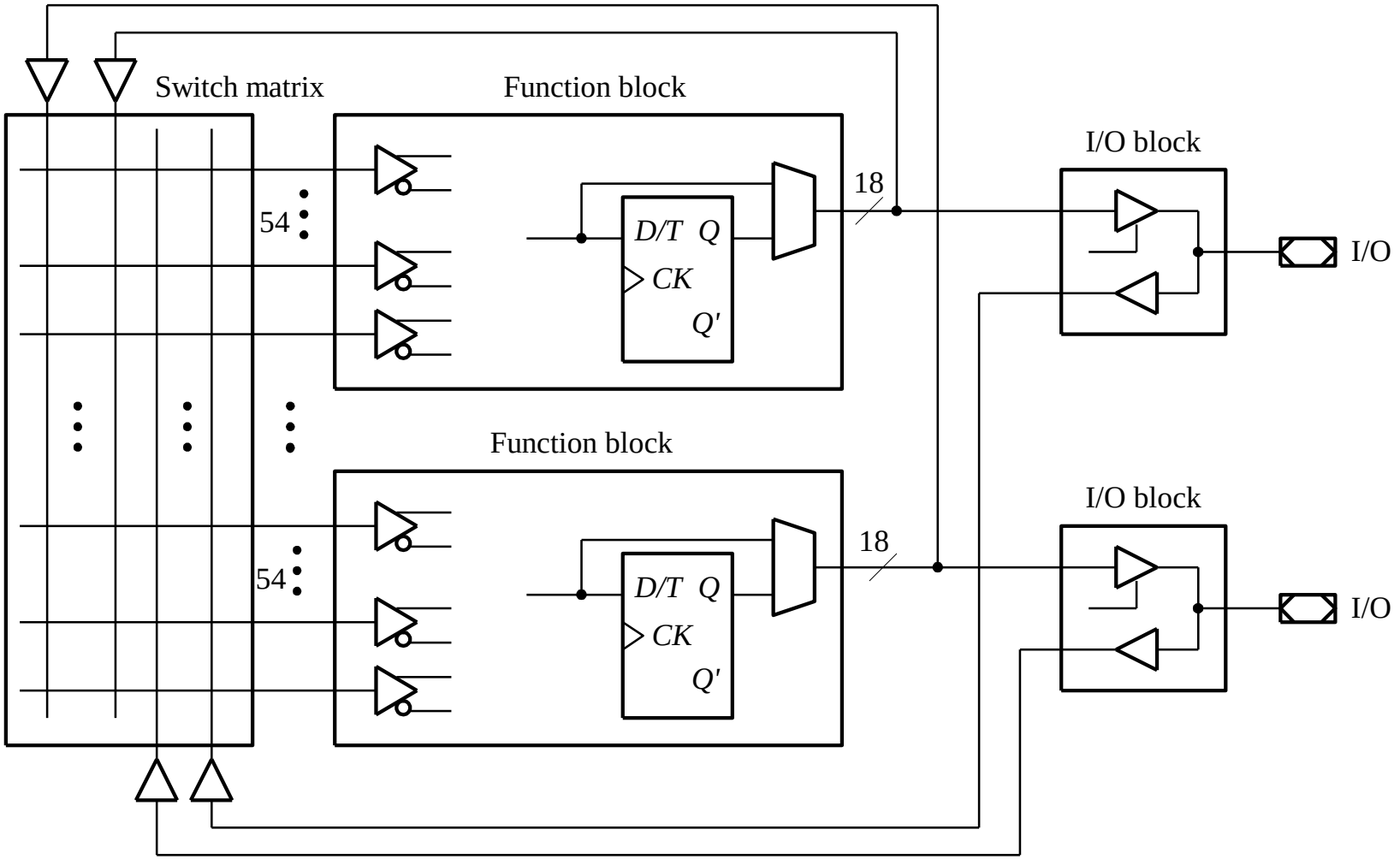
CPLD: XC9500XL --- Function Block



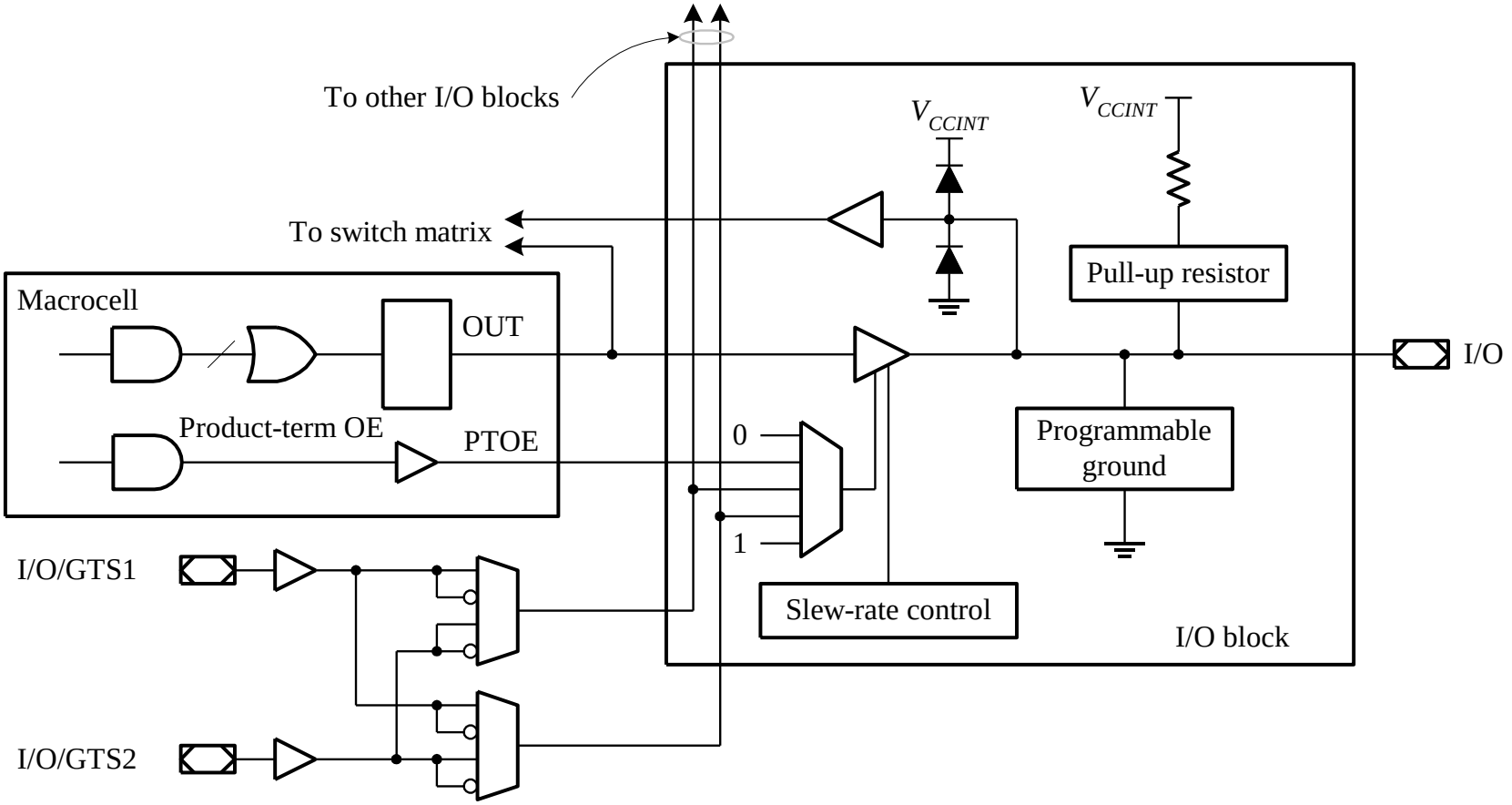
CPLD: XC9500XL --- Macro of Function Block



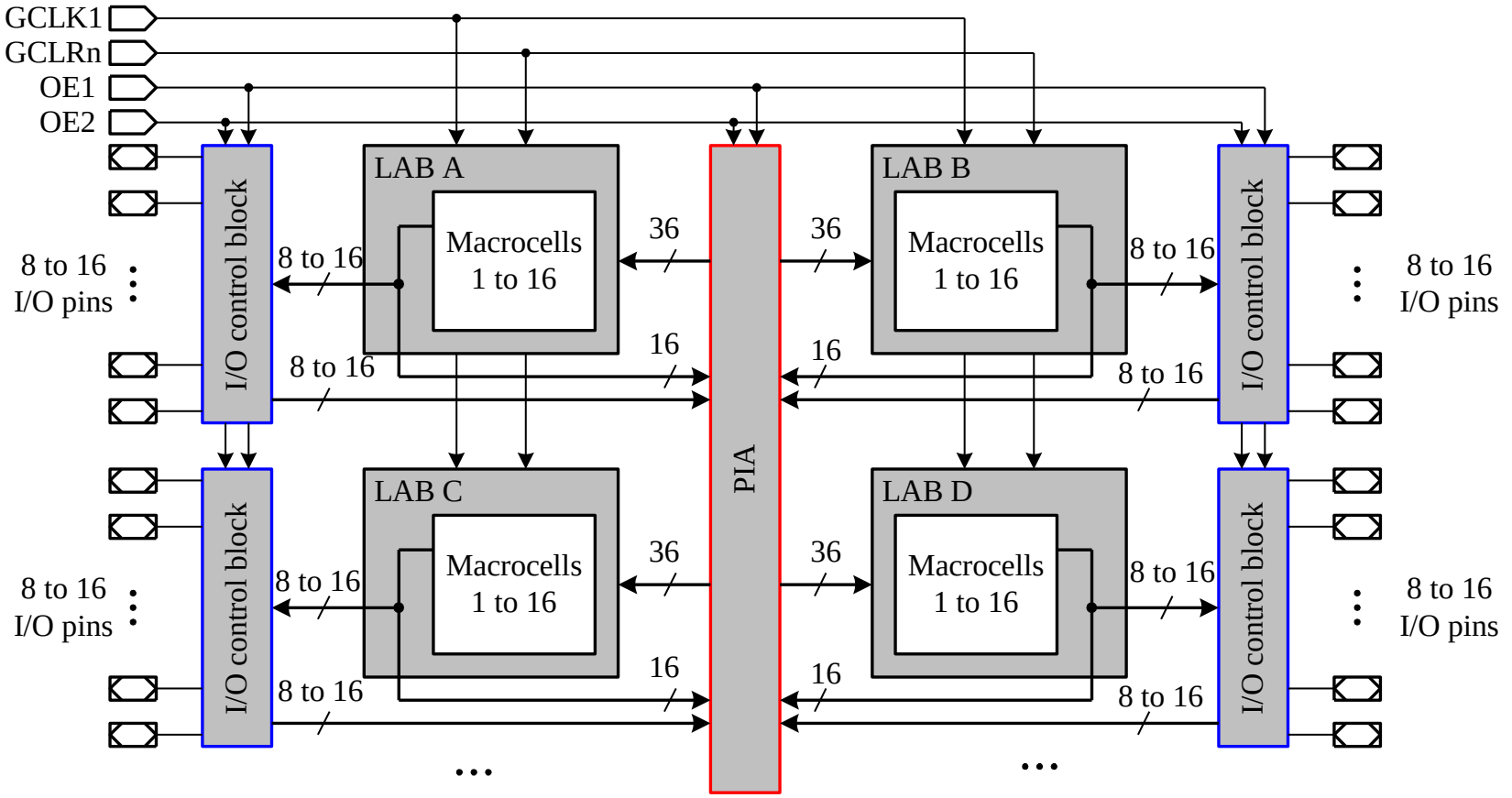
CPLD: XC9500XL --- Switch Matrix



CPLD: XC9500XL --- I/O Block

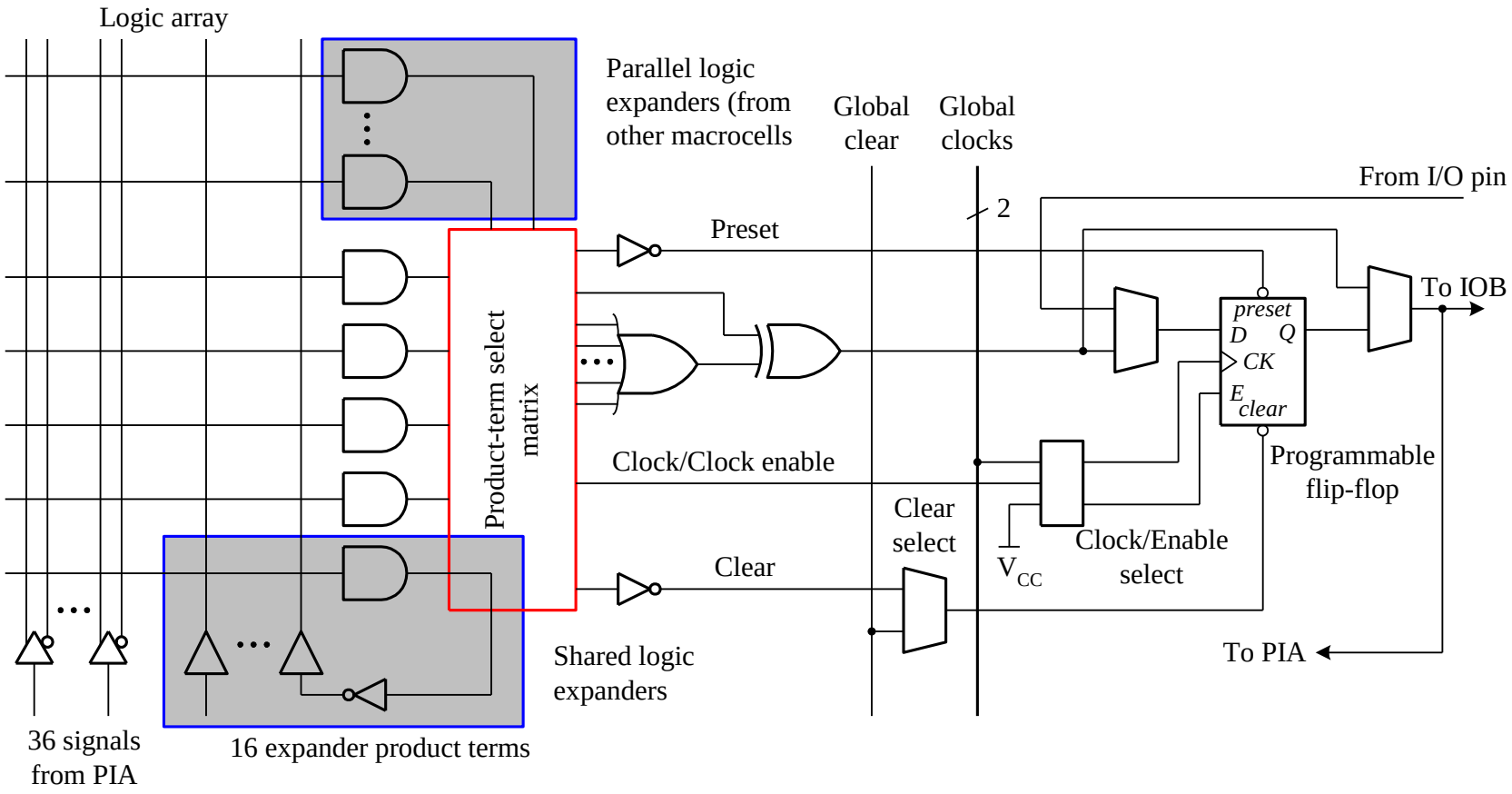


MAX7000 Family --- Basic Structure

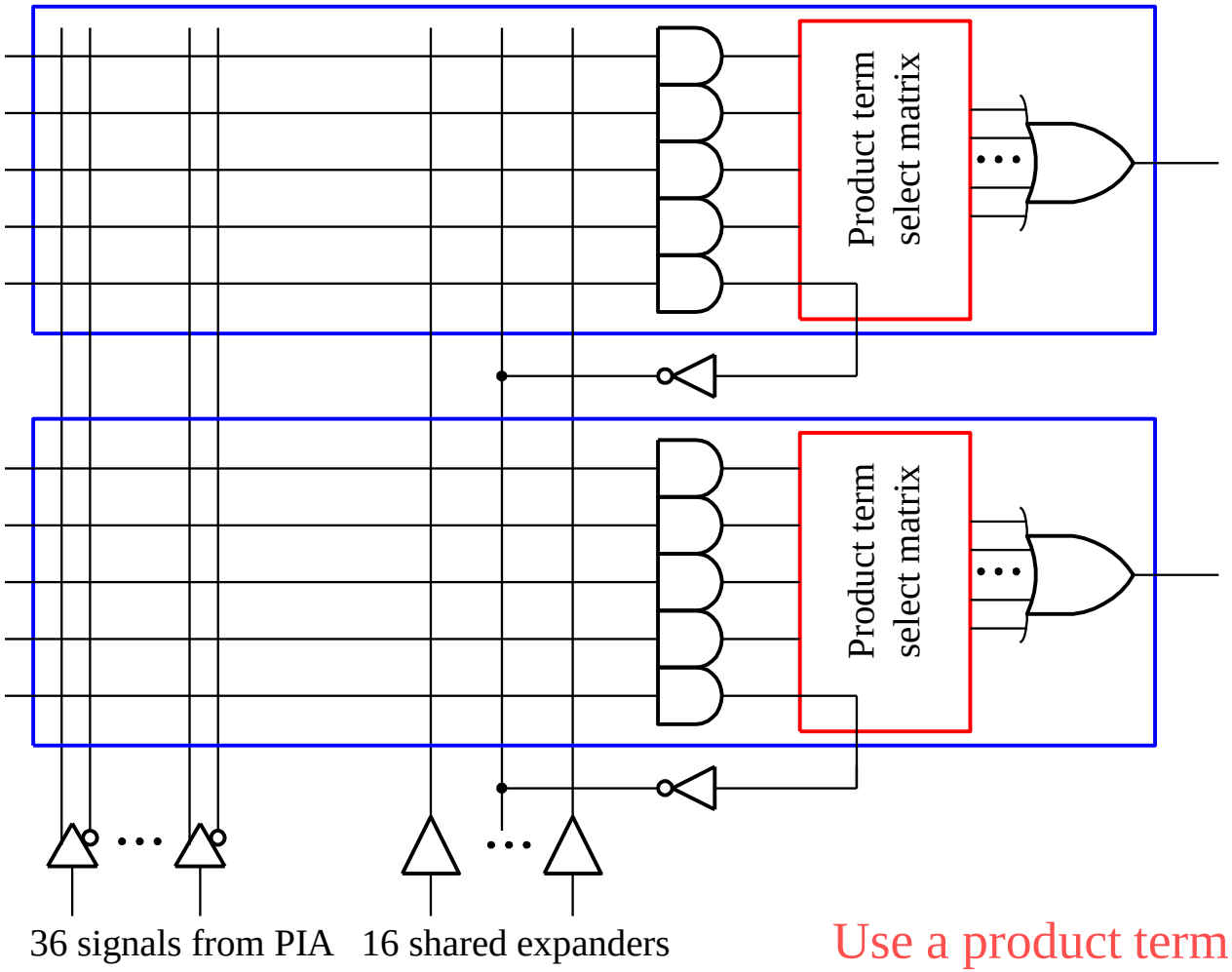


LAB: logic array block, PIA: programmable interconnect array

MAX7000 Family --- Macrocell Structure

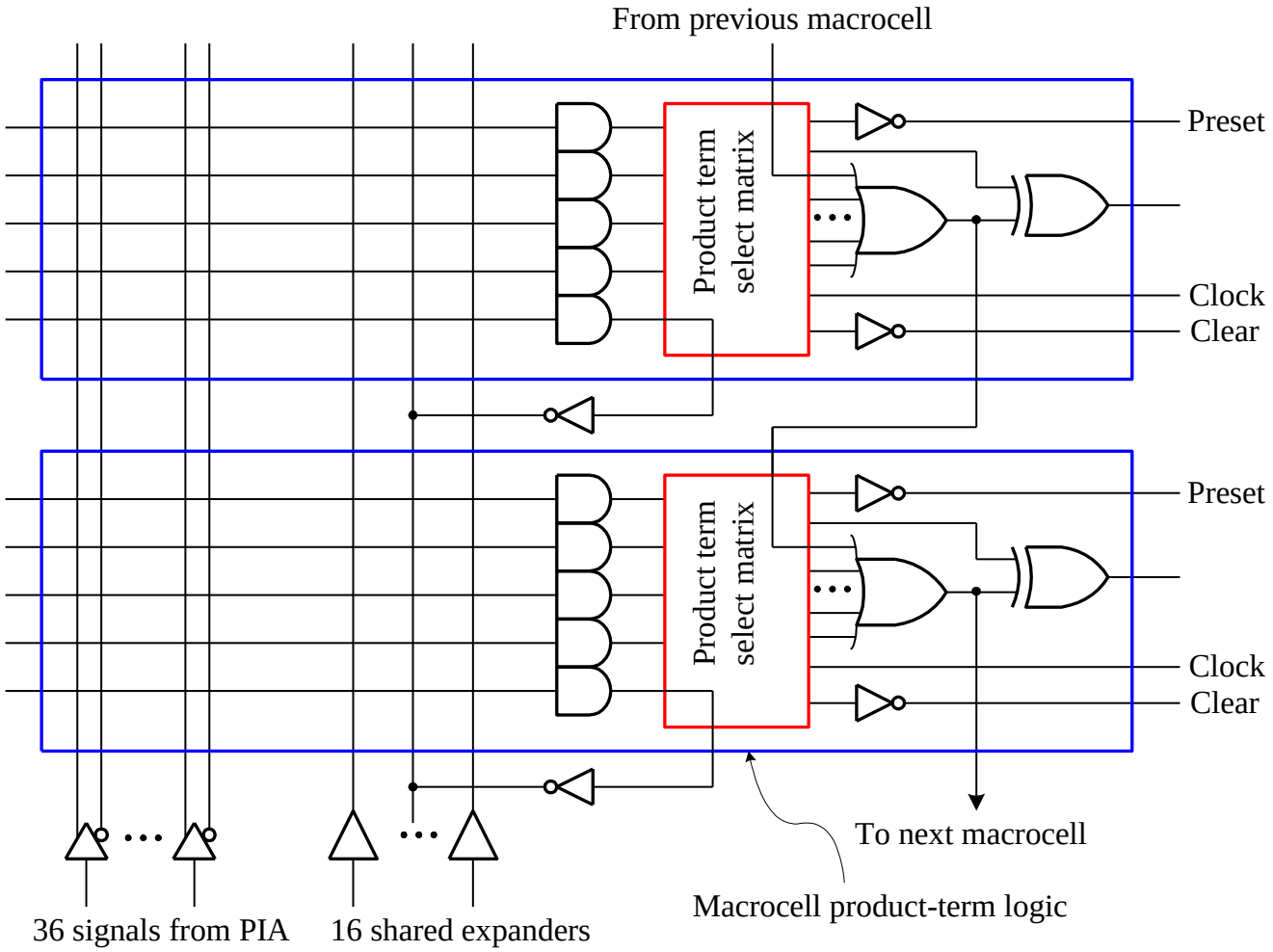


MAX7000 Family --- Sharable Expander



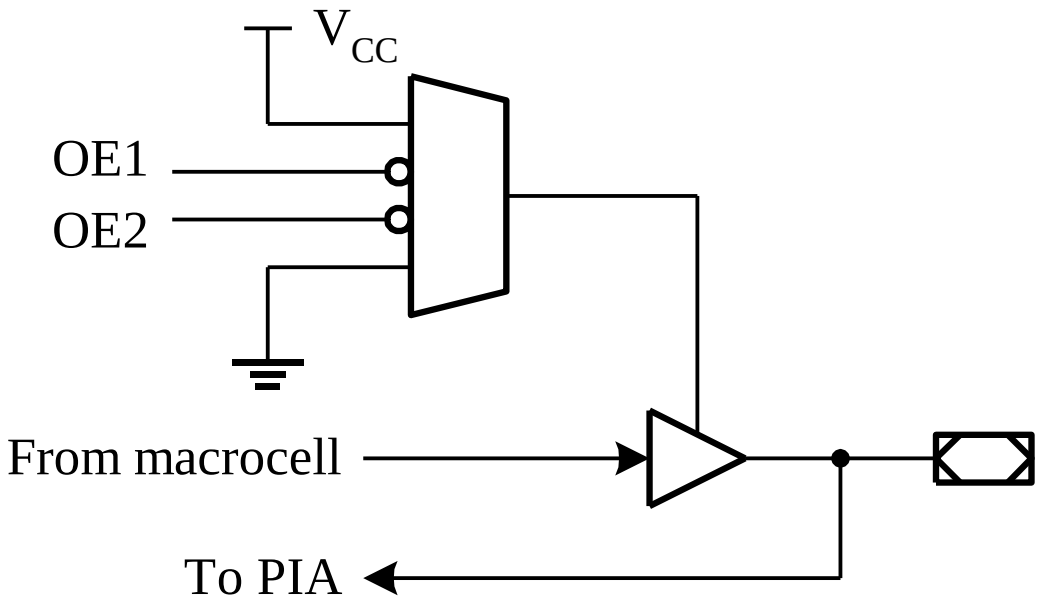
Use a product term as an input

MAX7000 Family --- Parallel Expander

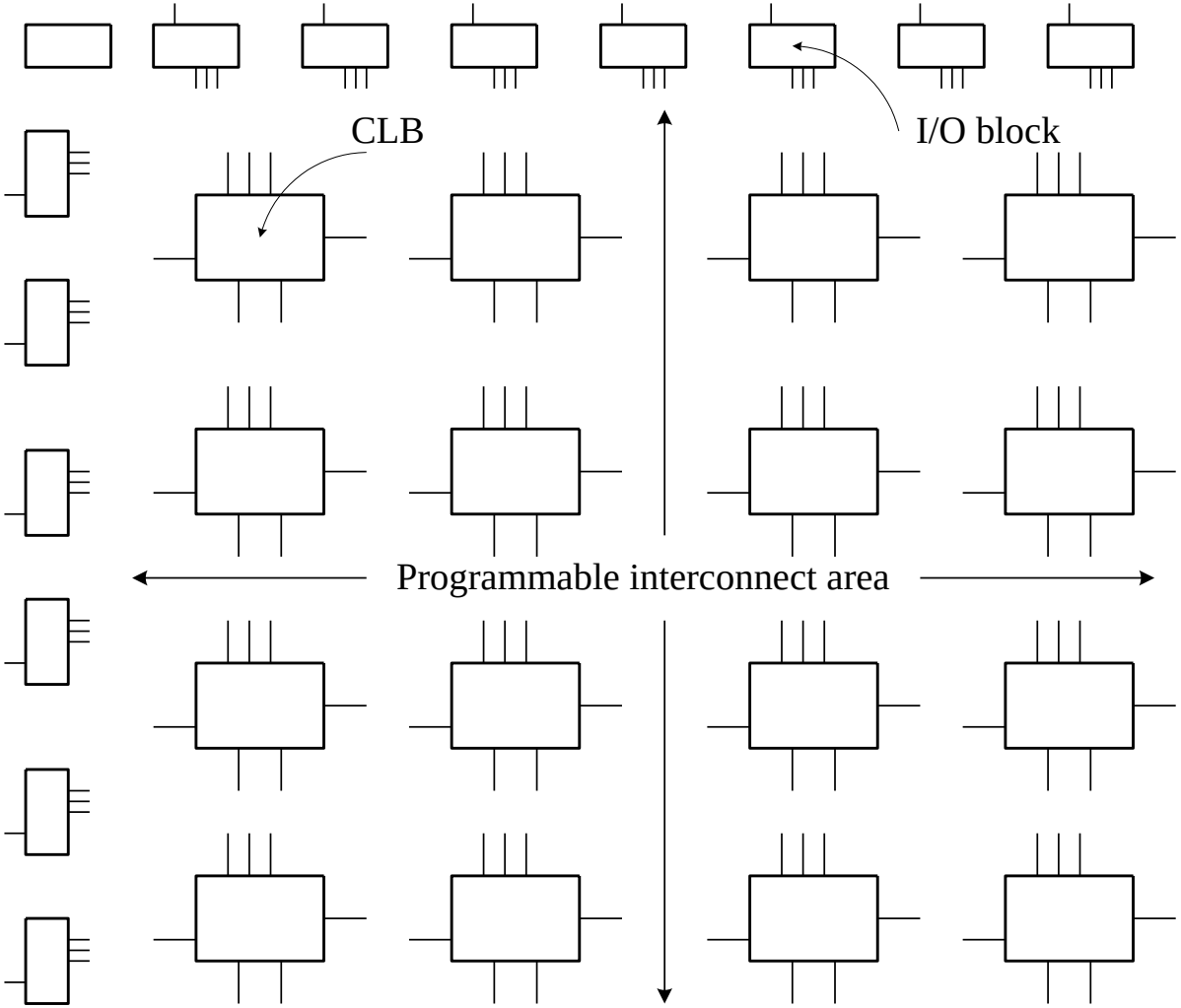


Use product terms up to 20 from the neighbor macrocell

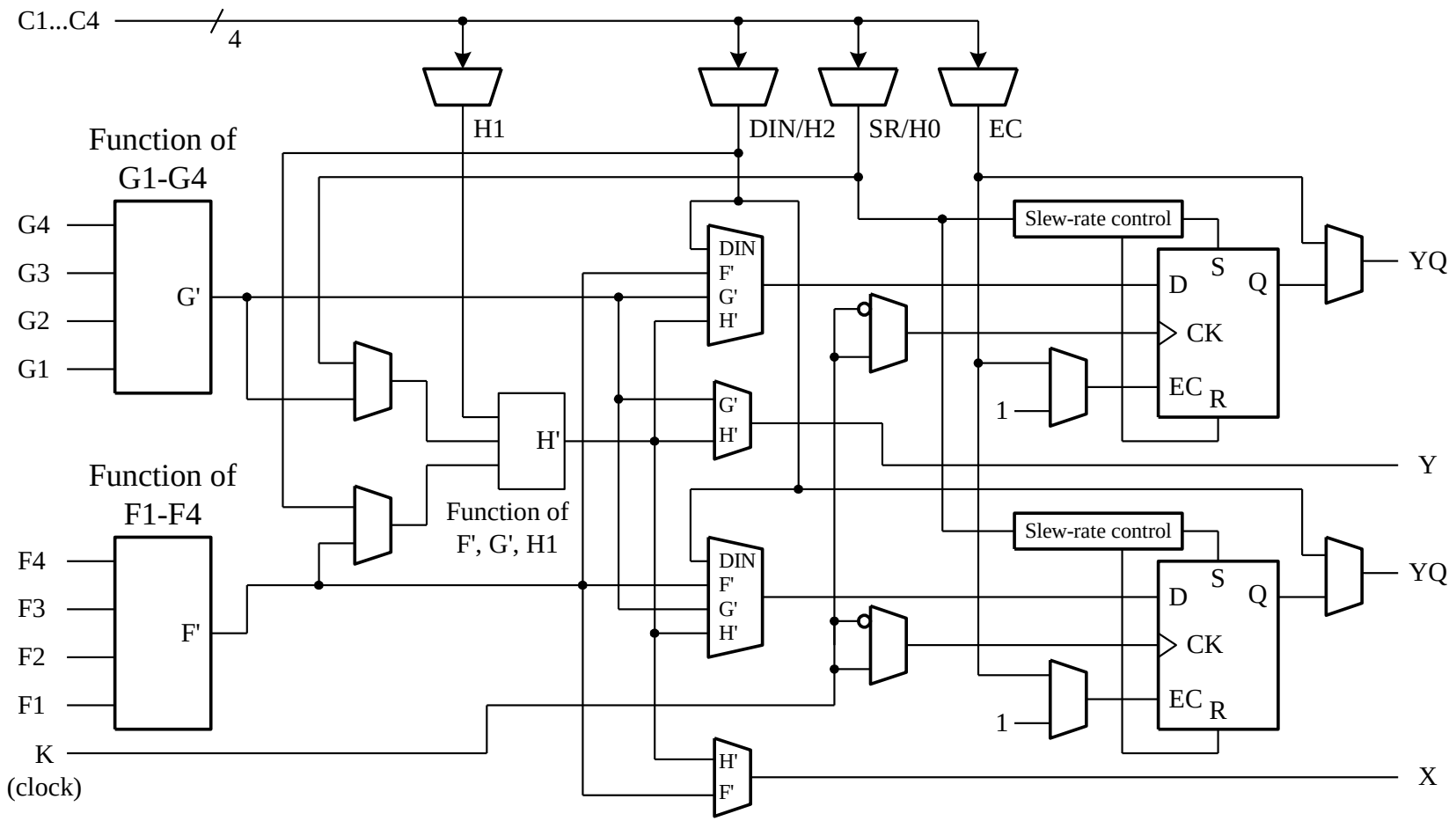
MAX7000 Family --- IOB



Xilinx Basic Structure of FPGA: XC4000XL

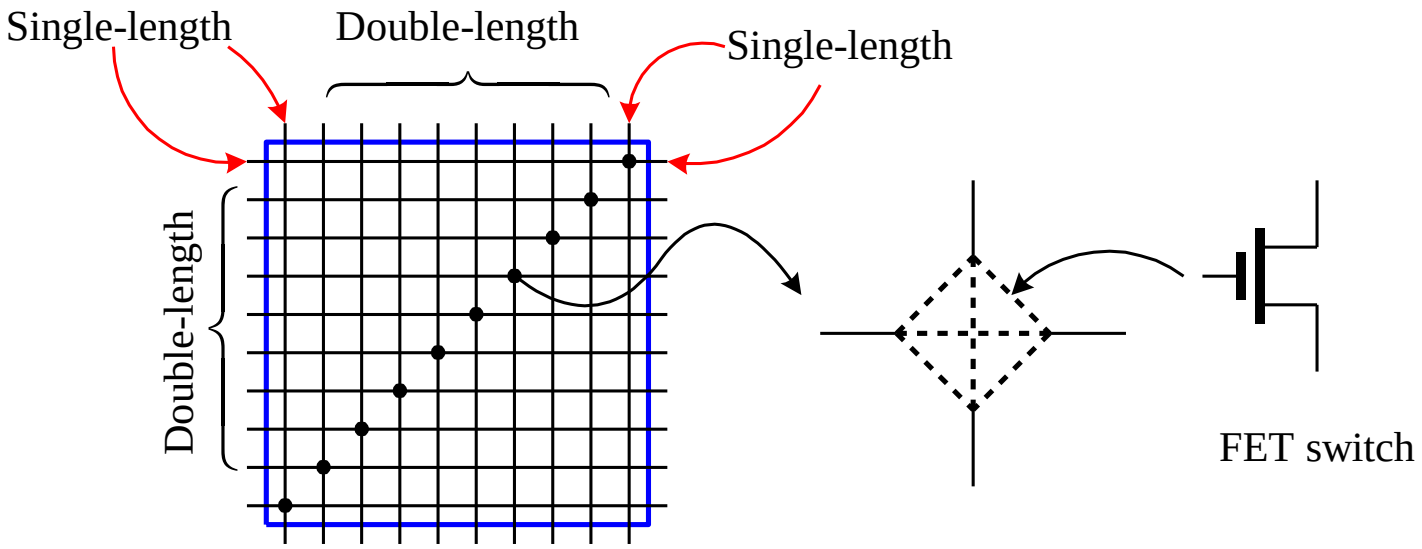


FPGA: XC4000XL --- Configurable Logic Block

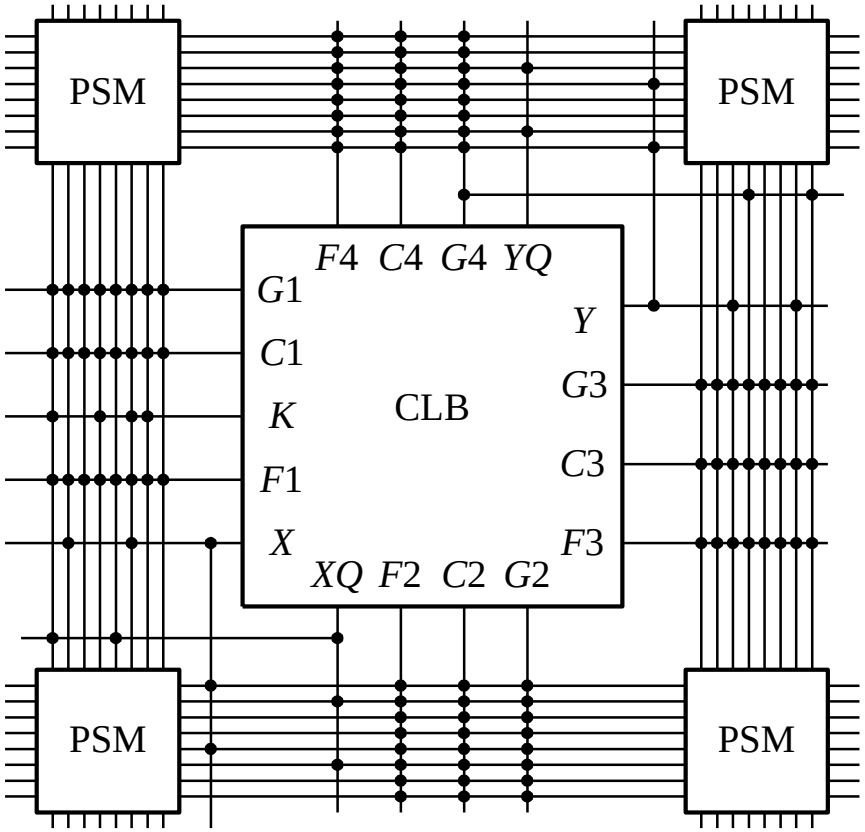


FPGA: XC4000XL --- PIA

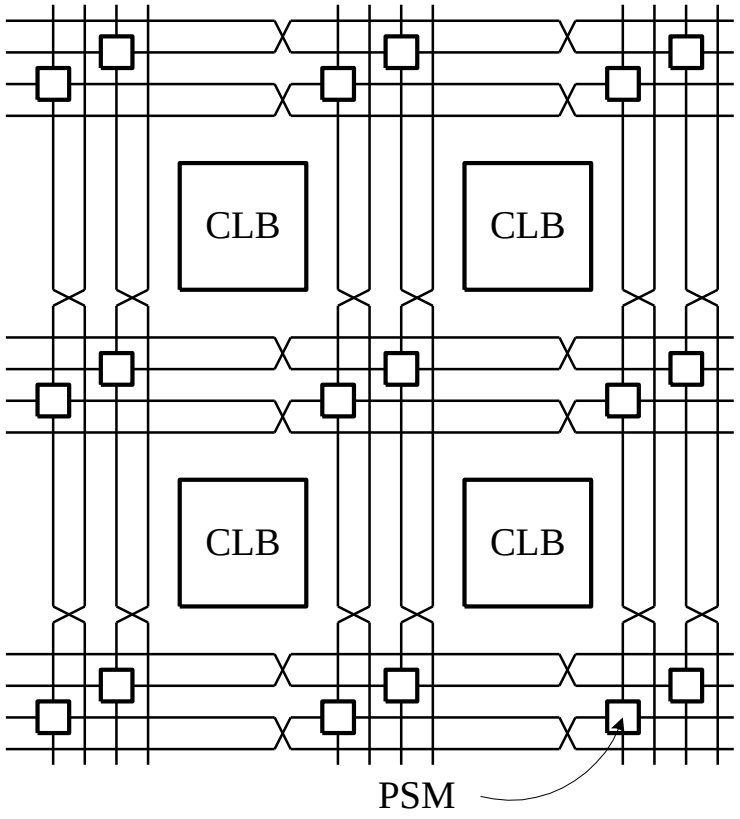
- Interconnection lines:
  - Single-length lines
  - Double-length lines
  - Global long-length lines



FPGA: XC4000XL --- PIA



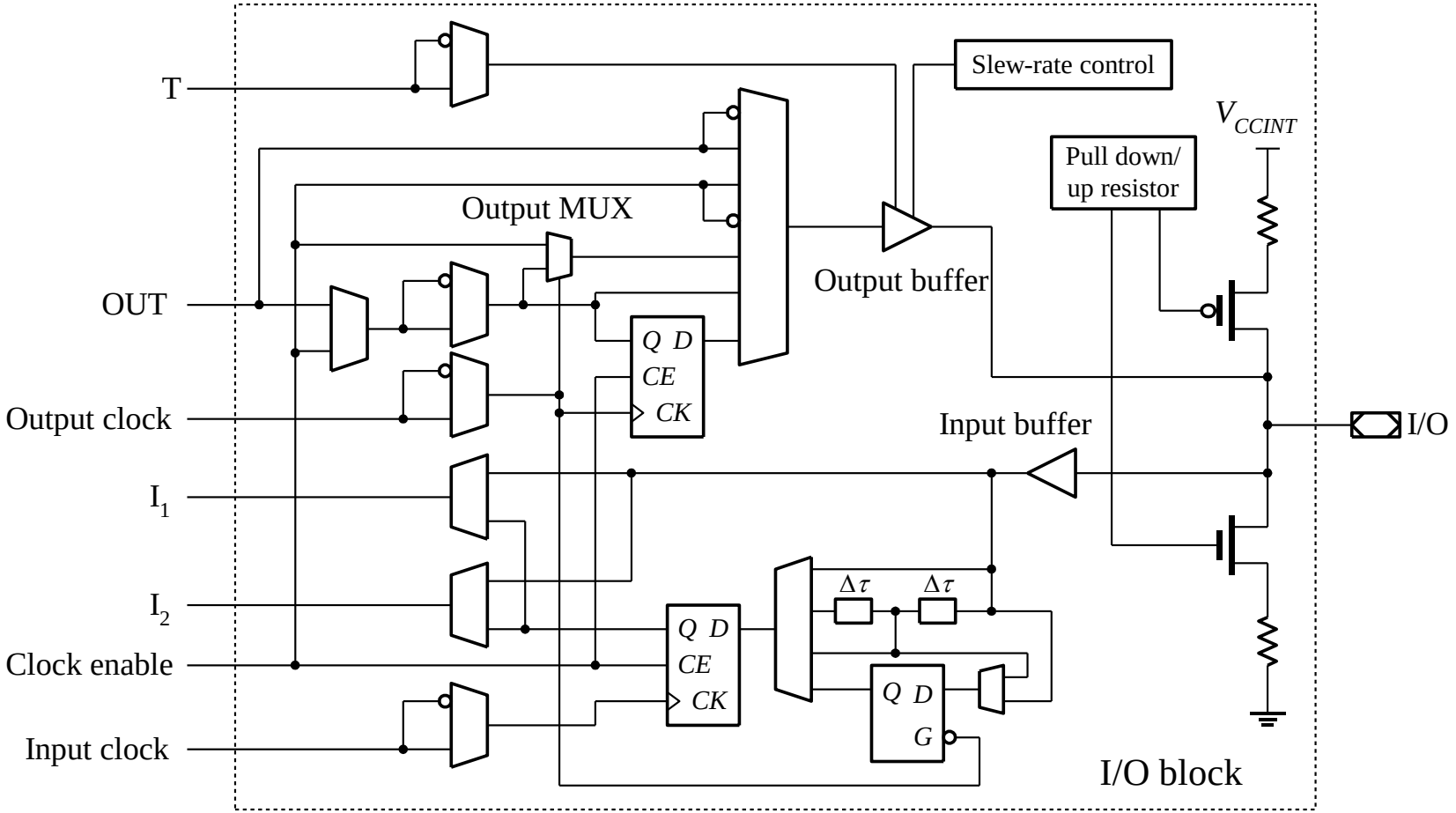
(a) Single-length lines



(b) Double-length lines

PSM: programmable switch matrix

FPGA: XC4000XL --- Input/Output Block



FPGA: XC4000XL --- Macro Library Example

- Macros may be categorized into two basic classes:
  - Soft macros only describe logic and interconnections.
  - Hard macros describe where the elements are placed.

Soft macros		Hard macros	
Accumulators	Adder/Subtractors	Accumulators	Adders
Counters	Comparators	Counters	Comparators
Data/shift registers	Dividers	Data/shift registers	Decoders
Flip-Flops/latches	Encoders/decoders	RAM	Dividers
FSMs	Gates/buffers		Priority encoders
RAMs/ROMs	Logical shifters		Logical shifters
	Multiplexers		Multiplexers
	Muitipliers		Muitipliers
	Parity checkers		Parity checkers
	Tristate buffers		

Features of Spartan-XL and Spartan-II Series

Spartan/XL FPGAs	XCS05/XL	XCS10/XL	XCS20/XL	XCS30/XL	XCS40/XL
System gates	5k	10k	20k	30k	40k
Logic cells	238	466	950	1,368	1,862
Max. no. of logic gates	3,000	5,000	10,000	13,000	20,000
Flip-flops	360	616	1,120	1,536	2,016
Max. RAM bits	3,200	6,272	12,800	18,432	25,088
Max. available I/O	77	112	160	192	224

Spartan-II FPGAs	XC2S15	XC2S30	XC2S50	XC2S100	XC2S150	XC2S200
System gates	15k	30k	50k	100k	150k	200k
Logic cells	432	972	1,728	2,700	3,888	5,292
Blocks RAM (kb)	16	24	32	40	48	56
DLL	4	4	4	4	4	4
Max. Dist. RAM (kb)	4	13.5	24	37.5	54	73.5
Max. available I/O	86	132	176	196	260	284

Features of Spartan-3 Series

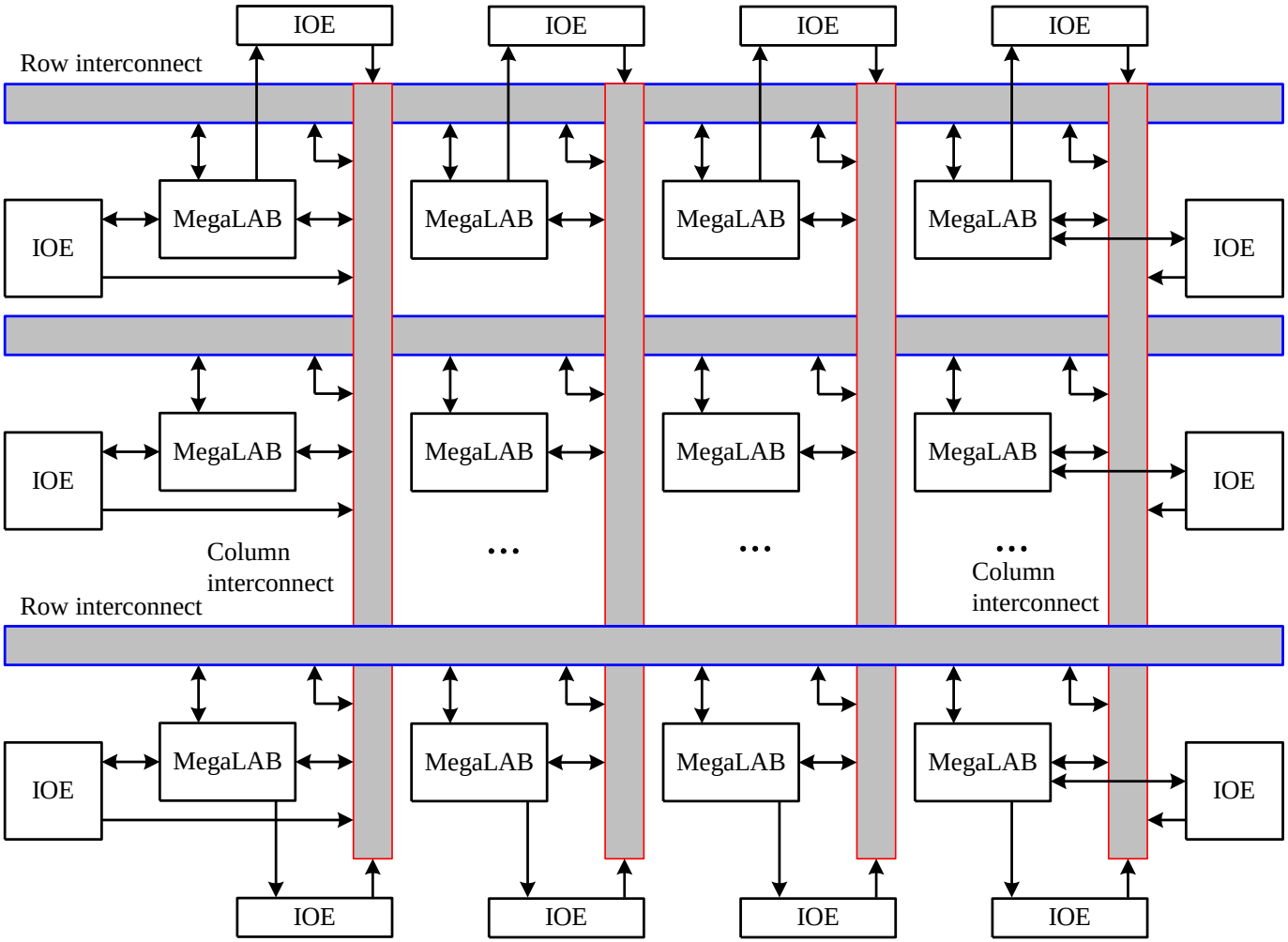
Spartan-III FPGAs	XC3 S50	XC3 S200	XC3 S400	XC3 S1000	XC3 S1500	XC3 S2000	XC3 S4000	XC3 S5000
System gates	50k	200k	400k	1000k	1500k	2000k	4000k	5000k
Logic cells	1,728	4,320	8,064	17,280	29,952	46,080	62,208	74,880
18 * 18 multipliers	4	12	16	24	32	40	96	104
Blocks RAM (kb)	72	216	288	432	576	720	1,728	1,872
Max. Dist. RAM (kb)	12	30	56	120	208	320	432	520
DCMs	2	4	4	4	4	4	4	4
I/O standards	24	24	24	24	24	24	24	24
Max. Diff. I/O pairs	56	76	116	175	221	270	312	344
Max. Single ended I/O	124	173	264	391	487	565	712	784

Features of Virtext-II Series

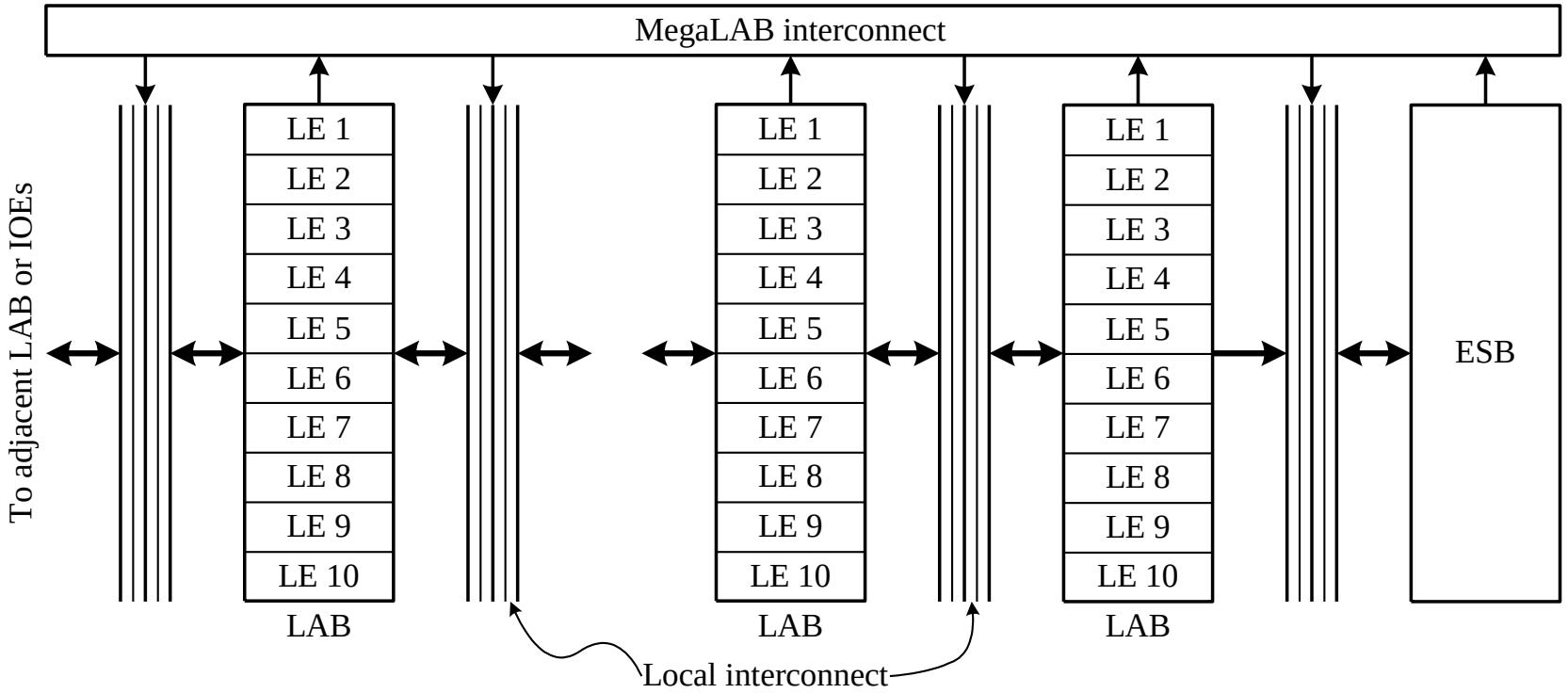
Table 1: **Virtex-II Field-Programmable Gate Array Family Members**

Device	System Gates	CLB (1 CLB = 4 slices = Max 128 bits)			Multiplier Blocks	SelectRAM Blocks		DCMs	Max I/O Pads(1)
		Array Row x Col.	Slices	Maximum Distributed RAM Kbits		18 Kbit Blocks	Max RAM (Kbits)		
XC2V40	40K	8 x 8	256	8	4	4	72	4	88
XC2V80	80K	16 x 8	512	16	8	8	144	4	120
XC2V250	250K	24 x 16	1,536	48	24	24	432	8	200
XC2V500	500K	32 x 24	3,072	96	32	32	576	8	264
XC2V1000	1M	40 x 32	5,120	160	40	40	720	8	432
XC2V1500	1.5M	48 x 40	7,680	240	48	48	864	8	528
XC2V2000	2M	56 x 48	10,752	336	56	56	1,008	8	624
XC2V3000	3M	64 x 56	14,336	448	96	96	1,728	12	720
XC2V4000	4M	80 x 72	23,040	720	120	120	2,160	12	912
XC2V6000	6M	96 x 88	33,792	1,056	144	144	2,592	12	1,104
XC2V8000	8M	112 x 104	46,592	1,456	168	168	3,024	12	1,108

Altera APEX 20K Family --- Basic Structure

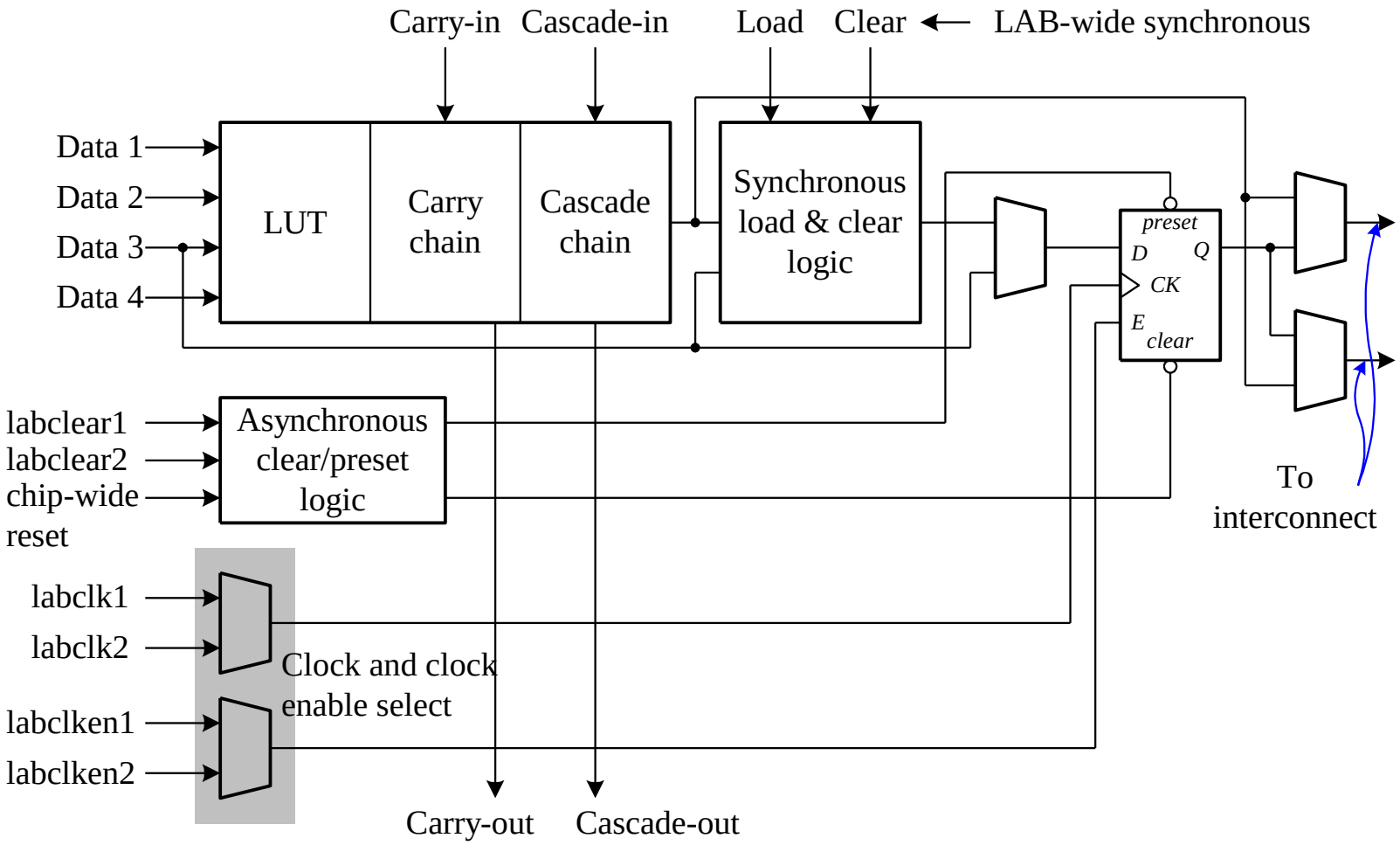


APEX 20K Family --- MegaLAB Structure

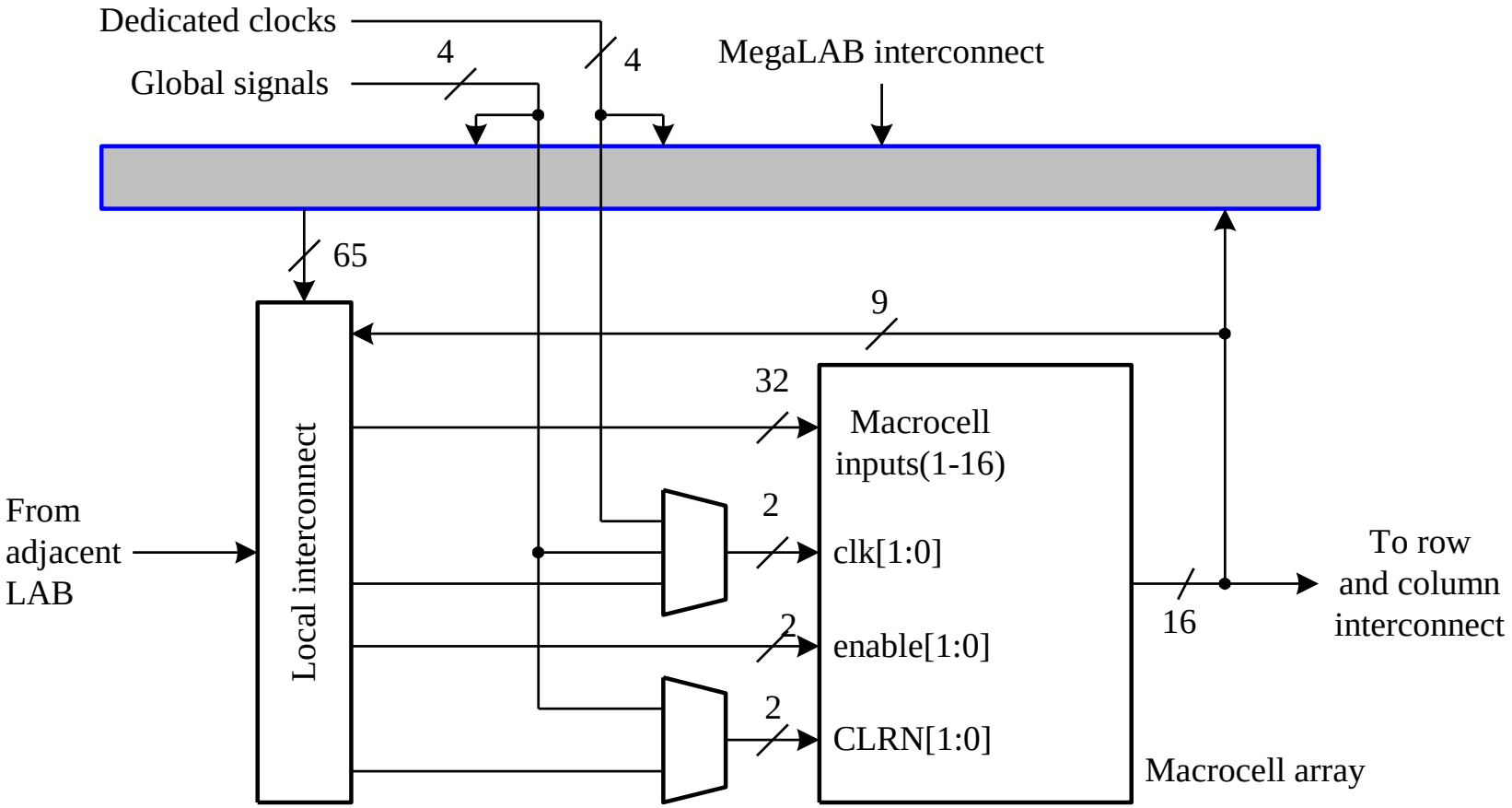


ESB (embedded system block) can be configured as a block of macrocells or implement various types of memory blocks

APEX 20K Family --- Logic Element Structure



APEX 20K Family --- Structure of ESB



Altera Stratix Family --- Basic Structure