HARDWARE DESCRIPTION LANGUAGE FOR DIGITAL DESIGN

數位設計硬體描述語言

Design Options of Digital Design

Materials partly adapted from "Digital System Designs and Practices Using Verilog HDL and FPGAs," M.B. Lin.

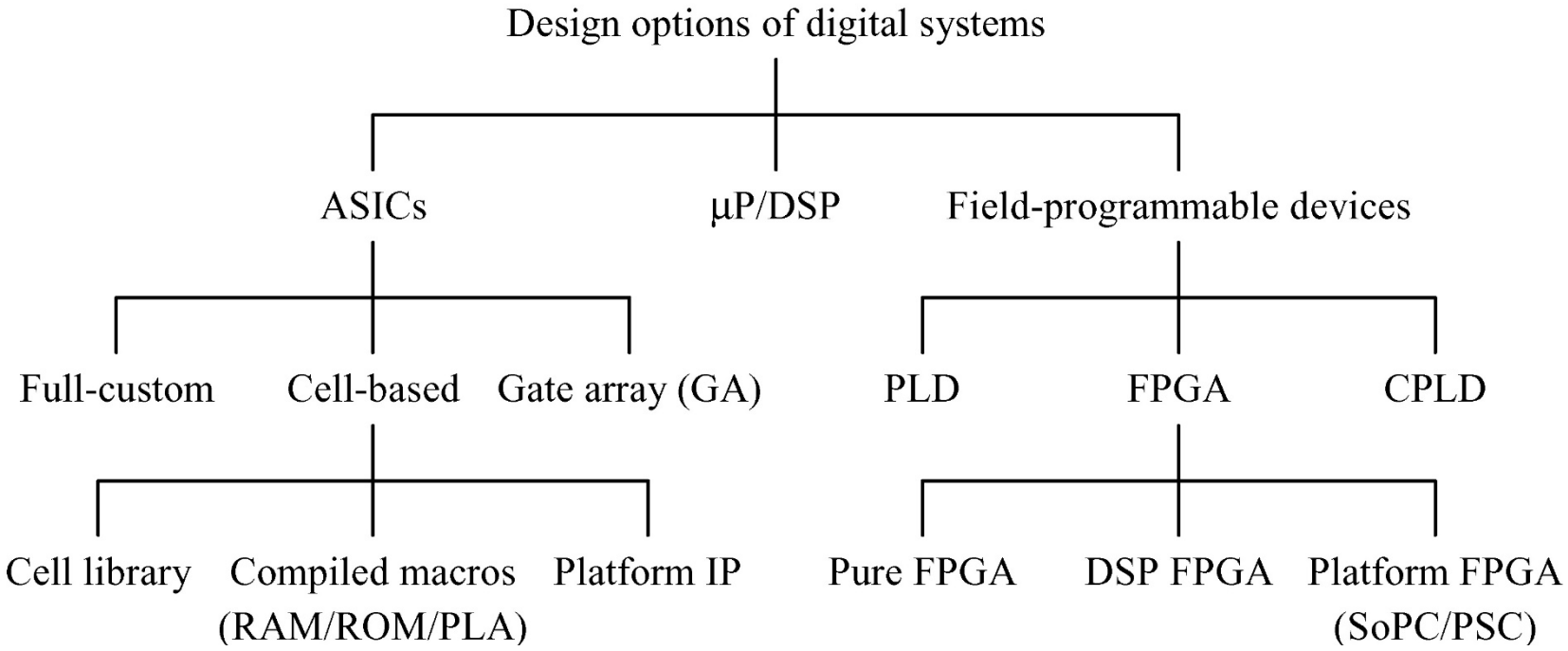
NCKU EE
LPHP Lab

OUTLINE

- Design options for digital systems
- PLD modeling
- CPLD
- FPGA

DESIGN OPTIONS OF DIGITAL SYSTEMS

DESIGN OPTIONS OF DIGITAL SYSTEMS



SYSTEM-LEVEL DESIGN OPTIONS

- PCB (printed-circuit board)-based platforms
- SoPC (System on a programmable chip)-based platforms
  - Also called programmable system chip (PSC)
- Soft IP+ cell_library-based platforms

PCB-BASED PLATFORMS

- Consist of discrete standard components
  - μP/μC, peripherals, and memory devices
- NRE cost
  - low
- May need additional CPLD/FPGA devices
- Examples
  - 8051
  - PIC
  - ARM family

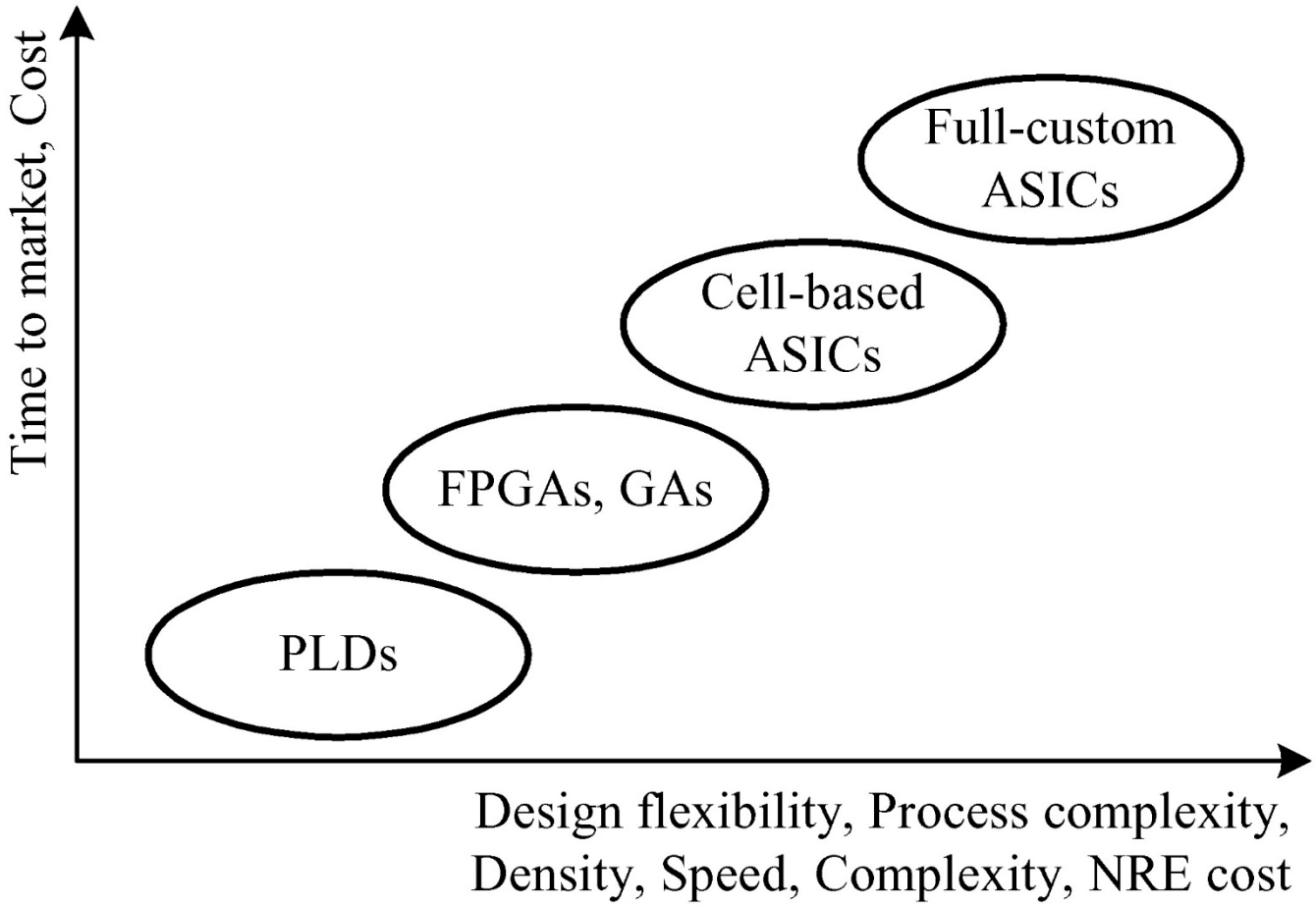
SOPC-BASED PLATFORMS

- Consist of hard or soft IPs
- May need additional customized logic modules
- Examples
  - Xilinx VirtexII pro
  - Spartan 3 (MicroBlaze)
  - Altera Cyclone Series (NIOS)

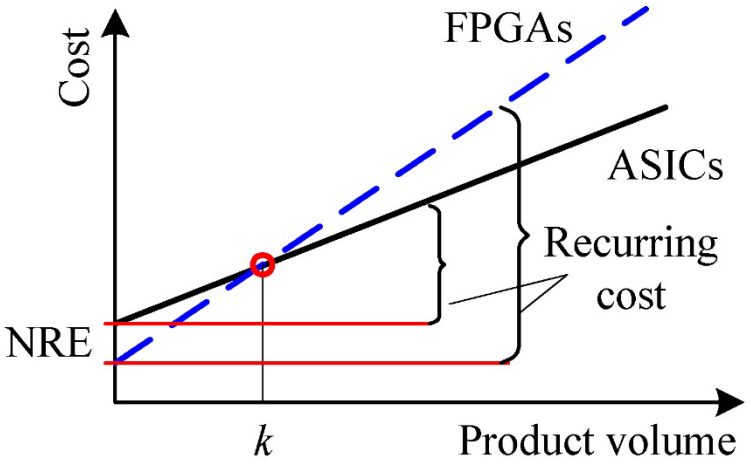
SOFT IP+ CELL LIBRARY-BASED PLATFORMS

- Consist of soft IP being configured into an optimized system hardware
- An example
  - Tensilica's Xtensa

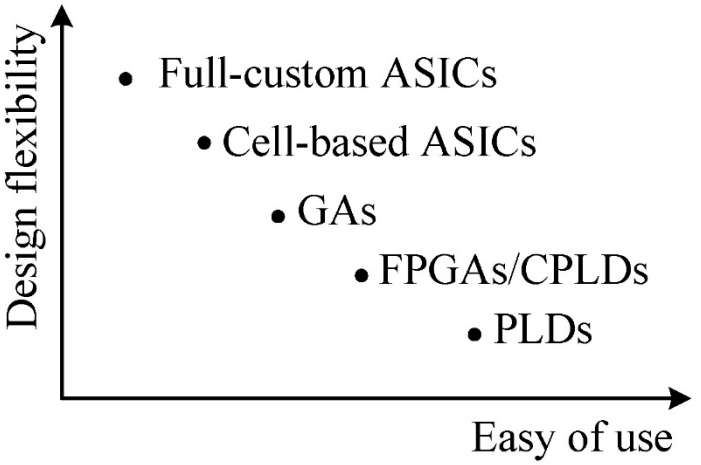
DESIGN ALTERNATIVES OF DIGITAL SYSTEMS



COMPARISON OF DESIGN OPTIONS



(a) Cost versus product volume

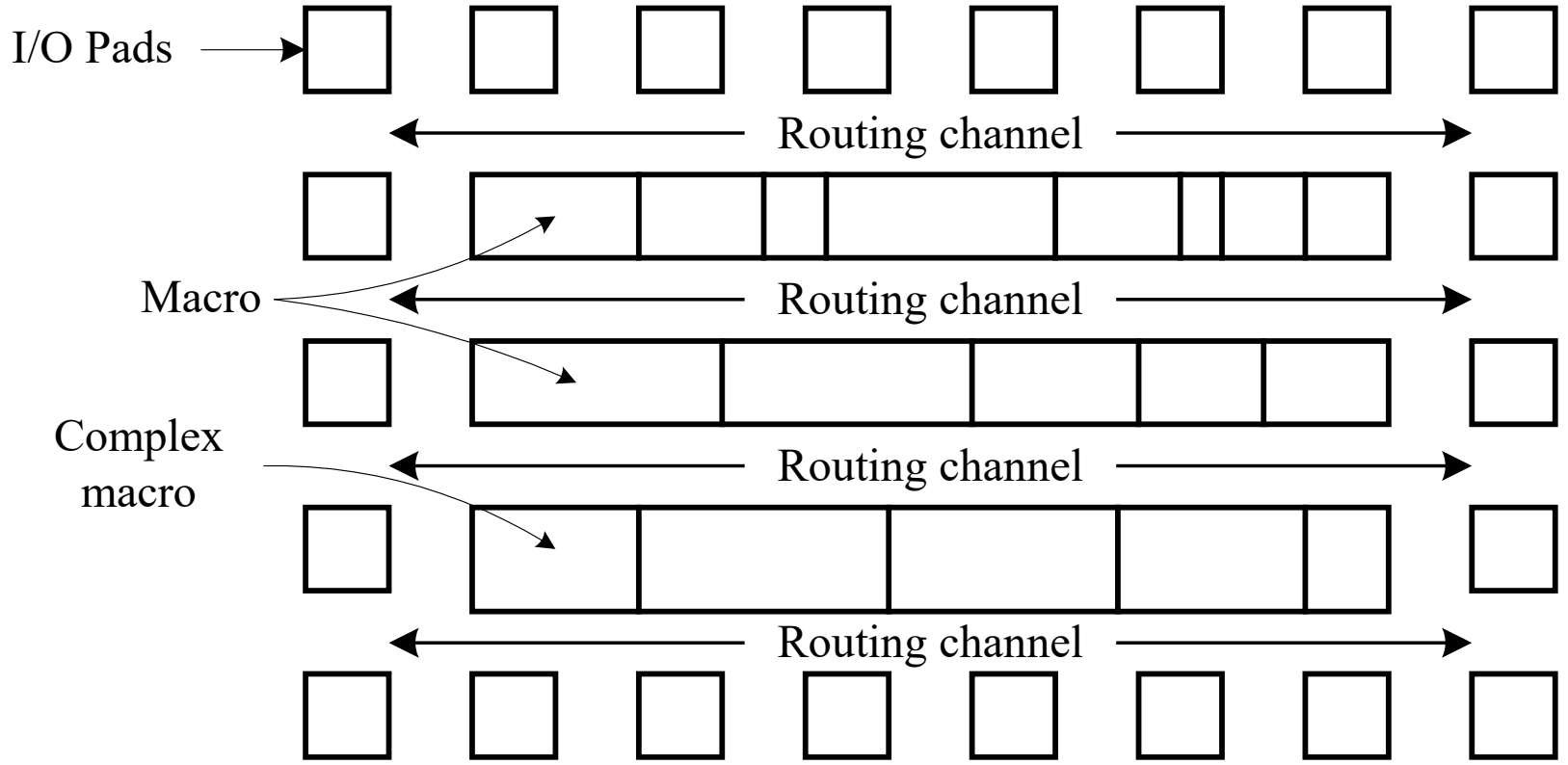


(b) Comparison of various design options of digital systems

SYLLABUS

- Objectives
- Design options of digital systems
  - Design options
  - Cell-based designs
  - Gate-array-based designs
  - Field-programmable devices
- PLD modeling
- CPLD
- FPGA
- Practical issues

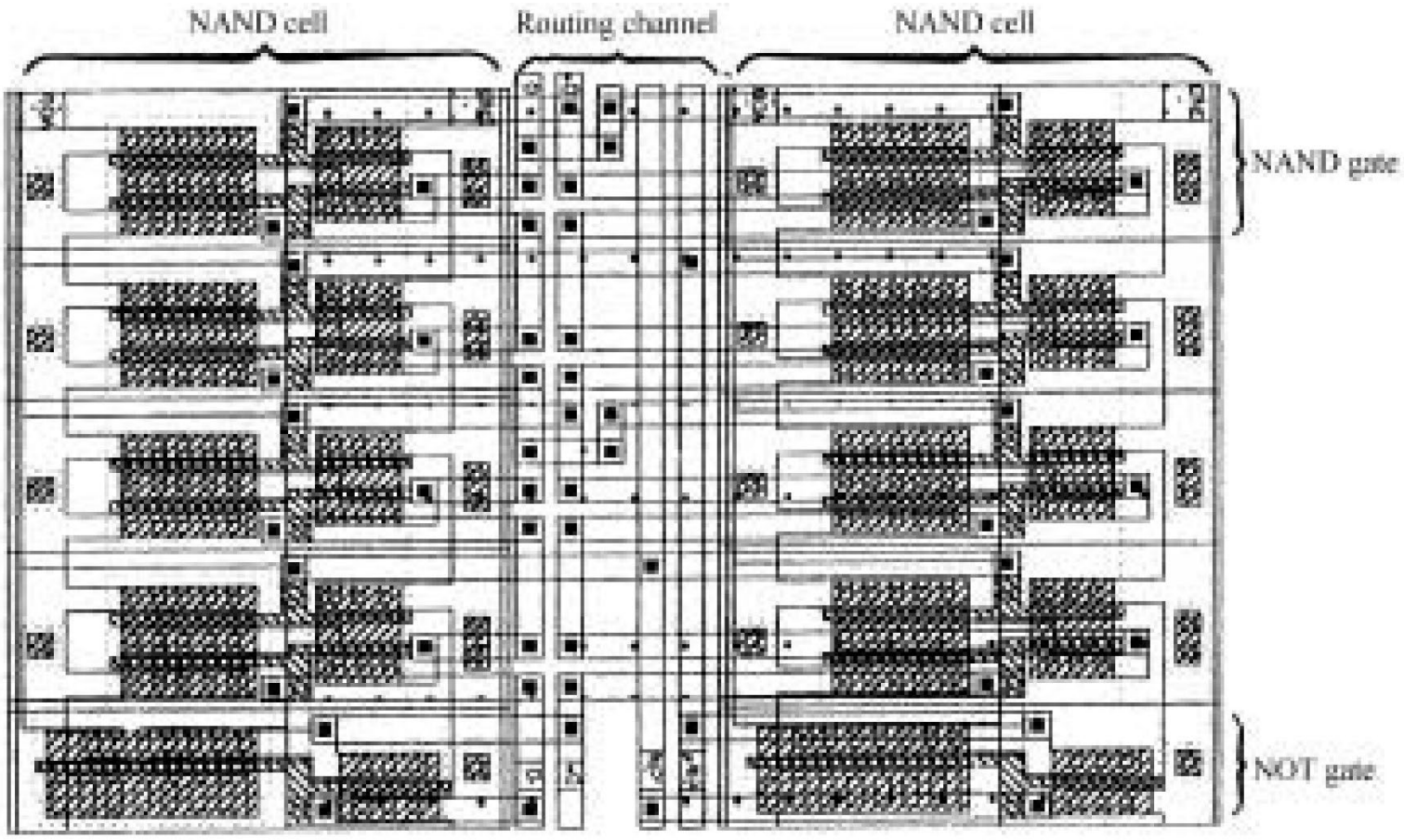
BASIC STRUCTURE OF CELL-BASED ASICS



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)

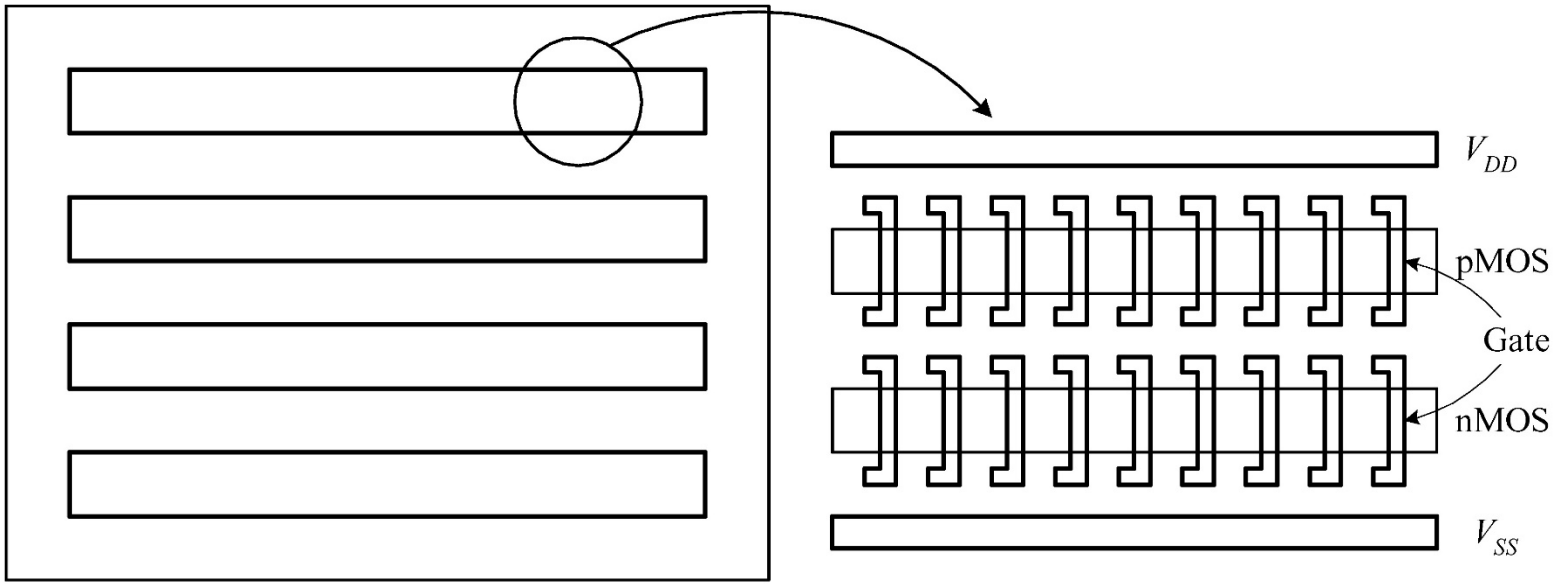
A CELL-BASED D-TYPE FLIP FLOP



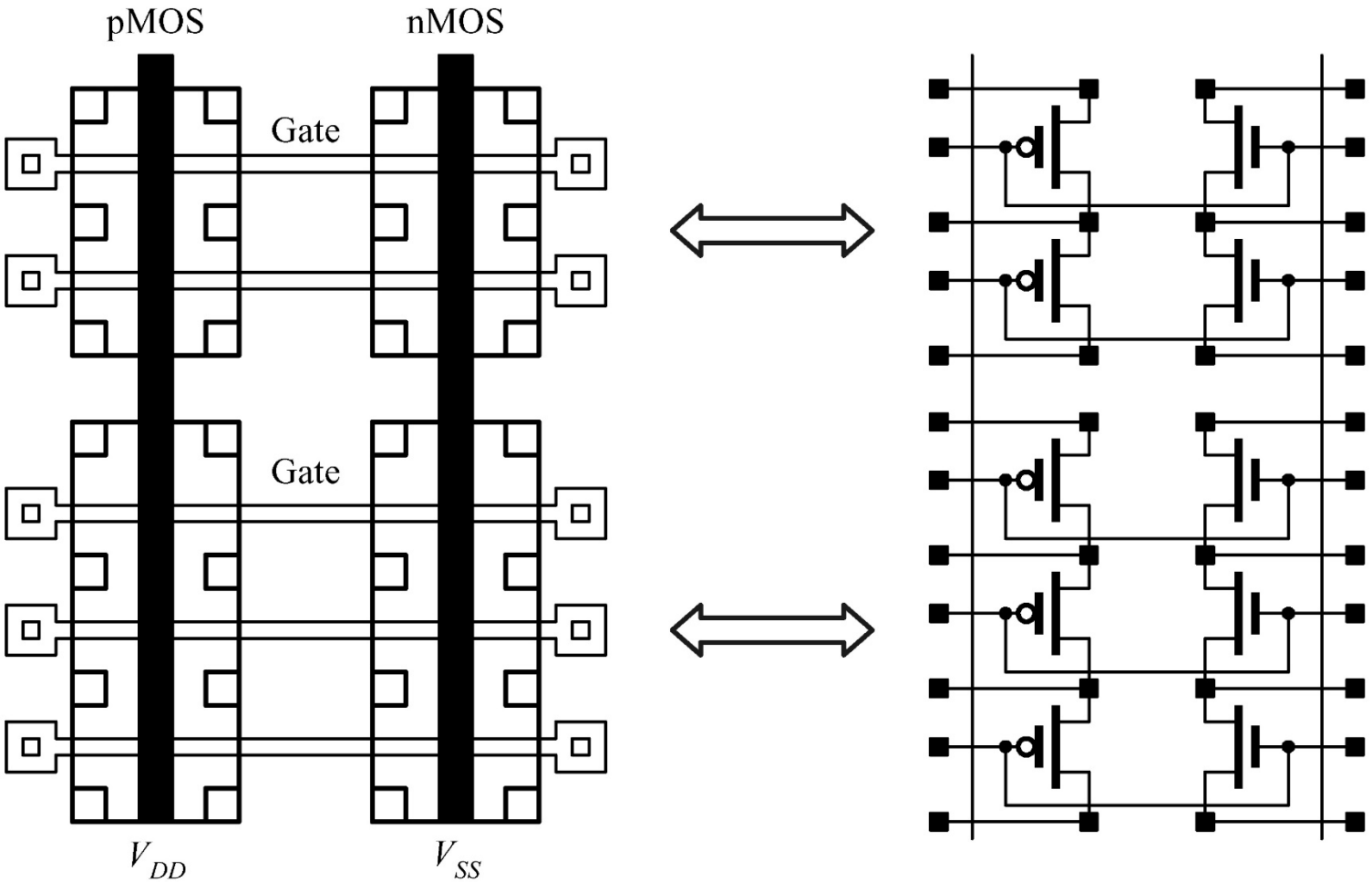
SYLLABUS

- Objectives
- Design options of digital systems
  - Design options
  - Cell-based designs
  - Gate-array-based designs
  - Field-programmable devices
- PLD modeling
- CPLD
- FPGA
- Practical issues

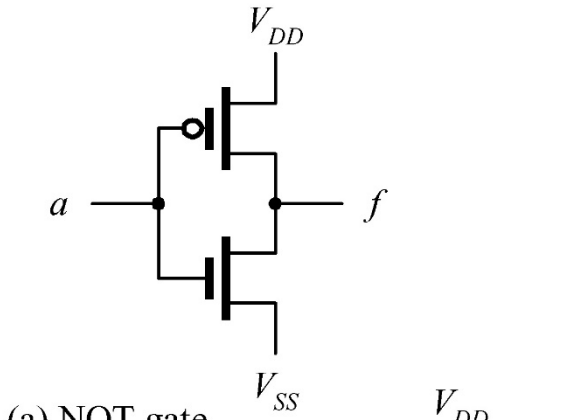
BASIC STRUCTURES OF GATE-ARRAY ASICS



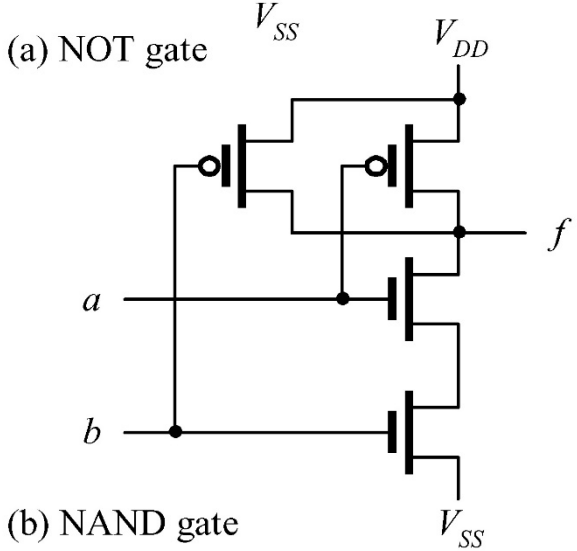
BASIC STRUCTURES OF GATE ARRAYS



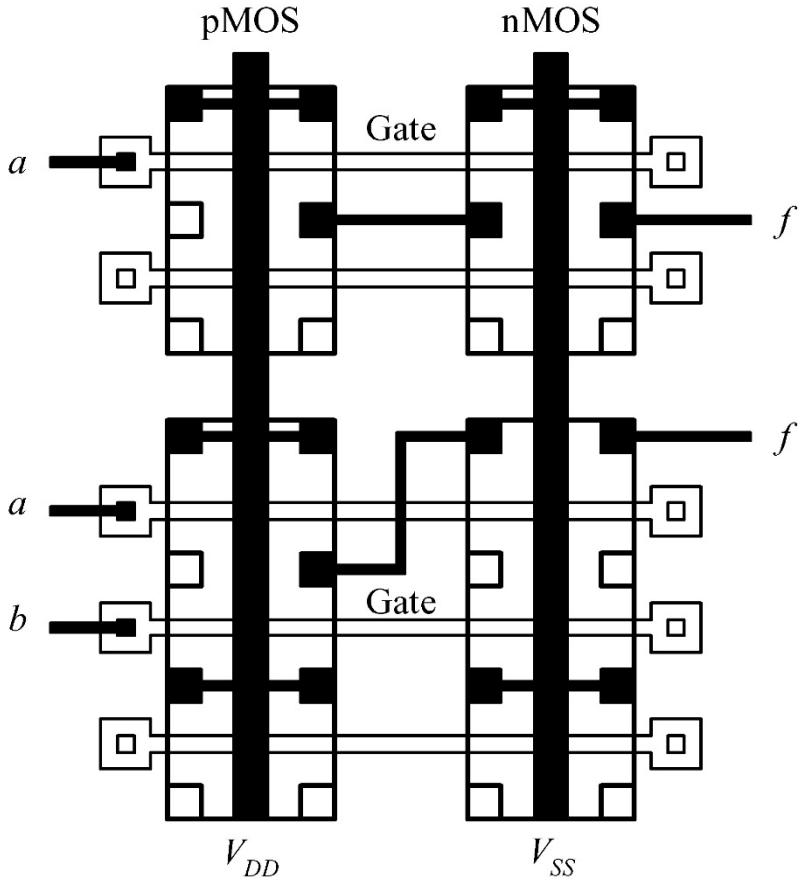
BASIC STRUCTURES OF GATE ARRAYS



(a) NOT gate



(b) NAND gate

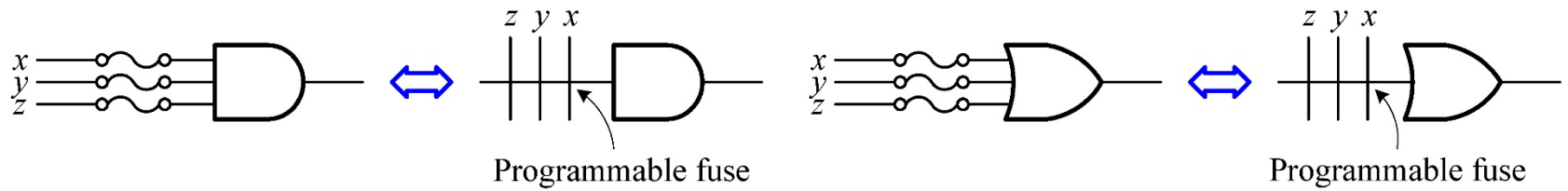


(c) Gate array interconnection

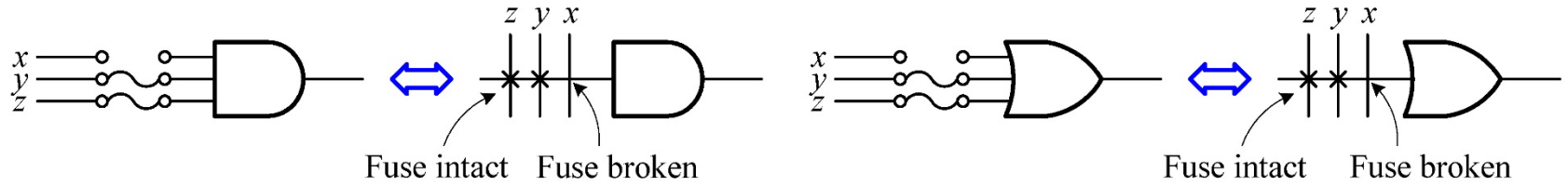
SYLLABUS

- Objectives
- Design options of digital systems
  - Design options
  - Cell-based designs
  - Gate-array-based designs
  - Field-programmable devices
- PLD modeling
- CPLD
- FPGA
- Practical issues

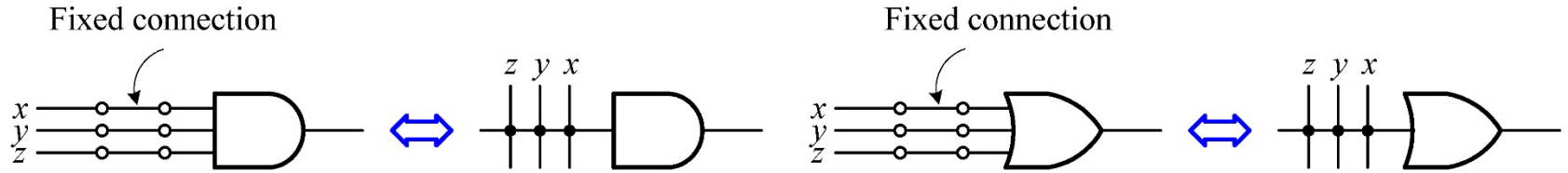
SHORTHAND NOTATIONS



(a) Before programming

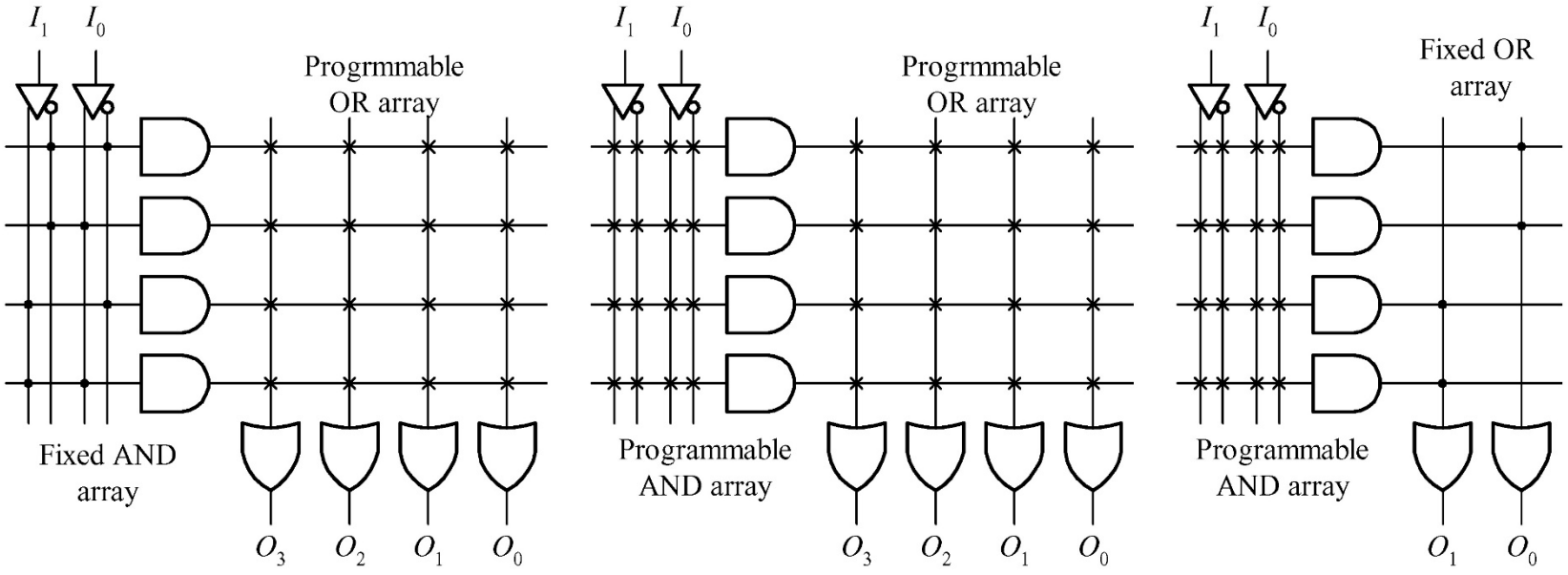


(b) After programming



(c) Fixed connection

PROGRAMMABLE LOGIC DEVICES

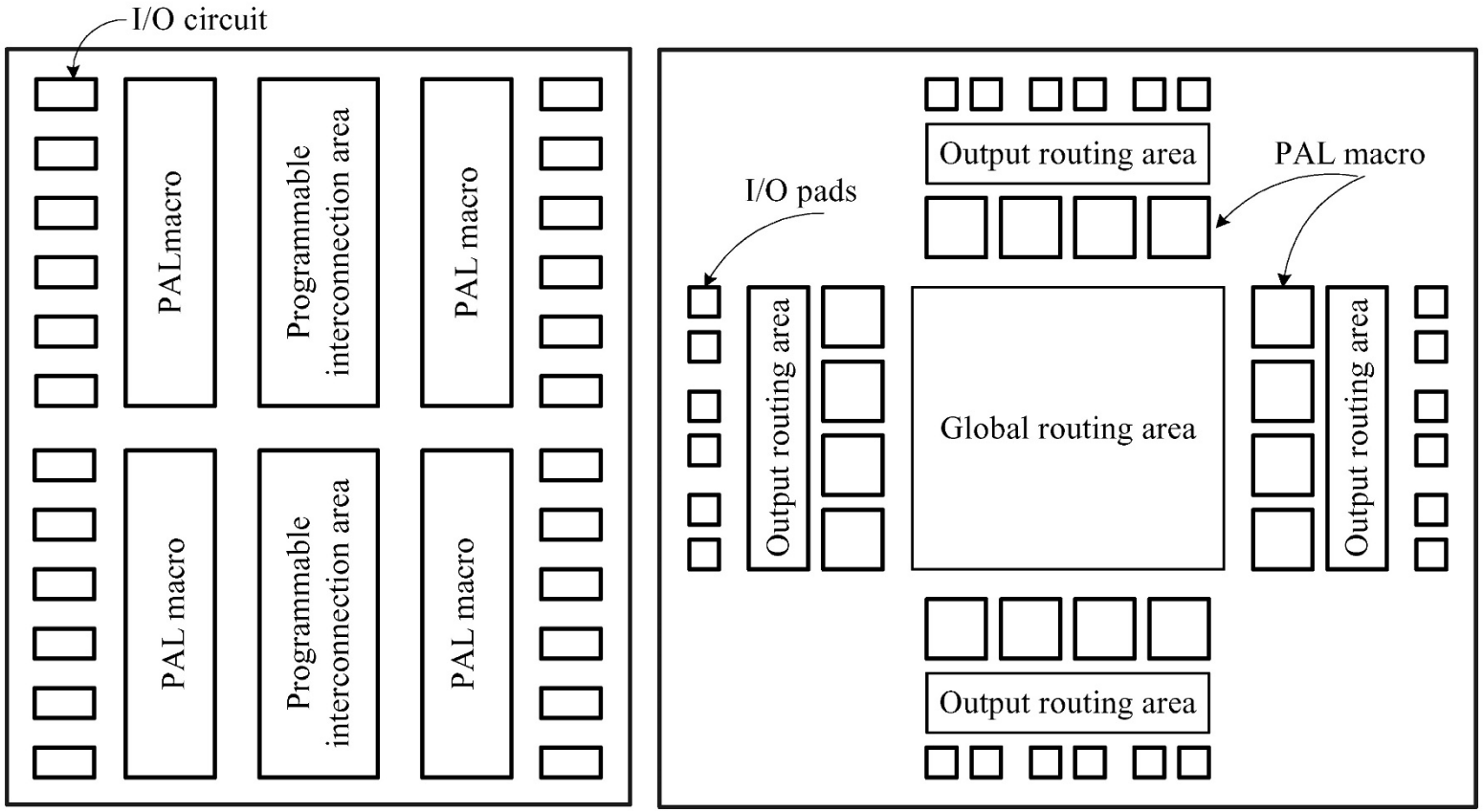


(a) ROM structure

(b) PLA structure

(c) PAL structure

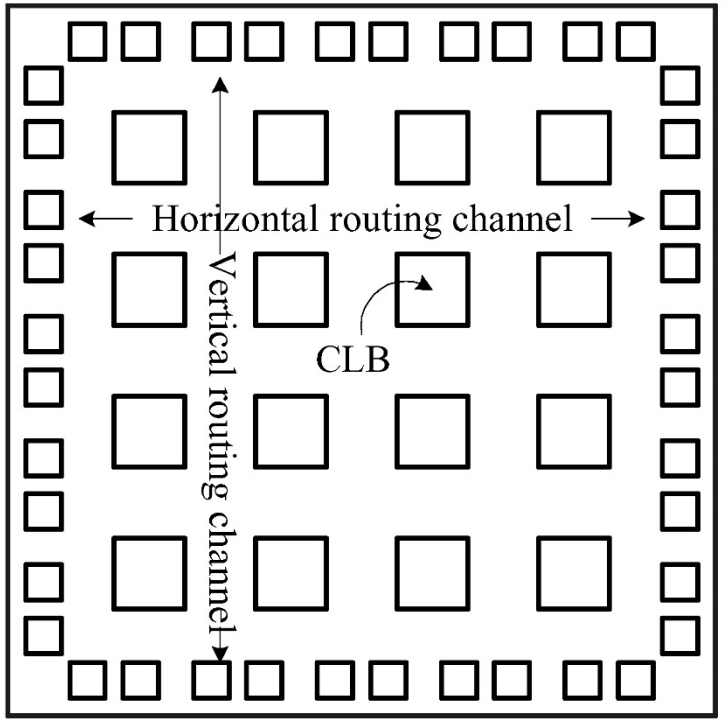
BASIC STRUCTURES OF CPLDS



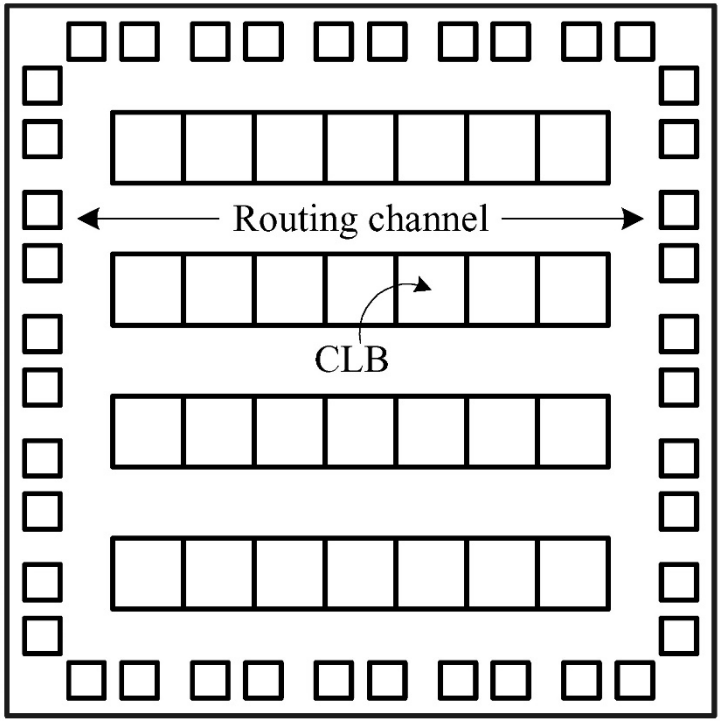
(a) CPLD basic structure

(b) pLSI basic structure

BASIC STRUCTURES OF FPGAS

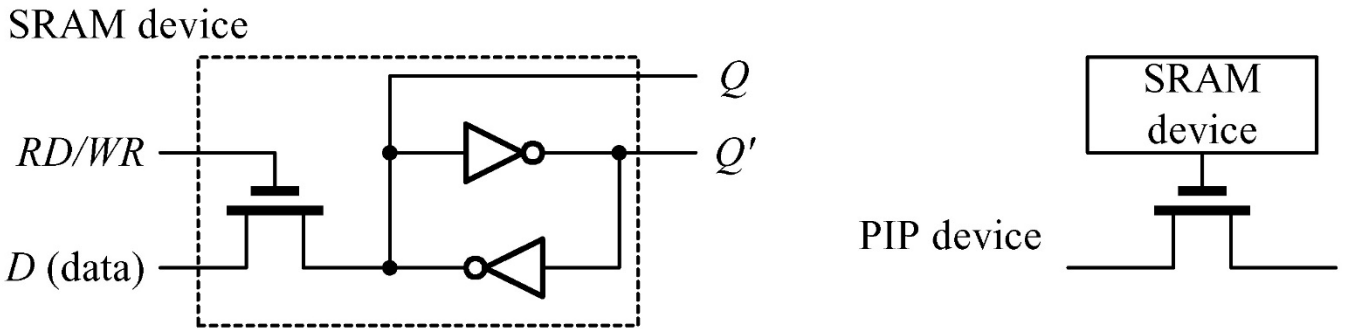


(a) Matrix type

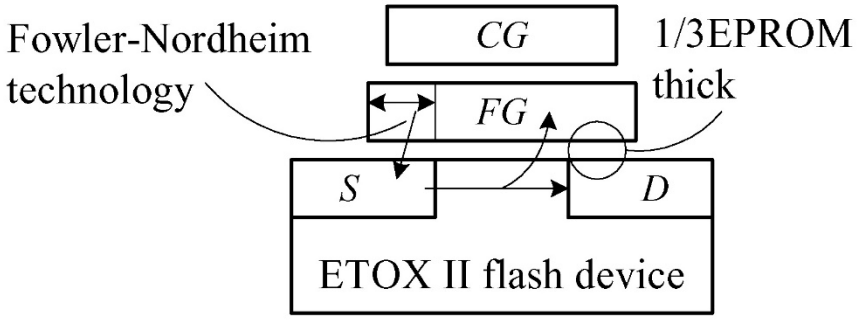


(b) Row type

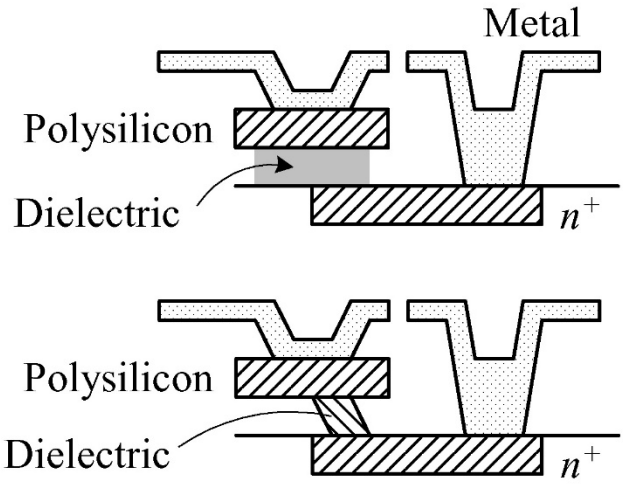
BASIC STRUCTURES OF PICS



(a) SRAM device



(b) Flash device



(c) Antifuse device

COMPARISON OF PICS

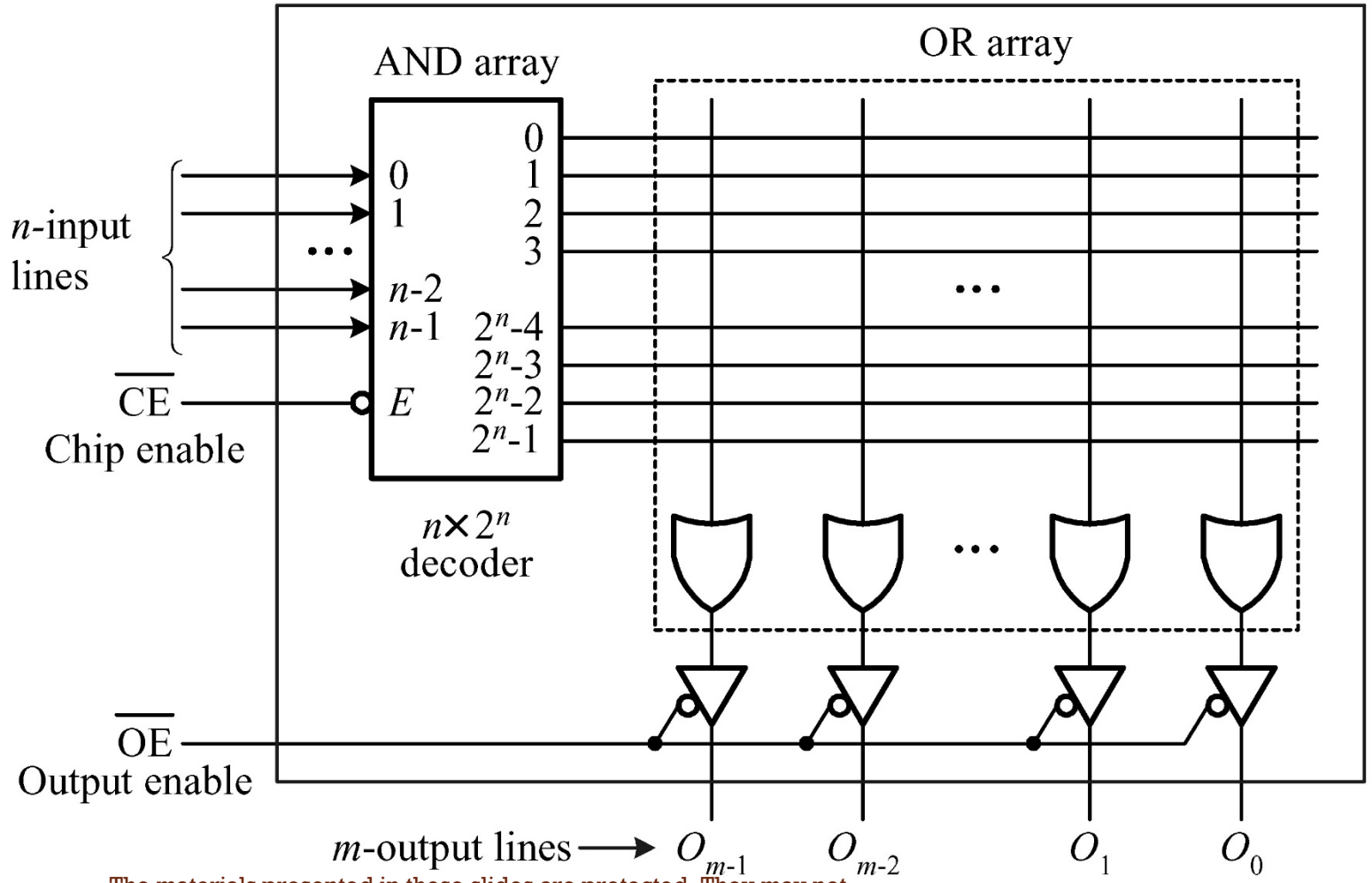
	SRAM	Flash	Antifuse
Process technology	Standard CMOS	Standard 2-level polysilicon	New type polysilicon
Programming approach	Shift register	FAMOS	Avalanche
Cell area	Very large	Large	Small
Resistance	≈ 2 kΩ	≈ 2 kΩ	≈ 500 Ω
Capacitance	≈ 50 fF	≈ 50 fF	≈ 10 fF

SYLLABUS

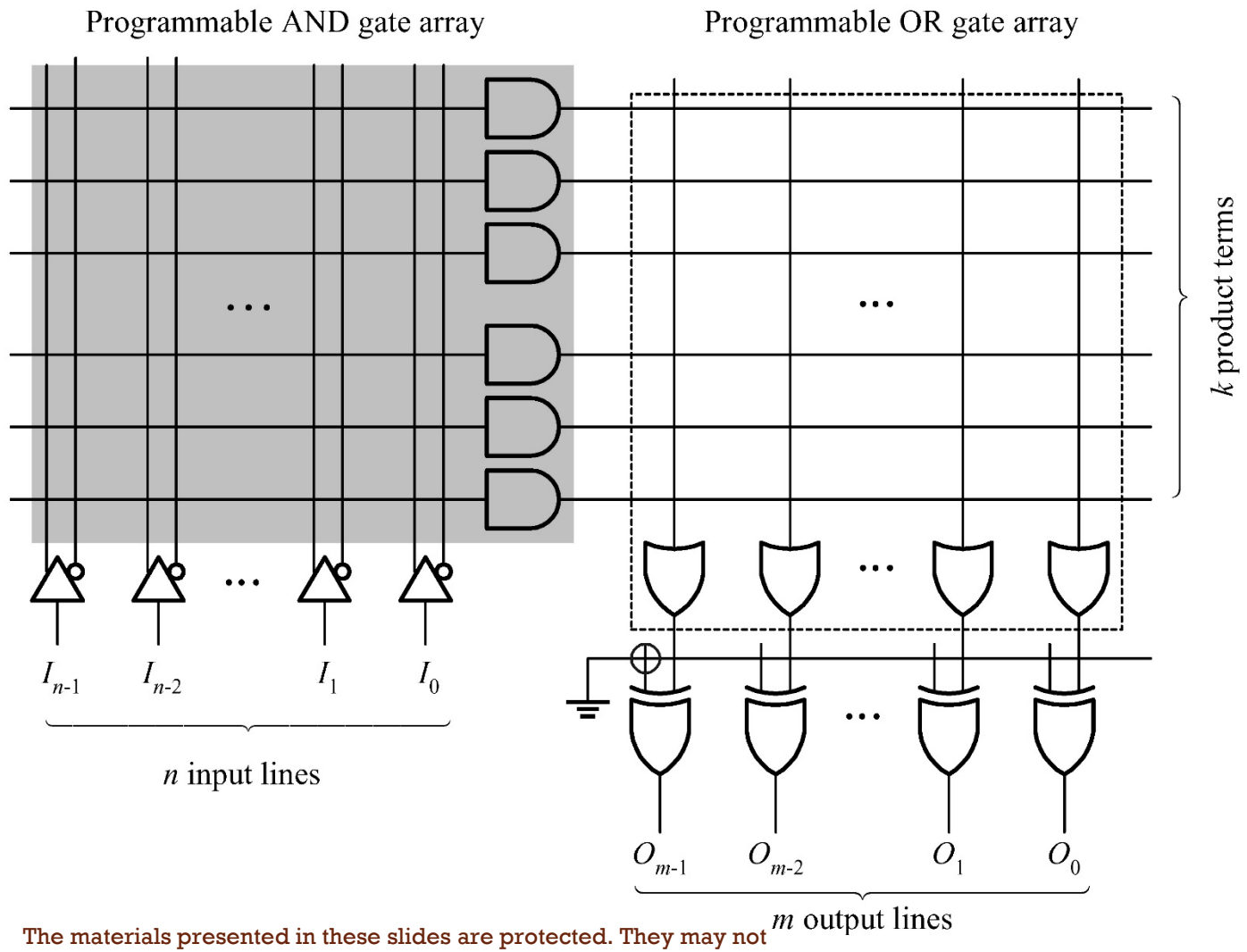
- Objectives
- Design options of digital systems
- PLD modeling
  - ROM /PLA /PAL
  - PLA modeling
- CPLD
- FPGA
- Practical issues

PLD

BASIC STRUCTURES OF ROMS



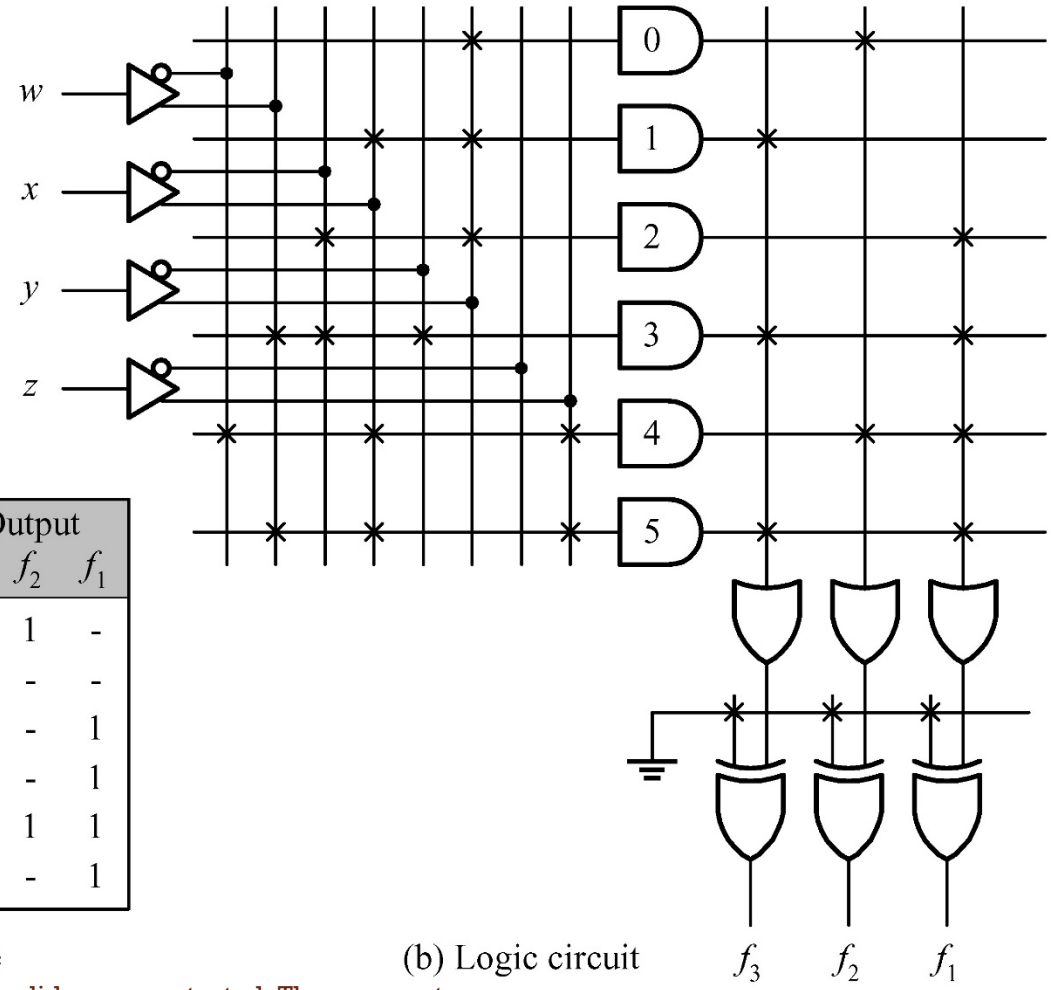
BASIC STRUCTURES OF PLAS



BASIC APPLICATIONS OF PLAS

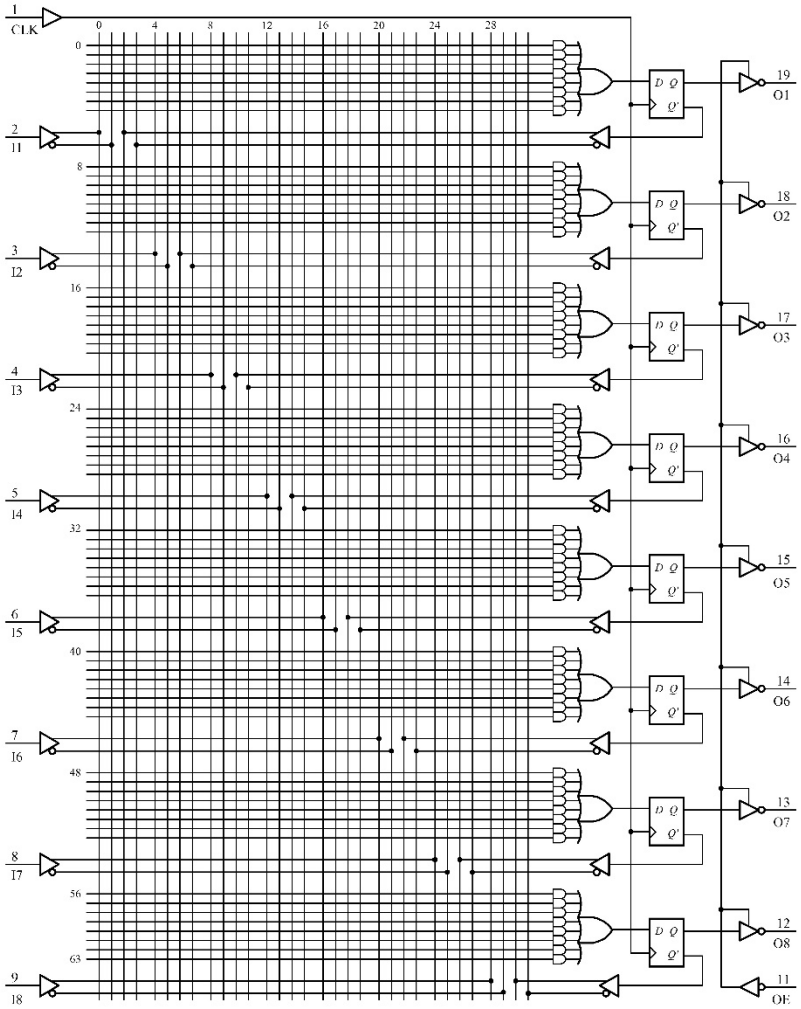
Product term		Input w	x	y	z	Output f_3	f_2	f_1
y	0	-	-	1	-	-	1	-
xy	1	-	1	1	-	1	-	-
$x'y$	2	-	0	1	-	-	-	1
$wx'y'$	3	1	0	0	-	1	-	1
$w'xz$	4	0	1	-	1	-	1	1
wxz	5	1	1	-	1	1	-	1

(a) PLA programming table



(b) Logic circuit

A PAL EXAMPLE --- 16R8



SYLLABUS

- Objectives
- Design options of digital systems
- PLD modeling
  - ROM /PLA /PAL
  - PLA modeling
- CPLD
- FPGA
- Practical issues

PLA MODELING

- PLA device structures
  - SOP (sum of product) = AND plane + OR plane
    - AND plus OR array
    - NAND plus NAND array
  - POS (product of sum) = OR plane + AND plane
    - OR plus AND array
    - NOR plus NOR array
- An AND plane or an OR plane → a personality array (or memory)

PLA MODELING

- A PLA device is modeled by
  - a group of system tasks
  - an appropriate combination of two system tasks
- A PLA device can be modeled as:
  - Asynchronous
  - Synchronous

PLA MODELING

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

LOGIC ARRAY PERSONALITY

- Defined as an array of reg
  - Width: the number of input terms
  - Depth: the number of output terms
- Loaded into memory using
  - System tasks $readmemb or $readmemh
  - The procedural assignment statements
- Can be changed dynamically during simulation

PLA MODELING

- The array format
  - 1: taken
  - 0: not taken
- The plane format (complies with Espresso)
  - 1: true
  - 0: complement
  - ? or z: don't care

PLA MODELING

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

```
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
…
```

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

1100000
0011100
0000111

PLA MODELING – THE ARRAY FORMAT

- Define PLA personality using procedural assignment statements
  - In this example only one and plane is considered

```
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
```

PLA MODELING

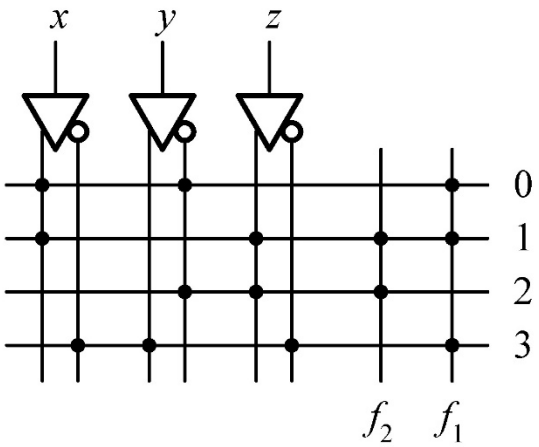
A = {a0, a1, a2, a3, a4, a5, a6, a7}
B = {b0, b1, b2}
mem[0] = 8'b11001100;
mem[1] = 8'b00110011;
mem[2] = 8'b00001111;

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

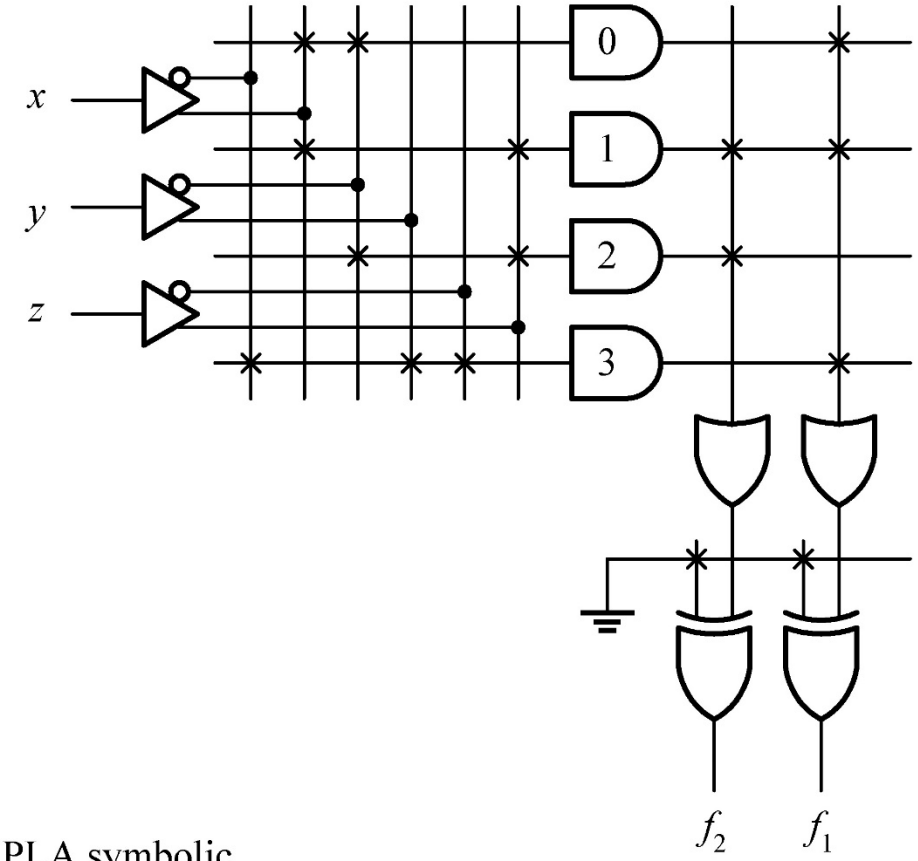
A PLA MODELING EXAMPLE --- LOGIC CIRCUIT

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



(c) Logic circuit

A PLA MODELING EXAMPLE – THE ARRAY FORMAT

```
// 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;   // 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;        // OR plane
    mem_or[2] = 4'b0110;
    $async$or$array(mem_or, {p0, p1, p2, p3}, {f1, f2});
end
```

```
# 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

```
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;
   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?;
end
```

```
# 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
```

INITIALIZING MEMORY FROM FILES

$readmemb("<file_name>",<memory_name>);
$readmemb("<file_name>",<memory_name>, <start_addr>);
$readmemb("<file_name>",<memory_name>, <start_addr>, <end_addr>);

$readmemh("<file_name>",<memory_name>);
$readmemh("<file_name>",<memory_name>, <start_addr>);
$readmemh("<file_name>",<memory_name>, <start_addr>, <end_addr>);

```
reg  [7:0] mem[1:256];

initial $readmemb("mem.data", mem);
initial $readmemb("mem.data", mem, 16);
initial $readmemb("mem.data", mem, 128, 1);
```

DATA FILE FORMATS

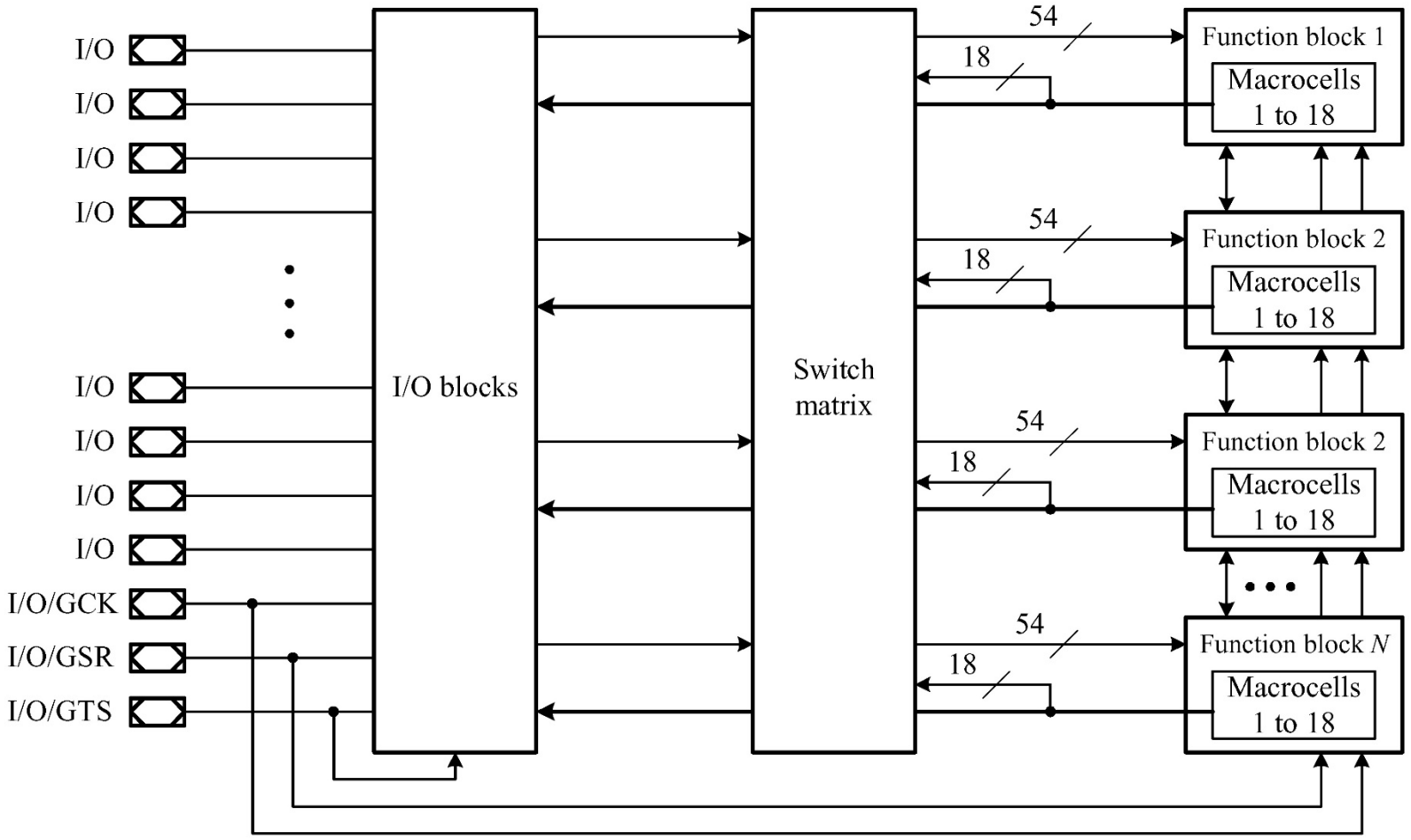
- Addresses
  - specified with @<address>
  - specified as hexadecimal numbers
- Data
  - can contain x or z and is separated by white spaces
- Uninitialized locations default to x

For example:

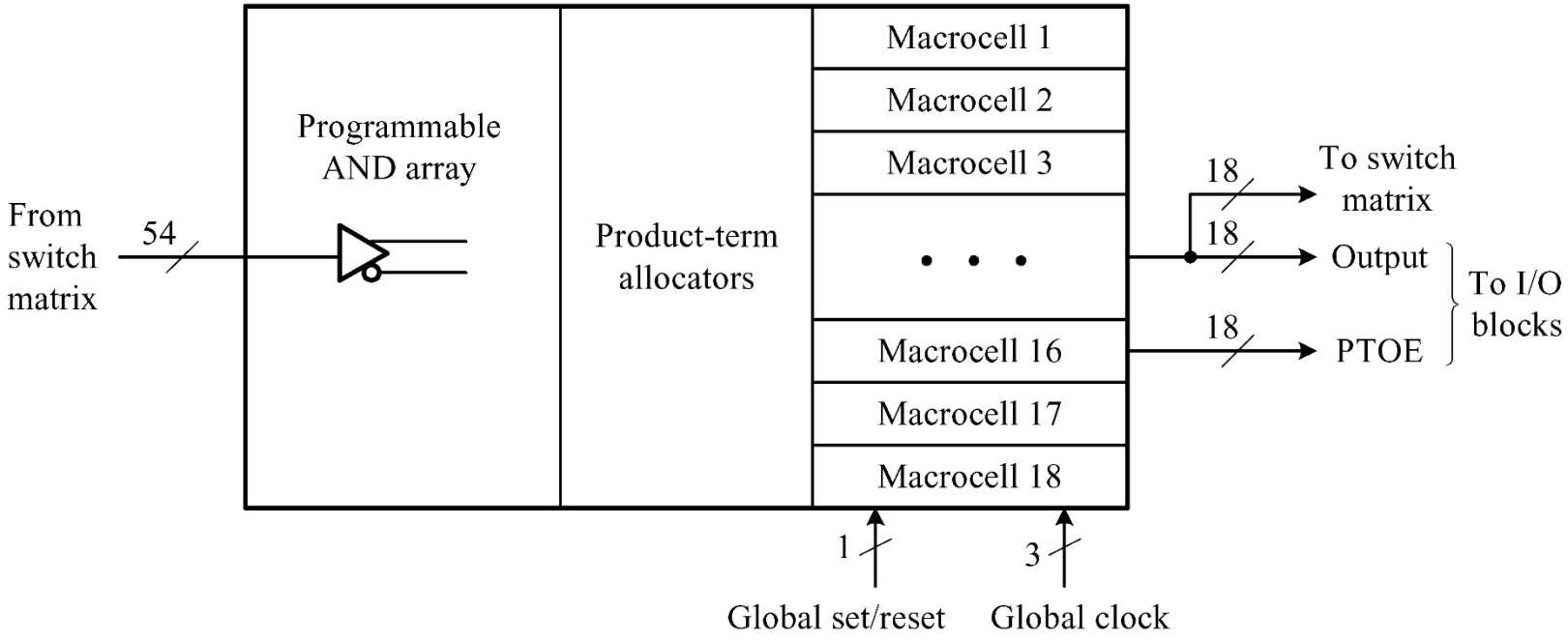
```
@002    // data will be read into memory from location 2
0000 1011
0001 11zz
@008    // data will be read into memory from location 8
1111 zz1z
```

CPLD (COMPLEX PROGRAMMABLE LOGIC DEVICE)

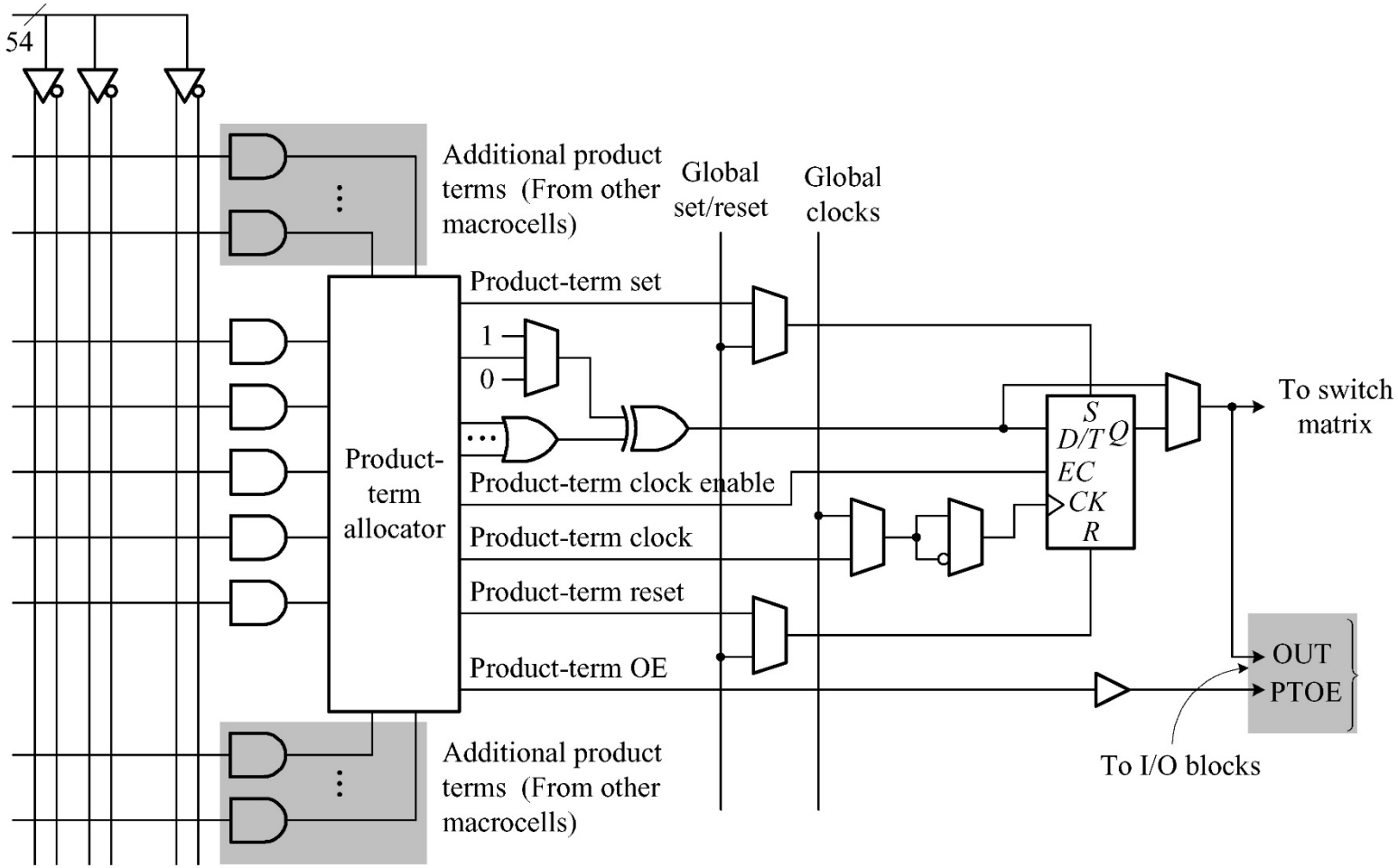
BASIC STRUCTURE OF CPLD: XC9500XL



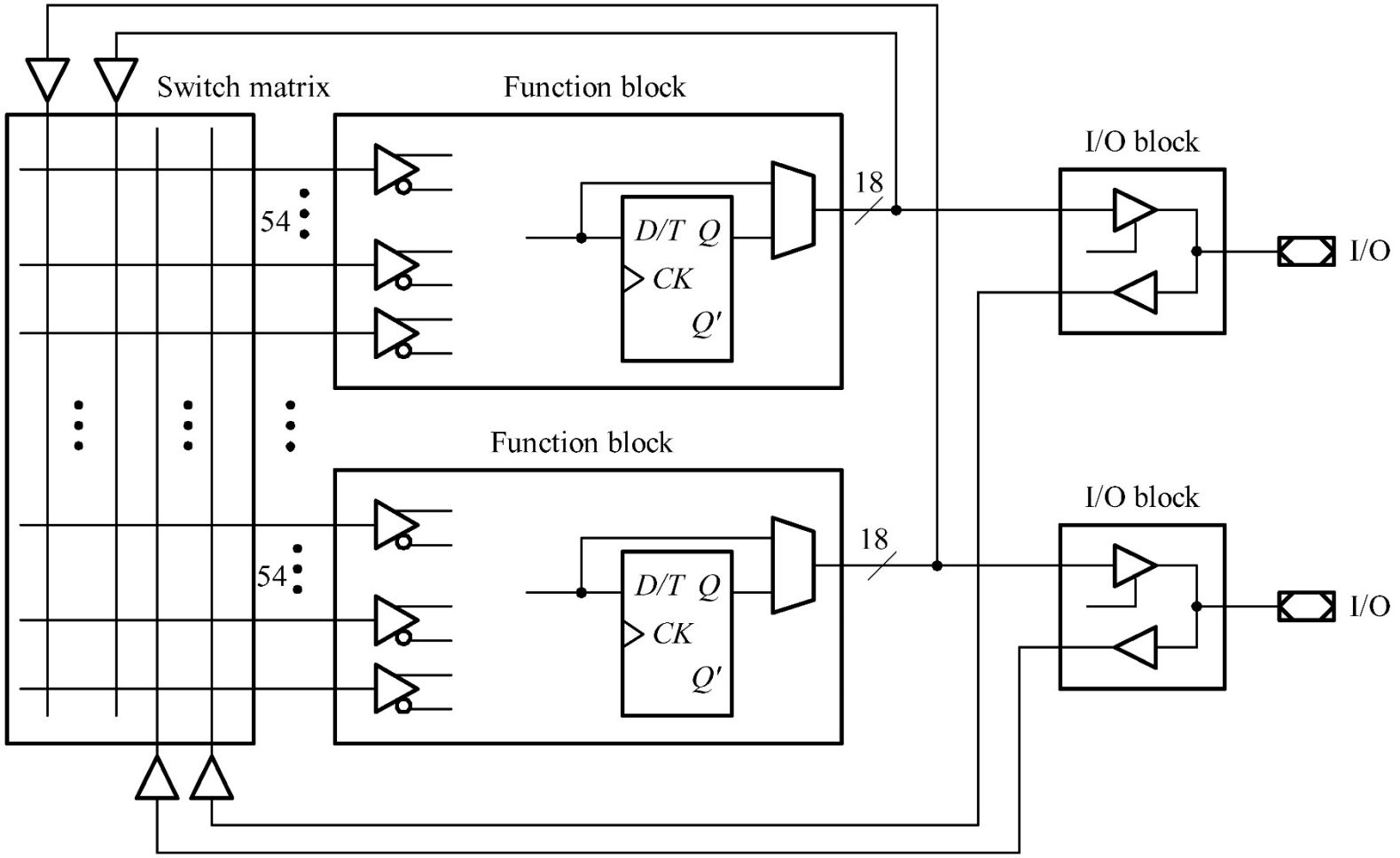
CPLD: XC9500XL --- FUNCTION BLOCK



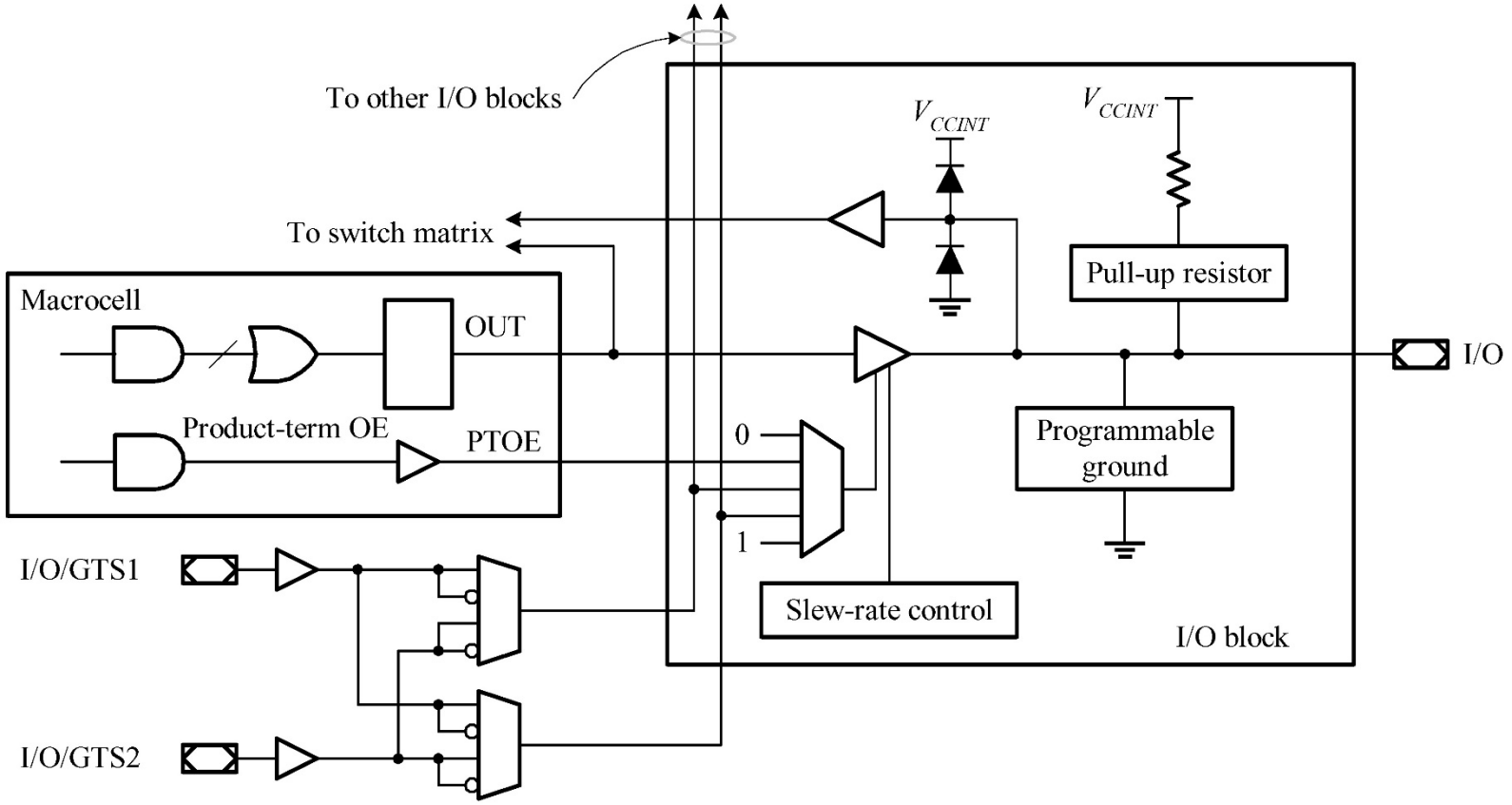
CPLD: XC9500XL --- MACRO OF FUNCTION BLOCK



CPLD: XC9500XL --- SWITCH MATRIX



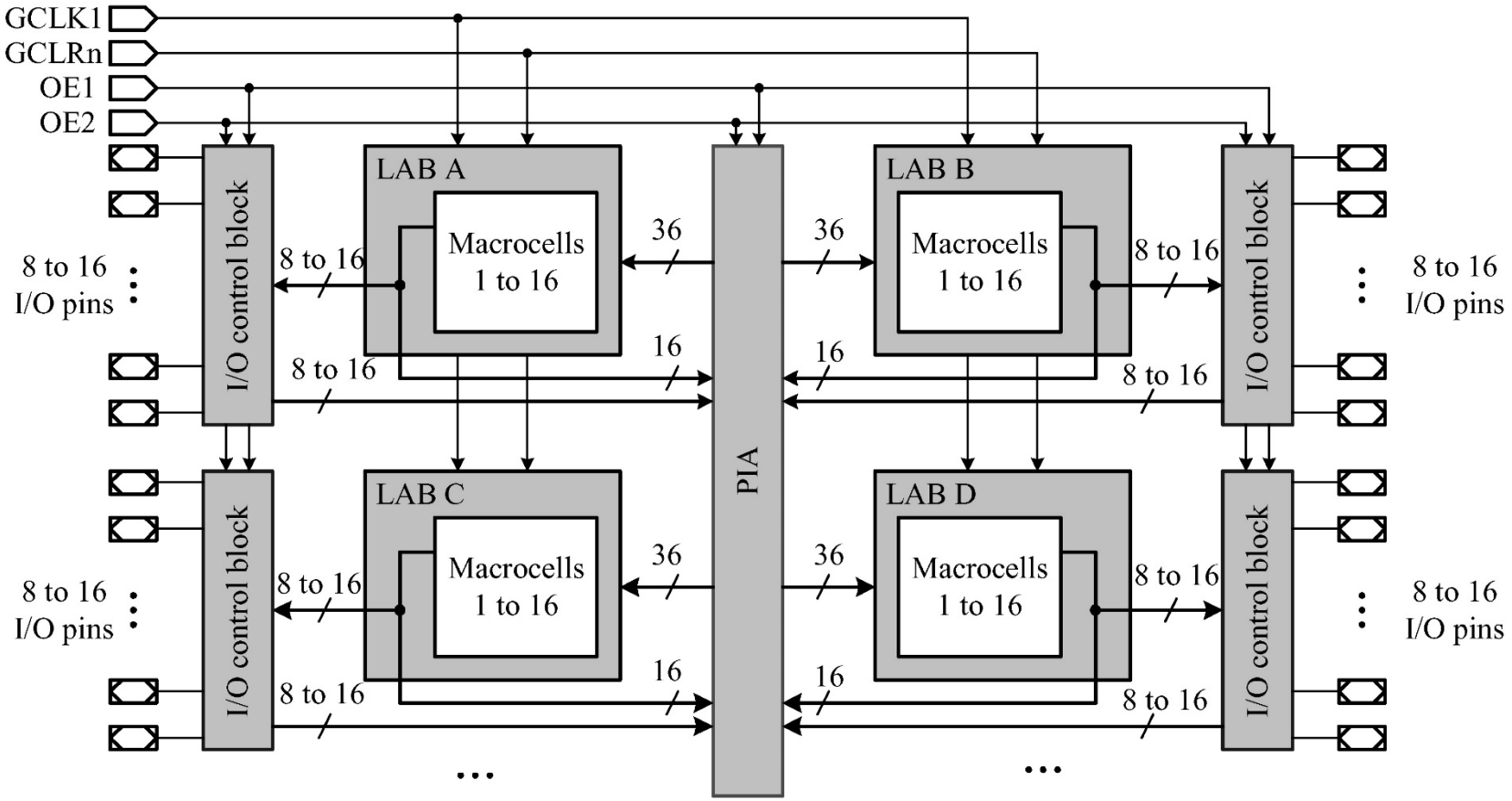
CPLD: XC9500XL --- I/O BLOCK



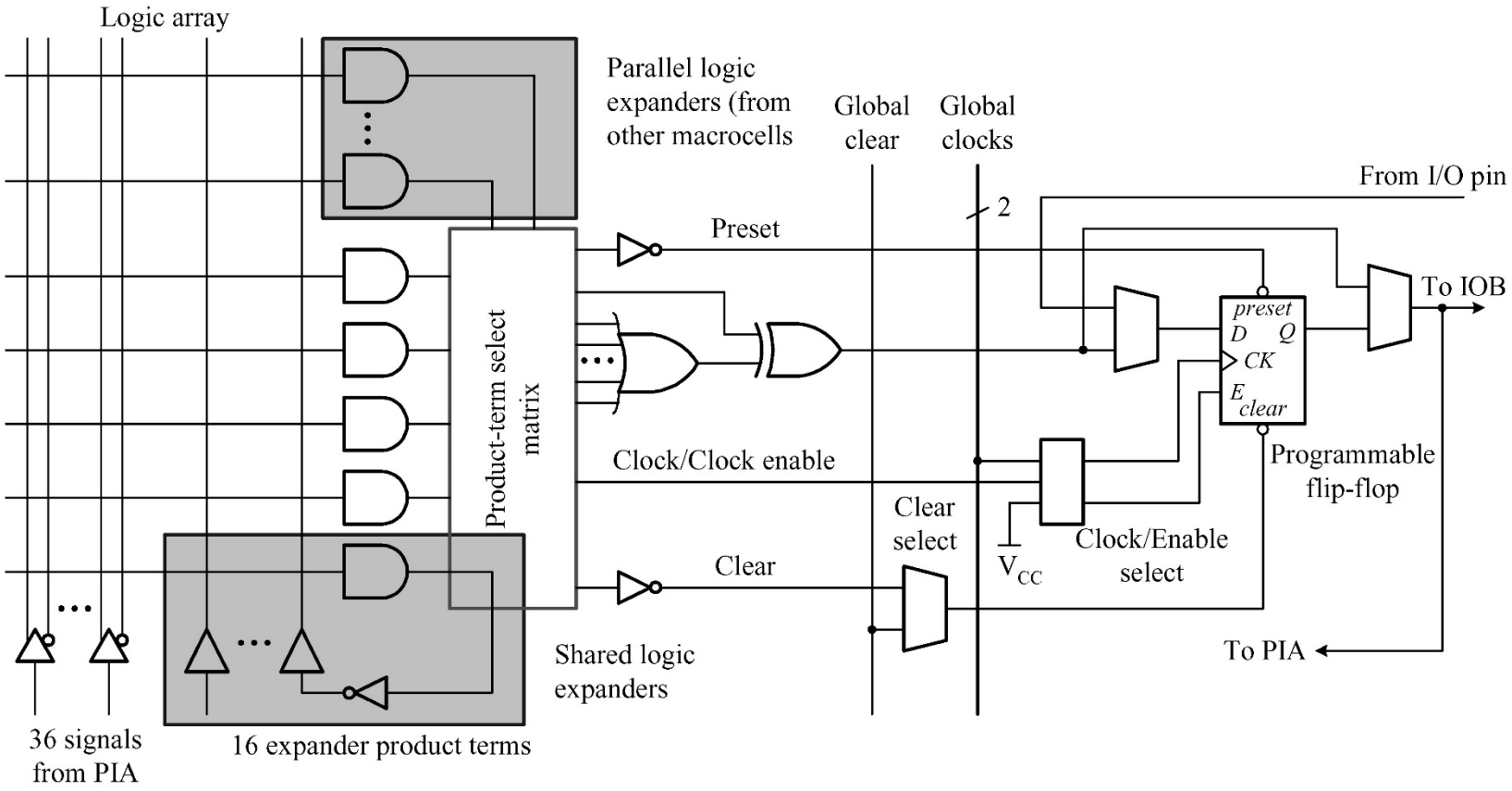
SYLLABUS

- Objectives
- Design options of digital systems
- PLD modeling
- CPLD
  - XC9500 family
  - MAX7000 family
- FPGA
- Practical issues

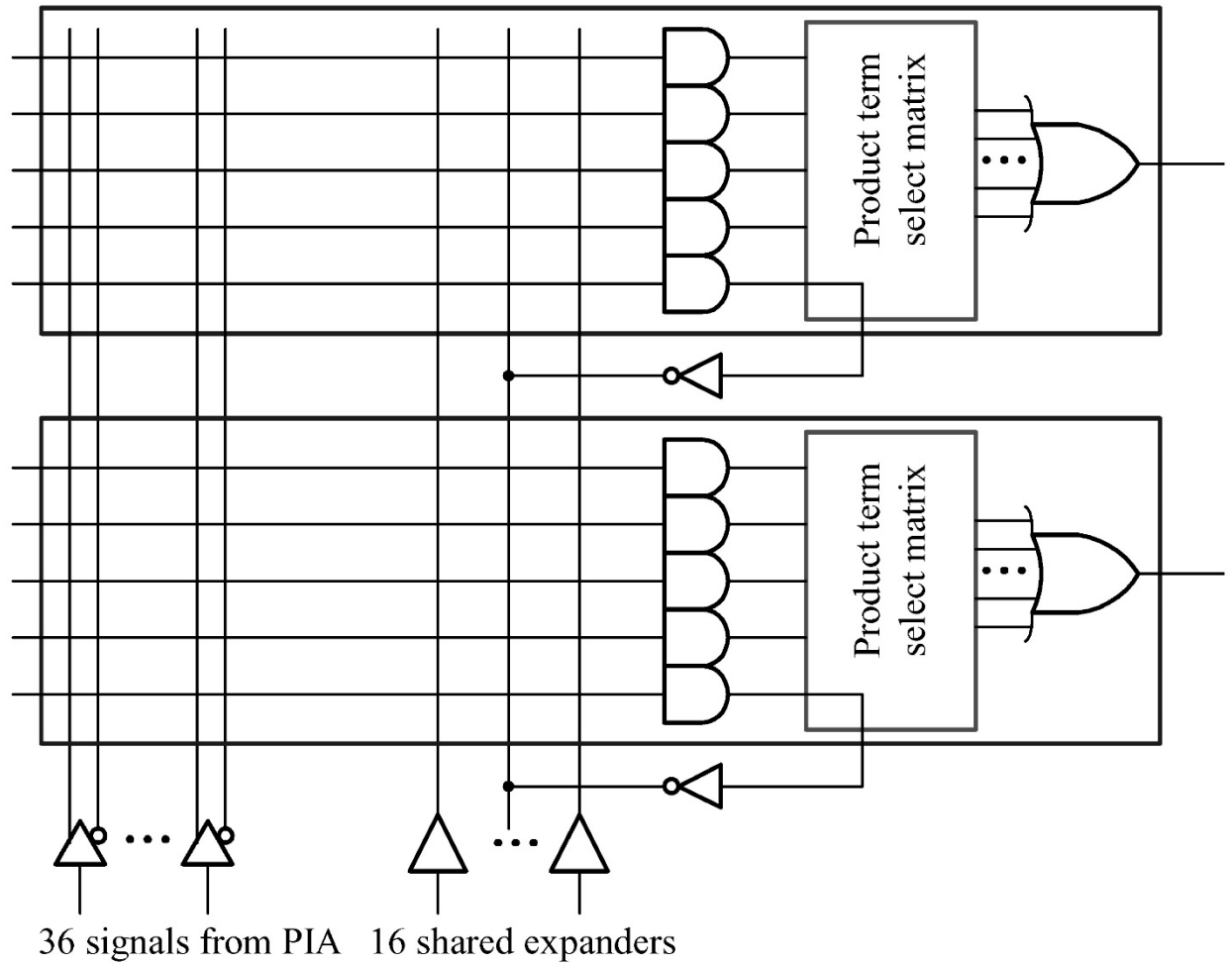
MAX7000 FAMILY --- BASIC STRUCTURE



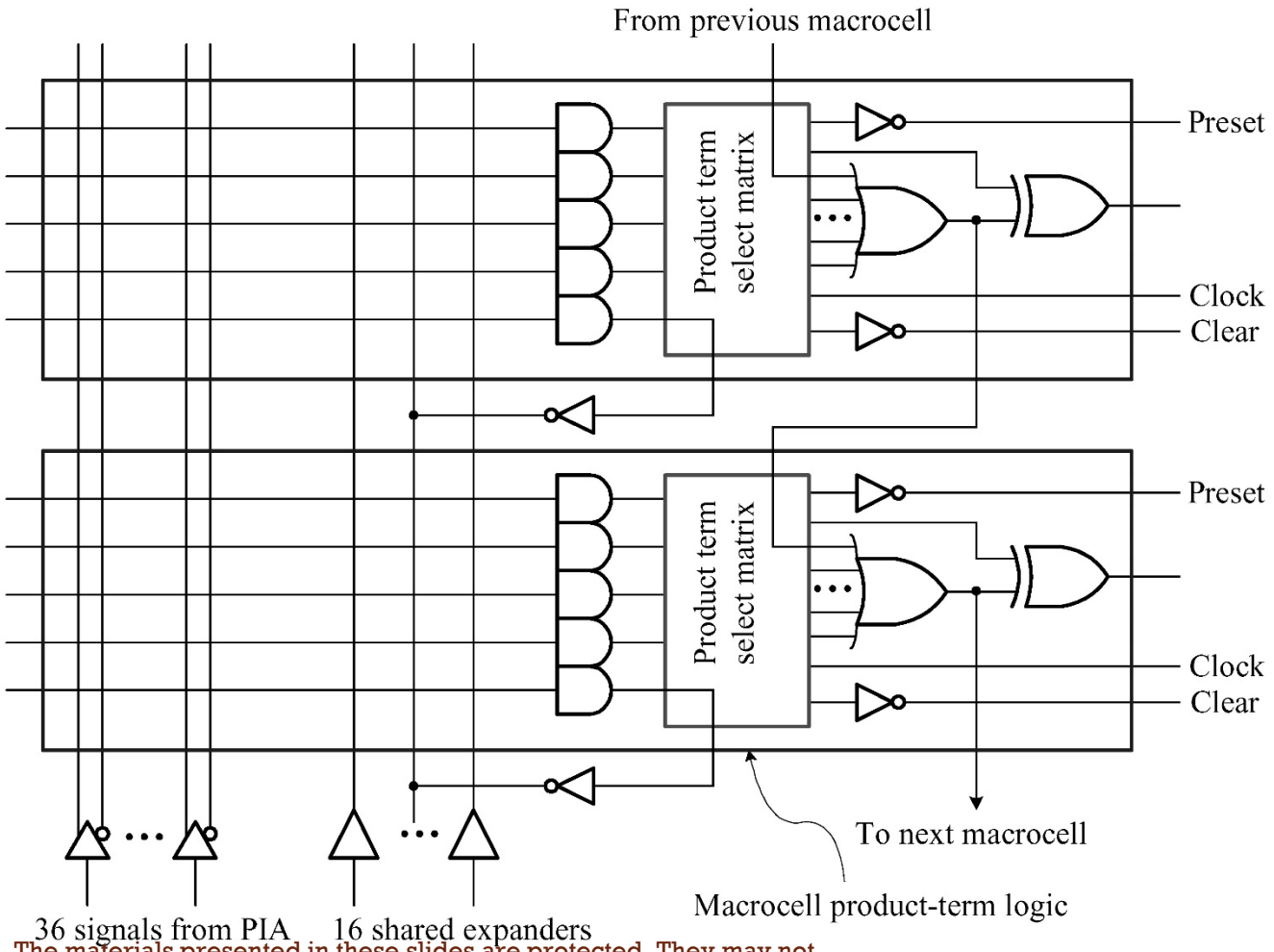
MAX7000 FAMILY --- MACROCELL STRUCTURE



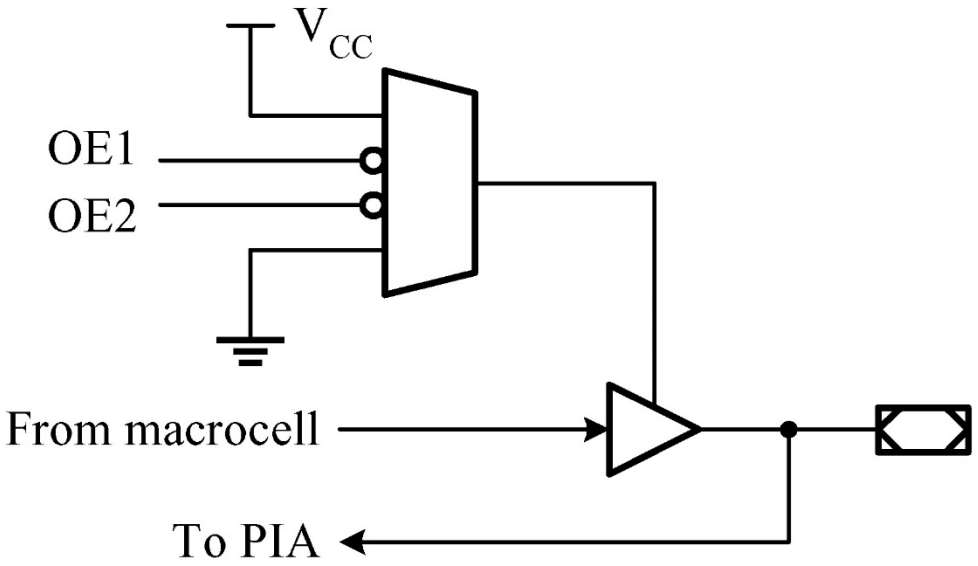
MAX7000 FAMILY --- SHARABLE EXPANDER



MAX7000 FAMILY --- PARALLEL EXPANDER

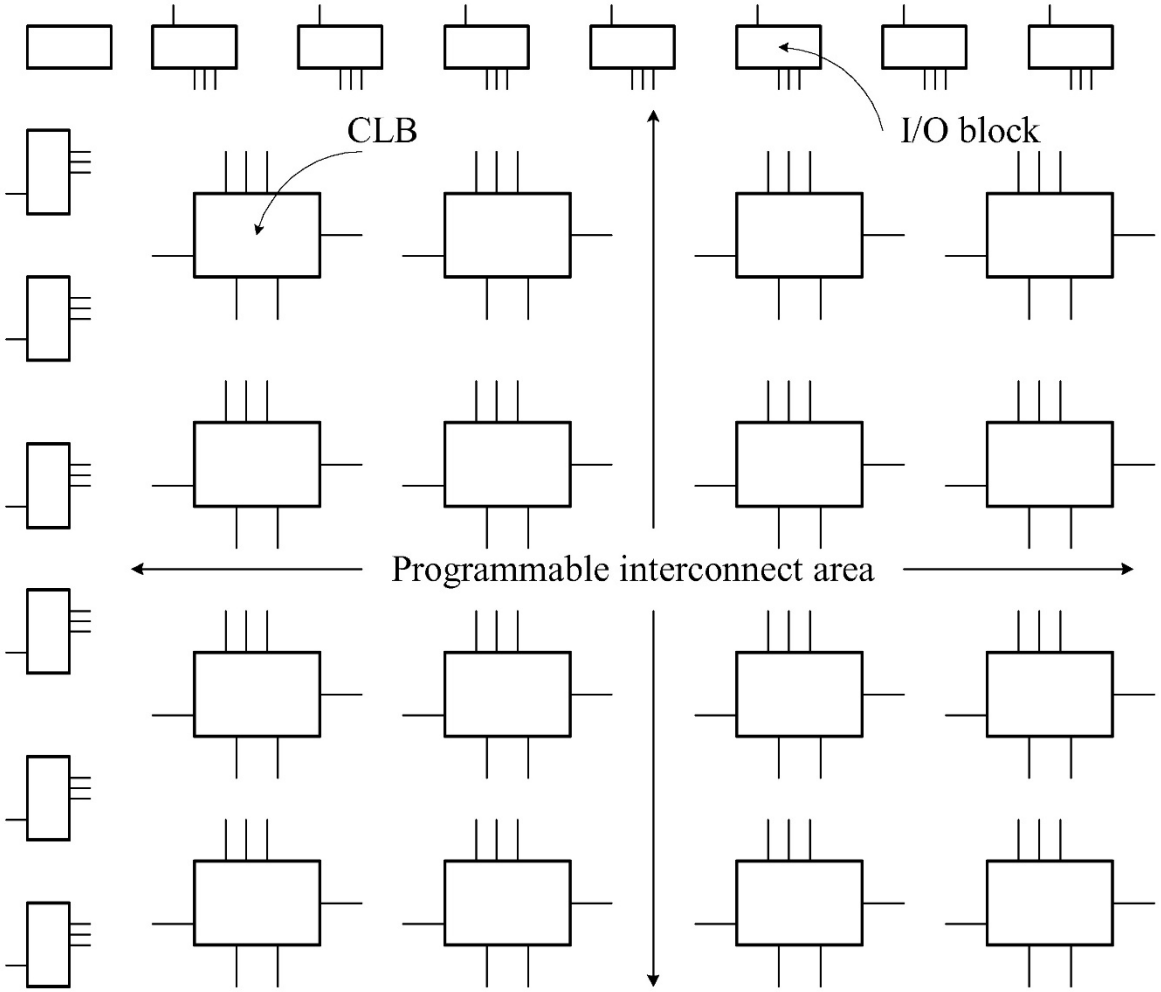


MAX7000 FAMILY --- IOB

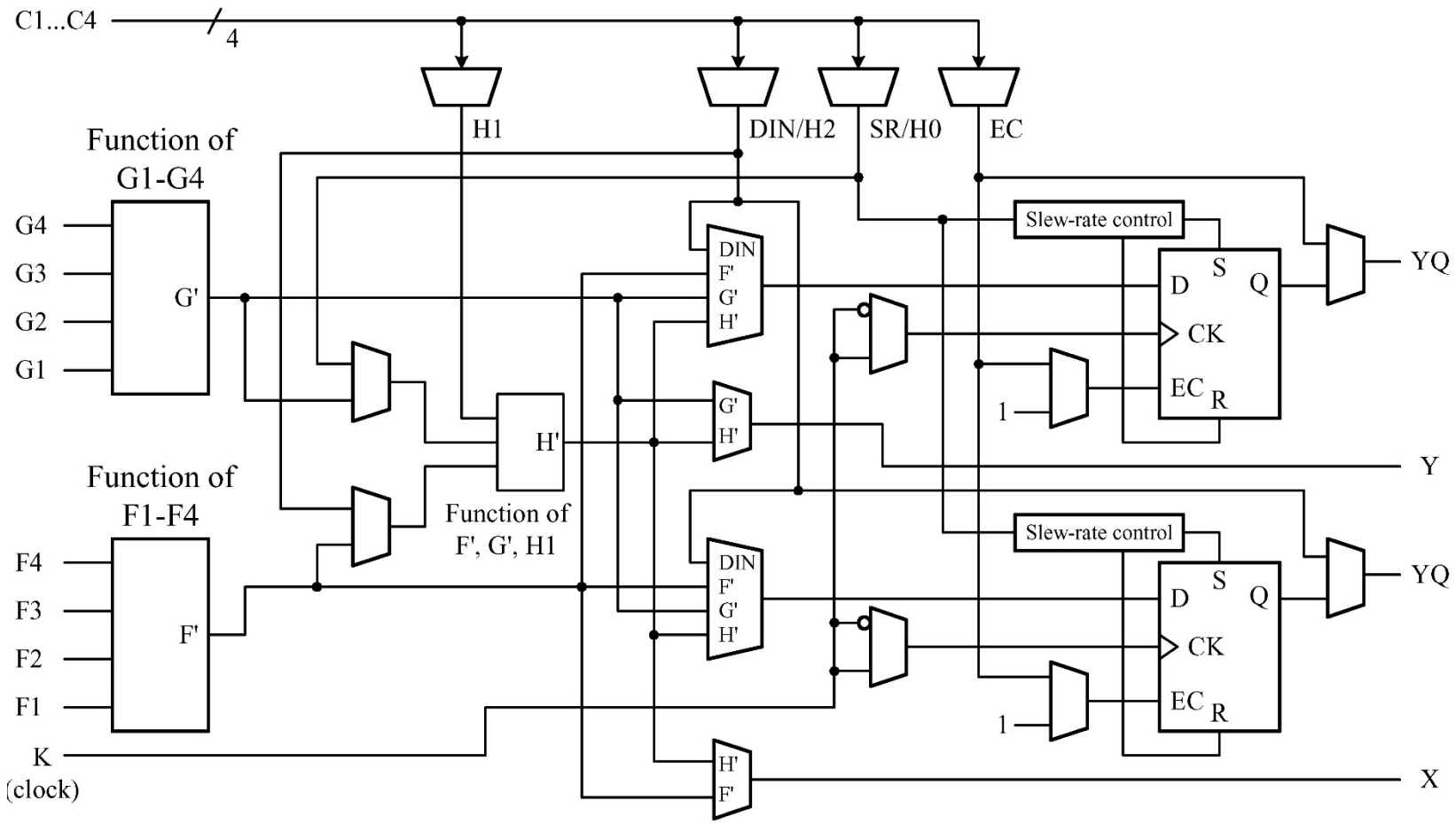


FPGA

BASIC STRUCTURE OF FPGA: XC4000XL

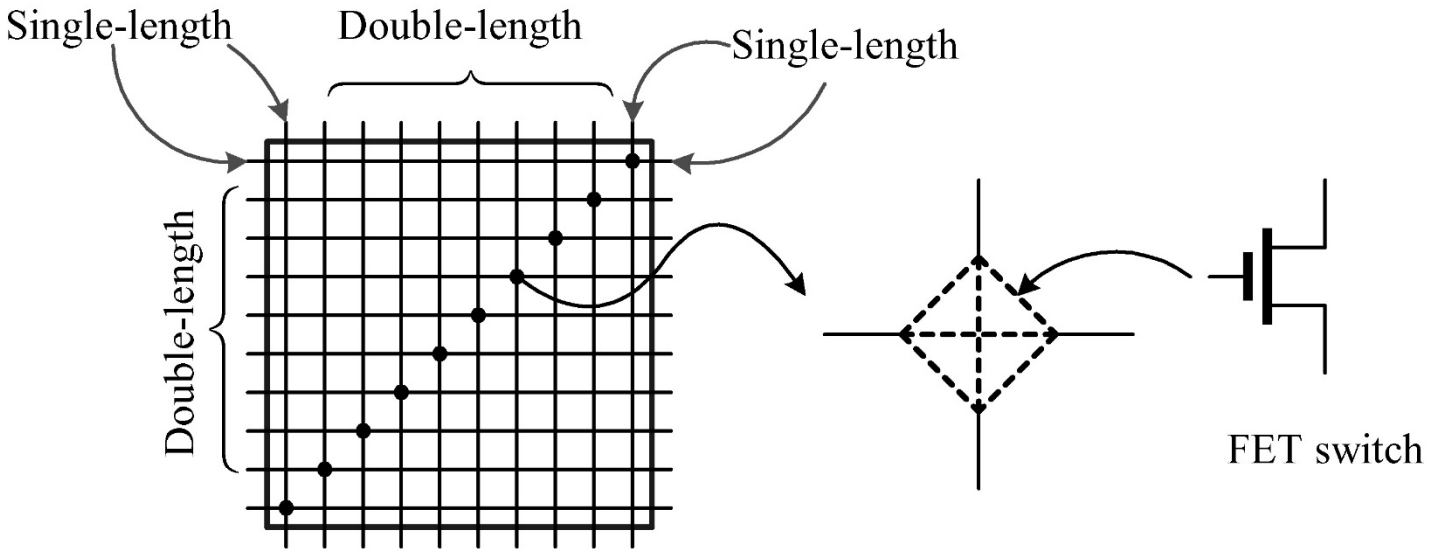


FPGA: XC4000XL --- CONFIGURABLE LOGIC BLOCK

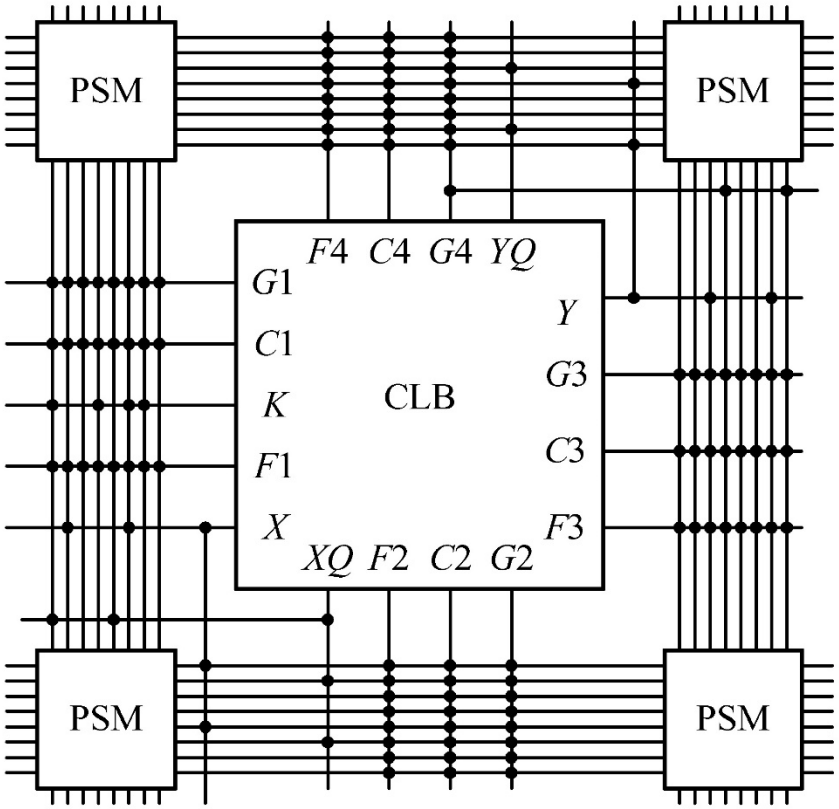


FPGA: XC4000XL --- PIA

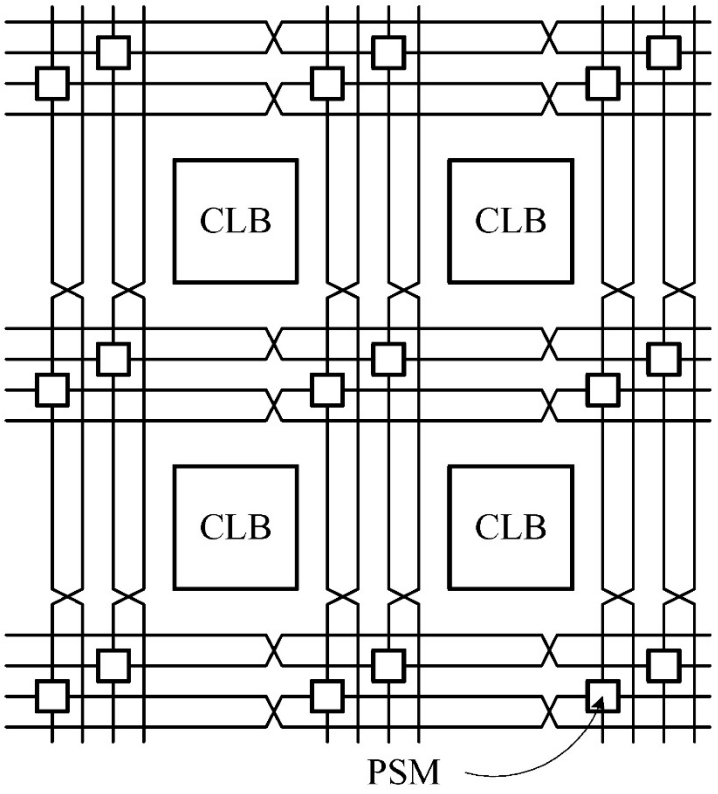
- Interconnection lines:
  - Single-length lines
  - Double-length lines
  - Global long-length lines



FPGA: XC4000XL --- PIA

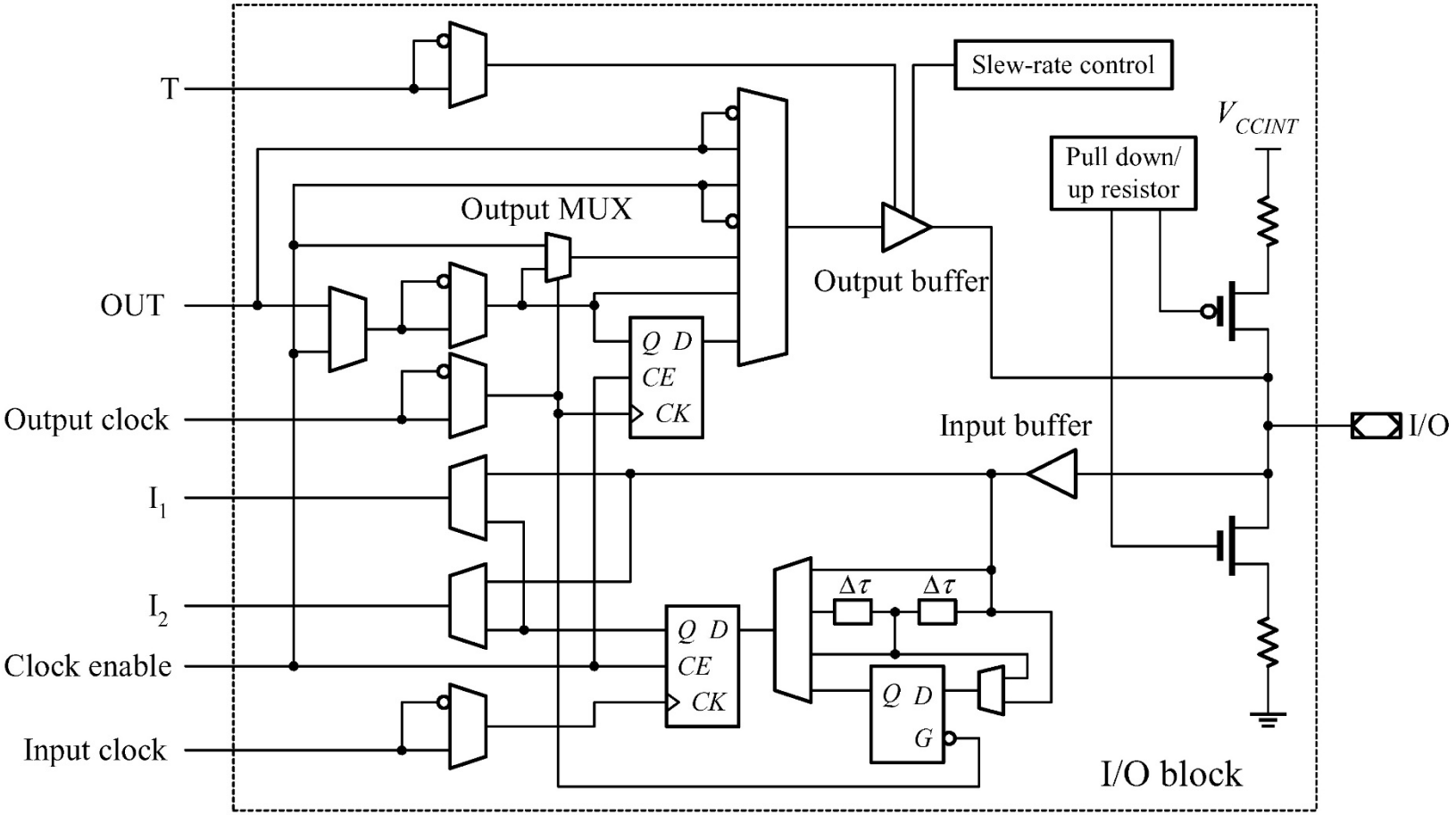


(a) Single-length lines



(b) Double-length lines

FPGA: XC4000XL --- INPUT/OUTPUT BLOCK

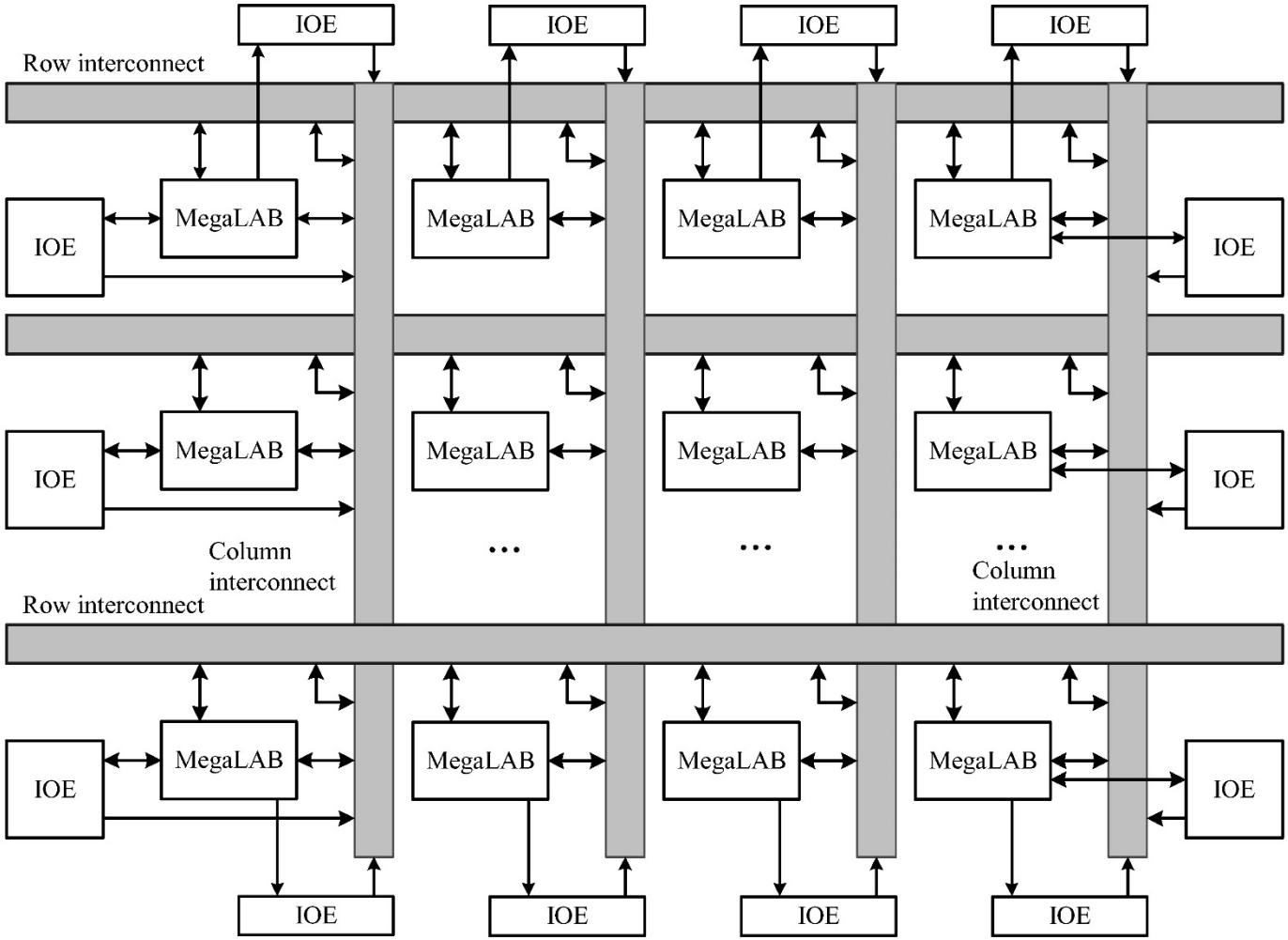


FPGA: XC4000XL --- MACRO LIBRARY EXAMPLE

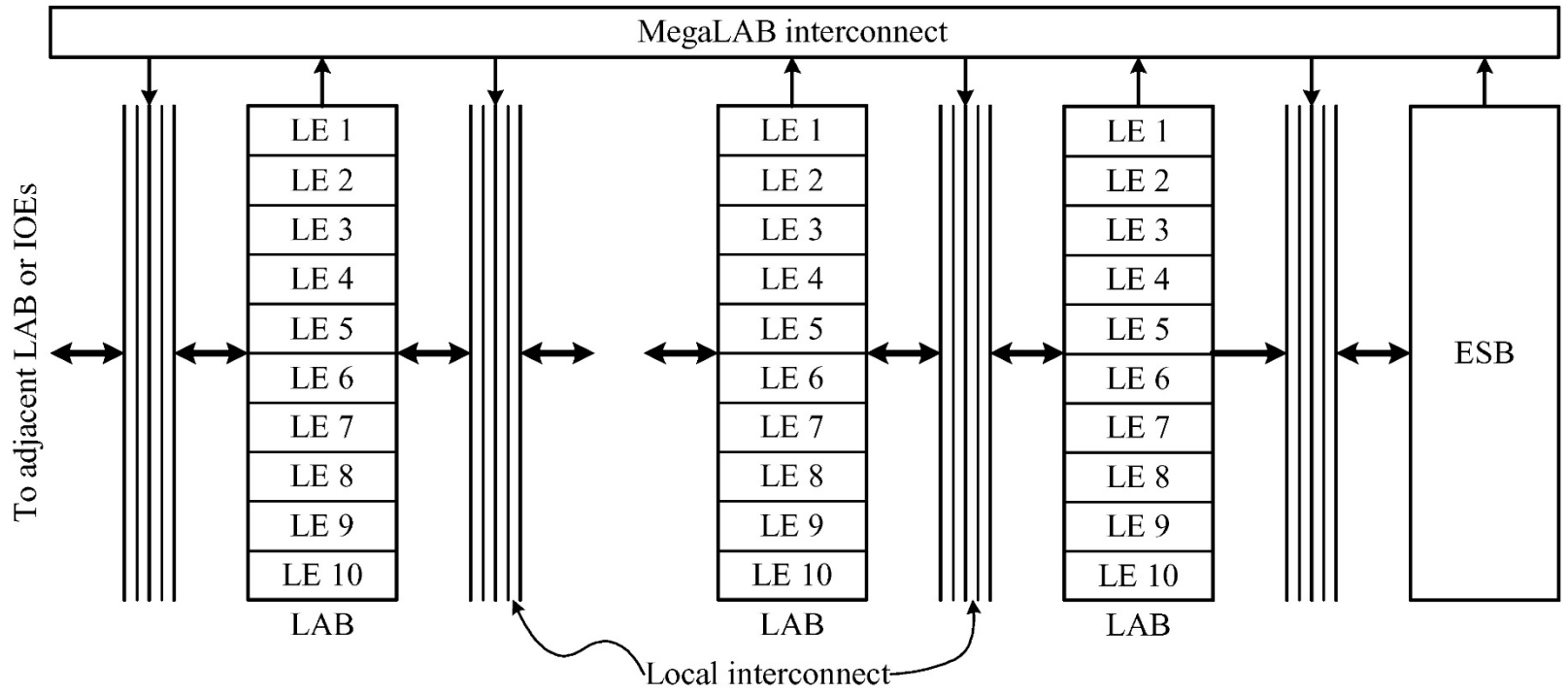
- Types of macros
  - Soft macros
  - Hard macros

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		

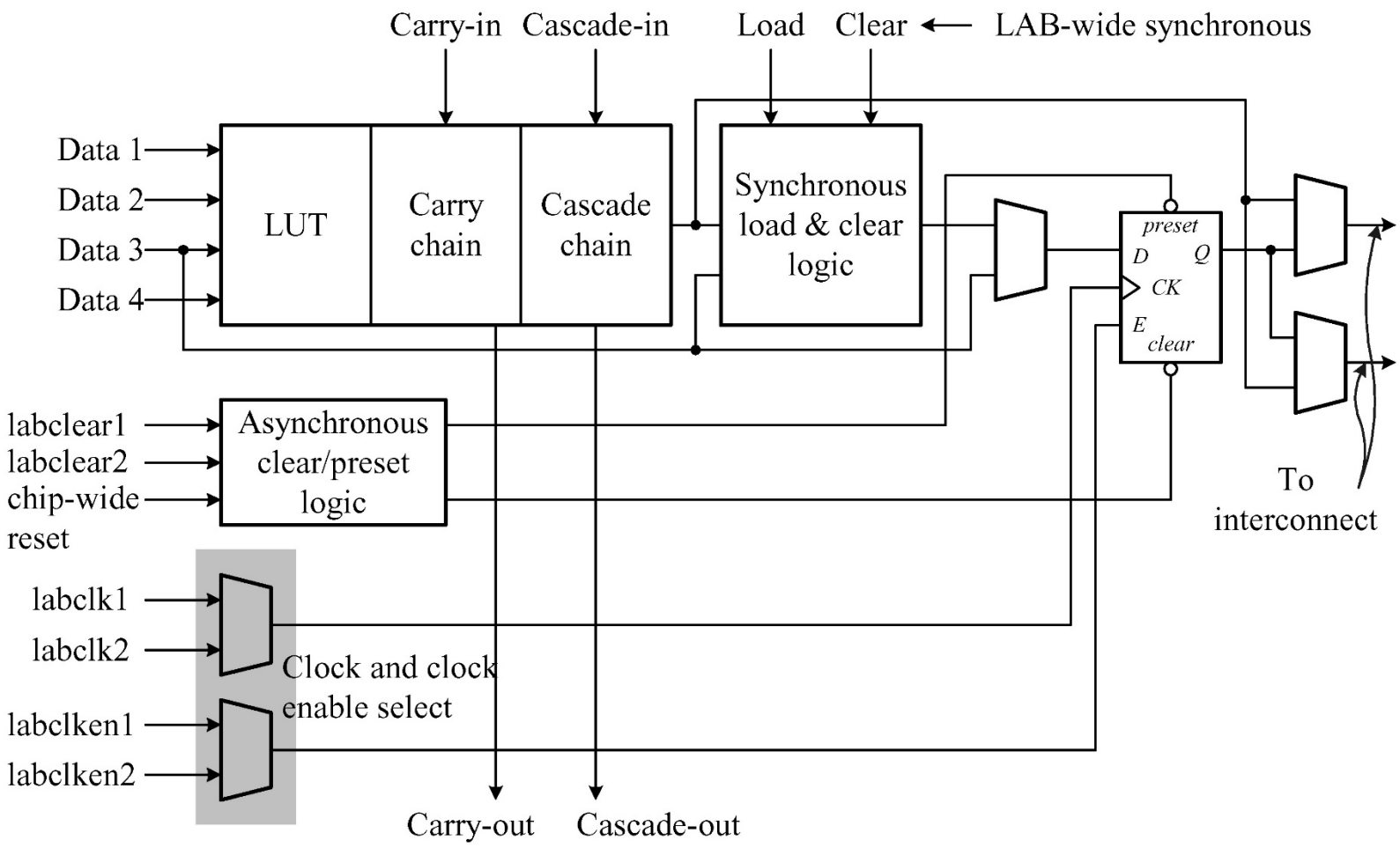
APEX 20K FAMILY --- BASIC STRUCTURE



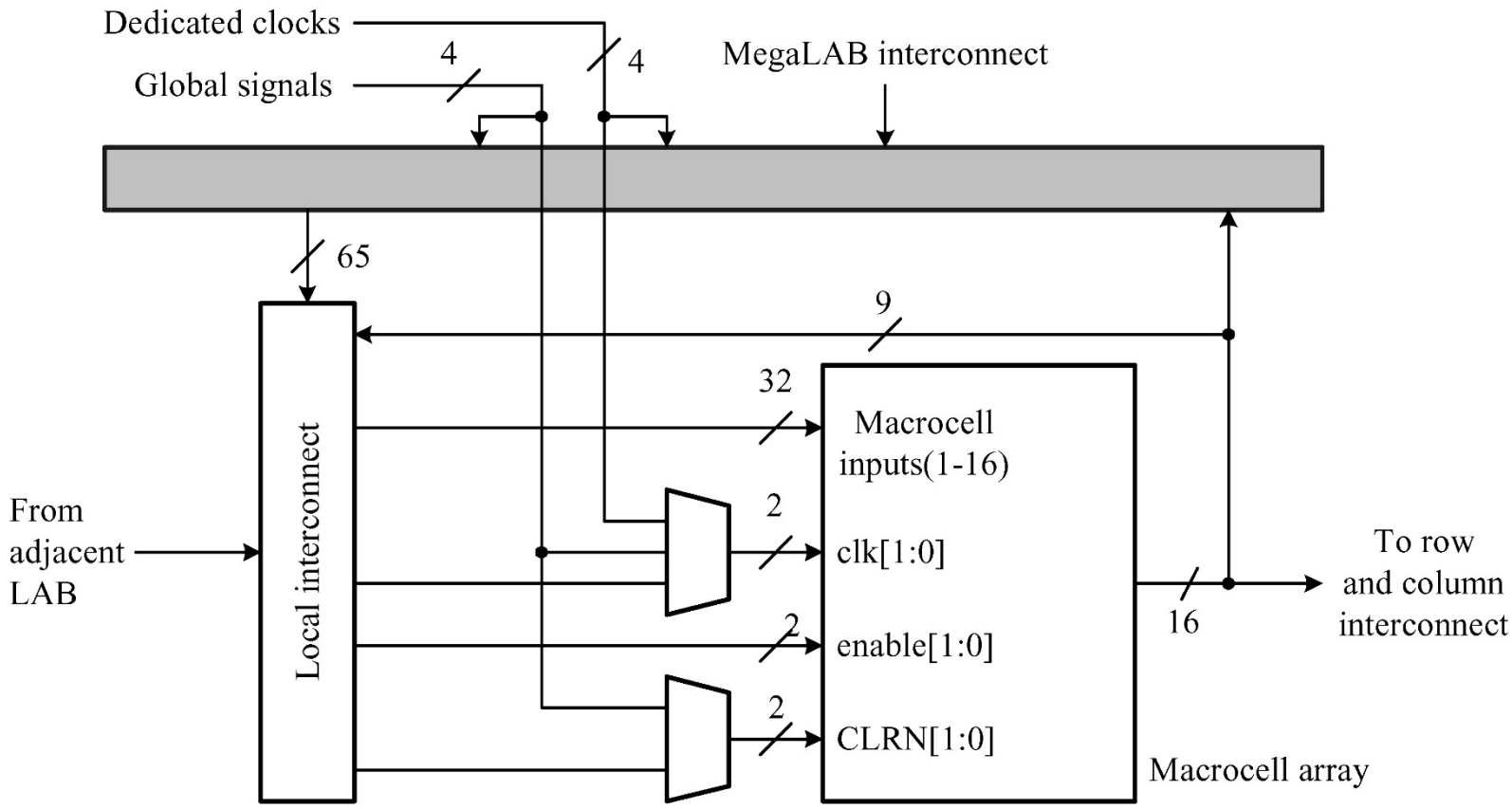
APEX 20K FAMILY --- MEGALAB STRUCTURE



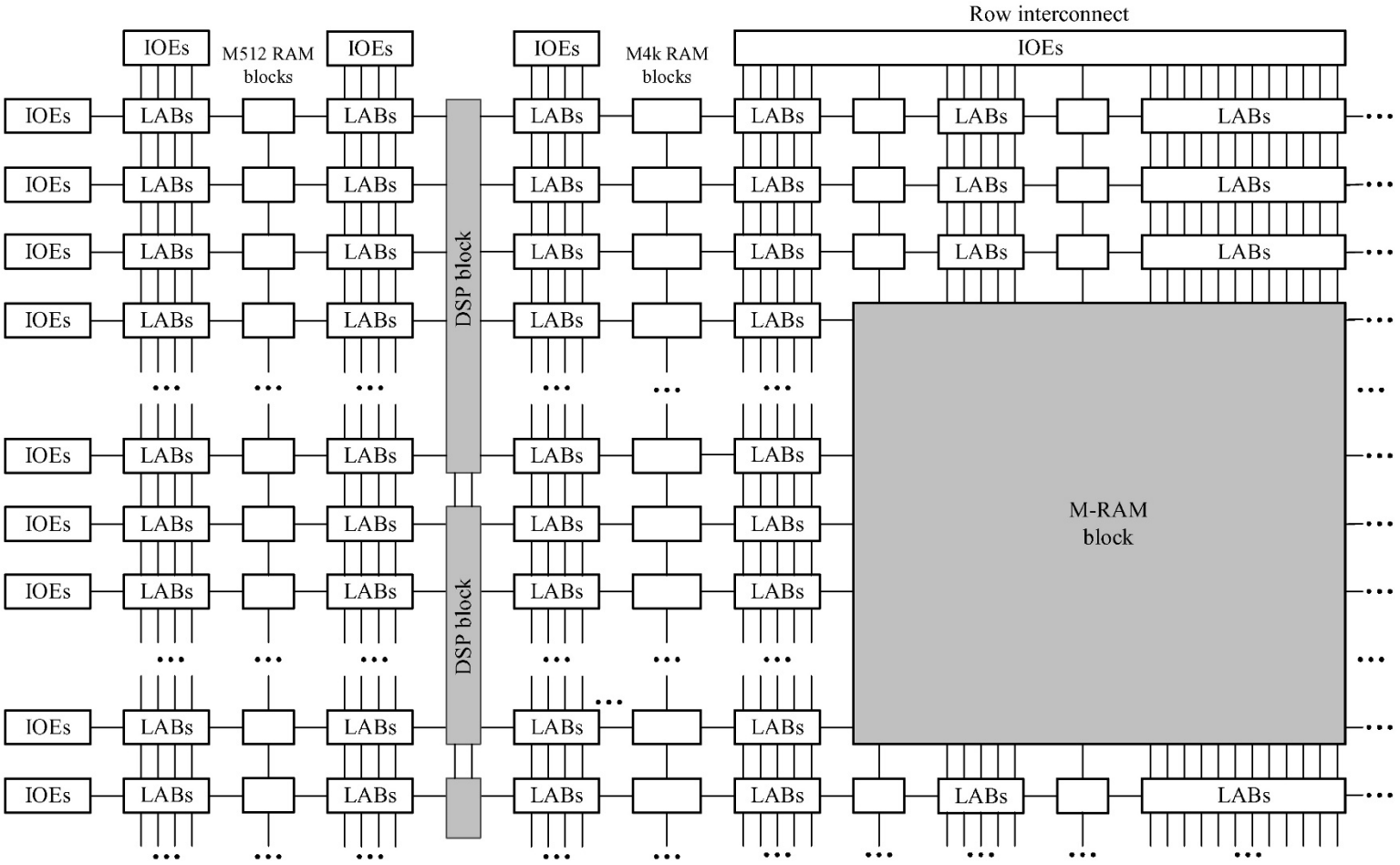
APEX 20K FAMILY --- LOGIC ELEMENT STRUCTURE



APEX 20K FAMILY --- STRUCTURE OF ESB



ALTERA STRATIX FAMILY --- BASIC STRUCTURE



SUMMARY

- Similarity
  - FPGA and CPLD are programmable ASIC devices
- Differences
  - CPLDs
    - Use of E2PROM or Flash. A limited number of writing.
    - Suitable for combinational logic and product terms
    - Uniform and predictable timing delay
    - No extra storage, faster, smaller, and simpler
  - FPGAs are for sequential logic
    - Use of SRAM. No limit on the number of writing. Reload every time.
    - Suitable for sequential logic with many registers
    - Timing delay depending on the routing
    - Better for complex design
    - Extra non-volatile memory needed for configuring