

BEE 271 Digital circuits and systems

Spring 2017

Lecture 14: FSMs, memories, FPGAs

Nicole Hamilton

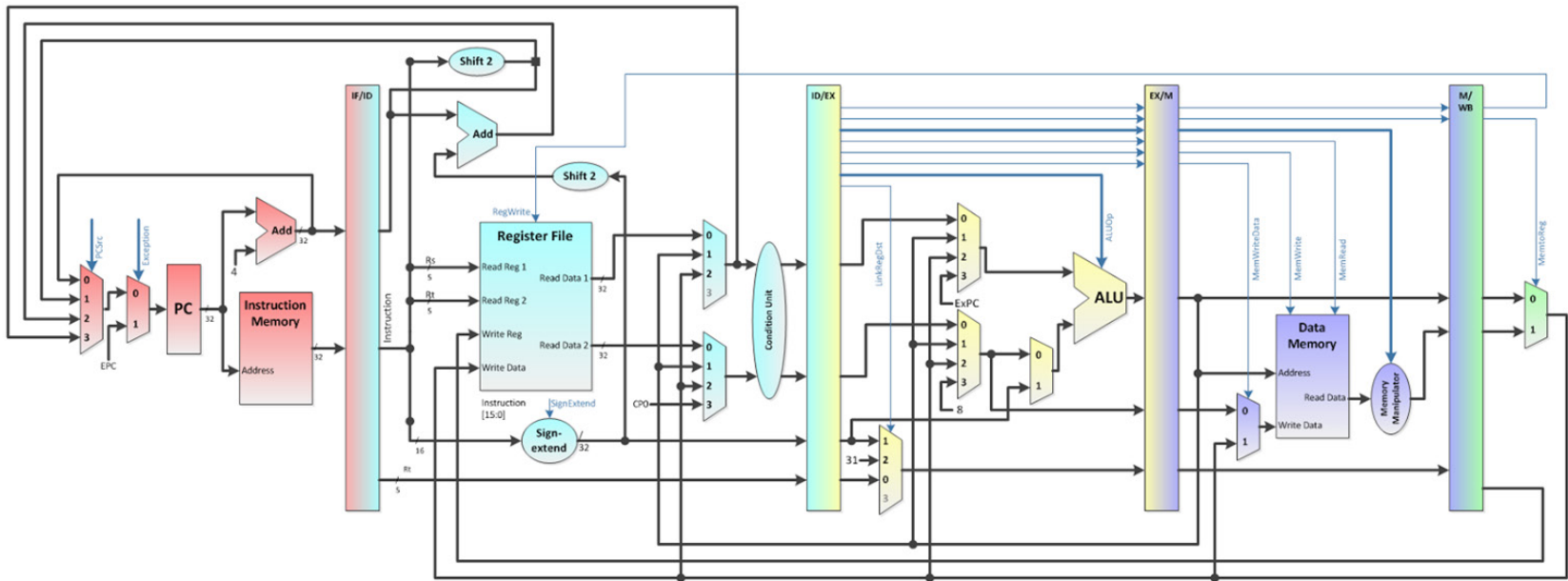
<https://faculty.washington.edu/kd1uj>

Memory

Appears in several forms in a hierarchy of performance and capacities in a modern machine:

1. **Individual flip-flops and counters** used within a module.
2. **Pipeline registers** that hold the state of an instruction as it's passed from one stage of the processor's pipeline to the next.
3. **Register files**, groups of perhaps 16 or 32 registers, each the word length of the processor (e.g., 32 bits) and available to the programmer via the instruction set.
4. **Cache**, typically fast static RAM (i.e., latches) used to buffer data in and out of main memory.
5. **Main memory**, typically SDRAM, which stores data as small electrical charges on tiny capacitors and which must be refreshed constantly.
6. **Non-volatile storage**, e.g., hard disk, flash drive.

The classic RISC pipeline



Instruction
fetch

Instruction
decode

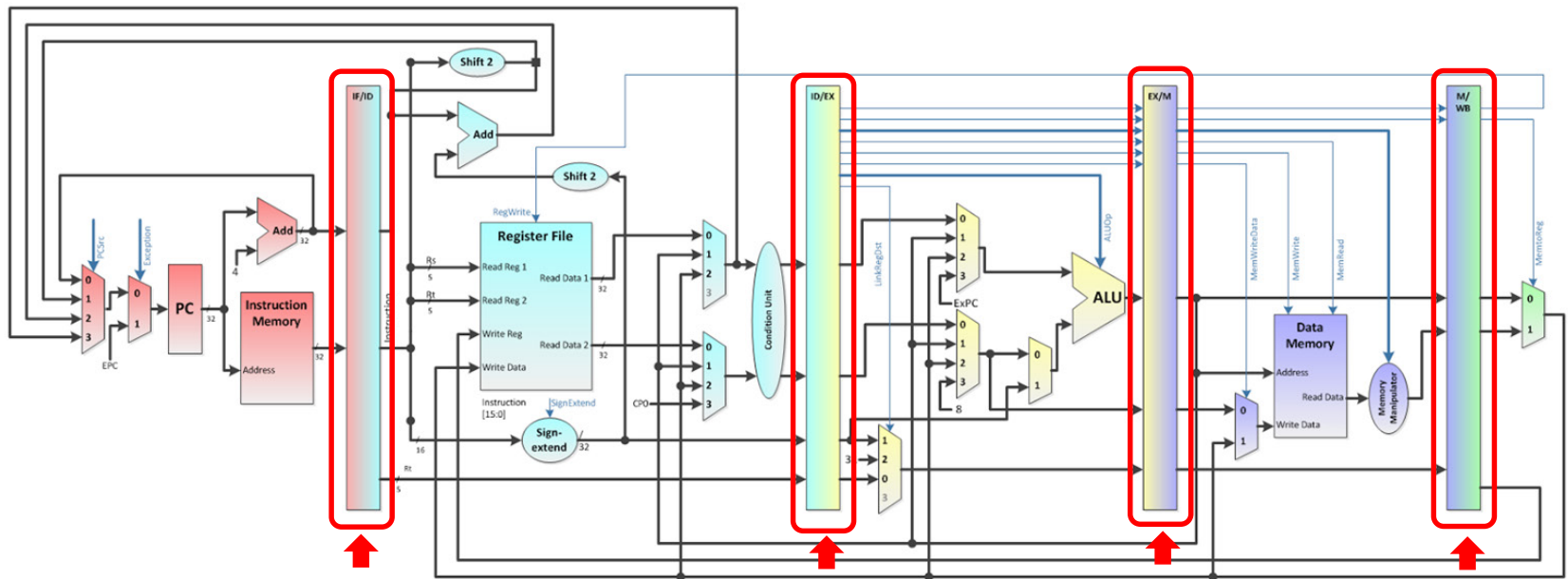
Execute

Memory

Writeback

The classic RISC pipeline

The tall registers between each stage capture the state of an instruction as it moves through the pipeline.



Instruction
fetch

Instruction
decode

Execute

Memory

Writeback

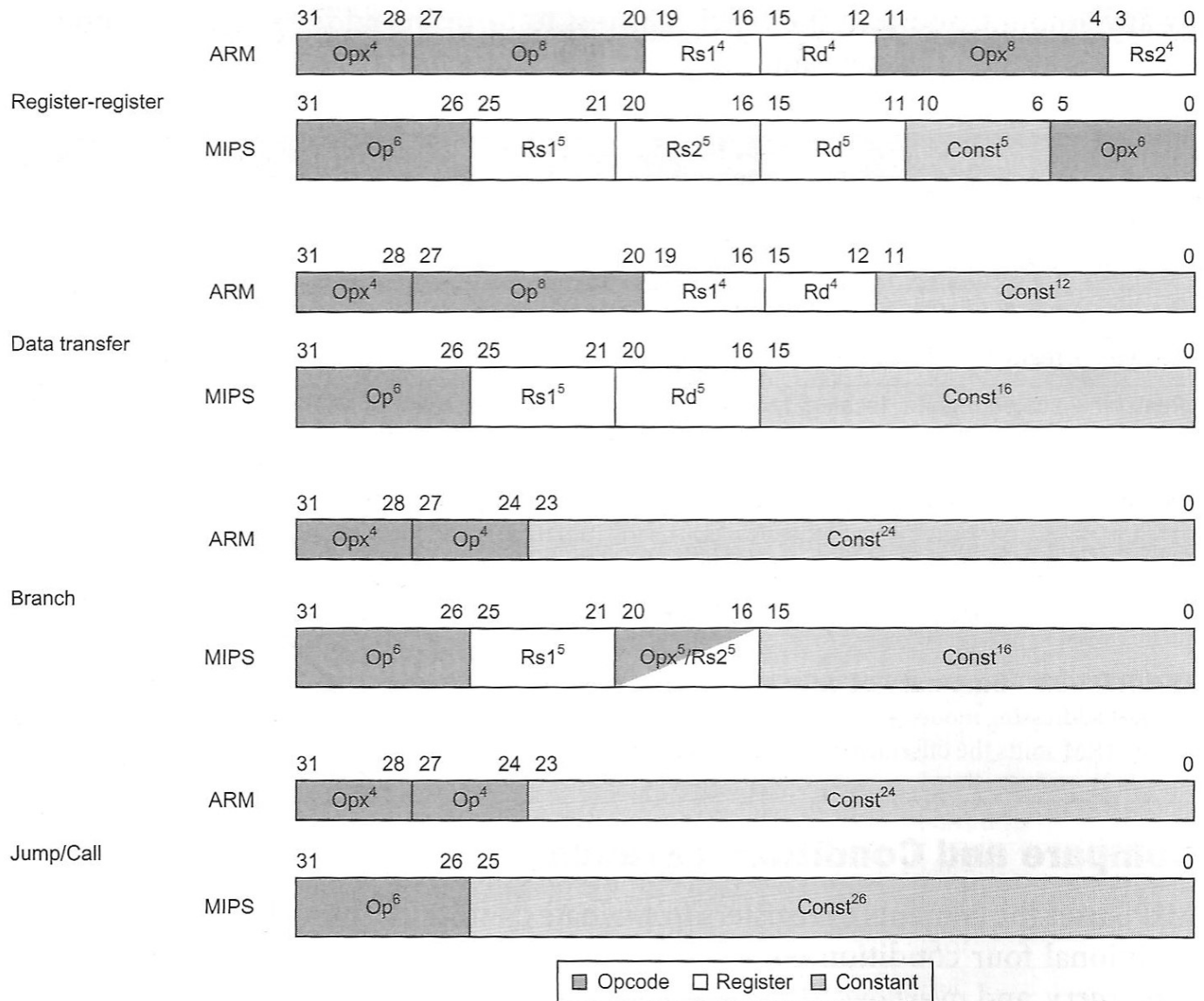
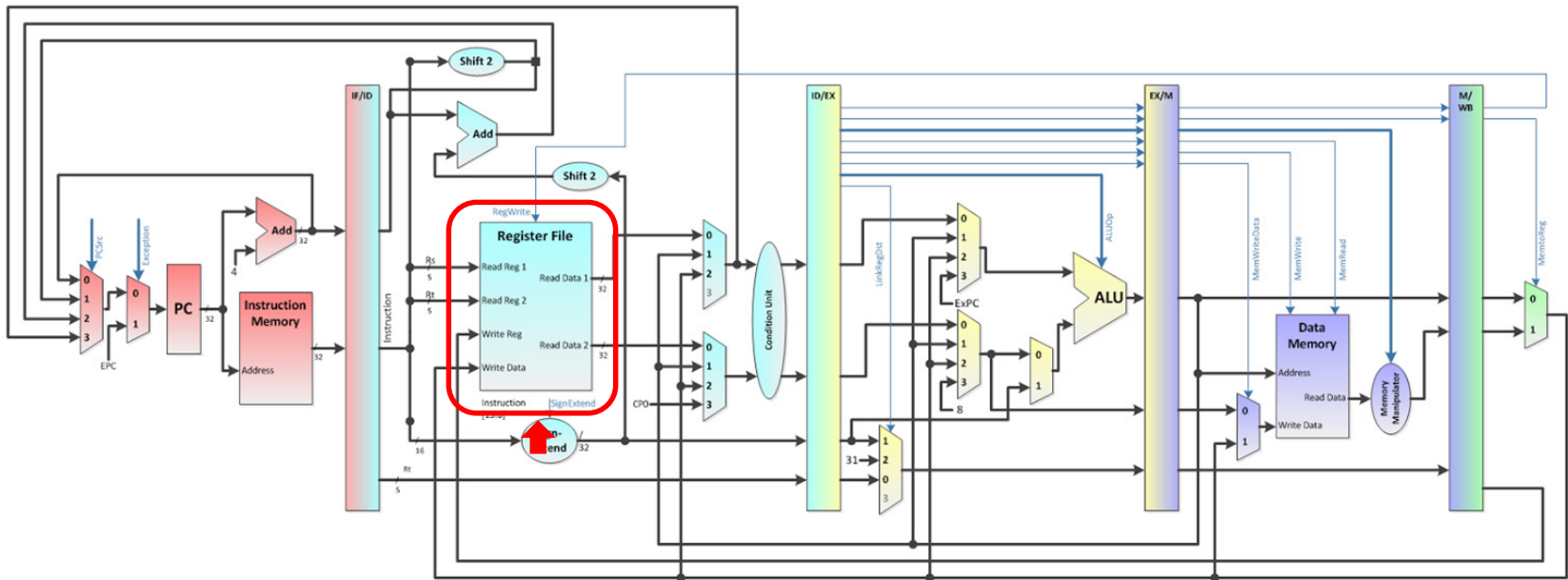


FIGURE 2.34 Instruction formats, ARM and MIPS. The differences result from whether the architecture has 16 or 32 registers.

The register file

The register file is accessible to the instruction set.



Instruction
fetch

Instruction
decode

Execute

Memory

Writeback

Register file in the MIPS

31 registers numbered 1 to 31

Register 0 is always 0

Three ports:

1. Read ports for A and B that continuously report the contents of registers A and B
2. Write port for C with an enable.

Create a Verilog module that does this.

```

module RegisterFile( input clock, reset,
    input [4:0] regA, regB, output [31:0] Aout, Bout,
    input [4:0] regC, input writeC, input [31:0] Cin );

    // Three-port register file. Registers A and B are read
    // continuously, register C can be written.
    // Register 0 is always 0.

    reg [ 31:0 ] rf[ 1:31 ];
    assign Aout = regA != 0 ? rf[ regA ] : 0;
    assign Bout = regB != 0 ? rf[ regB ] : 0;

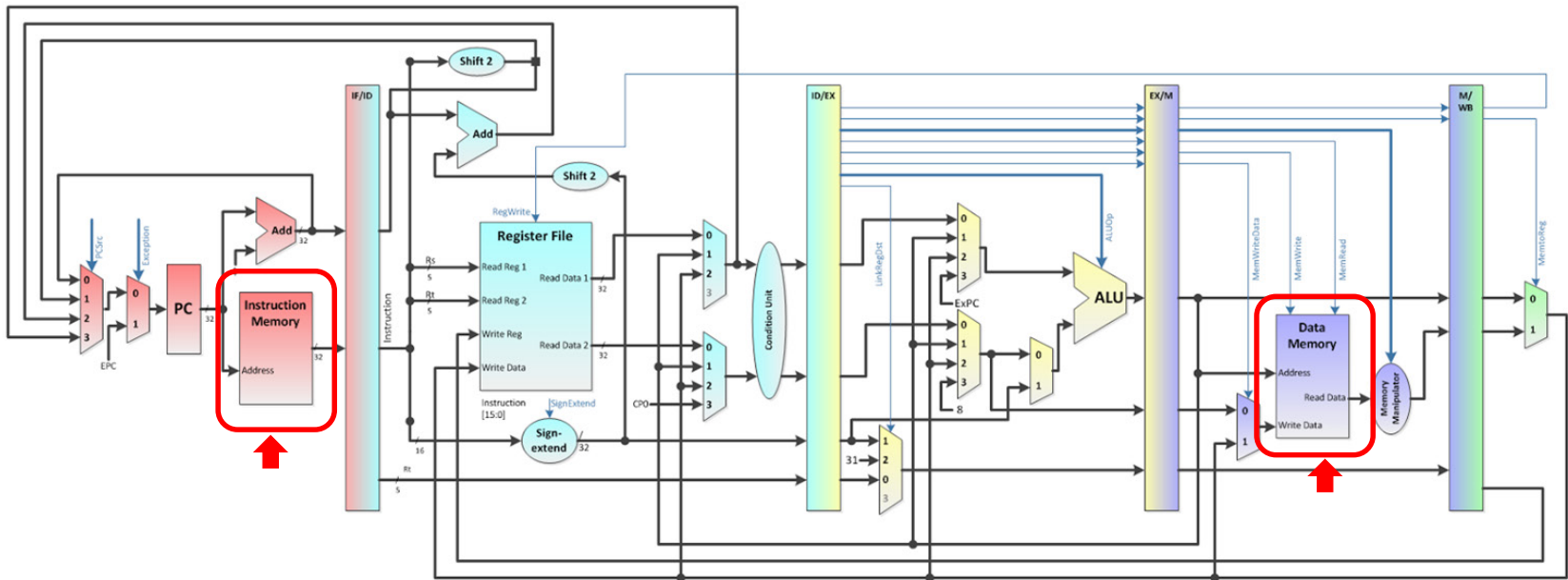
    always @( posedge clock )
        if ( reset )
            begin
                integer i;
                for ( i = 1; i < 32; i = i + 1 )
                    rf[ i ] <= 0;
            end
        else
            if ( writeC && regC != 0 )
                rf[ regC ] <= Cin;

endmodule

```


Instruction and data memories

The MIPS is a Harvard architecture with separate instruction and data memories.



Instruction
fetch

Instruction
decode

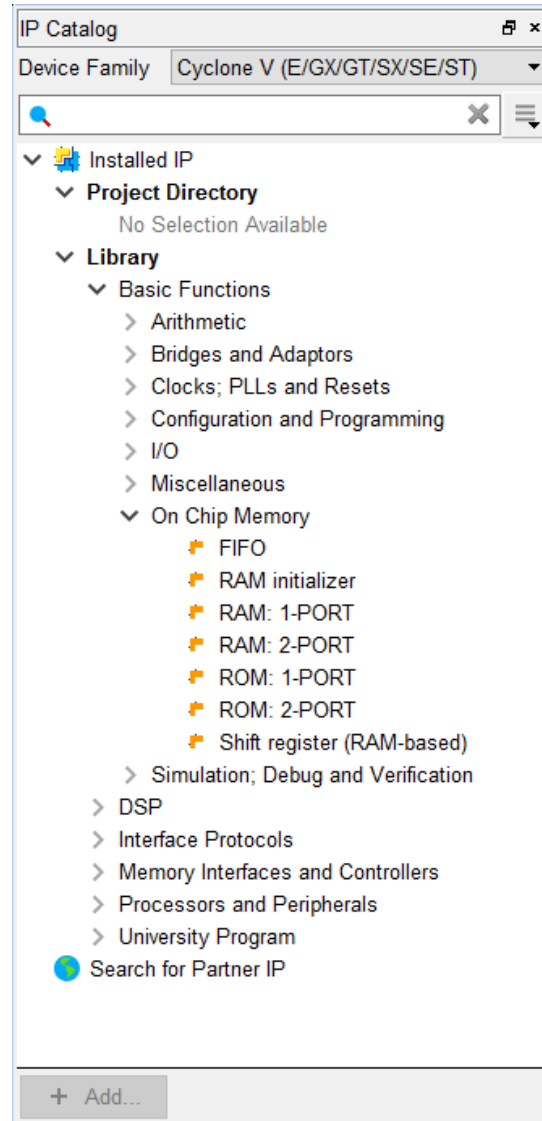
Execute

Memory

Writeback

Static RAM on an FPGA

Created from the vendor's "IP library" using their tool to describe what you want.



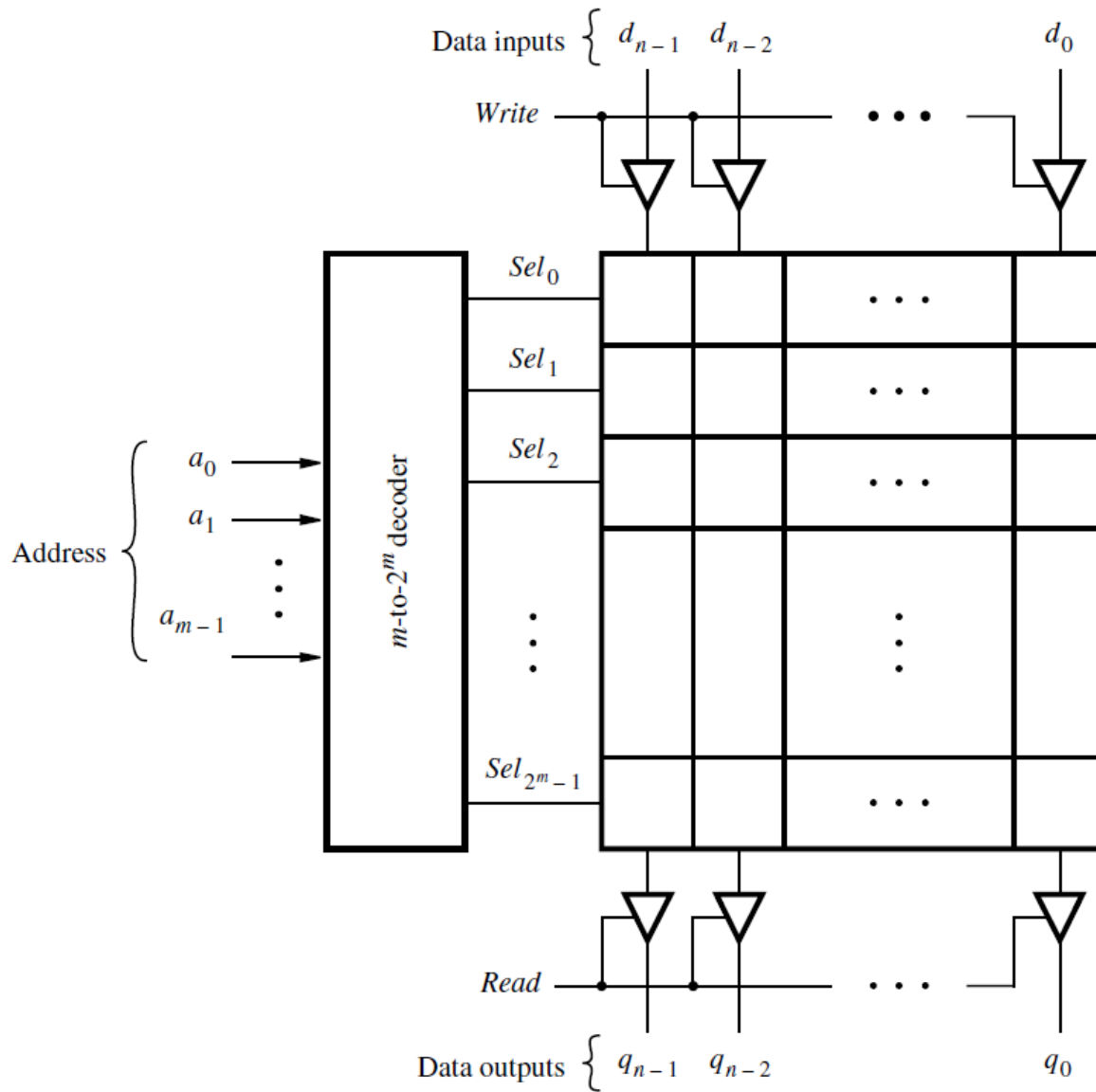


Figure B.66. A $2^m \times n$ SRAM block.

An example 2-port RAM

MegaWizard Plug-In Manager [page 1 of 10]

RAM: 2-PORT

About Documentation

1 Parameter Settings | 2 EDA | 3 Summary

General > Widths/Blk Type > Clks/Rd, Byte En > Regs/Clkens/Aclrs > Output1 > Mem Init >

Currently selected device family: Cyclone V

Match project/default

How will you be using the dual port RAM?

With one read port and one write port

With two read/write ports

How do you want to specify the memory size?

As a number of words

As a number of bits

Block Type: AUTO

Resource Usage
43 lut + 64 M10K + 6 reg

Cancel < Back Next > Finish

Specify the size and width.

The screenshot shows the MegaWizard Plug-In Manager interface for configuring a RAM: 2-PORT block. The window title is "MegaWizard Plug-In Manager [page 2 of 10]". The main title is "RAM: 2-PORT". The interface is divided into several sections:

- Navigation:** A series of tabs: "Parameter Settings", "EDA", "Summary", "General", "Widths/Blk Type" (selected), "Clks/Rd, Byte En", "Regs/Clkens/Aclrs", "Output1", and "Mem Init".
- Block Diagram:** A diagram titled "twoportram" showing a RAM block with inputs: "data[7..0]", "waddress[15..0]", "wren", "rdaddress[15..0]", "rden", and "clock". The output is "q[7..0]". The RAM size is set to "65536 Word(s)". Below the diagram, it says "Block Type: AUTO".
- Configuration Options:**
 - "How many 8-bit words of memory?" is set to "65536".
 - "Use different data widths on different ports" is unchecked.
 - Read/Write Ports:**
 - "How wide should the 'q_a' output bus be?" is set to "8".
 - "How wide should the 'data_a' input bus be?" is set to "8".
 - "How wide should the 'q' output bus be?" is set to "8".
 - Note: "You could enter arbitrary values for width and depth".
 - What should the memory block type be?**
 - Radio buttons: "Auto" (selected), "MLAB", "M10K", "M144K", "LCs".
 - "Options..." button.
 - "Set the maximum block depth to" is set to "Auto" words.
- Resource Usage:** A box at the bottom left shows "43 lut + 64 M10K + 6 reg".
- Buttons:** "Cancel", "< Back", "Next >", and "Finish".

How it will be clocked.

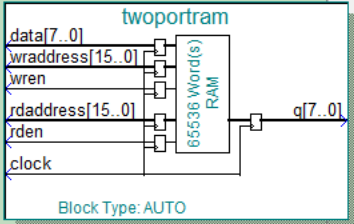
MegaWizard Plug-In Manager [page 3 of 10]

RAM: 2-PORT

About Documentation

1 Parameter Settings 2 EDA 3 Summary

General > Widths/Blk Type > Clks/Rd, Byte En > Regs/Clkens/Aclrs > Output1 > Mem Init >



Block Type: AUTO

Resource Usage
43 lut + 64 M10K + 6 reg

What clocking method do you want to use?

- Single clock
- Dual clock: use separate 'read' and 'write' clocks
- Dual clock: use separate 'input' and 'output' clocks
- No clock (fully asynchronous)
- Customize clocks for A and B ports

Create a 'rden' read enable signal

Byte Enable Ports

- Create byte enable for port A
- Create byte enable for port B

What is the width of a byte for byte enables? 8 bits

- Enable error checking and correcting (ECC) to check and correct single bit errors and detect double errors
- Enable ECC pipeline registers before the output decoder to achieve the same performance as non-ECC mode at the expense of one cycle of latency

Cancel < Back Next > Finish

Which ports are latched.

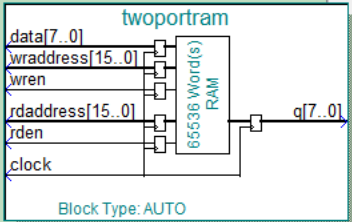
MegaWizard Plug-In Manager [page 5 of 10]

RAM: 2-PORT

About Documentation

1 Parameter Settings 2 EDA 3 Summary

General > Widths/Blk Type > Clks/Rd, Byte En > **Regs/Clkens/Aclrs** > Output1 > Mem Init >



Block Type: AUTO

Resource Usage
43 lut + 64 M10K + 6 reg

Which ports should be registered?

- Write input ports
'data', 'waddress', and 'wren'
- Read input ports
'rdaddress' and 'rden'
- Read output port(s)
'q'
- Create one clock enable signal for each clock signal
- Use different clock enables for registers
- Create an 'aclr' asynchronous clear for the registered ports

More Options... More Options... More Options...

Cancel < Back Next > Finish

What happens if you try to read a location that's being written.

MegaWizard Plug-In Manager [page 6 of 10]

RAM: 2-PORT

Help
About Documentation

1 Parameter Settings 2 EDA 3 Summary

General > Widths/Blk Type > Clks/Rd, Byte En > Regs/Clkens/Aclrs > **Output1** > Mem Init >

twoportram

Block Type: AUTO

Mixed Port Read-During-Write for Single Input Clock RAM

How should the q output behave when reading a memory location that is being written from the other port?

- New Data
- Old memory contents appear
- I do not care (The outputs will be undefined)

Note: M-RAM cannot be used with this behavior

Do not analyze the timing between write and read operations

Metastability issues are prevented by never writing and at the same address at the same time

Resource Usage
43 lut + 64 M10K + 6 reg

Cancel < Back Next > Finish

Initial values.

MegaWizard Plug-In Manager [page 8 of 10]

RAM: 2-PORT

About Documentation

1 Parameter Settings | 2 EDA | 3 Summary

General > Widths/Blk Type > Clks/Rd, Byte En > Regs/Clkens/Aclrs > Output1 > **Mem Init** >

Block Type: AUTO

Resource Usage
43 lut + 64 M10K + 6 reg

Do you want to specify the initial content of the memory?

- No, leave it blank
- Initialize memory content data to XX..X on power-up in simulation
- Yes, use this file for the memory content data
(You can use a Hexadecimal (Intel-format) File [.hex] or a Memory Initialization File [.mif])

Browse...

File name: _____

The initial content file should conform to which port's dimensions? PORT_B

Cancel < Back Next > Finish

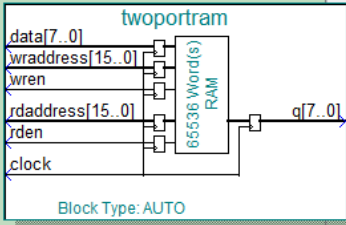
Other vendor IP libraries needed.

MegaWizard Plug-In Manager [page 9 of 10]

RAM: 2-PORT

About Documentation

1 Parameter Settings 2 EDA 3 Summary



Block Type: AUTO

Simulation Libraries

To properly simulate the generated design files, the following simulation model file(s) are needed

File	Description
altera_mf	Altera megafunction simulation library

Timing and resource estimation

Generates a netlist for timing and resource estimation for this megafunction. If you are synthesizing your design with a third-party EDA synthesis tool, using a timing and resource estimation netlist can allow for better design optimization.

Not all third-party synthesis tools support this feature - check with the tool vendor for complete support information.

Note: Netlist generation can be a time-intensive process. The size of the design and the speed of your system affect the time it takes for netlist generation to complete.

Generate netlist

Resource Usage

43 lut + 64 M10K + 6 reg

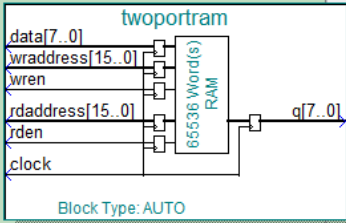
Cancel < Back Next > Finish

Files to be generated.

MegaWizard Plug-In Manager [page 10 of 10]

RAM: 2-PORT

1 Parameter Settings 2 EDA 3 Summary



Turn on the files you wish to generate. A gray checkmark indicates a file that is automatically generated, and a green checkmark indicates an optional file. Click Finish to generate the selected files. The state of each checkbox is maintained in subsequent MegaWizard Plug-In Manager sessions.

The MegaWizard Plug-In Manager creates the selected files in the following directory:
C:\Users\Nicole\Google Drive\bee271\FPGA-RAM\

File	Description
<input checked="" type="checkbox"/> twoportram.v	Variation file
<input type="checkbox"/> twoportram.inc	AHDL Include file
<input type="checkbox"/> twoportram.cmp	VHDL component declaration file
<input type="checkbox"/> twoportram.bsf	Quartus Prime symbol file
<input type="checkbox"/> twoportram_ins...	Instantiation template file
<input checked="" type="checkbox"/> twoportram_bb.v	Verilog HDL black-box file

Resource Usage
43 lut + 64 M10K + 6 reg

Cancel < Back Next > Finish

Dynamic RAM

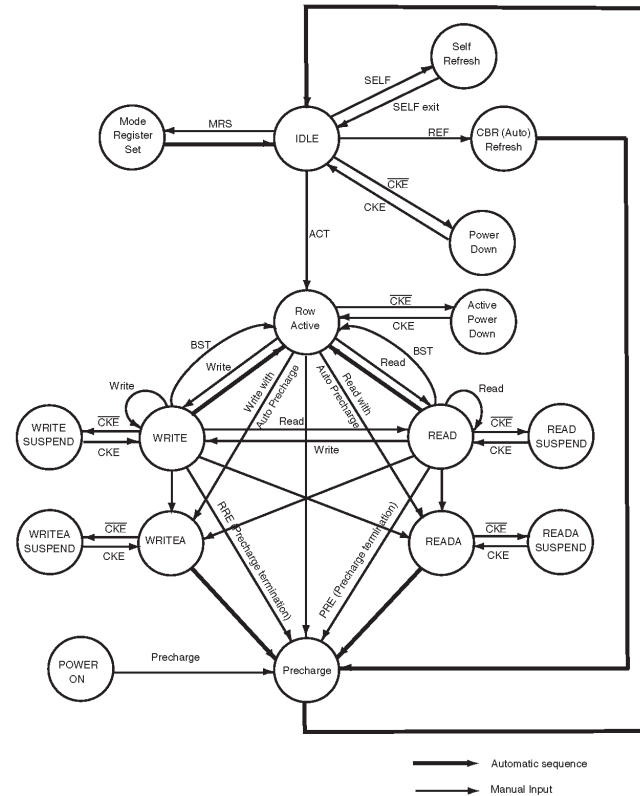
Requires an FSM to refresh periodically.

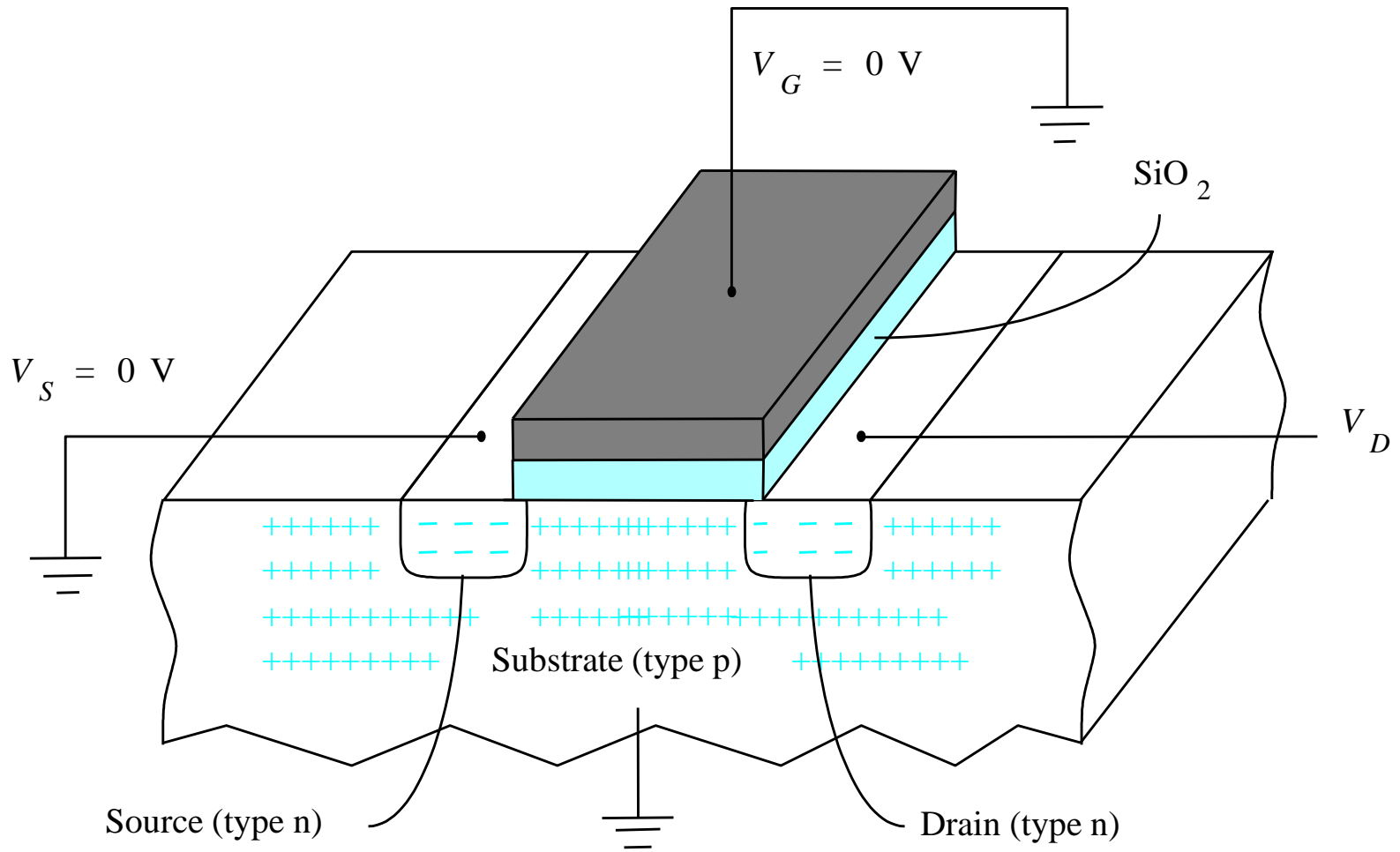
To speed access, usually read or written in burst mode.

IS42S83200B, IS42S16160B

ISSI®

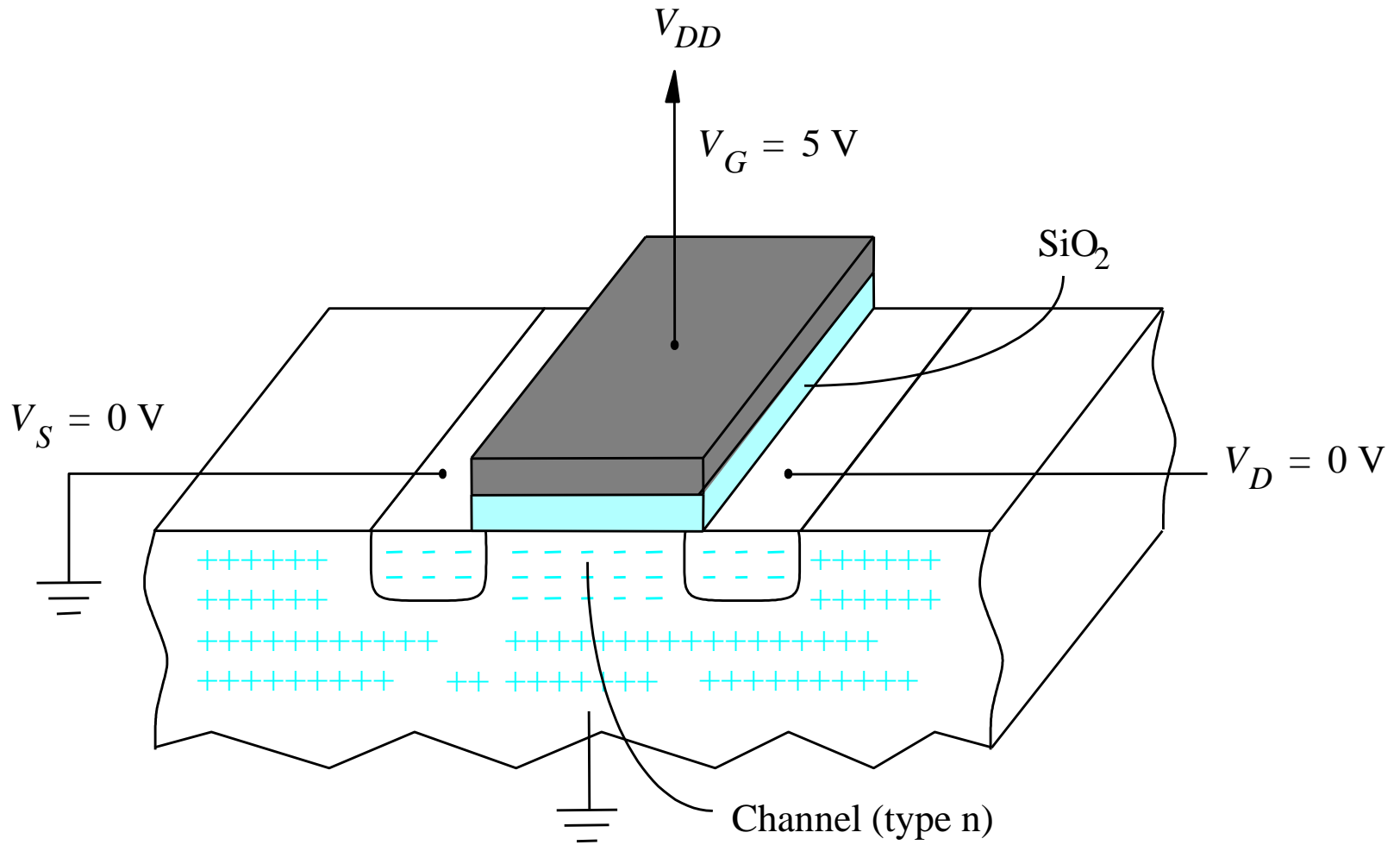
STATE DIAGRAM





(a) When $V_{GS} = 0\text{ V}$, the transistor is off

Figure B.43a. NMOS transistor when turned off.



(b) When $V_{GS} = 5\text{ V}$, the transistor is on

Figure B.43b. NMOS transistor when turned on.

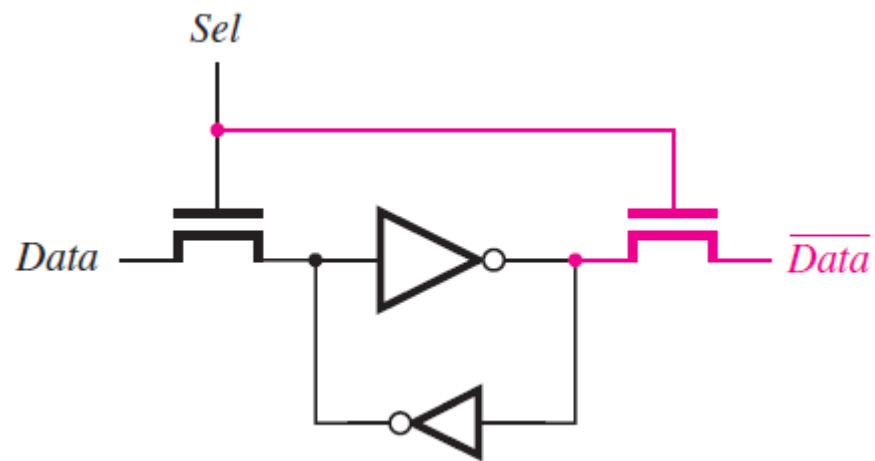


Figure B.64. An SRAM cell.

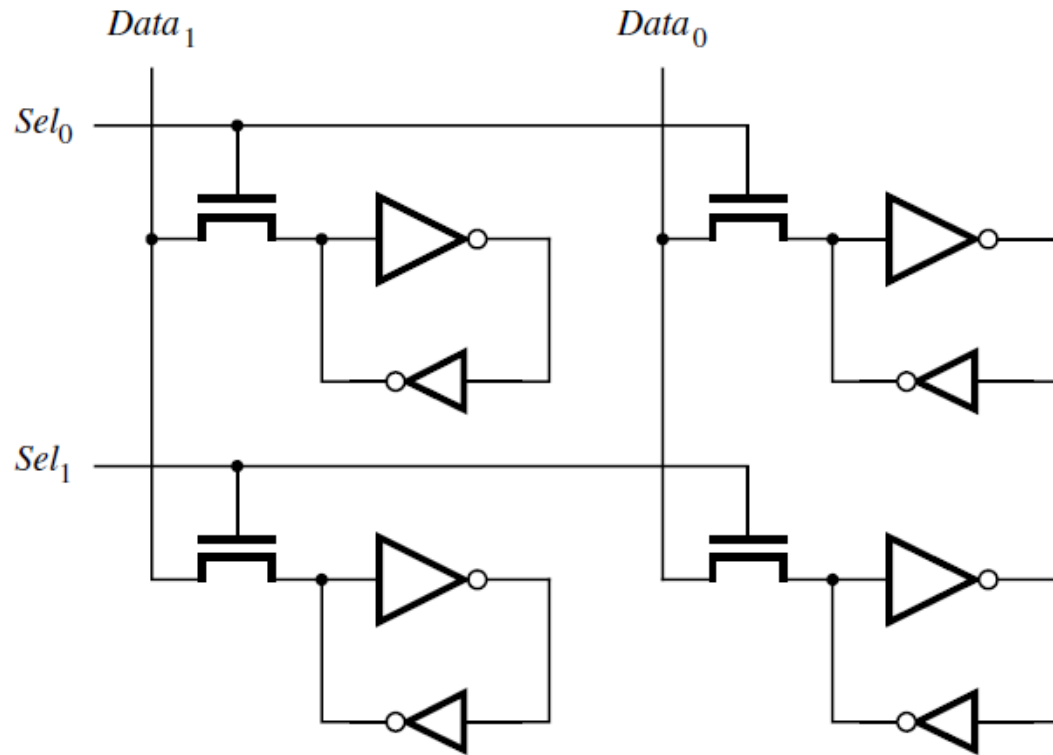


Figure B.65. A 2 x 2 array of SRAM cells.

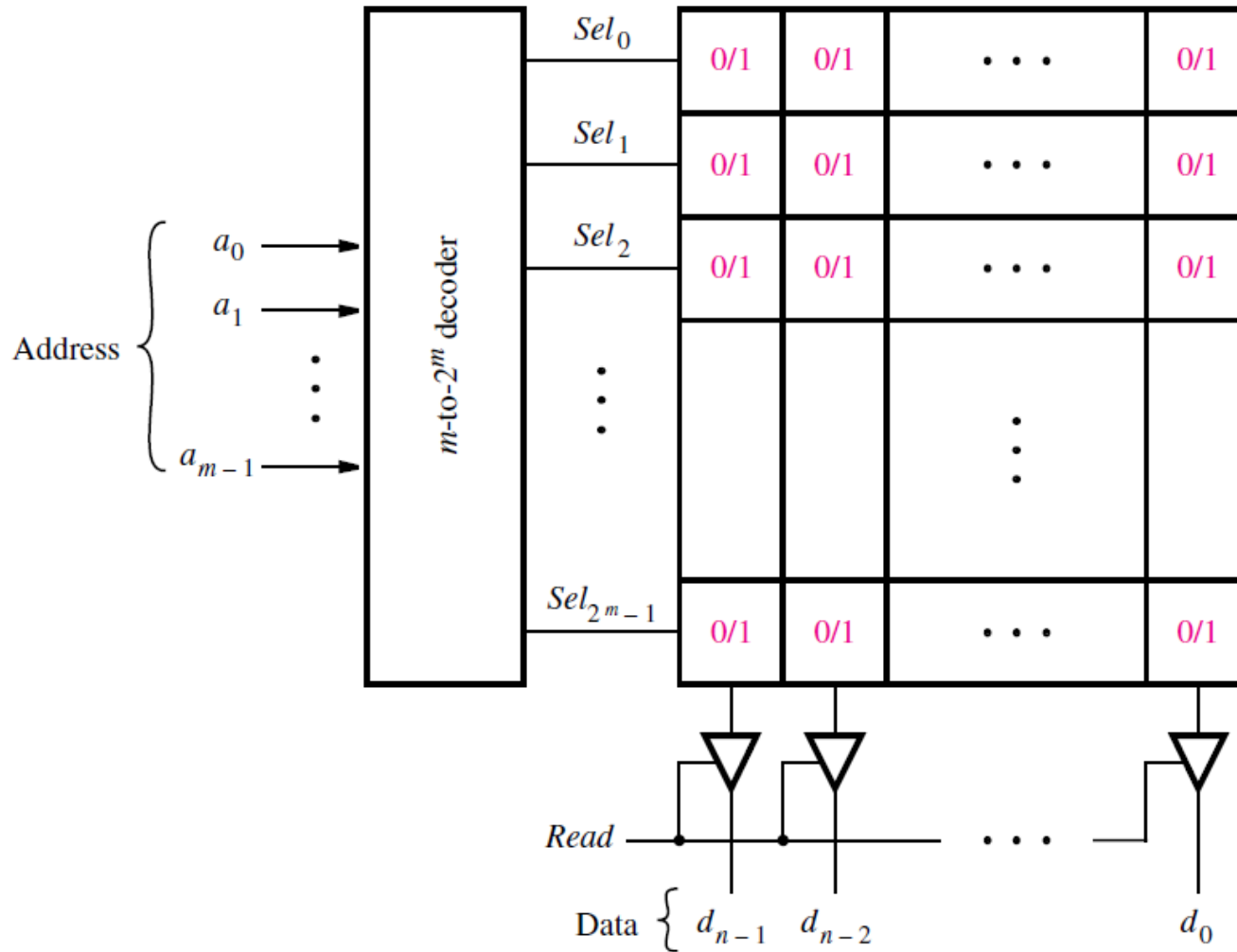


Figure B.72. A $2^m \times n$ read-only memory (ROM) block.

Programmable logic

Historical progression

1. Programmable Logic Arrays (PLAs)
2. Programmable Array Logic (PAL)
3. Complex Programmable Logic Devices (CPLDs)
4. Standard cells
5. Today's FPGAs

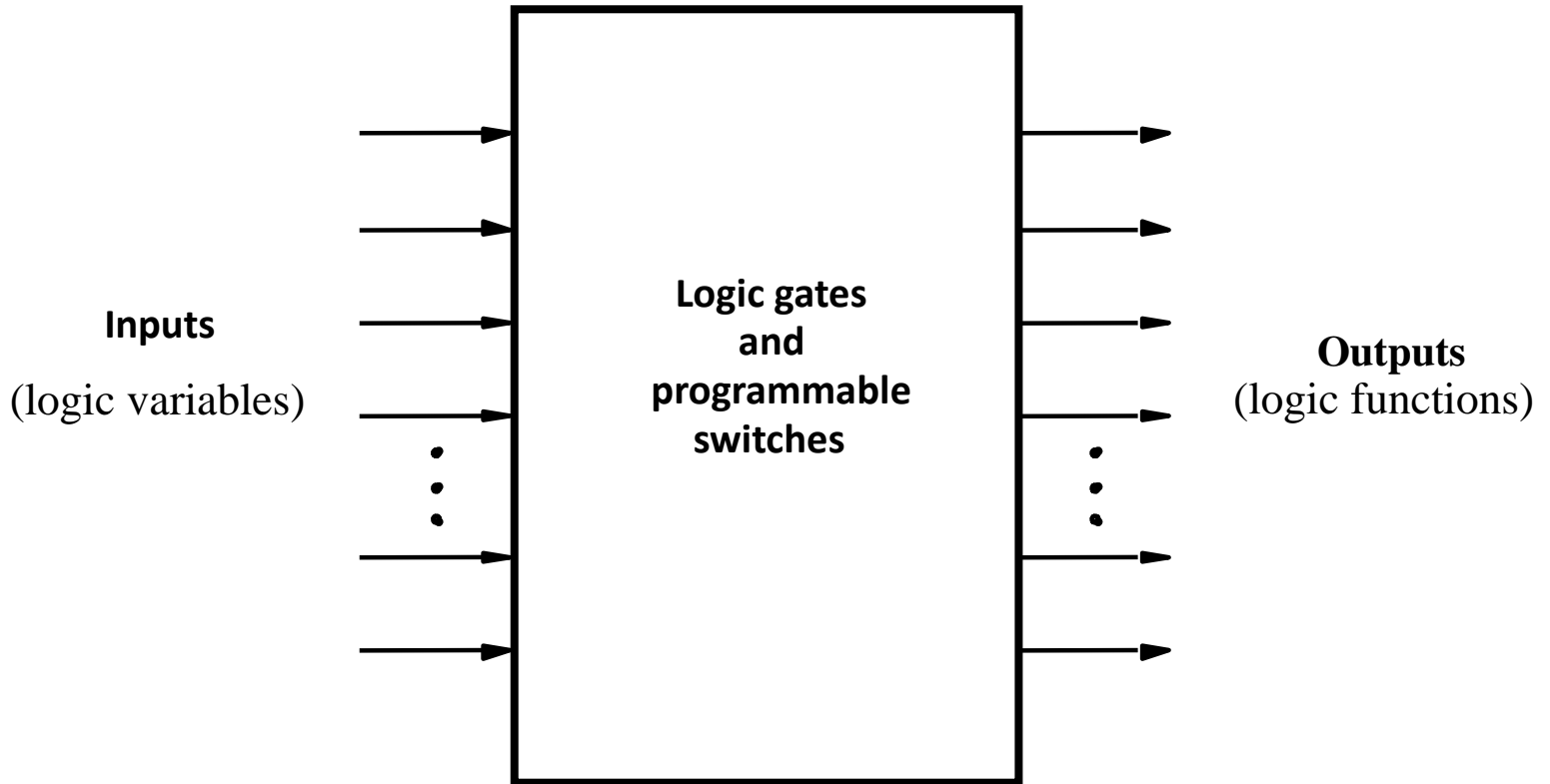


Figure B.24. Programmable logic device as a black box.

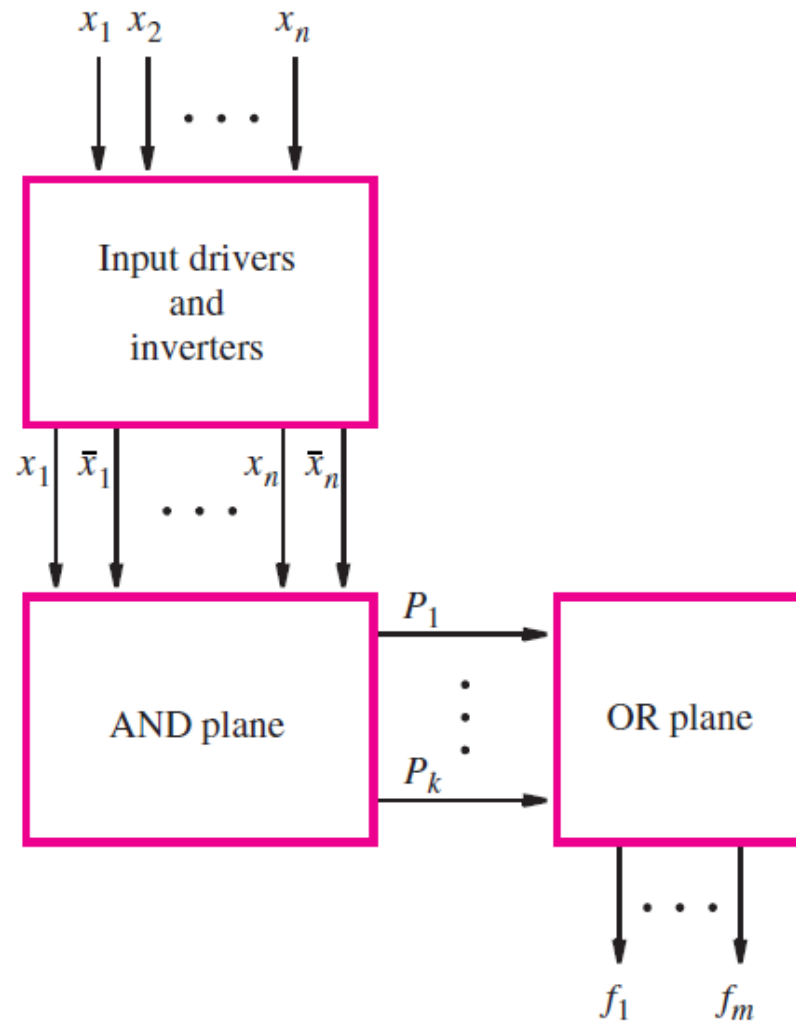


Figure B.25. General structure of a PLA.

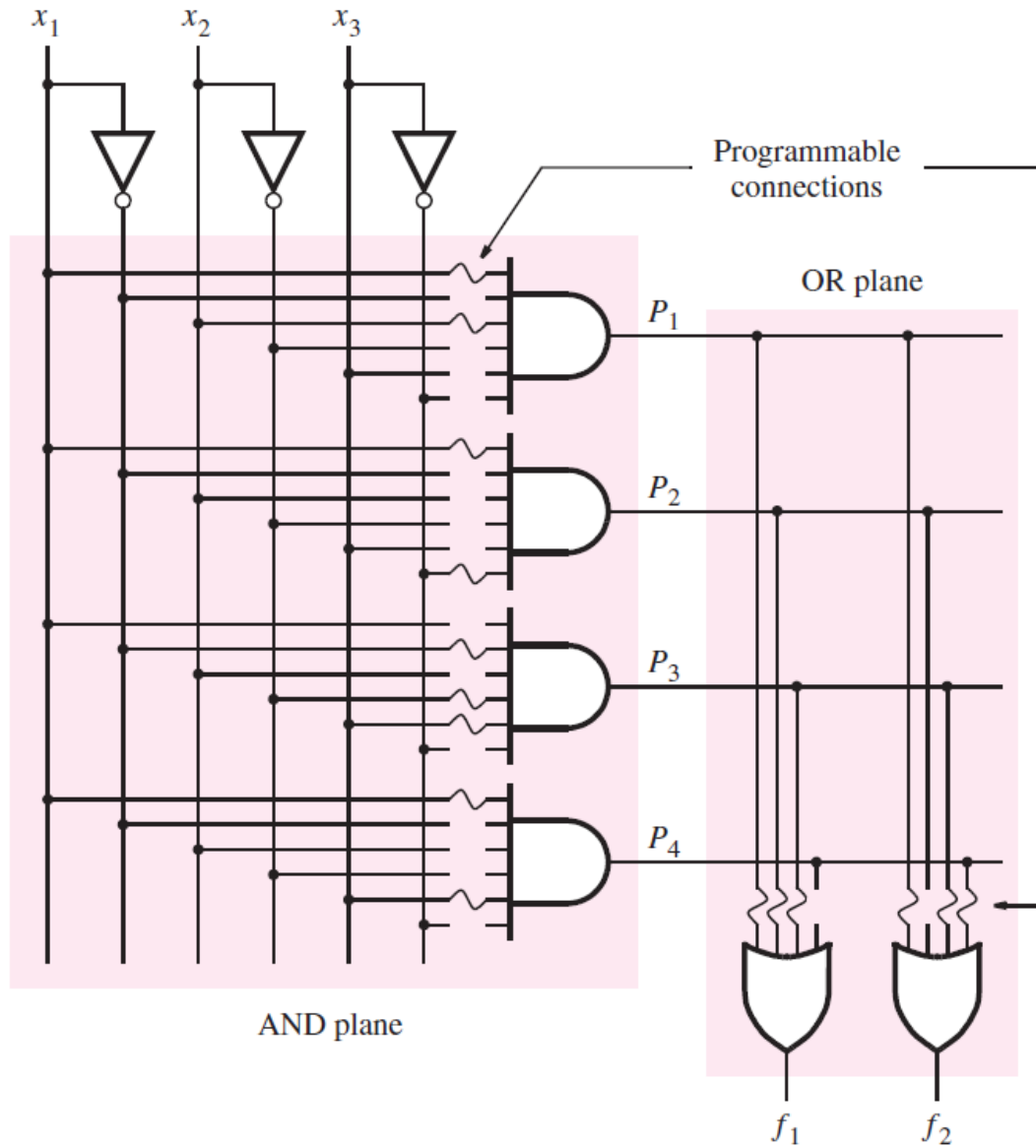


Figure B.26. Gate-level diagram of a PLA.

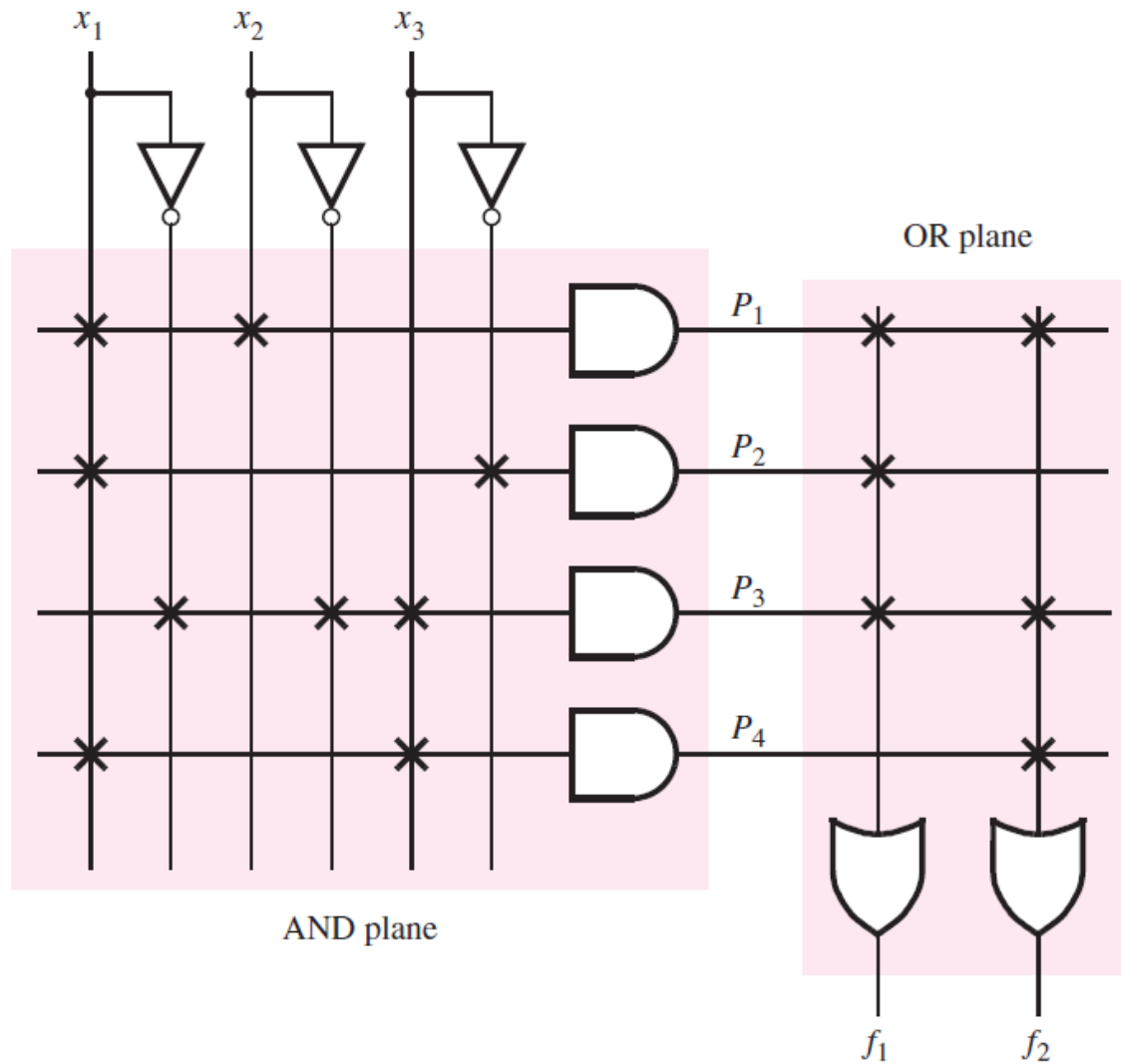


Figure B.27. Customary schematic for the PLA in Figure B.26.

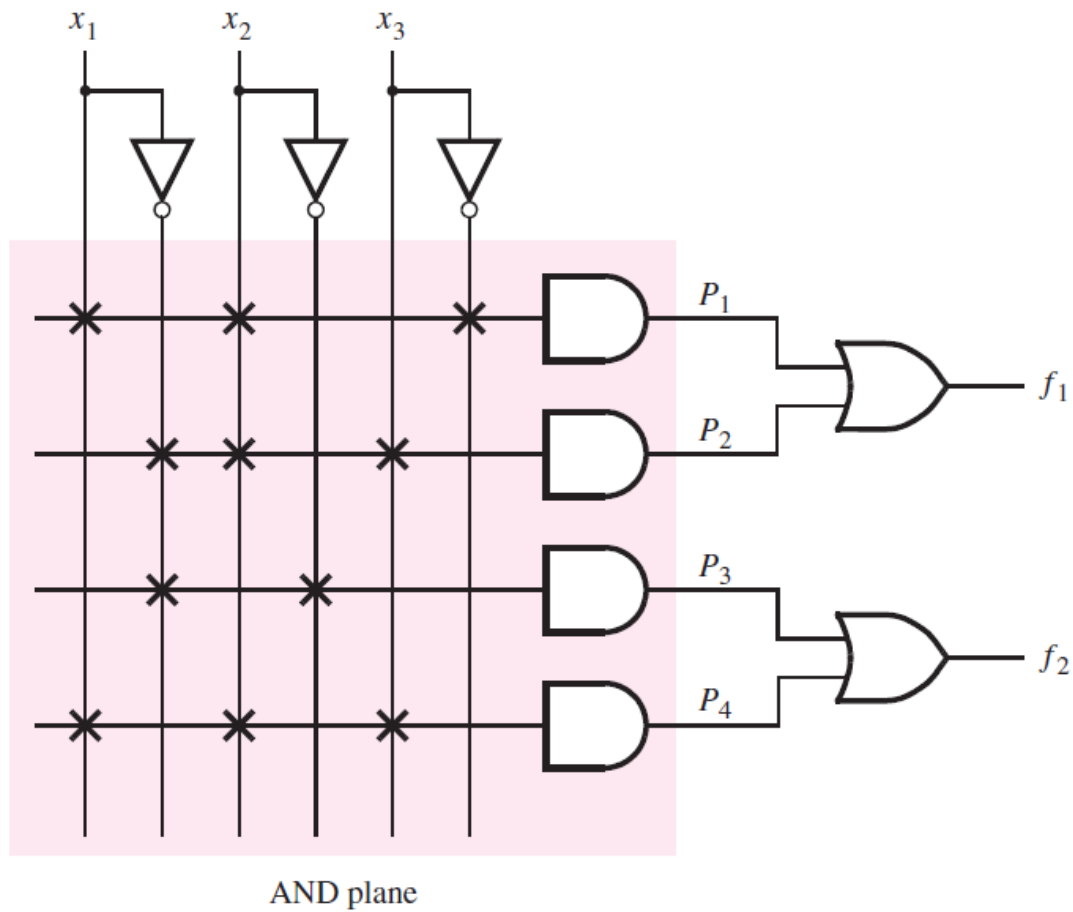


Figure B.28. An example of a PLA.

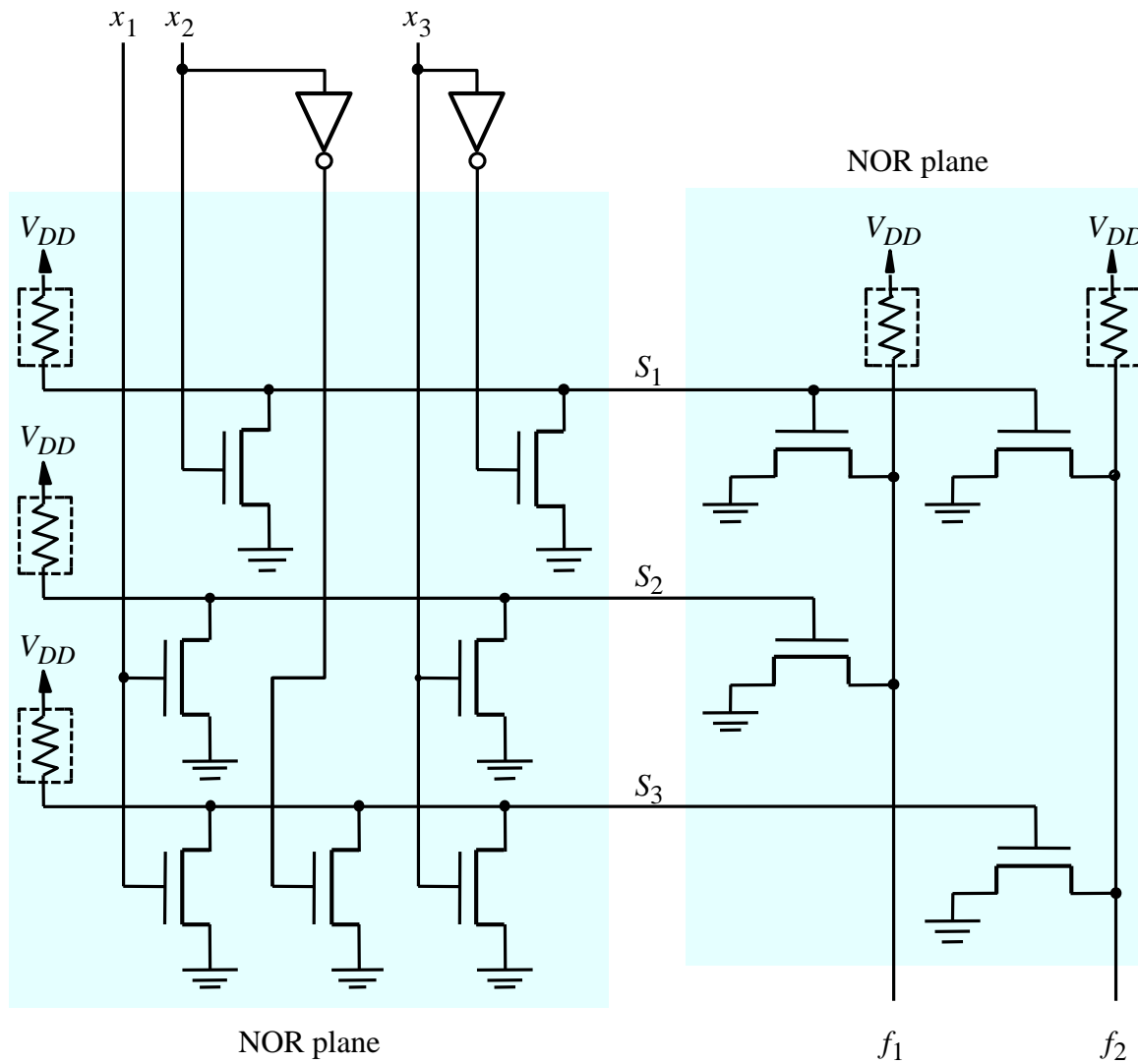
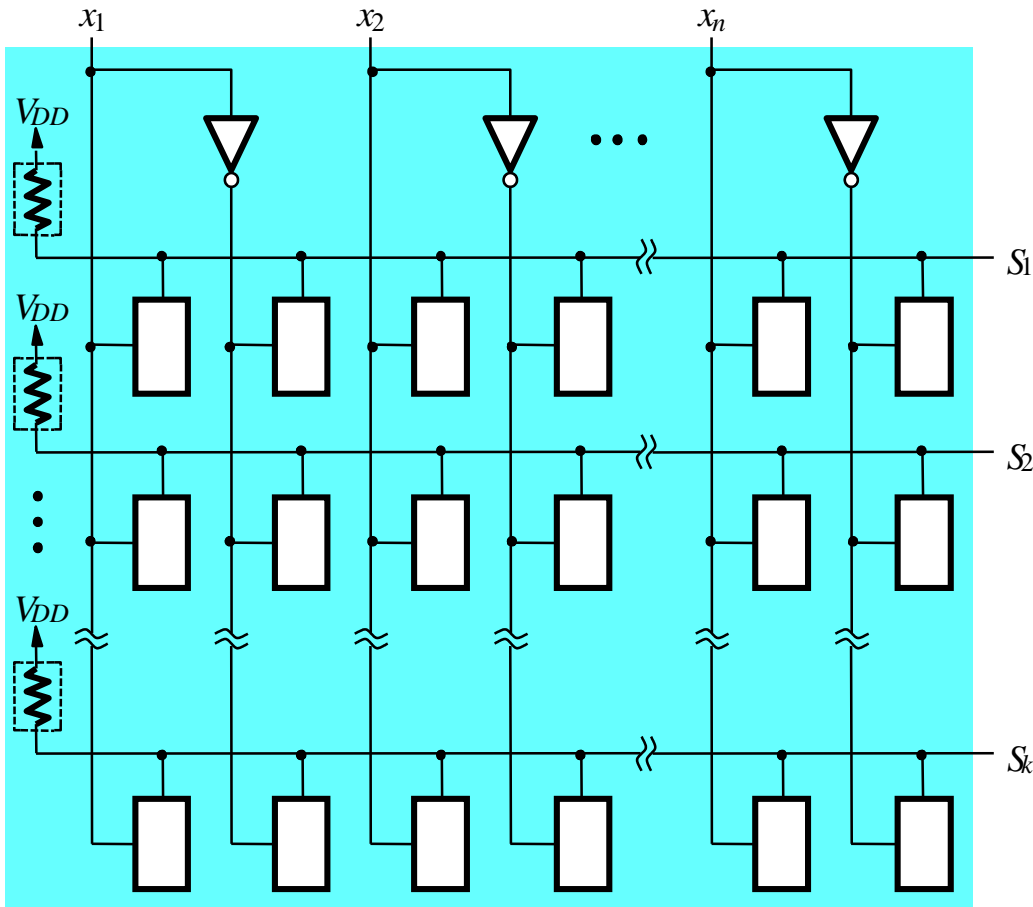
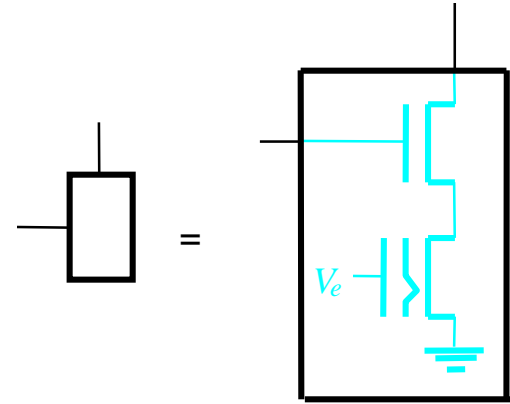


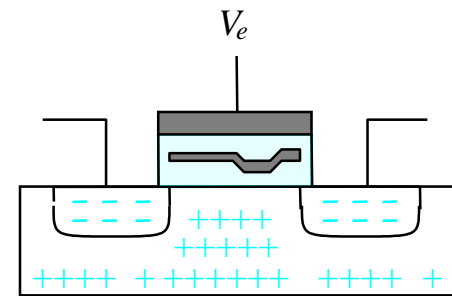
Figure B.67. An example of a NOR-NOR PLA.



(a) Programmable NOR-plane



(b) A programmable switch



(c) EEPROM transistor

Figure B.68. Using EEPROM transistors to create a programmable NOR plane.

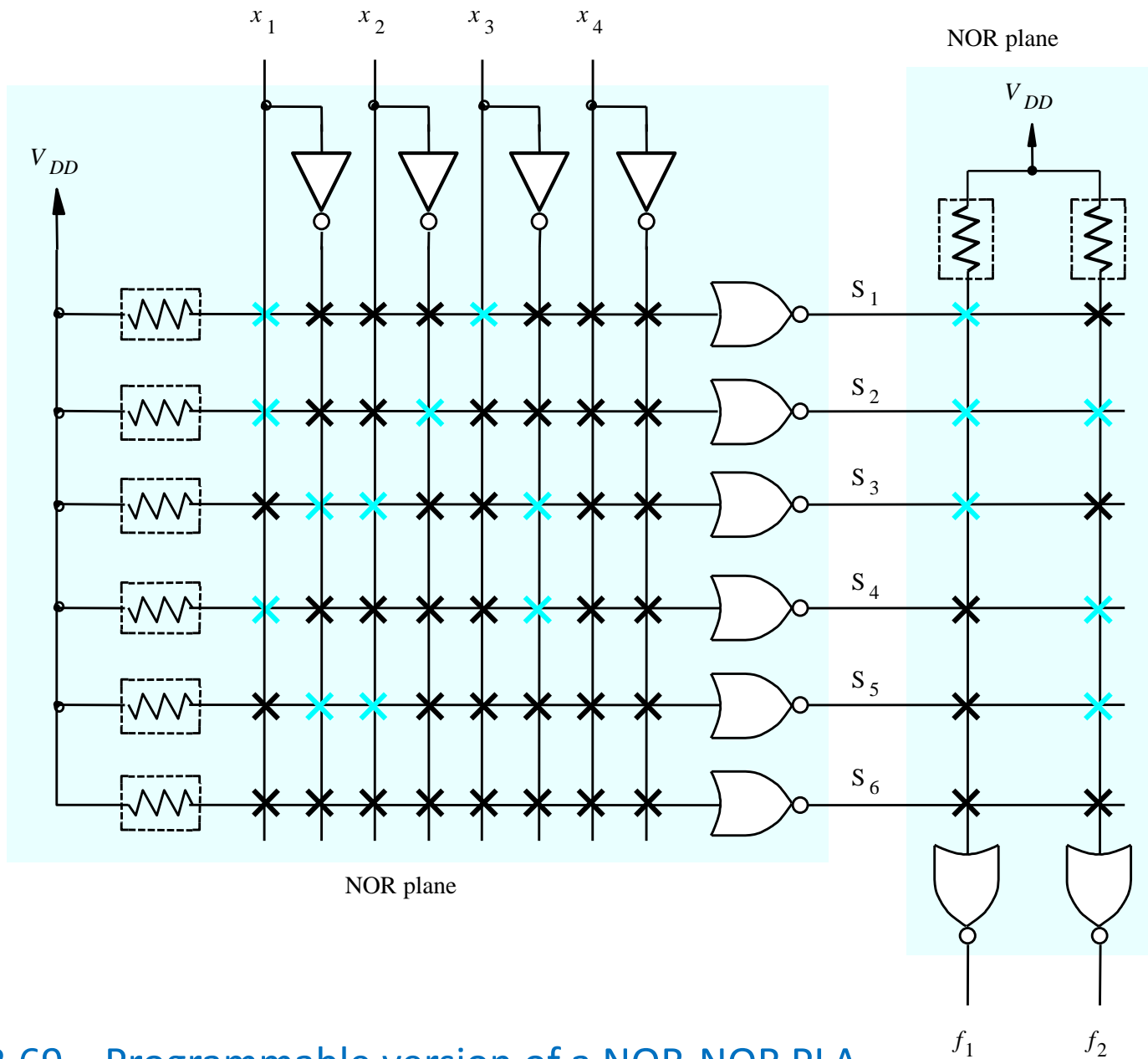


Figure B.69. Programmable version of a NOR-NOR PLA.

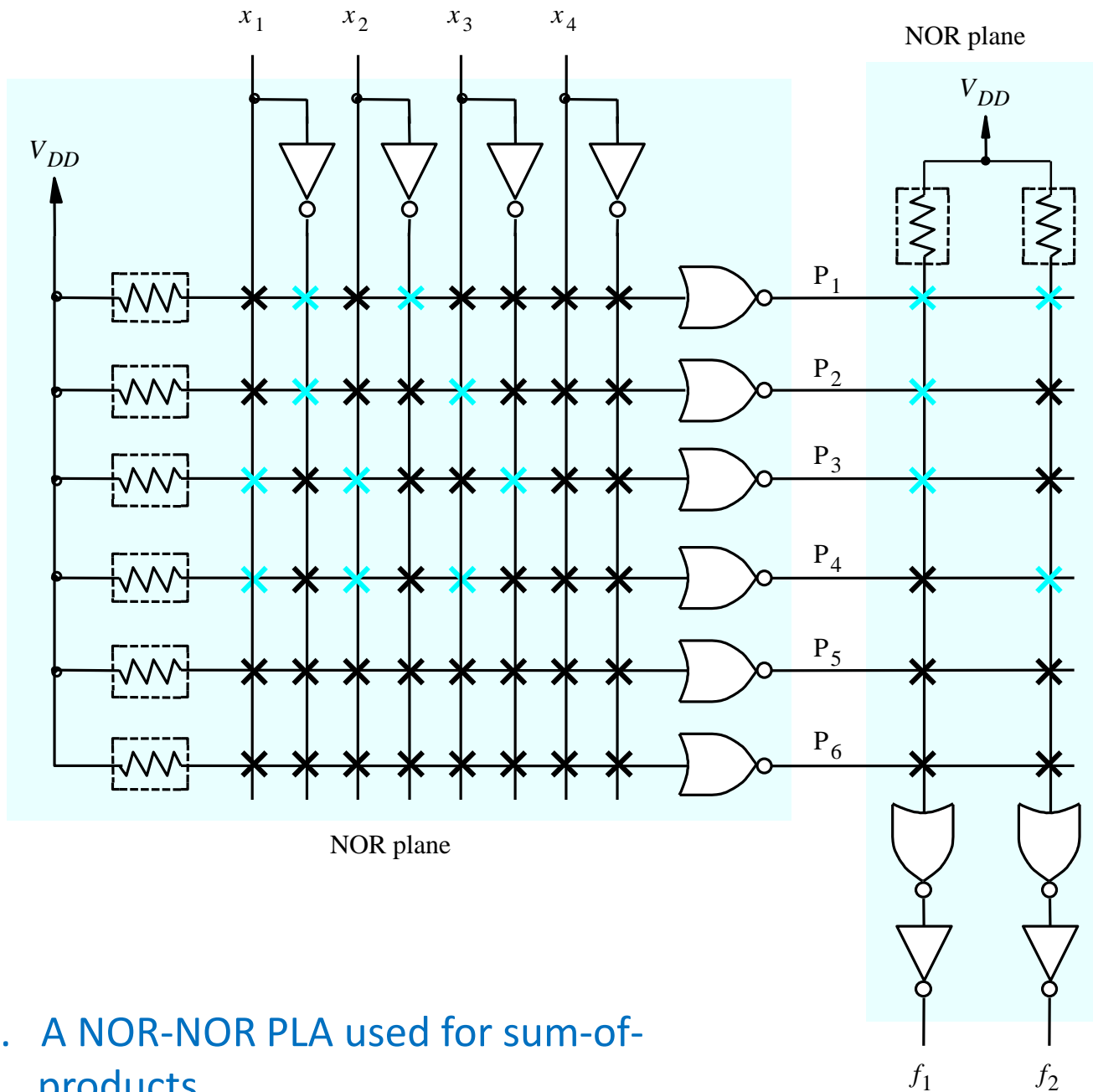


Figure B.70. A NOR-NOR PLA used for sum-of-products.

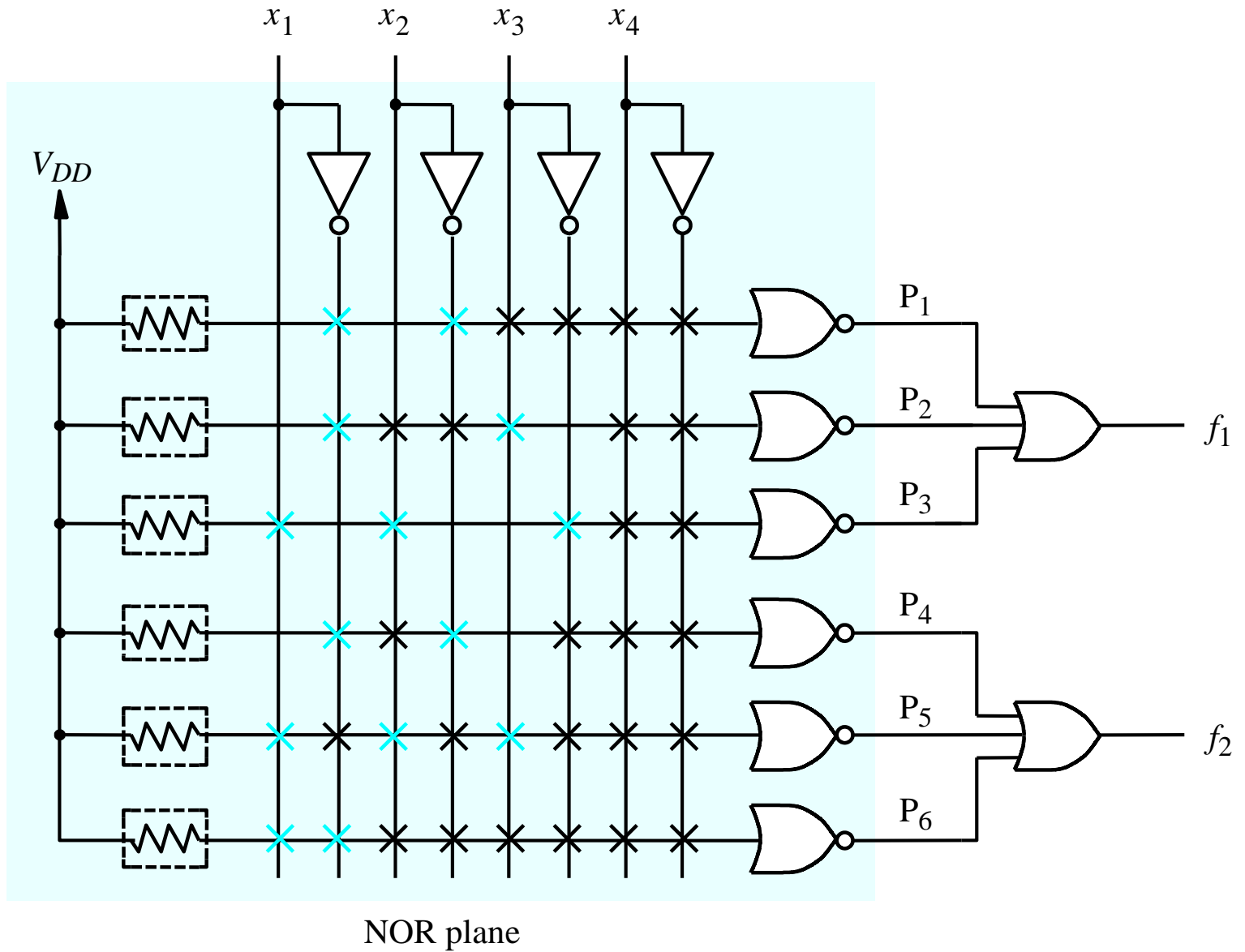


Figure B.71. PAL programmed to implement two functions in Figure B.70.

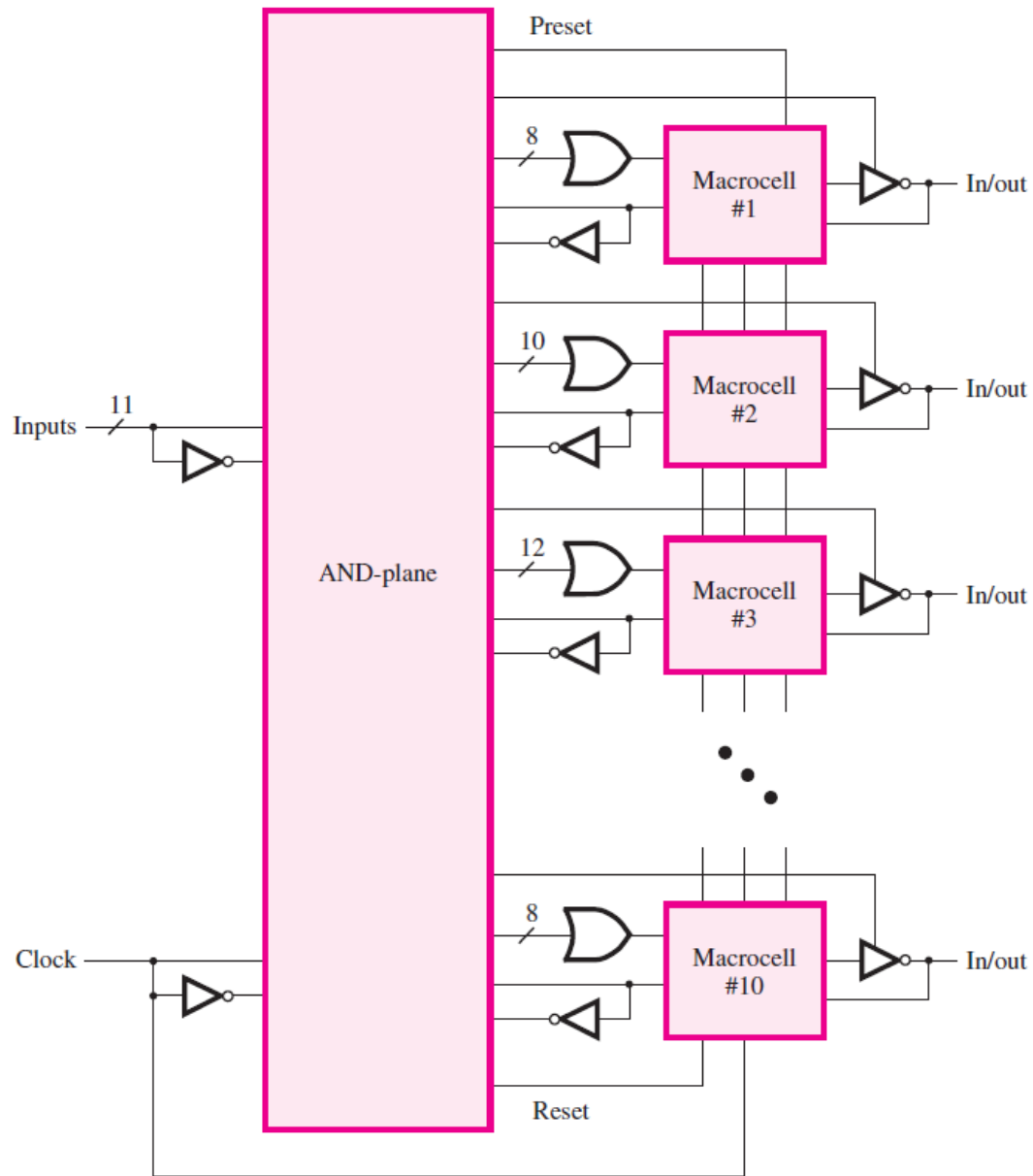


Figure B.29. The 22V10 PAL device.

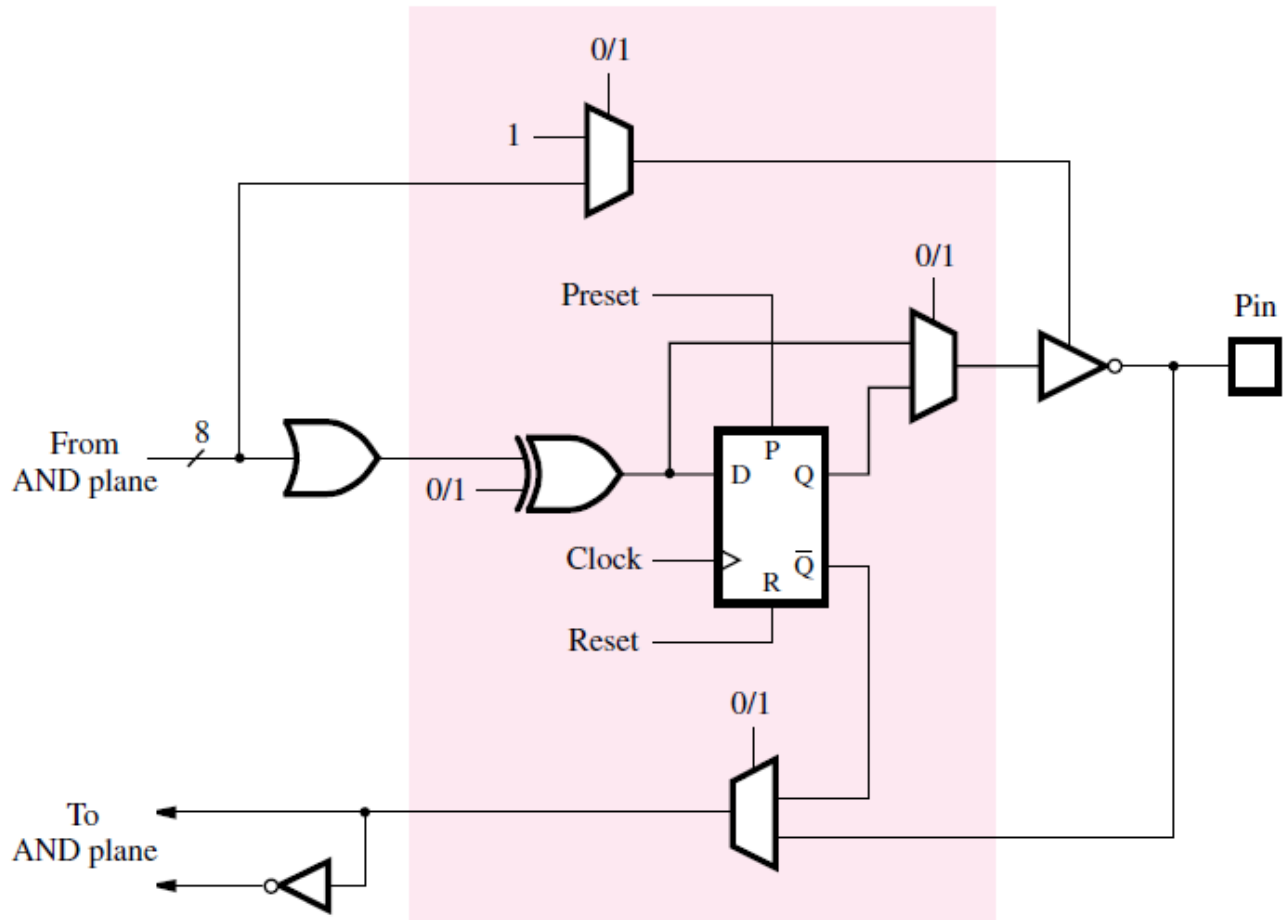


Figure B.30. The 22V10 macrocell.

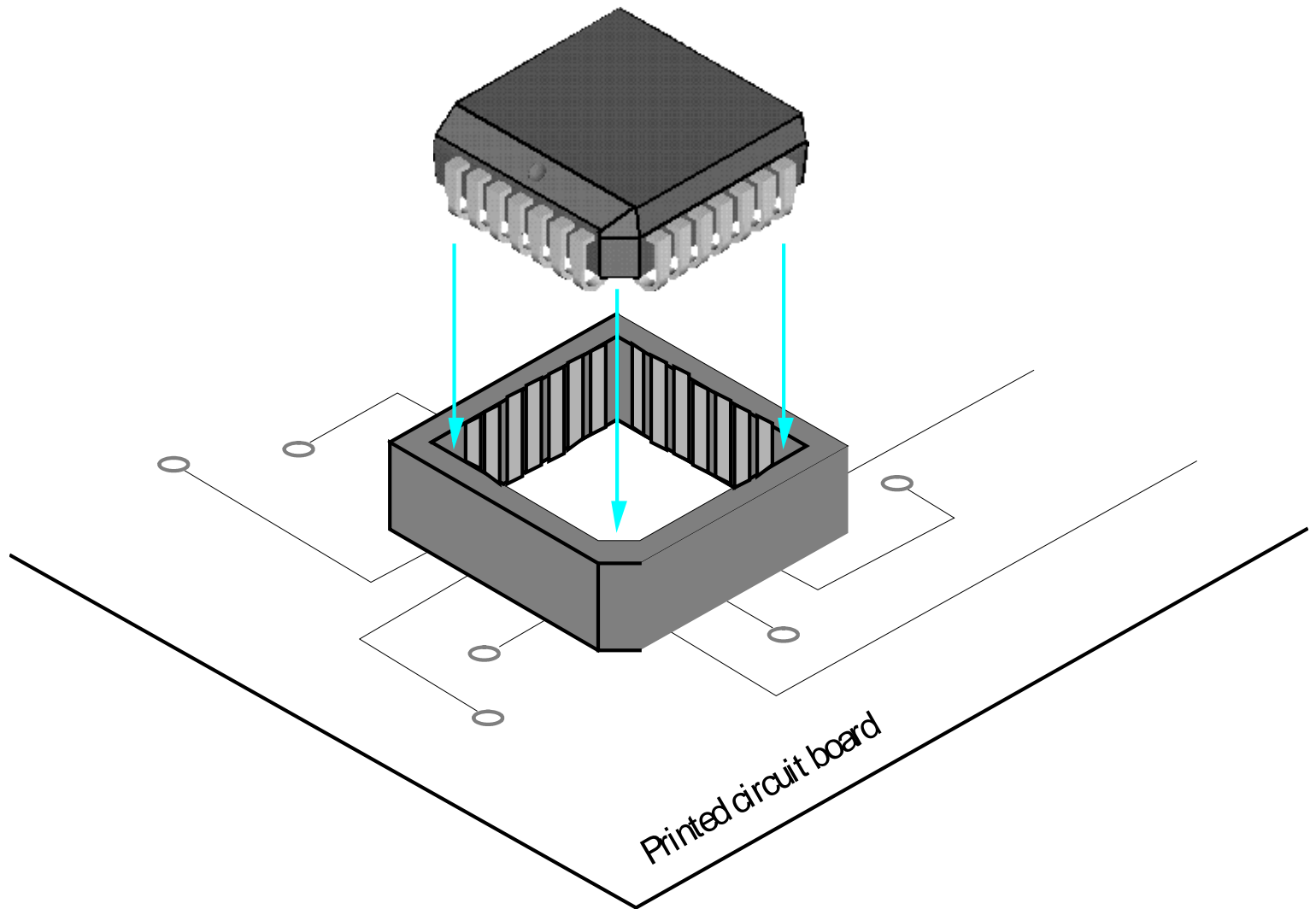


Figure B.31. A PLCC package with socket.

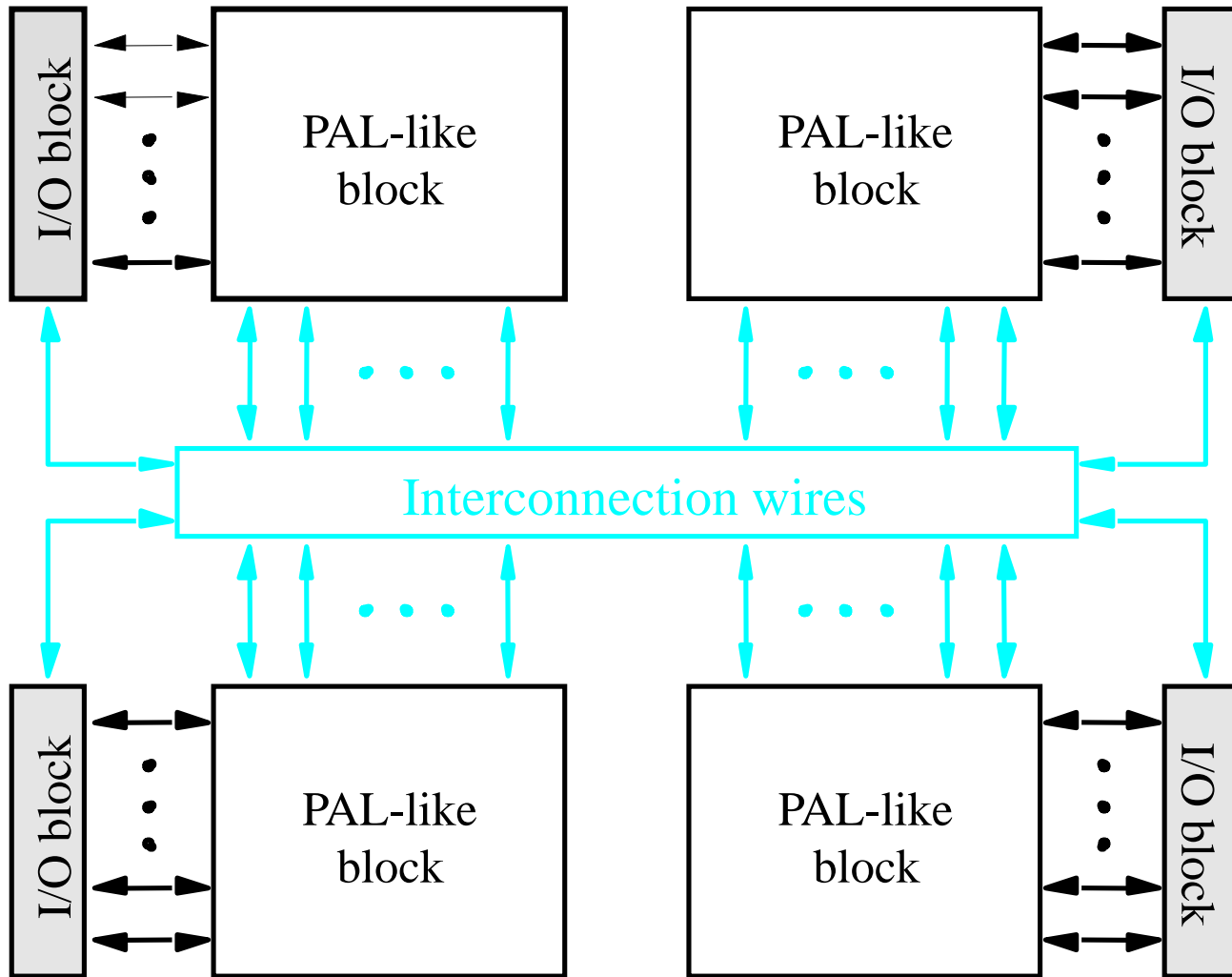


Figure B.32. Structure of a complex programmable logic device (CPLD).

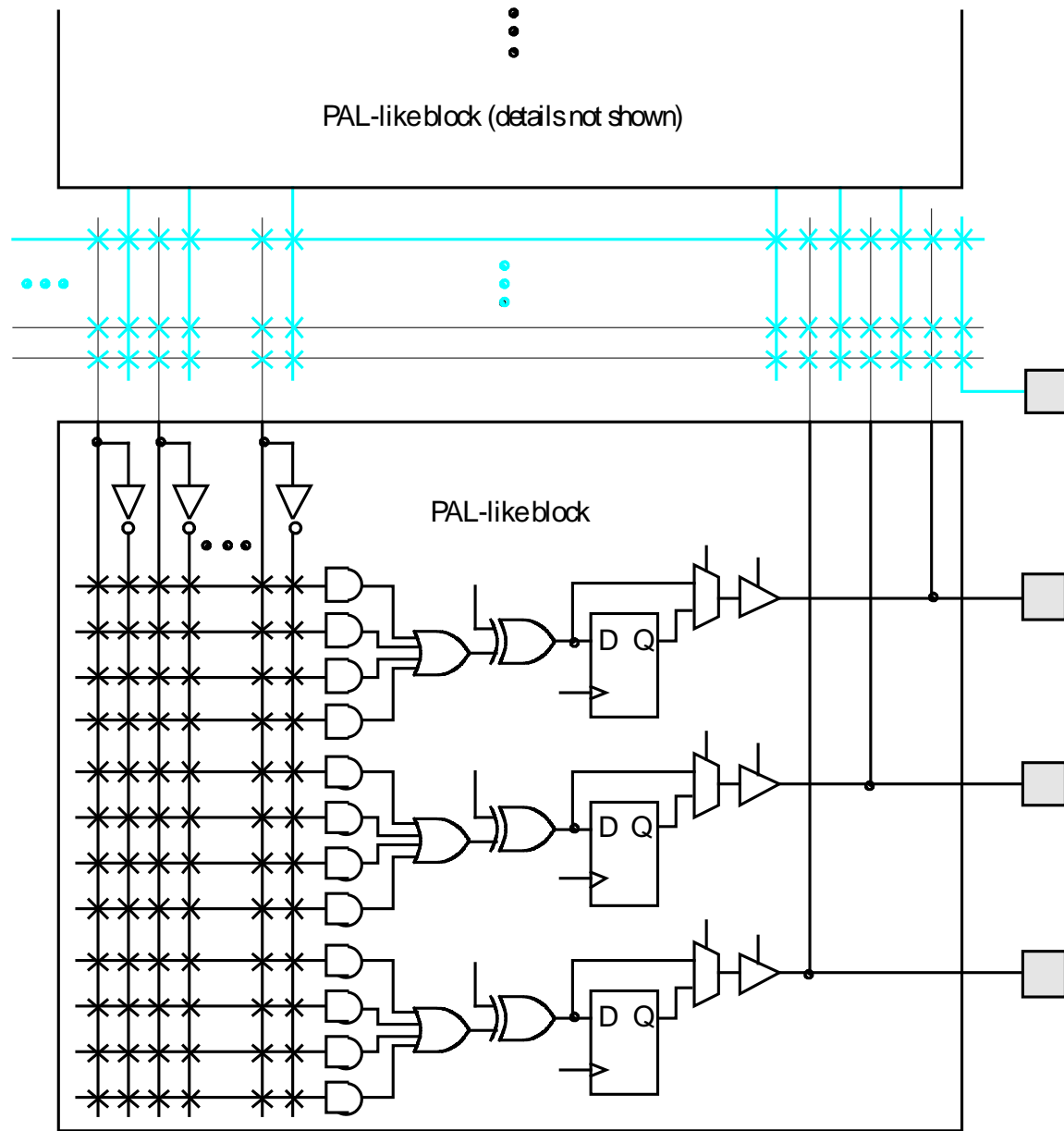
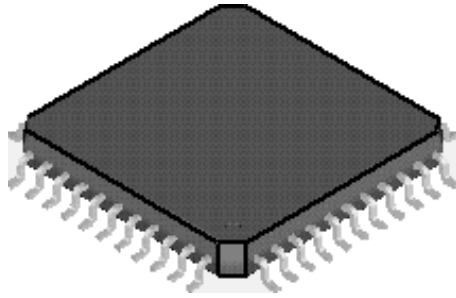
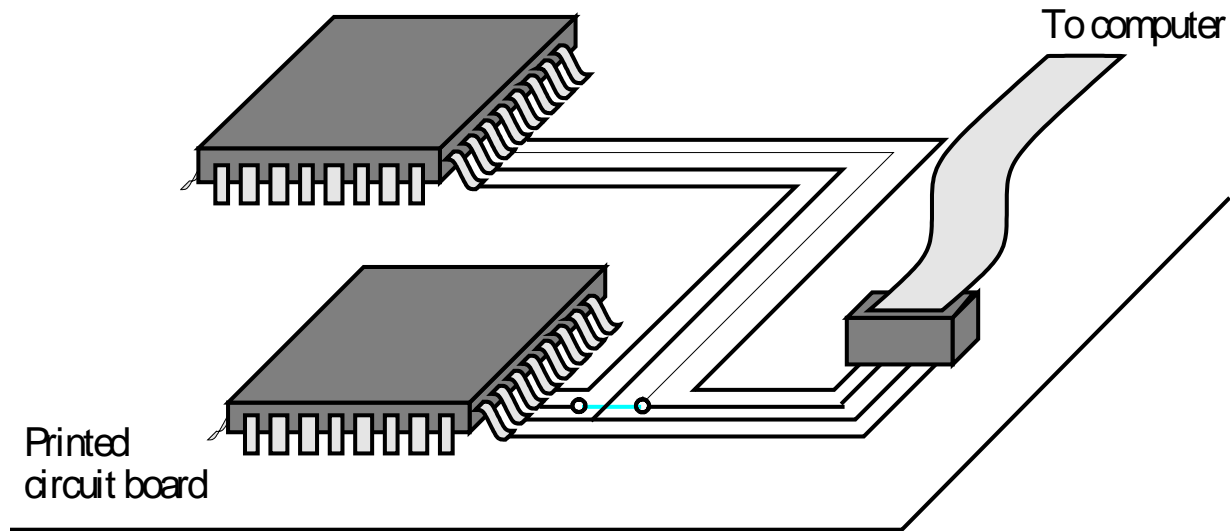


Figure B.33. A section of the CPLD in Figure B.32.



(a) CPLD in a Quad Flat Pack (QFP) package



(b) JTAG programming

Figure B.34. CPLD packaging and programming.

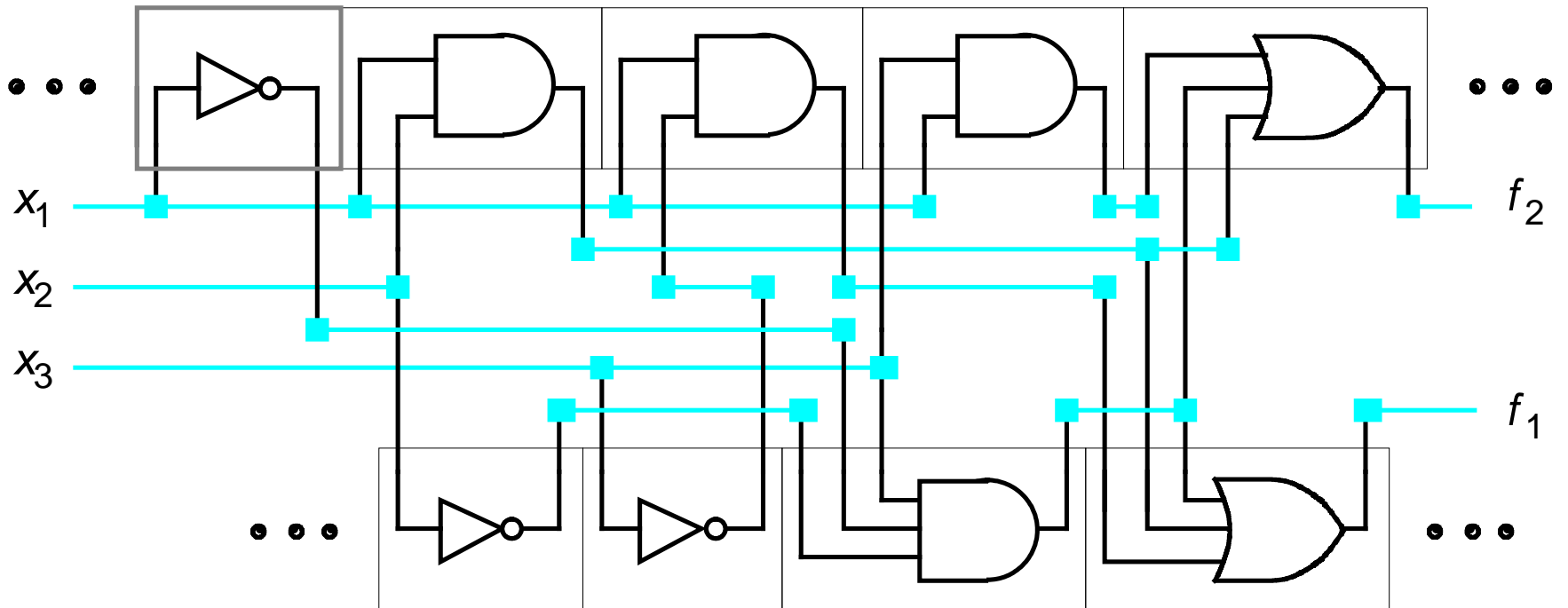


Figure B.40. A section of two rows in a standard-cell chip.

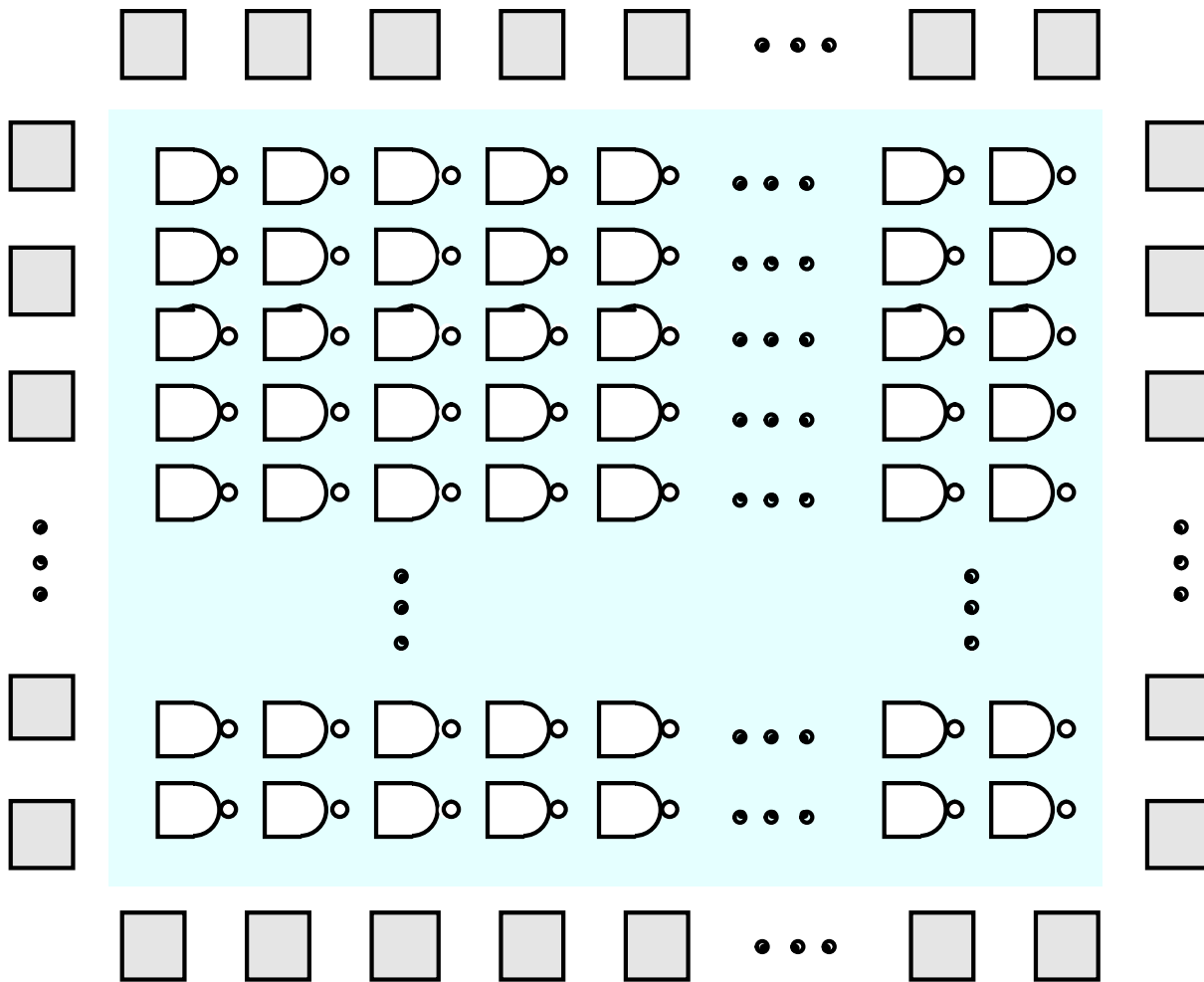


Figure B.41. A sea-of-gates gate array.

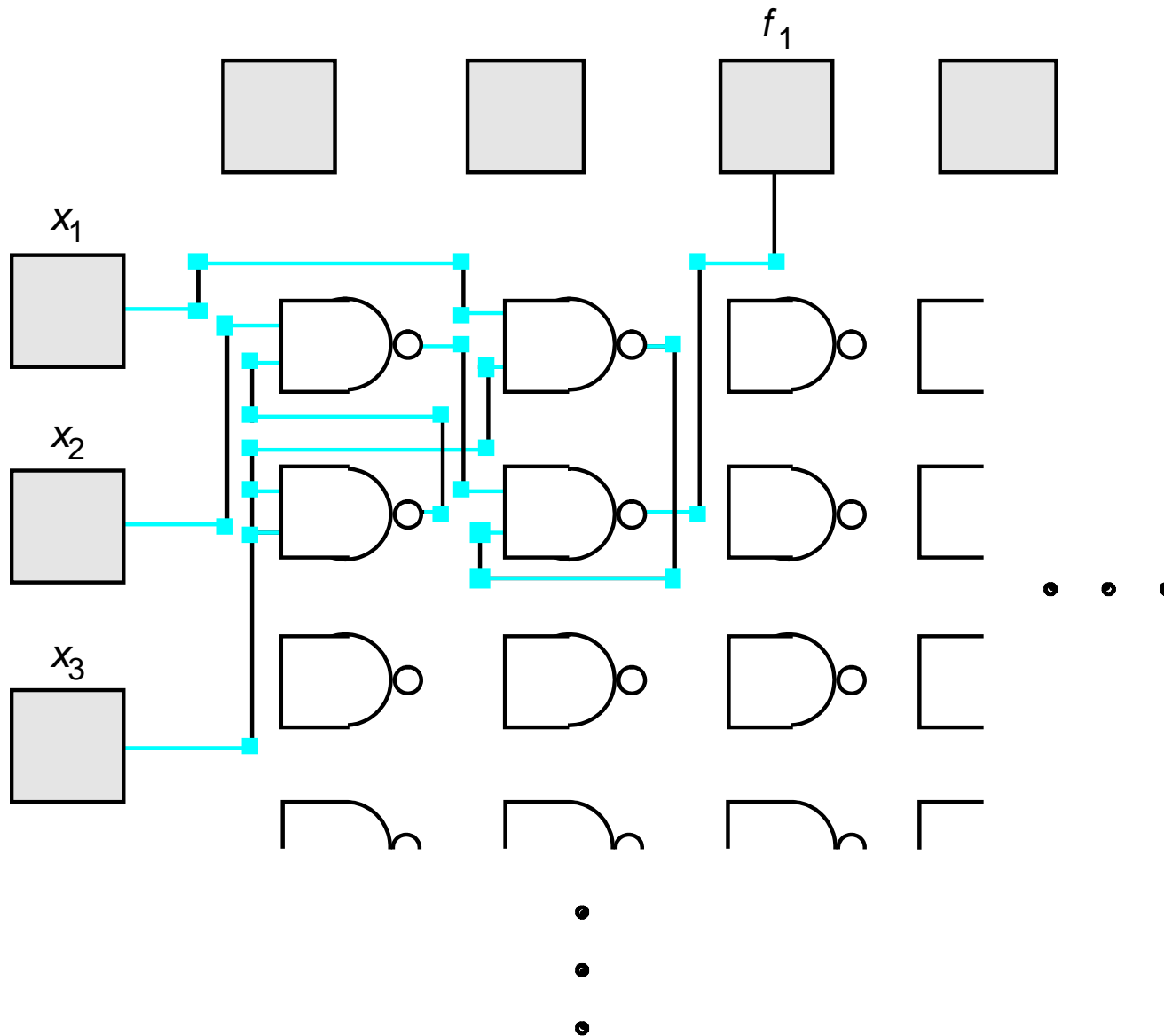
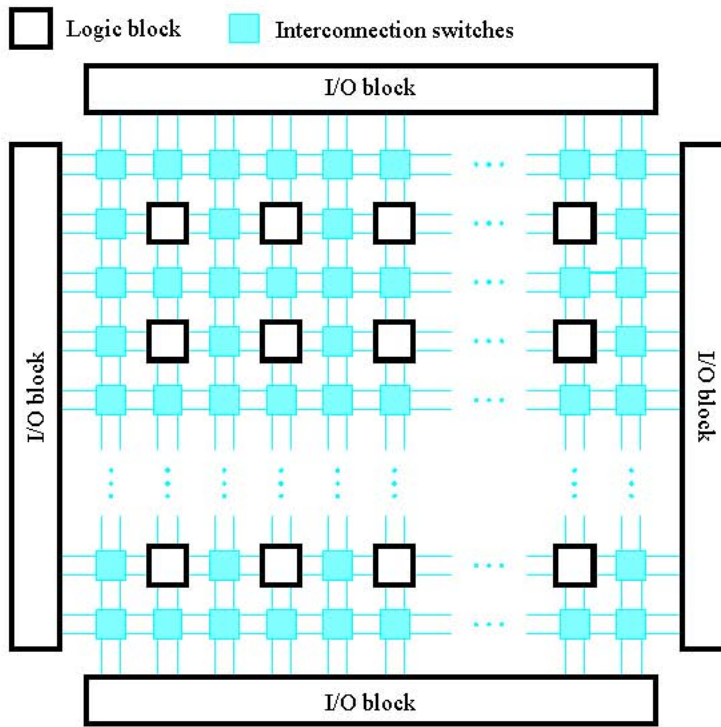
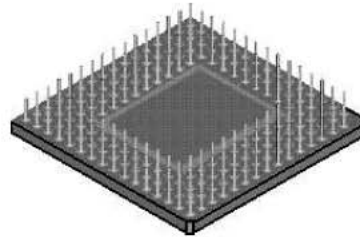


Figure B.42. The logic function $f_1 = x_2\bar{x}_3 + x_1x_3$ in the gate array of Figure B.41.

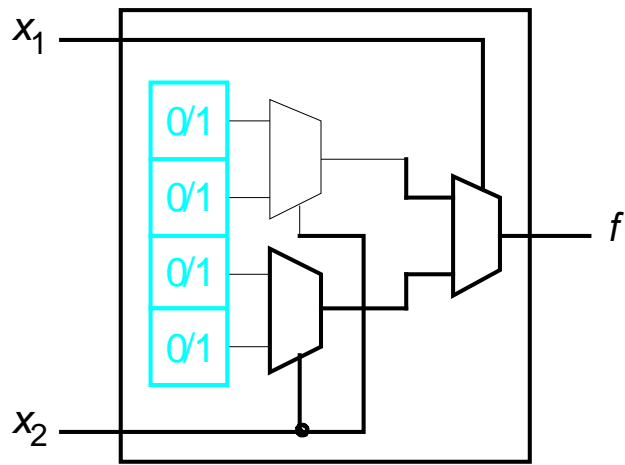


(a) General structure of an FPGA



(b) Pin grid array (PGA) package (bottom view)

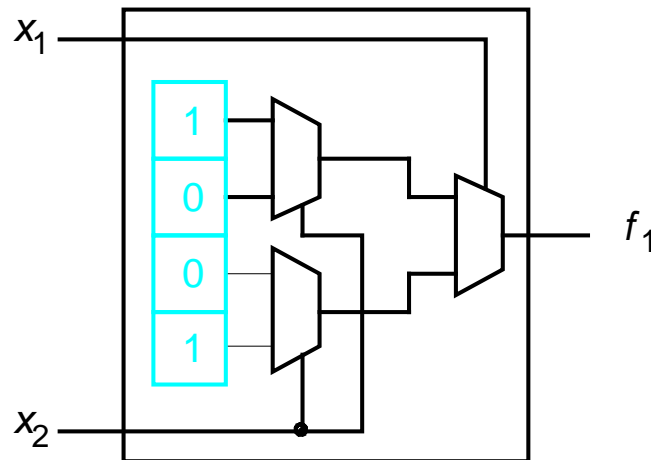
Figure B.35. A field-programmable gate array (FPGA).



(a) Circuit for a two-input LUT

x_1	x_2	f_1
0	0	1
0	1	0
1	0	0
1	1	1

(b) $f_1 = \bar{x}_1\bar{x}_2 + x_1x_2$



(c) Storage cell contents in the LUT

Figure B.36. A two-input lookup table (LUT).

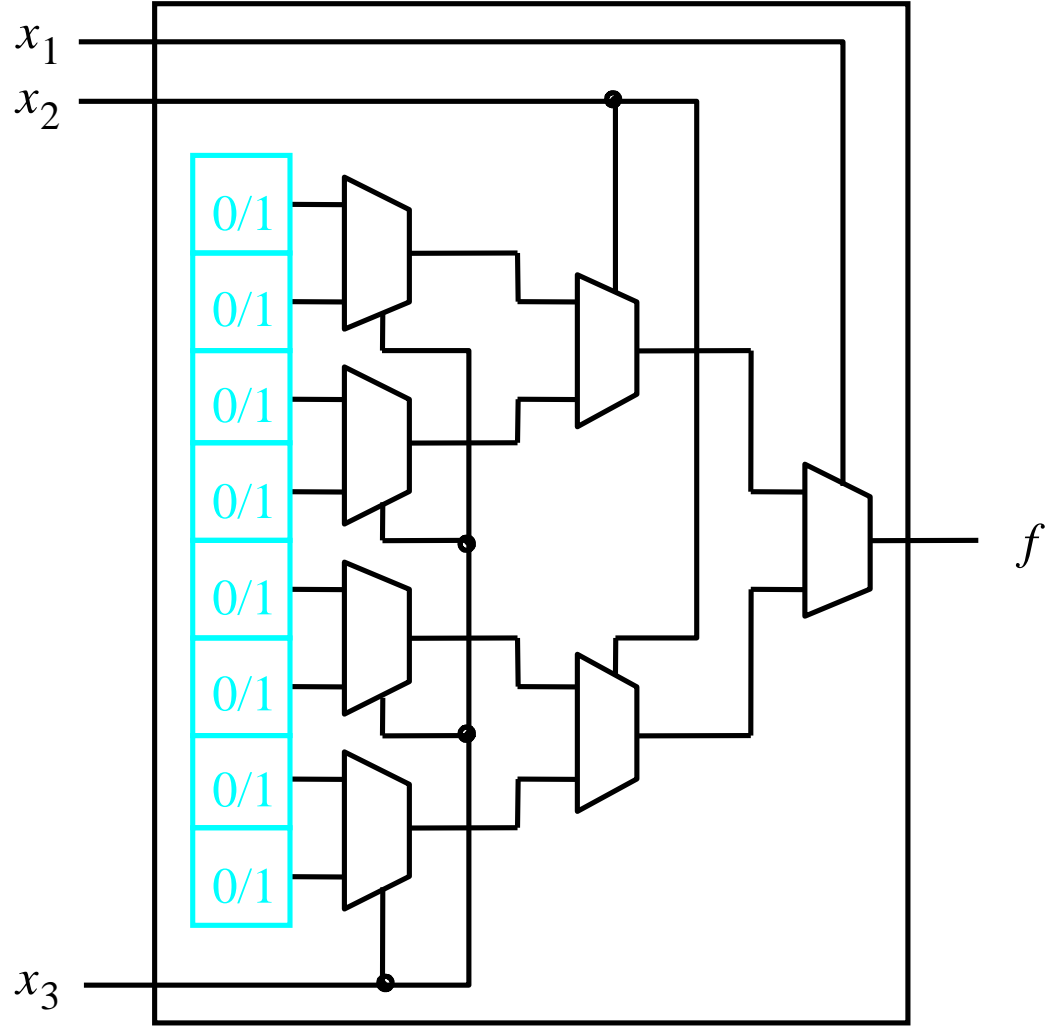


Figure B.37. A three-input LUT.

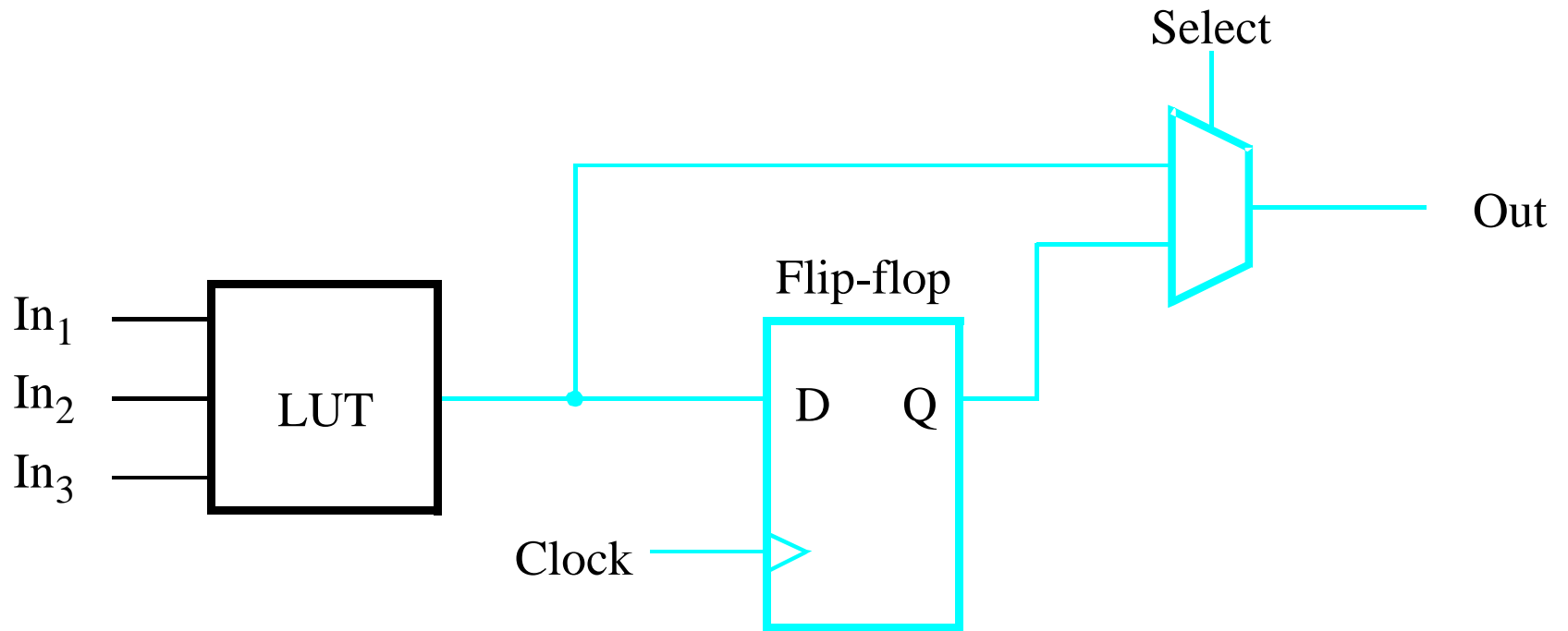
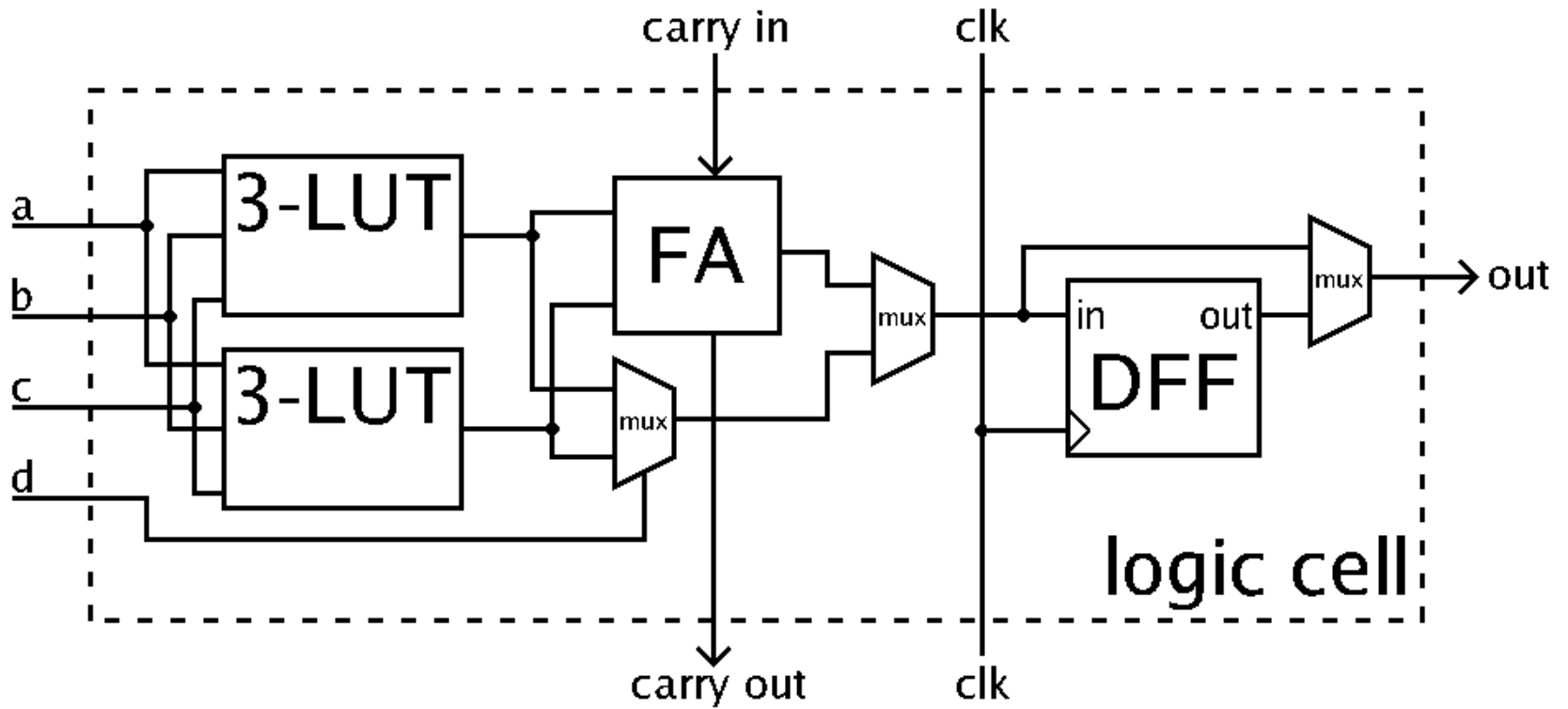


Figure B.38. Inclusion of a flip-flop in an FPGA logic block.



Source: https://upload.wikimedia.org/wikipedia/commons/1/1c/FPGA_cell_example.png

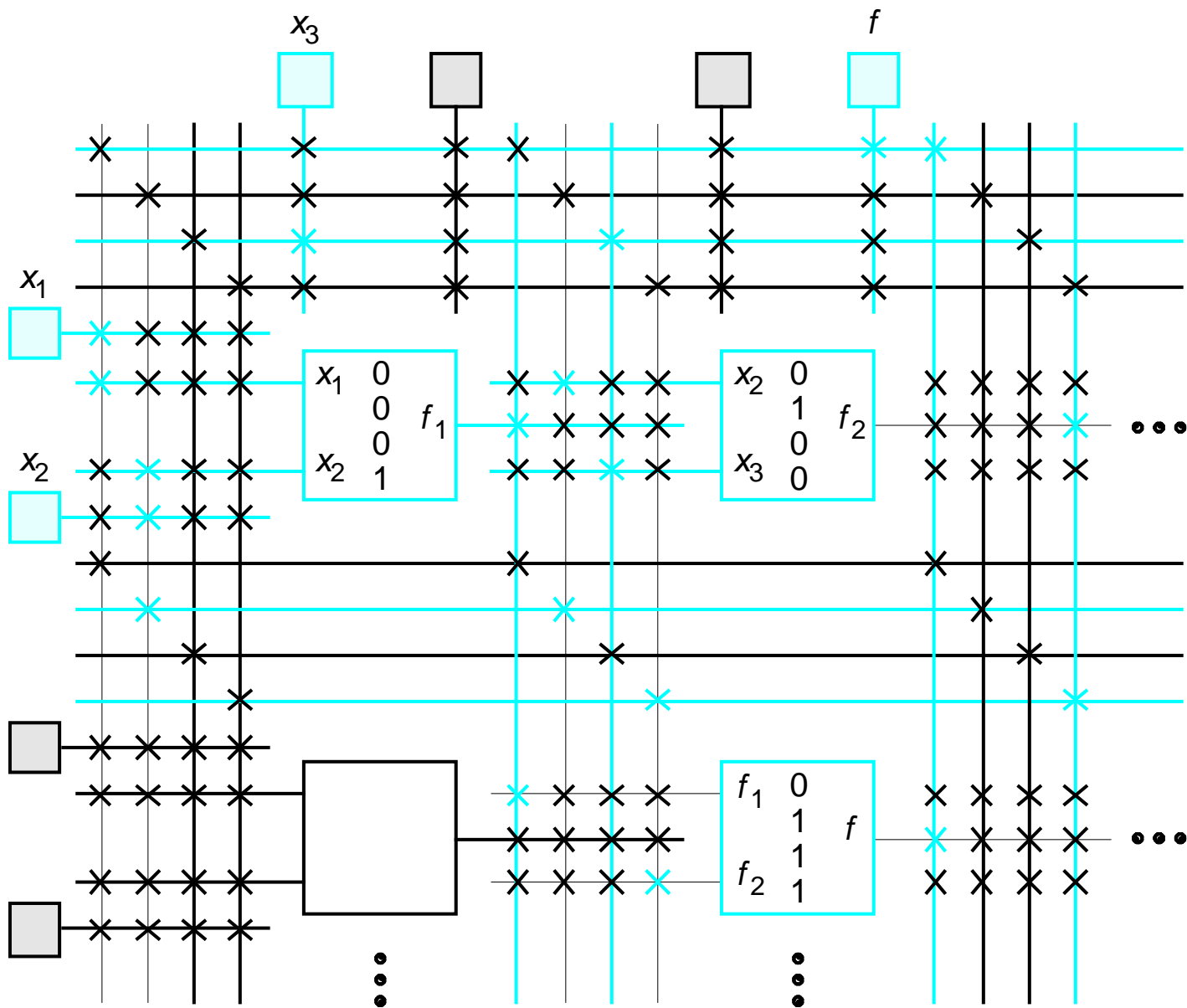


Figure B.39. A section of a programmed FPGA.

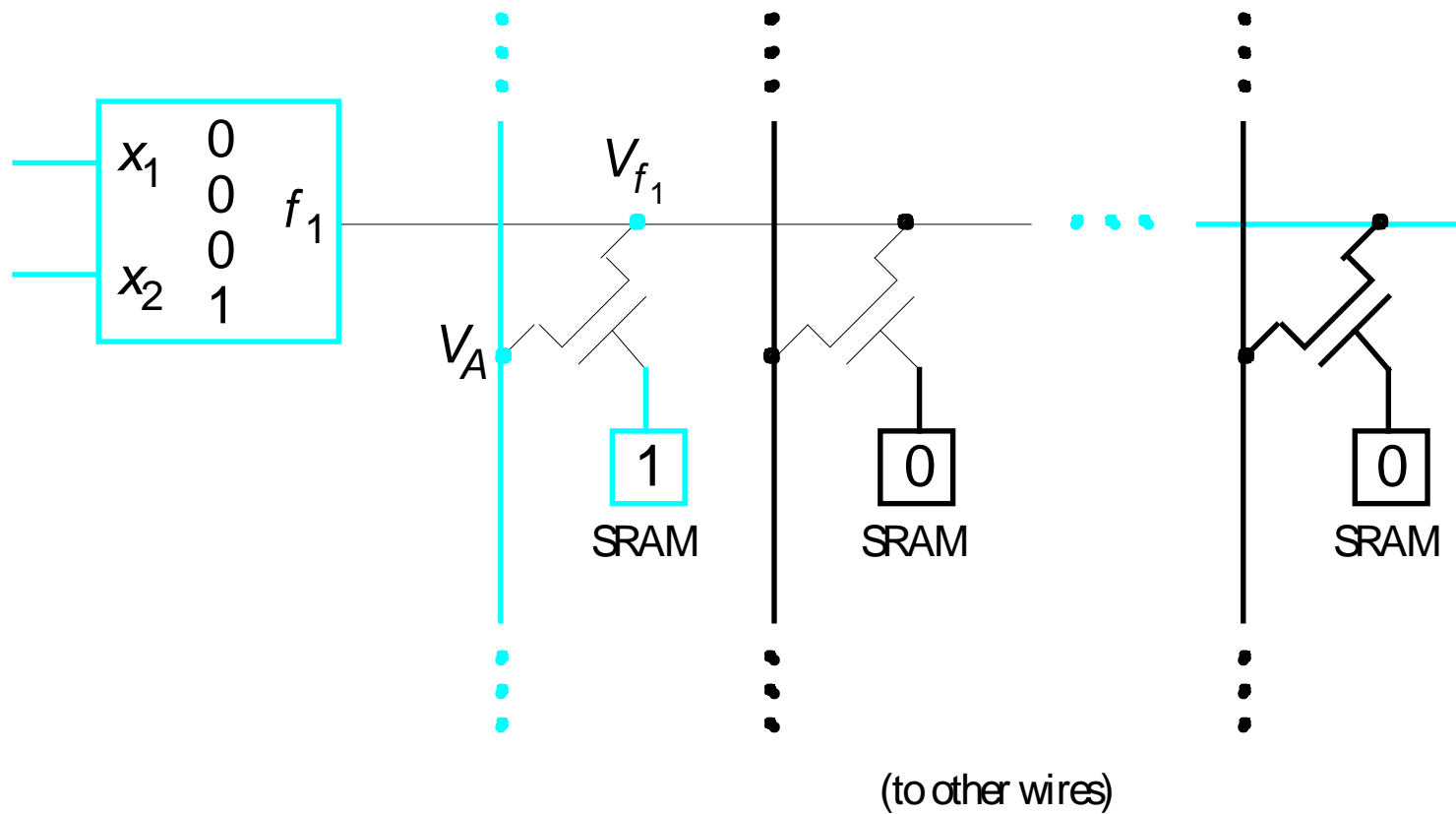


Figure B.73. Pass-transistor switches in FPGAs.