BEE 271 Digital circuits and systems

Spring 2017

Lecture 14: FSMs, memories, FPGAs

Nicole Hamilton

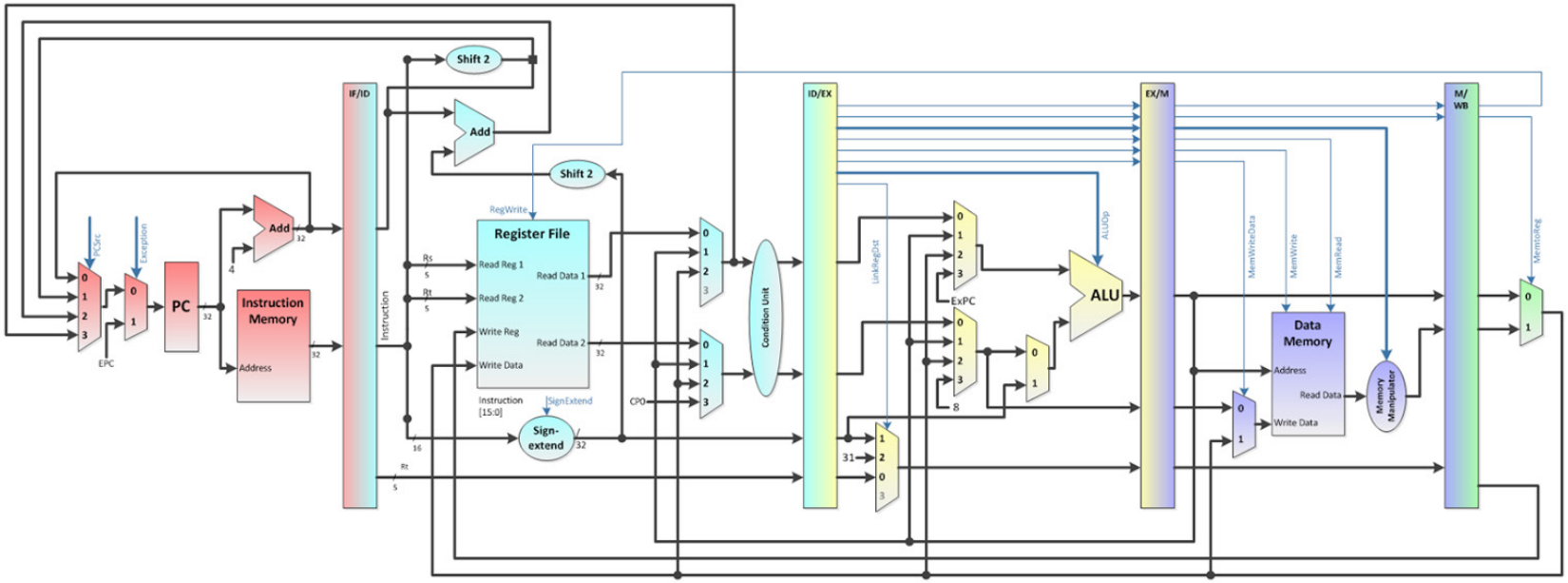
https://faculty.washington.edu/kd1uj

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.

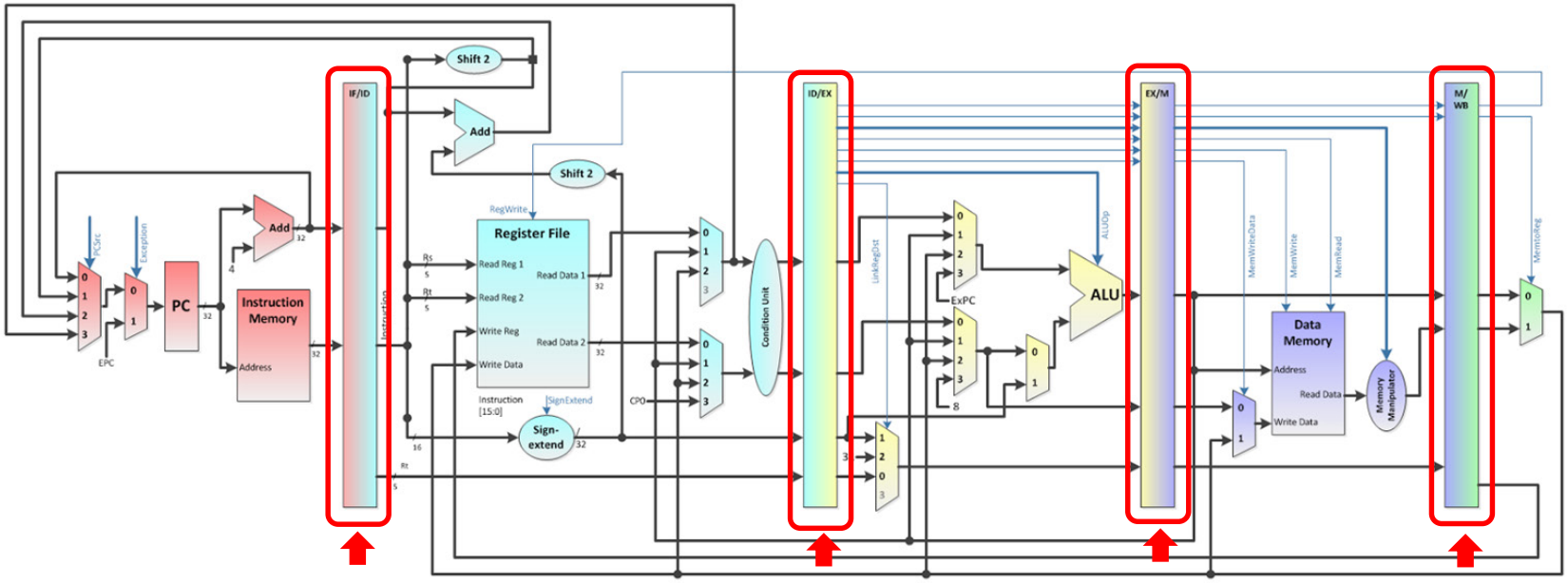
The classic RISC pipeline



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

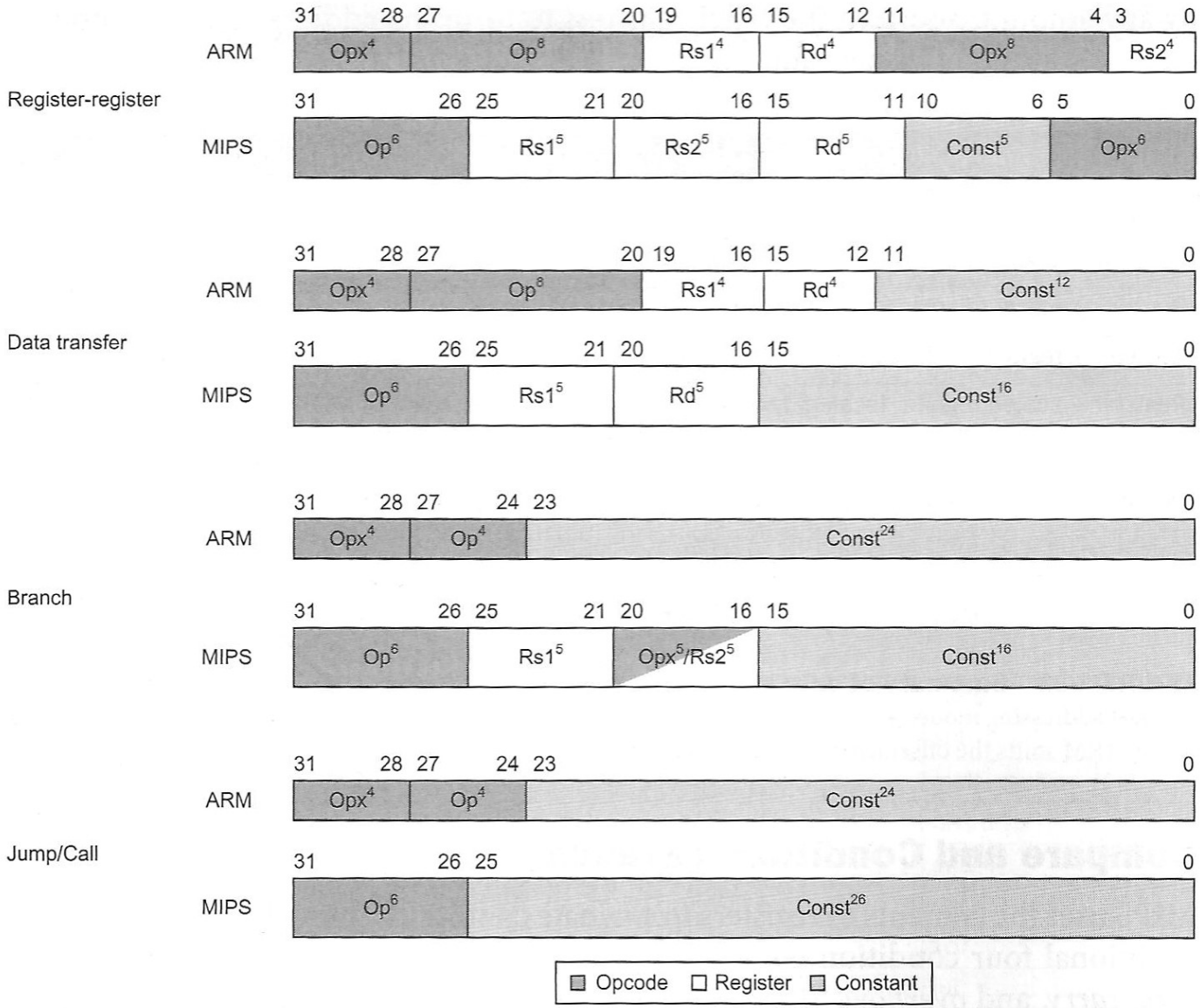
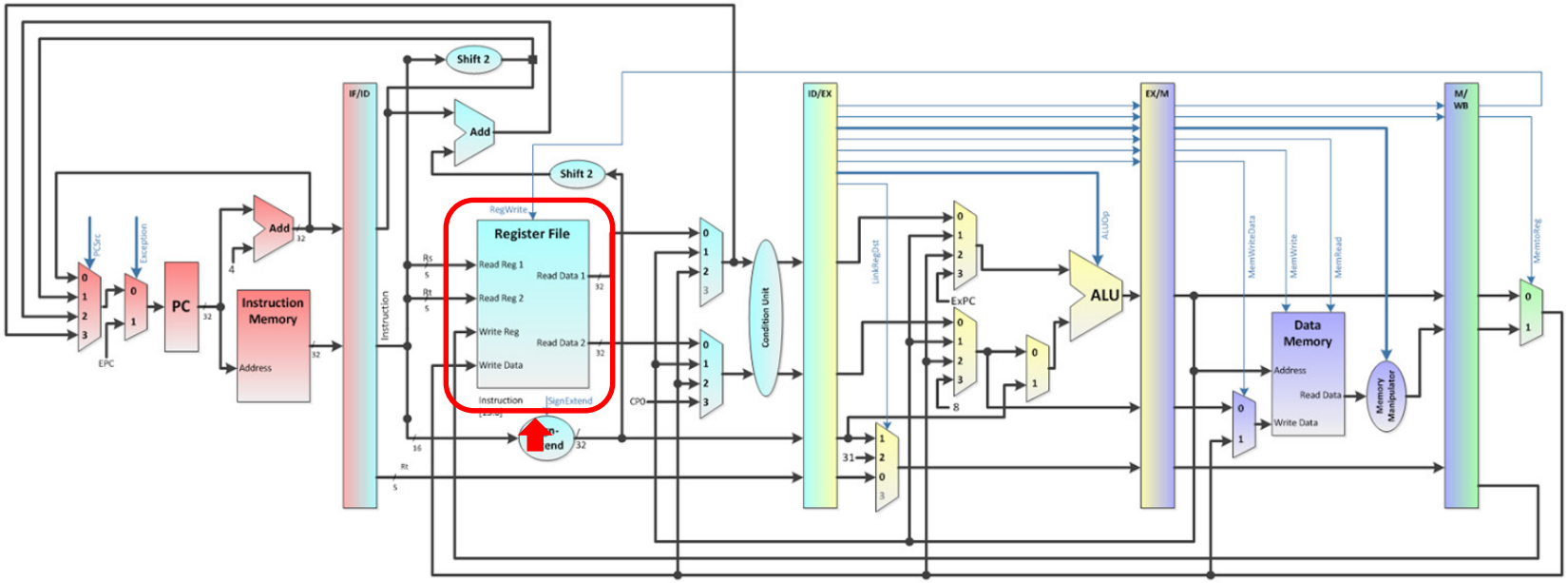


FIGURE 2.34 Instruction formats, ARM and MIPS. The differences result from whether the architecture has 16 or 32 registers.

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:

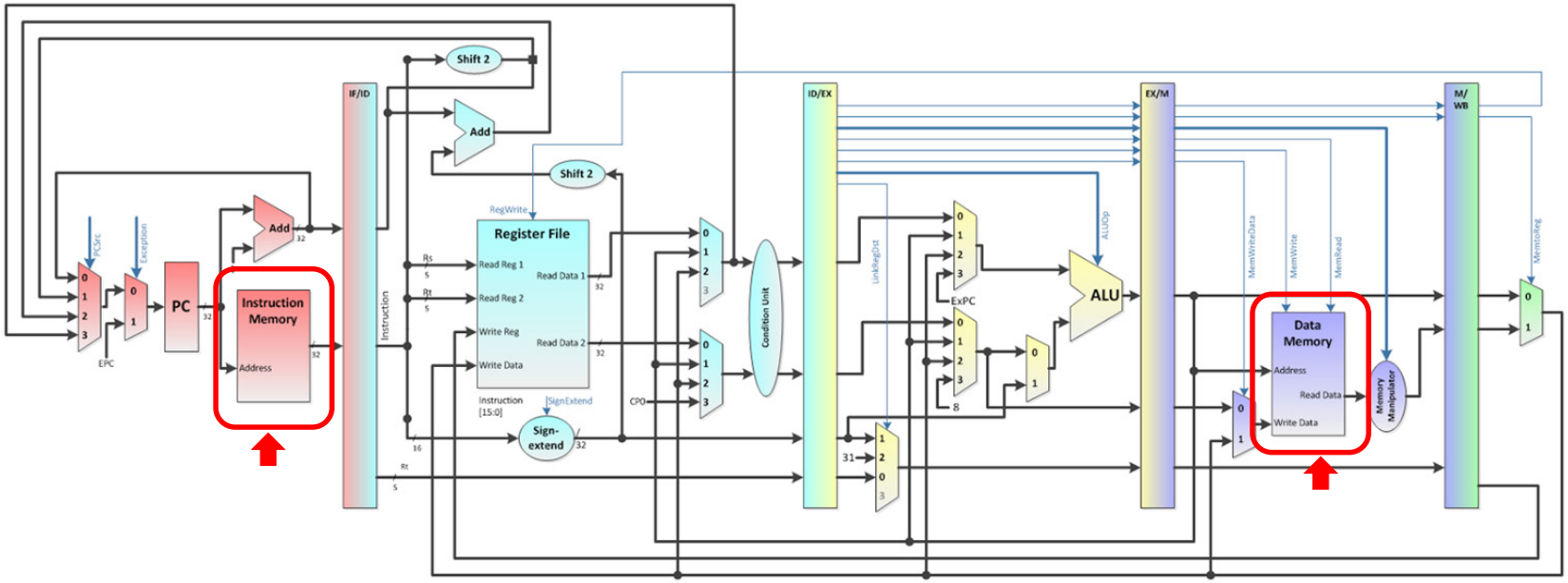
1. Read ports for A and B that continuously report the contents of registers A and B
2. Write port for C with an enable.

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

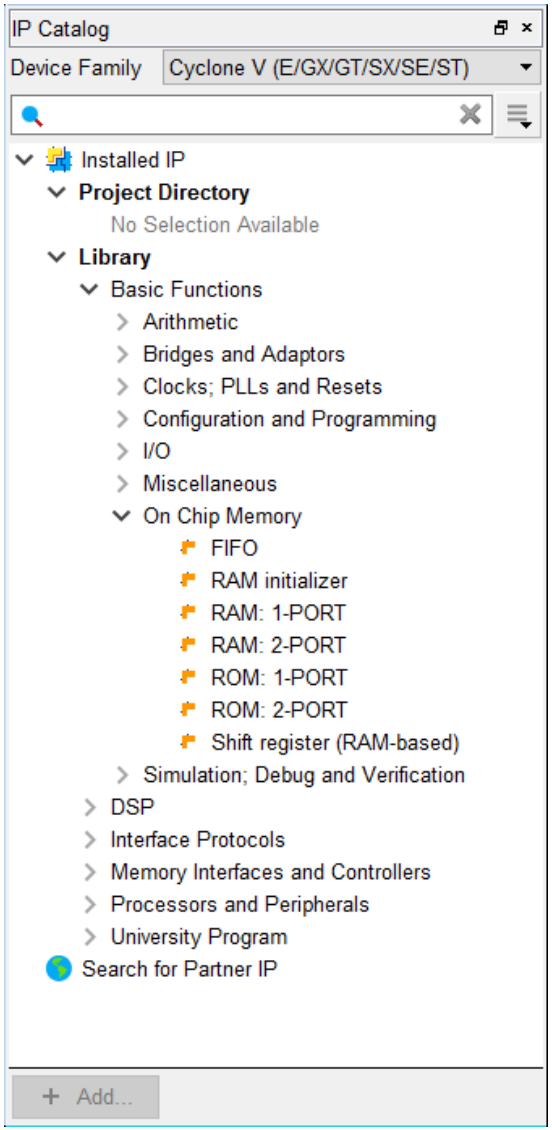
The MIPS is a Harvard architecture with separate instruction and data memories.



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.



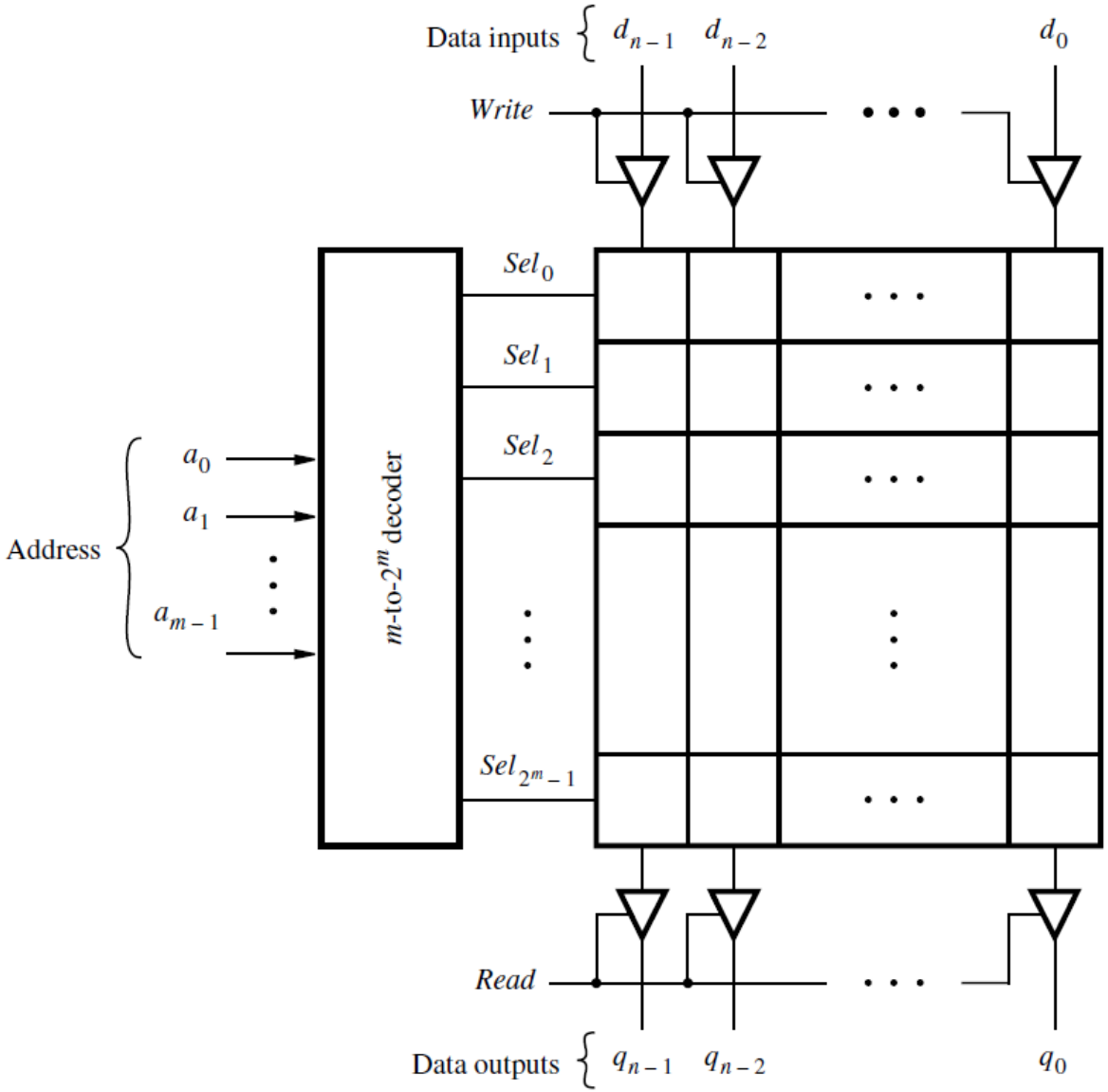
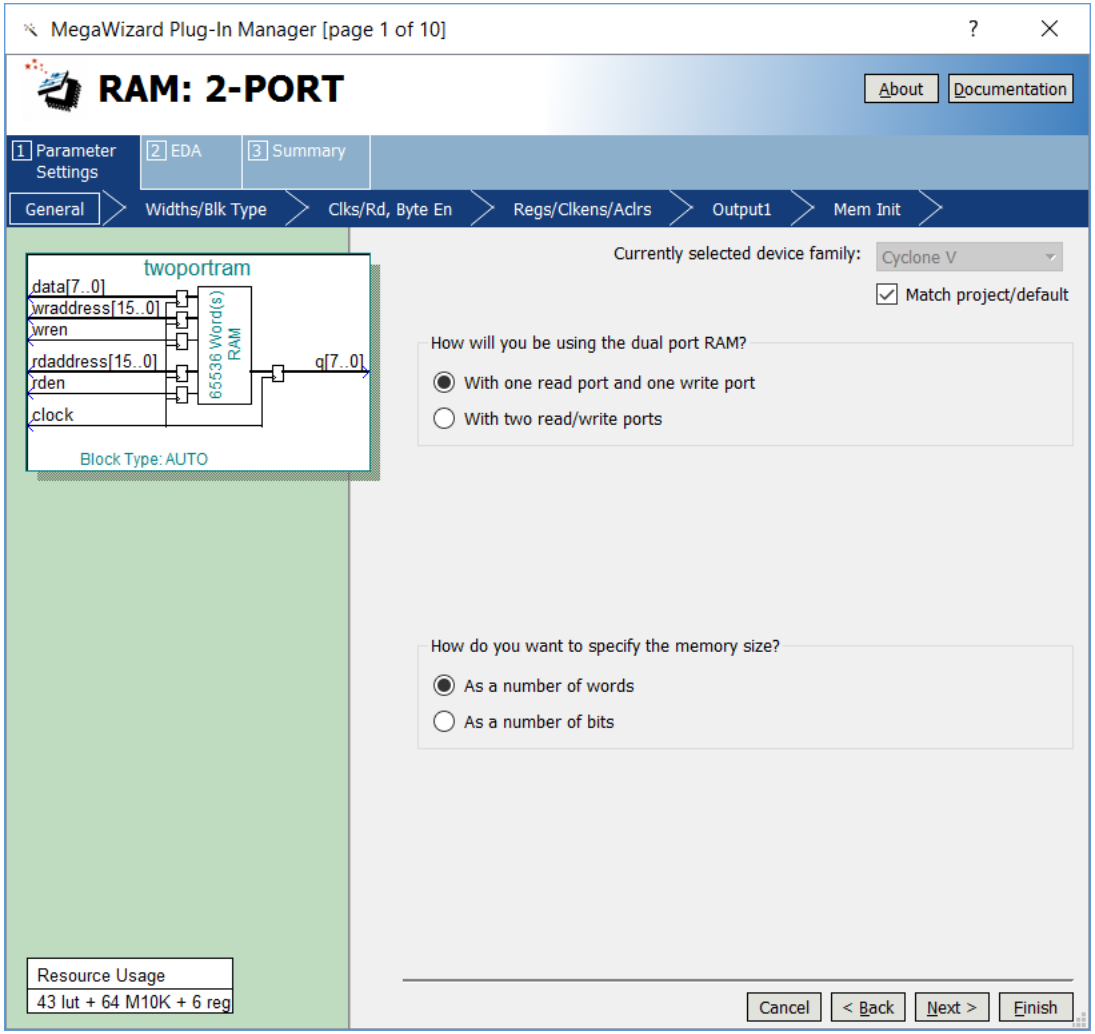


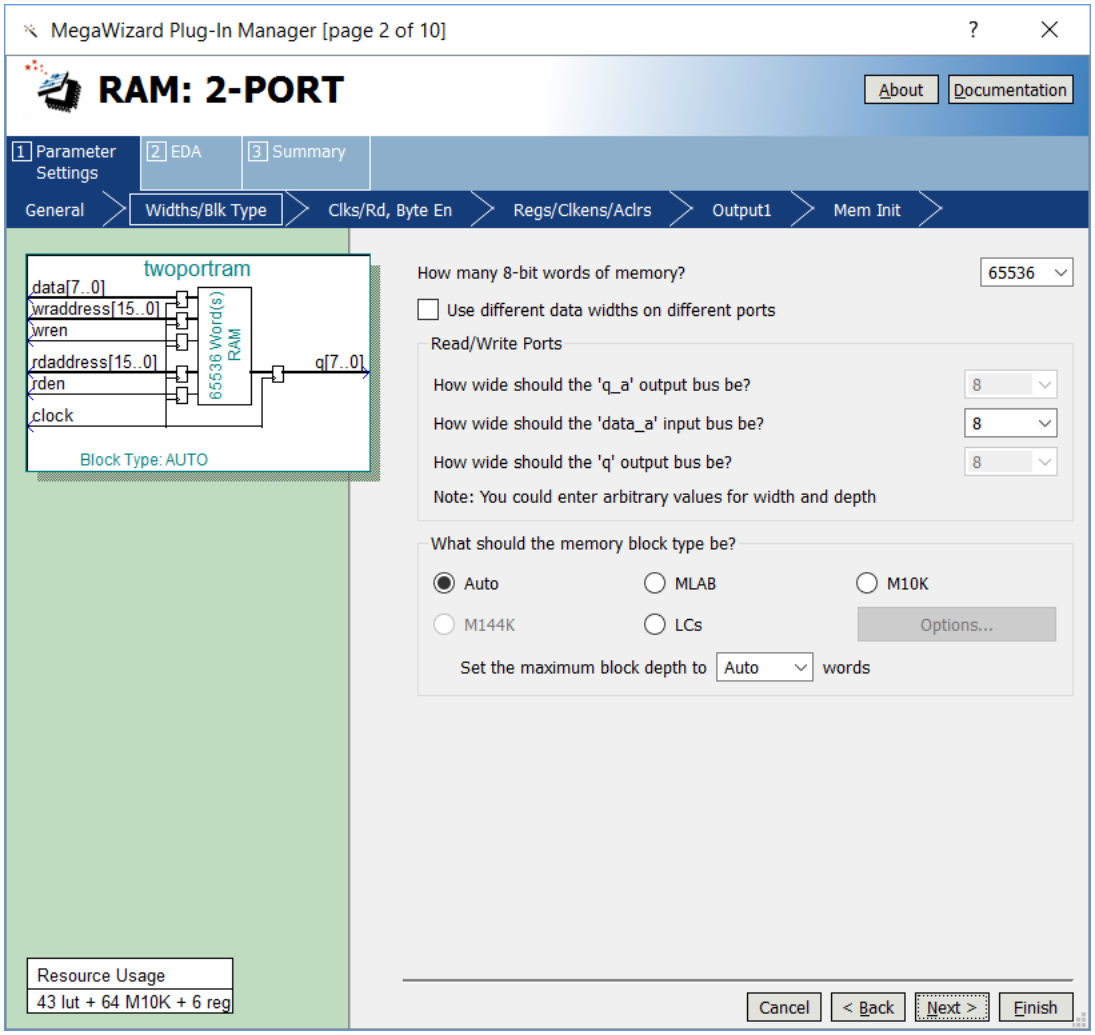
Figure B.66. A 2^m x n SRAM block.

An example

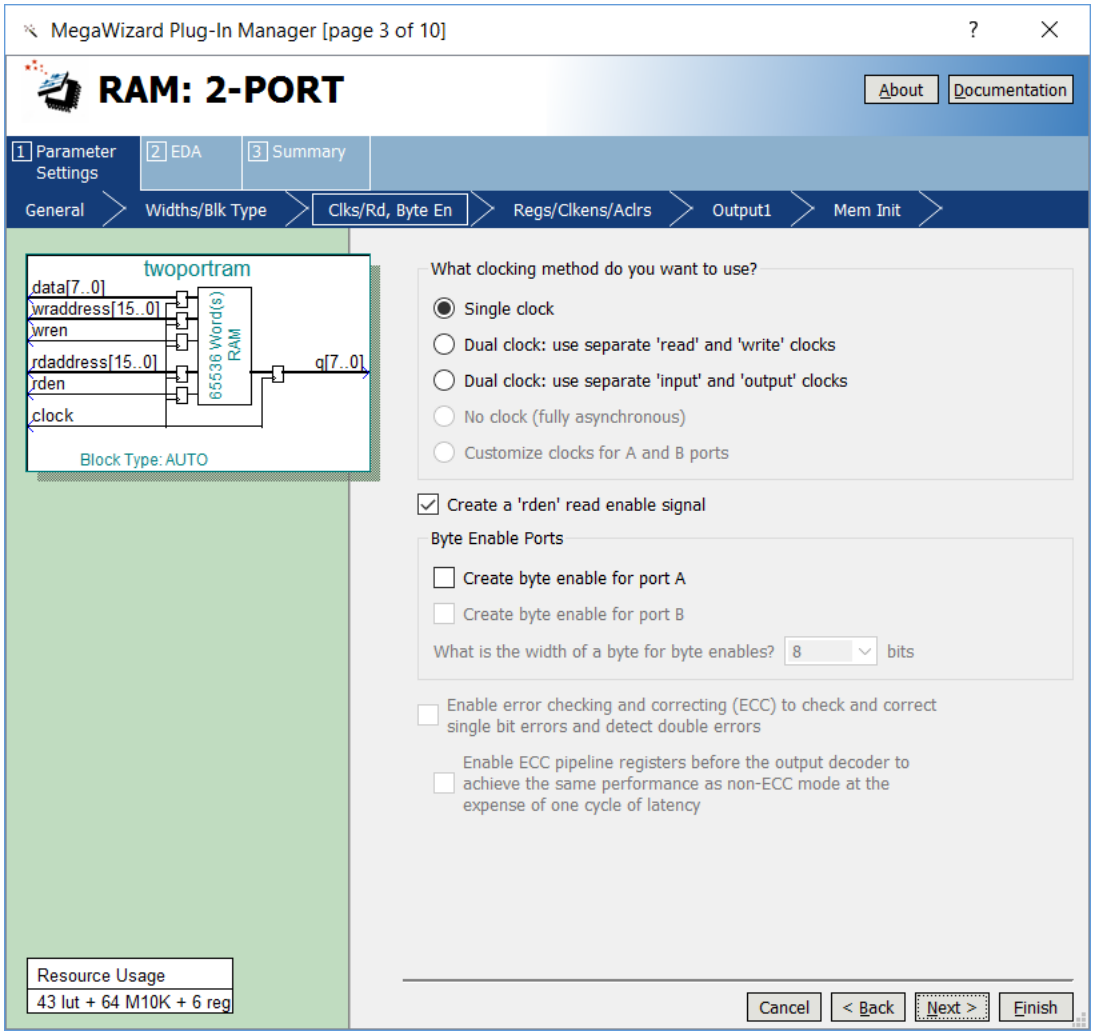
2-port RAM



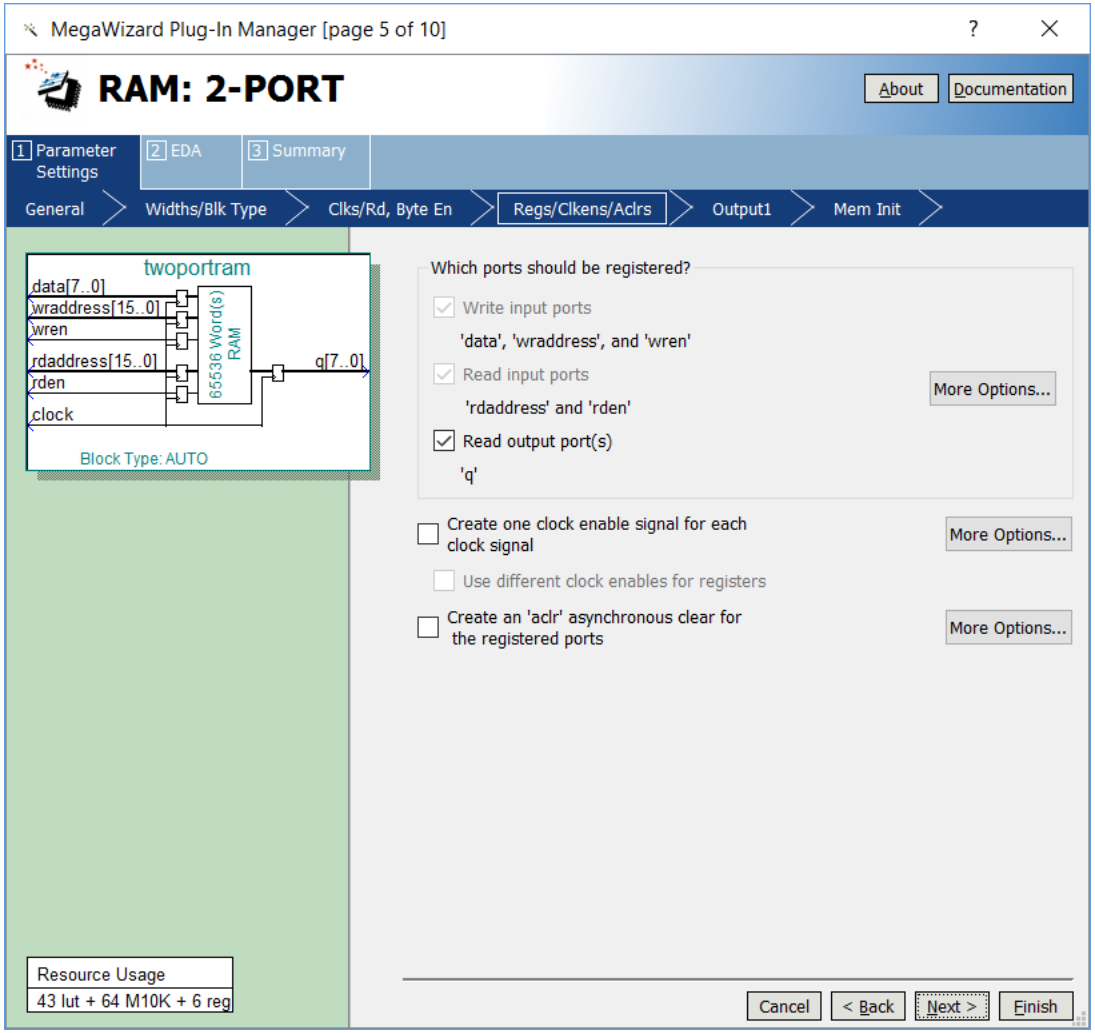
Specify the size and width.



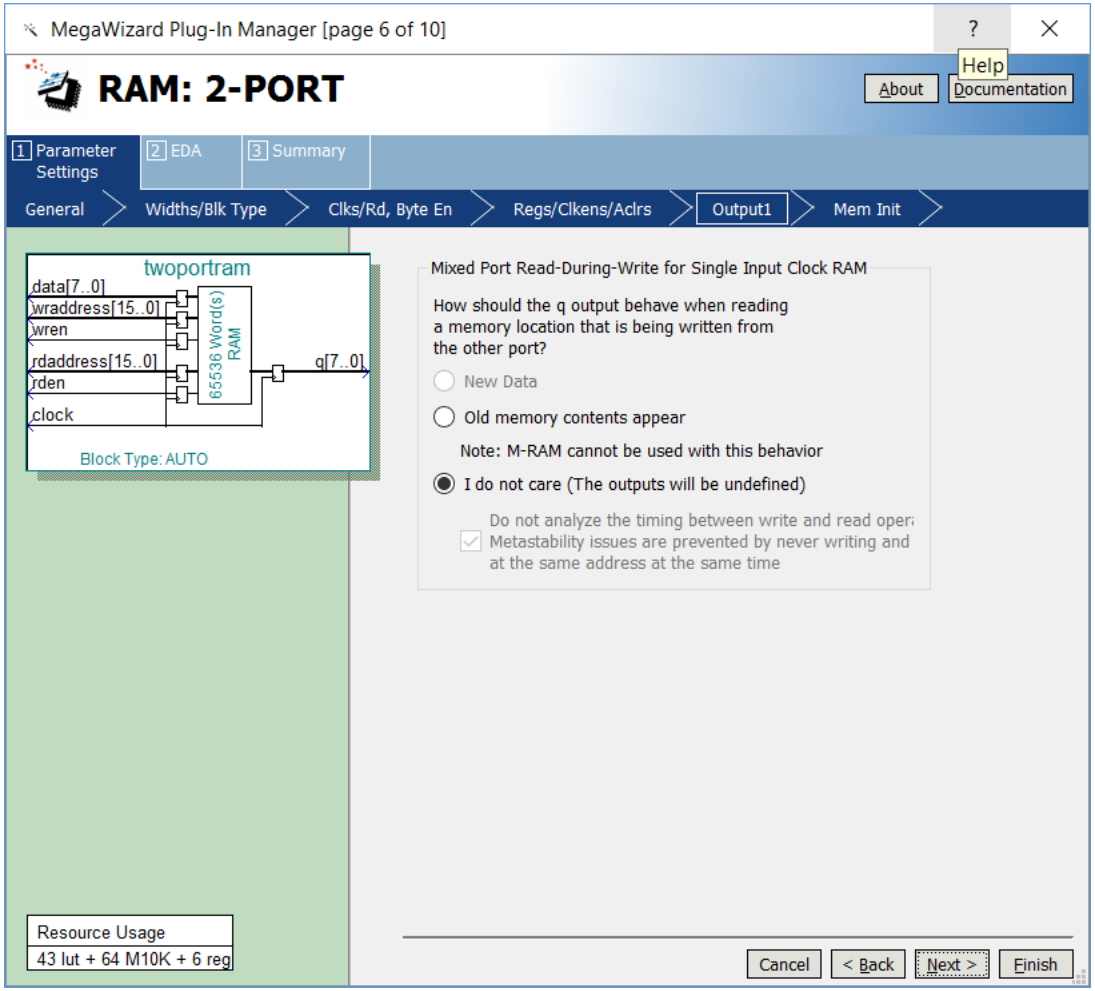
How it will be clocked.



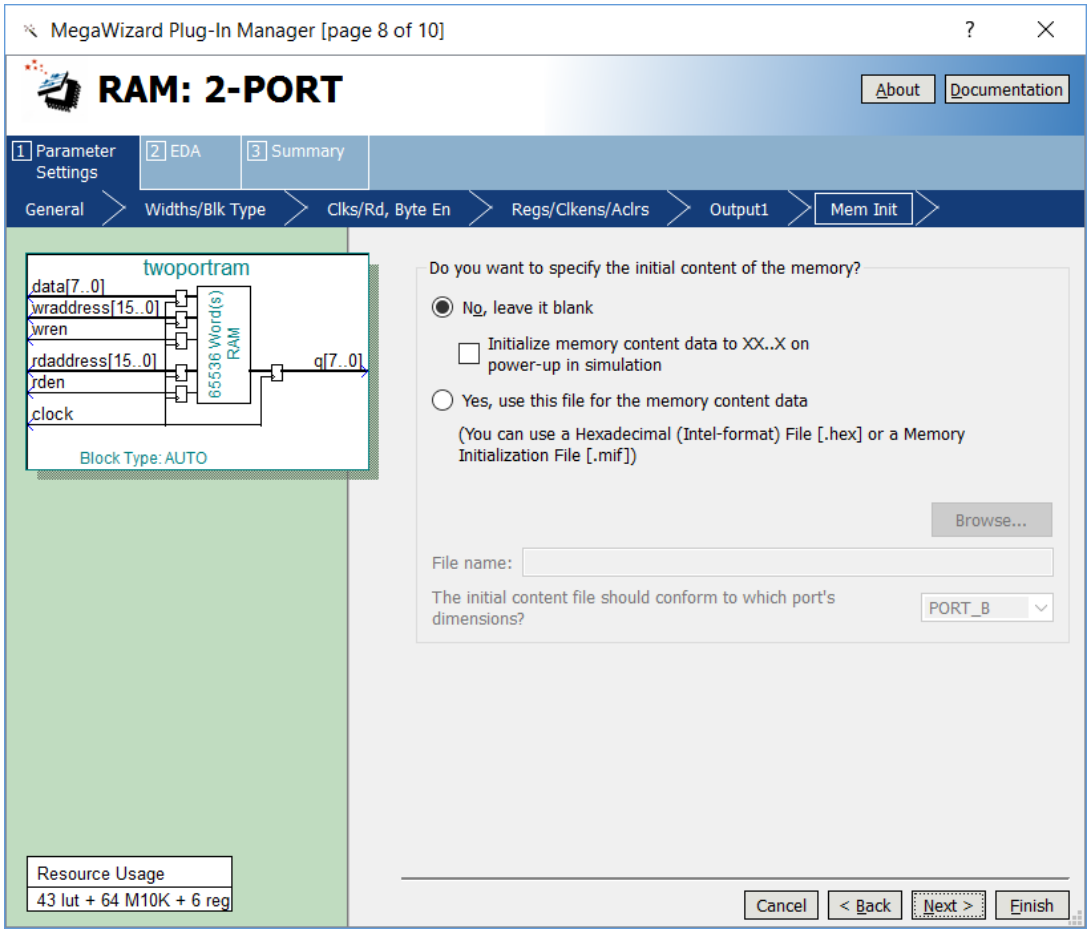
Which ports are latched.



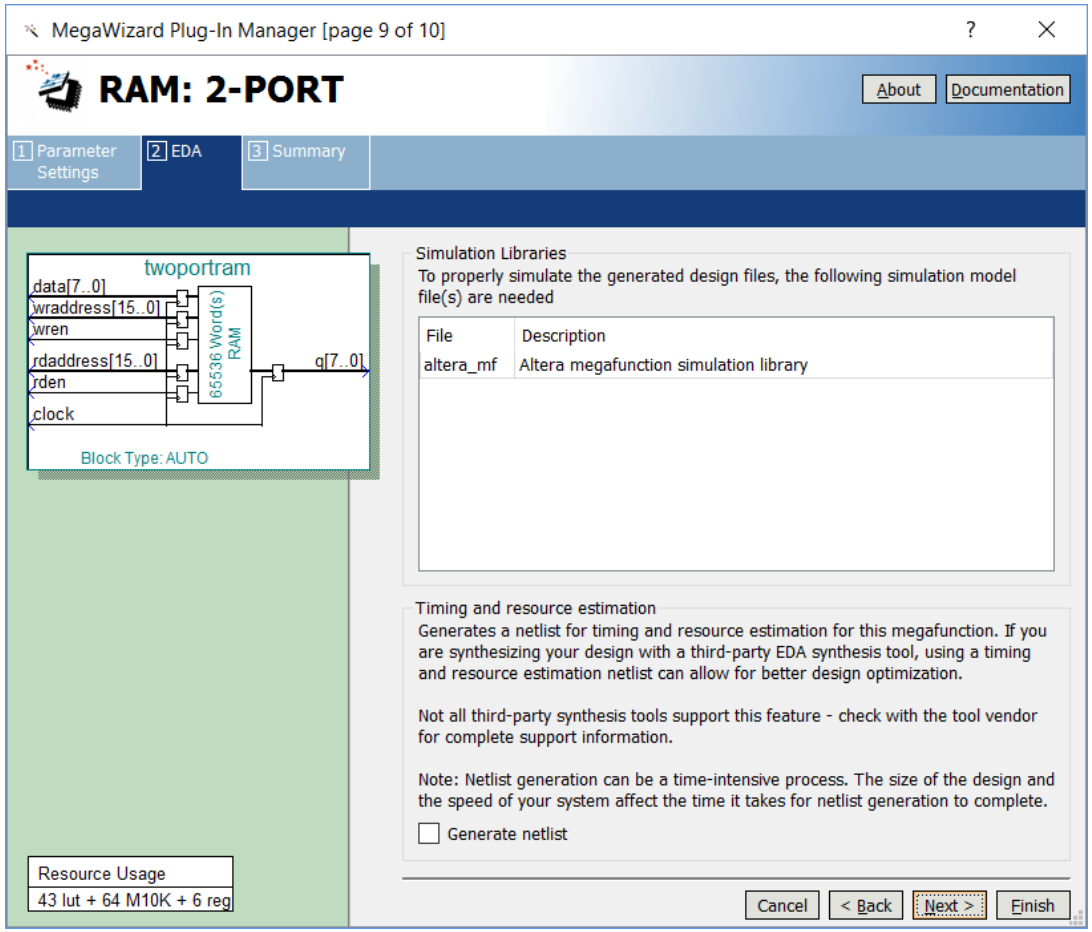
What happens if you try to read a location that's being written.



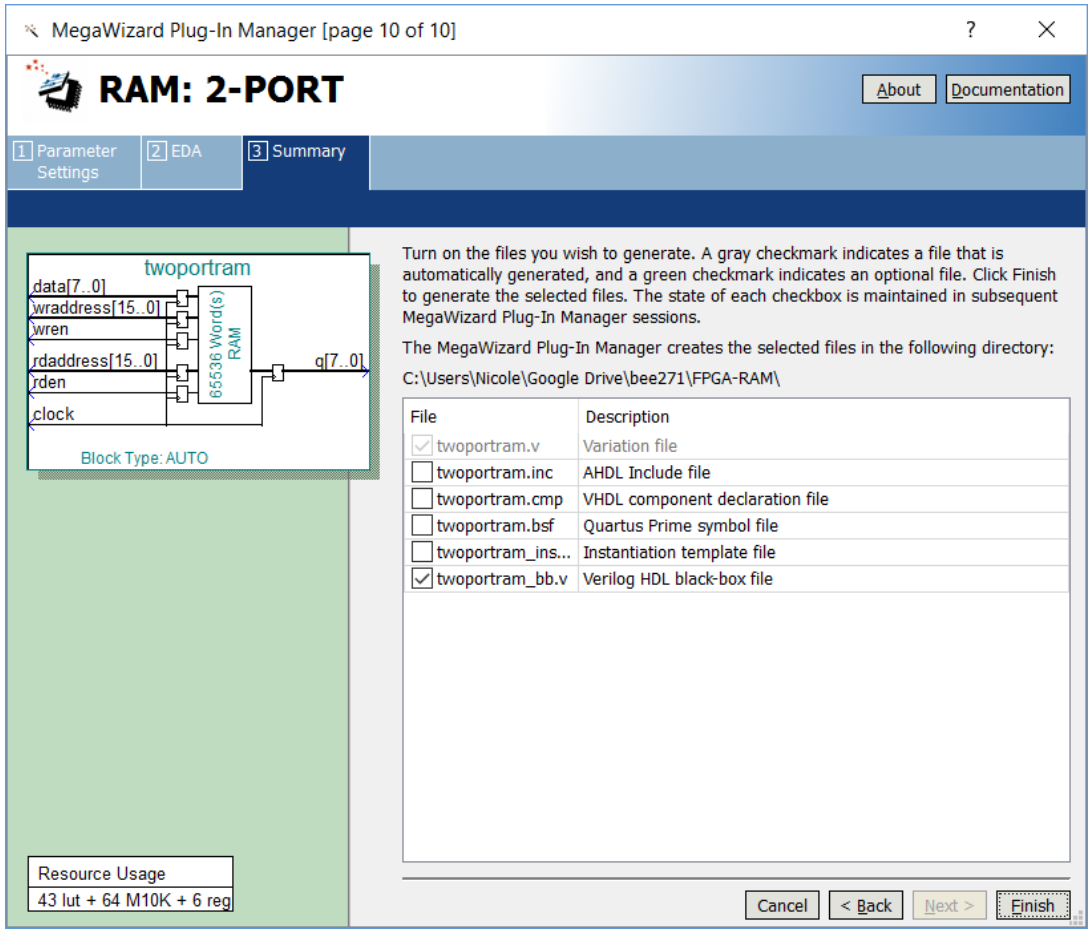
Initial values.



Other vendor IP libraries needed.



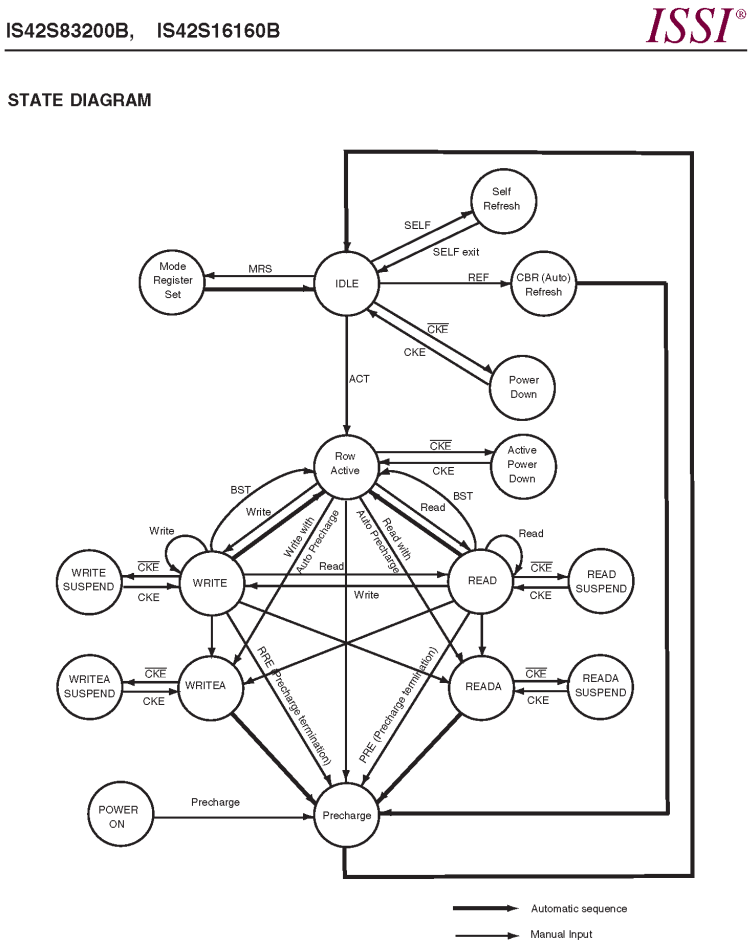
Files to be generated.



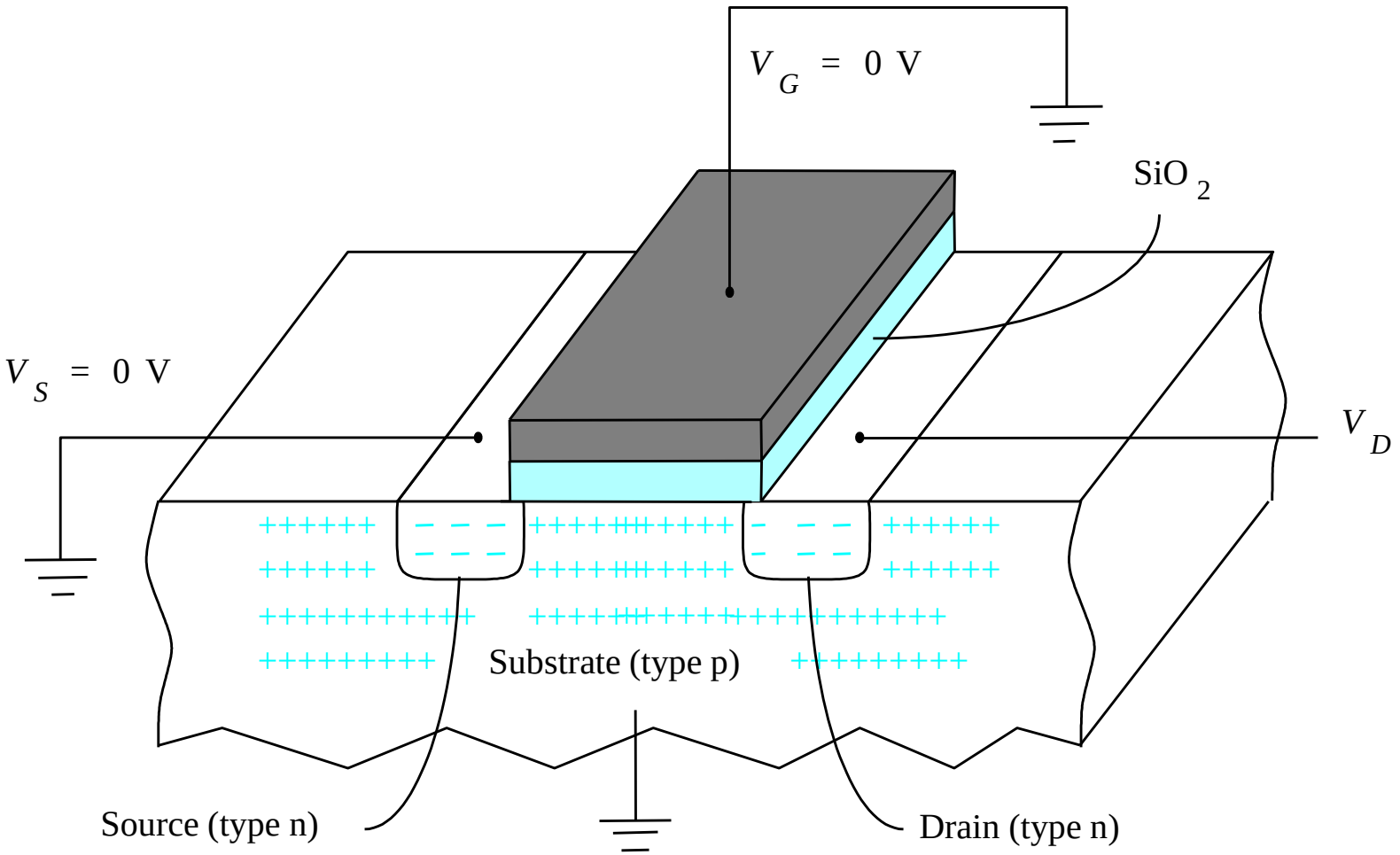
Dynamic RAM

Requires an FSM to refresh periodically.

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

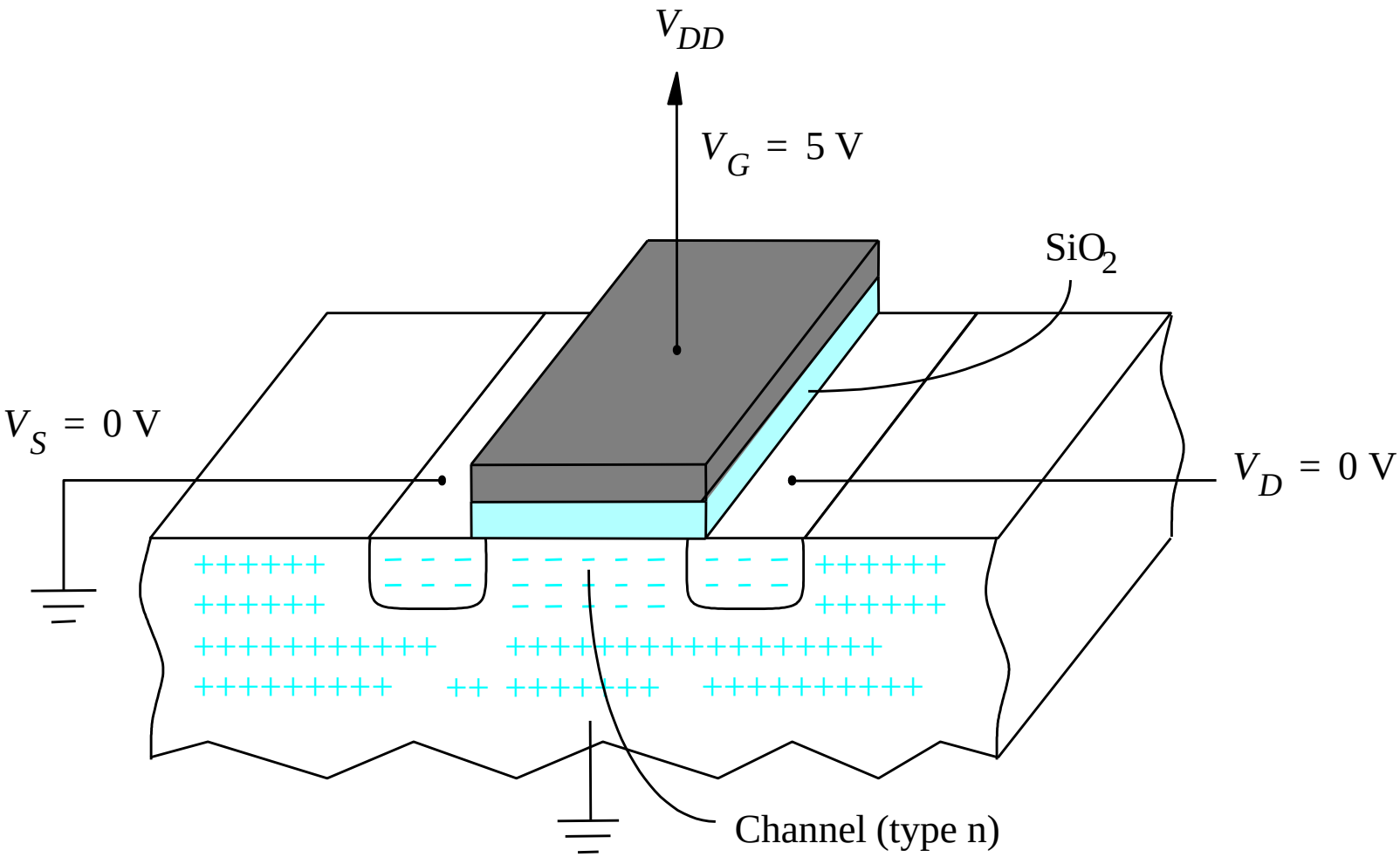


Integrated Silicon Solution, Inc. — www.issi.com — 1-800-379-4774
PRELIMINARY INFORMATION Rev. 00C
12/14/05



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

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



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

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

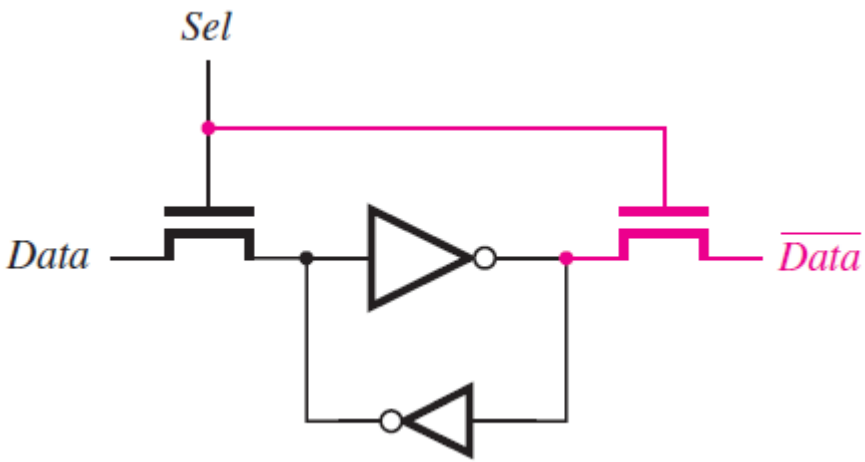


Figure B.64. An SRAM cell.

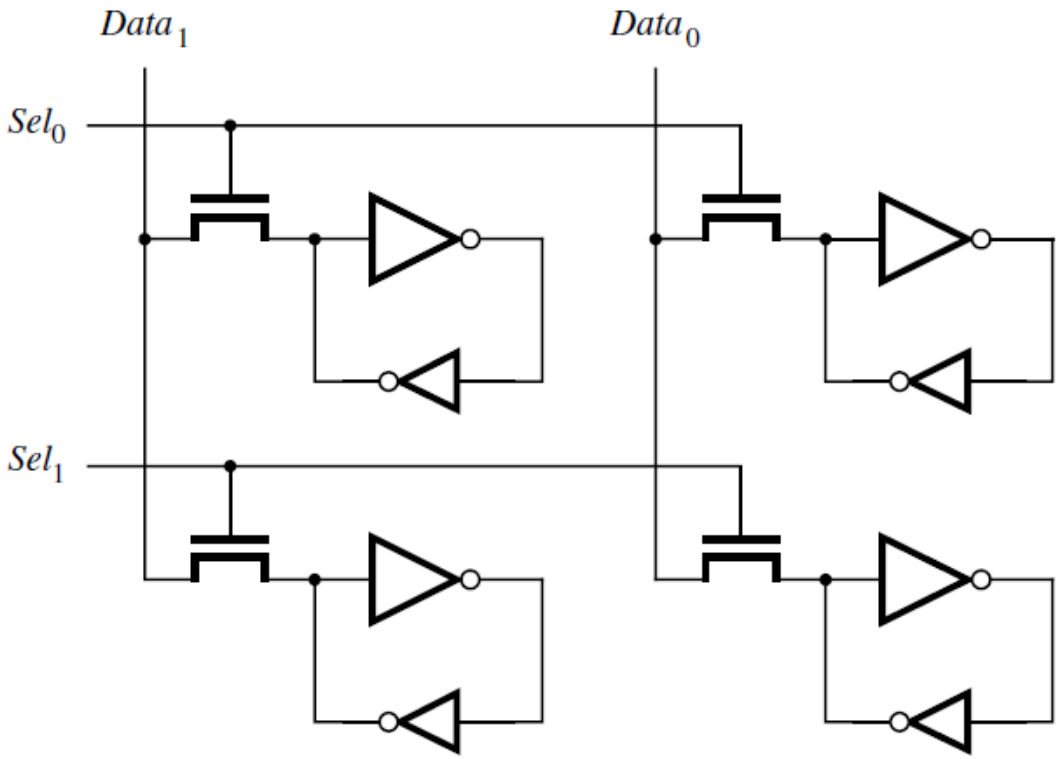


Figure B.65. A 2 x 2 array of SRAM cells.

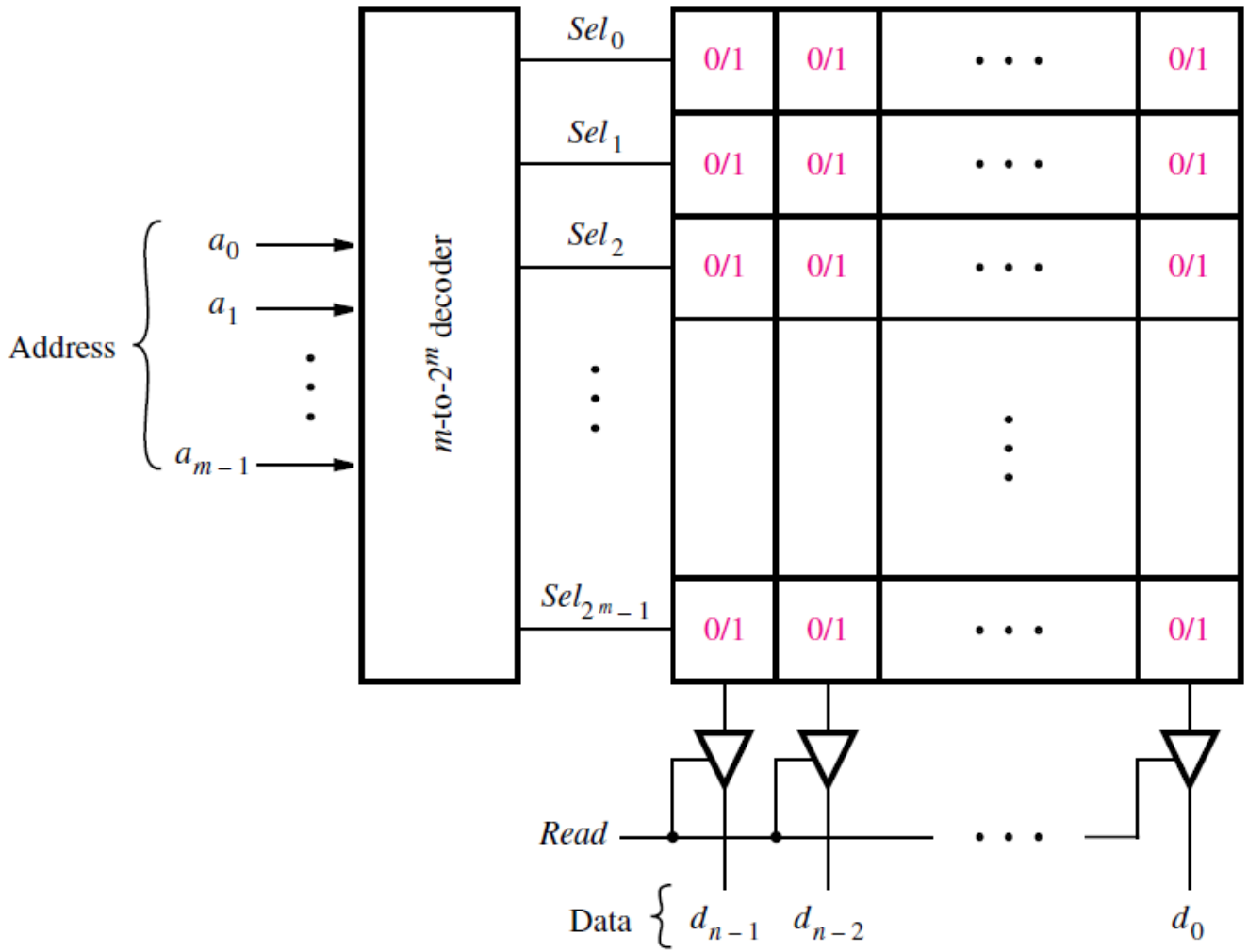


Figure B.72. A 2^m x n read-only memory (ROM) block.

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

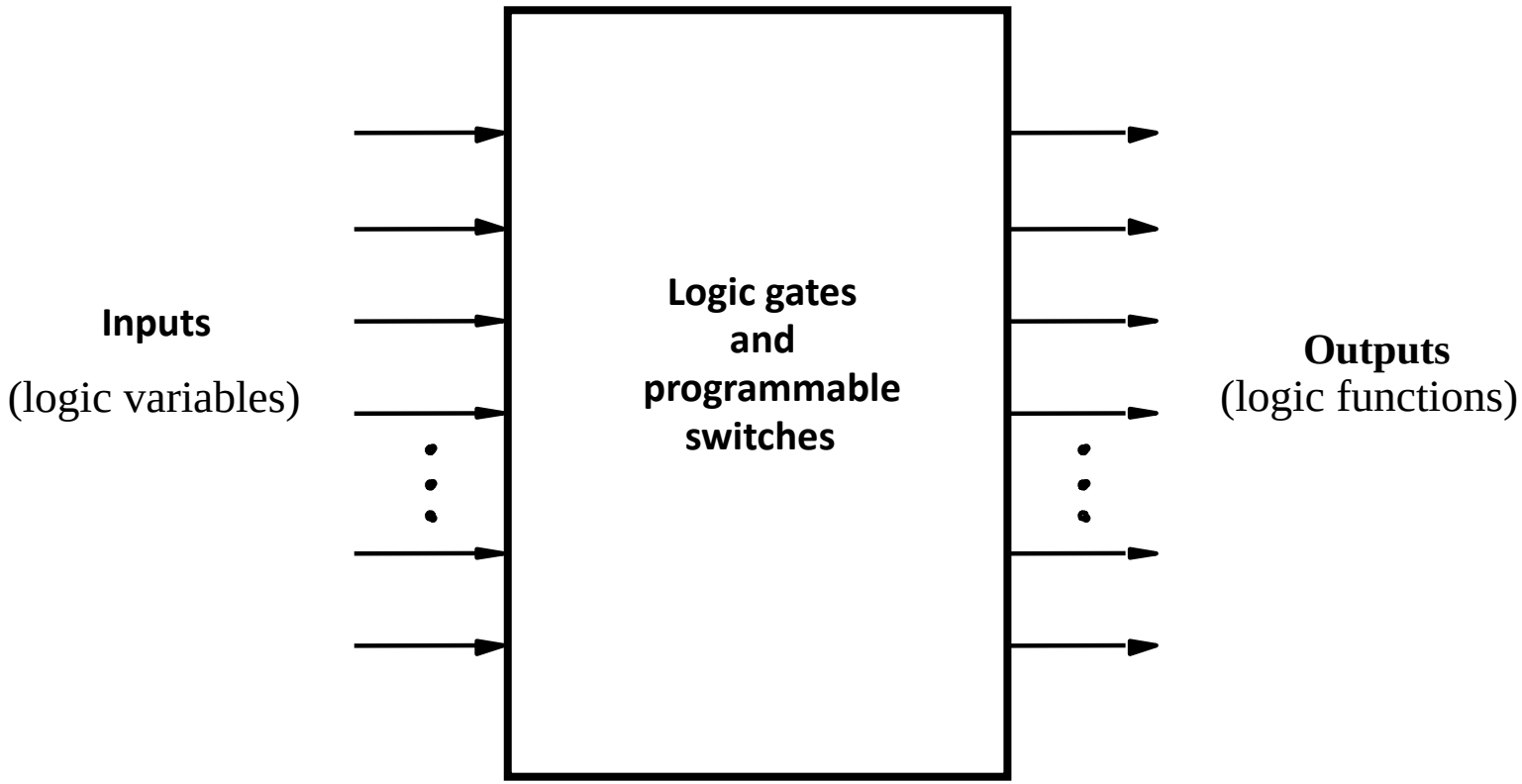


Figure B.24. Programmable logic device as a black box.

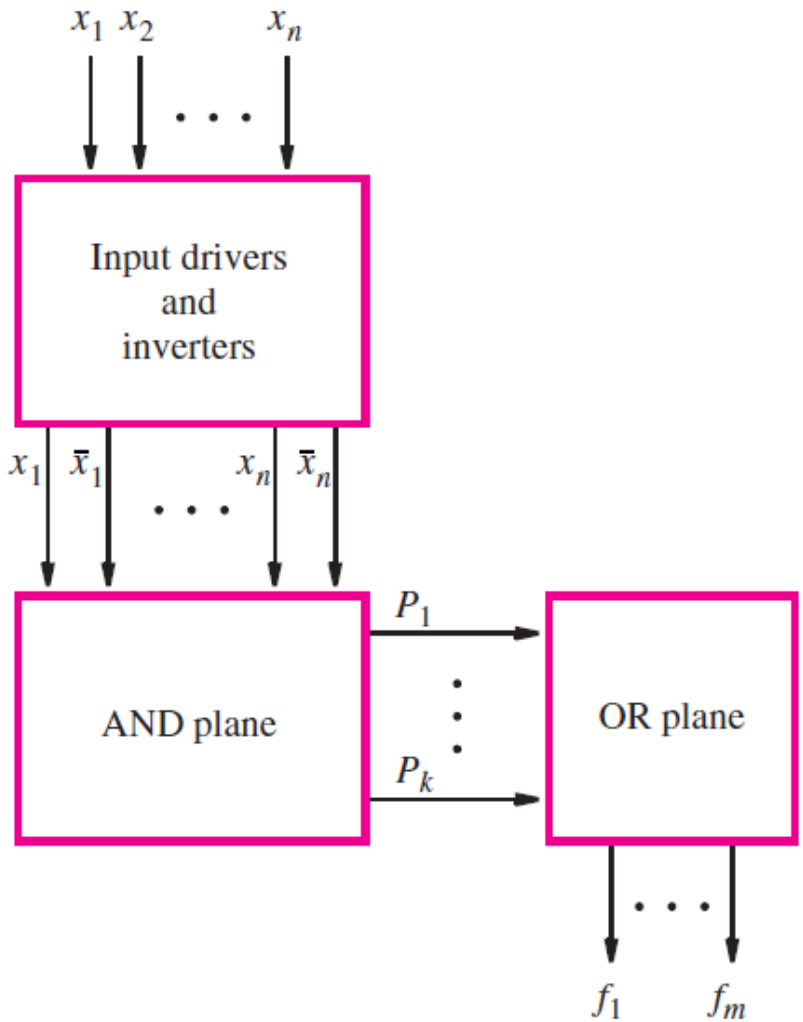


Figure B.25. General structure of a PLA.

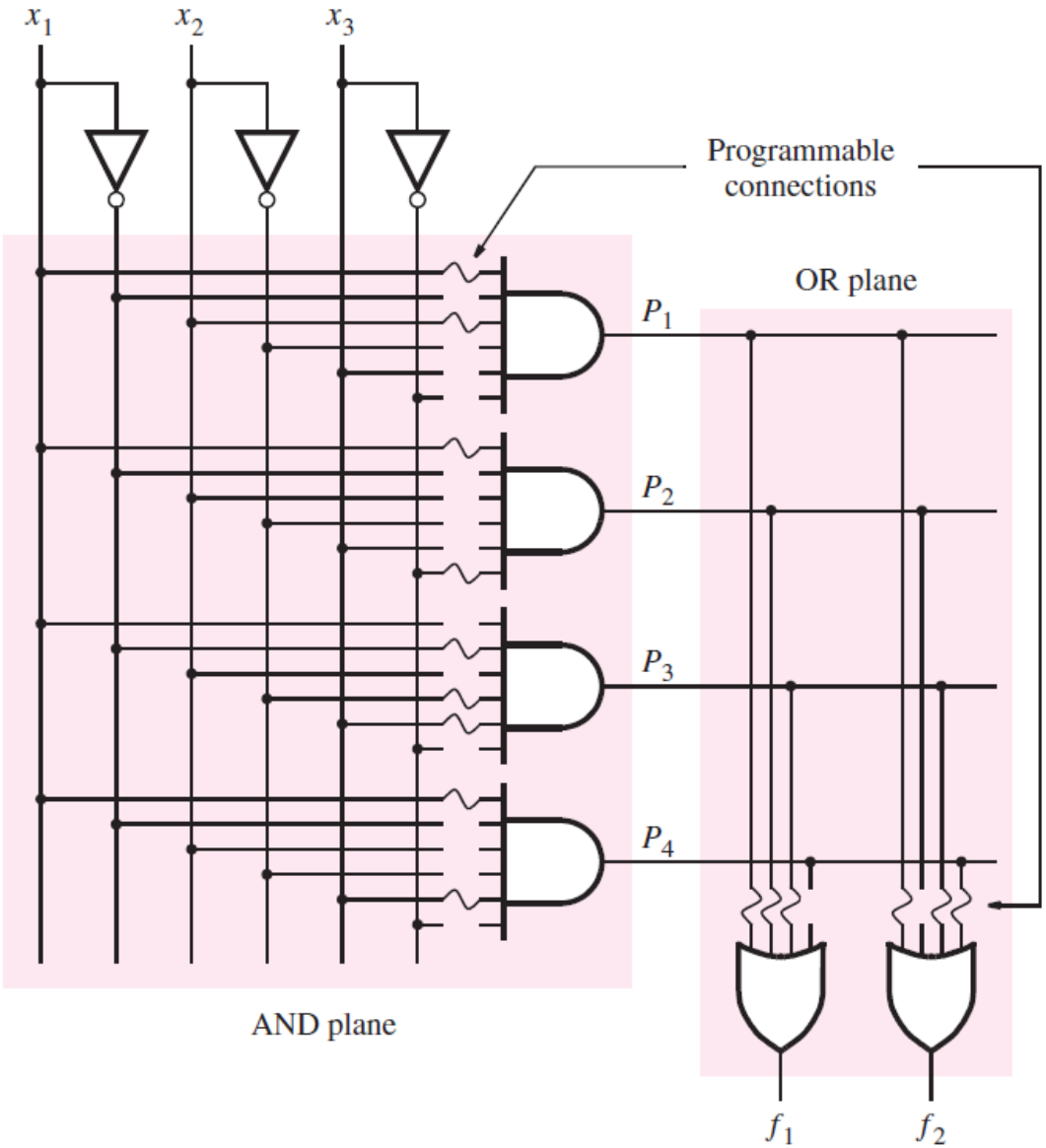


Figure B.26. Gate-level diagram of a PLA.

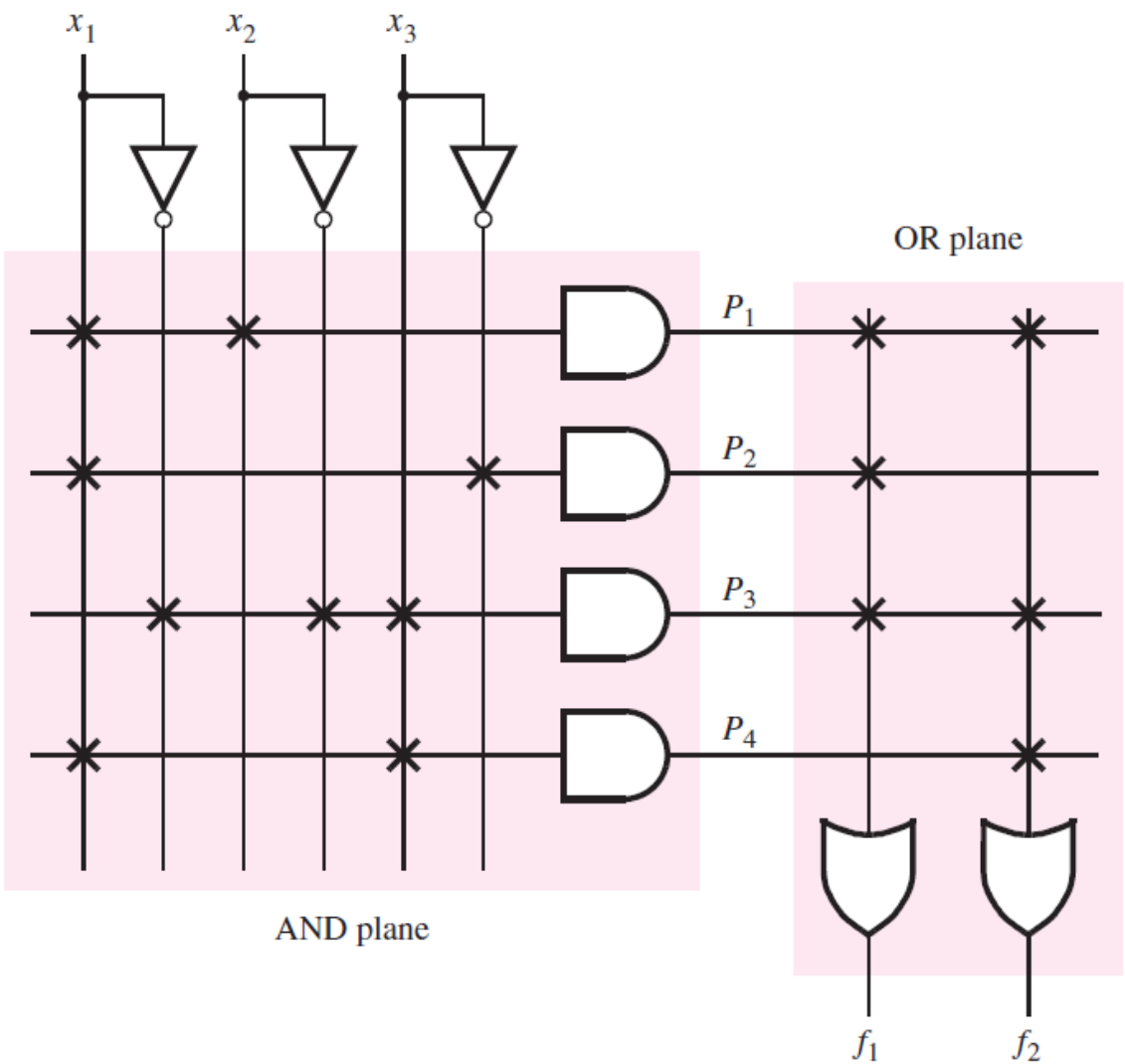


Figure B.27. Customary schematic for the PLA in Figure B.26.

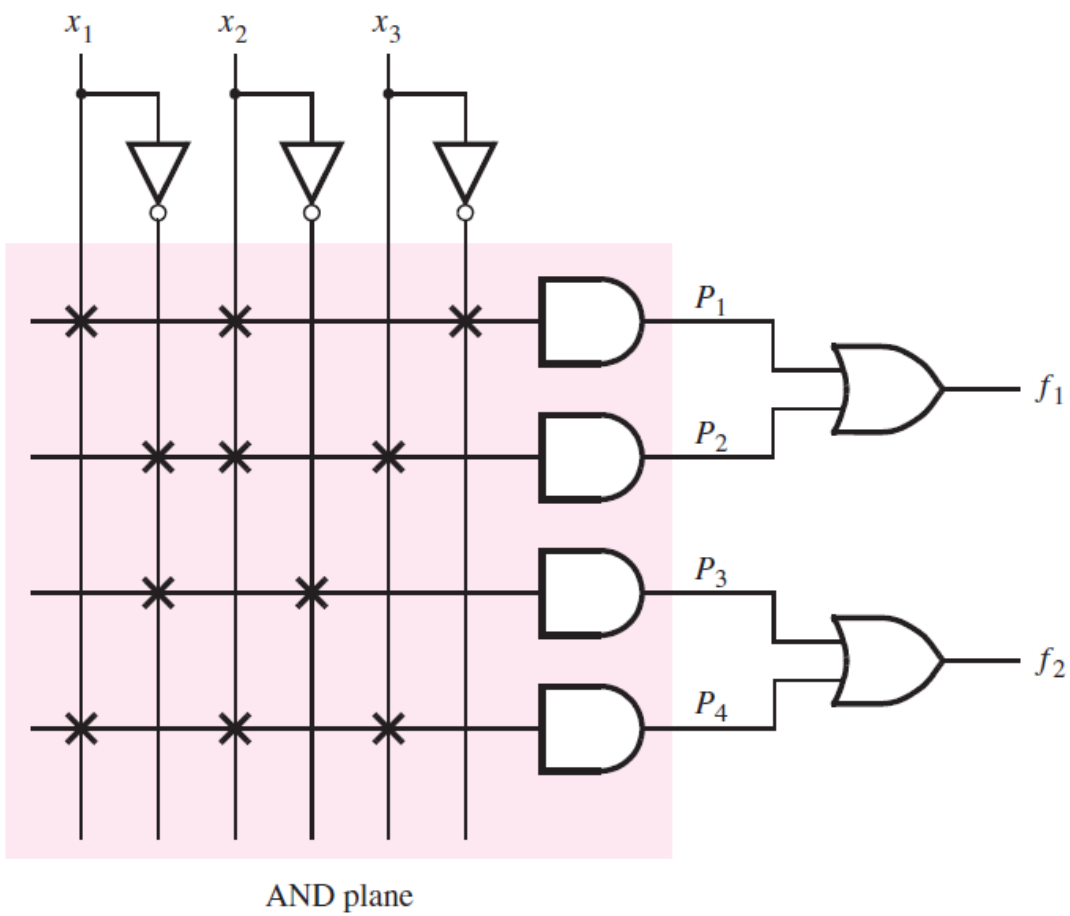


Figure B.28. An example of a PLA.

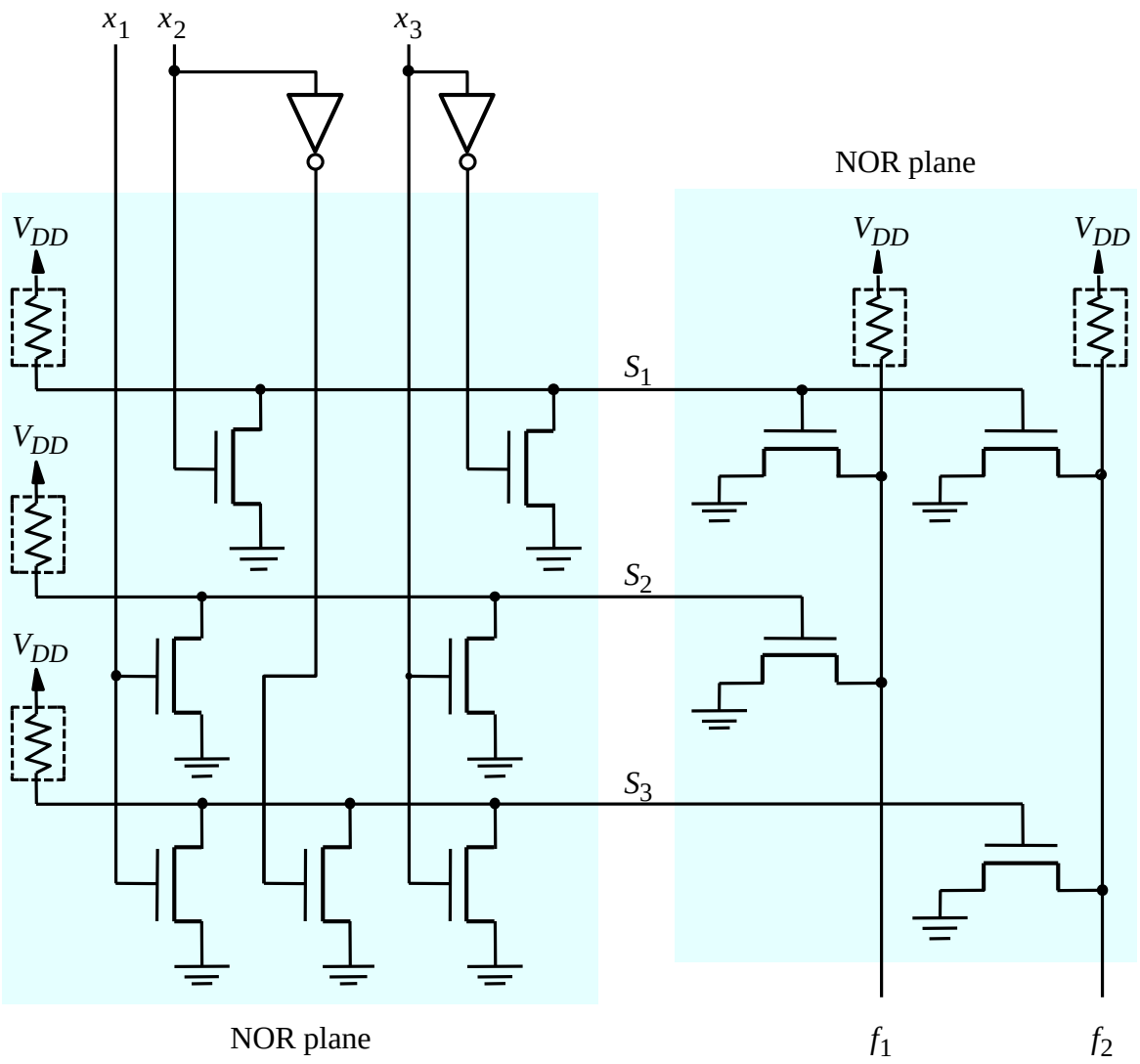
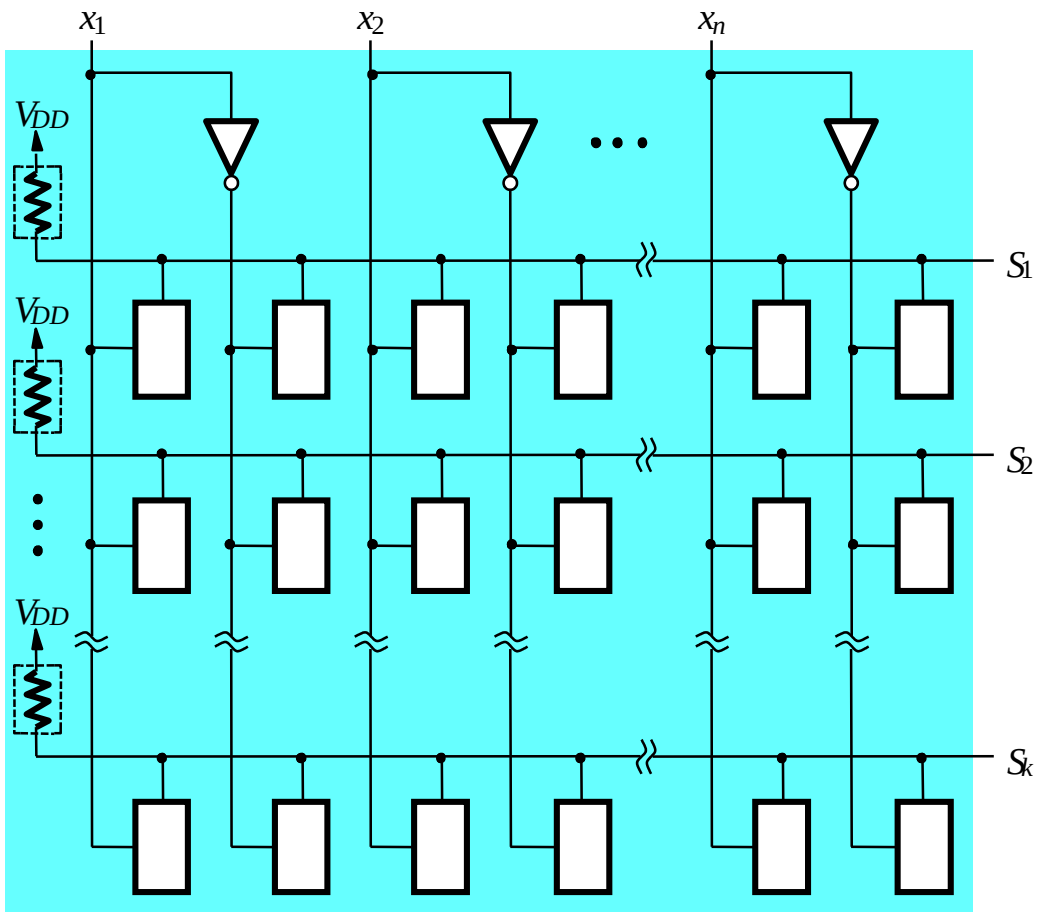
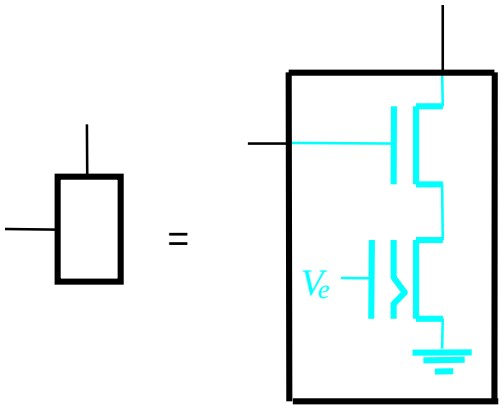


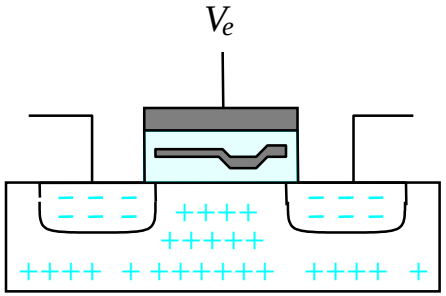
Figure B.67. An example of a NOR-NOR PLA.



(a) Programmable NOR-plane



(b) A programmable switch



(c) EEPROM transistor

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

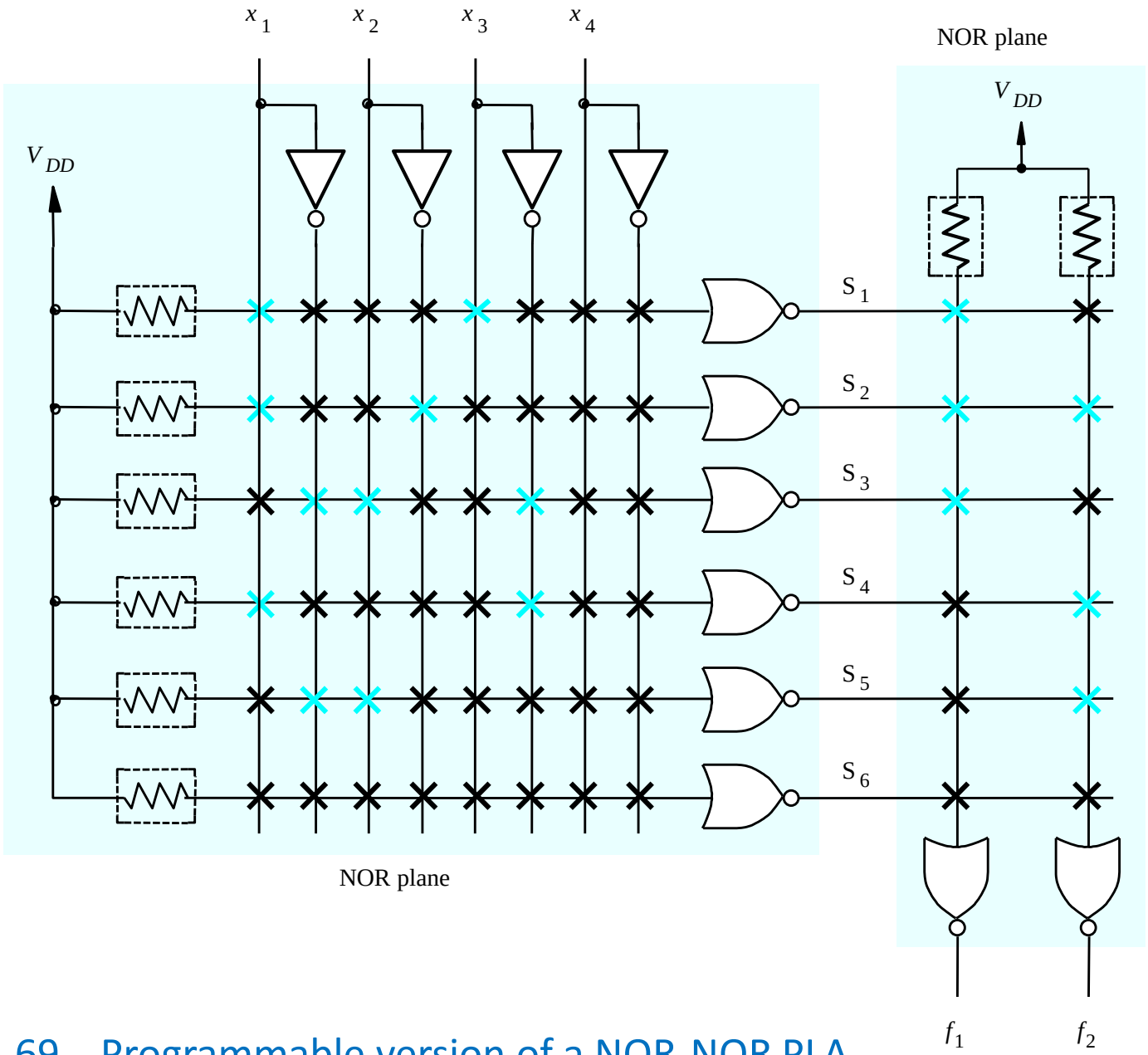


Figure B.69. Programmable version of a NOR-NOR PLA.

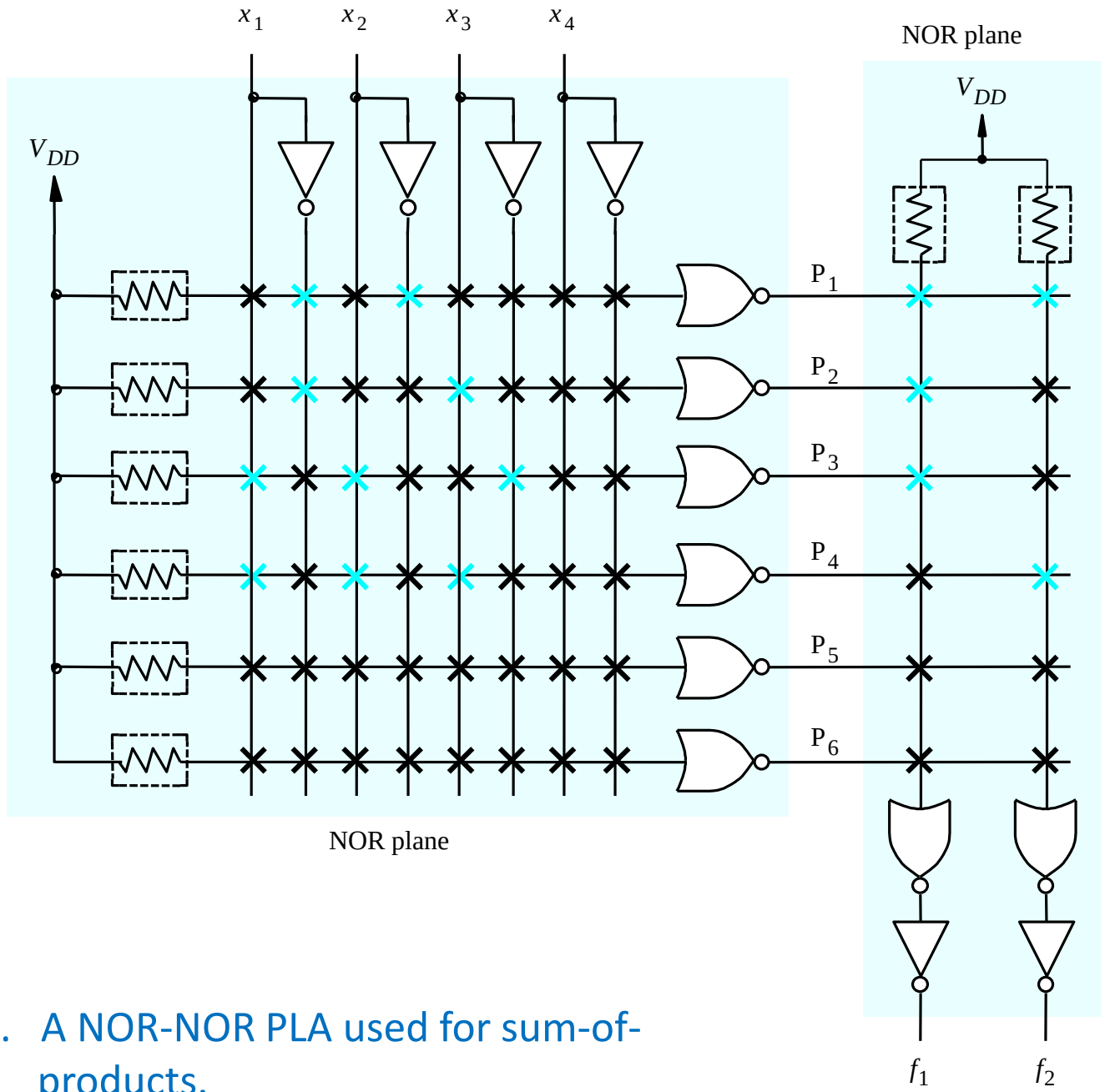


Figure B.70. A NOR-NOR PLA used for sum-of-products.

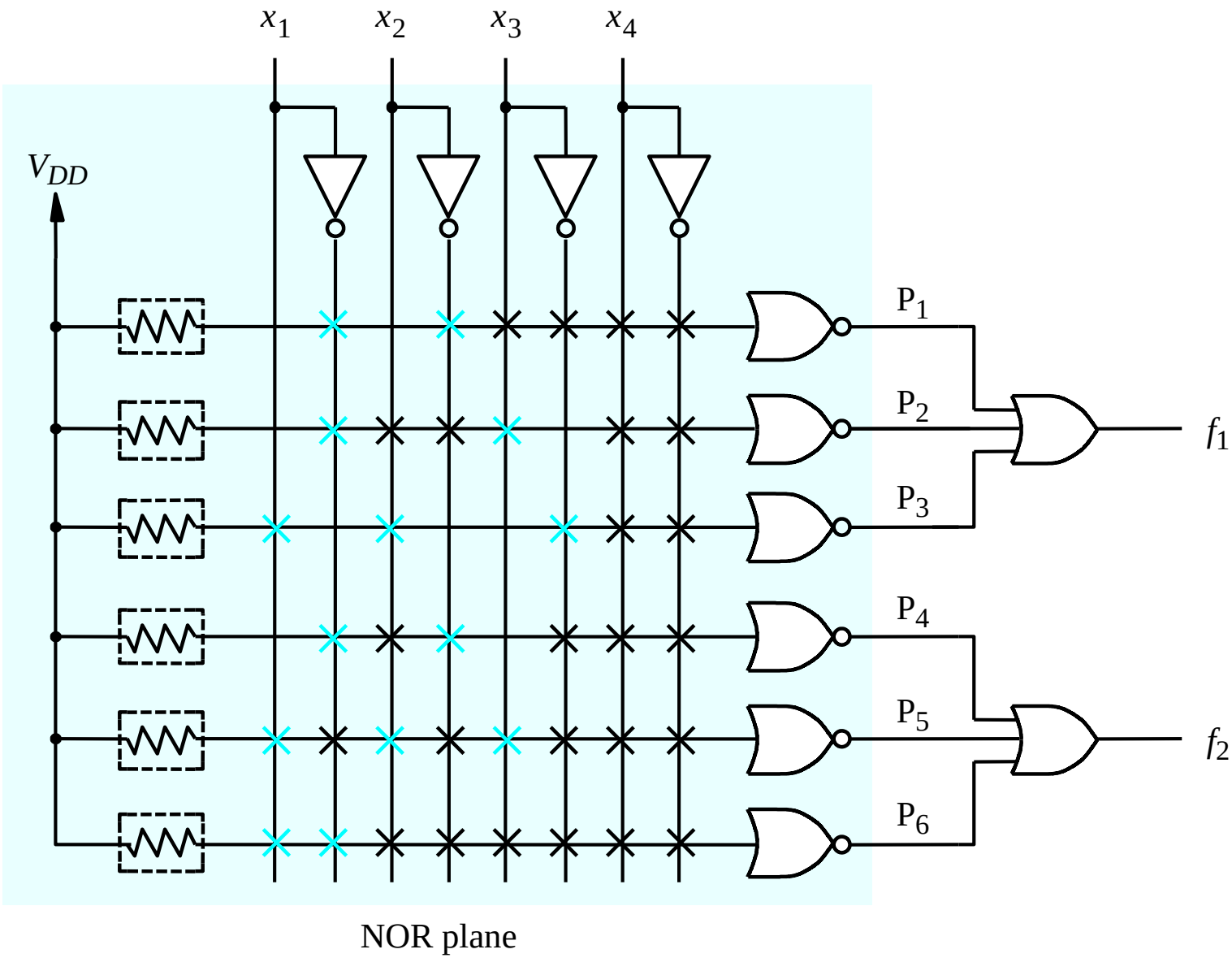


Figure B.71. PAL programmed to implement two functions in Figure B.70.

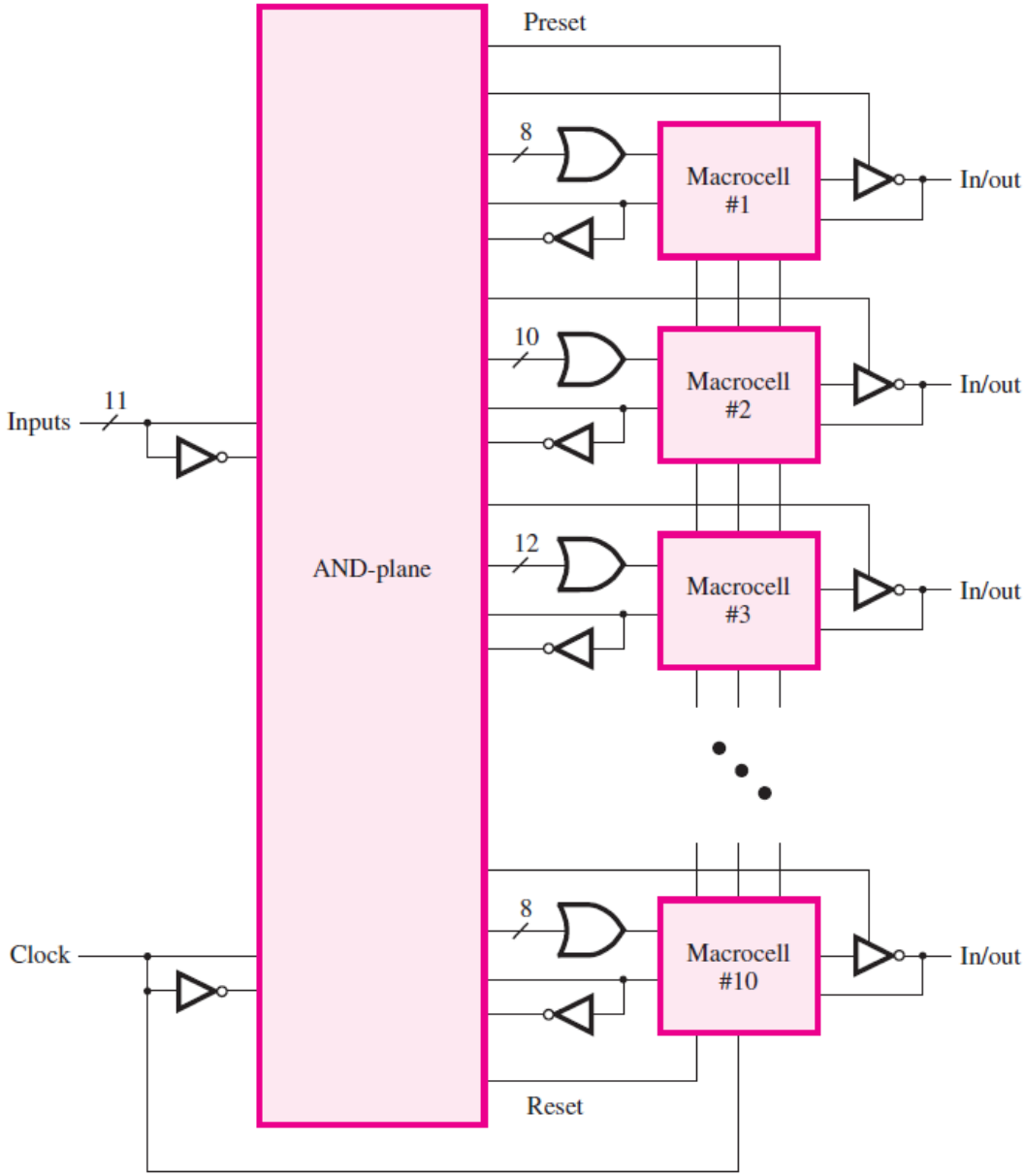


Figure B.29. The 22V10 PAL device.

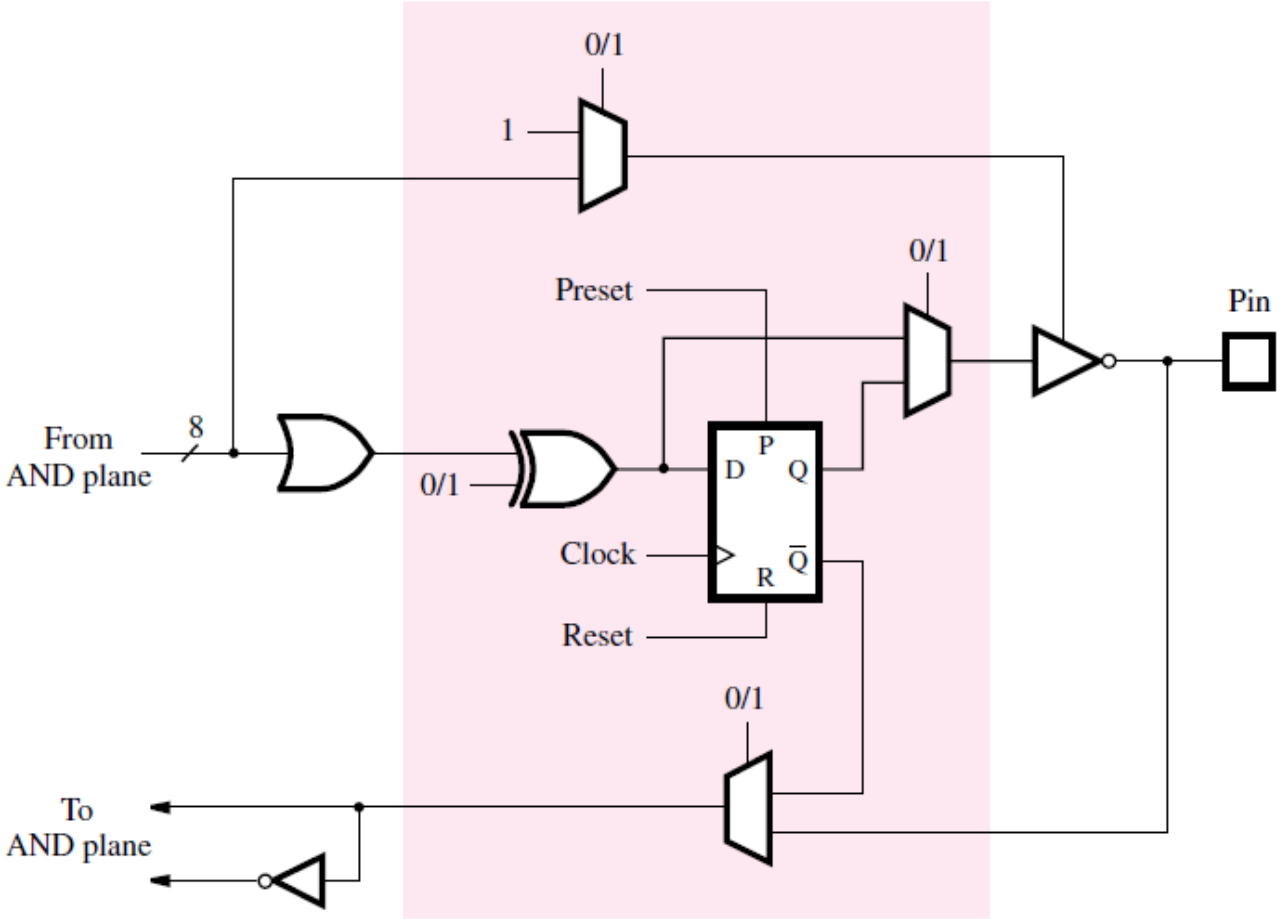


Figure B.30. The 22V10 macrocell.

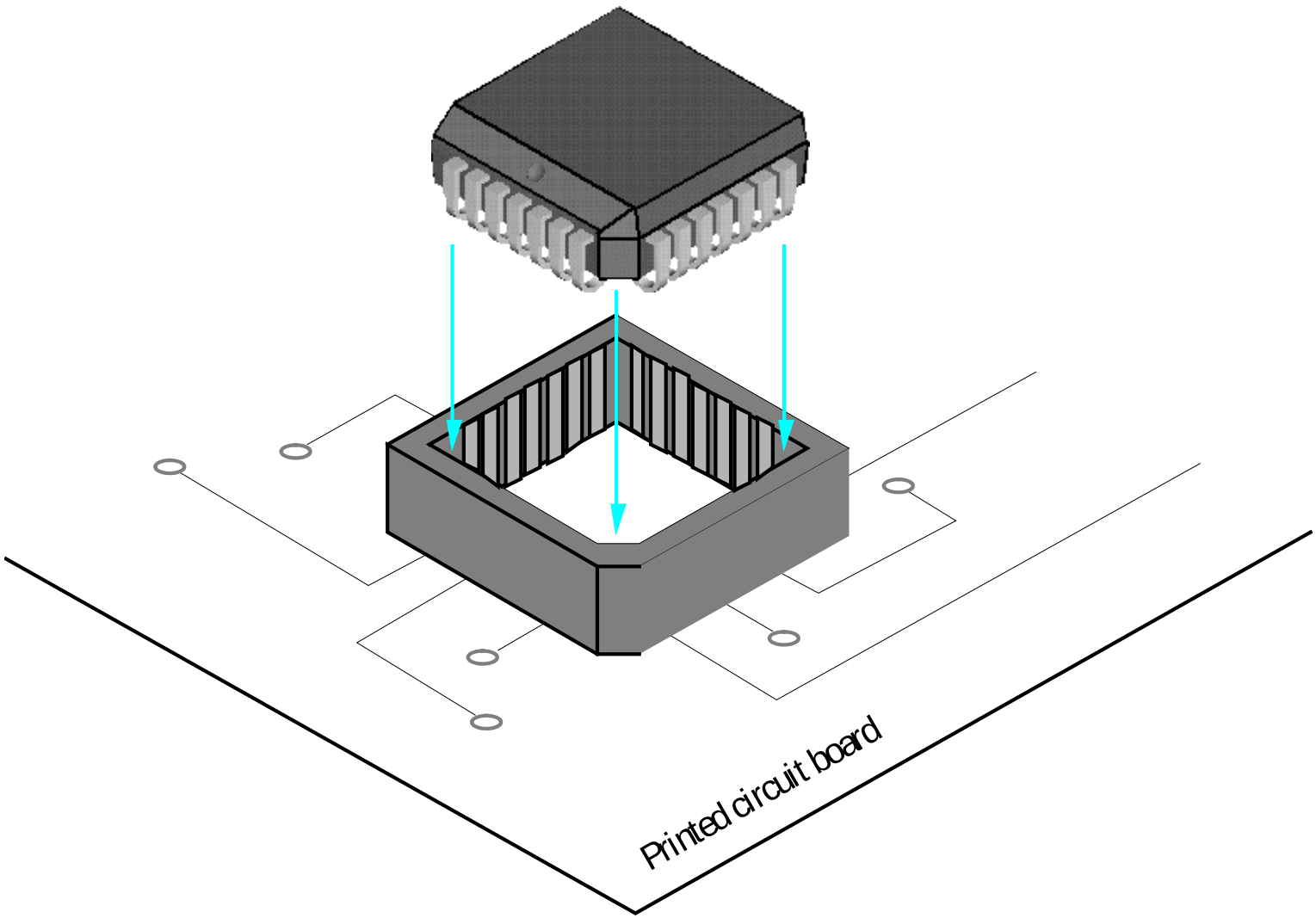


Figure B.31. A PLCC package with socket.

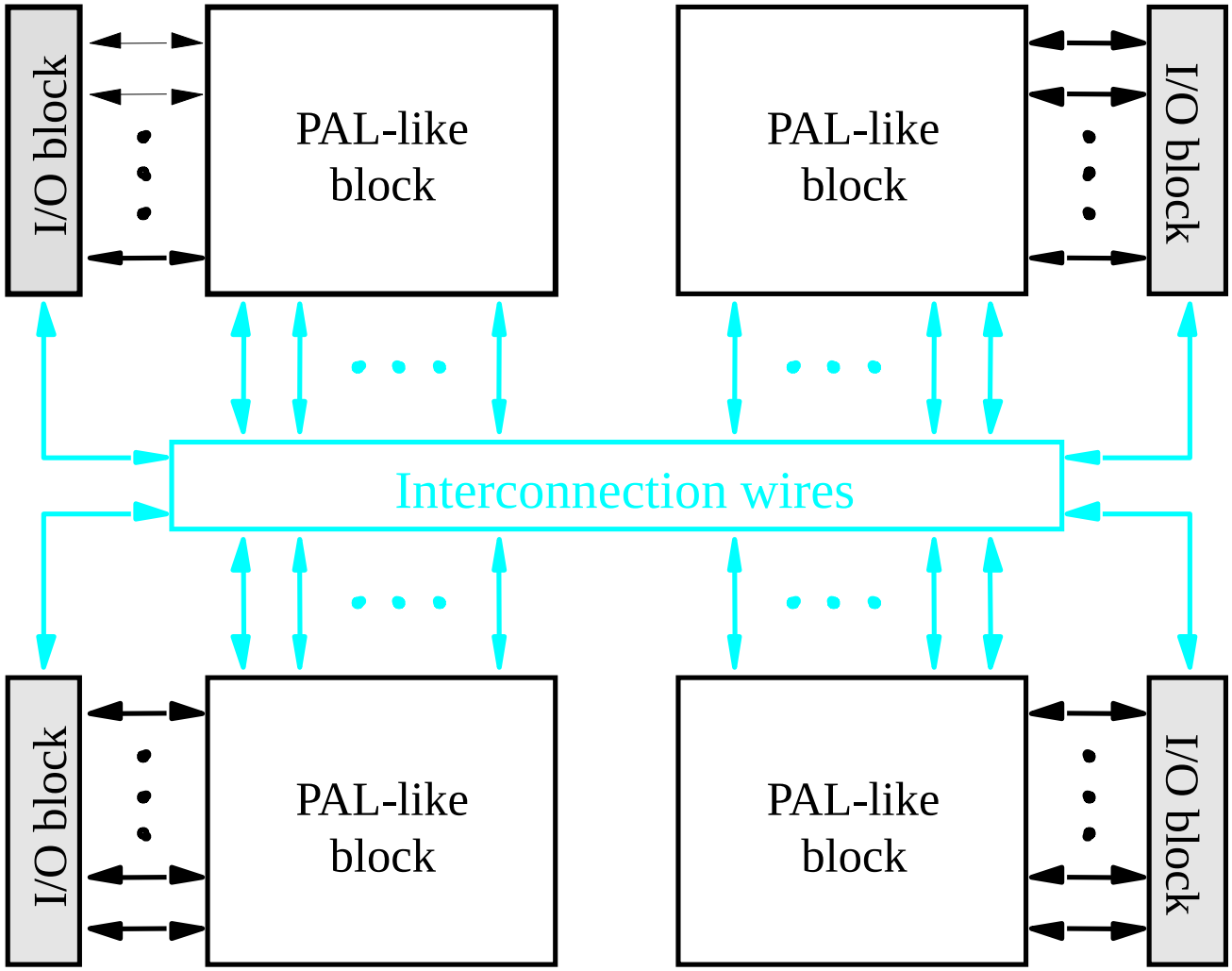


Figure B.32. Structure of a complex programmable logic device (CPLD).

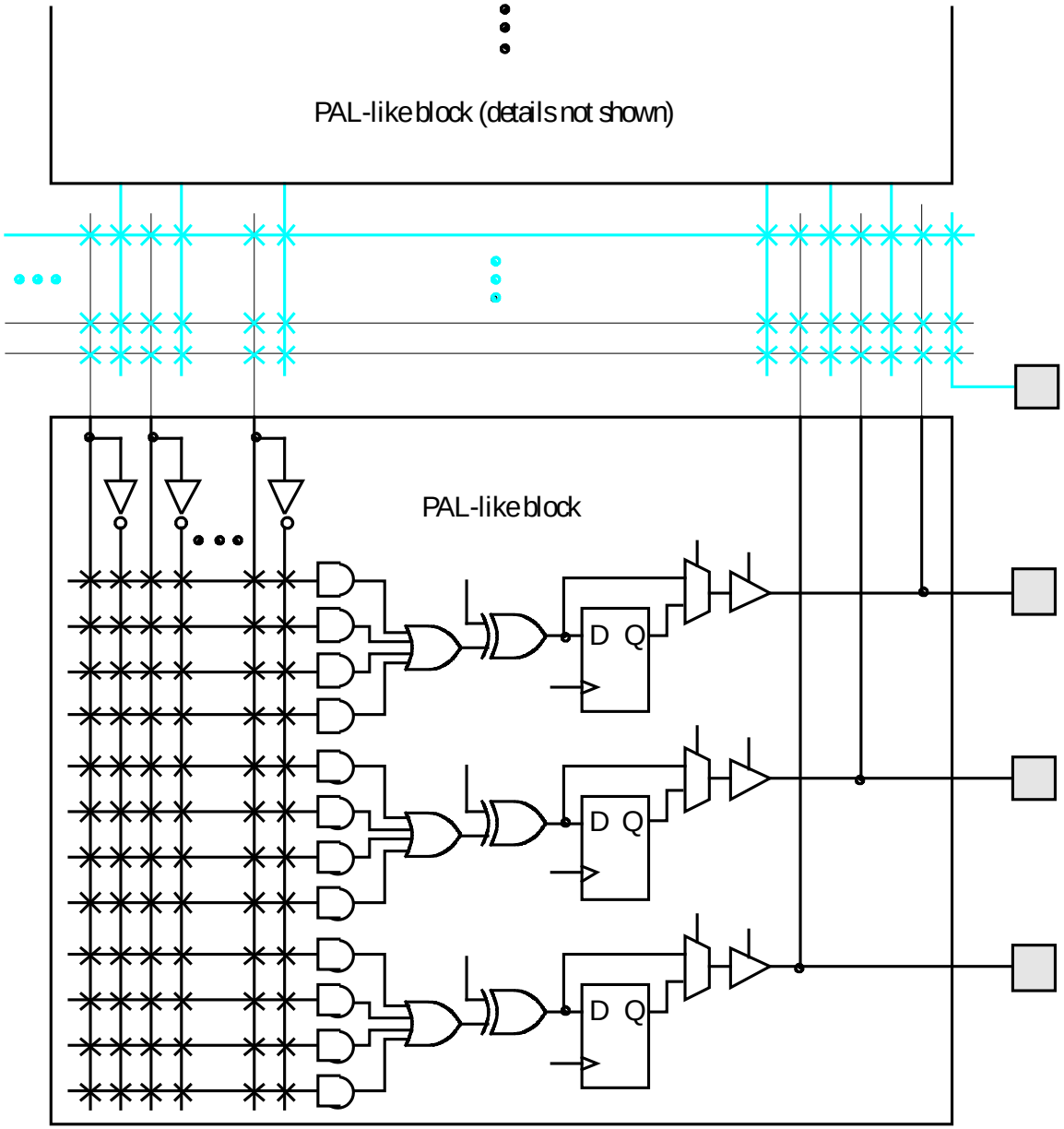
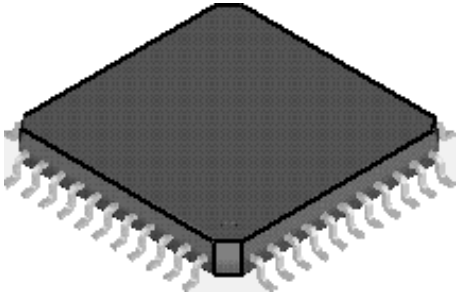
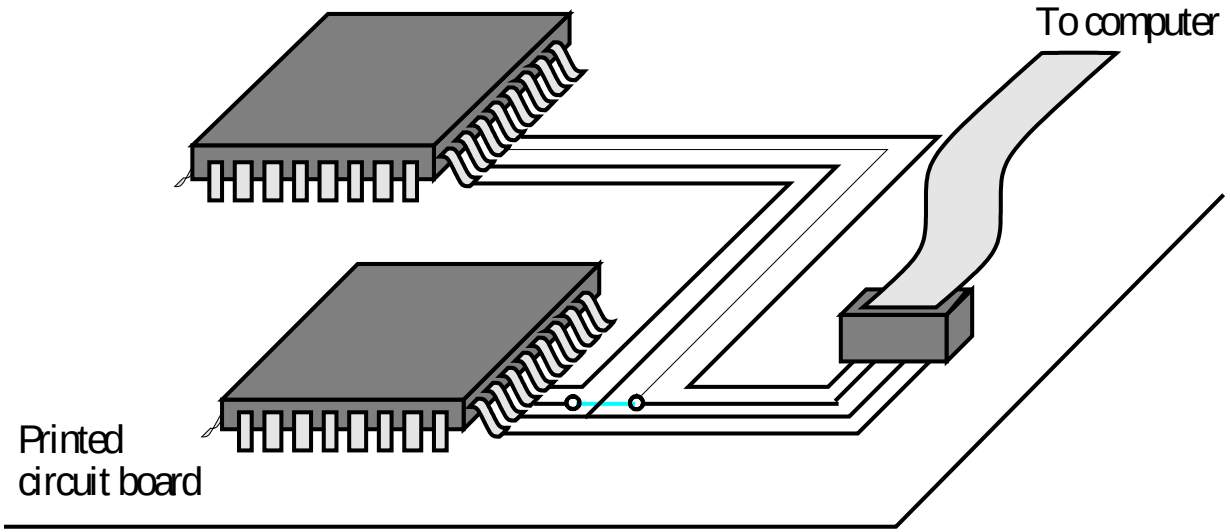


Figure B.33. A section of the CPLD in Figure B.32.



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



(b) JTAG programming

Figure B.34. CPLD packaging and programming.

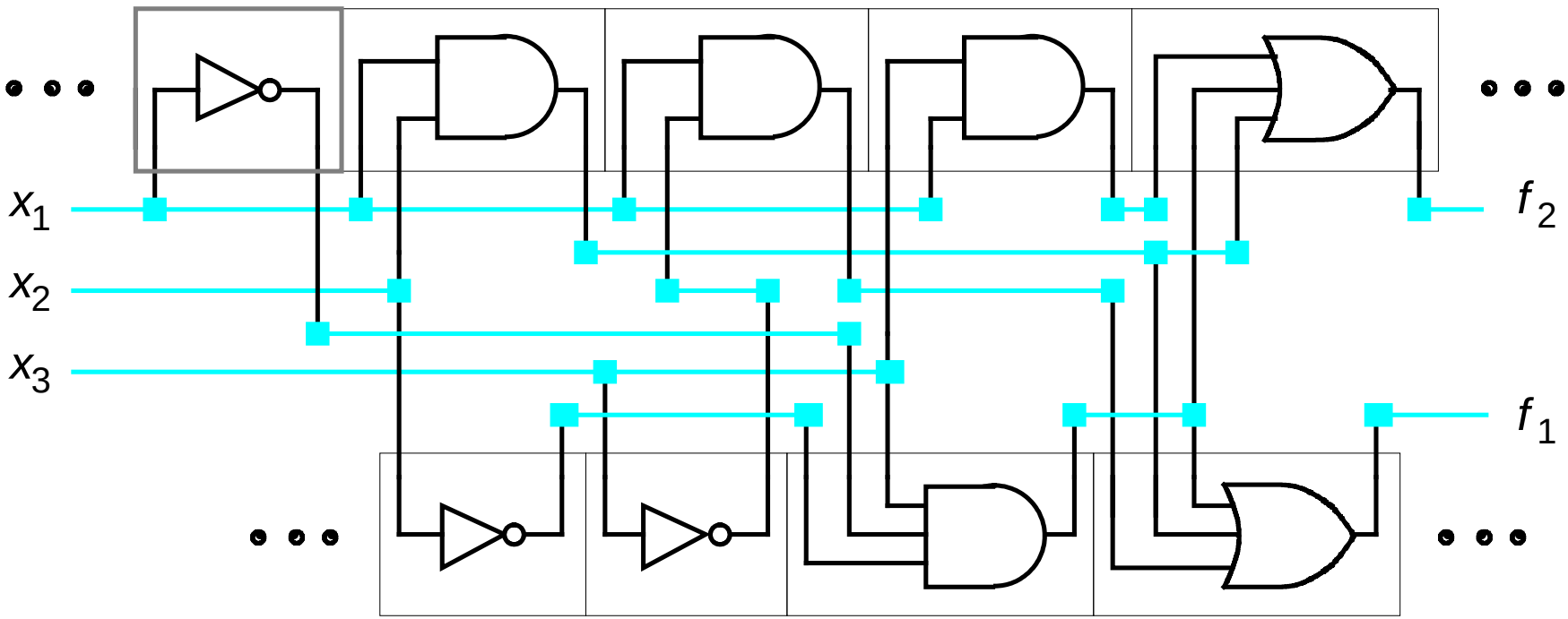


Figure B.40. A section of two rows in a standard-cell chip.

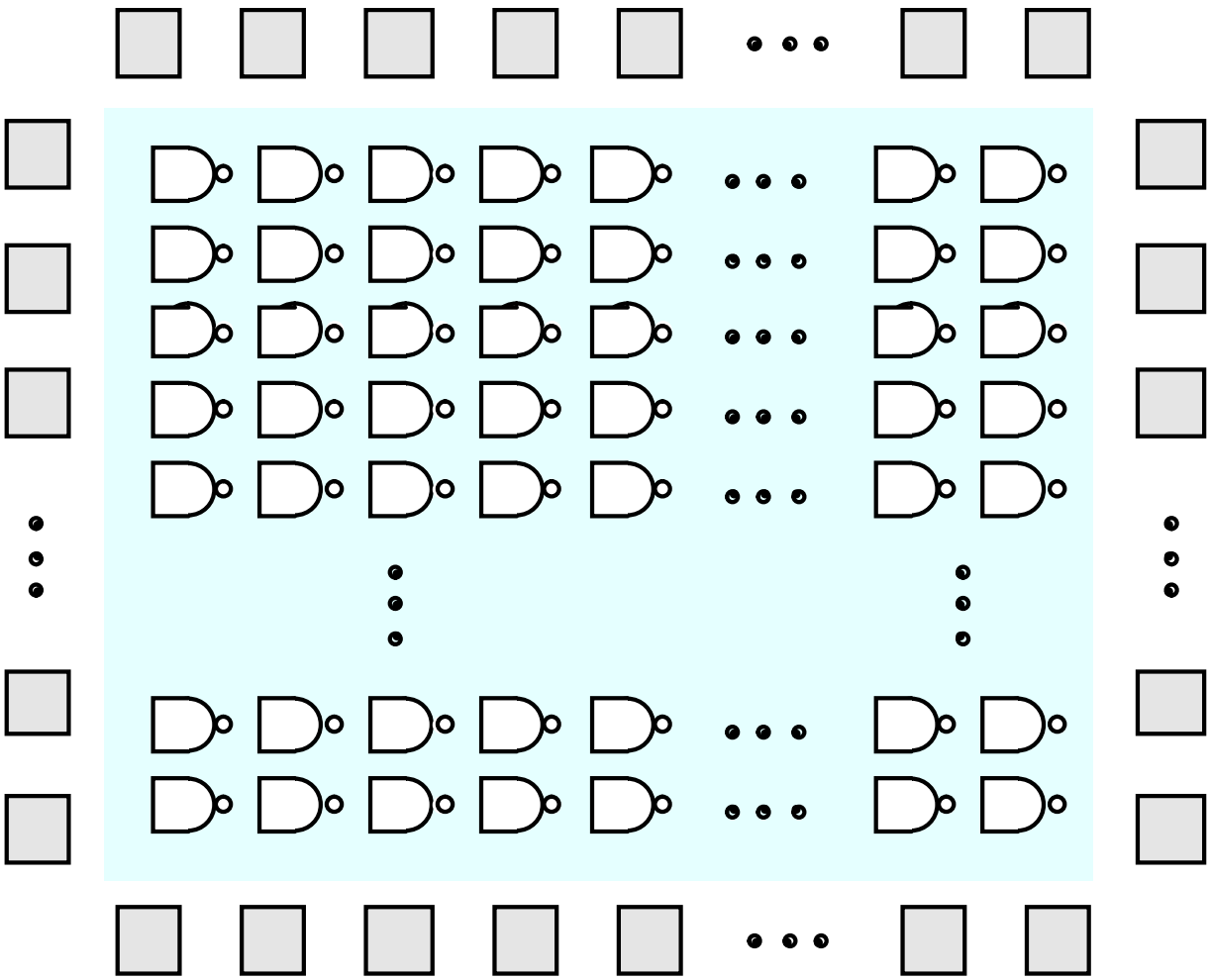


Figure B.41. A sea-of-gates gate array.

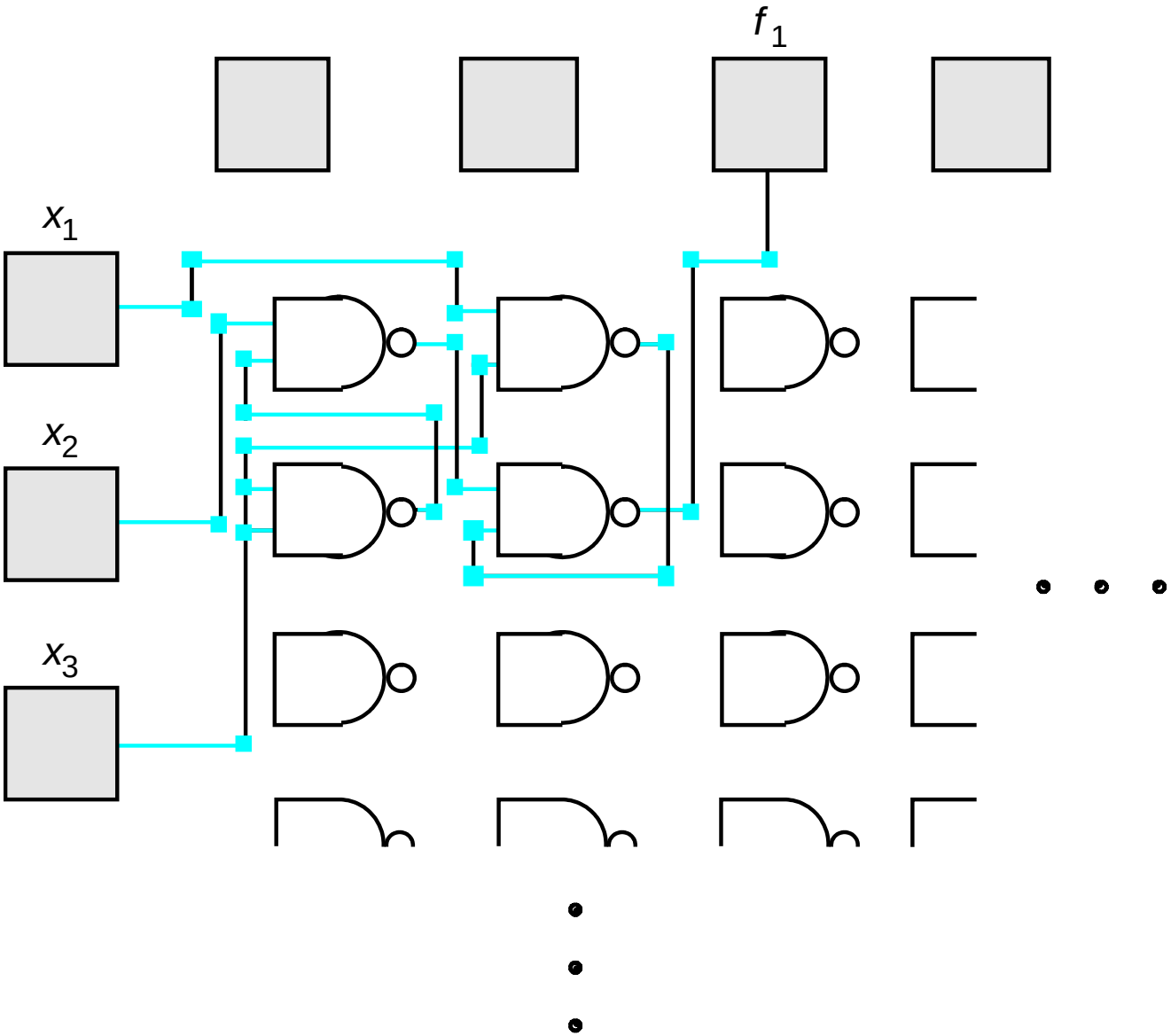
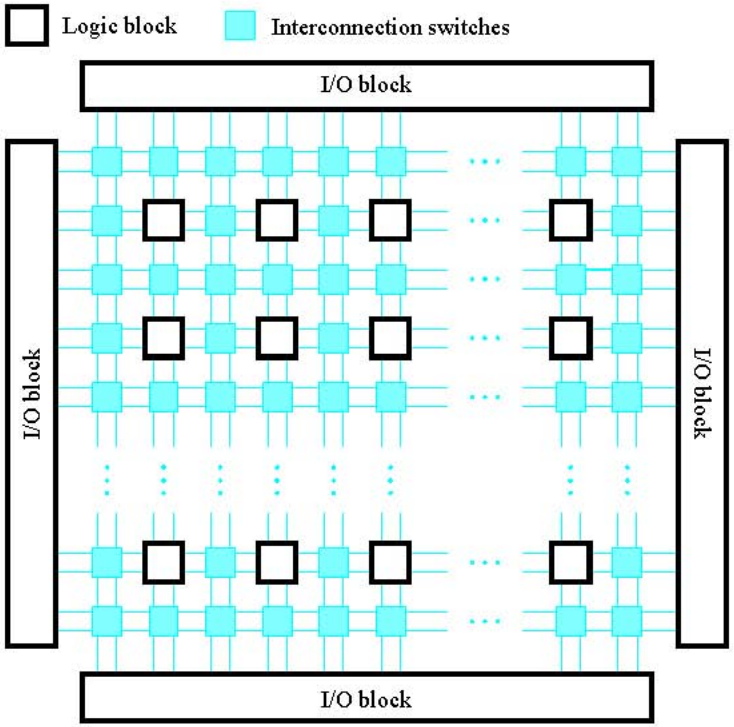
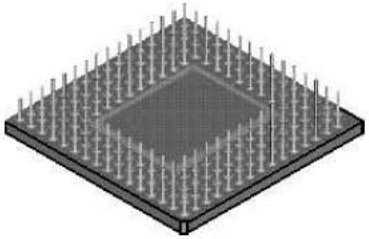


Figure B.42. The logic function $f_1 = x_2\bar{x}_3 + x_1x_3$ in the gate array of Figure B.41.

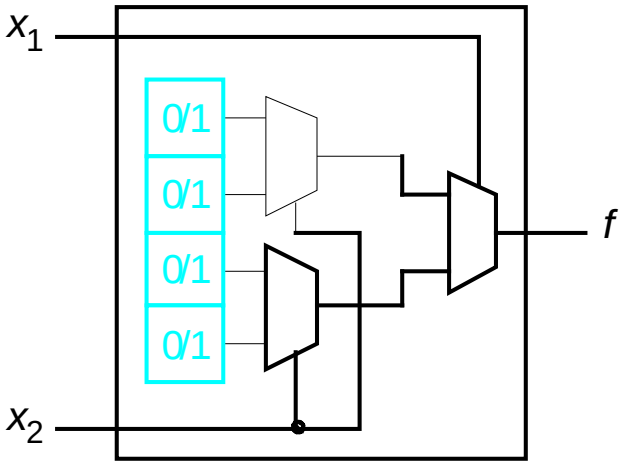


(a) General structure of an FPGA



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

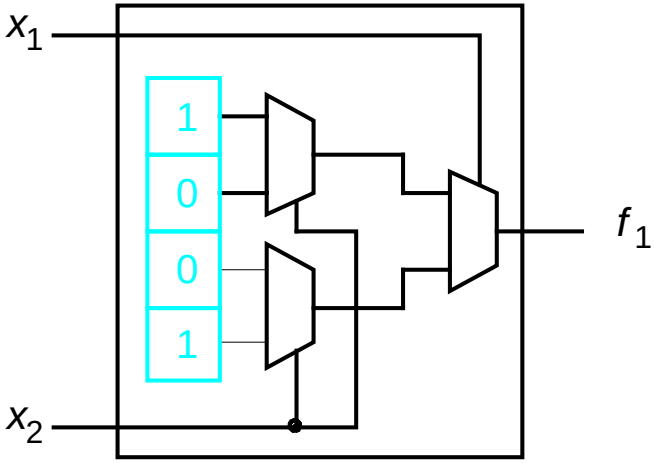
Figure B.35. A field-programmable gate array (FPGA).



(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).

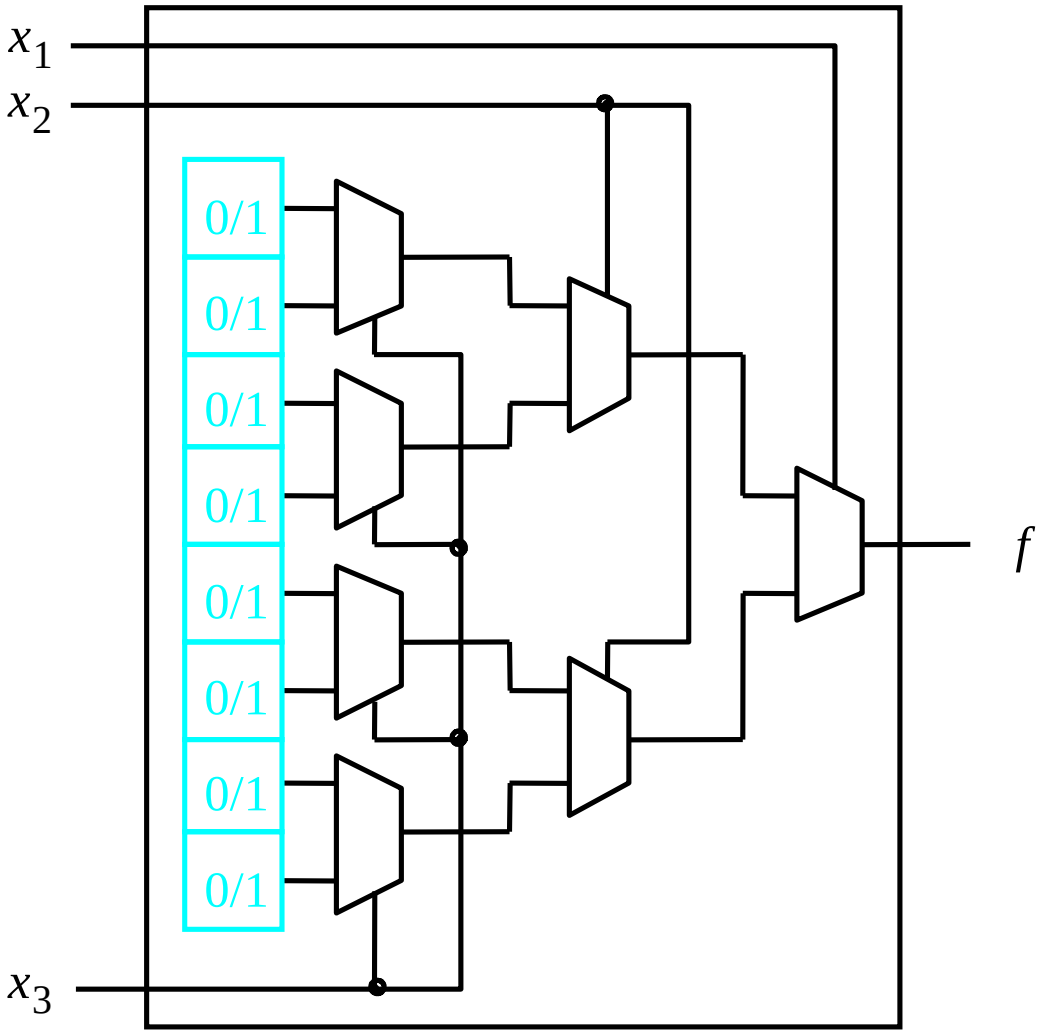


Figure B.37. A three-input LUT.

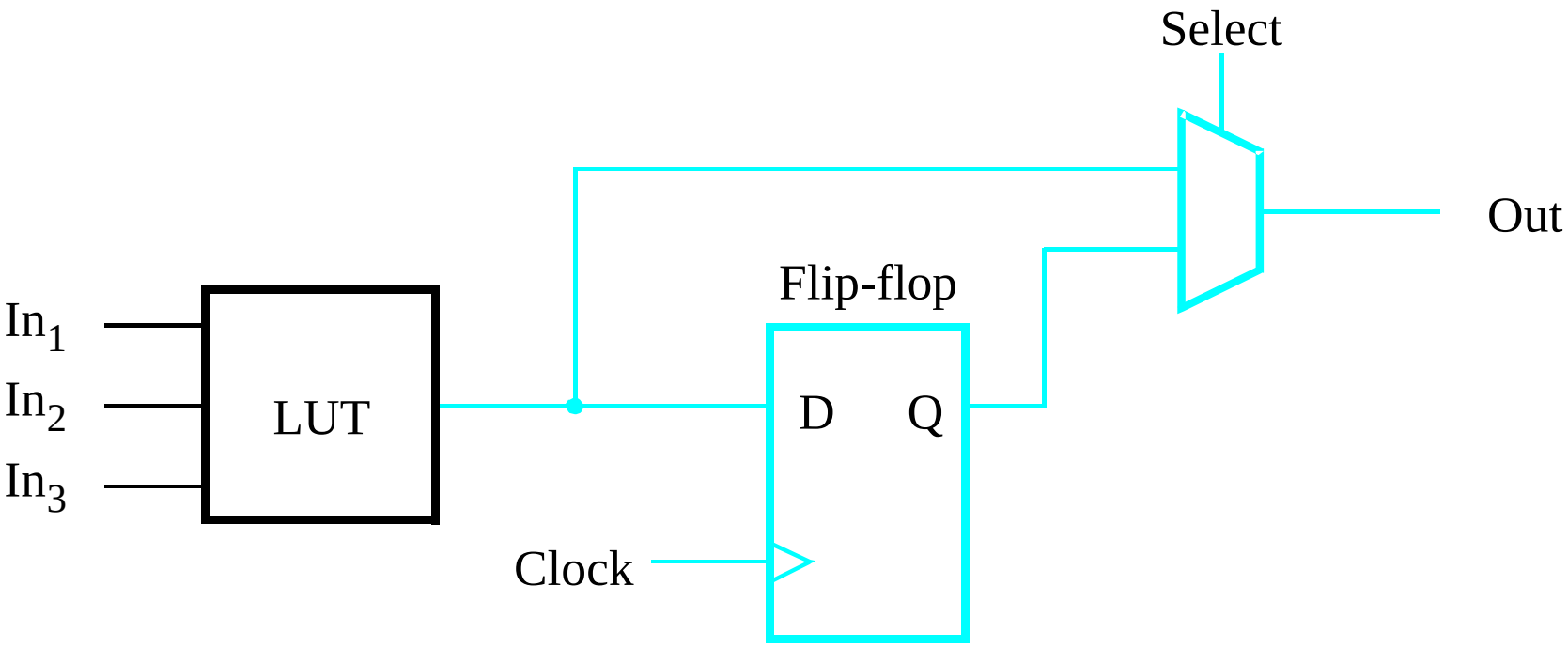
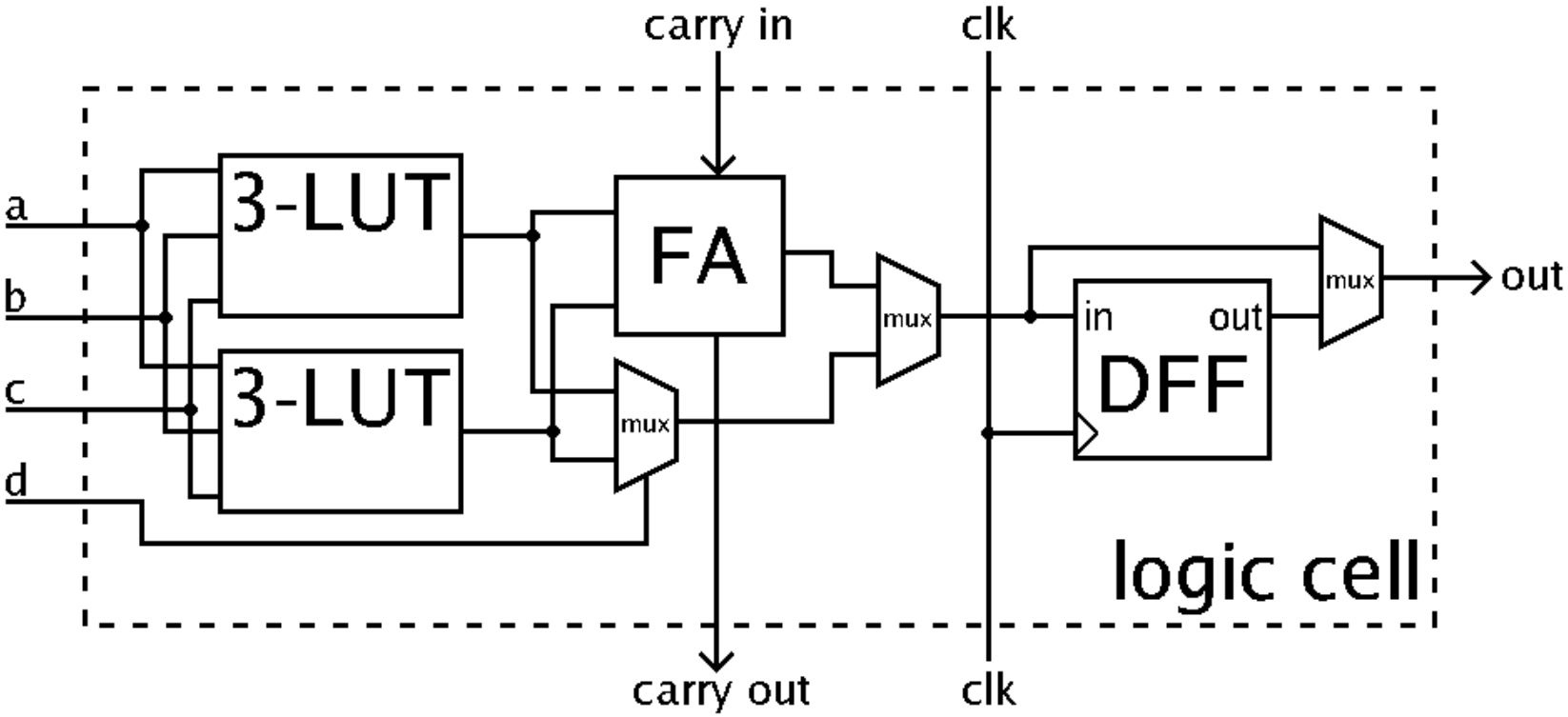


Figure B.38. Inclusion of a flip-flop in an FPGA logic block.



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

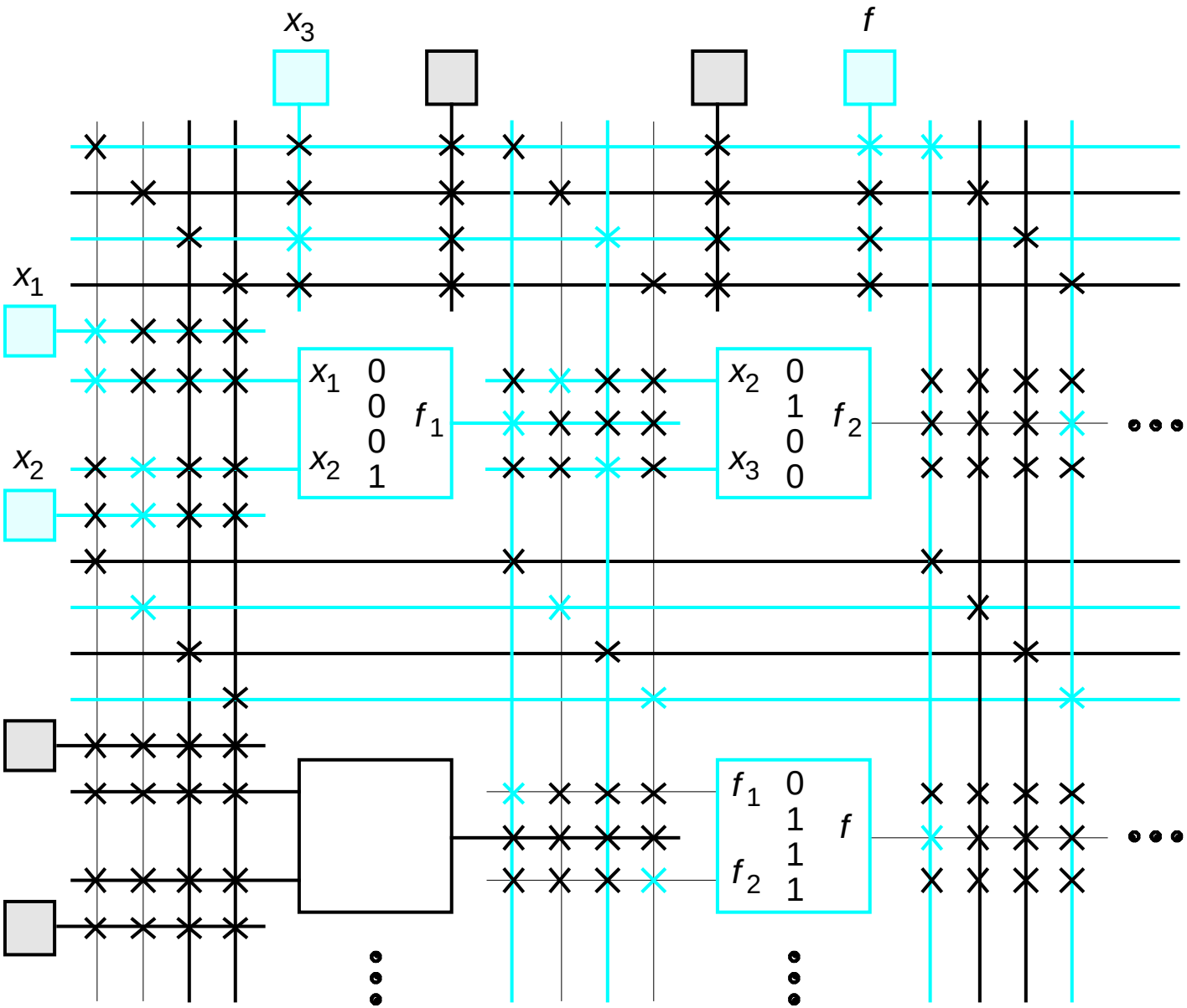


Figure B.39. A section of a programmed FPGA.

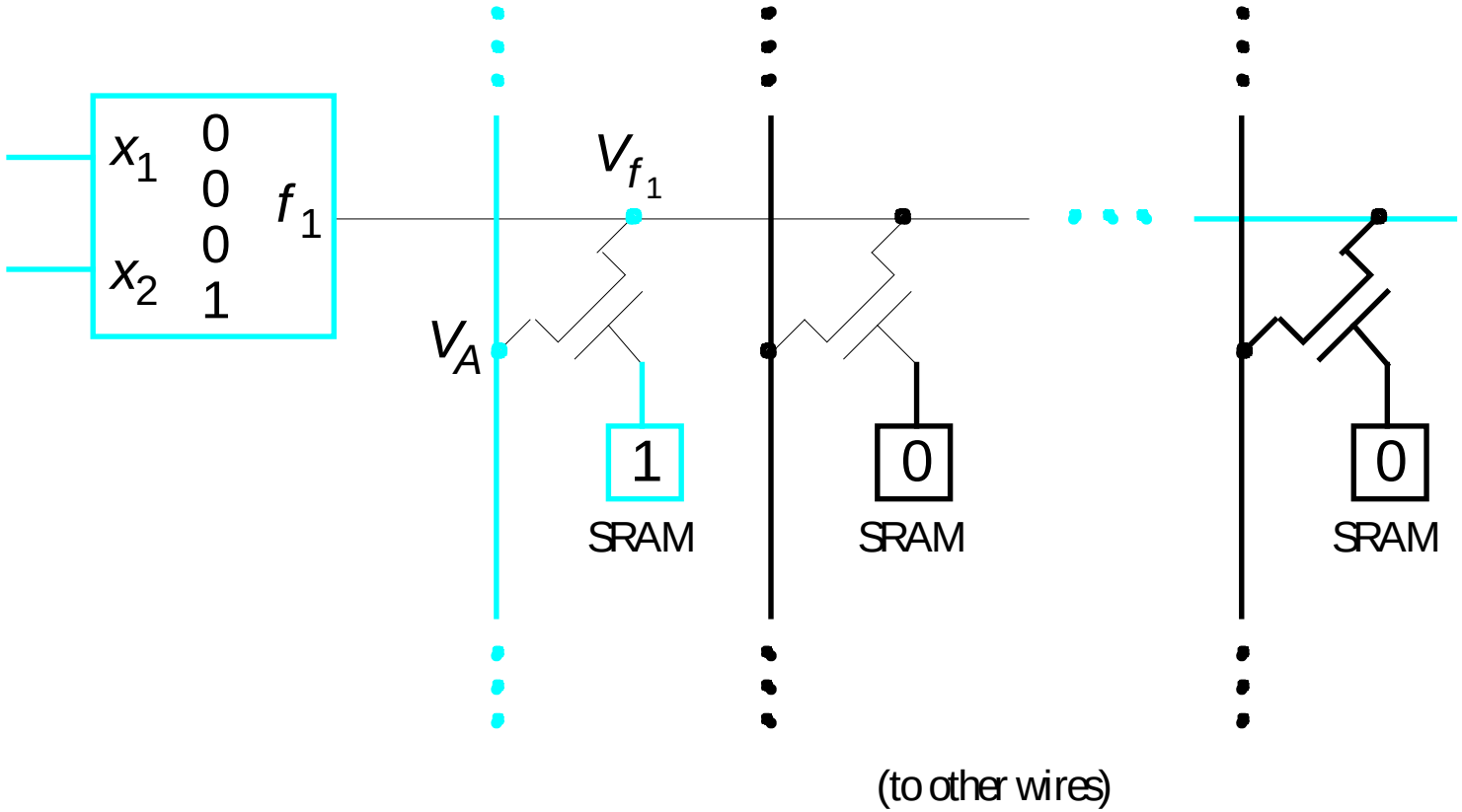


Figure B.73. Pass-transistor switches in FPGAs.