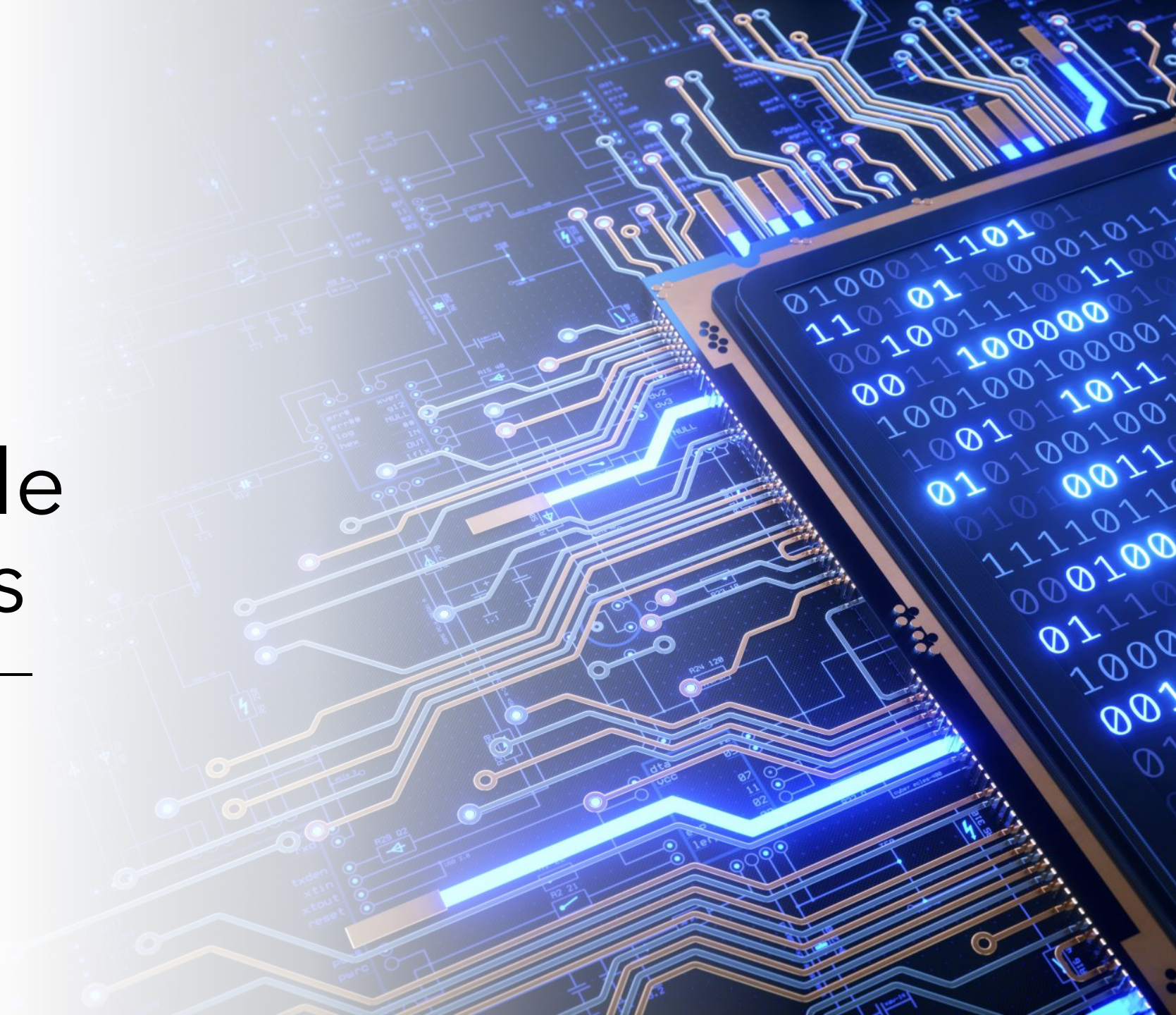
Programmable Logic Devices

History and Concepts



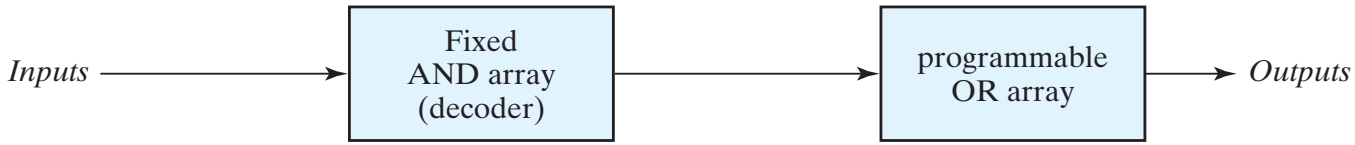
Non-Programmable Design

- Define the required circuit
- Develop using transistors or off-the-shelf components
- Issues
  - Errors require going back to the drawing board and rebuilding the circuit
  - Each application requires a custom solution
  - Long dev cycles --> expensive development

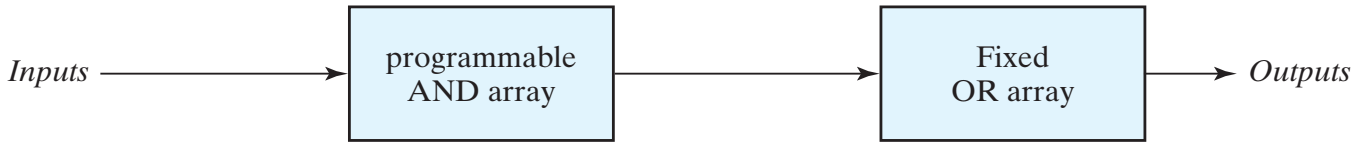
Programmable Logic Approach

- Design a circuit that can be configured programmatically to deliver a desired outcome
- Advantages
  - Make a mistake? Just fix the program
  - Fast development time
  - A single hardware solution can solve many application problems
- Disadvantage
  - Circuit is complex – might be more expensive than other approaches?

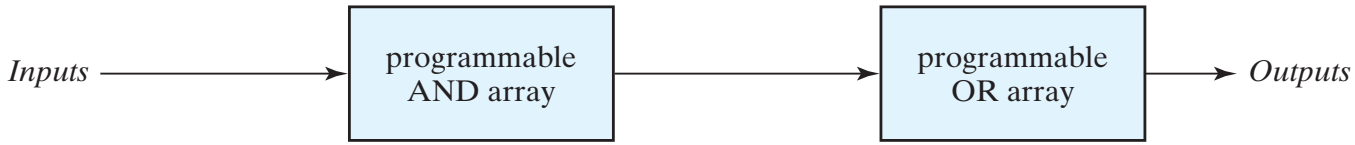
Categories of PLDs



(a) Programmable read-only memory (PROM)

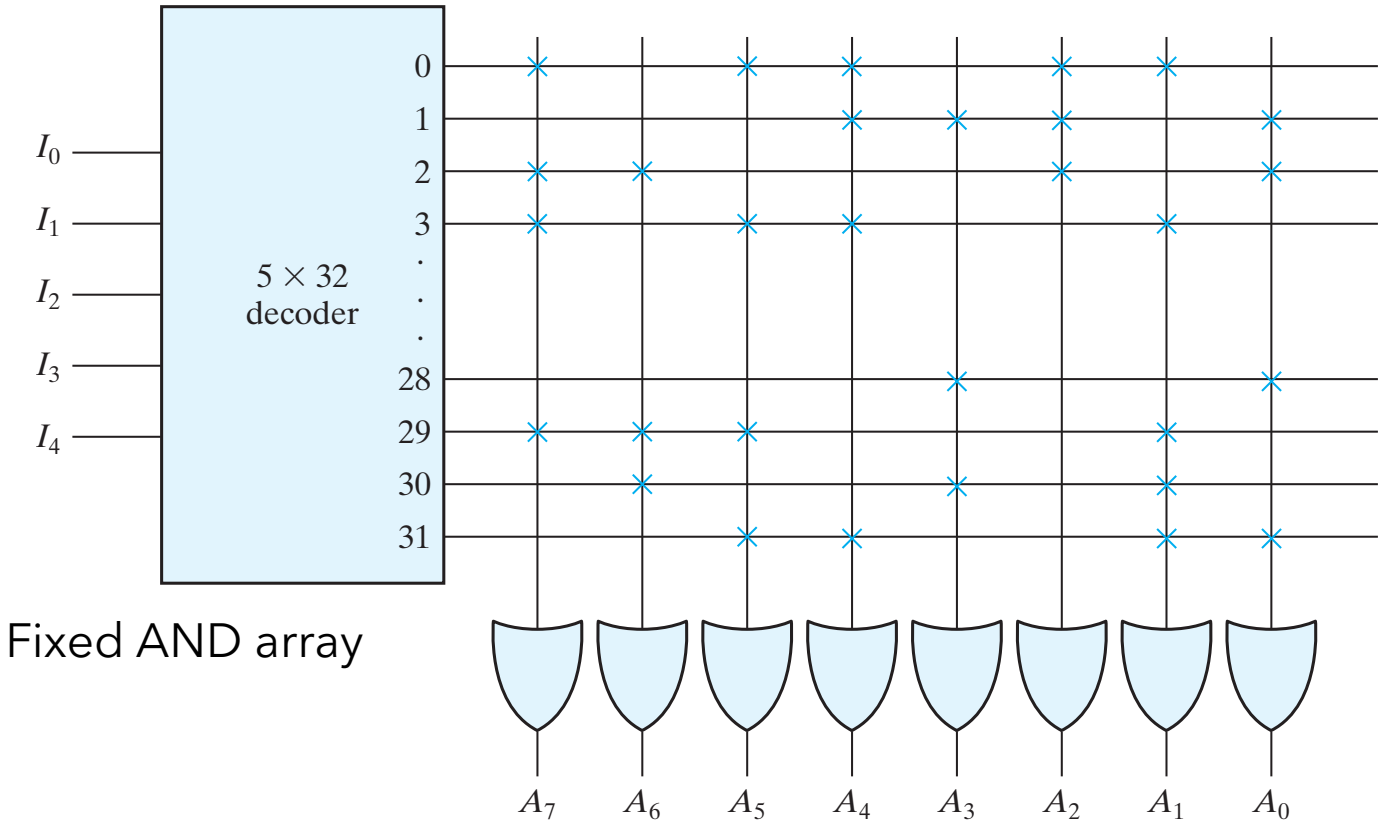


(b) Programmable array logic (PAL)



(c) Programmable logic array (PLA)

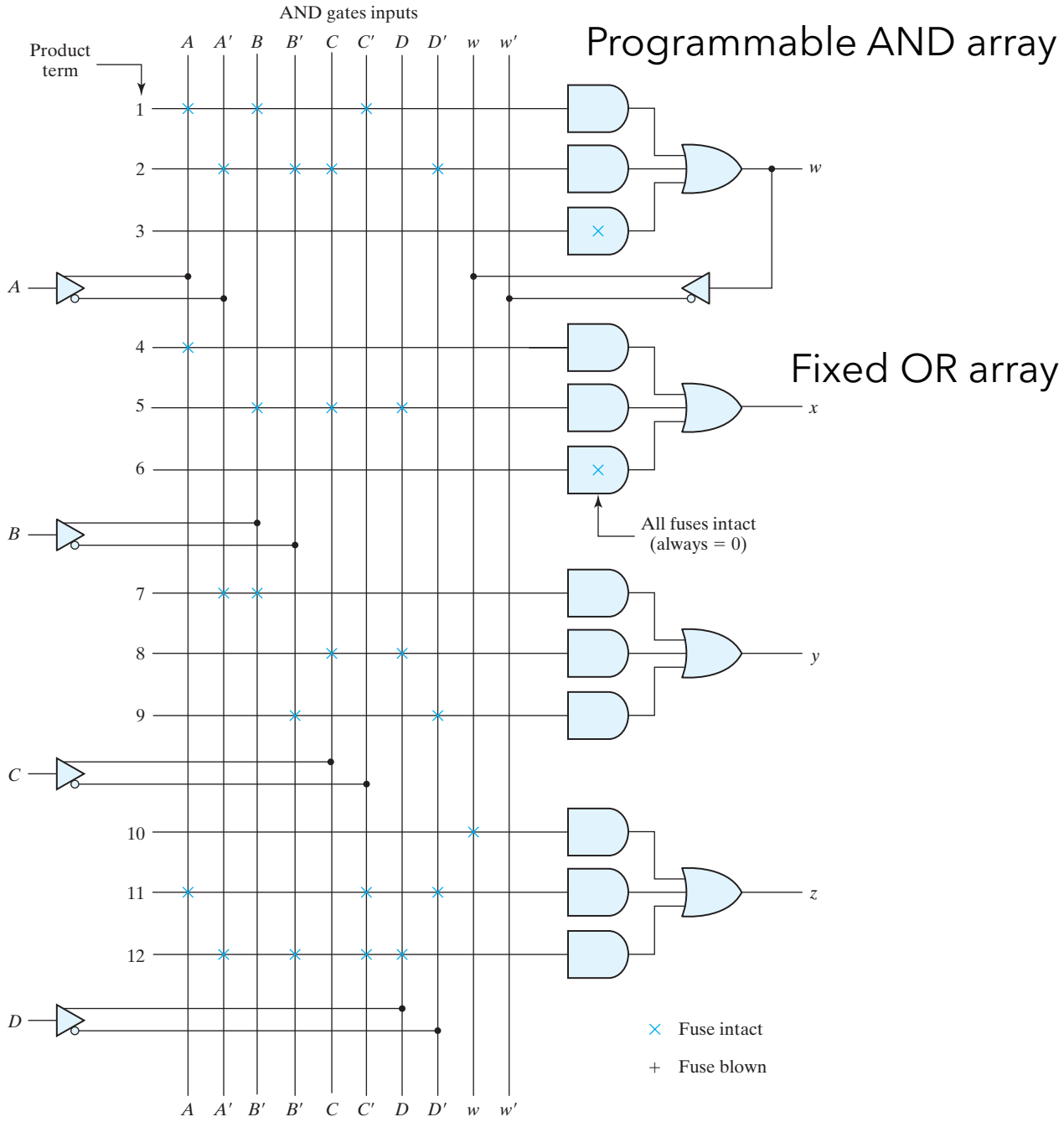
PROM Example



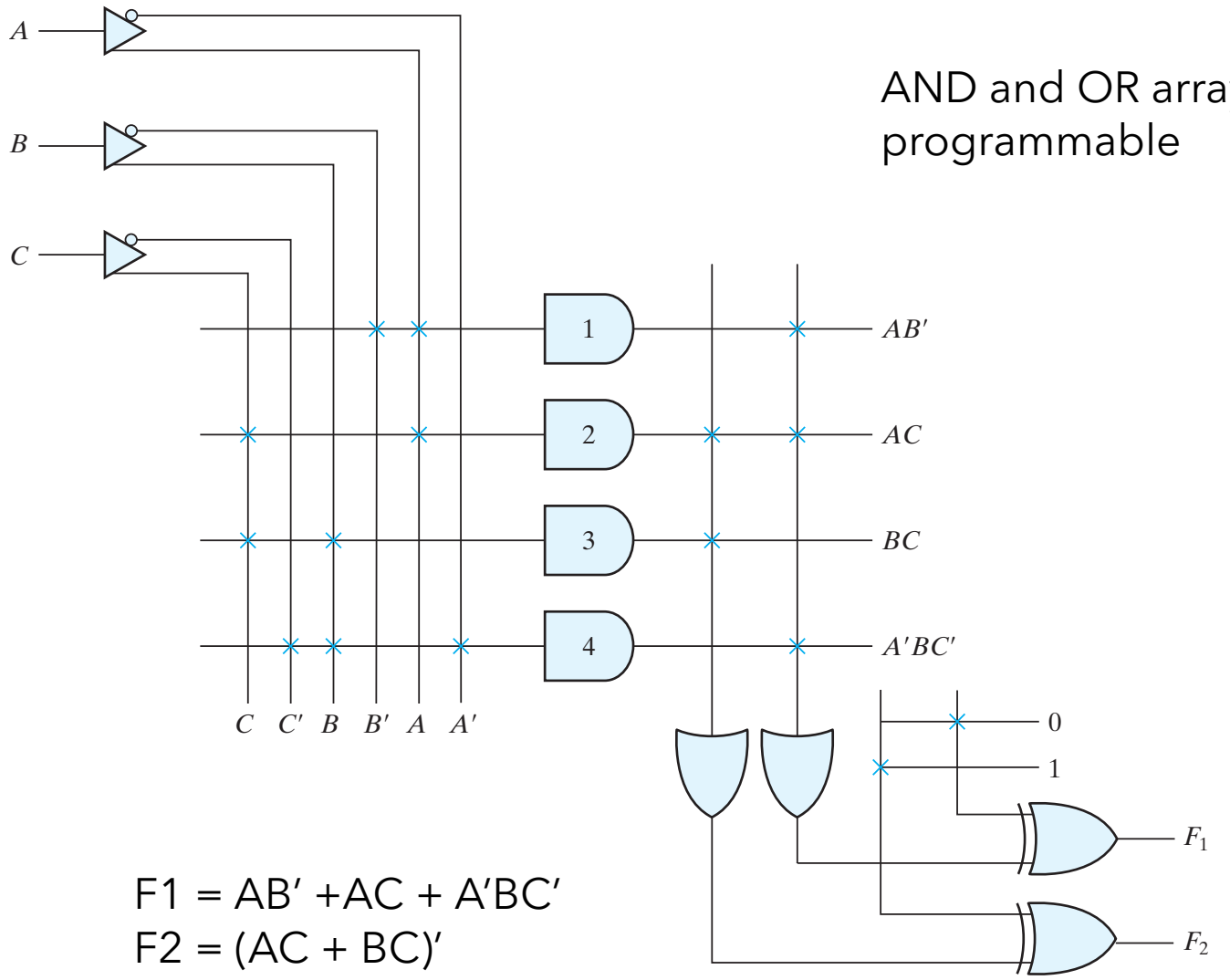
Fixed AND array

Programmable
OR array

PAL Example


AND gates inputs
Programmable AND array
Product term
A A′ B B′ C C′ D D′ w w′
1
2
3
A
4
5
6
B
7
8
9
C
10
11
12
D
w
x
y
z
Fixed OR array
All fuses intact (always = 0)
Fuse intact
Fuse blown

PLA Example

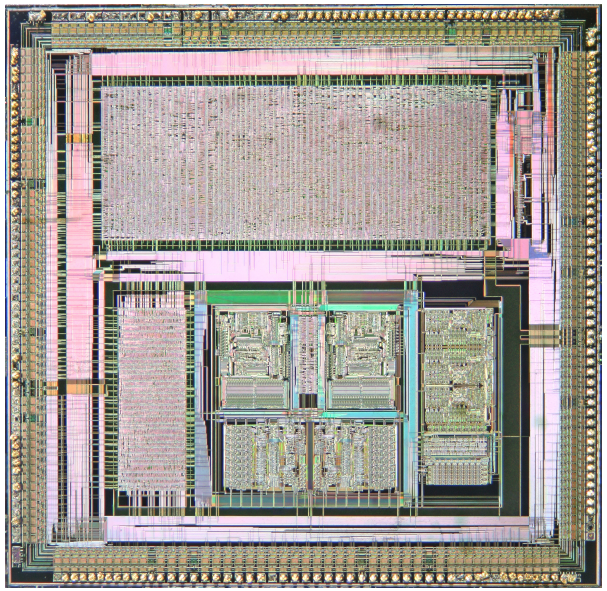


AND and OR arrays programmable

F1 = AB′ +AC + A′BC′
F2 = (AC + BC)′

Application Specific Integrated Circuits (ASIC)

- ASICs are chips that are built for a specific purpose
- Designed using hardware description languages (HDL)
- Very expensive to create, however
  - Economical at scale
- Complex / time consuming development cycle



Field Programmable Gate Arrays (FPGA)

- Programmable device capable of *almost* any digital function
- Unlike a microcontroller, it is effectively a blank slate
- The designer programs the chip to determine the functionality. Based on programming, it could be
  - Digital signal processor
  - Microcontroller
  - Complex combinational/sequential logic system

FPGA Advantages

- Field programmable
  - Updates can be performed after deployment (“in the field”)
- Infinitely (well pretty much) configurable
- Highly parallel
  - Can be configured to run many completely independent functions in parallel – no bottleneck from the CPU
- Extremely fast (again, no CPU in the way)
- Capable of handling a high number of I/O ports
  - Cyclone V from Intel has up to 550 I/O pins

FPGA Downsides

- Difficult to program
- Very expensive in comparison to other solutions
- Tools are complex
- Most require external chips (SRAM) to operate
- Power consumption can be high

Progression of FPGAs

- **Gates**
  - 1987: 9,000 gates, Xilinx
  - 1992: 600,000, Naval Surface Warfare Department
  - Early 2000s: Millions
  - 2013: 50 Million, Xilinx

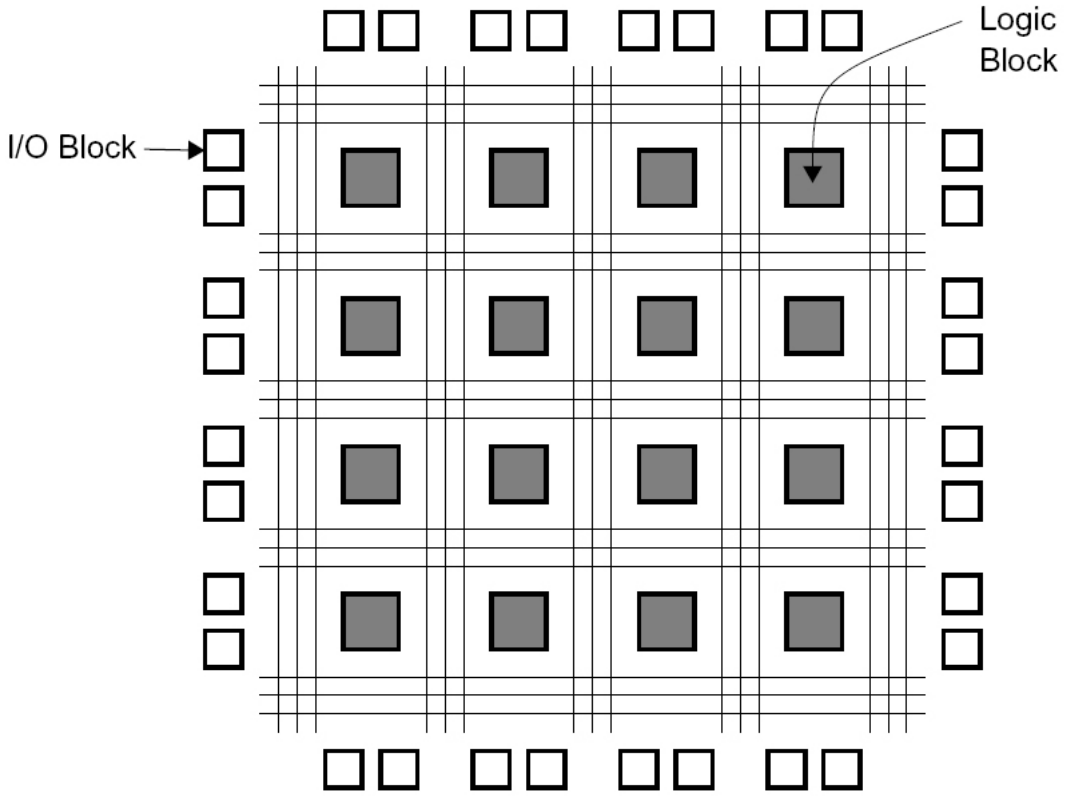
Market size

1985: First commercial FPGA : Xilinx XC2064
1987: $14 million
≈1993: >$385 million
2005: $1.9 billion
2010 estimates: $2.75 billion
2013: $5.4 billion
2020 estimate: $9.8 billion

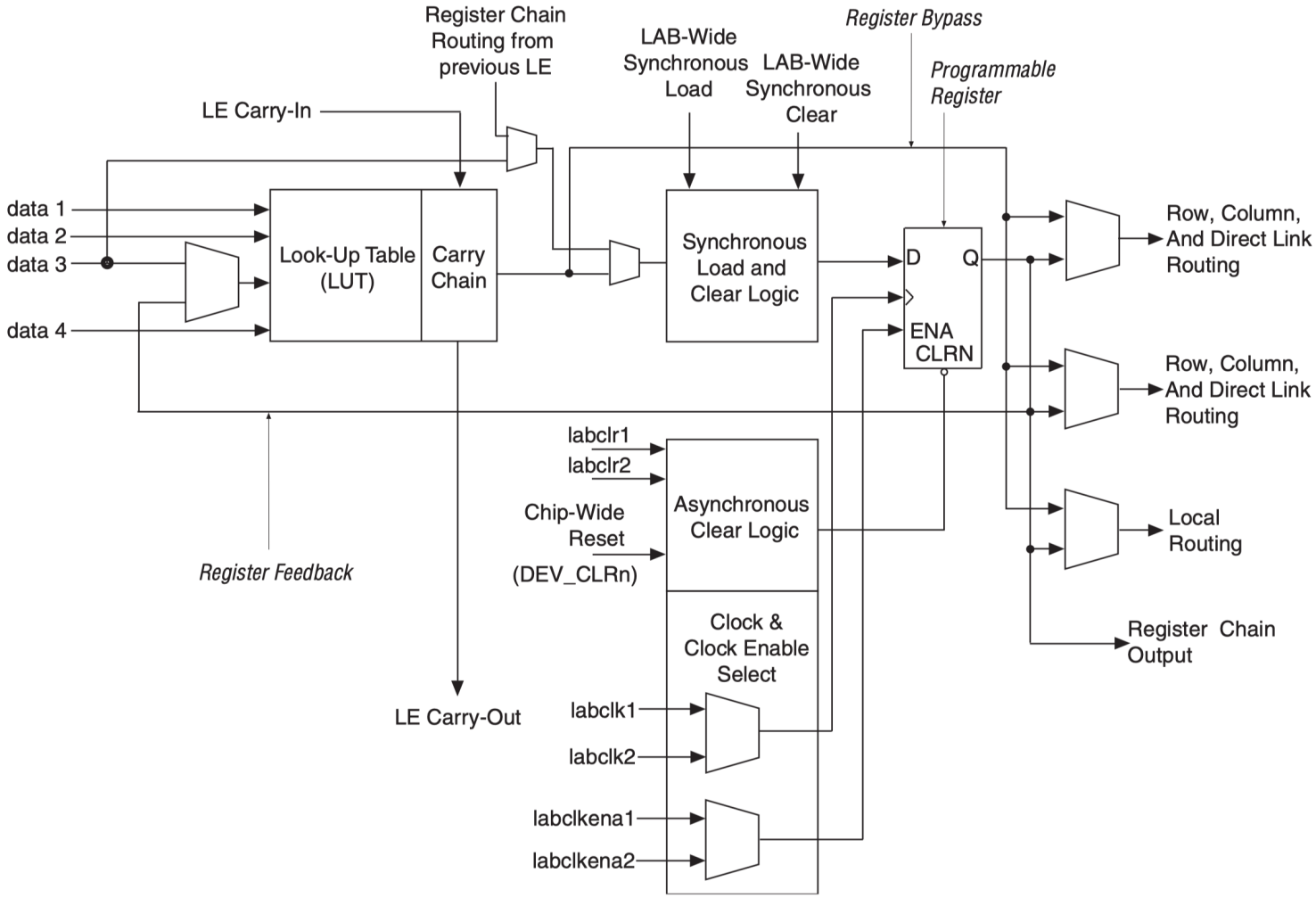
http://en.wikipedia.org/wiki/Field-programmable_gate_array

FPGA Structure

- Logic blocks are the building blocks of the system
- Blocks are programmatically connected via the underlying core fabric to form the required circuit
- Intel® Stratix® 10 GX has up to 10.5 million logic blocks

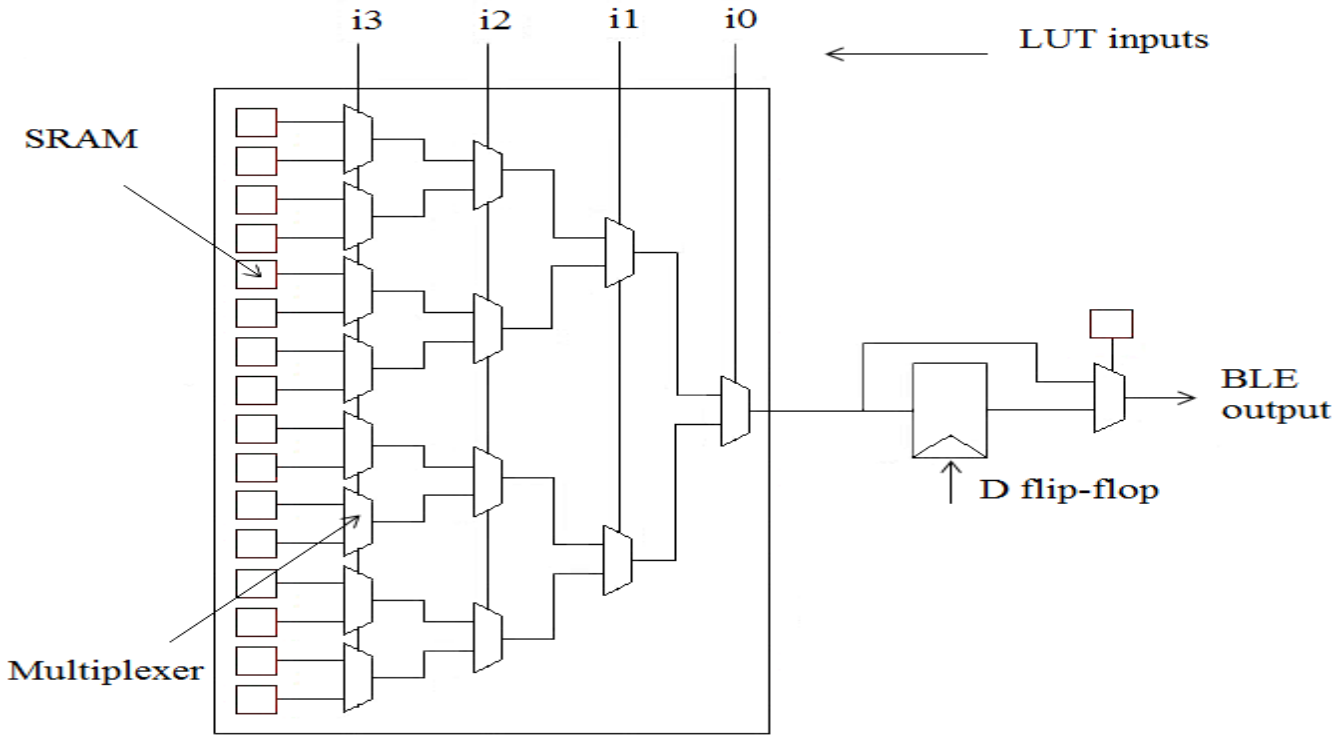


Example Logic Element from Cyclone IV



The Look Up Table (LUT)

- The LUT can implement any 4-variable Boolean expression
- SRAM is programmed with the output values for the the function
- The input variables select values from MUXes based on the table



Alizadeh, Bijan & Mehri, Hossein. (2014). An Analytical Dynamic and Leakage Power Model for FPGAs. 10.1109/IranianCEE.2014.6999552.

FPGA

FPGA

FPGAs belong to a class of devices known as programmable logic, or sometimes referred to as programmable hardware.

Essentially, an FPGA doesn’t do anything itself but it can be configured to be just about any digital circuit you want.

Nothing physically changes.

You simply load a configuration into the FPGA and it starts behaving like the circuit you wanted.

The configuration is RAM based which means it can essentially be reconfigured an unlimited number of times.

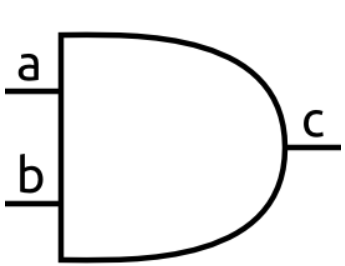
FPGA

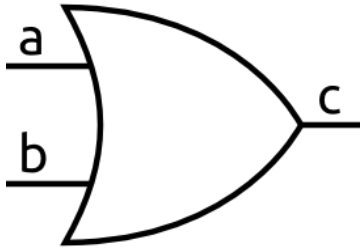
Even though we talk about using FPGAs to create digital *circuits*, you don’t typically draw schematics to create designs for them.

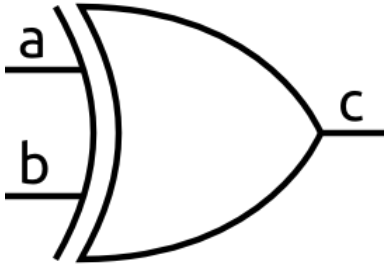
The size and complexity of the circuits FPGAs can contain would become very cumbersome should you actually draw out a schematic.

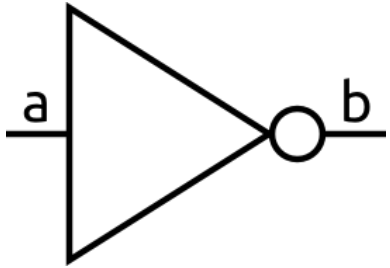
Instead, you can describe the behavior of the circuit you want and the tools will use this to create a circuit that matches that behavior.

Logic gates









AND

In A	In B	Out
0	0	0
1	0	0
0	1	0
1	1	1

NAND

In A	In B	Out
0	0	1
1	0	1
0	1	1
1	1	0

OR

In A	In B	Out
0	0	0
1	0	1
0	1	1
1	1	1

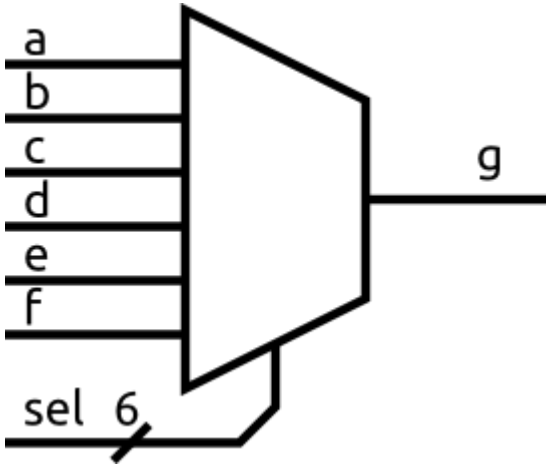
NOR

In A	In B	Out
0	0	1
1	0	0
0	1	0
1	1	0

XOR

In A	In B	Out
0	0	0
1	0	1
0	1	1
1	1	0

A multiplexer selects a single input out of set based on the value of its select input.

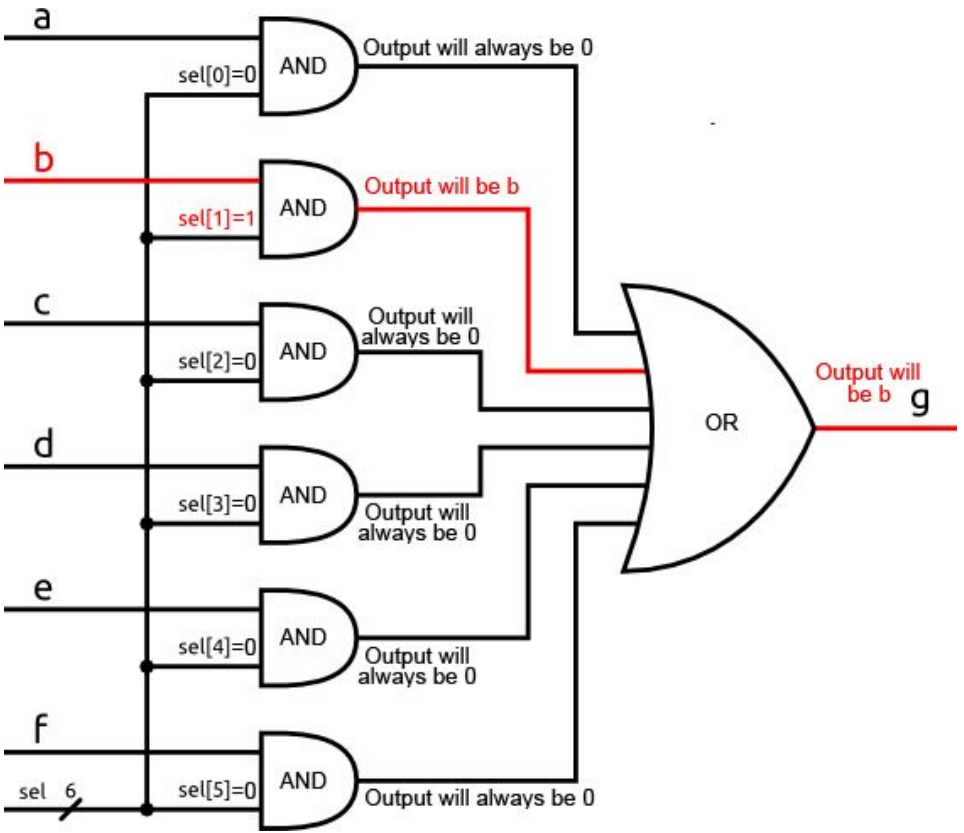


Setting *sel* value to be 000010

A large matrix of multiplexers with a programmable *sel* input.

Allows to route signals based on the design.

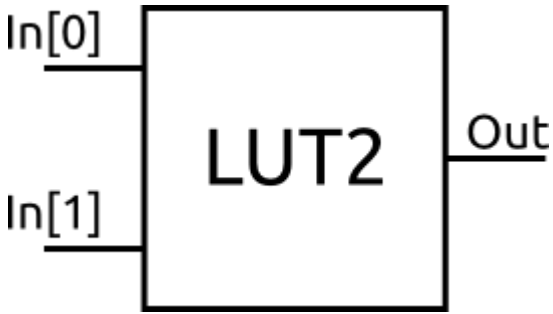
This is how FPGAs get their signals where they need to be and it is called the general routing matrix.



Look Up Table

Imagine we have a multiplexer with four inputs and a 2 bit binary select.

Instead of exposing the main inputs to the world, let’s hook them up to some programmable memory.



A LUT consists of a block of SRAM that is indexed by the LUT's inputs. The output of the LUT is whatever value is in the indexed location in it's SRAM.

Although we think of RAM normally being organized into 8, 16, 32 or 64-bit words, SRAM in FPGA's is 1 bit in depth. So for example a 3 input LUT uses an 8x1 SRAM ($2^3=8$)

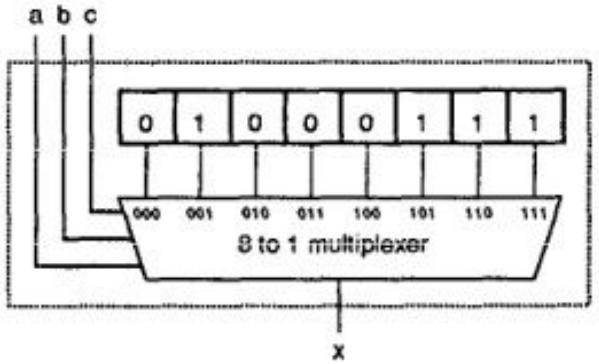
Because RAM is volatile, the contents have to be initialized when the chip is powered up. This is done by transferring the contents of the configuration memory into the SRAM.

For a two-input AND gate,

Address (In[1:0])	Value (Out)
00	0
01	0
10	0
11	1

The LUT is actually implemented using a combination of the SRAM bits and a MUX:

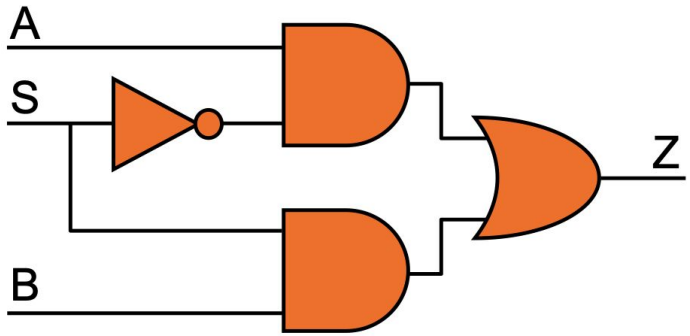
Here the bits across the top 0 1 0 0 0 1 1 1 represents the *output* of the truth table for this LUT. The three inputs to the MUX on the left a, b, and c select the appropriate output value.



b) 3-Input LUT

Gates are combined to create complex circuits

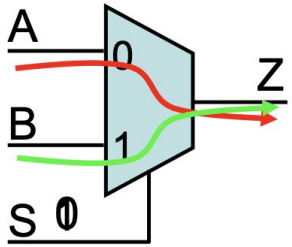
- Multiplexer example
  - If S = 0, Z = A
  - If S = 1, Z = B
  - Very common digital circuit
  - Heavily used in FPGAs

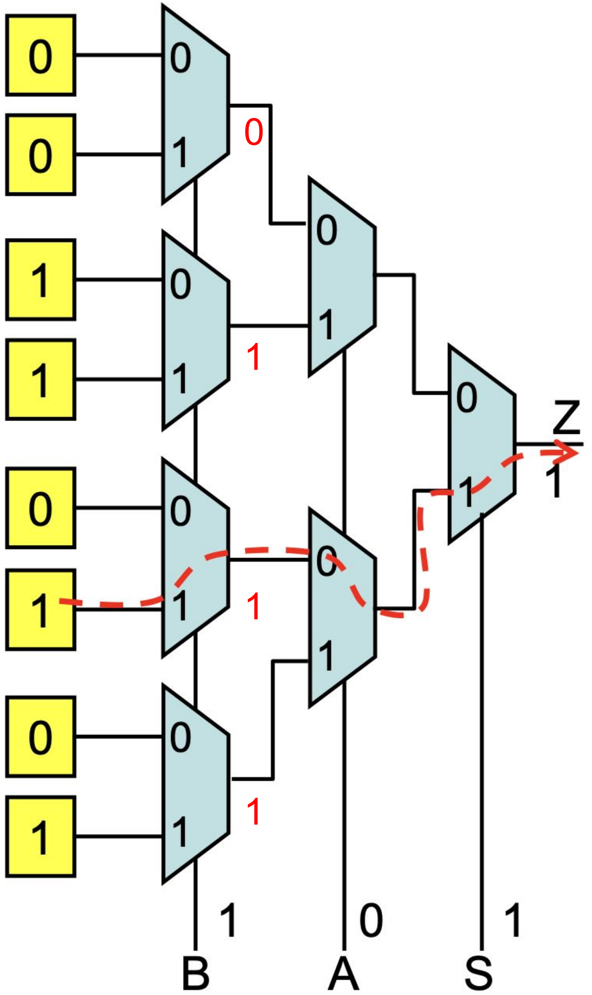


Truth table

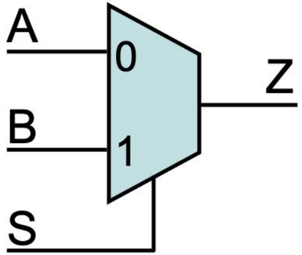
S	A	B	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Logic symbol



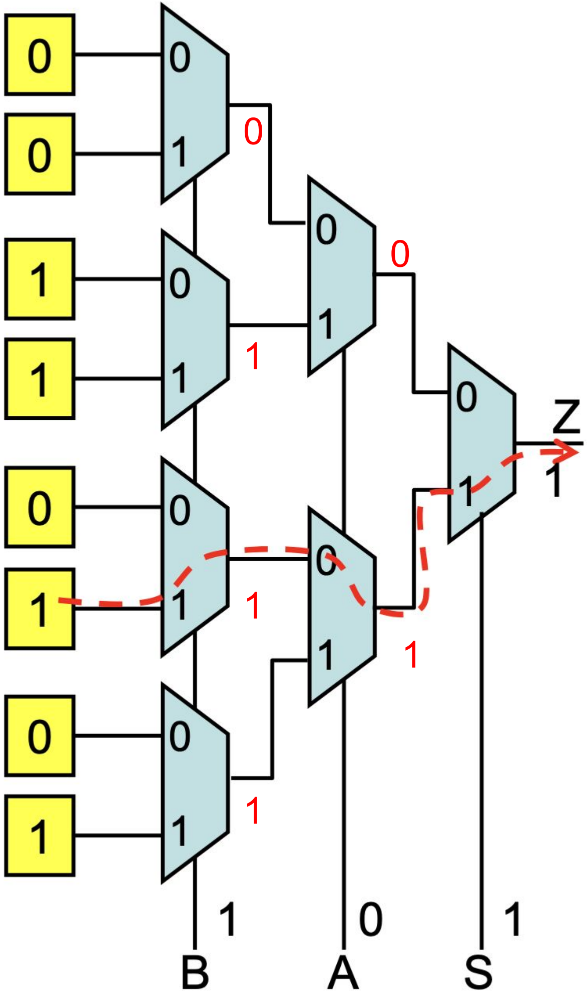


Multiplexer

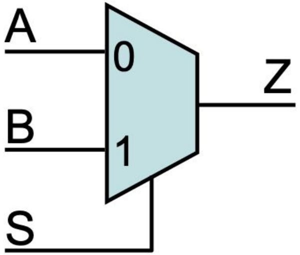


Truth table

S	A	B	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



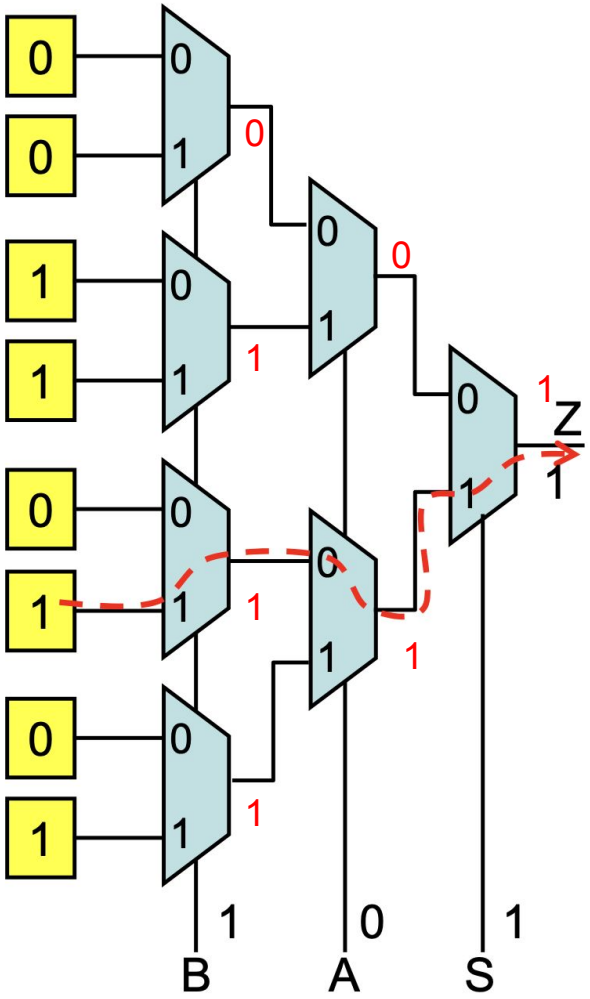
Multiplexer



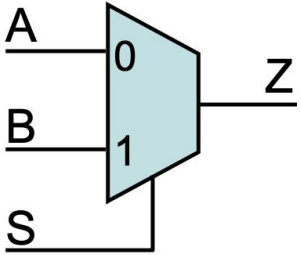
Truth table

S	A	B	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

up Tables



Multiplexer



Truth table

S	A	B	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Why Use an FPGA?

Let’s look at a trivial example of turning an LED on when you press a button.

In Arduino, the processor would run a small loop of code that would read the state of a pin then update the state of another pin based on that value.

In the case of simply connecting a button to an LED with an FPGA, you simply connect the button and the LED. The value from the button passes through some input buffer, is fed through the routing matrix, then output through an output buffer. This process happens continuously all the time. The only delay comes from the switching delays of the transistors in the chip, which are incredibly small.

hardware description languages (HDLs)

To describe digital circuits is to use a textual language that is specifically intended to clearly and concisely capture the defining features of digital design.

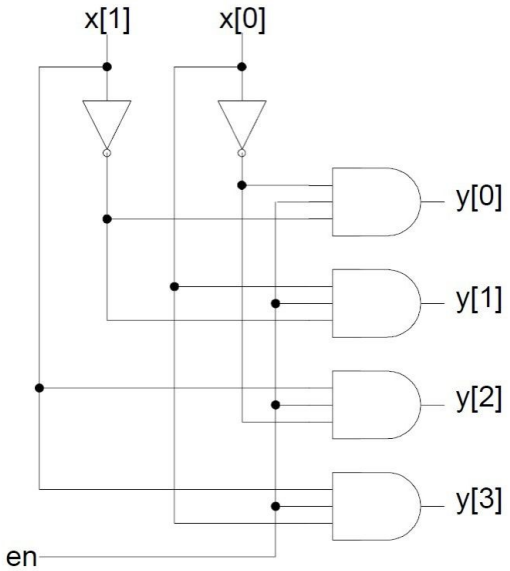
What Do HDLs Do?

The most popular hardware description languages are Verilog and VHDL. They are widely used in conjunction with FPGAs, which are digital devices that are specifically designed to facilitate the creation of customized digital circuits.

Hardware description languages allow you to describe a circuit using words and symbols, and then development software can convert that textual description into configuration data that is loaded into the FPGA in order to implement the desired functionality.

```
module gate2 (in_a, in_b, out);
input wire in_a; // declare input variable in_a
input wire in_b; // declare input variable in_a
output wire [5:0] out; // declare out variable out
assign out[5] = in_a&in_b; // AND operation
assign out[4] = ~(in_a&in_b); // NAND operation
assign out[3] = in_a | in_b; // OR operation
assign out[2] = ~(in_a | in_b); // NOR operation
assign out[1] = in_a ^ in_b; // XOR operation
assign out[0] = ~(in_a ^ in_b); // NXOR operation
endmodule
```

2:4 decoder



Enable	INPUTS		OUTPUTS			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

https://learn.sparkfun.com/tutorials/how-does-an-fpga-work/all

https://www.eng.auburn.edu/~strouce/class/elec4200/FPGAoverview.pdf

https://www.tutorialspoint.com/vlsi_design/vlsi_design_verilog_introduction.htm