FYSIKUM

Digital System Construction

Lecture 1: Introduction

Course Overview

Digital electronics

FPGAs and programmable logic

General Information

- Instructor
  - Sam Silverstein (5537 8693)
- Lab assistant
  - Katie Dunne
- Course material on Athena

Goals of the course

- Fundamentals of digital logic
- Learn to specify and digital designs in the VHDL language.
- Simulate and implement digital circuits in programmable logic devices
- Important topics in digital design, including:
  - Finite state machines
  - Memories and their applications
  - Digital system processing
  - CPU/microprocessor fundamentals

Course Organization

- Focus: practical exercises
  - Supervised labs in conjunction with introductory lectures.
- Lectures
  - Attendance strongly encouraged
- Lab exercises
  - Required for course completion

How will the course be run this year?

- Remote learning as much as possible
- Lectures will be held over ZOOM
- Lab exercises at home (as much as possible)
  - You will be loaned a Digilent BASYS3 FPGA board for the course
  - You need to install Xilinx Vivado on your computer (multiple ways to do this)
  - On-site work possible by appointment

Grading for the course:

- Bologna grading system:
  - A, B, C, D, E, Fx, F
- Grading criteria based on:
  - Well-written code that works correctly
  - Number/difficulty of projects completed
- How your work will be evaluated:
  - Submitted code, screenshots from completed labs
  - Oral discussions with instructor

Lab exercises

- Required for C grade:
  - Combinatorial logic (parallel adder)
  - Sequential logic (serial adder)
  - Pseudo-random number generator and programmable fixed-delay buffer
  - Digital stopwatch
- Advanced projects (one for B, two+ for A):
  - UART serial data receiver
  - Simple microprocessor in VHDL
  - Arbitrary function generator
  - Digital notch filter

Course material

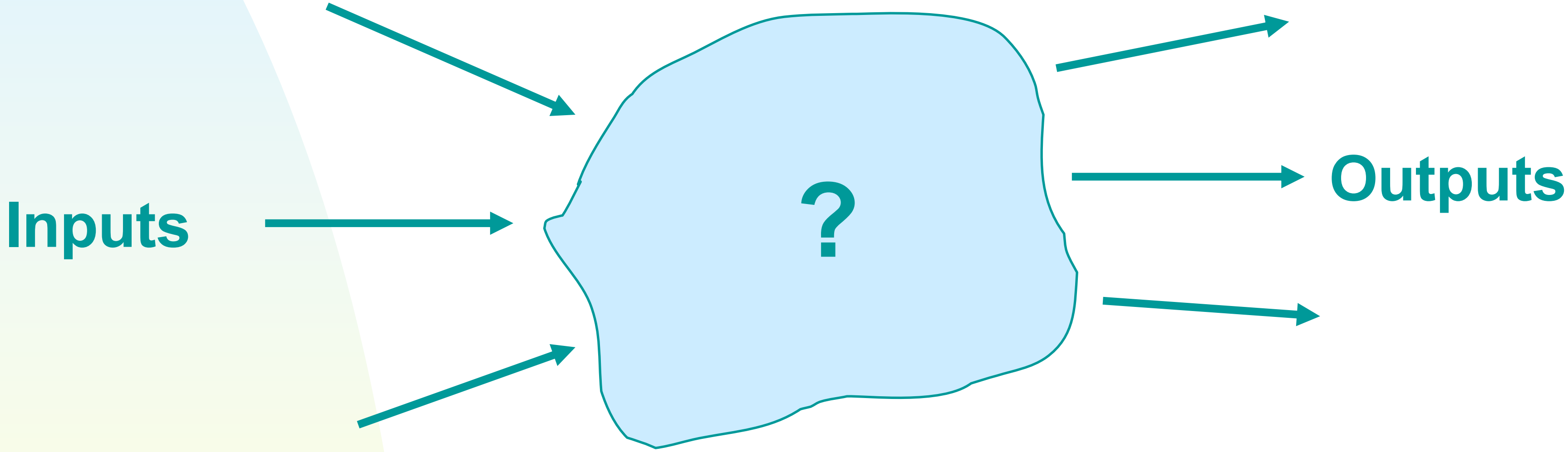
- Textbook:
  - Free Range VHDL (Mealy, Tappero)
    - Well-written open souce textbook
    - Downloadable PDF on Athena
- Lecture notes and labs
  - PDFs posted on Athena
- Many other good web pages and videos available online.

Digital electronics - a short introduction (review)

Digital logic

- Combinatorial
  - Output is a function only of the current inputs to the circuit
- Sequential
  - Output is a function of both the current inputs and the previous states of the circuit

Combinatorial logic:



$$outputs = f(inputs)$$

Basic logic gates

Truth tables

- AND
  - True if <u>all</u> inputs are true
  - $C = A \cdot B$

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

- OR
  - True if <u>any</u> input is true
  - $C = A + B$

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

Basic logic gates

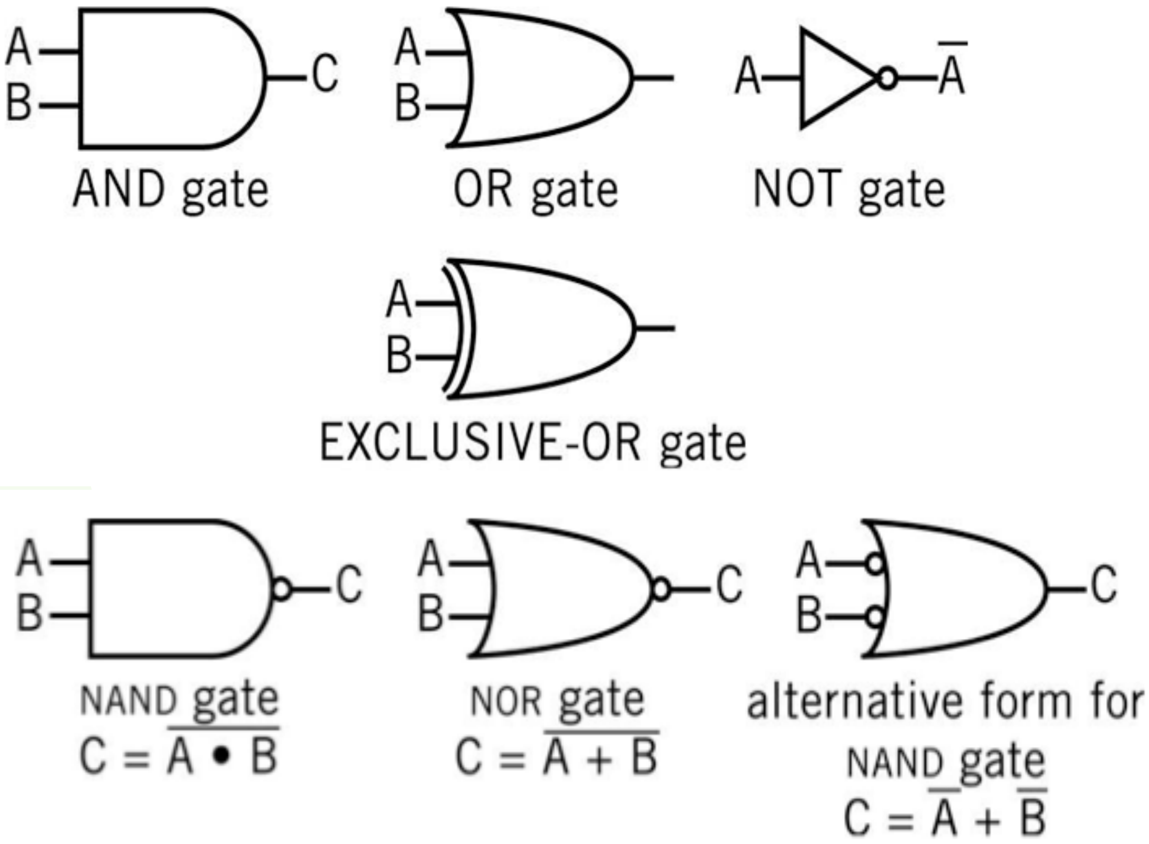
- NOT
  - Output is inverse of the input
  - C = $\overline{A}$

A	C
0	1
1	0

- XOR
  - True if only one input is true
  - C = $A \oplus B$

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

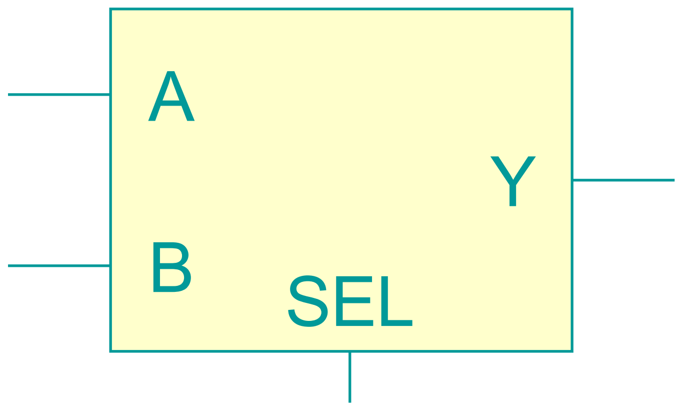
Gate representation (American)



Example 1: Multiplexer

- Select from multiple inputs
  - Y = A when SEL = 1
  - Y = B when SEL = 0

A	B	SEL	Y
a	b	1	a
a	b	0	b



Gate implementation

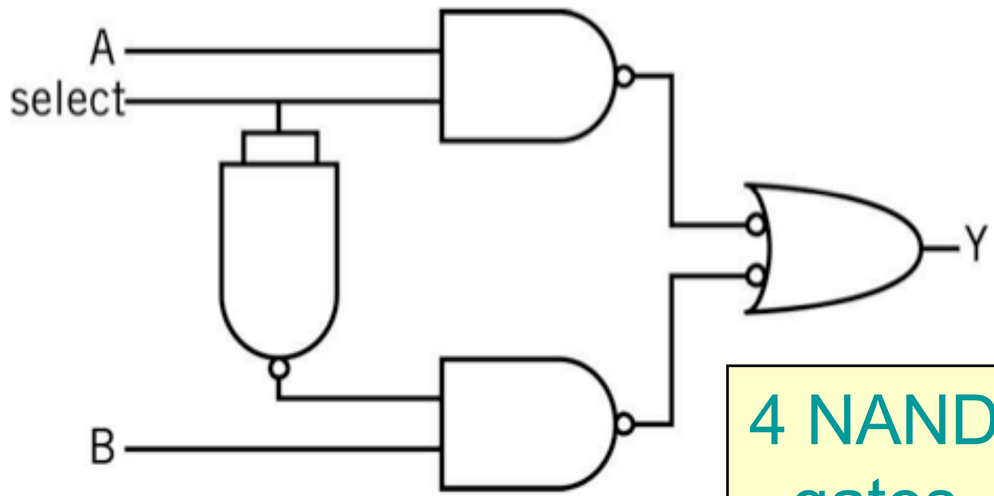
Truth table

A	B	SEL	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$Y = (A \bullet S) + (B \bullet \overline{S})$$

$$= \overline{\overline{(A \bullet S)}} + \overline{\overline{(B \bullet \overline{S})}}$$

$$= \overline{(A \text{ nand } S)} + \overline{(B \text{ nand } \overline{S})}$$



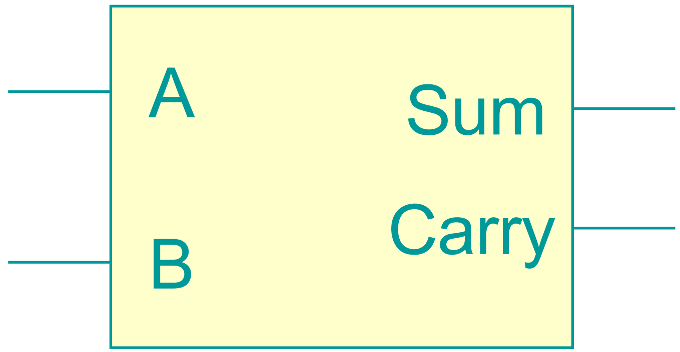
4 NAND gates

Example 2: Half-Adder

- Add two 1-bit binary numbers
- Two output bits: sum and carry

A	B	Sum
0	0	0
0	1	1
1	0	1
1	1	0

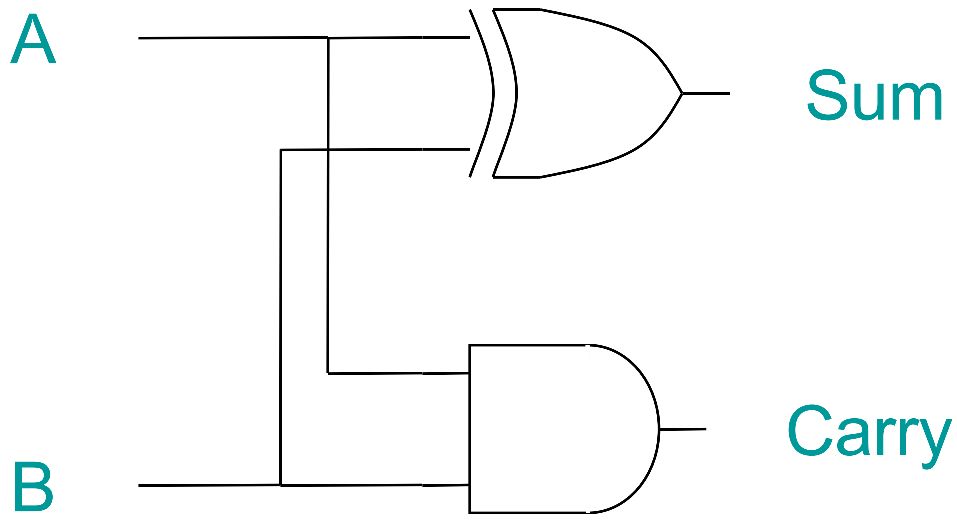
Sum = XOR



A	B	Carry
0	0	0
0	1	0
1	0	0
1	1	1

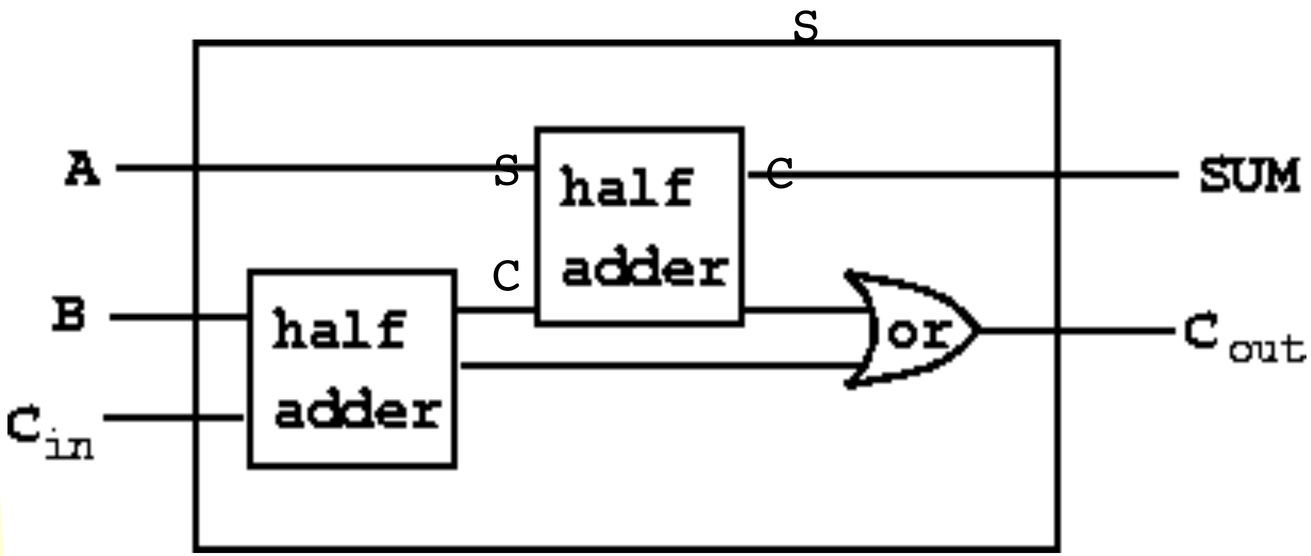
Carry = AND

Gate Implementation

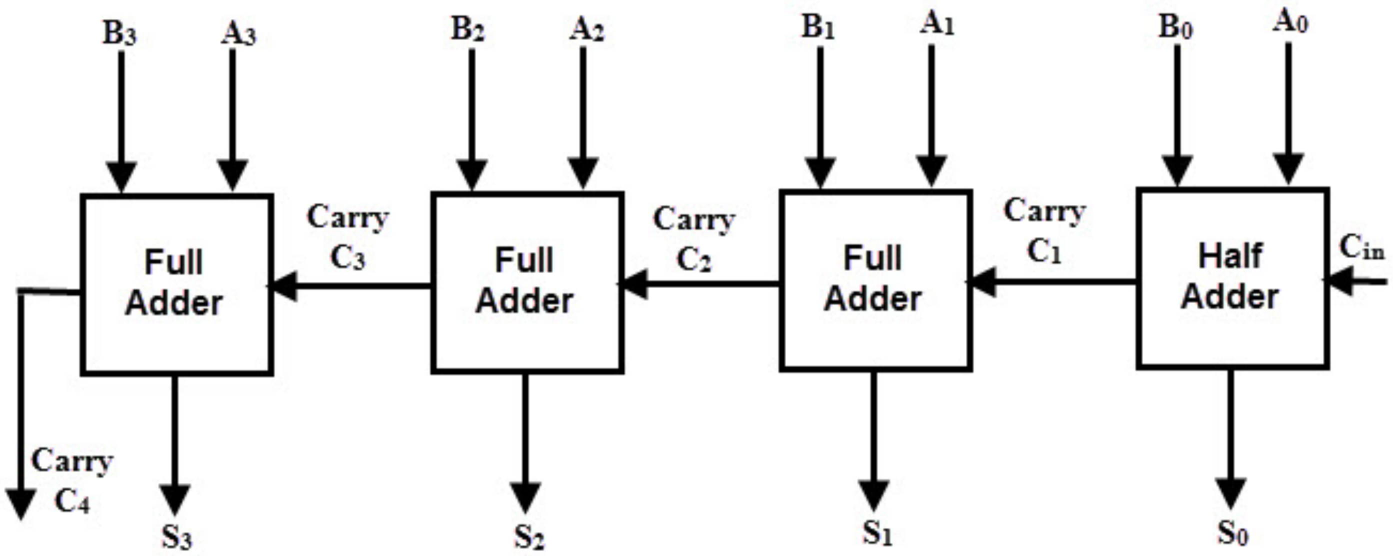


Example 3: Full adder

- Need to add three bits:
  - A and B
  - carry bit from previous sum



Adding larger numbers



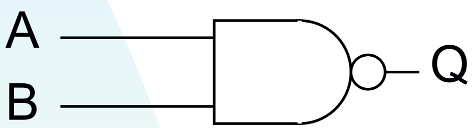
Daisy-chain of full adders

Digital logic has speed limits

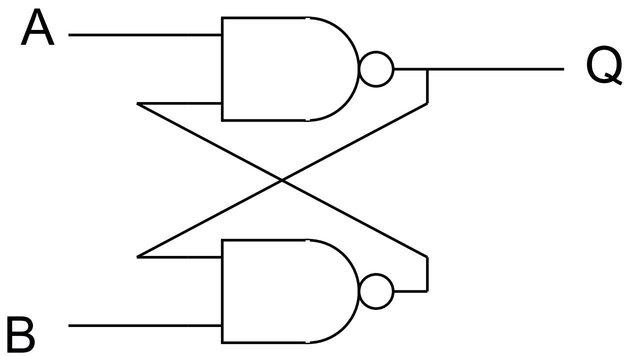
- Non-zero gate delays
  - More gates in series ➔ longer delay ➔ slower logic
    - For example, carry chain the in previous slide
  - There are ways to get around it
    - Modern “fast-carry” logic can reduce a little
    - Pipelined algorithms: Break into several steps (has its own cost…will discuss later)

Sequential logic

Combinatorial

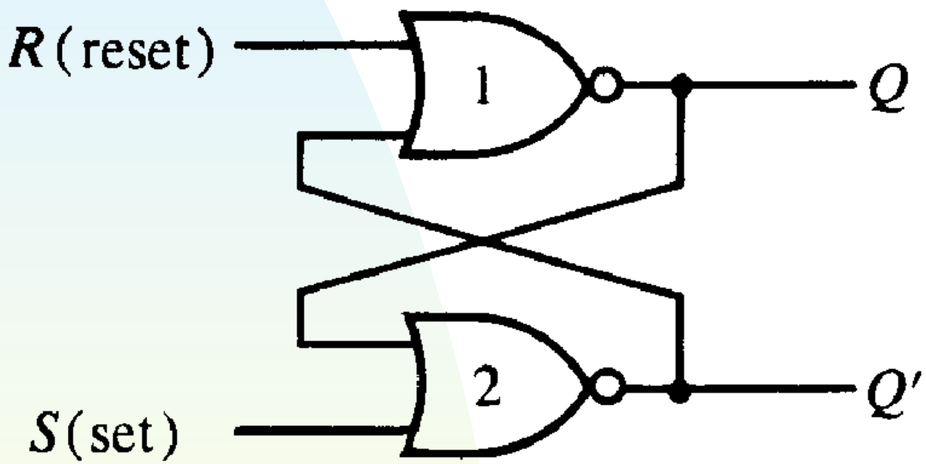


Sequential



Output depends on previous <u>state</u> of the circuit

Example: Bistable Circuit



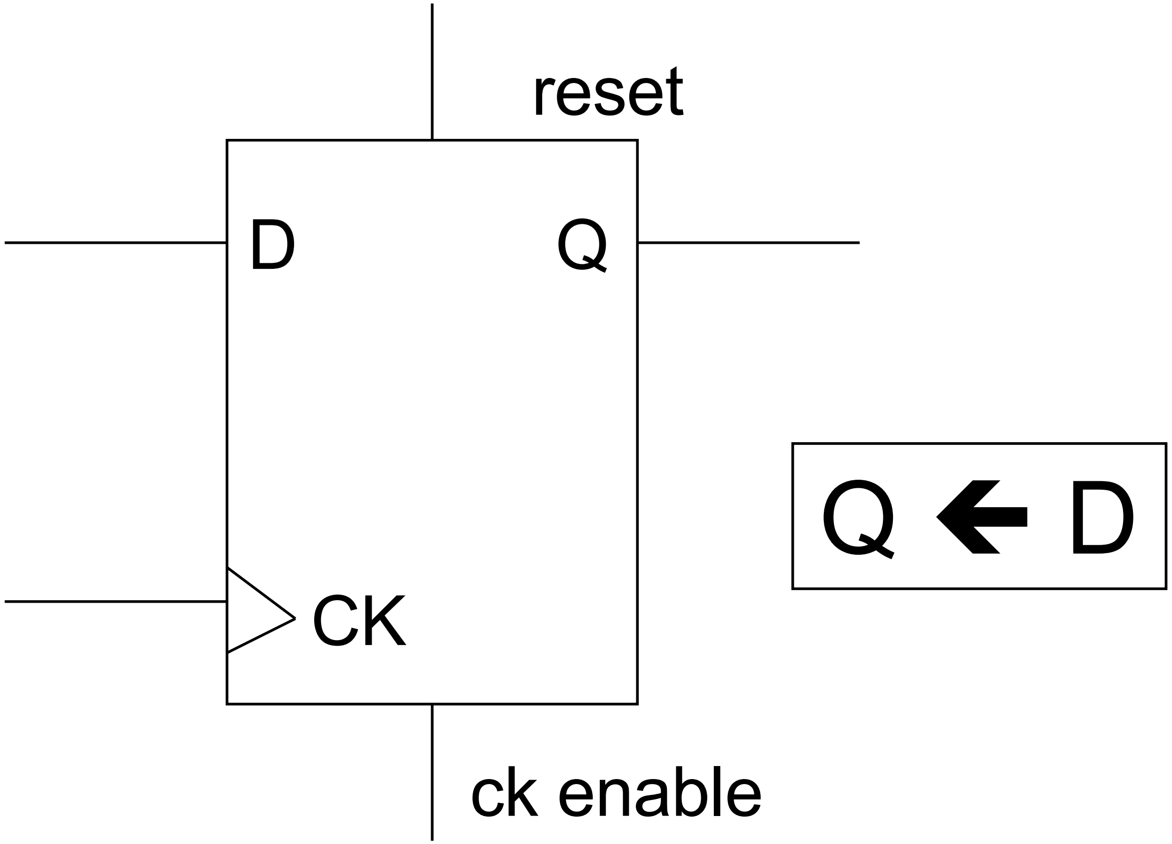
(a) Logic diagram

S	R	Q	Q′	
1	0	1	0	
0	0	1	0	(after $S = 1, R = 0$)
0	1	0	1	
0	0	0	1	(after $S = 0, R = 1$)
1	1	0	0	

(b) Truth table

Registers hold information

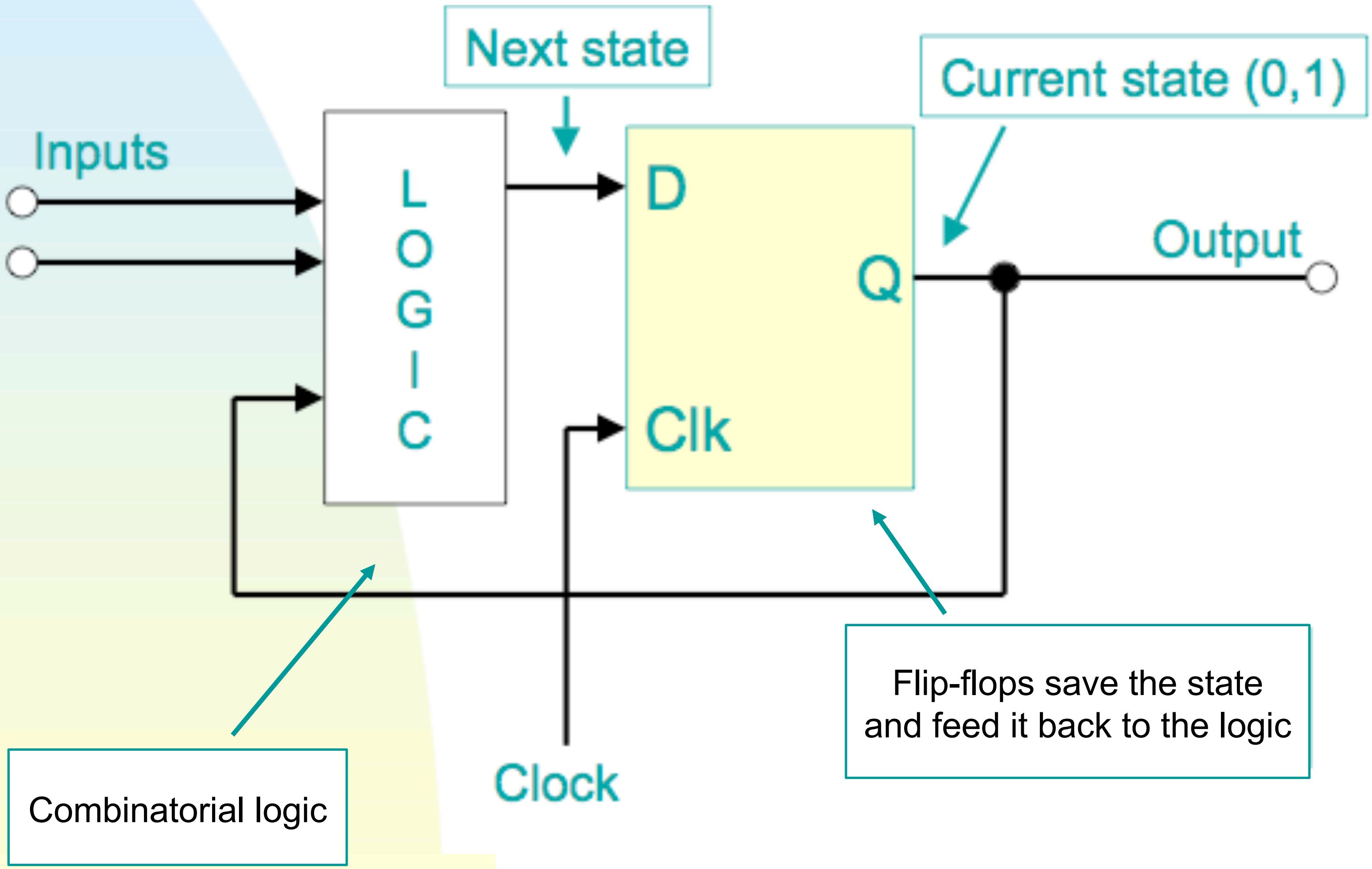
(D “flip-flop”. Holds data “D” until next clock cycle)



Finite state machine (FSM)

- Most general form of sequential circuit
- Output depends on current and previous inputs
- Circuit has different "states"
  - Next state = F (Current state, inputs)
  - Output = G (Current state, inputs)

FSMs have combinatorial logic and flip-flops

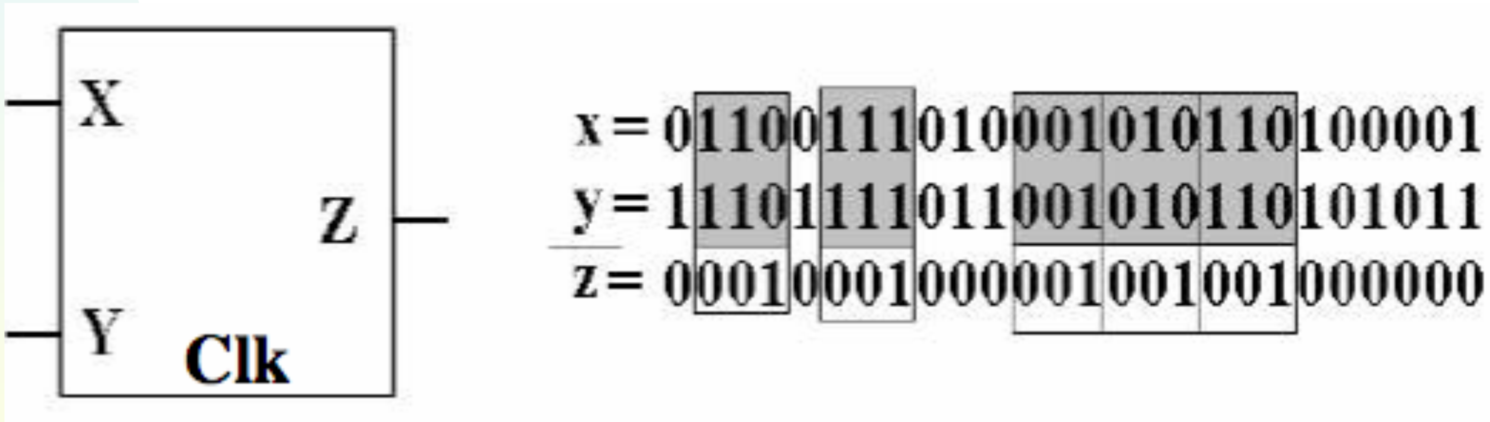


Different types of FSM

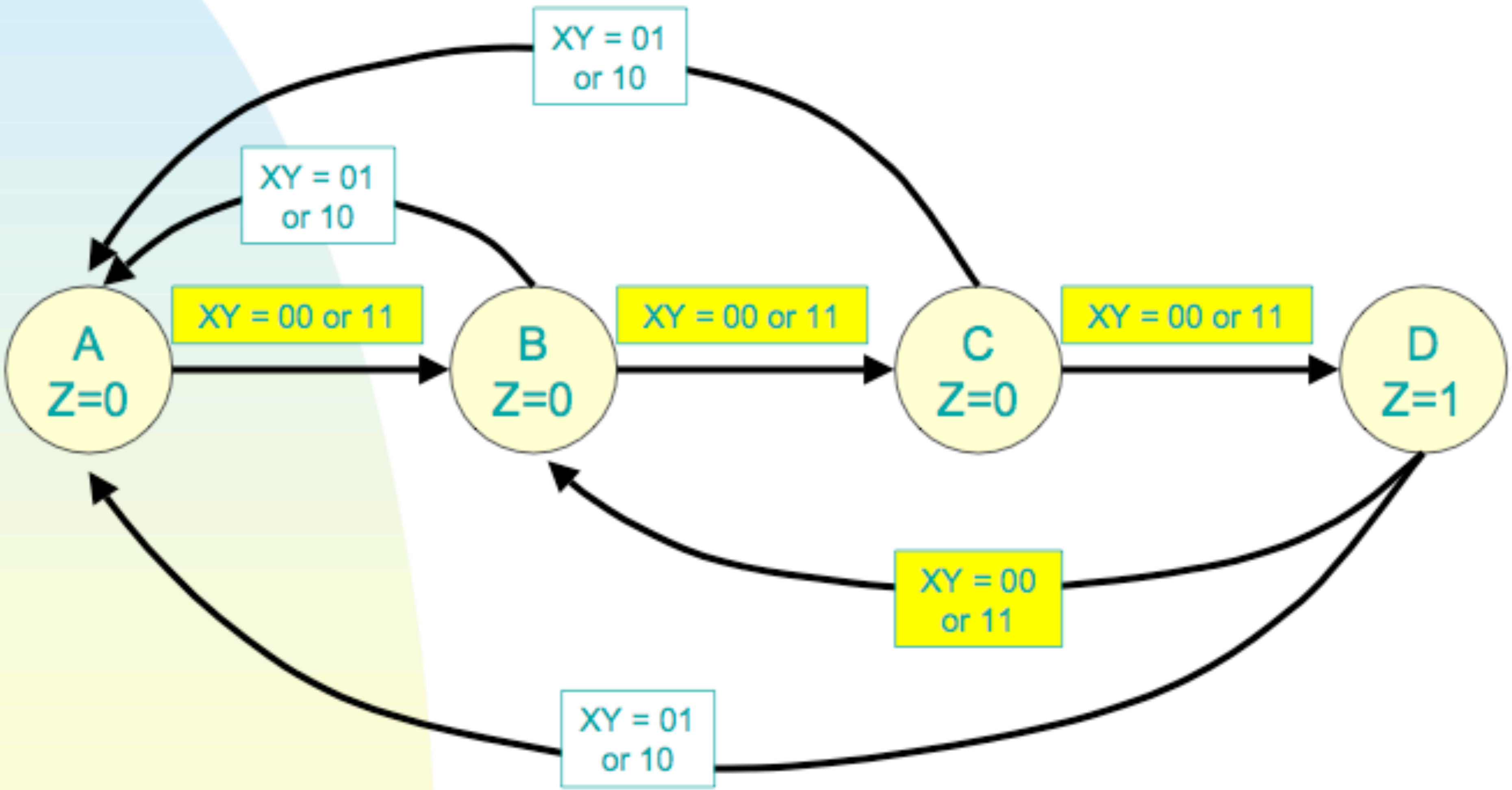
- Clock:
  - Synchronous (with clock)
  - Asynchronous (without clock)
- Output dependency:
  - Moore: outputs depend only on state information
  - Mealy: outputs depend on state information *and* input data

Example: Synchronous Moore FSM

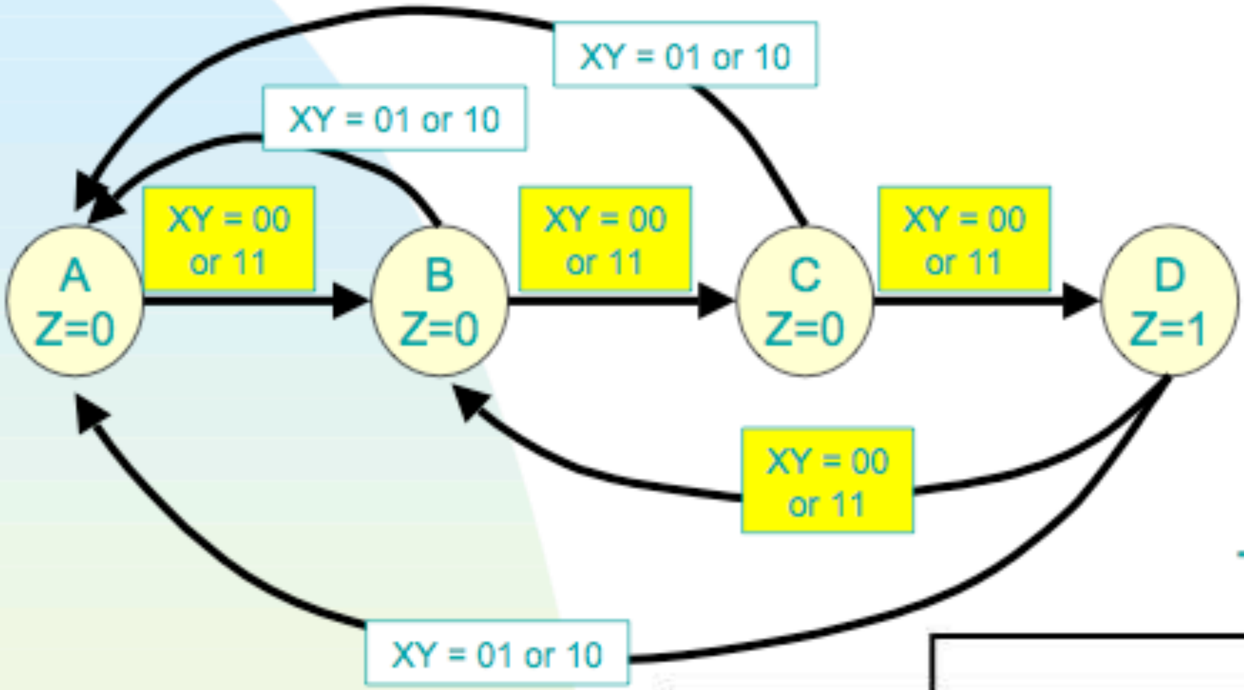
- Let Z equal 1 after inputs X and Y are equal 3 times in a row:



Flow chart for the FSM



Truth table for the states



Truth table:

	xy			
S	00	01	10	11
A	B	A	A	B
B	C	A	A	C
C	D	A	A	D
D	B	A	A	B
	S*			

S = current state
S* = next state

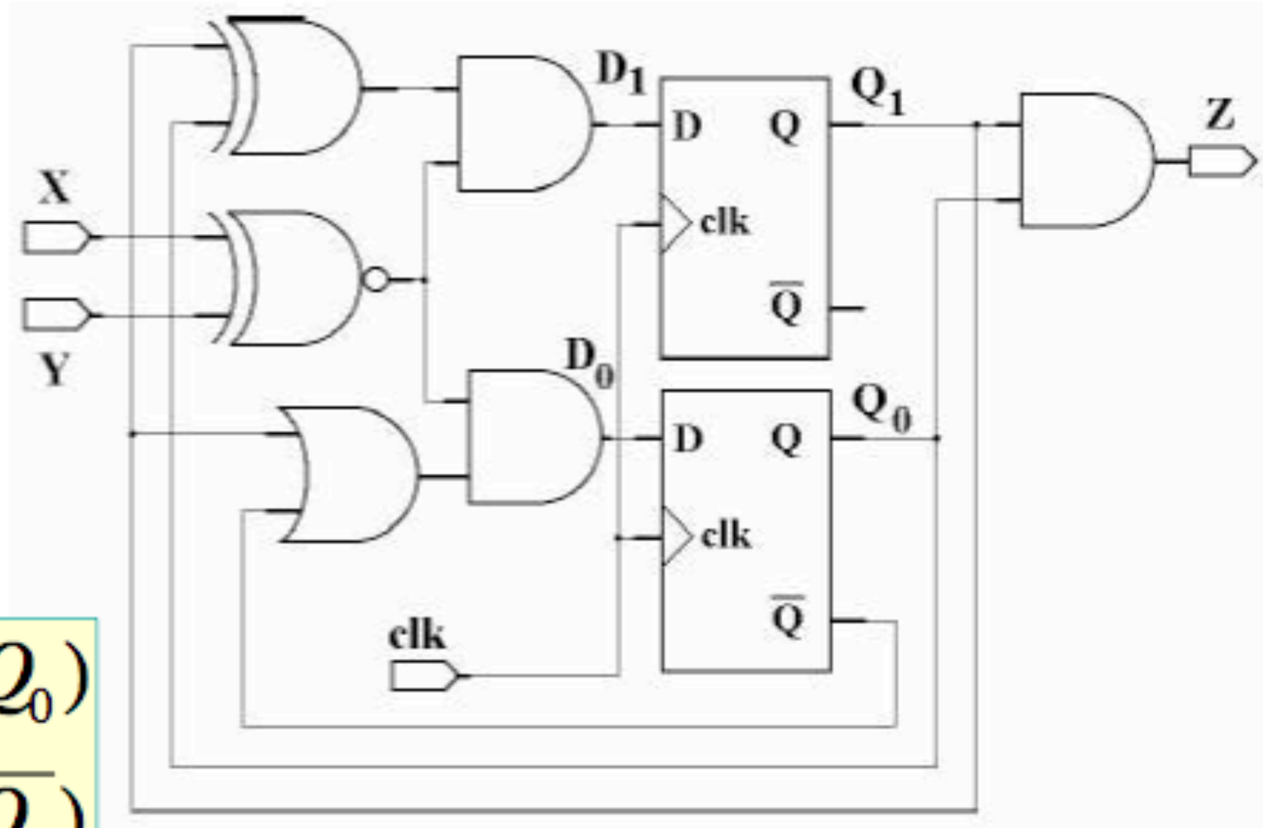
Circuit has two flip flops to encode four states

Q_0, Q_1
$\Rightarrow D_0, D_1$

Clk

$$D_1 = (\overline{x \oplus y}) \bullet (Q_1 \oplus Q_0)$$

$$D_0 = (\overline{x \oplus y}) \bullet (Q_1 + \overline{Q_0})$$



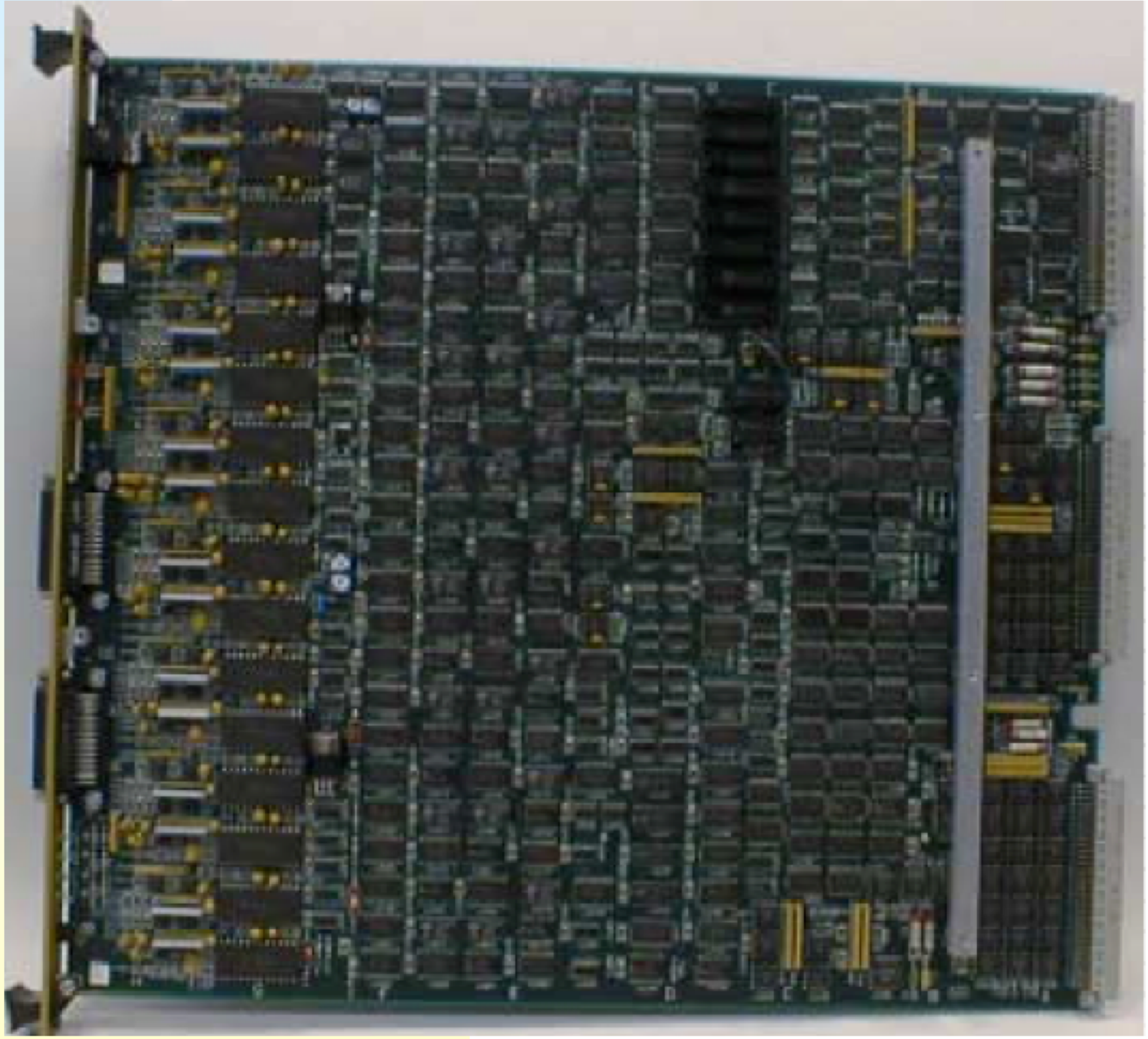
Digital logic summary

- Two types of digital logic
  - Combinatorial: output depends on immediate inputs
    - Can be implemented in gates
  - Sequential: output also depends on “history” of circuit
    - Use gates for combinatorial part, and flip-flops to store previous results

Implementing digital logic

- Discrete logic components
  - Commercial, general-purpose
  - Limited functionality, small range of inputs
- Custom integrated circuits (ASIC)
  - Large-scale integration of complex designs on single chips
  - Single purpose, costly and time-consuming to make
- Programmable logic
  - Commercially available devices that can be configured to implement large, complex designs

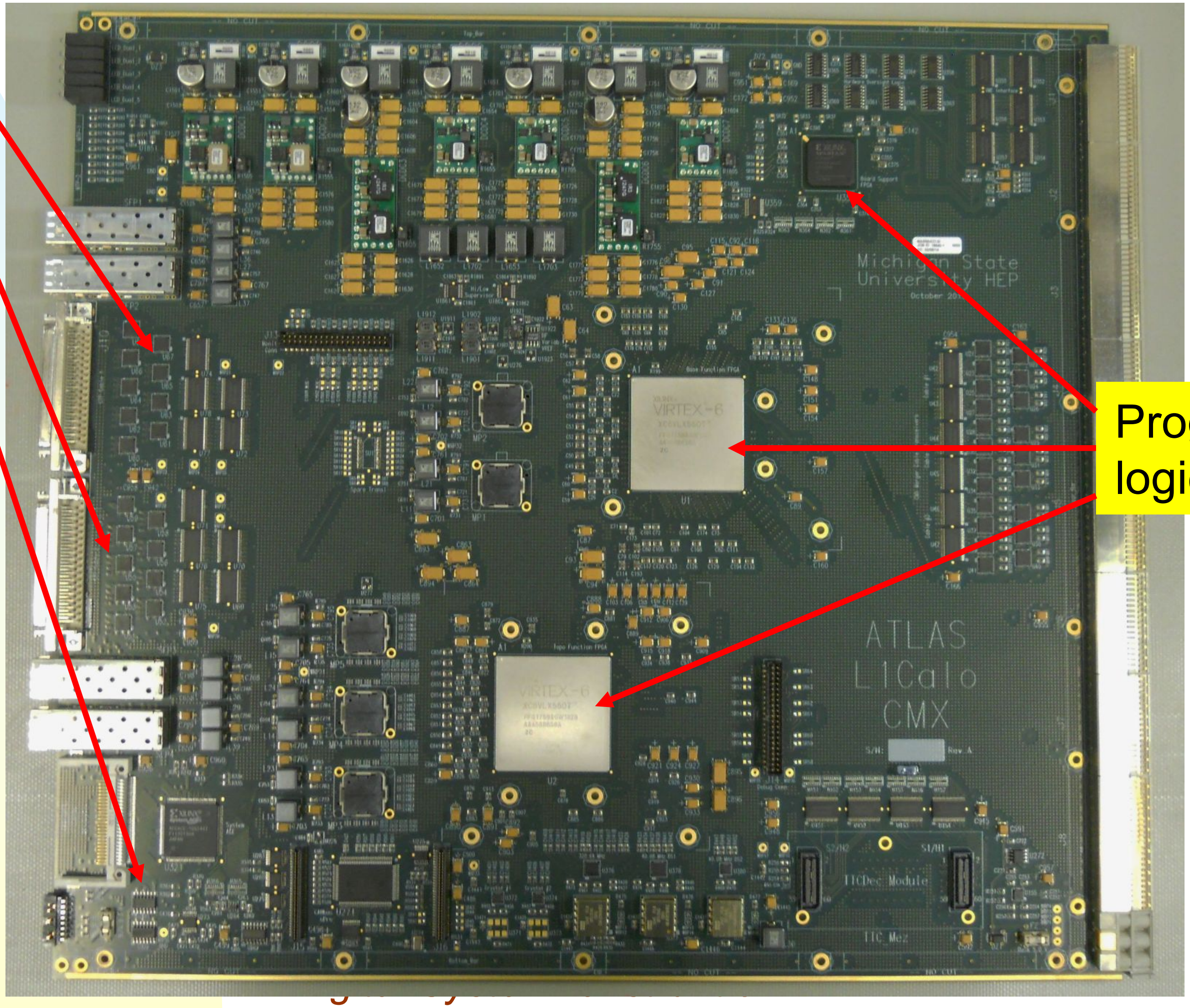
ZEUS trigger encoder card (1990)



Many discrete components

ATLAS trigger data merger module (2013)

Integrated circuits

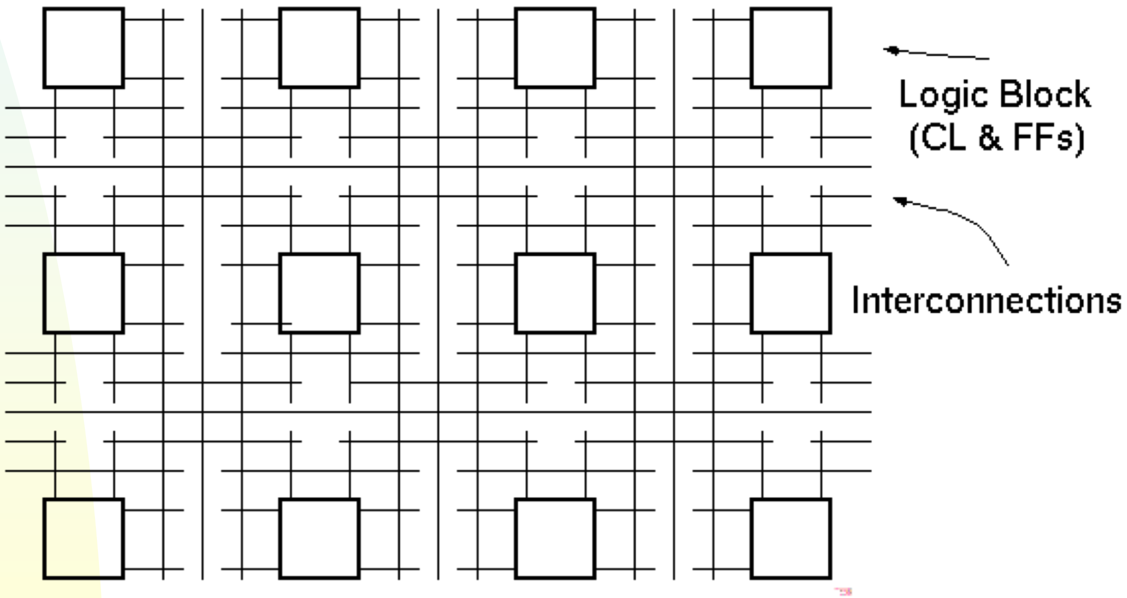


Programmable logic

Programmable logic

Programmable Logic

- **Basic idea**: an array of general-purpose logic and registers
- The user can configure:
    1. The function of each logic block
    2. Connections between logic blocks



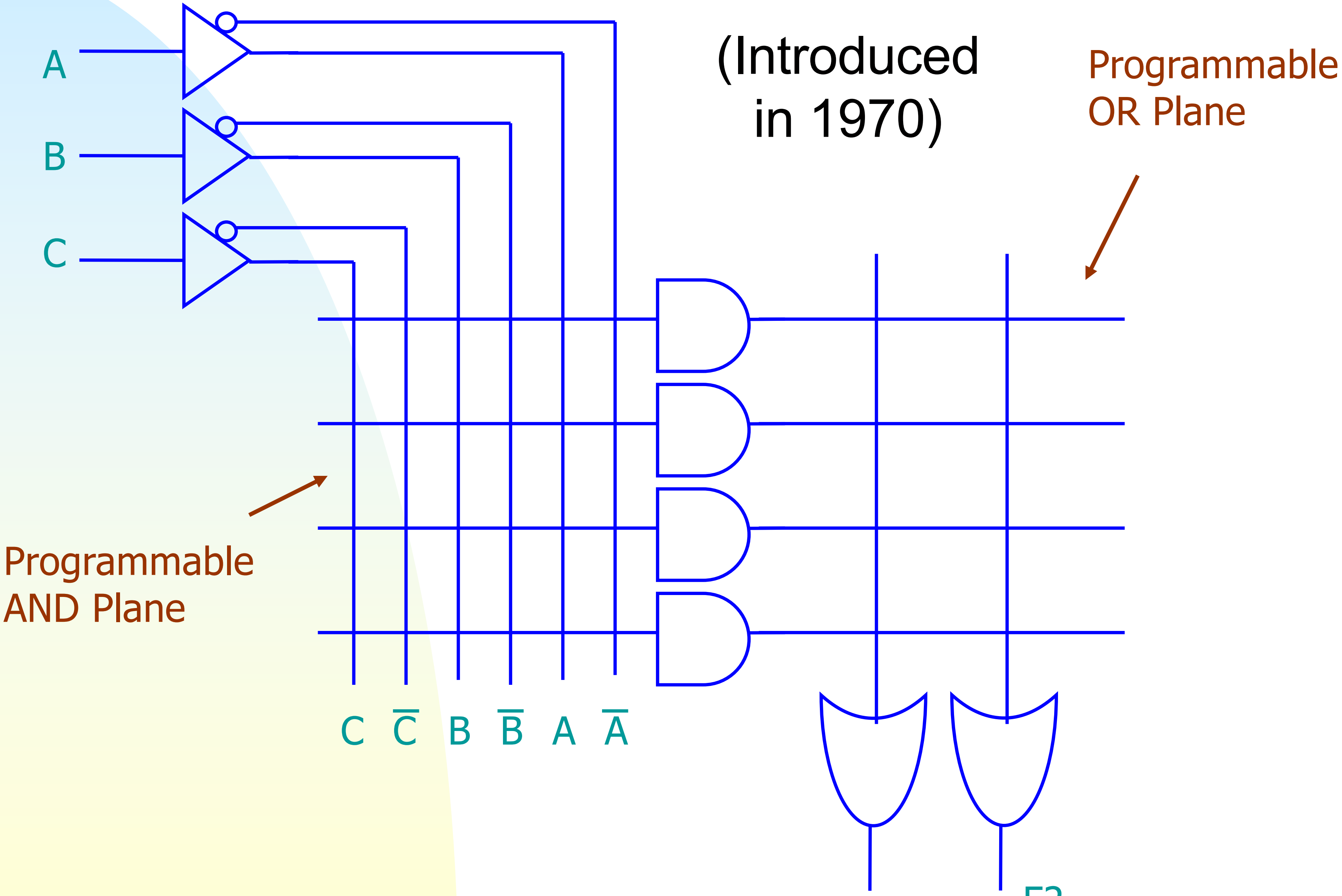
Combinational PLDs

(Historical…)

- Program Logic Array (PLA):
  - Programmable AND and OR arrays
  - More flexible than PAL.
- Programmable Array Logic (PAL):
  - Programmable array of AND gates
  - Fixed array of OR gates
  - Configure to producing AND-OR sums
- Programmable memory (PROM)
  - Implemented with fixed AND array and programmable OR array

Programmable Logic Array (PLA)

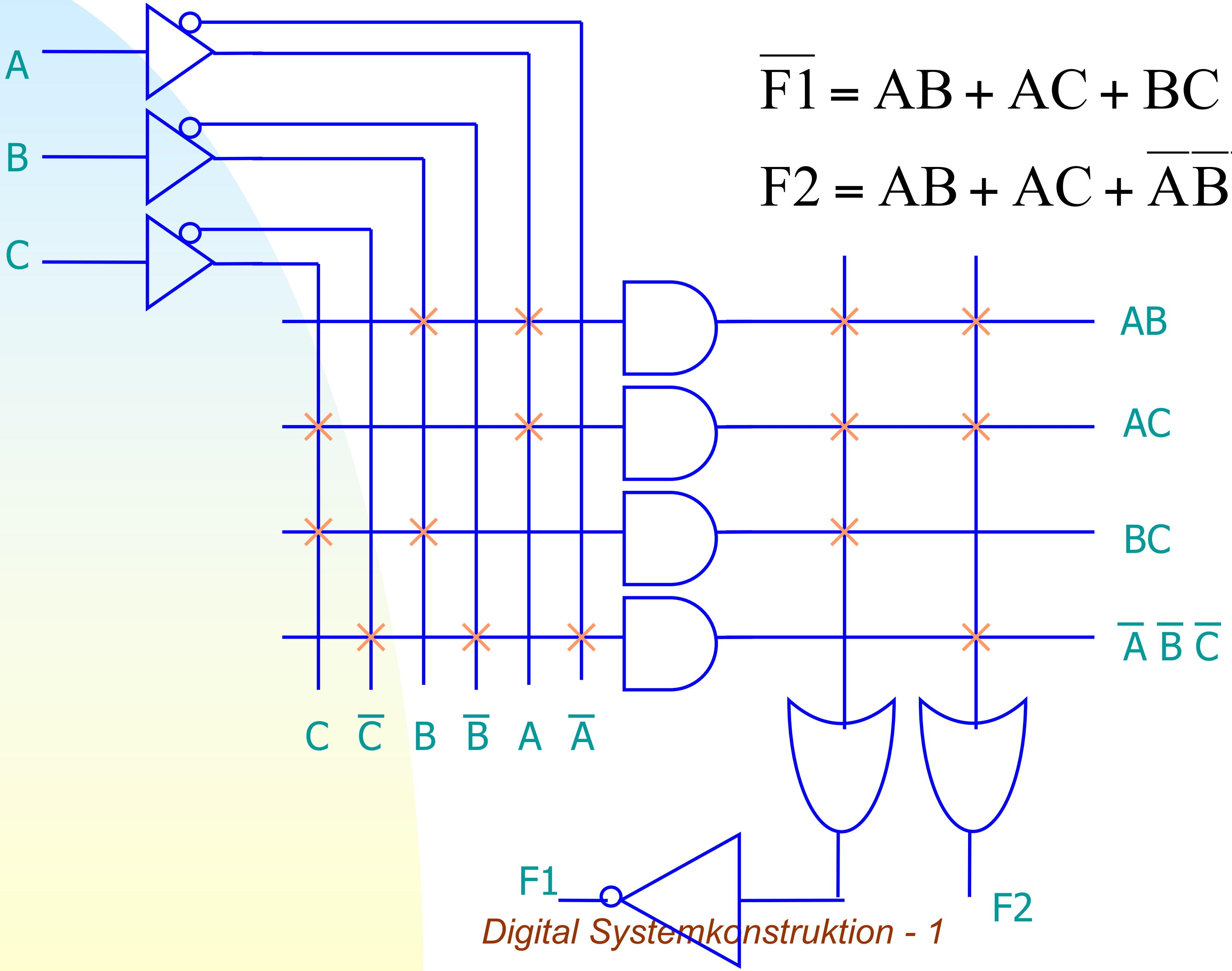
(Introduced in 1970)



PLA example

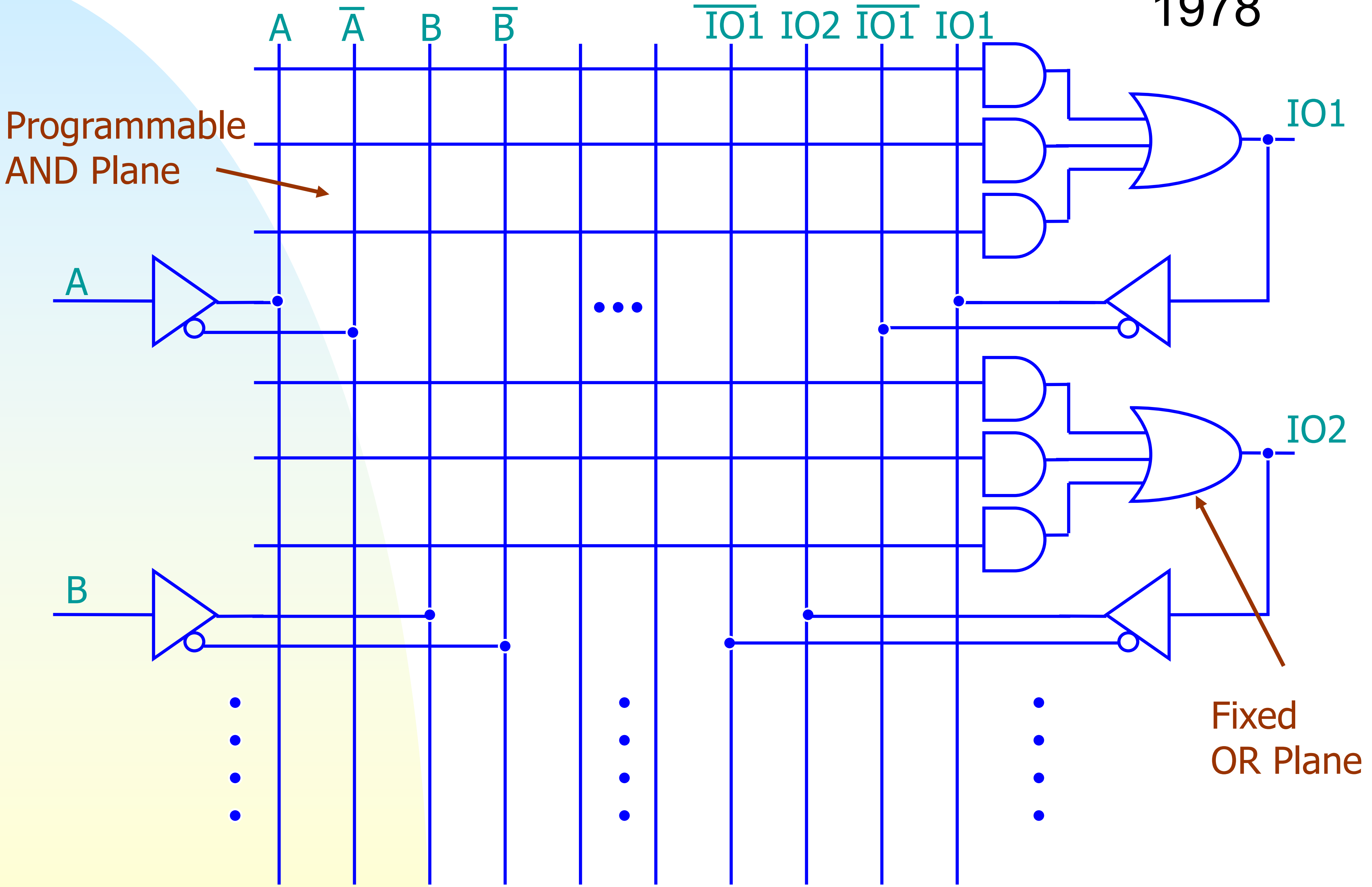
$$\overline{F1} = AB + AC + BC$$

$$F2 = AB + AC + \overline{A}\,\overline{B}\,\overline{C}$$

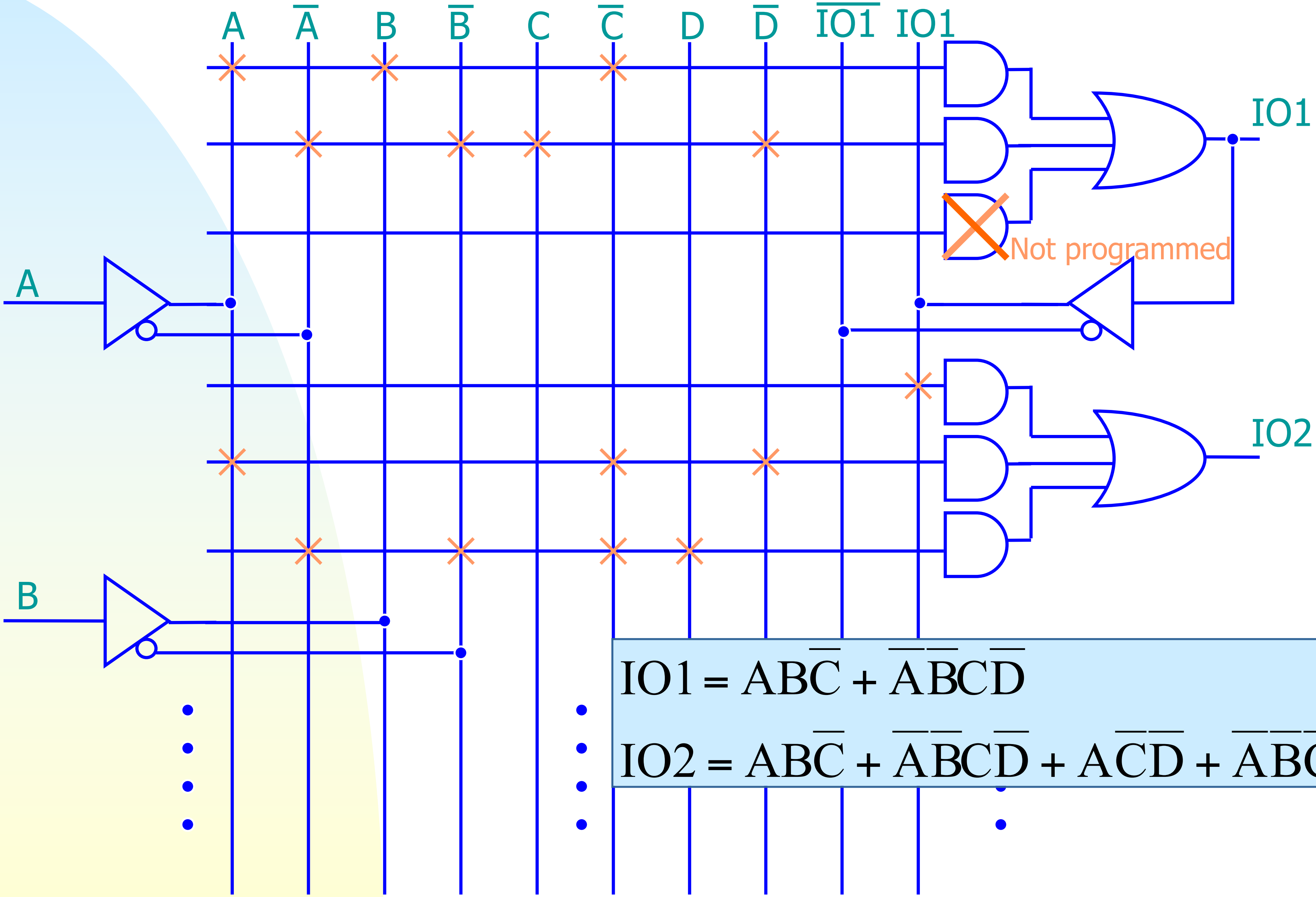


PAL Device

Introduced in 1978



PAL Device Design Example



$$IO1 = AB\bar{C} + \bar{A}\bar{B}C\bar{D}$$

$$IO2 = AB\bar{C} + \bar{A}\bar{B}C\bar{D} + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$$

Sequential devices

- Complex Programmable Logic Device (CPLD)
  - Multiple PLDs (e.g. PALs, PLAs) on same device
  - Programmable interconnects and registers
- Field-Programmable Gate Array (FPGA)
  - Lots of logic and large, distributed interconnect structure
  - Each logic block has fewer inputs than in a CPLD
    - But FPGAs contain many more logic blocks
  - Higher register/logic ratio than CPLDs
    - Especially suitable for sequential designs

Sequential PLD

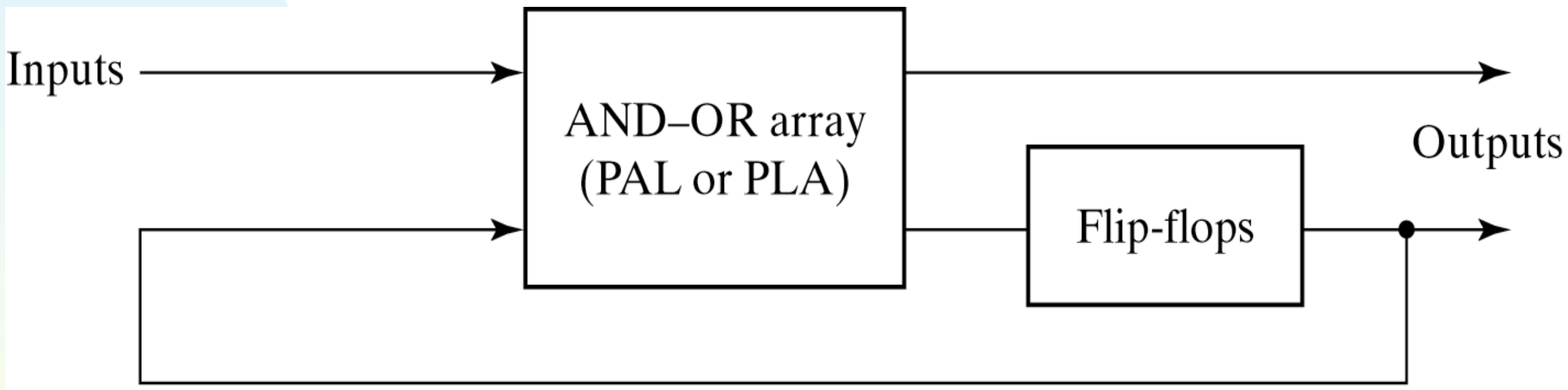


Fig. 7-18 Sequential Programmable Logic Device

Example PLD Macrocell

PLA-type logic combined with a flip-flop for sequential designs

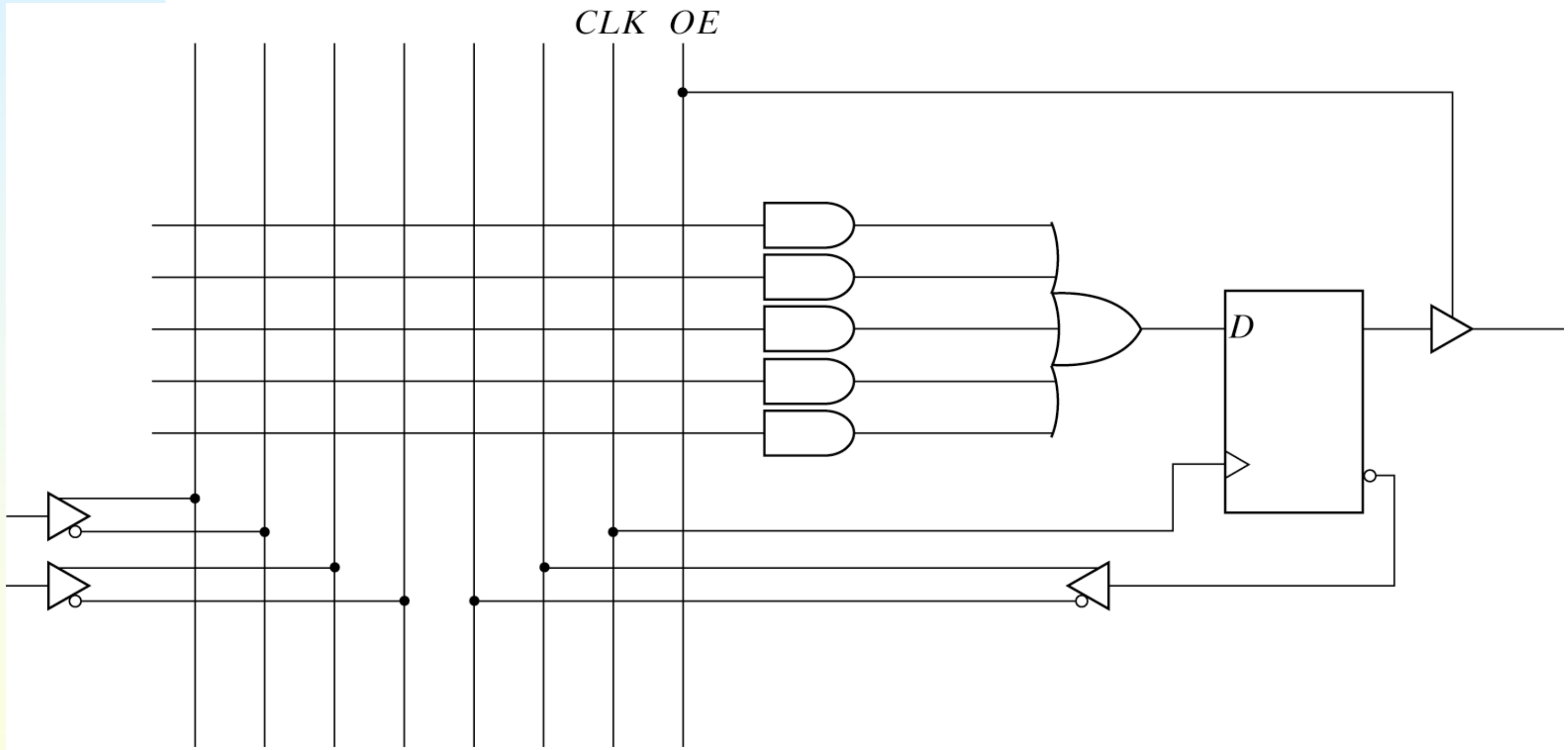
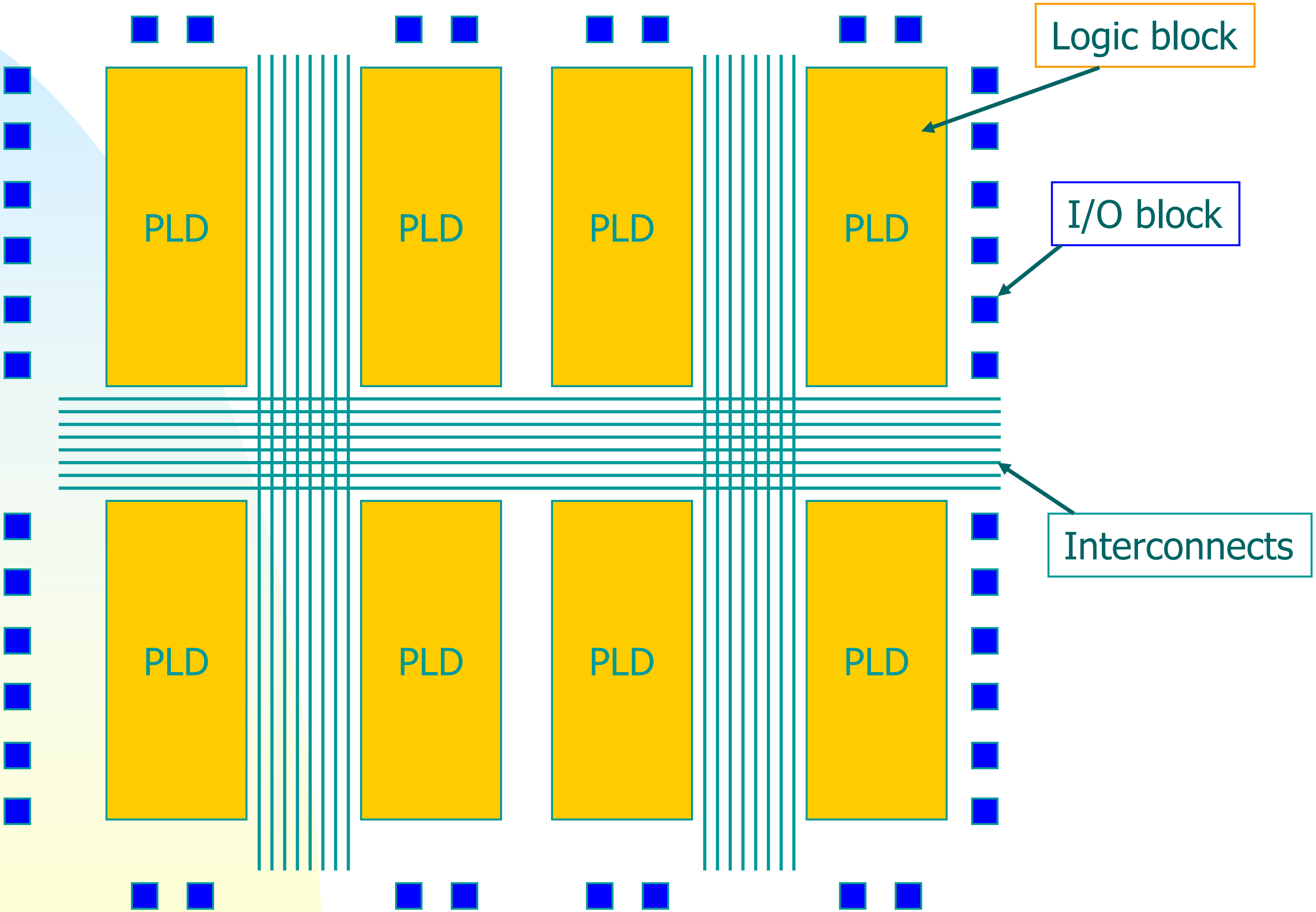
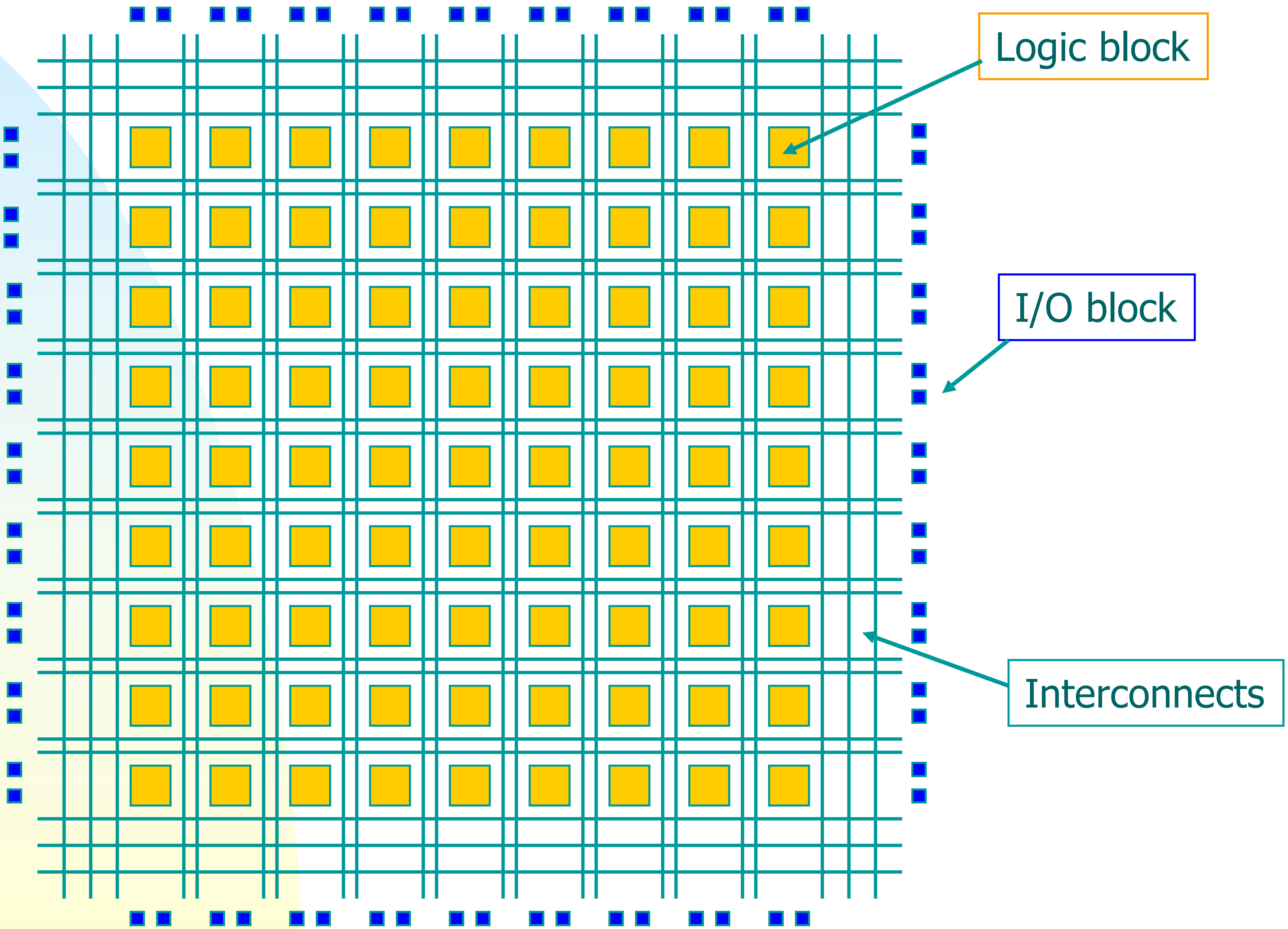


Fig. 7-19 Basic Macrocell Logic

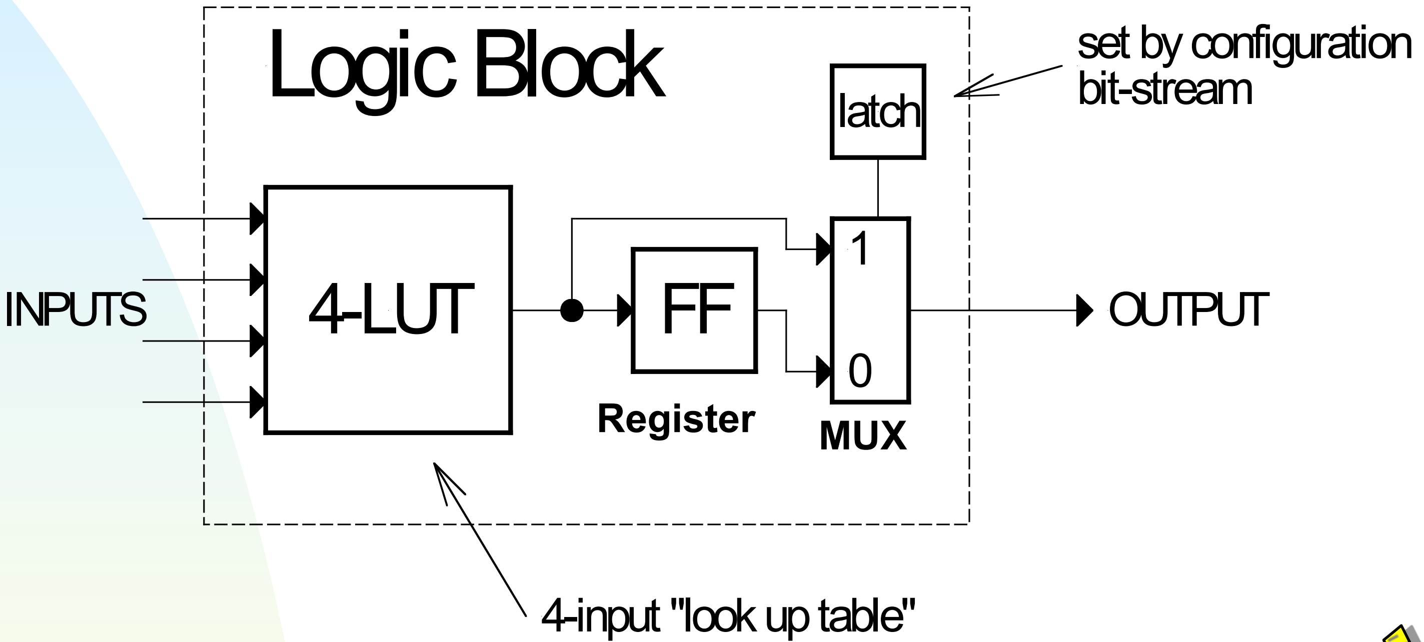
CPLD structure



FPGA Structure



FPGA Logic Block (idealized)



- 4-input *look up table (LUT)*
  - implements combinatorial logic
- Register
  - optionally stores output of LUT

LUT as general logic gate

Example: 2-lut

INPUTS	AND	OR
00	0	0
01	0	1
10	0	1
11	1	1

. . .

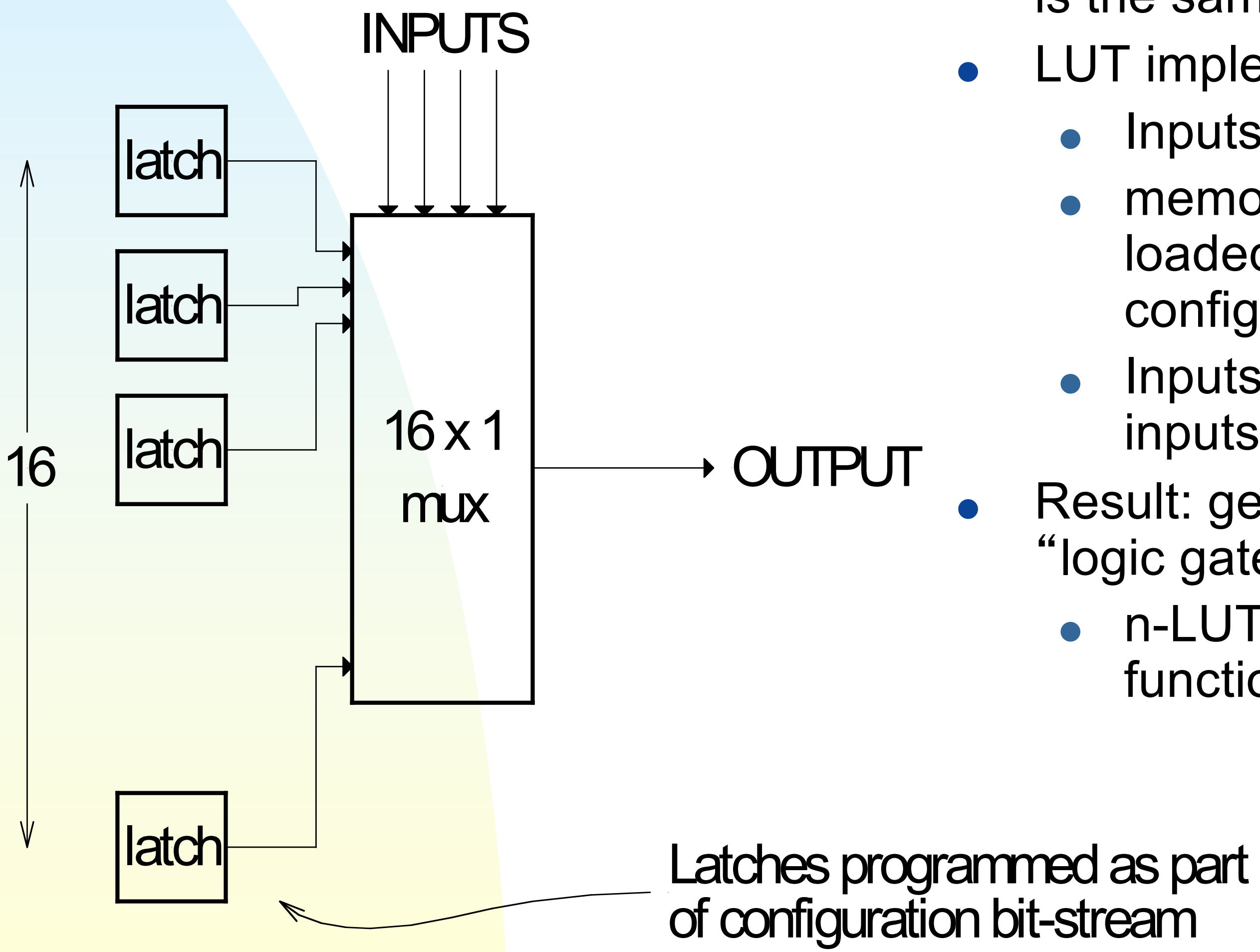
Implements *any* function of 2 inputs.

- LUT programmed as a **truth-table**.
- Each latch stores the programmed function output for one set of inputs

Example: 4-lut

INPUTS		
0000	F(0,0,0,0)	← store in 1st latch
0001	F(0,0,0,1)	← store in 2nd latch
0010	F(0,0,1,0)	←
0011	F(0,0,1,1)	←
0011		
0100	⋮	
0101		
0110		
0111		
1000		
1001		
1010		
1011		
1100		
1101		
1110		
1111		

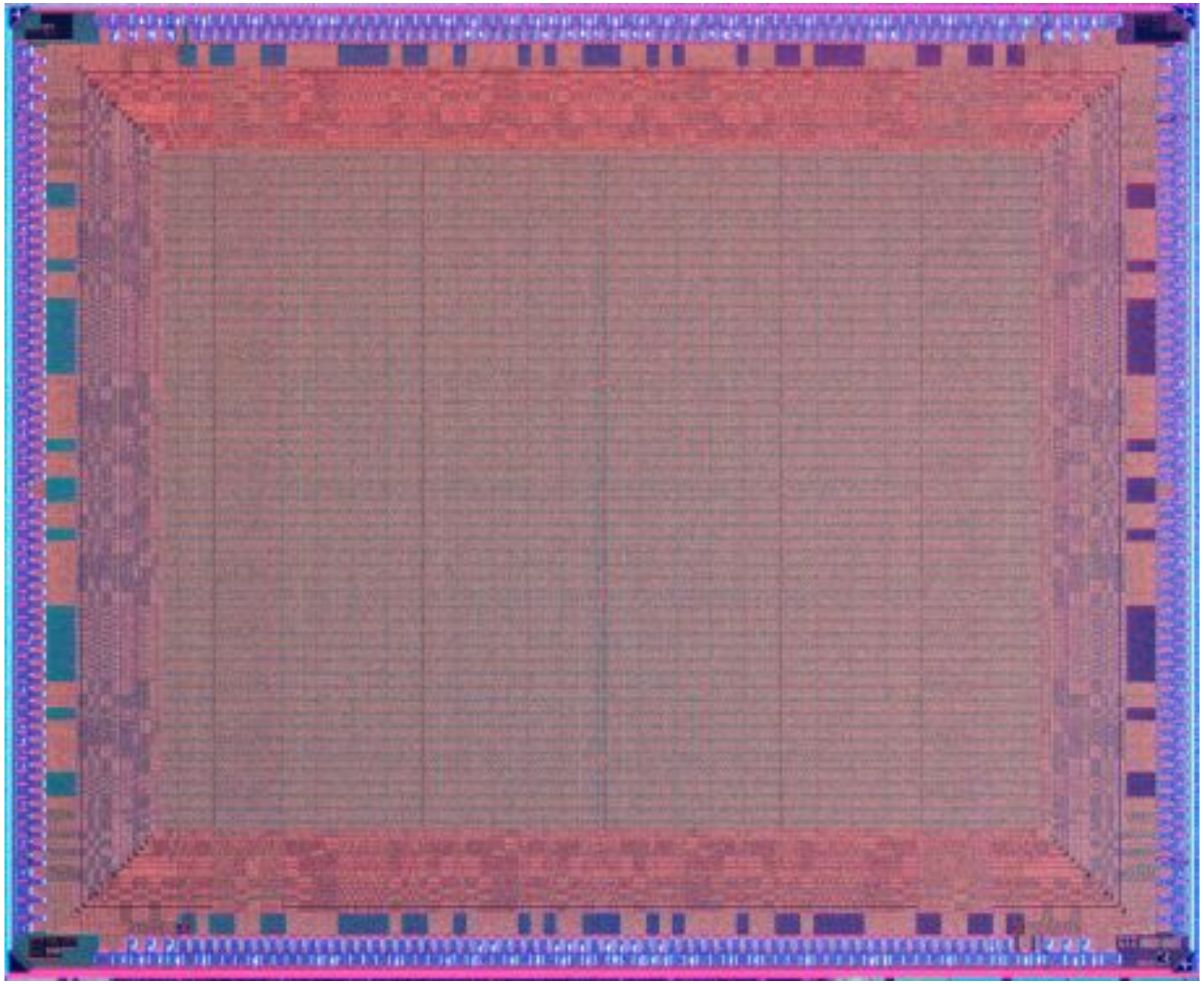
4-LUT Implementation



- Newer FPGAs use 6-LUTs, but idea is the same
- LUT implemented as 2^n x 1 memory:
  - Inputs select memory location.
  - memory contents (latches) are loaded with values from the configuration bit stream.
  - Inputs to MUX are the CLB inputs.
- Result: general purpose "logic gate":
  - n-LUT can implement *any* function of n inputs

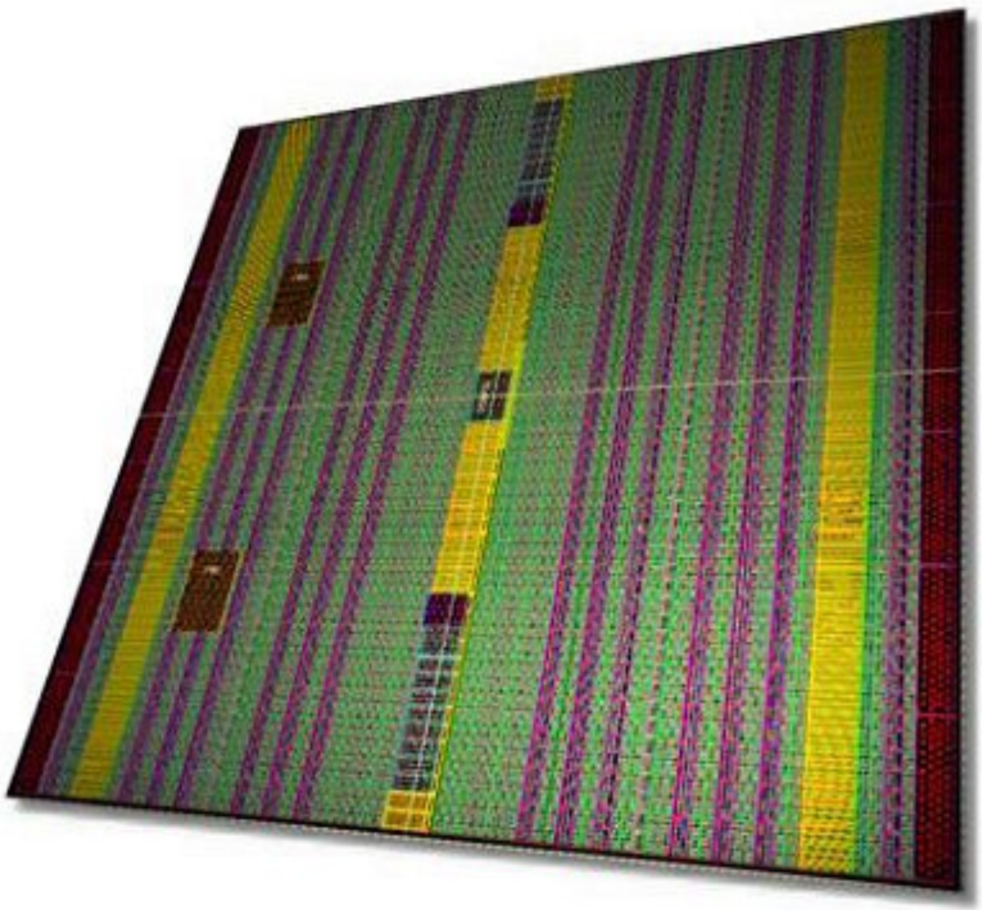
Field-Programmable Gate Arrays

- Xilinx Spartan-3 die image. Note the regularity



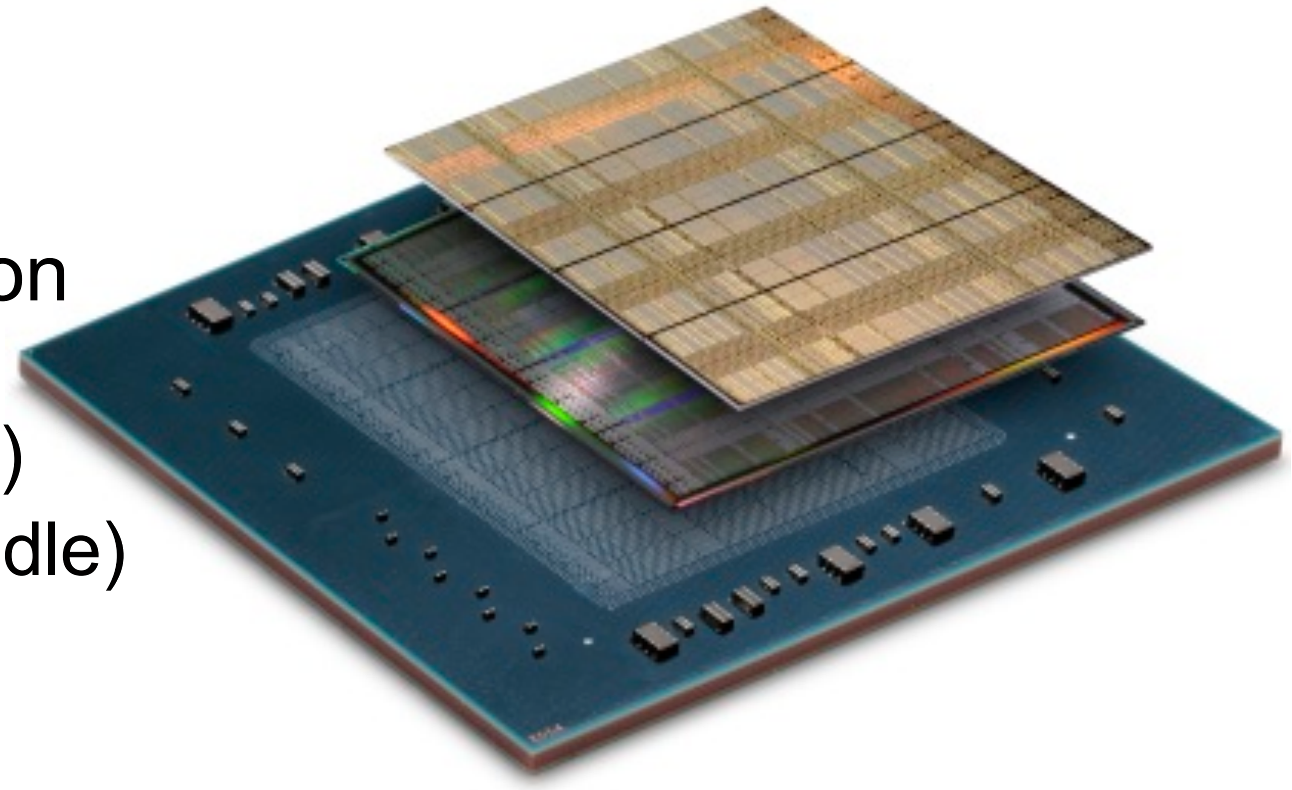
More modern FPGA

- Xilinx Virtex 4
- Different areas:
  - Logic
  - Memory
  - Embedded processors
  - I/O



Xilinx Virtex 7

- Multi-chip integration
  - Packaging substrate (bottom)
  - Si interposer (middle)
  - FPGA logic (top)

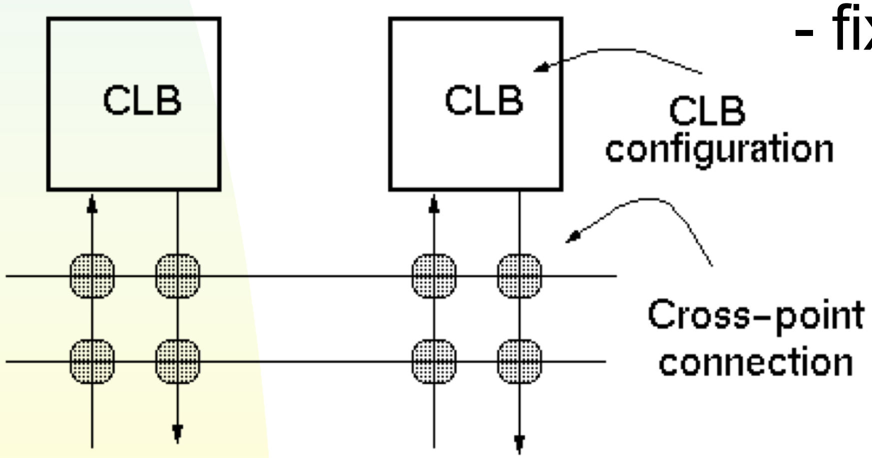


FPGAs can be reconfigured

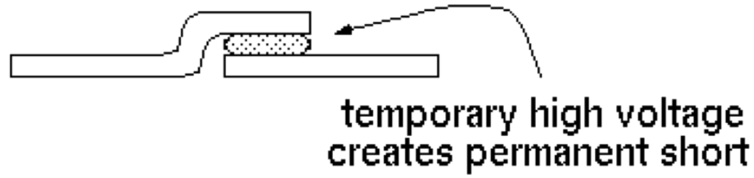
- Fix errors by simply loading a new configuration
  - Design cycles take days or weeks (not months)
- Hardware can be upgraded in the field
- Can make multi-purpose hardware
  - Use same circuit boards loaded with different FPGA configurations
- Newer FPGAs even have dynamic partial reconfiguration
  - Add/change functionality while running!

FPGA Variations

- Variations include:
  - Physical programming method
  - Arrangement of interconnects
  - Functionality of logic blocks.



- Example: Anti-fuse based (ex: Actel)

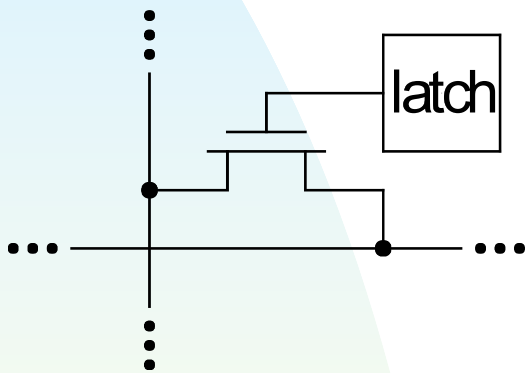


+ Non-volatile, relatively small

- fixed (non-reprogrammable)

Field Programmable devices

- Latch-based (Xilinx, Altera, …)



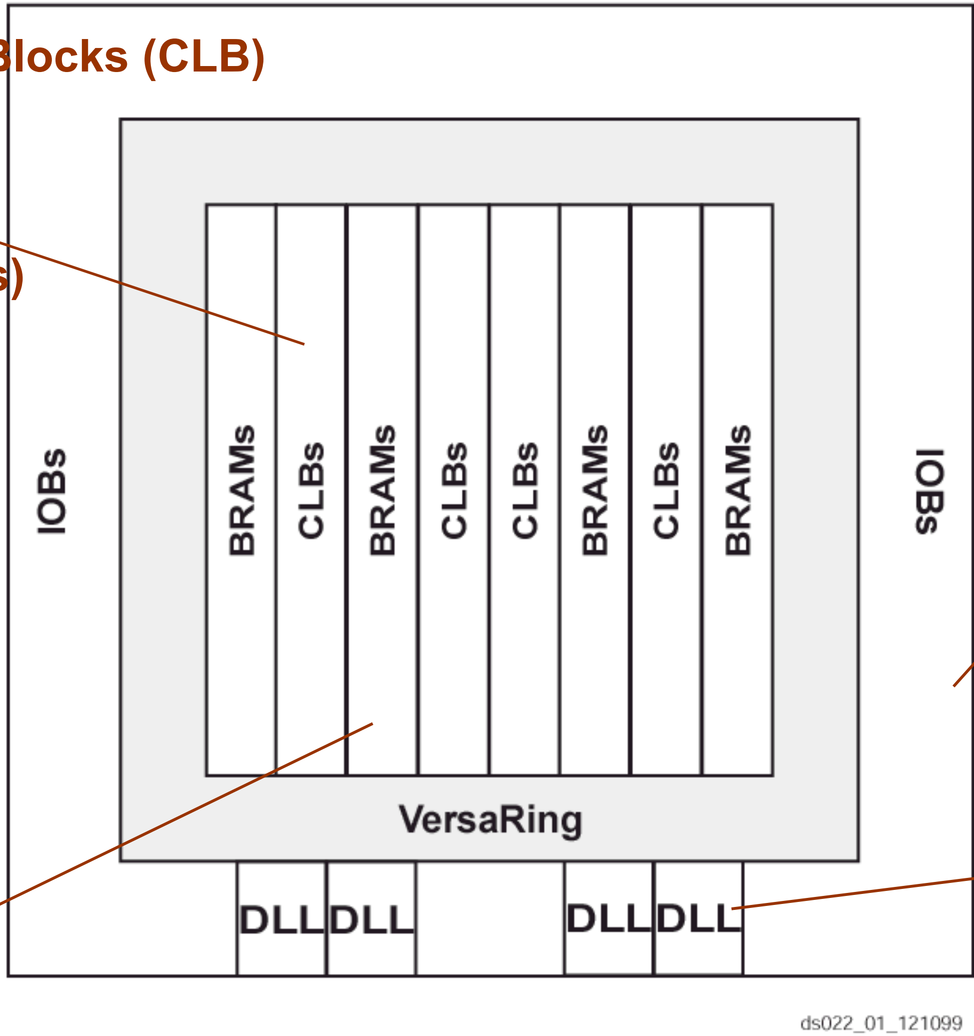
  + Reconfigurable!
  - volatile
  - relatively large.

- Latches used to:
  1. Make or break connections
  2. Program logic blocks
  3. Set user options:
     - within the logic blocks
     - inputs/outputs
     - global reset/clock
- "Configuration bit stream" loaded under user control
  - All latches tied together in a shift chain
  - FPGA active after a full configuration has been successfully loaded

Xilinx Virtex-E Floorplan

Configurable Logic Blocks (CLB)

- **4-input function generators**
- **Registers (flip-flops)**



Input/Output Blocks

- **combinational, latch, and registered output**
- **clocked or unclocked inputs**

Clock signal conditioning and synthesis

Block RAM

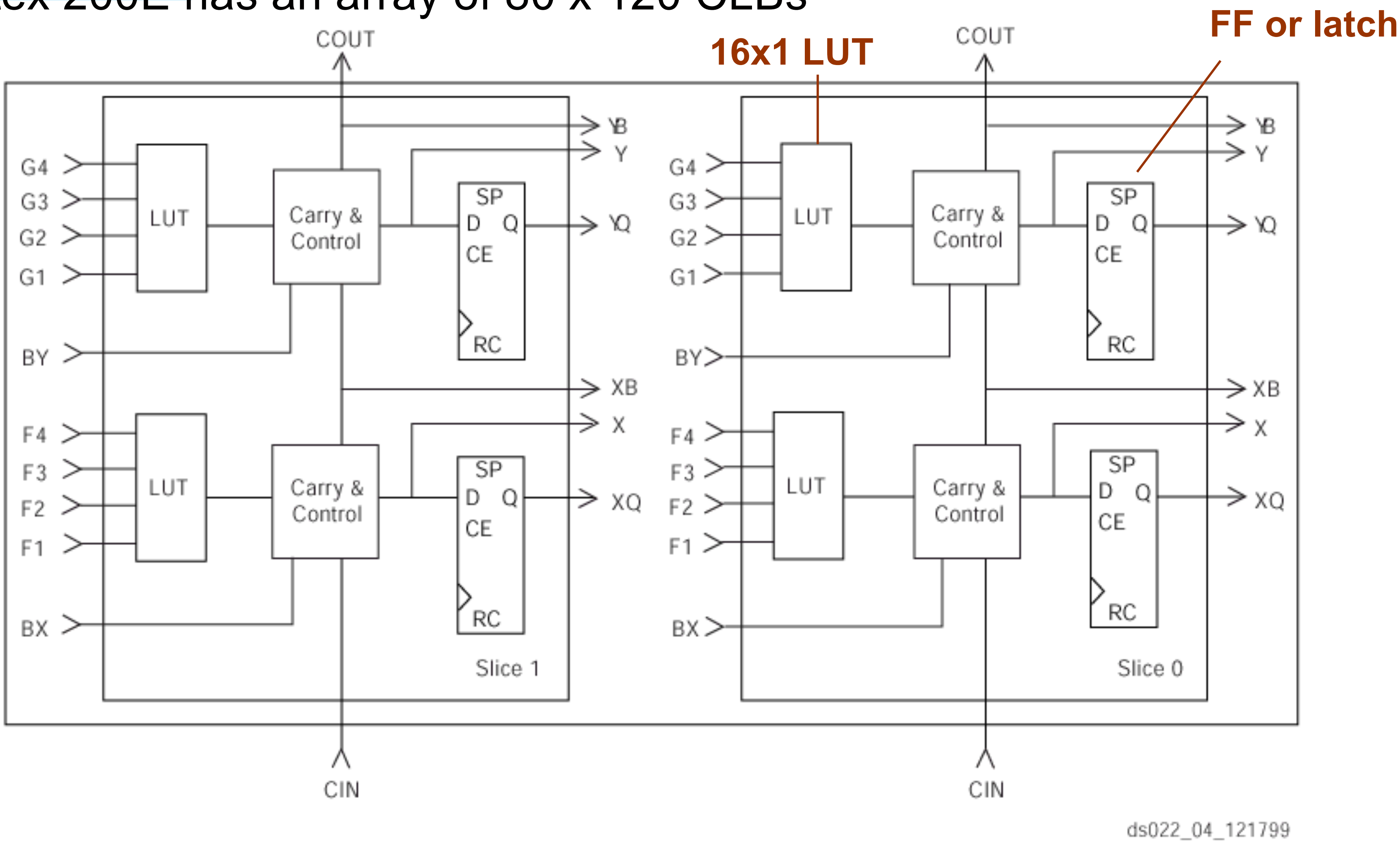
- **4096 bits each**

Virtex-E CLB

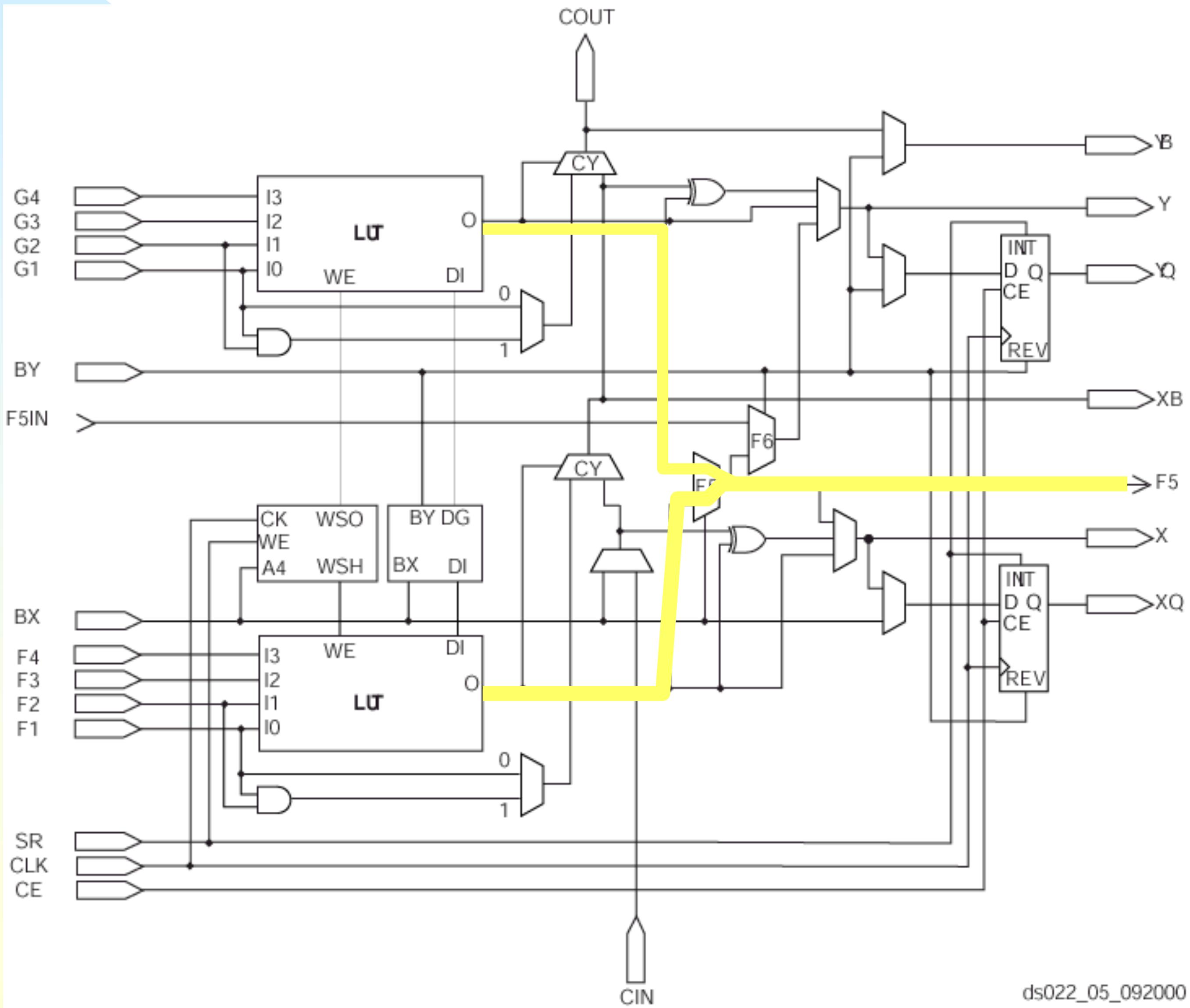
4 logic cells (LC) in two “slices”

LC contains: 4-input function generator, carry logic, storage element

Virtex 200E has an array of 80 x 120 CLBs



Details of Virtex-E Slice



LUT

- •4-input function, or
- •16x1 SRAM, or
- •32x1 or 16x2 / slice, or
- •16 bit shift register

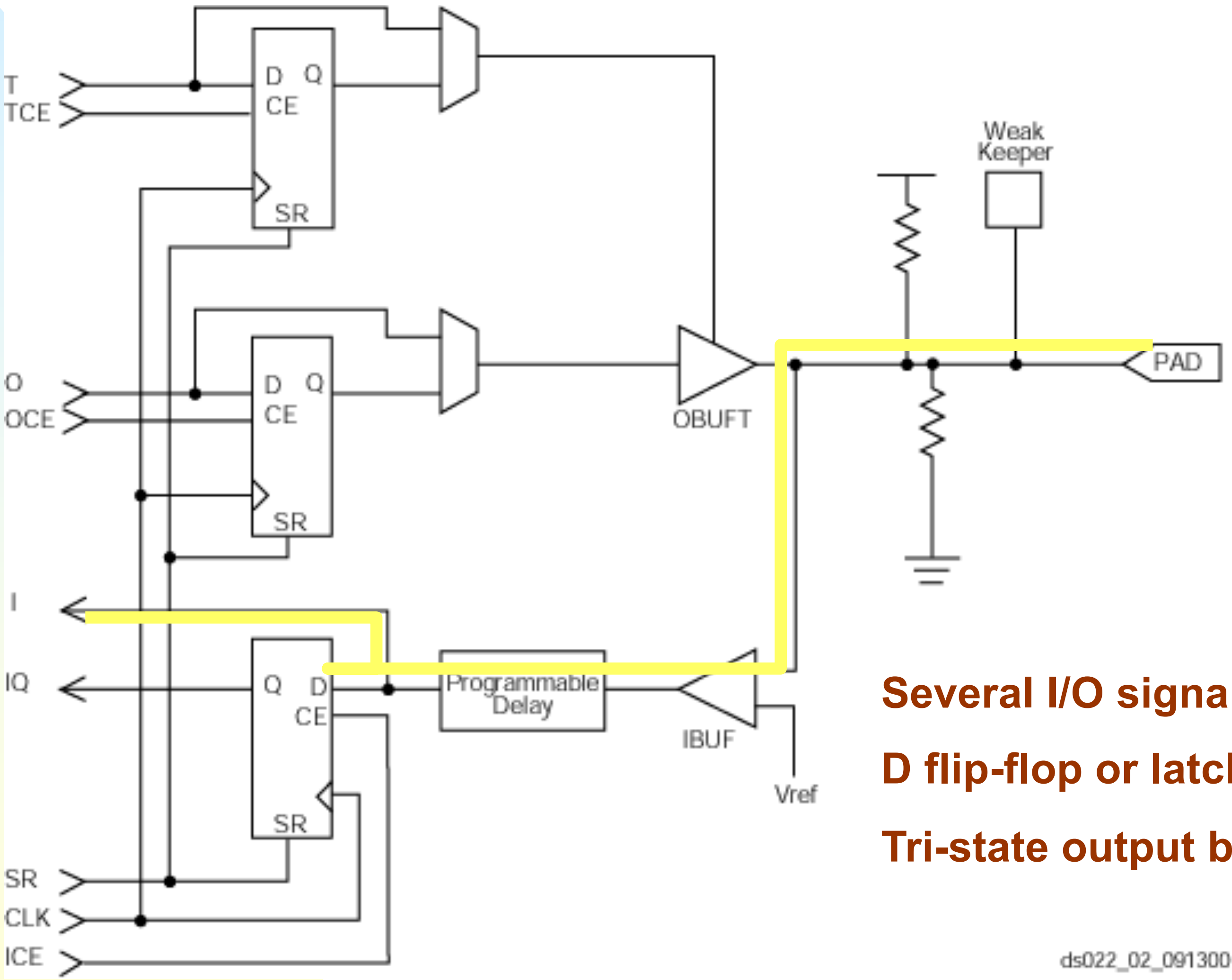
Storage element

- D flip-flop, or
- Latch

Carry chain

- Arithmetic along row or column

Virtex-E
Input/Output block (IOB)

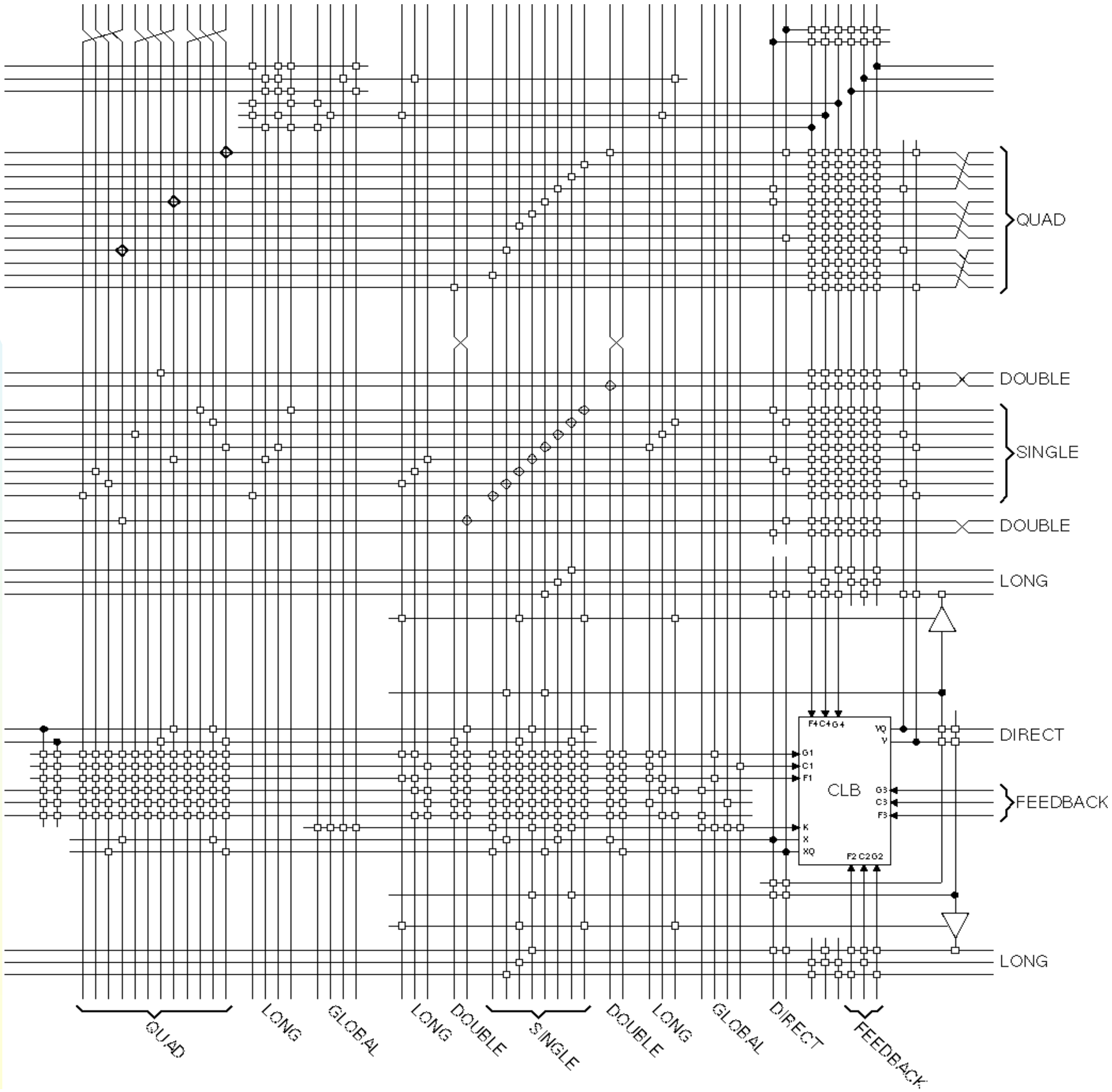


Several I/O signaling standards

D flip-flop or latch

Tri-state output buffer

FPGA interconnect



Other FPGA features:

- Dedicated block memories
  - Dual-port static RAM
- Digital clock management/synthesis
- Dedicated multipliers
  - Important for digital signal processing
- "Hard" CPU cores
  - Xilinx: ARM 9 (Zynq)
  - Altera: Nios II (Stratix)
- Multi-Gb/s transceivers

Next lecture:

- FPGA design flow
- Introduction to the VHDL hardware description language