Sequential Building Blocks

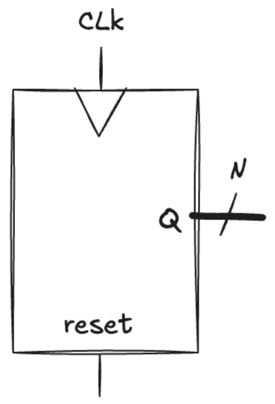
Index

Building Blocks

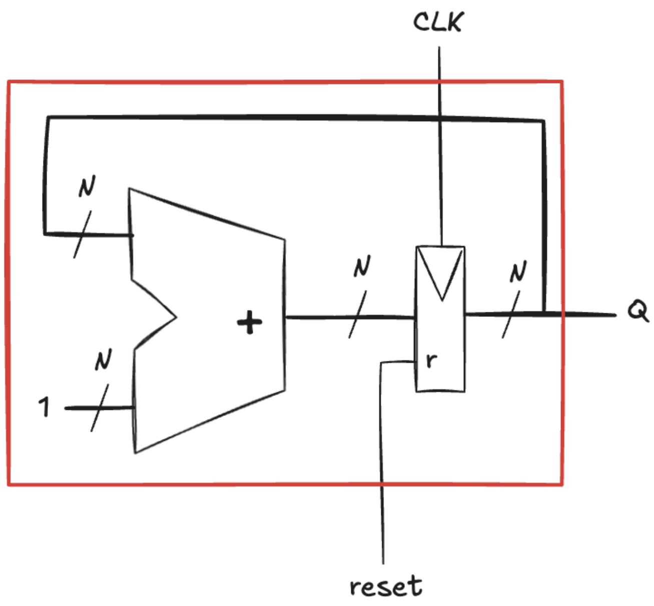
- Like Combinational logic, also Sequential logic is often grouped into **larger building blocks** to build more complex systems
  - principles of **abstraction, hierarchy, modularity, and regularity**
  - hiding the unnecessary gate-level details to emphasize the function of the building block
  - hierarchically assembled from simpler components
  - a well-defined interface and can be treated as a black box
- Examples:
  - Counters
  - Registers
  - Memory
  - PLA
  - FPGA
- We will use many of these building blocks to build a microprocessor

Counters

- A device that advances through all 2^N possible outputs in binary order
  - incrementing on the rising edge of the clock
  - Reset initializes the output to 0



- A possible implementation composed of an adder and a resettable register
  - on each cycle, the counter adds 1 to the value stored in the register
  - the most significant bit toggles every 2^N cycles



- It can be used to reduces the frequency of the clock by a factor of 2^N
  - useful for slowing down fast signals
  - a digital system with a 50MHz clock, can be slowed with a 24-bit counter to produce a 2.98Hz signal that blinks a LED at a rate the human eye can observe

Digitally Controlled Oscillator (DCO)

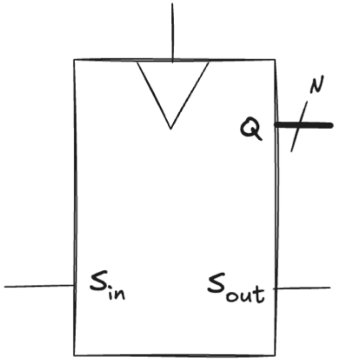
- A counter generalization to produce arbitrary frequencies

- A N-bit counter that adds p on each cycle, rather than 1
  - if the clock has frequency f_{clk}, the most significant bit now toggles at $f_{out} = f_{clk} * p/2^N$

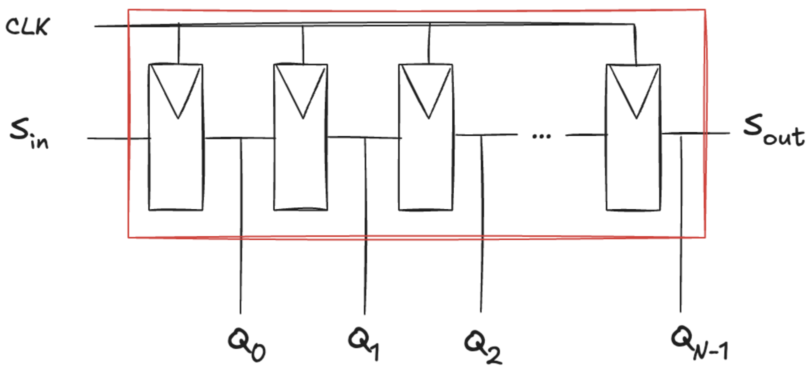
- Selecting p and N, we can produce an output of any frequency
  - larger N gives more precise control at the expense of more hardware

- Suppose we have a 50 MHz clock and want to produce a 500 Hz output
  - consider using an N=24 or 32 bit counter
  - we want $p/2^N$ = 500Hz/50MHz = 0.00001
  - if N=24, choose p=168 to get f_{out}=500.68Hz
  - if N=32, choose p=42950 to get f_{out}=500.038 Hz

Shift Registers (1)



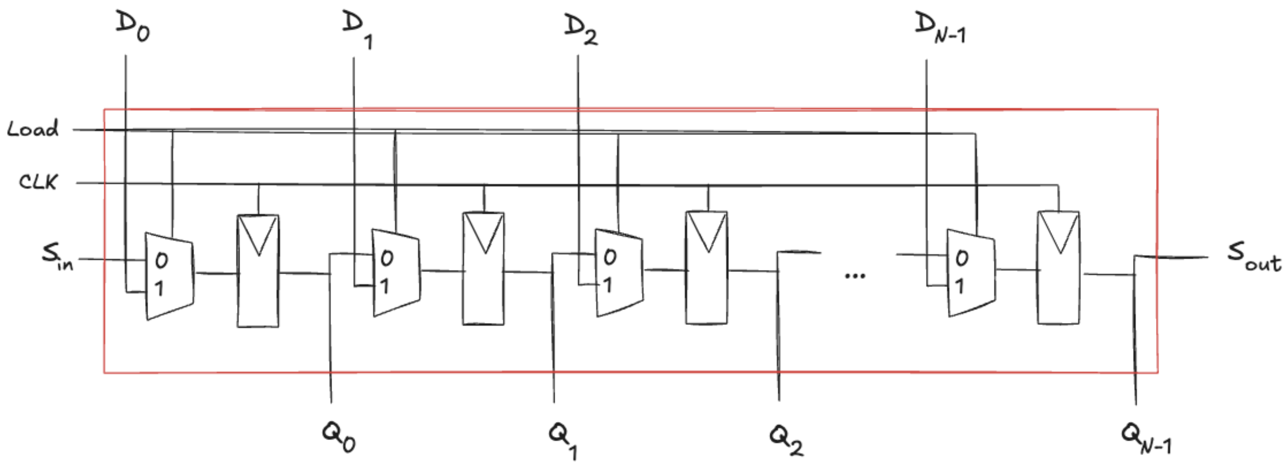
- A device that shift in a new bit on each clock edge
  - on each rising edge of the clock, a new bit is shifted in from S_{in} and all the subsequent contents are shifted forward
  - the last bit in the shift register is available at S_{out}
- It can be constructed from N flip-flops connected in series



- Don't confuse shift registers with shifters
  - shifters are un-clocked combinational blocks that shift an input by a specified amount
- Can be viewed as **serial-to-parallel converter**
  - the input is provided serially (one bit at a time) at S_{in}
  - after N cycles, the past N inputs are available in parallel at Q

Shift Registers (2)

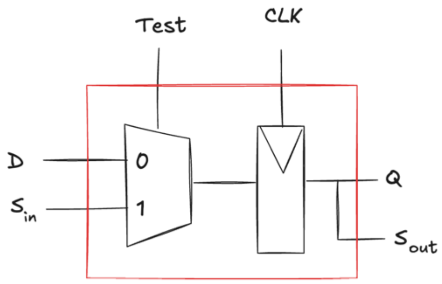
- A related circuit is a **parallel-to-serial converter**
  - it loads N bits in parallel
  - then shifts them out one at a time
- A shift register can be modified to perform both serial-to-parallel and parallel-to-serial operations by adding a parallel input $D_{N-1:0}$ and a control signal

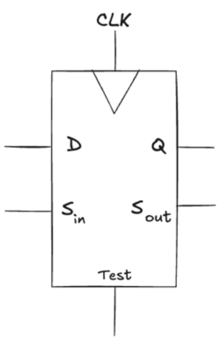


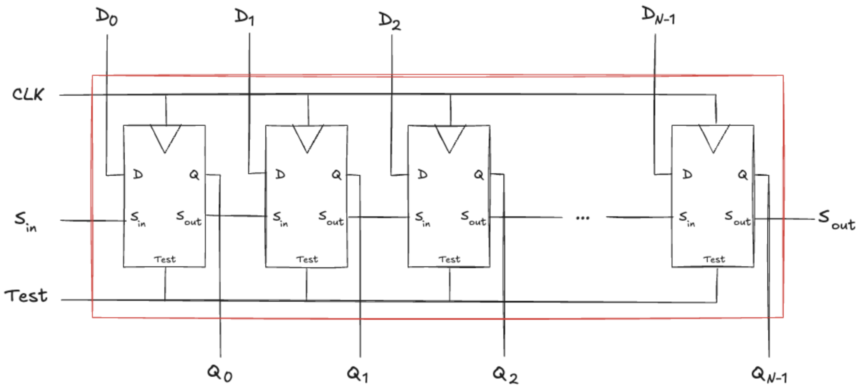
  - when Load is asserted, the flip-flops are loaded in parallel from the D inputs
  - otherwise, the shift register shifts normally

Scan Chains

- Testing combinational circuits is relatively straightforward
  - known inputs (**test vectors**) are applied, and outputs are checked
- Testing sequential circuits is more difficult because the circuits have state
  - many cycles of test vectors may be needed to put the circuit into a state
  - testing that the msb of a 32-bit counter requires 2^{31} clock pulses!
- Directly control the state, flip-flops connected into a shift register (**scan chain**)
  - in normal operation, flip-flops load data from their input D
  - in test mode, flip-flops serially shift contents using S_{in} and S_{out}







- The example:
  - 32-bit counter can be tested by shifting in the pattern 011111...111 in test mode, counting for one cycle, then shifting out the result, which should be 100000...000
  - this requires 32 + 1 + 32 = 65 cycles

Memory (1)

- Digital systems require **memories** to store data used and generated by circuits
  - registers are a kind of memory that stores small amounts of data
- Memory arrays can efficiently store **large amounts of data**
- Memories are classified based on how they store bits
  - Random Access Memory (**RAM**)
    - **volatile**: loses data when the power is turned off
  - Read Only Memory (**ROM**)
    - **non-volatile**: retains its data indefinitely, even without a power source
- Names for **historical reasons**: no longer very meaningful
  - RAM accesses data with the same delay, in contrast with sequential access memory (e.g. tape recorder accesses nearby data more quickly than faraway data)
  - ROM historically could only be read, but not written
- These names are **confusing**
  - ROMs are also randomly accessed...
  - worse yet, modern ROMs can be written as well as read...
- Just one important distinction: volatile and non-volatile

Memory (2)

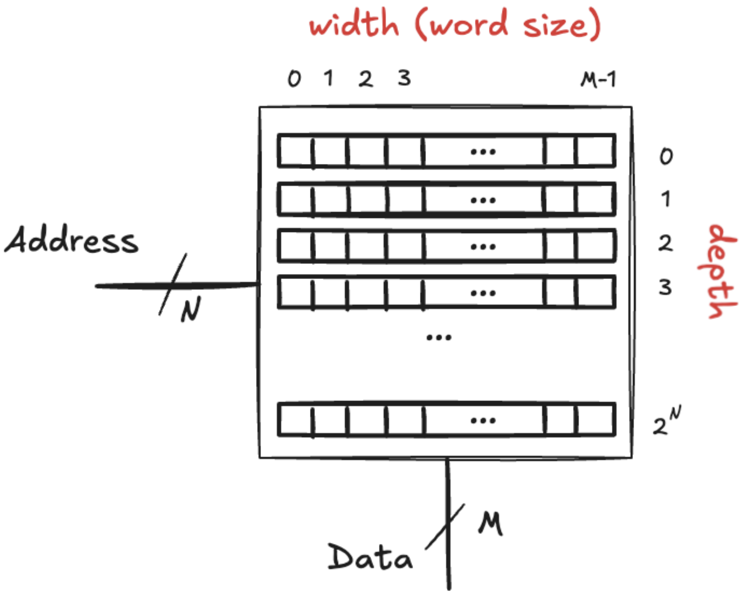
IBM 305 RAMAC system: processing unit, magnetic process drum, magnetic core register, electronic logical and arithmetic circuits (1956)

It's 5MB Hard Drive

3.200$ (today 30.874$)...
..per month!

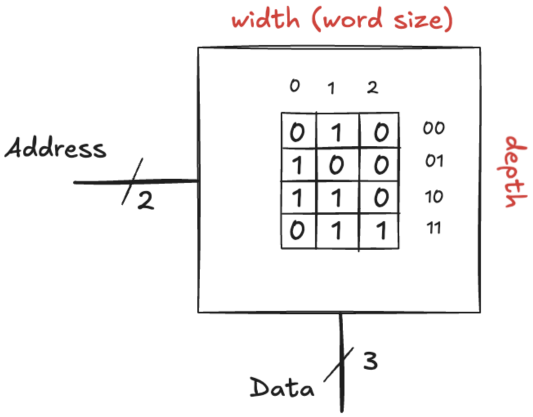
Memory Arrays

- Memory is organized as a two-dimensional array of memory cells
  - row is specified by an **address**
    - **depth** of an array is the number of rows
  - value read or written is called **data**
    - **width** is the number of columns, also called the **word size**

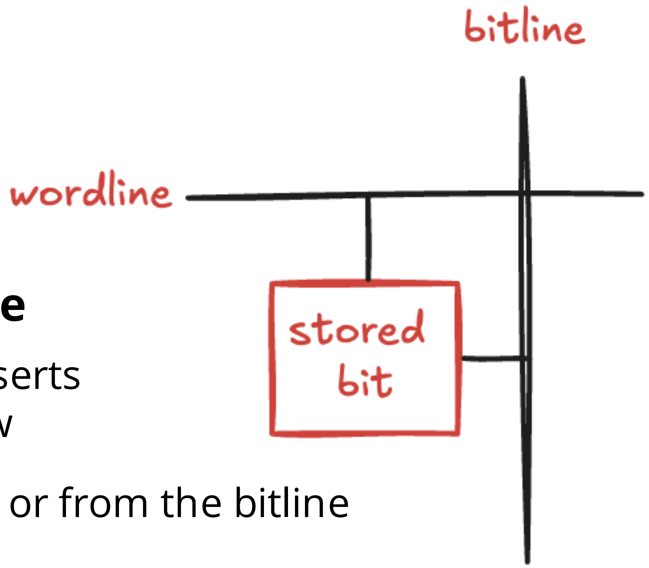


- Memory with N-bit address and M-bit data has 2^N rows and M columns
  - Memory size is given as **depth * width**

- Example:
  - a memory array with two address bits and three data bits
  - the two address bits specify one of the four rows (data words)
  - each data word is three bits wide
  - it is a 4-word * 3-bit array, or simply 4 × 3 memory



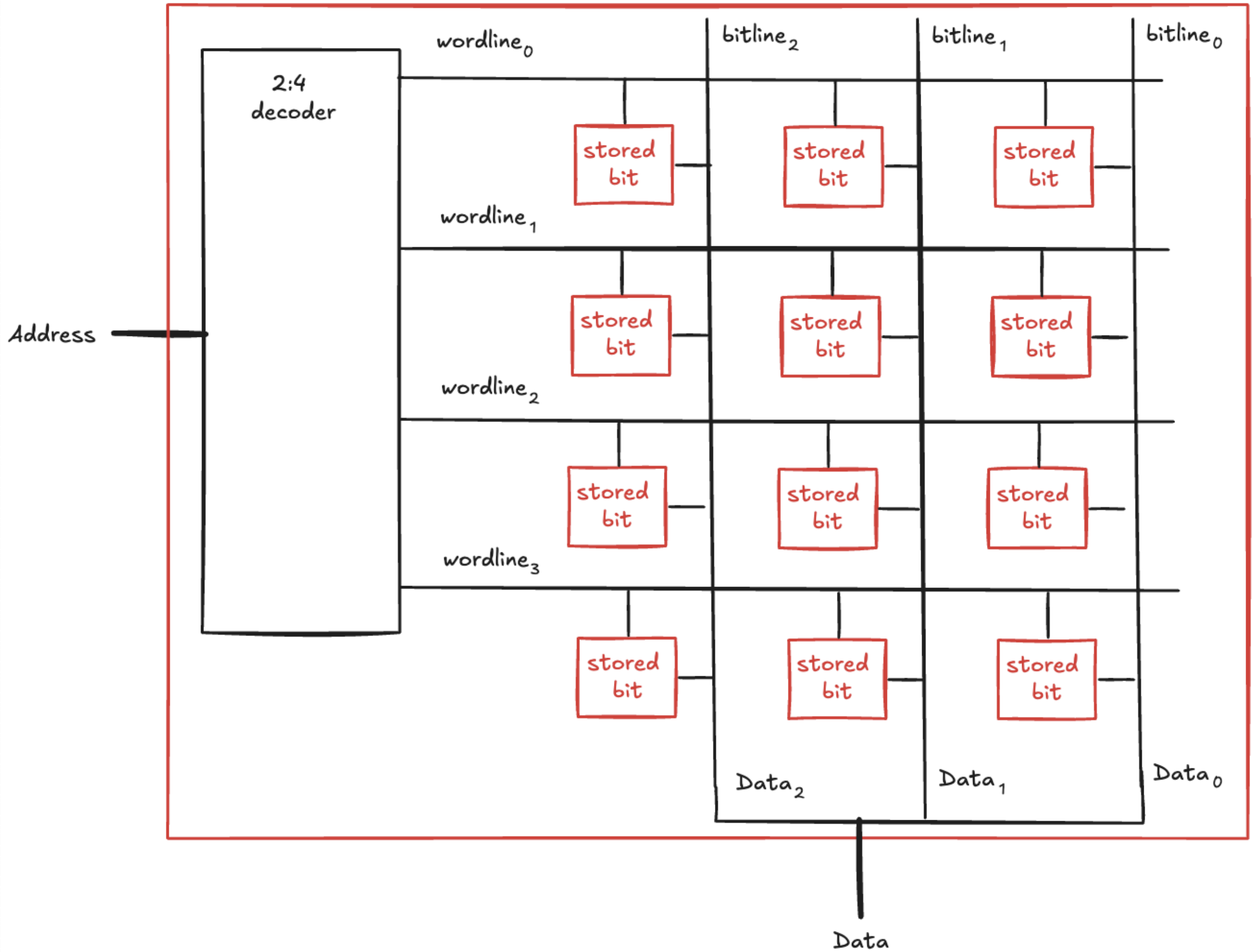
Bit Cell (1)



- Memory arrays are built as an array of **bit cells**
  - each of which stores 1 bit of data
- Each bit cell is connected to a **wordline** and a **bitline**
  - for each combination of address bits, the memory asserts a single wordline that activates the bit cells in that row
  - when the wordline is HIGH, the stored bit transfers to or from the bitline
  - otherwise, the bitline is disconnected from the bit cell
- The circuitry to store the bit varies with memory type
- To **read** a bit cell
  - the bitline is left floating (Z)
  - then the wordline is turned ON, allowing the stored value to drive the bitline
- To **write** a bit cell
  - the bitline is strongly driven to the desired value
  - then, the wordline is turned ON, connecting the bitline to the stored bit
  - the strongly driven bitline overpowers the contents of the bit cell, writing the desired value into the stored bit

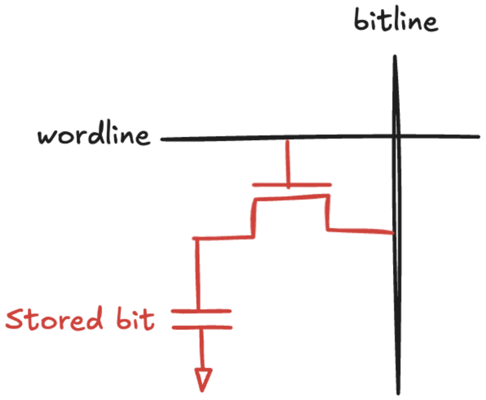
Bit Cell (2)

- The figure shows the internal organization of a 4 * 3 memory array



Dynamic Random Access Memory (DRAM)

- DRAM stores data as a charge on a **capacitor**
  - transistor behaves as a switch
  - connects or disconnects the capacitor from the bitline
    - when the capacitor is charged to VDD, the stored bit is 1
    - when it is discharged to GND, the stored bit is 0



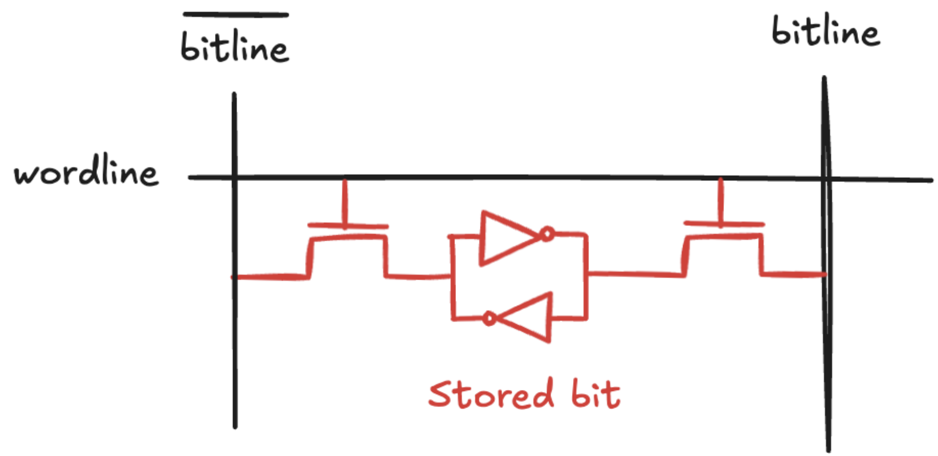
- Upon a write, data values are transferred from the bitline to the capacitor
- Upon a read, data are transferred from the capacitor to the bitline

- Reading **destroys** the bit stored on the capacitor
  - data must be restored (rewritten) after each read
  - even when it is not read, the contents must be **refreshed** (read and rewritten)
    - every few milliseconds
    - the charge on the capacitor gradually leaks away

Robert Dennart
(1932, 2024)
He invented DRAM (1966) at IBM. Many were skeptical, but by mid-70 DRAM was in all computers!
He received 35 patent in semiconductors and microelectronics.

Static Random Access Memory (SRAM)

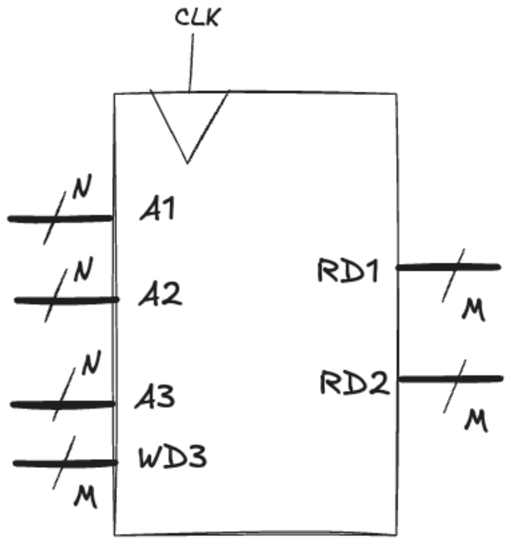
- The data bit is stored on cross-coupled inverters



- Each cell has two outputs, bitline and $\overline{\text{bitline}}$
  - when the wordline is asserted, both transistors turn on
  - data values are transferred to or from the bitlines

- Do not need to be refreshed

Register Files



- A set of registers used to store **temporary variables**
- Usually built as a small, multi-ported SRAM array
  - more compact than an array of flip-flops
  - multi-ported means that can access several addresses simultaneously

- For example, a 32-register (N=5) with 32-bit (M=32) three-ported register file
  - two read ports (A1->RD1 and A2->RD2)
  - one write port (A3->WD3)
  - the 5-bit addresses (A1, A2, A3) can access all $2^5 = 32$ registers
  - two registers can be read and one register written simultaneously

- Sometimes, a particular register is **hardwired** to always read the value 0
  - because 0 is a commonly used constant

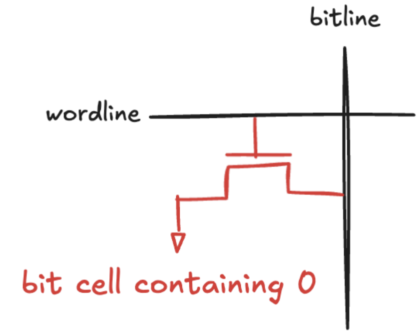
Memory comparison

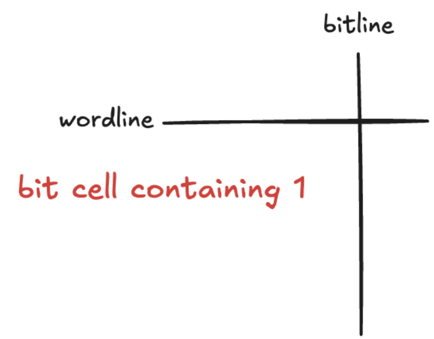
- Flip-flops, SRAM, and DRAM are all volatile memories, but each has different **area** and **delay** characteristics

Memory Type	Transistors per bit cell	Latency
Flip-Flop	20	Fast
SRAM	6	Medium
DRAM	1	Slow

- Flip-flop data is available immediately in output, but it takes at least 20 transistors
- DRAM is slower
  - must wait for charge to move (relatively) slowly from the capacitor
  - must refresh data periodically and after a read
- Today, modern DRAM technologies (synchronous and double data rate, DDR)
  - uses both rising and falling edges of the clock to access data
  - first standardized in 2000 (100 MHz), today (2024) speeds over 5 GHz (DDR5)
- The **best memory type** for a particular design **depends on the speed, cost, and power constraints**

Read Only Memory (ROM) (1)

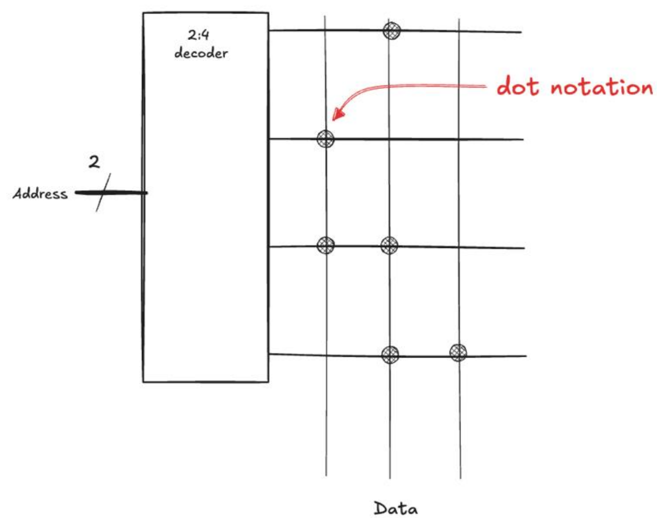




- ROM stores a bit as the **presence** or **absence** of a transistor
- To read the cell:
  - the bitline is weakly pulled HIGH
  - then, the wordline is turned ON
    - if the transistor is present, it pulls the bitline LOW
    - if it is absent, the bitline remains HIGH

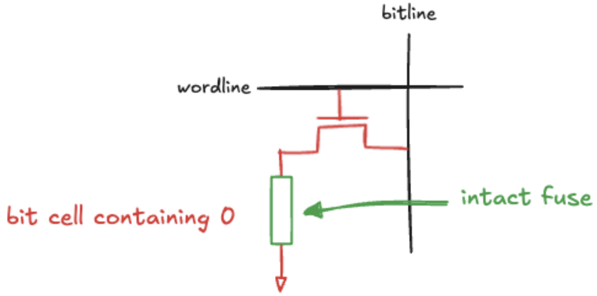
- The contents of a ROM can be indicated using dot notation

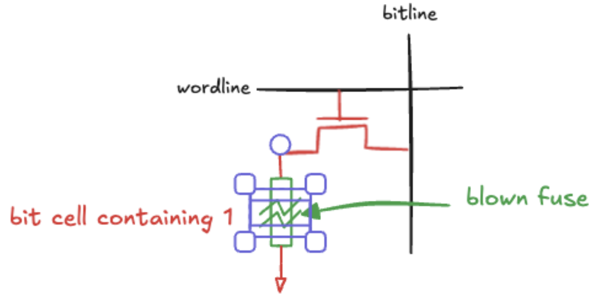
Address	Data		
00	0	1	0
10	1	0	0
01	1	1	0
11	0	1	1



Programmable ROM

- Places a transistor in **every** bit cell, but provides a **way to connect or disconnect** the transistor to ground
- User programs the ROM by applying a high voltage to selectively blow fuses
  - if the fuse is present, transistor is connected to GND and the cell holds 0
  - if the fuse is destroyed, transistor is disconnected from ground and the cell holds 1





- **One-time programmable ROM**
  - because the fuse cannot be repaired once it is blown
  - The process id called **burning** a ROM

- Exist also **reversible mechanism** for connecting or disconnecting the transistor

Re-programmable ROM

Fujio Masuoka (1943–) invented Flash memory at Toshiba during the late 1970s as an unauthorized project he pursued during nights and weekends. The name "Flash" was inspired by the way the memory is erased, resembling the quick flash of a camera. Although Toshiba developed the technology, they were slow to commercialize it, allowing Intel to bring the first Flash memory product to market in 1988.

- Erasable PROMs (**EPROM**)
  - replace the transistor and fuse with a floating-gate transistor
    - not physically attached to any other wires
    - when high voltages are applied, electrons tunnel through an insulator onto the floating gate, turning on the transistor and connecting the bitline to the wordline
    - when exposed to ultraviolet light (half an hour), the electrons are knocked off the floating gate, turning the transistor off
    - these actions are called **programming** and **erasing**
- Electrically erasable PROMs (**EEPROM**) and **Flash memory**
  - use similar principles but include circuitry for erasing and programming
  - in 2024, Flash cost about $0.05 per GB, and price is dropping
    - a way to store large amounts of data in portable battery-powered systems
- ROMs are **not really read only**: they can be programmed
  - difference between RAM and ROM: ROMs **take a longer time to write** but are **non-volatile**

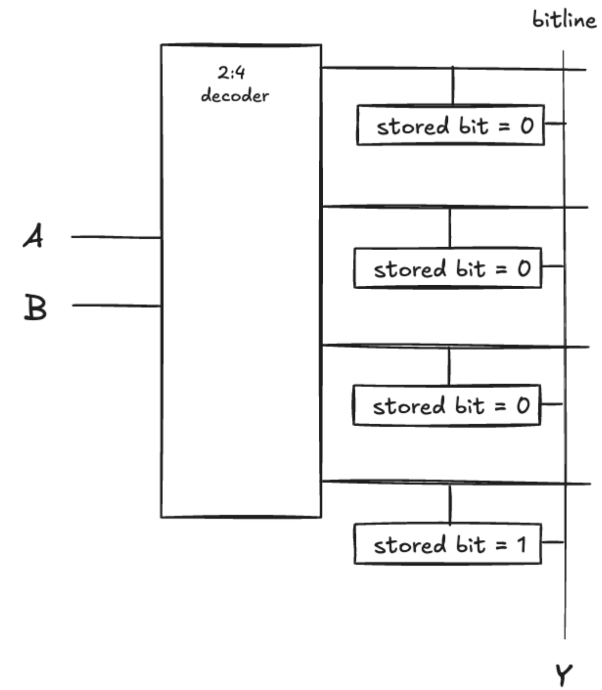
Lookup tables (LUT)

- Memory arrays can also **perform combinational logic functions**
  - each address corresponds to a row in the truth table
  - each data bit corresponds to an output value

- A 2^N-word × M-bit memory can perform any combinational function of N inputs and M outputs

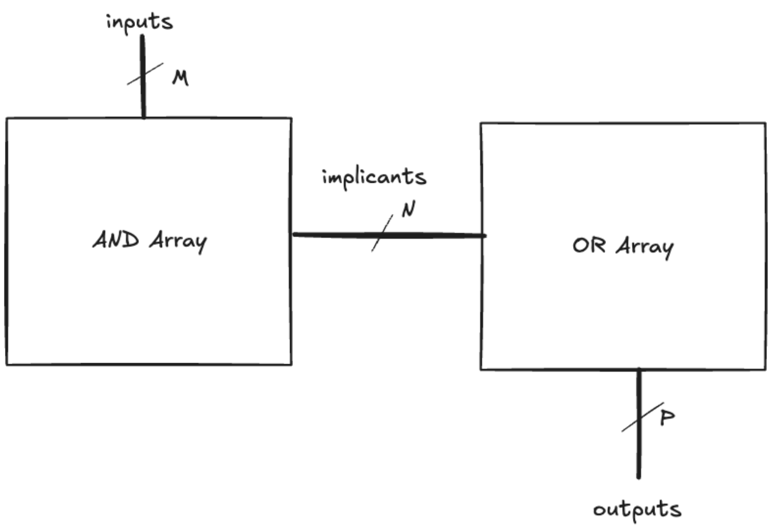
- The following figure shows a LUT (4-word × 1-bit) to perform the function Y = AB

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



Programmable Logic Array (PLA)

- Like memory, gates can be **organized into regular arrays**
  - if connections are made programmable, they can be configured to perform any function
- PLA implement two-level combinational logic in sum-of-products form
  - the inputs (in true and complementary form) drive an AND array
    - which produces implicants
  - the implicants, in turn, are ORed together to form the outputs
- An M×N×P-bit PLA has M inputs, N implicants, and P outputs

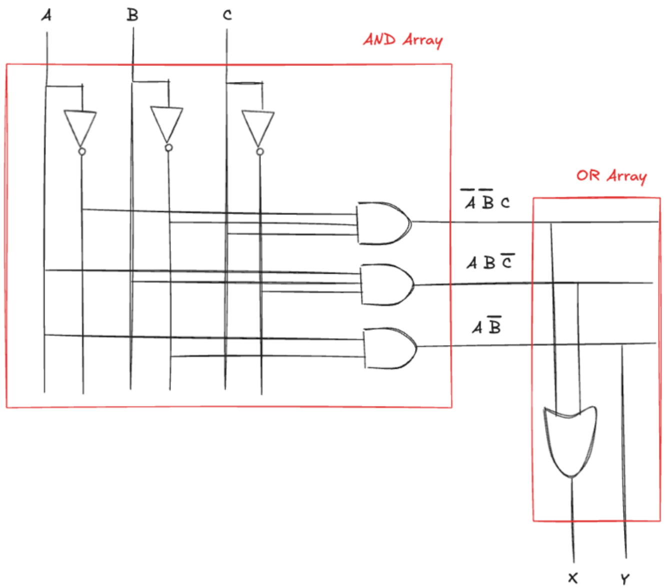


Programmable Logic Array (PLA)

- Example:

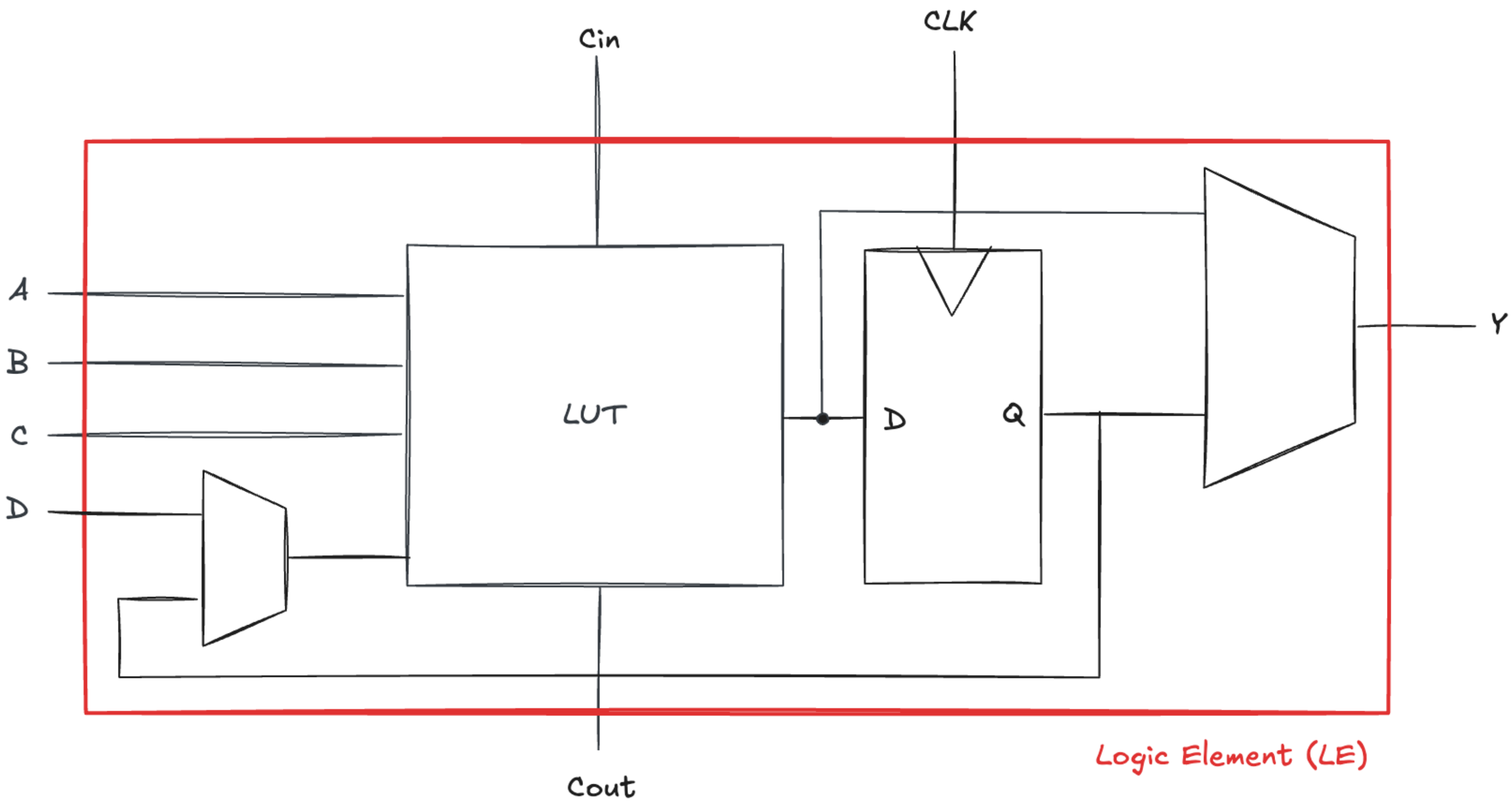
  $X = \bar{A}\,\bar{B}\,C + A\,B\,\bar{C}$

  $Y = A\,\bar{B}$



Field Programmable Gate Array (FPGA) (1)

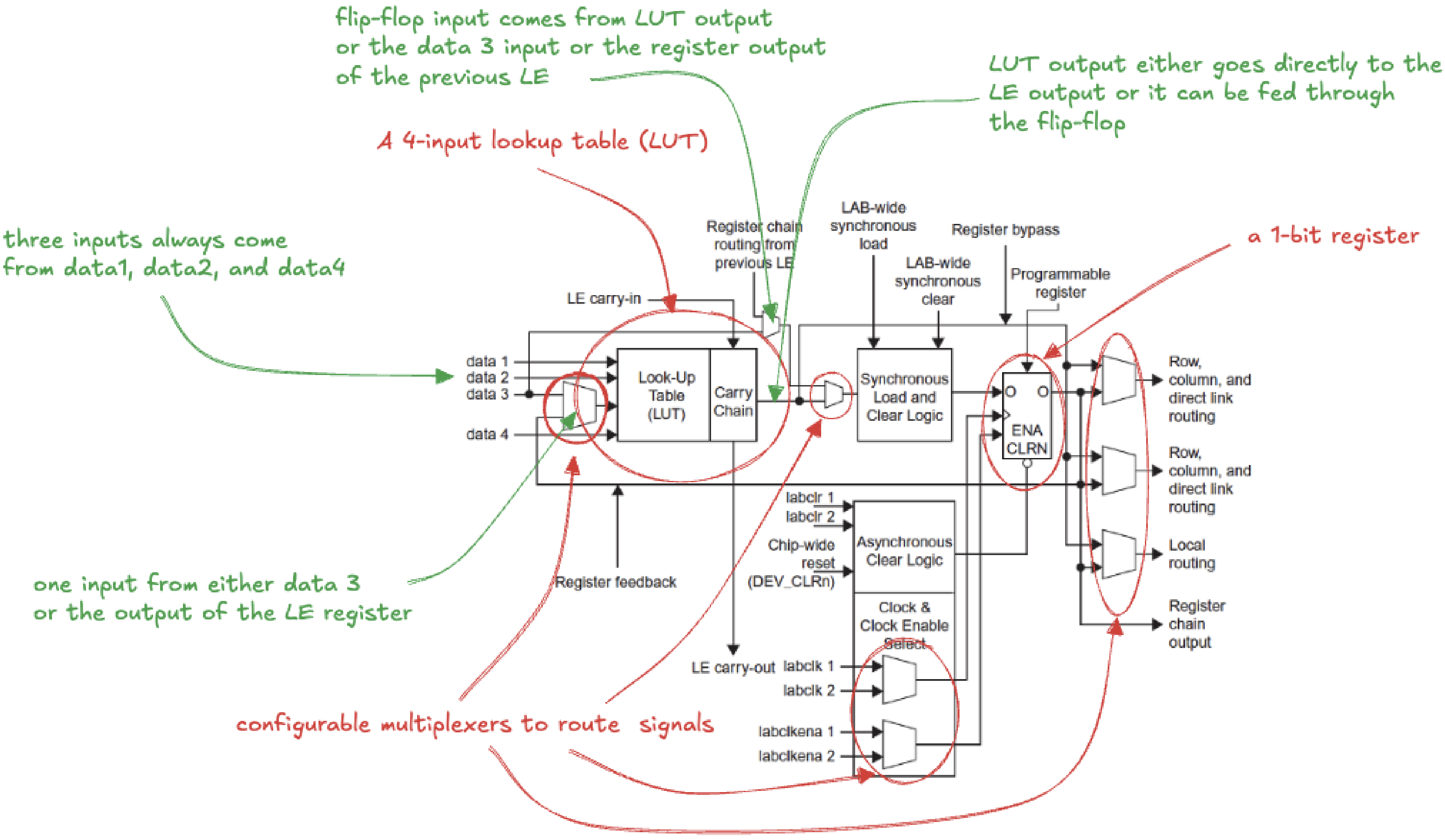
- An array of **reconfigurable logic elements (LE)**



- LE can be configured to perform combinational or sequential functions

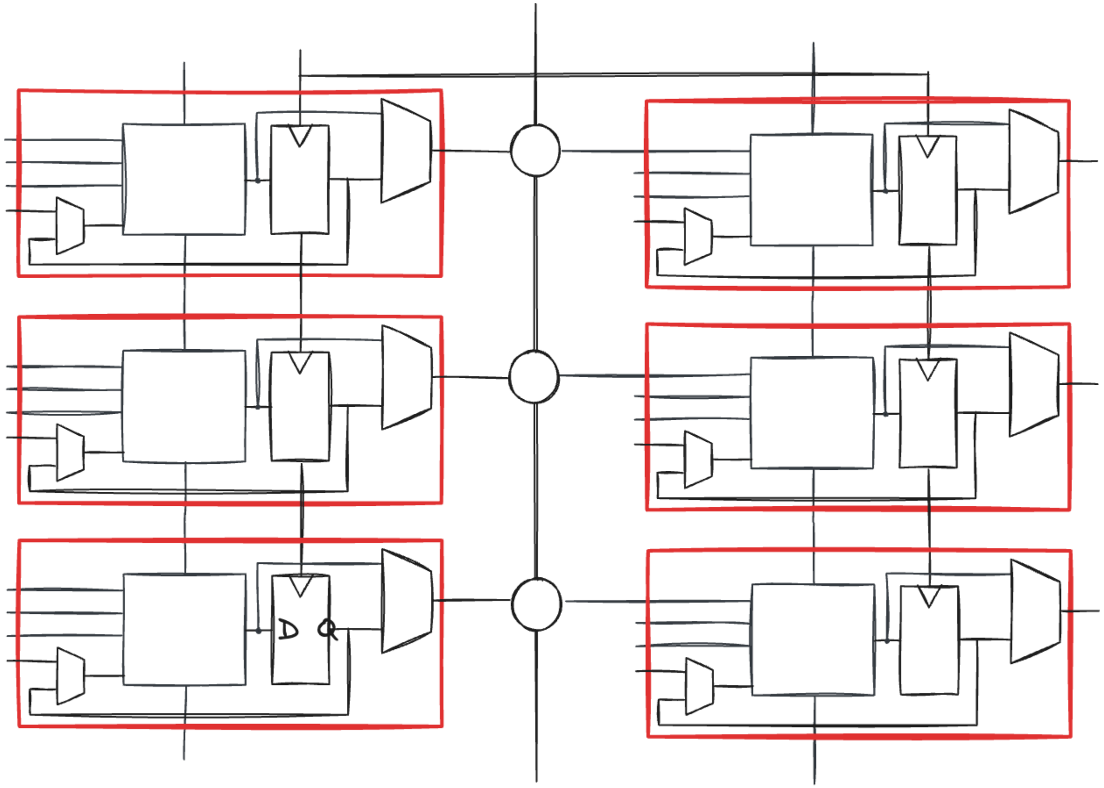
Field Programmable Gate Array (FPGA) (2)

- A single LE from Intel's Cyclone IV FPGA



Field Programmable Gate Array (FPGA) (3)

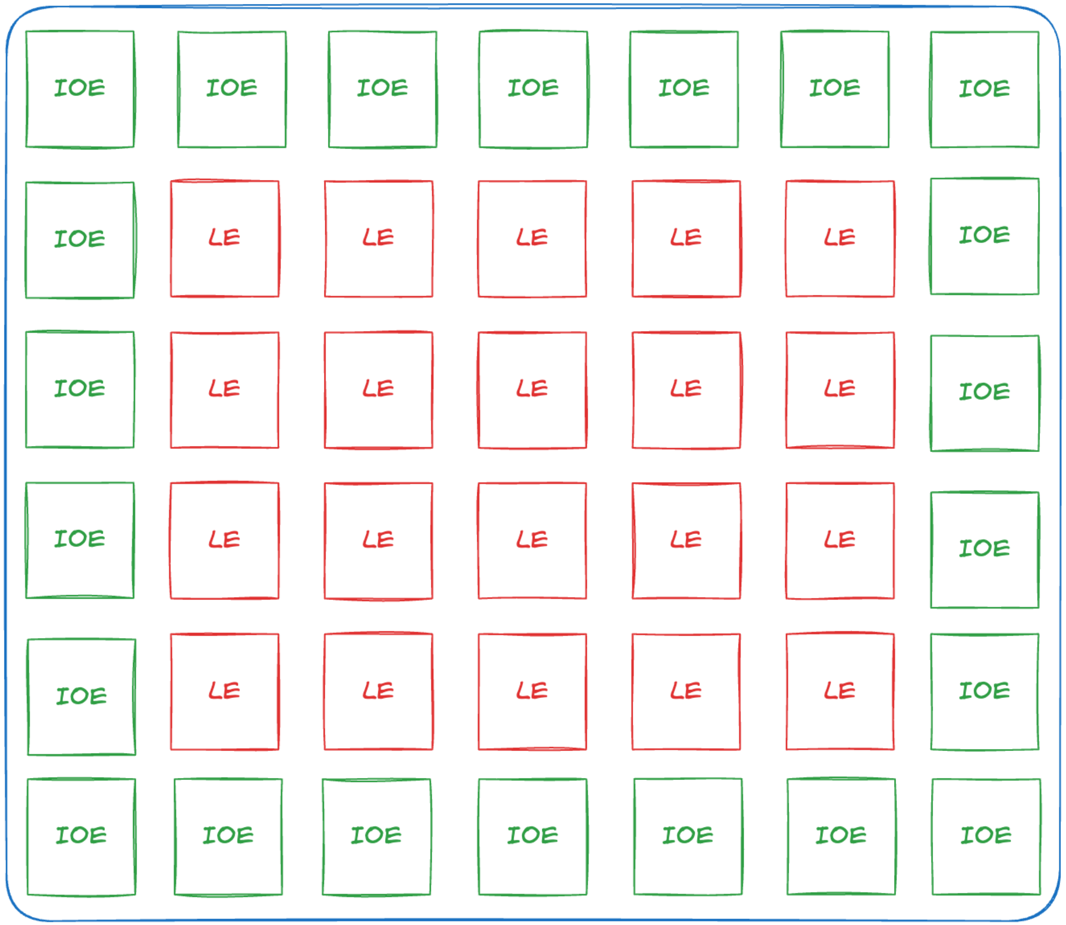
- The power of an FPGA emerges from **scale**
  - each LE can be connected (flexibly and dynamically) to many others
  - allowing thousands of simple elements to cooperate in complex digital systems



- Circles represent **programmable routing switches** that allow the output of a logic cell to be directed almost anywhere on the chip

Field Programmable Gate Array (FPGA) (4)

- LEs are surrounded by **input/output elements** (**IOE**) for interfacing with the outside world
  - IOEs inputs and outputs to pins on the chip package



Field Programmable Gate Array (FPGA) (5)

- The designer configures an FPGA by first creating an HDL description
- The design is then synthesized onto the FPGA
  - determines how the LUTs, multiplexers, and routing channels should be configured to perform the specified function
- This configuration information is then downloaded to the FPGA
- DEEDS is capable of creating an HDL description of our schematic that can be deployed on a real FPGA, as we will see in the next semester!