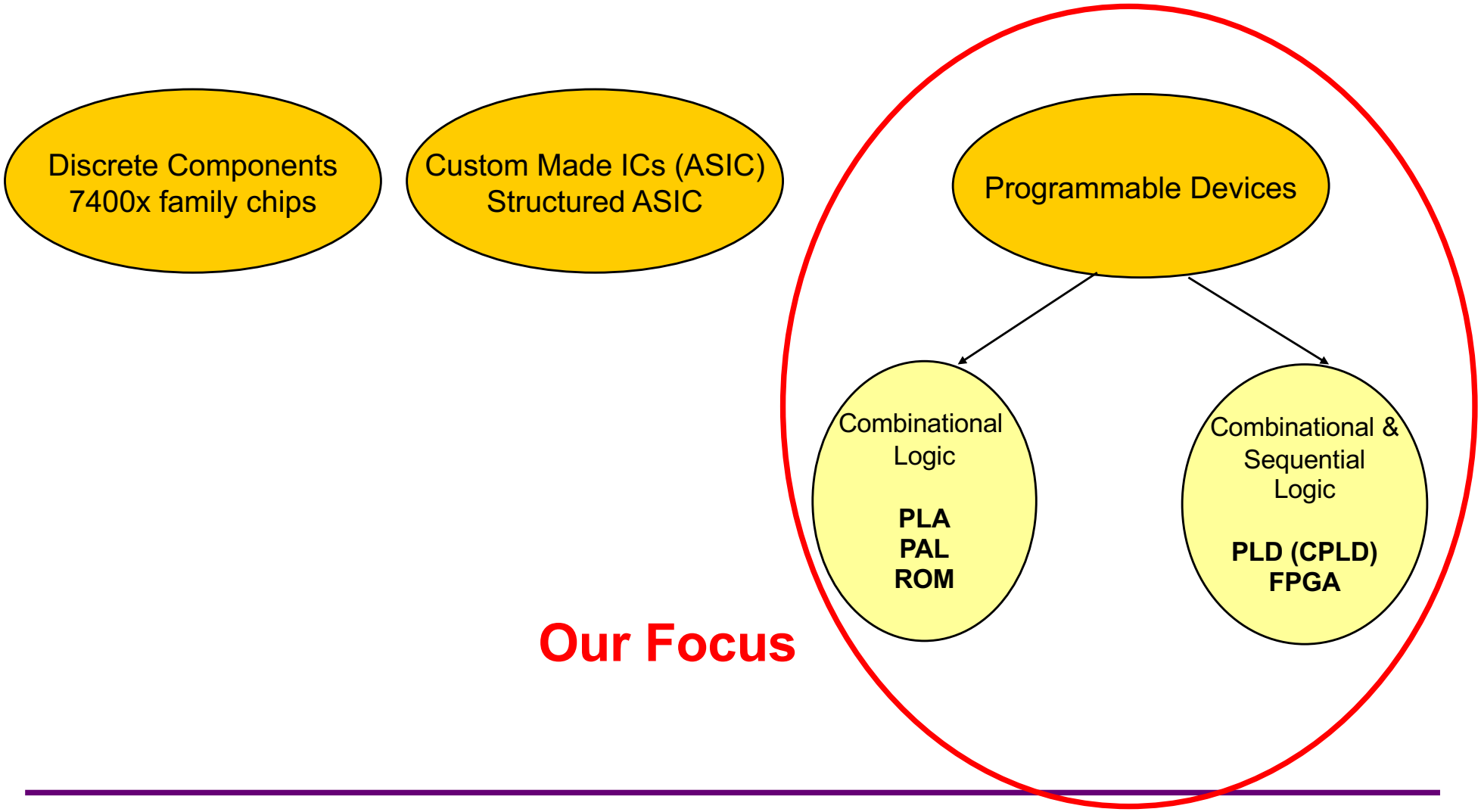
Lecture 13
Programmable Logic Technologies

Implementation Technologies Overview



Overview

- Why focus on the programmable logic devices?
  - Their structure is simple and well-defined
    - Such as regular arrays of AND gates and OR gates
    - Highly suitable to run a two-level logic optimization on a Boolean function and then map it onto one such device
      - Logic minimization techniques discussed in previous lecture modules directly applicable for the design process targeting these devices
  - Advanced programmable devices such as FPGAs are really useful these days
    - Prototype design
    - Lower development cost
    - Reconfigurability

BIG PICTURE

- We'll start with simplest structure and go towards the most sophisticated
- Simple in two meanings
  - Architecture (what are they made of?)
    - AND and OR gates versus Look up Table, MUX, D-Flip Flop combinations
  - Programmability
    - One-time versus repeatedly
    - Off the board using a special device versus inserting the programmable device to where it is supposed to function and then program repeatedly there without taking it off and inserting it into some programming machine
      - The name "Field Programmable" comes from this

Combinational Logic Technologies

- MOS transistors
  - We had discussed basics of the MOS transistors before
- PALs/PLAs
- ROM
- Field Programmable Gate Arrays (FPGAs)

Programmable Arrays of Logic Gates

- You have implemented Boolean functions using discrete logic gates in 203
  - NOT, AND, OR, NAND, NOR, XOR, and XNOR
  - 74000-family chips
- Can arrange AND and OR gates (or NAND and NOR gates) into a general and systematic array structure
- Program array to implement logic functions
- Two popular variants
  - Programmable logic arrays (PLA) and programmable array logic (PAL)

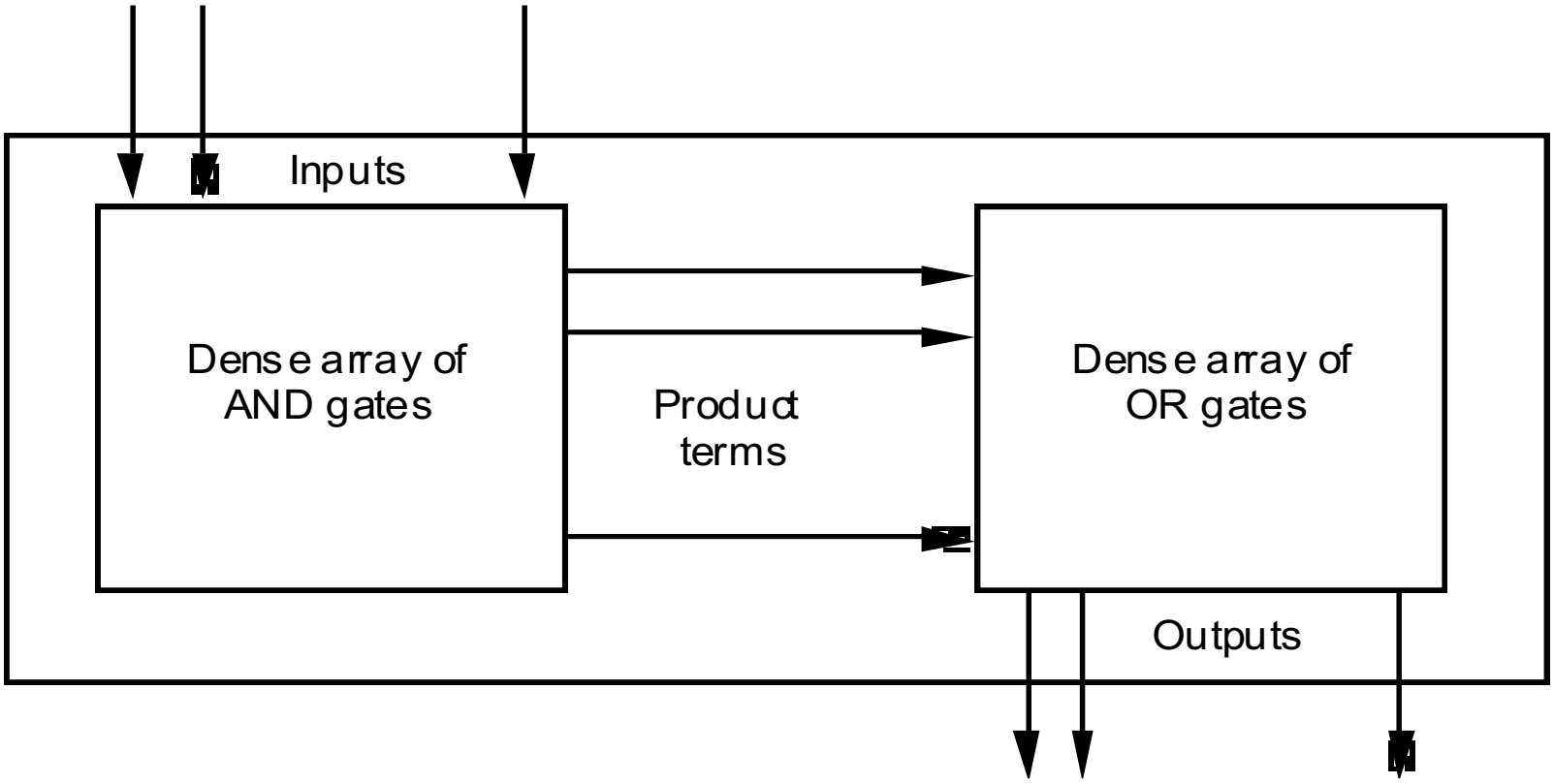
Programmable Arrays of Logic Gates

- When the semiconductor technology discovered efficient ways to build these devices the need for optimally representing 2-level logic has increased
  - This fueled the need to develop automated CAD algorithms (such as ESPRESSO) to perform 2-level logic minimization

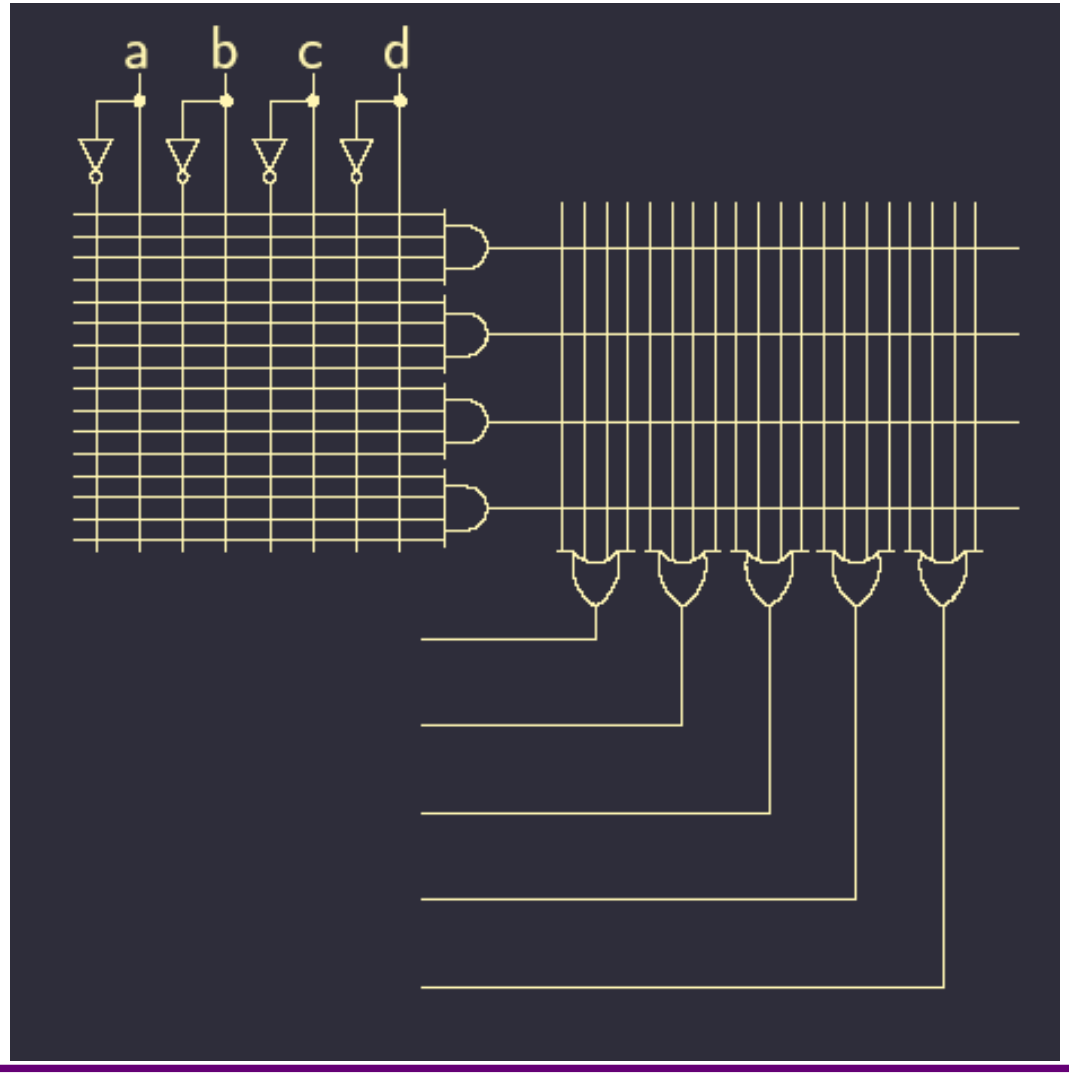
PALs and PLAs

- Pre-fabricated building block of many AND/OR gates (or NOR, NAND)
- "Personalized" by making or breaking connections among the gates

Programmable Array Block Diagram for Sum of Products Form



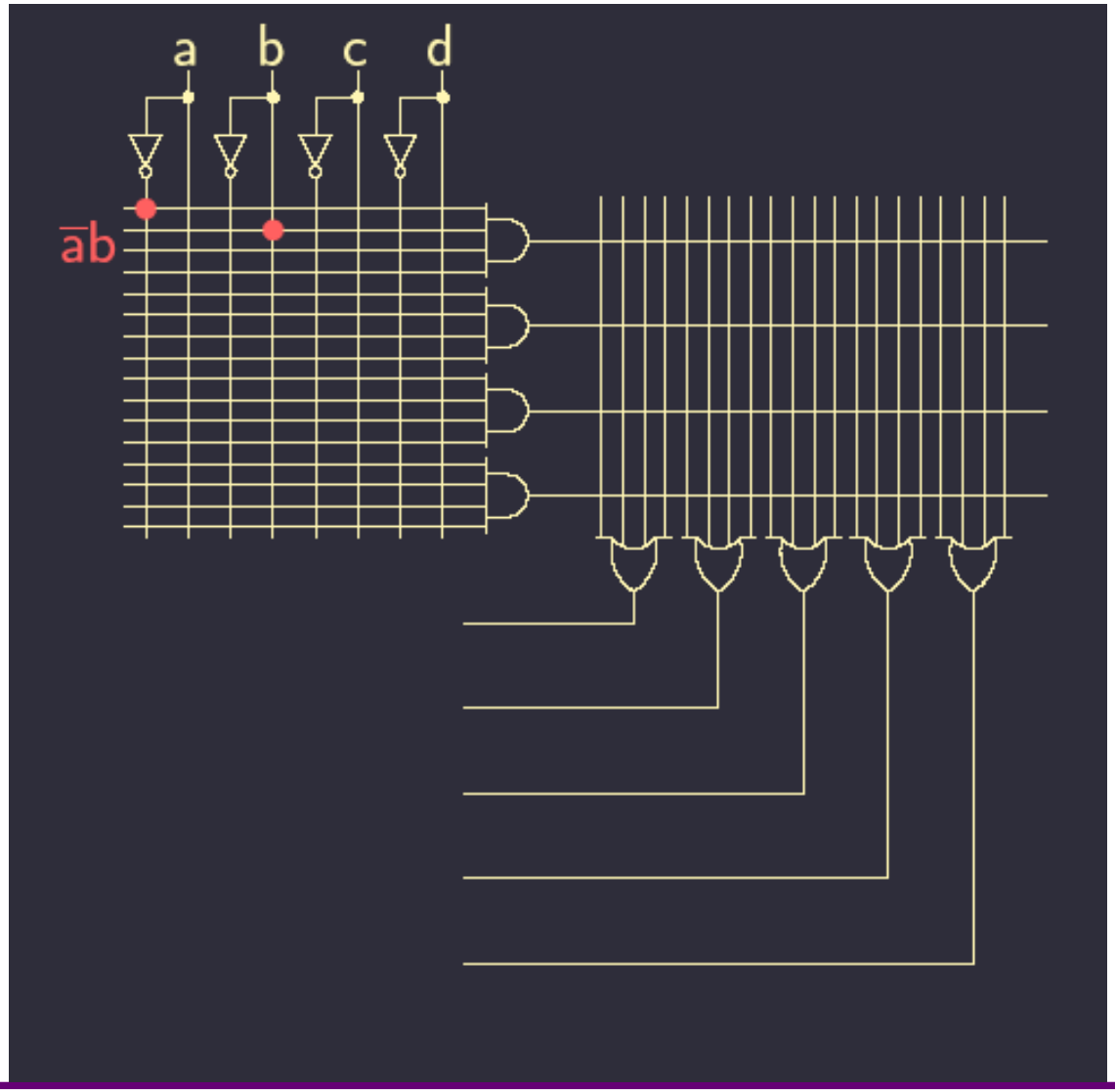
PLA Programming



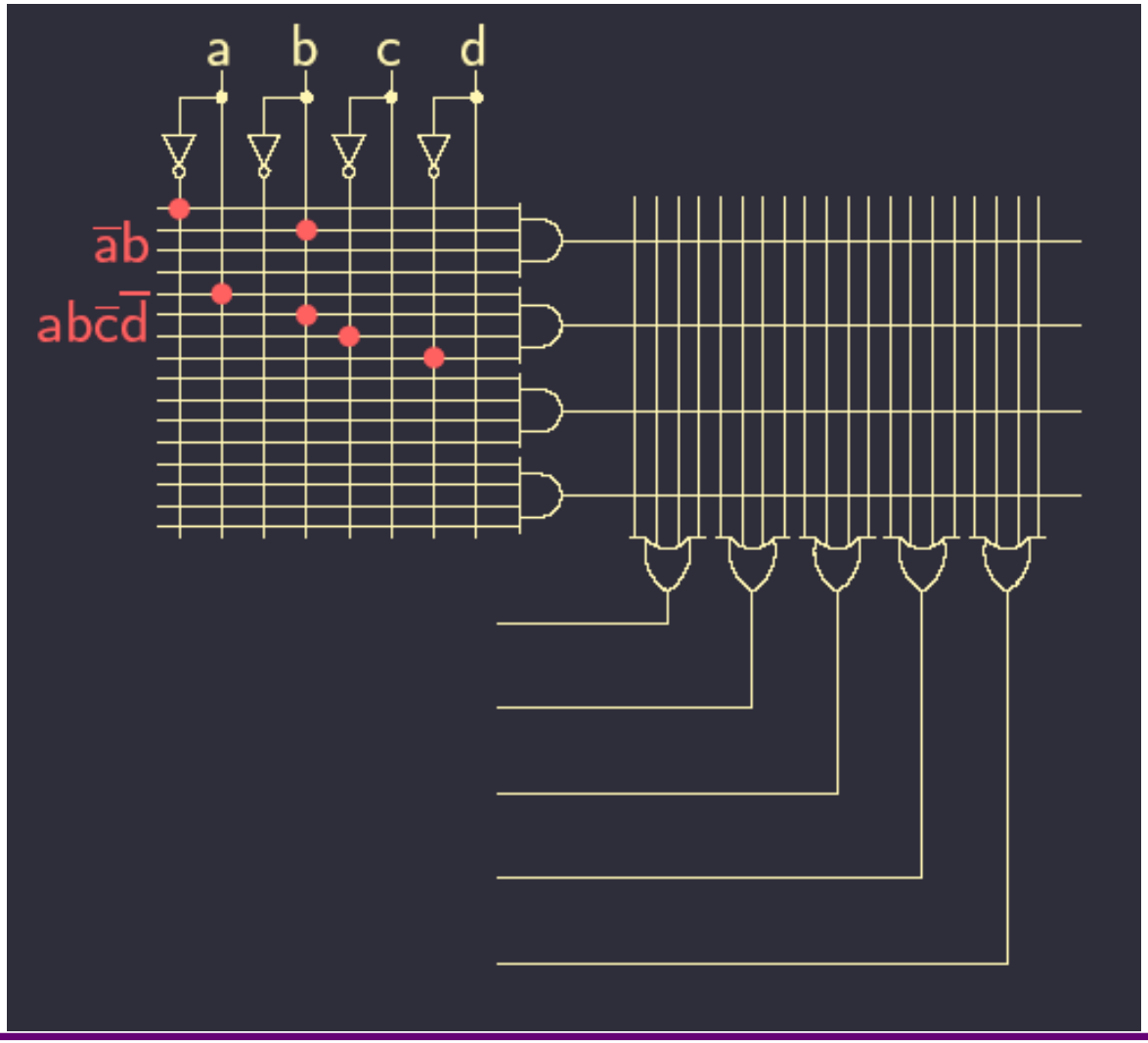
PLA Programming

- All connections available
  - All exist
    - Some need to be removed during programming
  - None exist
    - Connections need to be made during programming

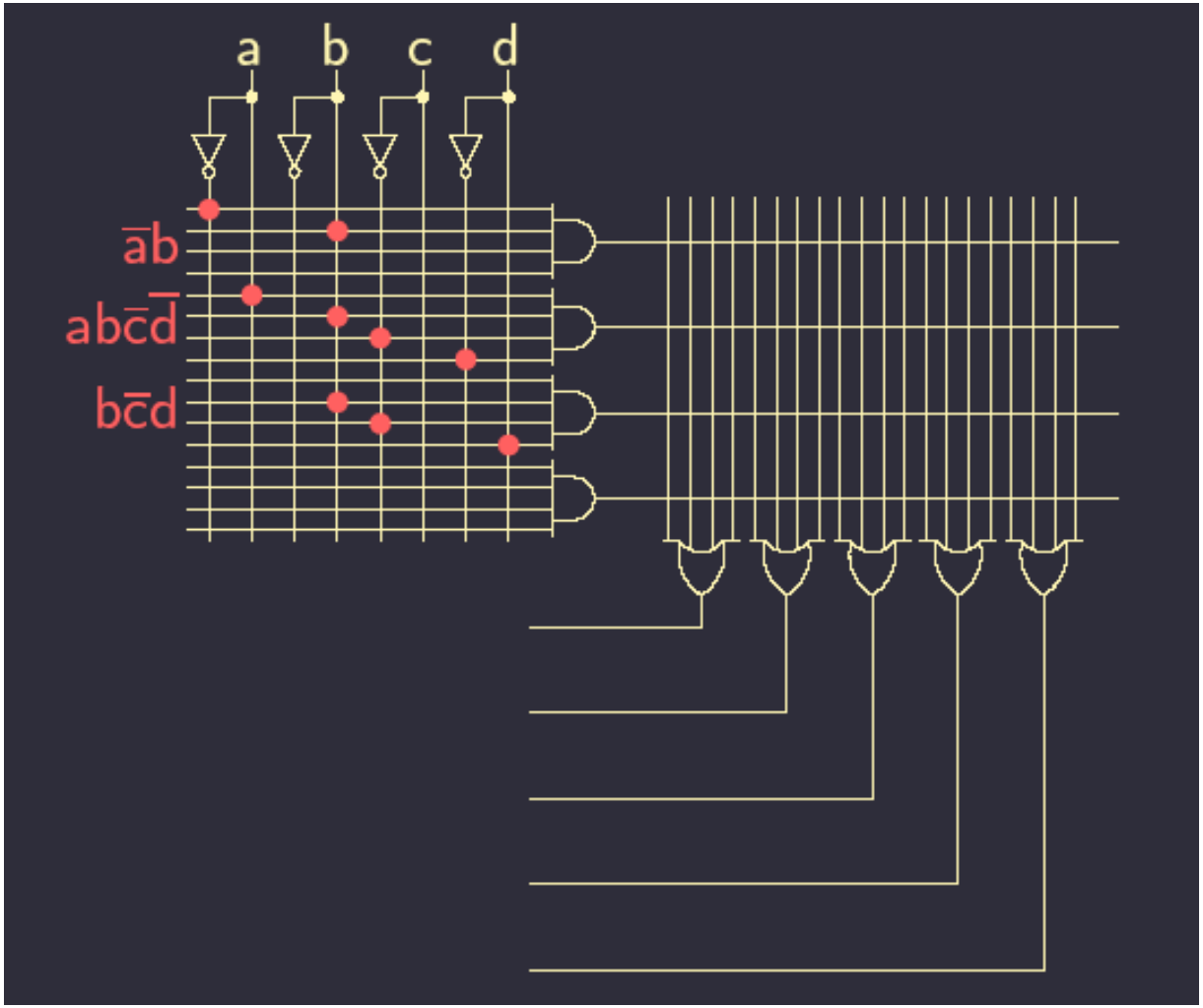
PLA Programming



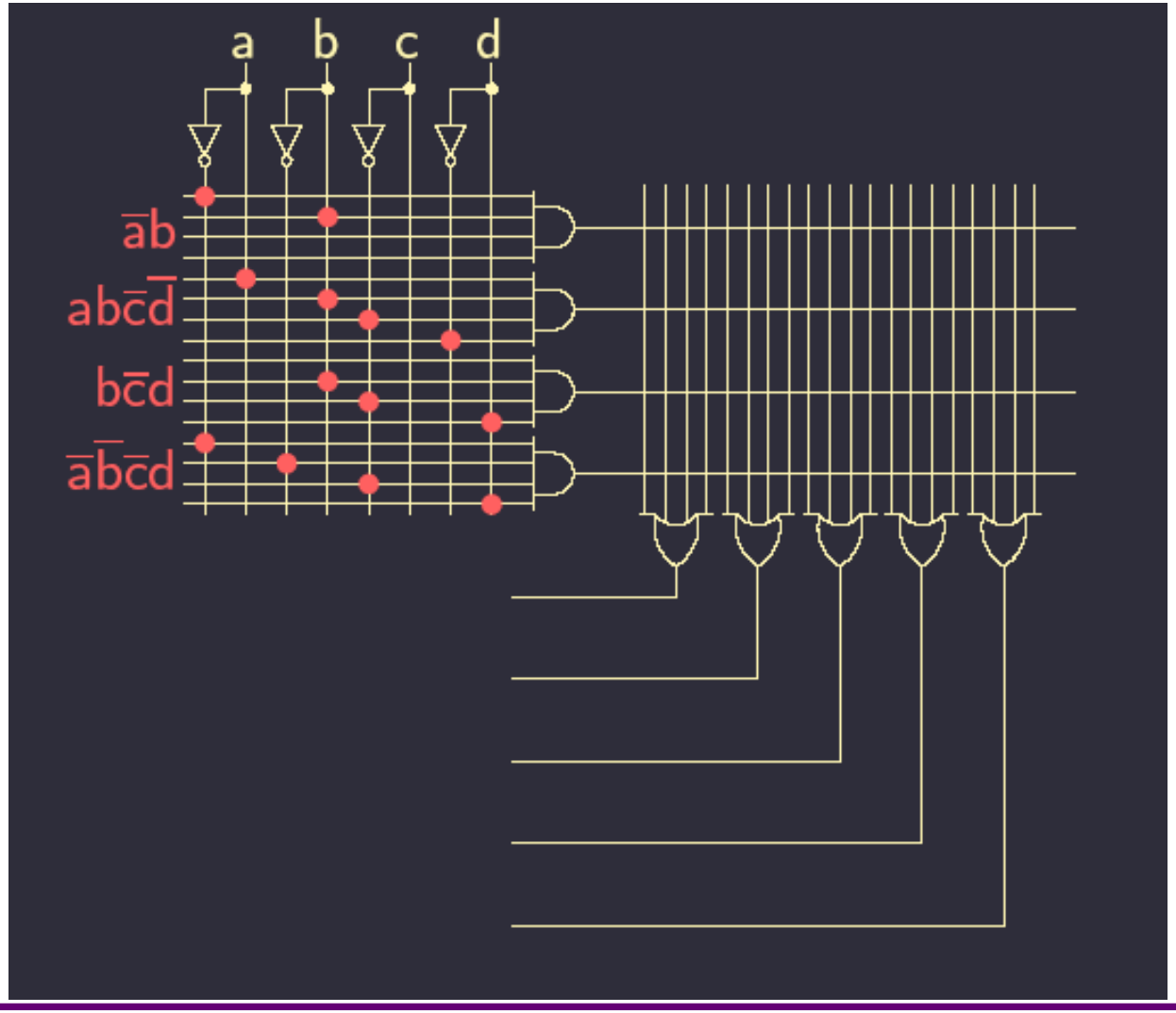
PLA Programming



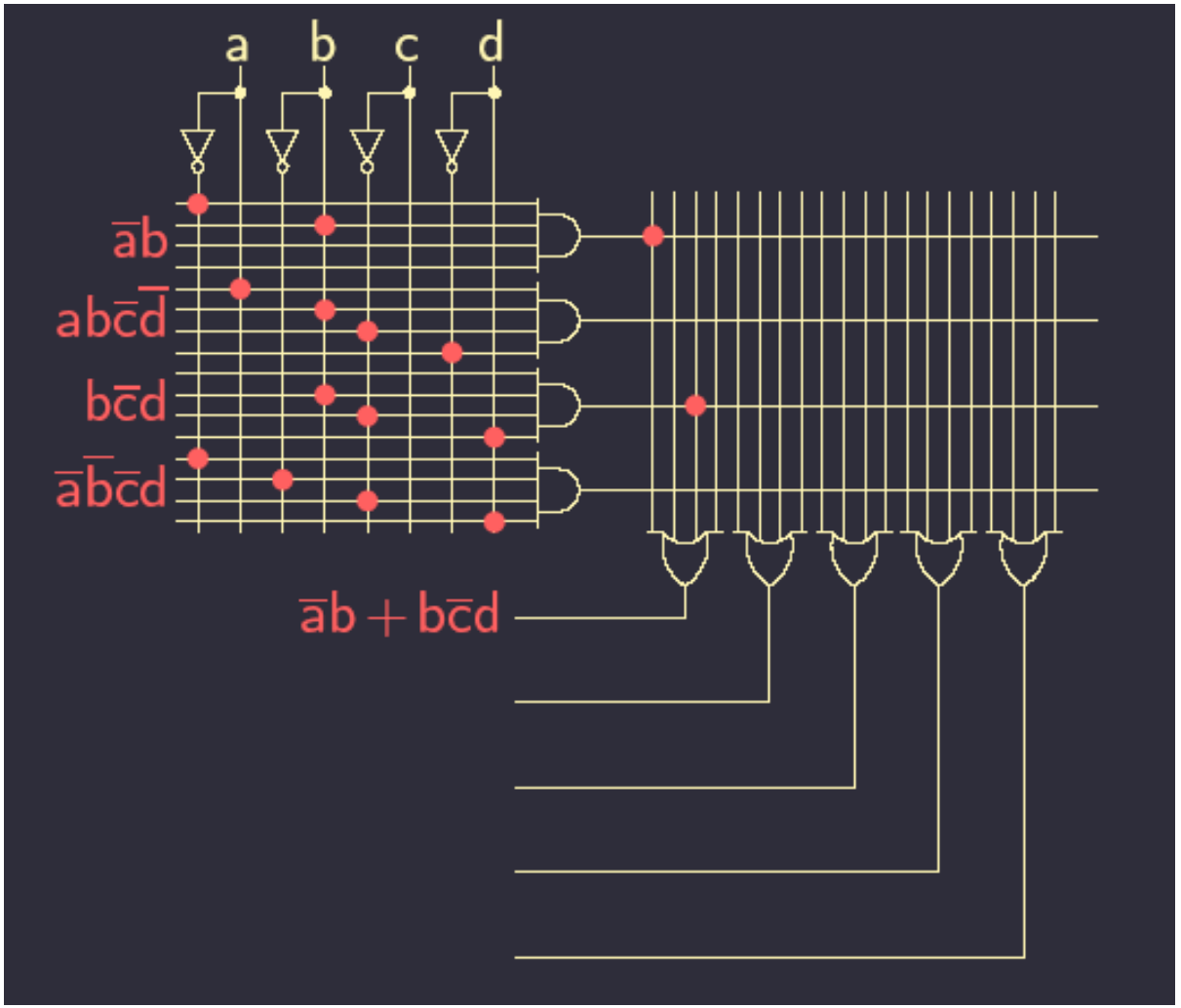
PLA Programming



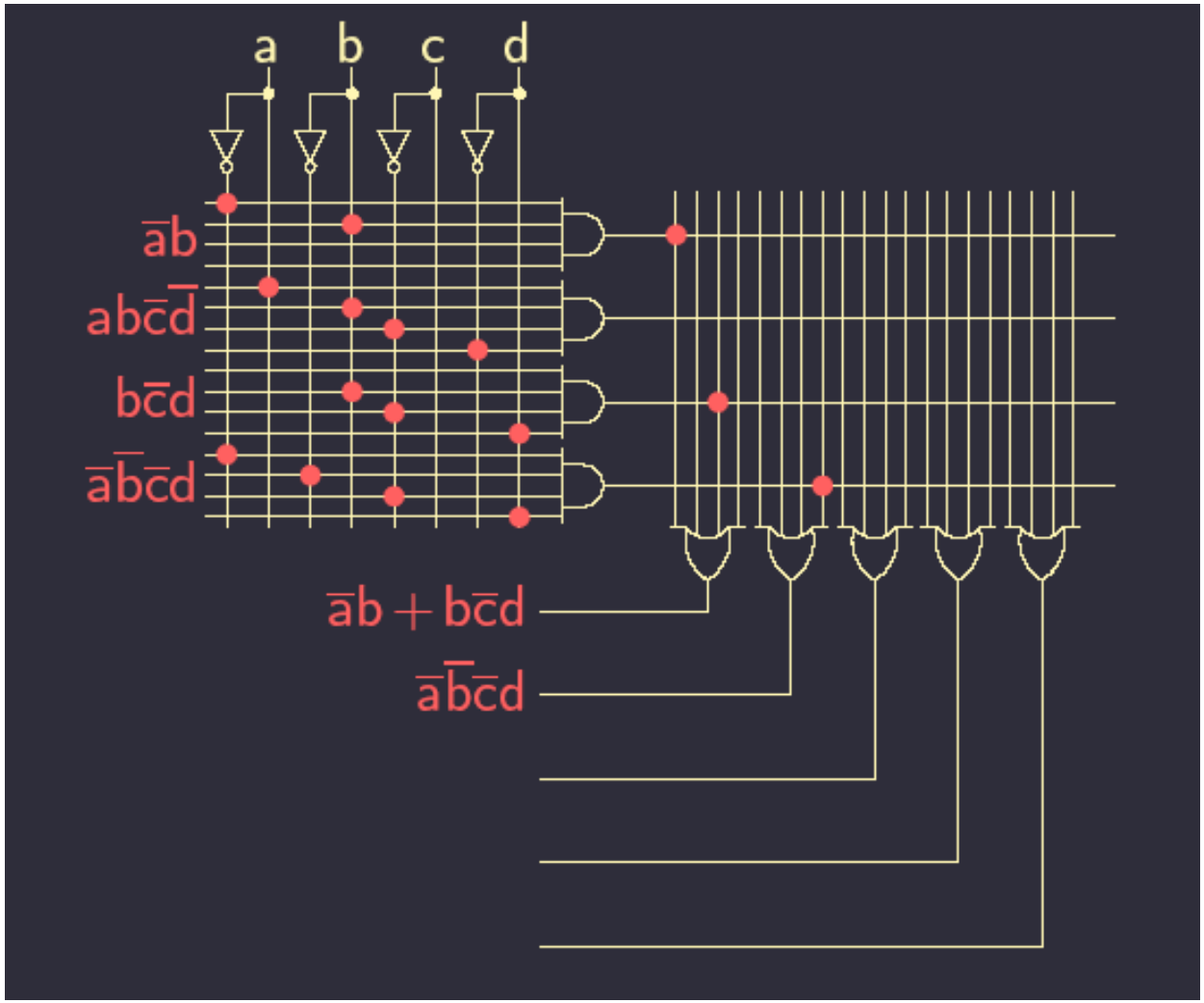
PLA Programming



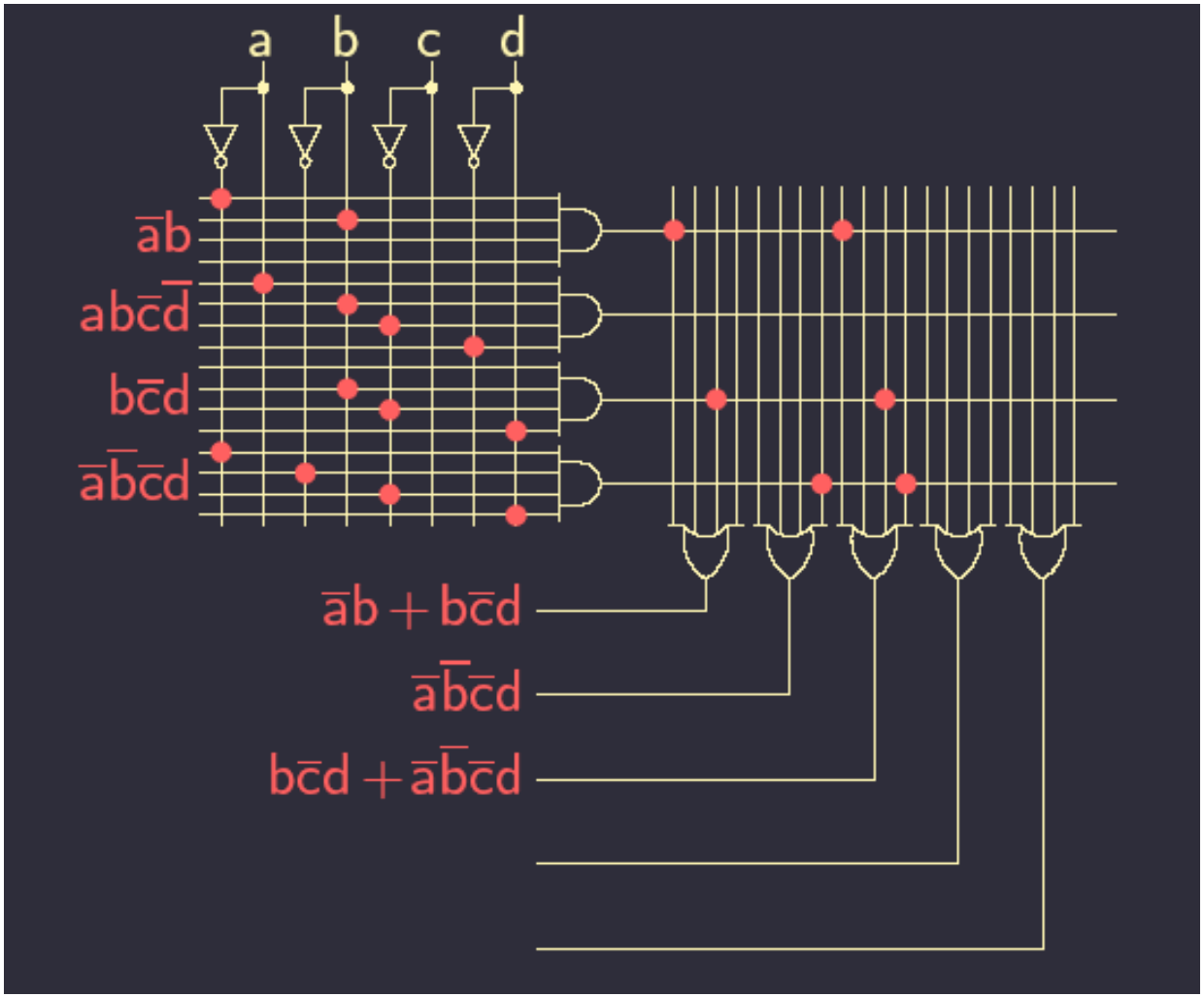
PLA Proramming



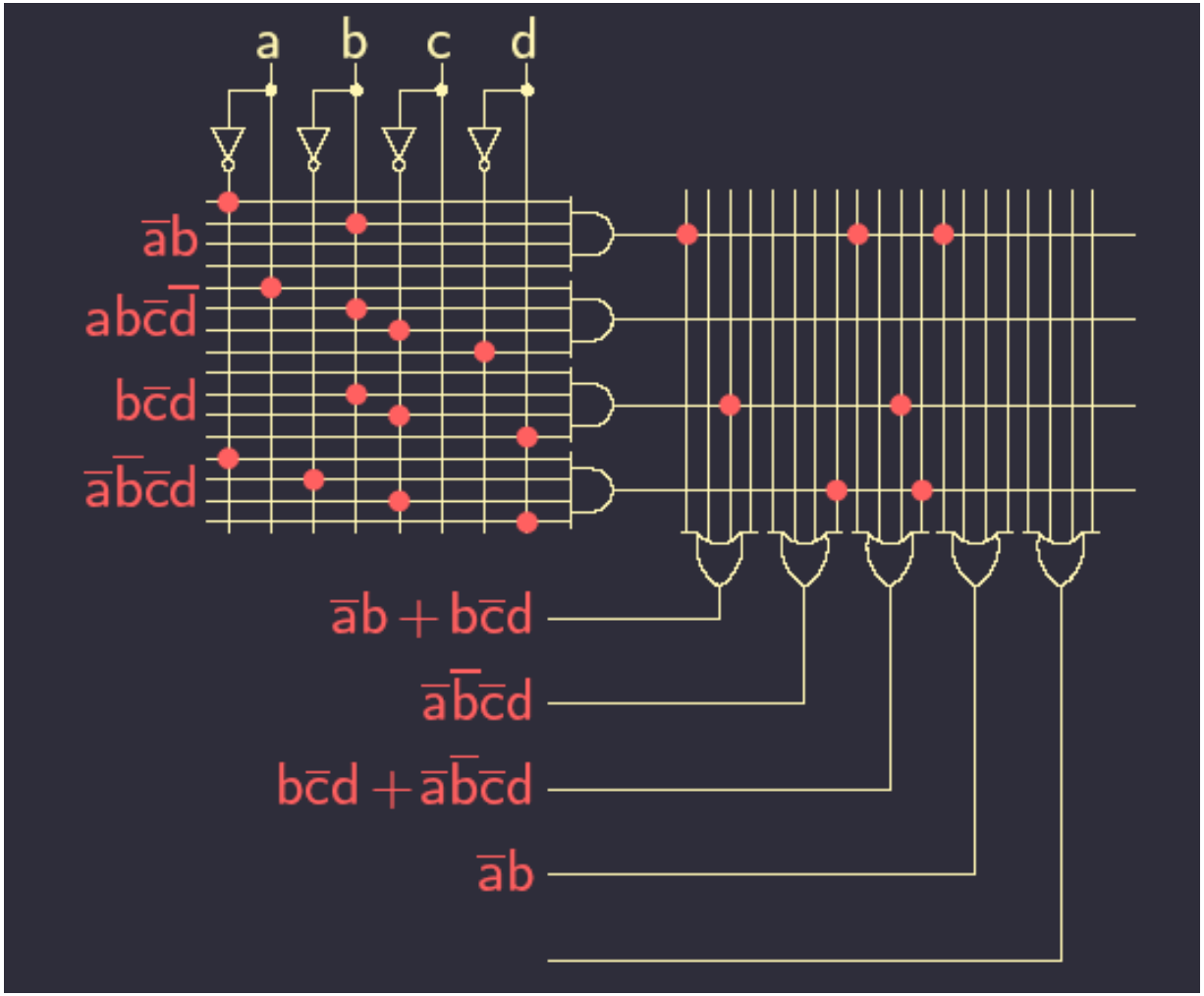
PLA Programming



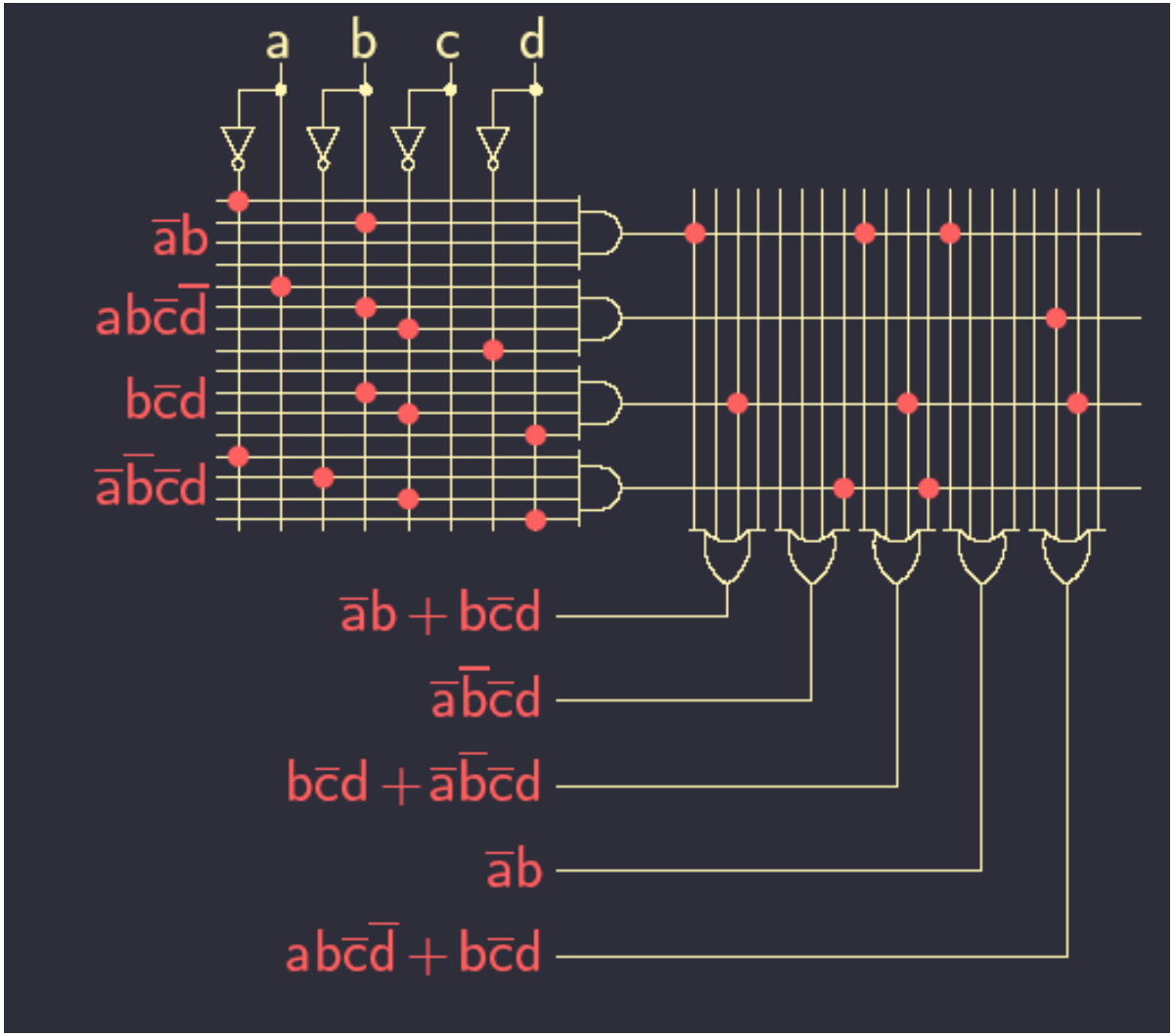
PLA Programming



PLA Programming



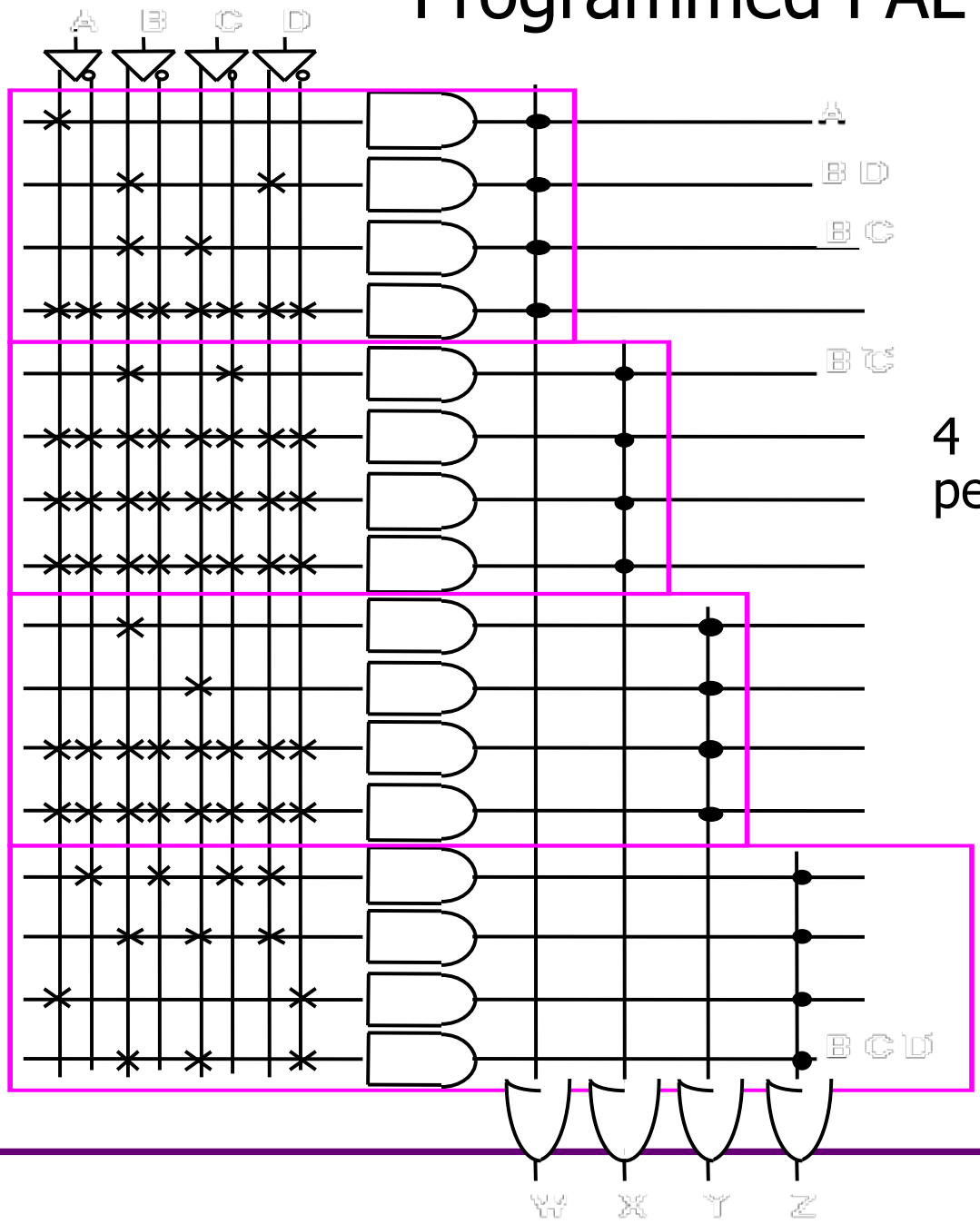
PLA Programming



PLA versus PAL

- They are built according to the same principle and for the same purpose
- The difference is:
  - In a PLA device all OR gates have access to any and all product terms
  - In a PAL device each OR gate is associated with a fixed number of AND gates
    - Hence, if the same product term is used in more than one function, each would need to generate the same product to feed to its respective OR gate

Programmed PAL

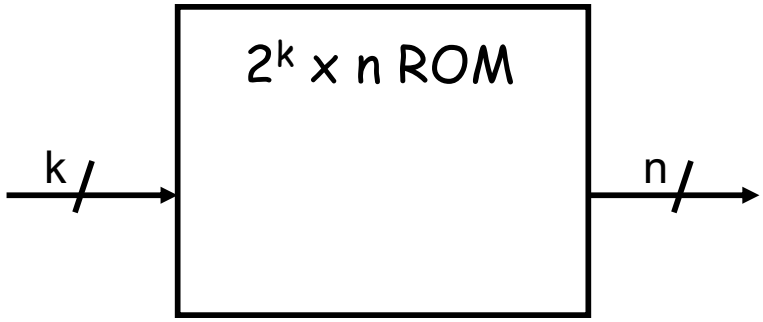


4 product terms per OR gate

PAL Programming Table

Product Term	AND Inputs				Outputs
	A	B	C	D	
1	1	-	-	-	W= A + BD + BC
2	-	1	-	1	
3	-	1	1	-	
4	-	-	-	-	
5	-	1	0	-	X = BC'
6	-	-	-	-	
7	-	-	-	-	
8	-	-	-	-	
9	-	1	-	-	Y = B + C
10	-	-	1	-	
11	-	-	-	-	
12	-	-	-	-	
13	0	0	0	1	Z = A' B' C' D + BCD + AD' + B' CD'
14	-	1	1	1	
15	1	-	-	0	
16	-	0	1	0	

Read-only memory



- A read-only memory, or ROM, is a combinational circuit with k inputs and n outputs
  - It takes an address as input, and produces some data as the output
- The internal organization is similar to a memory device
  - But no sequential circuit, No flip flops or latches!

Read-only memory

- ROMs are useful for holding data that never changes
  - Computation of complex calculations (trigonometric functions, logarithms, etc.) can be accelerated by looking the results up from a ROM
  - Many computers use a ROM to store important programs that should not be modified, such as the system BIOS.
  - Firmware in devices: Fonts for laser printers, Sound data in electronic music instruments, Code in PDAs, smartphones, etc.

Read-Only Memory

Five basic ROM types: ROM, PROM, EPROM, EEPROM, Flash memory.

Each type has unique characteristics, the following are common:

- Data stored in these chips is **nonvolatile** -- it is not lost when power is removed
- Data stored in these chips is either **unchangeable** or requires a special operation/device to change

Read-Only Memory

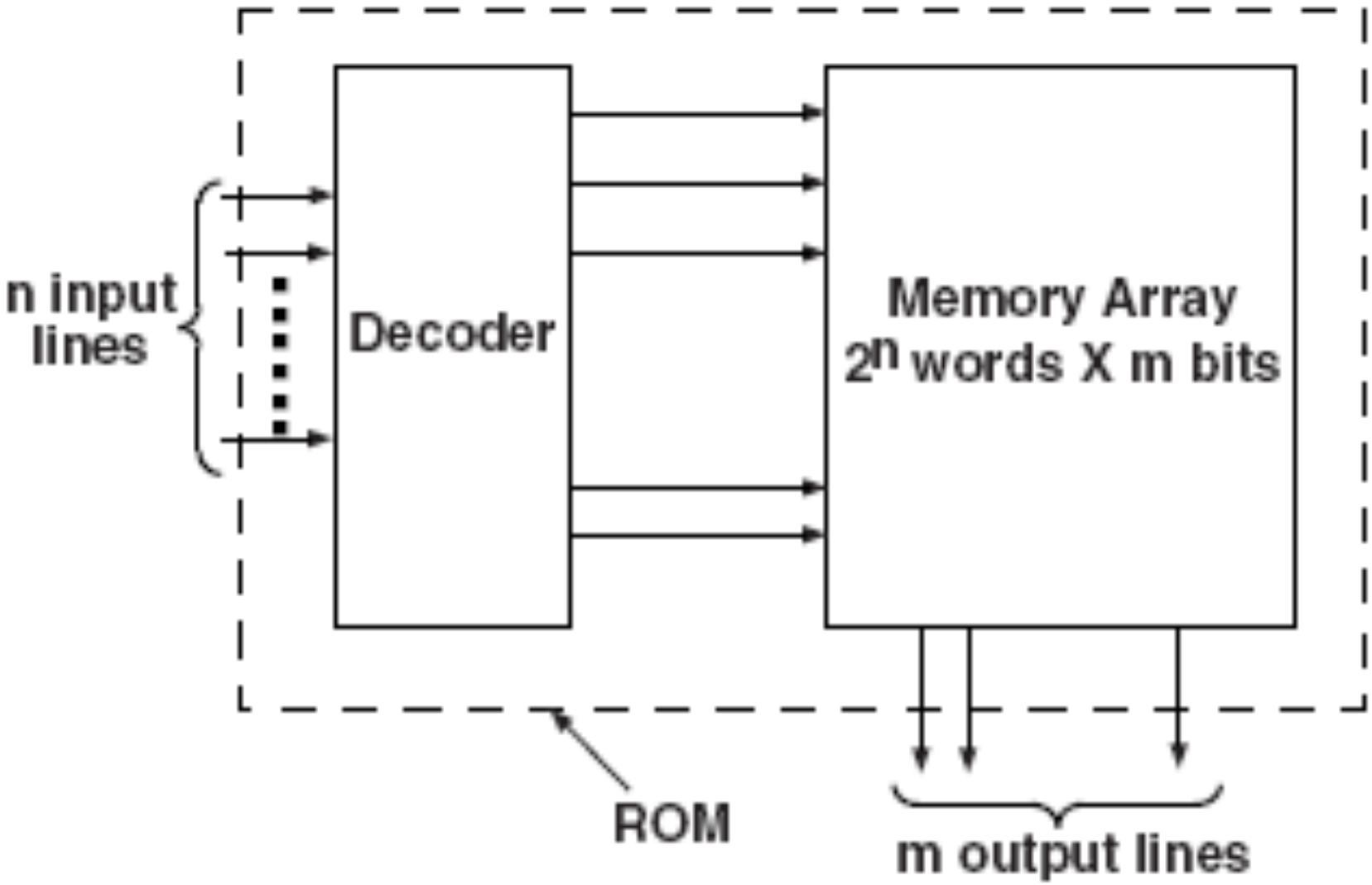
Data is usually installed during the production process

ROMs are produced as an integrated circuit

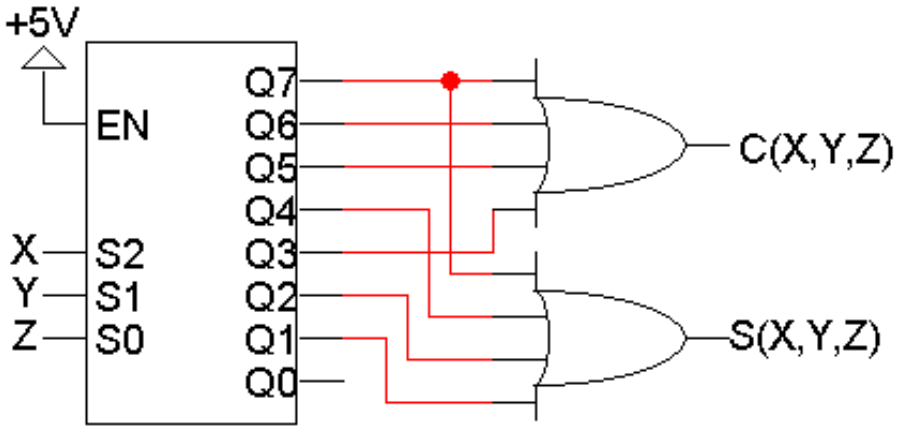
This is done using a “ROM-Mask” which contains all data that needs to be stored.

This mask is used during the fabrication of the integrated circuit

Basic ROM Structure

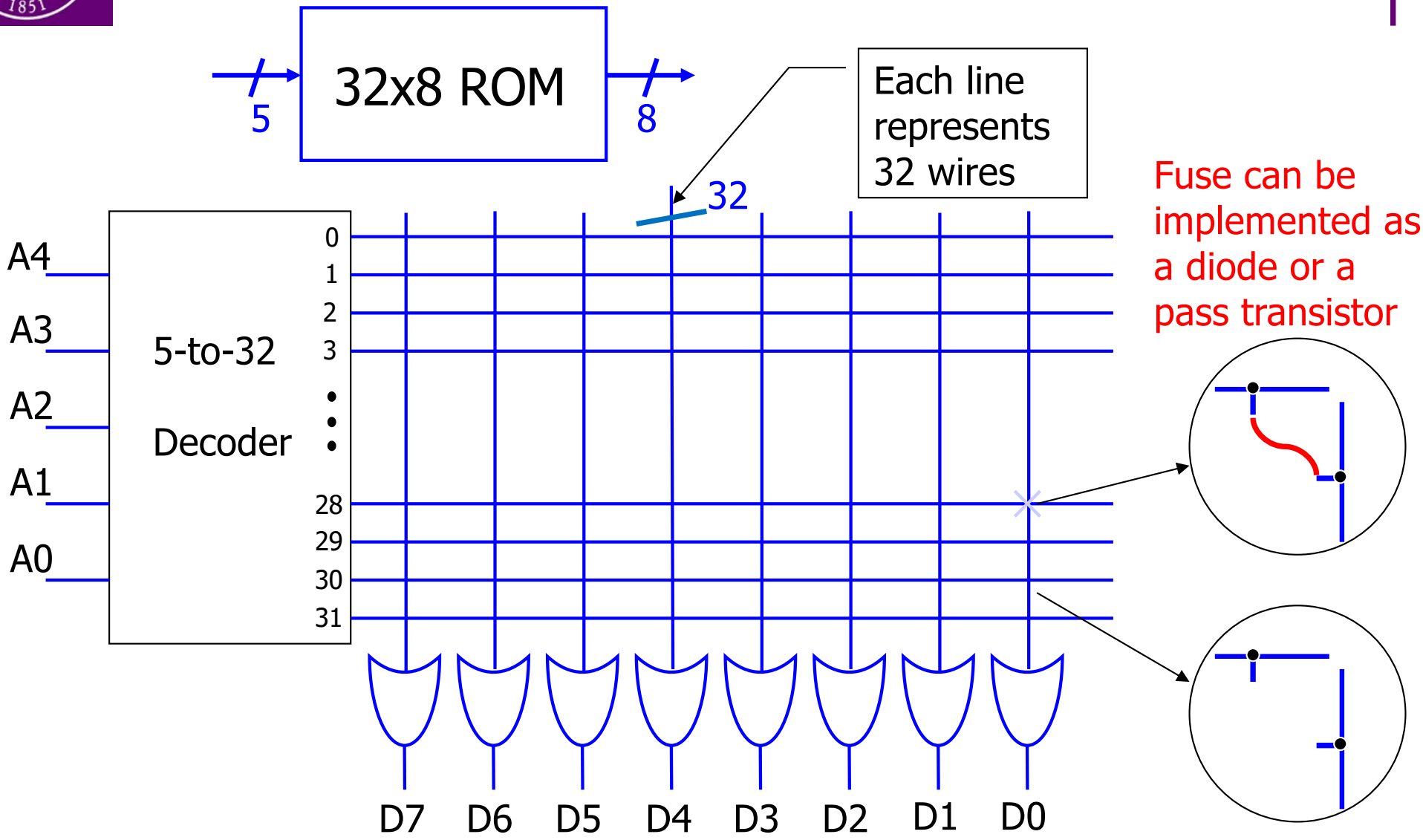


Leveraging decoders to store pre-calculated functions
Pre-stored Addition



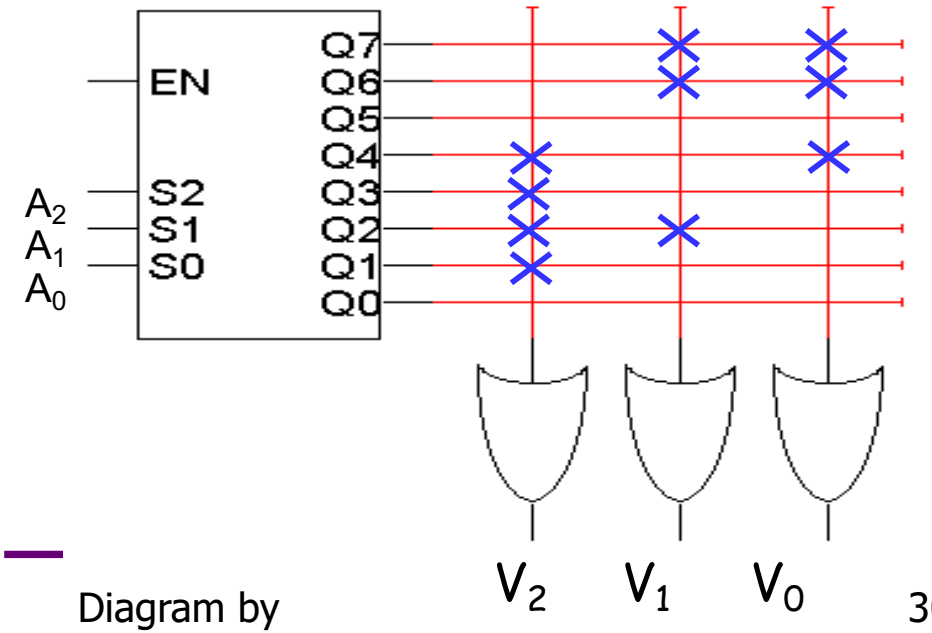
X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

32x8 ROM


32x8 ROM
5
8
Each line represents 32 wires
32
A4
A3
A2
A1
A0
5-to-32
Decoder
0
1
2
3
28
29
30
31
D7
D6
D5
D4
D3
D2
D1
D0
Fuse can be implemented as a diode or a pass transistor

Combinational circuit can be considered a ROM

- Stores eight words of data, each consisting of three bits.
- The decoder inputs act as an address, which refers to one of the eight available words.
- Every input combination corresponds to an address, which is retrieved to produce a 3-bit data output.



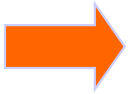
Address $A_2A_1A_0$	Data $V_2V_1V_0$
000	000
001	100
010	110
011	100
100	101
101	000
110	011
111	011

Diagram by Henry Hexmoor

Example: Lookup Table

- Design a square lookup table for $F(X) = X^2$ using ROM

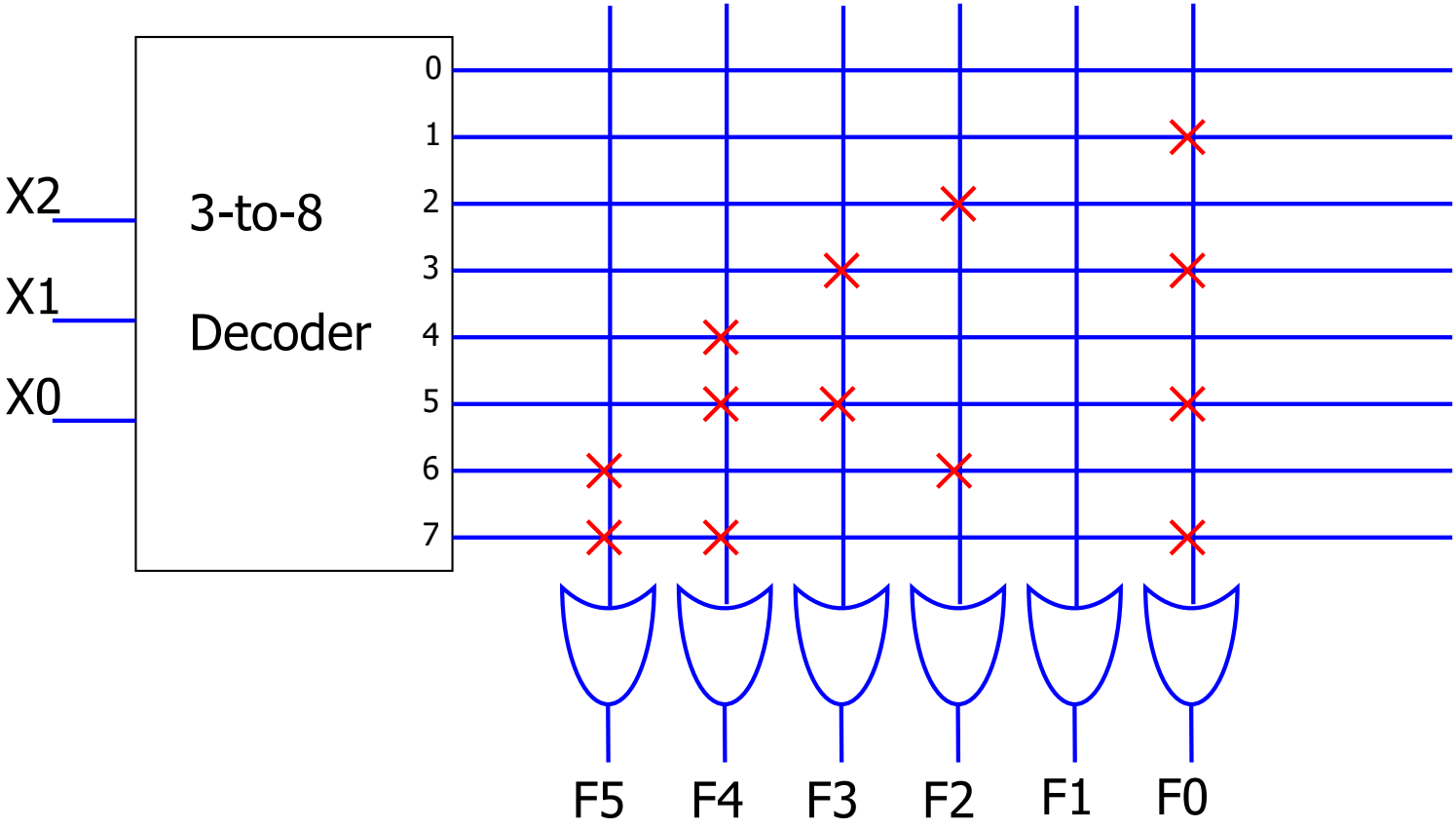
X	$F(X)=X^2$
0	0
1	1
2	4
3	9
4	16
5	25
6	36
7	49



X	$F(X)=X^2$
000	000000
001	000001
010	000100
011	001001
100	010000
101	011001
110	100100
111	110001

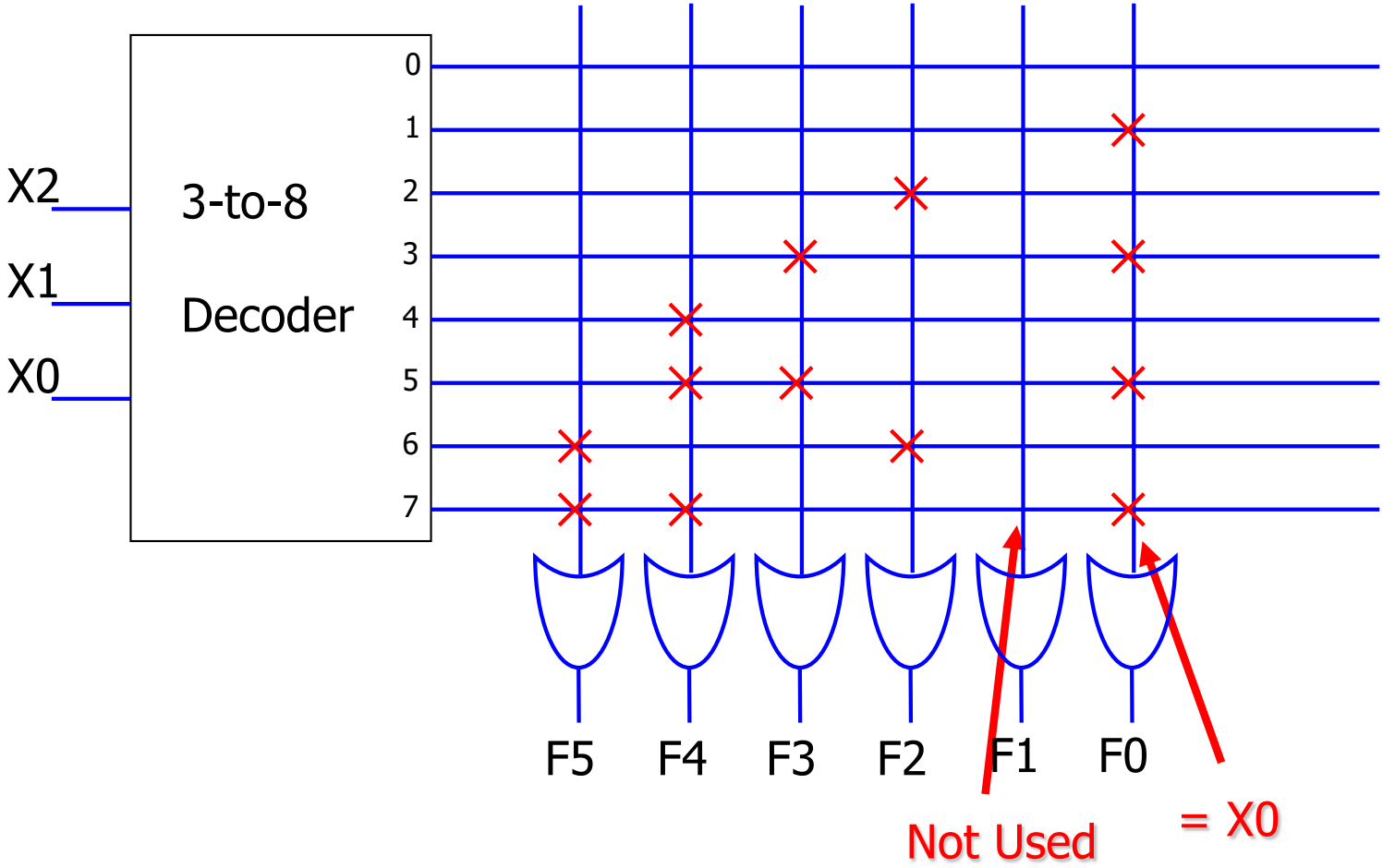
Square Lookup Table using ROM

X	$F(X)=X^2$
000	000000
001	000001
010	000100
011	001001
100	010000
101	011001
110	100100
111	110001



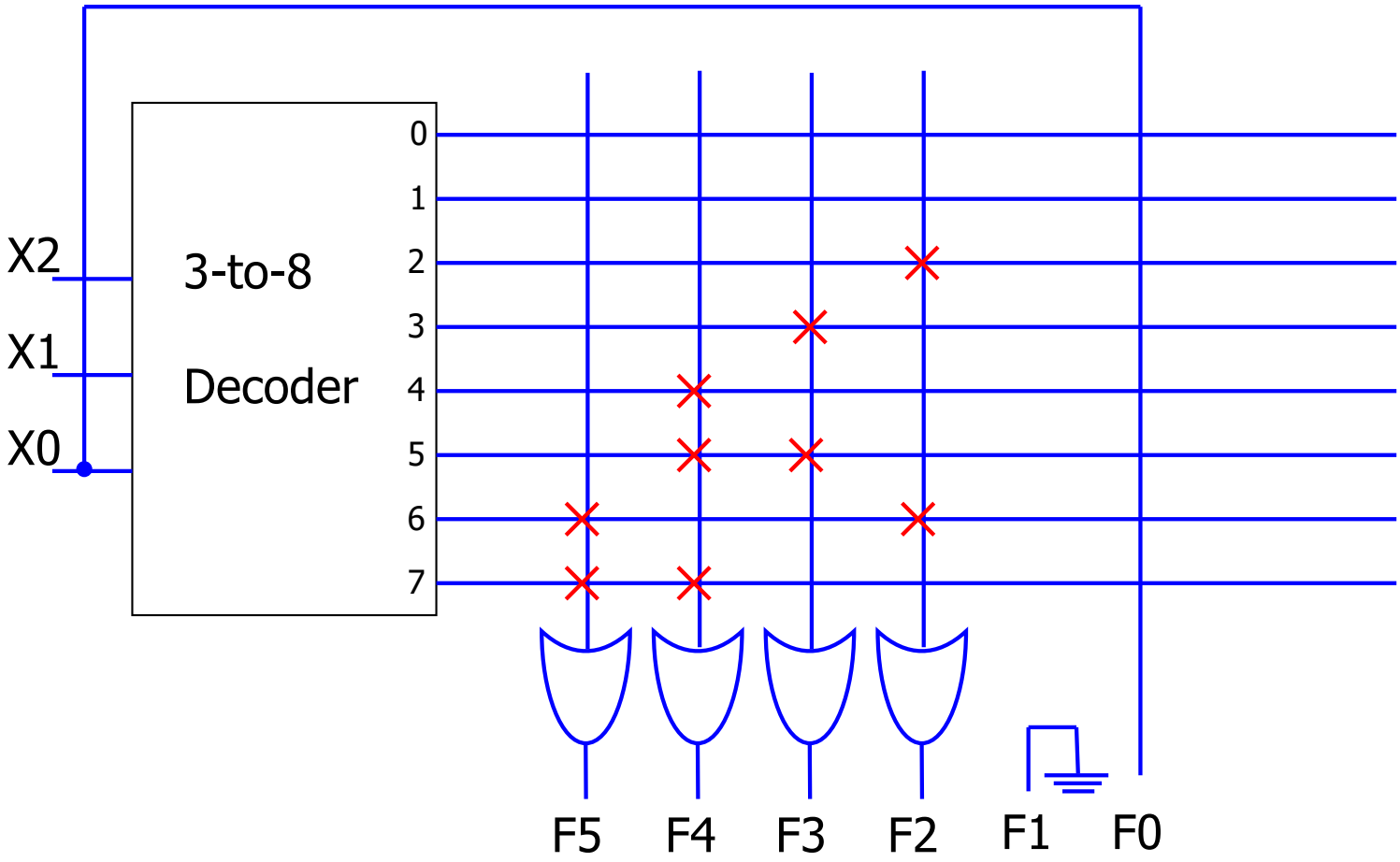
Square Lookup Table using ROM

X	$F(X)=X^2$
000	000000
001	000001
010	000100
011	001001
100	010000
101	011001
110	100100
111	110001

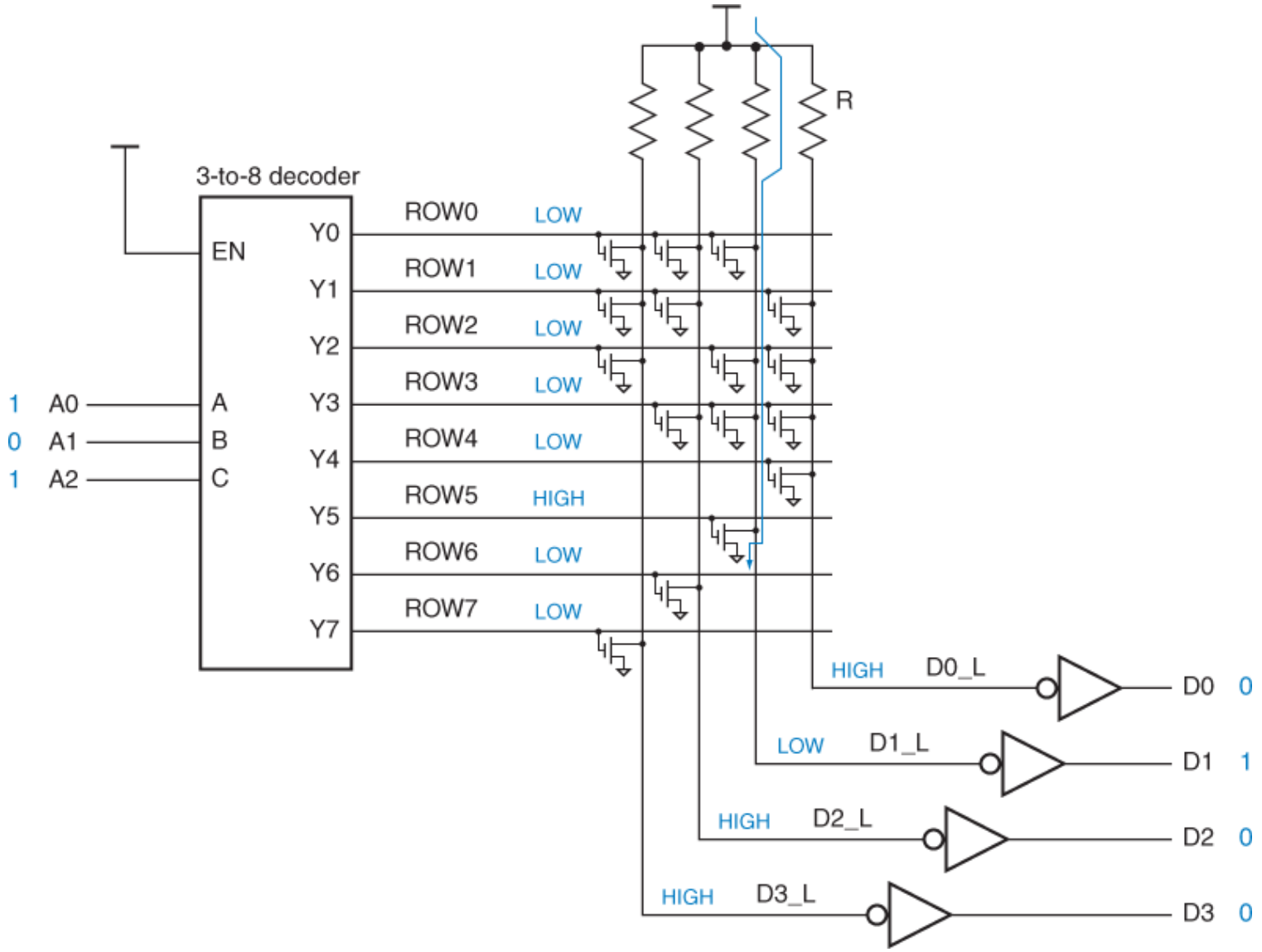


Square Lookup Table using ROM

X	$F(X)=X^2$
000	000000
001	000001
010	000100
011	001001
100	010000
101	011001
110	100100
111	110001



Simple ROM Architecture



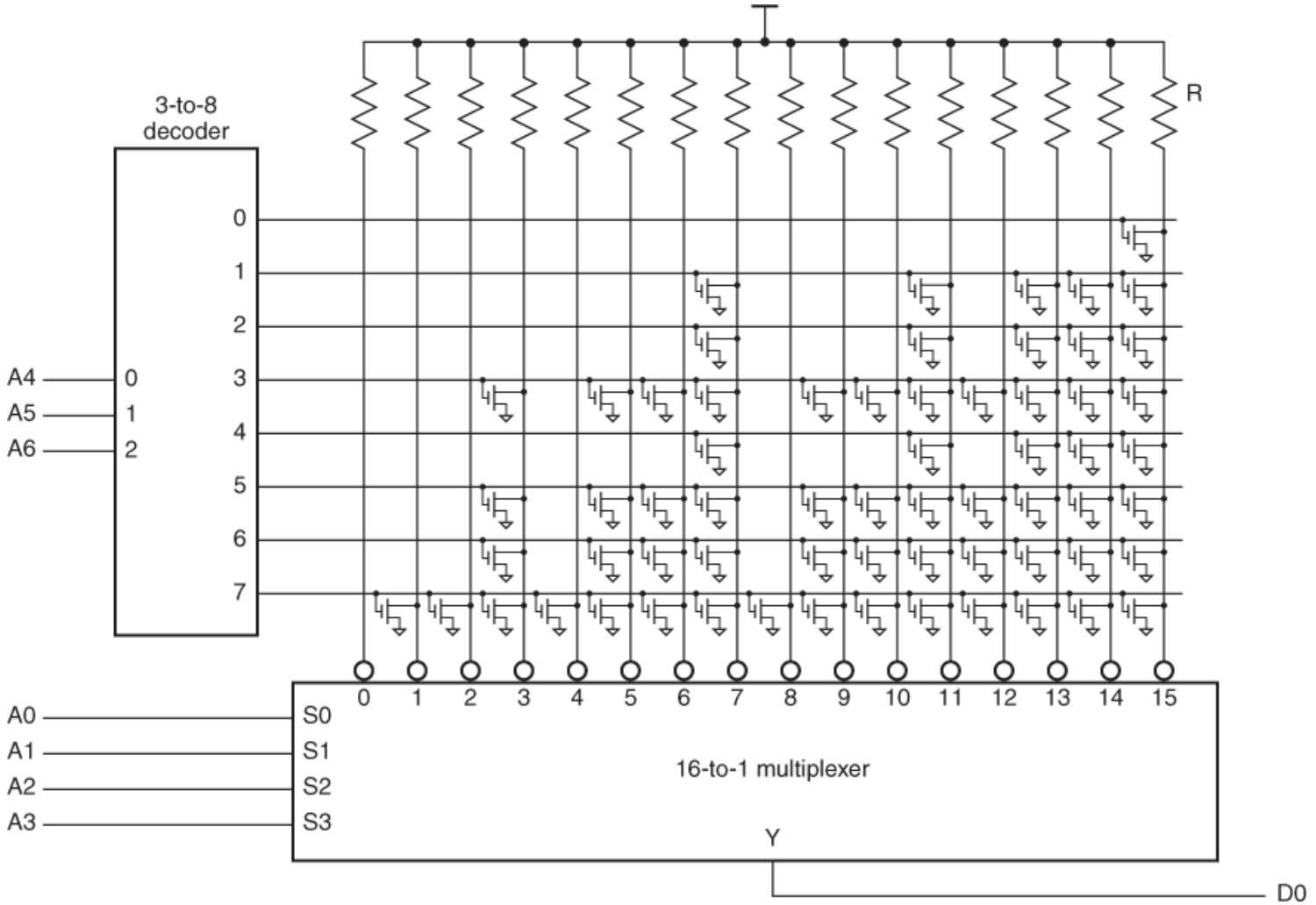
ROM Structure

- Each decoder output (ROW) is a called a word line
- Each vertical line is called a bit line
- Each bit line is connected to the supply through weak pull-up resistors
- Each intersection between a word line and bit line may contain a (pass) transistor (NMOS) or not
- The pattern of specific locations of the NMOS pass transistors correspond to the specific data content that is embedded into the ROM

Is this structure efficient?

- No!
- Building very large ROMs, containing millions of bits, gets challenging with this simple decoder structure
- A megabyte capacity ROM would require a 20x1,048,576 Decoder

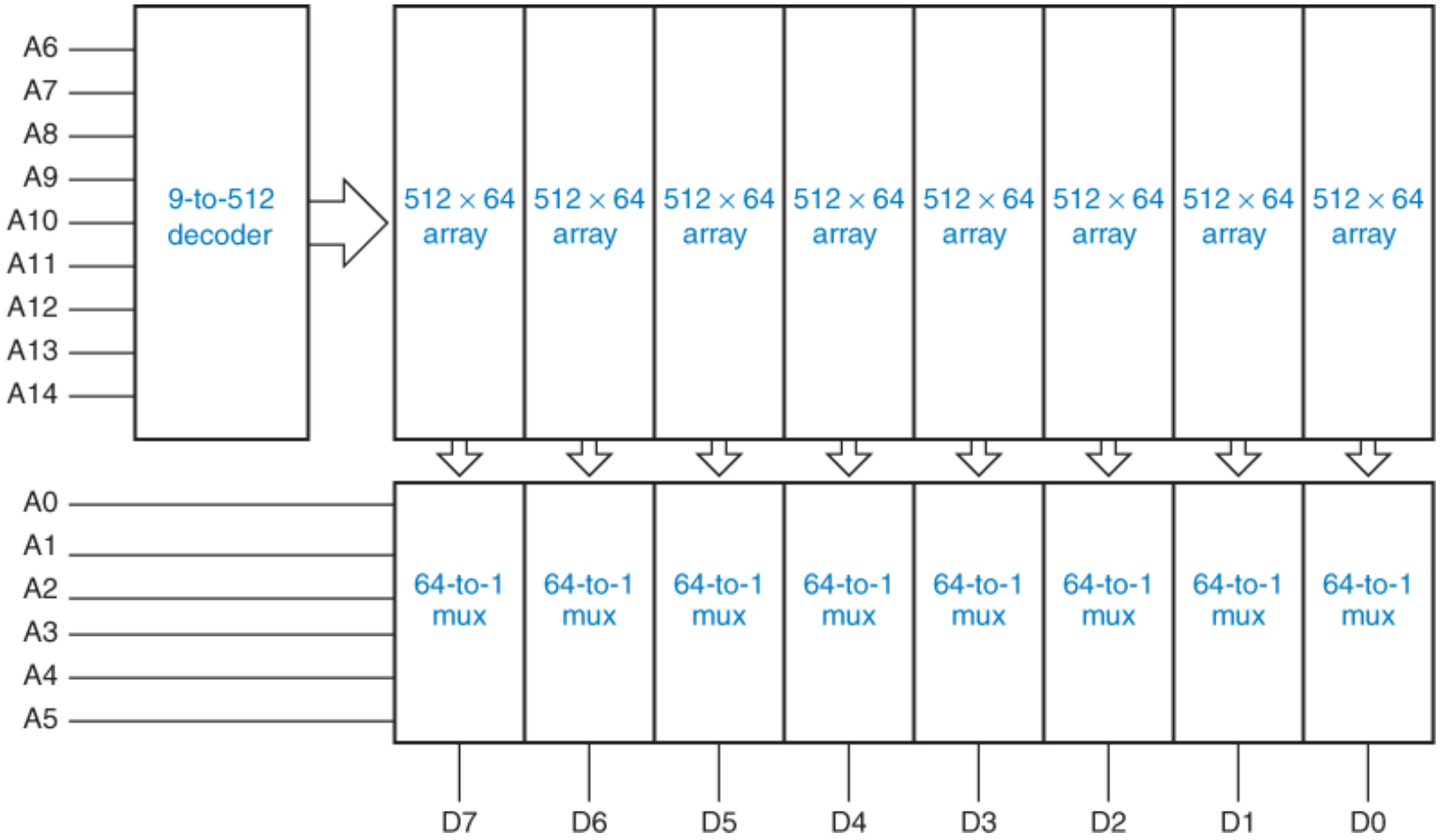
2 Dimensional Decoder



Efficient addressing with 2D Decoder

- In this example: 128x1 (2^7x1) Decoder
- Three highest order address bits control a 3x8 decoder
- Each decoder output "retrieves" all 16 bit values that would be located in all possible 16 locations addressed by the four lowest order address bits
- A 16x1 MUX selects the specific output bit as dictated by the value of the four lowest order address bits

Larger Decoders can be built by combining smaller building blocks: 32Kx8 ROM



Commercial ROMs

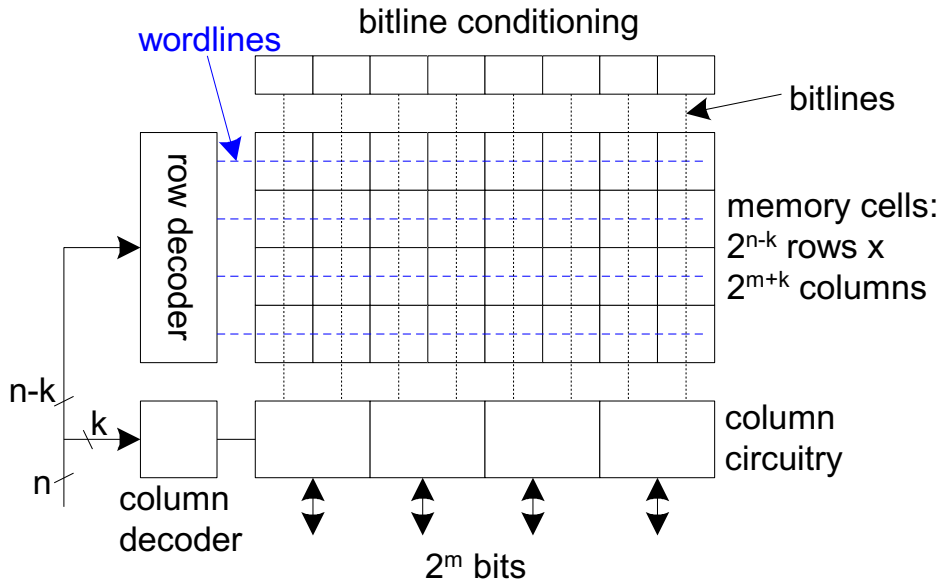
- ROM - Mask Programmable ROM
  - Specific pattern of NMOS transistors corresponding to data is created on the integrated circuit at production time
- PROM - Programmable ROM
  - All word/bit line intersections are manufactured with an NMOS in place
  - Select transistors can be enable/disables with a programming device by the customer using the ROM
  - Can be used for small volume manufactured products for specialization

Commercial ROMs

- EPROM
  - Erasable PROM
  - The selective enabling process can be reversed to default and then ROM can be reprogrammed
  - Device is exposed to UV light
  - The pass transistors are made with a special technology: Floating Gate MOS
- EEPROM / Flash Memory – Electrically Erasable
  - Uses Floating Gate transistors

Let's Generalize the Array Architecture

- 2^n *words* of 2^m *bits* each
- If n >> m, fold by 2^k into fewer *rows* of more *columns*



- Good regularity – easy to design
- Could become high density, if good cells are used

Classification on Memory Technologies

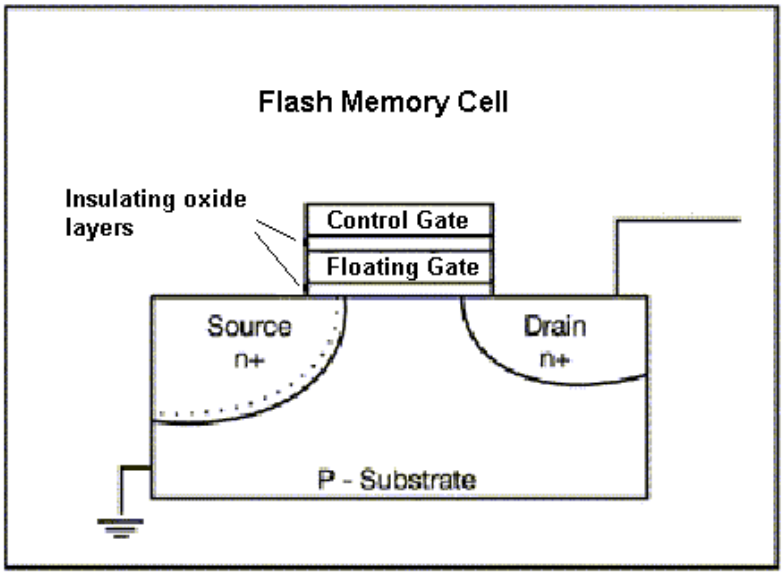
- Access Capability
  - Read-Only (ROM)
  - Random Access – dynamic ability to both Read and Write
- Volatility
  - Non-Volatile
    - Retains data even if powered off
    - We will discuss Flash Memory soon as an example
  - Volatile
    - Loses data if power off or over time (even if powered on)
    - **Static Memory**: retains data as long as powered ON
    - **Dynamic Memory:** loses data even if constantly powered ON (needs additional refresh mechanisms)
- Performance/Density
  - Unit chip area needed per bit/Kb/Mb, etc.
  - Time to access a block of data

FLASH Memory

- Like a normal transistor but:
  - Has a floating gate that can hold charge
  - To write: raise or lower word line high enough to cause charge to tunnel
  - To read: turn on word line as if normal transistor
    - presence of charge changes threshold/conductivity of transistor and thus measured current
- Two varieties:
  - NAND: denser, must be read and written in blocks
  - NOR: much less dense, fast to read and write

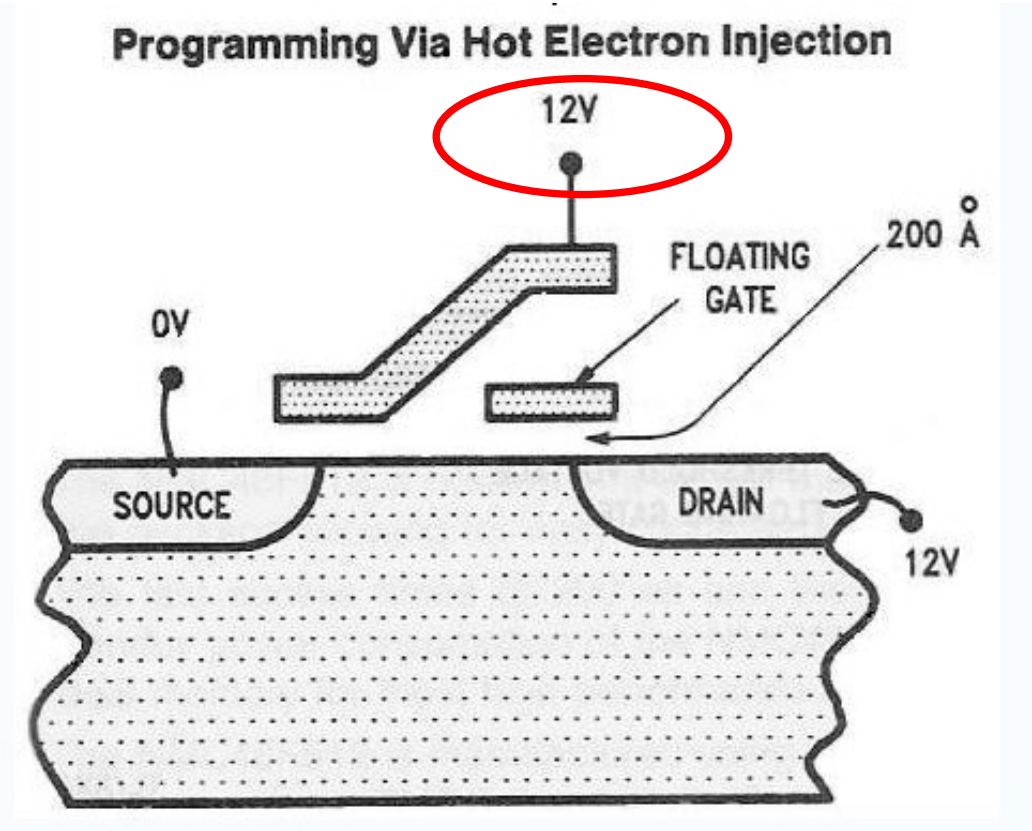
Device Structure

- Similar to MOS Transistor Structure with added Floating Gate (FG) between Control Gate (CG) and channel layer
- FG surrounded by insulators, can trap electrons "permanently" (~50 years)
- CG is same as an ordinary MOS transistor
- Charged FG disrupts / affects the conducting channel



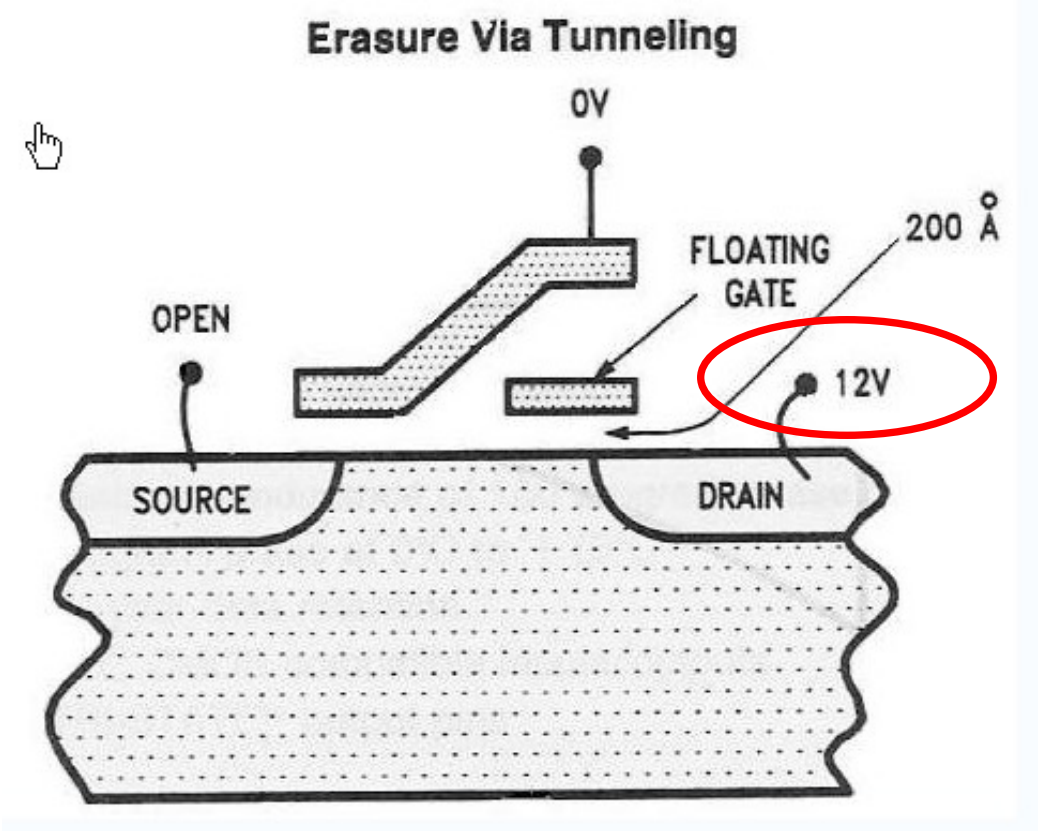
If FG is charged it will prevent a conducting channel through the transistor, so the transistor will stay OFF even if a positive voltage is applied at the Control Gate

Floating Gate: Programming



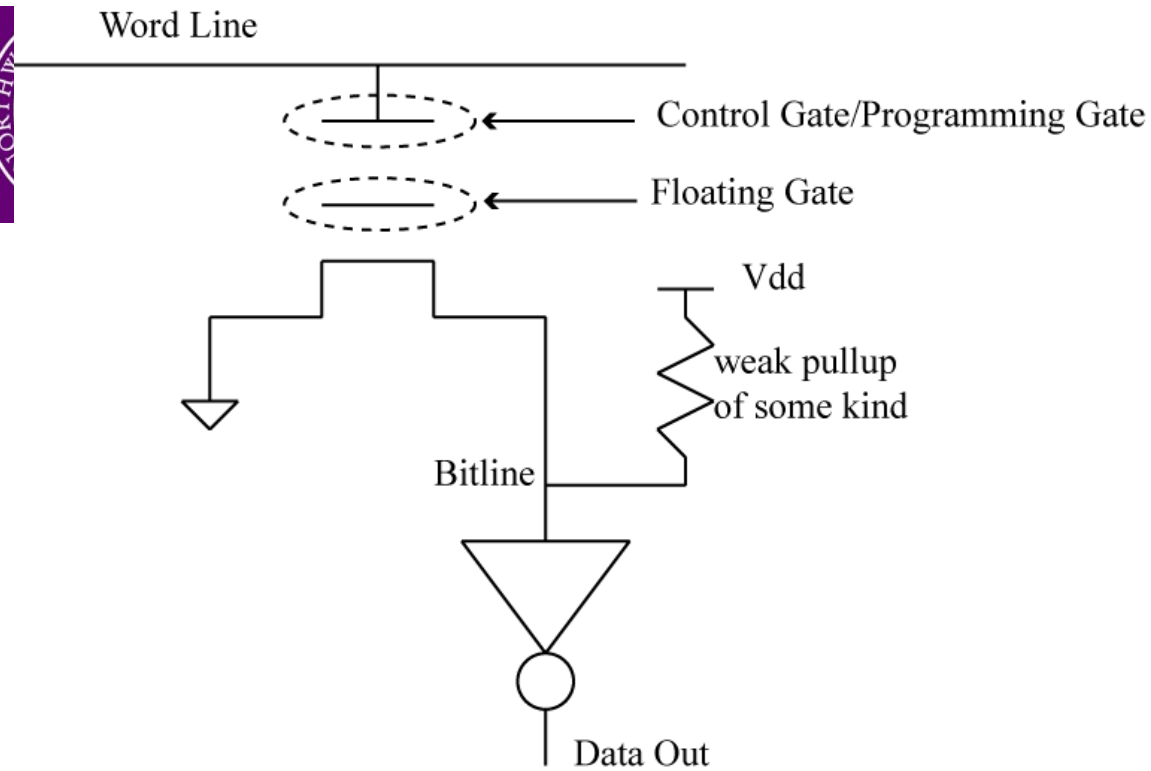
Electrons flowing in channel from drain to source attracted to the very high voltage +12V on the Programming/Control gate, become trapped within the Floating Gate, giving it a negative charge accumulation.

Floating Gate : Erase



+12V on drain attracts trapped charge in the Floating Gate and removes the electrons.

Floating Gate Operation



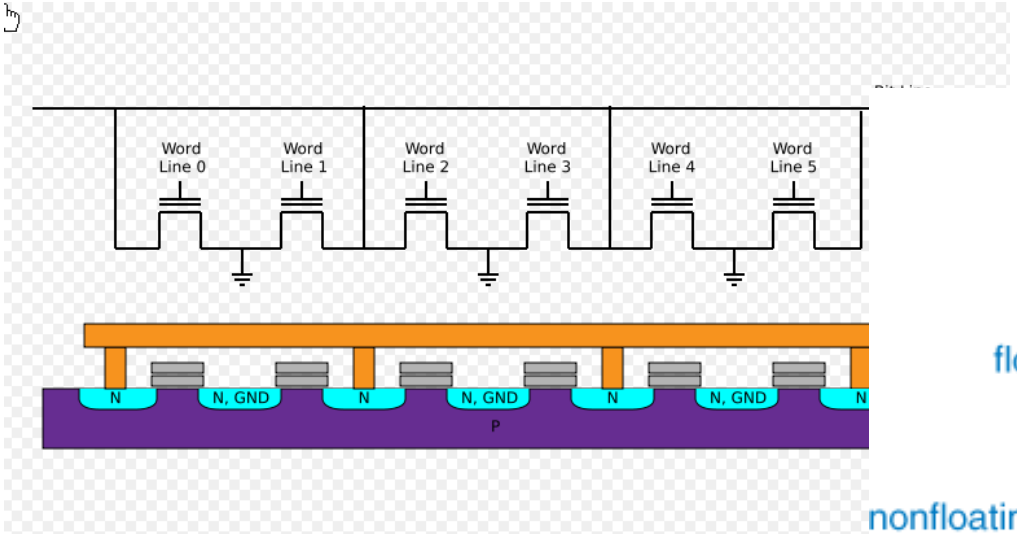
When Floating Gate has no electrons trapped (Bit location storing 1) , then WL = '1' turns on transistor, pulling Bitline low, and Data Out = '1'

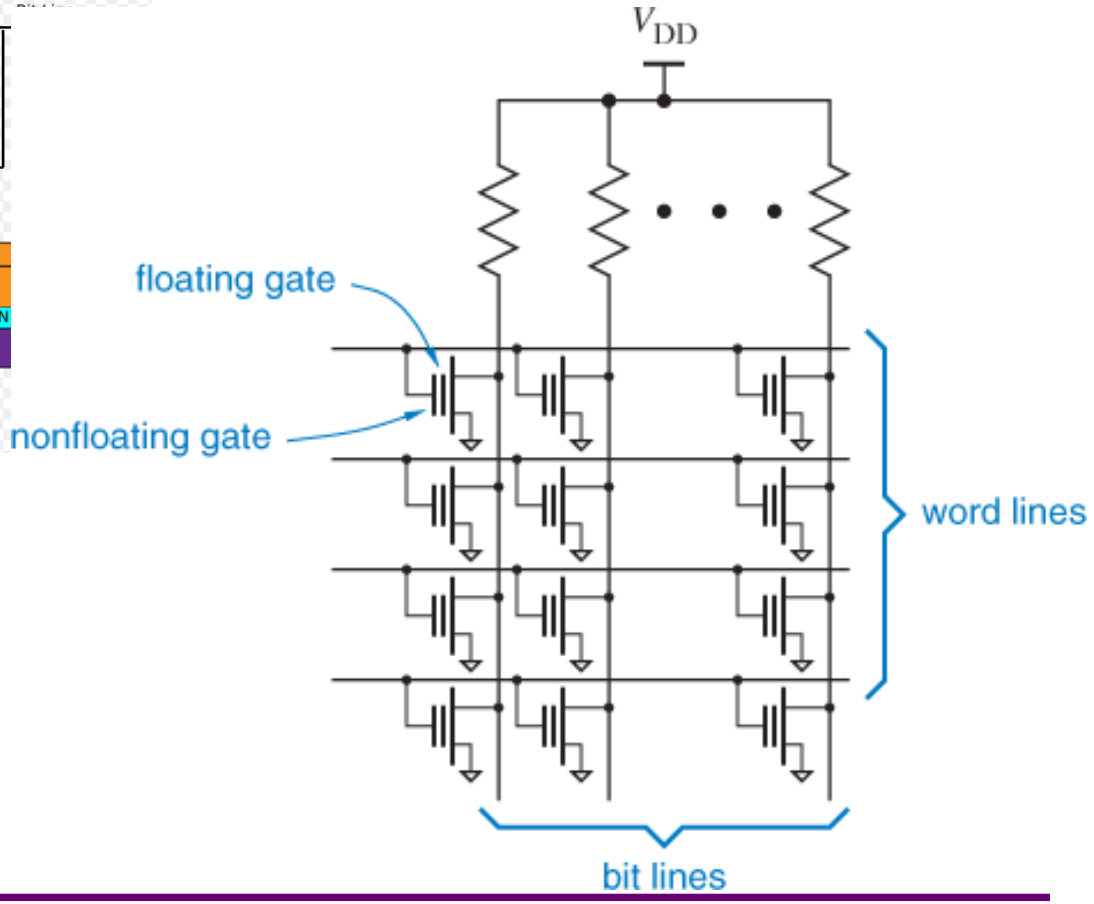
Unprogrammed → WL=1 → transistor on as floating gate has no effect → Bitline = 0 → Data Out = 1

When Floating gate has electrons trapped on it (negative charge), this increases the Vt of the transistor, so now transistor remains off when WL is 1. (Bit location storing 0)

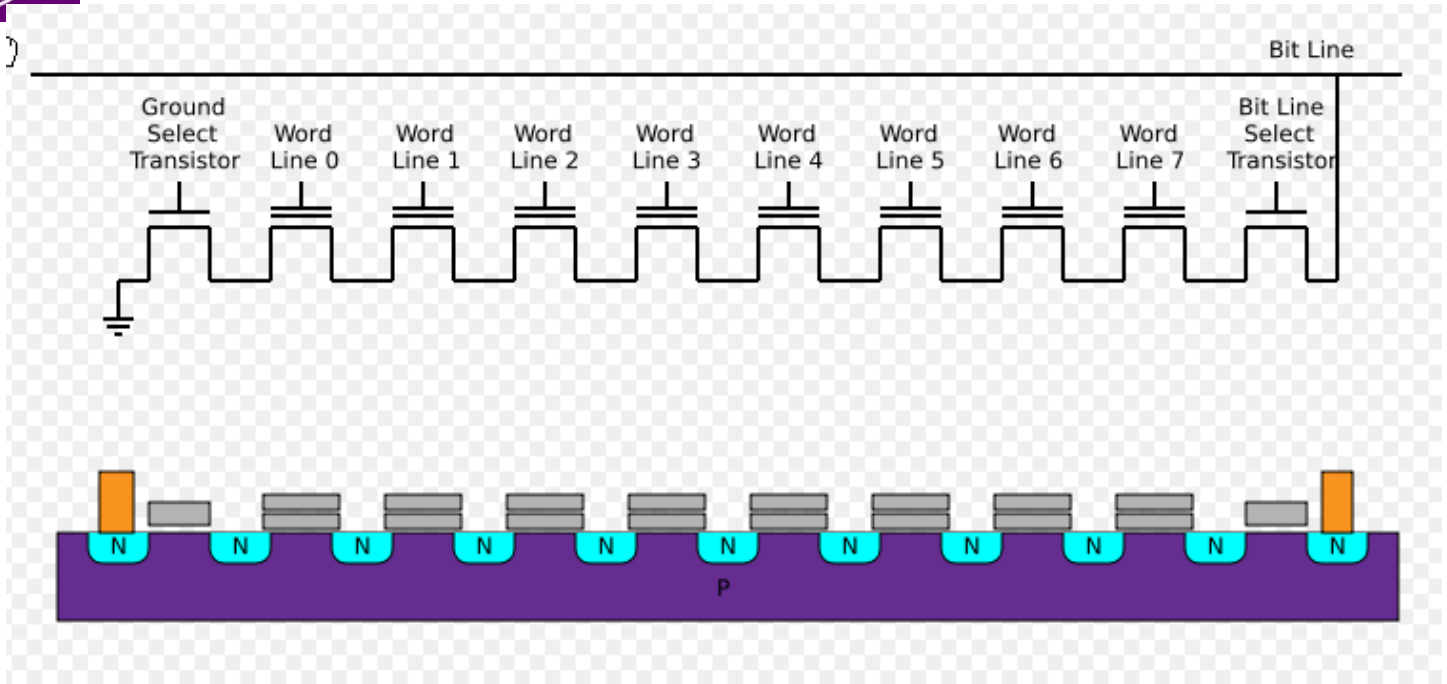
Programmed → WL=1 → transistor off as negative charge on floating gate raises Vt → Bitline = 1 → Data Out = 0

NOR Flash – any transistor can singlehandedly pull down the bit-line (like NOR logic function)





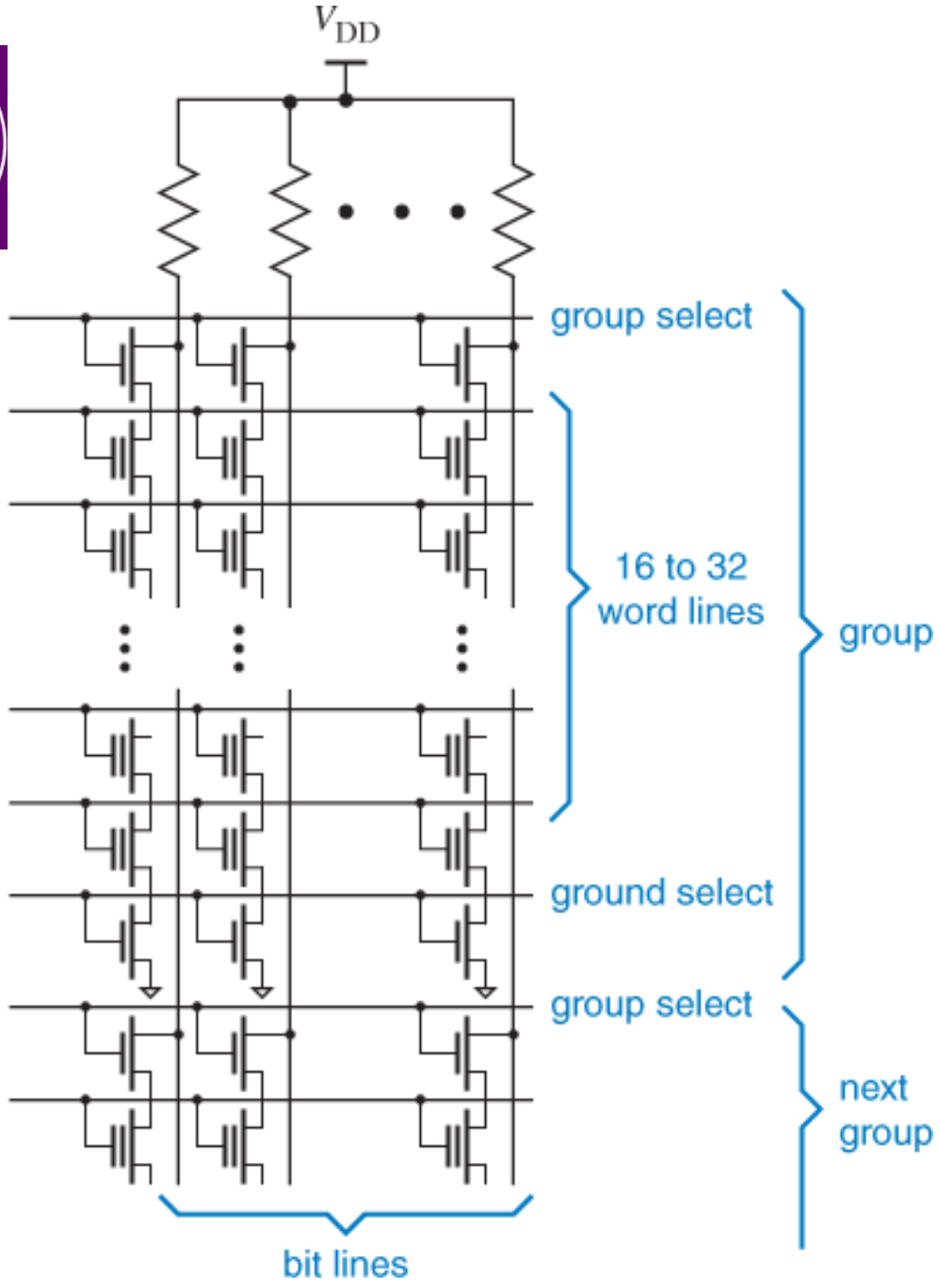
NAND Flash



Standard NAND flash has a group of 8-32 transistors in series, chained with one preceding transistor acting as Group Select (Bit Line Select) and one final transistor acting as Ground Select

While one cell is accessed the rest act as pass transistors

NAND Architecture



NAND Architecture

- The Floating Gate and its effect on the conductivity of the MOS switch is slightly modified:
  - Even if a FG is charged to store a 0 in a particular transistor, it is still capable of conducting a small amount of current (can turn ON) if the Word line is HIGH
  - Any transistor will be able to conduct if it stores a 0, even if the Word Line is LOW

NAND Architecture

- To read a particular word:
  - Select a particular Group
  - Set all Word Lines in that Group HIGH, except for the target word that is desired to be retrieved
  - Current will pass through all transistors in all word lines that are HIGH
  - Current will pass/or not pass through some transistors along that selected Word

Read Function

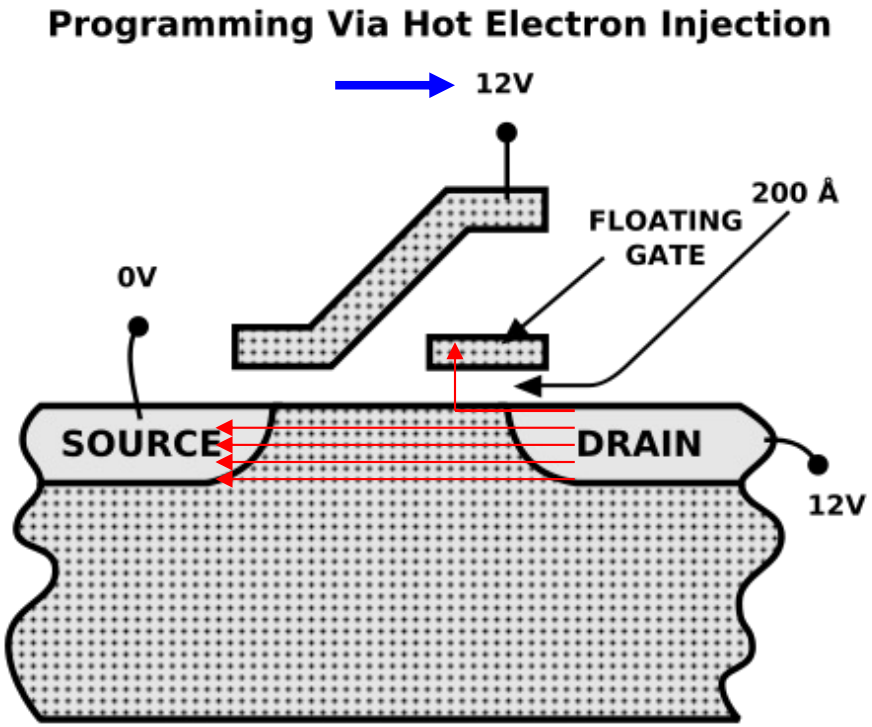
Logic state determined by current flow amount

- Let I_D be the current flow through a normal MOSFET
- Let I_{DF} be the current flow through a "flash transistor"
  - If $I_{DF} \approx I_D \rightarrow$ LOGIC 1
  - If $I_{DF} < I_D$ (significantly less than) $\rightarrow$ LOGIC 0

Flash Memory senses the amount of current flowing through its transistors' channels as a means of logic state determination.

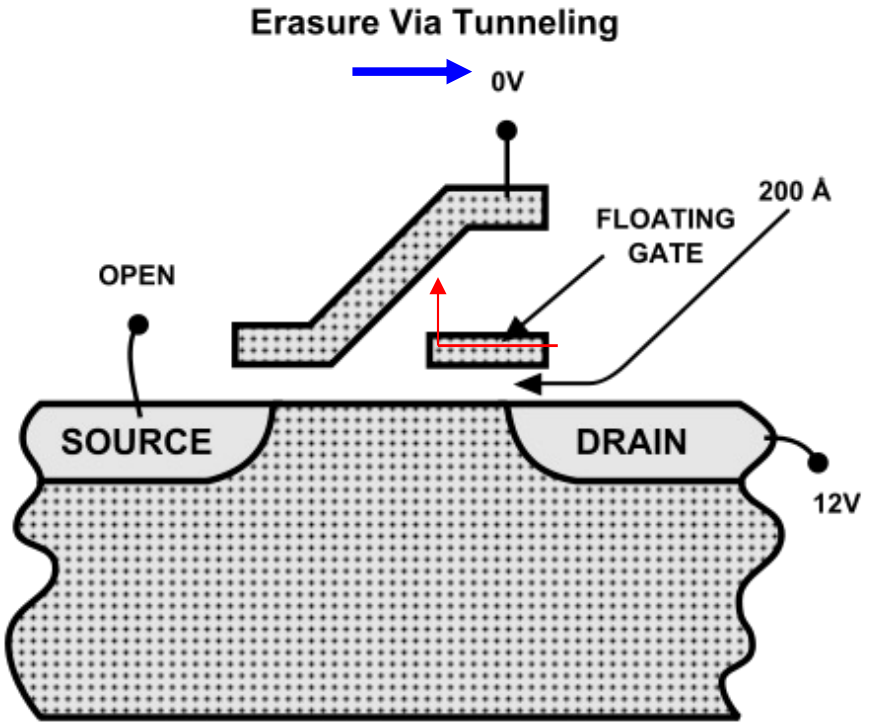
Write Function – Logic 0

- Electrons are injected into the FG via **hot-electron injection**
- The channel of the transistor becomes "harder" to conduct (higher V_t), changing current flow (lesser flow)
- Reduced current flow in inversion layer - logic 0



Write Function – Logic 1

- Electrons are **tunneled** out of the FG
- FG no longer partially cancels the Control Gate's E-field
- V_t (conductivity of channel) is back to default
- Current flow in inversion layer returns to normal - logic 1



Lifespan

- Hot electron injection or tunneling results in device deterioration
- Electrons have a probability of becoming trapped in oxide layer, electron traps
- Trapped electrons in oxide disrupt V_t
- Flash Memory can “wear out”
- Between 1,000 10,000 and up to 100,000 write cycles

NAND vs. NOR Flash

- NAND denser than NOR because of smaller cell size
  - preferred for data memory storage (flash drives)
  - NAND is about 2x denser than NOR because of layout efficiencies due to series transistors (less metal contacts)
- NAND slower read time because of stacked cell arrangement
  - NOR preferred as program memory storage for microcontrollers because of faster access time
- NAND is always block read/block write; NOR allows read of individual memory locations
- NAND programming/erase is faster than NOR programming/erase

NAND/NOR Flash Limitations

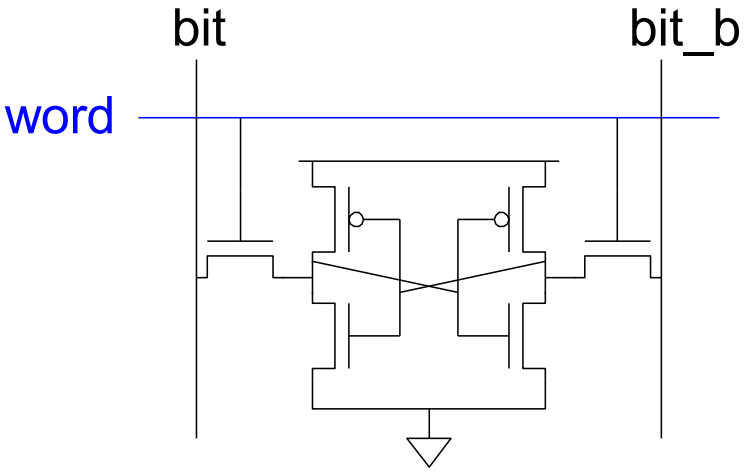
- Both NAND/NOR memory has limited number of programming/erasure cycles
- About 100,000 cycles is a typical number, even though cells with higher cycle numbers can be designed
  - The Data memory EEPROM cells (100K minimum cycles, 1M max) in the PIC microprocessors are designed to have about 10X the maximum number of programming/erase cycles as the program memory EEPROM cells (10K minimum cycles, 100K max)

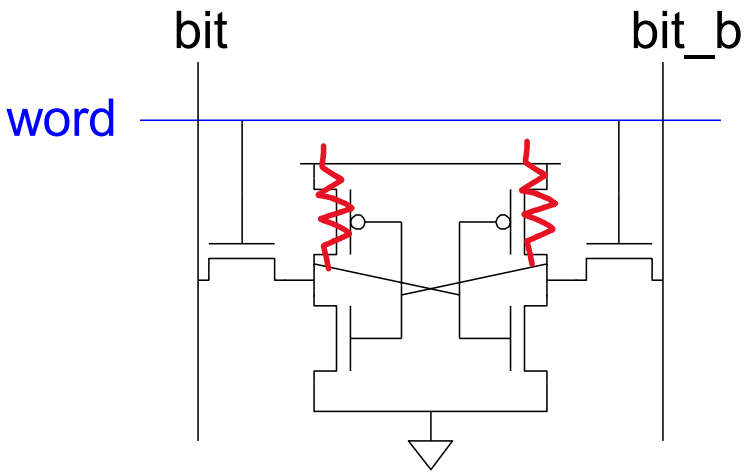
Static Random Access Memory (SRAM)

- Storage with continuous updates, reads and writes allowed
- Access to an arbitrary location allowed (via addressing)
- Memory arrays can be large
  - Need to optimize cell design for area and performance, and high density
  - Peripheral circuits can be complex
    - 60-80% area to store bits, 20-40% in periphery, i.e., address decoding, reading of bits, etc.

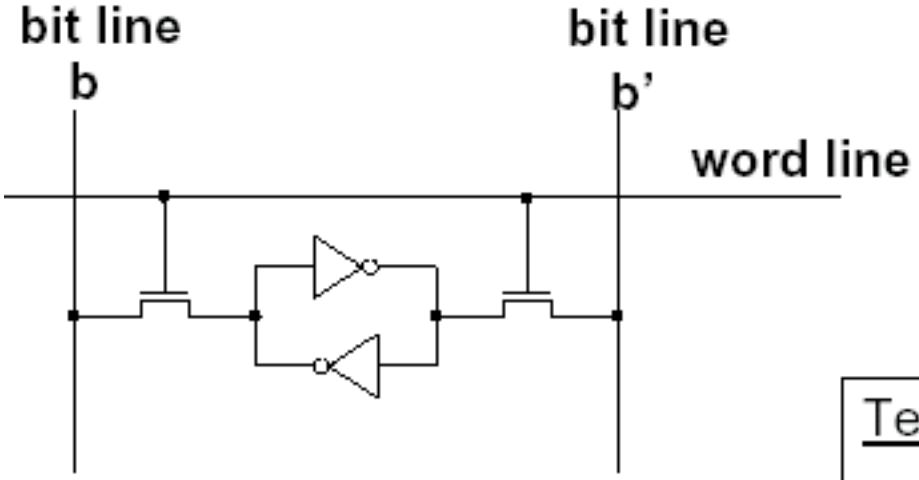
Static Random Access Memory (SRAM)

- Classical SRAM Memory cell design
  - 6T cell full CMOS
  - 4T cell
    - Replaces PMOS transistors with high resistance poly material





Anatomy of the SRAM Cell



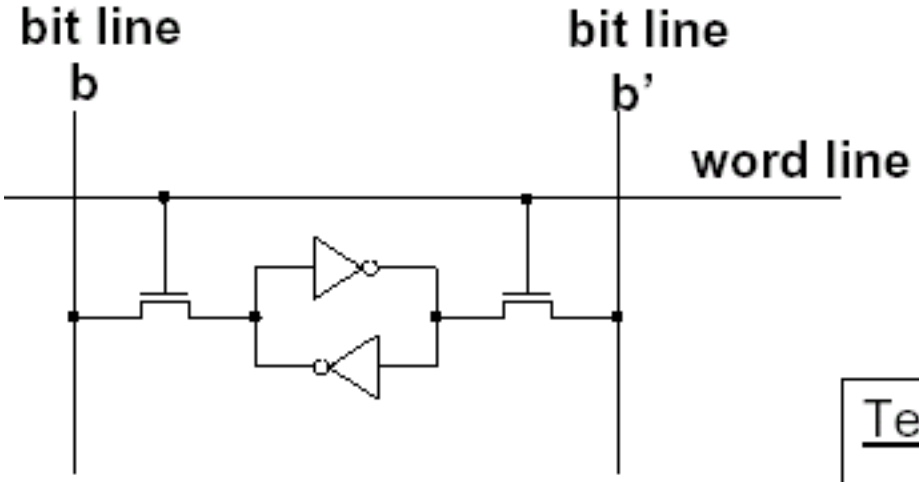
Stable Configurations

Terminology:

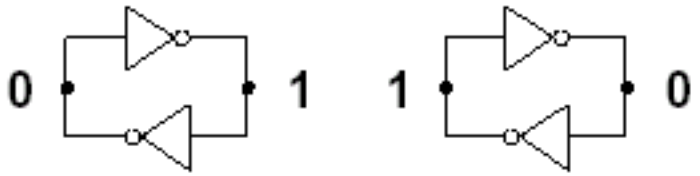
bit line:	carries data
word line:	used for addressing

Static: holds data as long as powered ON
Volatile: cannot hold data if powered OFF
Three modes of operation **Write Read Retain**

Anatomy of the SRAM Cell



Stable Configurations



Terminology:

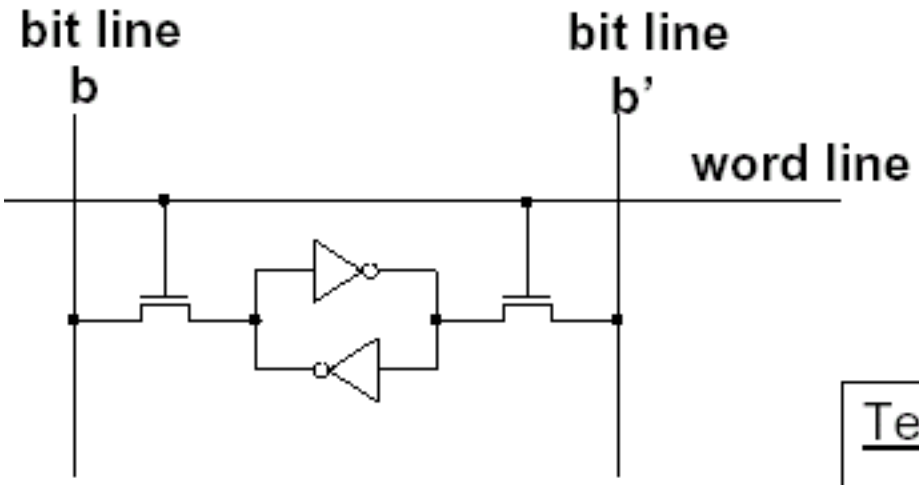
bit line: carries data

word line: used for addressing

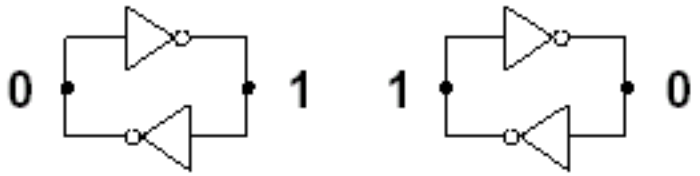
Write:

- **set Bit Lines to new data value to be stored**
- **raise Word Line to "high" '1'**
  - **sets cell to new state**

Anatomy of the SRAM Cell



Stable Configurations



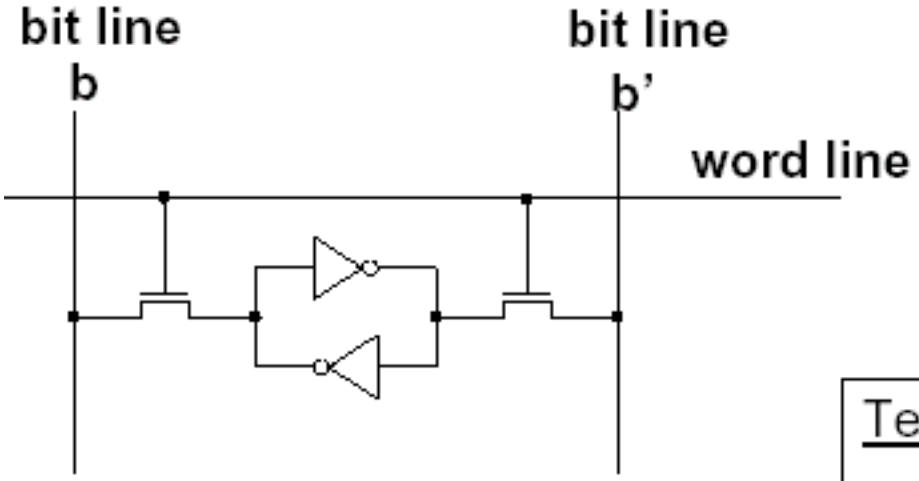
Terminology:

bit line:	carries data
word line:	used for addressing

Retain:

- **Word Line = 0**
- **Access Transistors are OFF**
- **Data Held within the inverter cycle**

Anatomy of the SRAM Cell



Stable Configurations

Terminology:

bit line:	carries data
word line:	used for addressing

Read:

- **set Bit Lines HIGH**
- **set Word Line HIGH**
- **detect which bit line drops to LOW**

Reading Data from SRAM

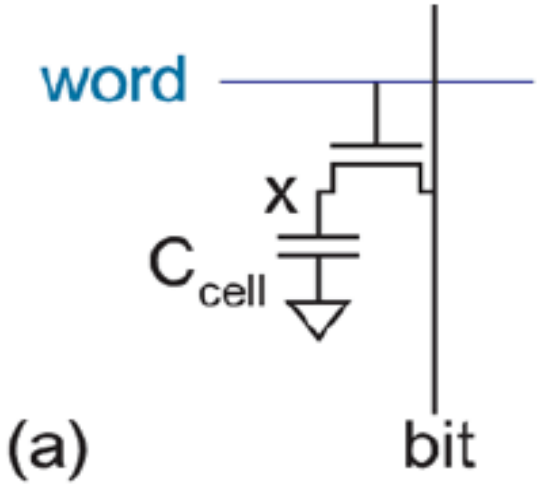
- Utilizes a structure called Sense Amplifier
  - It's an analog circuit: a simple differential amplifier
  - comparing the difference between bit and bit_bar
  - if bit>bit_bar, output is 1
    - if bit<bit_bar, outputis 0
    - allows output to be set quickly without fully charging/discharging bitline

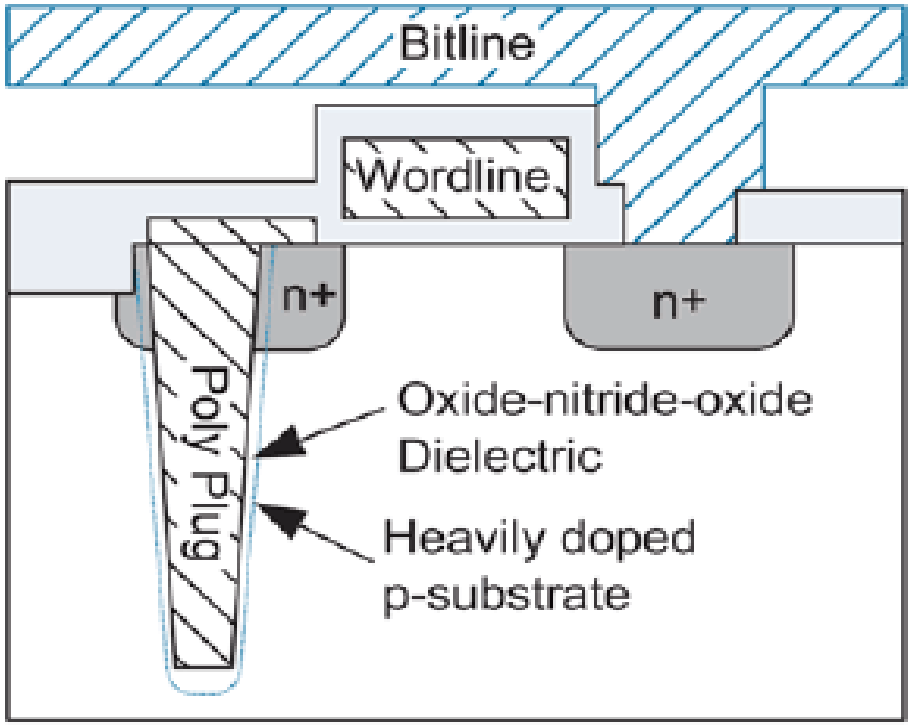
SRAM

- A good trade-off point between speed of access and density
- Used in building
  - Register Files and Cache Memories in computer systems
  - Local memory blocks in ASICs
  - FPGAs
- Take up significant area in a processor chip

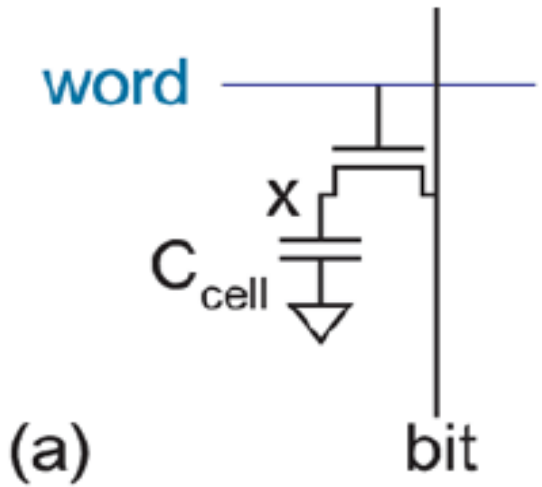
DRAM: Dynamic RAM

- Stores its contents as charge on a capacitor rather than in a feedback loop
- 1T dynamic RAM cell has a transistor and a capacitor

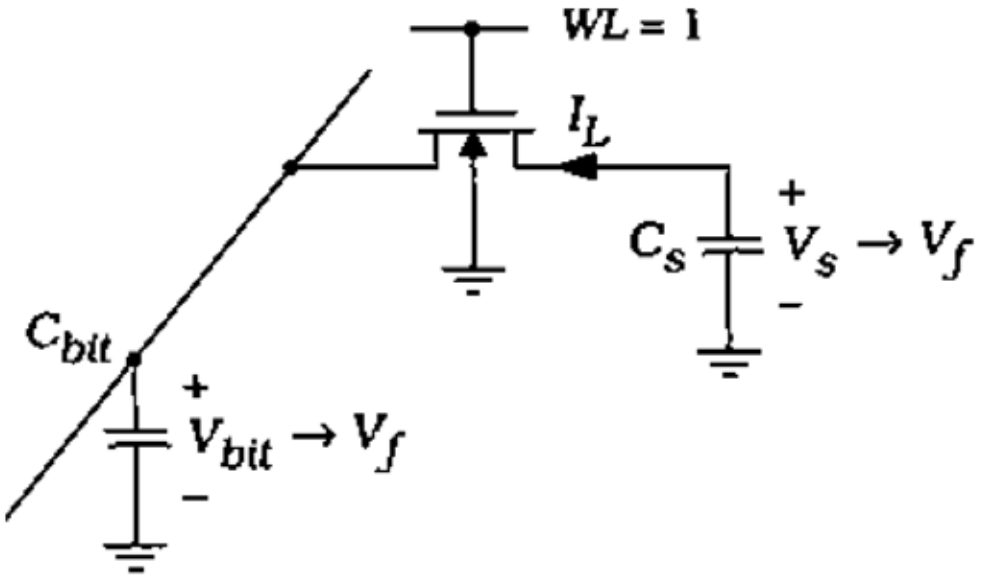




DRAM Read



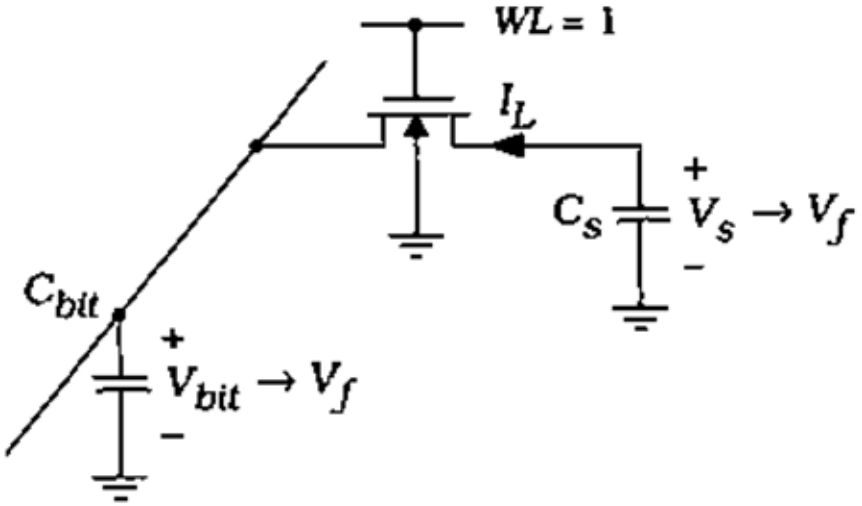
1. bitline precharged to $V_{DD}/2$
2. wordline rises, capacitor shares its charge with bitline, causing a voltage ΔV
3. read disturbs the cell content at x, so the cell must be rewritten after each read



DRAM Read

Charge Redistribution –
initial charge on C_S: Qcell=C_S*VDD redistributed on C_{bit} until

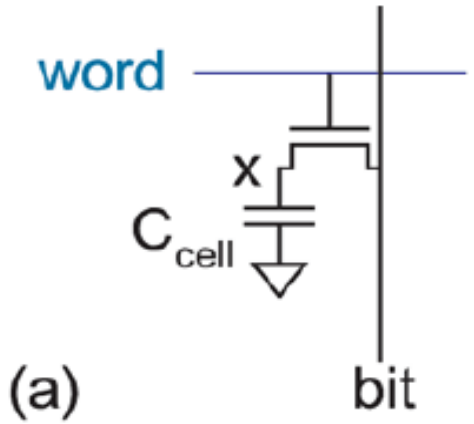
- $V_{bit} = V_s = V_f$ (final voltage)
  - $Qs=C_sV_f+C_{bit}V_f$
  - $CsV_s=V_f(C_s+C_{bit})$



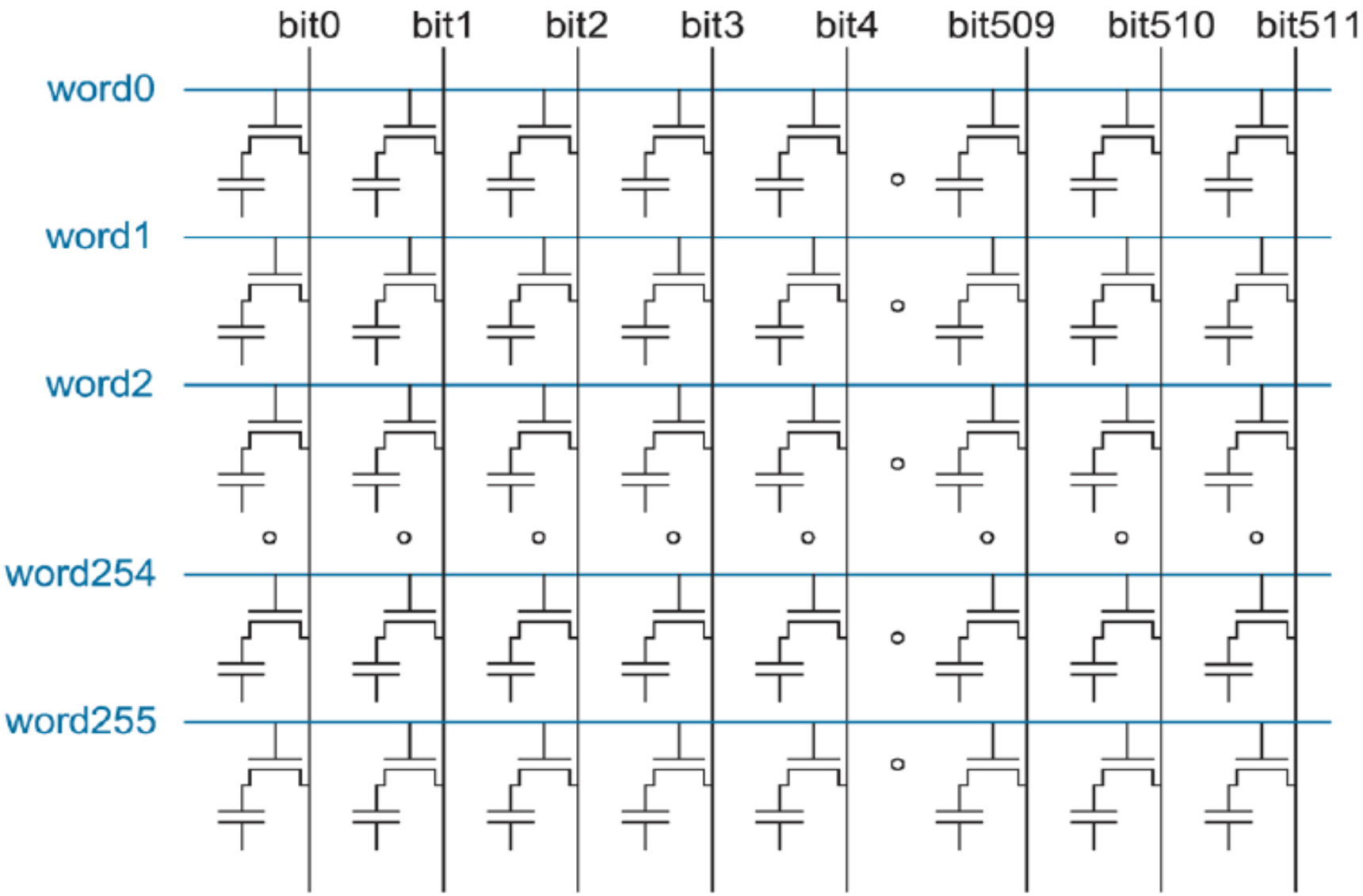
- V_f typically very small and requires a good sense amplifier

DRAM write

During WRITE, the bitline is driven high or low and the voltage is forced to charge (or discharge) the capacitor



DRAM Array


bit0
bit1
bit2
bit3
bit4
bit509
bit510
bit511
word0
word1
word2
word254
word255

DRAM

- Bitline capacitance is an order of magnitude larger than the cell, causing very small voltage swing
- A sense amplifier is used to detect that small fluctuation, which represents the "data being READ"

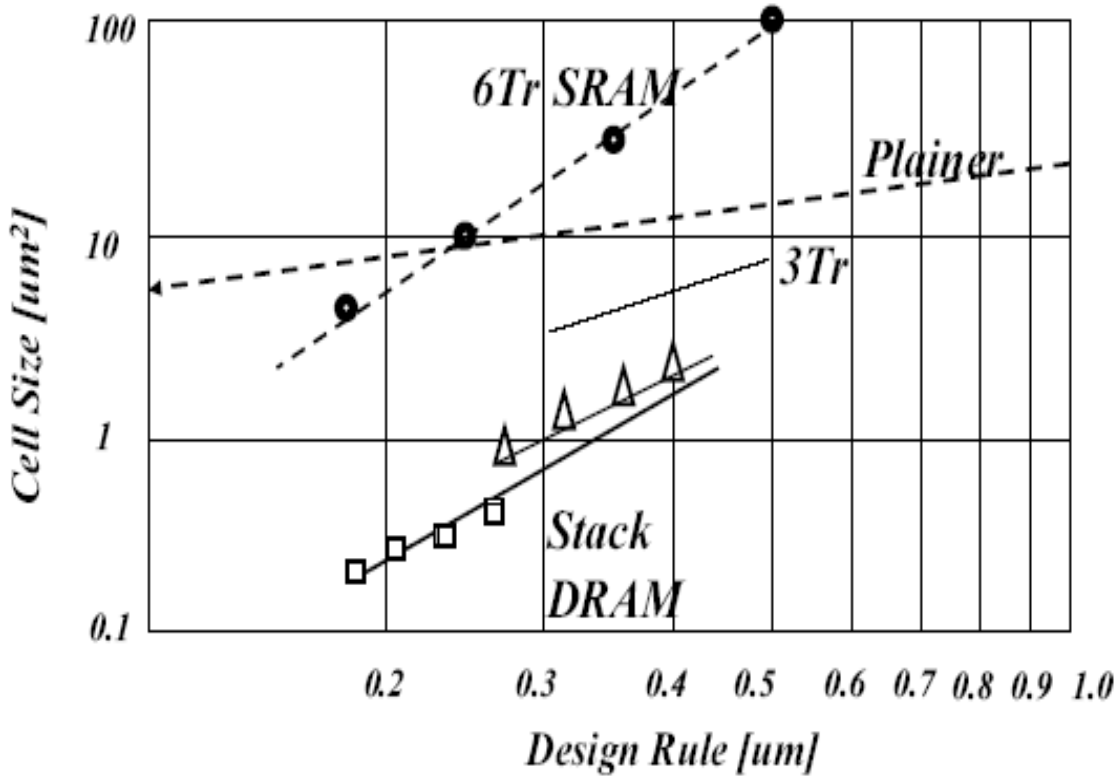
DRAM in a nutshell

- Based on capacitive (non-regenerative) storage
- Highest density (Gb/cm2)
- Large external memory (Gb) for processors or embedded DRAM for image, graphics, multimedia ASIC chips

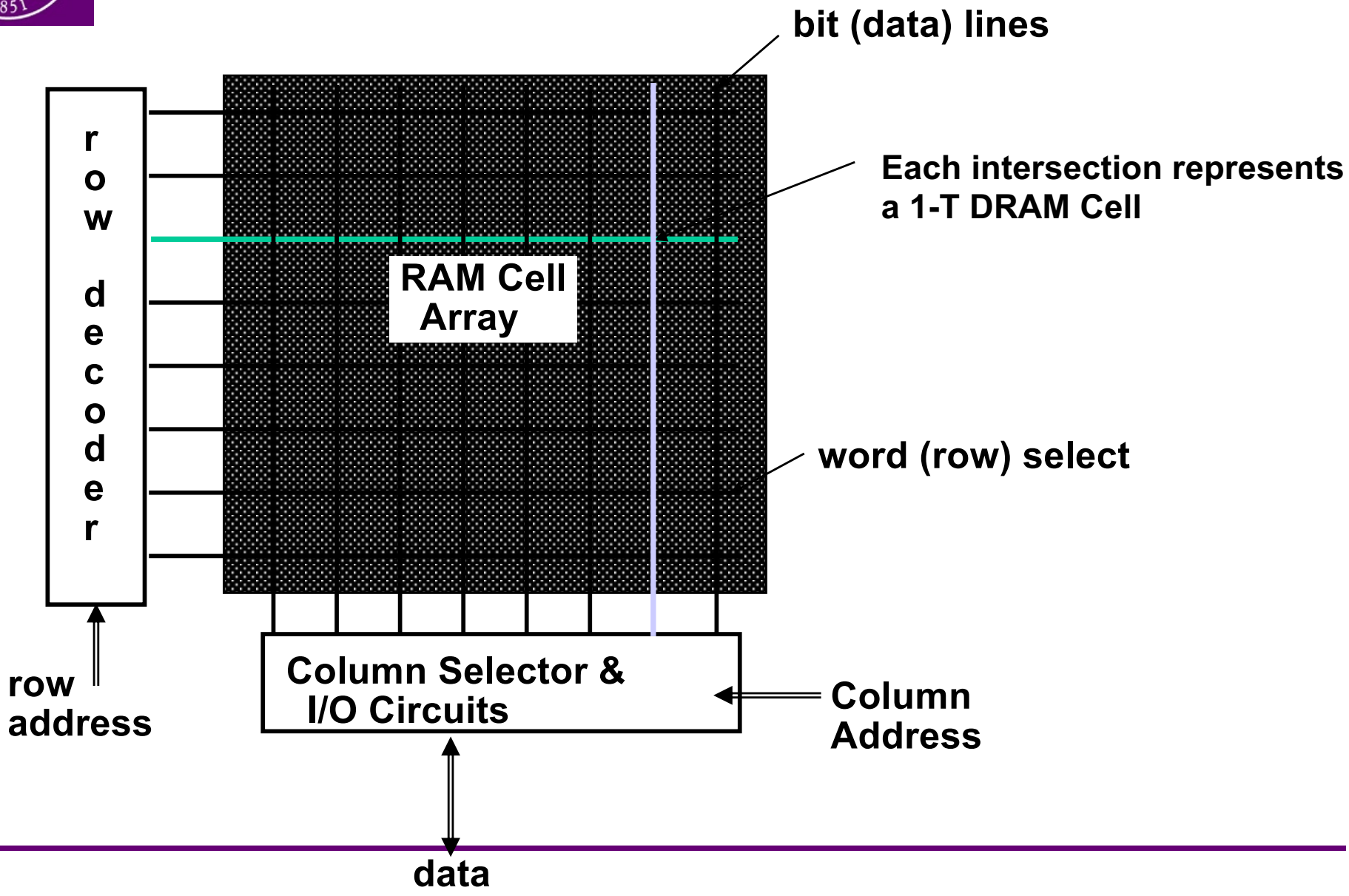
DRAM in a nutshell

- DRAM READ is destructive
  - Charge redistribution destructs the charge collected
  - Every READ must be "restored" with a "re-write"
- Furthermore, DRAM "leaks" even if not READ
- Needs periodic refresh -> overhead, slower
  - Capacitor leaks and discharges over time
  - Worsens with temperature
  - Time that DRAM needs to set aside to refresh itself appears as "time that DRAM is NOT accessible", translates to "slower" access time

Comparison of SRAM and DRAM Cell Size

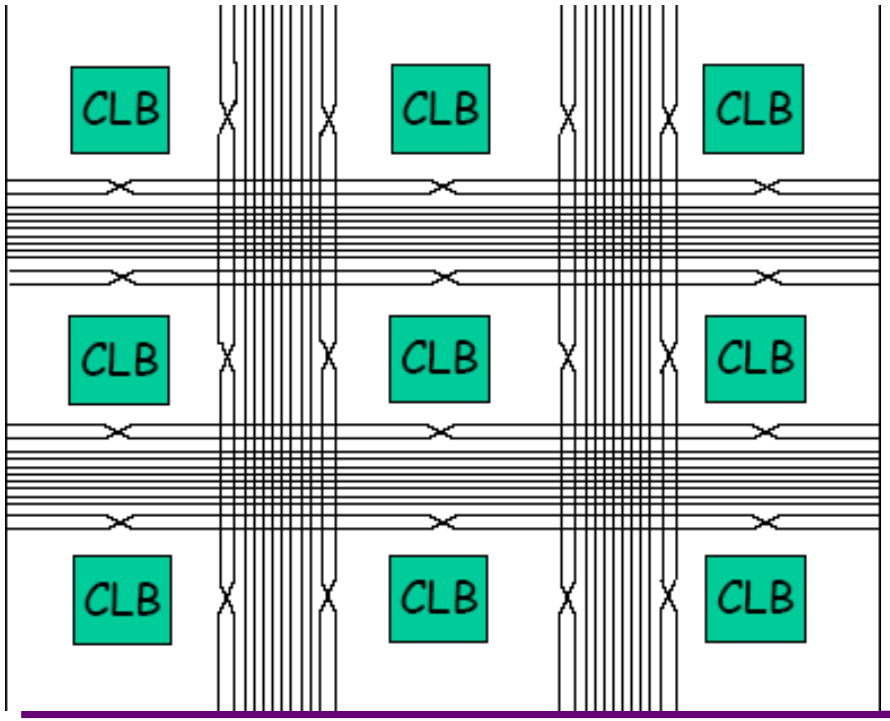


Classical DRAM Organization (square)



FPGA Basics

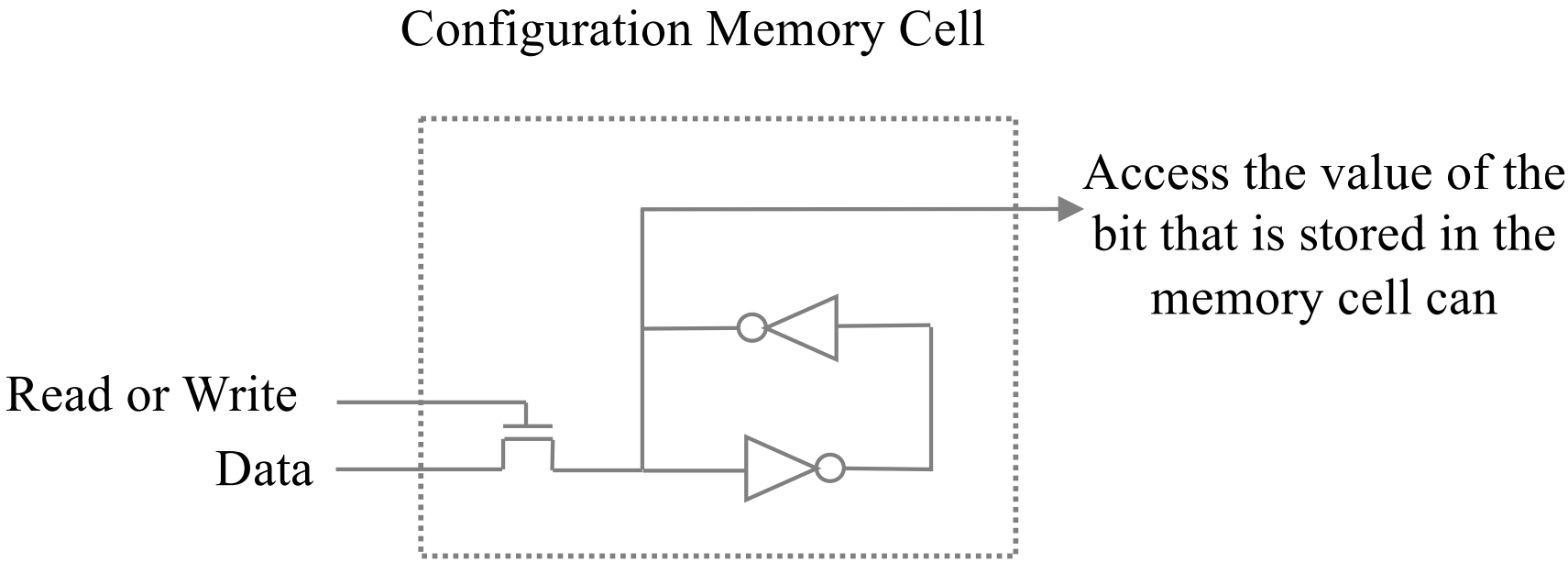
- Think of a cell (or block)
  - That can be programmed to implement any small combinational or sequential circuit
  - Tile the surface of a silicon die with these cells
  - Wires running vertically and horizontally between these cells



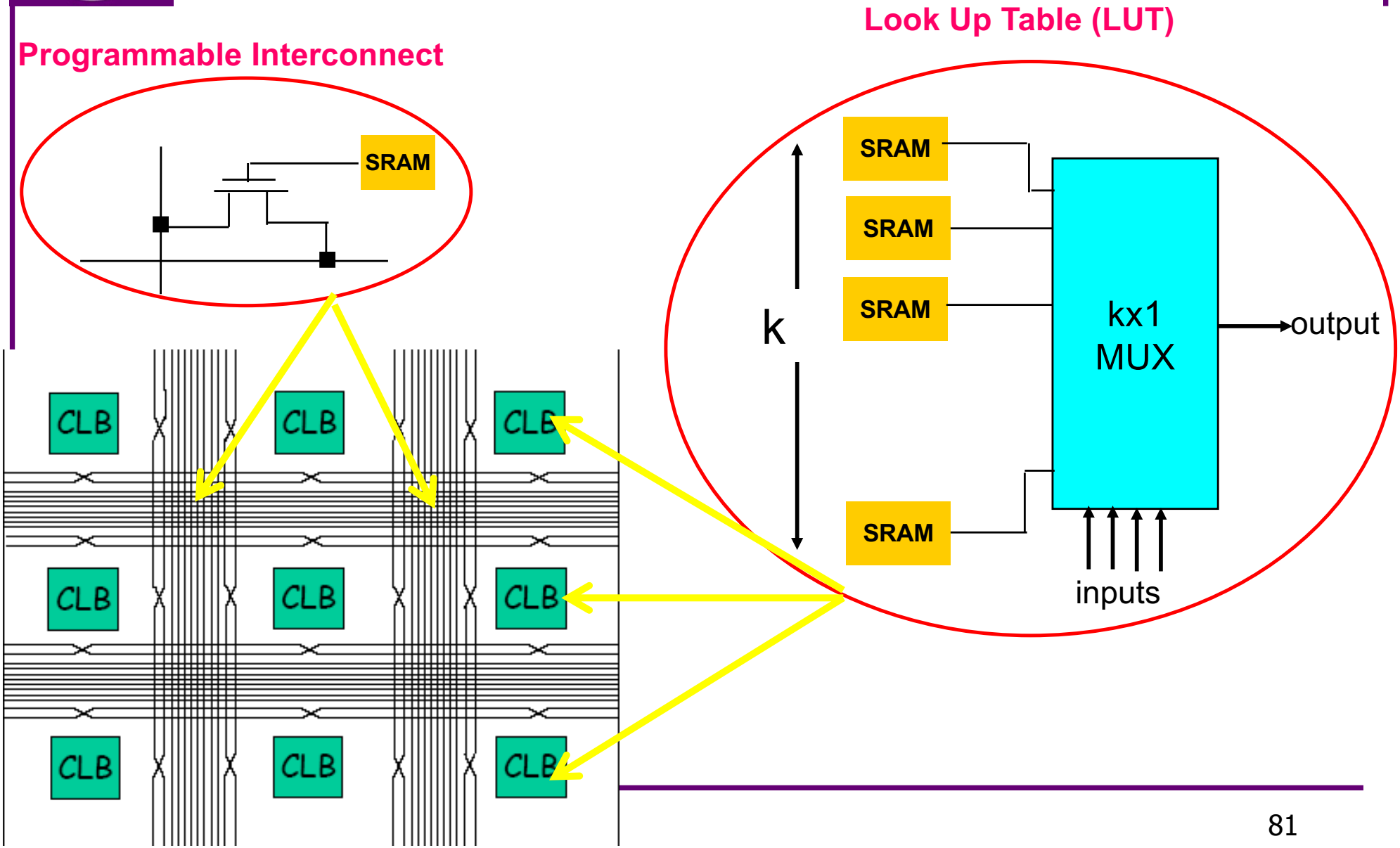
Any block can connect itself to horizontal or vertical wires that are adjacent to it.

Interconnect of the wires are called switching matrix.

SRAM Technology



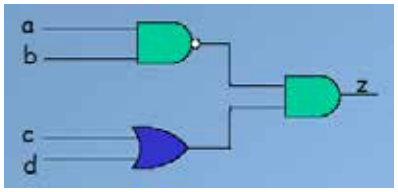
Programmability



Inside the CLB

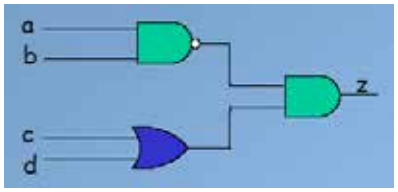
- Many FPGAs look similar
  - But they differ in how Configurable Logic Blocks (CLBs) are implemented
  - CLB is actually Xilinx terminology
  - Main components in a CLB
    - SRAM array can be used as LUT (Look Up Table) to implement combinational logic through truth tables
      - Also called Function Generators
    - Fast carry logic
    - D flip-flops
    - Multiplexers to configure the interconnection within the CLB

Look Up Tables



a	b	c	d	z
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Look Up Tables



a	b	c	d	z
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

16

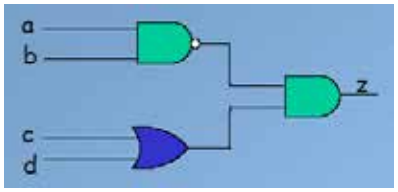
SRAM

SRAM

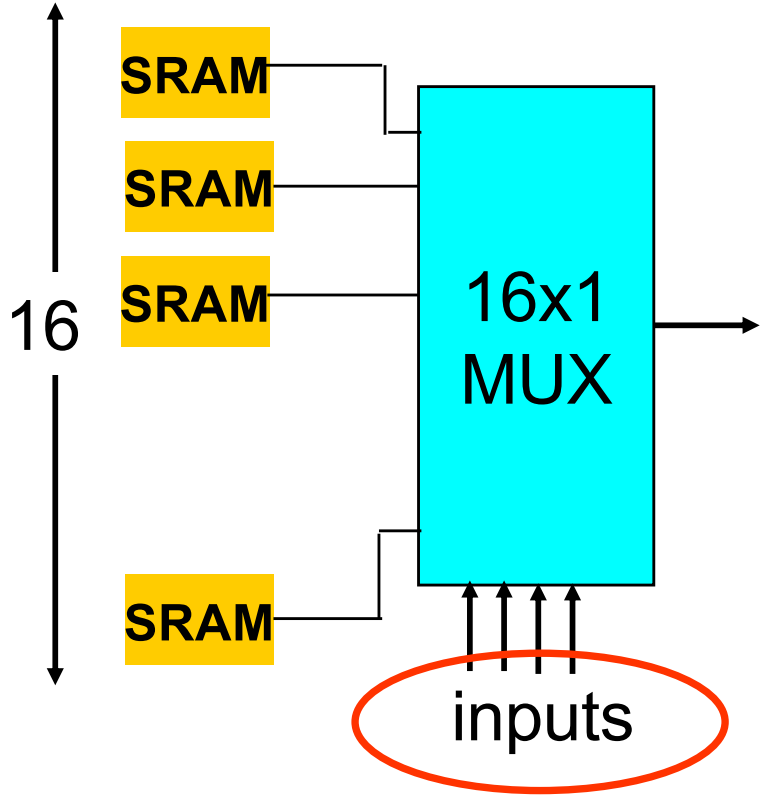
SRAM

SRAM

Look Up Tables

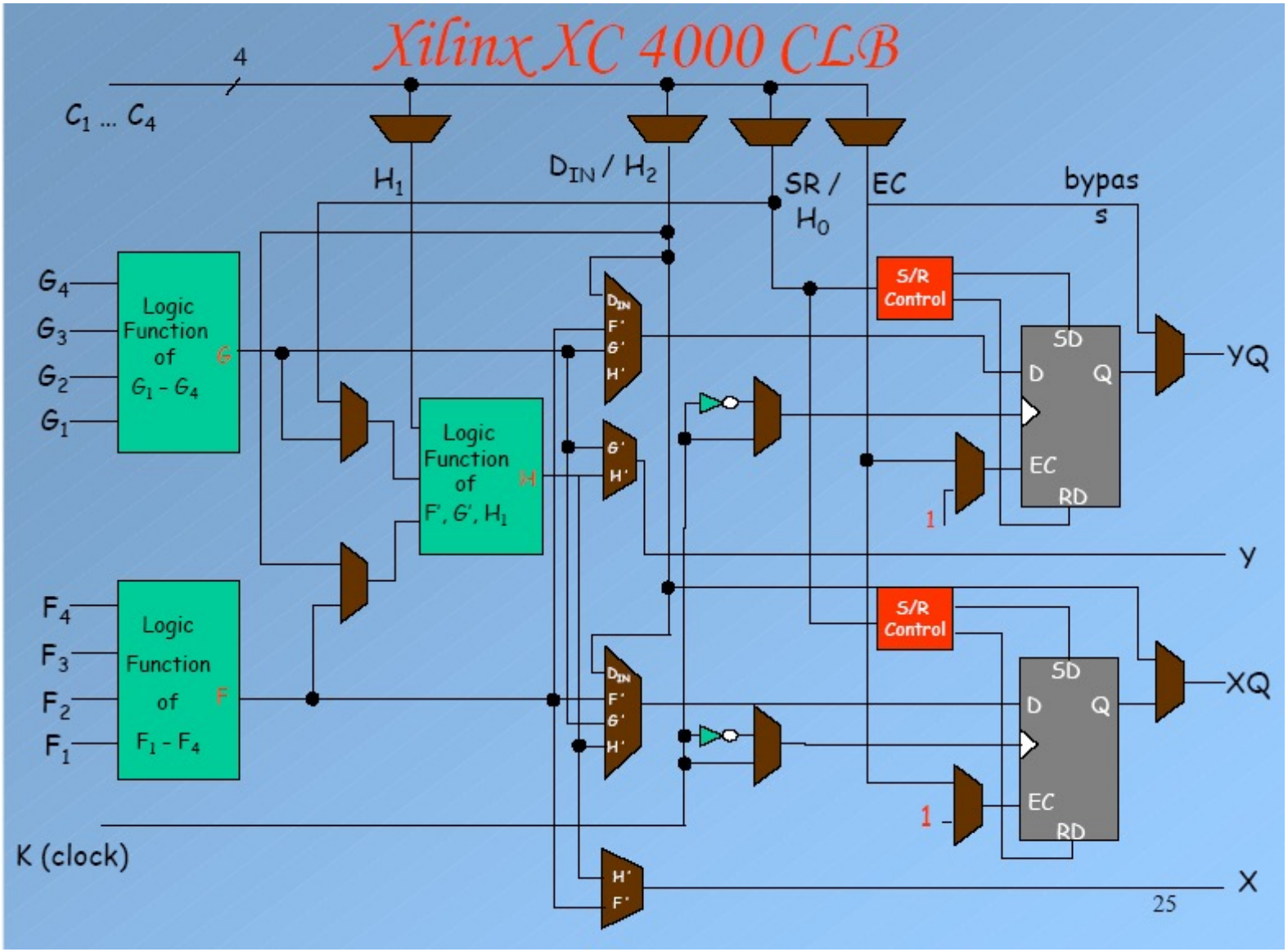


a	b	c	d	z
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



Organization of the FPGA

- An array of configurable (a.k.a. programmable) logic blocks (CLBs)
- A typical basic logic block contains
  - A few LUTs
  - D Flip Flops
  - Multiplexers
  - Special logic to compute the carry signal for a one-bit addition operation
    - In case the LUTs are used to perform addition operations, then they will be programmed to compute the SUM output and the special carry logic will produce the CARRY output


Xilinx XC 4000 CLB
4
C1 ... C4
H1
DIN / H2
SR / H0
EC
bypass
G4
G3
G2
G1
Logic Function of G1 - G4
G
Logic Function of F', G', H1
H
F4
F3
F2
F1
Logic Function of F1 - F4
F
K (clock)
S/R Control
SD
D
Q
EC
RD
1
YQ
y
XQ
X
25

Use of LUTs for Logic

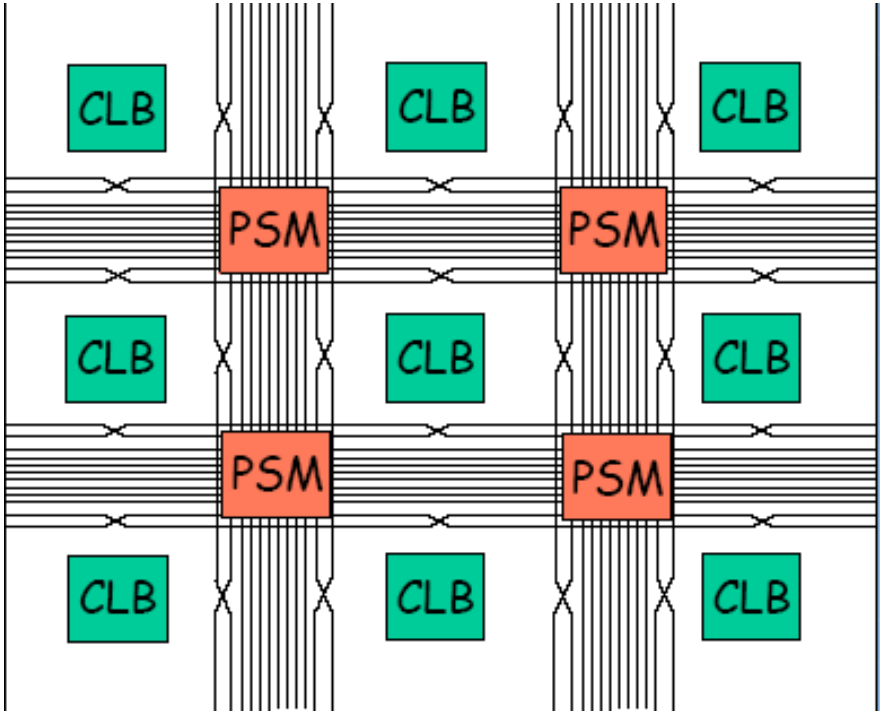
- Using three function generators F, G, and H
  - A single configurable block can implement Boolean function of nine variables
  - It has nine logic inputs: F1, F2, F3, F4, G1, G2, G3, G4, and H1
  - General form of Boolean function of nine variables:
    - F = F(F1, F2, F3, F4)
    - G = G(G1, G2, G3, G4)
    - H = H(F, G, H)

Example

- Implement the following functions on a single configurable block of the XC4000 FPGA:
- $X = A'B'(C + D)$
- $Y = AK + BK + C'D'K + AEJL$
- Use look up table F to implement X
- Use look up table G for AEJL
- Use F, G and H for Y:
  - $Y = K(A+B + C'D') + AEJL = KX' + AEJL = KF' + G$

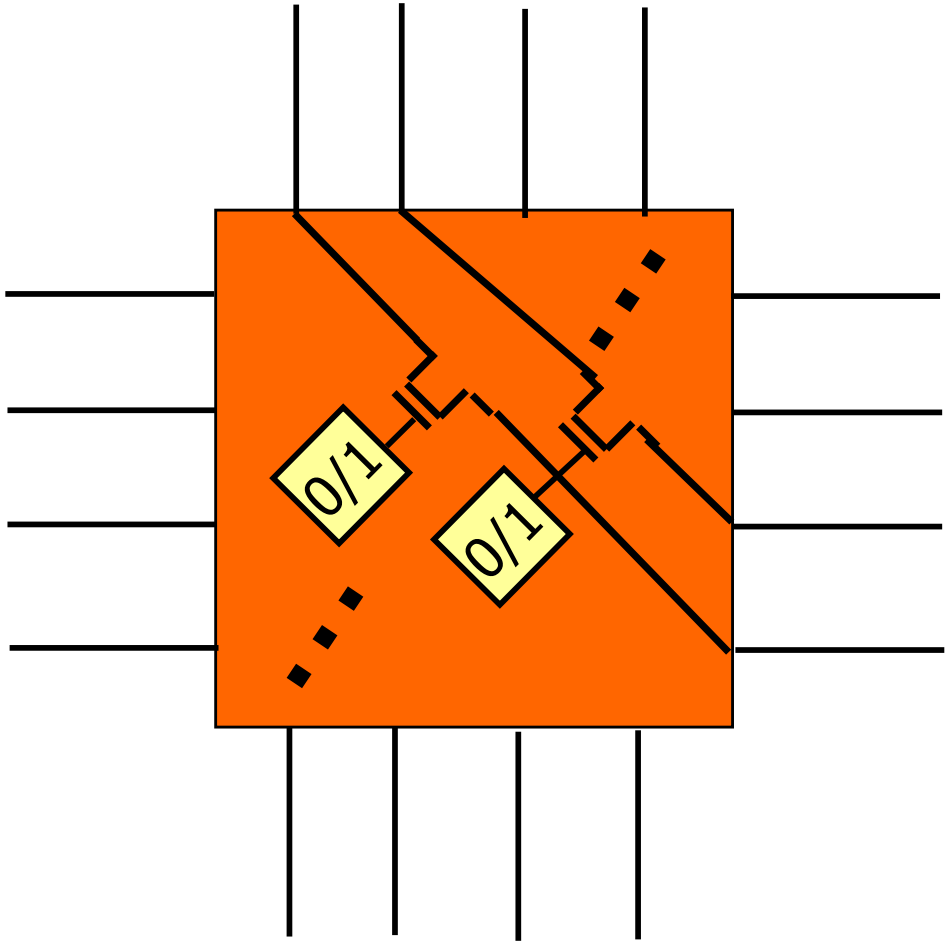
Switch Matrix

- Different levels of interconnections
  - Direct interconnects between CLBs. Short wires to connect adjacent CLBs in both directions

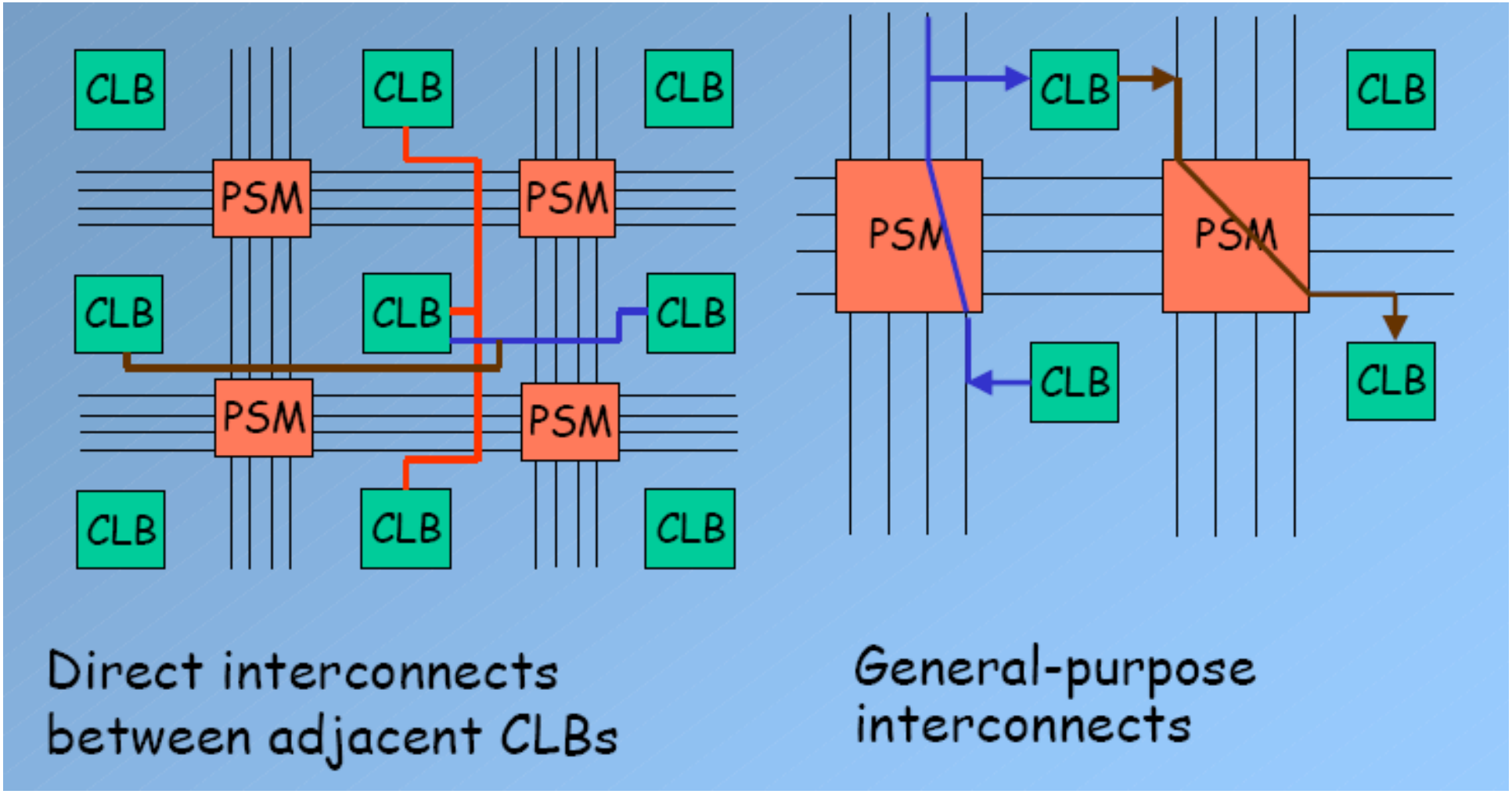


- PSM: **P**rogrammable **S**witch **M**atrix
- PSM can be configured to connect a horizontal wire to a vertical one
- One wire can be connected to multiple wires
- This way output of a CLB can be routed through multiple PSMs to the input of another CLB

Programmable Switch Matrix (PSM)



Examples for Connections



Configuration using SRAM bits

- Configuration data for CLBs
  - Provide the contents of the LUTs
  - F, G, H need (16+16+8 = 40 bits)
  - Configure multiplexors within the CLB
  - To determine the input sources of the flipflops, and the inputs to the H function etc.
  - Bits for configuring flip-flops
- Configuration data for PSM
  - Connecting horizontal and vertical lines
  - Tristate buffers within the CLBs

FPGA vs. ASIC

- ASIC
  - Lengthy design process
  - Expensive for small runs – lithography (making masks for the shapes and patterns to be created on the die and the process of imposing the masks on the dies is extremely expensive)
  - Cannot be changed
- FPGA
  - Can be reprogrammed to change what it implements
    - Creating a design on an FPGA several orders of magnitude quicker than the design process for creating a custom IC
  - Cost/benefit ratio is meaningful for small to medium volume markets
  - The speed of the circuit implemented on the FPGAs can (for most cases) not be as fast as the ASIC implementation of the same circuit, but it may be reasonable/acceptable.

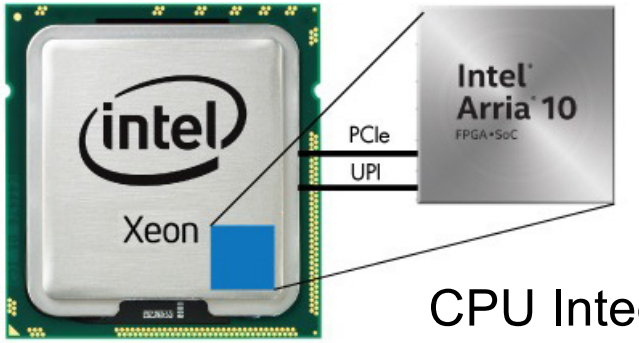
Modern FPGAs

- Can be configured to act like any circuit
- Can do many things, but often deployed for computation acceleration
  - Sub-routines carved out of a software application and offloaded onto FPGA hardware

FPGAs Come In Many Forms

PCIe-Attached

Standalone as a System-on-Chip



CPU Integrated

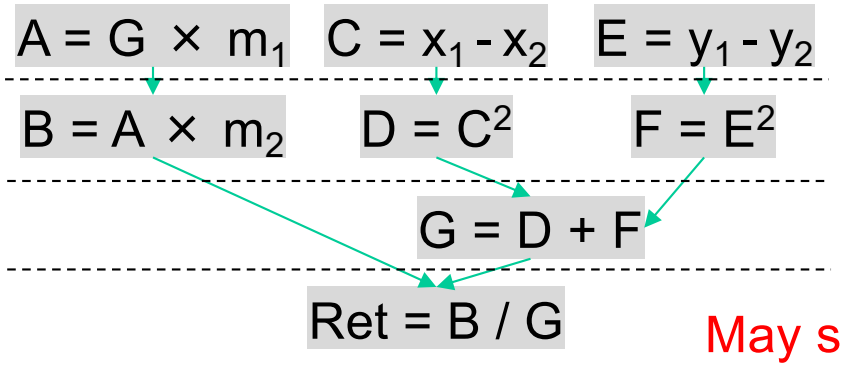
In-Network

How Is It Different From CPU/GPUs

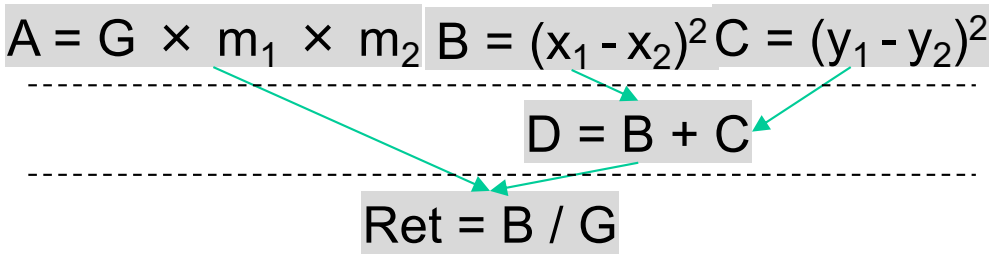
- GPU – The other major accelerator
- CPU/GPU hardware is fixed
  - "General purpose"
  - we write programs (sequence of instructions) for them
- FPGA hardware is not fixed
  - "Special purpose"
  - Hardware can be whatever we want
  - Will our hardware require/support software? Maybe!
- Optimized hardware is very efficient
  - GPU-level performance (on certain occasions)
  - 10x power efficiency (300 W vs 30 W)

Fine-Grained Parallelism of Special-Purpose Circuits

- Example -- Calculating gravitational force: $\frac{G \times m_1 \times m_2}{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
- 8 instructions on a CPU → 8 cycles
- Much fewer cycles on a special purpose circuit



4 cycles with basic operations



3 cycles with compound operations

May slow down clock

Ret = (G × m_1 × m_2) / (($x_1 - x_2$)^2 + ($y_1 - y_2$)^2)

1 cycle with even further compound operations

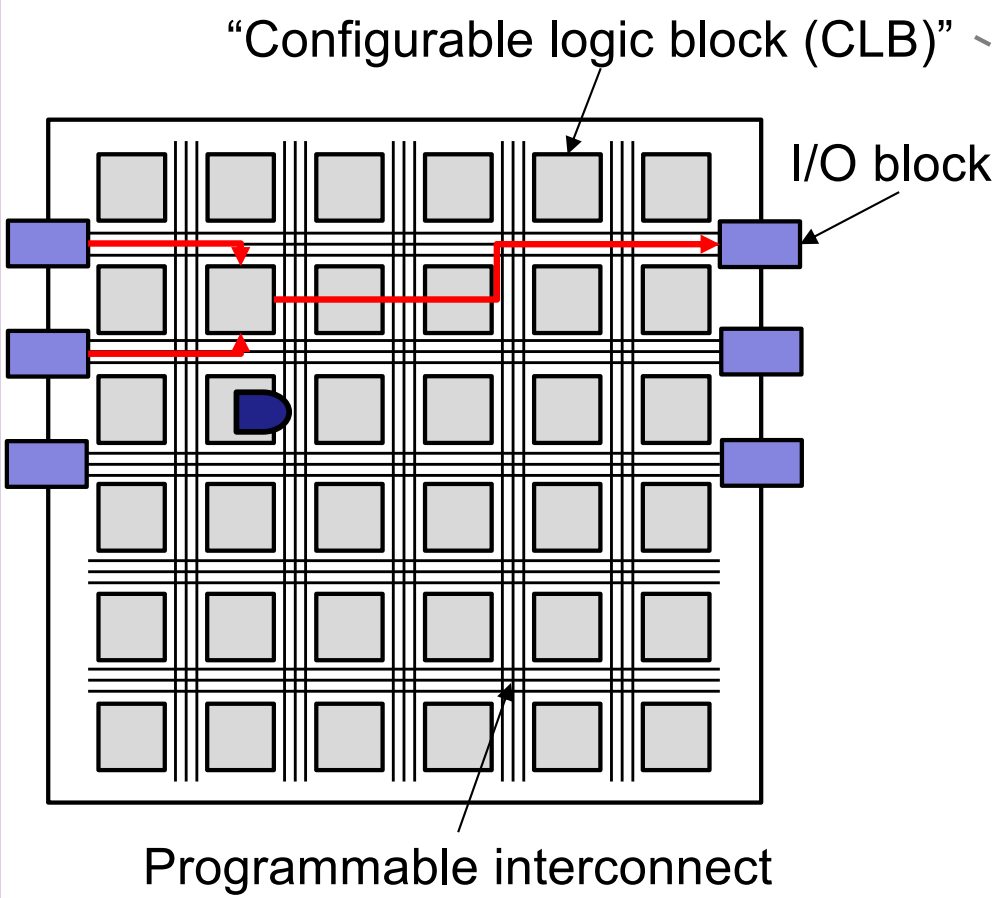
Coarse-Grained Parallelism of Special-Purpose Circuits

- Typical unit of parallelism for general-purpose units are threads ~= cores
- Special-purpose processing units can also be replicated for parallelism
  - Large, complex processing units: Few can fit in chip
  - Small, simple processing units: Many can fit in chip
- Only generates hardware useful for the application
  - Instruction? Decoding? Cache? Coherence?

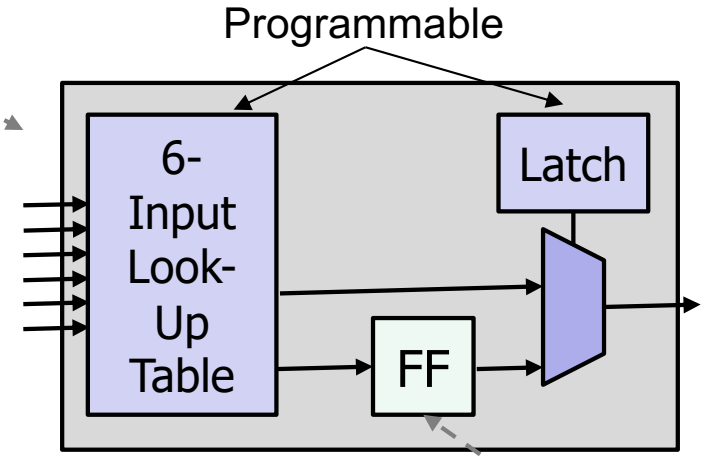
How Is It Different From ASICs

- ASIC (Application-Specific Integrated Circuit)
  - Special chip purpose-built for an application
  - E.g., ASIC bitcoin miner, Intel neural network accelerator
  - Function cannot be changed once expensively built
- \+ FPGAs can be ***field-programmed***
  - Function can be changed completely whenever
  - FPGA fabric ***emulates*** custom circuits
    - Often used as a platform for prototyping ASICs
- \- Emulated circuits are not as efficient as bare-metal
  - ~10x performance (larger circuits, faster clock)
  - ~10x power efficiency

Basic FPGA Architecture



≈

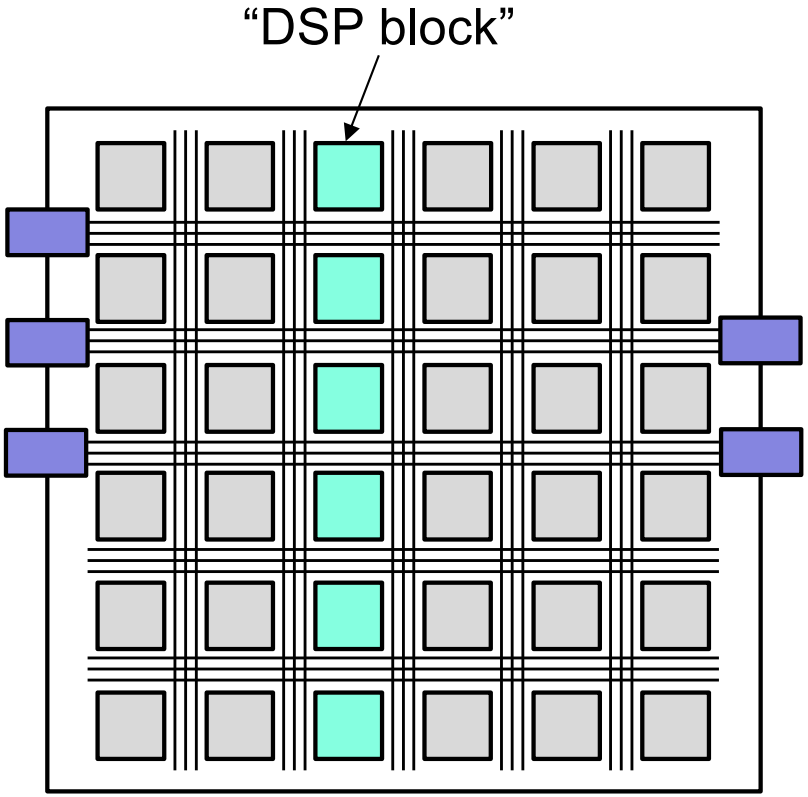


Ex) 2-LUT for "AND"

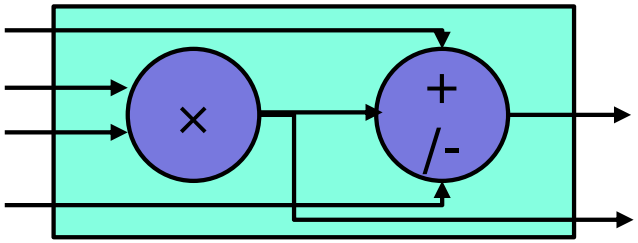
Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

Sequential circuit construction

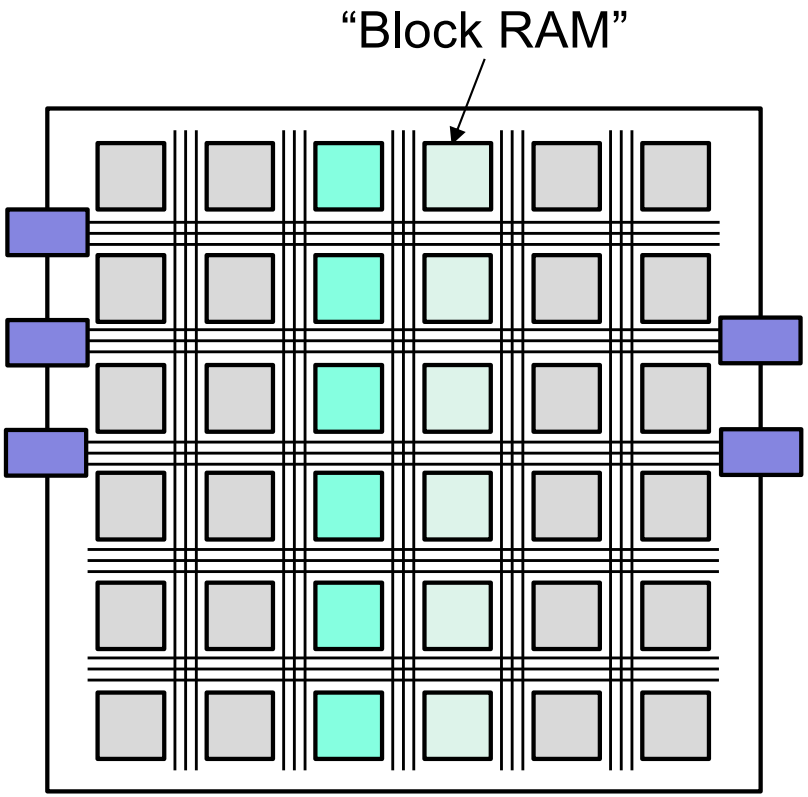
Modern FPGA Architecture – DSP Blocks



- CLBs act as gates – Many needed to implement high-level logic
- Arithmetic operation provided as efficient hardwired ALU blocks
  - "Digital Signal Processing (DSP) blocks"
  - Each block provides an adder + multiplier

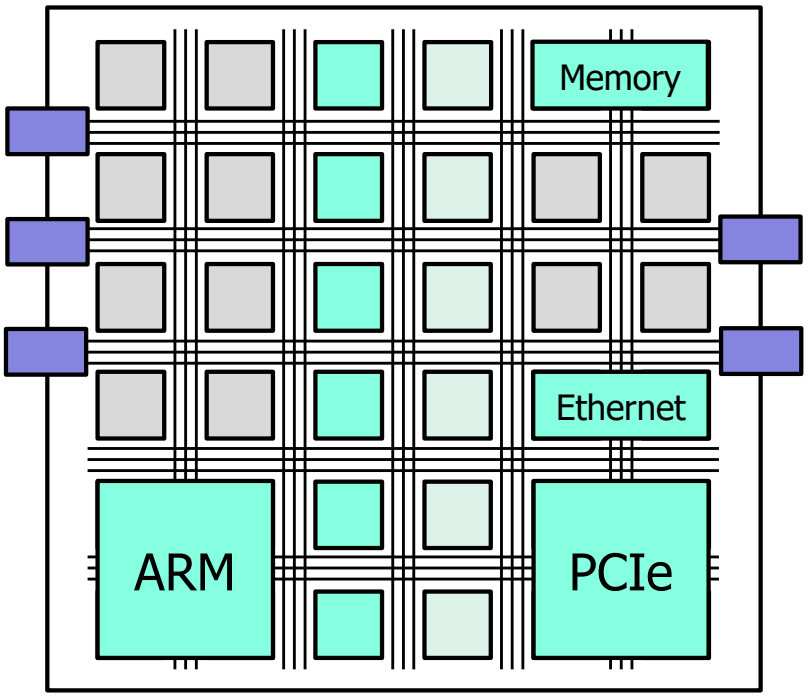


Modern FPGA Architecture – Block RAM



- CLB can act as flip-flops
  - (~1 bit/block) – tiny!
- Some on-chip dedicated pre-fabricated SRAM provided as memory blocks
  - ~18/36 Kbit/block, MBs per chip
  - Massively parallel access to data → multi-TB/s bandwidth

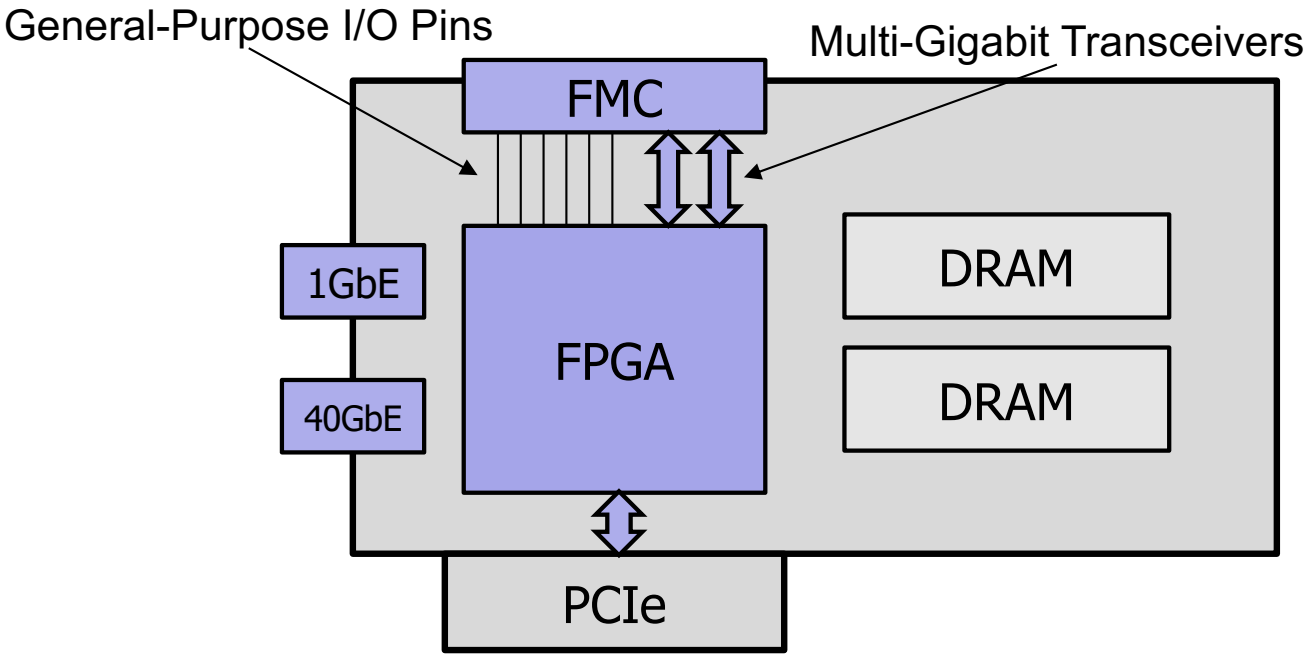
Modern FPGA Architecture – Hard Cores



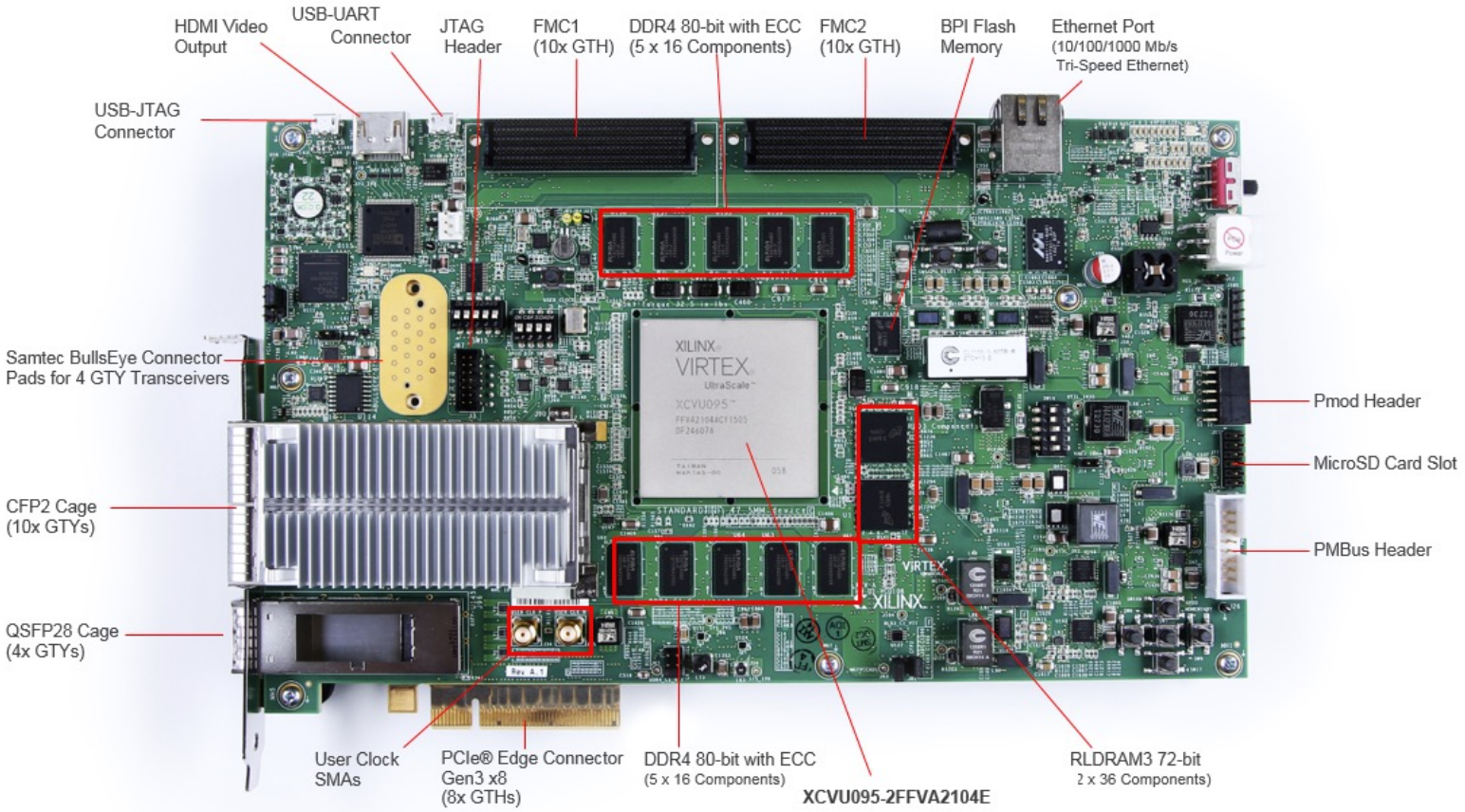
- Some functions are provided as efficient, non-configurable "hard cores"
  - Multi-core ARM cores ("Zynq" series)
  - Multi-Gigabit Transceivers
  - PCIe/Ethernet PHY
  - Memory controllers
  - …

Example Accelerator Card Architecture

- "FPGA Mezzanine Card" Expansion
  - Network Ports, Memory, Storage, PCIe, …



Example Accelerator Card (VCU108)



Programming FPGAs

- Languages and tools overlap with ASIC/VLSI design
- FPGAs for acceleration typically done with either
  - Hardware Description Languages (HDL): Register-Transfer Level (RTL) languages
  - High-Level Synthesis: Compiler translates software programming languages to RTL
- RTL models a circuit using:
  - Registers (state), and
  - Combinational logic (computation)

Major Hardware Description Languages

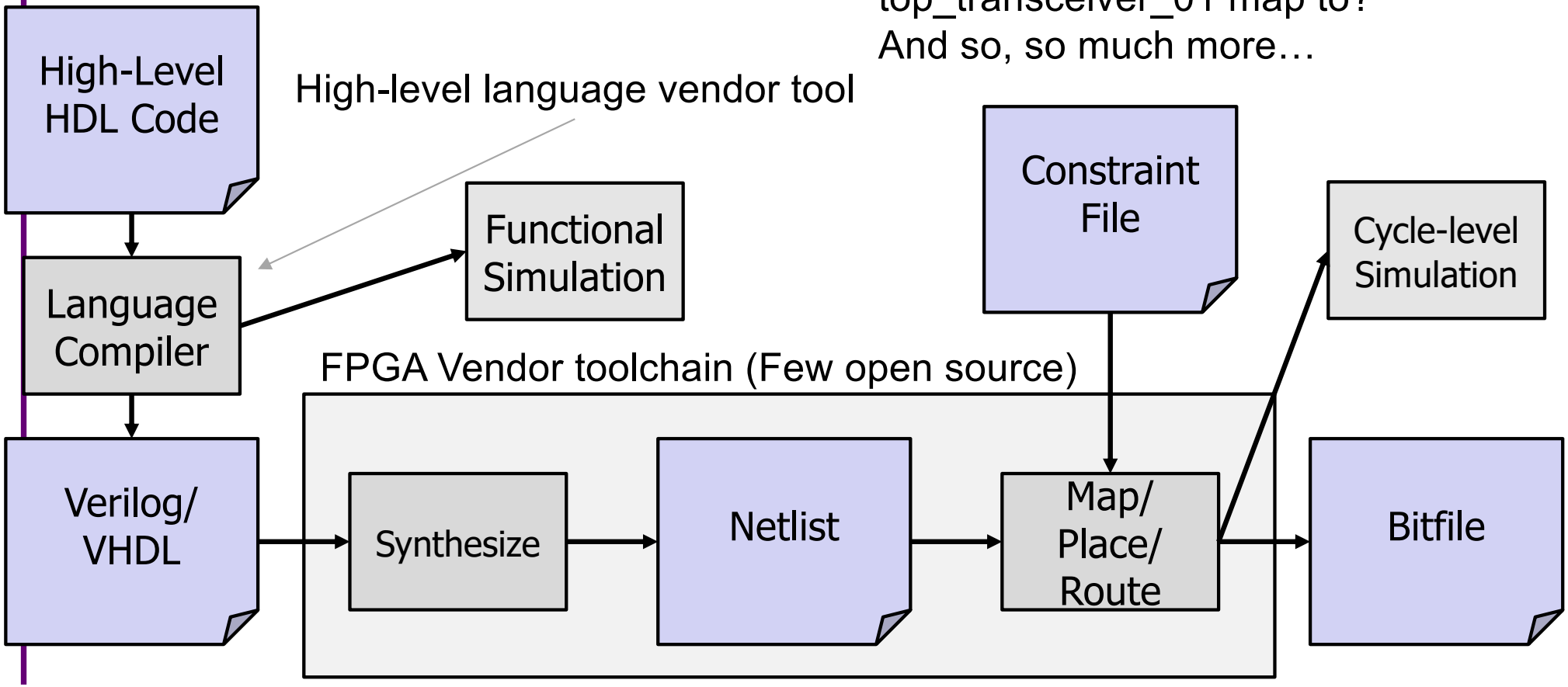
- Verilog: Most widely used in industry
  - Relatively low-level language supported by everyone
- Chisel – Compiles to Verilog
  - Relatively high-level language from Berkeley
  - Embedded in the Scala programming language
  - Prominently used in RISC-V development (Rocket core, etc)
- Bluespec – Compiles to Verilog
  - Relatively high-level language from MIT
  - Supports types, interfaces, etc
  - Also active RISC-V development (Piccolo, etc)

High-Level Synthesis

- Compiler translates software programming languages to RTL
- High-Level Synthesis compiler from Xilinx, Altera/Intel
  - Compiles C/C++ annotated with ***#pragma***'s into RTL Verilog
  - Needs to be INTELLIGENTLY annotated to get performance
- OpenCL
  - Inherently parallel language more efficiently translated to hardware
  - Stable software interface

FPGA Compilation Toolchain

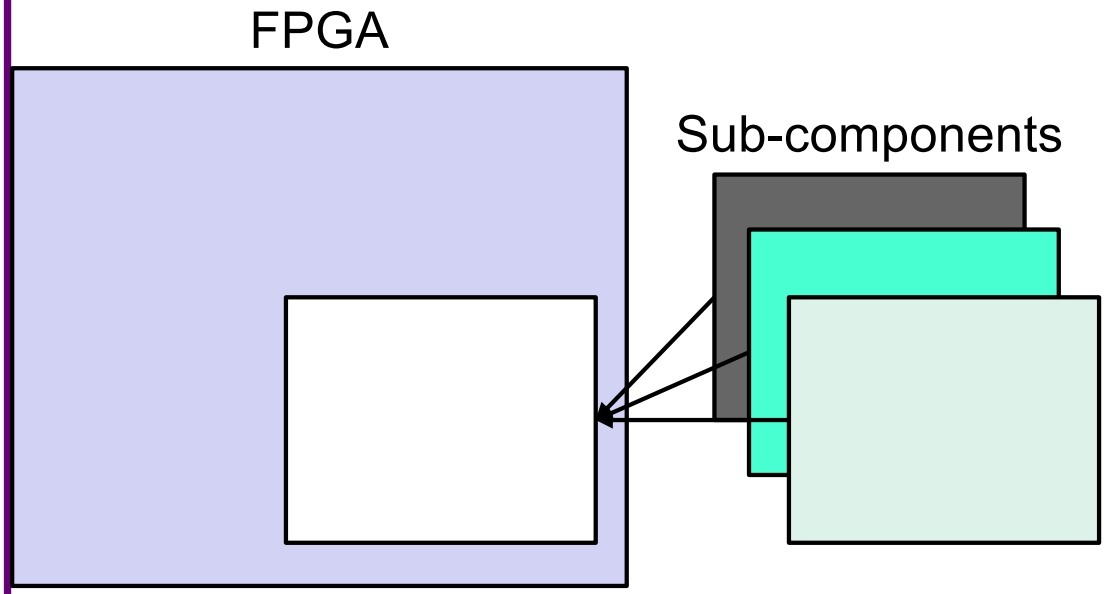
"Which transceiver instance should top_transceiver_01 map to?"
And so, so much more…



Programming/Using an FPGA Accelerator

- Bitfile is loaded into FPGA over "JTAG" interface
  - Typically used over USB cable
  - Supports FPGA programming, limited debugging access, etc.
- PCIe-attached FPGA accelerator card is typically used similar to GPUs
  - Program FPGA, execute software
  - Software copies data to FPGA board or places onto shared main (DRAM) memory space, notifies FPGA
    -> FPGA logic performs computations
    -> Software copies data back from FPGA (or FPGA stores data back into shared space)

Partial Reconfiguration



- Parts of the FPGA can be swapped out dynamically without turning off FPGA
  - Physical area is drawn on chip
- Used in Amazon F1, etc
- Toolchain support for isolation

FPGAs In The Cloud

- Amazon EC2 F1 instance (1 – 4 FPGAs)
- Microsoft Azure, etc…