



AN HONORS UNIVERSITY IN MARYLAND

Department of Computer Science and Electrical Engineering

# CMPE 415

## Programmable Logic Devices

### FPGA Technology

Prof. Ryan Robucci

Some slides (blue) developed by Jim Plusquellic

Some images credited to book: Maxfield, Clive. The design warrior's guide to FPGAs: devices, tools and flows. Elsevier, 2004.

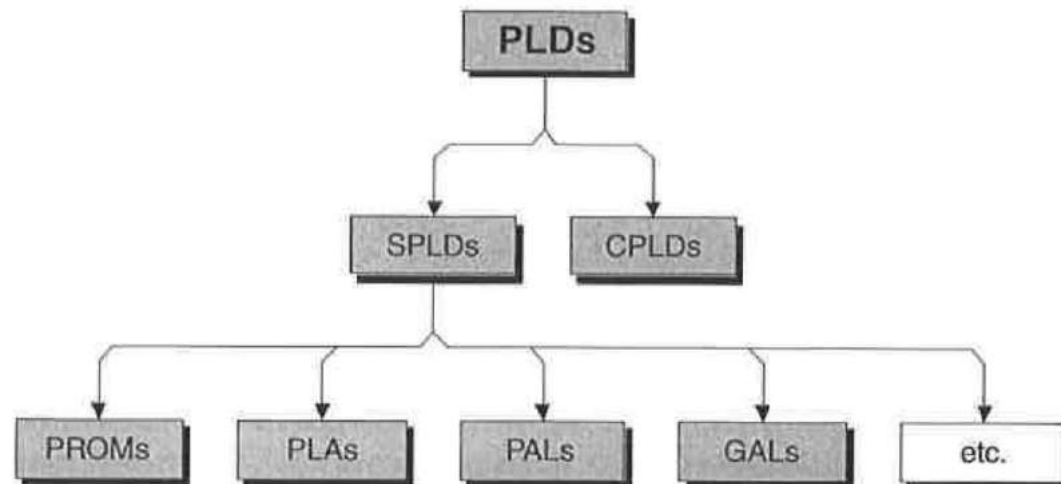
# History

---

- Origins of FPGAs
- Xilinx introduced first FPGA in '84, but engineers didn't embrace them until early '90s.
- History:
  - '47: Shockley, et. al. introduce first transistor at Bell Labs.
  - '50: Bipolar junction transistor (BJT) introduced.
  - '62: Hofstein, et. al. introduce metal-oxide semiconductor field-effect transistor (MOSFET) at RCA.
  - '58: Jack Kilby introduced the integrated circuit.
    - (Jack Kilby, Nobel Prize winner, 2000)
  - '70: Intel introduced 1024-bit DRAM, Fairchild introduced 256-bit SRAM.
  - '71: Intel introduced first microprocessor, 4004.

# Family of Programmable Dev.

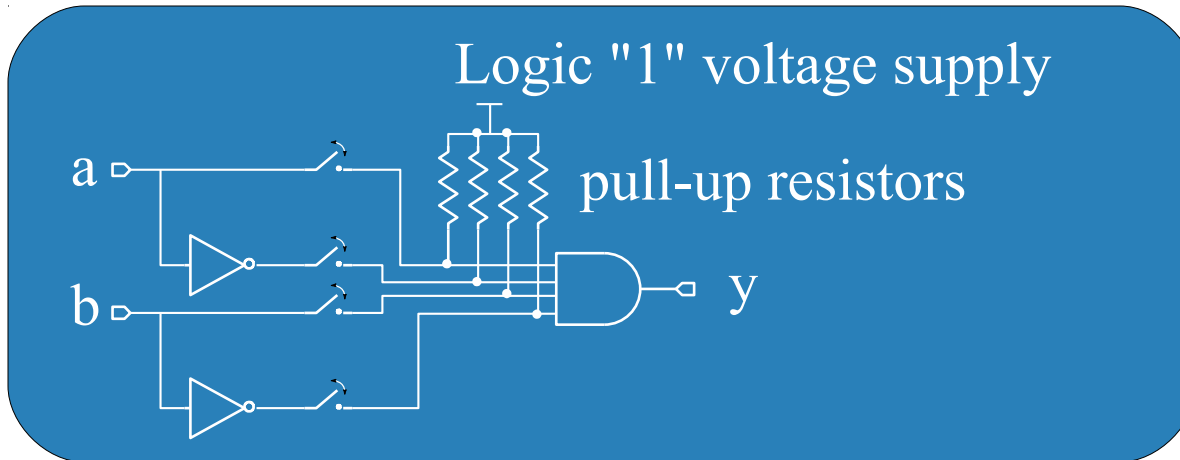
- Programmable Logic Devices PLDs arrived in 70's and but only end of the 70's did more complex variations emerge
- New complex variations were called complex PLDs (CPLDs) while the original line became known as SPLDs



**Figure 3-2. A positive plethora of PLDs.**

# Programmable Logic

- Here, the data on input line may be used or not.
  - If it is not used it is pulled to "1", the AND identity, so it has no effect on the output.
  - If the line is used, the pull-up to "1" is overridden by the driven value.

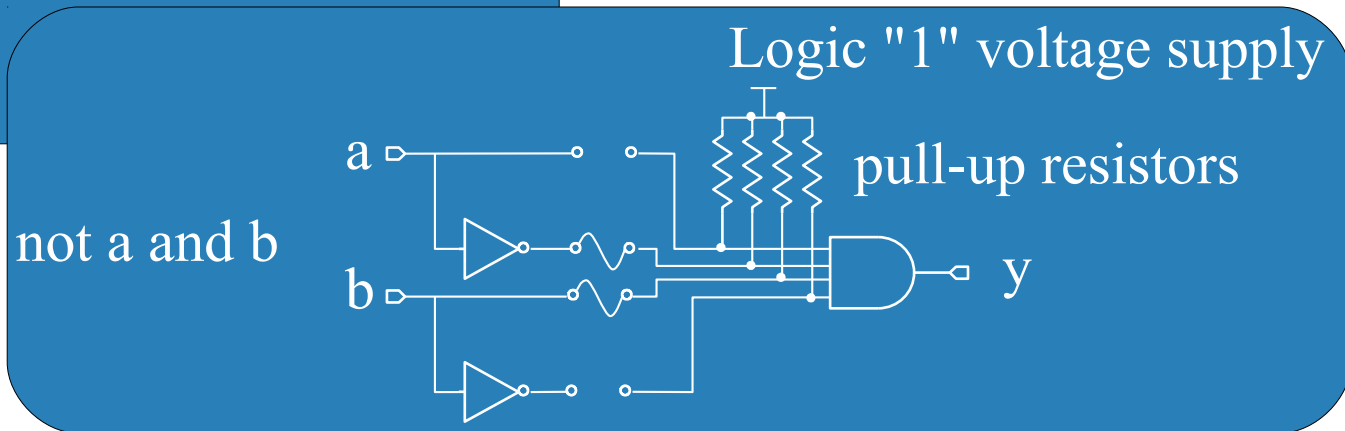
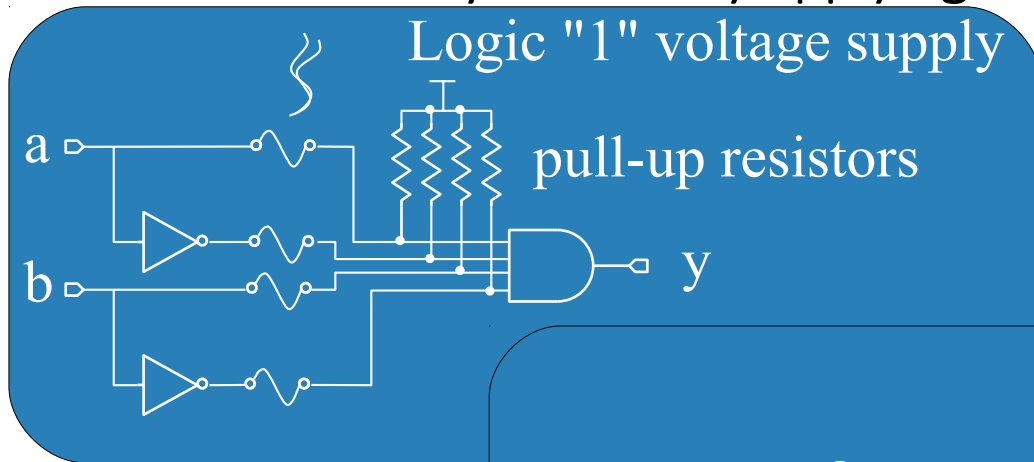


Try to configure switches to implement not a and b.

We'll now discuss technology for implementing the switches....

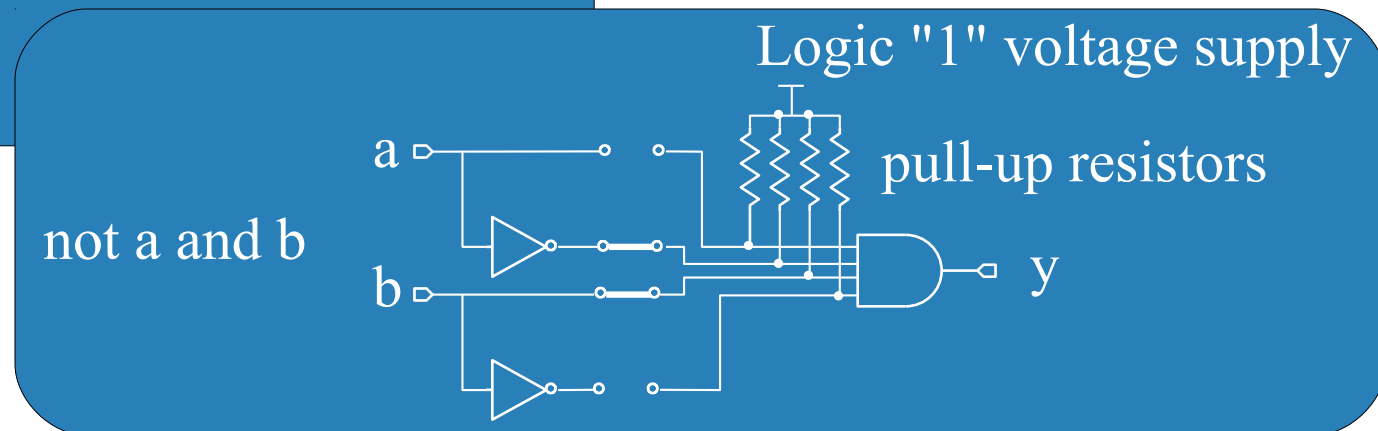
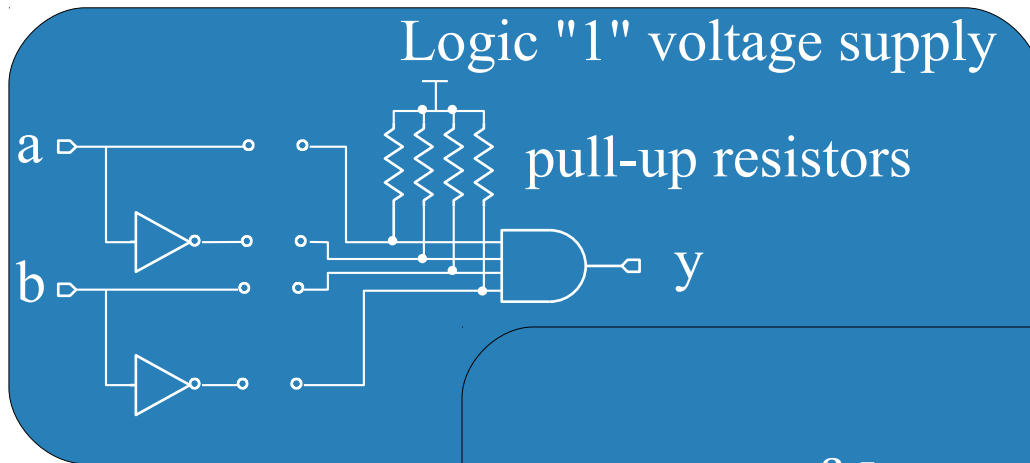
# Fuses

- A fusible-link technology is required to implement links as fuses
- All links are initially active and must be selectively removed
- One-time programmable --- fusible-link-based devices are **one-time programmable**, unless redundant fuses or devices can be switched in to provide **twice-programmable** devices
- Fuses are burned by selectively applying large voltages.



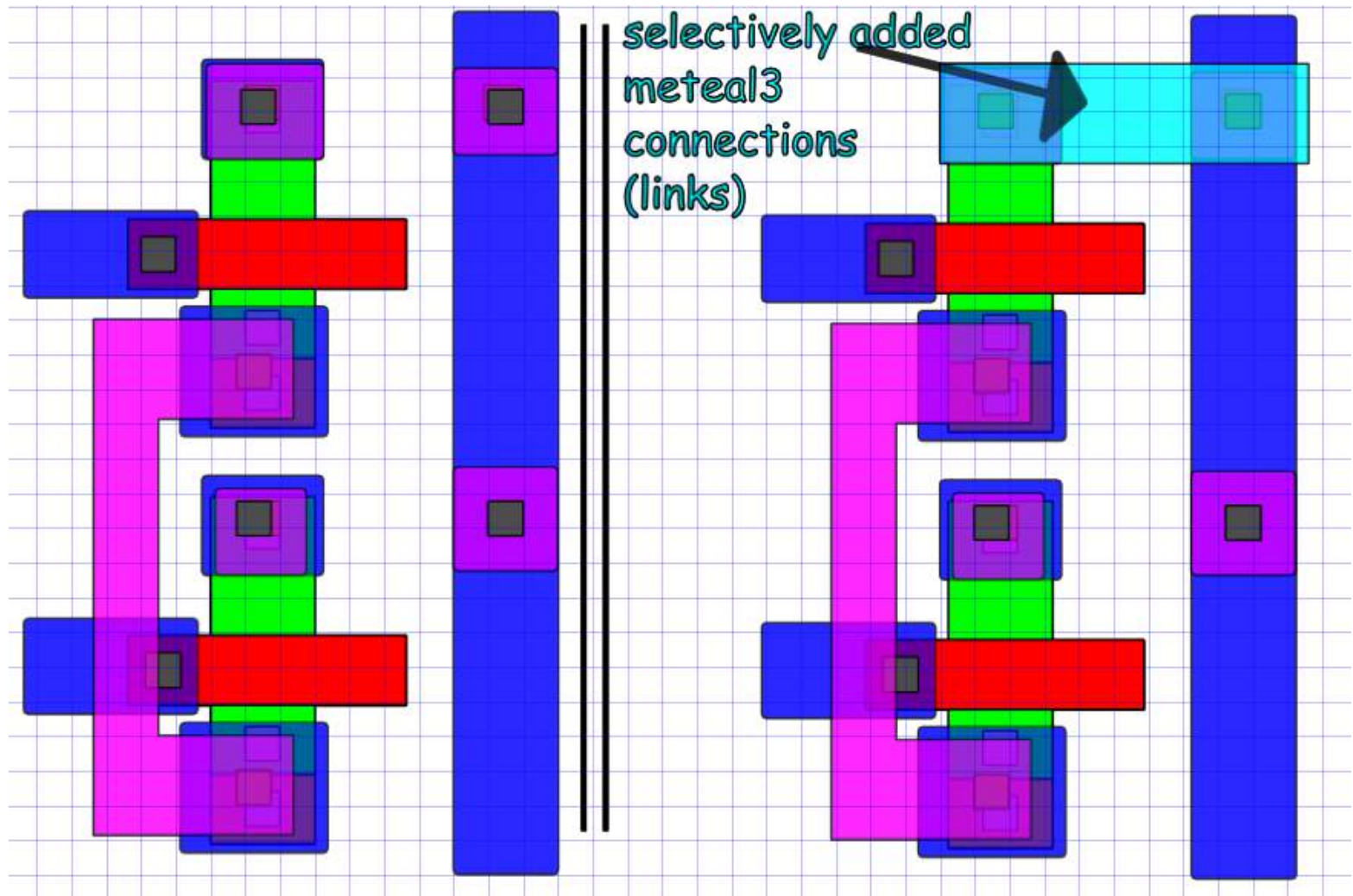
# Anti Fuse Technology

- All links are initially inactive and must be selectively added.
- Links are implemented as insulators that are destroyed or changed such that a conducting path is realized.
- There are **one-time** programmable unless redundant devices are provided.
- No need to program unused sections of IC



# Mask-Programmable Technology

- Most layers are predesigned and prefabricated, requiring a minimal number of custom layers and masks per new design.



# Memory as Logic

- Mask programming was used to create memories known as a ROM (read only memory). This is shown on the next slide.
- It turns out a memory can mimic the behavior of an arbitrary circuit by effectively storing the circuit's truth table and recalling entries using the inputs as the address (index) into the table.

Description of a circuit:

**$x = a \text{ xor } b \text{ xor } c$**

**$y = ((a \text{ and } b) \text{ or } (a \text{ and } c) \text{ or } (b \text{ and } c))$**

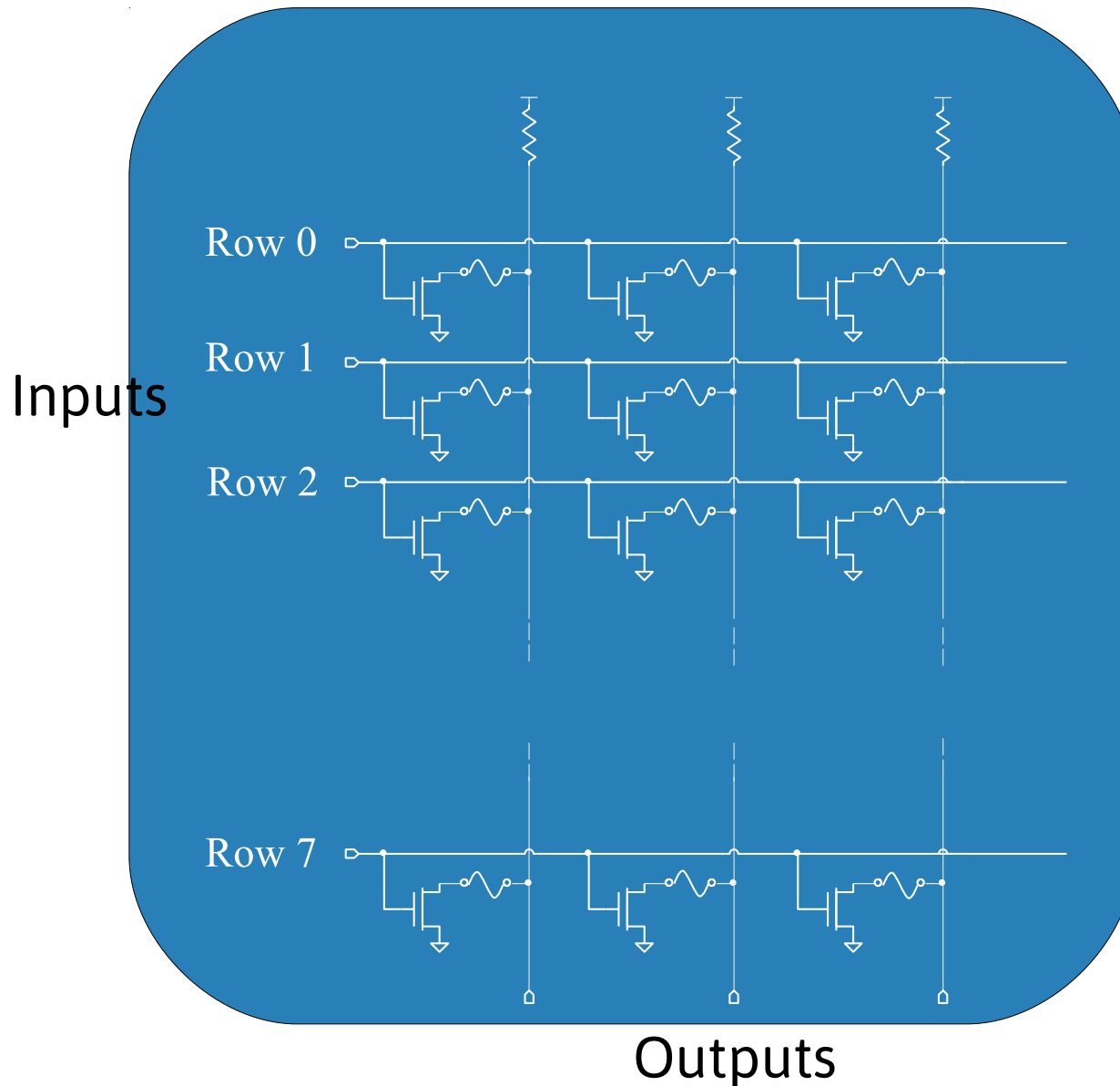
**$z = a \text{ and } b \text{ and } c$**

Truth table:

abc	zyx
000	000
001	001
010	001
011	010
100	101
101	110
110	110
111	111
Address (Input)	Stored Data (output)



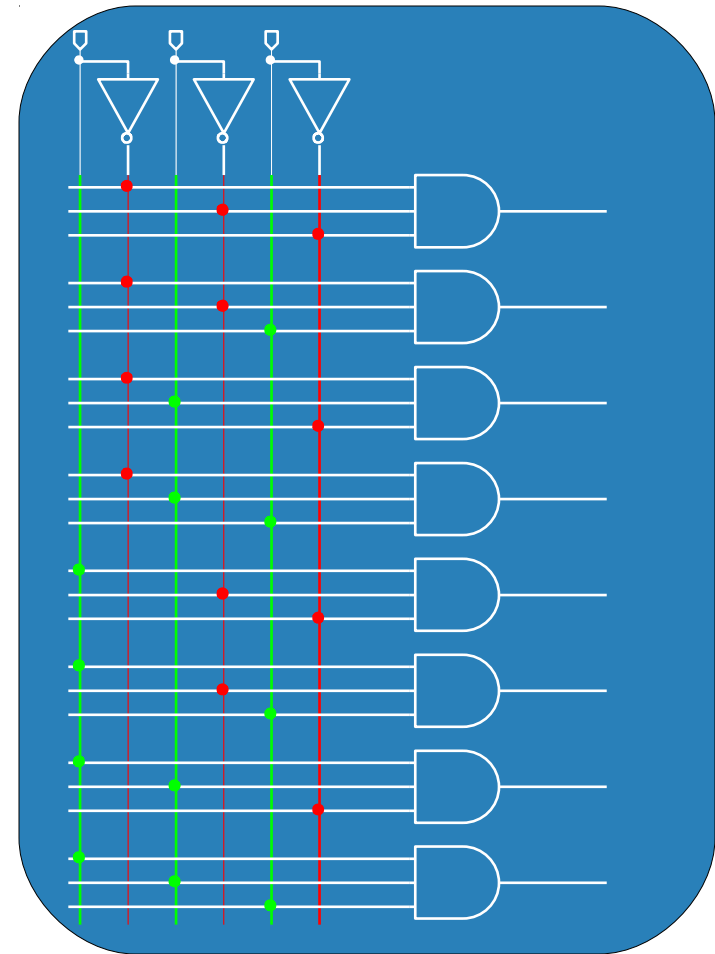
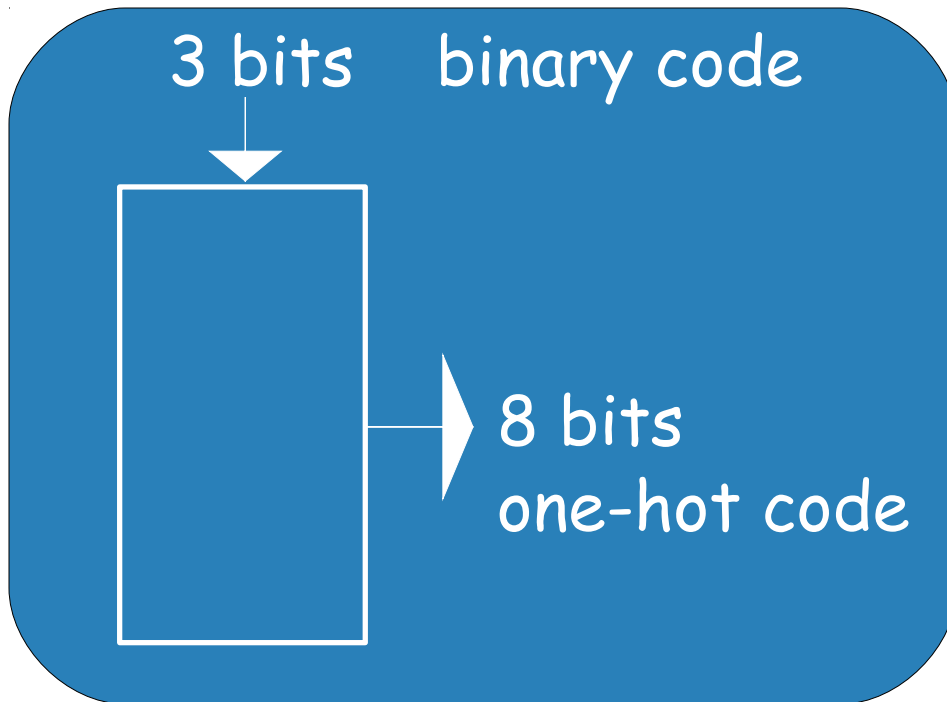
# 2D Data Store



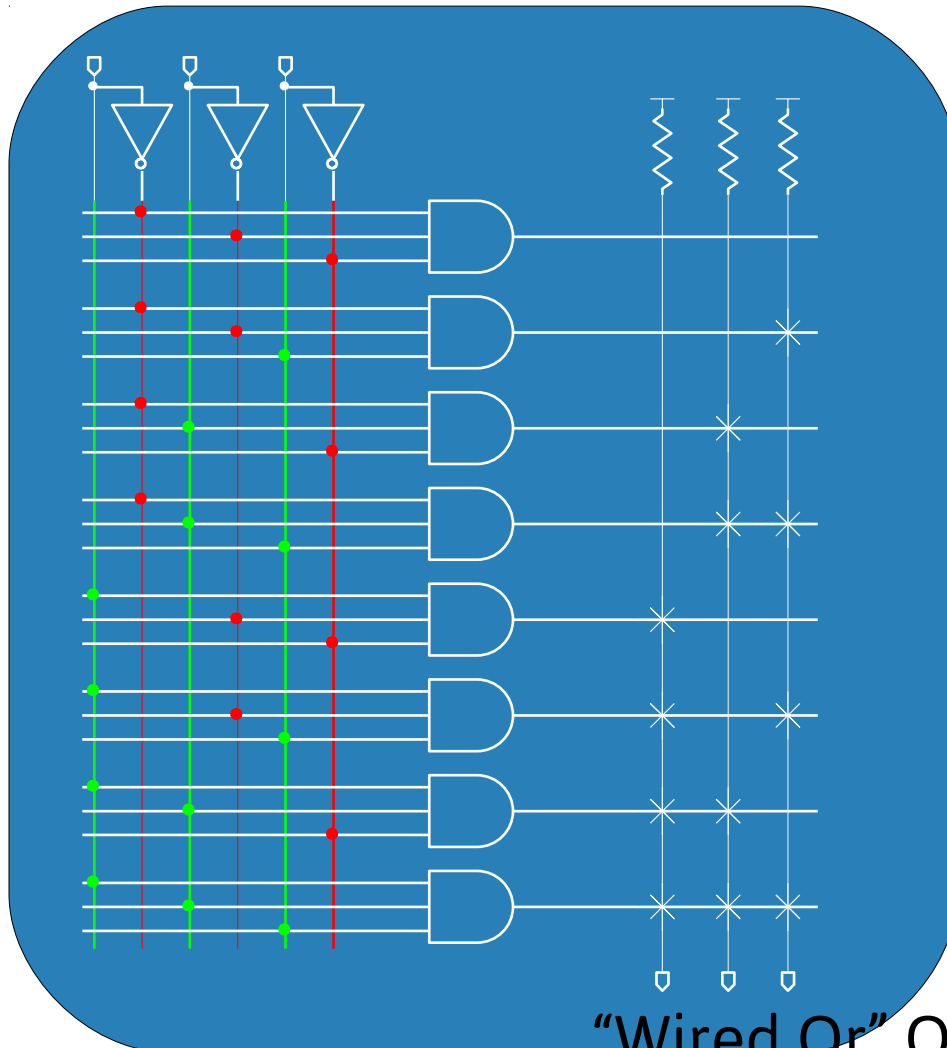
Now for the piece needed to select a row based on the input....

# Decoder

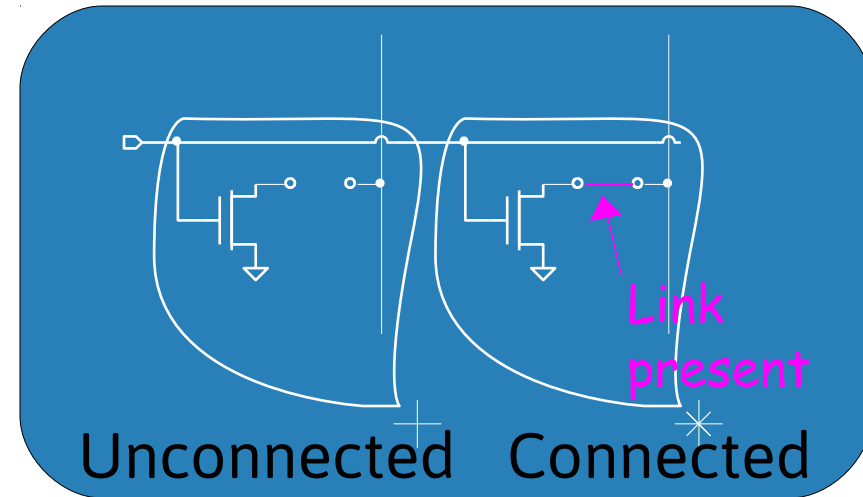
Functional schematic of decoder  
(actual circuitry may differ, e.g.  
wired and may be used):



# Metal Mask LUT

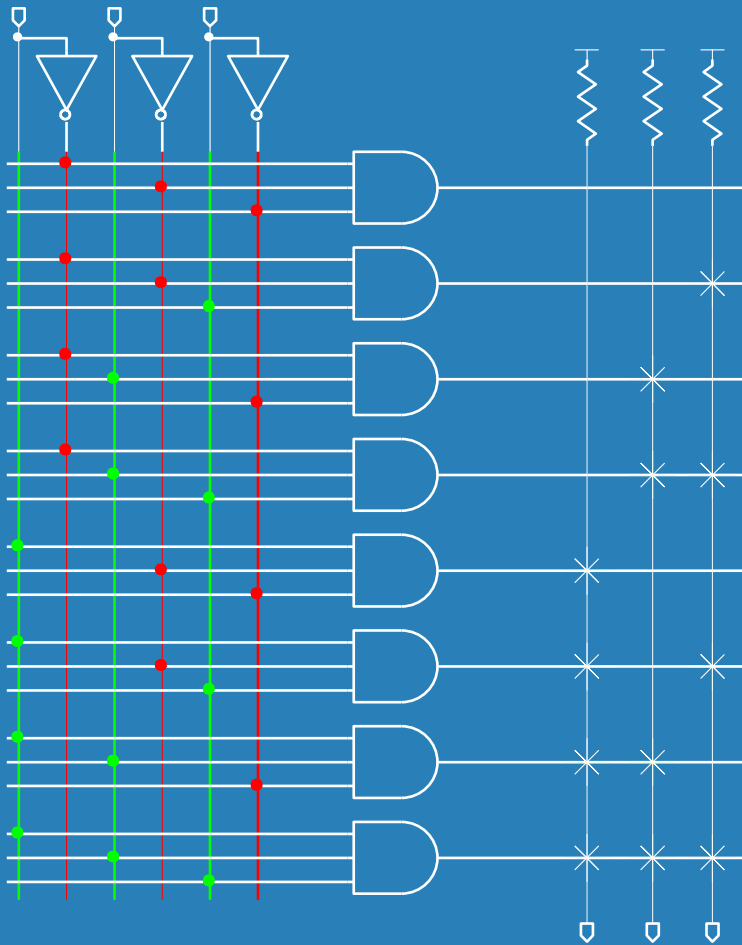


“Wired Or” Outputs

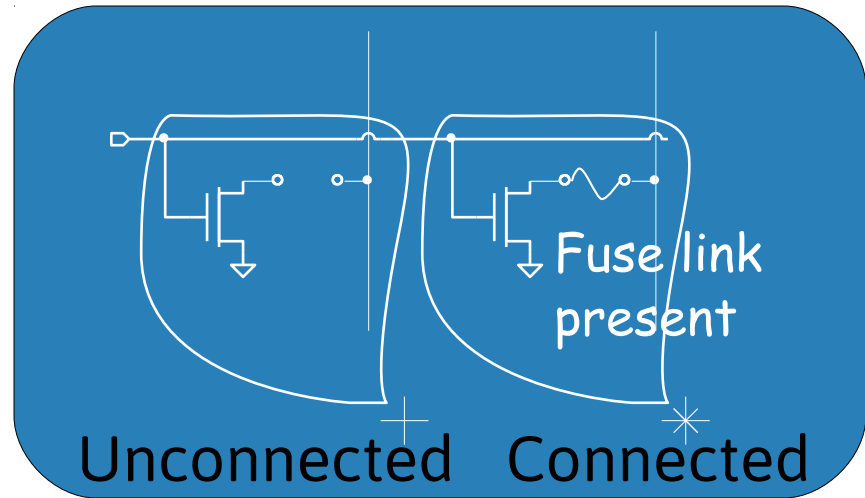


- Transistors are same, but metal layer is added
- Can en mass prefab wafers except top metal and added it later

# Fuse/Anitfuse Link



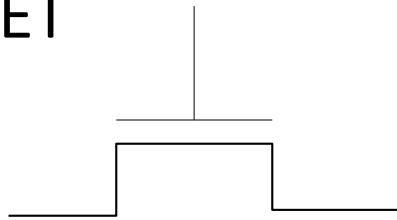
"Wired Or" Outputs



- No need to re-manufacture mask and incur fab costs and time with every change. To change design, throw IC out and "burn" another.
- Even better case is being able to "erase" programming on IC and reuse it...

# Floating Gate Transistor (FLASH)

NFET

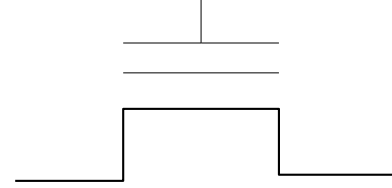


Input

Gate



Floating Gate NFET



Input

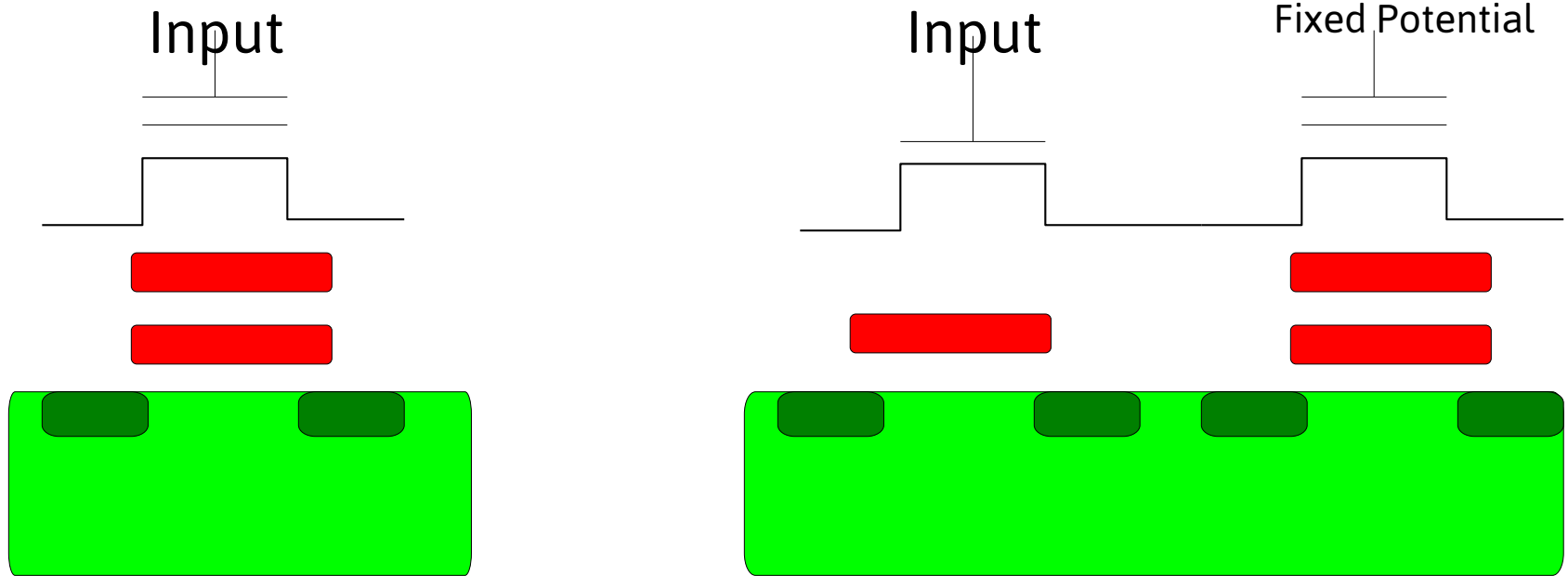
Input Gate

Floating Gate

- A positive potential at the gate input turns transistor on by using capacitive coupling to collect charge to form a channel

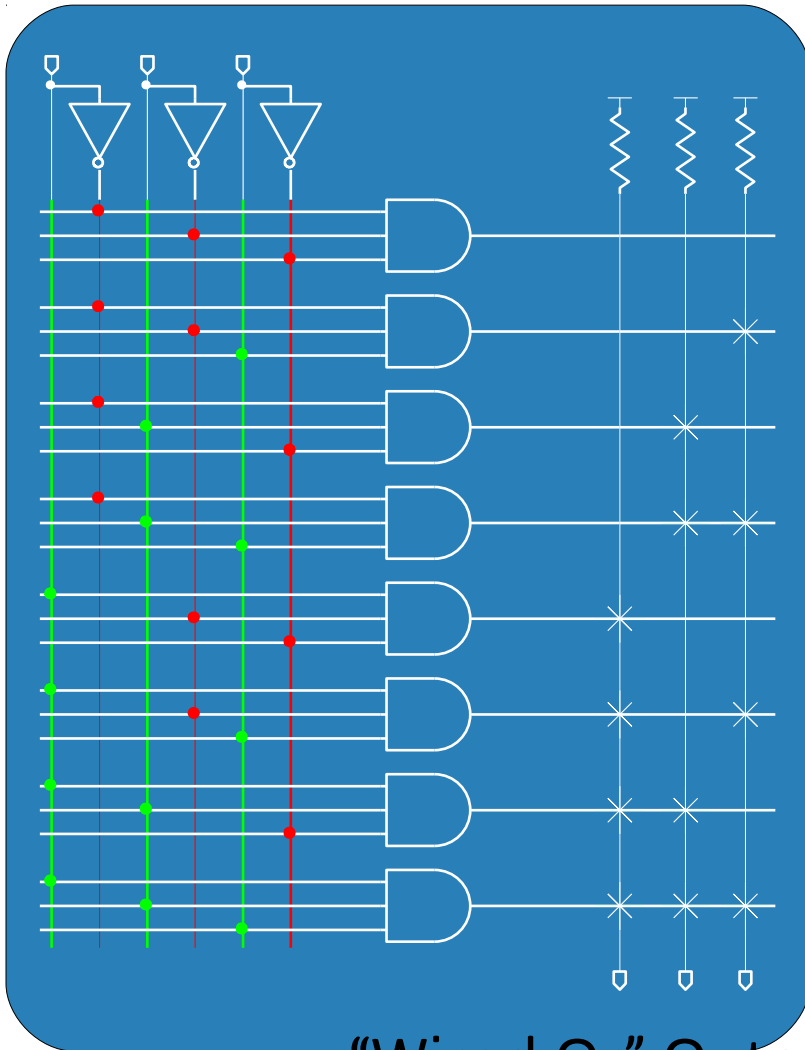
- Here, the input influences channel through a series capacitance, but a stored potential on the floating gate has an effect too.
- Total effect is that of input through series cap + effect of stored potential.
- Negative charge storage prevents channel formation.
- Positive charge storage assists channel formation.
- PFET works opposite

- Depending on design, and stored charge a floating-gate fet can be set to switch on and off with the input or it can be set to be always on or always off.

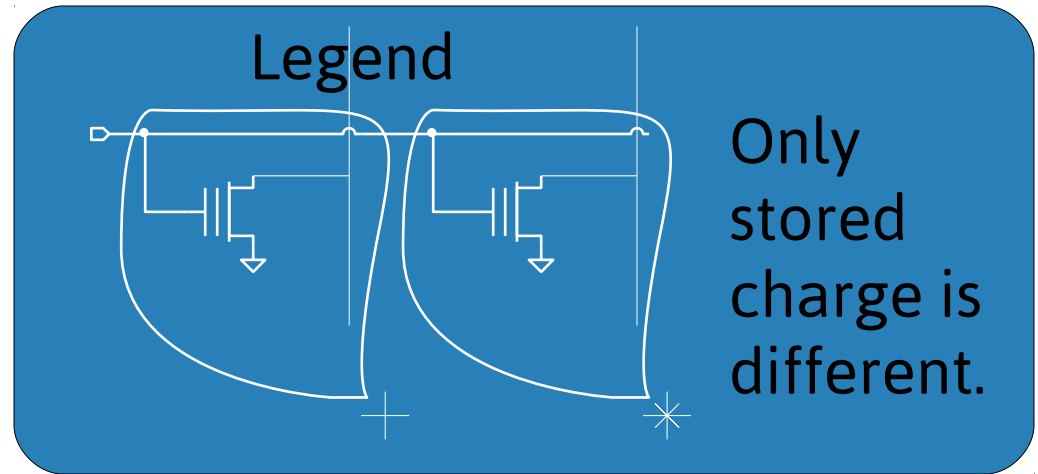


Here are two ways to design a programmable switch cell. The second one uses two FETs...including the space between them it is about 2.5 times larger.

# Floating Gate LUT



"Wired Or" Outputs



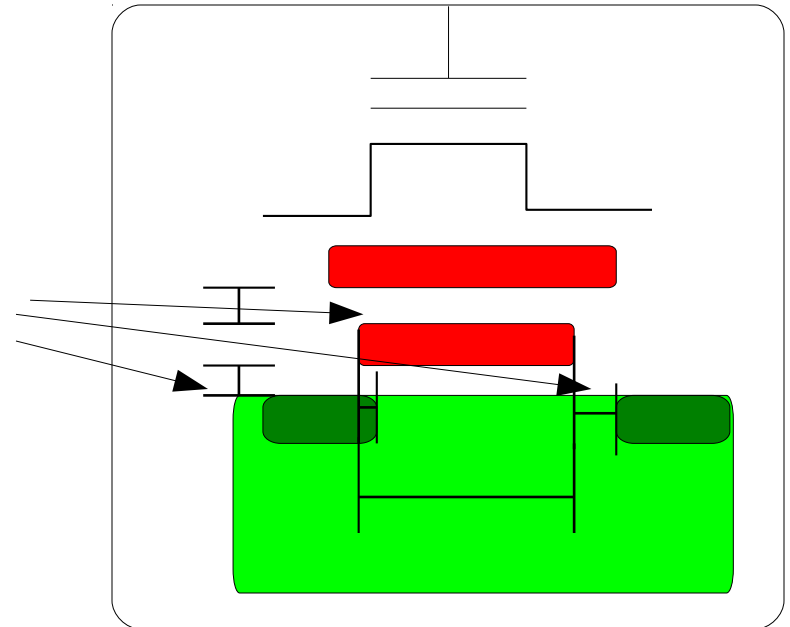
- Nothing in manufacturing sets function.
- Charge stored on "floating" node determines transistors function.
- Charge can be changed by electric fields and UV light....next slide.
- No need to re-manufacture mask and use fab every time, or even get a new IC. To change design, just "erase" IC and reprogram it.
- Opportunity for in-system programming (ISP) and in-the-field updates (field programmable).

# EPROM and EEPROM

- EPROM - Erasable Programmable ROM. This refers to the fact that with normal operating voltages it functions as a ROM, but UV light can be used to erase it (around 20 minutes). Cells typically implemented using single FET. order of magnitude smaller than fusible links->better density
- EEPROM (E2PROM)- Electronically Erasable programmable ROM. This refers to the fact that with normal operating voltages it functions as a ROM, but special high voltages can be used to erase it. Cells typically implemented using two-FET design, thus it is typically larger than EPROM.

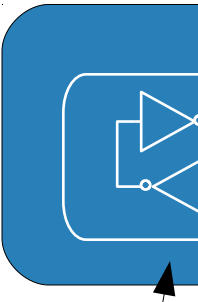
Fundamental structure and operation are same.

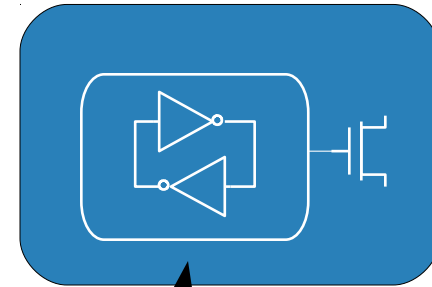
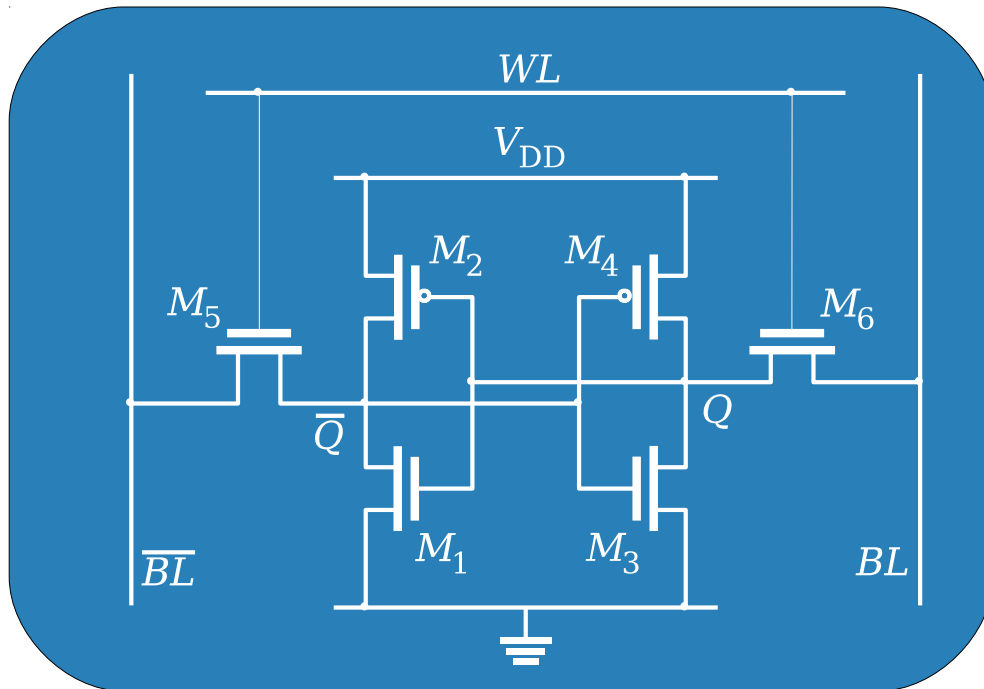
Difference in details of material, dimensions, and spacings to allow for UV or electronic erasing and proper capacitive couplings.





# SRAM

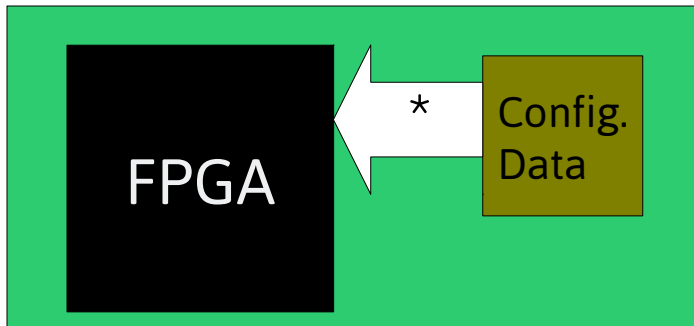
- Typically larger than EEPROM cell
  - Electronically Erasable
  - VERY fast reprogramming
  - Volatile
- 
- The diagram shows a 1T1R1C memory cell. It consists of a blue rounded square containing a white rounded square. Inside the white square is a circuit diagram with a triangle (representing a transistor) and a circle (representing a resistor) connected in a specific configuration. A black arrow points to the bottom of the blue square.



1 or 0  
Stored in each  
latch

# SRAM

- Most devices use SRAM
- Fast reprogramming
- SRAM is an extremely common building block in IC design, this means the structure will be well tested and sure to be reliable in most any technology
- Can be implemented in standard CMOS fabrication technology without need for extra layers or processes for special materials that significantly increase cost
- Volatile...must be reprogrammed at powerup
  - System needs may demand a non-volatile configuration memory on-board (not fuse and flash technologies store the configuration as non-volatile on-chip)



\*A security concern is that a design is transferred as a data stream at boot..it should be encrypted if IP theft is a concern

- Programming is so fast, it can support dynamic reconfiguration where hardware is reconfigured on-the-fly during operation to accelerate functions on-demand

# Antifuse

---

- One-time programmable (unless redundant fuses are provided to provide twice-programmable)
- Radiation hard – not as susceptible to radiation induced “bit flips” that alter configuration in SRAM
  - In particular SRAM is most susceptible as data is being loaded, during programming

# Flash

---

- About 2.5 times larger cells than EPROM, but still smaller than SRAM
  - Note area impacts logic delay and power
- Dedicated flash processes require ~5 additional process steps
- Flash technology integrated with logic is not quite as rapidly updated as SRAM on newer, smaller technology nodes
- Vulnerable to long term effects from Radiation
- “Hybrid flash/SRAM” can use local flash to store configuration and SRAM to implement switches

# Programmable Element Technology

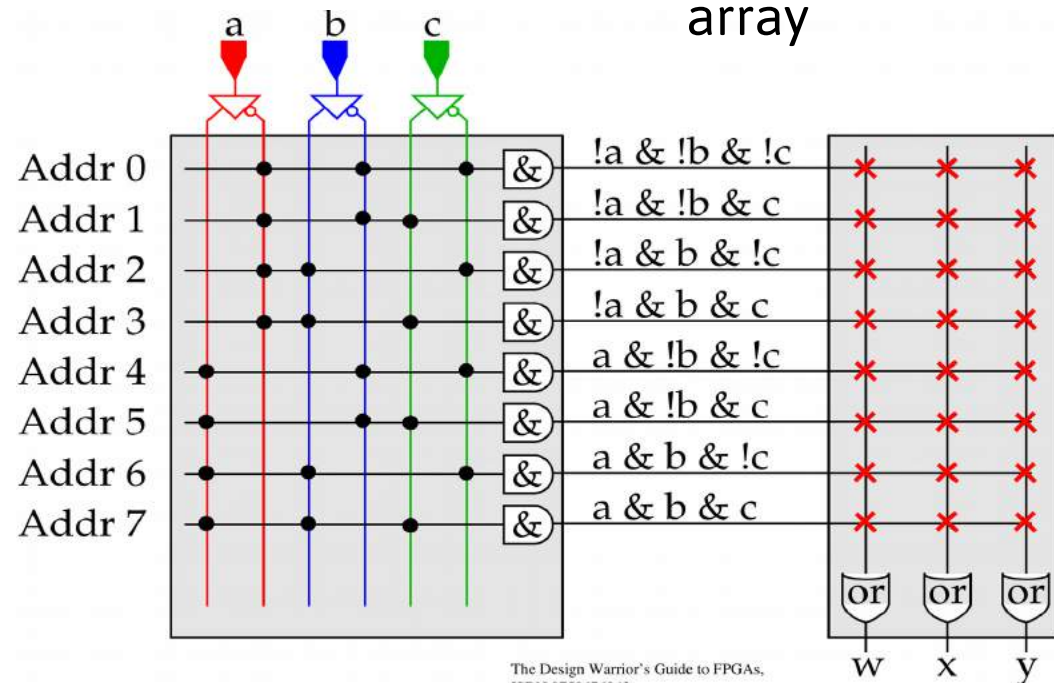
	(in system)	No	or offline)
	Fast	----	3x slower than SRAM
	Yes	No	No (but can be if required)
	Yes	No	No
	Yes (very good)	No	Yes (reasonable)
	No	Yes	Yes
	Acceptable (especially when using bitstream encryption)	Very Good	Very Good
	Large (six transistors)	Very small	Medium-small (two transistors)

# PROM

- Implements sum of products
- Built from TWO arrays:
  - Fixed AND gates (example uses 3-input ANDs)
  - Programmable OR gates (variable 1 to 8 inputs)
- For the fixed AND array, all inputs outputs must be fabricated since you don't know which product terms are needed. (8 gates required to exhaustive cover possibility based on 3 inputs), whether they are needed or NOT
- For programmable OR portion, the programmer can decide which terms are needed. The number to fabricate in silicon is decided based on past experience of the manufacturer while the number to actually wire is decided by the programmer.

Fixed, exhaustive  
prewired AND  
Array

Programmable,  
typically non-  
exhaustive OR  
array



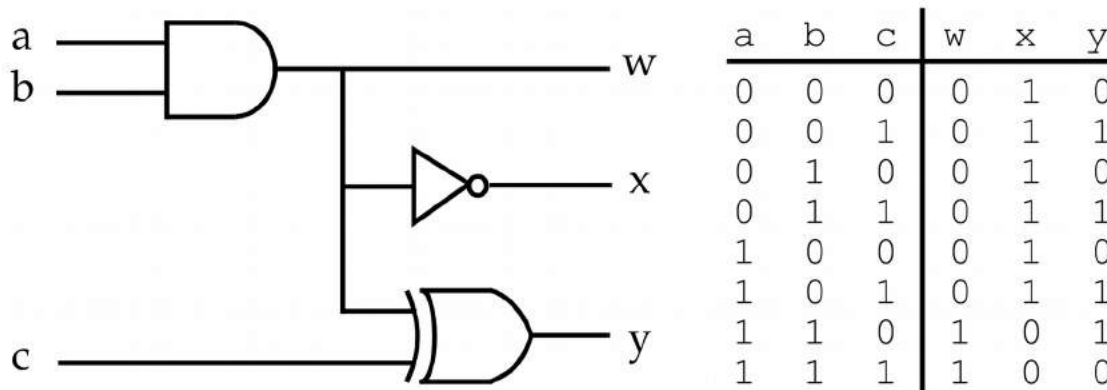
The Design Warrior's Guide to FPGAs,  
ISBN 0750676043,  
Copyright(C) 2004 Mentor Graphics Corp

## PROMs

PROMs were originally intended for use as *computer memories* to store programs and constant data.

However, engineers used them to implement lookup tables and state machines.

PROMs can be used to implement any block of combinational logic.



The Design Warrior's Guide to FPGAs,  
ISBN 0750676043,  
Copyright(C) 2004 Mentor Graphics Corp

Programming these functions is a simple matter of choosing the correct links in the OR array.

NOTE: Real PROMs have significantly more inputs and outputs.

As you know from previous courses, any truth table can be translated to a boolean sum of products or product of sums.

# PLA

- Implements sum of products
- Built from TWO **Programmable** arrays:
  - **Programmable** AND gates (example uses 3-input ANDs )
  - Programmable OR gates (variable 1 to 8 inputs)
- For the **programmable** AND array, not all inputs outputs must be fabricated. The number decided by the manufacturer sets how many product terms can be implemented
- Down side is that programmable technology tends to be slower...PLAs were never significant

**Programmable,  
non-exhaustive  
AND Array**

Programmable,  
typically non-  
exhaustive  
OR/NOR array

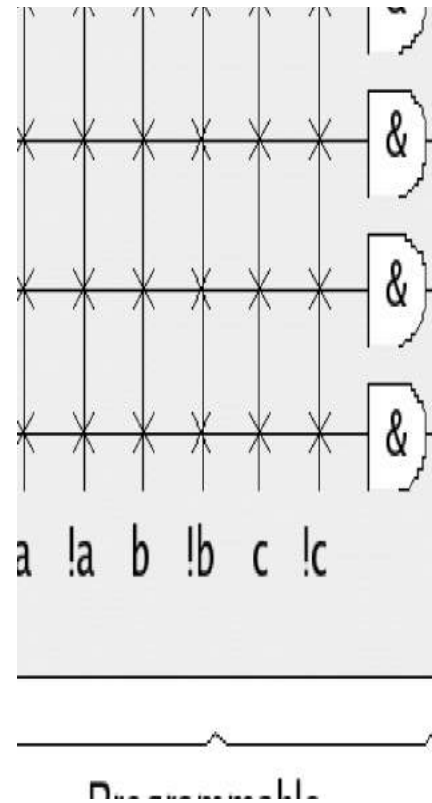
$$\begin{array}{l}
 c) \mid (!b \ \& \ !c) \\
 ) \ \& \ c) \mid (!b \ \& \\
 ) \ \& \ c) \quad \text{—————}
 \end{array}$$



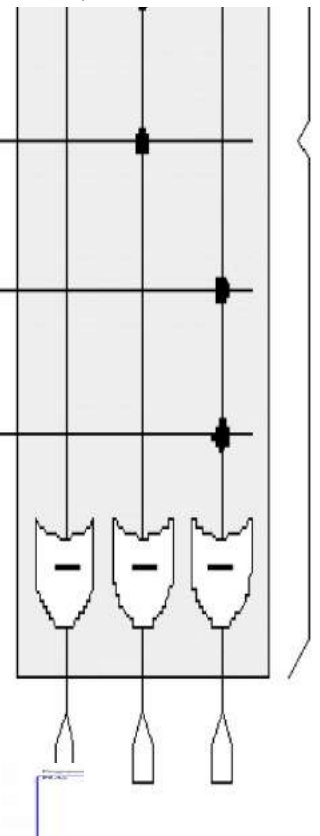

# PALs

- Implements sum of products
- Built from TWO arrays in opposite approach of PROM:
  - **Programmable** AND gates (example uses 3-input ANDs)
  - **Predefined** OR gates
- Introduced in late 70s to address speed problems with PLAs. Since second array is not programmable it was designed to be faster.
- Downside compared to PLAs is the limited number of terms that can be OR'ed (illustrated is only two at a time)

**Programmable,  
non-exhaustive  
AND Array**



Fixed,  
**predefined**  
typically non-  
exhaustive  
OR/NOR array



# More Complex Devices

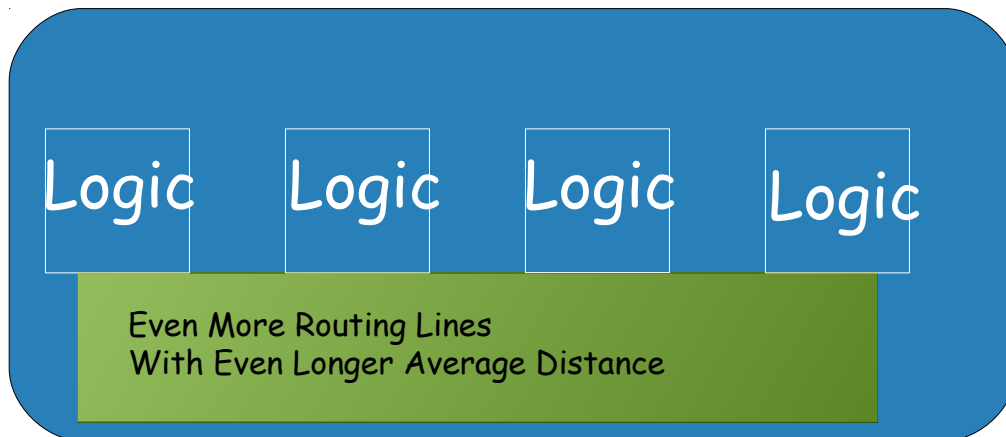
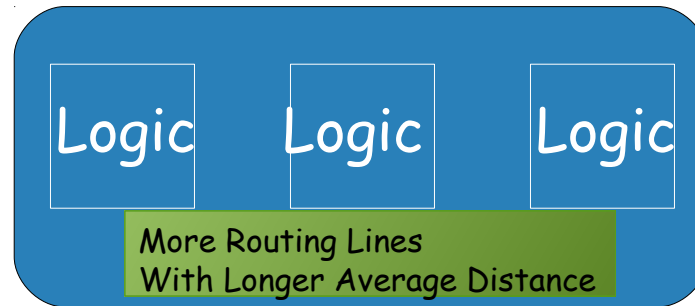
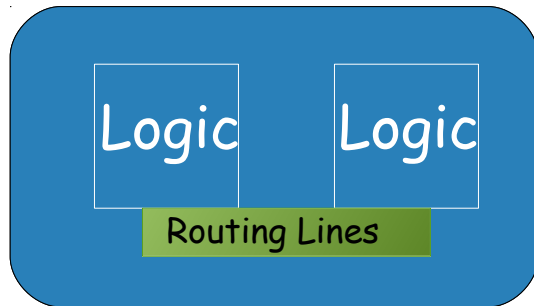
---

- Devices sold supported additional features
  - Programmable Output Inversion
  - Tristable Outputs (useful for buses and wired-or logic)
  - Ability to latch outputs
  - Configure pins as input or output (useful on smaller packages with limited physical pins)
- CPLDs replaced PLDs (which became known as SPLDs)
  - Were essentially multiple SPLDs on a chip but a key new enable technology was programmable interconnect (by Altera)
  - Introduced in '84 by Altera (now Intel) based on CMOS and EPROM technologies

# Cost of Interconnect on Complex Programmable ICs

- As more blocks are added, exhaustive or fixed % global interconnect grows rapidly ( $>O(n)$ ), in area, average delay, power...

Less percentage of IC was “logic” and more was interconnect

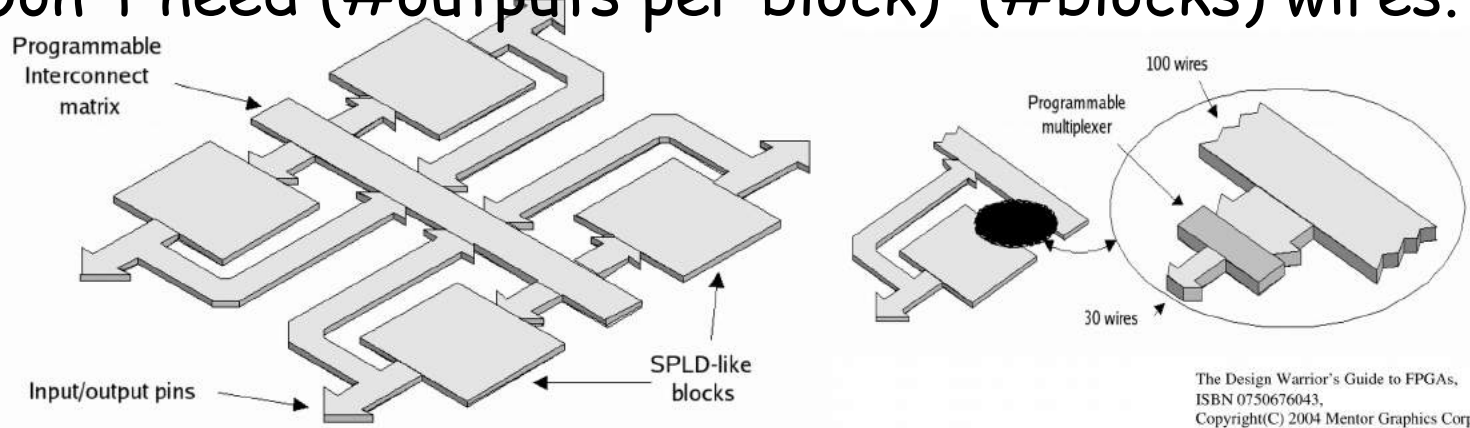


A replacement for exhaustive interconnect was critical...

## CPLDs

A generic **CPLD** structure typically consists of several **SPLD blocks** sharing a common **programmable interconnection matrix**.

Don't need  $(\# \text{outputs per block})^{(\# \text{blocks})}$  wires.



Both the SPLDs (usually PALs) and the interconnect can be programmed.

Interconnection matrix usually has more wires than the individual SPLD blocks

Therefore, a MUX is used to connect them.

The programmable switches may be EPROM, EEPROM, FLASH or SRAM based.



**PALASM, JEDEC, etc.**

In the early days, the design flow consisted of a hand-drawn schematic diagram that was later converted to tabular format and typed into a file.

The file (used by the *device programmer*) defined which fuses were to be blown (or which antifuses were to be grown).

Each PLD vendor developed its own file format, which made this task time consuming and error prone.

The *Joint Electron Device Engineering Council* (JEDEC) intervened and defined a standard language that everyone adopted.

*PAL Assembler* (PALASM) was also developed and allowed designers to specify the function in a sum-of-products form.

PALASM read the *HDL src file* and generated the text programming file.

PALASM and other early HDLs laid the foundation for Verilog and VHDL, and synthesis tools used today for ASIC and FPGA designs.

ABEL and CUPL are two other languages designed for programming CPLDs



---

**ASICs**

**ASIC (gate array, etc.)**

Four main classes exist today, in order of increasing complexity:

- Gate arrays
- Structured ASICs
- Standard cell devices
- Full-custom chips

**Full-custom**

In the early days, only two classes of chips existed.

- Standard off-the-shelf components
- Full-custom ASICs (such as microprocessors)

For the full-custom class, nothing is predefined, not even standard logic gates.

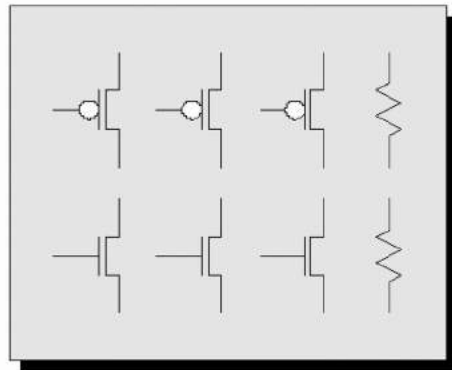
All wires and gates are hand-crafted individually, and optimized for speed, area and power.



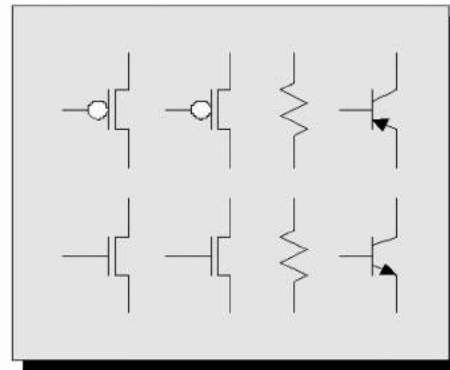


### ASIC (gate array, etc.)

Gate arrays are based on the idea of a *basic cell* consisting of a collection of unconnected transistors and resistors.



(a) Pure CMOS basic cell



(b) BiCMOS basic cell

The Design Warrior's Guide to FPGAs,  
ISBN 0750676043,  
Copyright(C) 2004 Mentor Graphics Corp

The ASIC vendor *prefabricates* silicon chips containing arrays of these **basic cells**.

Channels are typically provided between rows or columns of these arrays for routing.

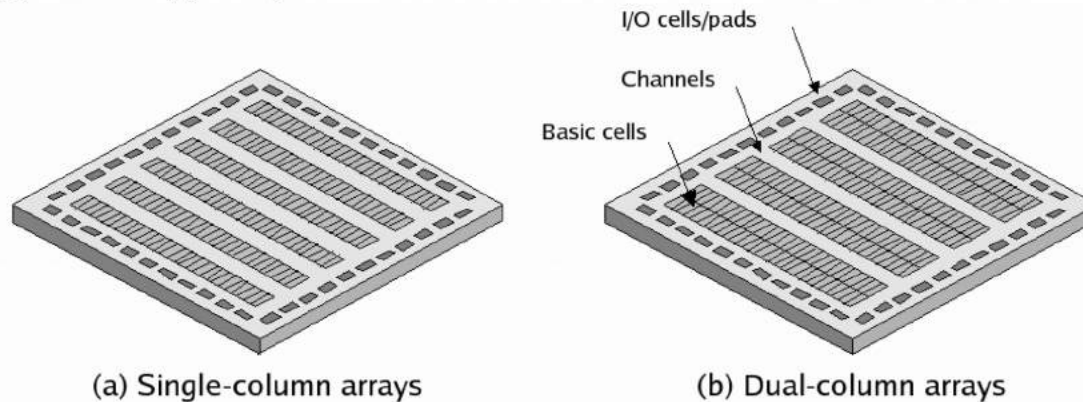
Alternative, *sea-of-gates* arrays do not have routing channels.

The **vendor also provides a cell library** (a set of basic logic gates, MUXs, etc).

**Gate Arrays: Cheaply build arrays of FETs but don't connect them until later.**



## ASIC (gate array, etc.)



The Design Warrior's Guide to FPGAs,  
ISBN 0750676043,

Designers use the latter and generate a *netlist*.

Special mapping, placement and routing CAD tools are used to assign the logic gates to *basic cells* and to design the interconnect.

The output of this process are *photo-masks* that define the metalization layers.

Since the transistors and other components are **prefabricated**, **there is a considerable cost savings**.

The main drawback is *resource under-utilization* and *less-than-optimal routing* (because of routing constraints).

Vendors provided well-characterized blocks and supported tool flows or rapidly "designed" IC in-house

### Standard Cell Devices

Became available in the early '80s to address the problems with gate arrays.

Similar to gate arrays, the ASIC vendor defines the *cell library*.

Designers **generate a gate-level netlist** using CAD tools (similar to gate arrays) but none of the components are prefabricated.

P&R tools **place and route** each of the gates and optimize for area and delay.

The standard cells themselves are designed as **constant height cells**, which simplifies their placement (PWR and GND are connected via abutment).

The output of the process is a *complete* set of photomasks.

### Pre-designed Blocks:

The **vendor supplies hard-macro and soft-macro libraries**, which include elements such as processors, comm. functions, RAM, ROM, etc.

**Designers also have the option of purchasing blocks of intellectual property (IP).**



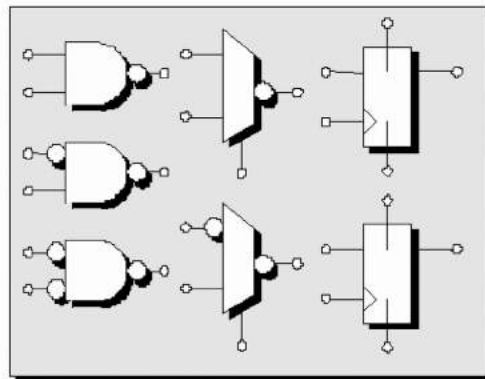
## Structured ASICs

Entered the scene in the early '90s but were not accepted until '03.

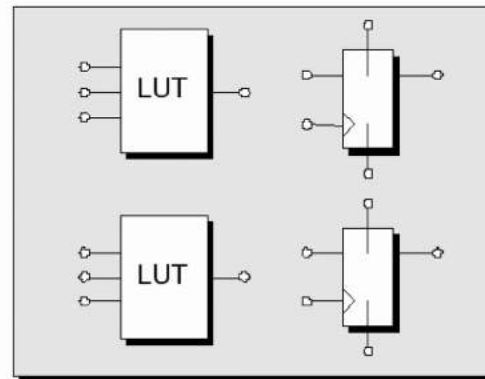
ASIC manufacturers were **looking for ways to reduce ASIC design costs** and *development times*, without reverting to gate arrays.

The basic element is called **a module or tile**.

It contains a mixture of prefabricated generic logic (implemented as gates, MUXs or lookup tables), registers and some local RAM.



(a) Gate, mux, and flop-based



(b) LUT and flop-based

The Design Warrior's Guide to FPGAs,  
ISBN 0750676043,  
Copyright(C) 2004 Mentor Graphics Corp

An array (sea) of these elements are prefabricated.

### Structured ASICs

**Peripheral elements** are also prefabricated, and include RAM blocks, clock generators, boundary scan logic, etc.

Similar to gate arrays, the chip can be **customized using only metalization layers**.

Since structured ASIC tiles are more complex than gate arrays, *most* of the metalization layers are predefined.

Most require only 2 or 3 layers to be customized, in the extreme, one requires the definition of only a single *via* layer.

This saves considerable time and cost since only a couple photo-masks need to be designed.

The disadvantages include about **3X more area and 2-3X more power**, when compared with a standard cell chip.

Among all the ASIC options: Tradeoff design time and cost versus level of customization and performance.

---

**FPGAs**

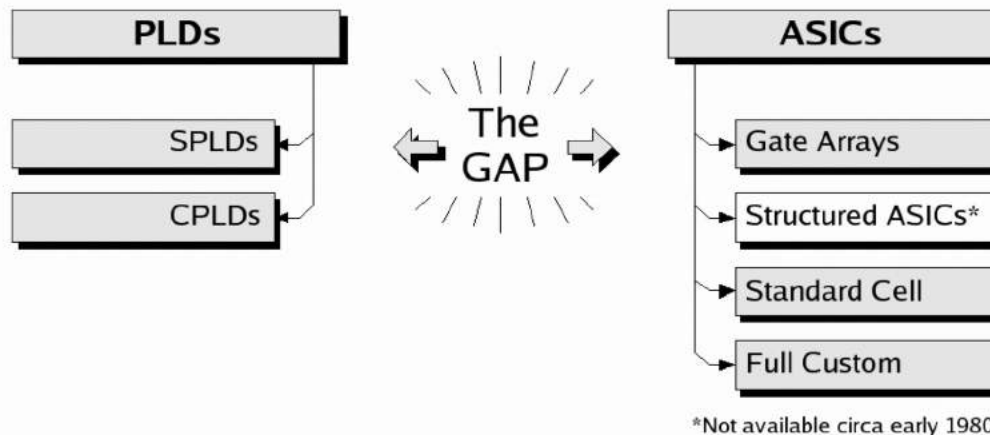


## FPGAs

In early '80s, a gap emerged in the digital IC continuum.

At one end, SPLDs and CPLDs provided high configurability, fast design and modification times, but supported only small to moderate functions.

At the other end, ASICs supported large complex designs but were immutable once fabricated, expensive and time-consuming to design.



The Design Warrior's Guide to FPGAs,  
ISBN 0750676043,  
Copyright(C) 2004 Mentor Graphics Corp

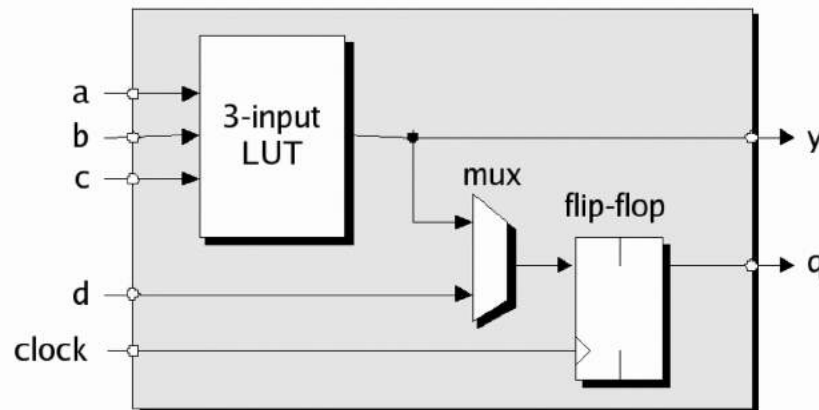
Xilinx developed and made available in '84 a new class of IC called the **FPGA** to fill the gap.



## FPGAs

The first FPGAs were **based on CMOS** and used **SRAM cells** for configuration.

The early chips used an array of **programmable logic blocks (PLBs)**, which comprised a 3-input **lookup table (LUT)**, a register and a MUX.



The Design Warrior's Guide to FPGAs,  
ISBN 0750676043,  
Copyright(C) 2004 Mentor Graphics Corp

Each PLB can be programmed individually to perform a unique function.

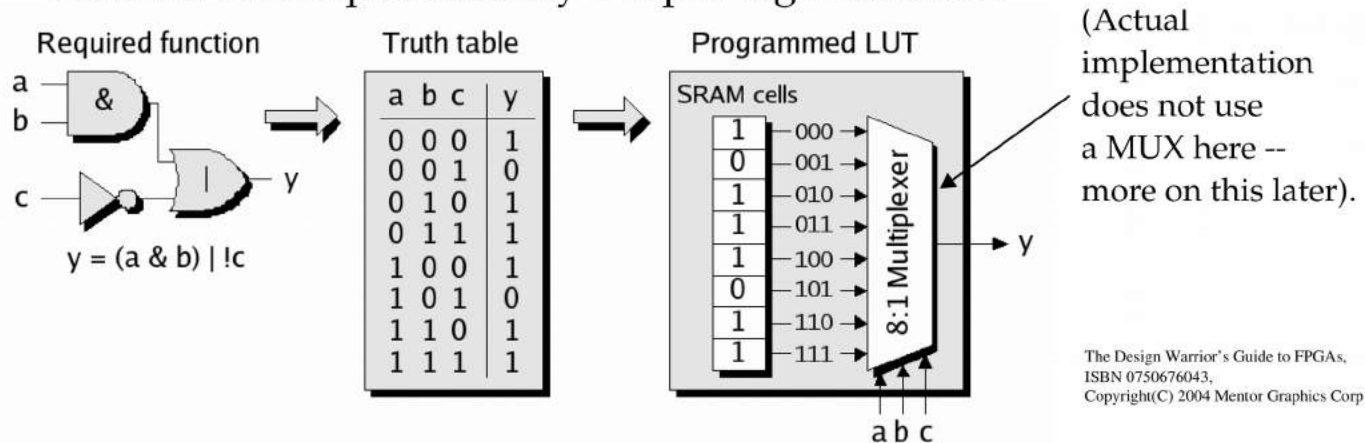
The FF can be triggered by a positive or negative-going clk.

The MUX allows selection of the LUT output or an external input.

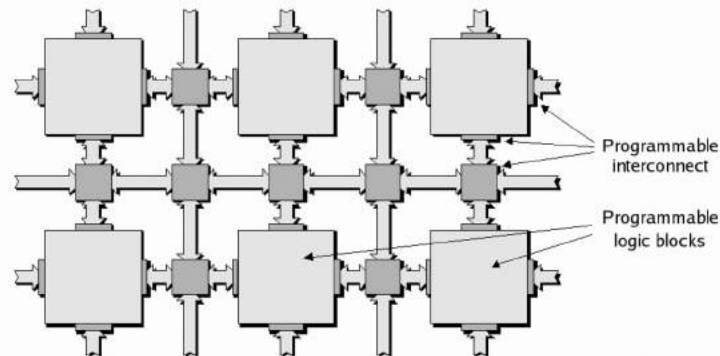


## FPGAs

The LUT can implement any 3-input logic function.



The FPGA architecture consisted of a **2-dimensional array of PLBs separated by a sea of programmable interconnect**.





## FPGAs

Today's FPGAs are much more complex (to be discussed).

For example, in addition to the local interconnect, an FPGA typically has a **global** (high-speed) **interconnection** network.

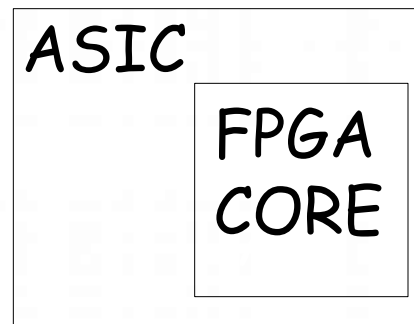
This allows signals to cross the chip without having to pass through local switching elements.

## FPGA-ASIC hybrids

Although it doesn't make sense to embed an ASIC inside an FPGA, it is meaningful in the other direction, i.e., embedded FPGA cores.

This is useful for *platform design*, a term used at the board level to refer to a design from which **multiple products** can be derived.

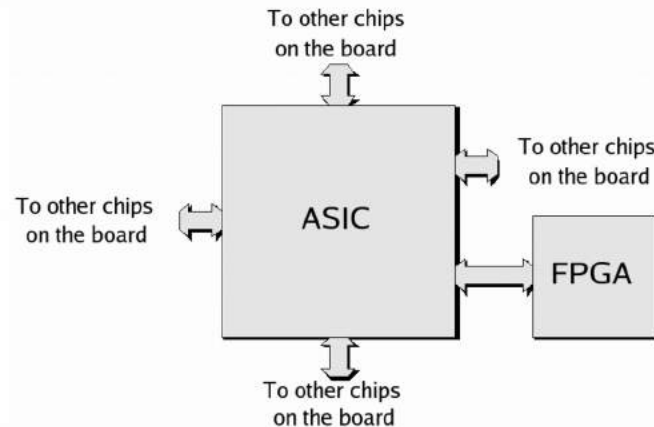
Here, the *platform* is the ASIC and the embedded FPGA allows it to be customized for a specific application.



## FPGA-ASIC hybrids

Another driver is provided by the increasing number of incidences of FPGAs being used to augment ASIC designs.

This has traditionally been accommodated at the board level.



The Design Warrior's Guide to FPGAs,  
ISBN 0750676043,  
Copyright(C) 2004 Mentor Graphics Corp

Here, the designers of the ASIC *off-loads* to the FPGA any part of the ASIC design that is subject to modifications or enhancements.

Board level realization incurs a performance penalty because signals must travel off-chip.

Embedding the FPGA solves this problem.

## FPGA-ASIC hybrids

Designing these hybrids however is challenging because ASIC and FPGA design tools/flows are significantly different underneath.

ASIC are **fine-grained** because they are implemented at the *primitive logic gate* level.

FPGAs are **medium-grained** (or **coarse-grained** according to others) because they are realized using higher-level blocks (PLBs).

The CAD tools that do the synthesis and P&R need a consistent view (fine-grained or medium grained) of the world in order to do a good job.

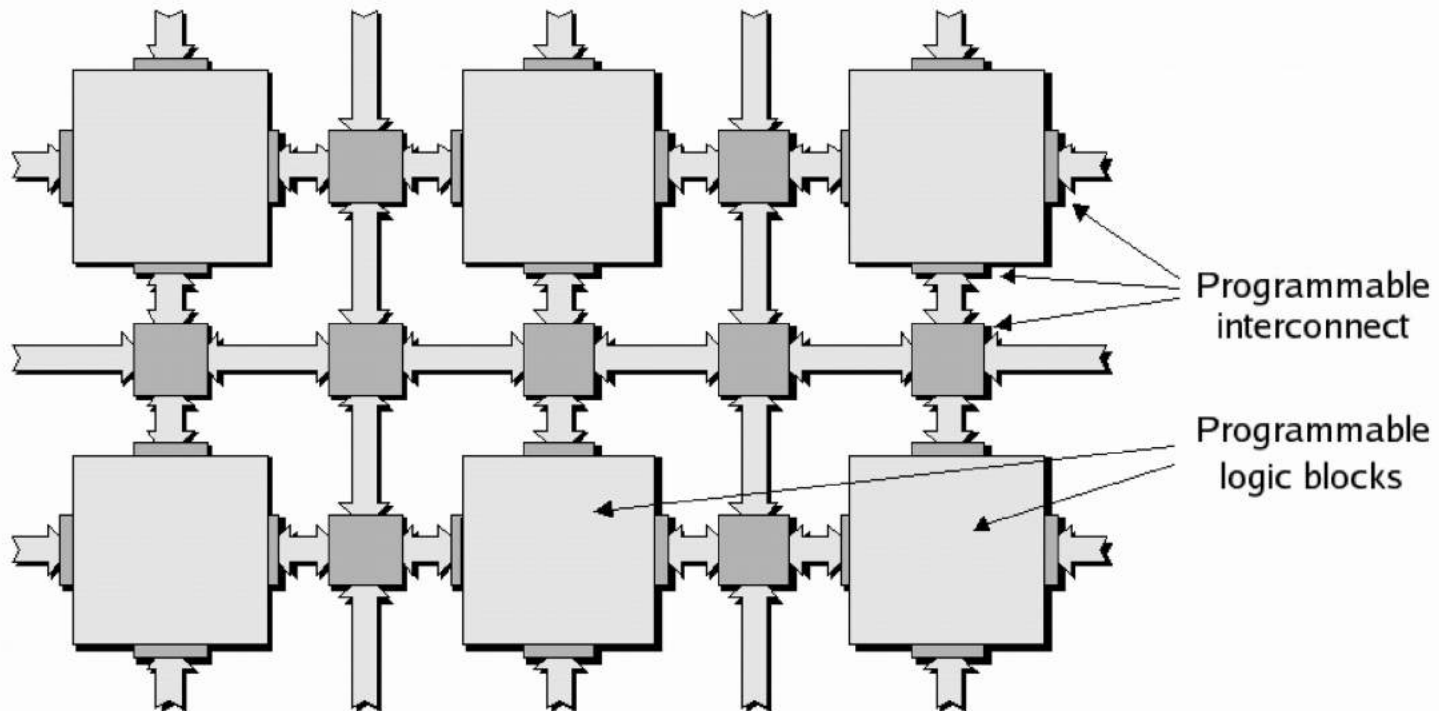
*Structured* ASIC and FPGAs however are both medium-grained and therefore can enjoy a unified tool and design flow.

That is, the same block-based synthesis and P&R engines can be used for both the ASIC and FPGA portions.



## Fine-, Medium-, and Course-Grained Architectures

Basic architecture consists of a large number of PLB islands embedded in a sea of programmable interconnect.



The Design Warrior's Guide to FPGAs,  
ISBN 0750676043,  
Copyright(C) 2004 Mentor Graphics Corp





### Fine-, Medium-, and Course-Grained Architectures

"Level of granularity", when used in the context of an FPGA refers to the complexity of the PLB.

The PLBs of **fine-grained** architectures can only implement simple functions, e.g., 3-input logic gate or storage element.

Good for glue logic, state machines, systolic algorithms (massively parallel), and traditional logic synthesis.

The PLBs of **medium-grained** architectures include more logic and more functionality, i.e., a 4-input LUTs, 4 MUXs, 4 D-FFs + fast carry logic.

This helps with the interconnect problem, e.g., more "compute power" per wire dedicated for interconnections.

**Large-grained** architectures incorporate FFT engines and microprocessor cores.

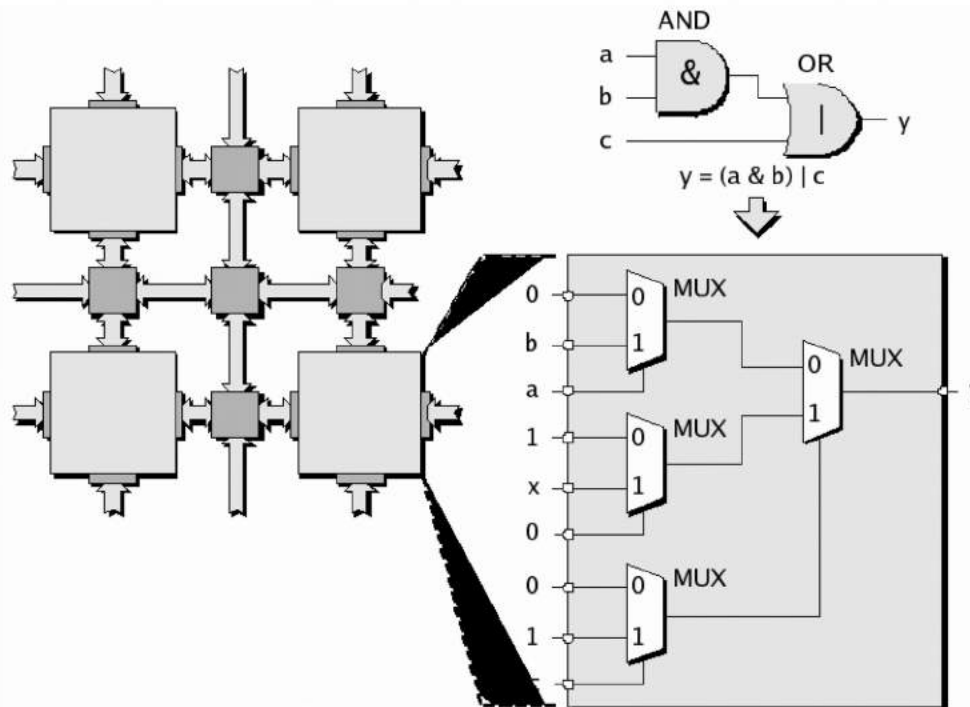
*Fine-grained* architectures of the mid-90's gave way to the *medium grained* architectures.



## MUX- vs. LUT-based Logic Blocks

There are 2 basic flavors of PLBs for *medium-grained* architectures, *multiplexer* (MUX) and *lookup table* (LUT).

In the MUX-based version, each input can be programmed with a logic 0, 1, or the true or inverted version of a variable.



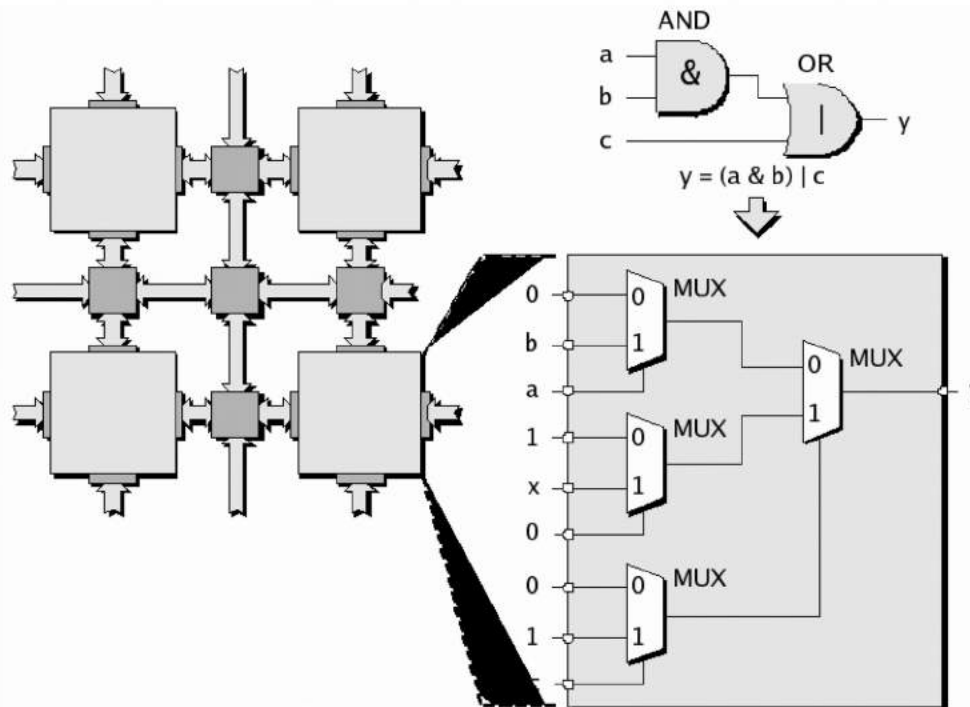
The Design Warrior's Guide to FPGAs,  
ISBN 0750676043,  
Copyright(C) 2004 Mentor Graphics Corp

Any logic can be built from only muxes.  
For review: implement  $(a \wedge (\neg b))$  using a 2-input mux

## MUX- vs. LUT-based Logic Blocks

There are 2 basic flavors of PLBs for *medium-grained* architectures, *multiplexer* (MUX) and *lookup table* (LUT).

In the MUX-based version, each input can be programmed with a logic 0, 1, or the true or inverted version of a variable.

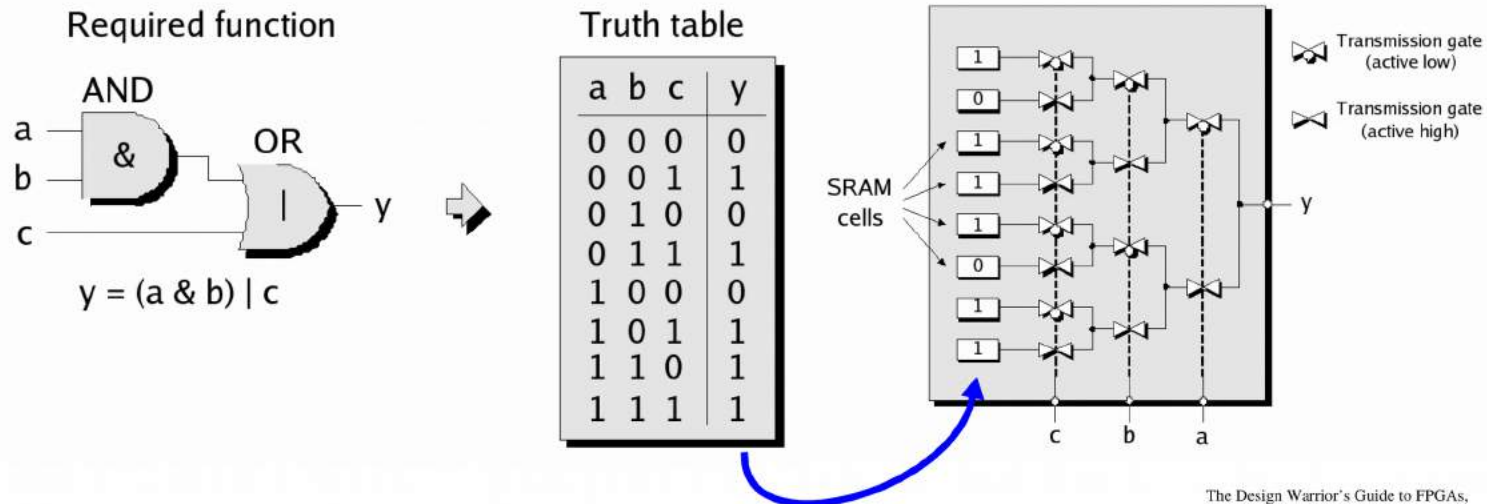


The Design Warrior's Guide to FPGAs,  
ISBN 0750676043,  
Copyright(C) 2004 Mentor Graphics Corp

Any logic can be built from only muxes.

## MUX- vs. LUT-based Logic Blocks

Most FPGAs today are LUT-based -- here, the input signals are used as a pointer into a lookup table.



The Design Warrior's Guide to FPGAs,  
ISBN 0750676043,  
Copyright(C) 2004 Mentor Graphics Corp

Input signals can be decoded using a hierarchy of *transmission-gate* MUXs.

Transmission gates *pass* the value on their inputs or are **high-impedance**.

Note that the diagram does not show the serial connection of the cells (scan chain) for simplicity.

Transmission gates are basically switches implemented with FETs.



## MUX- vs. LUT-based Logic Blocks

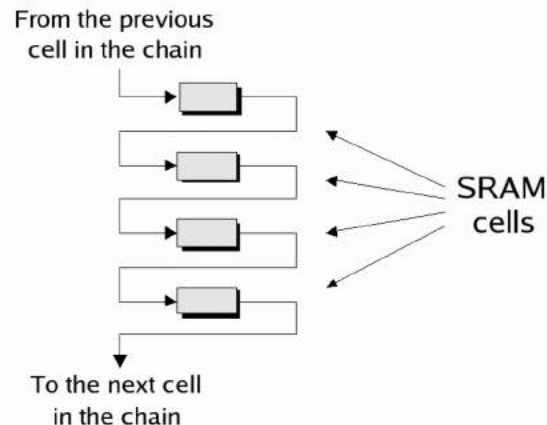
Larger LUTs are possible, e.g., 3-, 4-, 5- and 6-input versions.

Every time an input is added, the size of the table doubles.

4-input versions are believed to provide the optimal balance today.

Some vendors allow the 16 cells in the LUT to play the role as a *16x1 RAM*, and sets to be strewn together to form larger RAMs.

Some vendors allow the 16 cells of the LUT to be decoupled from the larger chain and used as a shift register.



The Design Warrior's Guide to FPGAs,  
ISBN 0750676043,  
Copyright(C) 2004 Mentor Graphics Corp

Basically,  
can use  
memory  
functionally  
as memory  
instead of as  
logic

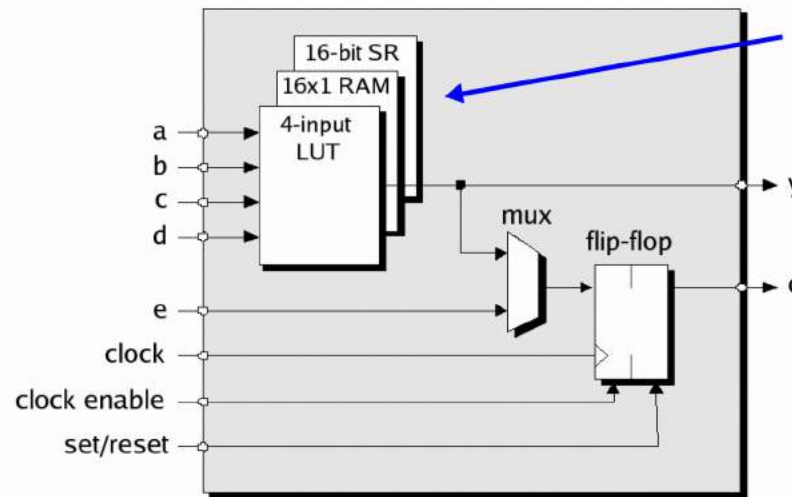


---

**Moving up the internal hierarchy**

## Terminology

Xilinx calls them *logic cells* (LC) while Altera calls them *logic elements* (LE).



Any one of several functions

As described,  
LUT is more  
versatile for feature  
expansion than MUX

The Design Warrior's Guide to FPGAs,  
ISBN 0750676043,  
Copyright (C) 2004 Mentor Graphics Corp.

A *slice* is defined as 2 LCs.

Each instance has its own inputs and outputs but the *clock*, *clock enable*, and *set/reset* signals are common.

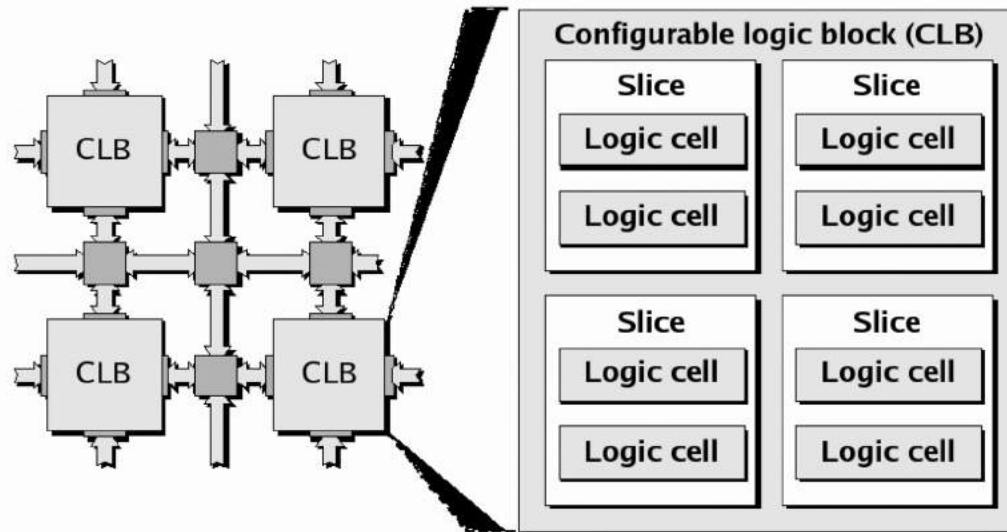
A *configurable logic block* (CLB, Xilinx) or *logic array block* (LAB, Altera).

CLBs consist of 2 or 4 *slices*, and conform to the islands shown earlier.



## Terminology and Hierarchy

The CLB also has some fast interconnect (not shown), that is used to connect neighboring slices.



The Design Warrior's Guide to FPGAs,  
ISBN 0750676043,  
Copyright(C) 2004 Mentor Graphics Corp

The organization of *LC* -> *Slice* -> *CLB* is complemented by an equivalent hierarchy in the interconnect.

That is, fast interconnect between LCs in a slice, slightly slower between slices in a CLB, followed by the interconnect between CLBs.

FPGA IC designers have to design local and global routing (and switches) and decide how much space to allocated to each

# Complex Logic Block Cells

- the CLB used in the XC4000 series of Xilinx FPGAs. This is a fairly complicated basic logic cell containing 2 four-input LUTs that feed a three-input LUT. The XC4000 CLB also has special fast carry logic hard-wired between CLBs. MUX control logic maps four control inputs (C1–C4) into the four inputs: LUT input H1, direct in (DIN), enable clock (EC), and a set / reset control (S/R) for the flip-flops. The control inputs (C1–C4) can also be used to control the use of the F' and G' LUTs as 32 bits of SRAM.

