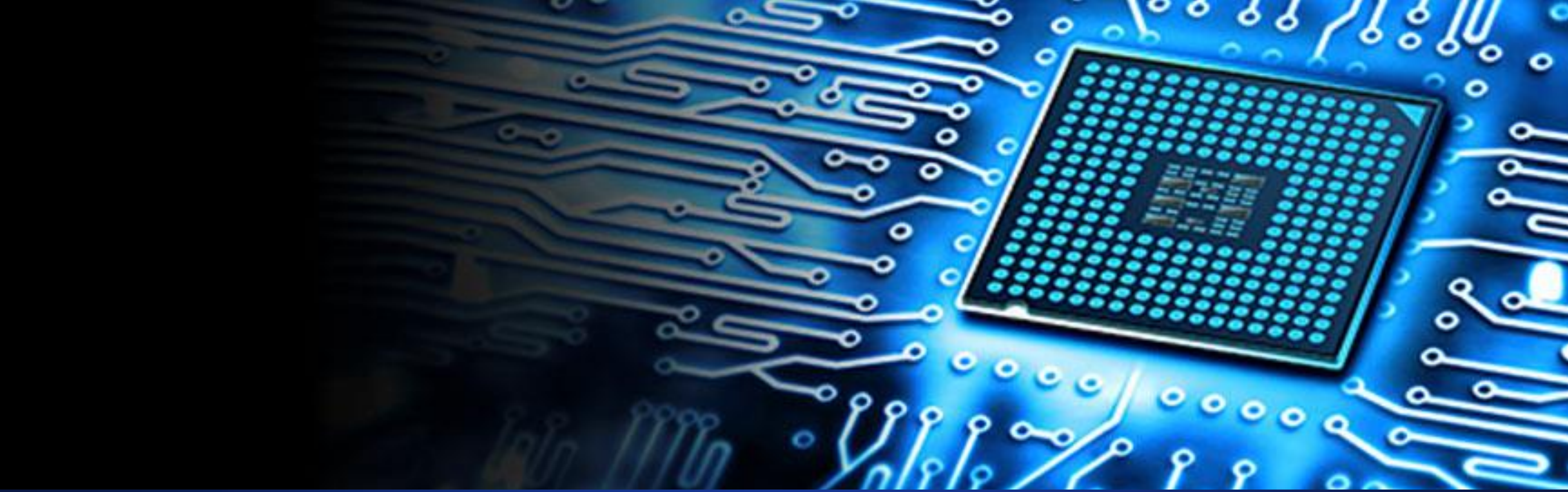
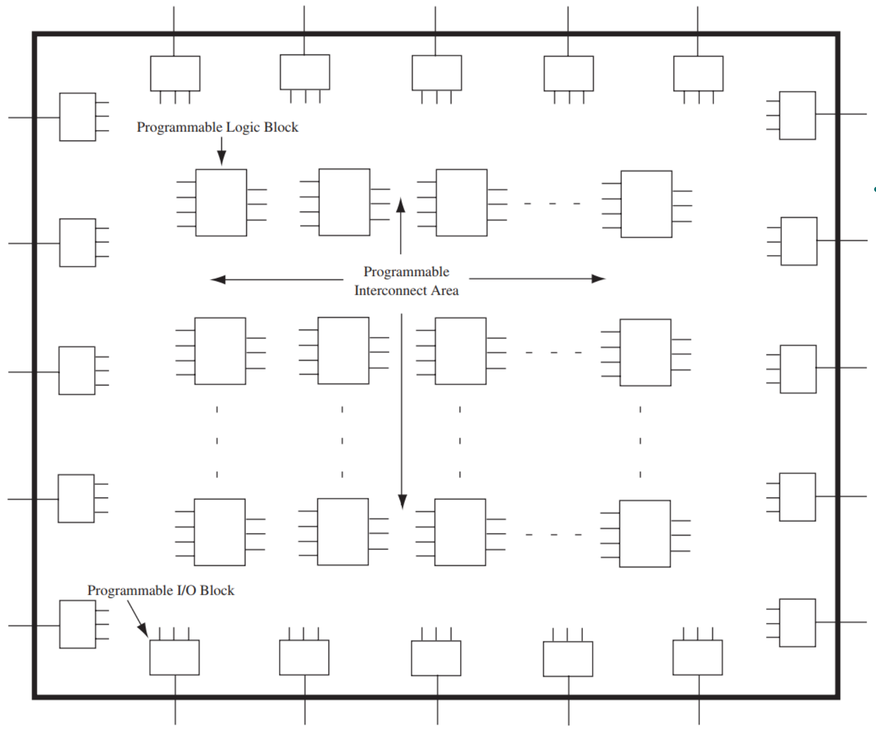


Chapter 3
Introduction to FPGAs

Version: 2024/01/02

FPGA: Field Programmable Gate Array

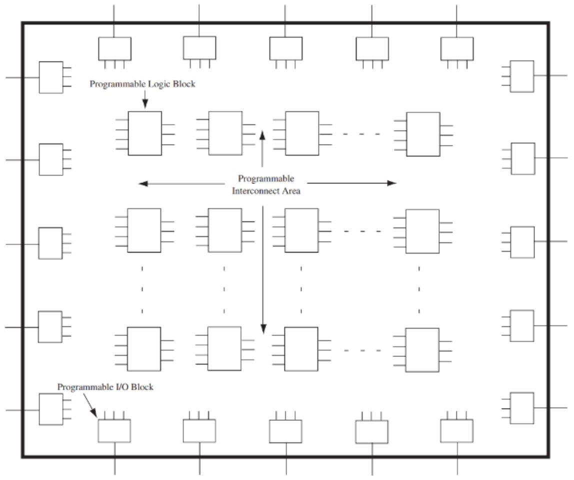


FPGA is IC that contain

- an array of identical logic blocks with
- programmable interconnections

The user can program the functions realized by each logic block and the connections between the blocks

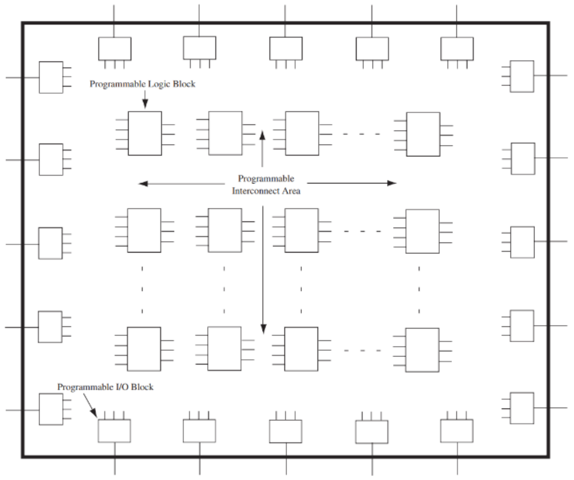
Advantages of FPGA



FPGAs provide several advantages over traditional gate arrays or mask programmable gate arrays (**MPGAs**)

- A traditional gate array can be used to implement any circuit, but it is programmable only in the factory
- A specific mask to match the particular circuit is created in order to fabricate the gate array
- The design time of a gate-array-based IC is a few months

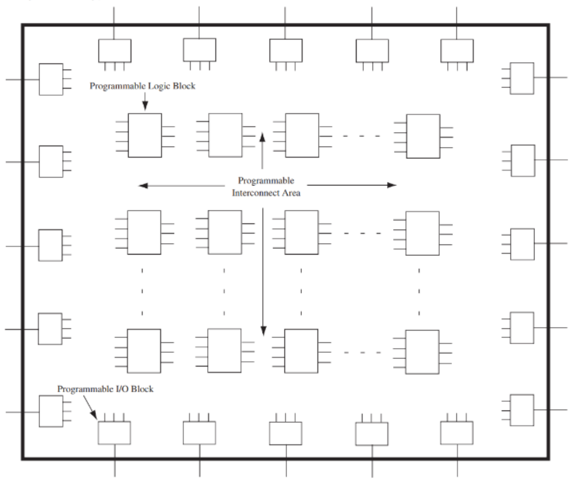
Advantages of FPGA



- FPGAs are standard off-the-shelf products. Manufacturing time reduces from months to hours as one adopts FPGAs instead of MPGAs
- Design iterations become easier with FPGAs. This is a tremendous advantage when it comes to time to market

- It becomes easy to correct mistakes that creep into designs. Mistakes and design specification changes become less costly
- Prototyping cost is reduced. At low volumes, FPGAs are cheaper than MPGAs

Disadvantages of FPGA



- FPGAs are less dense than traditional gate arrays
- In FPGAs, a lot of resources are spent merely to achieve the needed programmability

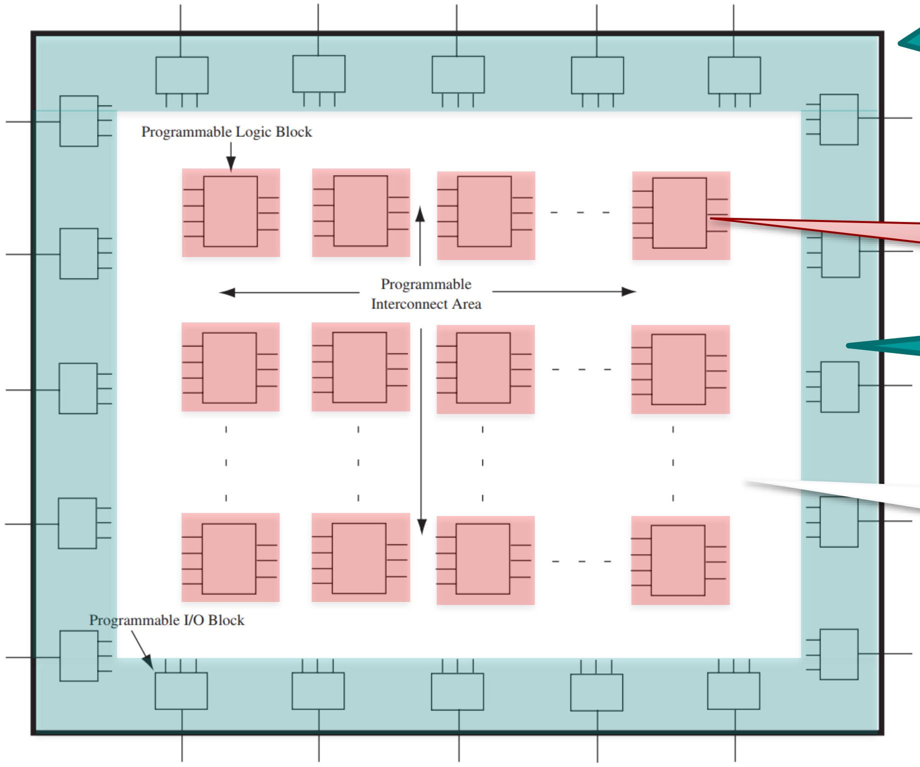
- Programmable points have resistance and capacitance. They slow down signals, so FPGAs are slower than traditional gate arrays
- Interconnection delays are unpredictable in FPGAs

	Contents
1	**Organization of FPGAs**
2	FPGA Programming Technologies
3	Programmable Logic Block Architectures
4	Programmable Interconnects
5	Programmable I/O Blocks in FPGAs

3.4.1 Organization of FPGAs

"Field" programmability is achieved by reconfigurable elements, which can be programmed or reconfigured by the user

Layout of a Typical FPGA



The interior of FPGAs typically contains three elements that are programmable

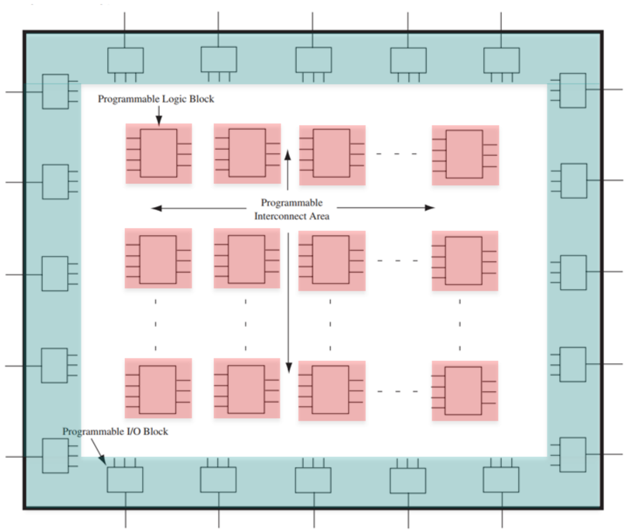
Programmable logic blocks

Programmable input/output blocks

Programmable routing resources

3.4.1 Organization of FPGAs

Layout of a Typical FPGA

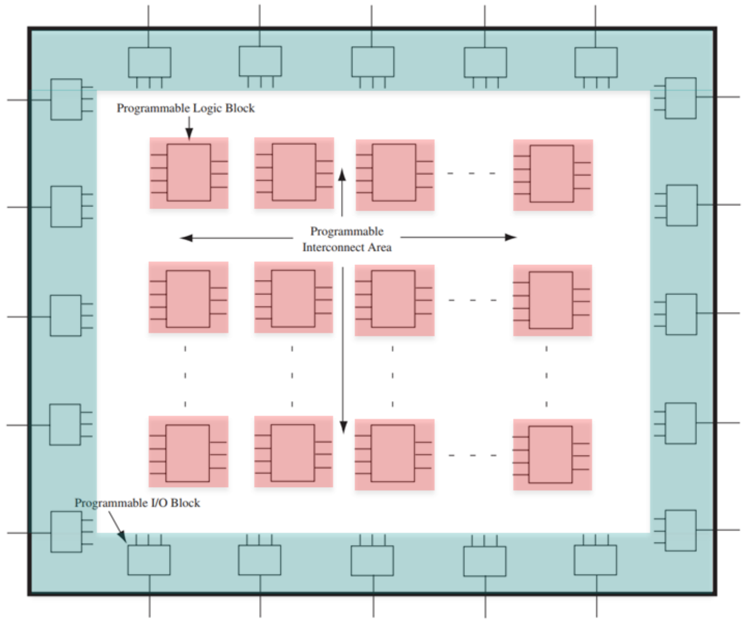


- Arrays of programmable logic blocks are distributed within FPGA
- Logic blocks are surrounded by input/output (I/O) interface blocks

- These I/O blocks can be considered to be on the periphery of the chip. They connect the logic signals to FPGA pins
- The space between the logic blocks is used to route connections between the logic blocks

3.4.1 Organization of FPGAs

Layout of a Typical FPGA



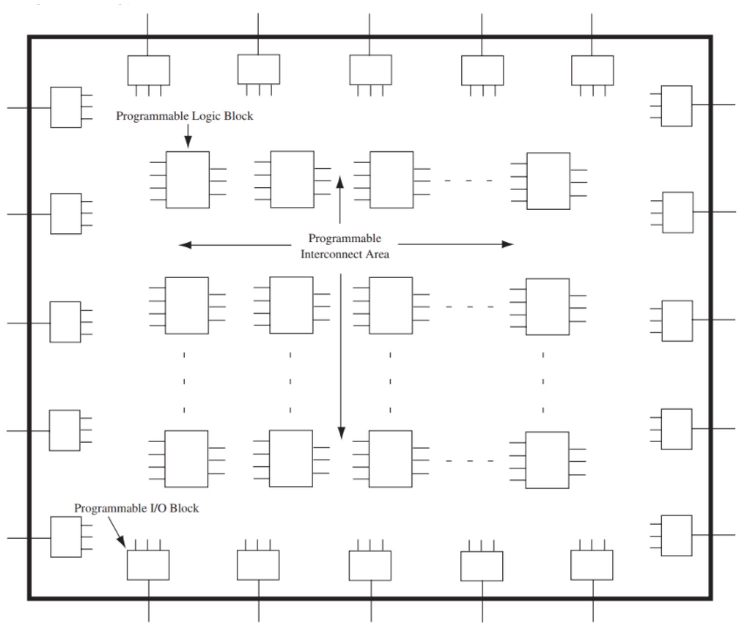
1. **Programmable logic blocks**: created by using multiplexers, look-up tables, and AND-OR or NAND-NAND arrays

"Programming" means:

- changing the input or control signals to the multiplexers
- changing the look-up table contents, or
- selecting or not selecting particular gates in AND-OR gate blocks

3.4.1 Organization of FPGAs

Layout of a Typical FPGA

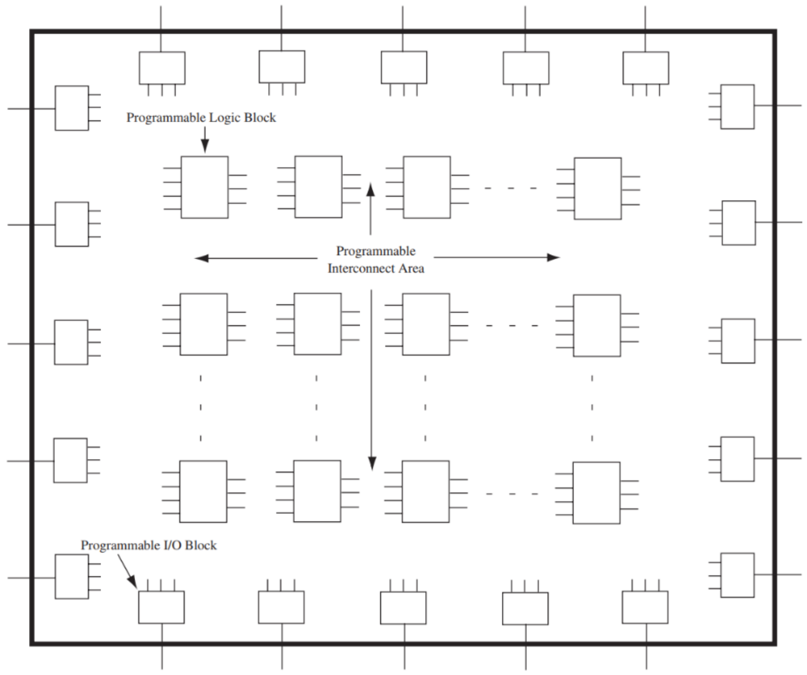


2. **Programmable interconnect**:
- be required to interconnect various blocks in the chip and to connect specific I/O pins to specific logic blocks
- "programming" means making or breaking specific connections

3. **Programmable I/O blocks**:
- can be programmed to be input, output, or bidirectional lines

3.4.1 Organization of FPGAs

Layout of a Typical FPGA

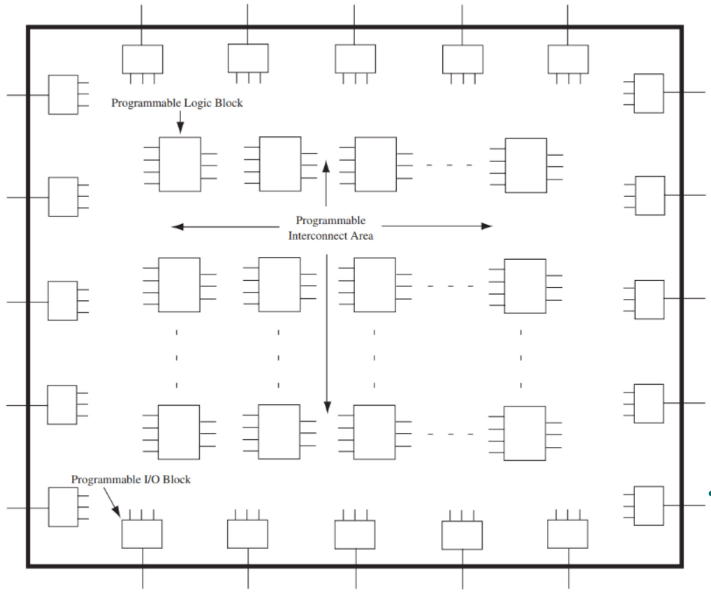


- The general-purpose interconnect gives FPGA a lot of flexibility
- But it has the disadvantage of being slow

- A connection from one part of the chip to another part might have to travel through several programmable interconnect points
- This results in large and unpredictable signal delays

3.4.1 Organization of FPGAs

Layout of a Typical FPGA



FPGA architecture or organization:
the manner or topology in which the logic blocks and interconnect resources are distributed inside the FPGA

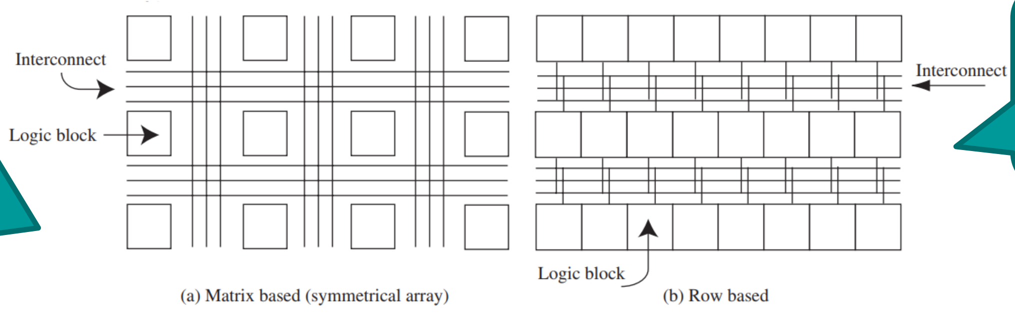
- General structure of an FPGA, not all FPGAs look like that
- FPGAs use a variety of architectures

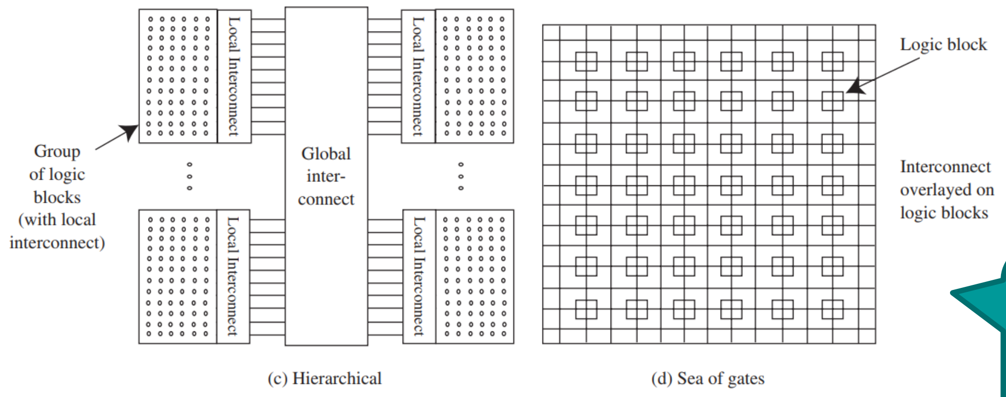
3.4.1 Organization of FPGAs

Typical Architectures for FPGAs

Matrix-based (symmetrical array) architectures
矩阵（对称阵列）型

Row-based architectures
横向型



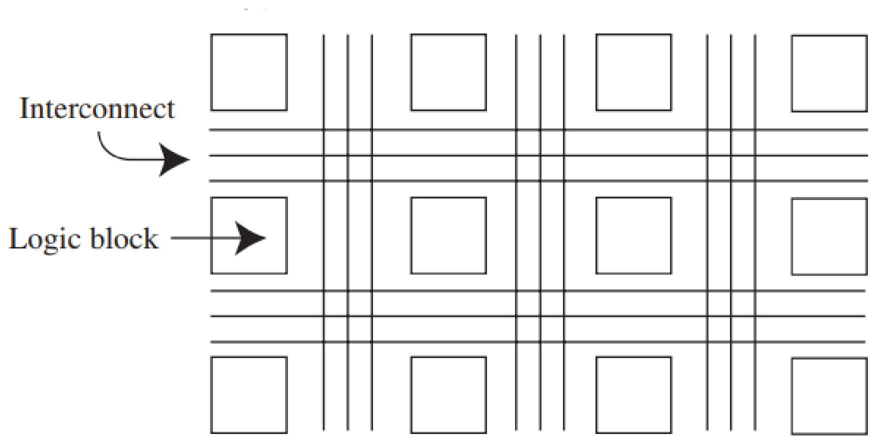


Hierarchical PLD architectures
从属型

Sea-of-gates architecture
门海型

This classification is based on the layout of the general-purpose logic region in the FPGAs

Matrix-Based (Symmetrical Array) Architectures

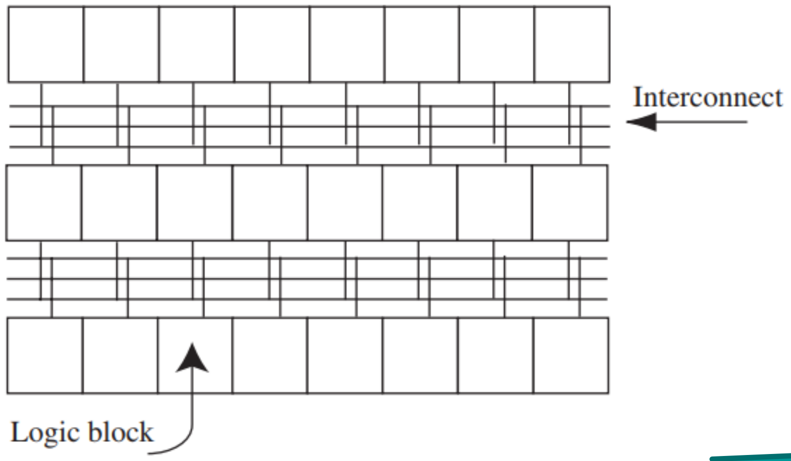


The logic blocks are organized in a matrix-like fashion

- The logic blocks in these architectures are typically of a large granularity (capable of implementing 4-variable functions or more)
- These architectures typically contain 8 x 8 arrays in the smaller chips and 100 x 100 or larger arrays in the bigger chips

Two-dimensional channeled routing (二维通道布线):
routing resources are generally available in horizontal and vertical directions

Row-Based Architectures

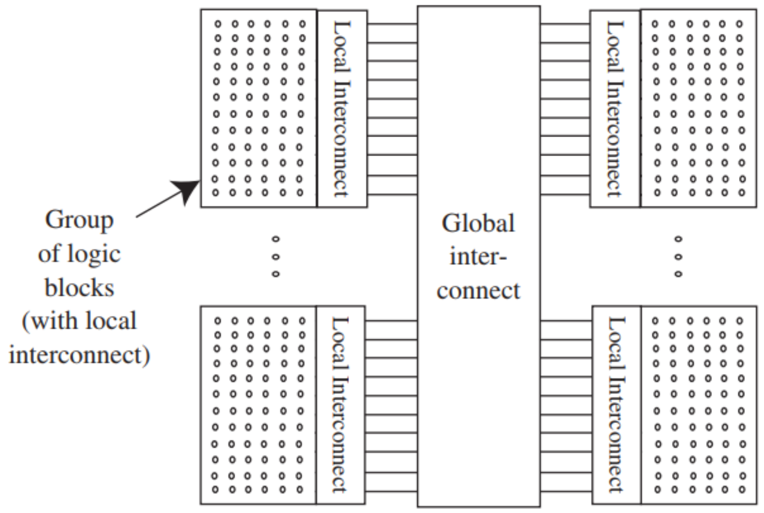


- The logic blocks in this architecture are organized into rows
- There are rows of logic blocks and routing resources

One-dimensional channeled routing（一维通道布线）：
the routing resources are located as a channel in between rows of logic resources

Hierarchical Architectures

Hierarchical Architectures

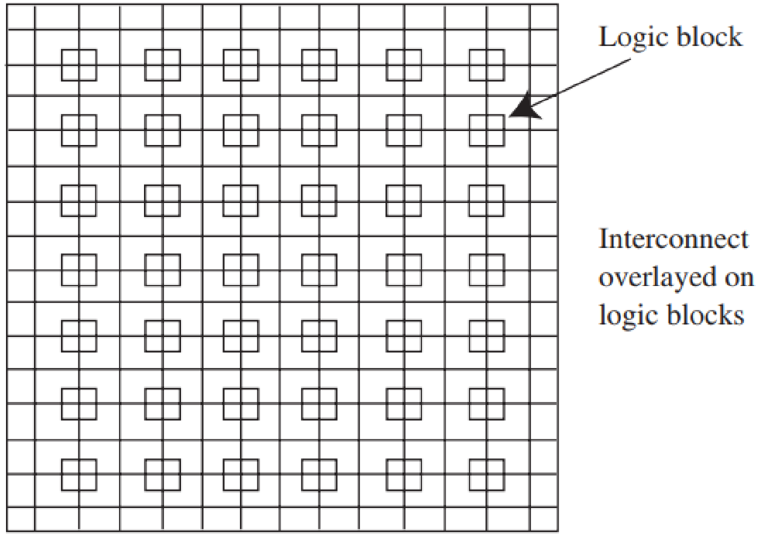


- ➢ Blocks of logic cells are grouped together by a local interconnect
- ➢ Several such groups are interconnected by another level of interconnect

There is a hierarchy in the organization of these FPGAs

- ➢ These FPGAs contain clusters of logic blocks with localized resources for interconnection
- ➢ The global interconnect network is used for the interconnections between the clusters of logic blocks

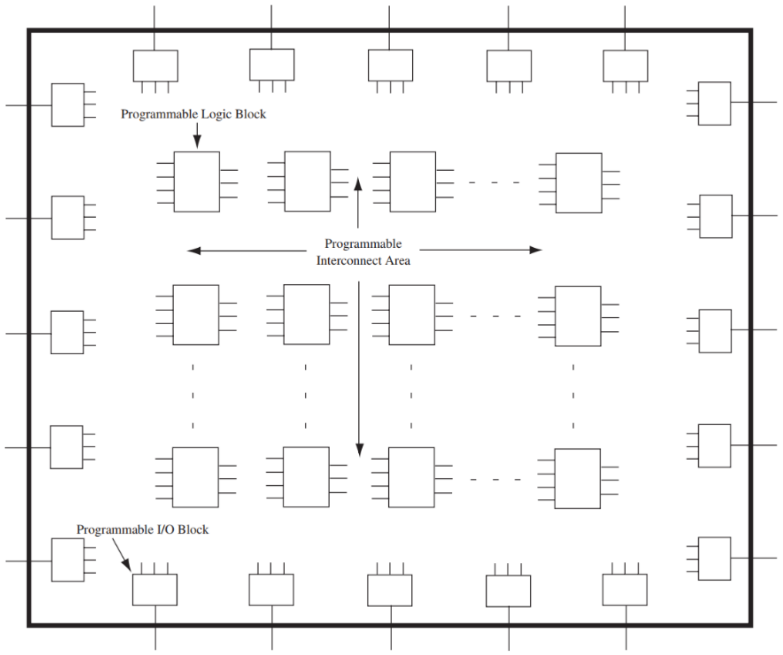
Sea-of-Gates Architectures



- The general FPGA fabric consists of a large number of gates
- There is an interconnect superimposed on the sea of gates

	Contents
1	Organization of FPGAs
2	**FPGA Programming Technologies**
3	Programmable Logic Block Architectures
4	Programmable Interconnects
5	Programmable I/O Blocks in FPGAs

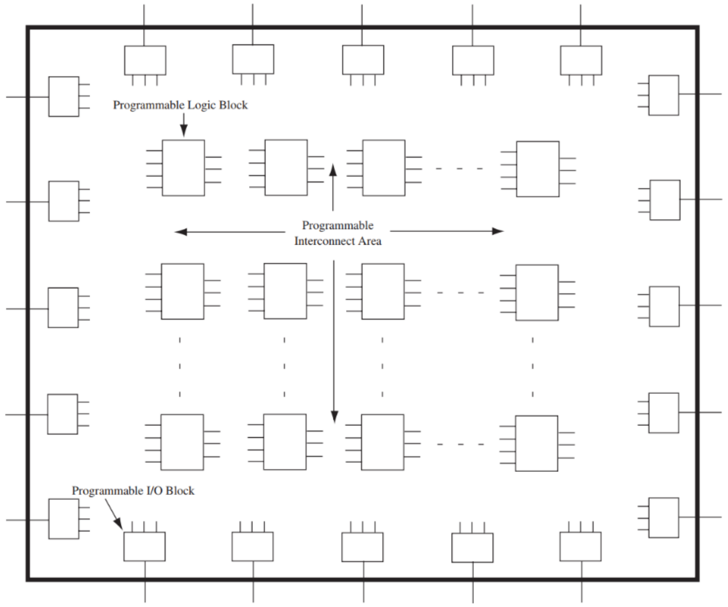
3.4.2 FPGA Programming Technologies



FPGAs consist of a large number of logic blocks interspersed with a programmable interconnect

- The logic block is programmable in the sense that the same building block can be “programmed” or “configured” to create any desired circuitry
- There is also programmability in the interconnections between the logic blocks

3.4.2 FPGA Programming Technologies



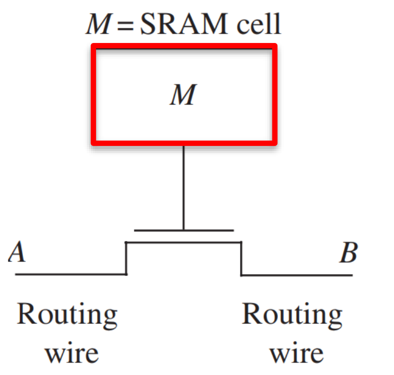
The reconfigurability can be achieved by

- ➢ changing the contents of static RAM cells
- ➢ changing the contents of flash memory cells
- ➢ fusing metal links

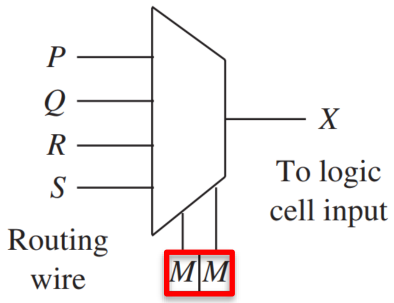
In general, FPGAs use one of the following programming methods:

- StaticRAM programming technology
- EPROM/EEPROM/flash programming technology
- AntiFuse programming technology (反熔丝)

SRAM Programming Technology



(a) Pass transistor connecting two points

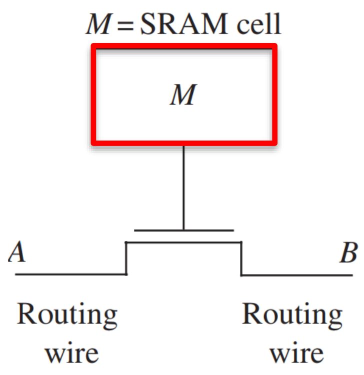


(b) Multiplexer controlled by two memory cells

The SRAM programming technology involves creating reconfigurability by bits stored in static RAM (SRAM) cells

The logic blocks, I/O blocks, and interconnects can be made programmable by using configuration bits stored in SRAM

SRAM Programming Technology



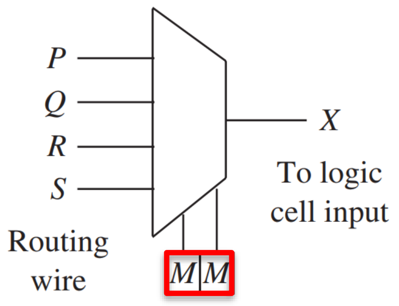
(a) Pass transistor connecting two points

- The programmable interconnect can also be achieved by SRAM
- The key idea is to use pass transistors to create switches and then control them using SRAM content

The SRAM cell is connected to the gate of the pass transistor

- SRAM cell content is **0**: pass transistor is OFF, i.e., no connection exists between points A and B)
- SRAM cell content is **1**: pass transistor is ON, i.e., a closed path can be achieved

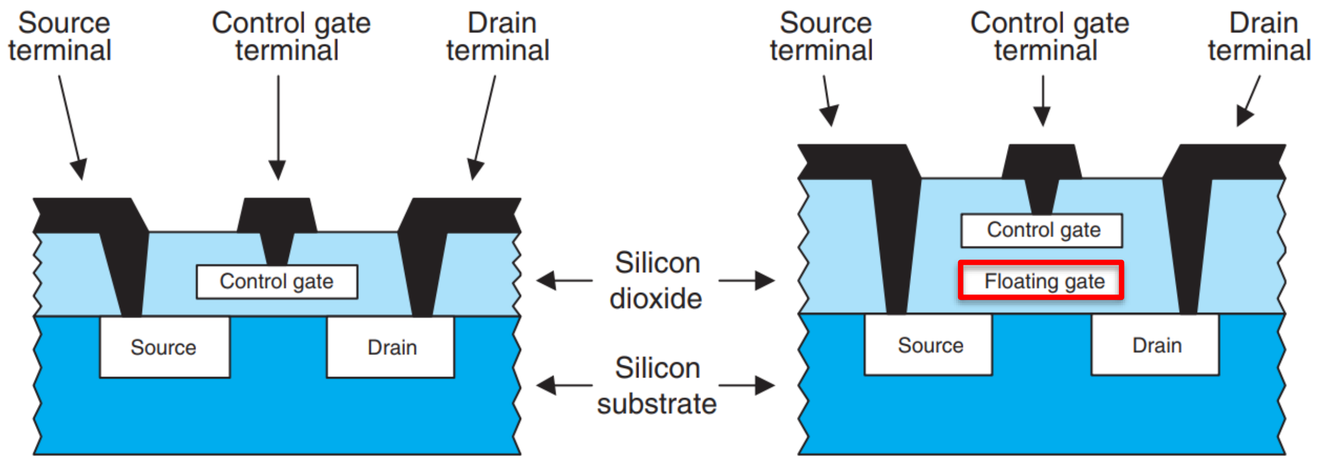
SRAM Programming Technology



(b) Multiplexer controlled by two memory cells

- SRAM bits can be used to construct routing matrices by using multiplexers
- Changing the contents of the SRAM in the arrangement will allow the designer to change what is connected to point X

*EPROM Programming Technology



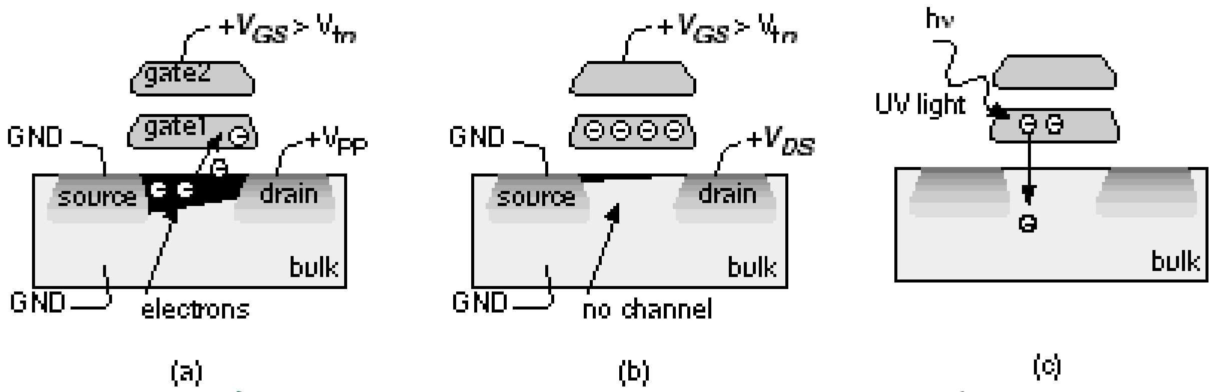
(a) Standard MOS transistor

(b) EPROM transistor

An EPROM transistor has the same basic structure as a standard MOS transistor, but with the addition of a second polysilicon floating gate isolated by layers of oxide

*EPROM Programming Technology

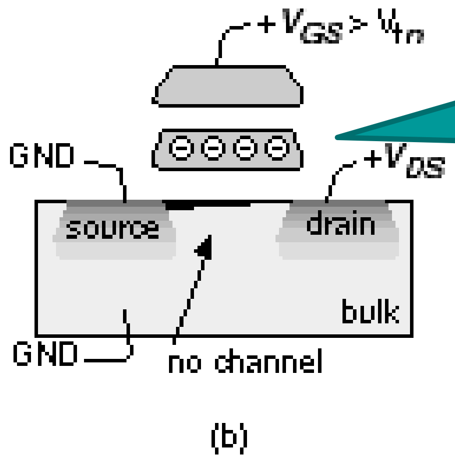
2. Electrons stuck on gate1 raise the threshold voltage so that the transistor is always off for normal operating voltages



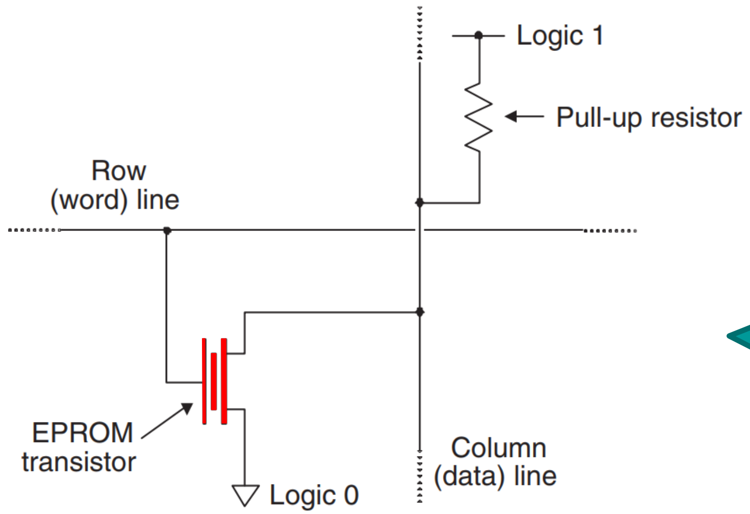
1. With a high (> 12 V) programming voltage, V_{PP} , applied to the drain, electrons gain enough energy to “jump” onto the floating gate (gate1)

3. Ultraviolet light provides enough energy for the electrons stuck on gate1 to “jump” back to the bulk, allowing the transistor to operate normally

*EPROM Programming Technology

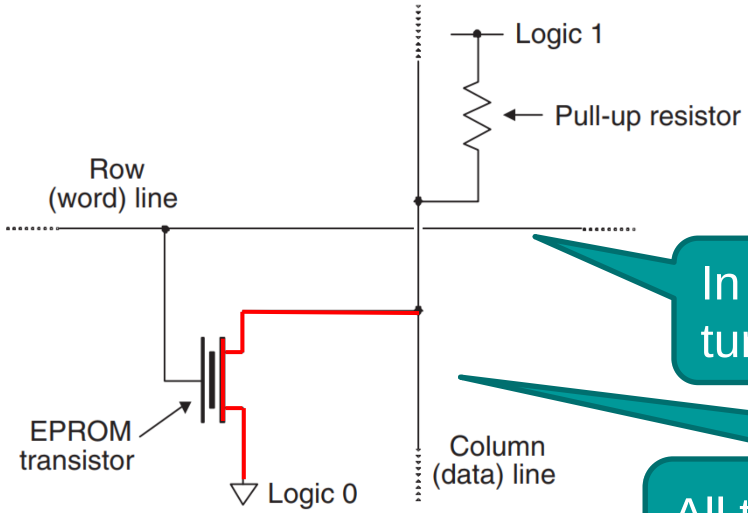


The stored charge on the floating gate inhibits the normal operation of the control gate and, thus, distinguishes those cells that have been programmed from those that have not



Thus, we can use such a transistor to form a memory cell

*EPROM Programming Technology

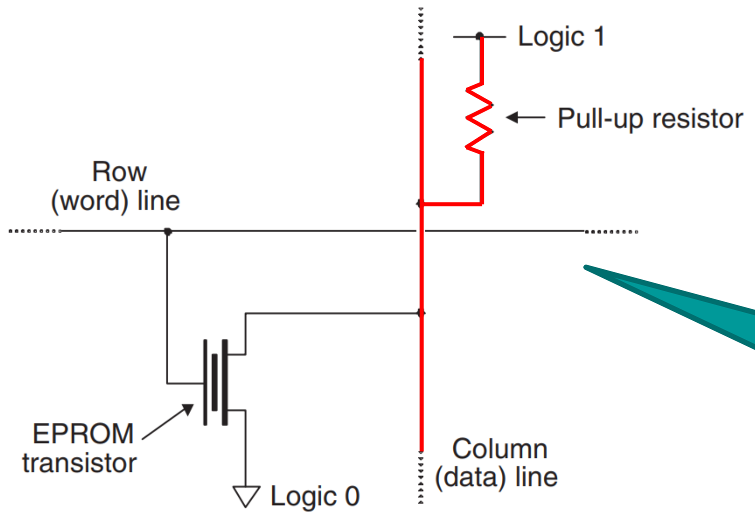


In its unprogrammed state, all of the floating gates in EPROM transistors are uncharged

In this case, placing a row line in its active state will turn on all the transistors connected to that row

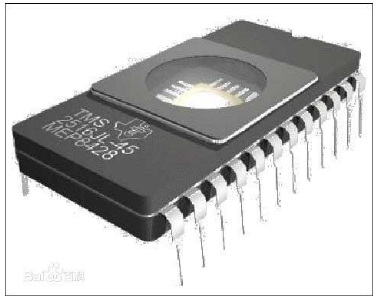
All the column lines are pulled down to **logic 0** via their respective transistors

*EPROM Programming Technology



In order to program the device, engineers can use the inputs to the device to charge the floating gates associated with selected transistors, thereby disabling those transistors

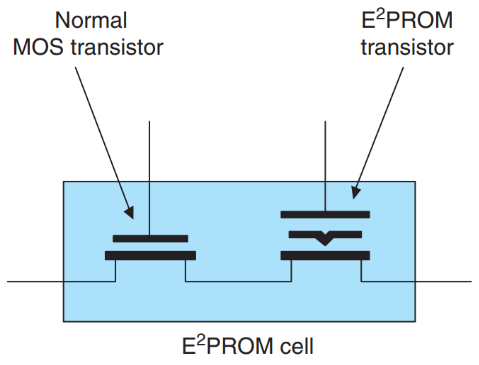
In these cases, the cells will appear to contain **logic 1** values



Main problems with EPROMs:

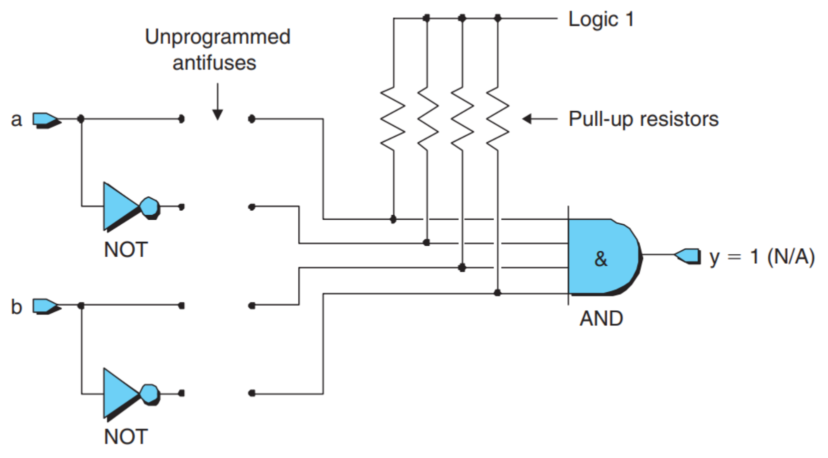
- expensive packages (with quartz windows through which ultraviolet (UV) radiation is used to erase the device)
- the time it takes to erase them, on the order of 20 minutes

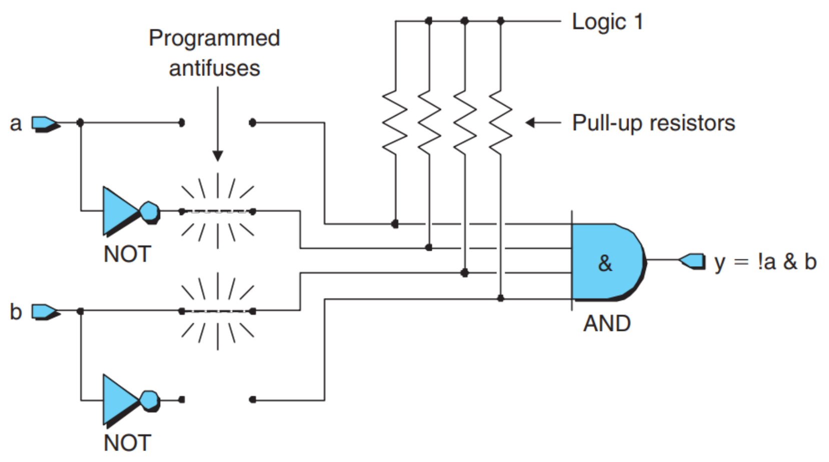
*EEPROM Programming Technology



EEPROM is similar to EPROM, but removal of the gate charge can be done electrically

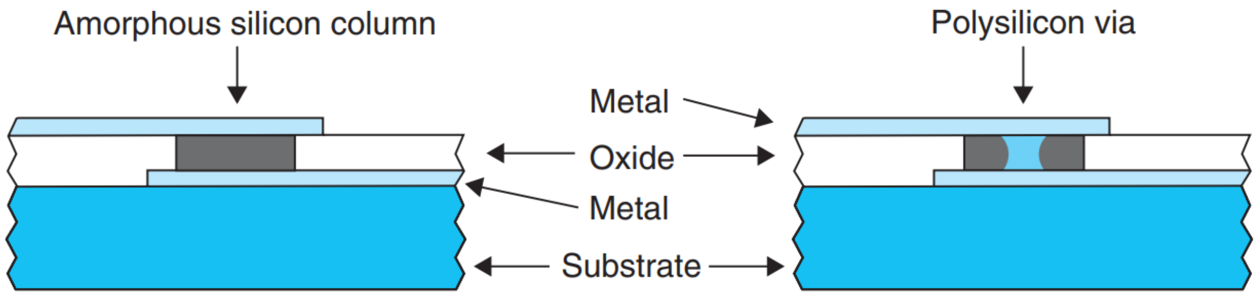
Antifuse Programming Technology





Contrary to fuse wires that blow open when high current passes through them, the “antifuse” programming element changes from high resistance (open) to low resistance (closed) when a high voltage is applied to it

Antifuse Programming Technology



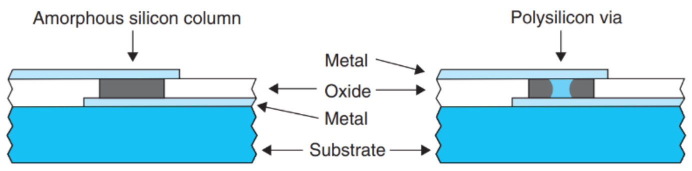
(a) Before programming

(b) After programming

- Antifuses are often built using amorphous silicon(非晶硅) between metal layers
- Antifuses are normally OFF

- Permanently connected links are created when they are programmed
- The process is irreversible (antifuse FPGAs are one-time programmable)

The Antifuse Programming Technology



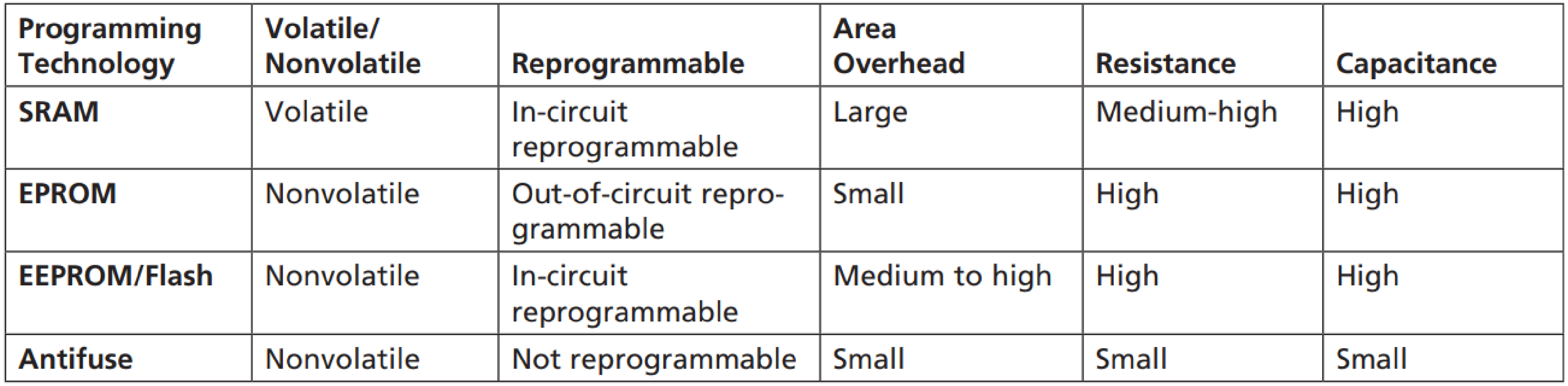
(a) Before programming (b) After programming

- Advantages of antifuse technology
  - The area consumed by the programmable switch is small
  - Antifuse-based connections are faster than SRAM- and EEPROM-based switches

- Disadvantage of antifuse technology
  - not reprogrammable. It is a permanent connection; if an error or design change necessitates reprogramming, a new device is required

Comparison of Programming Technologies

Characteristics of the Major FPGA Programming Technologies



Programming Technology	Volatile/ Nonvolatile	Reprogrammable	Area Overhead	Resistance	Capacitance
SRAM	Volatile	In-circuit reprogrammable	Large	Medium-high	High
EPROM	Nonvolatile	Out-of-circuit repro-grammable	Small	High	High
EEPROM/Flash	Nonvolatile	In-circuit reprogrammable	Medium to high	High	High
Antifuse	Nonvolatile	Not reprogrammable	Small	Small	Small

Comparison of Programming Technologies

Characteristics of the Major FPGA Programming Technologies

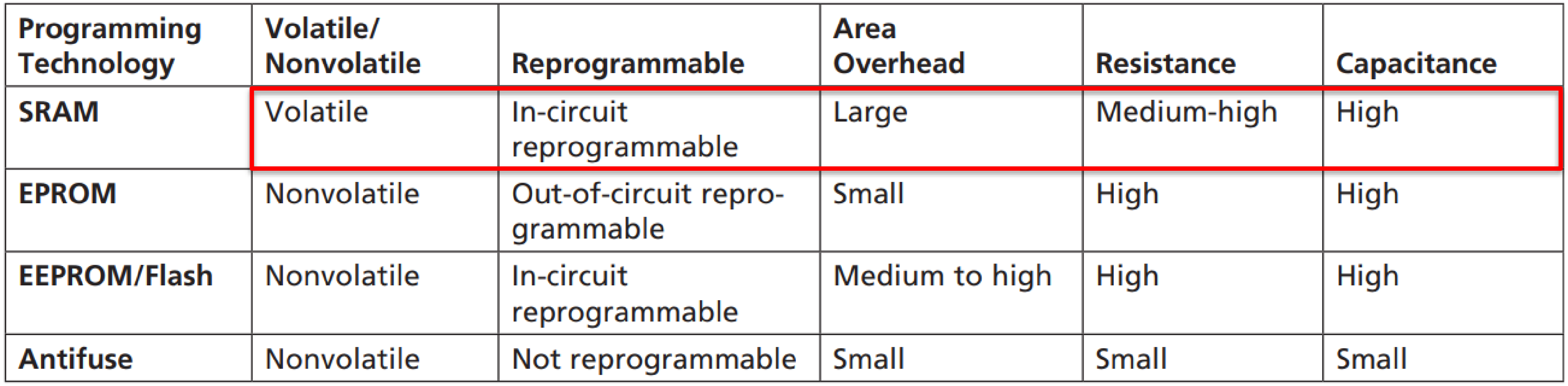
Programming Technology	Volatile/ Nonvolatile	Reprogrammable	Area Overhead	Resistance	Capacitance
SRAM	Volatile	In-circuit reprogrammable	Large	Medium-high	High
EPROM	Nonvolatile	Out-of-circuit repro-grammable	Small	High	High
EEPROM/Flash	Nonvolatile	In-circuit reprogrammable	Medium to high	High	High
Antifuse	Nonvolatile	Not reprogrammable	Small	Small	Small

In-circuit programmability
（在线可重构编程）
an FPGA can be reprogrammed without removing it from the board in which it is used

Only SRAM and EEPROM/Flash programming technologies allow in-circuit programmability

Comparison of Programming Technologies

Characteristics of the Major FPGA Programming Technologies

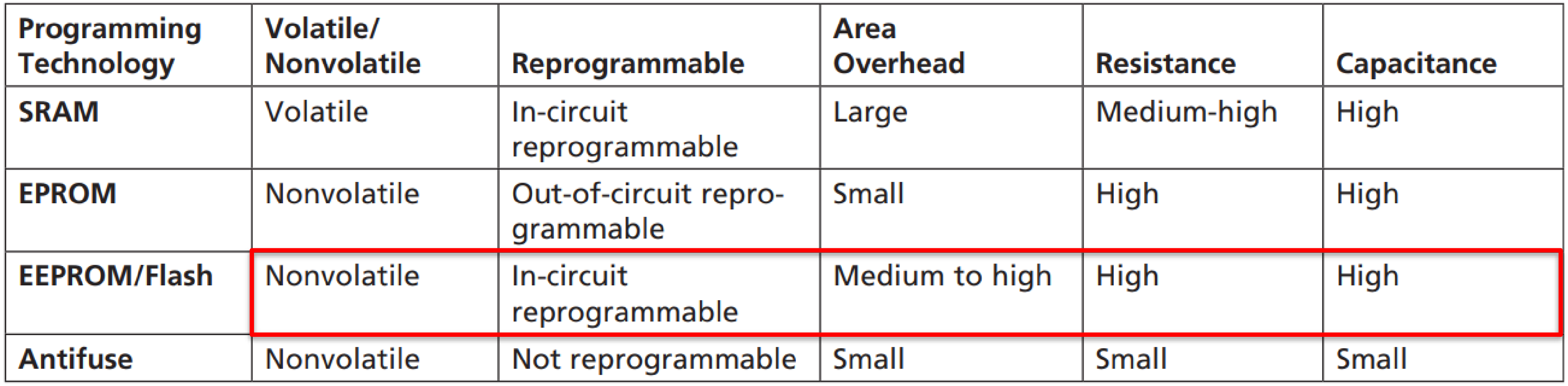


Programming Technology	Volatile/ Nonvolatile	Reprogrammable	Area Overhead	Resistance	Capacitance
SRAM	Volatile	In-circuit reprogrammable	Large	Medium-high	High
EPROM	Nonvolatile	Out-of-circuit repro-grammable	Small	High	High
EEPROM/Flash	Nonvolatile	In-circuit reprogrammable	Medium to high	High	High
Antifuse	Nonvolatile	Not reprogrammable	Small	Small	Small

- SRAM FPGAs have several disadvantages: high area overhead, large delays, volatility, and others
- However, the in-circuit programmability and fast programmability have made them very popular
- SRAM FPGAs are more expensive than other types of FPGAs because each programmable point uses more hardware

Comparison of Programming Technologies

Characteristics of the Major FPGA Programming Technologies



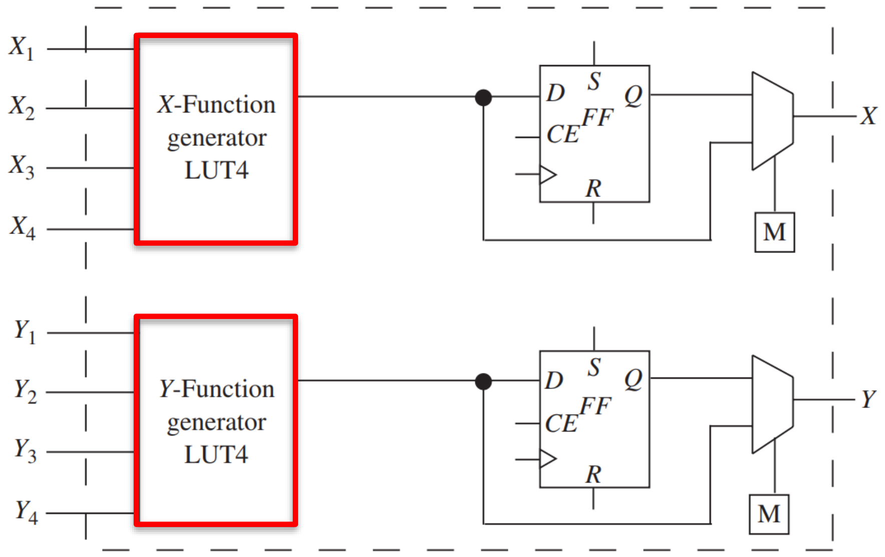
Programming Technology	Volatile/ Nonvolatile	Reprogrammable	Area Overhead	Resistance	Capacitance
SRAM	Volatile	In-circuit reprogrammable	Large	Medium-high	High
EPROM	Nonvolatile	Out-of-circuit repro-grammable	Small	High	High
EEPROM/Flash	Nonvolatile	In-circuit reprogrammable	Medium to high	High	High
Antifuse	Nonvolatile	Not reprogrammable	Small	Small	Small

EEPROM- and **Flash-based FPGAs** are comparable to SRAM FPGAs in many respects; however, they are not as fast as SRAM FPGAs.

	Contents
1	Organization of FPGAs
2	FPGA Programming Technologies
3	**Programmable Logic Block Architectures**
4	Programmable Interconnects
5	Programmable I/O Blocks in FPGAs

3.4.3 Programmable Logic Block Architectures

Look-Up-Table-Based Programmable Logic Blocks



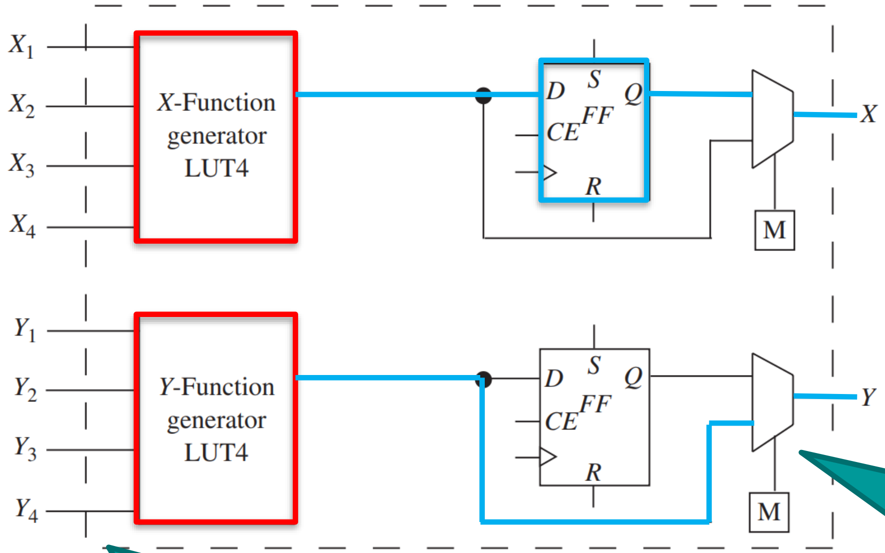
Many look-up-table–based FPGAs use a 4-variable look-up table plus a flip-flop as the basic element and then combine several of them in various topologies

An example structure: two 4-variable look-up tables (**LUT4**) and two flip-flops in this programmable logic block

LUT4 can generate any function of four variables (a **4-variable function generator**)

3.4.3 Programmable Logic Block Architectures

Look-Up-Table-Based Programmable Logic Blocks

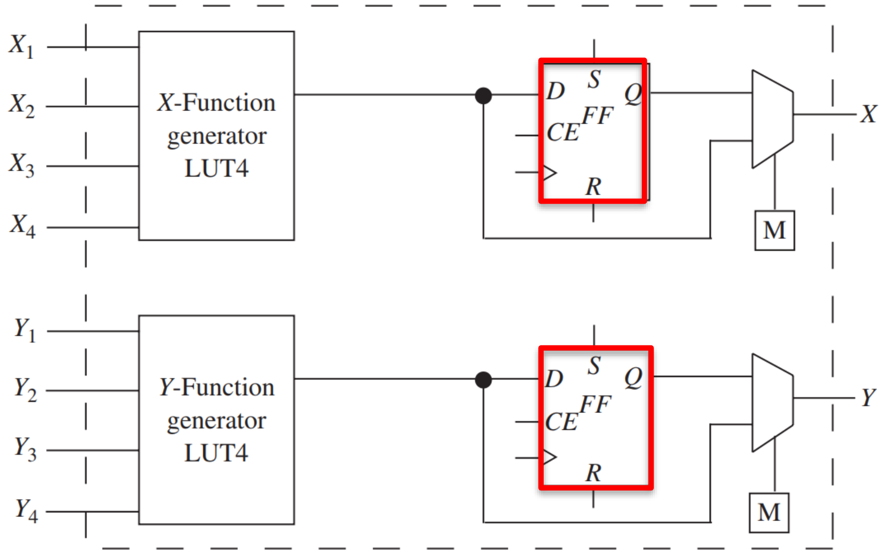


Two LUT4s can generate any two functions of four variables

The functions can be steered to the output of the block (*X* and *Y*) in combinational or latched form

- *X1~4*: inputs to *X*-function generator
- *Y1~4*: inputs to *Y*-function generator

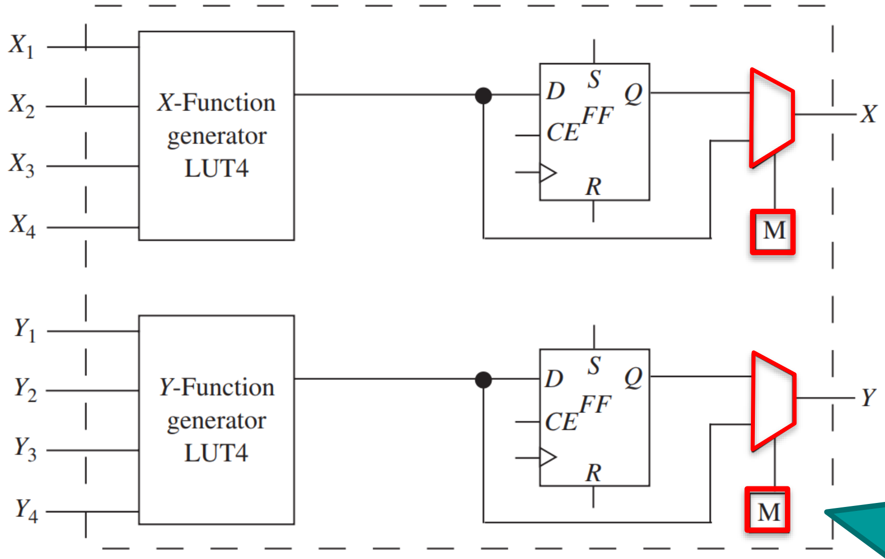
Look-Up-Table-Based Programmable Logic Blocks



There are two D flip-flops in the logic block

D flip-flops are versatile (they have clock enable, direct set, and direct reset inputs)

Look-Up-Table–Based Programmable Logic Blocks

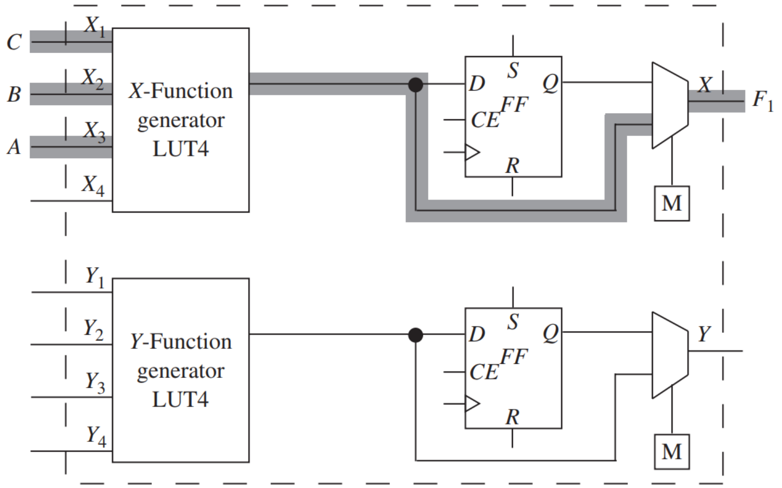


A multiplexer selects between the combinatorial output and the latched version of the output

The little box with "**M**" in it indicates a memory cell that is required to provide appropriate select signals to select between the latched and unlatched form of the function

Look-Up-Table–Based Programmable Logic Blocks

Highlighting Paths for Function F_1



$F_1 = A'B'C + A'BC' + AB$

- This is a 3-variable function
- A 4-input LUT is more than sufficient to implement the function

- The path highlighted assumes that the *X*-function generator (top LUT) is used
- Since function F_1 uses only three variables, input *X4* is not used
- A truth table can be constructed to represent the function, and the LUT contents can be derived

Look-Up-Table–Based Programmable Logic Blocks

X4	X3 (A)	X2 (B)	X1 (C)	F1
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
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	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

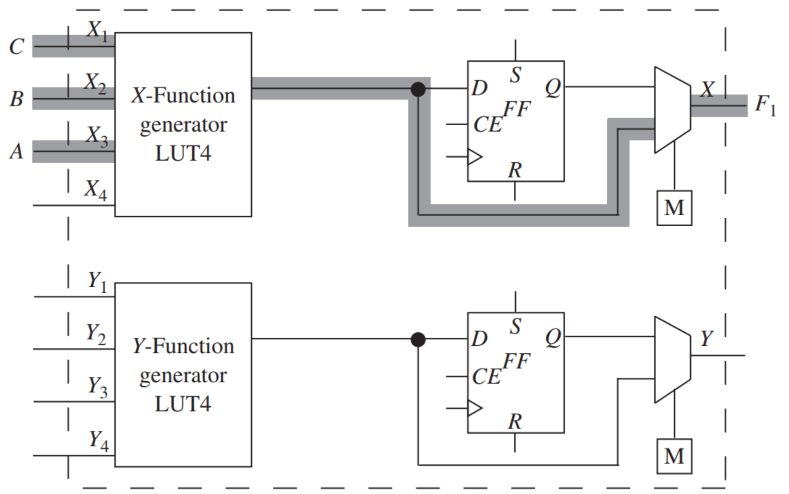
$F_1 = A'B'C + A'BC' + AB$

- Assume that *X1* is the LSB and *X4* is the MSB
- The LUT contents to implement function F1 will be 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1

- The first 8 bits in LUT reflect the truth table outputs when the function is represented in a truth table form
- Since input *X4* is not grounded, the first 8 bits are repeated to take care of the possibility that the *X4* input might stay at a logic 1 when it is unused

Look-Up-Table–Based Programmable Logic Blocks

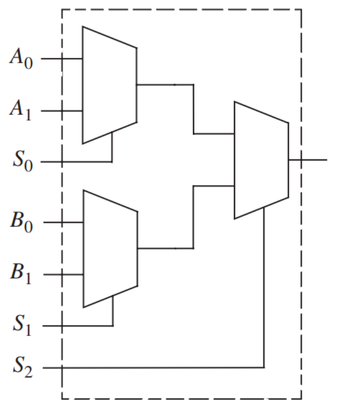
Highlighting Paths for Function F_1



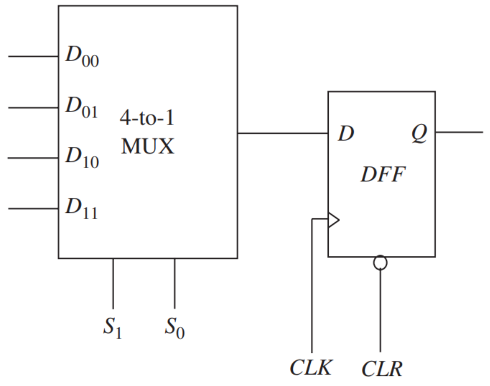
- Since the functions are stored in LUT form, the number of terms in the function is not important. Common minimizations to reduce the number of terms are not relevant
- The number of variables is what is important

Logic Blocks Based on Multiplexers and Gates

Multiplexer-Based Logic Blocks in FPGAs



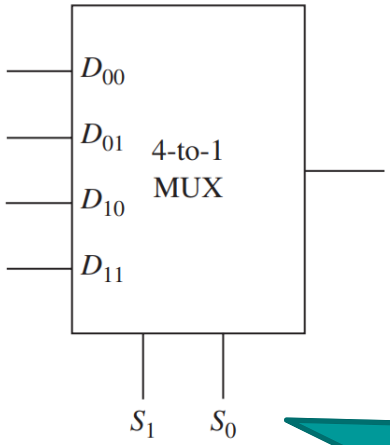
(a) A logic block with three 2-to-1 MUXes



(b) A logic block with a 4-to-1 MUX and a flip-flop

- Some FPGAs use multiplexers as the basic building block
- Any combinational function can be implemented using multiplexers alone

Logic Blocks Based on Multiplexers and Gates



$F_1 = A'B'C + A'BC' + AB$

use an FPGA with programmable logic blocks consisting of 4-to-1 multiplexers

- Two of the 3-input variables can be connected to the multiplexer select lines
- Appropriate signals should be provided to the multiplexer data input lines in order to realize the function

Logic Blocks Based on Multiplexers and Gates

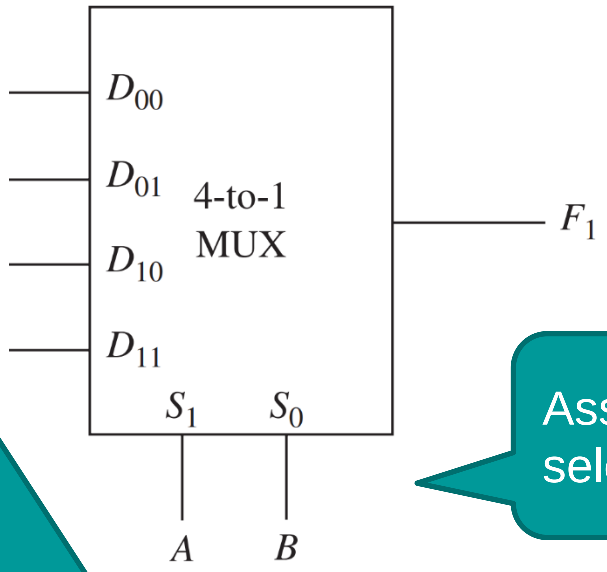
$F_1 = A'B'C + A'BC' + AB$

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

To derive multiplexer data inputs, we will first construct a truth table of the function

Logic Blocks Based on Multiplexers and Gates

Multiplexer Implementing Function *F*1



Assume that *A* and *B* are connected to the select inputs of the multiplexer

Next, we will derive values of inputs to provide to the multiplexer input lines in terms of the third variable in the function

Logic Blocks Based on Multiplexers and Gates

$F_1 = A'B'C + A'BC' + AB$

A	*B*	*C*	*F*	*MUX input in terms of {0,1,C,C'}*
0	0	0	0	} *C*
0	0	1	1	
0	1	0	1	} *C'*
0	1	1	0	
1	0	0	0	} 0
1	0	1	0	
1	1	0	1	} 1
1	1	1	1	

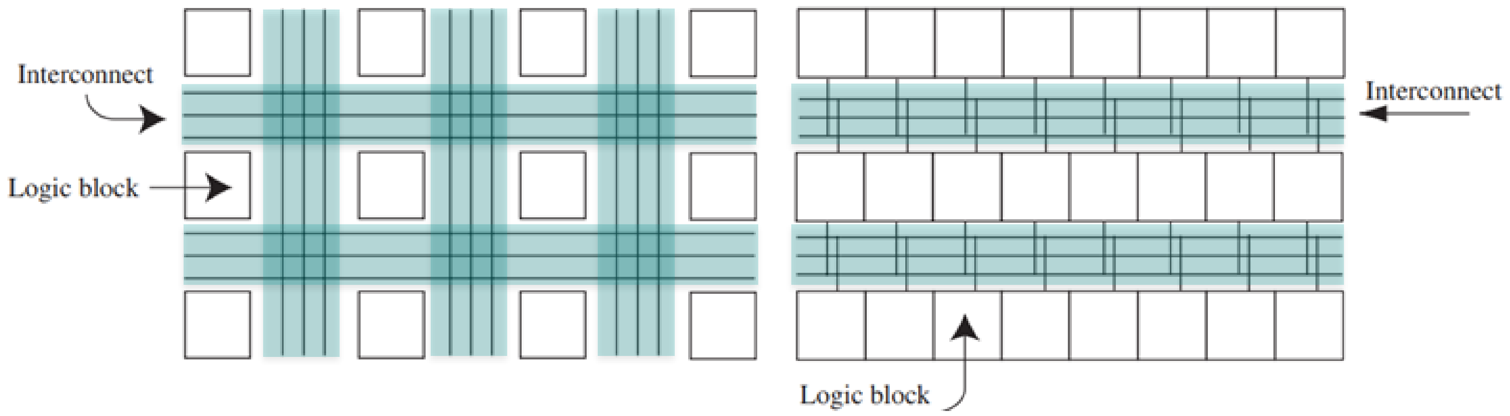
The third variable is *C*, and by providing one of the four values {*C*, *C'*, 0, 1}, any 3-variable function can be expressed

- **Rows 1, 2**: *F* = *C* when *AB* = 00
- **Rows 3, 4**: *F* = *C'* when *AB* = 01
- **Rows 5, 6**: F = 0 when *AB* = 10 (irrespective of the value of *C)*
- **Rows 7, 8**: F = 1 when *AB* = 11

- The last column in the truth table presents the required multiplexer inputs
- Hence, one 4-to-1 multiplexer with the connections can implement function *F1*

	Contents
1	Organization of FPGAs
2	FPGA Programming Technologies
3	Programmable Logic Block Architectures
4	**Programmable Interconnects**
5	Programmable I/O Blocks in FPGAs

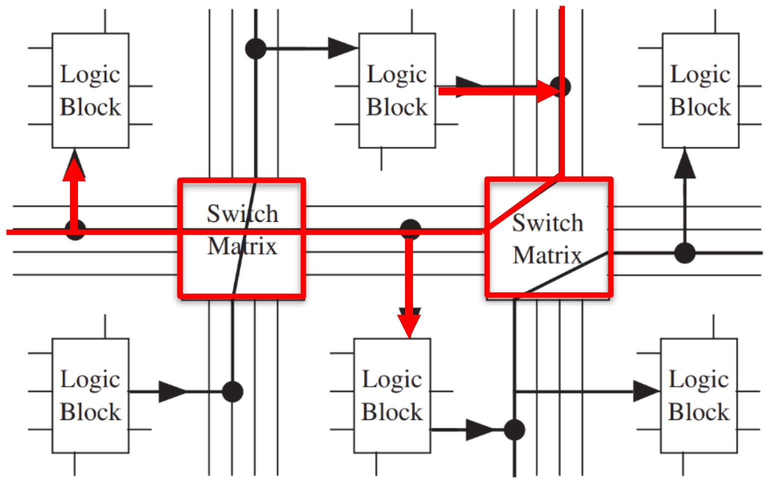
3.4.4 Programmable Interconnects



A key element of an FPGA is the general-purpose programmable interconnect interspersed between the programmable logic blocks

Interconnects in Symmetric Array FPGAs

Routing Matrix for General-Purpose Interconnection in an FPGA



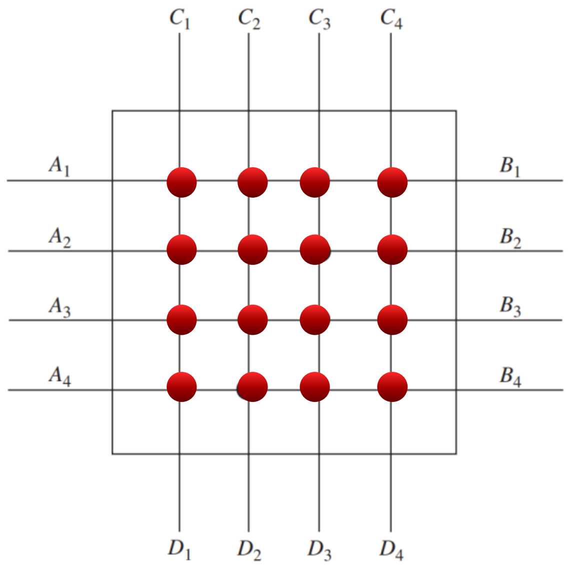
(a) Logic blocks interconnected with switch matrices

Many FPGAs use switch matrices that provide interconnections between routing wires connected to the switch matrix

Interconnecting logic blocks in an FPGA using switch matrices

Interconnects in Symmetric Array FPGAs

Internals of a switch matrix



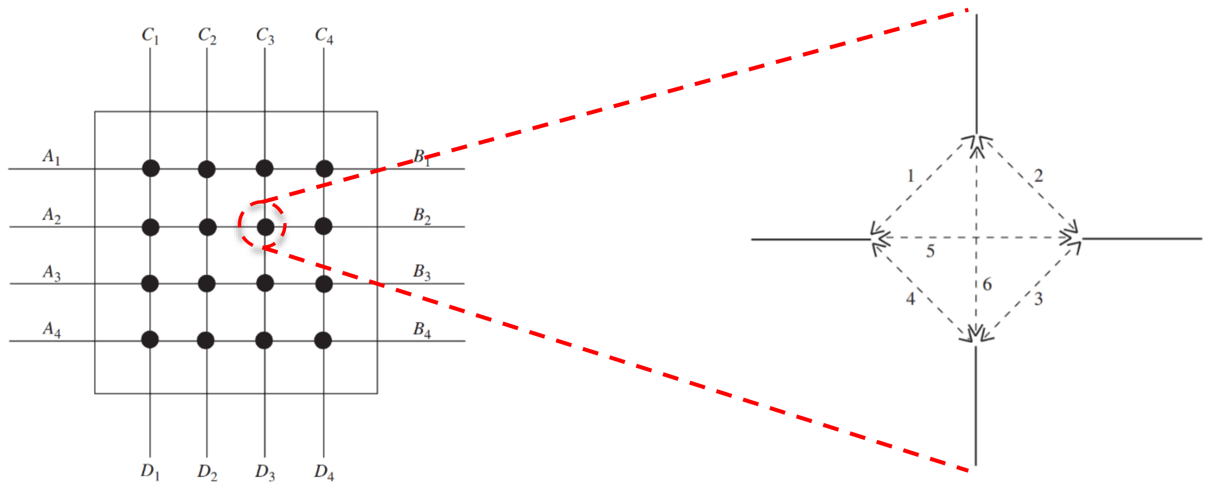
A typical switch matrix, where there is a switch at each intersection (i.e., wherever the lines cross)

- A switch matrix that supports every possible connection from every wire to every other wire is very expensive
- The connectivity is often limited to some subset of a full crossbar connection; moreover, not all connections might be possible simultaneously

Interconnects in Symmetric Array FPGAs

Internals of a switch matrix

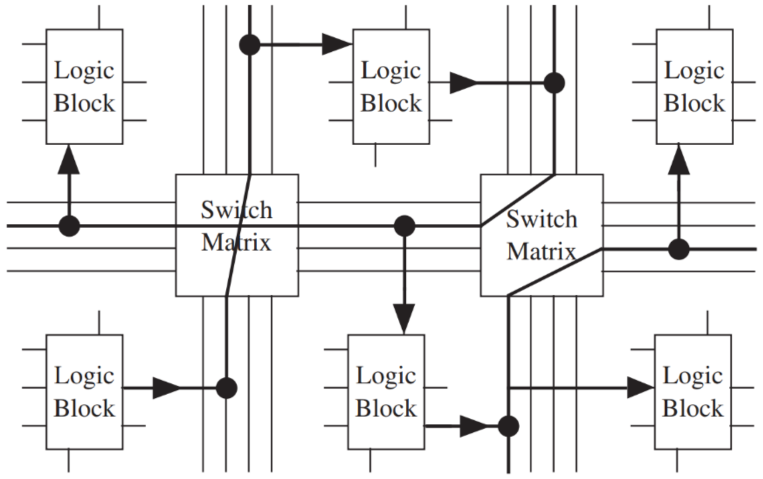
A 6-way switch



- Each wire from a side of the switch can be routed to other wires using some combination of the switches
- In order to support this type of connection, each cross point in the switch matrix must support six possible interconnections

Interconnects in Symmetric Array FPGAs

Internals of a switch matrix



(a) Logic blocks interconnected with switch matrices

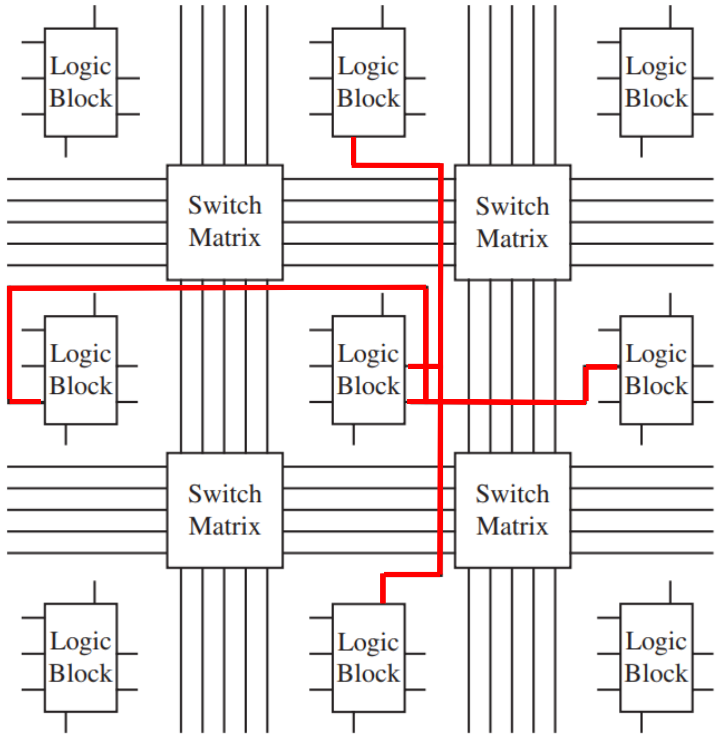
Depending on the programming technology, SRAM cells, flash memory cells, or antifuse connections control the configuration of the switches

The switch matrices interspersed between the logic blocks in an FPGA allow general-purpose interconnectivity between arbitrary points in the chip

- However, the switch matrices are expensive in area and time (delay). If a signal passes through several of these switch matrices, it could contribute to a significant signal delay
- The delays are variable and unpredictable depending on the number of the switch matrices involved in each signal

Interconnects in Symmetric Array FPGAs

Direct Interconnects between Neighboring Logic Blocks



(a) Direct interconnects to four near neighbors

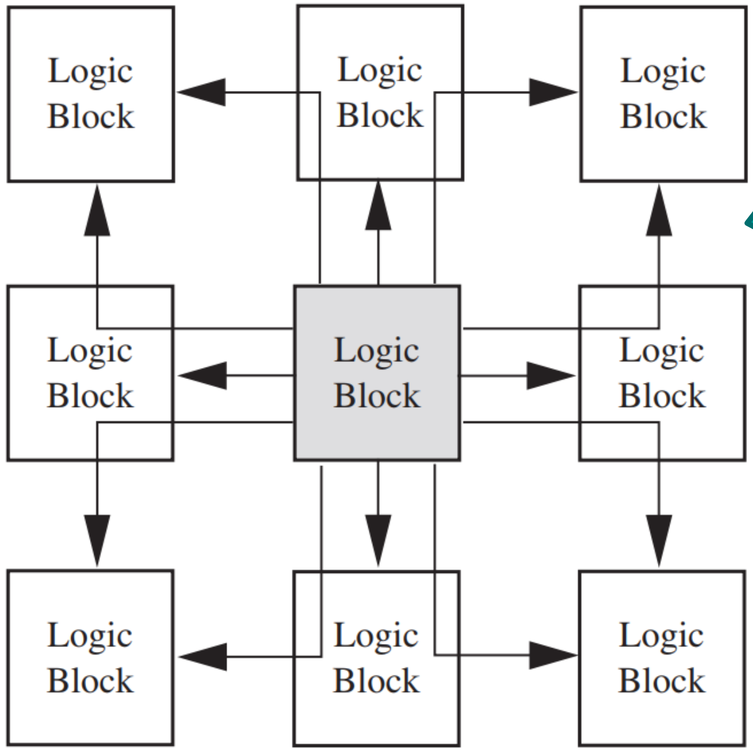
Many FPGAs provide special connections between adjacent logic blocks

These interconnects are fast because they do not go through the routing matrix

Many FPGAs provide direct interconnections to the four nearest neighbors

Interconnects in Symmetric Array FPGAs

Direct Interconnects between Neighboring CLBs



(b) Special connections to 8 neighbors

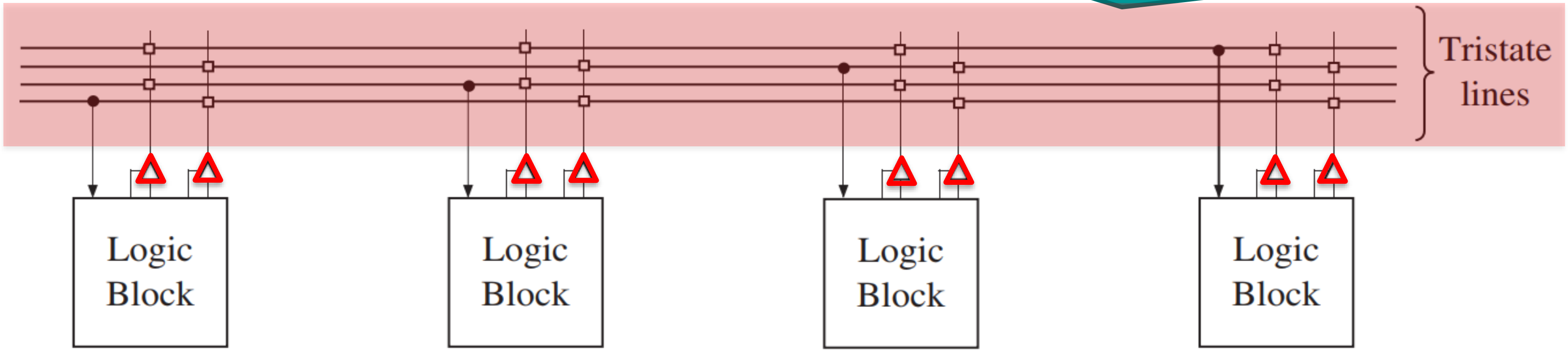
- In some cases, there are special interconnections to 8 neighboring blocks, including the diagonally located logic blocks
- The direct interconnections do not go through the switch matrix but are implemented with dedicated switches resulting in smaller delays

Interconnects in Symmetric Array FPGAs

Global Lines

For purposes such as high fan-out and low-skew clock distribution, most FPGAs provide routing lines that span the entire width/height of device

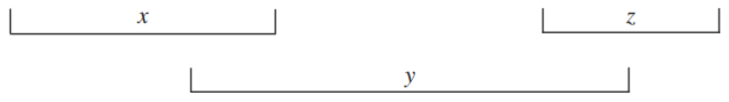
A limited number (two or four) of such global lines are provided by many FPGAs in the horizontal and vertical directions



Logic blocks often have tristate buffers to connect to the global lines

Interconnects in Row-Based FPGAs

Typical Routing Resources in a Row-Based FPGA



(a) Example nets

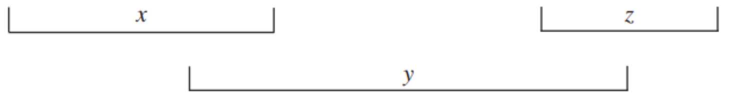
The interconnects in row-based channeled architecture can be classified into two categories:

- non-segmented routing (非分段布线)
- segmented routing （分段布线）

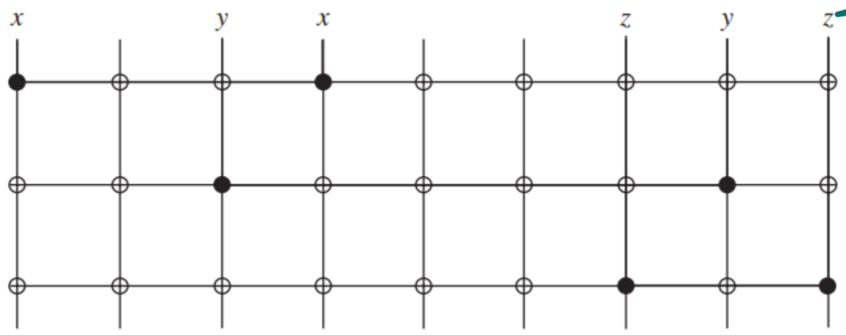
In order to understand different types of channel routing, consider the connections *x*, *y*, and *z*

Interconnects in Row-Based FPGAs

Nonsegmented channel routing of example nets



(a) Example nets

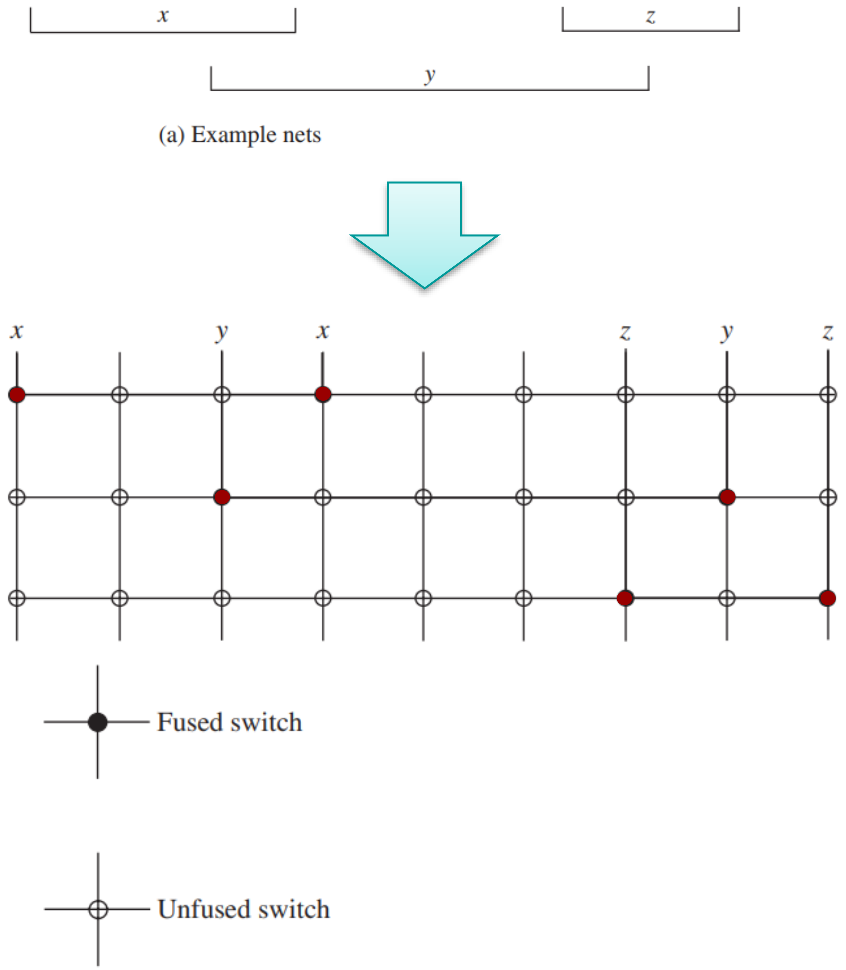


- There are three horizontal rows or tracks
- There are several vertical wires and switches at the crosspoints

- The switches technically can use any programming technology (SRAM, EPROM, or antifuse)
- Desired connectivity is obtained by programming the appropriate switches

Interconnects in Row-Based FPGAs

Nonsegmented channel routing of example nets

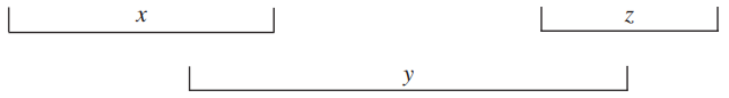


- Connectivity between the points marked "x" is obtained by the two switches at row 1, columns 1 and 4
- Typically this is called net "x". Net 'x' simply means a wire that is named "x"

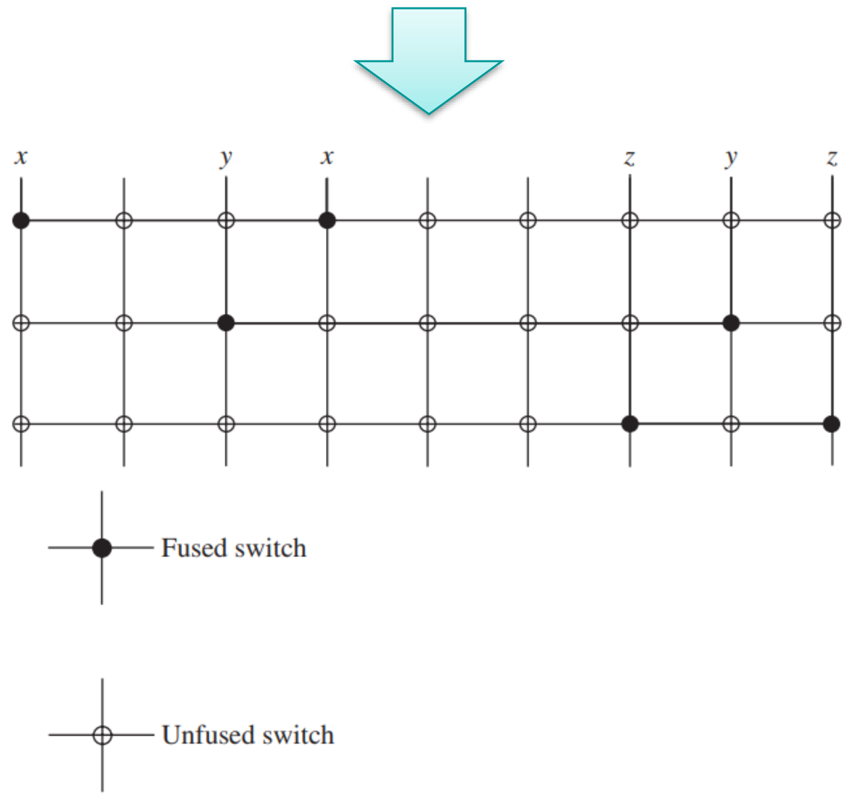
The connectivity for net 'y' is obtained by programming the switches at row 2, columns 2 and 7

Interconnects in Row-Based FPGAs

Nonsegmented channel routing of example nets



(a) Example nets

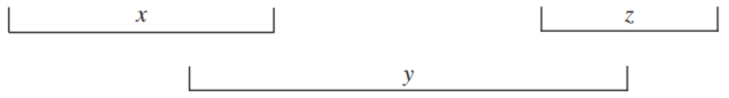


- It may be noticed that row 1 cannot be used for any other connections other than net “x”
- Similarly row 2 is exclusively used for net “y”

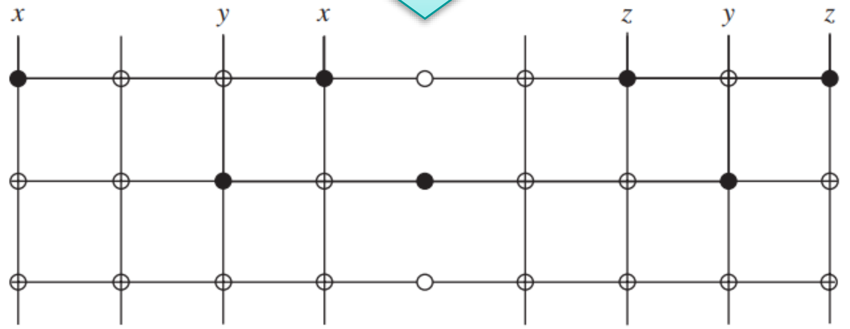
- A problem with this type of interconnect resource is that an entire row is used even for a short net
- The area overhead of this type of routing is very high

Interconnects in Row-Based FPGAs

Segmented channel routing of example nets



(a) Example nets

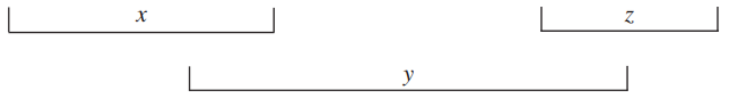


In order to reduce the area overhead associated with using full-length tracks for each net, we can use **segmented tracks**

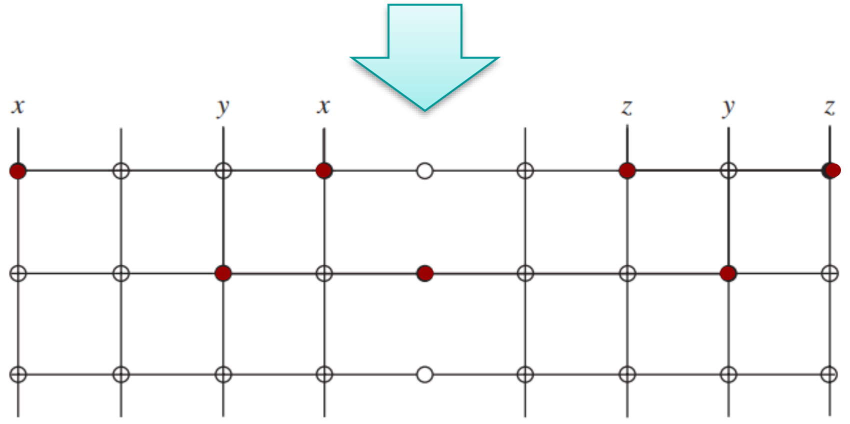
- Instead of being full length, a track is divided into segments
- If a track in row 1 is segmented into two segments, one could use the same track for one more net

Interconnects in Row-Based FPGAs

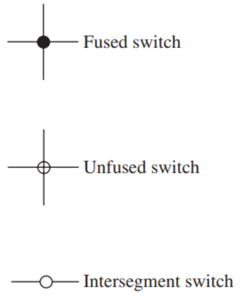
Segmented channel routing of example nets



(a) Example nets



- Nets “x” and “z” can both be routed on row 1
- More nets can be routed using the same number of tracks

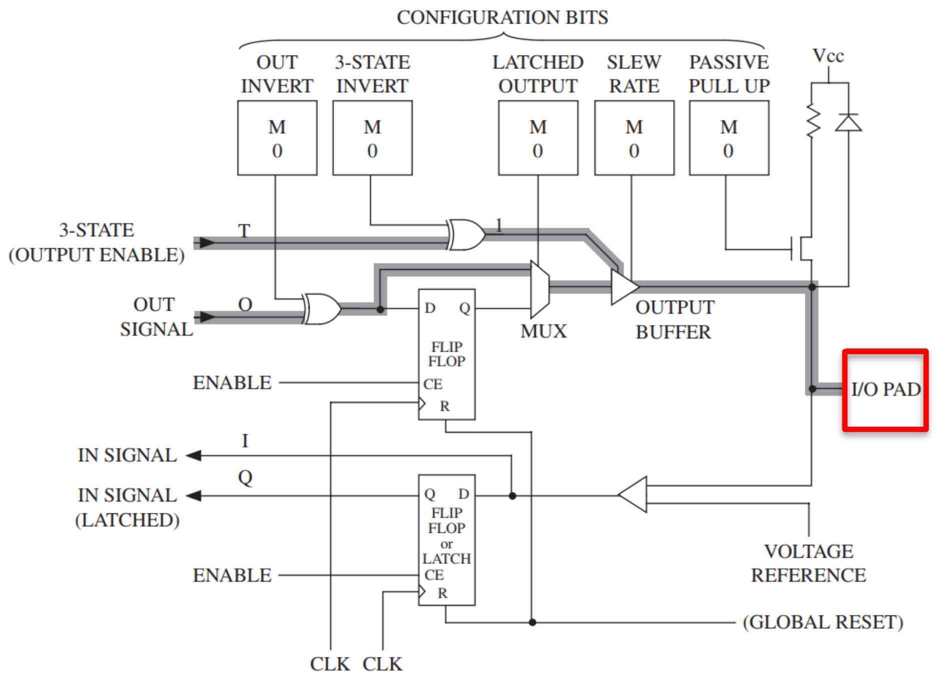


- However, when long nets are desired, intersegment switches must be used to join the segments. These switches introduce more resistance and capacitance into the net
- However, the overall routing resource area will reduce with segmented routing

	Contents
1	Organization of FPGAs
2	FPGA Programming Technologies
3	Programmable Logic Block Architectures
4	Programmable Interconnects
5	**Programmable I/O Blocks in FPGAs**

3.4.5 Programmable I/O Blocks in FPGAs

Programmable I/O Block for an FPGA

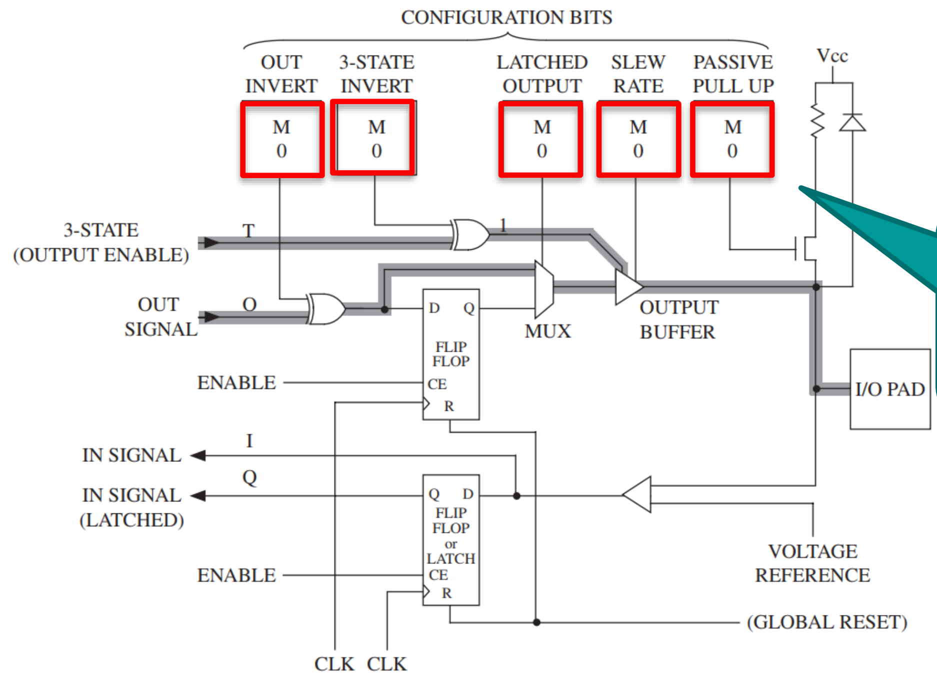


- I/O pads on an FPGA are connected to programmable input/output blocks
- They facilitate connecting the signals from FPGA logic blocks to the external world in desired forms and formats

I/O blocks on modern FPGAs allow use of the pin as input and/or output, in direct (combinational) or latched forms, in tristate true or inverted forms, and with a variety of I/O standards

3.4.5 Programmable I/O Blocks in FPGAs

Programmable I/O Block for an FPGA

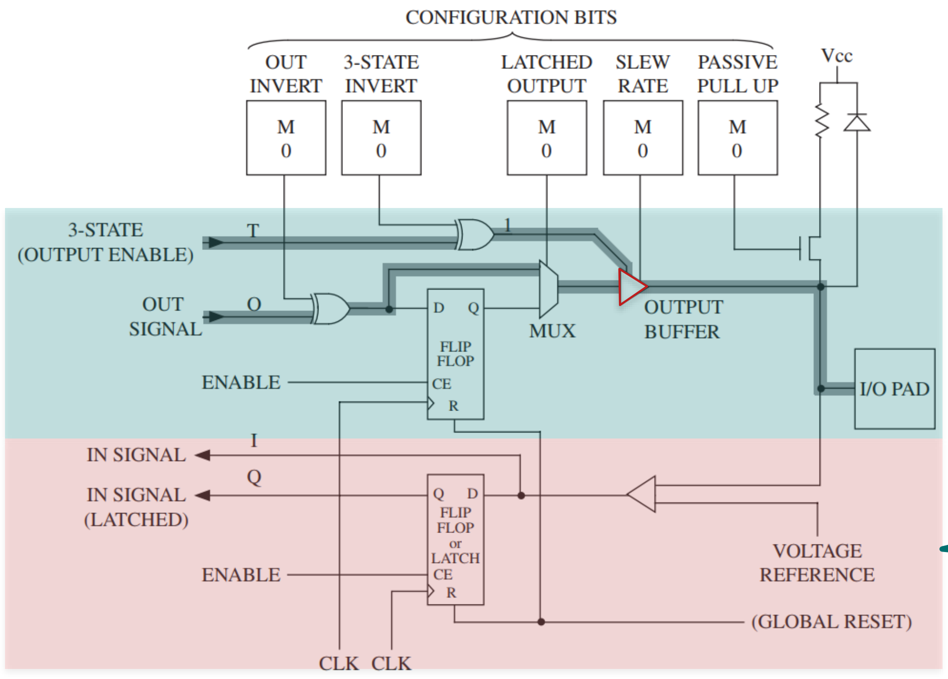


An example configurable input/output block (I/OB)

- Each I/OB has a number of I/O options
- They can be selected by configuration memory cells ("M" boxes)

3.4.5 Programmable I/O Blocks in FPGAs

Programmable I/O Block for an FPGA

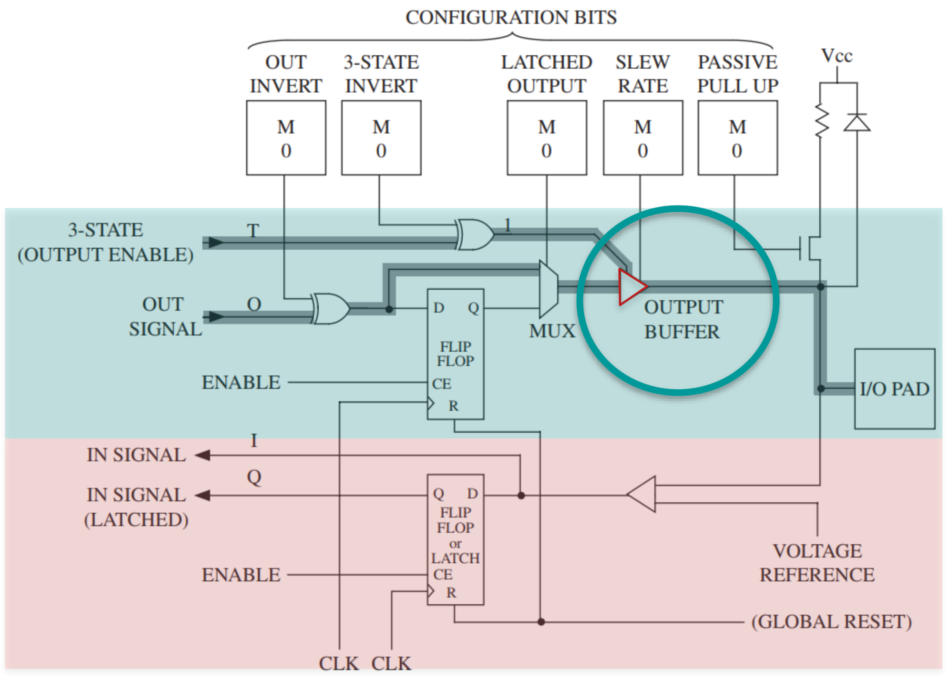


The I/O pad can be programmed to be an output or

or an input

3.4.5 Programmable I/O Blocks in FPGAs

Programmable I/O Block for an FPGA

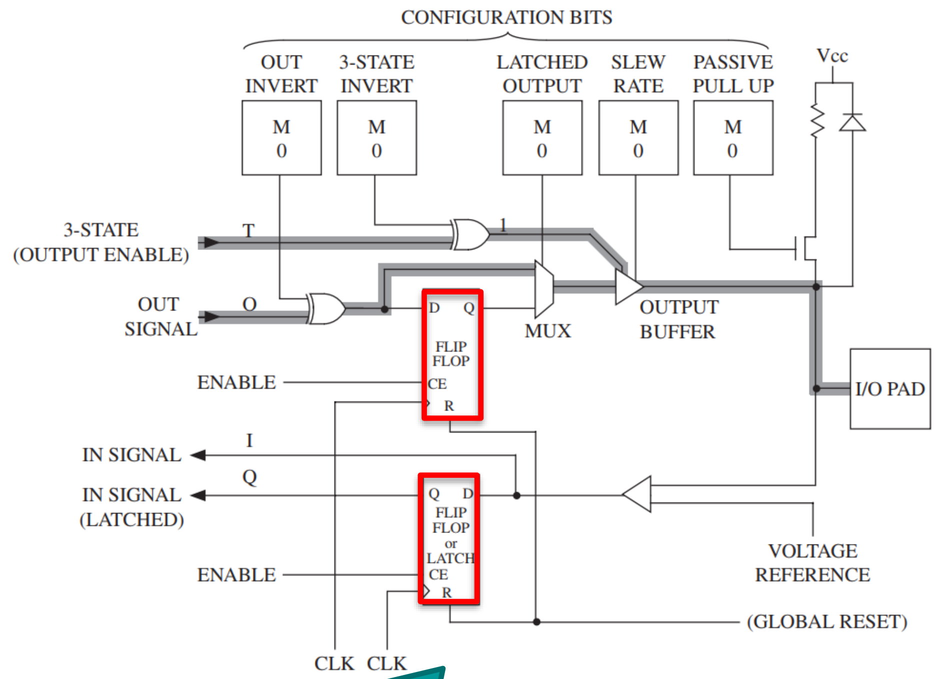


To use the cell as an output, the tristate buffer must be enabled

To use the cell as an input, the tristate control must be set to place the tristate buffer, which drives the output pin, in the high-impedance state

3.4.5 Programmable I/O Blocks in FPGAs

Programmable I/O Block for an FPGA



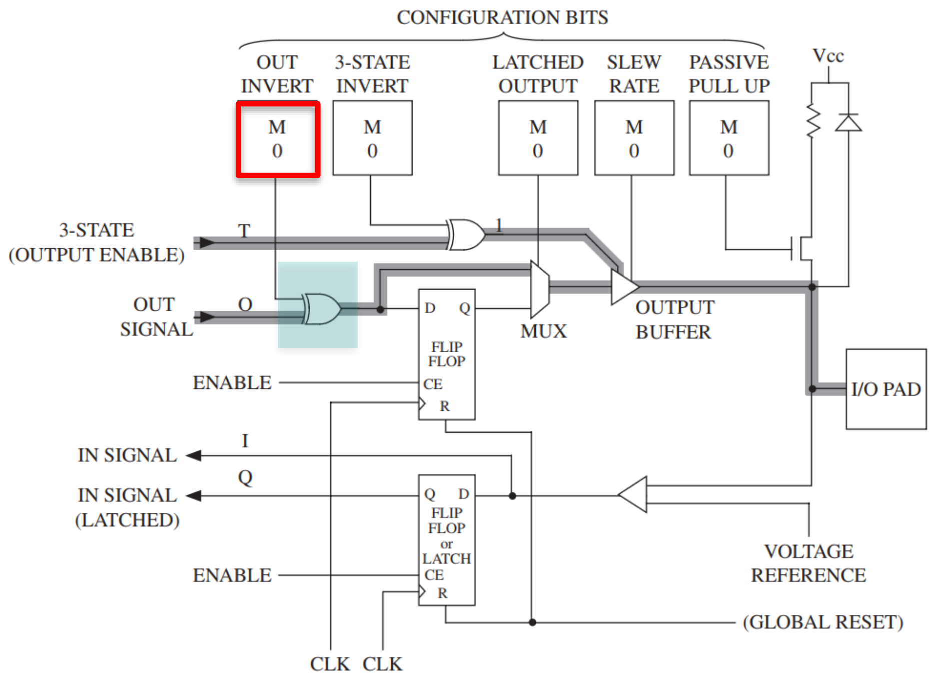
- ➢ Flip-flops are provided so that input and output values can be stored within the I/O block
- ➢ The flip-flops are bypassed when direct input or output is desired

The input flip-flop on many FPGAs can be programmed to act
- as an edge-triggered D flip-flop or
- as a transparent latch

Even if the I/O pin is not used, the I/O flip-flops can still be used to store data

3.4.5 Programmable I/O Blocks in FPGAs

Programmable I/O Block for an FPGA



O XOR 1 = O’
O XOR 0 = O

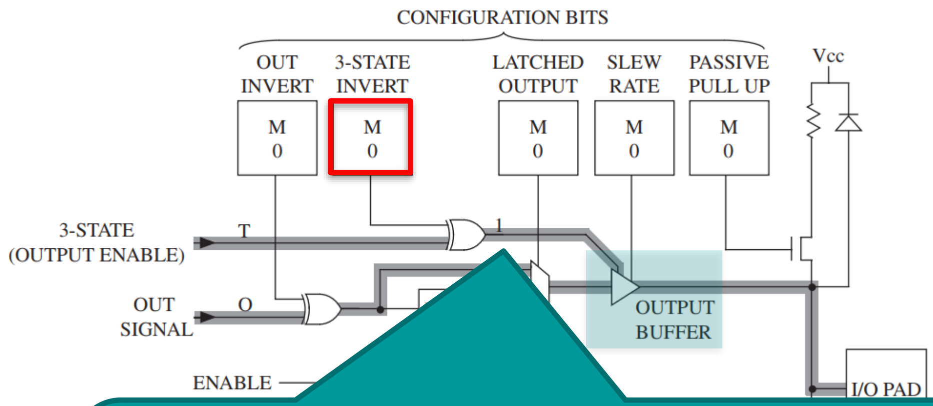
Configuration memory cells (“M”) allow control of various aspects associated with the I/O block

An output signal can be inverted by the I/O block if desired

- The output signal goes through an XOR gate
- OUT-INVERT cell determines whether the output signal is complemented or not

3.4.5 Programmable I/O Blocks in FPGAs

Programmable I/O Block for an FPGA



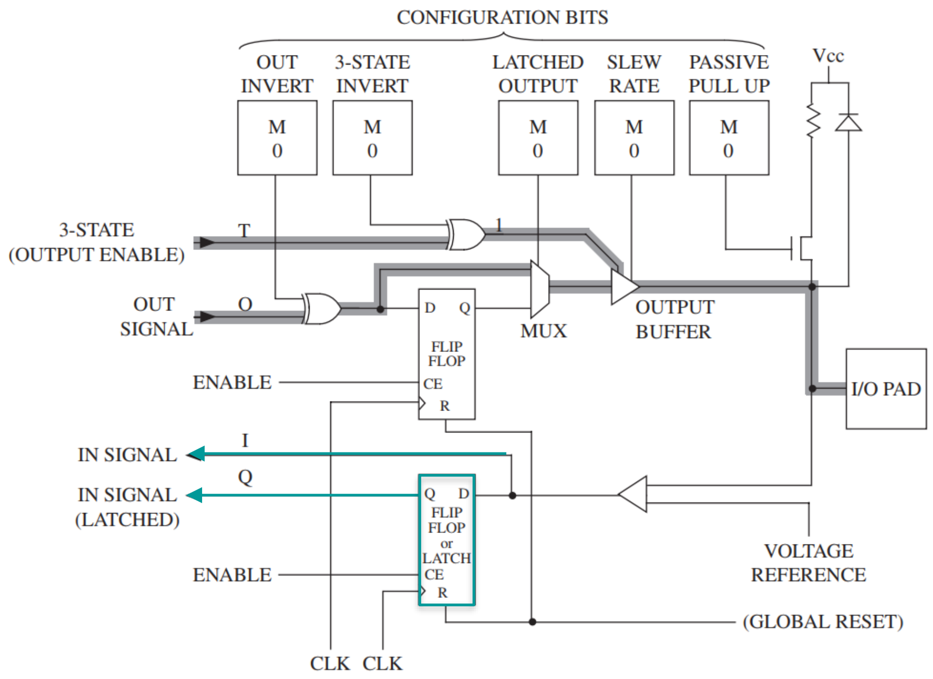
3-STATE INVERT configuration bit allows one to create an active high or active low tristate control signal

- If 3-STATE signal is 1 and 3-STATE INVERT bit is 0 or
- if 3-STATE signal is 0 and 3-STATE INVERT bit is 1,

the buffer drives the output signal to the I/O pad

3.4.5 Programmable I/O Blocks in FPGAs

Programmable I/O Block for an FPGA



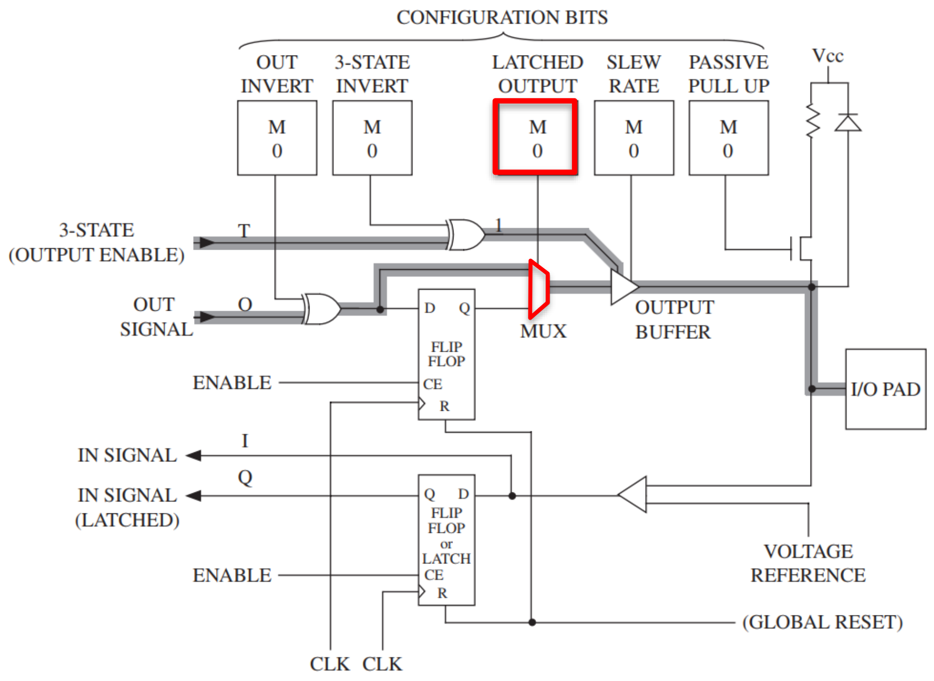
When the I/O pad is used as an input, the output buffer must be in the high-impedance state

An external signal coming into the I/O pad goes through a buffer and then to the input of a D flip-flop

- The buffer output provides a DIRECT IN signal to the logic array
- The input signal can be stored in D flip-flop, which provides LATCHED IN signal

3.4.5 Programmable I/O Blocks in FPGAs

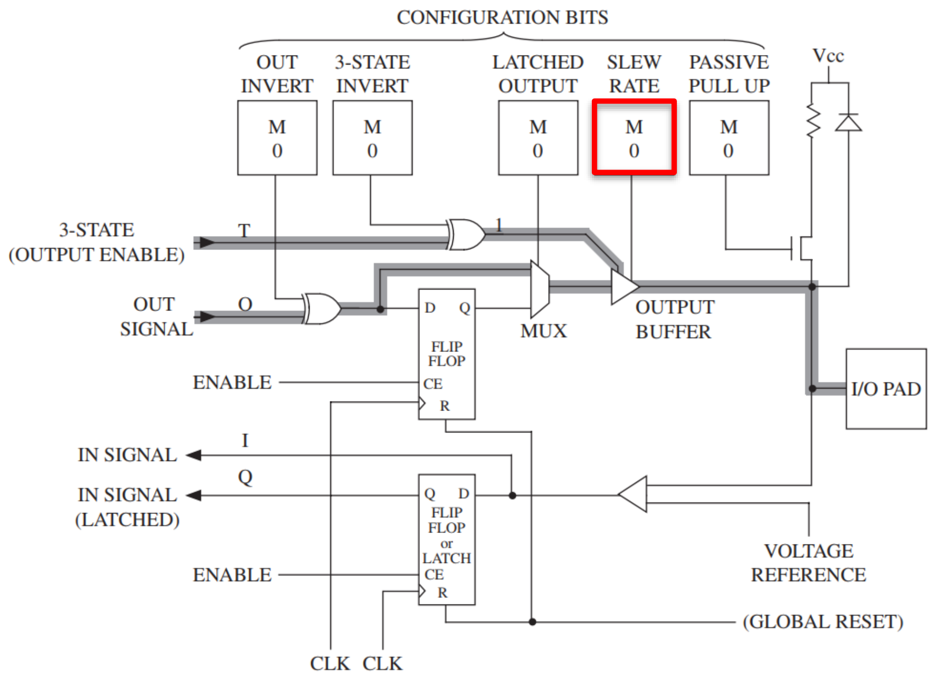
Programmable I/O Block for an FPGA



- LATCHED OUTPUT configuration bit allows one to provide the output in latched or combinational form
- Depending on how the LATCHED OUTPUT bit is programmed, either the OUT signal or the flip-flop output goes to the output buffer

3.4.5 Programmable I/O Blocks in FPGAs

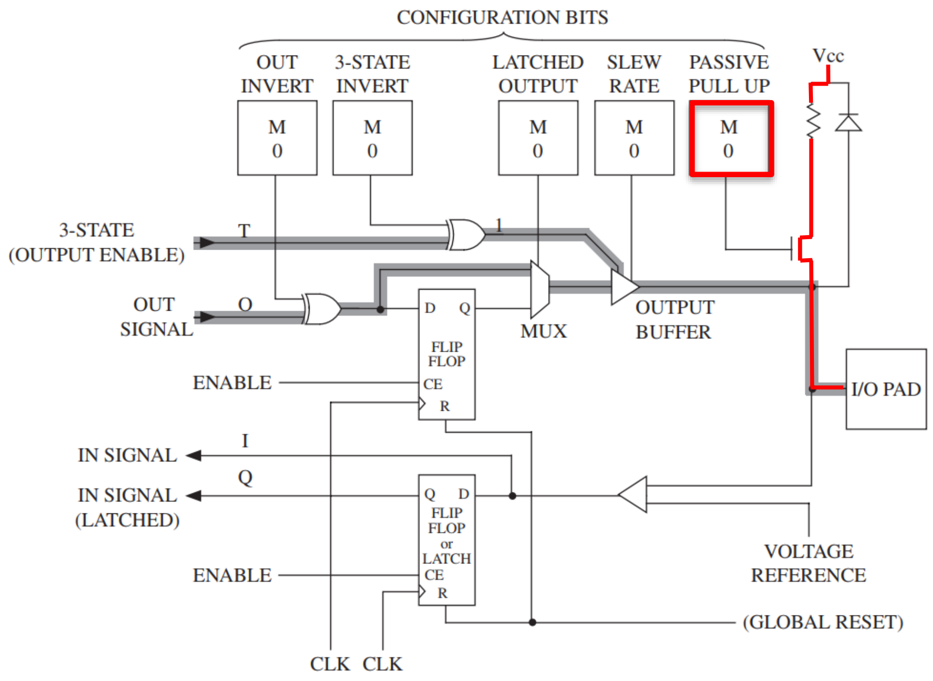
Programmable I/O Block for an FPGA



SLEW RATE bit controls the rate at which the output signal can change

3.4.5 Programmable I/O Blocks in FPGAs

Programmable I/O Block for an FPGA



- When PASSIVE PULL-UP bit is set, a pull-up resistor（上拉电阻) is connected to the I/O pad
- This internal pull-up resistor can be used to avoid floating inputs