Programmable Logic Devices

Yao-Wen Chang
Department of Computer and Information Science
National Chiao Tung University
Hsinchu 300, Taiwan
E-mail: ywchang@cis.nctu.edu.tw

D. F. Wong
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712, USA
E-mail: wong@cs.utexas.edu

C. K. Wong
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Hong Kong, China
E-mail: wongck@cse.cuhk.edu.hk

1 Introduction

Performance demand, design complexity, cost-effectiveness, and time-to-market have become the most crucial concerns for contemporary logic designers. With its user-programmability, field-programmable logic has emerged as an unparalleled solution to the cost and time-to-market challenges by providing short turnaround time of designs with low risk/cost, allowing easy design changes.

A *field-programmable logic device* (*FPLD*) refers to any logic device whose function is specified by the user after the device is manufactured; general-purpose commercially available FPLDs include *Simple Programmable Logic Devices* (*SPLDs*), *Complex Programmable Logic Devices* (*CPLDs*), and *Field-Programmable Gate Arrays* (*FPGAs*).

An SPLD is capable of implementing hundreds of gates and is usually programmed by users at their sites (fields) using inexpensive programming hardware. For example, a *Programmable Logic Array* (*PLA*), introduced by Monolithic Memories (later acquired by Advanced Micro Devices, Inc.) and the first FPLD developed specifically for implementing logic functions, consists of two-level AND-OR planes that can be programmed by users to realize any sum-of-products logic function, subject to the size constraints of the device. See Figure 1 for a realization of a 4-bit comparator using a small PLA with eight inputs, eight AND gates, one OR gate, and one output. Connections are made by programming the switches, indicated by solid circles, to be ON. The field-programmability provides SPLDs the attractive advantages of low startup cost, low risk, short manufacturing time, and easy design changes.

The primary limitation of SPLDs lies in their low logic capacities (the gate count in a single chip), due to the restricted nature of the AND-OR plane; the size of the plane grows too quickly as the number of inputs

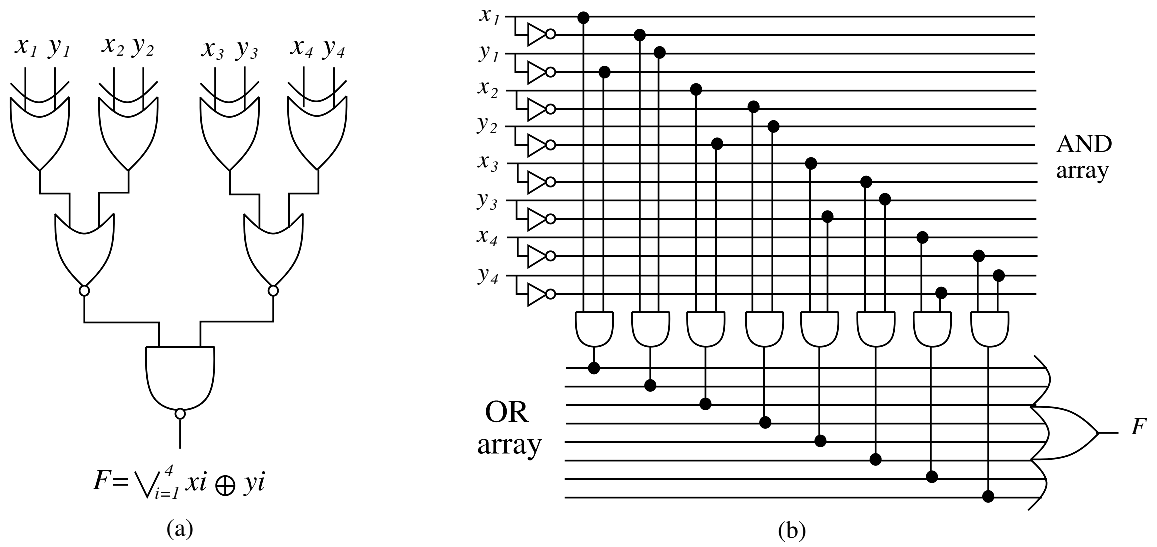


Figure 1: Realization of an example circuit (4-bit comparator) using a PLA. (a) The example circuit. (b) An implementation using a PLA with eight inputs, eight AND gates, one OR gate, and one output.

increases. One feasible approach to substantially improving chip capacities based on SPLD architectures is to incorporate multiple two-level SPLDs on a single chip with programmable switching network interconnecting them; the resulting devices are known as CPLDs. Typically, a currently existing CPLD can provide a logic capacity equivalent up to tens of SPLDs; however, it is hard to obtain a higher-capacity logic device based on the SPLD architectures.

The need to build a programmable device with a very high capacity leads to the development of FPGAs which were first introduced by Xilinx in 1984 [7]. An FPGA is a (re)programmable logic device that implements multi-level logic. It combines the scalable interconnection structure of a *Mask-Programmable Gate Array* (*MPGA*) and the programmability of an SPLD/CPLD; this combination results in a programmable device with high-capacity logic. For example, the usable gates in a current commercially available FPGA can be more than 100K, and the gate count is expected to grow rapidly as new architectures are introduced. Like SPLDs/CPLDs, FPGAs are programmable by end users at their sites (fields), eliminating the time-consuming fabrication step, and thus result in lower prototyping cost and shorter manufacturing time. These advantages have made FPGAs very popular in a wide variety of applications such as system prototyping, small- and medium-volume production, logic emulation, device controllers, data communications and telecommunications systems, digital signal processing, reconfigurable computing, etc. As an important type of field-programmable devices that provides very high-capacity logic, FPGAs have been the fastest growing segment of the total logic market during the past decade.

A typical FPGA is composed of three major components: logic modules, routing resources, and input/output (I/O) modules [4]. Figure 2 depicts the conceptual FPGA model. In an FPGA, an array of logic modules is surrounded or overlapped by general routing resources bounded by I/O modules. The logic modules contain combinational and sequential circuits that implement logic functions. The routing resources comprise pre-fabricated wire segments and programmable switches. The interconnections between the logic modules and the I/O modules are user-programmable. A logic circuit is implemented in an FPGA by partitioning logic into individual logic modules and then interconnecting the modules by programming switches. A large circuit that cannot be accommodated into a single FPGA is divided into several parts; each part is realized by an FPGA and these FPGAs are then interconnected by a *Field-Programmable Interconnect Component* (*FPIC*) (see Figure 3).

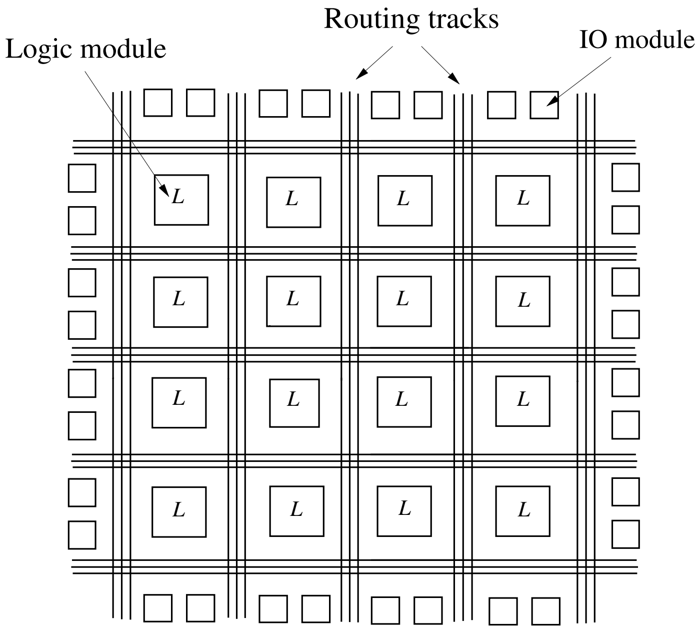


Figure 2: A typical FPGA architecture with three major components, logic modules, routing resources, and I/O modules.

As a relatively new technology, high-capacity programmable logic devices such as CPLDs and FPGAs are still constantly evolving in their architectures, and virtually each CPLD/FPGA vendor (and even each generation of products from the same vendor) has its own unique architectural features. Nevertheless, each product can be distinguished from others by three key segments: programming technology, logic-module architecture, and routing architecture. In the rest of this article, we shall focus on the discussion of the popular high-capacity programmable logic, CPLDs and FPGAs, based on the three segments.

2 Programming Technologies

CPLDs/FPGAs use programmable switches to make connections, and a typical CPLD/FPGA may contain millions of switches. Therefore, the properties of the switches, such as manufacturing flow, physical size, ON resistance, OFF capacitance, power consumption, and manufacturing reliability greatly affect the CPLD/FPGA architectural decision. Specifically, the following properties for a switch are desirable [4]:

- a simple manufacturing process
- a small chip area
- a low ON resistance and a very high OFF resistance
- a low parasitic capacitance
- low power consumption

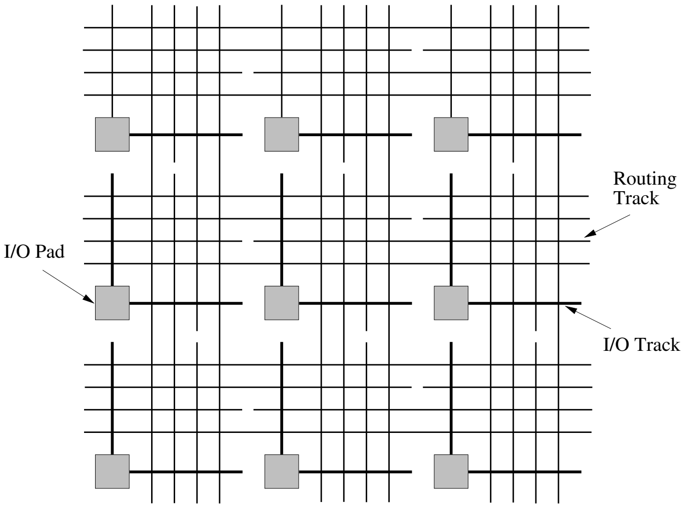


Figure 3: An FPIC architecture.

- high manufacturing reliability.

In addition to the above, reprogrammability is also an attractive feature because it makes possible to reconfigure a CPLD/FPGA, reducing risk and allowing easy design changes.

The programming technology of PLAs introduced by Monolithic Memories was based on bipolar fuses. However, most commercially available CPLDs/FPGAs are based on *Complementary Metal Oxide Silicon* (*CMOS*) technology which offers many attractive advantages such as low power dissipation, low risk, world-wide research and development investment, etc. While programming technologies such as *Erasable Programmable Read-Only Memory* (*EPROM*) and *Electrically Erasable PROM* (*EEPROM*) are popular in SPLDs and CPLDs, *Static Random Access Memory* (*SRAM*) and *antifuse* are two major switch technologies for commercial FPGAs/FPICs and some CPLDs. Each technology has its strengths as well as weaknesses. The following subsections detail important properties of the programming technologies. Although examples are taken from commercial architectures, it is not our intent here to survey all existing structures.

2.1 SRAM Programming Technology

The SRAM programming technology employs SRAM cells to control pass transistors or multiplexers as shown in Figures 4(a) and (b). For the pass-transistor element in Figure 4(a), the state of the SRAM cell controls the ON and OFF of the transistor (switch): When ON, the pass transistor exhibits a relatively low resistance ($< 2k\ \Omega$) between the two adjacent routing wires, and thus the switch is closed; when OFF, the switch is open and the transistor incurs a very high resistance between the two routing wires. For the multiplexer approach in Figure 4(b), the states of the SRAM cells serve as select signals and control the choice of the multiplexer's inputs.

The SRAM cells shown in Figures 4(c) and (d) are implemented using five and six transistors, respectively; four of them form two CMOS inverters interconnected to establish a bistable circuit element. In the case of five transistors (Figure 4(c)), the remaining one transistor serves as Read/Write control component to load the cell and read back the programming; for the six-transistor SRAM, an additional transistor, on

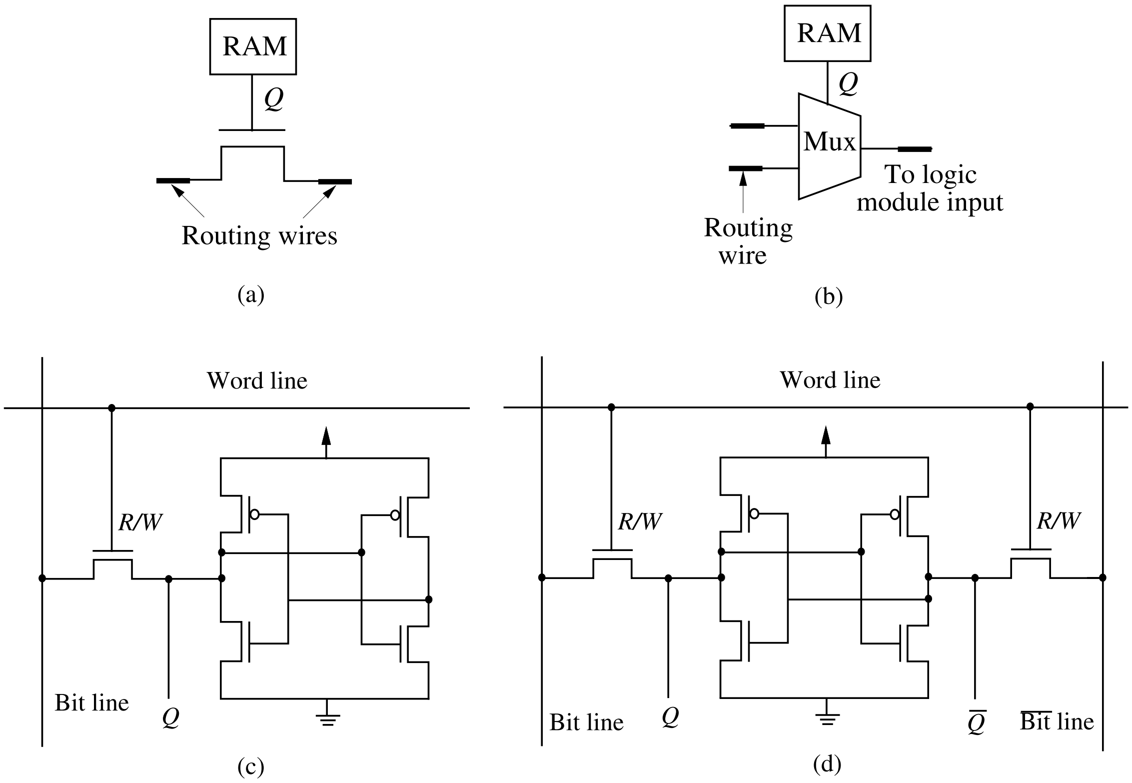


Figure 4: The SRAM programming technology. (a) A pass-transistor switch. (b) A multiplexer switch. (c) A five-transistor SRAM cell. (d) A six-transistor SRAM cell.

the other side of the cell, serves as another control component (see Figure 4(d)). The Q output of the SRAM cell is connected to and control a separate switch (could be a pass transistor or a multiplexer, as shown in Figure 4(a) or (b)). The five-transistor SRAM is more compact, whereas the six-transistor one has higher noise immunity and is easier to design. Due to its stability, the six-transistor SRAM gets advantages for deep-submicron process and low-power supply technologies [22].

SRAM programming technology has the following major advantages: simple manufacturing flow, fast in-circuit reprogrammability, and low power consumption. However, a major disadvantage of SRAM programming technology is its large physical size. For example, in Xilinx FPGAs [32], it takes at least one pass transistor to serve as a programmable switch, and five (e.g., XC4000, XC5200) or six (e.g., XC6200) transistors to implement an SRAM cell, as illustrated in Figure 4(c) or (d).

One interesting property of SRAM is its volatility—a configuration is lost when its power is off. On one hand, this implies a system overhead that external permanent memory such as a *Read-Only Memory* (*ROM*), *Programmable Read-Only Memory* (*PROM*), or a disk is needed to store and provide programming configurations. On the other hand, volatility gives SRAM-based FPGAs better design security because a design configuration is lost when power is removed.

2.2 Antifuse Programming Technology

An antifuse is a two-terminal, one-time programmable circuit element with high resistance ($> 100M\ \Omega$) between its terminals in the unprogrammed state and low resistance ($\leq 500\ \Omega$) in the programmed state. Figures 5(a) and (b) illustrate the structures of two antifuses, called a PLICE (Programmable Low-Impedance Circuit Element) and a ViaLink, used in Actel FPGAs [1, 14] and QuickLogic FPGAs [25], respectively. The PLICE antifuse consists of an Oxygen-Nitrogen-Oxygen (ONO) dielectric layer (served as an insulator) sandwiched between a polysilicon and an n+ diffusion layers (served as conductors). Programming is performed by applying a higher-than-operating voltage (say, 18 V) across the antifuse's terminals, causing the dielectric breakdown and drastically reducing the device resistance. The ViaLink antifuse is composed of a sandwich of very high resistance layer ($> 1G\ \Omega$) of programmable amorphous silicon between two mental layers. When a programming voltage is applied, a mental-to-mental link is formed by permanently converting the silicon to a low resistance state (30–80 Ω).

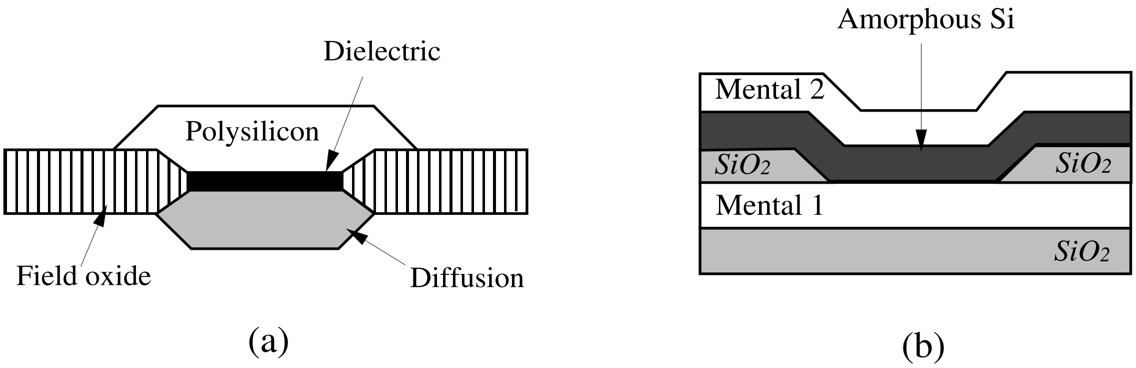


Figure 5: The antifuse programming technology. (a) The ONO antifuse structure. (b) The amorphous antifuse structure.

Two major advantages of the antifuse are its small size and relatively low ON resistance and OFF capacitance; for example, the size of an amorphous antifuse is approximately 1 μm^2 in a 0.65-micron process [25]. These advantages allow a much denser switch population and thus could alleviate the routing constraints imposed by the limited connectivity of routing resources. However, antifuse programming technology has the following major disadvantages: relatively complex manufacturing process and non-reprogrammability.

2.3 EPROM and EEPROM Programming Technology

An ultraviolet (UV)-erasable PROM (EPROM) is typically based on a floating gate structure as illustrated in Figure 6. If sufficient charge is trapped on the floating gate by applying higher-than-operating voltages between the control gate (13–14 V) and the drain of the transistor (12 V), the floating gate becomes charged negatively. The process increases the threshold voltage of the transistor to around 7 V, setting the transistor to the OFF state for all normal circuit voltages, maximally 5–6 V. The process can be reversed out-of-circuit by exposing the floating gate to UV light, giving the trapped charge sufficient energy to escape from the gate dielectric; this process reduces the threshold voltage and makes the transistor function normally.

The Electrically Erasable PROM (EEPROM) programming technology typically uses two transistors, one access and one programmed transistors, in a ROM cell. The programmed transistor performs the same function as the floating gate in an EPROM, with both charge and discharge being done electrically in-circuit,

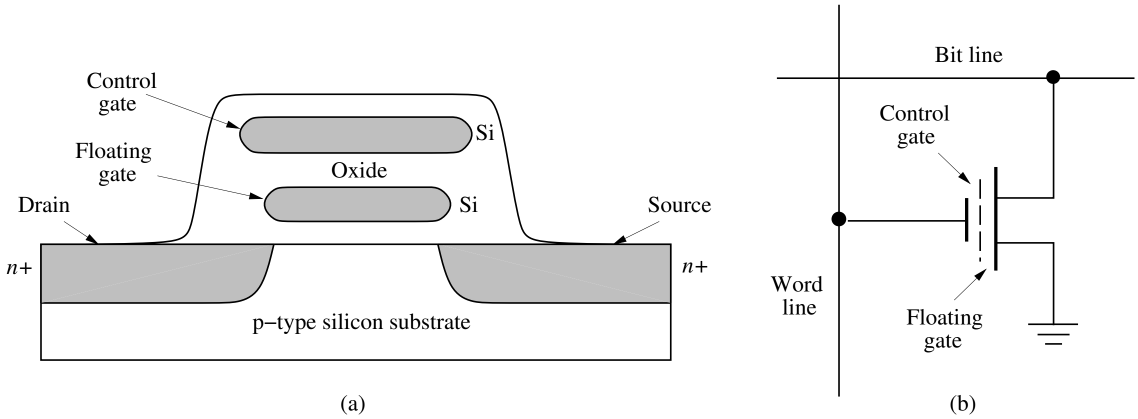


Figure 6: The EPROM programming technology. (a) UV-erasable EPROM structure. (b) Circuit symbol for a floating-gate EPROM cell.

without UV light. The access transistor allows programming of a cell.

The major advantages of EPROM and EEPROM technologies are their reprogrammability and full testability before shipment (especially for EEPROM). Also, unlike the SRAM technology, EPROM or EEPROM requires no external permanent memory to program the chip at power-up. However, the EPROM/EEPROM technology suffers from some drawbacks such as relatively high ON-resistance, high static power consumption, and complicated manufacturing processes [28].

Recently, a new non-volatile, electrically erasable programming technology called *flash memory* is introduced. A flash memory element occupies smaller physical size (about one-third of the size of an EEPROM cell), offers higher reliability, and has faster device programming time than the EEPROM one. Due to the advantages, flash memory is getting popular on the market.

2.4 Summary of Programming Technologies

Table 1 compares the properties of SRAM, ONO antifuse, amorphous antifuse, EPROM, and EEPROM programming technologies based on a 0.65 μm process technology; it gives the name of the technology, the relative complexity of manufacturing process (measured in the number of extra processing steps required beyond standard CMOS [28]), the availability of reprogramming, the approximate relative size of the switch, the series resistance of an ON switch, the parasitic capacitance of an OFF switch, the relative degree of power consumption, and the volatility of the configuration.

3 Logic-Module Architecture

A logic circuit is implemented in a CPLD/FPGA by partitioning logic into individual logic modules and then interconnecting the modules by programming switches. A logic module has a fixed number of inputs and outputs. Certain set of functions can be implemented using a logic module, depending on the functionality of the module. The logic modules of a high-capacity commercial programmable device are typically based on lookup tables, multiplexers, transistor pairs, basic logic gates, or SPLDs, with the first four types being popular in FPGAs and the last in CPLDs. We detail lookup table- and multiplexer-based logic modules in the following. Due to their simplicity, we only brief those based on transistor pairs, basic gates, and SPLD

Programming technology	SRAM	ONO anti-fuse	Amorphous anti-fuse	EPROM	EEPROM
Manufacturing Complexity	$+++$	$+$	$+$	$-$	$--$
Reprogrammable?	Yes in circuit	No	No	Yes out of circuit	Yes in circuit
Physical size	Large (20X)	Small (2X)	Small (1X)	Large (40X)	Large (80X)
ON resistance (Ω)	600–800	100–500	30–80	1–4 K	1–4 K
OFF capacitance (fF)	10–50	3–5	1	10–50	10–50
Power consumption	$++$	$+$	$+$	$-$	$-$
Volatile?	Yes	No	No	No	No

Table 1: Comparison of programming technologies ($+$: desirable; $-$: not desirable).

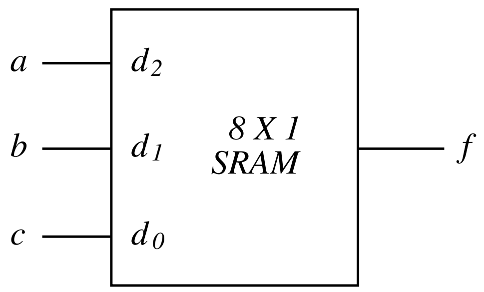
at the end of this section.

3.1 Lookup Table-Based Logic Modules

A lookup table (LUT) based logic module is a segment of SRAM. The programming data defining the logic configuration are loaded into the SRAM at power-up. A K-input LUT is a $2^K \times 1$ SRAM with K address lines served as inputs and the 1-bit SRAM output as the LUT output. For example, if the function $f = ab + \overline{a}c$ needs to be implemented using a 3-input LUT, then the truth table shown in Figure 7(a) is loaded into the LUT, and thus the $2^3 \times 1$ SRAM would have a 0 stored at address 000, a 1 at 001, etc, as given in the truth table.

a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

(a)



(b)

Figure 7: Lookup table based logic. (a) Truth table for $f = ab + \overline{a}c$. (b) An 8×1 SRAM.

The main advantage of LUTs lies in their high functionality—a K-input LUT can realize any function of up to K inputs and there are 2^{2^K} such functions. However, the LUTs are only feasible for small values of K because 2^K memory cells are required for a K-input LUT with only combinational logic; typically, the value of K is 3, 4, 5 or 6. Further, the value of K may be even smaller in order to implement multi-output or sequential logic.

3.2 Multiplexer-Based Logic Modules

A multiplexer-based logic module is typically composed of a tree of 2-to-1 multiplexers. For example, the Actel ACT 3 C-Module [1] shown in Figure 8 consists of three 2-to-1 multiplexers, implementing the following function:

$$z \;=\; \overline{(s_3+s_2)}\,\overline{s_1 s_0}\,d_0 + \overline{(s_3+s_2)}\,s_1 s_0 d_1 + (s_3+s_2)\,\overline{s_1 s_0}\,d_2 + (s_3+s_2)\,s_1 s_0 d_3.$$

The inputs to the logic module are either data inputs d_0, d_1, d_2, d_3 or the multiplexer select inputs s_0, s_1, s_2, s_3. Therefore, the logic module can be used to implement a wide range of different functions of up to eight variables (specifically, 766 functions for the logic module); for example, the function $f = ab + \overline{a}c$ can be realized by setting $d_0 = \times, d_1 = \times, d_2 = c, d_3 = b, s_0 = a, s_1 = 1, s_2 = 1$, and $s_3 = \times$, where $\times$ means don't-care. Also, flip-flops can be incorporated into the multiplexer-based logic module to implement sequential logic.

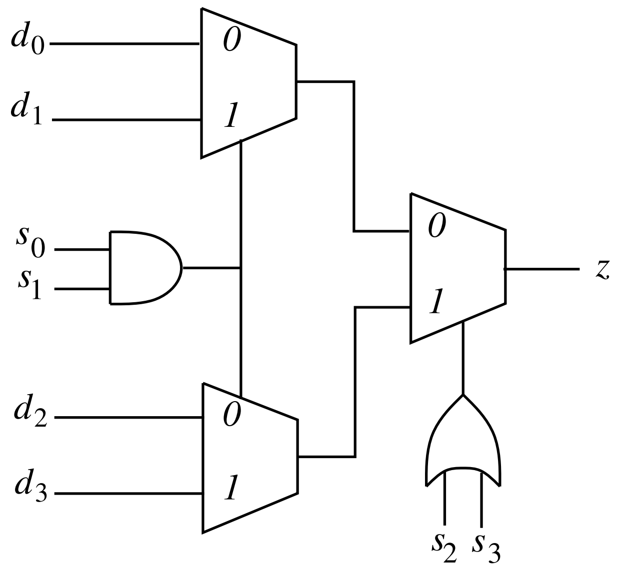


Figure 8: A multiplexer-based logic module with data inputs d_0, d_1, d_2, and d_3, select inputs s_0, s_1, s_2, and s_3, and output z.

3.3 Logic-Module Trade-offs

Besides lookup tables and multiplexers, transistor pairs and basic gates such as 2-input NAND, EXOR, AND gates are used to form logic modules in several commercial FPGAs [11]. Unlike lookup tables and multiplexers which can implement a versatile set of logic functions (but are substantially larger in size), the transistor pairs and basic gates can implement only a small set of functions (but are much smaller).

To capture the differences, logic modules are often classified by their *granularity*: *coarse grain* or *fine grain* [28]. A coarse-grain logic module refers to one which can implement a large number of Boolean functions, whereas a fine-grain one would implement just a few. The choice of granularity lies in the trade-offs in logic density, utilization, and performance [28]. Typically, a coarse-grain logic module requires using a much smaller number of programmable switches and local interconnect, implying smaller area and delay. Therefore, coarse-grain programmable devices often have higher logic capacity and better timing performance. On the other hand, a fine-grain logic module can achieve a much higher logic utilization rate because it is easier to use small gates efficiently.

While FPGAs employ lookup tables, multiplexers, transistor pairs, and/or basic gates for their logic modules, most commercial CPLDs uses some type of SPLDs such as *Programmable Array Logics* (*PALs*) or macrocells as their basic building modules [2, 33]. (A PAL comprises a programmable AND-plane feeding into a *fixed* OR-plane, whereas the OR-plane in a PLA is also programmable). Often, the SPLD-based logic modules are grouped together, in a hierarchical manner, to form larger logic components.

4 Routing Architecture

The interconnect architecture of a CPLD/FPGA includes routing resources and their interconnection topology on the CPLD/FPGA. CPLD/FPGA routing resources consist of pre-fabricated wire segments and programmable switches; routing in a CPLD/FPGA is performed by programming the switches to connect the wire segments. There are four major interconnection topologies for commercial CPLDs/FPGAs: array-based, row-based, sea-of-gates, and hierarchical models.

An array-based FPGA consists of a two-dimensional array of logic modules which can be connected by general routing resources. (See Figure 9(a).) As mentioned earlier, the logic modules house circuits that implement logic functions. The routing resources comprise horizontal and vertical routing tracks and user-programmable switches.

A row-based CPLD/FPGA consists of multiple rows of logic modules. (See Figure 9(b).) Routing wires run in the channels between two adjacent rows and also inside the channels vertically. There are additional global vertical wires providing connections among different rows (not shown in Figure 9(b)).

Unlike the array- and row-based architectures where there are routing channels separating logic modules, all logic modules in a sea-of-gates structure directly abut. (See Figure 9(c).) The close proximity of logic modules allows direct connections between adjacent modules, significantly improving circuit performance. A routing network runs over the top of the modules, enabling efficient use of the available chip area.

The hierarchical structure is composed of logic modules connected by multiple levels of interconnect. (See Figure 9(d).) At the first hierarchical level, several logic modules form a group, and local routing resources for the modules are provided. Together, the logic modules and their interconnect form a local component. Again, the second level of interconnect ties together a group of the components of the first level, with shared routing resources for the components. The construction for higher levels of components proceeds in the same manner.

5 Commercial CPLD/FPGA Examples

CPLD/FPGA architectures differ significantly from vendor to vendor. Due to the limitation of space, we shall give an overview for only the major programmable logic devices on the market today. Table 2 summarizes the architectural features of popular commercially available CPLDs/FPGAs.

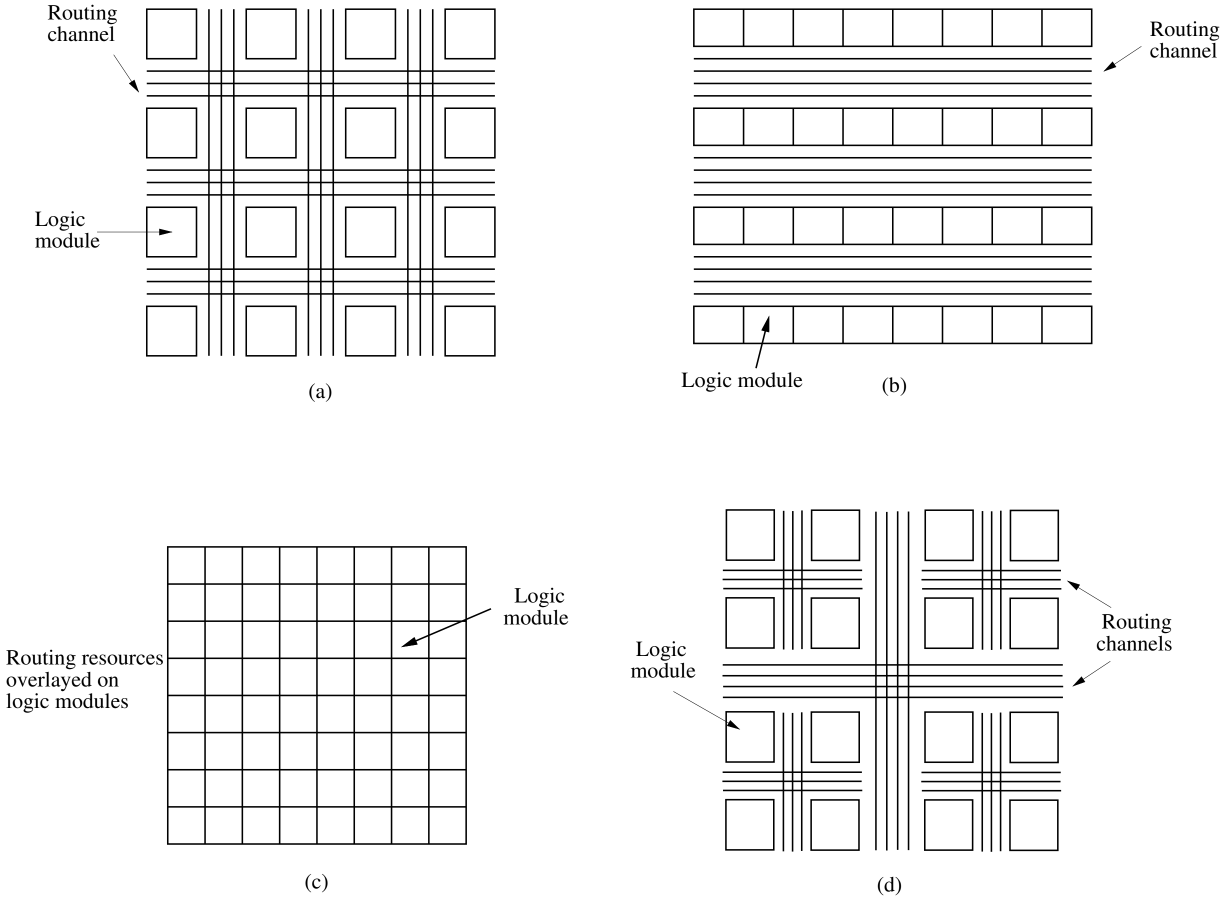


Figure 9: The four major CPLD/FPGA architectures. (a) The array-based model. (b) The row-based model. (c) The sea-of-gates model. (d) The hierarchical model.

5.1 Xilinx FPGAs/CPLDs

Xilinx introduced the first FPGA, the XC2000 series, in 1984 and now offers several more generations of FPGAs, including SRAM-based XC3000, XC4000, XC5200, and XC6200, and antifuse-based XC8100, etc. Besides the FPGA products, Xilinx also offers several families of CPLDs such as EPROM-based XC7300 and Flash-based XC9500. We focus on the SRAM-based XC4000 FPGAs because of their popularity on the market.

With the capacity ranging from 3K to 85K logic gates (1997), the Xilinx XC4000-series FPGAs provide a wide range of selection. The XC4000-series FPGA adopts the symmetrical array-based architecture; each chip consists of a two-dimensional array of logic modules called configurable logic block (CLBs) and peripheral input/output blocks interconnected by horizontal and vertical routing channels.

5.1.1 XC4000 Logic Modules

The principal CLB structure is shown in Figure 10. Each CLB of the XC4000 FPGA contains three lookup tables implementing combinational functions; two 4-input LUTs (implementing two functions labeled F' and G') feed a 3-input LUT which implements a function of F', G', and input $H1$. Setting the multiplexers properly, F' or H' can be connected to the combinational output X, and G' or H' can be assigned to the Y

Company	Programmable logic	Architecture	Logic-module type	Programming technology	Example products
Actel	FPGA	Row-based	Multiplexer	Antifuse	ACT 3
Altera	CPLD	Hierarchical-SPLD	SPLD block	EPROM EEPROM	MAX 9000
		2-D array	Lookup table	SRAM	FLEX 10K
AMD/Vantis	CPLD	Hierarchical-SPLD	SPLD block	EEPROM	MACH 5
Aptix	FPIC	Symmetrical array	-	SRAM	FPIC/R
Cypress	CPLD	Hierarchical-SPLD	SPLD block	EPROM Flash	Flash370
Lattice	CPLD	Hierarchical-SPLD	SPLD block	EEPROM	ispLSI 6000
Lucent	FPGA	Symmetrical array	Lookup table	SRAM	ORCA 2
Motorola	FPGA	Hierarchical	Basic gates	SRAM	MPA1000
QuickLogic	FPGA	Symmetrical array	Multiplexer	Antifuse	pASIC2
Xilinx	FPGA	Symmetrical array	Lookup table	SRAM	XC4000
		Sea-of-gates	Multiplexer	SRAM	XC6200
		Sea-of-gates	Basic gates	Antifuse	XC8100
	CPLD	Hierarchical-SPLD	SPLD block	EPROM Flash	XC7300 XC9500

Table 2: Summary of major commercially available programmable devices.

output. Therefore each CLB can implement any two functions of up to four variables, any single function of five variables, or selected functions of up to nines variables.

Each CLB also contains two edge-triggered D-type flip-flops for implementing sequential logic. Each of the D inputs is selected by one of the two 4-to-1 multiplexers which select from F', G', H', and input DIN. Thus the CLB can pass any of the combinational outputs to a flip-flop. If the combinational functions F', G', and H' are used as outputs and do not drive the flip-flops, DIN can be assigned to the flip-flops to store values from other sources.

5.1.2 XC4000 Routing Architecture

The routing channels in a Xilinx XC4000E-series FPGA consist of three types of interconnect, distinguished by the relative length of their segments: single-length lines, double-length lines, and long lines. (See Figure 11.) The single-length lines form a grid of horizontal and vertical lines that intersect at switch modules. The double-length lines consist of a grid of segments twice as long as the single-length lines. The long lines are a grid of segments that run the entire vertical or horizontal channel.

The horizontal and vertical single- and double-length lines can be interconnected using a programmable switch module shown in Figure 12(a). There are six pass transistors per switch module interconnect point, used to establish connections between the lines. Several examples of connections are shown in Figure 12(b). At point a, all four lines segments are electrically connected together by closing three transistors. At point b or c, two distinct signal nets pass through the interconnect point simultaneously by closing two non-adjacent transistors.

In addition to the wire segments mentioned above, there are also special interconnections for clocks and three-state buffers. Recently Xilinx offers the XC4000EX-series FPGAs which have several additional types

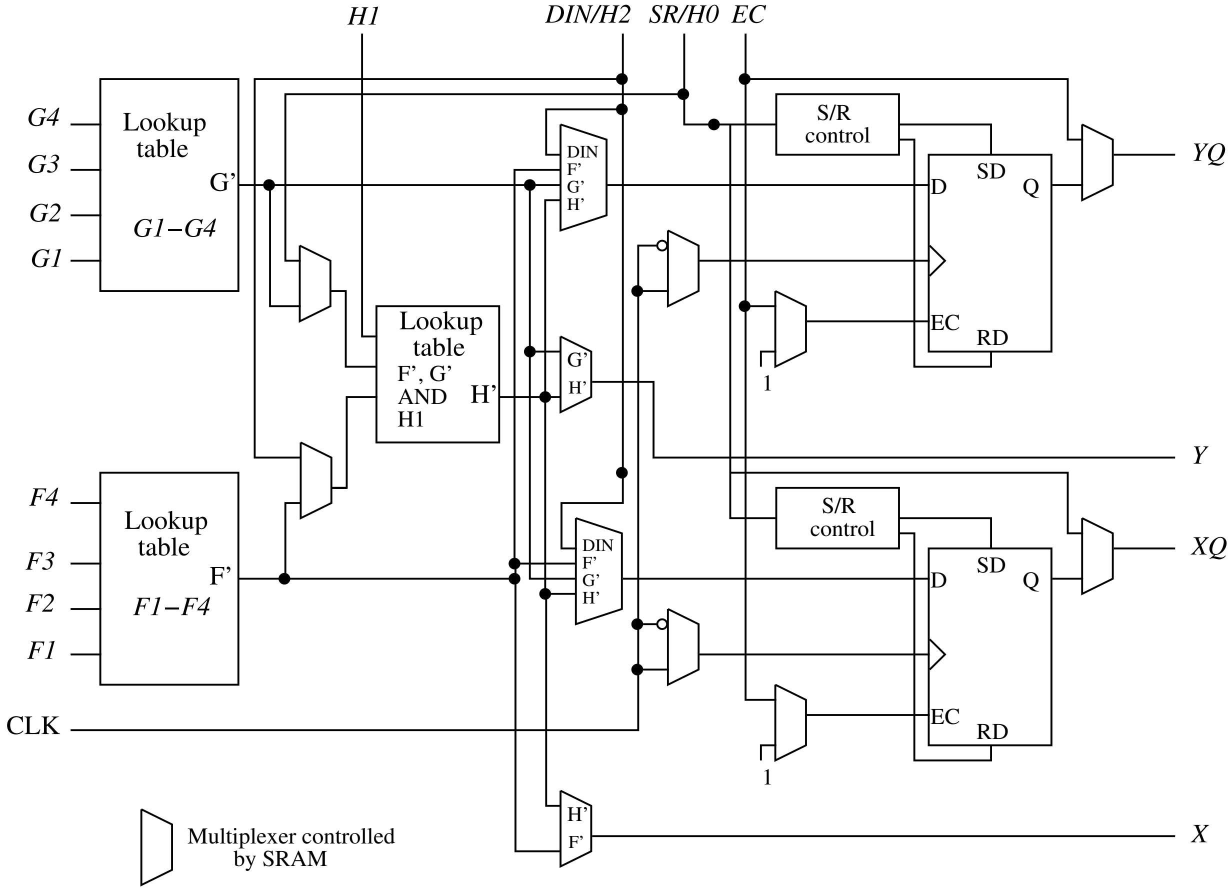


Figure 10: The Xilinx XC4000 logic module. Each logic module contains two 4-input lookup tables, one 3-input lookup table, and two edge-triggered D-type flip-flops.

of interconnect such as quad lines that span the length of four CLBs and direct interconnects that connects CLB outputs to adjacent neighbors.

5.2 Actel FPGAs

Actel now offers two major series of FPGAs, the Integrator and the Accelerator series, both with the row-based structure and the antifuse technology that provides one-time and non-volatile programming. The Accelerator series is intended for higher-performance applications, while the Integrator offers a more attractive price. Due to the small size and low resistance/capacitance of the antifuse, the Actel FPGA is compact and fast. We focus on the Accelerator ACT 3 FPGA in the following discussion. (See Figure 13 for the basic architecture of an ACT 3 chip.)

5.2.1 ACT 3 Logic Module

The ACT 3 consists of two types of logic modules, C (combinational) modules and S (sequential) modules (see Figure 14). The C module, as shown in Figures 14(a) (and also in Figure 8), is a multiplexer-based logic module that implements combinational logic functions. With the four data inputs d_0, d_1, d_2, d_3 and the four select inputs s_0, s_1, s_2, s_3, the C module can be used to implement 766 distinct functions of up to 8

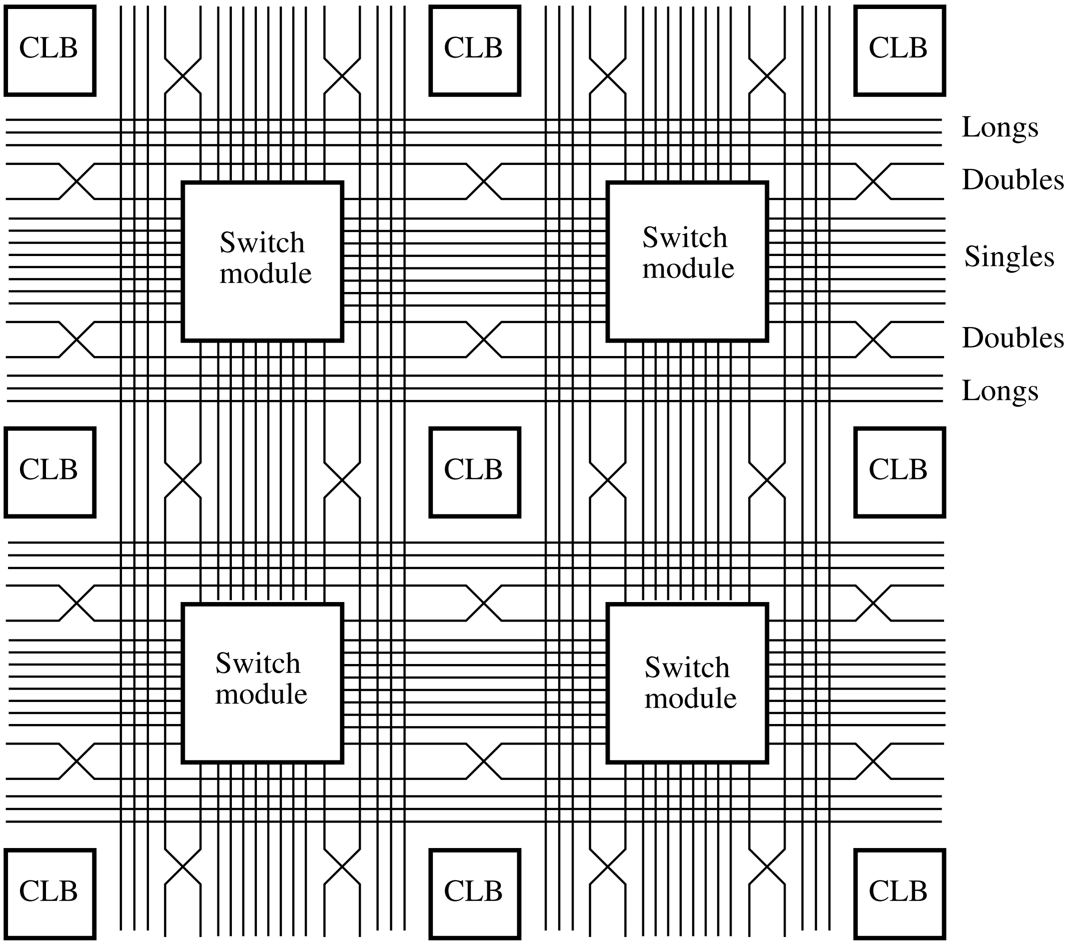


Figure 11: The Xilinx XC4000E routing architecture. Each routing channel consists of three major types of interconnect: single-length lines, double-length lines, and longlines.

variables. The S module uses a C-module front end with a dedicated edge-triggered D-type flip-flop on the output of the module, used for implementing sequential logic functions.

5.2.2 ACT 3 Routing Architecture

The Actel routing architecture consists of horizontal and vertical routing tracks. The routing tracks may either be of continuous length or broken into pieces called *segments*. Two adjacent wire segments can be joined together by programming an antifuse switch to construct a longer connection. Many horizontal tracks lie in each channel between two adjacent rows of logic modules. The vertical tracks run through the logic modules and the horizontal channels. The segmented routing architecture leads to a better performance and routability trade-off.

5.3 Altera CPLDs

Altera offers several types of CPLDs, including Classic, MAX, and FLEX series. The Classic- and MAX-series CPLDs are based on EPROM and/or EEPROM technologies and hierarchical-SPLD logic modules, whereas the FLEX-series CPLDs adopt the SRAM technology and lookup table-based logic modules. We

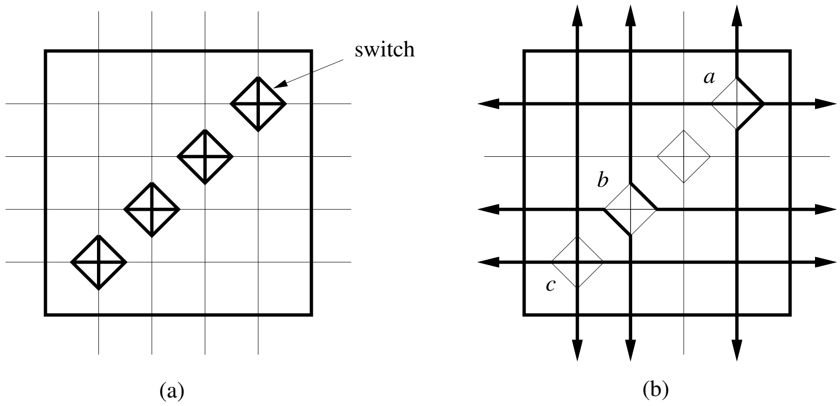


Figure 12: The Xilinx XC4000 switch-module structure. (a) Six pass transistors per switch-module interconnect point. (b) Examples of connections.

focus on the FLEX-series CPLDs because of their state-of-the-art technology.

The FLEX programmable logic device, FLEX 8K or FLEX 10K, exhibits a combination of FPGA and CPLD technologies. Like CPLDs, the device consists of three levels of hierarchy and non-segmented routing architectures; like FPGAs, its logic module is based on 4-input lookup tables. Figure 15 illustrates the FLEX 10K architecture. The FLEX 10K chip consists of a two-dimensional array of two types of logic modules, logic array blocks (LABs) and embedded array blocks (EABs), interconnected by *continuous* row and column channels. Each LAB consists of eight logic elements that communicate through a local interconnect. Each logic element contains a 4-input lookup table, a D-type flip-flop, carry circuitry (for arithmetic operation), and cascade circuitry (for wide fan-in functions). Located on the left end of each row, each EAB provides 2K bits of memory that is good for RAM and ROM functions. Also, the EAB can be configured to implement a more sophisticated function by using it as a large lookup table. The row and column routing channels consist of non-segmented tracks that run the entire width and length of the device—similar to the long lines of Xilinx XC4000 FPGAs. Connections between horizontal and vertical routing channels are via active buffers and multiplexers. The homogeneous routing tracks make the FLEX CPLDs not only easier for design automation but also more predictable for delay estimation.

5.4 AMD/Vantis CPLDs

AMD/Vantis offers several series of CPLDs, including MACH 1, 2, ..., 5. The MACH-series CPLDs use the EEPROM programming technology and the hierarchical architecture, with their logic modules based on multiple PALs. In the following discussion, we focus on the most advanced MACH 5 series CPLDs.

The MACH 5 architecture consists of PAL modules connected by two levels of interconnect. (See Figure 16.) Each group of four PAL modules has its own local routing resource called block interconnect. Together, the four modules and their block interconnect form a super-module. The second level of interconnect, the global interconnect, then ties all of the super-modules together.

The MACH 5 PAL block consists of a product-term array feeding into macrocells. The product-term array exhibits a sum-of-products architecture used in PAL devices. Comprising a D-type flip-flop, a control bus, and local routing resources, each macrocell can be configured for both combinational and sequential operations. A logic allocator is used to assign product terms to macrocells, and up to eight groups of four

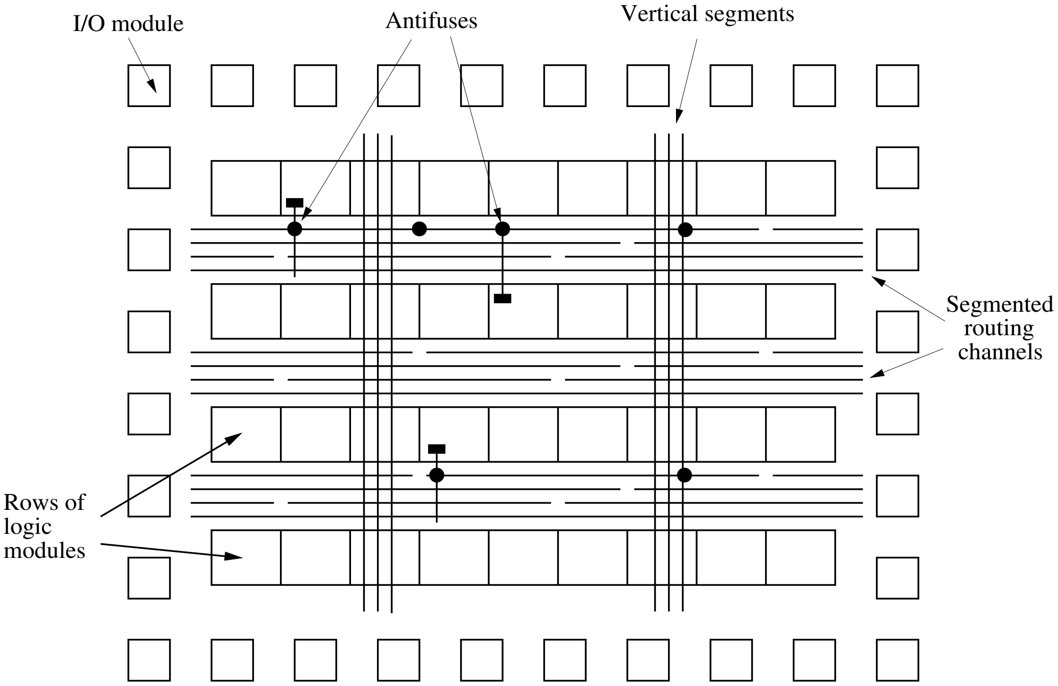


Figure 13: The Actel ACT FPGA structure. An ACT FPGA consists of multiple rows of logic modules. Routing wires run in the channels between two adjacent rows and also inside the channels vertically.

product terms can be fed to one macrocell. The logic allocator can also perform logic switching functions: as a design changes, the allocator will reassign logic to macrocells to retain pinout.

6 Design Process for Programmable Logic

This section briefly describes the CPLD/FPGA design flow; interested readers are referred to Cross Reference 1, [4, 32], and data books from vendors [1, 2, 11, 20, 25, 33, 34] for more details. The CPLD/FPGA design cycle consists of three major stages described as follows:

1. System design: The design process of CPLDs/FPGAs starts with system design, including specifying formal specifications of a system, drawing functional diagrams, and designing logic and circuit structures to realize the system. Typically, logic design uses schematic capture or hardware description language such as Verilog and VHDL. In system design, a designed circuit is usually optimized by a CAD tool and represented by Boolean expressions or circuit diagrams. Then, the optimized, technology-independent circuit is mapped to a set of logic modules of the target CPLD/FPGA; the mapping process is called *technology mapping*. Together, the process of logic optimization and technology mapping is called *logic synthesis*.

2. Physical design: In the physical design, the optimized and mapped circuits are then converted into geometric patterns. The CPLD/FPGA physical design process consists of three major steps: partitioning, placement, and routing. A mapped circuit is partitioned into a set of subcircuits so that each subcircuit can fit into a logic module of the device. Then, the logic modules are placed in the device by a placement program. In the final step of the physical design, a router assigns wire segments and

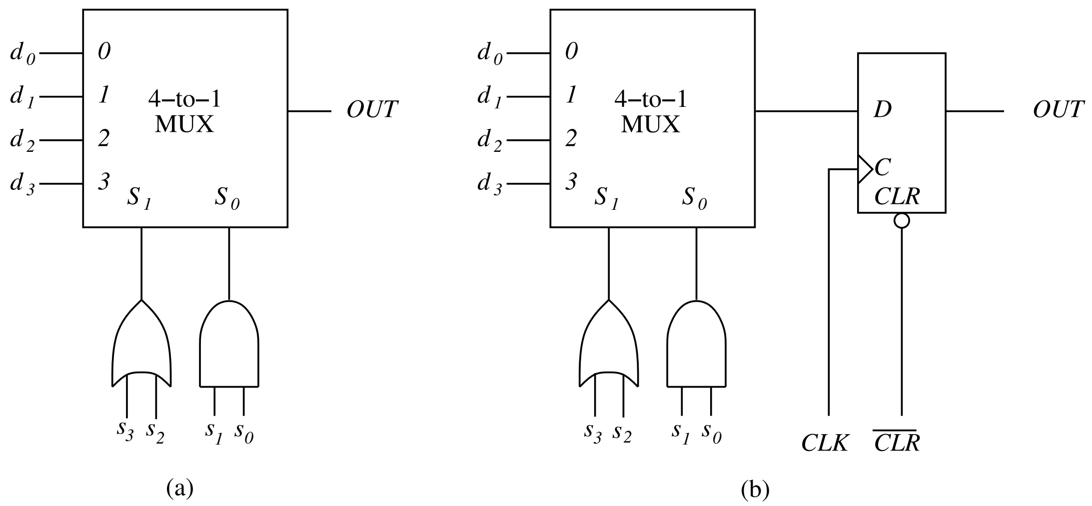


Figure 14: The Actel ACT 3 logic module. (a) C (combinational) logic module. (b) S (sequential) logic module.

chooses programmable switches to establish the required connections among the logic modules.

3. Customization: Upon successful completion of routing, the output from the physical design process is fed to a programming unit, which customizes the programmable logic chip.

Note that, though not mentioned in the above, testing, simulation, and verification are integrated into many stages of the design process. While the tasks in the system design, except technology mapping, are common between programmable logic designs and traditional semi-custom designs, those in technology mapping and the physical design are specific to the target technologies.

7 Research in CPLD/FPGA architectures

This section briefly describes important research results in the architectural design for CPLDs/FPGAs.

Most research in CPLD/FPGA architectures is based on the model shown in Figure 17. In the model, a CPLD/FPGA consists of an array of logic modules which can be connected by routing resources. The routing resources comprise segments of wires and two kinds of modules, *switch modules* and *connection modules*, which contain user-programmable switches. An intersection of a horizontal and a vertical routing channels is referred to as a switch module; the switch module serves to connect wire segments, and this requires using programmable switches inside it. Connection modules are used to connect logic-module pins to wire segments; this is performed by programming switches inside connection modules.

7.1 Logic-Module Design

7.1.1 Effect of Logic-Module Granularity on CPLD/FPGA Density and Performance

The effect of logic-module granularity on CPLD/FPGA density and performance has been studied extensively [17, 18, 27, 29]. The larger the granularity of a logic module, the smaller number and level of modules are required to implement a design. However, larger granularity of a logic module needs more circuitry in a module, implying more area is consumed and longer routing delay is incurred for a module. Hence, there

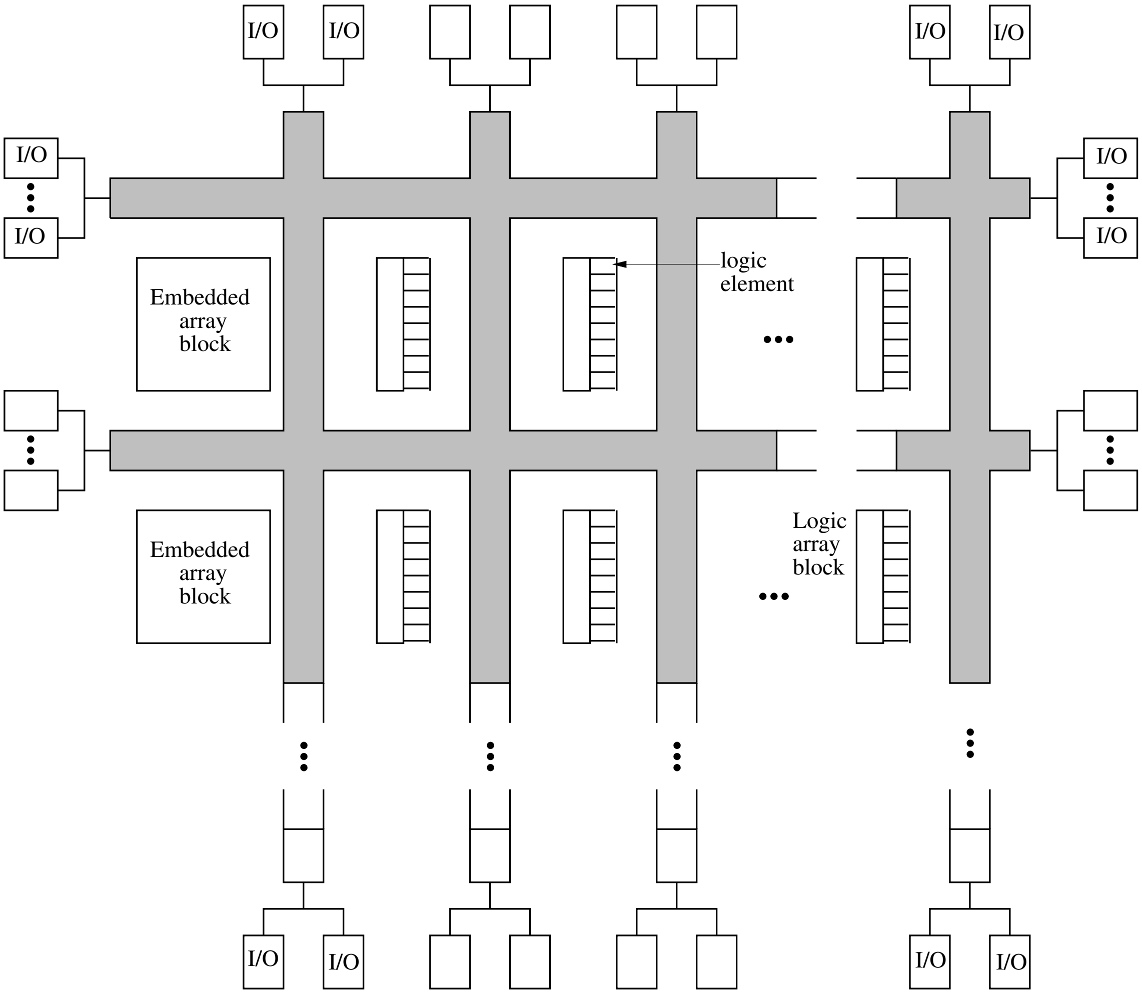


Figure 15: The Altera FLEX 10K structure. A FLEX 10K chip consists of a two-dimensional array of two types of logic modules, logic array blocks and embedded array blocks, interconnected by continuous row and column channels.

are trade-offs between logic-module granularity and area efficiency, and between the granularity and timing performance. The study by Kouloheris and El Gamal [18] shows that a 4-input, 1-output LUT gives best area efficiency of any K-input, M-output LUT-based logic module. Based on the study of [17, 29], the effect of granularity on performance depends on the RC delay in the programmable switch used: If the RC delay is relatively small, the best granularity is around $K = 3$ or 4; on the other hand, if the RC delay is larger, the best granularity is around $K = 6$ or 7.

7.1.2 Effect of Logic-Module Architecture on CPLD/FPGA Density and Performance

In addition to the study on the effect of a fixed-input LUT, other researchers have also investigated other logic-module architectures, including PLA-based logic modules [18], heterogeneous modules [15], hybrid modules [16], and universal logic modules [19, 31].

Kouloheris and El Gammal explored the trade-off between the area of a PLA-based CPLD/FPGA and its module granularity [18]. They observed that a PLA-based architecture achieves the best area efficiency

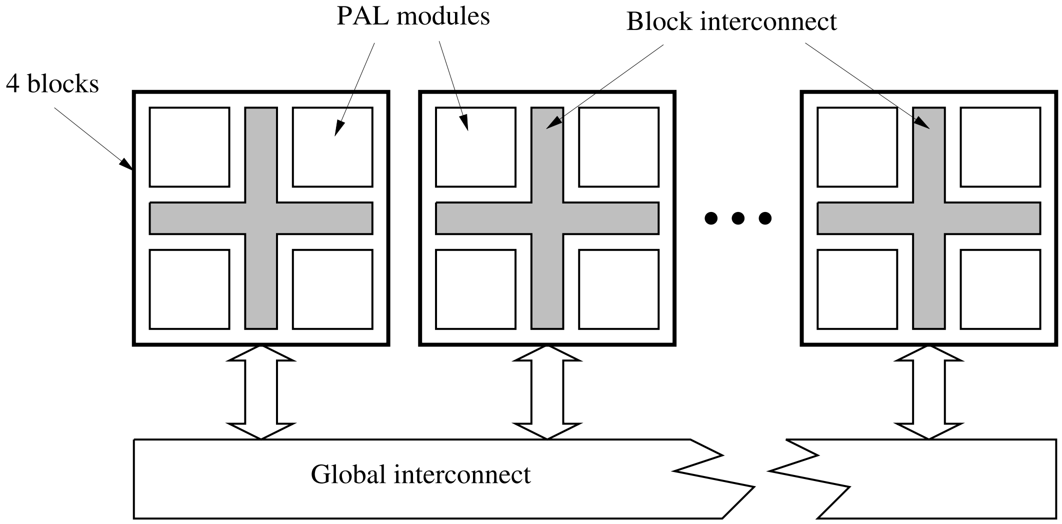


Figure 16: The AMD/Vantis MACH structure. A MACH 5 CPLD consists of PAL modules connected by two levels of interconnect.

when a module with 8–10 inputs, 3–4 outputs, and 12–13 product terms, and the areas for the best PLA-based logic module and the 4-input 1-output LUT-based one are comparable when implemented by the same programming technology.

He and Rose noticed that 4-input LUTs make most efficient use of area, whereas more coarse-grain modules such as 5-, 6-, and 7-input LUTs are superior in terms of timing performance. Based on this observation, they proposed to use various sized LUTs within a CPLD/FPGA chip to provide a better trade-off between density and performance [15]. They investigated the effect of the combination of two differently sized LUTs, say a p- and q-input LUT combination (denoted by (p, q)), and suggest the combinations such as (5, 2), (4, 2), or (4, 3) to be better than a homogeneous 4-input LUT.

Kaviani and Brown presented a hybrid logic-module architecture based on a combination of LUTs and PALs/PLAs [16]. The basis for this idea is that the strength of LUT-based architectures lies in their very high logic density, while the PLA/PAL-based ones usually achieve very high speed performance. Their hybrid architecture employs 4-input LUTs in the mix, since they have been found to be best for many applications. The PAL/PLA hybrid uses a programmable AND-plane connected to a half fixed and a half programmable OR-plane. Kaviani and Brown suggested allocating half of the logic-module area to 4-input LUTs and the other half to hybrid PLAs/PALs.

One other segment of research in logic modules focuses on designing the modules with high functionality, called *universal logic-module design*. The goal is to design a function $U(y_1, \ldots, y_m)$, with the minimum m, which can realize all the n-input functions $F(x_1, \ldots, x_n)$. The general approach is to specify the set of functions that can be covered by U by assigning y_i to 0, 1, x_j, or $\overline{x_j}$, permuting or complementing some inputs, or negating the output of U [10, 23]. Lin, Marek-Sadowska, and Gatlin [19] and Thakur and Wong [31] independently extended the above idea to the logic-module design for CPLD/FPGAs. While the techniques used in [19] are computationally intensive, the work by [31] gives a more efficient algebraic approach which is suited to larger designs.

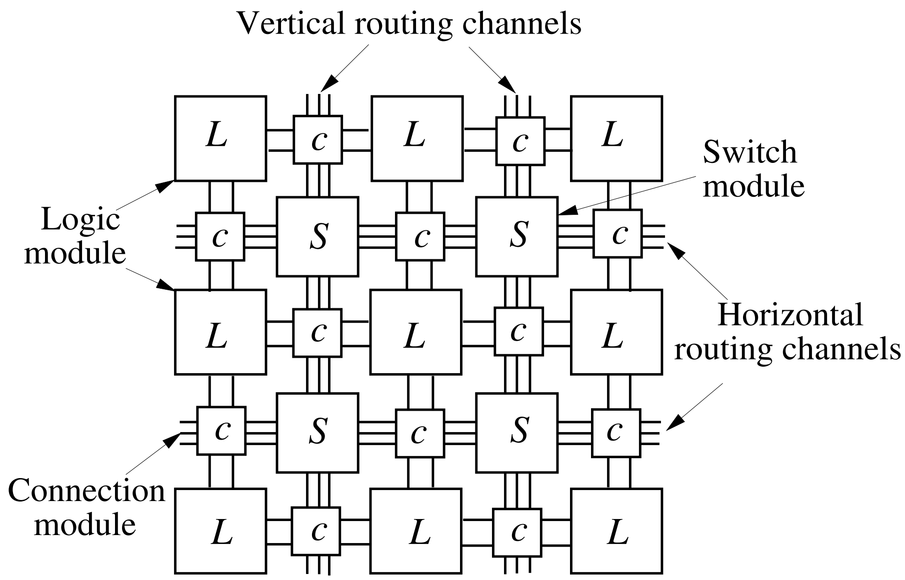


Figure 17: The FPGA model. An FPGA consists of an array of logic modules which can be connected by routing resources (switch modules, connection modules, and wire segments).

7.2 Routing Resources

CPLD/FPGA routing resources consist of programmable switches and wire segments. Routing in CPLD/FPGA is performed by programming the switches to connect the wire segments. The programmable switches usually incur significantly high RC delay, and consume a large amount of area. Due to the area and delay constraints, the number of switches that can be placed in a switch module is usually limited, implying limited routability. Therefore, there is a basic trade-off between routability and area/performance for switch-module architectures. Also, to achieve high performance, the number of switches used for routing a net is restricted. On the other hand, fewer switches would reduce routability. The trade-off between routability and performance suggests the existence of an "optimal" segmentation architecture that needs to be explored.

7.2.1 Switch-Module Modeling

Zhu, Wong, and Chang gave the switch-module modeling as follows [36]. A switch module is a square block with W terminals on each side. There are two types of switch modules, *switch blocks* and *switch matrices* (see Figure 18). In a switch block (Figure 18(a)), terminals on different sides may be connected by programmable switches, and connecting two terminals requires using exactly one switch. Switch blocks are used for connections among various types of interconnect in an FPGA [34]. A switch matrix (Figure 18(b)) consists of a grid of horizontal and vertical tracks. There are two types of switches in a switch matrix, *crossing switches* and *separating switches*. A crossing switch is used to connect two crossing tracks, and a separating switch can be used to make two terminals of a track electrically disconnected, and thus these two terminals can be used independently. In Figure 18(b), the crossing switches are represented by solid circles and the separating switches by hollow ones. Switch matrices are used in various commercially available FPGAs [1, 2, 20, 34], CPLDs [2, 33], and FPICs [3].

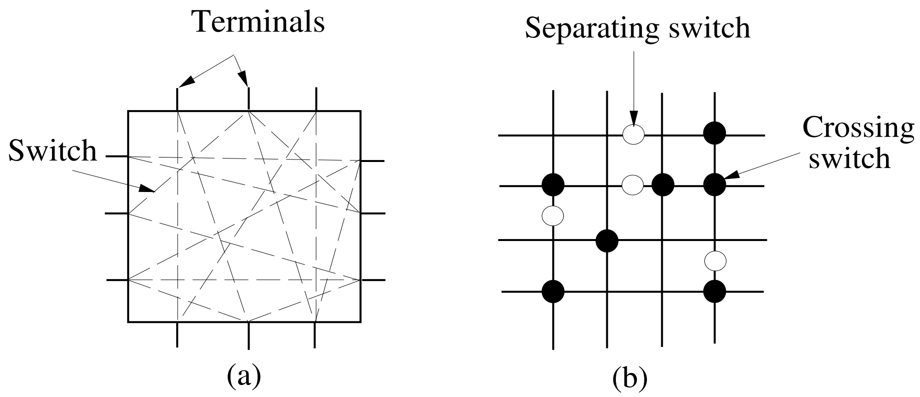


Figure 18: Two types of switch modules. (a) A switch block. (b) A switch matrix.

7.2.2 Switch-Block Architectures

The effects of switch-block architectures on routing were first studied experimentally by Rose and Brown [26]. The work in [26] defines the flexibility of a switch block, represented by F_S, as the number of programming switches between a terminal and others, and the flexibility of a connection module, denoted by F_C, as the number of tracks to which a logic-module pin can be connected; for example, the switch block in Figure 19(a) has $F_S = 3$, and the connection module in Figure 19(b) has $F_C = 2$. These definitions assume that all terminals and logic-module pins have the same degree of connectivity.

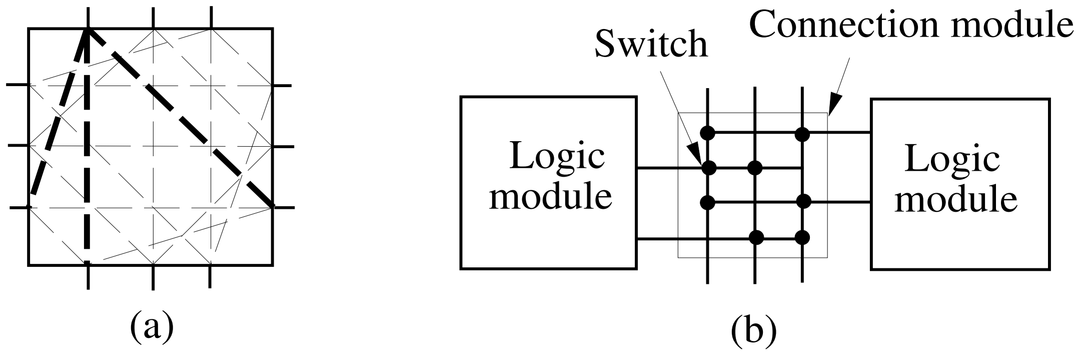


Figure 19: Flexibility. (a) A switch module with $F_S = 3$. (b) A connection module with $F_C = 2$.

Rose and Brown investigated the effects of different switch-block and connection-module flexibilities on routing [26]. The primary conclusions of the study in [26] are that for a high probability of 100% routing completion:

- F_C usually needs to be greater than one-half of the number of tracks in a routing channel.
- A relatively low F_S with as small as three is often sufficient.

These conclusions lead to the architectural choice: $F_S = 3$ combined with a high F_C, which gives the best area and routability trade-off. A theoretical study of flexibility and routability was later presented based on a stochastic model [6], which confirms the experimental results in [26].

Chang, Wong, and Wong further investigated the various topologies of switch blocks of $F_S = 3$ and presented a class of *universal switch blocks* [9]. A switch module M with W terminals on each side is said to be *universal* if every set of nets satisfying the dimensional constraint (i.e., the number of nets on each side of M is at most W) is simultaneously routable through M. They showed that $6W$ switches and $F_S = 3$ are sufficient to construct a "cheapest" universal switch block. They also proved that each of the universal switch blocks can accommodate significantly more routing instances than the Xilinx XC4000-type one of the same size. Figure 20(b) illustrates a routing on a symmetrical, universal switch block for the instance of (a); the same routing instance is not feasible for the XC4000-type switch block (see Figure 20(c)). The work also provides a theoretical insight into the important observation by Rose and Brown [26] that $F_S = 3$ is often sufficient to provide high routability.

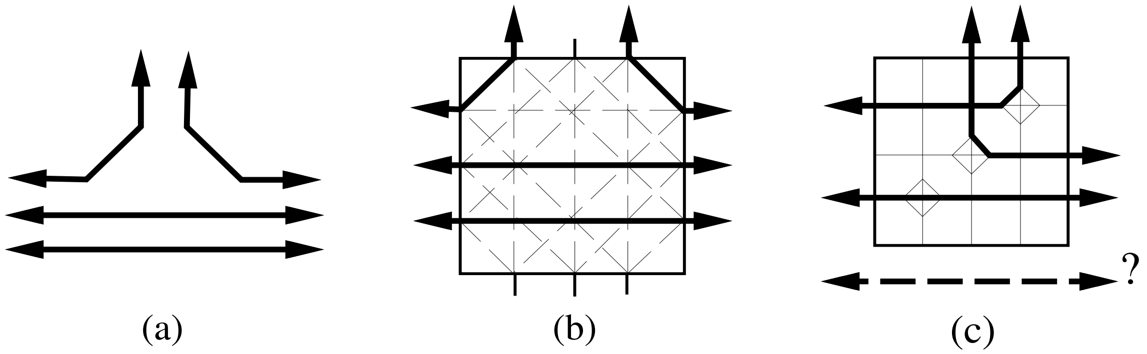


Figure 20: Routing on two types of switch blocks of the same size and both with 18 switches. (a) A routing instance with four nets. (b) A symmetrical, universal switch block on which the routing instance of (a) is routable. (c) A Xilinx XC4000-type switch block on which the routing instance of (a) is not routable. Notice that one straight net can not be routed in the figure shown.

7.2.3 Switch-Matrix Architectures

The switch-matrix architecture was first modeled and investigated by Zhu, Wong, and Chang [36]. They studied the feasibility conditions of switch matrices and presented a heuristic guided by stochastic information for switch-matrix design. Chang, Wong, and Wong later proposed a network-flow-based approach for switch-matrix design [8]; this approach leads to switch matrices with substantially higher routability. Wu and Chang studied the universality of switch *matrices* [37]. They showed that there exist no universal switch matrices; however, they presented a class of "cheapest" *quasi-universal* switch matrices which have asymptotically the same routing capability as universal switch *blocks*.

Sun, Wang, Wong, and Liu [30] considered a variation of the switch-matrix architecture on which "separating switches" are placed outside a switch matrix. According to their study, this class of routing architectures has the advantage of using a smaller number of switches (and thus smaller signal delays) at the expense of consuming a larger number of routing tracks per channel, compared with the devices with switch *blocks*. Based on a stochastic model, they proved that 100% routing completion in that architecture can be achieved with a high probability using a reasonable track count in each channel.

7.2.4 Wiring Segmentation

The segmentation design problem is to arrange segmented tracks in the routing channels to optimize routability and performance simultaneously, preferably comparable to the case in mask programmable gate arrays.

El Gamal *et al.* showed that with appropriate arrangement of segment lengths, the channel width needed for routing completion can be close to that for gate arrays [12, 13]. Zhu and Wong presented an algorithm for the channel segmentation problem based on a stochastic analysis [35]. Given an arbitrary distribution of nets and routing requirements, they computed the number of segmented tracks of various types. Pedram, Nobandegani, and Preas later extended the stochastic approach to an analytical model for designing effective segmented channel architectures [24].

For the segmentation design for a symmetrical array-based programmable logic device, Zhu, Wong, and Chang observed that it can be done in two phases: channel segmentation design followed by switch-module design [36]. In the first phase, a channel segmentation is constructed for each horizontal and vertical channel independently. In the second phase, a switch module is constructed at each channel intersection. Based on the similar idea, Mak and Wong later employed a decomposition procedure and showed in details how the two-phase approach can be done [21]. Chang, Lin, and Wong recently proposed graph matching-based algorithms for segmentation design [38]; the algorithms substantially improve the routability of segmented FPGAs. More significantly, the matching-based formulation can readily extend to higher-dimensional segmentation design, which is crucial to the design of large-scale FPGAs.

8 Cross Reference

1. K. Zhu and D. F. Wong, "CAD for field-programmable gate arrays," in the area *CAD for Integrated Circuits*.

2. S. R. Carlough, "Programmable logic devices," in the area *Solid State Circuits*.

References

[1] Actel Corp., *FPGA Data Book and Design Guide*, 1996.

[2] Altera Corp., *FLEX 10K Handbook*, 1996.

[3] Aptix Inc., *Programmable Interconnect System Data Book*, Nov., 1993.

[4] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field-Programmable Gate Arrays*, Boston, MA: Kluwer Academic Publishers, 1992.

[5] S. D. Brown and J. Rose, FPGA and CPLD architectures: A tutorial, *IEEE Design & Test of Comput.*, pp. 42–57, Summer 1996.

[6] S. D. Brown, J. Rose, and Z. G. Vranesic, A stochastic model to predict the routability of field-programmable gate arrays, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 12 (12), pp. 1827–1838, Dec. 1993.

[7] W. Carter, K. Duong, R. H. Freeman, H. Hsieh, J. Y. Ja, J. E. Mahoney, L. T. Ngo, and S. L. Sze, A user programmable reconfigurable gate array, *Proc. 1986 IEEE Custom Integr. Circuits Conf.*, pp. 233–235, May 1986.

[8] Y.-W. Chang, D. F. Wong, and C. K. Wong, Design and analysis of FPGA/FPIC switch modules, *Proc. IEEE Int. Conf. Comput. Des., VLSI in Comput. and Processors*, pp. 394–401, Austin, TX, Oct. 1995.

[9] Y.-W. Chang, D. F. Wong, and C. K. Wong, Universal switch modules for FPGA design, *ACM Trans. Des. Automation of Electronic Syst.*, 1 (1), pp. 80–101 , Jan. 1996.

[10] X. Chen and S. L. Hurst, Derivation of universal logic module, for $n \geq 3$, by algebraic means, *IEE Proc., Part E*, 128 (5), pp. 205–211, Sep. 1981.

[11] Concurrent Logic, *CFA6006 Field-Programmable Gate Arrays Data Sheet*, Sunnyvale, CA, 1991.

[12] A. El Gamal, J. Greene, J. Reyneri, E. El-Ayat, and A. Mohsen, An architecture for electrically configurable gate arrays, *IEEE J. Solid-State Circuits*, 24 (2), pp. 394–398, Apr. 1989.

[13] A. El Gamal, J. Greene, and V. Roychowdhury, Segmented channel routing is nearly as efficient as channel routing (and just as hard), *Advanced Research in VLSI*, pp. 193–211, UC Santa Cruz, Mar. 1991.

[14] E. Hamdy, J. McCollum, S. Chen, S. Chiang, S. Eltoukhy, J. Chang, T. Speers, and A. Mohsen, Dielectric-based antifuse for logic and memory ICs, *IEEE Int. Electron Devices Meet. Tech. Dig.*, Piscataway, NJ, pp. 786–789, June 1988.

[15] J. He and J. Rose, Advantages of heterogeneous logic block architectures for FPGAs, *Proc. IEEE Custom Integr. Circuits Conf.*, pp. 7.4.1–7.4.5, May 1993.

[16] A. Kaviani and S. D. Brown, Hybrid FPGA architecture, *Proc. ACM Int. Symp. Field Programmable Gate Arrays*, pp. 3–9, Feb. 1996.

[17] J. Kouloheris and A. El Gamal, FPGA performance vs. cell granularity, *Proc. IEEE Custom Integr. Circuits Conf.*, pp. 6.2.1–6.2.4, May 1991.

[18] J. Kouloheris and A. El Gamal, FPGA area versus cell granularity—lookup tables and PLA cells, *Proc. ACM First Int. Workshop on Field Programmable Gate Arrays*, pp. 9–14, Feb. 1992.

[19] C. Lin, M. Marek-Sadowska, and D. Gatlin, Universal logic gate for FPGA design, *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, pp. 164–168, San Jose, CA, Nov. 1994.

[20] Lucent Technologies Inc., *ORCA Series Field-Programmable Gate Arrays*, Aug. 1996.

[21] W.-K. Mak and D. F. Wong, Channel segmentation design for symmetrical FPGAs, *Proc. IEEE Int. Conf. Comput. Des., VLSI in Comput. and Processors*, Oct. 1997.

[22] J. Oldfield and R. Dorf, *Field Programmable Gate Arrays*, New York: John Wiley & Sons, 1995.

[23] F. P. Preparata and D. E. Muller, Generation of near-optimal universal Boolean functions, *J. Comput. and Syst. Sciences*, 4, pp. 93–102, Apr. 1970.

[24] M. Pedram, B. Nobandegani, and B. Preas, Design and analysis of segmented routing channels for row-based FPGA's, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 13 (12), pp. 1470–1479, Dec. 1994.

[25] QuickLogic Corp., *QuickLogic 1996/97*, Santa Clara, 1996.

[26] J. Rose and S. D. Brown, Flexibility of interconnection structures for field-programmable gate arrays, *IEEE J. Solid State Circuits*, 26 (3), pp. 277–282, Mar. 1991.

[27] J. Rose, R. J. Francis, D. Lewis, and P. Chow, Architecture of programmable gate arrays: The effect of logic block functionality on area efficiency, *IEEE J. Solid-State Circuits*, 25 (5), pp. 1217–1225, Oct. 1990.

[28] J. Rose, A. El Gamal, A. Sangiovanni-Vincentelli, Architecture of field-programmable gate arrays, *Proc. of the IEEE*, 81 (7), pp. 1013–1029, July 1993.

[29] S. Singh, J. Rose, D. Lewis, and P. Chow, The effect of logic block architecture on FPGA performance, *IEEE J. Solid State Circuits*, 27 (3), pp. 281–287, Mar. 1992.

[30] Y. Sun, T. -C, Wang, C. K. Wong, and C. L. Liu, Routing for symmetric FPGA's and FPIC's, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 16 (1), pp. 20–31, Jan. 1997.

[31] S. Thakur and D. F. Wong, On designing ULM-based FPGA logic modules, *Proc. ACM Int. Symp. Field Programmable Gate Arrays*, pp. 3–9, Monterey, CA, Feb. 1995.

[32] S. Trimberger (ed.), *Field-Programmable Gate Array Technology*, Boston, MA: Kluwer Academic Publishers, 1994.

[33] AMD/Vantis Inc., *The Mach 5 Family*, March 1996.

[34] Xilinx Inc., *The Programmable Logic Data Book*, 1996.

[35] K. Zhu and D. F. Wong, On channel segmentation design for row-based FPGAs, *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, pp. 26–29, Santa Clara, Nov. 1992.

[36] K. Zhu, D.F. Wong and Y.-W. Chang, Switch module design with application to two-dimensional segmentation design, *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, pp. 481–486, Santa Clara, Nov. 1993.

[37] G.-M. Wu and Y.-W. Chang, Switch-matrix architectire and routing for FPDs, *Proc. ACM Int. Symp. Physical Des.*, pp. 158–163, Monterey, CA, Apr. 1998.

[38] Y.-W. Chang, J.-M. Lin, and D. F. Wong, Graph matching-based algorithms for FPGA segmentation design, to appear in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, San Jose, Nov. 1998.

[39] S. D. Brown and J. Rose, FPGA and CPLD architecture: a tutorial, *IEEE Des. & Test of Comput.*, pp. 42–57, Summer 1996.