Architecture of Field-Programmable Gate Arrays

JONATHAN ROSE, MEMBER, IEEE, ABBAS EL GAMAL, SENIOR MEMBER, IEEE, AND ALBERTO SANGIOVANNI-VINCENTELLI, FELLOW, IEEE

Invited Paper

*A survey of Field-Programmable Gate Array (FPGA) architectures and the **programming technologies** used to customize them is presented. Programming technologies are compared on the basis of their volatility, size, parasitic capacitance, resistance, and process technology complexity. FPGA architectures are divided into two constituents: **logic block** architectures and **routing architectures**. A classification of logic blocks based on their **granularity** is proposed and several logic blocks used in commercially available FPGA's are described. A brief review of recent results on the effect of logic block granularity on logic density and performance of an FPGA is then presented. Several commercial routing architectures are described in the context of a general routing architecture model. Finally, recent results on the tradeoff between the flexibility of an FPGA routing architecture its routability and density are reviewed.*

I. Introduction

The architecture of a field-programmable gate array (FPGA), as illustrated in Fig. 1, is similar to that of a mask-programmable gate array (MPGA), consisting of an array of logic blocks that can be programmably interconnected to realize different designs. The major difference between FPGA's and MPGA's is that an MPGA is programmed using integrated circuit fabrication to form metal interconnections, while an FPGA is programmed via electrically programmable switches much the same as traditional programmable logic devices (PLD's). FPGA's can achieve much higher levels of integration than PLD's, however, due to their more complex routing architectures and logic implementations. PLD routing architectures are very simple but highly ineffiecient crossbar-like structures in which every output is directly connectable to every input through one switch. FPGA routing architectures provide a more efficient MPGA-like routing where each connection typically passes through several switches. In a PDL, logic is implemented using predominantly two-level AND-OR logic with wide input AND gates. In an FPGA logic is implemented using multiple levels of lower fanin gates, which is often much more compact than two-level implementations.

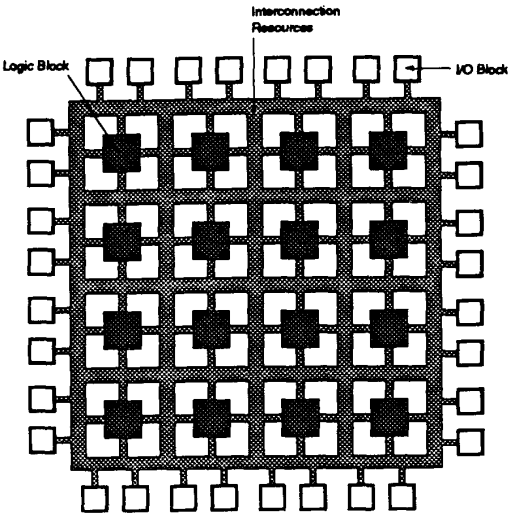


Fig. 1. FPGA architecture.

An FPGA logic block can be as simple as a transistor or as complex as a microprocessor. It is typically capapble of implementing many different combinational and sequential logic functions. Current commercial FPGA's employ logic blocks that are based on one or more of the following:

- Transistor pairs.
- Basic small gates such as two-input NAND's or exclusive-OR's.
- Multiplexers.

Manuscript received October 1, 1992. The work by The second author was partially supported under contract J-FBI-89-101.

J. Rose is with the Department of Electrical Engineering, University of Toronto, 10 King's College Road, Toronto, Ontario M5S 1A4, Canada.

A. El Gamal is with the Depratment of Electrical Engineering, Stanford University, Stanford, CA 94305.

A. Sangiovanni-Vincentelli is with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720.

IEEE Log Number 9210745.

- Look-up tables (LUT's).
- Wide-fanin AND-OR structures.

The routing architecture of an FPGA could be as simple as a nearest neighbor mesh [9] or as complex as the perfect shuffle used in multiprocessors [42]. More typically, an FPGA routing architecture incorporates wire segments of varying lengths which can be interconnected via electrically programmable switches. The choice of the number of wire segments incorporated affects the density achieved by an FPGA. If an inadequate number of segments is used, only a small fraction of the logic blocks can be utilized, resulting in poor FPGA density; conversely the use of an excess number of segments that go unused also wastes area.

The distribution of the lengths of the wire segments also greatly affects the density and performance achieved by an FPGA. For example, if all segments are chosen to be long, implementing local interconnections becomes too costly in area and delay. On the other hand if all segments are short, long interconnections are implemented using too many switches in series, resulting in unacceptably large delays.

Several different programming technologies are used to implement the programmable switches. There are three types of such programmable switch technologies currently in use. These are:

- SRAM, where the switch is a pass transistor controlled by the state of a SRAM bit,
- Antifuse, whci, when electrically programmed, forms a low resistance path, and
- EPROM, where the switch is a floating-gate transistor that can be turned off by injecting charge onto their floating gate.

In all cases, a programmable switch occupies larger area and exhibits much higher parasitic resistance and capacitance than a typical contact or via used in the customization of an MPGA. Additional area is also required for programming circuitry. As a result the density and performance achievable by today's FPGA's are an order of magnitude lower than that for MPGA's manufactured in the same technology.

The adverse effects of the large size and relatively high parasitics of programmable switches can be reduced by careful architectural choices. By choosing the appropriate granularity and functionality of the logic block, and by designing the routing architecture to achieve a high degree of routability while minimizing the number of switches, both density and performance can be optimized. The best architectural choices, however, are highly dependent on the programming technology used as well as on the type of designs implemented, so that no one architecture is likely to be best suited for all programming technologies and for all designs.

The complexity of FPGA's has surpassed the point where manual design is either desirable or feasible. Consequently, the utility of an FPGA architecture is highly dependent on effective automated logic and layout synthesis tools to support it. A complex logic block may be underutilized without an effective logic synthesis tool, and the overall utilization of an FPGA may be low without an effective placement and routing tool.

Commercial FPGA's differ in the type of programming technology used, in the architecture of the logic block and in the structure of their routing architecture. In this paper we survey the architectures of commercially available FPGA's and discuss the dependence of FPGA density and performance on these factors. The paper is organized as follows: Section II describes the most widely used programming technologies. Section III presents a survey of commercial FPGA logic block architectures, classified by their granularity. This includes a summary of recent research results concerning the effect of granularity on overall FPGA density and performance. Section IV describes several commercial routing architectures in the context of a general routing architecture model, and summarizes recent research results in this area. Section V concludes with a discussion of potential future architectural directions for FPGA's.

II. Programming Technologies

An FPGA is programmed using electrically programmable switches. The properties of these programmable switches, such as size, on-resistance, and capacitance, dictate many of the tradeoffs in FPGA architecture. In this section we describe the most commonly used programmable switch technologies and at the end will contrast each technology with respect to volatility, re-programmability, size, series on-resistance, parasitic capacitance, and process technology complexity.

A. SRAM Programming Technology

The SRAM programming technology uses Static RAM cells to control pass gates or multiplexers as illustrated in Fig. 2. It is used in the devices from Xilinx [23], Plessey [33] Algotronix, [2], Concurrent Logic [13] and Toshiba [32].

When a one is stored in the SRAM cell in Fig. 2(a), the pass gate acts as a closed switch, and can be used to make a connection between two wire segments. When a zero is stored, the switch is open and the transistor presents a high resistance between the two wire segments. For the multiplexer, the state of the SRAM cells connected to the select lines controls which one of the multiplexer inputs are connected to the output, as shown in Fig. 2(b).

Since SRAM is volatile, the FPGA must be loaded and configured at the time of chip power-up. This requires external permanent memory to provide the programming bits such as PROM, EPROM, EEPROM or magnetic disk.

A major disadvantage of SRAM programming technology is its large area. It takes at least five transistors to implement an SRAM cell, plus at least one transistor to serve as a programmable switch. However, SRAM programming technology has two major advantages; fast re-programmability and that it requires only standard integrated circuitprocess technology.

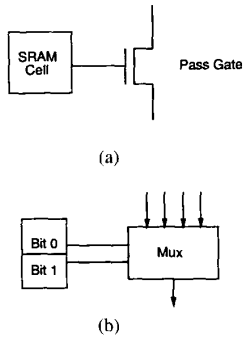


Fig. 2. Static RAM programming technology.

B. Antifuse Programming Technology

An antifuse is a two terminal device with an unprogrammed state presenting a very high resistance between its terminals. When a high voltage (from 11 to 20 volts, depending on the type of antifuse) is applied across its terminals the antifuse will "blow" and create a low-resistance link. This link is permanent. Antifuses in use today are built either using an Oxygen-Nitrogen-Oxygen (ONO) dielectric between N+ diffusion and poly-silicon [19], [15], [1] or amorphous silicon between metal layers [6] or between polysilicon and the first layer of metal [31].

Programming an antifuse requires extra circuitry to deliver the high programming voltage and a relatively high current of 5 mA or more. This is done in [15] through fairly sizable pass transistors to provide addressing to each antifuse. An associated paper in this issue discusses the programming of antifuse structures in more detail [18]. Antifuse technology is used in the FPGA's from Actel [15] [1], Quicklogic [6], and Crosspoint [31].

A major advantage of the antifuse is its small size, little more than the cross-section of two metal wires. This advantage is somewhat reduced by the large size of the necessary programming transistors, which must be able to handle large currents, and the inclusion of isolation transistors that are sometimes needed to protect low voltage transistors from high programming voltages. A second major advantage of an antifuse is its relatively low series resistance. The on-resistance of the ONO antifuse is 300 to 500 ohms [19], while the amorphous silicon antifuse is 50 to 100 ohms [6] [31]. Additionally, the parasitic capacitance of an unprogrammed amorphous antifuse is significantly lower than for other programming technologies.

C. Floating Gate Programming Technology

The floating gate programming technology uses technology found in ultraviolet erasable EPROM and electrically erasable EEPROM devices. The EPROM-based approach is used in devices from Altera [43] and Plus Logic [34].

The programmable switch, illustrated in Fig. 3, is a transistor that can be permanently "disabled." This is accomplished by injecting a charge on the floating gate (gate 2 in the figure) using a high voltage between the control gate 1 and the drain of the transistor. This charge increases

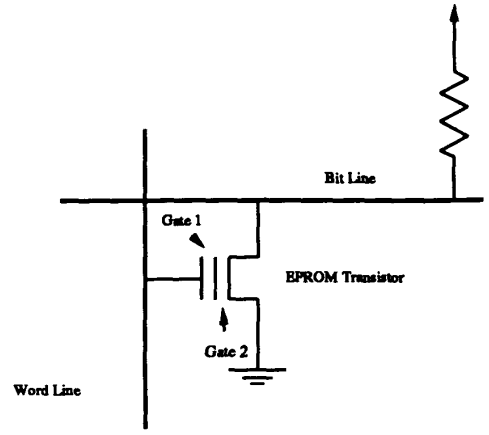


Fig. 3. Floating gate programming technology.

the threshold voltage of the transistor so that it turns off. The charge is removed by exposing the floating gate to UV light. This lowers the threshold voltage of the transistor and makes the transistor function normally.

Rather than using an EPROM transistor directly as a programmable switch, the unprogrammed transistor is used to pull down a "bit line" when the "word line" is set high, as illustrated in Fig. 3. While this approach can be simply used to make a connection between the word and bit lines, it can also be used to implement a wired-AND style of logic, thereby providing both logic and routing.

As with the SRAM programming technology, a major advantage of the EPROM technology is its reprogrammability. An advantage over SRAM, though, is that no external permanent memory is needed to program the chip on power-up. EPROM technology, however, requires three additional processing steps over an ordinary CMOS process. Two other disadvantages are the high ON-resistance of an EPROM transistor (about twice that of a similarly sized NMOS pass transistor) and the high static power consumption due to the pull-up resistor used (see Fig. 3).

The EEPROM-based programming technology is used in the devices from AMD [3] and Lattice [4]. It is similar to the EPROM approach, except that removal of the gate charge can be done electrically, in-circuit, without UV light. This gives the added advantage of easy reprogrammability, which can be very helpful in some applications such as hardware updates to equipment in remote locations. An EEPROM cell, however, is roughly twice the size of an EPROM cell.

E. Summary of Programming Technologies

Table 1 lists the properties of each programming technology. All data assumes a 1.2 μm CMOS process technology. The first column gives the name of the technology. Note that there is separate information for the two different types of antifuse. The second column indicates if the configuration is lost when power is removed from the

Table 1 Comparison of Programming Technologies

Technology and Process	Volatile?	Re-Prog?	Area	R (ohm) (on switch)	C (fF) (parasitic)	# Extra Fab Steps
SRAM Mux Pass Transistor 1.2 μm CMOS	Yes	Yes in Circuit	Large	0.5 - 2 k	10 - 20 fF	0
ONO Anti-fuse 1.2 μm CMOS	No	No	Fuse small (via) Prog. Tran. Large	300 - 500	5 fF	3
Amorphous Anti-fuse 1.2 μm CMOS	No	No	Fuse small (via) Prog. Tran. Large	50 - 100	1.1 - 1.3 fF	3
EPROM 1.2 μm CMOS	No	Yes out of circuit	Small in array	2 - 4 k	10 - 20 fF	3
EEPROM 1.2 μm CMOS	No	Yes in circuit	2x EPROM	2 - 4 k	10 - 20 fF	>5

device. The third column indicates if the technology permits reprogramming. The fourth column provides the relative size of the programmable switch. The fifth column gives the series resistance of an "on" switch, and the sixth column gives the parasitic capacitance of an "off" switch, *not* including any capacitance due to associated wiring or programming transistors. For reference, the capacitance of a 10 μm length of minimum-width wire in a 1.2 μm CMOS process is about 0.6 fF. The seventh column gives the number of additional processing steps required beyond standard CMOS.

III. Logic BlockArchitecture

In this section we survey the commercial FPGA logic block architectures in use today. In the first section we discuss the combinational logic portion of the logic block. A discussion of the sequential logic portion is deferred to Section III-D. In Section III-E, we present several recent research results on the effect of the choice of the logic block on the density and performance of an FPGA.

A. Survey of Commercial Logic Block Architectures

FPGA logic blocks differ greatly in their size and implementation capability. The two transistor logic block used in the Crosspoint FPGA can only implement an inverter but is very small in size, while the look-up table logic block used in the Xilinx 3000 series FPGA can implement any five-input logic function but is significantly larger. To capture these differences we classify logic blocks by their *granularity*. Granularity can be defined in various ways, for example, as the number of boolean functions that the logic block can implement, the number of equivalent two-input NAND gates, the total number of transistors, total normalized area, or the number of inputs and outputs. The matter is further confused because in some architectures, such as the Altera FPGA [43] or the AMD FPGA [3], the logic and routing are tightly intertwined and it is difficult to separate their contributions to the architecture. For these reasons, we choose to classify the commercial logic blocks into just two categories: *fine-grain* and *coarse-grain*.

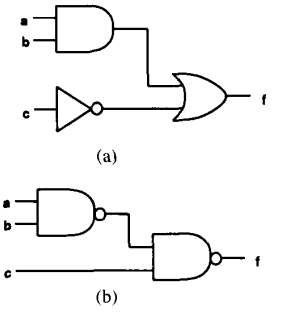


Fig. 4. Example logic function and two-input NAND gate implementation.

For all the logic blocks described below, we show how to implement the logic function $f = ab + \bar{c}$, as illustrated in Fig. 4(a). Note that this is equivalent to the two-input NAND gate implementation given in Fig. 4(b).

B. Fine-Grain Logic Blocks

Fine-grain logic blocks closely resemble MPGA basic cells. The most fine grain logic block would be identical to a basic cell of an MPGA and would consist of few transistors that can be programmably interconnected.

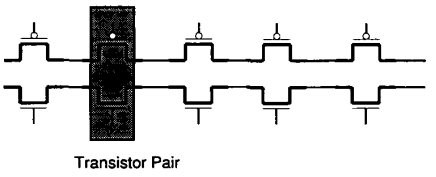


Fig. 5. Transistor pair tiles in cross-point FPGA.

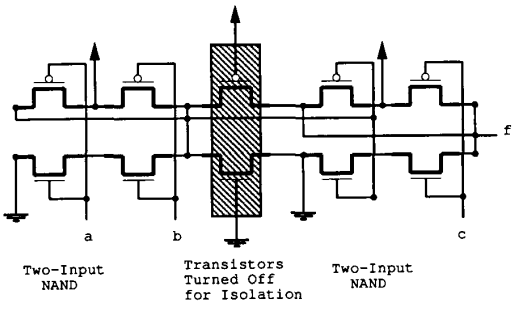


Fig. 6. Programmed cross-point FPGA for logic function $f = ab + \bar{c}$.

1) The Crosspoint FPGA: The FPGA from Crosspoint Solutions [31] uses a single transistor pair in the logic block, as illustrated in Fig. 5.

Figure 6 illustrates how the function of Fig. 4(b) is implemented with the transistor pair tiles of the cross-point FPGA. Since the transistors are connected together in rows, the two two-input NAND gates are isolated by turning off the pair of transistors between the gates.

In addition to the transistor pair tiles, the cross-point FPGA has a second type of logic block, called a RAM logic tile, that is tuned for the implementation of random access memory, but can also be used to build random logic functions in a manner similar to the Actel and The Quicklogic logic blocks described below.

2) The Plessey FPGA: A second example of a fine-grain FPGA architecture is the FPGA from Plessey [33]. Here the basic block is a two-input NAND gate as illustrated in Fig. 7. Logic is formed in the usual way by connecting the NAND gates to implement the desired function. The logic function $f = ab + \bar{c}$ illustrated in Fig. 4(a) is implemented exactly as shown in Fig. 4(b). If the latch in Fig. 7 is not needed, then the configuration store is set to make the latch permanently transparent.

Several other commercial FPGA's employ fine grain blocks. Algotronix [2] uses a two-input function block which can perform any function of two inputs. This is implemented using a configurable set of multiplexers. The logic block of Concurrent Logic's FPGA [13] contains a two-input AND gate and a two-input EXCLUSIVE-OR gate. The FPGA recently discussed by Toshiba in [32] also uses a two-input NAND gate.

The main advantage of using fine grain logic blocks is that the useable blocks[1] are fully utilized. This is because

[1] In all FPGA's, as well as in all MPGA's, only a fraction of the logic blocks available can be utilized in any design.

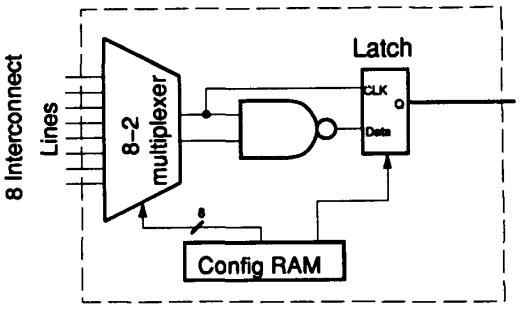


Fig. 7. The Plessey logic block.

it is easier to use small logic gates efficiently and the logic synthesis techniques for such blocks are very similar to those for conventional mask-programmed gate arrays and standard cells.

The main disadvantage of fine-grain blocks is that they require a relatively large number of wire segments and programmable switches. Such routing resources are costly in delay and area. As a result, FPGA's employing fine-grain blocks are in general slower and achieve lower densities than those employing coarse grain blocks. See Section III-A for results supporting this claim.

C. Coarse-Grain Logic Blocks

1) The Actel Logic Block: The Actel logic block [15], [1] is based on the ability of a multiplexer to implement different logic functions by connecting each of its inputs to a constant or to a signal [46]. For example, consider a two-to-one multiplexer with selector input s, inputs a and b and output $f = sa + \bar{s}b$. By setting signal b to logic 0, the multiplexer can implement the AND function $f = sa$. Setting signal a to logic 1 provides the OR function $f = s + b$. By connecting together a number of multiplexers and basic logic gates, a logic block can be constructed which can implement a large number of functions in this manner.

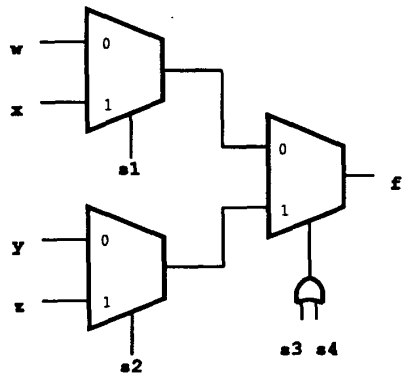
The Actel Act-1 logic block [15] is illustrated in Fig. 8(a). It consists of three multiplexers and one logic gate, has a total of 8 inputs and one output, and implements the function

$$f = \overline{(s_3 + s_4)}\,(\overline{s_1}w + s_1x) + (s_3 + s_4)(\overline{s_2}y + s_2z).$$

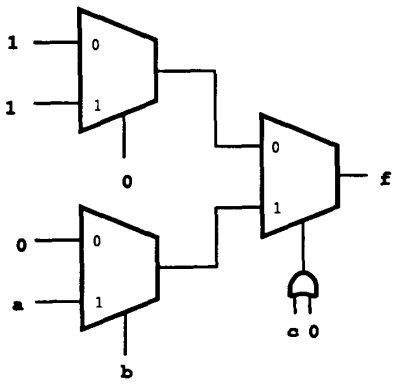
By setting each variable to an input signal, or to a constant, 702 logic functions can be realized. For example, the logic function $f = ab + \bar{c}$ is realized by setting the variables as shown in Figure 8b: $w = 1$, $x = 1$, $s_1 = 0$, $y = 0$, $z = a$, $s_2 = b$, $s_3 = c$, and $s_4 = 0$.

The Act-2 logic block [1] is similar to Act-1, except that the separate multiplexers on the first row are joined and connected to a two-input AND gate, as shown in Fig. 9. The Act-2 combinational logic module can implement 766 functions.

2) Quicklogic Logic Block: The logic block in the FPGA from QuickLogic [6] is similar to the Actel logic blocks in that it employs a four to one multiplexer. Each input of the



(a)



(b)

Fig. 8. The Actel Act-1 logic block.

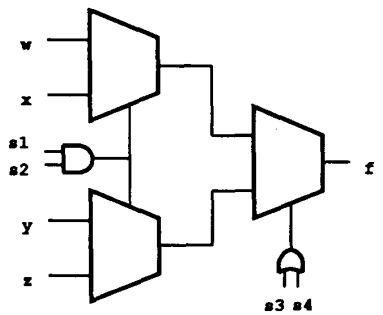


Fig. 9. The Actel Act-2 logic block.

multiplexer (not just the select inputs) is fed by an AND gate, as illustrated in Fig. 10. Note that alternating inputs to the AND gates are inverted. This allows input signals to be passed in true or complement form, thus eliminating the need to use extra logic blocks to perform simple inversions.

Multiplexer-based logic blocks have the advantage of providing a large degree of functionality for a relatively small number of transistors. This is, however, achieved at the expense of a large number of inputs (eight in the case of Actel and 14 in the case of QuickLogic), which when utilized place high demands on the routing resources. Such blocks are, therefore, more suited to FPGA's that use

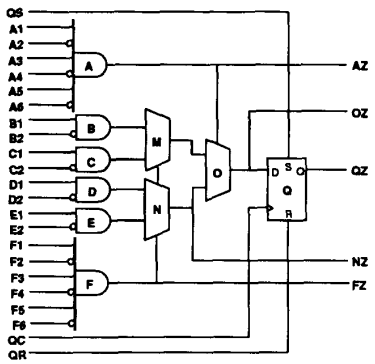
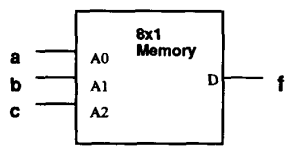


Fig. 10. The Quicklogic logic block.

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

(a)



(b)

Fig. 11. Lookup table-based logic.

programmable switches of small size such as antifuses.

3) The Xilinx Logic Block: The basis for the Xilinx logic block is an SRAM functioning as a look-up table (LUT). The truth table for a K-input logic function is stored in a $2^K \times 1$ SRAM. The address lines of the SRAM function as inputs and the output of the SRAM provides the value of the logic function. For example, consider the truth table of the logic function $f = ab + \bar{c}$ given in Fig. 11(a). If this logic function is implemented using a three-input LUT, then the SRAM would have a 1 stored at address 000, a 0 at 001 and so on, as specified by the truth table.

The advantage of look-up tables is that they exhibit high functionality—a K-input LUT can implement *any* function of K inputs and there are 2^{2^K} such functions. The disadvantage is that they are unacceptably large for more than about five inputs, since the number of memory cells needed for a K-input lookup table is 2^K. While the number of functions that can be implemented increases very fast, these additional functions are not commonly used in logic designs and are also difficult to exploit for a logic synthesis tool. Hence it is often the case that a large LUT will be largely underutilized.

The Xilinx 3000 series logic block [21] [2a] contains a five-input one-output LUT, as illustrated in Fig. 12. This block can be reconfigured into two four-input LUTs, with

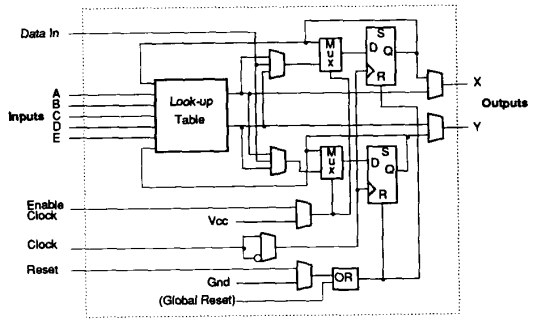


Fig. 12. The Xilinx 3000 logic block.

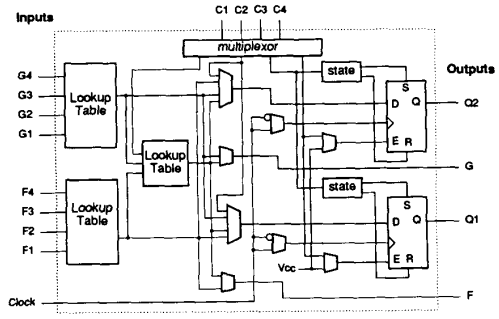


Fig. 13. The Xilinx 4000 logic block.

the constraint that together they use a total of no more than five distinct inputs. This reconfigurability provides flexibility that translates into better logic block utilization because many common logic functions do not require as many as five inputs. The block also contains sequential logic and several multiplexers that connect the combinational inputs and outputs to the flip-flops or outputs. These multiplexers are controlled by the SRAM cells that are loaded at programming time.

The Xilinx 4000 series logic block [23] contains two four-input LUT's feeding into a three-input LUT as illustrated in Fig. 13. In this block, all of the inputs are distinct and available external to the logic block. This block introduces two significant architectural changes from the 3000 series block. First, two differently sized LUT's are used: a four input LUT and a three input LUT, giving the complete block a heterogenous flavor. In general, heterogeneity allows for a better tradeoff between performance and logic density.

The second architectural change in the Xilinx 4000 logic block is the use of two nonprogrammable connections from the two four-input LUT's to the three-input LUT. These connections are significantly faster than any programmable interconnection since no programmable switches are used in series, and little is present in parallel. If proper use can be made of these fast connections FPGA performance can be greatly improved. There is a penalty for this type of connection, however; since the connection is permanent, the inflexibility means that the three-input LUT may often go unused, reducing the overall logic density.

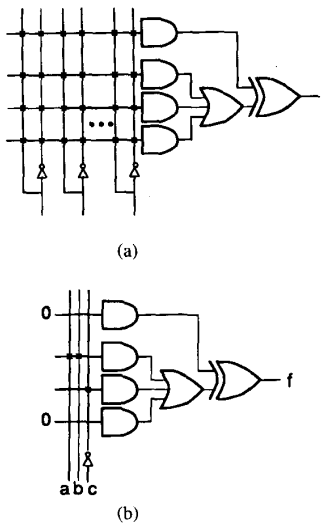


Fig. 14. The Altera 5000 Series logic block.

The Xilinx 4000 block incorporates several additional features. Each LUT can be used directly as an SRAM block. This allows small amounts of memory to be more efficiently implemented. Another feature is the inclusion of circuitry that can be used to implement fast carry addition circuits.

4) The Altera Logic Block: The architecture of the Altera FPGA [43] has evolved from the PLA-based architecture of traditional PLDs [28] with its logic block consisting of wide fanin (20 to over 100 inputs) AND gates feeding into an OR gate with three to eight inputs. Figure 14a illustrates the Altera MAX 5000 series logic block. Using the floating gate transistor-based programmable switch presented in Section II–C, any vertical wire passing by an AND gate can be connected as an input to the gate. The three product terms are then OR's together and can be programmably inverted by an exclusive OR gate, which can also be used to implement other arithmetic functions. Notice that each input signal is provided in both true and complement form, with two separate wires. This *programmable inversion* significantly increases the functional capability of the block.

Figure 14(b) illustrates the implementation of the logic function $f = ab + \bar{c}$. The x's in the figure indicate the wired-AND connections described in Section II–C.

The advantage of this type of block is that the wide AND gate can be used to form logic functions with few levels of logic blocks, reducing the need for programmable interconnect. It is difficult, however, to make efficient use of all of the inputs to all of the gates, resulting in loss of density. This loss is not as severe as it first appears because of the high packing density of the wired-AND gates, as well as the fact that logic connections also serve as the routing function. In other architectures where logic and routing are separate such unused inputs would incur a high penalty.

A disadvantage of the wired-AND configuration is the use of pull-up devices that consume static power. An array full of these pull-ups will consume significant amount of power. To mitigate this, each gate in the MAX 7000 series

block can be programmed to consume about 60% less power but at the expense of about 40% increase in delay [44]. This feature can be used in noncritical paths to reduce power consumption.

In addition to the wide AND-OR logic block, the MAX 5000 employs one other type of logic block, called a *logic expander*. This is a wide-input NAND gate whose output can be connected to the AND-OR logic block. While a logic expander incurs the same delay as the logic block, it takes up less area and can be used to increase its effective number of product terms.

The Altera MAX 7000 logic block [44] is similar to the MAX 5000 except that it provides two more product terms and has more flexibility because neighboring blocks can "borrow" product terms from each other. This is accomplished using a small routing structure between the AND and OR gates called the product term select matrix.

Several other FPGA's use the wide AND-OR style of logic block, including those produced by Plus Logic [34], AMD [3], and Lattice [4]. The device in [34] employs other logic types in combination with the wide AND-OR gate.

D. Sequential Logic

Most of the logic blocks described above include some form of sequential logic. The Xilinx devices [22, 23] have two D flip-flops that can be programmably connected to the outputs of the two lookup tables. The Altera device [43] has one flip-flop per logic block. In the Act-1 device from Actel [15], the sequential logic is not explicitly present and so must be formed using programmable routing and the purely combinational logic blocks. In the Act-2 device [1], there are two alternating types of logic block: the C-module which is the purely combinational block described in Section III–C1), and the S-module which has similar combinational functionality to the C-module but includes a D flip-flop.

The Plessey logic block [33] also incorporates one D latch. It thus requires two blocks to make a master-slave flip-flop. The Algotronix logic block [2] forms sequential logic using feedback around the basic combinational logic module.

E. Effect of Logic Block Granularity on FPGA Density and Performance

In recent years research efforts have been directed at determining choices for FPGA logic blocks that optimize density and performance [35], [36], [37], [39], [24], [40], [20], [26], [41]. In this section we briefly survey this research. For a more complete survey see [8]. The section is divided into two parts: the first deals with the effect of logic block granularity on FPGA density, while the second covers the effect of granularity on performance.

1) Effect of Granularity on Density As the granularity of a logic block increases, the number of blocks needed to implement a design should decrease. On the other hand a more functional (larger granularity) logic block requires more circuitry to implement it, and therefore occupies more

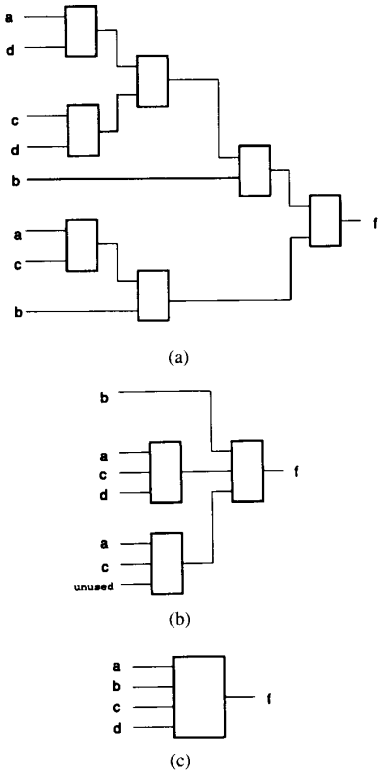


Fig. 15. Three implementations of $f = abd + bc\bar{d} + \bar{a}\bar{b}\bar{c}$.

area. This tradeoff suggests the existence of an "optimal" logic block granularity for which the FPGA area devoted to logic implementation is minimized. While this argument for logic area is straightforward, the effect of granularity on routing area is not as simple, and has a significant impact on the overall FPGA routing area, as discussed below.

As an example of the effect of block functionality on logic area, consider the implementation of the logic function $f = abd + bc\bar{d} + \bar{a}\bar{b}\bar{c}$ using logic blocks of three different granularities as illustrated in Fig. 15. The three logic blocks are a 2-input lookup table (denoted 2-LUT), a 3-LUT, and a 4-LUT. As shown, the 2-LUT implementation requires seven logic blocks, the 3-LUT needs three blocks, and the 4-LUT only one. As an area measure, consider the number of memory bits required to implement this logic function using a K-LUT. Since each K-LUT requires 2^K bits, the 2-LUT implementation requires a total of 28 bit, the 3-LUT needs 24 bits and the 4-LUT needs just 16. Using this area measure the 4-LUT achieves the minimum logic area.

To date, all research aimed at determining the logic block granularity that minimizes the overall area of an FPGA has been experimental. A variety of benchmark designs are mapped into FPGA's with different logic block granularities and the total logic block area as well as routing area are determined for each mapping. Results are then averaged and compared.

Figure 16 gives an example of such experimental results from [36]. It plots, for one benchmark design, both the

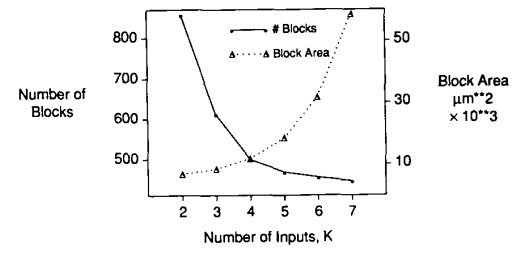


Fig. 16. Number of blocks and block area, for one circuit.

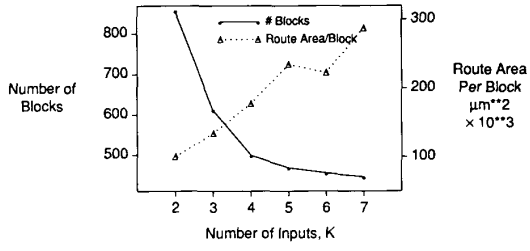


Fig. 17. Number of blocks and routing area/block, for one design.

number of K-LUT blocks needed to implement that design as well as the size of one K-LUT, versus K. These results assume the use of an SRAM programming technology and a 1.2 μm CMOS process. The number of logic blocks decreases rapidly as K increases, while the block size increases exponentially in K. The total *logic block area* (the product of the two curves) achieves a minimum at K=4. The total logic block area minimum exhibits a weak dependence on the size of the programmable switch as discussed in [36], [25].

Active logic area is only part of the total area. The area for routing is usually larger than the active area, particularly in FPGA's, representing from 70 to 90% of the total area. Figure 17 plots the routing area per logic block and the number of logic blocks used versus K for the same benchmark design. While the number of wires decreases as K increases, experimental results show a contrary effect: as K increases the routing area per block increases, which is not only consistent with the results of other experimental studies [27], but also predicted by theoretical studies [14]. The total routing area, obtained by the product of the two curves in Fig. 17, again exhibits a minimum at K=4 as concluded in [36].

The total chip area needed for an FPGA is the sum of the logic block area plus the routing area. This can be simply calculated by multiplying the curves in Figs. 16 and 17 and summing them. The normalized and averaged results for 12 benchmark designs are plotted in Fig. 18.

As the figure shows, total normalized area is minimized for K=4. Figure 18 uses a programming technology size equivalent to a static RAM cell, but these results appear to be only weakly dependent on the size of the programmable switch [36]. These results also appear to be insensitive to the tools used for logic synthesis and layout.

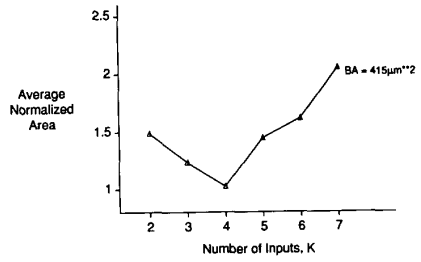


Fig. 18. Average normalized total area versus K for a K-LUT.

In [25], [26] Kouloheris and El Gamal investigated granularity versus density for K-input M-output LUT's [25] and K-input M-output N-product term programmable logic arrays. For K-input M-output LUT's it was shown that:

- A 4-input 1-output lookup table yields the minimum total area of any K-input M-output lookup table logic block for a wide range of programming technologies and routing pitches.
- The best K for area is determined largely by the ratio of memory bit area to the fixed overhead area and exhibits little dependence on the switch area.

For K-input M-output N-product term programmable logic array blocks it was shown that:

- A PLA with eight–10 inputs, three–four outputs, and 12–13 product terms logic block gave the smallest overall FPGA area.
- Assuming the same programming technology, the overall FPGA area using the best PLA logic block is comparable to that using a four-input one-output lookup table.

One other study has investigated another dimension of logic block architecture. Hill and Woo [20] observed that, as in the Xilinx 2000 and 3000 blocks, any K-LUT can be implemented by two (K-1)-LUT's connected by a two-to-one multiplexer, where the Kth input is the input to the multiplexer. If the outputs of the smaller LUT's are made accessible, then the entire block can be used as either two (K-1)-LUT's or one K-LUT. Hill and Woo investigate the benefits and drawbacks of this kind of flexibility.

2) Effect of Granularity on Performance: The granularity of the logic block has a significant effect on the performance of an FPGA. For example, Fig. 19(a) gives the implementation of the logic function $f = ab\bar{d} + abc + ac\bar{d}$ using two-input NAND gate logic blocks. The longest path requires four levels of blocks. Figure 19(b) shows an implementation of the same function using three-input lookup tables requiring only two levels of blocks. Assuming a 1.2μm CMOS technology, a two-input NAND gate delay is estimated at 0.7ns while a three-input LUT delay is estimated at 1.4 ns. Clearly, for a nonzero routing delay between the blocks, the higher granularity of the 3-LUT will result in a faster implementation.

The effect of logic block granularity on FPGA performance has also been studied using an experimental

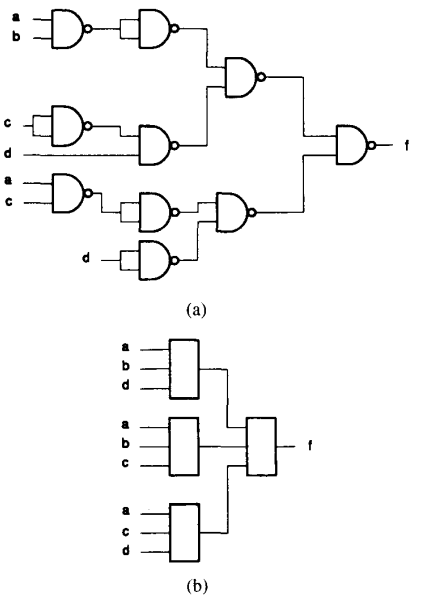


Fig. 19. Two implementations of the same logic function.

approach [24], [39], [40], and [41]. Figure 20 plots both the average number of logic block levels in the critical path and the logic block delay for an FPGA employing a K-LUT versus K. The curve was obtained by normalizing the results for each of 20 benchmarks of different complexities to the K=2 case and then averaging the normalized data over the benchmarks. This figure corroborates the tradeoff illustrated by the example in Fig. 19; as the logic block granularity increases, the number of levels of logic in the critical path decreases, while the delay of the logic block increases.

The routing delay per stage is also an increasing function of K as illustrated in Fig. 21 for the following reasons: Firstly, the average fanout is increasing, and secondly, the number of switches loading each wire is increasing because there are more pins per logic block. Finally, the wires increase in length as the logic block grows in size.

The tradeoff between the decrease in the number of levels of logic with increasing K and the increase in block and routing delay determines the granularity for "optimal" FPGA performance. Figure 22 is a plot of normalized critical path delay versus number of inputs to the lookup table. The figure gives several curves for different values of RC delay in the programmable switch, T_s. If the RC delay is relatively small the best granularity is relatively small, around K=3 or 4. On the other hand, if the RC delay is larger the best granularity is larger, around 6 or 7. Figure 23 plots the granularity which achieves the best performance as a function of the programmable switch time constant T_s = RC. (See Table 1 for typical values of R and C for various programming technologies.)

For more details on these results see [24], [41]. In addition, [41] investigates the delay properties of several other types of block, including NAND gates, multiplexers and wide AND-OR gates.

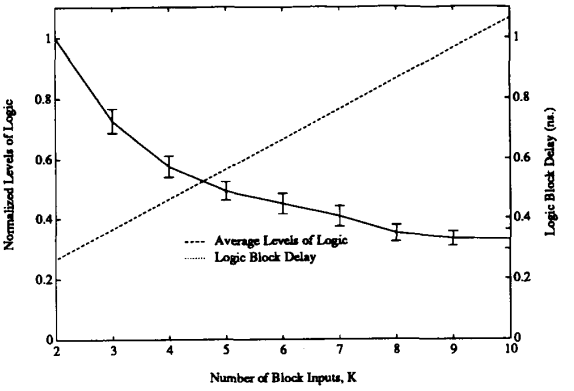


Fig. 20. Average number of logic block levels and block delay for K-LUT's.

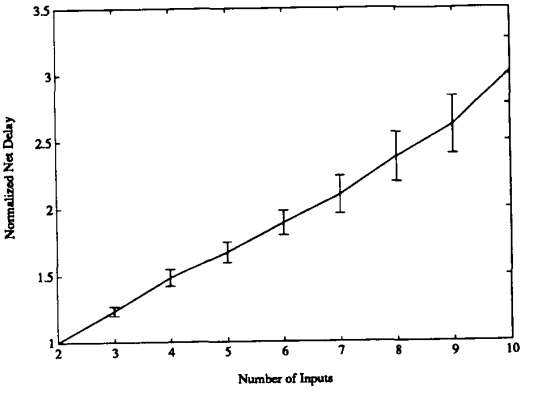


Fig. 21. Net delay versus K, number of inputs to lookup table.

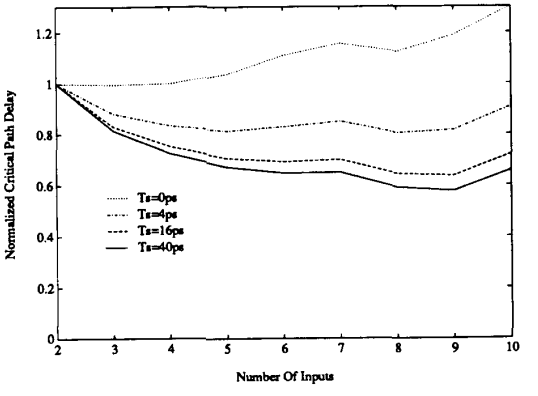


Fig. 22. Critical path delay versus K, number of inputs.

IV. Routing Architecture

The routing architecture of an FPGA is the manner in which the programmable switches and wiring segments are positioned to allow the programmable interconnection of the logic blocks. In this section we survey several commercial routing architectures and present a summary of recent research results on the tradeoff between the flexibility of a routing architecture and FPGA routability and density.

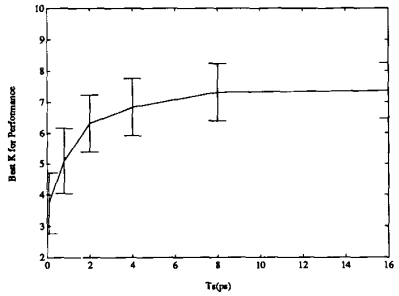


Fig. 23. Best performance K versus T_s.

A. Survey of Commercial Routing Architectures

Figure 24 illustrates a routing architecture model which we use to describe several commercial FPGA routing architectures. Before proceeding, a few definitions are necessary.

A *wire segment* is a wire unbroken by programmable switches. One or more switches may attach to the wire segment. Each end of a wire segment typically has a switch attached.

A *track* is a sequence of one or more wire segments in a line.

A *routing channel* is a group of parallel tracks, as illustrated in Figure 24.

As shown in Fig. 24, the model contains two basic structures. The first is the *connection block* which appears in all architectures. A connection block provides connectivity from the inputs and outputs of a logic block to the wire segments in the channels. Although not shown in the figure, there can be connection blocks in the vertical direction as well as in the horizontal direction. The second structure is the *switch block*, which provides connectivity between the horizontal as well as vertical wire segments. In Fig. 24, the switch block provides connectivity among the wire segments incident to its four sides. In some architectures, the switch block and connection block are intermingled, and in others they are combined into a single structure.

The following four sections describe four commercial FPGA routing architectures beginning with the context of the general model of Fig. 24, then proceeding to more unique features.

1) The Xilinx Routing Architecture: Figure 25 illustrates the routing architecture used in the Xilinx 3000 series FPGA [21] [22]. Connections are made from the logic block into the channel through a connection block. Since each connection site is large because of the SRAM programming technology, the Xilinx 3000 connection block typically connects each pin to only two or three out of the five tracks passing by a block as the expanded figure in the upper left corner of Fig. 25 illustrates. On all four sides of the logic block there are connection blocks that connect a total of 11 different logic block pins to the wire segments. The connections are implemented with pass transistors for the output pins and multiplexers for the input pins. The use of multiplexers reduces the number of SRAM cells needed per pin.

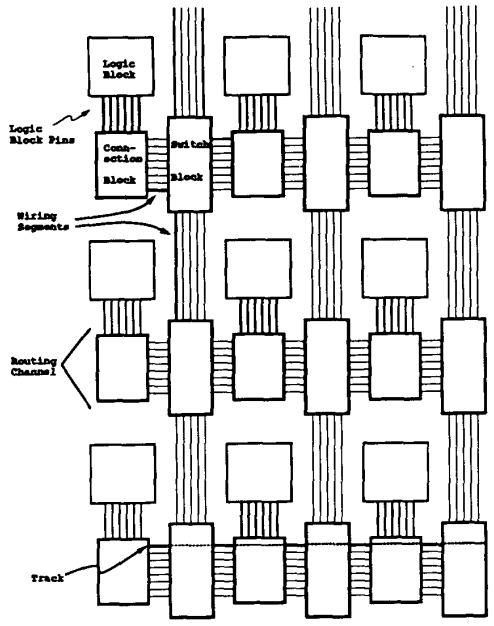


Fig. 24. General FPGA routing architecture.

Once the logic block pin is connected via the connection block, the switch block makes connections between segments in intersecting horizontal and vertical channels. As the expanded picture in the lower right-hand corner shows, each wire segment can connect to a subset of the wire segments on opposing sides of the block. Each wire segment can typically connect to five or six out of a possible 15 wire segments on the opposing sides. Again, this number is limited by the large size and capacitance of the SRAM programmable switches.

There are four types of wire segments provided in the Xilinx 3000 architecture:

- General-purpose interconnect consisting of wire segments that pass through switches in the switch block.
- Direct interconnect consisting of wire segments that connect each logic block output directly to four nearest neighbors as illustrated by the thick black lines emanating from the center block in Fig. 25.
- Long lines, which span the length or width of the chip, providing high-fanout uniform delay connections, indicated by the dashed lines in Fig. 25.
- A clock line, which is a single net that spans the entire chip and is driven by a high-drive buffer. This line is connected only to the clock inputs of the flip-flops, and provides for a low-skew clocking scheme.

The Xilinx 4000 series architecture [23] is similar to the 3000 series, with the key architectural differences being that there are many more general purpose tracks per channel—18 versus five. Also, connectivity between the logic block pins and the tracks is much higher as each logic block pin connects to almost all of the tracks. Finally, four of the tracks pass through a switch only every second switch block. These double-length lines are depicted in Fig. 26.

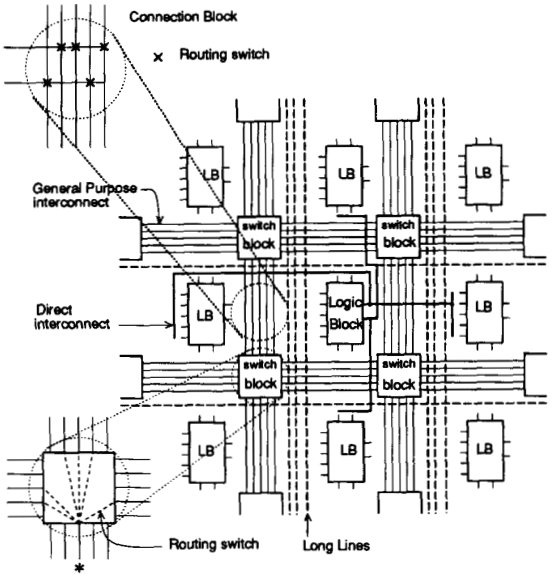


Fig. 25. Xilinx 3000 routing architecture.

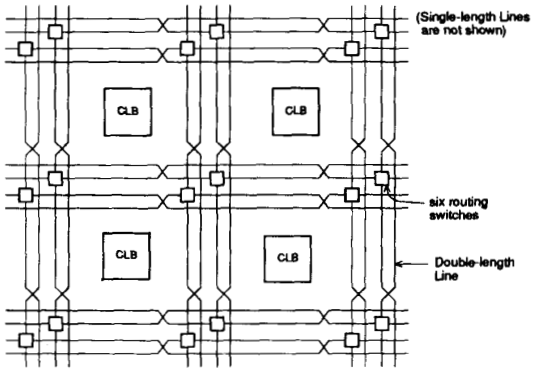


Fig. 26. Xilinx 4000 segments of length 2.

The advantage of longer wire segments is that the signal passes through less series resistance in travelling the same distance compared to wire segments of length one. In addition, since there are fewer switches and associated programming cells the FPGA logic density is superior.

2) The Actel Routing Architecture: Figure 27 illustrates the general Actel routing architecture [15]. The routing architecture is asymmetric because there are more uncommitted general purpose tracks in the horizontal direction than the vertical. While mask-programmed gate arrays and standard cell layout styles have always had this kind of directional bias, to our knowledge there is no definitive result that determines whether a biased approach is better or worse than an unbiased one.

The connection block (or routing channel) of the Actel routing architecture is shown in the middle of Fig. 27. The connectivity is different for input pins and output pins. For input pins, each pin can connect to *all* of the tracks in the channel that are on the same side as the pin. The output pins extend across two channels above the logic block and

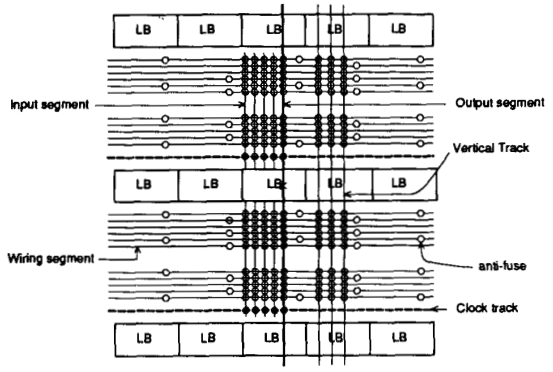


Fig. 27. Actel FPGA routing architecture.

two channels below it. An output pin can connect to every track in all four channels that it crosses.

There is no clearly separable switch block in the Actel architecture. Instead, the switching is distributed throughout the horizontal channels. Depending on the direction of the connection, different degrees of connectivity are available. All vertical tracks can make a connection with every incident horizontal track. Note that this provides a great deal more connectivity than the switch block in the Xilinx architectures. This flexibility allows the routing of the horizontal channels to be performed independently, because for a given route that travels through two channels, the choice of a track in one channel has no effect on the number of choices available in another channel. By contrast, in the Xilinx approach, the choice of a particular track in one channel severely limits the choice of tracks in subsequent channels. This flexibility simplifies the routing problem. The drawback is that more switches are needed which contribute extra capacitive loading.

Each horizontal channel consists of 22 routing tracks, and each track is broken up into segments of different lengths. Segments vary in length from two logic blocks long (allowing connections between just two blocks) to segments that are equal in length to the entire track. This wide distribution of segment lengths makes it likely that a segment of the exact or close length of any given connection can be found, so that very few series programmable switches are needed in any intra-channel connection.

As mentioned above, there are fewer uncommitted vertical segments than horizontal segments. There are three types of vertical segments. In addition to the input segments and output segments already described, there are uncommitted vertical *freeways* that either travel the entire height of the chip, or some significant portion of it. There is one freeway per logic block. This allows signals to travel longer vertical distances than permitted by the output segments.

The routing architecture of the Crosspoint FPGA [31] is similar to that of Actel. The Quicklogic architecture [6], which also uses antifuses, is again similar except that the segments are of two classes: short tracks of length one, and long tracks that traverse the entire chip.

3) The Altera Routing Architecture: The routing architecture of the Altera FPGA is illustrated in Figs. 28 and 29 This

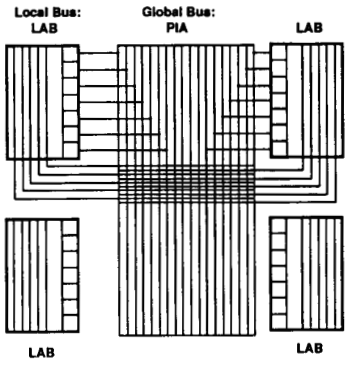


Fig. 28. Altera MAX 5000 global routing architecture.

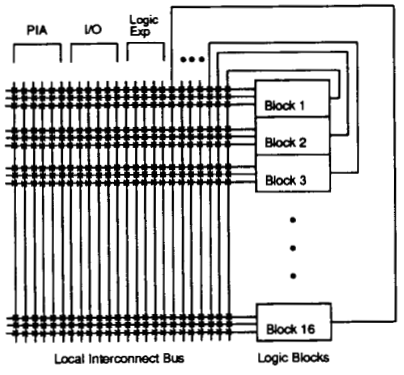


Fig. 29. Altera MAX 5000 local routing architecture.

architecture is novel in that it has a two-level hierarchy. This is an important feature because most designs exhibit some form of locality which a hierarchical organization may be able to utilize to obtain better density and performance. At the first level of the hierarchy, 16 or 32 of the logic blocks are grouped into a Logic Array Block, or LAB. The inner details of the LAB are illustrated in Figure 29, and it can be seen that the structure of the LAB is very similar to a traditional PLD [28]. Each 'x' in the figure indicates a point where a connection can be made. As described in Section 2.3, the connection is formed using EPROM-like floating-gate transistors. Here the channel is a set of dedicated vertical tracks that run the entire length of the LAB. The tracks are dedicated to one of four types of connections:

1) Connections from the outputs of all logic blocks in this LAB.
2) Connections from the logic expanders, described in Section III–C4).
3) Connections from outputs of logic blocks in other LAB's. These connections come from the global interconnect structure at the next level of hierarchy, called the PIA, and are described below.
4) Connections from the I/O pads of the chip.

All four types of tracks pass by every logic block in the LAB. In the connection block every such track can connect into every logic block pin. This makes the routing very simple, since any input can directly connect to any track.

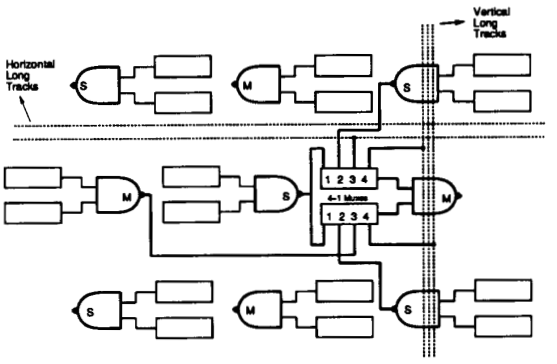


Fig. 30. Plessey routing architecture.

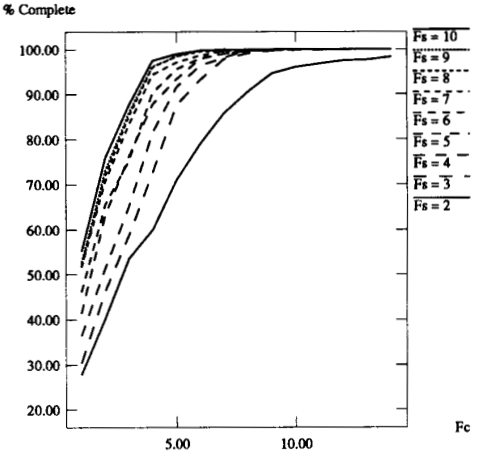


Fig. 31. Percent routing completion versus F_c, one circuit.

Using fewer connection points results in better density and performance, but yields a more complex routing problem. The intra-LAB routing structure could be considered a segmented channel, where the segments are all as long as possible. Recall that connections also perform wire-ANDing, and so the transistors in effect have two purposes.

Referring again to Fig. 28, connections are made among different LABs using a global interconnect structure called the Programmable Interconnect Array (PIA). It connects outputs from each LAB to inputs of other LAB's, and acts as one large switch block. Its internal structure is similar to that of the internal routing scheme within the LAB—a large number of vertical tracks (180 in the EPM 5128 device) are connected to the logic block outputs. There is full connectivity among the logic block outputs and LAB inputs within the PIA. Again, the advantage of this scheme is that it makes the routing problem very easy, and the regularity of the physical design of the silicon allows it to be packed tightly and efficiently. The disadvantage is that many switches are needed, and these may add more capacitive load than necessary.

A second advantage of this approach is that the delay through the PIA is the same regardless of which track is used since all tracks have identical loading. This is very helpful when attempting to predict system performance.

The price, however, may be circuits that are much slower than is possible with segmented tracks.

The routing architecture of AMD [3] and Plus Logic [34] are similar to that of Altera.

4) The Plessey Routing Architecture: The routing architecture of the Plessey FPGA is illustrated in Fig. 30. There are two alternating types of logic block. The Master block is indicated by an "M," and the Slave block is indicated by an "S." Also, the rows of logic blocks alternate in their direction of flow. Programmable routing is achieved using only a multiplexer as a connection block on the inputs of the two-input NAND gate as described in Section III–B2). The multiplexers are controlled by SRAM cells. Each input of the NAND gate comes from a four-to-one multiplexer. In Fig. 30, the connections for one logic block are shown, and indicate the basic pattern of connectivity. The inputs to each multiplexer are connected to:

1) The output of the previous NAND gate in the row.
2) The output of the NAND gate above or below this logic block, whichever is closer.
3) A vertical long track.
4) One of the following three connections depending on which NAND input the multiplexer drives:
   - A horizontal long track (the upper input).
   - The NAND gate output two blocks previous to the current one (lower input of Master block).
   - The output of the block diagonally away from the current one (lower input of Slave block).

   Although not shown, the other logic blocks are connected similarly.

There are three vertical long tracks per column of logic blocks, and two horizontal long tracks per row. There are two types of long tracks in each direction: short range tracks which travel 10 blocks, and long range tracks which travel the width or length of the entire chip.

While this type of nearest-neighbour routing architecture works well for circuits with primarily local connections, there is an insufficient number of longer horizontal and vertical routing tracks for typical circuits with more non-local connections. For this reason the logic blocks themselves must be used for routing which can be costly in area and performance.

The routing architectures of Algotronix [2], Concurrent Logic [13], and Toshiba [32] exhibit a similar mix of many local and few long range connections.

B. Routing Architecture Tradeoffs

In this section we summarize several recent results concerning FPGA routing architecture. The work in [37], [38], [7] [8] considers the relationship between the flexibility of FPGA routing architecture and both the routability and area-efficiency. A basic tradeoff is as follows: using a large number of programmable switches makes it easier to achieve routing completion, but these switches consume area, and therefore it is desirable to minimize the number used. The work in [38] defines the flexibility of the connection block, F_c, to be the number of tracks in the adjacent channel to which each logic block pin can connect. The flexibility of the switch block, F_s, as depicted in Fig. 24, is defined as the number of tracks to which each track entering the switch block can connect. Both of these definitions assume that all wires or pins has the same degree of connectivity.

The research in [38], [7] uses an experimental approach similar to the logic block experiments described in Section 3.5. Several benchmark designs are "implemented" using routing architectures with different flexibilities. For each benchmark the following was determined: 1) the percentage of connections successfully routed with a fixed number of tracks per channel and 2) the number of tracks per channel required to achieve 100% routing completion, assuming that every channel has the same number of tracks.

Figure 31 plots results of the first type for one benchmark design. The Y-axis is the percentage of connections that were successfully completed with a connection block of flexibility F_c as indicated on the X-axis, and a switch block of flexibility F_s varying from 2 to 10 across the nine curves. These data lead to two conclusions that are consistent across all benchmark designs considered. These are:

1) The connection block requires flexibility F_c greater than half the tracks for 100% routing completion to be possible.
2) The switch block requires little flexibility to achieve 100% routing completion. This is because a flexibility of $F_s = 3$ easily achieves completion and the maximum value of F_s is 36 if there are 12 tracks in a channel. These results make intuitive sense, as explained in [38].

As flexibility increases, the number of tracks needed to obtain 100% routing completion decreases. Table 2 gives the average, over five circuits, of the number of tracks in excess of channel density needed to successfully route architectures with different values of F_s and $\frac{F_c}{W}$. The table illustrates the diminishing returns, in terms of track count, of increasing the flexibility of the routing architecture.

Table 3 gives the average number of switches per logic block for the data in Table 2. This data shows that there is a point where the higher flexibility of the switch and connection blocks would cost more than the reduction in tracks, in terms of total number of programmable switches. By inspection of the table, the best architecture, in terms of total number of programmable switches, appears to be with a value of F_s between three and four, and the fraction $\frac{F_c}{W}$ (where W is the number of tracks per channel) between 0.7 and 0.8 [38], [8].

Asymptotic Results on Channel Segmentation In [16], El Gamal *et al.* give asymptotic results on the number of excess tracks (above channel density) needed to successfully route a segmented routing channel of the type used in the Actel FPGA. Another paper in this special section contains more details on this work.

In this paper various architectural and programming technology features of commercial FPGA's have been

Table 2 Average Number of Switches per Logic Block Tile for Each Routing Architecture

	Fc/W							
Fs	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
2	nr	nr	nr	nr	nr	11.2	10.8	9.0
3	nr	nr	nr	4.6	2.4	1.2	0.8	0.8
4	nr	nr	6.2	3.0	1.6	0.6	0.6	0.6
5	nr	nr	4.2	2.4	1.0	0.4	0.2	0.2
6	nr	7.6	3.8	1.8	0.8	0.4	0.4	0.4
7	nr	5.2	3.4	1.4	0.2	0.2	0.2	0.2
8	nr	4.4	1.4	0.6	0.2	0.0	0.4	0.2
9	10.0	4.2	1.0	0.4	0.2	0.2	0.0	0.0
10	8.0	3.2	1.4	0.6	0.2	0.0	0.0	0.0

Table 3 Average Excess Tracks for Each Routing Architecture

	Fc/W							
Fs	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
2	nr	nr	nr	nr	nr	349	381	306
3	nr	nr	nr	259	221*	223	241	260
4	nr	nr	270	229	231	249	267	286
5	nr	nr	306	257	257	254	271	288
6	nr	369	304	285	305	278	319	338
7	nr	372	357	313	285	302	319	336
8	nr	410	345	317	309	326	343	360
9	510	401	325	343	333	350	367	384
10	459	409	351	369	357	374	391	408

surveyed. The logic block architectures have been classified by granularity and the routing architectures have been described in terms of connectivity, symmetry, and hierarchy. The effects of logic block choice and routing architecture on FPGA density and performance.

V. Summary and Future Directions

In the future, we expect to see an even greater proliferation of architectural innovations as well as research that answers basic questions about FPGA architecture. We expect these to include:

- Development of "special-purpose" FPGA's tuned toward specific applications such as datapath circuits, DSP applications, finite state machines, and FPGA-based compute engines [5].
- The development of multi-chip field-programmable systems for emulation, acceleration and rapid prototyping (e.g. [5]).
- Development of nonhomogeneous logic block architectures. These hold the promise of providing better area/performance tradeoffs, because larger granularity blocks achieve better performance while smaller granularity blocks achieve higher density.
- Segmented routing architectures provide important performance and density advantages over nonsegmented architectures (those with only one or two lengths of segment). New insight is needed in determining segment length distribution.
- It is possible that the success of the FPGA in the digital world may be reproduced in the analog domain, with a field-programmable analog array (FPAA). Such work has already begun with the chip presented in [29] [30].

In addition, computer-aided design tools for FPGA's will become more sophisticated, enabling the development of architectural features aimed at improving performance and density.

Acknowledgment

The second author would like to thank Dana How and Jack Kouloheris for their useful comments on earlier drafts of this paper.

References

[1] M. Ahrens, *et. al.*, "An FPGA family optimized for high densities and reduced routing delay," in *Proc. 1990 CICC*, M pp. 31.5.1–31.5.4, May 1990.

[2] CAL 1024 Datasheet, Algotronix Ltd. Edinburgh, Scotland, 1989.

[3] *Mach Devices High Density EE Programmable Logic Data Book.* AMD, 1990.

[4] S. Baker, "Lattice fields FPGA," *Elect. Eng. Times,* no. 645, page 1, June 10, 1991.

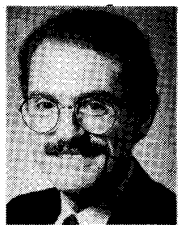
[5] P. Bertin, D. Roncin, and J. Vuillemin, "Programmable active memories: Performance measurements," in *ACM First Inter-*

national Workshop on Field-Programmable Gate Arrays, pp. 57–59, Feb. 1992.
[6] J. Birkner, A. Chan, H. T. Chua, A. Chao, K. Gordon, B. Kleinman, P. Kolze, and R. Wong, "A very high-speed field programmable gate array using metal-to-metal anti-fuse programmable elements," in New Hardware Product Introduction at CICC '91.
[7] S. Brown, "Routing architectures and algorithms for field-programmable gate arrays," Ph.D. Thesis, Dept. of Electrical Engineering University of Toronto, February 1992.
[8] S. Brown, R. Francis, J. Rose, and Z. Vranesic, *Field-Programmable Gate Arrays,* Kluwer Academic Publishers, 1992.
[9] W. Carter *et. al.*, "A user programmable reconfigurable gate array," *Proc. 1986 CICC,* pp. 233–235, May 1986.
[10] D. Chen, J. Rabaey, "A Reconfigurable Multiprocessor IC for Rapid Prototyping of Real-Time Data Paths," in *Proc. ISSCC 92,,* pp. 74–57, Feb. 1992.
[11] K. Chung, S. Singh, J. Rose, P. Chow, "Using hierarchical logic blocks to improve the speed of field-programmable gate arrays," in *FPGAs,* W. Moore and W. Luk Eds., Abingdon 1991, edited from the Oxford 1991 International Workshop on Field Programmable Logic and Applications.
[12] K. Chung and J. Rose, "TEMPT: Technology mapping for exploration of FPGA architectures with hard-wired connections," in *Proc. 29th Design Automation Conf.,* June 1992, Anaheim, CA, pp. 361–367.
[13] *Concurrent Logic CFA6006 Field-Programmable Gate Array Data Sheet,* Sunnyvale, CA, 1991.
[14] A. El Gamal, "Two-dimensional stochastic model for interconnections in master slice integrated circuits," *IEEE Trans. Circuits Systems,* vol. CAS-28, no. 2, pp. 127–138, February 1981.
[15] A. El Gamal, *et al.*, "An architecture for electrically configurable gate arrays," *IEEE JSSC,* vol. 124, No. 2, pp. 394–398, Apr. 1989.
[16] A. El Gamal, J. Greene, and V. Roychowdhury, "Segmented channel routing is nearly as efficient as channel routing (and just as hard)," in *Advanced Research in VLSI,* University California, Santa Cruz, March 25–27, 1991.
[17] J. Greene, et al, "Segmented channel routing," in *Proc. 27th Design Automation Conf.,* pp. 567–572, June 1990.
[18] J. Greene, "Antifuse field programmable gate array," *Proc. IEEE,* vol. 81, no. 7, July 1993..
[19] 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," *Int. Electron Devices Meeting Tech. Digest,* pp. 786–789, 1988.
[20] D. Hill and N.-S. Woo, "The benefits of flexibility in look-up table FPGAs," in *FPGAs,* W. Moore and W. Luk, Eds., Abingdon 1991, edited from the *Oxford 1991 Int. Workshop on Field Programmable Logic and Applications,* pp. 127–136.
[21] H. Hsieh, *et al.*, "A second generation user programmable gate array," in *Proc. 1987 CICC,* pp. 515–521, May 1987.
[22] H. Hsieh, et al, "A 9000-gate user-programmable gate array," in *Proc. 1988 CICC,* pp. 15.3.1–15.3.7, May 1988.
[23] H. Hsieh, *et al.*, "Third-generation architecture boosts speed and density of field-programmable gate arrays," in *Proc. 1990 CICC,* pp. 31.2.1–31.2.7, May 1990.
[24] J. Kouloheris and A. El Gamal, "FPGA performance vs. cell granularity," in *Custom Integrated Circuits Conf. 91,* CICC, pp. 61.2.1–6.2.4, May 191.
[25] J. Kouloheris and A. El Gamal, "FPGA area versus cell granularity - lookup tables and PLA cells," in *FPGA 92, ACM First Int. Workshop on Field-Programmable Gate Arrays,* pp. 9–14, Feb. 1992.
[26] J. Kouloheris and A. El Gamal, "FPGa area versus cell granularity - PLA cells," in *Custom Integrated Circuits Conference 92,* CICC, May 1992.
[27] J. Kouloheris, private communication.
[28] P. K. Lala, *Digital System Design Using Programmable Logic Devices.* Prentice Hall, 1990.
[29] E. Lee, "Field-programmable analog arrays - a cmos realization," M.A.Sc. thesis, Univ. of Toronto, Toronto, Ontario, Canada.
[30] E. Lee and P. G. Gulak, "A CMOS field-programmable analog array," *IEEE JSSC,* vol. 26, no. 12, pp. 1860–1867, Dec. 1991.
[31] D. Marple and L. Cooke, "An MPGA compatible FPGA architecture," in *FPGA 92, ACM First Int. Workshop on Field-Programmable Gate Arrays,* pp. 39–44, Feb. 1992.
[32] H. Muroga, H. Murata, Y. Saeki, T. Hibi, Y. Ohashi, T. Noguchi, and T. Nishimura, "A large scale FPGA with 10K core cells with CMOS 0.8 μm 3-layered metal process," *Custom Integrated Circuits Conf. '91,* CICC, pp. 6.4.1–6.4.4, May 1991.
[33] Plessey Semiconductor ERA60100 preliminary data sheet, Swindon, England, 1989.
[34] Plus Logic FPGA2040 Field-Programmable Gate Array Data Sheet, Plus Logic, San Jose, CA, 1989.
[35] J. S. Rose, R. J. Francis, P. Chow and D. Lewis, "The effect of logic block complexity on area of programmable gate arrays," in *Proc. 1989 Custom Integrated Circuits Conf.,* pp. 5.3.1–5.3.5, May 1989.
[36] J. S. Rose, R. J. Francis, D. Lewis, and P. Chow, "Architecture of field-programmable gate arrays: The effect of logic block functionality on area efficiency," *IEEE JSSC,* Vol. 25, No. 5, pp. 1217–1225, Oct. 1990.
[37] J. S. Rose and S. Brown, "The effect of switch box flexibility on routability of field programmable gate arrays," in *Proc. 1990 Custom Integrated Circuits Conf.,* pp. 27.5.1–27.5.4, May 1990.
[38] J. S. Rose and S. Brown, "Flexibility of Interconnection Structures for Field-Programmable Gate Arrays," *IEEE JSSC,* Vol. 26, No. 3, pp. 277–282, March 1991.
[39] S. Singh, J. S. Rose, D. Lewis, K. Chung, and P. Chow, "Optimization of Field-Programmable Gate Array Logic Block Architecture for Speed," in *Custom Integrated Circuits Conference 91,* CICC, pp. 6.1.1–6.1.6, May 1991.
[40] S. Singh, "The effect of logic block architecture on the speed of field-programmable gate arrays," M.A.Sc. Thesis, Department of Electrical Engineering, University of Toronto, August 1991.
[41] S. Singh, J. Rose, D. Lewis, and P. Chow, "The effect of logic block architecture on FPGA performance," *IEEE JSSC,* vol. 27, no. 3, pp. 281–287, Mar. 1992.
[42] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.,* vol. C-20, pp. 153–161, 1971.
[43] S. C. Wong, H. C. So, J. H. Ou, and J. Costello, "A 5000-gate CMOS EPLD with multiple logic and interconnect arrays," in *Proc. 1989 CICC,* pp. 5.8.1–5.8.4, May 1989.
[44] S. Vij, B. Ahanim, "A high density, high speed, array-based erasable programmable logic device with programmable speed/power optimization," in *ACM First Int. Workshop on Field-Programmable Gate Arrays,* pp. 29–32, Feb. 1992.
[45] Jean-Michel Vuillamy, "Performance enhancement in field-programmable gate arrays," M.A.Sc. Thesis, Department of Electrical Engineering, University of Toronto, April 1991.
[46] S. Yau and C. Tang, "Universal logic modules and their application," *IEEE Trans. Comput.,* Vol. C-19, pp. 141–149.

Jonathan Rose (Member, IEEE) received the B.A.Sc. degree in engineering science in 1980, and the M.A.Sc. and the Ph.D. degree in electrical engineering, in 1982 and 1986, respectively, from the University of Toronto.

During the summer of 1983, he was with Bell-Northern Research Ltd., Ottawa, in the Integrated Cicuits CAD/CAM group. From 1986 to 1989, he was a Research Associate in the Computer Systems Laboratory at Stanford University. In 1989, he joined the faculty of the University of Toronto, where he is currently an Assistant Professor of electrical engineering. He has served as General and Program Cochair for the 1992 ACM First International Workshop on Field Programmable Gate Arrays, and on the program committees of the Oxford Field-Programmable Logic workshop in 1991 and the Vienna Field-Programmable Logic Workshop in 1992, as well as the program committees for ICCD '92 and ICCAD '92. In 1990, a paper coauthored with Stephen Brown on detailed routing for FPGA's received a distinguished paper citation at the ICCAD conference. His research interests include CAD and architecture for Field-Programmable Gate Arrays, automatic layout, and parallel CAD algorithms.

Abbas El Gamal (Senior Member, IEEE), for photograph and biography please see the Prolog to the Special Section in this issue of the PROCEEDINGS.



Alberto Sangiovanni-Vincentilli (Fellow, IEEE) received the Dr. of Eng. degree from Politechnico di Milano, Italy, in 1971.

He is currently a Professor of electrical engineering and computer science at the University of California at Berkeley, where he has been on the Faculty since 1976. He was a Visiting Scientist at the IBM T.J. Watson Research Center in 1980, Consultant in residence at Harris in 1981, and Visiting Professor at MIT in 1987. He has held a number of visiting professor positions at the University of Torino, University of Bologna, University of Pavia, University of Pisa, and University of Rome. He is a member of the Berkeley Roundtable for International Economy (BRIE). He is a Corporate Fellow at Harris and Thinking Machines, has been a Director of ViewLogic and Pie Design System, on the Strategic Advisory Council of Cadence, and on the Technical Advisory Board of Cadence Analog Division and Synopses, companies that he helped funding. he consulted for several major U.S. (including ATT, IBM, DEC, GTE, Intel, NYNEX, General Electric, Texas Instruments), European (Alcatel, Bull, Olivetti, SGS-Thompson, Fiat), and Japanese companies (Kawasaki Steel), in addition to start-ups (Actel, CrossCheck, Biocad, Redwood Design Automation, SiArc, Wireless Access). He is the Technology Adviser to Greylock Management. He is on the International Advisory Board of the Institute for Micro-Electronics of Singapore.

In 1981 he received the Distinguished Teaching Award of the University of California. He has received three Best Paper Awards (1982, 1983, and 1990) and a Best Presentation Award (1982) at the Design Automation Conference and is the corecipient of the Guillemin-Cauer Award (1982–1983), the Darlington Award (1987–1988), and the Best Paper Award of the Cicuits and Systems Society of the IEEE (1989–1990). He has published over 250 papers and three books in the are of CAD for VLSI. He has been a Technical Program Chair of the International Conference on CAD for 1989 and the General Chair for 1990.